Roobot

Student ID: 1600779

ABSTRACT

Roobot was created with the aim of being unpredictable in a way which would provide direct benefits to both defense and offense. The agent's strategy consists of a risk function-based randomised scheme of movement supported by a rigorous targeting algorithm to hit targets moving linearly or in circles. Numerous techniques were considered but this one was deemed to be the most effective given the robot can only extend Robot and not AdvancedRobot. Roobot performs well in the majority of situations and has good survival rates thanks to its mobility and responsiveness to the distribution of opponents on the field.

1. INTRODUCTION

Roobot is a rule, knowledge and utility-based [14] artificial intelligence for a Robocode tank. It is adaptive to its surroundings and often acts based upon knowledge acquired through scanning rather than blindly on a strict set of rules; this allows it to be competitive against an assortment of opponent types and strategies. It has learning elements which are not persistent round by round but help it to judge the way in which a target moves after only a few scans. The risk function for movement is an example of utility based decision making.

Given the constraint of creating a class extending Robot versus one which extends AdvancedRobot, it seemed wise to not only rely on scanned information from opponents, but to also have an element of independence from scanned data to allow Roobot to be effective despite not being able to scan constantly like AdvancedRobots can. More advanced interpolation and prediction techniques were avoided in favour of an approach which would not be tricked into acting due to incomplete or inconsistent scanning data.

It was important for Roobot to move in long enough bursts that it does not sit still for too long waiting to fire (gun cooling enough during movement), but short enough that its scan data remains relevant whilst moving to the next position (moving great distances invalidates the risk utility value associated with the position as it will be out of date when Roobot reaches it due to movement of opponents etc.). This provided the basis of a set of challenges which the strategy outlined below attempts to overcome.

Coursework submitted for CS255 2018.
Department of Computer Science, The University of Warwick, CV4 7AL, UK.
Report compiled January 15, 2018

The produced agent is a fast moving and firing robot, especially when considering that it cannot do both at once; a robot which is able to move into space - or some other point of benefit - as well as kiting or strafing opponents thanks to the characteristics of the risk function. The agent behind Roobot's success attempts to learn from its enemies through storing information, and reacts to changes on the battlefield quickly as a result.

2. RELATED WORK

The Robowiki [5] was an obvious choice for information as Robocode is a community platform and it provides a good overview of some general strategies and fundamental rules to consider. Likewise, the sample robots which come with Robocode (all of which can be found on the Robowiki) offer a good resource to train against, as well as to highlight some comparisons between Roobot's strategy and more simple ones.

IBM's *Secrets From The Robocode Masters* [15] series provides some good strategic tips and the Anti-Gravity [7] and Circular Targeting tutorials [8] in particular were useful in developing Roobot's strategy.

The robot's movement is an adaptation of anti-gravity [6, 7, 2] and minimum risk [3] to be effective on a Robot rather than an AdvancedRobot. The *HawkOnFire* [13] robot served as a central inspiration throughout the development process; it provided a basis for Roobot's risk function and opponent data storage.

Roborumble and other documented competitions offered an immediate insight on what strategies were most effective across the board; clearly the longevity of the Robocode platform has led to an abundance of data regarding the evolution of tactics as new approaches rise and fall. It seemed that most of the top strategies [1] were hard coded rather than evolutionary which as stated in some research is not as effective as one might first imagine in most cases [11, 12].

It is this that prompted the decision for Roobot to be rule and knowledge based in the hope that it would be effective as an all-rounder, not built up to focus on countering one particular strategy. This was a mentality maintained throughout the project as there was only some very vague criteria for eventual external evaluation; Roobot was not to be confined to a niche.

Similarly, for this situation it seemed needlessly complex to store states between battles as only robots as simple as the sample robots would be consistent enough to safely judge their behaviour to a point where Roobot could pre-empt their actions, and this did not seem necessary given how easily Roobot beat them. It is an advantageous tactic for many advanced robots but incorrect assumptions and analysis due to anomalous occurrences may have done more harm than good to Roobot's performance given the limitations of the Robot class. There exists some research on dynamic decision making [10] in Robocode and this encouraged the flexible nature of Roobot.

