

情報学群実験第 1 最終レポート

1250373 溝口洸熙 *

2022 年 7 月 29 日

概要

目次

はじめに	2
第Ⅰ部 必須仕様	3
仕様 1	3
1.1 処理概要	3
1.2 処理	4
仕様 2	5
2.1 処理概要	5
2.2 処理	6
仕様 3	7
3.1 処理概要	7
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

* 高知工科大学 情報学群 2 年生

仕様 7	12
仕様 8	12
仕様 9	13
ソースコード	14

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．

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．

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		2 手目以降で <code>true</code> になる変数．
-2	旗が立っている	<code>Boolean tr</code>	1 手目で爆弾に当たることを回避するため．

第 I 部

必須仕様

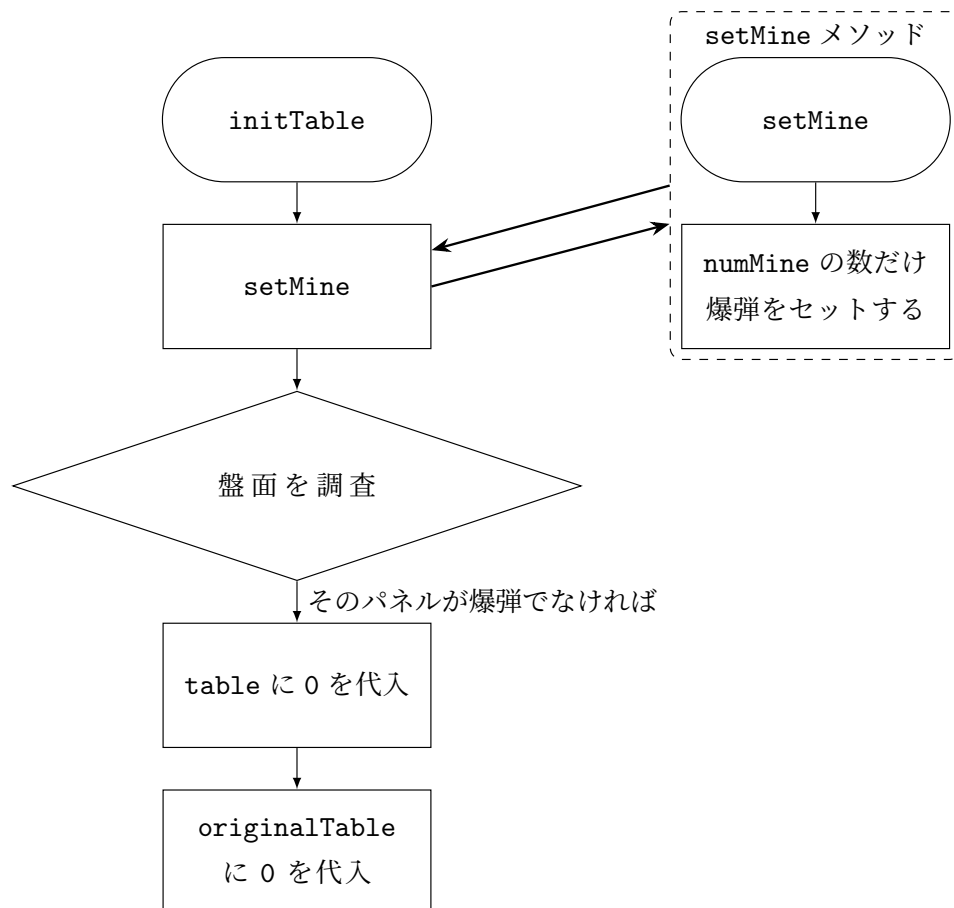
仕様 1

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

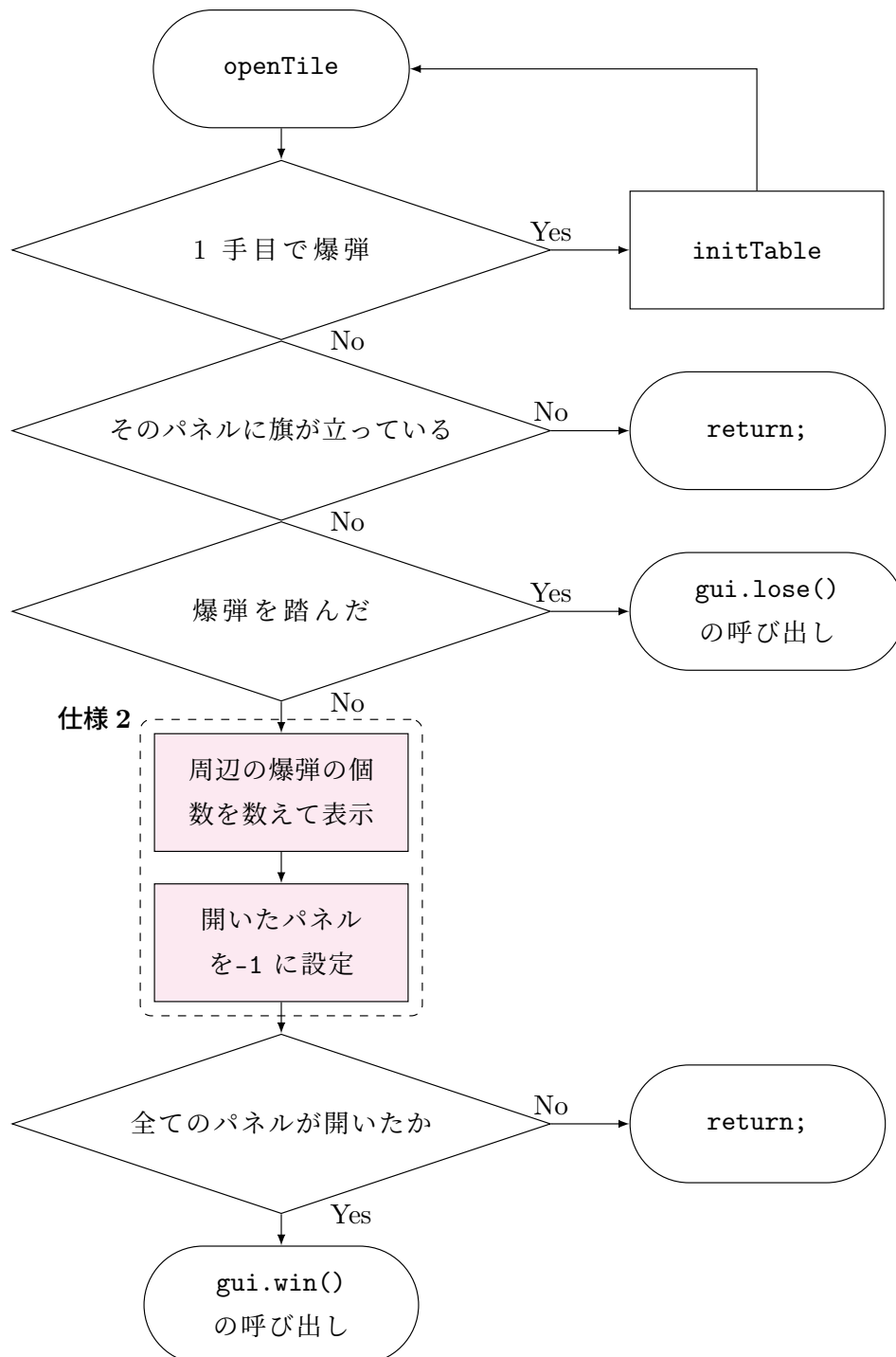
仕様 2

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();
```

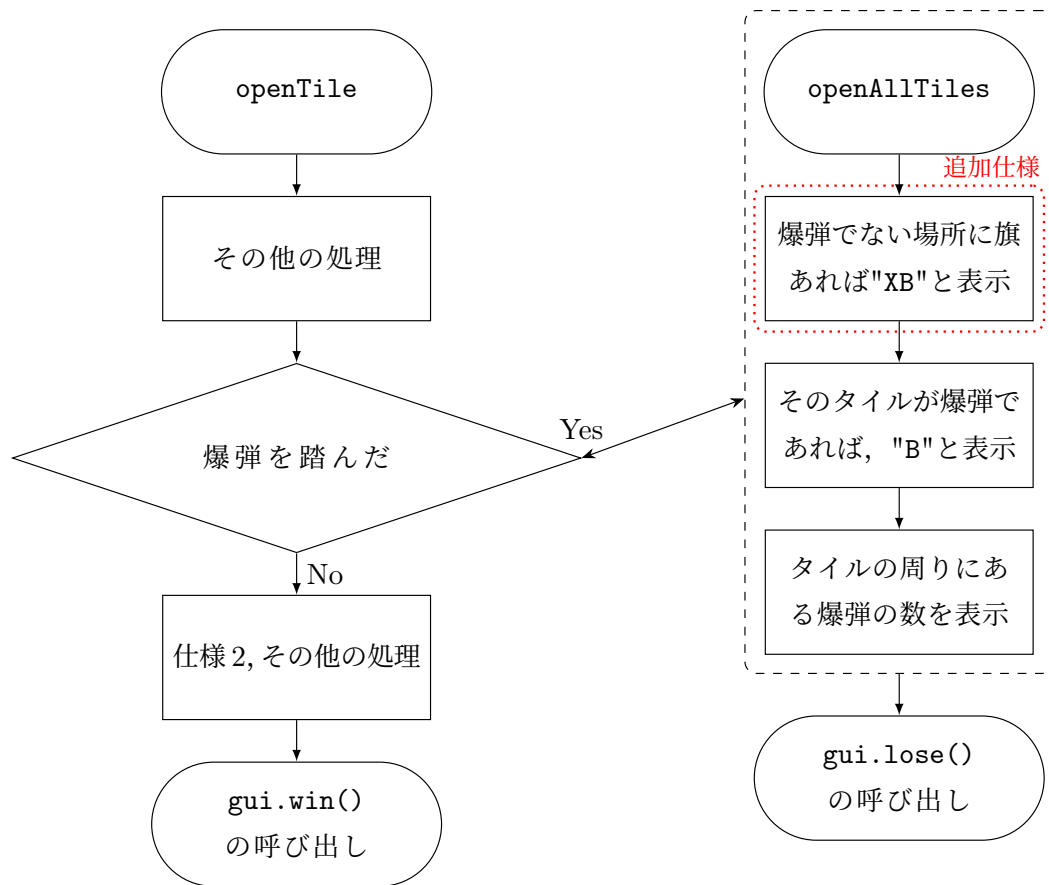
仕様 3

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



3.2 処理

`openAllTiles` (src. 5) の処理

全てのタイルに対して、条件分岐を行う.

- 1) そのタイルに旗は立っており、かつ爆弾ではない場合、タイルに"XB"と表示する. (第 II 部で詳説する.)
- 2) 1 には該当せず、そのタイルが爆弾である場合、タイルに"B"と表示する.

```
else if (this.originalTable[x][y] == -1) {  
    gui.setTextToTile(x, y, "B");  
    ...  
}
```

- 3) 1,2 いずれにも該当しない場合、周りにある爆弾の数を `returnMine` メソッドで数え上げ、その数をタイルに表示する.

