課題実験報告書

<Java による Robocode の作成>

~Tricky Hide~

高橋研究室　　4 年　No.29
田中　秀典

- 当実験の目的
  - Robocode を用いることにより、Java プログラムの理解とその内容の熟知
  - 他者のロボットに負ける事の無いロボット（プログラム）を作成、即ち勝てるロボットにするための戦略の模索

- Robocode について

  Robocode とは、IBM 提供のロボットシミュレータを指す。Robocode は Java 言語を用いて作成するロボットプログラムを対戦させるシミュレータで、作成するロボットはレーダー、砲台、車輪を持った戦車の形をしている。ゲーム感覚でプログラミングすることが出来、容易に Java 言語の習得が可能である。又、対戦は自分で作ったロボット同士はもちろん、他者が作ったロボットとも対戦することが可能である。

- ロボット作成について

  Robocode では、様々なイベント関数を用いてロボットを動かす。初期状態では run()関数と、onScannedRobot()関数のみしかなく、必要に応じて随時関数内のコードを追加したり、新たなイベント関数を設けたりすることで、ロボットの行動を変化させることが出来る。ちなみに初期状態では、前方(ahead)に 100 進み、大砲(gun)を一回りさせ、後方(back)に 100 戻り、大砲を再び一回りさせる、と言う動作を繰り返し、大砲回転時に同時に回転するレーダーが敵を索敵したときに大砲(fire)を発射する(強さの度合いは 1)という動きをする。

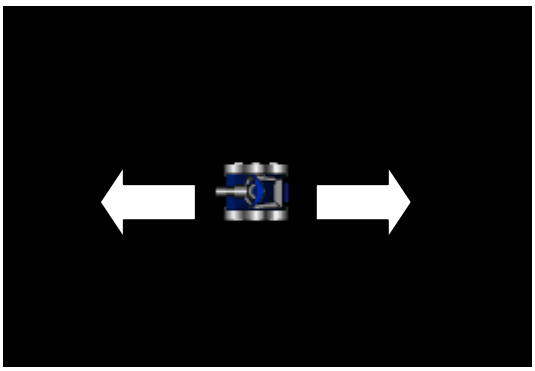

図 1.　初期状態ロボットの動き

簡単な変更の例を挙げてみる

```
public void run() {
        while (true) {
            ahead(100);
            turnGunRight(360);
            back(100);
            turnGunRight(360);
        }
    }
```

例えば、上記にある run()関数内のコードを一部変更し、

```
public void run() {
        while (true) {
            ahead(200);
            turnGunRight(360);
            back(100);
            turnGunRight(360);
        }
    }
```

ahead の部分の数値を 200 に変更してみる。こうなるとロボットの動きは、前方に 200 進み、大砲を一回りさせ、後方に 100 戻り、大砲を再び一回りさせるという動作になる。徐々に前方に進んでいるような動きをとるようになるのだ。

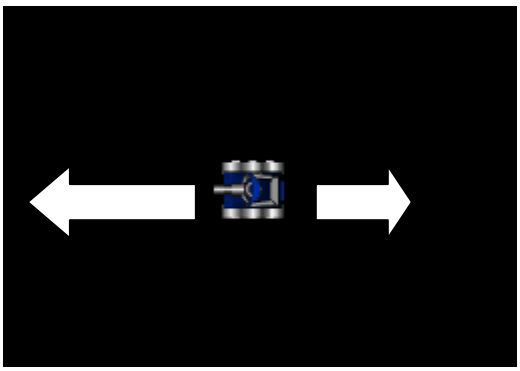

図 2.　変更後のロボットの動き

・ 対戦方法

Robocode 基本画面から[Battle]→[New]より新規に対戦をすることが出来る。次に対戦設定画面が表示され、参加させるロボット、対戦ラウンド数等の設定が行える。設定後、[Battlestart]で対戦開始。ロボットがランダムに配置後、互いに入力された動作を行いながら戦闘を始める。各ロボットのエネルギー(耐久力)は 100 であり、これが 0 になることでそのロボットは戦線離脱。最後に 1 台残ったロボットが勝者となる。尚、対戦終了時に各ロボットの対戦中のデータ(被弾数等)が表示される。

