

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 19 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。文責：鈴木

2 設計方針

図1にクラス図を示す。

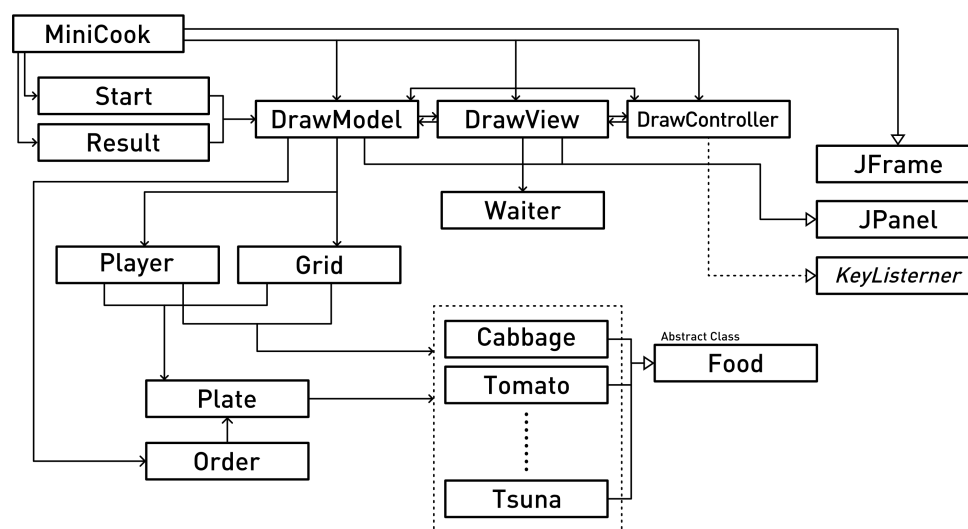


図1 クラス図

クラス図に示しているように、MiniCook というクラスが大元のクラスとなっている。その中でインスタンスとして5つのクラスを保持している。Start クラス・Result クラスはゲーム開始前の画面と、ゲーム終了時にスコアの表示等を行うためのクラスである。そして設計方針について、このゲームはプレイヤーがオーダーに従って各種料理をつくる。その中で、プレイヤーは皿や食材をいろんな座標に置くというプロセスが起きる。それに適応、そして拡張性を保つような構成にした。詳細に関しては以下で説明する。今回のプログラムは大量生産するインスタンスが存在しないという想定を最初の構想で予測したため Observer モデルは用いずに、基本的な MVC モデルを元にして作成した。

- Model

DrawModel クラスでは、各種データの管理とそれに伴ったメソッドの提供をおこなった。基本的にゲームの情報は各種クラスから Model に参照されて提供する。Player, Grid などの基盤にあるようなクラスはここでインスタンスを作成している。

- View

DrawView では、Model クラスより取得した情報を元に一括で描画処理を行う。フィールドのどの場所にどの物を描画するか元情報を Model より取得。その後その情報を元に画像を View クラスのメソッドを用いて判定して、画像を選択・描画している。当初の予定では、完全に 2D でゲームを作成する予定であったが、途中で擬似 3D にして立体感を出そうという構想が生まれた。しかし View モデルで一括で管理しているおかげで、プログラムの書き直しは最低限に抑えることができた。

- Controller

DrawController クラスでは、基本的にはプレイヤーからの入力を受取のみを行う。それぞれキーボードの入力を受取、それに応じた動作をそれぞれのクラス内のメソッドでおこなってもらう。

しかしゲームに動的なアニメーションを少し付ける都合でキー入力を行いたいときと行いたくないときがある。それに対応するために、あるクラスのメソッドを呼び出すときもあれば、キー入力中に boolean のフラグを用いて、動作先で参照してもらう形になっているものもある。

- Player・Grid

この2つのクラスは、このプログラムのいちばん重要なクラスである。名称が違うもののプレイヤーが食材を保持している場合とあるマス目が食材を持っている(食材を置いている)という違いがあるのみで、ほとんど同じものである。このクラスでは、そのマスないしはプレイヤーがなにを持っているかというクラスをインスタンス変数に保持している。そして、Player に許された行動や Grid(マス目)によってできる。行動についての自身の情報を持っており、それに対応したメソッドを提供している。

- Order

Order クラスでは、画面上部に定期的なタイミングで出現する、提供しなければならない料理の情報を持っているクラスである。注文1つごとにこのクラスが生成されて、その中にオーダーの制限時間、必要な材料などの情報をしている。

- Food

この Food クラスが、食材に関しての最小単位となるクラスである。抽象クラスという定義をしており、これを継承してキャベツであったり、トマトであったりのクラスを作成している。それぞれ継承されたクラスにおいて、それぞれ特有の調理される方法や、調理された情報を保持することができる。これを複数個ミックスして料理となったものが後述する Plate クラスである。

- Plate

この Plate クラスでは、Food クラスをいくつか保持していて、それによって例えばキャベツとトマトのサラダであったり、魚の切り身と海苔で巻き寿司といったものになる。これが Order に存在していれば正解、なければ不正解という形である。また各マス目と Player は Food クラスを単体で保持して食材を持っていたり、Plate クラスを持っていて、複数の食材からなる料理を持っていたりする。なお正誤判定については Order クラスで行わずこちらで Order クラスの内容を Model を経由して取得して、自分との合致があるかどうかで行っている。

クラス間の関係と全体の参照の流れを説明する。ほとんどの基本の流れは Model クラスを参照して行われる。ユーザーからの入力 DrawController から Model へ、描画は Model を参照して DrawVier クラスなどで行われる。プレイヤーは移動をして該当の場所に移動してアクションを行うことで、Food クラスを新たに生成したり、その場においたり、またそれらを調理してまとめて Plate クラスに保持する。それを提出口に提出した際に、現存しているオーダーとの正誤判定を行いスコアのアップダウンを行う。ここでは説明を省略したが、各種タイミングで SE や BGM を鳴らすようなコードも含まれている。

文責：米谷

3 プログラムの説明

以下にクラスとその説明を示す。

- MiniCook

このクラスでは、まず最初に MVC モデルの各種クラスを生成する。そして各種クラスの中にある、getter,setter を利用してそれぞれのクラスを連携させる。それに加えてオーディオの管理のクラスを生成したして、JFrame を用いてウィンドウを立ち上げる。

このインスタンスメソッドとしてスタート画面からゲームが画面に切り替える startGame() であったり、リザルト画面に切り替える showResult() というメソッド等を持っている。public static void main() は個々で宣言がしてあるため、このゲームは java MiniCook でスタートする。

文責：米谷

- Model

- Food

この Food クラスは、料理の食材を表現するための抽象クラスである。インスタンス変数として、食材の状態を整数値で表現するための foodStatu、それぞれの調理法が可能かどうかを示すフラグである canCut、canHeat、食材が皿の上にあるかどうかを示すフラグ isPlate、食材の名前を保持するための文字列 foodName を持つ。コンストラクタでは継承した子クラスの食材に併せて初期化が行えるように実装している。

文責：吉田

- Order

この Order は、料理の注文を管理するクラスである。注文の基本情報を保持するインスタンス変数として、注文の名前を文字列で管理する orderName、何個目のオーダーであるかを表す orderIndex、アニメーション用の座標を表す posAnim、subOrderPosY、subOrderPosYAnim を持つ。また、食材に関するインスタンス変数として、皿を持っているかのフラグである hasPlate、

必要な食材を表す `ingredient1 3` がある。さらに、注文の制限時間である `timeLimit`、注文が生成された時間を示す `createTime`、自動削除用のタイマーである `expirationTimer` をインスタンス変数として持つ。制限時間が経過した場合は、効果音を鳴らし、スコアを下げ、注文が削除される。

コンストラクタでは注文の名前や必要な食材、制限時間を注文ごとに設定できるようになっている。

注文の完成判定を行う `isCompleted` メソッドは `Plate` クラスのオブジェクトを引数に持つ。プレイヤーが作った料理である `plate.food` と注文の材料である `orderIngredients` を 1 つずつ比較して、一致していれば判定用の配列を `true` とする。全ての食材が揃っていれば `true` を返す。

残り時間を計算するメソッドとして `getRemainingTime` がある。これは現在時刻から注文作成時刻を引くことで経過時間を計算し、`timeLimit` から経過時間を引くことで残り時間を取得する。

`getRemainingTime` が 0 以下であるかで注文の期限切れを判定する `isExpired` メソッドと、手動で注文を削除する際にタイマーを停止するための `cancelTimer` メソッドも用意している。

文責：吉田

- View

この `DrawView` クラスは、`JPanel` を継承した、描画処理用のクラスである。ゲームの基本画面やプレイヤー、食材、ツール、オーダー、ウェイターなどを描画する。`DrawModel` を参照している。インスタンス変数として `model, cont, size` などを持つ。

背景やプレイヤー、オーダー、UI などの基本的なゲーム画面の描画については `paintComponent` で行っている。ウェイター描画は `addWaiter` で行っている。

食材の描画で `setFoodImage`、皿の描画で `setPlateImage`、オーダーの描画で `setOrderImage` が用いられる。これら関数では、引数として情報を受け取ってそれに対して適切な画像を返す。その判断は `if` 文や `switch` 文を用いているが、特に皿の上で食材を組み合わせた場合に、載っているべき食材を指定するだけでは、載っていないべき食材の有無に関わらず完成してしまう。そのため全ての食材の種類と加工の種類を指定しなければならないが毎行書くことは冗長で現実的ではない、ということに苦労した。一気に食材を指定するのではなく、まず先に共通して使っていない食材をジャンルごとに指定し、そのジャンルの `if` 文内で、持っている物持っていない物を指定することで文章量や比較を減らした。

文責：鈴木

– Player

- Controller
- Start
- Result
- CardLayout
- AudioManager

文責：

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後の画像 (f) に切り替わる。

ゲーム画面：組み合わせ

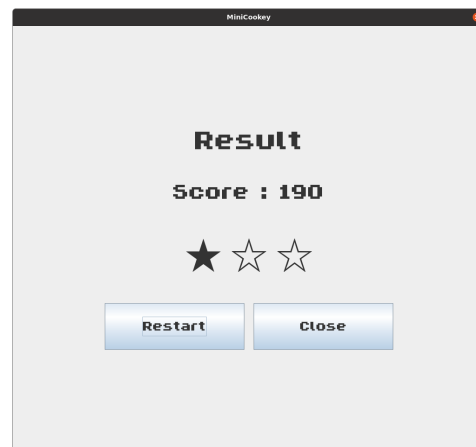
皿の上に各食材を載せると画像がそれに伴い完成品 (g) となる。

ゲーム画面：提供

完成した料理を提供口に置くと、ウェイターが取りに来る (h)。



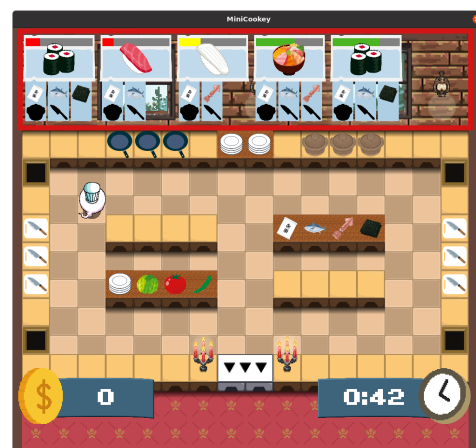
(a) スタート画面



(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



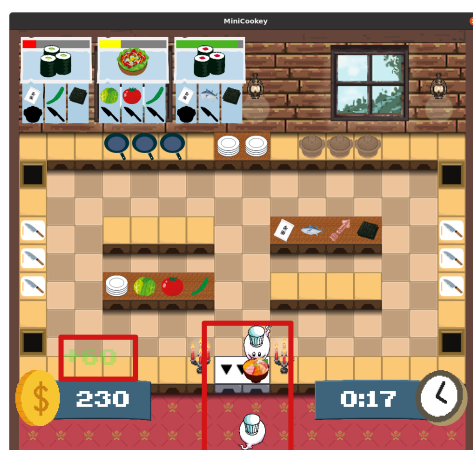
(e) ゲーム画面：加工前



(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

文責：鈴木

5 考察

予定していた以上のものが完成した。文責：吉田

6 感想

(米谷祐希)

