ソースコードの説明

電気通信大学大学院
情報理工学研究科情報学専攻
中溝雄斗

まず、担当した Controller.java で作成したクラスとメンバ変数、メンバ関数の概要を列挙します。

- Main クラス
  Sounds オブジェクトや Model オブジェクト、View オブジェクト、AllController オブジェクトをそれぞれ生成し、ゲームを開始します。
- AllController クラス
  画面の更新頻度などのゲーム全体の制御や Player 用の Controller と CPU 用の Controller の 2 つの Controller クラスと Model クラス及び View クラスとの対応の管理を行います。CPUController オブジェクト及び PlayerController オブジェクトの生成もこのクラスで行います。
- CPUController クラス
  CPU の制御を行います。CPU の動作を制御するメソッドとしては updateCPU 関数と hardupdateCPU 関数の 2 種類を実装しました。
- PlayerController クラス
  キー入力の処理など Player の制御を行います。

AllController クラス、CpuController クラス、PlayerController クラスでそれぞれ実装されている主なメソッドのソースコードについて、以下で詳しく説明します。

第一に AllController クラスについて説明します。AllController クラスに実装されている主なメソッドの働きは以下のとおりです。なお、AllController クラスのソースコードはソースコード 1 に示しています。

- actionPerformed メソッド
  actionPerformed メソッドは AllController オブジェクトで動作している画面更新のタイミングを管理するタイマーと同期してのシーン遷移や CPU 動作の変更、生成されている CPU の総数の管理、そしてプレイヤーからのボタンによる入力に対する処理を行うメソッドです。
  本メソッドにおける処理について、順を追って説明します。まず 36 行目から 62 行目までが AllController オブジェクトで動作している画面更新のタイミングを管理するタイマーと同期して行う処理です。Model オブジェクトの Scene 変数が 1 である、つまりゲームシーンである場合には、Model オブジェクトのゲーム内時間の更新やゲーム内時間による CPU の動作の管理を行っています。また、これらの処理の結果、Model オブジェクトの Scene 変数が 2 に変化した、つまりゲーム終了と判断された場合には、その情報を View オブジェクトと共有しています。そして、これらすべての処理が終わった後に、画面の更新を行っています。次に 63 行目から 82 行目までがプレイヤーからのボタンによる入力に対する処理です。ボタンによる入力を処理するのはシーンがゲームシーンではない場合であるため、その条件を満たしている場合に CPU や Player の初期化などゲームの準備を行っています。その

後、Model オブジェクトと View オブジェクトのシーンの情報をともにゲームシーンの状態を変化させ、画面の更新を行っています。

Listing 1　AllController クラス

```
15  class AllController implements ActionListener {
16  protected Model model;
17  protected CPUController cpu;
18  protected PlayerController player;
19  protected View view;
20  private javax.swing.Timer timer;
21  protected JButton start;
22  protected JButton replay;
23  public AllController(Model model, View view) {
24      this.model = model;
25      this.view = view;
26      //初期化はモデルで行う
27      start=view.getPanel().getStartButton();
28      replay=view.getPanel().getReplayButton();
29      start.addActionListener(this);
30      replay.addActionListener(this);
31      timer = new javax.swing.Timer(50, this);
32      timer.start();
33  }
34
35  public void actionPerformed(ActionEvent e) {
36      if(e.getSource()==timer){
37          if(model.getScene()==1){
38              model.setTime(model.getTime()+1);
39              int time=model.getTime();
40              int max=model.getMaxTime();
41              if(time==max){
42                  model.setScene(2);
43              }else{
44                  player.action();
45                  //ゲーム後半用動作の分岐
46                  if(time<max/2){
47                      cpu.updateCPU();
48                  }else{
49                      int loop=model.getLoop();
50                      cpu.hardupdateCPU(loop++);
51                      if(loop>=model.getCount()){
52                          loop=0;
53                      }
54                      model.setLoop(loop);
55                  }
56              }
57              if(model.getScene()==2){
58                  view.getPanel().setflag(model.getScene());
59                  model.clearCPU();
60              }
61          }
62          view.getPanel().repaint();
63          this.view.setFocusable(true);
64      }else{
65          if(model.getScene()!=1){
66              model.initPlayer();
67              for(int i=0;i<10;i++){
```

```
68                  model.createCpu();
69              }
70              if(cpu==null){
71                  cpu=new CPUController(model);
72                  player=new PlayerController(model,view);
73              }else{
74                  player.init();
75              }
76              model.setLoop(0);;
77              model.setScene(1);
78              view.getPanel().setflag(model.getScene());
79              model.setCount(0);
80              model.setTime(0);
81              view.repaint();
82          }
83      }
84   }
85 }
```

