プログラミングDレポート

担当教員　小南大智
提出日　2023年12月15日
氏名　山久保孝亮
学籍番号　09B22084
メールアドレス　u327468b@ecs.osaka-u.ac.jp

1　プログラムの操作方法と機能

今回のライフゲームの課題は Run ボタンが押されるとゲームが開始する．私が作成した機能と操作方法は以下のとおりである．

1.1　盤面の描画

今回のライフゲームではゲームが開始した際にウィンドウが開き, そこに next,undo,newgame のボタンとともに盤面が表示されるという仕様になっている．盤面のサイズの初期値は縦 300, 横 400 ピクセルで最小値は縦, 横である．生きている状態のセルを黒色, 死んでいる状態のセルを灰色で表すこととした．最初はすべてのセルが死んでいる状態となっている．また, 盤面の縦と横の座標は以下の図 1 ような仕様とした．これは,(i,j) と指定されると横軸が i, 縦軸が j の行と列をそれぞれ考え, それらが交差するセルを表す．

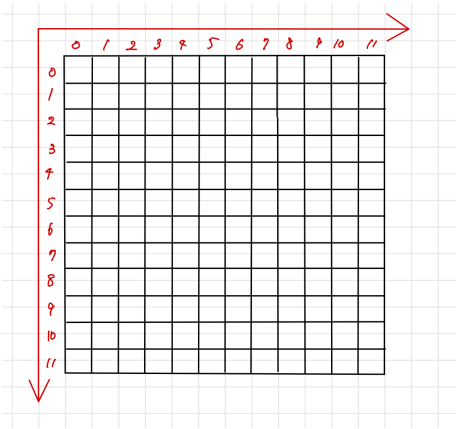

図 1: 座標の仕様

1.1.1　ウィンドウのサイズ変更

ウィンドウの大きさをユーザ側が変更した際にはその変更に合わせて盤面の大きさも変更される．以下の図 2 は Main クラス内のコードの一部である．赤く囲んだ部分の数字は縦のセルの個数, 青く囲んだ部分の数字は横のセルの個数を表す．ユーザはこの部分のコードを変更して盤面の個数を変更することができる．

```
public void run() {
    model = new BoardModel(12, 12,Undo);
    model.addListener(new ModelPrinter());
```

図 2: 該当部分のコード

ただしセルの形は常に正方形を保つ．パネル上にボタンを表示させる範囲を確保してから, そこを除いた部分に盤面を描写する．具体的な盤面内の座標の計算は実現方法のところで記述する．以下の図 2,3 はウィンドウを極端に横に大きくしたときの例と縦に大きくしたときの例である．

1.1.2　盤面のサイズ変更

盤面はそれぞれのセルの正方形が Main クラスで初期化されたセルの数だけそれぞれ縦と横に表示される．以下の 4,5 は RUN ボタンを押して出力した 12×12 と 18×12 の盤面である．

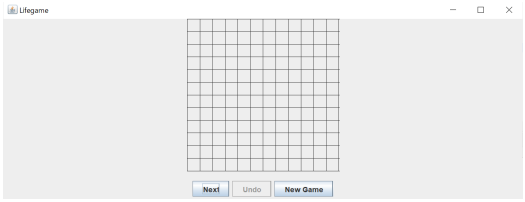

図 3: 極端に横に大きくしたときの盤面

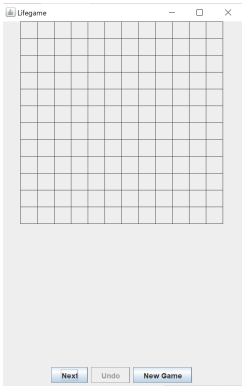

図 4: 極端に縦に大きくしたときの盤面

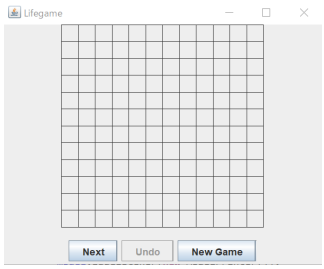

図 5: 12×12 の盤面

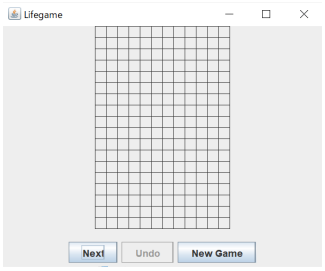

図 6: 18×12 の盤面

1.2 next,undo,newgame ボタン

ユーザは盤面の下部に表示されるそれぞれのボタンの上にマウスカーソルを移動し左クリックを押し込むことで以下の仕様を満たす処理を実行することができる. それぞれの処理は左クリックを押し込んで離したときに実行される.

