Achtung Online AI

MATHIAS LINDBLOM
och LUCAS WIENER

KTH Datavetenskap
och kommunikation

Examensarbete

Stockholm, Sverige 2012

Achtung Online AI

M A T H I A S L I N D B L O M

Storgatan 52, 17152 Solna

073-051 72 26

mathlin@kth.se

L U C A S W I E N E R

Strålgatan 2, 11263 Stockholm

070-797 72 17

lwiener@kth.se

DD143X, Examensarbete i datalogi om 15 högskolepoäng
vid Programmet för datateknik 300 högskolepoäng
Kungliga Tekniska Högskolan år 2013
Handledare på CSC var Per Austrin
Examinator var Mårten Björkman

Kungliga tekniska högskolan
Skolan för datavetenskap och kommunikation
KTH CSC
100 44 Stockholm
URL: www.kth.se/csc

Examensarbete

Stockholm, Sverige 2012

Sammanfattning

Rapporten är egentligen tvådelad men har skrivits som en helhet då delarna högst beror av varandra. Ena delen handlar om en teoretisk AI till spelet “Achtung Online”. Andra delen handlar om att implementera en demoversion, som är en enklare version av denna teoretiska AI. Huvudmålet är att skapa en underhållande AI som kan utmana mänskliga spelare. Detta har i sin tur delats upp i kvalitetskrav som ställts på bägge delarna. Bakgrunden beskriver spelet i detalj och det mer kända orginalet “Achtung die kurve”. Sedan beskrivs den historiska utvecklingen av AI inom spelutveckling. Studien som följer är omfattande och består dels av en litteraturstudie, dels av en framtagning av tydliga krav på de datorstyrda spelarna. De intressanta och mest passande koncepten inom AI lyfts även fram. Med bas i studien utformas en komplett teoretisk AI samt vilka delar som ingår i demoversionen. Utifrån detta har demoversionen sedan skapats och rapporten går sedan igenom de intressanta delarna av implementationen. Resultatet visar att demoversionen uppnått de krav som ställts och till sist diskuteras hur framtiden ser ut för spelet “Achtung Online”.

Abstract

The report can be devided into two main parts; the theoretical construction of AI to be used in the game “Achtung Online”, and the implementation of a demo version of the described theoretical AI. The main goal is to create an AI that is entertaining and challenging for a human player. This have led to a concretized list of quality goals for the two main report parts. The report background presents the history of the game concept “Achtung die kurve” and also the evolution of game AI theory. The main study consists of a litterature study and a comprehensive determination of concrete goals of the game bots behavior. Some interesting AI techniques to the game will also be discussed in detail. Building upon the study, a specification of the final game AI is constructed. The demo version of the AI will implement some of the concepts of the theoretical AI specification. The report will then discuss about the implementation and then finally present the results followed by a discussion.

Förord

Målgruppen för denna rapport är utvecklare med grundläggande kunskap kring artificiell intelligens inom spelutveckling. Det innebär att rapporten förutsätter att läsaren redan har god datateknisk kunskap och därmed inte förklarar begrepp som av rapportskrivarna anses vara allmän datateknisk kunskap. Istället fokuserar rapporten på att förklara de element inom AI som kan nyttjas för ett spel liknande “Achtung Online”. För den som är främmande för spelkonceptet finns en tabell med termer och förkortningar som används i rapporten i bilaga 9.1. Författarnas samarbete redogörs i bilaga 9.2.

Examensarbete

Stockholm, Sverige 2012

1. Introduktion

1.1. Syfte

Spelet “Achtung Online” är ett spel som i dagsläget endast kan spelas med andra fysiska personer. Då spelet helt saknar datorstyrda motståndare, *botar*, är syftet med denna rapport att skapa *artificiell intelligens* för botar och att även implementera detta i det befintliga spelet.

1.2. Mål

Målet med rapporten är att konstruera en teoretisk AI till spelet som ska vara utmanande och rolig för människor att spela mot. Vidare är målet att implementera en enklare version av denna teoretiska AI för att kunna testa och demonstrera de grundläggande teknikerna som fastställts i teorin. Det implementerade resultatet bör vara tillräckligt bra för att kunna erbjuda utmanande och roliga matcher mot människor som ej är särskilt bekanta med spelet. Sammanfattat kan målet beskrivas som följande:

Konstruera en teoretisk AI som ska vara utmanande och rolig för alla spelare, samt implementera ett enklare demo av denna AI som ska vara tillräckligt bra för nybörjare att spela mot utmanande och roliga matcher mot.

1.3. Problembeskrivning

Problemet kan brytas ned i tre delproblem; att göra *botarna* så smarta som möjligt, att göra de beräkningar som behövs så effektivt som möjligt samt att göra botarna så mänskliga som möjligt. Botarna kan antagligen göras oerhört smarta om antalet parametrar samt beräkningar tillåts gå mot oändligheten. Detta går förstås inte då beräkningarna ej får påverka spelets prestanda så mycket att spelet blir ospelbart. Huvudproblemet är således att konstruera denna balans av kvalitet, prestanda och naturlighet.

1.4. Dokumentöversikt

Dokumentet kan delas upp i tre delar; bakgrund, studie samt implementation.

Första delen kommer att beskriva spelet samt liknande kloner där även en analys av dessa kloners befintliga botar kommer att presenteras. En snabb genomgång av befintliga AI-metoder kommer även beskrivas i denna del.

Andra delen beskriver själva studien och konstruktionen av AI till spelet. Inledningsvis kommer en kravspecifikation att presenteras för att lättare kunna finna passande algoritmer och metoder. När kända algoritmer och koncept studerats kommer en egen algoritm att utformas i stycke 3.3.

Tredje delen kommer att beröra själva implementationen av de erhållna algorimerna som tagits fram i föregående del. I stycke 4.1 kan läsaren ta del av speciella implementationslösningar samt kodstycken som vi anser är särskilt intressanta. Efter det kommer testresultat att presenteras.

Rapporten avslutas med en sammanfattande presentation av de erhållna resultaten följt av en diskussion. Metodiken för denna studie kommer även beskrivas i den avslutande delen av rapporten. Sist listas en referenslista med analyser av källors trovärdighet samt vetenskapliga kvaliteter.

Det ska åter nämnas att i bilaga 9.1 finns en tabell som beskriver alla förkortningar och speciella termer som används i rapporten, vilket rekommenderas att läsas innan själva rapporten eller samtidigt vid sidan om.

2. Bakgrund

För att bättre förstå problembeskrivningen behövs en utförlig förklaring av spelets egenskaper och mekanismer. I kapitel 2.1 kommer spelkonceptets historia presenteras följt av en beskrivning av "Achtung Online" i kapitel 2.2. I kapitel 2.3 ges en introduktion till begreppet artificiell intelligens inom datalogi samt spelteori. Sist kommer en utvärdering av nuvarande botar till andra "Achtung, die Kurve!"-spel sammaställas i kapitel 2.4.

2.1. Spelkonceptets historia

Historien om spelkonceptet "Achtung, die Kurve!", eller "*Zatacka"* som vissa versioner har kallats, är undermåligt dokumenterad och på grund av bristande källor bör följande avsnitt läsas kritiskt. Om inget annat anges i texten är källan från wikipedia [4].

Namnet "Achtung, die kurve!" är tyskt och betyder "Akta dig för kurvan!" vilket beskriver vad spelet går ut på - nämligen att akta sig för motståndarens skapta *kurvor*. Spelet påminner en hel del om det kanske mer kända *singleplayer*-spelet "Snake", från 1978, där spelarens mål är att överleva så länge som möjligt utan att åka in i sig själv eller spelbanans väggar [5]. De mest markanta skillnaderna mellan ADK och "Snake" är att ADK kräver minst 2 spelare, kropparna försvinner inte och svängarna är mjuka istället för vinkelräta. Till en början, år 1993 då första kända versionen av ADK släpptes vid namnet "Cervi", var detta i princip allt som skilde spelen åt. År 1995 lanserades en ny ADK-version av Filip Oscad[1], som fick större uppmärksamhet än föregångaren "Cervi". Denna version fick namnet "Achtung, die kurve!" och att namnet har levt vidare i nästan samtliga nya kloner av spelet indikerar versionens popularitet. Till skillnad från "Cervi" så skapade nu maskarna små hål vilket gav spelet större djup och taktiska möjligheter, se figur 1 nedan. Detta tillsammans med passande musik och mer tilltalande design gav spelet sin popularitet som dess föregångare och dåvarande konkurrenter inte kunde leva upp till.

[1] Filip Oscad är även känd som Fred Brookersom.

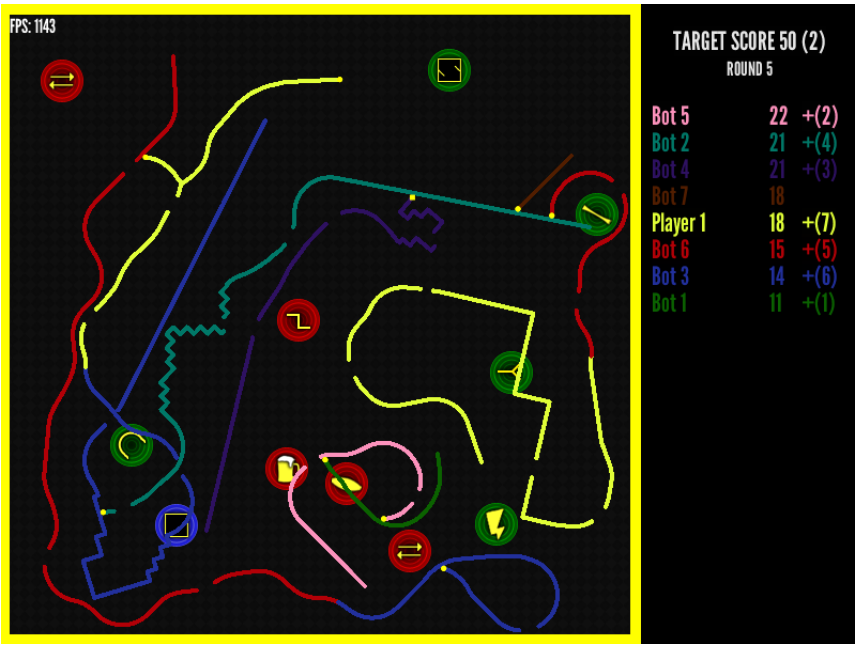
Figur 1. En bild av spelet “Zatacka”. En liknande klon av ADK-versionen från 1995. Slutbilden[2] av en match med 6 spelare.

Under årens lopp har spelet förändrats en del men grunderna, som nämnts ovan, finns med i varje modern klon av spelet. De populäraste varianterna[3] av spelet har även lagt till *powerups* som temporärt förändrar spelets egenskaper. De moderna spelen kan oftast spelas via en webbläsare. Vissa unika element skiljer varianterna. Till exempel kan endast spelet “Curve Fever” spelas med andra spelare över internet, “Achtung Online” möjliggör lagspel och “Achtung, Die Kurve 3D!” kan som namnet antyder spelas i 3D.

Spelet har störst intresse i norden och centrala delar av Europa vilket inte är så konstigt då namnet är tyskt, originalet kom från Tjeckoslovakien och skaparen av den nuvarande populäraste versionen “Curve Fever” är Holländsk. Ett nämnvärt intresse finns även i olika delar av Asien där turneringar i spelets ära har genomförts. Totalt har fyra större ADK-turneringar anordnats, med tävlande från flera länder, där den senaste hölls i Danmark år 2012.

2.2. Achtung Online

[2] Upphovsrätt undersöks och om inte tillåtelse givits innan publicering av rapporten tas bilden bort.
[3] “Achtung, die Kurve! Flash Remake” och “Curve Fever”.



Figur 2: Spelet “Achtung Online”. Spelet visar en match där spelarna består av 7 botar och en mänsklig spelare.

2.2.1. Bakgrund

År 2010 läste vi (Mathias och Lucas) tillsammans kursen “Introduktion till datalogi” vid KTH där ett slutprojekt skulle skapas i Java. Mathias ville gärna göra en egen version av ADK och intalade Lucas att det vore ett spännande och intressant projekt. En prototyp skapades därefter och visades upp inför resten av kursdeltagarna som tyckte att spelet var roligt. Då det långsiktiga målet med spelet är att spelare ska kunna spela mot varandra över internet namngavs spelet till “Achtung Online”. Efter kursens slut fortsatte utvecklingen i ungefär ett år tills vi blev nöjda med spelet. AO laddades därefter upp på en hemsida[4] tillgängligt att spela gratis med ett tillhörande forum[2]. Kort därefter upphörde utvecklingen av spelet då skolans tempo samt privata åtaganden upptog för mycket tid.

Då AO utvecklades gjordes, vid flertalet tillfällen, undersökningar kring de versioner av ADK som fanns tillgängliga för allmänheten. Utifrån dessa undersökningar har slutsatser i detta stycke framtagits av rapportskrivarna själva. Som nämnt är AO den enda kända klonen som möjliggör lagspel. Andra utmärkande drag är avancerade inställningar, valbar rund spelbana, upp till tolv spelare och unika powerups. Se bilaga 9.5.1 för en fullständig lista av powerups som finns i AO. Spelet var även först med möjligheten att bestämma riktning av maskarna vid matchstart. *Figur 2* ovan visar en match med olika powerups på spelbanan där några har blivit tagna och påverkat maskarnas beteendet. Notera att maskarnas *huvud* representeras som en gul cirkel.

