情報システム工学実験II
後半フルレポート

2022531033　関川謙人

2024 年 2 月 12 日

目次

1　背景

1.1　経緯

元々関数電卓や数字パズルゲームを作ることを画策していたのだが電卓は数式の表示方法やデザイン性を考えたときにいい物が思いつかなかった。またパズルゲームはデザインやゲーム性に独創性を見いだせずアイデアに行き詰まる。

一旦諦めて友人と話していた時に東方の話になり、高校時代に東方風神録をプレイしていたことを思い出した。そこで ZUN 氏のアイデアを借りて東〇風シューティングゲームを制作することを思いつき、制作するに至った。

1.2　制作目的

本アプリの制作目的は以下のとおりである。

- 知っているシューティングゲームの挙動を一通り実装できるようになる。
- ゲーム制作の基礎を体感し、ゲームプログラミングの基礎を理解する。
- 試行錯誤を行うことでゲームクリエイターの創造性を理解する。

1.3　本実験の目的

Processing という一つのツールを使えるようになり、自身の独創性を発揮する手段として用いる。このことで新しい開発プラットフォームを使いこなせるようにする学習能力を身に着けることが本実験の目的である。

2　実験方法

2.1　実験装置

本実験における実験や開発は、VisualStudioCode(以下 VSCode) 上で行った。拡張機能は以下のものを入れている。

- Processing VSCode
- Processing Formatter

Processing VSCode に Processing の実行ファイル、Processing.java を実行させる。こうすることで図 1 のように VSCode 上でコーデング、ファイル実行が行える環境を構築できる。

また Processing Formatter で Processing コードの整形を行った。

自機や敵機の画像は clip studio paint で制作した。

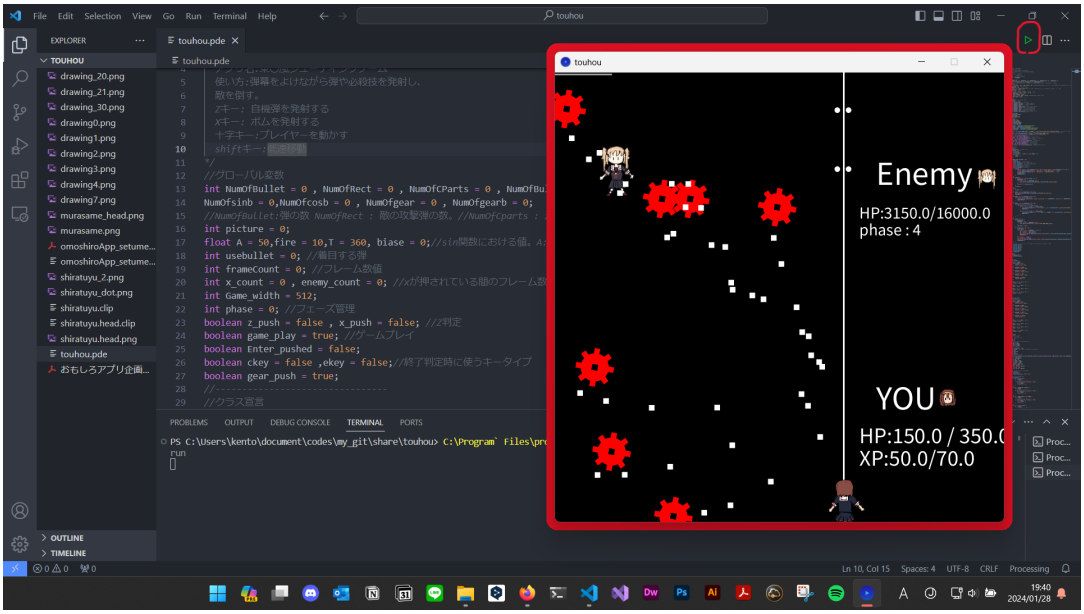

図 1: 開発環境

2.2 開発計画

開発の手順を計画段階と実際の手順とで比べると、以下の表のようになる。

計画段階	実際の開発
1. 自機を定義 2. 敵機を定義 3. 通常攻撃を定義 4. 必殺技を定義 5. 当たり判定を定義 6. 各種パラメータを定義	1. 自機の動作を定義 2. 自機の攻撃を定義 3. 敵機の当たり判定を定義 4. 敵機の動作を定義 5. 敵機の攻撃を定義 6. 自機の当たり判定を定義 7. 自機、敵機のパラメータを定義 8. 5,6 を繰り返す 9. 自機、敵機のグラフィックを定義 10. ボム (自機の必殺技) を定義 11. フェーズ、ゲーム終了等のシステム面を定義 12. 歯車型の敵を定義。 13. 5,6 を行い、最終フェーズを構築

3 アプリの概要

このゲームのコンセプトは敵の弾を避けながらプレイヤーの必殺技を駆使して敵を倒すことにある。この章では、完成したアプリについて以下の観点から説明する。

- システム・仕様
- 自機の動作
- 敵機の動作

また各処理の実装に用いたソースコードを一部省略して掲載している。

3.1 システム・仕様

3 節を説明するのに使う用語の定義は以下の通り。

用語	定義
自機	プレイヤーが操作するオブジェクト
敵	後述する Enemy クラスで定義されるオブジェクト全般
敵機	プレイヤーの討伐目標。
必殺技	敵の弾幕あるいはプレイヤーの攻撃手段
フェーズ	ゲームのプレイ段階

このゲームの仕様は以下のとおりである。

- 実行した際、スタート画面が表示される。
- スタート画面にて c キーでゲームスタート
- 敵機の HP が 0 になったら勝ち
- 自機の HP が 0 になったら負け
- 敵の攻撃は 4 通り存在する
- 敵の攻撃の種類は敵の残り HP によって決まる
- 勝敗が決まると終了判定画面に移行する。
- 終了判定画面にて e キーでゲーム終了、c キーでもう一度プレイする
- もう一度プレイする際、すべての変数がリセットされる。

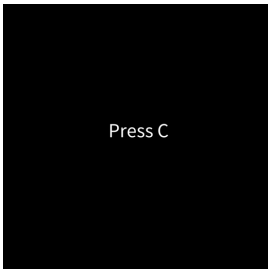

図 2: スタート画面

図 3: 終了判定画面 (勝利時)

3.1.1 使用した変数・関数

システムの構築に使用した変数は以下のとおりである

型	変数・関数	説明
int	Game_width	ゲーム画面の横幅。512 に設定。
int	phase	フェーズを表す変数
int	frameCount	フレーム数
boolean	game_play	ゲームプレイ画面かどうか
boolean	ckey	c キーが押されているかどうか
boolean	ekey	e キーが押されているかどうか
void	status()	ステータス画面を表示する関数
void	judge()	勝敗の判定を行い、結果を表示する関数
void	continue_judge()	継続判定を行う関数
void	judge_phase()	敵の HP を基にフェーズ判定を行う関数

