Robocode Rumble: 優勝者から得たヒント

Rumbleの優勝者が自らの勝利ロボットの戦略を明かす

Dana Barrow
Freelance technical writer

2002年 8月 01日

Robocode Rumbleは終わり、勝利者が宣言されました。それは誰だったのでしょう。そしてその成功の秘密は何だったのでしょうか。Dana Barrowが、勝利ロボットを作った熱狂的な科学者たちに突っ込んだ話を聞きます。さらにMat Nelsonにも話を聞き、彼はRobocode 2.0で計画していることを明かします。

Robocode Rumbleは世界中のプログラマーによってその幕が開かれました。プログラマーたちは自らのコーディング・スキルを駆使してあらん限りのすごいJava「ロボット」を作り上げ、そしてその自らのロボットを仮想競技場に解き放って死闘を演じました。TheArtofWar、BienatorII、SandboxLump、BulletMagnet、それにCakeのような名前のロボットは、いつものJavaオブジェクトよりも少々凄まじく、そして大変面白いものでした。わずかのロボットが勝ち残り、そして騒ぎの幕は閉じられました。総合勝利を収めたのはオランダのプログラマーEnno Petersでした。

普通のロボットとは一線を画する優勝ロボットを生み出した背景にある戦略決定と日々のコーディングの実践に興味があったので、Petersをはじめ他の4人の勝利した開発者にインタビューしました。そこで分かったのは、それぞれのロボットがそれぞれの側面において勝利者であったように、開発者それぞれが独自の明確なやり方を持っているということでした。

ここでPetersと勝利者仲間のSteve Coope、Daniel Johnsun、Arun Kishore、そしてManfred Schusterが、彼らのロボットの成功の鍵となった設計の要因について語ります。Rumbleを締めくくるにあたってMat Nelsonからも話を聞き出しました。彼はRobocode 2.0の計画についていくつか明らかにしています。

Yngwie: 予言者

Robocode Rumbleの勝利者のEnno Petersは、オランダ北部のフローニンゲンで人工知能を学ぶ24歳の学生です。昨年4月、彼と彼の友人のMartijn Muurmanはフローニンゲン大学で開かれる競技会についてのEメールを受け取り、そこでRobocodeのことを知りました。そのゲームを手に入れた2人はその虜になりました。Yngwieが大学のトーナメントで何とか優勝した後、Ennoはそのロボットに磨きをかけて、自分の根性をRobocode Rumbleで試してみることにしました。

Robocodeロボットを設計するときは多くの要因が出てくるが、鍵となるものはほんの幾つかだと、Ennoは語ります。「何と言っても非常に重要なのは、レーダーの動きが優れていることです。ロボットが新しい情報として得ることができるのは、レーダーの弧の範囲にいる敵のロボットの位置の情報だけなんですから。Yngwieはどのロボットが接近しているか、どのロボットが危険かということを判断して、可能な限り頻繁にそれらのロボットを効果的にスキャンしようとします」またEnnoのロボットは「敵の動きとアクションの履歴をできるだけ正確にするために、欠落情報を補完します」。予測不能であることも鍵です。「Yngwieは敵の周囲と [弾丸] が発射された位置の周囲に多少ランダムなスピードと方向で機銃掃射を行いますが、残っている敵ロボットが多いときは競技場の中の平穏なスポットに行こうともします」

優勝者

上級者レベル (Advanced level) 優勝: Yngwie、Enno Peters作 (オランダ)
準優勝: Vapour、Steve Coope作 (英国)

中級者レベル (Intermediate level) 優勝: Fermat、Arun Kishore作 (シンガポール)
準優勝: BigBear、Daniel Johnsun作 (オーストラリア)

初心者レベル (Beginner level) 優勝: Ares、Manfred Schuster作 (ドイツ)

