CS255 – ARTIFICIAL INTELLIGENCE

My RoboCode Bot is a rational agent and therefore I have used PEAS's to measure the effectiveness and the purpose of this bot. The aim of RoboCode is to win the battle by scoring as highly as possible myself and keeping the score of the enemy down. This is achieved my destroying enemy tanks and avoid being damaged myself. The environment consists of the battlefield, enemy tanks and bullets fired by enemy tanks and myself flying everywhere. The actuators or effectors are consists of the engine, the brakes and the gun of the tank. The engine and the brakes of the tank are used for movement whilst the gun is used for firing bullets of various sizes. The bullets are fired in a way to maximize the chance of them hitting the enemy. In order to do this, a radar exists which is used to gather various pieces of information about the enemy such as the relative bearing from my current position, its current velocity, current heading etc. For the design of my robot, I have used aspects from the hill climbing algorithm. For example, I have started off with a default targeting system which is designed to hit a certain type of robot. If this works, then it stays as it's the optimal. However, if my tank keeps missing the enemy tank, then the targeting system is switched. Similarly, I vary the bullet size as well to try and find the local optimal for the tank using two different heuristics. I have used this AI technique in my robot design as it is not possible to create an optimal tank for all kinds of enemy tanks. One feature will be good against one kind of enemy whilst failing miserably against a different kind of tank. As a result, a good way to beat a tank is to find a local optimal solution against that tank. This contains some aspects of the hill climbing algorithm.
When designing my robot, I have broken up the entire task into different sections. The different sections are:

- The targeting system
- Bullet evasion
- Radar Tracking
- Selecting the optimal bullet power

For the targeting system, I have implemented two different kinds of targeting systems, stationary targeting and linear targeting. Out of all the targeting systems, stationary targeting is the simplest one to implement. This is due to the lack of any complicated calculations. It involves firing a bullet at the current position of the enemy and hope the enemy does not move. In order to do this, the radar needs to scan the enemy and get the relative enemy bearing. After this, the gun is turned to the left or right depending on the relative bearing and a shot is fired. As this usually requires the enemy to remain at one spot in order for this to be effective, this type of targeting system is generally avoided as it has a poor hit rate to straight tanks in mobile tanks.

Linear targeting [1] is a targeting system which involves firing a bullet at an angle not directly aiming at the enemy but ahead of the enemy. This enables the firing tank to hit a moving tank by predicting where it is going to be in the very near future. This involves the radar scanning the enemy tank to get some information such as the enemy heading, velocity and the enemy bearing. Using these pieces of information and together with some trigonometry, we are able to predict the position of the enemy tank in the very near future. Using this and the velocity of the bullet we are going to use, we are able to get angle at which by firing our bullet now, we can hit the enemy tank soon. This sort of targeting system assumes that the enemy will keep moving at their current velocity without changing the direction. It is very powerful against robots like "Walls" which move in a straight line through the battle. However, most tanks do not move in a straight line continuously. Oscillating robots such as "My first robot" and spinning robots like the "Spin bot" are partially immune to this form of targeting due to their perpetual change in velocity.

The artificially intelligent robot tank will be battling against several different kind of enemy tanks with their own bullet dodging movements and techniques. As a result, using just one kind of targeting system makes us vulnerable to missing too many bullets, wasting time and precious energy. As a result, it is better and more effective to use both linear targeting and stationary targeting dynamically. Linear targeting is used for straight moving robots like "Walls". It is very vulnerable to this system as it always gets hit by my bullets due to prediction. This combined with my bullet dodging algorithm, it lets me win most of the battles against walls. On the other hand, when linear targeting is used against oscillating robots, the bullets go flying to the sides due to their velocity when the robot is scanned. My implementation of the targeting system includes how to decide on the best targeting system against a robot in and 1 vs 1 environment.
My robot starts off with linear targeting on my default. If my robot misses 5 shots in a row then my robot automatically switches stationary targeting from linear. This is because from testing, I have noticed that sometimes my implementation of linear targeting misses when the robot temporarily stops and starts to move again. In this situation, it would be bad to change the targeting system immediately as stationary targeting would be futile against any linear moving robots. From my personal experience and testing, I have found that stationary targeting works better for oscillating and spinning robots than linear. Like my robot changes its targeting system from linear to stationary, depending on the correct situation, it does the exact opposite. Instead of waiting for 5 misses in a row to switch to linear, it waits for 10 misses in a row. This is due to the fact stationary targeting has a much lower hit rate. As robots are oscillating or going round, all the bullet hits are just by chance.

