ゲーム性をもったセルオートマトンの設計と実装

2014SE020 廣瀬 隼大　2014SE059 増田 大輝　2014SE089 柴田 心太郎
2014SE112 安田 拓也　2014SE114 矢澤 拓海

2017 年 7 月 26 日

1　はじめに

1.1　要求分析

いままでの大学の講義において，様々なツールを使用し共同で開発する機会は少なかった．しかし，今後社会に出たときに様々なツールを用いて共同に開発することは積極的に行われている. そのため今回では様々なツールを用いて共同開発を行う. また，自分たちがこれまで学んできた内容を活かしたいと思い，ゲーム性をもったセルオートマトンのゲームを制作することにした. ただしここで，ゲーム性とは，勝ちもしくは負けが存在し，ユーザが何らかの形で干渉できることとする.

1.2　セルオートマトン

セルオートマトンとは計算システムにおける数理モデルの一種である．具体的な遷移は k 次元ユークリッド空間に一様なセルを配置し，与えられた条件 (遷移規則) に従って，一定時間ごとにセルの様相が変化することにより，計算を行う．セルオートマトンの定義は $A = (\mathbb{Z}^k, Q, (n_1, .., n_m), f, \#)$ とする [1]．ただしこのとき，$\mathbb{Z}$ は k 次元ユークリッド空間中の整数座標をもつ点の集合であり，つまりはセルの集合である.Q は各セルが取り得る内部状態の空でない有限集合である.$(n_1, ..., n_m)$ は $(\mathbb{Z}^k)^m$ の要素 (ただし $m = 1, 2, .)$，であり，近傍を表す. または，セルの状態が遷移する際に参照するセルのことである. 関数 f は $Q^m \rightarrow Q$ の局所関数であり，$q_1, ...q_m, q \in Q$ に対して $f(q_1, ..., q_m) = q$ を満たすときこの関係を遷移規則と呼ぶ. したがって，f は遷移規則の集合である.$\# \in Q$ は静止状態，つまり空白の状態を表す.

また，セルオートマトンを用いた代表的なものとしてライフゲームが挙げられる．ライフゲームとは，誕生，生存，死亡などのプロセスを簡易的なモデルで再現したシミュレーションゲームである．今回はこのライフゲームに新たにルールを設けてよりゲーム性の高いものを制作する.

1.3　目的

セルオートマトンを用いたゲームの制作と，その制作の上で開発ツールや共同開発の仕方，バージョン管理について学ぶ.

1.4 解決策

本目的を達する解決策として，二つの方法が挙げられる．一つ目は，1 から Java の勉強をし，プログラムを作り上げていく方法である. もう一つは，既存にあるプログラムを利用して今回作りたいものを作る方法である．前者では，Java の勉強ができ，自分たちでクラスなどの定義ができるためプログラムにミスがあった時に修正がしやすいという利点がある. 一方で，Java をより理解した上でプログラムを書くことが必要になるので，時間がかかり過ぎてしまうという欠点がある．後者では，既存のプログラムを利用するので比較的短時間でプログラムを書くことができるという利点がある. また，この方法は近年ソフトウェア開発においてよくみられる手法の一つなので今後に活かせるといった利点もある. 一方で，プログラムにミスがあった時の修正を行う際に，どこが悪いのかを見つけることが困難であり，修正をするのに時間がかかってしまうという欠点がある．

今回この二つの方法を比較して考えた結果，限られた時間のなかで作成しなければならないこともあり，Java を 1 から学ぶ時間をかけられないため，既存のプログラムを利用する方法を解決策として選択した.

2 設計

2.1 目標の設定

今回，基盤として利用する既存のプログラムはライセンスフリーの「LifeGame」である [2].
まず，ゲーム制作にあたり，ゲーム自体の大まかなルールを以下の様に設定する.

- 画面に 14 × 14 個のセルを配置.
- セルには生状態と死状態の 2 種類があり，さらに生状態は黄，赤，緑，青の 4 種類.
- シンプルなライフゲームの規則に新たな規則を付加.
- クリックでセルの移動.
- ゲームとして成り立つ様に Score と Gameover の表示.

