課題実験　最終報告書

Roocode　−情報収集型ロボットの研究−

情報工学科　4年　35番　松橋あずさ

研究室　　高橋研究室

実験期間　2003年10月23日～
　　2003年11月18日

提出日　　2003年12月15日

§目次§

1. 目的

Robocodeのロボット作成において、情報を収集・駆使するアルゴリズムを研究する。

また、Robocodeを通してJava言語を習得し、これまでに学習した言語と比較することで、プログラミング言語の標準的な記述方法や言語による相違点を研究する。経験を通して、簡潔でわかりやすくメンテナンスのしやすいプログラムの記述方法を追及する。

2. 概要

2. 1 Java言語

Java言語は、1991年 Sun Microsystems 社によって開発された言語である。その特徴は、「write once, run anywhere(一度作成すればどこでも実行できる)」という言葉で表されるように、異なるCPUとOSのもとで動作する点にある。近年のWWWとインターネットの爆発的な成長のもとで注目を集め、広く普及した言語の一つである。

2. 2 Robocodo

Robocodeとは、IBMがJava言語の普及を目的として開発・提供したソフトウェアである。その内容は、ディスプレイ上のフィールド内であらかじめ行動パターンをプログラムされたロボットの戦車が互いに攻撃しあって生き残りを競うゲームである。

■ルール

対戦形式	1対1または混戦
勝敗	多くの弾丸を敵に当て、最後まで生き残ったロボットの勝ち
エネルギー	ロボットはゲームスタート時にエネルギー100 を持つ
	敵に弾丸を当てるとエネルギーが増加
	敵の弾丸が当たるとエネルギーが減少

3. 開発環境

■開発環境

OS RedHat Linux 9

言語 Java言語(J2SDK 1.4.2)

■参考資料

Robocode API (Webページ、配布資料)

情報応用実験『UNIXに関する実験(1)』 実験テキスト

4. 方法

4. 1 ロボット作成までの流れ

今回の実験では、「情報収集型ロボットの研究」を目標に、次のような流れでロボットを作成した。

（1）RobocodeAPIに関する資料をもとに、クラスやメソッドを理解する。
（2）サンプルロボットのバトルを観戦する。
（3）簡単なコードでロボットの動作を確認する。
（4）自作ロボットの作成
　①行動パターンを思案
　②コーディング
　③テスト（動作確認）とデバッグ
　④バトル結果から改善点を考察
　①～④繰返し

4. 2 Robocodeの起動方法

Robocodeを起動するには、あらかじめRobocodeのソフトをHPから自分のディレクトリにダウンロードしておく。

（1）ターミナルの起動
　[スタート]→[ターミナル]
（2）Robocodeの起動
　コマンド入力画面で次のように入力する。
　　cd robocode↓
　　./robocode.sh↓
　Robocodeが起動する。

4. 3 エディタの使用方法

ロボットを作成するには、エディタを使用して、ファイルの作成・ソースコードの入力・コンパイルを行なう。

（1）エディタの起動
　[Robot]→[Editor]
（2）新規ファイル作成
　①ロボットファイル作成の場合
　　[File]→[New]→[Robot]
　　ダイアログボックスのメッセージにしたがって[ロボットの名前]と[パッケージ名]を入力。
　②Javaファイル作成の場合
　　[File]→[New]→[Java File]
　　ダイアログボックスのメッセージにしたがって[ファイル名]を入力。
（3）ソースコードの入力
　ロボットの行動パターンをプログラムする。

(4)ファイルの保存

[File]→[Save As]

ダイアログボックスのメッセージにしたがって、保存先を選択し[ロボットの名前]を入力して保存。

※以後、既存ファイルを開くときは[File]→[Open]から開く。

(5)コンパイル

[Compiler]→[Compile]

コンパイルが終了すると、エラーがある場合はエラー内容が表示される。

(6)デバッグ

(5)でコンパイルした結果、エラーがある場合はエラー箇所を修正。

[File]→[Save]でファイルを上書きし、コンパイルする。

4.4 バトルスタート方法

作成したロボットやサンプルロボットをバトルで対戦させるには、バトルに参加するロボットを選択し、ラウンド数を設定する。

(1)バトル設定

[Robot]→[Battle]→[New]

ダイアログボックスの[Robots]項目で、バトルに参加するロボットを選択し[Add]をクリック。

[Number of Rounds]項目にラウンド数を設定。

(2)バトルスタート

[Start Battle]をクリック。

4.5 ロボットのプログラム方法

今回の実験内容は、ロボットの行動パターンをプログラムすることが中心となる。

ここでは、プログラム本体、カスタムイベント、クラス定義のプログラム方法について説明する。

4. 5. 1 プログラム本体

はじめに、ロボットのプログラム全体像について説明する。
プログラム本体のモデルを以下に示す。

プログラム本体

```
package パッケージ名;
import robocode.*;
import java.awt.Color;

public class ロボット名 extends AdvancedRobot
{
    public void run() {
        初期設定
        イベント優先順位の設定
        カスタムイベントの追加（次章を参照）
                                    など

        while(true) {
            ロボットの基本行動
        }
    }

    public void イベント名(パラメータ) {
        イベント対応処理
    }
                :
    public 型　関数・手続き名(パラメータ) {
        　処　理
    }
                :
}
```

プログラム本体の1行目にパッケージ名を記述し、2行目以降にインポートファイルの宣言をする。
次行からはロボットのクラス定義である。ロボットの行動はバトル実行時のイベント発生によって決定されるので、イベント対応処理をメソッドで記述することが、クラス定義の中心となる。
ロボットのメインメソッドであるrunメソッドには、初期設定を記述し、ロボットの基本行動を無限ループの中に記述する。
以降には、イベント対応処理メソッドと、処理の中で使用する関数・手続きの記述を行なう。

4. 5. 2 カスタムイベント

Robocodeでは、あらかじめ用意されているイベントの他に、プログラマがカスタムイベントを作成することができる。

カスタムイベントを作成するには、run メソッド内でカスタムイベントの追加を行い、ロボットのクラス定義内でカスタムイベントの処理を記述する。

カスタムイベント追加、および、カスタムイベント対応処理のモデルを以下に示す。

カスタムイベント追加

```
addCustomEvent(
    new Condition("カスタムイベント名") {
        public boolean test() {
            return (カスタムイベント発生条件);
        };
    }
);
```

カスタムイベント対応処理

```
public void onCustomEvent(CustomEvent e) {

    if (e.getCondition().getName().equals("カスタムイベント名")) {
        カスタムイベント対応処理
    }
                :
}
```

カスタムイベントが複数ある場合は、カスタムイベントの数だけ追加を行ない、カスタムイベントの数だけ対応処理の if 文を記述する。

また、カスタムイベントの数が多い場合は、カスタムイベント対応処理を関数化し、onCustomEvent()メソッドをその関数へのテーブルのようにするとわかりやすい。

4. 5. 3 クラス定義

クラスを作成する場合は、Javaファイルを新規作成し、そのファイル名をクラス名として保存する。
クラス定義のモデルを以下に示す。

クラス定義

```
package パッケージ名;
import robocode.*;

    class クラス名 {
        private 型 [ 属 性 ];
                :
        public メソッド名(パラメータ) {
            [ 処 理 ]
        }
                :
    }
};
```

パッケージ名は、クラスを使用するロボットと同一のパッケージ名にする。
ここで定義されたクラスは、ロボットのプログラム本体でインスタンス化され、インスタンス化されたオブジェクトによってメソッドがコールされる。

5. 結果

5. 1　自作ロボットの戦略

■攻撃面

全ての敵の中で、最も近距離で低エネルギーの敵ロボットを攻撃。
敵ロボットのエネルギーや距離を考慮し、状況に応じて攻撃方法を選択。

■防御面

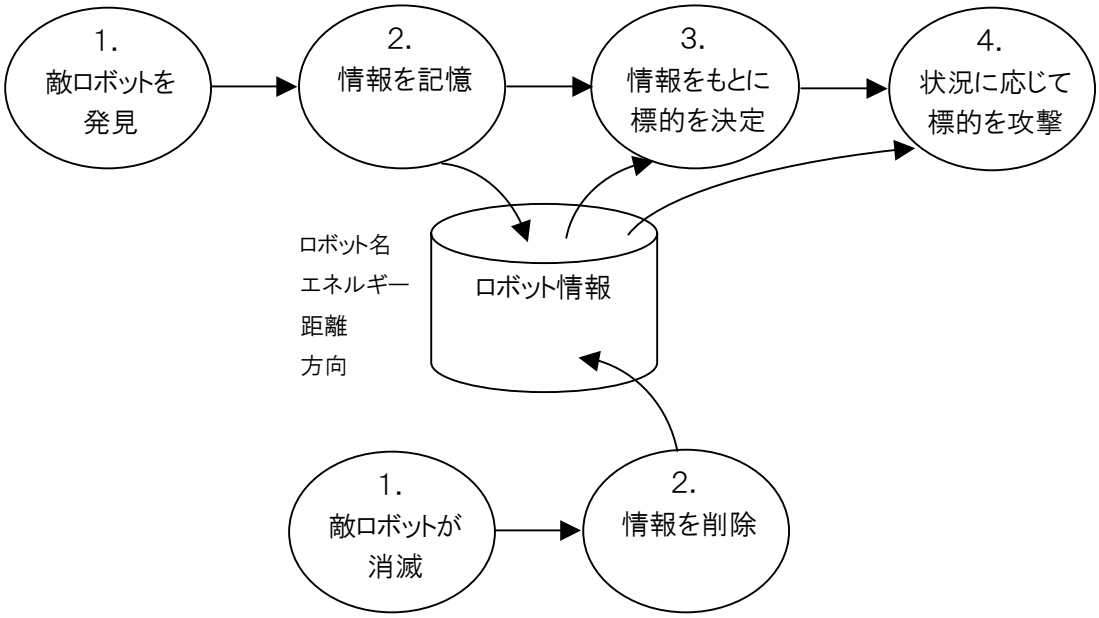
集中攻撃を避けるため、敵の群れを回避。

以上の要求を満たすロボットとして、レーダーを常に回転させ、敵ロボットの情報を収集し、記憶された情報に基づいて行動パターンを決定する「情報収集型ロボット」を考える。

5. 2　自作ロボットの戦術

5. 2. 1　情報に基づく敵の攻撃

（1）戦術アルゴリズム



敵ロボットを発見した場合、情報を記憶し、情報をもとに最も距離が近くエネルギーが低い敵ロボットを攻撃する。

敵ロボットが消滅した場合、そのロボットに関する情報を削除する。

（2）クラスとインスタンス

■ロボット情報クラス　RobotInformationClass
　　ロボット名
　　エネルギー
　　距離
　　方向

■インスタンス　　vRobotInformaiton

（3）情報の収集

■addsetRobotInformation メソッド

　ScannedRobotEvent が発生すると、ロボット名をキーとしてロボット情報を検索し、格納済みのロボットの場合ロボット情報を上書きし、それ以外の場合ロボット情報を追加する。