Ennoは、自分の戦略の要素のいくつかは独特なものではなかったと言います。「私のロボットと他のロボットには類似点がたくさんあることが分かったんです」多くのロボットは「できるだけ効果的にスキャンして敵の記録を維持しておこうとします。また、1対1の対戦用とグループの対戦用に別のモードを持とうとします」。その上「大半のロボットは困難な状況からできるだけ遠くに離れたところにいようとし、対処しやすいロボットをできるだけ早くやっつけようとしたがり、そして敵のエネルギーと距離に応じて弾丸のパワーを調整し、ランク付けされているいくつかの予言者 (predictor)を使いたがります」。予言者のランク付けは、その予言者でうまくいったときはその予言者にポイントを与えて、失敗したときはポイントを減らすという方法で行われます。時間が経てば、特定の予言者が他の予言者より高スコアになります。こうして、最も破壊力のある予言者を選んで、敵に対して行使することができるのです。

それぞれのロボットの違いは細部にあるとEnnoは語ります。たとえば「私は`Predictor` クラスを利用しましたが、これは弾丸のパワーの適切な量を選択して`Gunner` クラスに発射の標的を指示するものでした」。敵のエネルギーが少なくなっているときは、この予言者はその敵のとどめを刺すのにちょうどよい発射パワーの量をYngwieに選択させて、ロボットの貴重なエネルギー資源を節約するのです。Ennoの成功した予言者は他にもありますが、その1つは別のロボットが砲撃を受けた後に移動した量と方向の記録をYngwieに保持させるものです。「この予言者が選択されると、似た状況 (たとえば敵との距離や角度) が履歴から検索され、発射はそのときの敵の位置の予期偏差にもとづいて行われます。この予言者は弾丸から身をかわすほとんどの敵にとても有効でした」

Yngwieのソース・コードをダウンロードできます。参考文献をご覧ください。

Vapour: 栄光の彼方に

弾丸から身をかわすロボットの1つにSteve CoopeのVapourがいました。生粋の英国人Steveが Robocodeのことを知ったのは、Rumbleに出場していたある友人からでした。成功したVapourの設計の陰には、Steveが最初に設計したSilverというロボットでの試行錯誤がありました。「Silverを組み立てるのにおよそ1ヶ月かかりましたが、あまりうまくいったわけではありませんでした。

だけどそのおかげで自分のアーキテクチャーをいろいろ試すことができました」満足できる基本プラットフォームになると、Steveはロボットの動きの実験を始めました。その結果「パフォーマンスと観測される特徴においての飛躍的向上」を得たのです。新しいロボットはオリジナルを改良したものでしたので、Steveはそれを独り立ちさせようと決め、そしてVapourが誕生したのでした。

名前の由来は？

「Yngwieという名前は、ロック・スターのYngwie Malmsteenにちなんで付けたものです。彼はギターを世界で最も速く弾けるギタリストの1人ですが、私が奴を好きなのはそのことよりも、大げさな彼のスタイルとルックスなんです。それをロボットのYngwieは派手な勝利のダンスで真似しようとしています」-- Enno Peters (オランダ)

「Vapourは、やみくもに動き回って完璧にへまをやるよりも、競技場の中の平穏なスポットに移動したがります。そのときの一連の動きがちょっと不規則なんです。その様子が煙の粒子の拡散やブラウン運動のように思えました。それで "Vapour" という名前はそのイメージにまさにぴったりだと思ったんです。」-- Steve Coope (英国)

「"BigBear" という名前は単に頭の中に浮かんできただけなんですが・・・。ほかのロボットを打ち砕くのが好きな大きくてかわいいテディベアをイメージしていただけだと思います」--Daniel Johnsun (オーストラリア)

「私の最初のロボットの名前はFibonacciでした。それからEuclid、Eulerと続いて、そのあとにFermatです。私のロボットにはみんな数学者にちなんだ名前を付けてきています。Fermatはフランスの数学者のPierre de Fermatの名をとった名前です」-- Arun Kishore (シンガポール)

「Vendettaという名前のロボットがもう他にいたので、別の名前を選ぶことにしました。私のガールフレンドがギリシャ神話の戦いの神の名前を選んでくれたんです。それでAresが生まれたわけです」-- Manfred Schuster (ドイツ)

