プログラミングII

Report 1

提出日：　２００９年１１月９日

所属：　情報工学部一年次

学籍番号：　０９５７４５Ｄ

氏名：　西島本　純

Corner.java

```
package sample;

import robocode.DeathEvent;
import robocode.Robot;
import robocode.ScannedRobotEvent;
import static robocode.util.Utils.normalRelativeAngleDegrees;

import java.awt.*;

public class Corners extends Robot {
	int others; // int 型の変数。他の機体の数を保存する

	static int corner = 0; // static なのでラウンドが変わっても値は残る。
	boolean stopWhenSeeRobot = false; // boolean は真偽値を表す型

	public void run() {
		// 色の指定
		setBodyColor(Color.red);
		setGunColor(Color.black);
		setRadarColor(Color.yellow);
		setBulletColor(Color.green);
		setScanColor(Color.green);

		// getOthers()で残っている敵の数を調べて、others に代入
		others = getOthers();

		// 下記の goConer で説明
		goCorner();

		// gunIncrement に３を代入
		int gunIncrement = 3;

		// 永久ループ
		while (true) {
			//for 文の中で砲台を３度回転させるのを３０回行っている
			for (int i = 0; i < 30; i++) {
				turnGunLeft(gunIncrement);
			}
			//９０度見回したとに gunIncrement に－１をかけて
			//反対方向にもう一度９０度見回している
			//この９０度に敵が入らない限り砲撃することがない
			gunIncrement *= -1;
		}
```

```
}

public void goCorner() {
	// stopWhenSeeRobot を false を代入
	stopWhenSeeRobot = false;
	// turn to face the wall to the "right" of our desired corner.
	turnRight(normalRelativeAngleDegrees(corner - getHeading()));
	// stopWhenSeeRobot を true を代入
	stopWhenSeeRobot = true;
	// 下はすべて角に向かう命令になっている
	//前進し壁に向かい、左を向き前進し角に向かっている
	//その後砲台の向きを調整し、角から全部のエリアを
	//スキャンできるように向けている。
	ahead(5000);
	turnLeft(90);
	ahead(5000);
	turnGunLeft(90);
}

public void onScannedRobot(ScannedRobotEvent e) {
	// Should we stop, or just fire?
	if (stopWhenSeeRobot) {
		// 動作をすべて停止します
		stop();
		// e.getDistance()で敵との距離を測定する
		//smartFire は下記に詳細を記す
		smartFire(e.getDistance());
		// レーダーで他の敵を捜す
		scan();
		// resume()で stop()から復帰する
		resume();
	} else {
		//ゲームが始まってすぐの壁に向くまでの攻撃行動
		//制御である。目的の壁を向くことが敵を見つけて
		//集中攻撃するより優先されている。
		smartFire(e.getDistance());
	}
}

public void smartFire(double robotDistance) {
	//攻撃の火力を距離と自機のエネルギーで制御している
	if (robotDistance > 200 || getEnergy() < 15) {
		fire(1);
	} else if (robotDistance > 50) {
		fire(2);
	} else {
		fire(3);
	}
```

```
	}

	public void onDeath(DeathEvent e) {		//死んだ時のイベント
		// もし他の敵が０ならば、言い換えると自分の勝ちなら、
		//そのまま次のラウンドへ向かう
		if (others == 0) {
			return;
		}

		// もし敵の生存が７５％なら、corner に９０足す
		//もし corner が２７０ならば−９０にする
		if ((others - getOthers()) / (double) others < .75) {
			corner += 90;
			if (corner == 270) {
				corner = -90;
			}
			//画面に I died and did poorly... switching corner to corner 数を出力
			out.println("I died and did poorly... switching corner to " + corner);
		} else {
			//画面に I died but did well.  I will still use corner 数を出力
			out.println("I died but did well.  I will still use corner " + corner);
		}
	}
}
```

