

情報学群実験第 1 最終レポート

1250373 溝口洸熙 *

2022 年 7 月 30 日

概要

このレポートは、情報学群実験第 1 の最終レポートである。マインスイーパーを作成し、それに対して指定された仕様 1～5 と、追加した仕様 6～9 の処理概要や処理の詳解がされている。仕様に対するソースコードは p.14 から記載している。実行は問題なくできており、仕様通りの動作となっている。この課題に取り組むに当たって、オブジェクト指向プログラミングの概念を再修得できた。

* 高知工科大学 情報学群 2 年生

目次

はじめに	3
第Ⅰ部 必須仕様	4
仕様 1	4
1.1 処理概要	4
1.2 処理	5
仕様 2	6
2.1 処理概要	6
2.2 処理	7
仕様 3	8
3.1 処理概要	8
3.2 処理	8
仕様 4	9
4.1 処理の概要	9
4.2 処理	10
仕様 5	11
5.1 処理	11
5.2 実装箇所	11
第Ⅱ部 追加仕様	12
仕様 6	12
仕様 7	12
仕様 8	12
仕様 9	13
第Ⅲ部 ソースコード	14
ソースコード	14
第Ⅳ部 まとめ	17

はじめに

レポートについて

このレポートは、 $\text{L}^{\text{A}}\text{T}_{\text{E}}\text{X} 2_{\epsilon}$ を用いて作成している。図やグラフは TikZ を用いて描画しており、ソースコードは `listing` を用いて表記している。各参照には参照箇所まで誘導するリンクもついている。

符号化と変数

あるパネルのステータスを示す符号と、新たに追加したグローバル変数を、以下に示す。また、このマイ

符号とステータス		新たに追加した変数	
符号	ステータス	変数名	役割
0	爆弾以外	<code>int originalTable</code>	生成した盤面の初期状態を記憶する。
1	開かれたパネル		
-1	爆弾	<code>Boolean tr</code>	2 手目以降で <code>true</code> になる変数。
-2	旗が立っている		1 手目で爆弾に当たることを回避するため。

ンスーパーは Java の `Button` クラスが使用されている。ボタンを離した時にアクションが起こるので、以下の情報を参考に、`Main.java` を解説した。

ボタンが押されて離されると、AWT はボタンの `processEvent` を呼び出すことにより、ボタンに `ActionEvent` のインスタンスを送ります。ボタンの `processEvent` メソッドはそのボタンのすべてのイベントを受け取ります。ボタンは自身の `processActionEvent` メソッドを呼び出すことによってアクション・イベントを渡します。後者のメソッドはこのボタンによって生成されるアクション・イベントの通知を対象として登録されているアクション・リスナーにアクション・イベントを渡します。[1]

