Tweedejaars Project AI
Quakevolution

Universiteit van Amsterdam

Bram Stoeller, 0426857
Tjerk Kostelijk, 0418889
Folkert Huizinga, 0418862
Rik Tobi, 0448710

Juni 2006

Contents

1 Inleiding

Het doel van dit project is om te demonstreren dat het toepassen van door de natuur geinspireerde programmeertechnieken op de kunstmatige intelligentie van computerspellen de spelervaring voor mensen sterk kan verbeteren. Aangezien computerspellen een grote bron van invloed zullen gaan vormen voor de ontwikkeling van de AI leek het spel *Quake III* (volledige titel: *Quake III Arena*) ons een gepaste keuze als platform om ons project op te baseren. De broncode van *Quake III* is namelijk door de makers (*id Software*) voor het publieke domein beschikbaar gesteld. Een tweede partij, Icculus [1], heeft de broncode opgeschoond en geschikt gemaakt voor moderne compilers. Met deze versie als uitgangspunt hebben we een testomgeving opgezet binnen het spel waarin *bots* (computer-tegenstanders) worden geplaatst die zichzelf aanpassen door middel van gesimuleerde natuurlijke selectie. Vervolgens passen we vanuit de testomgeving een genetisch algoritme toe dat op de karakteristieken van deze *bots* opereert. Dit algoritme selecteert aan de hand van de uitkomst van een fitheids-functie twee *bots* (ouders) uit de populatie. De eigenschappen van deze twee ouders worden gecombineerd om zo nieuwe *bots* (kinderen) te genereren. Dit proces wordt een aantal generaties herhaald om de best aangepaste *bot* te genereren en te laten spelen tegen een standaard-*bot*.

[1] www.icculus.org/quake3

2 Projectomschrijving

2.1 Quake III Arena

Quake III is een zogenaamde 3D *first-person shooter*. De speler bestuurt een *avatar* door een virtuele 3D-omgeving (*level*) gedurende een aantal rondes. Deze speler heeft als doel zoveel mogelijk vijanden te *fraggen* (neer te schieten) gebruikmakende van verschillende door het level verspreidde wapens. Het fraggen levert de speler punten op afhankelijk van de gekozen spelmode. Vijanden worden ofwel door andere menselijke tegenstanders bestuurd of door een kunstmatige intelligentie; in het laatste geval wordt deze ook wel een *bot* genoemd. Wanneer de *health* van de speler zelf nul bereikt 'sterft' hij maar kan korte tijd later weer met volledige health *respawnen* (opnieuw aan de ronde deelnemen). Hierbij verliest de speler alle wapens die hij voor zijn dood had verzameld maar behoudt hij zijn score. Het spel eindigt zodra een speler of team een bepaalde maximale score heeft gehaald of een vooraf ingestelde tijdslimiet bereikt wordt.

2.2 Testomgeving

Omdat *Quake III* standaard geen mogelijkheden biedt om *bots* te evolueren moest er eerst een testomgeving geconstrueerd worden. Hiervoor was het onder andere nodig om het spel automatisch te laten herstarten, wat we bewerkstelligd hebben door een geforceerde game-loop te implementeren. Deze game-loop bestaat uit individuele rondes (generaties) die elk een tijdslimiet hebben van bijvoorbeeld tien minuten. Verder moest het ook mogelijk zijn om vanuit het spel *bot*-files te kunnen aanmaken en inlezen, om zo nieuw gegenereerde *bots* dynamisch aan een ronde toe te voegen. De statistieken van de *bots* worden opgeslagen in een log-bestand.
Bij de testomgeving hoort uiteraard ook het genetische algoritme dat het feitelijke evolutieproces aanstuurt. De implementatie hiervan wordt beschreven in hoofdstuk vier.

3 Bots

Een *bot* is een kunstmatige speler die het gedrag van mensen in een spel zo goed mogelijk probeert na te bootsen. Voor *Quake III* houdt dit in dat een *bot* op een menselijke manier en volgens dezelfde spelregels een zo hoog mogelijke score moet proberen te halen. Omdat een *bot* beschikt over informatie over de staat van de wereld waar mensen geen toegang toe hebben en daarom in principe vals kan spelen moeten er wel door het spel beperkingen worden opgelegd om dit zo eerlijk mogelijk te bereiken.

