プログラミングⅡ

report1

提出日：11 月 25 日(月)

所属：工学部情報工学科

学籍番号：135713B

氏名：天願 寛之

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-a. 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;_/*static_robocode.util.Utils.normalRelativeAn
gleDegrees をインポート*/
import java.awt.*; /*java.awt.*をインポート*/


/*このロボットはコーナーに移動する。ガンを前後にスウィングし、死んだら、次のラウンドでは新しいコーナーで試みる*/
public class Corners extends Robot {

            int others; /*int 型変数 others を宣言*//*ゲームでの他のロボットの数*/

            static int corner = 0; /*静的メソッドの corner に 0 を代入,この値は全ラウンドで共有される*/
            boolean stopWhenSeeRobot = false; /*goCorner()を参照*/

            public void run() { /*メインメソッド*/
                        /*配色*/
                        setBodyColor(Color.red); /*ボディーの色:レッド*/
                        setGunColor(Color.black); /*ガンの色:ブラック*/
                        setRadarColor(Color.yellow); /*レーダーの色:イエロー*/
                        setBulletColor(Color.green); /*バレットの色:グリーン*/
                        setScanColor(Color.green); /*スキャンの色:グリーン*/

                        others = getOthers(); /*他のロボットの数を取得し others に代入*/

                        goCorner(); /*コーナーに陣取り,goCorner を参照*/

                        int gunIncrement = 3; /*ガンを回す初期スピードに 3 を代入*/

                        while (true) {
                                    for (int i = 0; i < 30; i++) { /*int 型変数 i を宣言し、0 を代入          */
                                                turnGunLeft(gunIncrement); /*ガンを(引数として得た値)3 度左回
転させる*/
                                    }                                /*処理後 i をインクリメントし,          */
                                       /*i の値が 30 未満の間この処理を続ける      */
                                       /*つまり約 90 度左回転する                  */
                                    gunIncrement *= -1; /*for 文を抜けたら gunIncrement に-1 をかける*/
                        }                       /*つまり約 90 度右回転する                  */
            }

            public void goCorner() {
      /*stopWhenSeeRobot を false にする(回転の間は他のロボットを見つけても止まらない)*/
                        stopWhenSeeRobot = false;

      /*ロボットの現在の角度を 360 度形式で取得し、静的メソッド corner からその値を差し引き*/
      /*差し引いた値の通常の相対角度分右回転させる                                  */
                        turnRight(normalRelativeAngleDegrees(corner - getHeading()));
      /*stopWhenSeeRobot を true にする(回転後他のロボットを見つけたら止まる)*/
                        stopWhenSeeRobot = true;
      /*壁に向かって 5000 進む*/
                        ahead(5000);
      /*コーナーに向かって 90 度左回転する*/
                        turnLeft(90);
      /*コーナーに向かって 5000 進む*/
```

```
                        ahead(5000);
        /*スタート位置にガンを 90 度左回転させる*/
                        turnGunLeft(90);
             }


     /*他のロボットを見つけたときのイベントメソッド e*/
             public void onScannedRobot(ScannedRobotEvent e) {
                        if (stopWhenSeeRobot) { /*stopWhenSeeRobot が true ならば以下の処理*/
           /*止まる*/
                                    stop();
           /*ロボットとの距離を取得*/
                                    smartFire(e.getDistance());
           /*他のロボットを探す*/
                                    scan();
           /*stop から再開*/
                                    resume();
                        } else {
                                    smartFire(e.getDistance()); /*stopWhenSeeRobot が false ならばロボットとの
距離を取得*/
                        }
             }


             public void smartFire(double robotDistance) { /*ロボットとの距離を引数として受け取る*/
        /*ロボットとの距離が 200 より大きい,または            */
        /*ロボットの現在のエネルギーが 15 未満ならば威力 1 の弾丸を発射*/
                        if (robotDistance > 200 || getEnergy() < 15) {
                                    fire(1);
                        }
        /*ロボットとの距離が 50 より大きければ威力 2 の弾丸を発射*/
        else if (robotDistance > 50) {
                                    fire(2);
                        }
        /*それ以外は威力 3 の弾丸を発射する*/
        else {
                                    fire(3);
                        }
             }


    /*倒された後のイベントメソッド e*/
             public void onDeath(DeathEvent e) {
                        // Well, others should never be 0, but better safe than sorry.
        /*もし他のロボットがいなければ何もしない*/
                        if (others == 0) {
                                    return;
                        }

        /*他のロボットの生存率が 75%なら corner+90 する*/
                        if ((others - getOthers()) / (double) others < .75) {
                                    corner += 90;
           /*corner が 270 になったら-90 する*/
                                    if (corner == 270) {
                                                corner = -90;
                                    }
           /*"I died and did poorly... switching corner to [corner の値]を出力*/
                                    out.println("I died and did poorly... switching corner to " + corner);
                        }
        /*"I died but did well.  I will still use corner [corner の値]"を出力*/
        else {
                                    out.println("I died but did well.  I will still use corner " + corner);
                        }
             }
}
```

1-b. Crazy.java

```
package sample;


import robocode.*; /*robocode.*をインポート*/

import java.awt.*; /*java.awt.*をインポート*/



/*このロボットは狂ったように周りを移動する*/
public class Crazy extends AdvancedRobot {
            boolean movingForward; /*真偽値を示す変数を宣言*/


            public void run() { /*メインメソッド*/
       /*配色*/
                        setBodyColor(new Color(0, 200, 0)); /*ボディーの色:new Color(0, 200, 0)*/
                        setGunColor(new Color(0, 150, 50)); /*ガンの色:new Color(0, 150, 50)*/
                        setRadarColor(new Color(0, 100, 100)); /*レーダーの色:new Color(0, 100, 100)*/
                        setBulletColor(new Color(255, 255, 100)); /*バレットの色:new Color(255, 255, 100)*/
                        setScanColor(new Color(255, 200, 200)); /*スキャンの色:new Color(255, 200, 200)*/


                        while (true) {
          /*ゲームに対しロボットが 40000 移動するよう設定する(まだ実行されない)*/
                                    setAhead(40000);
          /*movingForward に true を代入*/
                                    movingForward = true;
          /*ゲームに対し,ロボットが 90 度右回転するよう設定する(まだ実行されない)*/
                                    setTurnRight(90);
          /*90 度右回転の完了待ち*/
                                    waitFor(new TurnCompleteCondition(this));
                                    setTurnLeft(180);
          /*180 度左回転の完了待ち*/
                                    waitFor(new TurnCompleteCondition(this));
                                    setTurnRight(180);
          /*180 度右回転の完了待ち*/
                                    waitFor(new TurnCompleteCondition(this));
                        }
            }

