

Konstruktion af et spil med kunstig intelligens

Gruppe C213



Teknisk Naturvidenskabelige Basisår
ved Aalborg Universitet

29. maj 2006

Titel:

Konstruktion af et spil
med kunstig intelligens

Tema:

Virkelighedens modeller -
netværk og algoritmer

Projektperiode:

P2, forårssemesteret 2006

Projektgruppe:

C213

Deltagere:

Allan Mørk Christensen
Anh Tuan Nguyen Dao
Esben Staunbjerg Pedersen
Kim Jung Nissen
Martin Midtgaard
Peter Heino Bøg

Vejledere:

Per Trosborg
Christina Grann

Oplagstal: 13

Sidetæl: 96

Bilagsantal og -art: 4 tekster + 1 CD

Afsluttet den: 29. maj 2006

Synopsis:

I dette projekt vil udviklingen af et action-baseret computerspil med kunstig intelligens, AI, blive gennemgået.

Kravspecifikationen baseres på resultatet af en spørgeskemaundersøgelse og analyser af spilgenrer og AI-principper fra problemanalysen. Spillet udvikles i C++ og benytter SDL, OpenGL og OpenAL, der alle gør det muligt at benytte spillet på flere platforme, som Windows og Linux.

Den kunstige intelligente modstanders handlingsmønster, inddeles i seks tilstande: *valg*, *streff omkring*, *led efter powerup*, *fejende i syne*, *jagt* og *flugt*. Til at skifte mellem disse tilstande, er der brugt fuzzy logic som beslutningsmotor. Når modstanderen er i tilstanden *jagt* eller *led efter powerup*, skal den kunne finde vej gennem banen, og til dette anvendes algoritmerne Dijkstra og A*.

Gennem projektet blev der, udover selve implementationen af AI i et spil, yderligere fundet ud af, at det kan være vanskeligt at gøre den kunstige intelligente modstanders handlinger ikkedeterministiske. Grundet manglende tilstande for den kunstige intelligens, kan modstanderne i spillet virke forudsigelige.

Forord

Denne rapport er udarbejdet i forbindelse med et P2-projekt af gruppe C213. Gruppen læser til softwareingeniører på Det Teknisk-Naturvidenskabelige Basisår ved Aalborg Universitet. Gruppen, som består af de samme seks medlemmer fra P0 og P1, har udarbejdet denne rapport i perioden 1. februar - 29. maj.

I rapporten vil specielle ord blive fremhævet med en stjerne (*) som refererer til ordlisten, der forefindes bagerst i rapporten. Efter hvert afsnit, eller sætning, i rapporten kan notation “[X]” forekomme; dette henviser til kilde nummer X i litteraturlisten, der ligeledes forefindes bagerst i rapporten. Undervejs i rapporten vil der blive nævnt flere spiltitler. Disse er noteret med et (⊗). Der kan findes yderligere information om disse spil i spillisten bagerst i rapporten.

Med denne rapport, medfølger et stykke software på en cd, som er blevet udviklet i programmeringssproget C++. Dette giver mulighed for at afprøve softwaren. På cd'en ligger også en kopi af softwarens kildekode. Dog kan der være sket ændringer i denne i tiden imellem denne udgivelse og eksamen.

Rapporten er skrevet, så projektforsløbet fremgår kronologisk, hvilket i realiteten ikke er tilfældet. Først analyseres forskellige aspekter indenfor spil & AI i problemanalysen, hvorefter det deciderede konstruktionsproblem ønskes løst, specificeres i problemformuleringen. Herefter påbegyndes arbejdet med at løse konstruktionsproblemet, hvilket resulterer i et endeligt produkt. Løsningen vil blive dokumenteret med udpluk af essentielle dele af koden - det resterende af koden forefindes på den vedlagte cd.

Gruppen vil gerne sige tak til følgende personer:

Beta-testerne

De, der deltog i spørgeskemaundersøgelsen
Gruppe C214, for at deltage i test af produktet

God fornøjelse med rapporten.

Allan Mørk Christensen

Anh Tuan Nguyen Dao

Esben Staunbjerg Pedersen

Kim Jung Nissen

Martin Midtgaard

Peter Heino Bøg

Indhold

1	Indledning	7
1.1	Initierende problem	7
2	Problemanalyse	8
2.1	Samfundsmæssige hensyn	9
2.1.1	Grunden til at lave spil	9
2.1.2	Aldersgrænser	9
2.1.3	Ulemper ved computerspil: Psykisk indvirkning	9
2.1.4	Fordele ved computerspil	10
2.2	Diskussion om AI	11
2.2.1	Generelt om AI	11
2.2.2	AI principper	12
2.2.3	Forskellige spiltypers AI	16
2.2.4	AI i spil	22
2.3	AI kontra Grafik	25
2.4	Målgruppe	26
2.4.1	Spørgeskema	26
2.4.2	Resultater	28
2.4.3	Valg af spiltype	29
3	Problemformulering	30
3.1	Probleformulering	31
3.2	Projektafgrænsning	32
4	Metode	33
4.1	Problemanalyse	34
4.1.1	Dataindsamlingsmetoder	34
4.2	Produktudvikling	35
4.2.1	Faser	35
4.2.2	Metodevalg	35
5	Kravsifikation	38
5.1	Funktionelle krav	39
5.1.1	Brugergrænsefladen	39
5.1.2	Display	39

5.1.3	Input fra brugeren	39
5.1.4	AI	40
5.1.5	Omgivelser	41
5.1.6	Gameplay	41
5.2	Ikkefunktionelle krav	42
5.2.1	Platform	42
5.2.2	Skalerbarhed	42
5.2.3	Brugervejledning	42
5.2.4	Samtidighedsproblemer	42
5.2.5	Performance	43
5.2.6	Åbenhed	43
5.3	Prioritering af de funktionelle krav	44
6	Analyse	45
6.1	Struktur	46
6.2	Spil idé	46
6.3	Samtidighed	47
6.4	Fuzzy logic	47
7	Design	48
7.1	Spilleregler	49
7.2	Real-time	50
7.3	Spiltilstande	51
7.4	Klient-server arkitektur	53
7.5	Vejfinding	54
8	Implementation	57
8.1	Introduktion til valgte teknologier	58
8.1.1	C++	58
8.1.2	SDL	58
8.1.3	OpenGL	58
8.1.4	OpenAL	58
8.1.5	Afrunding	58
8.2	Centrale funktioner	59
8.2.1	Kernen	59
8.2.2	Father-objektet	59
8.2.3	Mother-arrayet	60
8.2.4	Rabbit-objekt	61
8.2.5	Grafik	62
8.2.6	Kollision	63
8.3	Algoritmer til AI	64
8.3.1	Ændringer i tilstande	64
8.3.2	Vejfinding	64
8.3.3	Jagt	67
8.4	Debug	68
8.5	Gennemgang af produktet	69
8.5.1	Generelt	69
8.5.2	Head Up Display	69

8.5.3	Styring	69
8.5.4	Powerups	70
8.5.5	Våben	70
8.5.6	Modstandere	70
8.5.7	Omgivelserne	70
8.6	Vurdering	71
9	Test	73
9.1	Offentlig test	74
9.2	Målgruppetest	74
9.2.1	Sådan blev der testet	74
9.2.2	Diskussion af resultater	75
9.3	Evaluering	75
10	Konklusion	76
10.1	Konklusion	77
10.2	Perspektivering	78
11	Litteratur	80
11.1	Spilliste	82
11.2	Kildekritik	83
12	Bilag	84
12.1	Ordliste	85
12.2	Bilag 1 - Spørgeskemaundersøgelse	87
12.3	Bilag 2 - Resultater fra spørgeskemaundersøgelse	90
12.4	Bilag 3 - Beta-email	93
12.5	Bilag 4 - Spørgsmål og resultater fra målgruppetest	94

Kapitel 1

Indledning

I dag er computerspil en stor del af hverdagen for mange, og disse computerspil skal både være underholdende og udfordrende. Spillenes computerstyrede modstandere skal derfor give spilleren en fornemmelse af, at der er kamp til stregen, men alligevel formår at vinde over computeren. Her kommer kunstig intelligens (AI) ind i billedet. Formålet med AI er at få et objekt til at opføre sig på en intelligent måde. En god AI kan fx være en troværdig modstander i et computerspil.

Til enhver tid er en menneskelig modstander mere uforudsigelig end en computerstyret modstander, da computeren ikke er udstyret med de menneskelige træk som fx intuition og fornuft. En god AI bør derfor opføre sig menneskeligt, fx ikke bare blive ved med at løbe ind i en mur. Fremstillingen af en troværdig AI kan betragtes som et større problem, der kan inddeles i mindre delproblemer: vejfinding, taktik, færdigheder og læring.

I dette projekt bliver der kigget på forskellige aspekter indenfor AI i spil. Der diskuteres forskellige principper, teorier og algoritmer bag AI og spil. Det endelige mål med projektet er at producere et spil med en troværdig kunstig intelligens.

1.1 Initierende problem

AI er en vigtig del af computerspil. Selvom AI er blevet bedre igennem årene, så er der stadig spil, hvor computermodstanderne opfører sig uintelligent. Derfor vil der i dette projekt blive kigget nærmere på udvikling af en kunstig intelligent modstander i spil.

Hvordan bruges kunstig intelligens i spil?

Hvordan kan kunstig intelligens implementeres i et specifikt spil?

Hvilke samfundsmæssige hensyn er der at tage stilling til, når et spil designes?

Problemanalyse

I problemanalysen vil der først blive overvejet de samfundsmæssige hensyn, der skal tages op for at producere et spil. Herunder vil der blive set på, om det er psykisk skadeligt at spille meget computerspil, og hvad forskellige forskere mener på dette område.

Derudover vil der gennem problemanalysen blive undersøgt, hvad AI består af, og hvordan AI til spil kan fremstilles. Ydermere vil der blive kigget på forskellige delproblemer i AI.

For at få indblik i målgruppens meninger omkring de emner, der skal tages højde for i produktet, vælges at sende et spørgeskema ud til de studerende på det Teknisk-Naturvidenskabelige Basisår og Samfundsvidenskabeligt Basisår på Aalborg Universitet. Resultaterne kan bruges til at give et indblik i, hvilke AI aspekter der skal prioriteres højest, samt hvilke spilgenrer der foretrækkes. Analysen munder ud i en problemformulering og efterfølgende en afgrænsning.

Projektmetoder, der er anvendt til problemanalysen, er beskrevet i kapitel 4.

2.1 Samfundsmæssige hensyn

I dette afsnit vil grunden til at lave spil, aldersgrænser for spil, spillenes psykiske indvirkning på mennesker og fordele ved computerspil blive diskuteret. Afhængigt af hvor et spil skal udgives, er dette emner, der skal tages stilling til, inden der startes på udviklingen. Spil, der er lovlige at udgive til fx børn under 18 år i Danmark, er ikke nødvendigvis lovlige at udgive til samme målgruppe i andre lande, se afsnit 2.1.2.

2.1.1 Grunden til at lave spil

Der kan være mange grunde til at udvikle computerspil, og der kan være lige så mange grunde til at spille dem. En af grundene til at spille er ofte at underholde sig selv. Andre spiller spil professionelt, dvs. at de tjener penge på at spille computerspil som sport[1]. Dette er dog stadig meget få mennesker. Nogle spiller for at få spændingen ved at udfordre andre eller sig selv. Det er grunde som disse, der gør, at der er et marked for spil.

2.1.2 Aldersgrænser

I Danmark er der ingen lov om salg af computerspil til børn. I modsætning til England kan børn i Danmark købe spil, som er uegnet for børn under 18 år. GTA Vice City[⊗] er et eksempel på et voldeligt spil, der er uegnet for børn under 18 år, men som stadigvæk kan købes af alle i Danmark.[2]

Danmark er medlem af PEGI (Pan European Game Information), som er ansvarlig for mærkninger af computerspil. PEGI er et system oprettet af branchen selv, der laver klassifikation af computer- og videospil, som kan vejlede købere. PEGI laver ikke bedømmelser af sværhedsgraden i de enkelte spil, men udelukkende af spillets indhold, som fx om spillet er voldeligt, skræmmende eller om der blive brugt stødende sprog. I Danmark skal alle computerspil være mærket med et PEGI mærkat, som angiver en af aldersgrænserne 3+, 7+, 12+, 16+ eller 18+. Derudover kan der være mærket, om spillet fx indeholder voldselementer eller stødende sprog. I Danmark er mærkaterne kun til vejledning for købere og ikke til aldersbegrænsningen af spillet. På figur 2.1 kan ses et eksempel på PEGI mærker.[2]



Figur 2.1: Spillet F.E.A.R.[⊗] har PEGI mærkerne “18+”, “voldeligt” og “stødende sprog”.

2.1.3 Ulemper ved computerspil: Psykisk indvirkning

Der er mange forskellige holdninger til, hvorvidt computerspil er skadelige for spilleren. Der er blevet lavet mange undersøgelser, hvor forskere har forsøgt at vise, om spil er psykisk skadelige. Der er blevet fundet mange forskellige resultater, som ikke giver et entydigt svar. Stridighederne, om hvorvidt computerspil har en psykisk indvirkning, er hovedsageligt baseret på at se problemstillingen ud fra det kulturvidenskabeligt perspektiv eller effektforskningens perspektiv.[3]

Kulturvidenskabeligt perspektiv

I Danmark er det det kulturvidenskabelige perspektiv, der dominerer. Kulturvidenskabelige forskere henter bl.a. deres inspiration fra den humanistiske medievidenskab. Her lægges vægt på, at spillerens oplevelse af spillet er afhængig af omgivelserne og miljøet, der spilles i. I dette perspektiv er det ikke muligt at undersøge skadeligheden ved voldelige computerspil i et laboratorium, da dette adskiller sig drastisk fra den måde spillene faktisk bruges på. De prøver blot at finde ud af, hvad der sker med personen, når vedkommende spiller meget computer.[3]

Forskerne har ikke tendens til at generalisere ud fra de undersøgelser, der er foretaget. Desuden foretages der ikke eksperimenter, som kun er baseret på målbare data. Derimod kigges der på, hvilke betydninger spilmediet har for de givne personer. Forskerne undersøger dataene som en helhed, fordi de mener, at omgivelserne og miljøet er en stor faktor, samt hvilken rolle spilmediet har.[3]

Effektforskningens perspektiv

I lande som USA og England er det effektforskningens perspektiv, der dominerer. Effektforskningen har sin baggrund i den lægevidenskabelige og psykologiske verden. Forskerne prøver normalt at undersøge et spørgsmål ved at opstille eksperimenter, hvor personerne udsættes for forskellige typer spil. Dette sker under meget kontrollerede, men ikke så realistiske forhold.[3]

Her bliver forsøgene lavet i et kontrolleret miljø, hvor deres resultater bliver baseret på målbare data. Denne forskningsretning har en tendens til at konkludere, at computerspil er skadelige for børn og unge. I de forsøg, der bliver foretaget, tages der kun udgangspunkt i målbare data, som fx puls, stemme og kropstemperatur. Der forsøges ikke at kigge på andre faktorer såsom miljøet og omgivelserne.[3]

Disse to forskningsretninger er meget uenige, om hvorvidt det er skadeligt at spille. De kulturvidenskabelige forskere mener enten, at det ikke er til at sige, om det er skadeligt, eller at spørgsmålet er for generelt til at svare på. Effektforskerne mener derimod, at det er skadeligt at spille computerspil. Der kan derfor konkluderes, at der ikke er noget entydigt svar på, om computerspil er skadelige for børn og unge.[3]

Undertegnede er af den opfattelse, at det at spille computerspil ikke er skadeligt. Dette bliver blandt andet bygget på stor erfaring med computerspil. Selvom der kommer højtråbende kommentarer er det aldrig fortsat efter spillet. Hvad der sker i spillet, bliver i spillet. Derimod er der oftest efterfølgende blevet snakket med godt humør omkring situationer fra spillet.

2.1.4 Fordele ved computerspil

Mange tror, at det kun kan være skadeligt at spille computerspil, men der kan også være fordele ved at spille computerspil. Ofte forbinder forældre computerspil med, at spilleren isolerer sig fra omverdenen. Dette er dog ikke altid tilfældet, da mange børn bruger computerspil til at skabe et socialt netværk. Børnene bruger også computere og computerspil som et legeredskab, da de ofte sidder flere sammen og spiller. De behøver dog ikke sidde ved samme computer for at spille sammen, da onlinespil giver dem mulighed for at spille sammen uden fysisk at være sammen. Gennem online spil kan der således skabes et socialt netværk.[3]

Selvom mange tror, at computerspil kun er en underholdningsform, bruges spillene til meget mere end det. Ved at spille forskellige spiltyper, træner børnene forskellige evner, som fx reaktionsevne, logik og overblik. Mange spil i dag er på engelsk. Dette hjælper børnene med at lære og forstå det engelske sprog.[3]

Ud fra dette kan vi konkludere, at børnene, udover underholdning, får lært nye sprog eller forbedre deres evner ved at spille computer. Blandt andet skabes et socialt netværk, og nogle sanser kan endvidere skærpes.

2.2 Diskussion om AI

I de følgende afsnit vil forskellige aspekter af begrebet kunstig intelligens blive diskuteret. Der vil blive diskuteret generelt om, hvad kunstig intelligens er, hvad kunstig intelligens bliver brugt til og teknikker om, hvordan en computer kan lære af menneskelig interaktion.

For at få et større overblik over begrebet AI i spil, er det i denne rapport inddelt i kategorierne: vejfinding, læring, taktik og udnyttelse af færdigheder. Disse kategorier kan ses som selvstændige problemer og derved undersøges og løses hver for sig. I sidste ende skal løsningen til disse delproblemer kunne samles som en overbevisende AI til et spil. I de følgende afsnit i rapporten vil de vigtige elementer indenfor hver af disse kategorier blive gennemgået. Der vil også blive kigget på, hvilke spiltyper der findes, og hvilken slags AI disse har.

2.2.1 Generelt om AI

AI, kunstig intelligens, er en metode til at få et ikkeintelligent objekt til at virke intelligent. En succesfuld AI ville virke så intelligent, at der ikke ville blive lagt mærke til, at det var et ikkeintelligent objekt, der blev iagttaget.[4]

Der findes flere steder, hvor der er mulighed for at bruge AI. Der kan fx kigges på hjemmet, hvor der i dag allerede findes AI i husholdningsredskaber. Dette er en stor hjælp, da ejerne ikke skal tænke på fx at støvsuge, men hvor der er en lille robotstøvsuger, som sørger for, at der bliver støvsuget. Den kan måske lære, hvor de forskellige møbler står henne, og derved udregne en "støvsugebane" til næste gang, så det går hurtigere med at støvsuge. Dette er også overført til en græsslåmaskine, hvor det er samme principper, der går igen.

Generelt for de fire kategorier inden for AI i spil (vejfinding, taktik, færdigheder og læring), skal der yderligere tages stilling til emner som snyd og betænkningstid.

Der findes forskellige måder at få AI til at virke intelligent på. I forskellige tilfælde kunne det være en ide at overveje, hvorvidt der ville blive bedre AI ved at "snyde". Med snyd tænkes på, at modstanderen, altså computeren, kan drage fordel af viden om spilforløbet, som spilleren ikke har mulighed for at have. Fx vil modstanderen altid kende spillerens koordinater i spillet, hvorimod spilleren ikke har denne information om modstanderen.

Betænkningstid svinger meget inden for forskellige typer spil. Fx vil turbaserede* spil oftest allokere flere ressourcer til AI-delen, og give den rigeligt med tid til at udtænke næste træk. I real-time* spil derimod, har AI-delen ikke lige så mange ressourcer, da disse også skal bruges af spilleren. Yderligere er det også oftest nødvendigt, at AI-delen nærmest ingen betænkningstid har, da der skal reageres øjeblikkeligt.

Opsummering

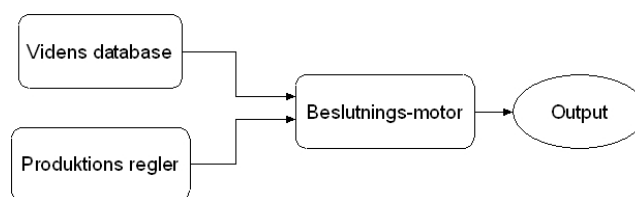
AI kan bruges til mange ting, som tidligere skrevet - bl.a. hjælpemidler. Disse hjælpemidler kunne bruges til fx ældre mennesker, handikappede og til hjælp i hjemmet. Så det er ikke kun i spil, der kan bruges AI. Det er dog ikke sikkert, at den AI, der kommer i det endelige produkt, vil kunne bruges andre steder end netop der.

2.2.2 AI principper

Der findes forskellige måder, hvorpå der kan laves AI. Dette kommer helt an på, hvad denne AI skal kunne. De forskellige måder AI kan realiseres på, bliver ofte delt op i tre forskellige kategorier[5], som er følgende:

- Ekspertsystemer
- Fuzzy Logic-systemer
- Neurale netværk

Ekspertsystemer

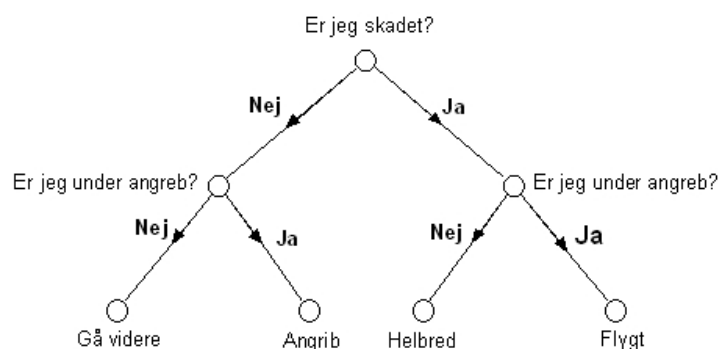


Figur 2.2: I et ekspertsystem giver beslutningsmotoren et output ud fra inputs fra en vidensdatabase og produktionsregler.[6]

Et ekspertsystem indeholder en vidensdatabase, hvor der er en masse informationer, som kan bruges til at løse et problem. Derudover er der produktionsregler, som basalt består af et stort antal *if-else* sætninger.[6]

Ved hjælp af spørgsmål, hvor der er fastlagte svarmuligheder, kan et ekspertsystem bruge udelukkelsesmetoden til at finde frem til et endegyldigt svar. Et ekspertsystem kan dog kun løse opgaver, hvis vidensdatabasen indeholder viden om den pågældende opgave. Hvis systemet møder et spørgsmål, som ligger uden for dens vidensdatabase, er der ingen mulighed for, at systemet kan finde en løsning.[6]

Den logiske tankegang bag et ekspert system kan opstilles som et rooted træ. Dette kan gøres, fordi der kun er fastdefinerede svarmuligheder til hvert spørgsmål, se et eksempel på figur 2.2. AI kan med andre ord ikke foretage handlinger, der ligger uden for det, af programmøren,

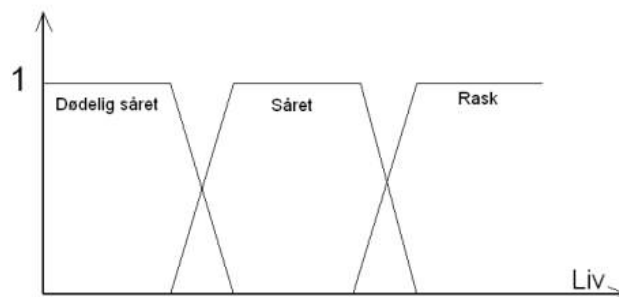


Figur 2.3: Et eksempel på et ekspertsystem, hvor en AI overvejer, hvorvidt den skal angribe på baggrund af, om den er skadet og/eller under angreb.

eksplicite foruddefinerede handlingsmønstre. Når træet er nået til ende, vil systemet returnere en værdi. Denne værdi kan fx være en procentsats for, hvor sandsynlig en diagnose er.[6]

Fuzzy logic systemer

“Fuzzy logic” er en upræcis logik til AI, hvor svarmulighederne ikke er begrænsede, som det er i ekspertsystemer. Det er smart, da det ikke altid er muligt at svare indenfor de fastlagte rammer. Hvis et spørgsmål med to svarmuligheder skal besvares af et fuzzy system, bliver svaret en blanding af begge. Skal der svares på om en person er syg eller rask, kan det være svært at sige præcist. Men med fuzzy logic ville svaret være, at personen fx var 15% syg og 85% rask. Det kan være svært at se, at en computer kan bruge dette resultat til noget, men her kan der laves en “defuzzifikation”, hvor systemet prioriterer de forskellige typer og derudfra beregner en værdi. Denne værdi bestemmer den handling, der skal foretages. Som eksempel kunne det være, om personen skulle sendes på hospital eller ej.[6]



Figur 2.4: En illustration af en “fuzzy” inddeling af liv med tre trapetzer.

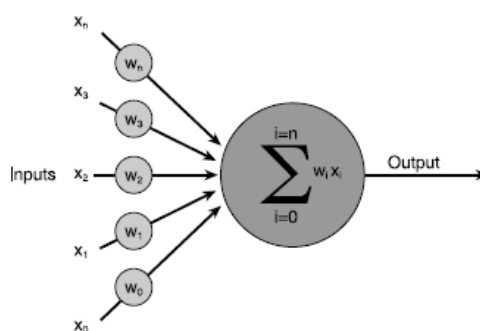
De forskellige faktorer kan visualiseres som overlappende trapetser i et koordinatsystem. Et eksempel på dette kan ses på figur 2.4. Trapetserne skal symbolisere de værdier, som de forskellige svar skal have. På 1. akse er værdien for det, der skal måles på, og på 2. akse er der den værdi, som de forskellige former for svar får. Fx kunne der laves en udregning ud fra, hvor meget liv spilleren i et skydespil har tilbage. Her kunne der være tre stadier: rask, såret og dødeligt såret. Her beskrives de tre stadier som tre trapetser sådan, at hvis spilleren har meget i liv, så er spilleren rask. Mistes der noget af livet, kan spilleren komme i overgangen mellem to stadier, hvor spilleren kan være både rask og såret på samme tid (dog med forskellige værdier). Der ville på denne måde kunne laves “grafer” for hver enkelt faktor, som AI skulle tage højde for, fx ammunition, penge eller opbakning.[6]

Fuzzy logic bruges fx til at forudsige vejret. Her kan bruges inputs som fx temperatur og luftfugtighed. Disse inputs hjælper fuzzy systemet med at forudsige, hvordan vejret måske vil udvikle sig.

