

令和6年度 メディア情報学プログラミング演習
グループプログラミング レポート
料理提供ゲーム「MiniCook」

2025年2月17日

学科	情報理工学域
クラス	J1
グループ番号	26
2210259	米谷祐希
2210730	鈴木早紀
2210743	吉田陽音

1 概要説明

このゲームは、レストランで働くプレイヤーが、制限時間内に料理を作るゲームである。以下の料理提供までの手順を繰り返すことでポイントを獲得し、制限時間終了時にスコアとランクが表示される。

1. オーダーの確認

まず、画面上部にランダムにオーダーが提示される。オーダーには、使う食材と調理方法が記載されている。各オーダーにはそれぞれ制限時間が設定されており、残り時間はオーダー上のゲージにリアルタイムに表示される。

2. 食材の調理

次に、オーダーに記載されている食材を、各食材ボックスから取り出す。各食材を持ったまま、各調理器具の前でアクションボタンを押すことで、食材が加工される。

3. 料理の完成と提供

料理は、加工された食材とお皿を組み合わせることで完成する。それらを組み合わせて料理ができあがれば、提供口に置くことで提供となり、オーダーと一致しているか判定される。一致していれば加点、間違っていれば減点となる。

また、ゲームは3画面に分かれており、スタート画面、ゲーム画面、リザルト画面がある。また、各画面や各動作にはBGMや効果音がついている。操作はキーボードのA,S,D,W,J,K,Spaceキーを用いている。

作業はGitHubを用い保存・共有を行った。米谷がModelと全体の管理、鈴木がView、吉田がControllerを主に担当したが、最終的には各自の担当領域を超えて協力しながら取り組んだ。

2 設計方針

図1にクラス図を示す。MVCモデルで設計した。

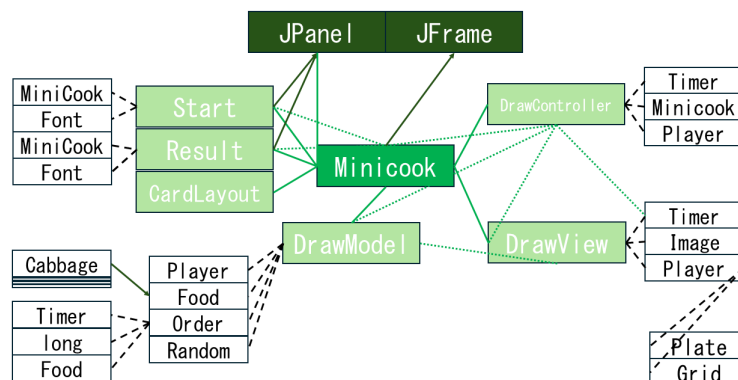


図 1: クラス図

3 プログラムの説明

以下にクラスとその説明を示す。

- MiniCook
- Model
 - Food
 - Order
- View
 - Timer
 - Image
 - Player
 - * Plate
 - * Grid
- Controller
- Start
- Result
- CardLayout
- AudioManager

4 実行例

スタート画面

実行すると始めにこの画面 (a) が現れる。スタートボタンを押すとゲーム画面：スタート時 (c) になる。

リザルト画面

ゲーム終了後はこのリザルト画面 (b) になる。スコアによってランクが星の数で表される。

ゲーム画面：スタート時

スタート時の画面 (c) では、食材などは何もなく、オーダーが 1 つ入るところから開始される。上部にはオーダー、中央にはゲーム部分、下部にはスコアと制限時間を表示している。

ゲーム画面：オーダー

画面上部のオーダー (d) では、完成品、必要な食材、加工方法、残り時間が示されている。

ゲーム画面：加工前

加工前の食材 (e) をボックスから取り出す。

ゲーム画面：加工後

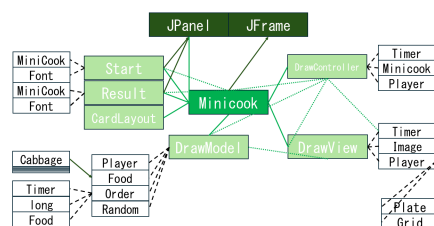
調理器具でアクションを行うと加工される。

ゲーム画面：組み合わせ

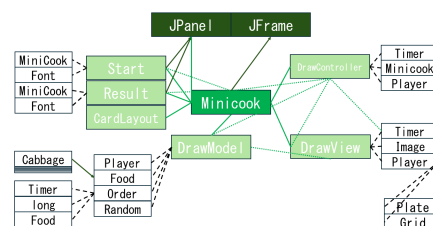
皿の上に各食材を載せると画像がそれに伴い完成品となる。

ゲーム画面：提供

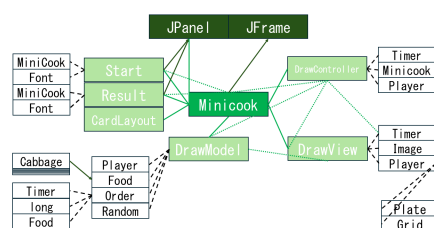
完成した料理を提供口に置くと、ホールスタッフが取りに来る。



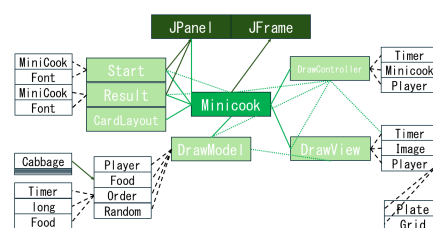
(a) スタート画面



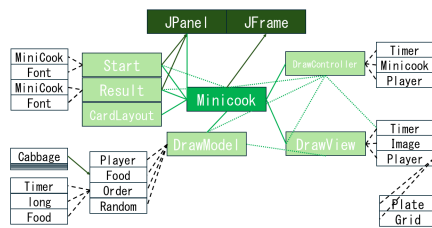
(b) リザルト画面



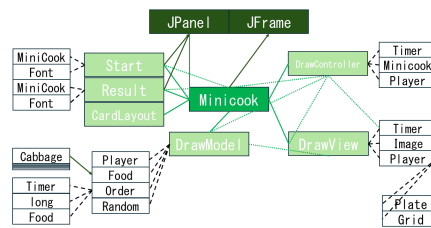
(c) ゲーム画面：スタート時



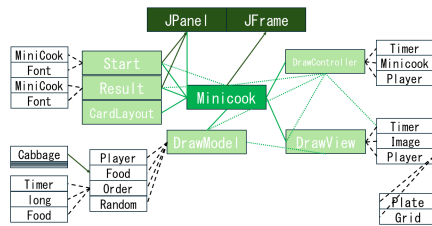
(d) ゲーム画面：オーダー



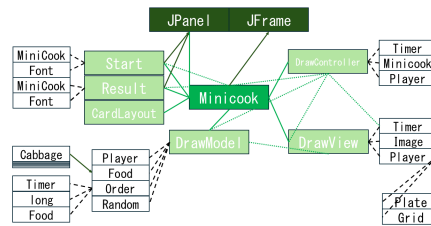
(e) ゲーム画面：加工前



(f) ゲーム画面：加工後



(g) ゲーム画面：組み合わせ後



(h) ゲーム画面：提供

5 考察

予定していた以上のものが完成した。

6 感想

(米谷祐希)

(鈴木早紀)

(吉田陽音)

付録1：操作マニュアル

(ストーリー)

キミはレストランのキッチンで働いているぞ！制限時間内にオーダー通りの料理を作れ！目指せ高得点！！

(実行方法)

「Java MiniCook」でゲームが開始する。

(操作方法)

このゲームはキーボードでキャラクターを操作する。図2にキー操作を示す。W,S,A,Dで上下左右を操作し、Jで取る、Kで置く、スペースキーでアクションを行う。



図 2: キーボード操作方法

(遊び方)

1. スタート

スタートボタンを押すとゲームが開始する。

2. オーダーの確認

まず、画面上部にランダムにオーダーが提示される。オーダーには、使う食材と調理方法が記載されている。各オーダーにはそれぞれ制限時間が設定されており、残り時間はオーダー上のゲージにリアルタイムに表示される。

3. 食材の調理

次に、オーダーに記載されている食材を、各食材ボックスから取り出す。各食材を持ったまま、各調理器具の前でアクションボタンを押すことで、食材が加工される。

4. 料理の完成と提供

料理は、加工された食材とお皿を組み合わせることで完成する。それらを組み合わせて料理ができあがれば、提供口に置くことで提供となり、オーダーと一致しているか判定される。一致していれば加点、間違っていれば減点となる。

5. リザルト

制限時間がなくなるとリザルト画面に遷移する。スコアとランクが表示される。リザルトを押せばもう一度ゲームが開始する。

- メニュー一覧

- － マグロ握り
- － イカ握り
- － 海鮮丼
- － カップパ卷
- － 鉄火巻き
- － サラダ

- 調理器具一覧

- － 包丁
- － 鍋

- 食材一覧

- － マグロ
- － イカ
- － 米
- － 海苔
- － キャベツ
- － トマト
- － キュウリ

付録2：プログラムリスト

以下にプログラムリスト全体を記述する。

- MiniCook

```

1 import javax.swing.*;
2 import java.awt.*;
3
4 class MiniCook extends JFrame {
5     DrawModel model;
6     DrawView view;
7     DrawController cont;
8     AudioManager audio;
9     Result resultScreen;
10
11     private CardLayout cardLayout;
12     private JPanel cardPanel;
13
14     public MiniCook() {
15         System.out.printf("\n---Start---\n\n"); 見やすいように//
16         model = new DrawModel();
17         view = new DrawView(model);
18         cont = new DrawController(model, view, this);
19         audio = new AudioManager();
20
21         model.getPlayer().setController(cont);
22         model.getPlayer().setView(view);
23         view.setController(cont);
24         view.addKeyListener(cont);
25
26         this.setBackground(Color.WHITE);
27         this.setTitle("MiniCookey");
28         this.setSize(1016, 950);
29         this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
30         setLocationRelativeTo(null);
31
32         // カードレイアウトの設定
33         cardLayout = new CardLayout();
34         cardPanel = new JPanel(cardLayout);
35
36         // 各画面の追加
37         Start startScreen = new Start(this);
38         resultScreen = new Result(this);
39
40         cardPanel.add(startScreen, "start");
41         cardPanel.add(resultScreen, "result");
42
43         // ゲーム画面
44         JPanel gamePanel = new JPanel(new BorderLayout());
45         gamePanel.add(view, BorderLayout.CENTER);
46
47         cardPanel.add(gamePanel, "game");
48
49         add(cardPanel);
50         cardLayout.show(cardPanel, "start");
51     }
52
53     // スタート画面からゲーム画面に切り替える
54     public void startGame() {
55         cardLayout.show(cardPanel, "game");
56         cont.startGame();
57         //audio.playBGM("./sound/music_background2.wav");
58
59         // キーボード入力を受け取るためにフォーカスを設定
60         view.requestFocusInWindow();
61     }
62
63     // ゲーム終了時にリザルト画面を表示する
64     public void showResult() {
65         audio.stopBGM();
66         System.out.printlnリザルト画面を表示します。("");
67         resultScreen.updateScore(model.score);
68         cardLayout.show(cardPanel, "result");
69     }
70
71     // リザルト画面からもう一度プレイ
72     public void restartGame() {
73         audio.playBGM("./sound/music_background2.wav");

```



```

74     model.reset(); // ゲームデータをリセット（必要なら実装）
75     startGame(); // ゲームを開始
76 }
77
78 public static void main(String[] args) {
79     new MiniCook().setVisible(true);
80 }
81 }

```

- Model

```

1  import javax.swing.*;
2  import java.awt.*;
3  import java.awt.event.*;
4  import java.util.Random;
5
6  class DrawModel extends JPanel {
7      private final int xsize = 16; // グリッドの幅
8      private final int ysize = 9; // グリッドの高さ
9      private final int cellSize = 60; // マスの大きさ 1
10     protected Grid[][] grid;
11     private Player player;
12     private Food food;
13     public int score;
14     //private static DrawModel instance;
15     public Order[] orders; //を入れる配列 order
16     private int gameTime;
17
18     public DrawModel() {
19         //System.out.println("DrawModel instance: " + this);
20         gameTime = 120/*3*60 + 30*/; // ゲーム時間は分秒//330 Yoshida
21         score = 0;
22         orders = new Order[5];
23         for(int i=0; i<5; i++){
24             orders[i] = null;
25         }
26         grid = new Grid[xsize][ysize];
27         //imageGrid = new int[xsize][ysize];
28         for (int i = 0; i < xsize; i++) {
29             for (int j = 0; j < ysize; j++) {
30                 grid[i][j] = new Grid(i, j);
31                 //imageGrid[i][j] = '\0';
32                 if (i == 0 || j == 0 || i == xsize - 1 || j == ysize - 1) {
33                     grid[i][j].wall = true; // 外周を壁に設定
34                 }
35             }
36         }
37         player = new Player(2, 2, this, grid);
38
39         grid[3][3].obstacle = true;
40         grid[4][3].obstacle = true;
41         grid[5][3].obstacle = true;
42         grid[6][3].obstacle = true;
43         grid[9][5].obstacle = true;
44         grid[10][5].obstacle = true;
45         grid[11][5].obstacle = true;
46         grid[12][5].obstacle = true;
47
48         grid[4][5].foodBox = 1;
49         grid[4][5].obstacle = true;
50         grid[4][5].tool = 2;
51
52         grid[5][5].foodBox = 2;
53         grid[5][5].obstacle = true;
54         grid[5][5].tool = 4;
55
56         grid[6][5].foodBox = 3;
57         grid[6][5].obstacle = true;
58         grid[6][5].tool = 5;
59
60         grid[9][3].foodBox = 4;
61         grid[9][3].obstacle = true;
62         grid[9][3].tool = 6;
63
64         grid[10][3].foodBox = 5;
65         grid[10][3].obstacle = true;
66         grid[10][3].tool = 7;
67
68         grid[11][3].foodBox = 6;
69         grid[11][3].obstacle = true;

