卒業研究報告書

題目

ぷよぷよにおけるサポート機能の開発

指導教員

石水　隆　講師

報告者

13–1–037–0099

前田 友輝

近畿大学理工学部情報学科

平成 30 年 1 月 25 日提出

概要

ぷよぷよは落ち物ゲームの一つである。フィールドは縦 12 マス×横 6 マスの格子で構成され、上からぷよぷよが 2 つ 1 組で落下し、それを横移動, 回転をさせることでぷよぷよを操作する。色は 3 から 5 色ありプレイヤーは同じ色を 4 つ以上組み合わせる事でぷよを消滅させる事ができる。消滅したぷよの連鎖数によって得点が追加され一番上まで埋まるとゲームオーバーとなるゲームである。ぷよぷよはただ消滅させれば良いだけではなく、いかに連鎖数を繋げることができるかが重要となる。その為競技性の高いゲームあり、積み方一つで局面が大きく変化する事もある。非常に奥が深いゲームではあるが、実際にプレイをしてみると難易度が高く、意図的に 5 連鎖以上を組むことは難しいと言われている。本研究では独自の連鎖判定を行い、初心者を対象としたサポートプログラムの開発を行っている。これにより初心者ユーザーの手助けをしたいと考えている。

目次

1 序論

1.1 本研究の背景

ぷよぷよは落ち物ゲームの一つである。フィールドは縦 12 マス×横 6 マスの格子で構成され、上からぷよぷよが 2 つ 1 組で落下し、それを横移動, 回転をさせることでぷよぷよを操作する。色は 3 から 5 色ありプレイヤーは同じ色を 4 つ以上組み合わせる事でぷよを消滅させる事ができる。消滅したぷよの連鎖数によって得点が追加され一番上まで埋まるとゲームオーバーとなるゲームである。ぷよぷよはただ消滅させれば良いだけではなく、いかに連鎖数を繋げることができるかが重要となる。その為競技性の高いゲームあり、積み方一つで局面が大きく変化する事もある。非常に奥が深いゲームではあるが、実際にプレイをしてみると難易度が高く、意図的に 5 連鎖以上を組むことは難しいと言われている。近年、人類対 AI という構図が注目されており、囲碁なら AlphaGO[7][8]、将棋なら Bonanza[9]、など世界トッププレイヤーに勝利するという歴史的快挙を遂げたこともある。同じくぷよぷよの AI も発展途上ではあるが存在する。mayah(AI) と呼ばれる物で、基本は単純な 2 手読みだがパターンマッチによる補完がされている [6]。mayah(AI) はより人間に近い連鎖と戦術を選択することから最強の AI と呼ばれている。現状では人類にも勝ち目があるが、完全に追い抜かされる日も遠くはない。そこで私は人間をサポートをする機能を開発することにした。

1.2 ぷよぷよに関する既知の結果

本節ではぷよぷよに関する既知の結果について述べる。ぷよぷよは 1991 年にコンパイルから発売されたコンピュータゲームである。人工知能分野の研究対象ともなっており、ぷよぷよを自動でプレイするプログラムの開発も行われている。落ちてくるぷよの順番が与えられたとき、最も連鎖数が大きくなる積み方を求める問題を連鎖判定問題と言う。一般化ぷよぷよの連鎖判定問題に対しては、松金らによりおじゃまぷよがある状態だと NP 完全であることが証明された [3][4][5]。一般化ぷよぷよにおける他の問題として、全消し判定問題がある。これは、入力 P のピース列が終了した時点で盤面上のぷよを全て消す事ができるかを判定する問題である。これも牟田らにより NP 完全であることが示されている [10]。また木場らによりおじゃまぷよがない場合でも 5 色以上であれば NP 完全であることが証明されている [11]。前述のよう連鎖判定問題および全消し問題は NP 完全であるが、ぷよぷよの上級者は連鎖あるいは全消しし易い積み方のパターンに従って積むことで多段連鎖や全消しをすることができる。富沢らは、連鎖し易い積み方を定石としてデータベース化しておく手法を提案し、それが有効であることを示した [12][13]。