予想

最優先で角に向かってまでフィールドの角に居座るメリットがよくわからない。動かないので、動いて玉を発射する敵にはすごく弱いだろう。smartFire で撃ち分けているが、ほとんど当たらずに無駄撃ちになってエネルギーの消費が早く、弱いだろうと考える。

Crazy.java

```
package sample;

import robocode.*;

import java.awt.*;


public class Crazy extends AdvancedRobot {
	boolean movingForward;	//真偽値を表す型

	//メインメソッド
	public void run() {
		// 色の指定
		setBodyColor(new Color(0, 200, 0));
		setGunColor(new Color(0, 150, 50));
		setRadarColor(new Color(0, 100, 100));
		setBulletColor(new Color(255, 255, 100));
		setScanColor(new Color(255, 200, 200));

		// 永久ループ
		while (true) {
			// 大きな数を入れて前進を予約する
			setAhead(40000);
			//真を代入
			movingForward = true;
			// 右旋回９０度を予約する
			setTurnRight(90);
			// ここで前進と右旋回を終わるまで実行する
			waitFor(new TurnCompleteCondition(this));
			// 左旋回１８０度を予約する
			setTurnLeft(180);
			// 前進と左旋回を終わるまで実行する。
			waitFor(new TurnCompleteCondition(this));
			// 右旋回１８０度を予約する
			setTurnRight(180);
			// 前進と右旋回を終わるまで実行する
			waitFor(new TurnCompleteCondition(this));
		}
	}

	public void onHitWall(HitWallEvent e) {
		// 下記に記述
```

```
            reverseDirection();
        }

        public void reverseDirection() {
            if (movingForward) {
                //前進をして壁にぶつかったなら後退を予約する。
                //後退するので movingForward に false を代入する
                setBack(40000);
                movingForward = false;
            } else {
                //後退をして壁にぶつかったらなら前進を予約する
                //前進するので movingForward に false を代入する
                setAhead(40000);
                movingForward = true;
            }
        }

        public void onScannedRobot(ScannedRobotEvent e) {
            //単純に敵を発見したら火力 1 で攻撃する
            fire(1);
        }

        public void onHitRobot(HitRobotEvent e) {
            // 自分からぶつかった場合は真を返し
            //reverseDirection()を呼び出します
            if (e.isMyFault()) {
                reverseDirection();
            }
        }
}
```

予想

砲撃の火力が 1 だけでなんのひねりもない弾を撃つので弱いだろう。加えて、意味のない動きを繰り返して、壁にぶつかりエネルギーの消費が激しいと思う。

Fire.java

```
package sample;

import robocode.HitByBulletEvent;
import robocode.HitRobotEvent;
import robocode.Robot;
import robocode.ScannedRobotEvent;
import static robocode.util.Utils.normalRelativeAngleDegrees;

import java.awt.*;              //カラーを使う場合に使用する

public class Fire extends Robot {
        int dist = 50; //onHitByBullet(HitByBulletEvent e)で出てくる。
                                //被弾したときに動く距離。

        public void run() {                             //メインメソッド
                // 色を設定
                setBodyColor(Color.orange);//ボディをオレンジ
                setGunColor(Color.orange);  //砲台をオレンジ
                setRadarColor(Color.red);     //レーダーを赤
                setScanColor(Color.red);      //スキャンを赤
                setBulletColor(Color.red);    //砲弾を赤

                // 砲台を５度右に回す。本体を動かすメソッドがこれだけなのでずっと回り
続ける
                while (true) {                          //while 文の中が true なので永久ループする
                        turnGunRight(5);
                }
        }

        public void onScannedRobot(ScannedRobotEvent e) {
                /* e.getDistance()で発見した敵ロボットとの距離の戻り値を返します。
                   getEnergy()で現在のロボットのエネルギーを返します。
                   距離が５０ピスセル以内で、かつ、エネルギーが５０以上残っている場合
に
                   火力３の砲撃をします。
                   それ以外は火力１で砲撃します。
                */
                if (e.getDistance() < 50 && getEnergy() > 50) {
                        fire(3);
                }
                else {
                        fire(1);
```

```
		}
		// scan()メソッドにより、引き続き見えているロボットを狙う
		//つまり onScannedRobot(e)をはじめからやり直します
		scan();
	}

	// 被弾したさいの動き
	public void onHitByBullet(HitByBulletEvent e) {
		turnRight(normalRelativeAngleDegrees(90 - (getHeading() - e.getHeading())));

		ahead(dist);	//前方へ移動
		dist *= -1;		//変数 dist に－１をかけて、移動する方向を反転している
		scan();			//onScannedRobot(e)を呼び出す。
	}

	//衝突した時の動き
	public void onHitRobot(HitRobotEvent e) {
		double turnGunAmt = normalRelativeAngleDegrees(e.getBearing() + getHeading() -
getGunHeading());
		//ぶつかったとき砲台をすぐさま相手に向け最大火力で反撃する
		turnGunRight(turnGunAmt);
		fire(3);
	}
}
```

