2015 – 後学期

プログラミング演習
グループプログラミングレポート

Shooting

学科	総合情報学科
クラス	J1
グループ番号	1
1410007	飯野　純平
1410019	江村　万里
1410055	佐藤　禎紀

1、概要説明

今回の課題で私達が作成したプログラムは「縦スクロール型シューティングゲーム」です。
自機・敵機を画面上に配置しており、自機を操作し、上下左右に移動、弾を発射して敵機を倒すなどしながら画面上部に進んでゆき、時間経過(目的地到達)でクリア画面に移行する。
敵機又は敵機が発射する弾に自機は触れるとライフが減り、ライフが 0 になるとゲームオーバー画面に移行する。

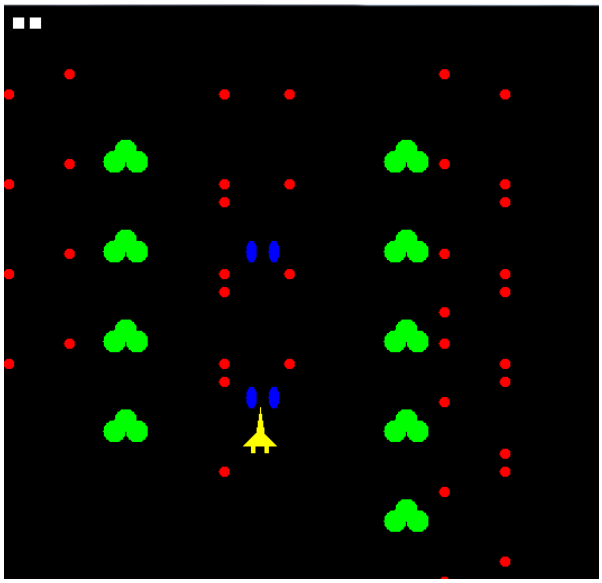


図.1　ゲーム画面

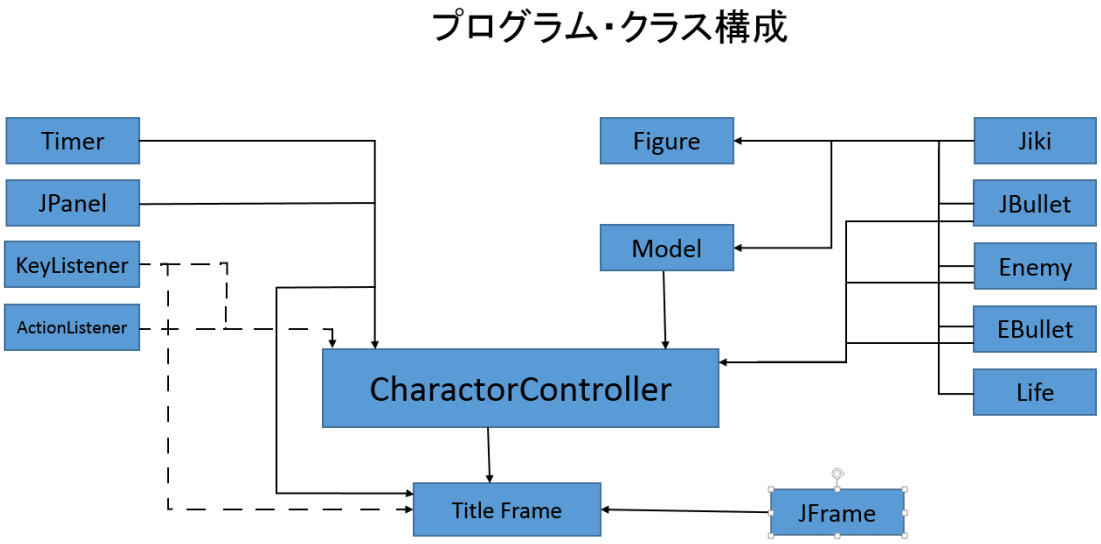
プログラム作成の分担は大まかに以下の通り
飯野：コントローラ部分一部
江村：ModelClass・オブジェクトの定義など
佐藤：TitleFrameClass・ゲームの大筋など
詳細は各担当部分の項にて記述。

作業は、基になる最低限のプログラム作成後各自が担当分のプログラムを作成しそれを共有、つなぎ合わせる形で作成した。

文責：飯野

2、設計方針

まず第一にプログラムの作成にあたって、既存のプログラムを機能拡張する形で行うのではなく一から全て自分達でプログラムを作成することを目標としてプログラムを作成しました。
使用した class は以下の通りである。



図,1　クラス相関図　プレゼンテーション用のスライドから

Jiki,JBullet,Enemy,EBullet,Life の各クラスで各オブジェクトを定義し、それらを Figure クラスによって描写する。またそれらは Model クラスで利用する。
ゲームの大筋は CharactorController クラスで成っており、Model からオブジェクトを配置してゲームを進行する。
メインとして用いられているのは TitleFrame クラスでウィンドウやタイトル画面・ゲーム画面など必要なものを用意し、ゲーム画面で CharactorController を呼び出す。

文責：飯野

3、プログラム説明

3.1 江村担当分

3.1.1 Figure クラス

ゲーム画面に表示させる物体のデータを格納するクラスの雛形。後述する Jiki クラス、Enemy クラス、JBullet クラス、EBullet クラス Life クラスにこれを継承させる。

(1) フィールド

int 型：x, y, width, height, speed

x, y は物体をゲーム画面に表示させる位置、width, height 物体の横と縦のの大きさ、speed は物体の移動する速さを表す。

Color 型：color

物体の色を示す。

(2) コンストラクタ

public Figure (int x,int y,int w,int h,int s,Color c)

引数 x, y, w, h, s, c をそれぞれフィールドの変数 x, y, width, height, speed, color に代入する。

(3) メソッド

Public void draw(Graphics g)

物体を描画する。ここではまだ内容を定義せず、子クラスでオーバーライドする。

3.1.2 Jiki クラス

自機のデータを格納する。Figure クラスを継承している。

(1) フィールド

Figyure クラスと同様

(2) コンストラクタ

public Jiki(int x,int y,int w,int h,int s,Color c)

自機のデータの初期化を行う。

(4) メソッド

public void draw(Graphics g)

自機を描画する。図形を描画するメソッドを用いて戦闘機のような形になるようにした。

3.1.3 Enemy クラス

敵機のデータを格納する。Figure クラスを継承している。

(1) フィールド

ここでは Figure クラスから追加した分について述べる。

int 型：type, shotcount

type は敵機の動作パターンを表し、shotcount は敵機が弾を発射する時間の間隔を表す。どちらも後述する Model クラスの EnemyMove メソッドで用いる。

boolean 型：dflag

このオブジェクトを削除する場合はこの変数を true にする。後述する Model クラスの DeleteFegure クラスで用いる。

(2) コンストラクタ

public Enemy(int x,int y,int w,int h,int s,int t,Color c)

敵機のデータの初期化を行う。

(3) メソッド

public void draw(Graphics g)

敵機を描画する。今回は３つの円を組み合わせた形で敵機を表現した。

3.1.4 JBullet クラス

自機が発射した弾のデータを格納する。Figure クラスを継承している。

(1) フィールド

ここでは Figure クラスから追加した分について述べる。

int 型：type

弾の動作パターンを表す。

boolean 型：dflag

Enemy クラスのものと同様である。

(2) コンストラクタ

public JBullet(int x,int y,int w,int h,int s,int t,Color c)

弾のデータの初期化を行う。

(3) メソッド

public void draw(Graphics g)

弾を描画する。今回はただの円形で弾を表現した。

3.1.5 EBullet クラス

敵機が発射した弾のデータを格納する。主な特徴は JBullet クラスと同様である。

3.1.6 Life クラス

自機のライフ（耐久値）の表示に関するデータを格納する。Figure クラスを継承している。

(1) フィールド

ここでは Figure クラスから追加した分について述べる。

int 型：count

自機の残りライフを表す。これが 0 になった時、ゲームオーバーとなる。

(2) コンストラクタ

public Life(int x,int y,int l)

ライフの初期化やゲーム画面に描画するライフ表示の初期化を行う。

(3) メソッド

public void draw(Graphics g)

ゲーム画面左上に自機の残りライフを表示させる。今回は自機の残りライフを白色の四角形の個数で表現した。

3.1.7 Model クラス

ゲーム全体のデータを管理し、物体の移動等に関するメソッドを格納している。

(1) フィールド

Jiki 型：jiki

自機のデータを表す。Figure クラスを継承している。主な特徴は

Life 型：life

自機のライフを表す。

配列：ArrayList<Enemy> enemy

ArrayList<JBullet> jbullet

ArrayList<EBullet> ebullet

それぞれ敵機、自機が発射した弾、敵機が発射した弾を表す。複数の敵機と弾を管理するために ArrayList を用いた。

(2) コンストラクタ

public Model()

