HÖGSKOLAN SKÖVDE

Institutionen för kommunikation och information
Examensarbete i datavetenskap 20p
C-nivå
Vårterminen 2007

Styrbeteenden för autonoma agenter i trafikmiljö

Peter Billander

Styrbeteenden för autonoma agenter i trafikmiljö

Examensrapport inlämnad av Peter Billander till Högskolan i Skövde, för Kandidatexamen (B.Sc.) vid Institutionen för kommunikation och information. Arbetet har handletts av Mikael Thieme.

2007-06-05

Härmed intygas att allt material i denna rapport, vilket inte är mitt eget, har blivit tydligt identifierat och att inget material är inkluderat som tidigare använts för erhållande av annan examen.

Signerat: _____

Styrbeteenden för autonoma agenter i trafikmiljö

Peter Billander

Sammanfattning

Denna rapport beskriver styrbeteenden för autonoma agenter och utvärderar hur användbar den tekniken är för att skapa flexibla agenter i kontinuerliga miljöer. Arbetet undersöker hur väl det går att använda styrbeteenden i en mer strikt kontrollerad miljö, nämligen biltrafik. De utvecklade agenterna agerar i en trafikmiljö där de navigerar för att följa en väg och undvika kollisioner med andra agenter. Agenterna använder en kombineringsarkitektur som gör det möjligt att använda olika beräkningsmodeller. Agenterna är testade i två scenarion där teknikens användbarhet och flexibilitet är studerad. Resultatet från utvärderingen visar att tekniken är väldigt flexibel men kan kräva mycket finjustering för att uppnå bästa resultat.

Nyckelord: Styrbeteenden, autonoma agenter, simulerad biltrafik, Designmönstret brygga, kombineringsarkitektur.

Innehållsförteckning

1 Introduktion

Artificiell intelligens (AI) är ett område som spänner över flera olika vetenskapliga områden, där varje vetenskap har tillfört nya idéer och tankar. Datavetenskapen har tillfört de verktyg som krävs för att skapa dessa intelligenser. Inom spel används framför allt artificiell intelligens för att simulera intelligens hos datorkontrollerade motspelare. I spel möts spelaren ofta av smarta entiteter av någon form, så kallade agenter, det kan vara fiendesoldater i ett krigsspel eller motspelaren i schack. Flera olika designmönster och tekniker finns för att skapa dessa agenter och teknikerna skiljer sig mycket åt. Det finns avancerade tekniker för att skapa agenter som lär sig av tidigare erfarenheter, till exempel neurala nätverk och evolutionära algoritmer. Men mer vanligt är att programmera tillstånd som agenterna använder. Tillståndet ändras beroende på olika kriterier. Dessa tekniker kan ses som motsatser till varandra och ger agenterna olika egenskaper. De utvecklande teknikerna, dvs. neurala nätverk och evolutionära algoritmer, är ofta avancerade och kan ge oväntade resultat medan de andra teknikerna ger stelare och mindre flexiblare agenter.

Artificiell intelligens kan även användas i simuleringar av individer, något som Craig Reynolds (1986) gjorde när han simulerade flockbeteenden. Reynolds simulerade flock påverkas av varje enskild individs agerande som tillsammans skapar ett beteende som är större än summan av de enskilda individernas beteende.

Beteendet hos Reynolds individer påminner om samverkan mellan bilisterna i trafiken där de enskilda bilisterna måste anpassa både hastighet och körstil efter den situation som råder. Det kan beskrivas som ett distribuerat system där de enskilda individerna kan ha olika agendor men där alla måste samarbete för att inte systemet ska stanna upp. Bilisterna måste sträva mot sina mål samtidigt som de anpassar sig efter andra.

Detta arbete fokuserar på just bilister och hur de kan simuleras i en datorvärld med hjälp av den teknik som Reynolds (1999) la grunden till när han skapade de simulerade flockarna. Arbetet är framför allt inriktat på agenternas rörelsebeteende och de metoder och modeller som beskrivs är främst utvecklade för detta. Uppsatsen tar upp tidigare forskning som ligger till grund för de styrbeteenden som används i arbetet. Grunden till arbetet är framför allt Craig Reynolds artiklar ”*Flocks, herds, and schools: A distributed behavioral model*” (1986) och ”*Steering behaviors for autonomous character*” (1999) men även Robin Greens mer ingående artikel ”*Steering behaviours*” (2000).

2 Bakgrund

2.1 Autonom agent

En agent beskrivs av Russell och Norvig (2002) som en enhet som med hjälp av sensorer läser av omgivningen och agerar utefter omgivningens tillstånd. Agenten baserar sina val på den kunskap som den har, vanligtvis är den kunskapen fördefinierad. Agenter kan ha en fysisk kropp så som en robot eller enbart existera i en virtuell värld som till exempel spel eller simuleringar. Enligt Russel och Norvig (2002) baserar autonoma agenter även sina beslut på tidigare erfarenheter.

En autonom agent kan beskrivas som en agent som beroende på miljön och agentens perception, dvs. agentens uppfattning av miljön utifrån dess sensorer eller sinnen, kan ta beslut om vilka handlingar som den ska ta utifrån tidigare erfarenheter. En autonom agent har inte ett förutbestämt beteende, utan agenten fattar sina egna beslut. Autonoma agenter lämpar sig väl för kontinuerliga miljöer, dvs. miljöer som är obegränsade. Agenten har ofta något mål som den strävar efter att uppfylla. Buckland (2005, s. 85) ger en tydlig definition på autonoma agenter:

> An autonomous agent is a system situated within and a part of an environment that senses that environment and acts on it, over time, in pursuit of its own agenda and so as to effect what it senses in the future.

2.2 Boids

1986 presenterade Craig Reynolds en artikel om simulerade flockbeteenden, i denna artikel beskriver Reynolds en distribuerad modell där individerna interagerar med varandra för att tillsammans skapa flockens beteende. Dessa autonoma agenter kallar han för *”bird-oids”* eller kort och gott *”biods”*. Reynolds (1986) utvecklade tre grundläggande regler som varje individ måste följa för att agenterna tillsammans ska skapa ett realistiskt flockbeteende. Dessa tre regler är:

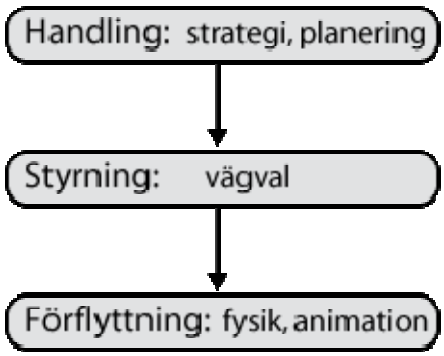
1. Undvik kollisioner med närliggande agenter.
2. Anpassa hastigheten till närliggande agenter.
3. Flyg nära närliggande agenter.

De tre reglerna tar alla hänsyn till närliggande agenterna i flocken. Det är just det som visar sig vara grunden i ett realistiskt flockbeteende. För att skapa realistiskt flockbeteende anser Reynolds (1986) att det krävs att agenterna har en lokal uppfattning av världen. Det vill säga att varje agent uppfattar resten av flocken utifrån dess egen orientering i världen.

Varje regel ger agenten ett förslag om hur den ska styra för att uppfylla den specifika regeln. Dessa förslag representeras av en acceleration som beräknas olika beroende på vilken regel det gäller. Accelerationerna förverkligas av en vektor som förutom riktning även har en längd. Det enklaste sättet att kombinera dessa accelerationer är att använda medelvärdet. Men i en tidig version av Reynolds (1986) implementation visade det sig att den metoden ger problem. Till exempel kan regler ta ut varandra om de ger motriktade krafter. Därför kom han istället fram till en metod som fungerar bättre. Agenten avgör vilken av reglerna som är viktigast, dels beroende på en inre ordning mellan reglerna och dels på storleken på accelerationen.

2.3 Abstrakt beteendemodell

Craig Reynolds (1999) fortsatte sitt arbete med att beskriva och utveckla de regler och beteenden som användes för att skapa boidsagenterna. 1999 presenterade han en artikel om styrbeteenden för autonoma agenter. I artikeln presenterar han en modell för agenternas rörelsebeteende uppdelad i tre olika abstraktionsnivåer, *handling*, *styrning* och *förflyttning*. De olika lagren bygger tillsammans upp en agents rörelsebeteende där handlingar bryts ned i flera enskilda styrbeteenden som till slut resulterar i styrsignaler till förflyttningslagret.



Figur 1: visar Reynolds abstrakta beteendemodell

Reynolds (1999) beskriver modellen med ett exempel om en cowboy som driver en hjord av djur, cowboyen upptäcker att ett djur har vandrat bort från hjorden och bestämmer sig för att driva tillbaka det. Cowboyen styr sin häst mot djuret runt hinder och driver tillbaka den till resten av hjorden. I exemplet upptäcker cowboyen att tillståndet i miljön har förändrats (ett djur har lämnat hjorden) och bestämmer sig för en *handling* (att driva tillbaka djuret). Cowboyen styr hästen runt hinder och tar sig fram till det vilsna djuret, detta motsvara *styrningslagret*. Hästen i exemplet motsvarar *förflyttningslagret* som styrs av ryttaren.

2.3.1 Handling

Handlingslagret motsvarar den nivå där agenterna väljer vad de ska utföra, en övergripande handling. Vilka handlingar som finns är helt beroende av hur agenterna är implementerade och vilken miljö de existerar i. Detta lager hanterar allt som ligger över själva styrningen och väljer vad agenten ska göra, medan de lägre lagren väljer väg och sköter förflyttningen. I exemplet med cowboyen sker en handling som en reaktion på en förändring i miljön men handlingar kan även komma från förändringar inom agenten själv, cowboyen kan välja att stanna för att äta eller liknande.

2.3.2 Styrning

Styrningslagret är det lager som hanterar de enskilda delmålen som varje handling går att bryta ned i. Dessa delmål är enkla och motsvarar endast ett styrbeteende. De är byggstenarna som bygger en handling. Reynolds (1999) artikel beskriver flera olika beteenden och innehåller även en generalisering av de tre beteendena, *separation*, *sammanhållning* och *formera*, som han tidigare använde för att skapa boids. Robin Green beskriver dessa styrbeteenden mer ingående med praktiska tips i sin artikel *”Steering behaviours”* (2000). Detta lager ansvarar även för att kombinera flera styrsignaler till en enda som förflyttningslagret kan använda, detta beskrivs mer ingående av Buckland (2005).

I cowboyexemplet resulterar handlingen i flera enskilda styrbeteenden, t.ex. undvika hinder och styra mot en punkt.

2.3.3 Förflyttning

Förflyttningslagret beskriver de mer mekaniska aspekterna av agenten, det vill säga hur agenten rör sig i miljön från en punkt till en annan. Detta lager kan skilja sig mycket mellan olika sorters agenter även om de övre lagren är oförändrade. I exemplet med cowboyen representeras detta lager av hästen som styrs av sin ryttare, hästen går att byta ut mot något annat transportmedel som till exempel en motorcykel eller helikopter som moderna boskapsdrivare använder, utan att de övre lagren förändras. Detta lager översätter de styrsignaler som kommer från styrningslagret och bryter ner dessa till direkta operationer som måste utföras för att tillfredställa styrsignalen. Det är även detta lager som hanterar animation.