3.1.2 ソースコード

システム面は以下のソースコードで管理している。敵機や自機、必殺技に関連する変数が含まれているがそれに関しては後述の節で説明する。

```
void setup(){
    //ウィンドウサイズ(800*800)
    size(800,800);
}
void draw() {
  background(0); //画面の初期化

  //スタート画面
  if (phase == 0) {
      color(255);
      textSize(60);
      text("Press C" , width / 2 - 80, height / 2);
      //Cキーを押してスタート
      if (ckey) {
          phase = 1;
      }
  }
  if (game_play && phase > 0) {
      status();
        judge_phase(); //フェーズ管理
    }
    judge();
  if (!game_play) {
      continue_judge();
  }
    frameCount++;
}
    //ゲーム状況を表示
void status() {
    strokeWeight(3);
    stroke(0,0,400);
```

```
32      fill(0,0,400);
33      line(Game_width + 2,0,Game_width + 2,height);
34      textSize(60);
35      text("YOU",Game_width + 60, height * 3 / 4);
36      text("Enemy",Game_width + 60,height / 4);
37      image(shiratuyu_head, width - 120, height * 3 / 4 - 40 ,40,40);
38      textSize(40);
39      text("HP:" + player.HP + "␣/␣" + player.Max_HP, Game_width + 30 , height * 3 / 4 + 60);
40      text("XP:" + player.XP + "/" + player.Max_XP, Game_width + 30 , height * 3 / 4 + 100);
41      textSize(30);
42      text("HP:" + enemy.HP + "/" + enemy.Max_HP , Game_width + 30, height / 4 + 60);
43      text("phase␣:␣" + phase , Game_width + 30, height / 4 + 90);
44      image(murasame_head, width - 50 , height / 4 - 30,40,40);
45  }
46  //勝敗の判定
47  void judge() {
48      textSize(60);
49      fill(0,0,400);
50      if (enemy.HP <= 0) {
51          text("YOU␣WIN!!",Game_width / 2, height / 2 - 50);
52          game_play = false;
53      }
54      else if (player.HP <= 0) {
55          text("GAME␣OVER",Game_width / 2, height / 2 - 50);
56          game_play = false;
57      }
58  }
59  //継続判定
60  void continue_judge() {
61      if (game_play == false) {
62          textSize(30);
63          text("C:Continue",width / 2 - 65, height / 2);
64          text("E:Exit",width / 2 - 65, height / 2 + 30);
65          if (keyPressed) {
66              if (ckey) {
67                  //初期化して再開
68                  game_play = true;
69                  player.x = Game_width / 2;
70                  player.y = height - 100;
71                  enemy.x = Game_width / 2;
72                  enemy.y = height / 4;
73                  player.HP = player.Max_HP;
74                  player.XP = player.Max_XP;
75                  enemy.HP = enemy.Max_HP;
76                  if (NumOfBullet > 0) {
77                      for (int i = NumOfBullet - 1; i >= 0; i--) {
78                          bullets.remove(i);
79                      }
80                  }
81                  if (NumOfBullet2 > 0) {
82                      for (int i = NumOfBullet2 - 1; i >= 0; i--) {
83                          bullet2.remove(i);
84                      }
85                  }
86                  if (NumOfRect > 0) {
87                      for (int i = NumOfRect - 1; i >= 0; i--) {
88                          Rect_attack.remove(i);
89                      }
90                  }
91                  if (NumOfCParts > 0) {
92                      for (int i = NumOfCParts - 1; i >= 0; i--) {
93                          Cparts.remove(i);
94                      }
95                  }
96                  if (NumOfsinb > 0) {
97                      for (int i = NumOfsinb - 1; i >= 0; i--) {
98                          sinb.remove(i);
99                      }
100                 }
101                 if (NumOfcosb > 0) {
102                     for (int i = NumOfcosb - 1; i >= 0; i--) {
103                         cosb.remove(i);
104                     }
```

```
105                 }
106                 if(NumOfgear > 0){
107                     for(int i = NumOfgear - 1; i >= 0; i--){
108                         gear.remove(i);
109                     }
110                 }
111                 if(NumOfgearb > 0){
112                     for(int i = NumOfgearb - 1; i >= 0; i--){
113                         gear_bullet.remove(i);
114                     }
115                 }
116                 NumOfBullet = 0;
117                 NumOfBullet2 = 0;
118                 NumOfRect = 0;
119                 NumOfCParts = 0;
120                 NumOfsinb = 0;
121                 NumOfcosb = 0;
122                 NumOfbomb = 0;
123                 NumOfgear = 0;
124                 NumOfgearb = 0;
125                 gear_push = true;
126                 usebullet = 0;
127                 frameCount = 0;
128                 phase = 1;
129             }
130             if (ekey) {
131                 exit();
132             }
133         }
134     }
135 }
136 void judge_phase() {
137     if (phase != 0) {
138         if (enemy.HP > enemy.Max_HP * 11 / 13) {
139             phase = 1;
140         }
141         else if (enemy.HP > enemy.Max_HP * 7 / 13) {
142             phase = 2;
143         }
144         else if (enemy.HP > enemy.Max_HP * 3 / 13) {
145             phase = 3;
146         }
147         else if (enemy.HP <= enemy.Max_HP * 3 / 13) {
148             phase = 4;
149         }
150     }
151 }
```

3.1.3 処理の説明

背景色は黒、ウィンドウサイズは縦 800、横 800 である。status() 関数では画面をゲームプレイ画面とステータス画面に分割する。そしてステータス画面に以下を表示する。

- 敵機の HP
- フェーズ
- 自機の HP
- 自機の XP

judge() 関数では敵機、自機どちらの HP が 0 であるかを判定する。自機の HP が 0 であれば”GAME OVER” と表示し、敵機の HP が 0 であれば”YOU WIN!!” と表示する。

仕様をわかりやすく示したのが以下の図である。

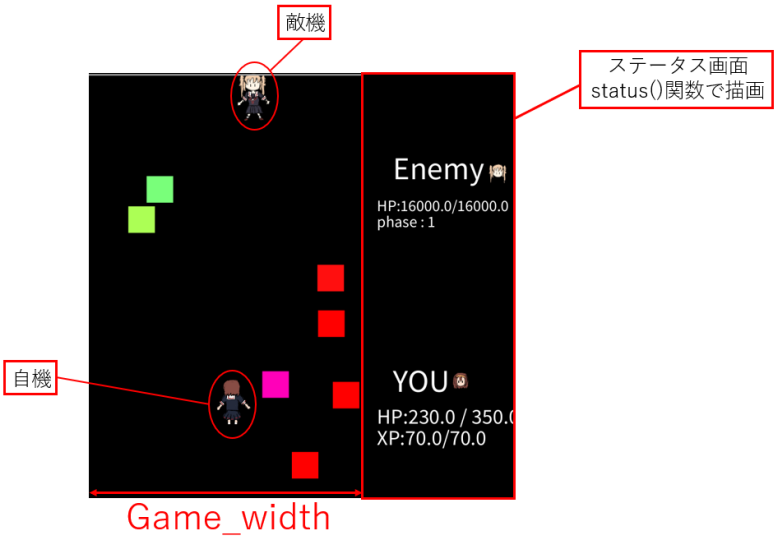

図 4: プレイ画面

judge_phase() 関数で制御されるフェーズの移行条件は以下の表のとおりである

フェーズ	移行条件
0	スタート画面
1	スタート画面または終了判定画面で c を押す
2	敵の HP が最大 HP の 11/13
3	敵の HP が最大 HP の 7/13
4	敵の HP が最大 HP の 3/13

各フェーズの説明については 3.3 項で細かく説明する。

3.1.4 必殺技の基本定義

必殺技の定義ついてもここで説明する。定義には Weapon クラスを使用している。

全ての必殺技は ArrayList<Weapon>という形で定義している。これによって Weapon クラスを基にしたオブジェクトの動的配列で必殺技を制御できる。以下の表はこのプログラムに使用した ArrayList のコマンドの一覧である

コマンド	説明
object = ArrayList.<class>	オブジェクトの動的配列の定義
object.add	オブジェクトを追加
object.remove(i)	i 番目のオブジェクトを削除
object.get(i)	i 番目のオブジェクトを参照

■3.1.4.1　**使用した関数・変数**　Weapon クラス内のフィールド（変数）は以下の表のとおりである。

型	変数・関数	説明
class	Weapon	必殺技を定義するクラス
float	x	弾の x 座標
float	y	弾の y 座標
float	speed	弾の速度
float	atk	弾の攻撃力
float	vecX	弾の x 進行方向
float	vecY	弾の y 進行方向

■3.1.4.2　**ソースコード**　以下に、Weapon クラスを定義するソースコードを示す。

1: ソースコード 3.1.4.2

```
    class Weapon{
    //フィールドの宣言
    float x;
    float y;
    float speed;
    float atk;
    float XP;
    float vecX;
    float vecY;
    //Weaponクラスのコンストラクタ
    Weapon(float xpos ,float ypos,float sped, float attack,float XP,float vecX,float vecY) {
        x = xpos;
        y = ypos;
        speed = sped;
        atk = attack;
        this.XP = XP;
        this.vecX = cos(radians(vecX));
        this.vecY = sin(radians(vecY));
    }
}
```

以降、Weapon クラス内のフィールド、コンストラクタの記述は省略する。

3.2 自機の動作

この項目では自機の動作とプログラムについて

1. 自機の基本動作
2. 自機の基本技
3. ボム

の三つの観点から説明する。

3.2.1 自機の基本動作

■3.2.1.1 使用した関数・変数 自機の基本動作を定義する際使用した変数は以下の表のとおり。

表 1: プレイヤー定義に使用した変数

型	変数・関数	説明
class	Player{}	自機を定義するクラス
Player	player	プレイヤーを表すオブジェクト
float	x	プレイヤーの x 座標
float	y	プレイヤーの y 座標
float	speed	プレーヤーの移動速度
float	HP	プレイヤーの HP
float	Max_HP	プレイヤーの最大 HP
float	XP	プレイヤーの XP。ボムの項目でも説明する。
float	Max_XP	プレイヤーの最大 XP
boolean	is_up	上移動するかどうか
boolean	is_down	下移動するかどうか
boolean	is_right	右移動するかどうか
boolean	is_left	左移動するかどうか
float	speed	プレイヤーの移動スピード
void	movement()	プレイヤーの移動を制御
void	display()	プレイヤーを表示する関数
なし	shiratuyu	自機の画像

■3.2.1.2 パラメータ 自機のパラメータを以下の表に示す。

パラメータ	値
初期 x 座標	Game_width / 2
初期 y 座標	height / 3
スピード (通常移動)	5
スピード (低速移動)	2
最大 HP	350
最大 XP	70

