卒業研究報告書

題目

シューティングゲームにおける弾道予測アルゴリズムの作成

指導教員

石水 隆　講師

報告者

13-1-037-0136

筒井 晴也

近畿大学理工学部情報学科

平成 29 年 1 月 31 日提出

概要

シューティングゲームとは，自分が自由に操作出来る自機を，様々な方法で撃墜しようとする敵機の攻撃を躱し，自機から発射される弾を敵機に当てて撃墜することでスコアを記録するコンピュータゲームである．

現在，様々なシューティングゲームが存在し，それらのクリアの定義は様々である．スコアを多く稼ぐもの，ある一定の距離を進んだ先に存在するボスキャラクターを倒すもの，無限に出現する敵機の攻撃を避け続けるものなど様々だ．しかし，どのようなクリア目標を掲げているものでも「自機に敵機や敵弾を当てないようにする」という点においてはあらゆるゲームを通して変わらない．

シューティングゲームの難易度は自機の性能，ゲームスピード，敵の攻撃の激しさ，弾の当たり判定の大きさ等様々な要因に依存し，適切な難易度にするためには多くのプレイヤーによるテストプレイが必要となる．そこで本研究では，シューティングゲームにおいて敵機及び敵弾の位置情報を把握し，それにあわせて自機が自動で避ける AI を開発し，その有用性を検証する．この AI を用いることで，ゲームのテストプレイ・デモンストレーションを簡単に行うことが出来る，自機が撃墜されるまでの時間を計測することでゲームの難易度の指標を立てることが出来るなどといったメリットを得られる．

本研究では，Java を用いてシューティングゲームおよびその自動操縦 AI を作成する．本研究で作成する自動操縦 AI は，自機の周りを探索し，一定距離以内に敵弾があればそれから避ける方向へ回避行動を取る．

目次

1　序論

1.1本研究の背景

シューティングゲームとは，自分が自由に操作出来る自機を，様々な方法で撃墜しようとする敵機の攻撃を躱し，自機から発射される弾を敵機に当てて撃墜することでスコアを記録するコンピュータゲームである．画面がスクロールしないタイプの固定画面シューティングゲーム「スペースウォー！」「スペースインベーダー」，画面が上から下へとスクロールする縦スクロールシューティング「ゼビウス」「スターフォース」，画面が右から左へとスクロールする横スクロールシューティング「グラディウス」「R-TYPE」，大量の弾幕を小さな当たり判定の自機ですり抜ける弾幕シューティング「怒首領蜂」「東方 project」と現在に至るまで様々なシューティングゲームが登場している．

それらのゲームのクリアの定義は様々である．敵機を撃墜しスコアを多く稼ぐもの，ある一定の距離を進んだ先に存在するボスキャラクターを倒すもの，無限に出現する敵機の攻撃を避け続けるものなど様々だ．しかし，どのようなクリア目標を掲げているものでも「自機に敵機や敵弾を当てないようにする」という点においてはあらゆるゲームを通して変わらない．

1.2本研究の目的

シューティングゲームの難易度は自機の性能，ゲームスピード，敵の攻撃の激しさ，弾の当たり判定の大きさ等様々な要因に依存し，適切な難易度にするためには多くのプレイヤーによるテストプレイが必要となる．テストプレイは多くの費用と時間がかかるため，シューティングゲーム開発のネックとなる．そこで本研究ではシューティングゲームにおいて敵機及び敵弾の位置情報を把握し，それにあわせて自機が自動で避ける AI を開発し，その有用性を検証する．この AI を用いることで，ゲームのテストプレイ・デモンストレーションを簡単に行うことが出来る，自機が撃墜されるまでの時間を計測することでゲームの難易度の指標を立てることが出来るなどといったメリットを得られる．

1.3シューティングゲームに関する既知の結果

シューティングゲームの先行研究としては，経路探索を用いて人間らしい弾避けをする AI[3]や，弾幕の認識に人間の視覚特性を取り入れた AI[4]などが存在し，これらは Influence Map を用いている．Infuence Map とは，画面をグリッド状に分割し，ある要素はその各グリッドへどのような影響を与えるかを指し示すものである．シューティングゲームにおいては，各グリッドにおける被弾危険度を表すために使われている．