■removeRobotInformation メソッド

　RobotDeathEvent が発生すると、ロボット名をキーとしてロボット情報を検索し、格納済みのロボットの場合ロボット情報を削除する。

（4）情報の駆使

■getNearAndLittleEnergy メソッド

　距離とエネルギーをキーとしてロボット情報を検索し、距離が150ピクセル以下のロボットが存在する場合、そのなかで最もエネルギーの低いロボットを、それ以外の場合、最も距離の近いロボットを標的とする。

■fireRobot メソッド

　標的のロボット情報をもとに、敵の方向を向き、距離とエネルギーに応じて攻撃パターンを選択する。

　攻撃パターン

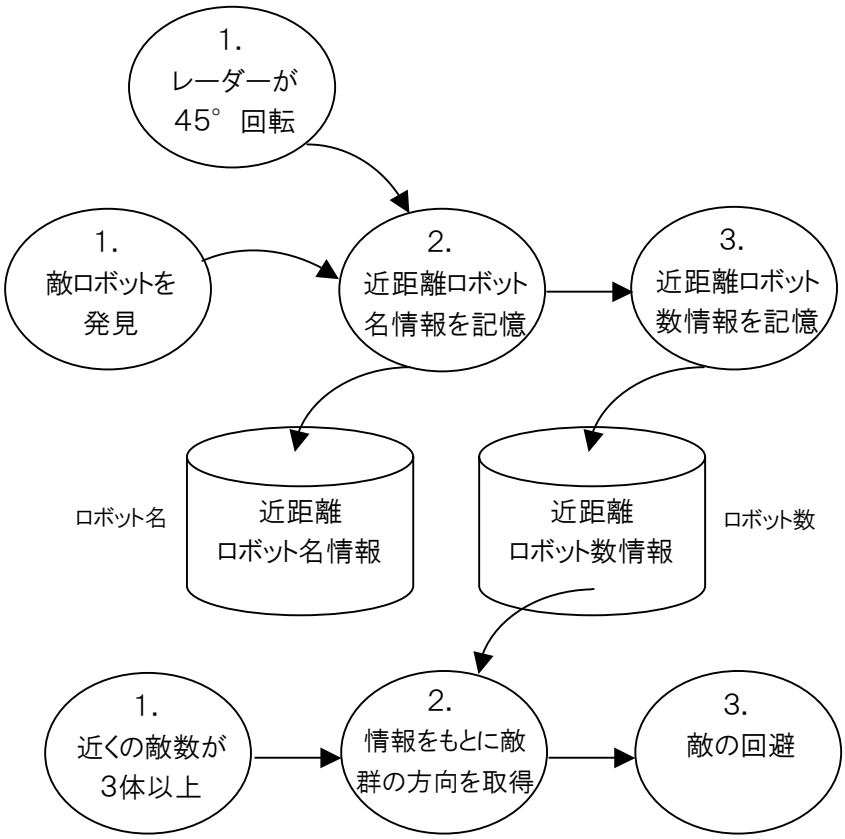
　　①敵のエネルギーがゼロの場合　→　衝突。

　　②自分のエネルギーが相手よりも充分に大きい場合　→　弾丸で攻撃し前進。

　　③上記以外の場合　→　弾丸で攻撃し距離を200ピクセルに保つ。

5. 2. 2 情報に基づく敵の回避

（1）戦術アルゴリズム

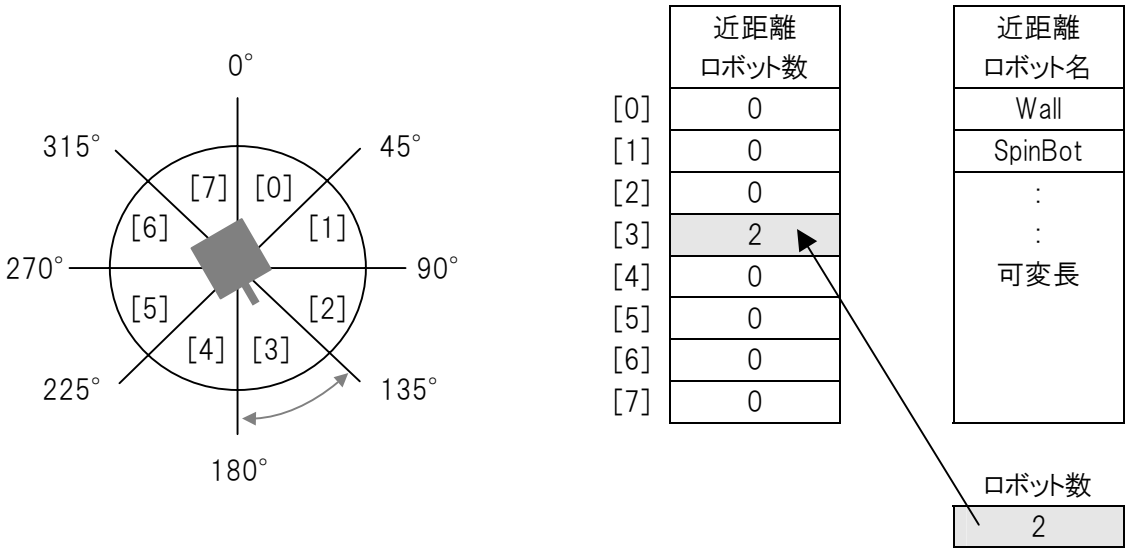


敵ロボットを発見した場合、そのロボットが自分のロボットを中心とした半径100ピクセル内に存在するならば、敵ロボットの数を記憶し、その数が3体以上であれば敵の群れの方向を求めてなるべく敵の存在しない方向へ移動する。

（2）クラスとインスタンス

■近距離ロボット名情報クラス NearRobotClass ロボット名 ■インスタンス vNearRobot	■近距離ロボット数情報クラス NearRobotNumberClass ロボット数 ■インスタンス vNearNumberRobot

(3)情報の収集



自分のロボットを中心とした半径100ピクセルの円を45° 刻みの8つのブロックに区切り、それぞれのブロック内に存在するロボット数を近距離ロボット数情報に記憶する。ブロック数は8なので、近距離ロボット数情報の要素数は8となる。

レーダーの回転角度はレーダー回転制御変数で制御し、45° 回転するたびにこの変数を更新する。

近距離ロボット数情報の要素番号は〔レーダー回転制御変数/45〕で表す。

■onPass45 メソッド

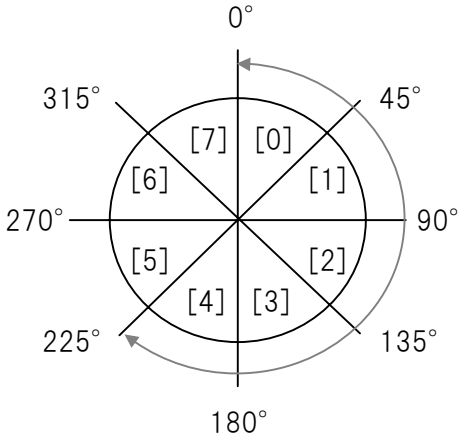
レーダーが45° 回転すると、ロボット数をゼロクリアし、レーダー回転制御変数を更新する。

敵ロボットの位置は時間の経過とともに変化するので、レーダーが360° 回転した場合、近距離ロボット名情報をクリアする。

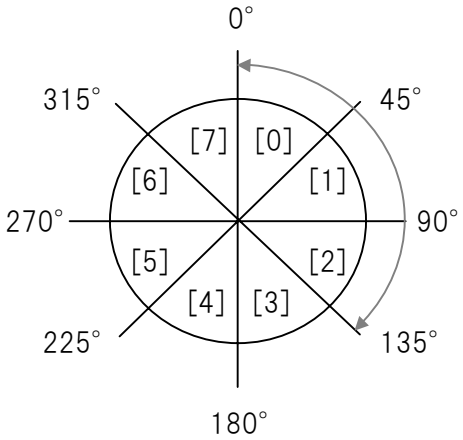
■addNearRobotNumber メソッド

1つのブロックに対して、発見した敵のロボット名を近距離ロボット名情報に記憶し、同時にロボット数をカウントしていく。このとき、発見した敵ロボットがすでに格納済みの場合、近距離ロボット名情報の記憶およびロボット数のカウントは行なわないものとする。レーダーが45° 回転した時点で、このブロックの合計ロボット数を近距離ロボット数情報に記憶する。

(4)情報の駆使



検索幅5



検索幅3

■getBeginNumber メソッド

近距離ロボット数情報について、要素番号0から7までのどの要素に敵が多く存在するかを求める。

まず、要素番号0を検索開始番号として、0°から225°までの範囲(検索幅5ブロック)に存在する敵ロボットの数を合計する。次に要素番号1を検索開始番号として、今度は45°から270°までの範囲(検索幅5ブロック)に存在する敵ロボットの数を合計する。このとき、先ほど求めた合計よりも値が大きい場合は、合計値を更新し、このときの要素番号を記憶しておく。同様に要素番号7までの敵ロボットの合計を求め、合計値と要素番号を更新していく。要素番号7までの処理が終了した時点で、ロボット数の合計が最大となるときの要素番号が求まっている。

次に、先ほど求めた要素番号を検索開始番号とし、今度は検索幅を3ブロックにし、くりかえし検索回数を3回にして、同様の処理を行なう。3回の処理が終了した時点で、ロボット数の合計が最大となるときの要素番号が求まっている。

以上の処理で、敵の群れの方向示す要素番号が求まる。

■runawayFromRobots メソッド

getBeginNumber メソッドをコールして敵の群れの方向示す要素番号を求め

〔要素番号〕＊45－〔自分のロボットの向き〕

だけ、右回転して敵の群れの方向を向き、後退する。

5. 3 クラスとメソッドの定義

作成したクラスとメソッドの詳細を以下に示す。

5. 3. 1 クラス定義

RobotInformationClass クラス（ロボット情報）
■ロボット情報を記憶。 ■addsetRobotInformation メソッドで情報を更新。 ■removeRobotInformation メソッドで情報を削除。 属性　敵ロボットの名前、エネルギー、距離、方向

NearRobotClass クラス（近距離ロボット名情報）
■近距離ロボット名情報を記憶。 ■addNearRobotNumber メソッドで情報を更新。 属性　敵ロボットの名前

NearRobotNumberClass クラス（近距離ロボット数情報）
■近距離ロボット数情報を記憶。 ■addNearRobotNumber メソッド、onPass45 メソッドで情報を更新。 属性　敵ロボットの数

5. 3. 2 メソッド定義

＊印はRobocodeで用意されているメソッド。

＊run メソッド
public void run ()
■イベントの優先順位、ロボットの色、レーダーの回転の設定。 ■レーダー回転制御変数と近距離ロボット数情報ベクトルの初期化。 ■カスタムイベントの追加。 ■ロボットの基本移動を実行。
パラメータ　なし　　リターン　なし

＊onScannedRobot メソッド
public void onScannedRobot (ScannedRobotEvent e)
■addsetRobotInformation メソッドをコール。 ■敵までの距離が近い場合、addNearRobotNumber メソッドをコール。 ■近距離ロボット数が3体以上の場合、runawayFromRobots メソッドをコール。
パラメータ　敵ロボットのインスタンス　　リターン　なし

＊onHitRobot メソッド
public void onHitRobot (HitRobotEvent e)
■自分のロボットと敵のロボットのエネルギーを比較して、状況に応じた攻撃を実行。
パラメータ　敵ロボットのインスタンス　　リターン　なし