■3.2.1.3　**ソースコード**　以下に、プレイヤーの動作定義に使ったソースコードを示す。

```
    void setup(){
        shiratuyu = loadImage("shiratuyu_dot.png");
        //クラス定義
    player = new Player(Game_width / 2,height / 3,false,false,false,false,5,350,350,70,70); //
      playerの初期化。
    //Player(x初期値,y初期値,上キー,下キー,右キー,左キー,プレイヤースピード,HP,Max_HP,XP,Max_XP)
    }
    //繰り返し処理。
void draw() {
    if (game_play && phase > 0) {
        player.display(); //プレイヤー表示
        player.movement(); //プレイヤーを動かす。
    }
}
    void keyPressed() {
    //プレイヤーの動作
    if (key == CODED) {
        if (keyCode == UP) {
            player.is_up = true;
        }
        if (keyCode == DOWN) {
            player.is_down = true;
        }
        if (keyCode == RIGHT) {
            player.is_right = true;
        }
        if (keyCode == LEFT) {
            player.is_left = true;
        }
        if (keyCode == SHIFT) {
            player.speed = 2;
        }
    }
}

// キーを離したとき
void keyReleased() {
    //プレイヤーの動作
    if (key == CODED) {
        if (keyCode == UP) {
            player.is_up = false;
        }
        if (keyCode == DOWN) {
            player.is_down = false;
        }
        if (keyCode == RIGHT) {
            player.is_right = false;
        }
        if (keyCode == LEFT) {
            player.is_left = false;
        }
        if (keyCode == SHIFT) {
            player.speed = 5;
```

```
53          }
54      }
55  }
56  class Player{
57      //フィールドの宣言
58      //プレイヤー表示
59      //プレイヤーの座標。
60      float x;
61      float y;
62      //十字キーが押されているかの判定
63      boolean is_up;
64      boolean is_down;
65      boolean is_right;
66      boolean is_left;
67      //---------------
68      boolean movable = true;
69      float speed; //プレイヤーの移動スピードを定義。
70      float HP;
71      float Max_HP;
72      float XP;
73      float Max_XP;
74      //コンストラクタを宣言
75      Player(float position_x,float position_y,boolean up,boolean down,boolean right,boolean left,
          float speed_arg , float health , float Max_HP,float XP, float Max_XP) {
76              x = position_x;
77              y = position_y;
78              is_up = up;
79              is_down = down;
80              is_right = right;
81              is_left = left;
82              speed = speed_arg;
83              HP = health;
84      }
85      // Playerクラス内で使用する関数
86      void display() {
87              noStroke();
88              image(shiratuyu,x - 50,y - 55);
89      }
90      // プレイヤーを動かす。
91      void movement() {
92              if (y > 0 && is_up) {
93                      y -= speed;
94                      //-----敵に当たった時-----//
95                      if (dist(x,y,enemy.x,enemy.y) < 40) {
96                              y += speed;
97                              player.HP -= 1;
98                      }
99              }
100             if (y <= height && is_down) {
101                     y += speed;
102                     //----敵に当たった時-----//
103                     if (dist(x,y,enemy.x,enemy.y) < 40) {
104                             y -= speed;
105                             player.HP -= 1;
106                     }
107             }
108             if (x <= Game_width && is_right) {
109                     x += speed;
110                     //-----敵に当たった時-----//
111                     if (dist(x,y,enemy.x,enemy.y) < 40) {
112                             x -= speed;
113                             player.HP -= 1;
114                     }
115             }
116             if (x >= 0 && is_left) {
117                     x -= speed;
118                     //-----敵に当たった時-----//
119                     if (dist(x,y,enemy.x,enemy.y) < 40) {
120                             x += speed;
121                             player.HP -= 1;
122                     }
123             }
124     }
```

```
125 }
```

■3.2.1.4　**処理の説明**　キー入力があり、かつプレイヤーが枠内に収まっているときプレイヤーは 1 フレームにつき 5 ずつ動く。

敵座標との距離が 40 以内であるときプレイヤーは 1 ダメージを受ける。敵機のいる方向に移動しようとすると以下の図のように逆方向に動かされる。これによって敵にぶつかる処理を実装している。

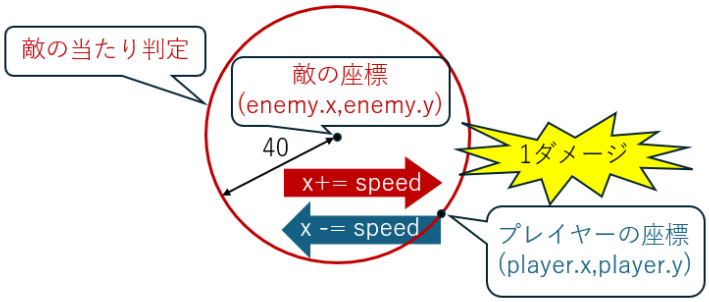

図 5: 敵との衝突処理

SHIFT キーで低速移動を行う。

3.2.2　自機の基本技

プレイヤーが z キーを押したとき弾が発射できるようになっている。この弾を自機弾、自機弾を発射する必殺技を基本技と定義する。

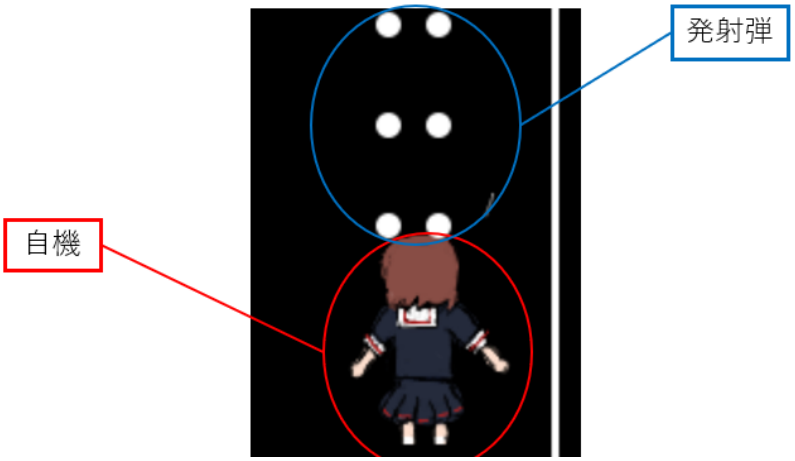

図 6: 基本技

■3.2.2.1　**使用した関数・変数**　新たに使用した関数・変数は以下の表のとおり

型・クラス	変数・関数	説明
ArrayList<Weapon>	bullets	左側の自機弾
ArrayList<Weapon>	bullets2	右側の自機弾
int	NumOfBullet	左側の自機弾の数
int	NumOfBullet2	右側の自機弾の数
boolean	z_push	z キーが押されているかどうか
void	movebullet()	自機弾を上に動かす関数
void	control_bullet()	自機弾の管理
float	dist()	二点間の距離を算出する組み込み関数

■3.2.2.2　**パラメータ**　自機弾のパラメータを表で示すと以下の通りになる。

パラメータ	説明
初期 x 座標	プレイヤーの x 座標-10(bullet),+10(bullet2)
初期 y 座標	プレイヤーの y 座標-22
弾速	20
攻撃力	10
進行方向	上 (y 負方向)

■3.2.2.3　**ソースコード**　以下に基本技を定義するソースコードを示す。

```
1  void setup(){
2      bullets = new ArrayList<Weapon>(); //味方の弾を定義
3      bullet2 = new ArrayList<Weapon>(); //味方の弾(二個目)を定義
4  }
5  void draw(){
6      if (game_play && phase > 0) {
7        //Zキーを押して弾を発射
8        if (z_push) {
9            if ((player.speed == 5 && frameCount % 5 == 0) || (player.speed == 2 && frameCount % 2
               == 0)) {
10               bullets.add(new Weapon(player.x - 10,player.y - 22,20,10,0,0,0));//弾の生成
11               bullet2.add(new Weapon(player.x + 10,player.y - 22,20,10,0,0,0));//弾の生成
12               //弾の数のカウントを増やす
13               NumOfBullet++;
14               NumOfBullet2++;
15       }
16       //Weapon(初期x座標,初期y座標,弾速,攻撃力,消費XP,x進行方向,y進行方向)
17       control_bullet();
18     }
19     }
20 }
21
22 // キーを押したとき
23 void keyPressed() {
24     if (key == 'z' || key == 'Z') {
```

```
25          z_push = false;
26      }
27  }
28  // キーを離したとき
29  void keyReleased() {
30      if (key == 'z' || key == 'Z'){
31          z_push = false
32      }
33  }
34  class Weapon{
35      // 球を動かす
36      void movebullet() {
37          fill(0,0,400);
38          noStroke();
39          y -= speed;
40          ellipse(x,y,10,10);
41      }
42  }
43  //自機弾の制御
44  void control_bullet() {
45      for (int i = 0; i < NumOfBullet - 1; i++) {
46          bullets.get(i).movebullet();
47          //画面外に出たとき、要素を削除する。
48          if (bullets.get(i).y < 0) {
49              bullets.remove(i);
50              NumOfBullet--;
51          }
52          //敵の当たり判定。敵に当たった時ダメージを与え、弾を消去する。
53          if ((bullets.get(i).x < enemy.x + 30 && bullets.get(i).x > enemy.x - 30) && (bullets.get(
              i).y > enemy.y - 30 && bullets.get(i).y < enemy.y + 30)) {
54              enemy.HP -= bullets.get(i).atk;
55              bullets.remove(i);
56              NumOfBullet--;
57          }
58      }
59      //自機弾（2個目）の制御
60      for (int i = 0; i < NumOfBullet2 - 1; i++) {
61          bullet2.get(i).movebullet();
62          //画面外に出たとき、要素を削除する。
63          if (bullet2.get(i).y < 0) {
64              bullet2.remove(i);
65              NumOfBullet2--;
66          }
67          //当たり判定
68          if ((bullet2.get(i).x < enemy.x + 30 && bullet2.get(i).x > enemy.x - 30) && (bullet2.get
              (i).y > enemy.y - 30 && bullet2.get(i).y < enemy.y + 30)) {
69              enemy.HP -= bullet2.get(i).atk;
70              bullet2.remove(i);
71              NumOfBullet2--;
72          }
73      }
74  }
```