予想

やはり動かないので、動き回る敵にはものすごく弱いだろう。被弾したときに少しだけ逃げるモーションをするけれど、あまり意味はなく集中砲火を食らってしまうだろう。

Interactive.java

```
package sample;

import robocode.AdvancedRobot;
import static robocode.util.Utils.normalAbsoluteAngle;
import static robocode.util.Utils.normalRelativeAngle;

import java.awt.*;
import java.awt.event.KeyEvent;
import static java.awt.event.KeyEvent.*;
import java.awt.event.MouseEvent;
import java.awt.event.MouseWheelEvent;


/**
 キーボード
 * - ↑　前進
 * - ↓　後退
 * - →　右旋回
 * - ←　左旋回

 マウス
 * - ホイール上　　前進
 * - ホイール下　　後退
 * - ボタン 1　　火力 1
 * - ボタン 2　　火力 2
 * - ボタン 3　　火力 3
 *
 攻撃の威力により砲弾の色が変化
 * -  1　黄色
 * -  2　オレンジ
 * -  3　赤
 *
 */
public class Interactive extends AdvancedRobot {

	int moveDirection;

	int turnDirection;

	double moveAmount;

	int aimX, aimY;
```

```
	int firePower;

	public void run() {

		// 色の指定
		setColors(Color.BLACK, Color.WHITE, Color.RED);

		for (;;) {			//ここでたくさんの行動を予約する
			setAhead(moveAmount * moveDirection);
			moveAmount = Math.max(0, moveAmount - 1);
			setTurnRight(45 * turnDirection); // degrees
			double angle = normalAbsoluteAngle(Math.atan2(aimX - getX(), aimY -
getY()));
			setTurnGunRightRadians(normalRelativeAngle(angle -
getGunHeadingRadians()));
			if (firePower > 0) {
				setFire(firePower);
			}
			execute();		//ここで予約を解除する
		}
	}

	// なにかキーを押された時の行動
	public void onKeyPressed(KeyEvent e) {
		switch (e.getKeyCode()) {		//switch 文でキーを判断
		case VK_UP:
			moveDirection = 1;
			moveAmount = Double.POSITIVE_INFINITY;
			break;

		case VK_DOWN:
			moveDirection = -1;
			moveAmount = Double.POSITIVE_INFINITY;
			break;

		case VK_RIGHT:
			turnDirection = 1;
			break;

		case VK_LEFT:
			turnDirection = -1;
			break;
		}
	}

	// 何も押されていない時は止まるように設定
	public void onKeyReleased(KeyEvent e) {
		switch (e.getKeyCode()) {
		case VK_UP:
```

```
        case VK_DOWN:
                moveDirection = 0;
                moveAmount = 0;
                break;

        case VK_RIGHT:
        case VK_LEFT:
                turnDirection = 0;
                break;
        }
}

// ホイールでの移動
public void onMouseWheelMoved(MouseWheelEvent e) {
        moveDirection = (e.getWheelRotation() < 0) ? 1 : -1;
        moveAmount += Math.abs(e.getWheelRotation()) * 5;
}

// マウスによるレーダーの位置設定
public void onMouseMoved(MouseEvent e) {
        aimX = e.getX();
        aimY = e.getY();
}

// マウスクリックで砲撃
public void onMousePressed(MouseEvent e) {
        if (e.getButton() == MouseEvent.BUTTON3) {
                // ボタン３ならば火力３に設定。砲弾は赤色
                firePower = 3;
                setBulletColor(Color.RED);
        } else if (e.getButton() == MouseEvent.BUTTON2) {
                // ボタン２ならば火力２に設定。砲弾はオレンジ
                firePower = 2;
                setBulletColor(Color.ORANGE);
        } else {
                // ボタン１ならば火力１に設定。砲弾は黄色
                firePower = 1;
                setBulletColor(Color.YELLOW);
        }
}

// マウスボタンから手を離すと砲撃終了
public void onMouseReleased(MouseEvent e) {
        firePower = 0;
}
public void onPaint(Graphics2D g) {
        g.setColor(Color.RED);
        g.drawOval(aimX - 15, aimY - 15, 30, 30);
        g.drawLine(aimX, aimY - 4, aimX, aimY + 4);
        g.drawLine(aimX - 4, aimY, aimX + 4, aimY);
```