第 I 部

必須仕様

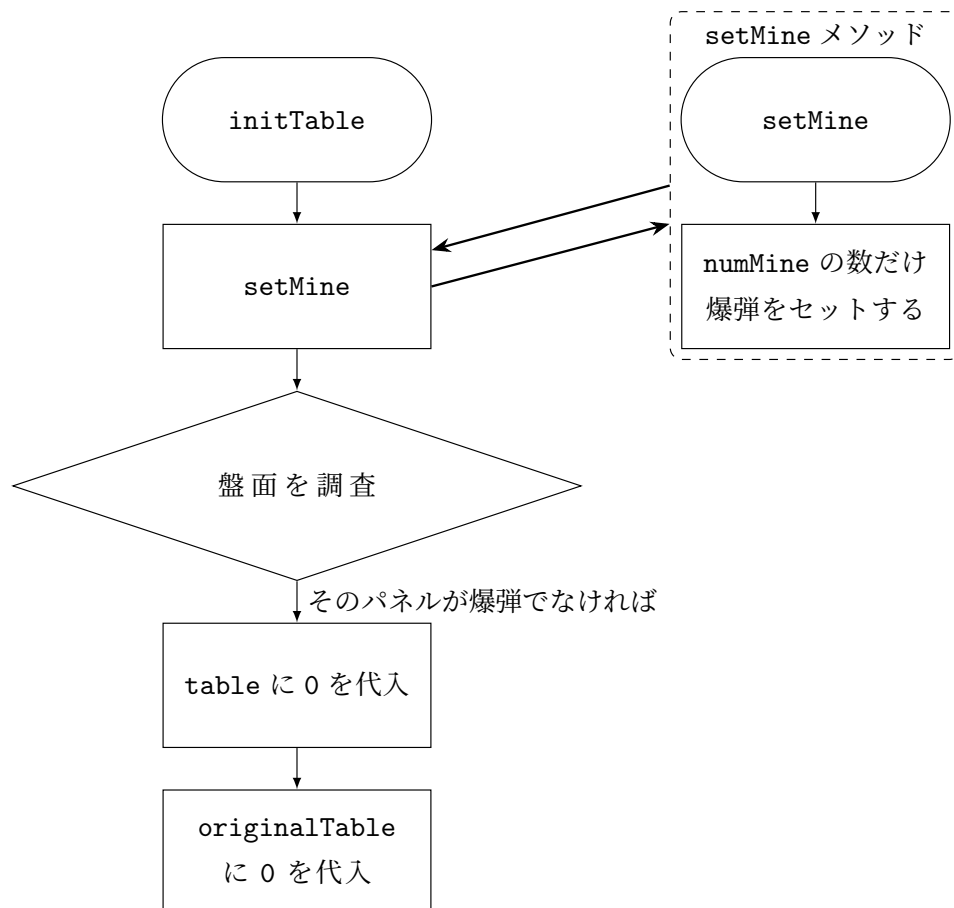
仕様 1

仕様 1.

ゲーム開始時に、盤面上へランダムに地雷を設置する.

1.1 処理概要

Fig 1: 盤面上へランダムに地雷を設置する



1.2 処理

initTable (src. 1) の処理

盤面を初期化するにあたって、以下の処理を行う。

- 1) table,originalTable の全ての行と列に 0 を代入する。ただし、爆弾であるパネルは上書きしない。
- 2) 爆弾の設置パネルを setMine メソッドで定める。

```
for (int i = 0; i < table.length; i++) {
    for (int j = 0; j < table.length; j++) {
        this.table[i][j] = 0;
        this.originalTable[i][j] = 0;
    }
}
this.setMine();
```

setMine (src. 2) の処理

盤面に爆弾を配置するにあたって、以下の処理を行う。

- 1) 爆弾の個数を数える count 変数を定義する。

```
int count = 0;
```

- 2) 指定された爆弾の個数が count になるまで、爆弾を配置する。

```
while (count != this.numMine) { //
    numMine の数だけ爆弾をセットできたらループを抜ける
    ...
}
```

- 3) 爆弾の配置はランダムである。乱数で指定されたパネルが既に爆弾であれば再度乱数を生成し、爆弾が新たにセットできる場所では、table,originalTable の乱数値 Index を-1 に設定し、count をインクリメントする。

```
...
int x = new java.util.Random().nextInt(getHeight());
int y = new java.util.Random().nextInt(getWidth());
if (this.table[x][y] == -1) {
    // [x][y]にすでに爆弾がセットされていたら、もう一度乱数を決め直す
    continue;
}
count++;
...
```

