2021 - 後学期

プログラミング演習
グループプログラミングレポート

お絵かきロジック

学科	Ｉ類
グループ番号	12
2010304	櫻井陽允
2010327	佐藤大樹
2010335	佐野遵平

１．プログラムの概要

「お絵かきロジック」というゲームを自分たちの手で一から作ることを目的とした。難易度選択、一人で行うか二人で行うかの選択、詳しいゲーム説明をブラウザで表示するなどをゲーム開始時に行えるようにした。ゲーム開始後には、残りライフ制のお絵かきロジックの他に、経過時間の表示、BGM の再生とオン・オフの切り替えを実装した。スタート画面とプレイ中の画面は図１、図２のようなものである。

MVC モデルで構築し、Model 部分を佐野、View を佐藤、Controller を櫻井が担当した。Google Drive でファイルの共有、LINE で授業時間外でのアイデアの共有、Zoom で打ち合わせなどを行った。主に授業時間内でアイデアを共有し、次回の授業までに各自が何をやってくるかを決め、プログラムを作成していった。

文責：櫻井

図 1　スタート画面

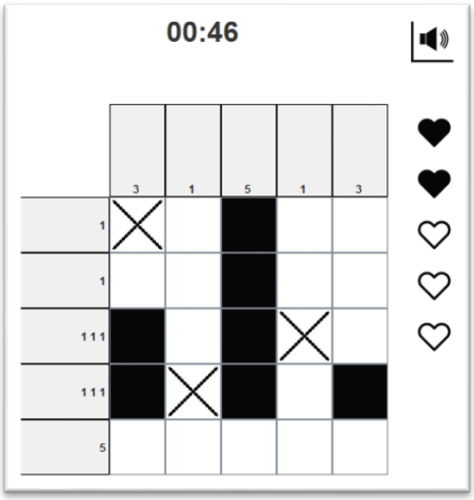


図 2　プレイ中の画面

２．設計方針

プログラムは MVC モデルで構成した。まず、それぞれのマス目に対応するような、正解のマスの情報を持っている二次元配列と、プレイヤーが押したマスの情報を持つ二次元配列を作った。プレイヤーが二次元配列のどこのマスをクリックしたかの情報を Controller が Model に送り、Model がその情報と正解のマスの情報を持っている二次元配列とを照らし合わせることで正誤判定を行い、Model がその結果を View に送り、View が描画した。Model で正解数、不正解数の管理を行い、それが一定数を超えるとゲームクリア・ゲームオーバーの判定を下し、それに応じて View で描画した。この Model が保持している正解数を共有することで、どちらのプレイヤーが早く問題を解けたかを競い合うネットワーク対戦を実現した。また、二次元配列は容易に大きさを変更できるため、固定の大きさに依存しないプログラムを作成したことで、難易度選択を可能にした。

下に示したクラス図をもとに各クラスの説明をしていく。NonogramModel とその周辺のクラスについて説明を行う。まず、NonogramModel クラスは先述のとおり NonogramController と NonogramFrame を仲介してゲームの判定を行うクラスである。NonogramServer/NonogramClient クラスは、通信対戦を行う際に使われる通信機能を実装したクラスである。NonogramFileNames クラスは、お絵描きロジックの問題データを外部のファイルから読み込んだり、難易度を設定したりするプログラムである。

次に NonogramController クラスとその周辺のクラスについて説明を行う。ButtonPanel クラスは、二次元配列のボタンを作成するクラスであり、NonogramController クラスでは、プレイヤーがクリックしたボタンの情報を NonogramModel クラスに送っている。

View に位置する NonogramFrame クラスでは、プレイ画面の作成とゲームオーバーとゲームクリアの画面を作成し、Model から送られる情報によってそれらの画面を描画したり、変更を加えたりしている。StartmenuFrame クラスでは、スタート画面を作成し、AudioPlay クラスでは、音楽ファイルを読み込み、音量調節をして再生するクラスとなっている。

文責：櫻井

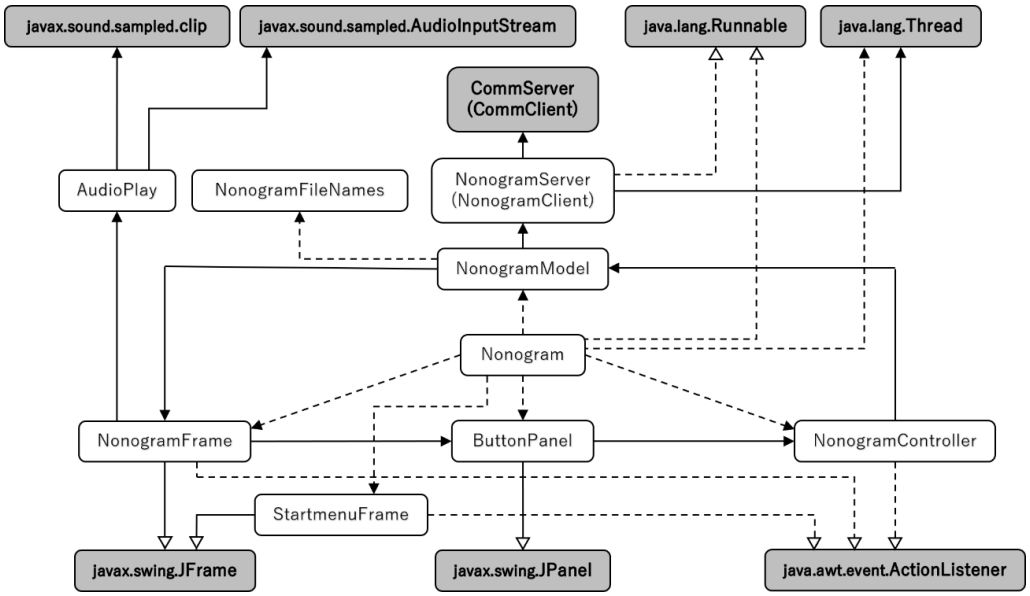
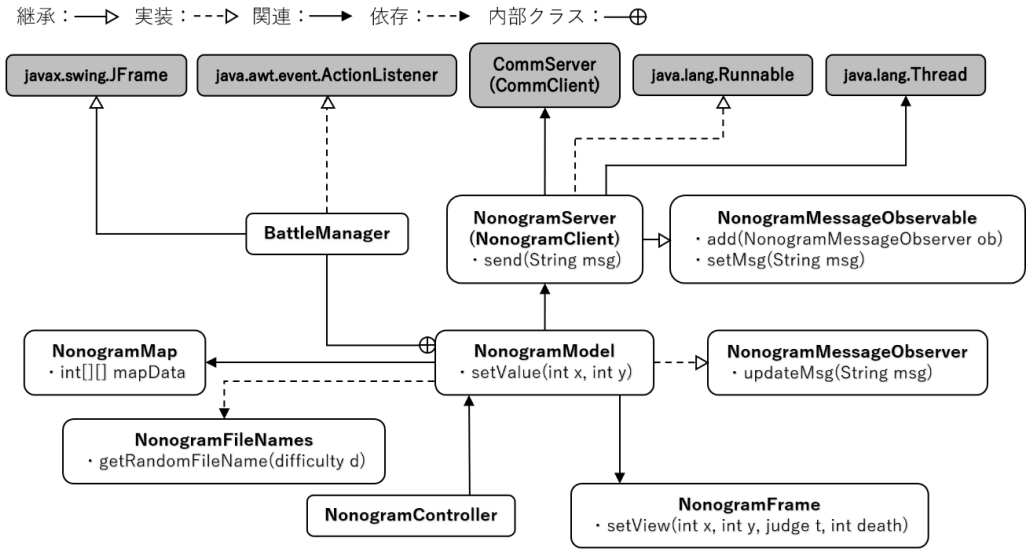


図 3　クラス図

3．プログラムの説明

3.1.佐野担当箇所



Model 周辺のクラス図

3.1.1　Nonogram クラス

このクラスは、ゲームのメインメソッドを含んだクラスである。ゲームのスタート画面を表示したり、ゲームを開始したりする処理を行う。また、スタート画面で指定された難易度に応じた、お絵描きロジックのボタン群のパネルを作成する。さらに、MVC モデルの Model、View、Controller の役割を担う NonogramModel、NonogramView、NonogramController クラスのインスタンス化も

行う。

3.1.2　NonogramModel クラス

NonogramModel クラスは、本ゲームの製作で使用した MVC モデルにおける Model 部分を実現するためのクラスである。本クラスは、主に次の二つの役割を担っている。

一つ目は、お絵描きロジックの問題データを扱う役割である。お絵描きロジックのゲームのどのマス目が正解であるか、あるいは不正解であるかを表す配列を扱う NonogramMap クラス（3.1.3 章）を保持することで、ゲームの中枢的な役割を担っている。

二つ目は、MVC モデルの Controller から受け取った情報を判定し、その判定結果を View に渡す役割である。この役割は後述の「setValue」メソッドが担っている（1 章）。

3.1.2.1 コンストラクタ

NonogramModel のコンストラクタは、オーバーロードを用いたことで引数の異なる 2 つのものが定義されている（現在不使用のものを含めると 3 つ存在する）。

一つ目は、引数として NonogramFrame クラス、ButtonPanel クラスを受け取るものである。NonogramFrame クラスは MVC モデルにおける View であり、この NonogramFrame のインスタンスをこのクラスのメンバ変数として保持することで、Model から View にアクセスすることが可能となっている。また、お絵描きロジックのマス目の大きさを保持する ButtonPanel クラスを受け取ることで、お絵描きロジックの問題のマス目の数を取得し、その値をメンバ変数に設定している。

```
NonogramModel(NonogramFrame v, ButtonPanel b)
{
    this.v = v;    // View をメンバ変数に保持
    this.mapWidth = b.mapWidth;    // お絵描きロジックの問題のマス目の幅を設定
    this.mapLength = b.mapLength;    // お絵描きロジックの問題のマス目の長さ（高さ）を
設定
}
```

二つ目のコンストラクタは、一つ目のコンストラクタの引数である NonogramFrame, ButtonPanel に加え、問題の難易度を表す列挙型の変数 difficulty と、ネットワーク対戦モードか否かを表す boolean 型の変数を受け取るものである。このコンストラクタでは、一つ目のコンストラクタの動作に加えて次の処理を行う。

一つ目は、指定された難易度の問題を外部のファイルから取得する処理である。お絵描きロジックのいくつかの問題ファイルの中から、指定された難易度のものをランダムで 1 つ選ぶ処理を行う。また、その問題ファイルの情報から、お絵描きロジックのゲームを解く際に必要なヒントを作成する。

二つ目は、ネットワーク対戦モードかそうでないかを設定する処理である。対戦モードの場合、内部クラスである BattleManager クラスのインスタンス化を行い、対戦が開始するまで待機する処理を行う。

3.1.2.2. void setValue(int x, int y)

このメソッドは public であり、MVC モデルの Controller から呼び出される。引数は、お絵描きロジックの押されたボタンの横、縦の番号(x,y)である。このメソッドでは、Controller から受け取った情報の正誤判定をし、その結果を View に送る役割を担っている。これらの処理は主に次のような順番で行われる。

まず、そのボタンが以前にも押されたことのあるボタンであるかどうかを調べる。すでに押されたことのあるボタンである場合、このメソッドでの処理を終了し、この後の処理は行わない。初めて押されたボタンであった場合、この次の処理を行う。

次に、押されたボタンが正解であるかどうかを判定する。お絵描きロジックの不正解のボタンが押されてしまったと判定した場合、View に不正解ボタンが押されたことと、現時点での間違えたボタンが押された回数の合計数を伝える。正解のボタンが押されたと判定した場合は、View に正解のボタンが押されたことを伝える。また、対戦モードの際は対戦相手に自分の残りの正解のマス目の数の送信を行う