・ 攻撃について

ロボットが敵に対し攻撃を行うには、fire 関数を用いる。fire 関数は前述の通り大砲から弾を発射させる関数であり、このゲームの基本的な攻撃手段である。fire 関数は

fire(x);

で宣言し、この()内にあるｘは弾を発射する強さの度合いを表し、同時に弾を発射した際に消耗するエネルギーも表す。敵機被弾時のダメージ計算は

ダメージ数＝4×ｘ(fire に入力された数値)

である。又、ｘが 1 以上の数値の場合は

ダメージ数＝4×ｘ＋2×(ｘ－1)

となる。

更に、自機が発射した弾に敵機が被弾した場合、ボーナスとしてエネルギーが回復する。敵機が被弾すると

回復数＝ｘ×3

分のエネルギーが自機に追加される。即ち、fire(1)の攻撃を行った場合、

消費エネルギー　＝　1
ダメージ　　＝　4
エネルギー回復量＝　3

となる。

又、攻撃方法の一つとして体当たりというものも存在する。自機が敵機に接触した時にもダメージを与えることが可能であり、その際には両者ともに 0.6 のエネルギーを消費する。

・ 今回の実験の過程

＜1 日目＞

Java 実行環境の準備

ROBOCODE ダウンロード

ROBOCODE 動作確認(デモ画面の確認)

単純なプログラムのロボット作成(デフォルドのプログラムを多少改造)

作成したプログラムの簡単な動作確認

＜2 日目＞

ROBOCODE プログラム作成(主に基本動作部分を作成)

動作テスト

＜3 日目＞

ROBOCODE プログラム作成(攻撃、回避動作を作成、基本動作部分の改善)

動作テスト

＜4 日目＞

ROBOCODE プログラム作成(回避部分の改善)

動作テスト

＜5 日目＞

最終動作テスト

室内発表

・ 今回作成したロボットについて

＜基本コンセプト＞

体力配分を考えることが出来るロボット

＜具体的内容＞

・自機の体力（エネルギー）が多い時積極的に攻撃（火力が高い砲撃等）
・自機の体力（エネルギー）が少ない時、なるべく砲撃を避ける行動に出る

＜戦略概要＞

体力が多いうちは火力が高い攻撃を頻繁に行い、早期に体力増加を図り、少しでも戦闘を有利にする。移動はなるべく一点に留まらずに移動を繰り返し、被弾を少なくする。又、体力が少ない場合、エネルギーを消費する砲撃はなるべく行わず、回避運動を行うことによって被弾率を最小限に抑え、敵の疲弊を誘う。