2.4 Styrbeteende

I Reynolds artikel (1999) identifierar han även flera styrbeteenden som används för att förverkliga styrningslagret. Denna teknik bygger på små enkla funktioner som bedömer situationen agenten befinner sig i och ger ett förslag på hur agenten ska styra för att hantera situationen. De beteenden som han presenterar använder en enkel och generell fordonsmodell som motsvarar förflyttningslagret, modellen beskrivs mer ingående av Green (2000). Fordonsmodellen har en position, p, och en hastighet, v. Dessutom har fordonet en massa, m, dess maximala hastighet, $v_{\max}$ och dess maximala styrkraft, $f_{\max}$. Gemensamt för alla beteenden är att de resulterar i en styrkraft, f_{styr}. Denna styrkraft skickas sedan vidare från styrningslagret till förflyttningslagret där fordonet omsätter kraften till en acceleration som påverkar fordonets hastighet och riktning. Med den nya hastigheten uppdateras sedan fordonets position, p. Mellan beräkningarna begränsas kraften och hasigheten med $v_{\max}$ och $f_{\max}$.

$$a = \frac{f_{styr}}{m} \qquad \text{(Kraftekvationen)}$$

$$v = v + a * \Delta t \qquad \text{(Hastighetsförändringen)}$$

$$p = p + v * \Delta t \qquad \text{(Positionsförändringen)}$$

För att beräkna styrkraften, f_{styr}, för ett enskilt beteende beräknas först vilken hastighetsvektor, $v_{önskad}$, som ska föra fordonet till målet. Därefter beräknas skillnaden mellan fordonets hastighet, v, och den önskade hastigheten, $v_{önskad}$, vilket ger en ny vektor som används som styrkraft.

$$f_{styr} = v_{önskad} - v \qquad \text{(Styrkraften)}$$

De viktigaste styrbeteendena som Reynolds identifierade beskrivs nedan, både matematisk och visuellt.

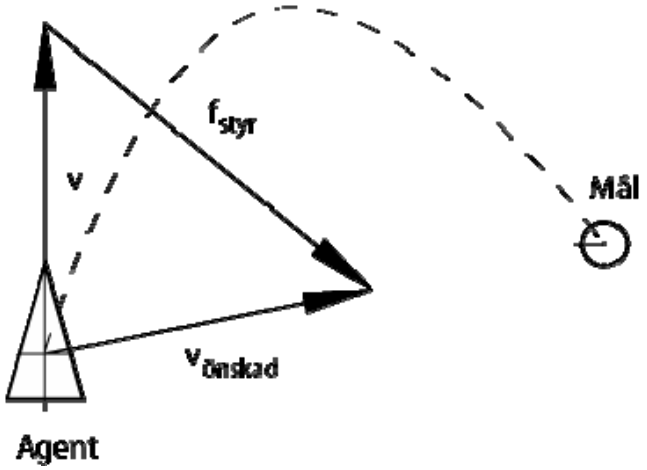
2.4.1 Sök (eng. seek)

Sökbeteendet styr agenten mot en specifik punkt, $p_{mål}$, vilket sker genom att beräkna vilken hastighetsvektor som agenten ska färdas med för att passera genom

målet, $v_{önskad}$, som fås genom att beräkna den vektor som går från agenten mot målet med längden, $v_{\max}$. Styrkraften, f_{styr}, beräknas som skillnaden mellan dessa två vektorer. Om en agent fortsätter söka mot målet kommer agenten att passera det och sedan vända om och på nytt söka mot det. Nedan visas den ekvation som används och en schematisk bild över beteendet, den streckade linjen visar den rörelse som beteendet resulterar i.

$$v_{önskad} = \left(\frac{p_{mål} - p}{|p_{mål} - p|} \right) * v_{\max} \qquad \text{(Önskade hastigheten)}$$

$$f_{styr} = v_{önskad} - v \qquad \text{(Styrkraften)}$$



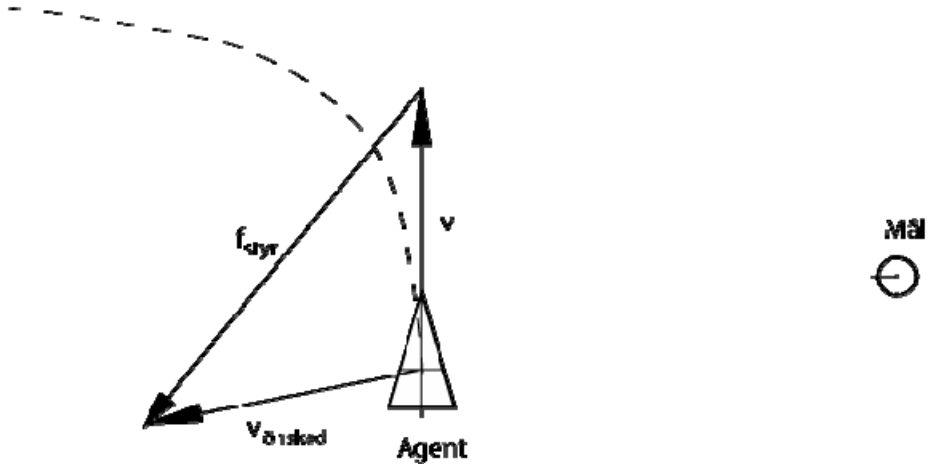
Figur 2 visar sökbeteendet.

2.4.2 Fly (eng. flee)

Flybeteendet fungerar motsatt till sökbeteendet. Agenten söker sig inte mot punkten utan styr för att ha målet bakom sig. Det sker genom att invertera den önskade hastigheten, $v_{önskad}$, på så sätt färdas agenten bort från målet.

$$v_{önskad} = \left(\frac{p - p_{mål}}{|p - p_{mål}|} \right) * v_{\max} \qquad \text{(Önskade hastigheten)}$$

$$f_{styr} = v_{önskad} - v \qquad \text{(Styrkraften)}$$



Figur 3 visar flybeteendet.

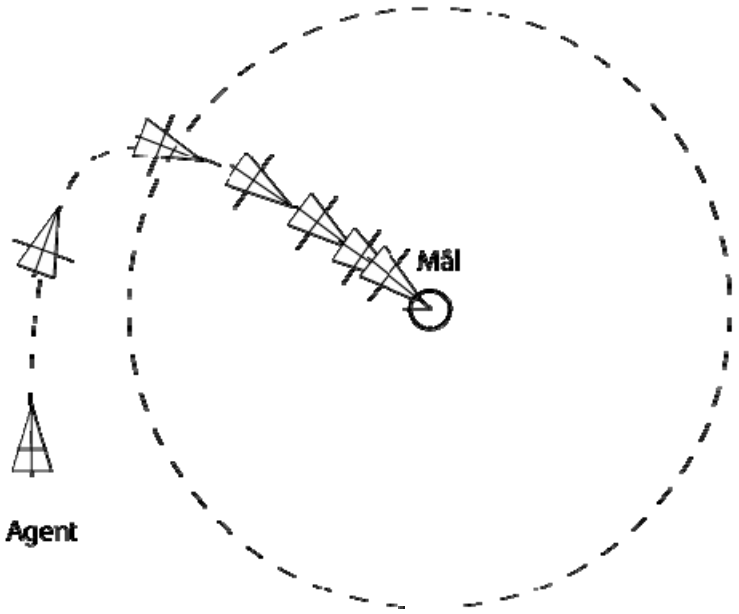
2.4.3 Ankomst (eng. arrival)

Ankomstbeteendet fungerar som sökbeteendet så länge som avståndet till målet är stort. När agenten kommer inom en radie som specificeras av målet skalas hastighetsvektorn linjärt mot avståndet mellan agenten och målet. Det gör att agenten gradvis saktar in för att till slut stanna på målet. Beteendet använder ett inbromsningsvärde, b, för att avgöra hur snabbt fordonet ska bromsa in. Med hjälp inbromsningsvärdet avgörs därefter vilken hastighet, $v_{önskad}$, som agenten ska ha baserat på avståndet till målet.

$$l = \left| p_{mål} - p \right| \qquad \text{(Avståndet mellan agenten och målet)}$$

$$v_{önskad} = \left(\frac{p_{mål} - p}{\left| p_{mål} - p \right|} \right) * \frac{l}{b} \qquad \text{(Önskade hastigheten)}$$

$$f_{styr} = v_{önskad} - v \qquad \text{(Styrkraften)}$$



Figur 4 visar ankomstbeteendet.

2.4.4 Förfölja (eng. pursue)

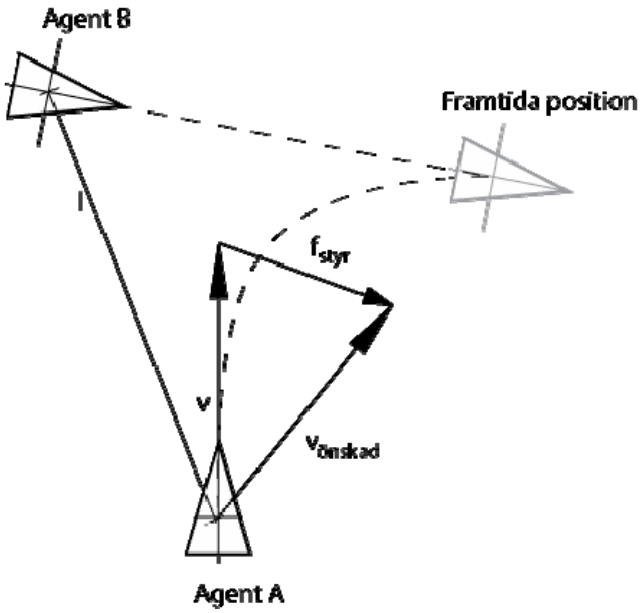
Förföljningsbeteendet påminner om sökbeteendet men söker inte mot en fast punkt utan en punkt som rör sig, till exempel en annan agent. För att effektivt förfölja målet förutser beteendet målets framtida position, $p_{framtid}$, det sker baserat på målets nuvarande hastighet och riktning, $v_{mål}$. Eftersom beteendet utvärderas vid varje uppdatering fungerar det även om målet kraftigt ändrar riktning. När beteendet har beräknat $p_{framtid}$ används sökbeteendet för att flytta agenten dit. En viktig del av detta beteende är hur agenten uppskattar målets framtida position. Det finns flera olika sätt med varierande komplexitet men grunden i alla metoder går ut på att försöka förutsäga hur lång tid det tar för agenten att nå målet baserat på avståndet mellan agenten och målet samt hur mycket agenten måste svänga för att styra mot målet. I Opensteer (Reynolds, 2004) används följande metod: först beräknas avståndet till målet, l, utifrån det beräknas hur lång tid, t, det tar för agenten att färdas den sträckan. Därefter uppskattas målets nya position, $p_{framtid}$,

med hjälp av målets hastighet, $v_{mål}$, och den uppskattade tiden, t. Mot den nya punkten styr sedan agenten med hjälp av sökbeteendet, se avsnitt 2.4.1.

$$l = \left| p_{mål} - p \right| \qquad \text{(Avståndet till målet)}$$

$$t = \frac{l}{v_{\max}} \qquad \text{(Tiden som krävs för att färdas avståndet)}$$

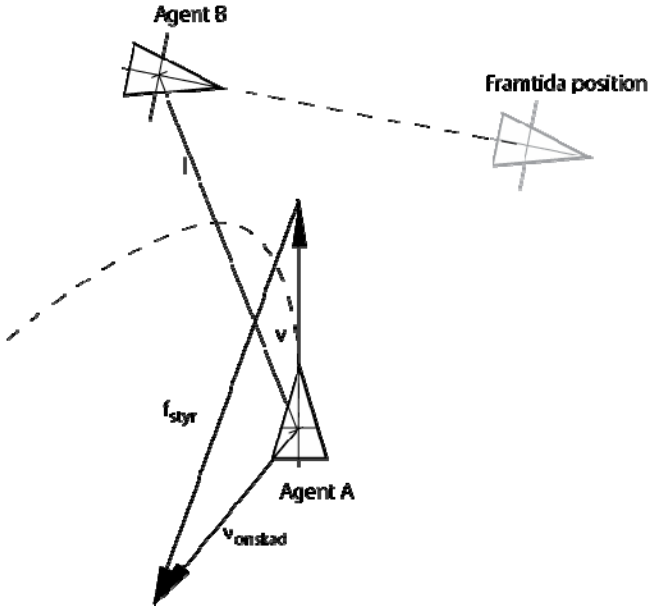
$$p_{framtid} = p_{mål} + t * v_{mål} \qquad \text{(Målets framtida position)}$$



Figur 5 visar förföljningsbeteendet**.**

2.4.5 Undvika (eng. evade)

Undvikningsbeteendet fungerar på samma sätt som förföljningsbeteendet, 2.4.4, men använder flybeteendet istället för sökbeteendet för att styra bort från agenten. Beteendet använder samma ekvationer som i 2.4.4.



