基礎工学 PBL 最終レポート

アドバイザ 松下　誠

8 班

学籍番号	氏名	電子メールアドレス
09B17039	高瀬　真樹	u719918e@ecs.osaka-u.ac.jp
09B17048	槌道　慎也	u963910f@ecs.osaka-u.ac.jp
09B17049	釣谷　周平	u354018b@ecs.osaka-u.ac.jp
09B17054	NANDEDKAR PARTH SHIRISH	u956380b@ecs.osaka-u.ac.jp
09B17070	水止　萌奈	u852594d@ecs.osaka-u.ac.jp

第 I 部
全体の作業計画

計画日	実際	作業内容	担当者
10/14	10/14	Robo wiki を見て考えてきた戦略の共有	全員
10/22	10/22	プログラムを書くために必要な情報の収集	全員
10/27	10/27	アドバイザーの意見を受けて戦略の見直し	全員
11/6	11/6	元にするプログラムの読解と実行、開発環境の整備	全員
11/16	11/16	戦術の細かい仕様の決定	全員
11/25	11/25	中間レポート提出	釣谷
12/2	12/2	ram と反重力の開発	パルト
12/9	12/9	wavesurfing、guessfactor 戦術のプログラムの作成	釣谷
12/9	12/9	チーム戦術のプログラムの作成	高瀬・樋道
12/14	12/14	guessfactor のテスト	釣谷
12/17	12/17	プログラムの動作確認を済ませて提出	釣谷
12/19	12/20	プレゼン提出	水止
1/2	1/2	円形予測を元にする射撃戦略の発案	釣谷
1/7	1/7	PBL 用のパソコンに.class ファイルが作成されないエラーの解決	樋道
1/13	1/13	円形予測の完成	釣谷
1/13	未完	Arraylist を用いた情報格納のプログラムの完成	パルト
1/23	1/23	スライドをまとめて提出	樋道

表 1　当初の作業計画と実際の作業状況

- 実際の進歩状況はどうだったか
  → GuessFactor のパフォーマンスが想定していたよりも低く、実用が厳しかったこと以外はほぼ計画通りで問題なし
- 作業計画の良かった点
  →毎回日付や担当者を細かくきめていた点→提出物は大体提出日の二日前に完成予定としていた点
- 作業計画の悪かった点
  →提出担当者をはっきり決めておらず、一度提出が遅れてしまった点

また、各々の担当は以下のようになっていた。
釣谷 ： 中間レポートの提出、wavesurfing と guessfactor のプログラムの作成、円形予測を元にする射撃戦略の発案
樋道 ： チーム戦術のプログラムの作成、PBL 用のパソコンに.class ファイルが作成されないエラーの解決
高瀬 ： チーム戦術のプログラムの作成
水止 ： プレゼンの提出・発表
パルト： ram と反重力の開発、円形予測の完成、Arraylist を用いた情報格納のプログラムの完成

第 II 部
作成したロボットについて

以下、移動の部分では章/節を参照するとき、第 2 部の「2」は接頭として使う。

1　全体の戦略

今回の PBL を通じて、robocode の様々な戦略、戦術を調べ、実践してきた。その中でも、我々の全体の戦略「いっぱい当てて、いっぱい避ける」を軸として、移動における ram 戦術や、線形、円形予測を元にした射撃などの様々な戦略、戦術を採用した。その結果、walls チームと戦わせた際のスコアを 90:10 まで高めることが出来、最終戦においても第 3 位になることが出来た。以下では最終戦で用いた戦略を移動、レーダー、射撃、チーム戦略の順に紹介していく。

1.1　移動

我々のチームの標語は「いっぱい当てて、いっぱい避ける」であるが、Robocode 経験者 (そして他の班) は「この考えは自明であろう」と思う可能性が高い。しかし、コーディングの世界において「攻撃」と「回避」に相関がなく、それぞれのソースコード及びそれぞれの移動・放射命令も並列に実行されるため、個々に考える必要がある。チームの各ロボットのベースとなり、回避時・攻撃時のそれぞれの移動戦略を考え出すべきである。実際、移動は次節以降の「射撃」、「チーム戦略」の基本ともなるため、2 章の最初に解説する。

「攻撃」とは「射撃によって行われるもの」と考えられることが多いと思われる。実は、本レポートに説明するように、Robocode 上の Melee 戦 (複数戦車のチーム戦) では移動自体を武器にする Ram 戦術が流行している。さらに、射撃するときの位置、速度および車体の向きが射撃の手法と同等に重要である。より具体的に、「移動の戦略」とは HP の多い Leader であれ、Sub であれ、いずれの戦車に対しても殆ど同様なものであり、戦時に常に実行されるソースコードである。前記のように、この移動のコードは通常の (特定の条件に合致しない場合) 移動の命令およびレーダから敵が見えた時や放射準備などの条件が満たされた時の移動命令に分けられる。例えば、先に述べた Ram 戦術は 8 班のロボットの主な戦略、「体当たり」の実現であり、これはレーダーが敵を捉えている時に実行される。一方、反重力移動は無条件に行われる戦術であり、「敵をレーダで捉えていない」時に実行される。以降は 8 班の移動の戦術の元となる戦略の考え方、及びその手法の概念について説明する。

本 PBL 授業の対戦仕様から分かるように、3 対 3 対 3 の形式は Melee 戦の一類になっている故、Melee 戦において効果を発揮する戦略が効果を持つか否かを以下の**チェックポイント (方針)** を通して分析した。

1. 敵の射撃戦略によって自機の位置が簡単にばれないこと。いわゆる**「受動的」な回避**。これは敵が弾を撃った後の回避でなく、敵の射撃用の計算を誤るようにさせる移動の実装のことである。例えば、移動経路が複雑なほど、「線型的推測」や「円形的推測」およびそれに基づいた平凡な射撃戦略は無効となることが知られている。
2. 　Team.SampleWalls の各戦車は壁に近い敵を狙うため、戦場の**壁から離れる戦略**。以下の図 1 の左上ように、Walls は常に角から近くの敵にレーダをロックし、3 パワーの大きな弾で攻撃するため、8 班のオレンジ色ロボットは壁から離れている。Walls は 6 体の敵の内、3 体であり、その戦術も既知である。よって、動作テストを行うことにより Walls が行う射撃は想定しやすく、テストして見ると Walls は角に到着した後のみ放射する。Walls のレーダは壁に沿った角度から始まり、レーダを回すときは壁や角に近い敵が襲われる可能性が高い。Walls からの攻撃発生の確率、及びその正確さを同時に減少できるため、これは戦場の壁を避ける理由の 1 つである。

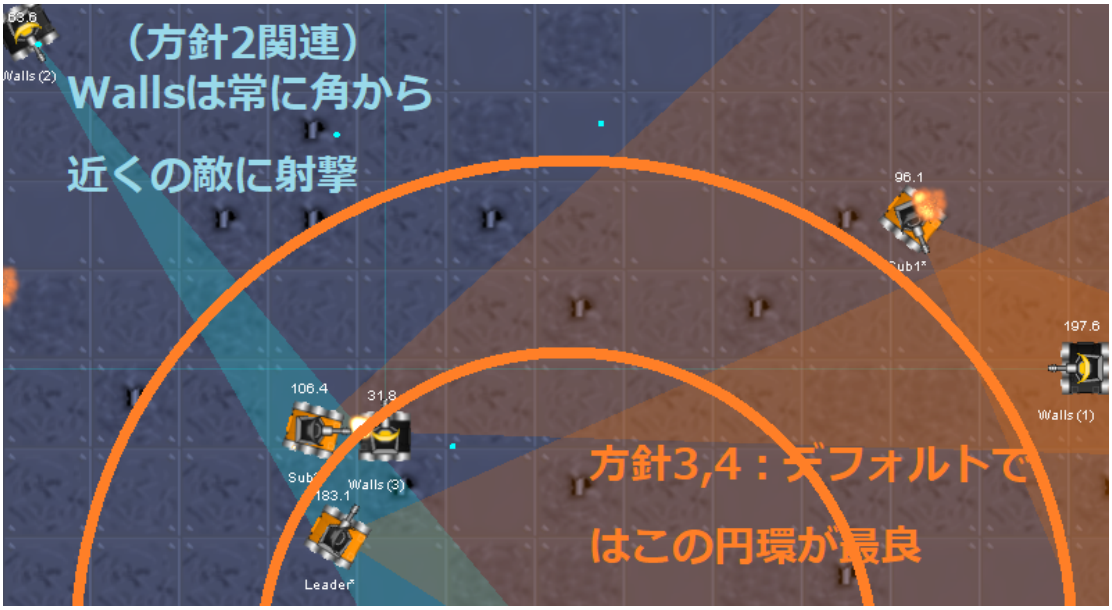


図 1　8 班はオレンジ色。方針 2 に記載の Walls の特徴、方針 3・4 を指摘する最良領域

3. 上記に加えて、**壁に当たるとダメージを受ける**ため、壁を避ける。実際、Robocode の戦時の規則を見れば、速度に応じたダメージ (＝ロボット速度/(2 ×エネルギー)) を受けることがあり、このような場合を避ける。
4. 敵が見つかったときに追跡する。それ以外は上記 *3* 点に加え、戦場の真ん中も避ける。上記の方針 3 と併せると、攻撃していない間に、図 1 内の円環領域が最も安全として考えられる (図 1 は戦場の上半である)。これらの理由として、800 × 800 の戦場は 9 体のロボットに対し、**かなり窮屈**である (Melee では 10 体で 1200 × 1200 が標準で 2.25 倍大きい、敵までの距離は通上 400px 以下)。それ故、**敵を追跡しやすい**。さらに、真ん中のロボットが弾で襲われる確率は非常に大きく、回避する時間も比較的短い。特に、次点に紹介される Ram 戦略に対し、無防備の状態になる。
5.「いっぱい当てる」のもう一つの解釈、敵にぶつかり (Ram)、大きな被害を受けさせる。前述及び以下の図 2 から分かるよう、初期状態を含め、敵までの距離は平均的にたった 200px 程度であると考えられる。しかも、敵が 6 体いて、最寄りの敵までの距離はこれよりも小さく、敵に寄りやすい。そのため、攻撃を「射撃」のみに限らず、いわゆる「Ram」(胴突き) も効果的に見える。2.1.2 節 (射撃関連の情報もあり) にこの戦術の仕様および具体的な実装方法を記載する。

