

ANT WARZ

Ett digitalt sällskapsspel för handhållna enheter

Per Blåviik
Adam Morén
Marcus Gladh
Isak Engström
Ester Lindgren
Samuel Svensson

Examinator: Daniel Jönsson

Sammanfattning

I den här rapporten beskrivs utvecklingsprocessen av ett sällskapsspel för handhållna enheter, vilket i detta avseende innefattar smarta telefoner och surfplattor. Sällskapsspelet byggs upp med hjälp av rörelsespårning och kommunikation över ett gemensamt nätverk vilket gör det möjligt för spelarna att interagera med varandra. Projektet utvecklas av ett arbetslag på sex personer och är en del av kursen TNM094 - Medietekniskt kandidatarbete, på Linköpings universitet.

Spelet kontrolleras med hjälp av handhållna enheter som agerar fönster in i en gemensam virtuell spelplan där två till sex spelare är uppdelade i två lag som tävlar mot varandra. Varje lag tilldelas sin egen planhalva och målet är att skicka myror mot sina motståndare för att äta upp deras sockerbit och samtidigt försvara sin egen.

Projektets arbete har planerats och utförts utifrån utvecklingsmetodiken Scrum. Arbetslaget har varit indelade i olika roller såsom Scrum-mästare, produktägare och utvecklare. Vidare har arbetet skett i sprintar på två veckor där uppgifter baserade på projektets krav och behov utfördes. Trello har använts för att föra en backlog till sprintarna samt annan dokumentation, som mötesprotokoll, har förts under utvecklingsprocessen.

Projektet har utvecklats i spelmotorn Unity med programspråket C#. Vidare har Unitys inbyggda nätverksarkitektur använts för nätverkskommunikation. För implementation av rörelsespårning användes *Förstärkt verklighets*-motorn Vuforia.

De slutsatser som kan dras efter detta arbete är att det är möjligt för mobila enheter att agera som vyfönster med hjälp av *Förstärkt verklighet*. Genom att tillämpa en klient/server arkitektur som nätverkslösning sätter det krav på användarna att vara uppkopplade mot samma nätverk. Detta ska bidra till att användarna samlas för att spela spelet och samarbeta för att vinna.

Innehåll

Sammanfattning	i
Figurer	iv
Tabeller	vi
1 Inledning	1
1.1 Bakgrund	1
1.2 Syfte	1
1.3 Frågeställning	1
1.4 Kundkrav	2
1.5 Avgränsningar	2
1.6 Grundläggande spelidé	3
1.7 Typografiska konventioner	4
2 Förstudier och relaterade produkter	5
2.1 Rörelsespårning	5
2.1.1 Accelerometer/Gyroskop	5
2.1.2 Förstärkt verklighet	5
2.2 Nätverk	7
2.3 Spellogik	9
2.3.1 Navigering och vägsökning	9
2.4 Relaterade produkter	10
2.4.1 Dota	10
2.4.2 Equilinox	10
2.4.3 Starcraft	11
3 Utvecklingsprocess	12
3.1 Scrum	12
3.1.1 Backlogg	13
3.1.2 Kundmöte	14

3.2 Tidsplan	14
3.3 Versionshantering	15
3.4 Dokumentationsprinciper	16
3.5 QA och testprinciper	16
4 Implementation av spelet	17
4.1 Tekniska verktyg	17
4.2 Systemets komponenter	17
4.2.1 Nätverk	17
4.2.2 Rörelsespårning	20
4.2.3 Spellogik	21
4.3 Modellering och Animering	24
4.4 Sammanställning av huvudkomponenter	24
5 Resultat	26
5.1 Användargränssnitt	26
5.2 Spelet	28
6 Analys och diskussion	30
6.1 Arbetsprocess	30
6.2 Resultat	30
6.2.1 Användargränssnitt	31
6.2.2 Spelet	31
6.3 Arbetet i ett vidare sammanhang	31
6.4 Källkritik	32
7 Slutsatser	33
7.1 Rörelsespårning	33
7.2 Nätverkshantering	33
7.3 Spelidé	34
7.4 Utvecklingsområden	34
Litteraturförteckning	35
A Personligt bidrag	37
B Användartester	39

Figurer

1.1	En skiss på hur spelet fungerar. De transparenta delarna symbolisera den virtuella spelplanen och är endast synliga genom de handhållna enheterna	3
2.1	Ett exempel på ett ArUco-bräde som bildmarkör, vars rutnät är 6×4 pixlar med ArUco-markörer av storlek 8×8 pixlar	6
2.2	Ett ArUco-bräde som implementerats som bildmarkör i Vuforia. De gula symbolerna markerar bildens unika drag	7
2.3	Klient/server nätverksmodell	8
2.4	Ett navigeringssystem från Unitys manual om <i>Navigation and Pathfinding</i>	9
2.5	Bilden är ett exempel på den standardiserade spelbanan för MOBA-spel som projektet baseras på	10
2.6	Equilinox använder sig utav Low Poly-modelleringsstil	11
3.1	En itererande rutin inom utvecklingsmetodiken scrum	13
3.2	Illustration av en del av backloggen som fördes av arbetslaget under arbetsprocessen där korten är arbetsuppgifter	13
3.3	Gantt-schema över sprintar, deadlines och milstolpar för projektet	14
4.1	Aktivitetsdiagram från startmeny till nätverkslobby	18
4.2	Två ArUco-markörer som skiljer i storlek. Den vänstra är 6×6 pixlar, medan den högra är 8×8 pixlar	21
4.3	Den slutgiltiga projektmarkören som är ett ArUco-bräde bestående av ett 6×4 -pixelrutnät av ArUco-markörer med storlek 8×8 pixlar	21
4.4	Första versionen av spellogiken med agenter som rör sig mot en destination i form av en blå kub	22
4.5	Basen i form av en sockerkub, med en mätare som visar basens hälsa implementerad	23
4.6	Antalet AntEggs visas med en siffra bredvid en ikon	23
4.7	Den 3D-modellerade myran från ovan med sitt animeringsskelett	24
5.1	Välkomstsidan som är det första användaren möts utav	26
5.2	Hjälpsidan som fungerar som guide för en användare som spelar för första gången . .	27
5.3	Lobbyn (War Zone) visar alla spelare samt vilket lag de tillhör	27
5.4	Myrorna skapas i myrstacksliknande strukturer och rör sig sedan mot motståndarens sida	28

5.5	Vid spelets slut ser användaren vilket lag som vunnit och får möjligheten att återvända till lobbyn	28
5.6	Den sluttgiltiga spelplanen med två spelare som skickar myror mot varandra	29
5.7	Hur sällskapsspelet ser ut när två användare spelar	29

Tabeller

1.1 Tekniska termers redogörelse	4
--	---

Kapitel 1

Inledning

Ant Warz är ett digitalt sällskapsspel anpassat för handhållna enheter som med hjälp av *Förstärkt verklighet* (AR) bygger upp en gemensam spelplan. Genom att fysiskt flytta på den mobila enheten kan spelaren förflytta sig i spelvärlden och se olika delar av spelplanen. Spelet utvecklas i spelmotorn *Unity* i kombination med AR-motorn *Vuforia*. Denna rapport redogör utvecklingen av spelet, vilka resurser och metoder som användes, samt en diskussion kring det slutgiltiga resultatet.

1.1 Bakgrund

Spelmarknaden har vuxit sig allt större de senaste åren och framförallt har det skett stora framsteg på den mobila sidan, vilket nu står för över hälften av den globala spelmarknaden enligt en undersökning av NewZoo [1]. Detta är inte förvånande då mobila enheter har blivit kraftfulla till den nivå att de är kapabla till att hantera tekniker som *Förstärkt verklighet* och *Virtuell verklighet*. Dessa är tekniker som låter användare interagera med en miljö med rörelse och se virtuellt konstruerade verkligheter med en mobil enhet eller någon form av *Head-mounted display*.

Med Förstärkt- och virtuell verklighet har gamla spelkoncept reformerats och nya tagits fram. Sällskapsspel är en spelform som har funnits sedan länge och kännetecknar ett fysiskt spel som kräver att spelarna samlas och umgås. Exempel på detta är brädspel och kortspel. Spelformen omfattar generellt inte digitala spel då dessa sällan kräver att personerna samlas, vilket gör att den typen av socialt umgänge försätts in. Projektet avser att försöka kombinera den typen av social aspekt med ett digitalt medium och på så vis expandera sällskapsspelkonceptet.

1.2 Syfte

Projektet ska resultera i ett digitalt sällskapsspel för handhållna enheter, vilket i detta fall omfattar smarta mobiltelefoner och surfplattor. Enheterna ska agera som spelarens vyfält, i en gemensam virtuell spelplan, där de endast kan se en begränsad del av spelplanen. För att sedan se andra delar av spelplanen måste spelaren förflytta sin enhet. Rapportens syfte är att undersöka relevanta tekniker för att ta fram ett spel utefter den angivna beskrivningen samt redovisa arbetet som har genomförts.

1.3 Frågeställning

Projektet avser att undersöka följande frågeställningar som sedan besvaras i projektrapporten.

- Vilka relevanta tekniker finns det för rörelsespårning av handhållna enheter och är någon av dessa mer lämpad vid utvecklingen av ett digitalt sällskapsspel för flera användare?
- För att användare ska kunna spela mot varandra behöver de vara uppkopplade mot ett nätverk. Hur bör en sådan nätverksarkitektur se ut och vilken information behöver skickas över nätverket när spelet är av en RTS/MOBA-genre?
- Hur kan en lagbaserad spelidé utvecklas för att främja ett socialt och interaktivt digitalt sällskapsspel för flera användare?

1.4 Kundkrav

Vid projektstart satte kunden ett antal krav på systemet som ges nedan.

1. Systemet skall kunna hantera flera användare.
2. Systemet skall kunna förstås och spelas av en 12-åring och högre.
3. Systemet skall använda sig utav enheter som endast visar en del av spelplanen.
4. Systemet skall kunna monteras upp och startas på mycket kort tid.
5. Systemet skall vara underhållande och engagerande.
6. Systemet skall vara responsivt och inte ha några märkbara fördröjningar i varken rörelsespårning, spelupplevelse eller nätverk.

1.5 Avgränsningar

Arbetets fokus låg på att skapa ett sällskapsspel i underhållande syfte och inte vara en produkt för professionellt bruk. Detta medförde att ingen utveckling skedde mot nätverkssäkerhet eller andra anti-fusk relaterande åtgärder, då fusk förekommer även i traditionella sällskapsspel. För att inkorporera den sociala aspekten begränsas spelare till att vara uppkopplade mot samma nätverk för att spela mot varandra.

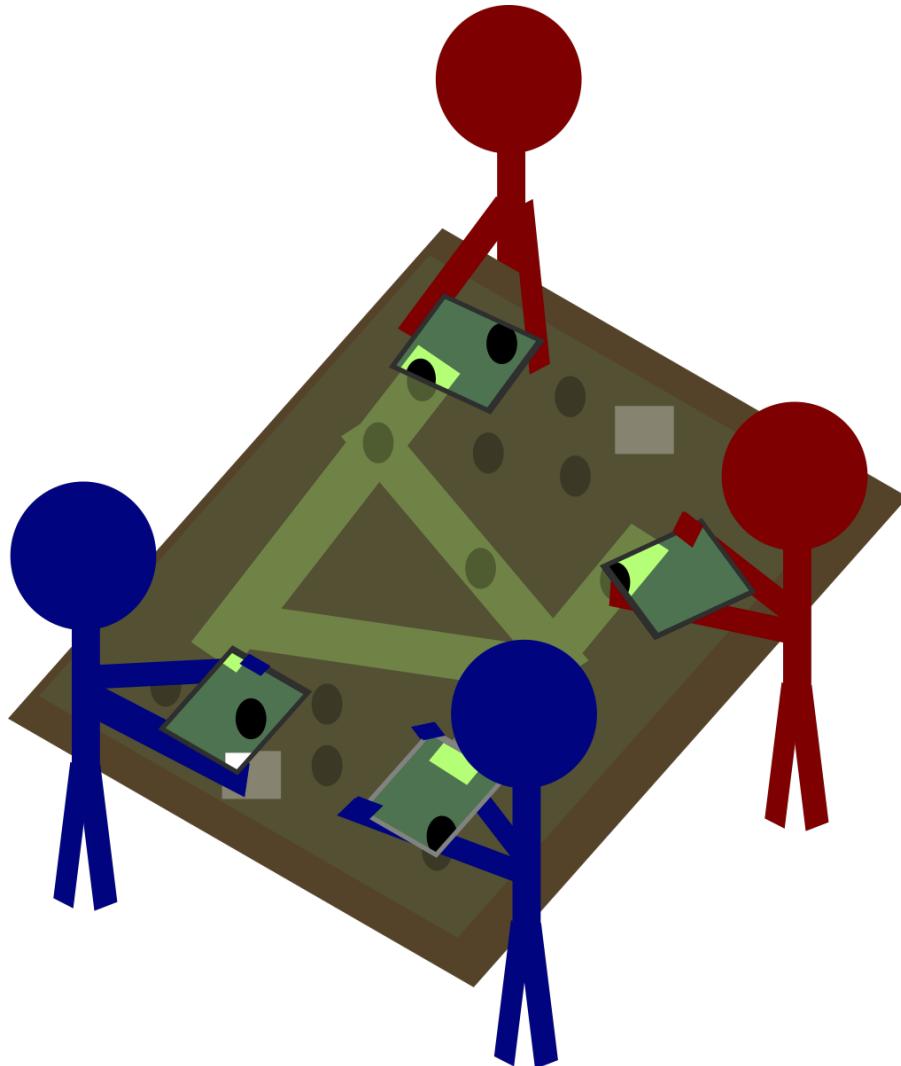
Tidigt i utvecklingsprocessen angav kunden inte några krav på att systemet skulle fungera på flera plattformer. Arbetslaget valde därför att avgränsa systemet till att stödja enheter med Android som operativsystem. Då spelmotorn Unity användes för att utveckla produkten medförde det även följande systemkrav för Android-enheterna [2]:

- Operativsystem: Android 4.1 eller senare
- Processor: ARMv7 CPU med NEON support eller Atom CPU
- Stöd för OpenGL ES 2.0 eller senare

1.6 Grundläggande spelidé

Grundidén för spelet är att två till sex spelare delar in sig två lag och spelar mot varandra. En enhet agerar servervärd och skapar ett rum genom att skicka en förfrågan till klientservern och resterande spelare kopplar upp sig mot enheten. Detta förutsätter att alla spelare är uppkopplade mot samma nätverk. När serverenheten skapar rummet initieras den virtuella spelvärlden vid en position som är definierad av ett fysiskt objekt i form av en markör. De andra spelarna måste därefter rikta sina enheter mot markören för att se spelplanen. Serverenheten kan efter skapandet av spelplanen röra sig fritt runt markören då spelplanen är förankrad på markörens position.

Innan spelet startar ska lagen ställa sig på sin halva av spelplanen. Båda lagen har varsin sockerbit som de måste försvara. Mellan dessa sockerbitar finns ett skogsområde som ens lags myror kommer att passera över. Målet med spelet är att skicka myror till sina fienders planhalva så att deras sockerbit till slut äts upp av myrorna. Båda lag startar med resurser men dessa tar slut snabbt efter att myror skapats. Det är möjligt att få fler resurser genom att krossa motståndarnas myror när de är inom synhåll. Detta resulterar även i att lagen försvarar sin egna sockerbit. Spelet avslutas när det ena lagets sockerbit inte har några liv kvar.