3.1.2.3. void updateMsg(String msg)

このメソッドは、対戦モードのときに相手から送られてきた残りの正解マスの数を、自分の画面に表示させるためのものである。インターフェースである NonogramMessageObserver で宣言された Public なメソッドであり、相手が正解のボタンを押したときに NonogramMessageObservable から呼び出される。また、もし相手が先にお絵描きロジックを完成させた場合、View に自分が負けたことを伝えることで敗北画面の表示を行う。

```
public void updateMsg(String msg)     // Observable から呼び出される
{
    v.setTimeScreen(msg);     // View に相手の残りの正解マス数を表示させる
    int num = Integer.parseInt(msg);      // 相手の残りの正解マス数を int に
    if (num == 0)     // 相手が終了したとき
    {
        didFinish = true;     // 負けフラグを立てる
          // view に負けたことを伝える
        v.setView(0, 0, NonogramFrame.judge.lose, deathCount);
     }
}
```

3.1.2.4. BattleManager クラス（内部クラス）

このクラスは、ネットワーク対戦モードの際に対戦に関する設定を行う画面を表示するクラスであり、JFrame を継承している。サーバー・クライアントなどを設定するための画面や、通信待機画面の表示を行う。接続が確立した後には、この JFrame の画面を閉じる。

サーバーが選択された際は NonogramServer クラスを、クライアントが選択された際は

NonogramClient クラスを NonogramMessageObservable 型のメンバ変数にアップキャストする。

3.1.3. NonogramMap クラス

このクラスは、お絵描きロジックの問題を表す二次元配列を保持しているクラスである。この二次元配列 mapData では、お絵描きロジックの正解のマスを 1、不正解のマスを 0 としてお絵描きロジックの問題を扱っている。また、外部のテキストファイルの文字列をこの二次元配列に読み込む役割も担っている（ReadMapData メソッド）。

コンストラクタの引数としてお絵描きロジックの問題ファイルのファイル名を渡すことで、MapData フォルダの中にあるファイル群から対象のファイルを見つけ出し、問題の配列を作成する。

```
// 問題となるお絵描きロジックのデータ（1 のところはタッチして OK、0 のところをタッチすると罰則）
public int[][] mapData = new int[mapMaxLength][mapMaxWidth];
```

3.1.4. NonogramFileNames クラス

本クラスは、お絵描きロジックの問題を表すテキストファイルをプログラムに登録しやすくするために作成したクラスである。メンバ変数として、難易度別に easy、normal、hard それぞれの String 型の ArrayList が存在している。これらの ArrayList にファイル名を登録することで、お絵描きロジックの問題ファイルをプログラムに設定することができる。このクラスは NonogramModel クラスによってインスタンス化される。

```
// 難易度別のファイル名のリストを保存する ArrayList
private ArrayList<String> easy = new ArrayList<>();
private ArrayList<String> normal = new ArrayList<>();
private ArrayList<String> hard = new ArrayList<>();
```

3.1.4.1. コンストラクタ

コンストラクタでは、ArrayList にお絵描きロジックの問題ファイル名の追加を行っている。もし新たなお絵描きロジックの問題を作成した場合、<その問題の難易度(easy, normal, difficult のいずれか)>.add(“<ファイル名>”); と記述することで、このプログラムにファイルの登録をすることが可能である。

```
NonogramFileNames()    // ここにファイル名を設定してください
{
    // easy モードのファイル名
    easy.add("yama.txt");
    easy.add("yen.txt");
    easy.add("king.txt");
    // normal モードのファイル名
    normal.add("sound.txt");
```

```
    // difficult モードのファイル名
    hard.add("uec.txt");
}
```

3.1.4.2. String getRandomFileName(Nonogram.difficulty d)

このメソッドは、引数で指定された難易度の問題ファイルのうちからランダムで 1 つ選び、そのファイル名を返す動作をする。このメソッドによって、ゲーム開始時に、指定された難易度の問題が毎回ランダムで選択されるようになっている。public のメソッドであり、NonogramModel クラスから呼び出される。

3.1.5. NonogramMessageObservable クラス

このクラスは、後述の NonogramServer クラスや、NonogramClient クラスで継承されることが想定された抽象クラスである。ネットワーク対戦モードにおいて、相手からの通信を受け取った時に NonogramMessageObserver インターフェースを実装したクラスに通知を行う役割を担う、Observer パターンを参考に作成されたクラスである。

このクラスは抽象クラスであり、抽象メソッドとして send メソッドを宣言している。したがって、このクラスの子クラスである NonogramServer クラスや NonogramClient クラスには、send メソッドが必ず定義されていることが保障される。これにより、このクラスのインスタンス化を行う NonogramModel クラスからは、NonogramMessageObservable 型の変数に NonogramServer クラスや NonogramClient クラスをアップキャストすることで、自分がサーバーであるかクライアントであるかを気にせずに send メソッドを利用することができるようになっている。

3.1.5.1. void add(NonogramMessageObserver ob)

このメソッドは、引数として渡された NonogramMessageObserver インターフェースを実装したクラスを、この Observable クラスに登録する役割を持っている。ネットワーク対戦モードで相手からメッセージを受信した際には、この add メソッドで登録されたクラスの updateMsg メソッドが呼び出される。

```
// Observer の記憶には ArrayList を使用(ただし今回のプログラムでは Observer は一つだけ)
private ArrayList<NonogramMessageObserver> Observer = new ArrayList<>();

public void add(NonogramMessageObserver ob)
{
    Observer.add(ob);
}
```

3.1.5.2. void setMsg(String msg)

このメソッドが呼び出されると、引数として渡された文字列が、上述の add メソッドで登録され

たすべてのクラスの updateMsg メソッドへ通知される。今回のプログラムでは、ネットワーク対戦で対戦相手からの文字列を受信した際にこのメソッドが呼ばれることで、その受信した内容を NonogramModel へ通知する役割を担っている。アクセス修飾子は protected であり、このクラスを継承した NonogramServer クラスや NonogramClient クラスから呼び出されることを想定している。

```
protected void setMsg(String msg)
{
    // add された Observer 実装クラスの updateMsg を呼び出し
    for (int i = 0; i < Observer.size(); i++)
    {
        Observer.get(i).updateMsg(msg);
    }
}
```

3.1.5.3. void send(String msg)

このメソッドは abstract の抽象メソッドであるため、メソッドの中身が記述されていない。したがって、このクラスを継承した NonogramServer クラスや NonogramClient クラスで send メソッドの中身を定義することが必要になっている。この抽象メソッドの定義によって、NonogramMessageObservable 型のインスタンスから send メソッドを呼び出すことが可能になっている。

```
abstract public void send(String msg);    // オーバーライドが必要
```

3.1.6. NonogramServer クラス・NonogramClient クラス

この二つのクラスは、本科目のホームページの、「プログラミング演習 FAQ グループプログラミング 【ネットワークプログラミング編】」の資料に掲載の「メッセージ送受信プログラム」のサンプルコードである「ChatServer」クラスと「ChatClient」クラスを参考に作成したものである。

ネットワーク対戦モードの際には、自分のお絵描きロジックの問題を解く処理と、相手からのメッセージの受信を待つ処理を同時に行わなければならない。したがって、これらのクラスでは Runnable インターフェースを実装し、Thread クラスをインスタンス化することで、メッセージの受信のための処理を他の作業と並行して行えるようになっている。

これら二つのクラスは前述の NonogramMessageObservable クラスを継承しているため、親クラスの add メソッドや setMsg メソッドを利用することが可能である。また、親クラスで抽象メソッドとして宣言された、メッセージを相手に送る処理をする send メソッドの定義が記載されている。

3.1.6.1. NonogramServer クラス

このクラスは、ネットワーク対戦モードの際に自身がサーバーとなって、クライアントに対し通信を行う役割を担っている。本科目のホームページに掲載のサンプルコードである CommServer クラスを利用することで、クライアント側にメッセージを送信する。

コンストラクタでは、引数として通信に使用するポート番号を指定する。

3.1.6.2. NonogramClient クラス

このクラスは、ネットワーク対戦モードの際に自身がクライアントとなって、サーバーと通信を行う役割を担っている。本科目のホームページに掲載のサンプルコードである CommClient クラスを利用することで、サーバー側にメッセージを送信する。

コンストラクタでは、通信相手のサーバーのホスト名とポート番号を指定する。

文責：佐野

3.2 櫻井担当箇所

3.2.1 ButtonPanel

このクラスはお絵かきロジックの各マスを、二次元配列を利用してボタンを配置することで実現するクラスである。フィールドは、次のようになっている。

```
public final int mapMaxWidth = 50;      // マス目の最大の幅
public final int mapMaxLength = 50;      // マス目の最大の縦の長さ

public int mapWidth = 5;      // マス目の幅、最初は 5 とする
public int mapLength = 5;      // マス目の縦の長さ、最初は 5 とする

public JButton[][] buttons = new JButton[mapMaxLength][mapMaxWidth];
```

それぞれについて説明していく。7 行目と 8 行目の mapMaxWidth,mapMaxLength は、マス目の幅と縦の長さの最大値を、定数を代入し決めている。10 行目と 11 行目の mapWidth,mapLength は、与えられた問題のマス目の幅と縦の長さを保存しておくためのフィールドであり、5 で初期化されている。13 行目の buttons はボタンの二次元配列であり、要素数は mapMaxWidth,mapMaxLength として初期化している。

コンストラクタは、プレイヤーが選択した難易度によるマス目の横幅と縦の長さを引数として受け取る。そして、フィールドの mapWidth と mapLength にそれら引数の値を代入することと、mapWidth と mapLength の大きさの情報をもとに二次元配列 buttons の必要な要素にボタンを作成、そしてそのボタンをパネルに貼り付けることを行っている。工夫点としては、ボタンを Panel に貼りつけるだけならボタンの二次元配列 buttons は必要なかったが、各ボタンに対して、Controller でのプレイヤーがどのボタンをマウスでクリックしたのかの判別や View でのボタンの色の変更などを実現するために二次元配列を使用した点である。また、ボタンの二次元配列 buttons の初期化は定数で行わなければいけないので、mapMaxWidth、mapMaxLength を定数で用意し、これを初期化に用いて、mapWidth,mapLength は選ばれた問題のマス目に応じて更新するようにした点も工夫した。

3.2.2 NonogramController

このクラスでは、ButtonPanel からマップの情報を受け取り、プレイヤーにクリックされたボタンがどこのボタンであるのかを座標変換してモデルに送るということを行っている。フィールドは、NonogramModel model を定義している。コンストラクタでは、まず、引数として受け取った NonogramModel 型の変数 m をフィールドの model に代入している。その後引数で受け取った ButtonPanel 型の変数 b 内の mapWidth,mapLength の情報をもとに b 内のボタンの二次元配列 buttons の各ボタンに addActionListener でクリックされた時の処理を追加している。具体的には、各ボタンの二次元配列の要素番号を 2 つの文字列 s1、s2 に文字列変換して代入し、「,」の文字を s1 と s2 の間に入れつなげたものを s3 として、setActionCommand で各ボタンに s3 の文字列を付けておく。そして、actionPerformed メソッドで押されたボタンに付いている s3 を取得し、s3 の「,」より前の文字列と「,」より後の文字列をそれぞれ xstr,ystr に取り出し、それを整数に変換し、model.setValue で Model に押されたボタンの x, y 座標を整数で渡している。