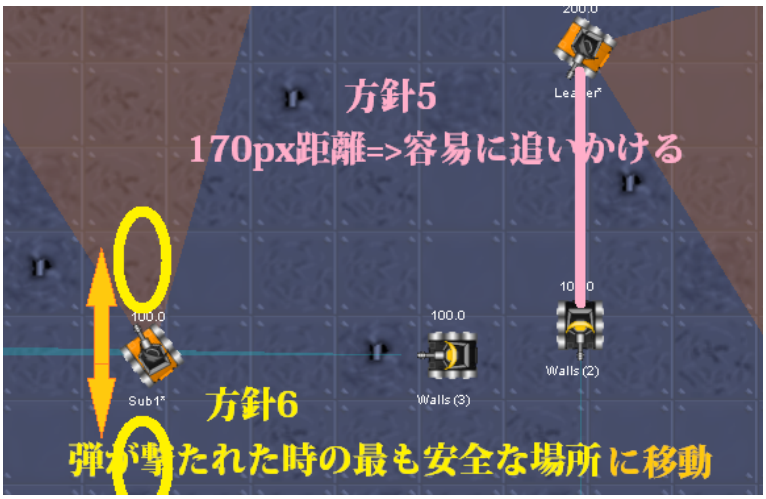


図 2　追跡戦略 (方針 5) の容易さ、能動的回避の目的移動

6. レーダに見える敵が放った弾を回避する特別な仕組みも必要。図 2 の下部 (黄色) に表示したいわゆる**「能動的」回避**。上記の点 1 から 5 までの目的動作は常に (条件なしで) 働く様々な移動命令と対応する動作であるように解釈できる。しかし、既に定められた 1 通りの動作に限らず、戦場上の現状も踏まえた移動戦略はこの静的・受動的戦略の発展となる。最新の情報を考慮する動的な移動の最も自然なものは目前の敵の位置および意思のデータ (Robocode の場合、放射済みか否か、敵のエネルギー、向きなど) を元にし、最も安全な動きを行う方針が挙げられる。

上記の全てのチェックポイントを満たす 1 つの移動戦略を考案するのではなく、それぞれの内容に応じて、2 つから 3 つの戦略の工夫が必要と考えられる。上記の方針は Robowiki 等のサイト上の Melee 戦略に関する情報や robocode の歴史も基にしている (参考文献 1)。実際、上記に加えて「誤射を避けるため、味方の戦車がよく散らばるようにする」という方針も考えられたが、2.1.1 節に説明される理由を元にし、省かれた。これらを踏まえて、上記の方針を満たす以下の 3 つの移動戦術が実装された。

- 回避時の「壁からの反重力」。方針 1，2，3 を活用。2 章「作成ロボットの仕様」の 2.1.1 項のように実装済み。
- 攻撃時の「体当たり *(Ram)* 戦術」。方針 4，5 を実現。2.1.2 項のように実装済み。
- 回避時の「*Wavesurfing*」。方針 6 の「能動回避」。実装未完了。2.1.3 項に概略を説明する。

1.2　レーダー

レーダ戦略としては、Infinity-Lock というものを採用する。Infinity-Lock の概要としては、 レーダを 1 周 (360 度) 回しながら、敵機を発見後すぐに折り返す (レーダーを 2 度当てる)。その後、その敵機を中心に 45 度を往復しつづけ、加速度、方向を計算しながら近づく。というものである。この戦略を採用した理由としては、狙っている 敵 1 体に時間をかけ詳しく観察することで、敵のより正確な動きを予想出来るため、高確率で射撃を成功させられることである。しかし、短所として、 背後からの敵に対して無防備になってしまうことが挙げられるので、 射撃後素早く全方向をチェックすることで、その短所を補うこととする。

1.3　射撃

8 班のチーム全体の戦略は「いっぱい避けて」、「いっぱい弾を当てる」ことであった。そして、その「いっぱい弾を当てる」戦略の実装を担当するのが射撃の部分である。よってその戦略を実現するための射撃アルゴリズムの戦略として以下の 2 つの戦略を採用した。

戦略 1：「弾の命中率を上げる」
敵は円形移動、線形移動、反重力移動など様々な手法を用いてこちらが撃つ弾をよけてくるので、なるべく多くの敵の移動に対応した射撃アルゴリズムを用いて敵の存在位置を正確に予測し、こちらの弾の命中率を上げる。それによって、たくさんの弾を敵に当てることができる。
戦略 2：「最適な弾エネルギーを設定する」
敵に弾を命中させるうえで、敵の存在位置の予測と同様に重要になってくるのが弾エネルギーの決定アルゴリズムである。どれだけ敵の位置が正確に分かったとしても、適した弾エネルギーがわからなければ敵の体力を効率的に減らすことはできない。よって最適な弾エネルギーを設定することは射撃において重要な要素である。

1.4 チーム戦略

我々のチームのチーム戦略は「一人狙い」である。生存点の獲得は robocode で勝つためには非常に大きな要因になるため、相手の班のロボットを素早く倒すことで相手の生存点を下げるためにこの戦略を採用した。また、相手のロボットの数を減らすことで、こちらが受けるダメージも減るため自分たちのチームの生存点が上がる。

2 作成したロボットの仕様

2.1 移動

本節では 1.1 節に列挙されたそれぞれの方針に対し、既に開発された移動戦術及び開発中や検討中の移動戦術の使用・実装方法について解説する。2.1.1 項では 1.1 節の方針 1，2，3 および 4 の全てを十分な程度まで満たすと考えられる戦術「反重力」を説明する。直後の 2.1.2 項では方針 5 を果たすために特化した「Ram 動き」の詳細の解説の後、2.1.3 項では方針 6 に特化した「Wavesurfing」戦術 (複雑であるため実装未完了) に関して記述する。
「壁からの反重力」と「Ram」戦術は既作成のロボットの移動戦略になっている。開発中、各戦略の有効性の確認テストとして、PBL 形式と全く同じ sample.WallsTeam(Walls3 体)、および参考文献に列記された様々なロボットから組んだチームとの対戦の点数を参考としている。

2.1.1 壁からの反重力

「反重力」とは実世界の重力 (各質点から各質点への距離の–2 乗に応じた) を斥力として Robocode 上に疑似的に実現する戦術である。この戦術は 8 班の無条件 (デフォルト) 移動戦略になっている。すなわち、上記の「壁から離れる」、「安全な円環に移る」等の戦略は攻撃していないときに限る。一方、攻撃時の移動戦術は次節の体当たり (Ram) である。攻撃時に壁に近くても、安全な場所にいなくても良いと考える。具体的に、戦場上のある定まった点や点の集合からの斥力と相似して見える。概念として本戦術では、この反重力の「斥力」を戦場の各部から各ロボットに及ぼすことにより、方針 2 及び 3 のように端を避けることになる。さらに、無条件的移動 (どのイベント関数もトリガされていない期間) の経路は単なる「線型的」、「円形的」等の固定された経路よりも随分予想しにくくなる。
図 3 の説明。前記のような効果の概略及びそれらの方針 2，3 との関係は図 3 に表示する。写真の直前の時点において、左側のロボットは Walls を狙い、次節のように Ram 攻撃をしていた (赤矢印の壁向きの部)。Walls が倒された直後 (がれきが見える)、レーダに敵がいないため、デフォルトの動作となる反重力を実施する。これにより、壁から大きな斥力 (計算式は以下のコード解説を参照) が働き、壁から離れる。離れる時に、方向かつ速度が変化し、ロボットは線型的経路でなく、放物線のような曲線に沿って振る舞う。