1.2.1 next ボタン

next ボタンは最初から押せる状態になっており, ライフゲームの仕様に基づいて世代を一世代進める. 以下の図 6,7 は next ボタンを押して盤面の状態を一世代進めたときの例である.

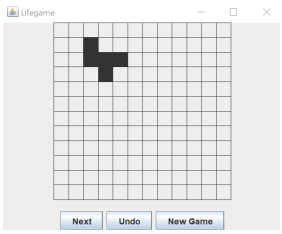

図 7: next ボタンを押す前の盤面

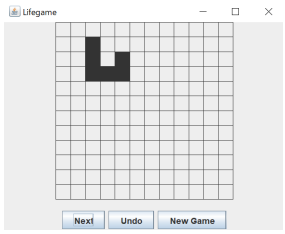

図 8: next ボタンを押した後の盤面

1.2.2 undo ボタン

undo ボタンは最初は押せない状態になっている.next ボタンを押して盤面の状態を 1 世代進めるか, マウスカーソルを使って盤面内をクリックまたはドラッグした時に押せるようになる. 最大で 32 の状態を記憶

しておき,33 回以上盤面の状態が変化した場合は直近の 32 個の盤面の状態を記憶しておく. 盤面の履歴がこれ以上ない状態まで巻き戻されると再び無効な状態に戻る. したがって,33 回盤面の状態が変化した際は,32 回 undo を押した段階で undo が押せなくなってしまう. また, 盤面の状態の記憶は 1 セルの状態が変化すると実行されるので,3 セル分ドラッグ操作をした場合はその操作で 3 つの状態が記憶されることになる. 以下の図 8 は RUN ボタンを押したときに Undo ボタンが押せなくなっている様子である.

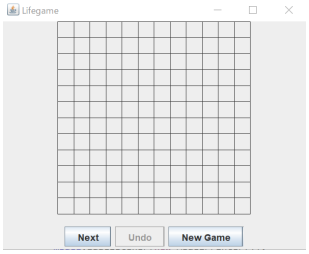

図 9: Undo ボタンが押せない様子

また, 以下の図 9,10 は undo ボタンを押して盤面の状態を一世代前に戻したときの例である.

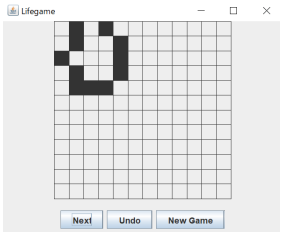

図 10: undo ボタンを押す前の盤面

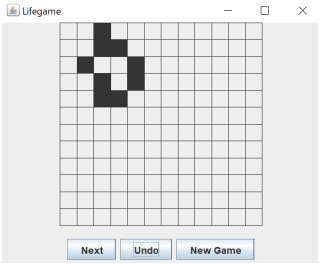

図 11: undo ボタンを押した後の盤面

1.2.3 newgame ボタン

newgame ボタンは最初から押せる状態になっており, ボタンを押すごとに新しく初期化されている盤面を新しいウィンドウで開く. 新しく作られたウィンドウはもともとあったウィンドウが作成された位置に作成される. ゲームはそれぞれ独立しており, 片方の盤面の状態を変化させてももう片方の盤面には影響されない. 以下の図 11 は newgame ボタンを押して新しくウィンドウを開き, 片方の盤面にだけマウスカーソルによって状態を変化させた様子である.

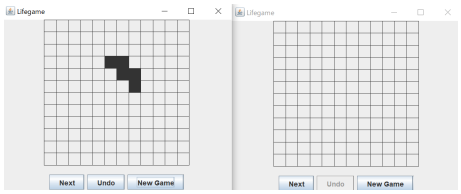

図 12: newgame ボタンを押して片方の盤面だけ変化させた様子

1.3　クリック, ドラッグしたときの処理

これから記述するクリック操作はマウスカーソルを動かさないで左クリックをすると実行できる. この操作はボタンが押し込まれた瞬間に実行される. また, ドラッグ操作は左クリックを押し込みながらマウスカーソルを移動させると実行できる. これもクリック操作と同様に, ボタンが押し込まれた瞬間に実行される.