また自機 AI ではなく，敵弾発射のアルゴリズムに関する考察し難易度を向上させる研究[5]も存在する．

1.4本報告書の構成

本報告書の構成は以下の通りである．まず 2 節はシューティングゲームの AI 作成に関して本研究で用いた手法，研究に際し作成したゲームの内容，AI のアルゴリズムについて説明する．3 章では本研究で作成した自動操縦 AI プログラムの説明を述べる．4 章では結果及び考察を述べ，5 章では本研究の結論を述べる．以降は謝辞と参考文献を示し，付録に作成したプログラムのソースコードを載せる．

2. 研究内容

シューティングゲームにおける自動操縦の有用性を検証するため，本研究では Java を用いてシューティングゲームとその自動操縦 AI を作成する．

2.1シューティングゲームとは

本節ではシューティングゲームとはどのようなゲームであるか述べる.

シューティングゲームとは，自分が自由に操作出来る自機を，様々な方法で撃墜しようとする敵機の攻撃を躱し，自機から発射される弾を敵機に当てて撃墜することでスコアを記録するものである.シューティングゲームといってもその体系は様々であり,『スペースインベーダー』『グラディウス』といったものは x, y 軸の動きのみでしかプレイできないが,『スターフォックス』『エースコンバット』といったものは x, y 軸だけでなく z 軸にも動ける，所謂三次元を自由に移動してプレイすることが可能である.

2.2研究方法

本研究では Java を用いてシューティングゲームとその自動操縦 AI を作成する．作成するシューティングゲームは固定画面シューティングゲームで，敵機及び敵弾をひたすら避け続けるものとする.

今回は自機が何フレームで撃墜されるかを 5 段階の難易度にわけて評価する．難易度が上がるにつれ敵機が発射する弾の数が増え，敵の出現間隔が短くなる.

2.3 ゲームの内容

以下に今回作成したゲームがどのようなものか，具体的にはゲーム画面の大きさ・敵のタイプ・自機，当たり判定の情報について簡単に説明する.

- **画面の大きさ：**横 500px，縦 500px の正方形のキャンバスである.
- **敵のタイプ**:多くのゲームでは複数の種類の敵機が出現し,撃ってくる弾の種類も敵毎に異なる.そこで本研究では 2 種類の敵機を作成した.
  - **敵 1**：ゆらゆら動きながら 50 フレーム毎に 6 方向に弾を発射する.
  - **敵 2**：出現し、200px 進んだところで静止し，30 フレーム毎にレベルに応じた回数自機に向けた 5way 弾を発射する.

  いずれも自機の弾が 1 発でも当たれば消滅する.
- **自機**：常時弾を 5 発撃ち続ける．敵弾が当たる，もしくは敵機に衝突すると消滅し，ゲームオーバーとなる.
- **当たり判定**:自機と敵性オブジェクトの距離が 8px 未満になる,または自弾と敵機の距離が 16px 未満になると衝突扱いとなる.

2.4 アルゴリズム

今回作成したアルゴリズムの詳細を示す.

まず自機の周囲，上下左右 100px を探索する．敵性オブジェクトがその範囲内に入ってきたら，それぞれのオブジェクトに合わせた動きをする.

自機は以下のアルゴリズムに従って動く.

- **敵弾が自機から一定距離以内にある場合**

敵弾の経路に対して法線ベクトル方向，すなわち敵弾の進む角度に対して直角になる方向へ自機を動かす．一定距離以内に敵弾が複数ある場合は，一番近い敵弾の動きから自機の動きを決定する.

- **敵機が自機から一定距離以内にある場合**

敵機の進む動きに対し，垂直方向へ移動する.

- **敵弾・敵機が自機から一定距離以内にない場合**

探索を行なわず，ゲーム開始地点に戻る.