   /*ロボットが壁にぶつかった時のイベントメソッド e*/
            public void onHitWall(HitWallEvent e) {
       /*跳ね返る*/
                        reverseDirection();
            }

            public void reverseDirection() {
                        if (movingForward) {      /*前進していた時の処理*/
                                    setBack(40000);       /*ゲームに後退するよう設定をする*/
                                    movingForward = false; /*movinfForward に false を代入*/
                        } else {                  /*前進以外の時の処理*/
                                    setAhead(40000);      /*ゲームに前進するよう設定する */
                                    movingForward = true;  /*movingForward に true を代入*/
                        }
            }

   /*他のロボットを見つけた時のイベントメソッド e*/
            public void onScannedRobot(ScannedRobotEvent e) {
                        fire(1); /*威力 1 の弾丸を発射する*/
            }

   /*ロボットにぶつかった時のイベントメソッド e*/
```

```
            public void onHitRobot(HitRobotEvent e) {
        /*もし他のロボットにぶつかったら引き返す*/
                        if (e.isMyFault()) {
                                    reverseDirection();
                        }
            }
}
```

1-c. Fire.java

```
package sample;


import robocode.HitByBulletEvent; /*robocode.HitByBulletEvent をインポート*/
import robocode.HitRobotEvent; /*robocode.HitRobotEvent 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;             /*static
robocode.util.Utils.normalRelativeAngleDegrees をインポート*/

import java.awt.*; /*java.awt.*をインポート*/


/*このロボットは座ったまま、ガンを回転させ、被弾したときに移動する*/
public class Fire extends Robot {
            int dist = 50; /*変数 dist(ロボットが弾丸にヒットしたときに動く距離)に 50 を代入*/
            public void run() { /*メインメソッド*/
                        // Set colors /*配色*/
                        setBodyColor(Color.orange); /*ボディーの色:オレンジ*/
                        setGunColor(Color.orange); /*ガンの色:オレンジ*/
                        setRadarColor(Color.red); /*レーダーの色:レッド*/
                        setScanColor(Color.red); /*スキャンの色:レッド*/
                        setBulletColor(Color.red); /*バレットの色:レッド*/

        /*ゆっくりとガンを一周させる*/
                        while (true) {
                                    turnGunRight(5); /*ガンを 5 度右回転させる*/
                        }
            }


    /*他のロボットを見つけた時のイベントメソッド e*/
            public void onScannedRobot(ScannedRobotEvent e) {
        /*他のロボットとの距離を取得し,それが 50 未満かつ,自分のエネルギーが 50 より多くあったら威力 3 の弾丸を発射*/
                        if (e.getDistance() < 50 && getEnergy() > 50) {
                                    fire(3);
                        }
        /*if 文の条件以外の時は威力 1 の弾丸を発射*/
                        else {
                                    fire(1);
                        }
        /*ガンを回転させる前に再びスキャンする*/
                        scan();
            }
    /*他のロボットの弾丸に当たった時のイベントメソッド e*/
            public void onHitByBullet(HitByBulletEvent e) {
        /*ロボットの現在の角度から敵の進行方向の角度を引いた値を 90 度から差し引いた通常の相対角度分右回転させる*/
        /*つまり弾丸が来た方向の垂直の方向にターンして少し移動する*/
                        turnRight(normalRelativeAngleDegrees(90 - (getHeading() - e.getHeading())));
        /*dist 分前進*/
```

```
                        ahead(dist);
        /*dist に-1 をかける*/
                        dist *= -1;
        /*スキャンする*/
                        scan();
             }


    /*他のロボットにぶつかった時のイベントメソッド e*/
             public void onHitRobot(HitRobotEvent e) {
        /*他のロボットと自分の相対角度と現在の角度からガンの角度を引いた通常の相対角度を turnGunmt に代入する*/
                        double  turnGunAmt  =  normalRelativeAngleDegrees(e.getBearing()  +  getHeading()  -
getGunHeading());
        /*turnGunAmt 分ガンを右回転する*/
                        turnGunRight(turnGunAmt);
        /*威力 3 の弾丸を発射する*/
                        fire(3);
             }
}
```

1-d. Interactive.java

```
package sample;


import robocode.AdvancedRobot; /*robocode.AdvancedRobot をインポート*/
import static robocode.util.Utils.normalAbsoluteAngle; /*static robocode.util.Utils.normalAbsoluteAngle をイ
ンポート*/
import static robocode.util.Utils.normalRelativeAngle; /*static robocode.util.Utils.normalRelativeAngle をイ
ンポート*/

import java.awt.*; /*java.awt.*をインポート*/
import java.awt.event.KeyEvent; /*java.awt.event.KeyEvent をインポート*/
import static java.awt.event.KeyEvent.*; /*static java.awt.event.KeyEvent.*をインポート*/
import java.awt.event.MouseEvent; /*java.awt.event.MouseEvent をインポート*/
import java.awt.event.MouseWheelEvent; /*java.awt.event.MouseWheelEvent をインポート*/



/*このロボットは、方向キーとマウスだけでコントロールされる。マウスボタンを押し続ける限り弾丸を発射し続ける。このロボッ
トのためのロボットコンソールウィンドウの "ペイント"ボタンを有効にすることにより、ロボットを目安にクロスをペイントする。
*/
public class Interactive extends AdvancedRobot {

    /*移動する方向 1 は前進,0 は居座る,-1 は後退*/
             int moveDirection;

    /*回転する方向 1 は右回転, 0 は無回転, -1 は左回転*/
             int turnDirection;
    /*移動量*/
             double moveAmount;

    /*目的の座標(x,y)*/
             int aimX, aimY;
    /*弾丸の威力 0 では発射しない*/
             int firePower;

             public void run() { /*メインメソッド*/

        /*配色,左からボディー:黒,ガン:ホワイト,レーダー:レッドを設定する*/
```

```
                        setColors(Color.BLACK, Color.WHITE, Color.RED);

                        for (;;) {
          /*ゲームに移動量×動く方向により前進するか後退するか居座るか決定するように設定する*/
                                setAhead(moveAmount * moveDirection);

          /*移動先0距離に届くまで移動量を-1ずつ減らすことで,マウスのホイールが止まったら*/
          /*ロボットが自動的に止まる                                        */
                                moveAmount = Math.max(0, moveAmount - 1);

          /*ゲームに回転する方向によりどのように回転するかしないかを決定するように設定する*/
                                setTurnRight(45 * turnDirection); // degrees

          /*現在のマウスの座標によりガンをエイムする座標に回転する*/
                                double angle = normalAbsoluteAngle(Math.atan2(aimX - getX(), aimY -
getY()));

                                setTurnGunRightRadians(normalRelativeAngle(angle                    -
getGunHeadingRadians()));

          /*弾丸の威力が0より大きい時は指定された火力で発射するように設定する*/
                                if (firePower > 0) {
                                        setFire(firePower);
                                }

          /*保留していた全ての設定を実行*/
                                execute();

                        }
           }

   /*キーが押されたときの呼び出しイベントメソッドe*/
            public void onKeyPressed(KeyEvent e) {
       /*e.getKeyCode()の値によって処理を変える*/
                        switch (e.getKeyCode()) {
                        case VK_UP:
                        case VK_W:
          /*上矢印キー:移動方向に前進*/
                                moveDirection = 1;
                                moveAmount = Double.POSITIVE_INFINITY;
                                break;

