CHALMERS

The Elder Crown

A Multi-Agent based Artificial Intelligence

Simon Eliasson

Denise Glansholm

Niklas Logren

Jonathan Orrö

Teodor Ostwald

Simon Österberg

Department of Computer Science & Information Technology
Chalmers University of Technology
Gothenburg, Sweden 2014
Bachelor Thesis 2014

Abstract

The objective of this thesis is to describe the development of The Elder Crown, a game aimed at exploring the concept of artificial intelligence and Multi-Agent Systems, while using the Belief-Desire-Intention model. The game depicts a world containing one or more inhabited villages which the player, as a god, reigns over. Unlike most games today, where the entertainment has its core in the player interaction, The Elder Crown is intended to have its enjoyment based on simulating a complex and unpredictable environment, with a small amount of interactive options. This means that although the player has the role of a god, she will be far from omnipotent, and cannot be sure what consequences her actions will have.

The main goal of the Elder Crown have been to create a simulation, based on a well designed game engine, that would make intricate story generation possible. Although this goal was not fully met, the resulting product contains relevant features, as well as future possibilities for expansion. These expansion possibilities will also be discussed.

Sammanfattning

Syftet med denna rapport är att beskriva utvecklingen av The Elder Crown, ett spel vars mål är att utforska koncepten artificiell intelligens och agentbaserade system, samt hur Belief-Desire-Intention-modellen har använts för detta ändamål. Spelet skildrar en värld som innehåller en eller flera bebodda byar, som spelaren är gud över. Till skillnad från de flesta spel idag, vars nöje har sin kärna i användarens interaktion, är The Elder Crown avsedd att ha sitt underhållningsvärde baserat på att simulera en komplex och oförutsägbar värld, med en begränsad mängd interaktiva möjligheter. Det innebär att även om spelaren har rollen som en gud, kommer man att vara långt från allsmäktig, och kan inte vara säker på vilka konsekvenser ens handlingar kommer att få.

Målet med The Elder Crown var i huvudsak att skapa en simulation, med hjälp av en genomtänkt spelmotor, som skulle ha möjlighet att skapa intressanta historier på egen hand. Trots att detta målet inte har uppnåtts, så innehar produkten ett flertal funktioner som också gör framtida expansioner av mjukvaran möjlig. Dessa expansioner kommer även de att behandlas.

Ordlista

Agent - Ett objekt som kommer att implementeras med AI.
AI - Artificiell intelligens.
Application programming interface (API) - En samling funktionsanrop som kan tillåta en programvara att kommunicera med en annan.
Belief-Desire-Intention (BDI) - En arkitektur som vår IPA inspirerats av.
Entity - Ett objekt i världen.
Intent-Plan-Action (IPA) - Vår egen arkitektur som vår AI följer.
Minimum Viable Product (MVP) - En produkt som enbart har sina kärnegenskaper för lansering implementerade.
Model-View-Controller (MVC) - Standardarkiterktur för att strukturera upp mjukvara.
Multi-Agent System (MAS) - Ett datorsystem som simulerar flera Agents och deras interaktioner med varandra.
Pathfinding - En algoritm som mjukvaran använder för att hitta den kortaste vägen mellan två punkter på en given karta.
Perception - Ett begrepp på den information som en bybo blir kontinuerligt uppdaterad med.
Project Backlog - En lista över funktionaliteter som projektet ämnar realisera.
Village Elder - Agent i byn som agerar överhuvud och har utmärkande egenskaper.
Villager - En agent som bor i en by. Även kallad bybo.

Innehåll

1

Introduktion

Simulatorer har alltid varit en framträdande del inom spelindustrin. Känslan av att kontrollera ett intressant och ofta komplext system är unik och många gånger givande. Emellertid har de simulatorer som utvecklats för spel hittills inte främst sin tyngdpunkt i de faktiska elementen av simulationen. De simulerade enheterna agerar istället ofta enkla verktyg för spelaren, vilket framställer spelmiljön på ett stelt och personlighetslöst sätt. Detta koncept leder vidare till frågan: Är det möjligt att skifta underhållningsvärdet från kontroll över omgivningen till att huvudsakligen bevittna och uppleva simulationen i sig? Finns det ett sätt att bygga en simulation som skapar intressanta historier och händelser av sig själv?

För vissa kan denna idé te sig poänglös och kanske även tråkig. Varför skapa ett spel utan interaktion? Denna fråga är emellertid grunden som projektet i sin helhet vilar på: att begränsa användarens kontroll för att förbättra spelupplevelsen. Att omvandla kontroll till kaos, förutsägelse till osäkerhet och motiv till förhindrande. Skapa ett spel där spelaren utmanas att genomföra sina viljor.

1.1 Beskrivning

Idén bakom The Elder Crown är i grunden att utforska en hittills relativt orörd nisch inom spelutveckling: att skapa att spel som i grunden spelas autonomt, men ändå ger ett underhållningsvärde till spelaren. Detta genom att utöka en simulator med en spelupplevelse men samtidigt begränsa spelarens kontroll. Tanken är att slutprodukten skall vara en kombination mellan ett klassiskt spel och en modern, “Multi-Agent”-baserad simulator.

Spelets handling skall bestå i de interaktioner som simulationselementen har med varandra, till viss del påverkade av spelarens interaktion med världen. Dock kommer de val

spelaren gör aldrig att leda till omedelbara förändringar och det kommer därför inte finnas några garantier för hur dessa interaktioner kommer att påverka spelet i slutändan. Det är viktigt att notera att avsikten är att skapa ett spel där det finns mycket för användaren att göra, som inte nödvändigtvis består av direkt interaktion, men att det är upp till denne själv att skapa ett eget syfte med spelandet.

Är det möjligt att på detta sätt göra en simulator intressant för gemene man? Kan det rent av leda till en kommersiellt gångbar produkt?

1.2 Syfte

Syftet med detta projekt är att utöka spelutvecklingen i en ny riktning samt undersöka hur och till vilken nivå ett spel kan kombineras med en avancerad simulator utan att det är varken endast ett spel eller endast en simulator. Utöver detta ska projektet även utforska underhållningsvärdet i en produkt som denna.

Vidare finns det även en förhoppning om att den simulator som ligger till grund för spelet även skall kunna användas mer akademiskt. Eftersom denna del av projektet skall kunna simulera samspel mellan olika enheter i ett samhälle, allt ifrån behovet att äta till att organisera sig i grupper, så ska den även kunna användas till att undersöka olika sociala fenomen.

Exempel på experiment att genomföra studier på är samhällsutveckling då polygami är normen jämfört med monogami, sjukdomsspridning och balansering av populationen efter mängden tillgänglig föda. Det är viktigt att förstå att syftet med denna simulator inte är att kunna påvisa några nya fenomen, utan experimenten kommer först och främst att visa på simulatorns korrekthet samt eventuellt ge ytterligare stöd för vissa sociala teorier.

1.3 Inspiration och influenser

Precis som i musik och film så influeras spel i hög grad av vad som redan skapats inom branschen. Lösningar på problem hämtas ofta fritt från andra spel, ibland nästan kusligt uppenbart. The Elder Crown är förstås inget undantag. Gudsspel och simulatorer är väletablerade genrer med många stora titlar på marknaden. Denna sektion kommer förklara innebörden av vad ett gudsspel faktiskt är, samt redogöra för några av de spel som influerat främst grundidén bakom The Elder Crown.

1.3.1 Genren Gudsspel

Det finns ingen fastställd definition för vad ett gudsspel faktiskt är. De grundkriterier som ett gudsspel behöver uppfylla är att spelaren vakar över en spelvärld och har gudalika krafter. Vad som räknas som gudalika krafter är dock svårare att definiera och det är

därför ibland svårt att fastställa huruvida ett spel tillhör genren eller ej. Det första gudsspelet, enligt många spelfantaster, var Populous som släpptes 1989 [1]. Det var det första spel där tanken var att man antog sig rollen som en gud. Idén var att spelaren som gud skulle hjälpa sitt folk att besegra ett annat folk som tillbad en annan gud. Peter Molyneux som designade Populous kan därmed sägas vara gudsspelens fader och han är fortfarande verksam inom genren med spelet Godus. [2]

1.3.2 Black & White

Black & White, även det designat av Peter Molyneux, släpptes 2001 och hyllades av kritiker [3]. Liksom Populous så antog sig spelaren rollen som gud och kämpade mot andra gudar genom att använda sina gudalika krafter och sitt folk. Det var unikt och nyskapande på flera punkter men hade också ett problem som till stor del ligger till grund för idén bakom The Elder Crown, nämligen att folket man styrde över inte var något mer än verktyg för spelaren. Människorna i Black & White gjorde vad spelaren beordrade, byggde hus där spelaren placerade dem och hade varken unika personligheter eller varierande beteendemönster. Spelaren uppenbarade sig som en hand som kunde användas för att direkt interagera med världen. I figur 1.1b ser vi hur spelaren är på väg att plocka upp en människa ur en folkgrupp. Dessutom så var spelkartorna designade med specifika mål i åtanke, vilket begränsade friheten för spelaren och därmed gudsaspekten. Att bibehålla detta gudsperspektiv, att göra de simulerade individerna mer intressanta samt att skapa en öppnare och friare värld har varit viktiga idéer som präglat konceptet bakom The Elder Crown.

1.3.3 The Sims

The Sims släpptes 2000 och serien har varit under konstant utveckling sedan dess. The Sims-spelen är inga gudsspel, men har ändå mycket av det som Black & White saknade när det gäller att göra de simulerade individerna mer intressanta för spelaren att iaktta och interagera med. Individerna i The Sims har olika personligheter, behov och önskningar som spelaren måste handskas med, vilka visas i figur 1.1c. Det finns heller inget sätt att vinna spelet, utan spelaren definierar ständigt sina egna mål. The Sims utspelar sig dock i en begränsad värld och spelaren har inga egentliga krafter, förutom att kunna styra sina individers handlingar och bygga hus. [4] Systemet för individernas personligheter och önskningar i The Sims gör det dock enkelt och roligt för spelaren att förstå vad som händer i simulationen. Något som skulle vara värdefullt för The Elder Crown.

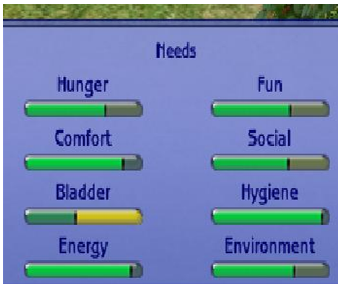
1.3.4 Minecraft

Minecraft utvecklades av Markus Persson och släpptes 2011 [5]. Minecraft är varken ett gudsspel eller en simulation och saknar egentligen likheter med någon av dessa genrer. Vad som gör Minecraft intressant är att det inte finns något väldefinierat mål utan det är något som spelaren måste skapa själv. Sandlådeläget, ett spelläge där spelaren kunde

spela ett spel utan de begränsningar som annars utgjorde utmaningen, hade länge funnits i andra spel, men få spel var en sandlåda i första hand. Den enorma (i praktiken oändliga) slumpgenererade världen och de många möjligheter som spelaren fick att interagera med den var intressant nog. Slumpgenererade världar, en helt fri spelstil och intressanta möjligheter att interagera med världen är alla aspekter som även kommer att finnas i The Elder Crown. Figur 1.1a visar ett exempel på en helt slumpgenererad värld i minecraft.

