Adfærdssimulering for autonome agenter

Skjalm Arrøe

Kgs. LYNGBY 2001
EKSAMENSPROJEKT
NR. IMM-THESIS-2001-41

IMM

Trykt af IMM, DTU

Indhold

Figurer

Forord

Dette speciale er skrevet ved numerisk analyse sektionen ved Informatik og Matematisk Modellering (IMM) på Danmarks Tekniske Universitet (DTU).

Projektet er udarbejdet af Skjalm Rosenkjær Arrøe, c958145, under vejledning af lektor Per Skafte Hansen.

Skjalm Arrøe

Resume

Simulering af menneskers adfærd i forskellige situationer har i mange år været anvendt i meget forskellige sammenhænge. Interessen for dette område varierer fra simuleringer af krisesituationer over trafikplanlægning til virtuelle mennesker i interaktive applikationer, for eksempel computerspil.

Formålet med dette eksamensprojekt har været at afklare, hvad der skal til for at simulere grupper af mennesker ved brug af autonome agenter. Dette er gjort ved at implementere et simpelt system, der kan fungere som prototype for et generelt simuleringssystem. Resultaterne fra arbejdet med prototypen er beskrevet og analyseret således, at der er fremkommet en model for det endelige systems udseende.

Abstract

The simulation of different kinds of human behaviour has been used in many different contexts for years. The areas of interest range from simulation of crises to traffic planning to the simulation of virtual humans in interactive applications such as computer games.

The purpose of this project is to clarify what is needed to simulate groups of people by using autonomous agents. This is achieved by implementing a simple system, which functions as the prototype for a general simulation system. A description of such a general simulation system is developed based on the documentation and analysis of the prototype.

Keywords

Autonomous agents, artificial intelligence, behaviour, simulation

Kapitel 1

Indledning

Med de seneste års hastige udvikling og øgede udbredelse af computere er der opstået stor interesse for gengivelser af verden, som vi oplever den. Mest kendt er nok brugen af computergrafik til at lave næsten fotorealistiske billeder og film af den fysiske verden, men der findes også andet end det umiddelbart synlige, der kan gengives ved brug af computere: gengivelser, forudsigelser og simuleringer af menneskers adfærd.

Anvendelsesområderne for denne type simuleringer spænder fra så alvorlige områder som krisesimuleringer over analyser af trafikfordeling til ren underholdning i form af virtuelle personer i computerspil og andre virtuelle applikationer. Et nærtbeslægtet forskningsområde er design af styreprogrammerne til de såkaldte AGV'er, Autonomous Guided Vehicles, der kan ses som en fysisk eksisterende udgave af autonome agenter. AGV'er benyttes til at udføre tankekrævende opgaver i omgivelser, der er farlige eller utilgængelige for mennesker. Dette kan være målinger i bunden af aktive vulkaner, udforskning af Mars' overflade eller gennemsøgning af søbunde.

Der vil ikke i rapporten være en detaljeret gennemgang af de forskellige typer af AGV'er. Er læseren interesseret i dette emne henvises der til [Borenstein], [Olsen] og Institut for Automation på DTU, `http://www.iau.dtu.dk/`.

En af de metoder, der blandt andet er meget udbredt i adfærdssimulering, er at benytte autonome agenter til at simulere menneskers adfærd. En autonom agent er ikke blot bevidst om den omkringliggende verden, men har også nogle generelle ønsker og mål. Ud fra de stimuli, agenten modtager, vil den forsøge at få opfyldt disse mål ved konstant at forholde sig til verden og tilpasse sig de umiddelbare omgivelser.

Brugen af disse autonome agenter giver med en velgennemtænkt implementering en realistisk simulering af den foreliggende situation. Det er her vigtigt at notere sig, at der med *realistisk* ikke menes, at man har en simulering, der er en nøjagtig gengivelse af en eksisterende hændelse. Ej heller menes der en nøjagtig forudsigelse af en fremtidig hændelse. Simuleringernes realisme måles subjektivt: i hvor høj grad kan de simulerede mennesker tænkes at være virkelige?

Formålet med dette projekt er at undersøge, hvad der skal til for at kunne lave sådanne simuleringer og udvikle en prototype, der kan benyttes til at undersøge de metoder, der er nødvendige for at lave et komplet simuleringssystem.

Baggrund for projektet

I løbet af de seneste år har forfatteren arbejdet med forskellige aspekter af numerisk analyse, logistik, kunstig intelligens, computer grafik og robotstyring. Specifikke projekter inden for disse områder inkluderer:

Korteste vej søgning
: Implementering af korteste vej søgning på et billede, hvor værdien af de enkelte pixels repræsenterer forskellige terrængtyper i forbindelse med kurset Lokalisering, Layout og Distribution på Danmarks Tekniske Universitet. Både den klassiske Dijkstras algoritme, beskrevet i [Nielsen] og [Cormen], og den heuristiske udvidelse A*, [Østerby], [Jönsson] og [Patel], er implementeret.

Collision detection
Specialkursus hos Per Skafte Hansen omhandlende collision detection i computer grafik. I kurset blev der udviklet et simpelt 3D grafik system med tilhørende collision detection, blandt andet ved brug af axis aligned bounding boxes til at effektivisere systemet.

Autonome robotsystemer
Kursus ved Institut for Automation, hvor hovedformålet er at lære de grundlæggende metoder til styring af AGV'er. Instituttet har et antal robotter, der benyttes til at eksperimentere med brugen af forskellige typer af sensorer, primært afstandssensorer og lyssensorer. Afslutningen på kurset var en gennemkørsel af en lille forhindringsbane i stil med den, der benyttes ved instituttets Robocup konkurrence, `http://www.iau.dtu.dk/robobcup/`.

Robotvision Som fortsættelse af ovennævnte kursus blev der i foråret 2001 arbejdet selvstændigt med at benytte et kamera til brug ved streggenkendelse[1] i Robocup konkurrencen. Dette arbejde resulterede i udarbejdelsen af en algoritme, der ud fra et kamerabillede benytter polynomiefit til at forudsige, hvordan stregen forløber frem til cirka en meter foran køretøjet.
Det endelige system kunne i en stor del af de udførte tests lave en korrekt genkendelse af, om der var et lige stykke, et sving eller en forgrening foran køretøjet. Projektet, samt resultater, er beskrevet i [Arrøe].

Efter at have arbejdet med en del forskellige, lavniveau teknikker har det været forfatterens ønske at kombinere erfaringerne fra disse projekter med henblik på at lave et samlet system til simulere eller styre autonome agenter, køretøjer og robotter.

Den oprindelige ide til projektet var at lave et system, der kunne koordinere de enkelte agenters bevægelsesmønstre, så et givent problem blev løst optimalt. Dette kunne eksempelvis være optimal udnyttelse af et eksisterende vejnet, hvor et centralt system fortæller de enkelte trafikanter, hvilken rute de skal benytte. Specielt kunne det have været interessant at benytte systemet i praksis ved brug af et antal AGV'er.

Før projektets start fremkom Per Skafte Hansen dog med ideen om at forsøge at simulere situationen på Expo2000. Dette satte gang i en del tanker om, hvordan individer i en gruppe bevæger sig i forhold til hinanden og, hvordan dette kan simuleres. Specielt bliver det interessant, når individerne har hver deres mål, de ønsker at opfylde, men der er så mange af dem, at de nødvendigvis må gå i vejen for hinanden før eller siden.

Herefter har projektet udviklet sig til at undersøge mulighederne for at simulere denne type autonome agenter og opstille en model for, hvordan et generelt simuleringssystem kan opbygges.

Rapportens opbygning

Projektet tager udgangspunkt i overvejelser om, hvilke elementer de simulerede agenter og verdner består af, hvilke mål/ønsker agenterne kan tænkes at have, samt hvordan agenterne kan løse problemet: "hvordan opnår jeg mit mål?". Kapitel 3 og 4 indeholder disse overvejelser samt resultaterne af dem.

Herefter følger i Kapitel 5 en beskrivelse af de såkaldte Behaviours, agenterne benytter til at imødegå komplikationer i deres færden i verden.

Kapitel 6 gennemgår den i projektet udviklede prototype og opsummerer, hvilke erfaringer arbejdet med den har givet. Disse erfaringer fører i Kapitel 7 til en model for, hvordan et større, mere generelt simuleringssystem kan implementeres. Der vil her være en gennemgang af forskellige metoder, der kan benyttes til at simulere agenternes kommunikation med hinanden og deres observation af verden.

Sidste del af rapporten indeholder nogle eksempler på den type situationer, man vil kunne simulere med det endelige system. Specielt er en af situationerne, *Synkende Skib*, simuleret ved brug af den implementerede prototype, og Kapitel 8 vil derfor hovedsageligt omhandle denne simulering.

De i rapporten benyttede termer er forklaret i det omfang, det er nødvendigt for at få et helhedsbillede af, hvad adfærdssimuleringer for autonome agenter er. Dog vil en forhåndsviden om programmering, softwaredesign, grafteori og kunstig intelligens øge udbyttet af rapporten, da en detaljeret beskrivelse af alle de benyttede systemer og metoder falder uden for rapportens rammer.

[1] Kort fortalt går Robocup konkurrencen ud på at bygge et autonomt køretøj, der kan følge en hvid streg. Se `http://www.iau.dtu.dk/robocup/` for en detaljeret beskrivelse af konkurrencen.

Da størstedelen af litteraturen er på engelsk eller amerikansk, er der benyttet engelske ord for de vigtigste nøglebegreber. Dette valg er foretaget for at undgå misforståelser som følge af dårligt dækkende betegnelser eller manglende begreber på dansk. I de situationer, hvor der ikke kan herske tvivl om meningen med et ord, eksempelvis "agent", benyttes de danske ord og bøjninger for at få en mere flydende sætningskonstruktion.

I rapporten er der benyttet små programstumper til at illustrere de beskrevne metoder. Der er i disse tilfælde benyttet ANSI C syntaks, og programstumperne er taget direkte fra kildekoden til prototypen eventuelt med kosmetiske modifikationer. Denne fremgangsmåde er valgt for at gøre det lettere for læseren at identificere de dele af kildekoden, der svarer til de gennemgåede metoder. [2]

Når der i rapporten benyttes begreber hentet direkte fra kildekoden benyttes der en skrifttype med fast bredde, eksempelvis `simplevehicle`. Kursiv benyttes til at fremhæve vigtige ord og således undgå misforståelser i læsningen af teksten.

[2] Selve kildekoden er ikke inkluderet på i denne rapport, men er at finde på den tilhørende CD og som en separat udgivelse.

Kapitel 2

Problemanalyse

Som nævnt i indledningen er formålet med dette projekt at undersøge de metoder, der skal til for at kunne lave adfærdssimuleringer. Dette kapitel svarer til den i [Hansen], Appendix 7, beskrevne User Requirements Phase og giver en præcisering af, hvad en bruger skal kunne med simuleringssystemet. I forbindelse med denne analyse er der en diskussion af de afgrænsninger, der er foretaget, i forbindelse med brugerens muligheder.

2.1 Kravspecifikation

Det umiddelbare krav til systemet er, at det skal give brugeren mulighed for at definere og efterfølgende se en simulering. Disse krav vil her blive udspecificeret, så de står mere klart.

2.1.1 Brugerinput

Brugeren skal have mulighed for at definere en simulering. Der skal derfor udformes en brugergrænseflade, der gør dette muligt. Til dette er der tre forskellige muligheder:

Programmeringsinterface
: Hele systemet kunne udvikles som et API[1], der giver brugeren mulighed for at programmere sine simuleringer. Dette vil gøre det muligt for brugeren at påvirke den måde, de enkelte dele af systemet benyttes på. Til gengæld vil det kræve, at brugeren er bekendt med programmering og kan sætte sig ind i brugen af et nyt API.

Dataformat
: Alternativt kan der defineres et dataformat til at beskrive simuleringer bestående af omgivelser, antallet samt typer af agenter og tilsvarende. Dette vil flytte brugergrænsefladen et skridt væk fra at være programmering, men vil for brugeren (specielt den ikke-tekniske) kunne virke besværligt og tungt.

Grafisk værktøj
: Endelig kan man vælge at udvikle et grafisk værktøj, der giver brugeren mulighed for at "tegne" simuleringerne. Et sådant værktøj vil lette arbejdet med at definere simuleringerne, idet agenter vil kunne tilføjes med den velkendte "træk og slip" metode.

Idet dette system tænkes anvendt af brugere med begrænset eller ingen programmørmæssig baggrund, bliver kravet til selve systemet, at det skal indeholde et dataformat, der kan benyttes til at definere simuleringerne. Når dette dataformat og resten af systemet er færdigudviklet kan der bygges et grafisk værktøj, så det bliver let for brugeren at definere simuleringer.

[1] Application Programmer's Interface

2.1.2 Indhold af simulering

En simulering skal som minimum indeholde et antal agenter og den verden, de eksisterer i. En komplet gennemgang af de autonome agenter gives i Afsnit 3.2, men de vigtigste egenskaber for dem og den simulerede verden er:

Agenter
En agent er i start situationen defineret ved
- en position i verden
- et eller flere problemer, den skal løse
- adfærdsmønstre knyttet til løsningen af problemerne
- en fysisk repræsentation

De mulige problemtyper og deres løsninger er beskrevet i Kapitel 4, den fysiske repræsentation er diskuteret i Afsnit 3.2.

Verden
Simuleringerne vil foregå i en simplificeret udgave af den fysiske verden. Brugeren skal have mulighed for at definere denne simple verden som indeholdende:
- statisk kort indeholdende forskellige terrænområder såsom skove, veje og åbent landskab
- en liste af fysiske objekter, der ikke kan betragtes som en del af det underliggende landskab

Brugeren har ikke mulighed for at påvirke, hvilke fysiske love, der gælder for systemet.

Disse krav til indholdet er fremkommet ud fra overvejelser om, hvordan verden kan simuleres. Det er værd at notere sig, at der ikke her foreligger en nøjagtig beskrivelse af, hvordan et fysisk objekt i verden ser ud, blot at der skal være mulighed for at angive et sådant. Dette skyldes, at valget om den endelige (geometriske) repræsentering af verden hører hjemme i systemdesignet, snarere end i kravspecifikationen.

2.1.3 Fremvisning og lagring af simulering

Selve fremvisningen af simuleringen skal kunne foregå på flere forskellige måder. Brugeren skal have mulighed for i sin definition af simuleringen at angive enten real time simulering, eller at systemet skal opdateres med faste intervaller af en given længde.

Under en real time simulering skal brugeren have mulighed for at se simuleringen grafisk og styre fremvisningen ved brug af funktioner af typen "Pause", "Genstart", "Ændring af synsfelt" og afslut.

Vælges der opdatering med faste intervaller skal simuleringen ikke vises under selve simuleringen, men derimods gemmes både som rådata og grafisk som en billedsekvens, som brugeren senere kan gennemse og analysere.

2.2 Afgrænsninger

Ovenstående gennemgang omhandler hovedsageligt, hvad brugeren *har* mulighed for. Udover disse ting er der gjort overvejelser om en del ting, som brugeren ikke har mulighed for:

Interaktivitet
Den største afgrænsning af systemet er, at brugeren ikke har mulighed for at påvirke agenterne og omgivelserne under selve simuleringen. Dette forhindrer også brugeren i dynamisk at tilføje eller fjerne agenter. Yderligere betyder dette, at alle hændelser, det vil sige type af hændelse og hvornår den sker, skal defineres på forhånd gennem brugergrænsefladen. Det er således ikke muligt at brugeren interaktivt aktiverer dem. Dette er primært valgt for at simplificere systemet, specielt den interaktive fremvisning af simuleringerne. I det øjeblik brugeren har mulighed for interaktivt at påvirke systemet kan det give problemer at genskabe simuleringerne.
Når dette er sagt skal det dog nævnes, at denne funktionalitet ville være et stort plus for systemet, da det vil give brugeren en meget større frihed.

Fysik

Derudover er der lagt en kraftig begrænsning på den simulerede verden, specielt med henblik på fysikken. Der er reelt set ikke noget minimumskrav for, hvad der skal gøre sig gældende, men tyngdekraft og objekt- og agentadskillelse kan betragtes som et minimum.

Begrundelsen for dette valg er, at systemets primære formål er at simulere adfærd og ikke en fysisk verden. Jo mere korrekt fysik, der indbygges i systemet, jo mere realistiske vil simuleringerne fremstå. Mere kompliceret fysik vil dog gøre systemet beregningsmæssigt tungere, hvilket kan komplicere eller umuliggøre real time fremvisning.

En komplet beskrivelse af de fysiske love, der skal gælde, hører ikke hjemme i specificeringen af brugerkravene, men bør alligevel nævnes her, da de har indflydelse på, hvor (fysisk) realistisk simuleringen vil fremstå for brugeren.

Kapitel 3

Autonome Agenter

3.1 Definition

En autonom agent er (en del af) et stykke software, der i et eller andet omfang er bevidst om sig selv og sine omgivelser. Graden af bevidsthed afhænger af den præcise type af agent, ligesom definitionen af omgivelser kan variere.

Nedenfor følger en kort beskrivelse af nogle forskellige typer af agenter. Idet dette projekt kun beskæftiger sig med én af typerne, vil de øvrige ikke blive beskrevet nøjere.

- **isolated agents** er agenter, der ikke (direkte) er i kontakt med andre agenter, eksempelvis agenter benyttet i forbindelse med data mining.
- **situated agents** befinder sig i en egentlig verden befolket af andre agenter. Disse kan yderligere inddeles:
  - **reactive agents** er styret af eksterne stimuli og deres egne (instinktive) reaktioner på disse.
  - **deliberate agents** er mere intelligente end ovenstående, forstået på den klassiske kunstig intelligens måde. Disse agenter vil ikke blot reagere på, hvad de ser nu og her, men vil forsøge at planlægge længere ud i fremtiden.

Herudover kan en autonom agent enten eksistere i en ren abstrakt verden bestående udelukkende af information, igen er data mining et eksempel, eller den kan være embodied i (en simulering af) en virkelig, fysisk ting, for eksempel en AGV. Hvis agenten ikke er en fysisk ting, men en (computer-) simulering af en sådan, siges agenten at være virtuel.

De i [Reynolds] beskrevne Autonomous Characters er situated, embodied, reactive, virtual agents. Det vil sige, at de befinder sig i en virtuel verden befolket af andre agenter og handler udfra de stimuli de modtager. Denne verden er dog en model af den fysiske verden og ikke blot en abstrakt samling data.

I dette projekt tages disse agenter et skridt videre, så de får en blanding af at være reactive og deliberate. Eksempelvis vil agenterne ikke blot gå rundt i verden, enten tilfældigt eller efter et enkelt mål, de vil også forsøge at definere en liste af mål udfra deres overordnede ønsker og mål. Denne planlægning antager forskellige former, hvoraf en af de simpleste er at finde den korteste vej fra punkt A til punkt Z som vejen fra A til B til C til ... til Z. Agenterne vil således være i stand til at opdele et stort problem i et antal mindre problemer.

