1　プログラムの操作方法と機能

1.1　行数・列数の決定

プログラム起動時に、セルの行数を尋ねるクエスチョンダイアログメッセージが表示される。ユーザーは、指定された範囲内の整数を入力する。全角・半角どちらでも良い。この範囲は、多くの PC のディスプレイは 9:16 の比率であると思われるので、行数は 10 以上 90 以下、列数は 10 以上 160 以下とした。それ以外の入力がなされた場合は再び入力を要求する。取り消しボタンが押された場合は、プログラムを終了する。同様に、列数を尋ねるクエスチョンダイアログメッセージが表示される。この入力に基づいて盤面が作成される。

1.2　盤面の編集

1.2.1　クリック操作

盤面上をクリックすることで、セルの状態を反転させる。

1.2.2　ドラッグ操作

盤面上をドラッグすることで、カーソルが通過したセルの状態を反転させる。巻き戻しを行うと、まとめてドラッグ操作を行った場合でもセルは 1 つずつ戻る。

1.3　世代の更新・巻き戻し

本プログラムでは、ユーザーのクリック・ドラッグ、または paste 機能による編集操作が行われた状態を第 1 世代とし、以下で述べる next,jump,undo 機能によって盤面の状態を更新し、世代数を増減させる。

1.3.1　世代数の確認機能

ウィンドウ上部のラベルによって現在の世代数を確認できる。

1.3.2　巻き戻し可能な範囲

本プログラムでは、第 1 世代の盤面のみ記録し、undo 機能等によって盤面の状態を遡る際には、第 1 世代から更新処理を連続して行う。

例えば、第 n 世代の状態で undo 機能を実行する時、記憶された最新の盤面に n-2 回更新処理を行うことで第 n-1 世代の状態を求める。しかし世代数が大きすぎる場合は更新処理に時間がかかるため、世代数の最大値は 10000 としている。

ただし、巻き戻し可能な回数は 32 回までに制限されている。

1.3.3　next 機能

next 機能では、盤面を次の世代に進める。next ボタンを押すと、世代数を 1 増加させ盤面を更新する。世代数が 10000 である場合は next 機能は利用できない。

1.3.4 jump 機能

jump 機能では、指定した世代の状態に盤面を更新する。現在の世代数よりも小さい世代に更新することも可能である。

jump ボタンを押すと、世代数を尋ねるクエスチョンダイアログメッセージが表示される。ユーザーは、1 以上 10000 以下の整数を入力する。それ以外の入力がなされた場合は再び入力を要求する。取り消しボタンが押された場合は、盤面の編集画面に戻る。

正しい値が入力されると、入力された世代数の状態に盤面を更新する。ただし、1 が入力された場合は第 1 世代の状態に更新し、jump 機能以外で盤面に変更が加えられるまで jump 機能が利用できなくなる。

1.3.5 undo 機能

undo 機能では、直前に行った操作を取り消す。undo ボタンを押すと、世代数が 2 以上の場合は世代数を 1 減じ、世代数が 1 である場合はマウス操作やペーストを行う前の状態に戻す。

jump 機能を行っていた場合でも、その前に戻るのではなく世代を 1 つ戻す。これは、jump 機能が next 処理を連続して実行する機能であるからである。但し、直前に jump 機能で第 1 世代を指定していた場合は盤面を変化させない。

undo 機能は最大で 32 回まで実行できる。

1.4 セルの選択・コピー・ペースト

1.4.1 セルの選択

シフトキーを押しながら盤面上でマウスをクリック・ドラッグすることで複数のセルを選択状態にできる。選択状態では、死んでいるセルはシアン、生きているセルは青色になる。

世代数が表示されている部分より下、編集ボタンより上の領域をクリックすることで選択状態を解除する。

1.4.2 copy 機能

選択状態のセルがある状態で Edit メニューの Copy を押すことで選択したセルをコピーする。

1.4.3 paste 機能

コピーされたセルがあるとき、ペースト機能が利用できる。Edit メニューの Paste を押すことで、選択されたセルの左上からコピーされたセルをペーストする。盤面又は選択範囲をはみ出す部分についてはコピーされない。コピーされたセルがない場合は利用できない。

paste 後に undo 機能を実行すると、paste を行う前に戻る。

1.5 スクリーンショット機能・新規ウィンドウ生成機能

1.5.1 スクリーンショット機能

File メニューの Export image を押すことで、ファイルの保存場所を選びファイル名を入力するダイアログが開く。ファイル名を入力し保存ボタンを押すと、ウィンドウのスクリーンショットが png 形式で保存される。