上記のルールを元により詳細なルールを決める.

- 画面に 14 × 14 個のセルが配置されている，セルには生状態と死状態の 2 種類があり，さらに生状態は黄，赤，緑，青の 4 種類がある.
- セルはプレイヤーが干渉できる部分とセルオートマトンで動く部分，ランダムに上書きされる部分に分かれる.
- 基本的にセルオートマトン全体は単位時間ごとに遷移規則どおりに動く，
- 遷移規則には大きく分けて誕生，死亡の二つがある.
- 誕生は自身のセルが死状態である際に行われる.
- 自身の周囲に生状態のセルがちょうど 3 つあるならば生状態へと遷移する. その際何色になるかは周囲のセルによって決定される.
- 自身と同じ状態のセルが 3 つ以上連続して隣接しているならば死状態へと遷移する.
- 死亡したセル 1 つにつきスコアが 1 増加する.

- セルオートマトンの一番外側の部分は単位時間ごとにランダムで生状態へと遷移する. この遷移は通常の遷移の後に行われる.
- プレイヤーは任意のタイミングでセルをクリックすることで干渉できる.
- セルをクリックするとセル自身とそのセルの右、右下、下のセルが時計回りに入れ替わる.
- 生状態のセルの数が 120 より大きくなるとプレイヤーは敗北する.

2.2 要求仕様書

- このシステムは 1 人での利用を想定している.
- ユーザがシステムを起動すると，有限長のセルオートマトン全てのセルが死状態であり，2 つのボタンが存在する画面が表示される.
- 最も外側の一回りのセル全てを外部セルとして扱う. それ以外のセルを内部セルとして扱う.
- ボタンはスタート，リセットがある.
- 内部はセルオートマトンの遷移規則通りに遷移する.
- リセットボタンを押すとセルオートマトンの全てのセルが死状態となる.
- ユーザはスタートボタンを押すと，外部セルの状態がランダムに決定されゲームが開始される.
- ゲームがスタートした後，一定の時間が経過する毎にセルオートマトンの 1 単位時間が経過したことになり，遷移が実行される.
- ゲーム中にリセットボタンを押すと，全てのセルが死状態になり，スコアも 0 となる.
- ユーザはあるセルをクリックすると，クリックしたセルの相対座標で (0,0),(0,+1),(+1,+1),(+1,0) のセルを時計回りで 1 つずつ，ずらすことができる.
- 近傍中にちょうど 3 つの生状態の色のセルがあれば，注目セルは生となる.
- 死判定は 12 近傍となる．ただし，12 近傍は注目セルを中心として放射状に広がる 12 個のセルである．12 近傍は対称性を持つ.
- 注目セルが生状態であり，注目セルを含めた連続する 3 つのセルが同じ色の生状態であるとき，注目セルは死状態となる. このとき，スコアが足される.
- 上記の遷移規則に当てはまらないとき，注目セルは死状態となる.
- セルオートマトンの様相の遷移の 1 単位時間は現実時間での一定の秒数とする.
- 生状態のセルが 1 単位時間後に死状態のセルとなったとき，その数のカウントがスコアとなる.
- 生状態のセルが 120 以上になるとゲーム終了.
- 終了時にスコア，Gameover とゲーム画面が表示される.
- スコアは表示されるだけであり記録はされない.
- システムは終了せずに動き続ける.

2.3 プロセス仕様書

2.2 小節の要求仕様からプロセス仕様書を記述した.

