卒業研究報告書

題目

シューティングゲームにおける未経験者と経験者の差異の解析

指導教員

石水 隆　講師

報告者

14-1-037-0084

松下 継

近畿大学理工学部情報学科

平成28年1月29日提出

概要

シューティングゲームとは,自機を上下左右に操作し,敵機の攻撃から逃れながら,ゲーム毎に設けられた目標を達成するゲームである.

シューティングゲームは,未経験者と経験者のプレイングの差が大きいため,幅広い難易度が用意されることが多い.その中で作成者側の難易度とプレイヤー側の難易度に齟齬が生じる場合があるため,正しい難易度を設定する際には,未経験者,経験者の特徴を理解する必要がある.本研究では,敵機,敵弾を避ける際の未経験者と経験者の差異の解析により,それぞれの特徴を抽出する.

本研究では，両者の差異を検証するためにJavaで作成されたシューティングゲームを未経験者, 経験者,それぞれ10人に5回ずつプレイしてもらい,差異を解析する.

シューティングゲームでは,敵機を破壊することと敵弾を避けることの両方が求められるが,本研究では,敵弾の避け方の差異を重点的に調査するために,自機は弾を発射できない機体とし,被験者には敵弾を避けることのみを意識してプレイしてもらう.出現する敵機は,ゆらゆら進みながら6方向に拡散する弾を発射する機体,出現したらある程度進み,自機を狙って弾を発射した後，撤退する機体Bの2種類とし,敵機の種類により避け方がどのように差が出るかも検証する.

今回,未経験者,経験者の差異を解析するために得た特徴は,生存時間(フレーム),衝突した弾が生成された位置,そのときの自機の位置,衝突した位置,自機の移動した距離,等の11項目である.

目次

1. 序論

1.1本研究の背景

現在では,様々なジャンルのシューティングゲームが登場してきた.大別すると,x,y軸方向の二次元視点で行う2Dシューティングゲームと,x,y,z軸方向の三次元視点で行う3Dシューティングゲームがある.

2Dシューティングゲームでは,「スペースインベーダー」を始めとする画面がスクロールしないシューティングゲームである固定画面シューティング,「ゼビウス」を始めとする画面が上から下へとスクロールしていく縦スクロールシューティング,「グラディウス」を始めとする画面が右から左方向にスクロールする横スクロールシューティング,「東方project」を始めとする自機の当たり判定が小さく,大量の敵弾から避けることを重視した弾幕系シューティングゲーム,縦,横交互にスクロールする縦横両スクロールシューティング,斜め方向にスクロールするクォータービューシューティング,プレイヤーの任意の方向に自機を操り,画面がスクロールする多方向スクロールシューティング等,様々なジャンルが存在する.

一方、3Dシューティングゲームは,「スペースハリアー」をはじめとするz軸上でスクロールする奥スクロールシューティング,「Call of Duty」,「Battle Field」を始めとする一人称視点で3Dマップ上で兵士を自由に操るファーストパーソン・シューティング(FPS),「MetalGearSolid」を始めとする三人称視点で3Dマップ上で兵士を自由に操るサードパーソンシューティング(TPS),「ACE　COMBAT」を始めとする空など自由な空間上で戦闘機を操るフライトシューティング等のジャンルが存在する.また,最近ではFPSはeSports化されるなどで,人気が沸騰している.

1.2本研究の目的

シューティングゲームは,様々なジャンルが存在するが, ジャンルを問わずシューティングゲーム全体の傾向として未経験者と経験者のプレイングの差が大きいため,シューティングゲームは幅広い難易度が用意されることが多い.しかし,作成者側の想定する難易度とプレイヤー側が感じる難易度に齟齬が生じる場合があるため,作者側が初心者向けに調整したと思っていても、必ずしも初心者が楽しめるとは限らない．したがって，作者側はプレイヤーの視点から見た正しい難易度を設定するようにしなければならない．そのためには,未経験者,経験者の特徴 を理解する必要がある.

本研究では,敵機,敵弾を避ける際の未経験者と経験者の差異の解析により,それぞれの特徴を抽出する.

1.3本報告書の構成

本報告書の構成は以下の通りである.2章に研究内容,3章に調査結果・考察,4章に結論・今後の課題を述べる.以降は,謝辞と参考文献を示し,付録に調査で使用したプログラムのソー

スコードを載せる.

2. 研究内容

本節では,本研究で用いたシューティングゲームの仕様,調査項目,調査方法を記述する.

2.1ゲーム内容

本節では，調査をするために使用したJavaで作成された固定画面シューティングゲームの内容を説明する.

本研究で用いたシューティングゲームの仕様を以下に示す.

- 画面サイズ：500px × 500pxの正方形フィールド
- 敵機の種類：以下の2種類

  a機：y軸下方向にゆらゆらと動きながら,50フレーム毎に6方向に弾を発射する.

  b機：y軸下方向に200px移動したところで静止し,30フレーム毎に自機方向に5つの弾を発射する.
- 敵機出現間隔：80フレーム毎に出現
- 当たり判定:自機の中心が敵機,敵弾に対して距離8px未満

敵機の種類を上記の2種類にしたのは,あらゆるジャンルの代表作では,自機を狙う機体,無作為に弾を発射する機体のどちらかは大抵採用されていたためである.また,敵機出現間隔は,初心者であっても300フレーム以上の生存が期待できるように30フレーム毎の出現とした.

本研究で用いたシューティングゲームのプレイ画面を図1に示す.