自機とライフの初期設定と配列の生成を行う。

(3) メソッド

主に自機の動作、敵機と弾の生成・動作、ライフの変動、当たり判定、ゲームオーバー判定などを管理する。Model クラスはメソッドの数が多いが、ここでは特に説明が必要と思われるものについて述べる。

public void JikiMove(boolean r,boolean l,boolean d,boolean u)

自機の移動を表す変数が true の時、自機がその方向に移動する。変数の値はキーの操作によって変更される。佐藤担当の CharactorController クラスの actionPaformed

メソッドから呼び出すことで、自機をスムーズに移動させることができる。

public void EnemyMove()

敵機を移動させ、enemy の要素内のメンバー変数 shotcount の値が一定値になると弾を発射させる。for 文と ArrayList を用いて複数の敵機をすべて動かせるようにした。また enemy 要素内のメンバー変数 type の値によって敵機の動作パターンが変わるようにした。変数の値の種類を増やすことで動作パターンを追加できるのが特徴である。

public void BulletMove()

自機や敵機が発射した弾を移動させる。主な特徴は EnemyMove メソッドと同様である。

public void createBullet(int x,int y,int t)

自機や敵機が発射する弾を生成する。x, y は JBullet 型変数および EBullet 型変数のメンバー変数 x, y に代入され、t は JBullet 型変数および Ebullet 型変数のメンバー変数 type に代入される。t の値によって自機が発射した弾か敵機が発射した弾かがわかるようになっている。

public void Hitcheck()

後述する Hitbox メソッドを呼び出して自機、敵機、弾の当たり判定を行う。敵機や弾が衝突した場合、enemy, jbullet, ebullet の要素内のメンバー変数 dflag を true にする。dlag が true になった要素は後述する DeleteFigure メソッドによって削除される。自機が衝突した場合、life 変数のメンバー変数 counnt の値を 1 減少させる。これは自機のライフの減少を意味する。

public boolean Hitbox(int x1,int y1,int w1,int h1, int x2,int y2,int w2,int h2)

当たり判定に用いる。判定の方法は 2 つの四角形がかさなるかどうかで判定する。x, y は四角形の左上隅の点の座標、w, h は四角形の横と縦の大きさを表す。

public void DeleteFigure()

enemy, jbullet, ebullet の要素のうち、メンバー変数 dflag が true の要素を削除する。衝突した敵や弾を消すときに用いる。

public boolean GameOver()

自機のライフが 0 になったときに true を返す。ゲームオーバーの判定に用いる。

[文責：江村]

3.2 佐藤担当分

3.2.1CharactorController クラス

このクラスは JPanel を継承し、 ActionListener を実装しているクラスで、主にゲーム自体の構成やゲームの描画などを行っているクラスである。工夫した点は Timer に対する actionPerformed 内に Model クラス内の自機の動きに関するメソッドを入れることにより動きをスムーズにすることができたことである。

(1)コンストラクタ

```
29      public CharactorController()
30      {
31          timer = new javax.swing.Timer(50,this);      //timer に timer 挿入(0.05 秒おき)
32
33          model = new Model();  //model オブジェクトの生成
34
35          this.setBackground(Color.BLACK);  //背景設定
36          this.setFocusable(true);              //キー入力を GUI 部品が受け付ける
37          this.addKeyListener(this);            //キー入力を追加
38      }
```

このクラスのコンストラクタでは､Model クラスのオブジェクトを変数 model に生成し、その他上のように timer の生成、背景の設定、キー入力が受け付けるように関数を指定した。

(2)timerstart メソッド、timerstop メソッド

これらのメソッドはそれぞれコンストラクタで生成した timer をスタート、ストップするメソッドである。これらは次の TitleFrame クラスで使う。

(3)paintComponent メソッド

ここでは Model クラスで作ったメソッドを利用して自機、ライフ、敵、そして自機、敵の弾の描画を実行できるようにした。

(4)actionerPerformed メソッド

これは timer が更新するごとに実行されるメソッドである。つまり、時間が経つごとにこのメソッドは実行される。

```
86      public void actionPerformed(ActionEvent e)
87      {
88          time++;                         //timer を 0.1 秒に換算
89
90          model.JikiMove(jrflag,jlflag,jdflag,juflag);
91          if(jshotflag==true && time%5==0)
92              {
93                  model.createBullet(model.xJiki()+model.wJiki()/3,model.yJiki(),1);     //自機弾の発射
94                  model.createBullet(model.xJiki()-model.wJiki()/3,model.yJiki(),1);   //自機弾の発射
95              }
96
97          Game();
98
99          /*敵機の出現*/
100         if(time == 10)    //1 秒たったら
101             {
102                 for(int i=-320;i<=0;i+=80)
103                     model.createEnemy(100,i,1);
104
105                 for(int i=-320;i<=0;i+=80)
106                     model.createEnemy(350,i,1);
107             }
108         if(time == 150)
109             {
110                 for(int i=-320;i<=0;i+=80)
111                     model.createEnemy(50,i,2);
112
113                 for(int i=-320;i<=0;i+=80)
114                     model.createEnemy(400,i,3);
115             }
116         if(time == 200)
117             for(int i=-160;i<=0;i+=80)
118                 model.createEnemy(300,i,1);
119         if(time == 230)
120             for(int i=-160;i<=0;i+=80)
```

```
121                     model.createEnemy(150,i,1);
122             if(time == 260)
123                 for(int i=-160;i<=0;i+=80)
124                     model.createEnemy(350,i,3);
125             if(time == 290)
126                 for(int i=-160;i<=0;i+=80)
127                     model.createEnemy(100,i,2);
128             if(time == 360)
129                 for(int i=-160;i<=0;i+=80)
130                     model.createEnemy(270,i,1);
131
132
133
134
135             model.EnemyMove();
136             model.BulletMove();
137             model.HitCheck();
138             model.DeleteFigure();
139
140             repaint();                              //そして再描画
141
142         }
```

まず、timerの更新を視覚化するためにint型のtime変数を 更新ごとに1加算している。そして、model内のJikiMoveやcreateBulletを使って自機の動き、自機弾の発射を指定している。また、次に説明するGameメソッドを使用することにより、このあとのTitleFrameクラスでGameOver画面、GameClear画面に遷移できるようにしている。このあとの100行目から130行目に関しては、時間が経てばmodelクラス内の敵を出現するメソッドであるcreateEnemyメソッドを用いて敵を出現させるようにした。135行目から138行目のメソッドは当たり判定に関するメソッドで、詳しくはmodelクラスの説明内で解説してある。最後、時間が経つごとに再描画してある。

(5)Gameメソッド

このメソッドはゲームオーバーになったらint型変数iを1に、ゲームクリアになったらiを2にして、その後のTitleFrameクラスで画面の遷移ができるようにしている。

3.2.2TitleFrameクラス

このクラスはTitle画面、GameOver画面、GameClear画面も全て含めたこのプログラム全体の構成を担っているクラスである。工夫した点は CardLayout というレイアウトを

使い、画面の遷移を可能にすることができたことである。このクラスは JFrame を継承し、ActionListener 及び KeyListener を実装している。キー操作に関するメソッドは飯野が担当分であるため、ここではそれ以外のプログラムに関して説明していく。