Figur 1.1: En skiss på hur spelet fungerar. De transparenta delarna symbolisera den virtuella spelplanen och är endast synliga genom de handhållna enheterna

1.7 Typografiska konventioner

De tekniska termer som används i rapporten skrivs i kursiv text och redogörs i Tabell 1.7.

Tabell 1.1: Tekniska termers redogörelse

<i>Sprint</i>	Projektarbetet delas in i planerade etapper, så kallade <i>sprints</i> som har specifika mål och krav att uppfylla inför nästkommande milstolpe. Sprintar är en del av den agila utvecklingsmetoden <i>Scrum</i> .
<i>Scrum-mästare</i>	Scrum-mästaren leder projektgruppen och har närmast kontakt med kund och produktägare. Denna person leder även möten och ser till att arbetet fortskrider.
<i>Kund</i>	Kunden är den person eller företag som beställt en produkt av utvecklingsteamet och som sätter de initiala krav som produkten ska ha.
<i>Produktägare</i>	Produktägaren är en person i projektgruppen som har speciellt intresse av produkten i sin helhet och som ser till att kundens krav uppfylls.
<i>Daily scrums</i>	Varje morgon träffas Scrum-mästare och utvecklingsteamet för korta och concisa möten, så kallade <i>Daily scrums</i> , där arbetet och eventuella problem diskuteras.
<i>Backlog</i>	En lista med uppgifter som ska utföras av projektgruppen för att komma vidare i arbetet. Uppdateras efter varje sprint och kundmöte.
<i>Gren</i>	För versionshantering används olika grenar för att undvika kodkonflikter under arbetetsgång. <i>GitHub</i> används som tekniskt verktyg här.
<i>UML</i>	Vid den arkitekturella modelleringen av programdesignen används <i>UML</i> (<i>Unified modeling language</i>) som är ett diagram av systemets komponenter.
<i>IDE</i>	Vid programmering av systemets komponenter använder utvecklingsteamet sig av olika <i>IDE</i> (<i>Integrated Development Environment</i>) som kompilar koden och till exempel assisterar vid refaktorering.
<i>Listener</i>	En <i>listener</i> är en funktion som avfyras vid en specifik händelse. I detta projektet används detta för att registrera knapptryck.
<i>RPC</i>	<i>RPC</i> (<i>Remote Procedure Calls</i>) beskriver sättet som nätverksservern kommunikerar med klienten.
<i>Baka</i>	För att myrorna ska kunna navigera på ett korrekt sätt över spelplanen och undvika objekt <i>bakas</i> spelplanens <i>NavMesh</i> innan körning.
<i>API</i>	<i>API</i> (<i>Application Programming Interface</i>) eller på svenska Applikationsprogrammeringsgränssnitt, är en specifikation av hur olika applikationsprogram kan använda och kommunicera med en specifik programvara.

Kapitel 2

Förstudier och relaterade produkter

Innan uppstarten av arbetet lade arbetslaget tid åt sidan att genomföra förstudier samt hämta inspiration från relaterande produkter. Förstudier gjordes då samtliga i arbetslaget hade liten till ingen erfarenhet inom de tekniska områden som skulle tillämpas i projektet. Med mer kunskap inom områdena kunde arbetslaget enklare jämföra olika tekniker för att förenkla arbetsprocessen och förbättra resultatet. Inspiration från tidigare arbeten gav arbetslaget många olika idéer på spelkoncept som kunde appliceras i detta projekt, även om ett flertal inte kom med i den slutgiltiga produkten.

2.1 Rörelsespårning

För att enheterna ska kunna flytta sig på spelplanen med en acceptabel precision finns det två lösningar att undersöka. Dels spårning genom att använda enheternas kamera (*Förstärkt verklighet*) eller genom att utnyttja enheternas inbyggda accelerometrar och/eller gyroskop. Med acceptabel precision menas att systemets responsivitet är så pass god att användaren inte uppfattar markanta störningar som påverkar spelupplevelsen negativt.

2.1.1 Accelerometer/Gyroskop

De flesta handhållna enheter har flera olika sensorer där accelerometer och gyroskop är de sensorer som registerar positionsförändringar samt lateral orientering. En accelerometer mäter enhetens linjära acceleration och ett gyroskop mäter vinkelhastigheten under rotering. Med denna information beräknas enhetens position, i förhållande till dess initiala startposition. Om endast acceleratorn används kommer bara riktningshastigheten på enheten att mäts men inte dess lutning eller laterala orientering. Detta går att lösa genom att kombinera acceleratorns sensordata tillsammans med informationen från ett gyroskop. Den kombinerade informationen från sensorerna kan vara tillräckligt precis för vissa typer av applikationer men för detta projekt bör andra tekniker utforskas.

2.1.2 Förstärkt verklighet

Förstärkt verklighet (AR), eller förstärkt verklighet, är en interaktiv upplevelse av en verklig miljö där föremål som finns i verkligheten förstärks av datorgenererad information över flera sinnesintryck. Enligt en studie inom AR av Joseph Rampolla och Gregory Kipper [3] definieras AR utifrån dessa aspekter:

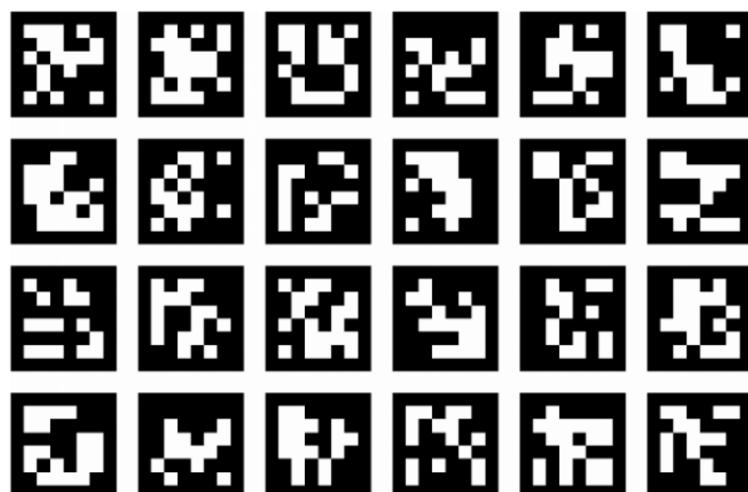
- AR kombinerar verklig och virtuell information i en gemensam miljö.

- AR körs interaktivt och i realtid.
- AR registrerar (justerar) reella och virtuella objekt i tre dimensioner.

Vid tillämpning av *mobil AR* [4] används enhetens kamera för att ta in information om omgivningen. Datan översätts och låter sedan skärmen fungera som ett fönster där element från den virtuella och den verkliga världen har förts samman. För att lokalisera enhetens position i förhållande till de virtuella objekten används oftast en eller flera markörer. Dessa markörer är QR-koder eller bilder med hög kontrast, för att urskilja sig från varandra. Detta för att inte skapa förvirring hos enheten som då inte tar in samma information från flera håll. Orienteringen av enheten i förhållande till markörerna möjliggör att den virtuella världen kan ses från olika perspektiv. Detta ställer även högre krav på enheterna då kameran alltid måste hållas mot en av dessa markörer för att konstant hålla spelplanens plats synkroniserad med den riktiga världen och vad kameran ser. Här finns det flera alternativ i formen av *API (Application Program Interface)* till Unity för att implementera AR så som ARCore (Google), ARKit (Apple), ARFoundation och Vuforia. I detta projekt valdes det att fokusera på ArUco och Vuforia närmare. Motiveringen till detta var för att ArUco rekommenderades till projektgruppen vid en teknisk konsultation för sin precision vid kamerakalibrering. Vid undersökning av Vuforia drogs slutsatsen att det fanns en god integration med spelmotorn Unity.

ArUco

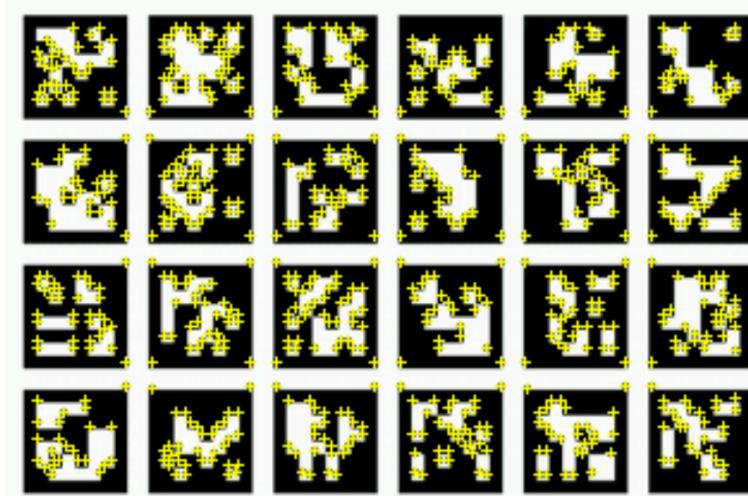
Ett alternativ till att använda ett API som nämnts i kapitel 2.1.2 är att implementera kamerakalibrering och punktdetektering med hjälp av biblioteket *ArUco* som är skrivet i C++ och OpenCV. ArUco detekterar så kallade *ArUco-markörer*, där en sådan är en svart fyrkant med mindre vita fyrkanter inuti sig för att lätt kunna urskilja kontrast. Varje ArUco-markör har ett individuellt ID kopplat till sig och placeras ut på ytan det virtuella objektet ska projicera på. Ett rutnät av ArUco-markörer kan skapas och placeras på ett bord. Då behöver kameran endast se en viss del av dem för att veta exakt hur den ska projicera spelplanen. Ett sådant rutnät kallas för ett *ArUco-bräde* och ett exempel av en sådan framstår i Figur 2.1.



Figur 2.1: Ett exempel på ett ArUco-bräde som bildmarkör, vars rutnät är 6×4 pixlar med ArUco-markörer av storlek 8×8 pixlar

Vuforia

Vuforia är ett *Software Development Kit* (SDK) för AR till mobila enheter och precis som ArUco använder Vuforia bildanalys för bildigenkänning av detektionsmarkör. Båda verktygen är förhållandevis lika men det finns några distinkta skillnader. Vuforia förser API i ett flertal programmeringsspråk, vilket inkluderar .NET med en extension till Unity. Detta medför att deras SDK stödjer utveckling till både Andriod och iOS. Detektionsmarkörerna som används med Vuforia är inte låsta till ett speciellt utseende utan ger friheten att ha valfri bild. Bilden som ska användas som detektionsmarkör laddas upp på Vuforias hemsida där ett konto är kopplat till kameran inuti Unity via en licensnyckel. När en bild kopplas till Vuforia får den ett betyg huruvida den uppfyller spårningskraven [5]. Ett högre betyg betyder att rörelsespårningen är mer precis och därmed mer lämplig att använda. Ett exempel på en bild som fått högsta betyg visas i Figur 2.2. I figuren framstår även de unika drag och mönster som bilden besitter i form av gula symboler.



Figur 2.2: Ett ArUco-bräde som implementerats som bildmarkör i Vuforia. De gula symbolerna markerar bildens unika drag

Då detektionsmarkören inte längre kan spåras av enhetens kamera aktiveras Vuforias inbyggda funktionExtended tracking som innebär att spårningen övergår till att baseras på sensordata. Informationen samlas in av både accelerometern och gyroskopet nämnt i avsnitt 2.1.1 vilket möjliggör fortsatt spårning, dock med försvagad precision.

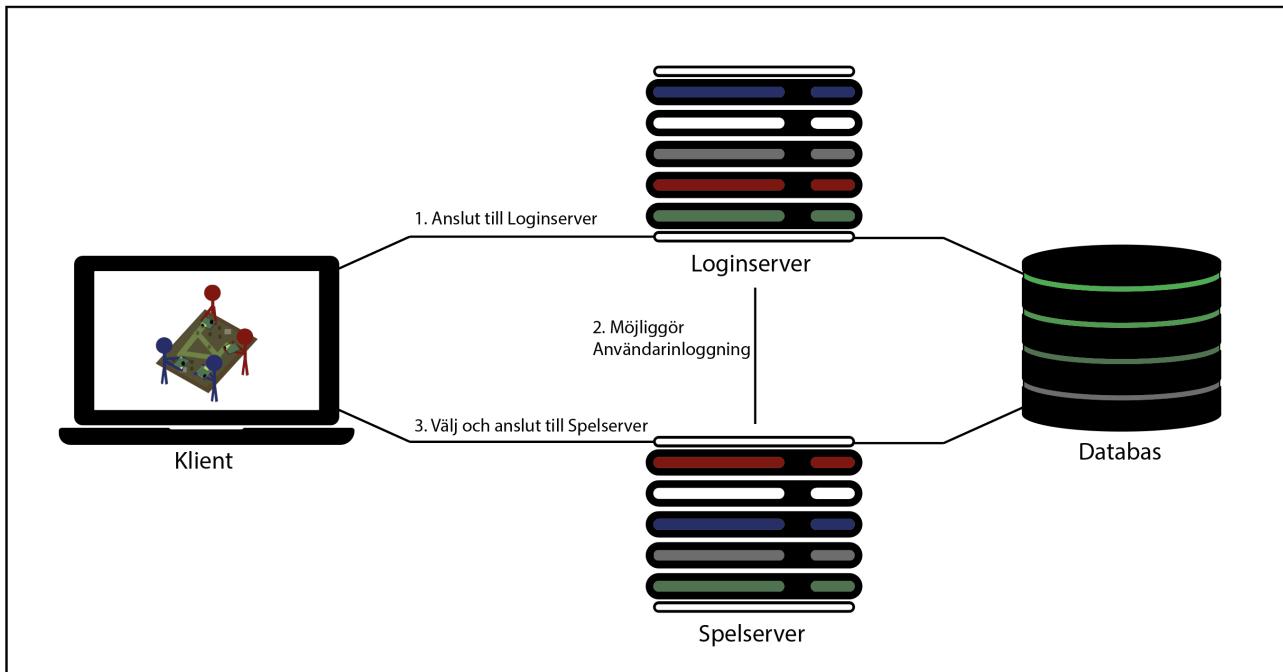
2.2 Nätverk

För att systemet ska uppfylla kraven enligt 1.4 behövs en nätverksmodell för att kunna hantera flera användare och skicka/ta emot data i realtid. Det finns två huvudmodeller kring detta område:

- Klient/Server nätverk (Client/Server networks)
- Icke-hierarkiska nätverk (Peer to peer networks)

De mest centrala skillnaderna är att icke-hierarkiska nätverk inte har en centraliserad server/databas utan varje enhet delar information mellan varandra. Det finns alltså ingen part som innehavar information som andra enheter hämtar ifrån. I en klient/server arkitektur finns det separata dedikerade servrar och klienter på nätverket där användare kan hämta information som är lagrad i databasen. Servern kan

begränsa information för vissa kunder i en hierarkisk uppbyggnad. Ett exempel på en klient/server arkitektur visas i Figur 2.3.



Figur 2.3: Klient/server nätverksmodell

I ett icke-hierariska nätverk finns nästan ingen nätverkssäkerhet vilket skulle äventyra användare i framtiden om spelet kommersialiseras. Att distribuera ett spel bland användare gör att kontrollen över spelet blir mer komplext [6]. Dessa typer av nätverk är inte heller byggda för att klara av många kunder på en och samma gång medan ett klient/server nätverk kan utvecklas och stödja miljoner av användare med bra säkerhetsprotokoll [7].