- ユーザはシステム起動係にシステム起動要求を行う.
- システム起動係はシステム起動要求を受けとると，システムを起動し，画面に画面表示命令を行う. そ

して，画面に有限長セルオートマトン全てのセルが死状態であり，3 つのボタンが存在するのを表示する．

- ユーザは，ゲーム開始係にゲーム開始要求をする．
- ゲーム開始係は，ゲーム開始要求を受けとると外部セルオートマトンにランダムに生状態，死状態になるよう命令し，内部セルオートマトンに遷移規則通りに動くよう命令する．そして，セル管理係内のセル状態係にセルのデータを送る．
- ゲーム開始係は，生状態のセルが一定数以上になった時，ゲーム終了係にゲーム終了要求を出す．
- ゲーム終了係は，要求を受けとると画面表示命令を出し，リセット以外の動作を受け付けなくする．
- セル状態係は，セル管理係内のセル描画係にセル状態を送る. また，現在の状態を判定して，遷移係に近傍を送る．
- セル描画係は，セルの遷移規則の結果を画面に表示する. また，遷移が動かないと判断したら，ゲーム終了係にゲーム終了要求を送る．
- 遷移係は，送られてきたデータを元に放射能状の 12 近傍を見て，注目するセルを含めた連続する 3 つのセルが同じ色で生状態であるとき，注目セルを死状態にする. また，セルオートマトンが消えた際，スコア係にスコア増加要求を出す．

2.4 用語集

2.3 小節において未定義であった言葉を用語集としてまとめる．

- 生状態とは，セルの状態が黄、赤、緑、青であること．
- 死状態とは，セルの状態が白であること．
- 注目セルとは，近傍においての中央のセル．
- システム起動係とは，システムを起動させる．
- システム起動要求とは，システムが起動するよう要求する．
- 画面表示命令とは，スコアやボタン，GameOver やセルオートマトンの状況を画面に表示するように画面に命令する．
- ゲーム開始係とは，ゲームを開始する．
- ゲーム開始要求とは，ゲームが開始するよう要求する．
- セル管理係とは，全てのセルの状態を管理する．
- 遷移係とは，遷移を実行する．
- スコア係とは，スコアを増加させる．
- スコア増加要求とは，スコアが増加するよう要求する．
- ゲーム終了係とは，ゲームを終了させる．
- ゲーム終了要求とは，ゲームが終了するよう要求する．
- 回転係とは，クリックされたセル，右，真下，右斜め下のセルを右回転させる．
- 回転要求とは，上記のセル回転をするよう要求する．
- 一時停止係とは，ゲームを一時停止させる．
- 一時停止命令とは，ゲームを一時停止させるよう要求する．
- ゲームリセット係とは，ゲームをリセットさせる。全てのセルを死状態にする. また，リセットボタン

意外動作しないようにする．

- ゲームリセット要求とは，ゲームがリセットされることと，全てのセルを死状態にする．そして，リセットボタン意外動作しないよう要求する．
- 全てのセルリセット命令とは，全てのセルを死状態にする．
- セル状態係とは，与えられたセルがどのような状態かを判定する．
- セル描画係とは，生状態の数を数える．

コンテキスト図

プロセス仕様を元にコンテキスト図を記述する.

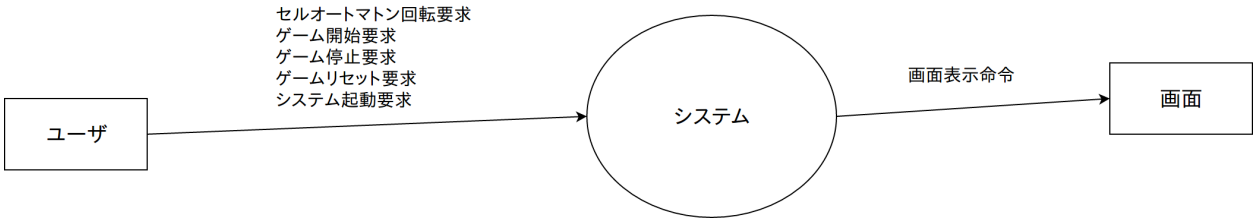

図 1　コンテキスト図

DFD レベル 1

コンテキスト図を記元に DFD を記述する.