Figur 6 visar undvikningsbeteendet.

2.4.6 Förskjuten förföljning (eng. offset pursuit)

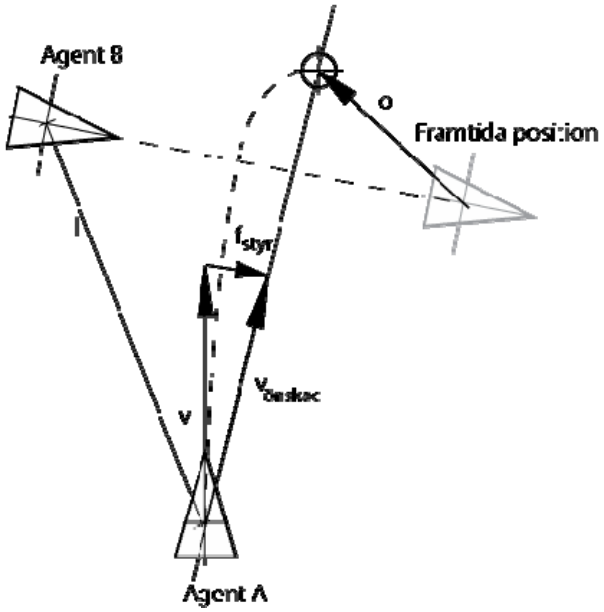
Beteendet är en utveckling av förföljningsbeteendet som efter att ha beräknat målets framtida position förskjuter den punkten med en vektor, o. Detta beteende kan användas för att skapa formationer av flera agenter. Beteendet beräknas på samma sätt som i 2.4.4 men den framtida positionen förskjuts innan agenten styr mot den.

$l = |p_{mål} - p|$ (Avståndet till målet)

$t = \frac{l}{v_{\max}}$ (Tiden som krävs för att färdas avståndet)

$p_{framtid} = p_{mål} + t * v_{mål}$ (Målets framtida position)

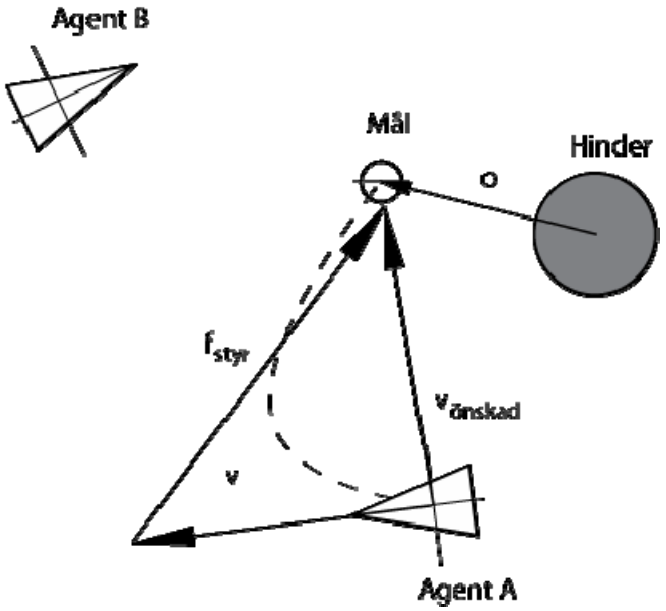
$p_{framtid} + o$ (Förskjuter målet)



Figur 7 visar förföljningsbeteendet med förskjutning.

2.4.7 Inskjuta (eng. interpose)

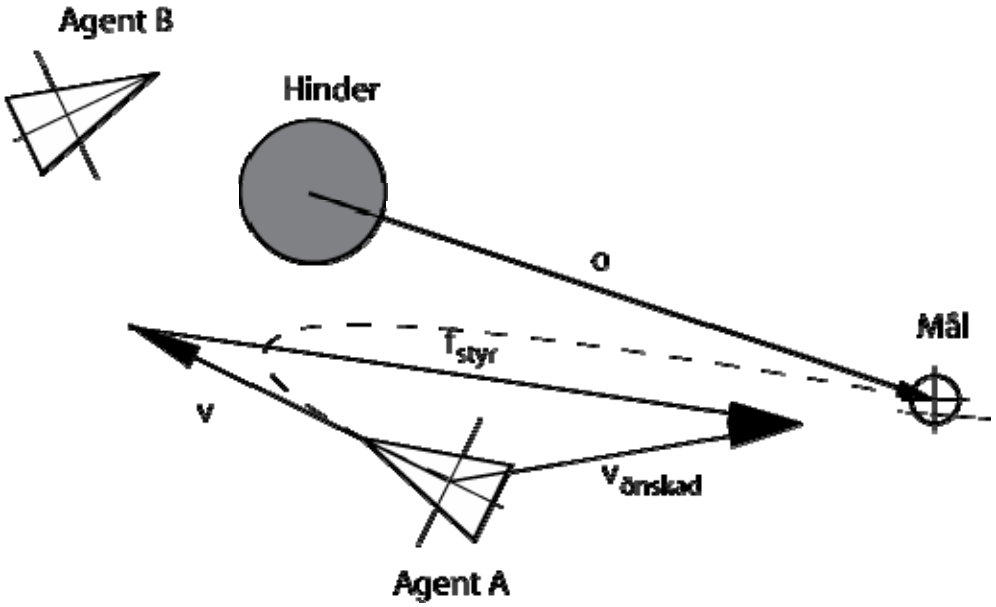
Idén till inskjutningsbeteendet föreslogs av Reynolds (1999) men beskrivs mer ingående av Green (2000). Beteendet kan antingen tillämpas på fasta punkter eller på andra agenter som rör sig. I det första fallet beräknas en målpunkt mellan de fasta punkterna och sedan används sökbeteendet för att nå målpunkten. När vektorn mellan punkterna är beräknat skalas den för att hitta den exakta punkten som beteendet använder. I det andra fallet måste agenternas framtida position uppskattas på samma sätt som i förföljningsbeteendet för att kompensera för att de rör sig.



Figur 8 visar inskjutningsbeteendet.

2.4.8 Gömma (eng. hide)

Gömningsbeteendet påminner om inskjutningsbeteendet, beteendet utnyttjar ett mål, fixerat eller rörligt, och ett hinder och strävar efter att placera det hindret mellan agenten och målet. Det sker genom att beräkna den vektor som går genom både agenten och hindret och förlänga den genom hindret agenten vill gömma sig bakom. Punkten som beräkningen tar fram ligger då bakom hindret och agenten kan söka sig dit. Hur långt bakom hindret agentens mål ligger avgörs av hindrets storlek.

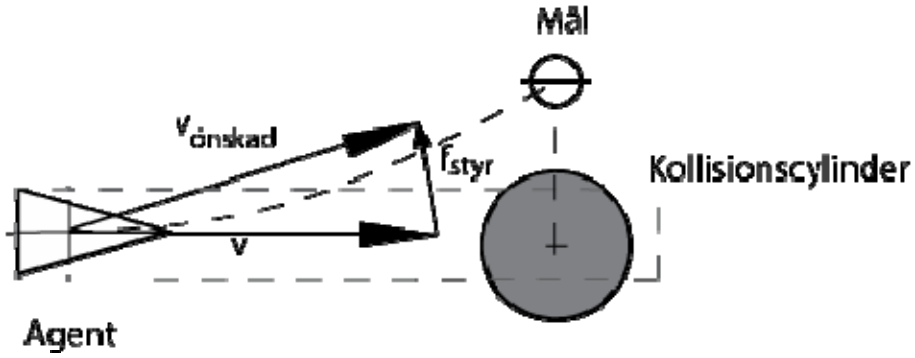


Figur 9 visar gömningsbeteendet.

2.4.9 Undvika hinder (eng. obstacle avoidance)

Detta beteende ger agenten egenskapen att styra undan från hinder som finns i dess närhet. Reynolds (1999) beskrivning av beteendet är en grov förenkling som

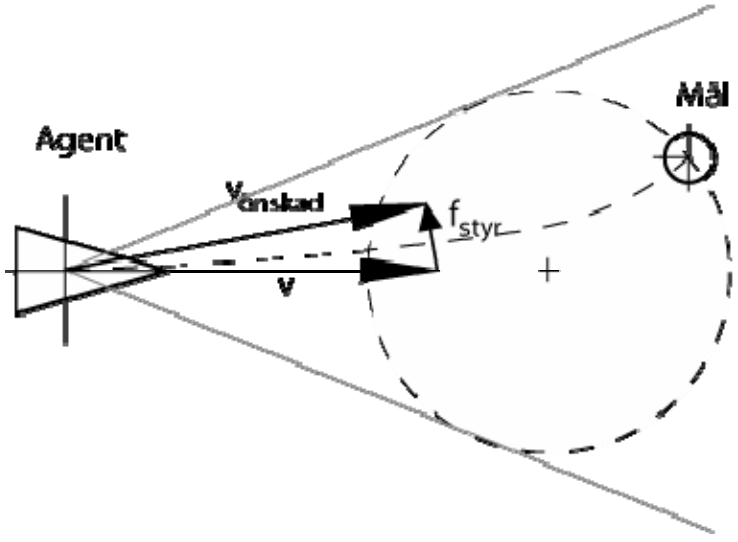
bygger på att varje hinder representeras av en sfär. Beteendet skapar en cylinder runt agenten som är riktad längs agentens färdriktning. Cylinderns längd bestäms av agentens hastighet och cylinderns radie av agentens bredd. Denna cylinder används för att finna hinder som agenten kan kollidera med, genom att hitta de sfärer som skär cylindern. Eftersom beteendet använder en cylinder som ligger i agentens färdriktning bedöms endast hinder som ligger rakt framför agenten, hinder på sidan av agenten påverkar inte dess styrning.



Figur 10 visar hinder undvikande.

2.4.10 Ströva (eng. wander)

Ströva är ett beteende som låter agenten röra sig slumpmässigt. För att undvika ryckiga rörelser som kan bli resultatet av att låta agenten använda slumpmässiga målpunkter föreslår Reynolds (1999) en metod som ger en mjukare och jämnare rörelse. Metoden tar ut en slumpmässig punkt på en sfärs yta som ligger framför agenten. Sfärens radie och avståndet mellan agenten och sfären bestämmer hur stora hastighetsförändringar som är möjliga. När en slumpmässig punkt är funnen används sökbeteendet för att styra agenten mot den punkten. Metoden tar bara ut slumpmässiga punkter som ligger framför agenten och på så sätt undviks ryckigt beteende.



Figur 11 visar strövningsbeteendet.

