プログラミング

Report# 1

提出日　：2009 年 11 月 09 日
所　属　：工学部情報工学科
学籍番号：095739K
氏　名　：當銘　孔太

＊目　次

1. 各サンプルプログラムソースにコメントをつけプログラムをくわしく説明しなさい.

1.1 Corners.java …… 1
1.2 Crazy.java …… 4
1.3 Fire.java …… 6
1.4 Interactive.java …… 8
1.5 RamFire.java …… 11
1.6 SittingDuck.java …… 13
1.7 SpinBot.java …… 15
1.8 Target.java …… 16
1.9 Tracker.java …… 17
1.10 TrackFire.java …… 20
1.11 Walls.java …… 22

2. 各ロボットを対戦させ，各ロボットの特徴を調査しなさい.

2.1 各ロボットを 1 対 1 で対戦させた結果と考察 …… 24

3. 感想

4. 参考文献

1. 各サンプルプログラムソースにコメントをつけプログラムをくわしく説明しなさい.

1.1 Corners.java

```
package sample;

import robocode.DeathEvent;
    // robocode.DeathEvent をインポート
import robocode.Robot;
    // robocode.Robot をインポート。
import robocode.ScannedRobotEvent;
    // robocode.ScannedRobotEvent をインポート。
import static robocode.util.Utils.normalRelativeAngleDegrees;
    // robocode.util.Utils.normalRelativeAngleDegrees をインポート。
import java.awt.*;
    // java.awt.* をインポート。

public class Corners extends Robot {
    int others;                 // int型変数others を宣言。
                                // 他のロボットの数を持つ。

    static int corner = 0;      // static int型変数corner を宣言。
                                // 現在、どこの隅なのかを記憶。

    boolean stopWhenSeeRobot = false;     // boolean は論理値型。

    public void run() {             // run メソッド。
      setBodyColor(Color.red);      // BodyColor を red に設定。
      setGunColor(Color.black);     // GunColor を black に設定。
      setRadarColor(Color.yellow);  // RadarColor を yellow に設定。
      setBulletColor(Color.green);  // BulletColor を green に設定。
      setScanColor(Color.green);    // ScanColor を green に設定。

      others = getOthers();   // getOthers() で残っている口ボットの数を取得。
                              // その値を int型変数others に代入する。

      goCorner();             // goCorner() を呼び出す。

      int gunIncrement = 3;   // int型変数gunIncrement を宣言。
                              // gunIncrement に 3 を代入。

      while (true) {
            for (int i = 0; i < 30; i++) {
            // int型変数 i は、カウンタとして使う。
                  turnGunLeft(gunIncrement);
                  // turnGunLeft() で砲台を左に回転。
            }     // gunIncrement が 3 なので、左に3度回転。
            gunIncrement *= -1;     // gunIncrement に -1 を掛ける。
        }
    }
```

```
public void goCorner() {        // Corner(隅)の方に行く。
  stopWhenSeeRobot = false;     // stopWhenSeeRobot に false を代入。

  turnRight(normalRelativeAngleDegrees(corner - getHeading()));
        // getHeading() で、現在のロボットの方向を取得。
        // corner - getHeading() の計算結果を
        // normalRelativeAngleDegrees() に渡す。
        // 返って来た値分右に回転する。

  stopWhenSeeRobot = true;      // stopWhenSeeRobot に true を代入。
  ahead(5000);                  // 5000px だけ前進。
  turnLeft(90);                 // 本体を左に 90度回転。
  ahead(5000);                  // 5000px だけ前進。
  turnGunLeft(90);              // 砲台を 90度左回転。
}

public void onScannedRobot(ScannedRobotEvent e) {
  // 敵をスキャンして、攻撃する。

  if (stopWhenSeeRobot) {       // stopWhenSeeRobot が true なら実行。
        stop();                 // stop() で停止。

        smartFire(e.getDistance());
        // getDistance() で相手との距離を取得し、
        // smartFire() に値を渡す。

        scan();                 // 他のロボットをスキャンする。
        resume();               // stop() から復帰。
  } else {
        smartFire(e.getDistance());
  }
}

public void smartFire(double robotDistance) {
  // 敵との距離によって砲弾の威力を変える。

  if (robotDistance > 200 || getEnergy() < 15) {
        fire(1);
        // getEnergy() で自機の残りエネルギーを取得。
        // 敵との距離が200以上 もしくは，残りエネルギーが15以下なら
        // 砲弾の威力を 1 に設定。

  } else if (robotDistance > 50) {
        fire(2);
        // 敵との距離が 200以下 50以上である時、
        // 砲弾の威力を 2 に設定。
```

```
    } else {
            fire(3);
            // 上記の条件以外の時、砲弾の威力を 3 に設定。

      }
    }

    public void onDeath(DeathEvent e) {
      // 自機が破壊された後の行動。

      if (others == 0) {
            return;
            // 残りのロボットの数が 0 ( 自機の勝利 ) の場合、次の試合へ。
      }

      if ((others - getOthers()) / (double) others < .75) {
            // 残りのロボットが、全体の 75% の場合。

            corner += 90;           // corner に 90 を加える。
            if (corner == 270) {    // corner が 270 の場合、
                  corner = -90;     // corner に -90 を加える。
            }
            out.println("I died and did poorly... switching corner to " + corner);
            // "I died and did poorly... switching corner to (cornerの数字)" を出力。

      } else {
            out.println("I died but did well.  I will still use corner " + corner);
            // "I died but did well.  I will still use corner (cornerの数字)" を出力。
      }
    }
}
```