■3.2.2.4　**処理の説明**　z キーが押されている間、以下の挙動を行う。

- プレイヤーのスピードが 5(通常移動) の時 5 フレーム間隔で弾を発射する。
- プレイヤーのスピードが 2(低速移動) の時 2 フレーム間隔で弾を発射する。

movebullet() 関数では 1 フレーム毎に弾を上に動かす。弾速 20 であるため 20 ずつ動かしている。
※図 7

control_bullet() 関数では以下の役割を果たしている。

1. 弾が範囲外に出たとき弾を消去する
2. 敵の当たり判定に当たった時敵にダメージを与え、弾を消去する。

自機弾については図 8 のように敵の上下左右 30 を敵の当たり判定としている。また弾が一つ当たるたびに敵に 10 ダメージを与える。

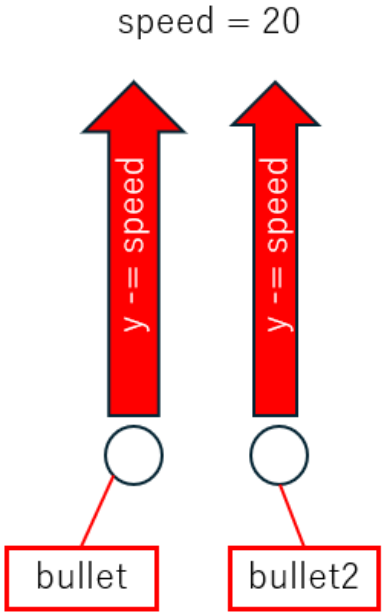

図 7: 弾が上に動く仕組み

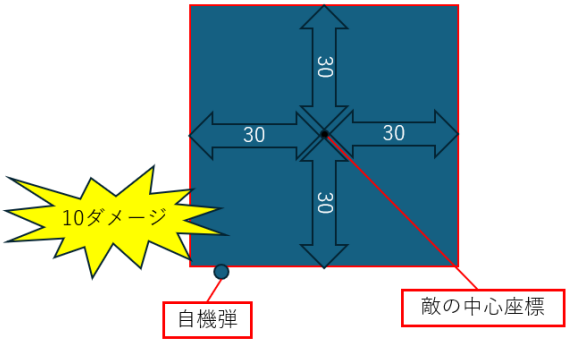

図 8: 弾の当たり判定

3.2.3 ボム

ボムには以下の特徴がある

- x キーで発射できる
- ランダムな位置に発生
- 円状である
- 時間経過に応じて半径が大きくなる ※図 10
- 半径が 200 を超えたとき消滅する
- 一度に 8 個発動できる
- 半径内の弾を消す。※図 11
- 半径内の敵にダメージを与える。※図 11

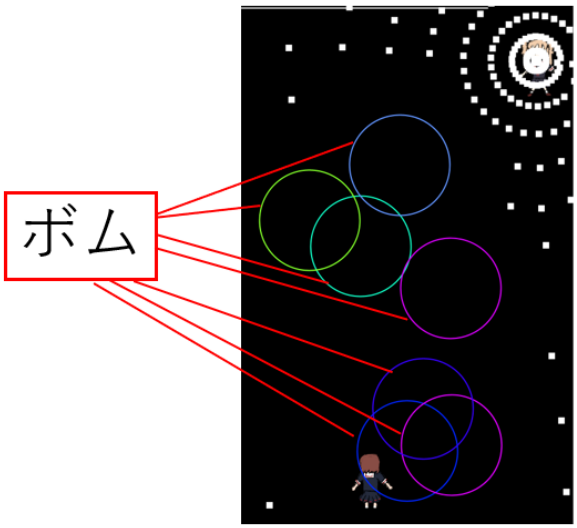

図 9

■3.2.3.1　**使用した変数・関数**　ボムの定義には新たに以下の変数・関数を使用した。

型・クラス	変数・関数	説明
ArrayList<Weapon>	bomb	ボムのオブジェクト
int	x_count	x ボタンを押してからの時間経過
int	NumOfbomb	ボムの数
float	r	ボムの半径
float	distance	ボムとオブジェクトの距離
boolean	x_push	x が押されているかどうか
void	bomb()	ボムの描画・拡大処理
void	control_bomb()	ボムの挙動を制御

■3.2.3.2　**パラメータ**　ボムのパラメータは以下の通り。

パラメータ	値
中心 x 座標	100〜400 までのランダム値
中心 y 座標	100〜700 までのランダム値
半径	初期値:0 最大値:200
拡大速度	2.5
攻撃力	30
消費 XP	10

■3.2.3.3 **ソースコード**　ボムの定義に使ったソースコードは以下の通り。ここには敵必殺技に関係する変数が含まれているが後述する。

```
    void setup(){
        bomb = new ArrayList<Weapon>(); //ボムを定義。
    }
    void draw(){
        //xキーでボム発射。
        if(game=play && phase>0){
        if (x_push) {
            //XPが足りているかの判定かつXPが余計に減るのを防ぐ。
            if (player.XP >= 10 && x_count == 1) {
                NumOfbomb = 8;
                for (int i = 0; i < NumOfbomb; i++) {
                    bomb.add(new Weapon(random(100,400),random(100,700),20,30,10,0,0));
                }
                player.XP -= bomb.get(0).XP;
            }
            //xボタンをどのくらいの間押しているか。
            x_count++;
        }
        else{
            //Xキーを離したとき、カウントを0にする
            x_count = 0;
        }
                control_bomb();
        }
    }
    void keyPressed(){
        if (key == 'x' || key == 'X') {
        x_push = false;
    }
    }
    void keyReleased(){
        if (key == 'x' || key == 'X') {
        x_push = false;
    }
    }
class Weapon{
        void bomb() {
        r += speed / NumOfbomb;
        strokeWeight(2);
        noFill();
        stroke(x,y,400);
        ellipse(x, y, r, r);
  }
}
void control_bomb() {
  float distance = 500; //ボムとオブジェクトの間の距離。500として初期化する。
  if (NumOfbomb > 0) {
      for (int i = 0; i < NumOfbomb - 1; i++) {
          //各ボムの挙動
          bomb.get(i).bomb();
          //ボムの大きさが200を超えたとき、ボムを消す。
```

```
52              if (bomb.get(i).r > 200) {
53                  bomb.remove(i);
54                  NumOfbomb--;
55              }
56              distance = dist(bomb.get(i).x,bomb.get(i).y,enemy.x,enemy.y); //ボムと敵の間の距離
57              if (distance < bomb.get(i).r) {
58                  enemy.HP -= bomb.get(i).atk;//敵がボムの射程圏内の場合、敵にダメージを与える。
59              }
60              //ボムと四角形の間の距離を計算し処理
61              //第一フェーズでのボムの挙動
62              if (phase == 1 && NumOfRect > 0) {
63                  for (int j = 0; j < NumOfRect; j++) {
64                      distance = dist(bomb.get(i).x,bomb.get(i).y,Rect_attack.get(j).x,Rect_attack.
                          get(j).y); //ボムと四角形の間の距離。
65                      if (distance < bomb.get(i).r + 25) {
66                          Rect_attack.remove(j); //ボムが四角形に触れた場合、四角形を削除する。
67                          NumOfRect--; //四角形のカウントを減らす。
68                      }
69                  }
70              }
71              //第二フェーズでのボムの挙動。
72              else if (phase == 2 && NumOfCParts > 0) {
73                  for (int j = 0; j < NumOfCParts; j++) {
74                      distance = dist(bomb.get(i).x,bomb.get(i).y,Cparts.get(j).x,Cparts.get(j).y);
                          //ボムと円状弾の距離。
75                      if (distance < bomb.get(i).r + 5) {
76                          Cparts.remove(j);//ボムが弾に触れた場合、弾を削除する。
77                          NumOfCParts--; //弾のカウントを減らす。
78                      }
79                  }
80              }
81              //第三フェーズでのボムの挙動。
82              else if (phase == 3) {
83                  //sinb弾の数が0より多い時。
84                  if (NumOfsinb > 0) {
85                      for (int j = 0; j < NumOfsinb; j++) {
86                          distance = dist(bomb.get(i).x,bomb.get(i).y,sinb.get(j).x,sinb.get(j).y);
                              //ボムとsin弾の距離。
87                          if (distance < bomb.get(i).r + 5) {
88                              sinb.remove(j); //sin弾を削除する。
89                              NumOfsinb--; //sin弾のカウントを減らす。
90                          }
91                      }
92                  }
93                  if (NumOfcosb > 0) {
94                      for (int j = 0; j < NumOfcosb; j++) {
95                          distance = dist(bomb.get(i).x,bomb.get(i).y,cosb.get(j).x,cosb.get(j).y);
                              //ボムとcos弾の距離。
96                          if (distance < bomb.get(i).r) {
97                              //ボムの射程範囲内にcos弾が入った時
98                              cosb.remove(j); //cos弾を削除する。
99                              NumOfcosb--; //cos弾のカウントを減らす。
100                         }
101                     }
102                 }
103             }
104             //最終フェーズでのボムの挙動。
105             else if (phase == 4) {
106                 if (NumOfgearb > 0) {
107                     for (int j = 0; j < NumOfgearb; j++) {
108                         distance = dist(bomb.get(i).x,bomb.get(i).y,gear_bullet.get(j).x,
                              gear_bullet.get(j).y); //ボムとギアの弾の間の距離。
109                         //ギアの弾がボムの射程圏内に入った時
110                         if (distance < bomb.get(i).r) {
111                             gear_bullet.remove(j); //ギアの弾を削除する。
112                             NumOfgearb--; //ギアの弾のカウントを減らす。
113                         }
114                     }
115                 }
116                 if (NumOfgear > 0) {
117                     for (int j = 0; j < NumOfgear; j++) {
118                         distance = dist(bomb.get(i).x,bomb.get(i).y,gear.get(j).x,gear.get(j).y);
                              //ボムとギアの間の距離。
```

```
119                         if (distance < bomb.get(i).r) {
120                             //ギアがボムの射程圏内に入った場合
121                             gear.get(j).HP -= bomb.get(i).atk; //ギア（歯車）にダメージを与える。
122                         }
123                     }
124                 }
125             }
126         }
127     }
128 }
```