2.4.11 Vägföljning (eng. path following)

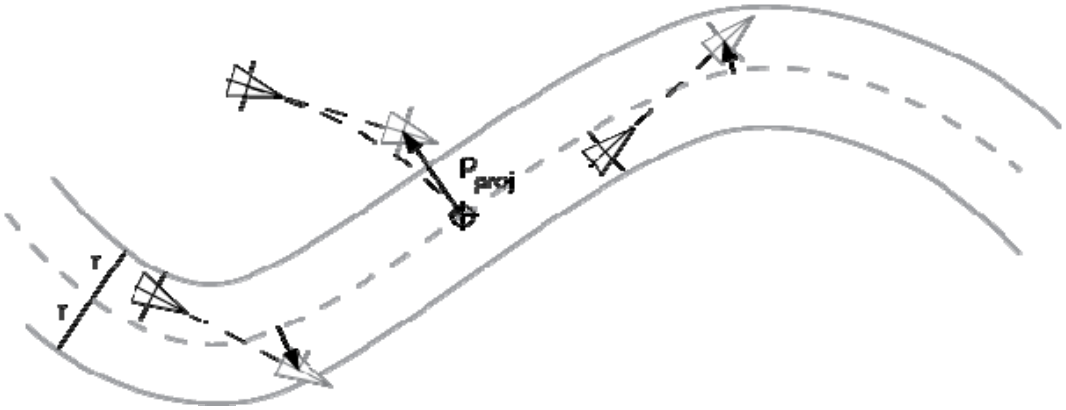
Vägföljningsbeteendet låter agenten följa en utstakad väg. Beteendet låser inte agenten till vägen, som om det gick på en räls, utan fungerar istället mer som en korridor för agenten att följa. Varje väg utgörs av en kurva som omsluts av en

cylinder med en radie, r, på så sätt blir vägarna som tunnlar, det skapas en tredimensionell volym runt kurvan. Beteendet förutser var agenterna kommer att hamna baserat på deras hastighet och orientering. Styrkraften beräknas sedan genom att projicera agentens framtida position på kurvan som utgör vägen, dvs. hitta den punkt på kurvan som är närmast. Avståndet mellan dessa punkter jämförs sedan med radien på cylindern som omsluter vägen och om den förutsedda punkten ligger inuti cylindern krävs ingen justering. Annars används sökbeteendet för att söka mot den projicerade punkten, p_{proj}.

$$l = \left| p_{väg} - p \right| \qquad \text{(Avståndet till vägen)}$$

$$t = \frac{l}{v_{\max}} \qquad \text{(Tiden som krävs för att färdas avståndet)}$$

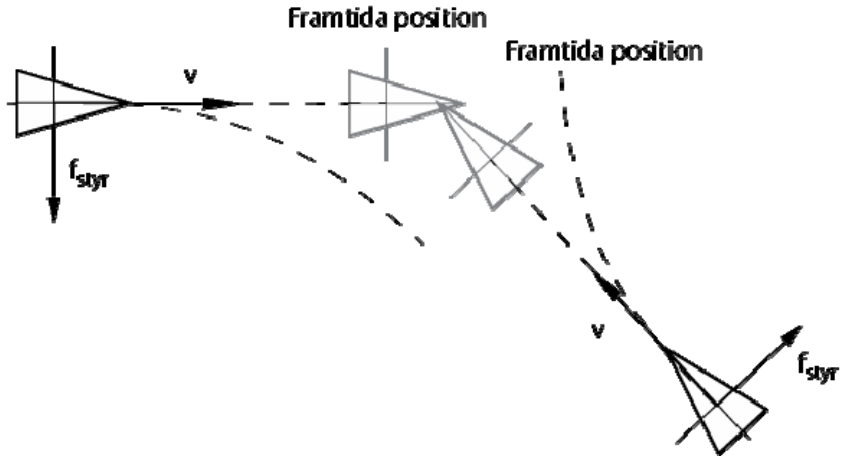
$$p_{framtid} = p_{mål} + t * v_{mål} \qquad \text{(Målets framtida position)}$$



Figur 12 visar vägföljningsbeteendet.

2.4.12 Undvika kollsion (eng. unaligned collision avoidance)

Detta beteende försöker undvika kollisioner mellan agenter som rör sig mot varandra. Beteendet undersöker alla agenter i närheten för att finna möjliga kollisioner i framtiden. Om en möjlig kollision upptäcks korrigerar beteendet agentens styrning genom att styra från det område där kollisionen kan uppstå. De framtida positionerna hos agenterna uppskattas som tidigare, om en kollision kan uppstå beräknas åt vilket håll som agenten ska svänga minst åt för att undvika kollision sedan korrigeras styrningen ditåt.



Figur 13 visar undvika kollsioner.

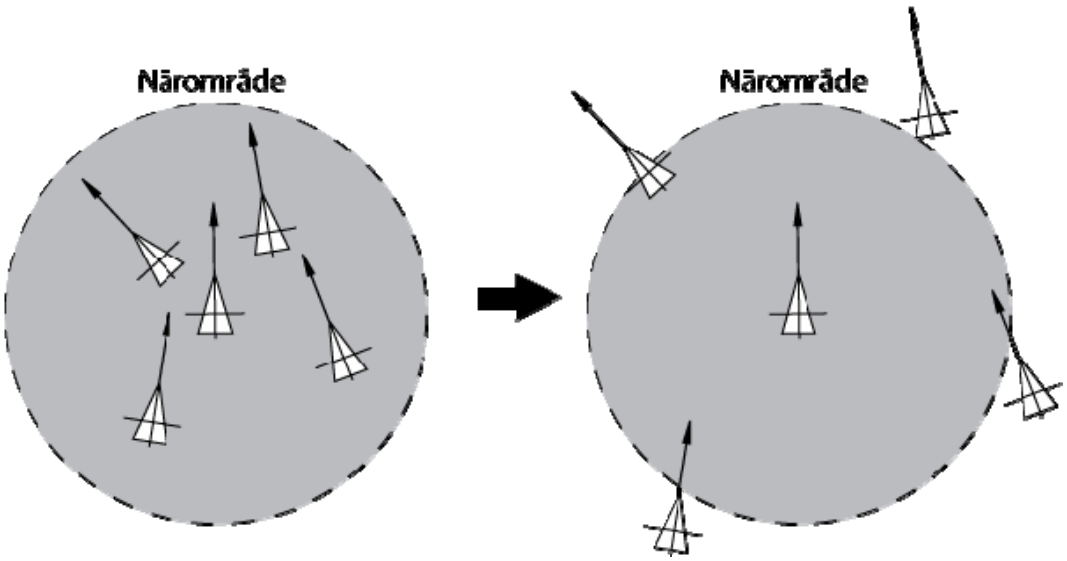
2.4.13 Separation (eng. separation)

Separation är generaliseringen av det första beteendet för boids (2.2). Beteendet verkar för att undvika kollisioner med andra agenter och styr agenten mot det område med lägst täthet av agenter. Beteendet bedömer alla andra agenter i närområdet och för varje agent skapas en vektor, l_i, som är riktad mot agenten. Sedan skapas en styrkraft, f_i, som är den normaliserade vektorn, l_i, som skalas med längden av l_i. Dessa krafter summeras och används som strykraft.

$$l_i = p - p_i \qquad \text{(Avståndet mellan agenterna)}$$

$$f_i = \frac{l_i}{2|l_i|} \qquad \text{(Styrkraften från en enskild agent)}$$

$$f_{styr} = \sum f_i \qquad \text{(Summan av alla styrkrafter)}$$

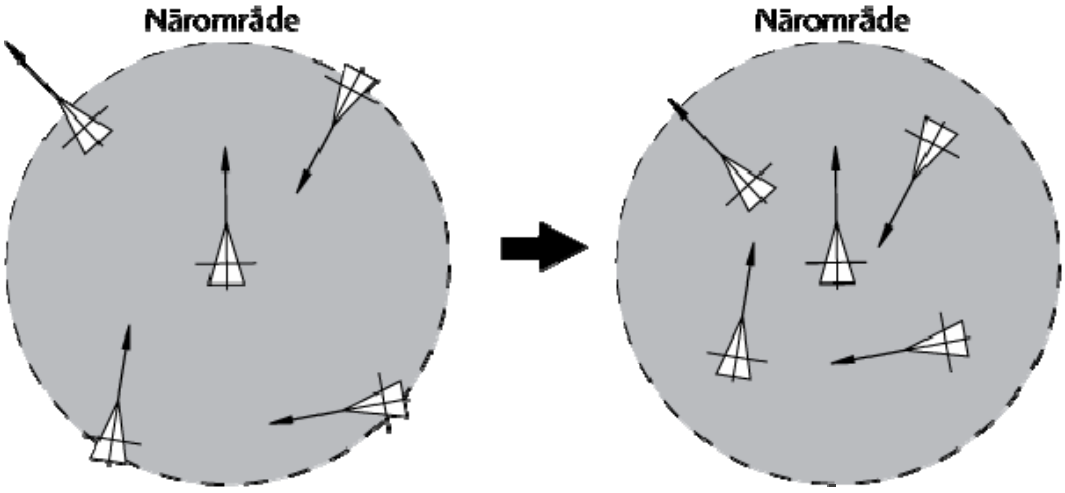


Figur 14 visar separationsbeteendet, före och efter.

2.4.14 Sammanhållning (eng. cohesion)

Sammanhållning är också en generalisering av boidsbeteendena (2.2). Sammanhållningen fungerar motsatt mot separation, beteendet räknar ut medelvärdet för alla agenters position i närheten och använder därefter sökbeteendet för att söka mot den punkten. Alla agenter i närområdets positioner summeras och resultatet delas med antalet agenter.

$$p_{mål} = \frac{\sum p_i}{i} \qquad \text{(Medelvärdet av alla agenters position)}$$

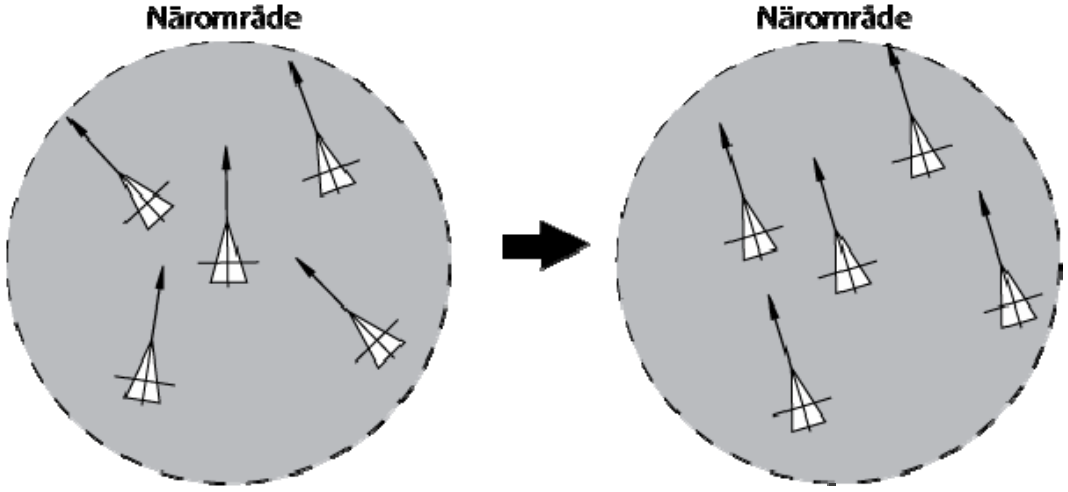


Figur 15 visar sammanhållningsbeteendet, före och efter.

2.4.15 Formera (eng. alignment)

Formeringsbeteendet är även det en ett boidsbeteende. Beteendet strävar efter att korrigera agentens orientering till andra agenter i närområdet. Det sker genom att använda medelvärdet av deras hastighetsvektorer som styrkraft. På så sätt styr agenten åt samma håll som de andra agenterna i dess närhet.

$$f_{styr} = \frac{\sum v_i}{i}$$ (Medelvärdet av alla agenters hastighet)



Figur 16 visar formationsbeteendet, före och efter.

2.5 Kombinera beteenden

Reynolds artikel (1999) fokuserar främst på styrningslagret, där han beskriver flera enskilda styrbeteenden som kan användas för att skapa ett mer avancerat sammansatt beteende hos agenterna. Reynolds identifierar även de problem som kan uppstå då flera beteenden ska kombineras och presenterar flera olika modeller för detta.

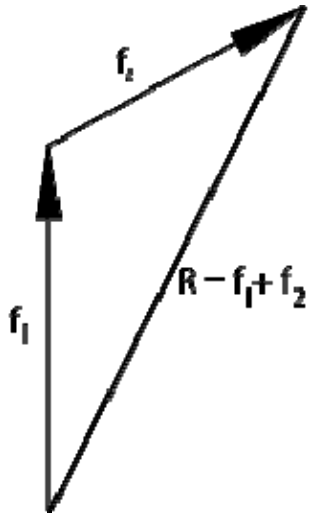
Ena modellen bygger på en prioriteringsordning mellan styrbeteendena där varje beteende bedöms i tur och ordning, det första beteende som vill ändra styrningen får påverka agentens styrning. En annan modell som Reynolds kallar *”prioritized dithering”* bygger på en liknande metod men ordningen är slumpmässig.

