

Uppdelning av ett artificiellt neuralt nätverk

Kandidatarbete utfört av:

Björn Eklund

Spelprogrammering årskurs 2009

Blekinge Tekniska Högskolan

bjorneklund86@gmail.com

Handledare:

Stefan Johansson

Blekinge Tekniska Högskolan

stefan.johansson@bth.se

ABSTRAKT

Artificiella neurala nätverk (ANN) har många användningsområden inom datavetenskap. Några av dessa är mönsterigenkänning, robotik, processkontroll, optimering och spel [1].

Detta examensarbete kommer att handla om hur en alternativ lösning på den traditionella arkitekturen av hur ett neuralnät kan se ut. Jag kommer att undersöka om man kan ta ett stort och komplext neuralnät och bryta ned detta till mindre neuralnät utan att förlora kvaliteten på botarna i en spelmiljö kallad Open Nero. Detta för att försöka minska beräkningshastigheten av neuralnäten och förhoppningsvis även göra så botarna lär sig ett bra beteende snabbare.

Mitt examensarbete kommer att visa att min lösning av arkitekturen för ett neuralt nätverk inte fungerar speciellt bra då botarna inte lärde sig tillräckligt fort. En fördel med min arkitektur är dock att den är något snabbare än originalets i exekveringshastighet.

Innehållsförteckning

| | |
|--|-----------|
| 1. Introduktion | 4 |
| 1.1 Målgrupp | 4 |
| 1.2 Artificiella neurala nätverk | 4 |
| 1.2.1 Uppbyggnad..... | 4 |
| 1.2.2 Träningmetoder | 5 |
| 1.2.3 Aktiveringsfunktioner..... | 5 |
| 1.2.4 NEAT..... | 6 |
| 1.3 Genetiska Algoritmer | 6 |
| 1.3.1 Överkorsning | 7 |
| 1.3.2 Mutation..... | 8 |
| 1.3.3 Urvalsmetod..... | 8 |
| 1.4 Open Nero | 8 |
| 1.5 Forskningsfråga..... | 9 |
| 1.6 Avgränsningar | 9 |
| 1.7 Valideringsmetod | 9 |
| 2. Utförande | 9 |
| 3. Experiment | 10 |
| 4. Resultat | 11 |
| 5. Diskussion | 13 |
| 6. Slutsats | 14 |
| 6.1 Framtida arbete | 14 |
| Referenser | 15 |
| Appendix A | 16 |

1. Introduktion

Neurala nätverk och genetiska algoritmer är två väldigt intressanta ämnen då de kan finna och lösa problem som du själv inte hade tänkt från början [7]. Då dessa neuralnät kan bli väldigt stora och komplexa så tyckte min handledare och jag att det vore intressant att se om man kan dela upp ett vanligt neuralnät till ett flertal mindre utan att förlora kvalitet. Mer om detta under punkt 2.

1.1 Målgrupp

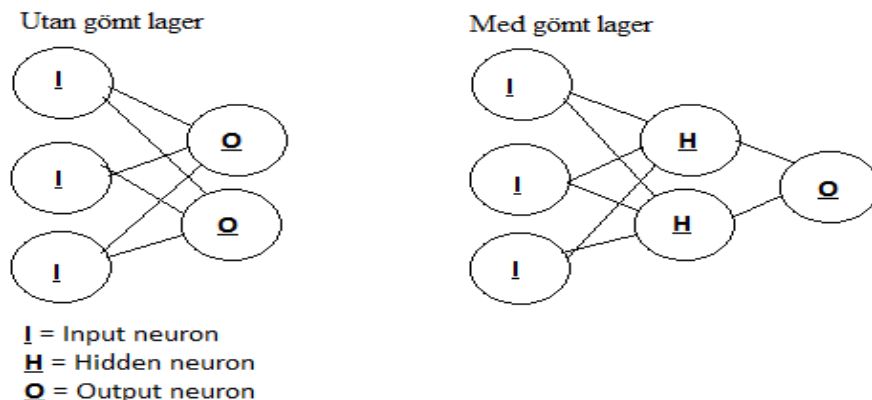
Då detta arbete kommer att handla om att bryta ned ett artificiellt neuralnät som används i en spelvärld som kallas Open Nero så är detta arbete främst till för personer som är intresserade av artificiell intelligens inom spel. Men då neuralnät och genetiska algoritmer även har många andra användningsområden så lämpas detta arbete även till personer med intresse för dessa och deras olika tillämningsmöjligheter.

1.2 Artificiella neurala nätverk

Ett artificiellt neuralnät är en informationsbehandlande struktur som har tagit mycket inspiration om hur en biologisk hjärna behandlar information [1][2]. Det fungerar på så vis att det tar in signaler för att sedan bearbeta dessa och skicka vidare dess resultat till nästkommande del av nätverket.

1.2.1 Uppbyggnad

Ett artificiellt neuralt nätverk är liksom den mänskliga hjärnan uppbyggd utav ett flertal neuroner som kommunicerar med varandra. Dessa neuroner kan vara ihopkopplade på flera olika sätt men det vanligaste är genom ett input lager, ett gömt lager samt ett output lager där varje lager skickar vidare sin output till nästkommande lager, detta även kallat ett feed-forward nätverk. Inputlagret är det första lagret i det neurala nätverket och dess uppgift är att ta hand om den indata som matas in i neuralnätet. Om dessa används i spel kan det vara olika observationer från spelvärlden. Det gömda lagret vilket är ett valbart och går att ha många av har enbart sin roll som neuron, vilket kommer beskrivas snart. Output lagret är det sista lagret och dess största roll är att ge oss det slutgiltiga resultatet. Bilden nedan visar oss två enkla neuralnäts uppbyggnad.



En artificiell neuron är uppbyggd på detta sätt. Se bild 2. Den består utav en serie inputlänkar som har en vikt kopplad till sig, vilket i vårt fall är ett enkelt flyttal. Denna vikt är det ofta man ändrar på vid träning av ANN. Denna vikt kan vara både negativ och positiv. Varje inputsignal multipliceras med denna vikt och summeras sedan med de andra inputsignalernas resultat vilket ger oss aktiveringssumman. Om denna summa är större än ett tröskelvärde så genereras en output. Denna output beror på vilken aktiveringsmetod som används. Mer om dessa i (1.2.3).