(鈴木早紀)

授業前半の個人の課題を最低限しか取り組まなかったために、2人より Java を理解していなくて2人に大変な部分を多く任せてしまいました。2人が進んでやってくれたので感謝しています。画像・音楽の準備やメニューの追加、スライドやレポートは積極的に行えたと思います。View としての課題は、変数や画像読み込みが多すぎることで、今後食材やメニューの追加を行うときにもひたすらこれを書いていくのは厳しいと感じました。せめて別ファイルにするなどして、View.java 内はシンプルにする方が分かりやすいのかなと思いました。また、メニューによって食材や調理方法を指定するときは、分量が少なくなるように if 文の順序に工夫はしましたが、他の班の発表を聞き、csv ファイルの読み込みにすることで管理もしやすくなるのかなと考えました。今回初めて本格的にグループプログラミングを行ったので、共同作業をする大変さや、作業を分割する便利さを知ることができました。先輩や2人のプログラムを特に参考にして理解を進めることができました。Java はこの授業で初めて触ったけれど、半年間という期間を考慮すると大きな成果が得られたと感じます。

(吉田陽音)

授業前半では Java の基礎知識やオブジェクト指向について学ぶことができたが、与えられた課題をこなすだけで受け身の学びであった。一方後半の、このゲーム製作では積極的に Java についての理解を深めていくことができた。Java はこの授業で初めて触ったので、ゲーム製作の課題を聞いた当初はそれなりの形にできれば良いかなと思っていたが、実際に製作を進めていくうちに夢中になり、楽しみながらゲームを作ることができた。

反省点としては、行き当たりばったりな開発となってしまう、クラスが煩雑になってしまったり、メソッドが冗長になってしまったりしたことである。もっと計画的な開発が行えていたら、他の要素を実装する時間が生まれ、より良いゲームを作れたと反省している。

世の中に存在しているゲームに比べると簡易的なゲームであるが、素人なりにかかなりの労力や知識を詰め込んだ気でいたため、友人や家族にこのゲームを見せた時にあまり良いリアクションを得られなかったことがかなりショックであった。この授業では、Java についてだけでなく、こうした体験を通して学ぶことも多く、さらにはグループ開発を経験できたこともあり、自分にとってかなり有意義であったと実感している。

付録 1：操作マニュアル

(ストーリー)

キミはおばけの国の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たくさん作ろう！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で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。

(ゲーム画面)

ゲーム画面は図3のように、オーダー、キャラクター、食材・皿ボックス、スコア、調理器具、提供口、制限時間で構成されている。

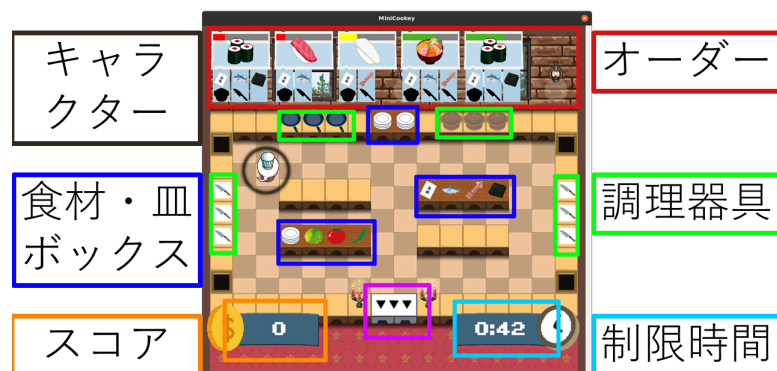


図3 ゲーム画面の説明

(データ)

- 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

文責：鈴木

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     private CardLayout cardLayout;
11     private JPanel cardPanel;
12
13     public MiniCook() {
14         System.out.printf("\n---Start---\n\n"); //見やすいように
15         model = new DrawModel();
16         view = new DrawView(model);
17         cont = new DrawController(model, view, this);
18         audio = new AudioManager();
19
20         model.getPlayer().setController(cont);
21         model.getPlayer().setView(view);
22         view.setController(cont);
23         view.addKeyListener(cont);
24
25         this.setBackground(Color.WHITE);
26         this.setTitle("MiniCook");
27         this.setSize(1016, 950);
28         this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
29         setLocationRelativeTo(null);
30
31         // カードレイアウトの設定
32         cardLayout = new CardLayout();
33         cardPanel = new JPanel(cardLayout);
34
35         // 各画面の追加
36         Start startScreen = new Start(this);
37         resultScreen = new Result(this);
38
39         cardPanel.add(startScreen, "start");
40         cardPanel.add(resultScreen, "result");
41
42         // ゲーム画面
43         JPanel gamePanel = new JPanel(new BorderLayout());
44         gamePanel.add(view, BorderLayout.CENTER);
45
46         cardPanel.add(gamePanel, "game");
47
48         add(cardPanel);
49         cardLayout.show(cardPanel, "start");
50     }
51
52     // スタート画面からゲーム画面に切り替える
53     public void startGame() {
54         cardLayout.show(cardPanel, "game");
55         cont.startGame();
56         // キーボード入力を受け取るためにフォーカスを設定
57         view.requestFocusInWindow();
58     }
59
60     // ゲーム終了時にリザルト画面を表示する
61     public void showResult() {
62         audio.stopBGM();
63         System.out.println("リザルト画面を表示します。");
64         resultScreen.updateScore(model.score);
65         cardLayout.show(cardPanel, "result");
66     }
67
68     // リザルト画面からもう一度プレイ
69     public void restartGame() {
70         audio.playBGM("./sound/music_background2.wav");
71         model.reset(); // ゲームデータをリセット (必要なら実装)
72         startGame(); // ゲームを開始
73     }
74
75     public static void main(String[] args) {
76         new MiniCook().setVisible(true);
77     }
78 }

```

● 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!
10    protected Grid[][] grid;
11    private Player player;
12    private Food food;
13    public int score;
14    public Order[] orders; //を入れる配列order
15    private int gameTime;
16
17    public DrawModel() {
18        gameTime = 120; //ゲーム時間秒()
19        score = 0;
20        orders = new Order[5];
21        for(int i=0; i<5; i++){
22            orders[i] = null;
23        }
24        grid = new Grid[xsize][ysize];
25        //imageGrid = new int[xsize][ysize];
26        for (int i = 0; i < xsize; i++) {
27            for (int j = 0; j < ysize; j++) {
28                grid[i][j] = new Grid(i, j);
29                //imageGrid[i][j] = '\0';
30                if (i == 0 || j == 0 || i == xsize - 1 || j == ysize - 1) {
31                    grid[i][j].wall = true; // 外周を壁に設定
32                }
33            }
34        }
35        player = new Player(2, 2, this, grid);
36
37        //以下で所定の場所に所定のオブジェクトを配置
38        grid[3][3].obstacle = true;
39        grid[4][3].obstacle = true;
40        grid[5][3].obstacle = true;
41        grid[6][3].obstacle = true;
42        grid[9][5].obstacle = true;
43        grid[10][5].obstacle = true;
44        grid[11][5].obstacle = true;
45        grid[12][5].obstacle = true;
46
47        grid[4][5].foodBox = 1;
48        grid[4][5].obstacle = true;
49        grid[4][5].tool = 2;
50
51        grid[5][5].foodBox = 2;
52        grid[5][5].obstacle = true;
53        grid[5][5].tool = 4;
54
55        grid[6][5].foodBox = 3;
56        grid[6][5].obstacle = true;
57        grid[6][5].tool = 5;
58
59        grid[9][3].foodBox = 4;
60        grid[9][3].obstacle = true;
61        grid[9][3].tool = 6;
62
63        grid[10][3].foodBox = 5;
64        grid[10][3].obstacle = true;
65        grid[10][3].tool = 7;
66
67        grid[11][3].foodBox = 6;
68        grid[11][3].obstacle = true;
69        grid[11][3].tool = 8;
70
71        grid[12][3].foodBox = 7;
72        grid[12][3].obstacle = true;
73        grid[12][3].tool = 9;
74
75        //カウンターを設置 Yoshida
76        grid[7][8].wall = true;
77        grid[7][8].isCounter = true;
78        grid[8][8].wall = true;
79        grid[8][8].isCounter = true;
80
81        grid[0][3].tool = 1; //ナイフ
82        grid[0][4].tool = 1; //ナイフ
83        grid[0][5].tool = 1; //ナイフ
84        grid[15][3].tool = 1; //ナイフ
85        grid[15][4].tool = 1; //ナイフ
86        grid[15][5].tool = 1; //ナイフ
87
88        grid[10][0].tool = 10; //なべ
89        grid[11][0].tool = 10; //なべ
90        grid[12][0].tool = 10; //なべ
91
92        grid[3][0].tool = 12; //フライパン
93        grid[4][0].tool = 12; //フライパン
94        grid[5][0].tool = 12; //フライパン
95
96        grid[3][5].plateBox = true;
97        grid[3][5].obstacle = true;
98        grid[3][5].tool = 3;
99
100       grid[7][0].plateBox = true;
101       grid[7][0].tool = 3; //皿ボックス
102       grid[8][0].plateBox = true;
103       grid[8][0].tool = 3; //皿ボックス
104
105       grid[0][1].tool=13;
106       grid[0][7].tool=13;
107       grid[15][1].tool=13;
108       grid[15][7].tool=13;
109
110       grid[6][8].tool = 14;
111       grid[9][8].tool = 14;
112   }
113