1.3 本研究の目的

本研究の目的は、人間のサポートになるサポート機能を開発することである。ぷよぷよは落下する時間があるので短時間で置く位置を判断しなければならない。更にぷよぷよには 4 種類の向きがあるので、全てのパターンを考えると (縦 12 マス×横 6 マスの場合)22 通りの置き方があることが分かる。そのこともあり、初心者が任意に連鎖を組むのは難しいと考えられる。そこで本研究では、初心者の参考になるようにぷよぷよを配置する推奨場所を指示するプログラムを開発する。

1.4 本報告書の構成

本報告書の構成は以下の通りである。まず第 2 章でぷよぷよについて述べる。続く第 3 章で作成したプログラムについて述べ、第 4 章で結果・考察、第 5 章で結論・今後の課題を述べる。

2 ぷよぷよについて

2.1 一般化ぷよぷよの定義

まず、ぷよぷよを以下のルールに従って、1 人で行う組み合わせパズルとして定義する。[3] ルール

1. 盤面は格子状の正方形のマスに区切られ、各マスに右方向を x 座標、上方向を y 座標として (x,y)(y>0) に整数の座標を持つ座標平面である。y = 0 を地面としてサイズの制限は考えない。盤面の一番左下を座標 (1,1) とする。
2. 盤面には「ぷよ」と呼ばれるブロックを盤面のマスに置くことができる。ぷよには「色ぷよ」と「おじゃまぷよ」の 2 種類存在するが、今回は「おじゃまぷよ」は考慮しないこととする。
3. 色ぷよ (以下、単に「ぷよ」と呼ぶ) には 5 種類の色があり、同色のぷよが縦、横に 4 個以上連結すると消える。色は赤、黄、緑、青、桃色が存在する。
4. ピースは 2 個のぷよが繋がったものであり、入力としてピース列が与えられる。
5. ピースは 1 つずつ順に盤面の十分高いところから現れ、プレイヤーはそれを順に操作する。
   - ピースに対して回転 (0 度,90 度,180 度,270 度) を行う。操作方法に関しては下記の表 1 に記す。
   - ピースに対して x 軸方向の移動を行う。
6. 下に空白ができたぷよは、他のぷよか地面に接触するまで落下する。

このようなぷよぷよを、一般化ぷよぷよと呼ぶ。また連鎖について述べる。ルール 7 の性質を利用して、他のぷよが消えて下に空白ができたときに、また同色のぷよが 4 個以上隣接し消えることがある。この現象を連鎖と呼ぶ。[3]

表 1　キー入力

	↑ 180° 回転	
← 左へ移動		→ 右へ移動
	↓ 90° 回転	

2.2 研究内容

本研究では Java を用いてぷよぷよのサポート機能を開発する。本研究で開発するぷよぷよのサポート機能は、ある程度ぷよぷよが積まれた段階の方が連鎖しやすいと判断した為、ぷよぷよの総数が 24 個を超えた段

階で表示するようにした。現在のぷよぷよの位置を p[p_y][p_x] で表現すると、(p[p_y][p_x],p[p_y][p_x]) が初期状態となる。全探索を行う際に、以下の 4 つの回転パターンを試す。

- パターン 1　(p[p_y][p_x],p[p_y][p_x+1])
- パターン 2　(p[p_y][p_x+1],p[p_y][p_x])
- パターン 3　(p[p_y][p_x],p[p_y+1][p_x])
- パターン 4　(p[p_y+1][p_x],p[p_y][p_x])

