Distributed Artificial Intelligence

Group Report

Project Warbot

Submitted By :
Kexin SHAO, You ZUO

1 Presentation of Context

Warbot is both a game and a platform for the evaluation and analysis of techniques of coordination between agents in a competitive situation where two teams of "robots" compete to kill the base of the opponent.

In this project, we are the developers of the agents. But we must do only one thing: develop the "brains" of these robots knowing that the "bodies" (body) are defined once and for all by the rules of the game. Therefore, the competition resides in the quality of the programming of these brains, and the proposed coordination strategies.

We will point out in the following report how we analyzed the possible situations and how we solved the problem of improving our team's odds of winning.

2 Analysis of Green team

From the original archive, we got the codes for both teams: green and red, and we are supposed to modify only codes of red team (red.nls). So, first of all, we analyzed the settings of green-team. In this way, we could find the shortcomings and the inspiration to concept our strategies. Below we will point out our analysis process step by step.

2.1 Analysis of functions in green.nls

```
to green-team-ask-for-energy [ b n ]
  ask b [ if (energy > 1000) [ give-energy myself n ]]
end
```

The function **green-team-ask-for-energy [b n]** in the green team means a robot asks n unities of energy from its nearest base b. What we are interested in is the limit value 1000 for the base b to give its energy to the robot, so we can modify it to attain different performances:

increase the value 1000 $\Longrightarrow$ base b will restore more energy in order to create more weapons, walls or robots

decrease the value 1000 $\Longrightarrow$ robot can get energy more easily from the base b

```
to green-team-go-back-to-base
  let b min-one-of my-bases [ distance myself ]
  if (b != nobody) [
    if ((breed = Harvesters) and (distance b > 10) and (distance b < 11)) [drop-wall]
    ifelse (distance b <= 2)
      [
        give-food b carrying-food?
        if (energy < 1000) [ green-team-ask-for-energy b 300 ]
        if ((Breed = RocketLaunchers) and (nb-missiles = 0)) [
          green-team-ask-for-energy b 500
          new-missile 5
        ]
        set mem4 0
        rt 180
      ]
      [
        set heading towards b - 20 + random 40
        ifelse (free-ahead? speed = nobody)  [ forward-move speed ][ rt random 360 if (
    free-ahead? 2 = nobody) [ forward-move speed ]]
      ]
    ]
end
```

This function indicates that robots will find the nearest base b and then go back to it.

If the robot is a harvester, it will drop the wall (if it has taken one) to the patch 10 to 11 units away from the base station. This action means that harvesters drop the wall to protect the base from 10 units of distance. Else if it gets 2 units away from the base b, it will give the carrying food to base b.

If robots' energy is less than x= 1000, it will ask 300 energy from base b.

If the robot is a Rocketlauncher and it has no missile, it will ask 500 energy from base b.

We consider the modifications from the following points:

1. Adjust the distance of drop-wall.
2. Change the energy to supplement for each breed of robots.
3. We should pay attention to the energy's lower bound 1000. If this value is too small, the robots are probably not able to get back.
4. We notice that the rocketlauncher could carry more missiles. The maximum missiles it can take is 1000. So the number of missiles the green team has set, which is 5, could be too small.

```
to green-team-harvesters-go-and-eat
  let w min-one-of (perceive-walls) [distance myself]
  let b min-one-of my-bases [ distance myself ]
  if ((w != nobody) and ((distance b < 10) or (distance b > 11))) [take-wall w]
```

Regarding this function, it shows that robots will take the wall which is less than 10 units or more than 11 units away from the base b. It means the green team will put all walls between 10 and 11 units away from the base to protect the base. So the modification we can make to this function is to change the distance to drop walls.

```
to green-team-select-target
```

After reading this function, it could tell us: if the robots of green team doesn't have a enemy as target, we could probe a enemy target and memorise its coordinate.

This action gives us an idea. We could add another function named select-base-target,this function could select a enemy base as target cause our purpose is to destroy all of enemy's base. We consider this function will help a lot.

```
to green-team-drive-harvesters
```

This function inspired us that different kinds of robots can drive each other mutually.

```
to goGreenExplorer
  ifelse (energy < 1000)
  [ green-team-go-back-to-base ]
  [ green-team-go-and-eat ]

  green-team-drive-harvesters

  ifelse (not green-team-no-target?) [
    green-team-call-rocket-launcher-xy mem0 mem1 mem2
    green-team-call-explorer mem0 mem1 mem2
  ]
  [ let h one-of perceive-robots ennemy
    if ( h != nobody ) [ green-team-call-rocket-launcher-t h [breed] of h]
  ]
end
```

This function shows us how explorers work. The explorer will find a food and then give its location to a near harvester. Also, if the explorer has found an enemy, it will give this enemy's position and breed (mem0,mem1,mem2) to a near rocketlauncher who doesn't have a target.

And after analysing the logic of this function, we considered a better way to cooperate between robots:

1. Firstly, if explorer is nearer to the base and it has less energy, we could let explorers take foods from harvesters instead of only let harvesters taking food to the base.
2. Then, we find that when the explorer has an enemy as the target, it will call these two functions green-team-call-rocket-launcher-xy and green-team-call-explorer to give the location of the target to the rocketlauncher, and it will let another explorer to make sure that the enemy target is dead. There will be a drawback to this: when the called explorer has a goal, it will be given a new goal because of being called, which is irrational. So we came up with a better way to solve this problem. We can first try to find an explorer without a target and then determine the next action. Of course, we may not find it. If we cannot find it, we will find the nearest explorer instead.

```
to goGreenRocketLauncher
```

```
to goGreenHarvester
```

```
to goGreenBase
```

2.2 Strategies not used in green.nls

According to the description file of the game rules and the functions above, we have noticed that actually there are still many strategies that the green team did not utilized.