[4] http://achtungonline.com, http://achtungonline.com/forum

2.2.2. Spelbeskrivning

Minst två till max tolv spelare kan spela spelet samtidigt. Det finns en fyrkantig och en rund spelbana. Ett spel består av flera matcher, tills någon spelares poäng uppnått en förvald målpoäng samt att det är minst två i poängskillnad mellan den spelare med högst poäng och den spelare med näst högst poäng. En spelare måste alltså vinna med två poäng. Spelare förlorar en match då dess masks huvud nuddar en vägg eller mask, och då ges poäng till alla spelare vars maskar fortfarande är vid liv. Ibland uppstår powerups vid godtycklig plats på spelbana, som spelare kan aktivera genom att låta deras masks huvud nudda. Dessa powerups ändrar matchens egenskaper på olika sätt. Det finns gröna, röda och blå powerups. Gröna powerups har positiva egenskaper och påverkar spelarens egna mask. Röda powerups har negativa egenskaper och påverkar alla motspelares maskar. Blå powerups har neutrala egenskaper och påverkar alla maskarna, inklusive den mask som tog den. Spelares maskar åker alltid framåt med bestämd hastighet och spelare kan endast påverka masken genom att styra höger eller vänster.

Det går att spela i tolv olika lag där antalet spelare i varje lag fördelas fritt. Till exempel går det att spela i lag om fyra, tre, två samt en spelare. Varje lag har en viss färg som spelarna får välja olika nyanser av. I lagspel delas poäng ut först då ett lags samtliga maskar dött. Röda powerups påverkar inte heller maskar tillhörande samma lag.

2.3. AI inom datalogi

2.3.1. Historia

Artificiell intelligens, förkortat AI, är ett relativt nytt begrepp inom datorpelsindustrin. För att redogöra vad begreppet AI omfattar är det viktigt att först förstå hur AI utvecklats de senaste åren. Den tidigaste dokumenterade formen av AI i spelvärlden dök upp först 1974 där motståndare rörde sig enligt simpla mönster som den mänskliga spelaren skulle försöka skjuta ned [7, kap 1 sid 5]. Första spelet som tillämpande FSM[5] var “Herzog zwei” år 1990 följt av, 3 år senare, det mer kända spelet “Doom” [7, kap 1 sid 5]. Fienderna i dessa spel kunde röra sig på olika sätt beroende på hur spelaren rörde sig. Spelet “Half-Life” släpptes 1998 och anses vara ett stort steg för AI inom spelutveckling då spelarna upplevde att deras motståndare agerade intelligent på en helt ny nivå jämfört med tidigare spel [7, kap 1 sid 5][9]. Det skulle kunna tänkas att det var någon revolutionerande ny teknik som speltillverkaren “Valve” använde sig utav men i själva verket var spelets AI väldigt *hårdskriptat*. Skillnaden mot andra speltillverkare var att utvecklarna hade tillsatt avsevärt mer tid och resurser åt sin AI [7, kap 1 sid 5]. Den positiva respons som “Half-Life” fick ledde till ett generellt ökat intresse för avancerad AI både hos speltillverkare och spelare [7, (har inte boken vid mig)]

2.3.2. Definition av AI inom spelutveckling

AI är svårdefinierat, inte minst på grund av att ordet intelligens har fått flera olika definitioner

[5] Finite-state machine. Modell för att beskriva än ändlig mängd tillstånd och övergångar mellan tillstånden. http://xlinux.nist.gov/dads/HTML/finiteStateMachine.html

genom tiden och debatteras än idag. Generellt brukar AI vara någon sorts människoskapad intelligensform som kan tänka och resonera, eller på andra sätt efterlikna en människas intelligens [7, kap 1]. I spelsammanhang passar inte denna definition då begreppen *tänka* och *resonera* kan vara ännu svårare att definera och är missvisande för vad AI inom spel verkligen innebär. AI inom spelutveckling går ut på att ge den mänskliga deltagaren en upplevelse av att möta intelligenta motståndare där motståndarna i själva verket ofta kan vara hårdskriptade och långt ifrån intelligenta [8]. En bättre definition av spelrelaterad AI skulle vara förmågan att samla och applicera kunskap [7, kap 1]. Notera att klassisk AI generellt inte går att definiera på detta sätt då det skulle innebära att till exempel en termostat är intelligent då den samlar information i form av värme och sedan agerar efter denna kunskap. En djupare analys skulle kunna göras av vad AI och spelrelated AI verkligen innebär men det viktigaste att förstå är att en bra AI inom spelutveckling ska upplevas intelligent och mänsklig, vara underhållande, belasta prestandan så lite möjligt och därför ignoreras frågan kring dess verkliga intelligens i denna rapport [7, kap 1].

Även om ovanstående definition är vedertagen så används begreppet AI löst inom datalogi. I modern spelutveckling beskrivs ibland AI som sättet ett användargränssnitt beter sig på beroende på indata eller effekterna kring kollisionsdetektering [7, kap 1]. I boken “AI Game Engine Programming” läggs följande begränsning till: *Karaktärsbaserat intelligent beteende.* Detta tar bort spelelement såsom kollisionsdetekning men vad intelligent beteende innebär är fortfarande oklart. Boken “AI Game Programming Wisdom 3” väljer istället att begränsa sig enligt följande: *System som tänker och/eller agerar rationellt eller som en människa skulle.* Att agera rationellt innebär kort att agera enligt vad som är logiskt korrekt och förklarar ganska bra vad intelligent beteende betyder inom spelrelaterad AI. Det ska dock tilläggas att målet oftast inte är att skapa den perfekta motståndaren som slår alla mänskliga spelare. Istället är målet att botarna upplevs mänskliga och då människor ibland beter sig irrationellt innebär det att en bra bot bör efterlikna detta beteende. Det irrationella beteendet kan antingen efterliknas medvetet eller omedvetet via bristfällig AI. Ena definitionen begränsar vad spelrelaterad AI är och den andra begränsar hur denna AI beter sig. Rapporten kommer därför definiera spelrelaterad AI till följande:

Karaktärsbaserat system som tänker och/eller agerar rationellt eller som en människa skulle.

Från och med nu kommer spelrelaterad AI refereras till som AI enligt ovan definition, om inget annat anges.

2.4. Botar till andra kloner av “Achtung die kurve”

Följande beskrivningar av existerande implementationer av AI i olika kloner av ADK kommer baseras av subjektiva bedömningar. Varje klon kommer testas ytligt och jämföras mot varandra.

- **“Achtung, die Kurve! Flash Remake”**

Figur 3: Spelet “Achtung, die Kurve! Flash Remake”. Röd mask är den enda mänskliga spelaren.

Botarna upplevs bra på att undvika kollisioner och de åker genom hål utan större bekymmer. De tar ofta bra kortsiktiga beslut men det verkar inte finnas någon övergripande plan då botarna ofta åker in i återvändsgränder eller till och med gör loopar in i sig själva (se grå samt turkos mask i *figur 3*). Det finns powerups i spelet men botarna verkar inte ha något taktisk funktionalitet då de endast aktiverar en powerup om deras vägar råkar korsas. Om spelaren tar en powerup som till exempel växlar botarnas styrkontroller så påverkas de inte av det. Det känns allmänt som att botarna inte agerar mänskligt när de blir påverkade av powerups. Det finns inte heller några tecken på att botarna besitter någon slags förmåga att spela offensivt, vilket verifierar att en taktisk funktionalitet antagligen ej existerar. Dock skapas ibland en känsla av att botarna försöker stänga in andra maskar på grund av att spelaren måste spela mot fem botar samtidigt och den slumpmässiga navigationen ger falska intryck av intelligens. Denna bot upplevdes roligast att spela mot jämfört med de andra varianterna.

- **“Achtung duo”**
  I detta mobilspel spelar spelaren mot enbart en bot i taget. Motståndaren i detta spel känns lättbesegrad och ganska enkelspårig. Boten undviker kollisioner genom att byta riktning men reagerar ofta för sent och kan inte avgöra vilket håll den ska svänga åt. Då boten till exempel åker rakt mot spelbanans hörn så kommer den garanterat dö utan hot från spelaren. Rapporten kommer återkoppla till dessa trattliknande situationer senare. Se *figur XYY* i kapitel *6.1 Allmän prestation av AI*.
- **“Psykurve”**
  Botarna i det här spelet var mest utmanande av de provspelade klonerna men det beror inte på kvaliteten av spelets botar. Istället beror det på en bugg i spelet, nämligen mänskligt osynliga hål i maskar som botarna nyttjade. Att masken är mer svårstyrd i spelet gör även att botarna upplevs mer utmanande. Som med “Achtung duo” så klarar inte botarna av trattsituationer då de ibland fortsätter framåt mot en garanterad död. I denna klon lyckas botarna dessutom ibland styra maskarna rakt in i hinder utan till synes

direkt förklaring. En misstanke är att det har att göra med de osynliga hålen som aktiverar en sväng hos en bot trots att hålet är för smalt.

Samma mönster upprepas i samtliga kloners AI. Botarna spelar rent defensivt och försöker aldrig aktivt eliminera motståndare. Det märks även att botarna inte har någon direkt överblick över spelbanan utan fokuserar endast på en lokal yta framför botarnas huvuden.

3. Studie

Detta kapitel kommer ligga som grund inför utvecklandet av botarnas AI. Inledningsvis kommer kapitel 3.1 redogöra studiens begränsningar och vidare kommer i kapitel 3.2 krav på spelets botar att listas. Den större delen av detta kapitel utgörs sedan av kapitel 3.3, vilket är en litteraturstudie som behandlar olika koncept och tekniker som kan tänkas vara intressanta att använda i AO. Sist i kapitel 3.4 finns en mindre prestandastudie som kort förklarar olika tekniker att minska belastningen av spelet.

3.1. Introduktion

Genom att reda ut vilken genre AO tillhör kan en begränsning av vilka tekniker som är tillämpbara lättare göras. Tyvärr finns det ingen genre som passar spelet perfekt, men några har flera egenskaper gemensamma med AO. Främst passar spelet in i genren *labyrintspel*. Denna genre tillhör den mer omfattande genren *actionspel* som också passar AO då snabba reflexer och kort betänketid ligger i fokus. I ett labyrintspel ska spelare ta sig igenom en labyrint vars banor ofta är fyrkantiga. Spelarnas karaktärer har ofta en viss hastighet och dessa karaktärer kan ofta kollidera med varandra, så det gäller för spelarna att vara kvicktänkta [13]. I AO skapas labyrinten av spelarna själva men den stora skillnaden är den totala friheten över hur spelare navigerar sig runt banan jämfört med typiska labyrintspel som till exempel “Pac-Man” [13]. *Pathfinding* används ofta för att lösa labyrintproblem där *Dijkstras algoritm* och *A*-algoritmen* är de mest använda [14][15]. Studien går djupare in på pathfinding och dess algoritmer vid rubrik 3.2.

3.2. Kravidentifiering

För att ta fram olika beteende- och prestandakrav har en deklarativ beteendedesign utnyttjats. Det innebär att kravidentifieringen har varit målorienterad där det inleds med att fastställa övergripande mål för spelets AI för att sedan, utifrån målen, strukturera upp mer konkreta krav [8, kap 1.3 sid 29] Detta ger en bra abstraktion och ser till att viktiga grunder för spelets AI inte blir förbisedda.

3.2.1. Övergripande mål

För att vinna spelet krävs det att boten är ensam överlevande. Alltså är det kritiska för att vinna spelet att överleva. Det är då naturligt att definiera vad en mask kan dö av under spelets gång:

- **Maskar:** Alla utritade maskar som befinner sig på banan dör en mask av att nudda. En spelare kan alltså förlora genom att styra in sin masks huvud i sin egna mask.
- **Banans kanter:** En mask dör av att nudda banans kanter.
- **Powerups:** Det finns ingen powerup som direkt dödar en mask men vissa kan öka

dödsrisken gravt genom att till exempel öka maskens hastighet när nästan hela banans yta är täckt.

Nästa kritiska moment, efter att överleva, är att kunna spela offensivt. I detta spel är konceptet *fly eller fäkta* i centrum och för att vara ensam överlevande är det ibland viktigt att försöka eliminera sina motståndare. Olika sätt att lyckas med detta är följande:

- **Tvär sväng:** När en annan mask är nära boten, speciellt när maskarna åker sida vid sida, så kan en tvär snabb taktisk sväng döda motståndaren lätt och effektivt.
- **Stänga flyktvägar:** En spelare som befinner sig i ett stort utrymme kan stänga till möjliga vägar till detta utrymme för att indirekt döda sitt motstånd.
- **Powerups:** En väl aktiverad powerup kan orsaka massdöd åt ens motståndares maskar.