仕様 1 終

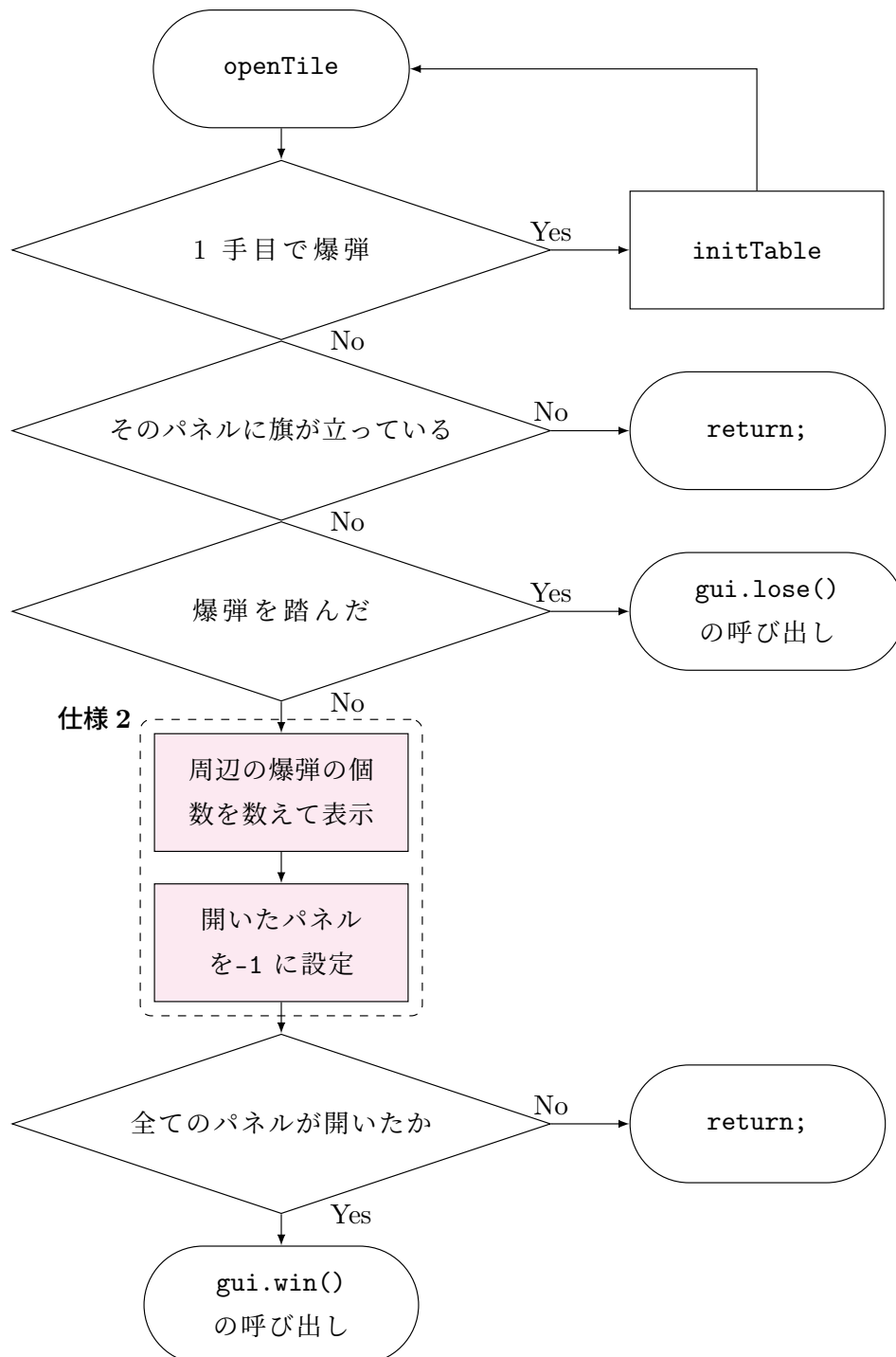
仕様 2

処理 2.

パネルを左クリックした際、クリックしたパネルを開く.

2.1 処理概要

Fig 2: タイルを開くときの処理



2.2 処理

openTile (src. 3) の処理の一部

- 1) パネルに爆弾がない場合、その周辺の爆弾個数を returnMine (src. 4) メソッドで取得し、そのパネルに表示する.

```
String mc = String.valueOf(mineCount);  
gui.setTextToTile(x, y, mc); // 爆弾の個数を表示
```

- 2) 全てのパネルが開いたか否か確認する. もし、全てのパネルが開いていたら勝利となるので gui.win を呼び出し、開いていないパネルが存在すれば、return; する.

```
int mineCount = this.returnMine(x, y, gui); // 周辺の爆弾の個数を調査  
this.table[x][y] = 1; // 開かれたパネルの値を1に設定  
...  
// 爆弾以外のパネルが全て開いているか確認  
for (int i = 0; i < getHeight(); i++) {  
    for (int j = 0; j < getWidth(); j++) {  
        if (this.table[i][j] == 0) { return; }  
    }  
}  
gui.win();
```