(a) Slumpmässigt genererad värld i Minecraft

(b) Bybor i spelet Black & White



(c) Statusindikatorer i The Sims

Figur 1.1: Bilder från spel som varit inspirationskällor

1.4 Avgränsningar

För att göra projektet genomförbart med den nivå av självständighet på spelet som beskrivits innan var naturligtvis begränsningar tvungna att göras.

1.4.1 Plattform

Spelet ämnas inte lanseras eller portas till andra plattformar än PC. Detta för att det inte är relevant för projektets mål att lägga tid på denna typ av utbyggnad.

1.4.2 Grafik och ljud

Spelgrafiken i projektet kommer ej att prioriteras högt. En tvådimensionell värld med vy ovanifrån, som genomfördes med hjälp av spelbiblioteket Slick2D. Det var även möjligt att använda andras bilder eller ljud (med tillstånd) för att ytterligare underlätta denna del av spelet. Dessa begränsningar kan få konsekvenser i att spelet inte blir så estetiskt tilltalande för en utomstående användare, men gjordes trots detta för att kunna fokusera på delar som är mer centrala för projektet.

1.4.3 Användbarhetstest

Inga användbarhetstest kommer att utföras. Användarupplevelsen vilar enbart på gruppens bedömningar. Detta anses vara rimligt då användarvänlighet ej är så viktigt för

projektets syfte.

2

Teoretisk bakgrund

Detta kapitel syftar till att redogöra för de områden som krävde akademiska förstudier för projektets genomförande. För att få en överblick över Artificiell Intelligens är det viktigt att förstå den teoretiska och historiska bakgrund som fältet vilar på idag. Detta kapitel går igenom definitionen av intelligens och går därefter vidare till de historiska tankar och filosofier som funnits kring AI samt exempel på strukturer för implementation som utnyttjats i spelet.

2.1 Definition av intelligens

Definitionen av intelligens varierar beroende på infallsvinkel. Tim Jones föreslår i sin bok om artificiell intelligens, *Artificial Intelligence—A Systems Approach*, två olika preciseringar; en som handlar om förmågan att planera, lösa problem och generellt sett resonera och en annan, enklare, definition som handlar om förmågan att fatta "rätt" beslut baserat på en mängd indata och ett antal olika möjliga handlingar. Författaren tar även upp problematiken med att definiera något som intelligent då det har en väldigt begränsad kompetens. Ett exempel på detta kan vara att ett program som kan vinna över världsmästaren i schack, men som trots detta inte kan genomföra enkla uppgifter som de flesta människor klarar. Utifrån detta kan man då förstå att intelligens kan definieras olika beroende på vilket perspektiv man väljer att bedöma efter.[6]

2.2 Artificiell Intelligens

Problematiken kring att på ett generellt sätt definiera intelligens (som togs upp i avsnitt 2.1) propagerar naturligtvis till definitionen av artificiell intelligens. Frågan om huruvida ett sant AI, det vill säga en maskin på samma kognitiva nivå som exempelvis en människa, någonsin kommer kunna utvecklas är något som varit oerhört omdebatterat.

Det finns ett flertal teorier och filosofier kring denna frågeställning, varav två av de mer kända är det så kallade Turingtestet och Chinese Room.

2.2.1 Turingtestet

Turingtestet är ett välkänt test som går ut på att bestämma när en maskin kan anses vara tänkande. I boken *Parsing the Turing Test* kan man läsa att detta test formulerades av den engelske matematikern Alan M. Turing år 1950, då han publicerade sin berömda artikel *Computing Machinery and Intelligence*. Testet bygger på den enkla principen att då en människa inte längre kan skilja en maskin från en människa i en längre konversation, antingen skriftlig eller verbal, måste möjligheten att maskinen är tänkande erkännas.[7]

2.2.2 Chinese Room

För att ytterligare utmana tanken om någon form av tänkande eller medvetande hos maskiner finns ett tankeexperiment kallat ”Chinese Room”. Detta myntades då den amerikanske filosofen John R. Searle år 1980 publicerade artikeln *Minds, brains, and programs* och är en filosofisk tanke som ifrågasätter huruvida en dator eller ett program faktiskt kan ha en uppnå samma kognitiva nivå som människor genom att ha ett medvetande. Searle skiljer i artikeln mellan ett så kallat ”starkt AI” och ett ”svagt AI”. Ett svagt AI definieras då som ett verktyg för att simulera ett medvetande, till skillnad från ett starkt AI som inte bara är ett verktyg, utan faktiskt *förstår* eventuella problem det ställs inför och kan användas för att förklara en människas förståelse för exempelvis en text.[8] Tankeexperimetet kan i korthet förklaras enligt nedan:

> *En man med engelska som modersmål stängs in i ett rum. I rummet har han tillgång till en stor hög med kinesisk skrift, trots att mannen inte kan kinesiska. Han har även fått en mängd instruktioner, på engelska, för hur han ska returnera en mängd särskild kinesisk skrift (svar), given en annan mängd särskild kinesisk skrift (frågor). Detta görs enbart genom att känna igen formella kinesiska symboler. I teorin skulle då mannen kunna ge svar på frågor, där både frågor och svar är på kinesiska, utan att faktiskt förstå ett enda ord.*

I denna tanke menar Searle att mannen uppträder som en dator. Detta använder filosofen då för att förkasta möjligheten till ett så kallat starkt AI, eftersom att mannen i experimentet inte kan sägas förstå kinesiska, utan bara följer en mängd regler givna till honom.

Utifrån detta kan även paralleller till turingtestet, som nämndes i avsnitt 2.2.1, dras. Searle påpekar nämligen att även om en person med kinesiska som modersmål inte kan avgöra om svaren är givna av en annan person med samma modersmål, eller genererade av en maskin, kan maskinen ändå inte sägas uppfylla kraven för starkt AI.

2.3 AI i historien

Tankar kring Artificiell intelligens har funnits genom alla tider, men forskningsområdets stora utveckling började först på 1900-talet. Denna rubrik ämnar att gå igenom de huvudsakliga skedena inom AI:ns fortskridande från 1940-talet till idag.

2.3.1 Grundandet av AI

Alan Turing anses ofta vara grundaren av forskning inom Artificiell Intelligens [6], mycket på grund av sin formulering av Turingtestet, som diskuterats närmare i avsnitt 2.2.1. Redan innan detta test definierades släppte han dock en tes kallad ”Intelligent Machinery”(1948). I denna text slår Turing hål på flera av dåtidens argument angående varför intelligenta maskiner aldrig skulle kunna skapas. Han ger även exempel på hur enkla beräkningsmekanismer kan anpassa och lära sig [9].

Trots Turings viktiga roll inom AI så myntades inte uttrycket Artificiell Intelligens förrän år 1956 av John McCarthy. Han och bland andra Marvin Minsky, Allen Newell och Herbert Simon (varav alla har kommit att bli inflytelserika personer inom AI) organiserade detta år en studiegrupp då McCarthy föreslog namnet [10]. Under sommaren då denna anordnades byggde Newell och Simon sin så kallade Logic Theorist, som lyckades bevisa de flesta teorem i kapitel två i Alfred North Whitehead och Bertrand Russells bok “Principia Mathematica”. Ännu mer anmärkningsvärt är att deras program även hittade ett bevis på ett teorem som var kortare än det som presenterades i Principia Mathematica. Detta kom att bli det första datorskrivna programmet under kategorin Artificiell Intelligens [11].

2.3.2 Den tidiga utvecklingen

Under 1950-talet gjordes flera försök att konstruera ett program som kunde spela brädspelet Dam. Dessa blev emellertid aldrig riktigt framgångsrika, utan kunde på sin höjd spela som en nybörjare, trots att programmen krävde stor datorkraft och lång tid för att spela. År 1952 byggde dock den amerikanske pionjären Arthur Samuel ett program under helt andra förutsättningar än de tidigare. Han hade nämligen visionen om ett program som kunde lära sig och utvecklas efterhand då det kördes, vilket aldrig tidigare gjorts. [6]

Samuel lät två instanser av programmet spela mot varandra och därigenom förbättra sig. Som väntat var kvaliteten på spelet högst klandervärd i början, men blev gradvis bättre. Då programmen spelat mellan 15 och 21 omgångar Dam ansågs de vara ”knepiga, men helt klart möjliga att besegra” av duktiga amatörspelare. Mellan spel 22 och 28 hade svårigheten nu ökat till ”bättre än en genomsnittlig spelare” [12]. Programmet fortsatte kontinuerligt att lära sig och år 1962 vann det över den föredetta mästaren i Dam i Connecticut [6].

2.3.3 AI:ns vinter

Det var under mitten av 1950-talet som forskarvärlden började få upp ögonen för AI ordentligt och förhoppningarna om utvecklingen inom området var höga. Fler och fler system utvecklades under denna tid inom olika grenar av ämnet, så som problemlösning, naturliga språk, lärande och planering. Snart insågs dock att de stora förväntningar som många lade på Artificiell Intelligens blev svåra att införliva i praktiken. Många problem i verkliga livet är oerhört komplexa och svåra att simulera på ett tillfredställande sätt.

Ett exempel på hur forskare misslyckades att uppfylla sin egna visioner är inom applicering på naturliga språk. De förväntade sig kunna skapa program som översatte mellan engelska och ryska och försökte då översätta frasen ”The spirit is willing but the flesh is weak” till ryska och sedan tillbaka till engelska. Resultatet blev då meningen ”The vodka is good but the meat is rotten”. Det visade sig alltså då att de naturliga språkens komplexitet gör det svårt att skapa program som kan hantera språket på ett godtagbart sätt. [10]

Till följd av dessa svårigheter att simulera system med koppling till verkligheten så drogs på 1970-talet en stor del av det finansiella stöd som fanns till forskningen in. Detta ledde till det som senare kom att kallas “AI:ns vinter”. Bristen på resurser gjorde det svårt att bedriva större studier inom ämnet vilket naturligtvis gjorde att utvecklingen blev lidande. [6]

2.3.4 Återuppståndelsen

Det var först i slutet på 1980-talet som utvecklingen av Artificiell Intelligens återupptogs. En anledning till denna återkomst var att forskare vid detta laget hade anammat en mer realistisk syn på de möjliga användningsområdena inom fältet. Istället för att som tidigare ha visionen att exempelvis simulera hela skalan av människans kognitiva förmågor, delades områden nu upp i mindre delar och isolerades för att möjliggöra implementation [6]. En annan anledning till AI:ns återupptagande var att bredden på fältet ökade för att inkludera fler områden, som exempelvis genetiska algoritmer inom biologi [10].