3.2.2. Botarnas beteendekrav

Dessa beteendekrav har formats utifrån de övergripande målen:

- **Navigering:** Detta är ett grundläggande krav då en bot måste kunna navigera genom spelbanan dels för att undvika att dö, dels för att kunna eliminera motståndare.
- **Planering:** För att botar inte ska råka ut för trattproblemet som nämnts i *kapitel 2.4* måste de kunna planera sin färd i någon utsträckning. Inte bara för att undvika direkta återvändsgränder utan även för att ha långsiktiga mål, som kan vara till botens fördel, att navigera sig mot.
- **Naturlighet:** För en bättre spelupplevelse är det viktigt att botar upplevs naturliga och känns människolika. Ju mer människolik en bot är desto roligare är den att spela mot. [8 kap. 1.5]
- **Oförutsägbarhet:** Denna punkt är främst kopplad till det psykiska elementet i spelet där spelare till exempel kan undvika att aktivera powerups de är på väg mot och därmed lura sina motståndare. Detta är starkt kopplat till kravet om naturlighet ovan då ingen människa är fullt förutsägbar i sitt agerande. Boken “AI Game Programming Wisdom 3” rekommenderar att utvecklare bör undvika en oförutsägbar AI då en mänsklig spelare kan tröttna på att aldrig lista ut hur sin motståndare bör kontras [7 kap 2]. I spelet AO är däremot de oberäkneliga elementen *A och O*. Charmen i spelet är det kaotiska och att varje spelomgång är en unik upplevelse. Dessutom tas bokens exempel upp i koppling till strategispel där spelens omfattning innebär att en oförutsägbar AI kan vara helt omöjlig att besegra [8 kap. 1.5].
- **Situationsanpassning**: Powerups varierar från att knappt ändra speldynamiken överhuvudtaget till att assistera spelare till omedelbar vinst. Spelets AI bör därför kunna agera efter närvaro av powerups och dess påverkan. Om en bot exempelvis har ett stort öppet område för sig själv så kan det vara taktiskt dumt att aktivera en powerup som rensar banan från utritade maskar. En bot måste också kunna anpassa sig efter hur andra maskar påverkar banan. Det är till exempel viktigt att välja när boten bör fly eller fäkta, vilket beror på hur spelbanan ser ut och motståndarnas tillstånd.

3.2.3. Prestanda

Det är självklart viktigt att implementationen av AI inte påverkar prestandan till sådan nivå att spelet blir ospelbart. Vilken gräns som som sätts för prestandan beror helt på hur viktig spelets AI är. I AO ses botarna som alternativa motståndare istället för människor. De ska utföra samma kommandon som en människa kan och för att hålla det simpelt är prestandakravet gällande AO följande:

Om spelet flyter på perfekt mellan flera mänskliga spelare så ska inte någon prestandaförändring märkas av då en spelare ersätts med en bot.

3.2.4. Prioritering

Att fullt implementera samtliga krav för en AI till AO är tidskrävande och inte rimligt genomförbart inom denna rapports tidsspann. På grund av detta, samt av strukturella skäl, så är det bra att ha en tydlig implementationsprioritering. Dessutom är det fördelaktigt att de viktigaste kraven implementeras först så spelare får en bra upplevelse redan i de tidigaste versionerna av AO med AI. Följande är den framtagna prioriteringslistan med korta motiveringar till positionsvalen.

1. **Navigering:** Utan en fungerande grundläggande navigering finns ingen AI överhuvudtaget.
2. **Prestanda:** När navigeringen fungerar bör det alltid regelbundet kontrolleras att prestandanpåverkan hålls inom rimliga gränser. Om detta inte görs så riskeras framförallt att genomföra vidareutvecklingar på implementationer som senare upptäcks vara flaskhalsar prestandamässigt.
3. **Naturlighet:** Detta är den viktigaste delen i att få boten människolik. När detta är implementerat har man en första AI-version färdig.
4. **Planering:** Resterande två punkter bygger en aning på detta och det är då naturligt att detta implementeras först.
5. **Situationsanpassning:** Att reagera på powerups och att försöka förgöra sitt motstånd är en tydligare mänsklig förmåga än att agera oförutsägbart och är därför prioriterad.
6. **Oförutsägbarhet**: Denna punkt är minst viktig då spelets slumpmässiga karaktär kommer göra det svårt att förutspå botarnas agerande även om implementationen inte garanterar oförutsägbarhet.

3.3. Kända koncept och tekniker inom AI

3.3.1. Olika skolor av AI

Många som har spelat dataspel har nog vid något tillfälle upplevt att sin datorstyrda motståndare fuskar. Detta är inte bara en känsla utan i många spel är det faktiskt så att sin motspelare är implementerad på sådant sätt att det inte går att tolka som annat än regelrätta fusk. Ett exempel på detta är strategispel där mänskliga spelare inte ser hela spelbanan från början medan botar ser allt. Dessa typer av AI brukar anses tillhöra den *gamla skolan* [7]*.* Grundprincipen är att låta datorn fuska så mycket den kan, men se till att inte bli upptäckt [8]. Lyckas denna princip följas

blir resultatet, på kort tid, en AI som upplevs mer intelligent än vad den verkligen är. Den *nya skolan* handlar i motsats till gamla skolan om att skapa en AI som inte fuskar. Istället fokuseras att verkligen göra botarna så intelligenta som möjligt [8]. Denna skola, som namnet antyder, är modernare och blir allt mer populär då den tillåter större utvecklingsmöjligheter [7].

3.3.2. Fält

Fält är en typ av underliggande datatyp som beskriver spelbanan på ett sådant sätt att botar kan navigera genom spelbanan baserat på fältet. Det vanligaste är att man delar upp spelbanan i en karta. Kartor kan vara av olika typer men den enklaste typen av kartor är rutnät. Principen är dock densamma för alla karttyper. Varje punkt i kartan beskriver ett område specifikt område av spelbanan. Det värde som sparas i punkterna är spelspecifika och beror dessutom av vilken typ av fält som används.

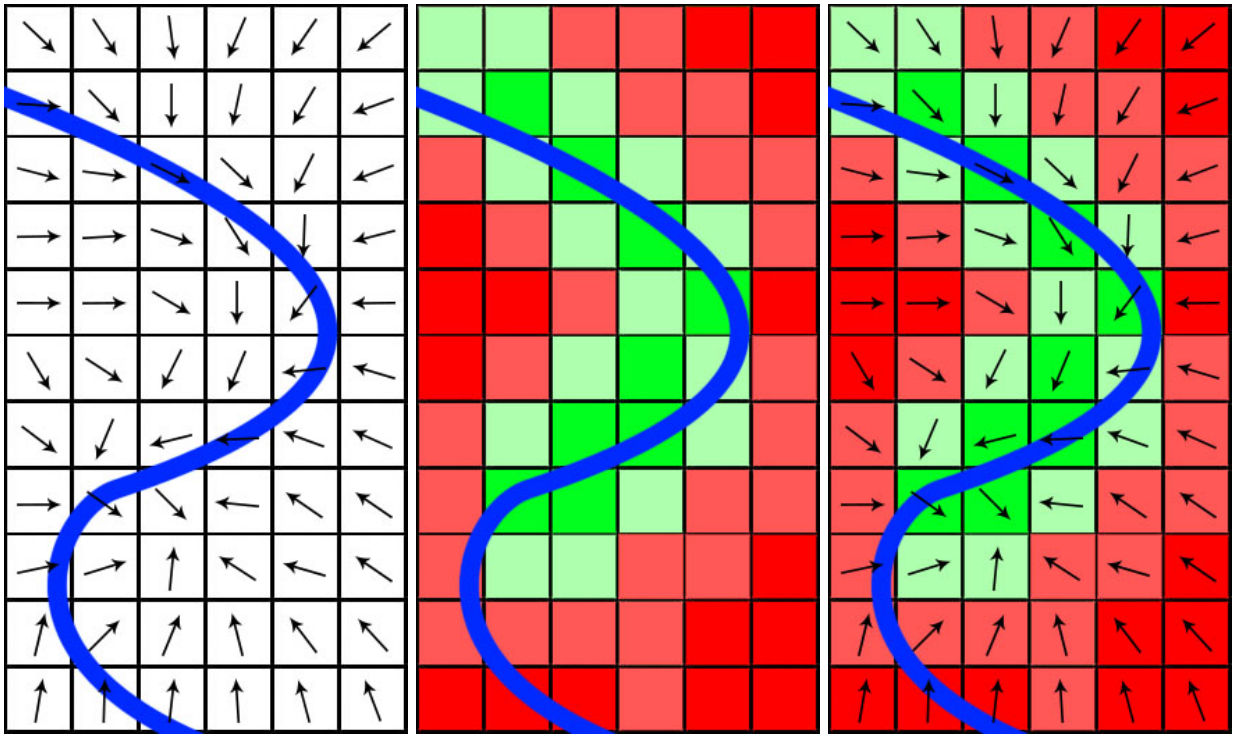
3.3.2.1. Flödesfält

I Flödesfält lagras vektorer i kartans alla punkter som antingen indikerar en attraktion mot punkten eller en repulsion ifrån den beroende på åt vilket håll en bot kommer ifrån. Vektorerna består av en riktning och magnitud. Idén är att vektorerna ska kunna navigera boten genom flödesfältet utan att boten själv behöver hantera sin styrning. På detta sätt kommer botar att repelleras från hinder och attraheras av positiva delar av spelbanan. [8 kap. 3]

3.3.2.2. Potentialfält

Potentialfält är mycket lika flödesfält till sin natur och de båda kan enkelt ersättas med varandra. Skillnaden mellan fälten är att i potentialfält sparas endast ett värde i varje punkt som beskriver om punktens laddning. Ofta används både negativa och positiva värden för att beskriva laddningars relationer, där botar ständigt eftersträvar att navigera genom punkter med höga positiva värden. Potentialfält kan till synes verka oriktade då de ej innehåller riktningsvektorer, men riktningsvektorer kan fortfarande beräknas utifrån cellers värden. [8 kap. 3] Exempelvis kan det tolkas att vektorer från negativa områden pekar mot positiva områden. Se *figur 4* nedan för en illustration av de olika fältens likheter.

Potentialfält kan delas in i tre kategorier: statiska, semistatiska samt dynamiska. Statiska fält är fält som enbart genereras vid ett tillfälle och förblir oförändrade under spelets gång. Semistatiska fält är fält som inte kräver frekventa uppdateringar och brukar ändras med *events*. Dynamiska fält uppdateras frekvent och uppdateras därför med *polling*. De olika fälten kombineras vilket tillsammans bildar potentialfältet. Ett simpelt sätt att göra detta på är att låta alla fältens värden ackumuleras. [18][8 kap. 3]

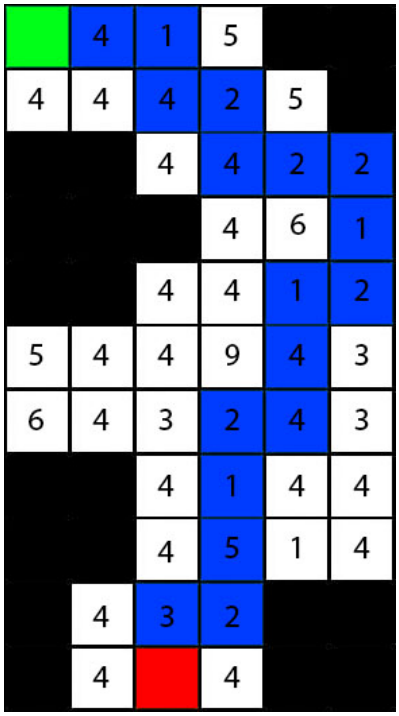


Figur 4: Flödesfält och potentialfält. Bilden till vänster är ett flödesfält. Bilden i mitten är ett potentialfält. Bilden till höger är ett flödesfält där värdesskillnaderna tolkas som vektorer. Det blåfärgade strecket representerar en tänkt bot som navigerar efter fälten. Gröna områden ska tolkas som positivt laddade och röda som negativt laddade. Intensiteten av färgerna anger styrkan av laddningarna.

3.3.3. Pathfinding

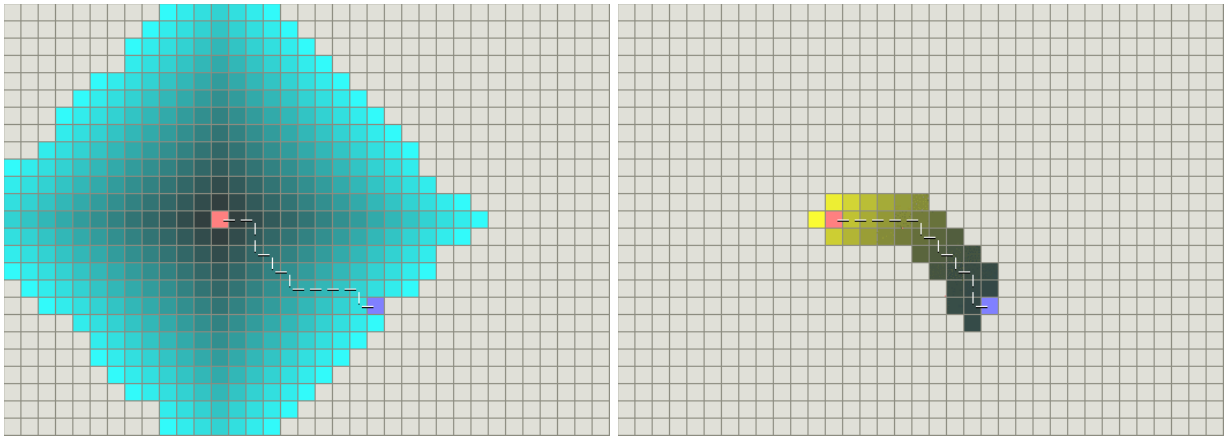
Pathfinding är en grundläggande komponent för navigationsbaserade botar. Pathfinding är en slags sökteknik för att hitta en optimal väg mellan två punkter på en spelbana. Med hänsyn till botkrav *3.2.4:4 Planering* inses snabbt att pathfinding är en kritisk teknik för att satisfiera kravet. Som tidigare nämnt används pathfinding flitigt för att skapa AI till labyrintliknande spel [14][15].

