目次

第1章　序論　1
- 1.1　研究背景　1
- 1.2　研究目的　1

第2章　Ruby　2
- 2.1　Ruby とは　2
- 2.2　Ruby の歴史　2
  - 2.2.1　CVS での最新ソースコードの入手方法　2
- 2.3　Ruby の特徴　3
- 2.4　Ruby の用途　7

第3章　dRuby　8
- 3.1　dRuby とは　8
- 3.2　dRuby の特徴　8
- 3.3　Ruby 風　9
  - 3.3.1　ブロック付きメソッド（イテレータ）　9
  - 3.3.2　例外処理機能　9
  - 3.3.3　マルチスレッド　9

第4章　RubyRobots　10
- 4.1　RubyRobots　10
  - 4.1.1　目的　10
  - 4.1.2　RubyRobots の仕様　10
  - 4.1.3　バトルフィールド　10
  - 4.1.4　ロボット　10
- 4.2　プログラマ・リファレンス　13
  - 4.2.1　サンプル実装　13
- 4.3　RubyRobots のしくみ　17

第5章　GUI　19
- 5.1　VisualuRuby　19
- 5.2　Ruby/SDL　19
  - 5.2.1　SDL　19
  - 5.2.2　SDL の機能概要　21
  - 5.2.3　周辺ライブラリ　22
  - 5.2.4　Ruby/SDL　23

第 6 章　システム構築　24
6.1　マルチエージェント環境　24
6.1.1　ロボット数　24
6.1.2　チーム分け　24
6.1.3　ロボット名　25
6.2　相互通信，協調動作環境　25
6.2.1　情報共有　25
6.2.2　相互通信　26
6.2.3　協調動作　27
6.3　GUI　27
6.3.1　チーム振り分け　27
6.3.2　ロボット選択　28
6.3.3　Ruby/SDL　28

第 7 章　評価実験　30
7.1　相互通信するエージェントの作成　30
7.2　対戦結果　30
7.3　GUI 評価　31

第 8 章　考察　32
8.1　相互通信について　32
8.2　GUI について　33
8.3　今後の課題　34

第 9 章　まとめ　37

謝辞

参考文献

付録

第1章　序論

1.1　研究背景

人間は2つの視点から協調的な問題解決をおこなっているとみることができる．ミクロな視点からは，人間の脳は膨大な数の神経細胞から構成される高度な並列システムと見なすことができる．人間のもつ知的能力がどのようにして実現されているのかを解明することは現代の科学における究極の課題であろう．またマクロな視点からは，人間は組織社会を形成し，互いに相互作用しながら日々の生活を送っている．近年のインターネットの社会における浸透は新しい人工社会の出現を予期するものである．その中で人と計算機が共存するための協調技術が必要とされている．

現在，こういった人間社会に代表されるような複数の計算主体（エージェント）による協調的な問題解決機構，つまりマルチエージェントシステムについて研究が進んでおり，多くのグループが取り組んでいる．代表例としてRoboCupが挙げられる．RoboCupでは分散・個人学習，リアルタイム推論などの研究がおこなわれ，マルチエージェントシステム環境をもちいた戦略プランニングの分野にも注目が集まっている．

一方プログラミングの世界では，今までにいろいろな言語が提案され利用されてきた．高速に動作する，短期間でプログラムできるなど，言語の目的はさまざまである．その中でRubyは，手軽なオブジェクト指向プログラミングを実現するためのスクリプト言語として開発され，めざましい勢いで発展している．

1.2　研究目的

本研究はRubyRobotsというロボット対戦シミュレーションにおいて，ロボット同士が協調的に行動するための性質，条件やエージェントが複数存在するときに生じる現象，効果などについて研究することを目的とする．より複雑な処理をするエージェントの作成を目標とし，それが可能となる環境とその処理を観察しやすい環境の構築をしていく．

RubyRobotsはプレイ中に集められた情報から相手の様子を推測しそれに基づいた決定を下すといった不完全情報ゲーム[¤]である．現状ではエージェント達はそれぞれ独立して行動しておりスキャン[¥]できる敵も1度に1体となっているため，4体で対戦させてもほぼ1対1と変わりがない．

そこで対戦できるロボット数を増やしチームプレイの要素を追加，また相互通信できる機能も追加することで，より複雑で協調する処理を可能にするエージェントの開発環境を構築する．その中で協調動作するエージェントの有効性をさぐる．

さらにRubyRobotsの欠点であるGUI部分を強化し視覚的なわかりやすさも追求する．

[¤] 相手の様子が分からないゲーム．ブリッジや麻雀など．

[¥] RubyRobotsのエージェントが敵を走査する行動

第2章　Ruby

2.1　Rubyとは

Ruby言語は「楽しく，手軽に本格的なプログラミングができる」というコンセプトのもとに，まつもとゆきひろ氏によって開発された，オブジェクト指向プログラミング言語である．ネーミングの由来は，代表的なスクリプト言語のひとつである，Perl（Pearl：真珠）にちなみ，宝石の名前からつけた．

2.2　Rubyの歴史

Rubyの歴史はTable2.1のようになっており，比較的新しい言語であることがわかる．また現在でも，まつもと氏を中心として，頻繁に更新がおこなわれている[1]．

Table 2.1: Rubyの歴史

1993年2月24日	Ruby誕生
1994年11月	実用的なレベルに到達．NetNewsで協力者を募集
1995年12月	fj.resoucesで初の一般公開（バージョン0.95）
1996年12月25日	Ruby1.0 リリース
1997年8月	Ruby1.1 リリース（ネイティブ32）
1997年12月	オンラインソフトウェア大賞'97Linux部門入賞
1998年12月25日	Ruby1.2 リリース
1999年8月13日	Ruby1.4 リリース
2000年9月	Ruby1.6 リリース
2001年12月25日	Ruby1.6.6 リリース
2002年3月	Ruby1.6.7 リリース
2002年12月14日	Ruby1.6.8 リリース
2003年現在	Ruby1.6.8

2.2.1　CVSでの最新ソースコードの入手方法

最新のソースコードを入手するのにはCVSを利用する．ただしネットに接続されていることが前提となる．

[1] URL: <http://www.ruby-lang.org>

前準備

前準備では「~/.cvsrc」を作成する．内容はつぎのように書く．

~/.cvsrc の内容

```
cvs -z9
update -d -P
```

cvs -z9 は，gzip の圧縮率を示す．CVS server の CPU を使うことになるため，あまり高圧縮はしないようにする．update -d -P は，cvs update コマンドを使うときに新しいディレクトリに対応し，空のディレクトリを削除する．

最初の接続

最初の接続の手順は次に示す通りになる．

- PPP など，ネットに接続
- CVS 用ディレクトリの作成 (ここでは /home/cvs とする)
- CVS 用の anonymous ユーザの登録 (~/.cvspass が作られる)
- ソースコードの checkout

最初の接続方法

```
$ vi ~/.cvsrc
$ cd /home/cvs
$ cvs -d :pserver:anonymous@cvs.ruby-lang.org:/ruby login
(Logging in to anonymous@cvs.ruby-lang.org)
CVS password: anonymous
$ cvs -d :pserver:anonymous@cvs.ruby-lang.org:/ruby checkout ruby
```

上記の手順で，最新のソースコードが入手できる．

日常の更新

CVS 上で修正がおこなわれた場合，最新の状態に保つには次のようにする．

日常の更新

```
$ cd /home/cvs/ruby
$ cvs update
```

修正されたソースコードを確認して最新の状態になる．

2.3 Ruby の特徴

インタプリタ

Ruby はインタプリタ型の言語であるために，コンパイルなどの手順なしに，プログラムを書いてすぐに実行が可能である．これにより，開発が非常に効率よくおこなうことができる．

変数や式に型がない

Ruby では変数や式による静的なチェックはおこなわず，実行の際に，値（オブジェクト）自身が適切な処理（メソッド）を選ぶようになっている．

また，式にも型がないため，変数や配列などにも，任意の値を格納することができる．

変数宣言が不要

変数をもちいる際に，宣言をする必要がない．ただし，ローカル変数と定数は，最初の代入の時に，宣言がされる．また，変数名の表記の仕方によって，変数や定数の分類がおこなわれている．

たとえば

変数宣言

foo（先頭が小文字）	⇒	ローカル変数
$foo（先頭が$）	⇒	グローバル変数
@foo（先頭が@）	⇒	インスタンス変数
Foo（先頭が大文字）	⇒	定数

というようにひと目で，変数や定数の種類を判別することができる．

すべてがオブジェクト

Ruby はオブジェクト指向言語として設計，作成されたため，すべてのデータがオブジェクトで，メッセージを送る統一的なインターフェースを持つ．

書きやすく読みやすい文法

文法が非常に簡単な構造になっているために，書くだけで，自然と読みやすい文書になる．よって後々の管理保守や他人のスクリプトの理解をしやすい．文法の特徴を以下にあげる．

- end で終わる構文
- 省略可能な引数
- 呼び出しの括弧が省略できる
- 改行で文を終了できる
- 演算子の再定義が可能

イテレータ

イテレータとは手続きの抽象化を実現する機能で，一連の手続き（ブロック）を渡されて実行されるメソッドのことである．イテレータを利用することにより，繰り返し構造，特に配列に対する処理を簡潔に記述することができる（List2.1，2.2）．

List 2.1: 配列に対する処理プログラム（イテレータなし）

```
1 list = [1, 3, 5, 7, 9]
2 sum = 0
3 for i in 0..4
4   sum += list[i]
5 end
6 print "合計：", sum, "nn"
```

List 2.2: 配列に対する処理プログラム（イテレータを利用）

```
1 list = [1, 3, 5, 7, 9]
2 sum = 0
3 list.each{|elem|
4   sum += elem
5 }
6 print "合計：", sum, "nn"
```

充実したクラスライブラリ

Ruby には以下のような充実した，使いやすいライブラリが標準で用意されている．

- 文字列処理
- 正規表現
- ファイル入出力
- ファイル操作
- データ構造
- 配列
- 数値演算
- プロセス操作
- ネットワーク入出力（ソケット）
- スレッド
- データベース操作（dbm，PostgreSQL など）
- GUI（Ruby/Tk，Ruby/Gtk）

これらによって提供される機能を組み合わせて使用すれば，手軽に思ったとおりのスクリプトを記述することが容易である．

システムプログラミング

Ruby にはほとんどの UNIX 系のシステムコールの呼び出し機能が備わっている．C 言語を一切使わず，Ruby 単体でシステムプログラミングをおこなうことも可能である．

ネットワークプログラミング

標準でネットワークにソケットインターフェースにアクセスするライブラリおよび，ネットワークプログラミングを支援するスレッド機能を備えているために，簡単にネットワークプログラミングをおこなうことができる．

GUI プログラミング

Tk，Gtk，などの GUI ツールキットへのインタフェースが提供されているため GUI プログラミングも容易である．

メモリの自動管理

Ruby にはポインタという概念がなく，あるのはオブジェクトに対する参照という概念だけである．オブジェクトの領域の割り当てとその管理はインタプリタが自動的におこなってくれるため，ユーザがメモリを気にしながらプログラミングをする必要がない．
また，ガーベージコレクタ（GC）により，自動的にメモリが開放されるので，厄介なメモリリークとも無縁となる．

例外処理

例外もオブジェクトになっているので，エラー処理を簡単に書くことができる．

C 言語による拡張

C/C++による拡張を前提としている．
C言語用の API[y]をもちいれば，Ruby と C 言語の関数や機能の相互補完をすることができ，最終的には実行速度が必要な処理は拡張ライブラリでカバーすることや，Ruby の機能を C/C++アプリケーションに組み込むといったことも可能である．

スレッド

Ruby はマルチスレッド機能を備えており，同時に複数の仕事を平行して実行できる．これは OS に依存しないアプリケーションレベル（ユーザレベル）スレッドなので，高速化には不向きであるが，効率化の面においては非常に有効である．
DOS などの低レベルな OS 上からでもスレッドを使うことができる．