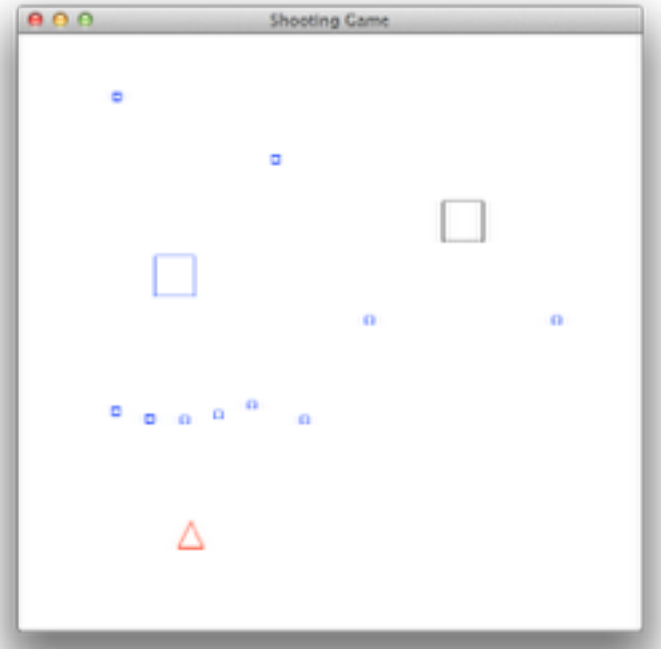

図1 プレイ画面

2.2調査項目

本研究では,プレイヤーの特徴を抽出するために以下の11項目を対象とした.

A.生存時間
B.2種類ある機体のうちどちらの弾に衝突したか
C.衝突した弾が生成された位置
D.衝突した弾が生成されたときのプレイヤーの位置
E.衝突した弾が生成された位置と衝突した弾が生成されたときのプレイヤーの位置の距離
F. プレイヤーの衝突した位置
G.衝突した弾が生成されたときのプレイヤーの位置とプレイヤーの衝突した位置の距離
H.衝突したときの画面上にある弾の数(位置)
I. 移動キーを使用した比率
J. 移動した距離
K.移動キー一回に対する移動距離

調査項目A.の生存時間は，経験者と未経験者で明らかに差があると思われる項目である.また,経験者と未経験者では,弾の予測範囲が異なり,それが生存時間に繋がると考え,C,D,Fの座標,E,Gの距離を調査項目とした.また,シューティングゲームでは,画面端に追い込まれ,囲まれた敵弾に衝突することがあるため,衝突した際の弾の位置を把握できるようHを調査項目とした.

I,J,Kの項目を把握することにより,経験者,未経験者の移動キーの使い方の違いに注目できるよう調査項目とした.

2.3調査方法

シューティングゲームの未経験者と経験者の差異を検証するために，被験者にプレイしてもらう．調査に先立ち「日常的にシューティングゲームをプレイしていたことがあるか」と問いを行い未経験者と経験者を区別し,ゲーム内容,操作方法の説明を行った.本調査では,未経験者，経験者それぞれ10人を被験者とした．各被験者には,一人5回ずつプレイしてもらい,計100回のデータを集めた.

3.　調査結果・考察

本節では,2.2節の調査項目の結果にて,経験者と未経験者で大きく差が見られた項目について考察を行う.調査結果を表1に示す.

表 1　経験者と未経験者の差異(平均値±標準偏差)

調査項目	経験者	未経験者
A	730±340	420±140
B：a機/b機	18/32	26/24
C：x/y	250±90/220±110	250±90/220±100
D：x/y	200±130/370±60	230±90/390±40
E：x/y	140±90/160±80	80±70/170±90
F：x/y	190±120/380±60	220±100/390±50
G：x/y	160±80/210±90	180±90/210±110
H：上/下/左/右	17/3/6/4	15/2/4/4
I：上/下/左/右	2.7/2.4/2.2/2.7	1.6/1.5/3.2/3.7
J：x/y	590±370/660±540	330±220/150±150
K：x/y	30±12/33±11	37±13/39±17

以下では表1の各項目の結果について考察する.

3.1生存時間についての結果

経験者の平均生存時間は,730フレーム,標準偏差は340フレームである．つまり,80フレーム毎に敵機が出現するため,4回から13回敵機が出現する間に弾に衝突することが68％という結果となった.生存時間のばらつきが大きいのは,a機,b機の出現確率がランダムなため,出現がb機に偏った際や5回プレイしてもらったため,1回目と5回目では操作の慣れで生存時間が大きく変わったためであると考えられる.

未経験者の平均生存時間は,420フレーム，標準偏差は140フレームである.つまり,敵機の

出現が3回から7回目の間に衝突することが68%という結果となった.敵機の出現の偏りや慣れなどの干渉が少ないため、経験者に比べブレが少ない結果となったと考えられる.

3.2どちらの機体が発射する弾に当たったかについての結果

2種類の機体のどちらが発射する弾に当たったかについては，経験者は,a機に18回,b機に32回と自機を狙うb機の弾に衝突した回数が多いのに対し,未経験者は,a機に26回,b機に24回と機体の種類に関係なく衝突している結果となった.

a機の弾は,b機の弾に比べ,弾道が予測しやすいため,経験者は衝突する回数が少なく,弾道が予測しづらいb機の弾に衝突する回数が多くなったと考えられる.また,未経験者は,弾道を予測するより,自機に近づいてきて弾を都度に避けているために,機体の種類に関係なく衝突したと考えられる.

3.3衝突した弾の生成位置についての結果

衝突した弾の生成位置については，y軸上では両者に差はあまり見られない結果となった.一方x軸上では,経験者の平均距離が140px,標準偏差は90pxであるのに対し,未経験者の平均距離は80px,標準偏差は70pxという結果である.

経験者は,未経験者に比べ広い視野で敵弾を予測し,弾を避けているため,未経験者より平均距離が長くなったと考えられる.経験者は,50pxから230pxの間に68%存在しているが,これは,生存時間が長くなるほど,画面上の敵が増え,敵弾の数も増える,そのため,全ての弾を把握できず,近くの敵機の弾に衝突することがあったため,このような結果となったと考えられる.

一方,未経験者は,自機に近づいてくる弾を都度に避けてるため,近づいてくる敵機に対しても近距離で弾を避けているため,10pxから150pxの間に68%が存在する結果となったと考えられる.

3.4衝突時の画面上の弾の数についての結果

衝突時の画面上の弾の数については，経験者,未経験者ともに衝突する際は,自機上側に弾が一番多く,次いで左右,一番少ないのは下側という結果となった.これは画面上側から敵機がくるため,自機上側が一番弾が多く,下側が一番少ない結果となったと考えられる.衝突した際の弾の位置関係は,経験者,未経験者ともにあまり差異が見られなかったが,経験者が衝突する際は,画面上におよそ30個,未経験者はおよそ25個という結果となっている.この結果より,対応できる画面上の弾の数には差異が見られることがわかった.