工夫した点としては、Model にどこのボタンが押されたかの情報を x,y 座標というシンプルな情報で送り、その後 Model での正誤の判断などをしやすくできるようにしたことである。具体的に、setActionCommand では、二次元配列の各ボタンに一つの文字列しか情報として付けることができないので、x 座標と y 座標を「,」で繋いで s3 とし、それを一つの情報として付け、actionPerformed 内で x,y 座標を復元できるようにし、x,y 座標で Model に情報を送ることを実現した。また、ButtonPanel の工夫した点で先述したとおり、ボタンの二次元配列 buttons を用意したことで、押されたボタンの要素数から x,y 座標を割り出すことができたので、二次元配列の利点をうまく活かせたと思う。

文責：櫻井

3.3 佐藤担当箇所

3.3.1 NonogramFrame

このクラスはプレイ画面、ゲームクリアやゲームクリアの画面を作成する。図 A にはプレイ画面での 9 つのパネルと中心のパネルに入る 4 つのパネルの配置を示している。

まず初めに NonogramFrame メソッドについて説明する。p1 や p2 など p○と p_all の 9 つのフィールドは frame の上に貼り付けられるパネルで、図 A の太線のように frame を 9 つに分割し、p2 には経過時間を表示する JLabel 型 time、p3 には BGM のオンオフを切り替えられる JButton 型 SoundOnOff、p_all にはヒントを含めたプレイ部分、p5 には残りライフ数の機能を果たす部品が載せられる。p5 では初めは Heart02 を表示する HeartlifeImage○を 5 つ載せている。p_all では図 A の細線のように 4 つに分割され、左上の空白部分の blank、上のヒントを表示する Width、左のヒントを表示する Length、実際に押すボタン群の ButtonPanel 型の b がある。Width や Length は b の縦と横のサイズの配列の JLabel[]型の Width_sub と Length_sub を載せている。

次に Override された setVisible メソッドでは、プレイ画面に移行したときに BGM が流れるようにしたものである。これがないとネットワーク対戦をした際にポート番号などを入力するときから BGM が再生してしまうため、作成した。

次に actionPerformed メソッドでは、時間関係の部分と BGM関係の部分に分かれている。時間関

係の部分では 1 秒ごとに 1 増える int 型 seconds を用いてテキストを設定し、時間表示を可能とし、BGM関係の部分ではボタンの状態によってSoundOnもしくはSoundOffのIconをボタンに設定し、BGM を再生もしくは停止させる。

次に setView メソッドでは、View から送られてくる情報がどんな種類なのかを表す judge 型 type と(x, y)座標、残りライフ数を表す deathCount を引数としている。type が correct の場合、正解のマスをクリックしたということなので、送られてきた座標のボタンを黒く塗る。type が miss の場合、不正解のマスをクリックしたということなので、送られてきた座標のボタンに×を表示する GIF を設定する。不正解すると残りライフ数は減っていくので、不正解数を表す deathCount によってHeartlifeImage○を 1 つ削除して、新たに HeartdeathImage○を追加することで残りライフ数が表示される。deathCount の値が 4 に達すると、gameover メソッドが実行されるようになっている。

次に gameover メソッドと gameclear メソッドでは、経過時間と BGMを止め、それぞれ専用の BGM を再生する。frame 上のものをすべて除去し、revalidate、repaint した後、専用のパネルを frame に追加している。

次に gameLose メソッドと gameWin メソッドでは、gameover メソッドと gameclear メソッドの最後に追加するパネルのテキストを変えて使用している。

次に setHintLine と setHintColumn はヒントとそれを表示する行・列の位置を受け取り、その位置にヒントを設定するものである。左のヒント欄は横書きだが、上のヒント欄は縦書きなので、HTML を用いて実装した。

最後に setTimeScreen ではネットワーク対戦時に相手の残り正解数を経過時間のところに表示するものである。

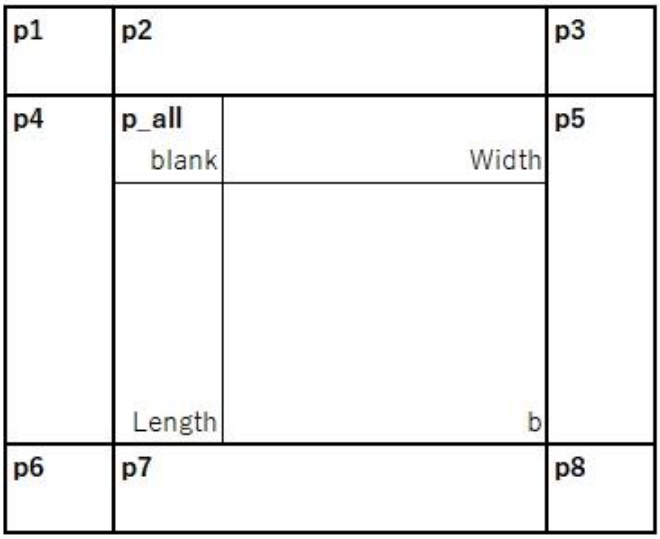


図 A プレイ画面のパネル配置図

3.3.2 StartmenuFrame

このクラスはスタート画面を作成するクラスである。

まず、StartmenuFrame メソッドでは、スタート画面の背景画像を貼り、3 つの難易度とルール説

明のための「How to play」の合計 4 つのボタンとネットワーク対戦のためのチェックボックスをそれぞれ配置している。テキストは HTML で書いた。

次に actionPerformed メソッドでは、上の 4 つのボタンが押された際の動作がかかれている。「How to play」のボタンが押されたらルール説明の PDF の URL をブラウザで開くようになっている。難易度のボタンが押された際は boolean 型の didStart を true にして difficulty に難易度を設定したのち、Started メソッドで didStart を、getDifficulty で difficulty を返すことでどの難易度で始まったのかどうかを別の Nonogram クラスに通知されるようになっている。

最後の isBattleMode もネットワーク対戦のチェックボックスの状態を Nonogram クラスに返すことでネットワーク対戦をするのかどうかを通知している。

3.3.3 Audioplay

このクラスは BGM を再生する設定をしている。ファイルの取得、クリップを展開、音量を下げる操作を行っている。授業では取り上げられていない内容なので以下のサイトを参考にしたが、そのままでは上手く動作しなかったため、ほとんどの部分を書き換えている。
https://nompor.com/2017/12/14/post-128/

4．実行例

スタート画面は図 1 で示したとおりである。Easy モードを選択したら図 2 のような 5×5、Normal モードを選択したら図 4 のような 10×10、Hard モードを選択したら図 5 のような 15×15 でゲームが開始される。図 2 のように正解マスをクリックしたら黒く塗られ、不正解マスをクリックしたら、バツが表示される。ライフが 5 つ減る前に正解のマスをすべてクリックしたらゲームクリア画面、ゲームクリアする前にライフがすべてなくなったらゲームオーバー画面が図 6 のように表示される。ネットワーク対戦を行う場合は、図 1 のスタート画面の右下の「Battle Mode」のチェックボックスにチェックを入れ、任意の難易度を選択する。すると、図 7 の設定画面が表示され、設定が完了すると対戦が図 8 のように開始される。画面の上部には相手の勝利に必要な残りの正解マス数が表示される。先に正解のマスをすべてクリックしたプレイヤーの画面に「You Win」、もう一方のプレイヤーの画面に「You Lose」が図 9 のように表示される。

文責：櫻井

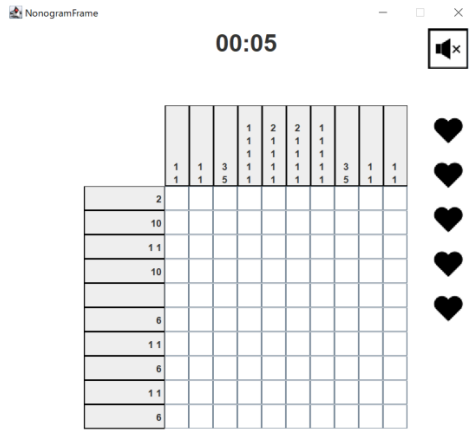


図 4　10×10 のゲーム開始画面

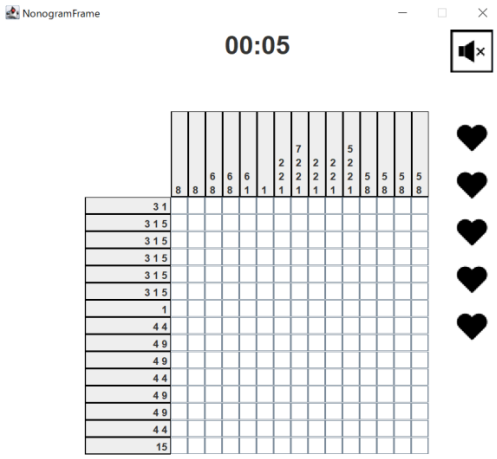


図 5　15×15 のゲーム開始画面

図 6　ゲームクリア・ゲームオーバー画面

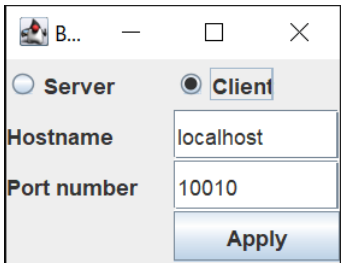


図 7　ネットワーク対戦の設定画面

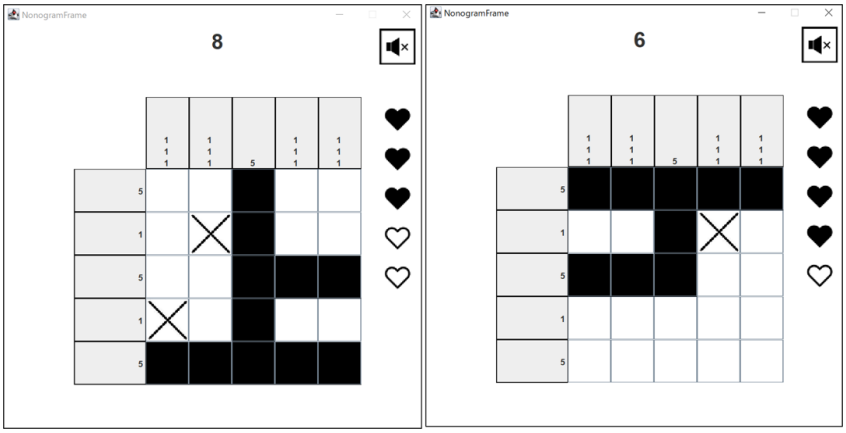


図 8　ネットワーク対戦時の画面

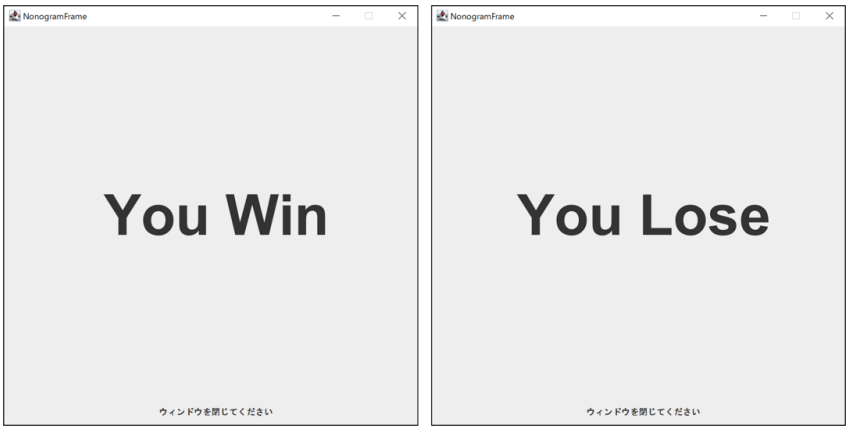


図 9　ネットワーク対戦時の勝利・敗北画面

5．考察

私達が作成したプログラムでは、「お絵かきロジック」に最低限必要なマスの選択、問題に合わせたヒント数字の表示などの機能を実装することができた。それに加えて、経過時間の表示、BGMとBGMのオン・オフの切り替え、難易度選択、遊び方の説明、ネットワーク対戦など、当初の予定よりも多くの機能を実装できたため、より満足のいくものができた。グループワーク初回からゲームの大まかな構成とそれぞれの能力に応じた役割分担ができ、授業で学んだMVCモデルに沿って作