Med projektets krav och ändamål blir valet av nätverksarkitektur trivialt. Ett klient/server nätverk lämpar sig bäst för systemet i fråga och är också det som kommer att undersökas. I denna projektrapport kommer två lösningar vid implementering av ett sådant nätverk undersökas.

- Unitys *Unet*
- Egen implementering av server i Java

Båda dessa lösningar kommer att undersökas i aspekter kring systemets krav och användbarhet. Mer avancerade nätverkplugins för Unity så som *DarkRift*, *Forge* och *Photon* kommer ej att undersökas då de två givna lösningarna ovan satisfierar systemets krav utan problem.

Unet

Spelmotorn *Unity* har en fleranvändarlösning vid namn *Unet* som kan användas för systemets ändamål. Unet använder sig utav ett högpresterande transportlager som baseras på nätverksprotokollet *UDP (User Datagram Protocol)* och är optimalt för spelsystem. Vidare använder det ett *LLAPI (Low-Level Application Programming Interface)* och ett *HLAPI (High-Level Application Programming Interface)*. En av funktionerna i HLAPI är att få möjligheten att som klient agera servervärd samtidigt som kunden själv är en spelare, vilket är förmånligt för spelets ändamål.

Egen implementation av server i Java

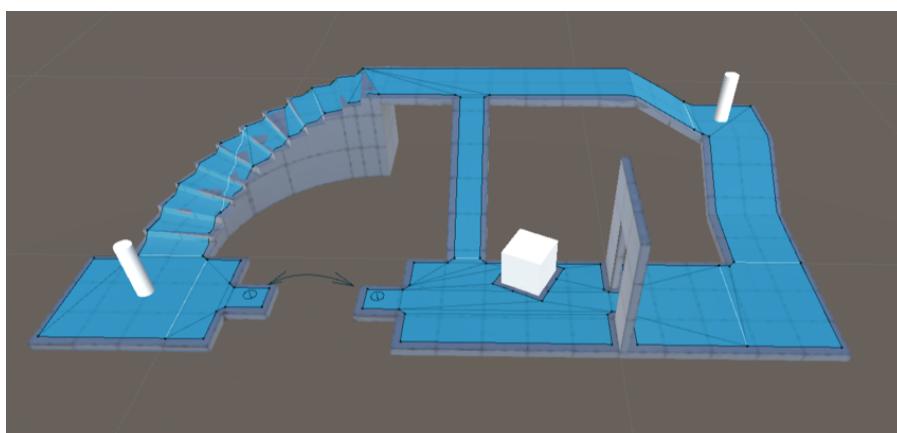
Fördelen med att programmera och bygga upp en server med ett programspråk så som Java är flexibiliteten. Här finns möjligheten att bygga en nätverkstopologi som Unity till exempel inte har bra stöd för. Det är även möjligt att producera en server som har bättre CPU och minnesanvändning. Programspråket som skrivs i Java körs av en *JVM* (*Java Virtual Machine*), en programvara som kompilerar koden så att den tolkas av alla maskiner som har JVM installerat. Klasser och annan information kan alltså överföras mellan flera plattformar och operativsystem utan problem vilket innebär att Java lämpar sig som programspråk för internetapplikeringar [9]. En annan fördel med en egen implementerad server framför UNet är att det sistnämnda kommer att fasas ut från och med Q4 2018 [10]. Därmed kommer både LLAPI och HLAPI att sluta fungera efter detta och systemet kommer att behöva uppdateras följaktligen.

2.3 Spellogik

I följande avsnitt beskrivs det förarbete och studier som har gjorts kopplat till systemets spellogik.

2.3.1 Navigering och vägsökning

För att skapa ett navigeringssystem för karaktärer i Unity kan Unitys egna navigations- och sökvägsystem användas [11]. Navigeringssystemet gör det möjligt att skapa spelobjekt som rör sig genom spelvärlden genom att skapa en *NavMesh* (*Navigation Mesh*). En NavMesh är en datastruktur som beskriver vilka delar av spelplanen som ett spelobjekt kan röra sig på samt hur spelobjekten ska bete sig vid olika områden av spelplanen. Processen för att skapa en NavMesh kallas för *NavMesh Baking* och går ut på att rendera spelplanen och dess objekt, för samla in data om spelplanens struktur. Med en NavMesh skapad kan sedan *NavMesh Agents* användas för att få spelobjekten att bete sig utifrån den renderade spelplanens struktur och geometri. En NavMesh Agent är en komponent till spelobjektet med flera variabler som beskriver hur objektet ska uppföra sig. Till exempel kan en variabel för hastighet ändras för att påverka hur snabbt som objektet ska kunna röra sig med. För att få en NavMesh Agent att röra sig mot en specifik destination sätts ett mål ut i spelplanen som agenten sedan använder för att beräkna kortaste möjliga väg. En NavMesh Agent förflyttar sig sedan mot målet utifrån den gjorda vägberäkningen. Vid uträkningen av en NavMesh är det också möjligt att beskriva olika typer av hinder i spelplanen, som en NavMesh Agent måste undvika. Ett exempel på en uträknad NavMesh med två NavMesh Agents och ett utsatt hinder visas i Figur 2.4.



Figur 2.4: Ett navigeringssystem från Unitys manual om *Navigation and Pathfinding*

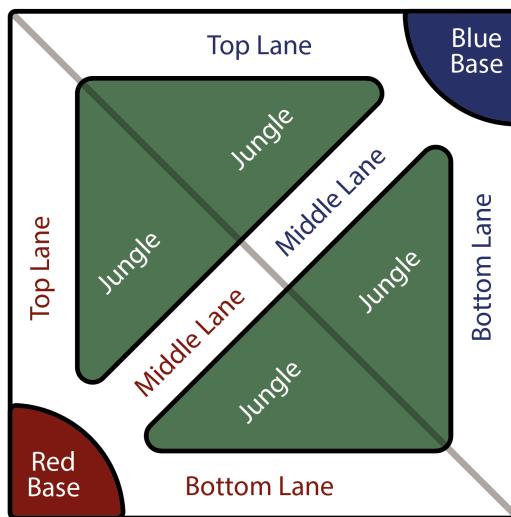
Alternativet till NavMesh skulle vara att använda *Spline-kurvor* som baseras på en matematisk funktion för att skapa en kontinuerlig kurva för spelobjekten att följa [12]. Detta skulle innebära att ytterligare studier om kollisionshantering måste undersökas.

2.4 Relaterade produkter

Vid skapandet av ett fleranvändarspel är det nyttigt att ta lärdom och inspiration från andra framgångsrika spel och spelformer. Detta för att vid utveckling av projektgruppens system skapa goda förutsättningar.

2.4.1 Dota

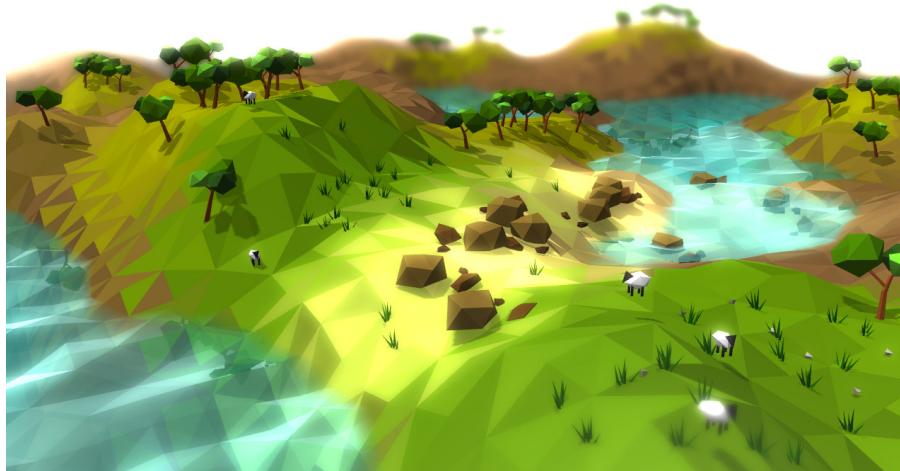
Dota är ett spel som till en början var en modifikation av spelet Warcraft III och populariserade spelgenren *Action Real-Time Strategy* (ARTS) och *Multiplayer Online Battle Arena* (MOBA) [13], som båda är undergrenar till genren RTS. Spelet är lagbaserat och utspelar sig på en symmetrisk bana liknande den som kan ses i Figur 2.5. Genom att kontrollera sin karaktär kan de genererade resurserna utnyttjas (arméer och inkomst) i spelet för att få ett övertag över motståndaren. Spelet avslutas när ett av lagen har lyckats förstöra sin motståndares bas [14].



Figur 2.5: Bilden är ett exempel på den standardiserade spelbanan för MOBA-spel som projektet baseras på

2.4.2 Equilinox

Equilinox är ett avkopplande naturspel som låter spelaren skapa och uppfostra sitt helt egna varierande ekosystem av växter och djur. Spelet är utvecklat i Java och OpenGL av *ThinMatrix*, en självständig spelutvecklare. Detta spel skiljer sig från det tänkta spelkonceptet men dess artistiska aspekt, som följer en Low Poly-modelleringss stil, är intressant [15]. Equilinox Low Poly-stil visas i figur 2.6 nedan.



Figur 2.6: Equinox använder sig utav Low Poly-modelleringss stil

2.4.3 Starcraft

Starcraft av *Blizzard Entertainment* är ett realtidsstrategi-spel (RTS-spel) där spelet går ut på att bygga upp en bas, samla resurser och skapa arméer. Vinst sker genom att strida mot olika fiender och besegra dem. Spelet utspelar sig i en fiktiv värld med en rik historia bakom varje spelbar ras för att satisfiera även de mest passionerade spelarna. Precis som andra RTS-spel sker detta spel i realtid och strategin av spelet bygger främst på hur användaren prioriterar sina resurser och tar vara på tiden [16].

Kapitel 3

Utvecklingsprocess

Genom att tillämpa en definierad utvecklingsmetodik är det möjligt att effektivisera arbetslaget genom att samtliga följer bestämda standarder och rutiner. Ett krav som arbetslaget hade inför projektet var att utvecklingsmetodiken skulle vara agil. Agil utvecklingsmetodik följer filosofin att arbeta ska ske inkrementellt och iterativt. Detta betyder att delleveranser av funktionaliteter sker regelbundet utefter ett förbestämt schema och att planer och metoder utvärderas och förbättras löpande under utvecklingsprocessen. Den agila utvecklingsmetodiken som tillämpades av arbetslaget var Scrum.

3.1 Scrum

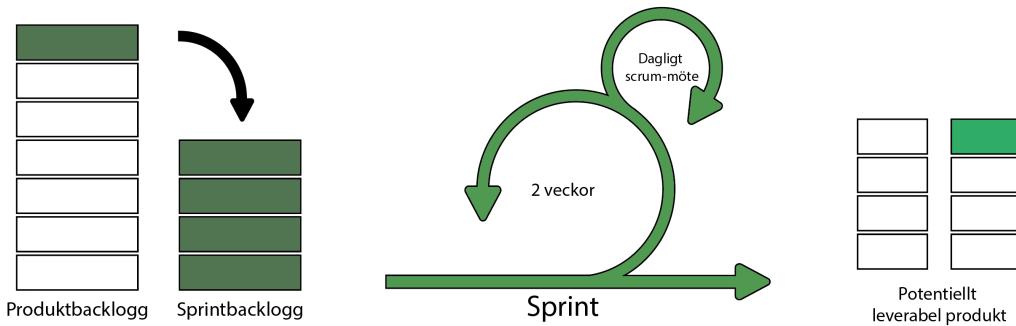
Scrum består utav sprintar, vilket är en arbetsperiod som sträcker sig mellan 14 och 30 dagar [17]. I detta projekt valde arbetslaget att arbeta i sprintar å två veckor. En sprint inleddes med en planeringsfas och avslutades med en demonstration och genomgång för kunden av vad som utvecklades under sprinten. Mer detaljerad beskrivning av vad som gjordes under sprintarna kan ses längre fram i kapitel 3.2. I planeringsfasen definierades ett antal arbetsuppdrag som skulle genomföras under sprinten utefter de krav som ställdes på den slutgiltiga produkten från kunden. Dessa arbetsuppgifter och krav fördes in i en *backlogg*. Under sprintarna skedde dagliga Scrum-möten som leddes av Scrum-mästaren, där arbetslaget samlades och diskuterade den kommande arbetsdagen. Vid demonstration och genomgång presenterade vardera grupp det dem hade genomfört under sprinten för de andra, som fick ställa frågor och komma med feedback. Vid denna fas tog även Produktägaren beslutet om produkten är något som kan publiceras vid den stunden eller om fler sprintar behövs. Illustration av detta kan ses i Figur 3.1. För att effektivisera utvecklingsprocessen utvecklades systemets komponenter parallellt genom att arbetslaget delade upp sig i mindre grupper. Detta innebar att två personer arbetade med att ta fram en lösning för rörelsespårning och implementerade den, två personer som ansvarade för spelutvecklingen samt två person som skapade de grafiska elementen i spelet. Gruppindelningen baserades på tidigare erfarenheter och intresse. Under projektets gång så var samtliga i arbetslaget flexibla och om en grupp upplevde att arbetsbelastningen blev för hög kunde en utvecklare från en annan grupp kliva in och hjälpa till.

Vid användning av Scrum delade arbetslaget även upp olika ansvarsområden till samtliga gruppmedlemmar. Till exempel tilldelades en person till Scrum-mästare, en till dokumentationsansvarig och en till produktägare. Detta resulterade i att arbetsfördelningen kunde lättare fördelas inom arbetslaget samtidigt som rutiner kunde hållas.

Gruppmedlemmarna delades in i följande roller:

- Scrum-mästare, hens roll var att agera ledare över arbetslaget och leda de möten som hölls under utvecklingsprocessen.

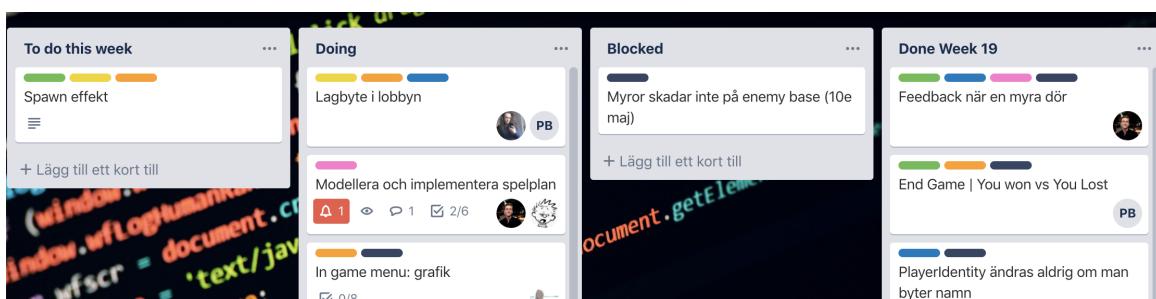
- Produktägare, hens roll var att föra in arbetsuppgifter och krav i backloggen under sprintplaneringen och se ordna den i en prioriteringsordning.
- Dokumentationsansvarig, hens roll var att agera sekreterare under sprintmöten och såg till att backloggen var uppdaterad tillsammans med produktägaren.
- Kontaktansvarig, hens roll var att uppdatera och boka möte med kund tillsammans med andra intressenter.
- Testansvarig, hens roll var att samla in feedback från externa testpersoner som tester de olika prototyper som hölls under utvecklingsprocessen och redogöra resultatet till arbetslaget.