1.2 Crazy.java

```
package sample;

import robocode.*;
import java.awt.*;

public class Crazy extends AdvancedRobot {
    boolean movingForward;                    // boolean型変数 movingForward を宣言。

    public void run() {
      setBodyColor(new Color(0, 200, 0));     // ロボットの色を設定。
      setGunColor(new Color(0, 150, 50));
      setRadarColor(new Color(0, 100, 100));
      setBulletColor(new Color(255, 255, 100));
      setScanColor(new Color(255, 200, 200));

      while (true) {
            setAhead(40000);                  // 40000px だけ前進するように設定。
            movingForward = true;             // movingForward に true を代入。
            setTurnRight(90);                 // 本体を右に 90度回転するように設定。
            waitFor(new TurnCompleteCondition(this));
                                              // 上記で設定した動きをする。
                                              // この場合、前進しながら右方向に進む。

            setTurnLeft(180);                 // 左に 180度回転するように設定。
            waitFor(new TurnCompleteCondition(this));

            setTurnRight(180);                // 右に 180度回転するように設定。
            waitFor(new TurnCompleteCondition(this));
      }                                       // 8の字に口ボットは動く。
    }

    public void onHitWall(HitWallEvent e) {    // 壁に当たった後の行動
      reverseDirection();                       // reverseDirection() を呼び出す。

    }

    public void reverseDirection() {
      if (movingForward) {                      // 前進の場合、
            setBack(40000);                     // 40000px 後進するように設定。
            movingForward = false;              // movingForward を false にする。

      } else {                                  // 後進の場合
            setAhead(40000);                    // 40000px 前進するように設定。
            movingForward = true;               // movingForward を true にする。
      }
    }
```

```
    public void onScannedRobot(ScannedRobotEvent e) {
      fire(1);                                  // 敵をスキャンしたら、1 の威力で攻撃。
    }

    public void onHitRobot(HitRobotEvent e) {   // ロボットに当たった時...
      if (e.isMyFault()) {                      // 自らぶつかった場合、
            reverseDirection();                 // reverseDirection() を呼び出す。
      }
    }
}
```