        }
}

予想

　自分で動かすことができるロボットがあると知って驚いた。少し動かしてみたが、とても難しい。マウスを持っていないのでホイールの前進後退も使用することができなかった。コンピュータなんかには負けたくないので上位を狙いたい。

RamFire.java

```
package sample;


import robocode.HitRobotEvent;
import robocode.Robot;
import robocode.ScannedRobotEvent;

import java.awt.*;


public class RamFire extends Robot {
	int turnDirection = 1; // 回転方向の変更に使う

	public void run() {
		// 色の指定
		setBodyColor(Color.lightGray);
		setGunColor(Color.gray);
		setRadarColor(Color.darkGray);

		while (true) {
			//メインはただ右に回るだけ。turnDirection で回転方向を変更する。
			turnRight(5 * turnDirection);
		}
	}

	public void onScannedRobot(ScannedRobotEvent e) {
		// 発見したロボットの方向を向くように回転を制御している
		if (e.getBearing() >= 0) {
			turnDirection = 1;
		} else {
			turnDirection = -1;
		}

		turnRight(e.getBearing());
		ahead(e.getDistance() + 5);
		scan();
		// 発見したロボットの方向を向き、その場所まで一直線に向かう
		//その場所に敵がいなくなっても向かう
	}

	public void onHitRobot(HitRobotEvent e) {
		// 発見したロボットの方向を向くように回転を制御している
		if (e.getBearing() >= 0) {
			turnDirection = 1;
		} else {
			turnDirection = -1;
```

```
                }
                // 発見したロボットの方向を向くように回転する
                turnRight(e.getBearing());

                // エネルギーの量に応じて火力をかえている。
                // ぶつからない限り砲撃しないので確実に当たると踏んで、ほぼ強めに設定
していると思う
                if (e.getEnergy() > 16) {
                        fire(3);
                } else if (e.getEnergy() > 10) {
                        fire(2);
                } else if (e.getEnergy() > 4) {
                        fire(1);
                } else if (e.getEnergy() > 2) {
                        fire(.5);
                } else if (e.getEnergy() > .4) {
                        fire(.1);
                }
                //ガシガシ突進してぶつかっていく
                ahead(40);
        }
}
```

予想

敵に突進するということは相手からの砲撃を食らう確立もあがるので、短期決戦向きなロボットだと思う。1 対 1 なら上位に食い込むと思うが多人数だとすぐに息が切れてしまうと思われる。とくに逃げながら戦う敵と、近づくにつれて砲撃の火力が上がる敵には勝率も下がるだろう。