3.2 Repræsentation

For at beskrive agenterne i en virtuel verden er det nødvendigt at definere de enkelte dele, en agent består af. Der findes forskellige måder at lave denne inddeling på, se eksempelvis [Reynolds] og Kapitel 4 i [Watson]. Den primære forskel på disse to modeller er, at Reynolds tager udgangspunkt i selve agenten og deler denne op i tre forskellige lag, hvorimod Watson lader hvert af sine `game_objects`[1] have en liste af agenter, der påvirker det. Forskellen på de to modeller ligger i den måde, Reynolds og Watson benytter begrebet "agent". Reynolds

[1] Et `game_object` er et implementeringsspecifik begreb Watson benytter, snarere end et konceptmæssigt begreb.

vælger at lade en agent beskrive et individ i sin verden, Watson lader et `game_object` beskrive et individ og benytter sine agenter til at påvirke dette objekts adfærd.

I dette projekt benyttes en udvidelse af den model, Reynolds gennemgår. Den primære udvidelse består i en beskrivelse af den anatomiske repræsentation af agenten. En del af denne repræsentation er lånt fra psykologien, nemlig agentens personlige sfære.

De nedenstående afsnit giver en oversigt over de forskellige dele af en agent.

3.2.1 Problemløsning

De autonome agenter skal være i stand til løse forskellige typer af problemer i verden. Dette gøres ved at lade hver agent have adgang til en række problemløsere, den kan benytte til at lægge en strategi for, hvad den skal foretage sig. En gennemgang af en række problemer, agenterne kan tænkes at stå overfor, er at finde i Kapitel 4. Fælles for alle problemtyperne er, at de lægger op til, at agenten skal bevæge sig mellem forskellige steder i verden.

3.2.2 Adfærd

Adfærd er for dette projekt den vigtigste del af agenternes repræsentation, idet en agents adfærd reelt set bestemmer alle agentens beslutninger, mens den bevæger sig gennem verden.

Når en agent har lagt en strategi for, hvordan den kan løse et givent problem, har den behov for at kunne bevæge sig rundt i verden. Selve den fysiske del af bevægelsen, såsom repræsentation af position og hastighed, beskrives nedenfor.

Agentens generelle holdninger og mål bestemmer, hvilken adfærd agenten beslutter sig for at udvise, og beslutningsprocessen er derfor afhængig af disse. Derudover sker der i beslutningsprocessen en analyse af agentens omgivelser, blandt andet med henblik på at undersøge, hvordan det fremtidige bevægelsesmønster kan være. Agenten forsøger således at kombinere tre ting:

- den fundne løsning på agentens problem(er)
- agentens generelle type
- omgivelserne i nærheden af agenten

Formålet med denne vurdering af situationen er, at forskellige typer agenter kan løse de samme problemer på forskellige måder afhængig af deres holdning og omgivelser.

Eksempel 3.1: Typer af agenter
Et eksempel på to forskellige typer af agenter er rovdyr og byttedyr. Disse to typer af agenter vil have vidt forskellige adfærdsmønstre, men findes i de samme omgivelser, og deres mål er nært beslægtede. Begge typer vil bevæge sig rundt i verden, og deres primære mål er at lede efter føde.

Føden findes forskellige steder i verden, så agenterne skal via deres problemløsning beslutte, hvor de skal bevæge sig hen for at få så meget føde som muligt. Mens de bevæger sig hen til føden, skal de undgå både fysiske forhindringer og andre agenter i verden, men selvom de har de samme metoder stillet til rådighed, vil de vælge at gøre det forskelligt afhængig af, om de er rovdyr eller byttedyr.

3.2.3 Bevægelse

Bevægelsesdelen af en autonom agent sørger for, at agenten har mulighed for at bevæge sig rundt i den simulerede verden. Dette betyder, at agenten gennem denne del skal have nogle grundlæggende og veldefinerede egenskaber. Reynolds' bevægelsesdel er et Simple Vehicle, der er beskrevet som havende følgende egenskaber:

- vægt (skalar)
- position (vektor)

- hastighed (vektor)
- maksimalstørrelse af påvirkende kraft (skalar)
- maksimal hastighed (skalar)
- retning (skalar)

Som et alternativ til at have en fast maksimal hastighed og maksimal kraft kan der laves en hastigheds- og retningsafhængig maksimal kraft og eventuelt en retningsafhængig maksimal hastighed, der repræsenterer de fysiske begrænsninger, et Simple Vehicle er underlagt. Eksempelvis kan man forestille sig, at et Simple Vehicle er en bil, der ikke må overskride vejens hastighedsgrænser og ikke må accelerere, decelerere eller dreje så kraftigt, at bilens hjul skrider på underlaget.

Formålet med at have et Simple Vehicle som en underdel af en agent er, at det vil være muligt at definere forskellige typer af Simple Vehicles, der kan knyttes til en agent. På denne måde vil det være muligt at benytte samme type Simple Vehicle, eksempelvis et menneske, til flere forskellige typer af agenter. Yderligere vil det i definitionen af simuleringerne være muligt udskifte typen af en agents Simple Vehicle, så man kan simulere et menneske, der kører i en bil.

En anden måde at beskrive bevægelsesdelen ville være ud fra en af IAUs AGV'er. Idet en AGV er underlagt er mere begrænsede i deres bevægelser end Reynolds' Simple Vehicles vil der være brug for en anderledes model. AGV'erne er differentialstyrede og de muligheder, der er for at bevæge dem, er at sætte hastigheden for hver af de to drivende motorer. Denne motorstyring kan enten simuleres ved at modificere de senere beskrevne Behaviours eller ved at lave en konvertering fra Reynolds' hastigheds- og kraftvektorer til motorhastigheder for højre og venstre motor. Dette vil dog blive kompliceret af, at motorerne ikke kan styres kontinuert, man sættes kun kan sættes til faste hastigheder i det diskrete interval fra -127 til $+127$. Disse værdier er bestemt af den underliggende hardware og tilhørende styringssoftware og repræsenterer "fuld fart bagud" og "fuld fart frem".

Udover at det er nødvendigt at gøre konverteringen fra vektorer til motorhastigheder kompliceret giver de diskrete motortrin problemer, når der skal foretages finjusteringer i styringen. Dette kan observeres, hvis man forsøger at få en AGV til at køre i en cirkel med en bestemt radius. Man vil her have to muligheder: enten bruger man en cirkelradius, der passer med to hastigheder på motorerne, eller også laver man små reguleringer under kørslen. Den første mulighed giver ganske pæne cirkler, men kun for bestemte radier. Den anden mulighed giver den ønskede radius, men kørslen vil være ujævn.

I projektet benyttes Reynolds' model for et Simple Vehicle, men AGV'en som bevægelsesdel nævnes for at vise, at det er muligt at benytte en fysisk ting fra den virkelige verden som bevægelsesdel. Og yderligere er AGV'en valgt for at give læseren et indtryk af de problemstillinger man skal forholde sig til, hvis man gør det. I og med at AGV'erne er differentialstyrede vil de kunne rotere om deres eget centerpunkt, men hvis der benyttes et køretøj med en anden type styring, for eksempel Ackerman styring, vil arbejdet med at konvertere fra hastigheds- og kraftvektorer til hastighed af de drivende motorer samt servoposition blive væsentlig større. En beskrivelse af bevægelsesligningerne for Ackerman og andre typer styring kan findes i [Borenstein] sammen med beskrivelser af, hvordan en robot i stil med AGV'erne kan opfatte sine omgivelser ved brug af forskellige sensortyper.

3.2.4 Anatomisk repræsentation

Idet en agent kan tænkes at være en repræsentation af mange forskellige former for selvtænkende individer fra den virkelige, verden er den anatomiske repræsentation adskilt ikke blot fra selve agenten, men også fra dens bevægelsesdel. Den anatomiske repræsentation af agenten vil således være en død ting, der reelt set bliver flyttet rundt af agentens Simple Vehicle og de fysiske love, der gælder i den simulerede verden.

Udover en geometrisk model til brug ved grafisk fremvisning af simuleringerne har en agent også en personlig sfære. Dette begreb er lånt fra psykologien, hvor det repræsenterer den afstand, mennesker ønsker at holde til andre, specielt fremmede. Hvis en (fremmed) person kommer for tæt på, vil man føle sig klemt og ubehageligt tilpas. Der vil ikke i dette projekt blive foretaget en nærmere analyse af størrelsen af den personlige sfære. Det skal dog nævnes, at der indenfor sociologien er udført adskillige eksperimenter med effekten af folks afstand til hinanden, eksempelvis [Argyle] og [Aiello].

I dette projekt benyttes en personlig sfære beskrevet som en cirkel med en radius på $0,8m$. Denne størrelse er valgt ud fra Argyle og Deans eksperimenter, beskrevet i [Argyle], hvor det er vist, at en afstand på to fod, cirka $0,6m$, til en anden person resulterer i ubehag. Hvorfra disse $0,6m$ måles fremgår ikke umiddelbart af artiklen.

Det er her antaget, at det er fra "ydersiden" af testpersonerne, så i dette projekt benyttes en afstand på $0,8m$ fra agentens centrum. Dette repræsenterer, at agenten har en fysisk krop inden for den personlige sfære.

Endvidere gør Argyle og Dean opmærksom på, at positionen af andre personer også har indflydelse på, hvor stort et ubehag man føler. Man er generelt mere tolerant overfor folk ved siden af eller bag ved en selv. Dette lægger op til, at man burde benytte en anden form end en cirkel, men for at simplificere

Valget af størrelsen og formen af den personlige sfære skal på ingen måde ses som et postulat for, at mennesker vil holde en afstand på $0,8m$ til hinanden, men blot som en anerkendelse af, at det ikke kun er et menneskes fysiske krop, der påvirker den måde det bevæger sig på.

Kapitel 4

Problemløsning

Noget af det, der er med til at definere en autonom agents intelligens, er dens evne til at tage et overordnet problem og splitte det op i mindre, mere overskuelige delproblemer. Denne problemopsplitning vil ikke løse selve problemet, men er et skridt på vejen, idet delproblemerne hver for sig almindeligvis er simple at løse.

Der er mange forskellige typer af problemer, en autonom agent kan blive udsat for. Denne gennemgang skal ikke ses som en udtømmende liste af planlægningsproblemer for autonome agenter, men derimod som en liste af de problemer dette system vil være i stand til at løse i den endelige udgave.

4.1 Problemtyper

Den primære type af problemer, dette system er rettet mod, har enten med bevægelser i en fysisk verden eller med planlægning at gøre. Sidstnævnte vil faktisk ligge temmelig tæt op af førstnævnte idet opgaveløsninge typisk vil involvere at agenten bevæger sig hen til et bestemt sted i verden og tilbringer noget tid der.

4.1.1 Vejsøgning

Dette er en af de konceptmæssigt simpleste problemtyper: agenten befinder sig i punkt A i verden og ønsker at bevæge sig frem til punkt B. For at kunne løse dette problem kræver det, at agenten har adgang til en repræsentation af verden, en repræsentation hvis detaljeringsgrad vil være afgørende for, hvor "god" en vej agenten vil kunne finde.

Definitionen af en "god" vej varierer afhængig af, hvad agenten egentlig ønsker. Basalt set er der to forskellige mål for en "god" vej, længde og hastighed, men det er også muligt at benytte en helt tredje vægtning af kanterne. Dette kunne eksempelvis være, hvor mange penge det vil koste at benytte en given kant mellem to knuder. Agenten må lave en afvejning af, om den ønsker at komme hurtigt frem til målet uden at tænke på udgifterne, prisen eller længden af den benyttede vej, eller om det er vigtigere at få minimeret den fysiske afstand agenten bevæger sig.

Eksempel 4.1: Rejse fra Roskilde til Århus
Lad os antage, at en agent repræsenterer en mand, der ønsker at rejse fra Roskilde til Århus. I den simulerede verden har han tre forskellige måder at komme frem på:

Apostlenes heste
: Han kan vælge at gå (og svømme) hele vejen. Dette vil ikke koste ham noget, men vil tage lang tid. Idet han ikke er bundet til at skulle blive på veje eller togspor kan han bevæge sig i en lige linie og opnår derved den mindste euklidiske afstand.

Tog
: Anden mulighed er at tage toget over Fyn. Dette er en både længere og dyrere tur, men til gengæld er rejsetiden blevet væsentligt forkortet.

Fly
Sidste mulighed er at tage til København og flyve videre derfra. For illustrationens skyld antager vi, at denne tur er længere end togturen på grund af omvejen til København, men på grund af flyveturen tager den kortere tid. Prisen for at flyve er væsentlig højere end togturens.

Hvis agenten ønsker at optimere den fundne vej med henblik på pris og/eller afstand vil han vælge at gå. Hvis han vil have den hurtigste tur uafhængig af prisen og afstanden vælger han at flyve. Laver han derimod en vægtning, hvor han ønsker at få en relativt hurtig og relativt billig tur, vil han ende med at vælge togturen.

Skulle man lave dette eksempel mere realistisk ville en fjerde omkostning, "fysisk udmattelse", være god at have med. Godt nok ville det være gratis at gå og svømme hele vejen, men turen ville have en anden pris.

Eventuelt kan man vælge at inddrage slitage på beklædningsgenstande samt den ekstra væske og føde, der skal indtages, som fordyrende elementer.

Til løsning af vejsøgningsproblemer findes der en række forskellige algoritmer varierende fra Dijkstra's algoritme beskrevet i [Cormen] til de heuristiske A* og D* søgninger. Der findes utallige beskrivelser af A*. [Østerby] giver en teoretisk gennemgang af A* til brug for søgetræer i kunstig intelligens mens [Patel] og [Jönsson] giver en mere implementeringsorienteret gennemgang. D* er en udvidelse af A*, der kan tage højde for ændringer i og manglende dele af terrænet efterhånden som agenten når til disse områder. Algoritmen er beskrevet i [Stentz].

4.1.2 Travelling Salesman Problem

Et andet grafteoretisk problem, agenterne kan have behov for at løse, er rejsende handelsmand problemet, herefter kaldet TSP fra det engelske Travelling Salesman Problem, der kort fortalt går ud på, at alle knuder i en graf skal besøges og den tilbagelagte distance skal være mindst mulig. Som ved vejsøgning er det muligt at benytte et andet vægtmål end fysisk afstand for distance, eksempelvis kunne det være tidsforbrug.

Hvis agenten kun skal besøge en delmængde af grafens knuder vil det være muligt at kombinere vejsøgning og TSP til at løse problemet. Med korteste vej findes afstandene mellem TSP knuderne og disse benyttes til at danne en ny graf, hvorpå TSP løses.

TSP adskiller sig kraftigt fra vejsøgningen idet TSP er NP komplet. Det betyder, at de eksakte TSP løsere, der findes, har en eksponentiel køretid og derfor tager utrolig lang tid at udføre, efterhånden som antallet af knuder i grafen vokser. Som et alternativ til de eksponentielt voksende algoritmer kan man benytte en ikke-optimal, men væsentlig hurtigere algoritme, til at give en "god" løsning. "God" betyder her ikke optimal, men så tæt på optimal at det kan accepteres i forhold til beregningstiden.

I [Cormen] findes en algoritme, der kan give en løsning til TSP, som er højst dobbelt så lang som den optimale. En komplet gennemgang af algoritmen falder uden for dette afsnits mål, men det bør nævnes, at algoritmen har en polynomiel køretid på $O(V^2)$, hvor V er antallet af knuder i grafen.

Alternativt kan man benytte en branch and bound metode til at mindske antallet af kombinationsmuligheder for problemet. En sådan metode er beskrevet i [Jacobsen].

4.1.3 Opgaveudførelse

For TSP gælder, at agenten skal rundt til alle knuder i grafen på den billigste måde. En anden problemstilling er task scheduling, hvor der er givet en række opgaver samt deres indbyrdes afhængighed.

Task scheduling problemer kan eksempelvis løses ved at opstille en afhængighedsgraf for opgaverne, hvor orienteringen af kanterne kan opfattes som "startknuden skal være færdig før slutknuden kan påbegyndes". Såfremt denne graf er acyklisk vil det være muligt at finde opgavernes indbyrdes rækkefølge ved at lave en topologisk sortering af grafen som beskrevet i [Cormen].

Eventuelt kan man efter den topologiske sortering udvide grafen med information om, hvor i verden de enkelte opgaver skal udføres, samt hvor lang tid det tager at bevæge sig mellem disse steder. Ud fra dette vil agenten kunne optimere den rækkefølge opgaverne udføres i. Det er naturligvis et krav, at rækkefølgen bliver overholdt,

men idet der kan opstå situationer, hvor agenten kan vælge mellem to eller flere opgaver er der mulighed for optimering.

En sådan optimering involverer, at man undersøger de forskellige rækkefølger agenten kan vælge udføre de enkelte opgaver på uden at bryde den topologiske sortering. Dette giver ligesom TSP en eksponentiel vækst af problemstørrelsen, så for at løse det på en effektiv måde bør enten en ikke-optimal eller en branch and bound metode benyttes.

4.1.4 Planlægning

En helt anden type af problemer er planlægningsproblemer. I disse går det for agenten ud på at udvælge en serie af operationer, der vil føre systemet fra startsituationen til den ønskede situation ud fra agentens forhåndsviden om verden.

Hele verden er i et planlægningssystem beskrevet ud fra:

Objekter
: De ting, der kan manipuleres så de på en eller anden måde bliver ændret. De kan enten flyttes (fysisk) eller udsættes for andre handlinger, der ændrer deres forskellige egenskaber.
Et eksempel på et objekt er en klods.

Udsagn
: Fortæller noget om objekterne i verden, deres egenskaber og forhold til hinanden. Udsagnene benyttes dels til at beskrive den totale verdenstilstand og dels til at undersøge om forskellige operationer er mulige for objektet eller andre objekter, der påvirker det.
Et eksempel på et udsagn er "klodsen ligger på bordet".

Tilstande
: Verden i et planlægningssystem beskrives ud fra den aktuelle tilstand. Ud fra de gældende udsagn om verden kan dens tilstand beskrives. Et mål i systemet kan være at opnå en given tilstand, hvor de for målet givne udsagn alle er opfyldt.
Et eksempel på en tilstand er "klodsen ligger på gulvet OG klodsen er på bordet". Det er her vigtigt at lægge mærke til *og* relationen mellem de to udsagn, ligesom det er værd at notere sig, at såfremt der kun findes en enkelt klods er dette udsagn ikke muligt. Et planlægningssystem skal derfor kunne afgøre om agentens mål er mulige, så agenten eventuelt kan frasortere dem.