1.5.2　新規ウィンドウ生成機能

File メニューの New Game を押すことで、新規にライフゲームのウィンドウを作成する。

プログラム起動時と同様、行数と列数をダイアログから入力する。ここで取り消し処理を行うと、新規ウィンドウは生成されない。

2　各クラスの役割

2.1　クラス Main

Main クラスでは、初めに行数と列数の入力をダイアログから受け付ける。その後、各種の GUI パーツをインスタンス化して配置する。

また、Main クラスは正整数の入力を受け付けるダイアログを表示する static メソッドを持つ。

2.2　クラス BoardModel

クラス BoardModel では、内部的な盤面の作成と編集・更新、すなわち boolean 型の二次元配列を作成して直接アクセスする役割を担う。また、盤面の変更が起こると BoardListener インターフェースに通知する。

2.3　クラス BoardView

クラス BoardView では、盤面の GUI 表示や、copy、paste を含む GUI 上での操作をもとにクラス BoardModel 等のメソッドを呼び出して編集を行う。

2.4　クラス ButtonActionListener

クラス BottonActionListener では、ボタンやメニューアイテムがクリックされたことを検知し、それに対応する操作を行うメソッドを呼び出す。

2.5　クラス Buttons

クラス Buttons では、Main クラス内で作成した GUI パーツのボタンなどの参照をクラス BoardView に引き渡す。これにより BoardView 内で、盤面の状況に合わせてボタンの有効・無効が切り替えられるようになる。

2.6　クラス ModelPrinter

クラス ModelPrinter はデバッグ用のクラスで、CUI 上で盤面の状態を出力する。

3　プログラムの各機能の実現方法の詳細

3.1　盤面の状態の更新に連動して更新される画面の表示項目と、その更新方法

盤面の状態に応じて、以下の項目が変化する。

- Undo ボタンが押せるか・押せないか
- Next ボタンが押せるか・押せないか
- Jump ボタンが押せるか・押せないか
- Paste メニューが押せるか・押せないか
- 世代数の表示

盤面の状態が変化すると、インターフェース BoardListener を実装したクラスが持つメソッド updated が実行される。

上記の項目の更新は、GUI 表示を担うクラス BoardView にインターフェース BoardListener を実装し、そのメソッド updated 内で、各ボタンを利用不可にするかを判定し、またラベルに現在の世代数をセットすることで実現している。

以降で各ボタン・メニューの有効・無効を判定する方法について述べる。

3.1.1　Undo ボタン

クラス BoardModel のメソッド Undoable で Undo ボタンの状態を判定する。盤面を保存するリスト history のサイズが 0 であるか、または Undo 可能回数 limitUndo が 0 以下である時、false を返し、それ以外の場合は true を返す。

3.1.2　Jump ボタン

クラス BoardModel のメソッド isJumpable で Jump ボタンの状態を判定する。盤面を保存するリスト history のサイズが 0 であるか、または Jump 可能であるかを表す limitJump が true であれば、false を返す。それ以外の場合は true を返す。

limitJump は、盤面が変更される度に呼び出されるメソッド fireUpdate 内で変更する。fireUpdate 内で、まず limitUndo に false を代入する。そして doneJump1 が true の時、すなわち Jump 機能で第一世代が指定された直後であれば limitJump を true にする。その後 doneJump1 に false を代入する。

3.1.3　Next ボタン

クラス BoardModel のメソッド isNextable で現在の世代数が 10000 以上であれば false を返し、それ以外の場合は true を返す。

3.1.4　Paste メニュー

クラス BoardView のメソッド isPastable で、コピー機能を実行した際に用いられるフィールド clipBoard が null であれば false を返し、それ以外の場合は true を返す。