3. STRATEGY DESIGN

Roobot's strategy can be broken down into three main ideas, all of which interact in order for the agent to be successful against an array of opponents in varying battlefield sizes. These environmental variables proved vital to consider whilst working on a strategy to ensure adaptability and overall success.

3.1 Knowledge Acquisition

The robot has four defined states based on its situation: scanning, fired, hitRobot and hitByBullet. The first of these states is its default; whenever Roobot is not executing movement or firing, it is scanning the field.

Almost every advanced and competitive robot utilises information gained from scanning to some extent; it was clear that this would be a key part of Roobot's strategy. The actual information stored grew as Roobot's functions did, and the data structure containing information on each opponent proved invaluable for movement and targeting.

Scanning another robot triggers the onScannedRobot() method which is where the bulk of Roobot's knowledge acquisition takes place. Roobot fills a data structure with an object of the opponentInfo class for each robot in the battle. Every time it scans a robot it either makes a new entry in the data structure if this robot has not been scanned before, or it updates the information associated with the robot if it is already present in the data structure.

The information pertains to each opponent's energy, location, heading and speed amongst other details. It is essential for Roobot to scan as much as possible so that this information remains relevant and can inform its decisions on how to behave. The basis of any knowledge-based agent requires good data, and it is also a necessity when considering the utility risk aspect of Roobot. The risk function can only be meaningful if the agent is aware of its environment.

3.2 Movement

The robot's movement methods are triggered by a number of events; namely, the three other self-explanatory situations (fired, hitRobot and hitByBullet) mentioned above, this results in nearly unpredictable movement due to the multitude of factors determining when and where the robot will decide to move. Whenever Roobot fires it is wise to assume another robot has probably fired at Roobot in its stationary position. Likewise, if Roobot is hit it should probably move to avoid being hit again.

Whenever Roobot's situation changes, i.e. it has been hit by something or fired a bullet, it breaks out of scanning mode and the pickNextPos() method is called. This method generates 100 points of varying distance and degree around Roobot's location, and feeds each one into a risk function eventually determining the point of lowest risk and setting it as Roobot's next position.

The risk function starts off by being set to the reciprocal of the sum of distances from Roobot's current and last positions to the proposed point. It then adds to the risk of that point for each opponent left alive on the field, calculating the risk to add from the ratio of Roobot's and the opponent's energy, the perpendicularity of the point to the opponent's location (these two parts define the risk function), and the inverse square distance from the opponent (this is the anti-gravity element).

These steps ensure Roobot does not stay in one place for too long (due to the initial risk applied to each point), as mobility is a key concept of its strategy. This along with picking from 100 randomly generated points means it would be almost impossible for another robot to predict Roobot's movements. On top of this, its movement is triggered by multiple different situations meaning it does not simply move and fire in a regular pattern, instead it depends on radar rotation, distance between movement positions and so on.

Additionally, it is worth noting that during a movement phase Roobot always accelerates to its maximum speed. Although robots such as painting and velocibot work well in their niche of oscillatory movement, the focus of Roobot is speed and it picks points regularly enough that linear targeting (along with most other methods) should not be effective against it. There was an idea during development to move in arcs of random radius intersecting two points, but this idea was scrapped in favour of a more direct and thus quicker approach which showed itself to be effective in practice.

Generating finite random points ensures atomicity and thus an element of randomness in movement, which is complimented by a tactically advantageous bias from the risk function. The factors considered when calculating risk mean that Roobot moves in a vaguely circular path through pockets of space on the battlefield, and often into locations which are at right angles to the majority of opponents.

Depending on the value of the situation variable, Roobot behaves differently when generating the points to choose from. If it has been hit by another robot, it will only generate points in narrow cones either side of the collision to try and avoid the robot which collided with it. Whilst if hit by a bullet it does not generate points along the line of the bullet's trajectory (in cones again but narrower cones of avoidance this time) so as to avoid a potential stream of bullets on the same path. Both restrictions make logical sense to implement, but were not present in other point generation algorithms as it is less critical for robot's which can continually scan to make distinctions between these situations.

