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25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. 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. また,(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.

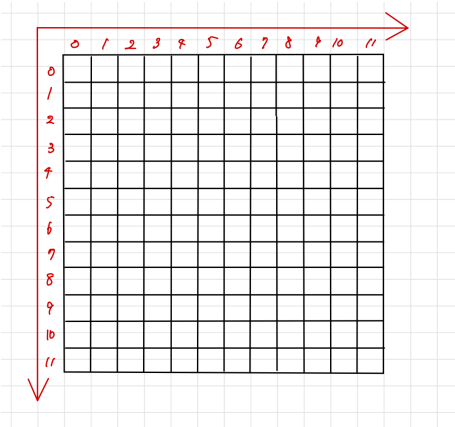

図 1: 座標の仕様

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. 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 200, 横 300 ピクセルである. 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.

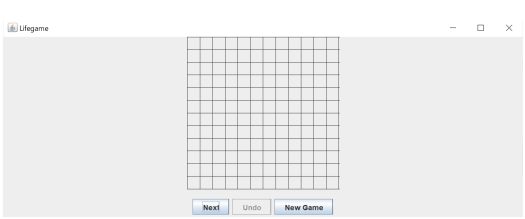

図 2: 極端に横に大きくしたときの盤面

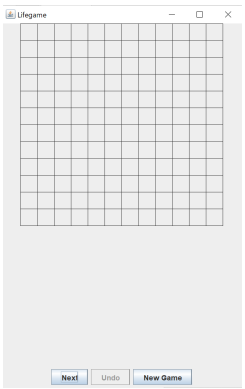

図 3: 極端に縦に大きくしたときの盤面

上の図のようにウィンドウのサイズが変更されてもセルの形は常に正方形を保つ. 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.

1.1.2 盤面のサイズ変更

ユーザは盤面のセル数を, 該当部のコードを書き換えることで指定することができる. 以下の図 4 は Main クラス内のコードの一部である.N は縦のセルの個数,M は横のセルの個数を表す int 型の内部変数である.

```
int M=18;
int N=12;
if(M<=0||M>282||N<=0||N>123) {
    return;
}
model = new BoardModel(M,N,Undo);
```

図 4: 該当部分のコード

また, 変更できる盤面のセル数の最大値は横が 282 個, 縦が 123 個である. また, 最小値は縦も横も 1 個である. この範囲外の数値を N と M に指定すると”Error”と出力してプログラムを終了する. 以下の図 5,6 は RUN ボタンを押して出力した 12×12 と 18×12 の盤面である.

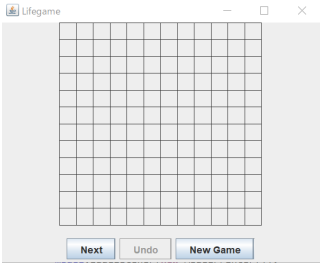

図 5: 12×12 の盤面

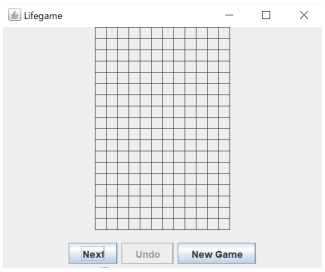

図 6: 18×12 の盤面

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. 具体的には, 生きているセルは周りに 2 または 3 個の生きているセルがあるとき次の世代でも生きている状態が続き, それ以外の個数であれば次の世代では死んでいる状態となる. また, 死んでいるセルは周りに 3 個の生きているセルがあるときに次の世代では生きている状態となるがそれ以外の個数であれば次の世代でも死んでいる状態を維持する. また, 隣に存在しないセルがある盤面の端のセルに関しては, その存在しないセルを死んでいるセルとして考えた. 以下の図 7,8 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.