業を進めていった。MVCモデルでは、グループワークでpublicな関数を共有した以降は、各自で独立した作業を行ったため、MVCモデルの利点を最大限活かした開発を行えたと考えられる。

しかし、時間的な制約や技術的な問題から以下のようなことが実装できなかった。1つ目は、お絵かきロジックの正誤選択のモードを切り替える機能である。今回作成したゲームは、正解のマスを探すゲームであり、マスをプレイヤーがクリックし、そこが正解だったら黒く塗りつぶされ、不正解であったらバツが表示されライフが一つ減る仕様にした。逆に、マスをプレイヤーがクリックし、そこが正解だったら黒く塗りつぶされるがライフが一つ減り、不正解であったらバツが表示されるだけの不正解のマスを探すモードを作り、2つモードの切り替えを実装すれば、よりゲームの操作性や面白みが増したのではないかと考えられる。この機能を実装できなかった理由として、モードの切り替え情報をModel、Controller、View間で共有することが、技術的な問題でできなかったことが挙げられる。2つ目は、ネットワーク対戦で二人に同じ問題を出題する機能である。ネットワーク対戦で遊びの幅を広げることができたが、対戦する二人で同じ問題が共有されないため、問題の難易度はある程度同じであるものの、公平性が失われる場合があった。この機能を実装できなかった理由は、対戦開始前に二人に対し、問題の情報を共有するプログラムを作成する時間が足りなかったことであると考えられる。

上述した改良点に加え、その他にも改良点や実装したかった機能はあったが、それは以下「6.各自の反省と感想」で触れるものとする。

文責：櫻井

6． 各自の反省と感想

グループでの作業を通して振り返ると、MVC モデルで作業分担を行うことができ、授業時間内だけでなく、授業時間外でもコミュニケーションを取り、全体的にはスムーズに作業を進めることができたと思う。MVC モデルでうまく作業を分担することができたため、各自で独立して作業することができたことと、チーム内での積極的なコミュニケーションを取れたこと、この 2 つがスムーズにプログラム開発が進んだ要因だと考える。しかし、実装したかった機能をすべて実装できなかったことや、作業分担に偏りがでてしまったことなどは反省点として挙げられると思う。私の担当部分でやり残したこと、実装できなかったことは、各マスのドラッグでの複数選択の機能である。今回作成したゲームでは、マス一つずつをクリックして、そのマス一つずつ正誤の判定が行われたが、私が実装したかった機能は、マウスでドラッグしながらカーソルがボタンの上を通ったら、上を通られたボタンすべては、クリックされたと判定され、正誤判定が行われる機能である。この機能を実装できなかった理由としては主に、実装するためのアイデアの不足、私の技術不足が挙げられる。インターネットなどで、こちらの機能を実装するためにいろいろと調べてみて、自分なりに試行錯誤を重ねたが、結局実装することはできなかった。この点はとても残念に思うが、これから技術力を磨き、将来この機能を実装させる事ができたら修正したい。また、今回のグループでのプログラム開発を通じで、オブジェクト指向のメリットを実感することができた。具体的には、MVC モデルで分担し、各自作業を独立させることで、プログラムを修正する際などに他の機能を気にせず、簡単に修正を加えることができ、オブジェクト指向の変更の容易さを感じ、多人数での開発には欠かせない考え方だなと思った。また、今回のプログラム開発では最初から仕様が完璧に決められたわけではなく、作業の途中で仕様を少し変更しながら、進めた。今回は、3 人での

プログラム開発だったので、それでも問題はなかったが、将来経験すると思われるより多くの人が関わるプログラム開発では、最初に仕様などをきっちり決め、各自で作業を進めることがとても重要だと感じた。「プログラミング演習」の授業を通しては、java を一から学んだが、演習を重ねるうちにある程度理解しコードを書けるようになり、チームメイトの協力有りきだが、ゲームを完成させることができたので、とても良い経験になったと思う。また、授業資料に情報が簡潔にまとめられていることや、TA の方が気軽に質問に答えてくれるなどの手厚いサポートがありとても助けられた。この授業でゲームを実際に初めて自分で作成してみて、自分でゲームを作成することに興味が湧いたので、今回学んだことを活かしながらなにかゲームを作成することに挑戦できたらいいと思う。

文責：櫻井

グループワークの初回授業ではどんな人とできるのか不安と楽しみが混在したが、顔合わせをするとしっかりと意思疎通が取れる 2 人で安心した。その日にどのようなものを作るのかを話し合い、私が提案したゲームに 2 人が同意してくれて嬉しかった。今になって感じるが、程よい難易度のゲームが選択できたのではないかと感じている。そのゲームについて一番理解があったのは私なので 2 人にルールを説明したり、お互い気軽に連絡をとったりなどして順調に作業を進めることができた。プログラミングのスキルは大したものではないが、提案した者として責任をもって 2 人の意見をまとめることができたのではないかと感じている。反省点としては、余裕をもって物事を進めたため、最低限の機能とすこし程度の追加機能は実装できたが、実装したい機能をすべて実装することは時間的に困難だった。また、作業負担も均等に配分できず、大変だったり、暇だったりと偏りが生じてしまったことも反省点として挙げられる。
私が担当した View ではゲームがシンプルなので、それに似合うデザインを目指した。初めは古い Windows のウィンドウではなく、今のようにシンプルで高級感のある UI を作りたかったが、それは調べてもできそうになかったので断念して自分にできる範囲で努めた。これに関しては Netbeans などでできるみたいなので春休みの期間に触れてみようと思う。作成自体は色々と調整しながらではあったが、全体的にそんなに大変ではなく、一番苦労した点としては BGM の実装であった。サイトに書かれたものの多くはうまく機能せず、Java のマニュアルを見ながら使われているものの仕様を調べ、なんとか実装することができた。また、画像を元のサイズよりも小さく表示しようとすると当然画質が落ちてしまうので、綺麗な表示方法を今後調べる必要があると感じた。
学習内容に関する感想として、Java の基本やオブジェクト指向は授業時に理解できたが、MVC モデルについては授業でなんとなく理解した程度だったので、グループワーク開始時に復習する必要があった。Swing などの GUI プログラミングでは継承やインターフェース、レイアウトなどもプログラムを書きながら理解を深めることができたと感じている。しかし、プログラムで使わなかったラッパークラスや親クラスを呼び出す super、Observer パターンなどは理解が浅いので復習する必要があると感じている。
この授業に関する感想として、授業資料が図だけでなく、説明や参考のためにリンクが張ってあり、非常に分かり易く充実していたので詰まることなく学習を進めることができ、満足している。グループワークの時の担当していただいた TA の方も質問する機会はほぼなかったものの、進歩状況の確認の際に評価していただいたのでたいへん感謝している。1 年次にはなかった複数人で 1 つの

ゲームなどを作る授業は入学前から楽しみにしていたので、受講できて良かったし、実際の開発現場も複数人で行うと聞くのでその一歩目としてメンバーとコミュニケーションが取れ、有意義な時間を過ごすことができた。3 年次も何かしらプログラミングすると思うが、この授業で作ったものよりもクオリティの高いものが作れるようこれからも勉学に励みたいと思う。

文責：佐藤

本科目でのプログラミング演習は、これまでに大学の講義で扱ってきた C 言語には無かった、オブジェクト指向のプログラミングで使える様々なスキルを取得できたという点で、大変有意義なプログラミング製作の体験ができたと感じた。私はこれまでにオブジェクト指向型のプログラミング言語として C#を少しだけ扱ったことがあったが、独学だけではクラスの継承やデータ構造など、少し複雑な考え方を必要とするプログラミング方法から逃げてしまいがちになっていた。しかし、本科目でこれらを学習したことで、オブジェクト指向型のプログラミング方法の有用性を体験することができた。

今回のグループでのプログラミング作業では、一人だけで行うプログラミングに比べて特に次の点が重要であると知ることができた。一つ目は、あらかじめ仕様をできるだけ綿密に決めておくことである。今回のグループ作業では、発表会までに最低限プログラムを完成させることが重要だったため、仕様決定の段階では最低限の機能以外は想定していなかった。したがって、後になって新たな機能を追加しようとした際に、プログラムの大幅変更やグループの他のメンバーへ仕様変更のお願いをしなければいけなくなってしまったことが何度か生じた。今回はたった 3 人のグループだったため何とかなったが、これ以上大規模なグループで作業をする場合、少しの仕様変更が大勢の作業に影響を与える可能性があると考えられる。したがって、できるだけ作業途中での仕様変更や機能の追加が無いよう、あらかじめソフトウェアの完成形を想定した仕様決定をすることがグループでのプログラミングで大切なことであると考えられる。

二つ目は、できるだけ独立した分野ごとに役割分担をすることである。今回の作業では MVC モデルを用いて Model、View、Controller の 3 人で分かれて作業を行ったが、この方法はかなり独立性が高く役割分担に適していると感じた。なぜなら、今回の作業では、他の人の担当したプログラムを完全に理解する必要なくプログラムを完成させることができたからである。授業時間に全員が集まった時に Public のメソッドについて話し合った後は、それぞれが自分の担当だけに集中して作業することができたため、カプセル化やブラックボックス化がうまく機能していたと考えられる。また、GUI のソフトウェアでは View と Controller を完全に分離することが難しいが、今回は View と Controller の間に ButtonPanel というボタンが多数配置されたパネルを挟んだことで、ButtonPanel の ActionListener を Controller、ButtonPanel の色の変更を View が扱うという形でうまく役割分担することができたと感じている。したがって、このように独立性を高めるような工夫を仕様決定の段階で決めておくことがグループワークの作業効率を高めることにつながると考えられる。

三つめは、バージョン管理をすることである。今回の製作ではプログラムの共有に Google Drive を利用した。これにより、各自でプログラムの更新を行った際は、新しくフォルダを作るなどして

対応していた。しかし、今回はこの方法で特に大きな問題は生じなかったものの、決して効率的な方法ではなかったと考えられるため、GitHub などのツールを使ってみることも今後試してみたいと感じた。

今回のプログラミングの個人的な反省点は、主に次の二点である。一つ目は、機能を追加するのに伴ってプログラムが複雑になってしまったことである。例えば、今回の製作ではもともと予定していなかったネットワーク対戦機能の追加を行った。この機能を無理やり追加したことにより、ゲームの正誤判定をする部分のプログラムにネットワーク対戦モードであるかどうかの分岐の記述を書く必要性が新たに生じた。このように分岐が増えてしまうことはプログラムの可読性が下がるだけでなく、様々なパターンに対応しなければいけなくなるため、バグが生じることにもつながってしまう。このような問題を解決するには、あらかじめソフトウェアの仕様を綿密に決めておいたり、機能の追加を容易に行うことのできるようなプログラムを意識して作成したりすることが重要であると考えられる。