Resultatet av den nya synen på Artificiell Intelligens blev en betydligt snabbare utveckling i var och en av de smalare grenarna i förhållande till tidigare i historien. Dessa framsteg ledde snart till framtagande av så kallade expertsystem. Detta är system som inkorporerar stora domäner av data och använder detta i kombination med mänsklig expertis inom ett specifikt ämne för att lösa problem och svara på frågor som annars skulle kräva en mänsklig expert. Ett exempel på detta som används i stor utsträckning är medicinsk diagnostisering. Expertsystemen kom att bidra till AI:ns utveckling som industri. [10]

2.3.5 Artificiell Intelligens idag

Ett av de största steg AI:n tagit i modern tid är utnyttjandet av internet som domän för data. Tidigare förlitade sig AI:n på algoritmer och djupare "förståelse" av problem, men i och med att internet snabbt växte och började innehålla stora mängder information, kunde AI:n snart övergå till att använda detta som databas i sitt problemlösande. Detta i kombination med att datorernas datorkraft snabbt ökade möjliggjorde stora uteckling-ar av fältet. [10]

Till följd av denna stora utveckling är Artificiell Intelligens idag ett både djupt och brett fält för forskning och utnyttjas även dagligen i det vardagliga livet. Några exempel på vanliga användningsområden av AI inkluderar exempelvis avgörande om ett e-postmeddelande är "skräppost", reseplanerare för trafik, utnyttjande av robotar inom många affärsområden (exempelvis transport, hälsovård och rymdutforskning), hitta ruttar i datornätverk et cetera. Alla dessa problem kan lösas av datorer, som använder sig av tekniker som faller under kategorin Artificiell Intelligens.

2.4 AI i spel

AI har alltid varit en central del i spel då det genererar en mycket mer dynamisk spelupplevelse vad gällande utmaningar och interaktion med spelvärlden jämfört med mänskligt genererad information. I de flesta genrer och spel finns det ett enspelarläge som kräver artificiellt motstånd av något slag.

Att utveckla datoriserade system som framgångsrikt besegrar människan i spel har länge varit ett aktivt forskningsområde och är fortfarande under ständig utveckling. Problemet med AI i spel är att antalet begränsningar och regler som styr spelet ofta är så stort att antalet möjliga utfall gör det omöjligt för en dator att kalkylera det optimala draget [13]. I vissa spel, såsom exempelvis schack, har emellertid AI utvecklats som kan besegra mänskliga mästare. Spel som implementerar en sådan AI är dock ofta av relativt simpel natur, med få och lättförståeliga regler. Spelplanen och spelpjäserna i schack bygger upp ett såpass enkelt spel att en dator utan större svårigheter kan räkna ut det till synes optimala draget och därmed vinna över den mänskliga hjärnan.

Idag så är dock spel ofta mycket mer komplicerade än så och ren logik spelar mindre roll. När antalet möjliga utfall blir så stort att en dator inte längre kan räkna ut vad som är bästa möjliga handlingen, så får programmeraren implementera ett system som försöker simulera hur en mänsklig spelare hade hanterat diverse situationer. Egentligen krävs inte mycket mer än Schack för att göra dessa problem uppenbara. Go är ett relativt enkelt spel med simpla regler och begränsad spelplan där AI fortfarande inte har en chans mot vältränade mänskliga hjärnor [14].

Att programmera ett så kallat script för AI:n var länge, och är fortfarande i stor ut-

sträckning, den dominanta lösningen på detta problem inom spel [15]. Ett script är en fördefinierad lista av kommandon [16] som AI:n utför, vilket innebär att den har ett beteende som aldrig förändras. Detta utesluter dock inte att den kan svara på vad den mänskliga spelaren gör, bara att den alltid, givet situationen, svarar på samma sätt. Detta är dock ett stort steg från AI:n i schack, som egentligen ignorerar motspelarens drag och endast analyserar spelvärlden för att dra slutsatser. Problemet med en scriptbaserad implementation är att en mänsklig spelare relativt lätt kan lära sig beteendemönstren hos AI:n och utnyttja dessa. Detta betyder att AI:n endast förblir utmanande tills man vunnit över den första gången.

Ett sätt att göra en scriptbaserad AI mer intressant är att implementera den som oberoende av förkunskaper om spelplanen. Första gången detta gjordes var i Age of Empires så kallade ”skirmish”-läge där spelplanen slumpgenererades varje spelrunda [17]. Slumpade kartor krävde en helt ny typ av scriptad AI som kunde analysera spelplanen och dra slutsatser utifrån den information den fick om omgivningen. Scriptet var fortfarande hugget i sten, men resulterade alltid i olika beteenden då spelplanen aldrig var densamma. Ett sådant script gör det förstås omöjligt för en mänsklig spelare att lära sig exakt hur man vinner över AI:n, men betyder fortfarande att AI:n är förutsägbar i ett större perspektiv. För att höja nivån ytterligare så finns också den AI som aktivt försöker lära sig hur den mänskliga spelaren agerar och sedan använda detta för egen vinning.

Att ge datorn möjligheten att lära sig genom införskaffandet av information är vitalt för att kunna utveckla en AI som inte längre är förutsägbar. AI:n måste kunna analysera sina handlingar gentemot spelaren och dra slutsatser som förändrar dess beteende. Mer specifikt så måste datorn både samla och använda ny information samtidigt som den försöker prioritera det den redan vet [15]. Programmerarens uppgift i detta fall är inte längre att designa ett beteende, utan snarare att implementera ett system som kan designa sig självt. Med en sådan modell så blir det näst intill omöjligt för den mänskliga spelaren att förutse vad AI:n kommer att göra härnäst. AI:n kommer göra det den tror är bäst och är dessutom villig att förkasta vad som inte fungerar. Det är svårt att definiera exakt vad som krävs av denna typ av intelligens, men det är det mest intressanta alternativet då det efterliknar spelaren bättre än ett script.

2.5 Belief-Desire-Intention

Belief-Desire-Intention (BDI) är en modell som tagits fram för att simulera de mänskliga kognitiva processer som utgör hjärnans beslutfattande. Arkitekturen har sitt ursprung i Michael Bratmans teoretiska arbete [18] samt det praktiska arbetet av Anand Rao och Michael Georgeff [19]. Modellen har bland annat tagits fram genom att analysera hur människor och djur tar beslut, vilket ger ett verklighetstroget intryck av hur agenter bestämmer vad de ska göra och planera hur de ska genomföra detta. BDI byggs upp genom att klassificera de tre stora komponenterna *Belief, Desire* och *Intention.*

Belief, som är den första av dessa tre beståndsdelar, utgör en agents informativa tillstånd. Detta representerar alltså agentens tro kring sitt omgivande universum. Detta speglar verkligheten, i och med att denna tro kan skilja sig från hur världen faktiskt ser ut. Desire står i sin tur för vad en agent skulle vilja åstadkomma, det vill säga dennes motivationstillstånd. Modellens tredje komponent, Intention (som inte bör beblandas med Desire), representerar vad en agent faktiskt väljer att göra. [20]

Med hjälp av dessa tre byggstenar konstrueras ett sätt att fatta beslut och planera. Genom att organisera och dela upp planer i mindre delmål genomförs sedan handlingar som syftar till att nå ett högre mål. [21]

En begränsning hos modellen är att den endast hanterar agenters kognitiva processer, utan att ta hänsyn till andra, exempelvis sociala, aspekter av deras agerande. Många Multi-Agent System (som kan läsas om i avsnitt 2.7)kräver alltså att dessa aspekter implementeras utanför BDI-arkitekturen för att få tillräckligt djup i systemet. [22]

2.6 Spelbiblioteket Slick2D

Slick2D är ett så kallat spelbibliotek, det vill säga ett API som tillåter en utvvecklare att använda fördefinierade funktioner för att lättare uppnå sitt mål. Slick2D är en vidarebyggnad på LWJGL (Light-Weight Java Game Library), ett kraftfullt bibliotek som dock är svårt att använda.

2.6.1 Light-Weight Java Game Library

LWJGL utgör grunden för Slick2D, och är skapat först och främst för att öka prestandan i spel skrivna med Java. LWJGL är också designat för att vara plattformsoberoende och där med underlätta utveckling för en rad olika enheter [23]. Dock är LWJGL svårt att arbeta i direkt och därmed finns det påbyggnader, såsom Slick2D, vilka har som mål att underlätta utvecklingen och tillföra ytterligare användbarhet.

2.6.2 Slick2D

Slick2D kan, liksom LWJGL, användas för ett flertal plattformar och underlättar ytterligare utvecklandet av tvådimensionella spel i programmeringsspråket Java. Slick2D tillför många verktyg och färdigskrivna klasser som man som utvecklare kan använda sig av direkt, bland annat stöd för bilder, animationer, patriklar, ljud och musik. Dessutom finns det färdigt stöd för väldigt många funktioner som behövs i ett spel. Exempel på dessa är pathfinding och möjligheten att nyttja rutnät för att bygga upp spelvärlden. En av de viktigaste funktionerna som Slick2D tillför är möjligheten att bygga upp ett tillståndsbaserat spel. Detta samt pathfinding beskrivs ytterligare nedan.

Tillståndslös spelstruktur

Denna struktur används framförallt för mindre program som enbart består av ett tillstånd, det vill säga en typ av läge som programmet kan vara i. Detta är lättare att implementera i och med att man inte behöver ha stöd för flera tillstånd, men begränsar därmed även programmet.

Tillståndsbaserad spelstruktur

I den tillståndsbaserade strukturen befinner sig mjukvaran vid alla tillfällen i något tillstånd, definierat av programmeraren. Vid vissa händelser växlar programmet till ett annat tillstånd. Ett exempel som är naturligt förekommande för spel är att menyskärmen utgör ett tillstånd och själva spelet ett eller flera andra tillstånd.

Användandet av denna struktur underlättar strukturerandet av implementationen och även applikationens modularitet.

Pathfinding

Pathfinding är ett vanligt problem inom spel. Problemet handlar om att på ett effektivt sätt hitta den kortaste vägen, med ett antal restriktioner i åtanke, mellan två punkter. Dessa restriktioner kan vara allt ifrån att spelets karaktärer inte kan gå igenom väggar till att det är mer kostsamt att vada genom ett träsk än att ta vägen runt. Slick2D implementerar detta med hjälp av en algoritm kallad A*(uttalas 'AStar') [24], vilken är en välkänd och mycket effektiv metod för att hitta en väg mellan två punkter. A*-algoritmen bygger på Dijkstras algoritm, en sök-algoritm som hittar kortaste vägen till målet i ett nätverk av noder. A* förbättrar dock Dijkstras system genom att implementera så kallad heuristik, som tillåter att vissa begränsningar och regler bestäms för sökningen [25], till exempel att sökning åt norr prioriteras om målobjektet ligger norrut.