第二に CPUController クラスについて説明します。CPUController クラスに実装されている主なメソッドの働きは以下のとおりです。なお、CPUController クラスのソースコードはソースコード 2 に示しています。

- updateCPU メソッド
  updateCPU メソッドは各 CPU の位置情報の更新、進行方向の画面外に出た場合や Player に食べられた場合の該当 CPU の削除、CPU と衝突時の Player のスコア及び HP の制御を行うメソッドです。
  本メソッドにおける処理について、順を追って説明します。まず 95 行目では各 CPU の位置情報を Speed 変数及び Direction 変数を用いて更新しています。次に 97 行目から 100 行目では CPU の位置情報をもとに、CPU の削除判定や実際の削除、削除した場合の新規 CPU の作成を行っています。そして 102 行目から 113 行目では Model オブジェクトによる当たり判定の結果をもとに CPU の削除判定や Player の HP 制御等を行っています。Model オブジェクトの checkCollision(i) 関数が 1 を返す、つまり該当 CPU が Player と衝突しておりかつ Player により捕食される場合には、Player のスコアの更新やサイズ調整を行った後に、捕食された CPU の削除と新規 CPU の作成を行い、Model オブジェクトの checkCollision(i) 関数が 2 を返す、つまり該当 CPU が Player と衝突しておりかつ Player により捕食されない場合には、Player の HP の更新とゲーム終了の判定を行っています。当たり判定は Controller で実装することも考えましたが、各担当者への負担の分散や Getter メソッド、Setter メソッドの呼び出し回数の増加などを考慮し、Model で実装することとしました。
- hardupdateCPU メソッド
  hardupdateCPU メソッドは上記の updateCPU に対して、ランダムな上下移動を追加したものです。本メソッドで updateCPU から追加された処理は以下のとおりです。まず 118 行目から 125 行目では各 CPU の上下移動の速度やその速度の更新周期をランダムに決定しています。ここでは、上下移動の速度の決定では Math.random() を、その速度の更新周期の決定では random.nextInt() を用いています。速度の更新周期の決定は Math.random() を用いた実装も可能でしたが、指定範囲の整数での乱数の生成であったため、random.nextInt() のほうが戻り値の型や範囲の可読性といった面で利点があると感じたため、random.nextInt() を用いて実装しました。次に 126 行目から 152 行目では updateCPU で行っている処理に加えて、各 CPU の上下方向の位置情報の更新や CPU が上下方向の画面外に出た場合の該当 CPU の削除を行っています。なお、上下方向の画面外に出た場合にはその後画面内に再度戻ってくる可能性も考えられますが、今回は画面外一度出た時点で削除するものとして実装しました。