二つ目は、プログラムが雑なまま完成させてしまった箇所が生じてしまった点である。とりあえず動けば良いという気持ちや、後で直そうという気持ちのまま結局最後までそのまま直さなかった部分や、より効率的で分かりやすいプログラムに書き直せそうだと感じていながらも結局そのままにしてしまった部分が何か所か生じてしまった。これはプログラムの可読性が低下するだけでなく、実行効率が落ちてしまったりバグが生じてしまったりするリスクが高まるため、今後のプログラム製作においてはより丁寧な作業を心掛けたい。

文責：佐野

付録１：操作法マニュアル(ユーザーズマニュアル)

ルールがわからない場合は How to play で確認する。

付録 1.1　一人で遊ぶ場合

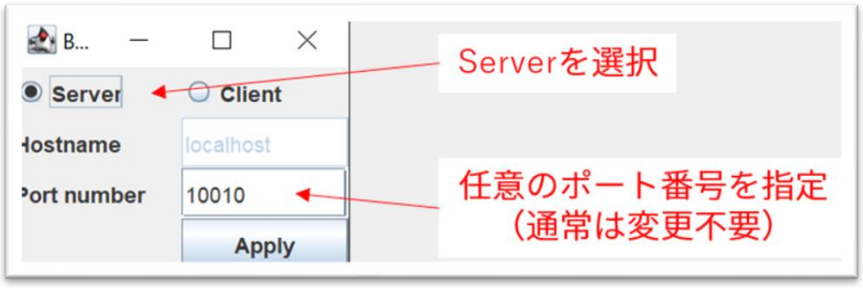
1. 右下の Battle mode にチェックがついてないことを確認して、難易度を選択する。

2. お絵描きロジックを遊ぶ。もし、BGM を切りたい場合は、右上のサウンドボタンをクリックする。

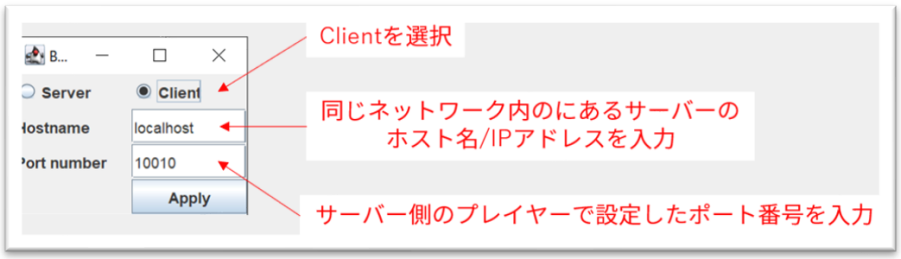
付録 1.2　二人で遊ぶ場合

1. 右下の Battle Mode にチェックを入れ、難易度を選択する。

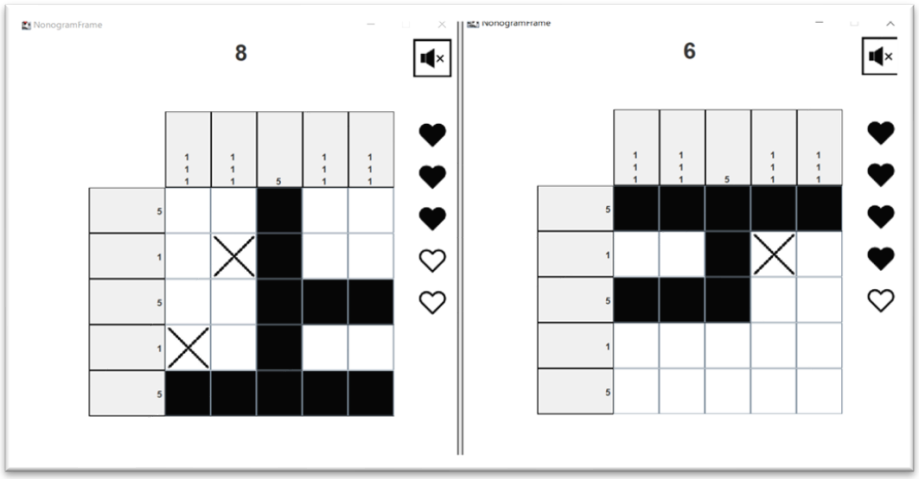
2. サーバー側の設定



3. クライアント側の設定



4. 二人で対戦する



付録 2:　プログラムリスト

ソースコード 1: Nonogram.java

```
1
2  import javax.swing.*;
3  import java.awt.*;
4  import java.awt.event.*;
5  import java.nio.FloatBuffer;
6
7  class Nonogram implements Runnable
8  {
9      public static boolean restart = false;
10
11     // 難易度を表す列挙型
12     public enum difficulty
13     {
14         easy,
15         normal,
16         hard
17     }
18
19     // スタート画面でゲーム開始ボタンが押されるまで待機する（スタートしたら
           trueを返す）
20     private boolean gameStart(StartmenuFrame start)
21     {
22         try{ Thread.sleep(100); }
23         catch(InterruptedException e) { return true; }
24         if(start.Started()) { return true; }
25         else { return false; }
26     }
27
28     // ボタンのパネルのサイズを難易度によって決定する
29     private ButtonPanel setbuttonPanel(difficulty d)
30     {
31         ButtonPanel b;
32         switch (d)
33         {
34             case easy:
35                 b = new ButtonPanel(5, 5);    // easyなら5マスx5マス
36                 break;
37             case normal:
38                 b = new ButtonPanel(10, 10);
39                 break;
40             case hard:
41                 b = new ButtonPanel(15, 15);
42                 break;
43             default:
44                 b = new ButtonPanel();
45                 break;
```

```
46          }
47          return b;
48      }
49
50      public void run()
51      {
52          /* スタート画面の表示 */
53          StartmenuFrame start = new StartmenuFrame();
54              start.repaint();
55          while (!gameStart(start));    // スタート画面が終わるまでループ
56          difficulty d = start.getDifficulty();    // modelに伝えるための難易度を取得
57          boolean battleMode = start.isBattleMode();    //
                modelに伝えるためのバトルモード選択の有無を取得
58          start.dispose();    // スタート画面を終了
59          /* ここまでスタート画面 */
60
61          // 難易度に応じたボタンのパネルの設定
62          ButtonPanel b = setbuttonPanel(d);
63
64          // View, Model, Controllerの生成
65          NonogramFrame view = new NonogramFrame(b);
66          NonogramModel model = new NonogramModel(view, b, d, battleMode);
67          NonogramController controller = new NonogramController(model, b);
68      }
69
70      public static void main(String argv[])
71      {
72          // 結局リスタート機能を実装しなかったためループは不使用
73          while (true)
74          {
75              restart = false;
76              Nonogram n = new Nonogram();
77              Thread t = new Thread(n);
78              t.start();
79
80              while (!restart)    // restartフラグが立つまで無限ループ
81              {
82                  try{ Thread.sleep(100); }
83                  catch(InterruptedException e) { break; }
84              }
85              try { t.join(); } // スレッドでの処理が終わるまで停止（多分要らない？よ
                    くわからない）
86              catch (InterruptedException e) { break; }
87              System.out.println("Restart");
88          }
89      }
90  }
```