(1)コンストラクタ

```
21      /*コンストラクタ(フレーム内の設定)*/
22      public TitleFrame()
23      {
24      /*JPanel 設定*/
25      p1 = new JPanel();       //大本のパネル
26      p2 = new JPanel();      // 1 ページ目のパネル（タイトル画面）
27      p3 = new JPanel();      // 2 ページ目のパネル（ゲーム画面）
28      p4 = new JPanel();      //3 ページ目のパネル（ゲームオーバー画面）
29      p5 = new JPanel();      //4 ページ目のパネル（クリア画面）
30      chara = new CharactorController();    //2 ページ目に貼り付けるゲーム画面
31      timer = new javax.swing.Timer(1,this);
32      timer.start();
33
34      p2.setBackground(Color.BLACK);   //背景設定
35      p4.setBackground(Color.BLACK);   //背景設定
36      p5.setBackground(Color.BLACK);   //背景設定
37
38
39      /*CardLayout の設定*/
40      layout = new CardLayout();
41
42      /*KeyListener 追加*/
43      p1.addKeyListener(this);
44      p1.setFocusable(true);
45
46      /*p2,p4 を BorderLayout で設定*/
47      p2.setLayout(new BorderLayout());
48      p4.setLayout(new BorderLayout());
49      p5.setLayout(new BorderLayout());
50
51      /*p2 に JLabel を貼り付ける*/
52      JLabel l2 = new JLabel("<html>"
53                                      +"<span
style='font-size:110pt;font-family:Impact,Charcoal;"
54                              +"color:yellow;'>"
```

```
55                          +"Shooting</span></html>",
56                          +JLabel.CENTER);
57
58
59          JLabel l3 = new JLabel("<html>"
60                          +"<span style='font-size:30pt;color:white;'>"
61                          +"十字キーで移動、z で弾発射。<br/>"
62                          +"<span style='font-size:30pt;color:white;'>"
63                          +"Enter でゲームスタート!!"
64                          +"</span></html>",
65                          +JLabel.CENTER);
66
67          p2.add(l2,BorderLayout.CENTER);
68          p2.add(l3,BorderLayout.SOUTH);
69
70          /*2 ページ目*/
71          p3.setLayout(new BorderLayout());
72          p3.add(chara,BorderLayout.CENTER);
73
74          /*p4 に JLabel を貼り付ける*/
75          JLabel l4 = new JLabel("<html>"
76                                      +"<span
style='font-size:90pt;font-family:Impact,Charcoal;"
77                          +"color:blue;'>"
78                          +"GAMEOVER</span></html>",
79                          +JLabel.CENTER);
80
81
82          JLabel l5 = new JLabel("<html>"
83                          +"<span style='font-size:30pt;color:white'>"
84                          +"2 秒後、タイトル画面を表示します。"
85                          +"</span></html>",
86                          +JLabel.CENTER);
87
88          p4.add(l4,BorderLayout.CENTER);
89          p4.add(l5,BorderLayout.SOUTH);
90
91          /*p4 に JLabel を貼り付ける*/
92          JLabel l6 = new JLabel("<html>"
93                                      +"<span
style='font-size:120pt;font-family:Impact,Charcoal;"
94                          +"color:red;'>"
```

```
 95                          +"GAME<br/>CLEAR!!</span></html>",
 96                          +JLabel.CENTER);
 97
 98
 99      JLabel l7 = new JLabel("<html>"
100                          +"<span style='font-size:30pt;color:white'>"
101                          +"2 秒後、タイトル画面を表示します。"
102                          +"</span></html>",
103                          +JLabel.CENTER);
104
105
106      p5.add(l6,BorderLayout.CENTER);
107      p5.add(l7,BorderLayout.SOUTH);
108
109      /*p1 を CardLayout で設定し、p1 にパネルを貼り付ける。*/
110      p1.setLayout(layout);
111      p1.add(p2,BorderLayout.CENTER);
112      p1.add(p3,BorderLayout.CENTER);
113      p1.add(p4,BorderLayout.CENTER);
114      p1.add(p5,BorderLayout.CENTER);
115      getContentPane().add(p1,BorderLayout.CENTER);          //これがないと
表示されない(はず)
116
117
118      /*Frame 設定*/
119      this.setSize(550,550);                                  //Frame は
```

JPanel は全体、タイトル画面、ゲーム画面、ゲームオーバー画面、ゲームクリア画面の 5 つの Panel を用意した。そして、timer の生成、背景設定、キー入力が受け付けるための設定、レイアウト設定をした。

51 行目から 68 行目に関しては Title 画面の生成をしている。JLabel を使い、HTML でフォントや文字の大きさなどを指定しながらゲームタイトルと簡単なゲーム操作を記載した。また、74 行目から 89 行目、91 行目から 107 行目はそれぞれ GameOver 画面、GameClear 画面を Title 画面と同様に JLabel、HTML を使用しながらゲームオーバーもしくはゲームクリアである旨と画面が一定時間後に自動的に遷移するということを記載した。また、変数 chara に前記の CharacterController クラスのオブジェクトを生成し、その chara を p3 に貼り付けた。

そして、p1 に複数の部品を切り替えて表示できる CardLayout を設定し、p2~p5 を貼り

付け、それが表示されるように 115 行目のようなメソッド記述をした。
最後には Frame のサイズなどの設定をした。

(2)actionPerformed メソッド

このメソッドは timer が更新されるごとに実行されるメソッドで、主にここでは画面の遷移について設定してある。

前記の Game メソッドにより、ゲームオーバーになれば i==1、ゲームクリアになれば i==2 となるように指定したある。このメソッドではこの i==1 である時、layout.next メソッドを使ってページを遷移し、GameOver 画面に移行するように設定してある。i==2 のときも同様に layout.next メソッドを使って GameClear 画面に移行するようにしている。そして、変数 i を 4 に変えて、int 型の time 変数を更新されるごとに加算。一定時間たったら setVisible メソッドでこのフレームを消し、新たなフレームを生成することによって 新たなゲームがプレイできるようにしている。

(3)main メソッド

このメソッドではフレームの生成しか行っていない。

[文責：佐藤]

3.3 飯野担当分

自機の操作等

```
19      boolean jrflag = false;     //右ボタンに関するフラグ
20      boolean jlflag = false;     //左ボタンに関するフラグ
21      boolean jdflag = false;     //下ボタンに関するフラグ
22      boolean juflag = false;     //上ボタンに関するフラグ

140   /*フラグ boo が立っていたら*/
141   if(boo == true)
142     {
143     if(k == KeyEvent.VK_RIGHT)      //右押すと
144       chara.jrflag = true;  //CharactorController クラスの jrflag が立つ
145
146     else if(k == KeyEvent.VK_LEFT)      //左押すと
147       chara.jlflag = true;  //CharactorController クラスの flflag が立つ
148
149       else if(k == KeyEvent.VK_UP)
150         chara.juflag = true;
151
152       else if(k == KeyEvent.VK_DOWN)
153         chara.jdflag = true;
154
155       else if(k == KeyEvent.VK_Z)
156         chara.jshotflag = true;
157     }
158   }
159
160
161   /*キーをはなした時の操作*/
162   public void keyReleased(KeyEvent e)
163   {
164   int k = e.getKeyCode();         //キーの情報を取得
165   if(boo == true)                 //フラグが有効なら
166     {
167     switch(k)
168       {
169
170       case KeyEvent.VK_RIGHT:     //右ボタンを離すと jrflag 無効
171               chara.jrflag = false;
172               break;
173
174       case KeyEvent.VK_LEFT:      //左ボタンを離すと jlflag 無効
175               chara.jlflag = false;
176               break;
177
178         case KeyEvent.VK_UP:
179               chara.juflag = false;

180                 break;
181
182           case KeyEvent.VK_DOWN:
183               chara.jdflag = false;
184               break;
185
186           case KeyEvent.VK_Z:
187               chara.jshotflag = false;
188               break;
189
190       }
191     }
```

まず各ボタンに関するフラグを boolean 変数で定義し、ボタンが押されているとき true、放されているとき false となるように関数を定義する。

また、自機を移動する関数、つまり自機の座標に一定値を加減算して再描画する関数を別で用意しておき、各ボタンのフラグが true のときのみ関数を実行するように条件分岐を設定する。また Z キーに関しても、自機の弾を生成する関数を用意し、ボタンが押された場合に関数を実行するように設定している。

文責：飯野

4.実行例

まず、プログラムの実行をする。Java ファイルで実行するにはメインクラスである TitleFrame クラスを実行。Jar ファイルで実行する場合には以下のように実行する。

```
Yoshiki$ java -jar Shooting.jar
```

すると以下のようなタイトル画面が表示される。

図１：タイトル画面

そして、Enter キーを押すとゲームがスタートし、次のようなゲーム画面に遷移する。

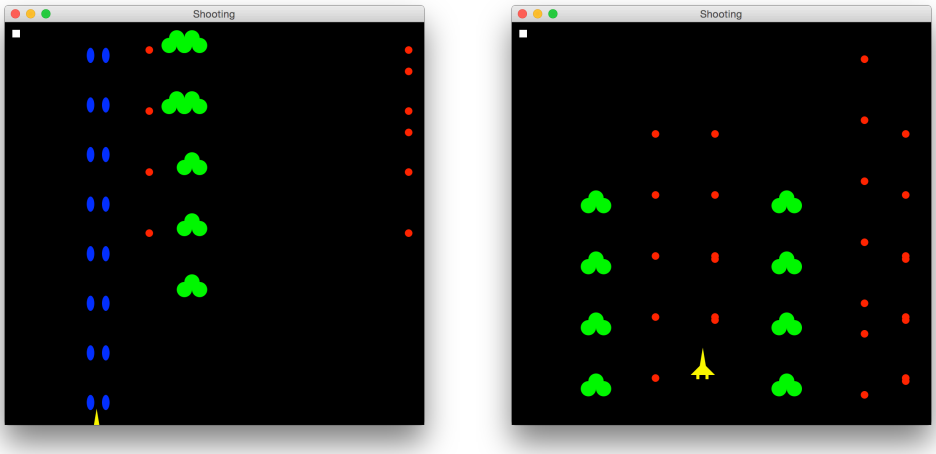


図２：ゲーム画面

ゲーム画面に遷移したら、ゲームが開始。操作は付録１にも添付してあるように十字キーで上下左右の移動、z キーで自機弾の発射をする。ゲーム画面の右上にある白い四角は自機の残り残機で、最初は 3 機、0 になるとゲームオーバーとなり、以下のようなゲームオーバー画面に遷移する。