```

```

70     grid[11][3].tool = 8;
71
72     grid[12][3].foodBox = 7;
73     grid[12][3].obstacle = true;
74     grid[12][3].tool = 9; カウンターを設置
75
76     // Yoshida
77     grid[7][8].wall = true; 元々壁だったところをカウンターにしたい//
78     grid[7][8].isCounter = true;
79     grid[8][8].wall = true; 元々壁だったところをカウンターにしたい//
80     grid[8][8].isCounter = true;
81
82     grid[0][3].tool = ナイフ 1;//
83     grid[0][4].tool = ナイフ 1;//
84     grid[0][5].tool = ナイフ 1;//
85     grid[15][3].tool = ナイフ 1;//
86     grid[15][4].tool = ナイフ 1;//
87     grid[15][5].tool = ナイフ 1;//
88
89     grid[10][0].tool = なべ 10;//
90     grid[11][0].tool = なべ 10;//
91     grid[12][0].tool = なべ 10;//
92
93     grid[3][0].tool = フライパン 12;//
94     grid[4][0].tool = フライパン 12;//
95     grid[5][0].tool = フライパン 12;//
96
97     grid[3][5].plateBox = true;
98     grid[3][5].obstacle = true;
99     grid[3][5].tool = 3;
100
101     grid[7][0].plateBox = true;
102     grid[7][0].tool = 3; 皿ボックス//
103     grid[8][0].plateBox = true;
104     grid[8][0].tool = 3; 皿ボックス//
105
106     grid[0][1].tool=13;
107     grid[0][7].tool=13;
108     grid[15][1].tool=13;
109     grid[15][7].tool=13;
110
111     grid[6][8].tool = 14;
112     grid[9][8].tool = 14;
113 }
114
115 public Grid[] getGrid() {
116     return grid;
117 }
118
119 public int[] getFieldSize() {
120     return new int[]{xsize, ysize};
121 }
122
123 public int getCellSize() {
124     return cellSize;
125 }
126
127 public Player getPlayer() {
128     return player;
129 }
130
131 public Food getFood() {
132     return food;
133 }
134
135 public void movePlayer(int dx, int dy) {
136     player.move(dx, dy, grid);
137 }
138
139 public void printInfo(){
140     System.out.printlnデバッグ用情報("<>");
141     // デバッグ用
142     System.out.println("配列の状態 orders:");
143     for (int i = 0; i < 3; i++) {
144         if (orders[i] != null) {
145             System.out.println("orders[" + i + "]: " + orders[i].orderName);
146         } else {
147             System.out.println("orders[" + i + "]: null");
148         }
149     }
150 }

```

```

149     }
150
151     public void generateOrder() {
152         String[] menu={"salad","tekkamaki","kappamaki","tunanigiri","ikanigiri","kaisendon"};
153         int num_menu=6;
154         Random random=new Random();
155         for (int i = 0; i < orders.length; i++) {
156             if (orders[i] == null) {
157                 System.out.println("orders[" + i + "] はです null 新しいオーダーを生成します");
158                 String randommenu=menu[random.nextInt(num_menu)];
159                 orders[i] = new Order(randommenu, i , this);
160                 //orders[i] = new Order("tekkamaki", i , this);
161                 System.out.println生成されたオーダー (" : " + orders[i].orderName);
162                 break;
163             } else {
164                 System.out.println("orders[" + i + "] は存在しています : " + orders[i].orderName);
165             }
166         }
167     }
168     public Order matchOrder(Plate plate) {
169         for (Order order : orders) {
170             if (order != null && plate.matchesOrder(order)==true) {
171                 System.out.println(order.orderName + が完成!");
172                 return order;
173             }
174         }
175         return null;
176     }
177     public Order getOrder(int index) {
178         if(index < orders.length || index >= 0)return orders[index];
179         else return null;
180     }
181     public void scoreUp(Order order){
182         switch(order.orderName){
183             case "salad" : score += 50;
184             case "tekkamaki" : score += 50;
185             case "kappamaki" : score += 50;
186             case "tunanigiri" : score += 30;
187             case "ikanigiri" : score += 30;
188             case "kaisendon" : score += 60;
189         }
190         System.out.println("scoreUp()が呼ばれました");これは料理が提供された瞬間の方がいいかも知れない
191
192         //
193         for(int i=0; i<orders.length; i++){
194             //if(orders[i].orderName == order.orderName)
195             if(orders[i] == order){ こっちのほうが重複した料理があったときに対応できる//
196                 removeOrder(i);
197                 return;
198             }
199         }
200     public void scoreDown(Order order){
201         System.out.println("socreDown() called");
202         if(score == 0) return;
203         if(order == null){
204             score -= 50;
205             if(score < 0) score = 0;
206             return;
207         }
208         switch(order.orderName){
209             case "salad" : score -= 30;
210             case "tekkamaki" : score -= 30;
211             case "kappamaki" : score -= 30;
212             case "tunanigiri" : score -= 20;
213             case "ikanigiri" : score -= 20;
214             case "kaisendon" : score -= 30;
215         }
216         if(score < 0) score = 0;これは料理が提供された瞬間の方がいいかも知れない
217
218         //それな
219         // てかこれ失敗したときだからならんくね trueKome
220         for(int i=0; i<orders.length; i++){
221             if(orders[i].orderName == order.orderName){
222                 removeOrder(i);
223                 return;
224             }
225         }

```

```

226     }
227     public void removeOrder(int i){
228         System.out.println("get =" + i);
229         if (i >= 0 && i < orders.length && orders[i] != null) {
230             orders[i].cancelTimer(); // タイマーの停止
231             System.out.println注文 (" " + orders[i].orderName + " を削除します。");
232             orders[i] = null;
233             formatOrder();
234         }
235     }
236     private void formatOrder(){ //を前に詰めていくメソッド order
237         for(int s = 0; s < orders.length - 1; s++){
238             for(int t = s; t < orders.length - 1; t++){
239                 if(orders[t] == null) {
240                     orders[t] = orders[t+1];
241                     if(orders[t] != null) { orders[t].orderIndex = t; }
242                     orders[t+1] = null;
243                 }
244             }
245         }
246     } 以下時間に関わるメソッド
247
248     // Yoshida
249     public int getGameTime(){
250         return gameTime;
251     }
252
253     public void decreaseTime(){
254         if(gameTime > 0){
255             gameTime--;
256         }
257     }
258
259     public void reset() {
260         //System.out.println("DrawModel instance: " + this);
261         gameTime = 120/*3*60 + 30*/;
262         score = 0;
263         for(int i=0; i<5; i++){
264             //orders[i].cancelTimer();
265             orders[i] = null;
266         }
267         //grid = new Grid[xsize][ysize];
268         for (int i = 0; i < xsize; i++) {
269             for (int j = 0; j < ysize; j++) {
270                 //grid[i][j] = new Grid(i, j);
271                 //imageGrid[i][j] = '\0';
272                 grid[i][j].food = null;
273                 grid[i][j].plate = null;
274                 grid[i][j].isPlatePlaced = false;
275                 if (i == 0 || j == 0 || i == xsize - 1 || j == ysize - 1) {
276                     grid[i][j].wall = true; // 外周を壁に設定
277                 }
278             }
279         }
280         grid[3][3].obstacle = true;
281         grid[4][3].obstacle = true;
282         grid[5][3].obstacle = true;
283         grid[6][3].obstacle = true;
284         grid[9][5].obstacle = true;
285         grid[10][5].obstacle = true;
286         grid[11][5].obstacle = true;
287         grid[12][5].obstacle = true;
288
289         grid[4][5].foodBox = 1;
290         grid[4][5].obstacle = true;
291         grid[4][5].tool = 2;
292
293         grid[5][5].foodBox = 2;
294         grid[5][5].obstacle = true;
295         grid[5][5].tool = 4;
296
297         grid[6][5].foodBox = 3;
298         grid[6][5].obstacle = true;
299         grid[6][5].tool = 5;
300
301         grid[9][3].foodBox = 4;
302         grid[9][3].obstacle = true;
303         grid[9][3].tool = 6;
304
305         grid[10][3].foodBox = 5;
306         grid[10][3].obstacle = true;

```

```

307     grid[10][3].tool = 7;
308
309     grid[11][3].foodBox = 6;
310     grid[11][3].obstacle = true;
311     grid[11][3].tool = 8;
312
313     grid[12][3].foodBox = 7;
314     grid[12][3].obstacle = true;
315     grid[12][3].tool = 9; カウンターを設置
316
317     // Yoshida
318     grid[7][8].wall = true; 元々壁だったところをカウンターにしたい//
319     grid[7][8].isCounter = true;
320     grid[8][8].wall = true; 元々壁だったところをカウンターにしたい//
321     grid[8][8].isCounter = true;
322
323     grid[0][3].tool = ナイフ 1;//
324     grid[0][4].tool = ナイフ 1;//
325     grid[0][5].tool = ナイフ 1;//
326     grid[15][3].tool = ナイフ 1;//
327     grid[15][4].tool = ナイフ 1;//
328     grid[15][5].tool = ナイフ 1;//
329
330     grid[10][0].tool = なべ 10;//
331     grid[11][0].tool = なべ 10;//
332     grid[12][0].tool = なべ 10;//
333
334     grid[3][0].tool = フライパン 12;//
335     grid[4][0].tool = フライパン 12;//
336     grid[5][0].tool = フライパン 12;//
337
338     grid[3][5].plateBox = true;
339     grid[3][5].obstacle = true;
340     grid[3][5].tool = 3;
341
342     grid[7][0].plateBox = true;
343     grid[7][0].tool = 3; 皿ボックス//
344     grid[8][0].plateBox = true;
345     grid[8][0].tool = 3; 皿ボックス//
346
347     grid[0][1].tool=13;
348     grid[0][7].tool=13;
349     grid[15][1].tool=13;
350     grid[15][7].tool=13;
351
352     grid[6][8].tool = 14;
353     grid[9][8].tool = 14;
354 }
355 }

```

- View

```

1  import javax.swing.*;
2  import java.awt.*;
3  import java.awt.event.*;
4  import java.awt.image.BufferedImage;
5  import java.awt.image.ImageObserver;
6  import java.io.File;
7  import java.io.IOException;
8
9
10 import java.util.concurrent.*;
11
12 class DrawView extends JPanel {
13
14     //int orderXAnim = 2000;
15     int speed = 20;
16     static final double easingFactor = 0.2;
17     static final double easingFactorText = 0.2;
18     double scoreAnim = 0;
19
20     private BufferedImage cacheFloorAll = null;
21
22     private Timer drawTimer60fps; //60で Hzpaintcomponent()を呼び出すために使う Kome
23     protected DrawModel model;
24     private DrawController cont;
25     Grid[] grid;
26     int[] size;
27     final int cellSize;

```

```
28
29 private Image ImagePlayer;
30 private Image imgPlayerUp;
31 private Image imgPlayerLeft;
32 private Image imgPlayerDown;
33 private Image imgPlayerRight;
34 private Image imgErrorBlock;
35 private Image imgKnife;
36 private Image imgBoil;
37 private Image imgBoilRice;
38 private Image imgPlateBox;
39 private Image imgPlate;
40 private Image imgPan;
41 private Image imgCabbageBox;
42 private Image imgCabbage;
43 private Image imgCabbageCut;
44 private Image imgTomatoBox;
45 private Image imgTomato;
46 private Image imgTomatoCut;
47 private Image imgCucumberBox;
48 private Image imgCucumber;
49 private Image imgCucumberCut;
50 private Image imgCabTom;
51 private Image imgCabCuc;
52 private Image imgTomCuc;
53 private Image imgCabTomCuc;
54 private Image imgRiceBox;
55 private Image imgRice;
56 private Image imgRiceBoil;
57 private Image imgTunaBox;
58 private Image imgTuna;
59 private Image imgTunaCut;
60 private Image imgSquidBox;
61 private Image imgSquid;
62 private Image imgSquidCut;
63 private Image imgSeaweedBox;
64 private Image imgSeaweed;
65 private Image imgRicTun;
66 private Image imgRicSqu;
67 private Image imgRicSea;
68 private Image imgRicCuc;
69 private Image imgTunSea;
70 private Image imgTunSqu;
71 private Image imgCucSea;
72 private Image imgRicCucSea;
73 private Image imgRicTunSea;
74 private Image imgRicTunSqu;
75
76 private Image imgTrash;
77
78
79 private Image[] imgCounter = new Image[5];
80 private Image orderPaper;
81 private Image imgKnifeBlack;
82 private Image imgBoilBlack;
83 private Image imgFloor1;
84 private Image imgFloor2;
85 private Image imgFloor3;
86 private Image imgTable;
87 private Image imgSampleSalad;
88
89 private Image imgA;
90 private Image imgB;
91 private Image imgC;
92 private Image imgF1;
93 private Image imgF2;
94 private Image imgF3;
95
96
97 private Image testWall;
98 private Image sideWall;
99 private Image longShadow;
100
101 private Image imgWaiterUp;
102 private Image imgWaiterDown;
103
104
105 private Image imgFire;
106
107 private Image imgUIBG;
108 private Image imgCoin;
```

```

109 private Image imgTimer;
110 private Image imgCandle;
111
112
113 Player player;
114 static final int headerBlank = 220;
115 static final int footerBlank = 300;
116 static final int rightBlank = 20;
117 static final int leftBlank = 60;
118 double playerSpeed;
119
120 Waiter[] waiters = new Waiter[5];
121
122 private ScheduledExecutorService executor;
123 private int frameCount = 0; // フレーム数をカウント
124 private double fps = 0.0; // 計算したを格納FPS
125 private long lastTime = System.nanoTime(); // 前回の時間
126 private static final long FPS_UPDATE_INTERVAL = 100_000_000; // 100 (ナノ秒) ms
127 int passedFlame = 0; 全体の経過フレーム、様々なアニメーションにつかう//
128 int flameScoreGet = 0;
129 int getScore = 0;
130
131
132 //public boolean moving = true;
133 private Font customFont;
134 public DrawView(DrawModel m) {画像読み込み
135     {
136         imgPlayerUp = new ImageIcon("img/test/ghost_up.png").getImage();
137         imgPlayerLeft = new ImageIcon("img/test/ghost_left.png").getImage();
138         imgPlayerDown = new ImageIcon("img/test/ghost_down.png").getImage();
139         imgPlayerRight = new ImageIcon("img/test/ghost_right.png").getImage();
140         //imgErrorBlock = new ImageIcon("img/error_image.png").getImage();
141         imgErrorBlock = new ImageIcon("img/miss.png").getImage(); //皿とツール
142
143         //
144         imgKnife = new ImageIcon("img/knife.png").getImage();
145         imgBoil = new ImageIcon("img/boil.png").getImage();
146         imgBoilRice = new ImageIcon("img/rice_boil.png").getImage();
147         imgPlateBox = new ImageIcon("img/plate_box.png").getImage();
148         imgPlate = new ImageIcon("img/plate.png").getImage();
149         imgPan = new ImageIcon("img/pan.png").getImage();
150
151         imgCabbageBox = new ImageIcon("img/cabbage_box.png").getImage();
152         imgCabbage = new ImageIcon("img/cabbage.png").getImage();
153         imgCabbageCut = new ImageIcon("img/cabbage_cut.png").getImage();
154
155         imgTomatoBox = new ImageIcon("img/tomato_box.png").getImage();
156         imgTomato = new ImageIcon("img/tomato.png").getImage();
157         imgTomatoCut = new ImageIcon("img/tomato_cut.png").getImage();
158
159         imgCucumberBox = new ImageIcon("img/cucumber_box.png").getImage();
160         imgCucumber = new ImageIcon("img/cucumber.png").getImage();
161         imgCucumberCut = new ImageIcon("img/cucumber_cut.png").getImage();
162
163         imgCabTom = new ImageIcon("img/cab_tom.png").getImage();
164         imgCabCuc = new ImageIcon("img/cab_cuc.png").getImage();
165         imgTomCuc = new ImageIcon("img/tom_cuc.png").getImage();
166         imgCabTomCuc = new ImageIcon("img/cab_tom_cuc.png").getImage();
167
168         imgRiceBox = new ImageIcon("img/rice_box.png").getImage();
169         imgRice = new ImageIcon("img/rice.png").getImage();
170         imgRiceBoil = new ImageIcon("img/rice_boil2.png").getImage();
171
172         imgTunaBox = new ImageIcon("img/tuna_box.png").getImage();
173         imgTuna = new ImageIcon("img/tuna.png").getImage();
174         imgTunaCut = new ImageIcon("img/tuna_cut.png").getImage();
175
176         imgSquidBox = new ImageIcon("img/squid_box.png").getImage();
177         imgSquid = new ImageIcon("img/squid.png").getImage();
178         imgSquidCut = new ImageIcon("img/squid_cut.png").getImage();
179
180         imgSeaweedBox = new ImageIcon("img/seaweed_box.png").getImage();
181         imgSeaweed = new ImageIcon("img/seaweed.png").getImage();
182
183         imgRicTun = new ImageIcon("img/ric_tun.png").getImage();
184         imgRicSqu = new ImageIcon("img/ric_squ.png").getImage();
185         imgRicSea = new ImageIcon("img/ric_sea.png").getImage();
186         imgRicCuc = new ImageIcon("img/ric_cuc.png").getImage();
187         imgTunSea = new ImageIcon("img/tun_sea.png").getImage();
188         imgTunSqu = new ImageIcon("img/tun_squ.png").getImage();
189         imgCucSea = new ImageIcon("img/cuc_sea.png").getImage();