Operationer
: Skiftet mellem de forskellige tilstande, verden kan befinde sig i, sker ved udførelse af forskellige operationer. En operation kan i bund og grund beskrives som "noget, der påvirker verden", men følgende gør sig gældende for en definitionen af en operation: For det første har hver operation en liste af udsagn, der skal være opfyldt for at operationen er mulig, dernæst er der en liste af udsagn, der bliver gjort ugyldige og fjernet som følge af operationen. Og endelig er der en liste af udsagn, der opstår på grund af operationen. Et eksempel på en operation kunne være "flyt klodsen fra bordet til gulvet". Forudsætningen for dette er, at "klodsen ligger på bordet". Udsagnet "klodsen ligger på bordet" fjernes, men tilgengæld tilføjes "klodsen ligger på gulvet".

Et klassisk kunstig intelligens eksempel på et planlægningssystem er den i [Østerby] beskrevne blokverden, hvor målet er at få bygget en sorteret stabel af klodser ud fra de givne operationer. Der findes i litteraturen, blandt andet i [Østerby], metoder til at løse denne type af problemer, blandt andet ved at opstille et søgetræ for udførelsen af de forskellige operationer og gennemsøge dette. Eventuelt kan man lave en heuristisk søgning for at mindske antallet af undersøgte knuder i søgetræet.

4.1.5 Andre systemer

Udover ovenstående systemer findes der andre metoder til problemløsning, blandt andet de regelbaserede produktions- og ekspertsystemer. Brugen af disse systemer falder dog udenfor dette projekts opgave, idet de autonome agenters mål er så håndgribelige, at de beskrevne metoder vil kunne anvise en måde, hvorpå målene kan opfyldes.

Det bør dog nævnes, at såfremt det på et senere tidspunkt findes ønskeligt at indbygge et større niveau af beslutningstagen for agenterne, kan disse systemer benyttes til at analysere den simulerede verden og hjælpe agenterne med at tage beslutninger.

Kapitel 5

Behaviours

Dette kapitel vil gennemgå de forskellige simple adfærdsmønstre, der benyttes til at simulere de autonome agenter. Efter en gennemgang af de enkelte typer af adfærd vil der være en beskrivelse af, hvordan de kan kombineres til at give agenterne en mere kompleks adfærd, hvor de ikke blot blindt stoler på en enkelt adfærd, men laver en vægtning mellem de forskellige resultater.

5.1 Simple Behaviours

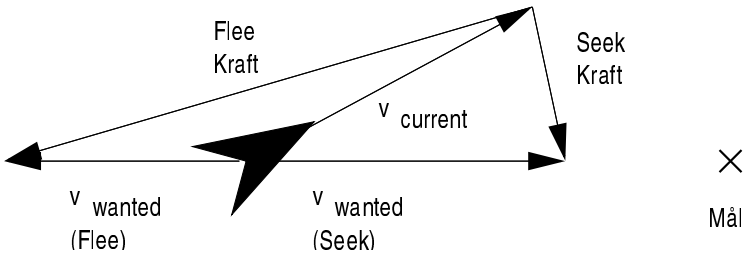
Dette afsnit giver en gennemgang af de forskellige Behaviours, der bliver benyttet i projektet. Disse er oprindeligt taget fra Craig Reynolds' *Steering Behaviors For Autonomous Characters*, [Reynolds], men nogle af dem er ændret lidt for at give en bedre adfærd.

5.1.1 Seek

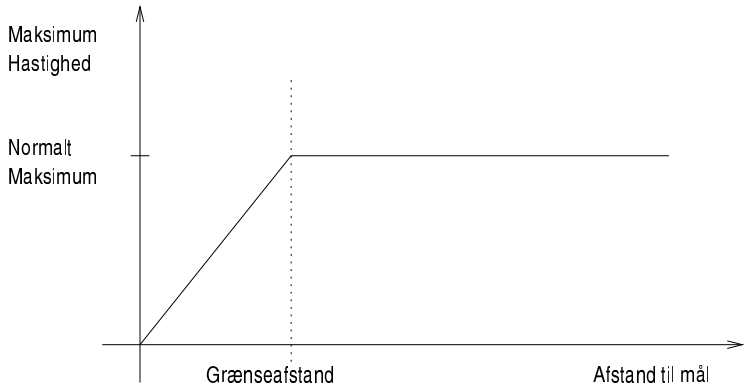
Agentens mål i Seek er at bevæge sig hen imod et bestemt punkt i verden. For at opnå dette vil agenten forsøge at ændre sin hastighedsvektor, så den peger direkte hen mod agentens mål.

På Figur 5.1 ses, hvordan agentens nuværende og ønskede hastighedsvektorer er. I hvert tidsskridt vil agenten påvirke sin hastighed med en kraft bestemt som $v_{wanted} - v_{current}$. Idet den maksimale kraftpåvirkning per tidsskridt er begrænset af agentens egenskaber vil der kunne forekomme situationer, hvor agenten "skyder over målet". Idet Seek vil forsøge at opnå så stor hastighed som muligt vil agenten, når den rammer ved siden af målet, have en stor hastighed, hvilket kan resultere i, at den først sent får drejet rundt og kører tilbage mod målet. Er man rigtig uheldig vil der opstå en situation, hvor agenten aldrig rammer målet, men blot kredser om det.

Det bør bemærkes, at Seek ikke er det samme som en tyngdekraft hen imod målet. Hvis det var tilfældet havde kraftpåvirkningen været rettet direkte fra agenten mod målet, samme retning som v_{wanted}.



Figur 5.1: Seek og Flee



Figur 5.2: Arrival hastighed

5.1.2 Flee

Flee minder meget om Seek, men i stedet for at ændre agentens hastighed mod målet ændres hastigheden, så den peger væk derfra. Dette princip er vist på Figur 5.1.

5.1.3 Arrival

Som en udvidelse til Seek findes Arrival. Når en agent ønsker at ankomme til sit mål, er det ikke nok blot at ramme det og fortsætte på den anden side. Agenten skal stoppe når den når til målet. Det betyder, at agenten skal bremse på en sådan måde, at den både kommer frem til målet og kan stoppe der.

Dette implementeres ved at gøre agentens maksimumhastighed afhængig af afstanden til målet. Hvis agenten er mere end en vis afstand fra målet er maksimumhastigheden den normale, hvis den er tættere på sænkes maksimumhastigheden. Den simpleste måde at gøre dette på er at gøre hastigheden lineært afhængig af afstanden fra målet som vist på Figur 5.2. Alternativt kunne man lave en ikke lineær funktion for den maksimale hastighed, hvilket ville svare til, at agenten bremser med en ikke konstant kraft. En begrundelse for dette kunne være, at jo hurtigere agenten bevæger sig, jo dårligere bremser den, så den er derfor nødt til at bremse forsigtigt indtil den når under en bestemt hastighed. En analyse af de fysiske forhold for agenten og verden, med henblik på blandt andet friktionskoefficienter, ville give en meget realistisk implementering af agentens egenskaber. En sådan analyse falder dog uden for rammerne af dette projekt.

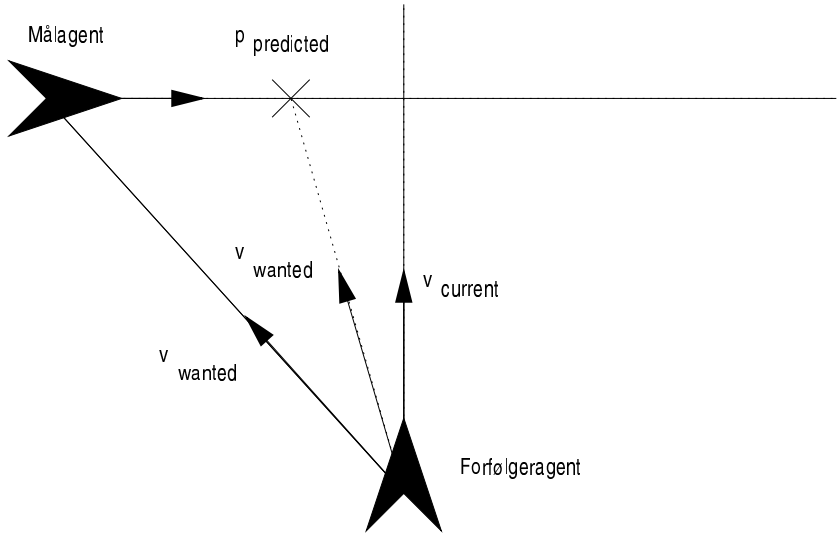
5.1.4 Pursuit

Der er en del forskellige måder at simulere forfølgelse på. På Figur 5.3 er der vist to forskellige. Den første er at lave en simpel Seek hen imod målets nuværende position. Den anden er at benytte målets nuværende hastighed til at give et gæt på, hvor målet er et stykke tid ude i fremtiden, positionen betegnet $p_{predicted}$.

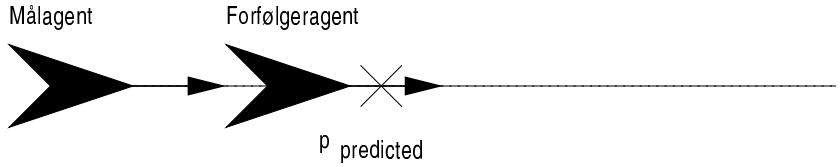
Sidstnævnte metode involverer et gæt på, hvor målet er efter "et stykke tid". Dette lægger op til en del forskellige måder at beregne det punkt, der skal søges hen imod. Nedenstående liste giver et par af de forskellige muligheder:

lineær interpolation
: Det antages, at forfølgeren kender målets (og sin egen) nuværende hastighed og position. Ud fra dette beregnes afstanden mellem forfølger og mål samt den tid, det ville tage forfølgeren at bevæge sig i lige linie hen til målet. Denne tid benyttes som et estimat for, hvornår forfølgeren når målet, og ud fra målets hastighed beregnes den position, målet forventes at være i til denne tid. Forfølgeren søger derefter hen imod det nye punkt.

vurdering af relativ position og hastighed
: Formålet med dette er at undgå nogle af de problemer, der opstår med den ovenfor beskrevne metode. Et eksempel på dette er vist på Figur 5.4, hvor $p_{predicted}$ ligger foran forfølgeren på en (nogenlunde) ret linie. Når forfølgeren laver Seek efter dette punkt vil den forsøge at bevæge sig så hurtigt hen imod $p_{predicted}$



Figur 5.3: Pursuit



Figur 5.4: Pursuit

som muligt. Hvis de to agenter har samme maksimal hastighed vil forfølgeren beholde sin relative position foran målet.
Som en udvidelse til den lineære interpolation kan man indbygge et check for, hvordan forfølgeren og målet bevæger sig i forhold til hinanden. Den ovenfor beskrevne situation ville med et sådant check kunne undgås idet forfølgeren ikke blindt vil køre med fuld hastighed, men vil bremse og vente på målet.
Udover at vurdere den relative position kan man også ændre på valget af prediktionstiden. En god variant ville være at benytte forfølgerens og målets relative hastigheder til at se, om tiden skal gøres længere eller kortere end ovenstående.

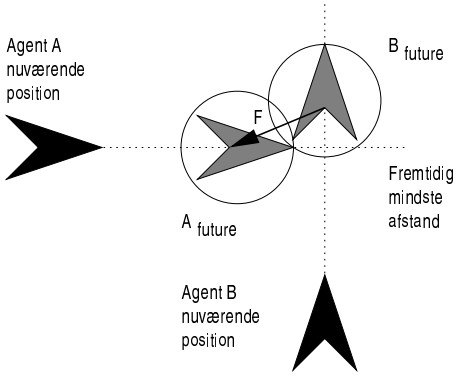
5.1.5 Evasion

Evasion er det modsatte af Pursuit på samme måde som Flee er det modsatte af Seek. Her vil den undvigende agent lave samme type forudsigelse som beskrevet under Pursuit, men vil lave Flee væk fra det endelige punkt i stedet for Seek hen imod det.

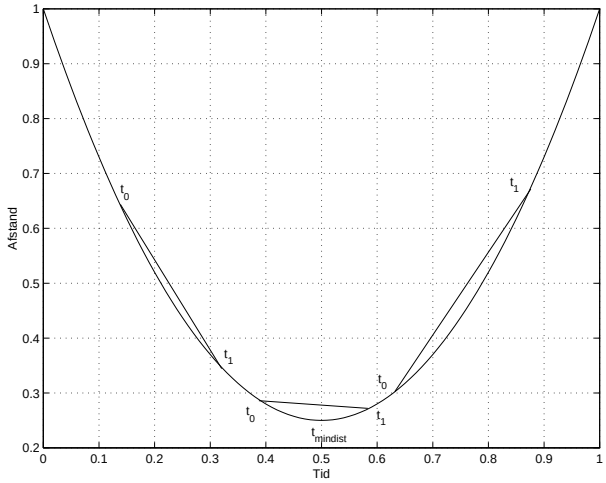
5.1.6 Avoidance

Når to eller flere agenter bevæger sig gennem verden uden at forfølge eller (forsøge at) undslippe hinanden er der en risiko for uønskede kollisioner. Disse undgås ved at lade agenterne have en Behaviour, der forudser kollisioner og give agenter involveret i potentielle kollisioner en kraftpåvirkning, der styrer dem væk fra hinanden.

På Figur 5.5 er vist en situation, hvor to agenter kan forudse, at de vil passere tæt på hinanden. Det tidspunkt, hvor agenterne er tættest på hinanden kan bestemmes ud fra afstanden mellem agenterne som funktion af tiden. Det kan vises, [Eising], at dette bliver et andengradspolynomie, der kan løses analytisk. Yderligere er polynomiets andenordens koefficient positiv, hvilket gør det let at bestemme tidspunktet $t_{mindist}$, hvor agenterne er tættest på hinanden.



Figur 5.5: Avoidance



Figur 5.6: Tidsafhængig afstandsfunktion

Det bemærkes, at idet tiden $t_0 = 0$ er tidspunktet, hvor agenterne begynder at forudsige en eventuel kollision vil en negativ tid, $t_{mindist} < 0$ betyde, at agenterne bevæger sig væk fra hinanden og den mindste afstand mellem dem således er til $t_{mindist} = 0$.

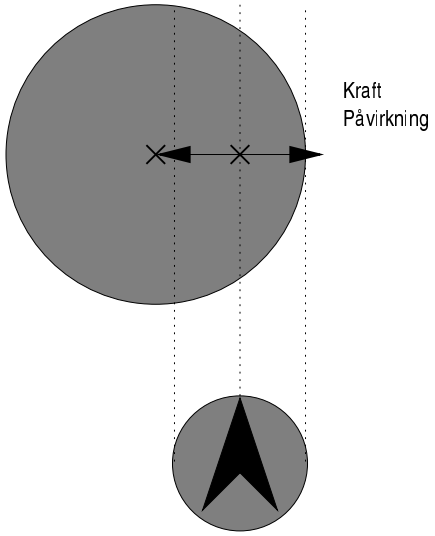
Når tiden, og dermed den mindste afstand mellem agenterne, er fundet kan det undersøges, om der egentlig er tale om en kollision eller ej. Hvis afstanden er mindre end summen af radierne af de to agenter vil der være en kollision og agenterne skal styre udenom hinanden.

For at finde den kraft hver agent skal påvirkes af beregnes forskellen mellem de to agenter, $F = A_{future} - B_{future}$, og Agent A og B påføres henholdsvis F og $-F$, dog skaleret så den bliver så stor som agentens maksimale kraftpåvirkning tillader. Dette vil på Figur 5.5 tvinge Agent A til højre samtidig med at den bremser, og Agent B accelererer og drejer til højre. Herved får de to agenter begge flyttet sig væk fra kollisionen.

En ting, der med fordel kan implementeres i denne Behaviour, er en grænse for, hvor langt ud i fremtiden agenterne forudser kollisioner. Med mindre alle agenterne bevæger sig ligeud uden at dreje vil det kun være kollisioner inden for en kort tidshorisont, der er interessante for agenterne. Hvis der lægges en øvre grænse, t_1, for hvor langt ud i fremtiden en agent undersøger kollisioner, vil der være følgende situationer, når kollisionstiden bestemmes:

1. $t_0 < t_1 \leq t_{mindist}$: her vil den mindste afstand være til tiden $t = t_1$.
2. $t_0 \leq t_{mindist} \leq t_1$: her vil den mindste afstand være til tiden $t = t_{mindist}$.
3. $t_{mindist} \leq t_0 < t_1$: her vil den mindste afstand være til tiden $t = t_0$.

Hvordan disse tre muligheder ligger i forhold til afstandsfunktionen er skitseret på Figur 5.6.



Figur 5.7: Reynolds' Obstacle Avoidance

5.1.7 Obstacle Avoidance

Når der indføres statiske objekter i verden vil agenterne få brug for at kunne styre uden om disse. I [Reynolds] beskrives en måde, hvor man ved at benytte omskrevne cirkler som bounding circles for både agenter og objekter kan lave et hurtigt check for mulige kollisioner. Formålet med Obstacle Avoidance er ikke at lave et perfekt collision detection system, men at lave en Behaviour, der får agenterne til at undgå objekterne.

På Figur 5.7 og Figur 5.8 er der vist to forskellige metoder til beregning af agentens kraftpåvirkning ved en kollision mellem agenten og et objekt repræsenteret ved en cirkel. Ved begge måder undersøges der for en potentiel kollision ved at finde afstanden mellem cirklens centrum og en linie i forlængelse af agentens hastighedsvektor. Hvis afstanden fra cirklens centrum til denne linie er mindre end summen af radierne for cirklen og agentens omskrevne cirkel er der en mulig kollision og agenten bør styre udenom. Yderligere checkes det, hvor langt fra agenten kollisionen sker. Hvis denne afstand er større end en valgt grænse, vil der ses bort fra kollisionen, idet den ikke udgør et umiddelbart problem for agenten.

Forskellen på de to måder ligger i, hvordan den endelige kraftpåvirkning bestemmes. Reynolds foreslår, at man benytter en vektor vinkelret på agentens hastighedsvektor rettet væk fra cirklens centrum. Størrelsen af kraften er afstanden fra cirklens centrum til linien gennem agentens hastighedsvektor.

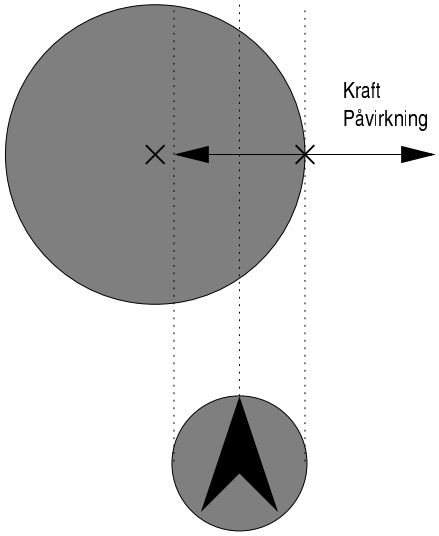
Idet denne kraftpåvirkning ikke afhænger af det egentlige kollisionspunkt, det vil sige kanten af objektet, vil der opstå situationer, hvor en agent peger så tæt ind mod objektets centrum, at kraftpåvirkningen er for lille til at styre agenten udenom objektet.