Neurale netværk

Princippet for hvordan neurale netværk er opbygget, er inspireret af den måde den menneskelige hjerne fungerer. Den menneskelige hjerne består af ca. 50 milliarder neuroner, der hver især kan beregne og behandle givne informationer. Hjernen bruger kun 5-10% af disse, så ved en effektiv realistisk computersimulering, ville dette også være tilstrækkeligt. Dog bliver der oftest kun brugt et par tusind neuroner i et neuralt netværk i stedet for flere milliarder.[5]

Generelt gribes problemet, i den datalogiske verden, an ved at se på hver enkelt neuron for sig, med n input og 1 output. Hver neuron får n vægtede input i form af flydende værdier, hvilke dens ene output så afhænger af. Neuronens output kan ses som - afhængig af summen af input-værdierne - “1”, hvis værdien er over en bestemt grænse, ellers “0”. [7]



Figur 2.5: Illustration af en kunstig neuron, som får flere inputs med forskellige vægte ind for derefter at give et output.[7]

På figur 2.5 er x_i input, der enten er "0" eller "1". Hvert input har så en vægt w_i . Neuronen beregner herefter summen af $x_i * w_i$ for herefter at sammenligne med den førnævnte, interne, grænse i neuron. Hvis den totale sum er over grænsen, bliver output "1", ellers "0".

Et eksempel på den beregning, der foregår i neuron, kan ses på nedenstående tabel. Her bliver den totale sum 1.1, så hvis grænsen fx var 1.0, ville output være "1".

Input	Vægt	Input x Vægt	Løbende total
1	0.5	0.5	0.5
0	-0.2	0	0.5
1	-0.3	-0.3	0.2
1	0.9	0.9	1.1
0	0.1	0	1.1

Når massevis af disse neuroner kombineres, kaldes dette et neuralt netværk. Læring kommer ind i billedet, når vægtene og grænserne kan justeres løbende. Dette foregår ved at tjekke, om outputtet stemmer overens med, hvad der er ønsket - hvis ikke, justeres vægte og/eller grænser.[7]

Neuronerne i et neuralt netværk inddeles i lag, oftest tre. Et input-lag, et output-lag, og et antal skjulte lag imellem disse. Outputtet fra hver neuron, i hvert lag, forbindes til alle neuroner i det næste lag, indtil output-laget, hvor det endelige output er blevet genereret.[7]

Neurale netværk benyttes ofte til mønstergenkendelse, da dette er et sted, hvor læringen kan udnyttes. Netværket oplæres til at genkende de forskellige mønstre, som fx bogstaver.

I spil kan neurale netværk give egenskaber, som fx læring.

Afrunding

Hvis et problem skal løses med kunstig intelligens, og det kan deles op i regler, hvor alle mulige situationer er lette at definere, er eksperter-systemer at foretrække. Ulemper ved eksperter-systemer er, at de er afhængige af præcise, opdaterede, regler. Anvendelsesmuligheder for eksperter-systemer, kunne være: strategi, planlægning og undervisning.[8]

Neurale netværks styrke er derimod mønstergenkendelse, som det er meget svært at finde præcise regler for. Neurale netværk kræver dog, at de er blevet "trænet". Ligesom den menneskelige hjerne, anvender neurale netværk også princippet med sammenkædede neuroner. Andre anvendelsesmuligheder for neurale netværk er: finansielle forudsigelser, foretningsbeslutninger, person-genkendelse og medicinsk diagnose.[8]

Fuzzy logic-systemer behandler, som neurale netværk, upræcis data - hvilket vil sige, at der er flydende grænser for, hvornår forskellige stadier nås. Fuzzy logic-systemer bruger også, som

ekspertsystemer, regler til problemløsningerne. Anvendelsesmuligheder af fuzzy logic indbefatter: kontrol af undergrundsbaner, elektroniske systemer til biler, forbrugerelektronik og andre apparater.

Efter gennemgangen af de forskellige AI principper, vælges AI princippet fuzzy logic, til at være den kunstige intelligente beslutningstager i det endelige produkt. Dette skyldes, at fuzzy logic giver en god mulighed for at lave en dynamisk AI, som kan have forskellige opførelser alt efter, hvilken situation modstanderen er i. Et eksempel kunne være, at en modstander bliver skudt og mister liv. I fuzzy logic tjekkes der, om modstanderens liv er under en fastsat grænse. I så fald skal modstanderen flygte eller leder efter powerup, der giver liv.

2.2.3 Forskellige spiltypers AI

Herunder vil AI i forskellige typer af spil blive diskuteret. Formålet er at adskille disse spiltypers AI, så der senere kan findes frem til den type, som vil blive brugt i dette projekt.

I spillenes barndom var det ikke svært at inddele spil i få typer, men efterhånden er der kommet flere og flere undertyper. Desuden har mange spil blandet diverse typer sammen og dannet nye. I dette afsnit vil der blive gennemgået fem overordnede typer af spil, og hvordan deres AI fungerer. Disse typer er action, rollespil, strategi, brætspil og simulation. Gennemgangen indeholder først en forklaring af den aktuelle type og derefter, hvordan AI bruges i denne, samt hvad målene og successkriterierne er. De fleste af de undersøgte spiltyper kan både foregå i 2D og 3D, derfor vil der først i dette afsnit blive diskuteret forskellen mellem 2D og 3D.

2D versus 3D

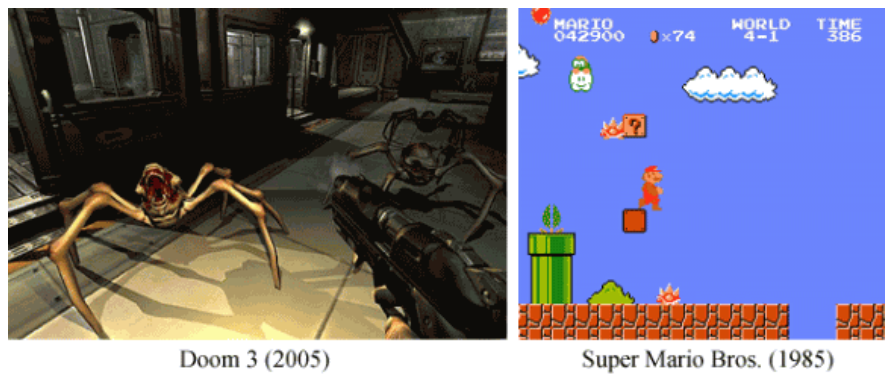
Der bruges forskellige termer, når der ses på det grafiske, og hvordan dette er bygget op. Her ses på 2D og 3D. Et 2D-spil kunne fx være et platformspil*, hvor spilverdenen ses fra siden, og spilleren bevæger sig horisontalt og vertikalt, og billedet eller banen flytter med. Hvis der ses på det grafiske, ser brugeren det kun fra siden, og alt er "fladt" på skærmen. 2D kan også bruges til spil hvor karakteren ses oppefra, og styres rundt på banen. Her er alt også "fladt" på skærmen, men det hele ses oppe fra, i stedet for at det ses fra siden.

I 3D er alt ikke "fladt", men derimod er der mulighed for at kunne komme et niveau højere op på banen eller længere ned, fx op på et tag eller ned i en kælder. Banen er altså i 3D, hvor spilleren stort set kan bevæge sig i tre dimensioner, dog inden for banens begrænsning. Der er mulighed for at kigge op eller ned, da der er flere niveauer i banen. Denne mulighed for at se op og ned, er der ikke i 2D, da alt er fladt og kun på et niveau.

Hvis der ses på AI i disse sammenhænge, så har denne flere muligheder og skal bruge flere ressourcer på at finde ud af, hvad den skal foretage sig, hvis banen er lavet i et 3D-univers. I stedet for planet, skal AI'en nu arbejde i rummet. Her er der mulighed for at kunne komme niveauer højere op, og ikke kun bevæge sig frem og tilbage, samt til siderne. I 2D er der muligheden for at inddele banen i et gitter, hvilket kan gøre det lettere at lave vejfinding. Det er der ikke på samme måde i 3D. Generelt bliver der meget mere at tage højde for i 3D og beregningerne bliver mere tidskrævende, da de skal tage højde for den 3. dimension.

I et platformspil, som fx Super Mario Bros.[⊗], kan modstanderne kun bevæge sig til siden og op og ned, så den skal ikke bruge så mange ressourcer, som hvis spillet var i 3D.

Når der snakkes om dimensioner i et spil mht. AI, så er det vigtigt at skelne mellem den visuelle del, og selve opbygningen af spilverdenen. Selvom spillet er visualiseret i 3D, kan spillet stadig teknisk set foregå i en 2D-verden. Med det menes, at alle bevægelser foregår i 2D, fx som i det gamle spil Wolfenstein 3D[⊗]. Her kan de samme taktikker også benyttes til AI i 2D spil.



Figur 2.6: Skærbilleder fra 3D-spillet Doom 3 (tv), og 2D-spillet Super Mario Bros. (th).

Action

I actionspil går det ofte ud på, at der skal skydes vej igennem en masse modstandere. Der sker hele tiden noget på skærmen, så der er sjældent lange ventetider for spilleren. I selve spillene er der ofte nogle missioner, der skal gennemføres. I disse missioner er der masser af modstandere, våben og andre former for objekter, som kan interagere med spilleren.

AI varierer fra spil til spil, da det ikke er i alle actionspil, hvor modstanderne skal kunne det samme. I nogle action spil skal modstanderne jage spilleren og forsøge at dræbe denne. Modstanderne stormer frem, og kæmper så godt som muligt. Et eksempel på et spil af denne type er Serious Sam[®].

I andre actionspil “tænker” modstanderne. De har mange flere muligheder for at udnytte fx terrænet eller svagheder, som spilleren har. Modstanderen har mulighed for at kunne søge dækning og udøve andre former for intelligente handlinger - dette ses fx i spillet F.E.A.R.[®]. En anden mulighed er, at modstanderne kan arbejde sammen, og måske udnytte de forskellige styrker, som hver modstander har.

Succeskriterierne for denne type AI er at få udslettet spilleren, og forhindre spilleren i at komme videre. Et andet succeskriterie kunne være, at modstanderne kan samarbejde taktisk.



Figur 2.7: Skærbillede fra action-spillet Serious Sam II.

Rollespil

Gameplayet i rollespil bygger på, at spilleren går rundt i en virtuel verden med en figur og angriber diverse modstandere. Hvert angreb udregnes efter terningekast-princippet, forstået på den måde, at der indgår en vis form for tilfældighed. Et spil som Dungeons & Dragons Online[®] bruger dette bogstaveligt. Spilleren kan forbedre sine chancer ved at få nye våben og udstyr eller stige i niveau og dermed blive bedre. At spilleren bliver bedre og bedre er en nødvendighed, da modstanderne også løbende bliver stærkere. Modstandernes opgave er udelukkende at angribe spilleren, så vedkommende kan få erfaring samt diverse genstande, som fjenden kan tabe.

AI i rollespil behøver ikke at være speciel effektiv for at udfordre spilleren. Dens primære opgave er som sagt udelukkende at angribe spilleren. I de fleste rollespil kan modstanderne ikke ret meget mere, men efterhånden kommer der flere og flere eksempler på speciel AI i rollespil, som fx det at flygte. Ofte kan nogle specielle fjender dog udføre nogle specielle handlinger, såsom at lave et meget specielt angreb eller få verden omkring til at deltage i kampen. Disse er ofte de såkaldte "boss'er", altså meget stærke fjender der kæmpes imod som en afslutning på en opgave. Disse specielle handlinger er foruddefineret AI.

Der er eksempler på meget avanceret AI i rollespil, hvor NPC'erne* i spillet udfører deres daglige opgave af sig selv, blandt andet det at spise, arbejde og snakke med andre folk de møder på gaden. Dette er ikke en nødvendighed for at lave et godt og udfordrende rollespil, men derimod en måde at gøre spiloplevelsen bedre på, således at spilleren virkelig føler sig som en del af spilverdenen.

Succeskriterierne for en AI i et rollespil er forholdsvis simple. Fjenden skal kunne finde hen til spilleren. Derefter skal den angribe spilleren, indtil den selv eller spilleren dør. Ydermere skal modstanderen kunne bruge sine forskellige færdigheder på den rigtige måde og på de rigtige tidspunkter. Det nytter fx ikke noget, at modstanderen helbreder sig selv, når der ikke er nogen grund til det. Et andet succeskriterie er, at modstanderen skal opføre sig troværdigt. En hund skal altså opføre sig som en hund, medmindre universet, som spillet foregår i, ændrer på vores grundlæggende verdensopfattelse.



Figur 2.8: Skærmbilleder fra rollespillene Neverwinter Nights[®](tv) og The Elder Scrolls IV: Oblivion[®](th).

Strategi

I strategispil går det hovedsageligt ud på, at en base skal bygges op, ressourcer skal indsamles og enheder, som kan bruges til at angribe eller forsvare sig med, skal konstrueres. Derudover gælder det som oftest om, at fjendens base skal ødelægges, hvilket gør at spilleren vinder. Det hele foregår i real-time, derfor bliver genren ofte kaldt RTS*, eller Real-Time Strategy. Spilleren styrer sine enheder og base uden at skulle vente på, at modstanderen blive færdig med sit træk. Der kan flyttes rundt med enhederne og bygges, som der ønskes. Der er ofte flere forskellige racer at vælge i mellem, med hver deres stærke og svage sider, som kan udnyttes af spilleren, men også af modstanderen.

AI i RTS-spil skal være i stand til at tage beslutninger om, hvilke enheder, der skal flyttes, eller hvilke ting, der skal bygges løbende og i nuet. Modstanderen skal derfor kunne tage beslutninger i forhold til, hvad der sker i spillet. Hvis modstanderen fx bliver angrebet af kampvogne, skal den vide, at infanteri med panserværnsraketter, kampvogne eller måske fly med panserværnsvåben, vil være det bedste valg til at ødelægge angriberens kampvogne. Modstanderen skal kunne træffe de rigtige beslutninger i de givne situationer.

Under spillet vil der være mange forskellige enheder, hvor hver enkelt har deres styrker og svagheder. Modstanderen skal kunne producere en god hær, hvor der er en god blanding af enheder, således at hæren har mulighed for at kunne klare sig i næsten alle situationer. Der gælder de samme sejrskrav for AI'en som det gør for spilleren, da spillene oftest går ud på at udslette hinanden.

AI'en er en succes, hvis den kan finde ud af at udnytte de taktiske muligheder som banen, den race den spiller og de enheder, den har til rådighed, giver den. Den skal kunne udføre taktiske angreb, hvor den udnytter alt dette, og i sidste ende vinde spillet.

Der er dog en afvigelse fra RTS, som bliver kaldt TBS. TBS står for Turn-Based* Strategy, som i store træk er det samme som RTS, men blot hvor der flyttes på tur. Her skal AI'en ikke tænke hele tiden, men den får besked om, hvornår den skal tænke og flytte sine enheder.

Målene for AI'en i et TBS-spil er stort set de samme som AI'en i et RTS. Modstanderen skal udslette sine modstandere for at vinde.



Figur 2.9: Skærmbilleder fra strategispillene Dune II[®](tv) og Command & Conquer Generals: Zero Hour[®](th).

Brætspil

Brætspil er spil såsom skak eller backgammon. Her er der nogle simple regler, som alle skal følge, fx kan brikker kun flyttes til bestemte positioner på en bestemt måde, eller der skal slås med en terning og rykkes det antal, som øjnene viser. Det essentielle ved brætspil er, at spillet går på tur mellem spillerne, hvilket gør, at computeren får at vide, hvornår den skal tænke, og har derfor mulighed for at kunne bruge de krævede ressourcer til trækket. Dette kan ses i forhold til hvis det var i real-time, hvor AI'en kun får tildelt et vis antal ressourcer til at "tænke" med.

Formålet med AI'en er at finde et godt træk, som kan give spilleren noget at tænke over. Afhængig af spillet kan modstanderen tænke flere træk frem i spillet og derefter vælge det træk, der er bedst for den selv. Den kan også lære af sine erfaringer, hvis den nuværende situation ligner noget, den har set før. Hvis det gik skidt sidste gang, skal den prøve et andet træk næste gang.



Figur 2.10: Skærbillede fra brætspillet Chessmaster 10th Edition[⊗].

Simulation

Når der ses på simulationsspils-genren, er der mange forskellige undergenrer. Her kan nævnes genrer som bilspil (fx Need for Speed[®]), flyspil (fx Flight Simulator[®]) og konstruktionsspil (fx Sim City[®]). I bilspil gælder det oftest om at køre gennem en bane og komme først i mål. I flyspil med AI gælder det om at skyde andre fly ned eller gennemføre forskellige missioner. Konstruktionsspil handler om at bygge et imperium op og sørge for, at alt fungerer som det skal, fx at økonomien er i orden. Imperiet kan både være forretningsmæssige imperier, men også bystyrer.

Der er varierende typer af AI inden for hver enkelt undergenre af simulation. Hvis det er et bilspil, skal AI'en kunne køre banen så optimalt som muligt, men samtidig ikke bare køre ind i alle de ikke-statiske forhindringer, som der vil være i form af fx andre køretøjer. Den skal også kunne udnytte bilens egenskaber, som fx hvis bilen er tung i bagenden og skrider nemt, kan dette udnyttes til dens fordel.

I flyspil skal AI'en styre flyet og udnytte de egenskaber som hvert enkelt fly har. Den skal også kunne lave undvigelsesmanøvrer, hvis den bliver angrebet af et andet fly.

Et eksempel på AI i et konstruktionsspil er at holde styr på økonomien, samt at finde ud af, hvad der skal bygges, hvornår, og hvorfor. Dette er for at give en troværdig AI samt en smart modstander.

Succeserne for AI'en i disse former for spil afhænger af, hvad den skal gøre og være i stand til. I et bilspil er det en succes, hvis den kan vinde løbet over den menneskelige spiller. I et flyspil er der fx succes, hvis den kan skyde modstanderen ned og undgå selv at blive skudt ned. Når der skal bygges byer, skal AI'en vide, hvad der skal bygges og hvorhenne. Den skal kunne opfylde eventuelle krav fra "beboeren" i spillet, men den skal også kunne opretholde den økonomiske side af spillet, for at kunne have penge nok til at bygge andre ting.



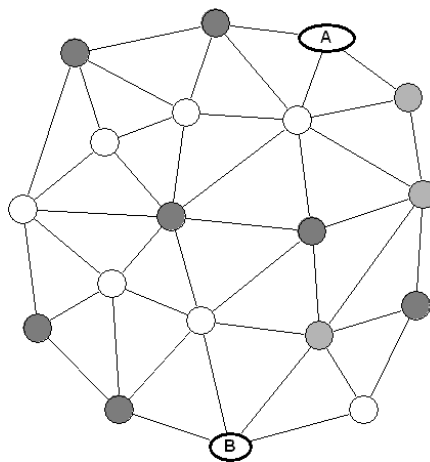
Figur 2.11: Skærbilleder fra simulationsspillene Flight Simulator 2004 (tv) og Grand Turismo 2[®] (th).

2.2.4 AI i spil

Vejfinding

I mange af de singleplayer* computerspil, der i dag udvikles, er der i større eller mindre omfang behov for at finde vej gennem spillet. For at løse denne opgave med at finde vej gennem en bane, kan banen med fordel inddeles i mindre punkter - såkaldte waypoints*. Fra et waypoint vil der være direkte forbindelse til mindst et andet waypoint. Med direkte forbindelse menes en forbindelse uden ugennemtrængelige forhindringer.[9]

Problemet med at finde vej fra et givent startpunkt A til et givent slutpunkt B, via waypoints, kan illustreres som vist på figur 2.12. Den menneskelige intelligens kan relativt hurtigt gennemskue det illustrerede problem og finde den optimale løsning, der i dette tilfælde, foruden punkterne A og B, går gennem tre waypoints (lysegrå) i højre side, se figur 2.12.



Figur 2.12: Eksempel på et vejfindings-problem. Hvide prikker repræsenterer gyldige waypoints. Grå prikker repræsenterer ugyldigt waypoints, hvori der fx befinder sig en forhindring.

Problemet med vejfinding i AI er herved at omskrive denne menneskelige logik til computerinstruktioner. Løsningen vil kunne findes ved at konstruere en algoritme, der med udgangspunkt i ovenstående definition af problemet, tager højde for følgende punkter[9]:

- Statiske objekter der ikke kan passeres.
Statiske objekter er objekter, der aldrig flyttes i spillet, som fx huse, klipper og træer. En rute uden om disse objekter kan beregnes inden ruten påbegyndes.
- Dynamiske objekter der ikke kan passeres.
Dynamiske objekter er derimod objekter, der ikke har en fast position i spillet. Herunder tænkes fx på andre spillere, køretøjer o.l. I modsætning til statiske objekter, kan en rute uden om disse objekter ikke beregnes inden ruten påbegyndes. Efter at ruten er påbegyndt, kan et dynamisk objekt bevæge sig ind på ruten og derved blive en uforudset forhindring.
- At der findes flere løsninger, hvoraf en af de korteste altid skal vælges.
I de fleste tilfælde vil der være nærmest uendeligt mange veje fra A til B, men der vil altid være en eller flere veje, der har den mindste længde, og det er denne løsning, der skal returneres.

Der bør bemærkes, at destinationspunktet B ikke nødvendigvis behøver at være et bestemt punkt. Der kunne også være tale om en mængde af mulige destinationer, hvoraf vejen til den ene destination, der kræver den korteste vej, ønskes.

Udover, at modstanderen skal kunne finde den korteste vej mellem to punkter, er det også vigtigt, at turen mellem punkterne foregår realistisk. I princippet vil modstanderen kunne blive flyttet fra A til B øjeblikkeligt, men dette ville være urimeligt overfor spilleren. Det er vigtigt, at AI'en har samme begrænsninger som spilleren. Yderligere skal der også tages højde for, at det oftest ville se mest naturligt ud, hvis AI-modstanderen vender samme vej som den bevæger sig.

Som et alternativ til metoden med vejfinding via waypoints, der kort blev introduceret ovenfor, er der også andre muligheder. Modstanderen kan selv tolke en simplificeret udgave af selve spilverdenen og derudfra finde gyldige ruter. Her kan der yderligere skelnes mellem, om AI'en opfatter spilverdenen diskret eller kontinuert[10]. I spil ses den diskrete spilverden oftest som gitter-baseret, hvorimod den kontinuerte spilverden er inddelt i den mindste enhed, fx pixels. AI i den kontinuerte spilverden vil som udgangspunkt virke mere realistisk, da den har flere bevægelsesmuligheder, men dette vil være på bekostning af ressourcer. AI i den diskrete spilverden vil derimod kræve mindre ressourcer på bekostning af, at der kan risikeres, at AI'en vil virke for mekanisk.

Læring

Mennesker er i stand til at lære af deres egne fejl. Skal computeren efterligne menneskelig adfærd, og opføre sig intelligent, skal der overvejes, hvorvidt den kunstige intelligens skal være i stand til at lære af sine egne fejl.

En AI modstander i skak kan oplæres af spilleren, således AI'en er i stand til at forbedre sine evner til at spille skak. Hver gang den foretager et træk, som medfører en tabt brik, vil denne registrere dette som et dårligt træk, og derefter undgå at begå den samme fejl. Hvis modstanderen ender i en identisk situation, ved den, at den ikke skal foretage det dårlige træk. I situationer, hvor modstanderen ikke kan undgå at tabe en brik, må denne tage højde for hvilken brik, der er taktisk bedst at tabe. Modstanderen vil derfor blive bedre med tiden. Resultatet vil blive, at computeren i sidste ende er i stand til at slå sin modspiller.

En modstander i et skydespil kunne fx være af ringe karakter i begyndelsen. Senere i spillet, efter at have overvundet en del modstandere, kan computeren lære af, hvordan den undgår at blive skudt. Dette vil bevirke, at spillet vil stige i sværhedsgrad, jo mere spilleren spiller. Læring kan give en dynamisk form for modstand i et spil, til forskel fra at have en statisk kunstig intelligens, som fx foruddefineret AI, se afsnit 2.2.4.

Taktik

At AI'en prøver at benytte en eller anden form for taktik, kan være afgørende for, hvorvidt den i sidste ende vil virke intelligent.

Taktik kan forbindes til læring, da information, om hvad spilleren "plejer" at gøre, kan udnyttes taktisk. Det taktiske element kan computeren først udnytte, når den kan forudsige spillerens næste handling. Hvis den kan forudsige spillerens handlinger, kan den finde et taktisk godt modtræk til dette. På denne måde vil computeren altid være et skridt foran. Men hvis den ikke kan forudsige, hvad spilleren vil gøre, så har den ikke længere mulighed for at spille på denne måde, her bliver den nødt til at spille efter, hvad den "tror" er optimalt.

Et taktisk angreb kunne fx indebære, at AI'en ikke bare tager den korteste vej hen til sin modstander, men måske tager en alternativ længere vej for at lade spilleren gå i et baghold. En udbyggelse til denne taktik, kunne være at sende en mindre afledningsmanøvre ad den korteste vej. Taktiske angreb er ofte bygget på foruddefinerede taktikker, da det er sværere at lære angrebstaktikker ud fra spillers handlinger.

Hvis AI'en skal styre flere enheder, kunne taktikken gå ud på at koordinere de forskellige enheder med forskellige evner, meningsfuldt. Således kunne mindre vigtige enheder agere "skjold", mens de vigtigere enheder holder sig i baggrunden. Det kunne også være en taktik at efterlade enheder til et eventuelt forsvar.

I mange spil er der oftest flere forskellige racer, der kan vælges imellem. Her skal AI'en kunne udnytte de færdigheder og stærke sider, som de forskellige racer giver. Den skal også kunne mindske mulighederne, for at andre kan udnytte de svagheder, som følger med de forskellige racer.

Generelt skal AI'en kunne bruge de taktiske muligheder, som der bliver stillet til rådighed. Ved at kunne udnytte de forskellige muligheder, bliver der derved mere modstand for spilleren, da vedkommende skal begynde at tænke mere taktisk. Hvis AI'en var dårlig til at bruge taktik, kunne spillet hurtigt blive ensformigt, da AI'en vil udføre de samme rutiner. Derfor er det vigtigt, at AI'en kan udnytte dette, således spilleren får en god spiloplevelse.