■3.2.3.4　**処理の説明**　bomb_contorl() においてはボムの半径内に敵弾が入っている、また敵が入っているかの判別条件は以下のとおりである

ボムの座標､敵･敵弾の座標間の距離 < ボムの半径

全てのボムについてすべての弾に対する距離を解析し、消去の有無を判断している。

また各フェーズで条件分けして弾を消す処理を記述しているがこれは処理の負担を減らすためである。

図 10: ボムは毎フレーム 2.5 ずつ拡大する

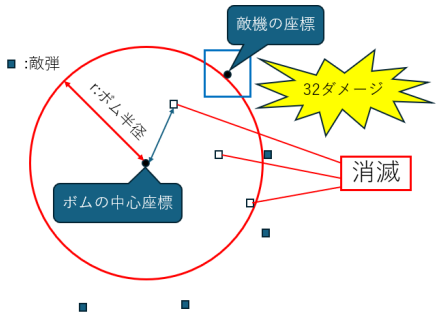

図 11: ボムの敵・敵弾に対する処理

3.3　敵機の動作

3.3.1　敵機の基本動作

■3.3.1.1　**使用した関数・変数**　敵機の基本動作の定義に使用した関数・変数は以下の表のとおり。

型	変数・関数	説明
class	Enemy	敵を定義するクラス
float	x	敵の x 座標
float	y	敵の y 座標
float	xmove	敵が向かう x 位置
float	ymove	敵が向かう y 位置
float	HP	敵の HP
bool	is_boss	敵機かどうか
void	display()	敵を表示する関数
void	enemy_move()	ランダムに敵を動かす関数
void	HP_gauge()	敵の HP ゲージを表示する関数
Enemy	enemy	敵機を表すオブジェクト
なし	murasame	敵機の画像

■3.3.1.2　**パラメータ**　以下に敵機の基本パラメータを記述する。

パラメータ	値
初期 x 座標	256
初期 y 座標	200
スピード	5
最大 HP	16000
ボスかどうか	true

■3.3.1.3　**ソースコード**　以下のソースコードで敵機の動作処理を記述した。

```
1     void setup(){
2         murasame = loadImage("murasame.png");
3         enemy = new Enemy(Game_width / 2,height / 4,50,50,16000,16000,true);
4     //Enemy(x初期値,y初期値,xサイズ,yサイズ,HP,Max_HP,ボスかどうか)
5     }
6  void draw(){
7         if (game_play && phase > 0) {
8         enemy.display(); //敵を表示
9         enemy.enemy_move(); //敵の動き
10        enemy.HP_gauge(); //敵のHP
11    }
12 }
13    // 敵機のクラス
14    class Enemy{
15            //フィールドの宣言
16            float x;
17            float y;
18            float xmove = Game_width / 2;
19            float ymove = height / 4;
20            float HP;
```

```
21              float Max_HP;
22              boolean is_boss;
23              Enemy(float x, float y,float HP,float Max_HP, boolean is_boss) {
24                      this.x = x;
25                      this.y = y;
26                      this.HP = HP;
27                      this.Max_HP = Max_HP;
28                      this.is_boss = is_boss;
29              }
30              void display() {
31                      noStroke();
32                      image(murasame, x - 50, y - 45);
33              }
34              void enemy_move() {
35                      if (phase ==  1 && frameCount % 1000 == 0) {
36                              xmove = random(512);
37                              ymove = random(200);
38                      }
39                      else if (phase == 2 && frameCount % 500 == 0) {
40                              xmove = random(512);
41                              ymove = random(200);
42                      }
43                      else if (phase == 3 && frameCount % 200 == 0) {
44                              xmove = random(512);
45                              ymove = random(200);
46                      }
47                      else if (phase == 4 && frameCount % 300 == 0) {
48                              if (is_boss == false) {
49                                      xmove = random(512);
50                                      ymove = random(height);
51                              }
52                              if (is_boss == true && frameCount % 600 == 0) {
53                                      xmove = random(512);
54                                      ymove = random(400);
55                              }
56                      }
57                      if (!(x > xmove - 10 && x < xmove + 10)) {
58                              if (x < xmove) {
59                                      x += 5;
60                              }
61                              else if (x > xmove) {
62                                      x -= 5;
63                              }
64                      }
65                      if (!(y > ymove - 10 && y < ymove + 10)) {
66                              if (y < ymove) {
67                                      y += 5;
68                              }
69                              else if (y > ymove) {
70                                      y -= 5;
71                              }
72                      }
73              }
74              void HP_gauge() {
75                      if (is_boss) {
76                              stroke(255);
77                              strokeWeight(3);
78                              line(0, 3, HP * (Game_width / Max_HP), 3);
79                      }
80              }
81      }
```

■3.3.1.4　**処理の説明**　ある一点をランダムに決める。するとその点を目指して敵が 5 ずつ動く。目的店の四方 10 以内に敵が入ると敵は止まる。図 12 この際敵の動作を数式で表すと以下の通り

になる。

敵機の座標を(x, y)、目的点の座標を(x', y')とすると、

$$x = \begin{cases} \begin{cases} x+5 & (x < x') \\ x-5 & (x > x') \end{cases} & (x \geqq x'+10), (x \leqq x'-10) \\ x & (x'-10 < x < x'+10) \end{cases}$$

$$y = \begin{cases} \begin{cases} y+5 & (y < y') \\ y-5 & (y > y') \end{cases} & (x \geqq y'+10), (y \leqq y'-10) \\ y & (y'-10 < y < y'+10) \end{cases}$$

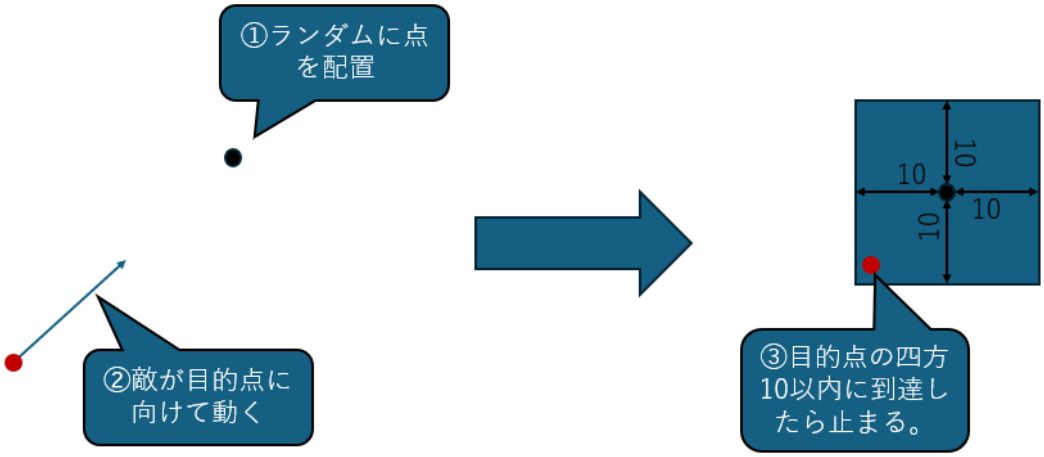

図 12: 敵がランダムに動く原理

また初めは敵が目的点に到達したときに止まるように定義したが

- スピードを 1 にすると目的点に到達するのが遅く位置が定まらない
- スピードを 5 にすると調整が難しく、位置が定まらない

という問題があったため範囲を設定した。また各フェーズによって動作頻度は異なる。その表を以下に示す。

フェーズ	移動頻度	移動範囲
1	1000 フレーム	x 座標:0〜512　y 座標：0〜200
2	500 フレーム	上に同じ
3	200 フレーム	上に同じ
4	300 フレーム	x 座標：0〜512　y 座標：0〜800

3.3.2　フェーズ 1

フェーズ 1 では、プレイヤーはランダムに生成される 50 × 50 平方の四角形を避けることになる。※図 13

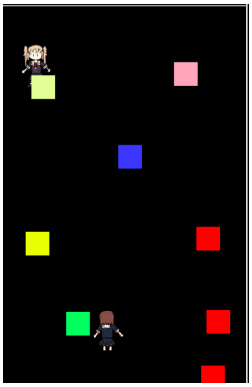

図 13: フェーズ 1 におけるゲーム画面

■3.3.2.1　定義に使用した変数・関数　新たに以下の変数・関数をフェーズ 1 の動作定義に使用した。

型・クラス	変数・関数	説明
ArrayList<Weapon>	Rect_attack	四角形を落とす必殺技のオブジェクト
int	NumOfRect	四角形の数
void	rect_fall()	四角形を落とす関数
void	control_rect	四角形の制御

■3.3.2.2　パラメータ　オブジェクト Rect_attack のパラメータを以下に示す。

パラメータ	値
初期 x 座標	0〜500 までのランダム値
初期 y 座標	0〜200 までのランダム値
弾速	10
攻撃力	15
発射頻度	10 フレーム
進行方向	下 (y 正方向)