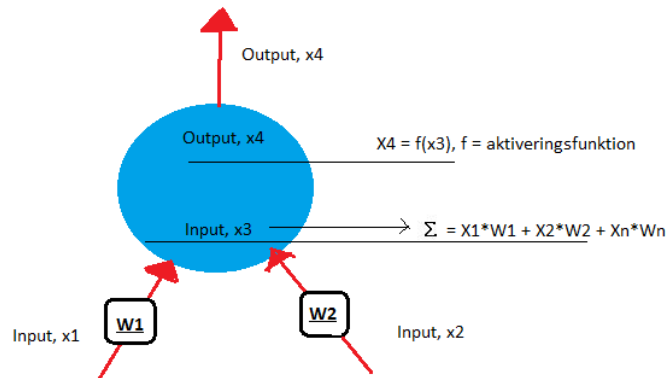


Bild 2

1.2.2 Träningmetoder

Liksom den biologiska hjärnan måste man träna för att bli bra på något och inom ANN finns det två huvudspår om hur man tränar sitt NN. Dessa är handled träning samt ej handled träning. Inom handled träning så kan man även här göra på två olika sätt. Den första metoden som kallas för bakåtpropagering är att du själv matar in par av input och väntad output-data till det neurala nätverket för att sedan rätta till vikterna med hjälp av felmarginalen som kom som output. Denna träning fortskrider tills ett man får ett acceptabelt resultat. Den andra metoden inom handled träning är återkopplat lärande. I denna metod så lämnas inte någon väntad output till det NN utan du tränar det genom att belöna/straffa ett visst beteende[2].

Inom ej handled träning så kan man använda sig av genetiska algoritmer (GA) och låta dessa ordna vikterna för ens ANN. Mer om dessa i punkt 1.3.

1.2.3 Aktiveringsfunktioner

Om ni kommer ihåg från bild 2 så hade vi aktiveringsfunktionen f och dess givna input x_3 . Aktiveringssumman från funktionen f ger oss outputen x_4 . Det finns flera olika sorters aktiveringsfunktioner som kan användas till ett ANN varav dessa är några av dem.

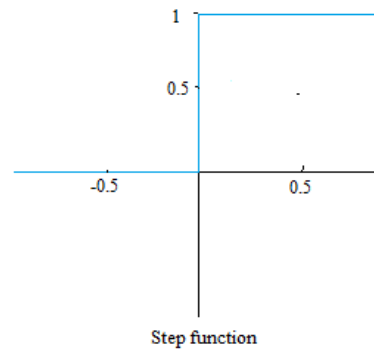
Den linjära aktiveringsfunktionen är den enklaste av de olika aktiveringsfunktionerna men det ger ofta inte heller några direkt intressanta resultat då den returnerar funktionens inputdata.

$$\text{Output } x_4 = F(x_3) = x_3$$

Den andra aktiveringsfunktionen man kan använda sig av är steg-funktionen vilken kan beskrivas som en allt eller ingen funktion då den antingen skickar ut 1 eller 0 från funktionen beroende på om inputvärdet är större än ett tröskelvärde.

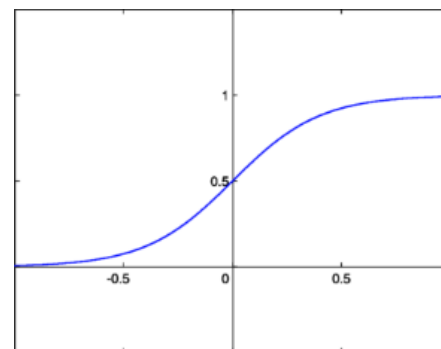
$$\text{Output } x_4 = F(x_3) = \begin{cases} 1 & \text{om } x_3 \geq Y \\ 0 & \text{om } x_3 < Y \end{cases}$$

$Y = \text{Tröskelvärde}$



Den sista vi ska nämna är den Sigmoida funktionen vilken bäst beskriver den biologiska neuronens input-output relation..

$$\text{Output } x_4 = F(x_3) = \frac{1}{1 + e^{-x}}$$



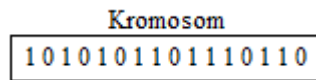
1.2.4 NEAT

Neuro Evolution of Augmenting Topologies är en neural nätverksmetod skapad utav Kenneth O. Stanley och Risto Miikkulainen på Universitetet av Texas i Austin. Tanken med NEAT är att man ska hålla det neurala nätverket så simpelt som möjligt och enbart göra det mer komplext vid behov. Det sätt på vilket detta sker inom NEAT är att använda sig av genetiska algoritmer då man vid behov kan införa nya neuroner och länkar vid en ny generation[8][12]. NEAT finns i flera olika sorters implementationslösningar varav en av dessa är rtNEAT vilket står för real-time NEAT. Denna metod har använts inom ett spel kallat NERO (Neuro Evolving Robotic Operatives) där det används i realtid för att träna ett antal virtuella robotar inför ett krig.

1.3 Genetiska algoritmer

Genetiska algoritmer (GA) är inspirerade från Darwins teori om hur evolutionen fungerar inom biologin. Genom att skapa nya populationer med nya kromosomer hoppas man på att utveckla fram en lösning på ett givet problem. Varje kromosom kan därför ses som en lösning för det givna problemet. En kromosom kan representeras utav ett par binära tal där varje bit motsvarar en gen. Då en kromosom ses som en lösning för ett problem så kan en gen ses som

en del lösning för samma problem. För att välja ut föräldrar till en ny avkomma brukar man använda sig av något som kallas fitnessvärde. Med fitnessvärde menar man hur bra en viss lösning är på att lösa det givna problemet, ju bättre fitness man har ju större chans har man i urvalsmetoden(1.3.3). För varje ny generation som skapas så finns det en chans att det ska ske en överkorsning(1.3.1) eller en mutation(1.3.2) för den nya kromosomen. Om den nya genförändringen är positiv så finns det en chans att denna fortsätter att existera och kommer då även kanske att föras vidare till nästkommande generation [9] [11].