Figur 3.1: En itererande rutin inom utvecklingsmetodiken scrum

3.1.1 Backlogg

Kravhantering och hantering av arbetsuppgifter fördes in en backlogg. Backloggen skapades i början av utvecklingsprocessen och uppdaterades av Produktägaren under varje planeringsfas. Produktägaren ordnade även innehållet i en prioriteringsordning utefter kraven som ställdes från kund. Under en sprint såg Dokumentationsansvarige till att samtliga i arbetslaget uppdaterade backloggen, allt eftersom de slutförde arbetsuppdrag. Vid behov ansvarade Produktägaren för att förändra prioriteringen av arbetsuppgifter under en sprint, efter diskussion med berörd arbetsgrupp. För hantering av en backlogg använde arbetslaget *Trello*. Trello är en digital Kanban-tavla där arbetsuppgifter lades till arbetsuppdrag på tavlan i form av kort. Dessa kort innehöll en kort beskrivning av funktionen som skulle implementeras. Korten delades in i spalter beroende på vilket stadie uppgiften var i, exempelvis om uppgiften ska påbörjas eller om den har avslutats. Krav från kund placerades också på tavlan i separata spalter med korta användarberättelser som ger en generell beskrivning av vad användare skulle kunna göra med systemet.



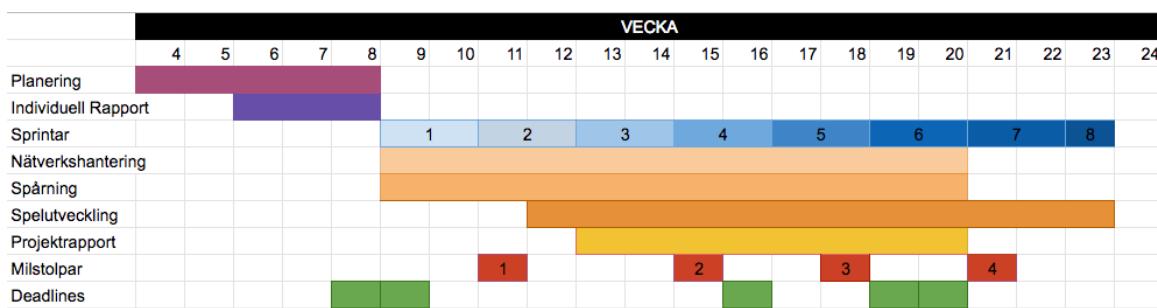
Figur 3.2: Illustration av en del av backloggen som fördes av arbetslaget under arbetsprocessen där korten är arbetsuppgifter

3.1.2 Kundmöte

Kundmöten skedde under utvecklingsprocessen där kunden lade fram krav och idéer och diskuterade dessa med arbetslaget. I tidigt skede av utvecklingsprocessen hade laget ett möte med kunden, som lade fram sin version av den färdiga produkten samt krav som produkten behövde uppnå. Under utvecklingsprocessen hade laget ytterligare två möten när Produktägaren ansåg att arbetslaget hade en prototyp som var värdig att visas upp. Detta för att ge kunden en förståelse för hur långt in i utvecklingsprocessen arbetslaget hade kommit. Utvecklarna ville även säkerställa att prototypen uppfyllde de krav som kunden hade och om det hade tillkommit nya krav.

3.2 Tidsplan

I början av projektet skapades en tidsplan som under projektets gång har reviderats allt eftersom arbetet utvecklats. Den slutgiltiga tidplanen kan ses i Figur 3.3.



Figur 3.3: Gantt-schema över sprintar, deadlines och milstolpar för projektet

Projektets uppstart

Projektet pågick under totalt 19 veckor varav de första fem veckorna tillägnades åt planering och arbete på individuella rapporter som beskrev projektplaneringen. Under planeringen gjordes förstudier och undersökningar med avseende på systemets krav och begränsningar. Planering av projektet påbörjades med givna mötesrutiner där projekthanteringsverktyg, så som Trello och Git, förbereddes och sattes upp. Innan sprint 1 skapades även en backlog så att alla i gruppen kunde tilldelas olika arbetsområden. Vidare beskrivs tidsplanen och arbetet under sprintarna utifrån projektets milstolpar.

Milstolpe 1: Enkel nätverksprototyp

Den första milstolen bestod av att ha en fungerande nätverksprototyp där flera spelare kunde vara aktiva och interagera med varandra över nätverket. Utifrån de förstudier som gjordes under planeringen kunde enkla prototyper för detta tas fram. Samtidigt med att en nätverksprototyp utvecklades studerades även olika verktyg för rörelsespårning. I början av sprint 1 delades projektgruppen in i mindre grupper där en grupp arbetade med nätverkshantering och den andra gruppen med spårning. Parprogrammering tillämpades där alla arbetade på en grundimplementering av systemet. Efter sprint 1 kunde spelarna koppla upp sig mot varandra samt skicka och ta emot information över nätverket.

Milstolpe 2: Fungerande spårning

Efter sprint 2-3 nåddes den andra milstolpen, fungerande spårning. Med fungerande spårning menas att enhetens kamera ska kunna spåra en markör och placera ett objekt på markörens position. Under sprintarna undersöktes de spårningstekniker som nämns i avsnitt 2.1 och genom att skapa enkla prototyper valdes Vuforia som det mest lämpade VR-verktyget för projektet. Motivering till valet kan läsas i Kapitel 4.2.2. Mot slutet av sprint 3 hade en prototyp för rörelsespårning utvecklats. Prototypen visade att enhetens fysiska position i förhållande till spelet förändrades vid rörelse utan större fel. Nätverksprototypen utökades med en förenklad spellogik och spårningsfunktionalitet började implementeras.

Milstolpe 3: Spårning och nätverk

Under sprint 4-5 delade projektgruppen upp sig ytterligare en gång i mindre grupper. Några började utveckla spellogiken och ta fram ett grafiskt koncept för spelet och resten fortsatte med integrering av nätverk och spårning. Mot slutet av dessa sprintar slutfördes en prototyp med god nätverks- och spårningsfunktionalitet som uppfyllde kundkrav givna i 1.4. Spellogiken fortsattes att utvecklas parallellt som tidigare där fler funktionaliteter implementerades, även här med utgångspunkt från kundkraven.

Milstolpe 4: Ett spel

Den sista milstopen var att ha ett färdigt spel med en tydlig början och slut. Detta mål nåddes efter sprint 6 då spellogik, spelgrafik, nätverk och spårning var implementerat och tillsammans utgjorde ett digitalt sällskapsspel för Android enheter. De sista sprintarna 7-8 är avsatta för förberedande av presentation och rapportoppresentation. Förbättringar och justeringar av spelet kommer också att ske under dessa sprintar i mån av tid.

3.3 Versionshantering

Genom att använda ett versionshanteringsverktyg kan arbetslaget på ett smidigt sätt hämta den senaste versionen av projektet, arbeta parallellt och gå tillbaka till en tidigare version om det skulle ske något fel. Detta effektiviseras utvecklingsprocessen då samtliga kan arbeta på projektet utan att störa någon annans arbete.

I detta projekt användes versionshanteringsverktyget *Git* tillsammans med webbhotelllet *GitHub*. För att förenkla användningen av GitHub skrevs en kort instruktionsmanual för hur eventuella problem skulle lösas som kunde uppstå under projektets gång. Verktyget tillät arbetslaget att dela upp arbetet i olika grenar så samtliga kunde arbeta på sin komponent, exempelvis nätverk eller spårning. En persons gren kunde sedan förgrenas ytterligare, om nya funktioner skulle implementeras. Detta arbetssätt förhindrade att konflikter uppstod under utvecklingsprocessen av funktionaliteter. Då spelmotorn Unity användes var det problematiskt i början att få GitHub att synkronisera med projektets Unity-filer. Exempel på detta var att få synkroniseringen att ske över olika operativsystem, då arbetslaget arbetade på både MacOS och Windows. Senare i projektet insåg arbetslaget att förgreningar inte behövdes då det räckte att arbeta i olika scener i Unity. Under de få tillfällen då konflikter skedde, samarbetade några i arbetslaget för att lösa dem.

3.4 Dokumentationsprinciper

Genom att föra dokumentation under utvecklingsprocessen fick arbetslaget en bra överblick över projektet, vilket hjälpte till att strukturera upp arbetet. Längre möten, exempelvis kundmöten och planeringsmöten, protokollfördes av Dokumentansvarige. Dessa protokoll, tillsammans med annan dokumentation lagrades på en gemensam *Google Drive* så att alla hade tillgång till dokumentationen [18].

Annan dokumentation skedde internt i kodfilerna. Detta skedde i form av kommentering av större funktioner och en kort beskrivning av filernas syfte i början av respektive fil. Detta förenklade kodgranskningsprocessen genom att ge de personer som inte skrivit koden en bra överblick över filerna.

3.5 QA och testprinciper

För att en slutprodukt ska erhålla en god kvalitet kräver det att kvalitetssäkringar sker kontinuerlig under utvecklingsprocessen.

För att försäkra att arbetslagets produkt levereras till kund med god kvalité avsattes tid i utvecklingsprocessen åt att granska varandras kod. Genom att kodgranska andra persons kod fick samtliga gruppmedlemmar en utökad förståelse för systemets komponenter. Chansen att hitta optimala lösningar ökade även då flera personer undersökte koden med olika angreppsvinklar. Detta satte även krav på utvecklarna att skriva lättläst kod och följa de dokumentationsprinciper som gruppmedlemmarna hade kommit överens om.

Användartester är en metod som används frekvent inom spelutveckling för att säkerställa att produkten som utvecklas tilltalar den avsedda målgruppen. Användartester skedde i det slutgiltiga skedet av utvecklingsprocessen. Dessa tester avsåg att se hur utomstående personer interagerade med systemet och hur pass bra återkopplingen fungerade. De personer som medverkade i testet fick navigera sig igenom spelets olika menyer, skapa en lobby och spela spelet tills att något lag vunnit. Efter testet fick testpersonerna lämna feedback till Testansvarige om eventuell avsaknad av spelfunktionalitet och återkoppling.

Den feedback som levererades till arbetslaget kan läsas i bilaga B och bidrog till vidare utveckling av systemet. All feedback kunde inte åtgärdas då tiden var begränsad.

Kapitel 4

Implementation av spelet

Spelet består utav tre huvudkomponenter (nätverk, rörelsespårning och spellogik) som alla implementerades kontinuerligt under projektets gång. För att kunna skapa ett spel krävs det noggrann planering tillsammans med flera olika verktyg så som spelmotorer, 3D-modelleringsprogram och bildbehandlingsprogram. I följande kapitel redogörs detaljerat hur implementationen av spelet har gått till.

4.1 Tekniska verktyg

Gruppen valde i detta projekt att utveckla systemet i spelmotorn Unity. Genom att använda en spelmotor som redan har verktyg och andra funktionaliteter inbyggt kunde arbetslaget fokusera på utvecklingen av systemets tre huvudkomponenter. Unity valdes över andra tillgängliga spelmotorer, exempelvis *Unreal Engine* och *Phaser*, då majoriteten av projektgruppen hade mer erfarenhet av Unity samt att spelmotorn kunde packetera android-applikationer. Unity har också en integrerad nätverkslösning UNet som användes av arbetslaget, då UNet uppfyllde de krav som getts från kund.

För implementering av tracking användes Vuforia då denna metod har en extension till Unity, vilket förenklade utvecklingsprocessen. Vidare användes Blender för modellering och animering av samtliga virtuella objekt i systemet. En detaljerad beskrivning och redogörelse av implementationen beskrivs vidare i kapitlet.

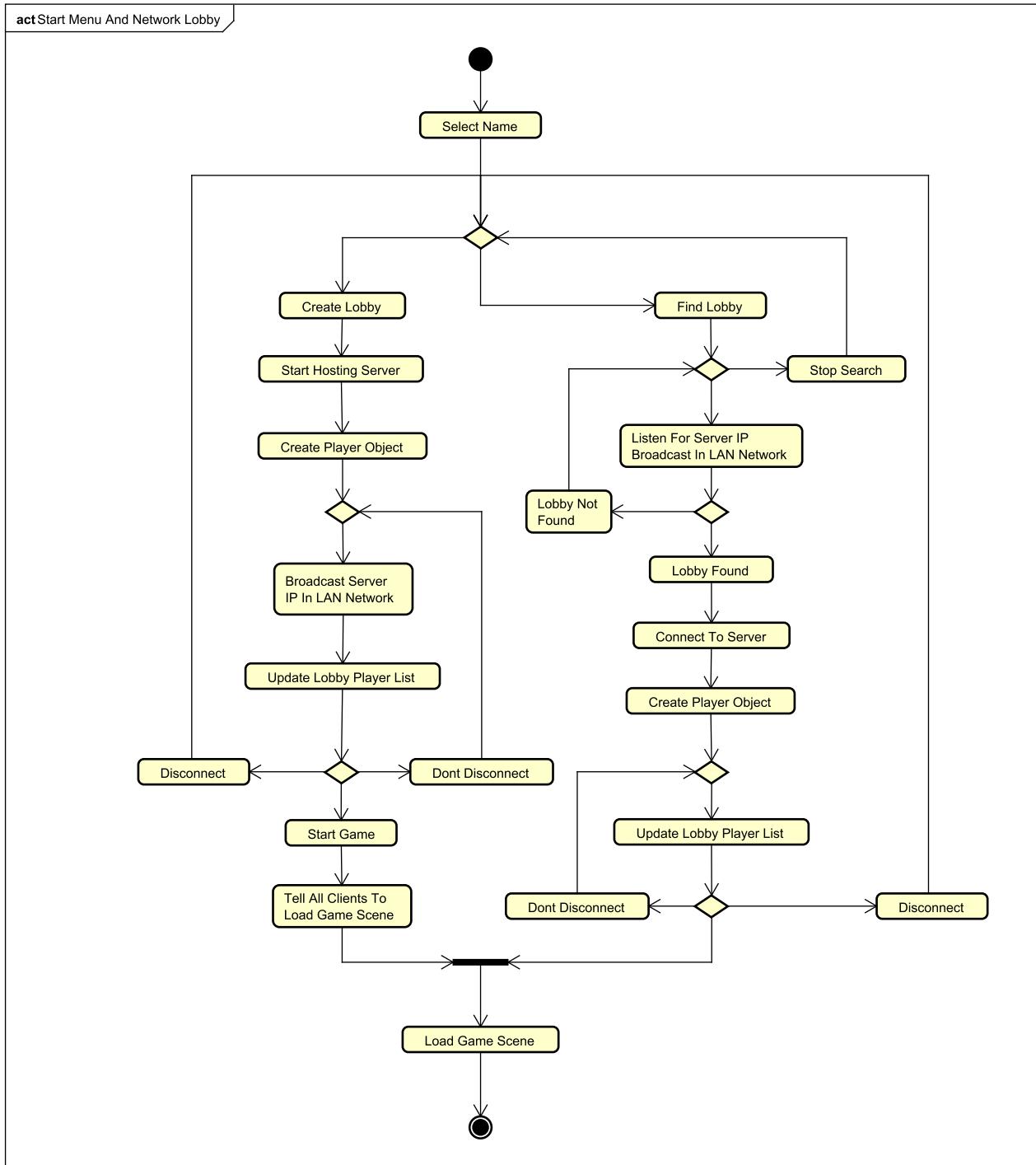
4.2 Systemets komponenter

Som tidigare nämnt består systemet av tre huvudkomponenter som utvecklades parallellt under projektets gång. Dessa komponenter utgjorde var för sig en viktig del för att spelet skulle uppfylla kraven för systemet som anges i avsnitt 1.4. I detta avsnitt beskrivs implementationen av nätverket, spellogiken och rörelsespårningen.