2.5.1 Viktad trunkerad summa

Denna modell för att kombinera beteenden är den enklaste som beskrivs av både Reynolds (1999) och Green (2000). Modellen beräknar först samtliga styrkrafter, f_i, för varje beteende, de olika beteendena har olika vikter, w_i, som multipliceras med styrkraften. Vikterna används för att värdera styrbeteendena inbördes, högre vikt ger större påverkan på resultatet. Därefter summeras alla enskilda styrkrafter till en total styrkraft, f_{tot}.

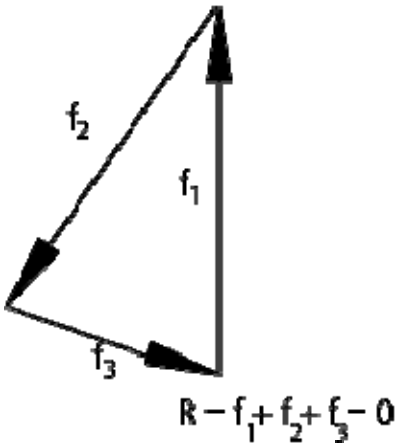
$$f_{tot} = \sum f_i * w_i \qquad \text{(Samtliga styrkrafter summeras)}$$

Den slutgiltiga styrkraften beskärs för att inte vara större än fordonets maximala styrkraft, $f_{\max}$. Med denna modell påverkar alla styrbeteenden resultatet något, vilket kan vara eftertraktat i vissa situationer men om agenten använder många beteenden blir varje beteendes påverkan något försvagat. Men modellen kan ge flera problem, något som belyses av Green (2000), framför allt måste man betrakta vad som händer när flera styrbeteenden summeras. Bilden nedan visar två vektorer, f_1, f_2, och resultanten, R. Detta kanske inte nödvändigtvis är ett problem om agenten kan röra sig fritt längs med resultanten. Detta kan ge problem om agenten är mer begränsad i hur den kan röra sig, som till exempel i biltrafik där agenterna inte får avvika från filen.



Figur 17 visar resultanten av två vektorer.

Ett annat problem är att flera motriktade krafter kan ta ut varandra. I bilden nedan visas ett extremfall då resultanten av alla krafter blir 0 och agenten fortsätter i den riktning den har utan att ta hänsyn till något beteende.



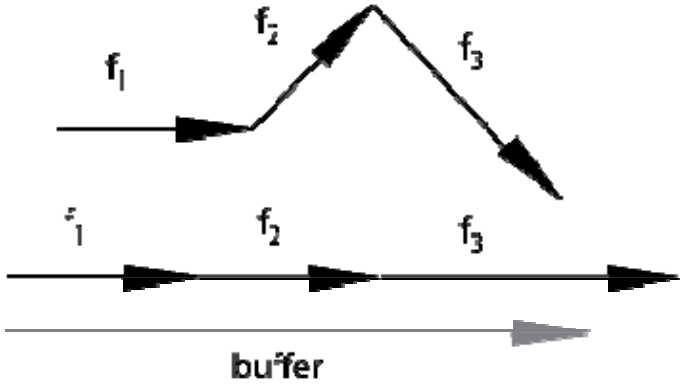
Figur 18 visar summeringen av vektorer som resulterar i nollvektorn.

I detta fall försvinner flera styrningar som kan vara kritiska för agenten, för att väga upp får de styrbeteendena ges en högre vikt. Men detta påverkar alla beteenden då dessa blir relativt mindre viktiga, vilket då kan kräva en justering av flera vikter.

Ett annat bekymmer med denna modell är att samtliga beteenden bedöms, något som i värsta fall kan vara väldigt resurskrävande. Till sist är det problem med vikterna. För att få agenter som fungerar väl krävs ett stort antal tester för att anpassa vikterna så agenten har en bra balans mellan beteendena.

2.5.2 Viktad trunkerad summa med prioritering

Denna beräkningsmodell bedömer beteendena i en given ordning. Innan en styrkraft adderas till resultanten kontrolleras om resultanten inte blir för stor om styrkraften adderas. Det gör att styrkraften aldrig behöver trunkeras men även att styrkraften sällan når sin fulla längd. Resultanten betraktas som en buffert som fylls av styrkrafter, när bufferten är fylld används de styrkrafter som dittills har adderats. I jämförelse med modellen *viktad trunkerad summa* är denna modell i bästa fall något mindre beräkningsintensiv, när bufferten är fylld bedöms inga flera styrbeteenden. Om flera komplexa styrbeteenden används kan det vara beaktningsvärt, annars är styrbeteendena överlag väldigt enkla.



2.5.3 Prioriterad slump

Denna beräkningsmodell föreslogs av Reynolds (1999) under namnet *prioritized dithering*, modellen använder sannolikhet för att avgöra om ett beteende ska påverka styrkraften. Styrbeteendena är ordnade i en prioriteringsordning och varje beteende har ett värde som bestämmer sannolikheten för att beteendet ska

utvärderas. Modellen bedömer beteendena i tur och ordning och avgör med hjälp av slump om beteendet ska utvärderas. När ett styrbeteende ger en styrkraft används denna, annars fortsätter modellen med nästa beteende. Denna modell använder alltid bara styrkraften från ett enda styrbeteende.

3 Problem

Målet med arbetet är att utveckla styrbeteenden för agenter i trafikscenarion samt att utvärdera dem med avseende på användbarhet och flexibilitet. I tidigare arbeten har styrbeteenden främst använts inom områden där agenternas rörelser varit relativt fria, i detta arbete undersöks hur väl styrbeteendena går att använda under mer strikta förhållanden. Ett område som är starkt styrt av regler och där agenternas rörelse begränsas av dessa regler är biltrafik. Biltrafik har även nämnts av Reynolds (1999) som ett område där tekniken skulle fungera. Syftet med arbetet är att öka förståelsen för styrbeteenden samt utforska flera områden där de kan användas.

Arbete är indelat i två delar, *utveckling* och *utvärdering*. Under utvecklingsfasen utvecklas den arkitektur som sen används under utvärderingen för att bedöma teknikens användbarhet och flexibilitet.

3.1 Delmål 1: utveckling

Denna fas går ut på att utveckla den applikation som ska användas för att utvärdera hur användbara styrbeteenden är med avseende på flexibilitet. I applikationen ska agenter använda styrbeteenden för att navigera i ett scenario. Agenterna ska använda och kombinera flera styrbeteenden för att hantera de situationer som kan uppstå i scenariot.

Fasen består av två underliggande delar, *analys* och *implementation*. Dels ska problemet analyseras för att finna fungerande lösningar och arkitekturer, dels ska de enskilda delarna implementeras.

3.2 Delmål 2: utvärdering

I denna fas utvärderas styrbeteendetekniken med hjälp av den applikation som utvecklats. Utvärderingen sker främst med avseende på hur användbar tekniken är samt hur flexibla de lösningar som använder den är. Under utvärderingen ska helheten undersökas, det är mer intressant med det totala resultatet än de enskilda delarna. Därför kommer utvärderingen ske genom att analysera agenternas sammansatta beteende, dvs. resultatet av flera enskilda styrbeteenden.

4 Metod

Detta kapitel beskriver hur arbetet ska ske för att lösa de problem som presenterats i föregående kapitel.

4.1 Metod för delmål 1: utveckling

Delmålet går ut på att utveckla de scenarion, agenter och beteenden som ligger till grund för den undersökning som detta arbete går ut på. Till en början genomförs en mindre litteraturanalys för att finna bra scenarion, det vill säga scenarion som innehåller ett fåtal beteenden som måste kombineras för att agenterna skall kunna agera i situationen. Litteraturanalys är en bra metod att samla kunskap och öka förståelse för ett område och de problem som existerar i domänen. Därefter väljs de enklaste styrbeteendena och de beteenden som inte redan finns implementerade i programbiblioteket OpenSteer (2004) implementeras. Även en arkitektur utvecklas som kan kombinera flera beteenden till en enda styrkraft som styr agenten, som även den utvecklas. Utvecklingen av de enskilda delarna sker parallellt.

Alla scenarion ska använda samma styrbeteenden för att mängden beteenden ska begränsas. Det är mer intressant att under arbetet undersöka hur pass flexibel och användbar tekniken är i olika scenarion än att implementera flera olika styrbeteenden, eftersom de oftast bygger på samma princip. Generellt beräknas en punkt vars placering är mer eller mindre optimal för den situation som beteendet är gjort för, sedan används sökbeteendet för att styra mot den punkten.

4.2 Metod för delmål 2: utvärdering

Tekniken utvärderas via experiment där varje skapat scenario körs och agenternas beteende analyseras. Varje experiment testar hela scenariot och på så sätt testas samverkan mellan beteendena och hur väl kombineringen fungerar. För varje scenario som utvecklas måste också ett eller flera experiment beskrivas som undersöker och utvärderar det aktuella scenariot.

Att utvärdera hela scenarion är mer intressant än enskilda styrbeteenden då agenternas beteenden är resultatet av flera enskilda styrbeteenden. Självklart måste varje beteende valideras så att det fungerar men det är mer intressant med helheten då det är samspelet mellan beteendena som skapar agenternas beteende. Dessutom är kombineringen av beteenden en ytterst viktig process som ger agenterna sitt flexibla beteende och därför är det viktigt att det ingår i utvärderingen.

5 Resultat

Detta kapitel beskriver arbetet med att tillämpa metoderna från kapitel 4, dels beskrivs hur arbetet fortskred samt dels de delar av arbetet som är av särskilt intresse.

5.1 Resultat för delmål 1: utveckling

Arbetet har utgått från de mest grundläggande regler och därför valdes två scenarion som testade dessa enkla regler. Det kan vara förvirrande att kalla dem regler eftersom de är så självklara i trafiken så de knappt kan kallas för regler men de är ändå ytterst viktiga för att det underliggande systemet skall fungera. Valet att arbeta med dessa var kritiskt då det är omöjligt att arbeta med mer komplexa regler utan att först ha skapat de underliggande funktionerna. Arbetet har fokuserat på följande krav eller regler för agenterna:

1. Agenterna ska följa vägen.
2. Agenterna ska anpassa hastigheten till andra agenter.
3. Agenterna ska undvika kollisioner med agenter.

Ingen av dessa är som sagt tydliga regler så som ”högerregeln” utan de är mer förhållningsregler för agenterna som ligger till grund för andra mer komplexa regler. Varenda en av dessa regler går att bryta ner till ett eller flera styrbeteenden som strävar efter att agenten ska följa dessa regler.

Anledningen till att flera scenarion undersöks är för att se hur väl agenternas beteenden går att tillämpa och utöka för att klara mer avancerade scenarion. På så sätt undersöks teknikens flexibilitet och utbyggbarhet. Det intressant att undersöka hur väl agenterna klarar av nya situationer.

Utveckling kan delas in i fyra delar som är mer eller mindre beroende av varandra: *agenterna*, *styrbeteendena*, *kombineraren* och *testscenarierna*. Den testapplikation som skapades använder programbiblioteket OpenSteer (2004) som är skapat av Reynolds med flera. Till biblioteket finns en demonstrationsapplikation som gör det möjligt att enkelt skapa och visualisera scenarion för agenter. I biblioteket finns det exempelkod och även färdiga agenter och styrbeteenden. All utveckling har genomförts med hjälp av OpenSteer.