表 2 に各パターンのぷよぷよの形を示す。

表 2　各ぷよぷよの形

パターン 1	パターン 2	パターン 3	パターン 4
■□	□■	■ □	□ ■

上記のパターンを置ける全ての箇所に落下させることで、最大連鎖する箇所を特定することができる。連鎖をするごとに評価を与え、一連鎖すれば +1、二連鎖すれば +2、連鎖がなければ 0 を入力する。そして評価が一番高い箇所を推奨場所とし、向きと箇所を表示させる。評価値が同じであればその中から、最も多くぷよぷよを削除できる箇所を選択するようにする。

3　ぷよぷよプログラム

本章では、本研究で作成したぷよぷよプログラムについて説明する。付録に本研究で作成したぷよぷよプログラムのソースを示す。プログラムの主要なフィールド変数および各メソッドについて、以下の節で述べる。

3.1　主要なフィールド変数

本節では主要なフィールド変数について、その役割を説明する。以下に、本研究で作成した Game2 クラスのフィールド変数を挙げる。

- int p_x = 0：ぷよの x 座標
- int p_y = 0：ぷよの y 座標
- int row = 0：行
- int col = 0：列
- int p[][] = new int[row][col]：ぷよの位置
- int height[] = new int[6]：ぷよの高さ
- int color1：ぷよの左の色
- int color2：ぷよの右の色
- int sum：ぷよの総数
- int speed：落下スピード
- int evaluationYoko1[] = new int[6]：パターン 1 の評価

- int evaluationYoko2[] = new int[6]：パターン 2 の評価
- int evaluationTate1[] = new int[7]：パターン 3 の評価
- int evaluationTate2[] = new int[7]：パターン 4 の評価
- int evaluationYoko1Max：パターン 1 の最大連鎖数
- int evaluationYoko2Max：パターン 2 の最大連鎖数
- int evaluationTate1Max：パターン 3 の最大連鎖数
- int evaluationTate2Max：パターン 4 の最大連鎖数
- int evaluationYoko1MaxPosition：パターン 1 の最大連鎖数の位置
- int evaluationYoko2MaxPosition：パターン 2 の最大連鎖数の位置
- int evaluationTate1MaxPosition：パターン 3 の最大連鎖数の位置
- int evaluationTate1MaxPosition：パターン 4 の最大連鎖数の位置
- st = 0：パターンの状態、サポート時は 1、2、3、4 となる

3.2 init メソッド

init メソッドはまず背景とゲームパネルの配置とぷよぷよの生成を行うメソッドである。init メソッドは、ルール 5 に基づき新たなピースが出現するときに、出現させるピースを選択し出現位置をランダムに決める。

3.3 run メソッド

run メソッドはピースの落下から次のピースを生成するまでのメソッドである。

run メソッドはプレイヤーまたはぷよぷよ AI がピースを操作する通常操作モードと、お勧め位置を表示するためのサポートモードの 2 種類の動作を行う。各モードでは、init メソッドで出現したピースに対して以下の操作を行う。

[通常操作モード]

1. ピースをプレイヤーの操作に従いぷよまたは床にぶつかるまで落下させる。
2. 以下の操作を同じ色のぷよが 4 つ以上繋がっている限り繰り返す。
   (a) search メソッドで連続して繋がっている同じ色のぷよを探す。
   (b) 4 つ以上繋がっているぷよを消去する。
   (c) 連鎖数をカウントする。
   (d) 消去したぷよの上にあるぷよを落下させる。
3. 連鎖数に応じた得点を加える。
4. 次に出現させるピースを選択し、出現させる。

[サポートモード]

1. copy メソッドで現在の盤面にあるぷよを記録する。
2. 出現したピースに対し、以下の操作を全ての位置および回転に対して繰り返す。
   (a) ピースを指定した位置および回転でぷよまたは床にぶつかるまで落下させる。
   (b) 通常操作モードの 2. と同様にぷよを消去し、連鎖数をカウントする。