1.3 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.*;

public class Fire extends Robot {
    int dist = 50;                          // int型変数 dist を宣言。
                                            // 初期値を 50 に設定。
    public void run() {
      setBodyColor(Color.orange);           // ロボットの色を設定。
      setGunColor(Color.orange);
      setRadarColor(Color.red);
      setScanColor(Color.red);
      setBulletColor(Color.red);

      while (true) {
            turnGunRight(5);                // 砲台を右に 5度回転させる。
      }
    }

    public void onScannedRobot(ScannedRobotEvent e) {
      if (e.getDistance() < 50 && getEnergy() > 50) {
            fire(3);
            // 敵との距離が 50以下 かつ 残りのエネルギーが 50以上の場合、
            // 砲弾の威力 3 で攻撃。

      } else {
            fire(1);
            // 上記以外の条件の場合、
            // 砲弾の威力 1 で攻撃。
      }
      scan();                   // 敵をスキャンする。
    }

    public void onHitByBullet(HitByBulletEvent e) {  // 敵の攻撃に当たった場合、

      turnRight(normalRelativeAngleDegrees(90 - (getHeading() - e.getHeading())));
                        // 敵の攻撃に対して垂直になるように、右回転する。
      ahead(dist);      // dist分だけ前進する。
      dist *= -1;       // dist に -1 を掛ける。
      scan();           // 敵をスキャンする。
    }

    public void onHitRobot(HitRobotEvent e) {  // 敵ロボットにぶつかった場合。
```

```
      double turnGunAmt = normalRelativeAngleDegrees(e.getBearing()
                                    + getHeading() - getGunHeading());
            // double型変数 turnGunAmt にぶつかった敵ロボットと自機との相対角度を設定。