Simpel pseudokod utav en genetisk algoritm [6]

```

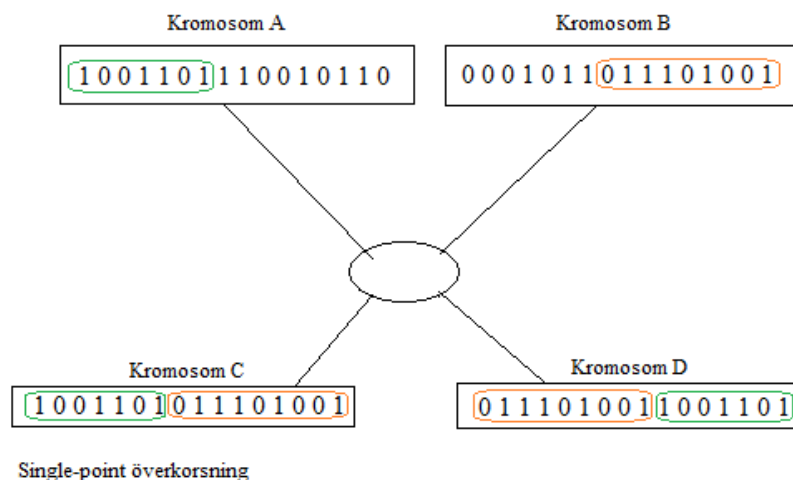
population = array[ 1...n] av individer
Init(population);
Evaluate(population);

While( not terminate )
{
population' = Recombine( population );
population'' = Mutate( population' );
Evaluate( population'' );
population = Select( population'' U population )
}

```

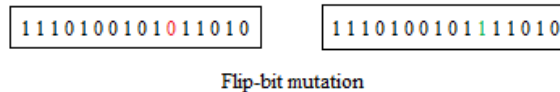
1.3.1 Överkorsning

Med överkorsning inom GA så menar man att man använder sig utav två kromosomer för att sedan skapa två nya genom att korsa dessa. Det finns flera olika sorter av överkorsningsmetoder. Några av dessa är enpunkts överkorsning vilket betyder att man genom slump väljer ut en punkt på föräldrarnas kromosomsträngar för att sedan skapa nya kromosomer med denna punkt som referens. Till den ena kromosomen tar du första halvan från förälder ett och den andra halvan av den andra föräldern. Till den andra kromosomen gör du tvärt om. Sedan finns det flerpunkts överkorsning vilket fungerar på samma sätt som med enpunkts överkorsningen, men med skillnaden att man väljer ut två punkter och byter plats på de gener som finns inom dessa punkter [11].



1.3.2 Mutation

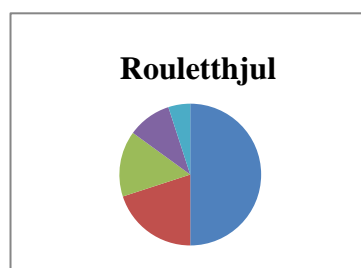
Genom att införa mutationer när man skapar nya kromosomer så ökar man chansen att finna nya lösningar på problemet man försöker lösa då det kan vara svårt att finna alla lösningar genom enbart överkorsning. En mutation kan vara att man använder bit-omkastnings mutation vilket betyder att man inverterar en gen [11].



I NEAT finns även lite special-mutationer då det bygger på att börja enkelt och vid behov öka komplexiteten. Då händer det ibland att det muterar in en ny neuron i nätet eller så kan det läggas in en ny länk mellan vissa neuroner.

1.3.3 Urvalsmetod

Med urvalsmetod menar man vilken metod som ska användas vid val av föräldrar till den nya avkomman. Några av dessa metoder är bland annat rouletthjulsmetoden vilken går ut på att man skapar ett "rouletthjul" med alla potentiella föräldrars fitness som då används som vinstplattser. Ju bättre fitness en förälder har ju bättre odds har denna att bli utvald. En annan metod som kan användas är Turneringsmetoden som går ut på att man väljer ut två individer från populationen, sedan väljs ett slumpat värde ut mellan 0 och 1. Om det slumpade värdet är mindre än ett tröskelvärde som du har bestämt så väljs den individen med bäst fitness utav de två annars så väljs den andra ut till att vara förälder. Sedan returneras de båda individerna in till populationen igen och har möjlighet att bli utvalda igen [9][11]. Den mest enkla metoden man kan använda sig utav är slumpvalsmetoden, vilken går ut på att man slumpmässigt väljer ut en kromosom som man ska använda.



En bild på hur ett rouletthjul kan se ut med 5 individer.

1.4 Open Nero

Open nero är ett open-source projekt baserat på spelet NERO. Open Nero är skapat utav studenter från Neurala nätverks forskargrupp och Institutet för teknik på Texas universitet i Austin. Detta projekt är främst till för att forskning och undervisning då man på enkelt sätt kan skapa eget innehåll till spelet genom scripting men även genom att ändra i källfilerna. För att läsa mer och för att testa dessa program besök deras hemsidor [4][5].

1.5 Forskningsfråga

Får man bättre, samma eller sämre resultat utav att använda sig utav flera mindre neuralnät mot att använda sig utav ett stort.

1.6 Avgränsning

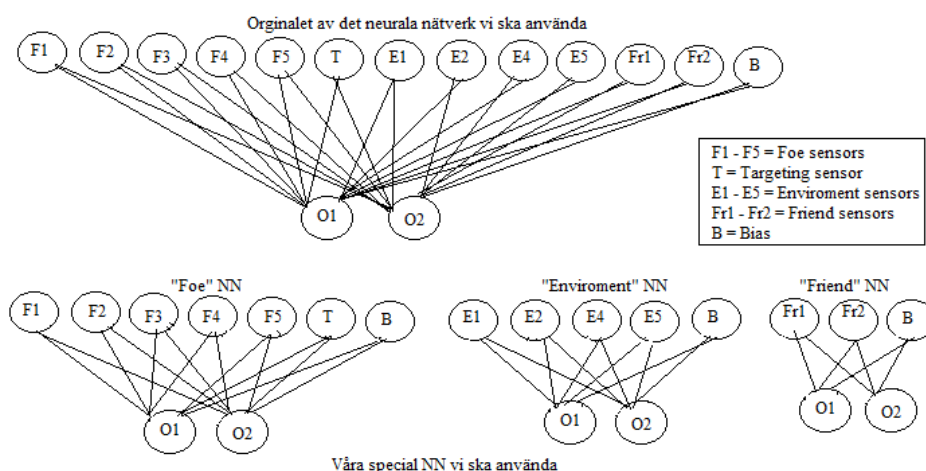
Då mitt arbete handlar om arkitekturen av ett neuralt nätverk så kommer jag inte gå in på olika parameterinställningar utan kommer att använda mig utav standardinställningarna som kommer med Open Nero. Det kommer dock ske två ändringar utav dessa inställningar vilka jag tar upp i punkt 3. Jag kommer även enbart testa en ny arkitektur då valideringstesterna tar en hel del tid att utföra.