                        case VK_DOWN:
                        case VK_S:
          /*下矢印キー:移動方向に後退*/
                                moveDirection = -1;
                                moveAmount = Double.POSITIVE_INFINITY;
                                break;

                        case VK_RIGHT:
                        case VK_D:
          /*右矢印キー:右回転*/
                                turnDirection = 1;
                                break;

                        case VK_LEFT:
                        case VK_A:
          /*左矢印キー:左回転*/
                                turnDirection = -1;
                                break;
                        }
           }

   /*キーを離した時の呼び出しイベントメソッドe*/
            public void onKeyReleased(KeyEvent e) {
```

```
    /*e.getKeyCode()の値によって処理を変える*/
                    switch (e.getKeyCode()) {
                    case VK_UP:
                    case VK_W:
                    case VK_DOWN:
                    case VK_S:
        /*上下矢印キー:移動方向を向いて居座る*/
                                moveDirection = 0;
                                moveAmount = 0;
                                break;

                    case VK_RIGHT:
                    case VK_D:
                    case VK_LEFT:
                    case VK_A:
        /*左右矢印キー:回転方向を向いて止まる*/
                                turnDirection = 0;
                                break;
                    }
         }

/*マウスのホイールが回転した時の呼び出しイベントメソッドe*/
         public void onMouseWheelMoved(MouseWheelEvent e) {
    /*ホイールが前に移動したら前進し,そうでなければ後退する*/
                    moveDirection = (e.getWheelRotation() < 0) ? 1 : -1;

    /*ホイールの1回転毎に5ピクセル分だけ移動する,高い値ほど速く移動する*/
                    moveAmount += Math.abs(e.getWheelRotation()) * 5;
         }

/*マウスが移動した時の呼び出しイベントメソッドe*/
         public void onMouseMoved(MouseEvent e) {
    /*エイムの座標をマウスのポインタの座標に設定する*/
                    aimX = e.getX();
                    aimY = e.getY();
         }

/*マウスボタンを押した時の呼び出しイベントメソッドe*/
         public void onMousePressed(MouseEvent e) {
                    if (e.getButton() == MouseEvent.BUTTON3) {
        /*ボタン3:威力3の弾丸を発射,弾丸の色はレッド*/
                                firePower = 3;
                                setBulletColor(Color.RED);
                    } else if (e.getButton() == MouseEvent.BUTTON2) {
        /*ボタン2:威力2の弾丸を発射,弾丸の色はオレンジ*/
                                firePower = 2;
                                setBulletColor(Color.ORANGE);
                    } else {
        /*ボタン1か想定外のボタン:威力1の弾丸を発射,弾丸の色はイエロー*/
                                firePower = 1;
                                setBulletColor(Color.YELLOW);
                    }
         }
/*マウスボタンを離した時の呼び出しイベントe*/
         public void onMouseReleased(MouseEvent e) {
    /*弾丸の威力は0,つまり撃たない*/
                    firePower = 0;
         }

