CS255 Artificial Intelligence: Strategy Design Report
Alex Macpherson – u1409675

Tasked with the design and implementation of a robot tank, employing a number of artificial intelligence techniques to control its behaviour, a number of different strategies and designs had to be considered. With the goal being a well-rounded bot which performed well in a number of different environments and scenarios, changes and improvements often resulted in performance compromises in certain areas. This report will cover not only the justifications for design choices of the final behaviours defined, but also a selection of different strategies researched, attempted and improved upon through the course of the project, most notably covering the facets of various movement and targeting techniques.

The Robocode environment is a stochastic, partially observable state space. Though any given robot (the agents of this environment) knows its own state, the states of other agents and the rest of the environment is limited to what can be scanned by the radar in a single turn. Moreover, every agent's behaviour patterns and rationality, defined by its coding, differs from the next, making the environment as a whole particularly unpredictable. In order to build a reasonably rational agent, the defined behaviours must be dynamic, changing according to changes in the environment. With this in mind, it is also considered a necessity to scan the environment as often as possible, allowing for the agent to store a relatively recent image of the state space, from which actions are determined and, in an ideal scenario, learned from and improved upon. Though basic robot implementations often fall under the simple-reflex or reflex-with-state agent categories, generally the best are learning agents, capable of assessing and modifying behaviour patterns based on the successes and failures of previous actions and consequent states.

Wilde, the final iteration robot for this project, is a learning agent which employs 'guess factor' targeting[1], tracking and learning which potential firing angles are most likely to hit an enemy of a number of different states, including their distance and lateral direction, based on previously successful hits. Successful angles are cached and can be stored between rounds, meaning the more *Wilde* faces an opponent, the more accurate it will become; this can result in markedly poor performance in early rounds of a battle, but this quickly improves over time as it learns its opponents.

In order to scan both the environment and its target, *Wilde* uses a 'weak target lock', attempting to remain focused on its target in order to provide incredibly up-to-date information required for targeting. The radar does not aggressively pursue the target's position however, meaning the target is often somewhat-deliberately lost, regularly forcing a scan of the entire battlefield to provide reasonable information on which to base risk calculations for the movement as well as to find closer, more-viable targets.

In terms of its movement, *Wilde* features two distinct behaviours, choosing between them based on the number of enemies it currently faces. In a melee (a free-for-all battle between multiple robots) scenario, a utility-based 'minimum risk'[2] strategy is used. This generates 100 potential points at a random distance within a given range around the current

position, each of which is assessed according to the risk incurred by moving there, based largely on the proximity and current energy values of enemies. The robot will then move to the point of least risk, essentially moving as far away from enemies as possible in order to make it a less likely target choice as well as being at a reasonable distance to avoid being caught in accidental crossfire. In one-vs-one combat, however, a pseudo-random movement method is employed, attempting to remain as perpendicular as possible to the absolute bearing (enemy bearing relative to the robot's current heading) of the enemy's current position to allow a maximal escape angle in order to dodge any incoming fire. The movement itself begins relatively linear, prioritising 'perpendicularity', though it becomes increasingly random the more it is hit to hinder the accuracy of enemy targeting algorithms.

Tasked with defining behaviours capable of withstanding a host of different scenarios and environments, *Wilde* is a fairly versatile robot, being based on a number of well-established and proven strategies. This was attained from the outset, as, before diving into the building of a new robot, it was deemed best to explore the Robowiki strategy guides[3, 4] and tutorials, granting a fair understanding of the advantages and compromises surrounding a number of unique implementations, highlighting complementing and versatile strategies. This helped considerably by defining a path of development prior to any hard-coding, with a clear idea of how to improve the robot between iterations. From this point, it was easier to study and understand existing robots[5], ranging from the simplest of sample bots to far more advanced and well-established agents – the latter often being highly specialised in certain fields – analysing the performance of varied strategies in action. For example, *Rozu's HawkOnFire*[6] bot was built predominantly for melee, performing well in this scenario due to its excellent risk-based movement algorithm but being outperformed in one-vs-one combat by bots more specialised in that area due to its simple head-on targeting technique. What was perhaps most notable from these studies was how well the *Sample.Walls* bot performs, in spite of its simplicity, against low-to-mid level robots – often of a far higher behavioural complexity. This is likely due to its constant movement on the outer-most perimeter of the battlefield, making it an unlikely target in melee battles, where most robots attempt to track and fire at the enemy nearest to them, albeit not a difficult target to hit with a more advanced targeting algorithm due to its incredibly linear movement patterns.

With this in mind, *Wilde*'s initial movement patterns were based on those of *Sample.Walls*, though an element of randomness was introduced to break the linearity of movement along the walls, travelling a random distance within a given range before deciding whether to reverse the lateral direction. It was soon decided to actually crawl a small distance away from the wall, lowering the likelihood of another wall-crawling robot to score many successive hits by firing parallel to the wall once it reached a corner, quickly resulting in an often-inescapable demise. This albeit-small degree of wall-avoidance carried through all further iterations of movement, notably with later point generating functions deeming a point too close to the wall as invalid for this reason.