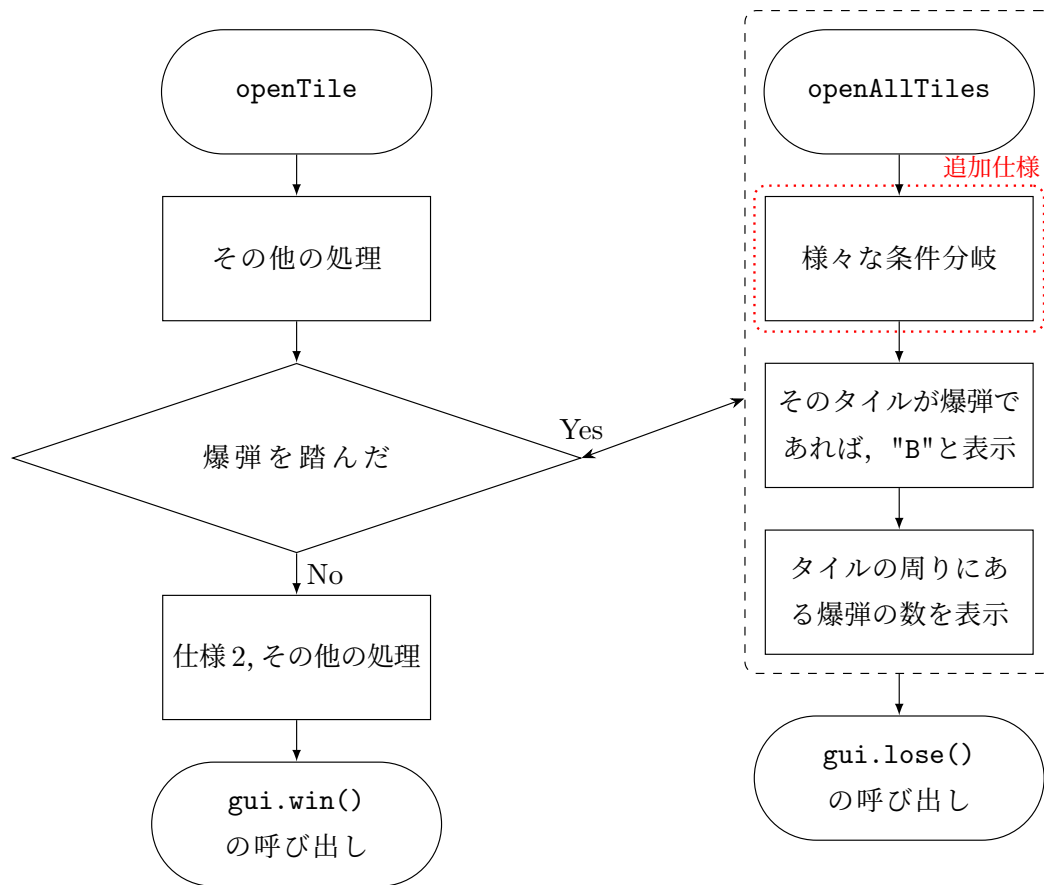
仕様 3

仕様 3.

開いたパネルに地雷が隠されている場合、全てのパネルを開く。

3.1 処理概要

Fig 3: 全てのパネルを開く



追加仕様に関しては、仕様 8 で詳説する。

3.2 処理

全てのタイルに対して、条件分岐を行う。ただ単に全てのタイルに対して `openTile` メソッドを呼び出してもよかったが、その仕様では、自分がどこにフラグを立てて、そのフラグが正しいか否かの判断ができかねる。従って、`openAllTiles` メソッド (src. 5) に関しては、仕様 8 で詳解する。