3.2 巻き戻しのための盤面の状態の記録方法、巻き戻し可能かどうかを判定する方法

1.3.2 で述べたような jump 機能や undo 機能を実現するために、フィールドとして以下の要素を用いている。

- 現在の状態の世代数を記録する整数 currentGen
- 1 つ前の状態の世代数を記録する整数 prevGen
- 盤面、すなわち boolean 型の二次元配列を保存するリスト history
- 盤面とともに prevGen を保存するリスト generations
- Undo 可能回数を計測する整数 limitUndo
- Jump 機能で 1 が指定されると立つフラグ doneJump1

これらリストの実装には、操作が容易でデータの追加・削除が高速と言われる LinkedList を用いている。prevGen は、undo 機能での巻き戻し可能回数に関わる。以下に、各種の盤面の編集・更新操作がなされた時に、これらの要素がどう用いられるかを具体的に述べる。

3.2.1 マウス操作による編集時

マウスのクリックやドラッグによってセルの状態を反転させるメソッド changeCellState が呼ばれると、現在の世代数が 1 であれば、編集前の盤面の状態をリスト history に追加し、同時に、prevGen の値をリスト generations へ追加し、また currentGen の値を prevGen へ代入する。

そしてセルを反転させた後 currentGen に 1 を代入する。これは、マウスによって編集された盤面は常に第 1 世代であることを表す。

また Undo 可能回数 limitUndo が 32 未満であれば 1 増やす。

3.2.2 paste 時

paste 機能はメソッド isPastable が true の時実行される。マウス操作時と同様に現在の世代数が 1 であれば、まずペースト前のセルをリスト history に追加し、同時に prevGen をリスト generations に追加する。そして、すべてのセルをペーストしたのち、prevGen に currentGen の値を代入し、currentGen には 1 を代入する。ペーストされた盤面は常に第 1 世代となる。

また Undo 可能回数 limitUndo が 32 未満であれば 1 増やす。

3.2.3 next 時

next 機能はメソッド isNextable が true の時実行される。まずマウス操作時等と同様にまず現在の盤面が第 1 世代であるかを確認する。第 1 世代であれば、現在の盤面はまだリスト history にないので追加する。同時に、prevGen の値を generations に追加する。

そして盤面の状態を 1 回更新し、currentGen の値を prevGen に代入した後、currentGen の値を 1 増加させる。これは世代数が 1 増えたことを表す。

また Undo 可能回数 limitUndo が 32 未満であれば 1 増やす。

3.2.4　jump 時

jump 機能はメソッド isJumpable が true の時実行される。、まず盤面の状態がゲーム開始時の状態であれば何もせずリターンする。またマウス操作時等と同様に同様現在の盤面が第 1 世代であれば現在のセルと prevGen をリストに追加する。ただし、直前に jump 機能で第一世代を指定していた場合、すなわち現在のセルがリストに保存された最新のセルと同一であれば追加を行わない。

ジャンプ先の世代数として、現在の世代数 currentGen よりも大きい値 n が指定された場合、n-currentGen 回更新を行うことでジャンプを行う。また、現在の世代数 currentGen よりも指定された値 n が小さい場合は、リストに保存された最新のセルを読み込み、n-1 回更新を行うことでジャンプを行う。この 2 つの場合では、ジャンプ後 prevGen に n-1 を代入する。

ジャンプ先の世代数として 1 が指定された場合は、リストの最新のセルを読み込み更新は行わない。prevGen にはリスト generations に保存された最新の値を代入する。

Undo 可能回数 limitUndo は、更新を行った回数だけ増やす。ただしジャンプ先の世代として 1 が指定された場合は、limitUndo を 1 増やし、さらに doneJump1 に true を代入する。limitUndo が 32 を超えると 32 を代入する。最後に currentGen に n を代入する。

3.2.5　undo 時

undo 機能はメソッド isUndoable が true の時実行される。まずリスト history に保存された最新のセルを読み込み、currentGen に prevGen を代入する。また、prevGen を 1 減じ、この値を m とする。その後、読み込んだセルに m 回の更新処理を行う。つまり、(1 つ前の状態の世代数-1) 回の更新を行う。

m=0 の時は、更新せず、リスト history の最新のセルを削除する。また、リスト generations の最新の値を prevGen に代入し、またそれを generations から削除する。つまり、prevGen=1 の時に undo を実行すると、保存された最新の盤面をポップする。