ソースコード 2: NonogramModel.java

```
1
2  import javax.security.auth.login.FailedLoginException;
3  import javax.swing.*;
4  import java.awt.*;
5  import java.awt.event.*;
6  import java.io.*;
7  import java.security.cert.TrustAnchor;
8  import java.time.chrono.JapaneseChronology;
9  import java.util.ArrayList;
10 import java.util.Random;
11
12 // お絵描きロジックの配列を扱うクラス
13 class NonogramMap
14 {
15     private ButtonPanel b = new ButtonPanel();    // マップのサイズの最大値を取得するためbuttonPanelを参照
16     public final int mapMaxWidth = b.mapMaxWidth;
17     public final int mapMaxLength = b.mapMaxLength;
18
19     public int correctSquareNum = 0;    // 正解のマス目の数
20
21     // プレイヤーが開けたマスの情報（0ならまだタッチしてないマス、1ならすでにタッチして開けたマス）
22     public int[][] userMap = new int[mapMaxLength][mapMaxWidth];
23
24     // 問題となるパズルのデータ（1のところはタッチしてOK、0のところをタッチすると罰則（ライフが減るなど））
25     public int[][] mapData = new int[mapMaxLength][mapMaxWidth];
26
27     NonogramMap(String fileName)
28     {
29         // ユーザーマップの初期化
30         for (int i = 0; i < mapMaxWidth; i++) {
31             for (int j = 0; j < mapMaxLength; j++) {
32                 userMap[j][i] = 0;    // 初期値はすべてゼロ
33             }
34         }
35         System.out.printf("File: %s\n", fileName);
36         ReadMapData(fileName);    // 任意のファイル（MapData下に配置）
37     }
38
39     // 外部ファイルのデータを配列に落とし込む
40     // https://www.sejuku.net/blog/20924 などを参考に作成
41     private void ReadMapData(String fileName)
42     {
43         try
44         {
45             InputStream in = this.getClass().getResourceAsStream("/MapData/" + fileName);
46             BufferedReader br = new BufferedReader(new InputStreamReader(in));
```

```
47              String line;
48              int j = 0;
49              while ((line = br.readLine()) != null)    // 1行ずつ
50              {
51                  for (int i = 0; i < line.length(); i++)    // 1文字ずつ
52                  {
53                      mapData[j][i] = Character.getNumericValue(line.charAt(i));
54                      if (mapData[j][i] == 1) { correctSquareNum++; }
55                  }
56                  j++;
57              }
58              br.close();
59          }
60          catch (IOException e)
61          {
62              e.printStackTrace();
63          }
64      }
65  }
66
67
68  /**********  お絵描きロジックのModel部分 **********/
69  class NonogramModel implements NonogramMessageObserver
70  {
71      private NonogramFrame v;    // マスをタッチした情報をViewに送るためのメンバ
72      private NonogramMap map;    // お絵描きロジックの問題データのクラス
73
74      private int mapWidth;
75      private int mapLength;
76
77      private int deathCount = 0;    // 間違ったマスを押した回数を記録
78      private int correctSquareCount = 0;    // 残りの正解のマスの数を記録
79
80      private String[] hintLine;    // ヒントの文字列の配列（行）
81      private String[] hintColumn;    // ヒントの文字列の配列（列）
82
83      private NonogramMessageObservable msgOb;    // 通信対戦時のメッセージ受信用
84      private boolean didstart = false;
85      private boolean didFinish = false;
86
87      private final String _newLine = System.getProperty("line.separator");
88      private NonogramModel nonogramModel = this;    // 内部クラスからアクセスできる
            ように
89
90      /********** コンストラクタ **********/
91      NonogramModel(NonogramFrame v, ButtonPanel b)
92      {
93          this.v = v;    // Viewをメンバ変数に保持
94          this.mapWidth = b.mapWidth;    // お絵描きロジックの問題のマス目の幅を設定
95          this.mapLength = b.mapLength;    // お絵描きロジックの問題のマス目の長さ（
```

```
            高さ）を設定
96      }
97
98      // コンストラクタ（ファイル名が指定された場合）
99      // 現在不使用
100     NonogramModel(NonogramFrame v, ButtonPanel b, String fileName)
101     {
102         this(v, b);    // 上のコンストラクタを呼び出し
103         map = new NonogramMap(fileName);
104         hintLine = new String[map.mapMaxLength];
105         hintColumn = new String[map.mapMaxWidth];
106         makeHint();    // ヒントの作成
107     }
108
109     // コンストラクタ（難易度が指定された場合）
110     NonogramModel(NonogramFrame v, ButtonPanel b, Nonogram.difficulty difficulty,
            boolean battleMode)
111     {
112         this(v, b);    // 上のコンストラクタを呼び出し
113         // 難易度に応じたファイルをランダムで選択
114         NonogramFileNames f = new NonogramFileNames();
115         map = new NonogramMap(f.getRandomFileName(difficulty));
116         // ヒントを生成
117         hintLine = new String[map.mapMaxLength];
118         hintColumn = new String[map.mapMaxWidth];
119         makeHint();    // ヒントの作成
120         // ネットワーク対戦モードなら
121         if (battleMode)
122         {
123             // 対戦設定画面を準備
124             BattleManager bm = new BattleManager();
125             while (!didstart)    // 対戦開始まで待機
126             {
127                 try { Thread.sleep(100); }
128                 catch(InterruptedException e) {}
129             }
130             System.out.println("Connected");
131         }
132         v.setVisible(true);
133     }
134
135
136     /********** お絵描きロジックのヒント **********/
137     // ヒントを作成して配列に入れる
138     private void makeHint()
139     {
140         for (int i = 0; i < mapLength; i++)    // 行のヒント
141         {
142             hintLine[i] = "";
143             for (int j = 0; j < mapWidth; j++)
```

```
144             {
145                 if (map.mapData[i][j] == 0) { continue; }
146                 int count;
147                 for (count = 0; j < mapWidth && map.mapData[i][j] != 0; j++, count
                        ++);
148                 hintLine[i] += Integer.toString(count) + " ";
149             }
150             v.setHintLine(i, hintLine[i]);
151         }
152         for (int i = 0; i < mapWidth; i++)    // 列のヒント
153         {
154             hintColumn[i] = "";
155             for (int j = 0; j < mapLength; j++)
156             {
157                 if (map.mapData[j][i] == 0) { continue; }
158                 int count;
159                 for (count = 0; j < mapLength && map.mapData[j][i] != 0; j++, count
                        ++);
160                 hintColumn[i] += Integer.toString(count) + " ";
161             }
162             v.setHintColumn(i, hintColumn[i]);
163         }
164     }
165
166
167     /********** ゲームの中枢部分 **********/
168     // コントローラーからアクセスされるメソッド
169     public void setValue(int x, int y)    // 引数：ユーザーがタッチしたマスの座標
170     {
171         if (didFinish) { return; }    // 対戦終了している場合何もしない
172         if (map.userMap[y][x] >= 1)    // すでにタッチした場所をもう一度タッチした
                とき
173         {
174             return;    // 無視する
175         }
176         else if (map.mapData[y][x] == 0)    // タッチしてはいけないマスをタッチした
                とき
177         {
178             // ここに間違えた場所をタッチしたときの処理を書く
179             System.out.println("Incorrect position");
180             map.userMap[y][x] = 2;    // 2番は間違った場所をタッチしたことを記録
181             v.setView(x, y, NonogramFrame.judge.miss, deathCount);
182             deathCount++;
183             return;
184         }
185         // 正解のマスをタッチしたとき
186         map.userMap[y][x] = 1;    // ユーザーが開けた位置を記録
187         // ここにViewの更新処理を書く
188         v.setView(x, y, NonogramFrame.judge.correct, deathCount);    // 塗りつぶし
                た場所のtypeを1とする
```

```
189
190         this.correctSquareCount++;    // 正解数を+1する
191         sendCorrectCount();    // バトルモードなら正解数を相手に送信
192         if (this.correctSquareCount == map.correctSquareNum)
193         {
194             if (msgOb != null)    // バトルモードのとき
195             {
196                 v.setView(0, 0, NonogramFrame.judge.win, deathCount);
197             }
198             else { v.setView(0, 0, NonogramFrame.judge.clear, deathCount); }
199         }
200     }
201
202     // 機能実装予定だったが、結局未使用（
            Controllerから呼び出す文があるため消せない）
203     public void setType(int type) {}
204
205     // ゲームリスタートを行う関数、結局未使用
206     public void restartGame()
207     {
208         v.dispose();
209         Nonogram.restart = true;
210     }
211
212
213     /********** 通信対戦関連 **********/
214     // 残りのマス目の数を送信
215     private void sendCorrectCount()
216     {
217         if (msgOb != null && !didFinish)    // バトルモードかつ終了していないとき
218         {
219             msgOb.send(String.valueOf(map.correctSquareNum - this.
                    correctSquareCount));    // 残りの正解数を相手に送信
220         }
221     }
222
223     public void updateMsg(String msg)    // Observableから呼び出される
224     {
225         didstart = true;
226         v.setTimeScreen(msg);    // Viewに相手の残りの正解マス数を表示させる
227         int num = Integer.parseInt(msg);    // 相手の残りの正解マス数をintに
228         if (num == 0)    // 相手が終了したとき
229         {
230             didFinish = true;    // 負けフラグを立てる
231             // viewに負けたことを伝える
232             v.setView(0, 0, NonogramFrame.judge.lose, deathCount);
233         }
234     }
235
236     // 通信対戦の設定画面用フレーム
```

```
237     class BattleManager extends JFrame implements ActionListener, Runnable
238     {
239         String hostname = "localhost";
240         int portNum = 10010;
241 
242         private JPanel p;
243         private JRadioButton radioS, radioC;
244         private JTextField hostnameField, portNumField;
245         private JButton b;
246 
247         BattleManager()
248         {
249             // JFrameのボタンや文字列の設定
250             this.setSize(200,150);
251             this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
252             this.setTitle("Battle setting");
253 
254             p = new JPanel();
255             p.setLayout(new GridLayout(4, 2));
256 
257             radioS = new JRadioButton("Server", true); radioS.addActionListener(
                    this); radioS.setActionCommand("radioS");
258             radioC = new JRadioButton("Client", false); radioC.addActionListener(
                    this); radioC.setActionCommand("radioC");
259             hostnameField = new JTextField(hostname); hostnameField.setEnabled(
                    false);
260             portNumField  = new JTextField(String.valueOf(portNum));
261             b = new JButton("Apply"); b.addActionListener(this); b.setActionCommand
                    ("b");
262 
263             p.add(radioS); p.add(radioC);
264             p.add(new JLabel("Hostname")); p.add(hostnameField);
265             p.add(new JLabel("Port number")); p.add(portNumField);
266             p.add(new JPanel()); p.add(b);
267             this.add(p);
268 
269             this.setVisible(true);
270 
271         }
272 
273         public void run()    // 設定終了後
274         {
275             this.portNum = Integer.parseInt(portNumField.getText());
276             this.hostname = hostnameField.getText();
277             if (radioS.isSelected())
278             {
279                 // サーバーのとき
280                 System.out.println("I am server");
281                 msgOb = new NonogramServer(this.portNum);    // アップキャスト
282             }
```

```
283             else
284             {
285                 // クライアントのとき
286                 System.out.println("I am client");
287                 msgOb = new NonogramClient(hostname, portNum);    // アップキャスト
288             }
289             msgOb.add(nonogramModel);    // Observableにクラスを設定
290             sendCorrectCount();
291 
292             this.dispose();    // 接続が完了したらこのフレームは終了
293         }
294 
295         private void waitFrame()
296         {
297             // 通信待機画面を表示
298             this.setVisible(false);
299             this.remove(p);
300             this.revalidate();
301             this.repaint();
302             this.add(new JLabel("Waiting for connection..."));
303             this.setSize(300, 100);
304             this.setVisible(true);
305         }
306 
307         public void actionPerformed(ActionEvent e)
308         {
309             String es = e.getActionCommand();
310             switch (es)
311             {
312                 case "radioS":
313                     radioC.setSelected(false);
314                     hostnameField.setEnabled(false);
315                     break;
316                 case "radioC":
317                     radioS.setSelected(false);
318                     hostnameField.setEnabled(true);
319                     break;
320                 case "b":    // スタートボタンが押されたとき
321                     Thread t = new Thread(this);
322                     t.start();
323                     waitFrame();
324                     break;
325             }
326         }
327     }
328 }
```

ソースコード 3: NonogramFileNames.java

```
1 
```

```
2  import javax.swing.*;
3  import java.awt.*;
4  import java.awt.event.*;
5  import java.util.ArrayList;
6  import java.util.Random;
7
8  // お絵描きロジックのファイル名を扱うクラス
9  class NonogramFileNames
10 {
11     // 難易度別のファイル名のリストを保存するArrayList
12     private ArrayList<String> easy = new ArrayList<>();
13     private ArrayList<String> normal = new ArrayList<>();
14     private ArrayList<String> hard = new ArrayList<>();
15
16     NonogramFileNames()    // ここにファイル名を設定してください
17     {
18         // easyモードのファイル名
19         easy.add("yama.txt");
20         easy.add("yen.txt");
21         easy.add("king.txt");
22
23         // normalモードのファイル名
24         normal.add("sound.txt");
25
26         // difficultモードのファイル名
27         hard.add("uec.txt");
28     }
29
30     // 指定された難易度のファイル群の中から、ランダムで一つファイル名を返す
31     public String getRandomFileName(Nonogram.difficulty d)
32     {
33         ArrayList<String> fileNames;
34         switch(d)
35         {
36             case easy:
37                 fileNames = this.easy;
38                 break;
39             case normal:
40                 fileNames = this.normal;
41                 break;
42             case hard:
43                 fileNames = this.hard;
44                 break;
45             default:
46                 System.out.println("Unsupported difficulty setting");
47                 fileNames = null;
48                 break;
49         }
50         Random rand = new Random();
51         int num = rand.nextInt(fileNames.size());    // 0からファイル数までの乱数
```

```
52          return fileNames.get(num);    // リストの乱数番目のファイル名を返す
53      }
54  }
```

ソースコード 4: NonogramServer.java

```
1
2   import javax.swing.*;
3   import java.awt.*;
4   import java.awt.event.*;
5   import java.util.*;
6   import java.io.*;
7
8   interface NonogramMessageObserver
9   {
10      public void updateMsg(String msg);
11  }
12
13  // NonogramModelで使う受信用メソッド
14  // 送信用のメソッド"send"をオーバーライドさせるため抽象クラス
15  abstract class NonogramMessageObservable
16  {
17      //
            Observerの記憶にはArrayListを使用（ただし今回のプログラムではObserverは一つだけ）

18      private ArrayList<NonogramMessageObserver> Observer = new ArrayList<>();
19
20      public void add(NonogramMessageObserver ob)
21      {
22          Observer.add(ob);
23      }
24
25      protected void setMsg(String msg)
26      {
27          // addされたObserver実装クラスのupdateMsgを呼び出し
28          for (int i = 0; i < Observer.size(); i++)
29          {
30              Observer.get(i).updateMsg(msg);
31          }
32      }
33
34      abstract public void send(String msg);    // オーバーライドが必要
35  }
36
37  // サーバー、本科目のHP掲載のChatServerクラスを参考に作成
38  class NonogramServer extends NonogramMessageObservable implements Runnable
39  {
40      private CommServer sv;
41      private Thread t;
42
```

```
43      NonogramServer()
44      {
45          this(10010);
46      }
47
48      NonogramServer(int portNum)
49      {
50          // CommServerオブジェクトをポート番号を指定して生成.
51          sv = new CommServer(portNum);
52          t = new Thread(this);
53          t.start();
54      }
55
56      public void run()
57      {
58          String msg;
59          // メッセージ待ちのループに入ります.
60          // sv.recv()で受信. sv.send()で送信.
61          // 相手のプログラムが通信を切断したら，終了.
62          while((msg = sv.recv()) != null)
63          {
64              setMsg(msg);    // 親クラスのメソッドを呼び出し
65          }
66      }
67
68      public void send(String msg)
69      {
70          sv.send(msg);
71      }
72  }
73
74
75  // クライアント、本科目のHP掲載のChatClientクラスを参考に作成
76  class NonogramClient extends NonogramMessageObservable implements Runnable
77  {
78      private CommClient cl;
79      private Thread t;
80
81      NonogramClient()
82      {
83          this("localhost", 10010);
84      }
85
86      NonogramClient(String hostName, int portNum)
87      {
88          cl = new CommClient(hostName, portNum);
89          t = new Thread(this);
90          t.start();
91      }
92
```

```
93      public void run()
94      {
95          String msg;
96          while((msg = cl.recv()) != null)
97          {
98              setMsg(msg);
99          }
100     }
101
102     public void send(String msg)
103     {
104         cl.send(msg);
105     }
106 }
```