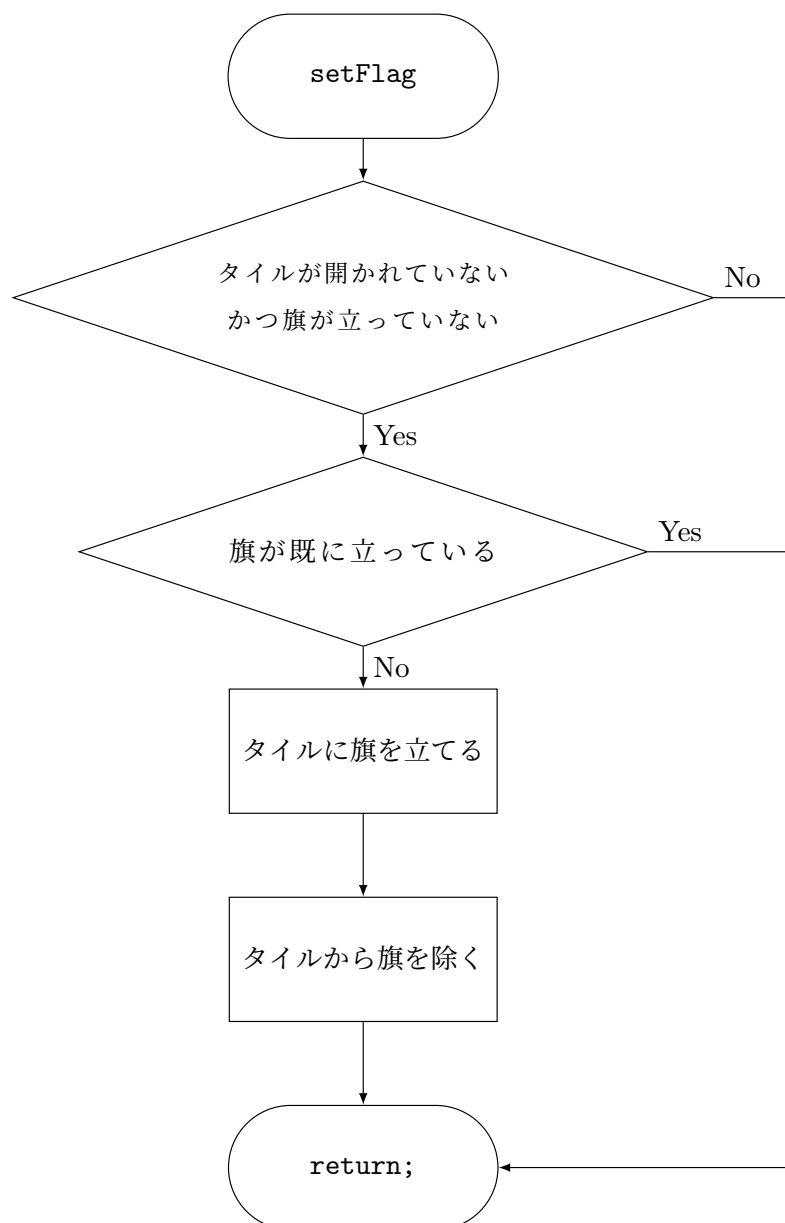
仕様 4

仕様 4.

開いていないパネルを右クリックした際、そのパネルに旗を立てる。また、旗が立てられているパネルの場合には畑を取り除き、旗が取り除かれるまで左クリックでパネルは開けない。

4.1 処理の概要

Fig 4: 旗を立てるときの処理



4.2 処理

setFlag (src. 6) の処理

今, 入力として x 行 y 列 の Index が与えられた.

- 1) そのタイルが開いていない時に旗を立てる.

```
if (this.table[x][y] == 0 || this.table[x][y] == -1) {  
    this.table[x][y] = -2; // 旗を立てる場所に-2を入れる  
    gui.setTextToTile(x, y, "F");  
}
```

- 2) そのタイルに既に旗が立っているとき, そのタイルを初期状態に戻す.

```
else if (this.table[x][y] == -2) {  
    this.table[x][y] = this.originalTable[x][y];  
    gui.setTextToTile(x, y, "");  
}
```

.....

openTile メソッドで, タイルにはたがあるか, 既に開けられたタイルであれば, return; する処理がある.

```
if (this.table[x][y] == 1 || this.table[x][y] == 2) {  
    return;  
}
```

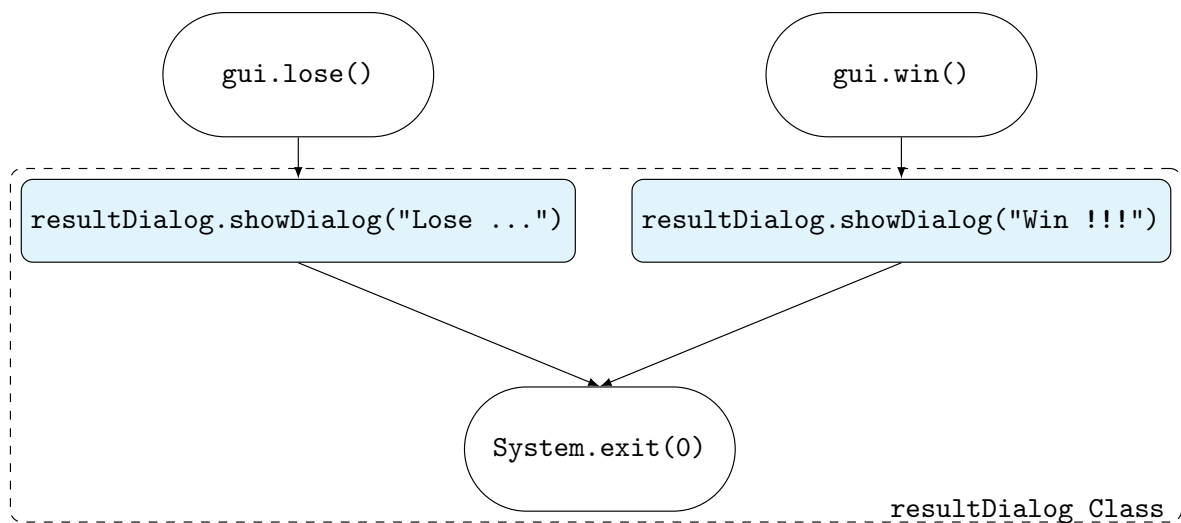
仕様 5

仕様 5.