Once the next position has been determined, Roobot moves to the point via a simple movement function similar to the standard GoTo method implemented by most robots. It does so in the quickest possible way, i.e. with minimum turning which is when the robot is at its most vulnerable.

This scheme of movement assisted by randomness seemed to be a better choice than the extremely popular wave-surfing algorithms [4], given Roobot would not be able to scan enough to have sufficient data to attempt to consciously dodge bullets. Instead Roobot relies on its unpredictability and risk function based movement to pre-emptively move into less dangerous pockets of space, which has the added bonuses associated with the risk function, e.g. biases towards weaker enemies and away from locations it has just been.

3.3 Targeting

Based on the information gathered through scanning, Roobot determines its target to be the closest robot to it (as accurately as its scanning allows for) as this robot is more intuitively likely to be a threat. This corroborates with the risk function above which assigns a higher risk to closer opponents. The process of targeting and firing is called from the onScannedRobot() method whenever the scanned opponent is equal to Roobot's current target.

It then utilises an iterative linear or circular targeting algorithm (based upon whether the target's change in heading is significant) to predict its target's future position and fire. This is done via iteratively predicting the target's location (starting from its current position) at a time, and then calculating a new time for its bullet to reach that position, in order to then predict how far the target will have moved in that time and so on. This results in very accurate targeting along a straight line or on a regular circular path but cannot really predict any movement beyond this.

Roobot only uses circular targeting on targets which consistently have a significant rate of change of heading. It determines this based upon the turningCountRatio() function which is the ratio of the number of times a robot is scanned with the number of times its heading change during a scan was significant (greater than 0.01 degrees per tick). headingChangeRate is calculated from the difference in heading with the previous scan of that opponent divided by the time between scans. It is very likely that a robot which has had little change in heading for the majority of scans is still moving linearly when its headingChangeRate is significant, it may have just encountered an obstacle and changed direction. The turningCountRatio() method allows Roobot to ignore these anomalies and only use circular targeting on robots which consistently turn in a circular fashion. This is another example of a careful adaptation that had to be made from the standard circular targeting algorithms to allow for Roobot's less consistent scanning ability.

Bullet power is determined based on Roobot's distance from the opponent. A lower bullet power results in less energy lost in the event of a bullet missing, something which lets down a lot of robots such as SpinBot which always fire full power bullets and often misses unless an opponent strays into the path of the bullet or the opponent is very close. Therefore, firing bullets of power inversely proportional to the distance from the target has two benefits: they travel more quickly over large distances and so are more likely to reach the target before its predicted position is invalidated by a change in direction etc., if it does miss (which it is clearly more likely from the reasoning above) it makes for less wasted energy.

The robot's gun only moves during the targeting phase, so sometimes it may be pointing in the complete wrong direction for the shot that is to be fired. As such, it was important to adapt the circular targeting method to take into account the time it will take to rotate the gun to point at the target's predicted position. Roobot makes a final check before firing, to see if the predicted position is in the battlefield; it is likely to miss if this is not the case and so it is not worth expending the energy.

4. EVALUATION METHODOLOGY

In order to determine whether Roobot achieved its goal of defensive and offensive effectiveness according to the specifics of the strategy above, it was vital to test the robot in numerous environments. Roobot was mainly tested against the sample robots, excluding interactive and sitting duck as they did not offer any actual competition (neither did a few of the others once Roobot got past its early stages but these were kept in for reference).

Battlefield size can have a huge effect on the performance of a robot; some such as RamFire have tactics which are obviously designed to be more effective the closer they are to their next target. Whilst testing on different dimensions was important, the importance of spawn position in a smaller battlefield surrounded by lots of robots could skew results; these tests are given less weight in judgement. Sometimes a robot may be immediately surrounded in a crowded melee and have no way to escape which is not particularly reflective of its effectiveness.

Important discoveries were made about Roobot when testing on a large scale (battles with thousands of rounds providing results for analysis), but also in smaller battles taken turn by turn to observe micro movements of the robot. It was important to get both perspectives especially when making tweaks to the robot's design after it was largely finalised. Generally, long battles were tested throughout the process to see how Roobot was doing statistically, compared to earlier versions of itself as well as a variety of opponents. Whereas watching specific movements and rounds often helped to iron out bugs in performance and code.