På Figur 5.8 er der vist en alternativ måde at bestemme kraftpåvirkningen. Her vælges den samme retning som i Reynolds' metode, men kraftens størrelse vælges som afstanden fra objektets kant til projektionen af agentens modsatte kant, i dette tilfælde den venstre side af agenten. Herved bliver kraftens størrelse afhængig af både agentens og objektets størrelser samt, hvor langt "inde i" objektet kollisionen sker.

Endnu en måde at forbedre denne Behaviour på ville være at beregne kollisionen mellem objektets og agentens omskrevne cirkler, finde agentens position på det tidspunkt og foretage styring og/eller nedbremsning ud fra agentens position relativt til objektets centrum. Specielt nedbremsningen vil være en god forbedring, idet den kan hjælpe til med at undgå situationer, hvor agentens fremadrettede hastighed er for stor til at den kan komme udenom objektet.

5.1.8 Wander

Wander dækker over tilfældig vandring. Denne Behaviour kan bruges til agenter, der ikke har noget umiddelbart mål, men som ikke ønsker at stå stille eller bevæge sig i en lige linie. Ud fra agentens definition er der umiddelbart



Figur 5.8: Alternativ Obstacle Avoidance

tre egenskaber, der kan gøres tilfældige: position, hastighed og kraftpåvirkning. Der er dog også en sidste mulighed, nemlig at lave en tilfældig *ændring* af den kraft, der påvirker agenten. Fordelen ved dette er, at kraftpåvirkningen ikke vil springe synderlig meget og som følge heraf vil agenten bevæge sig i bløde kurver og ikke i ryk.

I praksis gøres dette ved at generere en tilfældig vektor af passende kort længde sammenlignet med agentens maksimale kraftpåvirkning og lægge denne vektor til den aktuelle kraftvektor. Herefter forgår opdateringen som normalt ved at kraften påvirker hastigheden, der efterfølgende påvirker positionen.

5.2 Kombination af Behaviours

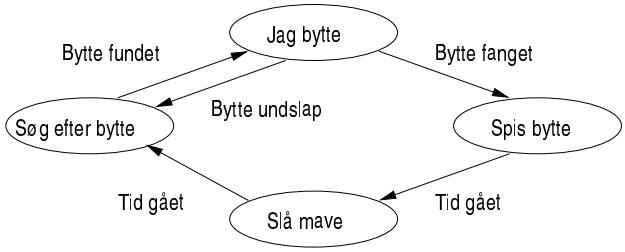
For at give en agent et mere nuanceret adfærdsmønster er det muligt at kombinere flere Behaviours. Der er basalt set to forskellige måder at gøre dette på:

vægtning
Hver enkelt Behaviour tillægges en vægt. Når agenten skal beslutte sig for om den skal ændre retning beregnes hver enkelt Behaviours kraftpåvirkning og agenten laver ud fra disse og deres vægte den endelige kraft.

Eksempel 5.1: Vægtning af Behaviours
Et eksempel på anvendelsen af vægtning er et rovdyr, der både vil løbe udenom skovens træer og komme tættere på det byttedyr, den jager. Rovdyret vil forsøge at finde vægte for Obstacle Avoidance og Pursuit i håbet om, at den derved kan indhente byttet uden at løbe ind i træer og lignende forhindringer.

prioritering
I stedet for at lave en vægtning kan en agent vælge at undersøge sine Behaviours i en bestemt rækkefølge og benytte den første Behaviour, der resulterer i en kraftpåvirkning. Dette kræver, at agenten på forhånd har lavet en prioritering af, hvad der er vigtigst at vælge, når den skal bevæge sig.

Eksempel 5.2: Prioritering af Behaviours
Hvis rovdyret fra Eksempel 5.1 ønsker at være sikker på at undgå træer og andre forhindringer, ville det i stedet for vægtning benytte en prioritering af Obstacle Avoidance og Pursuit: hvis Obstacle Avoidance forudser en kollision, benyttes dette til at navigere (og Pursuit bliver aldrig beregnet), hvis der ikke er nogen kollisioner, benyttes Pursuit til at styre hen mod byttedyret.



Figur 5.9: Rovdyr adfærdscyklus

5.3 Skift af Behaviours

Ovenstående vægtning og prioritering af en agents forskellige Behaviours kan som nævnt benyttes til at give agenten et nuanceret adfærdsmønster. Men i mange situationer vil en agents adfærd være styret af flere eksterne faktorer end man umiddelbart kan bygge ind i vægtning og prioritering.

For at give agenten mulighed for at skifte fra et adfærdsmønster til et andet kan der bygges en tilstandsmaskine op rundt om de forskellige mønstre agenten bør have. Denne tilstandsmaskine styres af både agentens egne handlinger og eksterne faktorer fra omgivelserne.

Eksempel 5.3: Rovdyr adfærdscyklus
Den helt simple måde at implementere et rovdyr på ville være kun at have to tilstande: "Jag bytte" og "Søg efter bytte". Dette er dog ikke en særlig nuanceret beskrivelse af et rovdyr, så derfor indføres der nogle flere tilstande og tidsafhængige skift, der kan variere fra rovdyr til rovdyr.

På Figur 5.9 ses en tilstandsmaskine for et (primitivt) rovdyr, hvis eneste formål her i livet er at jage, fange og spise byttedyr. Som udgangspunkt vil rovdyret bevæge sig tilfældigt (Wander) og søge efter bytte (eventuelt med Avoidance i forhold til andre rovdyr). Når det har fået øje på et byttedyr, vil rovdyret skifte over til at forfølge (Pursuit) byttedyret, indtil det enten er blevet fanget eller er sluppet væk. Hvis et byttedyr er blevet fanget, bruger rovdyret først noget tid på at spise det og derefter noget tid på at slappe af, "slå mave". Når rovdyret har slået mave skifter det over i "Søg efter bytte" tilstanden igen og således bliver det ved. Disse tidsforsinkelser er valgt for at give rovdyret lidt personlighed.

Kapitel 6

Prototype

En del af projektet har været at implementere en prototype af det endelige system. Det primære formål med prototypen var at undersøge nogle af de teknikker, der skulle bruges i det endelige system. Denne prototype bestod primært af en implementering af Reynolds' Steering Behaviours for en simpel agent.

6.1 Implementering

Prototypen er skrevet i ANSI C og er splittet op i et antal forskellige moduler, der hver især håndterer de forskellige dele af det samlede system. En listning af kildekoden findes ikke i denne rapport, men er udgivet som et separat hæfte.

6.1.1 Moduler

I prototypen findes der et antal forskellige moduler, hvoraf de vigtigste er beskrevet i detaljer nedenfor. De moduler, der ikke bliver beskrevet nærmere, håndterer "trivielle" opgaver såsom vektormatematik og indlæsning samt fremvisning af agenternes grafiske repræsentation.

simplevehicle

Dette modul varetager de datastrukturer og funktioner, der beskriver agentens bevægelsesdel, et Simple Vehicle. I forhold til Afsnit 3.2.3 er der lavet et par udvidelser for at kunne beskrive en agents Behaviours og den grafiske repræsentation af et SV.

Vigtigst at bemærke er, at hvert SV har mulighed for at have en målposition, `target_position`, og en målagent, `target_vehicle`. Grunden til dette er, at for nogle Behaviours har agenten brug for et fast referencepunkt og for andre har agenten brug for at følge en anden agent. Det optimale ville have været at implementere et generelt `target`, der afhængigt af agentens Behaviour enten er en position eller et andet Simple Vehicle. I prototypen er valget dog faldet på at have den programmeringsmæssigt simplere løsning med begge felterne.

Den komplette datastruktur for et Simple Vehicle er:

```
/*******************************
 * struct SimpleVehicle
 * Definition of a simple vehicle. The
 * struct has the following elements:
 * position : vehicle's position
 * velocity : vehicle's velocity
 * force : current force on the vehicle
 * mass : vehicle's mass
 * max_force : maximum length of the force vector
 * max_speed : maximum length of the speed vector
 * orientation : angle between vector (1,0) and velocity
 * color[3] : R, G and B components of vehicle's color
 * behavior : used to specifiy how the vehicle behaves
```

```
 * target_position : a point that is targeted
 * targe_vehicle : a vehicle that is targeted
 * display_list : GL display list used for drawing vehicle
 * bounding_list : bounding volume's DL display list
 ******************************/
struct SimpleVehicle {
  VectorPtr position;
  VectorPtr velocity;
  VectorPtr force;
  double mass;
  double max_force;
  double max_speed;
  double orientation;
  GLfloat color[3];
  int behavior;
  VectorPtr target_position;
  struct SimpleVehicle* target_vehicle;
  GLuint display_list,bounding_list;
};
```

behavior [1]

I Afsnit 5.1 er der gennemgået, hvordan en række Behaviours kan benyttes til at give agenten en passende kraftpåvirkning. `behavior` modulet implementerer funktioner til disse Behaviours samt en wrapper funktion, der for hver enkelt agent kalder den rigtige kombination af underliggende Behaviour funktioner.

statemachine

I prototypen benyttes dette modul til at foretage skift mellem agenternes adfærdsmønstre. Der er implementeret fire forskellige tilstandsmaskiner:

- `sailor` beskriver det på Figur 8.2 viste adfærdsmønster for en matros
- `passenger` beskriver det på Figur 8.3 viste adfærdsmønster for en passager

De to sidste tilstandsmaskiner er nærmere beskrevet i Afsnit 8.1.

Fordelen ved Grosbergs måde, [Grosberg], at implementere tilstandsmaskiner er, at tilstandsskift sker ved brug af funktionspointere, hvilket mindsker antallet af checks når det undersøges, hvilken tilstand agenten befinder sig i. Godt nok er dette check ikke synderligt dyrt, men at fjerne tilstandschecket giver kortere og (i forfatterens mening) pænere programmer.

Et alternativ ville være at implementere det hele som en tabel over tilstandsskift som funktion af den nuværende tilstand og input, men denne metode ville give en del arbejde, hver gang der skulle tilføjes nye tilstande eller inputtyper, idet hele tabellen så skulle opdateres.

sphere [2]

Som nævnt i Afsnit 5.1.7 benyttes cirkler til at repræsentere forhindringer i det Environment agenterne bevæger sig i. Dette modul varetager repræsentationen af objekterne ved brug af en simpel liste af objekter/cirkler beskrevet ved deres position samt radius. Yderligere indeholder modulet funktionen `sphereAvoidance`, der for et givent Simple Vehicle returnerer den kraftpåvirkning, der skal til for at styre udenom det nærmeste objekt, såfremt dette Simple Vehicle og objektet kolliderer indenfor den valgte tidsramme.

rendering

Til trods for, at en komplet gennemgang af renderingsmodulet er udeladt her, bør det dog alligevel nævnes. Som repræsentation af agenterne benyttes en lille 3D model af en pil, der viser agentens position og retning. Modellen er gemt som en VRML model og bliver i prototypen vist ved brug af 3D API'et OpenGL. En af grundene til denne kombination er, at både VRML og OpenGL er vidt benyttede standarder. VRML er godt understøttet af 3D modelleringsapplikationer (til at lave agenternes modeller) og OpenGL er understøttet af en lang række forskellige kombinationer af operativsystemer og 3D grafik hardware.

Visualiseringen af agenterne sker med en simpel model: en ekstruderet pil, der peger i den retning agenten bevæger sig. Agentens personlige sfære er repræsenteret som en cirkel ved agentens "fødder". Idet rendering af en agents udseende sker ud fra en VRML fil vil det ikke være svært at ændre modellen, så der benyttes en mindre abstrakt model.

[1] Det bør bemærkes, at der ved udarbejdelsen af prototypen ikke blev taget højde for, at adfærd på engelsk hedder Behaviour, mens det på amerikansk hedder Behavior. I rapporten er de engelske termer benyttet, men prototypen blev lavet ud fra [Reynolds], så der benyttes de amerikanske.

[2] Modulnavnet `sphere` er valgt i stedet for `circle` da systemet forventes udvidet til at operere i tre dimensioner.

I forbindelse med implementeringen af funktionerne til simuleringen af det synkende skib blev der implementeret en række nye moduler til prototypen. Disse moduler bygger ovenpå de tidligere nævnte og giver tilsammen mulighed for at simulere matroser og passagerer på et synkende skib. Se Afsnit 8.1 for en komplet beskrivelse af simuleringens formål og bestanddele.

agent
De ovenstående moduler har primært beskæftiget sig med at undersøge de forskellige Behaviours enkeltvis, eventuelt i en prioriteret kombination. For at kunne lave mere komplicerede simuleringer har det dog været nødvendigt at gå et skridt videre og lave en komplet agent bestående af en tilstandsmaskine og et `SimpleVehicle`.
`agent` modulet implementerer funktioner til at styre de simulerede agenter, hvilket blandt andet involverer
- collision detection mellem agenter
- valg af adfærd udfra agentens tilstand samt de umiddelbare omgivelser

statemachine
Det ovenfor nævnte `statemachine` modul er blevet udvidet med tilstande og tilstandsskift for en matros, `sailor`, og en passager, `passenger`.
Det er værd at bemærke, at `statemachine` modulet udelukkende håndterer skiftene mellem de forskellige tilstande, hvorimod agenternes adfærd i de enkelte tilstande varetages af `agent` modulet.

ship
Det simulerede skib er en afgrænset verden, der basalt set består af følgende ting:
- en liste af vægge
- en liste af vandområder
- en liste af redningsflåder
- en liste af matroser
- en liste af passagerer

Udover at definere en datastruktur, der indeholder modulet også de funktioner, der foretager et tidsskridt i simuleringen og de hjælpefunktioner, agenterne bruger til at "se" deres omgivelser.

obstacle
Dette modul svarer til `sphere` modulet. Dog håndterer `obstacle` konvekse polygoner og ikke blot cirkler. Denne type objektet benyttes til at repræsentere skibets vægge og redningsflåder samt det vand, der omgiver skibet.

pathfind
I forbindelse med matrosernes adfærd var der brug for et navigationsmodul, der gav matroserne en vej fra deres nuværende position til et givent punkt på skibet. Til dette formål blev `pathfind` modulet implementeret. Modulets ene hovedopgave er at analysere de objekter, der er defineret på et `ship`, og ud fra disse lave et kort over, hvor på skibet matrosen kan bevæge sig. Den anden hovedopgave er at finde den korteste vej mellem to punkter på skibet og returnere denne vej som en målliste for den matros, der ønsker at finde vej.
Den automatiske generering af et kort over skibet blev lavet ved at forskyde hvert punkt i polygonerne i en retning bestemt af sidenormalerne for punktets to tilstødende sider. Idet agenterne er repræsenteret med cirkler giver denne metode ikke et optimalt kort, da der vil være dele af den nye polygon omkring hjørnepunkterne, der optimalt burde være cirkelbuer. Efter polygonerne på denne måde er blevet udvidet tegnes de ved hjælp af `gd` grafik biblioteket, se [Boutell] for en komplet beskrivelse, som et pixeleret kort over skibet.
På det genererede kort benyttes herefter A* algoritmen til at finde den korteste vej mellem de givne start- og slutpunkter. Implementeringen af A* er sket ud fra [Jönsson], dog ikke med den samme optimering af den benyttede datastruktur til repræsentation af hver knude i grafen. Udover A* er også Dijkstras algoritme implementeret. Sidstnævnte benyttes dog ikke i selve projektet, men fungerede under udviklingen af A* som reference algoritme til at checke, at den korteste vej blev fundet.

6.1.2 Programstruktur

Hovedstrukturen i systemet er basalt set som følger:

```
/* While we have moving vehicles do the
 * main loop and update active vehicles
 */
```

```
while (moving) {
  /* Calculate the length of the current
   * timestep as the time it took to do
   * the last one.
   */
  time = system_gettime();
  timediff = time - last_time;
  last_time = time;

  moving = 0;
  for (i = 0; i < num_vehicles; i++) {
    /* Loop through all vehicles */
    if (vehicle_moving[i]) {
      /* For each vehicle that is still "alive" we
       * apply the vehicles behvaior and draws it.
       */
      vehicle_moving[i] = Behavior(vehicles[i],timediff);
      drawVehicle(vehicles[i]);

      /* Increase the count of moving vehicles */
      moving++;
    }
  }
}
```

For hvert tidsskridt måles, hvor lang tid det foregående har taget, og ud fra dette beregnes og påføres hver agents Behaviour bestemte kræfter. Systemet opererer her kun med et enkelt stopkriterie, nemlig at der stadig er agenter, der skal behandles. Når alle agenterne af deres Behaviour er markeret som inaktive stoppes simuleringen. Da der i prototypen ikke er mulighed for dynamisk at tilføje og fjerne agenter er det muligt at benytte et array, `vehicle_moving`, til at holde styr på, om en agent stadig er aktiv (moving) eller ej.

6.1.3 Problemer

Udarbejdelsen af prototypen er foregået som en kombination mellem at implementere et fungerende system og at undersøge muligheden for at bruge de Behaviours Reynolds beskriver. Dette har resulteret i en temmelig uoverskuelig programstruktur, og efterhånden som systemet er vokset, er der opstået en række komplikationer. Selvom disse kunne have været undgået ved at lave en komplet specifikation af prototypen på forhånd, blev dette fravalgt, fordi det i starten af projektet stod uklart, hvor meget Reynolds' Behaviours kunne bruges til i praksis. Som nævnt i indledningen var formålet med prototypen netop at undersøge disse.

Dette Afsnit omhandler de erfaringer, der er gjort i forbindelse med udarbejdelsen af prototypen, samt overvejelser om, hvordan de fremkomne problemer kan undgås i det endelige system.

Globale variable

Adskillige af systemets centrale dele er bygget op omkring globale variable, der bliver tilgået fra de enkelte moduler i det omfang, modulernes funktioner mener de har behov for det. Eksempelvis tilgår funktionerne til at lave Avoidance det globale array af `simplevehicle`s direkte frem for at få udleveret de nødvendige data fra `simplevehicle` modulet.

Denne måde blev primært valgt for at undgå at bruge udviklingstid på at lave et veldesignet interface gennem hvilket agenterne og deres Behaviours kunne tilgå de andre agenters informationer. Det vigtige i prototypen var, at agenterne havde informationerne, ikke hvordan de fik fat i dem.

I det endelige system skal dette være mere struktureret, så det bliver lettere at udskifte dele af systemet og bruge den samme kildekode til forskellige simuleringer.

Dynamisk opdatering

Det faktum, at hele listen af agenter er opbygget som et array af `simplevehicle`s, medfører også problemer i forbindelse med at tilføje og fjerne de enkelte agenter i løbet af en simulering. Det er naturligvis muligt at udvide

det eksisterende array, men dette er ikke umiddelbart ønskeligt da det koster relativt mange systemressourcer at tildele lagerplads til det udvidede array og kopiere de gamle data herover.