4.2.1 Nätverk

Arkitekturen för spelets startmeny och nätverkanslutning utvecklades efter aktivitetsdiagrammet i Figur 4.1. Som grund för nätverksarkitekturen användes Unitys egna nätverkslösning UNet, eftersom det finns integrerat i spelmotorn vilket var tidssparande för utvecklingsprocessen. UNet har ett *HLAPI* (se beskrivningen i avsnitt 2.2) som används för att bygga upp server, klienter, identiteter och spelbeende för nätverksspel i Unity [19]. Gränssnittet följer en serverauktoritär arkitektur som hanterar

vanliga nätverksuppgifter automatiskt och tillåter en klient att agera server (host). Serverauktoritet innebär att servern strikt styr över vilken information som ska skickas över nätverket.



Figur 4.1: Aktivitetsdiagram från startmeny till nätverkslobby

För att upprätta ett nätverkssystem användes komponenten *NetworkManager* som hanterar: nätverksanslutningar, skapandet av komponenter med nätverksauktoritet och spelets tillstånd. Implementationen stöder endast *Local Area Network* (LAN), det vill säga samtliga spelares handhållna enheter måste vara uppkopplade mot samma Wi-Fi. Förutom *NetworkManager* användes komponenterna *NetworkDiscovery* och *NetworkIdentity* som förklaras i avsnitten nedan.

Nätverksanslutningar

Upprättning av nätverket mellan flera användare kan grovt förklaras genom fyra steg:

1. En värd (host), som agerar klient och server parallellt, kallar på funktionen *StartHost* från sin lokala NetworkManager-instans. I detta skede skapas en speciell nätverksklient-komponent (som har både klienträttigheter och serverrättigheter) och en server med värdens lokala IP-adress.
2. Värden använder sig därefter av komponenten NetworkDiscovery för att utsända IP-adressen i det lokala nätverket via en signal. Detta sköts av funktionen *StartAsServer*.
3. I samband med att värdens sänder ut sin IP-adress använder de andra användarna NetworkDiscovery-funktionen *StartAsClient*. Denna funktion lyssnar efter och samlar in värdens utsända signal.
4. När utsänd signal har mottagits används IP-adressen för att ansluta till värdens server. Först sätts IP-adresserna hos användarnas lokala NetworkManager-instanser till samma som serverns och därefter kallas funktionen *StartClient*. När denna funktion kallas skapas en nätverksklient-komponent som direkt kopplas upp mot servern.

Punkterna som beskrivs ovan kan kopplas till aktivitetsdiagrammet i Figur 4.1, där punkt 1-2 sker i diagrammets vänstra flöde (från *Create Lobby*) och punkt 3-4 i det högra flödet (från *Find Lobby*). Vad som inte är inkluderat i beskrivningen ovan är *Create Player Object* samt *Tell All Clients To Load Game Scene*.

Nätverkskomponenter

För att användare ska kunna skicka information över nätverket krävs det att de har en komponent med nätverksauktoritet. Detta uppnåddes genom att samtliga användare som ansluter till servern skapar egna spelareobjekt kallade *PlayerObject* med komponenten *NetworkIdentity*, detta kan kopplas till aktivitetsdiagrammet i Figur 4.1 (*Create Player Object*). *NetworkIdentity* kontrollerar spelobjekts unika nätverksidentiteter som är kopplade till klienterna i nätverket. Klienterna kan sedan använda dessa spelobjekt för att skicka kommandon till servern eller skapa nya spelobjekt över nätverket som endast klienten har tillstånd över. De objekt som tilldelades *NetworkIdentity* är: *PlayerObject* för samtliga klienter, myror som skapas samt sockerbitarna som myrorna anfaller. Spelarobjekten används främst för att skapa och döda myror eftersom detta måste ske genom nätverket. Sockerbitarna krävde också en *NetworkIdentity*-komponent eftersom all mottagen skada måste ske synkroniserat över nätverket så att bara ett av lagen blir vinnare.

Spelets tillstånd

Som samlingsplats för nätverksuppkopplingen användes ett lobbysystem. Processen för att ansluta till lobbyn beskrivs i avsnittet 4.2.1 ovan. När samtliga användare är anslutna och redo kallar lobbyvärdens på en funktion för att starta spelscenen. Detta sker genom att värdens skickar ett kommando till servern som sedan beordrar alla klienter att kalla på en funktion för att ladda samma scen. Denna typ av funktion kallas för *ClientRPC* och kan kallas av alla spelobjekt som har nätverksidentiteter. På samma vis synkroniseras meddelandet när ett lag vinner och matchen tar slut.

4.2.2 Rörelsespårning

Accelerometer/Gyroskop

Initialt utfördes tester genom att använda accelerometern för att utföra spårningen av enheternas rörelse. Accelerometern visade sig dock inte vara lämpad för den precision av rörelsespårning som spelet kräver. Sensorn mäter den mobila enhetens acceleration, som sedan kan integreras i två steg för att få fram dess position. Problemet uppstår när gravitationskrafter ska räknas bort för att få fram den resterande accelerationen. För att göra dessa beräkningar måste enhetens exakta orientering vara känd, vilket är svårt med de sensorer som finns i enheterna [20]. Positionsfelet tenderar även att öka kraftigt vid den dubbla integrationen som sker. Det räcker med ett orienteringsfel på 0.1 grader för att positionsfelet ska öka till över 1 meter på 10 sekunder. Accelerometern testades även i Unity. Den visade sig vara mycket opålitlig, vilket även andra användare upplevt [21]. Resultatet blev ryckigt i responsen vilket inte var användarvänligt. Idén att använda accelerometern övergavs.

ArUco

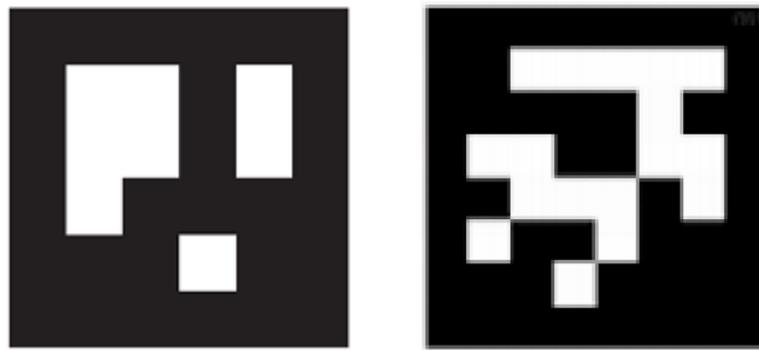
Den kvarvarande metoden för rörelsespårningen var förstärkt verklighet. ArUco var den implementation som prioriterades. Detta på grund av att denna API tillåter lovande punktdetektering och kamerakalibrering, som också hade rekommenderats av intressenter med expertiskompetens. Implementationen visade sig mindre lämpad. ArUco kalibrerar kameran för området vilket inte är lika lätt att implementera med andra API så som Unity. Resultatet blir däremot en mer noggrann rörelsespårning spelupplevelse. Projektgruppen spenderade en stor del av projekttiden för att försöka integrera ArUco med Unity men bristfällande dokumentation och ett flertal andra motgångar innebar att denna teknik också övergavs.

Vuforia

När ArUco visade sig vara svårimplementerat i Unity undersöktes Vuforia. Fördelen är att Vuforia, till skillnad från ArUco, är direkt integrerat med Unitys Editor [22]. Via intressenter av expertiskompetens framgick det även att denna API används vid rörelsespårning, i kombination med Unity, för att skapa andra produkter. Nackdelen är att det vid tester framgått att hela markören måste synas i enhetens kamera för att kunna placera ut den virtuella spelplanen. För att implementera AR-konceptet via Vufora krävs tillägg av två spelobjekt i scenen. Det ena är en AR-kamera som aktiverar AR-funktionaliteten [23], medan den andra är ett objekt kopplat till detektionsmarkören. Det sistnämnda objektet har samma form av yta som den verkliga markören, vilken i detta projekt var ett plan. De spelobjekt som sedan placeras under markörens objekt hierarkiskt placeras ut utifrån den verkliga markören.

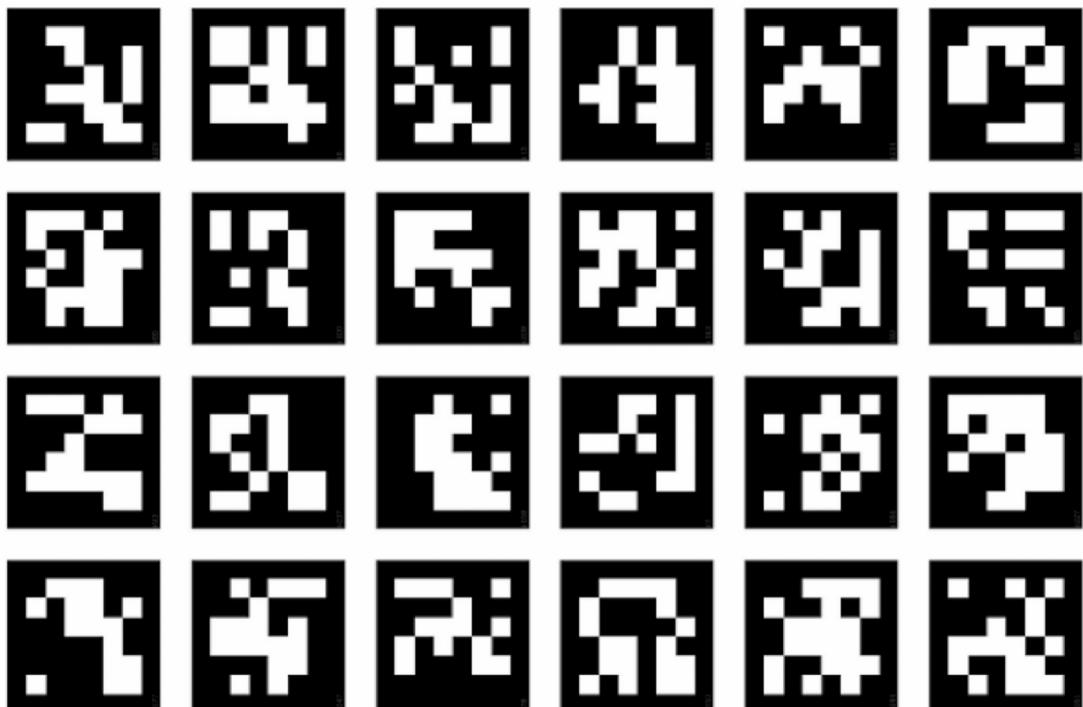
Med ovan nämnda konsekvenser valdes den slutgiltiga metoden för rörelsespårning att implementeras till Vuforia.

Flera markörer har undersökts i kombination med Vuforia. Vid valet av markör togs dess betyg i Vuforia och verkliga storlek i åtanke [5]. Uppskalningen av de bilder som skapats med rastergrafik bidrog med en pixlig markör som tappade precision vid punktdetektering. Istället användes vektorgrafik som innebar att markören kunde skalas oändligt. Från den tidigare undersökningen av ArUco hade ett rutnät av ArUco-markörer studerats. Dessa studerades även som punktdetekterings-bilder vid användandet av Vuforia. Noterbart var att ArUco-markörernas dimensioner påverkade betyget mer än förändringen av pixelrutnätets dimensioner. I Figur 4.2 demonstreras två olika dimensionsstorlekar av ArUco-markörer. Ett rutnät av de som är 8 x 8 pixlar resulterade i ett bättre betyg än ett rutnät av 6 x 6 pixlar.



Figur 4.2: Två ArUco-markörer som skiljer i storlek. Den vänstra är 6×6 pixlar, medan den högra är 8×8 pixlar

Den slutgiltiga bild som användes som markör i projektet visas i Figur 4.3. Bilden är ett 6×4 -pixelrutnät av ArUco-markörer, vars dimensioner är 8×8 pixlar. Den bedömdes till högsta betyg av Vuforia och på grund av att den skapats i vektorgrafik kan den skalas oändligt.

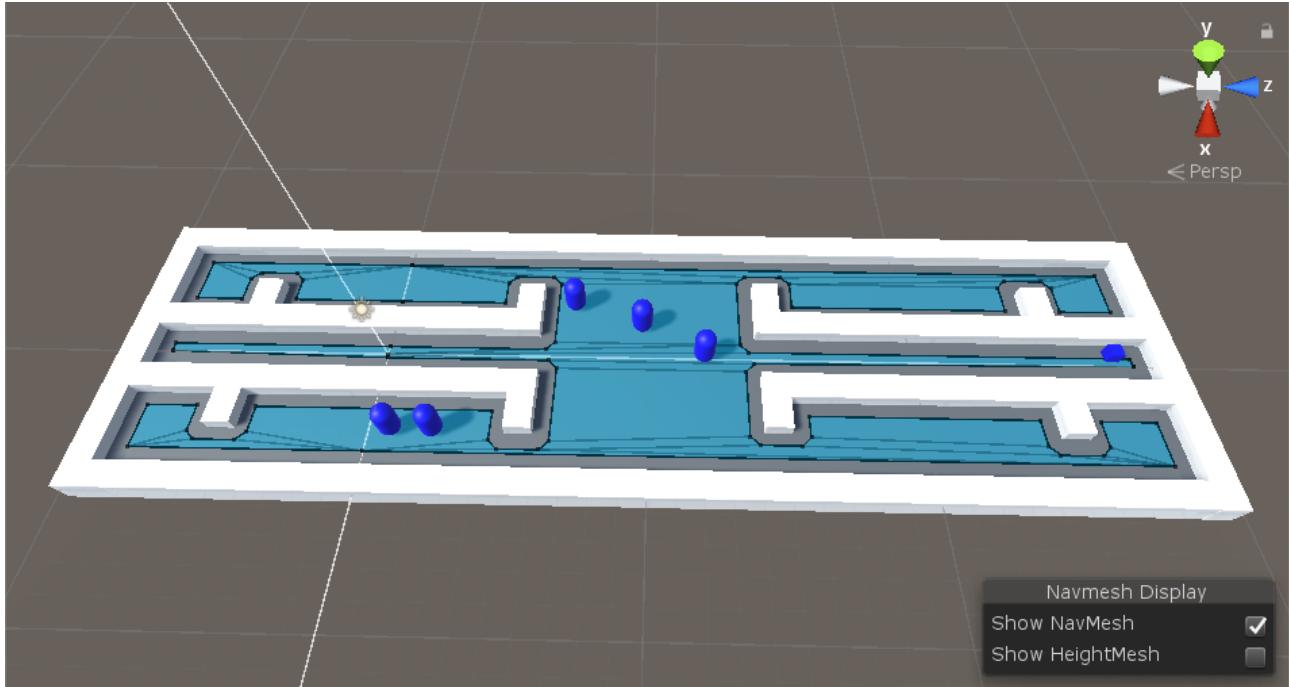


Figur 4.3: Den slutgiltiga projektmarkören som är ett ArUco-bräde bestående av ett 6×4 -pixelrutnät av ArUco-markörer med storlek 8×8 pixlar

4.2.3 Spellogik

Vid implementeringen av spellogiken användes idéer från de relaterade produkter som forskades kring under förstudierna av projektet. Spelets estetik valdes att försöka efterlikna spelet Equilinox med dess *Low Poly*-stil. Spelkonceptet bestämdes att utvecklas likt spel inom genren *MOBA*. Framförallt utnyttjades idén kring att ha tre olika huvudvägar som alla leder fram till samma mål (motståndarens bas). Till att börja med skapades en scen i Unity med varken nätverk eller rörelsespårning inblandat. Här skapades ett första utkast av ett spel genom användning av en *NavMesh* och olika *NavMesh Agents*

som kunde förflytta sig mot en specifik destination på banan. Första versionen av spelet syns i Figur 4.4. Anledningen till att en NavMesh användes istället för alternativet med att använda *spline-kurvor* var på grund av att implementeringsmetoden för en NavMesh var väl dokumenterad i Unitys egna manual om navigering. Vid användningen av NavMesh Agents löstes även problemet med kollisionshantering, vilket var en bidragande faktor till valet.