Vapourの成功の大部分は、そのロボットの柔軟性とそれによって可能になった実験のおかげだとSteveは考えています。「Vapourは簡単に拡張できて、モードを切り換えるのも簡単です。だから手っ取り早く何かを変えて別のバージョンと比較して評価するのもとてもやりやすいし、そうすることのリスクがないんです」SteveもEnnoと同じように、細部に念入りな注意を払いました。「私はちょっと完全主義者のようなところがあるので、完全にデバッグをしないと気がすみません。飽き飽きするときもありますけどね。だから私の開発の全体の進み具合は遅かったんですが、すべてのものが設計の意図に非常に近い状態で機能していることや、普通でない状況で厄介なことが突然いくつも起こっても大丈夫だろうということは、かなりの確信がありました」

戦略に関しては、単純で簡潔なものを目指したとSteveは言います。「実際の実装ではいつもそういうわけにはいかないにしても、アルゴリズムは概念的には単純にしておきたい、と私は思いました」彼は独創的であることも目指しました。「Vapourを純粋に私の創作物にしたかったんです」

Vapourの成功のもう1つの秘密は、動きと標的設定への配慮でした。「複数の敵に対する移動は、中継点を線状につないでいく方法です。中継点に達するたびに、可能性がある新しい中継点が複数生成されます。競技場の壁に近すぎるものは直ちに捨てられ、残りの候補の中のどれが他のすべてのロボットとの距離の合計が最大になるかが評価されます」彼独自の移動の方法を創り出すとき、Steveは一般的な「反重力」の方法は避けました。これを使うと無用な問題が発生してそれを解決するためにさらにコーディングが必要になると感じたからです。

標的設定ではロボットの短期学習の能力を使っています。Vapourは160動作を超える標的のすべての動きのリストを利用して、標的の動きに観測可能な周期性があるかどうかの判断を試みます。Vapourが、標的の動きの履歴を、同じ履歴上で時間をずらしたものと相関させることができると、最もよくマッチングしたときの時間のずれが標的の動きの周期とみなされます (そのマッチングが十分に有効であることが必要)。そうでない場合は、標的の動きに周期性はないとみなされます。「どちらにしても、」Steveは言います。「Vapourはそれを未来に対して適用し、標的の動作ごとに標的の過去の動きを再生して標的のいる場所を予測します」。標的設定のベストの解が、迎撃の可能性のある解のリストから選択されます。実際の解がないときでも、ロボットは標的の次の動作の解の可能性を高めようとします。

「標的の動きに周期性があるときとないときの主な違いを言えば、周期的な標的の場合はVapourは標的の履歴に戻ってその動きを順方向に再生しますが、観測可能な周期性がないときは標的の動きを時間の逆方向に再生するというものすごい単純化を行います。でもこれが意外にも効果があるんです。Vapourの精度は、非常に周期的な標的に対しては優秀、それ以外のほとんどの標的に対しては良好、時間とともに動きを巧妙に変えるものに対しては平均的、といったところです」

Vapourのソース・コードをダウンロードできます。参考文献をご覧ください。

BigBear: またの名を「ザ・体当たり」

SteveがVapourに作り込んだ柔軟性の要素は、Daniel JohnsunのBigBearにも見られます。Danielは、競技会の他のロボットを全部打ち負かせるような完全な戦略は1つもないということを自覚していました。「だから私がやろうとしたことは、」彼は言います。「状況に応じて戦略を切り換えることができるフレームワークを構築することでした」このことが新しい戦略を容易に追加できることにもなったのです。

彼のロボットの柔軟性の1例が、その照準戦略で実証されています。「`Enemy` オブジェクトはそれぞれ`InterceptManager` クラスのインスタンスを持ちます。このインスタンスにはさまざまな照準アルゴリズムが保存されているんです」各ロボットのスキャン後、`InterceptManager` は次のことを行います。

- それぞれの照準アルゴリズムを利用して「仮想」弾丸を発射します (つまり弾丸は実際には発射されませんが、その位置はまるで発射されたかのように監視されます)。
- 仮想弾丸のそれぞれが実際には標的からどれくらい離れていたかを判断します。

Danielは語ります。「`InterceptManager` は照準アルゴリズムのそれぞれの精度についての統計データを絶えず更新しています。だから実際の弾丸を標的に向けて発射する前に、`InterceptManager` は最も精度のよいアルゴリズムを選んで利用するんです。こうしてBigBearは照準をいくつものロボットに順応させることができます。バトルの間に相手の動きのパターンが変わっても、です」

