



NTNU – Trondheim
Norwegian University of
Science and Technology

Faculty of Information Technology, Mathematics and Electrical Engineering
Department of Computer and Information Science

TDT4280 - Exercise 1

Alexandre Balon-Perin

Chapter 1

Purely Reactive Agents

1.1 Theoretical Questions

1.1.1 Describe the environment in Robocode

The environment in Robocode is inaccessible. In fact, the agent has the ability to scan the environment and retrieve accurate and up-to-date information about it but this information is not complete. The agent is only capable of detecting one enemy at a time. The state of the environment is not known in the sense that our agent does not know the description (position, speed,...) of all agents at a given time.

The Robocode's environment is non-deterministic because the actions of the agents do not have a single guaranteed effect. (e.g. if the agent shoot at another agent, it is unsure that the bullet is going to hit the targeted agent.)

The environment is dynamic of course. It changes because of the other agents and not only because of our agent.

Eventually, the environment is design to be continuous. The variables such as position and bearing are of type "double". This means that they can theoretically have an infinite number of values. However, as we know, a computer only simulates continuity. As a consequence, the variables of type "double" have an enormous range of values but not an infinite one.

1.1.2 Discuss the limitations versus the advantages of purely reactive agents

The limitations and advantages of purely reactive agents are discussed in the Wooldridge book, chapter 5, section 5.2, page 97.

A purely reactive agent can be describe as a system that perceives its environment and trigger actions according to what was perceived. This kind of agent only uses the information about the present state in the decision making process. The ability to react to its environment is definitely an advantage of purely reactive agent in a changing environment such as Robocode. The decision making process is simple and computationally tractable which makes this type of agent very robust against failure.

However, the agent must have enough information about its local environment in order to make the right decision. Also, the impact of an action on the non-local environment is uncertain and might, as a result, disadvantage the agent. The inability of purely reactive agents to learn is a major problem in this kind of design as they use only information available at the present state. Finally, the overall behaviour emerges from the interaction between the different components meaning that the process to reach this averall behaviour is not understandable. An agent designer

will have a hard time designing an agent to fulfil a specific task in these conditions.

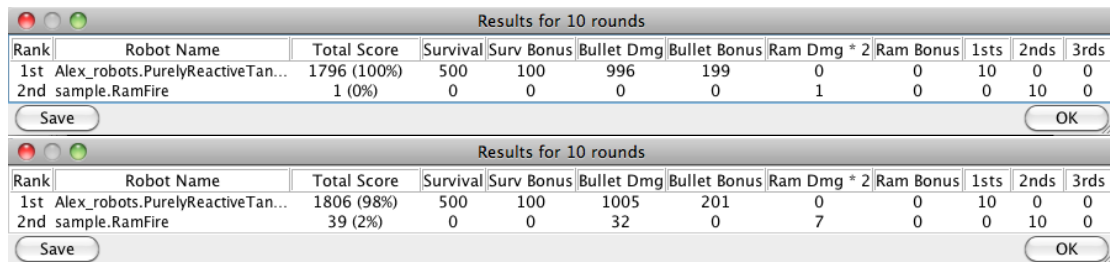
1.2 Programming in Robocode

The motion of the tank, its gun and radar, are inspired by the following tutorial:

<http://mark.random-article.com/weber/java/robocode/>

The purely reactive agent behaviour is the following:

1. The agent stands still and scans the neighborhood until it finds an enemy.
2. If an enemy is detected:
 - (a) Aim at it
 - (b) Fire if the distance to the target is smaller than a threshold
 - (c) Move to the side around the target



Rank	Robot Name	Total Score	Survival	Surv Bonus	Bullet Dmg	Bullet Bonus	Ram Dmg * 2	Ram Bonus	1sts	2nds	3rds
1st	Alex_robots.PurelyReactiveTan...	1796 (100%)	500	100	996	199	0	0	10	0	0
2nd	sample.RamFire	1 (0%)	0	0	0	0	1	0	0	10	0

Save OK

Rank	Robot Name	Total Score	Survival	Surv Bonus	Bullet Dmg	Bullet Bonus	Ram Dmg * 2	Ram Bonus	1sts	2nds	3rds
1st	Alex_robots.PurelyReactiveTan...	1806 (98%)	500	100	1005	201	0	0	10	0	0
2nd	sample.RamFire	39 (2%)	0	0	32	0	7	0	0	10	0

Save OK

The purely reactive agent always win but the score varies depending if the agent gets stuck in a corner or not.