マルチバイト文字に対応

マルチバイト文字（EUC，Shift-JIS，UTF-8 など）をサポートしており，日本語の文字列を扱った処理にも対応できる．

自由なライセンス

規制度の緩いライセンスで配布されているために，自由に改変，再配布，商用利用が可能である．

[y] Application Programming Interface．OS がアプリケーションに対して公開しているプログラムインターフェイス

2.4 Ruby の用途

Ruby は目的の種類に関わらず，大抵のものには無理なく使用することができる．Table2.2 に，Ruby の得意分野と苦手分野をまとめた．

Table 2.2: Ruby の得意，苦手分野

Ruby の得意分野	Ruby の苦手分野
・手早く作る小さなプログラム	・実行速度が要求されるプログラム[z]
・文字列の処理全般	・数値計算が主体のプログラム
・フィルタ系プログラム	・数万行以上にもなる巨大なプログラム
・システムプログラミング	・多人数によるプロジェクト
・ネットワークプログラミング	
・CGI プログラミング	

[z] ただし，実行速度は拡張ライブラリを使用することにより，ある程度はカバーできる

第3章　dRuby

3.1　dRubyとは

dRubyは関将俊氏が開発しRubyのオブジェクトをネットワーク上に拡張するライブラリである．ネットワーク上のRubyスクリプト間でメソッドの呼び出し，オブジェクトの交換を可能にする．ローカルのオブジェクトもリモートのオブジェクトも同じように操作することができる．

dRubyは現実的なよくある問題を解決するために設計されている．Rubyが気軽なオブジェクト指向プログラミングを提供するように，dRubyは気軽な分散システムを提供している．

ただし，大きなシステムの記述は想定されていない．

3.2　dRubyの特徴

dRubyには次の特徴がある．

- 対象をRubyに限定した分散オブジェクト環境
- 100％ Pure Ruby
- リモートオブジェクトをローカルと同じように使える
- 宣言（IDL等）が不要

dRubyは，対象をRubyオブジェクトに限定した分散オブジェクト環境である．Rubyオブジェクトだけしか扱えないが，その代わりにとてもシンプルになっている．また，Rubyが動けばdRubyも動作する．プラットフォームの差異はRubyが吸収してくれるので，UNIX-Windows間に分散するシステムも記述できる．

dRubyは純粋にRubyだけで書かれているライブラリである．Cなどによる拡張ライブラリは使用していない．dRubyの初期バージョンはわずか160ステップのRubyスクリプトであった．現バージョンのdRubyも基本部分はこのころと同じものである．Rubyの記述能力の高さを再認識させられる．

dRubyにはCORBAのIDLのような宣言や，javaRMI[¤]のスタブ／スケルトンの準備は必要ない．Rubyの変数に型宣言がないのと同様である．メソッドの結合は，Rubyと同じように動的におこなわれる．

つまりdRubyはRubyに強く依存しているということである．dRubyはできる限り自己主張せず，Rubyの特徴をそのまま残している．そうすることで，Rubyのもつ優れた能力（とくに気軽さ，楽しさ）を利用できるのである．

[¤] Remote Method Invocationの略で，別のホストのJavaオブジェクトのメソッドを呼び出すための通信手段

3.3 Ruby 風

Ruby 風な特徴を示す.

3.3.1 ブロック付きメソッド（イテレータ）

Ruby の特徴であるブロック付きメソッド（イテレータ）もそのまま使える．リモートオブジェクトによる繰り返しの処理に，ローカルなブロックを与えることができる．次のような例である．

List 3.1: dRuby の例

```
1 ary = ro.collect do jej
2   e.to_i
3 end
```

ro はリモートオブジェクト，do … end のブロックはローカルなブロックである．ro の collect メソッドは，ローカルなブロック付きでリモートの環境で実行される．collect は繰り返しごとにローカルな（そして ro から見るとリモートな）ブロックを実行し，その結果を返す.

3.3.2 例外処理機能

Ruby には例外処理機能が備わっている．dRuby でも同じように例外が扱える．リモートメソッドの処理中に発生した例外は，ローカルなメソッドの呼び出しと同じく例外が発生する.

3.3.3 マルチスレッド

Ruby には強力，かつ簡単に扱えるスレッド Thread がある．TCPServer と Thread をもちいて簡単にサーバを構築できる.

dRuby もこの機能を使って作られていて，複数の要求を同時に処理することができる．これにより，1 つのメソッド呼び出しを処理している間に別のメソッド呼び出しを受け付け，処理することができる.

第4章　RubyRobots

4.1　RubyRobots

1985年にTom Poindexterが「CROBOTS」を作成した.「CROBOTS」はCのサブセットになったインタプリタで作成されたものであり，内部に戦闘ロボットと，ロボットを操作するための関数が実装されている．プレイヤーは，インタプリタに対して，自分でロボットを操作するためのプログラムを作成し，実行させて戦わせる．1994〜96年にPoindexterは「CROBOTS」のTcl/Tk版，つまり「TclRobots」を公開した.「TclRobots」を2001年にarton氏がオブジェクト指向言語Rubyに移植したものが「RubyRobots」である．移植にあたっては，分散オブジェクト環境を実現するdRuby（分散Ruby）が利用されている．サーバ，ビュー，各ロボットプログラムはdRubyを利用してクライアント・サーバ形式で接続する．

4.1.1　目的

ロボットプログラムの目的は，他のロボットを破壊することと，自分のロボットを最後まで生き延びさせることのふたつである．

4.1.2　RubyRobotsの仕様

RubyRobotsでは，プレイヤーはロボットの行動を制御するプログラムを書きサーバ上で戦わせてどちらが優れているかを競う．ロボットプログラムの仕様は以下のとおりである．

- ロボットはバトルフィールド上を移動する
- ロボットはロボットを走査する
- ロボットはロボットにミサイルを撃つ
- ロボットはミサイルによってダメージを受ける

4.1.3　バトルフィールド

ロボットが戦う場所である．1000 × 1000[m] の広さで周囲には壁がある．壁にぶつかると5[%]のダメージをうけてしまう．バトルフィールドが1回のバトルで収容できるロボットは4台までになっている．また，左下の座標を（0，0）とする（Fig4.1）．

4.1.4　ロボット

ロボットのシステムの概要は次のとおりである．

- コンパスシステム

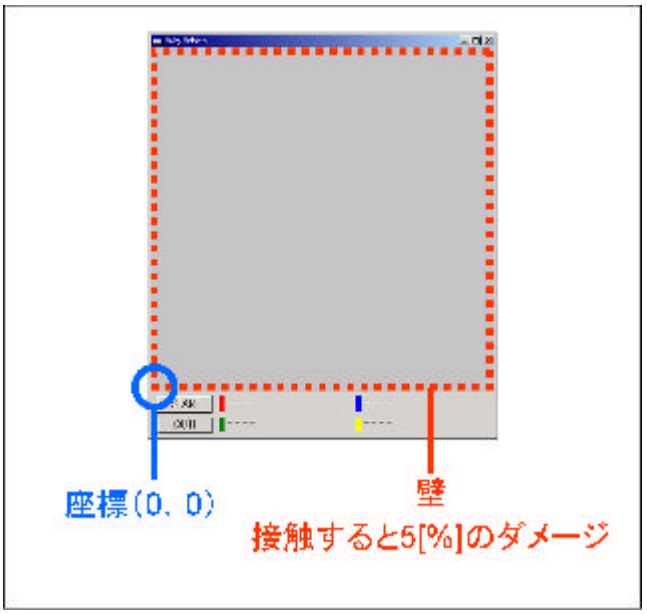


Fig 4.1: バトルフィールド

0～359 度までの全方位コンパスを備えている．0 が真東，90 が真北，180 が真西，270 が真南．すべてのインターフェースはコンパスシステムと連動しているので，方角についての指示はこのシステムを使用することになる．

- 移動システム
  速度はモータの回転率（0～100[%]）によって指定される．100[%] のフル回転だと秒速 200[m] になる．ただし，モーターが安全駆動可能なのは 35[%] までで，それを超えた速度で長時間駆動されるとオーバーヒートを起こす．オーバーヒートしたら（相対温度 200 を越えた場合）相対温度 0 に冷えるまで 35[%] が最高速度となる．加速，減速は，秒あたり 20[%] ずつ増減できる．

- スタビライザー
  移動システムのサブシステムである．ロボットの横転防止のために設置されている．許容以上の速度（Table4.1）で曲がろうとすると，この機構がモーターを強制的に停止させてしまう．同様に方向転換の度合いは速度に反比例する（Table4.2）．要するに速度が出てると，大回りになってしまうということである．

Table 4.1: 方向転換可能な速度

角度	最高速度 [%]
25 以下	100
25～50	50
50～75	30
75 以上	20

Table 4.2: 速度による方向転換率

速度	方向転換率 [%]
25 以下	90
25～50	60
50～75	40
75 以上	30

- **走査システム**
  スキャナーである．人間の目と違って前方しか見えないわけではないので，移動方向とは無関係にスキャン可能である．つまり 0～359 度が指定できる．また視野角はスキャン方向に対して± 10 度が設定できる．ただし，広角スキャンをおこなうと精度が落ちるという難点がある．1 度あたり ± 10[m] の誤差が生じてしまう．± 10 度のワイドスキャンだと最大 100[m] が誤差となる．当然± 0 度を指定した場合は正確な値が得られる．ただ複数のロボットがスキャンされた場合でも，一番距離が近い一台についてしかスキャンできない問題がある．

- **ミサイル**
  射程距離は最大 700[m]．速度は秒速 200[m]．発射方向は移動方向とは独立している．爆発すると半径 40[m] までダメージを与える（Table4.3）．ミサイルは 4 本 1 組で発射装置に装着する仕組みである．発射装置は一度発射すると 4 秒間は発射できない．4 本撃ち終わったら，次のミサイルの装着に 8 秒かかる（4 本おきに 12 秒発射できない時間ができる）．また目的地より手前にいるロボットにぶつかって爆発はしない．さらにミサイルを発射するとその熱の影響で走査システムにランダムで誤差が生じたり，広角スキャンに失敗することがある．ダメージによる影響はない．

Table 4.3: ミサイルの威力

爆心からの距離	ダメージ [%]
6	25
10	12
20	7
40	3

- **温度**
  移動によるオーバーヒートとミサイル発射による熱は独立した温度である．どちらかの温度がもう一方に影響することはない．

- **自己診断システム**
  現在位置，速度，現在のダメージなどのレポート機能．

4.2 プログラマ・リファレンス

Robot クラスには次のようなインスタンスメソッドが用意されている．

- r.scanner(dir, range)
  dir（0〜359）で指定した方角を，range（0〜10）で指定した視野角で走査する．一番近くの敵ロボットへの距離を返す．敵が見つからなかった場合は 0 を返す．

- r.drive(dir, speed)
  dir（0〜359）で指定した方角へ speed（0〜100）で指定したスピードで移動する．停止したい場合は speed に 0 を指定する．現在の速度を返す．

- r.cannon(dir, range)
  dir（0〜359）で指定した方角へ range（0〜700）で指定した射程距離でミサイルを発射する．再装填中の場合は 0 を，発射した場合は 1 を返す．

- r.damage
  ダメージを返す．

- r.speed
  現在の移動速度を返す．

- r.loc
  現在の X 座標と Y 座標を返す．

- r.heat
  現在のオーバーヒート状態とモーターの温度を返す．

- r.dsp
  最後に走査した敵ロボットの識別子とそのロボットのダメージを返す．

ユーザは各種のメソッドを活用して，他のロボットを破壊するプログラムを書くことになる（Robot クラスの派生クラスを作成して，その run メソッドにロボットを操作するプログラムを記述する）．基本仕様はすべて同じなので，あとはプログラム次第となる．