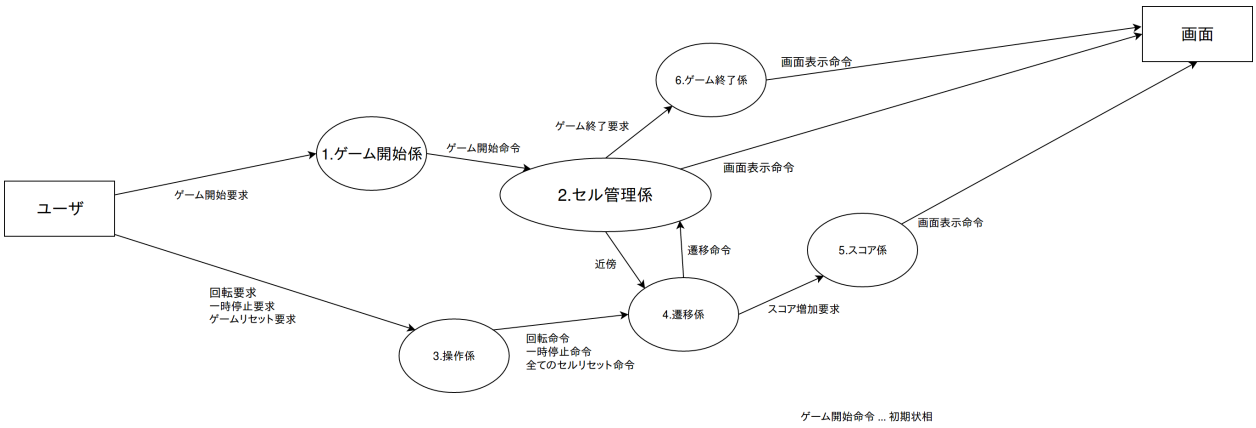

図 2　DFD レベル 1

DFD レベル 2-2

DFD の「2:セル管理係」を分割し，より詳細な DFD を記述する．

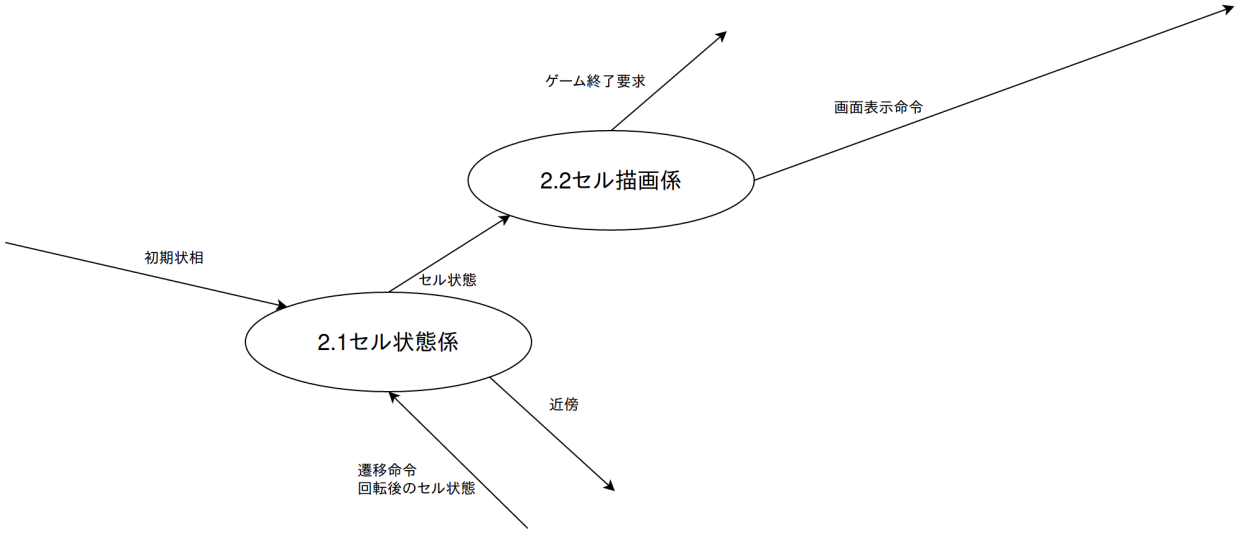

図 3　DFD レベル 2-2

DFD レベル 2-3

DFD の「3:操作係」を分割し，より詳細な DFD を記述する．

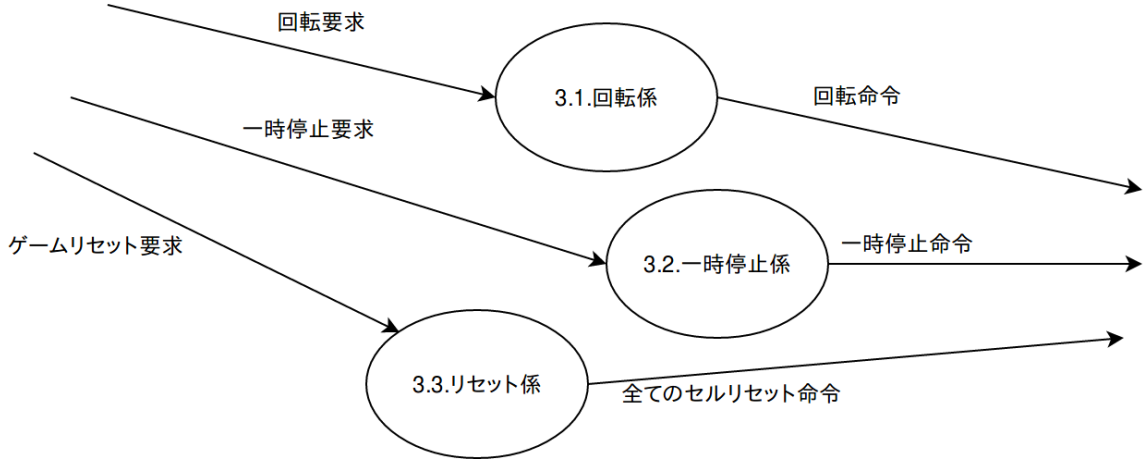

図 4　DFD レベル 2-3

3 実装

3.1 定義

1.1 小節の定義を用いて今回扱うセルオートマトン A' を以下に表す.

$$A' = (\{0,1,2,3,4,5,6,7,8,9,10,11,12,13\}^2, \{0,1,2,3,4\},$$
$$((0,0),(0,1),(1,0),(1,1),(0,-1),(-1.0),(-1,-1),(1,-1),(-1,1),(0,2),(2,0),(0,-2),(-2,0)), f, 0)$$

状態は 0 が白，1 が黄，2 が赤，3 が緑，4 が青を表す．関数 f に関しては，正確な定義が難しいので言葉で説明する．自身が生状態であり、かつ自身と同じ状態が 3 つ連続するならば死状態へと遷移する．自身が死状態であり，かつ $(0,1),(1,0),(1,1),(0,-1),(-1.0),(-1,-1),(1,-1),(-1,1)$ に生状態がちょうど 3 つ存在するならば生状態を $mod4$ の加算で合計し，その値に 1 加えた状態へと遷移する．

3.2 環境ツール

今回ゲームの制作，実装するにあたり，様々なプログラミング言語及び環境ツールを用いた．
