目次

[コーディング作業のやり方 2](#_Toc530694266)

[デバッグの方法 2](#_Toc530694267)

[基本な関数の説明 3](#_Toc530694268)

[敵ロボットの情報 5](#_Toc530694269)

[ラジアン表記について 5](#_Toc530694270)

[Util.javaについて 6](#_Toc530694271)

[レーダーの基本 7](#_Toc530694272)

# コーディング作業のやり方

標準のRobot Editorを使って編集作業をしてもいいのですが、使いづらいので自分は　visual studio codeで編集 > Robot Editorでコンパイル　という方法を取っています。（テキストエディタはなんでもいいです）

初めて作ったjavaファイルのコンパイルはもちろん必要ですが、それ以降は例えばG13\_Leader.javaだけをコンパイルすれば、そこで使われているクラスはすべてコンパイル対象になっているようで、G13\_Leader.javaだけをRobot Editorでコンパイルすればそこで使われているクラスも含めてコンパイルされるみたいです。G13\_Sub1.javaでも同様。

＊コメントは極力書くようにしよう

# デバッグの方法

デバッグの方法として2種類あります。

１．コンソールに文字列を表示

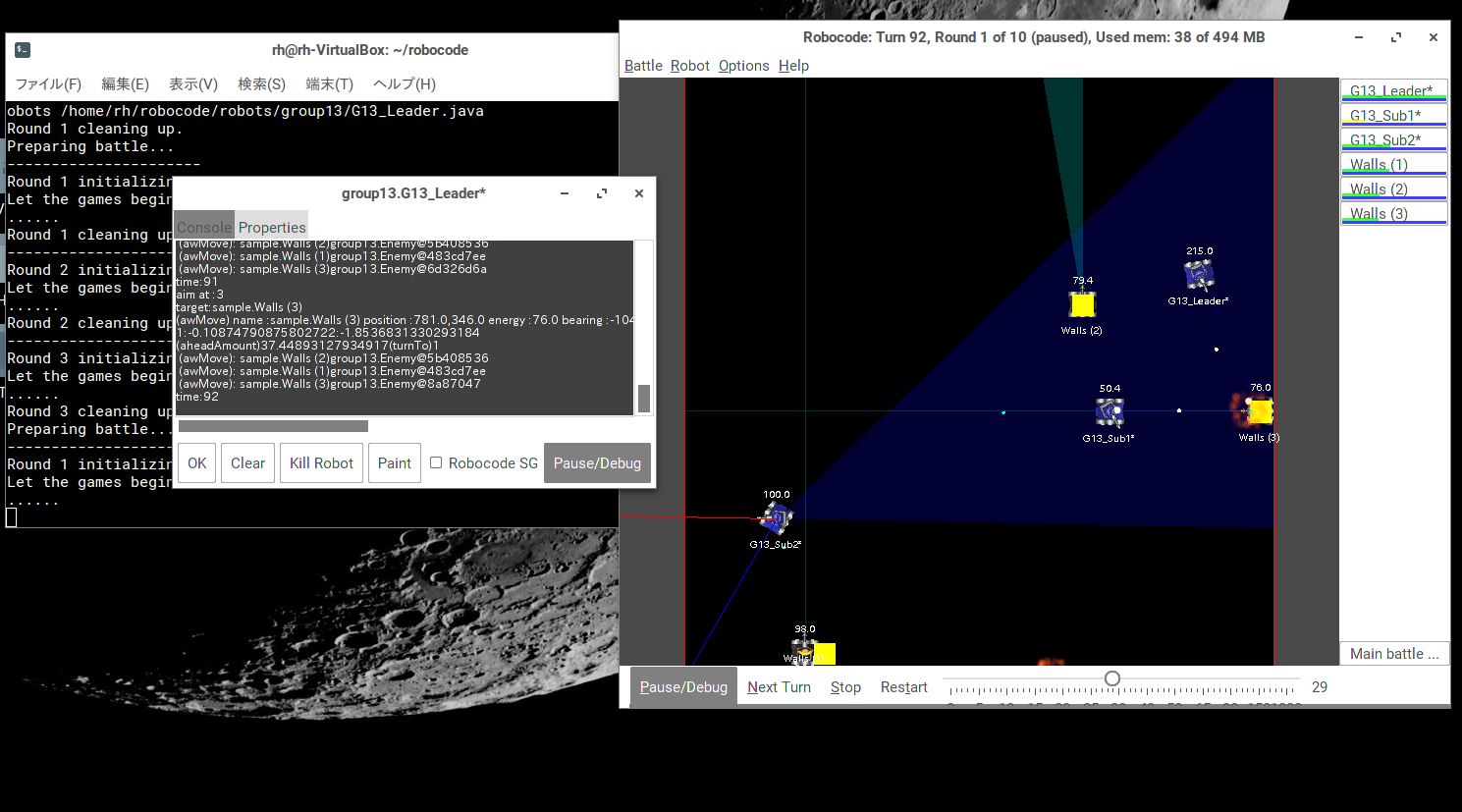
画面右側のロボットの名前をクリックすると、そのロボットの情報を表示するコンソールが出てきます。ここにSystem.out.println()で情報を表示することができます。