Tilsvarende vil det at fjerne en agent betyde, at der skal "klippes" et arrayelement ud. Alternativt kunne man implementere en liste over, hvilke arrayelementer der er aktive og så indsætte nye agenter på inaktive agenters pladser. En bivirkning ved dette er dog, at man risikerer at få en inkonsistens mellem de enkelte agenters numre/ID og de tidligere benyttede numre/ID i forbindelse med for eksempel Pursuit og Evasion. Implementeringen af `simplevehicle` indeholder en pointer til målagenten (`target_vehicle`), hvilket reelt set er endnu farligere end blot at gemme agentens nummer/ID. Det farlige ved en pointer er, at det ikke er muligt at checke om målagenten er den oprindeligt valgte.

Som en lille sidebemærkning bør det nævnes, at det man med en pointer rent faktisk risikerer at have en agent, der forfølger nogle data i et hukommelsesområde, der i forbindelse med fjernelsen af en agent er blevet frigivet og senere brugt til andre data. Dette fænomen, dangling pointers, har ikke direkte noget med den dynamiske opdatering at gøre, men er et generelt problem, der bør undgås i det omfang det er muligt.

For at få et ID-system til at fungere godt er det nødvendigt at implementere funktioner, der holder styr på, hvilke ID-numre der er i brug, hvilke der har været i brug, samt lave en opslagstabel, så man altid kan gå fra et ID-nummer til en agent og omvendt.

Obstacle Avoidance

Beregningen af Obstacle Avoidance Behaviour kraftpåvirkningen befinder sig *ikke* i `behavior` modulet, men i `sphere` modulet. Dette skyldes, at funktionerne til at checke for kollision mellem et enkelt Simple Vehicle og alle objekterne under implementeringen lagde op til en funktion, der blev kaldt med *et* `simplevehicle` og havde adgang til `sphere` modulets liste af objekter.

En god omskrivning af dette modul ville være at lade `sphere` modulet beregne, hvilket objekt agenten støder ind i og så returnere dette objekt. Herefter ville agenten selv kunne beregne sin kraftpåvirkning, eventuelt støttet af hjælpefunktioner i `sphere` modulet. Disse hjælpefunktioner kunne tænkes at være `sphereIntersection(...)`, `sphereNormalAtPoint(...)` og lignende.

Agentrepræsentation

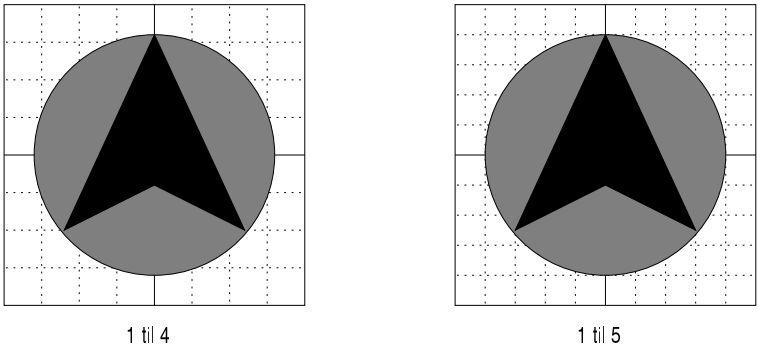
En af de fundamentale fejl i repræsentationen af en agent er, at det reelt set ikke er en agent, men et Simple Vehicle med en (og kun en) Behaviour. Det ville være mere korrekt at repræsentere en agent som en entitet, der bestod af et Simple Vehicle og en række Behaviours. Fordelen ved dette ville være at adskille beslutningstagning (Behaviours og tilstandsmaskiner) fra selve bevægelsesmaskineriet (Simple Vehicle).

Yderligere har prototypens agentrepræsentation den mangel, at det ikke er muligt at gemme og skifte mellem forskellige adfærdsmønstre for hver enkelt agent som beskrevet i Afsnit 5.3. Dette har i prototypen blandt andet resulteret i, at `statemachine` modulet ikke benyttes. `statemachine` modulet er i prototypen tomt, men parallelt med udarbejdelsen blev der implementeret tilstandsmaskiner, der repræsenterede høge og duer i stil med det i Eksempel 5.1 beskrevne rovdyr. Dette blev gjort for at undersøge den i [Grosberg] beskrevne metode til implementering af tilstandsmaskiner. Kildekoden til disse tilstandsmaskiner findes i `statemachine.c` og `statemachine.h`.

Behandling af agenter

Et sidste tankevækkende forhold er den rækkefølge agenterne bliver behandlet i. Der er ikke opstået egentlige problemer på grund af dette, blot interessante resultater.

I prototypen bliver hver enkelt agent helt færdigbehandlet før den næste agent undersøges. Det vil sige, at den første agent i listen undersøger sine Behaviours, vægter/prioriterer resultaterne *og* opdaterer sin position før den sidste agent i listen. Dette gør sig mest synligt, når agenterne laver Collision Avoidance. Betragtes de på cd'en vedlagte videosekvenser kan man se, at det ofte kun er en agent, der ser en potentiel kollision med en anden agent. Idet denne agent har set kollisionen ændrer den også sin hastighed og position, og når den anden agent i kollisionen undersøges ser den derfor ikke nogen problemer.



Figur 6.1: Kortinddeling

Automatisk generering af kort

I forbindelse med den automatiske generering af kortet over skibet er der en lille detalje, der er værd at have med. Det er som sådan ikke et problem, men falder derimod i kategorien "uafklarede spørgsmål".

Når kortet laves konverteres der fra punkter i et R^2 rum til et N^2 rum, da kortet repræsenteres som et todimensionelt array. Dette betyder, at det skal vælges, hvor stort et område hver pixel repræsenterer. Dette valg vil, kombineret med udvidelsen af polygonerne og indtegningen af dem, kunne resultere i "afrundingsfejl" som følge af konverteringen fra flydende tal til diskrete pixelværdier. Disse afrundingsfejl vil i værste tilfælde betyde, at to objekter gøres så meget større, at det ikke er muligt for agenten at passere mellem dem.

I de vedlagte simuleringer er der benyttet en 1 til 5 forstørrelse, så hver pixel kortet svarer til et $\frac{1}{5}$ gange $\frac{1}{5}$ område på skibet. Dette er gjort ud fra størrelsen af den personlige sfære agenterne har. Som beskrevet i Afsnit 3.2.4 er den personlige sfæres radius $80cm$, det vil sige $\frac{4}{5}m$, så ved at vælge 1 til 5 forstørrelse vil kanten af agenternes personlige sfærer tangere rækker og søjler på kortet. På Figur 6.1 er der vist 1 til 4 og 1 til 5 forstørrelserne samt, hvordan pixels på kortet ligger i forhold til en agent. Hvis en 1 til 4 forstørrelse var valgt, ville der være situationer, hvor agenten enten vil gå lidt gennem en væg eller holder en unødigt stor afstand til den.

Det optimale ville være at vælge denne inddeling ud fra viden om agentens bredde, men da dette i værste tilfælde ville resultere i en generering af kortet for hver enkelt agent i simuleringen risikerer man at anvende en stor del af den tilgængelige lagerplads på at gemme de forskellige kort. Alternativt kunne man lave et nyt kort, hver gang en agent spørger efter vej. Dette vil mindske den anvendte lagerplads, men vil koste beregningstid.

6.2 Resultater

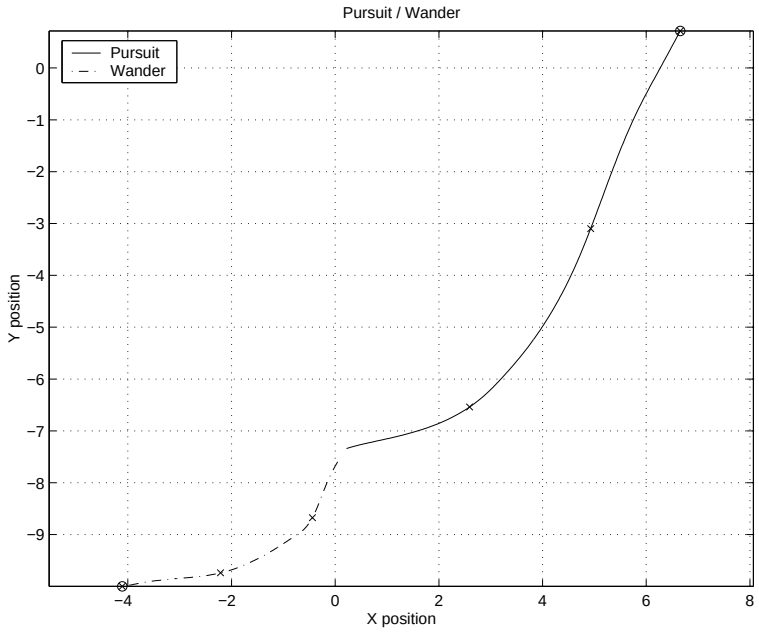
Til trods for de problemer, der har været i forbindelse med udbygningen af prototypen, har den dog tjent sit primære formål: at undersøge de i [Reynolds] beskrevne Behaviours.

Systemet har vist, at det er ganske enkelt at implementere Behaviours og at benytte dem til at styre autonome agenter rundt i en simpel verden. Implementeringen af agenter med en enkelt Behaviour har været let, idet beregningerne bag de enkelte agenter ikke involverer specielt kompliceret matematik.

Prototypen er også benyttet til at simulere matroser og passagerers opførsel på et synkende skib. Resultaterne fra disse simuleringer bliver behandlet i Afsnit 8.1, hvor hele simuleringen beskrives i detaljer.

6.2.1 Simple Behaviours

Alle de i Afsnit 5.1 beskrevne Behaviours er implementeret og virker efter hensigten. Derudover er der for Avoidance blevet indbygget en prioritering mellem Obstacle Avoidance og (agent) Avoidance. Hvis Obstacle Avoidance ikke forudser en kollision checkes der for kollisioner med andre agenter. Såfremt agenten er helt kollisionsfri benyttes Wander.



Figur 6.2: Pursuit / Wander

Testkørsler med specielt Seek har vist, at agenternes bevægelser og muligheder for at nå deres mål afhænger kraftigt af den maksimale kraftpåvirkning, `max_force`. I de første testkørsler blev der benyttet en for lille `max_force`, hvilket resulterede i, at agenterne aldrig nåede målet for deres Seek. Efter at have kørt en Seek simulering i nogen tid var det muligt at se, hvordan agenterne bevægede sig i baner rundt om målet, men aldrig kom helt tæt på.

Efter `max_force` blev (empirisk) tilpasset, så den fik en god værdi kørte de simpleste Behaviours, Seek, Flee og Arrival, som ønsket. De øvrige Behaviours har haft forskellige kuriositeter, der vil blive beskrevet nedenfor. På de tilhørende figurer er der et par ting, der gør sig gældende:

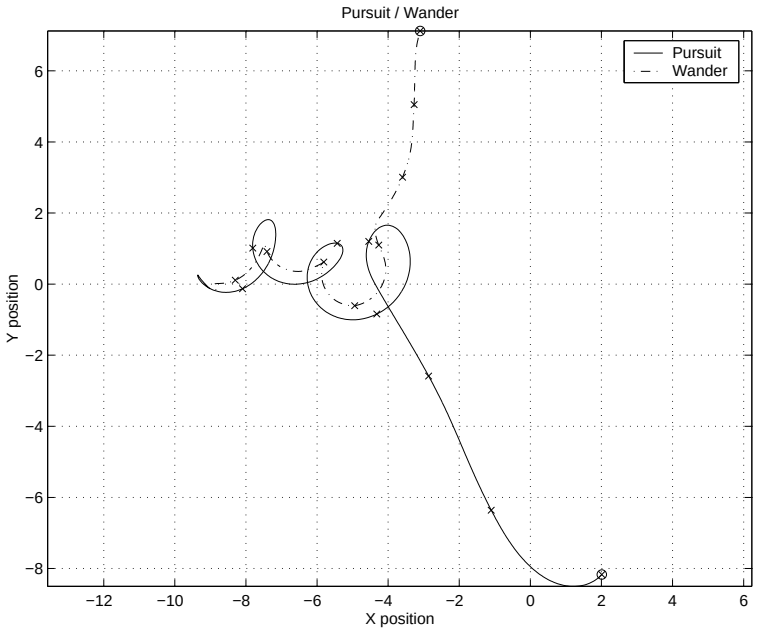
- agenternes startposition er markeret med et 'O'.
- de indtegnede baner er agentens centrum.
- figurer med to eller flere agenter har 'X' markeringer langs agenternes baner. Disse markeringer er tidssynkroniseret mellem agenterne, således at det er muligt at se, hvordan de to agenter har bevæget sig i forhold til hinanden.
- forhindringer er markeret med to cirkler. Den inderste er selve objektet, den yderste har en radius på $objekt_{radius} + agent_{radius}$.
- hvis en agent når kanten af det simulerede område udsættes den for "billard reglen" og får sin hastighedsvektor spejlet om den kant den har nået.

Pursuit

Virker ganske som beskrevet i Afsnit 5.1.4, inklusive de deri gennemgåede småfejl når målagenten befinder sig bag forfølgeren. I testkørslerne har det vist sig, at når forfølgeren kommer forfra eller fra siden af målagenten virker Pursuit godt. Målet bliver fanget og forfølgeren har et "angrebsforløb", der synliggør den benyttede forudsigelse.

Eksempel 6.1: Pursuit På Figur 6.2 ses en situation, hvor en agent forfølger en anden, tilfældigt vandrende agent. Det lykkes forfølgeren at fange sit mål uden nogen større problemer og ud fra banen kan det ses, hvordan forfølgeren forudser målets fremtidige position.

Eksempel 6.2: Forvirret Pursuit Som nævnt i Afsnit 5.1.4 risikerer en forfølgende agent at ende i en situation, hvor den normale forudsigelse af målets bevægelse giver problemer. På Figur 6.2 ses dette fænomen.



Figur 6.3: Pursuit / Wander

Forfølgeren bevæger sig fint henimod målet, men et pludseligt "knæk" i målets bevægelser får forfølgeren til at ramme ved siden af målet og de efterfølgende cirkelbevægelser forvirrer forfølgeren så meget, at der går længere tid, før den fanger målet.

Evasion

Evasion gemmer ikke på de helt store overraskelser. Den flygtende agent drejer rundt, så den bevæger sig i en lige linie væk fra den forfølgende agent. Idet agenten relativt hurtigt vil kunne dreje væk fra sin forfølger, vil der kun være tre muligheder for at en Pursuit agent vil kunne fange en Evasion agent:

- Pursuit agent hurtigst. Såfremt de to agenter ikke bliver påvirket af andet end deres Pursuit og Evasion Behaviours vil den hurtigere agent med tiden indhente den langsomme.
- De to agenter starter tæt på hinanden. Hvis startafstanden er lille og de to agenter bevæger sig mod hinanden vil det kunne lykkes for en langsommere Pursuit agent at fange en hurtigere Evasion agent.
- Forhindringer. Hvis Evasion agenten skal bevæge sig rundt om forhindringer eller andre agenter vil den kunne blive indhentet.

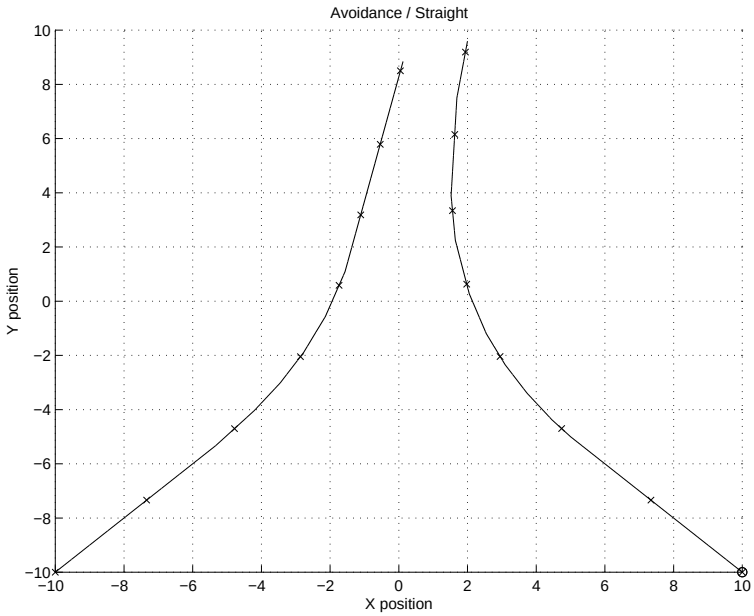
Avoidance

Figur 6.4 viser et eksempel på to agenter, der styrer mod samme punkt men opdager, at de er ved at kollidere. Agenterne er i eksemplet startet i henholdsvis $(-10, -10)$ og $(10, -10)$ og deres hastighedsvektorer har samme størrelse og er rettet ind mod $(0, 0)$. Det ses, at til trods for dette bevæger de sig ikke synkront ind mod midten, men "hælder" mod højre. Ideelt set burde de begge foretage en korrigerende styring, men idet hele programstrukturen færdigbehandler agenterne hver for sig vil kun en af agenterne se en potentiel kollision i samme tidsskridt.

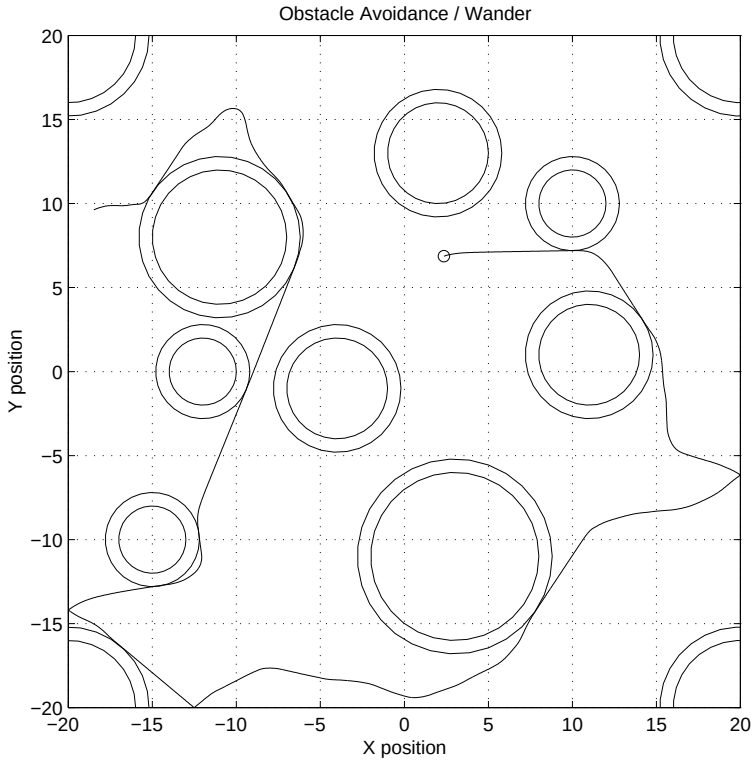
Testkørslerne viser også, at hvis to agenter bevæger sig i lige linie mod hinanden vil de ikke foretage nogen korrigerende styring, men blot fortsætte lige ud og derved støde ind i hinanden. Dette skyldes, at den korrigerende kraft beregnes som forskellen mellem de to agenters centrum ved kollisionen. Eftersom agenternes centre vil ramme ind i hinanden vil forskellen mellem disse være en nulvektor, hvilket resulterer i den manglende styring.