上記のように自動操縦 AI は自機から一定距離に敵性オブジェクトがある場合のみ回避行動を取る.これは自機の被弾に関係ない遠い敵性オブジェクトを無視し，探索範囲を絞ることにより計算量を減

らすためである．

以下に敵性オブジェクトが近づいてきた場合の処理の詳細を述べる．

- **敵弾が近づいてきた場合**

自機の動きは(bulletXFlag * Xmove, bulletYFlag * Ymove)で計算する．
敵弾が自機に対して 90°～270°の角度で進んでいる場合，bulletXFlag を 1 に設定する．それ以外の場合は-1 とする．

次に敵弾の侵入角度を度からラジアンへ変換する．変換したものから自機と敵弾の位置を算出する．算出したものを ArrayList に格納し比較を行う．そして最小の数値を持つもの，すなわち自機と一番近いものの番号を取り出し，その番号を持つ弾の角度情報を取り出し再びラジアンへ変換する．この変換したラジアンの X 座標を Xmove とし，Y 座標を Ymove とする．自機は向かってくる弾に対し，法線ベクトル上に避ければその弾には当たらない．よって Xmove と Ymove の値を入れ替える．

自機が端に寄りすぎると自由な動きを損なうと仮定する．
そこでまずは X の動きについて考える．

- 敵弾が右側に存在し，且つ自機が画面左端から 50px 未満になった状態
- 敵弾が左側に存在し，且つ自機の画面右側に余裕がある状態
- 敵弾が真上に存在し，且つ自機が画面左端から 50px 未満になった状態

これら 3 つの場合 bulletXFlag を 1 とし，

- 敵弾が右側に存在し，且つ自機の画面左側に余裕がある状態
- 敵弾が左側に存在し，且つ自機が画面右端から 50px 未満になった状態
- 敵弾が真上に存在し，且つ自機が画面右端から 50px 未満になった状態

これら 3 つの場合 bulletXFlag を-1 とする．
それ以外の場合 bulletXFlag を 1，-1 をランダムに選択する．

次に Y の動きについて考える．

- 敵弾が上側に存在し，且つ自機が画面下限から 50px 未満になった状態
- 敵弾が下側に存在し，且つ自機の上側に余裕がある状態
- 敵弾が真横に存在し，且つ自機が画面下限から 50px 未満になった状態

これら 3 つの場合 bulletYFlag を 1 とする．

- 敵弾が上側に存在し，且つ自機が画面下限から 50px 未満になった状態
- 敵弾が下側に存在し，且つ自機の上側に余裕がある状態
- 敵弾が真横に存在し，且つ自機が画面下限から 50px 未満になった状態

これら３つの場合 bulletYFlag を-1 とする．
それ以外の場合 bulletYFlag を 1，-1 をランダムに選択する．

- **敵機が近づいてきた場合**

自機の動きは(enemyXFlag，enemyYFlag)とする．敵弾が近づいてきた時同様，自機が端に寄りすぎると自由な動きを損なうと仮定し，まず X の動きについて考える．

- 敵機が右側に存在し，且つ自機が画面左端から 50px 未満になった状態
- 敵機が左側に存在し，且つ自機の画面右側に余裕がある状態
- 敵機が真上に存在し，且つ自機が画面左端から 50px 未満になった状態

これら 3 つの場合 enemyXFlag を 1 とし，

- 敵機が右側に存在し，且つ自機の画面左側に余裕がある状態
- 敵機が左側に存在し，且つ自機が画面右端から 50px 未満になった状態
- 敵機が真上に存在し，且つ自機が画面右端から 50px 未満になった状態

これら 3 つの場合 enemyXFlag を-1 とする．
それ以外の場合 enemyXFlag を 0 とする．

次に Y の動きについて考える.

- 敵弾が上側に存在し，且つ自機が画面下限から 50px 未満になった状態
- 敵弾が下側に存在し，且つ自機の上側に余裕がある状態
- 敵弾が真横に存在し，且つ自機が画面下限から 50px 未満になった状態

これら 3 つの場合 enemyYFlag を 1 とする.

- 敵弾が上側に存在し，且つ自機が画面下限から 50px 未満になった状態
- 敵弾が下側に存在し，且つ自機の上側に余裕がある状態
- 敵弾が真横に存在し，且つ自機が画面下限から 50px 未満になった状態