To be successful in RoboCode, it is very important to avoid being hit by enemy tanks just as important, if not more to hit enemy tanks myself. There are various strategies that can be implemented to evade enemy bullet. It can be moving randomly in any direction like "Crazy" bot, move in a circle like "SpinBot" or have an oscillatory movement like "My first bot". Out of all the various choices, I have chosen to go with "Stop n' go" oscillation for my bullet dodging technique. This is because although "Crazy" is more successful at dodging bullet as it does not follow any pattern, it is much harder to damage an enemy with a normal "Robot" class. Without having the advantage of multi-threading with "Advanced Robot", moving my tank and shooting simultaneous would not be possible to implement. Shooting when my robot had stopped proved quite successful during testing.
My robot is able to detect when the enemy has fired a bullet by looking at its energy levels. When the enemy fires a bullet, its energy level drops by the energy of the bullet fired. This can be as low as 0.1 to as high as 3.0. As a result, by tracking the energy drops between these amounts, we know when a bullet has been fired. To dodge it, all we have to do is move out of the way. My robot does not move every single time a bullet has been fired. Moving every time would waste too much time moving. Furthermore, moving too much in an oscillatory motion would also mean I would get hit a lot by the enemy as they could just fire in both the positions I could potentially be in by using pattern matching. By not moving after every single bullet has been fired by the enemy, I still get hit sometimes. However, in my opinion this is more effective in dodging bullets as I have found out from testing.

Just like choosing the correct targeting system is important, so is choosing the right bullet power [3]. The right bullet power will enable us to conserve energy by not wasting too much in forms of missed bullets. If the robot misses too many high energy bullets, it will be soon left very vulnerable to enemy attack with low energy. To choose the correct bullet power, I have used two different methods.
The first method involves looking at the distance between my tank and the enemy tank. As a general rule of thumb, if the enemy is far away, then it is better to use a smaller, fast moving bullet. When it closer, use a larger bullet for more damage. This is because when the enemy tank is far way so it will

take the bullet a longer amount of time to reach the tank. In this time, it is able to change the velocity making it easier to dodge the bullet. To reduce this amount of time, choosing a smaller bullet is better as the bullet is able to travel significantly faster. In the tests I have carried out, this has proved to be very effective instead of just using one sized bullet for all enemies.
The second kind heuristic I use to determine the bullet size is the number of times I have missed the enemy in a row. This is done to preserve energy by reducing energy loss in forms of missed bullets. If I have missed too many bullets in a row, then the size of the bullets is reduced significantly. This works on top of the first "distance" based heuristic. If everything is going well then only the first heuristic applies. If there are too many missed bullets in a row, then a number is returned which then gets subtracted from the bullet power given by the distance heuristic. Using both these methods, I have managed to create a successful bullet power method.

During the robot evaluation phase, it will be tested against many enemies at the same time and sometimes tested in one on one battlefields. For 1 vs 1 battles, radar lock is very useful as it allows my tank to track the enemy efficiently and keep it under my radar a lot of the time. There are quite a few radar locks which can be implemented. Out of those, I have found turn multiplier lock [2] to be the most effective in my case. This kind of lock involves pointing the radar at the enemy's last known location; resulting in a narrow beam which follows the enemy around the battlefield. This works most of the time and keeps the enemy under the radar. However, in some circumstances when my robot skips turns, the enemy is able to move outside the radar area. In this case, the radar just needs to be rotated fully in order to reacquire the lock. Even when the lock is lost, it is still better to implement this kind of lock instead of just repeatedly spinning the radar.

My robot generally is rule based incorporating some aspects of artificial intelligence heuristics to decide the best next move. When all the different parts of my robot comes together and works cooperatively, it acts as an artificially intelligent robot able to destroy many enemies.

References:

1. Matthew Bardoe (July, 2013) – Linear Targeting: https://www.youtube.com/watch?v=3KWYhhdlAUs
2. RoboCode Wiki – Turn multiplier lock: http://robowiki.net/wiki/One_on_One_Radar
3. RoboCode Wiki – Energy management: http://robowiki.net/wiki/Energy_Management