Perfekt AI?

Et problem, der kan opstå ved at lade computeren beregne de valg, der skal foretages, er, at den på matematiske problemer kan være for nøjagtig, og derved i visse tilfælde være for perfekt. Et eksempel på dette kunne være i et skydespil, der benytter såkaldte hitboxes* på spilleren. Her vil computeren oftest for hurtigt og præcist kunne beregne de korrekte vinkler for at ramme den mest sårbare hitbox på en spiller. Den mest sårbare hitbox er oftest meget lille, netop for at gøre den svær at ramme for spilleren - men dette er ikke nødvendigvis en forhindring for AI'en.

Hvis dette problem opstår under udviklingen af AI'en til et spil, kan det derfor blive nødvendigt at tage højde for. En mulig løsning kunne være at tilfældiggøre en mindre del af den pågældende AI-rutine. Denne tilfældighed kunne også flyttes fra AI'en, til skudet generelt - således at det også gør sig gældende for spilleren. En anden løsning kunne være, at AI'en i stedet for at skyde efter den mest sårbare hitbox, skyder efter den største, fx maven.

Hvis AI'en får lov til at sigte og rammer plet hver eneste gang, vil det være umuligt for den menneskelige spiller, at vinde over computeren.

Foruddefineret AI

Når den normale AI ikke er tilstrækkelig, bruges ofte foruddefineret AI, også kaldet "scriptet". Det er en AI som på forhånd har fået at vide lige præcist, hvad den skal foretage og hvornår. Foruddefineret AI bliver oftest brugt i FPS* og RTS. Eksempler på foruddefineret AI kan være samtaler mellem computerstyrede spillere og det at åbne en dør, så spilleren kan komme videre. Eller det, at der kommer forstærkninger, der med det samme angriber en specifik bygning i RTS. Et andet eksempel kunne være det, at modspilleren vælter et bord og gemmer sig bag det. Førhen ville sådan en handling være foruddefineret og derved ske hver eneste gang spilleren kom dertil. Andre spiltyper kan også have foruddefineret AI, men det er ikke så udbredt, fx bruges det sjældent i brætspil.

I nogle af de nyeste spil er en handling, som at vælte et bord og gemme sig bag dette, indført i AI'en. Derved kan det ske på et hvilket som helst tidspunkt og altid overraske spilleren. Andre handlinger, som at åbne en dør for spilleren, kan være farlige at indføre i selve AI'en og bliver derfor foruddefineret. Grunden til dette er, at hvis nu AI'en ikke vælger at åbne døren, kommer spilleren ikke videre i spillet.

I et singleplayerspil, er der normalt en historie. I løbet af spillet bliver der brugt meget foruddefineret AI for at få historien frem på den måde, der ønskes. Nogle spil har flere slutninger, men ofte er der kun en. Derfor bruges foruddefineret AI også på en måde, så der ikke sker

uventede ting, som kan ødelægge denne slutning. Fx ved, at en person, som skal bruges til sidst, dør.

Afrunding

I afsnit 2.4.3 er det valgt, at spillet skal være et actionspil. I sådan et spil, skal modstanderen kunne bevæge sig rundt, da denne skal angribe spilleren, derfor vil der blive brugt vejfinding.

Udover vejfinding vil modstanderen have forskellige taktiske egenskaber, som denne kan bruge til at slå spilleren. Her vil forsøges at give spilleren den illusion, at modstanderen kan forudsige spillerens træk.

Perfekt AI vil ikke være at foretrække, da denne altid vil kunne slå spilleren. Derved ville spillet hurtigt miste spillerens interesse. Derfor bør AI'en i produktet ikke være perfekt, men derimod kunne begå fejl så som upræcis skydning.

Når der arbejdes med AI, er der mulighed for, at AI'en kan programmeres således, at den er i stand til at lære. Læring er meget komplekst at implementere, derfor vil AI'en i produktet kun benytte begrænset læring.

I produktet vil der ikke være nogen form for foruddefineret AI, da dette ikke er nødvendigt for at få en tilfredsstillende modstand i denne spiltype.

2.3 AI kontra Grafik

Når et nyt spil skal udgives, er det vigtigt for udgiverne, at forbrugerne kender til spillet på forhånd, og at de får et godt indtryk af det. Den letteste måde at få potentielle købere til at lægge mærke til et spil, er ved at vise forbrugerne nogle flotte billeder eller film fra spillet. Det er lettere at huske flotte billeder end fx en tekst, der fortæller, at der er en god AI i spillet.[9]

Der er dog nogle spil, hvor producenterne viser film af, hvor god deres AI er, men det er svært at vide, om det er scriptet eller, om det virkeligt er AI. Hovedsageligt gør spilproducenterne kun opmærksom på, hvad de har gjort af fremskridt inden for det grafiske i spillet. Det kunne være, at de viser bygninger blive ødelagt, eller at spillet er animeret i virkelighedstro 3D.

På spilmarkedet kan der være mange flotte grafiske spil, men hvor gameplayet kommer til at halte, hvis en vigtig AI ikke opfører sig hensigtsmæssig i forhold til den givne situation. Det kan være, at spilproducenterne ikke har nok tid til at kunne udvikle spillet ordentlig færdigt, og de derfor ligger mere vægt på grafikken og forsømmer andre dele af spillet, som måske kunne have gjort det en bedre. Hvis de nu havde længere tid til at udvikle spillet, så kunne de måske også koncentrere sig mere om andet end det grafiske, således at gameplayet blev bedre.

I modsætning til kutymen på det etablerede spilmarked, vil der i dette projekt ikke blive lagt stor vægt på den grafiske del af produktet, men der vil blive lagt større vægt på at få udviklet en troværdig AI og et godt gameplay. Det bliver måske ikke det letteste spil at sælge, men der kan altid laves bedre grafik efter, at de centrale dele til produktet er færdigudviklet.

2.4 Målgruppe

I dette projekt er 18- til 24-årige valgt som målgruppe til produktet.

Spørgeskemaet er udleveret til basisårsstuderende på Aalborg Universitet, da størstedelen af de studerende på universitet er en del af denne målgruppe. Ydermere er det også spil til denne målgruppe, som projektgruppen har størst erfaring med, da medlemmerne selv er en del af den.

2.4.1 Spørgeskema

For at få klarhed omkring målgruppens ønsker til spil og AI, laves en spørgeskemaundersøgelse. Da der på AAU's basisuddannelse er et stort udvalg af studerende, der er en del af den valgte målgruppe, med vidt forskellige interesser og spilerfaringer, bliver spørgeskemaerne udelt der. For ikke kun at få svar fra de, ofte i forvejen, teknisk-interesserede på Det Teknisk-Naturvidenskabelige Basisår ved Aalborg Universitet, sendes spørgeskemaet også ud til de studerende på Det Samfundsvidenskabelige Basisår. Dette giver en større bredde i undersøgelsen, og sørger for, at kønsfordelingen bliver mere lige.

Spørgeskemaet sendes ud i form af en dynamisk hjemmeside. Adressen til hjemmesiden¹ sendes ud til de studerende via deres AAU e-mail-adresse. Da undersøgelsen foregår elektronisk, kan der hurtigt tælles stemmer sammen, hvilket gør, at der uden besvær kan modtages svar fra flere hundrede studerende. Successkriteriet for denne kvantitative undersøgelse er, at de modtagne svar skal være repræsentative for målgruppen. Dvs., at der skal kunne ses tendenser i resultaterne. Spørgeskemaet sendes ud til 1291 studerende. Generelt kan siges, at flere svar giver et så generelt indblik i målgruppens ønsker, som muligt.

At spørgeskemaundersøgelsen foregår over internettet kunne forhindre en del i at deltage, da ikke alle har adgang til internettet. Men da universitetets kommunikation til de studerende i forvejen foregår pr. e-mail, antages det, at størstedelen af de studerende har en eller anden form for adgang til internettet, og at de tjekker den e-mail-adresse, de har tilknyttet universitetet. Der skal dog tages højde for evt. passive studerende og studieinaktive.

Med udgangspunkt i resultaterne fra spørgeskemaundersøgelsen, vil målgruppen kunne præciseres, men der vil også kunne skabes et overblik over, hvordan de forskellige elementer i produktet skal prioriteres. Spørgeskemaet kan ses på bilag 1.

Gennemgang og begrundelse af spørgsmål

Hvilket køn er du?

En hypotese er, at mænd og kvinder ikke er interesseret i de samme spiltyper. Ved at stille dette spørgsmål, kan der efter, at resultaterne er modtaget, ses, hvorvidt der skal være forskel på det endelige produkt, afhængigt af, om der ønskes en kvindelig eller mandlig målgruppe.

Hvilken aldersgruppe tilhører du?

Da produktet ønskes målrettet til unge, hjælper dette spørgsmål til at se på denne gruppes ønsker. Svarene på dette spørgsmål kan endvidere vise, om det færdige produkt uden videre vil henvende sig til en større aldersgruppe end beregnet.

Hvor mange timer om ugen spiller du computerspil/konsolspil?

Det er vigtigt at vide, hvor meget den pågældende person spiller. Hvis personen spiller meget, har personen måske stor erfaring med AI i spil. I modsætning vil en person, der spiller sjældent,

¹<http://www.109.dk/survey/>

kunne beskrive, hvad der kan gøres bedre indenfor AI, før de ville være interesseret i at spille disse spil.

Hvilke spilgenrer foretrækker du?

Afhængigt af, hvilke spilgenrer personen spiller, kan der være forskel på dennes ønsker omkring, hvad de mener AI skal opfylde. Yderligere vil de mest populære genrer også kunne findes ud fra svarene på dette spørgsmål. Derved kan fås et billede af, hvilken genre målgruppen ønsker, og derved lade denne være udgangspunkt for genren i produktet.

Hvor højt prioriterer du grafik i et spil?

En hypotese er, at grafik prioriteres meget højt af brugerne, i modsætning til AI. Svarene på dette spørgsmål kan vise, hvor meget der skal gøres ud af den grafiske del af produktet, for at tilfredsstille flest mulige brugere.

Hvor højt prioriterer du kunstig intelligens i et spil?

Som fortsættelse til ovenstående spørgsmål, af- eller bekræftes hypotesen om grafik kontra AI her. At de adspurgte angiver en lav prioritet til dette spørgsmål, kan skyldes, at de ikke mener, at den AI, der kan leveres i dag, er meget værd, eller at de måske udelukkende interesserer sig for multiplayer*-spil uden nævneværdig AI.

Hvor højt prioriterer du at et spil overholder realisme mht. fysiske love og menneskelige begrænsninger - fx tyngdekraft, hastigheder og dødelighed?

Spørgsmålet giver klarhed om, hvorvidt målgruppen ønsker, at spillet skal være realistisk, således at spilleren fx dør ved ét skud. Dette kan ses i forhold til, at det måske giver bedre gameplay* ikke at holde realismen højt.

Hvor højt prioriterer du modstanderens evne til at finde vej i spilverdenen - altså at komme udenom forhindringer, og ikke sidde fast i blindgyder?

Da en del af emnet for dette projekt er AI, er det vigtigt at vide, hvilke punkter inden for AI, brugerne prioriterer højt. Ved dette spørgsmål er målet at finde ud af, hvor meget fokus, der skal rettes mod vejfinding i spillet.

Hvor højt prioriterer du modstanderens udnyttelse af dens færdigheder (f.eks. sigte eller våbenvalg)?

Her ønskes at vide, hvor meget færdighederne skal vægtes, som fx at sigte, samle forsyninger op, og skifte våben.

Hvor højt prioriterer du modstanderens taktiske egenskaber (f.eks. samarbejde og forudsigelser)?

I dette tilfælde, ønskes at vide i hvilken grad, de taktiske egenskaber skal prioriteres i forhold til produktets AI, som fx at søge dækning, hvis det er umuligt at overleve et angreb, eller at tilkalde hjælp.

Hvor højt prioriterer du modstanderens egenskaber til at lære af sine fejl (f.eks. ikke at løbe i samme bagehold hver gang)?

Noget af det unikke ved den biologiske intelligens er, at den kan lære af sine fejl. Det er vigtigt at vide, i hvor høj grad der skal fokuseres på, at få dette med i produktet.

Hvor højt prioriterer du at modstanderen har en kort betænkningstid - altså at modstanderen

ikke bruger for lang tid på hvert træk?

I visse spil har stræben efter en god AI bevirket, at modstanderen bruger urimeligt lang betænkningstid. Der ønskes at vide om brugerne mener, at det er vigtigt at undgå denne lange betænkningstid.

Nævn nogle spil du mener har en god kunstig intelligens.

Formålet her er at få forslag på spil med god AI, og derved få inspiration.

2.4.2 Resultater

Spørgeskemaet er i alt blevet sendt ud til 1291 personer. Ud af dem er der 431, der har svaret, hvilket svarer til 33,4%.

Undersøgelsen viser, at ca. 89% af de, der har svaret, er i aldersgruppen 18-24 år, hvilken også er målgruppen. Dette var også forventet, da denne aldersfordeling på Basisåret var grunden til, at spørgeskemaet blev udleveret der.

Antallet af timer de studerende bruger på at spille, er generelt begrænset, men der er dog over hundrede, der har svaret, at de spiller mere end seks timer om ugen.

De foretrukne spilgenrer var Action og Strategi. Omkring halvdelen af svarene viste dette.

Hypotesen om, at der spilles forskellige typer spil afhængig af køn, blev bekræftet, da kvinderne der svarede på vores spørgeskema foretrak at spille brætspil og simulationsspil; mens mændene foretrak actionspil og strategispil.

Resultaterne af de følgende spørgsmål, der blev besvaret som prioriteter, er her omregnet til det aritmetiske² gennemsnit af prioriteterne.

Resultaterne af, hvor højt henholdsvis AI og grafik blev prioriteret viser, at AI foretrakkes marginalt i forhold til grafik. De fik de gennemsnitlige prioriteter 3,9 og 3,6 ud af 5. AI bliver kun prioriteret en smule højere end grafik, og derfor kan grafik ikke helt glemmes i produktudviklingen. Dog vil den grafiske side, i dette projekt, blive afgrænset meget, for at hovedvægten kan lægges på aspekterne inden for AI.

Meningerne om vigtigheden af realisme i spil var så forskellige, at der ikke kan udtrækkes nogle generelle tendenser - gennemsnitlig prioritet: 3,1. Dette resultat kan skyldes, at spørgsmålet ikke var formuleret tydeligt nok, således at de adspurgte var i stor tvivl om, hvad der var ment med spørgsmålet.

Indenfor de specifikke AI-spørgsmål, var vejfinding en klar favorit, med højeste prioritet som den mest valgte, og en gennemsnitlig prioritet på 4,1. Ved de resterende var 4. prioritet favorit. I prioriteret rækkefølge var disse følgende: Læring (3,93), taktik (3,91), betænkningstid (3,8) og til sidst udnyttelse af færdigheder (3,6).

Det sidste spørgsmål på spørgeskemaet er der tvivl om, hvorvidt de adspurgte forstod korrekt. Der blev spurgt om, hvilke spil de mente havde en god kunstig intelligens, men svarene tyder på, at nogle af de adspurgte i stedet valgte spil, de godt kunne lide at spille. Denne konklusion drages på baggrund af, at mange spil med næsten ingen AI fik stemmer, og under valgmuligheden "Andet" blev der valgt flere spil, hvor der slet ikke indgår AI. Vinderne var henholdsvis "The Sims"-serien, "Half-Life"-serien og herefter "Grand Theft Auto"-serien. Det skal siges, at de svar, der blev givet efter at "Andet" var valgt, var så spredte, at de ikke havde indflydelse på resultatet.

²Værdierne adderes og divideres med antallet af målinger.

Resultaterne kan bruges til at afgøre, hvordan de forskellige elementer i spillet skal prioriteres, specielt indenfor AI. På denne måde vil det være muligt at kunne tilfredsstille målgruppen bedst muligt.

I dette projekt afgrænses den grafiske side meget, således der kan fokuseres på aspekterne inden for AI. Specifikt indenfor AI, vil der så vidt muligt blive taget højde for brugernes ønsker i førnævnte prioriterede rækkefølge: vejfinding, læring, taktik, betænkningstid og til sidst udnyttelse af færdigheder. Dvs., at der vil blive gjort mest ud af, at vejfindingsalgoritmen finder den bedste rute gennem spilverdenen.

En komplet oversigt over resultaterne fra spørgeskemaundersøgelsen kan ses på bilag 2.

2.4.3 Valg af spiltype

Efter spørgeskemaundersøgelsen var tendensen, at actiongenren var foretrukket. På baggrund af dette og gennemgangen af spiltyper, se afsnit 2.2.3, er der valgt, at produktet skal være et actionspil. Actiongenren er i realtime, og der vil også være designmæssige overvejelser i forbindelse med dette. Det er valgt, at actionspillet skal være i 2D. Derved bliver AI'en mere simpel at lave, da der ikke skal tages højde for den tredje dimension.

Kapitel 3

Problemformulering

Gennem problemanalysen blev målgruppen valgt, og de forskellige aspekter indenfor AI blev diskuteret. Spørgeskemaet gav indblik i, hvordan målgruppen vægter de forskellige dele af AI og grafik - disse resultater vil bl.a. blive brugt til at formulere det problem, der ønskes løst. Actiongenren blev valgt som udgangspunkt for produktet.

3.1 Probleformulering

De 18- til 24-årige er valgt som målgruppe, og dette gør det muligt at udvikle et skydespil uden at tage hensyn til aldersgrænser. Skydespil med voldeligt indhold kan sælges til yngre personer i Danmark, hvorimod andre lande har andre regler for dette, der ikke tillader børn og unge under spillets aldersgrænse at købe spillet.

Spørgeskemaet viste, at målgruppen foretrækker spilgenrene action og strategi, samt en lille tendens til, at de foretrak god AI i forhold til god grafik. For at få indblik i målgruppens mening om AI mere specifikt, blev AI delt op i mindre under-kategorier, hvorefter målgruppen skulle prioritere disse. Disse AI-egenskaber blev prioriteret som følgende: vejfinding, læring, taktik, betænkningstid og til sidst udnyttelse af færdigheder. Målgruppens prioriteter vil så vidt muligt blive forsøgt overholdt i produktet, og udviklingsfasen af dette.

Igennem problemanalysen er der også blevet kigget på forskellige AI-principper. Disse principper bestemmer, hvordan en AI udregner og vælger, hvilken opførelse den skal have, eller hvordan den skal reagere. Gennemgangen blev lavet, for at få et større indblik i, hvordan disse virker og hvordan disse skulle programmeres. De hjalp også til, hvilken form for AI, der blev valgt til spillet.

For at få et overblik over de forskellige spiltyper, som findes i dag, blev der lavet en gennemgang af de forskellige spiltyper. Her blev der fundet ud af, hvad de forskellige typer handler om, og hvad der kræves af AI i hver af disse. Denne gennemgang af spiltyper havde også indflydelse på, hvilken spiltype der er blevet valgt til spillet.

På baggrund af problemanalysen er der fremkommet en række spørgsmål, som beskriver problemet nærmere. Disse spørgsmål er formuleret nedenunder.

- Hvordan udvikles et skydespil med kunstig intelligens?
 - Hvordan kan kunstig intelligens implementeres i et skydespil?
 - Hvordan kan den kunstige intelligens i produktet virke ikkedeterministisk, men stadig intelligent?

3.2 Projektafgrænsning

Der er visse emner, som ikke vil blive gennemgået eller brugt yderligere i dette projekt. Dette afsnit vil indeholde en diskussion herom.

Spillets genre er valgt til at være action. Dette valg er foretaget ud fra gennemgangen af spiltyper, samt resultaterne fra spørgeskemaundersøgelsen. Spillet udvikles på Windows-platform. Det er ikke et krav, at det fungerer på andre platforme. Som programmeringssprog vil C++ blive anvendt, og SDL-biblioteket vil blive brugt til basal vindues-håndtering og initialisering. OpenGL vil blive brugt til spillets grafiske side, og OpenAL til lydsiden.

Da projektet ønskes fokuseret på konstruktion af en fungerende AI, nedprioriteres grafikken. Spillet vil altså kun indeholde den mest basale grafik, der er nødvendig for at vise AI færdigheder. AI vil være baseret på fuzzy logic til at tage beslutninger. Grunden til, at denne metode er valgt skyldes, at der er bedre mulighed for at lave en mere dynamisk AI, i forhold til hvad der ville være muligt udelukkende ved brug af et ekspert system. Til vejfinding vil den anerkendte algoritme Dijkstra, og dens afart A* blive anvendt - afhængigt af om destinationen er kendt. Læring vil ikke blive implementeret. Der er også valgt, at AI skal have en vis form for taktik, men her er der valgt, at den eneste taktik AI skal have er, at den skal kunne forudsige, hvor spilleren bevæger sig hen.

Der er ikke blevet valgt en decideret metode, men det er mere en sammenblanding af flere forskellige metoder, hvor der er valgt de ting, som passer bedst til projektet. De metoder, der er blevet kigget på og taget fra, er følgende:

- XP
- Scrum
- Dataindsamlingsmetoder
 - Informationssøgning
 - Spørgeskema

Kapitel 4

Metode

I dette kapitel bliver der reflekteret over de anvendte metoder, som blev brugt under problemanalysen. Ydermere vil der blive set på hvilke metoder, der anvendes til produktudviklingen. De valgte metoder til produktudvikling baseres på eksisterende metoder.

4.1 Problemanalyse

4.1.1 Dataindsamlingsmetoder

Informationssøgning

Under informationssøgning har Aalborg Universitetsbiblioteks litteratursøgningsværktøj¹ været benyttet til at finde relevant materiale til projektet.

Udover Aalborg Universitets bibliotekshjemmeside er søgemaskiner som www.google.com og online leksika, som www.wikipedia.org taget i brug, til at finde materiale og inspiration til emneinddeling af rapporten.

Med hensyn til troværdighed omkring kilder henvises til afsnittet om kildekritik, se afsnit 11.2.

Spørgeskema

For at få indblik i målgruppes meninger om de emner, der skal tages højde for i dette produkt, blev spørgeskemaet sendt ud til de studerende på Det Teknisk-Naturvidenskabelige Basisår og Det Samfundsvidenskabelige Basisår. Resultaterne skulle bruges til at give et indblik i, hvilke AI aspekter der skulle prioriteres højt, og hvilke spilgenrer der foretrakkes.

Spørgeskemaet er bevidst en kvantitativ målgruppeundersøgelse, i stedet for en kvalitativ undersøgelse, som fx interviews. Spil har en stor målgruppe, og af personlig erfaring vides, at der kan være stor uenighed om, hvad brugerne mener, der skal prioriteres højt i spil. Ved at anvende en kvantitativ målgruppeundersøgelse, fås et bredt og generelt billede af målgruppens ønsker.[11]

Fordelen ved en spørgeskemaundersøgelse er, at det er lettere at nå ud til mange mennesker i forhold til interviews, men ulempen er, at det er lettere at afvise en skriftlig anmodning om en besvarelse. En anden ulempe ved denne undersøgelsesmetode er, at mange mennesker oftest foretrækker at få et mundtligt spørgsmål frem for at skulle læse sig til spørgsmålet.[11] Af andre ulemper kan nævnes, at hvis ikke den adspurgte har forstået spørgsmålet, er der ingen eller ringe mulighed for at få en uddybende forklaring af spørgeren. Ved spørgeskemaundersøgelser kan der derimod ses tendenser fra en større målgruppe, hvor der ved interviews let kun kommer individuelle meninger tilkende.

Troværdigheden underbygges i og med at spørgeskemaet og de tilhørende resultater, er vedlagt som bilag 1 og 2.

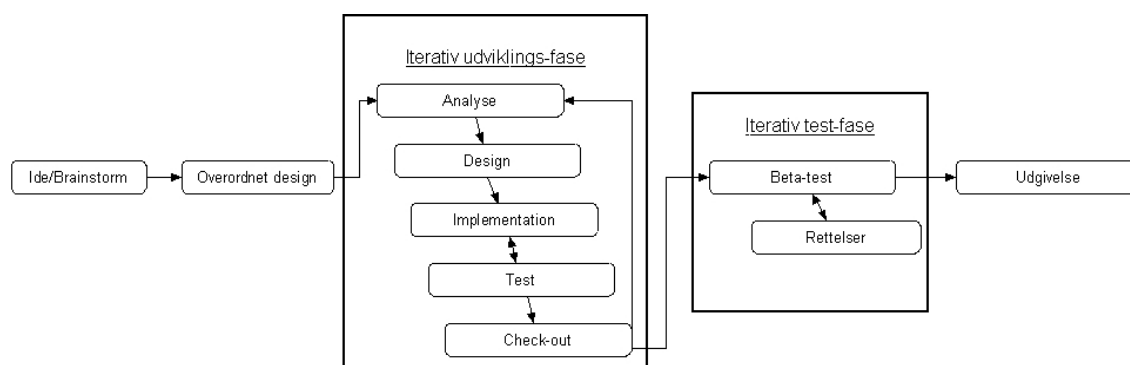
¹<http://www.aub.aau.dk>

4.2 Produktudvikling

For at løse det opstillede problem, er det yderligere nødvendigt, at se på hvordan det kan gribes metodisk an. Dette kan gøres gennem forskellige udviklingsmetoder, der kan anvendes til software udvikling. Ud fra kendte metoder som XP og Scrum har vi fået indspiration til vores egen metode, som vil blive beskrevet nedenunder.

4.2.1 Faser

Der er forskellige faser i den valgte udviklingsmodel, som skal gennemgås i den iterative proces for at producere et spil.



Figur 4.1: Illustration af udviklings-forløbet, ved brug af metoden.

Når spil idé og design er på plads, starter første delfase i den iterative udviklings-fase. Dette er en analysedel, hvor de forskellige problemstillinger for, hvilket indhold der skal være i hver del af spillet, bliver analyseret. Der bliver også kigget på, hvilke værktøjer og teorier der kan bruges til at løse problemet.

Derefter bliver der startet på anden delfase, hvor der bliver lavet design over den valgte problemstilling. Her findes ud af, hvad problemstillingen skal indeholde, og hvad der skal til for at løse denne.

Tredje delfase er implementation, hvor funktioner forsøges at blive implementeret. Under implementationen vil de forskellige værktøjer og teorier blive brugt til at løse problemet.