2.7 Multi-Agent System

Multi-Agent System (MAS) är en beteckning på ett vanligt sätt att modellera ett system som har många, individuella agenter. Det går ut på att systemets enheter agerar autonomt och intelligent i en miljö, och att de inte styrs av någon central AI, utan snarare fattar beslut individuellt. [26]

En viktig begränsning som rör agenterna i Multi-Agent Systems är att de inte ska total kontroll över miljön, utan bara kan påverka denna partiellt. Detta leder till att handlingar och beslut inte behöver vara entydigt bestämda med tillgång till en viss indata, vilket alltså innebär att systemet är icke-deterministiskt och decentraliserat. En annan funktion i Multi-Agent Systems är att miljön är dynamisk. Ett flertal processer påverkar omgivningen samtidigt, vilket gör att förändringarna är utom kontroll för den enskilde agenten. [27]

Multi-Agent Systems möjliggör studier av agenternas växelverkan med varandra och miljön. På detta sätt kan alltså helheten observeras och analyseras på en makronivå. Multi-Agent Systems är en modelleringstyp som används i stor utsträckning och inom många fält. Exempel på detta är för att skapa animationer med realistiska gruppbeteenden för både spel och film, visualisera arkitekturella modeller och simulera tekniska system med parametrar såsom hållbarhet eller framkomlighet.[28]

3

Visioner för produkten

I det här kapitlet beskrivs den initiala vision av produkten samt vilka mål som sattes från början och vilka avgränsningar som gjordes för att inte göra projektet för stort. Den första rubriken fokuserar på hur visionen av själva spelet såg ut och den andra beskriver den AI som skulle ligga bakom alla bybors beteende.

3.1 Beskrivning av spelet

Spelets fokus ligger i att utforska möjligheterna till ett självständigt spel där spelaren har en del, men tämligen begränsade, interaktiva möjligheter. Vid uppstartandet av spelet genereras en värld med miljöobjekt såsom träd, vatten, plantor och djur. Utöver detta kommer minst en bebodd by att skapas. Byborna kommer i sin tur att ha individuella egenskaper såsom namn, ålder, kön, behov (exempelvis hunger, törst, energi, socialt), flera personlighetsdrag, aktuella tankar, minnen, relationer med andra bybor och så vidare.

Spelaren innehar på något plan rollen som gud över denna genererade värld. Fastän spelaren inte kommer att vara allsmäktig på något sätt, innehar denna ändå flera gudalika krafter, varav en av de mest framträdande är förmågan att påverka bybors behov. Utöver detta ska man ha förmågan att när som helst direkt kunna påverka en invånares känslor. Därigenom kan man ändra utfallet av olika scenarion på både lägre och högre nivåer, såsom att få två bybor att slåss eller skapa rivalitet mellan hela byar.

I varje by bor en så kallad Village Elder, en Villager som är unik på ett antal sätt och bär en krona kallad The Elder Crown, efter vilken spelet är döpt. Denna Elder besitter viss auktoritet och kan därmed påverka simulationsprocessen på ett betydligt större sätt än andra bybor. Denna Villager ska planera byns fortskridande, exempelvis genom att tilldela byborna uppgifter utefter vilka behov byn har för tillfället.

3.2 Beskrivning av AI

Ett AI behövdes för att få en simulation där agenterna faktiskt ska kunna fatta några beslut. En generell tanke är att varje Villager har sitt egna "mindre" AI, som påverkas av dess basala behov, men också på extern stimulans i form av kommenderingar från Village Elder. För att avgöra reaktionen på extern stimuli kontrolleras först de basala behoven. Kommer en Villager att samla trä när den är utsvulten, även om Village Elder har givit en sådan order? Eller kommer den att trotsa ordern och se till sina egna behov?

Även olika personlighetsdrag och minnen vägs in. En stark, hårt arbetande bybo som avgudar Village Elder kanske skulle kunna lyda, medan en mer rebellisk eller lat Villager kan samla mat istället.

Ovanpå detta finns även en AI implementerad som en del av Village Elder. Denna AI ska inte bara ta hänsyn till sina egna behov, som är desamma som hos andra bybor, men också behoven hos andra. Detta görs genom att exempelvis organisera matsamlare och grupper inom skogsavverkning för att samla resurser. Här kommer även givandet av det dagliga tidsschemat som ges till alla Villagers, samt reaktioner på matbrist in.

AI:n var uppenbarligen en av de största delarna av projektet, då det måste reagera på många och varierande situationer samt minnas specifika, viktiga händelser. Designen av detta krävde därmed omfattande efterforskning.

3.3 Sammanfattade mål

Idéer angående vad produkten skulle resultera i formulerades tidigt. De krav som specifikt definierades för att produkten skulle bedömas som fungerande var:

- Att produkten byggde på en fungerande, väl genomtänkt och välstrukturerad spelmotor.
- Att det i spelet fanns en slumpgenererad värld som hade möjlighet att ge bybor det de behövde för att överleva, samt att slumpgenereringen var såpass balanserad att överlevnad för bybor alltid förblev möjligt.
- Att spelvärlden kunde befolkas av bybor som var simulerade på ett sätt som gjorde dem till en del av spelvärlden, snarare än oberoende objekt.
- Att byborna hade en AI som kunde prioritera, planera och utföra handlingar på ett sätt som kunde beskrivas som rationellt.
- Att bybor hade möjlighet att skapa sociala relationer samt att systemet tillät att dessa relationer påverkade hur de interagerade med varandra.

Eftersom att The Elder Crown var tänkt att bli ett spel så var det underförstått att en eventuell utvärdering av produkten skulle vara svår att genomföra. Detta var ett faktum

då ett spel i första hand bedöms efter subjektiva värderingar, snarare än objektiva. Olika spels underhållningsvärde varierar kraftigt beroende på spelare. Samtidigt så kan vissa spel vara tekniskt avancerade och fulländade, men ändå uppfattas som tråkiga. Trotts detta så sattes också mål upp som skulle kunna definiera en underhållande produkt. Dessa punkter sågs dock snarare som exempel på vad simulationen skulle resultera i, snarare än konkreta mål. De mål som förhoppningsvis skulle uppnås för att göra det fungerande spelet underhållande var:

- Att byborna var tillräckligt sofistikerade för att kunna skapa intressanta sociala fenomen, såsom till exempel slagsmål eller vänskap.
- Att bybors personligheter på ett rationellt sätt speglades i det de gjorde och hur de tänkte, så att spelaren kunde få en trovärdig överblick över dem.
- Att oförutsägbara händelser skulle kunna ske, på ett sätt som antingen belönade eller utmanade spelaren. Till exempel om byns Elder blev sjuk eller att någon i byn mördades.
- Att spelaren skulle ges verktyg för att interagera med byborna och att dessa verktyg var av indirekt karaktär, exempelvis genom att påverka bybornas känslor.

4

Metod

Detta avsnitt ämnar gå igenom hur projektet realiserats med avseende på teorin och praktiken bakom utförandet.

Utvecklingen av spelet skedde i enlighet med Scrum, som förklaras närmare i avsnitt 4.1. Regelbundna möten hölls, minst ett per vecka, där projektets status och utvecklingsrelaterade frågor diskuterades. Utöver detta upprätthölls även flera olika kanaler för kommunikation för att säkerställa snabb respons på eventuella frågor eller kommentarer. För att strukturera ordningen som spelets olika funktioner skulle implementeras i sattes även en preliminär tidsplan upp. I och med att alla då var informerade om aktuellt fokus var detta ett sätt att effektivisera arbetet.

På grund av projektets storlek var det viktigt att koden hölls strukturerad. För att säkerställa detta samt för att möjliggöra kollaboration och versionshantering så användes mjukvaran Git samt tjänsten GitHub.

4.1 Scrum

Utvecklingen har framför allt skett genom Scrum-processen. [29] Denna process går ut på att en så kallad 'project backlog' byggs upp. I denna finns alla funktioner som ur ett kund-/spelarperspektiv bör vara implementerade i slutprodukten. Dessa delas upp i delfunktioner, vilka sedan prioriteras och tilldelas poäng efter hur mycket tid de uppskattas ta att implementera.

Ur project backlog skapas sedan en sprint, vilket kan ses som en iteration i utvecklingen, där utvecklarna väljer de högst prioriterade delfunktionerna och implementerar dessa under sprintens gång, vilken vanligtvis pågår i en eller ett par veckor. Efter sprinten hålls ett möte där den utvärderas, och nästa veckas sprint planeras. Denna iterativa

metod används för att planera och effektivisera utvecklingsprocessen.

Vid utvecklingen har en förenklad variant av Scrum använts, med sprintar som varat en vecka. Också de dagliga möten som vanligtvis är en del av Scrum har bytts ut mot kontinuerlig kommunikation via de digitala kommunikationskanalerna.

4.2 Programmeringsspråk och Miljö

I utvecklingen av spelet har det objektorienterade programmeringsspråket Java använts. Detta valdes eftersom att det är språket som projektmedlemmarna hade störst kunskap inom och för att det bedömdes vara ett robust och säkert alternativ till många andra språk.

Efter att detta val gjorts kom även möjligheten att avgöra vilken programmeringsmiljö som skulle användas. Ett naturligt val som medlemmarna sedan tidigare hade erfarenhet av blev då programvaran Eclipse. Andra fördelar med denna miljö är att den har stöd för ett flertal olika operativsystem, automatisk generering av viss kod samt kan hantera trädstrukturen bland klassfilerna.

5

Implementation

I detta kapitel beskrivs hur The Elder Crown implementerades. Det tar dels upp vilka designmönster som följdes, och vilka bibliotek som användes under utvecklingsprocessen. Här beskrivs också mer detaljerat hur simulationens arkitektur ser ut, och mer specifikt hur världen genereras och är uppbyggd, samt hur byborna resonerar och fattar beslut.

5.1 Gränssnittet Model-View-Controller

Systemet har implementerats med ett Model-View-Controllergränssnitt, vilket är ett designmönster som innebär att mjukvaran delas upp i tre delar. Den första delen, Model, representerar en modell av spelvärlden och innehåller således all data som associeras med varje objekt. Var och en av dessa enheter har även en motsvarande grafisk representation i mjukvarans andra del, View, som hanterar sin egen grafiska representation i spelfönstret. Varje Viewobjekt lyssnar på sin motsvarighet i Model och uppdaterar sin grafik då informationen lagrad i modellen förändras. Den tredje och sista delen kallas för Controller. Denna behandlar bland annat indata från användaren och kan framkalla förändringar i modellen.

Hela mjukvaran bygger på detta utvecklingsmönster, vilket har underlättat strukturerandet av kod, samt gjort det lättare att utöka med ny funktionalitet på ett enkelt sätt.

5.2 Spelbibliotek

För att underlätta uppbyggnaden av spelet utnyttjades ett färdigt spelbibliotek. Efter efterforskning avgjordes att biblioteket och spelmotorn Slick2D skulle användas i och med att det är vältestat och användarvänligt. Mer detaljerad information om hur Slick2D fungerar står att läsa i kapitel 2.6.