3.5方向キーの入力比率についての結果

方向キーの入力比率については，経験者は上下左右の移動キーを万遍なく使用しているのに対し,未経験者は,上下の使用率が低く,左右の使用率が高い結果となった.

経験者は,x軸y軸どちらの弾も平均的に避けるために上下左右のキーを満遍なく使用しているが,未経験者はx軸からの弾に対応することで,y軸方向の弾への対応ができていないため上下の使用率が低くなっていると考えられる.

3.6 1回のキー入力に対する移動距離についての結果

経験者のx軸,y軸それぞれの移動キー一回に対する平均移動距離は,30px,33px,標準偏差は12px,11pxである.一方,未経験者のx軸,y軸それぞれの移動キー一回に対する移動距離は,37px,39px,標準偏差は13px,17pxである.経験者は,未経験者に比べ,細かく移動することで,b機の5つの弾の間など,小さな隙間にも対応していたため,経験者の平均移動距離が短くなったと考えられる.

もう少し差がでると予測していたが,経験者は,敵機が1点へ集中しはじめると,敵機が少ない方向へと大きい移動をすることがあったためこのような結果となったと考えられる.敵弾を避ける挙動としての移動キー一回に対する移動距離では,今回の結果と違ったものが得られると考えられる.

3.7 考察

経験者と未経験者の明らかな差は生存時間の長さである．生存時間に差異が生じる主な要因としては,視野の広さ,移動キーの使い方の2点にあると考えられる.E,F,Hの項目の結果より,自機が衝突した際の状態を経験者,未経験者をそれぞれ,図2,図3に示す.なお,青線がEより予測できた視野の範囲,赤三角がFの項目より得られた自機が衝突した平均座標,黒線で区切られた範囲内の数字がHの項目より得られた範囲ごとに存在する平均の弾の数である.

図2,図3より,経験者と未経験者では,視野の広さが大きく差があることがわかる.また未経験者は経験者に比べ,青線が縦長四角型になっており,弾の数が多い上側に対しては,視野を広く保つことができているが,x軸の視野は狭くなっている.そのため,未経験者は上側からの弾を避けることに集中し,上下の移動キーの使用率が低くなったと考えられる.

本調査結果より，未経験者は,一方向からの攻撃を意識し,左右のみで避ける傾向があるため,難易度を低く設定する場合には,弾を一方向からのみにし左右の動きのみで避けられるようにすればよいと分かる．一方,経験者は,画面全体を意識し,上下左右に避ける傾向があるため,難易度を高く設定する場合には,上下左右全ての方向からの攻撃が来るようにするのが好ましいと考えられる.

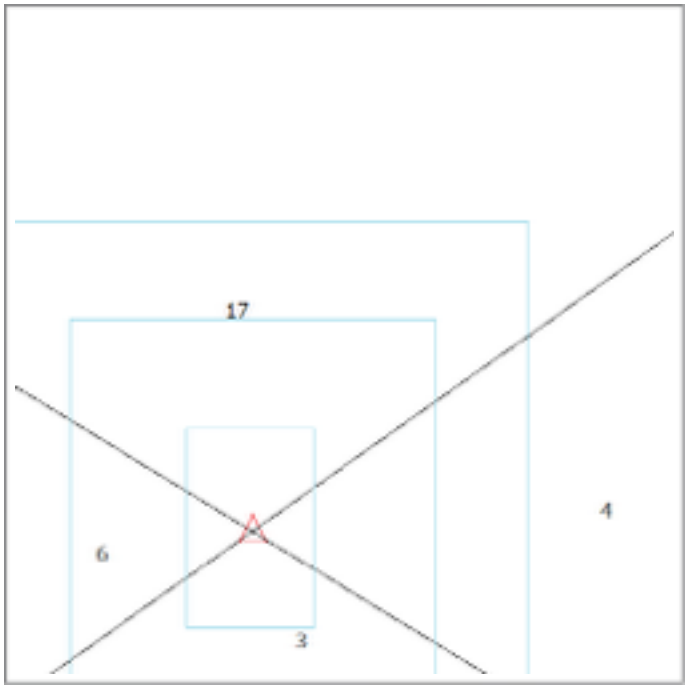

図2 経験者が弾に衝突したときの状態

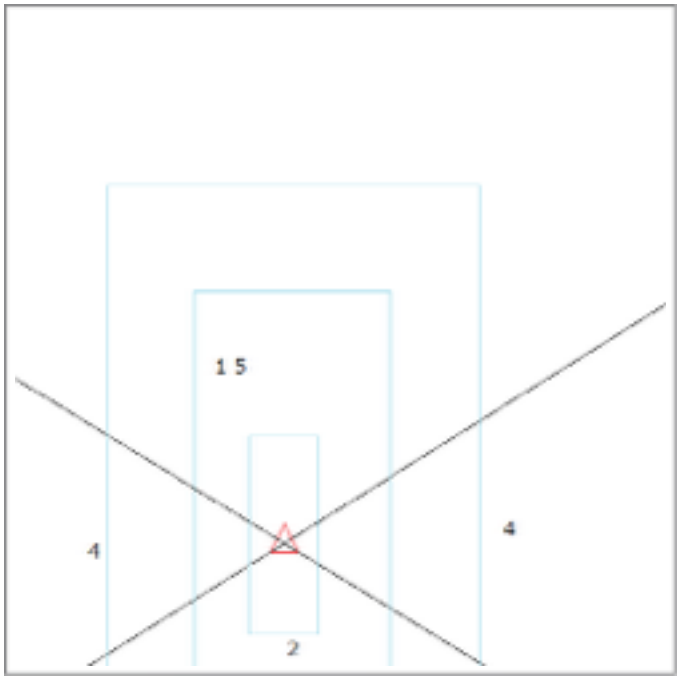

図3 未経験者が弾に衝突したときの状態

4.結論・今後の課題

本研究では,シューティングゲームにおいて未経験者と経験者の差異の解析を行った.調査項目として11項目の調査を行ったが,6項目において差異が見られた.また,未経験者,経験者の生存時間に関わる大きな要因として,視野の広さ,移動キーの使い方にあることがわかった.未経験者は弾が向かってくる画面上部のみに意識が向かい，左右の動きのみで敵弾を避けようとする傾向が強い．従って未経験者向けの難易度とするためには，左右の動きのみで避けられるような上方向からの攻撃にする必要がある．一方、上級者は画面全体を意識しており，上下方向も使って敵弾を避けようとする．従って上級者向けの難易度にするには，あらゆる方向からの攻撃が来るようにすればよい.