SittingDuck.java

```
package sample;

import robocode.AdvancedRobot;
import robocode.RobocodeFileOutputStream;

import java.awt.*;
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.IOException;
import java.io.PrintStream;


public class SittingDuck extends AdvancedRobot {          //SittingDuck というアドバンスドロボット
        static boolean incrementedBattles = false;

        public void run() {
                setBodyColor(Color.yellow);          //ボディを黄色にする
                setGunColor(Color.yellow);           //砲台を黄色にする

                int roundCount, battleCount; // ２つの変数を宣言

                try {
                        BufferedReader r = new BufferedReader(new
FileReader(getDataFile("count.dat")));

                        // カウントを取得する
                        roundCount = Integer.parseInt(r.readLine());
                        battleCount = Integer.parseInt(r.readLine());
                } catch (IOException e) {
                        roundCount = 0;
                        battleCount = 0;
                } catch (NumberFormatException e) {
                        roundCount = 0;
                        battleCount = 0;
                }

                // roundCount に＋１する
                roundCount++;
                if (!incrementedBattles) {
                        // もしバトル数を増加していなければ
                        //バトルカウントを＋１する
                        battleCount++;
                        //incrementedBattles に true を設定する
                        incrementedBattles = true;
```

```
		}

		PrintStream w = null;

		try {
			// PrentStream はプリント中に IOException を投げず、フラグをセットするだけである。
			//ここでチェックを試みる
			w = new PrintStream(new RobocodeFileOutputStream(getDataFile("count.dat")));

			w.println(roundCount);
			w.println(battleCount);
			if (w.checkError()) {
				out.println("I could not write the count!");
			}
		} catch (IOException e) {
			out.println("IOException trying to write: ");
			e.printStackTrace(out);
		} finally {
			if (w != null) {
				w.close();
			}
		}

		out.println("I have been a sitting duck for " + roundCount + " rounds, in " + battleCount + " battles.");
	}
}
```

予想

名前の通りカモ！！やるまでもなく全戦敗北だろう！！

SpinBot.java

```
package sample;


import robocode.AdvancedRobot;
import robocode.HitRobotEvent;
import robocode.ScannedRobotEvent;

import java.awt.*;


public class SpinBot extends AdvancedRobot {

	public void run() {
		// 色の指定
		setBodyColor(Color.blue);
		setGunColor(Color.blue);
		setRadarColor(Color.black);
		setScanColor(Color.yellow);

		// set コマンドで右に旋回しながら前進します。
		//つまり円運動をします。
		while (true) {
			setTurnRight(10000);
			// 回転速度を５に設定
			setMaxVelocity(5);
			ahead(10000);
		}
	}

	public void onScannedRobot(ScannedRobotEvent e) {
		//敵を見つけると最大火力で砲撃します
		fire(3);
	}

	public void onHitRobot(HitRobotEvent e) {
		//もし敵が自分の砲台の前にぶつかったなら
		//最大火力で反撃します
		if (e.getBearing() > -10 && e.getBearing() < 10) {
			fire(3);
		}
		//もし自分の進行でぶつかったなら,isMyFault で
		//真の値を返します。そして右に１０度旋回します。
		if (e.isMyFault()) {
			turnRight(10);
		}
```

        }
}

予想

　動きながら撃つので攻守はそこそこ良いと思われる。衝突時の反撃もきっちり当たると思う。しかし、すべて最大火力なのと、同じところをぐるぐる回るので出現場所によってかなり強さにムラが出ると思う。