```

```

190 imgRicCucSea = new ImageIcon("img/ric_cuc_sea.png").getImage();
191 imgRicTunSea = new ImageIcon("img/ric_tun_sea.png").getImage();
192 imgRicTunSqu = new ImageIcon("img/ric_tun_squ.png").getImage();
193
194
195
196 imgCounter[0] = new ImageIcon("img/test/counter1.png").getImage();
197 imgCounter[1] = new ImageIcon("img/test/counter2.png").getImage();
198 imgCounter[2] = new ImageIcon("img/test/counter3.png").getImage();
199 imgCounter[3] = new ImageIcon("img/test/counter4.png").getImage();
200 imgCounter[4] = new ImageIcon("img/test/counter5.png").getImage();
201 orderPaper = new ImageIcon("img/order_paper_short.png").getImage();
202 imgKnifeBlack = new ImageIcon("img/knife_black.png").getImage();
203 imgBoilBlack = new ImageIcon("img/boil_black.png").getImage();
204
205 imgTrash = new ImageIcon("img/trash.png").getImage();
206
207 imgFloor1 = new ImageIcon("img/floor1.jpg").getImage();
208 imgFloor2 = new ImageIcon("img/floor2.jpg").getImage();
209 imgFloor3 = new ImageIcon("img/floor3.png").getImage();
210 imgA = new ImageIcon("img/test/B.png").getImage();
211 imgB = new ImageIcon("img/test/D_long.png").getImage();
212 imgC = new ImageIcon("img/test/C.jpg").getImage();
213 imgF1 = new ImageIcon("img/test/floor_a_4.png").getImage();
214 imgF2 = new ImageIcon("img/test/floor_b_4.png").getImage();
215 imgF3 = new ImageIcon("img/test/floor_c_3.png").getImage();
216
217 imgTable = new ImageIcon("img/table.png").getImage();
218
219 imgSampleSalad = new ImageIcon("img/cab_tom_cuc.png").getImage();
220
221 imgFire = new ImageIcon("img/fires.png").getImage();
222
223
224 imgUIBG = new ImageIcon("img/ui_background.png").getImage();
225 imgCoin = new ImageIcon("img/coin.png").getImage();
226 imgTimer = new ImageIcon("img/timer.png").getImage();
227
228 testWall = new ImageIcon("img/test/wallpaper_11.png").getImage();
229 sideWall = new ImageIcon("img/test/wall_side.png").getImage();
230 imgWaiterUp = new ImageIcon("img/test/ghost_up.png").getImage();
231 imgWaiterDown = new ImageIcon("img/test/ghost_down.png").getImage();
232 longShadow = new ImageIcon("img/long_shadow.png").getImage();
233
234 imgCandle = new ImageIcon("img/test/candle.png").getImage();
235 }
236 model = m;
237 this.setFocusable(true);
238 this.setDoubleBuffered(true);
239 player = model.getPlayer();
240 grid = model.getGrid();
241 size = model.getFieldSize();
242 cellSize = model.getCellSize();
243 loadCustomFont();
244
245
246
247 /*
248 executor.scheduleAtFixedRate(() -> {
249     SwingUtilities.invokeLater(this::repaint); // スレッドで描画 Swing
250 }, 0, 50, TimeUnit.MILLISECONDS);
251 */
252
253 executor = Executors.newScheduledThreadPool(1); //60での描画を開始 fps
254 executor.scheduleAtFixedRate(() -> {
255     long currentTime = System.nanoTime();
256     frameCount++;
257
258     // 100ms ごとにFPS を計算
259     if (frameCount >= 30) {
260         double timeDiff = (currentTime - lastTime) / 1_000_000.0;
261         double fps = 1000.0 * 30 / timeDiff;
262         frameCount = 0; // フレーム数をリセット
263         lastTime = currentTime; // 時間を更新
264         //System.out.println("FPS: " + fps); // デバッグ出力
265     }
266
267     SwingUtilities.invokeLater(this::repaint); // スレッドで描画 Swing
268 }, 0, 16, TimeUnit.MILLISECONDS);
269
270 playerSpeed = player.getPlayerSpeed();

```



```

271         createCacheFloorAll();
272     }
273
274     }
275     public void setController(DrawController cont) { this.cont = cont; }床の画像を
        キャッシュする関数、
276     //のコンストラクタで一回だけ呼ぶ DrawView
277     private void createCacheFloorAll() {
278         int cS = cellSize;
279         int overCell = 6;
280         cacheFloorAll = new BufferedImage(cS*size[0], cS * (size[1]+overCell),
            BufferedImage.TYPE_INT_RGB);
281         Graphics2D g2 = cacheFloorAll.createGraphics();
282
283         // 必要に応じて他の背景パーツを描画する
284         int rB = rightBlank;
285         int hB = headerBlank;
286         for(int i = 1; i < size[0] -1; i++){
287             for(int j = 1; j < size[1] -1; j++){
288                 g2.setColor(Color.DARK_GRAY);
289                 if((i + j)%2 == 0){g2.drawImage(imgF1, i * cS, j * cS, cS, cS, this)
                };
                else {g2.drawImage(imgF2, i * cS, j * cS, cS, cS, this);}
290             }
291         }
292         for(int j = size[1]; j < size[1] + overCell; j++){
293             for(int i = 0; i < size[0]; i++){
294                 g2.setColor(new Color(200,0,0));
295                 g2.drawImage(imgF3, i * cS, j * cS, cS, cS, this);
296             }
297         }
298         g2.dispose();
299     }
300
301     protected void paintComponent(Graphics g) {
302         super.paintComponent(g);
303         passedFlame++;
304         final int dD3d = 20; 疑似//3の実装のために床を実際よりが正向きにずれる。Dy
305         g.drawImage(testWall,rightBlank,0,cellSize*16, headerBlank,this); 奥の壁
        // テスト用
306         //g.drawImage(testWall,0,0,cellSize*18, headerBlank,this); 奥の壁//
307         g.setColor(new Color(101,68,59));
308         g.drawImage(cacheFloorAll, 0+rightBlank, 0+headerBlank + dD3d, this); 床の
        画像だけキャッシュ一時保存//()して処理を軽く
309         g.fillRect(0, 0, rightBlank, 1200);
310         g.fillRect(0 + rightBlank + size[0]*cellSize, 0, rightBlank, 1200);
311         //g.drawImage(sideWall, 20, 0, 20, 1000, this);
312         //g.drawImage(sideWall, 16*60 + rightBlank, 0, 20, 1000, this);
313         final int rB = rightBlank;
314         final int hB = headerBlank;
315         final int cS = cellSize;プレイヤーの座標のアニメーション処理
316
317         //
318         if(Math.abs(player.x - player.xAnim) <= playerSpeed){ //について x
319             player.xAnim = player.x;
320             player.moving = false;
321         }else if(player.x > player.xAnim){
322             player.xAnim += playerSpeed;
323             player.moving = true;
324         }else if(player.x < player.xAnim){
325             player.xAnim -= playerSpeed;
326             player.moving = true;
327         }
328
329         if(Math.abs(player.y - player.yAnim) <= playerSpeed){ //について y
330             player.yAnim = player.y;
331             player.moving = (player.moving || false);
332         }else if(player.y > player.yAnim){
333             player.yAnim += playerSpeed;
334             player.moving = true;
335         }else if(player.y < player.yAnim){
336             player.yAnim -= playerSpeed;
337             player.moving = true;
338         }プレイヤーの下影の描画
        //
339         g.setColor(Color.BLACK);
340         g.setColor(new Color(0,0,0,128));
341         g.fillRect((int)(player.xAnim*cellSize) + rB + 10, (int)(player.yAnim*
            cellSize) + hB +dD3d + 10, 40, 40);テーブルの描
342         画
343

```

```

344 //
345 for (int j = 0; j < size[1]; j++) {
346     for (int i = 0; i < size[0]; i++) {
347         if (grid[i][j].wall) {
348             if ((i == 0 || i == size[0] - 1) && j != size[1] - 1 && j != 0) {
// 右と左のテ
// ー
349                 g.drawImage(imgA, i * cellSize + rB, j * cellSize + hB,
cellSize, cellSize, this);
350             } else {
351                 g.drawImage(imgB, i * cellSize + rB, j * cellSize + hB,
cellSize, cellSize + dD3d + 14, this);
352             }
353         } else if (grid[i][j].obstacle) {
354             g.drawImage(imgB, i * cellSize + rB, j * cellSize + hB, cellSize
, cellSize + dD3d + 14, this);
355         }
356     }
357     if (grid[i][j].isPlatePlaced == true) { 皿は食材の土台にあるべきなので、皿
のみの特殊描画処理//
358         if (grid[i][j].wall == false && grid[i][j].obstacle == false) {
359             g.drawImage(imgPlate, i * cellSize + rB, j * cellSize + hB +
dD3d, cellSize, cellSize, this);
360         } else 土台の上なら疑似{ //3座標ズレを考慮 D
361             g.drawImage(imgPlate, i * cellSize + rB, j * cellSize + hB,
cellSize, cellSize, this);
362         }
363     } 食材画像を描画
364 //
365     Image selectedImage = null;
366     if (grid[i][j].plate == null && grid[i][j].food != null) { そのマスは
//をもっていない plate かつそのマスにはしょくざいがあるときつまり皿の描画
はなく
368         //だけの描画の場合です。Food
369         selectedImage = setFoodImage(grid[i][j].food);
370     } else if (grid[i][j].plate != null && grid[i][j].plate.hasAnyFood()
== true) { 皿があって食材がおいてある場合
//
371         selectedImage = setPlateImage(grid[i][j].plate);
372     }
373     if (selectedImage != null) {
374         int length = (int)(cellSize*0.7); 描画画像の一辺の長さ//
375         int cenOffSet = (cellSize - length)/2; 画像のサイズが変わったときに、
描画位置の調整をするもの//
376         if (grid[i][j].wall == false && grid[i][j].obstacle == false) { 台
上じゃなかったら//
377             g.drawImage(selectedImage, i * cS + rB + cenOffSet, j * cS +
hB + dD3d + cenOffSet, length, length, this);
378         } else { 台上だったら//
379             g.drawImage(selectedImage, i * cS + rB + cenOffSet, j * cS +
hB + cenOffSet, length, length, this);
380         }
381     }
382 }
383 } 影を落とす
384 //
385 g.drawImage(longShadow, 0+rightBlank, 0+headerBlank, 960, 14, this);
386
387 g.drawImage(imgCounter[(passedFlame/15)%5], 7*cellSize + rB, 8*cellSize +
hB, cellSize*2, cellSize + dD3d, this); カウンターを座標指定して描画//すべ
ての座標について重
388
389 //2文 for
390 for (int i = size[0]-1; i >= 0; i--) {
391     for (int j = size[1]-1; j >= 0; j--) {
392         Image selectedImage = null; ツールマスに関する描画
393         //
394         if (grid[i][j].tool != 0) {
395             selectedImage = setToolImage(grid[i][j].tool);
396             if (grid[i][j].foodBox != 0)
397                 g.drawImage(imgB, i * cellSize + rB, j * cellSize + hB, cellSize
, cellSize, this);
398         }
399         if (selectedImage != null) {
400             if (grid[i][j].wall == false && grid[i][j].obstacle == false) { 台
上じゃなかったら//
401                 g.drawImage(selectedImage, i * cS + rB, j * cS + hB + dD3d,
cellSize, cellSize, this);
402             } else { 台上だったら//