Många delar av spelet, som exempelvis grafik, Pathfinding och kollisionshantering ges i princip färdigt av biblioteket, vilket leder till en snabbare och mer effektiv utveckling där man inte behöver uppfinna hjulet på nytt varje gång det skall användas. Vissa delar behöver dock byggas ut manuellt, då de inte kan skrivas generella nog för att utnyttjas i alla spel. The Elder Crown använder sig till exempel av pathfinding från Slick2D, men för att hitta den punkt som byborna skall navigera till måste egen kod skrivas. Detta beskrivs mer ingående i avsnitt 5.7.4.

I detta projekt användes även Slick2Ds klasser och struktur för att hantera spelets olika tillstånd (startmeny, körande och pausad), samt en färdig struktur för att hantera en spelkarta byggd på ett rutnät, vilket hade tagit mycket tid att skriva på ett bra sätt från start.

5.3 Intent-Plan-Action

En omfattande del av arbetet har gått ut på att designa och implementera vår egen arkitektur för spelets AI, hur byborna resonerar och planerar. Detta system kallas IPA, och är döpt efter sina beståndsdelar Intent, Plan och Action. Denna grundar sig på Belief-Desire-Intentionmodellen som förklarats i avsnitt 2.5. Nedan följer en beskrivning av IPAs olika beståndsdelar, samt hur de relaterar till BDI-modellen. En övergripande bild av systemet kan även ses i figur 5.1.

Intent i IPA motsvarar ungefär Desire i BDI-modellen, det vill säga vad en Agent har för avsikt eller önskan att utföra. Flera Intents kan finnas samtidigt, i vilket fall de rangordnas internt. Hur en Agent rangordnar intents, dvs. bestämmer sig för vad denne ska göra, förklaras närmare i avsnitt 5.7.2.

Plan är den andra stora komponenten i systemet, som går ut på att en Agent tänker ut exakt hur en specifik Intent ska genomföras. Detta gör den genom att utgå ifrån in världsuppfattning och dela upp planeringen i mindre delmål. En Plan kan alltså ses som en konkretisering av en Intent. Hur Agents planeringsalgoritm fungerar förklaras mer utförligt i avsnitt 5.7.3.

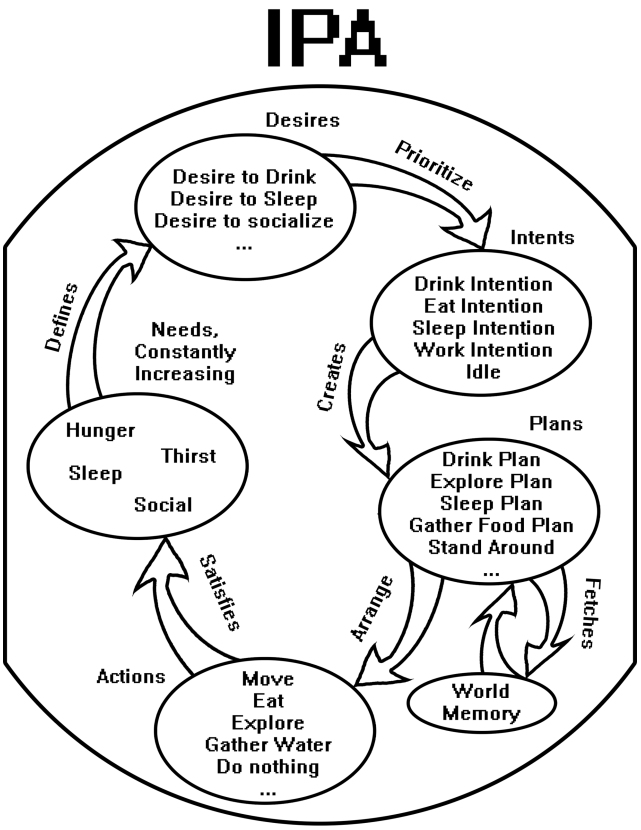
Den tredje och sista beståndsdelen i modellen är Action, som svarar mot Intention i BDI (se avsnitt teoriavsnittet 2.5). Denna del är av mer konkret natur och består av något specifikt som kan utföras, ofta som en del av en större plan. Om en Agent till exempel planerar att dricka vatten, så yttrar det sig i konkreta Actions för att gå till vattnet, och sedan för att dricka det.

Belief-delen av BDI har till skillnad från Desire och Intention inte någon motsvarighet i IPA-strukturen. Trots detta har denna beståndsdel implementerats i mjukvaran. Detta sker exempelvis genom att varje agent får sinnesintryck, så kallade Perceptions,

av sin omgivning och använder dessa för att bilda sig en uppfattning av sin omvärld.

Varje Agent har alltså en egen intern karta över vad denne vet om världen, som inte nödvändigtvis stämmer överens med den verkliga världen lagrad i modellen. Detta ger upphov till en verklighetstrogen gestaltning av att det byborna vet inte alltid stämmer överens med verkligheten, vilket är i enlighet med de specifikationer som berör denna del i avsnitt 2.5.

Detta IPA-system har designats och specificerats tidigt under projektets gång, och det har underlättat strukturerandet av kod och skapat mallar för hur nya funktioner ska implementeras.



Figur 5.1: Struktur av IPA-systemet

5.4 Världsgenerering

Världen i The Elder Crown är slumpgenererad för att göra varje spelomgång unik. Positioner för vatten, träd, byar och bybor påverkas av denna generering. Genereringen kan sägas ske i tre steg: Initieringen av infrastrukturen, skapandet av kartan samt skapandet

av agenter.

I det första steget så initieras samtliga listor för alla Entities i världen och positioner för byarna sätts ut. Konstanter som styr diverse algoritmer för genereringen sätts och tiden på dygnet då simuleringen ska starta bestäms. Detta steg ser till att alla resurser finns på plats inför skapandet av världen.

Sedan inträffar själva skapandet av världen, vilket är den mest sofistikerade delen av världsgenereringen. I detta steg skapas terrängen, träden och byggnaderna. Det som i huvudsak särskiljer olika kartor från varandra är genereringen av vattnet. Vattnet är av stor vikt då det är bybornas enda källa för att släcka sin törst samt huvudsakliga hinder för att ta sig runt i världen. Algoritmen som genererar vattnet använder sig av tre parametrar. Genom att ge olika värden till dessa parametrar kan en mängd olika resultat uppnås. När vattnet genererats så börjar byarna att skapas. Byarna består av tre komponenter: Ett antal hus, en brunn och ett matförråd.

Husen som byborna bor i är dynamiska objekt som byggs upp av många små delar snarare än ett enda stort objekt. Varje hus består av väggar, golv och sängar som alla är fristående, men beroende av rutnätet. Detta tillåter att hus i teorin kan genereras precis hur som helst, men för realismens skull så finns det fördefinierat hur ett hus kan se ut. När ett hus har fått ett väderstreck för dörren så placeras väggarna ut. Efter detta så placeras sängar ut i huset intill väggarna på ett sätt som är tänkt att minska kollisioner mellan villagers, till exempel i hörn, och golvet genereras.

Brunnen och matförrådet är endast en ruta stora och består av enskilda, fristående objekt. Egentligen finns dock inget hinder för att placera brunnar och matförråd i hus precis som sängar. När byarna har genererats färdigt så sätts träd ut över hela kartan och alla eventuella tomrum fylls med gräs och blommor.

Byborna är det sista som skapas. Bybor placeras ut i de byar som finns på kartan så att alla byar får lika många invånare. Alla bybor är unika på så sätt att deras behov varierar. Vissa bybor blir hungriga fortare än andra, andra blir lättare törstiga och så vidare. Bybor får också varsitt namn som beror på deras kön. En Village Elder skapas också i varje by (Se 6.1.9).

5.5 Världssimulation

Det finns två koncept som utgör en värld, nämligen entiteter och agenter. Entiteter representerar alla fysiska ting som existerar i världen, vare sig det är träd, marken eller levande varelser. Entiteter har en plats i världen och det är dem man interagerar med. Olika entiteter har olika egenskaper, t.ex. så kan man gå på vissa, medan andra blockerar vägen.

Allt som på något sätt skall kunna utföra handlingar i världen är Agents. Detta kan t.ex. vara att röra på sig, plocka frukt från träd eller äta. En Agent har inte en fysisk representation i världen, utan är bara ett abstrakt ting med en vilja som kan utföra handlingar. För att en agent skall kunna ha en fysisk representation måste den också vara en entitet. Objekt i världen kan således vara både en entitet och en agent på samma gång, och detta är fallet för t.ex. Villagers. I nuläget är det endast Villagers som är Agents.

Agenter och entiteter lagras i samlingar hos världen där de binds till en viss position i världen. Detta innebär att en given entitet och agent kan inte direkt förändra sin egen position, utan måste gå via världen för att göra det.

5.5.1 Entiteters interaktion med världen

I världen finns det tre olika lager av entiteter, där två från samma lager aldrig kan finnas på samma position. Dessa lager består av ett undre lager för mark och vatten, ett mellanlager för t.ex. Villagers, trädstammar och husväggar, samt ett taklager för allt som skall finnas ovanför marknivån, exempelvis trädkronor.

Alla entiteter i världen är dock inte helt statiska ting, utan vissa kan förändras över tid. Det finns exempelvis träd som bär frukt. Den här frukten kan plockas av agenter, för att sedan växa tillbaka med tiden. För att detta skall ske måste entiteten också veta att tiden faktiskt går. Vissa entiteter kan därför uppdateras av världen och de finns lagrade i en egen samling. Varje gång världen uppdateras går den igenom den här samlingen och säger till varje entitet att uppdatera sig själv. Entiteten är sedan själv ansvarig för vad den skall göra, i fallet med fruktträdet är det att öka sin frukthalt med en liten summa.

5.5.2 Agenters interaktion med världen

Allt som på något sätt vill påverka världen är agenter. Detta är viktigt att skilja från att bara påverka sig själv, då information inte behöver beröra världen i övrigt. Vi såg tidigare exemplet med fruktträd som kunde öka sin egen frukthalt. Då behöver fruktträdet inte påverka världen utan kan förändra sig själv direkt.

Varje gång världen uppdateras så går den igenom och uppdaterar sin samling av agenter. Detta innebär att den först ger agenten ett antal synintryck. Dessa inkluderar position och allt som agenten ser just nu. Det är sedan upp till agenten själv att tolka det den ser och utifrån detta välja vad den vill göra. Den skickar sedan tillbaka vad den vill göra till världen i form av en Action. Världen skall nu kolla på vad agenten vill göra, och sedan bestämma om det här är möjligt.