4.2.1 サンプル実装

ロボットプログラムの書き方を紹介する．

生きているロボット

基本となるロボットプログラムを作成する．このロボットはなにも行動はしない．基本形なのですべてのロボットはこれをもとに作成することになる．

List 4.1: 生きているロボット

```
1  require 'robot'
2
3  class Sample1 < Robot
4
5    def run
6      while alive do
7        sleep 1
8        p "sample1 !!"
9      end
10   end
11
12 end
13
14 if __FILE__ == $0
15   if ARGV.size != 1
16     $stderr.puts "usage: # f$0g rubyot_uri"
17     exit 1
18   end
19   Robot.main Sample1, ARGV[0]
20 end
```

実行時には run メソッドが呼び出されることになっているため，プログラムには必ず run メソッドが必要となる．また，ロボットが破壊された場合 run メソッドを抜けるのもルールになっているので，実際には「while alive do 〜 end」といったループの中に動作を記述することになる．List4.1，14 行目の記述が実行文である．引数の数が違う場合に実行させないようにしているのと，親クラスである「Robot クラス」の main を実行して dRuby を起動するといった処理をする．List4.1 をサーバで実行すると，1 秒ごとに"sample1 !!"という文字列を出力する．これ以外なにもしないので，破壊されるのを待つかサーバを停止させるかのどちらかで終了させることになる．

移動するロボット

破壊されるまで移動し続けるロボットである．実行文の部分は省略してある．

List 4.2: 移動するロボット

```
1  require 'robot'
2
3  class Sample2 < Robot
4
5    def initialize
6      cdir = [0, 90, 180, 270]
7      @cur_dir = cdir[rand(3)]
8    end
9
10   def run
11     while alive do
12       x, y = loc
13
14       bef = @cur_dir
```

```
15        case @cur_dir
16        when 0
17          @cur_dir = 90 if x > 900
18        when 90
19          @cur_dir = 180 if y > 900
20        when 180
21          @cur_dir = 270 if x < 100
22        when 270
23          @cur_dir = 0 if y < 100
24        end
25
26        p "#fbefg␣to␣#f@cur_dirg" if bef != @cur_dir
27
28        drive @cur_dir, 100
29        sleep 1
30      end
31    end
32
33  end
```

このロボットは壁にそって移動し続ける．方角は前述のとおり 0 が真東，90 が真北，180 が真西，270 が真南になる．現在位置から自分の進む方向を決定し，@cur_dir に方角の値をいれて drive メソッドで移動している．

攻撃するロボット

スキャン後，ミサイルを撃つロボットである．

List 4.3: 攻撃するロボット

```
1  require 'robot'
2
3  class Sample2 < Robot
4
5    def run
6      dir = rand 360
7      closest = 0
8      while alive do
9        rng = scanner dir, 10
10
11       if rng > 0 && rng < 700
12         start = (dir + 20) % 360
13         limit = 1
14         while alive && limit <= 30
15           dir = (start ¡ limit + 360) % 360
16           rng = scanner dir, 1
17           if rng > 0 && rng <= 700
18             cannon dir, rng
19             limit ¡= 4
20             next
21           end
```

```
22            limit += 1
23          end
24        else
25          if rng > 700
26            closest = dir
27          end
28        end
29
30        return unless alive
31
32        dir = (dir ¡ 20 + 360) % 360
33      end
34    end
35
36  end
```

このロボットはまず視野角±10度の広角スキャンを実行する．そこで敵を発見すると視野角を1に設定してその付近をもう一度スキャンする．再度敵を発見できればその方向にミサイルを撃つ．スキャンに関してはこのようなプログラムが基本になる．視野角±10度では誤差が生じているため，ミサイルを命中させるのは難しい．仕様の問題で角度にもずれが生じる可能性がある（Fig4.2）．

Table4.4に視野角とダメージ量の関係を示す．400[m]離れた地点にいる動かないロボットを，それぞれの視野角でサーチし攻撃した時の実験結果である．視野角が6度以上ではほとんどダメージを与えられていないことがわかる．

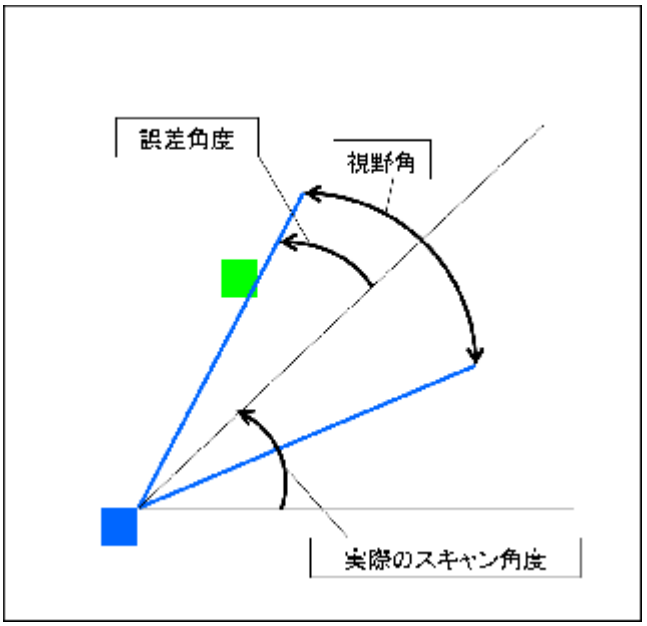


Fig 4.2: 広角スキャンで生じる誤差

Table 4.4: 視野角とダメージ量

視野角度	命中した回数（命中率）	1回の攻撃で与える平均ダメージ量 [%]
10	3（15%）	0.45
8	5（25%）	0.75
6	12（60%）	1.80
4	17（85%）	3.00
2	18（90%）	5.35
1	18（90%）	9.25

4.3 RubyRobotsのしくみ

RubyRobots は大きく分けて下の３つのプログラム群によって動作する．

- **メインプログラム**
  ロボットやミサイルの動きをシミュレートし，ロボットと通信して，ゲームを進行させる

- **プレイヤープログラム**
  プレイヤーごとにロボットの行動を規定する

- **ビューワ**
  Windows 専用の GUI を利用し，戦闘の様子を表示する

メインプログラムを実行することにより，サーバであるバトルフィールドが生成され，ロボットの参加を待つ状態になる．次に，各プレイヤーはサーバに dRuby **を利用して接続する．そのとき，参加機数が最大値を越えていなければ，そのプレイヤーは参加が認められる．このままでは内部で処理がおこなわれるだけなので観戦するためにビューワを使う．ビューワも** dRuby **を利用し，サーバに接続することで戦闘の様子を表示する（**Fig4.3**）．**

サーバとビューワには Observer **パターンが適用されており，サーバは自分の状態が変化したら，登録されているビューワ全てのメソッドをコールバックするようにしている．これにより，ひとつのサーバに複数のビューワが接続することが可能となる．戦闘の開始，終了などはこのビューから捜査する．参加プレイヤーがそろったら，戦闘を開始する．**

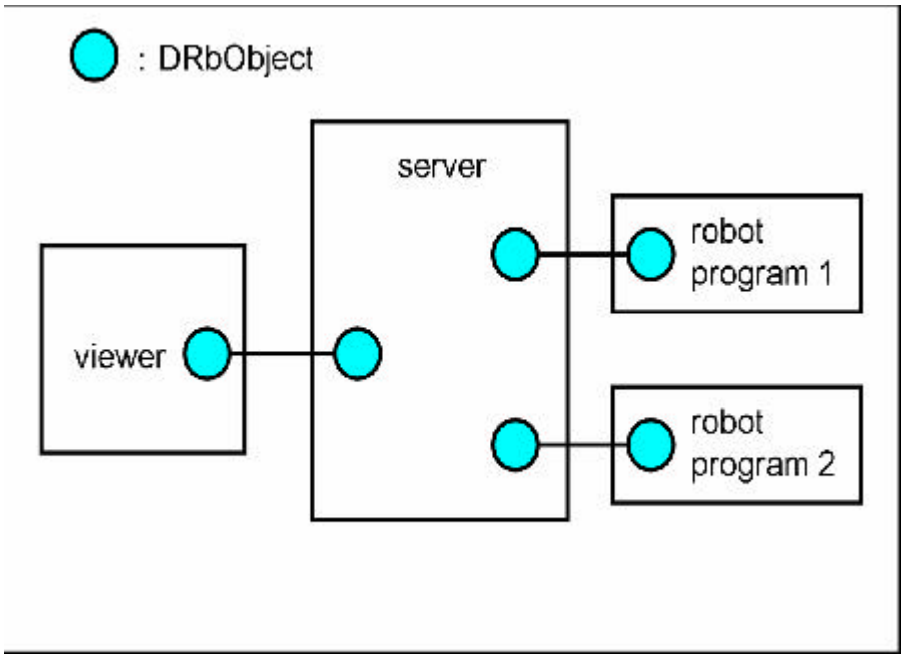


Fig 4.3: RubyRobots の仕組み

第5章　GUI

5.1　VisualuRuby

VisualuRuby は Windows での GUI を簡単に書けるようにするために開発された．特徴としては以下のものがある．

- ウィンドウを作り，表示し，消せる
- ウィンドウにコントロール（子ウィンドウ）を乗せられる
- コントロールのイベントを拾える
- Windows システムからのメッセージも選択して拾える
- コモンコントロールも使える
- 動画などのマルチメディア再生もできる
- DDE が扱える
- コントロールなどのサブクラス化が可能
- メニューが使える
- コモンダイアログも一部使える
- ウィンドウに描画できる
- ビットマップを扱える
- フォントもカーソルも扱える
- メッセージループのアイドル時処理を書ける
- モーダルダイアログが開けられる
- クリップボードもテキストなら扱える

5.2　Ruby/SDL

5.2.1　SDL

SDL の公式ページではSDL とは「Simple DirectMedia Layer の略で，さまざまなプラットフォームで使えるマルチメディア用のライブラリである」と紹介されている．DirectX[¤]のようなものであり，ゲームを作るためによく使われるライブラリである．DirectX と大きく違うのは，多くのプラットフォームで使えるというところである．作者は Lokigames にいた Sam Lantinga（現 Blizzard）他で，Windows などで動くゲームを Linux に移植する際に作られていたものだそうである．実際 Lokigames のいくつかの移植作にこれが使われている．

メインの用途はゲームだが，その他エミュレータ，MPEG のプレーヤなどに使われている．以下にその特徴を書いていく．

[¤] Windows にビルトインされた API 群

多くの対応プラットフォーム

SDL の最大の特徴である．公式に対応しているものに

- Linux
- Win32
- BeOS
- MacOS
- Solaris
- IRIX
- FreeBSD

の 7つがある．通常のプラットフォームは網羅されているといってよい．さらに非公式で対応しているものに，OpenBSD，NetBSD，AIX，OSF/Tru64，QNX，AmigaOS，WindowsCE などがある．

高速な描画

SDL のもうひとつの主要な特徴である．高速化の実現のために SDL 内部では DirectX や GDA[y]などを利用している．

さまざまな言語から使用できる

このライブラリは「C」で書かれ,「C」から使用するようになっている．C のほぼ上位互換言語である C++でも使える．しかしユーザの声からさまざまな言語から使用可能になっており，公式のサイトに挙げられているなかには，Ada，Ei®el，Euphoria，Guile，Java，ML，Pascal，Perl，PHP，Python，Ruby，C#などがある．

単純な API

SDL の API は非常に単純である．基本的な API の数は少なく，把握しやすい．ただそれは同時に，できることがあまり多くはないということでもある．描画処理と音声処理に関しては初歩的な機能しか持っておらず，回転縮小拡大機能がないことからもそれがわかる．

豊富な周辺ライブラリ

上記の不足している機能を補い，便利にするため SDL 用にさまざまなライブラリが存在する．公式のページに行けば，有用なライブラリが用意されている．さまざまなフォーマットの画像を使用できるようにするライブラリ，動画の表示，文字の描画，などである．SDL を使うときはこれらのライブラリを併用することが多くなる．