/*"ペイント"でこのロボットをウインドウでペイントすることが出来る*/
         public void onPaint(Graphics2D g) {
    /*現在の目的である座標(x,y)を中心としたレッドクロスを描く*/
                    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-e. Interactive_v2.java

```
package sample;


import robocode.AdvancedRobot; /*robocode.AdvancedRobot をインポート*/
import robocode.util.Utils; /*robocode.util.Utils をインポート*/
import static robocode.util.Utils.normalAbsoluteAngle; /*static robocode.util.Utils.normalAbsoluteAngle をインポート*/
import static robocode.util.Utils.normalRelativeAngle; /*static robocode.util.Utils.normalRelativeAngle をインポート*/

import java.awt.*; /*java.awt.*をインポート*/
import java.awt.event.KeyEvent; /*java.awt.event.KeyEvent をインポート*/
import static java.awt.event.KeyEvent.*; /*static java.awt.event.KeyEvent.*をインポート*/
import java.awt.event.MouseEvent; /*java.awt.event.MouseEvent をインポート*/
import java.awt.event.MouseWheelEvent; /*java.awt.event.MouseWheelEvent をインポート*/
import java.util.HashSet; /*java.util.HashSet をインポート*/
import java.util.Set; /*java.util.Set をインポート*/



/*このロボットは方向キーとマウスだけでコントロールされる。マウスボタンを押し続ける限り弾丸を発射し続ける。このロボットのためのロボットコンソールウィンドウの "ペイント"ボタンを有効にすることにより、ロボットを目安にクロスをペイントする。
*/
public class Interactive_v2 extends AdvancedRobot {

    /*目的の座標(x,y)*/
            int aimX, aimY;

    /*弾丸の威力 0 では発射しない*/
            int firePower;

    /*スクリーン上での方向*/
            private enum Direction {
                        UP,
                        DOWN,
                        LEFT,
                        RIGHT
            }

    /*現在の移動方向*/
            private final Set<Direction> directions = new HashSet<Direction>();

            public void run() { /*メインメソッド*/

        /*配色,左からボディー:黒,ガン:ホワイト,レーダー:レッドを設定する*/
                        setColors(Color.BLACK, Color.WHITE, Color.RED);

                        for (;;) {
            /*移動する距離を比較してロボットを移動するように設定する*/
                                    setAhead(distanceToMove());

            /*ボディーを回転させ正しい方向を指すように設定する*/
                                    setTurnRight(angleToTurnInDegrees());
```

```
            /*現在のマウスの座標により現在のガンのエイム座標を回転させコントロールする*/
                                    double angle = normalAbsoluteAngle(Math.atan2(aimX - getX(), aimY -
getY()));

                                    setTurnGunRightRadians(normalRelativeAngle(angle                    -
getGunHeadingRadians()));

            /*弾丸の威力が 0 より大きい時は指定された火力で発射するように設定する*/
                                    if (firePower > 0) {
                                              setFire(firePower);
                                    }

            /*保留していた全ての設定を実行*/
                                    execute();

                          }
             }
    /*キーが押されたときの呼び出しイベントメソッド e*/
              public void onKeyPressed(KeyEvent e) {
        /*e.getKeyCode()の値によって処理を変える*/
                          switch (e.getKeyCode()) {
                          case VK_UP:
                          case VK_W:
            /*上矢印キー:前進*/
                                    directions.add(Direction.UP);
                                    break;

                          case VK_DOWN:
                          case VK_S:
            /*下矢印キー:後退*/
                                    directions.add(Direction.DOWN);
                                    break;

                          case VK_RIGHT:
                          case VK_D:
            /*右矢印キー:右移動*/
                                    directions.add(Direction.RIGHT);
                                    break;

                          case VK_LEFT:
                          case VK_A:
            /*左矢印キー:左移動*/
                                    directions.add(Direction.LEFT);
                                    break;
                          }
              }

    /*キーを離した時の呼び出しイベントメソッド e*/
              public void onKeyReleased(KeyEvent e) {
        /*e.getKeyCode()の値によって処理を変える*/
                          switch (e.getKeyCode()) {
                          case VK_UP:
                          case VK_W:
            /*上矢印キー:止まる*/
                                    directions.remove(Direction.UP);
                                    break;

                          case VK_DOWN:
                          case VK_S:
            /*下矢印キー:止まる*/
                                    directions.remove(Direction.DOWN);
                                    break;

                          case VK_RIGHT:
                          case VK_D:
```

```
        /*右矢印キー:止まる*/
                                directions.remove(Direction.RIGHT);
                                break;

                    case VK_LEFT:
                    case VK_A:
        /*左矢印キー:止まる*/
                                directions.remove(Direction.LEFT);
                                break;
                    }
         }

/*マウスのホイールが回転した時の呼び出しイベントメソッドe*/
         public void onMouseWheelMoved(MouseWheelEvent e) { /*何もしない*/
         }

/*マウスが移動した時の呼び出しイベントメソッドe*/
         public void onMouseMoved(MouseEvent e) {
    /*エイムの座標をマウスのポインタの座標に設定する*/
                    aimX = e.getX();
                    aimY = e.getY();
         }

/*マウスボタンを押した時の呼び出しイベントメソッドe*/
         public void onMousePressed(MouseEvent e) {
                    if (e.getButton() == MouseEvent.BUTTON3) {
        /*ボタン3:威力3の弾丸を発射,弾丸の色はレッド*/
                                firePower = 3;
                                setBulletColor(Color.RED);
                    } else if (e.getButton() == MouseEvent.BUTTON2) {
        /*ボタン2:威力2の弾丸を発射,弾丸の色はオレンジ*/
                                firePower = 2;
                                setBulletColor(Color.ORANGE);
                    } else {
        /*ボタン1か想定外のボタン:威力1の弾丸を発射,弾丸の色はイエロー*/
                                firePower = 1;
                                setBulletColor(Color.YELLOW);
                    }
         }

/*マウスボタンを離した時の呼び出しイベントメソッドe*/
         public void onMouseReleased(MouseEvent e) {
    /*弾丸の威力は0,つまり撃たない*/
                    firePower = 0;
         }

/*"ペイント"でこのロボットをウインドウでペイントすることが出来る*/
         public void onPaint(Graphics2D g) {
    /*現在の目的である座標(x,y)を中心としたレッドクロスを描く*/
                    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);
         }

/*現在のロボットの方向と望んだ方向の間のデルタ部分の角度にターンする為の角度を返す*/
         private double angleToTurnInDegrees() {
                    if (directions.isEmpty()) {
                                return 0;
                    }
                    return Utils.normalRelativeAngleDegrees(desiredDirection() - getHeading());
         }

/*移動距離を返す*/
         private double distanceToMove() {
```

```
        /*もし何もキーが打たれなかったら移動するべきではない*/
                        if (directions.isEmpty()) {
                                return 0;
                        }
        /*もしロボットが 45 度以上回転したら 5 ピクセルだけ移動する*/
                        if (Math.abs(angleToTurnInDegrees()) > 45) {
                                return 5;
                        }
        /*そうでなければ,フルスピードで移動する*/
                        return Double.POSITIVE_INFINITY;
                }

    /*未決定の移動方向に応じて移動したい方向を返す*/
    /*1 つのキーを押しながらの移動,上,右,下,左*/
    /*2 つのキーを押しながらの移動,北東,北西,南東,南西*/
                private double desiredDirection() {
                        if (directions.contains(Direction.UP)) {
                                if (directions.contains(Direction.RIGHT)) {
                                        return 45;
                                }
                                if (directions.contains(Direction.LEFT)) {
                                        return 315;
                                }
                                return 0;
                        }
                        if (directions.contains(Direction.DOWN)) {
                                if (directions.contains(Direction.RIGHT)) {
                                        return 135;
                                }
                                if (directions.contains(Direction.LEFT)) {
                                        return 225;
                                }
                                return 180;
                        }
                        if (directions.contains(Direction.RIGHT)) {
                                return 90;
                        }
                        if (directions.contains(Direction.LEFT)) {
                                return 270;
                        }
                        return 0;
                }
}
```

1-f. MyFirstjuniorRobot.java

```
package sample;


import robocode.JuniorRobot;



/*シーソーのように移動し、他のロボットを見つける事が出来なかったらガンを振り回す。他のロボットを見つけたら即時にガンを回転させ、それに向かって弾丸を発射する。*/
public class MyFirstJuniorRobot extends JuniorRobot {

            public void run() { /*メインメソッド*/
       /*配色,左からボディー:グリーン,ガン:ブラック,レーダー:ブルー*/
                        setColors(green, black, blue);

                        while (true) {
                                    ahead(100); /*100 前進*/
                                    turnGunRight(360); /*360 度ガンを回転*/
                                    back(100); /*100 後退*/
                                    turnGunRight(360); /*360 度ガンを回転*/
                        }
            }
   /*他のロボットを見つけたときのメソッド*/
            public void onScannedRobot() {
       /*他のロボットを見つけた場所へガンを回転*/
                        turnGunTo(scannedAngle);

       /*威力 1 の弾丸を発射*/
                        fire(1);
            }

   /*他のロボットの弾丸に被弾したときのメソッド*/
            public void onHitByBullet() {
       /*飛んできた弾丸に対して垂直に左回転しながら 100 前進*/
                        turnAheadLeft(100, 90 - hitByBulletBearing);
            }
}
```

1-g. paintingRobot.java

```
package sample;


import robocode.HitByBulletEvent; /*robocode.HitByBulletEvent 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 java.awt.*; /*java.awt.*をインポート*/



/*シーソーのように移動し、ガンを振り回す。このロボットがペイントすることが出来るとき、このロボットの周りに赤い円がペイントされる*/
public class PaintingRobot extends Robot {

	public void run() {/*メインメソッド*/
		while (true) {
			ahead(100); /*100 前進*/
			turnGunRight(360); /*360 度回転*/
			back(100); /*100 後退*/
			turnGunRight(360); /*360 度回転*/
		}
	}

   /*他のロボットを見つけたときのイベントメソッド e*/
	public void onScannedRobot(ScannedRobotEvent e) {
      /*ロボットダイアログ上のデバッグプロパティの機能を示す*/
		setDebugProperty("lastScannedRobot", e.getName() + " at " + e.getBearing() + " degrees at time " + getTime());
		/*威力 1 の弾丸を発射*/
		fire(1);
	}

   /*他のロボットの弾丸に被弾したときのイベントメソッド e 被弾したときにオレンジ色の円を描く*/
	public void onHitByBullet(HitByBulletEvent e) {
      /*ロボットダイアログ上のデバッグプロパティの機能を示す*/
		setDebugProperty("lastHitBy", e.getName() + " with power of bullet " + e.getPower() + " at time " + getTime());

      /*デバッグプロパティを削除する方法を見せる*/
		setDebugProperty("lastScannedRobot", null);

      /*戦闘表示をペイントすることによってデバッグする*/
		Graphics2D g = getGraphics();

		g.setColor(Color.orange);
		g.drawOval((int) (getX() - 55), (int) (getY() - 55), 110, 110);
		g.drawOval((int) (getX() - 56), (int) (getY() - 56), 112, 112);
		g.drawOval((int) (getX() - 59), (int) (getY() - 59), 118, 118);
		g.drawOval((int) (getX() - 60), (int) (getY() - 60), 120, 120);

		turnLeft(90 - e.getBearing());
	}

   /*私たちのロボットの回りに赤い円を描くメソッド*/
	public void onPaint(Graphics2D g) {
		g.setColor(Color.red);
		g.drawOval((int) (getX() - 50), (int) (getY() - 50), 100, 100);
		g.setColor(new Color(0, 0xFF, 0, 30));
		g.fillOval((int) (getX() - 60), (int) (getY() - 60), 120, 120);
	}
}
```