1.3.1　盤面内のクリック, ドラッグ

上述のクリック操作により, 現在の盤面の状態は反転する. 例えば, もともと死んでいる状態のセルをクリックすると生きている状態のクリックに変更し, 生きている状態のセルをクリックすると死んでいる状態に変更する. 以下の図 12,13 は実行例である. この例では (4,3) のセルの状態を反転させている.

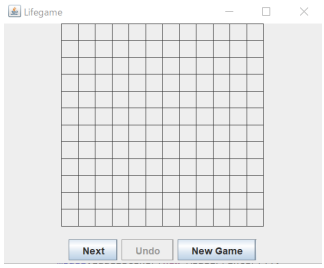

図 13: クリック前の盤面

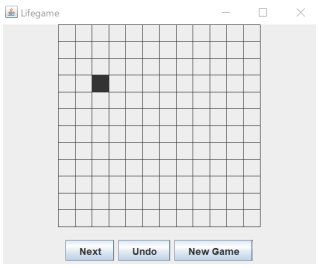

図 14: クリックした後の盤面

上述のドラッグ操作により, マウスカーソルがドラッグを開始したセル以外のセルに侵入した直後にそのセルの状態が変更される. ただし, 同じセル内を移動するだけだと結果としてはクリック操作と同じように押し込んだセルの状態を変更しただけとなる. 以下の図 14,15 はドラッグをした時の動作の例である.

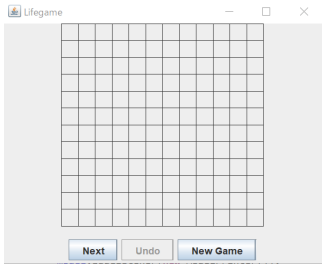

図 15: ドラッグ前の盤面

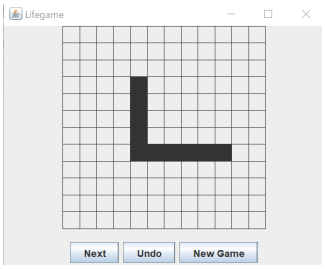

図 16: ドラッグ後の盤面

1.3.2　ドラッグ中にマウスカーソルが盤面以外に移動したときの処理

盤面外にマウスカーソルを移動させた場合には何も変化が起こらないという仕様にした. また, 図 16 のようにドラッグしながらマウスカーソルを移動させた場合, 侵入された盤面の状態のみを変更するので右の図 17 のように盤面が変更される.

1.3.3　盤面以外をクリックしたときの処理

盤面以外の場所をクリックした際はドラッグの時と同様に何も起こらないという仕様にした.

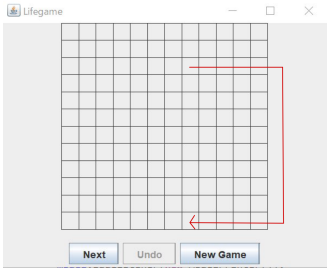

図 17: マウスカーソルの動かし方

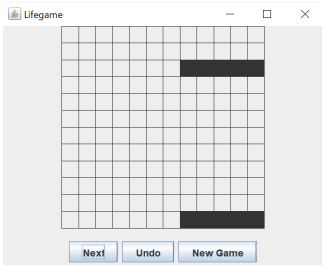

図 18: ドラッグ後の盤面

1.3.4 ドラッグ, クリック時の undo ボタンの巻き戻し

1.1.2 で記述した undo ボタンはクリック, ドラッグ操作による状態の変更も記憶して巻き戻すという仕様とした. ドラッグ操作に関しては, 各セル一つずつが状態変化するたびに新しく盤面が記憶される.

2 クラスと機能の対応表

今回作成したライフゲームのプログラムのクラスと機能の対応表は以下のようになる.

クラス名	機能
Main クラス	
BoardModel クラス	
ButtonListener クラス	
Boardlistener インターフェース	
ModelPrinter クラス	
BoardView クラス	

表 1: クラスと機能の対応表

3 各機能実装方法

3.1 状態の更新に連動して画面表示の変更

状態の更新に関する処理は以下の Listing1 から Listing3 のコードで示した部分に記述した.

3.1.1 BoardModel クラス内の処理

next ボタンやマウスカーソルによる盤面の状態の変化が起こるたびに changeCellState メソッドが呼び出され, 後述の cells の状態が更新される. BoardModel クラス内の changeCellState メソッドのコードは以下のようになる.