Chapter 2

Agents with State and Utility

2.1 Theoretical Questions

2.1.1 Discuss the advantages and limitations of agents with state and utility versus purely reactive agents

Adding states and utility to an agent increases greatly its complexity. For example, the agent programmed for this assignment precalculates the future position of its enemy. This works fine but an even greater way would be to add learning ability to the agent so that it could "understand" the motion pattern used by Crazy and calculate the exact position where it should fire at. These new features make the agent more "intelligent" but increase drastically the complexity of its structure. As a result, its robustness can be affected. Only agents using states can learn things from previous mistakes. Learning is an important part of artificial intelligence but it requires more than 10 rounds in order to obtain good results.

2.1.2 Elaborate on how much work it was to program a state-utility agent versus a reactive agent that could beat ramFire

As explained in the previous paragraph, adding state and utility to an agent increases the complexity of the code. The calculus to determine the future position of the enemy requires knowledge of mathematics and basic mechanics as well. This future position calculus is not the best against Crazy because this agent often changes direction. This kind of state is a short-term optimization for the agent. However, no utility function was needed to implement this new feature. A new option would be to allow an agent to learn the optimal distance to fire against one opponent but in practice, it requires much more than 10 runs to obtain acceptable results. In all cases, it requires much more work to program a state-utility agent than a reactive agent.

2.1.3 Describe and justify your utility functions

The utility function is:

$$U(Energy) = \max(Energy, BestEnergy)$$

At the end of the round, the energy of the agent is compared with the best value of the energy obtained so far. If this new value is better than the previous one, it is updated and the value of the distance to fire is also updated. The score depends on the agent's energy at the end of each round so trying to maximize this value is a good idea to improve the agent's behaviour.

Obviously, it is a very simplified application of state and utility because the fact that the energy of the agent is high at the end of the round does not depend entirely on the distance to fire.

A local search is performed during three rounds each time a better value is found for the energy. After a predefined round (6 here), the value of the best distance to fire is used in the last games. Again, it would be much better to run the learning phase hundred of rounds to obtain an optimal value.

2.2 Programming in Robocode

The calculus required to determine the future position of the enemy was implemented following the tutorial on Linear targeting at:

http://robowiki.net/wiki/Linear_Targeting

Some comments have been added to prove the understanding of this code.

The state and utility agent's behaviour is the same than the purely reactive one but when aiming at the target, the agent precalculates the future state of the targeted agent. This is very useful for moving targets such as Crazy.

The result against ramFire is quite good:

Results for 10 rounds											
Rank	Robot Name	Total Score	Survival	Surv Bonus	Bullet Dmg	Bullet Bonus	Ram Dmg * 2	Ram Bonus	1sts	2nds	3rds
1st	Alex_robots.StateUtilityTan...	1806 (99%)	500	100	1005	201	0	0	10	0	0
2nd	sample.RamFire	17 (1%)	0	0	16	0	1	0	0	10	0
Save OK											

The result against Crazy is not perfect but still satisfying:

Results for 10 rounds											
Rank	Robot Name	Total Score	Survival	Surv Bonus	Bullet Dmg	Bullet Bonus	Ram Dmg * 2	Ram Bonus	1sts	2nds	3rds
1st	Alex_robots.StateUtilityTan...	1290 (90%)	400	80	685	124	0	0	8	2	0
2nd	sample.Crazy	151 (10%)	100	20	28	1	2	0	2	8	0
Save OK											