Obstacle Avoidance

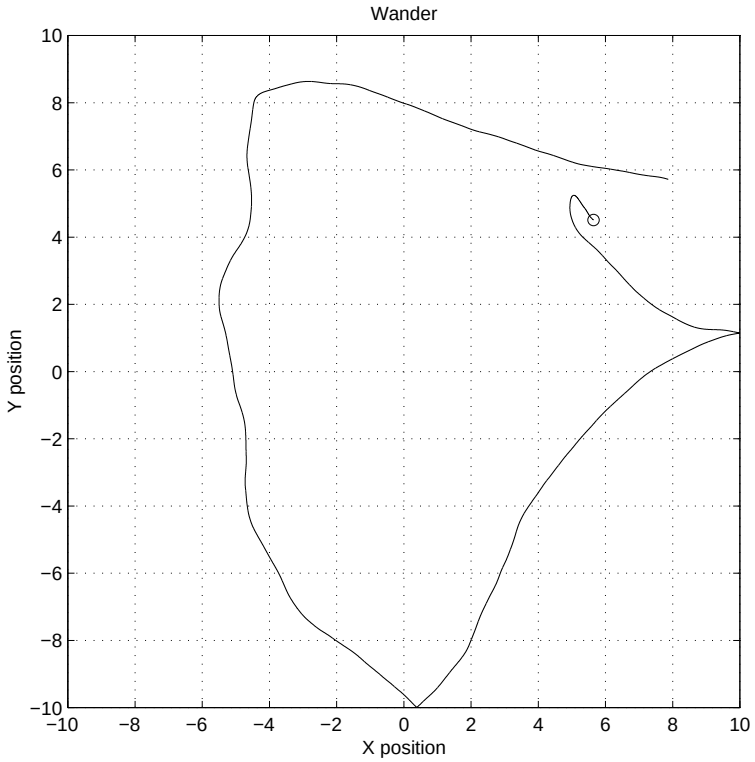
Udover de i Afsnit 5.1.7 beskrevne overvejelser angående, hvordan man kan bestemme en god kraftpåvirkning, har denne Behvaiour virket efter hensigten. På Figur 6.5 kan man se, hvordan en agent har bevæget sig rundt blandt en håndfuld forhindringer. Hvis agenten ikke har foretaget styring for at undgå en forhindring har den benyttet Wander.



Figur 6.4: Avoidance



Figur 6.5: Obstacle Avoidance



Figur 6.6: Wander

Wander

Alt i alt er det ikke så ligetil at teste denne Behaviour, da den ikke har et håndgribeligt formål. Ideen med Wander er at give en agent mulighed for at bevæge sig tilfældigt rundt i verden. På Figur 6.6 ses en simulering, hvor en Wander agent bevæger sig rundt i en åben verden.
Som det ses bevæger agenten sig altid nogenlunde fremad, men kan finde på at dreje rundt om sig selv. Der er ikke nogen skarpe "knæk" i bevægelsen (bortset fra ved kanterne) hvilket skyldes at tilfældigheden ligger i ændringen af kraftpåvirkningen snarere end i selve kraftpåvirkningen eller agentens hastighed.

6.2.2 Behaviour animationer

For at give et bedre indtryk af, hvordan agenterne opfører sig er der lavet nogle små animationer. På den tilhørende CD findes i biblioteket `prototype` nedenstående filer. Fælles for simuleringer er, at:

- agenterne er normalt blå
- gule cirkler[3] er forhindringer, som agenterne vil prøve at undgå
- når agenterne er røde forudser de en kollision med en anden agent. Den anden agent i kollisionen er dog ikke markeret, med mindre den selv forudser kollisionen
- når agenterne er gule forudser de en kollision med en forhindring
- den magentafarvede streg foran agenten er hastigheden
- den blå streg er kraftpåvirkningen
- centrum af billedet er markeret med en lille, rød kugle
- grænsen for det simulerede område er markeret med et rødt kvadrat
- hvis en agent rammer grænsen for området bliver dens hastighedsvektor spejlet om den side, der støder ind i

seeksingle.mpg

Viser tre forskellige Seeks mod centrum. Når afstanden fra agenten til centrum er mindre end 0, 4, halvdelen

[3] I implementeringen er der reelt set tale om kugler.

af den personlige sfæres radius, se Afsnit 3.2.4, siges agenten at være nået frem til målet.

seek.mpg

Viser fem agenter, der alle søger ind mod centrum, men de forsøger at undgå hinandens personlige sfærer. Dette resulterer i en lille "dans" rundt om centrum som følge af, at de vil skifte mellem Seek og Avoidance.

arrival.mpg

Som beskrevet i Afsnit 5.1.3 benyttes der her en linæer afhængighed til at dæmpe maksimal hastigheden, når agenten kommer tæt på centrum. I modsætning til Seek benyttes ikke kun afstanden til centrum, men også agentens hastighed, som kriterie for, at den er nået frem til sit mål. Dette stemmer overens med definitionen af Arrival.

wandersingle.mpg

En enlig agent, der vandrer gennem en mørk og dunkel verden. Bemærk, at idet tilfældigheden ligger i ændringen af kraftpåvirkningen vil denne ikke foretage store spring. Efterfølgende vil hastighedsvektoren og agentens position give bløde bevægelser.

Det er dog ikke umuligt for agenten at snurre (næsten) helt rundt om sig selv, som det ses kort før halvvejs gennem animationen (efter cirka 4 sekunder). Her er agentens kraftpåvirkning så lille, at den tilfældige ændring får bremset og drejet agenten næsten hele vejen rundt om sig selv.

wander.mpg

Her benytter 10 agenter Wander til at bevæge sig rundt i verden. Hvis de er ved at støde ind i hinanden vil de foretage styring som beskrevet i Afsnit 5.1.6.

Idet der er tale om et forholdsvis lille område til de ti agenter vil der være situationer, hvor de ikke når at styre uden om hinanden. Når dette sker vil agenterne overlappe hinanden, hvilket skyldes, at der ikke er nogen (fysisk) begrænsning, der forhindrer overlap. Der vil i forbindelse med beskrivelsen af det synkende skib, Afsnit 8.1, blive beskrevet, hvordan en sådan begrænsning kan laves og få agenterne til at skubbe til hinanden.

pursuitwander.mpg

Den hvide agent i denne animation benytter Wander til at bevæge sig rundt. Den blå agent benytter Pursuit til at fange den hvide og forudser den fremtidige position af byttet ved lineær interpolation. Det punkt, der bliver brugt til det endelige Seek er vist ved en lille grøn kugle. Denne vil som følge af den lineære forudbestemmelse befinde sig på en linie gennem byttets hastighedsvektor.

obstacleavoidance.mpg

Her ses fem agenter i en verden med tolv cirkelformede, gule forhindringer. For både at undgå de andre agenter og forhindringerne benytter hver agent først Avoidance og dernæst Obstacle Avoidance.

I løbet af arbejdet med prototypen er der jævnligt lavet små simuleringer, for at illustrere nye effekter eller specielle mønstre. Disse simuleringer følger kun sjældent ovenstående konventioner med hensyn til farvekodning og agentmodeller og størrelser. De er inkluderet for at give et indtryk af, hvordan prototypen har udviklet sig, og vil derfor ikke blive beskrevet i detaljer her. En kort beskrivelse af de enkelte animationer findes på CD'en.

Kapitel 7

Systemanalyse

Dette kapitel beskriver Software Requirements Phase fra [Hansen] for simuleringssystemet. Formålet med systemet er, som gennemgået i Kapitel 2, at designe og implementere et system, der kan benyttes til at simulere adfærd under forskellige former for ekstern påvirkning.

Kapitlet er inddelt i følgende hovedsektioner:

Model
: beskriver programmets opsplitning i forskellige moduler, samt disse modulers generelle formål og sammenhængen mellem dem i forhold til programmet som helhed.

Funktionalitet
: giver en mere indgående beskrivelse af de enkelte moduler.

Dataflow
: gennemgår den måde de enkelte moduler i programmet kommunikerer med hinanden på, nemlig et centraliseret beskedsystem.

Hovedstruktur
: indeholder en oversigt over, hvordan en komplet simulering foregår, inklusive en oversigt over de dele, en simulering består af.

7.1 Model

På Figur 7.1 er programmets opsplitning i moduler vist. Grundlæggende er ideen at beskrive en verden bestående af to dele: de autonome agenter og den verden de eksisterer i. Udover disse to dele er der i systemet en brugergrænseflade, der giver brugeren mulighed for at definere og se simuleringerne.

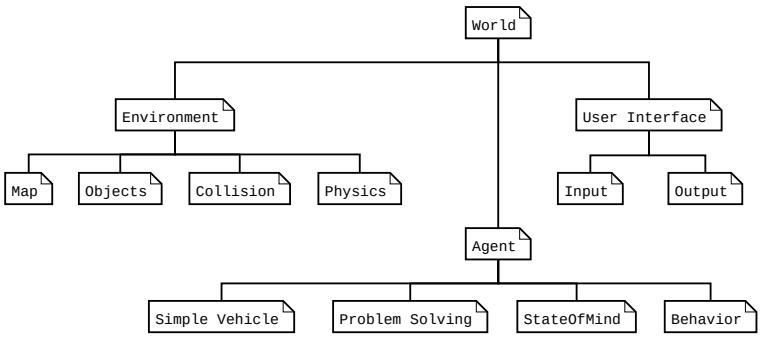
Agents
: autonome individer med mulighed for at tage pseudointelligente beslutninger om både lang- og kortsigtede mål, henholdsvis Problem Solving og Behaviours.

Environment
: den fysiske verden. Dette Environment består således af selve beskrivelsen af verden, et Map samt Objects, og en beskrivelse af, hvilke fysiske love der gælder i verden. Yderligere indeholder dette modul en funktionalitet der, indenfor de givne fysiske rammer, foretager collision detection og response for Objects.

User Interface
: giver brugeren mulighed for at definere agenter og den simulerede verden. Herudover vil der gennem denne del være mulighed for at se og gemme simuleringen.

Denne model er fremkommet ud fra de erfaringer, arbejdet med prototypen har givet. Det primære valg har været at lave en todeling af verden, så "levende" ting er adskilt fra "døde" ting. Teoretisk set kunne man lade en sten være en autonom agent, hvis adfærdsmønster er bestemt af de fysiske love. Umiddelbart vil dette dog ikke virke rigtigt i forhold til den fysiske verden.

Efter dette første valg er foretaget har det været nødvendigt at analysere hvert af modulerne og lave indelinger ud fra modulets funktionalitet.



Figur 7.1: Model

7.2 Funktionalitet

Dette afsnit beskriver de forskellige moduler i projektet. For hvert af de to hovedmoduler vil der være en beskrivelse af, hvad modulet generelt skal kunne, samt en uddybende forklaring af de på Figur 7.1 viste elementer.

7.2.1 World

Selve den simulerede verden er beskrevet ved et World modul, der primært varetager kommunikation mellem agenter og det benyttede Environment.

Modulets primære funktioner vil være at:

- initialisere simuleringen
- varetage kommunikation mellem undermodulerne
- afslutte simuleringen når passende stopkriterie(r) er opfyldt

Den interne kommunikation mellem de enkelte moduler bliver styret fra World modulet ved brug af et beskedsystem, en Message Router[1]. Denne Router vil kunne varetage de forespørgsler agenterne har om hinanden og resten af verden. En af de grundlæggende ideer med World modulet er således at implementere en kommunikationsmodel, der minder om den virkelige verden. I teorien kunne de enkelte agenter have adgang til hinandens data, men idet den virkelige verden fungerer ved at den enkelte agent "spørger verden" om, hvad den kan se, benyttes her det samme princip. I Afsnit 7.3 beskrives, hvordan disse data kan udleveres ved brug af Polling og Events.

For at agenterne kan tage deres beslutninger har de brug for en vis viden om verden. De er selv klar over deres egen position og hastighed, men de vil få brug for at stille følgende spørgsmål til verden:

Hvem kan jeg se?
: En agent ønsker at "se", hvilke agenter der er i nærheden af den selv.

Hvem støder jeg ind i?
: En agent ønsker at se den eller de agent(er), den støder ind i inden for den nærmeste fremtid. Hvis der ikke er nogen nært forestående kollision får agenten dette at vide.

Hvad kan jeg se?
: En agent ønsker at "se", hvilke fysiske objekter, der er i nærheden af den selv.

Hvad støder jeg ind i?
: En agent ønsker at få at vide, hvilket fysisk objekt, den støder ind i inden for den nærmeste fremtid. Hvis der ikke er nogen nært forestående kollision får agenten dette at vide.

Hvordan løser jeg dette problem?
: En agent ønsker at få en løsning på sit nuværende problem, eksempelvis at finde den korteste vej fra dens nuværende position til et andet sted i det simulerede Environment.

Disse spørgsmål bliver af World modulet besvaret med de data agenten har brug for for at kunne forholde sig til et eventuelt problem. For agent kollisioner vil dette være position, størrelse og hastighed for den anden agent i kollisionen.

[1] Begrebet Router benyttes i denne sammenhæng til at beskrive et koncept, snarere end en konkret teknologi.

7.2.2 Agent

En agent er et selvtænkende individ, der på egen hånd vil forsøge at opfylde et eller flere mål. De vigtigste egenskaber ved en agent er dens viden om sin egen position i verden samt dens bevægelsesmønster. Ud fra denne viden kan agenten forholde sig til sine mål i verden, samt de umiddelbare komplikationer, der opstår.

Som beskrevet i Kapitel 3 har en agent en fysisk repræsentation i verden. Denne fysiske del er som nævnt i Afsnit 3.2 modelleret som et Simple Vehicle, der har sin egen fysiske repræsentation i form af et objekt.

Behaviour

Begrebet Behaviour dækker over en agents forskellige måder at forholde sig til de nærmeste omgivelser. Dette kan være alt lige fra simpel højresøgning rundt om et objekt til Flocking. Dette modul skrives som et undermodul til Agent modulet og indeholder ikke blot de i Kapitel 5 beskrevne Behaviours, men også de øvrige fra [Reynolds].

Det bør nævnes, at en af de beskrevne Behaviours omhandler kollisioner med de fysiske objekter i verden. Selve checket for, om agenten støder ind i et objekt er knyttet til Environment modulet, men det valg, der skal til for at undgå objektet, ligger i Behaviour delen af Agent modulet.

Problem Solving

Denne del af en agents beslutningstagning berører de overordnede mål en agent ønsker at opfylde. Agenten skal gennem StateOfMind kunne spørge om den har nået sit nuværende delmål samt modtage det næste, såfremt der er flere i mållisten.

Det er ikke på nuværende tidspunkt fast besluttet, præcis hvilke problemtyper, der skal kunne løses. Der er dog det krav, at Problem Solving delen af en agent stiller et standardiseret interface til rådighed, således at det med tiden vil være muligt at tilføje nye problemtyper uden at skulle lave fundamentale ændringer i designet.

StateOfMind

For at knytte problemløsningsdelen sammen med de simple Behaviours en agent har, benyttes StateOfMind modulet, der varetager, hvilke Behaviours, der skal benyttes og vægtes, afhængig af hvilken type mål agenten er igang med at nå. Denne sammenhæng kunne tænkes implementeret som en tilstandsmaskine, men et alternativ ville være at lade et regelbaseret system analysere målet og udfra dets indhold bestemme, hvilke Behaviours, der er nødvendige, og hvordan de skal benyttes.

Simple Vehicle

Denne del af en agent repræsenterer det, der i [Reynolds] kaldes Locomotion. En agents Simple Vehicle kan ses som bindeleddet mellem den adfærd en agent udviser og den måde, den kan bevæge sig på i den simulerede verden.

Formålet med at have Simple Vehicle som en underdel af en agent er, at det vil være muligt at definere forskellige typer af Simple Vehicles, der kan knyttes til en agent. På denne måde vil det være muligt at benytte samme type Simle Vehicle, eksempelvis et menneske, til flere forskellige typer af agenter. Yderligere vil det i definitionen af simuleringerne være muligt udskifte typen af en agents Simple Vehicle.

7.2.3 Environment

Den anden hoveddel af programmet er et modul, der varetager den fysiske verden, det simulerede Environment. Dette modul har med selve den fysiske del af den simulerede verden at gøre. Der vil her findes de funktioner, der skal til for at repræsentere verden og dens udseende.

Map

Dette er selve beskrivelsen af den fysiske verdens opbygning. Map indeholder beskrivelser af, hvor agenter kan gå, samt hvilke hastigheder, der er mulige i de forskellige områder. Det er ikke planen, at Map skal indeholde en fuldstændig korrekt gengivelse af samtlige detaljer i den fysiske verden. Ideen er at benytte Map på samme måde som en normal person ville benytte eksempelvis et vejkort.

Idet målet med projektet er at simulere (en del af) den virkelige verden vil Map kunne indeholde forskellige typer af abstraktioner af den fysiske verden. Formålet med modulet er således at gøre den egentlige repræsentation af verden transparent for agenten. Modulet skal videregive informationerne om verden på en måde, så agenten kan forholde sig til dem.

Som udgangspunkt bør Map indeholde alle oplysninger om den fysiske verdens opbygning, men idet dette vil give en unødvendig stor datamængde vil den færdige repræsentation være tilpas simplificeret til, at det er realistisk at lave beregninger på dataene. Grundet den valgte metode til at lave ruteplanlægning, se beskrivelsen af `pathfind` i Afsnit 6.1.1, repræsenteres kortet som et todimensionelt array af landskabstyper, hvor det for hver landskabstype er defineret, hvor hurtigt agenterne kan bevæge sig. En fordel ved dette er, at hele landskabet kan beskrives som en graf, hvor det kun er nødvendigt at gemme information om knuderne idet kanterne er givet ud fra den rektangulære opbygning af grafen.

Objects og Physics

Som en udvidelse til det statiske kort over verden findes her en liste over alle de dynamiske objekter i verden. Disse objekter kan bevæges rundt i verden og påvirkes af hinanden og de autonome agenter. Yderligere kan der for simuleringerne være ønske om, at de skal overholde naturlovene fra den fysiske verden.

Idet dette projekt ikke direkte omhandler denne funktionalitet henvises til eksempelvis [Watt], [Foley] eller [Lin] for en beskrivelse af, hvordan dette modul i sin endelige version kan se ud.