Listing 1: BoardModel クラス内の該当部のコード

```
1 public void changeCellState(int x,int y) {
2   undoButton.setEnabled(true);
3   if(cells[y][x]==true) {
4     cells[y][x]=false;
```

```
5         }else {
6       cells[y][x]=true;
7     }
8     boolean[][] currentBoard = new boolean[rows][cols];
9     for (int i = 0; i < rows; i++) {
10      System.arraycopy(cells[i], 0, currentBoard[i], 0, cols);
11    }
12    if(counter!=32) {
13      counter++;
14      History.add(currentBoard);
15    }else {
16      History.remove(0);
17      History.add(currentBoard);
18    }
19    this.fireUpdate();
20    }
21
22    private void fireUpdate() {
23      for(BoardListener listener:listeners) {
24        listener.updated(this);
25      }
26    }
```

この処理を行うにあたって使用したメンバは以下のとおりである.

変数名	変数があらわす内容
cells	盤面の状態を表す boolean 型の二次元配列
N	盤面の縦のマス数
width	パネルの横幅を paint メソッド内の getWidth メソッドによって格納する int 型の変数
height	パネルの縦幅を paint メソッド内の getHeight メソッドによって格納する int 型の変数
masu	盤面に描写する正方形のマスの一辺の長さを格納する int 型の変数

表 2: BoardModel クラスのメンバとその内容

また,changeCellState クラスの内部変数は以下のようになる.

変数名	変数があらわす内容
x	int 型の変数
y	int 型の変数
currentBoard	現在の盤面の状態をコピーするための boolean 型の二次元配列

表 3: changeCellState クラスの内部変数

3.1.2

Listing 2: コード

```
1   public interface BoardListener {
2     public void updated(BoardModel m);
3   }
```

- 行目の処理
- 行目の処理
- 行目の処理
-

3.1.3

Listing 3: コード

```
1  public void updated(BoardModel model) {
2                  this.repaint();
3          }
```

3.2 巻き戻しのための盤面の記憶

巻き戻しのための盤面の記憶に関する処理はクラスのメソッドに記述した．この処理を行うにあたって使用したメンバは以下のとおりである．

変数名	変数があらわす内容
M	盤面の横のマス数
N	盤面の縦のマス数
width	パネルの横幅を paint メソッド内の getWidth メソッドによって格納する int 型の変数
height	パネルの縦幅を paint メソッド内の getHeight メソッドによって格納する int 型の変数
masu	盤面に描写する正方形のマスの一辺の長さを格納する int 型の変数

表 4: メンバとその内容

3.3 盤面の描画

盤面の描画に関する処理は BoardView クラスの paint メソッドに記述した．この処理を行うにあたって使用したメンバは以下のとおりである．

変数名	変数があらわす内容
M	盤面の横のマス数
N	盤面の縦のマス数
width	パネルの横幅を paint メソッド内の getWidth メソッドによって格納する int 型の変数
height	パネルの縦幅を paint メソッド内の getHeight メソッドによって格納する int 型の変数
masu	盤面に描写する正方形のマスの一辺の長さを格納する int 型の変数

表 5: メンバとその内容

3.4 マウスカーソルによる盤面の状態の変更

マウスカーソルによる盤面の状態の変更に関する処理は BoardView クラスに記述した. 以下にクリックに関する処理とドラッグに関する処理の実装方法を記述する. この処理を行うにあたって使用したメンバは以下のとおりである.

変数名	変数があらわす内容
width	パネルの横幅を paint メソッド内の getWidth メソッドによって格納する int 型の変数
height	パネルの縦幅を paint メソッド内の getHeight メソッドによって格納する int 型の変数
masu	盤面に描写する正方形のマスの一辺の長さを格納する int 型の変数
M	盤面の横のマス数
N	盤面の縦のマス数
StateX	マウスカーソルが現在存在するセルの左上の頂点の x 座標を格納する int 型の変数
StateY	マウスカーソルが現在存在するセルの左上の頂点の y 座標を格納する int 型の変数

表 6: メンバとその内容

また,masu の計算は 2.3 で記述した通りである.

クリック時の処理

クリック時の処理は mousepressed メソッド内に記述した. mousepressed メソッド内で使用する内部変数は以下のとおりである. クリック操作時の具体的な処理内容は以下のような流れで実行される.