Agenter har inte tillgång till sin egen position och de har endast en personlig karta som inte nödvändigtvis stämmer ihop med den faktiska världen. Den personliga kartan

består av det som en agent har sett och hur det såg ut när agenten såg det. Om en agent ser ett fruktträd fullt med frukt, kommer den alltid att minnas ett fruktträd med frukt. När en annan bybo då plockar all frukt från fruktträdet kommer den personliga kartan och den faktiskt världen inte längre att stämma ihop. Därför skulle det vara olämpligt för en agent att uppdatera sin egen världsrepresentation. Världens uppgift här är då att kolla så att handlingen agenten vill göra går att utföra, och sedan utföra den. Man kan se det som att agenter i det här fallet representerar en hjärna som har en viss vilja att göra något, och världen är alla fysiska lagar som existerar. Hur mycket en person än vill gå igenom en vägg så kommer de fysiska lagarna att stoppa personen.

5.6 Villager

En av de mest centrala och komplexa entiteterna i The Elder Crown är Villager. Det är i detta objekt som all planering beräknas utifrån dess behov och önskningar.

5.7 Behov

De behov som en Villager styrs av är hunger, törst, sömn samt socialisering. Anledningen till att just dessa behov valts ut är att de ansågs vara de mest vitala behoven hos en människa. Behov ökar konstant och pressar ständigt bybon till handling.

5.7.1 Uppdatering

En Villager uppdateras 1 till 1000 gånger varje sekund med nya intryck från sin direkta omgivning (Perceptions), beroende på vilken hastighet som användaren ställer in. Dessa intryck avkodas och uppdaterar stora delar av den information som varje Villager behöver handha, exempelvis, hur världen ser ut, vilka andra entiteter som är i närheten samt vissa delar av statusen hos dessa entiteter. I samma uppdatering så fortgår även uppdatering av alla attribut som en Villager har, såsom ålder, hunger, törst, sociala behov och liknande. Här undersöks också huruvida entiteten borde dö eller inte.

Därpå utförs planeringen av nya val och handlingar utifrån de uppdaterade behoven och intrycken, förutsatt att entiteten inte redan är upptagen av en Plan i vilket fall denna plan fortsätter utföras. Planering är ett stort begrepp inom AI, där det har bedrivits mycket forskning. Detta har här implementerats på två olika nivåer: planering av Intent och planering av Actions.

5.7.2 Planering av Intents

Planering av Intents går ut på att en agent ska fatta ett beslut om vad denne vill göra. Det kan finnas flera motstridiga impulser inom en agent, om denne till exempel är törstig och ser en kompis på samma gång. Vad agenten då väljer att göra beror på hur denne planerar sina Intents.

Varje Intent har både en Cost och en Desire. Cost betecknar hur tidskrävande det är att utföra en viss handling, emedan Desire betecknar hur mycket agenten önskar det. Det är dessa två egenskaper som avgör vilken Intent som en agent slutligen beslutar sig för att göra, enligt algoritmen.

Detaljerna kring hur Cost och Desire beräknas är upp till varje specifik Intent att bestämma. Cost kommer alltid att vara summan av kostnaderna för varje individuell Action i planen som genereras, hur kostnaden för en Action beräknas är också upp till varje specifik Action.

En Villager har t.ex. alltid en önskan att äta. Detta representeras med en EatIntent, vars Desire beräknas baserat på bybons hunger. Cost för hela vår EatIntent är summan av kostnaderna för alla actions i den EatPlan som genereras vid planering.

Ett exempel: om bybon skulle stå ute i skogen i närheten av ett fruktträd skulle planen med största sannolikhet att bestå av först en MoveAction som säger att bybon skall gå till trädet, sedan en EatAction som säger att bybon skall äta av trädet. Cost för en MoveAction är avståndet man skall gå i den och Cost för en EatAction är 0. Cost för EatIntent i det här fallet skulle alltså vara avståndet från bybon till trädet.

5.7.3 Planering av Actions

När en agent har valt Intent och ska utföra denna, så behöver agenten planera hur denna intent ska realiseras. Med andra ord, ska agenten utifrån den abstrakta idén formulera en konkret plan.

För att realisera en specifik Intent kan det behövas många Actions. En Intent kan till exempel vara att dricka vatten. Detta kommer då att bestå av att först leta fram vatten, sedan att gå dit, och slutgiltigen att dricka. För att hålla koll på dessa små Actions, som tillsammans låter oss tillfredsställa en Intent, har begreppet Plan myntats.

Den struktur för planering som The Elder Crown använder sig av är relativt enkel. För varje Intent så finns det en associerad Plan, som består av ett antal Actions. Dessa actions kan misslyckas på två sätt - antingen så försöker agenten att göra om dem, eller är de sådana att de får hela planen att misslyckas.

Mer avancerade strukturer för planering som stötts på under projektets förstudier tillåter att alternativa Actions väljs om en Action misslyckas, och också att en plan i sin tur kan innefatta flera planer. Behovet för detta uppstod dock inte i projektet, då simulationen simulerar jämförelsevis enkla förhållanden och inte så avancerade situationer.

5.7.4 Pathfinding & Objektsökning

Beslutet togs att utnyttja den inbyggda algoritmen för Pathfinding som finns fördefinierad i Slick2D. Denna algoritm var dock mycket begränsad i sitt användningsområde då kriterierna för vad som ska hittas nödvändigtvis måste skräddarsys efter applikation. För The Elder Crown specifikt så behövdes ett system som först kunde identifiera objekt för att sedan använda Slick2Ds A*-algoritm (Se **??**), då denna endast hittar vägen till ett objekt om målpunkten redan är känd.

Denna objektsökning behövde hitta rätt objekt i världen, men också hitta detta objekt i rätt tillstånd. Till exempel vill en hungrig bybo knappast hitta träd utan frukt. Objektsökaren söker därför inte bara efter objekt utan kräver också att objekten i fråga uppfyller vissa kriterier, såsom till exempel ”har mat”-, eller ”har vatten”-kriterier, för att sedan använda en variant av Dijkstras algoritm för att skanna världen runt bybon. [25] Detta medför dessutom att bybon alltid hittar till det närmaste alternativet.

Om objektsökaren misslyckas med att hitta något objekt i bybons minne som uppfyller det sökta kriteriet, så börjar bybon att utforska världen i hopp om att hitta det den söker.

6

Resultat

Under projektets implementationsfas har en fungerande världssimulation med egenhändigt tänkande bybor implementerats. Under denna rubrik beskrivs simulationen som projektet har resulterat i mer utförligt, på vilket sätt denna kan användas för att simulera verkliga förhållanden i enlighet med syftet, och till slut så utvärderas huruvida de målsättningar som uppsattes i projektets början har uppnåtts.

6.1 Produkt

Slutprodukten av arbetet är en världssimulation, som i stora drag uppfyller de mål och krav som vi ursprungligen definierade som produktens visioner (se avsnitt 3). Den stödjer existensen av godtyckligt många bybor, och den tillåter byborna att påverka både världen och varandra.

Nedan kommer en beskrivning av först bybornas ”hjärnor”, hur de i dagsläget resonerar och planerar. Sedan följer en mer detaljerad beskrivning av världen, och simulationens olika kapabiliteter.

6.1.1 Villagers

Simulationens bybor tänker och handlar individuellt, fattar beslut och planerar sitt agerande beroende på vad de har för uppfattning om världen.

Varje bybo har sina egna behov, och väljer vad den vill göra baserat på dessa. De grundläggande behoven är hunger, törst, sömn samt social interaktion, vilket kan visas individuellt för varje bybo i enighet med figur 6.1. När bybon bestämt sig för vad denne vill göra, planerar den sedan vilka handlingar som måste utföras för att detta ska bli verklighet. Varje bybo har också sin egen verklighetsuppfattning, som baseras på intrycken av världen som bybon uppdateras med flera gånger i sekunden.

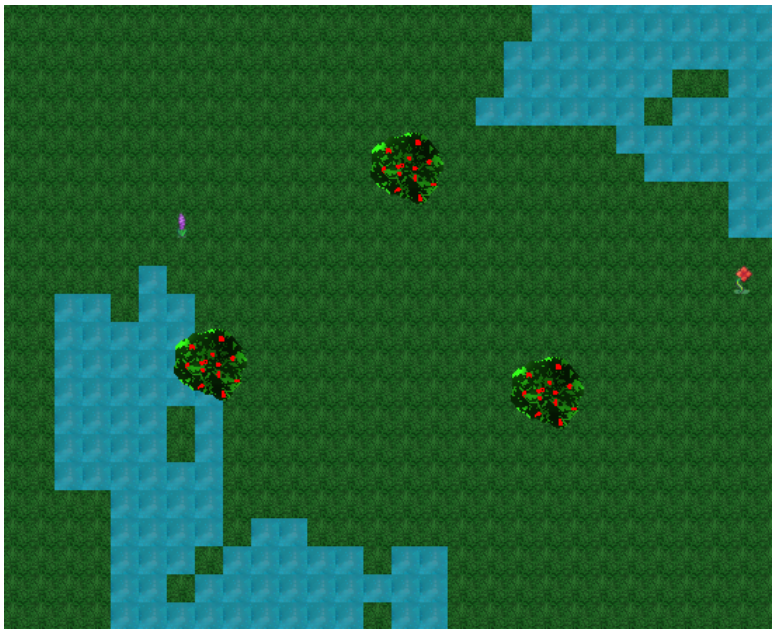
Att bybor på detta sätt är agenter, och fristående från världen, har varit ett stort fokus i projektet från början. Resonemangs- och planeringssystemet var en betydande del av utvecklingen. Nedan följer exempel på scenarion som kan ske i världen, och på hur simulationens bybor påverkas av olika situationer.

Figur 6.1: En bybo och hennes behov

6.1.2 Mat och vatten

Simulationen har allting som krävs för att stödja grundläggande liv: mat och vatten. Maten i simulationen är mer specifikt frukter som växer på träd. Frukterna tar slut när de plockas, för att sedan växa tillbaka. Vattnet finns i form av sjöar, som kan drickas ur men ej passeras (se figur 6.2).

Byborna som finns i världen äter då denna frukt och dricker detta vatten. Att frukten tar slut kan leda till intressanta scenarion, då en by har fler invånare än dess matkällor kan stödja. Detta kan leda till att byborna söker sig till andra områden för att stilla sin hunger, eller att de rent av dör av näringsbrist.



Figur 6.2: Vatten och mat i spelet

6.1.3 Dygnscykel

Förutom mat och vatten, så stödjer världen skiftningen mellan dag och natt. Denna dygnscykelsskiftning sker kontinuerligt under hela simulationens gång, och är något som byborna anpassar sina liv efter.

Precis som verkliga människor, så påverkas simulationens bybor av bristen på solljus. När det blir natt så får byborna en ökad önskan att sova, vilket leder till att de oftast inte är ute och ägnar sig åt andra aktiviteter på natten. Detta kan dock inträffa om behovet är tillräckligt stort.

6.1.4 Byn

I världen finns alltid en eller flera byar, som byborna hör hemma i. Dessa byar består av ett antal hus, samt en plats att samla mat och vatten på. En av dessa byar kan ses i figur 6.3.

