Team MrRobot: Jay Chen, Tyler Stamp
Robocode Report
CS 141

The basic concept for MrRobot is that it tries to move toward and attack the robot with the lowest health on the battlefield while avoiding the robots with high health.

MrRobot keeps track of the information of other robots on the field using a Hashtable<String, Enemy> that stores the enemy robots' names as the key and an Enemy object corresponding to that name as a value. The reason we used a Hashtable is because it is a more efficient structure, and we wanted to avoid looping through an ArrayList or the like in order to access and update the current scanned enemy's information.

The Enemy object was a class we created within the main MrRobot class, in order to facilitate access to its functions and avoid having multiple Java files to reference when writing the code. The Enemy class contains instance variables for the last known location of an enemy, as well as its current energy.

In order to store the enemy's last scanned location in the instance variable Hashtable<String, Enemy> enemies, the location of the enemy was calculated in the onScannedRobot() method using trigonometry. The x and y inputs were calculated using the following code, and stored in a Point2D.Double, which became the location value in the Enemy object.

```
double enemyX = (location.getX() + Math.sin(getHeadingRadians() +
e.getBearingRadians()) * e.getDistance()); //e is a ScannedRobotEvent
double enemyY = (location.getY() + Math.cos(getHeadingRadians() +
e.getBearingRadians()) * e.getDistance()); //e is a ScannedRobotEvent
```

MrRobot uses risk based movement to determine when to move and when to avoid walls or other robots. An attempt to throw off prediction based trackers was made in that the robot will not move until hit, as it is in a "safe space" until then. However, there was a bug in the code that did not assign a target to the robot until a robot on the field died. The Enemy target variable was used in order to determine which robot to move toward and implement the risk movement strategy, and without an assigned value MrRobot was a sitting duck until it or something else died first. This bug occurred because there was no assignment of a target in the onScannedRobot() method, only the onRobotDeath() method where the target variable updates based on if the current target died and the Hashtable enemies removes that robot from its roster. It has since been fixed.

The risk based movement strategy is implemented in the risk() method and carried out in the main run() method. The risk() method takes in a possible new location to move to and has 3 main cases: (1) when the robot is at low health and the point passed is near a wall, the risk is very high to denote that it should not move there (2) if the location passed is the location of the target enemy, the risk is 0 because we want to keep moving towards the target (3) any other possible location being assessed is calculated using the following formula:

```
riskVal *= hp*location.distance(closest.getLoc());
```

Where hp refers to the closest robot's hp, location is the current location of MrRobot, and closest is the same closest robot to MrRobot's location. In this way, when the closest robot is very close and its hp is very high, the point is high risk, while when the closest robot has a low hp and is close, the risk is low. Risk is also higher the farther away a robot is, as MrRobot would need to move farther to reach that point. The initial point passed to the method is compared against the values of the closest robot, and if there are closer robots that point is foregone in favor of calculating the risk based on the closest enemy.

Team MrRobot: Jay Chen, Tyler Stamp
Robocode Report
CS 141

In the run() method, the risk based movement starts by predicting a point within an area the size of the battlefield width and height -50, to ensure that the robot will not be moving too close to the walls. The point is then compared to the next location for the robot, and if the point is less risky as calculated by the risk method, the robot will move to that point, which is now defined as the next point for the robot. Otherwise, the robot will continue turning to find a better place to move, or keep moving toward its target. In every case, the instance field storiing the Point2D.Double next is key to movement, and it is adjusted based on when a robot hits a wall, scans a new low risk point, or when the robot hits another robot.

The onScannedRobot() method deals with the scanner and gunfire. The power of each bullet was set at a flat damage of 3 if it scanned an enemy robot. This overrides the damage of 1 set in the run() method only if an enemy robot is scanned. Since the robot will only shoot at an enemy that is currently scanned, the damage set in the run() method is not needed and was a small mistake on our part. The enemyX and enemyY inputs used in calculating the enemy's location are also used in determining the scanner's target. The enemyLoc variable stores the x and y values of the target's location, which then gets stored in a variable en with the robot's energy:

```
Point2D.Double enemyLoc = new Point2D.Double(enemyX, enemyY);
Enemy en = new Enemy(enemyLoc, e.getEnergy());
```

The scanner targets weaker opponents which is why MrRobot was seen switching targets often. The first if statement in the method shows how the robot switches between these targets:

```
 if(target==null || target.getE()>en.getE())
      target = en;
```

If MrRobot does not have a target or the current target is not the weakest enemy on the field, then it switches to the weaker target. The setTurnGunRightRadians() method from the Robocode package was used to turn the gun so MrRobot would be facing the targeted opponent at all times. A variable, gunTurnAmt, was used to calculate exactly how many radians the gun would need to turn:

```
double gunTurnAmt;
if (e.getDistance() > 150)
      gunTurnAmt =
robocode.util.Utils.normalRelativeAngle(absBearing-
      getGunHeadingRadians()+latVel/22);
      setTurnGunRightRadians(gunTurnAmt);
else
      gunTurnAmt =
robocode.util.Utils.normalRelativeAngle(absBearing-
      getGunHeadingRadians()+latVel/15);
      setTurnGunRightRadians(gunTurnAmt);
```

Since the movement of enemy robots is unpredictable, the value for gunTurnAmt had to be a changing value that can keep up with dynamic movement.

The complete code for MrRobot is located on Github:
`https://github.com/jaykafkachen/CS141-Robocode-TeamMrRobot`
This code is meant to be used in the Robocode desktop application, which is available online:
`https://sourceforge.net/projects/robocode/`

Team MrRobot: Jay Chen, Tyler Stamp
Robocode Report
CS 141

This robot is categorized as an “advanced robot” and is best used against the default robots in Robocode. This robot can handle any default robots, “nano bots,” or any robot that extends the “robot” class as opposed to the advanced robot class. This robot has only been tested in one unofficial competition so it is not ready for any tournaments. It is also untested against many robots that extend the advanced robot class, despite being a robot that does the same.

To use this robot for yourself, copy the code from Github and open Robocode. In Robocode, click the “Robot” tab at the top and click “Source Editor.” The robot editor will open, and you will need to click “File” at the top. Click “New” and go to “Java File.” Paste the code in the editor and click “compile” under the Compiler option at the top. If you completed every step, MrRobot should be ready for testing.