```

```

403         g.drawImage(selectedImage, i * cS + rB, j * cS + hB,
404                       gridSize, gridSize, this);
405     }
406 }
407
408
409 for (int i = size[0]-1; i >= 0; i--){
410     for (int j = size[1]-1; j >= 0; j--){
411         if(grid[i][j].isPlatePlaced && grid[i][j].plate.hasAnyFood()){
412             setIngredientsImage(cellSize, grid[i][j].x*cS, grid[i][j].y*cS,
413                                0, 0, grid[i][j].plate, g, 0);
414         }
415     }
416 }
417
418 // 向きによってプレイヤーの向きを決定して、プレイヤーを描画
419 switch(player.direction){
420     case 1: ImagePlayer = imgPlayerUp; break;
421     case 2: ImagePlayer = imgPlayerLeft; break;
422     case 3: ImagePlayer = imgPlayerDown; break;
423     case 4: ImagePlayer = imgPlayerRight; break;
424 }
425 g.drawImage(ImagePlayer, (int)(player.xAnim*cellSize)-10 + rB, (int)(
426     player.yAnim*cellSize) + hB -10, 80, 80, this);
427
428 if(player.hasPlate == true){ プレイヤーが皿を持っていたら//皿と画像の比率を調整
429     //
430     int foodSize = (int)(0.68*cellSize);
431     int offsetX = (cellSize - foodSize)/2;
432     int offsetY = (cellSize - foodSize)/2;
433     if(player.direction == 1) offsetY -= (int)(.92*cellSize);
434     else if(player.direction == 2) offsetX -= (int)(0.8*cellSize);
435     else if(player.direction == 3) offsetY += (int)(0.72*cellSize);
436     else if(player.direction == 4) offsetX += (int)(0.8*cellSize);
437     g.drawImage(imgPlate, (int)(player.xAnim*cS) + offsetX + rB + 1, (int)(
438         player.yAnim*cS) + offsetY + 4 + hB, foodSize, foodSize, this); //
439         は微調整項
440         +1,+4
441 }
442 Image heldFoodImage = null;
443 if(player.hasPlate == true && player.plate.hasAnyFood() == true){ 食材あり
444     の皿を持っていたら//
445     heldFoodImage = setPlateImage(player.plate);
446 }else if(player.getFood() != null){ 単体の食材を持っていたら//
447     heldFoodImage = setFoodImage(player.getFood());
448 }
449 if (heldFoodImage != null) {
450     // 少し小さめにプレイヤーの上に描画
451     int foodSize = (int)(0.55*cellSize);
452     int offsetX = (cellSize - foodSize)/2;
453     int offsetY = (cellSize - foodSize)/2;
454     if(player.direction == 1) offsetY -= (int)(.92*cellSize); 上のブロックのパ
455     ラメータと共通//
456     else if(player.direction == 2) offsetX -= (int)(0.8*cellSize);
457     else if(player.direction == 3) offsetY += (int)(0.72*cellSize);
458     else if(player.direction == 4) offsetX += (int)(0.8*cellSize);
459     g.drawImage(heldFoodImage, (int)(player.xAnim*cS) + offsetX + rB + 2, (
460         int)(player.yAnim*cS) + offsetY + hB, foodSize, foodSize, this);
461     は微調整項
462     //+1
463 }
464 if(player.hasPlate == true && player.plate.hasAnyFood()){
465     int offsetX = cellSize / 4;
466     int offsetY = cellSize / 4;
467     if(player.direction == 1) {offsetX = 0; offsetY -= cellSize *2/ 3;}
468     else if(player.direction == 2) {offsetX -= cellSize *2/ 3; offsetY =
469         0;}
470     else if(player.direction == 3) {offsetX = 0; offsetY += cellSize ;}
471     else if(player.direction == 4) {offsetX += cellSize / 3; offsetY = 0;}
472     setIngredientsImage(cellSize, (int)(player.xAnim*cS), (int)(player.
473         yAnim*cS), offsetX, offsetY, player.plate, g, player.direction);
474     //setIngredientsImage(cellSize, player.x, player.y, offsetX, offsetY,
475         player.plate, g, player.direction);
476 }装飾品の描画
477
478 //
479 //g.drawImage(imgCandle, 0*cellSize + rightBlank, 0 * cellSize +
480     headerBlank - 60, 60, 120, this);
481 //g.drawImage(imgCandle, 15*cellSize + rightBlank, 0 * cellSize +
482     headerBlank - 60, 60, 120, this);

```

```

468 //g.drawImage(imgCandle, 1*cellSize + rightBlank, 8 * cellSize +
    headerBlank - 60, 60, 120, this);
469 //g.drawImage(imgCandle, 14*cellSize + rightBlank, 8 * cellSize +
    headerBlank - 60, 60, 120, this);
470 g.drawImage(imgCandle, 6*cellSize + rightBlank, 8 * cellSize +
    headerBlank - 60, 60, 120, this);
471 g.drawImage(imgCandle, 9*cellSize + rightBlank, 8 * cellSize +
    headerBlank - 60, 60, 120, this);
472
473 //の描画 UI
474 g.drawImage(imgUIBG, 55, 750, 250, 90, this); 得点表示の背景//
475 g.drawImage(imgCoin, 0, 730, 120, 120, this); 得点表示の背景//
476
477 g.drawImage(imgUIBG, 655, 750, 250, 90, this); 時間表示の背景//
478 g.drawImage(imgTimer, 868, 730, 120, 120, this); 時間表示の背景//
479 Graphics2D g2d = (Graphics2D) g;
480 g2d.setFont(customFont);
481 g2d.setColor(Color.WHITE);
482 int leftTimeAllSec = model.getTime();
483 int leftTimeMin = leftTimeAllSec/60;
484 int leftTimeSec = leftTimeAllSec%60;
485 g2d.drawString(String.format("%d:%02d", leftTimeMin, leftTimeSec), 712,
    820);
486
487 double dScore = model.score - scoreAnim;
488 if(dScore != 0.0 && flameScoreGet == 0){ getScore = (int)dScore;
    flameScoreGet = 1; } 増加スコアエフェクトのトリガー
    //
489 scoreAnim += dScore * easingFactorText;
490 if (Math.abs(dScore) < 2.0) { scoreAnim = model.score; }
491
492 String text = Integer.toString((int)scoreAnim);
493 FontMetrics fm = g2d.getFontMetrics();
494 int textWidth = fm.stringWidth(text);
495 int centerX = 202; // 中央に配置したい座標 x
496 g2d.drawString(text, centerX - textWidth / 2, 820);
497
498 if(1 <= flameScoreGet && flameScoreGet <= 60){
499     text = Integer.toString(getScore);
500     if(getScore >= 0){
501         g.setColor(new Color(50, 255, 50, 200 - 2*flameScoreGet));
502         text = "+" + text;
503     } else {
504         g.setColor(new Color(255, 50, 50, 200 - 2*flameScoreGet));
505     }
506     fm = g2d.getFontMetrics();
507     textWidth = fm.stringWidth(text);
508     centerX = 175; // 中央に配置したい座標 x
509     g2d.drawString(text, centerX - textWidth / 2, 770 - 2*flameScoreGet/3);
510     flameScoreGet++;
511 }else if(flameScoreGet > 60){ flameScoreGet = 0; }オーダー用紙の描画
512
513 //
514 for(int i = 0; i < model.orders.length; i++){
515     Image orderImage;
516     int orderW = 160;
517     int orderH = 100;
518     if(model.orders[i] != null){
519         Order order = model.orders[i];
520         orderImage = setOrderImage(order);
521         int targetPos = 20 + i * (orderW + 5);
522         double dx = targetPos - order.posAnim;
523         order.posAnim += dx * easingFactor;
524
525         if (Math.abs(dx) < 1.0) {
526             order.posAnim = targetPos;
527             if(order.timeAnim == 0){
528                 order.timeAnim = 1;
529             }
530         }
531     }
532     if(1 <= order.timeAnim) {
533         if(30 <= order.timeAnim){
534             dx = order.subOrderPosY - order.subOrderPosYAnim;
535             order.subOrderPosYAnim += easingFactor * dx;
536             if(Math.abs(dx) < 1.0){
537                 order.subOrderPosYAnim = order.subOrderPosY;
538             }
539             int sOPYA = (int)order.subOrderPosYAnim; 文字が長いんで//型に
                キャストして入れ直し int
540             int interval = cellSize-11;

```

```

541         int wid = 45;
542         if(order.ingredient1 != null){
543             g.setColor(new Color(174, 207, 227));
544             g.fillRect((int)order.posAnim+7+interval*0, sOPYA, wid,
545                 90);
546             g.drawImage(setCorrectRaw(order.ingredient1), (int)order
547                 .posAnim+interval*0 + 8, sOPYA+10, 42,42,this);
548             if(setCorrectMethod(order.ingredient1)!=null){
549                 g.drawImage(setCorrectMethod(order.ingredient1), (int)
550                     order.posAnim+interval*0 + 9, sOPYA+50, 42,42,
551                     this);
552             }
553         }
554         if(order.ingredient2 != null){
555             g.setColor(new Color(174, 207, 227));
556             g.fillRect((int)order.posAnim+7+interval*1, sOPYA, wid,
557                 90);
558             g.drawImage(setCorrectRaw(order.ingredient2), (int)order
559                 .posAnim+interval*1 + 8, sOPYA+10, 42,42,this);
560             if(setCorrectMethod(order.ingredient2)!=null){
561                 g.drawImage(setCorrectMethod(order.ingredient2), (int)
562                     order.posAnim+interval*1 + 9, sOPYA+50, 42,42,
563                     this);
564             }
565         }
566         if(order.ingredient3 != null){
567             g.setColor(new Color(174, 207, 227));
568             g.fillRect((int)order.posAnim+7+interval*2, sOPYA, wid,
569                 90);
570             g.drawImage(setCorrectRaw(order.ingredient3), (int)order
571                 .posAnim+interval*2 + 8, sOPYA+10, 42,42,this);
572             if(setCorrectMethod(order.ingredient3)!=null){
573                 g.drawImage(setCorrectMethod(order.ingredient3), (int)
574                     order.posAnim+interval*2 + 9, sOPYA+50, 42,42,
575                     this);
576             }
577         }
578         order.timeAnim++;
579     }
580     //g.fillRect((int)order.posAnim, 0 * cellSize +20, 3*(cellSize-2),
581     60);
582     g.drawImage(orderPaper, (int)order.posAnim, 15, orderW, orderH,
583         this);
584     drawGauge(g, "down", (int)(order.posAnim)+8, 22, orderW-16, 17,
585         order.getRemainingTime()/order.timeLimit);
586     //g.drawImage(orderImage, 53 + (int)order.posAnim, 70, cellSize+5,
587     cellSize+5, this);
588     //g.drawImage(imgSampleSalad, 42 + (int)order.posAnim, 30, 75, 75,
589     this)プレビューのためです
590     ;// Kome
591     g.drawImage(orderImage, 42 + (int)order.posAnim, 30, 75, 75, this)プ
592     レビューのためです;// Kome
593
594     }
595 }
596
597 if(cont.spacePushing == true){
598     if(player.getFrontGrid().tool == 12){player.actionCharge += 0.5;} フライ
599     パンの時は長め//
600     else player.actionCharge += 1;
601 }
602 else{ player.actionCharge = 0; }
603 if(0 < player.actionCharge && player.actionCharge < 60){
604     drawGauge(g, "up", (int)(player.xAnim*cellSize)+rightBlank + 10, (int)
605         (player.yAnim*cellSize)+headerBlank,(int)(0.7*cellSize),8,player.
606         actionCharge/60.0);
607 }else if(player.actionCharge == 60) player.action();
608
609 // しょぼいんですけど、フライパンの火の描画ですYoshida
610 if(player.food != null && player.food.canHeat){
611     if(player.getFrontGrid().tool == 12 && cont.spacePushing == true){
612         if(player.actionCharge>0 && player.actionCharge<60){
613             float fireScall = player.actionCharge % 30;行目は大きめ、行目は小
614             さめ
615             //12
616             //g.drawImage(imgFire, player.getFrontGrid().x * cellSize +30-(
617                 int)(fireScall), player.getFrontGrid().y * cellSize +
618                 headerBlank+55-(int)(fireScall), (int)(fireScall*cellSize

```

```

597         /30), (int)(fireScall*cellSize/30), this);
598         g.drawImage(imgFire, player.getFrontGrid().x * cellSize +30-(
599             int)(fireScall/2), player.getFrontGrid().y * cellSize +
600             headerBlank+55-(int)(fireScall/2), (int)(fireScall*cellSize
601             /60), (int)(fireScall*cellSize/60), this);
602     }
603     }米炊く
604     //Yoshida
605     for (int i = 0; i < size[0]; i++) {
606         for (int j = 0; j < size[1]; j++) {
607             if(grid[i][j].tool == 10 && grid[i][j].hasFood()){
608                 if(grid[i][j].cookingGauge < 60.0)grid[i][j].cookingGauge +=
609                     0.1;
610                 if(grid[i][j].cookingGauge > 0 && grid[i][j].cookingGauge < 60){
611                     drawGauge(g, "up", i*cS+7 + rightBlank, j*cS+headerBlank-10,
612                         (int)(0.7*cS), 8, grid[i][j].cookingGauge/60.0);
613                 }
614                 else if(grid[i][j].cookingGauge >= 60.0){
615                     if(grid[i][j].food.foodName == "rice"){
616                         g.drawImage(setToolImage(11), i * cS +rightBlank, j * cS
617                             + headerBlank, cS, cS, this);
618                     }
619                 }
620             }
621         }
622     }
623     for(int i = 0; i < 5; i++){
624         if(waiters[i] != null && waiters[i].active == true){
625             //System.out.printf("waiters[%d]のdrawMe()を呼びます\n", i);
626             waiters[i].drawMe(g, this);
627         }
628     }
629     if(passedFlame == 60) AudioManager.playBGM("./sound/music_background2.wav
630     ");
631 }
632 private void drawFloorAll(Graphics g, ImageObserver io){
633     int cS = cellSize; この中で略語を定義//
634     int rB = rightBlank;
635     int hB = headerBlank;
636     for(int i = 0; i < size[0]; i++){
637         for(int j = 0; j < size[1]; j++){
638             g.setColor(Color.DARK_GRAY);
639             if((i + j)%2 == 0){g.drawImage(imgF1, i * cS + rB, j * cS + hB, cS,
640                 cS, this);}
641             else {g.drawImage(imgF2, i * cS + rB, j * cS + hB, cS, cS, this);}
642         }
643     }
644     private void drawGauge(Graphics g, String type, int x, int y, int width, int
645         height, double ratio){
646         if(ratio > 1) { System.out.println("Warning : ゲージの割合がを超えています
647             100%"); }
648         //System.out.printf("ratio = %.1f\n", ratio); デバッグ用//
649         if(type == "up"){
650             g.setColor(Color.WHITE);
651             g.fillRect(x-2, y-2, width+4, height+4);
652             g.setColor(new Color(75, 180, 35));
653             g.fillRect(x, y, (int)(width*ratio), height);
654         }
655         else if(type == "down"){
656             g.setColor(Color.GRAY);
657             g.fillRect(x, y, width, height);
658             if(ratio >= 0.5) { g.setColor(new Color(75, 180, 35)); }
659             else if(ratio >= 0.25) { g.setColor(Color.YELLOW); }
660             else{ g.setColor(Color.RED); }
661             g.fillRect(x, y, (int)(width*ratio), height);
662         }
663     }
664     private Image setToolImage(int toolId){
665         switch(toolId){
666             case 1: return imgKnife;