(c) undo メソッドで盤面を記録した状態に戻す。

3. 最も連鎖数が多くなるピースの位置、回転の仕方を表示する。
4. サポートモードで用いたピースと同一のピースを出現させ、通常モードに移行する。

3.4 copy メソッド

copy メソッドは現在の盤面を退避領域にコピーするメソッドである。

3.5 undo メソッド

undo メソッドは盤面を copy メソッドでコピーした盤面に戻すメソッドである。

3.6 sequenceMaxValue メソッド

sequenceMaxValue メソッドは各パターンの配列の評価の最大値を求めるメソッドである。評価の Max の値と位置を探索する。

3.7 heightAndSum メソッド

heightAndSum メソッドは盤面上に存在するぷよぷよの総数を求めるメソッドである。

3.8 position メソッド

position メソッドはぷよぷよの初期位置を決めるメソッドである。通常操作モード時はランダムで落下させ、サポートモード時は一番左端から右端まで順に落下させていく。

3.9 select メソッド

select メソッドはぷよぷよの色を選択するメソッドである。select メソッドは通常時とサポート時では処理の仕方が異なる。通常時ではぷよぷよの色、初期位置をランダム生成する。サポート開始前のぷよぷよの色をコピーし、サポート時にその色と同じ色のぷよぷよを落下させる。

3.10 search メソッド

search メソッドは同じ色のぷよぷよを探すメソッドである。search メソッドは、あるぷよぷよの周りに同じ色のぷよぷよがないかを探索しあれば繋がっている同色のぷよぷよの数カウントしていく。

3.11 delete メソッド

delete メソッドは同じ色のぷよぷよを削除するメソッドである。削除後で下が空白であれば下へ詰めていく。

3.12　実行結果

以下の図 1、2、3 に実行結果の例を示す。図 1 は現状では最も連鎖数が高い 2 連鎖の箇所を出力した。出力結果には縦そのままと書かれているので、初期向きから 90 度回転させ 4 列目に落下させると 2 連鎖させることができる。図 2 は連鎖数が同じ場合の優先順位として、最もぷよぷよを削除できる箇所を選択した図である。この場合は桃色のぷよぷよが削除され、黄色のぷよぷよが連鎖し 2 連鎖となる。図 3 は現状では連鎖をすることができない場合の処理である。連鎖がない場合は連鎖はありませんと出力される。

図 1　実行結果 1

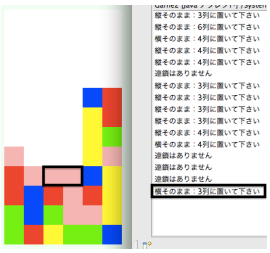

図 2　実行結果 2

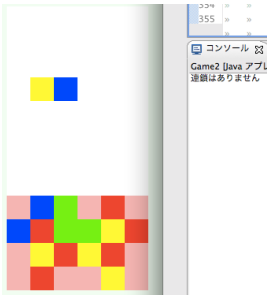

図 3　実行結果 3

4　実験結果・考察

本研究ではサポート機能の性能を検証する為に、サポート機能がある場合とない場合の連鎖数の平均を求め確認した。表 3 に 4 人のプレイヤーに 10 回プレイをしてもらった結果を示す。表 3 の 4 人のプレイヤーは A、B、C、D とし、それぞれ完全な初心者、経験はある初心者、階段積みができる中級者、5 連差以上を組む事ができる上級者とする。性能検証はぷよが 24 個以上積まれている状態で起きた連鎖を記録する。サポート機能の性能を検証するので、サポートの指示 (任意では動かさない) で動くことにした。更に 2 つ目標を設定し実験を行った。目標 1 はサポート機能を使用し 10 回の平均連鎖数が 3 連鎖を超えること、目標 2 はサポートがプレイヤーの連鎖数を上回ることとする。表 3 の結果では、4 人のサポート機能がある場合の平均は 1.8 となり目標である平均連鎖数が 3 連差を超えることができなかった。また 4 人のサポート機能がある場合とない場合の比較も、A と B には効果的であったが、C と D には逆効果となってしまった。このことから本サポート機能プログラムの性能は、初心者には効果的であるが、中級者以上にはそれほど効果がないことが分かった。ぷよぷよ上級者は 5、6 連差を組む事ができ、場合によっては 10 連差を超えることもあることと比較すると、本サポート機能プログラムの性能は高いとは言えない。本サポート機能プログラムは、現状だと一手読みしかできていないので、高い連鎖数になりそうな場所をうまく判定しづらいことが原因だと考えられる。