以下，用いたプログラム言語及び環境ツールの解説，また，どのように用いたかを記述する．

Java
: ゲームということで操作のしやすいウィンドウ化をしたかった．そのため，ユーザーインターフェースの機能が標準であり，広く使われている点から Java を採用した．
また，オブジェクト指向の講義で Java を習っているため理解しやすい.

github
: GitHub では，ブランチという同じリポジトリ中で複数の変更を同時に進めていくことができる機能がある. そのため，皆が同時にプログラムを管理できる環境ツールとして GitHub を用いた．具体的な操作としては，ブランチは master,develop,develop2 を作成した.master では基盤となるとなるプログラム (LifeGame)[2] をおいて皆が触れるようにした. そこから develop では master 内のプログラムをもとに遷移規則をプログラムし,develop2 ではゲームのフレーム及びボタン等の表示のプログラムを追加した. そこから，develop を develop2 にマージして形を完成させた. 最終的に master にマージし，一つの形にした.

Makefile
: 今回, Java のファイルが複数あり実行させるために全てをそれぞれコンパイルしなければならなかった，そのため Makefile を作成した．この Makefile は本プログラムをいれたディレクトリ内においてmake コマンドを打つことで make 内に記述されたファイルを同時にコンパイルしてくれる利点がある．また，コンパイルした後にプログラムの一部を変更した場合，その変更したファイルのみをコンパイルしてくれることも利点のひとつある.