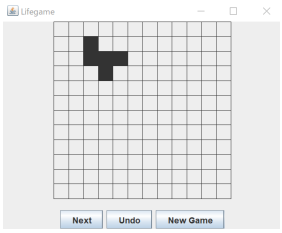

図 7: next ボタンを押す前の盤面

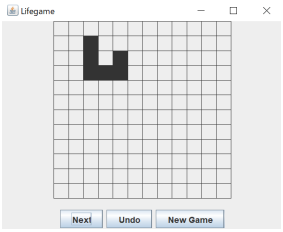

図 8: next ボタンを押した後の盤面

1.2.2 undo ボタン

undo ボタンは最初は色が薄くなって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 個の状態を記憶することができ,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. また,undo が押されると細心の盤面の状態が履歴から消える. 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 9 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.

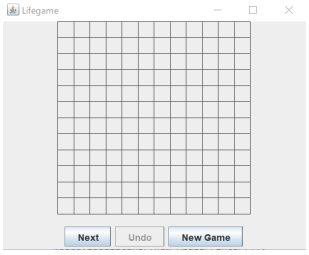

図 9: Undo ボタンが押せない様子

また, 以下の図 10,11 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.

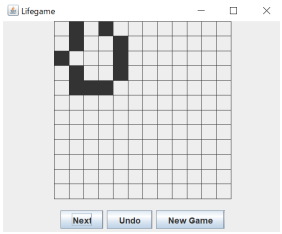

図 10: undo ボタンを押す前の盤面

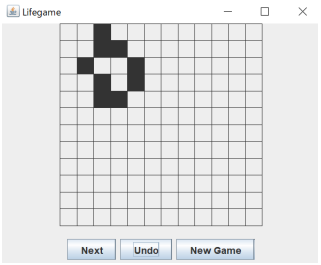

図 11: undo ボタンを押した後の盤面

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 12 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.

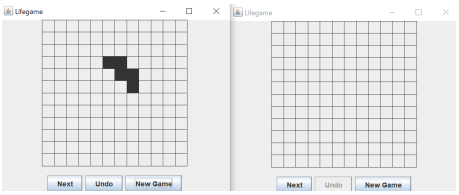

図 12: newgame ボタンを押して片方の盤面だけ変化させた様子

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 13,14 は実行例である. この例では (3,4) のセルの状態を反転させている.

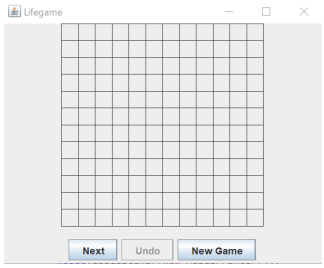

図 13: クリック前の盤面

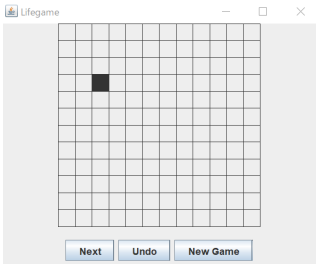

図 14: クリックした後の盤面

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 15,16 はドラッグをした時の動作の例である.

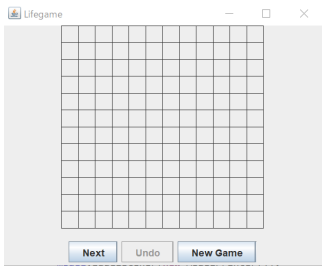

図 15: ドラッグ前の盤面

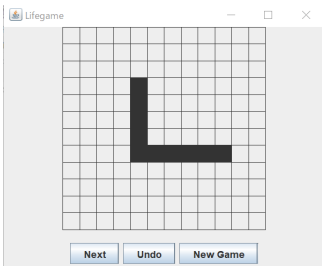

図 16: ドラッグ後の盤面

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 17 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 18 のように盤面が変更される.

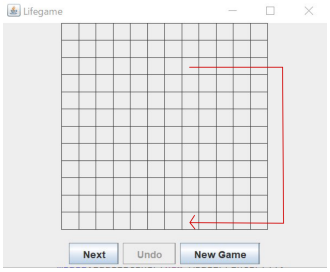

図 17: マウスカーソルの動かし方

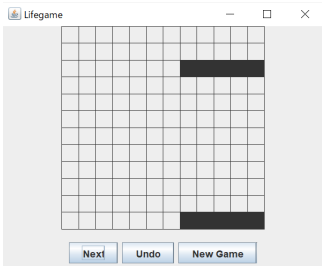

図 18: ドラッグ後の盤面

1.3.3　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.