7.2.4 User Interface

Det primære formål med systemets brugergrænseflade er at lade brugeren definere forskellige simuleringer. Basalt set består dette modul af to dele, Input og Output, der tilsammen giver brugeren følgende muligheder:

- definere simuleringens Environment
- definere simuleringens agenter
- se en grafisk repræsentation af simuleringen
- gemme data fra simuleringen

Der er forskellige måder at udforme disse på, men det er forestillingen, at selve systemet indeholder et filformat, der definerer en simulering. Initialiseringen af simuleringen vil ud fra brugerens datafil generere hele den simulerede verden.

Til at konstruere datafiler til systemet bør der udvikles et hjælpeværktøj, der lader brugeren arbejde grafisk med verden og placeringen af agenterne. Til trods for, at et sådant hjælpesystem ikke er en del af selve simuleringssystemet bør det kraftigt overvejes at implementere det, idet det vil være til hjælp i forbindelse med afprøvning af systemet.

7.3 Dataflow

Den vigtigste metode til informationsudvikling i projektet er en standardiseret kommunikation mellem de forskellige moduler. I [Rabin] har Rabin gjort sig nogle overvejelser for og imod brugen af Polling i forhold til et Event styret system. Hans konklusion er, at selvom Polling virker mere naturlig som interaktionsform, er det i situationer med et stort antal agenter ikke et fornuftigt valg.

Dog er hans overvejelser primært rettet mod den generelle beslutningstagen samt de handlinger, der direkte påvirker andre agenter. Idet en stor del af en agents beslutninger tages ud fra informationer om agentens "lokalområde" vil dataflowet i systemet blive en kombination af Polling og Events.

De efterfølgende afsnit vil beskrive, hvordan Polling og Events benyttes til at give agenterne de oplysninger de har brug for.

7.3.1 Polling

Under udarbejdelsen af modellen for dataflowet i projektet har der været forskellige overvejelser angående brugen af Polling til at give agenterne informationer om det simulerede Environment. Som nævnt i [Rabin] er Events en god måde at styre beskeder fra eksempelvis en agent til en anden. Men når det drejer sig om agenternes muligheder for at "se" på verden for at få informationer om deres umiddelbare nærhed vil Events resultere i, at en stor datamængde skal sendes rundt til, potentielt, mange agenter.

En bedre måde at give agenterne adgang til de relevante data er at lade dem Poll'e verden og derigennem fortælle dem, hvad de bør interessere sig for. Når den enkelte agent har brug for data kan den gennem et passende interface til de andre agenter eller den fysiske verden bede om data.

I kombination med denne type "forespørgsler" vil der som en del af de enkelte moduler i systemet være en funktionalitet, der holder styr på hvad der kan forventes at påvirke den enkelte agent. På denne måde er det således ikke den enkelte agent, der analyserer hele det samlede system. Den enkelte agent får af systemet at vide, hvad den kan "se" og forholder sig deref ter til det. Dette vil involvere en sortering i, hvilke agenter og objekter, der eksempelvis skal laves collision detection mellem. Fordelen ved dette er, at der skæres ned i antallet af nødvendige beregninger per tidsskridt, hvilket vil gøre det muligt at køre større simuleringer, specielt i real time.

7.3.2 Events

Event styret kommunikation foregår ved at lade de forskellige dele af systemet sende beskeder til hinanden. Beskederne sendes gennem systemets centrale Message Router, der sørger for at levere hver besked til dens modtager på det rigtige tidspunkt. Når en besked er blevet leveret bliver modtageren gjort opmærksom på dette og vil behandle dette Event.

Nedenfor er beskrevet opbygningen af beskeder samt hvordan de leveres. Den benyttede model for beskeder er beskrevet i [Rabin].

Message

Formålet med en Message er at videregive informationer om et Event til en agent, der er berørt af dette Event. For at kunne opnå dette er der nogle informationer, der skal være i beskeden:

Type
: identifikation af hvad beskeden omhandler, eksempelvis *vent*

Afsender
: afsenderen af beskeden, eksempelvis *agent 42*

Modtager
: den modtagende agent, eksempelvis *agent 54*

Data
: data der kan uddybe typen af besked, eksempelvis *10* (sekunder, tidsskridt etc)

Disse fire informationer gør det muligt for kommunikationssystemet at lade *agent 42* fortælle *agent 54*, at den skal vente i et stykke tid. Dette er umiddelbart godt nok til at varetage de fleste ting, men der kan dog opstå problemer. Forestiller man sig en situation, hvor *agent 54* er ved at blokere vejen for *agent 42* vil det ikke være ønskeligt for *agent 42* at *agent 54* stopper og holder pause mens den blokerer vejen. Dette vil eksempelvis ske, hvis beskeden kommer så sent, at *agent 54* har bevæget sig så meget, at den står i en forkert position. Tilsvarende kan der være beskeder, der ikke skal leveres med det samme, men først på et bestemt tidspunkt.

Disse to ting lægger derfor op til, at beskederne udvides med en leveringstid samt en eller flere forudsætninger for at beskeden skal behandles.

Forsinkede beskeder

For at give mulighed for at levere beskeder et stykke ud i fremtiden udvides en besked til også at indeholde en leveringstid, eventuelt regnet relativt. En besked af denne type kan, i modsætning til den oprindelige type af beskeder, kaldes en forsinket besked, en Delayed Message.

Denne leveringstid vil i bund og grund ikke ændre noget på den oprindelige ide om en besked, idet en sådan kan betragtes som en besked, hvor modtagelsestiden automatisk bliver sat til tiden for afsendelse.

Forudsætninger for behandling

Derudover kan en besked indeholde informationer til modtageren om, hvilke forudsætninger, der skal være opfyldt, for at beskeden skal behandles.

Denne udvidelse er en anelse svævende, idet forudsætninger er meget nært knyttet til implementeringen af agenternes intelligens og adfærd. Da det kan forventes at agenterne bliver styret gennem en tilstandsmaskine vil det være nærliggende at benytte tilstande som forudsætninger. Et stort problem vil dog være, at de enkelte agenter kan være, og formentlig er, styret af forskellige tilstandsmaskiner. Det vil derfor som hovedregel kun give mening at benytte forudsætninger, når en agent sender en besked til sig selv - eller eventuelt til en modtager agenten ved er af samme type som den selv.

Levering

Umiddelbart vil måden at levere beskeder på være at anvende en kø, hvor de først leverede beskeder bliver behandlet først. Som nævnt ovenfor kan alle beskeder dog betragtes som havende en leveringstid, selv de beskeder, der skal leveres og behandles behandles hurtigst muligt. Disse beskeder vil ikke umiddelbart have en brugerbestemt leveringstid, men vil af systemet automatisk få tildelt en leveringstid, der gør, at de bliver leveret og behandlet i næste tidsskridt. Dette lægger op til, at man kan prioritere beskederne efter deres leveringstid, eventuelt kombineret med en prioritet baseret på typen af beskeden, afsenderen og/eller modtageren.

For at holde leveringssystemet simpelt er den ekstra prioritering fravalgt, og det er således kun modtagelsestiden, der benyttes til at sortere beskederne.

Behandling

Når systemets Message Router har leveret en besked til en agent vil agenten blive sat til at behandle denne besked. Dette vil involvere et check for, hvad beskeden betyder i agentens nuværende adfærdsmønster og som følge heraf muligvis en ændring af adfærd. Behandlingen af beskeden kan resultere i, at der bliver sendt en eller flere nye beskeder, men idet disse nye beskeder kommer ind i den normale beskedkø vil der være en adskillelse af behandlingen af beskeder og afsendelsen af nye. På denne måde vil der ikke opstå rekursion, hvor en besked resulterer i en besked, der resulterer i en besked, der resulterer i en besked og så videre.

7.4 Hovedstruktur

Ideen med dette afsnit er at klargøre, hvordan et typisk programforløb vil være. Basalt set vil systemet bestå af tre forskellige dele:

Initialisering
: Her indlæses data om simuleringen, det vil sige beskrivelser af Environment, antal og typer af agenter og tilsvarende. Disse beskrivelser fås fra brugergrænsefladen og benyttes til at initialisere hele den simulerede verden.

Simulering
Selve simuleringen foregår i skridt af enten fast eller variabel længde. I hvert skridt gennemløbes systemets hovedløkke, der sørger for at alle agenter bliver opdateret. En komplet beskrivelse af hovedløkken findes nedenfor.

Afslutning
Efter simuleringens afslutning gemmes de indsamlede data og en objektiv analyse af simuleringens udfald. Denne analyse indeholder en oversigt over, hvilke af deres mål de forskellige agenter nåede.

7.4.1 Hovedløkke

Hele systemet vil komme til at køre i et såkaldt sampled loop, hvor der for hver opdatering af systemet udføres en række faste ting. I forbindelse med real time simuleringer vil opdateringshastigheden være afhængig af, hvor mange beregninger der skal foretages i de enkelte gennemløb og det kan forventes, at der bliver tale om forskellige tider fra gennemløb til gennemløb. Idet der benyttes eulerintegration til at beregne agenternes position, hastighed og kraftpåvirkning kan dette resultere i, at der i tilfælde af meget beregningskrævende tidsskridt kan forekomme upræcise simuleringer.

I stedet for real time simuleringer vil der med systemet være mulighed for at lave en simulering, der benytter et fast opdateringsinterval. Efter hvert gennemløb gemmes agenternes data, enten som rådata eller som et enkelt billede fra den grafiske repræsentation af simuleringen til brug ved senere fremvisning. Dette vil give mulighed for at simulere store systemer, hvor antallet af agenter gør real time simuleringer upræcise.

Aflevering af Messages
Systemets Message Router leverer de beskeder, der skal behandles i det aktuelle tidsskridt.

Behaviours
Alle agenterne spørger deres aktive Behaviours og får derfra en eller flere kraftvektorer, der vil påvirke agenten.

Vægtning af kræfter
For hver agent gennemgås de fundne kræfter og ved brug af en for agenten passende vægtning af de enkelte Behaviours' kræfter findes nu den endelige kraftvektor for agenten.

Opdatering af position
Hver agent påvirkes af sin kraft og efterfølgende af sin hastighed. Dette giver agenten en ny position.

Check af position
Agenterne checker, om de bør ændre adfærd ud fra deres nye position. Ud fra det simulerede Environment tager agenterne stilling til, om deres position resulterer i deres død, en kollision med et (fysisk) objekt eller om de har nået et af deres mål.

Stopkriterier
Idet der er risiko for, at en eller flere agenter ikke kan få opfyldt alle deres mål, bør hovedløkken have stopkriterier, der forhindrer en uendelig løkke. Selvom det kunne være interessant at lade systemet køre i uendelig lang tid bør der dog være et stopkriterium, der sikrer at simuleringen stopper efter et passende tidsrum. Eventuelt kan der indbygges overvågning af agenternes tilstande og hvis alle agenter ender i en sluttilstand, eksempelvis "alle mål opfyldt" eller "død", vil hovedløkken terminere.

Denne måde at opdatere agenterne på er valgt ud fra de overvejelser der er gjort i forbindelse med prototypen af programmet. I prototypen opstod det problem, at hvis to agenter var ved at støde ind i hinanden var det kun den ene agent, der "så" problemet. Dette skyldtes, at hver agent blev helt færdigbehandlet før systemet gik videre til den næste. Det vil sige, at hvis en agent så en kollision ville den ændre retning og når så den anden agent i kollisionen blev behandlet ville der ikke være en kollision. Resultatet af dette var således, at kun den først behandlede af de to agenter ville være klar over kollisionen og forholde sig til den. Dette betød, at behandlingsrækkefølgen var vigtig for udfaldet af simuleringen, hvilket ikke er ønskeligt.

Kapitel 8

Simuleringer

8.1 Synkende skib

Udover at have tjent sit formål med at teste de enkelte Behaviours har prototypen også været brugt til at simulere matroser og passagerer ombord på et synkende skib. Dette afsnit vil først beskrive oplægget til simuleringen og derefter gennemgå de resultater der er opnået ud fra en udvidelse af prototypen.

8.1.1 Situation

Denne situation går kort fortalt ud på at simulere adfærdsmønstre hos matroser og passagerer på et synkende skib. Ideen er, at i startsituationen er alting normalt, og både matroser og passagerer går tilfældigt rundt på skibet.

Efter et stykke tid begynder skibet at synke. Dette har ingen indflydelse på, hvordan agenterne kan bevæge sig rundt, men deres adfærd vil ændre sig. Det antages, at skibet ikke vipper, og at der ikke kommer vand indenbords, så repræsentationen af skibet vil ikke ændre sig.

Matroserne vil først søge hen til hver deres "ansvarsområde" og derefter til hver deres redningsflåde. Et ansvarsområde er et sted på skibet, matrosen skal besøge før han må sætte sin redningsflåde i vandet. Ideen med at give hver matros et ansvarsområde er at sikre, at alle skibets afkroge bliver gennemsøgt for passagerer før matroserne forlader skibet.

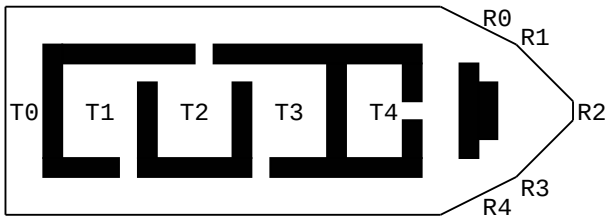
Passagererne vil løbe forvirret rundt, indtil de kan "se" en matros. Så længe, de kan se matrosen, vil de følge efter ham. Hvis han forsvinder fra deres synsfelt, vil de gå hen til det sidste sted, de så ham, og fortsætte i nogenlunde samme retning, men med lidt tilfældighed i deres gang. Hvis en passager når til en redningsflåde, vil han kravle op i den og blive der resten af simuleringen.

En redningsflåde kan højst indeholde et vist antal passagerer. I første omgang vil det samlede antal pladser i redningsflåderne være det samme som antallet af personer om bord på skibet. Denne simplificering er lavet for at gøre det nemt for en matros at vurdere om han skal sætte sin redningsflåde i vandet eller ej: så længe der er ledige pladser må han ikke sætte den i vandet.

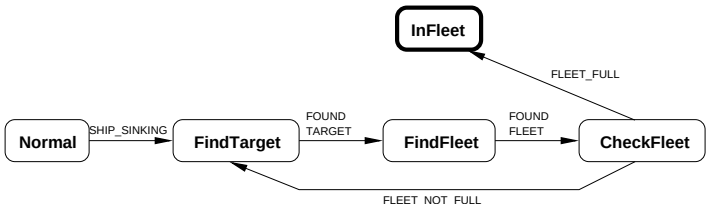
En sidste bemærkning om skibet er, at redningsflåderne er den eneste måde at undslippe druknedøden på. Der er ingen redningsveste, vragstumper eller lignende som vil kunne hjælpe passagererne til at overleve i tilfælde af, at de ikke når ombord i redningsflåden.

8.1.2 Environment

På Figur 8.1 er vist et muligt Environment for simuleringen. $T0$ til $T4$ er matrosernes "ansvarsområder", Targets, og $R0$ til $R4$ er matrosernes redningsflåder, Rafts.



Figur 8.1: Synkende Skib Environment



Figur 8.2: Matros tilstandsmaskine

Agenterne har mulighed for at bevæge sig i de hvide felter, de sorte felter repræsenterer vægge og andre forhindringer.

Hvis en agent bliver presset ud over rælingen (den sorte streg rundt om skibet) antages det, at vedkommende falder i vandet og drukner - det er en hård verden, vi lever i.

8.1.3 Agenter

Der findes i denne simulering to typer af agenter:

Matros
: Matroserne har til hovedopgave at guide passagerer hen til redningsflåderne. Dette gøres ved, at hver matros bevæger sig ud til sit "ansvarsområde" og derefter hen til sin redningsbåd. Formålet med denne fremgangsmåde er at samle passagerer op og guide dem hen i nærheden af flåderne.
Matroserne har kendskab til hele kortet og ved derfor, hvor flåderne er, og hvordan man kommer derhen. Yderligere har de "forkørselsret" i forhold til passagererne, så matroserne burde ikke risikere at blive fanget i en menneskemængde.

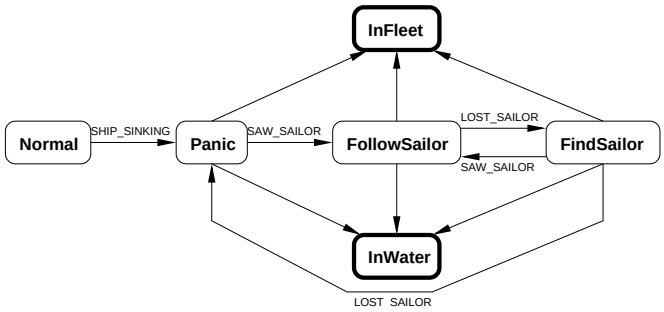
Passager
: Passagererne kan i denne simulering betragtes som relativt dumme. Efter skibet er begyndt at synke, kan de gøre tre ting:
  - gå tilfældigt rundt
  - følge efter den nærmeste matros
  - gå hen til det sidste sted de så en matros

Agenternes adfærdsmønstre realiseres ved to tilstandsmaskiner, der er vist på Figur 8.2 og Figur 8.3. Tilstande, der er markeret med en tyk ramme er sluttilstande, hvorfra agenten ikke kan komme videre.

I designet af tilstandsmaskinerne er der foretaget det valg, at matroserne er tænkende individer og passagererne er (stort set) ikke-tænkende individer. På baggrund af dette valg er det bestemt, at matroserne har overblik nok over situationen til at de ikke falder i vandet. Omvendt vil det være muligt for passagererne at falde i vandet som følge af, at de skubber til hinanden.

8.1.4 Mål

Målet for simuleringen er at se hvor mange passagerer, der når at blive reddet over i redningsflåderne inden skibet til sidst synker. Når dette sker, vil alle ombordværende drukne, hvad enten de er matroser eller passagerer.



Figur 8.3: Passager tilstandsmaskine

8.1.5 Udvidelsesmuligheder

En mere realistisk simulering af forholdene ombord på et skib vil have en forskel mellem antallet af pladser i redningsflåderne og antallet af personer om bord på skibet. Under antagelse af, at skibets ejer ikke overfylder skibet vil der være overskydende pladser, så hver redningsflåde ikke nødvendigvis behøver være helt fyldt op. Dette vil stille krav om at matroserne kommunikerer indbyrdes, så de ved hvor mange passagerer der stadig går rundt på skibet. Derudover kan man lade matroserne tage flere ture på skibet indtil de enten har fyldt deres redningsbåd eller skibet er faretruende tæt på at gå helt under.

Begge disse udvidelser til matrosernes adfærd kræver en vurdering af antallet af passagerer i forhold til antallet af ledige pladser i en redningsflåde. En matros skal kunne afgøre, om det kan betale sig at lede efter flere passagerer eller om det er bedst at redde dem, der allerede er i redningsflåden.