När en bybo har fått sina behov tillräckligt tillfredsställda, händer det att denne ger sig ut och samlar mat och vatten till byn. Då en bybo känner en önskan att sova, söker denna sig till det hus den hör hemma i och sover i husets tillhörande säng. Detta tillfredsställer sömnbehovet.

Har bybon inte en tillhörande säng, som fallet blir med nya bybor, så lägger den beslag på närmaste, icke upptagna säng som den känner till. Finns det inga lediga sängar,

eller bybon inte känner till några, så lägger den sig och sover var den än råkar vara då trötttheten blir för stor. Efter en natt av sömn utan säng känner sig bybon motiverad att lösa detta problem och bygger således ett hus till i byn.

Dessa byar lägger till en dimension till simulationen, då de tillåter att flera civilisationer samexisterar i samma simulation. Detta kan göra simulationen intressantare att bevittna, då det kan gå olika bra för de olika samhällena. Att ha flera samhällen ökar också den akademiska nyttan, då man lätt kan ge olika samhällen olika förutsättningar, och på detta vis testa olika sociala teorier.

Figur 6.3: Byn och dess invånare

6.1.5 Graviditet och barnafödsel

Simulationen stödjer också att byarna får nya invånare, genom att byborna kan föröka sig. När byborna går och lägger sig för natten, finns möjligheten att en man och en kvinna lägger sig i samma säng. Detta kan (men behöver inte) resultera i en graviditet.

Den kvinnliga bybon blir då gravid, och efter en tidsperiod som motsvarar ungefär 9 månader föds en ny bybo. Detta tillåter samhällen att fortsätta utvecklas och simulationen att köras under en längre tid, vilket är önskvärt.

6.1.6 Sjukdomar

Under simulationens gång kan bybor ibland få sjukdomar. För att inte göra simulationen för komplex så har dessa sjukdomar ingen direkt källa utan uppstår slumpmässigt. En smittad individ kan sprida sin sjukdom till andra bybor genom att befinna sig i närheten av dem. Sjukdomar gör att det finns en liten risk att bybon dör varje gång den uppdateras, och denna risk ökar ju äldre bybon blir. Med dessa enkla funktioner kan spelet simulera sjukdomsspridning i en mindre grupp av människor.

6.1.7 Arbeten

Bybor kan tilldelas arbeten som innebär att de antingen samlar vatten eller mat och fyller byns matförråd. Arbeten kan ges olika intensitet, vilket avgör hur många timmar en bybo arbetar. En bybo behöver inte arbeta om den har behov som måste uppfyllas, men kommer prioritera arbete relativt högt. Byborna har ett antal platser att förvara föremål och det är dessa platser de använder när de plockar frukt eller fyller kärl med vatten.

6.1.8 Nybyggnation

Varje bybo har möjligheten att utvidga byn som den lever i genom att helt enkelt bygga upp ett nytt hus. Detta görs genom att bybo-objektet skickar iväg en önskan om att det ska byggas ett nytt hus vid dennes by till världen. Världen i sin tur slumpar fram platser, först nära byn, sedan längre och längre bort, och när den hittar en giltig plats för ett hus så konstrueras det med slumpmässiga proportioner. Denna slumpalgoritm genererar mer ostrukturerad infrastruktur vilket är passade då sättningen är relativt primitiv.

6.1.9 Elder

En av de tänkta nyckelaspekterna i The Elder Crown var en så kallad Elder som skulle fungera som en ledarfigur för de övriga byborna. Då sociala interaktioner i princip inte existerar i produkten ännu så har utvecklingen av denna Elder lagts på is. Detta på grund av att interaktioner mellan en Elder och övriga bybor ansågs vara ett grundläggande kriterie för dennes funktioner. Just nu så finns en Elder implementerad, men utan att fylla något särskilt syfte. Den har dock ett unikt utseende.

6.2 Applikationer av programmet

I den primitiva version av mjukvaran så som den är nu kan man, med specifikationer, bland annat simulera effekter av polygami kontra monogami på befolkningsstorlek eller avstånd till resursers inverkan på befolkningsökning. Det är mycket rimligt att simulera scenarion där fortplantning är en central del och det finns även stöd för sociala strukturer då ett ordersystem är fungerande, detta möjliggör då även simuleringar för att undersöka individualismens inverkan på ett samhälle.

6.3 Mål

Projektets mål var medvetet ambitiösa när utvecklingen startade. I dag så är produkten i ett nästan fungerande skick om man utgår från den kravlista som specificerades i 3.3. Slutprodukten idag är stabil och lätt att bygga vidare på, då stor fokus lades på mjukvarans robusthet och utökbarhet. Det finns dock en del punkter som ej uppnåddes, som kommer att diskuteras närmare nedan.

Som programmet fungerar nu så kan vi simulera ett antal agenter som ser till sina grundläggande behov på olika vis i världen. Detta simpla beteende ger upphov till mer komplexa fenomen när fler bybor verkar sida vid sida vilket är grunden i Multi-Agent Systems.

6.3.1 Spel och simulator

En viktig del av projektets syfte var att skapa en slutprodukt som var både underhållande från en spelsynvinkel samt gav en intressant och korrekt simulation av samhället. Slutprodukten, som beskrivs ingående i 6.1, kan ställas mot syftet (1.2) samt mot de mål som finns uppställda under visionen för slutprodukten i 3.3.

I avsnitt 6.1 och 5.4 så uppfylls flera av målen, dock inte alltid till den nivå som var ambitionen vid början av projektet. Något som exempelvis saknas är den sociala interaktionen mellan byborna. De kan träffas och få utlopp för sina sociala behov, dock har de inte personliga minnen eller några mer omfattande relationer med andra enligt de visioner som fanns vid projektets start.

När man jämför detta med projektet syfte så är det till viss del uppfyllt. Ett spel som bygger på en simulator är implementerat, dock med betydligt mer begränsad interaktion från spelaren än vad som var tänkt. På grund av detta har det även varit svårt att på ett tillförlitligt sätt mäta något slags underhållningsvärde i produkten mer än glädjen över att se samhället utvecklas.

6.3.2 En fungerande produkt

Samtliga av de målfunktioner som hade satts upp på listan över krav för en underhållande produkt finns alltså inte implementerade i dagsläget. Listan över krav för en fungerande produkt har dock nästan helt uppfyllts.

Vi har framgångsrikt utvecklat en spelmotor som bygger på den model-view-controller-arkitektur som beskrivs i 5.1. Att helt separera modell, vy och kontroller har underlättat övrig implementation betydligt, men har också, förhoppningsvis, inneburit en bättre prestanda för spelaren. I denna spelmotor har vi lyckats implementera en spelvärld som, trots tveksam grafisk estetik, uppfyller alla de krav som ställts för att möjliggöra utvecklingen av byborna. Den har dessutom byggts på ett sätt som gör implementationen

av nya funktioner relativt enkel. Systemet för världsgenerering är slumpartad och stabil, på så sätt att unika världar skapas utan att bybor sätts i ohållbara situationer. Då vi avgränsat oss från grafisk skönhet kan spelvärlden sägas vara lyckad.

Kravet på att bybor skulle modelleras som individer i en faktisk värld har även det uppfyllts. Byborna är helt integrerade i spelvärlden och interagerar med den på samma villkor som andra entiteter, såsom exempelvis träd. Världen sätter stopp för bybor som försöker vandra där det inte är möjligt och den uppdaterar bybornas handlingar likt den uppdaterar tiden på dygnet eller återväxten av frukt. Byborna är alltså implementerade på ett sätt som tvingar dem att anpassa sig efter de begränsningar som världen sätter, vilket i sin tur innebär att externt kodade lösningar som annars hade kunnat lösa diverse problem inte har varit möjliga att implementera. Det har förstås inneburit mycket extra tid för testning och felsökning, men ansågs samtidigt vara nödvändigt för att bibehålla en trovärdig simulation.

I visionen för spelet så specificerades även en huvudpunkt för vad en lyckad AI. skulle bestå av. Med IPA-systemet har byborna i spelet möjligheten att känna av sina behov, prioritera sina intentioner och planera sina handlingar. Dock så är det svårt att avgöra om rationaliteten är på en önskvärd nivå då bybornas behov fortfarande är såpass primitiva att icke-rationella handlingar knappt existerar. IPA har dock varit framgångsrikt nog för att låta bybor överleva genom att äta, sova, dricka och orientera sig runt i världen. Frågan om huruvida bybor hade handlat lika effektivt i diverse sociala situationer kvarstår dock. Men bortsett från hur logiska bybornas handlingar egentligen är, så har IPA-systemet uppfyllt de krav som ställdes på en fungerande AI.

Slutligen så fanns ett krav på att bybor skulle kunna socialisera med varandra och detta var också kärnan för vad som skulle göra simulationen intressant och därmed underhållande. I dagsläget så finns dock endast en funktion för socialisering som innebär att bybor förstår när de är i närheten av andra bybor och därigenom tillfredsställer sitt sociala behov. Detta är dock inte vad som specificerats i kravlistan eller spelbeskrivningen och därmed har kravet inte uppfyllts.

6.3.3 Underhållningsvärde och Spelarglädje

Bra spelarinteraktion var en av de saker som tyvärr fick utelämnas ur slutprodukten, på grund av tidsbrist. Fokus har istället lagts på att få simulationen att fungera så bra som möjligt. Av denna anledning har ett av projektets mål, att bygga en intressant och spännande simulation med underhållingvärde för konsumenter, ej realiserats.

6.3.4 Akademisk nytta

I avsnitt 1.2 klargjordes att en del av projektets syfte var möjligheten att utnyttja programvaran för att genomföra studier på hur olika parametrar i mjukvaran påverkar simulationsprocessen och därigenom dra paralleller till verkligheten. Exempel på denna

typ av experiment som nämndes i avsnittet var resultatet av polygami gentemot monogami, sjukdomsspridning samt balansering av populationen efter mängden tillgänglig föda.

Undersökning av effekterna av polygami samt monogami krävde ett fungerande reproduktionssystem. Som kan läsas i avsnitt 6.1.5 implementerades ett fungerande, om än primitivt, sådant. Det får plats två personer i varje säng och som nämns i avsnitt 6.1.4 har byborna som standard en egen säng som de återkommer till var natt. De prioriterar att para ihop sig så att två av olika kön hamnar i samma säng (förutsatt att detta är möjligt). Detta definieras då som ett monogamt samhälle.

För att då testa polygami (i detta fall heterosexuell sådan) ändrades kravet att byborna har en designerad säng, utan istället sover i olika sängar och därigenom potentiellt sover med olika partners varje natt. Det senare alternativet visade sig (kanske något motsägelsefullt) resultera i färre barn. Detta var främst på grund av att en Villager inte nödvändigtvis prioriterar att sova med någon annan över huvud taget, utan kan välja att sova själv då antalet lediga sängar är många.

Som kan läsas i avsnitt 6.1.6 finns det stöd för sjukdomar och sjukdomsspridning. Detta är emellertid så primitivt att inga riktiga slutsatser kan dras genom att iaktta simulationen. På denna punkt uppfyller alltså programvaran inte målet.