```

```

114     public Grid[][] getGrid() {
115         return grid;
116     }
117
118     public int[] getFieldSize() {
119         return new int[]{xsize, ysize};
120     }
121
122     public int getCellSize() {
123         return cellSize;
124     }
125
126     public Player getPlayer() {
127         return player;
128     }
129
130     public Food getFood() {
131         return food;
132     }
133
134     public void movePlayer(int dx, int dy) {
135         player.move(dx, dy, grid);
136     }
137     public void printInfo(){
138         System.out.println("デバッグ用情報<>");
139         // デバッグ用
140         System.out.println("配列の状態orders:");
141         for (int i = 0; i < 3; i++) {
142             if (orders[i] != null) {
143                 System.out.println("orders[" + i + "]: " + orders[i].orderName);
144             } else {
145                 System.out.println("orders[" + i + "]: null");
146             }
147         }
148     }
149
150     public void generateOrder() {
151         String[] menu={"salad","tekkamaki","kappamaki","tunanigiri","ikanigiri","kaisendon"};
152         int num_menu=6;
153         Random random=new Random();
154         for (int i = 0; i < orders.length; i++) {
155             if (orders[i] == null) {
156                 System.out.println("orders[" + i + "]はnull、新しいオーダーを生成します");
157                 String randommenu=menu[random.nextInt(num_menu)];
158                 orders[i] = new Order(randommenu, i, this);
159                 System.out.println("生成されたオーダー: " + orders[i].orderName);
160                 break;
161             } else {
162                 System.out.println("orders[" + i + "]は存在しています: " + orders[i].orderName);
163             }
164         }
165     }
166     public Order matchOrder(Plate plate) {
167         for (Order order : orders) {
168             if (order != null && plate.matchesOrder(order)==true) {
169                 System.out.println(order.orderName + "が完成!");
170                 return order;
171             }
172         }
173         return null;
174     }
175     public Order getOrder(int index) {
176         if(index < orders.length || index >= 0)return orders[index];
177         else return null;
178     }
179     public void scoreUp(Order order){
180         switch(order.orderName){
181             case "salad" : score += 50;
182             case "tekkamaki" : score += 50;
183             case "kappamaki" : score += 50;
184             case "tunanigiri" : score += 30;
185             case "ikanigiri" : score += 30;
186             case "kaisendon" : score += 60;
187         }
188         System.out.println("scoreUp()が呼ばれました");
189         for(int i=0; i<orders.length; i++){
190             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
191                 removeOrder(i);
192                 return;
193             }
194         }
195     }
196     public void scoreDown(Order order){
197         System.out.println("scoreDown()が呼ばれました");
198         if(score == 0) return;
199         if(order == null){
200             score -= 50;
201             if(score < 0) score = 0;
202             return;
203         }
204         switch(order.orderName){
205             case "salad" : score -= 30;
206             case "tekkamaki" : score -= 30;
207             case "kappamaki" : score -= 30;
208             case "tunanigiri" : score -= 20;
209             case "ikanigiri" : score -= 20;
210             case "kaisendon" : score -= 30;
211         }
212         if(score < 0) score = 0;
213
214         for(int i=0; i<orders.length; i++){
215             if(orders[i].orderName == order.orderName){
216                 removeOrder(i);
217                 return;
218             }
219         }

```

```

219     }
220 }
221 public void removeOrder(int i){
222     System.out.println("get␣=" + i);
223     if (i >= 0 && i < orders.length && orders[i] != null) {
224         orders[i].cancelTimer(); // タイマーの停止
225         System.out.println("注文␣" + orders[i].orderName + "␣を削除します。");
226         orders[i] = null;
227         formatOrder();
228     }
229 }
230 private void formatOrder(){ //を前に詰めていくメソッドorder
231     for(int s = 0; s < orders.length - 1; s++){
232         for(int t = s; t < orders.length - 1; t++){
233             if(orders[t] == null) {
234                 orders[t] = orders[t+1];
235                 if(orders[t] != null) { orders[t].orderIndex = t; }
236                 orders[t+1] = null;
237             }
238         }
239     }
240 }
241
242 // 以下時間に関わるメソッド Yoshida
243 public int getGameTime(){
244     return gameTime;
245 }
246
247 public void decreaseTime(){
248     if(gameTime > 0){
249         gameTime--;
250     }
251 }
252
253 public void reset() {
254     gameTime = 120/*3*60 + 30*/;
255     score = 0;
256     for(int i=0; i<5; i++){
257         orders[i] = null;
258     }
259     for (int i = 0; i < xsize; i++) {
260         for (int j = 0; j < ysize; j++) {
261             grid[i][j].food = null;
262             grid[i][j].plate = null;
263             grid[i][j].isPlatePlaced = false;
264             if (i == 0 || j == 0 || i == xsize - 1 || j == ysize - 1) {
265                 grid[i][j].wall = true;
266             }
267         }
268     }
269     grid[3][3].obstacle = true;
270     grid[4][3].obstacle = true;
271     grid[5][3].obstacle = true;
272     grid[6][3].obstacle = true;
273     grid[9][5].obstacle = true;
274     grid[10][5].obstacle = true;
275     grid[11][5].obstacle = true;
276     grid[12][5].obstacle = true;
277
278     grid[4][5].foodBox = 1;
279     grid[4][5].obstacle = true;
280     grid[4][5].tool = 2;
281
282     grid[5][5].foodBox = 2;
283     grid[5][5].obstacle = true;
284     grid[5][5].tool = 4;
285
286     grid[6][5].foodBox = 3;
287     grid[6][5].obstacle = true;
288     grid[6][5].tool = 5;
289
290     grid[9][3].foodBox = 4;
291     grid[9][3].obstacle = true;
292     grid[9][3].tool = 6;
293
294     grid[10][3].foodBox = 5;
295     grid[10][3].obstacle = true;
296     grid[10][3].tool = 7;
297
298     grid[11][3].foodBox = 6;
299     grid[11][3].obstacle = true;
300     grid[11][3].tool = 8;
301
302     grid[12][3].foodBox = 7;
303     grid[12][3].obstacle = true;
304     grid[12][3].tool = 9;
305
306     grid[7][8].wall = true;
307     grid[7][8].isCounter = true;
308     grid[8][8].wall = true;
309     grid[8][8].isCounter = true;
310
311     grid[0][3].tool = 1; //ナイフ
312     grid[0][4].tool = 1; //ナイフ
313     grid[0][5].tool = 1; //ナイフ
314     grid[15][3].tool = 1; //ナイフ
315     grid[15][4].tool = 1; //ナイフ
316     grid[15][5].tool = 1; //ナイフ
317
318     grid[10][0].tool = 10; //なべ
319     grid[11][0].tool = 10; //なべ
320     grid[12][0].tool = 10; //なべ
321
322     grid[3][0].tool = 12; //フライパン
323     grid[4][0].tool = 12; //フライパン

```



```

324         grid[5][0].tool = 12; //フライパン
325
326         grid[3][5].plateBox = true;
327         grid[3][5].obstacle = true;
328         grid[3][5].tool = 3;
329
330         grid[7][0].plateBox = true;
331         grid[7][0].tool = 3; //皿ボックス
332         grid[8][0].plateBox = true;
333         grid[8][0].tool = 3; //皿ボックス
334
335         grid[0][1].tool=13;
336         grid[0][7].tool=13;
337         grid[15][1].tool=13;
338         grid[15][7].tool=13;
339
340         grid[6][8].tool = 14;
341         grid[9][8].tool = 14;
342     }
343 }

```

● 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     private BufferedImage cacheFloorAll = null;
20     private Timer drawTimer60fps; //60でHspaintcomponent()を呼び出すために使う Kome
21     protected DrawModel model;
22     private DrawController cont;
23     Grid[][] grid;
24     int[] size;
25     final int cellSize;
26
27     private Image ImagePlayer; //画像のための変数
28     private Image imgPlayerUp;
29     private Image imgPlayerLeft;
30     private Image imgPlayerDown;
31     private Image imgPlayerRight;
32     private Image imgErrorBlock;
33     private Image imgKnife;
34     private Image imgBoil;
35     private Image imgBoilRice;
36     private Image imgPlateBox;
37     private Image imgPlate;
38     private Image imgPan;
39     private Image imgCabbageBox;
40     private Image imgCabbage;
41     private Image imgCabbageCut;
42     private Image imgTomatoBox;
43     private Image imgTomato;
44     private Image imgTomatoCut;
45     private Image imgCucumberBox;
46     private Image imgCucumber;
47     private Image imgCucumberCut;
48     private Image imgCabTom;
49     private Image imgCabCuc;
50     private Image imgTomCuc;
51     private Image imgCabTomCuc;
52     private Image imgRiceBox;
53     private Image imgRice;
54     private Image imgRiceBoil;
55     private Image imgTunaBox;
56     private Image imgTuna;
57     private Image imgTunaCut;
58     private Image imgSquidBox;
59     private Image imgSquid;
60     private Image imgSquidCut;
61     private Image imgSeaweedBox;
62     private Image imgSeaweed;
63     private Image imgRicTun;
64     private Image imgRicSqu;
65     private Image imgRicSea;
66     private Image imgRicCuc;
67     private Image imgTunSea;
68     private Image imgTunSqu;
69     private Image imgCucSea;
70     private Image imgRicCucSea;
71     private Image imgRicTunSea;
72     private Image imgRicTunSqu;
73     private Image imgTrash;
74     private Image[] imgCounter = new Image[5];
75     private Image orderPaper;
76     private Image imgKnifeBlack;
77     private Image imgBoilBlack;
78     private Image imgFloor1;
79     private Image imgFloor2;
80     private Image imgFloor3;

```