例えば、runのwhile文内でSystem.out.println(getTime())で試合開始からの時刻を表示しています

２．グラフィカルデバッグで直接描画

各ロボットのコンソールの左から4つ目にPaintと書かれたボタンを押すとonPaint()内で書かれた描画命令が呼ばれます。（後述）

例えば、自分の向いている向きに直線を引く、レーダーで取得した相手の位置に正方形を表示するなどしています。（下のスクショ参考）



# 基本な関数の説明

G13\_Leader.javaで説明します。（かなり簡素化して書いています）

|  |
| --- |
| package group13;  public class G13\_Leader extends TeamRobot  {  //run()が実際の動作で呼ばれる      public void run() {    //setAdjust関数　現在antiWall.javaの関係で上記２つのみ使用          setAdjustGunForRobotTurn(true); //ロボットの回転に対して砲台の回転を無効          setAdjustRadarForGunTurn(true); //砲台の回転に対してレーダーの回転を無効          //setAdjustRadarForRobotTurn(true); //ロボットの回転に対して砲台の回転を無効            //whileでは毎ターン実行する関数を呼び出します          while(true) {  //ロボットの動作はこれだけです。マイナスの値を入れれば逆の動作になります。（setahead(-1)ならバックする）実際はこれらの動作を他のオブジェクトで実行　　　　　させることになります  setTurnRadarRight(double 回転量); //レーダーを回す  setTurnGunRight(double 回転量);　//砲台を回す  setTurnRight(double 回転量);　//ロボットを回す  setahead(double 移動量); //ロボットを前進させる  setfire(double パワー); //撃つ          }      }    //以下のon~ と名前がついた関数は何かが発生したときのみ呼ばれます。  //レーダーで敵を補足したときに呼ばれる      public void onScannedRobot(ScannedRobotEvent e) {      }        public void onHitByBullet(HitByBulletEvent e) {      }      public void onHitWall(HitWallEvent e) {      }      public void onRobotDeath(RobotDeathEvent e){      }    //味方のロボットからのメッセージを受け取ったときに呼ばれる      public void onMessageReceived(MessageEvent e){      }      //実際の画面上に描画することができる（graphical debbuging）  ここではjava.awt.Graphics2Dの関数を使用します  public void onPaint(Graphics2D g) {  // (例)レーダで取得した敵の座標の場所に四角形を表示        g.fillRect((int)enemy.x - 20, (int)enemy.y - 20, 40, 40);  }  } |

# 敵ロボットの情報

敵の情報は以下の通りname,role以外はそのスキャン時の情報となる。（レーダーでスキャンしない限り更新されない）

あとMap<String, Enemy>という構造になっているので、例えば敵の情報を取得したいときは

Enemy enemy = enemyMap.get(name);

という書き方になります。

そのほかは下記参照

https://www.sejuku.net/blog/19796

|  |  |
| --- | --- |
| 変数 | 意味 |
| name | 名前 |
| x | ｘ座標 |
| y | ｙ座標 |
| bearing | 自分のheadingの向きから見た敵の相対角度 |
| distance | 敵までの距離 |
| heading | 敵の向き（戦車に前後がある、ここでは戦車の前進の向き） |
| headingRadians | 敵の向き（ラジアン表記） |
| velocity | 速さ |
| energy | HP |
| role | 役割(リーダー　一般機　ドロイド) |
| heat | 砲塔の熱（冷やさないと撃てない 0になると撃てる） |
| scanned | 1度スキャンされたか |
| alive | その時点で生きているか |

# ラジアン表記について

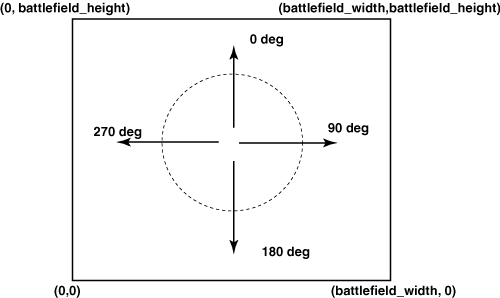
回転量を代入する関数には360度またはラジアン(π)で代入できる2種類存在します。

（例）

setTurnRight()

setTurnRightRadian()

ラジアンの方が計算しやすいかなとか思ったりしますが、一つだけ注意が必要です。robocodeがどういう座標系を用いているかは以下の図で分かります。



座標自体はいつもの(x,y)のですが、回転方向が逆です。（なんで右回り…）

これに対する対処は簡単でsin,cosを入れ替えるだけで済みます

（例）

極座標表示は本来

x =ｄcosθ y = d sinθ

のはずですが、robocodeでは

x = d sinθ y = d cosθ

# Util.javaについて

複数回用いそうな関数についてはUtil.javaに記述してあります。例えば、2点間の距離を求める関数はよく使いそうなのでUtil.javaに入っています。

基本的にチーム開発では、１人1クラスを担当することが望ましいと思われますが、Util.javaに関しては融通が利くように特に相談する必要があるかもしれません

# レーダーの基本

1度レーダーでスキャンした敵の方向にレーダーを向け続けたい場合は以下のコードでできます

|  |
| --- |
| onScannedRobot(ScannedRobotEvent e){  ……  radarTurnAmount=Utils.normalRelativeAngleDegrees(my.heading+my.bearing- my.radarHeading);  setTurnRadarRight(radarTurnAmount);  } |

※Utils.normalRelativeAngleDgreesはrobocode.util.Utilsの中の関数で正規化をするだけの関数

基本的にこれだけでもレーダーを相手の動きに追従することができますが、その間も他の敵もスキャンするために、敵をレーダーの範囲内に収めながら少しだけ左右に振るという動作を実現させたいところです。