最後に limitUndo を 1 減じる。

3.2.6　実際のプログラム上での挙動

3.2 のフィールドが実際にどのように用いられているかを、図 1 に示す。

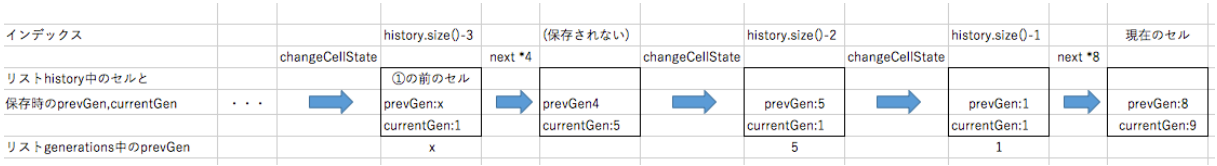

図 1　巻き戻しに関するフィールドの様子

図 1 では、現在の prevGen が 8 であるから、undo を行うとリスト history に保存された最新の第 1 世代の盤面をよみ出し、8-1=7 回の更新処理を行うことで盤面を第 8 世代にする。このようにして、8 回 undo を行うことで第 1 世代に戻るが、この 8 回目を行う時点で prevGen が 1 であるから、1-1=0 となりリスト history とリスト generations からのポップが起こる。その後 undo を行うと、prevGen が 1 であるから再びポップされる。この時点で prevGen は 5 であるから、読み出したセルに対して 5-1=4 回更新が行われ第 5 世代となる。

3.3　盤面の描画において、セルの境界線の位置を計算する方法・計算式

本プログラムでは、盤面がウィンドウの縦幅・横幅の小さい方よりやや小さくなるように、以下のようにしてセルの辺の長さを決定している。

```
s = Math.min(
        (int) (this.getWidth() * (cols-1) / (cols*cols)),
        (int) (this.getHeight() * (rows-1) / (rows*rows))
        );
```

そして、左端と上端に辺の半分のスペースを空けることで、盤面を見易くしている。具体的には、盤面上の座標に対してウィンドウ上の座標を返すメソッドを次のように定義した。

```
private int cnvX(int x){

    return (int) (s/2 +  (x)*s);

  }
```

これを用いることで、セルの境界線を簡単に描画することができる。

3.4　マウスカーソルの座標から対応するセルの座標を計算する方法・計算式

カーソルの座標からセルの座標を計算するには、前節で示した関数の逆関数を用いる。具体的には、次のようにメソッドを定義した。

```
private int invX(int x){
    if(x < s/2 ) return -1;
    return (int)( x -((int)s/2) )/s;
  }
```

3.5　新しいウィンドウを開く方法

新しいウィンドウを開く際には、main メソッド内と同様に invokeLater メソッドを実行している。これにより、行数・列数の決定、GUI パーツの構成などの lifegame の一連の処理が再び実行される。

この方法では、元のウィンドウと新たなウィンドウが駆動しているスレッドが同一であるため、一方のウィンドウで非常に時間のかかる処理を行っている間、他方のウィンドウでの処理が行われなくなってしまう。しかし swing はシングルスレッド設計である [2] から、これは当然であると言える。

3.6　次の世代の盤面の状態を求める方法

next,jump,undo 機能等で実際に盤面を更新する際には、メソッド alternateGen を用いている。前述した通り jump や undo では連続して多くの回数の更新を行う為、高速な処理が求められる。以下のように、複数のアルゴリズムを考案し可能な限り処理速度を追求した。

3.6.1　あるセルの周囲の生存セル数を数え上げる方法

あるセルに対してその周囲 8 マスのセルが生存しているかどうかを確かめる。配列の範囲外にアクセスしない為に if 文での分岐が多く必要になる。付録 A のように if 文中の論理積をなくすような記述をすることで、比較の回数をセル 1 つあたり最悪でも 16 回まで減らせる。