本研究では,弾を避けることに注目し,自機からたは弾を発射できないものとしたが,自機から発射できる機体の場合の検証,2種類の敵機を採用したが,自機を追撃する追撃弾を放つ機体や無作為に大量の弾を放つ機体など他の種類の機体を採用した際の検証,敵弾を避ける挙動としての移動キー一回に対する移動距離の検証,被験者には5回プレイしてもらったが1回目から5回目までの変化の検証,未経験者,経験者の2種類ではなく,初心者,初級者,中級者,上級者など細かく分けた際の検証,固定画面スクロールシューティング以外のジャンルでの検証が今後の課題である.

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. 筒井春也：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,情報学科　2016年度卒業研究報告書(2017)

2. 大村平：統計のはなし-基礎・応用・娯楽,日科技連, (2002)

3. 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, 研究報告 ゲーム情報学, Vol.2006-GI-016, pp.61-68, 情報処理学会 (2006)

4. 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

本研究で使用したシューティングゲームのソースコードを以下に示す.

- **Bullet.java**

```
package stgplay.ver3;
import java.awt.*;

public class Bullet extends GameObject{

	double direction;
	double speed;
	double speedX;
	double speedY;
	/*
	 * 松下継
	 */
	double firstx;          //生成位置x
	double firsty;         //生成位置y
	double px;                      //生成されたときのプレイヤーの位置x
	double py;                      //生成されたときのプレイヤーの位置y
	int aorr;                       //0:FireAim  1:FireRound

	int id;
	MyCanvas canvas;

	Bullet() {
		active = false;

	}

	public void move() {
		x += speedX;
		y += speedY;

		if ( (x < 0)||(500 < x)||(y < 0)||(500 < y) ) {
			active = false;
		}
	}

	public void draw(Graphics g) {
		g.setColor(Color.blue);
		g.drawRect((int)(x-3), (int)(y-3), (int)6, (int)6);
	}

	/**
	 * インスタンスを有効にする。インスタンスの使い回しをしているので、
	 * 初期化処理もここで行う。
	 * @param ix 生成する位置(X座標)
	 * @param iy 生成する位置(Y座標)
	 * @param idirection 向き(単位は度  0-360)
```

```
         * @param ispeed 速度(単位はピクセル)
         *
         * 松下継
         * 生成位置,生成されたときのプレイヤーの位置を代入
         */
        public void activate(double ix, double iy, double idirection, double
ispeed,double ifirstx,double ifirsty,double ipx,double ipy,int iaorr) {
                x = ix;
                y = iy;
                direction = idirection;
                speed = ispeed;
                active = true;                //弾のインスタンスを有効にする
                firstx = ifirstx;
                firsty = ifirsty;
                px = ipx;
                py = ipy;
                aorr = iaorr;

                //高速化のため、極座標からXY速度に変換しておく。
                double radian;
                radian = Math.toRadians(direction);   //度をラジアンに変換
                speedX = speed * Math.cos(radian);
                speedY = speed * Math.sin(radian);
        }


        public void activate(double speedX, double speedY) {
                active = true;                //弾のインスタンスを有効にする

                //高速化のため、極座標からXY速度に変換しておく。
                double radian;
                radian = Math.toRadians(direction);   //度をラジアンに変換
                speedX = speed * Math.cos(radian);
                speedY = speed * Math.sin(radian);

        }

        /**
         * 全方位に弾を撃つ
         */
        public static void FireRound(double x, double y,Player player) {
                for (int i = 0; i < 360; i += 60 )
                {
                        ObjectPool.newBullet(x, y, i, 3,player.x,player.y,0);

                }
        }

        /**
         * 自機狙い
         */
                public static void FireAim(double x, double y, Player player) {
                        double degree = Math.toDegrees(Math.atan2(player.y - y, player.x -
x));
                        ObjectPool.newBullet(x, y, degree, 4,player.x,player.y,1);
                        ObjectPool.newBullet(x, y, degree+20, 4,player.x,player.y,1);
```

```
                        ObjectPool.newBullet(x, y, degree-20, 4,player.x,player.y,1);
                        ObjectPool.newBullet(x, y, degree+10, 4,player.x,player.y,1);
                        ObjectPool.newBullet(x, y, degree-10, 4,player.x,player.y,1);

                }
}
```