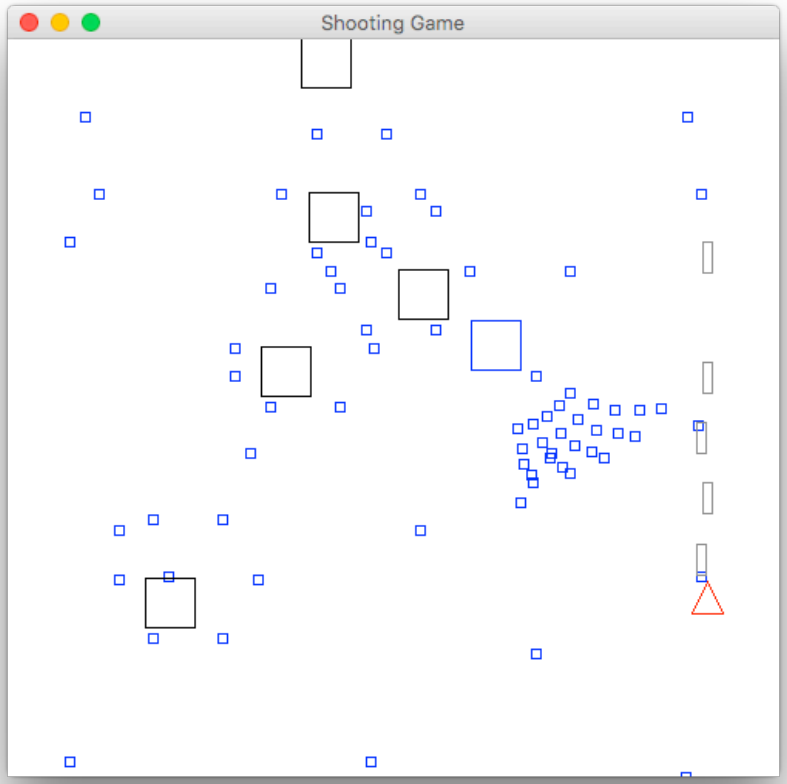
これら 3 つの場合 enemyYFlag を-1 とする.
それ以外の場合 enemyYFlag を 0 とする.

3. 自動操縦 AI プログラム

本章では,本研究で作成したシューティングゲームの自動操縦 AI プログラムについて説明する.
付録に本研究で作成した自動操縦 AI プログラムのソースを示す。また，図 1 に本研究で作成したシューティングゲームの実行の様子を示す.

- GameObject.java
  Player.java の抽象クラス.
  自機・敵機・敵弾・自機弾のゲームオブジェクト位置情報を表す x,y と画面に出現しているかをを表す active をフィールドに持ち，それぞれの動きを持つ move()メソッド，それぞれを描画する draw(Graphics g)メソッドを持つ.
- Player.java
  自機の速さ・敵機・敵弾・自機の探索範囲・敵弾のラジアン・自機の動きを表す double 型，敵機や敵弾，敵弾の距離を格納する ArrayList，ゲーム時間を図る int 型，探索範囲に敵性オブジェクトがいる場合かどうかを判別する boolean 型，各種クラス変数をフィールドに持つ.
  コンストラクタには，自機の位置情報・自機の速度・探索範囲に敵性オブジェクトがあるかどうかの判定，描画の有無のフラグを持つ.
  - move(double mx, double my)：自機がゲーム画面外に移動できないようにするメソッド
  - move()：敵性オブジェクトをサーチし，補足した場合はそれに対応した自機の動きを行う.
  - searchEnemy(double ex, double ey)：敵機を探索する.
  - searchBullet(double bx, double by)：敵弾を探索する.
  - returnMove()：自機を初期位置へ戻るように動かす.
  - compare()：自機に最も近い敵弾を判別する.
  - draw(graphics g)：自機の描画を行う.

図 1：ゲーム画面



4. 結果・考察

本章では，本研究で作成した自動操縦 AI の性能について述べる．

AI の性能評価は平均生存時間と安定率で行う．平均生存時間はゲームをスタートしてゲームオーバーになるまでにかかった時間である．安定率は，ゲームオーバーになるまでの時間が 200 フレームを超える確率である．

作成した AI の有用性を検証するため，5 段階の難易度をそれぞれ 50 回ずつプレイした．表 1 に実行結果を示す．

表 1：AI のシューティングゲームプレイ結果

難易度	平均生存時間(フレーム)	安定率(%)
1	490	88
2	413	82
3	322	78
4	296	66
5	210	50

表 1 より，難易度が低い場合は平均生存時間が長く，安定率も高いことから，本研究で作成した AI が敵性オブジェクトの位置情報を把握し，回避行動を取っていることが示される．しかし，難易度が上がるにつれ大幅に平均生存時間・安定率が下がっており，AI の敵性オブジェクト回避能力には限界

があることが示される.