Target.java

```
package sample;


import robocode.AdvancedRobot;
import robocode.Condition;
import robocode.CustomEvent;

import java.awt.*;


public class Target extends AdvancedRobot {

	int trigger; // カスタムイベントで使う変数

	public void run() {
		// 色の指定
		setBodyColor(Color.white);
		setGunColor(Color.white);
		setRadarColor(Color.white);

		// 80 を代入する
		trigger = 80;
		// "triggerhit"という名前のカスタムイベントを設定する
		addCustomEvent(new Condition("triggerhit") {
			public boolean test() {
				// 自機のエネルギーが trigger 以下になると真を返す
				return (getEnergy() <= trigger);
			}
		});
	}

	public void onCustomEvent(CustomEvent e) {
		//カスタムイベントが起きたときの内容
		if (e.getCondition().getName().equals("triggerhit")) {
			// tregger から－２０する
			trigger -= 20;
			// 減ったエネルギーを文章と一緒に表示
			out.println("Ouch, down to " + (int) (getEnergy() + .5) + " energy.");
			// 少し移動する
			turnLeft(65);
			ahead(100);
		}
	}
}
```

予想

　砲撃するメソッドがないのでもちろん弱い。逃げるだけ SittingDuck よりは少しましかと思う。

TrackFire.java

```
package sample;


import robocode.Robot;
import robocode.ScannedRobotEvent;
import robocode.WinEvent;
import static robocode.util.Utils.normalRelativeAngleDegrees;

import java.awt.*;


public class TrackFire extends Robot {

	public void run() {
		// 色の指定
		setBodyColor(Color.pink);
		setGunColor(Color.pink);
		setRadarColor(Color.pink);
		setScanColor(Color.pink);
		setBulletColor(Color.pink);

		while (true) {
			turnGunRight(10); // 右に砲台を回転し続ける
		}
	}

	public void onScannedRobot(ScannedRobotEvent e) {
		//double 型の absoluteBearing を作成する。
		//自機の絶対角度と敵の相対角度を取得して足したものを保存する
		double absoluteBearing = getHeading() + e.getBearing();
		//double 型の bearingFromGun を作成する
		double bearingFromGun = normalRelativeAngleDegrees(absoluteBearing -
getGunHeading());

		// bearingFromGun の値が 3 以下ならば右に砲台を bearingFromGun だけ向ける
		if (Math.abs(bearingFromGun) <= 3) {
			turnGunRight(bearingFromGun);
			// 砲台の熱が 0 なら（打てる状態なら）撃つ
			if (getGunHeat() == 0) {
				//3 - bearingFromGun と getEnergy - .1 を比較し
				//小さい方を撃つ
				fire(Math.min(3 - Math.abs(bearingFromGun), getEnergy() - .1));
			}
		}
		// 他の場合、bearingFromGun だけ右回転
```

```
                else {
                        turnGunRight(bearingFromGun);
                }
                //bearingFromGun が 0 なら、もう一度スキャンする。
                if (bearingFromGun == 0) {
                        scan();
                }
        }

        public void onWin(WinEvent e) {
                // 勝利した時のイベントで、生き残ったとき右に 36000 度回転する
                turnRight(36000);
        }
}
```

