Examensarbete

HÖGSKOLAN I SKÖVDE

NAVIGERING FÖR MINIONER I MOBA-SPEL

MINION NAVIGATION IN MOBA GAMES

Examensarbete inom huvudområdet
Informationsteknologi
Grundnivå 30 högskolepoäng
Vårtermin 2023

Rasmus Hallbäck

Handledare: Peter Sjöberg
Examinator: Sanny Syberfeldt

Sammanfattning

Detta arbete beskriver en undersökning som har utvärderat två navigeringstekniker: flockbeteenden och potential fields. I undersökningen har minioner i ett MOBA-scenario simulerat dessa två navigeringstekniker för att avgöra vilken av dem som ger den mjukaste navigeringen. Mjukhet i detta sammanhang innebär att minionerna kontinuerligt ska kunna navigera utan att fastna, samt minimera förändringar av riktningen mellan bilduppdateringar.

En prototyp av ett MOBA-scenario utvecklades för att genomföra två experiment där mätningar togs när minionerna navigerade med de två navigeringsteknikerna. Resultatet visade att algoritmen för flockbeteendena medförde den mjukaste navigeringen. Dock konstaterades det att minionerna fastnade alltför ofta, vilket indikerar att framtida utveckling bör fokusera på att eliminera detta oönskade fenomen.

Nyckelord: navigering, creep, minion, multi-agent, MOBA, spel

Innehållsförteckning

1 Introduktion

Vägplaneringsproblemet existerar i många domäner, men är i datorspel en mycket viktig beståndsdel. Att få en agent att gå från en punkt till en annan i en spelvärld kan åstadkommas med hjälp av algoritmer såsom A*. Dock fungerar A* inte bra i dynamiska världar, som i MOBA-spel, där många agenter konstant flyttar på sig på grund av att de inte får beröra varandra. Därför har algoritmer som använder boids och potential fields för att styra agenter utvecklats. Potential fields använder ett fält med rutor som har olika värden, där en agent tar sig till den ruta som har högst värde. Boids är en algoritm baserad på naturen där agenter härmar hur flockdjur rör på sig tillsammans. Forskning finns inom detta område och en tävlingsinriktad bot har även skapats med hjälp av dessa två tekniker för att utforska sannolikheten för vinst och hur algoritmernas exekveringstid skiljer sig åt. Det finns dock områden inom spel där utseendet för navigeringen hos agenter kan spela roll.

Detta arbete har utgått från att jämföra hur boids och potential fields skiljer sig visuellt från varandra i MOBA-spel. Hur mjuk navigeringen är har undersökts, det vill säga hur snabbt agenterna ändrade sin riktning, men också hur mycket tid de spenderade i lokala optima. Datan av dessa jämfördes sedan mellan implementationerna. För att undersöka minionernas mjukhet har en experimentmiljö utvecklats som mätte minionernas riktningsskillnader, samt hur länge de stod stilla. Minionerna använde antingen potential fields eller flockbeteenden beroende på experimentmiljöns inställningar och navigerade enligt dessa när andra enheter var i närheten. Om ingen annan enhet befann sig i närheten gick de istället mot sitt mål. Implementationen för flockbeteendena inkluderade ett enkelt separationsbeteende som höll minionerna på avstånd från varandra. Ett annat styrbeteende implementerades också, som gjorde att minionerna svängde mot den del av flocken som hade färre enheter. I implementationen för potential fields genererades ett fält som hade en attraktionskraft vid punkten som en minion skulle följa. Kraften var som starkast vid mittpunkten och försvagades ju längre bort från mitten den var. Dessutom fanns det bortstötande kraftfält vid varje enhet för att hålla minionerna borta från varandra. Flera omgångar kördes där mätningar togs vid varje omgång, men var separata då resultaten var relaterade till antalet minioner i världen. Efter avslutad simulering sparades datan ned i en textfil.

Därefter skapades grafer för skillnaderna i riktning samt för tiden som minionerna tillbringade i lokala optima. Det framkom att riktningsskillnaderna var högre vid färre antal minioner för båda algoritmerna. Förklaringen låg i att olika formationer bildades när minionerna navigerade, i form av en rak linje eller en triangel. Detta resuluterade i att minioner som kom i närhet av formationerna navigerade jämns med raka kanter och spenderade därför mycket tid åt att gå rakt vid fler antal minioner. Vid färre antal minioner tillbringades en större del av tiden åt att cirkulera en enda minion, vilket resulterade i högre medelvärden för riktningsskillnaderna. Tiden spenderat i lokala optima ökade även proportionellt mot antalet minioner, vilket skulle kunna ha berott på att minionerna fastnade i mitten av triangelformationerna. Grafen för lokala optima ökade exponentiellt för potential fields medan den ökade linjärt för flockbeteendena. Generellt sett var båda graferna lägre för implementationen av flockbeteendena, vilket gjorde denna algoritm mjukare. Trots detta fanns fortfarande problemet med lokala optima, och slutsatsen var att framtida utveckling behövs för att bemöta detta problem eftersom minionerna inte bör fastna i ett verkligt spelscenario.

2 Bakgrund

I denna sektion diskuteras användningsområdet för vägplanering samt dess utveckling i dynamiska situationer. Därefter presenteras två olika algoritmer som kan användas för dynamisk vägplanering, följt av en introduktion till MOBA-spel och dess minioner, vilket är fokus för detta arbete. Avslutningsvis tar kapitlet även upp de artiklar som är relaterade till denna undersökning.

2.1 Vägplanering i datorspel

Vägplanering används inom många områden, som till exempel för robotar, GPS-system och artificiell intelligens. Men det är också grundläggande inom spel. Att få en agent i ett spel att nå en viss position, givet den kortaste vägen, är en viktig del för spel. Ett exempel på en sådan algoritm är A*, som infördes av Hart, Nilsson och Raphael år 1967, och är en modifierad version av Dijkstras algoritm för vägplanering.

A* använder en heuristisk funktion för att skapa en uppskattning av avståndet från en given nod till målet. Detta uppskattade värde används sedan för att beräkna den kortaste vägen till målet. Genom att använda en uppskattning av avståndet kan A* hitta den kortaste vägen till målet snabbare än Dijkstras algoritm. Därför har fokuset för forskningen inom vägplanering primärt varit på A* (Sabri, Radzi & Samah 2018).

2.2 Dynamiska världar

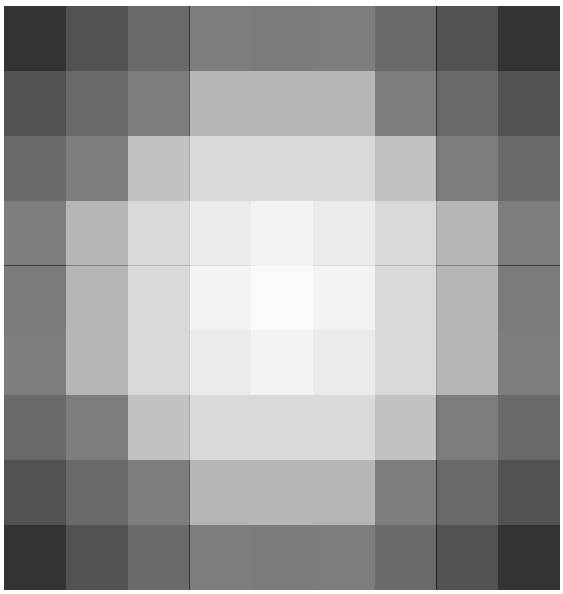
A* är inte särskilt bra när det kommer till mer dynamiska världar. Om ett objekt rör sig fram och blockerar den genererade vägen, kommer vägen därför att behöva genereras om på nytt (Hagelbäck 2012). I spelet StarCraft (1998) finns det enheter som behöver röra sig dynamiskt och kunna omringa sin fiende. Därför föreslog Hagelbäck (2016) att man implementerar en hybridlösning för enheter i StarCraft som använder A* men också potential fields eller boids. En enhet börjar med att använda A*, men byter sedan till potential fields eller boids när den kommer i närheten av en fiende. Förutom att den presterar bättre ur ett perspektiv av tidseffektivitet, kan nu enheter även omringa en fiende utan att krocka med varandra.

2.2.1 Potential Fields

Potential Fields fungerar som så att olika objekt eller punkter i spelvärlden har krafter genererade i ett rutnät runt om sig, vilka antingen kan vara attraherande eller bortstötande (Hagelbäck 2012). Dessa fält kan ha olika storlekar och se olika ut beroende på vilket beteende man vill åstadkomma. Rutor med positiva värden har en attraherande kraft. Negativa rutor har bortstötande krafter, medan rutor som har värdet noll inte har någon påverkan alls (Hagelbäck 2016).

Om man vill att en enhet ska röra sig mot en viss position, placerar man en attraherande kraft vid den punkten. Denna kraft bör täcka hela spelvärlden eftersom man vill att den ska påverka enheten var den än befinner sig. Vid mittpunkten av det attraherande fältet kommer värdet att vara som högst, och kommer att gå neråt tills den når noll vid fältets kant. Det vill säga, vid det avstånd man vill att kraften ska täcka (Hagelbäck 2012). Figur 1 visar ett fält för en attraherande kraft runt en punkt. Fälten som genereras kan sedan sättas ihop till ett stort fält som enheterna kan använda för att navigera igenom. För att navigera kollar enheterna på

några av rutorna runt om sig själv, för att sedan gå mot den ruta som är mest attraherande, det vill säga den ruta som har högst värde (Hagelbäck 2016).