また自機がどの敵弾によって撃墜されたかを表 2 に示す.

表 2：難易度別各種敵弾の自機撃破率

難易度	6way 弾(%)	自機狙い 5way 弾(%)
1	22	78
2	20	80
3	20	80
4	16	84
5	8	92

表 2 より, 自機の撃破率は自機狙い 5way 弾の方が高いことから, 作成した AI が自機狙い 5way 弾に対して脆弱性が極めて高いが把握できた.

また，目視による観測の結果，以下の撃墜パターンがあることがわかった.

- 敵弾を避けても避けた先に別の敵弾があるパターン
- 自ら敵弾に当たりに行くパターン
- 自機狙い 5way 弾の間を通り抜けられないパターン
- 画面端まで追い詰められ満足に動けないパターン

それぞれの撃墜パターンによる考察を行う．まず『敵弾を避けても避けた先に別の敵弾があるパターン』だが，一番近くの敵弾 1 つのみから自機の動きを決定していることによるものだと推測出来る．改善策として挙げられるのは，まず自機中心とした探索範囲を左上・左下・右上・右下の 4 つに分割する．その分けた探索範囲に存在する敵弾の数を把握する．そして，その 4 つに分けた探索範囲の中で最も敵弾が少ない方向へ x, y の動きを算出するといった方法である.

次に『自ら敵弾に当たりに行くパターン』だが，これは自機に対して 0 度，45 度に向かってくる敵弾に対してランダムに x, y の動きを決めた結果によるものだと考えられる．これは敵弾の動く角度によって自機の動きに変化をつければよい.

『自機狙い 5way 弾の間を通り抜けられないパターン』は，敵弾を避けても避けた先に別の敵弾があるパターンと同じ理屈によるものと予想される．あるいは，敵弾が発射された直後で弾の間が十分に開ききってない故に避ける先がないというのも考えられる．改善策としては敵弾を避ける際に一度後退してから U 字を描くように動けば被弾し辛くなるのではないかと考えられる.

最後に『画面端まで追い詰められ満足に動けないパターン』だが，これは画面端まで行かないようにアルゴリズムを設定しているので，その制約を取り除けば改善できるものと推測する.

5. 結論

本研究ではシューティングゲームにおいて敵機及び敵弾の位置情報を把握し，それにあわせて自機が自動で避ける AI を開発した．本研究で作成した AI は敵性オブジェクトを自動で避けることに成功した．しかし，難易度が高くなるに連れ著しく生存率が低くなるのが難点であるので，改良の余地は多々ある.

今後の課題としては，探索範囲にある敵弾全ての弾道予測を行い，線ではなく面での予測を行う，画面の端に追いやられた場合の特殊なアルゴリズムを作成するなどが挙げられる.

謝辞

本論文を作成するにあたり石水 隆講師から、丁寧かつ熱心なご指導を賜りました。ここに感謝の意を表します。

参考文献

[1] 松浦健一郎,司ゆき, シューティングゲームプログラミング,SoftBank Creative, 2006

[2] 長久勝,Java ゲームプログラミング, SoftBank Creative, 2004

[3] 佐藤直之, Sila Temsiririrkkul, Luong Huu Phuc, 池田心, Influence Map を用いた経路探索による人間らしい弾避けのシューティングゲーム AI プレイヤ, ゲームプログラミングワークショップ 2016 論文集, Vol.2016, pp.57-64, 情報処理学会, (2016), http://id.nii.ac.jp/1001/00175304/

[4] 平井弘一, Reijer Grimbergen, 弾幕の認識に人間の視覚特性を取り入れたシューティングゲーム AI の研究, ゲームプログラミングワークショップ 2016 論文集, Vol.2016, pp.158-161, 情報処理学会, (2016), http://id.nii.ac.jp/1001/00175321/

[5] 川野洋, シューティングゲームの敵機攻撃弾発射アルゴリズムに関する考察, 情報処理学会研究報告ゲーム情報学(GI), Vol.2006-GI-016, No.70, pp.61-68, 情報処理学会, (2006), http://id.nii.ac.jp/1001/00058512/

[6] 片岡俊, 遺伝アルゴリズムを用いたシューティングゲームの攻撃弾発射手法, 高知工科大学情報システム工学科平成 18 年度学士学位論文, (2006), http://www.kochi-tech.ac.jp/library/ron/2006/2006info/1070381.pdf

[7] Java でシューティング, 株式会社アイプランニング, (2007),https://www.ipl.co.jp/item/JavaShootingGame.html

付録

本研究で作成したシューティングゲーム AI のソースプログラムを以下に示す.

