UNIVERSITEIT VAN AMSTERDAM

Tweedejaarsproject A.E.S.I.

Artificial Evolution and Swarm Intelligence

Werkplan

Jeroen Rooijmans
5887410

Maarten Inja
5872464

Maarten de Waard
5894883

June 4, 2010

Contents

1 Introduction

During this project we will attempt to create a program that artificially evolves tank like agents in the RoboCode environment[1]. RoboCode is a simple game, where virtual tanks can fight eachother. More information and rules of the game can be found in section 4.

It would be fairly easy to write a powerful bot ourselves, but using genetic programming[2] we can let a computer evolve these bots. To test our evolved bots, we use the RoboCode environment to generate test data of battles between our evolved bots and various enemy bots. After a certain amount of battles, RoboCode generates results, these results consist of a overall score. The score is calculated using the survival rate, damage done and ranking of the previously fought battles.

There also is a tool we use, called RoboResearch[3]. This tool gives bots a score in a similar way the RoboCode environment does, but uses default bots as enemies. We use similar program that evolves RoboCode bots created by Klaus Meffert, the program is called RoboCode with JGAP[4]. We can use this program to generate enemy bots.

Other enemies can be bots that won online tournaments, informations and rankings of these tournaments can be found on RoboRumble[5]. The code for this project will be written in Java and available on our Google Code page [6]. We planned our activities with the Gantt chart that can be found in Appendix A.

When we are done evolving the single bots, we will try to adapt our program to evolve a team of bots. These will be evolved and tested the same way as the single bots.

2 Project goals

The main goal of this project is to create a Java program that evolves RoboCode bots. These bots will evolve based on their performance, this performance needs to be tested. We use the results of multiple battles, acquired by the benchmark tool in RoboCode environment to give a bot a certain "fitness" score. The evolution is done by manipulating the genotype of the bots. Therefor we need to create a representation that can manipulate the genotype and translate it to the phenotype.

3 Method

In order to evolve a powerfull agent using genetic programming, we first need to understand the underlying principles. This implies research on two subjects, powerful behavior in RoboCode and genetic programming.

Powerful behavior is documentated on RoboRumble, information about how to design our genetic algoritm is found in the article "Evolving 3D Morphology and Behavior by Competition"Sims (1994). After acquiring enough information, we will start implementing the genetic algorithm, this will be done using the JGAP framework[7].

3.1 Genome

We have to plan and create the genome of our bots. Each chromosome represents a potential solution (strong behavior) and is divided into several genes, these genes represent various aspects of the bot (shooting, moving, ect).

These "chromosomes" result in a Java file consisting of several methods, for example robotScannedEvent and robotHitWallEvent. These methods can make calls to various functions such as rotateGun. There are countless of different variations that can be disabled, modified or enabled by reproduction, crossover and mutations.

[1] RoboCode home page: http://robocode.sourceforge.net/
[2] Genetic Programming wiki: http://en.wikipedia.org/wiki/Genetic_programming
[3] RoboResearch home page: http://robowiki.net/wiki/RoboResearch
[4] RoboCode with JGAP home page: http://jgap.sourceforge.net/doc/robocode/robocode.html
[5] RoboRumble home page: http://robowiki.net/wiki/RoboRumble
[6] Google Code page: http://code.google.com/p/aesi/
[7] JPAG home page: http://jgap.sourceforge.net/

3.2 Phenome

The phenome is the resulting behaviour of the bot. This behaviour will be tested and a bot is given a certain fitness score. This is explained in "Fitness function". Bots with a higher fitness score, meaning it has more powerful behaviour, have a better chance to pass on their genome to the next generation.

3.3 Meta language

We need to design a meta language that translates a genetic code, consisting of a string of digits, to the behavior of the bot. This means the result of translation is Java code. We have to find an appropriate representation that gives the evolutionary process enough room to evolve without forcing it in a certain direction. We hope this leads to interesting and original behaviour.