Roobot has a lot of elements to its design which require extreme precision for any definite optimisation to occur. With aspects such as the risk function and various ratios it was important to try and home in on an optimal value or formulation through making small tweaks alongside constant testing. Other changes such as the introduction of an adapted point generation algorithm for when Roobot is collided with or shot at were binary decisions. They could easily be implemented and contrasted with the previous version to see how well they performed. This evaluation process took place throughout development and helped decision making for almost every stage of Roobot's strategy.

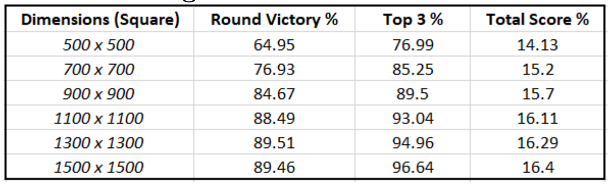
5. EVALUATION RESULTS AND DISCUSSION

The two tables below show Roobot's percentage performance across 10000 rounds of each situation. These testing situations consisted of one on one duels against each sample robot, as well as melee situations on fields of 6 different dimensions involving all of the sample robots at once. In terms of performance characteristics to be presented in the tables, the percentage of rounds won overall and the total share of the score generated in all the rounds was of most interest.

Note that although they are not present in the tables, values for bullet damage and survival scores remained in fairly equal proportion throughout the tests. Ramming damage was consistently minimal, especially relative to the other robots and Roobot's total score, this is testament to its movement being effective in dodging opponents.

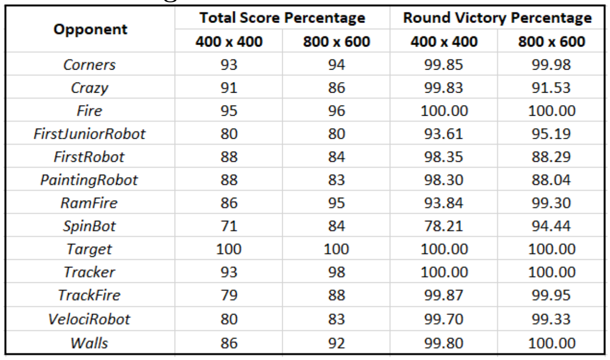
The robot clearly fares better as the map size increases in the majority of the situations. This is likely down to the agent's risk function being able to effectively determine areas of space on the field. As mentioned in the section above, there is more of a luck element involved regarding starting position on smaller fields especially in melee where Roobot could immediately be surrounded by robots and thus

Figure 1: Melee Results



Dimensions (Square)	Round Victory %	Top 3 %	Total Score %
500 x 500	64.95	76.99	14.13
700 x 700	76.93	85.25	15.2
900 x 900	84.67	89.5	15.7
1100 x 1100	88.49	93.04	16.11
1300 x 1300	89.51	94.96	16.29
1500 x 1500	89.46	96.64	16.4

Figure 2: 1v1 Duel Results



Opponent	Total Score Percentage		Round Victory Percentage	
	400 x 400	800 x 600	400 x 400	800 x 600
Corners	93	94	99.85	99.98
Crazy	91	86	99.83	91.53
Fire	95	96	100.00	100.00
FirstJuniorRobot	80	80	93.61	95.19
FirstRobot	88	84	98.35	88.29
PaintingRobot	88	83	98.30	88.04
RamFire	86	95	93.84	99.30
SpinBot	71	84	78.21	94.44
Target	100	100	100.00	100.00
Tracker	93	98	100.00	100.00
TrackFire	79	88	99.87	99.95
VelociRobot	80	83	99.70	99.33
Walls	86	92	99.80	100.00

be unlikely to do well.