```

81     private Image imgTable;
82     private Image imgSampleSalad;
83     private Image imgA;
84     private Image imgB;
85     private Image imgC;
86     private Image imgF1;
87     private Image imgF2;
88     private Image imgF3;
89     private Image testWall;
90     private Image sideWall;
91     private Image longShadow;
92     private Image imgWaiterUp;
93     private Image imgWaiterDown;
94     private Image imgFire;
95     private Image imgUIBG;
96     private Image imgCoin;
97     private Image imgTimer;
98     private Image imgCandle;
99
100     Player player;
101     static final int headerBlank = 220;
102     static final int footerBlank = 300;
103     static final int rightBlank = 20;
104     static final int leftBlank = 60;
105     double playerSpeed;
106
107     Waiter[] waiters = new Waiter[5];
108
109     private ScheduledExecutorService executor;
110     private int frameCount = 0; // フレーム数をカウント
111     private double fps = 0.0; // 計算したを格納FPS
112     private long lastTime = System.nanoTime(); // 前回の時間
113     private static final long FPS_UPDATE_INTERVAL = 100_000_000; // 100 (ナノ秒) ms
114     int passedFrame = 0; // 全体の経過フレーム、様々なアニメーションにつかう
115     int flameScoreGet = 0;
116     int getScore = 0;
117
118
119     //public boolean moving = true;
120     private Font customFont;
121     public DrawView(DrawModel m) { //初期化
122         //画像読み込み
123         imgPlayerUp = new ImageIcon("img/test/ghost_up.png").getImage();
124         imgPlayerLeft = new ImageIcon("img/test/ghost_left.png").getImage();
125         imgPlayerDown = new ImageIcon("img/test/ghost_down.png").getImage();
126         imgPlayerRight = new ImageIcon("img/test/ghost_right.png").getImage();
127         imgErrorBlock = new ImageIcon("img/miss.png").getImage();
128
129         //皿とツール
130         imgKnife=new ImageIcon("img/knife.png").getImage();
131         imgBoil=new ImageIcon("img/boil.png").getImage();
132         imgBoilRice=new ImageIcon("img/rice_boil.png").getImage();
133         imgPlateBox = new ImageIcon("img/plate_box.png").getImage();
134         imgPlate = new ImageIcon("img/plate.png").getImage();
135         imgPan = new ImageIcon("img/pan.png").getImage();
136
137         imgCabbageBox=new ImageIcon("img/cabbage_box.png").getImage();
138         imgCabbage=new ImageIcon("img/cabbage.png").getImage();
139         imgCabbageCut = new ImageIcon("img/cabbage_cut.png").getImage();
140
141         imgTomatoBox = new ImageIcon("img/tomato_box.png").getImage();
142         imgTomato = new ImageIcon("img/tomato.png").getImage();
143         imgTomatoCut = new ImageIcon("img/tomato_cut.png").getImage();
144
145         imgCucumberBox = new ImageIcon("img/cucumber_box.png").getImage();
146         imgCucumber = new ImageIcon("img/cucumber.png").getImage();
147         imgCucumberCut = new ImageIcon("img/cucumber_cut.png").getImage();
148
149         imgCabTom = new ImageIcon("img/cab_tom.png").getImage();
150         imgCabCuc = new ImageIcon("img/cab_cuc.png").getImage();
151         imgTomCuc = new ImageIcon("img/tom_cuc.png").getImage();
152         imgCabTomCuc = new ImageIcon("img/cab_tom_cuc.png").getImage();
153
154         imgRiceBox = new ImageIcon("img/rice_box.png").getImage();
155         imgRice = new ImageIcon("img/rice.png").getImage();
156         imgRiceBoil = new ImageIcon("img/rice_boil2.png").getImage();
157
158         imgTunaBox = new ImageIcon("img/tuna_box.png").getImage();
159         imgTuna = new ImageIcon("img/tuna.png").getImage();
160         imgTunaCut = new ImageIcon("img/tuna_cut.png").getImage();
161
162         imgSquidBox = new ImageIcon("img/squid_box.png").getImage();
163         imgSquid = new ImageIcon("img/squid.png").getImage();
164         imgSquidCut = new ImageIcon("img/squid_cut.png").getImage();
165
166         imgSeaweedBox = new ImageIcon("img/seaweed_box.png").getImage();
167         imgSeaweed = new ImageIcon("img/seaweed.png").getImage();
168
169         imgRicTun = new ImageIcon("img/ric_tun.png").getImage();
170         imgRicSqu = new ImageIcon("img/ric_squ.png").getImage();
171         imgRicSea = new ImageIcon("img/ric_sea.png").getImage();
172         imgRicCuc = new ImageIcon("img/ric_cuc.png").getImage();
173         imgTunSea = new ImageIcon("img/tun_sea.png").getImage();
174         imgTunSqu = new ImageIcon("img/tun_squ.png").getImage();
175         imgCucSea = new ImageIcon("img/cuc_sea.png").getImage();
176         imgRicCucSea = new ImageIcon("img/ric_cuc_sea.png").getImage();
177         imgRicTunSea = new ImageIcon("img/ric_tun_sea.png").getImage();
178         imgRicTunSqu = new ImageIcon("img/ric_tun_squ.png").getImage();
179
180
181
182         imgCounter[0] = new ImageIcon("img/test/counter1.png").getImage();
183         imgCounter[1] = new ImageIcon("img/test/counter2.png").getImage();
184         imgCounter[2] = new ImageIcon("img/test/counter3.png").getImage();
185         imgCounter[3] = new ImageIcon("img/test/counter4.png").getImage();

```

```

186         imgCounter[4] = new ImageIcon("img/test/counter5.png").getImage();
187         orderPaper = new ImageIcon("img/order_paper_short.png").getImage();
188         imgKnifeBlack = new ImageIcon("img/knife_black.png").getImage();
189         imgBoilBlack = new ImageIcon("img/boil_black.png").getImage();
190
191         imgTrash = new ImageIcon("img/trash.png").getImage();
192
193         imgFloor1 = new ImageIcon("img/floor1.jpg").getImage();
194         imgFloor2 = new ImageIcon("img/floor2.jpg").getImage();
195         imgFloor3 = new ImageIcon("img/floor3.png").getImage();
196         imgA = new ImageIcon("img/test/B.png").getImage();
197         imgB = new ImageIcon("img/test/D_long.png").getImage();
198         imgC = new ImageIcon("img/test/C.jpg").getImage();
199         imgF1 = new ImageIcon("img/test/floor_a_4.png").getImage();
200         imgF2 = new ImageIcon("img/test/floor_b_4.png").getImage();
201         imgF3 = new ImageIcon("img/test/floor_c_3.png").getImage();
202
203         imgTable = new ImageIcon("img/table.png").getImage();
204
205         imgSampleSalad = new ImageIcon("img/cab_tom_cuc.png").getImage();
206
207         imgFire = new ImageIcon("img/fires.png").getImage();
208
209
210         imgUIBG = new ImageIcon("img/ui_background.png").getImage();
211         imgCoin = new ImageIcon("img/coin.png").getImage();
212         imgTimer = new ImageIcon("img/timer.png").getImage();
213
214         testWall = new ImageIcon("img/test/wallpaper_11.png").getImage();
215         sideWall = new ImageIcon("img/test/wall_side.png").getImage();
216         imgWaiterUp = new ImageIcon("img/test/ghost_up.png").getImage();
217         imgWaiterDown = new ImageIcon("img/test/ghost_down.png").getImage();
218         longShadow = new ImageIcon("img/long_shadow.png").getImage();
219
220         imgCandle = new ImageIcon("img/test/candle.png").getImage();
221     }
222     model = m;
223     this.setFocusable(true);
224     this.setDoubleBuffered(true);
225     player = model.getPlayer();
226     grid = model.getGrid();
227     size = model.getFieldSize();
228     cellSize = model.getCellSize();
229     loadCustomFont();
230
231     executor = Executors.newScheduledThreadPool(1); //60での描画を開始fps
232     executor.scheduleAtFixedRate(() -> {
233         long currentTime = System.nanoTime();
234         frameCount++;
235
236         // 100ms ごとに FPS を計算
237         if (frameCount >= 30) {
238             double timeDiff = (currentTime - lastTime) / 1_000_000.0;
239             double fps = 1000.0 * 30 / timeDiff;
240             frameCount = 0; // フレーム数をリセット
241             lastTime = currentTime; // 時間を更新
242         }
243
244         SwingUtilities.invokeLater(this::repaint); // スレッドで描画Swing
245     }, 0, 16, TimeUnit.MILLISECONDS);
246
247     playerSpeed = player.getPlayerSpeed();
248
249     createCacheFloorAll();
250
251 }
252 public void setController(DrawController cont) { this.cont = cont; }
253 //床の画像をキャッシュする関数. のコンストラクタで一回だけ呼ぶDrawView
254 private void createCacheFloorAll() {
255     int cS = cellSize;
256     int overCell = 6;
257     cacheFloorAll = new BufferedImage(cS*size[0], cS * (size[1]+overCell), BufferedImage.TYPE_INT_ARGB);
258     Graphics2D g2 = cacheFloorAll.createGraphics();
259
260     // 必要に応じて他の背景パーツを描画する
261     int rB = rightBlank;
262     int hB = headerBlank;
263     for(int i = 1; i < size[0] -1; i++){
264         for(int j = 1; j < size[1] -1; j++){
265             g2.setColor(Color.DARK_GRAY);
266             if((i + j)%2 == 0){g2.drawImage(imgF1, i * cS, j * cS, cS, cS, this);}
267             else {g2.drawImage(imgF2, i * cS, j * cS, cS, cS, this);}
268         }
269     }
270     for(int j = size[1]; j < size[1] + overCell; j++){
271         for(int i = 0; i < size[0]; i++){
272             g2.setColor(new Color(200,0,0));
273             g2.drawImage(imgF3, i * cS, j * cS, cS, cS, this);
274         }
275     }
276     g2.dispose();
277 }
278
279 protected void paintComponent(Graphics g) {
280     super.paintComponent(g);
281     passedFlame++;
282     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
283     g.drawImage(testWall, rightBlank, 0, cellSize*16, headerBlank, this); //奥の壁
284     g.setColor(new Color(101,68,59));
285     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
286     g.fillRect(0, 0, rightBlank, 1200);
287     g.fillRect(0 + rightBlank + size[0]*cellSize, 0, rightBlank, 1200);
288     final int rB = rightBlank;
289     final int hB = headerBlank;
290     final int cS = cellSize;

```

```

291
292 //プレイヤーの座標のアニメーション処理
293 if(Math.abs(player.x - player.xAnim) <= playerSpeed){ //についてx
294     player.xAnim = player.x;
295     player.moving = false;
296 }else if(player.x > player.xAnim){
297     player.xAnim += playerSpeed;
298     player.moving = true;
299 }else if(player.x < player.xAnim){
300     player.xAnim -= playerSpeed;
301     player.moving = true;
302 }
303 if(Math.abs(player.y - player.yAnim) <= playerSpeed){ //についてy
304     player.yAnim = player.y;
305     player.moving = (player.moving || false);
306 }else if(player.y > player.yAnim){
307     player.yAnim += playerSpeed;
308     player.moving = true;
309 }else if(player.y < player.yAnim){
310     player.yAnim -= playerSpeed;
311     player.moving = true;
312 }
313 //プレイヤーの下の方の描画
314 g.setColor(Color.BLACK);
315 g.setColor(new Color(0,0,0,128));
316 g.fillRect((int)(player.xAnim*cellSize) + rB + 10, (int)(player.yAnim*cellSize) + hB + dD3d + 10, 40, 40);
317
318 //テーブルの描画
319 for (int j = 0; j < size[1]; j++) {
320     for (int i = 0; i < size[0]; i++) {
321         if (grid[i][j].wall) {
322             if ((i == 0 || i == size[0] - 1) && j != size[1] - 1 && j != 0) { // 右と左のテーブル
323                 g.drawImage(imgA, i * cellSize + rB, j * cellSize + hB, cellSize, cellSize, this);
324             } else {
325                 g.drawImage(imgB, i * cellSize + rB, j * cellSize + hB, cellSize, cellSize + dD3d + 14, this);
326             }
327         } else if (grid[i][j].obstacle) {
328             g.drawImage(imgB, i * cellSize + rB, j * cellSize + hB, cellSize, cellSize + dD3d + 14, this);
329         }
330
331         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
332             if (grid[i][j].wall == false && grid[i][j].obstacle == false){
333                 g.drawImage(imgPlate, i * cellSize + rB, j * cellSize + hB + dD3d, cellSize, cellSize, this);
334             }else{//土台の上なら疑似座標ズレを考慮
335                 g.drawImage(imgPlate, i * cellSize + rB, j * cellSize + hB, cellSize, cellSize, this);
336             }
337         }
338     }
339     //食材画像を描画
340     Image selectedImage = null;
341     if (grid[i][j].plate == null && grid[i][j].food != null){ //そのマスはもっていないがplate かつそのマスにはよくざいがあるとき
342         //つまり皿の描画は全くの描画の場合です。Food
343         selectedImage = setFoodImage(grid[i][j].food);
344     }else if (grid[i][j].plate != null && grid[i][j].plate.hasAnyFood() == true){ //皿があって食材がいてある場合
345         selectedImage = setPlateImage(grid[i][j].plate);
346     }
347     if (selectedImage != null) {
348         int length = (int)(cellSize*0.7); //描画画像の一边の長さ
349         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
350         if (grid[i][j].wall == false && grid[i][j].obstacle == false){ //台上じゃなかったら
351             g.drawImage(selectedImage, i * cS + rB + cenOffSet, j * cS + hB + dD3d + cenOffSet, length, length, this);
352         }else{ //台上だったら
353             g.drawImage(selectedImage, i * cS + rB + cenOffSet, j * cS + hB + cenOffSet, length, length, this);
354         }
355     }
356 }
357 }
358 //影を落とす
359 g.drawImage(longShadow, 0+rightBlank, 0+headerBlank, 960, 14, this);
360
361 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
362
363 //すべての座標について重文for
364 for (int i = size[0]-1; i >= 0; i--){
365     for (int j = size[1]-1; j >= 0; j--){
366         Image selectedImage = null;
367         //ツールマスに関しての描画
368         if (grid[i][j].tool != 0){
369             selectedImage = setToolImage(grid[i][j].tool);
370             if (grid[i][j].foodBox != 0)
371                 g.drawImage(imgB, i * cellSize + rB, j * cellSize + hB, cellSize, cellSize, this);
372         }
373         if (selectedImage != null) {
374             if (grid[i][j].wall == false && grid[i][j].obstacle == false){ //台上じゃなかったら
375                 g.drawImage(selectedImage, i * cS + rB, j * cS + hB + dD3d, cellSize, cellSize, this);
376             }else{ //台上だったら
377                 g.drawImage(selectedImage, i * cS + rB, j * cS + hB, cellSize, cellSize, this);
378             }
379         }
380     }
381 }
382
383 for (int i = size[0]-1; i >= 0; i--){
384     for (int j = size[1]-1; j >= 0; j--){
385         if (grid[i][j].isPlatePlaced && grid[i][j].plate.hasAnyFood()){
386             setIngredientsImage(cellSize, grid[i][j].x*cS, grid[i][j].y*cS, 0, 0, grid[i][j].plate, g, 0);
387         }
388     }
389 }
390
391 // 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
392 switch(player.direction){
393     case 1: ImagePlayer = imgPlayerUp; break;
394     case 2: ImagePlayer = imgPlayerLeft; break;

```