Figur 1 En potential field som är attraherande. Punkten i mitten är positionen där enheter som följer fältet ska röra sig mot. Ju ljusare färgen är desto högre värde har rutan.

2.2.2 Flockbeteende

”Boids” kallas även för ”flockbeteenden” och är en modell tagen från naturen, som beskriver hur vissa djur, såsom fiskar, fåglar eller får, rör på sig i en grupp. När många djur rör sig tillsammans i en grupp, kan det se ut som om de alla är en enda varelse, där ingen utav djuren är en ledare utan varje individ hjälper till att styra hela gruppen (Jae, Se & Rafael 2009).

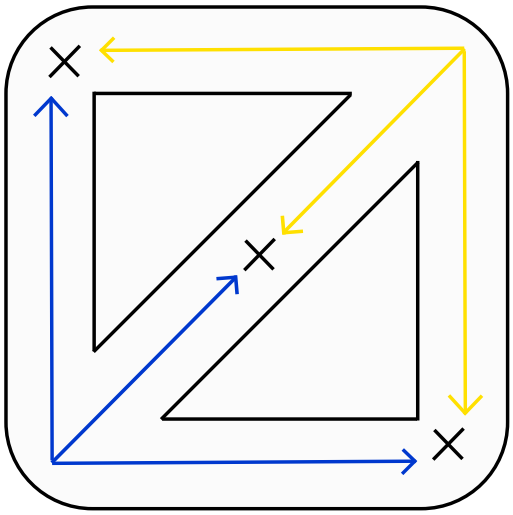
Flockbeteende-algoritmen introducerades först för att återskapa djurens beteenden på en dator (Reynolds 1987). Algoritmen fungerar så att vissa styrbeteenden skapar en kraft i form av en vektor för varje enhet i gruppen. Varje kraft har en styrka som bestämmer hur stor påverkan den har för enhetens rörelse. Eftersom krafterna är viktade baserat på deras styrka, går det att skapa en gemensam vektor som har ett medelvärde för alla krafter kombinerat. Enheterna använder sedan denna slutgiltiga vektor som sin riktning. Det är sedan upp till implementationen vilka krafter en enhet ska använda (Reynolds 1987).

Ett av beteendena som kan användas för en bot i spelet StarCraft (1998) är sammanhållning, vilket är en kraft som pekar mot mittpunkten av en grupp. Denna kraft gör att enheterna kommer att dra sig mot varandra och försöka hålla sig tillsammans. Ett annat viktigt beteende är separation. Detta är en kraft som motverkar kollision med sina egna eller fiendens enheter, vilket gör att de håller sig på avstånd från varandra (Hagelbäck 2016).

2.3 Multiplayer Online Battle Arena

I multiplayer online battle arena (MOBA)-spel som League of Legends (2009) finns det en viss typ av enhet som kallas för minioner. Dessa är datorstyrda med hjälp av artificiell intelligens (League of Legends Wiki 2023a). De här enheterna kallas även för creeps (League of Legends Wiki 2022). När en minion skickas ut i spelvärlden går den från sin egen bas till

fiendens bas, som också skickar ut minioner. När de båda lagens minioner möts på vägen börjar de attackera varandra, se figur 2 (League of Legends Wiki 2023a).



Figur 2 Den ena basen befinner sig i hörnet längst ner till vänster. Den andra basen befinner sig på den motsatta sidan. Minioner skickas ut från baserna och vandrar in i korridorerna längs pilarna. De möts sedan i mitten vid kryssen och börjar där att attackera varandra.

På grund av att dessa minioner är dynamiska agenter räcker det därför inte att endast använda A* eftersom algoritmen inte presterar särskilt bra i ett realtidssystem som detta (Nguyen, T., Nguyen, K. & Thawonmas 2013). Eftersom minionerna påminner mycket om enheterna i StarCraft kan en hybridlösning användas även här. Tack vare att minionerna alltid tar samma väg i korridorerna kommer hybridlösningen inte behöva använda sig utav A* alls. Det räcker med att använda en förutbestämd väg skapad för varje korridor (League of Legends Boards 2016).

2.4 Relaterad forskning

Denna sektion diskuterar den forskning som ligger till grund för detta arbete. Fokus läggs på utseendet och förekomsten av lokala optima i spel. Den forskning som arbetet har utgått ifrån presenteras också här.

2.4.1 Vinst och prestanda

Hagelbäck (2016) jämförde sannolikheten för vinst mellan StarCrafts inbyggda AI med potential fields eller boids. Resultaten visade att båda alternativen presterade likvärdigt mot StarCrafts AI. Potential fields presterade dock bättre än boids när de fick möta varandra. Denna skillnad är ett argument för att potential fields är mer anpassningsbar eftersom de individuella rutorna i fältet kan manipuleras. Å andra sidan var exekveringstiden för potential fields hundra gånger sämre än för boids, vilket kan vara ett argument för att välja boids istället (Hagelbäck 2016).

2.4.2 Utseende

Dolui och Hancox (2021) diskuterade att valet av teknik för deras monster i spelet Wolcen (2020) skulle tagit hänsyn till artistisk effekt. Monstren måste se bra ut när de rör på sig. Här användes context steering för att uppnå ett bra resultat, och de ansåg att det såg bättre

ut än vad boids gör i deras spel. Leigh, Louis och Miles (2007) talade också om utseende. De presenterade en modifierad implementation av vägplaneringsalgoritmen A* med en genetisk algoritm som användes för att styra en båt i vatten. Eftersom A* skapade den absolut närmsta vägen såg den inte så realistisk ut i ett scenario som detta, eftersom båtar inte sitter tätt intill när de åker förbi varandra i verkligheten. Denna modifierade algoritm skapade därför vägar som såg mer ut som den väg en människa hade tagit, vilket gjorde vägarna mer realistiska.

2.4.3 Lokala optima

I komplexa miljöer som i StarCraft (1998) kunde agenter som navigerade genom potential fields fastna. Detta skedde på grund av att det högsta värdet runt enheten (vilket är det värde den vill nå) kunde vara rutan som den redan stod på. Detta ledde till att enheten inte kunde ta sig framåt och stannade därför kvar på samma ruta. För att motverka att de fastnade, användes A* som ett komplement när enheterna navigerade genom kartan, medan potential fields användes när de skulle omringa en fiende (Hagelbäck 2012).

3 Problemformulering

Hagelbäcks (2016) hybridimplementationer med potential fields och boids användes för att skapa en bot till StarCraft (1998). Därför kan man anta att utseendet på hur enheterna rör sig inte spelade någon roll, eftersom den algoritm som har högst sannolikhet för vinst var den som var bäst. Algoritmerna kan dock även användas i MOBA-spel för rörelsen av minioner. Eftersom detta är en faktisk mekanik i ett spel av denna genre och inte en bot är det viktigare i denna situation att se till att de navigerar på ett sätt som ser mjukt ut. Mjukhet i denna kontext betyder att riktningsskillnaden mellan varje bildruta för minionerna bör vara minimal och inte ändras för hastigt, så att navigeringen ser mer naturlig ut. Att en minion hastigt byter riktning kan få dess väg att se skarpt krokig ut. Mjukheten innebär även att de kontinuerligt ska navigera utan att fastna i ett lokalt optima.

Leigh, Louis och Miles (2007) artikel visar hur de gjorde för att skapa säkrare vägar för en båt som användes i ett RTS-spel genom att ta en mer människoliknande väg, som därför såg mer naturlig ut. Ett resultat av detta blev att vägen såg mjukare ut när den åkte förbi andra båtar, även om detta inte var tanken. Att utseendet i realtidsspel ska se bra ut kan bekräftas av Dolui och Hancox (2021), som hade utseendet i åtanke när de skapade sina monster i Wolcen (2020). De ville att deras navigering skulle flöda fram mjukt runt hinder i spelvärlden och se naturlig ut. De föreslog först att använda en teknik som heter Optimal Reciporal Collision Avoidance, men en av anledningarna till att den inte användes var för att navigeringen inte var artistiskt tilltalande. Det vill säga såg den inte ut att ha ett naturligt flöde. Trots att spelet inte är en MOBA, utan en RTS, är det dock viktigt att förstå att deras monster också är en form av enhet som är en faktisk entitet inbyggd i spelet, och inte en bot. MOBA-spel är också ett typ av spel som simuleras i realtid, vilket gör att de påminner om varandra. Det finns därför en strävan efter att ha en mjuk navigering i spel som dessa och därför kan antagandet tas även här att minionernas navigering i MOBA-spel borde se mjuk ut.

MOBA-spelens miljöer är dessutom enkla för minionerna att navigera igenom och problemet med lokala optima borde därför inte uppstå. Skulle en minion fastna påverkar det spelets mekanik negativt och det ser inte bra ut. När minionerna inte attackerar något ska de alltid vara i rörelse och bör därför inte stå stilla över huvud taget. Det är alltså när de navigerar som de bör röra på sig mjukt.