One of the most immediate observations from the data and watching Roobot function is that there are some issues with circular targeting and the targeting methodology in general. These issues could not be entirely fixed in Roobot's implementation; so it is important to realise that it is usually Roobot's movement scheme that causes it to do so well against circular moving robot's such as SpinBot in duels rather than its targeting. Roobot was tested with and without the restriction of a turnCountRatio() (discussed in Targeting section) and it performed better with it meaning it would use circular targeting less often. All of this leads to the conclusion that the circular targeting element of this robot is not very effective and although it does sometimes hit correctly it is not one of the strongest components of Roobot's strategy.

The movement of Roobot is satisfyingly unpredictable whilst also usually being of strategic benefit. All the points generated for movement are in a range of 75 to 210 distance away. This was a range decided after numerous rounds of testing but anything around these values yielded similar results and Roobot often picks the furthest away point thanks to the first component of the risk function being instrumental to the eventual risk value in most cases. There were a lot of situations similar to this which require extreme precision to optimise fully. However, it seems reasonable to conclude given its performance that the agent ended up with good values and formulations, which corroborated with similar robots such as HawkOnFire who assumedly went through a similar development process.

Part of the risk function's purpose is to encourage movement in a perpendicular direction to Roobot's opponents, this was often hard to distinguish when observing the robot in action and perhaps outweighed by the bias against positions where Roobot has just been. Roobot's risk function assisted randomness worked well due to its unpredictability, but even better results were achieved through using strict rules for certain situations, e.g. when Roobot is in a collision or hit by a bullet only move in certain cones of possibility.

The robot performs better when some of the weaker robots (determined by score) are present in the battle. This can be attributed to the risk function's bias towards robots with a lower energy than Roobot, meaning it is more likely to be near weak robots and thus more likely to target them. This is assuming a weak robot will have lower energy for a greater proportion of the battles, which is a fair assumption given the survivability of more advanced robots.

Roobot seems to fare worst against robots which also move unpredictably. This makes sense as its targeting algorithm only allows for extrapolation of a relatively simple target's current movement pattern. Robots such as MyFirstJuniorRobot which move back and forward can be seen as stunted versions of Roobot and it seemed that these robots offered the greatest challenge.

One interesting thing that can be observed through watching Roobot in battle is that given the nature of the calls to Roobot's firing methods, it can overcome a limitation of the class to sometimes fire during movement if the radar is lined up correctly. This often occurs when Roobot is focussed on one target and can result in an impressive volley of bullets. It was hard to judge how effective stopping the robot from firing when the predicted position of the target at bullet impact is outside of the battlefield was in practice.

The evaluation process indicated that movement is the most important part of a robot's strategy. During development, the robot performed well before targeting was even properly implemented (winning over half of the rounds in melee and finishing top 3 in around three quarters of them on a 1000 by 1000 field). It is therefore reasonable to assume that most plausible improvements would focus on its targeting function, where it is hard to pick a single technique as each have their own specialities. It was assumed that most of the competitors to Roobot would move in a linear type fashion as well; moving fluidly is more characteristic of the AdvancedRobot class.

6. CONCLUSIONS AND FUTURE DEVELOPMENT

Roobot achieves what it set out to do in terms of all round performance, assuming the sample robots give a wide enough perspective of the possible tactics an agent might have to counter. Clearly though, testing against a wider array of opponents would test this hypothesis more rigorously. The evaluation process provided a lot of information on Roobot's performance and indicates the possibility and potential of some more advanced methods where it fell short. The knowledge, utility and learning aspects of the agent's design all worked well.

The limiting factor in all of these possibilities is the fact that the robot can only extend the Robot class; not being able to move and scan continuously is its biggest weakness. Some of the ideas outlined below were trialled and dropped as Roobot developed because they did not seem to have consistent effectiveness. In a lot of cases a change would benefit the agent in one situation and hinder it in another, making it hard to judge which approach to take.

Although genetic algorithms were decided against in the direct development of Roobot, developing specific robots to counter certain strategies (GA's main strength as discovered in other papers cited earlier) which Roobot struggled against could provide valuable insight on how to improve the agent to overcome its weaknesses; without the need for excessive experimentation. Observing robots which did well in the Roborumble leaderboards is what first inspired Roobot's design and thus it makes sense to also try and develop countering strategies for these robots. This could be achieved through the simultaneous development of many genetic evolutionary robots and picking the best characteristics from each to inspire elements of Roobot's design.