Inom pathfinding föredras ofta att beskriva spelbanan som en graf, för att kunna applicera grafsökningsalgoritmer [7 kap 4]. Grafen kan skapas på många sätt, och som redan nämnt är ett enkelt och populärt sätt är att beskriva spelbanan på som ett rutnät. Vad för slags information som lagras i varje punkt är spelspecifikt men kritiskt för pathfindingalgoritmer är att kunna avgöra om en ruta går att bevandra eller ej. Alltså bör varje ruta i det minimala fallet spara information om den del av spelbanan punkten representerar går att bevandra. Ofta sparas även ett värde som anger hur kostsamt det är för botar att bevandra vissa punkter. De delar av spelbanan som är täckta av hinder går till exempel ej att bevandra. [7 kap 2] Se *figur 5*.



Figur 5: Pathfinding: Exempel på pathfinding där de blå punkterna demonstrerar den billigaste vägen från grön till röd punkt då punkternas värden som passeras summeras. Svarta områden representerar hinder.

Eftersom AI inom spelutveckling har begränsningar, komplexitetsvis samt beräkningsvis, har många grafsökningsalgoritmer utvecklas för att passa bra till olika typer av spel. Den mest kända och använda algoritmen är A* (uttalas “A-star” på Engelska), och de flesta spelspecifika grafsökningsalgoritmerna baseras på A*. Algoritmen utför en *best-first* sökning för att ta reda på den minst kostsamma vägen mellan två punkter på en karta. A* är en vidareutveckling av Dijkstras algoritm. Den största skillnaden mellan de två algoritmerna är att Dijkstras algoritm söker sig jämnt utåt från startnoden, vilket resulterar i att många onödiga punkter kontrolleras. A* riktar istället sökningen så att endast de punkter med stor chans att leda till målnoden kontrolleras, se *figur 6*. [7 kap 5.3] A* använder alltså någon typ av heuristik för att välja punkter att kontrollera, vilket gör algoritmen mer effektiv än Dijkstras algoritm [10].



Figur 6: En jämförelse mellan antalet punkter kontrollerade vid användning av Dijkstras algoritm (till vänster) och A* (till höger). Startpunkten i bilderna är röda och målpunkten är lila.

Ett alternativ till A* är algoritmen IDA*, som är mer minneseffektiv. Algoritmen utför rekursivt begränsade sökningar av typ *djupet-först*, där begränsningarna minskar vid varje iteration tills målpunkten hittats. Då algoritmen ej sparar de punkter som kontrollerats är minnesanvändningen låg jämfört med A*. Däremot finns det inget sätt för IDA* att se till att redan kontrollerade punkter ej kontrolleras igen, och vid varje iteration finns en stor risk att IDA* kontrollerar redan kontrollerade punkter, vilket innebär stora prestandaförluster. Detsamma gäller för algoritmen SMA* som genom att endast spara intressanta punkter minskar minneskonsumptionen. Dock lider SMA* av precis samma problem som IDA*, och passar därför endast bra att använda då minne är begränsat. [17 kap 5]

Sökalgoritmen “Fringe search” är en vidareutveckling av IDA* där mer minne används för att förbättra algoritmens prestanda. Algoritmen ser till att djupet-först-sökningarna ej startar från startpunkten vid varje iteration, utan startar istället där föregående söknings slutat. Detta innebär att antalet punkter som blir kontrollerade flera gånger minskar avsevärt, vilket ökar prestandan. Detta innebär att “Fringe search” är ca 25 % snabbare än A*, men kräver också avsevärt mer minne än A*. [17 kap 5]

De ovan beskrivna algoritmerna kan grupperas som *diskreta* algoritmer [17 kap 5]. Diskreta grafsökningsalgoritmer är specialiserade på att hitta en lösning till ett givet grafsökningsproblem, vilket innebär att algoritmerna i normala fall ej delar information mellan sökningar. Dessa algoritmer belastar endast systemet vid framtagningar av problemlösningar. Dock måste en ny lösning beräknas då grafen (spelplanen) ändras. Om grafen ändras frekvent resulterar det i att nya lösningar beräknas kontinuerligt, vilket kan belasta systemet tungt. I dynamiska spelmiljöer bör därför istället *kontinuerliga* algoritmer användas, då dessa delar data mellan sökningar vilket ökar prestandan avsevärt. I vanlig ordning nyttjar de kontinuerliga algoritmerna istället mer minne för att öka prestandan. [17 kap 6] Att välja rätt algoritmtyp blir en fråga om vilket typ av resurs, beräkningskraft eller minne, som är mest begränsad i det system spelet ska köras på.

De vanligaste kontinuerliga algoritmerna som används inom spel är alla inkrementiella sökalgoritmer. Det innebär att en utförlig sökning först görs (precis som en diskret algoritm) och sedan utförs omplaneringar av sökningen varteftersom grafen ändras. Det innebär att algoritmerna alltså endast planerar om sökningen efter de ändringar som skett i grafen, vilket återanvänder den redan planerade vägen. Detta kan ses som en slags uppdatering av lösningen med hänsyn till ändringar av grafen. De populäraste inkrementiella algoritmerna inom spelutveckling är LPA* samt D*lite, som är en vidareutveckling av LPA*. [17, kap 6]

3.3.4. Beteendesystem

När botarna kan styra genom banan på ett sådant sätt att de håller sig vid liv på ett naturligt sätt (botkrav *3.2.4:1 Navigering* samt *3.2.4:3 Naturlighet*), är det dags att boten även ska kunna fatta taktiska beslut. Detta kallas inom för beslutsfattande[6] och görs ofta av ett beteendesystem. Det är beteendesystemet som i huvudsak kommer att satisfiera botkrav *3.2.4:5 Situationsanpassning* och *3.2.4:6 Oförutsägbarhet*. Beteendesystemet kommer även vara det system som avgör när vägar bör hittas med pathfinding och därför satisfierar även systemet tillsammans med pathfinding botkrav *3.2.4:4 Planering*.

Beteendesystemet består av tre delsystem; uppfattningssystem, beslutssystem samt agerandesystem. Uppfattningssystemet ser till att samla den data som behövs för att beslutssystemet ska kunna fatta beslut. Agerandesystemet utgår ifrån det beslut som beslutssystemet fattat och ser till att ta beslutet tas i kraft genom att utföra nödvändiga ändringar eller skapa events.

3.3.4.1. Uppfattningssystem

Innan själva besluten kan tas, måste först data samlas in från spelet. Det är uppfattningssystemets uppgift att samla och förse beslutssystemet med denna data. Det finns två dominerande underliggande tekniker för att samla data. Dessa två tekniker är *eventdrivna system*[7] samt *pollingbaserade system*[8]. [7, kap 17] [7, kap 19]

Ett eventdrivet system är ett system där objekt kommunicerar genom att notifiera varandra med events och pollingbaserade system pollar låter istället objekt polla varandra efter ändringar. Pollingbaserade system är ett mer passivt sätt att förse data till beslutssystemet. Detta system innefattar oftast någon slags datastruktur som frekvent uppdateras [7 kap 19 sid 373]. Beslutsystemet läser sedan i det område den finner intressant för att bilda sig en uppfattning om spelets nuvarande tillstånd, för att slutligen fatta beslut.

3.3.4.2. Beslutssystem

När uppfattningssystemet samlat nödvändig data är det dags för beslutssystemet att bestämma hur boten ska agera utifrån denna data. Det är i detta system som den långsiktiga planeringen kommer att ske. Långsiktig planering innefattar exempelvis att sätta ut mål som boten vill ta sig

[6] Översättning av den Engelska termen *decision-making.*
[7] Översättning av den Engelska termen *Message-based system.*
[8] Översättning av den Engelska termen *Location-based information system.*

till på spelbanan, bestämma strategier, situationsanpassa nuvarande planer, etc. När beslutssystemet kommit fram till ett beslut, informeras agerandesystemet om beslutet som förväntas se till att besluten tas i kraft.

3.3.4.3. Agerandesystem

När beslutssystemet fattat ett beslut om hur boten ska agera är det dags för agernadesystemet att se till att dessa aktioner utförs. Om beslutssystemet exempelvis beslutat att boten ska navigera sig till ett viss punkt på spelbanan (ett mål) så är det agerandesystemets uppgift att sköta den faktiska navigeringen. Det innebär att lägga upp en plan för hur boten ska ta sig till målet, undvika hinder, röra sig naturligt, etc. Detta system är alltså kritiskt för att boten ska verka smart och naturlig. Det spelar exempelvis inte så stor roll om beslutssystemet fattat smarta beslut, om boten inte klarar att utföra dessa beslut på ett bra sätt. [7, kap 2]

3.4 Prestanda

3.4.1. Kartor

En uppenbar fördel med att spara spelets tillstånd i en karta, är att alla spelobjekt kan dela på kartan. Beräkningarna som krävs för att konstruera den data som behövs för AI att fatta beslut kan alltså centraliseras och behöver endast göras en gång, oberoende av hur många botar som är aktiva [7, kap 19 sid 375]. Kartor möjliggör även ett enkelt sätt att dynamiskt ändra balansen mellan prestanda och uppfattningssystemets noggrannhet. Då antalet punkter i en karta är direkt kopplat till prestanda samt noggrannhet, kan spelet själv justera detta vid olika datorers prestanda. Om spelet spelas på en lågpresterande dator kan spelet minska antalet rutor, vilket ökar prestandan men minskar noggrannheten, och tvärt om. Den vanligaste typen av kartor är rutnät, då de är mycket enkla att implementera [3].

3.4.2. Event vs polling

AI bygger på beslut utifrån information. Därför är det viktigt att välja rätt teknik att samla information för belasta spelet så lite som möjligt. De två mest använda teknikerna för objekt att kommunicera kallas för event samt polling. [7 kap 2] I Eventbaserade system notifierar objekt själva andra objekt vid händelser. Detta innebär exempelvis att om $objekt_A$ måste veta positionen av $objekt_B$ för beslutsfattande, så notifierar $objekt_B$ $objekt_A$ vid positionändringar. Den andra tekniken, polling, låter istället alla objekt ständigt läsa av varandra för att ta reda på om objekten ändrat tillstånd. Då skulle i exempel innan $objekt_A$ istället hela tiden kontrollera $objekt_B$ om dess position har ändrats. Om $objekt_B$ ändrats utför $objekt_A$ beslut efter ändringarna och återgår efter det till att ständigt kontrollera $objekt_B$ om ändringar. [7 kap 17]

Ur ett spelutvecklingsperspektiv kan det vara skönt att abstrahera det ständiga kontrollerandet genom att istället bli notifierad vid ändringar av spelets tillstånd. Detta är dock subjektivt och varierar med utvecklares programmeringsfilosofi [7, kap 17]. Däremot kan prestanda förbättras genom att använda rätt teknik vid rätt tillfälle. Att använda polling för objekt som ändras sällan är ineffektivt då det utförs onödiga instruktioner vid varje cykel, samtidigt som det skapas onödig

overhead när events skapas för ändringar som sker frekvent. Generellt sett lämpar det sig alltså att använda polling för ständigt ändrande värden, medan events lämpar sig för ickefrekventa ändrande värden. [7 kap 2 sid 39]

Anledningen till detta förklaras lättast med ett exempel. Ponera att ett spel innehåller n stycken $objekt_A$ som måste veta när ett $objekt_B$ ändras. I det fallet då $objekt_B$ ändras sällan, är det onödigt att alla n stycken $objekt_A$ kontrollerar om $objekt_B$ ändrats (polling). I det fallet kommer alltså n stycken kontroller att ske vid varje cykel. Om spelet istället utnyttjar events, så kommer endast spelets eventmotor istället att kontrollera om $objekt_B$ ändrats. Vid en ändring av $objekt_B$ kommer eventmotorn att skicka ut notifikationer till alla n $objekt_A$. I detta fall ser vi att endast 1 kontroll sker varje spelcykel, och n notifikationer sker då $objekt_B$ ändras. Spelets prestanda har alltså förbättrats då spelet vid varje cykel minskat antalet kontrolleringar från n till 1. Däremot sker istället n notifikationer vid en ändring av $objekt_B$. Notifikationsinstruktioner är generellt mer kostsamma än kontrollinstruktioner, men även om de två instruktiontyperna skulle belasta systemet lika mycket så har den eventbaserade tekniken utfört $n+1$ instruktioner medan pollingtekniken endast gjort n instruktioner om $objekt_B$ ändras vid varje cykel. Det innebär att spel alltid tjänar prestanda på att använda polling i de fall då $objekt_B$ ändras vid varje cykel. I de fall då $objekt_B$ ej ändras varje cykel krävs en mer noggrann analys, för att bestämma vilken teknik som bör användas, vilket ges i bilaga *9.3 Analys av event vs polling*. Resultatet av studien lyder som följande:

Om $f > w$, $n > \frac{f}{f-w}$ gäller så bör ett eventbaserat system användas. Annars bör ett pollingbaserat system användas.

Konstanten f definieras som antalet spelcykler mellan ändringar av $objekt_B$ där, och konstanten w beskriver hur många gånger mer belastande en notifikationsinstruktion är än en kontrollinstruktion. Det bör tydliggöras att enligt de givna definitionerna gäller $f \geq 1$ samt $w \geq 1$.