1.7 Valideringsmetod

Då man arbetar med neurala nätverk och genetiska algoritmer så stöter man på ett problem med att man har svårt att återskapa ett exakt likadant scenario. För att jämna ut resultaten så har jag valt att använda mig av kryss-validering [10] och närmare bestämt k-vikning metoden. Denna går ut på att man delar upp sin data i K lika stora delar och sedan kör man K antal valideringar. Där k-1 tränas och den sista valideras. Detta kommer jag göra för samtliga permutationer av mina banor som kommer att användas för att försäkra mig om att det inte har någon betydelse i vilken ordning neuralnäten tränas.

2. Utförande

Jag kommer att till en början sätta mig och försöka förstå hur Open Nero och modden Nero fungerar och även NEAT då Open Nero använder sig utav detta. Modden Nero är en mod till Open Nero som är byggd efter spelet NERO. När jag har förstått hur det är uppbyggt så kommer jag att försöka dela upp det normala ANN hos botarna i Open Nero till mindre delnät men då uppdelade i en "fiende", "omgivning" samt "vän" hjärna. Med detta menar jag att för varje del-nät kopplar jag enbart på de sensorer som har med denna del av nätverket att göra. När detta är avklarat och jag har mina mindre nät så kommer jag att koppla ett fitnessvärde till varje nät för att senare kunna använda dessa i ett rouletthjul, då botarna ska ta reda på vilken delhjärna de ska använda vid just det tillfället. Detta bör minska kostnaden av användningen av ANN en aning och förhoppningsvis så kommer inte resultatet bli lidande. Jag kommer sedan även lägga in en egen specialöverkorsning där jag kommer ge möjligheten att korsa hela delnätverk.



3. Experiment

Som jag påpekade i punkt 1.6 så kommer jag att använda mig utav Open Neros standard parameter inställningar. Dock kommer två undantag ske då jag kommer att lägga till parametern av min specialöverkorsning samt så kommer jag enbart använda mig utav 25 individer för varje population. För att se inställningarna se appendix A.

Jag kommer att använda mig utav fyra stycken banor som alla innehåller en statisk fiende som roterar, denna fiende kommer att vara ”odödlig” och då inte försvinna när den tar skada. Dock så räknas en dödsopöng vid varje träff som om den enbart hade ett i hälsopöng. Jag kommer låta alla botar träna på tre utav dessa banor för att sedan valideras på den fjärde. För varje bana kommer det att bli sex stycken valideringar då samtliga permutationer av de andra banorna ska träna botarna. Bilder på banorna finns i appendix A.

Botarna kommer att köras i 15000 game ticks per bana och samtliga botar kommer att börja med samma startpopulation. Beroende på om botarna är under träning eller valideringsfasen så kommer populationen att sparas ned och data kommer att sparas men enbart data från valideringsfasen kommer att räknas i resultatet.

Under träningsfaserna så kommer vissa beteenden att belönas och dessa beteenden är att botarna ska närma sig fienden samt attackera fienden. De kommer inte bli bestraffade utav att ta skada då det skulle kunna ge fel resultat då visa botar då skulle kunna få bra fitness utav att bara hålla sig borta från fienden. För att se vilka belöningsvärden jag använde se appendix A.

Under valideringsfasen kommer inte botarna längre att få tränas utan då är det enbart resultatet som kommer vara utav intresse. Som jämförelse i mina experiment så kommer jag att använda mig utav döda/dö förhållandet utav botarnas prestation.

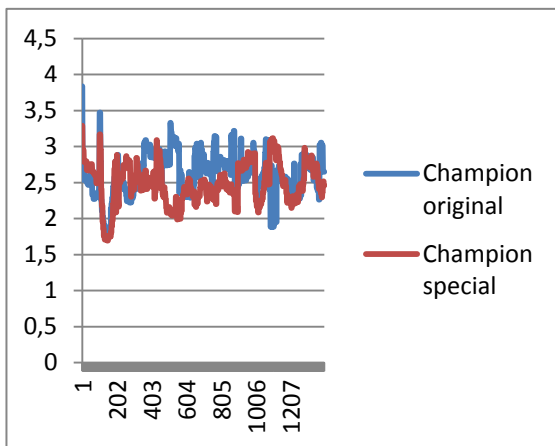
Jag kommer sedan också mäta prestandaskillnaderna mellan originalet av rtNEAT samt min variant. Jag kommer att mäta detta genom att ta medelvärdet av den tid det tar att göra en uppdatering under 60 sekunder. Denna mätning kommer att ske under träningsfas tre då näten kommer att vara som mest komplexa då de redan tränat under två banor. Jag väljer att utföra detta test på bana 4 då denna bana har flest hinder och mer att rita ut.

En tredje mätning kommer även att ske under en utav permutationerna i träningsfasen utav botarna då jag vart tionde tick kommer att spara ned den bot med bäst helhets fitness. Permutationen jag valde blev under träningsfaserna med bana 1, 2 och 3 med validering på bana 4.

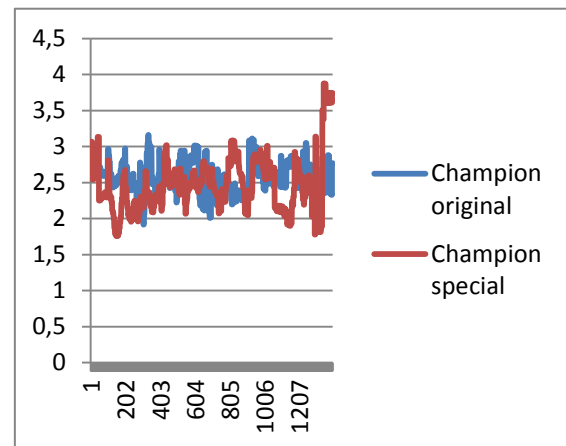
4. Resultat

Här är resultaten då jag tagit stickprov av bästa helhets fitnessvärdet för varje arkitektur. Efter varje träningsfas har populationen sparats ned och använts i nästkommande körning.