Som den sidste delfase i den iterative udviklings-fase, se figur 4.1, kommer test. Her vil den kode, som er blevet lavet under implementationen, blive testet, så evt. fejl vil kan findes. Hvis der er en fejl, sendes koden tilbage til implementationsdelfasen, hvor fejlen vil blive rettet. Dette fortsættes, til alle fejl er rettet.

Når koden er færdig vil der blive fortsat til den iterative testfase hvor det endelige produkt vil blive testet. Hvis flere fejl bliver fundet vil processen gå videre til en delfase der retter disse. Når produktet endeligt er fejlfrit nås udgivelses fasen.

4.2.2 Metodevalg

Her vil der blive beskrevet, hvilke metoder der vil blive brugt igennem produktudviklingen. Igennem denne fase, vil der blive brugt en kombination af iterations- og vandfaldsmetoden. Som et eksempel, bliver der taget en funktionalitet som skal være i spillet. Her bliver der først lavet design, herefter implementation og til sidst test. Herefter påbegyndes samme trin for en anden funktionalitet. Denne metode er illustreret på figur 4.1.

Programmering i par

Når der skal kodes i grupperummet, vil der så vidt muligt blive brugt parprogrammering. Ved at bruge parprogrammering, er der mulighed for, at kodestrukturen kan blive diskuteret. Det kan også afhjælpe skrivefejl eller kodefejl, ved at der er to par øjne som kigger på koden samtidig. Arbejdet kan også blive mere effektivt, da begge kommer med input til koden, mens der skrives. Det er også en hjælp til, at der ikke mistes fokus, som kan ske, hvis vedkommende sidder alene.[12]

Ulemperne kan være, at der kan opstå en diskussion, som måske ikke har noget med koden at gøre, eller at parret ikke kan blive enige. Ved at lave parprogrammering, er der kun en mand som sidder ved computeren, mens den anden sidder ved siden af og hjælper til.[12]

Kodestandarder

Der er besluttet at bruge en fælles kodestandard, da det gør læsningen af andres kodestumper lettere. Når alle dele af programmet er skrevet efter samme standard, er det lettere at rette i koden for alle.[12]

Det kan dog være hårdt for nogen at vende sig til at skrive i en ny "stil". Det kan tage længere tid at skrive efter en fremmed kodestandard, men det er prisen for, at alle kan læse hinandes kode.[12]

Korte deadlines med opgavelister

Der vil blive brugt korte deadlines, hvor der under hver deadline vil være en opgaveliste. Denne opgaveliste indeholder forskellige problemstillinger som skal løses, før deadline nås. Hver deadline repræsenterer en funktionalitet, som skal implementeres i programmet. Fordelene er, at opgaver kan blive meget præcise, så alle ved, hvad der skal laves. Ved hjælp af dette, kommer der også et større overblik over, hvor langt produktet som helhed er nået.[12]

Ved at bruge korte deadlines, kan det være svært at nå en deadline, hvis projektet bliver indelt i for store problemer. Når deadline skal nås, og vedkommende bliver presset, kan koden nemt blive uoverskueligt og dårligt struktureret, så det bliver svært for andre at læse den. Der kan også opstå flere fejl, fordi det skal gå hurtigt, inden deadline udløber.[12]

Løbende dokumentation

Der vil gennem implementationsfasen blive skrevet dokumentation løbende, dette gøres for at der bliver skrevet dokumentation så hurtigt som muligt. Dette vil hjælpe med til at andre lettere kan sætte sig ind i hvad andre har skrevet. Det vil derfor lette ændringer og omskrivninger når der er en god dokumentation. Når der er fælleseje af koden er det også vigtigt at alle hurtigt kan sætte sig ind i hvad andre har skrevet, da det højner vidensdelingen i gruppen og sørger for at alle kan stå til ansvar for koden.[13]

Dog kan det være meget tidskrævende at skrive dokumentation af alt hvad der laves. Det kan derfor gøre at produktet tager længere tid at udvikle, men til gengæld skal der ikke skrives dokumentation til sidst.[13]

Fælles ansvar for koden

Alle i gruppen har ansvar for den kode, der bliver skrevet. Dvs. at hvis et gruppemedlem finder en fejl i koden, som en anden har skrevet, er det blevet vedkommendes opgave at udbedre denne fejl. Dette kan afhjælpe små fejl i koden, som måske er overset af programmøren og vedkommendes

partner. Det vil også være en hjælp til at få optimeret koden, da alle har ansvar for koden. Derfor har andre også pligt til at optimere koden, hvis de finder en mulighed for dette.[12]

Et problem kan være, at en anden skal rette i noget kode, som vedkommende ikke har skrevet. Dette ville kunne give risiko for, at vedkommende ødelægger koden, i stedet for at rette fejlen som vedkommende fandt.[12]

Refaktorering*

Når der skrives kode til spillet, vil dette kunne gøres således, at koden i første omgang fås til at virke. Derefter vil det ofte være en fordel at gå i gang med at omskrive koden, således den kan bruges i andre sammenhænge, og derved højne abstraktionsniveauet i koden.

Dette kan gøre koden mere gennemskuelig, og nemmere at forstå for udefrakommende. Til gengæld tager det tid at omskrive koden. Ved ikke at skrive koden abstrakt i første omgang, er der større sandsynlighed for korrekthed af koden.

Løbende test

Der vil blive lavet løbende test, for at tjekke for eventuelle fejl, og for at afgøre om en mindre del af koden, der er tilføjet, også har den ønskede virkning. I vores udviklingsproces vil vi ikke beskæftige os med test inden kodning, som er en del af eXtreme Programming[12].

Ulempen ved løbende tests kan være, at det tager tid, at lave alle disse test, men det sikrer en højere grad af korrekthed af koden.

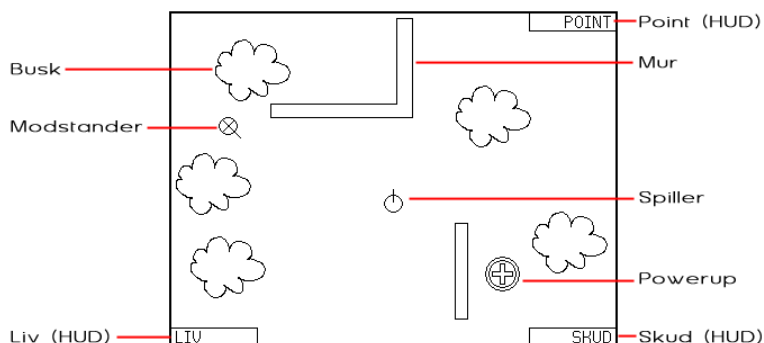
Kapitel 5

Kravsifikation

Før udviklingen af produktet kan påbegyndes, udspecificeres de gældende krav for spillet. Disse krav er hovedsageligt udmundet af problemanalysen, og senere designafsnittet. Nogle få krav, og beskrivelsen af disse, introduceres først i dette afsnit.

5.1 Funktionelle krav

5.1.1 Brugergrænsefladen



Figur 5.1: Skitse over GUI'en med forskellige elementer fra spillet.

I dette afsnit vil brugergrænsefladen og HUD'en* til spillet blive beskrevet. Spillet er et 2D actionskydespil.

Spilverdenen ses ovenfra, og spillerens figur er placeret centreret et stykke nede på skærmen således, at spilleren får en fornemmelse af, at der er et større overblik fremad, mens overblikket bagud er forringet. Derudover står spillerens figur fast på skærmen og har altid ansigtet i en fast retning. Dvs. at når figuren bevæger sig, er det i virkelighed spilverdenen, der flyttes. Dette er besluttet, da det giver mere udsyn i spillet, end hvis spilleren flytter sig rundt i spilverden.

Udover spillerens figur skal spilleren have informationer om, hvor meget liv og ammunition spilleren har tilbage. Indikatoren over hvor meget liv, der er tilbage, placeres nederst i venstre hjørne, mens ammunitionsindikatoren placeres i højre hjørne. For at holde styr på hvor mange point, der er opnået, placeres der desuden en pointindikator i øverste højre hjørne.

Disse elementer i GUI'en illustreres på figur 5.1.

5.1.2 Display

Alle objekter og banelementer i spillet skal vises på skærmen. Der skal hentes oplysninger om, hvor de forskellige objekter skal tegnes, og hvilke dele der skal tegnes forrest og bagerst. Følgende elementer skal tegnes: Omgivelserne, spilleren, modstandere, powerups, og HUD'en. Det er vigtigt at de forskellige elementer bliver vist rigtigt, dvs. at banen skal vises bagerst og at spilleren og modstanderne bliver vist foran denne. HUD'en bliver så lagt øverst, så den altid er synlig for brugeren.

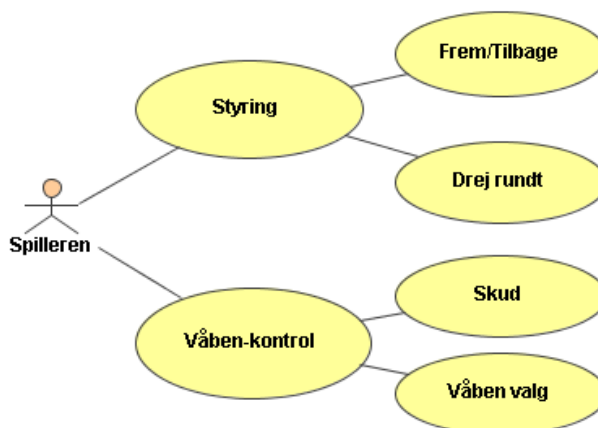
Denne funktion er en af de mest vigtige for spillet, da der ellers ikke ville blive vist noget for brugeren.

5.1.3 Input fra brugeren

Figuren i spillet skal kunne styres af spilleren, og disse input vælges at blive registreret fra tastaturet. Ved hjælp af piletasterne på tastaturet kan spilleren styre figuren i spillet i den retning, som der ønskes. Her vil *pil-op* få figuren til at gå fremad, mens *pil-ned* får figuren til at gå tilbage. *Venstre-pil* og *højre-pil* får figuren til at dreje til henholdsvis venstre og højre.

Ydermere skal brugeren kunne skyde modstandere i spillet. Her vil et tryk på *mellemlumstasten* affyre et skud i spillet i figurens retning. Våbenvalg foregår med de numeriske taster *1-5*.

De muligheder spilleren har, er opstillet i use-casen der vises på figur 5.2. Denne use-case gælder desuden også for den kunstige intelligente modstander, da denne skal være begrænset til samme handlinger som spilleren.



Figur 5.2: Use-case for spiller og den kunstige intelligente modstander

5.1.4 AI

I dette afsnit af kravsspecifikationen vil kravene til den kunstige intelligente modstander blive formuleret, og der vil blive beskrevet hvordan denne skal opføre sig.

Vejfinding

Modstanderen vil komme i situationer, hvor det er nødvendigt at finde vej fra et punkt til et andet. Der vil først blive beskrevet, for hvilke situationer modstanderen skal bevæge sig, og dernæst, hvad kravene til en vejfindingsalgoritme skal være.

Hvis modstanderen kommer i en situation, hvor denne enten mangler våben, liv eller ammunition, er det nødvendigt at finde vej til den nærmeste powerup indeholdende dette. Som udgangspunkt kender modstanderen banen, således vejfindingsalgoritmen kan blive gennemført uden at modstanderen først skal lede efter en powerup. Dette gør det muligt for modstanderen at finde vejen til den nærmeste powerup.

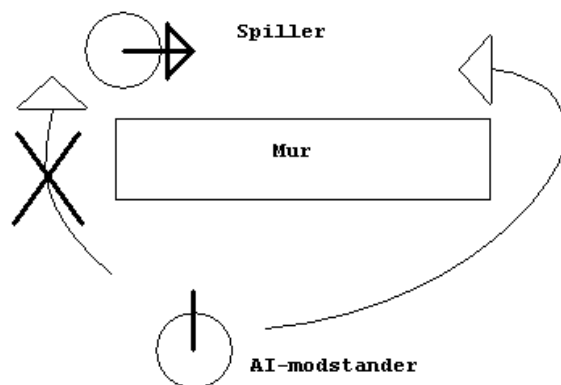
Når modstanderen har besluttet sig for en handling, hvor der skal bruges vejfinding, vil vejfindingsalgoritmen også blive brugt.

Taktik

I dette afsnit vil modstanderens taktiske egenskaber blive beskrevet.

Modstanderen skal kunne forudse spillerens træk., dvs. spillerens løbebane. Når modstanderen har forudset løbebanen, vil modstanderen ikke skulle indhente spilleren, men kan i stedet løbe foran spilleren og overraske denne. Læring ville være nærliggende at implementere i dette tilfælde, da det ville gøre det mulige at forudsige spillerens handlinger ud fra spillerens bevægelsestendenser i lignende situationer. Dette aspekt er dog afgrænset fra dette projekt.

Et eksempel på dette kan ses på figur 5.3, hvor en spiller er løbet om hjørnet af en forhindring. Den umiddelbare løsning ville være, at følge efter spilleren, men i stedet ville det være en bedre ide, at løbe den modsatte vej, og overraske spilleren.



Figur 5.3: Eksempel på situation, hvor der skal tænkes taktisk

Et andet taktisk element skal være, at få modstanderen til at sigte foran spilleren. Dette skal gøre at modstanderne kan ramme spilleren, selv om vekomende bevæger sig.

For at imødekomme kravet om at forudsige spillerens bevægelser, vil det være nødvendigt at forudsige spillerens position ud fra spillerens retning og hastighed.

5.1.5 Omgivelser

Selve banen, som spillet forgår i, skal være opbygget af mindre kvadrater. Hvert kvadrat kan indeholde forskellige elementer: Græs, mur, vand, strandkant, bro, træer, buske eller bilspor. Et element kan enten helt gennemtrænges, eller slet ikke gennemtrænges. De gennemtrængelige elementer skal være: Græs, strandkant, bro, og bilspor. Udover at de resterende elementer er ugennemtrængelige, skal der også være en begrænsning af selve banen. Banen skal kun kunne gennemtrænges i det omfang hvor elementerne eksplicit er angivet for banen. Når spilleren kommer i nærheden af en bane-kant, skal det sidste element fra bane-kanten, efterfølges af tilsvarende elementer, således at spillerens synsfelt ikke vil vise manglende elementer uden for banen.

Udover gameplay-objekterne (Spillere og powerups) i spillet, skal der også være andre objekter, hvis formål skal være at gøre omgivelserne mere livagtige. Af synlige objekter skal der være dyr, i form af kaniner. Af usynlige objekter skal der være diverse naturlyde, der kan medvirke at spilleren kan leve sig ind i spillet. Udover naturlyde, vil der også forekomme baggrundsmusik, for at øge underholdningsværdien af spillet.

5.1.6 Gameplay

Spillet skal gå ud på, at spilleren skal gå rundt med sin figur, og skyde de modstandere som findes på banen. Spilleren skal have mulighed for at se, hvor meget liv, ammunition, antal dræbte og point vedkommende har.

Der er forskellige spilleregler, som spilleren og modstanderne ikke kan ændre eller overskride. Hele banen skal være firkantet, med forskellige elementer såsom træer, sten, mur og græs. Disse elementer kan enten være solide, så spilleren ikke kan gå igennem dem, eller de kan være passerbare, fx græs. Der vil ikke være nogen mulighed for, at gå uden for banen.

Når spillet startes, skal spilleren og modstandere starte med et standart våben, en pistol.

I løbet af spillet skal spilleren og modstanderne have mulighed for, at samle andre våben op i form af powerups, som så kan benyttes. Hvis spilleren har flere våben, så skal vedkommende have mulighed for at skifte, alt efter hvilket våben spilleren vil bruge. De våben der skal kunne benyttes er som følger: Pistol, dobbelt pistol, shotgun, riffel og bazooka.

Der skal yderligere være andre former for powerups, der giver mulighed for at få fuldt liv, eller få en øget hastigheden med 100% i 10 sekunder.

Der skal være dyr, i form af fx kaniner, på banen. Der skal være mulighed for at dræbe disse dyr. Efter et stykke tid, skal de dræbte dyr genoplives igen, så disse ikke forsvinder. De modstandere, der spilles imod, kan også dø. For hver af disse der dør, vil spilleren opnå ét point. Efter et stykke tid, skal modstanderne spawnes igen, så der altid er modstandere på banen.

Spillet slutter, når spilleren dør. Der skal ikke være nogen drabs- eller tidsbegrænsning, så spillet skal ikke slutte uden videre, men fortsætte indtil spilleren dør. De point, som spilleren har indsamlet i løbet af spillet, skal til slut blive sammenlignet med den highscore som findes i forvejen. Hvis spilleren har nok point, til at komme på highscoren, så skal vedkommendes navn sættes ind på denne.

5.2 Ikkefunktionelle krav

5.2.1 Platform

Spillet skal kunne eksekveres på alle pc'er der kører Microsoft Windows XP.

5.2.2 Skalerbarhed

Spillet skal have mulighed for at håndtere mindst 200 objekter (personer, kaniner, skud og powerups) på samme tid. Det skal desuden være let at implementere nye objekter i spillet. Banerne i spillet har en fast størrelse, men kan defineres i læsbare separate filer. Der kan altså være alle de baner i spillet, som spilleren har behov for.

Spillet skal skrives således, at det uden fundamentale omskrivninger kan videreudvikles til en anden type spil, fx et multiplayerspil.

5.2.3 Brugervejledning

Der skal være en kort vejledning til spillet, så brugeren kan læse en kort forklaring om, hvad spillet går ud på. Desuden skal der også være en beskrivelse af, hvordan spilleren skal styre sin karakter. Der skal være en beskrivelse af de forskellige powerups, som kan forefindes i spillet, så spilleren ved, hvad disse gør.

5.2.4 Samtidighedsproblemer

I spillet vil der være mange objekter der bevæger sig samtidigt. Da en computer, med kun en CPU, kun kan foretage en beregning ad gangen, vil dette kunne forårsage et problem, hvor objekter ville støde sammen. Computeren rykker hvert objekt for sig efter tur, så hvis to objekters oprindelig bane viser, at de burde støde sammen, er det ikke sikkert, at de gør det. Dette kan ske da et objekt altid vil blive rykket før et andet, og derfor ville et objekt måske blive flyttet ud af banen for det andet objekt.

Et andet problem kan også opstå, når der registreres en kollision. Når dette sker, er problemet at finde ud af, hvilket objekt der skal kalde de givne funktioner, således at begge objekter ikke kalder den samme funktion.

5.2.5 Performance

Det endelige produkt skal kunne eksekveres tilfredsstillende, dvs. med flydende billedeopdateringer, på en pc med disse minimum specifikationer:

- CPU på 1500 MHz
- 512 MB RAM
- Grafikkort (med OpenGL understøttelse)
- Windows XP

Kravene til performance vil blive vægtet højere end kravene til grafik.

5.2.6 Åbenhed

De ressourcer, som bliver brugt i spillet, skal være tilgængelig for alle. Derved har alle mulighed for, at kunne ændre i koden og udbygge spillet efter behov, eller bruge grafikken eller lydene i anden sammenhæng.

5.3 Prioritering af de funktionelle krav

Der er forskellige krav til, hvordan spillet skal fungere. Der er dog nogle af dem, som er mere vigtigere end andre. Derfor laves der en prioritering, så de krav med højest prioritet bliver lavet først.

Prioriteringen skyldes, at de bærende aspekter af spillet skal programmeres først. Her er der tale om, at det visuelle bliver lavet først, samt de input som skal komme fra spilleren. Til sidst bliver den kunstige intelligens programmeret, da dette ikke er et af de bærende aspekter til spillet.

Der skal tages højde for at disse prioriteter ikke modstrider med hinanden som fx hvis performance og grafik begge er vægtet højt.

De funktionelle krav prioriteres således:

1. Display
2. Omgivelser
3. Input fra brugeren
4. Gameplay
5. Brugergrenseflade
6. Vejfindning for AI
7. AI's udnyttelse af færdigheder
8. Taktik for AI
9. Læreevner for AI

Der er også opstillet ikke funktionelle krav, som ser således ud:

1. Platform
2. Skalerbarhed
3. Samtidighedsproblemer
4. Brugervejledningen
5. Åbenhed

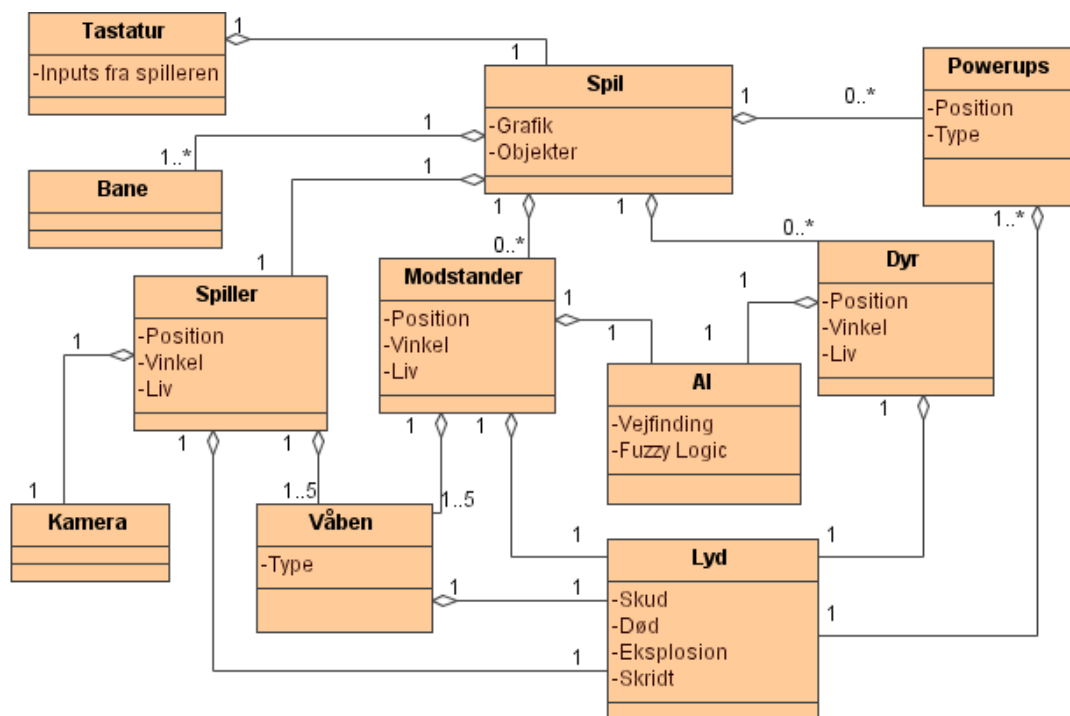
Kapitel 6

Analyse

I dette kapitel gennemgås en analyse af, hvorledes de udspecificerede krav kan opfyldes i det endelige produkt. Der vil blive kigget på strukturen og spildéen samt på samtidighedsproblemet og hvilken type AI, der skal benyttes i spillet.

6.1 Struktur

Det endelige produkt skal udvikles objektorienteret. Derfor kan det give overblik over produktet at illustrere sammenhænge mellem dets klasser. Klasserdiagrammet på figur 6.1 viser disse sammenhænge, og hvorledes de skal aggregere i forhold til hinanden.



Figur 6.1: Klassediagram over opbygningen af spillet

Spil er den centrale klasse som indeholder *grafik* og *objekter*. *Grafikken* sørger for at tegne objekterne i spillet. Til hvert spil tilhører kun et tastatur, som giver input fra spilleren. Spillet har flere baner, som spilleren frit kan vælge imellem ved start af spillet.

Derudover findes der også kun en spiller til hvert spil. Dette er valgt, da der ikke er taget højde for skalerbarhed ved udvidelse til multiplayer-spil, hvor der kan være flere spillere i samme spil. Til hver spiller tilhører der kun ét kamera og en lyd. Med en lyd menes, at spilleren har en enkelt lydkilde. Spilleren har ét til fem våben, det vil sige, at spilleren altid har mindst ét våben ved start. Ligesom spilleren har våben også kun én lydkilde.

Til spillet findes der et ubestemt antal fjender og dyr. En modstander har ligesom spilleren ét til fem våben og én lydkilde. Dyr har også én lydkilde, men intet våben. Til både fjenden og dyret tilhører der en AI.

Der er et ubestemt antal powerups i spillet. Der hører én lydkilde til alle powerups i spillet.

6.2 Spil idé

Under afgrænsningen blev det besluttet at lave et 2D action-skydespil. I afsnit 2.2.3 blev actiongenren diskuteret. I dette afsnit vil der blive fremlagt, hvad dette spil skal indeholde, og hvordan gameplayet skal foregå.

Spillets mål skal være at skyde så mange af computerens modstandere som muligt, og undgå selv at blive skudt. Dør spilleren skal spillet stoppe, og de point, spilleren har erhvervet sig under

spillet, skal skrives på en highscore-liste*.

I spillet skal spilleren være alene mod resten af modstanderne i spilverdenen, som er styret af computeren. Der skal hele tiden være et fast antal modstandere i spillet. Når en modstander dør, skal en ny blive oprettet i spillet. Modstanderen skal have de samme forudsætninger for spillet, som spilleren. Computeren skal have mulighed for at bevæge sig i samme retninger og samle de samme powerups op som spilleren.

I spilverdenen skal der forefindes powerups i form af fx ammunition, ekstra liv og nye våben. Både spilleren og computeren skal have mulighed for at samle disse powerups op og få udbytte af dem. I spilverdenen skal der forefindes forhindringer, som fx træer og mure. Der er ingen mulighed for at gå gennem disse forhindringer, se afsnit 7.1.

Spillet skal ses ovenfra, og spilleren peger fremad. Der skal være kontrol over spillerens bevægelser og våben. Der skal dog ikke være mulighed for at styre spilleren i den tredje dimension.

6.3 Samtidighed

I et realtime actionspil findes der en del samtidighedsproblemer. Oftest opstår problemerne når objekter kolliderer. Her findes der tre situationer:

- To objekter flyver over hinandens baner hvor de burde kolliderer.
- Objekter der flyver over et stillestående objekt.
- Et objekt kolliderer med et andet, og bagefter kolliderer det andet objekt med det første.

I den sidste situation er problemet, hvilket objekt der skal kalde kollisionsfunktionerne, og hvordan det sikres at det andet objekt ikke kalder dem. I de to første situationer er problemet at objekter skal tjekke hele den rute de skal bevæge sig, om de kolliderer med noget. Dog vil den første situation kræve nogle yderligere tjek.