図３：ゲームオーバー画面

また、ゲームが始まってから一定時間（40 秒）たったらゲームクリアとなり、ゲームクリア画面に遷移する。

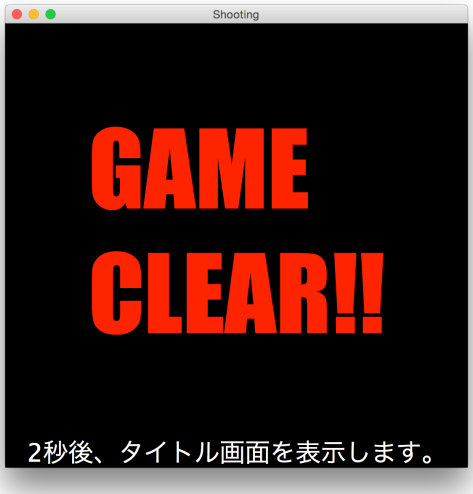


図４：ゲームクリア画面

ゲームオーバー画面、ゲームクリア画面はどちらも２秒たったら自動的に消えて、新たなタイトル画面が出てくる。

［文責：佐藤］

5.考察

我々はシューティングゲームを作るにあたって、最低でも次の５つの要素を実装することを目標とした。それは (1)プレイヤーが操作する自機 (2)標的となる敵機 (3)弾 (4)それらの当たり判定 (5)タイトル・ゲームオーバー画面への遷移である。

まず(1)に対しては自機が縦横に移動できること、一定間隔で連続して弾を発射できること、という２つの条件を課した。これらはどちらも KeyListener と Timer を組み合わせることで実現している。KeyListener の keyTyped メソッドのみではキーを押してから操作のためのメソッドの呼び出しまでタイムラグが発生してしまった。そこで keyPressed メソッドと keyReleased メソッドから操作のためのメソッドを呼び出す条件となる変数を変動させ、操作のためのメソッドは Timer から呼び出されるメソッドに組み込んだ。これによりタイムラグは改善されている。また、連続で弾を発射する際に一定の時間間隔を置くために、Timer から呼び出されるメソッドによって変動する time 変数の値が一定値になることを弾の発射の条件に追加した。この変数は自機のデータやキー操作に関するメソッドとは独立した変数であるため、キー操作と弾の発射に若干のズレが生じてしまう可能性がある。

次に(2)に対しては複数の敵機を同時に生成・動作させること、それぞれの敵機が弾を発射できること、動作にバリエーションを持たせること、という３つの条件を課した。複数の敵機は配列と for 文を用いたメソッドによって管理できるようにした。当初は処理が重くならならないかという懸念があったが、プログラムを整理して各種メソッドの呼び出しを

Timer によって統合することで､問題なく動作するようになった｡敵機の弾の発射は Enemy クラスの shotcount 変数によって一定の時間間隔で行われるようにした。これについての処理でも配列や for 文を使用している。敵機のバリエーションについては、Enemy クラスに type 変数を設けることで複数種類の動きを実現した。まだ動作の種類は少ないが、追加の実装は比較的容易になっている。

次に(3)については、自機が発射した弾と敵機が発射した弾をどうやって区別するかが問題となった。これは Enemy クラスと同じく type 変数を使用し、また jbullet, ebullet のように 2 種類のオブジェクトを設けることで解決している。改善点としては、JBullet クラスと EBullet クラスという 2 つに分かれたクラスはほぼ同じ形をしているため、統合することでよりすっきりしたプログラムになると考えられる。また、弾の動きが単調であるため、三角関数を用いるなどしてより面白い動きを実装することが今後の課題となる。

次に(4)を実装するにあたって、2 つの物体が衝突しているかどうかを判定するメソッド、衝突した後の（消滅する等の）処理を作るメソッドが必要になった。これらはどちらも配列と for 文によって実現している。(2)と同様に処理が重くなることが懸念されていたが、今のところ問題なく動作している。敵機を増やした際にどのようになるかは今後試してみる必要がある。また、当たり判定は 2 つの四角形が重なり合っているかどうかで判定する。しかし自機も敵機も弾も四角形ではないため、判定に違和感が出る可能性がある。小さい物体同士の当たり判定であればそれほど大きな違和感にはならないと考えるが、判定の仕方の改善は今後の課題となる。

そして(5)についてはシーン遷移のための TitleFreme クラスを設けることによって実現した。タイトル画面とゲームオーバー画面のほかに、ゲームクリア画面も実装した。タイトル画面からはキー入力でゲーム画面に移動する。ゲーム画面からは一定条件を満たすとゲームオーバー画面かゲームクリア画面に移動する。その条件は Timer から呼び出されるメソッドによって管理されている。ゲームオーバー画面、ゲームクリア画面からは一定時間たつとタイトル画面に戻るようになっている。改善点としてはタイトル画面で「はじめる」「やめる」等の選択肢が選べるようにすることが考えられる。また当初は自機の操作のための KeyLisner を CharactorControler クラスで管理する予定だったが、途中でこの TitleFrame クラスに移動させることになった。このときの名残が CharactorControler クラスに残っているため、これを整理する必要がある。

このほか、残った時間で自機のライフを導入した。最初は変数を 1 つ追加して当たり判定のメソッドに加え、ゲームオーバーを判定する際に参照するようにしていた。ここからさらに自機の残りライフを視覚的にわかりやすくするために、Figure クラスを継承した専用のクラスを設けた。

最後に全体を振り返ると、まずは予定していた必要最低限度の要素の実装には成功したと考えられる。またそこからさらに自機のライフなどの追加要素まで加えることができた。ただし敵や弾の動きのパターンなどはまだまだ少なく、ゲームのステージは一つも完成していない。これらの要素を拡張することで、よりクオリティの高いゲームを作ることができる。全体の改善点としては、各クラスの細分化をはじめとしたプログラムの整理が挙げ

られる。自分たちでプログラムを一から作るという方針のもとで今回のプログラムを作成したことで、かなり行き当たりばったりの作業になった。途中で追加されたり変更されたりしたプログラムも多い。その結果として、長くて見づらいクラスや非効率的なプログラムもできている。今後はゲームの要素の拡張とともにこれらの整理を行うことが必要になる。

文責：江村

6.感想

6.1 飯野

まずグループ全体としては初期構想通り既存のプログラムに頼ることなく一からプログラムを組み立て、一応の形としてのゲームを完成させることが出来たのでその部分では満足しています。しかし、細かい部分を見ていくとあまり上手くいかなかったところも多く、一番大きなこととしては、段取りが悪かったと感じています。ある作業を行い始めるまでが長かったり、作業が一段落してからも次に取り掛かるのがスムーズに行かなかったりするなどいちいち作業がとまっている時間が長いように感じました。その結果として最後になって時間が押してしまったので、一応形になったとはいえ簡素なつくりになってしまい、ゲームボリュームとしても物足りないものになってしまったと思います。作業を始める前の構想の段階でもっと具体的な話を多めにして、明確な指標をもって作業を行うようにすればより円滑に作業が進み、時間に余裕もできてもっと完成度を高められたのではないかと思います。
また私個人としては、非常に要領悪く作業を行っておりグループメンバーに頼る部分が多くなってしまい、とても迷惑をかけたと思っています。私個人のスキルアップでも上の状況が少しは改善したのではと考えると、とても申し訳ないと思います。
この講義自体は上手くいけばここで終わってしまいますが、プログラミングは３年次以降も必要になることで、情報系の道では将来でも必要だと思うので何不自由なく自分の手足のようにプログラミングを動かせるようになれるよう精進していきたいと思います。

文責：飯野

6.2 佐藤

(1)グループでの作業を通しての感想

グループ内での作業の感想として、全体的にあまり作業がはかどらなかったというこ

とである。最初、シューティングゲームを作ろうということまではすぐ決まったのだが、そこから実際にプログラムを作り出すまでに非常に時間がかかってしまった。また、プログラム書き始めても誰がどこを担当するかが大まかにしか決めていなかったため、自分の担当分が明確でないまま作業を進めてしまっていた。なので、大きなプログラムを書く際は、まず全体的にどのような構成のプログラムにするか紙に書き出し、誰がどこを担当するか明確にした上で、そのプログラムのどこまでをいつまでに終わらせるか期限をしっかり設ける必要性があったなと感じた。そうすれば自分が何をやればいいか明確になり、作業にも取り掛かりやすかったのではないかと感じている。