```

395         case 3: ImagePlayer = imgPlayerDown; break;
396         case 4: ImagePlayer = imgPlayerRight; break;
397     }
398     g.drawImage(ImagePlayer, (int)(player.xAnim*cellSize)-10 + rB, (int)(player.yAnim*cellSize) + hB -10, 80, 80, this);
399
400     if(player.hasPlate == true){ //プレイヤーが皿を持っていたら
401         //皿と画像の比率を調整
402         int foodSize = (int)(0.68*cellSize);
403         int offsetX = (cellSize - foodSize)/2;
404         int offsetY = (cellSize - foodSize)/2;
405         if(player.direction == 1) offsetY -= (int)(.92*cellSize);
406         else if(player.direction == 2) offsetX -= (int)(0.8*cellSize);
407         else if(player.direction == 3) offsetY += (int)(0.72*cellSize);
408         else if(player.direction == 4) offsetX += (int)(0.8*cellSize);
409         g.drawImage(imgPlate, (int)(player.xAnim*cS) + offsetX +rB + 1, (int)(player.yAnim*cS)+ offsetY + 4 + hB, foodSize,
410             foodSize, this); // は微調整項
411             +I, +4
412     }
413     Image heldFoodImage = null;
414     if(player.hasPlate == true && player.plate.hasAnyFood() == true){ //食材ありの皿を持てたら
415         heldFoodImage = setPlateImage(player.plate);
416     }else if(player.getFood() != null){ //単体の食材を持っていたら
417         heldFoodImage = setFoodImage(player.getFood());
418     }
419     if (heldFoodImage != null) {
420         // 少し小さめにしてプレイヤーの上に描画
421         int foodSize = (int)(0.55*cellSize);
422         int offsetX = (cellSize - foodSize)/2;
423         int offsetY = (cellSize - foodSize)/2;
424         if(player.direction == 1) offsetY -= (int)(.92*cellSize); //上のブロックのパラメータと共通
425         else if(player.direction == 2) offsetX -= (int)(0.8*cellSize);
426         else if(player.direction == 3) offsetY += (int)(0.72*cellSize);
427         else if(player.direction == 4) offsetX += (int)(0.8*cellSize);
428         g.drawImage(heldFoodImage, (int)(player.xAnim*cS) + offsetX +rB + 2, (int)(player.yAnim*cS) + offsetY + hB, foodSize,
429             foodSize, this); //は微調整項
430             +I
431     }
432     if(player.hasPlate == true && player.plate.hasAnyFood()){
433         int offsetX = cellSize / 4;
434         int offsetY = cellSize / 4;
435         if(player.direction == 1) {offsetX = 0; offsetY -= cellSize *2/ 3;}
436         else if(player.direction == 2) {offsetX -= cellSize *2/ 3; offsetY = 0;}
437         else if(player.direction == 3) {offsetX = 0; offsetY += cellSize ;}
438         else if(player.direction == 4) {offsetX += cellSize / 3; offsetY = 0;}
439         setIngredientsImage(cellSize, (int)(player.xAnim*cS), (int)(player.yAnim*cS), offsetX, offsetY, player.plate, g, player.
440             direction);
441     }
442
443     //装飾品の描画
444     g.drawImage(imgCandle, 6*cellSize + rightBlank, 8 * cellSize + headerBlank - 60, 60, 120, this);
445     g.drawImage(imgCandle, 9*cellSize + rightBlank, 8 * cellSize + headerBlank - 60, 60, 120, this);
446
447     //の描画UI
448     g.drawImage(imgUIBG, 55, 750, 250, 90, this); //得点表示の背景
449     g.drawImage(imgCoin, 0, 730, 120, 120, this); //得点表示の背景
450
451     g.drawImage(imgUIBG, 655, 750, 250, 90, this); //時間表示の背景
452     g.drawImage(imgTimer, 868, 730, 120, 120, this); //時間表示の背景
453     Graphics2D g2d = (Graphics2D) g;
454     g2d.setFont(customFont);
455     g2d.setColor(Color.WHITE);
456     int leftTimeAllSec = model.getGameTime();
457     int leftTimeMin = leftTimeAllSec/60;
458     int leftTimeSec = leftTimeAllSec%60;
459     g2d.drawString(String.format("%d:%02d", leftTimeMin, leftTimeSec), 712, 820);
460
461     double dScore = model.score - scoreAnim;
462     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
463     scoreAnim += dScore * easingFactorText;
464     if (Math.abs(dScore) < 2.0) { scoreAnim = model.score; }
465
466     String text = Integer.toString((int)scoreAnim);
467     FontMetrics fm = g2d.getFontMetrics();
468     int textWidth = fm.stringWidth(text);
469     int centerX = 202; // 中央に配置したい座標x
470     g2d.drawString(text, centerX - textWidth / 2, 820);
471
472     if(1 <= flameScoreGet && flameScoreGet <= 60){
473         text = Integer.toString(getScore);
474         if(getScore >= 0){
475             g.setColor(new Color(50, 255, 50, 200 - 2*flameScoreGet));
476             text = "+" + text;
477         } else {
478             g.setColor(new Color(255, 50, 50, 200 - 2*flameScoreGet));
479         }
480         fm = g2d.getFontMetrics();
481         textWidth = fm.stringWidth(text);
482         centerX = 175; // 中央に配置したい座標x
483         g2d.drawString(text, centerX - textWidth / 2, 770 - 2*flameScoreGet/3);
484         flameScoreGet++;
485     }else if(flameScoreGet > 60){ flameScoreGet = 0; }
486
487     //オーダー用紙の描画
488     for(int i = 0; i < model.orders.length; i++){
489         Image orderImage;
490         int orderW = 160;
491         int orderH = 100;
492         if(model.orders[i] != null){
493             Order order = model.orders[i];
494             orderImage = setOrderImage(order);
495             int targetPos = 20 + i * (orderW +5);
496             double dx = targetPos - order.posAnim;
497             order.posAnim += dx * easingFactor;
498         }
499     }

```

```

495         if (Math.abs(dx) < 1.0) {
496             order.posAnim = targetPos;
497             if (order.timeAnim == 0) {
498                 order.timeAnim = 1;
499             }
500         }
501         if (1 <= order.timeAnim) {
502             if (30 <= order.timeAnim) {
503                 dx = order.subOrderPosY - order.subOrderPosYAnim;
504                 order.subOrderPosYAnim += easingFactor * dx;
505                 if (Math.abs(dx) < 1.0) {
506                     order.subOrderPosYAnim = order.subOrderPosY;
507                 }
508                 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
509                 int interval = cellSize-11;
510                 int wid = 45;
511                 if (order.ingredient1 != null) {
512                     g.setColor(new Color(174, 207, 227));
513                     g.fillRect((int)order.posAnim+7+interval*0, sOPYA, wid, 90);
514                     g.drawImage(setCorrectRaw(order.ingredient1), (int)order.posAnim+interval*0 + 8, sOPYA+10, 42,42,this);
515                     if (setCorrectMethod(order.ingredient1)!=null){
516                         g.drawImage(setCorrectMethod(order.ingredient1), (int)order.posAnim+interval*0 + 9, sOPYA+50, 42,42,
517                             this);
518                     }
519                 }
520                 if (order.ingredient2 != null) {
521                     g.setColor(new Color(174, 207, 227));
522                     g.fillRect((int)order.posAnim+7+interval*1, sOPYA, wid, 90);
523                     g.drawImage(setCorrectRaw(order.ingredient2), (int)order.posAnim+interval*1 + 8, sOPYA+10, 42,42,this);
524                     if (setCorrectMethod(order.ingredient2)!=null){
525                         g.drawImage(setCorrectMethod(order.ingredient2), (int)order.posAnim+interval*1 + 9, sOPYA+50, 42,42,
526                             this);
527                     }
528                 }
529                 if (order.ingredient3 != null) {
530                     g.setColor(new Color(174, 207, 227));
531                     g.fillRect((int)order.posAnim+7+interval*2, sOPYA, wid, 90);
532                     g.drawImage(setCorrectRaw(order.ingredient3), (int)order.posAnim+interval*2 + 8, sOPYA+10, 42,42,this);
533                     if (setCorrectMethod(order.ingredient3)!=null){
534                         g.drawImage(setCorrectMethod(order.ingredient3), (int)order.posAnim+interval*2 + 9, sOPYA+50, 42,42,
535                             this);
536                     }
537                 }
538             }
539             order.timeAnim++;
540         }
541         g.drawImage(orderPaper, (int)order.posAnim, 15, orderW, orderH, this);
542         drawGauge(g, "down", (int)(order.posAnim)+8, 22, orderW-16, 17, order.getRemainingTime()/order.timeLimit);
543         g.drawImage(orderImage, 42 + (int)order.posAnim, 30, 75, 75, this);
544     }
545 }
546
547 if (cont.spacePushing == true) {
548     if (player.getFrontGrid().tool == 12) {player.actionCharge += 0.5;} //フライパンの時は長め
549     else player.actionCharge += 1;
550 }
551 else { player.actionCharge = 0; }
552 if (0 < player.actionCharge && player.actionCharge < 60) {
553     drawGauge(g, "up", (int)(player.xAnim*cellSize)+rightBlank + 10, (int)(player.yAnim*cellSize)+headerBlank,(int)(0.7*
554         cellSize),8,player.actionCharge/60.0);
555 } else if (player.actionCharge == 60) player.action();
556
557 //フライパンの火の描画です
558 if (player.food != null && player.food.canHeat) {
559     if (player.getFrontGrid().tool == 12 && cont.spacePushing == true) {
560         if (player.actionCharge > 0 && player.actionCharge < 60) {
561             float fireScall = player.actionCharge % 30;
562             g.drawImage(imgFire, player.getFrontGrid().x * cellSize + 30 - (int)(fireScall/2), player.getFrontGrid().y *
563                 cellSize + headerBlank + 55 - (int)(fireScall/2), (int)(fireScall*cellSize/60), (int)(fireScall*cellSize/60),
564                 this);
565         }
566     }
567 }
568
569 //米炊く Yoshida
570 for (int i = 0; i < size[0]; i++) {
571     for (int j = 0; j < size[1]; j++) {
572         if (grid[i][j].tool == 10 && grid[i][j].hasFood()) {
573             if (grid[i][j].cookingGauge < 60.0) grid[i][j].cookingGauge += 0.1;
574             if (grid[i][j].cookingGauge > 0 && grid[i][j].cookingGauge < 60) {
575                 drawGauge(g, "up", i*cS+7 + rightBlank, j*cS+headerBlank-10, (int)(0.7*cS), 8, grid[i][j].cookingGauge/60.0);
576             }
577             else if (grid[i][j].cookingGauge >= 60.0) {
578                 if (grid[i][j].food.foodName == "rice") {
579                     g.drawImage(setToolImage(11), i * cS + rightBlank, j * cS + headerBlank, cS, cS, this);
580                 }
581             }
582         }
583     }
584 }
585
586 for (int i = 0; i < 5; i++) {
587     if (waiters[i] != null && waiters[i].active == true) {
588         waiters[i].drawWe(g, this);
589     }
590 }
591
592 if (passedFlame == 60) AudioManager.playBGM("./sound/music_background2.wav");
593

```