I dette produkt vil det største problem komme med skudobjekter, da de har en høj hastighed. Problemet vil være større på en langsom computer da objekterne her vil flyve længere for hver opdatering og derfor har større mulighed for at flyve forbi et objekt den ellers burde have kollideret med.

6.4 Fuzzy logic

Modstanderne i spillet skal kunne gøre nogle forskellige forudbestemte handlinger, alt efter hvilke situationer der er tale om. Ved at bruge fuzzy logic til den kunstige intelligens, gives mulighed for at lave en mere dynamisk AI. Dette kan gøres ved hjælp af de trapetzer som er med til at udregne fuzzy-summen. Når denne sum er udregnet, kan modstanderen sættes i tilstande som gør, at modstanderen handler på en passende måde.

Ved at bruge fuzzy logic, kan modstanderne få forskellige fuzzy-grænser. Dette gør, at spillet ikke bliver ensformigt ved at lade alle modstanderne have samme fuzzy-grænser. Alt efter hvilken fuzzy-grænser modstanderen har, vil de opføre sig på en bestemt måde.

En sådan grænse kunne laves, således at modstandere med lav fuzzy-grænse vil være mere aggressive, da disse kan tage mere skade, før de begynder at flygte. Dette er så omvendt med dem med en høj fuzzy grænse. Når en modstander er færdig med at flygte, vil denne begynde at søge efter powerups, for at få sin fuzzy-sum over fuzzy-grænsen. Dvs. at modstanden prøver at holde sig selv i live, for at kunne kæmpe videre senere.

Kapitel 7

Design

Dette kapitel omhandler, hvordan spillet skal opbygges. Spilleregler vil blive fastlagt, og der vil blive kigget på arkitekturen af spillet. Ydermere vil en grundig gennemgang af vejfindingsalgoritme gennemgås.

7.1 Spilleregler

Bevægelsesbegrænsninger

Spillerens såvel som modstandernes bevægelsesmuligheder, skal være begrænset af spilverdens fysiske miljø. De kan ikke bevæge sig gennem de objekter, der i den virkelige verden betegnes som ugennemtrængelige: Vægge, klipper, bygninger, træer og vand.

Spilleren skal have den fordel, at vekommende kan løbe hurtigere end sine modstandere. Dette skal gøres, for at forbedre gamplayet. Andre objekter, så som skud, skal have individuelle hastigheder.

Dødelighed

I spillet er det muligt at dø efter at være blevet ramt for mange gange. Spilleren skal som udgangspunkt have sværere ved at dø end modstanderen således, at spillet kan vare længere. Her kan henvises til spørgeskemaundersøgelsen, der viste at realisme ikke var højt prioriteret, for det er ikke realistisk at en person kan overleve flere skud fra et skydevåben. Hvor meget skade spilleren kan holde til i forhold til modstanderne, kan afhænge af den valgte sværhedsgrad.

Overlevelseskamp

Formålet med modstanderne i spillet, er at de skal jage spilleren. Spilleren er helt alene mod alle modstandere, hvilket også er grunden til at spilleren kan holde til mere - se ovenstående. Modstanderne skal derfor ikke kæmpe mod hinanden, men kun mod spilleren.

Skudbegrænsninger

Lige som ved karaktererne, skal skud også have begrænsede bevægelsesmuligheder. Skuds bevægelser skal dog ikke være begrænset af vand, træer og buske. Skud skal bevæge sig i en ret linje indtil de rammer et ugennemtrængeligt objekt, hvorefter de "forsvinder". Hvis der er en karakter der bliver ramt, mister denne en mængde liv, der afhænger af våbentypen. Det skal dog ikke være muligt at skyde sig igennem ugennemtrængelige objekter.

Synsfelt

Spilleren skal kunne se det udsnit af spilverdenen som skærmen viser, inkl. modstandere og lignende. Modstanderne derimod skal have et mere begrænset synsfelt, igen for at gøre gameplayet bedre, ved at give spilleren fordel i forhold til modstanderne.

Point

Der opnås kun point hver gang en modstander dør. Disse point tælles op indtil spilleren dør. Når spilleren selv dør, kan vedkommende komme på highscorelisten, hvis det opnåede antal point er større end den highscore med mindst antal point.

Powerup

Gennem spilverdenen, skal spilleren have mulighed for at opsamle powerups. Disse kan give en af følgende forstærkninger: Mere liv, mere ammunition, andre våben. De forskellige powerups placeres forskellige steder i spilverdenen, og der kan komme nye til imens spillet er igang. Det sammeg ælder for modstanderne

Våben

Spilleren vil kunne vælge mellem en række forskellige våben med forskellige egenskaber og fordele. Spilleren skal kun starte med en pistol, hvorefter sortimentet skal kunne udvides, ved at opsamle de førnævnte powerups. Det totale våbensortiment skal se således ud: Pistol, dobbelt pistol, riffel, shotgun og bazooka.

7.2 Real-time

Spillet er i realtime. Dette betyder, at der ikke vil være nogle stop i spillet. Spilmotoren skal således køre uden afbrydelser. Måden dette kan gøres på minder om, hvordan det gøres i turbaserede spil, men der vil ikke være nogen pause, hvor spiller eller computer skal tænke. Fx vil dette se ud, som følgende:

```
while(condition){
    //run game
    playersTurn();
    computersTurn();
}
```

Inde i *playersTurn()*, ville der i et turbaseret spil være en form for pause ved spillerens input, og spillet ville ikke køre videre, før spilleren har givet input. I et realtime-spil, venter spillet ikke på dette input. Hvis spilleren ikke giver noget input, vil dette blive opfattet som en passiv tur. Dette kan give en illusion af, at spillet foregår i realtime.

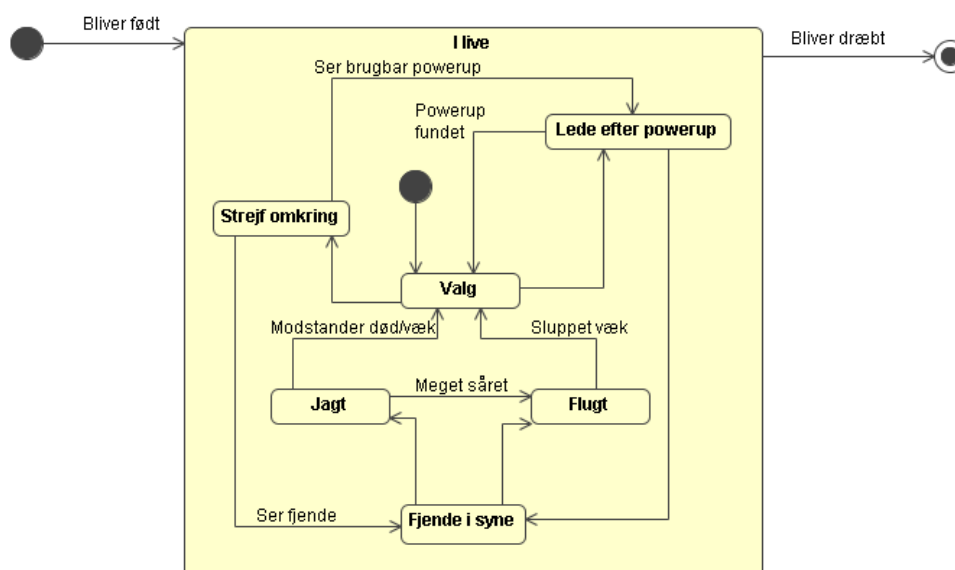
7.3 Spiltilstande

En af grundelementerne i et spil er tilstande, der er afgørende for begrænsninger og muligheder. De to grundtilstande i dette produkt er “død” og “levende”, de er altafgørende for en spillers muligheder. Hvis spilleren er død, kan vedkommende hverken skyde eller bevæge sig. Hvis spilleren er levende er det undertilstandene der er afgørende for, hvad vedkommende har mulighed for at gøre.

En af undertilstandene en spiller kan komme i, er at være løbet tør for ammunition; her vil spilleren ikke kunne skyde, så det er en begrænsende tilstand. Mens at tilstanden, hvor han har ammunition, er en muliggørende tilstand. Hvis der er et uigennemtrængeligt objekt foran spilleren, kan spilleren ikke gå frem, men hvis spilleren står lidt skrå på objektet kan spilleren “glide” langs kanten.

Tilstandsdiagram for modstanderen

Den kunstige intelligente modstander kan komme i mange tilstande. På figur 7.1 vises et tilstandsdiagram for sammenhænge mellem de forskellige tilstande som en modstander kan komme i. Et sådant diagram er vigtigt for at få struktur på modstanderens handlinger, så det i sidste ende bliver mere overskueligt at programmere den kunstige intelligens.



Figur 7.1: Tilstandsdiagram for den kunstige intelligente modstander

På figuren er der vist, hvordan tilstandene for den kunstige intelligente modstander ser ud. Der er en start og slut tilstand, som hhv. er kaldet *bliver født* og *bliver dræbt*. Midt i mellem disse to, er der en *i live* tilstand, hvori der er et diagram over, hvilke undertilstande modstanderen kan gå i.

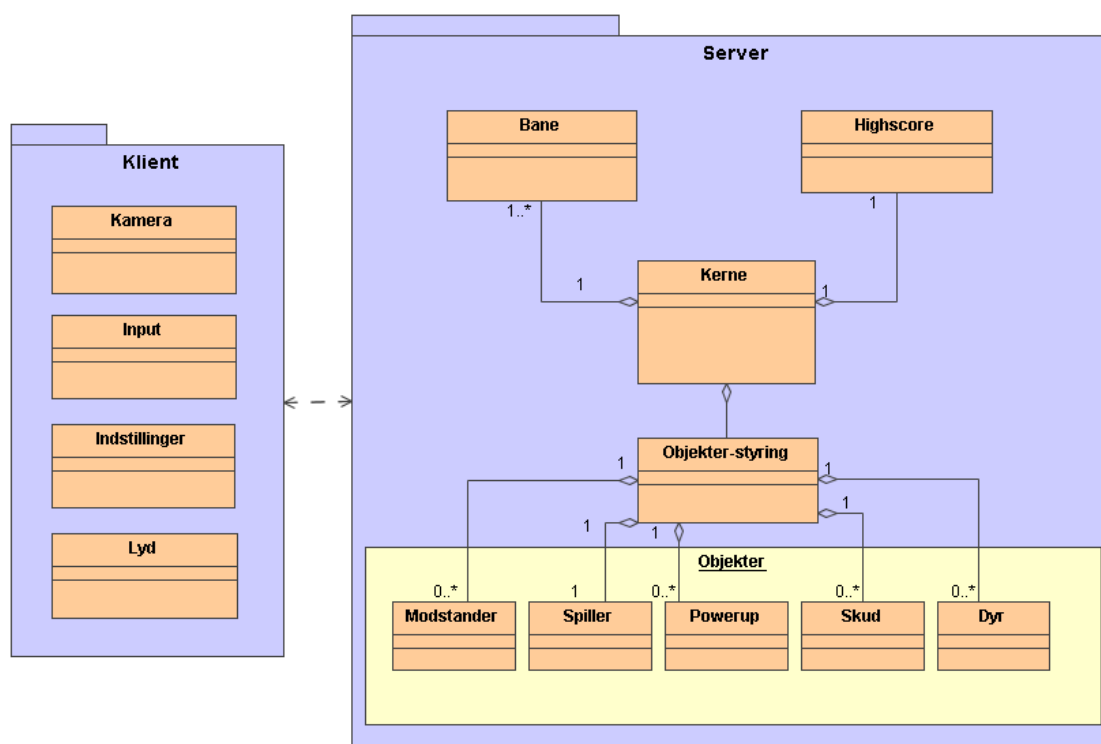
Den sorte cirkel repræsenterer den initierende tilstand for modstanderen. Efter den initierende tilstand i *i live* kommer den til *valg*. I denne tilstand skal modstanderen vælge, hvad den skal foretage sig. Udfra *valg*, kan modstanderen komme i to andre tilstande som er *strejf omkring* og *lede efter powerup*, alt efter hvad modstanderen har af liv og ammunition. Hvis modstanderen har fuldt i alt, eller er over den givne fuzzy logic grænse, vil modstanderen så begynde at gå

tilfældigt rundt på banen. Hvis modstanderen når under fuzzy logic grænsen, så vil denne ende i tilstanden *leder efter powerup*, hvor modstanderen vil lede efter powerup indtil, at fuzzy logic summen er over eller lig med fuzzy logic grænsen. Efter at denne tilstand er færdig, ender modstanderen igen i tilstanden *valg*.

Fra *strejf omkring* kan modstanderen komme i tilstandene *ffjende i syne* eller *lede efter powerup*. Modstanderen kommer kun i tilstanden *lede efter powerup*, hvis den fx ser et våben som den ikke har i dens våbensortiment. Hvis modstanderen får øje på spilleren, vil denne komme i tilstanden *ffjende i syne*, hvor den vil tage et valg, om spilleren skal jagtes eller flygtes fra. Her kommer fuzzy logic summen og grænsen igen ind i billedet, da alt efter hvad summen er i forhold til grænsen, har en påvirkning på, hvad den næste tilstand bliver.

Hvis summen er under grænsen, vil modstanderen gå i tilstanden *flugt*. Hvis den slipper væk fra spilleren, vil den ende i tilstanden *valg*. Hvis summen er større en grænsen, vil modstanderen gå i tilstanden *jagt*, hvor denne vil jagte spilleren, indtil at spilleren er død, eller den selv er blevet hårdt såret. Hvis modstanderen bliver meget såret, så fuzzy logic summen er under grænsen, vil den gå over i tilstanden *flugt*. Hvis det lykkedes for modstanderen at slå spilleren ihjel eller spilleren slipper væk, vil modstanderen gå over i tilstanden *valg*, hvor den vil foretage et nyt valg, og dermed finde ud af hvad den skal gøre.

7.4 Klient-server arkitektur



Figur 7.2: Klient-server model over arkitekturen

Figur 7.2 illustrerer hvorledes objekterne kan inddeles i klient- og server-moduler. Denne inddeling kan give større skalerbarhed mht. fx senere udvidelse til et multiplayer-spil. Der er valgt en tynd klient, idet serveren skal holde styr på alle spillets regler, for at forhindrer uregelmæssige afvigelser fra klient-siden. Dette kan dog medføre større belastning af servermodulet, og modsat mindre belastning for klientmodulet.

Klientmodulet indeholder kun brugerens input/output-muligheder og evt. individuelle indstillinger, så som hvorvidt spillet skal køre i fuldskærm osv. Med input tænkes på tastatur, og med output tænkes på grafikken, og videresendelse af denne til skærmen.

Servermodulet står for samling af alle objekter, banelementer og highscore, i den del der kaldes "Kerne". Aggregeringerne svarer til de der tidligere blev illustreret på figur 6.1. Her er der blot blevet lavet et klient-server snit, der gør at ikke alle aggregeringspile kan vises. Der er derimod blevet lavet en forbindelse mellem klienten og serveren.

7.5 Vejfinding

Dijkstras algoritme

Dijkstras algoritme tager en vægtet graf*, et start- og et slutpunkt som input. Ud fra disse input beregner algoritmen den korteste vej fra start- til slutpunktet.

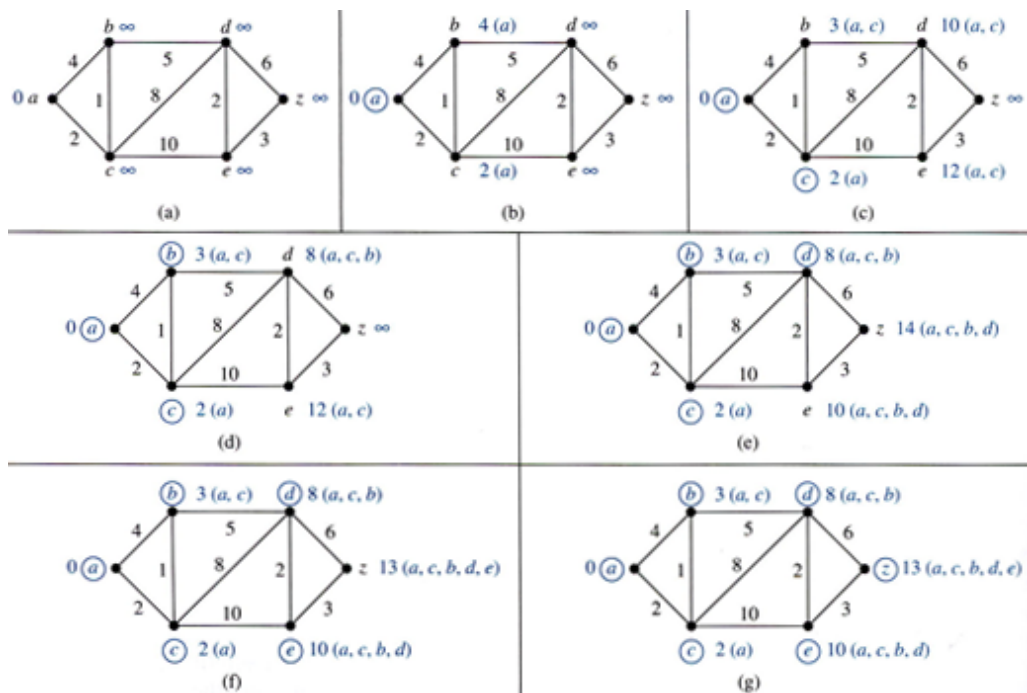
```

procedure Dijkstra(G:weighted connected simple graph, with all weights positive)
  for i := 1 to n
    L(vi) := infinity
  L(a) := 0
  S := ∅
  while z != S
  begin
    u := a vertex not in S with L(u) minimal
    S := S united {u}
    for all vertices v not in S
      if L(u) + w(u,v) < L(v) then L(v) := L(u) + w(u,v)
  end {L(z) = lenght of a shortest path from a to z}
    
```

Det første algoritmen gør er at initialisere alle punkterne til tomme værdier. Længden hen til startpunktet i grafen bliver sat til 0. S kan beskrives som ruten mellem 2 punkter.[14]

Herefter sættes en while-løkke i gang, hvor u bliver sat til det nabopunkt, der har den mindste vægt. u adderes til mængden S, som indeholder den midlertidige rute.[14]

For-løkken gennemløbes for alle de punkter, der ikke er med i S. Herved findes den korteste vej til næste punkt i grafen.[14]



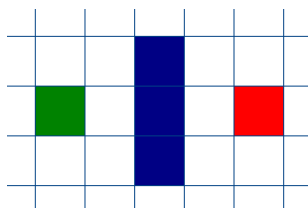
Figur 7.3: Figur over en vægtet graf.[14]

Dijkstras algoritme til vejfinding vil blive forklaret med udgangspunkt i den vægtede graf på figur 7.3. Algoritmen skal finde den korteste vej fra a til z. Algoritmen starter i a, hvor den går ud af de kanter, som fører væk fra a. Så vurderes der, hvilken rute som er kortest, i dette tilfælde c. Herefter bliver afstanden fra knuden a til knuden c opdateret. Ruterne fra c bliver vurderet, og igen bliver den korteste rute valgt og lagt til den nye knude. Dette bliver den ved med, indtil

endepunktet nåes. En anden måde at forklare det på er, at Dijkstra tjekker alle veje til slutpunktet, hvorefter den vælger den korteste vej.

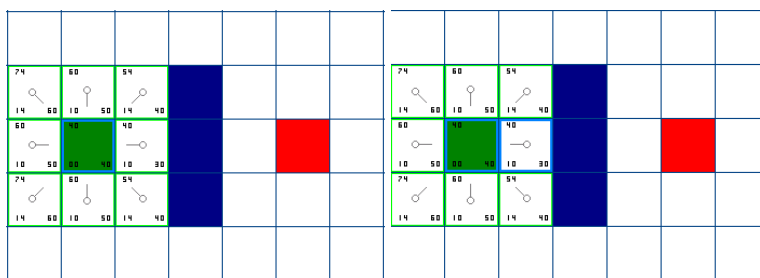
A*-algoritmen

Der er blevet lavet en udbygget vejfindingsalgoritme baseret på Dijkstras, som kaldes A*. Algoritmen finder ikke alle ruter til slutpunktet, men arbejder sig imod punktet. Dvs. at hvis algoritmen finder ud af, at ruten vil blive for lang, vil den ikke beregne mere på denne.



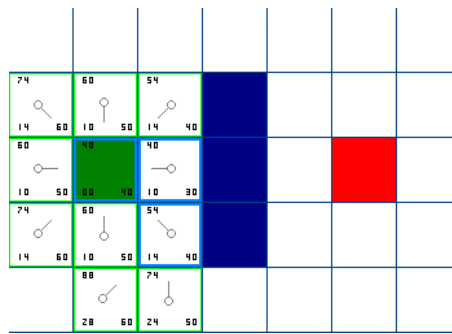
Figur 7.4: Figur over to punkter, hvor den firkant til venstre er startpunktet, mens den til højre er slutpunktet.[15]

Med udgangspunkt i figur 7.5 følger her et eksempel på hvordan A* algoritmen fungerer.



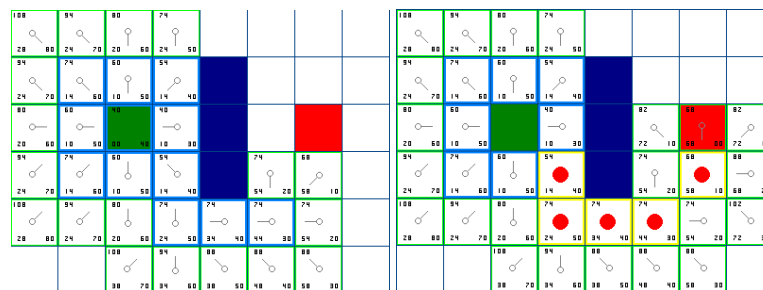
Figur 7.5: Starter vejfinding samt næste skridt i vejfindingen.[15]

På 7.5 er A* begyndt på, at finde den korteste vej til slutpunktet. I hvert punkt beregnes tre tal, der bruges til at vægte punkterne under vejfindingen. G er den distance der er gået for at komme til dette punkt. I dette tilfælde er vægten 10, hvis der bevæges vandret eller lodret, men 14 hvis der bevæges skråt. Som det sidste er H den estimerede afstand fra nuværende punkt til slutpunkt. F beregnes som G+H, og er den vægt, som algoritmen prioriterer punkterne efter, dvs. at jo mindre tallet er desto mere vil A* gå gennem dette punkt. Da feltet, med markeret kant, er det felt med mindst F, tager A* i dette eksempel, sit første skridt ad dette felt.



Figur 7.6: Andet skridt er udregnet - udenom muren.[15]

En mur, de farvede firkanter i midten, er i vejen for A*, så den direkte vej mod slutpunktet er blokeret. Da begge veje uden om muren er stort set ens, så er det bare et valg, hvor A*, i dette tilfælde, tager den nederste vej, som det kan ses på figur 7.5.



Figur 7.7: Hele ruten er blevet udregnet.[15]

På figur 7.5 kan det ses, at A* bliver ved med at lave udregninger til den korteste rute er opnået. A* finder, ligesom Dijkstra, den korteste vej, men afsøger oftest et mindre område, da den medtager den estimerede afstand til slutpunktet i de værdier der bruges til at prioritere punkterne i ruten. Dette gør også at A* kræver at slutpunktet kendes på forhånd hvor imod Dijkstra bliver ved med at lede indtil den finder et punkt der opfylder kravene for et slutpunkt.

Kapitel 8

Implementation

I dette kapitel vil produktet blive dokumenteret. De teknologier der bliver brugt i produktet vil blive introduceret i begyndelsen af kapitlet. Herefter vil de centrale dele af koden blive gennemgået og forklaret. Til sidst vil det færdige produkt blive gennemgået og vurderet.

8.1 Introduktion til valgte teknologier

Til udviklingen og afviklingen af produktet bliver der brugt flere teknologier. Herunder vil disse blive præsenteret.

8.1.1 C++

C++ er et programmeringssprog, udviklet af danskeren Bjarne Stroustrup i 1983. Det udspringer af det ældre programmeringssprog C. I forhold til C, er det gjort meget ud af at gøre sproget objektorienteret. Netop objekt-orienterede udvikling vil blive benyttet til det endelige produkt til dette projekt.[16]

8.1.2 SDL

SDL er et krydsplatform multimedie bibliotek. Denne teknologi fungerer som et bindeled mellem et program og en platforms API*-kald og input/output-enheder, bl.a. lyd, grafik, tastatur og mus.[17]

I det endelige produkt bliver SDL brugt til viduestyring og til at holde styr på tiden. Disse elementer ville normalt blive håndteret af API-kald, men med SDL er funktionerne mere simple, og så virker de også på flere platforme.[17]

8.1.3 OpenGL

OpenGL er en teknologi der er beregnet til at tegne 3D-grafik. Grundet den store udvikling inden for 3D-acceleration, på hardware markedet, kan teknologier som disse med fordel anvendes til 2D-grafik i dag, hvor der blot ikke tages højde for den 3. dimension. Når 3D-grafik nævnes, skal der bemærkes at modelleringen foregår i 3 dimensioner, hvorefter der beregnes hvordan grafikken skal projekteres på en 2D-skærm.[18]

OpenGL bliver i produktet brugt til at tegne alt det visuelle. Der bliver tegnet firkanter, hvor der bliver "trukket" en tekstur ud over. Alle elementerne på skærmen bliver tegnet på denne måde. OpenGL bliver også brugt til at holde styr på hvilke elementer der skal tegnes over hinanden.[18]

8.1.4 OpenAL

OpenAL er en teknologi der genererer 3D-lyd ud fra lydkilder der placeres i 3D-koordinater, som tilsvarende OpenGL's koordinat-system. Dette kan medvirke en større indlevelse i spillet, uanset om brugeren har stereo-, eller surround-højtaler.[19]

I det færdige produkt bliver det muligt at høre hvor de forskellige lyde kommer fra, da der bliver brugt OpenAL. Med denne teknologi bliver der placeret lyde i spilverdenen, så lydene kommer fra det sted hvor objektet er placeret i spillet.[19]

8.1.5 Afrunding

De fire teknologier understøttes af alle større platforme som Microsoft Windows og Linux. Dette kan medvirke, at det endelige produkt til dette projekt, bliver delvist platformsuafhængigt.