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表 1 のようになる.

クラス名	機能
Main クラス	パネル内の情報の初期設定
BoardModel クラス	盤面の世代更新, 巻き戻し処理, 盤面の状態変更
Button クラス	パネルのボタンが押された時の処理の分類
BoardView クラス	マウスカーソルによる処理及び盤面の描画

表 1: クラスと機能の対応表

2.1　役割分担の方針及びその達成度

各クラスの役割分担の方針としては, 以下の通りになった.

- Main クラス
  パネルの情報の初期化をする.
- BoardModel
  盤面の状態を格納している配列の値の操作を記述する.

- Button
  ボタンが押された時の処理内容を記述する.

- BoardView
  マウスカーソルによる状態の変更を記述する.

また, その達成度は以下の通りになった.

- Main クラス
  このクラスは方針で挙げた内容を実装することができた.

- BoardModel
  このクラスは方針で挙げた配列の値の処理のほかに盤面の状態を記憶する処理を行った. 方針とは少し違った内容を含めてしまったがもともとの目標であった機能を実装することはできた.

- Button
  このクラスは方針で挙げた内容を実装することができた.

- BoardView
  このクラスは方針で挙げた内容を実装することができた.

3 各機能実装方法

3.1 画面の表示項目

今回のライフゲームでの画面の表示項目とその実装方法は以下のとおりである.

- 盤面

- next,undo,newgame ボタン

3.2 状態の更新に連動して画面表示の更新方法

今回のライフゲームでの画面表示が変更されるのは以下のような場合である.

- next ボタンや undo ボタンによる盤面の更新及び巻き戻し

- マウスカーソルによる盤面の状態の変更

これらの処理が行われた後に盤面が再描写されるための処理について記述する.
以下の Listing1 から Listing4 のコードはこの処理を実装するためにそれぞれのクラスから必要な個所のみを抜粋したコードである.

Listing 1: BoardListener インターフェース

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

Listing 2: Main クラス

```
1    model.addListener(new BoardView(model));
```

Listing 3: BoardView クラス

```
1     public BoardView(BoardModel model) {
2                 this.model = model;
3                 this.addMouseListener(this);
4                 this.addMouseMotionListener(this);
5                 model.addListener(this);
6         }
7     @Override
8         public void updated(BoardModel model) {
9                 this.repaint();
10        }
```

Listing 4: BoardModel クラス

```
1     private ArrayList<BoardListener> listeners;
2
3     public BoardModel(int c,int r,JButton undoButton) {
4                 listeners = new ArrayList<BoardListener>();
5         }
6
7     public void changeCellState(int x,int y) {
8                 this.fireUpdate();
9         }
10
11    public void addListener(BoardListener listener) {
12                listeners.add(listener);
13        }
14
15        private void fireUpdate() {
16                for(BoardListener listener:listeners) {
17                        listener.updated(this);
18                }
19        }
```

listing1 の 2 行目において,BoardListener インターフェースには updated クラスが呼び出されている. この updated クラスの処理内容は listing3 の 8 から 10 行目のように, 上で挙げた盤面が更新されるための処理を実装するクラス内に記述されている. listing2 のコードにより,BoardListener インターフェースを implement したクラスのオブジェクトを listing4 の 10 から 12 行目の addListener クラスで可変長配列 listeners に登録している. これを行うことで, 盤面を変更する各操作が実行されるたびに listing4 の 9 から 12 行目の ChangeCellState メソッドが呼び出されて盤面を変更し, その後で 18 から 21 行目の fireupdated メソッドが呼び出されて repaint が実行される. これにより, すべての盤面の変更が起こったときに listeners に登録されたオブジェクトを使って BoardListener を介して盤面の再描画が実行される.

3.3 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 BoardModel クラスに記述した. 盤面の情報は可変長配列を持つリスト型である Arraylist を使用し, 変数名は History とした.

3.3.1 next による盤面の状態の変更の記憶

以下の Listing5 は BoardModel クラスの next メソッドの一部である.