```

594     }
595     private void drawFloorAll(Graphics g, ImageObserver io){//床
596         int cS = cellSize; //この中で略語を定義
597         int rB = rightBlank;
598         int hB = headerBlank;
599         for(int i = 0; i < size[0]; i++){
600             for(int j = 0; j < size[1]; j++){
601                 g.setColor(Color.DARK_GRAY);
602                 if((i + j)%2 == 0){g.drawImage(imgF1, i * cS + rB, j * cS + hB, cS, cS, this);}
603                 else {g.drawImage(imgF2, i * cS + rB, j * cS + hB, cS, cS, this);}
604             }
605         }
606     }
607     private void drawGauge(Graphics g, String type, int x, int y, int width, int height, double ratio){//時間ゲージ・料理中ゲージ
608         if(ratio > 1) { System.out.println("Warning: 1.ゲージの割合がを超えています100%"); }
609
610         if(type == "up"){
611             g.setColor(Color.WHITE);
612             g.fillRect(x-2, y-2, width+4, height+4);
613             g.setColor(new Color(75, 180, 35));
614             g.fillRect(x, y, (int)(width*ratio), height);
615         }
616         else if(type == "down"){
617             g.setColor(Color.GRAY);
618             g.fillRect(x, y, width, height);
619             if(ratio >= 0.5) { g.setColor(new Color(75, 180, 35)); }
620             else if(ratio >= 0.25) { g.setColor(Color.YELLOW); }
621             else { g.setColor(Color.RED); }
622             g.fillRect(x, y, (int)(width*ratio), height);
623         }
624     }
625
626     private Image setToolImage(int toolId){//ツールを引数としてその画像を返すID
627         switch(toolId){
628             case 1: return imgKnife;           //ナイフ
629             case 2: return imgCabbageBox;      //キャベツボックス
630             case 3: return imgPlateBox;        //皿ボックス
631             case 4: return imgTomatoBox;       //トマトボックス
632             case 5: return imgCucumberBox;     //キュウリボックス
633             case 6: return imgRiceBox;         //米ボックス
634             case 7: return imgTunaBox;         //マグロボックス
635             case 8: return imgSquidBox;        //イカボックス
636             case 9: return imgSeaweedBox;      //海苔ボックス
637             case 10: return imgBoil;          //鍋
638             case 11: return imgBoilRice;      //炊けた米
639             case 12: return imgPan;           //フライパン
640             case 13: return imgTrash;         //ごみ箱
641             case 14: return null;
642         }
643         return imgErrorBlock; //以外ならエラー!4
644     }
645     private Image setCorrectRaw(Food foodInfo){//食材情報を受け取って加工前食材の画像を返す
646         if(foodInfo.foodName == "cabbage") return imgCabbage;
647         else if(foodInfo.foodName == "tomato") return imgTomato;
648         else if(foodInfo.foodName == "cucumber") return imgCucumber;
649         else if(foodInfo.foodName == "rice") return imgRice;
650         else if(foodInfo.foodName == "tuna") return imgTuna;
651         else if(foodInfo.foodName == "squid") return imgSquid;
652         else if(foodInfo.foodName == "seaweed") return imgSeaweed;
653
654         else return imgErrorBlock;
655     }
656     private Image setCorrectMethod(Food foodInfo){//オーダー用。食材の調理方法を受け取って調理法画像返す
657         if(foodInfo.foodStatus == 2) return imgKnifeBlack;
658         else if(foodInfo.foodStatus == 3) return imgBoilBlack;
659         else return null;
660     }
661     private Image setFoodImage(Food foodInfo){//食材情報を受け取って、状態に応じた画像を返す。未加工カットゆで1:,2:,3:
662         // 文にしてみてもいいかもねswitch
663         if(foodInfo.foodName == "cabbage"){
664             if(foodInfo.foodStatus == 1) return imgCabbage;
665             else if(foodInfo.foodStatus == 2) return imgCabbageCut;
666             else return imgErrorBlock;
667         }else if(foodInfo.foodName == "tomato"){
668             if(foodInfo.foodStatus == 1) return imgTomato;
669             else if(foodInfo.foodStatus == 2) return imgTomatoCut;
670             else return imgErrorBlock;
671         }else if(foodInfo.foodName == "cucumber"){
672             if(foodInfo.foodStatus == 1) return imgCucumber;
673             else if(foodInfo.foodStatus == 2) return imgCucumberCut;
674             else return imgErrorBlock;
675         }else if(foodInfo.foodName == "rice"){
676             if(foodInfo.foodStatus == 1) return imgRice;
677             else if(foodInfo.foodStatus == 3) return imgRiceBoil; //はboil?heita
678             else return imgErrorBlock;
679         }else if(foodInfo.foodName == "tuna"){
680             if(foodInfo.foodStatus == 1) return imgTuna;
681             else if(foodInfo.foodStatus == 2) return imgTunaCut;
682             else return imgErrorBlock;
683         }else if(foodInfo.foodName == "squid"){
684             if(foodInfo.foodStatus == 1) return imgSquid;
685             else if(foodInfo.foodStatus == 2) return imgSquidCut;
686             else return imgErrorBlock;
687         }else if(foodInfo.foodName == "cucumber"){
688             if(foodInfo.foodStatus == 1) return imgCucumber;
689             else if(foodInfo.foodStatus == 2) return imgCucumberCut;
690             else return imgErrorBlock;
691         }else if(foodInfo.foodName == "seaweed"){
692             if(foodInfo.foodStatus == 1) return imgSeaweed;
693             else return imgErrorBlock;
694         }
695         return imgErrorBlock;
696     }
697     public Image setPlateImage(Plate targetPlate){//乗っている食材の画像を返す
698         Food food[] = new Food[3];

```