■3.3.2.3　ソースコード　フェーズ 1 の動作定義に使用したソースコードを以下に示す。

```
void setup(){
    Rect_attack = new ArrayList<Weapon>(); //敵の玉を定義
}
void draw(){
    if(game_play && phase > 0){
        //第一フェーズの弾を生成
    if (phase == 1 && frameCount % 10 == 0) {
      Rect_attack.add(new Weapon(random(500),random(200),10,15,0,0,0));
      NumOfRect++;
    }
        control_rect();
    }
}
class Weapon{
    //四角形を落とす
    void rect_fall() {
        noStroke();
        fill(x,y,400);
        y += speed;
        rect(x,y,50,50);
    }
}
void control_rect() {
    for (int i = 0; i < NumOfRect - 1; i++) {
        Rect_attack.get(i).rect_fall(); //四角形を落とす。
        if (NumOfRect > 1) {
            if (Rect_attack.get(i).y> height || Rect_attack.get(i).x > Game_width) {
                //四角形弾が画面外に出たとき。
                Rect_attack.remove(i); //四角形を削除
                NumOfRect--;
            }
            if ((Rect_attack.get(i).x < player.x + 25 && Rect_attack.get(i).x > player.x - 25)
              && (Rect_attack.get(i).y < player.y + 25 && Rect_attack.get(i).y > player.y - 25))
               {
                //四角形がプレイヤーの当たり判定と衝突したとき
                player.HP -= Rect_attack.get(i).atk; //プレイヤーが四角形の攻撃力分だけダメージ
                  を受ける
                Rect_attack.remove(i); //四角形を削除する
                NumOfRect--; //四角形のカウントを減らす。
            }
        }
    }
}
```

■3.3.2.4　**処理の説明**　このフェーズでは以下のようにオブジェクトを処理する。

- 四角形の色は、色相を x 座標の値、彩度を y 座標の値、明度を 400 としている。14
- 画面外に出たもしくはプレイヤーに当たった時弾を削除する。
- 四角形は 1 フレームにつき 10 下に落ちる。※図 14
- プレイヤーの当たり判定は四方 25 である。※ 図 15
- プレイヤーが四角形に当たった時プレイヤーは 15 ダメージを受ける。※図 15

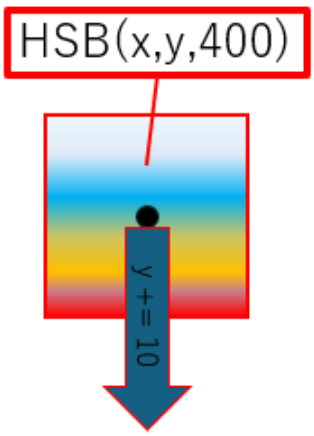

図 14: rect_fall() 関数での処理

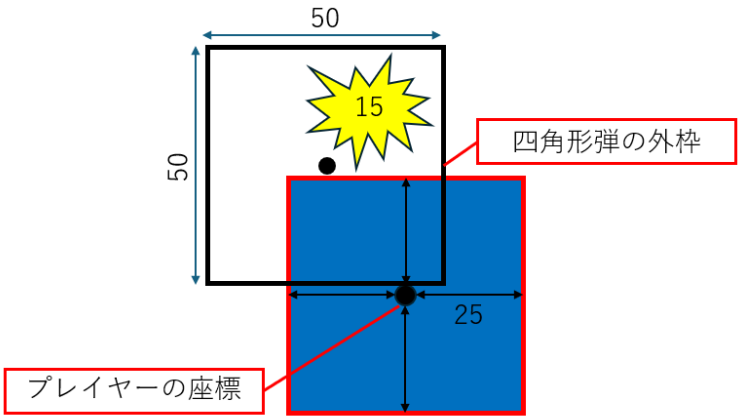

図 15: rect_control() 関数での処理

3.3.3 フェーズ 2

フェーズ 2 ではプレイヤーは軌道をずらしながら発射される円状の攻撃弾を避けることになる。※図 16

また敵の中心座標から弾が発射されるため、敵が移動すると軌道が大きくずれる※図 17

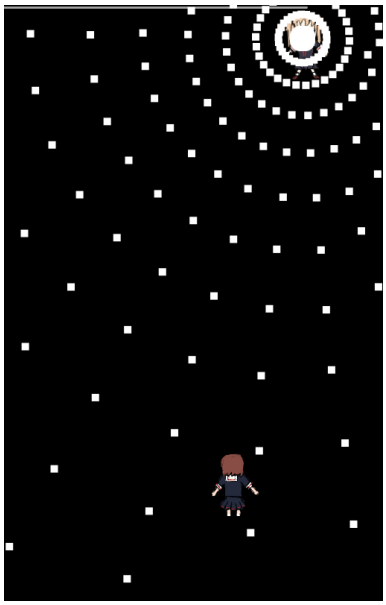

図 16: 通常時の弾幕

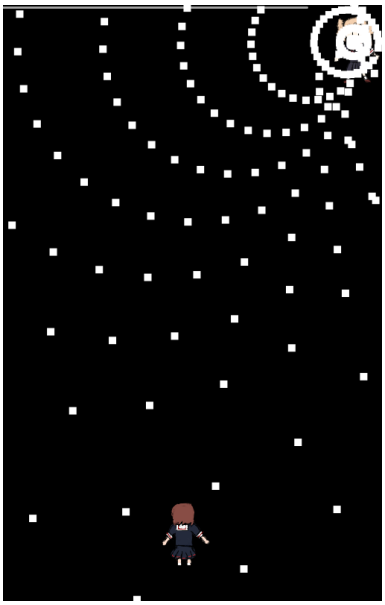

図 17: 敵が動いた時の弾幕

■3.3.3.1 **定義に使用した関数・変数** 以下の変数・関数をフェーズ 2 の動作定義に使用した。

型	変数・関数	説明
ArrayList<Weapon>	Cparts	円状の攻撃弾を表すオブジェクト
int	NumOfCParts	攻撃弾の数
float	vecX	弾の x ベクトル
float	vecY	弾の y ベクトル

■3.3.3.2　**パラメータ**　オブジェクト Cparts のパラメータは以下の表のとおり。

パラメータ	値
x 初期位置	敵機の x 座標
y 初期位置	敵機の y 座標
弾速	3
攻撃力	5
発射頻度	10 フレーム
x 進行方向 (単位ベクトル)	(12 * (1〜36 の整数)) + (弾の数 / 30)
y 進行方向 (単位ベクトル)	(12 * (1〜36 の整数)) + (弾の数 / 30)

■3.3.3.3　**ソースコード**　フェーズ 2 の動作定義に使用したソースコードを以下に示す。

```
1  void setup(){
2      Cparts = new ArrayList<Weapon>(); //敵の弾を定義
3  }
4  void draw(){
5      if(game_play && phase > 0){
6          //第二フェーズの弾を生成
7          if (phase == 2 && frameCount % 10 == 0) {
8                  for (int i = 0; i < 30; i++) {
9                          //敵の中心から円状に弾を発射する。弾の数に応じて軌道を少しづつずらす。
10                         Cparts.add(new Weapon(enemy.x,enemy.y,3,5,0,(12 * i) + NumOfCParts /
                             30,(12 * i) + NumOfCParts / 30));
11                         NumOfCParts++;
12                 }
13         }
14         control_cparts();
15         frameCount++;
16     }
17 }
18 class Weapon{
19     //円状に弾を発射する
20     void cbullet() {
21             noStroke();
22             fill(0,0,400);
23             r += speed / 30;
24             x += r * vecX;
25             y += r * vecY;
26             rect(x,y,10,10);
27     }
28 }
29 //円状弾の制御
30 void control_cparts() {
31     if (NumOfCParts > 0) {
32         for (int i = 0; i < NumOfCParts - 1; i++) {
33             Cparts.get(i).cbullet(); //円形弾幕の操作。
```

```
34              if (Cparts.get(i).y > height || Cparts.get(i).x > Game_width) {
35                  //弾が画面外に出たとき
36                  Cparts.remove(i); //弾を削除。
37                  NumOfCParts--;
38              }
39              if ((Cparts.get(i).x < player.x + 10 && Cparts.get(i).x > player.x - 10) && (Cparts.
                  get(i).y < player.y + 10 && Cparts.get(i).y > player.y - 10)) {
40                  //プレイヤーに弾が当たった時(当たり判定サイズ:10)
41                  player.HP -= Cparts.get(i).atk; //プレイヤーがダメージを受ける
42                  Cparts.remove(i); //弾を削除する。
43                  NumOfCParts--; //弾のカウントを減らす。
44              }
45          }
46      }
47  }
```

■3.3.3.4　**処理の説明**　フェーズ 2 では以下の処理を行う。※図 18

1. 12° 感覚で弾を 30 個配置する。
2. 弾と敵機の距離（発射半径）をスピード分大きくする。
3. 発射角度を弾の数に応じてずらす。
4. 1〜3 を繰り返す。