その他

API がゲームなどを作りやすいようになっている．通常の GUI とゲームの UI は相容れないところがあるので，ゲームを作るにあたっては非常に都合が良い．ただし通常の GUI をもったソフトは作りにく

[y] X Window System 上でフレームバッファに直接アクセスできる仕組み．描画の高速化を狙ったもの

いということでもある．
　またSDL のライセンスはLGPL である．つまりソースは公開されており，ソース非公開のソフトでもSDL を利用できる．

5.2.2　SDL の機能概要

　個々の機能について説明する．

Video

　SDL の主要な機能である画面描画機能．DirectX でいうDirectDraw[z]と同等である．以下にその機能をならべる

- 初期化
- フレームバッファへの直接アクセス{線や円を描くことができる
- BMP ファイルの読み込み
- 読み込んだ画像を描画
- 四角形の塗りつぶし

　ただし線や円は点で描き，画像描画には拡大縮小はない．多くのプラットフォームで使うためにそのような実装になっている．このような不足は別ライブラリで補うことができる．

Audio

　音を出す機能である．「ミキシング用のコールバック関数を登録してそれを別スレッドから呼び出して音を出す」という機能になっている．使用に関して不便であるが，この設計のおかげで柔軟性が高くなっている．実際はSDL_Mixer というライブラリを介することが多い．

Event

　入力機能．キーボードやマウスからの入力を取り扱うことができる．OS ごとに違う入力まわりの機能を隠蔽しているのでつかいやすい．終了処理（ウィンドウの終了ボタンを押した，kill でシグナルを送ったなど）の処理もここでおこなう．

Joystick

　ジョイスティック，ジョイパッドといった入力デバイスからの入力を取り扱う機能である．

CD-ROM

　CD-ROM の再生機能．Audio とは別になっている．

---

[z] DirectX の 2D 描画担当

Timer

SDL の初期化からの経過時間をミリ秒単位で取得する機能，ミリ秒単位で指定した時間だけ待機する機能，指定した時間ごとに指定した関数を呼び出す機能，などがある．

Thread

スレッド生成，セマフォ，Mutex などがある．OS ごとに違うスレッド関係の API を同じように使えるようになっている．

3D 描画

SDL にも 3D 描画機能がある．SDL の 3D 機能は OpenGL をそのまま利用している．ほぼ普通の OpenGL と一緒である．

その他

ウィンドウの最小化やキャプションの設定といった機能や，エンディアンの違いを対処するための機能などがある．

5.2.3 周辺ライブラリ

上述の通り SDL 用には多くのライブラリがある．Ruby/SDL でもこれを利用しているので以下に示す．

SDL_mixer

SDL の Audio 機能を使いやすくするためのライブラリである．Wave の演奏（WAV ファイルでのゲームの効果音など）と MOD，MIDI，MP3，Ogg Vorbis[x]などの演奏（ゲームならば BGM）が手軽にできる．

SDL_image

BMP 以外のグラフィックファイルを使うためのライブラリ．GIF や JPEG，PNG などその他にも様々なものが使える．

SDL_ttf

SDL で TrueTypeFont を描くためのライブラリ．日本語も描ける．

SGE

点，線，円，四角の描画，拡大縮小回転の実現，その他多くの機能が詰まったライブラリである．

[x]非独占的で，特許＆ロイヤリティーフリーなオーディオフォーマット．MP3 のようなもの

SMPEG

SDL で MPEG を再生するためのライブラリ

5.2.4 Ruby/SDL

手軽で強力なオブジェクト指向機能，扱いやすい例外機能をもつ Ruby と SDL を一緒にすることでゲームを作りやすくする，といった考えから Ruby/SDL は作られた．上で示した SDL 本体の機能は Thread，Audio，Timer の一部の機能を除いた，Video，Event，Joystick，CDROM，Window 関連，などが使用できる．Audio の機能がないかわりに SDL_mixer の機能が使えるようになっており，SGE を利用することで拡大縮小回転や線，円を描くこともできる．

SDL_ttf を使って文字を描くことや，SDL_image で PNG などの画像ファイルを利用することも可能である．SMPEG で MPEG のムービーを流すこともできる．また，3D の画像を表示することもできるが，これは OpenGL Interface という別の Ruby 拡張ライブラリをもちいることで可能になっている．

また，現在のところ Ruby/SDL が使用可能であることがわかっているのは Linux，Windows，FreeBSD，BeOS の 4 つとなっている．

第6章　システム構築

6.1　マルチエージェント環境

協調動作にむけて環境を構築した.

6.1.1　ロボット数

バトルフィールドの MAX_ROBOTS 定数（ロボットの状態が入る配列の要素数などを決定する定数）を 4から 8 に変更し，最大参加ロボット数を 8 に増やした（付録 List9.1，498 行目参照）.

6.1.2　チーム分け

チーム分けをおこなえるように各ロボットにチームナンバーの要素を追加した．また，サーチしたときに相手のチームナンバーを dsp メソッド（4.2 節参照，本来は最後にサーチしたロボットの ID とダメージを返すだけのメソッド）の返り値のひとつとして得られるようにしている．dsp メソッドはインスタンス変数である@sig という配列の値を返すことでその機能を実現している．その値を変更しているのが set_sig メソッドであり，このメソッドは scan メソッドが呼ばれたときに実行される do_scanner メソッドにて更新される（List6.1）．@sig 配列の 3 番目の要素が追加したチームの要素である.

List 6.1: チーム判別メソッド

```
1  def set_sig(dsp, dmg, dteam)
2      if @btemp >= SCANBAD
3        @sig.fill 0
4        return false
5      else
6        @sig[0..2] = dsp, dmg, dteam
7      end
8      true
9    end
10 end
11
12 def do_scanner(robot, deg, res)
13   rob = @robs[robot]
14   return 0 if rob.nil?
15   show_scan rob, deg, res
16   dsp, dmg, dteam, near = 0, 0, 0, 9999
17   @robs.each do |robot|
18     next if (robx.nil? || robx == rob || !robx.alive)
19     t, near = robx.check_scan(rob, deg, res, near)
20     if t
21       dsp = robx.num
```

```
22        dmg = robx.cdamage
23        dteam = robx.team.to_i
24        iteam = rob.team.to_i
25        if dteam != iteam
26          @scanlist[iteam] = [[near, deg, res, rob.pos], @ticks, rob.num]
27        end
28      end
29    end
30
31    if rob.set_sig(dsp, dmg, dteam)
32      return near unless (near == 9999)
33    end
34    0
35  end
```

これによって最後にサーチした相手のチームナンバーがわかるようになった．同じチームナンバーを持つロボットに攻撃を加えないようにするには，攻撃する際に自分のチームナンバーと相手のチームナンバーを比較することが必要になる．具体的な例は List6.2 の通り．ちなみに myteam メソッドは自分のチームナンバーを返すメソッドである．

List 6.2: 攻撃する際の比較の例

```
1 rng = scanner dir, 10
2 if (rng > 0 && rng < 700) && (myteam != dsp[2])
3   cannon dir, rng
4 end
```

6.1.3 ロボット名

ロボット名は適当な数字で与えられていたものを「ロボット ID — チームナンバー」のように与えるようにした（付録 List9.1，266 行目参照）．ロボット ID が 2 でチームナンバーが 3 ならば「#2 — 3」と表示される．

6.2 相互通信，協調動作環境

ロボット同士が相互通信，協調動作する環境を構築した．

6.2.1 情報共有

チーム情報をサーバ側で保存し，共有できるようにした．List6.1 の 26 行目の部分にて共有情報を作成している．@scanlist 配列はチームごとの情報を持っているインスタンス変数である．共有するのは同じチームのロボットが最後にスキャンに成功（敵を発見）した方向，距離，スキャン範囲，現在の位置とスキャンした時間，スキャンした相手のロボット ID である．情報取得用のメソッドを実行すると，@scanlist 配列に入っている自分のチームの情報が返ってくる．

6.2.2 相互通信

ロボット同士が dRuby を使用して相互通信（情報共有）するプログラムを作成した．Fig6.1 は通信時のつながりを示したものである．ロボットのひとつがサーバの役割を担い共有情報を保存，そのサーバに他のロボットが接続し情報をえる，という形で実装した．List6.3 にサーバロボットの 1 部分を示す．20 行目で DataKeeper オブジェクトを作成しており，21 行目で dRuby のサービスを開始するための DRb モジュールのメソッドを実行している．第 2 引数の DataKeeper オブジェクトは，第 1 引数の URI に関連付けられるオブジェクトであり，他のロボットがこのサーバにアクセスするとこのオブジェクトが反応することになる．

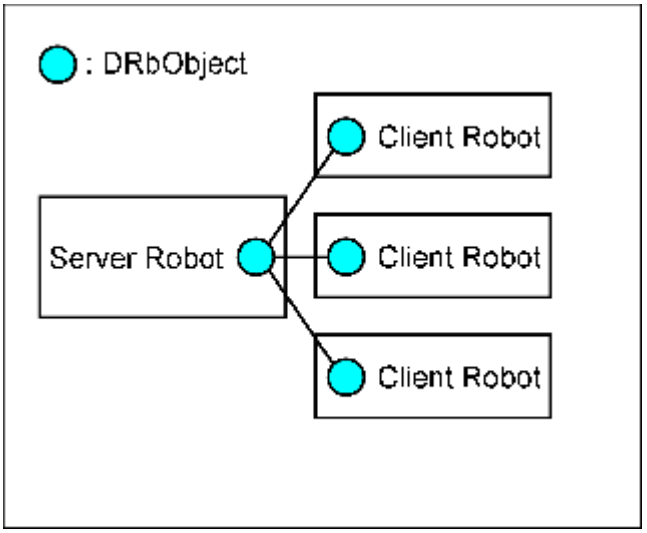


Fig 6.1: 相互通信するロボット

List 6.3: Server の役割を担うロボット

```
1  class Datakeeper
2    def initialize
3      @data = []
4      @mutex = Mutex.new
5    end
6
7    def updata(near, deg, res, x, y, time)
8      @mutex.synchronize do
9        @data = [[near, deg, res, [x, y]], time]
10       return @data
11     end
12   end
13
14   def popdata
15     return @data
16   end
17
18 end
19
20 data = Datakeeper.new
```

```
21 DRb.start_service('druby://:8470', data)
22 puts 'Data URI: ' + DRb.uri
```

List6.4 にクライアント側のロボットのプログラムを 1 部分示す．この記述だけでその URI に関連付けられたオブジェクトを扱えるようになる．「data.updata(serch, dir, res, x, y, time)」などとすればサーバにデータを送ることができ，「tsc = data.popdata」でサーバ側のデータを入手できる．

List 6.4: Client の役割を担うロボット

```
1 DRb.start_service
2 uri = "druby://PINOKIO:8470"
3 data = DRbObject.new(nil, uri)
```

6.2.3 協調動作

相互通信を実装したことにより協調動作がとれるようになった．実際に RubyRobots のサンプル「Charger」（敵を見つけるとその方角にミサイルを打ちながら近づくロボット）を基本として協調動作するロボットも作成した（付録 List9.2 参照）．

6.3 GUI

GUI 表示の変更や，機能の追加などをおこなった（Fig6.3）．

6.3.1 チーム振り分け

チームナンバーはビューの側のコンボボックスにて与えられるようにした（List6.5）．

List 6.5: チーム振り分けメソッド

```
1 def change_team(robot, team)
2   name = @robs[robot].changeteam(team)
3
4   p name
5
6   Thread.start do
7     sleep 0.1
8     @observer_peers.each do |i|
9       p i
10      i.update ["team", robot, name]
11    end
12  end
13
14 end
```