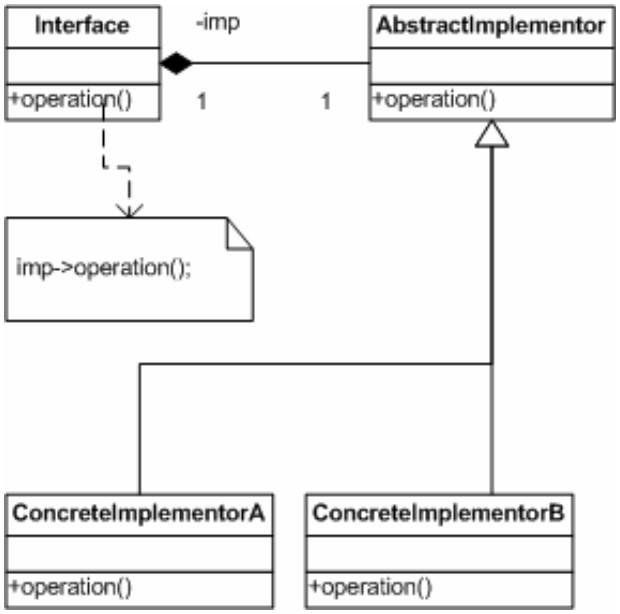
5.1.1 Agenter

Agenterna som utvecklades använder en enkel fysikmodell som motsvarar förflyttningslagret, se 2.3.3, i Reynolds abstraktionsmodell (1999). Agenterna bygger vidare på den grundläggande agentklass som finns implementerad i OpenSteer (2004). Agenterna har en enkel fordonsmodell som stämmer överrens med den modell som presenterats i 2.3.3. Agentklassen har utökats med en *kombinerare* som kombinerar styrkrafterna från agentens styrbeteenden till en slutgiltig styrkraft. Därutöver skiljer sig inte agenterna mycket från den grundläggande agentklass som finns i OpenSteer (2004).

5.1.2 Kombinerare

Eftersom valet av kombineringsmodell kan påverka resultatet var det intressant att undersöka flera av de modeller som Reynolds (1999) och Green (2001) föreslog. Därför utvecklades en arkitektur som möjliggör användandet av olika

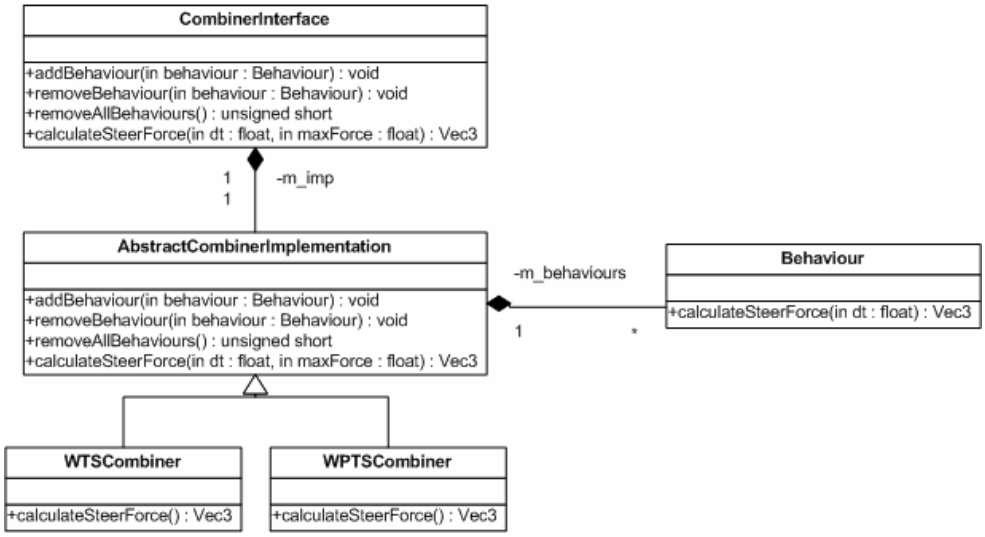
beräkningsmodeller, på så sätt kan de olika modellerna undersökas. För att uppnå detta krävs en flexibel arkitektur som dels klarar av att hantera olika styrbeteenden men även att ändra beräkningsmodell. Designmönstret *brygga* (eng. bridge) som beskrivs i *Design patterns: Elements of reusable object-oriented software* (Gamma et al., 1995) löser de designkrav som ställs på kombineraren. Designmönstret bygger på ett generellt gränssnitt som använder en separat abstrakt implementation.



Figur 19 visar designmönstret *bridge*.

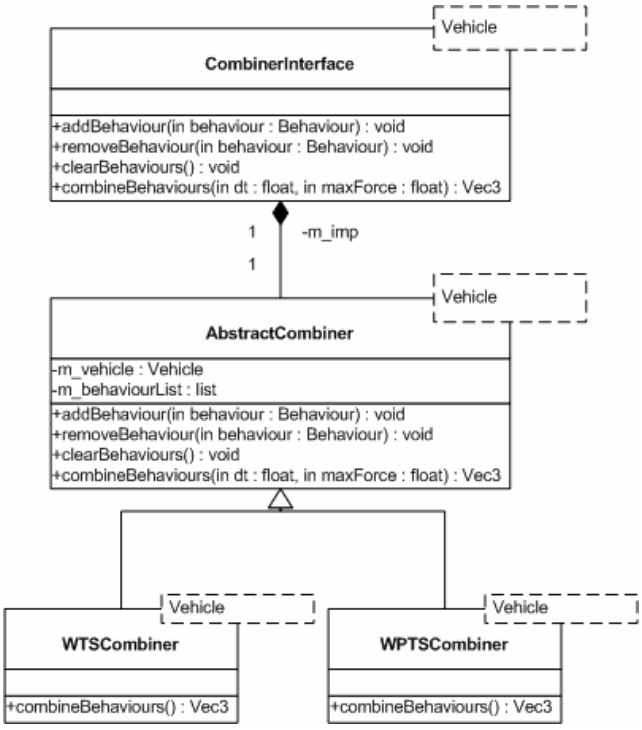
Figur 19 visar ett UML-diagram över designmönstret, där gränssnittet, *Interface,* använder den abstrakta implementation, A*bstractImplementor,* för att utföra operationer. Implementationsklassens beteenden bestäms av vilken konkret subklass som används, t.ex. *concreteImplementorA*, på så sätt kan beteendet varieras. Gränssnittet känner inte till hur en operation utförs utan vidarebefordrar bara anropet vidare till sin implementationsklass.

Detta designmönster tillämpades för att skapa en kombineringsarkitektur som gör det möjligt att variera beräkningsmodell utan att ändra gränssnittet. I Figur 20 representeras gränssnittet av klassen *CombinerInterface.* Gränssnittet utnyttjar i sin tur den abstrakta klassen *AbstractCombinerImplementation* för att utföra kombineringen. Eftersom *AbstractCombinerImplementation* är en abstraktklass saknar den i sig någon funktion utan funktionen finns specificerad i någon av subklasserna, *WTSCombiner* eller *WPTSCombiner*. Genom att ändra vilken subklass som används ändras beräkningsmodell.



Figur 20 visar den ursprungliga designen för kombineraren.

Den slutgiltiga arkitekturen som används är något förändrad för att anpassas för OpenSteer (2004). Grunden är densamma men klassen *Behaviour* är utbytt mot funktionspekare till de medlemsfunktioner som implementerar styrbeteendena. Dessutom har varje beteende fått ett viktningsvärde som används för att vikta varje enskilt beteende. I beräkningsmodellen *viktad trunkerad summa med prioritering* används den ordning som styrbeteendena läggs till i för att avgöra prioriteringsordningen. Det beteende som läggs till först kommer att utvärderas först. Valet att använda funktionspekare gjordes därför att Opensteer implementerar styrbeteenden som medlemsfunktioner. Funktionspekare är inte vanligt förekommande i C++ för de ger en otymplig och svår syntax som inte är lätt att använda på rätt sätt. Det är ett stort område och till viss del okänd mark för många programmerare. Arbetet fokuserar inte på hur dessa används och det kommer därför inte beskrivas i detta arbete. Figur 21 visar ett UML-diagram över den anpassade design som använder funktionspekare.



Figur 21 visar ett UML-diagram över den slutgiltiga kombineringsarkitekturen.

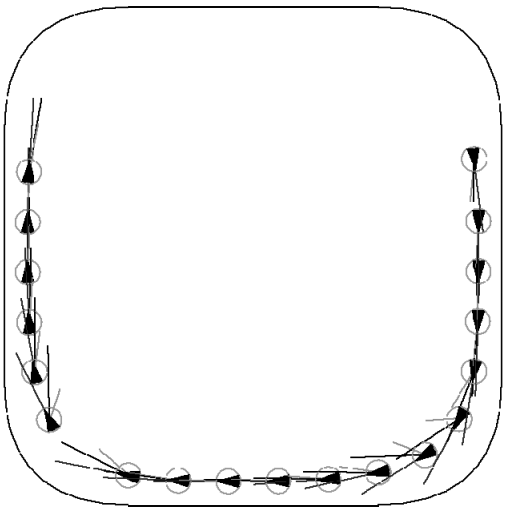
5.1.3 Styrbeteenden

Följande styrbeteenden skapades för att agenterna skulle klara att navigera i scenariona. De bygger alla på existerande beteenden som har förändrats för att passa de situationer som valts. De beteenden som utvecklats är *följa vägen i rätt fil, anpassa hastigheten* och *undvika kollisioner*.

Det första beteendet är en utveckling av vägföljningsbeteendet, 2.4.11. Förutom att beteendet har anpassats för att placera agenterna på rätt sida om vägen så anpassar det också agenternas hastighet för att de ska hantera kurvorna bättre utan att hamna för långt utanför sin fil. Precis som i vägföljningsbeteendet beräknas agentens framtida position, den används sedan för att finna en punkt som ligger på vägen. Sedan används sökbeteendet för att nå den punkten. I vanliga fall ligger den punkten mitt på vägen men detta beteende förskjuter den i sidled förhållande till agenten. Förskjutningen sker beroende på om det är högertrafik eller vänstertrafik. Hastigheten beräknas med hjälp av skillnaden mellan agentens riktning och riktningen mot den beräknade punkten på vägen. Om dessa skiljer mycket får agenten en lägre hastighet, om agenten däremot färdas rakt mot punkten anpassas inte hastigheten.

Stycket nedan visar C++ funktionen som beräknar styrkraften, funktionen kan delas in i två samverkande delar, i den första delen beräknas den punkt, *onPath*, på vägen som ligger närmast agentens kommmande position. Mot den punkten används sökbeteendet för att få en styrkraft. Den andra delen av funktionen går ut på att bromsa agenten genom att applicera en styrkraft motriktad mot agentens rörelse. Genom att beräkna skillnaden mellan agentens färdriktning och vektorn som är riktad mot målpunkten fås ett mått på hur mycket agenten måste svänga. Bromskraften skalas linjärt med den skillnaden. Figur 22 visar resultatet av beteendet och den väg som en agent färdas längs med en cirkulär väg.

```
Vec3 steerToFollowRoad(const float predictionTime, Pathway& path, Vec3 offset)
{
        Vec3 futurePosition = predictFuturePosition(predictionTime);

        Vec3 tangent;
        float distance;
        Vec3 onPath = path.mapPointToPath(futurePosition, tangent, distance);
        Vec3 seekForce = steerForSeek(onPath + offset);

        Vec3 localOffset = onPath + offset - position();
        float forwardComponent = localOffset.dot(forward());
        Vec3 forwardOffset = forwardComponent * forward();
        Vec3 offForwardOffset = localOffset - forwardOffset;

        Vec3 breakForce = -forward() * offForwardOffset.length();
        seekForce += breakForce;

        return seekForce;
}
```



Figur 22 visar en agents färd med hjälp av vägföljningsbeteendet.