ソースコード 5: ButtonPanel.java

```
1  import javax.swing.*;
2  import java.awt.*;
3  import java.awt.event.*;
4
5  class ButtonPanel extends JPanel
6  {
7      public final int mapMaxWidth = 50;    // マス目の最大の幅
8      public final int mapMaxLength = 50;    // マス目の最大の縦の長さ
9
10     public int mapWidth = 5;    // マス目の幅、最初は5とする
11     public int mapLength = 5;    // マス目の縦の長さ、最初は5とする
12
13     public JButton[][] buttons = new JButton[mapMaxLength][mapMaxWidth];
14
15     ButtonPanel()
16     {
17         this(5, 5);
18     }
19
20     ButtonPanel(int widthSize, int lengthSize)
21     {
22         this.mapWidth = widthSize;
23         this.mapLength = lengthSize;
24         this.setLayout(new GridLayout(mapWidth, mapLength));
25         for (int i = 0; i < mapLength; i++)
26         {
27             for (int j = 0; j < mapWidth; j++)
28             {
29                 buttons[i][j] = new JButton();
30                 this.add(buttons[i][j]);
31             }
32         }
33     }
```

```
}
```

ソースコード 6: NonogramController.java

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;

class NonogramController implements ActionListener
{
    private NonogramModel model;

    public NonogramController(NonogramModel m, ButtonPanel b)
    {
        model = m;
        String s1,s2,s3;
        for (int i = 0; i < b.mapLength; i++) {
            for (int j = 0; j < b.mapWidth; j++) {
                b.buttons[i][j].addActionListener(this);
                s1 = Integer.toString(i);   //s1にiを文字列にしたものを入れる。
                s2 = Integer.toString(j);   //s2にjを文字列にしたものを入れる。
                s3 = s2 + ',' + s1;
                b.buttons[i][j].setActionCommand(s3);
            }
        }
    }

    public void actionPerformed(ActionEvent e)
    {
        String es = e.getActionCommand();   //押されたボタンのs3取得
        int index = es.indexOf(",");        //  ','の位置取得
        String xstr = es.substring(0, index);   //最初から','の位置までの文字取得（x座標）
        String ystr = es.substring(index + 1);  //取得した','の位置の次から最後までの文字取得（y座標）
        int xint = Integer.parseInt(xstr);
        int yint = Integer.parseInt(ystr);
        model.setValue(xint, yint);
    }
}
```

ソースコード 7: NonogramFrame.java

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import javax.swing.Timer;
import javax.swing.border.*;
import javax.sound.sampled.Clip;

```

```
8  class NonogramFrame extends JFrame implements ActionListener {
9      ButtonPanel b;
10     NonogramModel m;
11
12     private Timer timer; int seconds=0;
13     private JLabel time = new JLabel("00:00");
14
15     private JLabel close_window_c = new JLabel("ウィンドウを閉じてください");
16     private JLabel close_window_o = new JLabel("ウィンドウを閉じてください");
17
18     private JPanel p5 = new JPanel();
19
20
21     ClassLoader cl = this.getClass().getClassLoader(); // jarファイル作成のため追加
22     //private ImageIcon RedFlash = new ImageIcon("parts/RedFlash.gif"); //
           jarファイル作成のため変更
23     private ImageIcon RedFlash = new ImageIcon(cl.getResource("parts/RedFlash.gif
           "));
24     //private ImageIcon Heart01 = new ImageIcon("parts/Heart01.png"); //
           jarファイル作成のため変更
25     private ImageIcon Heart01 = new ImageIcon(cl.getResource("parts/Heart01.png"));
26     //private ImageIcon Heart02 = new ImageIcon("parts/Heart02.png"); //
           jarファイル作成のため変更
27     private ImageIcon Heart02 = new ImageIcon(cl.getResource("parts/Heart02.png"));
28     private JLabel HeartlifeImage0 = new JLabel(Heart02);
29     private JLabel HeartlifeImage1 = new JLabel(Heart02);
30     private JLabel HeartlifeImage2 = new JLabel(Heart02);
31     private JLabel HeartlifeImage3 = new JLabel(Heart02);
32     private JLabel HeartlifeImage4 = new JLabel(Heart02);
33     private JLabel HeartdeathImage0 = new JLabel(Heart01);
34     private JLabel HeartdeathImage1 = new JLabel(Heart01);
35     private JLabel HeartdeathImage2 = new JLabel(Heart01);
36     private JLabel HeartdeathImage3 = new JLabel(Heart01);
37     private JLabel HeartdeathImage4 = new JLabel(Heart01);
38
39     private JPanel gameover_screen = new JPanel();
40     private JPanel gameclear_screen = new JPanel();
41     private JLabel gameover_text;
42     private JLabel gameclear_text;
43
44     private JLabel[] Width_sub;
45     private JLabel[] Length_sub;
46
47     private JPanel frame = new JPanel();
48
49     private AudioPlay AP, GO;
50     //private ImageIcon SoundOn = new ImageIcon("parts//SoundOn.png"); //
           jarファイル作成のため変更
51     private ImageIcon SoundOn = new ImageIcon(cl.getResource("parts/SoundOn.png"));
52     //private ImageIcon SoundOff = new ImageIcon("parts/SoundOff.png"); //
```

```
        jarファイル作成のため変更
53      private ImageIcon SoundOff = new ImageIcon(cl.getResource("parts/SoundOff.png
            "));
54      private Boolean OnOff = true;
55      private JButton SoundOnOff = new JButton(SoundOn);
56
57      private boolean battleMode = false; //対戦モードのときtrue
58
59      public enum judge
60      {
61          correct,
62          miss,
63          clear,
64          win,
65          lose
66      }
67
68      public NonogramFrame(ButtonPanel b) {
69        this.b = b;
70
71        JPanel p1 = new JPanel(); JPanel p2 = new JPanel();
72        JPanel p3 = new JPanel(); JPanel p4 = new JPanel();
73        JPanel p6 = new JPanel();
74        JPanel p7 = new JPanel(); JPanel p8 = new JPanel();
75
76        JPanel p_all  = new JPanel();
77        JPanel blank  = new JPanel();
78        JPanel Width  = new JPanel();
79        JPanel Length = new JPanel();
80        Width_sub  = new JLabel[b.mapMaxLength];
81        Length_sub = new JLabel[b.mapMaxWidth];
82
83        p1.setBackground(Color.WHITE);p2.setBackground(Color.WHITE);
84        p3.setBackground(Color.WHITE);p4.setBackground(Color.WHITE);
85        p5.setBackground(Color.WHITE);p6.setBackground(Color.WHITE);
86        p7.setBackground(Color.WHITE);p8.setBackground(Color.WHITE);
87        blank.setBackground(Color.WHITE);
88
89        p2.add(time);
90        p5.add(HeartlifeImage0);
91        p5.add(HeartlifeImage1);
92        p5.add(HeartlifeImage2);
93        p5.add(HeartlifeImage3);
94        p5.add(HeartlifeImage4);
95
96        //経過時間の設定
97        time.setFont(new Font("Arial", Font.BOLD, 30));
98        timer = new Timer(1000, this);
99        timer.start();
100
```

```
101         //ゲームオーバー時に出る画面の設定
102         gameover_text = new JLabel("Game Over");
103         gameover_text.setPreferredSize(new Dimension(580, 525));
104         gameover_text.setFont(new Font("Arial", Font.BOLD, 80));
105         gameover_text.setHorizontalAlignment(JLabel.CENTER);
106         gameover_screen.add(gameover_text, BorderLayout.CENTER);
107         gameover_screen.add(close_window_o, BorderLayout.SOUTH);
108         gameover_screen.setBounds(0, 0, 600, 600);
109
110         //ゲームクリア時に出る画面の設定
111         gameclear_text = new JLabel("Game Clear");
112         gameclear_text.setPreferredSize(new Dimension(580, 525));
113         gameclear_text.setFont(new Font("Arial", Font.BOLD, 80));
114         gameclear_text.setHorizontalAlignment(JLabel.CENTER);
115         gameclear_screen.add(gameclear_text, BorderLayout.CENTER);
116         gameclear_screen.add(close_window_c, BorderLayout.SOUTH);
117         gameclear_screen.setBounds(0, 0, 600, 600);
118
119         //BGMのオンオフのボタンの設定
120         SoundOnOff.setPreferredSize(new Dimension(50, 50));
121         SoundOnOff.setBackground(Color.WHITE);
122         SoundOnOff.setBorderPainted(false);
123         SoundOnOff.addActionListener(this);
124         p3.add(SoundOnOff);
125
126         for(int i=0; i<b.mapLength; i++) { //ボタンの背景色を白に設定
127           for(int j = 0; j<b.mapWidth; j++)
128             b.buttons[i][j].setBackground(Color.WHITE);
129         }
130
131         for(int i=0; i<b.mapWidth; i++) { //上のヒントの設定
132           Width_sub[i] = new JLabel();
133           Width_sub[i].setHorizontalAlignment(JLabel.CENTER);
134           Width_sub[i].setVerticalAlignment(JLabel.BOTTOM);
135           Width.add(Width_sub[i]);
136           Width_sub[i].setBorder(new LineBorder(Color.BLACK,1));
137         }
138         for(int i=0; i<b.mapLength; i++) { //右のヒントの設定
139           Length_sub[i] = new JLabel();
140           Length_sub[i].setHorizontalAlignment(JLabel.RIGHT);
141           Length.add(Length_sub[i]);
142           Length_sub[i].setBorder(new LineBorder(Color.BLACK,1));
143         }
144
145         p_all.setLayout(null);
146         blank.setBounds(0, 0, 100, 100);
147         b.setBounds(100, 100, 300, 300);
148         Width.setBounds(100, 0, 300, 100);
149         Length.setBounds(0, 100, 100, 300);
150
```

```
151       frame.setLayout(null);
152       p_all.setBounds(100, 100, 400, 400);
153       p1.setBounds(0, 0, 100, 100);    p2.setBounds(100, 0, 400, 100);
154       p3.setBounds(500, 0, 100, 100);  p4.setBounds(0, 100, 100, 400);
155       p5.setBounds(500, 100, 100, 400);p6.setBounds(0, 500, 100, 100);
156       p7.setBounds(100, 500, 400, 100);p8.setBounds(500, 500, 100, 100);
157
158       Width.setLayout(new GridLayout(1,b.mapWidth));
159       Length.setLayout(new GridLayout(b.mapLength,1));
160
161       p_all.add(blank);
162       p_all.add(Width);
163       p_all.add(Length);
164       p_all.add(b);
165
166       frame.add(p1);frame.add(p2);
167       frame.add(p3);frame.add(p4);
168       frame.add(p_all);
169       frame.add(p5);frame.add(p6);
170       frame.add(p7);frame.add(p8);
171
172       this.add(frame);
173       this.setTitle("NonogramFrame");
174       this.setSize(600,600);
175       this.setResizable(false);
176       this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
177
178       //BGMの再生
179       AP = new AudioPlay("parts/BGM_play.wav");
180       AP.clip.loop(Clip.LOOP_CONTINUOUSLY);
181       AP.clip.stop();
182     }
183
184     @Override
185     public void setVisible(boolean b) {
186       super.setVisible(b);
187       if (b) {AP.clip.loop(Clip.LOOP_CONTINUOUSLY);}
188     }
189
190     public void actionPerformed(ActionEvent e) {
191       if(e.getSource() == timer) { //経過時間のカウント
192         seconds++;
193         if (!battleMode)
194           time.setText(String.format("%02d:%02d", seconds/60, seconds%60));
195         return;
196       }
197
198       if(e.getSource() == SoundOnOff) {
199         if(OnOff==true) { //オフならアイコン替えてBGMを停止
200           SoundOnOff.setIcon(SoundOff);
```

```
201           AP.clip.stop();
202           OnOff = false;
203         } else { //オンならアイコン替えてBGMを再生
204           SoundOnOff.setIcon(SoundOn);
205           AP.clip.loop(Clip.LOOP_CONTINUOUSLY);
206           OnOff = true;
207         }
208         return;
209       }
210     }
211
212     public void setView(int x, int y, judge type, int deathCount) {
213       if(type == judge.correct) { //正解ならそのマスを黒く塗る
214         b.buttons[y][x].setBackground(Color.BLACK);
215         return;
216       } else if (type == judge.miss) { //不正解ならそのマスに^^e2^^98^^93の
              GIFを設定
217         Image image = RedFlash.getImage().getScaledInstance((int)(RedFlash.
                getIconWidth()*0.6), (int)(RedFlash.getIconHeight()*0.6), Image.
                SCALE_DEFAULT);
218         ImageIcon RedFlash1 = new ImageIcon(image);
219         b.buttons[y][x].setIcon(RedFlash1);
220
221         if(deathCount==0) { //残りライフ4
222           p5.remove(HeartlifeImage4); p5.add(HeartdeathImage4);
223         } else if(deathCount==1) { //残りライフ3
224           p5.remove(HeartlifeImage3); p5.add(HeartdeathImage3);
225         } else if(deathCount==2) { //残りライフ2
226           p5.remove(HeartlifeImage2); p5.add(HeartdeathImage2);
227         } else if(deathCount==3) { //残りライフ1
228           p5.remove(HeartlifeImage1); p5.add(HeartdeathImage1);
229         } else if(deathCount==4) { //残りライフ0
230           p5.remove(HeartlifeImage0); p5.add(HeartdeathImage0);
231           gameover(); //ゲームオーバー画面を表示
232         }
233       }
234       else if (type == judge.clear) {gameclear();} //ゲームクリアならゲームクリア画
              面表示
235       else if (type == judge.win)   {gameWin();}   //ネットワーク対戦で勝ったなら
              Win画面表示
236       else if (type == judge.lose)  {gameLose();}  //ネットワーク対戦で負けたなら
              Lose画面表示
237     }
238
239     public void gameover() {
240       System.out.println("Game Over");
241       timer.stop(); //タイマーとBGMを停止
242       AP.clip.close();
243
244       GO = new AudioPlay("parts/BGM_gameover.wav");
```

```
245       GO.clip.start(); //専用BGMの再生
246
247       frame.removeAll();
248       frame.revalidate();
249       frame.repaint();
250       frame.add(gameover_screen);
251     }
252
253     public void gameclear() {
254       System.out.println("Game Clear");
255       timer.stop();
256       AP.clip.close();
257
258       GO = new AudioPlay("parts/BGM_gameclear.wav");
259       GO.clip.start();
260
261       frame.removeAll();
262       frame.revalidate();
263       frame.repaint();
264       frame.add(gameclear_screen);
265     }
266
267     public void gameLose() {
268       System.out.println("Lose");
269       gameover_text.setText("You Lose");
270       timer.stop();
271       AP.clip.close();
272
273       GO = new AudioPlay("parts/BGM_gameover.wav");
274       GO.clip.start();
275
276       frame.removeAll();
277       frame.revalidate();
278       frame.repaint();
279       frame.add(gameover_screen);
280     }
281
282     public void gameWin() {
283       System.out.println("Win");
284       gameclear_text.setText("You Win");
285       timer.stop();
286       AP.clip.close();
287
288       GO = new AudioPlay("parts/BGM_gameclear.wav");
289       GO.clip.start();
290
291       frame.removeAll();
292       frame.revalidate();
293       frame.repaint();
294       frame.add(gameclear_screen);
```

```
295     }
296
297     public void setHintLine(int lineNumber, String hint) {
298       Length_sub[lineNumber].setText(hint); //横書きなのでそのまま
299     }
300
301     public void setHintColumn(int columnNumber, String hint) {
302       String[] str = hint.split(" ");
303       String html = "<html>";
304       for (int i=0; i<str.length; i++)
305           html += str[i]+"<br>";
306       html += "</html>"; //縦書きへ変更
307       Width_sub[columnNumber].setText(html);
308     }
309
310     //ネットワーク対戦時に経過時間のところに相手の残り正解マス数
311     public void setTimeScreen(String str) {
312       this.battleMode = true;
313       time.setText(str);
314     }
315 }
316
317 //BGM：魔王魂
318 //効果音素材：ポケットサウンド ^^e2^^80^^93 https://pocket-se.info/
```