ビュー側はコンボボックスで与えられたチームナンバーをサーバの change_team メソッドに渡す．2 行目の changeteam メソッドは新しく追加したロボットのメソッドで，チームナンバーを変更し更新されたロボット名を返す．これを利用してロボットの持つチームナンバーを変更し，その後 6〜12 行目の記述でサーバは各 Observer（ビューワ）に新しいロボット名を通知するという形になっている．

6.3.2 ロボット選択

オリジナルではCUIでロボットを参加させていたが，これをビュー側の「OPEN」ボタン（GUI）で参加できるようにした．

6.3.3 Ruby/SDL

Ruby/SDLを利用したビューワを作成．Fig6.4がその現状である．８人プレイでもスムーズに動いていることを確認した．また，こちらのビューワでは爆発の描画を画像を利用して表示している．Fig6.2のような画像を用意しておき，左から連続で表示することで実現した．List6.6のプログラムは爆発を表示するためのメソッドである．

List 6.6: 爆発描画メソッド

```
@image    = SDL::Surface.loadBMP('explosion.bmp')
@image.setColorKey( SDL::SRCCOLORKEY|SDL::RLEACCEL, @image.getPixel(0, 0))
@image = @image.displayFormat

def show_explode(rob, x, y)
  l, t, r, b = x / 2, y / 2, 3, 3
  m = @missile[rob]
  unless m.nil?
    del_rect m
    @missile[rob] = nil
  end

  @canvas.updateRect(0, 0, 0, 0)

  xx, yy = l-12, t-12
  for i in 0..10
    SDL.blitSurface(@image, i*24 , 0, 24, 24, @canvas, xx, yy)
    @canvas.updateRect(xx, yy, 24, 24)
    sleep 0.02
  end
end
```



Fig 6.2: 爆発描画用画像

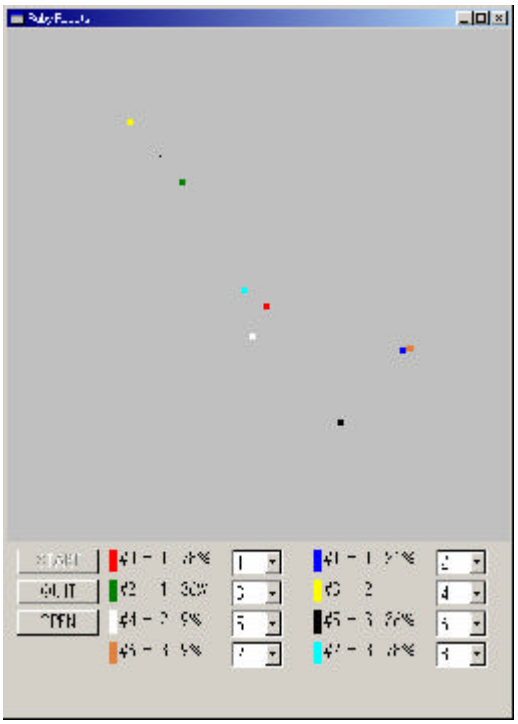


Fig 6.3: 機能を追加したビューワ

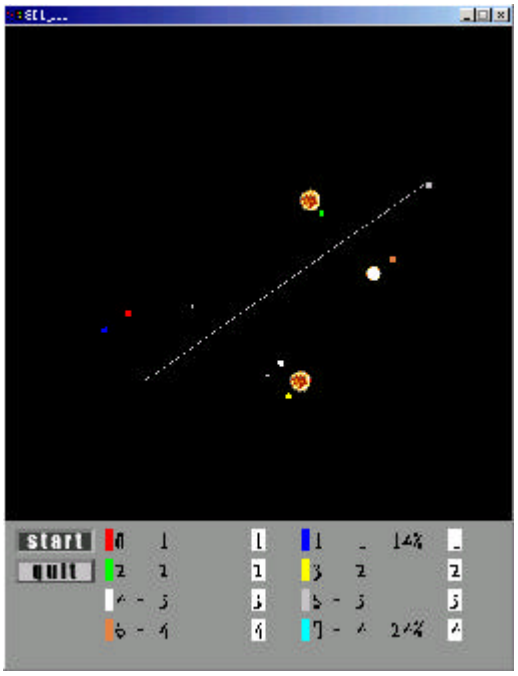


Fig 6.4: Ruby/SDL を利用したビューワ

第7章　評価実験

7.1　相互通信するエージェントの作成

相互通信を実装したことにより協調動作がとれるようになった．実際にRubyRobotsのサンプル「Charger」（敵を見つけるとその方角にミサイルを打ちながら近づくロボット）を基本として協調動作するロボットを作成した．機能として2つ，追加している．1つは同じチームのロボット（自分も含む）が敵を発見するとそのチームのロボットすべてがその付近を±1度の視野角でスキャンする，というもの．もう1つは±1度の視野角で敵を発見したら，同時にその地点にミサイルを発射するというものである．
付録List9.2はクライアント側のChargerのプログラムである．このプログラムは以下の2つのメソッドを利用して同時攻撃を可能にしている．

calcメソッド

List9.2，26行目．共有情報から敵の座標を計算するもの．

zatackメソッド

List9.2，5行目．与えられた敵の座標と，自分のいる座標から攻撃する方向と距離を計算するもの．

7.2　対戦結果

比較対象として「測研ロボット」（前年度の文献講読で優勝したロボット）を選び戦わせた．付録List9.3がプログラムである．
このロボットは「Charger」に対して強い，という特性を持っている．ポイントは2つある．ひとつはスレッドで「keep_moving」メソッドを常に実行しているところだ．これにより常に壁に沿って時計回りの方向に移動することになる.「Charger」は時計回りにしかスキャンしないため，見つかりにくいのである．もうひとつは最初に±10度の視野角でスキャンした後，±3度でスキャンしている（「Charger」の場合，±1度でスキャンしている）ところである．こうすると誤差が多少生じるが発見する確率が上がる．つまり,「Charger」に対しては発見されにくく，ダメージ量は少ないが確実にダメージを与えるので時間をかければ勝てるのである．
下の表は2つのロボットをそれぞれ2対ずつチームにして20回戦わせた結果である．

Table 7.1: 相互通信を実装していない「Charger」と「測研ロボット」の対戦結果

ロボット名	測研ロボット2体	Charger（相互通信なし）2体
勝数	16	4

Table 7.2: 相互通信を実装した「Charger」と「渕研ロボット」の対戦結果

ロボット名	渕研ロボット 2 体	Charger（相互通信あり）2 体
勝数	5	15

7.3 GUI 評価

ロボット数の増加とチームプレイの追加に関する評価，オリジナルのビューワと Ruby/SDL で作成したビューワとの比較の評価が必要である．02 年度文献購読生に動いているものを見てもらいアンケートをとった．主な意見としては

- 画面が黒い SDL の方が見やすい
- 爆発のエフェクトがわかりやすい
- SDL の方がスムーズに動いているように感じる
- ロボットが四角なので状態がわかりにくい
- スキャン時の線の描画がちらついてみづらい
- ダメージの表示方法がわかりにくい
- ロボットが重なることがあり，みづらい

などがあった．

第8章　考察

8.1　相互通信について

Table7.1，7.2 より，相互通信が有効であるということがわかる．勝因としては

- スキャン成功率のアップ
  相互通信しているのでひとつのロボットが敵を見失っても，もうひとつのロボットが見つけることができる．
- 集中攻撃
  ひとつの敵に集中して攻撃しているので確実に相手ロボット数を減らせる．

が挙げられる．また，敗因として以下のようなものがあった．

- 同じチームのロボットと重なる
  約 30[%] の割合でだが，同じ敵を追っていて同じチームのロボットと重なる，という現象が発生し敵のミサイルに同時にダメージを受けてしまうということがあった（Fig8.1）．これは「Charger」の敵を追うという特性が災いしている．

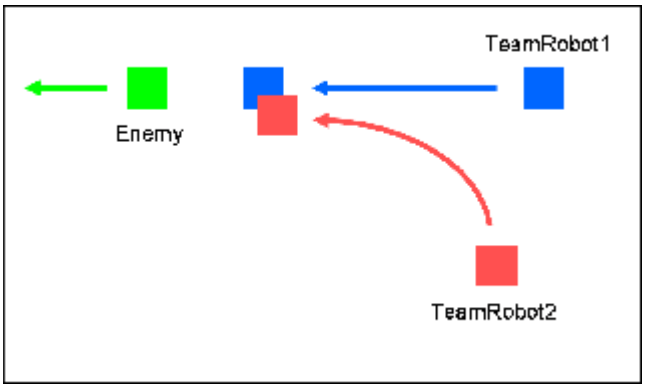


Fig 8.1: 同じチーム同士で重なるロボット

- 敵と重なる
  敵との距離が近くなり，そのうち追い越してしまうことがある（Fig8.2）．これもこのロボットの特性によるものなのだが，一度見失ったあと後ろをスキャンし始めるのには時間がかかってしまうし，その間は硬直してしまう．また，敵は常に壁ぎわを移動するロボットなので，追い越してしまったあと壁によるダメージを受けることがあった．

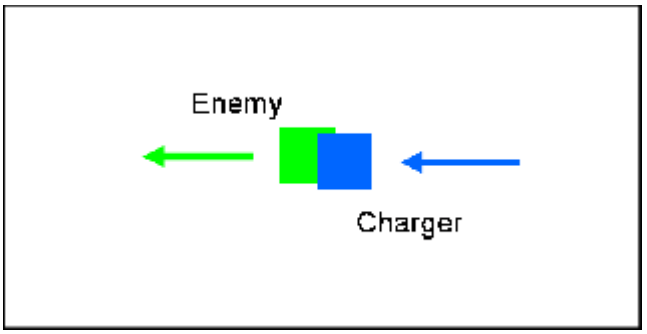


Fig 8.2: 敵と重なるロボット

ロボットが重なってしまう現象の改善方法としては，常に自分のチームのロボットの位置を把握し，距離をとるということが考えられる（Fig8.3）．

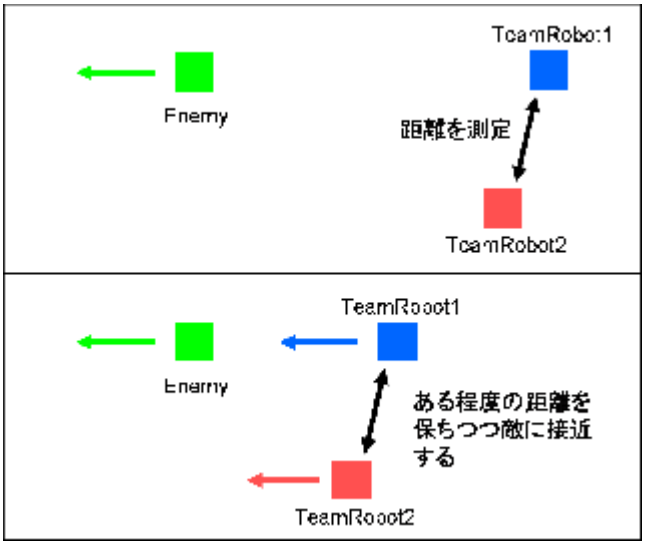


Fig 8.3: チーム同士で距離をとるロボット

敵と重なる場合の改善方法は，敵と自分の距離を把握し，ある程度距離をとるということが考えられるだろう（Fig8.4）．

8.2 GUI について

アンケートからもわかる通り，ロボットが四角でしか表示されないというのはビューワの欠点である．これではロボットがどの方向に動いているのかを瞬間的に把握することが困難である．改善方法としてロボットを矢印などで表示することが考えられる．ロボットをグラフィックで表示し，それを回転させることでも同等の効果を挙げられるだろう．

ロボットが重なってしまうというシステム的な面の不自然さも指摘されている．ロボット同士が接触した場合の処理（たとえば両方ダメージを受ける，など）も追加するべきである．