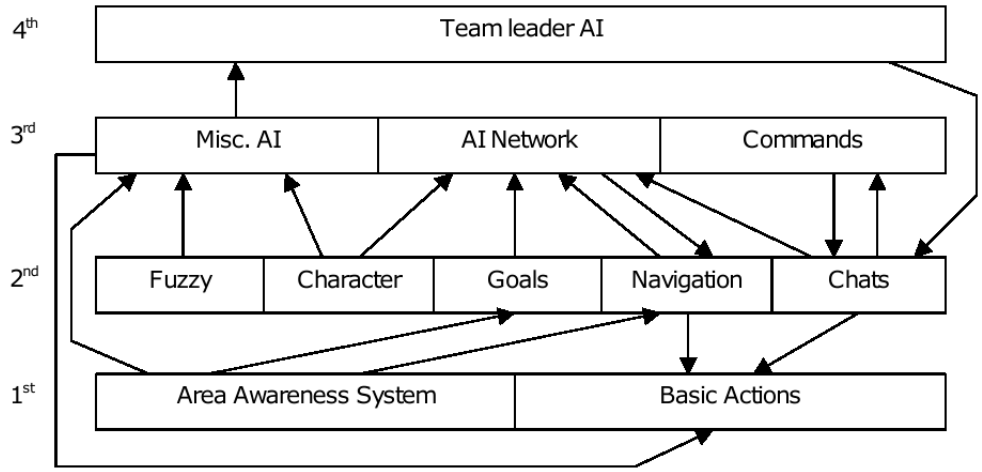
3.1 Structuur

Een *bot* in *Quake III* bestaat uit een set van vier bestanden: *bot_c.c*, *bot_w.c*, *bot_i.c*, en *bot_t.c*. Elk van deze bestanden heeft een eigen taak. *bot_c.c* bevat de karakteristieken van de *bot*, zoals bv. *jumper* en *croucher*. Deze beïnvloeden het gedrag en daardoor ook de kwaliteit van de *bot* tijdens het spel; *croucher* geeft bv. aan hoezeer de *bot* de neiging heeft te bukken en *jumper* hoezeer de *bot* de neiging heeft te springen. *bot_w.c* en *bot_i.c* bevatten een aantal *fuzzy logic weights* die beinvloeden hoe graag de *bot* in een situatie een bepaald wapen of object wil hanteren danwel verkrijgen. *bot_t.c* bevat voorgedefinieerde *chat*-text die de *bot* 'uitspreekt' en regels die controleren onder welke omstandigheden wat gezegd wordt.

3.2 Architectuur

Bots in *Quake III* functioneren volgens een gelaagde architectuur die de figuur hieronder schematisch is weergegeven. Het bewustzijn van de beslissingen die een *bot* maakt terwijl het spel wordt gespeeld wordt groter in de hogere lagen.

3.2.1 *Afbeelding 'Bot Architectuur'*



De eerste laag is de input- en output-laag voor de *bot*. Deze verschaft de *bot* alle informatie over de huidige status van de wereld via het *Area Awareness System*. De tweede laag levert de intelligentie die bij menselijke spelers vaak onbewust aanwezig is, en gebruikt *fuzzy logic* om bepaalde goals te selecteren. De derde laag is een mix van productieregels (if-then-else) en een AI-netwerk met speciale nodes voor verschillende situaties en 'mentale' states waarin de *bot* zich kan bevinden. Dit netwerk lijkt heel erg op een finite state machine en neemt alle redenatieprocessen van de *bot* voor zijn rekening. De vierde en hoogste laag tenslotte vormt het brein van de teamleider in het geval dat de *bot* in een team speelt. De AI-component van *Quake III* handelt de interactie van en naar deze lagen van een *bot* af; ons project richt zich op de functionaliteit van de tweede laag.

3.3 Karakteristieken

Om enigszins controle te kunnen uitoefenen op het evolutieproces hebben we de evolutie beperkt tot de *characteristics* van een *bot*. We hebben besloten de karakteristieken uit *bot_c.c* te isoleren en de gewichten in de overige bestanden constant te houden om het effect van mutaties hierop te verduidelijken. Van de karakteristieken evolueren we bovendien alleen een subset omdat niet alle karakteristieken voor het genetisch algoritme interessant zijn. Daarnaast bleek het praktisch niet tijdig haalbaar om ook de overige *fuzzy logic*-gewichten in beschouwing te nemen.

4 Genetisch Algoritme

Een genetisch algoritme is een door de natuur geinspireerde zoektechniek uit de Kunstmatige Intelligentie die vooral toegepast wordt om oplossingen te vinden voor multidimensionale optimalisatie- en zoekproblemen. Hierbij worden evolutionaire processen als selectie, overerving en mutatie

aangewend om de beste eigenschappen van kandidaatsoplossingen te combineren. Voor ons project zijn de karakteristieken (genotype) van de *bots* zelf deze oplossingen.

4.1 Fitheids-functie

Aan de hand van het resultaat dat een *bot* behaald heeft wordt zijn zogenaamde *fitness-value* (fitheids-waarde) bepaald. Het aantal *frags* en *deaths* dat een *bot* verzameld heeft vormen de parameters van de *fitness-function* (fitheids-functie). We hebben geëxpirimenteerd met verscheidene fitheids-functies, welke behandeld worden in het hoofdstuk *Methode*.