BigBearの戦略の他の特徴は計画されたものではなく、予想外のものでした。「BigBearが弾丸から身をかわす動きの戦略は、実のところ、まったく偶然の産物でした」Danielが明かします。「私の最初の戦略は、弾丸は円形の標的設定で私に向かって発射されたものとして、その弾丸から身をかわして避けることでした。何度か改訂をして、弾丸回避戦略はついに期待通りに作動するよ

うになりましたが、一方、標的設定を円形にしない優れたロボットに対してはうまくいかないということが分かりました。以前の完成していないコードの方がましなように思えたので、その以前のコードに戻って動きの全体に手当たり次第に補足的な要素をいくつか付け加えたんです」

Danielのロボットは、他のロボットが明らかに有利な状況のときに勇猛果敢ににそれらに体当たりするように設計されました。BigBearにRam Bonus (体当たりボーナス) ポイントを獲得させようという狙いです。Danielは振り返ります。「 それが最終スコアに大いに寄与したかどうかは分かりません。でもBigBearのそのアクションを見るのは面白かったですよ」

BigBearのソース・コードをダウンロードできます。参考文献をご覧ください。

Fermat: 身のかわしが上手

Arun KishoreはシンガポールのNanyang工科大学コンピューター工学部の学生です。彼もまた、動きと標的設定に焦点を合わせて彼のロボットFermatを設計しました。彼が言うには、Fermatの成功につながった戦略は、全般的に見れば競技会に参加した他の多くのロボットの戦略と同じようなものでした。直線方式の標的合わせ、円形方式の標的合わせ、反重力移動など、それらの戦略は、"Secrets of the Robocode masters" シリーズ (参考文献を参照) に記載されています。しかしもっと具体的に言うと、彼のロボットの勝利の鍵となった要因は、Fermatのために設計した動作アルゴリズムだったとArunは言います。興味がおありでしょう ?Arunの`DodgeBullet` クラスのコードとコメントを見てみましょう。

リスト 1. DodgeBullet

```
class DodgeBullet
{
Point2D.Double source; // origin point of the bullet
Point2D.Double target; //target position of the
bullet
double velocity;       //velocity
double startTime;      //start time
int bot;               //id of the bot which fired
this bullet
DodgeBullet(double sourceX,double
sourceY,Point2D.Double target,double power,double
startTime,int bot)
{
source = new Point2D.Double(sourceX,sourceY);
this.target = new
Point2D.Double(target.x,target.y);
this.startTime = startTime;
this.velocity = 20-3*power; //velocity can be
determined from the energy drop value
this.bot = bot;
}
Point2D.Double getCurrentPosition(double time)
{
double distance = velocity*(time-startTime);
//uses linear interpolation to calculate the exact
position of the bullet
// uses the formula currPoint =
distanceFromStart*(targetPoint-sourcePoint)/(totalDistanceTravelled)
return new Point2D.Double(
source.x +
(distance*(target.x-source.x)) /
  getDistance(target.x,target.y,source.x,source.y),
source.y +
```

```
(distance*(target.y-source.y)) /
  getDistance(target.x,target.y,source.x,source.y)
);
}
}
```

Fermatは`DodgeBullet`クラスとオブジェクトのリストを利用して、どの時点でも、向かってくる弾丸についての情報を保存することができます。Fermatは、敵ロボットの砲撃は、エネルギー低下を測定し、それが0.1 ~ 3.0以内かどうかを調べて行われたものと想定しています。そして発射された弾丸の衝突地点を計算し、検出したすべての弾丸をそのときのリストに追加し、次にそのリストにある各弾丸の位置を計算します。

Fermatは弾丸の弾道を識別し、その弾道に対して直角になる姿勢にして弾丸から身をかわします(壁で動けなくなることがないようにします)。このロボットは5つの砲撃戦略のすべてについて衝突地点を計算するので、その戦略にかかわらず砲撃を回避することができます。Fermatの設計の詳細は、参考文献を参照してください。