      turnGunRight(turnGunAmt);     // turnGunAmt の分だけ砲台を右に回転。
      fire(3);                      // 砲弾の威力 3 で攻撃。
    }
}
```

1.4 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;

public class Interactive extends AdvancedRobot {

    int moveDirection;              // int型変数 moveDirection を宣言。
    int turnDirection;              // int型変数 turnDirection を宣言。
    double moveAmount;              // double型変数 moveAmount を宣言。
    int aimX, aimY;                 // int型変数 amiX と amiY を宣言。
    int firePower;                  // int型変数 firePower を宣言。

    public void run() {

      setColors(Color.BLACK, Color.WHITE, Color.RED);       // ロボットの色を設定。

      for (;;) {
            setAhead(moveAmount * moveDirection);
                  // [moveAmount * moveDirection] の分だけ前進するように設定。
            moveAmount = Math.max(0, moveAmount - 1);
                  // 0 と moveAmount -1 の内大きい方を moveAmount に代入。
            setTurnRight(45 * turnDirection);
                  // [45 * turnDirection] の分だけ右に回転するように設定。
            double angle = normalAbsoluteAngle(Math.atan2(aimX - getX(),
                                                          aimY - getY()));
                  // grtX() で X座標を、getY() で Y座標を取得。
                  // Math.atan2 は、座標 ( {amiX - getX()} , {amiY - getY()} ) を
                  // 極座標表示したときの角度を返す。
                  // normalAbsoluteAngle() で、-180度～180度の範囲に直したものを、
                  // double型変数 angle に代入する。
            setTurnGunRightRadians(normalRelativeAngle(angle
                                        - getGunHeadingRadians()));
                  // getGunHeadingRadians() で、砲台の向いている方向をラジアンで返す。
                  // [angle - getGunHeadingRadians()] を, -180度～180度の範囲に直し、
                  // その分だけ、砲台が右回転するように設定する。
```

```
            if (firePower > 0) {          // firePower が 0 以上の場合、
                  setFire(firePower);     // firePower の威力の砲弾で攻撃する。
            }

            execute();                    // 上記で設定した行動をする。
      }
  }

  public void onKeyPressed(KeyEvent e) {      // キーが押された時の行動。
      switch (e.getKeyCode()) {           // getKeyCode() で,押されたキーのコードを取得。
      case VK_UP:
            moveDirection = 1;                      // moveDirection に 1 を代入。
            moveAmount = Double.POSITIVE_INFINITY;
            // double型の正の無限大値を保持する定数、POSITIVE_INFINITY を moveAmount に代入。
            break;

      case VK_DOWN:
            moveDirection = -1;                       // moveDirection に-1 を代入。
            moveAmount = Double.POSITIVE_INFINITY;
            break;

      case VK_RIGHT:
            turnDirection = 1;                        // turnDirection に 1 を代入。
            break;

      case VK_LEFT:
            turnDirection = -1;                       // turnDirection に -1 を代入。
            break;
      }
  }

  public void onKeyReleased(KeyEvent e) {     // キー入力がない場合、
      switch (e.getKeyCode()) {
      case VK_UP:
      case VK_DOWN:
            moveDirection = 0;                  // moveDirection に 0 を代入。
            moveAmount = 0;                     // moveAmount に 0 を代入。
            break;

      case VK_RIGHT:
      case VK_LEFT:
            turnDirection = 0;                  // turnDirection に 0 を代入。
            break;
      }
  }
```

```
    public void onMouseWheelMoved(MouseWheelEvent e) {      // マウスホイールによる移動。
      moveDirection = (e.getWheelRotation() < 0) ? 1 : -1;
            // getWheelRotation() で、マウスホイールを回転させた「クリック」数を返す。
            // クリック数が、0 以下なら 1 を,そうでないならば、-1 を moveDirection に代入。
      moveAmount += Math.abs(e.getWheelRotation()) * 5;
            // getWheelRotation() で得た値の絶対値を取り、その値に 5 を掛ける。
            // 上記で計算した値を moveAmount に加える。
    }

    public void onMouseMoved(MouseEvent e) {
      aimX = e.getX();          // getX() で得た X座標を amiX に代入。
      aimY = e.getY();          // getY() で得た Y座標を amiY に代入。
    }

    public void onMousePressed(MouseEvent e) {
      if (e.getButton() == MouseEvent.BUTTON3) {      // ボタン3 の場合
            firePower = 3;                            // 砲弾の威力は、3
            setBulletColor(Color.RED);                // 砲弾の色は、Red

      }else if (e.getButton() == MouseEvent.BUTTON2) {      // ボタン2 の場合
            firePower = 2;                                  // 砲弾の威力は、2
            setBulletColor(Color.ORANGE);                   // 砲弾の色は、Orange

      } else {                                        // ボタン1 の場合
            firePower = 1;                            // 砲弾の威力は、1
            setBulletColor(Color.YELLOW);             // 砲弾ん色は、Yellow
      }
    }

    public void onMouseReleased(MouseEvent e) {       //ボタンを押していない場合、
      firePower = 0;                                  // 砲弾の威力は、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);
    }
}
```

1.5 RamFire.java

```
package sample;

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

import java.awt.*;

public class RamFire extends Robot {
    int turnDirection = 1;                      // int型関数 turnDirection を宣言。
                                                // turnDirection の初期値を 1 に設定。
    public void run() {
      setBodyColor(Color.lightGray);            // ロボットの色を設定。
      setGunColor(Color.gray);
      setRadarColor(Color.darkGray);

      while (true) {
            turnRight(5 * turnDirection);       // turnDirection * 5 だけ右に回転。
      }
    }

    public void onScannedRobot(ScannedRobotEvent e) {

      if (e.getBearing() >= 0) {    // 敵のロボットからの攻撃を受けた方向が、0度以上なら、
            turnDirection = 1;      // turnDirection に 1 を代入。

      } else {                      // それ以外なら、
            turnDirection = -1;     // turnDirection に -1 を代入。
      }

      turnRight(e.getBearing());    // getBearing() 分だけ右に回転。
      ahead(e.getDistance() + 5);   // 敵ロボットとの距離 + 5 だけ前進。
      scan();                       // 敵ロボットをスキャンする。
    }

    public void onHitRobot(HitRobotEvent e) {
      if (e.getBearing() >= 0) {
            turnDirection = 1;
      } else {
            turnDirection = -1;
      }
      turnRight(e.getBearing());

      if (e.getEnergy() > 16) {
            fire(3);
            // ロボットのエネルギーが、16以上なら、砲弾の威力 3 で攻撃。
```

```
      }else if (e.getEnergy() > 10) {
            fire(2);
            // 残りエネルギーが、16以下 10以上なら、砲弾の威力 2 で攻撃。

      } else if (e.getEnergy() > 4) {
            fire(1);
            // 残りエネルギーが、10以下 4以上なら、砲弾の威力 1 で攻撃。

      } else if (e.getEnergy() > 2) {
            fire(.5);
            // 残りエネルギーが、4以下 2以上なら、砲弾の威力 0.5 で攻撃。

      } else if (e.getEnergy() > .4) {
            fire(.1);
            // 残りエネルギーが、2以下 0.4以上なら、砲弾の威力 0.1 で攻撃。

      }
      ahead(40);          // 40px だけ前進する。
    }
}
```

