Genetic Algorithm

Artificiell intelligens för digitala spel

Simon Bothén

VT16

Ordlista

Unit(s)– En enhet i populationen i simuleringen.

GA – Genetic Algorithm

Self-check – Testar så a != b

Health – Hälsa/liv

Speed – En *Units max*hastighet i radianer/s.

Firerate – Hur snabbt en *Unit* kan skjuta.

Problemet

Min frågeställning är: ”Vilka attribut(gener) för en *Unit* är dem mest optimala i en arenamiljö”.

Att sätta alla attribut till dess maxvärde är den mest oslagbara lösningen men inte den rimligaste eller den mest passande, speciellt inte för ett spel. För att göra min *GA* mer intressant introducerade jag därför negativa effekter till samtliga attribut. Här listas de tre gener varje *Unit* har.

Health

Fördel: Du tål mer.
Nackdel: Du blir större och därmed lättare att träffa.
Omvänd effekt: Du blir mindre, men tål inte lika mycket.

Speed

Fördel: Du blir snabbare och blir svårare att träffa.
Nackdel: Du blir svagare och skadar mindre.
Omvänd effekt: Du skadar mer, men blir långsammare.

Firerate

Fördel: Du skjuter snabbare.
Nackdel: Du har mindre träffsäkerhet.
Omvänd effekt: Du blir mer träffsäker, men skjuter inte lika snabbt.

Begränsningar

Positioneringen av alla *Units* är begränsade till en oval. Rörelse kan bara ske moturs eller medurs. Fördelen är att rörelse bara sker i 1D vilket förenklar många problem då projektet har en snar deadline.

Att med diskret-matematik lösa ut vilken riktning man ska skjuta för att träffa en *Unit* med hastigheten V_0 i en oval är omöjligt. En approximation används istället.

Simuleringen kan bara köras i max 1200Hz innan det blir för mycket för min *CPU*. Det är ungefär 200x så snabbt.

Fitness-funktionen

Fitness-funktionen:

$$F = (\text{TotalTimeAlive}) + (\text{DamageDealt} * 0.05\text{f}) + (20 * \text{Wins});$$

Fitness-funktion tar hänsyn till tre olika variabler; *TotalTimeAlive*, *DamageDealt* och *Wins*. Enheten kan tänkas som “värdefulla sekunder”. Varje sekund du är i liv och i strid ökar *TotalTimeAlive* variabel med 1. Varje vunnen match är värd 20 sekunder och var 100:e skada är värd 1 sekund.

- *TotalTimeAlive* belönar en att vara vid liv.
- *DamageDealt* belönar en att ge ut mycket skada.
- *Wins* belönar en att vinna.

Tanken är att fitness-funktionen skall vara balanserad och belöna olika typer av spelstilar. Variablerna återställs till 0 efter varje generation för att börja om profileringen.

Iteration 1

Iteration 1 byggdes hela simulering upp. Och hela GA kompilerade och kördes rätt. Men vad gick fel? För att göra det lättare försöker jag förklara vad som hände i iteration 1.

Populationen: 100

Antalet generationer klarade: 13

Tid för körning: ca 18h

Selekteringen: Efter varje generation så sorteras populationen och den sämre hälften plockas bort. Sen delas den bättre hälften in i par som sen får vars två barn. Varje barn kopierar slumpmässigt ut gener från antingen mamman eller pappan till sig själv.

Mutation: Varje *Unit* har 10% chans att muteras efter varje generation. Om det blir en mutering muteras alla tre gener. Då sker en *Real-Value* mutation på varje gen vilket innebär att genen kan utvecklas både positivt och negativt. Chansen finns också att en gen förändrar sig minimalt.

Matcherna: Algoritmen för vem som möter vem var uppbyggd av en nestlad *for loop* och en simple *self-check*. Den kör på $O(n^2)$.

Endast 13 generationer på 18h var inte det jag förväntade mig, speciellt inte med en simuleringshastighet på 200x. I början gick det snabbt, ungefär 4sek/100matcher men då får man tänka på att varje generation bestod av 100^2 matcher. I slutet bestod populationen av mer eller mindre exakt lika *Units*, och både missade varandra hela tiden.

Iteration 2

För att lösa problemen i iteration 1 så krävdes det att tänka om lite. Varför stannade den på generation 13? Kan göra göra den snabbare? Och varför bestod hela populationen av mer eller mindre samma gener?

Jag började med att fixa algoritmen så istället för $O(n^2)$ så testas varje par unikt och ingen returmatch tillåtes. Detta drog ner antalet matcher med nästan hälften vilket gjorde stor skillnad hur snabbt simuleringen kunde ske.

För att lösa problemet att hela populationen hade snarlika gener så ändrade jag på mutation och selekteringsalgoritmen. Detta för att få en lite mer unik population och inte fastna i lokalmaxima. Mutationen ändrades från 10% till 25% chans, men sedan har den 60% chans för varje gen att muteras. Fortfarande enligt *Real-Value.* Selekteringen ändrades så bästa hälften får nu endast 1 barn per par istället för 2, vilket leder till att 25% av populationen blir värken föräldrar eller ersätta av nya barn.