4.2 Selectie

Tijdens de selectie wordt bepaald welke twee *bots* gekozen worden voor de evolutie. Hierbij volgen we het concept van 'survival of the fittest'. Net zoals bij de evolutietheorie van Darwin is het idee dat de best aangepaste individuen overleven en hun genen doorgeven aan de volgende generatie.

4.2.1 Elite-selectiemodel

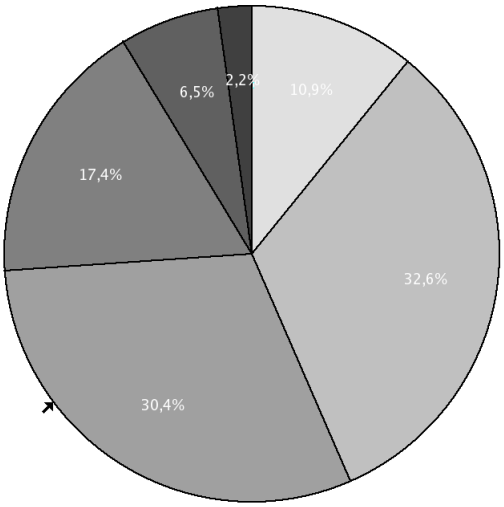
In dit selectiemodel worden altijd n *bots* met de hoogste fitheids-waarde geselecteerd. Dit in tegenstelling tot andere modellen van selectie waarbij ook *bots* met een lagere fitheids-waarde een kans hebben om geselecteerd te worden. Met behulp van dit model zal de evolutie het snelst voortschreiden. Echter staat het niet zo dicht bij de natuur als het roulette-selectiemodel.

4.2.2 Roulette-selectiemodel

In dit selectiemodel heeft elke *bot* een kans om te overleven en zo bij te dragen aan de evolutie. Dit model maakt het mogelijk dat er andere *bots* worden gekozen dan de beste. Het wordt gebruikt om de processen, zoals deze in de natuur plaatsvinden, nauwkeuriger te simuleren.

Er wordt een virtuele schijf in 'taartpunten' verdeeld waarbij iedere *bot* een bepaald percentage van de schijf krijgt toegewezen (zie afbeelding 4.2.3). De percentages corresponderen met de fitheids-waardes van de *bots*. Evenals bij het spel roulette wordt een 'balletje' op een draaiend 'wiel' geworpen. Het 'balletje' wijst het deel van de schijf aan dat overeenkomt met de *bot* die geselecteerd wordt.

4.2.3 *Afbeelding 'Het roulette diagram'*



4.2.4 *Afbeelding 'Het roulette-lijn diagram'*

In onze implementatie worden de fitheids-waardes van de *bots* aangrenzend op een lijn geprojecteerd (zie afbeelding 4.2.4). De lengte van ieder segment komt overeen met de fitheids-waarde van de corresponderende *bot*. Er wordt een random getal gegenereerd tussen nul en de totale lengte van de segmenten. Het getal valt altijd in het bereik van een segment van een *bot*, welke dan geselecteerd wordt.

4.3 Genen

Een *bot* bezit een aantal karakteristieken. Het veranderen van deze karakteristieken (in de evolutietheorie ook wel genen genoemd) heeft dus indirect invloed op de fitheids-waarde, omdat een deel van het gedrag van de *bot* en daarmee zijn spelprestaties door de karakteristieken geregeld wordt. Alle karakteristieken samen worden ook wel het genotype genoemd.

4.3.1 Overerving

Na het selecteren van twee ouders aan de hand van één van de bovenstaande selectiemodellen wordt de overervingsmethode toegepast. M.b.v. deze overerving worden een aantal kinderen geconstrueerd. Een kind bestaat uit een combinatie van de genen van zijn ouders. Per gen wordt bekeken van welke ouder het wordt overgeërfd, waarbij de kansen voor beide ouders gelijk zijn.

4.3.2 Mutaties

Per gen is er een kleine kans dat een mutatie plaatsvindt. Hierbij wordt een gen overschreven met een random waarde. Door deze mutatie kan een gen ontstaan met een betere waarde dan het gen dat een kind anders van zijn ouders zou overerven.

5 Methode

5.1 Initialisatie

We starten het evolutieproces met zes random *bots*. Deze *bots* worden gegenereerd door iedere karakteristiek een random waarde te geven. Na veel experimenten blijkt dat een speelduur van tien minuten het beste resultaat geeft. Ook blijkt uit onderzoek dat een minimum van vijftien rondes nodig is voor zichtbare evolutie. Om deze reden versnellen wij het spel met een factor vijftien. Dit bleek op de testopstelling de maximaal haalbare factor. Ook zou het mogelijk zijn de grafische module uit te schakelen. Dit hebben we echter niet doorgevoerd omdat dit veel extra tijd zou hebben gekost. Tevens zou het weinig voordeel opleveren omdat het grootste deel van de berekeningen door de grafische kaart wordt uitgevoerd en niet door de algemene processor.
Het versnellen heeft verder geen invloed op het gedrag van de *bots* omdat alle spelprocessen evenredig versneld worden. Het versnellen van de tijd maakt het mogelijk om vele rondes te spelen, waarbij we hebben gekozen voor een totaal van 160 rondes. Na 120 rondes stabiliseren de resultaten, de marge van 40 extra rondes zorgt voor een continue stabiliteid van de data.