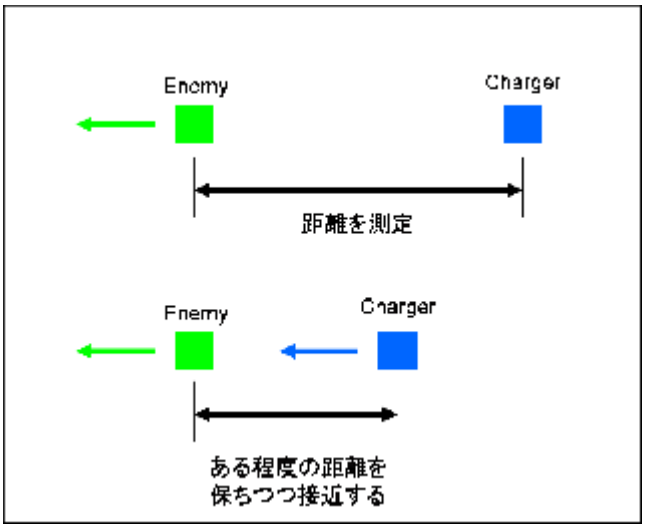


Fig 8.4: 敵と距離をとるロボット

8.3 今後の課題

ロボットが重ならないようにするのはシステム側で対処するとしても，ロボットが同じ位置に移動してしまうという戦略的な問題を解決しなければならない．これはそれぞれのロボットが行動する領域を決めておき，その領域内でしか移動できないようにすることで対処できる．例えば領域を左上，左下，右上，右下の 4 つに分け，その領域内を巡回するなどが考えられる（Fig8.5）．

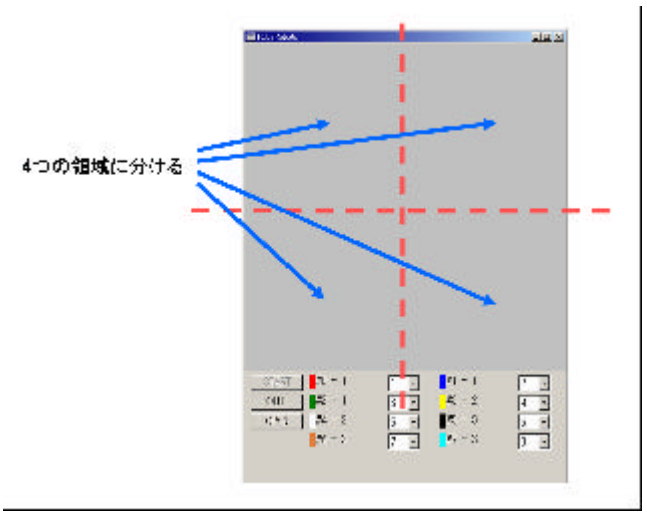


Fig 8.5: 4 つの領域

自分のチームのロボットの位置が把握できていないことも原因のひとつであると考えられる．解決するためには敵の位置をサーバに送るときに作ったような dRuby 用の新しいメソッドを作ればいいだろう．しかし，このように必要になったときに毎回 dRuby 用のメソッドを追加するという方法は PC に負荷がかかってしまうし手間がかかる．

つまり dRuby 用にはクライアントが「送るためのもの」と「受けるためのもの」，それぞれひとつずつのメソッドだけを用意し，送られてくる情報に応じてその処理を変更していけばいいのである．

クライアントがデータを「送るためのメソッド」は現在開発してあるメソッドをある程度改良すればできるだろう．問題はサーバがクライアントにデータを渡すときのメソッドである．実装方法には 2 つ考えられる．

1つ目は必要なデータをクライアントが好きなときに取り出せる方法．Fig8.6 がデータのやりとりを想定した図である．

1. 敵を発見したり自分の状態が変化すると，メッセージを送る．
2. データが欲しいときは,「受け取るためのメソッド」の引数に欲しいデータに対応した文字列を指定して送る．するとサーバがそれに対応したデータを返り値として渡す．

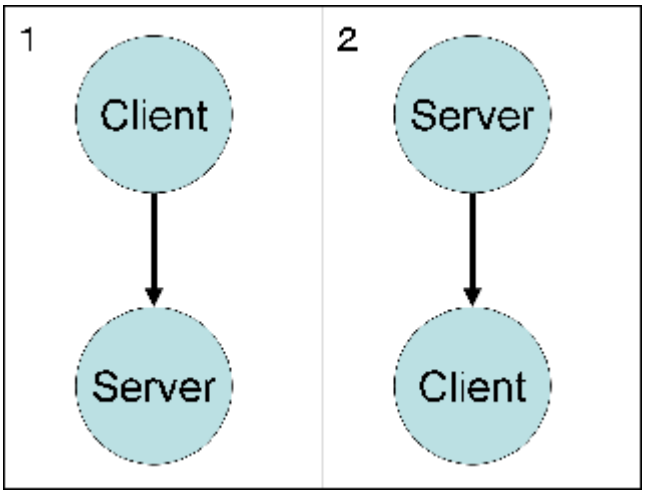


Fig 8.6: データのやりとり 1

この方法ではサーバが送る情報は少なくて済むだろう．ただしデータのやりとりの回数が増えて競合問題などが発生してしまうと思われる．

2 つ目はクライアントがサーバに Observer 形式で接続する方法．Fig8.7 がデータのやりとりの図である．

1. 1 つ目の 1. と同じ
2. Server は自分の持っているデータの状態が変化するとそのデータを Client すべてに送る．実際にはクライアントが一定時間ごとに update メソッドを実行してサーバのデータを取り出す．

この方法では一定の間隔で 1 度にすべてのクライアントにデータを送ることができる．ただしデータの量が増えてしまうことが考えられ，PC の負担が大きくなるだろう．

これらの方法の選択，改良が今後の課題となるだろう．

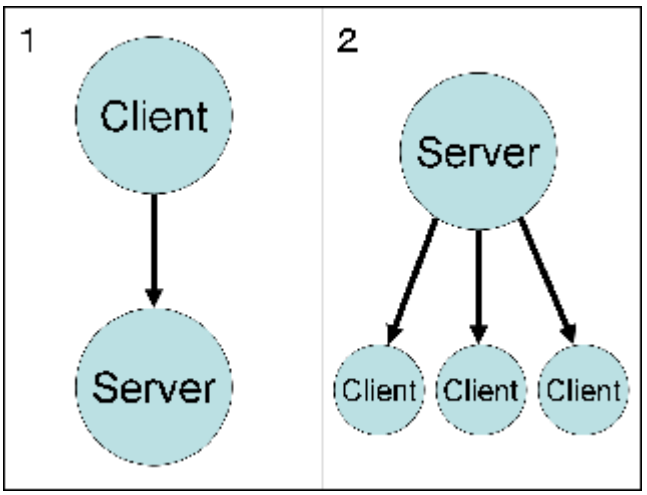


Fig 8.7: データのやりとり 2

第9章　まとめ

相互通信するロボットプログラムで大事なことは

- 常に動き続けること
  具体的には付録 List9.3 の測研ロボットのように動き続けるためのメソッドを用意し，それを別スレッドで実行し続けることが有効である．

- チームのロボット同士と敵とは距離をとること
  dRuby 用に新しいメソッドを作り，同じチームのロボット全ての位置がわかるようにするなどが考えられる．

であるということがわかった．特に距離をとることについてはサッカーなどでもおこなわれている，スペースを作る，使うといったところに通じるだろう．

研究を通じて感じたことは，協調動作するだけで強いロボットができるというわけではないということだ．実際，相互通信を実装しているものとしていないものの対戦でも実装していないものに 100%勝つことができていない．各エージェントが共有した情報からもっと多くの計算をしなければならないだろう．

しかしこのようにエージェントが協調し，ひとつの問題に取り組むことができる環境を構築できたことは，Ruby の可能性を提示すると共にマルチエージェントシステムの発展に役立てたのではないだろうか．ただし，いまだ実装できていない機能の追加や研究は必要ではある．

本研究では GUI についても検討した．作成した GUI への多くの意見は見づらい，使いにくい，というものであった．小さな四角が動いているだけでは観察のしやすさという点において不十分であると考えられた．GUI に優れた既存プログラムの検討，さらに人間工学に基づくアイテムの配置などの研究の必要性を感じる．

謝辞

本研究に際し，終始ご指導いただきました渕一博教授には深く感謝いたします．

また，検討しながら共に研究してきた中野友裕氏に感謝します．辛くなったときにはお互いに励ましあいました．彼がいるからこそこの研究を続けられたのだと思います．

研究の方向性におきまして良きアドバイスをいただいた税田竜一氏には感謝いたします．税田氏には研究でつまるといつも的確な言葉で道を開いていただきました．

研究の補助をしていただいた岩附拓治氏に感謝いたします．岩附氏には研究室の中では最もお世話になりました．特に SDL を利用したビューワのデザインに関して決定的な助言をいただきました．SDL ビューワが完成したのは岩附氏のおかげです．

ソフトを提供していただいた比留間亮一氏には，この場を借りて感謝の意を述べたいと思います．比留間氏にはゲームに欠かせない log の記録／再生ソフトを作成していただきました．思えばそのときから研究が大きく前進したと感じます．

