Lam Lieu
Juan Vera
PolyBot
Dr. Husain

Robocode Final Report

At the start of the project, we brainstormed some ideas that we wanted to implement into our robot. Some of these ideas include, conserving energy/gun heat, avoiding the walls and the center, move to fixed points based on the coordinates of the robot, and to switch modes based on the number of robots left. From there, PolyBot was born.

Throughout the entire round, our robot's gun is turning constantly. To conserve both energy and gun heat from bullets shot, we created a formula that fires a bullet with the power depending on how far the scanned robot is(amount = 800/e.getDistance()). If the robot is close, we shoot a big bullet with power 3, and if the robot is far, we shoot a small bullet with <= 1 power. This method increased our gun accuracy since the speed of the bullet is relative to the power, and it also conserved our energy and gave us more points overall during tests.

For the different robot modes, we decided to cut the round into 3 phases (beginning, middle, end). If there are 2 or less robots, it is the end phase, if there are half or less than half of the robots left, but more than 2, it is the middle phase, and if there are more than half of the robots left, it is the beginning phase.

To avoid walls, we created a method called checkWalls(). In this method, we compared the coordinates of the robot to the boundaries that we set. One example of a boundary that we set for the robot is shown here:

1. getX() >= getBattleFieldWidth() / 10
    a. In this conditional, we compared the x coordinate of the robot to one-tenths of the battlefield. This covers the left side of the battlefield.

If this conditional is true, then the robot will turn right 55 degrees for every 2 turns.

For the movement in the beginning, we wanted the robot to move in a diamond pattern. The movement of the beginning phase starts off by comparing the X and Y coordinates to the vertices of the diamond to determine which vertex that we want the coordinate to go to. One vertex that we used has the coordinates:
X = getBattleFieldWidth() / 20 and Y = getBattleFieldHeight() / 2. This coordinate is on the middle of the right side of the battlefield. Next, the method checks if the robot is on one of the vertices. If it is, it will change the target location to the next vertex that we want the robot to go to. If not, then the robot will continue to target the closest vertex.

For the second phase, we wanted to have the robot move in circles (while avoiding walls) and to shoot at whoever was shooting at us. However, the latter caused too many problems and wasn't efficient, so we stuck to just having the robot move in circles, and since

there are fewer robots left, we made the robot shoot only if they're at a certain range in order to not miss a lot of bullets.

The last phase was made to focus on the last two targets on the battlefield. Figuring out the calculations was tough, but we managed to create a system that follows the target and shoots them down 1 by 1. This system proved to be very accurate and consistent.

Once we finished with the last phase, there was about half a week left until submission. Sadly, we had a problem with the PolyBot.java file and half of the content got mixed and deleted due to a saving error. Nevertheless, it was the only bump we came across, and we were back up to speed in a couple of hours.

After testing the robot and reading up more on how robocode works, we found that robots get extra points for ramming other robots and also killing them by it. Because of this, we made the robot run such that if it hits another robot, it turns to their direction and continuously rams and shoots them until they die or until we hit another robot (*see onHitRobot(HitRobotEvent e)*). This gave us a huge increase in the score, and got us first place on nearly every test against sample bots. However, our tracking in the last phase became buggy because it would end up losing sight of the robot and doing a whole 360° turn before shooting/following again, and we couldn't figure out why or find a fix to it.