1-h. RamFire.java

```
package sample;


import robocode.HitRobotEvent; /*robocode.HitRobotEvent 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 java.awt.*; /*java.awt.*をインポート*/



/*このロボットは敵にタックルをかますために機動します。敵にぶつかったら弾丸を発射します。*/
public class RamFire extends Robot {
    int turnDirection = 1; /*時計回り,または反時計回りに回転し,ターゲットを探す*/
                           /*int 型変数 turnDirection に 1 を代入*/
    /*回転してターゲットを探す*/
    public void run() {  /*メインメソッド*/
                // Set colors                       /*色の指定*/
                setBodyColor(Color.lightGray);       /*ボディーの色：ライトグレー*/
                setGunColor(Color.gray);             /*砲塔の色：グレー          */
                setRadarColor(Color.darkGray);       /*レーダーの色：ダークグレー*/

                while (true) {/*起動直後にロボットを右回転させる,turnRight の引数の値で回転度数を指定する*/
                        turnRight(5 * turnDirection);/*引数は(5*1)より while 文で 5 度回転をループさせる*/
                        }
            }

    /*他のロボットを発見した時のイベントメソッド e 見つけたらタックルをかまします*/
    public void onScannedRobot(ScannedRobotEvent e) {

            if (e.getBearing() >= 0) {  /*ターゲットと自分の相対角度を比較する(車体の向きが基準 0 度)*/
                                turnDirection = 1;  /*ターゲットとの相対角度が 0 度以上(右方向)なら 1 を代入*/
                        } else {
                                turnDirection = -1; /*ターゲットとの相対角度が 0 度以下(左方向)なら-1 を代入
*/
                        }

                        turnRight(e.getBearing()); /*ターゲットの方向へ回転*/
                        ahead(e.getDistance() + 5); /*ターゲットと自分との距離+5 前進*/
                        scan(); /*再び前進するかもしれないよ?*/
            }

    /*他のロボットにぶつかったときのイベントメソッド e ロボット正面を回転、激しく弾丸を発射しながら再びタックルをかまし
ます*/
            public void onHitRobot(HitRobotEvent e) {
                        if (e.getBearing() >= 0) { /*ターゲットと自分の相対角度を比較する(車体の向きが基準 0 度)*/
                                turnDirection = 1;  /*ターゲットとの相対角度が 0 度以上(右方向)なら 1 を代入*/
                        } else {
                                turnDirection = -1;  /*ターゲットとの相対角度が 0 度以下(左方向)なら-1 を代入
*/
                        }
                        turnRight(e.getBearing());  /*ターゲットの方向へ回転*/

       /*ロボットを殺さないショットを決めて、ボーナスポイントの代わりにタックルをかましたいね*/
                        if (e.getEnergy() > 16) { /*ロボットの現在のエネルギーを取得し,16 以上なら*/
                                fire(3);             /*パワー3 の弾丸を発射する*/
                        } else if (e.getEnergy() > 10) { /*ロボットの現在のエネルギーを取得し,16 以上なら*/
                                fire(2);                    /*パワー2 の弾丸を発射する*/
                        } else if (e.getEnergy() > 4) { /*ロボットの現在のエネルギーを取得し,4 以上なら*/
                                fire(1);                    /*パワー1 の弾丸を発射する*/
                        } else if (e.getEnergy() > 2) { /*ロボットの現在のエネルギーを取得し,2 以上なら*/
                                fire(.5);                   /*パワー0.5 の弾丸を発射する*/
                        } else if (e.getEnergy() > .4) { /*ロボットの現在のエネルギーを取得し,0.4 以上なら*/
```

```
                                fire(.1);                       /*パワー0.1の弾丸を発射する*/
                        }
                        ahead(40); /*40前進 再度タックル!!!!*/
                }
}
```

1-i. SittingDuck.java

```
package sample;


import robocode.AdvancedRobot; /*robocode.AdvancedRobotをインポート*/
import robocode.RobocodeFileOutputStream; /*robocode.RobocodeFileOutputStreamをインポート*/

import java.awt.*; /*java.awt.*をインポート*/
import java.io.BufferedReader; /*java.io.BufferedReaderをインポート*/
import java.io.FileReader; /*java.io.FileReaderをインポート*/
import java.io.IOException; /*java.io.IOExceptionをインポート*/
import java.io.PrintStream; /*java.io.PrintStreamをインポート*/



/*このロボットは座ったまま何もしない.このロボットは固執を示す*/
public class SittingDuck extends AdvancedRobot {
            static boolean incrementedBattles = false;

            public void run() { /*メインメソッド*/
       /*配色,ボディー:イエロー,ガン:イエロー*/
                        setBodyColor(Color.yellow);
                        setGunColor(Color.yellow);

                        int roundCount, battleCount;

                        try {
                                    BufferedReader reader = null;
                                    try {
              /*ラウンドカウントとバトルカウントの2行を含むファイル"count.dat"を読む。*/
                                                reader          =          new          BufferedReader(new
FileReader(getDataFile("count.dat")));

              /*カウントの取得を試みる*/
                                                roundCount = Integer.parseInt(reader.readLine());
                                                battleCount = Integer.parseInt(reader.readLine());

                                    } finally {
                                                if (reader != null) {
                                                            reader.close();
                                                }
                                    }
                        } catch (IOException e) {
          /*IOExceptionファイルの読み込みが悪かった場合、カウントを0にリセット*/
                                    roundCount = 0;
                                    battleCount = 0;
                        } catch (NumberFormatException e) {
           /*NumberFormatExceptionファイルの読み込みが悪かった場合、カウントを0にリセット*/
                                    roundCount = 0;
                                    battleCount = 0;
                        }

       /*ラウンドの#をインクリメント*/
```

```
                        roundCount++;

        /*先刻のバトルの#をインクリメントしなかったら...
         注釈:ロボットは一回毎のバトルでインスタンス化しただけなので、メンバ変数は依然として有効*/
                        if (!incrementedBattles) {
           /*バトルの#をインクリメント*/
                                battleCount++;
                                incrementedBattles = true;
                        }

                        PrintStream w = null;
                        try {
                                w                =                new                PrintStream(new
RobocodeFileOutputStream(getDataFile("count.dat")));

                                w.println(roundCount);
                                w.println(battleCount);

           /*PrintStreams はプリント中に IOExceptions を投げない,彼らは単純にフラグを設定するのでここでそれをチェッ
クしてください*/
                                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-j. SpinBot.java