- GameObject.java

```
import java.awt.*;


public abstract class GameObject {
//敵弾・自機弾・自機・敵弾の情報をまとめる抽象クラス
	public boolean active;


	public double x;
	public double y;


	abstract void move();


	abstract void draw(Graphics g);
}
```

- Player.java

```
import java.awt.*;
import java.util.ArrayList;
import java.util.Random;


public class Player extends GameObject{


  double speed;     //自機の移動速度
  Enemy  enemy;
  Bullet bullet;
  MyCanvas myCanvas;
  int time;                 //ゲーム内時間
  public boolean searchReactE, searchReactB;
  double ex,ey, bx,by, Lpx,Upy,Rpx,Dpy, dx,dy, Xdiff,Ydiff, distance, Xmove,Ymove;
//ex,ey は敵機の位置、bx,by は敵弾の位置、Lpx,Upy,Rpx,Dpy はそれぞれ自機の左上右下の 4 方向の距離
  ArrayList<Double>  enemyXData = new ArrayList<Double>();              //敵機の x 座標の位置を格納する List
  ArrayList<Double>  enemyYData = new ArrayList<Double>();              //敵機の y 座標の位置を格納する List
  ArrayList<Double> bulletXData = new ArrayList<Double>();              //敵弾の x 座標の位置を格納する List
  ArrayList<Double> bulletYData = new ArrayList<Double>();              //敵弾の y 座標の位置を格納する List
  ArrayList<Double> bulletDirectionData = new ArrayList<Double>();      //敵弾と自機の距離を格納する List
```

```
 Player(double ix, double iy, double ispeed) {
        x = ix;
        y = iy;
        speed = ispeed;
        active = false;
        //最初は探索をfalseに設定しておく
        searchReactE = false;
        searchReactB = false;

        }

 public void move(double mx, double my) {
        //Canvasの外には移動できないようにする
        double postX = x + mx * speed;
        double postY = y + my * speed;

        if ((0 < postX)&&(postX < 500)) {
                x = postX;
        }
        if ((0 < postY)&&(postY < 480)) {
                y = postY;
        }

}

 public void move(){
        //敵性オブジェクトをサーチ
        for(int i = 0; i < ObjectPool.enemy.length; i++) {
        if((ObjectPool.enemy[i].active) == true){
                searchEnemy(ObjectPool.enemy[i].x, ObjectPool.enemy[i].y);
        }
        if((ObjectPool.bullet[i].active) == true){
                searchBullet(ObjectPool.bullet[i].x, ObjectPool.bullet[i].y, ObjectPool.bullet[i].direction);
        }
  }
```

```
//敵弾を補足した場合
if(searchReactB == true){

    ArrayList<Double> dist = new ArrayList<Double>();
    //初期化
    double bulletXFlag = 0;
    double bulletYFlag = 0;

    for(int i = 0; i < bulletDirectionData.size(); i++) {

        if (90< bulletDirectionData.get(i) && bulletDirectionData.get(i) <270){  //90度～270度の場合
            bulletXFlag = 1;
        } else {
            bulletXFlag = -1;
        }

        double radian = Math.toRadians(bulletDirectionData.get(i));    //度をラジアンに変換
        Xdiff =  Math.cos(radian);
        Ydiff =  Math.sin(radian);
        dist.add(Math.sqrt(Math.pow(Xdiff,2) + Math.pow(Ydiff,2)));
        int minDist = compare(dist);

        double radian2 = Math.toRadians(bulletDirectionData.get(minDist));

        //度をラジアンに変換
        Xmove =  Math.cos(radian2);
        Ymove =  Math.sin(radian2);

        //数値を入れ替え
        double dtmp = Xmove;
        Xmove = Ymove;
        Ymove = dtmp;

        for(int j = 0; j < bulletXData.size(); j++) {

            if(bulletXData.get(j) - x > 0 && x < 50) {          //敵が右にいて、ギリギリまで左に寄ってる時
                bulletXFlag =  1;
            } else if (bulletXData.get(j) - x > 0  && x >= 50){  //敵が右にいて、左に余裕がある時
                bulletXFlag = -1;
```

```
            } else if (bulletXData.get(j) - x < 0  && x > 450){  //敵が左にいて、ギリギリまで右に寄ってる時
                        bulletXFlag = -1;
            } else if (bulletXData.get(j) - x < 0 && x <= 450){  //敵が左にいて、右に余裕がある時
                        bulletXFlag =  1;
