UNIVERSITEIT VAN AMSTERDAM

Tweedejaarsproject A.E.S.I.

Artificial Evolution and Swarm Intelligence

Verslag

Jeroen Rooijmans
5887410

Maarten Inja
5872464

Maarten de Waard
5894883

June 15, 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 5.

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" [2]. After acquiring enough information, we will start implementing the genetic algorithm, this will be done using the JGAP framework[7].

3.1 Genotype

The genotype descripes a genetic constitution of the robot. The JGAP framework has predefined classes such as *Chomosome*, *Gen* and *Population* that represent the genotype. To implement evolutionary processes, we need to create two different representations of the genotype.

One datastructure holds information about certain method call that control the bot. These calls need arguments to initiate, these are also captured in the datastructure. These arguments can be mathematical formulas, control flow statements or small chunks of code that describe basic bot manoeuvres.

[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/

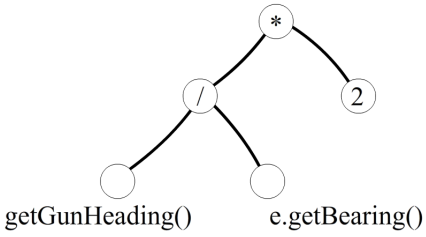


Figure 1: An expression tree.

The second datastructure was needed because we value the potential of our program to evolve impressive mathematical functions highly. An *ExpressionTree* is a binary tree holding both operators and values. Figure 1 shows the expression tree that represents a equation 1. This representation allows us to easily mutate expressions. To add, remove or alter a node one only has to recursively walk down a tree to the preferred node or leaf that needs to be altered.

$$\frac{getGunHeading()}{e.getBearing()} * 2 \tag{1}$$

These datastructures are implemented in *AESIGene*. This class is based on the *Gene* which is needed in the JGAP framework. We have decided that each relevant event such as *onHitByBullet* or *onHitWall* can be filled with Java code so each such an event represents a gene. Genes are filled with the previously mentioned datastructures and are mutated during evolution. Representing information on such a gene is currently not yet implemented but should be so in the future when we need to safe data in order to benchmark.

3.2 Phenotype

The phenome is the resulting behaviour of the bot. Currently no real powerful behaviour has been generated. This is because not every important aspect is implemented. More of this can be found in the section *result*.

3.3 Meta language

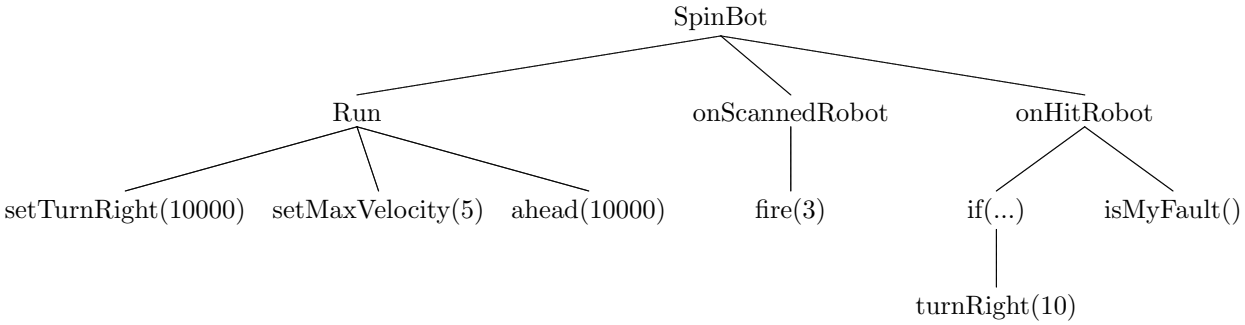
We need to design a meta language that creates java code from a genetic code. The genetic code is a string of digits, the location of the digit tells what gene is described, the value describes what this specific gene does. It is important to keep the genetic code meaningful and understandable so we can use the genetic code to "see" what kind of behaviour is evolved. This prevents our evolutional process to become a black box we use to evolve bots but where it is hard to really understand what is going on.