Eftersom resurser har lagts ned på att bevisa vilken teknik som har högst sannolikhet för vinst eller vilken som presterar bättre när det kommer till exekveringstid, skulle det vara intressant att veta vilken av de två som medför en mjukare navigering för just MOBA-spel, eftersom de båda teknikerna fungerar att använda till en dynamisk navigering för minioner. Men då algoritmerna skiljer sig i hur de väljer riktningen de ska navigera mot, är det intressant att veta vad skillnaden är. Frågeställningen är därför:

Vilken av teknikerna, potential fields och boids, medför en mjukare navigering för minioner i ett MOBA-scenario?

3.1 Metodbeskrivning

För att kunna undersöka vilken av implementationerna som medför en mjukare väg i ett MOBA-scenario har ett kvantitativt experiment utförts. Experimentet har undersökt vad en

implementation av potential fields medför för skillnad i riktningsändring för minionerna mellan varje bilduppdatering, samt hur mycket tid de spenderar i ett lokalt optima jämfört med en implementation av boids.

Minionernas beteende hade två olika hybridlösningar som antingen använde potential fields eller boids (Hagelbäck 2016). Minionerna använde boids eller potential fields när de behövde undvika en annan minion. När de inte längre behövde undvika något ändrades deras beteende till att följa en manuellt skapad väg som förde dem framåt i korridoren mot fiendens bas. Detta fungerade därför inte exakt som Hagelbäcks (2016) lösning som använde A*, men eftersom vägen minionerna gick efter alltid är densamma, var en algoritm för grafsökning onödig och denna väg kunde därför skapas i förväg och var konstant för hela experimentet.

Under tiden minionerna navigerade mätte programmet deras skillnader i riktning mellan varje bilduppdatering. Det var viktigt att bara mäta när de navigerade, eftersom riktningskillnaden alltid skulle varit noll under tiden de attackerade, vilket är ointressant. Ett annat problem skulle kunna ha uppstått om mätningar togs när de bytte vilken fiendeminion de skulle attackera. Om fiendeminionen till exempel befann sig bakom skulle riktningen ändras mycket hastigt under den bilduppdateringen. Dock var detta ett normalt beteende och mätvärden togs därför inte i detta tillfälle. Detta tog bort eventuella extremvärden.

Lokala optima var också något som togs hänsyn till, eftersom det också kunde vara något som leder till att riktningen fortsätter vara densamma medan minionen hade fastnat. Därför togs inga mätvärden för riktningsskillnader när en minion stod på samma position som den nyss stod på. Istället togs nya mätvärden som mätte hur länge den stod på denna position, vilket i sin tur besvarar hur länge minionerna stod stilla. Detta kunde endast utföras under tiden den navigerade eftersom de naturligt stod stilla när de attackerade.

I MOBA-spel finns det även hjältar som är styrda av människor. Eftersom människor inte kan återskapa samma beteende skulle resultatet blivit annorlunda vid varje omgång då minionerna även här hade varit tvungna att undvika hjältarna. Torn som också är viktiga objekt i MOBA-spel, implementerades eftersom de inte var styrda av människor. De var statiska objekt som alltid betedde sig likadant och attackerade den närmsta fiendeminionen som var inom sin räckvidd. Detta gjorde att resultatet alltid förblev detsamma även om en omgång skulle köras om.

Experimentet simulerade de båda implementationerna separat. Omgångar med olika antal minioner kördes för att få mätvärden när antalet medlemmar i minionflocken var olika. Detta var viktigt eftersom antalet minioner inte alltid är detsamma vid en viss tidpunkt i en riktig match. Därför kördes flera omgångar som sträckte sig från att simulera en minion till tjugo minioner. Dessutom skiljer potential fields och boids exekveringstid mot varandra (Hagelbäck 2016). Om man därför skulle ha låtit simuleringen köra så snabbt den kunde, hade en av implementationernas skillnader i riktning blivit mindre endast för att den var snabbare än den andra. Det beror på att skillnaden i tid mellan bilduppdateringarna (delta-tid) skiljde sig åt mellan algoritmerna. I ett vanligt scenario hade delta-tiden tagits i hänsyn i uträkningarna för varje minions riktning och position. Detta skulle dock göra att mätresultaten hade fått för mycket inflytande från exekveringstiden. För att motverka detta fick de båda implementationerna exekvera med en konstant delta-tid som om spelet körde i 30 bilduppdateringar per sekund. Detta är dessutom en viktig del i onlinespel som MOBA-spel, då även de har en konstant uppdateringsfrekvens(League of Legends Wiki 2023b).

Varje omgång kördes i 9000 uppdateringar, vilket motsvarade fem minuter i scenariots simuleringstid.

3.1.1 Metoddiskussion

Eftersom de värden som experimentet mätte var siffermässiga när man pratar om skillnad i riktning och hur länge en minion stod stilla, var det ett kvantitativt experiment som utfördes (Borg & Westerlund 2014). En viktig del i metoden var att experimentet skulle ha en hög reliabilitet, vilket betyder att resultatet hade förblitt densamma varje gång experimentet skulle utföras med samma parametrar (Eliasson 2019). Eftersom det som jämfördes var hur boids och potential fields, samt hur olika antal minioner påverkade resultatet, togs andra bakomliggande variabler bort eftersom de skulle kunna störa resultatet (Eliasson 2019). Påverkan av människor och exekveringstid är exempel på variabler som skulle kunna ha förändrat experimentets resultat. Om reliabiliteten blir högre genom att oönskade variabler tas bort, kommer även validiteten att bli högre. Det innebär att det som ska mätas har högre sannolikhet att vara det som faktiskt ska mätas. (Eliasson 2019).

För att vidare öka validiteten var metoden specifik eftersom den beskriver exakt när värdena fick mätas. Detta gjordes för att undvika att inkludera mätvärden som var ointressanta för resultatet. Till exempel skulle mätning av data när en minion attackerade vara vilseledande eftersom minionen var statisk i detta tillstånd. Detta skulle ge extremvärden som skulle kunna ha påverkat slutsatserna.

Ett problem med denna metod var dock att resultatet inte blev subjektivt. Eftersom undersökningens fråga handlade om utseendet för minionernas rörelse, återspeglar resultatet inte nödvändigtvis hur riktiga spelare skulle föredra att minionerna rör på sig. En annan kvalitativ undersökning skulle kunna ha genomföras där människor skulle få titta på hur minionerna rör sig i olika scenarion. Därefter skulle de kunna delta i en intervju där de besvarar vilken teknik de tyckte såg bäst ut och sedan beskriva varför någon av dem såg bättre eller sämre ut än den andra. Ett alternativ skulle vara att utföra en annan kvantitativ undersökning som använder ett frågeformulär där människor hade fått svara på vilken teknik de föredragit (Williamson 2002).

3.1.2 Analys

Efter att experimentet genomfördes fanns det mätvärden som användes för att skapa ett linjediagram som visade vad medelvärdet av riktningsändringen var totalt för minionerna. Ett annat linjediagram skapades också som visar hur länge de stod stilla totalt beroende på implementation och minionantal. Den linje som var lägst kommer därför att vara den som hade mjukast väg. Vilken algoritm som var bättre när det kommer till mjukhet utifrån graferna analyserades. Det var inte säkert att en av implementationerna var bäst hela tiden eller att skillnaden i riktning ens var densamma vid varje minionantal. Resultaten kunde även därför beskriva vilket minionantal som var optimalt.

4 Implementation

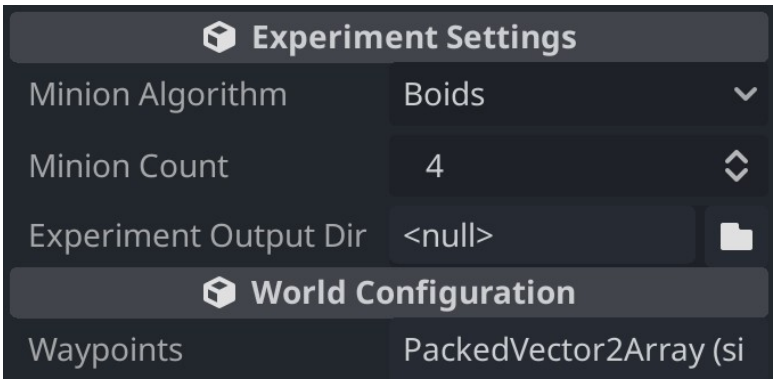
Detta avsnitt presenterar hur arbetet utvecklades innan experimentets data samlades in. Med andra ord beskriver detta kapitel uppbyggnaden av experimentmiljön, de val som gjordes för denna, samt de utmaningar som uppstod. Inställningarna för algoritmernas vikter och krafter finns dokumenterade i Appendix A. Slutligen inkluderas även ett pilottest i avsnittet, vilket utfördes för att testa experimentmiljöns funktionalitet.

4.1 Experimentmiljö

Själva experimentmiljön utvecklades i spelmotorn Godot 4 med hjälp av dess inbyggda programmeringsspråk GDScript (Godot 2007). Valet av Godot grundades på dess enkla installationsprocess och kompatibilitet med olika operativsystem. Spelmotorn i sin helhet, inklusive programmeringsspråket, är väldokumenterad, vilket gjorde den till en tillfredsställande valmöjlighet. GDScript är även ett användarvänligt programmeringsspråk som liknar Python och, även om det innebär att exekveringstiden är något längre, så passade det utmärkt för snabb prototypning, vilket var mer relevant för denna undersökning än exekveringstidens hastighet.