3.4.3. Uppdelning av beräkningar

Då AI generellt är mycket belastande för datorers processorer kan en kritisk del av att utforma en bra AI vara att se till att beräkningarna görs så effektivt som möjligt. En teknik för att utföra beräkningar effektivare är att göra dessa beräkningar i olika trådar distribuerade över datorns processorer. Detta är dock en avancerad uppgift som ställer höga krav på utvecklare då detta gör AI-systemet mer komplext och mindre lätt att felsöka. [8 kap 1.6]

En annan teknik för att reducera belastningen av AI inom spelprogrammering är att dela upp beräkningar över flera cykler. Det innebär att pausa beräkningsintensiva algoritmer då de överstigit sin tidsbegränsning för en cykel, och fortsätter beräkningarna nästa cykel. Detta upprepas tills beräkningen är fullbordad. Exempelvis kan pathfinding väljas att beräknas över flera cykler då spelet knappt ändrats under en cykel. Dock är även detta en komplex uppgift att

utveckla en schemaläggning av sådana beräkningar. Därför bör först en analys först göras av den algoritm som ska beräknas över flera cykler, för att se om det är värt prestandavinsten. [17, kap 4.3.1]

4. Utformning av AI till Achtung Online

Det här kapitlet blir länken mellan litteraturstudien och implementationen. Första kapitlet 4.1 diskuterar kring vilken skola spelet bör följa samt prestandaprioriteringar. Huvudkapitlet 4.2 redogör hur spelets AI ska konstrueras och återkopplar till de botkrav som listats i kapitel 3.2. En mindre identifiering av potentiella problem med resultatet ges i kapitel 4.3.

4.1. Introduktion

Innan utformningen kan börja, måste det först fastställas vilken form av AI som ska följas. Då AO är ett klassiskt spel som inte kommer få uppleva några större konceptuella förändringar på lång tid passar den gamla skolans form av AI att tillämpa. Problemet med att följa gamla skolan är att det inte riktigt går för botarna att fuska utan att bli upptäckta i spelet. Hastighetsskillnader och olika svängradier går lätt att upptäcka. Eftersom spelbanan är fullt synlig för samtliga spelare är det lätt för spelare att upptäcka om botarna vet vart powerups kommer dyka upp i förväg. Det finns helt enkelt inga naturliga sätt för botarna att fuska oupptäckt på. En AI till spelet måste därför följa principerna enligt den nya skolan och skapa en AI som är så intelligent som möjligt.

Eftersom spelets målplattform är persondatorer och spelet redan är CPU-krävande (många tunga matematiska operationer) så kommer prestanda ständigt vara i fokus. Då en studie utförd av spelföretaget “Valve” visar att mer än 96% av de persondatorer som kör deras spel har minst 2 GB primärminne så kommer AO alltid att prioritera prestanda över minne [19].

4.2. Användning av kända koncept inom AI

På grund av den dynamiska strukturen av AO skulle det vara svårt att göra en tillståndsmaskin som på ett bra sätt bestämmer hur masken ska navigera utifrån spelbanans utseende. Dessutom skulle denna tillståndsmaskin bli väldigt komplex, då det ej finns ett självklart agerande vid flera situationer. Exempelvis kan en tillståndsmaskin passa som AI för skjutspel där det alltid är bra att skjuta mot fiender, där det enkelt skulle kunna implementeras så att botarna skjuter så fort en fiende dyker upp. Sådana enkla situationer existerar inte i AO. En exakt bedömning av spelbanan, för att ta ett beslut, hade krävt kontroller av en oerhörd mängd parametrar. Att göra en sådan total bedömning av spelbanans utseende hade teoretiskt varit det optimala, men skulle vara allt för prestandatungt att utföra. Dessutom skulle det sätta stor press på utvecklare att utveckla en sådan exceptionellt komplex bedömningsmodell. Istället inses att självreglerande system passar bättre till AO. Därför kommer ett potentialfält ligga som grund för botarnas AI. Det som framförallt är bra med ett potentialfält är att det tillgodoser flera botkrav på samma gång. När en bot rör sig genom fältet skapas små naturliga rörelser som löser botkrav *3.2.4:3 Naturlighet*. Ett sådant system löser även botkrav *3.2.4:1 Navigering* då lokal kollisionsdetektering hanteras av fälten. Alla botar kan dela på samma flödesfält vilket är bra för botkrav *3.2.4:2 Prestanda* [3]. Anledningen till att använda ett potentialfält istället för ett flödesfält

är att flödesfält passar utmärkt för objekt som kan bromsa, backa och navigera fritt som exempelvis en människa. I AO har spelare bestämda hastigheter och svängradier vilket innebär att botarna i många fall inte kommer kunna följa vektorerna eller att vektorerna navigerar boten mot hinder.

Att dela upp potentialfältet i olika typer av fält passar bra till AO då det finns flera sorters element i spelet med unika egenskaper. I AO kommer tre delade fält att finnas samt två privata fält för varje bot. De tre delade fälten består av ett kartfält, powerupfält samt maskfält. Då spelbanan ej ändras under spelets gång, bör kartfältet vara ett statiskt fält och därmed endast skapas en gång. Powerupfältet bör vara ett semistatiskt fält då ändringar av fältet endast behöver ske då en powerup läggs till eller tas bort i spelet. Detta sker med tidsintervall vilket gör det tydligt att fältet bör vara eventbaserat. Maskfältet bör vara ett dynamiskt fält då fältet ständigt behöver uppdateras, vilket utförs med polling. Det ena privata fältet ska lösa problem som uppstår kring en masks huvud då en bot ska påverkas av sina kroppar men inte sitt huvud. Dessutom behövs detta fält för att helt satisfiera botkrav *3.2.4:5 Situationsanpassning* så att botarna kan agera offensivt mot andra maskar. Detta uppnås genom att punkterna i dessa fält påverkas olika från bot till bot . Fältet måste vara dynamiskt då det precis som maskfältet behöver uppdateras frekvent. Det andra privata fältet, som ansvarar för botarnas planering, kommer däremot vara semistatiska då planeringen kommer vara för prestandakrävande för att utföras dynamiskt. I AO kommer alltså de statiska fälten endast att beräknas en gång, de semistatiska fälten kommer uppdateras med events och de dynamiska fälten kommer att uppdateras med polling. Dessutom möjliggör uppdelning av fält att kunna stänga av vissa fält under en viss tid. Exempelvis kan maskfältet temporärt stängas av om spelarna för en stund är odödliga.

Botarnas beteendesystem kommer helt att bero av potentialfältet. Uppfattningssystemet kommer endast bestå av potentialfältet samt en lista med befintliga powerups på spelkartan. När beslutssystemet fattat ett beslut, utför agerandesystemet ändringar i botens privata fält så att beslutet appliceras på potentialfältet. Exempelvis kan den beräknade väg som leder till en powerup göras mer attraktiv i fältet, vilket kommer leda till att boten följer vägen. På detta sätt blir det enklare att utveckla botarnas AI då utvecklarna kan läsa av det potentiella fältet som ger en helhetsbild av hur botarna agerar. Dessutom behöver inte ett avancerat system utvecklas som avgör när botarna ska följa potentialfältet eller inte. Det sätter dock press på navigationsalgoritmen, som kommer vara direkt kopplad till hur de andra systemen uppfattas. Exempelvis spelar det ingen roll hur bra vägar agerandesystemet ritat ut på potentialfältet om botarna bristfälligt navigerar sig genom potentialfältet.

Nedan beskrivs lösningar på hur de krav som definierats i sektion *3.2.4 Prioriteringar* ska satisfieras.

4.2.1. Navigering

Genom att ständigt styra mot de punkter, framför huvudet, med högst värde kommer botarna ständigt attraheras av positiva områden på spelbanan samtidigt som de repelleras av negativa

områden. Punkterna bör väljas så att en bot kontrollerar de punkter som befinner sig något framför masken i den riktning den styr mot.

4.2.2. Naturlighet

Prioriteringar för punkterna ser till att botarna inte svänger oerhört och slumpmässigt i stora öppna ytor då de kontrollerade cellerna har samma värde. Potentialfältet, tillsammans med de strikta svängradierna i spelet, bör dessutom automatiskt leda till naturliga svängar genom de föränderliga labyrinterna som skapas. Begränsningar av botars reaktionsförmåga hjälper även maskarna att utföra mjukare runda svängar.

4.2.3. Prestanda

Alla fält som kan bör delas mellan botarna. Andra knep som bör nyttjas är förberäkning av värden. Till exempel går att förberäkna punkters värden utifrån en utgångspunkt istället för att beräkna detta i varje cykel. Detta förklaras lättast med ett exempel. Ponera att en powerup dyker upp på spelkartan med centrum i en punkt med koordinat (10,10). Utifrån denna punkt ska ett värde delas med linjär fördelning åt alla håll. Naturligt är då att beräkna avståndet från centrumpunkten (10,10) till närliggande punkter och sedan beräkna ett värde. För att slippa dessa beräkningar kan istället en tabell användas med alla möjliga punktvärden utifrån utgångspunkten. Då är återstår endast att leta upp rätt punkt i den förberäknade listan, vilket går mycket snabbare än att beräkna värdet.

4.2.4. Planering

Planeringen kommer fungera som ett komplement till navigeringen. Idén är att planeringen ska guida boten till positiva områden samt förutse och motverka att boten åker mot negativa områden. Mer konkret kommer planeringens uppgift vara att detektera återvändsgränder samt avlägsna powerups. Avlägsna powerups definieras som powrups som befinner sig så långt bort från boten att boten ej påverkas av dess positiva fält.

För att detektera återvändsgränder kommer pathfinding att användas med mål att skapa en väg från boten till ett bestämt antal punkter framför botens huvud, se bild xx. Om vägen markant skiljer sig från botens framtida väg så har en återvändsgränd detekterats. Boten bör då styra iväg från området.

Pathfinding kommer även att användas för att hitta vägar till avlägsna powerups. När beslutssystemet gjort en prioritering av vilken powerup på spelbanan som bör tas så kan boten låta pathfinding skapa en stig mellan boten och den valda powerupen. Sedan kan boten följa stigen.

Det är viktigt att välja en pathfindingalgoritm som är snabb hellre än en som alltid hittar bästa vägen, dels då spelet redan är beräkningstungt, dels då spelbanan förändras så frekvent att den väg som är bäst varierar hela tiden. Då botarnas planerade vägar kontinuerligt måste uppdateras passar det bra att använda en kontinuerlig pathfinding-algoritm och därför kommer D*lite att användas för pathfinding. Algoritmen bör modifieras så att den tar hänsyn till det redan

beräknade potentialfältet, så att algoritmen prioriterar att välja vägar vars punkter har högre värden. Detta kan leda till att boten aktiverar powerups på vägen samt att den undviker trånga utrymmen nära väggar där den lätt kan dödas av andra maskar.

Problemet med pathfinding är att även om pathfindingalgoritmen lyckats hitta en stig mellan två punkter på spelbanan så finns en risk att boten ej kommer kunna följa stigen. Detta på grund av spelares styrbegränsningar i AO. Därför måste maskens svängradie vara en extra parameter för att se om det är möjligt för boten att följa stigen, till exempel via någon form av masksimulering. Om en stig är omöjlig att följa måste andra stigar planeras, tills en stig inom maskens navigationsförmågor har nåtts.

4.2.5. Oförutsägbarhet

Som nämnt kommer spelets slumpmässiga form göra det svårt att förutspå hur en bot navigerar. Simpla tillägg som gör det ännu svårare är att se till att boten inte alltid väljer de beräknade bästa beslutet. Till exempel om botar inte alltid aktiverar röda eller blå powerups som de är på väg mot så kan inte spelarna veta om deras mask kommer påverkas av effekten som följer. Samma sak gäller planerade rutter där det i valet ska ingå slumpfaktorer.

4.2.6. Situationsanpassning

Bästa sättet att situationsanpassa botarna är att ha olika eventsystem. När olika powerups tas som påverkat botarna måste ett event skapas som gör att botarna beter sig som de ska. Till exempel då en bot börjar svänga med vinklar om 90 grader av en powerup bör planeringen och navigeringen anpassa sig efter denna helt annurlunda svängradie. Andra situationer som när botar ska försöka döda andra maskar kommer automatiskt lösa sig med hjälp av det potentiella fältet. Denna punkt satisfieras därför främst genom att skapa olika events gällande powerups.

4.3. Problemidentifiering

Att utveckla ett navigationssystem är helt klart en utmaning då det finns många aspekter navigationsmässigt att ta hänsyn till i AO. Då spelet redan är beräkningstungt, kommer det även bli en utmaning att se till att skapa botarnas AI utan att spelets prestanda försämras avsevärt. Det kanske största potentiella problemet med skapandet av en AI som beskriven är att det kommer kräva mycket tid. Dels kommer det ta lång tid att lyckats implementera alla beskrivna system utan stor prestandaförlust, dels kommer det ta lång tid att finjustera botarnas beteende.

5. Implementation

I detta kapitel beskrivs hur implementationen genomfördes. Kapitel 5.1 anger de begränsningar som gjorts i enligthet med rapportens mål. I kapitel 5.2 beskrivs de mest intressanta lösningar som gjorts i samband med implementationen. Sist beskrivs tester och de prestandaförbättringar som utförts efter att botarna uppnått målets krav i kapitel 5.3.

5.1. Begränsningar

Enligt rapportens mål kommer implementationen bestå av en förenklad version av den teoretiska

AI som specificerats i kapitel 4. Implementationen kommer därför fokusera på att satisfiera de tre första botkraven; *3.2.1:1 Navigering*, *3.2.1:2 Prestanda* samt *3.2.1:3 Naturlighet*.