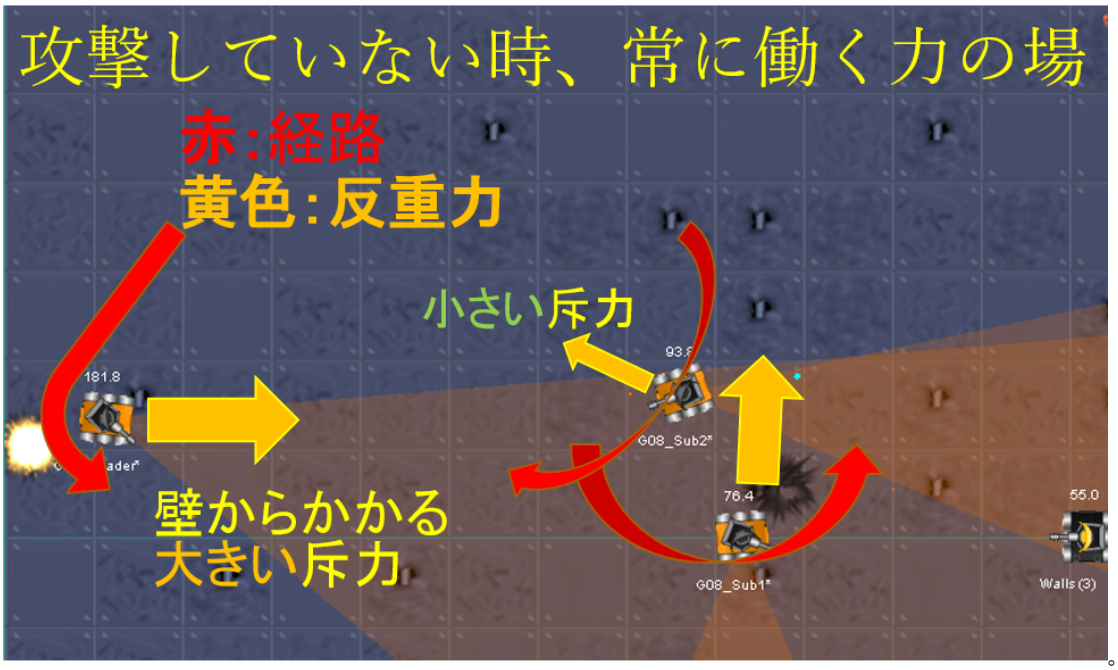


図 3　攻撃していない間の反重力の下の動き (赤矢印)

そして、反重力の概略の図の右部には残りのオレンジ色の 2 体とも攻撃する敵が見つかっていないため、左側のロボットのような (より弱い) 斥力が働き、既読の「最も安全な円環」に移るようになる。赤矢印の形から分かるよう、敵より予想しにくい放物線経路が現れる。図 3 のような反重力はランダムな方向にランダムな距離を動き出す戦術等より予想しやすいが、乱数を使った移動方針は 1.1 節の仕様の 2，3，4 及び 5 の方針のいずも満たさない (安全な円環に居られない・追いかけられないため)。

実装するときは実際に Robocode　API 上ではロボットの速度の大きさも加速度も直接編集できない。そのため、「加速度」を与えることは不可能であることが分かる。API 上、速度 (速さ) は現時点での最新の移動命令 (setAhead(int) 等) の実行のために走った距離 (pixel) のみによる。似たように、加速度は現時点の速度のみによる関数として定義されている (運動方程式)。だが、様々な異なった形に現れる「反重力」戦術は Melee Rumble 等の競争において最も流行している戦術の 1 つである。その理由として、実際、「加速度」のような効果を得るためには加速度そのものの制御は必要でなく、通常の setAhead,setBack よりも実現できる。但し、この時、移動命令の実行の前に「斥力」の X,Y 成分の大きさの計算、及びその向きの計算も必要である。壁からの斥力は以下のような antiGravity() 関数内の命令より実装できる。まず、その「斥力」の計算方法を考える。

```
public void antiGravity(){
double xForce = 0,yForce = 0;
final double antikabe = 7500;
xForce += antikabe/Math.pow(getBattleFieldWidth()-getX(),3);//左向きの成分
xForce -= antikabe/Math.pow(getX(),3);  //右向きの成分
yForce += antikabe/Math.pow(getBattleFieldHeight()-getY(),3);//下向きの成分
yForce -= antikabe/Math.pow(getY(),3);//上向きの成分
//「斥力」の成分を使った移動命令のコード
}
```

上述の xForce,yForce の斥力のそれぞれの成分の計算方法のみ参考文献の *AntiGravityBot(Guangbochen 作者)* を考慮した。式に見られるように、左方向の成分は右側の壁からの距離の 3 乗を取り、これがある定数 (antikabe) を割るように設定する。右方向の成分は左の成分を相殺するため、それに引くように考える。同様に、Y 軸の成分も計算す

る。距離の 3 乗から割っていることの理由として、実際ここで計算する斥力は直線的にかかる力であり、質点からの力とは異なる。そのため、距離の 3 乗で割った方法は実際、それぞれの直線の交点から働く距離の 2 乗 (ニュートンの法則) に従う力とほとんど同じと考えられる。これらの 2 つの成分を以下のように動き出す方向の計算に用いる。

```
public void antiGravity(){
//「斥力」の成分の計算
double angle = getHeadingRadians()+Math.atan2(yForce, xForce)-Math.PI/2;
if(angle > Math.PI/2){angle -= Math.PI; }
else if(angle<-Math.PI/2){angle += Math.PI; }
double distance=3000*(Math.sqrt(xForce*xForce+yForce*yForce));
setTurnRightRadians(angle);
setAhead(-1*distance);
}
```

angle 変数に反重力と似たような向きを計算し、保持する。getHeadingRadians() よりロボットの向き自体が得られ、これのみで setTurn より回転すると何も起こらない。そのため、これに斥力の成分からなる角度 (tan の逆数より) を足し、力学的なオフセット角度となる 90 度を引く。この計算後、setTurnRight がロボットを 90 度以上の角度にかけて回転させないようにするため (逆回転がより早いので)、場合分けを導入する。その後、斥力の大きさを 2 次元ベクトルの大きさの式から出した後、計算された angle,distance に対する移動命令を実施する。distance(斥力の大きさに比例) が大きいほど、遠い所まで移動する。これにより、実世界の重力と似たように、得られた速度および加速度も大きくなる (これらは API 上の設定であるため詳細は省く)。以上の全ての定数値 (7500,3000 等) はテストにより最良の結果を出すように最適化された。実際、全く同様に、戦場上の数十から数百の固定の直線からの斥力も実施したが、「壁からの反重力」ほど良い結果が得られなかったのである。

antiGravity() 関数は壁からかかる反重力の命令を実施するが、「敵が見えた」等の特定な関数呼び出しの場合以外、この斥力を常に掛ける必要がある。そのため、反重力の関数をロボットのデフォルト働きの run 関数内、以下のように永遠に続く while ループに呼び出す。

```
 do { antiGravity();
execute();//API 上、移動命令の正常な実行ために必要な関数
 } while (true);
```

さらに、様々なイベント関数 (onScannedRobot 等) の処理終了後にも反重力が続くように antiGravity を呼び出す。

2.1.2 体当たり (Ram) 戦術

この戦術は本チームの攻撃戦略の鍵となり、実際、既読の反重力戦術よりも随分効果的と考えられる。何故なら、Ram 攻撃には大幅なボーナス点数が得られ、我がチームの点数結果の大きな部分を占める。それに加え、実装はかなり容易である。しかし、敵に近づいて攻撃しすぎて生き残るロボットがいなくなるという欠点もあり、「両刃の剣」になる可能性もある。Walls は体当たりに対して、かなり弱いため、この場合は、近づきすぎて生き残れないといった事態は殆ど起こらないと考えられる。前節で既述したように、Ram する際、ロボットのいる場所とその安全性は考えない。つまり、敵が見つかった後、敵までに全力で (絶え間なく) 走り出し、同時に弾を放つ攻撃的な戦略を実現する。この振る舞いを図 4 に表示する。

図 4 の説明。Ram の概略図では 8 班の 2 体と Walls の 2 体を表示する。左部は Ram の始点 (追いかけ戦略)、右部は衝突時を示す。左上のロボットはレーダに既に入っていた Walls を狙い続け、同時に射撃 (青円) もする。それ対し、下のロボットはこの瞬間で Walls の位置を見つかったところであり、自分の方向を変えつつである。

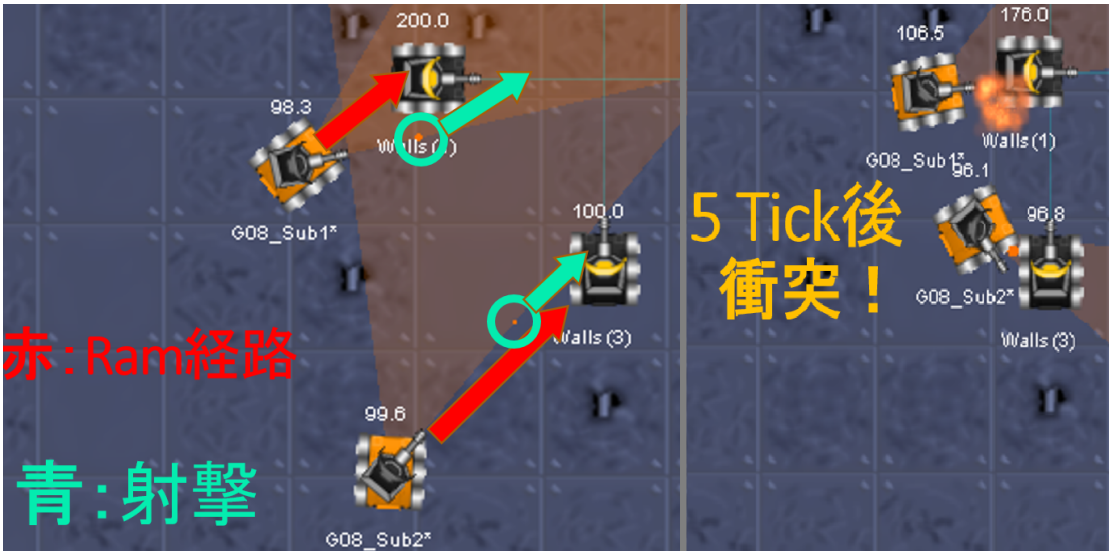


図 4　攻撃時の動き。左部は追いかけ戦略、右部では体当たりが成功。(赤矢印は経路、青円は弾)