• Enemy.java

```
package stgplay.ver3;
import java.awt.*;

public class Enemy extends GameObject {
        //生存時間（弾を撃つタイミングに使用）
        int counter = 0;
        //体力
        int hp;
        //種類
        int type;
        //あたり判定フラグ
        boolean ishit = false;
        //プレイヤーの位置を知っておくため、Playerインスタンスへの参照をを保持
        Player player;
        //打ち始めフラグ
        boolean startshoot;
        int shootnum;

        Enemy(Player iplayer) {
                //プレイヤーの位置を把握
                player = iplayer;
                //初期化時はactiveでない
                active = false;
        }
        Enemy() {
                //初期化時はactiveでない
                active = false;
        }

        /**
         * インスタンスを有効にする。インスタンスの使い回しをしているので、
         * 初期化処理もここで行う。
         * @param ix 生成する位置(X座標)
         * @param iy 生成する位置(Y座標)
         */
        public void activate(double ix, double iy) {
                x = ix;
                y = iy;
                type = (int)(Math.random()+0.5);
                active = true;                  //弾のインスタンスを有効にする
                hp = 1;
                counter = 0;
                boolean ishit = false;
                shootnum = Level.getLevel();
                startshoot = false;
        }
```

```
public void hit() {
	//体力減らす
	hp--;
	ishit = true;
	if (hp < 0) {
		active = false;
	}
}

/**
 * 動作を規定する。メインループ１周につき一回呼ばれる
 */
public void move() {
	//種類で分岐
	switch(type) {
		case 0:
			move_enemy0();
			break;
		case 1:
			move_enemy1();
			break;
		default:
	}
}

/**
 * 敵その１の動作
 */
void move_enemy0() {
	counter++;
	y++;
	//ゆらゆら
	x += Math.sin(y / 20);

	//画面外に出たら消去
	if ( (500 < y) ) {
		active = false;
	}

	//一定間隔で弾を撃つ
	if ((counter%50)==0) {
		Bullet.FireRound(x, y,player);
	}
}

/**
 * 敵その２の動作
 */
void move_enemy1() {
	counter++;
	double p = 60; //静止までの時間
	double q = 200;		//画面のどこで静止するか
	//二次関数で動きを表現
	y = (-q / Math.pow(p,2) * Math.pow((counter - p), 2) + q);
```

```
                //画面外に出たら消去
                if ( (-30 > y) ) {
                        active = false;
                }

                //一定間隔で弾を撃つ
                if ((counter%30)==0) {
                        //撃ち始め
                        startshoot = true;
                }

                //撃ち始めフラグが立っていたら、レベルと等しい回数、弾を撃つ
                if (startshoot) {
                        if (((counter%5)==0)&&(shootnum>0))
                        {
                                Bullet.FireAim(x, y, player);
                                shootnum--;
                        }
                }
        }

        /**
         * 描画処理。
         * １ループで一回呼ばれる。
         */
        public void draw(Graphics g) {
                //弾が当たっていたら色をオレンジに
                if (ishit) {
                        g.setColor(Color.orange);
                } else {
                        switch(type) {
                                case 0:
                                        g.setColor(Color.black);
                                        break;
                                case 1:
                                        g.setColor(Color.blue);
                                        break;
                                default:
                        }
                }
                ishit = false;
                g.drawRect((int)x-16, (int)y-16, (int)32, (int)32);
        }
}
```

• Game.java

```
package stgplay.ver3;

import java.awt.*;
import java.awt.event.*;

public class Game extends Frame{

        public static void main(String args[]) {
                //フレームの作成
                new Game();
```

```
        }

        Game() {
                // ウィンドウの初期化
                // タイトル
                super("Shooting Game");

                //クローズボタンによる終了処理の実装
                addWindowListener(new WindowAdapter() {
            public void windowClosing(WindowEvent evt) {
                 System.exit(0);
            }
         });

                setSize(500, 500);              //ウィンドウのサイズ

                // キャンバスの初期化
                MyCanvas mc = new MyCanvas();
                add(mc);                                //フレームにキャンバスを追加
                setVisible(true);               //ウィンドウの表示
                //ゲームデータの初期化
                mc.init();
                //スレッドを作成
                mc.initThread();
        }

}
```

• GameObject.java

```
package stgplay.ver3;
import java.awt.*;

public abstract class GameObject {

        public boolean active;

        public double x;
        public double y;

        abstract void move();

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

• KeyInput.Java

```
package stgplay.ver3;
import java.awt.event.*;

/**
*キーボードの入力を管理するクラス
*スペースキーで撃つ
*カーソルキーで移動
*/

public class KeyInput extends KeyAdapter{
```

```
boolean keyup;
boolean keydown;
boolean keyleft;
boolean keyright;

/*松下　継
 *
 * 移動ボタンを押した回数、各種
 *
 */
static int rightDirection=0,leftDirection=0,upDirection=0,downDirection=0;

/**
 * 押された瞬間を判別するため、0-2の値をとる
 * 0:押されていない 1:押されている 2:ついさっき押されたばかり
 */
int keyshot;

KeyInput() {
	keyup = false;
	keydown = false;
	keyleft = false;
	keyright = false;
	keyshot = 0;
}

/**
 * キーが押されたときに呼ばれる処理。
 * 変数にキー状態を保存する。
 */
public void keyPressed(KeyEvent e) {
	int keycode = e.getKeyCode();
	if (keycode == KeyEvent.VK_LEFT) {
		keyleft = true;
		leftDirection++;
	}
	if (keycode == KeyEvent.VK_RIGHT) {
		keyright = true;
		rightDirection++;
	}
	if (keycode == KeyEvent.VK_UP) {
		keyup = true;
		upDirection++;
	}
	if (keycode == KeyEvent.VK_DOWN) {
		keydown = true;
		downDirection++;
	}
	if (keycode == KeyEvent.VK_SPACE) {
		//初めて押された
		if (keyshot == 0) {
			//押された瞬間を表すフラグ
			keyshot = 2;
		} else {
			//押されている状態
			keyshot = 1;
```

```
                }
        }

        if (keycode == KeyEvent.VK_ESCAPE) {
                System.exit(0);
        }
}

/**
 * 押されていたキーを放したときに呼ばれる処理
 */
public void keyReleased(KeyEvent e) {
        int keycode = e.getKeyCode();
        if (keycode == KeyEvent.VK_LEFT) {
                keyleft = false;
        }
        if (keycode == KeyEvent.VK_RIGHT) {
                keyright = false;
        }
        if (keycode == KeyEvent.VK_UP) {
                keyup = false;
        }
        if (keycode == KeyEvent.VK_DOWN) {
                keydown = false;
        }
        if (keycode == KeyEvent.VK_SPACE) {
                keyshot = 0;
        }
}

/**
 * x軸の入力を取得
 * -1:右  0:なし  1:左
 */
public int getXDirection()
{
        int ret = 0;   //静止状態
        if (keyright) {
                ret = 1;

        }
        if (keyleft) {
                ret = -1;

        }
        return ret;
}

/**
 * y軸の入力を取得
 * -1:上  0:なし  1:下
 */
public int getYDirection()
{
        int ret = 0;   //静止状態
        if (keydown) {
                ret = 1;
```

```
        }
        if (keyup) {
                ret = -1;

        }
        return ret;
}

/**
 * ショットボタン（＝スペースキー）の状態を取得する
 * 0:押されていない 1:押されている 2:ついさっき押されたばかり
 */
public int checkShotKey()
{
        int ret = keyshot;
        if (keyshot==2) {
                keyshot = 1;
        }
        return ret;
}