Listing 5: next メソッドの一部

```
1     boolean[][] currentBoard = new boolean[rows][cols];
2     for (int i = 0; i < rows; i++) {
3         System.arraycopy(cells[i], 0, currentBoard[i], 0, cols);
4     }
5     if(counter!=32) {
6         counter++;
7         History.add(copiedcells);
8     }else {
9         History.remove(0);
10        History.add(copiedcells);
11    }
```

counter は History に記憶されている状態の数を表す int 型の変数で 0 に初期化されている.counter はメンバであるため counter の値はほかのメソッドにも共有される. 記憶されている状態の数が 32 個以下なら counter の値を 1 インクリメントしてから add メソッドを使って History に状態を記憶し,32 個なら History に記憶されている一番古い状態を削除してから add メソッドを使って現在の状態を新しく記憶している. また,copiedcells は lifegame の仕様に基づいて一世代進めたあとの盤面の状態を表す二次元配列である. これは現在の盤面の状態を表す二次元配列 cells のすべての要素をコピーしたものである.copiedcells を使用している理由としては, 次の世代へ進めたときに各セルが生きているかはセルの周りに生きているセルが何個存在するかで決定されるので全ての cells の要素に参照している途中で cells の状態を変えてしまうと周りに存在する生きているセルの数が変化してしまうためである.

3.3.2 undo による盤面の状態の変更の記憶

以下の Listing6 は BoardModel クラスの undo メソッドの一部である.

Listing 6: undo メソッドの一部

```
1     History.remove(counter);
2     counter--;
3     cells = History.get(counter);
```

まず一番最新の状態を History から削除し, そのあとに counter を 1 デクリメントする. これにより undo が押された際に記憶されていた状態が 1 つ減少する. その後現在の状態を表す二次元配列 cells にデクリメントした後の counter がさす History の最新の状態, 即ち undo を押したときの直前の状態を格納する.

3.3.3 マウスカーソルによる盤面の状態の変更の記憶

以下の Listing7 は BoardModel クラスの changeCellState メソッドの一部である.

Listing 7: changeCellState メソッドの一部

```
1     boolean[][] currentBoard = new boolean[rows][cols];
2     for (int i = 0; i < rows; i++) {
3         System.arraycopy(cells[i], 0, currentBoard[i], 0, cols);
4     }
5     if(counter!=32) {
6         counter++;
7         History.add(currentBoard);
8     }else {
```

```
9           History.remove(0);
10          History.add(currentBoard);
11      }
```

マウスカーソルによる盤面の状態はどのセルの状態を変更すればよいかを特定してから changeCellState を呼び出して実際に盤面の状態を格納している cells を変更する. 二次元配列である currentBoard を用意して変更後の盤面の状態を 2 から 4 行目でコピーする. その後 counter の値が 32 以下なら counter の値を 1 インクリメントしてから add メソッドを使って History に状態を記憶し,32 個なら History に記憶されている一番古い状態を削除してから add メソッドを使って現在の状態を新しく記憶している. add した配列が cells ではなく currentBoard である理由は,cells を格納してしまうと参照の追加になり,cells が変更されると History に追加されているほかのオブジェクトも同じように変更されてしまうためである. ゆえに一時的に cell の内容を別の二次元配列にコピーして参照の追加となることを避けた.

3.4 undo による巻き戻しが可能かどうかの判定

undo は盤面を生成したとき Main メソッド内で setEnabled メソッドを使って押せないように初期化されている.undo による巻き戻しが可能となるのは仕様でも説明した通り next ボタンによる世代の更新かマウスカーソルによる盤面の状態の変更が行われた時である.

- next ボタンによって世代更新が行われた時,3.2.1 で記述したように counter の値が更新された回数だけインクリメントされる. これを利用して BoardModel メソッド内に isUndoable メソッドを作成した. このメソッドは counter の値が 0 より大きいときは true を,0 のときは false を返す. 以下の Listing8 のコードは Button クラス内の next ボタンと undo ボタンが押された時に実行される処理である.