最後に，多大な迷惑をかけてしまったのに快く協力してくれた渕研究室の学部生の皆さん，また，共に励ましあった越田研究室，上原研究室の皆さんにも感謝いたします．

参考文献

[1] 高橋征義，後藤裕蔵著，『たのしい Ruby』，（オーム社，2002 年）

[2] ただただし，arton 著，『Ruby を 256 倍使うための本　網道編』，（アスキー，2001 年）

[3] Ruby を 256 倍使う会著，『Ruby を 256 倍使うための本　紅玉制覇編』，（アスキー，2001 年）

[4] Ruby を 256 倍使う会著，『Ruby を 256 倍使うための本　場外乱闘編』，（アスキー，2001 年）

[5] RubyUnit 編著，『RubyGemBox』，（オーム社，2002 年）

[6] RubyUnit 堀川久編著，『Ruby de CGI』，（オーム社，2002 年）

[7] 前田修吾，まつもとゆきひろ，やまだあきら，永井秀利著，『Ruby アプリケーションプログラミング』，（オーム社，2002 年）

[8] デビット・トーマス＋アンドリュー・ハント著，田和勝訳，まつもとゆきひろ監修，『プログラミング Ruby』，（ピアソン・エデュケーション，2001 年）

[9] 原信一郎著，まつもとゆきひろ監修，『Ruby プログラミング入門』（オーム社，2000 年）

[10] 関将俊著，『dRuby による分散オブジェクトプロブラミング』，（アスキー，2001 年）

付録

Server プログラム

List 9.1: 機能を追加したサーバ（rbrobot.rb）

```
=begin
 Copyright(c) 2001 arton
== ACKNOWLEDGMENTS
  I've␣used␣many␣ideas,␣architecture,␣and␣codes␣from
␣␣␣␣Mr.␣Tom␣Poindexter␣(tpoindex@nyx.net)'s TclRobots.
  This is almost TclRobots Ruby port except for using ORPC.
=end

require 'thread'
require 'drb/drb'
begin
  require 'drb/eq'
rescue LoadError
  class DRbObject
    def __drburi
      @uri
    end
    def ==(other)
      return false unless DRbObject === other
      (@ref == other.__drbref) && (@uri == other.__drburi)
    end
  end
end
require 'observer'
include Math

srand(((Time.new.to_i % 65277 + 13849) % 65536 + 65536) % 65536)
D2R = 180/PI
S_TAB, C_TAB = Array.new(360), Array.new(360)
(0...360).each do |i|
  S_TAB[i] = sin(i/D2R)
  C_TAB[i] = cos(i/D2R)
end

def hypot(x, y)
  sqrt(x ** 2 + y ** 2)
end

class Point
  def initialize(a)
    @x, @y = a
```

```
42    end
43
44    attr_accessor :x, :y
45
46    def ¡(other)
47      return @x ¡ other.x, @y ¡ other.y
48    end
49
50    def ==(other)
51      return false unless Point === other
52      @x == other.x && @y == other.y
53    end
54
55    def towin
56      return @x, 1000 ¡ @y
57    end
58  end
59
60  class Moveable
61    def initialize(a = [0,0], hdg = 0)
62      @cur = Point.new(a)
63      @org = Point.new(a)
64      @hdg, @range = hdg, 0
65    end
66    attr_reader :cur, :org
67    attr_accessor :range, :hdg
68
69    def pos
70      return @cur.x, @cur.y
71    end
72
73    def initnew(pnt, deg)
74      @cur, @org, @hdg, @range = pnt.cur.dup, pnt.cur.dup, deg, 0
75    end
76
77    def hypot(other)
78      sqrt((@cur.x ¡ other.cur.x)¤¤2 + (@cur.y ¡ other.cur.y)¤¤2)
79    end
80
81    def move(deg)
82      @cur.x = (C_TAB[@hdg] ¤ deg + @org.x).round
83      @cur.y = (S_TAB[@hdg] ¤ deg + @org.y).round
84    end
85  end
86
87  class Missile < Moveable
88    def initialize
89      @dist = 0
90      super
91    end
92
93    def fire(pnt, deg, rng)
94      @dist = rng
95      initnew pnt, deg
```

```
96    end
97
98    def update(spd)
99      @range += spd
100     move @range
101   end
102
103   def explode
104     move @dist
105   end
106
107   def explode?
108     @range > @dist
109   end
110 end
111
112 class Motor < Moveable
113   RATE = [90, 60, 40, 30, 20]
114   HRATE = 10
115   HEATSP = 35
116   HEATMAX = 200
117   TURN = [100, 50, 30, 20]
118   ACCEL = 10
119   SP = 10
120   COOLING = ¡25
121   COLL = 5
122
123   def initialize(a = [0,0], hdg = 0)
124     super
125     @dhdg = @hdg
126     @dir, @speed, @dspeed = '+', 0, 0
127     @heat, @overheat = 0, false
128     @dirty = true
129   end
130
131   attr_reader :speed, :heat, :overheat, :dirty
132
133   def drive(deg, spd)
134     d1 = (@hdg ¡ deg + 360) % 360
135     d2 = (deg ¡ @hdg + 360) % 360
136     d = (d1 < d2) ? d1 : d2
137
138     @dhdg = deg
139     @dspeed = (@overheat && spd > HEATSP) ? HEATSP : spd
140     idx = (d / 25).to_i
141     idx = 3 if idx > 3
142     if @speed > TURN[idx]
143       @dspeed = 0
144       @dhdg = @hdg
145     else
146       @org = @cur.dup
147       @range = 0
148     end
149     if (@hdg + d + 360) % 360 == deg
```

```
150       @dir = '+'
151     else
152       @dir = '¡'
153     end
154     @dspeed
155   end
156
157   def adjust
158     if @speed > HEATSP
159       @heat += ((@speed ¡ HEATSP) / HRATE).round + 1
160       if @heat >= HEATMAX
161         @heat = HEATMAX
162         @overheat = true
163         if @dspeed > HEATSP
164           @dspeed = HEATSP
165         end
166       end
167     else
168       if @overheat jj @heat > 0
169         @heat += COOLING
170         if @heat <= 0
171           @overheat = false
172           @heat = 0
173         end
174       end
175     end
176   end
177
178   def accelate
179       @speed != @dspeed
180     if @speed > @dspeed
181       @speed ¡= ACCEL
182       @speed = @dspeed if @speed < @dspeed
183     else
184       @speed += ACCEL
185       @speed = @dspeed if @speed > @dspeed
186     end
187   end
188
189   def stabilize
190     if @hdg != @dhdg
191       mrate = RATE[(@speed/25).to_i]
192       d1 = (@dhdg ¡ @hdg + 360) % 360
193       d2 = (@hdg ¡ @dhdg + 360) % 360
194       d = (d1 < d2) ? d1 : d2
195       if d < mrate
196         @hdg = @dhdg
197       else
198         @hdg = ((@hdg.__send__ @dir, mrate)+360)%360
199       end
200       @org = @cur.dup
201       @range = 0
202     end
203   end
```

```
204
205   def do_drive
206     damage = 0
207     if @speed > 0
208       @range += (@speed ¤ SP / 100)
209       move @range
210       @dirty = @cur != @org
211       if @cur.x < 0 jj @cur.x > 999
212         @cur.x = (@cur.x < 0) ? 0 : 999
213         @org = @cur.dup
214         @range, @speed, @dspeed = 0, 0, 0
215         damage = COLL
216       end
217       if @cur.y < 0 jj @cur.y > 999
218         @cur.y = (@cur.y < 0) ? 0 : 999
219         @org = @cur.dup
220         @range, @speed, @dspeed = 0, 0, 0
221         damage = COLL
222       end
223     end
224     damage
225   end
226
227 end
228
229 class Robot
230   include DRbUndumped
231
232   CANHEAT = 20
233   CANCOOL = ¡1
234   MISMAX = 700
235   MSP = 100
236   CLIP = 4
237   MRELOAD = ((MISMAX / MSP) + 0.5).round
238   LRELOAD = MRELOAD ¤ 3
239   ERRDIST = 10
240   SCANBAD = 35
241
242   attr_reader :alive
243   attr_reader :name
244   attr_reader :idnum
245   attr_reader :num
246   attr_reader :team
247
248   attr_accessor :cdamage
249
250   attr_reader :motor
251   attr_accessor :mstate
252   attr_reader :missile
253
254   def initialize(fld, ctrl, name, idx, quad)
255     @alive = true
256     @fld, @controller = fld, ctrl
257     @name, @idnum = name, idx
```

```
258       # @num = ("#fnameg#fidxg".hash % 65536).abs
259       @motor = Motor.new(quad, rand(360))
260       @cdamage = 0
261       @sig = [0, 0, 0]
262       @mstate, @reload, @mused = false, 0, 0
263       @missile = Missile.new
264       @btemp = 0
265       @team = 0
266       @num = "#fidxg␣¡␣#f@teamg"
267
268       p self if $DEBUG
269     end
270
271     def scanner(deg, res)
272       do_wait 2
273       return 0 unless @alive
274       return 0 if @fld.disabled?
275       return ¡1 if deg < 0 jj deg > 359
276       return ¡1 if res < 0 jj res > 10
277
278       @fld.do_scanner(@idnum, deg, res)
279     end
280
281     def changeteam(team)
282       @team = team
283       @num = "#f@idnumg␣¡␣#f@teamg"
284       return @num
285     end
286
287     def nlist
288           do_wait
289           return ¤@fld.numlist
290     end
291
292     def tscan
293           do_wait
294           return @fld.teamscan[@team.to_i]
295     end
296
297     def myteam
298           do_wait
299           return @team.to_i
300     end
301
302     def myname
303       @num = "#f@idnumg␣¡␣#f@teamg"
304       return @num
305     end
306
307     def dsp
308       do_wait
309       return ¤@sig
310     end
311
```

```
312   def cannon(deg, rng)
313     do_wait
314     p "cannon␣#f@aliveg,␣#fmstateg,␣#f@reloadg" if $DEBUG
315     return 0 if !@alive jj @mstate jj @reload != 0
316     return 0 if @fld.disabled?
317     return ¡1 if deg < 0 jj deg > 359 jj rng < 0 jj rng > MISMAX
318     @missile.fire @motor, deg, rng
319     @mstate = true
320
321     @btemp += CANHEAT
322     @mused += 1
323     if @mused == CLIP
324       @reload = LRELOAD
325       @mused = 0
326     else
327       @reload = MRELOAD
328     end
329     1
330   end
331
332   def drive(deg, spd)
333     do_wait
334     return 0 unless @alive
335     return 0 if @fld.disabled?
336     return ¡1 if deg < 0 jj deg > 359 jj spd < 0 jj spd > 100
337
338     @motor.drive deg, spd
339   end
340
341   def damage
342     return 100 unless @alive
343     do_wait
344     @cdamage
345   end
346
347   def speed
348     return 0 unless @alive
349     do_wait
350     @motor.speed
351   end
352
353   def loc
354     do_wait
355     @motor.pos
356   end
357
358   def tick
359     @fld.ticks
360   end
361
362   def heat
363     return @motor.overheat, @motor.heat
364   end
365
```

```
366   def kill(s)
367     Thread.start do
368       @controller.kill s
369     end
370   end
371
372   def start
373     @controller.start
374   end
375
376   def stop
377     @controller.stop
378   end
379
380   def pos
381     @motor.pos
382   end
383
384   def winpos
385     @motor.cur.towin
386   end
387
388   def winmpos
389     @missile.cur.towin
390   end
391
392   def update_missile
393     @missile.update MSP
394   end
395
396   def explode?
397     return false unless @missile.explode?
398     @mstate = false
399     @missile.explode
400     true
401   end
402
403   def burned?(other)
404     d = @motor.hypot(other.missile)
405     [[6, 25], [10, 12], [20, 7], [50, 3]].each do jdia, hitj
406       if d < dia
407         p hit
408         @cdamage += hit
409         return true
410       end
411     end
412     false
413   end
414
415   def check_motor
416     @motor.adjust
417     @motor.accelate
418     @motor.stabilize
419     dmg = @motor.do_drive
```

```
420     if dmg > 0
421       @cdamage += dmg
422       return true
423     end
424     false
425   end
426
427   def check_scanner
428     if @btemp > 0
429       @btemp += CANCOOL
430       @btemp = 0 if @btemp < 0
431     end
432   end
433
434   def check_cannon
435     @reload ¡= 1 if @reload != 0
436   end
437
438   def check_scan(rob, deg, res, near)
439     x, y = distance(rob)
440     d = (57.2958 ¤ atan2(y, x)).round
441     d += 360 if d < 0
442     d1 = (d ¡ deg + 360) % 360
443     d2 = (deg ¡ d + 360) % 360
444     f = (d1 < d2) ? d1 : d2
445     if f <= res
446       dist = hypot(x, y).round
447       if dist < near
448         derr = ERRDIST ¤ res
449         terr = ((res  > 0) ? 5 : 0) + rand(derr)
450         fud1 = (rand(2) == 1) ? '¡' : '+'
451         fud2 = (rand(2) == 1) ? '¡' : '+'
452         near = eval "#fdistg␣#ffud1g␣#fterrg␣#ffud2g␣#f@btempg"
453         near = 1 if near < 1
454         return true, near
455       end
456     end
457     return false, near
458   end
459
460   def set_sig(dsp, dmg, dteam)
461     if @btemp >= SCANBAD
462       @sig.fill 0
463       return false
464     else
465       @sig[0..2] = dsp, dmg, dteam
466     end
467     true
468   end
469
470   def moved?
471     @motor.dirty
472   end
473
```

```
474   def dead?
475     return false if @cdamage < 100
476     @alive, @cdamage = false, 100
477     true
478   end
479 
480 private
481   def do_wait(c = 1)
482     sleep 0.1 ¤ c
483   end
484 
485   def distance(other)
486     return @motor.cur ¡ other.motor.cur
487   end
488 
489 end
490 
491 class BattleField
492   include Observable
493 
494   MAX_ROBOTS = 8
495 
496   TICK = 500
497   QUADS = [[100, 100],
498            [500, 100],
499            [rand(800)+100, rand(800)+100],
500            [rand(800)+100, rand(800)+100],
501            [rand(800)+100, rand(800)+100],
502            [rand(800)+100, rand(800)+100],
503            [rand(800)+100, rand(800)+100],
504            [rand(800)+100, rand(800)+100]]
505 
506   attr_reader :running
507   attr_reader :ticks
508 
509   def initialize()
510     @canvas = Hash.new
511     @robs = Array.new(MAX_ROBOTS)
512     @ticks = 0
513     @mutex = Mutex.new
514     @quads = QUADS.dup
515     @update, @stop, @running = false, false, false
516     @q = Queue.new
517     @th = nil
518     @scanlist = [[],[],[],[],[],[],[],[]]
519   end
520 
521   def add_observer(observer)
522     @observer_peers = [] unless defined? @observer_peers
523     @observer_peers.push observer
524 
525     Thread.start do
526       sleep 0.1
527       (0...MAX_ROBOTS).each do |i|
```

```
528         next if @robs[i].nil?
529         observer.update ["newrobot", @robs[i].idnum, @robs[i].num]
530 
531       end
532     end
533     count_observers
534   end
535 
536   def teamscan
537         return @scanlist
538   end
539 
540   def change_team(robot, team)
541     name = @robs[robot].changeteam(team)
542 
543     Thread.start do
544       sleep 0.1
545       @observer_peers.each do jij
546         i.update ["team", robot, name]
547       end
548     end
549 
550   end
551 
552   def notify_observers(¤arg)
553     if @th.nil?
554       @update = true
555       @th = Thread.new do
556         while @update
557           ag = @q.pop
558           changed
559           a = [ag]
560           until @q.empty? jj a.size > 12
561             a << @q.pop
562           end
563           super ¤a
564         end
565       end
566     end
567     @q.push arg
568   end
569 
570   def notify_otherobservers(ob, ¤arg)
571     changed
572     if defined? @observer_state and @observer_state
573       if defined? @observer_peers
574         @observer_peers.each do jij
575           next if ob == i
576           i.update arg
577         end
578       end
579     end
580   end
581 
```

```
582   def add_robot(ctrl, name)
583     robo = nil
584     @mutex.synchronize do
585       (0...MAX_ROBOTS).each do |idx|
586         if @robs[idx].nil?
587                 if @quads.size == 1
588                         n = 0
589                 else
590                         n = rand(@quads.size)
591                 end
592                 robo = Robot.new(self, ctrl, name, idx, @quads[idx])
593                 # @quads.delete_at n
594                 @robs[idx] = robo
595                 break
596         end
597       end
598     end
599     unless robo.nil?
600
601       show_newrobot robo
602
603     end
604     robo
605   end
606
607
608
609   def do_scanner(robot, deg, res)
610     rob = @robs[robot]
611     return 0 if rob.nil?
612     show_scan rob, deg, res
613     dsp, dmg, dteam, near = 0, 0, 0, 9999
614     @robs.each do |robx|
615       next if (robx.nil? || robx == rob || !robx.alive)
616       t, near = robx.check_scan(rob, deg, res, near)
617       if t
618         dsp = robx.num
619         dmg = robx.cdamage
620         dteam = robx.team.to_i
621         iteam = rob.team.to_i
622         if dteam != iteam
623                 @scanlist[iteam] = [[near, deg, res, rob.pos], @ticks, rob.num]
624         end
625       end
626     end
627
628
629     if rob.set_sig(dsp, dmg, dteam)
630       return near unless (near == 9999)
631     end
632     0
633   end
634
635   def disabled?
```

```
636       @stop jj @running == false
637     end
638
639     def do_start(cv)
640       return false if @robs.nitems <= 1
641       notify_otherobservers cv, "start"
642       @running = true
643       Thread.start do
644         @robs.each do jrj
645           next if r.nil?
646           r.start
647         end
648         while @running && @stop == false
649           update_robots
650           sleep TICK / 1000.0
651         end
652       end
653       true
654     end
655
656     def do_stop(cv)
657       @running = false
658       notify_otherobservers cv, "stop"
659       true
660     end
661
662     def do_shutdown(cv)
663       @update = false
664       notify_otherobservers cv, "close"
665       Thread.start do
666         @robs.each do jrj
667           next if r.nil?
668           r.kill "user␣shutdown"
669         end
670       end
671       true
672     end
673
674     def do_initNew(cv)
675       return if @stop == false
676       notify_otherobservers cv, "initNew"
677       (0...MAX_ROBOTS).each do jij
678         next if @robs[i].nil?
679         r = @robs[i]
680         @robs[i] = nil
681         r.kill "user␣break␣at␣tick:#f@ticksg"
682       end
683       @running, @stop = false, false
684       @ticks, @quads = 0, QUADS.dup
685     end
686
687     def update_robots
688       @ticks += 1
689       num_miss = 0
```

```
690     num_rob = 0
691     num_team = []
692     @robs.each do |r|
693       next if r.nil?
694       if r.mstate
695         num_miss += 1
696         r.update_missile
697         if r.explode?
698                 show_explode r
699                 @robs.each do |rr|
700                         next if rr.nil? || rr.alive == false
701                         if rr.burned?(r)
702                                 up_damage rr
703                         end
704                 end
705         end
706       end
707       next if r.alive == false
708       r.check_cannon
709       r.check_scanner
710       if r.check_motor
711         up_damage r
712       end
713       num_team.unshift(r.team)
714     end
715     @robs.each do |r|
716       next if r.nil? || r.alive == false
717       if r.dead?
718         up_damage r
719         disable_robot r
720       else
721         num_rob += 1
722       end
723     end
724     if num_team.uniq.size <= 1
725       @stop = true
726       (0...MAX_ROBOTS).each do |i|
727         next if @robs[i].nil?
728         r = @robs[i]
729         @robs[i] = nil
730         notify_observers "win", i, r.cdamage
731         r.kill "Winner! at tick:#{@ticks}"
732       end
733     end
734     show_robots
735   end
736
737
738
739 private
740   def disable_robot(r)
741     unless @robs[r.idnum].nil?
742       @robs[r.idnum] = nil
743       r.kill "dead at tick:#{@ticks}"
```

```
744       end
745     end
746
747     def show_scan(r, deg, res)
748       x, y = r.pos
749       d0 = deg ¡ res / 2
750       d0 += 360 if d0 < 0
751       d1 = deg + res / 2
752       d1 ¡= 360 if d1 >= 360
753       x1 = (C_TAB[d0] ¤ 700 + x).round
754       x2 = (C_TAB[d1] ¤ 700 + x).round
755       y1 = 1000 ¡ (S_TAB[d0] ¤ 700 + y).round
756       y2 = 1000 ¡ (S_TAB[d1] ¤ 700 + y).round
757       y = 1000 ¡ y
758       notify_observers "scan", r.idnum, x, y, x1, y1, x2, y2
759     end
760
761     def show_explode(r)
762       p "explode␣#frg" if $DEBUG
763       notify_observers "explode", r.idnum, ¤r.winmpos
764     end
765
766     def show_robots
767       @robs.each do jrj
768         next if r.nil?
769         if r.alive && r.moved?
770           notify_observers "robot", r.idnum, ¤r.winpos
771         end
772         if r.mstate
773           notify_observers "missile", r.idnum, ¤r.winmpos
774         end
775       end
776     end
777
778     def up_damage(r)
779       if r.cdamage >= 100
780         txt = "dead"
781       else
782         txt = "#fr.cdamageg%"
783       end
784       p "#frg␣=␣#ftxtg" if $DEBUG
785       notify_observers "damage", r.idnum, txt, r.cdamage
786     end
787
788     def show_newrobot(r)
789       notify_observers "newrobot", r.idnum, r.num
790     end
791
792 end
793
794 class RobotOffice
795     include DRbUndumped
796     def initialize(field)
797       @fld = field
```

```
798   end
799
800   def join(ctrl, name)
801     return nil if @fld.running
802     r = @fld.add_robot ctrl, name
803     p "join new Robots:#fnameg #f(r.nil?)?'Rejected':''g" if $DEBUG
804     r
805   end
806
807   if __FILE__ == $0
808     def start(cv)
809       @fld.do_start cv
810     end
811
812     def stop(cv)
813       @fld.do_stop cv
814     end
815
816     def initNew(cv)
817       @fld.do_initNew cv
818     end
819
820     def shutdown(cv)
821       @fld.do_shutdown cv
822       Thread.start do
823         puts "waiting shutdown..."
824         sleep 5
825         DRb.stop_service
826       end
827       nil
828     end
829
830     def add_canvas(canvas)
831       @fld.add_observer canvas
832     end
833
834     def del_canvas(canvas)
835       @fld.delete_observer canvas
836     end
837
838     def cha_team(robot, team)
839       @fld.change_team(robot, team)
840     end
841
842   end
843
844 end
845
846 if __FILE__ == $0
847   field = BattleField.new
848   DRb.start_service('druby://:6852', RobotOffice.new(field))
849   puts DRb.uri
850   DRb.thread.join
851 end
```