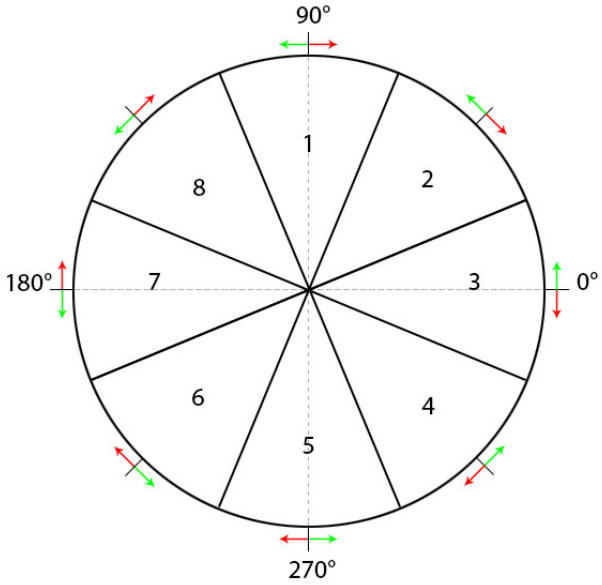
Med hänsyn till de ovan bestämda krav valdes följande att implementeras:

- Ett potentialfält med uppdelade och gemensamma fält för spelbana, powerups och maskar. Enligt specifikation med undantag för de privata fälten.
- Ett enklare navigationssystem.
- En grundläggande eventbaserad situationsanpassning för utvalda powerups med stor påverkan på maskarna eller spelet.

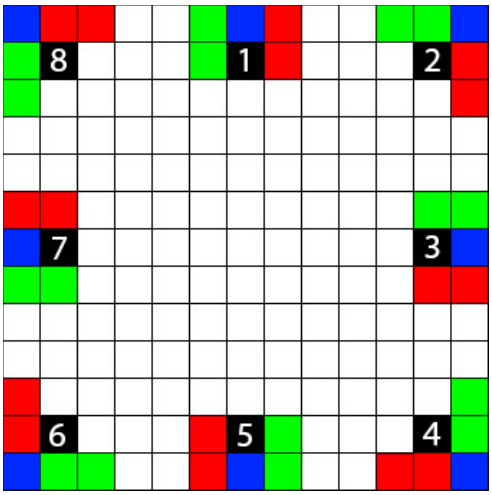
Det ska noteras att alla tre ovanstående punkter ingår i den teoretiska versionen. Mycket fokus låg därför på att programmera med hög abstraktion för att en vidareutveckling av implementationen ska vara möjlig. Implementationen ska därför tolkas som en demoversion av den mer kompletta teoretiskt beskrivna AI.

5.2. Intressanta lösningar

5.2.1. Lokal navigering

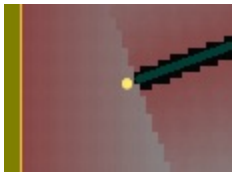


Figur 7a: Lokal navigering. Huvudets riktningar.



Figur 7b: Lokal navigering. Maskens val av punkter.

Ringen i *figur 7a* representerar en masks huvud. De åtta cirkelsektorerna är olika riktningar som masken kan färdas åt. Varje cirkelsektor leder till ett unikt urval av punkter i potentialfältet som kontrolleras av en bot innan den bestämmer om den ska åka rakt fram, till vänster eller höger. I *figur 7b* syns hur urvalet går till. Blå rutor undersöks för att fortsätta rakt fram, gröna rutor för vänstersvängar, röda rutor för högersvängar. Den svarta rutan är den punkt maskens huvuds centrum befinner sig i. Själva undersökningen av punkter är simpel. Den färg innehållande punkten med högst värde väljes som maskens nästa riktning. Blå punkter prioriteras i det fall då färgerna har samma värden. Om en röd och grön punkt delar högst värde gäller en viss speciell regel. Varje tårtbit är även uppdelad i ett grönt och rött område, se pilarna i *figur 7a*. Om maskens riktning befinner sig i gröna halvan av tårtbiten prioriteras gröna celler framför röda. Anledningen till detta syns tydligt i *figur 8* nedan. Botens riktning befinner sig i det gröna området i tårtbit 7. En röd cell kommer inte garanterat ha högre värde i denna situation än en grön cell. Om man betraktar *figur 7b* inses att de däremot har lika högt värde. Prioriteringen av gröna celler leder i dessa situationer till att masken snabbare undviker faran som i detta fall är väggen.

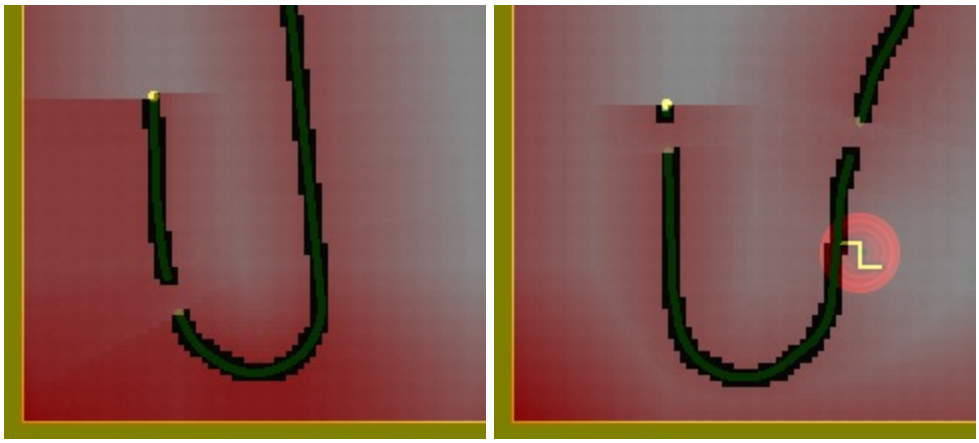


Figur 8: Bot på väg mot vägg

5.2.2. Villkorsbaserade fält

Som nämnt i kapitel 5.1 implementerades demoversionen med hög abstraktion. Detta ledde till framtagningen av ett avancerat villkorsbaserat kopplingsschema mellan fält, vilket innebär att vissa fält prioriteras över andra i vissa situationer. Dessa situationer avgörs med villkorliga operationer, som resulterar i att ett fält helt kan överskrida andra beroende på olika villkor. Följt av

detta kan väldigt avancerade fältsystem skapas. Meningen med schemat är att ange hur punkterna i de olika fälten ska blandas och/eller interagera med varandra. Det var ett problem mellan kartfältet och maskfältet som inspirerade lösningen. Maskfältets punkter fungerar så att punkterna bakom huvudet, i en halvcirkel, försöker ansättas ett värde. Man ser tydligt halvcirkelns avgränsning i *figur 8*. Värdet tillsätts enbart om det är ett starkare (lägre värde) än det som redan finns i detta fält. Detta görs för att undvika komplicerade problem med ackumulering i maskfältets celler som skulle kräva många parametrar för att fungera bra. Problemet dyker upp då maskfältet ska kombineras med kartfältet då det potentiella fältet ska räknas ut. När cellerna kombineras skapas ett mer repellerande område mellan mask och vägg jämfört med mask och mask, se *figur 9* nedan. Istället för att bara lösa detta enskilda problem skapades kopplingsschemat. För att skapa kopplingen anges 2 eller fler fält och vilken form av sammanslagning som gäller då det potentiella fältet beräknas. I detta fall anges att fälten ska använda samma metod som maskfältet använder för beräkning av sina egna celler, att det lägsta cellvärdet mellan fälten ska ignoreras.



Figur 9: Ackumuleringsproblem. Bilden till vänster är utan villkorsbaserade fält.

5.3. Testning

För att lättare kunna undersöka botarnas beteende implementerades funktioner för att visualisera hur botarna tänker under spelets gång. Som redan illustrerats i tidigare figurer ritades potentialfältet ut med olika grader av genomskinlighet beroende på valda inställningar. Funktioner för att göra spelet snabbare, långsammare samt pausa spelet utvecklades också för att lättare kunna följa botarnas beteende under spelets gång. Detta sätt att kontrollera AI är vanlig, då visuell representation av botarnas parametrar generellt sätt är lättare att kontrollera [8 kap 6.2]. Detta kallas för visuell avlusning.

5.3.1. Prestandaförbättringar

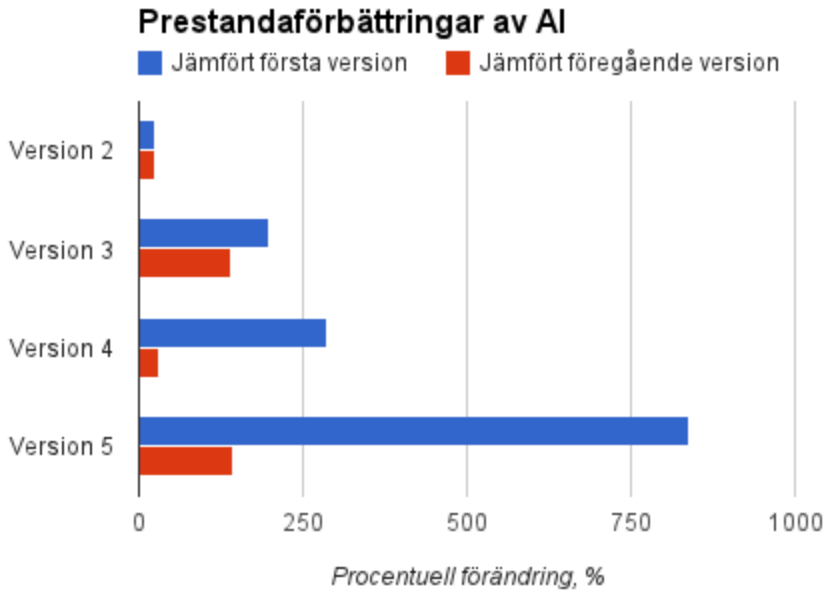
För att undersöka och upptäcka vilka delar av implementationen som var flaskhalsar prestandamässigt användes VisualVM[9]. Följande versionslista går igenom de största

[9] VisualVM är ett visuellt verktyg för att profilera (mäta och analysera olika former av prestanda) en

förbättringar som genomfördes och i *figur 10* kan man utläsa prestandaförändringen.

- **Version 1:** Versionen innan någon prestandaförbättring genomförts. Enbart 65 bilder per sekund uppmättes då sex botar spelade mot varandra, se *bilaga 9.6* för specifikation av dator som användes.
- **Version 2:** För att förhindra att maskarna påverkar det potentiella fältet precis framför huvudet användes kostsamma beräkningar med arcustangens. Detta var för att ta reda på vinkeln från maskens centerpunkt mot en punkt och sedan jämföra denna vinkel med maskens riktning. Dessa ersattes med att skapa en polygon där en kontroll genomfördes för att se till att punkten inte befann sig i inom denna polygon. En prestandaförbättring på cirka 20% uppmättes.
- **Version 3:** Tyvärr var största flaskhalsen nu skapandet av polygonen. Detta ersattes i sin tur av ett förberäknat rutnät där varje vinkel, från en centerpunkt som referens till alla andra punkter, lagrades. Istället för en masks pixelcenter användes nu maskens centerpunkt som referens som använde rutnätet för att direkt hämta vinkeln. Resultat var en prestandaförbättring på 140%.
- **Version 4:** Efter version 3 var den nya flaskhalsen en kvadratrotsberäkning (Pythagoras sats) som beräknade avståndet mellan centerpunkten till andra punkter. Detta användes för att, i nästa steg, beräkna det nya värdet som skulle tilldelas punkter i det dynamiska maskfältet. Lösningen blev detsamma som i version 3 där ett förberäknat rutnät användes. En prestandaförbättring på cirka 30% uppmättes.
- **Version 5:** Sista prestandaoptimeringen som behandlades var beräkningen av punkternas värde i maskfältet som nämndes vid version 4. Problemet var kostsamma exponentiella beräkningar. Även detta kunde förberäknas med ett rutnät och en förbättring på cirka 140% uppmättes som i version 3. Den totala förbättringen mellan version 5 och version 1 uppmättes till hela 840%.

java-applikation. http://visualvm.java.net/

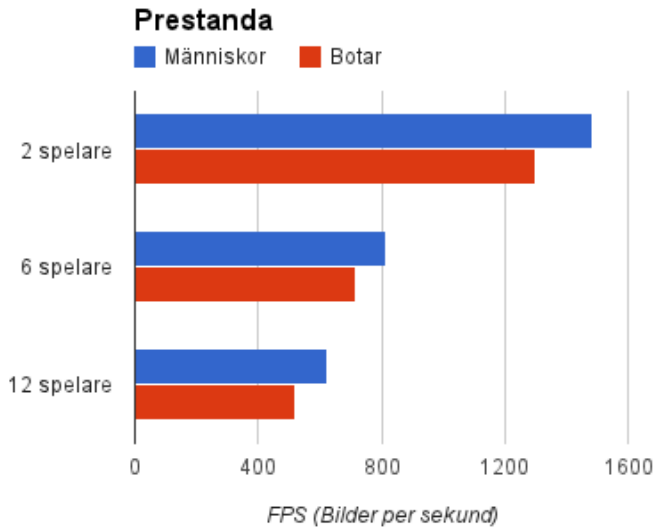


Figur 10: Prestandan mättes genom att jämföra FPS mellan förändringarna. Se *billaga 9.6* för datorns specifikation.