クリアもしくはゲームオーバーになった際、適切なダイアログを表示する。

5.1 処理

Fig 5: ダイアログの表示



5.2 実装箇所

gui.lose の実装箇所

openTile メソッド. 爆弾を開けたとき.

gui.win の実装箇所

openTile メソッド. 全てのタイルが開かれた, もしくは爆弾には旗が立っている場合.

Java 仮想マシンを終了するには, `System.exit(0)` の実行が必要. `System.exit` に関しては以下を参照した.

`System` クラスには有用なクラス・フィールドおよびメソッドがあります. インスタンス化することはできません.

`System.exit(int status)`

現在実行している Java 仮想マシンを終了します. [2]

第 II 部

追加仕様

仕様 6

「運」のみで勝敗が決まるのは、ゲームとしては面白くない。プレイヤーは 1 手目から爆弾か否かを判断することは、不可能なので、1 手目で爆弾のタイルを開けない仕様を追加した。

仕様 6 の実装

openTile メソッドにおいて、1 手目で爆弾のタイルを選択した場合の条件分岐を設けている。

```
public void openTile(int x, int y, MinesweeperGUI gui) {
    ...
    if (this.table[x][y] == -1 && !this.tr) {
        this.initTable();
        this.openTile(x, y, gui);
    }
    this.tr=true;
    ...
}
```

仕様 7

実際のマインスイーパーは、数字ごとに色が決められている。今回作成したマインスイーパーも同様にタイルのテキストに色を設定するメソッドを作成した。(src. 7)

さらに、爆弾のタイルや、既に開けているタイルを識別しやすくするため、タイルの背景色を変更するメソッドも作成した。(src. 8) 以下を参考にして、Main.java を変更した。

このコンポーネントのバックグラウンド・カラーを設定します。バックグラウンド・カラーが各コンポーネントにそれぞれ異なる影響を与えます。[3]

スタイルから取得できる型保証された色の列挙です。領域のフォアグラウンド用の ColorType です。

[4]

仕様 8

実際のマインスイーパーは、旗を立てたところが爆弾でなければ、旗にばつ印をつける使用になっている。また、旗の位置が正しければ、その旗は表示されたままだ。この機能を実装したいと思い、openAllTiles メソッド (src. 5) に幾らかの条件分岐を設けた。条件と表示の関係を Tbl 2 に示す。

Tbl 2: 条件とタイルの色及び表示

条件	真の時の処理		
	表示	背景色	文字色
爆弾の場所に旗が立っている	"F"	指定なし	黒
爆弾でない場所に旗が立っている	"XB"	黒	白
爆弾の場所に旗が立っていない	"B"	赤	black
周りの爆弾が 0 でかつ開けられていない	" "	白	—
その他	周りの爆弾の個数	指定なし	個数に応じた色

仕様 9

実際のマインスイーパーは、周辺にある爆弾の個数が 0 個の場合、0 と表示するのではない。周辺にある爆弾の数が 0 個の場合は、「その周辺のタイルを開けることができる」ということは、明らかであるため自動的に開けてくれるのだ。今回作成したマインスイーパーでも、その機能を実装した。


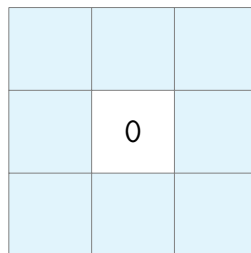

Fig 6 に対して、 の色のタイルは、爆弾がないことを示している。これは明らかであり、プレイヤーに無駄な労力をかけることになるので、プログラムで自動的に開けることにした。

Fig 6



範囲外を除いて、周りのタイルを再起的に開けていく方法をとった。これにより  の、タイルの周りの爆弾の個数が 0 個であったとしても、またさらにその周りのタイルを開けることができる。（これが再起の最大のメリットである。）

```

if (mineCount == 0) { // もし周辺に爆弾がなければ、そのマスも開ける。
for (int i = x - 1; i < x + 2; i++) {
    if (i < 0 || i >= getHeight()) { continue; } // 範囲外
    for (int j = y - 1; j < y + 2; j++) {
        if (j < 0 || j >= getWidth()) { continue; } // 範囲外
        openTile(i, j, gui); // 再起的に呼び出し
    }
}
}