5.2 Iteratief proces

Als de rondetijd verstreken is, worden de resultaten weggeschreven naar een bestand in CSV (comma-separated values) formaat. In dit bestand worden per regel de karakteristieken van een *bot* en zijn behaalde resultaten opgeslagen. Vervolgens worden middels een selectiemethode twee ouders geselecteerd. Het genetisch algoritme contrueert met deze ouders drie nieuwe kinderen. Deze kinderen worden als *bot_c.c* bestanden weggeschreven en vormen samen met de ouders de nieuwe generatie. Aan deze generatie voegen we een random *bot* toe. Vervolgens wordt er met deze 'pool' een nieuwe ronde gestart. Als de tijd verstreken is worden er weer twee winnaars geselecteerd en begint het proces opnieuw totdat de iteratielimiet is bereikt.
In vele genetische algoritmes stopt de evolutie wanneer een van te voren vastgesteld fitheidsniveau behaald wordt. Dit niveau is in *Quake III* echter lastig te bepalen. Onze stopconditie is daarom het verstrijken van een vooraf geconfigureerde rondelimiet.
De winnaars van de laatste ronde zijn statistisch gezien het meest geëvolueerd. Dit testen we door deze *bots* tegen standaard-*bots* met dezelfde *weapon*- en *item*-weights te laten spelen.

5.3 Weergave resultaten

De resultaten die opgeslagen zijn in het .csv bestand worden ingelezen met behulp van de scripttaal PHP. Op de PHP pagina's[2] kunnen we grafieken genereren van alle karakteristieken en statistieken inspecteren van iedere *bot* die in een spel heeft meegespeeld. Het is ook mogelijk de deviatie te plotten van iedere ouder-*bot*. De deviatie wordt berekend door het absolute verschil van de karakteristieken van de ouder-*bots* bij elkaar op te tellen. De verwachting is dat het verschil tussen de ouders kleiner wordt naarmate het evolutieproces vordert.

5.4 Selectiemodellen en fitheids-functies

We hebben geëxperimenteerd met zowel verschillende selectiemethodes als verschillende fitheidsfuncties. Ook hebben we alle mogelijke combinaties hiervan onder de loep genomen. Als selectiemechanismen zijn het elite-selectiemodel en het roulette-selectiemodel gebruikt. Verder hebben we vele fitheids-functies getest, waarvan we de twee interessantste zullen analyseren.

[2] http://student.science.uva.nl/~fhuizing/graph/ en http://student.science.uva.nl/~rtobi/shared/TJPAI/stats/

5.4.1 Fitheids-functies

Het aantal *frags* en *deaths* dat door een *bot* behaald is zijn de parameters van de fitheids-functie. Een *frag* wordt behaald als een andere speler wordt neergeschoten, een *death* als de speler doodgaat. Er kunnen ook negatieve *frags* behaald worden. Dit gebeurt als een speler door zijn eigen fout doodgaat. Het vallen in een krater en het vlak voor een muur afschieten van een raket komen daarbij het meest voor. Deze zelfmoordacties worden van de huidige *frag*-score afgetrokken, waardoor het resultaat negatief uit kan slaan. Hier is in de formules voor de fitheids-functie rekening mee gehouden.

Formule 1:

$$F(frags, deaths) = 2 \times frags - deaths$$

Hier hebben we een wegingsfactor ingevoerd: het aantal *frags* telt twee keer zo zwaar als het aantal *deaths*. Door deze factor te varieren wordt meer respectievelijk minder waarde gehecht aan karakteristieken als *aggression* en *self-preservation*, maar de schaal blijft lineair.

Formule 2:

$$F(frags) = \frac{frags}{|frags|} frags^2$$

Deze formule bestaat uit twee delen. Door het aantal *frags* te kwadrateren wordt de spreiding van de fitheids-waarde vergroot. De verdeling is hierdoor niet linear zodat een *bot* met een hoog aantal *frags* een veel hogere fitheids-waarde krijgt. Het eerste deel van de formule vangt bovendien het probleem van negatieve *frags* af. Door het aantal *frags* te delen door het absolute aantal zal hier altijd 1 of -1 uit komen. Deze scalar vermenigvuldigen we met het kwadraat van het aantal *frags*, zodat de fitheids-waarde daadwerkelijk negatief zal worden bij een negatieve *frag*-score.