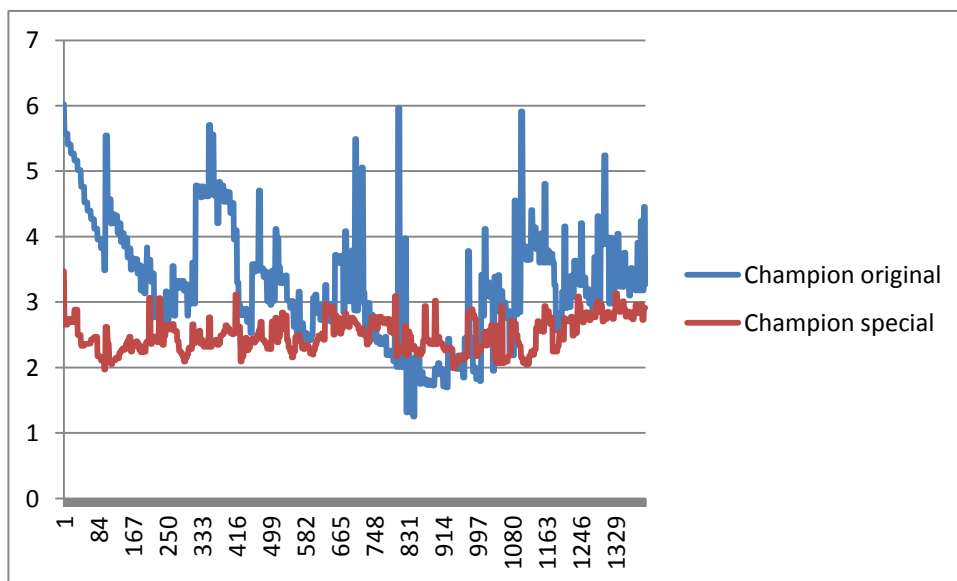
Champion är den bot med bäst fitness och med original menar jag original arkitekturen av NEAT medan special är min lösning av NEAT. I Y-led visas fitnessvärdet och i X-led visas vilket tick stickprovet gjordes.



Träningsfas steg 1

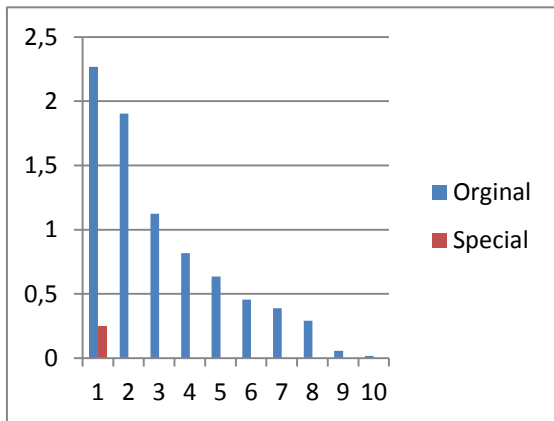


Träningsfas steg 2

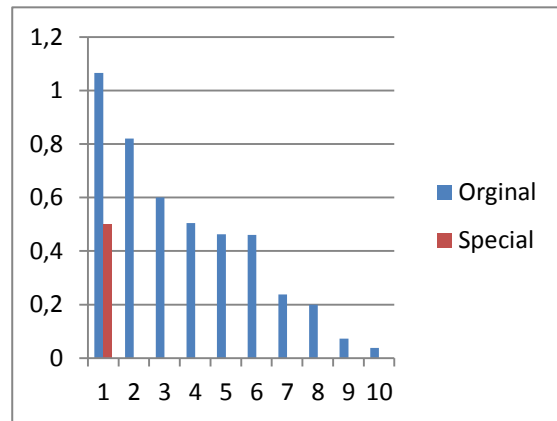


Träningsfas steg 3

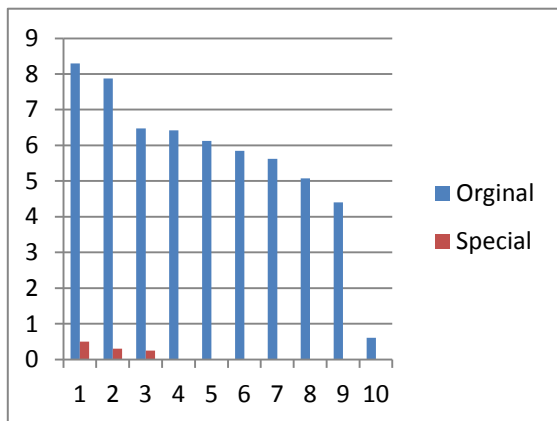
Här visas resultaten utav valideringsfasen där vi använder medelvärdet utav samtliga permutationers döda/dö förhållande per bana. Se appendix A för hela tabellen. De tio bästa individerna visas. Original är originallösningen utav NEAT medan special är min lösning av NEAT. I Y-led visas döda/dö förhållandet och i X-led visas de 10 bästa individerna.



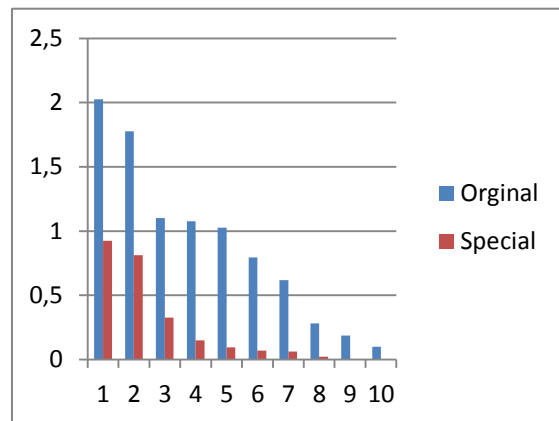
Validering på bana 1



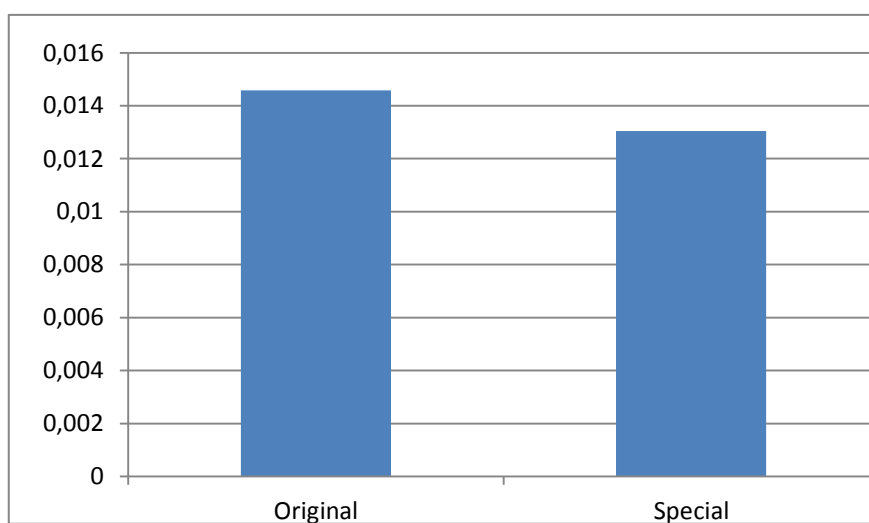
Validering på bana 2



Validering på bana 3



Validering på bana 4



Resultatet utav prestandamätningen vid körning på bana 4. För datorspecifikationer se appendix A. Medeltid för en uppdatering i millisekunder i Open Nero