座標定義を数式で表すと以下のようになる

弾の座標を(x, y)、敵機との距離をr、弾数をnとすると、

$$x = r\cos(12 \times i + \frac{n}{30})$$

$$y = r\sin(12 \times i + \frac{n}{30})$$

また、フレーム更新時に、

$$r = r + 5$$

また当たり判定はプレイヤーの中心座標から四方 10 以内である。※図 19

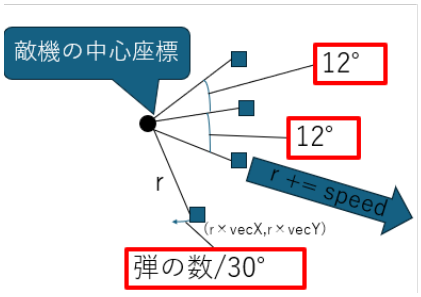

図 18: 円状攻撃弾の発射原理

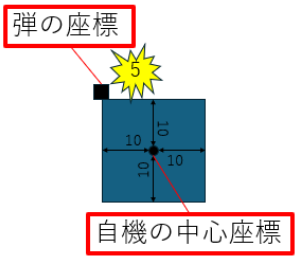

図 19: 円状攻撃弾の当たり判定

以上の処理で弾が座標をずらしながら円状に発射される必殺技を実装している。

3.3.4 フェーズ 3

このフェーズでは、プレイヤーは上から降ってくる sin グラフの形に一定間隔で配置された弾 (以降 sin 弾) と、下から降ってくる cos グラフの形に一定間隔で配置された弾 (以降 cos 弾) をよけることになる※図 20

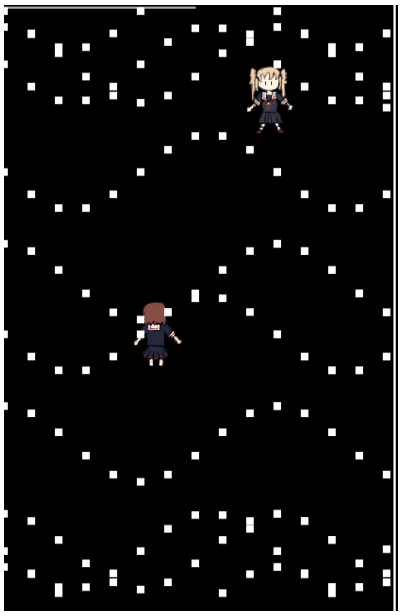

図 20: フェーズ 3 におけるゲーム画面

■3.3.4.1 使用した変数・関数 フェーズ 3 の定義に使用した変数・関数は以下のとおりである。

型	変数・関数	説明
ArrayList<Weapon>	sinb	sin 弾を定義
ArrayList<Weapon>	cosb	cos 弾を定義
float	A	sin,cos 弾の振幅
float	fire	一周期につき何個弾を並べるか
float	T	周期
float	biase	グラフをどれくらいずらすか。
int	NumOfsinb	sin 弾の数
int	NumOfcosb	cos 弾の数
void	tetrab()	sin,cos 弾の表示、落下
void	control_tetra()	sin,cos 弾の制御

■3.3.4.2　**パラメータ**　sin,cos 弾のパラメータを以下に示す。

パラメータ	値	対象
T(周期)	360	共通
fire(周期数)	10	共通
A(振幅)	50	共通
初期 biase	0	共通
i	何周期目か	共通
初期 x 座標	$360 \times i + 36 \times$ (整数)	共通
初期 y 座標	$50\sin(36 * j)$○	sin 弾
初期 y 座標	$50\cos(36 * j)$○	cos 弾
弾速	0.02	共通
攻撃力	5	共通
進行方向	下 (y 正方向)	sin 弾
進行方向	上 (y 負方向)	cos 弾

■3.3.4.3　**ソースコード**　フェーズ 3 の動作定義に使用したソースコードを以下に示す。

```
void setup(){
    sinb = new ArrayList<Weapon>(); //sin弾を定義。
  cosb = new ArrayList<Weapon>(); //cos弾を定義。
}
void draw(){
    if(game_play && phase > 0){
        //第三フェーズの弾を生成。
        if (phase == 3 && frameCount % 60 == 0) {
                for (int i = 0; i < 2; i++) {
                        for (int j = 0; j < T / fire; j++) {
                                if (T * i + (T / fire) * j < Game_width) {
                                        //Sin状に弾を生成。
                                        sinb.add(new Weapon(T * i + (T / fire) * j , A * sin(
                                          radians(36 * j)),0.02,5,0,90,90));
                                        NumOfsinb++;
                                }
                                if (T * i + (T / fire) * j < Game_width) {
                                        //cos状に弾を生成
                                        cosb.add(new Weapon(T * i + (T / fire) * j , height - A
                                          * cos(radians(36 * j)),0.02,5,0,90, -90));
                                        NumOfcosb++;
                                }
                        }
                }
        }
        control_tetra();
        frameCount++;
    }
}
class Weapon{
    void tetrab() {
        x += biase * vecX;
        y += biase * vecY;
        noStroke();
        fill(0,0,400);
        rect(x,y,10,10);
```

```
35          biase += speed;
36      }
37  }
38  //sin弾の制御
39  void control_tetra() {
40      if (NumOfsinb > 0) {
41          for (int i = 0; i < NumOfsinb - 1; i++) {
42              //各弾を動かす。
43              sinb.get(i).tetrab();
44              if (sinb.get(i).y > height) {
45                  //弾が画面外に出たとき
46                  sinb.remove(i); //sin弾を削除
47                  NumOfsinb--; //sin弾のカウントを減らす。
48              }
49              if ((sinb.get(i).x >= player.x - 10 && sinb.get(i).x <= player.x + 10) && (sinb.get(
                    i).y >= player.y - 10 && sinb.get(i).y <= player.y + 10)) {
50                  //プレイヤーに弾が当たった時
51                  player.HP -= sinb.get(i).atk; //プレイヤーがダメージを受ける。
52                  sinb.remove(i); //sin弾を削除。
53                  NumOfsinb--; //sin弾のカウントを減らす。
54              }
55          }
56      }
57      if (NumOfcosb > 0) {
58          for (int i = 0; i < NumOfcosb - 1; i++) {
59              cosb.get(i).tetrab();  //各弾を動かす。
60              if (cosb.get(i).y < 0) {
61                  //画面外に出たとき
62                  cosb.remove(i); //cos弾を削除
63                  NumOfcosb--; //cos弾のカウントを減らす。
64              }
65              if ((cosb.get(i).x >= player.x - 10 && cosb.get(i).x <= player.x + 10) && (cosb.get(
                    i).y >= player.y - 10 && cosb.get(i).y <= player.y + 10)) {
66                  //プレイヤーに当たった時
67                  player.HP -= cosb.get(i).atk; //プレイヤーがダメージを受ける。
68                  cosb.remove(i); //cos弾を削除する。
69                  NumOfcosb--; //cos弾のカウントを減らす。
70              }
71          }
72      }
73  }
```

■3.3.4.4　**処理の説明**　sin,cos 弾の実装において、振幅、周期、一周期における弾数を定義し各弾の y 座標に sin,cos をかけている。バイアスの値を増やすことによって位置をずらし、降ってくる動きを実装している。

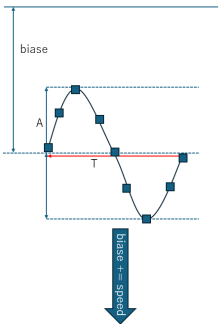

図 21: sin 弾の動く仕組み

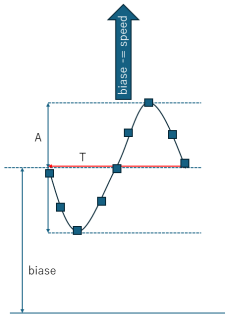

図 22: cos 弾の動く仕組み

当たり判定の定義はフェーズ 2 の弾※図 19 と同様である。

3.3.5　フェーズ 4

フェーズ 4 では敵機が歯車型の敵を召喚し、歯車型の敵はランダムに動きながらプレイヤーに向けて弾を打つ。それをよけながら敵を倒せばゲームクリアとなる。図 23

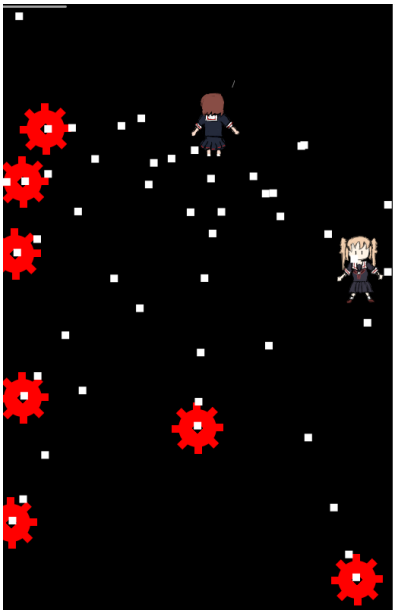

図 23: フェーズ 4 におけるゲーム画面

■3.3.5.1　使用した関数・変数　フェーズ 4 の動作定義に新たに用いた関数・変数は以下のとおりである。

型・クラス	関数・変数	説明
ArrayList<Enemy>	gear	歯車型の敵を定義するオブジェクト
ArrayList<Weapon>	gear_bullet	歯車の発射弾を定義するオブジェクト
boolean	gear_push	歯車を発射したかどうか
int	NumOfgear	歯車の数。初期値は 8
int	NumOfgearb	歯車の発射弾の数。
void	control_gear()	歯車型の敵を制御
void	gear()	歯車型の敵を表示。
float	atan2()	二点間の角度を求める関数。

■3.3.5.2　パラメータ　以下に歯車型の敵及びその発射弾のパラメータを示す。

初期位置	敵機の座標
動作範囲	x 座標:0～512　y 座標:0～800
移動頻度	300 フレーム毎
スピード	10
攻撃力	5
最大 HP	500
ボスかどうか	false