6 Resultaten

6.1 Waarnemingen en Analyse

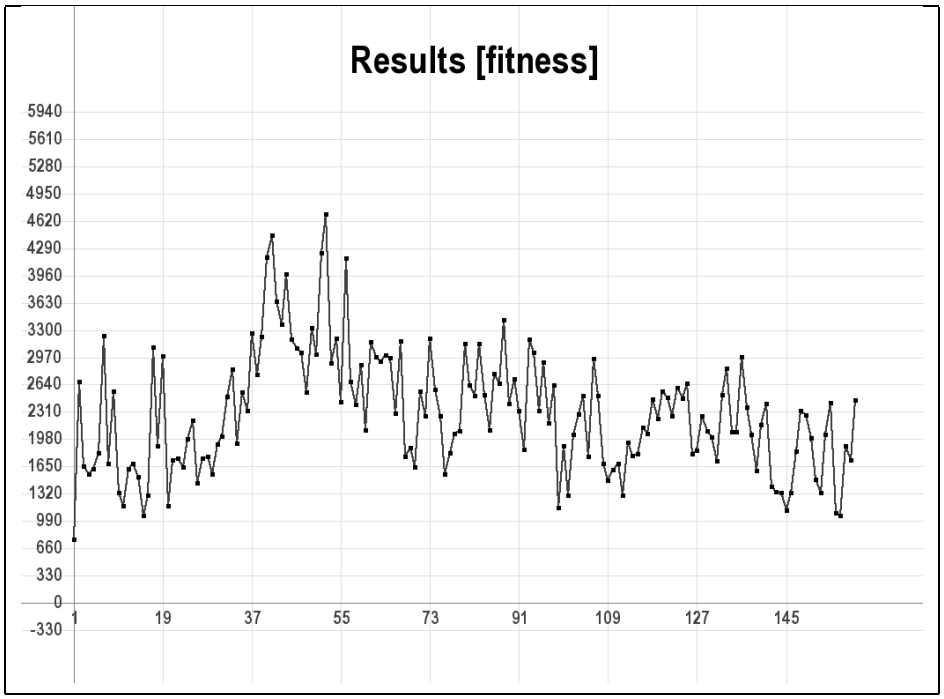
We hebben als eerste het roulette-selectiemodel i.c.m. formule 1 getest. Hieruit bleek dat er weinig evolutie was waar te nemen. De reden hiervoor is dat er te vaak *bots* met relatief lage fitheids-waardes werden geselecteerd, waardoor het resultaat fluctueerde. Om dit te verhelpen hebben we een nieuwe formule toegepast op deze selectie methode (zie grafiek 6.2.1). Ook dit resultaat was verre van gewenst. Om deze reden hebben we de elite-selectiemethode getest (zie grafiek 6.2.2). Hier kwam een grafiek uit waar duidelijk evolutie was waar te nemen.

Hieruit bleek dat de elite-selectiemethode de beste methode was. Deze methode hebben we dan ook aangehouden en toegepast op beide formules (zie grafiek 6.2.3 en grafiek 6.2.4). Er is bij beide grafieken een groei in evolutie te zien. Op de grafiek van formule 2 is dit het best zichtbaar omdat we hier een fitheids-functie gebruiken die voor een opgerekt resultaat zorgt.

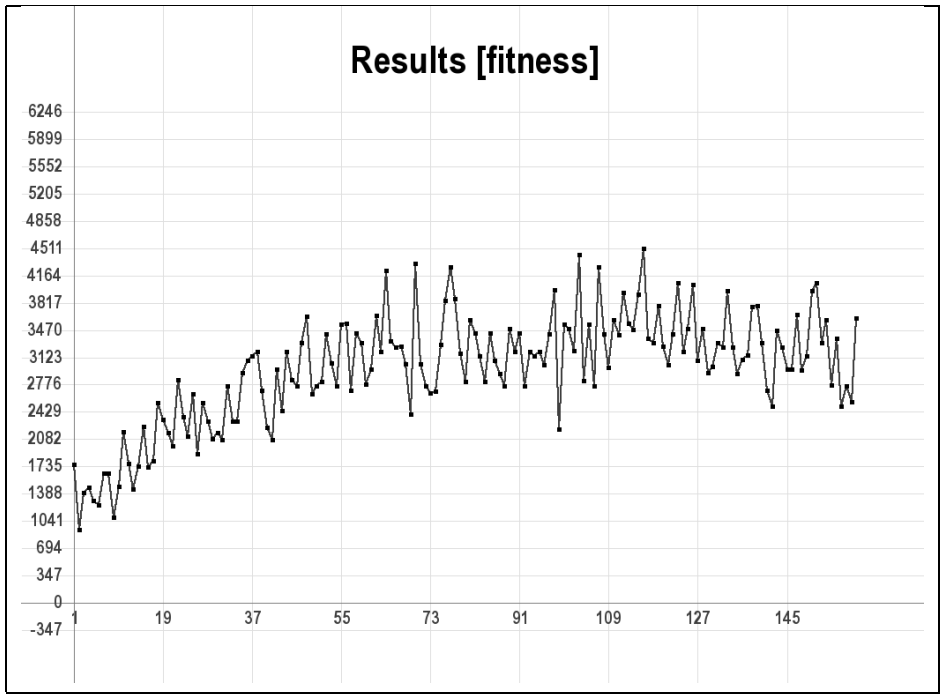
De grafiek van de deviatie geeft aan hoeveel de ouders van elkaar verschillen. Bij het elite-selectiemodel (zie grafiek 6.2.6) zien we dat de deviatie exponentieel snel afneemt. Bij het roulette-selectiemodel (zie grafiek 6.2.5) zien we een soortgelijk resultaat, echter springt de deviatie steeds enkele malen omhoog om vervolgens weer exponentieel af te nemen. Deze plotselinge