```
package sample;


import robocode.AdvancedRobot; /*robocode.AdvancedRobot をインポート*/
import robocode.HitRobotEvent; /*robocode.HitRobotEvent をインポート*/
import robocode.ScannedRobotEvent; /*robocode.ScannedRobotEvent をインポート*/

import java.awt.*; /*java.awt.*をインポート*/



/*このロボットは円で動く。敵を見つけた時弾丸を発射する*/
public class SpinBot extends AdvancedRobot {

            public void run() { /*メインメソッド*/
                        /*配色,ボディー:ブルー,ガン:ブルー,レーダー:ブラック,スキャン:イエロー*/
                        setBodyColor(Color.blue);
                        setGunColor(Color.blue);
                        setRadarColor(Color.black);
                        setScanColor(Color.yellow);

                        while (true) {
           /*ロボットが移動するときゲームに設定しておく*/
           /*また、このロボットはたくさん回転するだろう*/
                                    setTurnRight(10000);
           /*このこのロボットのスピードを 5 に制限する*/
                                    setMaxVelocity(5);
           /*動き始める、そして回り始める*/
                                    ahead(10000);
           /*繰り返し*/
                        }
            }

   /*他のロボットを見つけた時のイベントメソッド e  激しくファイヤー*/
            public void onScannedRobot(ScannedRobotEvent e) {
                        fire(3);
            }

   /*他のロボットにぶつかった時のイベントメソッド e  動きが止まるだろうから回転して脱出*/
            public void onHitRobot(HitRobotEvent e) {
                        if (e.getBearing() > -10 && e.getBearing() < 10) {
                                    fire(3);
                        }
                        if (e.isMyFault()) {
                                    turnRight(10);
                        }
            }
}
```

1-k. Target.java

```
package sample;


import robocode.AdvancedRobot; /*robocode.AdvancedRobot をインポート*/
import robocode.Condition; /*robocode.Condition をインポート*/
import robocode.CustomEvent; /*robocode.CustomEvent をインポート*/

import java.awt.*; /*java.awt.*をインポート*/



/*このロボットは居座り続ける。エネルギーが 20 落ちると毎回移動する。このロボットはカスタムイベントを示す*/
public class Target extends AdvancedRobot {

            int trigger; /*移動したときの軌跡を保持する*/

            public void run() { /*メインメソッド*/
                        // Set colors/*配色,ボディー:ホワイト,ガン:ホワイト,レーダー:ホワイト*/
                        setBodyColor(Color.white);
                        setGunColor(Color.white);
                        setRadarColor(Color.white);

       /*最初、このロボットのライフが 80 になったら移動するだろう*/
                        trigger = 80;
       /*"trigger hit"という名のカスタムイベントを加える*/
                        addCustomEvent(new Condition("triggerhit") {
                                    public boolean test() {
                                                return (getEnergy() <= trigger);
                                    }
                        });
            }

   /*カスタムした時のイベントメソッド e*/
            public void onCustomEvent(CustomEvent e) {
       /*もしカスタムイベント"triggerhit"がコースアウトしたら*/
                        if (e.getCondition().getName().equals("triggerhit")) {
          /*トリガーの値を調整したり、イベントが何度も起動します*/
                                    trigger -= 20;
                                    out.println("Ouch, down to " + (int) (getEnergy() + .5) + " energy.");
          /*少し周りを移動する*/
                                    turnLeft(65);
                                    ahead(100);
                        }
            }
}
```

1-l. TrackFire.java

```
package sample;


import robocode.Robot; /* robocode.Robot をインポート*/
import robocode.ScannedRobotEvent; /* robocode.ScannedRobotEvent をインポート*/
import robocode.WinEvent; /* robocode.WinEvent をインポート*/
import_static_robocode.util.Utils.normalRelativeAngleDegrees;_/*static_robocode.util.Utils.normalRelativeAn
gleDegrees;をインポート*/

import java.awt.*; /* java.awt.をインポート*/



/*このロボットは出現場所に居座り続けます。そのまま最寄りに見つけたロボットを狙い撃ちします。*/
public class TrackFire extends Robot {

            public void run() { /*メインメソッド*/
                        /*配色, ボディー:ピンク, ガン:ピンク ,レーダー:ピンク ,スキャン:ピンク ,バレット:ピンク*/
                        setBodyColor(Color.pink);
                        setGunColor(Color.pink);
                        setRadarColor(Color.pink);
                        setScanColor(Color.pink);
                        setBulletColor(Color.pink);

                        while (true) {
                                    turnGunRight(10); /*ガンを回転して自動でスキャン*/
                        }
            }

    /*ロボットを見つけた時のイベントメソッドe "ターゲットを発見,狩りの時間じゃいッ!!"*/
            public void onScannedRobot(ScannedRobotEvent e) {
        /*ロボットの正確な位置を計算*/
                        double absoluteBearing = getHeading() + e.getBearing();
                        double     bearingFromGun     =     normalRelativeAngleDegrees(absoluteBearing     -
getGunHeading());

        /*十分近ければファイヤー!!*/
                        if (Math.abs(bearingFromGun) <= 3) {
                                    turnGunRight(bearingFromGun);
                                    /*fire()を呼び出すと、他のロボットの軌跡を失う原因となるターンを使用しているの
でここでガンの熱を確認しましょう*/
                                    if (getGunHeat() == 0) {
                                                fire(Math.min(3   -   Math.abs(bearingFromGun),   getEnergy()
- .1));
                                    }
                        } /*そうでない場合は有効にガンを設定*/
                        /*これはscan()を呼び出している間効果は持てない*/
                        else {
                                    turnGunRight(bearingFromGun);
                        }
        /*もしロボットを見つけたら他のスキャンイベントを生成*/
        /*ガンとレーダーがターンしていないならスキャンイベントを呼び出す必要があるだけで、それ以外ならスキャンイベント
は自動的に呼び出される*/
                        if (bearingFromGun == 0) {
                                    scan();
                        }
            }
    /*勝利した時のイベントメソッドe "勝利の舞"*/
            public void onWin(WinEvent e) {
                        turnRight(36000);
            }
}
```

1-m. Tracker.java