1.6 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 {
    static boolean incrementedBattles = false;

    public void run() {
      setBodyColor(Color.yellow);   // ロボットの色を設定。
      setGunColor(Color.yellow);

      int roundCount, battleCount;  // int型変数 roundCount,battleCount を宣言。

      try {
            BufferedReader r = new BufferedReader(new
                                        FileReader(getDataFile("count.dat")));
                  // getDataFile() で、"count.dat" というファイルを作る。
                  // FileReader() で、"count.dat" を読み込む。
                  // BufferedReader() で、ファイルからの入力をバッファする。

            roundCount = Integer.parseInt(r.readLine());
                  // ファイルの 1行を読み取り、int型へ変換した値を、roundCount に代入。

            battleCount = Integer.parseInt(r.readLine());
                  // ファイルの 1行を読み取り、int型へ変換した値を、battleCount に代入。

      } catch (IOException e) {           // ファイルの読み込みに失敗した場合。
            roundCount = 0;               // roundCount をリセット
            battleCount = 0;              // battleCount をリセット

      } catch (NumberFormatException e) {// int型への変換に失敗した場合。
            roundCount = 0;               // roundDount をリセット
            battleCount = 0;              // bayyleCount をリセット
      }

      roundCount++;                       // roundCount をインクリメント。

      if (!incrementedBattles) {          // Battles がインクリメントされていなかった場合。
            battleCount++;                // battleCount をインクリメント
            incrementedBattles = true;    // incrementedBattles を true に設定。
      }
```

```
        PrintStream w = null;

        try {
                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.");
    }
}
```

1.7 SpinBot.java

```
package sample;

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

import java.awt.*;

public class SpinBot extends AdvancedRobot {

    public void run() {
      setBodyColor(Color.blue);           // ロボットの色を設定。
      setGunColor(Color.blue);
      setRadarColor(Color.black);
      setScanColor(Color.yellow);

      while (true) {
            setTurnRight(10000);          // 右に10000度だけ回転するように設定。
            setMaxVelocity(5);            // 回転速度を 5 に設定。
            ahead(10000);                 // 上記の設定と同時に、10000pxだけ前進。
      }
    }

    public void onScannedRobot(ScannedRobotEvent e) {
      fire(3);                            // 砲弾の威力 3 で攻撃
    }

    public void onHitRobot(HitRobotEvent e) {
      if (e.getBearing() > -10 && e.getBearing() < 10) {
            fire(3);
            // -10度 ~ 10度の範囲に敵ロボットがいる場合。
            // 砲弾の威力 3 で攻撃
      }
      if (e.isMyFault()) {    // 自分から敵にぶつかった場合。
            turnRight(10);    // 右に 10度だけ回転。
      }
    }
}
```

1.8 Target.java

```
package sample;