また、ゲーム自体の反省点としては敵の種類や敵弾と自機弾の種類をもっと増やせればよかったなと感じている。具体的には自機の弾としてビームを出したり、敵弾の動きのパターンを多くしたり、敵として最後にボスを設定するなどができればさらに面白いゲームになったのではないかとかに感じている。

(2)今後の各自の担当分の課題とやり残したこと

自分の担当分において、課題として感じていることは、プログラムが非常に長くなってしまったということである。プログラム自体が非常に長くなり、それらを一つクラスにしてしまったため、読みにくいプログラムとなってしまった。よって今後自分の課題としてはJavaでプログラムを書くときはプログラムをできるだけ多くのクラスに細分化して書くように工夫したいと思った。また、機能として、経過時間やスコア、自機の残機を分かりやすく表示できればよかったなと感じた。

(3)「プログラミング演習」に関する感想と学習内容に対する感想

個人的にはJavaに関してもっと色々なことを知りたかったと感じている。具体的にはもっと色々なパッケージやメソッドなどの説明や、ネットワークプログラミングの説明などである。また、MVCモデルに関してもちょっと難しく、最初はよくわからなかったのでもう少し詳しく説明してもらえるとありがたかったかなと感じた。なので、難しいとは思うが もっと時間数と単位数を増やしてくれると嬉しいと思った。

[文責:佐藤]

6.3 江村

グループ全体としてはまず分担や段取りが甘かったと感じている。それは、それほど経験がないメンバーで既存のサンプルになるべく頼らずに一からプログラムを作成しようとしたことも原因の一つであると考えている。ゲームの全体の構成から個々の要素までノウハウがほとんどない状態では、ある部分を作るときに分担やどのようなものが必要かその時になるまではっきりとせず、結果的にかなり行き当たりばったりの作業となった。拡張性は持たせたものの、時間内で最低限のものしか作ることができず、プログラムも非効率的なものが目立つ。ほぼ完全に一から作ることによって得た経験もあるが、よりクオリテ

ィの高いゲームを作るならば、もっとサンプルを見て参考にしたり一部や大部分を流用したりすることが必要だっただろう。実際のプログラミングでもこういったことは普通に行われているはずなので、今後はそれを視野に入れて作成していこうと思う。

そして自分の作成した部分は主にデータを管理するクラスである。Figure クラスを親クラスとして Jiki, Enemy, JBulet, EBullet, Life クラスを作成したが、行き当たりばったりで変数を追加した結果、親クラスの存在意義がほとんど形骸化してしまったようにも感じている。またゲームの全データを管理する Model クラスではメソッドを次々と追加して長く見づらいものになっていった。よって今後の課題としては、親クラスを作る際に必要になるものをあらかじめ想定しておくこと、役割によってクラスをより細分化すること、プログラム追加する際に既存のものを適宜整理していくことが挙げられる。また自分は一つ一つの作業に取り掛かるのが遅く、全体に迷惑をかけてしまったことは反省したい。

以上を踏まえてこのプログラミング演習では、自分にとって初めて触れる Java 言語の特性についていろいろと学べたほか、ある程度の規模のプログラムの作成やグループでの作業について自分の課題を洗い出すことができたという点で、とても有意義だったと思う。今後はこれらの改善点を踏まえて勉強を続けたい。また MVC モデルに関してはまだ理解が不十分であると感じている。特に Observer に関してはいまだによく理解できていない。そのため今回のプログラムの作成でもこれは導入しなかった。導入が必要かどうかも実は不明である。このような点からも、もっと勉強が必要だと思った。