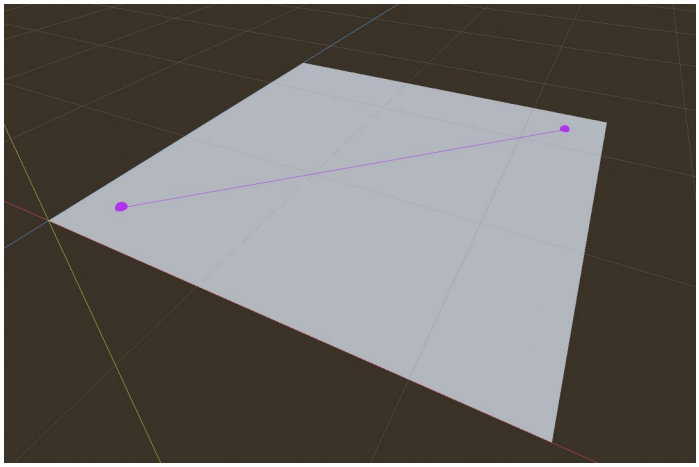
Eftersom simuleringen inte krävde en fysikmotor eller någon grafik, kunde den ha skapats utan en spelmotor. Grafikmotorn underlättade dock utvecklingen av prototypen genom att möjliggöra att implementationerna fungerade. Detta förbättrade tillförlitligheten i de utförda mätningarna.

I Godot är det möjligt att skapa variabler för objekt som blir synliga i redigeraren och därmed ändringsbara (Godot Docs 2014a). Projektet för detta experiment hade därför inställningar för simuleringen som var lätta att ändra och komma åt. Den första variabeln möjliggjorde valet av den algoritm som skulle användas i simuleringen: antingen "Boids" eller "Potential Fields". Den andra variabeln tillät anpassning av det maximala antalet minioner som skulle skapas för varje lag i simuleringen. Vid körningen började simuleringen med en omgång med en enda minion och därefter genomfördes så många omgångar som behövdes för att nå det specificerade antalet minioner. Eftersom metodbeskrivningen krävde att det maximalt fick finnas tjugo minioner, var det därför inte möjligt att sätta denna variabel till ett värde över detta. Den tredje variabeln gjorde det möjligt att ange destinationsmappen där resultatfilen från simuleringen skulle sparas när den var avslutad. Den fjärde och sista variabeln var en lista som innehöll varje vägpunkt som minionerna skulle följa. Eftersom fiendelaget använde samma lista, men i omvänd ordning, krävdes endast en lista. Figur 3 illustrerar hur inställningsgränssnittet såg ut.



Figur 3 Inställningsgränssnittet för simuleringsexperimentet.

Experimentet hade också inställningar som inte kunde ändras via inställningsgränssnittet. Dessa inkluderade antalet tidssteg som simulerades under varje omgång innan den avslutades samt den simulerade delta-tiden som användes av alla rörliga objekt i världen. Dessa värden var konstanter och ändrades inte mellan omgångarna. För att underlätta placeringen av vägpunkter skapades ett särskilt skript som kunde köras i redigeraren. Ett sådant skript (som kallas för "tool" i Godot) kan inte exekveras under spelets gång (Godot Docs 2014b). Målet med detta skript var att markera vägen genom att rita en linje mellan varje vägpunkt och placera en sfär vid vägpunkternas positioner. Detta tillförde ingen funktion till själva simuleringen, men det gjorde det enkelt att visuellt se att vägpunkterna var placerade korrekt. Eftersom ett av lagen använde samma lista men omvänd, var det viktigt att vägen var symmetrisk eftersom vägen borde sett likadan ut oberoende av vilken riktning den användes ifrån. Detta ledde till att de båda lagen möttes precis i mitten av kartan, vilket var det önskade resultatet. Figur 4 visar hur utritningen såg ut.



Figur 4 Utritning av två vägpunkter. En linje ritas ut mellan varje vägpunkt och sfärer ritas vid vägpunkternas individuella position.

4.2 Enheterna

Minionerna implementerades först. De hade en referens till listan med vägpunkter som de skulle följa, samt en indexvariabel som pekade på den vägpunkt som minionen skulle följa vid en given tidpunkt. När en minion närmade sig en vägpunkt ökade indexet med ett, och den rörde sig mot nästa vägpunkt i listan enligt indexet. Undantaget var när den redan förflyttade sig mot den sista vägpunkten i listan. Då ökade inte indexet, eftersom det inte fanns fler vägpunkter att följa. Det var viktigt att de inte nödvändigtvis förflyttade sig exakt

till den position där vägpunkten fanns. Det var tillräckligt att vara inom en viss radie från vägpunkten (League of Legends Boards 2016). Detta gjordes för att undvika att en minion skulle fastna runt en vägpunkt om den låg mitt i en grupp av andra minioner, vilket ledde till att minionen kunde fortsätta sin navigering.

I början hade minionerna en funktion som gjorde det möjligt att skada en annan enhet. Denna funktion användes av projektilerna, vilka var enkla objekt som följde fiender och utförde minionens skadefunktion när de träffade sitt mål. Efter träffen förstördes projektilen. Om minionens hälsa sjönk till noll eller lägre, förstördes själva minionen. Projektiler avfyrades när minionerna attackerade andra enheter i världen. Ett problem uppstod dock då detta system endast fungerade på andra minioner. Detta innebar att tornen inte kunde bli attackerade av minionerna när dessa senare implementerades. En potentiell lösning skulle ha varit att införa två olika typer av projektiler, där endast en av dem orsakade skada. Detta skulle dock ha komplicerat koden, eftersom minionerna då skulle behöva avgöra vilken typ av enhet de skulle attackera för att välja rätt projektil. Istället implementerades en basklass som delades av både tornen och minionerna. Skadefunktionen flyttades från minionen till denna gemensamma basklass. Där inkluderades även variabler som båda typerna av enheter behövde, såsom enhetens fiende, hälsa, attackhastighet, attackskada, räckvidd och vilket lag den tillhörde. Detta underlättade dessutom för en minion att söka efter olika enhetstyper i sin närhet genom att använda en enda funktion. Funktionen returnerade bastypen för den närmsta enheten och kunde sedan exempelvis användas som en fiende.

Tornens uppgift var att leta efter fiender inom sin omgivande räckvidd och sedan attackera den närmaste fienden. Om ingen fiende fanns inom tornets räckvidd, förblev det passivt. Minionernas funktioner var mer komplexa jämfört med tornens eftersom de även hade navigeringsförmåga. Därför användes en enkel tillståndsmaskin för att hantera minionernas beteende. Tillståndsmaskinen hade två lägen: attack och navigering. Navigeringsläget användes när en minion skulle följa en vägpunkt eller en fiende. Om en fiende var målet rörde sig minionen mot denna. Annars rörde den sig mot sin angivna vägpunkt. Minionen fortsatte att navigera tills den var i närheten av sin fiende, då den bytte till attackläge och började skjuta mot denna. Om fienden inte längre fanns eller befann sig utanför sin räckvidd, återgick minionen till navigeringsläget.

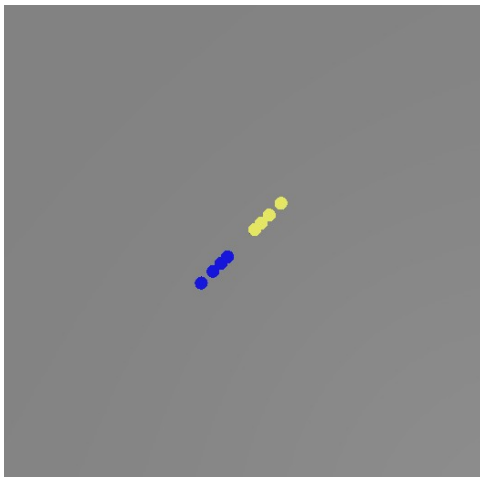
Navigeringsläget övervakade också kontinuerligt om det fanns enheter i närheten. Om ingen enhet upptäcktes fortsatte minionen att gå mot sin angivna vägpunkt eller fiende. Om en enhet däremot befann sig i närheten, användes antingen flockbeteendena eller potential fields beroende på experimentets inställningar.

Alla enheter tilldelades även visuella modeller, vilka, likt representationen av vägpunkterna, underlättade för att försäkra sig om att algoritmerna fungerade korrekt. Minionerna hade korta cylindrar som ändrade färg beroende på vilket lag de tillhörde. Tornen hade större cylindrar, medan projektilerna var utformade som sfärer.