予想

動かないので弱いとは思うが、砲撃の火力の計算に手が込んでいるので強くなるのかもしれない。

Tracker.java

```
package sample;


import robocode.HitRobotEvent;
import robocode.Robot;
import robocode.ScannedRobotEvent;
import robocode.WinEvent;
import static robocode.util.Utils.normalRelativeAngleDegrees;

import java.awt.*;


public class Tracker extends Robot {
	int count = 0; // int 型 count を 0 で初期化し宣言
	double gunTurnAmt; // double 型で gunTurnAmt を宣言
	String trackName; // 現在追跡しているロボットの名前を保存する

	public void run() {			//メインメソッド
		// 色の指定
		setBodyColor(new Color(128, 128, 50));
		setGunColor(new Color(50, 50, 20));
		setRadarColor(new Color(200, 200, 70));
		setScanColor(Color.white);
		setBulletColor(Color.blue);

		trackName = null; // null を代
		setAdjustGunForRobotTurn(true); // 旋回中は砲台は固定する
		gunTurnAmt = 10; // gunTurnAmt に 10 を代入

		// 永久ループ
		while (true) {
			// gunTurnAmt だけ砲台を右回転する
			//最初は１０度だけ
			turnGunRight(gunTurnAmt);
			// count に＋１する
			count++;
			// count が２になるまでに敵を見つけられなかった場合
			//gunTurnAmt を-10 に変更して辺りを見回す
			if (count > 2) {
				gunTurnAmt = -10;
			}
			// count が５になるまでに敵を見つけられなかった場合
			//gunTurnAmt を 10 に変更してさらに辺りを見回す
```

```
                if (count > 5) {
                        gunTurnAmt = 10;
                }
                // If we *still* haven't seen our target after 10 turns, find another target
                if (count > 11) {
                        trackName = null;
                }
        }
    }

    public void onScannedRobot(ScannedRobotEvent e) {

        // もしもう標的が決まっているのに他の敵を見つけてた場合、無視して
        //現在の標的を狙うようにする。
        if (trackName != null && !e.getName().equals(trackName)) {
            return;
        }

        // trackName に入っているのが null のとき、見つけた敵の名前を
        //e.getName()で取得して trackName に代入する
        if (trackName == null) {
            trackName = e.getName();
            out.println("Tracking " + trackName);
        }
        // count をリセット
        count = 0;
        // 敵と１５０ピクセルより遠い時は、その敵の方角を向いて接近する
        if (e.getDistance() > 150) {
            gunTurnAmt = normalRelativeAngleDegrees(e.getBearing() + (getHeading()
- getRadarHeading()));
            turnGunRight(gunTurnAmt);
            turnRight(e.getBearing());
            //標的にぶつかるほど近づかないように 140 ピクセルまで前進
            ahead(e.getDistance() - 140);
            return;
        }

        // 敵ロボが近い場合は火力３で攻撃
        gunTurnAmt = normalRelativeAngleDegrees(e.getBearing() + (getHeading() -
getRadarHeading()));
        turnGunRight(gunTurnAmt);
        fire(3);

        // 敵ロボが１００ピクセルより近い場合
        if (e.getDistance() < 100) {
            if (e.getBearing() > -90 && e.getBearing() <= 90) {
                back(40);       //前にいたら後退
            } else {
                ahead(40);      //後ろにいたら前進
```

```
                }
            }
            scan();
        }

        public void onHitRobot(HitRobotEvent e) {
            if (trackName != null && !trackName.equals(e.getName())) {
                out.println("Tracking " + e.getName() + " due to collision");
            }
            // ぶつかった相手を新しいターゲットに変更
            trackName = e.getName();
            // 砲台を敵に向け、火力３で砲撃。その後５０ピクセル後退
            gunTurnAmt = normalRelativeAngleDegrees(e.getBearing() + (getHeading() -
getRadarHeading()));
            turnGunRight(gunTurnAmt);
            fire(3);
            back(50);
        }

        //勝利時のイベント
        public void onWin(WinEvent e) {
            for (int i = 0; i < 50; i++) {
                turnRight(30);
                turnLeft(30);
            }
        }
}
```

予想

大人数では標的を狙いすぎるあまり、自分が標的にされていたということが多いのではないだろうか。1 対 1 なら RamFire よりも強いと思われる。しかし、よけるモーションがないということもあるので 1 位ではないと思う。

Walls.java

```
package sample;


import robocode.HitRobotEvent;
import robocode.Robot;
import robocode.ScannedRobotEvent;

import java.awt.*;


public class Walls extends Robot {

	boolean peek;				// 真理値 peek を設定
	double moveAmount;

	public void run() {
		// 色の指定
		setBodyColor(Color.black);		//ボディと砲台を黒にする
		setGunColor(Color.black);
		setRadarColor(Color.orange);		//レーダーを黄色にする
		setBulletColor(Color.cyan);		//砲弾とスキャンを
		setScanColor(Color.cyan);

		// バトルフィールドの幅と高さを取得して、その最大値を代入
		moveAmount = Math.max(getBattleFieldWidth(), getBattleFieldHeight());
		// 偽を代入
		peek = false;

		// 左方向に旋回して、一番近い上下左右の方向を向きます。
		// getHeading()を使って自機の絶対角度を取得し、それを９０で割っています。
		// その後
		turnLeft(getHeading() % 90);
		ahead(moveAmount);

		peek = true;		//真を代入
		turnGunRight(90);	//砲台を９０度右に回転
		turnRight(90);		//右に９０度旋回

		while (true) {
			peek = true;	//真を代入。
			ahead(moveAmount);
			peek = false;	//偽を代入。
						//フィールドの角で旋回する時は連射しない
```