Listing 8: next ボタン及び undo ボタンを押したときに実行される処理

```
1       case 1:
2           boardModel.next();
3           undoButton.setEnabled(true);
4           break;
5       case 2:
6           if(boardModel.isUndoable()) {
7               boardModel.undo();
8               if(!boardModel.isUndoable()) {
9                   undoButton.setEnabled(false);
10              }
11          }
12          break;
```

1 から 4 行目のように,next ボタンが押された時には next メソッドを呼び出して盤面の世代を更新してから setEnabled メソッドで undo ボタンを押せるようにしている. また,6 から 11 行目のように isUndoable の返り値が true の時は undo メソッドを呼び出して盤面の世代を一つ戻した後に再び isUndoable メソッドを呼び出して返り値が false であれば setEnabled メソッドで undo ボタンを押せなくしている. これは,undo メソッドによって世代を戻したときに History が空になったことをすぐに確認するためである. これを実装することで Undo ボタンが履歴がなくなった時に押せなくなることを実現できる.

- マウスカーソルによって盤面に変更が加えられたときは仕様で述べた通り, セルの状態が一つでも変わるとそれが一つの盤面の状態として記憶される.3.2.3 で記述した通り, マウスカーソルで状態が変更

されると changeCellState メソッドが呼び出されるのでこのメソッドが呼び出された時に setEnabled メソッドで undo ボタンを押せるようにすれば巻き戻しを可能にすることが実現できる.

3.5 セルの境界線の位置を計算する方法

以下の表 2 は計算するために使用した変数名と表す内容である.

変数名	変数があらわす内容
M	盤面の横のセル数
N	盤面の縦のセル数
width	パネルの横幅
height	パネルの縦幅
masu	盤面に描写する正方形のマスの一辺の長さ

表 2: 計算式中の変数名とその内容

盤面のセルの境界線の位置は以下のような手順で盤面の境界線の位置は計算される.

1. 現在の width と height から,masu を決定する. 具体的には以下のような不等式が成り立つかどうかについて考える.

$$\frac{height \times \frac{99}{100}}{N} > \frac{width}{M+2} \tag{1}$$

左辺は height の 99%の大きさを N 等分した時の長さを調べている. これは,height の 99%の長さを考えないと盤面の一番下の横線が表示されないことがあるという問題を解決するための処理である. 右辺は width を M+2 等分したときの長さを調べている. これは, パネルの中央に盤面を配置するための処理である.(1) が成り立った時はパネルが縦に長い状態であると判断して

$$masu = \frac{width}{M+2} \tag{2}$$

とし,(1) が成り立たなかったときはパネルが横に長い状態であると判断して

$$masu = \frac{height \times \frac{99}{100}}{N} \tag{3}$$

となるようにした.

2. 次に, 盤面の開始位置を決定する. 縦軸の開始位置は y 座標が 0 からである.x 座標の開始位置はパネルの中央に盤面を配置するために, 以下のような数式の値を開始位置とする.

$$\frac{width - masu \times M}{2} \tag{4}$$

この数式はパネルの横幅から盤面を表示している部分の長さを引いた長さを二等分した値になる. これにより, 盤面を描写していない部分の横幅が同じ長さとなっている.

3. ユーザが指定したセル数分だけ縦軸と横軸を描画する. それぞれのセルの一片の長さは (2) または (3) の値である.

3.6　マウスカーソルの座標からセルの座標を計算する方法

以下の表 3 は計算するために使用した変数名と表す内容である.

変数名	変数があらわす内容
M	盤面の横のセル数
N	盤面の縦のセル数
width	パネルの横幅
height	パネルの縦幅
masu	盤面に描写する正方形のマスの一辺の長さ
x	マウスカーソルがある場所の x 座標
y	マウスカーソルがある場所の y 座標
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
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
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
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
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 3: 計算式中の変数名とその内容

3.4 の表 2 の変数と同じ変数名のものは値も同じである. 以下のような流れでマウスカーソルがある位置のセルの座標はクリックとドラッグで異なる方法で計算される.