変数名	変数があらわす内容
x	マウスカーソルがある場所の x 座標を格納する int 型の変数
y	マウスカーソルがある場所の y 座標を格納する int 型の変数
i	for 文を使う際のカウンタ
j	for 文を使う際のカウンタ

表 7: 内部変数名とその内容

1. x,y の宣言及び初期化をする.x,y は getcol 関数により取得される.
2. マウスカーソルが盤面内に存在するかどうかを判定し, 存在すれば 3 以降の処理を実行する. 存在しなければクリックの処理を return 文により終了する
3. for 文を使って今現在のマウスカーソルが存在する座標のセルの y 座標を特定. このとき, 特定するための条件式 $y >= i * masu \&\& y < (i+1) * masu$ は, カウンタを表す変数 i によって盤面のセルのどの行に y が存在するかを判定している.
4. 3 で y がどの行にあるかを特定した後, 同様にして x がどの列にあるのかを判定する. 条件式は $x >= \frac{(wodth - masu*M)}{2} + masu * j \&\& x < \frac{width - masu*M}{2} + masu * (j+1)$ となる.2 の条件式との違いは, 盤面を描写し始める位置が異なるためである.
5. StateX と StateY を 3 と 4 で条件分岐をした時の i と j を使ってマウスカーソルが存在するセルの左上の頂点の座標を格納する. これは, 後のドラッグ操作ための処理であるため後に記述する. そのあと break 文を二回使って for 文から抜ける.
6. 特定した i と j を changeCellState メソッドの引数として使用しセルの状態を変える. そして paint メソッドを使って盤面を描写する.

ドラッグ時の処理

ドラッグ時の処理は mouseDragged メソッド内に記述した. mouseDragged メソッド内で使用する内部変数は以下のとおりである. クリック操作時の具体的な処理内容は以下のような流れで実行される.

変数名	変数があらわす内容
x	マウスカーソルがある場所の x 座標を格納する int 型の変数
y	マウスカーソルがある場所の y 座標を格納する int 型の変数
masu_x	0 で初期化されており, マウスカーソルが現在存在するセルの左上の頂点の x 座標を格納する int 型の変数
masu_y	0 で初期化されており, マウスカーソルが現在存在するセルの左上の頂点の y 座標を格納する int 型の変数
i	for 文を使う際のカウンタ
j	for 文を使う際のカウンタ

表 8: 内部変数名とその内容

1. クリック時の処理の 1 から 4 と同様の処理が行われる. また, 最初に masu_x と masu_y を 0 で初期化する.
2. StateX と StateY ではなく,masu_x と masu_y に特定した i,j を使って x と y が存在するセルの左上の頂点の座標を格納する.
3. masu_x と masu_y がそれぞれ StateX と StateY と一致しているかを判定する. 即ち直前にマウスカーソルがあったセルと今現在マウスカーソルが存在しているセルが一致しているかどうかを判定している. 一致していれば何も処理をせずに終了し, 一致していなければ以下の 4 の処理を行う.
4. for 文を二回使用してクリック時の StateX や StateY を求めた時と同じように changeCellState メソッドと repaint メソッドを使って変更された盤面を表示する.

ドラッグ操作では, クリック操作とは違って現在のセルの位置を格納する変数として masau_X と masu_y を使用した. これは, ドラッグ操作をする際に同じマス中の移動の際に盤面の状態が何度も変更されて最初の状態が維持されないという問題が発生したからである b. これを解決するために直前の状態を保存しておくことで同じセル内を移動するときには盤面の状態を変化させないようにした.

3.5 新しいウィンドウを開く

新しいウィンドウを開くことに関する処理はクラスのメソッドに記述した. この処理を行うにあたって使用したメンバは以下のとおりである.

変数名	変数があらわす内容
M	盤面の横のマス数
N	盤面の縦のマス数
width	パネルの横幅を paint メソッド内の getWidth メソッドによって格納する int 型の変数
height	パネルの縦幅を paint メソッド内の getHeight メソッドによって格納する int 型の変数
masu	盤面に描写する正方形のマスの一辺の長さを格納する int 型の変数

表 9: メンバとその内容

3.6　拡張機能

3.6.1　実現方法

4　プログラミングの講義・演習で学習したこと

4.1　演習前から知っていたこと

私は大学生になるまでプログラミングの経験がなく, 大学のプログラミングの講義等でしか触れる機会が無かった. そのため, 今回の講義・演習で初めてオブジェクト指向言語について学習した. 以前からオブジェクト指向プログラミングというものの存在は認知していたが, プログラミング歴の浅い自分にとっては難易度が高く思えていた.