＊onRobotDeath メソッド
public void onRobotDeath (RobotDeathEvent e)
■removeRobotInformation メソッドをコール。
パラメータ　敵ロボットのインスタンス　　リターン　なし

＊onCustomEvent メソッド

public void (CustomEvent e)

■カスタムイベント生成要因に応じて、onRadarStop メソッド、onNearWall メソッド、onPass45 メソッドのいずれかをコール。

パラメータ　カスタムイベントのインスタンス　　リターン　なし

onRadarStop メソッド

public void onRadarStop ()

■レーダーの回転を持続。

パラメータ　なし　　リターン　なし

onNearWall メソッド

public void onNearWall ()

■壁から離れる。

パラメータ　なし　　リターン　なし

onPass45 メソッド

public void onPass45 ()

■近距離ロボット数情報の更新。
■近距離ロボット数変数のゼロクリア。
■レーダー回転制御変数の更新。
■レーダーが1回転した場合、レーダー回転制御変数の調整、および、近距離ロボット数情報ベクトルのクリア。

パラメータ　なし　　リターン　なし

fireRobot メソッド

public void fireRobot (RobotInformationClass e)

■自分のロボットと敵のロボットのエネルギーを比較して、状況に応じた攻撃を実行。

パラメータ　敵ロボットのインスタンス　　リターン　なし

addsetRobotInformation メソッド

public void addsetRobotInformation (ScannedRobotEvent e)

■ロボット情報を検索し、パラメータで与えられた敵ロボットがすでに格納済みの場合、ロボット情報の上書き。それ以外の場合、ロボット情報の追加。

パラメータ　敵ロボットのインスタンス　　リターン　なし

removeRobotInformation メソッド

public void removeRobotInformation (RobotDeathEvent e)

■ロボット情報を検索し、パラメータで与えられた敵ロボットがすでに格納済みの場合、ロボット情報の削除。

パラメータ　敵ロボットのインスタンス　　リターン　なし

getNearAndLittleEnergyRobot メソッド

public RobotInformationClass getNearAndLittleEnergyRobot ()

■ロボット情報を検索し、距離が150ピクセル以下のロボットが存在する場合、そのなかで最もエネルギーの低いロボットをリターン。それ以外の場合、最も距離の近いロボットをリターン。

パラメータ　なし　　リターン　敵ロボットのインスタンス

addNearRobotNumber メソッド

public void addNearRobotNumber (ScannedRobotEvent e)

■近距離ロボット名情報を検索し、パラメータで与えられた敵ロボットが格納済みでない場合、近距離ロボット名情報の追加。
■近距離ロボット数情報の上書き。

パラメータ　敵ロボットのインスタンス　　リターン　なし

countNearRobotNumber メソッド

public int countNearRobotNumber ()

■近距離ロボット数情報を検索し、近距離ロボット数の総和をリターン。

パラメータ　なし　　リターン　近距離ロボット数の総和

runawayFromRobots メソッド

public void runawayFromRobots ()

■getBeginNumber メソッドをコールして敵の群れの方向を求め、敵の群れを回避。

パラメータ　なし　　リターン　なし

getBeginNumber メソッド

public int getBeginNumber (int a , int b , int c)

■与えられたパラメータに応じた範囲で、近距離ロボット数情報を検索し、敵の群れの方向を示す近距離ロボット数情報の要素番号をリターン。

パラメータ　検索開始要素番号
検索回数
検索幅

リターン　敵の群れの方向を示す
近距離ロボット数情報の要素番号

normalRelativeAngle メソッド
public double normalRelativeAngle (double angle) ■角度を-180から180の範囲に変換してリターン。 パラメータ　角度(-∞≦θ≦∞)　　リターン　角度(-180≦θ≦180)

5. 4 データディクショナリ

作成したデータについて、データ変数名・型名・内容を以下に示す。

5. 4. 1 クラス属性

RobotInformationClass クラス		
sName	String	敵ロボットの名前
dEnergy	double	敵ロボットのエネルギー
dDistance	double	敵ロボットの距離
dBearing	double	敵ロボットの方向
normalAbsoluteAngle メソッド		
angle	double	角度(-∞≦θ≦∞)
fixedAngle	double	角度(-180≦θ≦180)

NearRobotClass クラス		
sName	String	敵ロボットの名前

NearRobotNumberClass クラス		
iNumber	int	敵ロボットの数

5. 4. 2 グローバル変数

グローバル		
iRadarHeading	int	レーダーの方向
iNumber	int	敵ロボットの数
vRobotInformation	Vector	ロボット情報ベクトル
vNearRobot	Vector	近距離ロボット名情報ベクトル
vNearRobotNumber	Vector	近距離ロボット数情報ベクトル

5. 4. 3 ローカル変数

run メソッド		
e	RobotInformatonClass	ロボット情報のインスタンス

addsetRobotInformation メソッド		
temp	RobotInformatonClass	ロボット情報のインスタンス

removeRobotInformation メソッド		
temp	RobotInformatonClass	ロボット情報のインスタンス

getNearAndLittleEnergyRobot メソッド		
temp	RobotInformatonClass	ロボット情報のインスタンス
return1st	RobotInformatonClass	距離が150ピクセル以下のロボット情報のインスタンス
return2nd	RobotInformatonClass	距離が150ピクセル以上のロボット情報のインスタンス
dMinDistance	double	距離の最小値
dMinEnergy	double	エネルギーの最小値

addNearRobotNumber メソッド		
temp	NearRobotClass	近距離ロボット名情報のインスタンス

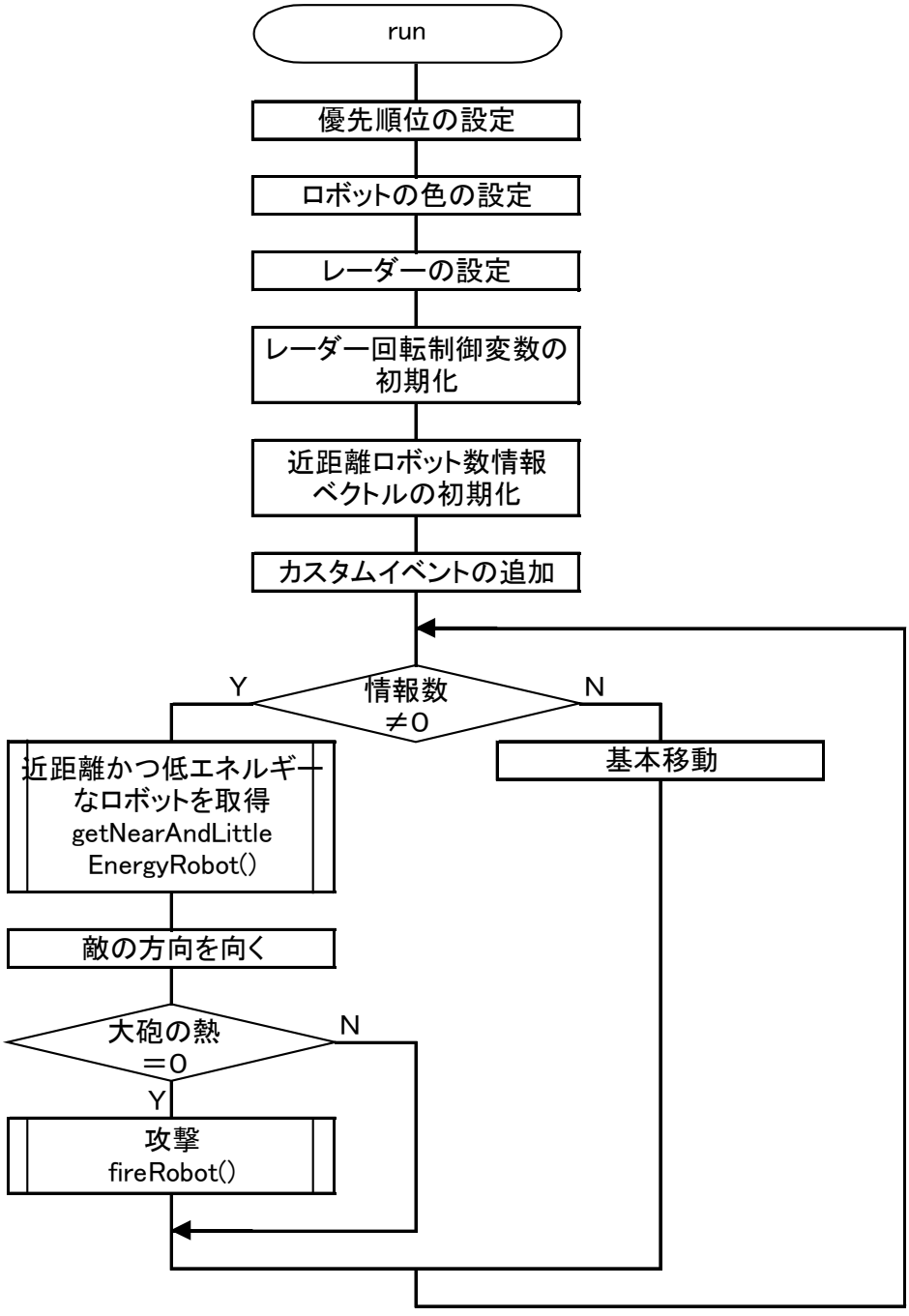
countNearRobotNumber メソッド		
temp	NearRobotNumberClass	近距離ロボット数情報のインスタンス
iNearRobotNumber	int	近距離ロボット数

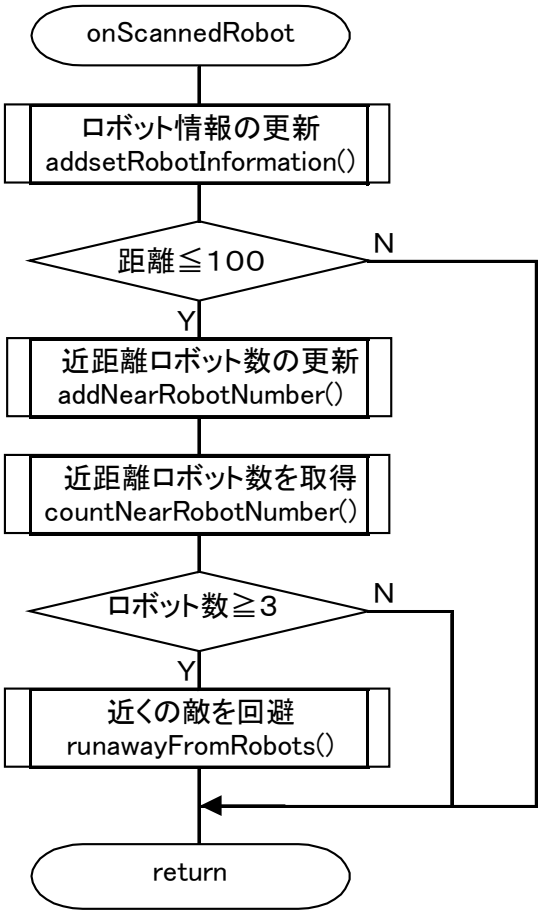
runawayFromRobot メソッド		
iBeginNumber	int	敵の群れの方向を示す近距離ロボット数情報の要素番号
turnRobotAmt	double	ロボット回転量