Figur 4.4: Första versionen av spellogiken med agenter som rör sig mot en destination i form av en blå kub

Utifrån den första versionen av spelet lades sedan ytterligare funktioner till i spelet för att förbättra spellogiken. Dessa beskrivs nedan.

Att skapa myror

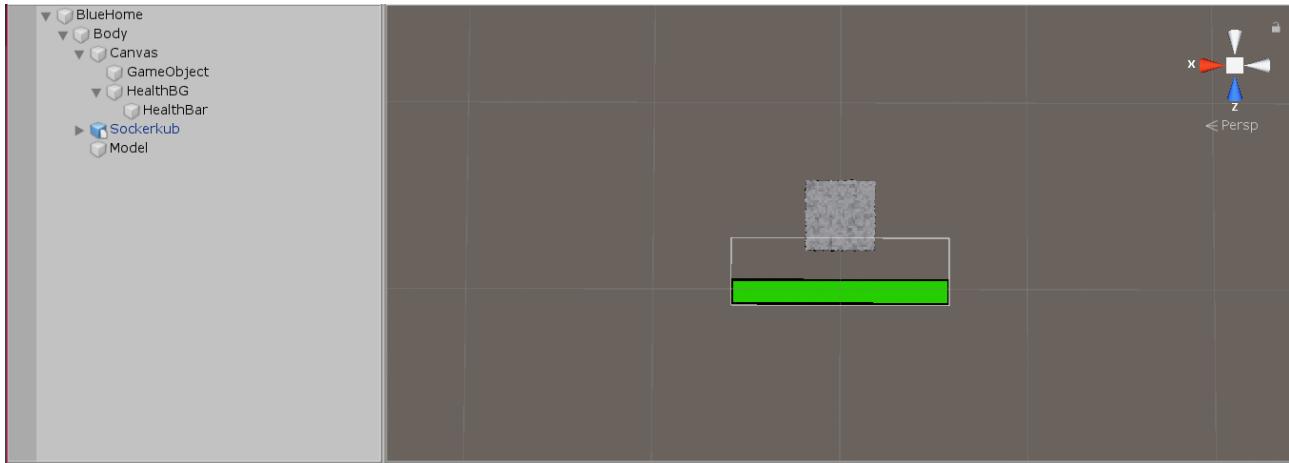
De tidigare implementerade NavMesh Agents byttes ut mot spelets karaktärer i form av modellerade myror. Hur myrorna modellerades och animerades kommer att redogöras för i kapitel 4.3. För att användaren skulle kunna skicka iväg myror implementerades tre olika myrstackar per lag på spelplanen. Dessa myrstackar motsvarar olika vägval för myrorna att gå och spelidén är lik den i spelgenren MO-BA, som nämndes i kapitel 2. När myrorna skapades på en av dessa platser kunde de genast använda sig av spelplanens *NavMesh* för att påbörja sin rörelse mot motståndarens bas.

Att döda myror

Efter att myrorna kunde skapas implementerades också en metod för att döda myrorna. Projektgruppen beslutade att en användare skulle kunna döda en myra genom att klicka på den. För att ge effekten av att en myra faktiskt dör infördes även en visuell effekt av att myran mosas genom att förändra myrans geometri till en mer tillplattad.

Lagens baser

Två olika baser implementerades i spelplanen, en för vardera lag. Baserna sattes som destinationer för de olika lagens myror (NavMesh Agents), tillsammans med varsin mätare som visade basernas hälsa. För att få en bas att ta skada när motståndarens myror når den tillämpades användningen av kollisionshantering. När en eller flera myror når en bas önskades en effekt som indikerar detta. En funktion implementerades som gör att mätaren börjar blinka kraftigt då basen tar skada. Baserna visualiseras i formen utav en sockerkub som myrorna vill äta på.



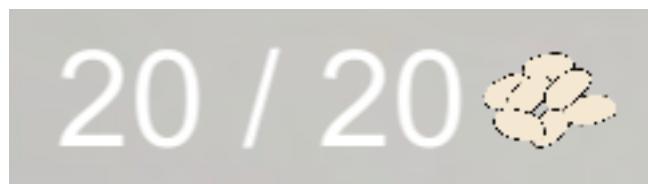
Figur 4.5: Basen i form av en sockerkub, med en mätare som visar basens hälsa implementerad

Utveckling av spelplanen

Vidare i projektet utvecklades spelplanen successivt till en spelplan med fler objekt och fler hinder. Detta påverkade spellogiken dels genom myrornas nya beteenden på spelplanen och dels genom användarnas nya navigerade genom spelplanen.

Resurssystem

För att lägga till ett strategiskt element i spellogiken implementerades ett resurssystem. Spelets resurs kallas *AntEggs* vilket symboliseras valutan. En spelares möjlighet att skapa myror begränsas av en maxkapacitet av *AntEggs*. Dessa fylls sakta på med avseende på tiden samtidigt som det sänks varje gång en spelare skapar myror som en kostnad. En myra kostar ett *AntEgg* och med tidens gång ökar antalet *AntEggs* spelaren får exponentiellt. Motiveringen för detta är att främja taktiskt tänkande genom att exempelvis ge en spelare möjligheten att samla på sig *AntEggs* och skapa många myror på en gång för att överraska motståndarna. Ytterligare en motivering för exponentiell ökning av resurser är att en match ska bli mer intensiv ju längre tid den pågår.

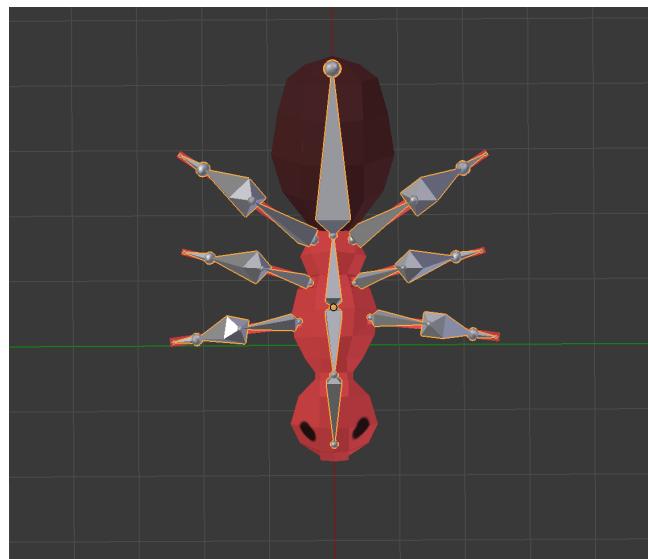


Figur 4.6: Antalet *AntEggs* visas med en siffra bredvid en ikon

4.3 Modellerings och Animering

Low Poly är en modelleringsstil där modellen medvetet innehåller ett lägre antal polygoner och används frekvent inom realtidsrendering, som exempelvis spel. Modelleringsstilen användes vid framtagandet av samtliga modeller i spelet då hårdvaran som applikationen ska köras på är begränsad. Modelleringstilen, tillsammans med enkla texturer, är också fördelaktigt i den aspekt att den tilltalar en större målgrupp [24].

Samtliga modeller modellerades och animerades i 3D modelleringsmjukvaran *Blender* då animatörer kunde enkelt exporteras till Unity i form av FBX-filer samt att mjukvaran var gratis för både kommersiellt och icke-kommersiellt bruk. Efter att en *mesh* hade modellerats utefter referensbilder konstruerades ett skelett med en benstruktur och ledar som ska kunna efterlikna rörelserna efter hur de reella referenserna rör sig. Exempel på en skelettuppbryggnad kan ses i Figur 4.7 som användes för myrorna i spelet. Större benen tilldelas en tyngre vikt än de mindre, vilket gör att de påverkas annorlunda beroende på interpolationsmetoden som används. Interpolationsmetoden varierade mellan myran och resterande objekt. För att få ett kontinuerligt flödeförlopp utan oregelbundenheter i animationen användes linjär interpolering. Detta innebär att ett linjärt rörelseförlopp med avseende på tiden beräknas mellan två *keyframes*. Vid animationen av objekt som inte krävde kontinuerliga animationsförlopp användes *Bezier* kurvor vilket resulterade i mjukare rörelser.



Figur 4.7: Den 3D-modellerade myran från ovan med sitt animeringsskelett

4.4 Sammanställning av huvudkomponenter

Eftersom alla tre huvudkomponenter utvecklades var för sig med hjälp av Unity krävdes endast små men väsentliga justeringar för att få komponenterna att fungera korrekt tillsammans. Den största utmaningen var att integrera *NavMesh*-systemet som beskrivs i avsnitt 4.2.3 tillsammans med nätverk och rörelsespårning. Spellogikskomponenten med ett implementerat *NavMesh*-system blev grunden som de andra två komponenterna fördes in i.

Den första sammanställningen gjordes mellan spellogik och nätverkslösning. För att myror skulle kunna skapas, förflyttas och raderas som en del av nätverket utvecklades en funktion hos klientens spelareobjekt (*PlayerObject* som nämns i avsnitt 4.2.1) för att instansiera dem. När ett spelareobjekt instansierar en myra tilldelas den en nätverksidentitet som endast skaparen har auktoritet över. Varje

myra blev då ett nätverksobjekt och dess position kunde uppdateras som vanligt med hjälp av sin *NavMesh Agent* som beskrivs i avsnitt 4.2.3. Slutligen för att radera myrorna från scenen krävdes det att spelareobjekten skickade kommandon till servern som i sin tur förstörde myrans spelobjekt.

Det största problemet uppstod när rörelsespårningskomponenten skulle integreras med de andra två komponenterna, specifikt med NavMesh. Unitys NavMesh-system är strikt utformat för att spelplanen ska vara statisk, det vill säga: inga modeller som ingår i NavMesh-komponenten får ändra sina positioner under spelets gång. Rörelsespårningsmetoden fungerar genom att spelvärlden förflyttas med avseende på den fysiska kamerans position, det vill säga NavMesh-navigeringen måste förflyttas på motsvarande sätt.

En lösning på problemet skulle vara att skapa en ny NavMesh för varje bilduppdatering vilket inte var ett alternativ för detta system då det skulle vara en prestandakrävande uppgift. Istället löstes problemet genom att endast transformera myrans modell och kollisionsyta till spelplanen. Konkret innebar detta att transformationsmatrisen för myrans modell blev en sammansättning av sin NavMesh Agents matris som *child* och spelplanens matris som *parent*. För att denna lösning ska fungera måste alla NavMesh Agent-komponenter tillsammans med dess utgångs- respektive målpositioner ligga utanför hierarkin för rörelsespårningens förflyttningar.

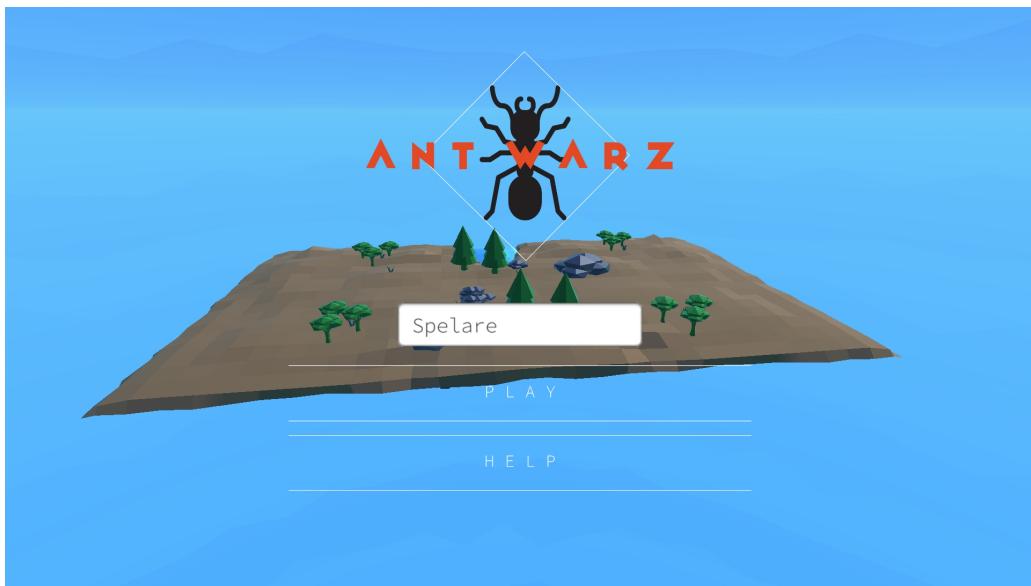
Kapitel 5

Resultat

Resultatet av projektet är ett digitalt sällskapsspel anpassat för handhållna enheter av operativsystemet Android. Via en lobby kopplar spelarna upp sig mot ett gemensamt nätverk och delas in i två olika lag. För att kunna se spelplanen krävs att alla användare befinner sig vid samma spelmarkör. Genom enhetens kamera kan sedan spelmarkören användas för att läsa in den virtuella spelplanen. När spelplanen är synlig för alla spelare kan spelet börja och lagen interagera med varandra över spelplanen. Det lag som först förstör det andra lagets bas har vunnit.

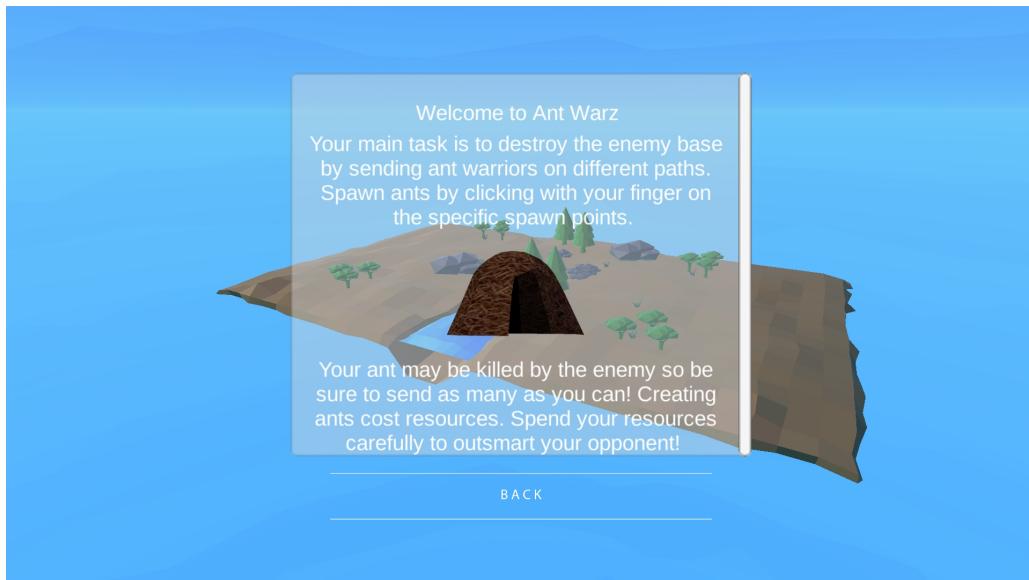
5.1 Användargränssnitt

När en användare startar spelet möts den först av en välkomstsida som presenterar spelet. På välkomstsidan, som visas i Figur 5.1, finns två knappar, en spelknapp (*Play*) och en hjälpknapp (*Help*).