本戦術では、敵が見つかった瞬間に、敵に向かって動き出す命令のみが必要である。実際、Robocode の移動命令の実行について、パラメータとなる移動距離が高いほど、ロボットの速度が速い。それ故、敵までに素早く近づくことができ、体当たりは非常に効果的である。何故なら、反重力の数多い繰り返し命令に対し、Ram の実現には敵までの距離を設定するという 1 つの命令のみが必要であるからだ。無論、ダメージを最大にするため、敵までに動く期間中、及びぶつかり中、2.3.4 項通りのパワーの弾も放ち続ける。しかし、本節では Ram の以下のような移動命令のみを説明する。

```
 public void onScannedRobot(ScannedRobotEvent e) {
if (isTeammate(e.getName())){//味方に対するコード}
else{//敵が見つかった
setTurnRight(e.getBearing());
setAhead(e.getDistance());
//射撃用コード（省略）
}
antiGravity();//戦車発見、対応後も反重力を続ける
}
```

上記は今まで作成した onScannedRobot の枠であり、レーダイベント呼び出しの直後、敵か否かを判定する。敵であれば、自機からかかる敵までの角度 (getBearing()) を用い、敵に向かう。敵に向かって直後、敵までの距離を参照し、敵まで動き出す。最後に、antiGravity 関数の働きが続くように、呼び出す。

2.1.3　Wavesurfing

本戦術の仕様として、レーダから見える敵のエネルギーは e.getEnergy() より参照可能であるため、エネルギーが 1 から 3 単位落下 (弾のエネルギーに相当) したとき、敵が弾を撃ったと仮定する。このようなの場合に、GuessFactor の 2.3.7 項通りのデータ収集による敵の位置の推測を行う。実際、敵の位置の推測地および敵のエネルギー落下の大きさの 2 つの情報のみにより敵の弾の位置の推測はできる。何故なら、弾の速度はそのエネルギー量のみによって決まる。弾の位置を推測した後、弾の推測地までの距離、敵の速度 (getVelocity より参照可)、敵までの距離などの全ての参照可能な情報を用い、戦場の各地域の危険度を計算する。このためには、戦場を離散的なブロックに分ける必要があり、危険度の最も小さいブロックに向かって走り出す。

Waves の生成方法を除くと、上記の危険度の計算が最も煩雑な部分である。本戦術に対し、最も安全なブロックとロ

ボットの間に非常に危険なブロックがあった場合、失敗するという問題がある。しかし、この問題は危険度の計算にそれぞれのブロックの到達性 (ブロックまでの距離等) も考慮することで解決する方針を実施する。

2.2 レーダー

Infinity-Lock は最も簡単なレーダーロックで、NanoBots で頻繁に使われてる。 ただ、In nity-Lock をする際、レーダがいわゆる “スリップ” をしてしまい、その瞬間、ロックが外れてしまうという欠点がある。 しかし、1 対 1 の対戦では他のものと比べ、はるかに優れている。（今回は 3 対 3 の対戦だが、それは 1 対 1 の対戦とほぼ同等の動きをすると統計学的観点から言えるため、1 対 1 で有効な手段を採用している）このレーダの例を以下に示す。

```
public void run() {
    // ...
    turnRaderRightRadians(Double.POSITIVE_INFINITY);

public void onScannedRobot(ScannedRobotEvent e){
     // ...
     setTurnRaderLeftRadians(getRaderTurnRemainingRadians());
```

このコードは、比較的単純なレーダーロックであり、書き込みが簡単で、通常は最小のバイトコードを生成する。これはコードサイズの制限に役立つ。

2.3 射撃

戦略 1、戦略 2 の 2 つを射撃の主な戦略として、以下にそれぞれの戦略を実現するための戦術を説明する。

2.3.1 線形&円形予測

戦略 1「弾の命中率を上げる」を実現する戦術として、線形&円形予測戦術を用いる。これは、敵の動きに応じて線形予測と円形予測を切り替えながら、その都度最適な方法で敵の存在位置を予測する戦術である。

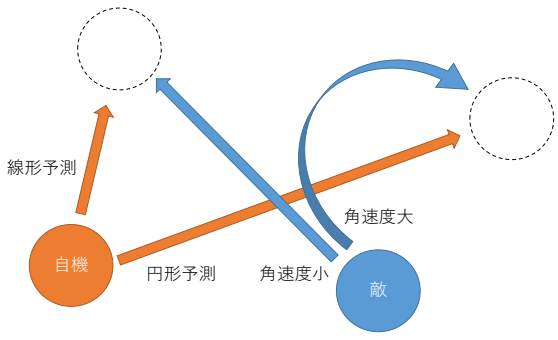


図 5 線形&円形予測の概略図

具体的には、図 5 のように敵を発見したときにその角速度が十分 0 に近ければ線形予測を用いて敵の位置を予測し、角速度がそれより大きければ円形予測を用いて敵の位置を予測する。

2.3.2　連続的弾エネルギー変化

戦略 2「最適な弾エネルギーを設定する」を実現する戦術として、連続的弾エネルギー変化戦術を用いる。これは、弾エネルギーを敵との距離に反比例して連続的に変化させる戦術である。

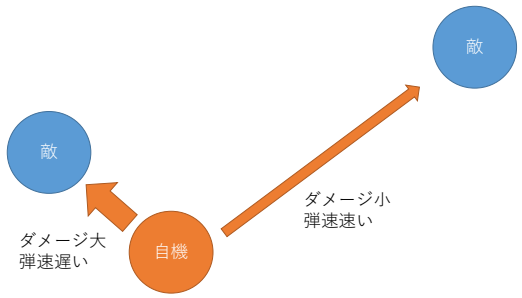


図 6　連続的弾エネルギー変化の概略図

図 6 のように敵との距離が小さい場合には弾エネルギーを大きくして大きなダメージを与え、敵との距離が大きい場合には弾エネルギーを小さくして弾の命中率を高く維持する。

2.3.3　線形&円形予測

まず線形予測について説明する。以下のように値を設定する。

(x, y) ... 現在の敵の座標
(nextX, nextY) ... 予測した敵の座標
v ... 敵の速度
t ... 撃った弾が敵に当たるまでの時間
θ ... 敵が現在向いている角度

(nextX, nextY) は以下のようにして計算する。

$$\begin{cases} nextX = x + t \times v \times cos(\theta) \\ nextY = y + t \times v \times sin(\theta) \end{cases} \tag{1}$$

この式の x 座標の予測を図示すると図 7 のようになる。y 座標に関しても同様である。

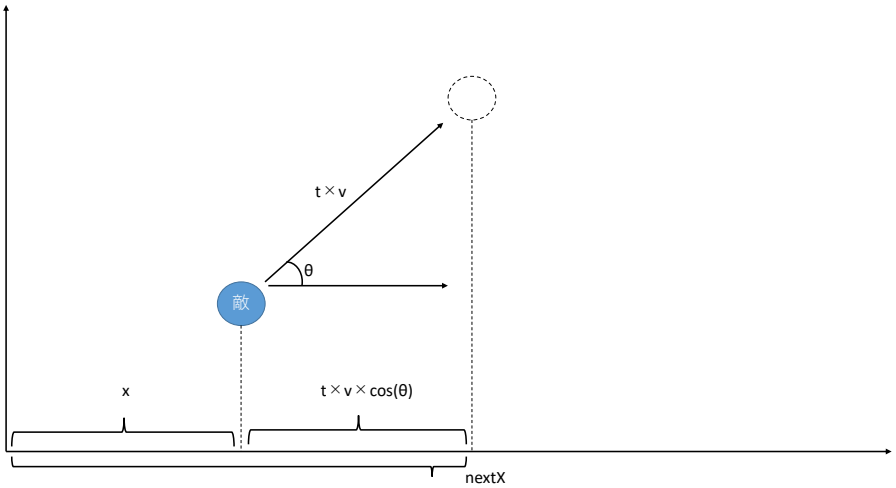


図 7　線形予測の概略図

次に円形予測について説明する。追加で以下のように値を設定する。

(rX, rY) ... 敵の円運動の中心の座標
R ... 敵の円運動の半径
θ ' ... t 時間後に敵が向いている角度

このとき、以下の式が成り立つ。

$$\begin{cases} x = rX - R \times cos(\,\theta\,) \\ y = rY - R \times sin(\,\theta\,) \end{cases} \tag{2}$$

$$\begin{cases} nextX = rX - R \times cos(\,\theta\,') \\ nextY = rY - R \times sin(\,\theta\,') \end{cases} \tag{3}$$

これらの式から rX, rY を消すと、

$$\begin{cases} nextX = x + R \times cos(\,\theta\,) - R \times cos(\,\theta\,') \\ nextY = y + R \times sin(\,\theta\,) - R \times sin(\,\theta\,') \end{cases} \tag{4}$$

の式が得られる。今回作成した円形予測はこの式を用いて (nextX, nextY) を計算している。

2.3.4　連続的弾エネルギー変化

弾エネルギー power は以下の式から求める。(x は敵との pixel 距離)

$$power = \begin{cases} 0 & (400 \leq x \text{ のとき}) \\ \frac{450}{2x} - \frac{1}{2} & (\frac{450}{7} \leq x \leq 400 \text{ のとき}) \\ 3 & (x \leq \frac{450}{7} \text{のとき}) \end{cases} \tag{5}$$