ソースコード 8: StartmenuFrame.java

```
1  import javax.swing.*;
2  import java.awt.*;
3  import java.awt.Desktop;
4  import java.io.IOException;
5  import java.net.URI;
6  import java.net.URISyntaxException;
7  import java.awt.event.*;
8
9  class StartmenuFrame extends JFrame implements ActionListener {
10     JLabel background;
11     JButton easy_button;
12     JButton nomal_button;
13     JButton hard_button;
14     JButton Htp_button;
15     JCheckBox battleChecbox;
16
17     private boolean didStart = false;
18     private Nonogram.difficulty difficulty;
19
20     public StartmenuFrame() {
21
22         this.setSize(600,600);
23         this.setVisible(true);
```

```
24          this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
25          this.setResizable(false);
26          this.setTitle("Nonogram");
27
28          ClassLoader cl = this.getClass().getClassLoader(); //
                jarファイル作成のため追加
29
30          //スタート画面の画像設定
31          //background = new JLabel(new ImageIcon("parts/NonogramStartmenu.png")); //
                jarファイル作成のため変更
32          background = new JLabel(new ImageIcon(cl.getResource("parts/
                NonogramStartmenu.png")));
33          background.setSize(600,600);
34          background.setLayout(null);
35          this.add(background);
36
37          //各難易度のボタン設定
38          easy_button=new JButton("<html><p text-align:center>Easy<br>5×5</p></html
                >");
39          easy_button.setFont(new Font("Arial", Font.BOLD, 25));
40          easy_button.setBorderPainted(false);
41          easy_button.setForeground(Color.WHITE);
42          easy_button.setBackground(new Color(135, 135, 135));
43          easy_button.setBounds(96,348,120,61);
44          easy_button.setBorderPainted(false);
45
46          nomal_button=new JButton("<html><p text-align:center>Nomal<br>10×10</p></
                html>");
47          nomal_button.setFont(new Font("Arial", Font.BOLD, 25));
48          nomal_button.setForeground(Color.WHITE);
49          nomal_button.setBackground(new Color(135, 135, 135));
50          nomal_button.setBounds(233,348,120,61);
51          nomal_button.setBorderPainted(false);
52
53          hard_button=new JButton("<html><p text-align:center>Hard<br>15×15</p></
                html>");
54          hard_button.setFont(new Font("Arial", Font.BOLD, 25));
55          hard_button.setForeground(Color.WHITE);
56          hard_button.setBackground(new Color(135, 135, 135));
57          hard_button.setBounds(368,348,120,61);
58          hard_button.setBorderPainted(false);
59
60          //遊び方説明のHow to playのボタン設定
61          Htp_button=new JButton("How to play");
62          Htp_button.setFont(new Font("Arial", Font.BOLD, 13));
63          Htp_button.setForeground(Color.WHITE);
64          Htp_button.setBackground(new Color(135, 135, 135));
65          Htp_button.setBounds(233,428,120,50);
66          Htp_button.setBorderPainted(false);
67
```

```
68          //ネットワーク対戦のためのチェックボックス設定
69          battleChecbox = new JCheckBox("Battle Mode");
70          battleChecbox.setFont(new Font("Arial", Font.BOLD, 13));
71          battleChecbox.setForeground(Color.GREEN);
72          battleChecbox.setBounds(368, 428, 120, 50);
73          battleChecbox.setBackground(new Color(135, 135, 135));
74
75          easy_button.addActionListener(this);
76          nomal_button.addActionListener(this);
77          hard_button.addActionListener(this);
78          Htp_button.addActionListener(this);
79
80          background.add(easy_button);
81          background.add(nomal_button);
82          background.add(hard_button);
83          background.add(Htp_button);
84          background.add(battleChecbox);
85      }
86      public void actionPerformed(ActionEvent e){
87          if(e.getSource() == Htp_button){
88              Desktop desktop = Desktop.getDesktop();
89              try {
90                  //ルール説明PDFのURLをブラウザで開く
91                  desktop.browse(new URI("https://drive.google.com/file/d/12
                        uqDnwCEAH269FS_dLhUDo42FyTpR6K-/view?usp=sharing"));
92              } catch (IOException f) {f.printStackTrace();} //例外処理
93                catch (URISyntaxException f) {f.printStackTrace();}
94          }
95          else if(e.getSource() == easy_button)  {didStart = true; difficulty =
                difficulty.easy;}
96          else if(e.getSource() == nomal_button) {didStart = true; difficulty =
                difficulty.normal;}
97          else if(e.getSource() == hard_button)  {didStart = true; difficulty =
                difficulty.hard;}
98      }
99
100     //始まったのかどうか
101     public boolean Started() {return didStart;}
102
103     //選択された難易度を取得
104     public Nonogram.difficulty getDifficulty() {return difficulty;}
105
106     //ネットワーク対戦のチェックボックスにチェックが入ったかどうか
107     public boolean isBattleMode() {return battleChecbox.isSelected();}
108 }
```

ソースコード 9: AudioPlay.java

```
1 import javax.sound.sampled.AudioInputStream;
2 import javax.sound.sampled.AudioSystem;
```

```
3  import javax.sound.sampled.Clip;
4  import javax.sound.sampled.LineUnavailableException;
5  import javax.sound.sampled.UnsupportedAudioFileException;
6  import javax.sound.sampled.FloatControl;
7  import java.io.File;
8  import java.io.IOException;
9  import java.net.MalformedURLException;
10
11 public class AudioPlay {
12     Clip clip;
13     AudioInputStream AudioIS;
14     FloatControl ctrl;
15
16     public AudioPlay(String s) {
17         try {
18             ClassLoader cl = this.getClass().getClassLoader(); //
                   jarファイルで動作させるため追加
19             //AudioIS = AudioSystem.getAudioInputStream(new File(s)); //
                   wavファイル取得
20             AudioIS = AudioSystem.getAudioInputStream(cl.getResource(s)); //
                   jarファイルで動作させるために書き換え
21             clip = AudioSystem.getClip();
22             clip.open(AudioIS);
23             FloatControl ctrl = (FloatControl)clip.getControl(FloatControl.Type.
                   MASTER_GAIN);
24             ctrl.setValue((float)Math.log10(0.3) * 20); //音量調整
25         }
26         catch (MalformedURLException e)         {e.printStackTrace();} //例外処理
27         catch (UnsupportedAudioFileException e) {e.printStackTrace();}
28         catch (IOException e)                   {e.printStackTrace();}
29         catch (LineUnavailableException e)      {e.printStackTrace();}
30     }
31 }
```