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

import java.awt.*;

public class Target extends AdvancedRobot {

    int trigger;                    // int型変数 trigger を宣言

    public void run() {
      setBodyColor(Color.white);    // ロボットの色を設定。
      setGunColor(Color.white);
      setRadarColor(Color.white);

      trigger = 80;                 // trigger に 80 を代入。

      addCustomEvent(new Condition("triggerhit") {
            // カスタムイベントを追加。コンディション trigerhit を作成。

            public boolean test() {
                  return (getEnergy() <= trigger);
                  // 残りのエネルギーが、trigger 以下なら新を返す。
            }
      });
    }

    public void onCustomEvent(CustomEvent e) {
      if (e.getCondition().getName().equals("triggerhit")) {

            trigger -= 20;          // trigger に -20 を加える。
            out.println("Ouch, down to " + (int) (getEnergy() + .5) + " energy.");
                  // "Ouch, down to (減ったエネルギー) energy."を出力。

            turnLeft(65);     // 左に、65度回転。
            ahead(100);       // 100px だけ前進。
      }
    }
}
```

1.9 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型関数 count を宣言。初期値は 0 に設定。
    double gunTurnAmt;          // double型関数 gunTurnAmt を宣言。
    String trackName;           // string型関数 trackName を宣言。
                                // 追跡している口ボット名を記憶する。

    public void run() {
      setBodyColor(new Color(128, 128, 50));    // ロボットの色を設定。
      setGunColor(new Color(50, 50, 20));
      setRadarColor(new Color(200, 200, 70));
      setScanColor(Color.white);
      setBulletColor(Color.blue);

      trackName = null;                         // trackName に NULL を設定。
      setAdjustGunForRobotTurn(true);           // 回転中、砲台は停止。
      gunTurnAmt = 10;                          // gunTurnAmt に 10 を代入。

      while (true) {
            turnGunRight(gunTurnAmt);           // gunTurnAmt の分だけ砲台を右に回転。
            count++;                            // count をインクリメント

            if (count > 2) {
                  gunTurnAmt = -10;
                  // count が 2 より大きい場合、gunTurnAmt に -10 を代入。
            }
            if (count > 5) {
                  gunTurnAmt = 10;
                  // count が 5 より大きい場合、gunTurnAmt に 10 を代入。
            }
            if (count > 11) {
                  trackName = null;
                  // count が 11より大きい場合、gunTurnAmt に null を設定。
            }
      }
    }
```

```
    public void onScannedRobot(ScannedRobotEvent e) {
      if (trackName != null && !e.getName().equals(trackName)) {
            return;
            // trackName が nullでない（追跡中のロボットがある）かつ、
            // スキャンした敵が、追跡中でない場合は、スキャンし直す。
      }

      if (trackName == null) {
            trackName = e.getName();
            out.println("Tracking " + trackName);
            // trackName が null（追跡中のロボットが無い）の場合、
            // getName() で、スキャンした敵のロボット名を取得し、trackName に設定する。
            // "Tracking (trackName)" を出力する。
      }

      count = 0;        // count をリセットする。

      if (e.getDistance() > 150) {
            // getDistance() で追跡中のロボットとの距離を取得、その値が、150以上の場合、

            gunTurnAmt = normalRelativeAngleDegrees(e.getBearing() +
                                                    (getHeading() - getRadarHeading()));
            turnGunRight(gunTurnAmt);
            turnRight(e.getBearing());
            ahead(e.getDistance() - 140);
            return;
            // 敵ロボットの方向に砲台、本体を回転させ、敵ロボットに近づく。
      }

      gunTurnAmt = normalRelativeAngleDegrees(e.getBearing() +
                                                    (getHeading() - getRadarHeading()));
      turnGunRight(gunTurnAmt);
      fire(3);
      // 敵ロボットが近い場合、砲台を敵方向に回転させ、威力 3 で攻撃。

      if (e.getDistance() < 100) {
            // 敵ロボットとの距離が、100以下の場合、

            if (e.getBearing() > -90 && e.getBearing() <= 90) {
                  back(40);
                  // 敵ロボットが前方にいる場合、40px だけ後進。
            } else {
                  ahead(40);
                  // 後方の場合、40px だけ前進する。
            }
      }
      scan();     // 敵ロボットをスキャン。
    }
```

```
    public void onHitRobot(HitRobotEvent e) {  // 敵ロボットに衝突したとき。
      if (trackName != null && !trackName.equals(e.getName())) {
            out.println("Tracking " + e.getName() + " due to collision");
            // 追跡中のロボットがいない場合、
            // "Tracking (衝突したロボットの名前) due to collision" を出力。
      }
      trackName = e.getName();      // 衝突したロボットを標的に...
      gunTurnAmt = normalRelativeAngleDegrees(e.getBearing() +
                                              (getHeading() - getRadarHeading()));
      turnGunRight(gunTurnAmt);
      fire(3);
      back(50);
      // 敵ロボット方向に砲台を回転、威力 3 で攻撃して、50px 後進。
    }

    public void onWin(WinEvent e) {       // 勝利した場合。
      for (int i = 0; i < 50; i++) {
            turnRight(30);                // 右に 30度回転。
            turnLeft(30);                 // 左に 30度回転。
      }
    }
}
```