ようにするため。

```
                turnRight(90);          //右に９０度旋回
            }
        }

        public void onHitRobot(HitRobotEvent e) {
            // もし敵機にぶつかった場合、自機の砲台を基準にして－９０～９０度にぶつかると
            //後退する
            if (e.getBearing() > -90 && e.getBearing() < 90) {
                back(100);
            } // それ以外は前進する
            else {
                ahead(100);
            }
        }

        public void onScannedRobot(ScannedRobotEvent e) {      //敵を見つけるとすぐ発射する
            fire(2);
            // フィールドの角で連射しないようにために peek に論理式を入れて制御している。
            // 移動しているときには peek は真なので、onScannedRobot を繰り返す。
            if (peek) {
                scan();
            }
        }
}
```

予想

いろいろ調べているとこれが sample の中で最強だというのがわかってきたが、なぜ強いのかはプログラムを見ただけではよくわからない。

対戦結果

SittingDuck と Target は攻撃をしないロボットなので、当たり前だが全敗だった。しかし、Target は若干動くので、近くの壁にぶつかって SittingDuck よりもさきにエネルギー切れになることがあった。

最も強かったのは、全勝した Walls だった。Walls と競るロボットはおらず、全戦でエネルギーを多く残し、圧倒的に勝っていた。

その次に勝利数が多かったのは、SpinBot であった。Corner や TrakFire など動かずに狙ってくるロボットと対戦すると、うまい具合に砲弾を避けながら反撃することができていた。　Tracker と RumFire は敵に向かっていって撃ち合うという、避けることをあまりしないという点で似ているのだが、Tracker の方が RumFire に勝ち、勝利数も多かった。

Interactive は 6 勝することができた。すべてのプログラムを見てきただけあって、弾を避けるのは楽勝だった。もう少し練習すれば 1 対 1 で負けることはないだろう。

勝利数が低かったのが、Corner と Fire と Crazy だった。

考察

今回の対戦を見て、強い機体とは、いかに敵に砲弾をたくさんあてるかではなく、いかに砲弾を避けるかだと思った。Walls がなぜ圧倒的に強かったのか考えてみたい。まず、Walls は壁際を一直線に長く走ることができたことが一番の強みだと思う。Fire や Corner などの移動しないロボットの攻撃はほぼ当たらないからである。追ってくる Tracker や RumFire も追いつくことができず、攻撃を当てることができなかった。Walls は避けながら砲撃するという攻防一体であったから強かったんだと思う。

もうひとつ、避けることが強いというのは理由がある。それは、Walls の命中率はそんなに高くないというところにある。勝利数の低い Cragy と戦うと、上位の TrackFire よりも砲撃が当たりづらい分、長い時間がかかるが負けることはない。長い時間チャンスを持つことが重要なのだと思う。

感想

少しのアイディアで強くなったり弱くなったりするので、奥が深いゲームだなと思いました。Java も授業で初めて触ったような初心者なのに、考察でもっともらしいことを書いたことが恥ずかしいです。もっと詳しく勉強してみようと思いました。

参考文献

・Java プログラマーにもなれちゃう　Robocode&ゲームプログラミング学習術
　２００３年発行　可知　豊　著　ソシム株式会社
・http://www.solar-system.tuis.ac.jp/Java/robocode_api/