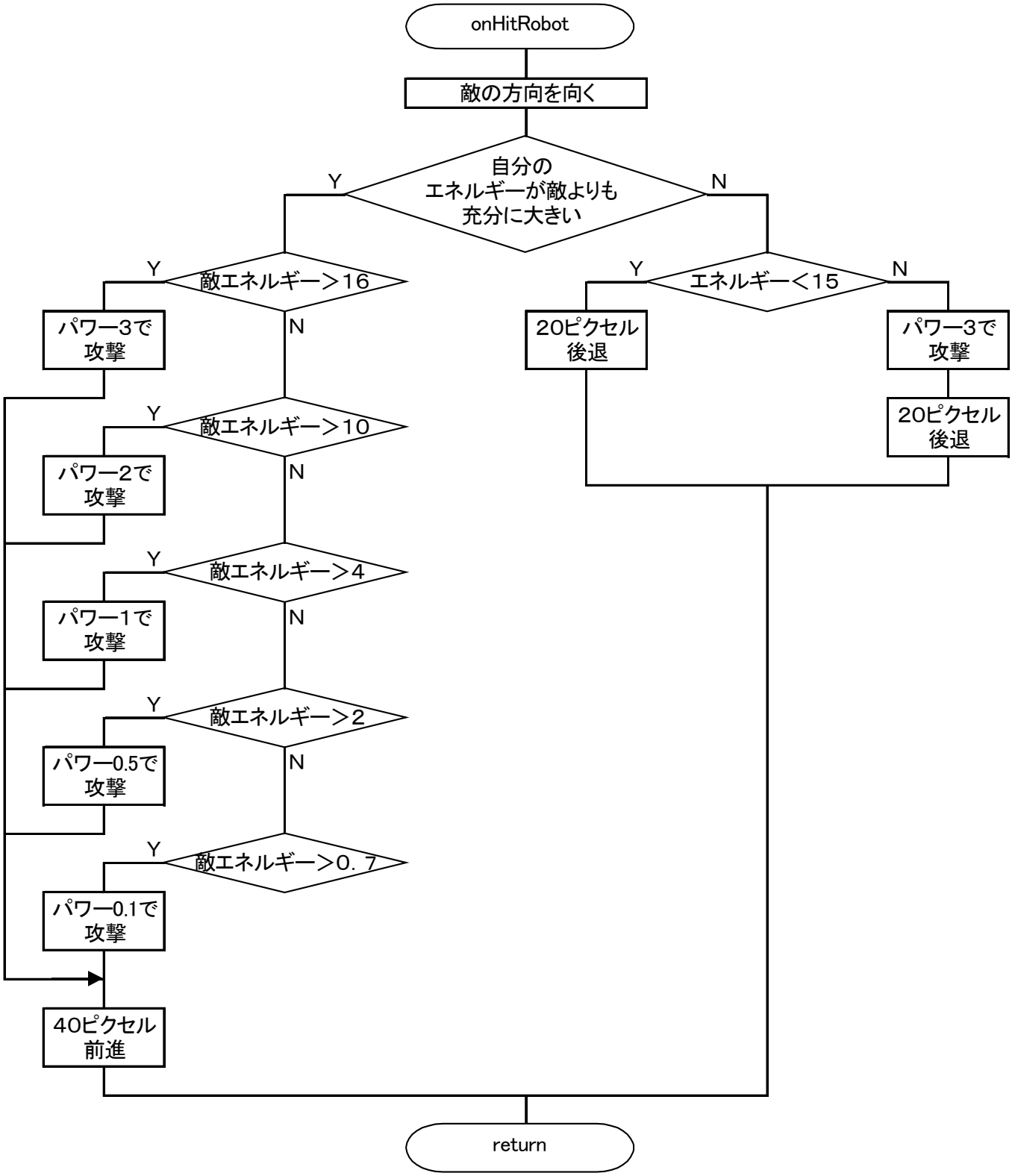
getBeginNumber メソッド		
a	int	検索開始要素番号
b	int	検索回数
c	int	検索幅
temp	NearRobotNumberClass	近距離ロボット数情報のインスタンス
iSum	int	近距離ロボット数情報の各要素におけるロボット数
iSumMax	int	ロボット数の最大値
iBeginNumber	int	敵の群れの方向を示す近距離ロボット数情報の要素番号

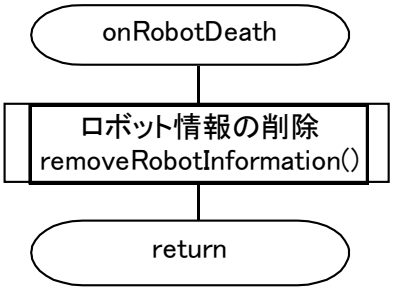
normalRelativeAngle メソッド		
angle	double	角度($-\infty \leqq \theta \leqq \infty$)
fixedAngle	double	角度($-180 \leqq \theta \leqq 180$)

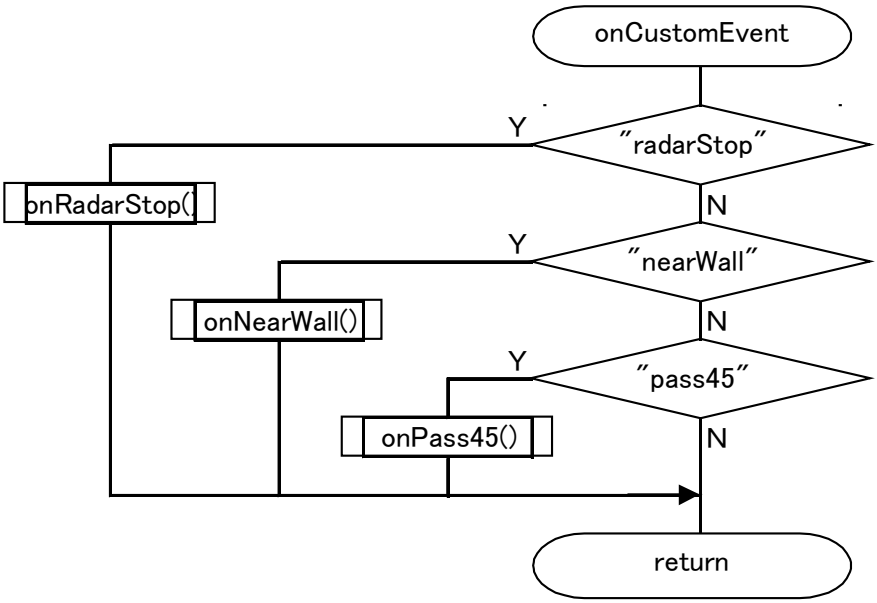
5. 5 フローチャート

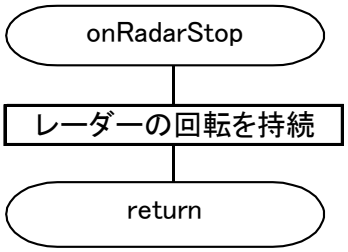

run
優先順位の設定
ロボットの色の設定
レーダーの設定
レーダー回転制御変数の
初期化
近距離ロボット数情報
ベクトルの初期化
カスタムイベントの追加
Y
情報数
≠0
N
近距離かつ低エネルギー
なロボットを取得
getNearAndLittle
EnergyRobot()
基本移動
敵の方向を向く
大砲の熱
＝0
N
Y
攻撃
fireRobot()

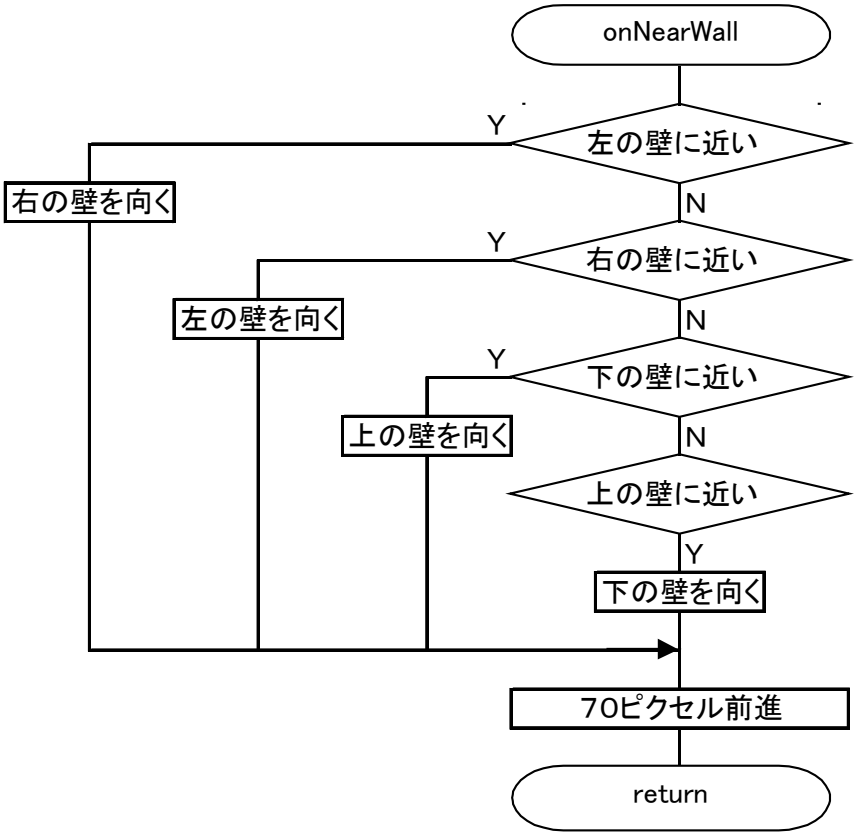

onScannedRobot
ロボット情報の更新
addsetRobotInformation()
距離≦100
N
Y
近距離ロボット数の更新
addNearRobotNumber()
近距離ロボット数を取得
countNearRobotNumber()
ロボット数≧3
N
Y
近くの敵を回避
runawayFromRobots()
return

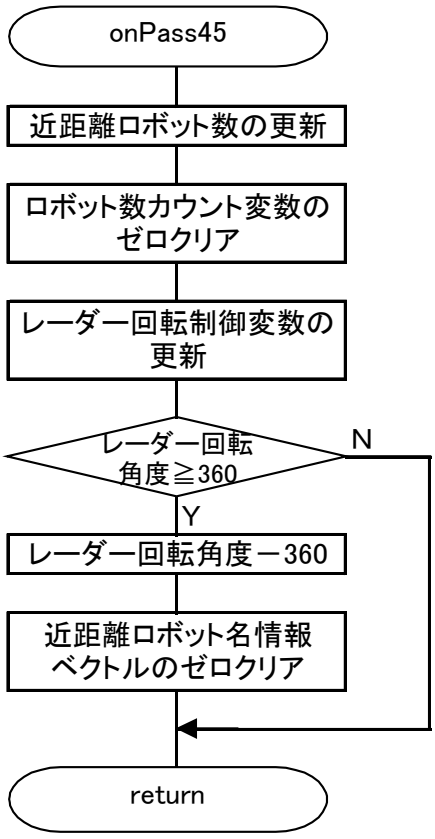

onHitRobot
敵の方向を向く
自分の
エネルギーが敵よりも
充分に大きい
Y
N
敵エネルギー＞16
パワー3で
攻撃
敵エネルギー＞10
パワー2で
攻撃
敵エネルギー＞4
パワー1で
攻撃
敵エネルギー＞2
パワー0.5で
攻撃
敵エネルギー＞0．7
パワー0.1で
攻撃
40ピクセル
前進
エネルギー＜15
20ピクセル
後退
パワー3で
攻撃
20ピクセル
後退
return