To properly describe how this works, we will show how java code of a simple bot, provided by RoboCode, can be represented in our genes. This is the code of the simple bot:

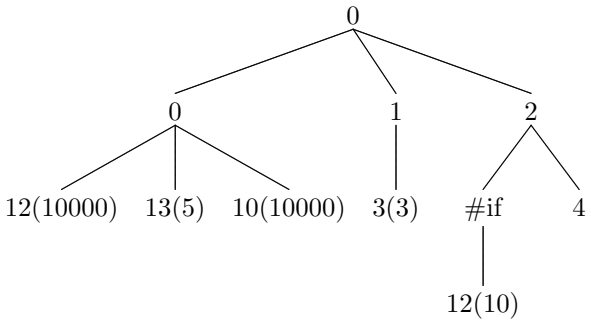
```
package sample;


import robocode.AdvancedRobot;
import robocode.HitRobotEvent;
import robocode.ScannedRobotEvent;

import java.awt.*;


public class SpinBot extends AdvancedRobot {

        public void run() {
                // Set colors
```

```
                setBodyColor(Color.blue);
                setGunColor(Color.blue);
                setRadarColor(Color.black);
                setScanColor(Color.yellow);

                // Loop forever
                while (true) {
                        setTurnRight(10000);
                        setMaxVelocity(5);
                        ahead(10000);
                }
        }

        public void onScannedRobot(ScannedRobotEvent e) {
                fire(3);
        }

        public void onHitRobot(HitRobotEvent e) {
                if (e.getBearing() > -10 && e.getBearing() < 10) {
                        fire(3);
                }
                if (e.isMyFault()) {
                        turnRight(10);
                }
        }
}
```

This code can be described in a tree, like this:



The computer will see the tree like this:



Here the upper number is an index of the gene, and the numbers on the second row are numbers indicating the index of the gene array. Under that we have array indices of the methods and their arguments, that are positioned in arrays with all the methods we can call and the arguments we can give them.

3.4 Fitness function

Fitness is crucial to determining the trajectory of the evolutionary process as a high fitness score will increase the chance a chromosome is transferred to the next generation of bots.

With the fitness function, we translate battle results into a appropriate fitness value. Our basic scoring measure is the fractional score F, which is computed using the score gained by our bot S_b and the score gained by its adversary S_a (Equation 2).

Calculating the fitness score this way, we encourage our bot not only to maximize its own score, but to do so at the expense of its enemy.

$$F = \frac{S_p}{S_p + S_a} \tag{2}$$

In early stages of the evolutionary process, bots obtain no points at all. To enhance population diversity during the initial phase of evolution we created a modified fitness function (Equation 3).

$$F = \frac{\alpha + S_p}{\alpha + S_p + S_a} \tag{3}$$

We added a small fixed constant α. Because of this modification a bot that scores a 0 for S_p but prevents his adversary to score a high S_a receives a higher fitness score.

3.5 Evolve

Because we still need to finish some major functions we have not started evolving as of yet. We do have some results which can be found in the section *results*.

4 Results

Currently our program has evolved a single bot, which is capable of doing something. The results are, as discussed in previous sections, minimal, but prove promising. The robot shows action such as moving forward and rotating both it's body and gun but not yet a change in behaviour when events are caught. This means we completed the basis but need to focus on the evolutionary process that will create bots with more interesting behaviour.

We need to be able to compare our results to other bots to give the results a meaning. During this project we will create some bots ourselves and download bots that have proven to be powerfull in online tournaments. These bots will be battling against our bots. This way we can find out if the bots evolved with genetic programming can stand against human made bots. There is a paper "GP-Robocode: Using Genetic Programming to Evolve Robocode Players" [1] about RoboCode bots evolved with a genetic framework that participated in a online tournament with 27 participants. Other bots were all human written. The bot ("GPBot") ended third.

5 Background information

5.1 RoboCode Game

The RoboCode game is a battle between virtual tanks. These can move around and shoot. 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.

5.2 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.

5.3 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.

5.4 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.