このように power を x に反比例させて連続的に求めている。ただし、$0 \leq power \leq 3$ の制限があるため、その範囲を超える場合は 0 と 3 で固定する。この式のグラフは図 8 のようになる。

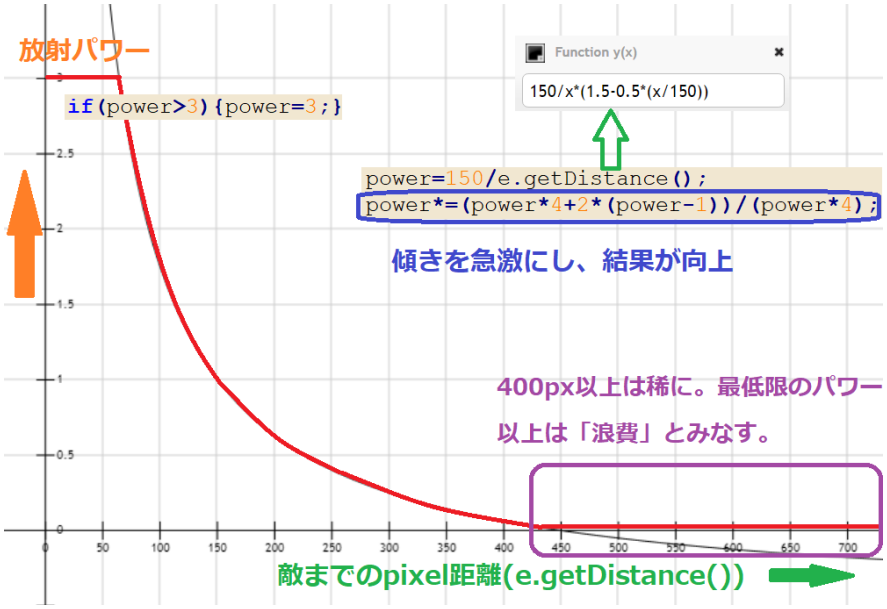


図 8　式 (1) のグラフ

敵戦車への攻撃は射撃と体当たりを並行させて行うため、敵との距離は単調減少し、いずれは十分近くなる。敵が近い距離にあればこちらの撃つ弾が遅くても十分命中率は高くなるため、100 ピクセル以内の距離の敵に対しては 2 以上の大きい弾エネルギーを設定する。また、遠くの敵に対しては遅い弾を撃つと避けられる確率が高くなるため、200〜300 ピクセル離れた敵には弾エネルギー 1 程度の速い弾を撃つ。

2.3.5　線形&円形予測の長所と短所

線形&円形予測の長所と短所を以下に記述する。

長所

- 線形予測に対して有効
- 円形予測に対して有効
- オンライン上に資料が豊富なので実装が容易

短所

- 反重力移動などの複雑な経路の移動に対して効果が薄い

予備戦の時点では線形・円形移動を採用している班が多かったため、この戦術が適していると考えられた。しかし最終戦においては多くの班が移動戦術を反重力移動に変えていたため、結果的にはあまり良い戦術ではなかった。

2.3.6　連続的弾エネルギー変化の長所と短所

連続的弾エネルギー変化の長所と短所を以下に記述する。

長所

- 遠い敵に対しても高い命中率を維持できる
- 近い敵には命中率を落とさずに高いダメージの弾を当てられる
- どんな距離の敵に対しても命中率を落とさない最大ダメージの弾を当てられる
- 実装が簡単

短所

- 特になし

2.3.7 GuessFactor Targeting について

予備戦の段階では線形&円形予測ではなく GuessFactor Targeting という敵の存在位置予測アルゴリズムを用いていた。これは、敵のこれまでの存在位置の情報を配列に格納し、その配列の中から最も敵の存在確立が高くなる位置を統計的に導出し、その位置に弾を撃つ方法である。

線形&円形予測が線形・円形移動にのみ有効であるのに対し、この方法は線形・円形・発振移動など様々な移動に対して有効であり、本来ならば線形&円形予測ではなくこの GuessFactor Targeting を用いるはずであった。しかし、その実装が完成した後のテストでその予測方法が当初想定していたほどのパフォーマンスが出なかった。原因は作成したコードの何らかのバグであると考えられたが、最終戦までの時間の都合でバグの修正を断念し、比較的実装が簡単な線形&円形予測を用いることにした。

2.4 チーム戦略

この戦略の実装の方法の概要を示す。一人狙いを実装するためには、狙うべき敵のロボットのか否か判別する必要がある。その方法としてロボットの名前を利用した。レーダーで敵のロボットを見つけたとき、そのロボットの名前を取得することができる。一人狙いをしたいロボットを見つけた後、自分自身の狙うべきロボットの名前として、その名前を保存して味方にその名前を送信する。ただし、ロボットの名前に”wall”が含まれている場合は送信を行わず、また自分自身が狙うべきロボットとして設定もしない。こうすることで敵の班のロボットのみを狙うことが出来る。受信した味方は、その名前を保存し、その名前を使ってレーダーで見つけたロボットが狙うべきものなのかどうかの判断に使う。このとき、レーダーで見つけたロボットの名前が自分が狙うべきロボットの名前と一致しており、なおかつ名前に”wall”が含まれていなければ攻撃し、それ以外ならレーダーを回し続けるといった処理に入る。ただし、敵の班のロボットがすでに全滅している場合は walls を狙うようにする。敵の班のロボットが全滅しているかどうかの説明は、以下の実装の詳細な説明で行う。

2.4.1 実装の詳細な説明

この作戦を実装するために使用したメソッドと、その用法を説明する。

onScannedRobot	敵のロボット名の送信と、狙うべきロボットか否かの判断を行う。
onRobotDeath	破壊されたロボットの名前を確認し、敵の班の残りのロボットを数える。
onMessageReceived	受信したメッセージを確認し、文字列なら敵の班のロボット名であるので、それを狙うべきロボットとして設定する。

onScannedRobot はレーダーでロボットを発見した時に呼び出され、このメソッドでは
`if(!isTeammate(e.getName()) && e.getName().indexOf("Walls") == -1 && enemyNum !=3)`,
`if(!isTeammate(e.getName()) && enemyNum == 0)`, それ以外で場合分けを行い処理を変更した。一つ目は見つけたロボットが仲間でも walls でもなく、敵の残りのロボットが 2 機以下の時に真になる。この時、取得した敵のロボットの名前を broadcastMessage で味方全員に送信を行う。ここで敵のロボットが 3 機残っているときに行わないのは最初は比較的、狙う敵のロボットの被りが発生しやすく、無理にプログラムで分担させるようにするよりも、自然に別れさせた方がスコアが高かったので、このように条件分岐を行った。二つ目は見つけたロボットが味方でもなく、敵の班のロボットが全滅していれば見つけた walls のロボットの名前を同様に送信する。それ以外では自分の倒す目標の名前を更新して終了する。

onRobotDeath は何かのロボットが破壊されたときに呼び出される。このメソッドでは破壊されたロボットの名前を取得し、そのロボットが敵の班のロボットであれば、敵の班のロボットの残りをカウントしている変数の値を 1 減らす。また、これに加えて、破壊されたロボットが自分の攻撃目標であった場合は、現在の自分の攻撃目標の名前を”none”に変更する。このように攻撃目標を”none”に変更し直す理由は onMessageReceived の説明で行う。また、

ラウンド開始時の攻撃目標も”none”である。

onMessageReceived はロボットがメッセージを受信したときに呼び出される。このメソッドでは `fi(e.getMessage() instanceof String && targetName.equals("none"))` が真のとき、自分の攻撃目標を送信されたロボット名に変更する。この条件文は受信したメソッドが string 型かつ、今の自分の攻撃目標が”none”のときに真になる。攻撃目標が”none”の時はロボットが現在どのロボットも狙っていないことを意味しており、この場合のみ送信されたロボット名に攻撃目標を変更する。もし、他に攻撃目標が存在するのに攻撃目標の変更を行ってしまうと、攻撃目標が短期間に変更されてしまい、ロボットが右往左往して時間とエネルギーを浪費してしまうので、攻撃目標の変更は狙っているロボットが存在しない場合のみに限定した。以上のように実装することで敵のロボットを一人狙いし、素早く敵のロボットを倒すことができて勝率も上げることが出来た。しかし、この実装は完全な 3 対 1 の状況を作るものではなく、多くの場合 2 対 1 の状況を作り出す。完全な 3 対 1 を作り出そうとすると、味方のロボット同士が近付きすぎてしまい衝突したり誤射をしたりしてしまい、逆に勝率が下がってしまったため一人狙いの条件を緩くし、3 対 1 の状況を極力作らないように実装した。

付録 A　議事録

A.1　第 1 回

グループ 8
10/5 G417
議長　NANDEDKAR PARTH SHIRISH
書記　槌道 慎也
出席者　高瀬, 槌道, 釣谷,NANDEDKAR PARTH SHIRISH, 水止
欠席者　なし
議題　今後の見通し
意見　・YouTube 等に動画があるので、見ることで robocode に対してのイメージを得る。
・すでに簡単なプログラムはオープンソースであるので、それを参考にしてプログラムを組む。
・作戦が載せられたサイトもあるので、それを参考にして、強い戦略を考える。
・特殊な戦車の動きが存在するので、それも調べる。
ToDo　・強いプログラムを探し、それを解読して強くするためのアイデアを出す。
・robocode wiki を見て戦略を考える
・対戦の点数の計算や戦車のエネルギーの概念を学ぶ