Figur 5.1: Välkomstsidan som är det första användaren möts utav

Genom att klicka på hjälpknappen kommer användaren till en hjälpsida som innehåller information om spelets regler och om hur spelet fungerar. Hjälpsidan som presenteras i spelet visas i Figur 5.2.



Figur 5.2: Hjälpsidan som fungerar som guide för en användare som spelar för första gången

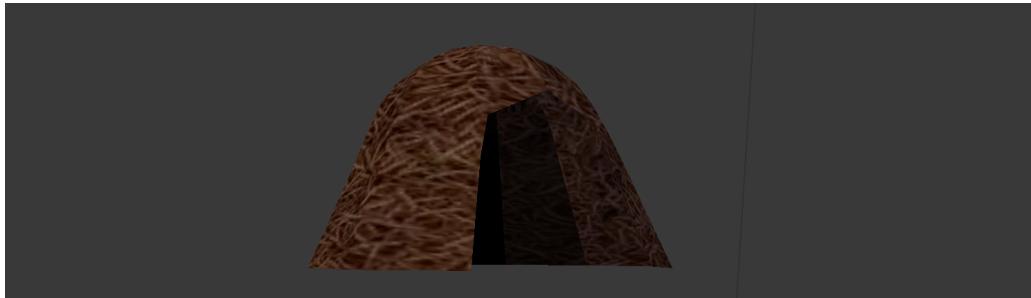
En spelare kan med hjälp av en meny skapa, eller söka och ansluta sig till, en lobby över ett nätverk. Lobbyen kallas i spelet *War Zone*. Alla spelare måste vara uppkopplade mot samma nätverk för att kunna ansluta sig till samma lobby. Därefter blir alla deltagare indelade i ett blått eller rött lag, varpå den spelare som skapade lobbyn sätter igång spelet. Lobbyen presenteras i Figur 5.3.



Figur 5.3: Lobbyen (War Zone) visar alla spelare samt vilket lag de tillhör

Väl inne i spelet finns en knapp uppe till höger som tar spelaren tillbaka till menyn om den inte vill spela längre. Utöver detta sker all interaktion på spelplanen. Varje spelare kan välja att trycka på en av tre myrstackar som skapar varsin myra vid varje tryck. Så fort myran skapas går den i riktning mot fiendebasen. Väl ute på spelplanen kan spelarna krossa varandras myror genom ett knapptryck och det är även möjligt att råka krossa sina egna myror. Myrstackarna visas nedan i figur 5.4.

Vid spelets slut presenteras båda lagen av en skärm som berättar vilket lag som har vunnit och det ges en möjlighet att återvända till startmenyn, för att skapa en ny lobby och spela igen. Detta visas nedan i figur 5.5.



Figur 5.4: Myrorna skapas i myrstacksliknande strukturer och rör sig sedan mot motståndarens sida



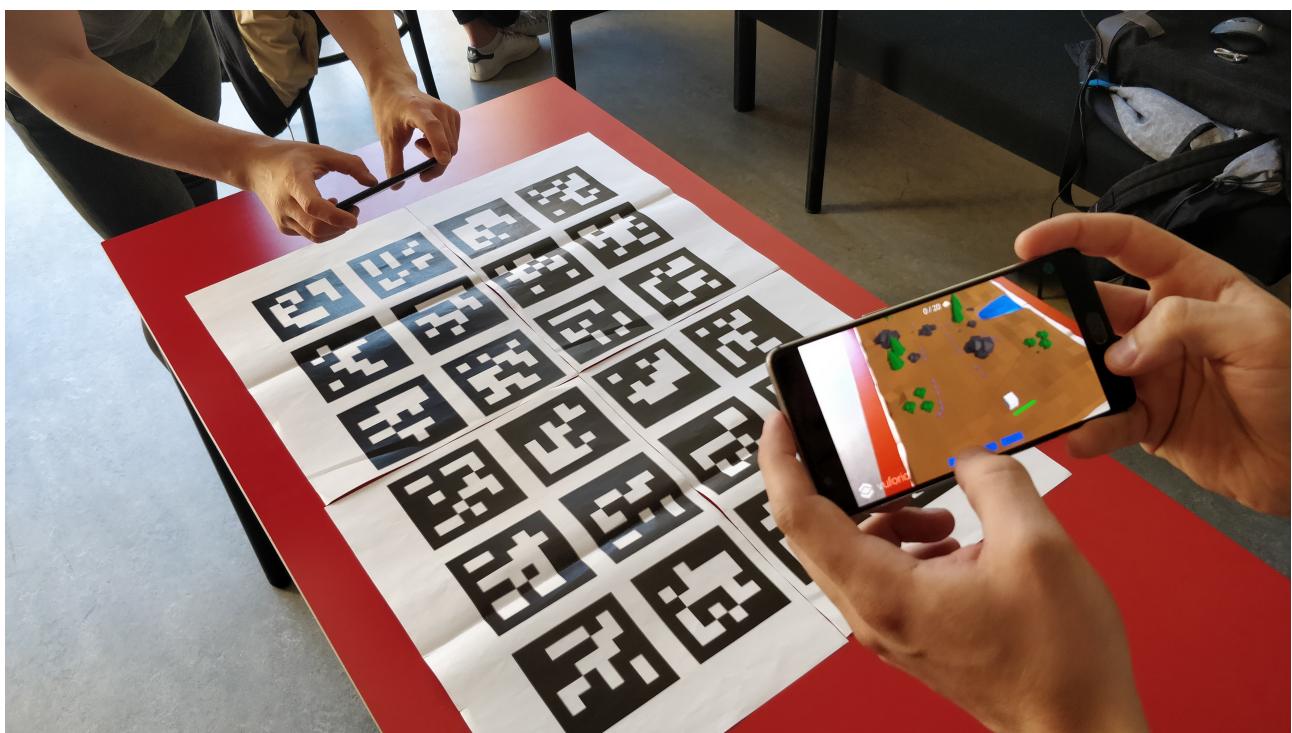
Figur 5.5: Vid spelets slut ser användaren vilket lag som vunnit och får möjligheten att återvända till lobbyn

5.2 Spelet

Spelet går ut på att två lag ska skicka iväg myror mot varandras bas samtidigt som de försvarar sin egen. För att skicka myror finns det tre olika fördefinierade startpunkter att välja mellan. Genom att trycka på en startpunkt skapas en myra som nавигeras sig från startpunkten mot fiendebasen. För att kunna skicka myror har laget en viss mängd resurser som visas i en mätare i hörnet av spelvyn och för varje myra som skickas minskas dessa resurser. Om resurserna hos ett lag tar slut måste de vänta med att skicka myror tills resurserna, som automatiskt ökar över tid, fyllts på igen. Basen, som utgörs av en sockerbit, försvaras genom att klicka på motståndarens myror och på så sätt döda dom innan de når fram till sockerbiten. Om myrorna hinner fram till motståndarens sockerbit utan att bli dödade ger de skada på sockerbiten, vilket visas av sockerbitens hälso-mätare. När sockerbiten skadats tillräckligt många gånger förstörs den och laget som då har sin sockerbit kvar vinner spelet. Det slutgiltiga spelet visas i Figur 5.6 och Figur 5.7.



Figur 5.6: Den sluttgiltiga spelplanen med två spelare som skickar myror mot varandra



Figur 5.7: Hur sällskapspelet ser ut när två användare spelar

Kapitel 6

Analys och diskussion

I följande kapitel diskuteras och analyseras arbetet som utförts under projektets gång. Detta görs utifrån arbetsprocessen och resultatet.

6.1 Arbetsprocess

Arbetsprocessen planerades och strukturerades upp på ett bra sätt av projektgruppen utifrån teori om metodiken Scrum. Att sedan implementera denna teorin visade sig dock svårare än tänkt. Exempelvis planerades sprintar att vara två veckor långa men under utvecklingsprocessen prioriterades andra åtaganden vilket resulterade i att sprintlängden överskreds. Detta medförde att ett flertal omstruktureringar i projektet blev nödvändiga för att projektmålen skulle uppfyllas inom tidsramen. För detta var det fördelaktigt att den agila utvecklingsmetodiken Scrum användes, eftersom det bidrog till att det snabbt och enkelt gick att omprioritera delar i arbetet.

Ytterligare en brist i tillämpningen av teorin var hur förstudierna till projektet organiserades. Det fanns inga tydliga mål och riktlinjer för när förstudier ansågs vara klara vilket resulterade i att för mycket tid gick åt för denna period. Om teorin bakom Scrum, som exempelvis att dela upp arbetet i sprintar, hade tillämpats redan under förstudierna hade detta kunna undvikits. Med tanke på att projektgruppen aldrig tidigare tillämpat Scrums metodik är detta dock förståeligt.

Något som fungerade bra i praktiken var indelningen av olika mindre fokusgrupper, där parprogrammering tillämpades. Genom denna uppdelning kunde projektgruppen arbeta parallellt med de olika systemkomponenterna och på så sätt effektivisera arbetet.

Försöken till att implementera ArUco så att det skulle samverka med Unity visade sig vara mer avancerat än projektgruppen förväntat sig. ArUco är skrivet i programspråket C++ och OpenCV medan Unity använder sig utav C#. Detta i samband med bristfällig dokumentation över tillämpning och över integration med Unity gjorde det svårt för projektgruppen att, med sina begränsade programmeringskunskaper, implementera ArUco. Vuforia är dock redan integrerat med Unity och tillhandager god dokumentation över hur tekniken tillämpas.

6.2 Resultat

I detta avsnitt diskuteras resultatet av projektets slutprodukt.

6.2.1 Användargränssnitt

För att ge användaren ett professionellt intryck och främja användarvänlighet ansåg projektruppen att det var viktigt att spelet skulle följa en tydlig grafisk profilering. Detta innefattar allt ifrån att knappar ska ge bra återkoppling vid tryck till att ett tydligt tema följs under hela användarupplevelsen. I resultatet för användargränssnitt presenteras spelets olika menyer och knappar som anses uppnå detta. Den valda designen utvecklades för att tilltalा en större målgrupp och göra det enkelt för användaren att komma igång. Med detta i åtanke skapades en logga, knappdesign, en spelplan och myror som passade in på det valda temat. Vid valen av knappars placering var spellogiken i åtanke. Till exempel placerades knappen för att avsluta spelet uppe i högra hörnet för att förhindra spelaren oavsiktligt råkar trycka på den knappen under spelets gång. Ett annat exempel på hur gränssnittet har anpassats efter spellogiken är placeringen av mätaren för basens hälsa. I detta fall placerades mätaren ovanför kuben för att motivera användare att röra på sin enhet över spelplanen.

6.2.2 Spelet

Resultatet av det här projektet var ett digitalt sällskapsspel för handhållna enheter som implementerar AR för att spåra enheters position över spelplanen, vilket var syftet med arbetet. Spellogiken uppfyller den initiala kraven vilket var att två lag skulle kunna skicka myror med mål om att förstöra motståndarens bas. Den simplistiska spelidén ger dock vidare utvecklingsmöjlighet i det avseende att ytterligare funktionalitet inte löper risk att förhindras av den redan implementerade spellogiken. Planerad vidarutveckling av spelidén nämns i avsnitt 7.4. Om gruppen skulle ha påbörjat utveckling tidigare med Vuforia, istället för att försöka implementera ArUco, skulle mer fokus kunna ha lagts på implementation av spellogik.

Som tidigare förklarat i avsnitt 2.4.2 valde projektgruppen att följa ett tema som baserades på Low Poly-modellering som innehåller ett lågt antal polygoner i 3D-modellerna. Detta gör att mindre påfrestning ställs på de handhållna enheterna under spelets gång. Användningen av Low Poly-modellering ger också ett stilrent utseende och tilltalar en stor andel människor men främst den yngre målgruppen. Projektgruppen tyckte att det kändes viktigt att göra spelet barnvänligt samt att det skulle vara enkelt att komma igång. Vidare är spelet väldigt färgglatt med tydliga kontraster och skuggor vilket tillför mycket för den visuella upplevelsen.

Målet för projektet var att skapa ett sällskapsspel för flera användare och detta är något som har tagits hänsyn till genom hela utvecklingsprocessen. Exempel på detta är att nätverksarkitekturen klient/server användes då denna struktur förutsätter att alla användare är uppkopplade mot samma nätverk. Gällande spellogiken finns ett resurssystem inbyggt. Vid skapandet av en myra minskas lägets sammanlagda resurser vilket skapar en strategisk aspekt på spelet som gynnar samarbete mellan lagmedlemmar.

6.3 Arbetet i ett vidare sammanhang

Spelkonceptet är framtaget att bjuda in människor till socialt umgänge, på samma vis som fysikaliska sällskapsspel. Med det i åtanke har åtgärder tagits så att spelarna endast behöver använda sin mobiltelefon samt att någon har tillgång till en markör för att spåra spelplanen. Detta anses vara rimligt då andra sällskapspel kräver att spelarna har tillgång till ett fysiskt spelbräde eller pjäser. Då tiden var begränsad valde arbetslaget att bygga mot *Android*, då operativsystemet används i större utsträckning på en global skala [25]. Det innebär att mobilanvändare med *iOS* inte kan ta del av produkten, vilket kan anses som exkluderande i visst avseende och är något som arbetslaget borde ha i åtanke vid vidare utveckling av produkten.

Det är vanligt att digitala spel har en integrerad chattfunktion som kommunikationsmedium. En undersökning av utförd av Friends visar att nätmobbning blir allt vanligare [26]. Orsaken till detta anses vara att användare känner sig trygga bakom datorskärmarna då fysikalisk konfrontation inte kan ske. Detta spel är framtaget så att kommunikationen inte sker över nätet och på så vis förhindrar mobbning av detta slag vid användning av produkten.

Spelidén i den resulterade produkten baserades på etablerade MOBA- och RTS-spel. Dessa spel, liksom den resulterade produkten, innefattar våld som kan påverka spelaren negativt. Detta är något som forskas om i stor utsträckning där somliga anser att barn som utsätts för våld tidigt blir mindre empatiska samt aggressivare [27], det finns dock studier som sägs motbevisa påståendet [28]. Något som är värt att ha i åtanke är att produkten även har element av Förstärkt verklighet, vilket kan resultera i att våldet förstärks. Detta gör modelleringssstilen Low Poly mer attraktiv då den sänker graden av realism och på så vis ska förhindra att våldet förstärks. Vidare bör ett beslut tas om spelet ska ha en åldersgräns.