3.6.2　isAlive メソッドで配列の範囲外にアクセスしないようにする方法

セルの生存状況を返すメソッド isAlive で、配列の範囲外にアクセスした際に false を返す実装をすると簡潔に生存セル数を数え上げることができる。しかし、付録 B のように、isAlive メソッド 1 回で最悪 5 回の比較を行い、それを周囲 8 マスに対して行う為最悪 40 回の比較文を実行することになる。

3.6.3　二次元配列の拡張

以降で述べる方法では、セルを保存する配列は (行数 +2)*(列数 +2) の二次元配列とし、外周のセルを常に死んでいるものとみなすことで、あるセルの周囲の生存セル数を数える際に配列の範囲外にアクセスしているかどうかを検出する if 文を省く。

3.6.4　漸化式を用いる方法

あるセル x の右上と右下のセルの生存セル数の合計を vertical0, 右のセルの生存セル数を right0 とする。同様に、x の 1 つ左のセルに対して vertical1 と right1 を、x の 2 つ左のセルに対して vertical2 と right2 を定義する。

すると、x の周囲 8 マスの生存セル数 livingcells は、以下の式で求められる。

$$livingcells = vertical2 + vertical1 + vertcal0 + right2 + right0 \tag{1}$$

livingcells を求めた後、vertical2 に vertical1 を、vertical1 に vertical0 を、right2 に right1 を、right1 に right0 を代入する。

このようにすることで、1 つの行の上で再帰的に生存セル数を数えることができる。この方法では、セル 1 つあたり比較文は vertical0 と right0 を求める為の 3 つで済む。付録 C に、この方法を実装したコードを示す。

3.6.5　整数の二次元配列に生存セル数を蓄積していく方法

前節で述べた方法でも、一つのセルの生存状況を確かめるために前世代のセルを 3 回参照している。これを 1 回で済ませる為に、次の方法を考案した。

まず、(行数 +2)*(列数 +2) のメモ用 2 次元整数配列 X を用意する。前世代の盤面 Y のあるセル Y[i][j] が生存していれば、それに対応する X[i][j] の周囲 8 つ (X[i-1][j-1],X[i-1][j],,,X[i+1][j+1]) をそれぞれ 1 増やす。これを Y のすべてのセルに対して行えば、Y[i][j] の周囲 8 マスの生存セル数が X[i][j] に格納される。

この方法では、比較 1 回につきインクリメントを 8 回行うが、比較回数はセル 1 つあたり 1 回となり非常に高速であると言える。本プログラムでは、この方法を用いた。そのコードを付録 D に示す。

3.7　盤面の行数・列数を決定する方法

列数・行数の入力をユーザーに求める際には、showInputDialog メソッドを用いている。これは、入力エントリ付きのダイアログを表示し、ユーザーの入力を string オブジェクトとして返すものである。受け付けた入力が整数でなかったり、最大値・最小値の範囲内でない限り、再びダイアログを表示する。

3.8　スクリーンショットを撮る方法

スクリーンショット機能の実装は、[1] 等を参考にした。

まず、ウィンドウの画面上での座標と縦幅・横幅を取得する。そしてメソッド showSaveDialog を用いて、ファイルの保存場所とファイル名を設定するダイアログを表示する。この情報をもとに file オブジェクトを生成する。

その後 Robot クラスの createScreenCapture メソッドで、取得した座標・サイズでスクリーンショットを撮影する。これは厳密にはウィンドウのスクリーンショットというよりも画面自体のスクリーンショットであるが、本機能の実行時には本プログラムが最前面に表示されているはずなので、他のウィンドウが映り込む可能性は低い。

最後に file オブジェクトに対して png 形式でスクリーンショットを書き込む。

3.9　セルの選択方法

セルの選択には、シフトキーの押下を認識する必要がある。このために、クラス BoardView にキーリスナーを実装した。

シフトキーが押下されると、フラグ isShiftDown を立てる。さらに、盤面の変更を防ぐためにフラグ isRocked を立てる。シフトキーが押された状態のままマウスをクリックすると、その座標 p を記録し、選択範囲とする。さらにドラッグすると、その座標と p で囲まれる範囲を選択範囲とする。選択範囲は、盤面の描画時に通常とは異なる色で描画する。