A.2　第 2 回

班番号：8
日時　：2018/10/12
場所　：基礎工学部 G 棟 417 号室
議長　：パルト
書記　：水止
出席者：パルト、水止、釣谷、高瀬、槌道
欠席者：なし
[議題・内容・テーマ]
・課題の確認
・これからのスケジュールの予定
・細かいルールの理解
・先輩のプログラムを参考に大まかなプログラム、戦略の方針

・ネットから 300 行程度で強めなロボのプログラムを探す、共有
・戦略のプレゼンテーションに向けて戦略を練る
[出された意見，議論の流れ]
・対戦成績よりオリジナリティーを重視
　ー＞ある程度基本のプログラムが出来てからオリジナルのアイディアを考えだし、適応させる。そのあとに強さの強化に当たる。
・WAVE とは何か
　ー＞レーダーとは別物
・体力とエネルギーは一緒
・リーダーとサブの関係の確認、プログラムを分けたほうがいいのか
　ー＞分けないほうが効率よく適応させられる、別々の動きをさせる時は条件分岐を用いる
・ネットにあるプログラムは 100 行から 3000 行以上あるが、、、
　ー＞ 300 行ほどのものを参考に基本プログラムを完成させよう
・先輩のものをベースにするか、より良さそうなものをネットから拾うか
　ー＞ネットから探す手間を考え、先輩のものをベースに改良
・コードリーディングの前に戦略プレゼンテーションの準備をしよう
・戦略の例として、反重力とは？
　ー＞近づきたくない場所に点をおき、そこからの距離によって加速度を定める。
・戦略の調査の役割分担をしよう
　ー＞移動（回避)、レーダー、ターゲットへの攻撃、
[結論]

先輩のプログラムをベースに、３隊に適応させる一つの 300 行ほどの基本的なプログラムを作る。
そこから、ネットにあるプログラムを参考にして、オリジナリティーを出すためのアイディアを練る。
具体的なコードリーディングの前に戦略の調査をしよう。
[次回までの準備]
各々戦略のアイディアを調べておく
[次回の予定]
・調べてきた戦略のアイディアの共有
・戦略プレゼンテーションの準備

A.3　第 3 回

班番号：8 班
日時　：2018/12/17
場所　：基礎工学部 G 棟 417 号室
議長　：高瀬
書記　：釣谷
出席者：高瀬 槌道 釣谷 NANDEDKAR PARTH SHIRISH 水止
欠席者：なし

[議題・内容・テーマ]
・射撃、移動、レーダー、チーム戦略の各自調べてきた内容を発表
・スケジュールを決める
・スライドの分担と発表者の決定
[出された意見，議論の流れ]

・各自調べてきた戦略を SNS（LINE）に投稿する。
・スケージュールの作成
・スライドの分担と発表者の決定
[結論]
・スライドは各自調べてきた部分をスライドにまとめると共に、発表者のためにスライドの内容をより細かく述べた文章も作る。
・スケジュール案
：10/19（スライドの準備）
：10/27（アドバイザーからの助言を受けて戦略の見直しと、プログラムを作成するために必要な情報収集）
：11/9（プログラムの分担を決定）
：11/16（各自プログラムを実装しつつ、余裕があれば他のメンバーのヘルプ）
：11/30（各自プログラムを実装しつつ、余裕があれば他のメンバーのヘルプ）
：12/7（スライドやレポートの作成）
：12/14（予備戦前のプログラムの最終確認）
：12/21（予備戦の結果を受けて戦略の見直し）
：12/26（勝つためにロボット間の情報共有の戦略を組み込む）
：1/4（勝つためにロボット間の情報共有の戦略を組み込む）
：1/11（プログラムの仕上げとスライド）
：1/25（プレゼンと最終戦）
：2/1（プレゼンと最終戦、レポートの作成）

・発表者：釣谷、高瀬
[次回までの準備]
・スライド
[次回の予定]
・発表

A.4 第 4 回

班番号：8 班
日時 ：2018/10/26
場所 ：基礎工学部 G 棟 417 号室
議長 ：水止 萌奈
書記 ：釣谷周平
出席者：高瀬 真樹 槌道 慎也 釣谷 周平
水止 萌奈 NANDEDKAR PARTH SHIRISH
欠席者：なし

[議題・内容・テーマ]
・戦略を考える
・オリジナリティ
[出された意見，議論の流れ]
・戦略
－回避第一・射撃第二・・・最も多くの弾をよけ、最も多くの弾を当てる
－ Walls よりも敵チームを優先して狙う・・・Walls はほとんどの試合で 2 位になっている。そのため、Walls を意図して狙わなくても問題ない。
・味方の流れ弾を避けるアルゴリズムは実装できなさそう

・オリジナリティはチーム戦略で出す
・メッセージの送信とかの部分でオリジナリティを出す
・オリジナリティ
ーレーダの中に Walls と敵チームがいたら敵チームを優先して狙う
ーレーダーに敵チームの戦車があれば (Walls は無視)、味方にその座標を伝える。受け取ったらその敵の方向を向く (距離が近ければ)。
[結論]
・戦術
ー反重力、Wavesurfing
ー Guess Factor Targeting
ー Infinity Lock、Walls と敵チームがいたら敵チームを選ぶ
ーレーダーに敵チームの戦車があれば (Walls は無視)、味方にその座標を伝える。受け取ったらその敵の方向を向く (距離が近ければ)。
[次回までの準備]
・基にするプログラムを読む、実行する。
・開発環境整える。
・API 調べる。
[次回の予定]
・コーディングの分担。

A.5 第 5 回

班番号：8 班
日時 ：2018/11/9
場所 ：基礎工学部 G 棟 417 号室
議長 ：NANDEDKAR PARTH SHIRISH
書記 ：高瀬 真樹
出席者：高瀬 真樹 槌道 慎也 釣谷 周平
NANDEDKAR PARTH SHIRISH
欠席者：水止 萌奈

[議題・内容・テーマ]
・戦略、戦術の確認（中間レポートに向けて）
・robocode で使われているクラス、メソッドの理解
・今後のスケジュールの決定
[出された意見，議論の流れ]
・戦略
ー回避第一・射撃第二・・・最も多くの弾をよけ、最も多くの弾を当てる
-理由：(回避に関して）melee の Top50 全てが避けることに重きを置いた戦術を用いている→避けることはとても重要だから
(射撃に関して) 球を当てることのメリットはかなり大きいため
ー Walls よりも敵チームを優先して狙う・・・Walls はほとんどの試合で 2 位になっている。そのため、Walls を意図して狙わなくても問題ない。
・中間レポートに関して
-戦略→戦術（移動、射撃、レーダー、チーム連携に関してそれぞれパルト、釣谷、水止、高瀬と槌道が担当して書く）→参考資料といった形で書く
[結論]

・戦略：「避ける第一、射撃第二！」
・戦術：今後は
移動：パルト
射撃：釣谷
レーダー：水止
チーム連携：高瀬と槌道
で分担してコードを書いていく
・今後のスケジュール
11/16：戦術の細かい仕様の決定
11/23：授業休み
11/24：各自の担当分の戦術をレポートに書いて共有
11/25：中間レポート完成、提出
[次回までの準備]
・サンプルコードの理解、改変
[次回の予定]
・robocode で使われているクラス、メソッドの理解（続き）
・戦術の細かい仕様の決定
・出来そうであればコーディング

A.6 第 6 回

班番号：
日時　：2018/11/16
場所　：基礎工学部 G 棟 417 号室
議長　：釣谷
書記　：水止
出席者：釣谷、水止、槌道、高瀬、パルト
欠席者：なし

[議題・内容・テーマ]

レポートの構成、内容の決定。締め切りの再確認。
レーダー戦略について、レポート内容の確認。
ロボの移動についての議論。
チーム戦略について、レポート内容の確認。
レポートのまとめ担当の決定。
チーム戦略についてコードの作成。
大まかな戦略の再確認。
[出された意見，議論の流れ]
レーダーについてレポートには、A4 1 ページほど。
1/3 は戦術、残りは仕様。数字を具体的に出すのが良い。
wave-surfing と guess-factor についての理解を深める。
反重力との相性はどうか。
使うか使わないか。
敵の速度や向きは参照できるか。
過去のデータとリアルタイムのデータを組み合わせて、統計的に分析？
危険度の計算。危険度の小さい場所にできるだけ移動したい。

小回り作戦はやめる。
guess-factor robot（ネットからのもの）と現在作成中のロボットの対戦は接戦。
チーム戦略
レーダーと砲塔の扱いをどうするか。
レーダーを向けるのには時間がかかる。砲塔だけあつかう。
コードの確認。
レポートを何で書くか。word?Latex?
誰がまとめるか。
チーム戦略のコード分析。
messege の送受信について調べる。
[結論]
難しい。
とりあえずレポートには全部記述する。
自分たちのロボットに何を盛り込むかは、試しながら決める。
コードに起こしつつ、かけそうかどうかも考えて選択。
walls より、敵のロボットを優先。receive で情報を共有。
wave-surfing は高度かもしれないが、試しながら、挑戦しよう。
レポートは Latex で、高瀬君が枠組みを作ってくれる。
戦術と仕様は別々で。
まとめは釣谷くん。
大まかな戦略は「いっぱい当てていっぱい逃げる」
[次回までの準備]
各自レポートの作成。