Disse udvidelser er ikke implementeret, da de stiller et krav om kommunikation mellem matroserne. Prototypen er ikke struktureret nok til at kunne opfylde uden så mange udvidelser, at den stort set vil blive til det endelige system.

8.1.6 Implementering

Den i Kapitel 6 beskrevne prototype er benyttet som grundskelet til at foretage skibsimuleringen. Der er dog lavet et par ændringer i forhold til det beskrevne system, som beskrevet i Afsnit 6.1.1. De vigtigste ændringer er opsummeret nedenfor sammen med en kort beskrivelse af, hvordan de forholder sig til netop denne simulering.

Nye moduler
: Udregninger vedrørende skibet og agenterne er flyttet ud i et `ship` modul og et `agent` modul. Disse to moduler definerer datastrukturerne `Ship`, der indeholder oplysninger om skibets fysiske udformning (forhindringer/vægge, vand, redningsflåder, matroser og passagerer) og `Agent`, der primært indeholder agentens tilstandsmaskine og dens `SimpleVehicle`.
Endelig er der tilføjet et `pathfind` modul, der kan omsætte et skibs repræsentering til et pixeleret kort og lave en korteste vej søgning udfra givne start- og slutpunkter.

Hovedløkke
: Idet et `Ship` indeholder to agentlister, `sailors` og `passengers`, er opdateringen af agenter splitter ud i to løkker, en til hver liste. Yderligere benyttes der ikke et globalt array til at indikere, om agenterne er aktive eller ej, dette er lagt ud i den enkelte `Agent`.

Collision detection
: I den oprindelige prototype var disse lagt ud i de enkelte Behaviours, der derved fik til opgave at udføre collision detection. Dette er nu ændret, så den enkelte Behaviour blot får at vide, at en given agent er ved at støde ind i et andet objekt, forhindring eller agent.
Som en opfølgning på collision detection er der tilføjet et simpel collision response. For passager-passager kollisioner skubbes hver agent direkte væk fra den anden, til de ikke længere overlapper. For passager-matros kollisioner er det kun passageren, der flyttes. Dette simulerer, at passagererne er mere tilbøjelige til at flytte sig, når der kommer en matros for tæt på, end hvis det er en anden passager.

8.1.7 Resultater

I forbindelse med det synkende skib er der blevet udført et antal forskellige simuleringer af stigende kompleksitet. Herved er de enkelte dele af simuleringen blevet testet:

- Matrosernes evne til at finde rundt på skibet
- Passagernes evne til at følge efter matroserne
- Hvordan matroserne udfører deres primære opgave med at få hjulpet passagerne hen til redningsflåderne.

Nedenstående er en gennemgang af, hvad de forskellige simuleringer på den tilhørende CD viser. Klippene ligger i biblioteket `skib`. For alle klippene gælder at:

- matroser er vist som en blå pil
- passagerer er grønne pile
- vand er blå polygoner
- forhindringer og vægge er grå polygoner
- redningsflåder er grønne polygoner

Der er i simuleringerne ikke indført nogen (fysisk) begrænsning, der forhindrer en agent i at gå ind i en væg. I prototypen accepteres dette, primært fordi formålet med prototypen har været at vise, at det er muligt at lave et system til adfærdssimuleringer. Derudover er de viste modeller for agenterne, specielt deres bounding circles, ikke det præcise fysiske udseende af agenterne, men blot et udtryk for deres personlige sfærer, altså et område de helst ikke ønsker at have andre agenter i. På grund af dette har de derfor lidt spillerum med hensyn til, hvordan de bevæger sig i forhold til væggene.

ship_sailorpath.mpg
: Dette klip viser, hvordan en matros går fra sit startpunkt i en af redningsflåderne til sit målpunkt i den anden ende af skibet og tilbage. I animationen er indtegnet den del af den fundne rute, som agenten mangler at følge.

ship_water.mpg
: Her ses en af de første simuleringer, hvor der er både passagerer og en matros. En særdeles underholdende detalje i denne simulering er, at matrosen få skubbet til de to passagerer, så en af dem falder i vandet. Dette er absolut ikke en heldig situation og illustrerer, at matrosernes blindhed overfor andet end deres målområde og redningsflåden ikke just er optimal.[1]

ship_simplesimulation0.mpg
: Her ses et simpelt skib, hvorpå en matros får reddet tre passagerer over til redningsflåden. Læg mærke til, hvordan passagererne begynder at forfølge matrosen, når han er blevet synlig. Inden de ser ham, bevæger de sig tilfældigt rundt. Efter de har set ham, bevæger de sig hen imod ham og følger efter ham. Bemærk også, at passagererne bliver skubbet rundt om matrosen som resultat af deres collision response.
I denne simulering er passagererne ikke istand til selv at finde hen til redningsflåden, selvom de kan se den. Dette resulterer i, at matrosen løber frem og tilbage en ekstra gang, før den sidste passager er blevet reddet.

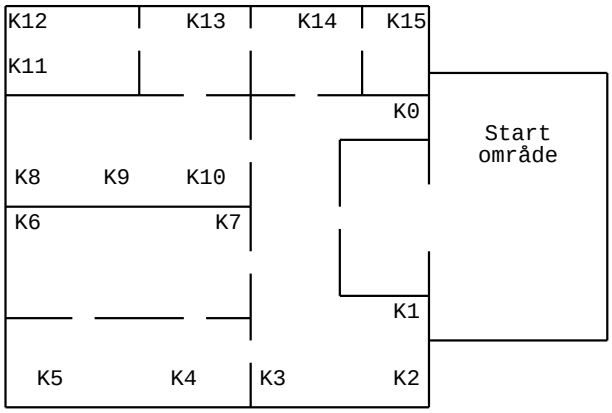
ship_simplesimulation1.mpg
: Dette er samme simulering som ovenfor beskrevet. Dog kan passagererne her se redningsflåden og selv finde derhen. Til trods for denne forbedring begynder matrosen dog på en ekstra tur, og går endda i vejen for passagererne, men får til gengæld en "besked" om, at redningsflåden er blevet fyldt op. Denne besked er dog ikke sendt via en Message Router som beskrevet i Afsnit 7.2.1, men er lavet som et tilstandsskift i matrosens tilstandsmaskine.

Det på Figur 8.1 viste environment beskriver en væsentlig større simulering end der er benyttet i ovenstående animationer. Dette skyldes primært, at problemerne med prototypen, nævnt i Afsnit 6.1.3, har gjort det besværligt at udvide den.

8.2 Udstilling

Denne simulering er inspireret af lektor Hansens oplevelser på udstillingen Expo 2000. Til trods for at besøgstallet var væsentlig lavere end forventet, var der stor trængsel og lange køer foran de enkelte udstillinger, så det var

[1] Alternativt kan denne handling skyldes matrosens mening om passageren, men *det* er en helt anden diskussion.



Figur 8.4: Udstilling Environment

ikke muligt at fordybe sig i det fremviste materiale. Ligeledes gjorde det store antal besøgende på Expo2000 det svært at nå at se mere end nogle få udstillinger.

Formålet med denne simulering er ikke at lave en komplet model af en udstilling i stil med Expo 2000, men blot at tage de første skridt i den retning.

Simuleringen vil forsøge at vise, hvad der sker, når et (lille) antal målbevidste agenter forsøger at opnå deres mål, mens de vandrer rundt blandt et (stort) antal målløse agenter.

I modsætning til simuleringen af det synkende skib er denne simulering ikke implementeret, og nedenstående skal ses som et oplæg til videre arbejde med systemet.

8.2.1 Situation

Ved starten af simuleringen befinder der sig et antal agenter på udstillingsområdet. Disse agenter har ikke nogen bestemte ønsker om, hvad de vil se. De går mere eller mindre tilfældigt rundt i området, så længe simuleringen varer.

Efter et stykke tid begynder der at ankomme nye agenter, der i modsætning til de andre ønsker at se nogle bestemte kunststykker. Deres mål er at besøge disse kunststykker inden for en begrænset tid. Når tiden er udløbet, vil de forlade udstillingen.

8.2.2 Environment

På Figur 8.4 er der vist et udstillingsområde, der består af en række forbundne lokaler, hvori der er et eller flere kunststykker, man kan beskue. På Figur 8.4 er der vist et udstillingsområde, hvorpå der er 16 kunststykker, markeret ved K_0 til K_15. En agent antages at have set et kunststykke, når den har været inden for en vis afstand af kunststykket.

De eneste begrænsninger på en agents bevægelsesevne er de indtegnede vægge og de øvrige agenter.

8.2.3 Agenter

Til denne simulering implementeres adfærd for to forskellige agenttyper:

Pøblen
: Pøblen består af agenter, der har hørt, at man bør besøge udstilingen. De forstår ikke at værdsætte udstillingens indhold og ender derfor med blot at bevæge sig formålsløst rundt i udstillingslokalerne.

Feinschmeckere

Disse agenter har en mening om hvilke kunststykker, der er værd at se, og vil derfor gøre deres bedste for at nå rundt til dem. Dette simuleres ved at give hver agent en liste af mål, realiseret ved positioner i området, som den skal nå at besøge indenfor en given tidsramme.
Idet feinschmeckerne har lavet et stykke forhåndsarbejde har de hver især planlagt, hvordan de bedst udnytter tiden på udstillingen. Dette betyder, at de har forsøgt at finde den rækkefølge, der sikrer, at de spilder mindst mulig tid på noget så uinspirerende som at gå fra et kunststykke til et andet.

8.2.4 Mål

Målet med simuleringen er at se, om feinschmeckerne når rundt til alle de kunststykker, de ønsker at se. Det interessante er at se de tilfældigt vandrende agenters påvirkning af feinschmeckerne. Specielt er det interessant at undersøge, om de ender med at blive "fanget" af pøblen og på den måde ikke kommer gennem deres målliste.

8.2.5 Udvidelsesmuligheder

Som nævnt er denne simulering inspireret af lektor Hansens oplevelser på Expo2000. For at kunne simulere en udstilling af denne størrelse og type vil det være nødvendigt at definere, hvordan de forskellige dele af udstillingen påvirker agenten, når den kan se dem. I simuleringen af det synkende skib var det muligt at lave en simpel tilstandsmaskine til at bekrive passagererne og deres reaktioner til forskellige begivenheder.

På en stor udstilling vil agenterne typisk blive påvirket af adskillige sanseindtryk på samme tid. Dette vil eksempelvis kunne simuleres ved at bruge forskellige vægte på disse stimuli, eventuelt afhængigt af, hvor langt agenten er fra deres kilde. Der kan herefter vælges det kraftigste stimuli eller de kan kombineres til at give et potentiale felt, som beskrevet i [Reynolds]. Et sådant potentiale felt vil dog formentlig resultere i, at agenten ikke går hen imod et bestemt mål eller begivenhed. Felterne vil, i stil med elektriske eller magnetiske felter, danne en kraftpåvirkning, der kan holde agenten i en form for "balance", hvor den ikke bevæger sig mod de to mål, men går mellem dem.

Kapitel 9

Resultater

I de foregående kapitler er resultaterne af de forskellige dele gennemgået. Dette kapitel vil opsummere disse resultater og vurdere, i hvor høj grad målet med projektet er opnået. Derudover vil der være et kort oplæg til, hvordan dette projekt kan bruges i forbindelse med videre arbejde med adfærdssimuleringer.

9.1 Prototype

Arbejdet med prototypen er i det store og hele forløbet som forventet. Der blev ikke lavet nogen større systemanalyse, før de beskrevne Behaviours blev implementeret, hvilke har resulteret i en række komplikationer. Disse har primært været i forbindelse med kommunikationen mellem de forskellige moduler, og når prototypen blev benyttet til at lave større simuleringer.

Prototypen har vist, at det er muligt at benytte simple virkemidler til at lave ganske komplicerede simuleringer, og arbejdet med den har givet erfaringer og ideer, der kan benyttes i udviklingen af et mere generelt simuleringssystem.

9.2 Systemdesign

Udfra analysen af prototypen er det lykkedes at opstille en velstruktureret model for, hvordan et adfærdssimuleringssystem kan se ud. Der findes naturligvis alternativer til det her gennemgåede, men de valg, der er foretaget her, bygger på grundige overvejelser og praktisk erfaring.

Der er endnu nogle ting i den beskrevne model, der fremstår en smule løst defineret, specielt i forbindelse med problemløsningsmodulet. Dette skyldes, at den endelige implementering vil være stærkt afhængig af de præcise problemertyper, der skal løses. Den her beskrevne model omhandler primært, hvordan det generelle flow i systemet skal være, og en diskussion af de enkelte algoritmer og implementeringsmuligheder falder derfor udenfor denne rapports rammer.

9.3 Fremtidigt arbejde

Som nævnt i indledningen er der mange anvendelsesmuligheder for denne type simuleringssystemer. Der er i Kapitel 8 gennemgået simulering af både en krisesituation med et synkende skib og den mere stilfærdige udstilling. Systemet kan således allerede som prototype benyttes til at simulere begrebsmæssigt forskellige situationer. Det er derfor ikke urimeligt at påstå, at det i sin endelige version vil kunne simulere et langt større udvalg af situationer.

Et projekt, der specielt har forfatterens interesse, er at benytte systemet i forbindelse med styring og simulering af AGV'er. Dette kunne ske ved at benytte en AGV som bevægelsesdel, enten virtuelt som beskrevet i Afsnit

3.2.3 eller ved rent faktisk at benytte en virkelig AGV og lade dens adfærd blive styret af Agent modulet. Til dette formål skal der dog laves en del ændringer i systemet, da AGV'en ikke umiddelbart kan se og genkende objekter og andre agenter (AGV'er) i omgivelserne. En mulighed vil være at udskifte Environment modulet med den fysiske verden og World modulet med AGV'ens sensorer (kameraer, afstandssensorer og lignende), der benyttes til at iagttage verden.

9.4 Konklusion

Dette projekt har haft til formål at analysere, hvordan man kan lave adfærdssimulering ved brug af autonome agenter. Til dette formål er der fundet informationer om, hvad autonome agenter er, om hvordan de kan repræsenteres og hvilke simple adfærdsmønstre de kan gives.

Implementeringen af de simple adfærdsmønstre er blevet testet, og udfra dette er der opstillet en model for, hvordan de i projektet erhvervede erfaringer kan benyttes i senere arbejde med adfærdssimulering for autonome agenter. Yderligere har der i projektet været tid og mulighed for at benytte den udviklede prototype til at lave en testsimulering, der viser, at en udvikling af et generelt system ikke er en umulighed.

Endelig er der i rapporten givet eksempler på, hvordan det udførte arbejde relaterer til den virkelige verden. Her tænkes ikke blot på de rent håndgribelige ting som AGV'erne, men også de udførte testsimuleringer, der med hjælp fra eksperter fra andre fagområder, såsom psykologien, vil kunne give et realistisk billede af, hvordan mennesker opfører sig i forskellige situationer.

Der er i projektet nået de primære mål, der er beskrevet i indledningen og anses derfor for at være vellykket. Forfatteren håber, at han selv eller andre får mulighed for at arbejde videre med det generelle simuleringssystem. Hvad enten det er med henblik på virtuelle simuleringer eller til styring af AGV'er vil de opnåede og nedskrevne erfaringer kunne benyttes til at lave et velfungerende system.

Litteratur

[Aiello] John R. Aiello, Donna T. DeRisi, Yakov M. Epstein, Robert A. Karlin, *Crowding and the Role of Interpersonal Distance Preference*, Sociometry, Volume 40, Issue 3, 271-282.

[Argyle] Michael Argyle, Janet Dean, *Eye-Contact, Distance and Affiliation*, Sociometry, Volume 28, Issue 3 (Sep., 1965), 289-304.

[Arrøe] Skjalm Arrøe, *Dynamisk vision til mobil robot*, Specialprojekt ved Institut for Automation, DTU, 2001, `http://www.iau.dtu.dk/~ex37/webcam/` .

[Borenstein] J. Borenstein, H. R. Everett, L. Feng, *Where am I? Sensors and methods for mobile robot positioning*, University of Michigan, 1996.

[Boutell] Thomas Boutell, *GD Graphics Library*, `http://www.boutell.com/gd/`.

[Cormen] T. Cormen, C. Leiserson, R. Rivest: *Introduction to Algorithms*, MIT Press 1990.

[DeLoura] Mark DeLoura (editor): *Game Programming Gems*, Charles River Media 2000.

[Eising] Jens Eising, *Lineær Algebra*, MAT, DTH, 1993.

[Foley] J. Foley, A. van Dam, S Feiner, J. Hugher, *Computer Graphics Principle and Practice*, Addison-Wesley 1990.

[Grosberg] Mark Grosberg: *Programming Alternatives: State Machines*, ukendt årstal, `http://www.conman.org/people/myg/essays/states.html`.

[Hansen] Per Skafte Hansen, *Minimalist Software Engineering I: Definition*, IMM 1994.

[Jacobsen] Søren Kruse Jacobsen, *Netværk 1*, IMSOR 1990.

[Jönsson] F. Markus Jönsson, *An optimal pathfinder for vehicles in real-world digital terrain maps*, The Department of Numerical Analysis and Computing Science, The Royal Institute of Science, Stockholm, Sweden, 1997, `http://www.student.nada.kth.se/~f93-maj/pathfinder/index.html`.

[Lin] Ming C. Lin, *Diverse artikler om collision detection og fysiske simuleringer*, `http://www.cs.unc.edu/~lin/`.

[Nielsen] Frank Nielsen, *Grafteori - metoder og netværk*, Matematiks Institut, DTU, 1995.

[Olsen] Mikkel H. Olsen, Jens-Peter V. Jensen, *Eurobot 2001, Autonom robot*, polyteknisk midtvejsprojekt ved Institut for Automation, DTU, 2001.

[Patel] Amit J. Patel, *Amit's Thoughts On Path-Finding*, 2000, `http://theory.stanford.edu/~amitp/GameProgramming/`.

[Rabin] Steve Rabin: *Designing a General Robust AI Engine*, *Game Programming Gems*, 221-236.

[Reynolds] Craig W. Reynolds: *Steering Behvaiors for Autonomous Characters*, `http://www.red3d.com/cwr/steer/`, oprindeligt SIGGRAPH97, opdateret 1999.

[Stentz] Anthony Stentz, *Optimal and Efficient Path Planning for Partially-Known Environments*, International Conference on Robotics and Automation (ICRA), 1994, `http://www.frc.ri.cmu.edu/~axs/doc/icra94.ps`.

[Watson] Mark Watson, *AI Agents in Virtual Reality Worlds*, John Wiley & Sons, Inc., 1996.

[Watt] Alan Watt, *3D Computer Graphics*, Addison-Wesley 1989.

[Woo] M. Woo, J. Neider, T. Davis, D. Shreiner: *OpenGL Programming Guide, Third Edition*, Addison-Wesley 1999.

[Østerby] Tom Østerby, *Kunstig Intelligens - metoder og systemer*, Polyteknisk Forlag 1992.