Befolkningstillväxt baserat på tillgång till exempelvis föda kan studeras i simulationen. I avsnitt 6.1.2 kan det läsas mer om hur hunger och föda implementerats i produkten. Med dessa fenomen implementerade var det möjligt att undersöka befolkningsmängden beroende på detta. Det kunde då bevittnas att folkmängden gick upp då tillgången på föda var stor, för att sedan nå en viss gräns där mängden föda inte längre kunde mätta byborna, vilket resulterade i en befolkningsminsking. På detta sätt kunde man se en pendlande folkningsmängd. Då detta ligger nära verkligheten kan produkten sägas uppnå detta mål.

Totalt sett kan det alltså konstateras att målen nåtts till varierande grad. Alla de fenomen som nämndes i 1.2 har kunnat undersökas, men vad gäller resultatet från undersökningarna som underlag för simulationens korrekthet nåddes inte målet helt. Detta då studierna inte nödvändigtvis motsvarar de resultat som liknande undersökningar i verkliga livet gett upphov till.

7

Diskussion

Detta kapitel inkluderar reflektioner kring problem som stöttes på under utvecklingen, och det skifte av inriktning som gjordes under projektets gång. Till slut diskuteras möjliga vidareutvecklingar av applikationen.

7.1 Kritisk diskussion

I tidigt planeringsstadium var gruppen väldigt ambitiösa och optimistiska. I den preliminära tidsplanen hade det ställts upp som mål att produkten skulle ha uppnått MVP(Minimum Viable Product)-status 1:a mars. Det skulle komma att visa sig flera gånger om hur grovt tidsåtgången underskattades för de olika funktionaliteter som önskades implementeras. Detta beror till största del på bristande erfarenhet inom Multi-Agent Systems och agil mjukvaruutveckling samt en svårighet att synkronisera strukturen på koden som de olika gruppmedlemmarna skrev.

De problem som uppstod på grund av detta visade sig mycket svåra för gruppen att förutse och krävde mycket tid och energi att tillrättalägga. Åtgärdandet av dessa problem gjordes oftast parallellt med vidareutveckling av produkten, då det var mycket ineffektivt för hela gruppen att fokusera på samma problemlösning. Detta gjorde dock att vidareutvecklingen ofta byggde på de delar i programmet som kom att förändras i och med problemlösningen, vilket i sin tur gav upphov till allt fler problem då även vidareutvecklingen blev tvungen att åtgärdas.

En av de största lärdomar som gruppens medlemmar tar med sig från detta projekt är således att det, i långa loppet, kan vara mer givande att stanna upp och lösa ett problem som uppstår helt, innan man går vidare och bygger in nya spännande funktioner. Detta skulle ha varit långsiktigt tidsbesparande, även om vissa nackdelar finns. Exempelvis kan det tänkas vara ett relativt ineffektivt sätt att arbeta, då stora delar

av programmet rimligtvis icke påverkas av ett problem och således kan vidareutvecklas utan att skapa fler. Att behöva vänta med vidare implementation bidrar till frustration. Sammanfattningsvis så det antagligen varit till projektets fördel att använda sig av denna metod, eftersom det kan vara mycket svårt att veta hur avgränsat ett problem är då oanade nya komplikationer dykt upp som sedan kunnat spåras tillbaka till tidigare versioner.

7.2 Lärdomar

I projektplanen specificerades det att gruppen skulle arbeta med utvecklingsmetoden Scrum med veckolånga sprintar. Detta genomfördes dock med varierande resultat. Bristande närvaro av olika anledningar samt avsaknad av Scrum-master gav högst informella möten vilket hade till följd att gruppmedlemmarna ej fick den överblick av projektet och dess status som hade hjälpt i strukturerandet av hur ny kod skulle skrivas.

Vad gällande ambitionsnivån så står det klart att det är effektivare och framförallt säkrare för slutproduktens resultat med många korta milstolpar där varje milstolpe är en relativt fungerande produkt. Med ”korta milstolpar” menar här att komplexiteten bör hållas till en låg nivå i början men med stöd för bredare påbyggnad. Det misstaget som gjordes i detta projekt var dels ambitionsnivån men även att fokus låg alltför långt in i framtiden.

7.3 Skifte av inriktning

Vid projektets start var visionen att skapa ett spel med spelarinteraktion, även om denna förväntades vara begränsad. Under utvecklingens gång byttes dock inriktning sakta men säkert till att utesluta spelarinverkan nästan totalt. Det nya målet blev att skapa en mjukvara med vilken man kunde, under en lång tid och med olika förutsättningar, experimentera med olika sociala och individuella strukturer. Exempel såsom samarbete kontra självständighet, polygami kontra monogami blev den nya visionen för applikationen.

Huvudanledningen till detta var den begränsade tiden som blev allt kortare då många problem tog en betydande del av den avsatta tiden i anspråk. Gruppen såg ändå ett underhållningsvärde i produkten då idén, redan från början, byggde på simuleringsaspekten av spelet och inte användarens kontroll.

7.4 Möjliga vidareutvecklingar

Det finns flera potentiella applikationsområden för den simulationsgrund som skapats. Mycket funktionalitet ligger även latent som behöver anpassas till den nuvarande versionen. Detta på grund av att medlemmar varit ivriga att implementera ny funktionalitet

och sett ett effektivt sätt att göra det. Dock har problem uppstått som gjort att prioriteringar nödvändigtvis behövts förändras.

Aspekter såsom mer avancerade simulationer av sjukdomsspridning, befolkningstillväxt och inverkan på djur och växtliv ligger inte långt bort i implementeringen. Lite mer ambitiösa applikationsområden innefattar exempelvis religionsspridning (vilket kan tänkas vara likt sjukdomsspridning i detta fall) och evolution av ideologier.

8

Slutsats

Det har varit svårt att planera upp och bygga en Artificiell Intelligens utan tidigare erfarenheter eller ingående kunskap i ämnet. Visionen med projektet var att skapa en simulation av en förhistorisk by, med fristående bybor som agerar autonomt och som interagerar med omvärlden och varandra. Denna simulation skulle utformas på ett sätt så att den blev underhållande att bevittna, och betraktaren skulle även ha begränsade möjligheter att påverka simulationen.

För att åstadkomma detta gjordes mycket förstudier om bland annat AI, Multi-Agent systems och BDI-modellen. Med denna kunskap i åtanke så designades och implementerades mjukvaran. Den består av en fungerande världssimulation, samt ett system för agenters beslutsfattande (kallat IPA-systemet).

Resultatet blev en simulation av en värld som innehåller träd, vatten och byar där bybor bor. Byborna har egna behov som de strävar efter att tillfredställa, och planerar och fattar egna beslut. Byborna kan dock inte interagera med varandra, och spelarinteraktionen fick inte heller plats inom projektets ramar.

Sammanfattningsvis, har det varit ett intressant och lärorikt projekt. Gruppmedlemmarna har fått stor insikt i flera redan existerande teoretiska modeller inom fältet artificiell intelligens, och hur man kan skapa tillämpningar av dessa för att simulera komplexa system av individuella agenter.

Litteraturförteckning

[1] E. Staff, 50 greatest game design innovations, http://www.edge-online.com/features/50-greatest-game-design-innovations/4/ (Nov. 2007).

[2] IGN, Godus preview, http://www.ign.com/games/godus/pc-148566# (September 2013).

[3] T. Blevins, God $%*# this is an amazing game!, http://uk.ign.com/articles/2001/03/27/black-white-3 (March 2001).

[4] EA Games, The sims official facebook page info, https://www.facebook.com/TheSims/info (2014).

[5] Mojang, Minecraft official web site, https://minecraft.net/game (2009).

[6] M. T. Jones, Artificial Intelligence—A Systems Approach, INFINITY SCIENCE PRESS LLC, Hingham, 2008.

[7] R. Epstein, G. Roberts, G. Beber, Parsing the Turing Test, Springer Science+Business Media, Hingham, 2009.

[8] J. R. Searle, Minds, brains, and programs (1980) 417–457.

[9] J. V. LEEUWEN, S. B. COOPER, Alan Turing—His Work and Impact, Elsevier, USA, 2013.

[10] G. Tecuci, Artificial intelligence, John Wiley and Sons, Inc, USA, 2012.

[11] S. J. Russell, P. Norvig, Artificial intelligence: a modern approach, Pearson Education, New Jersey, 2010.

[12] A. Samuel, Some studies in machine learning using the game of checkers, International Business Machines Corporation, USA, 2000.

[13] S. Lucas, Computational Intelligence and AI in Games: A New IEEE Transactions (2009) 1–3.

[14] J. Hendler, Computers play chess; humans play go (2006) 2–3.

[15] P. S. István Szita, Marc Ponsen, Effective and Diverse Adaptive Game AI (2009) 16 – 27.

[16] Tech Terms, Script Definition, `http://www.techterms.com/definition/script` (2014).

[17] Unknown, Skirmish mode, A.I. with attitude, Computer Gaming World.

[18] Hans-Dieter, M. Hannebauer, J. Wendler, Belief-desire-intention deliberation in artificial soccer, Association for the Advancement of Artificial Intelligence, USA, 1998.

[19] A. S. Rao, M. P. Georgeff, BDI Agents: From Theory to Practice, Proceedings of the First International Conference on Multiagent Systems, USA, 1995.

[20] G. Simari, S. Parsons, Markov Decision Processes and the Belief-Desire-Intentions Model, Springer Science+Business Media, London, 2011.

[21] L. Fichera, D. Marletta, V. Nicosia, C. Santoro, Flexible robot strategy design using belief-desire-intention model, Proceedings of the First International Conference on Research and Education in Robotics, Switzerland, 2011.

[22] G. M. P. O'Hare, N. R. Jennings, Foundations of Distributed Artificifial Intelligence, John Wiley & Sons, Inc., Canada, 1996.

[23] The LWJGL Wiki, About LWJGL, `http://lwjgl.org/wiki/index.php?title=About_LWJGL` (May 2014).

[24] K. Glass, AStarPathFinder, Slick2D API, `http://slick.ninjacave.com/javadoc/org/newdawn/slick/util/pathfinding/AStarPathFinder.html` (May 2014).

[25] A. Patel, Introduction to A*, `http://theory.stanford.edu/~amitp/GameProgramming/AStarComparison.html` (2009).

[26] D. Weyns, A. M. . Uhrmacher, Multi-Agent Systems Simulation and Applications, Taylor and Francis Group, LLC, Canada, 2009.

[27] M. Hadzic, PornpitWongthongtham, T. Dillon, E. Chang, Ontology-Based Multi-Agent Systems, Springer-Verlag, Berlin, 2009.

[28] M. Software, massivesoftware.com - simulating life, `http://www.massivesoftware.com/index.html` (May 2014).

[29] S. Staff, Scrum.org - the home of scrum, `https://www.scrum.org/` (Jun. 2014).