[次回の予定]
コードの開発。

A.7　第 7 回

班番号：8 班
日時　：2018/11/30
場所　：基礎工学部 G 棟 417 号室
議長　：水止
書記　：樋道
出席者：全員
欠席者：なし

[議題・内容・テーマ]
・プログラム開発の作業分担
・プログラムとプレゼンテーションのスケジュール
[出された意見，議論の流れ]
・12/17(月) ： プログラムの動作確認を済ませて提出。
・12/19(水) ： スライドの完成をさせて提出。
・プログラムの開発はレポートの分担と同じ。
移動：PARTH
射撃：釣谷
レーダー：水止

チーム戦略：槌道, 高瀬

・プレゼンの発表者：水止

・現在の問題点として味方が射線上に入っていても射撃してしまうので、それを回避するようにする。（レポートにて解決方法を考察しているので、それを参考にして作成する）

[結論]

・中間レポートに記載された wavesurfing, guessfactor 戦術, チーム戦術のソースコードの開発案が出されたので、その部分のプログラムを作成する。run と半重力は開発済み。

[次回までの準備]

・実際にプログラムを作る。困ったことがあれば、LINE か金曜日のグループ活動にて相談する。

[次回の予定]

・プログラムを実際に作って起きた問題についての議論。

・実際に動かした際の挙動の改善案。

A.8 第 8 回

班番号：8 班
日時　：2018/12/7
場所　：基礎工学部 G 棟 417 号室
議長　：高瀬
書記　：NANDEDKAR PARTH
出席者：NANDEDKAR PARTH, 釣谷, 槌道, 高瀬
欠席者：水止

[議題・内容・テーマ]

・Guessfactor の実装方法

・現在ロボットの大きな問題誤射を防ぐ案に関する議論

[出された意見，議論の流れ]

・12/17(月) ： プログラムの動作確認を済ませて提出。

・12/19(水) ： スライドの完成をさせて提出。

・Guessfactor 射撃戦術を現在の（敵の速度や Bearing を使った）線型的推測と似たような射撃計算の代わりに使う理由：特に 1v1 において、可なりのパーフォマンスの上昇が期待される

・Guessfactor 戦術は既に開発済みの反重力や Ram 戦術の働きに衝突しないという意見

・Guessfactor の実装方法：ネット上の Robowiki サイトの Guessfactor Tutorial を始点とし、授業中に開発を始めた。

・開発済みの Ram 戦術のパラメータの「最適化」、良い結果の為。

・Guessfactor の実装（第 1 版）コンパイルできました。

・情報共有：既に作成されたロボットのパッケージの最新版を各メンバーに zip ファイルとして共有した。

・自分からテストのやり方も教えた。

[結論]

・中間レポートに記載された wavesurfing, guessfactor 戦術, チーム戦術のソースコードの開発案が出されたので、その部分のプログラムを作成する。run と半重力は開発済み。

[次回までの準備]

・テストのやり方、Guessfactor 戦術の開発。

[次回の予定]

・Guessfactor を実際に作って、テストを行う。

・実際に動かした際の挙動の改善案。

A.9　第 9 回

班番号：8 班
日時　：2018/12/14
場所　：基礎工学部 G 棟 417 号室
議長　：釣谷 周平
書記　：高瀬　真樹
出席者：高瀬 真樹 槌道 慎也 釣谷 周平
NANDEDKAR PARTH SHIRISH　　水止 萌奈
欠席者：なし

[議題・内容・テーマ]
・分担に従って、各自コードを作成、テスト
・中間プレゼン、予備戦までのスケジュールの決定
[出された意見，議論の流れ]
・プログラム提出に関して
-各自で様々なパターン（定数を変更するなど）をテストして、一番結果がよいものを提出する。
プレゼンに関して
-各自でスライド 1〜2 枚を作成して最後に共有して提出する。
壁への衝突、味方への誤射、味方への衝突かなりの回数見られることがわかったので対策が必要である。
-そのためのコードも作成していく
[結論]
・今後のスケジュール
12/16(日)：各自が分担している部分のプログラムを LINE で共有
12/17(月)：共有されたプログラムをまとめたものを CLE に提出
12/18(火)：プログラム提出締め切り（13：00）、各自が分担しているプレゼンを LINE で共有
12/19(水)：共有されたプレゼンをまとめて CLE に提出
12/20(木)：プレゼン提出締め切り（13：00）
[次回までの準備]
・各自、分担されているコードとプレゼンを上記のスケジュールに従って提出。
[次回の予定]
・中間プレゼン、予備戦

A.10　第 10 回

班番号：08
日時　：2018/12/21
場所　：基礎工学部 G 棟 417 号室
議長　：パルト
書記　：槌道
出席者：全員
欠席者：なし

[議題・内容・テーマ]
：今回の反省点
：各班の発表を見て思ったこと。

：最終レポート、提出に向けての予定

[出された意見，議論の流れ]
：提出の際に意思疎通が取れていなかったので、しっかりと誰が出すのかを決めておく。
：今後の予定の決定。

[結論]
：各自のプログラム提出（1/20）
：各自のプレゼンの提出（1/22）
：スライドを一つにまとめて提出（1/23：担当者兼発表者、槌道）
：最終レポートの各自の提出（2/6）
[次回までの準備]
：弱っている敵を一人狙いにする
[次回の予定]
：各自プログラムを作って、もし困難なことがあれば相談する。

A.11　第 11 回

班番号：08
日時　：2018/12/26
場所　：基礎工学部 G 棟 417 号室
議長　：NANDEDKAR PARTH
書記　：釣谷
出席者：全員
欠席者：なし

[議題・内容・テーマ]
：今後の作業内容
：現状のプログラムの完成度、問題点の確認
：各メンバーの担当を更新
[出された意見，議論の流れ]
：[槌道の担当]
・味方への誤射をしないようにする (onHitbyBullet 関数内の送受信仕組みを利用）
・味方と衝突しないようにする (onHitRobot 関数内の送受信仕組みを利用）
：[高瀬の担当]
・最後に Walls だけ残ったときに何もしなくなるバグをなおす
：[釣谷の担当]
・Guess Factor Targeting 完成させる
・Guessfactor の実装方法：ネット上の Robowiki サイトの Guessfactor Tutorial を始点とし、授業中に開発を始めた。
・開発済みの Ram 戦術のパラメータの「最適化」、良い結果の為。
：[パルトの担当]
・弱ってる敵を優先して狙うのを実装する
・優先狙いのために、オブジェクトの Arraylist を用いたデータ構造を利用し、敵および味方の (energy, 座標 x,y）の情報を格納する仕組みをコードし始めた。
[結論]
：Guess Factor Targeting はうまくいかないかもしれない

：robocode のバージョンによってパーフォマンスが変わってくるかもしれない

[次回までの準備]

：各自担当の作業をする。Arraylist を用いた敵および味方の情報格納のコーディング。

：Guessfactor より簡単、パーフォマンスの良い射撃戦略の発案（円形予測をもとにする）。

[次回の予定]

：射撃アルゴリズムについて再度見直す

：Guessfactor 戦術に関する大きな問題（実行結果はよくない）について判断、より簡単な射撃戦略の考案

A.12　第 12 回

班番号：08
日時　：2018/1/4
場所　：基礎工学部 G 棟 417 号室
議長　：樋道
書記　：NANDEDKAR PARTH
出席者：全員
欠席者：なし

[議題・内容・テーマ]

：Guessfactor 戦術の欠点、それに関する判断

：円形予測射撃戦略の仕組みの検討に関する議論

：高瀬の担当を変更：より簡単な円形予測射撃戦略を担当する

[出された意見，議論の流れ]

：[樋道の担当]

・最新の robocode バージョンと古いバージョンの間のパーフォマンスの違いについてのバグ対策

・コンパイル通っても、.class ファイルが作成されないエラーについての解決方法の検討

：[高瀬の担当]

・最新の robocode バージョンと古いバージョンの間のパーフォマンスの違いについてのバグ対策

・コンパイル通っても、PBL 用パソコンに.class ファイルが作成されないエラーについての解決方法の検討

：[釣谷の担当]

・Guess Factor Targeting をあきらめ、円形予測の数学的な式の発案。

：[パルトの担当]

・弱ってる敵を優先して狙うのを実装するための Arraylist を用いた敵および味方の情報格納のコーディングを完成した

・onHitbyBullet 関数、onHitRobot 関数内の誤射避けの送受信仕組みを完成した

[結論]

：Guess Factor Targeting を諦め、円形予測についてネット上に学ぶ

：味方への誤射味方への衝突のための仕組みのテストより分かるように、微少の効果のみがあったため、そのっコーディングを個々の時点で終了する。

[次回までの準備]

：PBL 用パソコンに.class ファイルが作成されないエラーの解決。

[次回の予定]

：円形予測を完成。

：Arraylist を用いた敵および味方の情報の利用に関するコードの完成。

A.13　第 13 回

班番号：08
日時　：2018/1/11
場所　：基礎工学部 G 棟 417 号室
議長　：釣谷
書記　：NANDEDKAR PARTH
出席者：全員
欠席者：なし