相互通信 Charger プログラム

List 9.2: 相互通信する Charger

```
class ClientCharger < Robot

  include Math

  def zatack(x2, y2)
    x1, y1 = loc
    x = x2 ¡ x1
    y = y2 ¡ y1
    if x >= 0
      if y >= 0
        d = (57.2958 ¤ atan2(y.abs, x.abs))
        else
        d = (57.2958 ¤ atan2(x.abs, y.abs)) + 270
      end
    else
      if y >= 0
        d = (57.2958 ¤ atan2(x.abs, y.abs)) + 90
      else
        d = (57.2958 ¤ atan2(y.abs, x.abs)) + 180
      end
    end
    r = sqrt(x¤¤2 + y¤¤2)
    return d.round, r.round
  end

  def calc(t)
    r = t[0][0]
    d = t[0][1]
    x1, y1 = t[0][3]
    x = r ¤ cos(d/(180/PI)) + x1
    y = r ¤ sin(d/(180/PI)) + y1
    return x.round, y.round
  end

  def run

    t = 0
    dir = rand 360
    res = 10
    limit = 9999

    DRb.start_service
    uri = "druby://PINOKIO:8470"
    data = DRbObject.new(nil, uri)

    while alive do
      serch = scanner dir, res

      x, y = loc
      time = tick

```

```
52        if (serch != 0) && (myteam != dsp[2])
53          data.updata(serch, dir, res, x, y, time)
54        end
55
56        tsc = data.popdata
57
58        p tsc
59
60        if tsc != []
61          t1 = tsc[1]
62          if t1 > t
63            x, y = calc(tsc)
64            d, r = zatack(x, y)
65            if tsc[0][2] == 1
66              cannon d, r
67              drive d, 70
68              p "atack!"
69            else
70              res = 1
71              limit = 10
72            end
73            dir = d + 5
74            t = t1
75          end
76        end
77
78        if res == 10
79          dir = (dir ¡ 10 + 360) % 360
80        else
81          dir = (dir ¡ 1 + 360) % 360
82        end
83
84        limit ¡= 1
85        if limit < 0
86          res = 10
87
88        end
89      end
90    end
91
92  end
```

渕研文献購読優勝ロボットプログラム

List 9.3: **渕研ロボット**

```
require 'robot'

class Charger < Robot

  def initialize
    @cur_dir = 0
    super
  end

  def keep_moving
    x, y = loc

    bef = @cur_dir
    case @cur_dir
    when 0
      @cur_dir = 270 if x > 900
    when 90
      @cur_dir = 0 if y > 900
    when 180
      @cur_dir = 90 if x < 100
    when 270
      @cur_dir = 180 if y < 100
    end

    p "#fbefg␣to␣#f@cur_dirg" if bef != @cur_dir

    drive @cur_dir, 100
  end

  def run
    Thread.start do
      while alive
        keep_moving
        sleep 1
      end
    end

    dir = rand 360

    while alive do
      rng = scanner dir, 10
      if (rng > 0 && rng < 700) && (myteam != dsp[2])
        start = (dir ¡ 20) % 360
        limit = ¡1
        while alive && limit >= ¡20
          dir = (start ¡ limit + 360) % 360
          rng = scanner dir, 3
          if (rng > 0 && rng <= 700) && (myteam != dsp[2])
            cannon dir, rng
            limit += 8
            next
```

```
52            end
53            limit ¡= 3
54          end
55        end
56
57
58        p "dir␣=␣#fdirg,␣@cur_dir␣=␣#f@cur_dirg"
59
60
61        dir = (dir + 20 + 360) % 360
62      end
63    end
64
65 end
```