/*松下　継
 *
 * 各Directionをリセットする
 *
 */
public void resetDirection(){
        rightDirection=0;
        leftDirection=0;
        downDirection=0;
        upDirection=0;
}

/*松下　継
 *
 * ゲッター
 *
 */
public int getRightDirection(){
        return rightDirection;
}
/*松下　継
 *
 * ゲッター
 *
 */
public int getLeftDirection(){
        return leftDirection;
}

/*松下　継
 *
 * ゲッター
 *
```

```
         */
        public int getUpDirection(){
                return upDirection;
        }
        /*松下　継
         *
         * ゲッター
         *
         */
        public int getDownDirection(){
                return downDirection;
        }

}
```

• Level.java

```
package stgplay.ver3;

public class Level {

        static int level;

        /**
         * 現在のレベルを返す (0-8)
         */
        public static int getLevel()
        {
                return 1;
//              return level;
        }

        public static void addLevel() {
                //最大レベルは8
                if (level < 8) {
                        level++;
                }
                System.out.println("level:" + level);//DEBUG
        }

        /**
         * レベルを０に戻す
         */
        public static void initLevel() {
                level = 0;
        }
}
```

• MyBullet.java

```
package stgplay.ver3;
import java.awt.*;

public class MyBullet extends GameObject{

        MyBullet() {
                active = false;
```

```
        }

        /**
         * ステップ毎処理
         */
        public void move() {
         y -= 15;
                //画面の外に出たら消去
                if ( (y < 0) ) {
                        active = false;
                }
        }

        /**
         * 描画処理。
         * １ループで一回呼ばれる。
         */
        public void draw(Graphics g) {
                g.setColor(Color.gray);
                g.drawRect((int)x-3, (int)y-10, (int)6, (int)20);
        }

        //初期化処理もここで行う（オブジェクトを使い回しているので）
        public void activate(double ix, double iy) {
                x = ix;
                y = iy;
                active = true;                  //弾のインスタンスを有効にする
        }
}
```

• MyCanvas.java

```
package stgplay.ver3;
import java.awt.*;
import java.util.Random;

import stg.ver5.Level;

public class MyCanvas extends Canvas implements Runnable{

        ObjectPool objectpool;
        KeyInput keyinput;
        Image imgBuf;
        Graphics gBuf;
        Random random;
        Title title;

        /**
         * シーン管理変数<p>
         * 0:タイトル画面 1:ゲームのメイン画面
         */
        public int scene;
        static final int SCENE_TITLE = 0;
        static final int SCENE_GAMEMAIN = 1;

        public boolean gameover;
        static int counter;
```

```
	int start;
	int end;

	//押されている
	static final int SHOT_PRESSED = 1;
	//今押されたばかり
	static final int SHOT_DOWN = 2;
	int shotkey_state;


	MyCanvas() {
		//キーリスナ実装
		keyinput = new KeyInput();
		addKeyListener(keyinput);
		setFocusable(true);				//フォーカスを取得できるようにす
る。
		random = new Random();			//乱数オブジェクト
		title = new Title();
	}

	public void init() {
		objectpool = new ObjectPool();

		//シーンはタイトル画面
		scene = SCENE_TITLE;
		gameover = false;
		//レベルの初期化
		Level.initLevel();
	}

	/**
	 * 外部からスレッドを初期化する。
	 */
	public void initThread() {
		Thread thread = new Thread(this);
		thread.start();
	}

	public void paint(Graphics g) {
		//オフスクリーンバッファの内容を自分にコピー
		g.drawImage(imgBuf, 0, 0, this);
	}

	/**
	 * メインループ
	 */
	public void run() {
		//オフスクリーンバッファ作成
		imgBuf = createImage(500, 500);
		gBuf = imgBuf.getGraphics();
		for(counter = 0; ; counter++) {

			shotkey_state = keyinput.checkShotKey();
			//バッファをクリア
			gBuf.setColor(Color.white);
			gBuf.fillRect(0, 0, 500, 500);
```

```
                        //シーン遷移用の変数で分岐
                        switch (scene) {
                        //タイトル画面
                        case 0:
                                title.drawTitle(gBuf);
                                //スペースキーが押された
                                if (shotkey_state == SHOT_DOWN) {
                                        //メイン画面に行く
                                        System.out.println("-----Game Start-----");
                                        scene = SCENE_GAMEMAIN;
                                        counter = start;
                                }
                                break;
                                //ゲームのメイン画面
                        case 1:
                                gameMain();
                                //System.out.println();
                                break;
                        }
                        //再描画を要求
                        repaint();
                        try{
                                Thread.sleep(20);                               //ループのウェイ
ト

                        }
                        catch(InterruptedException e){
                        }
                }
        }

        /**
         * 画面をいちいちクリアしないようにするため、
         * updateメソッドをオーバーライドする。
         * g 更新先グラフィックハンドル
         */
        public void update(Graphics g) {
                paint(g);
        }

        public int getCounter(){
                return counter;
        }


        /**
         * ゲーム画面のメイン処理
         */
        void gameMain() {

                //ゲームオーバーか？
                if (objectpool.isGameover()) {

                        //System.out.println(counter);
                        //ゲームオーバー文字を表示
                        title.drawGameover(gBuf);
```

```
                        if (shotkey_state == SHOT_DOWN) {
                                //ゲームを再初期化
                                init();
                        }
                } else {

                }

                //衝突の判定
                objectpool.getColision();
                objectpool.movePlayer(keyinput);
                //objectpool.shotPlayer();
                //敵出現間隔：80フレームごとに出現する
                if (counter % 80  == 0) {
                        ObjectPool.newEnemy(100 + random.nextInt(300), 0);
                }

                /*松下　継
                 * 自機から発射できない仕様に変更したため,コメントアウト

                //スペースキーが押されており、かつカウンタが３の倍数なら弾を撃つ（等間隔に弾を撃てる）
                 if ((!objectpool.isGameover())&&(counter % 3 == 0)) {
                        objectpool.shotPlayer();
                 }
                 */


                //ゲームオブジェクトの一括描画処理
                objectpool.drawAll(gBuf);

        }
}
```

• ObjectPool.java

```
package stgplay.ver3;
import java.awt.*;

public class ObjectPool {

        static Bullet[] bullet;
        static Enemy[] enemy;
        static MyBullet[] mybullet;
        Player player;