GoogleDrive
: 会議の際，記録したメモを共有する場として用いた. また，各自が進めた進捗をまとめることにより各自の役割が明確化し，効率的に開発できた.

3.3 追加及び変更点

既存プログラムからの変更点は主に以下の 9 つ挙げられる．

セルの色

基盤のプログラムではライフゲームのセルに当たる変数の型が boolean 型、つまり True or False の二者択一の型であったため，セルの色を二色の黒と白以上に増やすことが出来ないようになっていた，今回はセルの色を増やしたかったため，変数の型を整数型である *int* 型に変更し，0 の場合は白色，1 の場合は黄色，2 の場合は赤色，3 の場合は緑色，4 の場合は青色のようにした．

遷移規則の変更

- 誕生セルの場合
  セルの色が合計で 5 色に増えたことで，セルの誕生時にどの色のセルが誕生するかを決定することが必要となった．今回は周囲のセル内の数字の合計を算出し，その数字を 4 で割った余りに 1 を足した数が誕生したセルの数字になるようにプログラムし直した． 例えば，周囲のセルが 3 つで，色がそれぞれ赤，緑，青だった場合，誕生するセルは，$(1+3+4) mod 4+1=4$ より，青が誕生する．
- 死亡セルの場合
  基盤のプログラムのライフゲームに設定されていた 9 近傍を放射状の 13 近傍へ拡大し，3 つ以上隣接した際に死亡するよう遷移規則を変更した．

外周の実装

左上から順番に全てのセルを参照し，そのセルが何番目であるかによって外周かどうかの判定を行う，その結果から，外周のセルの状態をランダムで書き換えるよう変更した．

セルの回転

基盤のプログラムにおけるセルをクリックした際の挙動は，生状態と死状態の入れ替えであった．今回は，クリックされたセルとその右，右下，下のセルを参照し，状態を時計回りに入れ替えるよう変更した．

スコアの表示

今回はゲームを作成するという目的であったため，死亡したセルの個数を記録して表示する機能を加えた．その場合，セルが死亡する度に表示を更新しなければならないため，セルの更新後にスコア表示を読み込む必要がある．具体的には，まず，周りのセルを調べて世代交代を行うメソッド「checkSroungings」 が返り値を持つように変数を *void* から *int* に変更する．その後，セルが死亡を意味する変数 willLiving = 0 を読み込む度に，変数 score に 1 を足していくようにする．最後に，返り値を持つようにしたため score に返り値を読み込むように変更した．

セル数及び Gameover の表示

ゲーム性を持たせるため画面内に表示されている生状態のセルをカウント及び表示し，生状態のセルの個数が 120 を越えるとクリックによる回転ができなくなる，かつセルの更新が止まり Gameover が表示されるプログラムを追加した.

スコアのリセット

スコアが Gameover になった後，同じスコアが残っていてはゲームとしてよくないためスコアをリセットする必要がある．そのため Gameover になった後にゲームリセットが押され，次のゲームがはじまると同時にスコアもリセットするようにプログラムを書き換えた.

スコア及び Gameover の表示カラー

スコアやボタンのカラーが見にくいとゲームとしてよいものにはならないと考えたので色を変更した.色の変更には背景色と文字の二種類のメソッド「setBackground」と「setForeground」のプログラムを書いた.

ボタンの変更

基盤のプログラムに元々付いていたボタンは，今回のゲームをする上では必要がないため，スタートとリセット以外のボタンを削除した.

3.4 成果物

今回作成したゲームは GitHub 上にアップロードした.
URL:〈 https://github.com/yokoyama-lab/Cell-Automaton-Game 〉

4 テスト

4.1 遷移規則について

- クリック時のテスト
  ・テスト内容:すべてのセルをクリックする.
  ・予想出力:セルの中に外周のセルが含まれる場合は回転せず, それ以外の場合は回転する.
  結論:セルの中に外周のセルが含まれる場合は回転せず, それ以外の場合は回転した.