・ プログラムコード

```
 package ht;
import robocode.*;
//import java.awt.Color;
import java.awt.Color;




/**
 * Hide1 - a robot by (your name here)
 */
public class Hide1 extends Robot
{
	double xfield=0,yfield=0;

	/**
	 * run: Hide1's default behavior
	 */
	public void run() {

		setColors(new Color(0,0,50),new Color(0,0,70),new Color(50,0,70));
		xfield = getBattleFieldWidth() / 2;
		yfield = getBattleFieldHeight() / 2;


		// After trying out your robot, try uncommenting the import at the top,
		// and the next line:
		//setColors(Color.red,Color.blue,Color.green);
		while(true) {
			// Replace the next 4 lines with any behavior you would like
			if(getEnergy() <= 50)
			{
```

```
        ahead(20000);
        if(getY() >= yfield)
        {
            while(true)
            {
                    ahead(20000);
                    turnLeft(90);
                    turnGunLeft(90);
                    ahead(20000);
                    turnLeft(90);
                    ahead(20000);
                    turnLeft(90);
                    turnGunRight(90);
                    ahead(20000);
                    turnLeft(90);
                    if(getEnergy() > 50)    break;
            }
        }
        else
        {
                while(true)
                {
                        ahead(20000);
                        turnRight(90);
                        turnGunRight(90);
                        ahead(20000);
                        turnRight(90);
                        ahead(20000);
                        turnRight(90);
                        turnGunRight(90);
                        ahead(20000);
                        turnRight(90);
                        if(getEnergy() > 50)    break;
                }
            }
    }
```

```
        if(getX() <= xfield && getY() <= yfield)
        {
                ahead(200);
            turnGunRight(360);
            turnLeft(150);
            back(150);
            turnGunRight(180);
            turnLeft(70);
    }
    else if(getX() >= xfield && getY() <= yfield)
    {
                ahead(150);
            turnGunRight(180);
            turnLeft(120);
            back(120);
            turnGunRight(360);
            turnRight(45);
        }
        else if(getX() <= xfield && getY() >= yfield)
        {
                ahead(60);
                turnLeft(45);
                back(100);
                turnGunRight(360);
                turnRight(180);
        }
        else
        {
                ahead(100);
                turnGunRight(360);
                turnLeft(60);
                back(120);
                turnGunRight(180);
                turnRight(10);
```

```
                }


        }


}


/**
 * onScannedRobot: What to do when you see another robot
 */
public void onScannedRobot(ScannedRobotEvent e) {
        if(e.getDistance() <= 75)
        {
                turnRight(e.getBearing() - 90);
            back(100);
    }
        else
        {
        if(getEnergy() >= 80)
            fire(2.5);
        else if(getEnergy() < 80 && getEnergy() >= 60)
            fire(1.7);
        else if(getEnergy() < 60 && getEnergy() >= 40)
            fire(1.2);
        else if(getEnergy() < 40 && getEnergy() >= 20)
            fire(1);
        else
            fire(1.5);
    }
}

/**
 * onHitByBullet: What to do when you're hit by a bullet
```

```
 */
public void onHitByBullet(HitByBulletEvent e) {
          if(getEnergy() >= 80)
          {
               fire(2);
               turnLeft(90 - e.getBearing());
               ahead(75);
     }
          else if(getEnergy() < 80 && getEnergy() >= 50)
          {
                    fire(.5);
                    turnLeft(120 - e.getBearing());
                    back(100);
          }


}

public void onHitRobot(HitRobotEvent e){
     if(getEnergy() >75)
     {
               turnRight(e.getBearing());
               ahead(100);
               fire(1);
          }
          else if(getEnergy() <= 75 && getEnergy() >50)
          {
          turnRight(90 - e.getBearing());
          back(100);
     }
}

public void onHitWall(HitWallEvent e){
     if(getEnergy() >= 50)
     {
```

```
                turnRight(e.getBearing());
                back(150);
                turnGunRight(180);
            }
            else
            {
                turnRight(e.getBearing());

            }
        }

}
```

・ プログラムコード解説

前述の通り、Robocode では各イベント関数に入力されているコードによってロボットの動作を決めている。ここでは、今回プログラムで使用されているイベント関数、コードについて解説していく。

＜本プログラムで使用したイベント関数＞

void run()

ロボットの基本動作(主に移動や索敵)を行うイベント関数

void onScannedRobot(ScannedRobotEvent event)

索敵時、敵を発見したときの動作を表すイベント関数

void onHitByBullet(HitByBulletEvent event)

自機に被弾したときの動作を表すイベント関数

void onHitRobot(HitRobotEvent event)

自機と敵機が衝突したときの動作を表すイベント関数

void onHitWall(HitWallEvent event)

自機が壁と衝突したときの動作を表すイベント関数

＜本プログラムで使用したコード＞

void ahead(double distance)

機体の前進

void back(double distance)

機体の後退

void turnLeft(double degrees)

機体の左方向旋回

void turnRight(double degrees)

機体の右方向旋回

void turnGunLeft(double degrees)

機体の砲身左方向旋回（レーダー含む）

void turnGunRight(double degrees)

機体の砲身右方向旋回（レーダー含む）

void fire(double power)

攻撃（弾発射）

double getX()

現在地点の X 座標を収得

double getY()

現在地点の Y 座標を収得

double getEnergy()

自分の残りエネルギーの数値を収得

double getBattleFieldHeight()

現在のバトルフィールドの縦幅を収得

double getBattleFieldWidth()

現在のバトルフィールドの横幅を収得

e.getDistance()

索敵時、敵機と自機との距離を収得

e.getBearing()

索敵時、敵機と自機との角度を収得

void setColors()

機体の色設定

・ 我が戦略について

基本的には前述の通りで体力が大の時（残存エネルギー50 以上）と体力が小の時（残存エネルギー50 以下）によって行動パターンを変える手段を用いた。以下、私が採った戦略を詳しく述べる。