```
package sample;


import robocode.HitRobotEvent; /*robocode.HitRobotEvent 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 robocode.WinEvent; /*robocode.WinEvent をインポート*/
import             static             robocode.util.Utils.normalRelativeAngleDegrees;             /*static
robocode.util.Utils.normalRelativeAngleDegrees をインポート*/

import java.awt.*; /*java.awt.*をインポート*/



/*1 つのロボットに狙いを定めて、間をつめて、近づいたら弾丸を発射*/
public class Tracker extends Robot {
            int count = 0; /*ターゲットをサーチしていた時間の長さの軌跡を保持*/

            double gunTurnAmt; /*サーチした時いくらかガンを回転させる*/

            String trackName; /*現在追跡しているロボットの名前*/


            public void run() { /*メインメソッド*/
                        // Set colors/*配色,ボディー:new Color(128, 128, 50),ガン:new Color(50, 50, 20),レーダ
ー:new Color(200, 200, 70),スキャン:ホワイト,バレット:ブルー*/
                        setBodyColor(new Color(128, 128, 50));
                        setGunColor(new Color(50, 50, 20));
                        setRadarColor(new Color(200, 200, 70));
                        setScanColor(Color.white);
                        setBulletColor(Color.blue);

       /*ガンを準備*/
                        trackName = null; /*誰も追跡していないと初期化*/

                        setAdjustGunForRobotTurn(true); /*ターンする時ガンを固定*/

                        gunTurnAmt = 10; /*ガンのターンを 10 に初期化*/

                        while (true) {
           /*ガンをターンさせて敵を捜す*/
                                    turnGunRight(gunTurnAmt);
           /*ロボットが捜した時間の軌跡を保持*/
                                    count++;
           /*2 ターンまでにターゲットを見つけられなかったら、左を見る*/
                                    if (count > 2) {
                                                gunTurnAmt = -10;
                                    }
           /*5 ターンまでにターゲットを見つけられなかったら、右を見る*/
                                    if (count > 5) {
                                                gunTurnAmt = 10;
                                    }
           /*10 ターン後にまだターゲットを見つけられなかったら他のターゲットを見つける*/
                                    if (count > 11) {
                                                trackName = null;
                                    }
                        }
            }

   /*他のロボットを見つけた時のイベントメソッド e "この場所はよいものだ"*/
            public void onScannedRobot(ScannedRobotEvent e) {

       /*もしターゲットが定まっていても、他を見つけたら、即時に更なるスキャンイベントをゲットだぜ*/
```

```
                        if (trackName != null && !e.getName().equals(trackName)) {
                                return;
                        }

        /*もし、ターゲットを持っていなかったら、まぁ、今やっちまおう*/
                        if (trackName == null) {
                                trackName = e.getName();
                                out.println("Tracking " + trackName);
                        }
        /*これはターゲット。カウントをリセット(メインメソッド参照)*/
                        count = 0;
        /*もしターゲットが遠くにいたら、そいつに対してターンして移動*/
                        if (e.getDistance() > 150) {
                                gunTurnAmt = normalRelativeAngleDegrees(e.getBearing() + (getHeading() -
getRadarHeading()));

                                turnGunRight(gunTurnAmt);
            /*これらを setTurnGunRight に変えようや*/
                                turnRight(e.getBearing());
            /*Tracker をいくら改善するか参照してください*/
            /*(君は Tracker を高度なロボットにしなければならないだろう)*/
                                ahead(e.getDistance() - 140);
                                return;
                        }

        /*ターゲットは近い!!撃てっ！*/
                        gunTurnAmt   =   normalRelativeAngleDegrees(e.getBearing()   +   (getHeading()   -
getRadarHeading()));
                        turnGunRight(gunTurnAmt);
                        fire(3);

        /*何っ!!敵はかなり近いぞ!!さがれッッッッ*/
                        if (e.getDistance() < 100) {
                                if (e.getBearing() > -90 && e.getBearing() <= 90) {
                                        back(40);
                                } else {
                                        ahead(40);
                                }
                        }
                        scan();
            }

    /*他のロボットにぶつかった時のイベントメソッド e "こいつやっちまおう"*/
            public void onHitRobot(HitRobotEvent e) {
        /*もしこいつがまだターゲットではなかったら print するだけ*/
                        if (trackName != null && !trackName.equals(e.getName())) {
                                out.println("Tracking " + e.getName() + " due to collision");
                        }
        /*ターゲットを設定*/
                        trackName = e.getName();
        /*少しさがります。我々はこの行動の間スキャンイベントを取得することはない。*/
        /*高度なロボットは setBack()と execute()を使うだろう*/
                        gunTurnAmt   =   normalRelativeAngleDegrees(e.getBearing()   +   (getHeading()   -
getRadarHeading()));
                        turnGunRight(gunTurnAmt);
                        fire(3);
                        back(50);
            }

    /*勝利した時のイベントメソッド e "いざ勝利の舞!!!!"*/
            public void onWin(WinEvent e) {
                        for (int i = 0; i < 50; i++) {
                                turnRight(30);
                                turnLeft(30);
                        }
```

```
        }
}
```

1-n. VelociRobot.java

```
package sample;


import robocode.HitByBulletEvent; /*robocode.HitByBulletEvent をインポート*/
import robocode.HitWallEvent; /*robocode.HitWallEvent をインポート*/
import robocode.RateControlRobot; /*robocode.RateControlRobot をインポート*/
import robocode.ScannedRobotEvent; /*robocode.ScannedRobotEvent をインポート*/


/*このロボットは RateControlRobot クラスを使っているよ*/
public class VelociRobot extends RateControlRobot {

        int turnCounter;
        public void run() {

                turnCounter = 0;
                setGunRotationRate(15);

                while (true) {
                        if (turnCounter % 64 == 0) {
                                /*もし被弾したらターンして、まっすぐに正す*/
                                setTurnRate(0);
                                /*4 の速度で前進*/
                                setVelocityRate(4);
                        }
                        if (turnCounter % 64 == 32) {
                                /*素早く後退*/
                                setVelocityRate(-6);
                        }
                        turnCounter++;
                        execute();
                }
        }

   /*他のロボットを発見した時のイベントメソッド e  "ファイヤー!!!"*/
        public void onScannedRobot(ScannedRobotEvent e) {
                fire(1);
        }

   /*被弾した時のイベントメソッド e*/
        public void onHitByBullet(HitByBulletEvent e) {
                /*他のロボットを混乱させるような華麗なるターン*/
                setTurnRate(5);
        }

        /*壁にぶつかった時のイベントメソッド e*/
        public void onHitWall(HitWallEvent e) {
                /*壁から離れましょう*/
                setVelocityRate(-1 * getVelocityRate());
        }
}
```

1-o. Walls.java

```
package sample;


import robocode.HitRobotEvent; /*robocode.HitRobotEvent 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 java.awt.*; /*java.awt.*をインポート*/