文責：江村

付録 1、操作法マニュアル

ゲームは shooting.jar ファイルにまとめられている。起動するとタイトル画面が表示される。

図 1、タイトル画面

タイトル画面での操作：Enter キー　：ゲーム画面に移行

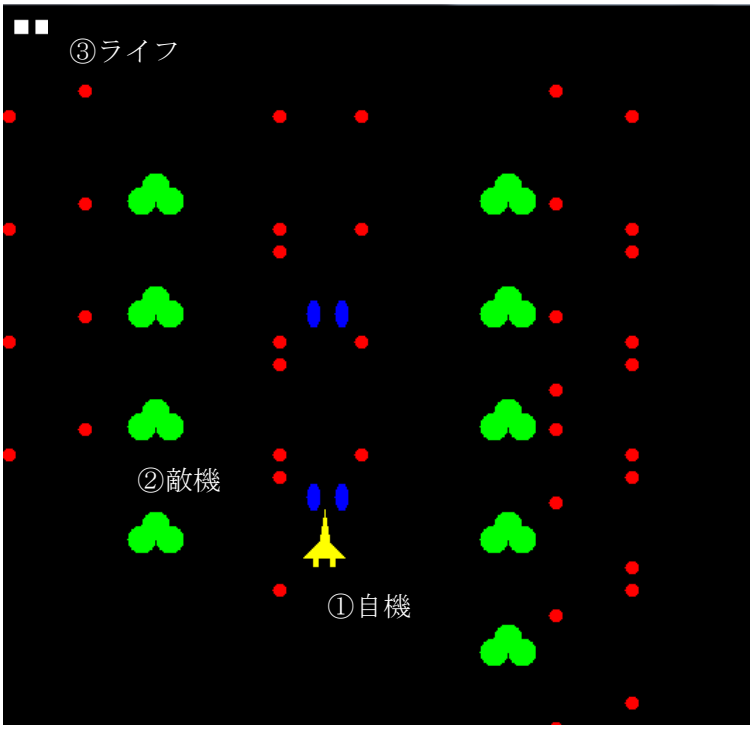


図 2 、ゲーム画面

画面上青い楕円が自機の弾、赤い円が敵機の弾

自機の弾を敵機に当てると該当敵機消滅、敵機の弾が自機に触れるとライフ 1 減少

ゲーム画面での操作：矢印キー　：自機移動

矢印キーを入力した方向に自機が移動する。

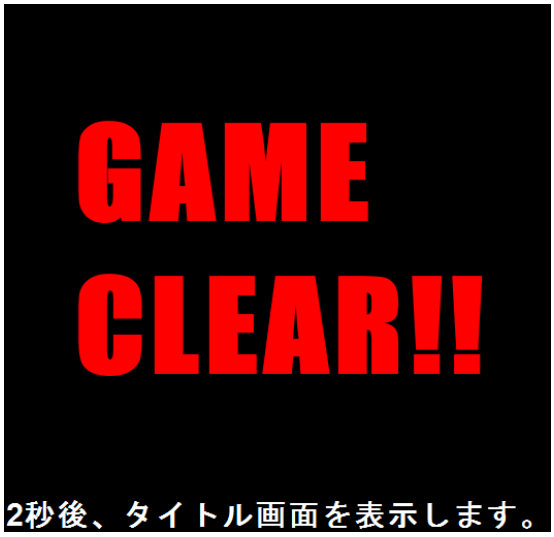
Z キー　　：自機弾発射

自機から上方向に弾を発射する。

ライフが０になるとゲームオーバー画面に移行、ライフを保ったまま一定時間経過でゲームクリア画面に移行

図３、ゲームオーバー画面

一定時間経過後自動的にタイトル画面に移動する。



一定時間経過後自動的にタイトル画面に移行する。

文責：飯野

付録 2:プログラムリスト

1.Model.java

```
1  import java.awt.*;
2  import java.awt.event.*;
3  import javax.swing.*;
4  import java.util.*;
5
6  //MVC の M の部分をつくる
7
8  /*Figure クラス*/
9  class Figure {
10       protected int x,y,width,height,speed;          //ローカル変数定義
11       protected Color color;
12
13       public Figure(int x,int y,int w,int h,int s,Color c) {
14            this.x = x; this.y = y;   // this.x, this.y はフィールド変数を指します.
15            width = w; height = h;    // ローカル変数で同名の変数がある場合は，this
16            speed = s; color = c;     // を付けると，フィールド変数を指すことになります.
17       }
18
19
20       /*draw メソッド定義*/
21       public void draw(Graphics g) {}
22 }
23
24 class Life extends Figure{ //ライフ
25       protected int count;
26
27       public Life(int x,int y, int l){
28            super(x,y,10,10,0,Color.white);
29            count = l;
30       }
31
32       public void draw(Graphics g){
33            g.setColor(color);
34            for(int i=0;i<count;i++)
35                 g.fillRect(x+i*15,y,width,height);
```

```
36        }
37  }
38
39  class Jiki extends Figure{ //自機クラス
40        public Jiki(int x,int y,int w,int h,int s,Color c){
41              super(x,y,w,h,s,c); //親コンストラクタの呼び出し
42        }
43
44
45        /*自機の描画*/
46        public void draw(Graphics g){
47              g.setColor(color);
48              int xPoints[] = {x,x-width/8,x-width/2,x+width/2,x+width/8};
49              int yPoints[] = {y-height/2,y,y+height/4,y+height/4,y};
50              g.fillPolygon(xPoints, yPoints,5);
51
52              g.fillRect(x+width/8,y+height/4,width/8,height/8);
53              g.fillRect(x-width/4,y+height/4,width/8,height/8);
54        }
55  }
56
57
58
59  class Enemy extends Figure{ //敵機クラス
60        /*フィールド*/
61        protected int type;
62        protected boolean dflag; //消滅フラグ
63        protected int shotcount;
64
65        /*コンストラクタ*/
66        public Enemy(int x,int y,int w,int h,int s,int t,Color c){
67              super(x,y,w,h,s,c);        //親クラス呼び
68              type = t; //敵のタイプ
69              dflag = false;
70              shotcount = 0;
71        }
72
73        /*敵機の描画*/
74        public void draw(Graphics g){
75              g.setColor(color);
```

```
76              g.fillOval(x,y,width,height);
77              g.fillOval(x+width/2,y+height/2,width,height);
78              g.fillOval(x-width/2,y+height/2,width,height);
79          }
80
81  }
82
83  class JBullet extends Figure{ //弾クラス
84          /*フィールド*/
85          protected boolean dflag;   //消滅フラグ
86          protected int type;     //敵弾の種類
87
88          /*コンストラクタ*/
89          public JBullet(int x,int y,int t){
90              super(x-3,y-3,10,20,13,Color.blue);
91              type = t;
92              dflag = false;
93          }
94
95          /*弾の描画*/
96          public void draw(Graphics g){
97              g.setColor(color);
98              g.fillOval(x,y,width,height);
99          }
100 }
101
102
103 class EBullet extends Figure{ //弾クラス
104         /*フィールド*/
105         protected boolean dflag;   //消滅フラグ
106         protected int type;     //敵弾の種類
107
108         /*コンストラクタ*/
109         public EBullet(int x,int y,int t){
110             super(x-3,y-3,10,10,4,Color.red);
111             type = t;
112             dflag = false;
113         }
114
115         /*弾の描画*/
```

```
116         public void draw(Graphics g){
117             g.setColor(color);
118             g.fillOval(x,y,width,height);
119         }
120 }
121
122
123 /*Model クラス*/
124 class Model {
125         protected Jiki jiki;                    //自機
126         protected Life life;                    //自機のライフ
127         protected ArrayList<Enemy> enemy;   //敵機格納用の配列
128         protected ArrayList<JBullet> jbullet; //自機弾格納用の配列
129         protected ArrayList<EBullet> ebullet; //敵機弾格納用の配列
130         protected Enemy drawEnemy;
131         protected boolean gameover;
132
133         /*オブジェクト*/
134         public Model(){
135             jiki = new Jiki(200,200,32,48,10,Color.yellow);     //自機の設定
136             life = new Life(10,10,3);               //ライフ 3 からスタート
137             /*配列格納*/
138             enemy = new ArrayList<Enemy>();         //Enemy 型の配列(敵)
139             jbullet = new ArrayList<JBullet>();     //Bullet 型の配列(自機弾)
140             ebullet = new ArrayList<EBullet>();     //Bullet 型の配列(敵機弾)
141             drawEnemy = null;
142         }
143
144         /*自機関連のメソッド*/
145         /*自機の draw メソッド*/
146         public void drawJiki(Graphics g){
147             jiki.draw(g);
148         }
149         /*自機の移動*/
150         public void JikiMove(boolean r,boolean l,boolean d,boolean u){
151             if(r==true && jiki.x <= 540) jiki.x += jiki.speed;
152             if(l==true && jiki.x >= 10) jiki.x -= jiki.speed;
153             if(d==true && jiki.y <= 530) jiki.y += jiki.speed;
154             if(u==true && jiki.y >= 10) jiki.y -= jiki.speed;
155         }
```

```
156     public int xJiki(){return jiki.x;} //自機の中心 x 座標を返す
157     public int yJiki(){return jiki.y;} //自機の中心 y 座標を返す
158     public int wJiki(){return jiki.width;} //自機の横幅を返す
159
160     /*敵機関連のメソッド*/
161     /*敵を記録している配列を返す関数*/
162     public ArrayList<Enemy> getEnemys() {
163         return enemy;
164     }
165
166     /*何番目かを指定して敵を取り出すメソッド*/
167     public Enemy getEnemy(int idx) {
168         return enemy.get(idx);
169     }
170
171     public int xEnemy(int idx) {
172         return enemy.get(idx).x; //敵機の中心 x 座標を返す
173     }
174     public int yEnemy(int idx) {
175         return enemy.get(idx).y; //敵機の中心 y 座標を返す
176     }
177
178     /*敵機の生成*/
179     public void createEnemy(int x,int y,int type){
180         Enemy e = new Enemy(x,y,20,20,5,type,Color.green);   //敵を生成して
181         enemy.add(e);                                        //配列に格納して
182     }
183
184     /*敵機の動作の管理*/
185     public void EnemyMove(){
186         for(int i=0; i<enemy.size(); i++){
187             switch(enemy.get(i).type){
188
189             case 1:                    //敵のタイプが 1 のとき
190                 enemy.get(i).y+=enemy.get(i).speed+1;          //敵機の y 座標を増やしていく
191                 enemy.get(i).shotcount++;                      //shotcount も増やす
192
193                 if(enemy.get(i).shotcount%50==0||enemy.get(i).shotcount==1)
194                     //shotcount が 50 で割り切れるようになった時、敵機の場所に弾生成
195                     {
```

```
196                         this.createBullet(this.xEnemy(i),this.yEnemy(i),10);
197                         this.createBullet(this.xEnemy(i),this.yEnemy(i),11);
198                         this.createBullet(this.xEnemy(i),this.yEnemy(i),12);
199                     }
200                 break;
201
202             case 2:                 //敵のタイプが 2 のとき
203                 enemy.get(i).x+=enemy.get(i).speed;          //敵機の x 座標を増やしていく
204                 enemy.get(i).y+=enemy.get(i).speed*2;          //敵機の y 座標も増やしていく
205
206                 enemy.get(i).shotcount++;                    //shotcount を増やす
207                 if(enemy.get(i).shotcount%50==0||enemy.get(i).shotcount==1)
208                     //shotcount が 50 で割り切れるようになった時、敵機の場所に弾生成
209                     {
210                         this.createBullet(this.xEnemy(i),this.yEnemy(i),10);
211                         this.createBullet(this.xEnemy(i),this.yEnemy(i),12);
212                     }
213                 break;
214
215             case 3:                 //敵のタイプが 3 のとき
216                 enemy.get(i).x-=enemy.get(i).speed;          //敵機の x 座標を増やしていく
217                 enemy.get(i).y+=enemy.get(i).speed*2;          //敵機の y 座標も増やしていく
218
219                 enemy.get(i).shotcount++;                    //shotcount を増やす
220                 if(enemy.get(i).shotcount%50==0||enemy.get(i).shotcount==1)
221                     //shotcount が 50 で割り切れるようになった時、敵機の場所に弾生成
222                     {
223                         this.createBullet(this.xEnemy(i),this.yEnemy(i),10);
224                         this.createBullet(this.xEnemy(i),this.yEnemy(i),11);
225                     }
226                 break;
227
228             }
229         }
230     }
231
232     /*弾関連メソッド*/
233     /*弾を記録している配列を返す関数*/
234     public ArrayList<EBullet> getBullets() {
235         return ebullet;
```

```
236     }
237     public ArrayList<JBullet> getjBullets() {
238         return jbullet;
239     }
240
241     /*弾の動作の管理*/
242     public void BulletMove(){
243         for(int i=0; i<jbullet.size(); i++){
244             switch(jbullet.get(i).type){
245             case 1:
246                 jbullet.get(i).y -= jbullet.get(i).speed;
247                 break;
248
249             case 2:
250                 jbullet.get(i).y -= jbullet.get(i).speed;
251                 jbullet.get(i).x += jbullet.get(i).speed;
252                 break;
253
254             case 3:
255                 jbullet.get(i).y -= jbullet.get(i).speed;
256                 jbullet.get(i).x -= jbullet.get(i).speed;
257                 break;
258
259             }
260         }
261         for(int i=0; i<ebullet.size(); i++){
262             switch(ebullet.get(i).type){
263             case 10:
264                 ebullet.get(i).x += ebullet.get(i).speed;
265                 ebullet.get(i).y += ebullet.get(i).speed*2;
266                 break;
267
268             case 11:
269                 ebullet.get(i).x += ebullet.get(i).speed;
270                 ebullet.get(i).y += ebullet.get(i).speed;
271                 break;
272
273             case 12:
274                 ebullet.get(i).x -= ebullet.get(i).speed;
275                 ebullet.get(i).y += ebullet.get(i).speed;
```

```
276                     break;
277                 }
278             }
279         }
280
281
282         /*弾の生成*/
283         public void createBullet(int x, int y, int t){
284             switch(t){
285             case 1:
286                 JBullet jb1 = new JBullet(x,y,t);
287                 jbullet.add(jb1);
288                 break;
289
290             case 2:
291                 JBullet jb2 = new JBullet(x,y,t);
292                 jbullet.add(jb2);
293                 break;
294
295             case 3:
296                 JBullet jb3 = new JBullet(x,y,t);
297                 jbullet.add(jb3);
298                 break;
299
300
301
302             case 10:
303                 EBullet eb10 = new EBullet(x,y,t);
304                 ebullet.add(eb10);
305                 break;
306
307             case 11:
308                 EBullet eb11 = new EBullet(x,y,t);
309                 ebullet.add(eb11);
310                 break;
311
312             case 12:
313                 EBullet eb12 = new EBullet(x,y,t);
314                 ebullet.add(eb12);
315                 break;
```

```
316         }
317     }
318
319     /*その他*/
320     /*衝突時の処理*/
321     public void HitCheck(){
322         for(int i=0; i<enemy.size(); i++){
323             for(int j=0; j<jbullet.size(); j++){
324                 if(HitBox(enemy.get(i).x,enemy.get(i).y,
325                           enemy.get(i).width,enemy.get(i).height,
326                           jbullet.get(j).x,jbullet.get(j).y,
327                           jbullet.get(j).width,jbullet.get(j).height)
328                     == true){
329                     jbullet.get(j).dflag = true; //敵機に当たった自機弾の消滅フラグを立てる
330                     enemy.get(i).dflag = true; //自機弾に当たった敵機の消滅フラグを立てる
331                 }
332             }
333         }
334         for(int i=0; i<ebullet.size(); i++){
335             if(HitBox(jiki.x,jiki.y,
336                       jiki.width,jiki.height,
337                       ebullet.get(i).x,ebullet.get(i).y,
338                       ebullet.get(i).width,ebullet.get(i).height)
339                 == true){
340                 ebullet.get(i).dflag = true; //自機に当たった敵機弾の消滅フラグを立てる
341                 life.count--; //ライフの減少
342             }
343         }
344         for(int i=0; i<enemy.size(); i++){
345             if(HitBox(jiki.x,jiki.y,
346                       jiki.width,jiki.height,
347                       enemy.get(i).x,enemy.get(i).y,
348                       enemy.get(i).width,enemy.get(i).height)
349                 == true){
350                 enemy.get(i).dflag = true; //自機に当たった敵機の消滅フラグを立てる
351                 life.count--; //ライフの減少
352                 GameOver();
353             }
354         }
355     }
```

```
356
357     /*矩形当たり判定*/
358     public boolean HitBox(int x1,int y1,int w1,int h1,
359                                 int x2,int y2,int w2,int h2){
360         if((x1 < x2 + w2) && (x2 < x1 + w1) &&
361            (y1 < y2 + h2) && (y2 < y1 + h1))
362             return true;
363         else
364             return false;
365     }
366
367
368     /*消滅の処理*/
369     public void DeleteFigure(){
370         for(int i=enemy.size()-1; i>=0; i--)
371             if(enemy.get(i).dflag==true)
372                 enemy.remove(i);
373
374         for(int i=ebullet.size()-1; i>=0; i--)
375             if(ebullet.get(i).dflag==true)
376                 ebullet.remove(i);
377
378         for(int i=jbullet.size()-1; i>=0; i--)
379             if(jbullet.get(i).dflag==true||jbullet.get(i).y<=0)
380                 jbullet.remove(i);
381     }
382
383     /*ゲームオーバー*/
384     public boolean GameOver()
385     {
386         if(life.count < 1)
387             {
388                 return true;
389             }
390         else
391             return false;
392     }
393
394     /*ライフの初期化*/
395     public void LifeInit(){
```

```
396             life.count = 3;
397         }
398         /*ライフの描画*/
399         public void drawLife(Graphics g){
400             life.draw(g);
401         }
402 }
403
404
```