- 遷移規則による死亡のテスト
  ・テスト内容:3 つ以上セルが隣接した場合そのセルが死亡するか.
  ・予想出力:各形状，各色に対し死亡する.
  結論:6 種類の形状の各色に対し，死亡することを確認した [図 5].

- 遷移規則による誕生のテスト

  丁度 3 つの際に誕生する点においては，基盤のプログラムを利用したことにより厳密なテストは必要ないと判断した.

  ・テスト内容:近傍のセルの色の状態により誕生するセルの色を決める．誕生する色に関してのテストケースは以下の 20 パターンである.

  ・予想出力:

テストケース	→	予想出力
$(1,1,1)$	→	4
$(1,1,2)$	→	1
$(1,1,3)$	→	2
$(1,1,4)$	→	3
$(1,2,2)$	→	2
$(1,2,3)$	→	3
$(1,2,4)$	→	4
$(1,3,3)$	→	4
$(1,3,4)$	→	1
$(1,4,4)$	→	2
$(2,2,2)$	→	3
$(2,2,3)$	→	4
$(2,2,4)$	→	1
$(2,3,3)$	→	1
$(2,3,4)$	→	2
$(2,4,4)$	→	3
$(3,3,3)$	→	2
$(3,3,4)$	→	3
$(3,4,4)$	→	4
$(4,4,4)$	→	1

  結論:以上のテストケースに対し遷移規則通りにセルが誕生することを確認した.

- Gameover 後の挙動及びスコア表示

  ・テスト内容:Gameover 後にクリックを行い動作しないか，またセル死亡時スコアが加算されているか.

  ・予想出力:Gameover 後には，クリックの動作や外周の書き換えを行わなくなる．消えたセルの数だけスコアが加算されている.

  結論:Gameover 後には，クリックの動作や外周の書き換えを行わなくなり，リセットが可能であることを確認した. また，[図 6] のタイミングに対し，消えたセルの数だけスコアが加算されていることを確認した. 具体的には，[図 6] の時点で 34 個のセルが死亡しており，スコアが 34 増加している.

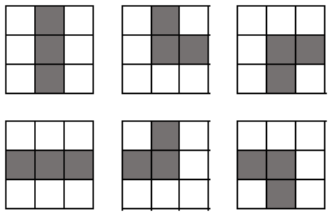

図 5　死亡パターンのテスト

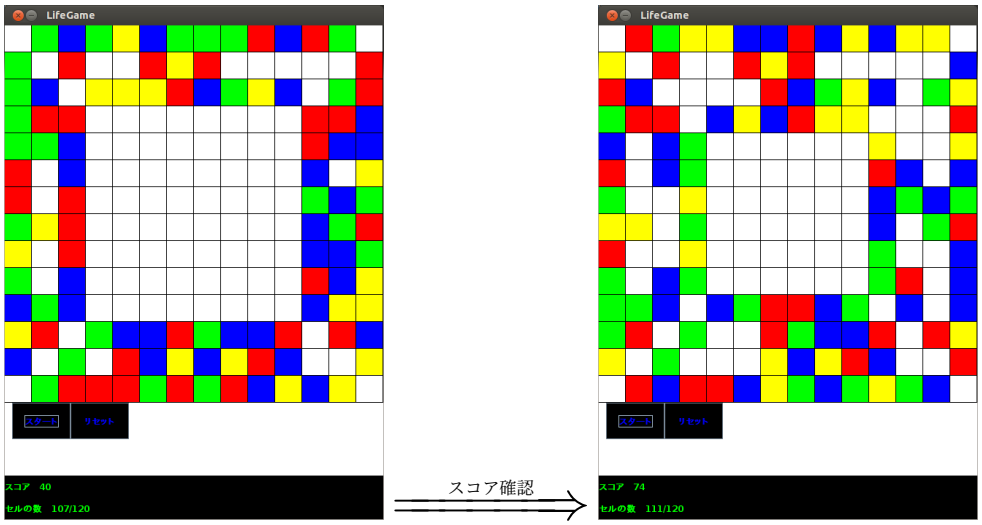

図 6　誕生の及びスコアのテスト

4.2　ユーザの干渉について

- スタートボタンのテスト

  ・テスト内容:スタートボタンをクリックした際の挙動.

