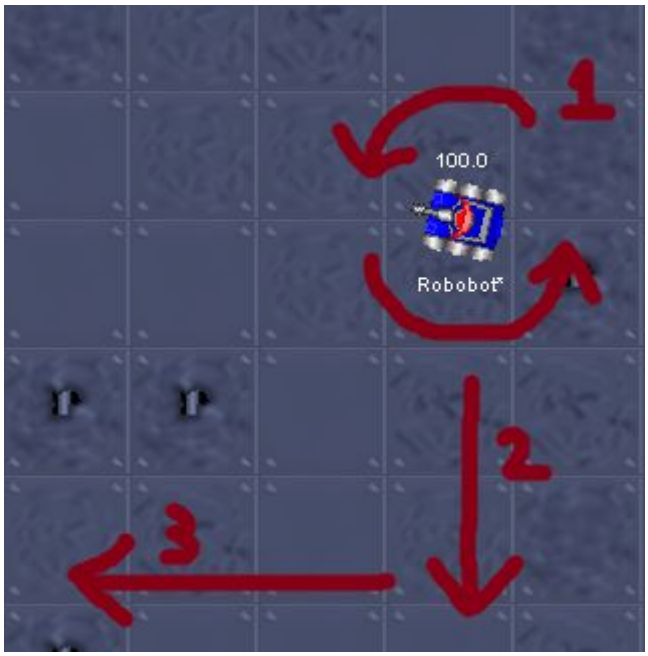
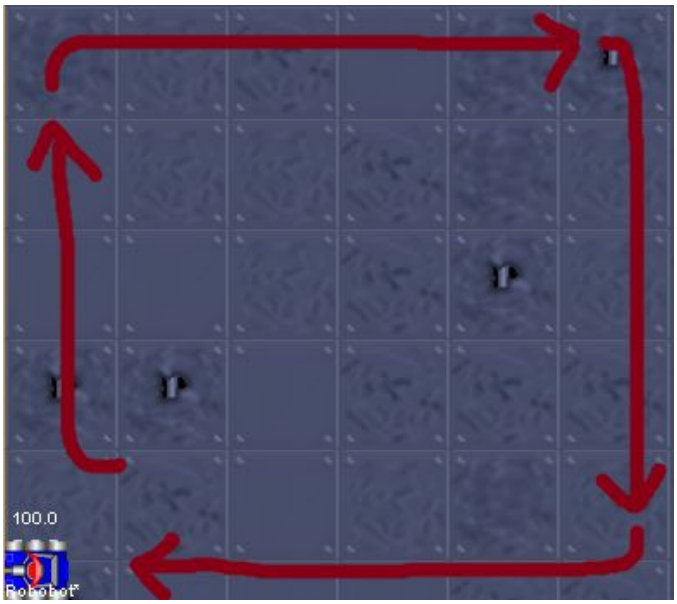
Robocode Follow-up

Our robot “Robobot” is a simple robot that fights by navigating around the sides of the battlefield. At the beginning of every match, it would find its way to the corner of the map as soon as possible, and then procede to move clockwise along the perimeter for the rest of the battle. Both the turret and the radar would constantly be turned 90 degrees, to always face to the right of the direction the robot is driving, and towards the inside of the battlefield. When the radar scanned an enemy, the gun would shoot at that position. Since most of our robot’s shots would be taken at range because of its positioning, we decided to make most of its shots be of power 1. We did this to limit the amount of energy the robot wasted on a missed shot, as the chance of hitting at such ranges would be low. However, the power of the shot would scale with how close the enemy robots were, where if the distance between them was low enough, our robot would shoot the most powerful bullet the game’s rules allowed. With the design we employed, our robot often seemed to last a long time, but not do too much damage to other robots. Because of its movement, the bot would stay out of the fray for most of the time and avoided taking a lot of damage. This, combined with its tendency to not waste a lot of energy on its shots, led to it staying alive for a long time, and was was enough for it to win a lot of the preliminary rounds. It was in the later rounds that this style of play began to not help much, and our robot would be quickly destroyed by bots with more advanced shooting algorithms.



At the start of every round, the robot would find its way to the bottom right corner of the battlefield using a few specific directions. If its x-coordinate (getX()) was not equal to the width of the battlefield (getBattleFieldWidth()), it would run the method alignRobottoDirection(double direction), which would go through a few calculations to turn the robot towards the direction in the argument, in this case 270 degrees (downward). This method was useful throughout the entire programming process for any task that involved using objective directions as opposed to turning just relative to the robot’s current direction. The robot was then told to move forward the exact amount of units needed to reach the bottom of the battlefield, or getBattleFieldWidth()-getX(). After the robot moves to the bottom of the battlefield, a similar maneuver is done to get it to the left edge of the battlefield using alignRobottoDirection(180), getY(), and getBattleFieldHeight(). The image above shows how the robot maneuvers itself at the start of the round to get itself to the corner.



Now that the robot was perfectly positioned in the bottom left corner of the battlefield, it was ready to start going through its basic movement, which it did through the navigateSides() method. This method is responsible for the clockwise movement of the robot around the perimeter of the battlefield, and it was to be called to loop through for the remainder of the battle. It would first turn the robot right 90 degrees using the turnRight(double degrees) method. Then it would use a method called moveGuntoDirection(double direction), which would position the gun to an absolute direction, similar to the method doing the same for the robot itself mentioned earlier. The argument entered here is getHeading()+90, so that the gun would always be facing 90 degrees to the right of the robot and towards the inside of the battlefield. Finally, using two if-statements, the robot would move to the next corner of the battlefield based on its distance from the boundaries, again using getX(), getBattleFieldWidth(), and getBattleFieldHeight. This method called back to back would cause the robot to do its clockwise, edge-skirting movement that we intended, as shown in the image.

The shooting algorithm for our robot was fairly straightforward. The gun is locked in place with the radar using setAdjustRadarForGunTurn(false), so the radar would be turned along with the gun every time the gun rotated. The onScannedRobot(scannedRobotEvent e) method would simply shoot where the gun was pointing, the power depending on how far away the enemy robot was. It would use e.getDistance() to determine if the enemy was less than 100, between 100 and 200, or more than 200 units away. If it was less than 100, the robot would shoot a bullet of RULES.MAX_POWER. Between 100 and 200, RULES.MAX_POWER / 2, and if more than 200, power 1.

Anyone who would like to have this robot to use or test for themselves can do so by following the following instructions:

1. Download and unzip the file from github: https://github.com/lukedoukakis/robocode-autobots
2. Copy the baron_doukakis folder and paste it in your robots folder (robocode>robots)
3. Open Robocode
4. Go to Robot Editor and open Robobot, found in the baron_doukakis folder
5. Compile the robot and gear up for battle!