[議題・内容・テーマ]
：円形予測射撃戦略の仕組みの実装方法に関する議論
：円形予測のより複雑な数学的な式の考案
：PBL 用パソコン上のテスト確認、結果
：レポート担当分、スライド担当分を決した
[出された意見，議論の流れ]
：[樋道の担当]
・古い robocode バージョンから生じたパーフォマンスの違いについてのバグ対策済み
・.class ファイルが作成されないエラーについての解決
・発表担当なのでそれに関する話
：[高瀬の担当]
・スライド、レポートでの「時間がなくてできなかったこと」を考えた。
・射撃担当に手伝った。
：[釣谷の担当]
・円形予測の数学的な式の考案。
：[パルトの担当]
・弱ってる敵を優先して狙うのを実装するための Arraylist を用いた敵および味方の情報格納のコーディングを完成した。
・移動の最終プレゼンの作成、完成。反重力、Ram を中心にする。
・担当なのでそれに関する話。
：[水止の担当]
・作業計画の進歩状況、計画の良い、悪い点を担当。
[結論]
：Guess Factor Targeting を諦め、残りの時間で円形予測を実装。
：各自のレポート、スライド担当分の決定。
：レポート内の「作業計画」について書くことが決まった。
：22 日まで各自のスライド提出
：20 日まで各自のプログラム提出
[次回までの準備]
：スライドを締め切りの随分前に終わらせる。
[次回の予定]
：スライド、プログラムを完成、提出。
：Arraylist を用いた敵および味方の情報の利用に関するコードの完成。

付録 B　最終プレゼン資料

8班最終プレゼン

高瀬 真樹　　釣谷 周平
槌道 慎也　　水止 萌奈
NANDEDKAR PARTH SHIRISH

- 移動戦略
- 射撃戦略
- チーム戦略
- 時間があれば実現したかったこと

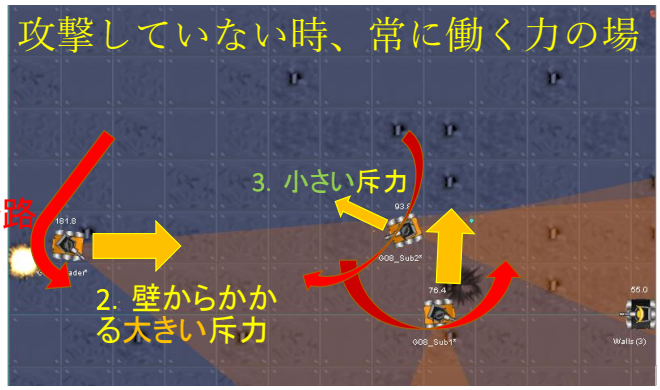
移動戦略:「いっぱい避ける！」

1. 敵の射撃戦略によって自機の位置が簡単に予測されないようにする。

経路の複雑化より、線型的(Wallsを倒すための流儀の戦略)や円形的推測、Pattern Matching射撃を無効にすると良い。

2. Wallsが壁に近い敵を狙い、壁に当たるとダメージを受けるため壁を避ける。

最適な「反重力」移動戦術



体当たり(Ram)戦術

3. 800*800で9体は窮屈であり、見えた敵を追い掛けやすい。

体当たり(Ram)戦術

5 Tick後...

衝突！

移動戦略の短所

「壁からの反重力」の短所：
戦場の真ん中に影響がない。

「Ram」の短所：
ハイリスク戦術。生存の点数は低い。

射撃戦略:「いっぱい当てる」

戦略1
弾の命中率を上げる。

戦略2
最適な弾エネルギーを設定する。

射撃戦略:「いっぱい当てる」

戦術1 −円形予測 & 線形予測−

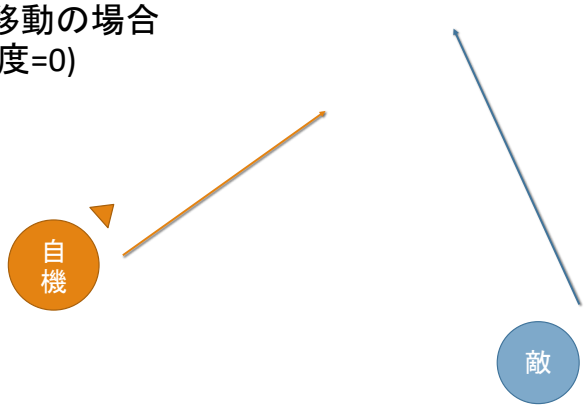
概要:敵の角速度に応じて円形予測と線形予測を切り替える。

長所:線形・円形移動に対して有効。

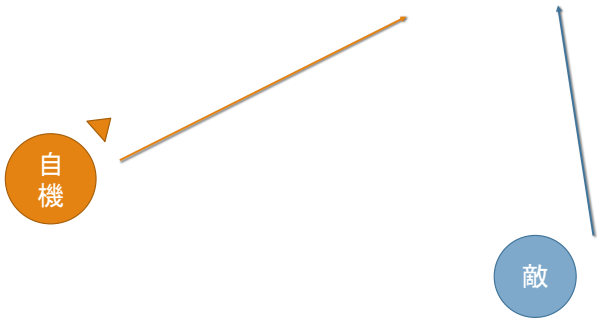
短所:不規則に動く敵に対して効果が低い。

実装:資料が豊富なため比較的簡単。

線形移動の場合
(角速度=0)



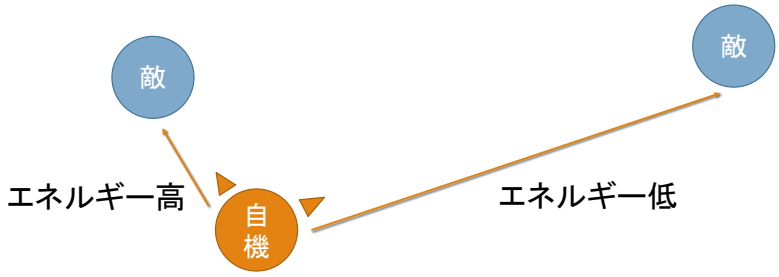
円形移動の場合
(角速度≠0)



射撃戦略:「いっぱい当てる」

戦術2 -連続的弾エネルギー変化-

概要:弾エネルギーが敵との距離に反比例する。
長所:どんな状況でも命中率を高く保てる。
短所:特になし。
実装:かなり簡単(敵との距離に反比例する式に代入するだけ)



射撃戦略:「いっぱい当てる」

予備戦からの変更点

- 当初はGuessfactor Targetingという手法を開発していたが・・・
  - 想定したより精度が低かった。
  - 多くの班が線形または円形移動を採用。

→ 射撃戦略を円形・線形予測に！！

チーム戦略:「集中攻撃」

・できるだけ同じ敵を狙う
　:レーダーで敵の班のロボットを見つけたとき、
　他のロボットに見つけたロボットの名前を送信して、
　このロボット以外は無視するようにする。

長所:同じ敵を狙うことで、より早く相手を倒すことができ、結果として相手の生存点を大きく下げられる
短所:同じ敵を狙うため味方同士近づきすぎてしまい、衝突や誤射によるダメージを受けてしまう。

チーム戦略:「集中攻撃」

開発の難しさ
・onScannedRobotメソッド内で味方への送信と
　狙うべき敵かどうかの判断
・onMessageReceivedメソッドで狙うべき敵の変更
以上の2点だけなので容易である。

予備戦からの変更点
・この戦略に関する全て

時間があれば実現したかったこと

・JavaのArrayListを用いた敵の情報収集
→その情報を利用してロボットの射撃に応用

・Guess Factor Targetingの実装
→うまく動作させることが出来なかった

・味方同士での衝突の回避
→敵に向けて体当たりを行う際に、味方同士での衝突が起こる

参考文献

[1] http://robowiki.net/wiki/Anti-Gravity_Tutorial（2018 年 10 月 30 日閲覧）および、本ページ内のリンクのサイト（省略）。

[2] http://robowiki.net/wiki/Wave_Surfing（2018 年 11 月 18 日閲覧）および、本ページ内のリンクのサイト（省略）。

[3] http://robowiki.net/wiki/GuessFactor_Targeting_Tutorial（2018 年 11 月 15 日閲覧）および、本ページ内のリンクのサイト（省略）。

[4] ソースコードの始点ロボット（OpenSource）:https://github.com/robocode/robocode/tree/master/robocode.samples/src/main/java/sampleteam（2018 年 10 月 10 日閲覧）

[5] 反重力の参照ロボット(斥力の成分の計算方法のみを使った):https://github.com/guangbochen/RoboCode/blob/master/src/com/hoanchen/robocode/AntiGravityBot.java（2018 年 10 月 13 日閲覧）

[6] Wavesurfing の参照ロボットかつテストの対称 1（3 体のチームを組んだ OpenSource）:http://robowiki.net/wiki/SuperMercutio（2018 年 10 月 30 日閲覧）

[7] GuessFactor の参照ロボットかつテストの対称 2（3 体のチームを組んだ OpenSource）:http://robowiki.net/wiki/GFTargetingBot（2018 年 11 月 15 日閲覧）

[8] テストの対称ロボット 3(OpenSource):http://robocode-archive.strangeautomata.com/robots/wiki.twin.KomariousTeam_1.0.jar（2018 年 11 月 15 日閲覧）

[9]「Robocode の高等技術 -円形予測-」＜ https://www.catch.jp/program/robocode/Appendix1.pdf ＞

[10] Alisdair Owens (2002)「Robocode 達人たちが明かす秘訣　円形方式の標的合わせ」＜ https://www.ibm.com/developerworks/jp/java/library/j-circular/index.html ＞