2.STG.java

```
 1  import java.awt.*;
 2  import java.awt.event.*;
 3  import javax.swing.*;
 4  import java.util.*;
 5
 6  /*ゲーム画面に関するクラス*/
 7  class CharactorController extends JPanel implements ActionListener
 8  {
 9      /*フィールド*/
10      private Model model;          //Model のデータを取り込む
11      private ArrayList<Enemy> enemy;   //敵格納配列定義
12      private ArrayList<EBullet> ebullet; //敵弾格納用配列定義
13      private ArrayList<JBullet> jbullet; //自機弾格納用配列定義
14      private javax.swing.Timer timer; //timer(敵機の動作に関する timer)
15      public int time =0;                  //timer を以下で time に換算。初期値は 0
16      protected int i = 0;
17
18      boolean jshotflag = false;
19      boolean jrflag = false;          //右ボタンに関するフラグ
20      boolean jlflag = false;          //左ボタンに関するフラグ
21      boolean jdflag = false;          //下ボタンに関するフラグ
22      boolean juflag = false;          //上ボタンに関するフラグ
23      boolean flag_timer = false;   //timer に関するフラグ
24      boolean gameover = false;      //gameover か否か判定するフラグ
25
26
27
28      /*コンストラクタ*/
```

```
29      public CharactorController()
30      {
31          timer = new javax.swing.Timer(50,this);      //timer に timer 挿入(0.05 秒おき)
32
33          model = new Model();  //model オブジェクトの生成
34
35          this.setBackground(Color.BLACK);  //背景設定
36          this.setFocusable(true);              //キー入力を GUI 部品が受け付ける
37
38      }
39
40      /*timer をスタートさせるメソッド*/
41      public void timerstart()
42      {
43          timer.start();  //timer をスタート
44      }
45
46      /*timer を止めるメソッド*/
47      public void timerstop()
48      {
49          timer.stop();
50      }
51
52
53      /*描画*/
54     protected void paintComponent(Graphics g)
55      {
56         super.paintComponent(g);
57
58         /*自機の描画*/
59         model.drawJiki(g);
60
61         /*ライフの描画*/
62         model.drawLife(g);
63
64         enemy = model.getEnemys();
65         /*敵機の描画*/
66          for(Enemy e:enemy){
67              e.draw(g);
```

```
68          }
69
70          ebullet = model.getBullets();
71          /*敵弾の描画*/
72          for(EBullet eb:ebullet){
73              eb.draw(g);
74          }
75
76          jbullet = model.getjBullets();
77          /*自機の弾の描画*/
78          for(JBullet jb:jbullet){
79              jb.draw(g);
80          }
81
82      }
83
84
85      /*timer に対する actionPerformed(敵の生成)*/
86      public void actionPerformed(ActionEvent e)
87      {
88          time++;                         //timer1 を 0.1 秒に換算
89
90          model.JikiMove(jrflag,jlflag,jdflag,juflag);
91          if(jshotflag==true && time%5==0)
92              {
93                  model.createBullet(model.xJiki()+model.wJiki()/3,model.yJiki(),1);   //自機弾の
発射
94                  model.createBullet(model.xJiki()-model.wJiki()/3,model.yJiki(),1);   //自機弾の
発射
95              }
96
97          Game();
98
99          /*敵機の出現*/
100         if(time == 10)    //1 秒たったら
101             {
102                 for(int i=-320;i<=0;i+=80)
103                     model.createEnemy(100,i,1);
104
```

```
105                     for(int i=-320;i<=0;i+=80)
106                         model.createEnemy(350,i,1);
107                 }
108             if(time == 150)
109                 {
110                     for(int i=-320;i<=0;i+=80)
111                         model.createEnemy(50,i,2);
112
113                     for(int i=-320;i<=0;i+=80)
114                         model.createEnemy(400,i,3);
115                 }
116             if(time == 200)
117                 for(int i=-160;i<=0;i+=80)
118                     model.createEnemy(300,i,1);
119             if(time == 230)
120                 for(int i=-160;i<=0;i+=80)
121                     model.createEnemy(150,i,1);
122             if(time == 260)
123                 for(int i=-160;i<=0;i+=80)
124                     model.createEnemy(350,i,3);
125             if(time == 290)
126                 for(int i=-160;i<=0;i+=80)
127                     model.createEnemy(100,i,2);
128             if(time == 360)
129                 for(int i=-160;i<=0;i+=80)
130                     model.createEnemy(270,i,1);
131
132
133
134
135             model.EnemyMove();
136             model.BulletMove();
137             model.HitCheck();
138             model.DeleteFigure();
139
140             repaint();                              //そして再描画
141
142         }
143
```

```
144     /*GameOver 画面に遷移するためのメソッド*/
145     public int Game()
146     {
147         if(model.GameOver() == true && i==0)
148             {
149                 i++;
150             }
151         else if(time == 400)
152             {
153                 i++;
154                 i++;
155             }
156         return i;
157     }
158
159 }
160
161
162
163
```