A later iteration of *Wilde*'s movement introduced an 'anti-gravity' based calculation[7] to the direction-changing decision, attempting to move away from enemies along the wall.

This calculation used the inverse-square law of gravity to generate a repulsion force away from all enemies, where closer enemies exhibited a stronger repulsion than those further away, pushing the robot as far away as possible so as to become a less-likely target in melee while still employing some degree of random movement to hinder targeting. Though these were effective at mitigating against linear –and other, more complex– targeting algorithms, both the simple randomness and 'anti-gravity' repulsion had a tendency to hover around a single point on the battlefield, not only enabling guess-factor-based targeting to quickly minimise the possible firing angles in order to score a hit, increasing enemy accuracy, but also proving a more frequent victim of head-on targeting.

By repurposing the enemy-repulsion calculation from the 'anti-gravity' function, a framework for the final risk-based movement system was created. Though minimum-risk movement appears relatively similar to 'anti-gravity', it differs in that the latter calculates an angle to move according to the forces acting on the robot's current location, where the former need only assess the cumulative strength of all forces acting on a given point rather than calculating an angle, making it mathematically far simpler, achieving the same effect through brute-force assessment of numerous randomly generated points. Effective risk-based movement, however, is not assessed solely on enemy locations, incorporating a number of factors in the function to adequately determine the risk at a given point. Notably, for each enemy, their energy level is considered alongside their distance; a constant is added to this value to ensure that low-energy targets are still considered a viable threat by the function. Moreover, a weakly repulsive force is added to both the current and last locations of the robot, ensuring some degree of movement from the current location to reduce the ease of targeting incurred by gravitating to a single point deemed 'safe' by the calculation.

The risk function *Wilde* employs is incredibly simple yet notably effective at sustaining low-risk movement around the battlefield, reducing the chance of being targeted and hit by enemies and increasing survivability. However, additional factors could be considered to further improve the risk calculation, with some implementations attempting to discern which enemies they are most likely being targeting by, assigning these a far greater risk value and, in some cases, attempting to move perpendicularly to the enemy's absolute bearing – a technique more popular in one-vs-one combat – allowing for a greater escape angle to better avoid incoming fire. Moreover, though the simplicity of *Wilde*'s risk function is effective in melee, it falters greatly in one-vs-one, often attempting to move laterally away from the enemy, affording very poor escape angles and easy targeting by all methods, most notably head-on targeting. For this reason, *Wilde* changes its movement strategy dynamically once only one enemy remains to one which attempts to remain perpendicular to the enemy's absolute bearing. Initially, this function assumes the enemy to be using head-on targeting, trying to maintain a constant lateral direction unless a point in the desired direction does not exist. By maintaining a relative direction of motion, head-on targeting becomes near-futile; the same is not true for more complex targeting algorithms, against which, the chance of being hit increases significantly. To mitigate against this, a degree of randomness is introduced to the movement according to the number of hits received – that is to say, the more the robot is hit, the more likely it is to act randomly. True randomness of direction often results in hovering around a single point, as previously noted,

increasing the likelihood of hits landing from enemies using head-on or guess-based targeting, so this pseudo-random melding of techniques provides a reasonable level of survivability against a number of enemy targeting strategies.

In terms of targeting, it was always the aim to build as robust and versatile behaviour as possible for *Wilde*, with a number of strategies explored. Head-on targeting is the simplest form imaginable, firing directly at the enemy's current location. Though this may seem infeasible as a tactic for hitting moving targets, the sample bots clearly demonstrate how practical it can be – sometimes even out-performing far more complicated targeting algorithms. This is predominant against enemies which attempt to gravitate around a single point, where strategies like circular or linear targeting will attempt to fire according to the enemy's current path. Head-on targeting is, however, largely useless against robots with reasonable movement strategies, even though the odd hit may be scored here-and-there, and, as such, was only featured in *Wilde*'s most basic iterations. In this sense, head-on targeting highlights the absolute necessity for reasonable movement algorithms.

Linear targeting was the first dedicated targeting strategy utilised and, though only a little simpler than head-on targeting, it performed fairly well. Calculating the optimal firing angle according to the target's current heading, velocity and distance, linear targeting is near-guaranteed to hit any enemy which maintains its heading and velocity. With respect to its simplicity and limited scope, it is excellent – unbeatable, even – in very specific situations, but becomes increasingly infeasible with the complexity of enemy movement, with targets rarely maintaining a single heading long enough to be hit. It can be improved, though, not only by selecting a bullet power relative to the target's distance – as lower-power bullets travel faster, they may be more likely to hit a far-away enemy before it changes direction than a slower, higher-powered one – but also by accounting for other factors, such as the changes in heading of an enemy over time, to allow for more accurate targeting of enemies which may not be traveling in a straight line, a technique known as circular targeting.