pieken worden door de aard van het model zelf veroorzaakt. Zoals eerder gezegd zorgt het roulette-model ervoor dat er soms een *bot* wordt verkozen die niet één van de besten was. De karakteristieken van deze *bot* wijken dan sterk af van de andere, veel betere, ouder, waardoor de deviatie erg groot is.

6.1.1 *Grafiek 'Gemiddelde Fitness - roulette-selectiemodel'*



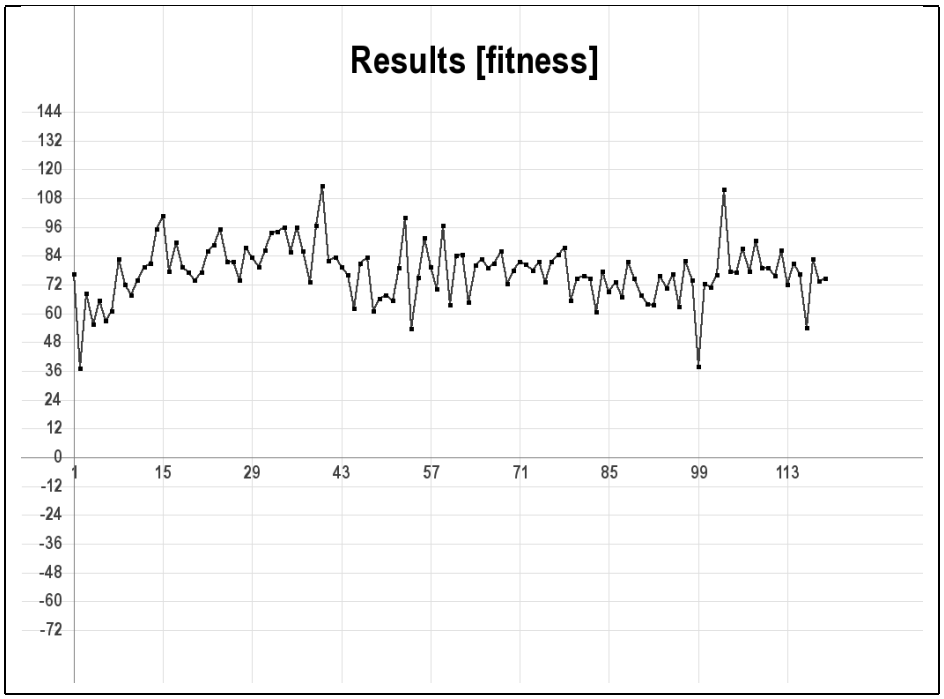
6.1.2 *Grafiek 'Gemiddelde Fitness - elite-selectiemodel'*