For example:

- Base can breed walls in order to defend an attack or block enemy robots;
- Harvester robot can give its burgers to an explorer(because explorers move faster than the harvesters do, so explorers can convey energy to base quickly);
- All the four kinds of robots can call and drive other robots in its perception range, the green team used this method only when explorer find some food and then drive an near harvester robot.

3 Strategy plans

What we concept is a relatively aggressive strategy that we want, first of all, to maintain sufficient resources for our teammates as quickly as possible and then actively attack the enemy. Here we represent our plans for each category of the robot.

3.1 Strategies for Bases

As for bases, we have three memory values mem6, mem7, mem8, which represent respectively the number of harvesters, rocket-launchers, and explorers to create.

1. At the first stage, we want to obtain more resources as soon as possible, so we decided to create at least 20 harvesters, and 4 rocket-launchers also explorers.

```
to initRedBase
  new-Harvester self
  set mem6 20
  set mem7 4
  set mem8 4
end
```

2. Then for each tour, we want to improve the rate to build first of all more harvesters, and then rocket-launchers, so we have set the possibilities of creating a harvester, an explorer and a rocket-launcher respectively $\frac{1}{2}$,$\frac{1}{6}$ and $\frac{1}{3}$

```
if (energy > 10000) [ ifelse (random 2 = 0) [ set mem6 mem6 + 1 ][ifelse (random 3 =
     0) [ set mem8 mem8 + 1 ][ set mem7 mem7 + 1 ]]]
```

3. Finally, for the lower bound of energy to give energy to other robots in the same team, we wanted to implement **space behavior**, but it seemed it required an input, but we did not know what it was supposed to be, so we gave up. After that, we set a slider to do the parameter sweeping. By varying the lower-bound-rng from 1000 to 3000, finally, we get the optimal value equals to 2500.

```
to swimming-ask-for-energy [ b n ]
  ask b [ if (energy > 2500) [ give-energy myself n ]]
end
```

4. Bases are capable of building walls, but since we chose an "aggressive" plan of strategies, we want to create more robots and weapons to attack rather than build walls to defend. So we omitted the function of building walls.

3.2 Strategies for Explorers

For explorers, since it has the most extensive perception range and the fastest moving speed. We want them to cooperate with other kinds of robots so that everybody could give full play to their best advantage.

1. To better cooperate with harvesters, we copied the function *drive-harvesters* used by the green team. When an explorer has detected a food in its perception range, it will call the nearest harvester to consider this food position as its target.

2. To better cooperate with harvesters, we designed two functions which were inspired by the function above: *drive-base-shooter* and *drive-harvester-shooter*

```
to swimming-drive-base-shooter
  let b min-one-of perceive-base ennemy [ distance myself ]
  if (b != nobody) [
    swimming-call-rocket-launcher-t b Bases
  ]
end

to swimming-drive-harvester-shooter
  let b min-one-of perceive-specific-robots ennemy Harvesters [ distance myself ]
  if (b != nobody ) [
    swimming-call-rocket-launcher-xy [xcor] of b [ycor] of b Harvesters
  ]
end
```

   Explorers will drive a rocker-launcher when they have detected a near enemy base since base usually possesses more energy, and we have to choose faf, which has more substantial attack power. The same thing for when explorers have detected an enemy harvester, but this time we will use a missile. Because to kill a robot, it doesn't worth wasting a faf. Moreover, harvesters move slowly, and usually, they are likely to be closer to other harvesters so that a missile would be a better choice.

3.3 Strategies for Rocket-Launchers

1. To cooperate with explorers, rocket-launcher need to check if explorers have set it a target, so we add this line to the front of *goRedRocketLauncher*

```
if(not swimming-no-target?) [ swimming-shoot ]
```

   So if a rocket-launcher does not has a goal set by an explorer, it will continue to find goals by itself or decide to go back to base or not.

2. Besides, rocket-launchers are supposed to attack the enemy's bases. That's to say we need them to be capable of creating fafs. So we also added this command to *goRedRocketLauncher*:

```
if ((energy > 1200) and (nb-fafs < 3)) [ new-faf 5 ]
```

3.4 Strategies for Harvesters

1. Harvesters are capable of picking up food, but it moves so slow, which is not beneficial for supplementing energy for bases. Therefore, we decided to add an order, that whenever a harvester finishes picking up foods around it, it will check if there is an explorer nearby. If it successes with finding one explorer it will give all its carrying food to this explorer:

```
let ex min-one-of  perceive-specific-robots friend Explorers  [distance myself]
if (ex != nobody) [ give-food ex carrying-food? ]
```

2. Another strategy is that we want to prevent enemy harvesters from picking up food, and the most natural place to find a harvester is where other harvesters are. With this assumption, we hope our harvesters can call a rocket-launcher to shoot the enemy harvesters around with an accurate rocket faf:

```
let h_ad min-one-of perceive-specific-robots ennemy Harvesters [distance myself]
if (h_ad != nobody) [ swimming-call-rocket-launcher-t h_ad [breed] of h_ad]
```