```

699     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
700     int tomato = 0; //存在しない0: 生1: カット、ボイル2:3:
701     int cucumber = 0;
702     int rice = 0;
703     int tuna = 0;
704     int squid = 0;
705     int seaweed = 0;
706
707
708     //に乗っている具材情報を取得plate
709     for(int i = 0; i < 3; i++){
710         food[i] = targetPlate.get(i);
711         if(food[i] == null){ break; }//これ以上の食材はないのでbreak
712         if(food[i].foodName == "cabbage") cabbage = food[i].foodStatus;
713         else if(food[i].foodName == "tomato") tomato = food[i].foodStatus;
714         else if(food[i].foodName == "cucumber") cucumber = food[i].foodStatus;
715         else if(food[i].foodName == "rice") rice = food[i].foodStatus;
716         else if(food[i].foodName == "tuna") tuna = food[i].foodStatus;
717         else if(food[i].foodName == "squid") squid = food[i].foodStatus;
718         else if(food[i].foodName == "seaweed") seaweed = food[i].foodStatus;
719     }
720
721     //取得した具材情報を利用してにセットする画像を返す。Imageの未所持未処理カットボイル:,1:,2:,3:,
722
723     if(rice==0 && tuna==0 && squid==0 && seaweed==0){
724         //System.out.printf("rice = %d", rice)デバッグ用;//
725         if(cabbage==1 && tomato==0 && cucumber == 0) return imgCabbage; //未加工キャベツ
726         else if(cabbage==0 && tomato==1 && cucumber == 0) return imgTomato; //未加工トマト
727         else if(cabbage==0 && tomato==0 && cucumber == 1) return imgCucumber; //未加工きゅうり
728         else if(cabbage==2 && tomato==0 && cucumber == 0) return imgCabbageCut; //カットキャベツ
729         else if(cabbage==0 && tomato==2 && cucumber == 0) return imgTomatoCut; //カットトマト
730         else if(cabbage==0 && tomato==0 && cucumber == 2) return imgCucumberCut; //カットキュウリ
731         else if(cabbage == 2 && tomato == 2 && cucumber == 0) return imgCabTom; //キャベツトマト
732         else if(cabbage == 2 && tomato == 0 && cucumber == 2) return imgCabCuc; //キャベツキュウリ
733         else if(cabbage == 0 && tomato == 2 && cucumber == 2) return imgTomCuc; //トマトキュウリ
734         else if(cabbage == 2 && tomato == 2 && cucumber == 2) return imgCabTomCuc; //キャベツトマトキュウリ
735     }
736     else if(cabbage==0 && tomato==0 && cucumber==0 && squid==0){
737         //System.out.printまぐろ(" ")デバッグ用;//
738         if(rice == 1 && tuna == 0 && seaweed== 0) return imgRice; //加工前
739         else if(rice == 0 && tuna == 1 && seaweed== 0) return imgTuna; //
740         else if(rice == 0 && tuna == 0 && seaweed== 1) return imgSeaweed; //
741         else if(rice == 3 && tuna == 0 && seaweed== 0) return imgRiceBoil; //加工後
742         else if(rice == 0 && tuna == 2 && seaweed== 0) return imgTunaCut; //
743         else if(rice == 3 && tuna == 2 && seaweed== 0) return imgRicTun; //まぐろにぎり
744         else if(rice == 3 && tuna == 0 && seaweed== 1) return imgRicSea; //
745         else if(rice == 0 && tuna == 2 && seaweed== 1) return imgTunSea; //
746         else if(rice == 3 && tuna == 2 && seaweed== 1) return imgRicTunSea; //鉄火巻
747     }
748     else if(cabbage==0 && tomato==0 && cucumber==0 && tuna==0 && seaweed==0){
749         //System.out.printいか(" ")デバッグ用;//
750         if(rice == 1 && squid == 0) return imgRice; //加工前
751         else if(rice == 0 && squid == 1) return imgSquid; //
752         else if(rice == 3 && squid == 0) return imgRiceBoil; //加工後
753         else if(rice == 0 && squid == 2) return imgSquidCut; //
754         else if(rice == 3 && squid == 2) return imgRicSqu; //いかにぎり
755     }
756     else if(cabbage==0 && tomato==0 && cucumber==0 && seaweed==0){
757         //System.out.print海鮮丼(" ")デバッグ用;//
758         if(rice == 1 && tuna == 0 && squid== 0) return imgRice; //加工前
759         else if(rice == 0 && tuna == 1 && squid== 0) return imgTuna; //
760         else if(rice == 0 && tuna == 0 && squid== 1) return imgSquid; //
761         else if(rice == 3 && tuna == 0 && squid== 0) return imgRiceBoil; //加工後
762         else if(rice == 0 && tuna == 2 && squid== 0) return imgTunaCut; //
763         else if(rice == 0 && tuna == 0 && squid== 2) return imgSquidCut; //
764         else if(rice == 3 && tuna == 2 && squid== 0) return imgRicTun; //まぐろにぎり
765         else if(rice == 3 && tuna == 0 && squid== 2) return imgRicSqu; //いかにぎり
766         else if(rice == 0 && tuna == 2 && squid== 2) return imgTunSqu; //
767         else if(rice == 3 && tuna == 2 && squid== 2) return imgRicTunSqu; //海鮮丼
768     }
769     else if(cabbage==0 && tomato==0 && tuna==0 && squid==0){
770         //System.out.printかつは巻き(" ")デバッグ用;//
771         if(rice == 1 && cucumber == 0 && seaweed== 0) return imgRice; //加工前
772         else if(rice == 0 && cucumber == 1 && seaweed== 0) return imgCucumber; //
773         else if(rice == 0 && cucumber == 0 && seaweed== 1) return imgSeaweed; //
774         else if(rice == 3 && cucumber == 0 && seaweed== 0) return imgRiceBoil; //加工後
775         else if(rice == 0 && cucumber == 2 && seaweed== 0) return imgCucumberCut; //
776         else if(rice == 3 && cucumber == 2 && seaweed== 0) return imgRicCuc; //
777         else if(rice == 3 && cucumber == 0 && seaweed== 1) return imgRicSea; //
778         else if(rice == 0 && cucumber == 2 && seaweed== 1) return imgCucSea; //
779         else if(rice == 3 && cucumber == 2 && seaweed== 1) return imgRicCucSea; //かつば巻
780     }
781
782     return imgErrorBlock; //どれにも当てはまらないときエラー
783 }
784
785 public Image setOrderImage(Order order){//オーダーを受け取ってそれぞれの完成品の画像を返す
786     if("salad".equals(order.orderName)){
787         return imgCabTomCuc;
788     }else if("tekamaki".equals(order.orderName)){
789         return imgRicTunSea;
790     }else if("kappamaki".equals(order.orderName)){
791         return imgRicCucSea;
792     }else if("tunanigiri".equals(order.orderName)){
793         return imgRicTun;
794     }else if("ikanigiri".equals(order.orderName)){
795         return imgRicSqu;
796     }else if("kaisendon".equals(order.orderName)){
797         return imgRicTunSqu;
798     }
799     else return null;
800 }
801
802 // 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
803 // 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。paintComponent

```



```

804 private void setIngredientsImage(int cellSize, int xAnim, int yAnim, int offsetX, int offsetY, Plate plate, Graphics g, int
    playerDirection){
805     Image ingredients[] = new Image[3];
806     int holdStatus[] = new int[3];
807     Food ing[] = new Food[3];
808     int size = cellSize/3;
809     int ingOffsetX = 20;
810     int ingOffsetY = 20;
811     final int hB = headerBlank;
812     final int rB = rightBlank;
813     if(playerDirection == 3){ingOffsetY = 0;}
814     for(int i=0; i<3; i++){
815         if(plate.foods[i] != null){
816             ing[i] = plate.foods[i];
817             holdStatus[i] = plate.foods[i].foodStatus;
818             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、歩あると画像が生になってしまうのでコメントアウトします。(1)
819         }
820     }
821
822     for(int i=0; i<3; i++){
823         if(ing[i] != null){
824             ingredients[i] = setFoodImage(ing[i]);
825             g.setColor(Color.WHITE);
826             g.fillOval(xAnim+ingOffsetX*i+offsetX-3+rB, yAnim+hB+offsetY-ingOffsetY-2, size+5, size+5);
827             g.drawImage(ingredients[i], xAnim+ingOffsetX*i+offsetX+rB, yAnim+hB+offsetY-ingOffsetY, size, size, this);
828             ing[i].foodStatus = holdStatus[i];
829         }
830     }
831 }
832
833 //時間に関するメソッド Yoshida
834 public void updateTime(int time){
835     //System.out.print(time秒+"); 仮のタイマー表示//
836 }
837
838 // 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いますController) Yoshida
839 public JFrame getFrame() {
840     return (JFrame) SwingUtilities.getWindowAncestor(this);
841 }
842 private void loadCustomFont() {
843     try {
844         //File fontFile = new File("font/CHEESE10.TTF"); // フォントファイルのパス
845         File fontFile = new File("font/ByteBounce.ttf"); // フォントファイルのパス
846         customFont = Font.createFont(Font.TRUETYPE_FONT, fontFile).deriveFont(90f); // フォントサイズ24
847     } catch (IOException | FontFormatException e) {
848         System.err.println("フォントの読み込みに失敗:␣" + e.getMessage());
849         customFont = new Font("Arial", Font.BOLD, 24); // 失敗時はデフォルトのフォント
850     }
851 }
852 public void addWaiter(Image mealImage){
853     for(int i = 0; i < 5; i++){
854         if(waiters[i] == null || waiters[i].active == false){
855             System.out.println("Waiter␣Instatance␣made.");
856             waiters[i] = new Waiter(model, mealImage,imgWaiterDown, imgWaiterUp, headerBlank, rightBlank, player.x);
857             return;
858         }
859     }
860 }
861 }

```

● 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(); //ここを取得しておくplayer
19        mainApp = app;
20    }
21
22    @Override
23    public void keyPressed(KeyEvent e) { //それぞれのキーボードの入力を個々で受け取る
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) {}
93
94 // 以下ゲーム時間に関わるメソッド Yoshida
95 public void startGame(){
96     //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都合上、このメソッド内でオーダータイマーとゲームタイマーを管理 Yoshida
97     model.generateOrder();
98     view.repaint();
99
100     //こんな文法あるんだね。知らなかった 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     gameTimer.start(); // タイマー開始
139
140 }
141 private void printCredit(){ //個々はクレジットをコンソールに表示する。
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         "<TeamMembers>\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         "<SpecialThanks>\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){
24         //コンストラクタでは完成形の値を設定
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;
31         //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()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 Seaveed();
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 Kame
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 && orderIngredients[j] != null) {
154                     if (plate.foods[i].getClass() == orderIngredients[j].getClass() &&
155                         plate.foods[i].foodStatus == orderIngredients[j].foodStatus) {
156                         matchedIngredients[j] = true;
157                         break;
158                     }
159                 }
160             }
161         }
162
163         for (boolean matched : matchedIngredients) {
164             if (matched == false){
165                 return false;
166             }
167         }
168         return true;
169     }
170
171     // 残り時間を計算
172     public double getRemainingTime(){
173         long elapsedTimeMill = (System.currentTimeMillis() - createTime);
174         double elapsedTime = elapsedTimeMill / 1000.0;
175         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(),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
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 && getFrontGrid().hasFood() == false){ //プレイヤー移動中は移動したくない
43            int newX = x + dx;
44            int newY = y + dy;
45            //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()
46            if (newX >= 0 && newX < grid.length && newY >= 0 && newY < grid[0].length) {
47                if (!grid[newX][newY].wall && !grid[newX][newY].obstacle && !grid[newX][newY].isCounter/*66 (newX != x || newY != y)
48                    */){
49                    x = newX;
50                    y = newY;
51                }else{
52                    if (grid[newX][newY].wall) System.out.printf("に衝突しましたwall\n");
53                    if (grid[newX][newY].obstacle) System.out.printf("に衝突しましたobstacle\n");
54                }
55            }
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;
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。");
91         }
92         return;
93     }
94 }
95
96 else if(frontGrid.tool == 10 && frontGrid.hasFood() && frontGrid.cookingGauge >= 60){
97     System.out.println("炊けた米をとります。");
98     frontGrid.food.foodStatus = 3;
99     food = frontGrid.food;
100    frontGrid.food = null;
101    frontGrid.cookingGauge = 0; //米をとったらリセット
102    return;
103 }
104 }
105
106 public void pick_up() {
107     Grid currentGrid = grid[x][y]; //自分の足元のグリッド
108     Grid frontGrid = getFrontGrid(); //自身の目の前のグリッド
109     System.out.printf("frontGrid:=(%d,%d)\n", frontGrid.x, frontGrid.y);
110     if(frontGrid.tool == 10){return;} //鍋からはアクションでしか食材をとれない。 Yoshida
111     if(hasPlate == false && frontGrid.tool == 3 ){ //は皿を持っていないplayer かつ目の前マスが皿ボックス
112         AudioManager se = new AudioManager();
113         se.playSE("./sound/music_have.wav");
114         System.out.println("皿を持ちました");
115         plate = new Plate(); //ここでお皿をもった
116         hasPlate = true; //皿を持つ
117     }else if(hasPlate == false && frontGrid.isPlatePlaced == true){ //は皿を持っていないplayer かつ目の前マスに皿がある
118         AudioManager se = new AudioManager();
119         se.playSE("./sound/music_have.wav");
120         hasPlate = true; //皿を持つ
121         plate = frontGrid.plate;
122         frontGrid.isPlatePlaced = false; //目の前マスから皿を回収
123         frontGrid.plate = null;
124         //food = frontGrid.food;
125         //frontGrid.food = null;
126     }
127     else if (food == null) { // 何も持っていない場合
128         if(frontGrid.foodBox == 1){ //目の前のマスがキャベツボックスだったら
129             AudioManager se = new AudioManager();
130             se.playSE("./sound/music_have.wav");
131             this.food = new Cabbage();
132             System.out.println("キャベツボックスから取得しました!");
133         }
134         else if(frontGrid.foodBox == 2){ //目の前のマスがトマトボックスだったら
135             AudioManager se = new AudioManager();
136             se.playSE("./sound/music_have.wav");
137             this.food = new Tomato();
138             System.out.println("トマトボックスから取得しました!");
139         }else if(frontGrid.foodBox == 3){ //目の前のマスがきゅうりボックスだったら
140             AudioManager se = new AudioManager();
141             se.playSE("./sound/music_have.wav");
142             this.food = new Cucumber();
143             System.out.println("きゅうりボックスから取得しました!");
144         }else if(frontGrid.foodBox == 4){ //目の前のマスが米ボックスだったら
145             AudioManager se = new AudioManager();
146             se.playSE("./sound/music_have.wav");
147             this.food = new Rice();
148             System.out.println("ライスボックスから取得しました!");
149         }else if(frontGrid.foodBox == 5){ //目の前のマスがまぐろボックスだったら
150             AudioManager se = new AudioManager();
151             se.playSE("./sound/music_have.wav");
152             this.food = new Tuna();
153             System.out.println("マグロボックスから取得しました!");
154         }else if(frontGrid.foodBox == 6){ //目の前のマスがいかボックスだったら
155             AudioManager se = new AudioManager();
156             se.playSE("./sound/music_have.wav");
157             this.food = new Squid();
158             System.out.println("イカボックスから取得しました!");
159         }else if(frontGrid.foodBox == 7){ //目の前のマスがのりボックスだったら
160             AudioManager se = new AudioManager();
161             se.playSE("./sound/music_have.wav");
162             this.food = new Seaweed();
163             System.out.println("のりボックスから取得しました!");
164         }
165     }
166     else if (frontGrid.hasFood()) { // 現在のマスに食材がある場合
167         AudioManager se = new AudioManager();
168         se.playSE("./sound/music_have.wav");
169         food = frontGrid.food; // 食材を拾う
170         frontGrid.food = null; // マスから食材を消す
171         System.out.println("食材を持ち上げました!");
172     } else {
173         System.out.println("ここには食材がありません。");
174     }
175 }
176
177 }
178
179 public void put(){
180     Grid currentGrid = grid[x][y];
181     Grid frontGrid = getFrontGrid();
182     if(frontGrid.tool == 13){
183         hasPlate = false;
184         plate = null;
185         food = null;
186         System.out.println("ゴミ箱に捨てられました");
187     }
188     //皿を持っていて 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
189     else if((hasPlate) && frontGrid.tool==0 && frontGrid.isCounter==false && frontGrid.food==null) {