```
else {  
    Integer i = this.returnMine(x, y, gui);  
    gui.setTextToTile(x, y, i.toString());  
    ...  
}
```

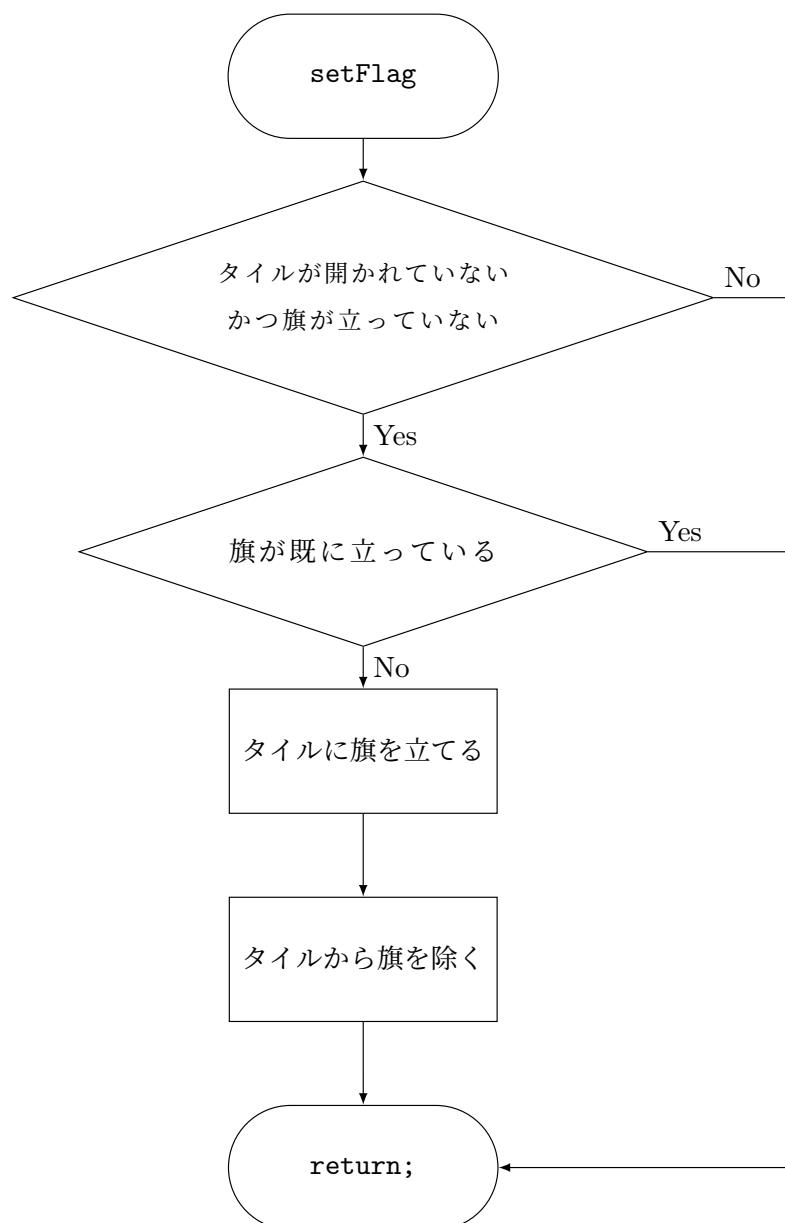
仕様 4

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, タイルにフラグがあれば return; する処理がある.

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

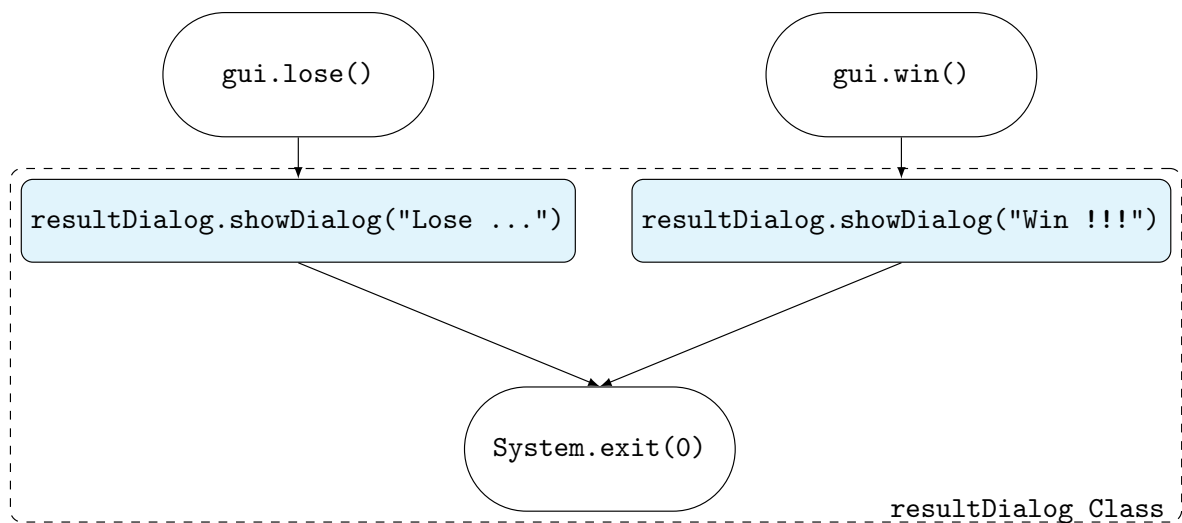
仕様 5

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.

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)

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。この機能を実装したいと思い、openAllTiles メソッドに幾らかの条件分岐を設けた。

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。


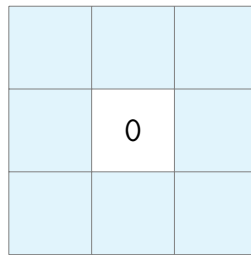

Fig 6 に対して、 の色のタイルは、爆弾がないことを示している。これは明らかであり、プレイヤーに無駄な労力をかけることになるので、プログラムで自動的に開けることにした。

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
    }
}
```

ソースコード

src. 1: initTable

src. 2: setMine

src. 3: openTile

src. 4: returnMine

src. 5: openAllTiles

src. 6: setFlag

src. 7: setColorText

src. 8: setColorBackground