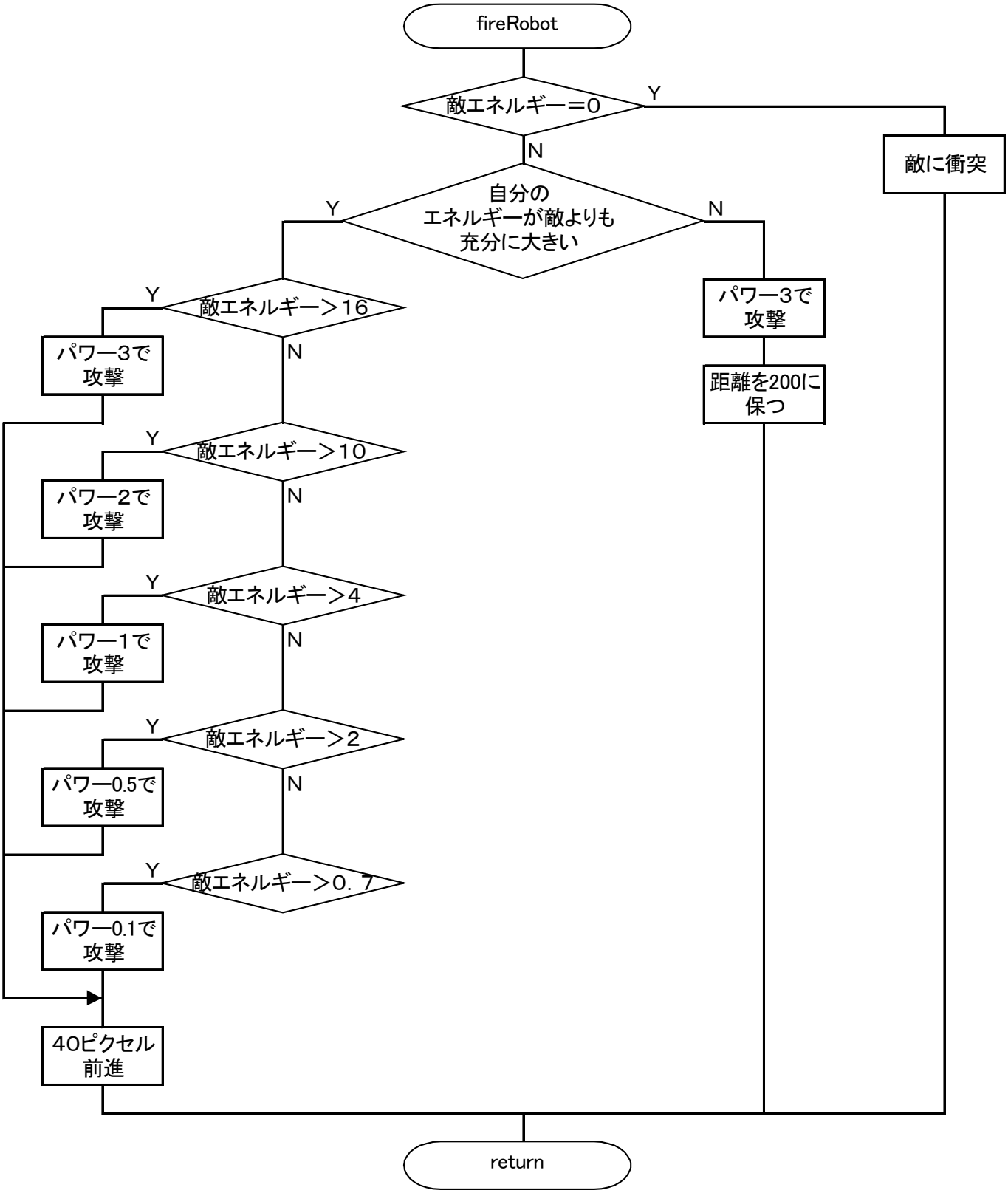

onRobotDeath
ロボット情報の削除
removeRobotInformation()
return

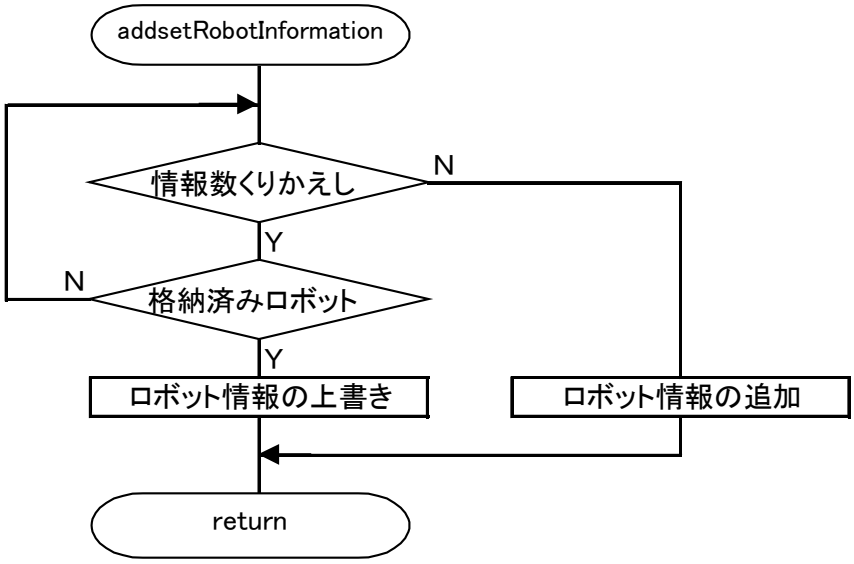

onCustomEvent
Y
"radarStop"
onRadarStop()
N
Y
"nearWall"
onNearWall()
N
Y
"pass45"
onPass45()
N
return

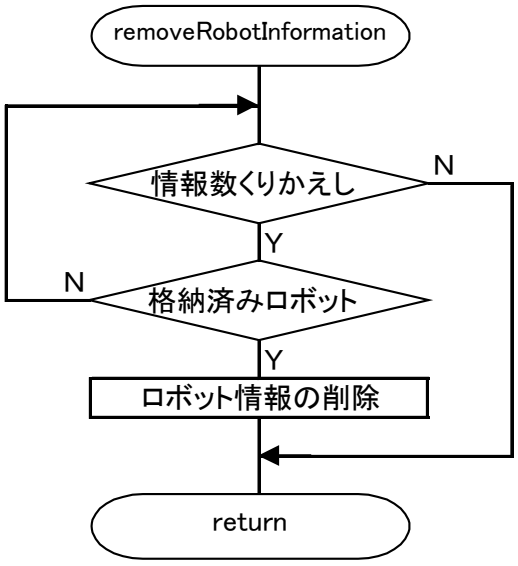

onRadarStop
レーダーの回転を持続
return

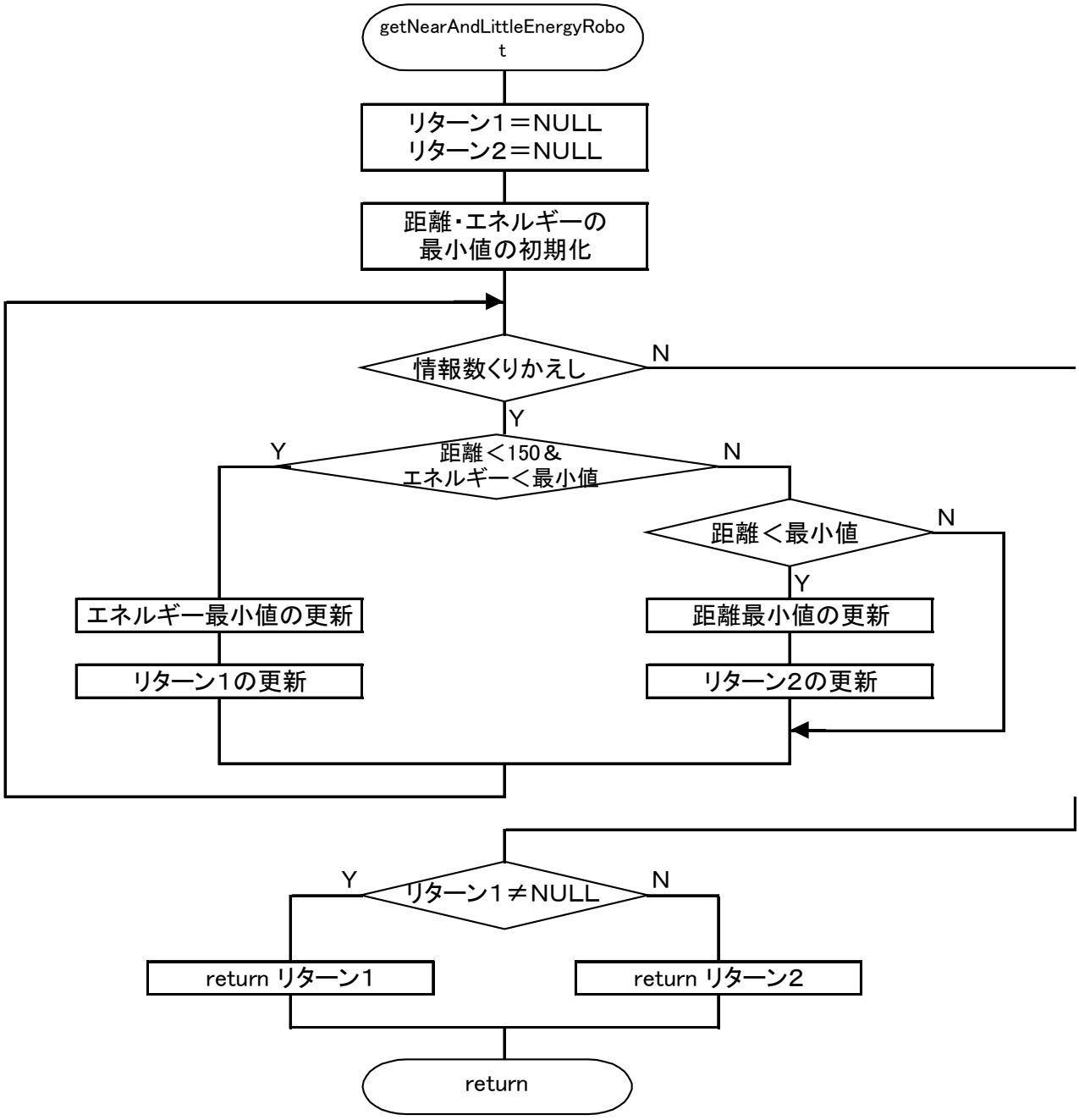

onNearWall
左の壁に近い
Y
右の壁を向く
N
右の壁に近い
Y
左の壁を向く
N
下の壁に近い
Y
上の壁を向く
N
上の壁に近い
Y
下の壁を向く
70ピクセル前進
return