        //定数
        static final int DIST_PLAYER_TO_BULLET = 8;
        static final int DIST_PLAYER_TO_ENEMY = 16;
        static final int DIST_ENEMY_TO_MYBULLET = 16;

        //最大数の設定
        static final int BULLET_MAX = 100;
        static final int ENEMY_MAX = 100;
        static final int MYBULLET_MAX = 5;
        /*松下　継
         * オブジェクト生成
```

```
 */
MyCanvas mc = new MyCanvas();
KeyInput ki = new KeyInput();

ObjectPool() {
        //プレイヤーを作る
        player = new Player(250, 400, 4);
        player.active = true;

        //弾の配列を確保し、配列の要素分インスタンスを作る
        bullet = new Bullet[BULLET_MAX];
        for(int i = 0; i < bullet.length; i++) {
                bullet[i] = new Bullet();
        }

        //敵の配列を確保し、配列の要素分インスタンスを作る
        enemy = new Enemy[ENEMY_MAX];
        for(int i = 0; i < enemy.length; i++) {
                enemy[i] = new Enemy(player);
        }

        //プレイヤーの弾の配列を確保し、配列の要素分インスタンスを作る
        mybullet = new MyBullet[MYBULLET_MAX];
        for(int i = 0; i < mybullet.length; i++) {
                mybullet[i] = new MyBullet();
        }
}

/**
 * 描画処理。すべてのゲームオブジェクトを描画する。
 */
public void drawAll(Graphics g) {
        doGameObjects(g, bullet);
        doGameObjects(g, enemy);
        doGameObjects(g, mybullet);
        player.draw(g);
}

/**
 * ゲームオブジェクトの配列を参照し、
 * activeになっているインスタンスを移動・描画する
 */
public void doGameObjects(Graphics g, GameObject[] objary) {
        for (int i = 0; i < objary.length; i++) {
                if (objary[i].active == true) {
                        objary[i].move();
                        objary[i].draw(g);                              //委譲
                }
        }
}

/**
 * 弾の生成・初期化（実際は配列のインスタンスを使い回す）
 * ix 生成先x座標
 * iy 生成先y座標
 * idirection 動かす方向
```

```
         * ispeed 動かす速度
         * 弾のID（空きが無ければ-1）
         */
        public static int newBullet(double ix, double iy, double idirection, double
ispeed,double px,double py,int aorr) {
                for (int i = 0; i < BULLET_MAX; i++) {
                        if ((bullet[i].active) == false) {
                                bullet[i].activate(ix, iy, idirection, ispeed, ix,
iy,px,py,aorr);
                                return i;
                        }
                }
                return -1;              //見つからなかった
        }

        public static int newEnemy(double ix, double iy) {
                for (int i = 0; i < ENEMY_MAX; i++) {
                        if ((enemy[i].active) == false) {
                                enemy[i].activate(ix, iy);
                                return i;
                        }
                }
                return -1;              //見つからなかった
        }

        /**
         * プレイヤー弾の生成・初期化（実際は配列のインスタンスを使い回す）
         * ix 生成先x座標
         * iy 生成先y座標
         * プレイヤー弾のID（空きが無ければ-1）
         *
         */
        public static int newMyBullets(double ix, double iy) {
                for (int i = 0; i < MYBULLET_MAX; i++){
                        if ((mybullet[i].active) == false) {
                                mybullet[i].activate(ix, iy);
                                return i;
                        }
                }
                return -1;              //見つからなかった
        }

        /**
         * プレイヤーが弾を撃つ
         */
        public void shotPlayer() {
                //プレーヤーが可視の時だけ弾が打てる
                if (player.active) {
                        newMyBullets(player.x, player.y);
                }
        }

        /**
         * プレイヤーが移動する処理
         * keyinput キー入力クラスのインスタンス。
```

```
 */
public void movePlayer(KeyInput keyinput) {

	player.move(keyinput.getXDirection(), keyinput.getYDirection());
}

//public void movePlayer() {
//	player.move();
//}

/**
 * ２点間の距離を返す
 * ga ゲームオブジェクト
 * gb 比較先ゲームオブジェクト
 * 距離
 */
public double getDistance(GameObject ga, GameObject gb) {
	//三平方の定理
	double Xdiff = Math.abs(ga.x - gb.x);
	double Ydiff = Math.abs(ga.y - gb.y);
	return Math.sqrt(Math.pow(Xdiff,2) + Math.pow(Ydiff,2));
}

/*松下　継
 * プレイヤーと弾の距離　絶対値
 */
public double getDistancePB(double a,double b){
	double ans;
	if(a>b){
		ans =a - b;
	}else if(a<b){
		ans = b - a;
	}else {
		ans =0;
	}return ans;

}
/*松下　継
 * FireAimかFireRoundか判定
 */


public String getAimOrRound(int i){
	String answer;
	if(i==1){
		answer = "FireAim";
	}else{
		answer = "FireRound";
	}
	return answer;
}


/*松下　継
 * 自機の上下左右の弾の数
 */
```

```
	public int getBulletsPlace(GameObject[] ibullet,GameObject iplayer){
		int up=0,down=0,right=0,left = 0,ans =0;

		for(int i = 0; i < bullet.length; i++){
			if(ibullet[i].y<iplayer.y &&
(getDistancePB(ibullet[i].x,iplayer.x)<=getDistancePB(ibullet[i].y,iplayer.y))&&!
(ibullet[i].x==0)&&!(ibullet[i].y==0)){ //上側
				up++;
			}else if(ibullet[i].y>iplayer.y &&
(getDistancePB(ibullet[i].x,iplayer.x)<=getDistancePB(ibullet[i].y,iplayer.y))){//下側
				down++;
			}else if(ibullet[i].x<iplayer.x &&
(getDistancePB(ibullet[i].x,iplayer.x)>getDistancePB(ibullet[i].y,iplayer.y))){//左側
				left++;
			}else if(ibullet[i].x>iplayer.x &&
(getDistancePB(ibullet[i].x,iplayer.x)>getDistancePB(ibullet[i].y,iplayer.y))){//右側
				right++;
			}
		}