```

```

667         case 2: return imgCabbageBox;
668         case 3: return imgPlateBox;
669         case 4: return imgTomatoBox;
670         case 5: return imgCucumberBox;
671         case 6: return imgRiceBox;
672         case 7: return imgTunaBox;
673         case 8: return imgSquidBox;
674         case 9: return imgSeaweedBox;
675         case 10: return imgBoil;
676         case 11: return imgBoilRice;
677         case 12: return imgPan;
678         case 13: return imgTrash;
679         case 14: return null;
680     }
681     return imgErrorBlock;
682 }
683 private Image setCorrectRaw(Food foodInfo){
684     if(foodInfo.foodName == "cabbage") return imgCabbage;
685     else if(foodInfo.foodName == "tomato") return imgTomato;
686     else if(foodInfo.foodName == "cucumber") return imgCucumber;
687     else if(foodInfo.foodName == "rice") return imgRice;
688     else if(foodInfo.foodName == "tuna") return imgTuna;
689     else if(foodInfo.foodName == "squid") return imgSquid;
690     else if(foodInfo.foodName == "seaweed") return imgSeaweed;
691
692     else return imgErrorBlock;
693 }
694 private Image setCorrectMethod(Food foodInfo){
695     if(foodInfo.foodStatus == 2) return imgKnifeBlack;
696     else if(foodInfo.foodStatus == 3) return imgBoilBlack;
697     else return null;
698 }
699 private Image setFoodImage(Food foodInfo){
700     // 文にしてもいいかもね switch
701     if(foodInfo.foodName == "cabbage"){
702         if(foodInfo.foodStatus == 1) return imgCabbage;
703         else if(foodInfo.foodStatus == 2) return imgCabbageCut;
704         else return imgErrorBlock;
705     }else if(foodInfo.foodName == "tomato"){
706         if(foodInfo.foodStatus == 1) return imgTomato;
707         else if(foodInfo.foodStatus == 2) return imgTomatoCut;
708         else return imgErrorBlock;
709     }else if(foodInfo.foodName == "cucumber"){
710         if(foodInfo.foodStatus == 1) return imgCucumber;
711         else if(foodInfo.foodStatus == 2) return imgCucumberCut;
712         else return imgErrorBlock;
713     }else if(foodInfo.foodName == "rice"){
714         if(foodInfo.foodStatus == 1) return imgRice;
715         else if(foodInfo.foodStatus == 3) return imgRiceBoil;//は boil3?heiwa
716         else return imgErrorBlock;
717     }else if(foodInfo.foodName == "tuna"){
718         if(foodInfo.foodStatus == 1) return imgTuna;
719         else if(foodInfo.foodStatus == 2) return imgTunaCut;
720         else return imgErrorBlock;
721     }else if(foodInfo.foodName == "squid"){
722         if(foodInfo.foodStatus == 1) return imgSquid;
723         else if(foodInfo.foodStatus == 2) return imgSquidCut;
724         else return imgErrorBlock;
725     }else if(foodInfo.foodName == "cucumber"){
726         if(foodInfo.foodStatus == 1) return imgCucumber;
727         else if(foodInfo.foodStatus == 2) return imgCucumberCut;
728         else return imgErrorBlock;
729     }else if(foodInfo.foodName == "seaweed"){
730         if(foodInfo.foodStatus == 1) return imgSeaweed;
731         else return imgErrorBlock;
732     }
733     return imgErrorBlock;
734 }
735 public Image setPlateImage(Plate targetPlate){
736     Food food[] = new Food[3];
737     int cabbage = 0; そのプレートにおいてそれぞれの食材がどうなっているか//
738     int tomato = 0; 存在しない//0: 生 1: カット、ボイル 2:3:
739     int cucumber = 0;
740     int rice = 0;
741     int tuna = 0;
742     int squid = 0;
743     int seaweed = 0;
744
745
746     //に乗っている具材情報を取得 plate
747     for(int i = 0; i < 3; i++){

```

```

748         food[i] = targetPlate.get(i);
749         if(food[i] == null){ break; これ以上の食材はないので }//break
750         if(food[i].foodName == "cabbage") cabbage = food[i].foodStatus;
751         else if(food[i].foodName == "tomato") tomato = food[i].foodStatus;
752         else if(food[i].foodName == "cucumber") cucumber = food[i].foodStatus;
753         else if(food[i].foodName == "rice") rice = food[i].foodStatus;
754         else if(food[i].foodName == "tuna") tuna = food[i].foodStatus;
755         else if(food[i].foodName == "squid") squid = food[i].foodStatus;
756         else if(food[i].foodName == "seaweed") seaweed = food[i].foodStatus;
757
758     }取得した具材情報を利用して
759     //にセットする画像を返す。Image0未所持未処理カットボイル:,1:,2:,3:,
760
761     if(rice==0 && tuna==0 && squid==0 && seaweed==0){
762         //System.out.printf("rice = %d", rice)デバック用;//
763         if(cabbage==1 && tomato==0 && cucumber == 0) return imgCabbage; 未加工
764             キャベツ//
765         else if(cabbage==0 && tomato==1 && cucumber == 0) return imgTomato; 未
766             加工トマト//
767         else if(cabbage==0 && tomato==0 && cucumber == 1) return imgCucumber;
768             未加工きゅうり
769         //
770         else if(cabbage==2 && tomato==0 && cucumber == 0) return imgCabbageCut;
771             カットキャベツ
772         //
773         else if(cabbage==0 && tomato==2 && cucumber == 0) return imgTomatoCut;
774             カットトマト
775         //
776         else if(cabbage==0 && tomato==0 && cucumber == 2) return
777             imgCucumberCut; カットキュウリ
778         //
779         else if(cabbage == 2 && tomato == 2 && cucumber == 0) return imgCabTom
780             キャベツトマト
781         ;//
782         else if(cabbage == 2 && tomato == 0 && cucumber == 2) return imgCabCuc
783             キャベツキュウリ
784         ;//
785         else if(cabbage == 0 && tomato == 2 && cucumber == 2) return imgTomCuc
786             トマトキュウリ
787         ;//
788         else if(cabbage == 2 && tomato == 2 && cucumber == 2) return
789             imgCabTomCucキャベツトマトキュウリ
790         ;//
791     }
792     else if(cabbage==0 && tomato==0 && cucumber==0 && squid==0){
793         //System.out.printまぐろ ("")デバック用;//
794         if(rice == 1 && tuna == 0 && seaweed== 0) return imgRice加工前;//
795         else if(rice == 0 && tuna == 1 && seaweed== 0) return imgTuna;//
796         else if(rice == 0 && tuna == 0 && seaweed== 1) return imgSeaweed;//
797         else if(rice == 3 && tuna == 0 && seaweed== 0) return imgRiceBoil加工後
798             ;//
799         else if(rice == 0 && tuna == 2 && seaweed== 0) return imgTunaCut;//
800         else if(rice == 3 && tuna == 2 && seaweed== 0) return imgRiceTunまぐろにぎ
801             り;//
802         else if(rice == 3 && tuna == 0 && seaweed== 1) return imgRiceSea;//
803         else if(rice == 0 && tuna == 2 && seaweed== 1) return imgTunSea;//
804         else if(rice == 3 && tuna == 2 && seaweed== 1) return imgRiceTunSea鉄火巻
805             ;//
806     }
807     else if(cabbage==0 && tomato==0 && cucumber==0 && tuna==0 && seaweed==0){
808         //System.out.printいか ("")デバック用;//
809         if(rice == 1 && squid == 0) return imgRice加工前;//
810         else if(rice == 0 && squid == 1) return imgSquid;//
811         else if(rice == 3 && squid == 0) return imgRiceBoil加工後;//
812         else if(rice == 0 && squid == 2) return imgSquidCut;//
813         else if(rice == 3 && squid == 2) return imgRiceSquいかにぎり;//
814     }
815     else if(cabbage==0 && tomato==0 && cucumber==0 && seaweed==0){
816         //System.out.print海鮮丼 ("")デバック用;//
817         if(rice == 1 && tuna == 0 && squid== 0) return imgRice加工前;//
818         else if(rice == 0 && tuna == 1 && squid== 0) return imgTuna;//
819         else if(rice == 0 && tuna == 0 && squid== 1) return imgSquid;//
820         else if(rice == 3 && tuna == 0 && squid== 0) return imgRiceBoil加工後;//
821         else if(rice == 0 && tuna == 2 && squid== 0) return imgTunaCut;//
822         else if(rice == 0 && tuna == 0 && squid== 2) return imgSquidCut;//
823         else if(rice == 3 && tuna == 2 && squid== 0) return imgRiceTunまぐろにぎり
824             ;//
825         else if(rice == 3 && tuna == 0 && squid== 2) return imgRiceSquいかにぎり
826             ;//
827         else if(rice == 0 && tuna == 2 && squid== 2) return imgTunSqu;//
828         else if(rice == 3 && tuna == 2 && squid== 2) return imgRiceTunSqu海鮮丼

```



```

806         };
807     else if(cabbage==0 && tomato==0 && tuna==0 && squid==0){
808         //System.out.printかつば巻き ("")デバッグ用;
809         if(rice == 1 && cucumber == 0 && seaweed== 0) return imgRice加工前;
810         else if(rice == 0 && cucumber == 1 && seaweed== 0) return imgCucumber
811         ;
812         else if(rice == 0 && cucumber == 0 && seaweed== 1) return imgSeaweed;
813         else if(rice == 3 && cucumber == 0 && seaweed== 0) return imgRiceBoil加工後;
814         else if(rice == 0 && cucumber == 2 && seaweed== 0) return
815         imgCucumberCut;
816         else if(rice == 3 && cucumber == 2 && seaweed== 0) return imgRicCuc;
817         else if(rice == 3 && cucumber == 0 && seaweed== 1) return imgRicSea;
818         else if(rice == 0 && cucumber == 2 && seaweed== 1) return imgCucSea;
819         else if(rice == 3 && cucumber == 2 && seaweed== 1) return imgRicCucSea
820         かつば巻
821         ;
822     }
823     return imgErrorBlock;
824 }
825
826 public Image setOrderImage(Order order){
827     //System.out.println(order.orderName の画像を取得します。+""); デバッグ用
828     if("salad".equals(order.orderName)){
829         //System.out.println(order.orderName の画像を取得しました。+""); デバッグ用
830         //
831         return imgCabTomCuc;
832     }else if("tekkamaki".equals(order.orderName)){
833         //System.out.println(order.orderName の画像を取得しました。+""); デバッグ用
834         //
835         return imgRicTunSea;
836     }else if("kappamaki".equals(order.orderName)){
837         //System.out.println(order.orderName の画像を取得しました。+""); デバッグ用
838         //
839         return imgRicCucSea;
840     }else if("tunanigiri".equals(order.orderName)){
841         //System.out.println(order.orderName の画像を取得しました。+""); デバッグ用
842         //
843         return imgRicTun;
844     }else if("ikanigiri".equals(order.orderName)){
845         //System.out.println(order.orderName の画像を取得しました。+""); デバッグ用
846         //
847         return imgRicSqu;
848     }else if("kaisendon".equals(order.orderName)){
849         //System.out.println(order.orderName の画像を取得しました。+""); デバッグ用
850         //
851         return imgRicTunSqu;
852     }
853     else return null;
854 }
855
856 // を返すわけではなく、この関数を呼び出せば画像を貼れる Image Yoshida
857 // に書いても良かったけど煩雑になりそうだったので関数化しました。引数が多くてすいません。
858 paintComponent
859 private void setIngredientsImage(int cellSize, int xAnim, int yAnim, int
860 offsetX, int offsetY, Plate plate, Graphics g, int playerDirection){
861     Image ingredients[] = new Image[3];
862     int holdStatus[] = new int[3];
863     Food ing[] = new Food[3];
864     int size = cellSize/3;
865     int ingOffsetX = 20;
866     int ingOffsetY = 20;
867     final int hB = headerBlank;
868     final int rB = rightBlank;
869     if(playerDirection == 3){ingOffsetY = 0;}
870     for(int i=0; i<3; i++){
871         if(plate.foods[i] != null){
872             ing[i] = plate.foods[i];
873             holdStatus[i] = plate.foods[i].foodStatus;
874             ing[i].foodStatus = 1; 生の状態を表示したい調理した食材を皿に置いて、歩ある
875             と画像が生になってしまうのでコメントアウトしてます。 //(1)
876         }
877     }
878     for(int i=0; i<3; i++){
879         if(ing[i] != null){
880             ingredients[i] = setFoodImage(ing[i]);
881             g.setColor(Color.WHITE);
882             g.fillOval(xAnim+ingOffsetX*i+offsetX-3 +rB, yAnim+hB+offsetY-
883             ingOffsetY-2, size+5, size+5);

```

```

872         g.drawImage(ingredients[i], xAnim+ingOffsetX*i+offsetX +rB, yAnim+
873             hB+offsetY-ingOffsetY, size, size, this);
874         ing[i].foodStatus = holdStatus[i];
875     }
876 }時間に関するメソッド
877
878 // Yoshida
879 public void updateTime(int time){
880     //System.out.print(time秒+""); 仮のタイマー表示//
881 }
882
883 // JFrame を取得するメソッド (でリザルト画面に移るときにゲームのウィンドウを閉じる時に使
    ます Controller) Yoshida
884 public JFrame getFrame() {
885     return (JFrame) SwingUtilities.getWindowAncestor(this);
886 }
887 private void loadCustomFont() {
888     try {
889         //File fontFile = new File("font/CHEESE10.TTF"); // フォントファイルのパス
890         File fontFile = new File("font/ByteBounce.ttf"); // フォントファイルのパス
891         customFont = Font.createFont(Font.TRUETYPE_FONT, fontFile).deriveFont
            (90f); // フォントサイズ
892     } catch (IOException | FontFormatException e) {
893         System.err.println("フォントの読み込みに失敗 (" + e.getMessage());
894         customFont = new Font("Arial", Font.BOLD, 24); // 失敗時はデフォルトのフォ
            ント
895     }
896 }
897 public void addWaiter(Image mealImage){
898     for(int i = 0; i < 5; i++){
899         if(waiters[i] == null || waiters[i].active == false){
900             System.out.println("Waiter Instance made.");
901             waiters[i] = new Waiter(model, mealImage, imgWaiterDown,
                imgWaiterUp, headerBlank, rightBlank, player.x);
902             return;
903         }
904     }
905 }
906 }