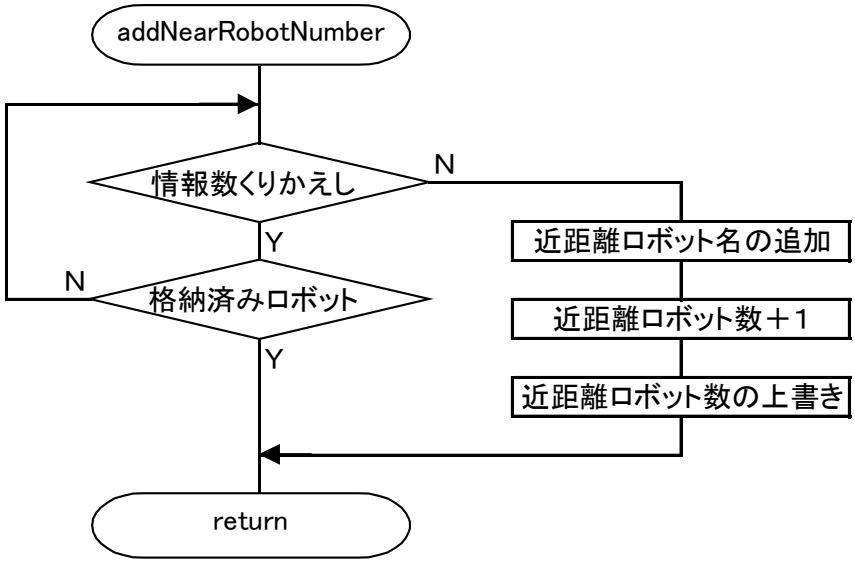

onPass45
近距離ロボット数の更新
ロボット数カウント変数の
ゼロクリア
レーダー回転制御変数の
更新
レーダー回転
角度≧360
N
Y
レーダー回転角度－360
近距離ロボット名情報
ベクトルのゼロクリア
return

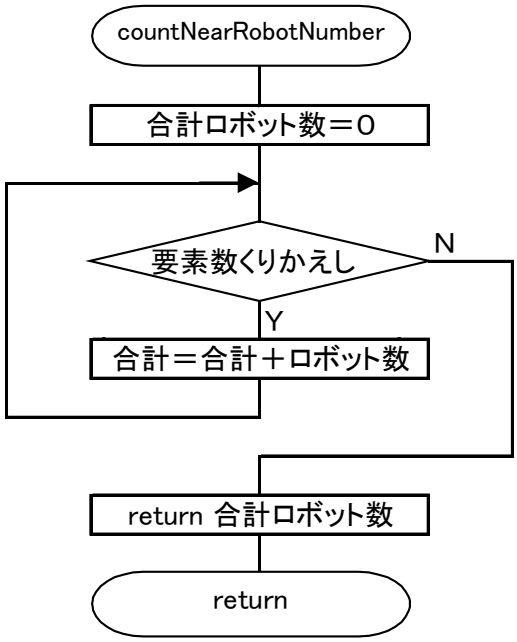

fireRobot
敵エネルギー＝0
Y
N
敵に衝突
自分の
エネルギーが敵よりも
充分に大きい
Y
N
パワー3で
攻撃
距離を200に
保つ
敵エネルギー＞16
Y
N
パワー3で
攻撃
敵エネルギー＞10
Y
N
パワー2で
攻撃
敵エネルギー＞4
Y
N
パワー1で
攻撃
敵エネルギー＞2
Y
N
パワー0.5で
攻撃
敵エネルギー＞0．7
Y
パワー0.1で
攻撃
40ピクセル
前進
return

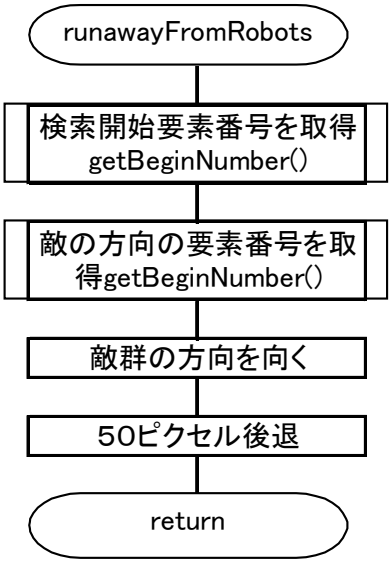

addsetRobotInformation
情報数くりかえし
N
Y
N
格納済みロボット
Y
ロボット情報の上書き
ロボット情報の追加
return

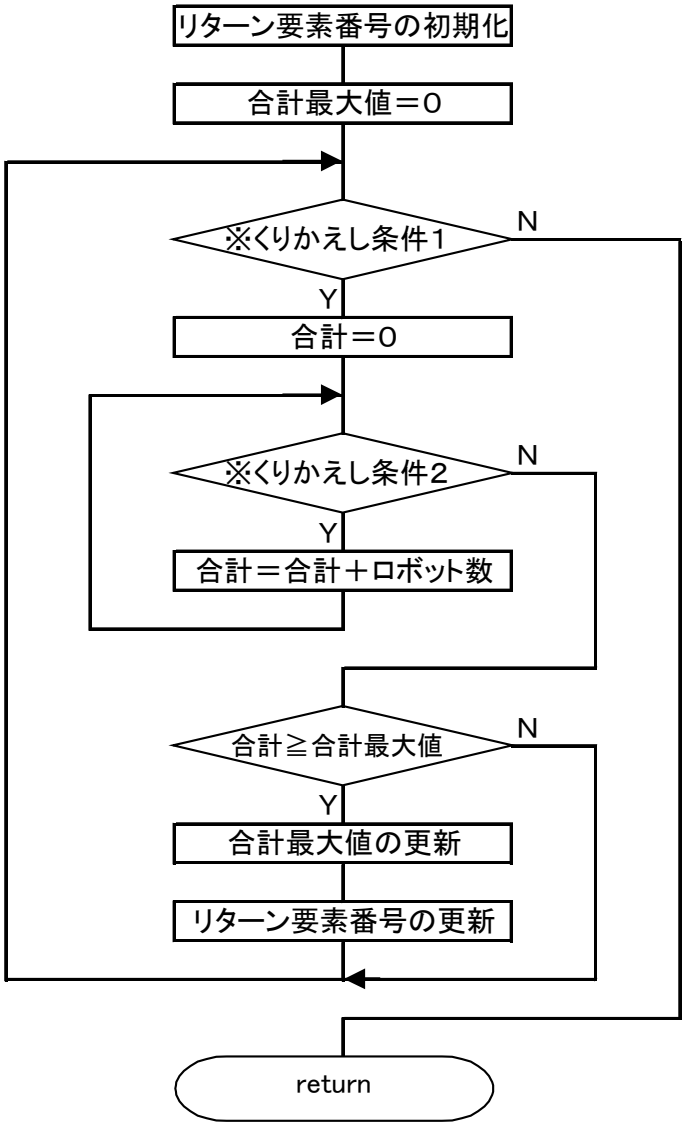

removeRobotInformation
情報数くりかえし
N
Y
N
格納済みロボット
Y
ロボット情報の削除
return


getNearAndLittleEnergyRobo
t
リターン1＝NULL
リターン2＝NULL
距離・エネルギーの
最小値の初期化
情報数くりかえし
N
Y
Y
距離＜150＆
エネルギー＜最小値
N
距離＜最小値
N
Y
エネルギー最小値の更新
距離最小値の更新
リターン1の更新
リターン2の更新
Y
リターン1≠NULL
N
return リターン1
return リターン2
return


addNearRobotNumber
情報数くりかえし
N
Y
近距離ロボット名の追加
N
格納済みロボット
近距離ロボット数＋1
Y
近距離ロボット数の上書き
return

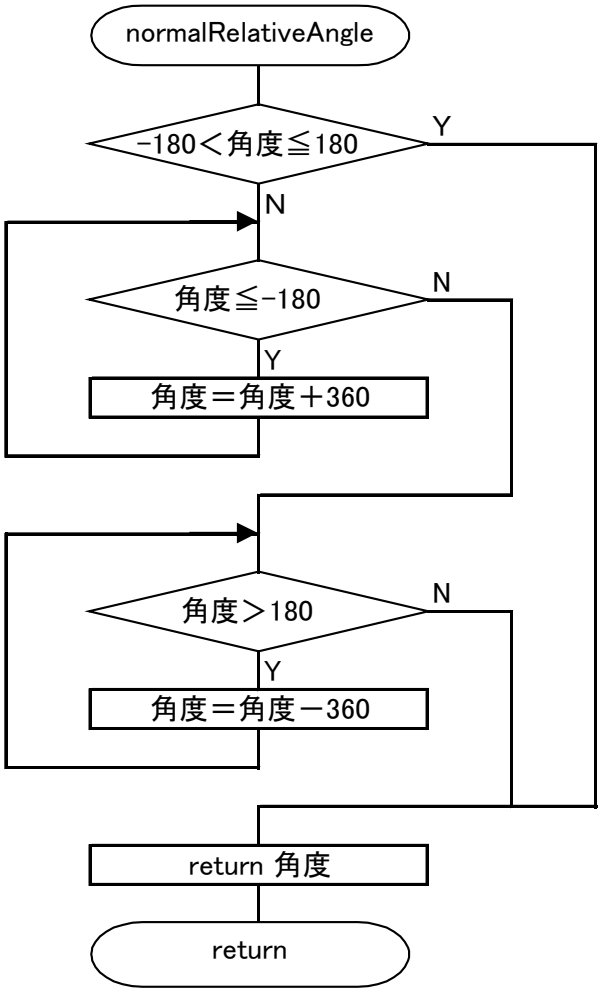

countNearRobotNumber
合計ロボット数＝0
要素数くりかえし
N
Y
合計＝合計＋ロボット数
return 合計ロボット数
return


runawayFromRobots
検索開始要素番号を取得
getBeginNumber()
敵の方向の要素番号を取
得getBeginNumber()
敵群の方向を向く
50ピクセル後退
return



※くりかえし条件1
　for (i=検索開始要素番号; i<=検索開始要素番号＋検索回数; i++)

※くりかえし条件2
　for (j=i; j<=i+検索幅; i++)


normalRelativeAngle
-180＜角度≦180
Y
N
角度≦-180
N
Y
角度＝角度＋360
角度＞180
N
Y
角度＝角度－360
return 角度
return