8.2 Centrale funktioner

8.2.1 Kernen

I alle programmer findes en *main()* funktion som bliver kaldt, når programmet startes. Denne funktion findes derfor også i spillet og kan betragtes som kernen.

Herunder ses dele af main-funktionen og -løkken:

```
main() {
    [...]<- Her initialiseres SDL, OpenGL, OpenAL og variabler

    boolRunning = true;
    boolMenu = true;
    while ( boolRunning ) {
        [...]<- Her tjekkes tasterne
        if(boolMenu) {
            [...]<- Her tjekkes og tegnes menuen
        } else {
            Father * CurrentObject;
            for(int i=0;i<MAX_OBJECTS;++i) {
                CurrentObject = Mother[i];
                if(!CurrentObject) {
                    continue;
                }
                if(CurrentObject->getNextThink() != 0
                    && CurrentObject->getNextThink() <= SDL_GetTicks()) {
                    CurrentObject->think();
                }
                CurrentObject->act();
            }
            destroyAllObjects();
            draw();
        }
    }
}
```

Det første der sker i denne funktion er initialisering af SDL, OpenGL, OpenAL og diverse globale variabler. Herefter startes main-løkken som bliver ved med at køre, indtil brugeren ønsker at stoppe spillet. Inde i løkken starter den med at tjekke, hvilke taster der er trykket ned. Dette bruges senere når spilleren skal flytte sig eller til at aktivere menuen. Herefter laves der et tjek, om menuen er aktiveret, og hvis det er tilfældet, så tjekker den menuen. Hvis ikke menuen er aktiveret gennemløbes alle objekterne, som eksisterer i spillet, og deres tænke- og handlingsfunktioner (*think()* og *act()*) bliver kørt. Dog “tænker” objekterne kun, hvis de er sat til at skulle tænke, og det er tid til en ny tanke. Disse to funktioner vil blive forklaret senere. Efter at alle objekterne er blevet opdateret køres først slettefunktionen (*destroyAllObjects()*), som fjerner alle “døde” objekter fra hukommelsen. Herefter køres tegnefunktionen (*draw()*), som tegner et nyt billede ud fra de opdaterede objekter, og sender det derefter til skærmen. Hele main-løkken starter til sidst forfra igen.

8.2.2 Father-objektet

Der findes mange forskellige objekter i produktet. En nem måde at tilgå objekternes fællesfunktioner, som *think()* og *act()*, er at lade alle være nedarvet fra det samme objekt. Dette objekt kaldes i dette produkt for *Father*. *Father* indeholder variabler og funktioner som alle objekter skal have. Fx objektets id, position, retning og billede. De fleste funktioner i *Father* eksisterer for at tilgå de private variabler som normalt ellers ikke kan blive tilgået. Men der findes også nogle vigtige virtuelle funktioner kaldet *spawn()*, *die()*, *think()*, *act()* og *collide()*. Disse

funktioner er virtuelle så et objekt kan lave sin egen version af disse og derved selv bestemme, hvad der skal ske. Objekterne behøver ikke selv at lave dem, da de allerede er i *Father*. *spawn()* og *die()* bliver brugt til at “føde” og “dræbe” objektet. *think()*, *act()* og *collide()* er lidt mere avanceret og vil blive forklaret herunder.

think()

think() er en funktion, der bliver kaldt hver gang et objekt skal tænke. Objektet fortæller først systemet, at den vil tænke på et bestemt tidspunkt i fremtiden ved hjælp af *setNextThink()* funktionen. I main-løkken tjekker den så denne værdi og sammenligner med det nuværende tidspunkt. Hvis *NextThink* er overskredet kaldes *think()* og objektet tænker. Dette bruges på AI'en til decideret at tænke over de situationer den måtte være i. Men den bruges også til at stoppe et objekt såsom kuglerne.

Dette er et udsnit fra main-løkken, hvor den tjekker, om det er tid til at tænke, og hvis det er, så kalder den *think()*. Det første tjek er om objektet i det hele taget skal tænke.

```
if(CurrentObject->getNextThink() != 0
&& CurrentObject->getNextThink() <= SDL_GetTicks()) {
    CurrentObject->think();
}
```

act()

act() funktionen kaldes hvert eneste frame* på alle eksisterende objekter. Dog efter at objektet har “tænkt”.

```
CurrentObject->act();
```

Den bruges primært til at flytte objekter som kan bevæge sig. Inde i funktionen vil der blive sat en *lastAct* værdi som holder styr på hvornår den sidst bevægede sig. Denne tid bruges i udregningen af hvor langt den skal bevæge sig. Dette gøres for, at et objekt altid flytter sig med samme hastighed på computere med lav framerate* som på en med høj framerate.

collide()

Når objekter bevæger sig rundt i verden kan de kollideres med hinanden. Hvis objekt A støder ind i objekt B, kaldes objekts B's *collide()*-funktion med objekt A som parameter. Denne situation bruges fx, når spilleren støder ind i en powerup. Her kaldes poweruppens *collide()*-funktion med spilleren som parameter, og den kan derefter give den bonus til spilleren og derefter deaktivere sig selv for at illustrere, at den er blevet taget.

8.2.3 Mother-arrayet

Alle de objekter der laves i spillet bliver gemt et mere eller mindre tilfældigt sted i hukommelsen. Derfor er det nødvendigt at samle alle objekter et sted, hvor de er nemme at tilgå. Dette er gjort ved at lave et stort array*, som indeholder pointers* til alle objekter, der bliver lavet. Dette array kaldes for *Mother*.

```
Father * Mother[MAX_OBJECTS]={NULL};
```

Mother indeholder en masse pointere til objekter af typen *Father*, men til at starte med er alle disse pointere *NULL*, hvilket svarer til, at de ikke peger på noget.

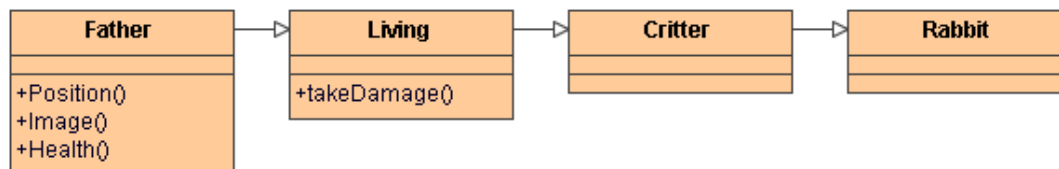
Objekterne skal selv sørge for at tilføje sig selv til *Mother* ved hjælp af funktionen *addToMother()*. Når objektet så skal fjernes igen, kalder den *removeFromMother()*. Objektet bliver kun fjernet fra *Mother* arrayet og ikke fra hukommelsen, til det formål bruges funktionen *destroyObject()*. Den tilføjer en pointer til *destroyQueue* som hvert frame bliver gennemløbet af *destroyAllObjects()*-funktionen, der sletter alle objekter i køen fra hukommelsen.

8.2.4 Rabbit-objekt

Som eksempel på et objekt, vil objektet *Rabbit* blive gennemgået. Hver kanin i spillet svarer til et *Rabbit*-objekt. Objektet kræver et (x, y) -koordinatsæt og en tid som input. Tiden bruges som en forsinkelse af næste genoplivningstid, når kaninen dør.

```
class Rabbit : public Critter {
public:
    Rabbit(float xp, float yp, int spawntime);
    ~Rabbit();
    void spawn();
    void die();
    void think();
    void act();
    void stepForward();
    void collide(Father * object);
private:
    float start_x, start_y;
    int option;
};
```

På figur 8.1 ses hvordan *Rabbit*-objektet er nedarvet fra *Father* via *Living* og *Critter*.



Figur 8.1: Rabbit objektets nedarvning.

Indeholdt i *Rabbit*-objektet findes syv funktioner, som vil blive beskrevet kort herunder:

- *Rabbit()*, objektets constructor. Bliver kaldt når rabbit-objektet oprettes. I denne funktion nulstilles variabler for at sikre, at de har en bestemt værdi og ikke noget tilfældigt som allerede lå i hukommelsen.
- *~Rabbit()*, objektets destructor. Bliver kaldt når rabbit-objektet bliver slettet fra hukommelsen.
- *spawn()*, bruges til at genoplive en død kanin.
- *die()*, når en kanin dør, skal billedet af kaninen skiftes ud med et dødsbillede.
- *think()*, under hver handling skal kaninen tænke over næste træk. En tilfældighedsgenerator afgør, om kaninen skal bevæge sig.
- *act()*, udfører en handling, hvis resultatet af *think()* tillader dette.

- *stepForward()*, sætter kaninen i bevægelse i en bestemt retning.
- *collide()*, bliver kaldt når et andet objekt kolliderer med kaninen. Hvis det kolliderende objekt er spilleren eller en modstander, dør kaninen ved at kalde *die()* funktion.

8.2.5 Grafik

Som tidligere nævnt, bliver den grafiske side af spillet lavet via OpenGL. *draw()*-funktionen bliver kaldt i hver eneste hovedløkke, og afhængigt af hastigheden på computeren, den kører på, så vil denne derfor blive kaldt 100-500 gange i sekundet. Hver gang den bliver kørt, bliver billedet på skærmen opdateret, hvilket vil sige, at jo oftere denne funktion har mulighed for at blive kørt, desto mere flydende foregår bevægelserne på skærmen. Det er derfor vigtigt, at denne funktion kan gennemføres så hurtigt som muligt. Hoved-elementerne i denne funktion, og rækkefølgen disse udføres i, forklares herefter.

Før selve billedegenereringen kan påbegynde, skal OpenGL-bufferen, som der tegnes i, opsættes og "renses". Herefter tegnes banen felt for felt, inden for en hvis margin omkring spilleren:

```
// Tegn mappet
int i, j, margin;
int x, y;
//Angiv hvor mange tiles der skal tegnes rund om spilleren,
//da det ikke er nødvendigt at tegne hele banen.
margin=(int)(SCREEN_WIDTH/1.2)/TILE_SIZE;
//Gennemløb de nødvendige tiles
for (i=((int)g_Player->getYPos()-margin;i<((int)g_Player->getYPos()+margin;i++) {
    for (j=((int)g_Player->getXPos()-margin;j<((int)g_Player->getXPos()+margin;j++) {
        [...] <- Tegn tile j,i med funktionen drawTile(...)
    }
}
```

Som tidligere nævnt, så gemmes referencer til alle spillets objekter i det array der kaldes *Mother*. For at de objekter, der skal tegnes, bliver tegnet, gennemløbes dette array, og de nødvendige objekter tegnes:

```
// Tegn objekterne i spillet
Father * object;
//Gennemløb mother-arreyet, som indeholder pointers til alle objekter.
for(int i=0;i<MAX_OBJECTS;++i) {
    object = Mother[i];
    // Eksisterer dette objekt?
    if (!object) { continue; }
    // Har den overhovedet en grafik?
    if (!object->getImage()) { continue; }
    drawObject(object);
}
```

Til sidst, bliver HUD'en tegnet. Dette statiske display viser spillerens ammunition, point, liv og nuværende våben. Når alle elementer er tegnet i OpenGL-bufferen, så bliver *SDL_GL_SwapBuffers()* kaldt. Denne funktion bytter nuværende skærbillede ud med det, der er tegnet i OpenGL-bufferen.

Når grafiske elementer tegnes i OpenGL, tegnes de som en tekstur trukket ud over fx en firkant. Disse firkanter tegnes i forskellige z-kordinater, der så endeligt bestemmer hvilke firkanter, der ligger over andre - dvs. f.eks. at objekterne ligger over banen. Det er herved ikke rækkefølgen de tegnes i, i *draw()*, som afgør hvad der ligger over hvad.

Her er et eksempel på, hvordan et felt bliver tegnet med OpenGL:

```
glBindTexture(GL_TEXTURE_2D, texture);
glBegin(GL_QUADS);
glTexCoord2f(0,0); //Top-left vertex (corner)
glVertex3f(x, y, z); //Top-left
glTexCoord2f(1,0); //Top-right vertex (corner)
glVertex3f(x+1, y, z); //Top-right
glTexCoord2f(1,1); //Bottom-right vertex (corner)
glVertex3f(x+1, y+1, z); //Bottom-right
glTexCoord2f(0,1); //Bottom-left vertex (corner)
glVertex3f(x, y+1, z); //Bottom-left
glEnd();
```

Variablen *texture* indeholder et indeks til den tekstur, der skal tegnes. Variableerne *x*, *y* og *z* indeholder det nuværende felts placering.

8.2.6 Kollision

Omgivelserne i spillet er meget forskellige, dette angår også, hvad der kan og ikke kan betrædes. Da banen er opbygget af tiles, vil det første være at tjekke, om den givne tile kan betrædes, eller om den ligger uden for banens grænser. Til sidst bliver der tjekket, om der er en kollision med andre objekter.

Banekollision

```
bool isWalkableTile(int x, int y) {
    if (x<0 || x>=MAP_WIDTH) {return false;}
    if (y<0 || y>=MAP_HEIGHT) {return false;}
    return aWalkableTiles[aMap[x][y]];
}
```

Denne funktion tager en tile-position ind og returnerer, om denne tile kan betrædes. Den tjekker først, om positionen ligger inden for banen. Funktionen bruger derudover to array's for at se, om tilen kan betrædes. Det inderste array indeholder banen, her er hver enkelt tile angivet som ét tegn. Dette tegn "sendes" nu videre til det andet array, der indeholder oplysninger om hvilke typer tiles der kan betrædes.

Objektkollision

```
bool objectCollision(Father*objectOne,Father*objectTwo,float temp_x, float temp_y) {
    float distance,radiusdistance;
    radiusdistance = objectOne->getRadius() + objectTwo->getRadius();
    distance = sqrt(square(objectTwo->getXPos()-temp_x) +
        square(objectTwo->getYPos()-temp_y));
    if (distance < radiusdistance ) {
        return true; // Collision!
    } else {
        return false; // Way off!
    }
}
```

Hvert objekt har en radius, så kollisionstjekket skal blot undersøge, om de objekters radier lagt sammen er mindre end afstanden mellem dem. Hvis dette er tilfældet, så kolliderer de to objekter. *temp_x* og *temp_y* bliver brugt for midlertidigt at flytte det ene objekt. Dette er vigtigt, da tjekket laves før objekterne flyttes.

8.3 Algoritmer til AI

8.3.1 Ændringer i tilstande

I det færdige produkt bliver fuzzy logic brugt til at afgøre vigtige valg. Det bliver brugt til at bestemme, om modstanderen skal angribe eller flygte, om modstanderen skal lede efter powerups, eller om den bare skal strejfe tilfældigt rundt. Den fuzzy logic, der bliver brugt, tager kun højde for liv og ammunition. Der er angivet tre trapetzer for liv og tre for ammunition. Det er vidt forskellige trapetzer og vægte for de to værdi-typer.

For at kunne bruge fuzzy logic skal der laves en “defuzzification”, hvilket består af en sammenligning af en speciel sum.

```

values[FUZZY_HEALTH] = object->getHealth();
values[FUZZY_AMMO] = object->getAmmo();

sum=0;
for( i=0 ; i<FUZZY_TYPES ; i++ ){
    for( j=0 ; j<FUZZY_TRAPS ; j++ ){
        if( values[i] > fuzzyTrap[i][j].x[0]
            && values[i] < fuzzyTrap[i][j].x[3] ){
            /*ligger inde i trapetsen /\ */
            if( values[i] < fuzzyTrap[i][j].x[1] ){
                //første interval /\
                sum += ( (values[i] - fuzzyTrap[i][j].x[0])
                        / (fuzzyTrap[i][j].x[1] - fuzzyTrap[i][j].x[0])
                        ) * fuzzyTrap[i][j].weight;
            } else if( values[i] <= fuzzyTrap[i][j].x[2] ){
                //anden interval /\
                sum += fuzzyTrap[i][j].weight;
            } else {
                /*tredje interval /\ */
                sum += ( 1 - (values[i] - fuzzyTrap[i][j].x[2])
                        / (fuzzyTrap[i][j].x[3] - fuzzyTrap[i][j].x[2])
                        ) * fuzzyTrap[i][j].weight;
            }
        }
    }
}

```

Dette er et uddrag af koden fra fuzzy logic-summen. Denne kode gennemløber de forskellige værdityper, den skal tage højde for, her liv og ammunition. Der er blevet lavet et dobbelt array af trapetzer, første indeks holder styr på hvilken værdi-type, trapetzen beskriver, mens andet indeks holder styr på, hvilket nummer trapetz det er. Der er derfor lavet en dobbelt *for*-løkke, der gennemløber disse to indekser.

if/else-sætningerne finder ud af hvilken del af trapetzen værdien ligger i, såfremt den ligger indenfor trapetzen. Afhængigt af hvor i trapetzen, værdien passer ind, bliver outputtet beregnet forskelligt. Outputtet bliver herefter ganget med vægten for trapetzen, før den til sidst bliver lagt til summen.

8.3.2 Vejfinding

Til implementationen af vejfinding i produktet, benyttes A*-algoritmen med det forbehold, at hvis koordinaterne til destinationen ikke er kendt, så vil H-værdierne konstant være 0. Dette medfører at A* i disse tilfælde vil fungere som Dijkstra-algoritmen. Dette vil være tilfældet, når der skal findes powerups, da der ikke kan vides, hvilken powerup der kan nås via den korteste vej. Når fx vejen til spilleren skal findes, benyttes A*-algoritmen korrekt med H-værdier der tilsvarende den løbende estimerede afstand til målet, beregnet via pythagoras ($\sqrt{(x1 - x2)^2 + (y1 - y2)^2}$). Som waypoints benyttes centrum af de samme felter som banen er inddelt i.

A*-algoritmen benytter løbende to forskellige lister: "openlist" og "closedlist". Når A* startes, tilføjes start-feltet til openlist. Herefter påbegyndes hovedløkken for denne algoritme[15]:

1. Find det felt med laveste F fra openlist, og lad denne være udgangspunktet for dette gennemløb.
2. Fjern den fra openlist, og tilføj den til closedlist.
3. For hver af de nærliggende 8 felter, gøres følgende:
 - (a) Hvis feltet ikke er gennemtrængeligt, eller på closedlist, så ignoreres det.
 - (b) Hvis feltet ikke er på openlist, så tilføj det. Opret en "foreldre"-reference til udgangspunktet. Beregn feltets G og H værdier.
 - (c) Hvis feltet i forvejen er på openlist, tjek så om denne rute til feltet giver en mindre G værdi, end den nuværende. Hvis det er tilfældet, opdater så både feltets "forældre"-reference og G-værdi.
4. Stop når destinationen tilføjes til closedlist (ruten blev fundet), eller når openlist er tom (ruten blev ikke fundet).
5. Hvis ruten blev fundet, findes denne ved at gennemgå "forældre"-referencerne baglæns, startende fra destinationsfeltet.

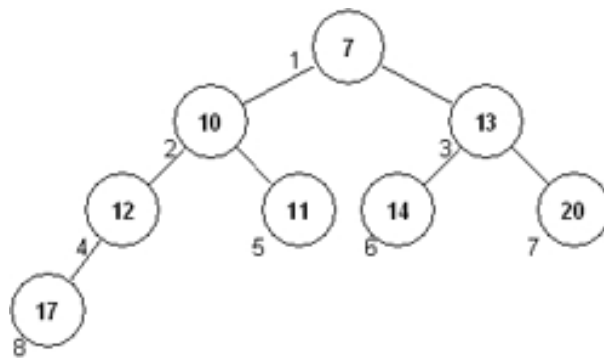
Under afviklingen af algoritmen, er det vigtigt konstant at have mulighed for at tage udgangspunkt i det felt med den laveste F-værdi fra den såkaldte "openlist". Da det kun er vigtigt at kende det felt med den laveste værdi, er det ikke nødvendigt at resten af felterne i openlist er sorteret. Derfor kan en "binary heapsort"-algoritme benyttes. Denne algoritme kan gøre, at det element med den laveste F-værdi altid vil være let tilgængelig, hvorimod resten af elementerne er usorterede. Listen kan arrangeres i et array startende med indeks 1. For at indeks 1 altid indeholder det element med den laveste værdi, tilføjes et nyt element således:

Placér det nye element bagerst i arrayet. Byt om på det pågældende element og dets forældre-element indtil forældre-elementet har den mindste værdi. Indekset til et foreldre-element vil altid være $elementets\ indeks / 2$.

Modsat, forgår en sletning af et element således:

Fjern elementet i indeks 1 og flyt det bagerste element herop. Byt om på dette element og det mindste af dets børn, så længe elementet er større end et af børnene. Indekserne til elementets børn vil altid være hhv. $elementets\ indeks * 2$ og $(elementets\ indeks * 2) - 1$.

Herunder vises hhv. en illustration af et binary heapsortet træ, og et udsnit af de essentielle dele af vejfindingsalgoritmen i produktet. Struktureringen følger ovenstående gennemgang af algoritmen.



Figur 8.2: Eksempel på et binary heapsortet træ.

```

pathTarget * findPath(Enemy * enemyObj, targetType tType, float xPos, float yPos){

  [...] <- Initialisering af variabler
  //Tilføj start-punktet til openlist
  oList->addListItem((listItem){currentX,currentY});

  //Hoved-løkken
  while(1){
    if(!oList->getListIndex()){
      [...] <- openlist er tom, dvs. ingen rute fundet.
    }
    //Find punkt med bedste F-værdi, slet denne fra openlist,
    //og tilføj til closedlist
    listItem li = oList->getListItem();
    oList->removeListItem();
    waypoints[li.x][li.y].onClosedList=true;

    if(isDestination(li.x,li.y,tType)){
      [...] <- Ruten blev fundet, så ruten gennemløbes baglæns, og returneres.
    }
    //Gennemløb de 8 nærliggende felter
    for(int i_x=li.x-1;i_x<=li.x+1;i_x++){
      for(int i_y=li.y-1;i_y<=li.y+1;i_y++){
        [...] <- Diverse tjek om tile: Uden for banen?
        [...] <- Ugennemtrængelig? På closedlist? Skæres hjørne?

        if (!waypoints[i_x][i_y].onOpenList) {
          [...] <- G og H kalkuleres, og ``forældre``-referencen angives.
          oList->addListItem((listItem){i_x, i_y});
        } else{
          //G rekalkuleres, og tjekkes op med den eksisterende G-værdi.
          float tmp_float = waypoints[li.x][li.y].gCost +
            ((i_x==li.x)||i_y==li.y)?1:M_SQRT2);
          if (tmp_float < waypoints[i_x][i_y].gCost) {
            [...] <- Ny G gemmes, og ``forældre``-referencen opdateres.
          }
        }
      }
    }
  }
  [...] <- Oprydning og returnering
}

```

Kildekoden til vejfindingsalgoritmen i dette projekt.

8.3.3 Jagt

Når Modstanderne jager spilleren, skal de både kunne sigte for at ramme, men de skal også vælge det rigtige våben til situationen. Det er ikke nok at sigte direkte på spilleren, da spilleren sikkert bevæger sig, derfor bliver modstanderne nødt til at være lidt taktiske.

Tage sigte

Modstanderens taktiske evner er blevet begrænset til at kunne sigte foran spilleren. Koden ser således ud:¹

```
float aim(Father * target, Father * shooter, float bulletSpeed){
    // Shooter og Target X og Y
    float sx, sy, tx, ty;
    sx = shooter->getXPos();
    sy = shooter->getYPos();
    tx = target->getXPos();
    ty = target->getYPos();

    // Target hastighed
    Player * player_target = static_cast<Player*>(target);
    float targetSpeed = player_target->getSpeed();
    int moving = player_target->getMoving();

    // Vs (Vinklen mellem Shooter og Target)
    float Vs; // grader!
    float sctx = sx-tx;
    if(sctx>0) {
        Vs = 180.0+atan((sy-ty)/(sctx))*(180/M_PI);
    }else if(sctx<0) {
        Vs = atan((sy-ty)/(sctx))*(180/M_PI);
    }else{
        Vs = (ty>sy ? 90.0 : -90.0);
    }

    // Vt (Vinklen for Target)
    float Vt = target->getAngle();

    // Alpha
    float alpha = targetSpeed/bulletSpeed;

    // Phi
    float phi = Vt - Vs;

    // VINKLEN!!!
    return Vs + (moving * asin(alpha*sin(phi*(M_PI/180)))*(180/M_PI));
}
```

Denne algoritme beregner sigtevinklen til det punkt, målet er på vej hen til. Det vil sige at hvis målet bevæger sig langs en ret linie, i den periode fra der bliver skudt til kuglen rammer, vil denne algoritme altid give den rigtige vinkel for at ramme målet. Denne algoritme vil ikke blive bevist eller yderligere forklaret, da den ikke er lavet af gruppen.

Våbenvalg

En af de færdigheder, som AI'en er blevet tildelt, er muligheden for at ændre det nuværende våben, således at det rigtige våben er valgt til den givne situation. Der bliver brugt et ekspert system til at skifte våben med. Den gør dette alt efter, hvilken afstand der er til målet, hvor den fx ved lang afstand vælger at bruge riflen og shotgun på kort afstand.

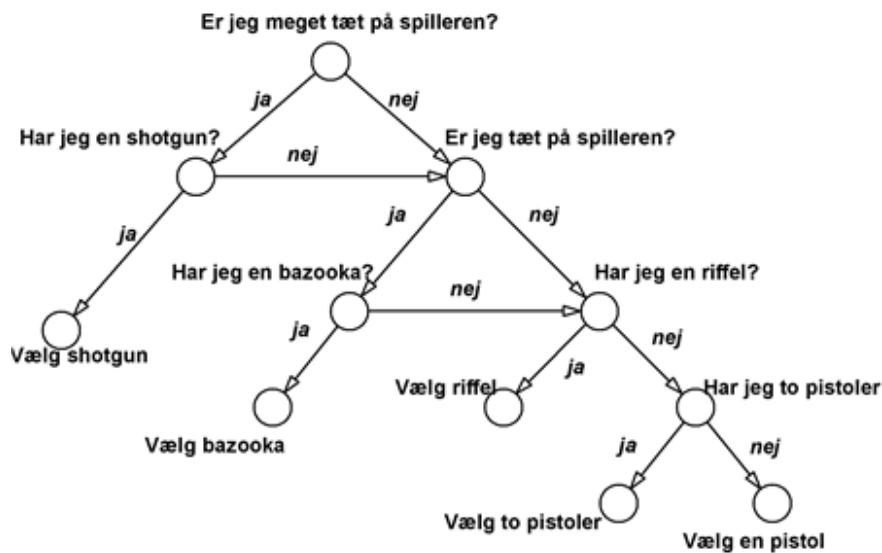
¹Denne algoritme er fundet på et forum til Game Maker: <http://forums.gamemaker.nl/index.php?showtopic=183979>

```

void Enemy::pickWeapon(float dist){
    if(dist < 5 && wb.wb_shotgun){
        setCurrentWeapon(weapon_shotgun);
    }else if(dist<7 && wb.wb_bazooka){
        setCurrentWeapon(weapon_bazooka);
    }else if(wb.wb_rifle){
        setCurrentWeapon(weapon_rifle);
    }else if(wb.wb_dual_pistols){
        setCurrentWeapon(weapon_dual_pistols);
    }else{
        setCurrentWeapon(weapon_pistol);
    }
}

```

I de to første *if*-sætninger bliver der tjekket, hvad afstanden er mellem modstanderen og spilleren. I alle *if*-sætninger bliver der også tjekket, om modstanderen har det givne våben, som ønskes brugt. Hvis dette er tilfældet, skifter modstanderen automatisk våben. Hvis afstanden kommer ud over en hvis grænse, vil modstanderen enten bruge riffel, dobbelt pistol eller pistol, alt efter hvad der er til rådighed. Disse valg er yderligere illustreret på figur 8.3 som et ekspertsystem.