```

第III部

ソースコード

MineSweeper.java > MineSweeper class

src. 1: initTable

```
1 void initTable() {
2     for (int i = 0; i < table.length; i++) {
3         for (int j = 0; j < table.length; j++) {
4             this.table[i][j] = 0;
5             this.originalTable[i][j] = 0;
6         }
7     }
8     this.setMine();
9 }
```

src. 2: setMine

```
1 void setMine() {
2     int count = 0;
3     while (count != this.numMine) { //
4         numMineの数だけ爆弾をセットできたらループを抜ける
5         int x = new java.util.Random().nextInt(getHeight());
6         int y = new java.util.Random().nextInt(getWidth());
7         if (this.table[x][y] == -1) {
8             // [x][y]にすでに爆弾がセットされていたら、もう一度乱数を決め直す
9             continue;
10        }
11        this.table[x][y] = -1; // 爆弾の場所を値-1としてセットする
12        this.originalTable[x][y] = -1;
13        count++;
14    }
```

src. 3: openTile

```
1 public void openTile(int x, int y, MineSweeperGUI gui) {
2     if (this.table[x][y] == 1 || this.table[x][y] == 2) {
3         return;
4     }
5     if (this.table[x][y] == -1 && !this.tr) { // 1手目で爆弾ならば再度爆弾をセ
6         ット
7         this.initTable();
8         this.openTile(x, y, gui);
9     }
10    this.tr = true;
11    if (this.table[x][y] == -1) { // パネルに爆弾があった場合
12        this.openAllTiles(gui);
13    }
```

```

12     gui.lose();
13 } else { // パネルに爆弾がなかった場合
14     int mineCount = this.returnMine(x, y, gui); // 周辺の爆弾の個数を調査
15     this.table[x][y] = 1; // 開かれたパネルの値を1に設定
16     if (mineCount == 0) { // もし周辺に爆弾がなければ、そのマスも開ける
17         for (int i = x - 1; i < x + 2; i++) {
18             if (i < 0 || i >= getHeight()) { // パネルの範囲外は除く
19                 continue;
20             }
21             for (int j = y - 1; j < y + 2; j++) {
22                 if (j < 0 || j >= getWidth()) { // パネルの範囲外は除く
23                     continue;
24                 }
25                 openTile(i, j, gui);
26             }
27         }
28     }
29     String mc = String.valueOf(mineCount);
30     if (mineCount == 0) {
31         gui.setColorBackground(x, y, 0);
32         gui.setTextToTile(x, y, "");
33     } else {
34         gui.setColorText(x, y, mineCount); // 色を設定
35         gui.setTextToTile(x, y, mc); // 爆弾の個数を表示
36     }
37     for (int i = 0; i < getHeight(); i++) { // 爆弾以外のパネルが全て開いて
        いるか確認
38         for (int j = 0; j < getWidth(); j++) {
39             if (this.table[i][j] == 0) {
40                 return;
41             }
42         }
43     }
44     gui.win();
45 }
46 }

```

src. 4: returnMine

```

1 private int returnMine(int x, int y, MinesweeperGUI gui) {
2     int mineCount = 0;
3     for (int i = x - 1; i < x + 2; i++) {
4         if (i < 0 || i >= getHeight()) { // パネルの範囲外は除く
5             continue;
6         }
7         for (int j = y - 1; j < y + 2; j++) {
8             if (j < 0 || j >= getWidth()) { // パネルの範囲外は除く
9                 continue;
10            }
11            if (this.originalTable[i][j] == -1) { // 爆弾の個数をカウント
12                mineCount++;

```

```

13         }
14     }
15 }
16 return mineCount;
17 }

```

src. 5: openAllTiles

```

1 private void openAllTiles(MineSweeperGUI gui) {
2     for (int x = 0; x < getHeight(); x++) {
3         for (int y = 0; y < getWidth(); y++) {
4             Integer i = this.returnMine(x, y, gui);
5             if (this.originalTable[x][y] == -1 && this.table[x][y] == -2) {//
                爆弾の場所にフラグ
6                 gui.setTextToTile(x, y, "F");
7                 this.table[x][y] = 1;
8             } else if (this.originalTable[x][y] != -1 && this.table[x][y] ==
                -2) {// 爆弾でない場所にフラグ
9                 gui.setTextToTile(x, y, "XB");
10                gui.setColorBackground(x, y, 3);
11                gui.setColorText(x, y, 0);
12            } else if (this.originalTable[x][y] == -1 && this.table[x][y] !=
                1) {// 爆弾の場所にフラグがない
13                gui.setColorBackground(x, y, 2);
14                gui.setColorText(x, y, 4);
15                gui.setTextToTile(x, y, "B");
16            } else if (i == 0 && this.table[x][y] != 1) {// 開けられていない
                場所の周りの爆弾の個数が0
17                gui.setColorBackground(x, y, 0);
18                gui.setTextToTile(x, y, "");
19            } else {
20                gui.setColorText(x, y, i);
21                gui.setTextToTile(x, y, i.toString());
22            }
23        }
24    }
25 }