```

- Controller

```

1  import javax.swing.*;
2  import java.awt.*;
3  import java.awt.event.*;
4
5  class DrawController implements KeyListener {
6      protected DrawModel model;
7      protected DrawView view;
8      protected Player player;
9      protected Timer orderTimer;
10     public boolean spacePushing = false;
11     private Timer gameTimer;
12     private MiniCook mainApp;
13     private int cCount = 0;
14
15     public DrawController(DrawModel m, DrawView v, MiniCook app) {
16         model = m;
17         view = v;
18         player = model.getPlayer(); ここで//を取得しておく player
19         mainApp = app;
20     }
21
22     @Override
23     public void keyPressed(KeyEvent e) {
24         int dx = 0, dy = 0;
25
26         switch (e.getKeyCode()) {
27             case KeyEvent.VK_W:
28                 dy = -1;
29                 player.direction = 1; プレイヤーの向きを変更//
30                 model.movePlayer(dx, dy);
31                 break;
32             case KeyEvent.VK_S:
33                 dy = 1;
34                 player.direction = 3;
35                 model.movePlayer(dx, dy);
36                 break;

```

```

37         case KeyEvent.VK_A:
38             dx = -1;
39             player.direction = 2;
40             model.movePlayer(dx, dy);
41             break;
42         case KeyEvent.VK_D:
43             dx = 1;
44             player.direction = 4;
45             model.movePlayer(dx, dy);
46             break;
47         case KeyEvent.VK_C:
48             cCount++;
49             if(cCount >= 5){ cCount = 0; printCredit(); }
50             break;
51         case KeyEvent.VK_SPACE: スペースキーで//action
52             spacePushing =true;
53             //player.action();
54             break;
55         case KeyEvent.VK_J: //キーで拾う J
56             player.pick_up();
57             break;
58         case KeyEvent.VK_K: //キーで置く K
59             player.put();
60             break;
61         case KeyEvent.VK_I: デバッグ用に//キーで情報を表示する I
62             model.printInfo();
63             break;
64         case KeyEvent.VK_ESCAPE: // キーでゲーム終了 ESC
65             System.exit(0);
66             break;
67     }
68
69     // 再描画
70     //view.repaint();
71 }
72 public void stopOrderTimer() {
73     if (orderTimer != null) {
74         for(int i=0; i<model.orders.length; i++){
75             if(model.orders[i] != null){
76                 model.orders[i].cancelTimer();
77             }
78         }
79         orderTimer.stop();
80     }
81 }
82 @Override
83 public void keyReleased(KeyEvent e) {
84     switch (e.getKeyCode()) {
85         case KeyEvent.VK_SPACE: // スペースキーを離したらfalse にする
86             spacePushing = false;
87             break;
88     }
89 }
90
91 @Override
92 public void keyTyped(KeyEvent e) {} 以下ゲーム時間に関わるメソッド
93
94 // Yoshida
95 public void startGame(){スタート画面、ゲーム画面、リザルト画面を同一ウィンドウで表示する
96     // Yoshida
97     model.generateOrder();
98     view.repaint();こんな文法あるんだね。知らなかった
99
100     // Kome
101     orderTimer = new Timer(12*1000, new ActionListener() {
102         public void actionPerformed(ActionEvent e){
103             model.generateOrder();
104             view.repaint();
105             System.out.println新しい注文が追加されました!("");
106         }
107     });
108     orderTimer.start();
109     System.out.println("Timer started: " + orderTimer);
110
111     if(gameTimer != null) return; 二重起動防止//
112
113     gameTimer = new Timer(1000, new ActionListener() {
114         public void actionPerformed(ActionEvent e) {
115             if (model.getGameTime() > 0) {

```

```

116         model.decreaseTime();
117         view.updateTime(model.getGameTime());
118         if(model.getGameTime() == 10){
119             AudioManager se = new AudioManager();
120             se.playSE("./sound/music_timer2.wav");
121         }else if(model.getGameTime() == 0){
122             AudioManager.playBGM("./sound/music_resultSE.wav");
123         }
124     }
125     else {
126         gameTimer.stop();
127         gameTimer = null;
128         stopOrderTimer()オーダータイマーも止める;//
129
130         // ゲーム終了時にResult 画面を表示
131         System.out.println("リザルト画面に切り替えます。("); デバッグ用//
132         AudioManager.playBGM("./sound/music_result.wav");
133         mainApp.showResult();
134     }
135 }
136 }
137 });
138
139 gameTimer.start(); // タイマー開始
140 }
141 private void printCredit(){
142     System.out.printf("\r\n" + //
143         "\r\n" + //
144         "-----\r\n" + //
145         "\r\n" + //
146         "--- Credit ---\r\n" + //
147         "\r\n" + //
148         "-----\r\n" + //
149         "\r\n" + //
150         "<Team Members>\r\n" + //
151         "\r\n" + //
152         "Y. Kometani\r\n" + //
153         "\r\n" + //
154         "H. Yoshida\r\n" + //
155         "\r\n" + //
156         "S. Suzuki\r\n" + //
157         "\r\n" + //
158         "\r\n" + //
159         "\r\n" + //
160         "<Special Thanks>\r\n" + //
161         "\r\n" + //
162         "S. Maejima (Character Designer)\r\n" + //
163         "\r\n" + //
164         "K. Isahaya (Background Designer)\r\n" + //
165         "\r\n" + //
166         "K. Kubo (Design Adviser)\r\n" + //
167         "\r\n" + //
168         "and All Players\r\n" + //
169         "\r\n" + //
170         "-----\r\n" + //
171         "\r\n" + //
172         "\r\n" + //
173         "");
174 }
175 }

```

- Order

```

1 import javax.swing.*;
2 import java.awt.*;
3 import java.awt.event.*;
4 import java.awt.image.ImageObserver;
5
6 class Order {
7     String orderName;
8     double posAnim;
9     int subOrderPosY = 110;
10    double subOrderPosYAnim = 40;
11    boolean hasPlate; まず皿が必要//
12    public Food ingredient1;
13    public Food ingredient2;
14    public Food ingredient3; 材料は多くてつまで? //3
15    public int timeLimit; 制限時間//
16    public int orderIndex;
17    private DrawModel model;

```

```
18 public int timeAnim = 0;
19
20 private long createTime; 注文が作成された時間//
21 private Timer expirationTimer; // 自動削除用タイマー
22
23 public Order(String orderName, int orderIndex, DrawModel model){コンストラクタ
    では完成形の値を設定
24 //
25 this.orderName = orderName;
26 this.hasPlate = true;
27 this.createTime = System.currentTimeMillis();
28 this.posAnim = 1200;
29 this.orderIndex = orderIndex;
30 this.model = model;オーダーによって必要な食材や状態切られてる、焼かれてる等
31 //()を設定
32 if("salad".equals(orderName)){
33     System.out.println("Order created: " + this.orderName);
34     this.timeLimit = 100;
35
36     this.ingredient1 = new Cabbage();
37     this.ingredient1.foodStatus = 2;
38     this.ingredient1.isOnPlate = true;
39
40     this.ingredient2 = new Tomato();
41     this.ingredient2.foodStatus = 2;
42     this.ingredient2.isOnPlate = true;
43
44     this.ingredient3 = new Cucumber();
45     this.ingredient3.foodStatus = 2;
46     this.ingredient3.isOnPlate = true;
47 }
48 if("tekkamaki".equals(orderName)){
49     System.out.println("Order created: " + this.orderName);
50     this.timeLimit = 100;
51
52     this.ingredient1 = new Rice();
53     this.ingredient1.foodStatus = 3;
54     this.ingredient1.isOnPlate = true;
55
56     this.ingredient2 = new Tuna();
57     this.ingredient2.foodStatus = 2;
58     this.ingredient2.isOnPlate = true;
59
60     this.ingredient3 = new Seaweed();
61     this.ingredient3.foodStatus = 1;
62     this.ingredient3.isOnPlate = true;
63 }
64 if("kappamaki".equals(orderName)){
65     System.out.println("Order created: " + this.orderName);
66     this.timeLimit = 100;
67
68     this.ingredient1 = new Rice();
69     this.ingredient1.foodStatus = 3;
70     this.ingredient1.isOnPlate = true;
71
72     this.ingredient2 = new Cucumber();
73     this.ingredient2.foodStatus = 2;
74     this.ingredient2.isOnPlate = true;
75
76     this.ingredient3 = new Seaweed();
77     this.ingredient3.foodStatus = 1;
78     this.ingredient3.isOnPlate = true;
79 }
80 if("tunanigiri".equals(orderName)){
81     System.out.println("Order created: " + this.orderName);
82     this.timeLimit = 80;
83
84     this.ingredient1 = new Rice();
85     this.ingredient1.foodStatus = 3;
86     this.ingredient1.isOnPlate = true;
87
88     this.ingredient2 = new Tuna();
89     this.ingredient2.foodStatus = 2;
90     this.ingredient2.isOnPlate = true;
91 }
92 }
93 if("ikanigiri".equals(orderName)){
94     System.out.println("Order created: " + this.orderName);
95     this.timeLimit = 80;
96
97     this.ingredient1 = new Rice();
```

```

98         this.ingredient1.foodStatus = 3;
99         this.ingredient1.isOnPlate = true;
100
101         this.ingredient2 = new Squid();
102         this.ingredient2.foodStatus = 2;
103         this.ingredient2.isOnPlate = true;
104
105     }
106     if("kaisendon".equals(orderName)){
107         System.out.println("Order created: " + this.orderName);
108         this.timeLimit = 100;
109
110         this.ingredient1 = new Rice();
111         this.ingredient1.foodStatus = 3;
112         this.ingredient1.isOnPlate = true;
113
114         this.ingredient2 = new Tuna();
115         this.ingredient2.foodStatus = 2;
116         this.ingredient2.isOnPlate = true;
117
118         this.ingredient3 = new Squid();
119         this.ingredient3.foodStatus = 2;
120         this.ingredient3.isOnPlate = true;
121     }
122
123
124     // 制限時間後に削除するタイマーを設定
125     expirationTimer = new Timer(timeLimit * 1000, new ActionListener() {
126         @Override
127         public void actionPerformed(ActionEvent e) {
128             AudioManager se = new AudioManager();
129             se.playSE("./sound/music_timeuporder3.wav");
130             model.scoreDown(null);
131             removeThisOrder();
132             System.out.println(orderIndex+orderName + " の制限時間が切れました!");
133         }
134     });
135     expirationTimer.setRepeats(false); // 一度だけ実行
136     expirationTimer.start();
137
138     private void removeThisOrder(){
139         model.removeOrder(orderIndex);
140     }
141
142     public boolean isCompleted(Plate plate) { オーダー判定処理// Kome
143         System.out.println("isCompleted() called");
144         boolean[] matchedIngredients = new boolean[3];
145         Food[] orderIngredients = {ingredient1, ingredient2, ingredient3};
146
147         for (int i = 0; i < plate.foods.length; i++) {
148             for (int j = 0; j < orderIngredients.length; j++) {
149                 if(orderIngredients[j] == null){
150                     matchedIngredients[j] = true;
151                     continue;
152                 }
153                 if (!matchedIngredients[j] && plate.foods[i] != null &&
154                     orderIngredients[j] != null) {
155                     if (plate.foods[i].getClass() == orderIngredients[j].getClass()
156                         &&
157                         plate.foods[i].foodStatus == orderIngredients[j].foodStatus)
158                     {
159                         matchedIngredients[j] = true;
160                         break;
161                     }
162                 }
163             }
164         }
165
166         for (boolean matched : matchedIngredients) {
167             if (matched == false){
168                 return false;
169             }
170         }
171         return true;
172     }
173
174     // 残り時間を計算
175     public double getRemainingTime(){
176         long elapsedTimeMill = (System.currentTimeMillis() - createTime);
177         double elapsedTime = elapsedTimeMill / 1000.0;
178         return (timeLimit - elapsedTime);

```

```

176     }
177
178     // 注文の期限切れ確認
179     public boolean isExpired(){
180         return getRemainingTime() <= 0;
181     }
182
183     // タイマーの停止（手動で注文を削除するとき用）
184     public void cancelTimer() {
185         expirationTimer.stop();
186     }
187
188     public String getOrderName() {
189         return orderName;
190     }
191 }

```

- Player

```

1 import javax.swing.*;
2 import java.awt.*;
3 import java.awt.event.*;
4
5 class Player {
6     public int x; プレイヤーの//座標 x
7     public int y; プレイヤーの//座標 y
8     public double xAnim; アニメーション用の座標変数//
9     public double yAnim;
10    public Food food;
11    public Plate plate;
12    public boolean hasPlate;
13    private DrawModel model;
14    private DrawController cont;
15    private DrawView view;
16    private double playerSpeed = 0.2;
17    public int direction; プレイヤーの向き//の順で WASD1上 ()左,2()下,3()右,4()
18    private Grid[][] grid;
19    public boolean moving = false;
20    public float actionCharge = 0;
21
22    public Player(int x, int y, DrawModel model, Grid[][] grid) {
23        this.x = x;
24        this.y = y;
25        this.xAnim = x;
26        this.yAnim = y;
27        this.food = null;
28        this.plate = null;
29        this.model = model;
30        this.direction = 1; 初期の向きは上に設定してあるけど、別になんでも//
31        this.grid = grid;
32        this.hasPlate = false;
33    }
34    public int getX() { return x; }
35    public int getY() { return y; }
36    public Food getFood() { return food; }
37    public double getPlayerSpeed() { return playerSpeed; }
38    public void setController(DrawController cont) { this.cont = cont; }
39    public void setView(DrawView view) { this.view = view; }
40
41    public void move(int dx, int dy, Grid[][] grid) {
42        if(moving == false && getFrontGrid().isPlatePlaced == false &&
43           getFrontGrid().hasFood() == false){ プレイヤー移動中は移動したくない
44           //
45           int newX = x + dx;
46           int newY = y + dy;障害物と重ならないように障害物である場合、移動を棄却する
47           //( )
48           if (newX >= 0 && newX < grid.length && newY >= 0 && newY < grid[0].
49              length) {
50               if (!grid[newX][newY].wall && !grid[newX][newY].obstacle && !grid[
51                  newX][newY].isCounter/*&& (newX != x || newY != y)*/) {
52                   x = newX;
53                   y = newY;
54               }else{
55                   if(grid[newX][newY].wall) System.out.printf("に激突しました
56                      wall\n");
57                   if(grid[newX][newY].obstacle) System.out.printf("に激突しました
58                      obstacle\n");
59               }
60           }
61       }
62   }
63 }