＜体力が 50 以上の時の基本行動＞

先ず最初にあらかじめバトルフィールドの広さを収得する（戦闘開始時に実行）。その後フィールドを 4 つに区切り、区切ったそれぞれの空間（zone とする）に対応した行動、及び索敵を行う。

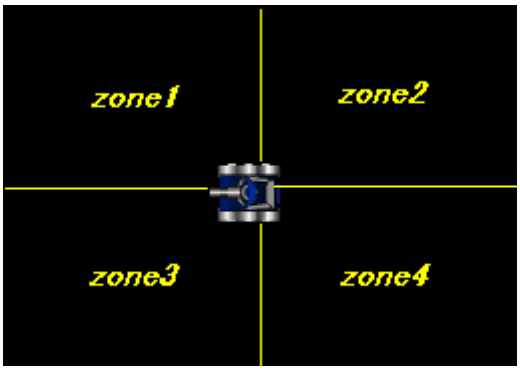

図 3.　戦略 1・区切られたフィールド

この 4 つの zone では、それぞれ違う動きをさせるようにプログラムしている。例えば zone3 で動いている時に何らかの拍子で zone4 に進入した場合、その時は zone4 に入力された行動をとるようになる。これにより、見かけ上では不規則な動きをとっている様に見え、さらに小刻みに動きを続けているので敵弾を避けやすくなる。

尚、各 zone の行動を以下に示すと

⑴zone 1⑴

```
ahead(60);
turnLeft(45);
back(100);
turnGunRight(360);
turnRight(180);
```

(zone 2)

```
ahead(100);
turnGunRight(360);
turnLeft(60);
back(120);
turnGunRight(180);
turnRight(10);
```

```
(zone 3)
        ahead(200);
        turnGunRight(360);
        turnLeft(150);
        back(150);
        turnGunRight(180);
        turnLeft(70);
 (zone 4)
        ahead(150);
        turnGunRight(180);
        turnLeft(120);
        back(120);
        turnGunRight(360);
        turnRight(45);
```

という行動を取る。

＜体力が 50 以下の時の基本行動＞

体力が 50 を切った地点で前項の行動を中断する。その時点で一番近い方向にある壁に向かって真っ直ぐ前進し、退避行動を行う。退避後は壁に沿って移動を繰り返す。

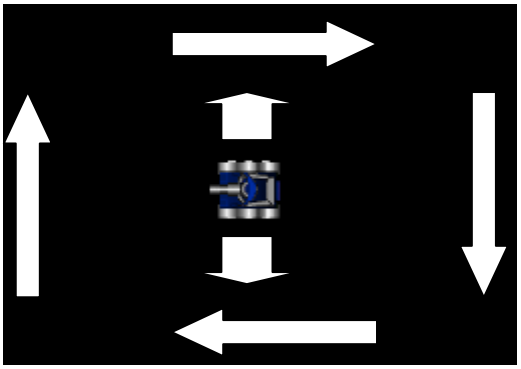

図 4.　戦略 2・退避行動

攻撃はなるべく行わず、行った場合は最小限のエネルギーを使い、敵を牽制する砲撃を行う。

＜攻撃について＞

攻撃は自機のエネルギーの残量によって決定する。

自機エネルギーが 80 以上　　･･･fire(2.5)

自機エネルギーが 80 以下 60 以上･･･fire(1.7)

自機エネルギーが 60 以下 40 以上･･･fire(1.2)

自機エネルギーが 40 以下 20 以上･･･fire(1)

自機エネルギーが 20 以下　　･･･fire(1.5)

基本的には残りエネルギーが少なくなるにつれて攻撃強度を弱めているが、20 以下になったときには前段階より強い攻撃をする。これは敢えて強い攻撃を発射することにより、撃墜される危険はあるが当たるとエネルギーを多く回復できるからである。

・ 戦闘結果と反省点

◇ 既存のサンプルプログラムにはそれなりの割合で勝利することが可能である

◇ 中人数の集団戦に強く、少人数（1 対 1）にも対応は可能である。だが、大人数の混戦の場合、小刻みに動くという動作が仇となり、自ら被弾し早期に撃墜される場合が多かった。