5. 6 プログラムリスト

```
/*-------------------------------------------------------------------------------------------------*
 *                                                                                                 *
 *       ファイル名   ：RoboInformationClass.java
*
 *       ファイル種別：クラス定義ファイル
*
 *       ファイル概要：ロボット情報クラスの定義
*
 *       言語         ：Java 言語                                                                   *
 *       完成日       ：2003.11.23.sun
*
 *       作成者       ：松橋あずさ                                                                  *
 *                                                                                                 *
*-------------------------------------------------------------------------------------------------*/

package am;
import robocode.*;

/*-------------------------------------------------------------------------------------------------*
 *       RobotInformationClass
*
*-------------------------------------------------------------------------------------------------*/
        class RobotInformationClass {

                private String sName;                                           //ロボットの名前
                private double dEnergy;                                 //ロボットのエネルギー
                private double dDistance;                               //ロボットの距離
                private double dBearing;                                //ロボットの方向

                public RobotInformationClass(Robot me , ScannedRobotEvent e) {
                        sName   = e.getName();
                        dEnergy  = e.getEnergy();
                        dDistance = e.getDistance();
                        dBearing  = normalAbsoluteAngle(me.getHeading() + e.getBearing());
                }
/*-------------------------------------------------------------------------------------------------*
 *       getName メソッド                                                  ロボットの名前を取得
*
*-------------------------------------------------------------------------------------------------*/
                public String getName() {
                        return sName;
                }
/*-------------------------------------------------------------------------------------------------*
```

```
 *       getEnergy メソッド                                                ロボットのエネルギーを取得
*
*-------------------------------------------------------------------------------------------------*/
                public double getEnergy() {
                        return dEnergy;
                }

/*-------------------------------------------------------------------------------------------------*
 *       getDistance メソッド                                              ロボットの距離を取
得              *
*-------------------------------------------------------------------------------------------------*/
                public double getDistance() {
                        return dDistance;
                }
/*-------------------------------------------------------------------------------------------------*
 *       getBearing メソッド                                               ロボットの方向を取得
*
*-------------------------------------------------------------------------------------------------*/
                public double getBearing() {
                        return dBearing;
                }
/*-------------------------------------------------------------------------------------------------*
 *       normalAbsoluteAngle メソッド                                      角度の変換              *
*-------------------------------------------------------------------------------------------------*/
                public double normalAbsoluteAngle(double angle) {

                        //角度を 0 から 360 の範囲に変換してリターン
                        if (angle >= 0 && angle < 360)
                                return angle;
                        double fixedAngle = angle;
                        while (fixedAngle < 0)
                                fixedAngle += 360;
                        while (fixedAngle >= 360)
                                fixedAngle -= 360;
                        return fixedAngle;
                }
        };
/*-------------------------------------------------------------------------------------------------*
 *       RobotInformationClass - End Of File -                                                     *
*-------------------------------------------------------------------------------------------------*/
```

```
/*-------------------------------------------------------------------------------------------------*
 *	ファイル名　：NearRobotClass.java                                                              *
 *	ファイル種別：クラス定義ファイル
*
 *	ファイル概要：近距離ロボット名情報クラスの定義                                                 *
 *	言語　　：Java 言語                                                                            *
 *	完成日　　：2003.11.23.sun
*
 *	作成者　　：松橋あずさ                                                                         *
*-------------------------------------------------------------------------------------------------*/

package am;
import robocode.*;

/*-------------------------------------------------------------------------------------------------*
 *	NearRobotClass                                                                                 *
*-------------------------------------------------------------------------------------------------*/
	class NearRobotClass {

		private String sName;                                          //ロボットの名前

		public NearRobotClass(String name) {
			sName = name;
		}
/*-------------------------------------------------------------------------------------------------*
 *	getName メソッド                                                  ロボットの名前を取得
*
*-------------------------------------------------------------------------------------------------*/
		public String getName() {
			return sName;
		}
	};
/*-------------------------------------------------------------------------------------------------*
 *	NearRobotClass - End Of File -
*
*-------------------------------------------------------------------------------------------------*/
```

```
/*-------------------------------------------------------------------------------------------------*
 *	ファイル名　：NearRobotNumberClass.java
*
 *	ファイル種別：クラス定義ファイル
*
 *	ファイル概要：近距離ロボット数情報クラスの定義　　*
 *	言語　　：Java 言語　　*
 *	完成日　　：2003.11.23.sun
*
 *	作成者　　：松橋あずさ　　*
*-------------------------------------------------------------------------------------------------*/

package am;
import robocode.*;

/*-------------------------------------------------------------------------------------------------*
 *	NearRobotNumberClass　　*
*-------------------------------------------------------------------------------------------------*/
	class NearRobotNumberClass {

		private int iNumber;　　//ロボット数

		public NearRobotNumberClass(int number) {
			iNumber = number;
		}
/*-------------------------------------------------------------------------------------------------*
 *	getNumber メソッド　　ロボット数を取得　　*
*-------------------------------------------------------------------------------------------------*/
		public int getNumber() {
			return iNumber;
		}
	};
/*-------------------------------------------------------------------------------------------------*
 *	NearRobotNumberClass - End Of File -　　*
*-------------------------------------------------------------------------------------------------*/
```

```
/*-------------------------------------------------------------------------------------------------*
 *       ファイル名   ：Robo711.java                                                               *
 *       ファイル種別：Robot ファイル                                                              *
 *       ファイル概要：Robo711 の戦術アルゴリズム
*
 *       言語         ：Java 言語                                                                  *
 *       完成日       ：2003.11.23.sun
*
 *       作成者       ：松橋あずさ                                                                 *
*-------------------------------------------------------------------------------------------------*/

package am;
import robocode.*;
import java.util.Vector;
import java.awt.Color;

/*-------------------------------------------------------------------------------------------------*
 *       Robo711Comment                                                                            *
*-------------------------------------------------------------------------------------------------*/

public class Robo711Comment extends AdvancedRobot
{
        int    iRadarHeading;                                    //レーダー回転制御変数
        int    iNumber;                                        //近距離ロボット数カウント変数
        Vector vRobotInformation = new Vector();                 //ロボット情報格納ベクトル

        Vector vNearRobot = new Vector();                       //近距離ロボット名格納ベクトル
        Vector vNearRobotNumber = new Vector();                  //近距離ロボット数格納ベクト
ル

/*-------------------------------------------------------------------------------------------------*
 *       run メソッド                                                  初期設定および移動ループ
*
*-------------------------------------------------------------------------------------------------*/
        public void run() {
                //優先順位の設定
                setEventPriority("RobotDeathEvent"   , 90);
                setEventPriority("HitRobotEvent"      , 80);
                setEventPriority("HitByBulletEvent"    , 70);
                setEventPriority("ScannedRobotEvent" , 60);
                setEventPriority("HitWallEvent"       , 10);
                setEventPriority("BulletHitEvent"     , 10);
                setEventPriority("BulletHitBulletEvent", 10);
                setEventPriority("BulletMissedEvent"  , 10);
/*-------------------------------------------------------------------------------------------------*/
```

```
		//ロボットの初期設定
		setColors(Color.white,Color.white,Color.white);
		setAdjustRadarForRobotTurn(true);
		setAdjustRadarForGunTurn(true);

/*-------------------------------------------------------------------------------------------------*/
		//レーダー回転制御変数と近距離ロボット数格納ベクトルの初期化
		iRadarHeading = 45;
		for(int i=0; i<=7; i++) {
			vNearRobotNumber.add(new NearRobotNumberClass(0));
		}
/*-------------------------------------------------------------------------------------------------*/
		//レーダーの回転を持続させるカスタムイベントの追加
		addCustomEvent(
			new Condition("radarStop") {
				public boolean test() {
					return (getRadarTurnRemaining() == 0);
				};
			}
		);
		//壁に近づかないようにするカスタムイベントの追加
		addCustomEvent(
			new Condition("nearWall") {
				public boolean test() {
					return (getX() <= 70 || getX() >= getBattleFieldWidth()  -
70 ||
						getY() <= 70 || getY() >= getBattleFieldHeight() -
70);
				};
			}
		);
		//近距離ロボット数の更新をするカスタムイベントの追加
		addCustomEvent(
			new Condition("pass45") {
				public boolean test() {
					return (getRadarHeading() >= iRadarHeading);
				};
			}
		);
/*-------------------------------------------------------------------------------------------------*/
		//ロボットの移動ループ
		while(true) {
			if(vRobotInformation.size() > 0) {
				//近距離かつ低エネルギーなロボットを攻撃
				RobotInformationClass e = getNearAndLittleEnergyRobot();
```

```
                                turnRight(e.getBearing() - getHeading());
                                        //大砲の熱がゼロなら攻撃
                                if(getGunHeat() == 0) {
                                        fireRobot(e);
                                }
                        }

                        else {
                        //スピンボットの移動
                        setTurnRight(10);
                        setMaxVelocity(5);
                        ahead(10);
                }
                        execute();
                }
        }
/*-------------------------------------------------------------------------------------------------*
 *       onScannedRobot メソッド                                   ロボット情報と近距離ロボット数の更新
*
*-------------------------------------------------------------------------------------------------*/
        public void onScannedRobot(ScannedRobotEvent e ) {
                //ロボット情報の更新
                addsetRobotInformation(e);
                //敵までの距離が近い場合 近距離ロボット数の更新
                if(e.getDistance() <= 100) {
                        addNearRobotNumber(e);
                        //近距離ロボット数が３体以上の場合 敵の群れから逃げる
                        if(countNearRobotNumber() >= 3) {
                                runawayFromRobots();
                        }
                }
        }
/*-------------------------------------------------------------------------------------------------*
 *       onHitRobot メソッド                                       敵のロボットを攻撃または移動
*
*-------------------------------------------------------------------------------------------------*/
        public void onHitRobot(HitRobotEvent e) {
                //敵の方を向く
                setTurnRight(e.getBearing());
                //自分のエネルギーが相手よりも充分に大きい場合 弾丸で攻撃し前進
                if (getEnergy() > 50 && getEnergy() > e.getEnergy() + 20) {
                        if (e.getEnergy() > 16)
                                fire(3);
                        else if (e.getEnergy() > 10)
                                fire(2);
```

```
                        else if (e.getEnergy() > 4)
                                fire(1);
                        else if (e.getEnergy() > 2)
                                fire(.5);
                        else if (e.getEnergy() > .7)
                                fire(.1);
                        ahead(40);
                        return;
                }
                //自分のエネルギーが小さい場合 その場から離れる
                if(getEnergy() < 15) {
                        back(20);
                        return;
                }
                //それ以外の場合 弾丸で攻撃し後退
                fire(3);
                back(20);
        }
/*-------------------------------------------------------------------------------------------------*
 *      onRobotDeath メソッド                                                ロボット情報の削除
*
*-------------------------------------------------------------------------------------------------*/
        public void onRobotDeath(RobotDeathEvent e) {
                //ロボット情報の削除
                removeRobotInformation(e);
        }
/*-------------------------------------------------------------------------------------------------*
 *      onCustomEvent メソッド                                          カスタムイベントの対応処理
*
*-------------------------------------------------------------------------------------------------*/
        public void onCustomEvent(CustomEvent e) {
                //カスタムイベントの対応処理
                if (e.getCondition().getName().equals("radarStop")) {
                        onRadarStop();
                }
                if (e.getCondition().getName().equals("nearWall")) {
                        onNearWall();
                }
                if (e.getCondition().getName().equals("pass45")) {
                        onPass45();
                }
        }
/*-------------------------------------------------------------------------------------------------*
 *      onRadarStop メソッド                                                レーダーの回転を持
続                                                                                              *
```

```
*-------------------------------------------------------------------------------------------------*/
        public void onRadarStop() {
                //レーダーの回転を持続
                setTurnRadarRight(10000);
        }
/*-------------------------------------------------------------------------------------------------*
 *      onNearWall メソッド                                                壁から離れる              *
*-------------------------------------------------------------------------------------------------*/
        public void onNearWall() {
                //壁から離れる
                if (getX() <= 70)
                        turnRight(normalRelativeAngle(90 - getHeading()));
                else if (getX() >= getBattleFieldWidth()  - 70)
                        turnRight(normalRelativeAngle(270 - getHeading()));
                else if (getY() <= 70)
                        turnLeft(normalRelativeAngle(getHeading()));
                else if (getY() >= getBattleFieldHeight() - 70)
                        turnRight(normalRelativeAngle(180 - getHeading()));
                ahead(70);
        }
/*-------------------------------------------------------------------------------------------------*
 *      onPass45 メソッド                                                近距離ロボット数の更新
*
*-------------------------------------------------------------------------------------------------*/
        public void onPass45() {
                //近距離ロボット数の更新
                vNearRobotNumber.set(iRadarHeading / 45 , new NearRobotNumberClass(iNumber));
                //ロボット数のゼロクリア
                iNumber = 0;
                //レーダー回転制御変数の更新
                iRadarHeading = iRadarHeading + 45;
                if(iRadarHeading >= 360) {
                        iRadarHeading = iRadarHeading - 360;
                        //近距離ロボット名ベクトルのクリア
                        vNearRobot.clear();
                }
        }
/*-------------------------------------------------------------------------------------------------*
 *      fireRobot メソッド                                                敵のロボットを攻撃        *
*-------------------------------------------------------------------------------------------------*/
        public void fireRobot(RobotInformationClass e) {
                //敵のエネルギーがゼロの場合 衝突
                if(e.getEnergy() == 0) {
                        ahead(e.getDistance() + 5);
                        return;
```

```
                }
                //自分のエネルギーが相手よりも充分に大きい場合 弾丸で攻撃し前進
                if(getEnergy() > 50 && getEnergy() > e.getEnergy() + 20) {
                        if (e.getEnergy() > 16)
                                fire(3);
                        else if (e.getEnergy() > 10)
                                fire(2);
                        else if (e.getEnergy() > 4)
                                fire(1);
                        else if (e.getEnergy() > 2)
                                fire(.5);
                        else if (e.getEnergy() > .7)
                                fire(.1);
                        ahead(40);
                        return;
                }
                //それ以外の場合 弾丸で攻撃
                fire(3);
                setAhead(e.getDistance() - 200);
        }
/*-------------------------------------------------------------------------------------------------*
 *      addsetRobotInformation メソッド                                          ロボット情報の更新
*
*-------------------------------------------------------------------------------------------------*/
        public void addsetRobotInformation(ScannedRobotEvent e) {
                RobotInformationClass temp;                     //ロボット情報のインスタンス
                //ロボット情報の検索
                for(int i=0; i<vRobotInformation.size(); i++) {
                        temp = (RobotInformationClass)vRobotInformation.get(i);
                        //すでに格納済みのロボットの場合 ロボット情報の上書き
                        if(temp.getName().equals(e.getName())) {
                                vRobotInformation.set(i , new RobotInformationClass(this , e));
                                return;
                        }
                }
                //それ以外の場合 ロボット情報の追加
                vRobotInformation.add(new RobotInformationClass(this , e));
        }
/*-------------------------------------------------------------------------------------------------*
 *      removeRobotInformation メソッド                                          ロボット情報の削除
*
*-------------------------------------------------------------------------------------------------*/
        public void removeRobotInformation(RobotDeathEvent e) {
                RobotInformationClass temp;                     //ロボット情報のインスタンス
                //ロボット情報の検索
```

```
                for(int i=0; i<vRobotInformation.size(); i++) {
                        temp = (RobotInformationClass)vRobotInformation.get(i);
                        if(temp.getName().equals(e.getName())) {
                                //ロボット情報の削除
                                vRobotInformation.remove(i);
                                return;
                        }
                }
        }
/*-------------------------------------------------------------------------------------------------*
 *      getNearAndLittleEnergyRobot メソッド                    近距離かつ低エネルギーなロボットの取得
*
*-------------------------------------------------------------------------------------------------*/
        public RobotInformationClass getNearAndLittleEnergyRobot() {
                RobotInformationClass temp;                     //ロボット情報のインスタンス
                RobotInformationClass return1st = null , return2nd =null;  //リターン用のインスタンス
                double dMinDistance = Double.MAX_VALUE , dMinEnergy = Double.MAX_VALUE;  //距離の最小
値
                //ロボット情報の検索
                for(int i=0; i<vRobotInformation.size(); i++) {
                        temp = (RobotInformationClass)vRobotInformation.get(i);
                        //敵のロボットまでの距離を比較
                        if(temp.getDistance() < 150 && temp.getEnergy() < dMinEnergy){
                                dMinEnergy = temp.getEnergy();
                                return1st = temp;
                        }
                        else if(temp.getDistance() < dMinDistance) {
                                dMinDistance = temp.getDistance();
                                return2nd = temp;
                        }
                }
                //近距離のロボット情報をリターン
                if(return1st != null) {
                        return return1st;
                }
                else {
                        return return2nd;
                }
        }
/*-------------------------------------------------------------------------------------------------*
 *      addNearRobotNumber メソッド                                近距離ロボット数の更新      *
*-------------------------------------------------------------------------------------------------*/
        public void addNearRobotNumber(ScannedRobotEvent e) {
                NearRobotClass temp;                                //近距離ロボット名のインスタ
ンス
```

```
                //近距離ロボット名の検索
                for(int i=0; i<vNearRobot.size(); i++) {
                        temp = (NearRobotClass)vNearRobot.get(i);
                        //格納済みのロボットの場合 リターン
                        if(temp.getName().equals(e.getName())) {
                                return;
                        }
                }
                //近距離ロボット名の追加
                vNearRobot.add(new NearRobotClass(e.getName()));
                //近距離ロボット数の上書き
                iNumber++;
                vNearRobotNumber.set(iRadarHeading / 45 , new NearRobotNumberClass(iNumber));
        }



/*-------------------------------------------------------------------------------------------------
 *      countNearRobotNumber メソッド                                        近距離ロボット数の
カ              ウ              ン              ト                                              *
*-------------------------------------------------------------------------------------------------*/
        public int countNearRobotNumber() {
                NearRobotNumberClass temp;                      //近距離ロボット数のインスタンス
                int iNearRobotNumber = 0;                       //近距離ロボット数の総和を求める変数
                //近距離ロボット数の検索
                for(int i=0; i<=7; i++) {
                        temp = (NearRobotNumberClass)vNearRobotNumber.get(i);
                        iNearRobotNumber = iNearRobotNumber + temp.getNumber();
                }
                //近距離ロボット数の総和をリターン
                return iNearRobotNumber;
        }
/*-------------------------------------------------------------------------------------------------*
 *      runawayFromRobots メソッド                                      敵の群れから逃げる        *
*-------------------------------------------------------------------------------------------------*/
        public void runawayFromRobots() {
                //近距離ロボット数から敵の群れの方向(ベクトル開始番号)を求める
                int iBeginNumber = getBeginNumber(getBeginNumber(0,7,5),3,3);
                //敵の群れから逃げる
                double turnRobotAmt = normalRelativeAngle(iBeginNumber * 45 - getHeading());
                turnRight(turnRobotAmt);
                back(50);
                scan();
        }
```

```
/*-------------------------------------------------------------------------------------------------*
 *      getBeginNumber メソッド                                          ベクトル開始番号の取得
*
*-------------------------------------------------------------------------------------------------*/
        public int getBeginNumber(int a , int b , int c) {
                NearRobotNumberClass temp;                      //近距離ロボット数のインスタンス
                int iSum , iSumMax = 0;                         //総和を求める変数
                int iBeginNumber = -1;                          //ベクトル開始番号の変数
                //近距離ロボット数から敵の群れの方向を求める
                for(int i=a; i<=a+b; i++) {
                        iSum = 0;
                        for(int j=i; j<=i+c; j++) {
                                if(j<=7) {
                                        temp = (NearRobotNumberClass)vNearRobotNumber.get(j);
                                }
                                else {
                                        temp = (NearRobotNumberClass)vNearRobotNumber.get(j-8);
                                }
                                iSum = iSum + temp.getNumber();
                        }
                        if(iSum >= iSumMax) {
                                iSumMax = iSum;
                                if(i<=7) {
                                        iBeginNumber = i;
                                }
                                else {
                                        iBeginNumber = i-8;
                                }
                        }
                }
                //近距離ロボット数のベクトル開始番号をリターン
                return iBeginNumber;
        }
/*-------------------------------------------------------------------------------------------------*
 *      normalRelativeAngle メソッド                                              角度の変換        *
*-------------------------------------------------------------------------------------------------*/
        public double normalRelativeAngle(double angle) {
                //角度を-180 から 180 の範囲に変換してリターン
                if (angle > -180 && angle <= 180)
                        return angle;
                double fixedAngle = angle;
                while (fixedAngle <= -180)
                        fixedAngle += 360;
                while (fixedAngle > 180)
                        fixedAngle -= 360;
```

```
                return fixedAngle;
        }
}
/*-------------------------------------------------------------------------------------------------------*
 *       Robo711Comment     - End Of File -                                                              *
*-------------------------------------------------------------------------------------------------------*/
```