5. Diskussion

Om man tittar på resultatet utav stickproven under träningsfaserna så ser man inte några större skillnader på de två första banorna för att sedan se en markant skillnad på den tredje banan. Detta tror jag beror på att alla neuralnät behöver en viss tid på sig för att justera sina vikter så de ska kunna tolka sin inputdata på "rätt" sätt. Att min arkitektur inte har blivit lika bra under den tredje rundan tror jag kan bero på att istället för att enbart ett nät ska bli bättre måste tre nät bli bättre för att boten ska visa ett bra beteende och öka sin fitness. En annan bidragande faktor kan vara att även om en viss hjärna bör användas och har hög chans att väljas i urvalsmetoden så är inte detta en garanti vilket kan påverka en hel del.

Redan genom att kolla på dessa tester så kan man anta att min variant utav arkitekturen av ett neuralnät inte fungerar speciellt bra vilket också resultatet utav valideringsstegen visar. När vi tittar på resultatet på dessa diagram så ser vi att original versionen utav neuralnätet fungerar mycket bättre än min implementation. Detta tror jag beror på att min version enbart använder sig utav den hjärnan som blev utvald och inte bryr sig om några andra sensorer som kan bidra med väsentlig information om spelvärlden. En annan bidragande faktor tror jag även kan vara att vi slumpar fram vilken hjärna som skall användas genom rouletthjuls val, vilket kanske inte är den bästa lösningen. Då originalet alltid får input ifrån samtliga sensorer så kan denna ta bättre beslut då ingen information från spelvärlden döljs.

När vi ser på valideringsdiagrammen så ser vi ganska tydligt att alla valideringar förutom den på bana tre har ganska liknande resultat på originaldelen. Den på bana tre visar en stor skillnad mot de andra banorna. Då jag skulle tro att bana fyra borde ha liknande resultat som bana tre då banlayouten är ganska lik med enda skillnaden att det är fler hinder på bana fyra. Detta är förbryllande och den ända anledningen jag tror kan spela in här är att med neurala nätverk så kan man få resultat man inte räknat med då de kan vara ganska opålitliga[2][7].

Jämför vi bana tre och fyra med min arkitektur så ser vi att de ger ganska lika resultat då det är fler som hittar till målet och lyckas döda det, samt att det är ungefär lika många botar som gör det. Detta tycker jag stämmer in på hur resultatet borde se ut mellan dessa två banor.

Om man tittar på resultatet av prestandamätningen mellan de båda arkitekturerna så ser vi att min arkitektur var något billigare prestandamässigt att använda, vilket var min tanke. Detta resultat beror troligtvis på grund utav att jag enbart använder mig utav en tredjedel utav samtliga inputnoder vilket då resulterar i att jag inte behöver processera lika mycket data som originalet behöver.

6. Slutsats

För att sammanfatta om jag tror att man kan dela upp ett stort neuralnät till mindre del-nät utan att förlora kvalitet så måste jag tyvärr säga att jag inte tror detta är möjligt baserat på mina resultat. Neuralnäten måste ha full tillgång till samtliga inputs för att kunna generera ett tillfredsställande resultat. Men prestandamässigt är det bättre att använda så små neuralnät som möjligt.

6.1 Framtida arbete

Även om min arkitektur inte fungerade speciellt bra så hade det varit roligt att se hur de olika arkitekturerna hade stått sig mot varandra. Genom att göra en implementation där man kan använda sig utav både min arkitektur och originalets arkitektur under samma körning och låta dessa träna och kriga mot varandra. Man hade även kunnat testa andra metoder för val utav delhjärna då valet utav nätverk hade en del påverkan utav slumpen i mitt arbete.

En tanke som vore intressant att utforska vore att se om man skulle kunna använda sig utav uppdelning utav ett neuralt nätverk och koppla varje delnätverks output till ett annat neuralt nätverk och låta detta göra de slutgiltiga beräkningarna. Detta för att se om man kan få snabbare upplärning utav botarna.

Något som hade varit kul att testa hade varit att använda sig utav neurala nätverk för att försöka hitta optimala parameterinställningar av NEAT, då det går att ändra väldigt många av dess parametrar. Hur dessa parametrar är inställda har en viss påverkan av hur resultatet blir och det vore intressant att se om det finns en optimal lösning.

Referenser

- [1] Online: <http://www.phil.gu.se/ann/annintr.html>
Helge Malmgren. Filosofiska institutionen. Hämtad 2012-05-04
- [2] Schwab, Brian. (2008), ”*AI-Game engine programming second edition*”. Bok. Charles river media, Hingham Massachusetts. ISBN- 9781584505723
- [3] Online: http://en.wikibooks.org/wiki/Artificial_Neural_Networks/Activation_Functions
Sigmoid function, Wikibooks. Hämtad 2012-05-07
- [4] Online: <http://nerogame.org/>, Officiell hemsida för NERO 2.0, Hämtad 2012-05-07
- [5] Online: <http://code.google.com/p/opennero/>, Officiell hemsida för Open Nero, Hämtad 2012-05-07
- [6] Online: <http://www8.cs.umu.se/kurser/KOGB05/HT02/kv00ebgfa1ck/>
En introduktion till artificiell evolution. Umeå Universitet. Hämtad 2012-05-07
- [7] Philip Ruuska Boquist. (2009-06-03) ”*Utveckling av artificiell intelligens med hjälp av Genetiska tekniker och Artificiella neurala nätverk*”. Högskolan i Skövde.
- [8] Kenneth O. Stanley, Risto Miikkulainen. (2000) ”*Evolving Neural Networks through Augmenting Topologies*”. The University of Texas at Austin.
- [9] Ylva Degerfeldt. (2005-10-21) ”*Introduktion till genetiska algoritmer*”.
Mälardalens Högskola.
- [10] Payam Refaeilzadeh, Lei Tang, Huan Liu. (2008) ”*Cross –Validation*”.
Arizona State University.
- [11] Melanie Mitchell (1998) ”*An introduction to genetic algorithms*”. Bok. Massachusetts Institute of Technology. ISBN- 9780262631853
- [12] Risto Mikkulainen et.al ”*Multiagent Learning trough Neuroevolution.*”.
The University of Texas at Austin.