            } else if (bulletXData.get(j) - x == 0 && x <  50){  //敵が真上にいて、ギリギリまで左に寄ってる時
                        bulletXFlag =  1;
            } else if (bulletXData.get(j) - x == 0 && x > 450){  //敵が真上にいて、ギリギリまで右に寄ってる時
                        bulletXFlag = -1;
            } else  {
                        Random rnd = new Random();
          int ran = rnd.nextInt(2);
          switch(ran) {
                        case 0:
                                    bulletXFlag = -1;
                                    break;
                        case 1:
                                    bulletXFlag = 1;
                                    break;
                        default:
                                    }
            }
}
for(int k = 0; k < bulletYData.size(); k++) {
            if(bulletYData.get(k) - y > 0 && y < 50) {           //敵が右にいて、ギリギリまで左に寄ってる時
                        bulletYFlag = -1;
            } else if (bulletYData.get(k) - y > 0  && y >= 50){  //敵が右にいて、左に余裕がある時
                        bulletYFlag =  1;
            } else if (bulletYData.get(k) - y < 0  && y > 450){  //敵が左にいて、ギリギリまで右に寄ってる時
                        bulletYFlag =  1;
            } else if (bulletYData.get(k) - y < 0 && y <= 450){  //敵が左にいて、右に余裕がある時
                        bulletYFlag = -1;
            } else if (bulletYData.get(k) - y == 0 && y <  50){  //敵が真上にいて、ギリギリまで左に寄ってる時
                        bulletYFlag =  1;
            } else if (bulletYData.get(k) - y == 0 && y > 450){  //敵が真上にいて、ギリギリまで右に寄ってる時
                        bulletYFlag = -1;
            } else  {
                        Random rnd = new Random();
          int ran = rnd.nextInt(2);
          switch(ran) {
```

```
                                        case 0:
                                                bulletYFlag = -1;
                                                break;
                                        case 1:
                                                bulletYFlag = 1;
                                                break;
                                        default:
                                                }
                                }
                        }
        }

        move(bulletXFlag * Xmove, bulletYFlag * Ymove);

        dist.clear();
        searchReactB =false;
} else {
        returnMove();
}

        //敵機を捕捉した場合
        if(searchReactE == true){
                int enemyXFlag = 0;
                int enemyYFlag = 0;
                for(int i = 0; i < enemyXData.size(); i++) {
                        if(enemyXData.get(i) - x > 0 && x < 50) {          //敵が右にいて、ギリギリまで左に寄ってる時
                                enemyXFlag =  1;
                        } else if (enemyXData.get(i) - x > 0  && x >= 50){  //敵が右にいて、左に余裕がある時
                                enemyXFlag = -1;
                        } else if (enemyXData.get(i) - x < 0  && x > 450){  //敵が左にいて、ギリギリまで右に寄ってる時
                                enemyXFlag = -1;
                        } else if (enemyXData.get(i) - x < 0 && x <= 450){  //敵が左にいて、右に余裕がある時
                                enemyXFlag =  1;
                        } else if (enemyXData.get(i) - x == 0 && x <  50){  //敵が真上にいて、ギリギリまで左に寄ってる時
                                enemyXFlag =  1;
                        } else if (enemyXData.get(i) - x == 0 && x > 450){  //敵が真上にいて、ギリギリまで右に寄ってる時
                                enemyXFlag = -1;
                        } else  {
                                enemyXFlag =  1;
```

```
                }
            }

            for(int i = 0; i < enemyYData.size(); i++) {
                if(enemyYData.get(i) - y > 0 && y > 450) {          //敵が上にいて、自機が上限まで下がってる時