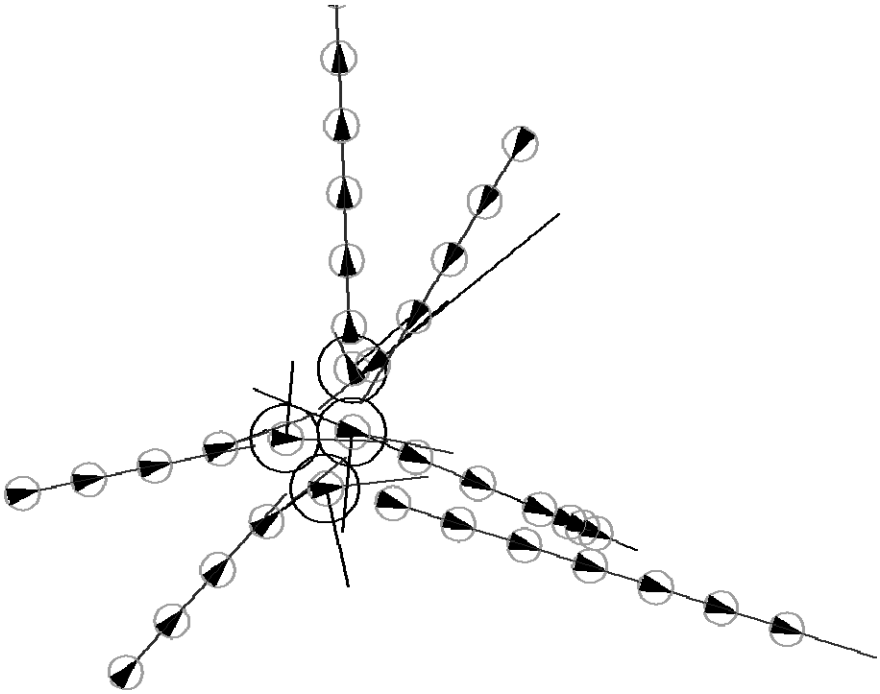
Det andra beteendet används för att anpassa hastigheten genom att söka efter andra agenter i närheten, om någon agent finns framför agenten minskar den hastigheten genom att skapa en bromskraft. Beteendet sorterar bort agenter vars position är vid sidan om agenten på ett liknande sätt som i 2.4.9. Om agenten hittar en möjlig kollision applicerar beteendet en bromskraft som är beroende av hastigheten hos agenten. Vid höga hastigheter bromsar agenten kraftigare än vid låga hastigheter. Funktionen för att beräkna bromskraften visas nedan.

```
Vec3 breakToAvoidCollision(const SimpleCar& car, const float minTimeToCollision)
{
        float minDistanceToCollision = minTimeToCollision * speed();
        float minDistanceToCenter = minDistanceToCollision + car.radius();

        float totalRadius = car.radius() + radius();

        Vec3 localOffset = car.position() - position();
        float forwardComponent = localOffset.dot (forward());
        Vec3 forwardOffset = forwardComponent * forward();
        Vec3 offForwardOffset = localOffset - forwardOffset;

        bool inCylinder = offForwardOffset.length() < totalRadius;
        bool nearby = forwardComponent < minDistanceToCenter;
        bool inFront = forwardComponent > 0;

        if(inCylinder && nearby && inFront)
        {
                float speedMod = 1.0f;
                if(car.speed() != 0.0f)
                        speedMod = speed() /car.speed();

                Vec3 breakForce = -forward() * speedMod;
                return breakForce;
        }
        return Vec3::zero;
}
```

Det sista beteendet försöker att undvika kollisioner mellan agenter i situationer som det inte går att bromsa sig ur. Detta är ett viktigt komplement till de tidigare beteendena då det även separerar på agenter som hamnat nära varandra. Beteendet söker igenom agentens närhet efter potentiella kollisioner, beteendet räknar på agenternas framtida positioner. I situationer då agenterna rör sig mot varandra styr de båda åt höger eller vänster, beroende på om det är höger- eller vänstertrafik. På så sätt undviker de kollisioner. Om agenterna färdas parallellt fungerar det inte att styra åt ett håll utan agenterna styr då istället åt varsitt håll. När väl beteendet har hittat en möjlig kollision avgörs åt vilket håll agenten ska gira undan, detta visas i funktionen nedan.

```
Vec3 steerToSideToAvoidNeighbor(SimpleCar* threat, bool rightHandedTraffic)
{
        float parallelness = forward().dot(threat->forward());
        float angle = 0.007f;

        float steer = 0;
        if(parallelness < -angle)            //Anti-parallel
        {
                Vec3 offset = hisPositionAtNearestApproach - position();
                float sideDot = offset.dot(side());
                steer = (rightHandedTraffic) ? 1.0f : -1.0f;
        }
        else if(parallelness > angle)        //Parallel
        {
                Vec3 offset = threat->position() - position();
                if(offset.length() <  m_parallellCollisionDistance)
                {
                        float sideDot = offset.dot(side());
                        steer = (sideDot > 0) ? -1.0f : 1.0f;
                }
        }
        Vec3 steerForce = side() * steer;
        return steerForce;
}
```

Figur 23 visar tre agenters färd, agenterna undviker kollisioner genom att ändra riktning.



Figur 23 visar tre agenter som undviker kollisioner.

5.2 Resultat för delmål 1: utvärdering

Under arbetet har två kombineringsmodeller undersökts, *viktad trunkerad summa* och *viktad trunkerad summa med prioritering*, de har använts i båda scenariona. Det visade sig att valet av kombineringsmodell inte påverkar resultatet nämnvärt vid en liten mängd styrbeteenden, då resulterar de olika modellerna oftast i samma styrkraft. Eftersom modellen *viktad trunkerad summa* bedömer alla styrkrafter kommer den med en större mängd styrbeteenden ge en mer urvattnad styrkraft,

styrbeteenden som ger krafter som starkt avviker från medelvärdet inte ger ett lika stort utslag. För att kompensera för detta kan kritiska beteenden få en högre viktning vilket gör att de får större utslag än de andra, att balansera detta kräver många, långa testsessioner. Modell *viktad trunkerad summa med prioritering* fungerar då bättre men den kräver att styrbeteendena prioriteras rätt. Det kan vara problematiskt då det är svårt att förutse påverkan på resultatet med olika ordningar. Överlag rekommenderas den senare modellen då denna är något mindre beräkningsintensiv och hanterar en större mängd beteenden på ett mer korrekt sätt.

I arbetet valdes denna prioriteringsordning för styrbeteendena:

1. Anpassa hastigheten
2. Undvik kollisioner
3. Följ vägen.

Prioriteringen valdes på detta sätt för att skapa ett sammansatt beteende som framför allt undviker kollisioner genom att anpassa hastigheten och som accepterar att agenterna avviker från vägen för att göra detta.

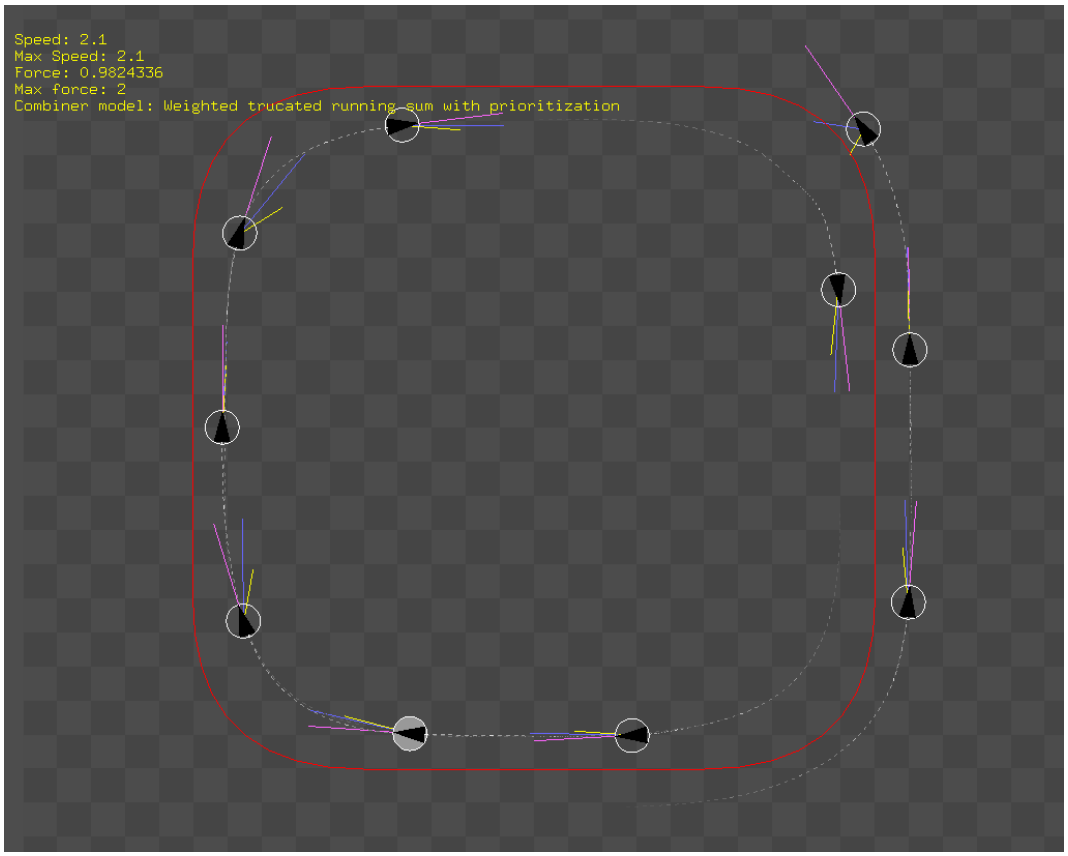
Agenternas beteenden har utökats under arbetet utan några förändringar i den kombineringsarkitektur som tidigare utvecklades. Dessutom har arkitekturen gjort det möjligt att variera kombineringsmodell hos enskilda agenter under körning, något som framförallt har underlättat testningen. Detta visar på att den utvecklade kombineringsarkitekturen varit mycket flexibel och stödjer utvecklaren under arbetet.

5.2.1 Scenario 1

Detta scenario är det mest grundläggande som testar de tre reglerna. Scenariot består av en cirkulär väg med agenter som färdas i båda riktningarna med varierande hastighet. För agenterna uppstår flera olika situationer som deras styrbeteenden måste hantera: agenterna måste följa vägen samtidigt som de hela tiden måste leta efter möjliga kollisioner med mötande trafik och agenterna måste anpassa hastigheten för att undvika att köra in i agenter framför. Detta scenario valdes för att det undersöker och testar flera samverkande grundläggande beteenden som agenterna måste kunna hantera för att kunna köra på en väg.

Varje enskilt beteende fungerar väl för sig och tillsammans i de flesta situationer, men det är framför allt när alla tre styrbeteenden ska verka samtidigt som agenterna kan få problem. Det är situationer då vägen svänger samtidigt som agenterna riskerar att kollidera med andra agenter både framför och vid sidan. När detta sker kan det bildas platser där agenterna fastnar i täta samlingar där det uppstår kollisioner.

Scenariot utvärderades med tio agenter, då uppstår dessa situationer sällan men om vägen blir för tätt trafikerad uppstår situationerna oftare. Så länge genomströmningen är tillräckligt hög klarar agenterna av scenariot bra.