Figur 8.3: Figur over, hvordan våbenvalg er opbygget som et ekspertsystem

8.4 Debug

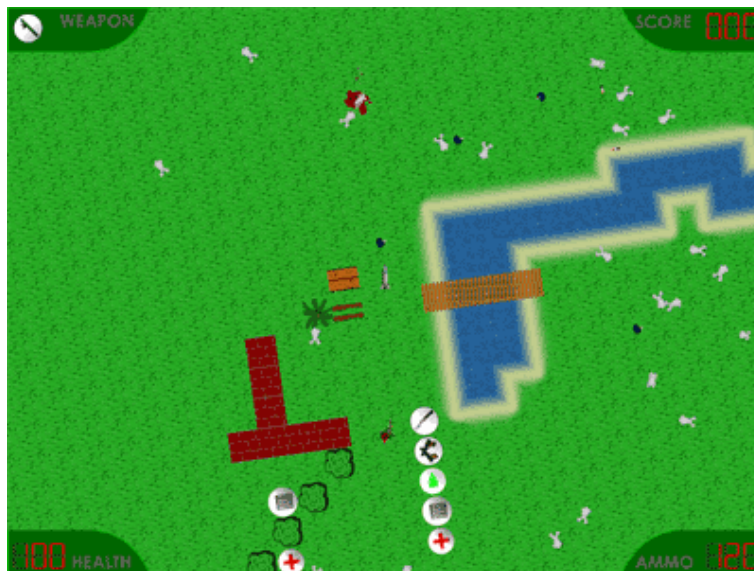
Under udvikling af spillet er der løbende behov for at teste spillet for fejl. I tilfælde af fejl, skal disse findes og rettes. Derfor bruges der en debug*-tilstand, der udskriver en bestemt tekst til en tekstfil, når spillet gennemløber en bestemt funktion eller position i koden. Som eksempel kan debug-tilstanden udskrive spillerens position i en bane, hver gang *Mother*-arrayet gennemløbes. Dette giver et overblik over, hvordan spilleren har flyttet sig.

Disse udskrifter sker til en ekstern fil, men der foregår også debug direkte i spillet. Fx visualiseres hvilken tilstand modstanderne er i, modstandernes skyderadius og den radius, de kan se spilleren indenfor. Disse debug-funktionaliteter giver en hjælp til at finde de fejl, der muligvis er opstået, og der kan ses, om modstanderen gør det forventede.

8.5 Gennemgang af produktet

I dette afsnit vil produktet blive beskrevet. Her vil der bl.a. blive beskrevet, hvordan spillet foregår, og hvordan spilleren bevæger sig rundt i spillet.

I kravsspecifikationen, afsnit 5, blev der opstillet en række krav til spillet. I dette afsnit, vil der også blive set på, hvordan disse er implementeret.



Figur 8.4: Skærbillede fra spillet.

8.5.1 Generelt

Spillet starter med en menu, hvor der er flere valgmuligheder. Her kan spilleren enten vælge sværhedsgrad for at komme igang, eller vælge *exit* for at forlade spillet. Denne menu fremkommer også, når spilleren trykker på *esc*-tasten. Hvis der vælges at spille, skifter skærbilledet over til at vise spillerens figur oppefra. Spilleren kan se længere fremad end bagud, og skærmen drejer sammen med spilleren, således at den fremadrettede synsvinkel altid er opad på skærmen. Spillet går ud på, at spilleren skal gå rundt i spilleverdenen og skyde diverse modstandere. Hver gang en modstander dør, får spilleren ét point. Banen består af forskellige elementer, som fx græs, mur og buske, hvor nogle af disse elementer er gennembrængelige og andre uigennemtrængelige. Spillet slutter, hvis spilleren bliver dræbt.

8.5.2 Head Up Display

HUD'en kan ses på figur 8.4. HUD'en er spillets måde at vise, hvilke værdier spilleren har i forskellige variabler. I spillet bliver der vist hhv. liv, ammunition, våbentype og point.

Dette krav er i hht. kravspecifikationen imødekommet.

8.5.3 Styring

I kravspecifikationen, afsnit 5.1.3, er det nævnt, at spilleren skal have mulighed for at bevæge sig rundt i spilleverdenen og skyde på modstanderne. For at bevæge sig bruger spilleren piletasterne, mens at vedkommende skal bruge *ctrl*-tasten til at skyde med. Spilleren har mulighed for at

bakke og bevæge sig fremad. Der er ingen mulighed for at bevæge sig til siderne, men kun mulighed for at dreje figuren.

8.5.4 Powerups

Rundt omkring i spilverdenen vil der forefindes powerups. Der er forskellige variationer af disse powerups. De kan enten give mere liv, mere ammunition, højere bevægelseshastighed for en periode eller et helt nyt våben. Når en af disse powerups er blevet taget, vil den give powerup forsvinde, men efter et stykke tid vil den dukke op igen.

8.5.5 Våben

De forskellige våben, som findes i spillet, har forskellige rækkevidder og giver forskellig skade. Der findes kun én type ammunition i spillet, som alle våbnene bruger. Alt efter, hvilket våben der er tale om, bliver der taget en vis mængde ammunition fra spilleren.

Her er en tabel over, hvilke våben der er i spillet, hvor meget ammunition våbnet tager fra spilleren når det affyres, hvor meget hvert skud skader og ladehastigheden:

Våben	Ammunition pr. skud	Kugler pr. skud	Skade pr. skud	Ladehastighed
Pistol	1	1	10	1 sekund
Dobbelt pistol	1	1	10	0.5 sekunder
Shotgun	10	10	7	1.5 sekunder
Riffel	1	1	3	0.08 sekunder
Bazooka	15	1 20	100 $\frac{100}{20}$	3 sekunder

Bazookaen er speciel i denne sammenhæng, da den i første omgang skyder en kugle afsted, men når denne kugle rammer noget bliver 20 kugler skudt ud i tilfældige retninger. Den første kugle skader 100 procent i liv, når den rammer, mens hver af de 20 kugler skader 1/20 af de 100 procent.

8.5.6 Modstandere

Der vil være modstandere i spilverdenen. Disse kan næsten det samme som spilleren. De kan samle powerups op, skyde og bevæge sig rundt i spilverdenen. Dog er der visse begrænsninger i forhold til spilleren. Modstanderne vil have en begrænset synsvidde, hvor spilleren derimod kan se hele sit skærbillede.

Hvis modstanderne får øje på spilleren, vil de jage vedkommende, indtil de ikke kan se spilleren mere. De vil også lave en vurdering af situationen: hvorvidt de skal angribe eller flygte, hvis de ser spilleren. Når en modstander dør, vil der efter et stykke tid, spawn en ny modstander i spilverdenen.

8.5.7 Omgivelserne

I spilverdenen er der forskellige former for vegetation og solide genstande, såsom mure, buske og vand. De solide genstande, kan spilleren ikke gennemtrænge, men skud kan flyve igennem vegetation og over vand, hvorimod et skud ikke kan flyve igennem en mur. Til gengæld kan spilleren gå på græs og henover broer. Disse regler gælder også for modstanderne i spillet. Selve banen er rektangulær, og hvis spilleren når kanten af banen, kan vedkommende ikke bevæge sig videre.

8.6 Vurdering

I dette afsnit vil der være en vurdering af det færdige spil i forhold til kravspecifikationen. Her vil der blive gennemgået, hvad der kan gøres bedre i spillet, og hvilke mangler der evt. er.

Det færdige produkt opfylder størstedelen af de opstillede krav fra kravspecifikationen, afsnit 5. De krav, som ikke er blevet opfyldt, er de lavest prioriterede krav i både funktionelle og ikke funktionelle krav. Dette er dog at forvente, da de højst prioriterede blev opfyldt først. Opfyldte krav:

Funktionelle krav

Display: Der er blevet brugt OpenGL til at tegne spilverdenen i den rigtige orden.

Brugrænseflade: Der bliver også brugt OpenGL til brugergrænsefladen.

Input fra brugeren: SDL bruges til at registrere tastetryk.

AI - vejfinding: Der er blevet brugt Dijkstra og A* til at finde vej.

AI - taktik: Dette krav er kun blevet delvist opfyldt, da den eneste taktiske funktion, som modstanderen har, er at kunne sigte foran eller bagved spilleren, så den derved rammer, når spilleren er i bevægelse.

Omgivelser: Til lyd er der blevet brugt OpenAL. Banen indlæses fra en tekstfil som indeholder tegn, der repræsenterer forskellige objekter.

Gameplay: Dette krav er opfyldt ved, at spillet fungerer som beskrevet i kravspecifikation.

Ikkefunktionelle krav

Platform: Spillet kan køre på Microsoft Windows XP.

Skalerbarhed: Dette krav er opfyldt, da der kan være flere baner, og der kan arbejdes med mange objekter på samme tid.

Brugervejledning: Kravet er opfyldt, da der er blevet lavet en brugervejledning til selve produktet, som medfølger i en fil.

Samtidighed: Der opstår kun samtidighedsproblemer, når spillet opdateres for langsomt. Dette opstår kun på computere, der ikke opfylder de listede krav om performance.

Performance: Spillet kører flydende på computere med de specifikationer, der er listet i afsnit 5.2.5.

Åbenhed: Under de ikke funktionelle krav, blev kravet om åbenhed kun delvist opfyldt. Dette skyldes, at når spillet bliver installeret, har brugerne kun mulighed for at ændre på billederne og ikke ændre på indstillinger eller på selve koden til spillet.

Den AI, som er blevet lavet til spillet, mangler lidt justering. Modstanderne er meget kujonagtige, da de kan finde på at flygte efter blot at være blevet skudt én gang. Når modstanderne flygter, kan det ske, at disse flygter til nogle meget upassende steder.

Grafikken er meget simpel, da den er lavet i 2D som beskrevet i kravspecifikationen. Grafikken fungerer som den skal, og spillet har fået sin egen stil mht. det grafiske.

Lydene i spillet er blevet retningsorienteret, så spilleren kan høre, hvor lydene kommer fra. De fleste af disse lyde er lavet af gruppen selv. Musikken er "Ieva's Polka" fra det finske band Loituma, og nogle af effekterne er blevet hentet fra internettet² Alle tilgængelige lyd-kilder, som den respektive pc's hardware understøtter, bliver ikke udnyttet i spillet. Det kan skabe problemer, hvis flere lyde bruger samme lyd-kilde. Det kan også ske, at nogle lyde, som ikke afspilles, hvis lyd-kilden er i brug.

Alt i alt er det det færdige produkt blevet et velfungerende spil, der opfylder de fleste af de opstillede krav. Det er blevet sendt ud til test og der er kommet konstruktive svar tilbage, som vil blive beskrevet i det efterfølgende kapitel.

²<http://shockwave-sound.com/>

Kapitel 9

Test

Før målgruppetesten, udgives en offentlig test af spillet til de, der gennem spørgeskemaet havde sagt, de ønskede at deltage. Dette giver mulighed for at få fundet og eventuelt udbedret tydelige fejl og mangler. E-mailen der blev sendt ud i forbindelse med den offentlige test, kan ses på bilag 3. Da testen var offentlig, er der også andre end spørgeskemadeltagere, der har haft mulighed for at deltage.

9.1 Offentlig test

Resultater

Ud fra de tilbagemeldinger der blev modtaget i forbindelse med den offentlige test, følger her en liste over de fundne problemer¹:

- Større skyderadius for modstanderne.
- Modstanderne skal ikke flygte så tidligt.
- Modstanderne skal oftere samle våben op.
- Modstanderne skal kunne skyde, når de flygter.
- Sigtekorn foran spilleren (det er svært at ramme med bazookaen).
- Mangler minimap - et lille kort over hele banen, som evt. kunne placeres i et hjørne af HUD'en.
- Forvirrende, at hele banen drejer om spilleren.
- Valg af bane inde i spillet.
- Manglende brugsanvisning.

Tilbagemeldingerne resulterede i, at der inden målgruppetesten blev ændret/rettet følgende:

- Tre sværhedsgrader (Easy, medium, hard) blev tilføjet.
- Tilstande for den kunstig intelligente modstander finpudset.
- Når modstanderen leder efter ammunition, kan denne også tage et våben, hvis det er tættere på.

9.2 Målgruppetest

For at evaluere produktet, er der foretaget en målgruppetest. Formålet med denne test er at se, om de krav og forventninger, som målgruppen stillede, er indfriet.

9.2.1 Sådan blev der testet

Gruppe C214² indvilligede i at teste produktet. C214 ligger i projektets målgruppe, og har ydermere deltaget i spørgeskemaundersøgelsen.

C214 fik besked på at installere og afprøve produktet. Efter at have haft mulighed for at afprøve produktet i 10 minutter fik gruppen lov til at komme med umiddelbare kommentarer, hvorefter der blev stillet en række spørgsmål om produktet. Spørgsmål og resultater af testen er at finde på bilag 4.

¹Disse tilbagemeldinger er at finde på <http://109.dk/jungle/?page=comments>

²Gruppe C214 er fra Det Teknisk Naturvidenskabelige Basisår ved Aalborg Universitet, 2. semester 2006

9.2.2 Diskussion af resultater

De umiddelbare kommentarer til spillet omhandlede udseende og styreregenskaber. Her var der forslag om at tydeliggøre tekst og sigte i spillet. Der var også forslag om at forstørre visse elementer i menuen. Under spillet var der visse elementer af den kunstige intelligens, som testgruppen ønskede ændret. Her er der tale om, at modstanderne opførte sig underligt, når den flygtede, og ønsker om at gøre modstanderen mindre deterministisk. Fx ville nogle af de adspurgte gerne have at modstanderne skød kaniner for "sjov". Der var kommentarer til nogle af de hjemmelavede lyde. Der var idéer til implementation af flere grafiske aspekter, og at styringen skulle foregå med musen. Alt i alt synes de godt om spillet, men de var ikke sikre på, om det stadig ville være sjovt efter et par timer.

Mange af de ovennævnte idéer kræver ikke det store at implementere. Her tales om de grafiske idéer, som farver og størrelser på elementer. Lydene er heller ikke svære at ændre, hvis der var bedre lyde til rådighed. Andre aspekter er sværere at ændre, fx er det problematisk at ændre styringen til musen. Modstanderne er også svære at gøre mindre deterministiske, da det ikke umiddelbart er muligt at tilfældiggøre en større del af modstanderens beslutninger, uden at det går udover den kunstige intelligens. Dette ville også kræve flere tilstande, da der ikke er ret mange i den version gruppen testede.

9.3 Evaluering

Med den offentlige test af spillet var det forventet at finde fejl og mangler i spillet og særligt i forhold til spillets AI. Dette var dog ikke tilfældet. En del af svarene tydede på, at de adspurgte ikke havde forstået formålet med testen.

Det generelle billede, der tegnede sig af den offentlige test var, at spillet manglede et overordnet formål og en historie. Nogle fejl i spillet er blevet rettet fra den offentlige test til målgruppetesten.

I målgruppetesten var formålet veldefineret på forhånd, dette gav et bedre resultat af testen. Det generelle billede på responsen var, at modstandernes tilstande skulle justeres og nye skulle tilføjes, således det ikke var let at forudsige modstanderens næste træk.

Kapitel 10

Konklusion

I dette kapitel samles der op på alle trådene fra de forgående kapitler. Derudover kigges der på, om de opstillede problem i problemformulering er besvaret. Til slut vil der være et perspektiveringsafsnit, hvor der kigges på problemer og mulige ændringer til spillet.

10.1 Konklusion

Gennem problemanalysen blev forskellige aspekter af spil og AI gennemgået. Dette mundede ud i en målgruppeundersøgelse. Målgruppen foretrak bl.a. spilgenren action, så denne blev valgt til det endelige produkt. Yderligere blev målgruppen spurgt, om hvordan de vægtede forskellige emner inden for AI: vejfinding, udnyttelse af færdigheder, taktik og læring. Disse prioriteter var vejledende for, hvad der skulle vægtes højt i det endelige produkt.

Et spil udvikles ved først at lave de vigtigste dele af spillet. Her er der tale om det visuelle, da denne del skal bruges for at kunne teste spillet igennem projektet. Herefter bliver det resterende, som fx AI, bygget på. Det er blevet valgt, at hvert AI aspekt skal udvikles og testes hver for sig, hvorefter de bliver afprøvet sammen. Hvis der er fejl, bliver disse rettet og igen bliver der gennemført en test.

I dette projekt blev AI'en implementeret som et fuzzy logic-system, se afsnit 8.3.1. Ved at bruge et sådant system, viste det sig at være let at ændre på den kunstige intelligente modstander, fx at justere modstandernes aggressivitet.

At få modstandernes handlinger i spillet til at virke ikkedeterministiske viste sig at være et svært problem at løse. Dette er forsøgt løst, men modstandernes træk er stadig forudsigelige. Havde flere tilstande været tilføjet, ville modstanderne måske have været mindre forudsigelige.

Temaet for dette projekt er modellernes virkelighed - netværk og algoritmer. Sammenhængen mellem temaet og produktet er, at der er blevet brugt forskellige algoritmer til at konstruere den AI, der forefindes i spillet. Der er forsøgt at skabe en AI, der skal efterligne en menneskelig karakter i spillet. Derved skabes der en model af virkeligheden, ligesom det overordnede tema antyder.

Igennem projektet blev der fundet ud af, at i en AI kræves en beslutningsmotor, i dette tilfælde er den lavet ved hjælp af fuzzy logic. Denne beslutningsmotor kalder så de givne funktioner, alt efter hvilken situation der skal tages stilling til. Hvis modstanderen i spillet bliver angrebet, bliver der sendt nogle værdier, såsom liv og ammunition, til beslutningsmotoren. Der bliver så taget en beslutning alt efter, hvilke værdier der kommer ind, og fx hvilke grænser beslutningsmotoren har.

For at kunne implementere AI i et specifikt spil, kræver det kendskab til spillets opbygning. Beslutningsmotoren i den kunstige intelligens beslutter, hvilke funktioner der skal tages i brug, og hvornår. Ud fra forskellige inputs fra spillet, giver beslutningsmotoren et output og herved får AI til at virke intelligent. Da skydespil blev valgt som spilgenre, vil et eksempel på input være spillerens position i spillet. Spillerens position udnyttes af modstanderen, som finder frem til, hvor det er bedst at skyde, så den med større sikkerhed kan ramme spilleren.

Når et spil designes, kan der være forskellige samfundsmæssige hensyn der skal tages stilling til, afhængigt af hvilke lande det ønskes udgivet i. Dette produkt ønskedes kun udgivet i Danmark, hvor der ikke er nogen love på området, men kun vejledning. Derfor er der på dette punkt ingen umiddelbar lovmæssig hindring for at udvikle et voldeligt spil.

For at kunne producere et spil, skal der bruges forskellige teknologier. I dette produkt er teknologierne C++, OpenGL, OpenAL og SDL blevet brugt. Dette har fungeret efter hensigten.

10.2 Perspektivering

I dette afsnit vil der blive diskuteret, hvad der kunne gøres bedre eller anderledes i forhold til produktet. Der vil blive diskuteret multiplayer, AI, 3D, salg og fremtiden.

Spillet har fået implementeret AI i form af vejfinding og en begrænset form for fuzzy logic. Spillets AI vil aldrig blive perfekt[20], og kan forbedres. Der er ingen form for læring i spillet, så dette kunne være et emne at tage op, hvis spillet skulle videreudvikles. Hvis læring skal implementeres, skal der foretages ændringer i koden, og der skal findes ud af, hvordan de forskellige oplysninger, der skal læres af, skal gemmes. AI'en skal også have at vide, hvordan denne skal fortolke de forskellige oplysninger, og hvordan disse skal hjælpe med til at foretage en beslutning. Her kunne det evt. være ideelt at kigge på neurale netværk, da disse kan oplæres, til at tage de mest optimale beslutninger i forhold til de givne situationer.

Et andet element, som kunne tilføjes til spillets AI, er taktik. Koden skal ændres for, at AI'en skal kunne foretage taktiske beslutninger, hvor den fx venter på spilleren rundt om et hjørne, eller at den går den anden vej rundt om en mur. Der skal implementeres forskellige funktioner til at tage disse beslutninger, og her kan der evt. bruges fuzzy logic-system eller neurale netværk som beslutningsmotoren.

Grafikken i spillet er kun lavet i 2D, da der er fokuseret på AI. En fremtidig udvidelse kunne være at lave grafikken om til ren 3D. Det vil både kræve ændringer i tegnefunktionerne, og i banerne. I øjeblikket bliver banerne gemt i en tekstfil, hvor hvert felt i banen svarer til et tegn i filen. For at banerne kan laves i 3D er det nødvendigt at dette system ændres. Banefilerne kunne med fordel laves i binært format for at spare pladsen, da de vil indeholde en stor mængde informationer. Udover banerne skal figurerne også ændres til 3D. Dette kræver endnu større ændringer i tegnefunktionerne, da de nu skal tage højde for 3D-modeller frem for et simpelt 2D-billede.

Spillet er kun konstrueret til at være et singleplayer-spil, men en udvidelse kunne være at tilføje multiplayer. Der er flere forskellige muligheder at gøre dette på, og en af dem kunne være, at der skulle sidde to spillere ved samme computer. For at kunne gøre dette, skal tasterne tjekkes for hver spiller. For at begge spillere skal kunne se, hvad de laver, skal skærbilledet laves om til splitscreen*. For at have flere end to spillere i multiplayer, bliver spillet, bl.a. af komfortable grunde, nødt til at spilles over netværk. Her kan der enten være en spiller, som laver en server på vedkommendes egen computer, hvor alle andre spillere så laver en forbindelse til. En anden mulighed kunne være, at der er en dedikeret server, som der ikke kan spilles på, hvor alle spillerne så forbinder til. I begge tilfælde vil spillerne være klienter, og en server vil så køre alle løkker og funktioner som fx dem der styre AI. I den første situation vil den ene spiller dog have både klient og server på sin computer, men i koden vil de stadig være adskilt.

Hvis spillet bliver videreudviklet, vil det evt. kunne sælges til brugere. Der er flere forskellige måder at få solgt spillet på. En måde kunne være, at der blev spurgt forskellige personer på gaden, men dette ville ikke være særlig effektivt. Derimod kunne der gøres reklame for spillet på diverse medier, så brugerne bliver informeret, om hvad spillet kan, hvad spillet handler om, og at spillet kan købes i butikkerne.

Det ville være en god ide at få lavet en speciel version af spillet, der kunne bruges som en demonstration, der frit kunne downloades. I demonstrationen kunne forskellige funktioner være låst for at lave en begrænsning af spillet. Hvis brugerne synes spillet er interessant, kan vedkommende købe en nøgle, som kan låse spillet op, således at alle funktioner bliver tilgængelige.

I fremtiden kunne der bruges tid på at forbedre produktets performance, og her er der flere elementer, der kan ændres. Det ene er i det grafiske, hvor billedfilerne har en større opløsning

end, hvad der bliver tegnet på skærmen, hvilket ikke er effektivt. Langt de fleste billeder kan derfor halveres i størrelse og derved forbedre både brugen af hukommelse og hastigheden, hvormed grafikken bliver tegnet på skærmen. Det andet element er det store *Mother*-array. Hver gang dette array bliver gennemløbet, tjekkes det 65536(2^{16}) gange. Dette er på ingen måde effektivt, da det hver frame kan risikere at blive gennemløbet over 30 gange, hvis der er et passende antal modstandere, kaniner og skud i spillet. *Mother* bliver i en sådan situation tjekket omkring 2 millioner gange. Hvis *Mother*-arrayet i stedet blev konstrueret som en linket liste, vil størrelsen af *Mother* kunne gøres dynamisk, og derved tilpasses det nuværende antal objekter. I en normal situation vil der findes omkring 250 objekter i *Mother*, og hvert element vil derfor kun blive tjekket cirka 7500 gange, hvilket er 99,6% færre tjek! Der er selvfølgelig andre faktorer, der her gør sig gældende, såsom hastigheden på RAM'ene, da objekterne vil ligge spredt i hukommelsen med linkede lister fremfor lige efter hinanden med et array. Dog vil CPU-forbruget blive effektiviseret drastisk, så de resterende faktorer har ikke så stor indvirkning i denne sammenhæng.