  ・予想出力:スタートボタンを押した際ゲームがスタートし，もう一度押すと一定時間後に停止する．さらに，もう一度押すとスコアなどに変化が起きることなく再開する.

  結論:スタートボタンを押した際ゲームがスタートし，もう一度押すと一定時間後に停止した．さらに，もう一度押すとスコアなどに変化が起きることなく再開できた.

- リセットのテスト

  ・テスト内容:リセットボタンをクリックした際の挙動.

  ・予想出力:リセットボタンを押した際にスコアとセルがリセットされる.

  結論:リセットボタンを押した際にスコアとセルがリセットされたのが確認できた.

5　おわりに

5.1　結果 (目標達成度)

今回の目的は，まずゲーム部分においては，遷移規則どおりにオートマトンが動いて，ゲームとして制作したものが成り立つようにすることであった．この目的に対して評価を行うと，実際に遷移規則通りに動き，ゲーム性をもたせるためにスコアの表示や Gameover の表示，ユーザをゲームに干渉させるために，クリックすることでセルを移動させることに成功した．また，開発ツールを使用しての共同開発という目的に対しても，開発ツールとして GitHub を利用して共同開発をすることができた. そして，バージョン管理については，GitHub 上に履歴が残るため管理が容易に行えた. これらの結果より今回の目的は達成することができた.

5.2　考察

今回，目的を達成することはできたが，まだまだ改良すべき点がたくさんある．一つ目にゲームの難易度や盤面が安定状態になってしまうということがあった．それぞれの改善策は，難易度に対しては，遷移する時間を変更することや難易度を変更するボタンなどを追加するなどの改善方法が挙げられる．盤面が安定状態になってしまうという点に対しては，現状外部セルにランダムで表示されるのが必ず色をもった状態であらわれるので，死状態 (色をもたない状態) もランダムで表示する改善策が挙げられる，そうすることで，盤面が安定状態になることがなくなるのではないかと思われる．二つ目にウィンドウのレイアウトについて時間が限られていたため考慮しておらず，より遊びやすいようにするために改良することが必要であると思われる (GUI の向上)．また，共同開発を行っていった上で，プログラムが競合してしまったときに GitHub では，競合している部分がプログラム上に表示されるため変更が容易に行うことができ，効率が良かった．GitHub は過去のバージョンも履歴として残るので間違いがあった場合に古いバージョンのものから作り直すことができるため便利であった.

共同開発において GitHub の様なツールは大変便利で作業効率が上がる．また，複数人と共同で開発することで作業の分担化ができ一人一人が行える作業の幅が広がった．そのため，共同開発がいかに重要かを今回の演習を通して確認できた.

6　役割分担

2014SE020 **廣瀬 隼大**

- 本稿 2 章

2014SE059 **増田 大輝**

- セル色の変更
- 遷移規則の変更
- スコア表示
- プレゼン資料作成

2014SE089 **柴田 心太郎**

- README，MakeFile の作成
- Gameover の表示
- セル数のカウントと表示
- 本稿 1 章及び 3 章
- 本稿の構成，まとめ

2014SE112 **安田 拓也**

- 既存プログラムの調査
- スコアのリセット
- スコア表示カラー変更
- 不要なボタンの消去
- 本稿 5 章

2014SE114 **矢澤 拓海**

- ゲームの大枠提案
- 遷移規則の変更
- セルの回転
- 外周の実装
- 本稿 4 章

7　参考文献

1) 森田 憲一:可逆計算,pp.87–92, 近代科学社 (2012).

2) 「Java でライフゲームを作る」,〈 http://qiita.com/natmark/items/5a0d05625d2c0e73777d 〉

3) 「Making for Java Programe」,〈 http://profesores.elo.utfsm.cl/ agv/elo329/Java/javamakefile.html 〉

4) 「README の書き方」,〈
http://www-or.amp.i.kyoto-u.ac.jp/algo-eng/db/template/README.template 〉

5) 「とほほの WWW 入門」,〈 http://www.tohoho-web.com/java/index.html 〉