6. 考察

作成したロボットでバトルを実行すると、仕様通りの動きをすることが確かめられた。
今後の課題は、敵ロボットの攻撃をかわすために予測不可能な動きをさせることや、弾丸の速度や到達時間を考慮すること、などが挙げられる。

参考までに、Robocodeの代表的な戦術アルゴリズムを以下に示す。(調査結果)

■Pattern Matching
敵の動き(角速度、速度変化)をログとして保存し、現在の動きが過去のログにある程度マッチしたら、アルゴリズム的に似たような軌跡を辿ると仮定して軌跡を先読みし、弾を打つ。

■Guess Factor Aiming
敵に対して弾を撃ち、目標位置に弾が到着するまでの時間に敵が動ける範囲は、多くとも敵がまっすぐ最大速度で前後に走った範囲であるから、その範囲で存在位置の統計を取って存在確率の高いところに弾を撃ち、平均より高い確率で弾を当てる。

■反重力移動
フィールドの任意の位置(敵の位置など)に斥力を配置し、ロボットが受ける斥力をベクトルで合成して、そのベクトルに従順にロボットを動かすことで、柔軟に敵ロボットを避ける。

7. 感想

■Robocodeについて
強いロボットの条件は

敵の裏をかくアイディア ＋ プログラミング技術 ＋ 結果に基づく試行錯誤

であり、そのなかでも試行錯誤を何度も繰り返すことで、それまで見えていなかったことに気付いたり、新しいアイディアが浮かぶのではないかと思います。
また、より高度なアルゴリズムには距離・時間・速さなどの物理的概念や、高度なプログラミング技術が必要となるので、Robocodeは奥の深い実験内容だと感じました。

■プログラミングについて
他人にわかりやすいプログラムは、第一に自分が見てわかりやすいプログラムでなければならない、ということを実感しました。
また、データ変数名の工夫や適切なコメントを加えるだけでも、数段わかりやすいプログラムになることが実験を通してよくわかりました。
報告書の作成にあたっては、データフローやデータディクショナリなど、これまで作成したことのないものを作ることができてよかったです。

■その他
Linaxや、Java言語、HPの作成など、新しいことに挑戦できてよかったです。