Kapitel 11

Litteratur

- [1] Cyberathlete Professional League. Cyberathlete professional league, Marts 2006. URL <http://www.thecpl.com>.
- [2] PEGI. Pan european game information, Marts 2006. URL <http://www.pegi.info>.
- [3] Dansk Medierådet for Børn og Unge. Medierådets website om børn, unge og computerspil, Marts 2006. URL <http://spil.medieraadet.dk>.
- [4] Wikipedia. Turing test — wikipedia, the free encyclopedia, 2006. [Online; accessed 26-May-2006]. URL http://en.wikipedia.org/w/index.php?title=Turing_test&oldid=54797398.
- [5] Bruno Miguel Teixeira De Sousa og Ronald Penton. *Game Programming All in One*. The Premier Press, 2002.
- [6] Martin Persson. *Development of three AI techniques for 2D platform games*. Department of Computer Science at Karlstad University, 2005.
- [7] Mat Buckland. *AI Techniques for Game Programming*. The Premier Press, 2003.
- [8] Jeannette Lawrence. *Neural Networks: Design, Theory and Applications - 5th edition*. California Scientific Software, 1993.
- [9] Alexander Nareyek. Guest Researcher at Carnegie Mellon University. AI in computer games. *ACM Queue*, 1(10), February 2004.
- [10] Alex J. Champandard. *AI Game Development: Synthetic Creatures with Learning and Reactive Behaviors*. New Riders Publishing, November 2003.
- [11] Mikael Togeby Jill Mehlbye Olaf Rieper. *Håndbog i evaluering*. AKF Forlaget, Oktober 2003.
- [12] Craig Larman. *Agile and Iterative Development*. Addison Wesley.
- [13] Barry Boehm og Richard Turner. *Balancing Agility and Discipline - A Guide for the Perplexed*. Addison Wesley, 2004.
- [14] Kenneth H. Rosen. *Discrete Mathematics and Its Applications*. McGraw Hill, 5. udgave, 2002.

-
- [15] Patrick Lester. A* pathfinding for beginners, Maj 2006. URL <http://www.policyalmanac.org/games/aStarTutorial.htm>.
- [16] Bjarne Stroustrup. Bjarne stroustrup's faq. URL http://www.research.att.com/~bs/bs_faq.html.
- [17] Mattias Engdegård, Julian Peterson, Ken Jordan, , Maxim Sobolev, Wesley Poole, Michael Vance, Andreas Umbach og Andreas Hofmeister. Sdl api reference guide, Maj 2006. URL <http://www.libsdl.org/archives/SDLRef.chm>.
- [18] Dave Shreiner, Mason Woo, Jackie Neider og Tom Davis. *OpenGL Programming Guide*. Addison-Wesley, 2. udgave, January 1997.
- [19] Garin Hiebert. *OpenAL Programmer's Guide*. Creative Technology Ltd., Oktober 2005.
- [20] Mette Hjermand McCall. Glem alt om kunstig intelligens. *Nyhedsmagasinet Ingeniøren*, 21:12, Maj 2006.
- [21] Ron Penton. *Data Structures for Game Programmers*. Premier Press, 2003.
- [22] David M. Bourg og Glenn Seeman. *AI for Game Developers*. O'Reilly, July 2004.
- [23] Wikipedia the Free Encyclopedia. Platform game, Marts 2006. URL http://en.wikipedia.org/wiki/Platform_game.

11.1 Spilliste

Chessmaster 10th Edition

<http://chessmaster10.ubi.com/us>.

Command & Conquer Generals: Zero Hour

http://www.ea.com/redesign/games/pccd/ccgenerals_zerohour/home.jsp.

Doom 3

<http://www.doom3.com>.

Dune II

http://en.wikipedia.org/wiki/Dune_II.

Dungeons & Dragons Online

<http://www.ddo.com>.

F.E.A.R.

<http://www.whatisfear.com>.

Flight Simulator 2004

<http://www.microsoft.com/Games/FlightSimulator>.

Grand Theft Auto (GTA) Vice City

<http://www.rockstargames.com/vicecity/>

Grand Turismo 2

[http://en.wikipedia.org/wiki/Gran_Turismo_\(game\)](http://en.wikipedia.org/wiki/Gran_Turismo_(game)).

Neverwinter Nights

<http://nwn.bioware.com>.

Serious Sam

<http://www.serioussam2.com>.

Sim City

<http://simcity.ea.com>.

Super Mario Bros.

http://en.wikipedia.org/wiki/Super_Mario_Bros.

The Elder Scrolls IV: Oblivion

<http://www.elderscrolls.com>.

Wolfenstein 3D

<http://www.3drealms.com/wolf3d>.

11.2 Kildekritik

Der findes megen information i bøger og på internettet, men det er ikke alle kilder, der er lige pålidelige. Derfor er det vigtigt, at der er en vis kritik overfor de kilder og informationer, der findes, specielt på internettet. Det kan være en nødvendighed at sammenligne informationen med andre kilder, for at se, om der er overensstemmelse mellem kilderne, og for at sikre pålideligheden af kilden.

De bærende kilder i denne rapport er dem, der omhandler AI-teorier.

En af de bærende kilder i dette projekt er kilde [15], som er en hjemmeside. Denne kilde bakkes op af kilde [21] på side 750 og kilde [6] på side 77, hvor A* algoritmen gennemgås.

Kilde [6] er hovedkilde for fuzzy logic. Nogle aspekter af fuzzy logic bliver også gennemgået i kilde [22] i kapitel 10.

Kapitel **12**

Bilag

12.1 Ordliste

Real-time	Dette er en spiltype, hvor spilleren og modstanderen kan lave træk samtidigt. Dette er den mest populære type på spilmarkedet.
Turn-Based	I modsætning til real-time, er spillet her baseret på at spilleren og modstanderen skiftevis laver træk. Altså de vil aldrig kunne lave træk samtidigt, hvilket afværger eventuelle samtidighedsproblemer.
RTS	Forkortelse for Real-Time Strategi. Strategispil der benytter real-time spiltypen, der er beskrevet ovenfor.
FPS	Her i en forkortelse for First Person Shooter, førstepersonsskydespil. En spiltype hvor spillet foregår gennem spillerens øjne - altid i 3D. FPS kan også betyde Frames Per Second, hvilket betyder antal billeder der vises i sekundet, men dette bliver ikke brugt i denne rapport af hensyn til forståelsen.
Platformsspil	En spiltype som oftest er kendt fra gamle gamle konsolspil. Det karakteristiske ved denne spiltype er at der klatres og hoppes op og ned mellem platforme. Dette var princippet i mange gamle 2D-spil, men mange af koncepterne er nu flyttet over i 3D.[23]
Waypoints	En engelsk betegnelse for navigations punkter. Betegnelse bruges ofte i forbindelse med vejfinding og AI
Multiplayer	En spiltype hvor modstanderne er andre spiller, der deltager i spillet fx over netværk. I denne spiltype er der sjældent brug for nogen form for AI.
Singleplayer	Denne spiltype spilles kun af en enkelt spiller. Hvis der er modstandere, er de styret af AI, altså computeren.
Gameplay	En engelsk betegnelse for et spils underholdningsværdi.
Hitbox	Indenfor computerspil bruges denne engelske betegnelse for inddelingen af hvor på kroppen en karakter kan kollideres med fx skud. Hver hitbox kan herefter fx have forskellige sårbarheder. I dette tilfælde har hovedet oftest den største sårbarhed, og armene den mindste.
NPC	En forkortelse for None Player Character. En figur i spillet der styres af computeren.
Refaktorering	I softwareudviklingsammenhænge er ordet refaktorering en betegnelse for, at ændre kildekoden, således denne opnår et højere abstraktionsniveau, og gør det derved muligt at genbruge koden.[12]
Highscore	Betegnelse for en pointtabel med de spillere, der har opnået flest point i et givent spil.
Powerup	En pakke som kan indeholde en bonus som hjælper spilleren. Dette kunne fx være i form af mere liv, ammunition eller et nyt våben.

HUD	Forkortelse for Head Up Display. HUD er en metode hvormed at der vises informationer om fx liv og skud til spilleren. HUD'en er ikke en del af spilverden men derimod fastsat på bestemte steder på skærmen.
API	Forkortelse for Application Programming Interface. API-kald gør det muligt for brugeren at udføre system-specifikke handlinger, så som at få returneret antallet af sekunder computeren har kørt.
Frame	Et billede tegnet på skærmen.
Framerate	Betegnelse for antal billeder/frames der bliver tegnet på skærmen over et specifikt rum tid (ofte et sekund (frames per second)).
Array	En tabel i koden. Kan indeholder et bestemt antal værdier eller objekter.
Pointer	En pointer er en adresse til et bestemt sted i hukommelsen hvor der ligger en variabel eller et objekt.
Spawn	Engelsk ord for fødsel. I denne rapport, og i spilindustrien, er det et begreb for når et objekt kommer til live i spil verdenen.
Spiltscreen	Betyder at splitte skærmen op i to, hvor hver spiller har sin egen halvdel med sin spiller på.
Debug	Står for fejlfinding eller fejlretning.

12.2 Bilag 1 - Spørgeskemaundersøgelse

Gruppe C213 - Spørgeskema om spil og AI

Spil og AI

Side 1 af 5

1. [*] Hvilket køn er du?

Mand
 Kvinde

2. [*] Hvilken aldersgruppe tilhører du?

Under 18
 18-24
 25-30
 Over 30

3. [*] Hvor mange timer om ugen spiller du computerspil/konsolspil?

0 timer
 0-2 timer
 2-4 timer
 4-6 timer
 6-8 timer
 8-10 timer
 10-15 timer
 15+ timer

Selvom du svarede at du spiller 0 timer om ugen, ønsker vi stadig at du svare på de resterende spørgsmål.

4. Hvilke spilgenrer foretrækker du?
Hvis du ikke foretrækker nogen genre, så bare fortsæt til næste side.

Action (Half-Life, Quake, Battlefield, GTA, ...)
 Rollespil (Diable, World of Warcraft, ...)
 Brætspil (Skak, Backgammon, ...)
 Strategi (Command & Conquer, Age Of Empires, ...)
 Simulation (Need For Speed, Fligt Simulator, Sim City, ...)
 Andet

Gruppe C213 - Spørgeskema om spil og AI

Spil og AI

Side 2 af 5

**Følgende spørgsmål omhandler udelukkende singleplayer spil:
Singleplayer betyder at modstanderen er baseret på kunstig intelligens - dvs. at den styres udelukkende af computeren selv.**

1 svarer til laveste prioritet, og 5 svarer til højste prioritet.

	1	2	3	4	5	Ved ikke
5. [*] Hvor højt prioriterer du grafik i et spil?	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
6. [*] Hvor højt prioriterer du kunstig intelligens i et spil?	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
7. [*] Hvor højt prioriterer du at et spil overholder realisme mht. fysiske love og menneskelige begrænsninger - fx tyngdekraft, hastigheder og dødelighed?	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

Gruppe C213 - Spørgeskema om spil og AI

Spil og AI

Side 3 af 5

**Følgende spørgsmål omhandler specifikt kunstig intelligens i singleplayer spil:
1 svarer til laveste prioritet, og 5 svarer til højste prioritet.**

	1	2	3	4	5	Ved ikke
8. [*] Hvor højt prioriterer du modstanderens evne til at finde vej i spilverdenen - altså at komme udenom forhindringer, og ikke sidde fast i blindgyder?	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
9. [*] Hvor højt prioriterer du modstanderens udnyttelse af dens færdigheder (f.eks. sigte eller våbenvalg)?	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
10. [*] Hvor højt prioriterer du modstanderens taktiske egenskaber (f.eks. samarbejde og forudsigelser)?	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
11. [*] Hvor højt prioriterer du modstanderens egenskaber til at lære af sine fejl (f.eks. ikke at løbe i samme baghold hver gang)?	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>
12. [*] Hvor højt prioriterer du at modstanderen har en kort betænkningstid - altså at modstanderen ikke bruger for lang tid på hvert træk?	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>	<input type="radio"/>

Gruppe C213 - Spørgeskema om spil og AI

Spil og AI

Side 4 af 5

13. Nævne nogle spil (serier) du mener har en god kunstig intelligens:

- ANDET
- The Sims
- Battlefield
- Half-Life
- Worms
- F.E.A.R.
- GTA (Grand Theft Auto)
- Soldat
- Unreal Tournament
- Quake
- Doom
- Commandos
- Chessmaster
- Need For Speed
- Atomic Bomberman
- Command & Conquer

Gruppe C213 - Spørgeskema om spil og AI

Spil og AI

Side 5 af 5

14. Skriv navnet på det, eller de, spil du mener har en god kunstig intelligens, men ikke kunne vælges på forrige side:

12.3 Bilag 2 - Resultater fra spørgeskemaundersøgelse

Hvilket køn er du?

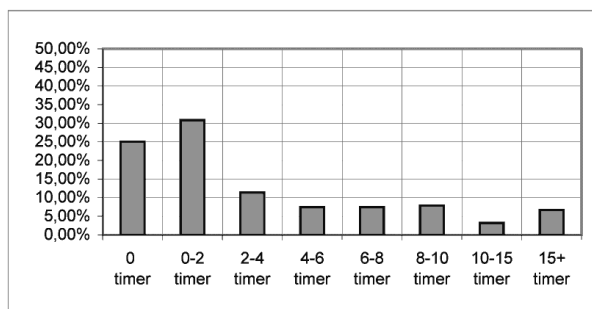
Mand	294	68,21%
Kvinde	137	31,69%
Antal svar	431	

Hvilken aldersgruppe tilhører du?

Under 18	2	0,46%
18-24	383	88,86%
25-30	27	6,26%
Over 30	19	4,41%
Antal svar	431	

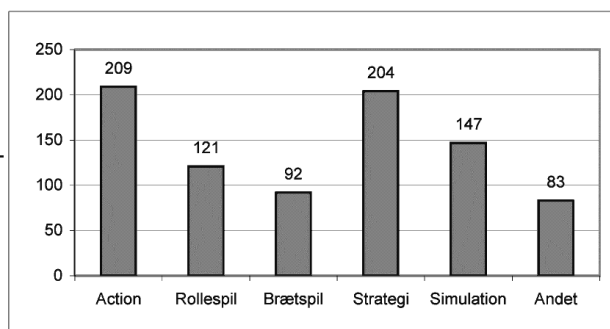
Hvor mange timer om ugen spiller du computerspil/konsolspil?

0 timer	108	25,06%
0-2 timer	133	30,86%
2-4 timer	49	11,37%
4-6 timer	32	7,42%
6-8 timer	32	7,42%
8-10 timer	34	7,89%
10-15 timer	14	3,25%
15+ timer	29	6,73%
Antal svar	431	



Hvilke spilgenrer foretrækker du?

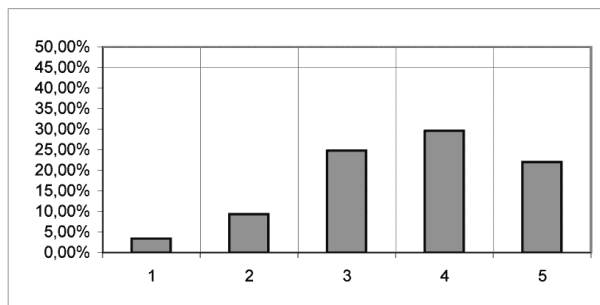
Action	209	24,42%
Rollespil	121	14,14%
Brætspil	92	10,57%
Strategi	204	23,83%
Simulation	147	17,17%
Andet	83	9,70%
Antal svar	856	



Hvor højt prioriterer du grafik i et spil?

1	19	3,45%
2	41	9,43%
3	108	24,83%
4	129	29,66%
5	96	22,07%
Ved ikke	42	9,66%
Antal svar	435	

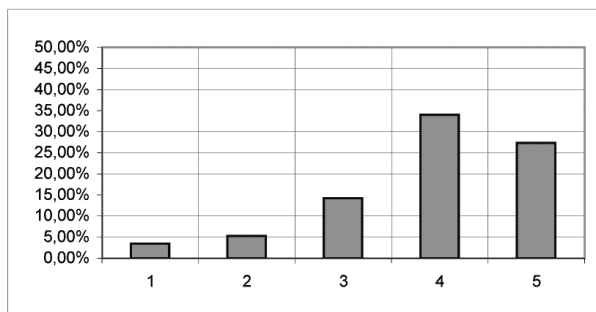
Gennemsnit: 3,6158



Hvor højt prioriterer du kunstig intelligens i et spil?

1	15	3,45%
2	23	5,29%
3	62	14,25%
4	148	34,02%
5	119	27,36%
Ved ikke	68	15,63%
Antal svar	435	

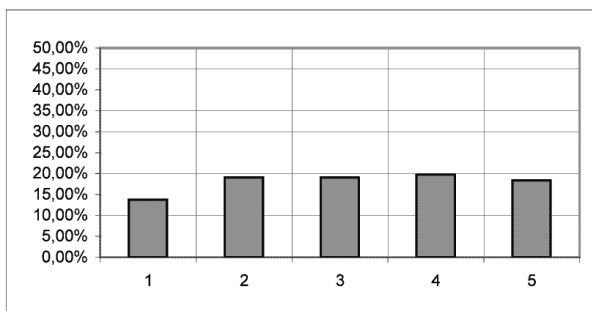
Gennemsnit: 3,9074



Hvor højt prioriterer du at et spil overholder realisme?

1	60	13,79%
2	83	19,08%
3	83	19,08%
4	86	19,77%
5	80	18,39%
Ved ikke	43	9,89%
Antal svar	435	

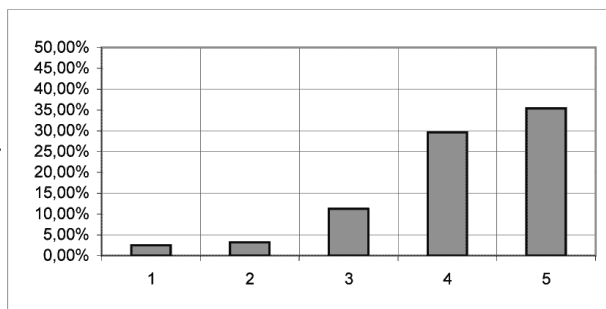
Gennemsnit: 3,1097



Hvor højt prioriterer du modstanderens evne til at finde vej i spilverdenen?

1	11	2,53%
2	14	3,22%
3	49	11,26%
4	129	29,66%
5	154	35,40%
Ved ikke	78	17,93%
Antal svar	435	

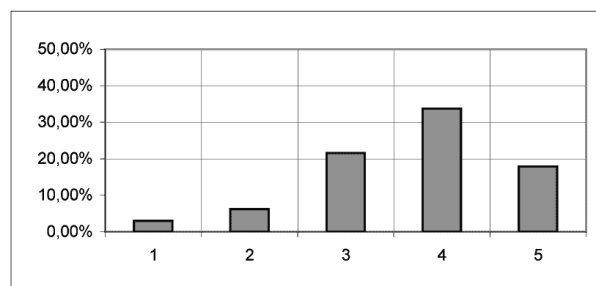
Gennemsnit: 4,1232



Hvor højt prioriterer du modstanderens udnyttelse af dens færdigheder?

1	13	2,99%
2	27	6,21%
3	94	21,61%
4	147	33,79%
5	78	17,93%
Ved ikke	76	17,47%
Antal svar	435	

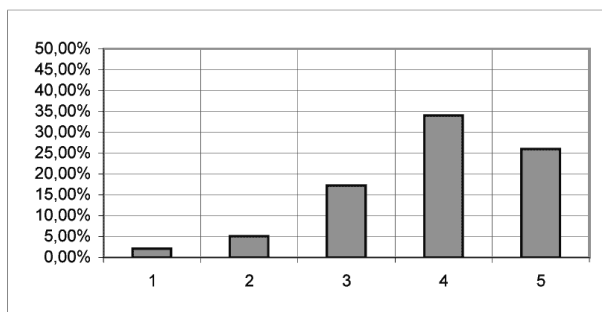
Gennemsnit: 3,6964



Hvor højt prioriterer du modstanderens taktiske egenskaber?

1	9	2,07%
2	22	5,06%
3	75	17,24%
4	148	34,02%
5	113	25,98%
Ved ikke	68	15,63%
<hr/>		
Antal svar	435	

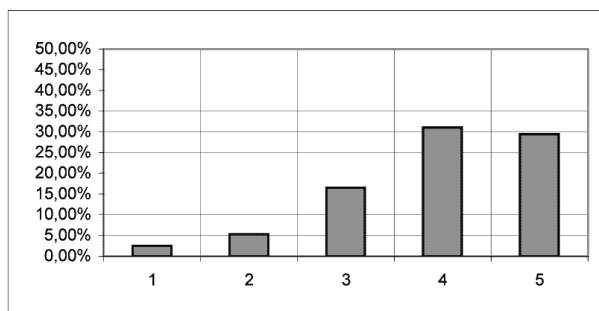
Gennemsnit: 3,9101



Hvor højt prioriterer du modstanderens egenskaber til at lære af sine fejl?

1	11	2,53%
2	23	5,29%
3	72	16,55%
4	135	31,03%
5	128	29,43%
Ved ikke	66	15,17%
<hr/>		
Antal svar	435	

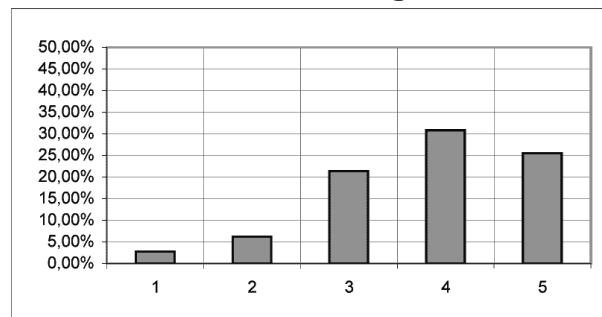
Gennemsnit: 3,9377



Hvor højt prioriterer du at modstanderen har en kort betænkningstid?

1	12	2,76%
2	27	6,21%
3	93	21,38%
4	134	30,80%
5	111	25,52%
Ved ikke	58	13,33%
<hr/>		
Antal svar	435	

Gennemsnit: 3,809



Nævne nogle spil (serier) du mener har en god kunstig intelligens

The Sims	83	12,05%	Chessmaster	37	5,37%
Half-Life	79	11,47%	Worms	34	4,93%
ANDET	72	10,45%	Quake	30	4,35%
GTA (Grand Theft Auto)	61	8,85%	F.E.A.R.	27	3,92%
Battlefield	59	8,56%	Doom	20	2,90%
Need For Speed	54	7,84%	Commandos	19	2,76%
Command & Conquer	52	7,55%	Atomic Bomberman	19	2,76%
Unreal Tournament	37	5,37%	Soldat	6	0,87%
<hr/>					
Total Answers	689				

12.4 Bilag 3 - Beta-email

Dette er emailen der blev sendt ud til vores betatestere. Da betatesten var offentlig var der mulighed for at andre, end modtagerne af denne email, deltog i testen.

Hej beta-testere :)

Nu er vi endelig klar med en beta-version af vores spil J.U.N.G.L.E.

For at teste spillet skal du gå ind på beta-test-siden: <http://www.109.dk/jungle> Her kan du downloade de nødvendige filer samt skrive en kommentar til spillet. Vi ved godt du nok sidder og lægger sidste hånd på dit eget projekt og hvis du derfor ikke ønsker at blive forstyret er det helt i orden og forståeligt. Hvis vi får noget konstruktivt ud af vores beta-test vil det indgå i vores rapport.

Kendte fejl (disse vil blive rettet så hurtigt som muligt):

- Når man trykker på "Start game" anden gang crasher spillet.
- Highscore giver nogle specielle tal og fungerer ikke optimalt.

Til sidst vil vi gøre opmærksom på at spillet udskriver informationer til en fil kaldet "stdout.txt". Hvis der sker noget meget mærkeligt (som fx et crash) vil det være rart hvis i gad sende denne fil til os, således at vi har mulighed for at se hvordan fejlen opstod.

Spillet kan startes med en Runner.exe. I denne kan der vælges at spille med fuldskræm, "debug"-mode eller med et andet map end standardet.

Der følger også en map-editor med som du kan bruge til at lave dine egne maps. Denne kan dog indeholde mange fejl og fungerer derfor ikke helt optimalt :) .

På forhånd tak

Gruppe C213 (Software)

- - - - -

Du modtager denne email fordi din email-adresse er blevet tilføjet til vores beta-mailing-liste (efter deltagelse i vores spørgeskema-undersøgelse).

Hvis dette ikke er tilfældet og/eller du ikke ønsker at få flere mails fra os så gå ind på http://www.109.dk/survey/thx_email.php?remove=true og skriv din email. Den vil så blive fjernet fra listen.

12.5 Bilag 4 - Spørgsmål og resultater fra målgruppetest

Testgruppens umiddelbare kommentarer:

- Startknappen i runner-programmet skal sidde i hjørnet.
- Instruktionerne i startmenuen skal være større eller tydeligere.
- Vis, hvilke våben man har samlet op.
- Spilleren bør gå hurtigere.
- Modstanderne bør flygte mod en powerup, så de hurtigere kan komme i kamp igen.
- Spillet kan virke forvirrende, når det er miljøet, der drejer i stedet for spilleren.
- Tilføj evt. mulighed for at bevæge sig sidelæns og bruge musen til at dreje spilleren med.

Spørgsmål til testgruppen og de tilhørende svar:

- *Er HUD'en, og dens funktion, tydelig?*
Den grå tekst må gerne ændres til sort, så kontrasten bliver større.
Vis navn på det våben man har, og det man samler op.
- *Er tile'enes typer tydelige?*
Modstandernes spawn-punkter er ikke så sigende.
Der kunne evt. laves en gravsten, hvor en modstander dør.
- *Er brugen af tasterne intuitiv?*
Ja, det er let at gå til.
Mere øvede spillere vil gerne have mulighed for at bruge mus.
- *Foregår spillerens bevægelse i spillet tilfredsstillende i forhold til taste-styringen?*
Ja.
- *Er det tydeligt, hvad de forskellige powerups repræsenterer?*
Speed- og ammo-powerups er ikke så sigende.
- *Er powerup'enes egenskaber tilfredsstillende?*
Ja.
- *Er AI'ens opførsel realistisk, når den ikke har set spilleren?*
Modstanderen burde samle de våben op, som den ser.
Modstanderen burde skyde kaniner for "sjov".
- *Er AI'ens vejfinding tilfredsstillende?*
Ja.
- *Er AI'ens reaktions-valg tilfredsstillende?*
Når modstanderen flygter, virker det underligt.
- *Er AI'en forudsigelig?*
Ja.
- *Er sværhedsgraderne tilpasse?*
Ja.

- *Hvordan er gameplayet?*
Det er sjovt, men ikke så langtidsholdbart.
- *Hvordan er lydene?*
Der burde ikke blive brugt hjemmelavede lyde til bazookaen.
- *Er spillet tilpas voldeligt?*
Ja, men der kunne evt. laves bloddryp, når de er blevet skudt.
- *Kører spillet tilfredsstillende på testerens computer?*
Ja.
- *Hvad er den overordnede vurdering af spillet?*
Sjovt, men forvirrende.