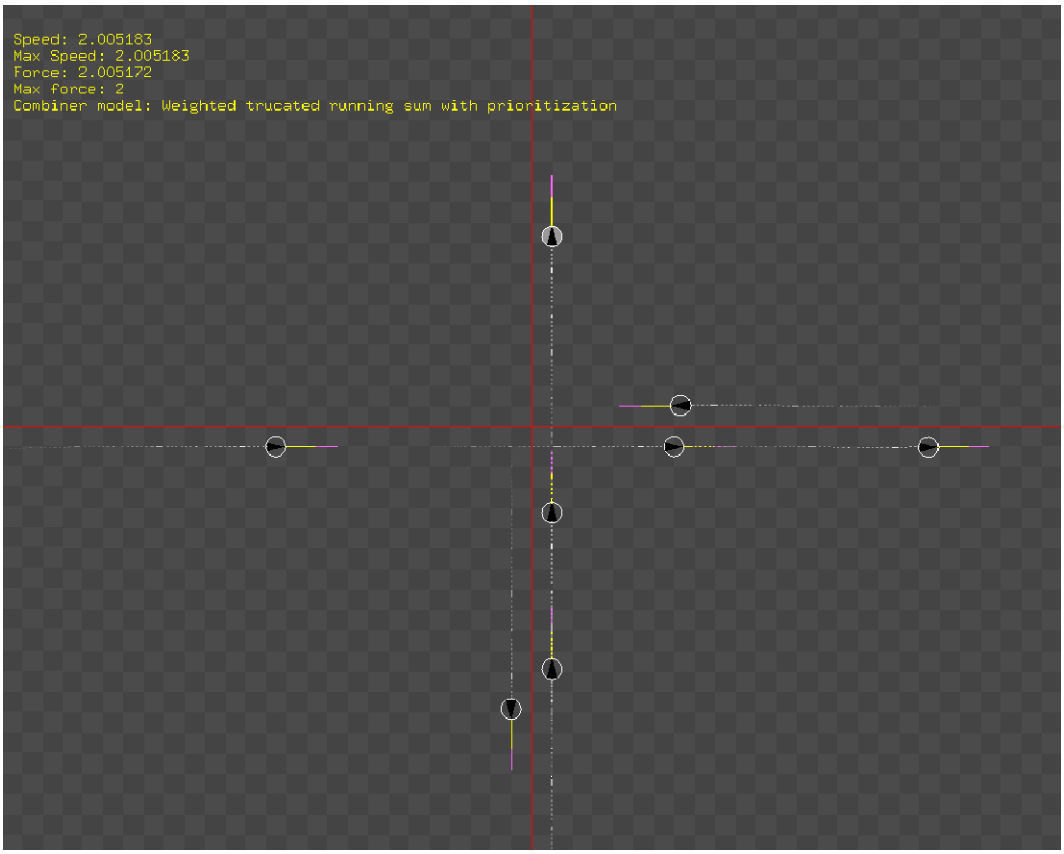
Figur 24 visar en skärmbild från testscenario 1.

5.2.2 Scenario 2

Detta scenario innehåller en vägsträcka där agenterna korsar en väg för att öka komplexiteten och för att agenterna ska bli tvungna att undvika kollisioner med korsande trafik. I detta scenario kan samma situationer som i tidigare scenario uppstå men det kan också uppstå en situation där agenterna måste väja för mötande eller korsande trafik.

Agenterna klarar direkt att navigera i detta scenario utan att viktningen mellan beteendena ändrats. I detta scenario används även beteendet för att undvika kollisioner oftare, då agenterna tvingas passera en korsning. De problem agenterna får är de samma som i det tidigare scenariot men resultatet blir något sämre, agenterna avviker mycket från vägen och kollisioner kan lätt uppstå. Dessa problem skulle antagligen kunna minimeras med mer finjustering av både beteendena och viktningen. Detta verkar vara ett genomgående problem med styrbeteendena som både Reynolds (1999) och Green (2001) identifierat. Dessvärre kan det vara tidskrävande då samtliga scenarion måste köras vid minsta förändring.

Även detta scenario utvärderades med tio agenter, antalet agenter påverkar inte allt för mycket men precis som i det första scenariot så uppstår problem oftare om vägen blir för tätt trafikerad.



Figur 25 visar en skärmbild från testscenario 2.

6 Slutsats

I detta arbete har tre nya styrbeteenden utvecklats för att styra agenter längs en väg. Agenterna anpassar hastigheten till andra agenter och undviker kollisioner samtidigt som de följer vägen. Arbetet visade att styrbeteenden är en flexibel och användbar teknik som resulterar i agenter med ett utbyggbart system för styrning. Arbetet påvisade även problem som uppstår när agenternas rörelse ska begränsas med hjälp av regler. I arbetet utvecklades en arkitektur som gav möjligheten att variera beräkningsmodell för hur flera styrkrafter kombineras.

6.1 Diskussion

Under arbetet har jag upptäckt flera fördelar och nackdelar med styrbeteenden. Det positiva med tekniken är framför allt att det är ytterst enkelt att bygga ut med nya beteenden och att varje beteende är relativt enkelt. Tekniken ger agenterna ett flexibelt beteende och möjligheten att anpassa sig efter rådande situation i miljön. Genom att utveckla varje beteende för sig minskar komplexiteten hos agenten men däremot ställs nya krav på hur flera beteenden kombineras. Detta är styrbeteendenas största svaghet, varje beteende måste värderas mot alla andra som agenten använder. Oavsett vilken kombineringsmodell som används så finns detta bekymmer. En annan styrka hos styrbeteenden är att de fungerar i kontinuerliga miljöer, vilket gör att agenterna inte kräver någon information om miljön i förväg.

Beteendena i OpenSteer (2004) är framför allt anpassade för en kontinuerlig rörelse framåt hos agenterna och beteendena håller ofta en konstant hastighet och ändrar istället bara riktning. Detta passade mindre bra för trafik då hastigheten oftare anpassas än riktningen. Rörelsen i trafiken är nästan en endimensionell rörelse i filens riktning, att återskapa detta med beteenden som ofta verkar i två eller flera dimensioner kan lätt ge problem, då agenterna avviker från vägen i stället för att bromsa. Att använda samma system för både styrning och bromsning gav även problem, de styrkrafter som många beteenden ger är riktade framåt och bromsbeteendena måste därför vara tillräckligt stora för att motverka en hastighetsökning. Detta skulle jag nog vilja påstå är det största problemet med agenterna i sitt nuvarande skick och är det problem som framför allt måste lösas för att kunna utveckla agenterna vidare. Utöver detta fungerar agenterna över förväntan och systemet i helhet fungerar mycket bra.

Agenterna som utvecklades under arbetet är väldigt grundläggande men klarar av de vanliga situationer som de utsätts för. I extrema fall då agenterna ges allt för många motsägelsefulla styrkrafter resulterar det dock i kollisioner eller att de avviker kraftigt från vägen. Den arkitektur som utvecklades för att kombinera flera styrkrafter var ytterst flexibel och gav mig möjligheten att undersöka resultatet från två olika beräkningsmodeller. De olika modellerna presterade likvärdigt i arbetet men detta är främst på grund av att så pass få styrbeteenden användes. Det är till fördel för modellen *viktad trunkerad summa*, vars resultat är mer beroende av antalet styrbeteenden. Kombineringsmodellen *prioriterad slump* har inte undersökts och jämförts med de andra modellerna. Detta skedde främst för att begränsa arbetet på något sätt men det skulle vara fullt möjligt att använda modellen utan större påverkan på resten av systemet, och detta är just styrkan med kombineringsarkitekturen. Styrbeteenden är en mycket användbar teknik som kan vara ytterst flexibel om man väljer rätt arkitektur.

6.2 Framtida arbete

Det finns flera olika infallsvinklar på framtida arbeten men framför allt skulle det vara intressant att utveckla ett eget flexibelt programbibliotek för styrbeteenden. Reynolds abstrakta beteendemodell, 2.3, skulle vara en utmärkt grund för uppdelningen i biblioteket. Dessutom skulle det ge utvecklarna möjligheten att iterativt utveckla biblioteket, fordonsmodeller kan utvecklas efterhand och styrbeteenden kan byggas efter behov. En liknande arkitektur som föreslagits i detta arbete kan användas för kombineringen av flera styrbeteenden. Biblioteket skulle kunna ge utvecklare utan djupare kännedom om tekniken möjligheten att använda styrbeteenden i nya applikationer. Som det fungerar nu är OpenSteer för sammanvävt med visualiseringen för att utan förändringar på biblioteket kunna användas i en ny miljö.

Ett annat bekymmer som har uppstått är hur man ska hantera inbromsningar, de styrbeteenden som används i OpenSteer justerar aldrig hastigheten så för dessa uppstår aldrig dessa bekymmer. Detta borde undersökas vidare, jag kan mycket väl tänka mig att det skulle vara fördelaktigt att ha två separata system, ett för styrning och ett för inbromsning. Det är även intressant att undersöka hur flera bromskrafter ska kombineras och vilka effekter olika modeller av detta ger.

Som fortsatt arbete skulle det även vara intressant att utveckla mer komplexa och mer riktiga trafikregler och se hur väl det går att använda samma teknik för att styra agenterna. Till exempel hade det varit intressant att utveckla ett vägnät där flera trafikregler gäller samtidigt. Stadstrafik skulle vara intressant som exempel på ett avancerat system med flera trafikregler. Där skulle det krävas en bättre representation av vägen som kan klarar av flera filer och som ger bättre projektionspunkter.

Det hade också varit intressant att kombinera styrbeteenden med till exempel evolutionära tekniker och på så sätt evolvera fram viktningen och prioriteringen mellan beteenden. Detta skulle kunna göra det möjligt att slippa för hand justera viktningen och automatisera denna tidskrävande process. Även en mer grundlig undersökning av beräkningsmodeller för hur styrbeteenden kombineras hade varit intressant, det är mycket möjligt att det finns bättre metoder än de tre som beskrivs i denna rapport. Kombinering av flera resultat är inte enbart intressant för detta projekt utan kan även användas där liknande situationer uppstår.

En annan infallsvinkel för framtida arbeten är att undersöka prestanda hos systemet, då alla tekniker som är avsedda för spel har en väldigt snäv prestandabudget. Green (2000) använder ett liknade system i spelet ”Dungeon keeper 2”, men det skulle vara intressant att se hur ett fullskaligt system skulle kunna fungera i spel som till exempel ”Grand theft auto 4”.

Liknande system skulle också kunna användas för andra användningsområden som tillexempel robotik, då denna teknik fungerar mycket väl för kontinuerliga miljöer. Systemet är inte begränsat till två dimensioner utan kan mycket väl anpassas för att fungera för tre dimensioner och kan då tillexempel användas för obemannade farkoster både i luften och under vattnet. I tre dimensioner är det viktigt att kunna hantera kontinuerliga miljöer då det dels krävs stort datautrymme för att spara all information om ett område och dels kan vara omöjligt att förutsäga allt som kan hända i en sådan situation.

Referenser

Buckland, M. (2005). *Programming game AI by example.* Plano, Texas: Wordware Publishing, Inc.

Gamma, E., Helm, R., Johnson, R. & Vlissides, J. (1995) *Design patterns: Elements of reusable object-oriented software.* Boston, Massachusetts: Addison-Wesley Professional.

Green, R. (2000). *Steering behaviours.* SIGGRAPH 2000, April 7, 2000, New Orleans, Louisiana.

Reynolds, C. W. (1987). *Flocks, herds, and schools: A distributed behavioral model*. SIGGRAPH '87 Conference Proceedings (s. 25-34). SIGGRAPH '87, 1987

Reynolds, C. W. (1999) *Steering behaviors for autonomous character.* Game Developers Conference Proceedings (s. 763-782). Game Developers Conference, 1999, San Jose, California

Russell, S. & Norvig, R. (2002). *Artificiall intelligens: A modern approach.* Englewood Cliffs, New Jersey: Prentice Hall

Hemsidor

Adzima, J. (2001). *AI madness: Using AI to bring open-city racing to life*.
Tillgänglig: http://www.gamasutra.com [Hämtad 07.02.28]

Haendel, L. (2005) *The function pointer tutorials* [Hemsida]
Tillgänglig: http://www.newty.de/fpt/index.html [Hämtad 07.05.18]

Schnellhammer, C. & Feilkas, T. (2001). *Steering behaviors* [Hemsida]
Tillgänglig: http://www.steeringbehaviors.de [Hämtad 07.05.07]

Programbiliotek

Reynolds, C. W. (2004) *OpenSteer* (Version: 0.82). [Program Bibliotek]
Tillgänglig: http://opensteer.sourceforge.net [Hämtad 07.02.28]