4.3 Flockbeteende

Vid flockbeteendenas implementering skapades först en vektor som pekade mot minionens aktuella vägpunkt eller fiende. Detta är algoritmens basvektor och utgjorde dess grund, eftersom den representerade den riktning minionen önskade röra sig i. Detta var ett målsökande styrbeteende och vikten på denna kraft var svag (Hagelbäck 2016). Alla vikter

för flockbeteendenas styrbeteenden är dokumenterade i Appendix A. Det andra implementerade styrbeteendet var separation, vilket är ett av de grundläggande styrbeteendena (Reynolds 1987). Syftet med detta styrbeteende var att hålla minionerna på avstånd från varandra. För varje enhet i närheten genererades en vektor som pekade bort från den andra enheten. Vektorn normaliserades sedan. Vikten för denna kraft varierades beroende på avståndet till enheten. Ju närmare desto starkare kraft. Detta uppnåddes genom att ta ett värde och dividera det med avståndet från minionen till enheten (Kawabayashi & Chen 2008). Denna vektor adderades sedan till grundvektorn. Efter att alla viktade vektorer hade adderats till grundvektorn normaliserades den och användes som riktning för navigeringen. Att endast använda dessa två styrbeteendena fungerade dock inte, eftersom minionerna på grund av detta radades upp och fastnade i detta läge. Se figur 5.



Figur 5 Uppradade minioner som enbart är utrustade med målsöknings- och separationsbeteenden.

Detta inträffade eftersom riktningen till samtliga enheter var antingen direkt framåt eller direkt bakåt i förhållande till minionen, medan målsökningsriktningen också var precis framåt. Detta resulterade i att den sammansatta riktningen inte resulterade i en vinkel bort från gruppen.

Ett nytt styrbeteende utvecklades som möjliggjorde svängar åt höger eller vänster i förhållande till minionens nuvarande riktning. För varje omgivande enhet kontrollerade minionen var enheten befann sig i relation till sin nuvarande riktning mot vägpunkten eller fienden. Om enheten låg på höger sida lades en kraft till grundvektorn som pekade åt vänster i förhållande till minionens riktning. Om enheten i stället låg på vänster sida, lades en kraft till som pekade åt höger. Detta innebar att minionen strävade efter att svänga åt det håll där det fanns färre enheter. Resultatet blev att minionerna inte längre radade upp sig i en rak linje utan spred istället ut sig.

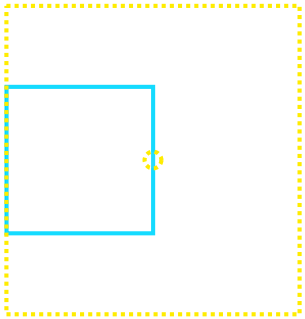
4.4 Potential Fields

En fältklass implementerades med en array som motsvarade fältet, där varje värde i arrayen representerade en ruta. Syftet med klassen var att förse funktioner som kunde modifiera fältets värden. Två funktioner skapades för att generera fälten: en attraherande och en bortstötande. Dessa fungerade på liknande sätt, med skillnaden att det attraherande fältet hade högst värde i det område som en minion skulle dras mot, medan det bortstötande fältet hade lägst värde i denna punkt. Därefter interpolerades värdena mellan kanten och

mittpunkten för att skapa ett fält med utseendet som visas i Figur 1. Inspiration för detta koncept hämtades från definitionerna av attraherande och bortstötande krafter av János och Matijevics (2010), där riktningsvektorer vid varje ruta användes för navigering. Även om deras metod involverade vektorer, gav den ändå insikt till hur potential fields fungerar (Hagelbäck 2016).

För att skapa ett attraherande fält valdes en punkt i ett tomt fält som fungerade som mittpunkt, belägen i mitten av fältet. Därefter valdes en radie för att definiera fältets utsträckning. Radiens värde sattes till hälften av fältets totala storlek så att den kunde täcka hela världen (Hagelbäck 2016). En maxstyrka angavs också för att bestämma kraftens intensitet vid mittpunkten(krafterna och storlekarna på fälten är dokumenterade i Appendix A). Sedan beräknades avståndet från varje ruta i fältet till mittpunkten (János & Matijevics 2010). Detta avstånd dividerades med radien för att skapa ett förhållande mellan noll och ett. Innan detta förhållande användes för att multiplicera med maxstyrkan, justerades det eftersom förhållandet nu var omvänd. När avståndet var kort var förhållandet nära noll, och när avståndet var långt var förhållandet nära ett. För att korrigera detta subtraherades förhållandet från ett. Det resulterande värdet multiplicerades sedan med maxstyrkan och sattes på motsvarande ruta i fältet. Den bortstötande kraften fungerade på liknande sätt, med skillnaden att förhållandet inte behövde justeras eftersom värdet skulle vara som lägst i mitten.

I simuleringen hade varje minion sitt eget världsfält som täckte hela spelvärlden. Det innebar att det attraherande fältet som genererades behövde ha en längre sida för att täcka hela världen om mittpunkten av fältet låg vid kanten av världen, som illustreras i figur 6. Medan sidan av den bortstötande kraften var mycket mindre eftersom dess radie var lika lång som en minions radie. Både det attraherande och det bortstötande fälten behövde endast genereras en gång. Vid varje uppdatering skapade varje minion sitt eget fält och placerade det attraherande fältet i sitt världsfält så att mittpunkten hamnade där minionen önskade röra sig. Sedan placerades en bortstötande kraft på minionens individuella världsfält vid varje enhet i världen, återigen med mittpunkten där enhetens position befann sig.



Figur 6 Ett attraherande fält med streckad linje där mittpunkten alltid nuddar världsfältet i heldragen linje eftersom världsfältets sida är hälften så lång.

Det sista som beräknades var vilken ruta en minion skulle röra sig mot. Minionen undersökte alla åtta rutor runt omkring sig och valde den ruta som hade högst värde. Därefter beräknades en vektor som pekade mot den valda rutan och användes för navigering.

I motsats till implementationen för flockbeteendena spred minionerna ut sig direkt efter detta steg, och ingen ytterligare modifiering av implementationen var nödvändig.

4.5 Datainsamling

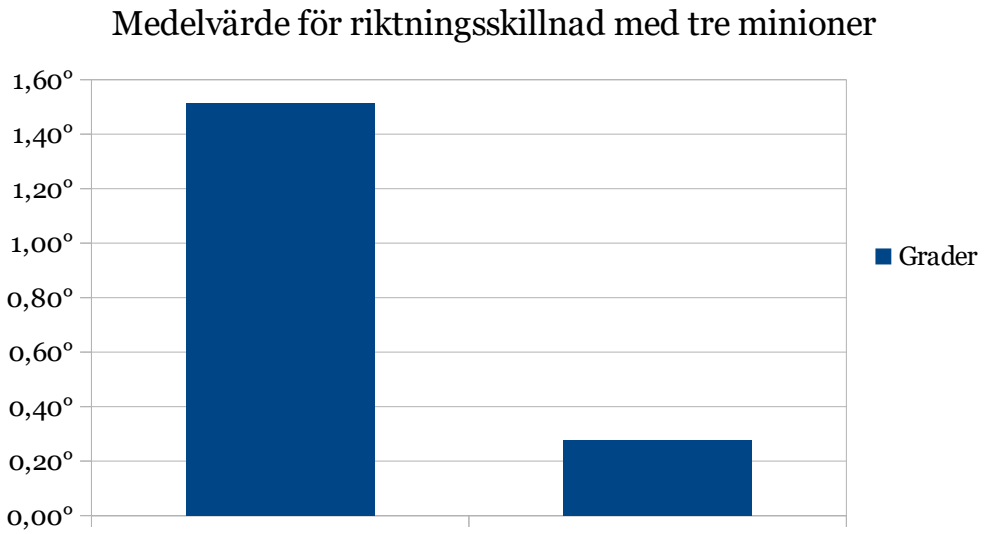
Vid varje omgång sparade alla minioner sin riktningsskillnad i en global lista vid varje uppdatering. Riktningsskillnaden beräknades genom att jämföra vinkeln från minionens föregående uppdatering och med dess nuvarande vinkel. Därefter summerades alla värden i listan och dividerades sedan med antalet riktningsskillnader i listan för att få det genomsnittliga värdet. Det resulterande talet representerade medelvärdet av riktningsskillnad för alla minioner. Minionerna sparade endast ned sina riktningsskillnader i listan när de navigerade med antingen flockbeteenden eller potential fields, enligt metodbeskrivningen. Dessutom konverterades medelvärdet till grader istället för radianer för att få en mer vardaglig mätning.

För att utvärdera förekomsten av lokala optima hade varje minion en individuell lista som höll reda på hur många gånger de hade besökt en specifik position. Om en minion hade besökt en viss position mer än en gång, ökade en global variabel med ett för att indikera att minionen hade stått still på en position under ett tidssteg. Samtidigt sattes även en boolesk variabel till sant för att ange att minionen nyligen hade varit i ett lokalt optima. Denna mekanism implementerades eftersom riktningsskillnader inte skulle sparas ned när detta inträffade. En annan faktor som beaktades var att undvika mätning när en minion bytte den enhet som den skulle attackera. Därför fanns en ytterligare boolesk variabel som sattes till sant när en minion bytte fiende. Om en minion nyligen var fri från lokala optima och inte nyligen hade bytt fiende, så lades dess riktningsskillnad till i listan.