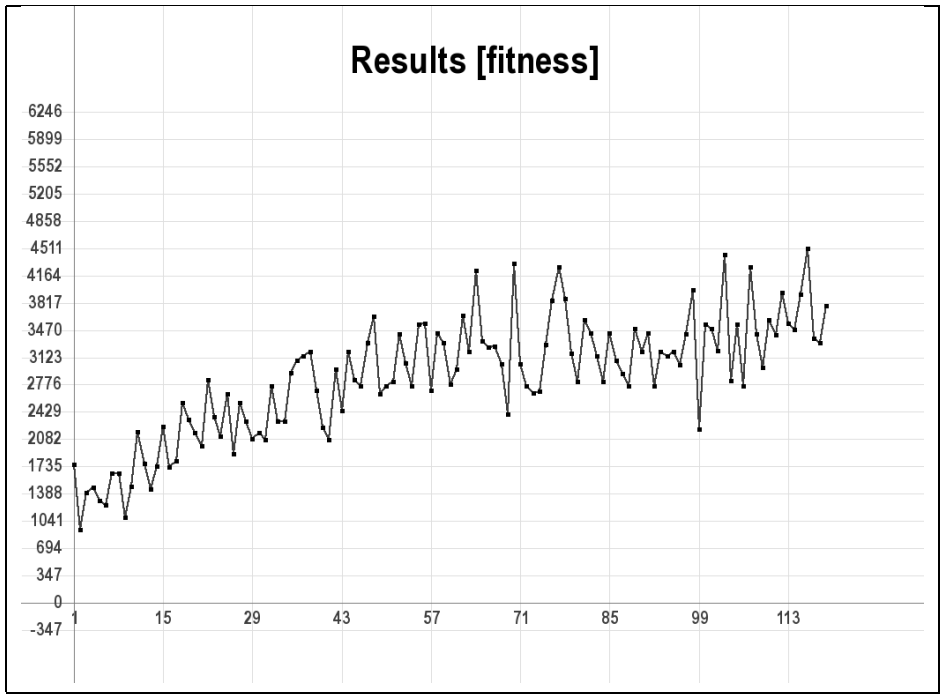
De gemiddelde fitness-waarde van de twee ouders uit iedere ronde, na 160 rondes van 10 minuten en fitnessfunctie:

$$F(frags) = \frac{frags}{|frags|} frags^2$$

6.1.3 *Grafiek 'Gemiddelde Fitness - Formule 1'*

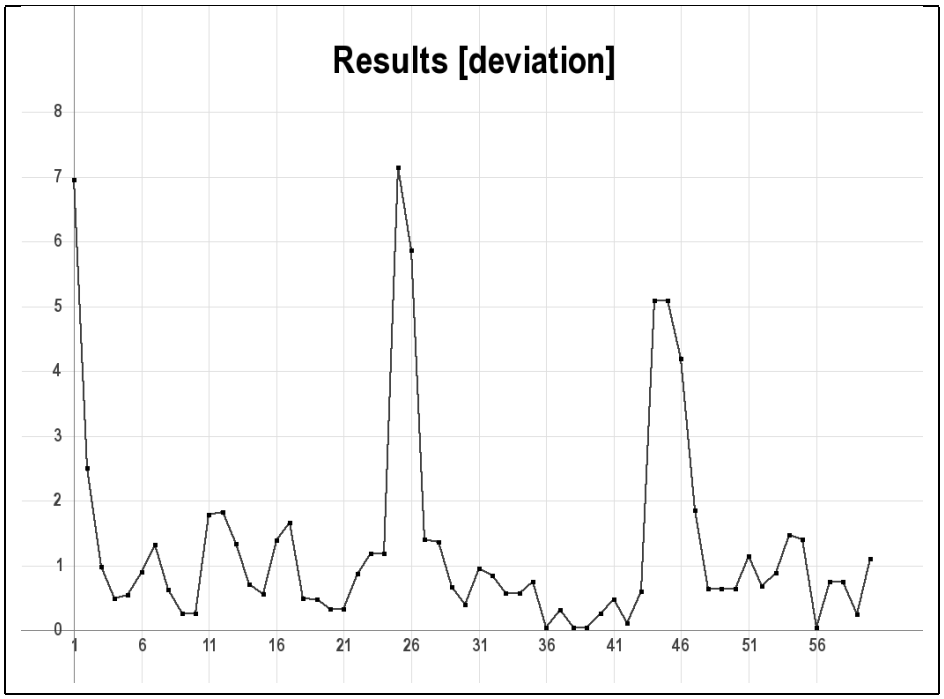


6.1.4 *Grafiek 'Gemiddelde Fitness - Formule 2'*

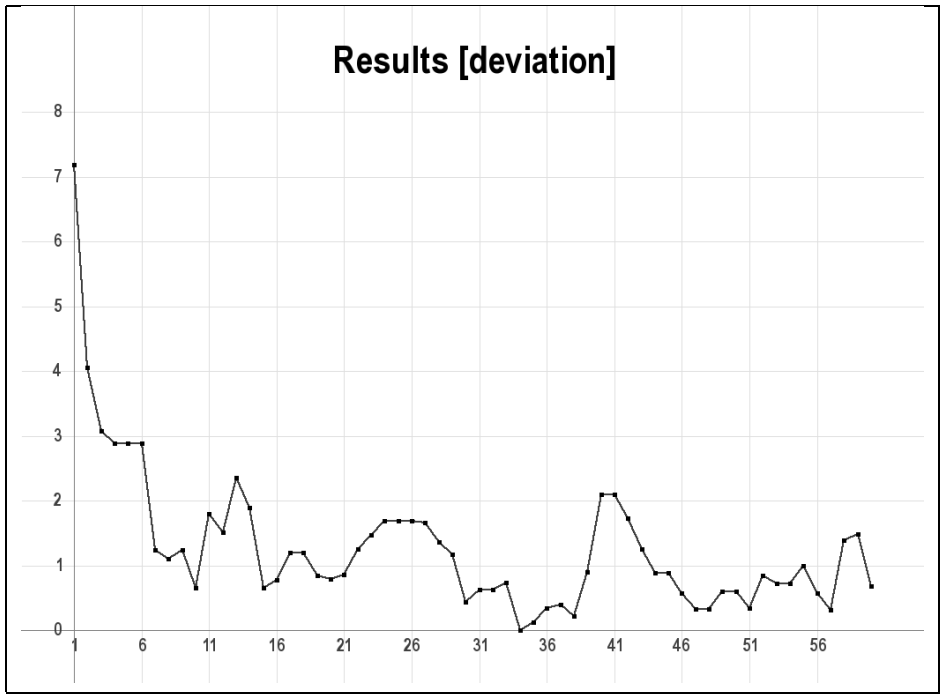


NB. Voor bovenstaande resultaten is het elite-selectiemodel gebruikt.
$F_1(frags, deaths) = 2 \times frags - deaths \qquad F_2(frags) = \frac{frags}{|frags|} frags^2$

6.1.5 *Grafiek 'Deviatie - roulette-selectiemodel'*



6.1.6 *Grafiek 'Deviatie - elite-selectiemodel'*



6.2 Controle

Het uiteindelijke resultaat is een *bot* die in vergelijking met de eerste zes random *bots* een hoge fitheids-waarde heeft. Dit verschil hebben we in de praktijk getest door de geëvolueerde *bot* tegen een standaard-*bot* uit ronde 1 te laten spelen. De geëvolueerde *bot* versloeg zijn overeenkomende standaard-*bot* met een gemiddelde score van 20-1, gemeten over tien rondes.

7 Conclusie

Wat we wilden aantonen is dat het loont om genetische programmeermethodes toe te passen op de kunstmatige intelligentie van computerspellen, om zo tegenstanders te creëren die zichzelf dynamisch aan hun spelomgeving aanpassen. In de context van *Quake III* leidt het toepassen van een genetisch algoritme op de karakteristieken van de reeds in het spel aanwezige *bots* ertoe dat deze steeds betere prestaties gaan leveren. Een van onze geëvolueerde *bots* versloeg zijn overeenkomende standaard-*bot* met een gemiddelde score van 20-1, gemeten over tien rondes.

We hebben voor dit algoritme verschillende selectiemethoden gehanteerd. De roulette-selectiemethode gaf daarbij minder goede resultaten dan de elite-selectiemethode. Er werden te vaak *bots* met relatief lage fitheids-waardes geselecteerd, waardoor de resultaten nooit stabiliseerden. De elite-methode was effectiever: na zo'n honderd generaties bleven de fitheids-waardes rond een maximum zweven.

8 Toekomstig werk

Doordat we het genetisch algoritme direct toepassen op karakteristieken zoals *AIM_SKILL*, welke allemaal naar een duidelijk maximum convergeren, is de uitkomst van het evolutieproces vrij voorspelbaar. Het zou interessanter zijn om bepaalde karakteristieken te combineren tot 'chromosomen' en die combinaties als geheel te evolueren. Op deze manier wordt het algemene gedrag (fenotype) van een *bot* veel complexer. Wanneer een *bot* bijvoorbeeld een chromosoom heeft met de genen *crouching* en *aim_skill* die dezelfde waarden moeten aannemen krijgt men of een *bot* die veel bukt en goed richt (sniper-*bot*) of een *bot* die snel beweegt maar juist minder zorgvuldig richt.

Naast het interne karakter zou het fenotype ook nog gevisualiseerd kunnen worden in termen van het uiterlijk (model en textures) van een *bot*. Een dikke *bot* zou bijvoorbeeld heel sterk maar langzaam kunnen zijn en een slanke *bot* weer wendbaar maar veel kwetsbaarder. Dergelijke hints zouden de spelervaring voor de speler extra kunnen versterken.

A Bronvermelding

1. **Stuart Russell & Peter Norvig**
   Artificial Intelligence: A Modern Approach

2. **Wikipedia**
   www.wikipedia.org
   http://en.wikipedia.org/wiki/Genetic_algorithm

3. **Gene Expression Programming**
   www.gene-expression-programming.com
   www.gene-expression-programming.com/GepBook/Chapter7/Section7.htm

4. **Genetic and Evolutionary Algorithm Toolbox**
   www.geatbx.com
   www.geatbx.com/docu/algindex-02.html

5. **Karl Sims**
   'Evolving Virtual Creatures'

6. **Jan-Paul van Waveren**
   'The Quake III Arena Bot'

7. **Pieter Spronck**
   'Adaptive Game AI'