Listing 2　CpuController クラス

```
87 class CPUController{
88     protected Model model;
89     public CPUController(Model m) {
90         model=m;
91     }
92     public void updateCPU(){
93         for(int i=0; i<model.getUOUUOs().size(); i++){
94             Cpu u=model.getUOUUOs().get(i);
95             u.setX(u.getX()+u.getSpeed()*u.getDirection());
96             //範囲外に言ったら消すように指示
97             if(u.getDirection()==-1&&u.getX()<0-u.getWidth()||u.getDirection()==1&&u.getX()>model.
                   getFrameWidth()){
98                 model.destroyCPU(i);
99                 model.createCpu();
100            }
101            //当たっているなら消すように指示
102            if(model.checkCollision(i)==1){
103                Player player=model.getPlayer();
104                player.setPoint((int)(player.getPoint()+model.getUOUUO(i).getPoint()));
105                model.resizePlayer();
106                model.destroyCPU(i);
107                model.createCpu();
108            }else if(model.checkCollision(i)==2){
109                model.getPlayer().setHP(model.getPlayer().getHP()-1);
110                if(model.getPlayer().getHP()<=0){
111                    model.setScene(2);
112                }
113            }
114        }
115    }
116
117    public void hardupdateCPU(int loop){
118        if(loop==0){
119            model.clearSpeedY();
120            for(int i=0; i<model.getUOUUOs().size(); i++){
121                model.addSpeedY((Math.random()-0.5));
122            }
123            Random random=new Random();
124            model.setCount(random.nextInt(37)+12);
125        }
126        for(int i=0; i<model.getUOUUOs().size(); i++){
127            Cpu u=model.getUOUUOs().get(i);
128            u.setX(u.getX()+u.getSpeed()*u.getDirection());
129
130            u.setY(u.getY()+u.getSpeed()*model.getSpeedY(i));
131            //範囲外に言ったら消すように指示
132            if(u.getDirection()==-1&&u.getX()<0-u.getWidth()||u.getDirection()==1&&u.getX()>model.
                   getFrameWidth()){
133                model.destroyCPU(i);
134                model.createCpu();
135            }else if(u.getY()<0-u.getHeight()||u.getY()>model.getFrameHeight()){
136                model.destroyCPU(i);
137                model.createCpu();
138            }
139            //当たっているなら消すように指示
140            if(model.checkCollision(i)==1){
141                Player player=model.getPlayer();
```

```
142                 player.setPoint((int)(player.getPoint()+model.getUOUO(i).getPoint()));
143                 model.resizePlayer();
144                 model.destroyCPU(i);
145                 model.createCpu();
146             }else if(model.checkCollision(i)==2){
147                 model.getPlayer().setHP(model.getPlayer().getHP()-1);
148                 if(model.getPlayer().getHP()<=0){
149                     model.setScene(2);
150                 }
151             }
152         }
153     }
154 }
```

第三に PlayerController クラスについて説明します。PlayerController クラスに実装されている主なメソッドの働きは以下のとおりです。なお、PlayerControllerr クラスのソースコードはソースコード 3 に示しています。

- keyPressed メソッド、keyTyped メソッド、keyReleased メソッド
  keyPressed、keyTyped、keyReleased の 3 つのメソッドはプレイヤーからのキー入力に対する処理を行うメソッドです。wasd 及び矢印キーの入力に合わせて、現在の入力状況を Player オブジェクトの move 配列に格納しています。なお、move 配列では move[0] に左右方向の入力情報を、move[1] に上下方向の入力情報を格納しています。
- action メソッド
  action メソッドは Player オブジェクトの move 配列、ratio 配列に格納されている情報に従って、Player の位置情報の更新を行うメソッドです。
  本メソッドにおける処理について、順を追って説明します。まず 251 行目から 270 行目では、左右方向の入力がない場合もしくは上下方向の入力がない場合にその方向の速度を 1 割ずつ小さくして、最終的には 0 としています。なお、このときは ratio 配列が左右方向及び上下方向へ最大の速さのどの程度の割合の速さでどちらの方向への速度なのかを格納しています。次に 271 行目から 280 行目では、左右方向もしくは上下方向に入力がある場合に入力の方向への速度を 2 割ずつ大きくし、その後左右方向への速度が 0 でない場合には Player の向きを速度の方向に合わせて設定しています。
  本メソッドでは、プレイヤーからのキー入力を速度を介して Player オブジェクトの座標に反映させることで、慣性が働いているような Player の動作を実装しています。
- init メソッド
  init メソッドは Player オブジェクトの move 配列、ratio 配列の初期化と Java のフォーカスの設定を行っています。