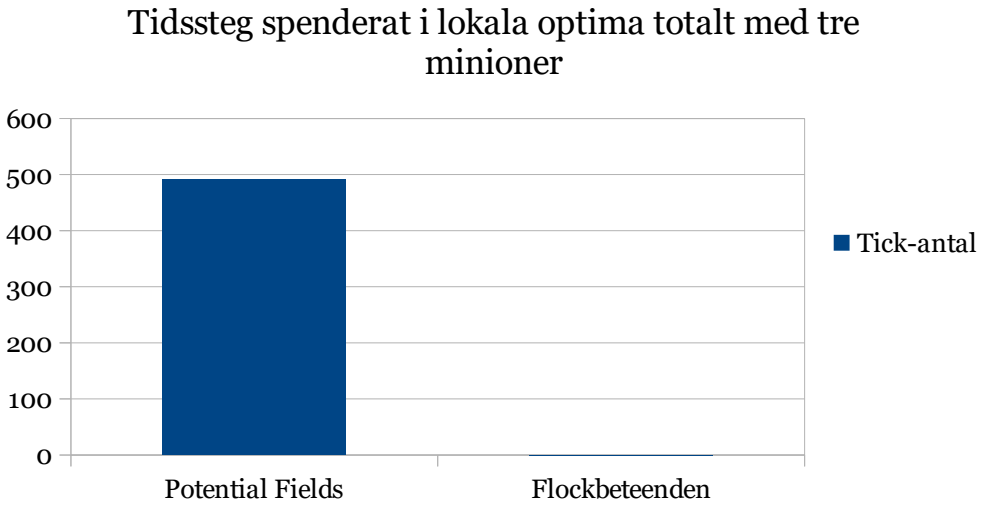
Efter varje omgång sparades medelvärdet av riktningsskillnaderna och tiden som minionerna tillbringade i lokala optima i en strängvariabel med CSV-format, vilket möjliggjorde skapandet av grafer med extern programvara. Därefter återställdes alla datainsamlingsvariabler till noll. Ett nytt set av minioner skapades därefter och simuleringen kördes igen. Den enda skillnaden var att varje lag hade en extra minion jämfört med den föregående omgången. Denna process upprepades tills antalet minioner översteg det maximala antalet på tjugo minioner, enligt experimentets inställningar. Slutligen sparades innehållet i strängvariabeln som en textfil.

4.5.1 Pilottest

Ett pilottest genomfördes med en simulering av endast tre minioner. Vid en inspektion av en omgång med detta minionantal framgår det att flockbeteendena uppvisade ett betydligt lägre medelvärde av riktningsskillnad vid varje uppdatering, vilket framgår i figur 7. Dessutom hade potential fields betydligt större problem med att fastna i lokala optima, medan implementationen för flockbeteendena inte påvisade detta problem alls, som illustreras i figur 8.



Figur 7 Graf som visar medelvärdet för riktningsskillnader för tre simulerade minioner i varje lag.



Figur 8 Graf som visar antalet tidssteg som tre minioner tillbringat i lokala optima totalt.

Som datan visade var det uppenbart att implementationen för flockbeteendena var mjukare än den för potential fields, eftersom värdena bör vara så låga som möjligt. Tre minioner utgav dock en relativt liten flock och skulle inte alltid vara förekommande i ett riktigt spel. För att avgöra om samma resultat skulle överföras mot andra minionantal krävdes ytterligare simuleringar och jämförelser.

5 Utvärdering

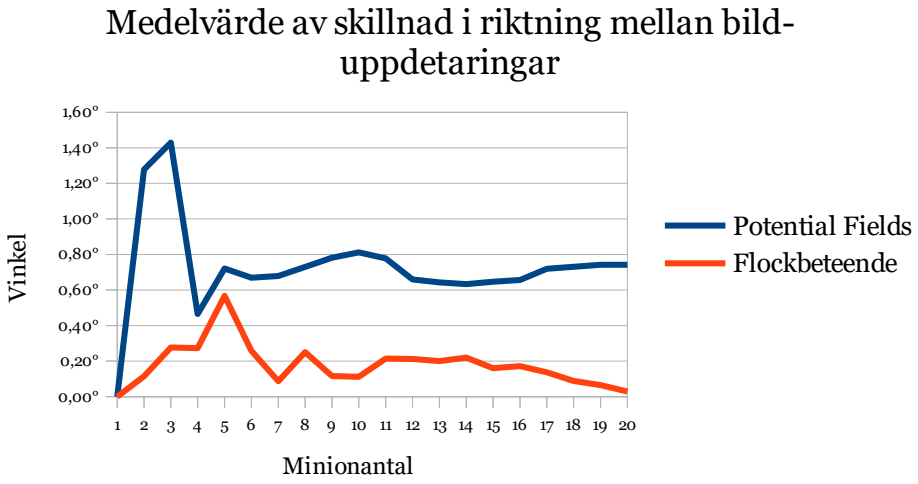
I detta kapitel presenteras den insamlade datan från experimentmiljön i form av grafer. En analys utfördes sedan utifrån denna data för att förklara varför de ser ut som de gör. Slutligen dras slutsatser utifrån analysen och datan.

5.1 Presentation och resultat av undersökning

För att expandera pilottestet har två nya simuleringar körts i experimentmiljön. Istället för att endast simulera tre minioner har nu hela intervallet med omgångar, som sträcker sig från en minion till tjugo minioner, genomförts vid varje simulering. Syftet var att undersöka hur riktningsskillnaderna och tiden som spenderades i lokala optima förändrats beroende på hur många minioner som existerade. Två separata simuleringar har genomförts med identiska inställningar i experimentmiljön, förutom valet av algoritm, vilket var den variabel som skulle utvärderas. Den ena simuleringen använde boids medan den andra använde potential fields. Storleken på fälten för potential fields var 260 i både bredd och höjd. Endast en enda körning av experimentet krävdes för varje navigeringsalgoritm, tack vare att hela simuleringen gav samma resultat varje gång den utfördes. De mätvärden som samlades in delades in i två huvudkategorier som beskriver minionernas mjukhet: riktningsskillnad och tid spenderad i lokala optima.

5.1.1 Riktningsskillnad

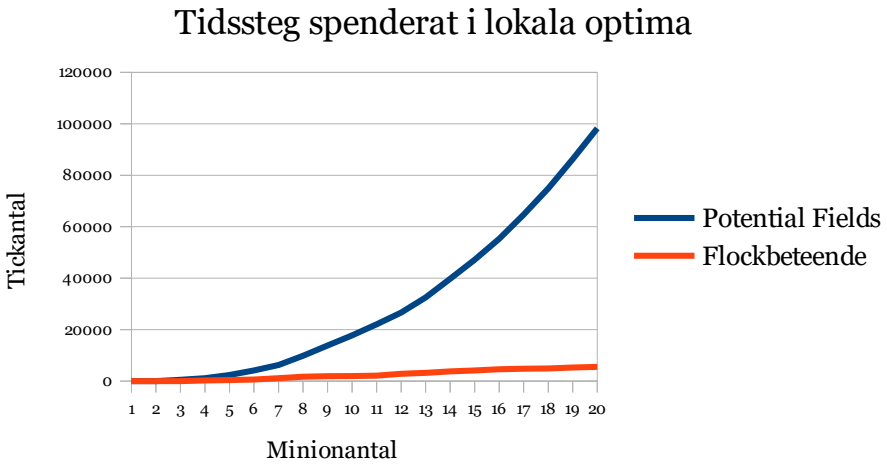
Grafen för riktningsskillnaderna visar medelvärdet för hur mycket en enskild minion svängde vid varje uppdatering i simuleringen. På den vertikala axeln visas medelvärdet för riktningsskillnad för ett specifikt antal minioner, medan den horisontella axeln representerar antalet minioner. Riktningsskillnaderna var mätta i grader. Se figur 9.



Figur 9 Graf som visar medelvärde för riktningsskillnad vid olika antal minioner.

5.1.2 Lokala optima

Grafen som visar minionernas lokala optima illustrerar antalet tidssteg då alla minioner i simuleringen var stillastående. Den vertikala axeln visar antalet tidssteg som minionerna stod stilla, medan den horisontella axeln visar, precis som i grafen för riktningsskillnaderna, antalet simulerade minioner vid en specifik omgång. Se figur 10.



Figur 10 Graf som visar antalet tidssteg som minionerna spenderade i lokala optima totalt för olika antal minioner.

5.2 Analys

Grafen för riktningsskillnaderna (se figur 9) visade att värdena var som högst vid färre antal minioner. Detta kan dock vara missvisande. Eftersom minionerna var tvungna att undvika varandra och gå runt de övriga minionerna, spenderade de en större del av tiden på att navigera runt dessa få minioner. När antalet minioner däremot ökade, radade de upp sig när de attackerade. När en ny minion skulle undvika dessa blev dess väg generellt sett rakare än om den skulle undvika en eller två minioner. Detta innebar att medelvärdet blev lägre vid högre antal minioner, endast för att minionerna spenderade mer tid på att gå rakt och en mindre del av sin tid på att svänga runt den minion som var sist i raden.