Appendix A

| | | | |
|--------------------------|------|---------------------------|-------|
| trait_param_mut_prob | 0.5 | mutate_link_weights_prob | 0.9 |
| trait_mutation_power | 1.0 | mutate_toggle_enable_prob | 0.00 |
| linktrait_mut_sig | 1.0 | mutate_gene_reenable_prob | 0.000 |
| nodetrail_mut_sig | 0.5 | mutate_add_node_prob | 0.03 |
| weigh_mut_power | 0.5 | mutate_add_link_prob | 0.05 |
| recur_prob | 0.0 | interspecies_mate_rate | 0.001 |
| disjoint_coeff | 1.0 | mate_multipoint_prob | 0.6 |
| excess_coeff | 1.0 | mate_multipoint_avg_prob | 0.4 |
| mutdiff_coeff | 0.4 | mate_singlepoint_prob | 0.0 |
| compat_thresh | 4.0 | mate_only_prob | 0.2 |
| age_significance | 1.0 | recur_only_prob | 0.15 |
| survival_thresh | 1.0 | pop_size | 25.0 |
| mutate_only_prob | 0.25 | dropoff_age | 15.0 |
| mutate_random_trait_prob | 0.1 | newlink_tries | 20.0 |
| mutate_link_trait_prob | 0.1 | print_every | 30. |
| mutate_node_trait_prob | 0.1 | babies_stolen | 0.0 |
| | | brain_crossover_prob | 0.02 |

Open Nero "standard inställningar" för NEAT

| | |
|-----------------|-----|
| STAND_GROUND | 100 |
| STICK_TOGETHER | 100 |
| APPROACH_ENEMY | 170 |
| APPROACH_FLAG | 100 |
| HIT_TARGET | 200 |
| AVOID_FIRE | 100 |
| EXPLOIT_EXPLORE | 100 |

*Belöningsparametrar:
negativ < 100 > positiv*

| Total Kills | Total death | K/D | (K/D) / Total validations(4) |
|-------------|-------------|------|--------------------------------|
| 127 | 14 | 9,07 | 2,2675 |
| 198 | 26 | 7,61 | 1,9025 |
| 84 | 17 | 4,94 | 1,235 |
| 36 | 11 | 3,27 | 0,8175 |
| 56 | 22 | 2,54 | 0,635 |
| 31 | 17 | 1,82 | 0,455 |
| 14 | 9 | 1,55 | 0,3875 |
| 14 | 12 | 1,16 | 0,29 |
| 6 | 26 | 0,23 | 0,0575 |
| 5 | 64 | 0,07 | 0,0175 |

Fullständig valideringsdata för bana 1. Original arkitektur

| Total Kills | Total death | K/D | (K/D) / Total validations(4) |
|-------------|-------------|-----|--------------------------------|
| 2 | 2 | 1 | 0,25 |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 0 | 0 |
| 0 | 4 | 0 | 0 |

Fullständig valideringsdata för bana 1. Special arkitektur

| Total Kills | Total death | K/D | (K/D) / Total validations(4) |
|-------------|-------------|------|--------------------------------|
| 226 | 53 | 4,26 | 1,065 |
| 171 | 52 | 3,28 | 0,82 |
| 72 | 30 | 2,4 | 0,6 |
| 81 | 40 | 2,02 | 0,505 |
| 52 | 28 | 1,85 | 0,4625 |
| 35 | 19 | 1,84 | 0,46 |
| 20 | 21 | 0,95 | 0,2375 |
| 20 | 25 | 0,8 | 0,2 |
| 14 | 47 | 0,29 | 0,0725 |
| 12 | 79 | 0,15 | 0,0375 |

Fullständig valideringsdata för bana 2. Original arkitektur

| Total Kills | Total death | K/D | (K/D) / Total validations(4) |
|-------------|-------------|-----|--------------------------------|
| 16 | 8 | 2 | 0,5 |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 |
| 0 | 2 | 0 | 0 |
| 0 | 3 | 0 | 0 |
| 0 | 4 | 0 | 0 |
| 0 | 21 | 0 | 0 |

Fullständig valideringsdata för bana 2. Special arkitektur

| Total Kills | Total death | K/D | (K/D) / Total validations(4) |
|-------------|-------------|------|--------------------------------|
| 864 | 26 | 33,2 | 8,3 |
| 662 | 21 | 31,5 | 7,875 |
| 804 | 31 | 25,9 | 6,475 |
| 1925 | 75 | 25,7 | 6,425 |
| 759 | 31 | 24,5 | 6,125 |
| 937 | 40 | 23,4 | 5,85 |
| 1348 | 60 | 22,5 | 5,625 |
| 1260 | 62 | 20,3 | 5,075 |
| 599 | 34 | 17,6 | 4,4 |
| 216 | 89 | 2,42 | 0,605 |

Fullständig valideringsdata för bana 3. Original arkitektur

| Total Kills | Total death | K/D | (K/D) / Total validations(4) |
|-------------|-------------|-----|--------------------------------|
| 16 | 8 | 2 | 0,5 |
| 6 | 5 | 1,2 | 0,3 |
| 1 | 1 | 1 | 0,25 |
| 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 0 | 0 |
| 0 | 3 | 0 | 0 |
| 0 | 4 | 0 | 0 |
| 0 | 6 | 0 | 0 |
| 0 | 17 | 0 | 0 |