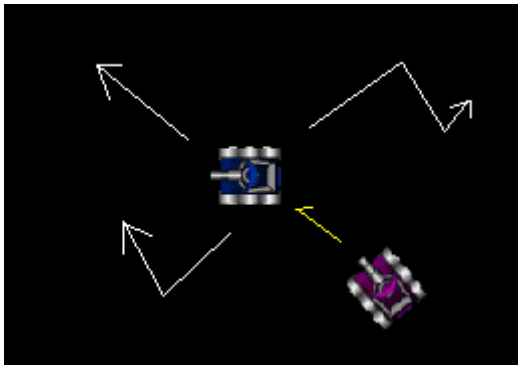

図 5.　中・少人数の対戦時

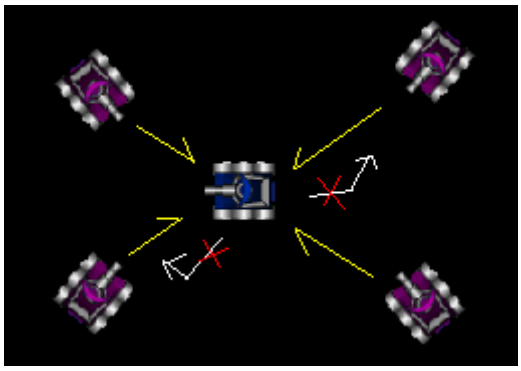

図 6.　大人数の対戦時

上図において、白線が自機の移動を、黄線が敵機の砲撃を表す。中・少人数の時は逃げるための空間が多いため、こちらに向かってくる弾を避けるのは容易ではあるが、大人数の場合、多方向から狙われるために避け切れない、又は避けた地点に流れ弾が通るというケースが多い。これに関しては改善する余地があるだろう。

◇　初期配置により勝率がばらつく（勝利に運の要素も絡んでしまう）

・　感想

今回は高橋研究室において Java を用いて Robocode の作成という実験を行いました。普段高橋先生は電子回路や電子デバイス等の電気関係の授業でお世話になっているので、今回の課題実験の内容もそれに準した内容になるのではないかと実験前には思っていました。自分は元から回路関係は苦手な分野であり、正直大丈夫なのかと不安には感じていたが、いざ実験内容を聞いて見るとプログラミングを用いる実験だったので、正直ほっとしました（こんなこと書いたら怒られるか）。ただ、プログラムの内容は Java と、自分にとって未習得の言語であったため、最初のうちは本当に何もかもが手探り状態で実験を進めていたことを覚えています。又、開発環境が Linux と、普段 Windows を主として使用している自分にとっては思い通りに行かない部分も多かったと思います。とにかく何もかもが目新しい事だらけであったので、最初のうちは困惑したものの、慣れてくると仕組みや言語がある程度理解できるようになり、途中からは作業速度が上がってきた様に感じました。又、分からない所も先生の助言も相まって理解するのも容易かったと思われます。とりあえず最後には多少は自分の考えた通りにはいきませんが、

形としては自分の個性が出ているロボットを作成することが出来、本当に良かったと思います。

ただ、実験の手ごたえは十分でしたが、実験の準備などに不十分な点が多く、一番失敗したのは室内発表の原稿（ファイル形式）を間違えるという大きなミスを犯したのが非常に心残りでした。そのほかにも数え切れない反省点も多く、自分の管理能力が低いと言う何よりの証拠となっているので改善していきたいと思います。

最後に、Robocode を実際に用いた感想ですが、半ばゲーム感覚で実験を出来たので非常に楽しいものであったと思います。最初のうちは試行錯誤でロボットの動きを設定したが、後から改善を繰り返し、最後にはそれなりに『勝つ』ロボットが出来たと思います。ただ、『勝つ』ロボットは作ることが出来ても、『常勝』するロボットを作成するのは難しいものであると改めて実感しました。

これらの経験や反省点を踏まえて、5 年に行う実験や卒業研究にも生かしておきたいと思います。

・ 参考文献

Robocode ホーム

http://www-6.ibm.com/jp/event/robocode/home/

Java-Developer ‐ Robocode を使った Java プログラミング撃破術 ‐

http://www.javadeveloper.jp/static/200208/robocode/robo1.html

Robocode

http://www.trident-com.com/taiken/Robocode.html

高橋研究室

http://nt.hakodate-ct.ac.jp/~takahasi/top.html