		ans = up*1000+down*100+right*10+left; //一つにまとめて返す

		return ans;
	}


	/**
	 * 衝突判定
	 */
	public void getColision() {
		int up=0,down=0,right=0,left=0;
		//弾vsプレイヤーの衝突
		for (int i = 0; i < bullet.length; i++) {
			if ((bullet[i].active)&&(player.active)) {
				//あたり判定
				if (getDistance(player, bullet[i]) <
DIST_PLAYER_TO_BULLET) {
					int ans=getBulletsPlace(bullet,player);

					up=ans/1000;
					down = ans%1000/100;
					right = ans%1000%100/10;
					left = ans%1000%100%10;
					//プレイヤー消滅（見えなくするだけ）
					player.active = false;
					//弾消滅
					bullet[i].active = false;
					/*松下継
					 * 自機が衝突した際,それぞれの項目を出力
					 */
					System.out.println("プレイヤーが生き残った時間(フレー
ム)["+mc.getCounter()+"]");
					System.out.println(getAimOrRound(bullet[i].aorr)+"
の弾に衝突した");
```

```
                                        System.out.println("衝突した弾の生成位置 x座標
["+bullet[i].firstx+"] y座標["+bullet[i].firsty+"]");
                                        System.out.println("衝突した弾が生成されたときのプレイ
ヤーの位置:x座標["+bullet[i].px+"]y座標["+bullet[i].py+"]");
                                        System.out.println("衝突した弾が生成されたときのプレイ
ヤーとの距離:x座標["+getDistancePB(bullet[i].px,bullet[i].firstx)+"]y座標
["+getDistancePB(bullet[i].py,bullet[i].firsty)+"]");
                                        System.out.println("プレイヤーの衝突した位置:x座標
["+player.x+"]y座標["+player.y+"]");
                                        System.out.println("衝突した時の弾の位置,自機の上
側:"+up+",下側:"+down+",右側:"+right+",左側:"+left);
                                        System.out.println("↑を押した回数
"+KeyInput.upDirection+"↓を押した回数"+KeyInput.downDirection+"→を押した回数
"+KeyInput.rightDirection+"←を押した回数"+KeyInput.leftDirection);
                                        ki.resetDirection();
                                        System.out.println("移動したx軸の距
離"+player.xDistance+"y軸"+player.yDistance);
                                        player.resetDistance();


                                }
                        }
                }

                /*松下継
                 *  自機から発射できないためコメントアウト
                //プレイヤーの弾vs敵の衝突
                 for (int i = 0; i < enemy.length; i++) {
                        if (enemy[i].active == true) {
                                for (int j = 0; j < mybullet.length; j++) {
                                        if (mybullet[j].active == true) {
                                                //あたり判定
                                                if (getDistance(enemy[i], mybullet[j]) <
DIST_ENEMY_TO_MYBULLET) {
                                                        //敵の体力を減らす
                                                        enemy[i].hit();
                                                        //弾消滅
                                                        mybullet[j].active = false;
                                                }
                                        }
                                }
                        }
        }*/

                //敵vsプレイヤーの衝突
                for (int i = 0; i < enemy.length; i++) {
                        if ((enemy[i].active)&&(player.active)) {
                                //あたり判定
                                if (getDistance(player, enemy[i]) < DIST_PLAYER_TO_ENEMY)
{
                                        //プレイヤー消滅（見えなくするだけ）
                                        player.active = false;
```

```
                                        //敵は消滅しない
                                }
                        }
                }
        }

        public boolean isGameover() {
                return !player.active;
        }

}
```

• Player.java

```
package stgplay.ver3;
import java.awt.*;
import java.util.ArrayList;
import java.util.Random;

public class Player extends GameObject{

        double speed;
        double xDistance=0,yDistance=0;
        Enemy  enemy;
        Bullet bullet;
        MyCanvas myCanvas;
        int time;
        public boolean searchReactE, searchReactB;
        double ex,ey, bx,by, Lpx,Upy,Rpx,Dpy, dx,dy, Xdiff,Ydiff, distance, Xmove,Ymove;
        ArrayList<Double>  enemyXData = new ArrayList<Double>();
        ArrayList<Double>  enemyYData = new ArrayList<Double>();
        ArrayList<Double> bulletXData = new ArrayList<Double>();
        ArrayList<Double> bulletYData = new ArrayList<Double>();
        ArrayList<Double> bulletDirectionData = new ArrayList<Double>();

    Player(double ix, double iy, double ispeed) {
        x = ix;
                y = iy;
                speed = ispeed;
                active = false;
                searchReactE = false;
                searchReactB = false;

        }

    /*松下　継
         * 移動した距離を保存
         */
    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)) {
                        double ix = getAbVal(x,postX);
                        xDistance += ix;
                        x = postX;


                }
```

```
            if ((0 < postY)&&(postY < 480)) {
                    double iy = getAbVal(y,postY);
                    yDistance += iy;
                    y = postY;
            }

    }
/*松下　継
     *２度目移行のプレイのためのリセット
     */
public void resetDistance(){
    yDistance=0;
    xDistance=0;

}

/*松下継
 *
 */
public double getAbVal(double a,double b){
            double ans;
            if(a>b){
                    ans =a - b;
            }else if(a<b){
                    ans = b - a;
            }else {
                    ans =0;
            }return ans;

    }

public void move(){
}

/*松下　継
     *
     */
    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));
            }
```

```
	}
}
```

• Title.java

```
package stgplay.ver3;

import java.awt.*;

public class Title {

	int count;
	Font titleFont;
	Font infoFont;

	Title() {
		count = 0;
		titleFont = new Font("sansserif", Font.BOLD, 30);
		infoFont = new Font("sansserif", Font.BOLD, 11);
	}

	public void drawTitle(Graphics g) {
		g.setColor(Color.black);
		count++;
		g.setFont(titleFont);
		g.drawString("S h o o t i n g",150,150);

		g.setFont(infoFont);
		g.drawString("hit SPACE key",200,350);

	}

	public void drawGameover(Graphics g) {
		g.setColor(Color.black);
		count++;
		g.setFont(titleFont);
		g.drawString("GAMEOVER",150,150);
	}
}
```