6 Planning

6.1 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

6.2 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

6.3 Genetic programming, single bots

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

6.4 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

6.5 Genetic programming, teams

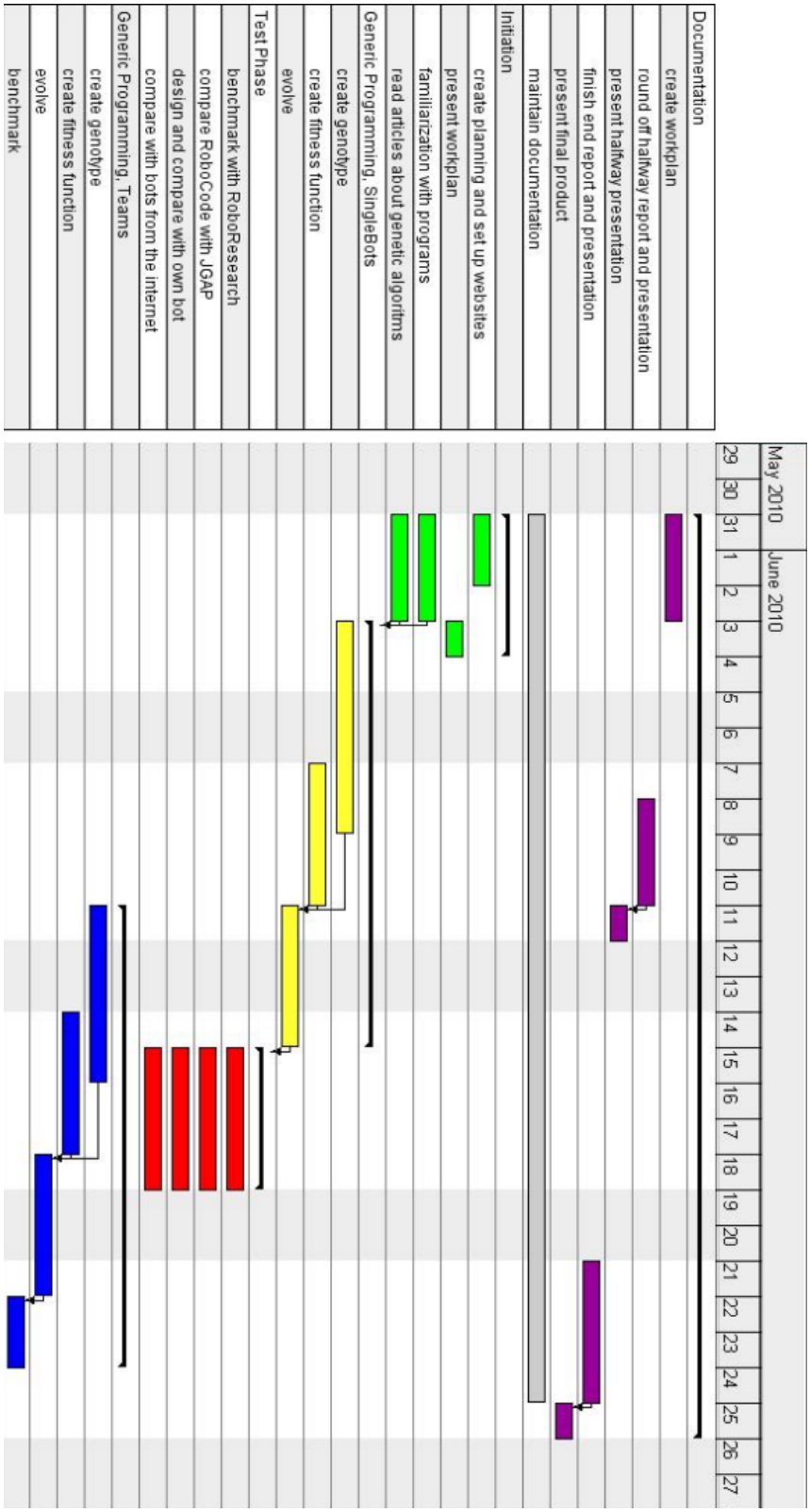
- create genotype, J. Rooijmans
- create fitness function, M. de Waard
- evolve, M. Inja
- benchmark, everybody

References

[1] Y. Shichel, E. Ziserman, and M. Sipper. GP-robocode: Using genetic programming to evolve robocode players. In M. Keijzer, A. Tettamanzi, P. Collet, J. I. van Hemert, and M. Tomassini, editors, *Proceedings of the 8th European Conference on Genetic Programming*, volume 3447 of *Lecture Notes in Computer Science*, pages 143–154, Lausanne, Switzerland, 30 Mar. - 1 Apr. 2005. Springer.

[2] K. Sims. Evolving 3D morphology and behavior by competition. *Artificial Life*, 1(4):353–372, 1994.

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