3.Title.java

```
 1 import javax.swing.*;
 2 import java.awt.*;
 3 import java.awt.event.*;
 4 import java.awt.CardLayout;
 5 import java.util.*;
 6
 7
 8 /*main クラス*/
 9 class TitleFrame extends JFrame implements KeyListener,ActionListener
10 {
11     /*フィールド*/
12     JPanel p1,p2,p3,p4,p5;      //パネル定義
13     CardLayout layout;      //cardlayout の定義
14     boolean boo = false;    //操作を変えるためのフラグ
15     boolean flag = false;
16     private CharactorController chara;    //ゲーム画面
```

```
17      int a = 20;
18      javax.swing.Timer timer;
19      private int time;
20
21      /*オブジェクト(フレーム内の設定)*/
22      public TitleFrame()
23      {
24      /*JPanel 設定*/
25      p1 = new JPanel();        //大本のパネル
26      p2 = new JPanel();      //１ページ目のパネル（タイトル画面）
27      p3 = new JPanel();      //２ページ目のパネル（ゲーム画面）
28      p4 = new JPanel();      //3 ページ目のパネル（ゲームオーバー画面）
29      p5 = new JPanel();      //4 ページ目のパネル（クリア画面）
30      chara = new CharactorController();    //2 ページ目に貼り付けるゲーム画面
31      timer = new javax.swing.Timer(1,this);
32      timer.start();
33
34      p2.setBackground(Color.BLACK);   //背景設定
35      p4.setBackground(Color.BLACK);   //背景設定
36      p5.setBackground(Color.BLACK);   //背景設定
37
38
39      /*CardLayout の設定*/
40      layout = new CardLayout();
41
42      /*KeyListener 追加*/
43      p1.addKeyListener(this);
44      p1.setFocusable(true);
45
46      /*p2,p4 を BorderLayout で設定*/
47      p2.setLayout(new BorderLayout());
48      p4.setLayout(new BorderLayout());
49      p5.setLayout(new BorderLayout());
50
51      /*p2 に JLabel を貼り付ける*/
52      JLabel l2 = new JLabel("<html>"
53                                      +"<span style='font-size:110pt;font-family:Impact,Charcoal;"
54                          +"color:yellow;'>"
55                          +"Shooting</span></html>",
56                          +JLabel.CENTER);
57
```

```
58
59      JLabel l3 = new JLabel("<html>"
60                      +"<span style='font-size:30pt;color:white;'>"
61                      +"十字キーで移動、z で弾発射。<br/>"
62                      +"<span style='font-size:30pt;color:white;'>"
63                      +"Enter でゲームスタート!!"
64                      +"</span></html>",
65                      +JLabel.CENTER);
66
67      p2.add(l2,BorderLayout.CENTER);
68      p2.add(l3,BorderLayout.SOUTH);
69
70      /*2 ページ目*/
71      p3.setLayout(new BorderLayout());
72      p3.add(chara,BorderLayout.CENTER);
73
74      /*p4 に JLabel を貼り付ける*/
75      JLabel l4 = new JLabel("<html>"
76                              +"<span style='font-size:90pt;font-family:Impact,Charcoal;"
77                      +"color:blue;'>"
78                      +"GAMEOVER</span></html>",
79                      +JLabel.CENTER);
80
81
82      JLabel l5 = new JLabel("<html>"
83                      +"<span style='font-size:30pt;color:white'>"
84                      +"2 秒後、タイトル画面を表示します。"
85                      +"</span></html>",
86                      +JLabel.CENTER);
87
88      p4.add(l4,BorderLayout.CENTER);
89      p4.add(l5,BorderLayout.SOUTH);
90
91      /*p4 に JLabel を貼り付ける*/
92      JLabel l6 = new JLabel("<html>"
93                              +"<span style='font-size:120pt;font-family:Impact,Charcoal;"
94                      +"color:red;'>"
95                      +"GAME<br/>CLEAR!!</span></html>",
96                      +JLabel.CENTER);
97
98
```

```
 99        JLabel l7 = new JLabel("<html>"
100                          +"<span style='font-size:30pt;color:white'>"
101                          +"2 秒後、タイトル画面を表示します。"
102                          +"</span></html>",
103                          +JLabel.CENTER);
104
105
106        p5.add(l6,BorderLayout.CENTER);
107        p5.add(l7,BorderLayout.SOUTH);
108
109        /*p1 を CardLayout で設定し、p1 にパネルを貼り付ける。*/
110        p1.setLayout(layout);
111        p1.add(p2,BorderLayout.CENTER);
112        p1.add(p3,BorderLayout.CENTER);
113        p1.add(p4,BorderLayout.CENTER);
114        p1.add(p5,BorderLayout.CENTER);
115        getContentPane().add(p1,BorderLayout.CENTER);                //これがないと表示されない
(はず)
116
117
118        /*Frame 設定*/
119        this.setSize(550,550);                                        //Frame は 550*550(仮)
120        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
121        this.setTitle("Shooting");
122        this.setVisible(true);
123        }
124
125        /*keyListener 内のメソッド、keyPressed の設定*/
126        public void keyPressed(KeyEvent e)
127        {
128
129        /*押されたボタンを読み取る*/
130        int k = e.getKeyCode();
131
132        /*押されたボタンが Enter キーかつフラグ boo が降りていたら*/
133        if(k == KeyEvent.VK_ENTER && boo ==false)
134             {
135             layout.next(p1);      //次のページへ移行し、
136             boo = true;           //フラグを有効にして操作をゲームの操作にする
137             chara.timerstart(); //timer をスタート
138             }
```

```
139
140     /*フラグ boo が立っていたら*/
141     if(boo == true)
142         {
143         if(k == KeyEvent.VK_RIGHT)              //右押すと
144             chara.jrflag = true;      //CharactorController クラスの jrflag が立つ
145
146         else if(k == KeyEvent.VK_LEFT)              //左押すと
147             chara.jlflag = true;       //CharactorController クラスの flflag が立つ
148
149         else if(k == KeyEvent.VK_UP)
150             chara.juflag = true;
151
152         else if(k == KeyEvent.VK_DOWN)
153             chara.jdflag = true;
154
155         else if(k == KeyEvent.VK_Z)
156             chara.jshotflag = true;
157         }
158     }
159
160
161     /*キーをはなした時の操作*/
162     public void keyReleased(KeyEvent e)
163     {
164     int k = e.getKeyCode();                  //キーの情報を取得
165     if(boo == true)                              //フラグが有効なら
166         {
167         switch(k)
168             {
169
170             case KeyEvent.VK_RIGHT:          //右ボタンを離すと jrflag 無効
171                 chara.jrflag = false;
172                 break;
173
174             case KeyEvent.VK_LEFT:             //左ボタンを離すと jlflag 無効
175                 chara.jlflag = false;
176                 break;
177
178             case KeyEvent.VK_UP:
179                 chara.juflag = false;
```

```
180                     break;
181
182                 case KeyEvent.VK_DOWN:
183                     chara.jdflag = false;
184                     break;
185
186                 case KeyEvent.VK_Z:
187                     chara.jshotflag = false;
188                     break;
189
190                 }
191             }
192
193         }
194
195             /*キーを押したときの操作(弾を出す)*/
196         public void keyTyped(KeyEvent e){}
197
198         public void actionPerformed(ActionEvent ev)
199         {
200         if(chara.i == 1)
201             {
202             layout.next(p1);
203             chara.i = 4;
204             }
205         if(chara.i == 2)
206             {
207             layout.next(p1);
208             layout.next(p1);
209             chara.i = 4;
210             }
211         if(chara.i == 4)
212             {
213             time++;
214             if(time == 2000)
215                 {
216                 new TitleFrame();
217                 timer.stop();
218                 time = 0;
219                 setVisible(false);
220                 }
```

```
221             }
222         }
223
224         /*main メソッド(Frame 生成のみ)*/
225         public static void main(String argv[])
226         {
227         new TitleFrame();
228         }
229 }
```

[文責：佐藤]