表 2: 歯車のステータス

初期位置	歯車の座標
弾速	15
攻撃力	5
発射頻度	10
進行方向	プレイヤーの方向

表 3: 歯車の弾のパラメータ

■3.3.5.3　**ソースコード**　フェーズ 4 の動作定義に使用したソースコードを以下に示す。

```
1  void setup(){
2      gear = new ArrayList<Enemy>(); //歯車型の敵を定義
3      gear_bullet = new ArrayList<Weapon>(); //ギアの発射弾を定義。
4  }
5  void draw(){
6      if(game_play && phase > 0){
7          //第四フェーズの要素を生成。
8          if (phase == 4) {
9                  if (gear_push) {
10                         NumOfgear = 8;
11                         for (int i = 0; i < NumOfgear; i++) {
12                                 //歯車を生成。
13                                 gear.add(new Enemy(enemy.x,enemy.y,50,50,5000,5000,false));
14                         }
15                         gear_push = false;
16                 }
17                 if (frameCount % 10 == 0) {
18                         float dx,dy;
19                         for (int i = 0; i < NumOfgear - 1; i++) {
20                                 //プレイヤーと歯車の間の距離を算出
21                                 dx = player.x - gear.get(i).x;
22                                 dy = player.y - gear.get(i).y;
23                                 //歯車の中心からプレイヤーの方向に進む弾を生成。
24                                 gear_bullet.add(new Weapon(gear.get(i).x,gear.get(i).y,10,5,0,
                                     degrees(atan2(dy,dx)),degrees(atan2(dy,dx))));
25                                 //歯車の弾をカウント。
26                                 NumOfgearb++;
27                         }
28                         //プレイヤーと敵の間の距離を算出。
29                         dx = player.x - enemy.x;
30                         dy = player.y - enemy.y;
31                         //敵が発射する玉を歯車の弾の一部として扱う。歯車の弾よりも早い速度で発射
                             する。
32                         gear_bullet.add(new Weapon(enemy.x,enemy.y,15,5,0,degrees(atan2(dy,dx)),
                             degrees(atan2(dy,dx))));
33                         //ギアの弾の数を増やす。
34                         NumOfgearb++;
35                 }
36         }
37     }
38 }
39 class Enemy{
40     //歯車を描画。
41     void gear() {
42         noStroke();
43         fill(400,400,400);
44         ellipse(x,y,50,50);
45         for (int i = 0; i < NumOfgear; i++) {
46             pushMatrix();
```

```
47              fill(0);
48              translate(x,y);
49              rotate(radians(5 * frameCount));
50              rect(0,0,10,10);
51              translate(30 * cos(radians((360 * i / 8))),30 * sin(radians(360 * i / 8)));
52              rotate(radians(360 * i / 8));
53              noStroke();
54              fill(400,400,400);
55              rect( -1, -1,10,10);
56              popMatrix();
57          }
58      }
59  }
60  //ギア（歯車）の制御。
61  void control_gear() {
62      for (int i = 0; i < NumOfgear - 1; i++) {
63          gear.get(i).gear(); //歯車を描画。
64          gear.get(i).enemy_move(); //歯車を動かす。
65          for (int j = 0; j < NumOfBullet - 1; j++) {
66              if (dist(bullets.get(j).x,bullets.get(j).y,gear.get(i).x,gear.get(i).y) < 30) {
67                  //歯車の当たり判定に自機弾が当たった時
68                  gear.get(i).HP -= bullets.get(j).atk; //歯車が自機弾の攻撃力分だけダメージを受け
                        る
69                  bullets.remove(j); //自機弾を消去
70                  NumOfBullet--; //弾の数のカウントを減らす。
71              }
72          }
73          for (int j = 0; j < NumOfBullet2 - 1; j++) {
74              if (dist(bullet2.get(j).x,bullet2.get(j).y,gear.get(i).x,gear.get(i).y) < 30) {
75                  //自機弾（二個目）が歯車に当たった時
76                  gear.get(i).HP -= bullet2.get(j).atk; //歯車のHP
77                  bullet2.remove(j); //自機弾（二個目）を削除する。
78                  NumOfBullet2--; //自機弾（二個目）のカウントを減らす。
79              }
80          }
81          if (gear.get(i).HP < 0) {
82              //ギアのHPが0を下回ったとき
83              gear.remove(i); //ギアを消す。
84              NumOfgear--; //ギアの数を減らす。
85          }
86      }
87      for (int i = 0; i < NumOfgearb - 1; i++) {
88          gear_bullet.get(i).cbullet(); //ギアの発射弾を動かす。
89          if ((gear_bullet.get(i).x < player.x + 25 && gear_bullet.get(i).x > player.x - 25) && (
              gear_bullet.get(i).y <player.y + 25 && gear_bullet.get(i).y > player.y - 25)) {
90              //ギアの発射弾がプレイヤーに当たった時。
91              player.HP -= gear_bullet.get(i).atk; //プレイヤーがダメージを受ける。
92              gear_bullet.remove(i); //ギアの発射弾を削除する。
93              NumOfgearb--; //ギアの発射弾のカウントを減らす。
94          }
95          if (gear_bullet.get(i).y > height || gear_bullet.get(i).y < 0 || gear_bullet.get(i).x >
              Game_width) {
96              //ギア弾が画面外に出たとき
97              gear_bullet.remove(i); //ギア弾の削除
98              NumOfgearb--; //ギア弾のカウントを減らす。
99          }
100     }
101 }
```

■3.3.5.4　**処理の説明**　歯車の描画から先に説明する。歯車は二重円を描画しそこに四角形を付け加えることで実現している。また付け加えた四角形を回転させながら二重円の周りを回らせることで歯車が回転する演出を実現している。※図 24

基本的な歯車の挙動は敵機の挙動に準拠する。

プレイヤーに向かって弾を発射する挙動については atan2 関数でプレイヤーと歯車との間の角度を求めてそれを基に発射方向を定めるといった処理を行っている。※図 25

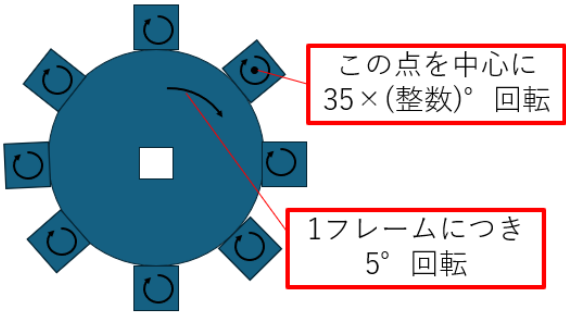

図 24: 歯車の描画

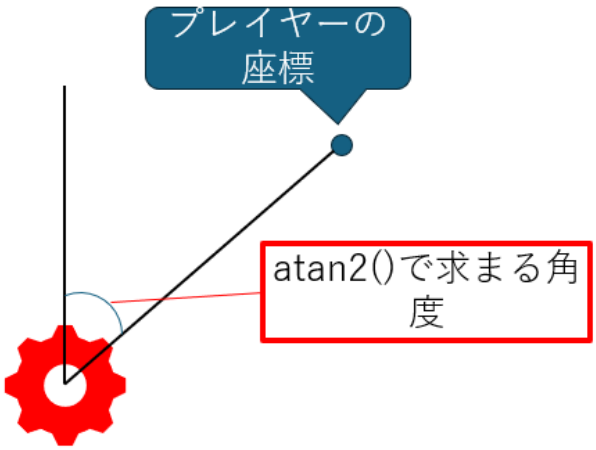

図 25: 歯車の弾の制御

歯車の弾の当たり判定はプレイヤーの四方 25 である。
歯車本体の当たり判定は半径 30 以内である。

4 結果と考察

以上のアプリを制作した結果、学科内ランキング 9 位という結果であった。

4.1 他者からの評価とそれに対する考察

以下のような他者からの評価が見られた。

- きちんと東方になっていた
- ゲームバランスが良かった
- 弾幕が手が込んでいてよかった。
- 複数の攻撃手段があり面白かった
- 弾幕に画面酔いしてしまった

4.2 問題点とその解決法

弾幕に画面酔いする問題については改善しなければならない。フェーズ 2,3,4 共に弾幕を白い四角形にしたため視認性が悪く画面酔いをしてしまったと思われる。この点については視認性の良いグラフィックを制作することで改善できる。

そしてその次はコードの視認性の悪さである。現在操作しているオブジェクトを指し示す「this」や、引数を活用すればもっと短く整理されたコードで記述できたのではないかと考えている。

5　まとめ

当初の目的であった知っているシューティングゲームの挙動を一通り実装できるようになるという目的については、基本的なものでさえ参考にした東方の挙動を実現するのは難しかったためこれは実現できなかったということになる。

ゲーム制作の基礎を体感しゲームプログラミングの基礎を理解することについては達成できた。ゲームの作り始め方すら知らなかった段階からこのゲームが作れたことは誇るべきことだろう。

ゲームクリエイターの独創性を理解することについては少し達成出来た。東方原作者の ZUN 氏の凄みを微量ながら知った。

本実験の目的である新しい開発プラットフォームを使いこなせるようにする学習能力を身に着けることについてはいい経験になった。現にいま Javascript のプラグイン Three.js や OPEN GL 等のツールに抵抗なく触れることができている。

5.1　感想

今回のゲーム制作及びレポート執筆を終えてわかったことは、一連のプログラムを作って極力わかりやすいように報告書を作ることはかなりの労力を要する難しい作業なのだということである。表や画像を極力用いたがそれでも自分の意図が伝わってくれという気持ちである。加えて説明に約 40 ページも要した。もっと簡潔に短く書くべきだったと後悔している。しかし今回の講義は自分を一番成長させる有意義な講義であった。