6. Resultat av demo-versionen

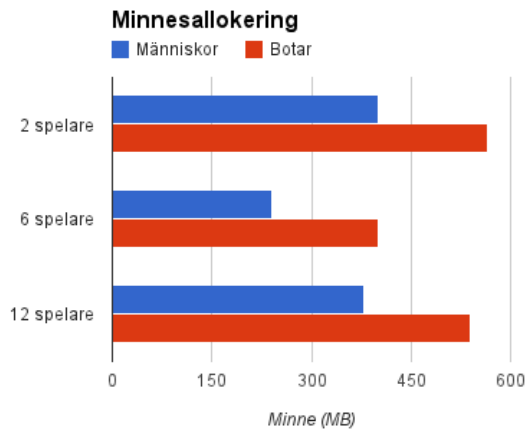
Resultatet kommer först gå igenom vilken prestandapåverkan den AI som implementerats haft på spelet AO. I kapitel 6.2 kommer en allmän beskrivning av hur botarna beter sig presenteras. Denna del kommer innehålla en del godtyckliga bedömningar av rapportskrivarna själva. Detta på grund av att det skulle krävas en väldigt omfattande testning samt studie för att vetenskapligt värdera botarnas beteende och agerande vilket inte ansågs rimligt.

6.1. Prestanda



Figur 11: Prestanda. Se *bilaga 9.6* för specifikation av dator som användes vid mätning.

Under implementationen förbättrades prestandan via modifikationer vid flertalet tillfällen. De med störst inverkan går att läsa om under kapitel 5.3.1. Som graf XXYY visar försämras prestandan med 10-13%. Detta är oberoende av antalet spelare.



.

Graf XZ: Minnesallokering. Grovt uppskattad, ska inte tolkas som en nogrann och exakt mätning.

Minnesallokeringen var svår att mäta då det varierade kraftigt under spelets gång. Rapportskrivarna misstänker detta beror på javas automatiska skräpsamlare. Mätningar under flera minuter gjordes ändå och ett ungefärligt snitt togs fram vilket visas i graf XZ. En total grovt uppskattad försämring uppmättes på 30%.

.

6.2. Allmän prestation av AI

Figur X: AO AI: Figuren visar 3 bilder av samma match där bilden till vänster visar hur en match ser ut för en mänsklig spelare. Bilden i mitten och till höger är en visualisering av hur det potentiella fältet ser ut för

botarna.

I sin helhet fungerar den lokala navigationen bra. Botarna undviker direkt kollision genom att avvika åt lämpliga håll. Kort in i matchen bildas mönster av vita, eller lätt rosa, slingor omringade av en starkare röd nyans som ses längst till höger i figur X. Botarna navigerar sig längs dessa vita längor som representerar potentiella värden nära 0. Då botarna befinner sig i närheten av en powerup navigeras maskarna, precis som planerat, mot dessa. På bilden ser man tydligt hur det potentiella fältet påverkas av powerups. Trattproblemet som beskrevs i kapitel *2.4*. är delvis löst. Då det potentiella fältet är starkare i hörnen undviker botarna att åka dit. Dock finns ännu ingen planering så risken finns att till exempel den röda boten, i figurerna ovan, svänger till höger där den troligtvis kommer åka in i den bruna masken. Det finns visserligen ett hål men det potentiella fältet beter sig inte tillräckligt bra runt hålen för att boten säkert ska kunna navigera sig igenom. Tittar man noga ser man att det är vitare runt hålens ena ände vilket gör att boten ofta träffar denna sida. Däremot är boten väldigt bra på att navigera sig runt trånga utrymmen. Dessutom hanterer botarna påverkan av olika powerups med bra resultat. Trattproblemet, hålproblemet samt hanteringen av powerups visualiseras tydligt i figur XYY nedan.

Figur XYY: Trattproblem. Exempel som tydligt visar trattproblemet men även hur botarna hanterar powerups som tron turn och wall hack.

Flera vänner till rapportskrivarna, som aldrig spelat ADK eller AO tidigare, fick testa en match mot en och flera botar. Till rapportskrivarnas glädje dödades och besegrades de mänskliga spelarna av botarna vid flera tillfällen. Det var tydligt att fler botar ökade svårighetsgraden markant för testarna. Då det endast var en bot på spelbanan besegrades denna lätt efter att testarna vant sig vid spelet. Gränsen då botarna övergick till att vinna flest matcher var vid cirka 6 botar. På grund av det låga antalet testare framtogs ingen exakt statistik.

7. Diskussion

Prestandan av demo-versionen låg helt inom rimliga gränser. Prestandasänkning på nästan bara 10% skulle troligtvis kunna minimeras ytterliggare om mer tid ges åt detta. Minnesallkoeringen visade något högre som misstänkt då vi använder en hel del fördefinera rutnät för att underlätta beräknar, se kap 5.3.1. Dock som nämnt vid kapitel 4.1. är detta inget stort problem då en total minnesanvändning på 600 mb är, i dagens mått, inte orimligt högt. Prestandakravet anses därmed vara satisfierat.

Testerna med de ovana spelare tydde på att botarna var utmanande samt upplevdes mycket roliga av samtliga deltagarna. Detta var huvudmålet med denna demo-version vilket innebär att det var ett mycket positivt resultat. Flera av testarna, samt rapportskrivarna själva, anser att spelets AI liknar mycket botarna i spelet *Achtung, die kurve! Flash remake* som beskrivs i kapitel 2.4. Skillnaden är att botarna i det spelet är betydligt bättre på att åka genom hål. Däremot är det bara i denna implementation som botarna ses rikta sin mask mot powerups. Utifrån dessa uttrycka åsikter anses de uppsatta navigationskravet samt naturlighetskravet vara satisfierade. Rapportskrivarna är mycket nöjda med samtliga resultat av demo-versionen. Målet var även att teoretiska versionen skulle vara utmanande och rolig vilket kan vara svårt att bestämma då den är teoretisk. Utifrån resultatet av demo-versionen måste även detta mål ses som uppnått. Det finns dock inga skäl till att tro att den teoretiska versionen ska vara mindre underhållande och utmanande än demo-versionen.

7.1. Utvecklingsmöjligheter

Då den implementerade versionen av AI utgör grunden för den teoretiska AI som specificerats i kapitel 4, finns det goda möjligheter att påbygga den redan implementerade AI för att uppnå den mer omfattande teoretiskt angivna AI. Prestandan går, som nämnt, att förbättra ytterliggare vilket ger mycket utrymme för vidareutveckling prestandamässigt. Framtida vidareutvecklingar kommer baseras på det potentiella fältet som har visat sig prestera mycket bra. Figur 5 i kapitel 3.3.3. visar tydligt hur passande det potentiella fältet är för planerande pathfindingalgoritmer. Rapportskrivarna har klara idér för vad som återstår att implementeras och ser inga hinder för att den fullutvecklade versionen av AI till AO kommer att kunna spelas inom kort.

7.2. Felkällor

Studien baseras främst på de två böcker [7] samt [8] skrivna av respekterade AI-utvecklare och professorer. Båda böcker har mottagit mycket bra omdömen av spelutvecklare och antas därför vara av god trovärdighet. Ibland utelämnar böckerna vissa detaljer, vilket resulterat i att rapportskrivarna själva dragit slutsatser eller utfört beräkningar. Dessa egna slutsatser och beräkningar kan vara en stor felkälla, men inga kritiska moment bygger på detta i studien. Ibland finns det även motsägelser mellan böckerna, vilket poängteras i rapporten. Dessa motsägelser finnes endast kring subjektiva frågor, vilket annars hade setts som ett varningstecken för böckernas trovärdighet.

Den näst mest använda källan [17] är ett examensarbete. Arbetet har granskats av rapportskrivarna, och då arbetet flitigt använder källor av akademisk kvalitet har en bedömning gjorts att även denna källa har god trovärdighet. Den information som extraherats ur källan har försökts verifierats av andra mindre källor samt böckerna.

Bakgrundskapitlet i rapporten anses av rapportskrivarna vara det kapitel med troligtvis störst felmarginal, då kapitlet ofta baseras på wikipedia [4] samt [5] och egna undersökningar. Informationen i bakgrundskapitlet är dock ej kritisk i studien.

8. Referenser

[1]GameDev, http://www.gamedev.net/ (hämtat 2013-03-14)
[2]AIGameDev, http://aigamedev.com/ (hämtat 2013-03-14)
[3] AI game programming guide, http://www.gameai.com/sites.php (hämtat 2013-03-14)
[4] Wikipedia, http://en.wikipedia.org/wiki/Achtung,_die_Kurve! (hämtat 2013-03-14)
[5] Wikipedia http://en.wikipedia.org/wiki/Snake_(video_game) (hämtat 2013-03-16)
[7] Schwab, Brian (2004)AI game engine programming, Charles river media
[8] Rabin, Steve (2006), AI game programming wisdom 3, Charles river media
[9] AI depot, http://ai-depot.com/GameAI/Design.html (hämtat 2013-03-16)
[10] Wikipedia, http://en.wikipedia.org/wiki/A*_search_algorithm, (hämtat 2013-04-05)
[11] A* pathfinding for beginners, http://www.policyalmanac.org/games/aStarTutorial.htm, (hämtat 2013-04-05)
[12] A* Delling, D. and Sanders, P. and Schultes, D. and Wagner, D. (2009). Engineering route planning algorithms. Algorithmics of large and complex networks. Springer. pp. 117-139. doi:10.1007/978-3-642-02094-0_7.
[13] Wikipedia http://en.wikipedia.org/wiki/Video_game_genres (hämtat 2013-04-05)
[14] Graham, Ross, McCabe, Hugh and Sheridan, Stephen, Pathfinding in computer games, http://gamesitb.com/pathgraham.pdf (hämtat 2013-04-05)
[15] Wikipedia http://en.wikipedia.org/wiki/Pathfinding (hämtat 2013-04-05)
[16]http://theory.stanford.edu/~amitp/GameProgramming/ (hämtat 2013-04-05)
[17]http://takinginitiative.files.wordpress.com/2011/05/thesis.pdf (hämtat 2013-04-07)
[18] AIGameDev, http://aigamedev.com/open/tutorials/potential-fields/ (hämtat 2013-04-07)
[19] http://store.steampowered.com/hwsurvey/

9. Bilagor

9.1 Termer och förkortningar

Nedan följer en lista med förklaringar till termer samt akronymer som används i rapporten.

KTH	Kungliga tekniska högskolan
Achtung online, AO	Ett spel skapat av Lucas Wiener och Mathias Lindblom som är en klon av spelkonceptet “Achtung die kurve”. Det är detta spel som rapporten kommer att referera till som “spelet” om inget annat anges.
Achtung die kurve, ADK, Zatacka	Det spelkoncept som spelet Achtung Online bygger på. Konceptet är oftast känt som “Achtung die kurve”, förkortat ADK, men kan vissa versioner av spelet heter även “Zatacka”.
AI, artificiell intelligens	Någon sorts människoskapt intelligensform som kan tänka och resonera, eller på andra sätt efterlikna en människas intelligens.
bot	En datorstyrd spelare som styrs av AI.
mask, kropp, kurva	Det färgade streck som motståndaren skapar på banan.
singleplayer	Ett låneord från Engelskans “single player” som betyder att spelet endast kan spelas av en enda mänskilg spelare.
powerup	En temporär förändring av spelets egenskaper som kan aktiveras av spelare.
hårdskriptat	Simpelt förklarat innebär det kodats med fokus på att det enbart ska fungera. En något djupare förklaring är att lösningarna inte är dynamiska utan består istället ofta av en massa if-satser
labyrintspel	Spel som påminner eller består av labyrinter. Pac-man är ett klassiskt exempel. Fokus i labyrintspelligger ofta på att ta sig ut eller försöka hålla sig inom en begränsad bana utan att “falla ut” eller dö.
actionspel	Spel med fokus på snabb reaktionsförmåga.
agent	Annat ord för bot, eller datorstyrd motståndare
overhead	Används inom datavetenskap då man menar beräkningar eller andra kostnader som krävs för att kunna utföra det som faktiskt betyder något. Runtomkringkostnader för dataoperationer.
best-first	En sökning genom en graf där man utgår från den troligtvis bästa noden utifrån en fördefinerad regel.
djupet-först	En sökning där man traverserar ett träd genom att söka längst ner i trädet för att sedan “backa”.
trattsituation	När en mask kommer från ett område med utrymme på sidorna som krymper i jämn takt. Till exempel då en mask åker från

	mitten rakt in mot ett hörn på en fyrkantig bana.
event	
polling	
tillståndsmaskin	

9.2 Författarnas samarbete

Båda författare har arbetat tätt tillsammans under hela projektet. Varenda mening i rapporten har lästs och diskuterats av båda, och demoversionen har utvecklats av båda.

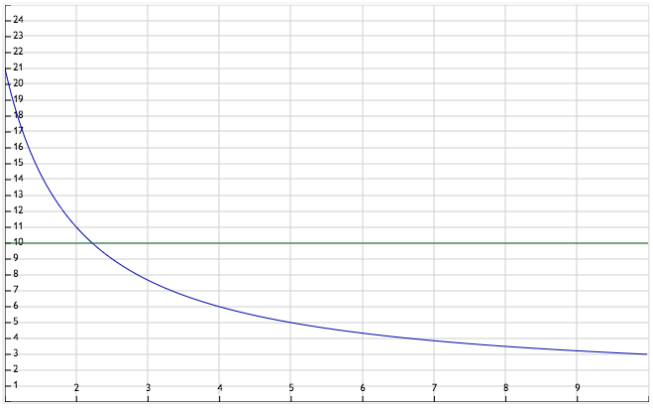
9.3 Analys av event vs polling

Här presenteras en mer utförliga analys av när event eller polling bör användas. Det här kapitlet är en förlängning av *3.3.2. Event vs polling* som bör läsas innan.