1.10 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() {
      setBodyColor(Color.pink);     // ロボットの色を設定。
      setGunColor(Color.pink);
      setRadarColor(Color.pink);
      setScanColor(Color.pink);
      setBulletColor(Color.pink);

      while (true) {
            turnGunRight(10);       // 砲台を 10度だけ右に回転。
      }
    }

    public void onScannedRobot(ScannedRobotEvent e) {
      double absoluteBearing = getHeading() + e.getBearing();
      // duble型変数 absoluteBearing に、
      // "現在の方向" に "敵ロボットの攻撃が当たった方向"分を加えた値を代入。

      double bearingFromGun = normalRelativeAngleDegrees(absoluteBearing
                                                            - getGunHeading());
      // double型変数 bearingFromGun に、normalRelativeAngleDegrees() から返された値を代入。

      if (Math.abs(bearingFromGun) <= 3) {      // bearingFromGun の絶対値が、3以上の場合、
            turnGunRight(bearingFromGun);       // bearingFromGun の分だけ砲台を右に回転。

            if (getGunHeat() == 0) {      // 砲台の熱さが、0である場合。
                  fire(Math.min(3 - Math.abs(bearingFromGun), getEnergy() - .1));
                  // 3 から (bearingFromGunの絶対値) を引いた値と、
                  // (残りエネルギー) から 0.1 引いた値の内、小さい値の威力で攻撃。
            }
      }
      else {                                    // それ以外の場合。
            turnGunRight(bearingFromGun);       // 砲台を bearingFromGun の分だけ右に回転。
      }
      if (bearingFromGun == 0) {                // bearingFromGun が 0 の場合、
            scan();                             // 敵ロボットをスキャン。
      }
    }
```

```
    public void onWin(WinEvent e) {             // 勝利したとき、
      turnRight(36000);                         // 右に 36000度回転する。
    }
}
```

1.11 Walls.java

```
package sample;

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

import java.awt.*;

public class Walls extends Robot {

    boolean peek;                   // boolen型変数 peek を宣言。
    double moveAmount;              // double型変数 moveAmount を宣言。

    public void run() {
      setBodyColor(Color.black);    // ロボットの色を設定。
      setGunColor(Color.black);
      setRadarColor(Color.orange);
      setBulletColor(Color.cyan);
      setScanColor(Color.cyan);

      moveAmount = Math.max(getBattleFieldWidth(), getBattleFieldHeight());
      // getBattleFildWidth() でバトルフィールドの横幅を取得。
      // getBattleFildHeight() でバトルフィールドの高さを取得。
      // 2つの値の内、大きい方を moveAmount に代入。

      peek = false;                 // peek を false に設定。

      turnLeft(getHeading() % 90);  // 現在の向きを 90 で割った余り分、左に回転。
      ahead(moveAmount);            // moveAmount の分だけ前進。
      peek = true;                  // peek を true に設定。
      turnGunRight(90);             // 砲台を 90度右に回転。
      turnRight(90);                // 本体を 90度右に回転。

      while (true) {
            peek = true;            // peek を true に設定。
            ahead(moveAmount);      // moveAmount の分だけ前進。
            peek = false;           // peek を false に設定。
            turnRight(90);          // 右に 90度回転。
      }
    }
```