3.4 Fitness function

When dealing with evolution, there has to be a process that favors stronger genotypes. We do this by defining a fitness function. This function evaluates every chromosome and defines a fitness, which indicates how well it performs. We use the benchmark in the RoboCode environment, that calculates an overall score based on the performance in previously fought battles. By taking the survival rate, damage done and ranking of the previously fought battles in consideration, we will create a function that evaluates every bot and gives them a relative fitness score.

But to simulate these battles, we have to generate Java code, therefore we have to compile the code which consumes a lot of time. Creating a very elaborate fitness function will increase the time-costs, a very constrained one will decrease both time and accuracy, so we will have to find a balance to strike the golden mean.

3.5 Evolve

The evolutionary processes are done by the JGAP framework. Genes with a stronger fitness value have a better chance of being selected for next generations. The selected genes will be written into the Java code and the process repeats itself until either a certain amount of generations have evolved or a certain predefined fitness value is achieved. We start our evolution with blank bot, the bot genome will consist of alot of functions, but these will need to be activated, or manipulated to actually work and therefor needs to be manipulated by the evolutionary process. The genome is manipulated by three processes: sexual reproduction, crossover, and mutation.

RoboCode with JGAP uses one bot to test a new genetion of bots, in "Evolving 3D Morphology and Behavior by Competition"Sims (1994) some alternatives are explained. We will implement a "all vs. best" strategy. This means the bots from a new generation fight the fittest bot of the previous generation.

4 Background information

4.1 RoboCode Game

Time is represented in ticks. There are several things happening each tick. For example the robot is allowed a certain amount of processing time to calculate and execute actions. The game world is also updated, meaning that objects are relocated according to the physical laws of the game and potential collisions between objects and the results of such collisions are calculated.

4.2 Robot Anatomy

A robot consists of three seperate parts, the radar which is mounted on the barrel which is mounted on the body. All parts can be rotated seperately. The barrel fires bullets, the radar scans for other tanks and the body drives around in the arena.

4.3 Robocode rules

The Robocode environment has specific rules constraining our bot. I will explain the basic rules of the RoboCode game and how a bot can behave. A fight starts by placing all participating bots in the arena at random places. All bots have a certain amount of energy. Energy is lost by getting shot by an enemy or ramming walls and enemies. Certain actions also use up energy, for example shooting a bullet. A bot can gain energy by hitting an opponent.

When a bot is out of energy, it is disabled and explodes.

4.4 Programming the Robot

Programming a robot means *extending* the robot class from RoboCode. This allows us to program methods that catch events that are launched by RoboCode (such as *robotScannedEvent*) and program what we would like the robot to do in such cases (such as *rotateGun*). The robots behavior relies completely on what events occur during a battle. The robots actions during a certain event is generated by the evolutionary process.

5 Planning

5.1 Documentation

- create workplan, everybody
- round of halfway report and presentation, M. de Waard
- present halfway presentation, everybody
- finish end report and presentation, J. Rooijmans
- present final product, everybody
- maintain documentation, M. Inja

5.2 Initiation

- create planning and set up websites, M. Inja
- present workplan, everybody
- familarization with programs, M. Inja, M. de Waard
- read articles about genetic algoritms, J. Rooijmans

5.3 Genetic programming, single bots

- create genotype, J. Rooijmans
- create fitness function, M. de Waard
- evolve, M. Inja

5.4 Test phase

- benchmark with RoboResearch, M. de Waard
- compare RoboCode with JPAG, M. Inja
- design and compare with own bot, everybody
- compare with bots from the internet, J. Rooijmans