6.4 Källkritik

En av projektets främsta källa för dokumentation är Unitys maunal och en mindre del deras forum. Forum är i allmänhet otrovärdiga. Det är svårt att veta om skribenten har teknisk kunskap som backar den information de delar. Trovärdheten ökar dock på grund av att Unitys forum modereras flitigt och att information ofta kommer direkt från Unitys utvecklare.

Kapitel 7

Slutsatser

7.1 Rörelsespårning

Vilka relevanta tekniker finns det för rörelsесpråning av handhållna enheter och är någon mer lämpad vid utvecklingen av ett digitalt sällskapsspel för flera användare?

Det finns flera olika tekniker för att implementera rörelsespårning. Ett alternativ är att utnyttja de mobila enheternas accelerometer/gyroskop men detta ger missvisande respons vid rörelsespårning. Det visar sig att positionsfelet som uppstår vid dubbel integration av den uppmätta accelerationen ökar med tiden. ArUco är ett bibliotek som tillåter lovande punktdetektering och kamerakalibrering. Däremot saknas det relevant dokumentation för integrering med Unity som projektgruppen har skapat spelet med.

Det bäst lämpade alternativet är att använda ett bibliotek/ramverk som är integrerat med en spelmotor. Vuforia har god integration med spelmotorn Unity. Unity har även möjligheten att bygga applicationer till de givna handhållna enheternas operativsystem Android, vilket var ett av kundens krav. När kameran inte kan detektera markören kombinerar Vuforia rörelseinformationen från accelerometern/gyroskopet för att hjälpa till vid spårningen.

7.2 Nätverkshantering

För att användare ska kunna spela mot varandra behöver de vara uppkopplade mot ett nätverk. Hur bör en sådan nätverksarkitektur se ut och vilken information behöver skickas över nätverket när spelet är av en RTS/MOBA-genre?

För att användarna ska kunna spela mot varandra lämpar sig en klient-/server nätverksarkitektur bäst. Med detta kan en spelare vara värd åt resterande spelare med sin enhet. Den information som behöver skickas över nätverket är den absoluta positionen för alla interagerbara spelobjekt på spelplanen i realtid, båda sockerbitarnas livspoäng samt spelares och myrornas nätverksidentifikationer. Motiveringen till varför all denna information behöver överföras över nätverket är för att en spelare måste kunna se när en myra dör och skapas, men också för att veta när ett lag har vunnit. Ett exempel på information som ej behöver överföras är en spelares resurser då dessa är personliga för sin klient. Informationen som skickas över nätverket baseras på nätverksprotokollet UDP (User Datagram Protocol) som är optimalt för spelsystem där låg latens krävs.

7.3 Spelidé

Hur kan en spelidé utvecklas för att främja ett socialt och interaktivt digitalt sällskapsspel för flera användare?

Spelidén bör utformas för att engagera alla spelare i givande uppgifter som är varierande, för att ge en stark spelupplevelse. Genom att ge möjligheten för spelarna inom lagen att delegera roller främjas samarbete där alla interagerar på olika sätt. Eftersom spelplanen är förankrat vid ett gemensamt bord kan lagen tillämpa olika strategier för att få övertaget över sina motståndare. Då nätverksarkitekturen klient/server användes ska det främja den sociala aspekten i det avseende att användarna behöver vara uppkopplade mot samma nätverk.

7.4 Utvecklingsområden

Vid vidarutveckling av systemet önskar arbetslaget att utveckla spelidén till att ha flera myror som användarna kan skicka. Exempelvis ska det finnas tre sorters myror som användaren kan skicka, en vanlig myra, en svartmyra och en hästmyra. Dessa myror skulle då ha styrkor och svagheter i form av att svartmyror är längsammare men hårdigare och hästmyror är snabba men skadar inte sockerbiten lika mycket. Spelet skulle även kunna innehålla *power ups*, så som andra metoder för att döda myror. Exempel att det är möjligt att aktivera attacker som dödar kluster av myror. På samma sätt skulle spelaren kunna skapa kluster av myror samtidigt. Tanken med detta är att göra spelupplevelsen mer dynamisk och införa ett mer distinkt skicklighetsmoment i spelet. Ett annat utvecklingsområde skulle vara att generera varierande hinder på spelplanen för varje omgång. Detta skulle kunna bidra till att spelmiljön känns utforskad och på så vis bibehålla användarens intresse för produkten.

För att aktivera power ups, byta myror och liknande kan spelet behöva ett mer avancerat användargränssnitt. Till slut kan utvecklingen nå ett stadie där inte enheternas skärmar tillåter fler tillägg av spelfunktioner. För att lösa detta skulle vardera lag kunna ha en separat markör vid sidan av spelplanen. Denna markör blir då en plats där olika funktioner kan kontrolleras. Därav hålls användargränsnittet minimalistiskt när användaren hovrar över den huvudsakliga spelplanen. Vid en fristående markör skulle vissa användare även kunna spela minispel för att samla in AntEggs.

Enligt användartester, se Bilaga B, önskades vidareutveckling av gränssnittet i form av tydligare menyer och bakgrund till text-element samt andra grafiska element för att se resurserna i spelet. Arbetslaget har tagit hänsyn till detta och önskar ta fram fler prototyper i ett senare sammanhang för att satisfiera spelarbasen.

Litteraturförteckning

- [1] Tom Wijman, *Global Games Market Revenues 2018*, NewZoo, 2018-04-13, hämtad: 2019-05-09
<https://newzoo.com/insights/articles/global-games-market-reaches-137-9-billion-in-2018-mobile-games-take-half/>
- [2] Unity Technologies, *System Requirements*, hämtad: 2019-05-12
<https://unity3d.com/unity/system-requirements>
- [3] Jospeh Rampolla och Gregory Kipper, *Augmented Reality: an emerging technologies guide to AR*, Elsevier, 2012
- [4] Steven Finer, Balir MacIntyre m.f, *A touring machine: Prototyping 3D mobile augmented reality systems for exploring the urban environment*, Personal Technologies, 1997
- [5] Parametric Technology Corporation, *How To Define Image Target Parameters*, 2016-02-05, hämtad: 2019-05-15
<https://library.vuforia.com/articles/Training/Image-Target-Guide>
- [6] Amir Yahyavi och Bettina Kemme, *Peer-to-Peer Architectures for Massively Multiplayer Online Games: A Survey*, 2013-09-01, hämtad: 2019-02-15 <https://www.researchgate.net/publication/262233529-Peer-to-Peer-Architectures-for-Massively-Multiplayer-Online-Games-A-Survey>
- [7] FairCom, *Server Advantages vs. Multiuser Standalone*, 2014-06-19, hämtad: 2019-05-15
<https://docs.faircom.com/doc/knowledgebase/55837.htm>
- [8] UNet, *Multiplayer and Networking*, Organisation, 2019-04-15, hämtad: 2019-05-15
<https://docs.unity3d.com/Manual/UNet.html>
- [9] IT-ord, *Java virtual machine*, Organisation, 2014-08-05, hämtad: 2019-05-15
<https://it-ord.idg.se/ord/java-virtual-machine/>
- [10] Brandi House, *Evolving multiplayer games beyond UNet*, Unity, 2018-08-02, hämtad: 2019-02-15
<https://blogs.unity3d.com/2018/08/02/evolving-multiplayer-games-beyond-unet/>
- [11] Unity Technologies, *Navigation and Pathfinding*, hämtad: 2019-05-12
<https://docs.unity3d.com/Manual/Navigation.html>
- [12] Donald H. House, *Spline Curves*, Computer Graphics CPSC 4050, spring 2014, hämtad: 2019-05-15
<https://people.cs.clemson.edu/~dhouse/courses/405/notes/splines.pdf>
- [13] Wikipedia, *MOBA*, u.å. hämtat: 2019-05-11,
<https://sv.wikipedia.org/wiki/Multiplayer-online-battle-arena>

- [14] Valve, *Dota 2*, u.å. hämtad: 2019-05-15
<http://blog.dota2.com/?l=english>
- [15] ThinMatrix, *About*, 2018, hämtad: 2019-05-15
<https://equilinox.com/about/>
- [16] Blizzard Entertainment, *Starcraft Remastered*, 2019, hämtad: 2019-05-15
<https://starcraft.com/en-us/>
- [17] Ken Schwaber och Jeff Sutherland, *Scrumguiden™*, Scrum-guide, 2013-07-11, hämtad: 2019-02-16
<https://www.scrumguides.org/docs/scrumguide/v1/Scrum-Guide-SE.pdf>
- [18] Google, *A safe place for all your files*, u.å, hämtad: 2019-05-30
<https://www.google.com/drive/>
- [19] Unity Technologies, *Manual: The Multiplayer High Level API*, hämtad: 2019-05-13
<https://docs.unity3d.com/Manual/UNetUsingHLAPI.html>
- [20] CH Robotics, *Introduction*, u.å. hämtad: 2019-05-12
<http://www.chrobotics.com/library/accel-position-velocity>
- [21] Jjules, *Unity, and the Accelerometer vs. the Gyroscope: A Complete Guide*, Unity, 2017-01-17, hämtad: 2019-02-15
<https://forum.unity.com/threads/unity-and-the-accelerometer-vs-the-gyroscope-a-complete-guide.451496/>
- [22] Parametric Technology Corporation, *Develop for free*, 2019, hämtad: 2019-05-15
<https://developer.vuforia.com/vui/pricing>
- [23] Unity Technologies, *Activating Vuforia in Unity*, 2019-04-15, hämtad: 2019-05-15
https://docs.unity3d.com/Manual/vuforia_get_started_project_setup.html
- [24] Mary Keo, *Graphical Style in Video Games*, 2017, hämtad: 2019-05-16 https://www.theseus.fi/bitstream/handle/10024/133067/Keo_Mary.pdf?sequence=1&isAllowed=y
- [25] Macworld, *iPhone vs Android market share*, hämtad: 2019-05-30 <https://www.macworld.co.uk/feature/iphone/iphone-vs-android-market-share-3691861/>
- [26] Friends, *Vårt arbete mot nätmobbing*, 2017, hämtad: 2019-05-15 https://www.dropbox.com/sh/w9pd82tqelkb4o2/AACIWnFZTp5-UXa3FYitEb0ha?dl=0&preview=Friends_natrapport_2017.pdf
- [27] Tilo Hartmann och Christoph Klimmt, *Gender and computer games: Exploring females' dislikes*. Journal of Computer-Mediated Communication, 2006
- [28] Carin Falkner, *Datorspelande som bildning och kultur: en hermeneutisk studie av datorspelande*. Diss, Örebro universitetsbibliotek, 2007

Bilaga A

Personligt bidrag

Per

Utöver utvecklare tilldelades Per rollen som Scrum-mästare. Som Scrum-mästare hade Per ansvar att leda Scrum-möten och sprintmöten under utvecklingsprocessen. Som utvecklare arbetade Per mest med att integrera systemets huvudkomponenter med varandra, med störst fokus på nätverk.

Adam

Adams främsta ansvar under projektet var att ansvara för spellogiken och att tidigt i projektet lösa hur spelets grundidé skulle implementeras. Genom diskussioner med Marcus gavs råd om vilka modeller som behövde göras och Adams ansvar var sedan att tillföra dessa modeller i spelplanen och integrera dem med spellogiken. Adam satt också med att implementera nätverket ihop med spellogiken. Detta innebar att skapa en förståelse för vilka delar i spelplanen som behövde skickas via nätverk och hur det skulle gå till. Han skrev, med hjälp av Per, ett flertal skripts för att få ihop spellogikens integration med nätverket.

Marcus

Tidigt i utvecklingsprocessen tilldelades Marcus rollerna som Produktägare samt Kontaktansvarig. Detta innebar att Marcus hade god kontakt med samtliga i arbetslaget och arbetade med att ta fram lösningar på samtliga områden. Utöver ansvarsområdena arbetade Marcus med modelleringen av de virtuella objekten samt animering.

Isak

Isak arbetade som utvecklare och dokumentationsansvarig under projektets gång. Han utförde de initiala testen av Vuforia som alternativ rörelsespårning till ArUco. Vidare undersökte Isak om rörelsespårningen kunde läsas i höjdled för att simulera ett konstant avstånd till spelplanen. Därefter låg hans fokus på att skapa en dynamisk användardesign för spelmenyn. Som Dokumentationsansvarig ansvarade Isak även över att anteckna under alla större möten och att ha en överblick av vad som planerats för projektet.

Ester

Som utvecklare i projektet arbetade Ester i början av projektet med att förstå nätverkskommunikationen och ta fram en första prototyp av en nätverksuppkoppling. Vidare arbetade hon tillsammans med Marcus för att ta fram ett användargränsnitt till spelet utifrån dess kund- och systemkrav. Hade huvudansvar för design av logo och spelets meny-element.

Samuel

Jobbade med de initiala försöken till att implementera ArUco innan Vuforia valdes som spårningsverktyg. Med Vuforia kopplade Samuel rörelsespårningen med spelet som ett första utkast. Därefter skapade han den slutgiltiga detektionsmarkören i vektorformat och försedde gruppen med den. Samuel hjälpte även Per vid menyfunktionaliteter och vid projektets slutstadie uppdaterade han spelgrafiken för meny-elementen som Ester skapat. Vid ett visst stadio behövdes en större omstrukturering av filhierarki och skript göras vilket Samuel hade största ansvar för. Utöver detta jobbade Samuel även lite med 3D-modellering av sockerbitarna, höll Trello-tavlorna uppdaterade samt integrerade animationer för 3D-modellerna som Marcus skapat.

Bilaga B

Användartester

I slutet av projektiden gjordes ett antal användartester för att samla in åsikter från användare över vad spelet saknar. Detta möjliggjorde för projektgruppen att reflektera kring vissa val av spellogik och hur spelet kan förbättras. Nedan presenteras direkta citat från användare.

- Man skulle kunna ha tre sockerbitar som tvingar en att försvara flera områden men också fler objekt som blockerar myrorna. Storleken är bra på myrorna.
- Råkade klicka på disconnectnär ett lag har vunnit, kanske ha en delay innan det fönstret visas?
- Det finns inget som indikerar att spelet är igång när man startat
- Det är svårt att veta när spelet är igång, kanske ha en ready-knapp så att båda spelare är redo och inne på spelplanen?
- Coolt att man måste zooma in med kameran och följa myror med kameran för att döda dem!
- Healthbars och knappar var ganska små.
- Vore coolt om man kunde byta till att lägga ut hinder för att stoppa fiendes myror.
- Mer höjdskillnader skulle varit ge en mer rolig bana upplevelse!
- Knapparna är lite svåra att läsa och behöver kanske ha någon bakgrund och större text.
- Lite dumt att man kan bara disconnecta från servern när ett lag har vunnit. Man borde kunna gå tillbaka till lobbyn istället.
- Det är svårt att se vilket lag man är med i när man väl kommer in på spelplanen!
- Vore bra att kunna klicka rematch"istället för att behöva skapa ett nytt rum när ett lag har vunnit.
- Vore coolt att ha myror med mer liv men som rör sig längsammare men som skadar mer på sockerbiten. Och att dessa tar fler klick för att döda.
- Någonting som visar vilken sida som är röda lagets och vice versa.
- Ha olika typer av myror! Kanske ha fler objekt att kunna interagera med, t ex torn som skjuter!
- Det skulle vara roligare om det gick längsammare. Sockerbiten borde ha mer liv och myrorna borde kanske röra sig längsammare med. Att man kan spela strategiskt.