Fermatのソース・コードをダウンロードできます。参考文献をご覧ください。

Ares: 初心者の勝利

Java言語のコードを書き始めたばかりで、ここで登場したいくつかの戦略はあまりにも複雑と思われている読者に申し上げますが、比較的基本的なコードでも強力なロボットを創り出すことができるのです。Manfred Schusterは初心者レベルで頂点に立ったロボットを設計しましたが、彼はRobocodeの競技に参加する前はJavaコードを1行も書いたことがなかったのです。彼はRamFireというサンプルのロボットで実験することから始めました。それからRobocode Repositoryから他のロボットたちをダウンロードして、それらの機能を調べたのです。彼がRobocode RumbleのためのAresを創造するまでに、彼の研究と実験から、キラー・ロボットには何が必要かということが彼にはよく分かっていました。

「1発も発射しなくても1対1のバトルに勝てるんです」とManfredは言います。その要領は、敵の砲撃をことごとく巧みに回避して生存者ボーナスを獲得し、スコアを上げることです。「そのために私は、動きのアルゴリズムを4つ実装しました」あるアルゴリズムが敵に対してうまくいかなかったときは、彼のロボットは4つのオプションの中の別のアルゴリズムを試します。これらのアルゴリズムがAresに対して指示するのは、90度回転することと、ランダムな速度と距離で前方と後方に絶えず動くことです。こうしたAresの動きはパターン・マッチングを利用する敵にとっては難しい標的になります。

しかしこれでは、できることは回避だけになります。Manfredは、10ラウンドのすべてを生き残るだけではロボットは勝利者になれないと言います。勝利のためにはもっと積極的な戦術が必要です。そのために彼は、Aresが4つの照準アルゴリズムを試して、命中した数が最も多かったアルゴリズムを選ぶように設計しました。また彼は、Aresは発砲する前に敵の方に進んで行って命中の可能性を高めるという大胆不敵な戦略もとりました。この戦略では、敵を砲撃することによってAresはポイントを稼げるかもしれませんが、その反面、Aresは簡単な標的になってしまう可能性もあります。Manfredは次のように言いました。この戦術のために他のロボットに負けるときもあったが、それでも「上位8ロボットに入れる」だけのポイントは獲得するだろうと自分は思い描いていたと。しかし彼は、この種の戦略でチャンピオンシップを闘うのは難しいだろうとも警告しています。

Aresのソース・コードをダウンロードできます。参考文献をご覧ください。

おわりに

さて、いよいよ最後ですが、優勝ロボットを作っていく中で戦略が果たす役割は何でしょうか?ここでインタビューした勝利者たちのほとんどが、実証されている戦略と革新を組み合わせることで自分のロボットを最先端のものにすることができたと示唆しています。「最終的には運が大きな役割を果たしました」とEnno PetersはYngwieの勝利を謙虚に総括しています。

勝利者の創作の背景にあるもう1つの要素は、柔軟性です。Arun Kishoreは、彼のロボットのコーディングを、彼のそれまでのプログラミング経験のどれとも違うと表現しています。「私の以前のプログラミング作業のほとんどは、あくまでも仕様書通りにやることが必要で、プログラマーが自由にできることはそれほど多くありませんでした。しかしRobocodeでは、プログラマーはアイデアを集めてそれをロボットに合わせて設計し、そしてそれを実装する必要があります。これは普通、大きな満足感につながります。そしてもっと大切なことは、とても面白くなるということです」

挑戦者が、気合を入れて猛烈に取組む意欲に加えて、面白いというその気持ちを持ち続けることで、Robocodeの生みの親であるMat Nelsonは益々元気づけられています。彼はRobocode 2.0を準備しているのです。「私はコミュニティーにフレームワークを渡したのですが、彼らは私の最初の期待の域を超えてはるか遠へ行ってしまいました!」彼は興奮して語ります。「彼らは私が思い描いていたものよりはるかに複雑なロボットを作ってしまったんです」

Matは現在、ゲーム用の他にいくつかの新しい特性のためのAPIの再設計に取り組んでいます。彼は、力学のベクトル、ロボットの装甲板の強度に変化をつけること、そして障害物をRobocode 2.0の競技場に導入したいと考えています。彼は、このゲームを最終的にはオープン・ソース化できればとも思っています。彼は言います。「こいつにとって、きっとそれがちょうどいい環境だと思うんです」

著者について

Dana Barrow

Dana Triplett Barrow氏は、バーモント州に拠点を置くフリーランスのライターです。