Det fanns även en likhet i grafen för lokala optima (se figur 10). Båda algoritmernas grafer visade högre värden vid ett specifikt antal minioner jämfört med det tidigare antalet. Detta betyder att båda algoritmerna tenderade att fastna proportionellt till hur många minioner som fanns i världen. Detta fenomen uppkom eftersom minionerna, när de gick till anfall, agerade som hinder för varandra. Ju fler minioner som stod i vägen, desto svårare blev det att navigera genom miljön. Formationen som uppstod, liknande en triangel, berodde på att minionerna alltid försökte röra sig mot den närmaste fienden. Utmaningen uppstod när fienden var i mitten av fiendelagets sida, vilket resulterade i att en minion försökte passera genom hela formationen. Detta ledde till att de som befann sig närmast mittpunkten i formationen inte kunde röra sig fritt tills de minioner som stod i vägen antingen flyttade på sig eller försvann. Med ökat antal minioner ökade även antalet som fastnade i formationens mitt.

Grafen över riktningsskillnaderna för implementationen av flockbeteendena visade en nedåtgående trend ju fler antalet minionerna blev, och anledningen till detta skulle kunna ha att göra med den triangelformation som skapades, vilket nämndes tidigare. Detta gjorde att minionerna navigerade, som tidigare nämnts, jämsides med en "vägg" av andra minioner, men nu med en vinkel hos en triangel. I en fullständig triangel skulle en minion endast behöva justera sin vinkel något för att följa väggen. Detta skiljer sig från en linjeformation vid färre antal minioner, där en minion var tvungen att svänga 90 grader. Samma fenomen inträffade även för potential fields, men de fastnade betydligt mer, även på triangelns kanter,

vilket resulterade i att de inte kunde navigera alls och väntade istället på en möjlighet att ta sig ut. Detta ledde till att formationen inte längre riktigt liknade en triangel, utan mer en klump. Detta kunde förklara varför grafen över potential fields såg ut att vara rakare än den för flockbeteendena. Detta kunde bero på att potential fields hade fler begränsningar för hur en enhet kunde röra sig. Eftersom en minion med potential fields endast kunde röra sig i åtta fördefinierade riktningar var det svårt för den att röra sig i riktningar som låg mellan två av dessa åtta. Detta var även anledningen till att potential fields generellt hade högre riktningsskillnader.

Båda graferna(figur 9, och figur 10) visade dessutom noll när antalet minioner var ett, vilket också är missvisande. Detta berodde på att inga mätvärden faktiskt togs eftersom det inte fanns någon annan minion att navigera runt eller fastna. Detta skedde på grund av att vardera minion på båda lagen endast gick rakt mot varandra och började direkt att attackera varandra när de kom i närhet.

5.3 Slutsatser

Experimentet som har genomförts i detta arbete jämförde flockbeteenden med potential fields för att navigera minioner i en MOBA-miljö (Multiplayer Online Battle Arena). Målet med experimentet var att besvara frågan om vilken av algoritmerna som gav en mjukare navigering för minionerna.

De sammanställda mätningarna från experimentet indikerade att flockbeteenden var det bästa alternativet att använda. Enligt grafen över riktningsskillnaderna (se figur 9) och grafen över antalet tidssteg som spenderades i lokala optima(se figur 10), hade potential fields alltid högre värden än flockbeteendena. Eftersom målet med experimentet var att hitta den algoritm som minimerar förekomsten av lokala optima och riktningsskillnader, innebär detta att den implementation som visar lägre värden är att föredra. Med dessa resultat kunde det konstateras att potential fields inte utgjorde ett alternativ när det gäller mjukhet.

I MOBA-spelet League of Legends (2009) används vanligtvis sex till sju minioner, beroende på vilken våg det är. I riktiga MOBA-spel genereras minionerna i vågor med trettio sekunders mellanrum (League of Legends Wiki 2023a). Eftersom båda algoritmerna redan hade visat en minskning vid dessa minionantal, skulle båda algoritmerna fungerat väl i denna situation. Minionerna fastnade inte heller i lokala optima så mycket vid dessa minionantal. Dock skulle antalet minioner kunna variera på grund av faktorer såsom attacker från hjältar, vilket skulle kunna öka riktningsskillnaderna. Antalet minioner skulle också öka om flera vågor kombineras, vilket skulle resultera i att de fastnar mer.

Lokala optima utgjorde fortfarande ett stort problem, vilket inte förväntades i ett scenario som detta. Agenter i StarCraft skulle ibland kunna navigera genom svår terräng (Hagelbäck 2016), och även om minionerna i experimentet inte hade någon terräng att navigera genom, blev de andra minionerna som attackerade hinder, vilket gjorde omgivningen komplex. Eftersom minionerna var cirkelformade sett från simuleringens perspektiv, innebar det att om två minioner stod bredvid varandra och fungerade som hinder, kunde en annan minion fastna mellan dem eftersom det skapades ett konkavt utrymme mellan minionerna som var svårt att ta sig ur. Trots att flockbeteendena hade de lägsta värdena, återstod problemet med lokala optima och det skulle krävas ytterligare utveckling av algoritmen för att helt förebygga detta. Som Dolui och Hancox (2021) förklarade, bör agenterna navigera smidigt och mjukt framåt, men för närvarande fastnade minionerna alltför mycket för att uppnå detta. Av

denna anledning ansågs ingen av algoritmerna vara tillräckligt bra för användning i ett riktigt spel.

6 Avslutande diskussion

Detta avslutande kapitel inleds med en sammanfattning av hela arbetet. Därefter diskuteras de brister som har uppstått och som eventuellt har påverkat resultatet. Slutligen presenteras förslag på framtida arbeten om arbetet skulle vidareutvecklas.

6.1 Sammanfattning

Syftet med detta arbete har varit att utvärdera mjukheten i navigeringen för minioner i ett MOBA-scenario genom att jämföra två olika dynamiska navigeringstekniker: flockbeteenden och potential fields. Mjukhet hänvisar till hur mycket minionernas riktning ändrades mellan bilduppdateringar och hur ofta de fastnade i lokala optima.

Det finns forskning inom detta område där agenter i realtidsspel kan styras med hjälp av flockbeteenden och potential fields. Fokuset har dock oftast legat på exekveringstid och vinstfrekvens istället för mjukhet (Hagelbäck 2016). Däremot hade Dolui och Hancox (2021) utseendet i åtanke när de skapade sina monster i sitt spel. De strävade efter att uppnå en navigering som flöt mjukt framåt, vilket var det som detta arbete ville undersöka.

En experimentmiljö utvecklades där mjukheten testades genom att variera antalet minioner i varje omgång mellan en miniont till och med tjugo. Programmet mätte automatiskt mjukheten vid varje bilduppdatering och sammanställde sedan resultaten. Resultaten visade att flockbeteendena generellt var bättre än potential fields både när det gällde riktningsskillnaderna och tiden som spenderades i lokala optima. Dock var lokala optima fortfarande ett problem för båda algoritmerna, vilket inte förväntades. Slutsatsen var att ingen av algoritmerna skulle fungera tillräckligt bra i ett MOBA-spel.

6.2 Diskussion

Eftersom algoritmen för flockbeteendena använde viktade krafter innebar det att resultaten var associerade till de vikter som användes i implementationen. Både Hagelbäcks artikel (2016) och artikeln av Kawabayashi och Chen (2008) beskrev olika vikter för styrbeteendena. Då dessa vikter inte var exakt likadana, skulle simuleringar med vikter från den ena artikeln inte generera samma resultat som den andra. Tack vare att prototypen av simuleringen använde spelmotorns grafikmotor, var det också möjligt att observera detta. Exakta vikter från någon av artiklarna användes inte, utan istället användes egna inställningar(se Appendix A). Eftersom det tog tid att finjustera de korrekta inställningarna för flockbeteendena och få algoritmen att fungera som tänkt, var det tydligt att olika inställningar påverkade simuleringen. Detta innebar att resultaten blev specifika för den implementation och det MOBA-scenario som användes i detta arbete, men det behöver nödvändigtvis inte gälla generellt för flockbeteenden. Dessa inställningar konfigurerades manuellt och testades noggrant för att uppnå en visuellt tilltalande navigering. En slutsats av detta är att algoritmen skulle kunna förbättras genom att undersöka och identifiera de mest optimala vikterna för krafterna. En liknande situation skulle uppstå för potential fields. Eftersom storleken på fälten i denna algoritm påverkade resultaten skulle det vara möjligt att potential fields faktiskt kunnat prestera bättre, men att storleken(se Appendix A) som användes i detta experiment var för liten.

Anledningen till att större fält inte användes var att simuleringen tog alltför lång tid att exekvera. Hela simuleringen för potential fields tog totalt fem timmar, vilket inte var ett problem vid datainsamlingen men blev det vid prototypandet. Anledningen till att storleken 260 valdes var att det var en rimlig storlek för felsökning, där man kunde observera navigeringen utan att det tog för lång tid, samtidigt som fälten inte blev för små för MOBA-scenariot. Trots att exekveringstiden inte var i fokus för detta arbete, innebar det ändå att denna implementation av potential fields inte skulle vara användbar i ett verkligt spel eftersom exekveringstiden var för lång. Hagelbäck (2016) nämnde att potential fields var hundra gånger långsammare än en implementation med flockbeteenden. Denna långsammare exekvering kunde också observeras i detta experiment. Det är dock svårt att bedöma hur skillnaden skulle vara i en mer optimerad version, eftersom dessa implementationer var naiva. Att använda större fält skulle eventuellt kunna göra spelet ospelbart även vid en optimerad lösning.