```

```

56     }
57
58     public Grid getFrontGrid(){ 自分が立っている目の前の//オブジェクトを返す関数 Grid
59         if(direction == 1) return grid[x][y-1];
60         else if(direction == 2) return grid[x-1][y];
61         else if(direction == 3) return grid[x][y+1];
62         else if(direction == 4) return grid[x+1][y];
63         return null;
64     }
65
66     public void action() {
67         Grid frontGrid = getFrontGrid();
68         if(frontGrid.tool == 0){
69             System.out.printfアクションができる場所ではありません ("\\n");
70             return;
71         }
72         /*if (this.food == null) {
73             System.out.println食材を持っていません! ("");
74             return;
75         }*/
76         if(food != null){
77             if(frontGrid.tool == 1 && food.canCut == true){
78                 AudioManager se = new AudioManager();
79                 se.playSE("./sound/music_cut2.wav");
80                 food.foodStatus = 2; これで切ったこととなるのだ// Kome
81                 //food.cut();
82                 System.out.printf食材を切りました ("\\n");
83                 return;
84             }else if(frontGrid.tool == 10 && food.canHeat == true){
85                 if(!frontGrid.hasFood()){
86                     AudioManager se = new AudioManager();
87                     se.playSE("./sound/music_boil.wav");
88                     frontGrid.food = food;
89                     food = null;
90                     System.out.println釜に米を入れました。("")デバッグ用;//
91                 }
92                 //System.out.printf食材をゆでました。("%のは sstatus%です
93                     d\\n", food.foodName, food.foodStatus);
94                 return;
95             }
96         }
97         else if(frontGrid.tool == 10 && frontGrid.hasFood() && frontGrid.
98             cookingGauge >= 60){
99             System.out.println炊けた米をとります。("");
100             frontGrid.food.foodStatus = 3;
101             food = frontGrid.food;
102             frontGrid.food = null;
103             frontGrid.cookingGauge = 0; 米をとったらリセット//
104             return;
105         }
106     }
107
108     public void pick_up() {
109         Grid currentGrid = grid[x][y]; 自分の足元のグリッド//
110         Grid frontGrid = getFrontGrid(); 自身の目の前のグリッド//
111         System.out.printf("frontGrid = (%d,%d)\\n", frontGrid.x, frontGrid.y);
112         if(frontGrid.tool == 10){return;} 鍋からはアクションでしか食材をとれない。
113         // Yoshida
114         if(hasPlate == false && frontGrid.tool == 3 ){ //は皿を持っていない player か
115             つ目の前マスが皿ボックス
116             AudioManager se = new AudioManager();
117             se.playSE("./sound/music_have.wav");
118             System.out.println皿を持ちました ("");
119             plate = new Plate(); ここで皿をもった//
120             hasPlate = true; 皿を持つ//
121         }else if(hasPlate == false && frontGrid.isPlatePlaced == true){ //は皿を持っ
122             ていない player かつ目の前マスに皿がある
123             AudioManager se = new AudioManager();
124             se.playSE("./sound/music_have.wav");
125             hasPlate = true; 皿を持つ//
126             plate = frontGrid.plate;
127             frontGrid.isPlatePlaced = false; 目の前マスから皿を回収//
128             frontGrid.plate = null;
129             //food = frontGrid.food;
130             //frontGrid.food = null;
131         }
132         else if (food == null) { // 何も持っていない場合
133             if(frontGrid.foodBox == 1){ 目の前のマスがキャベツボックスだったら//
134                 AudioManager se = new AudioManager();

```



```

131         se.playSE("./sound/music_have.wav");
132         this.food = new Cabbage();
133         System.out.printlnキャベツボックスから取得しました!("");
134     }
135     else if(frontGrid.foodBox == 2){ 目の前のマスがトマトボックスだったら//
136         AudioManager se = new AudioManager();
137         se.playSE("./sound/music_have.wav");
138         this.food = new Tomato();
139         System.out.printlnトマトボックスから取得しました!("");
140     }else if(frontGrid.foodBox == 3){ 目の前のマスがきゅうりボックスだったら//
141         AudioManager se = new AudioManager();
142         se.playSE("./sound/music_have.wav");
143         this.food = new Cucumber();
144         System.out.printlnきゅうりボックスから取得しました!("");
145     }else if(frontGrid.foodBox == 4){ 目の前のマスが米ボックスだったら//
146         AudioManager se = new AudioManager();
147         se.playSE("./sound/music_have.wav");
148         this.food = new Rice();
149         System.out.printlnライスボックスから取得しました!("");
150     }else if(frontGrid.foodBox == 5){ 目の前のマスがまぐろボックスだったら//
151         AudioManager se = new AudioManager();
152         se.playSE("./sound/music_have.wav");
153         this.food = new Tuna();
154         System.out.printlnマグロボックスから取得しました!("");
155     }else if(frontGrid.foodBox == 6){ 目の前のマスがいかボックスだったら//
156         AudioManager se = new AudioManager();
157         se.playSE("./sound/music_have.wav");
158         this.food = new Squid();
159         System.out.printlnイカボックスから取得しました!("");
160     }else if(frontGrid.foodBox == 7){ 目の前のマスがのりボックスだったら//
161         AudioManager se = new AudioManager();
162         se.playSE("./sound/music_have.wav");
163         this.food = new Seaweed();
164         System.out.printlnのりボックスから取得しました!("");
165     }
166
167     else if (frontGrid.hasFood()) { // 現在のマスに食材がある場合
168         AudioManager se = new AudioManager();
169         se.playSE("./sound/music_have.wav");
170         food = frontGrid.food; // 食材を拾う
171         frontGrid.food = null; // マスから食材を消す
172         System.out.println食材を持ち上げました!("");
173     } else {
174         System.out.printlnここには食材がありません。("");
175     }
176 }
177
178 }
179
180 public void put(){
181     Grid currentGrid = grid[x][y];
182     Grid frontGrid = getFrontGrid();
183     if(frontGrid.tool == 13){
184         hasPlate = false;
185         plate = null;
186         food = null;
187         System.out.printlnゴミ箱に捨てられました("");
188     }皿を持っていて
189     // 目の前がツールマスではなくカウンターでもない、目の前に食材なし
190     else if((hasPlate) && frontGrid.tool==0 && frontGrid.isCounter==false &&
191         frontGrid.food==null) {
192         hasPlate = false; 皿を捨てる置く//()
193         frontGrid.isPlatePlaced =true;
194         frontGrid.plate = plate; プレイヤーが持っている皿をグリッドにわたす//
195         plate = null; プレイヤーは皿を離す//
196     }皿を持って、目の前はツールマスではなくカウンターでもない、目の前に食材がある
197     //
198     else if((hasPlate) && frontGrid.tool==0 && frontGrid.isCounter==false &&
199         frontGrid.food!=null){
200         plate.add(frontGrid.food); まず最初に自分の//にを追加する。platefood
201         frontGrid.isPlatePlaced = true;
202         frontGrid.plate = plate;
203         plate = null;
204         hasPlate = false;
205         frontGrid.food = null;
206         System.out.printfデバッグ ("%n");
207         //plate.printPlate();
208     }
209     /* else */if(hasPlate==true && frontGrid.isCounter==true) { いま皿を持ってい

```

```

208         // かつ目の前がカウンター
209         System.out.printlnカウンターに提供します。("");
210         hasPlate = false; 皿を捨てる置く__()
211         frontGrid.plate = plate;
212         plate = null;
213         frontGrid.isPlatePlaced = true;
214         Order currentOrder = model.matchOrder(frontGrid.plate);
215         if(currentOrder == null){// 料理が失敗だったとき
216             System.out.println失敗作が提出されました("");
217             model.scoreDown(currentOrder);失敗した場合、回収されて減点
218             //
219             view.addWaiter(view.setPlateImage(frontGrid.plate));
220             hasPlate = false;
221             plate = null;
222             frontGrid.food = null;
223             frontGrid.plate = null;
224             frontGrid.isPlatePlaced = false;
225             return;
226         }else{ 注文が正しかったとき//
227             //view.addWaiter(currentOrder);
228             AudioManager se = new AudioManager();
229             se.playSE("./sound/music_success.wav");
230             view.addWaiter(view.setOrderImage(currentOrder));
231             model.scoreUp(currentOrder);
232             hasPlate = false;
233             frontGrid.plate = null;
234             frontGrid.food = null;
235             frontGrid.isPlatePlaced = false;
236         }
237         /* 個々のコード必要なのか問題があります。一応怖いので残してます// Kome
238         System.out.println(currentOrder.orderName + が提供されました! "");
239         if(currentOrder.isCompleted(frontGrid.plate) == true){
240             model.scoreUp(currentOrder);
241         }
242         else model.scoreDown(currentOrder);
243         */
244     }
245     if(food != null) { // 既に食材を持っている場合
246         if(frontGrid.isPlatePlaced == true){ 目の前のマスに皿が置いてある場//
247             System.out.println皿に食材を追加します!("");
248             frontGrid.plate.add(food);
249             food = null;
250             Order currentOrder = model.matchOrder(frontGrid.plate);
251             System.out.println皿に食材を追加しました!("");
252             frontGrid.plate.printPlate();
253         }else if (!frontGrid.hasFood() && frontGrid.tool == 0) { // 現在のマスが
254             空いている場合かつそのマスがツールマスではない
255             frontGrid.food = food; // 食材を置く
256             food = null; // 手持ちを空にする
257             System.out.println皿がないマスに対して食材を置きました!("");
258         }
259         else {
260             if(frontGrid.hasFood() == true) System.out.printlnここには既に食材があ
261             ります!("");
262             if(frontGrid.tool != 0) System.out.printfここはツールなので食材は置けま
263             せん("");
264         }
265     }
266 }

```

● Start

```

1 import javax.swing.*;
2 import java.awt.*;
3 import java.awt.event.ActionEvent;
4 import java.awt.event.ActionListener;
5 import java.io.File;
6 import java.io.IOException;
7
8 public class Start extends JPanel {
9     private MiniCook mainApp;
10    private Font pixelFont;
11
12    public Start(MiniCook mainApp) {
13        this.mainApp = mainApp; // MiniCook のインスタンスを保持
14
15        setLayout(new GridBagLayout()); // グリッドバグレイアウトを使用
16    }

```

```

17     GridBagConstraints gbc = new GridBagConstraints();
18     gbc.gridx = 0;
19     gbc.gridy = 0;
20     gbc.anchor = GridBagConstraints.CENTER;
21     gbc.insets = new Insets(20, 0, 20, 0); // 上下の余白を設定
22
23     // フォントを読み込む
24     loadCustomFont();
25
26     // タイトルラベルの作成
27     JLabel titleLabel = new JLabel("MiniCook", SwingConstants.CENTER);
28     titleLabel.setFont(pixelFont.deriveFont(100f));
29     add(titleLabel, gbc); // ラベルを追加
30
31     // スタートボタンの作成
32     JButton startButton = new JButton("Start");
33     startButton.setFont(pixelFont.deriveFont(80f));
34     startButton.addActionListener(new ActionListener() {
35         @Override
36         public void actionPerformed(ActionEvent e) {
37             AudioManager se = new AudioManager();
38             se.playSE("./sound/music_start2.wav");
39             mainApp.startGame(); // MiniCook のstartGame() を呼び出し
40         }
41     });
42
43     gbc.gridy = 1; // ボタンを行目に配置 2
44     add(startButton, gbc); // ボタンを追加
45 }
46
47 private void loadCustomFont() {
48     try {
49         File fontFile = new File("font/ByteBounce.ttf"); // フォントのパス
50         pixelFont = Font.createFont(Font.TRUETYPE_FONT, fontFile);
51     } catch (IOException | FontFormatException e) {
52         e.printStackTrace();
53         pixelFont = new Font("Monospaced", Font.PLAIN, 24); // フォールバック用
54         // フォント
55     }
56 }

```

● Result

```

1 import javax.swing.*;
2 import java.awt.*;
3 import java.awt.event.ActionEvent;
4 import java.awt.event.ActionListener;
5 import java.io.File;
6 import java.io.IOException;
7
8 public class Result extends JPanel {
9     private MiniCook mainApp;
10    private Font pixelFont;
11    private int score;
12    private JLabel scoreLabel; // スコア表示用ラベル
13    private JLabel starLabel;
14
15    public Result(MiniCook mainApp) {
16        this.mainApp = mainApp;
17        this.score = 0; // 初期スコア
18
19        setLayout(new GridBagLayout());
20        GridBagConstraints gbc = new GridBagConstraints();
21        gbc.gridx = 0;
22        gbc.anchor = GridBagConstraints.CENTER;
23        gbc.insets = new Insets(20, 0, 20, 0);
24
25
26
27        // フォントを読み込む
28        loadCustomFont();
29
30        // タイトルラベル
31        JLabel titleLabel = new JLabel("Result", SwingConstants.CENTER);
32        titleLabel.setFont(pixelFont.deriveFont(100f));
33        gbc.gridy = 0;
34        add(titleLabel, gbc);
35
36        // スコアラベル (変更可能にする)