Eftersom belastning av kontrollinstruktioner sällan är lika med notifikationinstruktioner, bör en konstant p samt e för belastning av spelet vid kontrollinstruktioner respektive notifikationinstruktioner införas i beräkningarna. Då $e \geq p$ så kan e beskrivas av funktionen $e(p) = w \cdot p$, vilket innebär att e beror av p och vidare blir konstanten w ett mått på hur mycket mer belastande e är jämfört med p. Dessutom måste en kostant f som beskriver ändringsfrekvensen av objekt B tas med i beräkningarna. Mer strikt definieras f som antalet spelcykler mellan ändringar av objekt B. Alltså måste $f > 1$ gälla då objekt B maximalt kan ändras en gång varje cykel. Då det ej går att bestämma ett exakt f måste ett statistiskt värde beräknas, vilket ej kommer behandlas i denna rapport.

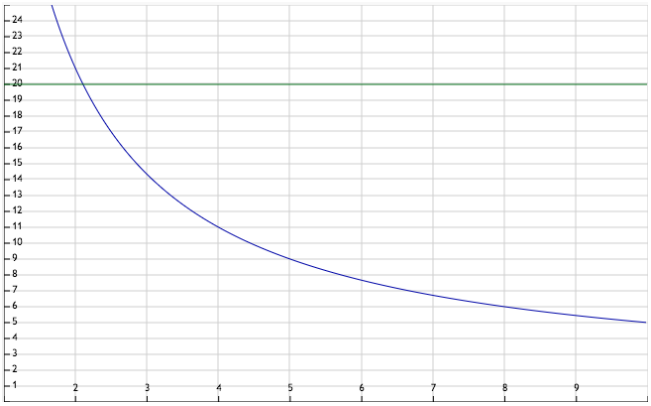
Med de ovan definierade konstanterna erhålles nu de nya belastningsformlerna $B_e = p + \frac{n \cdot e(p)}{f}$ för events samt $B_p = n \cdot p$ för polling, där B är belastningen för de olika teknikerna. Ur formlerna kan direkt utläsas att polling garanterat presternar bättre då $n = 1$ och därför kommer vidare beräkningar att inskränkas till det fall då $n > 1$. Nedan presenteras grafer för olika konstanta värden på n med f varierande. Blå graf representerar B_e och grön graf representerar B_p. Lägre y-värde är bättre i samtliga grafer.

$e(p) = 2 \cdot p,\ n = 10,\ x : f,\ y : B$



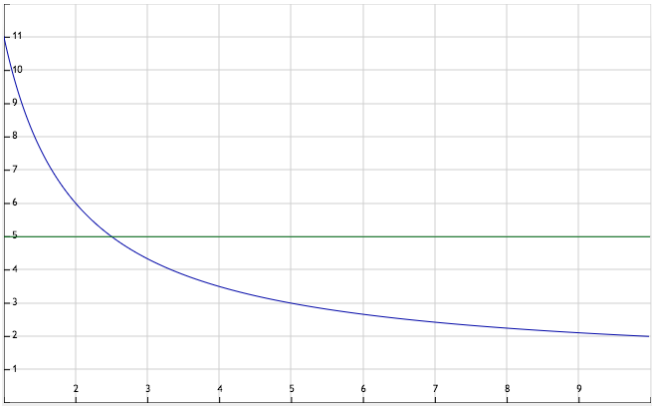
Figur 1: brytpunkt vid ca 2 cykler mellan ändringar.

$e(p) = 2 \cdot p,\ n = 20,\ x : f,\ y : B$



Figur 2: brytpunkt vid ca 2 cykler mellan ändringar.

$e(p) = 2 \cdot p,\ n = 5,\ x : f,\ y : B$



Figur 3: brytpunkt vid ca 2,5 cykler mellan ändringar.

Ur graferna avläses snabbt ett mönster, nämligen att brytpunkten för samtliga grafer ungefär är w. Nedan följer beräkningar för brytpunkten:

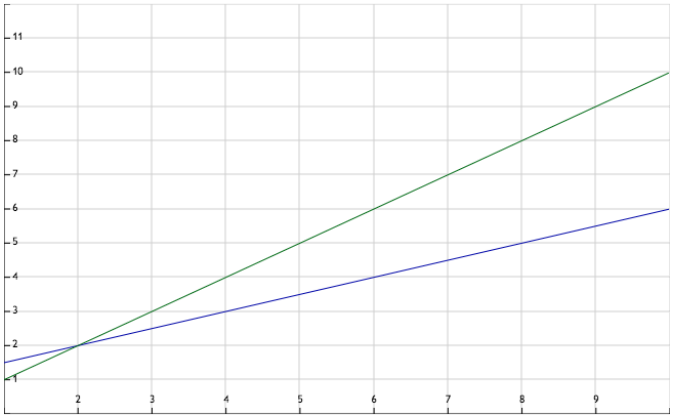
$$B_e = B_p \leftrightarrow$$
$$p + \frac{n \cdot e(p)}{f} = n \cdot p \leftrightarrow$$
$$f \cdot p + n \cdot e(p) = f \cdot n \cdot p \leftrightarrow$$
$$n \cdot e(p) = f \cdot n \cdot p - f \cdot p \leftrightarrow$$
$$n \cdot e(p) = f \cdot (n \cdot p - p) \leftrightarrow$$
$$\frac{n \cdot e(p)}{n \cdot p - p} = f \leftrightarrow$$
$$\frac{n \cdot e(p)}{(n-1) \cdot p} = f \leftrightarrow$$
$$\frac{n \cdot w \cdot p}{(n-1) \cdot p} = f \leftrightarrow$$
$$\frac{n}{(n-1)} \cdot w = f$$

Det ovan identifierade mönstret verifieras då med $f = \frac{n}{(n-1)} \cdot w$, där $1 < \frac{n}{(n-1)} \leq 2$. Vidare kan följande slutsats fastställas:

- $f > \frac{n}{(n-1)} \cdot w$**:** eventbaserat system bör användas.
- $f < \frac{n}{(n-1)} \cdot w$: pollingbaserat system bör användas.

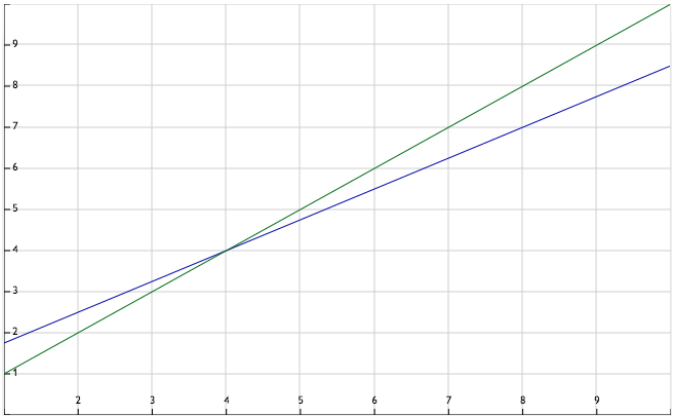
Detta verifieras i graferna nedan:

$e(p) = 2 \cdot p, f = 4, x: n, y: B$



Figur 4: brytpunkt vid 2 objekt A.

$e(p) = 6 \cdot p, f = 8, x: n, y: B$



Figur 5: brytpunkt vid 4 objekt A.

En vidare förenkling av ekvationen kan göras:

$$f = \frac{n}{(n-1)} \cdot w \leftrightarrow$$
$$f = (\frac{1}{(n-1)} + 1) \cdot w \leftrightarrow$$
$$f \cdot (n-1) = (1 + (n-1)) \cdot w \leftrightarrow$$
$$f \cdot (n-1) = w + (n-1) \cdot w \leftrightarrow$$
$$f \cdot (n-1) - (n-1) \cdot w = w \leftrightarrow$$
$$(n-1) \cdot (f - w) = w \leftrightarrow$$
$$n - 1 = \frac{w}{f-w} \leftrightarrow$$
$$n = \frac{w}{f-w} + 1 \leftrightarrow$$
$$n = \frac{w}{f-w} + \frac{f-w}{f-w} \leftrightarrow$$
$$n = \frac{w+f-w}{f-w} \leftrightarrow$$
$$n = \frac{f}{f-w}$$

Slutsatsen är då följande:

Om $f > w, n > \frac{f}{f-w}$ gäller så bör ett eventbaserat system användas. Annars bör ett pollingbaserat system användas.

9.4. A*

Den mest kända och använda algoritmen inom pathfinding är A* (uttalas *A-star* på Engelska), och de flesta pathfinding-algoritmerna i spel baseras på A*. Algoritmen utför en *best-first* sökning för att ta reda på den minst kostsamma vägen mellan två noder i en graf. A* är en vidareutveckling av Dijkstras algoritm. Den största skillnaden mellan de två algoritmerna är att Dijkstras algoritm söker sig jämnt utåt från startnoden, vilket resulterar i att många onödiga noder kontrolleras. A* riktar istället sökningen så att endast de noder med stor chans att leda till målnoden kontrolleras (se figur x). [7 kap 5.3] A* använder alltså någon typ av heuristik för att

välja noder att kontrollera, vilket gör algoritmen mer effektiv än Dijkstras algoritm [10].

A* arbetar i huvudsak med två listor av noder; $lista_{öppen}$ samt $lista_{stängd}$. Den öppna listan innehåller potentiella noder som skulle kunna bestigas, medan den stängda listan innehåller noder som garanterat ej ska bestigas. Då återstår att välja de noder att bestiga som ger bäst resultat (kortast väg i de flesta fall). A* använder en teknik som heter *path scoring* som innebär att varje nod tilldelas ett beräknat kostnadsvärde F, där det eftersträvas att vandra via de noder som har lägst F. Värdet beräknas med $F = G + H$ där G är kostnaden för att ta sig från startnoden till den nod vars F ska beräknas. Intressantast är H som är ett estimerat värde med hjälp av heuristik som anger den estimerade kostnaden att ta sig från nuvarande nod till målnoden. Eftersom den exakta kostnaden att ta sig från en nod till målnoden inte kan bestämmas exakt innan vägen bestämts, måste alltså heuristik användas för att bestämma detta. [11]

A* fungerar i huvudsak som följande:

1. Utgå från startnoden nod_{start} och se till att $nod_s \in lista_{öppen}$.
2. Välj ut den nod $nod_F \in lista_{öppen}$ med lägst värde F och se till att $nod_s \in lista_{stängd}$.
3. Gör följande för alla närliggande noder $nod_i \notin lista_{stängd}$ till nod_F:
    a. Undersök om nod_i ej kan bevandras. Om så är fallet se till att $nod_i \in lista_{stäng}$.
    b. Om $nod_i \notin lista_{öppen}$ så ska värdena F, G samt H beräknas för nod_i och därefter ska nod_F ska läggas till som förälder till nod_i. Sedan ska $nod_i \in lista_{öppen}$.
    c. Om $nod_i \in lista_{öppen}$ redan gäller så ska värdet G_{exst} av den redan existerande vägen till nod_i jämföras värdet med G_{pot} av den potentiella vägen genom nod_F till nod_i. Om $G_{pot} < G_{exst}$ så ska den potentiella vägen användas istället för den redan existerande vägen. Alltså ska nod_F istället registreras som förälder till nod_i.
4. Om $|lista_{öppen}| = 0$ så kan ingen stig skapas från nod_{start} till nod_{slut}.
5. Om $nod_{slut} \notin lista_{stängd}$ så ska algoritmen repetera från punkt 2.
6. Om $nod_{slut} \in lista_{stängd}$ så har en väg hittats. Genom att utgå från nod_{slut} och följa nodernas föräldrar kommer tillslut stigen mellan nod_{start} och nod_{slut} erhållas.

9.5. Element i Achtung Online

9.5.1 Powerups i AO

- **Switcharoonie:** Samtliga maskar byter plats med varandra slumpmässigt. Man kan inte få sin egna plats. **Unik** för AO.

- **Drunk:** Motståndarnas maskar hastighet varierar en stund samt att **masken** upplever problem med att åka rakt fram, likt många människor kring högtider. Unik för AO.
- **Twin:** Masken delar sig i 2 och man står båda samtidigt. En spelare kan dela sina maskar teoretiskt sätt oändligt antal gånger. Samtliga maskar måste dö för att spelaren ska räknas som död. **Unik** för AO.
- **Fast turn:** Spelaren svänger snabbare. **Unik** för AO.
- **Slow turn:** Motståndarna svänger långsammare. **Unik** för AO.
- **Clear:** Hela banan rensas från allt som ritats ut av maskarna.
- **Invisible:** Spelarens maskar ritar inte längre ut och kan åka igenom andra maskar under en viss period.
- **Switch keys:** Motståndaras knappar byter plats ett tag, vänster blir höger och höger blir vänster, deras huvuden ändrar även färg för att man ska reagera på bytet.
- **Tron Turn:** Motståndarnas maskar svänger nu 90-gradigt och huvudena blir fyrkantiga vilket inte alltitid är en nackdel.
- **Wall hack:** Spelarens maskar kan nu åka igenom kartans väggar under en period.
- **Thick:** Motståndarnas maskar blir större.
- **Thin:** Spelarens maskar blir mindre.
- **Speed:** Spelarens maskar blir snabbare.
- **Slow:** Motståndarnas maskar blir långsammare.

9.6. Specifikation av dator vid tester

Typ av dator: Clevo, laptop
CPU: I7 3630QM (4 kärnor)
GPU: Nvidea GTX 680M 4 GB
Primärminne: 16 GB (2x8 Crucial DDR3 1600 MHz)