Fullständig valideringsdata för bana 3. Special arkitektur

| Total Kills | Total death | K/D | (K/D) / Total validations(4) |
|-------------|-------------|------|--------------------------------|
| 503 | 62 | 8,1 | 2,025 |
| 234 | 33 | 7,1 | 1,775 |
| 316 | 72 | 4,4 | 1,1 |
| 65 | 15 | 4,3 | 1,075 |
| 41 | 10 | 4,1 | 1,025 |
| 188 | 59 | 3,18 | 0,795 |
| 104 | 42 | 2,47 | 0,6175 |
| 28 | 25 | 1,12 | 0,28 |
| 27 | 36 | 0,75 | 0,1875 |
| 24 | 59 | 0,4 | 0,1 |

Fullständig valideringsdata för bana 4. Original arkitektur

| Total Kills | Total death | K/D | (K/D) / Total validations(4) |
|-------------|-------------|------|--------------------------------|
| 41 | 11 | 3,7 | 0,925 |
| 13 | 4 | 3,25 | 0,8125 |
| 9 | 7 | 1,3 | 0,325 |
| 3 | 5 | 0,6 | 0,15 |
| 6 | 16 | 0,38 | 0,095 |
| 2 | 7 | 0,28 | 0,07 |
| 2 | 8 | 0,25 | 0,0625 |
| 1 | 11 | 0,09 | 0,0225 |
| 0 | 24 | 0 | 0 |
| 0 | 50 | 0 | 0 |

Fullständig valideringsdata för bana 4. Special arkitektur

Datorspecifikationer för datorn som användes till prestandamätningarna:

Processor: Intel Core i7- 960 3.2 GHz

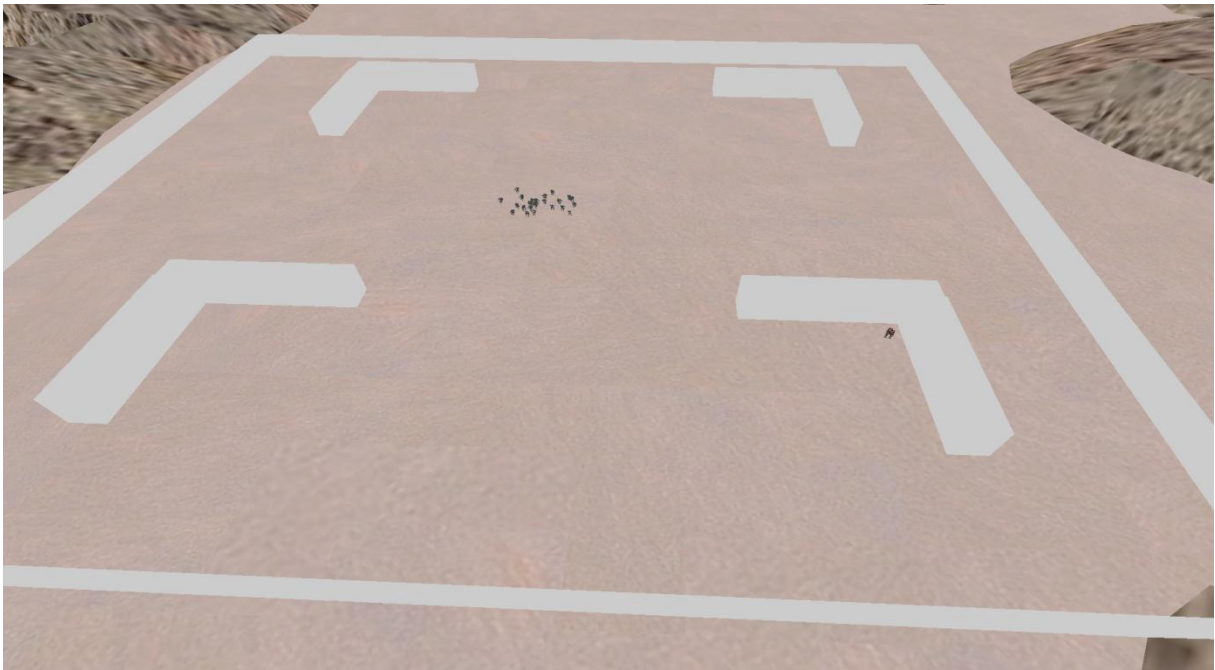
Grafikkort: NVIDIA GeForce GTX 560

Ram minne: 12 GB

Operativsystem: Windows 7 professional 64 bitars



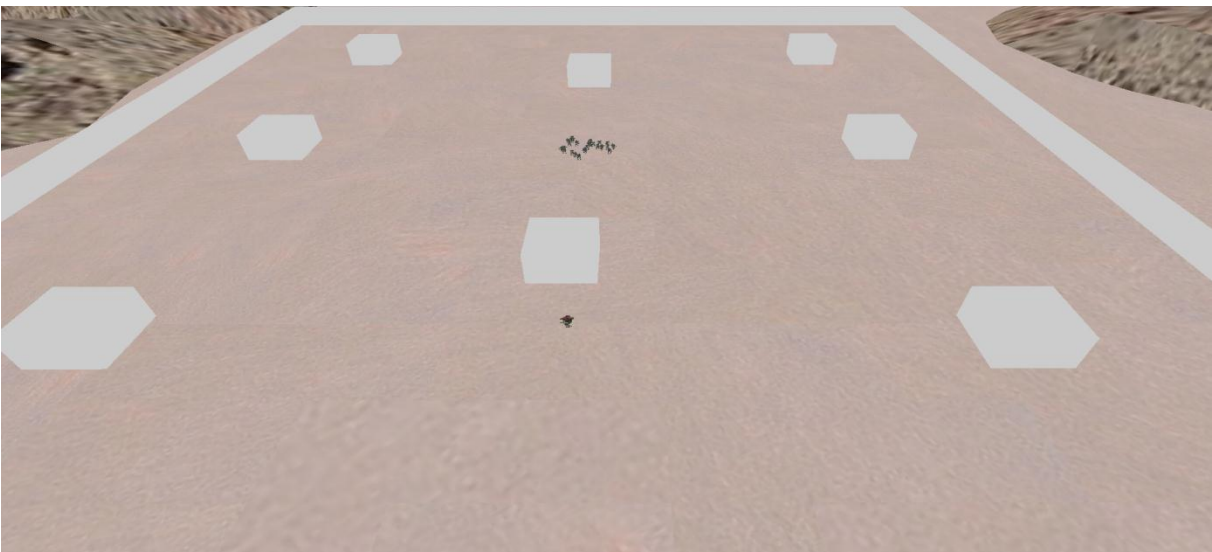
Bana 1



Bana 2



Bana 3



Bana 4