Hur gick det?

Populationen: 60

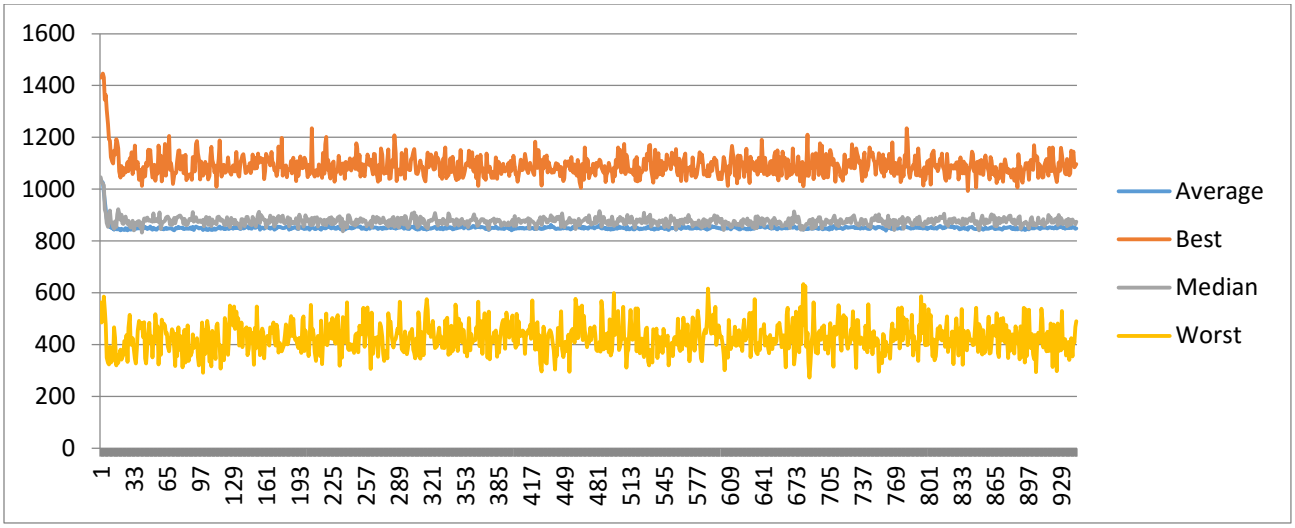
Antalet generationer klarade: 944

Tid för körning: ca 24h

Vi ser nu att vi hann många fler generationer. Se resultat nedan för hur det gick.

Resultat

Fitness-values för iteration 2:



Generation 1:

Bästa gener: ”Health: 170.000000, Speed: 0.262807, Firerate: 2.302243”
Sämsta gener: ”Health: 62.000000, Speed: 0.498032, Firerate: 1.410895”

Generation 944:

Bästa gener: ”Health: 184.000000, Speed: 0.200000, Firerate: 3.148537”
Sämsta gener: ”Health: 200.000000, Speed: 0.383973, Firerate: 3.336467”

Den *Unit* som vann hade relativt mycket i *health*, men var därför också ganska stor. Hon satsade även på skada mer än att vara snabb. För att hennes skada var så hög passade det bra med en hög *firerate* som spred ut kulorna vilket gjorde att hon träffar ofta, och starkt.

Iteration 3 – Den icke-existerande iterationen

På grund av tidsbrist hann jag inte med en tredje iteration, men vill gärna spekulera och analysera vad som hänt.

944 generationer var bra men kunde varit bättre. En bugg gjorde att när jag efter 24h kollade hur simuleringen gått hade hängt sig. Felet var att den höga 200x takten jag ställt in simuleringen på hade gjort att loggar-modulen inte fått tillräckligt med andrum att skriva ner datan till hårddisken. Detta hade kunnat lösas genom att köra loggar-modulen på en egen tråd.

Datan som GA kom fram till var långt ifrån det jag sökte. Jag förväntade mig att funktionen skulle stiga men blev istället någon ”noisy” linjär funktion. En tanke varför det blir så här är att varje *Unit* blir bättre och bättre men även motståndaren blir bättre för varje generation. Då är det möjligt att dem dödar varandra så snabbt att *TotalTimeAlive* I fitness-funktionen blir lägre än föregående generation. Istället skulle en andra population introduceras med lika många *Units* i. Denna population skulle fungera som en statisk motståndare som aldrig ändrar på sig I form av gener. Om min hypotes stämmer så skulle alla generationer vara på samma villkor; d.v.s de möter exakt samma *Units* med exakt samma gener. Då skulle varje *Unit* växa relativt till en statisk population istället för sina dynamiska grannar.

Referenser

AI Game Engine Programming - Brian Schwab

Steve Dahlskog