```
    public void onHitRobot(HitRobotEvent e) {  // 敵ロボットに衝突した時。
      if (e.getBearing() > -90 && e.getBearing() < 90) {
            back(100);
            // 衝突した敵ロボットが、-90度 ~ 90度の範囲にいる場合、100px 後進。
      }
      else {
            ahead(100);
            // それ以外の場合、100px だけ前進
      }
    }

    public void onScannedRobot(ScannedRobotEvent e) {
      fire(2);                  // 威力 2 で攻撃
      if (peek) {
            scan();             // peek が true なら敵ロボットをスキャン。
      }
    }
}
```

2. 各ロボットを対戦させ，各ロボットの特徴を調査しなさい．

2.1　各ロボットを 1 対 1 で対戦させた結果と考察

＊ 対戦条件は以下のように設定。

・ 各ロボットを 1 対 1 で対戦させる、総当たり戦とする。
・ 3 ラウンド勝負とする。
・ フィールドの広さは、800 × 600 とする。

＊ 対戦した結果を以下に、表でまとめた。

	Cor	Cra	Fir	Int	Ram	Sit	Spi	Tar	Tra	TraF	Wal
Cor		×	×	○	×	○	×	○	×	×	×
Cra	○		○	○	×	○	×	○	×	×	×
Fir	○	×		○	×	○	×	○	×	×	×
Int	×	×	×		×	○	×	○	×	×	×
Ram	○	○	○	○		○	○	○	×	×	×
Sit	×	×	×	×	×		×	○	×	×	×
Spi	○	○	○	○	×	○		○	○	○	×
Tar	×	×	×	×	×	×	×		×	×	×
Tra	○	○	○	○	○	○	×	○		×	×
TraF	○	○	○	○	○	○	×	○	○		×
Wal	○	○	○	○	○	○	○	○	○	○	

＊ 考 察

Corners	角から攻撃するロボットなので、止まっている敵には強いが、動いている敵には弱い。
Crazy	動き回っているロボット。中々強いが、追跡する敵には弱い。
Fire	敵の攻撃が当たるまで動かないので、動かない敵にしか勝てない。
Interactive	自分で操縦するロボット。操作方法がよく分からなくて、ほとんど勝てなかった。
RamFire	敵を追跡するロボット。７戦３敗と好成績を残している。
SittingDuck	何もしないロボット。ファイル入出力のサンプルロボット。
SpinBot	廻り続けるロボット。砲弾の威力が高いため強い、しかし、追跡されると弱い。
Target	何もしないロボット。カスタムイベントのサンプルロボット。
Tracker	敵を追跡するロボットなので結構強いが、動きが遅いため、速い敵には中々勝てない。
TrackFire	固定砲台。スキャンが速く、連射が出来る上に、砲弾の威力も高いので、強い。
Walls	壁伝いに移動するロボット。全然全勝と最強のロボットである。

3. 感　想

今回のレポートは、robocode のサンプルプログラムソースにコメントをつけるということで、
最初は、量も多くて辛いレポートだなぁ～と思ったんですが、実際にやってみると面白くて、あっという間に、
終えることが出来ました。

Java 言語は初めてなので、この robocode のプログラムソースをもう少し読んでみて、Java に慣れることが
出来たらいいなと思いました。

参考文献

- Robocode API 詳細解説
- Robocode 日本語オンラインヘルプ　http://www.geocities.co.jp/SiliconValley/9155/help/
- J2SE 5.0 API　http://java.sun.com/j2se/1.5.0/ja/docs/ja/api/