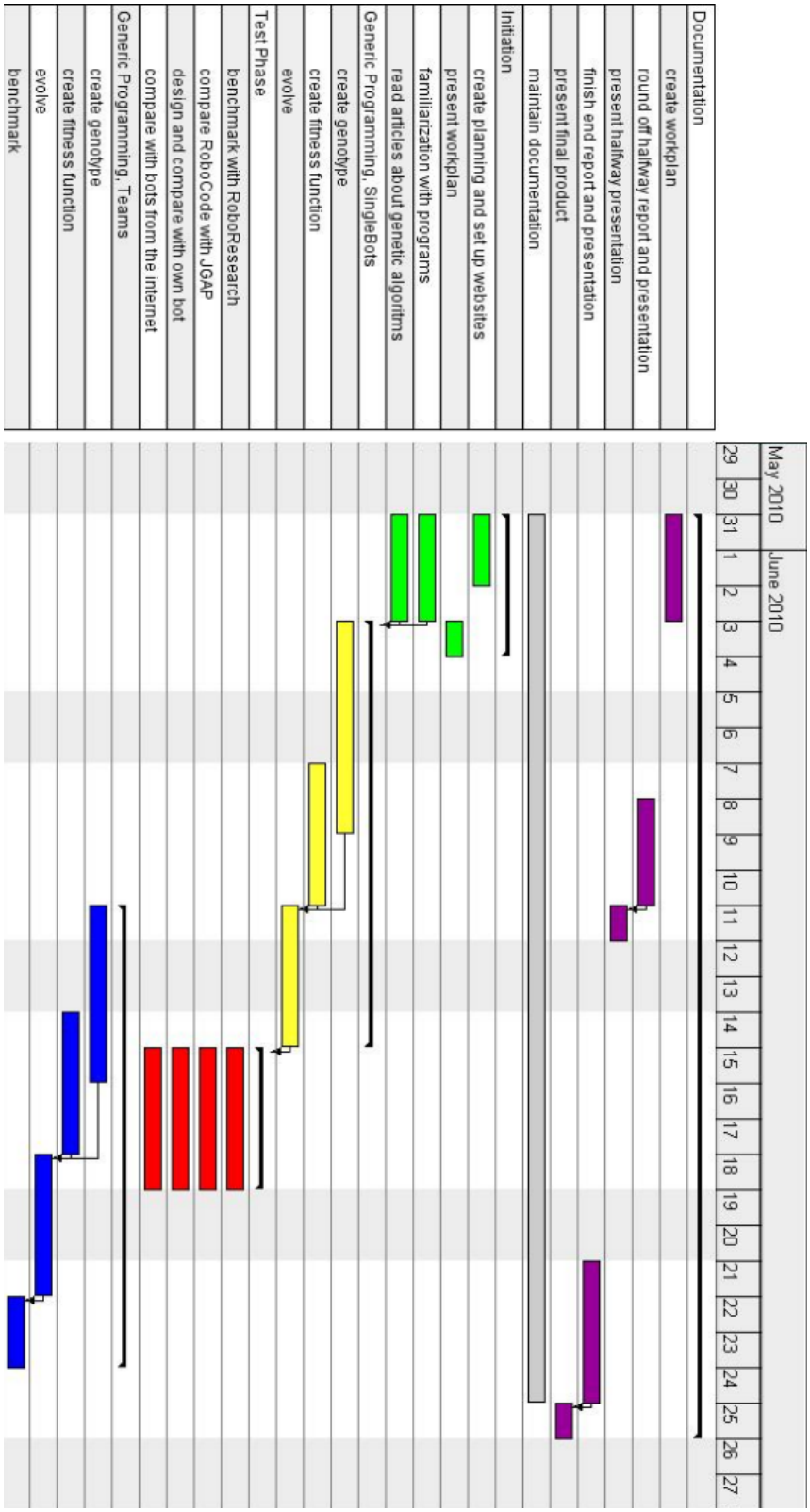
5.5 Genetic programming, teams

- create genotype, J. Rooijmans
- create fitness function, M. de Waard
- evolve, M. Inja
- benchmark, everybody

References

Sims, K. (1994), 'Evolving 3d morphology and behavior by competition', *Artificial Life IV Proceedings* pp. 28–39.

A Planning


Documentation
create workplan
round off halfway report and presentation
present halfway presentation
finish end report and presentation
present final product
maintain documentation
Initiation
create planning and set up websites
present workplan
familiarization with programs
read articles about genetic algorithms
Generic Programming, SingleBots
create genotype
create fitness function
evolve
Test Phase
benchmark with RoboResearch
compare RoboCode with JGAP
design and compare with own bot
compare with bots from the internet
Generic Programming, Teams
create genotype
create fitness function
evolve
benchmark
May 2010
June 2010
29 30 31 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27