```

```

190         hasPlate = false; //皿を捨てる置く()
191         frontGrid.isPlatePlaced = true;
192         frontGrid.plate = plate; //プレイヤーが持っている皿をグリッドにわたす
193         plate = null; //プレイヤーは皿を離す
194     }
195     //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
196     else if((hasPlate) && frontGrid.tool==0 && frontGrid.isCounter==false && frontGrid.food!=null){
197         plate.add(frontGrid.food); //まず最初に自分のにを追加する。platefood
198         frontGrid.isPlatePlaced = true;
199         frontGrid.plate = plate;
200         plate = null;
201         hasPlate = false;
202         frontGrid.food = null;
203         System.out.printf("デバッグ\n");
204         //plate.printPlate();
205     }
206     /* else */if(hasPlate==true && frontGrid.isCounter==true) { //いま皿を持っていて かつ目の前がカウンター
207         System.out.println("カウンターに提供します。");
208         hasPlate = false; //皿を捨てる置く()
209         frontGrid.plate = plate;
210         plate = null;
211         frontGrid.isPlatePlaced = true;
212         Order currentOrder = model.matchOrder(frontGrid.plate);
213         if(currentOrder == null){ // 料理が失敗だったとき
214             System.out.println("失敗作が提出されました");
215             model.scoreDown(currentOrder);
216             //失敗した場合、回収されて減点
217             view.addWaiter(view.setPlateImage(frontGrid.plate));
218             hasPlate = false;
219             plate = null;
220             frontGrid.food = null;
221             frontGrid.plate = null;
222             frontGrid.isPlatePlaced = false;
223             return;
224         }else{ //注文が正しかったとき
225             //view.addWaiter(currentOrder);
226             AudioManager se = new AudioManager();
227             se.playSE("./sound/music_success.wav");
228             view.addWaiter(view.setOrderImage(currentOrder));
229             model.scoreUp(currentOrder);
230             hasPlate = false;
231             frontGrid.plate = null;
232             frontGrid.food = null;
233             frontGrid.isPlatePlaced = false;
234         }
235     }
236     if(food != null) { // 既に食材を持っている場合
237         if(frontGrid.isPlatePlaced == true){ //目の前のマスに皿が置いてある場
238             System.out.println("皿に食材を追加します!");
239             frontGrid.plate.add(food);
240             food = null;
241             Order currentOrder = model.matchOrder(frontGrid.plate);
242             System.out.println("皿に食材を追加しました!");
243             frontGrid.plate.printPlate();
244         }else if (!frontGrid.hasFood() && frontGrid.tool == 0) { // 現在のマスが空いている場合かつそのマスがツールマスではない
245             frontGrid.food = food; // 食材を置く
246             food = null; // 手持ちを空にする
247             System.out.println("皿がないマスに対して食材を置きました!");
248         }
249         else {
250             if(frontGrid.hasFood() == true) System.out.println("ここには既に食材があります!");
251             if(frontGrid.tool != 0) System.out.printf("ここはツールなので食材は置けません");
252         }
253     }
254 }
255 }

```

● 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()); // グリッドバッ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); // フォールバック用フォント
54     }
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         JButton restartButton = new JButton("Restart");
51         restartButton.setFont(pixelFont.deriveFont(50f));
52         restartButton.setPreferredSize(new Dimension(300, 100));
53         restartButton.addActionListener(e -> mainApp.restartGame());
54
55         JButton closeButton = new JButton("Close");
56         closeButton.setFont(pixelFont.deriveFont(50f));
57         closeButton.setPreferredSize(new Dimension(300, 100));
58         closeButton.addActionListener(e -> System.exit(0));
59
60         buttonPanel.add(restartButton);
61         buttonPanel.add(closeButton);
62
63         gbc.gridy = 3;
64         add(buttonPanel, gbc);
65     }
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、何もしなければなる0 カットしてたら1
8      public boolean canCut; //その食材がカット可能ならtrue
9      public boolean canHeat; //その食材が加熱可能ならtrue
10     public boolean isOnPlate; //皿の上に置かれているか
11     public String foodName;
12     public abstract int getFoodStatus();
13
14     public Food(int foodStatus, boolean canCut, boolean canHeat, boolean isOnPlate, String foodName){
15         this.foodStatus = foodStatus;
16         this.canCut = canCut;
17         this.canHeat = canHeat;
18         this.isOnPlate = isOnPlate;
19         this.foodName = foodName;
20     }
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
34 //クラスを継承したクラスですFoodTomato
35 class Tomato extends Food{
36     public Tomato(){
37         super(1, true, false, false, "tomato");
38     }
39     public int getFoodStatus(){ //そのフードの状態を返す
40         return foodStatus;
41     }
42 }
43
44 //クラスを継承したクラスですFoodcucumber
45 class Cucumber extends Food{
46     public Cucumber(){
47         super(1, true, false, false, "cucumber");
48     }
49     public int getFoodStatus(){ //そのフードの状態を返す
50         return foodStatus;
51     }
52 }
53
54 //クラスを継承したクラスですFoodrice
55 class Rice extends Food{
56     public Rice(){
57         super(1, false, true, false, "rice");
58     }
59     public int getFoodStatus(){ //そのフードの状態を返す
60         return foodStatus;
61     }
62 }
63
64 //クラスを継承したクラスですFoodtuna
65 class Tuna extends Food{
66     public Tuna(){
67         super(1, true, false, false, "tuna");
68     }
69     public int getFoodStatus(){ //そのフードの状態を返す
70         return foodStatus;
71     }
72 }
73
74 //クラスを継承したクラスですFoodsquid
75 class Squid extends Food{
76     public Squid(){
77         super(1, true, false, false, "squid");
78     }
79     public int getFoodStatus(){ //そのフードの状態を返す
80         return foodStatus;
81     }
82 }
83
84 //クラスを継承したクラスですFoodseaweed
85 class Seaweed extends Food{
86     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             if (foods[i] == null) {
105                 foods[i] = food;
106                 System.out.println(food.foodName + "を皿に追加しました。");
107                 return; // 追加が完了したら終了
108             }
109         }
110         System.out.println("これ以上皿に食材を追加できません。");
111     }
112
113     public Food get(int i){
114         if(i<0 || i>=foods.length){return null;}
115         else return foods[i];
116     }
117
118     public void printPlate(){
119         String state = "";
120         System.out.print("現在、皿の上には:");
121         for(int i=0; i<3; i++){
122             if(foods[i] != null) {
123                 switch(foods[i].foodStatus){
124                     case 1: state = "raw"; break;
125                     case 2: state = "cut"; break;
126                     case 3: state = "grilled"; break;
127                 }
128                 System.out.print(foods[i].foodName+"(" + state + ") " + " ");
129             }
130         }
131         System.out.print("\n");
132         return ;
133     }
134
135     public boolean matchesOrder(Order order) {
136         boolean[] matchedIngredients = new boolean[3];
137         Food[] orderIngredients = {order.ingredient1, order.ingredient2, order.ingredient3};
138
139         // 皿にある食材の数をカウント
140         int plateFoodCount = 0;
141         for (int i=0; i<3; i++) {
142             if (foods[i] != null) {
143                 plateFoodCount++;
144             }
145         }
146
147         // オーダーの食材リストを作成
148         int orderFoodCount = 0;
149         for (int i=0; i<3; i++) {
150             if (orderIngredients[i] != null) {
151                 orderFoodCount++;
152             }
153         }
154
155         // 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****
156         if (plateFoodCount != orderFoodCount) {
157             System.out.println("料理の食材数がオーダーと一致しません。");
158             return false;
159         }
160
161         for (int i = 0; i < foods.length; i++) {
162             for (int j = 0; j < orderIngredients.length; j++) {
163                 if(orderIngredients[j] == null){
164                     matchedIngredients[j] = true;
165                     continue;
166                 }
167                 if (!matchedIngredients[j] && foods[i] != null) {
168                     if (foods[i].getClass() == orderIngredients[j].getClass() &&
169                         foods[i].foodStatus == orderIngredients[j].foodStatus) {
170                         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);
74             //飯で正方形を描画してるよ
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             flame++;
78         }else if(comeFlame <= flame && flame < 2*comeFlame){
79             g.drawImage(imgWaiterUp, xBefore-10, receiveY + rightBlank, cS+20, cS+20, io);
80             flame++;
81         }else if(2*comeFlame <= flame && flame < 3*comeFlame){
82             g.drawImage(imgWaiterDown, xAfter-10, (int)((waitY*(flame-2*comeFlame) + receiveY*(3*comeFlame-flame))/comeFlame) +
                rightBlank, cS+20, cS+20, io);
83             flame++;
84         }else if(flame == 3*comeFlame){ active = false; flame++;}
85     }
86 }

```

● 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BGM
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SE
31     public static void playSE(String filePath) {
32         new Thread(() -> {
33             try {
34                 File soundFile = new File(filePath);
35                 AudioInputStream audioStream = AudioSystem.getAudioInputStream(soundFile);
36                 Clip seClip = AudioSystem.getClip();
37                 seClip.open(audioStream);
38                 seClip.start(); // 短いならそのまま再生SE
39             } catch (UnsupportedAudioFileException | IOException | LineUnavailableException e) {
40                 e.printStackTrace();
41             }
42         }).start();
43     }
44 }
```

文責：米谷・鈴木・吉田