```

src. 6: setFlag

```

1 public void setFlag(int x, int y, MineSweeperGUI gui) {
2     if (this.table[x][y] == 0 || this.table[x][y] == -1) {
3         this.table[x][y] = -2; // 旗を立てる場所に-2を入れる
4         gui.setTextToTile(x, y, "F");
5     } else if (this.table[x][y] == -2) {
6         // すでに旗が立っているときは元の盤面の値に戻す
7         this.table[x][y] = this.originalTable[x][y];
8         gui.setTextToTile(x, y, "");
9     }
10 }

```


src. 7: setColorText

```
1  @Override
2  public void setColorText(int x, int y, int num) {
3      switch (num) {
4          case 1:
5              this.tileTable[y][x].setForeground(Color.blue);
6              break;
7          case 2:
8              this.tileTable[y][x].setForeground(new Color(0, 125, 0));
9              break;
10         case 3:
11             this.tileTable[y][x].setForeground(Color.red);
12             break;
13         case 4:
14             this.tileTable[y][x].setForeground(new Color(0, 0, 97));
15             break;
16         case 0:
17             this.tileTable[y][x].setForeground(Color.white);
18             break;
19         default:
20             this.tileTable[y][x].setForeground(Color.black);
21     }
22 }
```

src. 8: setColorBackground

```
1  public void setColorBackground(int x, int y, int color) {
2      switch (color) {
3          case 0:
4              this.tileTable[y][x].setBackground(Color.white);
5              break;
6          case 1:
7              this.tileTable[y][x].setBackground(new Color(200, 255, 255)); //
              new Color(RGB)
8              break;
9          case 2:
10             this.tileTable[y][x].setBackground(Color.red); // new Color(RGB)
11             break;
12         case 3:
13             this.tileTable[y][x].setBackground(Color.black);
14             break;
15     }
16 }
```

第Ⅳ部

まとめ

今回のマインスイーパーの課題を通して、様々な学びがあった。

ゲームを模範して作るとなると、そのゲームのルールや性質をしっかりと学ぶ必要がある。幸にして、私は小学生の頃からマインスイーパーに触れており、(Windows xp 内蔵) ルールの理解という面では苦労しなかった。

しかし、実際にアルゴリズムとしてマインスイーパーを考えると、なかなか簡単にはいかず、一つ一つの処理を詳細に、かつ順を追って見ていく必要があった。コンピュータは正直者なので、少しでも違う条件を与えると、当然だが予想だにしない挙動をする。これがプログラミングの面白いところであるが、同時に難点でもある。一つ一つの処理が何を指すのか、その処理が無駄ではないのかななどを熟考した期間はとても充実していた。

今回作成したマインスイーパーにはまだ課題もある。それは、やや処理効率がわるいということだ。今回の課題では、それほど大きな2次元配列を扱うことはなかったが、これが大きな2次元配列になると、二重ループは好ましくない。 $O(n^2)$ の時間がかかる故。) 従って、今後は別の配列に対して、今の情報を格納する方法も検討したい。メモリーの問題も生じると思うので、実験の結果を見て、より良い解決策を模索したい。

最後に、このレポートをここまで読んでくださった指導者の方に感謝申し上げる。

参考文献

- [1] Java(tm) Platform Standard Edition 8 クラス Button (2022 年 7 月 30 日最終確認)
<https://docs.oracle.com/javase/jp/8/docs/api/java/awt/Button.html>
- [2] Java(tm) Platform Standard Edition 8 クラス System (2022 年 7 月 30 日最終確認)
<https://docs.oracle.com/javase/jp/8/docs/api/java/lang/System.html>
- [3] Java(tm) Platform Standard Edition 8 クラス Component (2022 年 7 月 30 日最終確認)
<https://docs.oracle.com/javase/jp/8/docs/api/java/awt/Component.html>
- [4] Java(tm) Platform Standard Edition 8 クラス ColorType (2022 年 7 月 30 日最終確認)
<https://docs.oracle.com/javase/jp/8/docs/api/javax/swing/plaf/synth/ColorType.html>