1. 盤面外にマウスカーソルがある状態でクリックやドラッグがされると何もしないようにするために x と y が以下の条件式を満たしているか確認する.

$$x < \frac{width - masu \times M}{2} \quad or \quad x > \frac{width + masu \times M}{2} \quad or \quad y > masu \times (N) \tag{5}$$

これらの 3 つの不等式の一番左の不等式はマウスカーソルの x 座標が盤面よりも左側にあるとき真になる. 真ん中はマウスカーソルの x 座標が盤面よりも右側にあるとき真になる. 一番右の不等式はマウスカーソルの y 座標が盤面よりも下側にある時に真になる. そしてこれら 3 つの不等式は全て or としているのでこれらのうちの 1 つでも真になれば盤面外にマウスカーソルがあるという判定になる.

2. 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 座標を特定する. このとき, 特定するための条件式は以下のようになる.

$$y \geqq i * masu \quad and \quad y < (i+1) * masu \tag{6}$$

この条件式が真になった時の i の値が求めるセルの y 座標となる. このとき,and の左側の不等式にイコールが入っていることから, セルの境界線の上側はそのセル内に含まれるという判定になる.

3. y 座標を特定してから再び for 分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 x 座標を特定する. このとき, 特定するための条件式は以下のようになる.

$$x \geqq \frac{width - masu \times M}{2} + masu * j \quad and \quad x < \frac{width - masu \times M}{2} + masu * (j+1) \tag{7}$$

この条件式が真になった時の j の値が求めるセルの x 座標となる. このとき,and の左側の不等式にイコールが入っていることから, セルの境界線の左側はそのセル内に含まれるという判定になる.

3.7　新しいウィンドウを開く

以下の Listing9 は Button クラスの ActionPerformed メソッドの一部で,listing は Main メソッドのコードの一部である.

Listing 9: Button クラスの一部

```
1     case 3:
2     SwingUtilities.invokeLater(new Main());
3     break;
```

Listing 10: Button クラスの一部

```
1     public static void main(String[] args) {
2           SwingUtilities.invokeLater(new Main());
3       }
```

Listing9 の 2 行目は空の盤面を作り新しいウィンドウを開くという処理を表しており, ゲームが開始された時に実行されてパネルが一つ生成される. それと同じ処理を Listing10 の 2 行目に記述した. この処理は Newgame ボタンが押された時に実行されるため,Newgame ボタンを押したときの処理を実現できた.

4　プログラミングの講義・演習で学習したこと

4.1　演習で試すまでに理解できなかったこと

私はこの授業を通して継承とインターフェースについて深く学ぶことができたと考えている.java の基本的な構文や, オブジェクトというものの存在などは講義資料や参考図書の教材を読むことで理解することができたが, この二つだけは実際にプログラムを書いて動かしてみないと理解することができなかった. 特に, インターフェースはライフゲームのプログラム中の AddListener で使用したが理解するのにかなり時間がかかってしまった.

4.2　同一のプログラムを現段階で書き直すときに変更する点

今回のプログラムでは BoardModel クラスで盤面の状態を表す配列の世代を進めたり戻したりする処理と, 盤面の状態の記憶を実現した. しかし, オブジェクト指向プログラミングの観点からなるべく処理内容は細分化したほうが今後の大人数での開発にも適していることから, 先ほど挙げた二つの処理は分割して別々のクラスに記述すべきだと考えた. 具体的には BoardModel クラスには盤面の状態を格納する配列の値を変更する処理をそのまま書いて, 新たに History クラスを作成してそこに盤面の記憶に関する処理を書くべきだと考えた.
また,BoardView クラスに関しても, マウスカーソルによる変更するセルの座標の取得と盤面の描写という二つの役割が同じクラス内に記述されているので同様の理由から別々のクラスに記述すべきだと考えた. 具体的には BoardView クラスには盤面の描写に関する処理をそのまま記述し, 新たに Mouse クラスを作成してドラッグやクリックが行われた時の処理を記述すればよいと考えた. したがって, 新たに作成するクラスの役割分担表は以下の表 4 ようになる.

クラス名	機能
Main クラス	パネル内の情報の初期設定
BoardModel クラス	盤面の世代更新, 巻き戻し処理, 盤面の状態変更
History クラス	盤面の状態の記憶
Button クラス	パネルのボタンが押された時の処理の分類
Boardlistener インターフェース	盤面の情報
BoardView クラス	盤面の描画
Mouse クラス	マウスカーソルによる処理

表 4: クラスと機能の対応表