Circular targeting is one of the most basic learning algorithms a robot can employ, extending the already reasonable capabilities of linear targeting to more successfully hit targets which do not maintain a constant heading. Sharing much of its code with linear targeting, it is a reasonably versatile strategy, performing well across most scenarios, though it also shares a number of its flaws, with complex and random movement often significantly lowering the number of successful hits.

With complex movement behaviours in mind, the requirement of a more complex learning algorithm for targeting becomes ever more apparent, and it is for this reason that *Wilde* features an implementation of 'guess-factor' targeting. A guess-factor[8] reflects the possible positions an enemy can reach according to its current relative direction, maximum escape angle and potential velocity, each being numerically assigned a value between 1 and -1, where 0 represents its current position. When targeting, *Wilde* bases its firing angle on previously successful guess-factors – that is to say that the history of an enemy's states can guess its next states with increasingly reasonable accuracy the more historic data it has. It is important to note that this is a learning algorithm which improves significantly over time.

Segmentation[9] increases the overall guess-factor accuracy by more discretely defining possible enemy states, allowing guesses to be made according to far more specific situations. *Wilde* segments its data in a number of ways; most notably, the distance of the enemy from the current location. This helps to overcome the main issues linear and circular targeting have, with the enemy having changed its course by the time the fired bullet reaches their projected position, by recognising this may be the case and guessing based on how the enemy has moved in the time it takes the bullet to travel this distance historically. Though the guesses are often less accurate at further distances due to the increased unpredictability of enemy behaviour further into the future, they are still a marked improvement. Other segmentations include distinguishing between one-vs-one and melee behaviours, which often change dynamically, and the lateral direction of the enemy – whether they are moving parallel to, towards or away from the current location *(if at all!)*, with potential angles changing significantly according to this information. Enemy robots of the same type also share their data, affording swifter learning of behaviours where possible.

With so many possible states and segmentations, it would be infeasible to learn which guess-factors are most successful for an enemy of a given state simply by noting actual hits, so a virtual bullet wave is fired every time an enemy is scanned, calculating and caching successful guess-factors were a bullet to be fired at that point in time. However, due to the inconsistency of scan intervals as a result of the weak locking of *Wilde*'s radar, virtual wave data, particularly in melee battles, was often lost or returned incorrectly, corrupting cached information with sometimes-wildly incorrect guesses. As this more notably became an issue, linear interpolation was added to the virtual wave function, allowing a reasonably accurate assumption of the correct guess-factors to be made.

Though initially guess-factor targeting resembles little more than head-on targeting in practice, it has the capability to both assimilate and surpass most other targeting algorithms as it learns enemy patterns through successful guesses based on previous behaviour, affording it a very reasonable success rate across most movement patterns, where the majority of other targeting excel against a single movement form but perform poorly otherwise. It is far from perfect, however, and will take significantly longer to generate accurate guesses for complex enemy movement patterns, but is, on the whole, a far more versatile algorithm than other more-basic targeting implementations, suitable for both melee and one-vs-one.

Wilde employs a number of other basic survivability techniques in order to improve its performance, largely in melee battles. These include only firing at the current target, for firing at an enemy for which there exists insufficient data will not only likely be a wasted shot, but also a waste of energy which may need to be conserved, and never firing a shot if it will disable itself, attempting to outlive opponents even if it can't directly beat them, amongst others.

In closing, *Wilde* is far from a perfect robot, and a number of areas for potential improvement exist – from improved risk calculation for movement to increased scan regularity, improving the accuracy of interpolated guess-factors. One of the more prominent

areas for improvement is the implementation of a function to accurately determine where the enemy will be when a fired bullet hits, allowing the prevention of firing wasted shots at the wall if it is clear the enemy cannot reach that point. This would help to save a considerable amount of energy wasted by the guess-factor targeting algorithm, significantly improving survivability. In spite of this, *Wilde* is still an effective and viable bot, behaving rationally to the best of its ability and performing reasonably well across both melee and one-vs-one battles, and though it is often beaten by far more specialised bots in a given scenario, it more than makes up for this in its versatility.

References:

1. Robowiki: GuessFactor Targeting Tutorial
   <http://robowiki.net/wiki/GuessFactor_Targeting_Tutorial>
2. Robowiki: Minimum Risk Movement
   <http://robowiki.net/wiki/Minimum_Risk_Movement>
3. Robowiki: Melee Strategy Guide
   <http://robowiki.net/wiki/Melee_Strategy>
4. Robowiki: Robocode FAQ
   <http://robowiki.net/wiki/Robocode/FAQ>
5. Robowiki: Understanding Coriantumr
   <http://robowiki.net/wiki/Melee_Strategy/Understanding_Coriantumr>
6. Robowiki: HawkOnFire – Rezu
   <http://robowiki.net/wiki/HawkOnFire>
7. Robowiki: Anti-Gravity Movement Tutorial
   <http://robowiki.net/wiki/Anti-Gravity_Tutorial>
8. Robowiki: GuessFactor
   <http://robowiki.net/wiki/GuessFactors>
9. Robowiki: Segmentation
   <http://robowiki.net/wiki/Segmentation>

Further references listed in Wilde.java.