En annan aspekt som bör beaktas är att MOBA-spel kräver dedikerade servrar, vilket är en tjänst. För att hantera så många matchinstanser som möjligt på en enda server är det önskvärt att de algoritmer som används i spelet exekveras så snabbt som möjligt. Därför skulle exekveringstiden kunna vara en viktig faktor i MOBA-spel och kanske borde den ha fått mer fokus. Även om mjukhet är viktigt, kan exekveringstiden påverka valet av algoritm. Även om potential fields skulle visa sig vara mjukare än flockbeteendena kanske det ändå inte är värt det om den presterar, som Hagelbäck (2016) säger, "hundra gånger långsammare”.

Inom forskningsetiken bidrar reproducerbarhet och validitet till en bättre kvalitet i forskningen (Vetenskapsrådet 2017). Trots att resultaten var beroende av implementationen av algoritmerna har arbetet strävat efter reproducerbarhet och validitet genom att välja metoder som ökade trovärdigheten. En viktig del av detta har varit att utesluta faktorer som kunde införa slumpmässiga moment eller oväntade variabler. Till exempel använde båda algoritmerna samma delta-tid för att säkerställa att minionernas rörelse skulle förbli densamma om experimentet utfördes på nytt. Dessutom har inga människor eller djur varit involverade i arbetet, vilket innebar att inga etiska avväganden behövde göras, då ingen individ påverkades av framställningen av resultaten. Detta beror på att experimentet endast genomfördes med en enkel artificiell intelligens programmerad på en dator, där agenterna enbart simulerades mot varandra utan någon påverkan från den verkliga världen.

6.3 Framtida arbete

Om detta arbete ska fortsätta genom att undersöka om de implementationsspecifika algoritmerna kan optimeras (se kapitel 6.2), kan ett nytt experiment utföras där man justerar vikterna för flockbeteendenas styrbeteenden och ändrar storleken på implementationen för potential fields för att ytterligare optimera dem. Det är värt att utforska om potential fields kan producera lägre riktningsskillnader än flockbeteendena genom att öka storleken på fälten. Det är också viktigt att implementera en optimerad algoritm för att kunna undersöka hur stora fälten kan vara innan de blir för långsamma att exekvera.

Ett annat experiment skulle kunna genomföras som inför ytterligare restriktioner för när experimentmiljön får mäta värdena. Till exempel borde mätvärden för riktningsskillnader endast sparas ned när riktningen inte är densamma som den föregående riktningen. Detta kan eventuellt motverka de extremvärden som uppstår vid lägre minionantal (se kapitel 6.2).

Eftersom dessa algoritmer hade svårigheter med att fastna i lokala optima, skulle de inte vara särskilt användbara i verkliga situationer om ett faktiskt spel skulle utvecklas. Därför skulle nya beteenden inom algoritmerna behöva undersökas för att motverka problemet med lokala optima. Ett exempel på detta skulle kunna vara att förhindra att en minion går till den position den nyligen var på (Hagelbäck 2012). Ett annat exempel skule vara att använda en algoritm som inte lider av lokala optima alls, såsom potential flows (Nguyen, T., Nguyen, K. & Thawonmas 2013).

Referenser

Borg , E. & Westerlund, J. (2014). *Statistik för beteendevetare*. (3:e uppl.) Kina: Liber.

Dolui, D. & Hancox, J. (2021) Steering, Formations, and a Trail of Blood: AI in Wolcen. I *Game Developer Conference 2021*. Kalifornien, USA 19-21 juli 2021.

Eliasson, A. (2019). *Kvantitativ metod från början*. (4:e uppl.) Lund: Studentlitteraturer.

Godot. (2007). Windows, macOS & Linux [Mjukvara]. Linietsky, J., Manzur, A. & Contributors. Tillgänglig på internet: https://godotengine.org/

Godot Docs (2014a). *The Inspector*. https://docs.godotengine.org/en/stable/tutorials/editor/inspector_dock.html [2023-04-07]

Godot Docs (2014b). *Running Code in the editor*. https://docs.godotengine.org/en/stable/tutorials/plugins/running_code_in_the_editor.html [2023-04-07]

Hagelbäck, J. (2012). Potential-field based navigation in StarCraft. I *2012 IEEE Conference on Computational Intelligence and Games*. Granada, Spanien 11-14 september 2012, ss. 388-393. doi.org/10.1109/CIG.2012.6374181

Hagelbäck, J. (2016). Hybrid Pathfinding in StarCraft. *IEEE Transactions on Computational Intelligence and AI in Games,* 8(4), ss. 319-324. doi.org/10.1109/TCIAIG.2015.2414447

Jae, M. L., Se, H. C. & Rafael, A. C. (2009). A fast algorithm for simulation of flocking behaviour. I *2009 International IEEE Consumer Electronics Society's Games Innovations Conference*. London, Storbritannien 25-28 augusti 2009, ss. 186-190. doi.org/10.1109/ICEGIC.2009.5293611

János, S. & Matijevics, I. (2010). Implementation of potential field method for mobile robot navigation in greenhouse environment with WSN support. I *IEEE 8th International Symposium on Intelligent Systems and Informatics*. Subotica, Serbien 10-11 september 2010, ss. 319-323. doi.org/10.1109/SISY.2010.5647434

Kawabayashi, H. & Chen, Y. (2008). Interactive System of Artificial Fish School Based on the Extended Boid Model. I *2008 International Conference on Intelliget Information Hiding and Multimedia Signal Processing*. Harbin, Kina 15-17 augusti 2008, ss. 721-724. doi.org/10.1109/IIH-MSP.2008.209

League of Legends (2009). Windows & macOS [Game]. Riot Games: West Los Angeles, Kalifornien, USA. Tillgänglig på internet: https://www.leagueoflegends.com/

League of Legends Boards (2016). *Minion AI Rules Documentation*. *http://web.archive.org/web/20200308182225/https://boards.na.leagueoflegends.com/en/c/developer-corner/qRHotV9k-minion-ai-rules-documentation [2023-02-17]*

League of Legends Wiki (2023a). *Minion (League of Legends)*. https://leagueoflegends.fandom.com/wiki/Minion_(League_of_Legends) [2023-01-30]

League of Legends Wiki (2022). *Minion (Species)*. https://leagueoflegends.fandom.com/wiki/Minion_(Species) [2023-01-30]

League of Legends Wiki (2023b). *Ticks and updates*. https://leagueoflegends.fandom.com/wiki/Tick_and_updates [2023-03-04]

Leigh, R., Louis, S. J. & Miles, C. (2007). Using a Genetic Algorithm to Explore A*-like Pathfinding Algorithms. I *2007 IEEE Symposium on Computational Intelligence and Games*. Honolulu, USA 01-05 april 2007, ss. 72-79. doi.org/10.1109/CIG.2007.368081

Nguyen, T., Nguyen, K. & Thawonmas, R. (2013). Potential flow for unit positioning during combat in StarCraft. I *2013 IEEE 2nd Global Conference on Consumer Electronics*. Tokyo, Japan 01-04 oktober 2013, ss. 10-11. doi.org/10.1109/GCCE.2013.6664759

Reynolds, C. W. (1987). Flocks, herds and schools: A distributed behavioral model. *ACM SIGGRAPH Computer Graphics,* 21(4), ss. 25-34. doi.org/10.1145/37402.37406

Sabri, A. N., Radzi, N. H. M. & Samah, A. A. (2018). A study on Bee algorithm and A* algorithm for pathfinding in games. I *2018 IEEE Symposium on Computer Applications & Industrial Electronics*. Penang, Malaysia 28-29 april 2018, ss. 224-229. doi.org/10.1109/ISCAIE.2018.8405474

StarCraft (1998). Windows, Classic Mac OS, macOS & Nintendo 64 [Game]. Blizzard Entertainment: Irvine, Kalifornien, USA. Tillgänglig på internet: https://us.shop.battle.net/en-us/product/starcraft

Vetenskapsrådet (2017). *God Forskninssed*. https://www.vr.se/analys/rapporter/vara-rapporter/2017-08-29-god-forskningssed.html [2023-05-14]

Williamson, K. (2002). Research techniques: Questionnaires and interviews. I Williamson, K. & Bow, A. (red.) *Research methods for students, academics and professionals: information management and systems*. Wagga Wagga, N.S.W.: Chandos Publishing, ss. 235-247.

Wolcen: Lords of Mayhem (2020). Windows [Game]. Wolcen Studios: Nice, Frankrike. Tillgänglig på internet: https://wolcengame.com/

Appendix A – Algoritmernas vikter och krafter

Vikten för målsökarbeteendet var 1 / 100.

Vikten för separationsbeteendet var 0.33 / (distansen till den andra minionen upphöjd med två).

Vikten för sidostyrningsbeteendet var 0.01 / (distansen till den andra minionen).

Världens totalstorlek för potential fields var 260x260.

Det attraherande kraften för potential fields hade en diameter på 520 rutor och hade en mittkraft på 100 och som sedan gick ner mot 0 till kanterna.

De bortåtstötande fälten hade en diameter på 13 rutor och hade en kraft på -50 i mitten och interpolerades mot -25 vid kanterna.