キーが離されると、フラグ isShiftDown を倒す。また、マウスが離されるとフラグ isRocked を倒す。

3.10　copy を行う方法

選択されたセルが存在するときに copy を行うと、その範囲分の二次元配列を生成し、選択されたセルの状態をコピーし、フィールド clipBoard に参照を代入する。

3.11　paste を行う方法

フィールド clipBoard が null でなく、かつ選択されたセルが存在するときに paste を行うと、clipBoard と選択範囲の縦幅・横幅それぞれの小さい方をとり、その範囲内でセルの状態を変えるメソッド setCellState を用いて clipBoard の状態を盤面へ貼り付ける。

付録 A　周囲 8 マスを数え上げる方法のコード

```
//上 3 マス
        if(i>1){
          //左上
          if(j>1){
            if(cells[i-1][j-1]==true) livingcells++;
          }
          //上
          if(cells[i-1][j]==true) livingcells++;
          //右上
          if(j<=cols-1){
            if(cells[i-1][j+1]==true) livingcells++;
          }
        }
        //右 2 マス
        if(j<=cols-1){
          //右
          if(cells[i][j+1]==true) livingcells++;
          //右下
          if(i<=rows-1){
            if(cells[i+1][j+1]==true) livingcells++;
          }
        }
        //下 2 マス
        if(i<=rows-1){
          //下
          if(cells[i+1][j]==true) livingcells++;
          //左下
          if(j>1){
            if(cells[i+1][j-1]==true) livingcells++;
          }
        }
        //左 1 マス
        if(j>1){
          if(cells[i][j-1]==true) livingcells++;
        }
```

付録 B　isAlive のコード

```
    if( x < 0 || y < 0 || x >= cols || y >= rows) return false;
    if(cells[y+1][x+1]==true) return true;
```

付録 C　漸化式を用いる方法のコード

```
for(int i=1;i<=rows;i++){
      vertical2=vertical1=right2=right1=0;
      for(int j=1;j<=cols;j++){
        vertical0 = right0 = 0;
        int livingcells = 0;
        //右上
        if(tmpcells[i-1][j+1]==true) vertical0++;
        //右
        if(tmpcells[i][j+1]==true) right0++;
        //右下
        if(tmpcells[i+1][j+1]==true) vertical0++;
        livingcells = vertical2 +vertical1+vertical0+right2+right0;
        right2=right1;
        right1=right0;
        vertical2=vertical1;
        vertical1=vertical0;
        if(tmpcells[i][j]==true){
          if(livingcells==2||livingcells==3) tmp[i][j] = true;
          else tmp[i][j] = false;
        }else{
          if(livingcells==3) tmp[i][j] = true;
          else tmp[i][j] = false;
        }
      }
    }
```

付録 D　整数の二次元配列に生存セル数を蓄積する方法のコード

```
int tmpint[][] = new int[rows+2][cols+2];
    for(int i=1;i<=rows;i++){
      for(int j=1;j<=cols;j++){
```

```
      if(tmpcells[i][j]==true){
        tmpint[i-1][j-1]++;
        tmpint[i-1][j]++;
        tmpint[i-1][j+1]++;
        tmpint[i][j-1]++;
        tmpint[i][j+1]++;
        tmpint[i+1][j-1]++;
        tmpint[i+1][j]++;
        tmpint[i+1][j+1]++;
      }
    }
  }

  //周囲の生存セル数に応じて生存・誕生を決定
  for(int i=1;i<=rows;i++){
    for(int j=1;j<=cols;j++){
      if( tmpint[i][j]==3 ||
        (tmpint[i][j]==2 && tmpcells[i][j]==true)
      )tmpcells[i][j] = true;
      else tmpcells[i][j] = false;
    }
  }
```

参考文献

[1] 私の歴史と今 Java で定期画面キャプチャアプリを作る
http://tsurugeek.hatenablog.jp/entry/20090530/1243686971
2017 年 1 月 13 日閲覧

[2] Swing とスレッド
http://wisdom.sakura.ne.jp/system/java/swing/swing4.html
2017 年 1 月 13 日閲覧

[3] JavaDrive
http://www.javadrive.jp/
2017 年 1 月 13 日閲覧