```

```

37     scoreLabel = new JLabel("Score : " + score, SwingConstants.CENTER);
38     scoreLabel.setFont(pixelFont.deriveFont(80f));
39     gbc.gridy = 1;
40     add(scoreLabel, gbc);
41
42     starLabel = new JLabel(getStarRating(score), SwingConstants.CENTER);
43     starLabel.setFont(new Font("Meiryo", Font.PLAIN, 80));
44     gbc.gridy = 2;
45     add(starLabel, gbc);
46
47     // ボタンパネル
48     JPanel buttonPanel = new JPanel(new FlowLayout(FlowLayout.CENTER, 20, 10)
49         );
50
51     JButton restartButton = new JButton("Restart");
52     restartButton.setFont(pixelFont.deriveFont(50f));
53     restartButton.setPreferredSize(new Dimension(300, 100));
54     restartButton.addActionListener(e -> mainApp.restartGame());
55
56     JButton closeButton = new JButton("Close");
57     closeButton.setFont(pixelFont.deriveFont(50f));
58     closeButton.setPreferredSize(new Dimension(300, 100));
59     closeButton.addActionListener(e -> System.exit(0));
60
61     buttonPanel.add(restartButton);
62     buttonPanel.add(closeButton);
63
64     gbc.gridy = 3;
65     add(buttonPanel, gbc);
66
67     // スコアを更新するメソッド (ゲーム終了時に呼び出す)
68     public void updateScore(int newScore) {
69         this.score = newScore;
70         scoreLabel.setText("Score : " + score);
71         starLabel.setText(getStarRating(score));
72         repaint(); // 再描画
73         revalidate(); // レイアウト更新
74     }
75
76     // スコアに応じた星の文字列を返す
77     private String getStarRating(int score) {
78         if (score >= 500) {
79             return "\u2605 \u2605 \u2605"; // ★★★
80         } else if (score >= 250) {
81             return "\u2605 \u2605 \u2606"; // ★★☆
82         } else if (score > 0) {
83             return "\u2605 \u2606 \u2606"; // ★☆☆
84         } else {
85             return "\u2606 \u2606 \u2606"; // ☆☆☆
86         }
87     }
88
89     private void loadCustomFont() {
90         try {
91             File fontFile = new File("font/ByteBounce.ttf");
92             pixelFont = Font.createFont(Font.TRUETYPE_FONT, fontFile);
93         } catch (Exception e) {
94             e.printStackTrace();
95             pixelFont = new Font("Monospaced", Font.PLAIN, 24);
96         }
97     }
98 }

```

- Meal

```

1  import javax.swing.*;
2  import java.awt.*;
3  import java.awt.event.*;
4  import java.awt.image.ImageObserver;
5
6  abstract class Food { 継承させる前提の//クラス abstract
7      public int foodStatus; 食材のステータスの変数、何もしてなければになる//0 カットしてたら
8          1
9      public boolean canCut; その食材がカット可能なら//true
10     public boolean canHeat; その食材が加熱可能なら//true
11     public boolean isOnPlate; 皿の上に置かれているか//
12     public String foodName;
13     public abstract int getFoodStatus();

```

```
14     public Food(int foodStatus, boolean canCut, boolean canHeat, boolean
15         isOnPlate, String foodName){
16         this.foodStatus = foodStatus;
17         this.canCut = canCut;
18         this.canHeat = canHeat;
19         this.isOnPlate = isOnPlate;
20         this.foodName = foodName;
21     }
22 }
23 //クラスを継承したクラスです FoodCabbage
24 class Cabbage extends Food{
25     public Cabbage(){
26         super(1, true, false, false, "cabbage");
27     }
28
29     public int getFoodStatus(){ そのフードの状態を返す//
30         return foodStatus;
31     }
32 }
33 //クラスを継承したクラスです FoodTomato
34 class Tomato extends Food{
35     public Tomato(){
36         super(1, true, false, false, "tomato");
37     }
38     public int getFoodStatus(){ そのフードの状態を返す//
39         return foodStatus;
40     }
41 }
42 //クラスを継承したクラスです Foodcucumber
43 class Cucumber extends Food{
44     public Cucumber(){
45         super(1, true, false, false, "cucumber");
46     }
47     public int getFoodStatus(){ そのフードの状態を返す//
48         return foodStatus;
49     }
50 }
51 //クラスを継承したクラスです Foodrice
52 class Rice extends Food{
53     public Rice(){
54         super(1, false, true, false, "rice");
55     }
56     public int getFoodStatus(){ そのフードの状態を返す//
57         return foodStatus;
58     }
59 }
60 //クラスを継承したクラスです Foodtuna
61 class Tuna extends Food{
62     public Tuna(){
63         super(1, true, false, false, "tuna");
64     }
65     public int getFoodStatus(){ そのフードの状態を返す//
66         return foodStatus;
67     }
68 }
69 //クラスを継承したクラスです Foodsquid
70 class Squid extends Food{
71     public Squid(){
72         super(1, true, false, false, "squid");
73     }
74     public int getFoodStatus(){ そのフードの状態を返す//
75         return foodStatus;
76     }
77 }
78 //クラスを継承したクラスです Foodseaweed
79 class Seaweed extends Food{
80     public Seaweed(){
81         super(1, false, false, true, "seaweed");
82     }
83     public int getFoodStatus(){ そのフードの状態を返す//
84         return foodStatus;
85     }
86 }
87 class Plate {
88     Food[] foods;
89     public Plate(){
90         foods = new Food[3];
91         foods[0] = null;
92         foods[1] = null;
```

```

93     foods[2] = null;
94 }
95
96 public boolean hasAnyFood(){ //になにかしら乗っているかの plateboolean
97     if(foods[0]==null && foods[1]==null && foods[2]==null) return false;
98     else return true;
99 }
100
101 public void add(Food food) {
102     for (int i = 0; i < foods.length; i++) {
103         if(foods[i] != null && foods[i].foodName == food.foodName) { continue;
104         }
105         if (foods[i] == null) {
106             foods[i] = food;
107             System.out.println(food.foodName + " を皿に追加しました。");
108             return; // 追加が完了したら終了
109         }
110     }
111     System.out.printlnこれ以上皿に食材を追加できません。("");
112 }
113
114 public Food get(int i){
115     if(i<0 || i>=foods.length){return null;}
116     else return foods[i];
117 }
118
119 public void printPlate(){
120     String state = "";
121     System.out.print現在、皿の上には：("");
122     for(int i=0; i<3; i++){
123         if(foods[i] != null) {
124             switch(foods[i].foodStatus){
125                 case 1: state = "raw"; break;
126                 case 2: state = "cut"; break;
127                 case 3: state = "grilled"; break;
128             }
129             System.out.print(foods[i].foodName+"(" + state + ")" + " ");
130         }
131     }
132     System.out.print("\n");
133     return ;
134 }
135
136 public boolean matchesOrder(Order order) {
137     boolean[] matchedIngredients = new boolean[3];
138     Food[] orderIngredients = {order.ingredient1, order.ingredient2, order.ingredient3};
139
140     // 皿にある食材の数をカウント
141     int plateFoodCount = 0;
142     for (int i=0; i<3; i++) {
143         if (foods[i] != null) {
144             plateFoodCount++;
145         }
146     }
147
148     // オーダーの食材リストを作成
149     int orderFoodCount = 0;
150     for (int i=0; i<3; i++) {
151         if (orderIngredients[i] != null) {
152             orderFoodCount++;
153         }
154     }
155
156     // オーダーの食材数と皿の食材数が違ったら不一致とする****
157     if (plateFoodCount != orderFoodCount) {
158         System.out.println料理の食材数がオーダーと一致しません。("");
159         return false;
160     }
161
162     for (int i = 0; i < foods.length; i++) {
163         for (int j = 0; j < orderIngredients.length; j++) {
164             if(orderIngredients[j] == null){
165                 matchedIngredients[j] = true;
166                 continue;
167             }
168             if (!matchedIngredients[j] && foods[i] != null) {
169                 if (foods[i].getClass() == orderIngredients[j].getClass() &&
170                     foods[i].foodStatus == orderIngredients[j].foodStatus) {
171                     System.out.println(foods[i].foodName + は満たされました。 "");

```

```

171         matchedIngredients[j] = true;
172         break;
173     }
174     }
175 }
176 }
177
178 for(int i=0; i<matchedIngredients.length; i++){
179     if(matchedIngredients[i]){
180         System.out.println材料 (""+(i+1)は満たされています。+""");
181     }
182     else System.out.println材料 (""+(i+1)は満たされいません。+""");
183 }
184
185 for (boolean matched : matchedIngredients) {
186     if (!matched){
187         System.out.println料理は未完成です。("");
188         return false;
189     }
190 }
191 System.out.println料理は完成しています。("");
192 return true;
193 }
194 }
195 }

```

- Other

```

1 import javax.swing.*;
2 import java.awt.*;
3 import java.awt.event.*;
4 import java.awt.image.ImageObserver;
5
6 class Grid {
7     int x, y;
8     boolean wall = false;
9     boolean obstacle = false;
10    Food food = null;
11    Plate plate = null; 各グリッドは//という食材をいくつか持つクラスを持つ Plate
12    public boolean isPlatePlaced = false; そのマスにさらがおかれているか//
13    public int foodBox = 0; フードボックスがキャベツなら、トマトならみたいな感じボックスが無ければ//12...(0) Yoshida
14    public boolean plateBox = false; 皿ボックスだった場合//になる true
15    /*はツールではない
16    0,は包丁
17    1,はキャベツボックス
18    2,は皿ボックス
19    3,トマトボックス
20    4:,キュウリ
21    5:,米
22    6:,マグロ
23    7:,イカ
24    8:,のり
25    9:,なべ
26    10:,なべ米
27    11:(),フライパン
28    12:ゴミ箱
29    13:キャンドル特に効果はない
30    14:()
31    */
32    public int tool = 0;
33    boolean isCounter; そのマスがカウンターではないか//
34    public float cookingGauge = 0; ご飯を炊いてる時のゲージ用 //Yoshida
35
36    public Grid(int x, int y) { this.x = x; this.y = y; }
37
38    public boolean hasFood() { return food != null; }
39 }
40
41
42 class Waiter{
43     int waitY = 1000; ウェイタースタンバイ位置//
44     int receiveY = 710; ウェイターが料理を受け取る場所//
45     boolean active = true;
46     private Image imgMeal;
47     private Image imgWaiterUp;
48     private Image imgWaiterDown;
49     DrawModel model;

```

```

50 static final int xBefore = 470;
51 static final int xAfter = 470;
52 static final int counterX = 7;
53 static final int counterY = 8;
54 final int headerBlank;
55 final int rightBlank;
56 final int cellsize;
57 int playerX;
58 int flame = 0;
59 static final int comeFlame = 90; ウェイターが来るときの片道のフレーム数//;
60 public Waiter(DrawModel model, Image imgMeal, Image imgWaiterDown, Image
    imgWaiterUp, int headerBlank, int rightBlank, int playerX){
61     this.model = model;
62     this.imgMeal = imgMeal;
63     this.cellsize = model.getCellSize();
64     this.headerBlank = headerBlank;
65     this.rightBlank = rightBlank;
66     this.imgWaiterDown = imgWaiterDown;
67     this.imgWaiterUp = imgWaiterUp;
68     this.playerX = playerX;
69 }
70 public void drawMe(Graphics g, ImageObserver io){
71     final int cS = cellsize;
72     if(0 <= flame && flame < comeFlame){
73         g.drawImage(imgMeal, playerX*cellsize + rightBlank, counterY*cellsize
            + headerBlank, cS, cS, io);仮で正方形を描画して
            るよ
74         //
75         g.setColor(Color.pink);
76         g.drawImage(imgWaiterUp, xBefore-10, (int)((waitY*(comeFlame-flame) +
            receiveY*flame)/comeFlame) + rightBlank, cS+20, cS+20, io);
77         //g.fillRect(xBefore, (int)((waitY*(comeFlame-flame) + receiveY*flame)
            /comeFlame) + rightBlank, cS, cS);
78         flame++;
79     }else if(comeFlame <= flame && flame < 2*comeFlame){
80         g.drawImage(imgWaiterUp, xBefore-10, receiveY + rightBlank, cS+20, cS
            +20, io);
81         //g.fillRect(xBefore, receiveY + rightBlank, cS, cS);
82         flame++;
83     }else if(2*comeFlame <= flame && flame < 3*comeFlame){
84         //g.drawRect(xBefore, (int)((waitY*(flame-2*comeFlame) + receiveY*(3*
            comeFlame-flame))/comeFlame) + rightBlank, cS, cS);
85         g.drawImage(imgWaiterDown, xAfter-10, (int)((waitY*(flame-2*comeFlame)
            + receiveY*(3*comeFlame-flame))/comeFlame) + rightBlank, cS+20, cS
            +20, io);
86         flame++;
87     }else if(flame == 3*comeFlame){ active = false; flame++;}
88 }
89 }

```

- AudioManager

```

1 import javax.sound.sampled.*;
2 import java.io.File;
3 import java.io.IOException;
4
5 public class AudioManager {
6     private static Clip bgmClip;
7
8     // を再生 (のみ対応) BGMWAV
9     public static void playBGM(String filePath) {
10         stopBGM(); // 既存のを停止 BGM
11         try {
12             File soundFile = new File(filePath);
13             AudioInputStream audioStream = AudioSystem.getAudioInputStream(
                soundFile);
14             bgmClip = AudioSystem.getClip();
15             bgmClip.open(audioStream);
16             bgmClip.loop(Clip.LOOP_CONTINUOUSLY); // ループ再生
17             bgmClip.start();
18         } catch (UnsupportedAudioFileException | IOException |
            LineUnavailableException e) {
19             e.printStackTrace();
20         }
21     }
22
23     // 停止 BGM
24     public static void stopBGM() {
25         if (bgmClip != null && bgmClip.isRunning()) {
26             bgmClip.stop();

```



```
27     }
28 }
29
30 // を再生 (のみ対応) SEWAV
31 public static void playSE(String filePath) {
32     new Thread(() -> {
33         try {
34             File soundFile = new File(filePath);
35             AudioInputStream audioStream = AudioSystem.getAudioInputStream(
36                 soundFile);
37             Clip seClip = AudioSystem.getClip();
38             seClip.open(audioStream);
39             seClip.start(); // 短いならそのまま再生 SE
40         } catch (UnsupportedAudioFileException | IOException |
41             LineUnavailableException e) {
42             e.printStackTrace();
43         }
44     }).start();
45 }
46
47 /*
48 import javafx.scene.media.Media;
49 import javafx.scene.media.MediaPlayer;
50 import java.io.File;
51
52 public class AudioManager {
53     private static MediaPlayer bgmPlayer;
54
55     // を再生するメソッド BGM
56     public static void playBGM(String filePath) {
57         if (bgmPlayer != null) {
58             bgmPlayer.stop(); // すでに再生中なら停止
59         }
60         Media media = new Media(new File(filePath).toURI().toString());
61         bgmPlayer = new MediaPlayer(media);
62         bgmPlayer.setCycleCount(MediaPlayer.INDEFINITE); // ループ再生
63         bgmPlayer.play();
64     }
65
66     // を停止するメソッド BGM
67     public static void stopBGM() {
68         if (bgmPlayer != null) {
69             bgmPlayer.stop();
70         }
71     }
72
73     // を再生するメソッド (複数同時再生可能) SE
74     public static void playSE(String filePath) {
75         Media media = new Media(new File(filePath).toURI().toString());
76         MediaPlayer sePlayer = new MediaPlayer(media);
77         sePlayer.play();
78     }
79 }
80 */
```