Given the knowledge acquisition part of Roobot's strategy seemed to be so successful (linear targeting and the risk function relied heavily on it and both worked well), it could be worth expanding on this functionality in future development. It was uncertain as to whether a more complex approach would be possible, but it could have perhaps held the answer to targeting oscillating and other less predictable robots which offered the biggest challenge for Roobot due to limitations of its targeting. If scan data could be more reliable it may be possible to keep track of an array of velocities, headings and positions for each opponent rather than just the most recent, and then determine a likely next move given the change in velocity over time or even what it did last time it was in a similar position.

One outcome of this data could be to try and predict future risk values of Roobot's potential next positions, by determining how long it will take Roobot to travel to that point and what the risk of the point might be at that time. This clearly leads to an increase in uncertainty but there is reason to attempt this given how effective iteration is in targeting algorithms, and there are certainly parallels to be drawn with this approach.

Another consequence of this approach could be the possibility for another heuristic to determine how likely a robot will actually move to the predicted position generated in the targeting functions of Roobot; this builds upon the concept of not wasting energy on shots that are aiming at points outside of the battlefield.

In terms of trying to improve scan data, radar movement could be made more intelligent, based on Roobot's own movement or reversing direction when a certain number of opponents had been scanned for example. The radar sweeping continuously to the right is certainly not the most offensively effective way to scan but it does provide a good balance in that it ensures Roobot does not move and fire too often without stopping to scan the field. It is likely that Roobot could intelligently predict which way to turn the radar to get to its target quicker once there are less robots on the field but any attempts during development did not seem to offer a good enough solution. Turning the radar right all the time at least ensures the whole battlefield is scanned fairly regularly.

Considering the complexity of the risk function and situation, a genetic search [9] to determine the most effective formulation would likely have given some improvement. It was, however, simply not convenient in terms of time or development especially given the diminishing returns nature of the problem. This was a problem faced throughout Roobot's implementation and is unavoidable given its sensitivity to the environment. Optimal values could eventually be found but it is debatable whether they would have any real observable impact on the agent's performance.

REFERENCES

[1] Roborumble leaderboard. `http://literumble.appspot.com/`.

[2] Robowiki: Anti-Gravity Movement. `goo.gl/LsH1Uk`.

[3] Robowiki: Minimum Risk Movement. `goo.gl/atBt31`.

[4] Robowiki: Wave-Surfing. `goo.gl/MiJ6uE`.

[5] Various Contributors. Robowiki. `http://robowiki.net/`.

[6] Alexandrescu, Andrei and Miller, Jessica. SamuraiBot: A Ruthless Neurofighter. 2003.

[7] Alisdair Owens. Anti-Gravity Movement Discussion and Tutorial. `https://www.ibm.com/developerworks/library/j-antigrav/index.html`.

[8] Alisdair Owens. Circular Targeting Discussion and Tutorial. `https://www.ibm.com/developerworks/library/j-circular/index.html`.

[9] D. E. Goldberg and J. Richardson. Genetic algorithms with sharing for multimodal function optimization. In *Genetic Algorithms And Their Applications*, pages 41–49, 1987.

[10] M. Kim, H. Jeong, and E. Lee. Dynamic decision making for self-adaptive systems considering environment information. *Journal of KIISE*, 43(7):801–811, 2016.

[11] Liang, Richard and Zhao, Peng. Applying and Comparing Evolutionary Algorithms for Robot Tanks.

[12] Nielsen, J and Jensen, B. Modern AI For Games: RoboCode. 2011.

[13] Rozu. HawkOnFire Robot Design. `http://robowiki.net/wiki/HawkOnFire`.

[14] Russell, Stuart and Norvig, Peter. Knowledge Based Agents. In *Artificial Intelligence: A Modern Approach*, pages 235–236. Prentice Hall.

[15] Various Authors. Secrets from the Robocode masters. `https://www.ibm.com/developerworks/library/j-robotips/index.html`, 2002.