Listing 3　PlayerController クラス

```
157 class PlayerController implements KeyListener {
158     protected Model model;
159     protected View view;
160     public PlayerController(Model m, View view) {
161         model=m;
162         this.view=view;
163         this.view.setFocusable(true);
164         this.view.addKeyListener(this);
165     }
166     public void keyPressed(KeyEvent e){
```

```
167         // カーソルキーのイベントはで取得 keyPressed
168         int c=e.getKeyCode();
169         Player player=model.getPlayer();
170         switch (c) {
171             case KeyEvent.VK_LEFT: // ←
172             player.setMove(-1, 0);
173             break;
174             case KeyEvent.VK_RIGHT: // →
175             player.setMove(1, 0);
176             break;
177             case KeyEvent.VK_UP: // ↑
178             player.setMove(-1, 1);
179             break;
180             case KeyEvent.VK_DOWN: // ↓
181             player.setMove(1, 1);
182             break;
183             case KeyEvent.VK_A: // A
184             player.setMove(-1, 0);
185             break;
186             case KeyEvent.VK_D: // D
187             player.setMove(1, 0);
188             break;
189             case KeyEvent.VK_W: // W
190             player.setMove(-1, 1);
191             break;
192             case KeyEvent.VK_S: // S
193             player.setMove(1, 1);
194             break;
195         }
196     }
197     public void keyTyped(KeyEvent e) {}
198     public void keyReleased(KeyEvent e){
199         int c=e.getKeyCode();
200         Player player=model.getPlayer();
201         switch (c) {
202             case KeyEvent.VK_LEFT: // ←
203             if(player.getMove(0)==-1){
204                 player.setMove(0, 0);
205             }
206             break;
207             case KeyEvent.VK_RIGHT: // →
208             if(player.getMove(0)==1){
209                 player.setMove(0, 0);
210             }
211             break;
212             case KeyEvent.VK_UP: // ↑
213             if(player.getMove(1)==-1){
214                 player.setMove(0, 1);
215             }
216             break;
217             case KeyEvent.VK_DOWN: // ↓
218             if(player.getMove(1)==1){
219                 player.setMove(0, 1);
220             }
221             break;
222             case KeyEvent.VK_A: // A
223             if(player.getMove(0)==-1){
224                 player.setMove(0, 0);
225             }
```

```
            break;
            case KeyEvent.VK_D: // D
            if(player.getMove(0)==1){
                player.setMove(0, 0);
            }
            break;
            case KeyEvent.VK_W: // W
            if(player.getMove(1)==-1){
                player.setMove(0, 1);
            }
            break;
            case KeyEvent.VK_S: // S
            if(player.getMove(1)==1){
                player.setMove(0, 1);
            }
            break;
        }
    }

    public void action() {
        Player player=model.getPlayer();
        int move_x=player.getMove(0);
        int move_y=player.getMove(1);
        double ratio_x=player.getRatio(0);
        double ratio_y=player.getRatio(1);
        if(move_x==0){
            if(ratio_x>0){
                ratio_x=ratio_x-0.1;
            }else if(ratio_x<0){
                ratio_x=ratio_x+0.1;
            }
            if(Math.abs(ratio_x)<0.1){
                ratio_x=0;
            }
        }
        if(move_y==0){
            if(ratio_y>0){
                ratio_y=ratio_y-0.1;
            }else if(ratio_y<0){
                ratio_y=ratio_y+0.1;
            }
            if(Math.abs(ratio_y)<0.1){
                ratio_y=0;
            }
        }
        if(Math.abs(ratio_x)<1.0||ratio_x*(double)move_x<0){
            ratio_x+=0.2*(double)move_x;
        }
        if(Math.abs(ratio_y)<1.0||ratio_y*(double)move_y<0){
            ratio_y+=0.2*(double)move_y;
        }

        if(move_x!=0){
            player.setDirection(move_x);
        }
        double x=player.getX()+ratio_x*player.getSpeed();
        double y=player.getY()+ratio_y*player.getSpeed();
        player.setMove(move_x,0);
        player.setMove(move_y,1);
```

```
285         player.setRatio(ratio_x,0);
286         player.setRatio(ratio_y,1);
287         if(x<0){
288             player.setX(0.0);
289         }else if(x>model.getFrameWidth()-player.getWidth()){
290             player.setX(model.getFrameWidth()-player.getWidth());
291         }else{
292             player.setX(x);
293         }
294         if(y<0){
295             player.setY(0.0);
296         }else if(y>model.getFrameHeight()-player.getHeight()){
297             player.setY(model.getFrameHeight()-player.getHeight());
298         }else{
299             player.setY(y);
300         }
301     }
302 
303     public void init(){
304         Player player=model.getPlayer();
305         player.setMove(0,0);
306         player.setMove(0,1);
307         player.setRatio(0.0,0);
308         player.setRatio(0.0,1);
309         this.view.setFocusable(true);
310     }
311 }
```