/*このロボットはガンが直面している外縁を移動します*/
public class Walls extends Robot {

            boolean peek; /*ロボットがいる場所ではターンしない*/
            double moveAmount; /*移動する値*/


            public void run() { /*メインメソッド*/
                        /*配色，ボディー:ブラック，ガン:ブラック，レーダー:オレンジ，バレット:シアン，スキャン:シ
アン*/
                        setBodyColor(Color.black);
                        setGunColor(Color.black);
                        setRadarColor(Color.orange);
                        setBulletColor(Color.cyan);
                        setScanColor(Color.cyan);

                        /*moveAmount をこのバトルフィールドで可能な限り最大に初期化*/
                        moveAmount = Math.max(getBattleFieldWidth(), getBattleFieldHeight());
                        /*peek の初期値を false に*/
                        peek = false;

       /*壁に向けて左ターン*/
       /*"getHeading() % 90"は 90 で getHeading()の残りを分けるという意味*/
                        turnLeft(getHeading() % 90);
                        ahead(moveAmount);
                        /*ガンを 90 度右ターン*/
                        peek = true;
                        turnGunRight(90);
                        turnRight(90);

                        while (true) {
                                    /*ahead()を完了したときターンする前に見なさい*/
                                    peek = true;
          /*壁に移動*/
                                    ahead(moveAmount);
                                    /*今見んな*/
                                    peek = false;
                                    /*次の壁にターン*/
                                    turnRight(90);
                        }
            }

   /*他のロボットにぶつかった時のイベントメソッド e "少し離れよう"*/
            public void onHitRobot(HitRobotEvent e) {
        /*もし私たちの正面にいたら少し後退します*/
                        if (e.getBearing() > -90 && e.getBearing() < 90) {
                                    back(100);
                        } /*私たちの後ろにいたら少し前進します*/
                        else {
                                    ahead(100);
                        }
            }
```

```
    /*他のロボットを見つけた時のイベントメソッドe "ファイヤー!!!"*/
            public void onScannedRobot(ScannedRobotEvent e) {
                        fire(2);
                        // Note that scan is called automatically when the robot is moving.
                        // By calling it manually here, we make sure we generate another scan event if there's
a robot on the next
                        // wall, so that we do not start moving up it until it's gone
        /*ロボットが移動している時はスキャンは自動的に呼び出されます。*/
        /*手動で呼び出す事によって、次の壁にロボットがいたら別のスキャンイベントを生成することを確認、それが行われるま
で移動を開始しない*/
                        if (peek) {
                                    scan();
                        }
            }
}
```

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-a. 各ロボット同士のタイマンによる戦闘結果

	Co	Cr	Fi	In	In2	MF	Pa	Ra	SD	SB	Ta	TF	Tr	VR	Wa
Co		○	●	●	●	●	●	○	○	●	○	●	●	○	●
Cz	●		●	●	●	●	●	●	○	●	○	●	●	●	●
Fi	○	○		○	○	●	●	○	○	●	○	●	●	●	●
In	○	○	●		D	●	○	●	○	○	○	○	●	○	●
In2	○	○	●	D		○	○	●	○	○	○	○	○	○	●
MF	○	○	○	○	●		○	●	○	○	○	●	●	○	●
Pa	○	○	○	●	●	●		●	○	●	○	○	●	●	●
Ra	●	○	●	○	○	○	○		○	●	○	●	●	○	●
SD	●	●	●	●	●	●	●	●		●	D	●	●	●	●
SB	○	○	○	●	●	●	○	○	○		○	○	○	●	●
Ta	●	●	●	●	●	●	●	●	D	●		●	●	●	●
TF	○	○	○	●	●	○	●	○	○	●	○		○	○	●
Tr	○	○	○	○	●	○	○	○	○	●	○	●		●	●
VR	●	○	○	●	●	●	○	●	○	○	○	●	○		●
Wa	○	○	○	○	○	○	○	○	○	○	○	○	○	○	

ルール

ラウンド数 ___3 回

形式___タイマン

フィールド___800×600

Interactive&Interactive_v2 は主が操作

なお Interactive と Interactive_v2 の戦闘結果は引き分けとする

引き分けあり

目つぶしあり

金的あり

精神攻撃は基本

表の見方

○=勝ち,　●=負け,　D=引き分け

2-b. 各ロボットの特徴

イ.Corners

基本的に弱い。コーナーに居座り続けるため狙い撃ちされやすく、弾丸の威力も弱い気がする。

ロ.Crazy

まず弱い。本当に狂ったように動き回って Interactive などでの狙いが定まらなかったこともあるけど、このロボット自身、敵を見つけてもすぐに見失うので手数で勝てない。

ハ.Fire

回転して他のロボットを見つけたら弾丸を発射するので、一旦見失うと次の攻撃までタイムロスしてしまう。しかもその間に被弾すると移動する為にさらにタイムロスしてしまうため移動しながら狙ってきたり、集中して攻撃してくるようなロボットには弱い。

ニ.Interactive

操作しづらい。何処が前で後ろか分かりずらい。

ホ.Interactive_v2

良い操作性。感覚的な操作で移動しやすく、結構勝てる。

ニ.MyFirstJuniorRobot

常に反復移動を繰り返し、撹乱しながらロボットを狙い撃ちするので、強い。被弾するとその射線上を避けようとするのもプラスになっている。

ホ.PaintingRobot

MyFirstJuniorRobot との違いは常にガンを回転させて、その間に見つけたロボットに対して攻撃するので、手数が MyFirstJuniorRobot に対して劣っている。MyFirstJuniorRobot ほど戦績は良くない。

ニ.RamFire
ロボットを見つけるとタックルを仕掛けるので、生き残りポイントで負けてもRam ポイントで稼いでトータルで勝つという面白い結果をみせた。

ホ.SittingDuck
戦闘ではなにもしないので勝てません。

ヘ.SpinBot
常に回転を続けながら、ロボットを見つけると攻撃するので相手の攻撃を上手く避けていた。

ト.Target
エネルギーが減ると移動するだけなので戦闘では勝てません。

チ.TrackFire
全く動かず、１つのロボットを狙い続ける固定砲台のようなロボットで、タイマンではかなりの勝ち数をあげている。だが狙い撃ちされやすいので多数でのサバイバルなら生き残りにくり。

リ.Tracker
ロボットに近づいてから集中的に攻撃をするので、基本的に強いが、近づいて来るロボットから避けるように一旦引いて、また近づこうとするので、回転するロボットには弱い。

ヌ.VelociRobot
反復運動や避ける移動に緩急をつけて複雑になっているので狙いを定められにくい。つよい。

ル.Walls
壁側を移動しながら敵を見つけたら攻撃するロボット。常に移動を続ける上に攻撃される方向が１つ減るので攻撃を受けにくい。サンプル最強。

3. 感想

Java の勉強になりました。それ以上にロボットの動きがおもしろくてそちらを楽しんでいました。なにもしないロボットはなんで作ったのか今でも理解出来ませんが、walls を考えた人はすごいと思います。

こういったオブジェクト指向というものに興味がわいてきました。