                    enemyYFlag = -1;
                } else if (enemyYData.get(i) - y > 0 && y <= 450){ //敵が上にいて、自機が前に出てる時
                    enemyYFlag =  1;
                } else if (enemyYData.get(i) - y < 0  && y <  50){ //敵が下にいて、自機が下限まで上がってる時
                    enemyYFlag =  1;
                } else if (enemyYData.get(i) - y < 0  && y >= 50){ //敵が下にいて、自機が前に出てる時
                    enemyYFlag = -1;
                } else if (enemyYData.get(i) - y == 0 && y > 450){ //敵が真横にいて、自機が上限まで下がってる時
                    enemyYFlag = -1;
                } else if (enemyYData.get(i) - y == 0 && y <  50){ //敵が真横にいて、自機が下限まで上がってる時
                    enemyYFlag =  1;
                } else {
                    enemyYFlag =  1;
                }
            }

            move(enemyXFlag, enemyYFlag);

            searchReactE =false;
        } else {
            returnMove();
        }

        //保存しておいた敵性オブジェクトの情報をクリア
        enemyXData.clear();
        enemyYData.clear();
        bulletXData.clear();
        bulletYData.clear();
        bulletDirectionData.clear();
}

//敵機のサーチ
public void searchEnemy(double ex, double ey){
        Lpx = x - 100; //自機の左側
```

```
        Rpx = x + 100; //自機の右側
        Upy = y - 100; //自機の上側
        Dpy = y + 100; //自機の上側

        if(Upy <= ey && Dpy >= ey){
                if(Lpx <= ex && Rpx >= ex){
                        searchReactE = true;
                        enemyXData.add(ex);
                        enemyYData.add(ey);
                        String exData = String.format("%.0f", ex);
                        String eyData = String.format("%.0f", ey);
                        System.out.println("敵機(" +  exData + ", " +  eyData + ")");

                }
        }
}

//敵弾のサーチ
public void searchBullet(double bx, double by, double bd){
        Lpx = x - 100; //自機の左側
        Rpx = x + 100; //自機の右側
        Upy = y - 100; //自機の上側
        Dpy = y + 100; //自機の上側

        if(Upy <= by && Dpy >= by){
                if(Lpx <= bx && Rpx >= bx){
                        bulletXData.add(bx);
                        bulletYData.add(by);
                        bulletDirectionData.add(bd);
                        searchReactB = true;
                        String bxData = String.format("%.0f", bx);
                        String byData = String.format("%.0f", by);
                        String bdData = String.format("%.0f", bd);
                        System.out.println("敵弾(" +  bxData + ", " +  byData + ", " +  bdData +"度)");
                }
        }
}

//最初の位置(250,400)地点に自機を戻す
```

```
public void returnMove(){
	if(x > 250 && y > 400){
		move(-0.5, -0.5);
	} else if (x > 250 && y < 350) {
		move(-0.5, 0.5);
	} else if (x < 250 && y > 350) {
		move(0.5, -0.5);
	} else if (x < 250 && y < 350) {
		move(0.5, 0.5);
	} else if (x == 0 && y > 350) {
		move(0, -0.5);
	} else if (x == 0 && y < 350) {
		move(0, 0.5);
	} else if (x < 250 && y == 350) {
		move(0.5, 0);
	} else if (x > 250 && y == 350) {
		move(-0.5, 0);
	}  else if (x == 0 && y == 0) {

	}

}

//最も敵弾と自機の距離が近いものを判別する
public int compare(ArrayList<Double> bulletDist){
	int com = 0;
	double min = bulletDist.get(0);
	for(int i=0; i < bulletDist.size(); i++){
		if (bulletDist.get(i) < min){
			min = bulletDist.get(i);
			com = i;
		}
	}
	return com;
}

//描画処理
public void draw(Graphics g) {
	if (active) {
```

```
                g.setColor(Color.red);
                //三角形の描画
                g.drawLine((int)(x), (int)(y-14), (int)(x-10), (int)(y+7));
                g.drawLine((int)(x), (int)(y-14), (int)(x+10), (int)(y+7));
                g.drawLine((int)(x-10), (int)(y+7), (int)(x+10), (int)(y+7));
        }
}
}
```