5　結論・今後の課題

多くの連鎖に導けるサポート機能にする為にはまだ改善するところが多々ある。例えばぷよぷよの積み方自体に評価値を与えたり、予め連鎖が発生し易いぷよぷよの積み方をパターンとして保持する方法を加える事で、より高性能なサポート機能が得られると期待される。他にはどのタイミングで連鎖をすればよいかなども

表 3　実験結果 (試行回数各 10 回)

被験者	サポート無し		サポート有り	
	平均連鎖数	最大連鎖数	平均連鎖数	最大連鎖数
A	1.2	2	2.0	3
B	1.4	2	1.6	3
C	2.4	3	2.0	3
D	3.1	5	1.7	3

システムに加えたり、ネクストぷよの情報を使い二手先まで読む方法や、ぷよを消し易い積み方を定石として持っておく方法で更に連鎖数を多くすることもできると考えている [12][13]。また、今回は一人用のとことんぷよぷよを想定して開発したが、対戦用のぷよぷよのサポート開発も今後の課題である。

謝辞

本研究を行うにあたって、近畿大学理工学部情報学科情報論理工学研究室の石水 隆講師には大変お世話になりました。研究に関する指南やサポート、アドバイスなど適切な指導をしていただいたことを、ここに感謝をこめて記します。

参考文献

[1] 楽しく学べる Java ゲーム・アプレット, 工学社 (2002).

[2] Java ゲームプログラミング,SB クリエイティブ (2007).

[3] 松金輝久, 武永康彦：一般化ぷよぷよの NP 完全性, 数理解析研究所講究録 147-152 (2005).
http://www.kurims.kyoto-u.ac.jp/~kyodo/kokyuroku/contents/pdf/1426-24.pdf

[4] 松金 輝久, 武永 康彦：一般化ぷよぷよの連鎖数判定問題, 電子信学会技術研究報告, COMP, コンピュテーション Vol.104, No.743, pp.95–103, 電子情報通信学会 (2005).

[5] 松金 輝久, 武永 康彦：組合せ最適化問題としてのぷよぷよの連鎖数判定問題, 電子情報通信学会論文誌. D, 情報・システム, Vol.J89-D No.3, pp.405–413, 電子情報通信学会 (2006).

[6] mayahjp：ぷよぷよ AI mayah(AI) の実装 (2015).
https://www.slideshare.net/mayahjp/ai-mayah

[7] Kazuto Seki: 囲碁の最強人口知能 AlphaGo(アルファ碁) の仕組みとは?, TECH::NOTE, 2017 年 9 月 17 日, 株式会社 div, (2017).
https://tech-camp.in/note/technology/32855/

[8] 伊藤毅志, 村松正和: ディープラーニングを用いたコンピュータ囲碁〜 Alpha Go の技術と展望〜, 情報処理, Vol.57, No.4, pp.335–337, 情報処理学会, (2016).
http://id.nii.ac.jp/1001/00158059/

[9] 保木邦仁：ゲーム木探索の最適制御：将棋における局面評価の機械学習, 東北大学大学院理学研究科.
https://www.ipsj.or.jp/10jigyo/forum/software-j2008/hoki-print.pdf

[10] 牟田秀俊：ぷよぷよは NP 完全, 電子信学会技術研究報告,COMP, コンピュテーション Vol.105,No.72,pp.39-44, 電子情報通信学会 (2005).

[11] 木場裕矢, 宗重成央, 上嶋章宏: 色数とおじゃまぷよを制限した一般化ぷよぷよの連鎖数判定問題の NP 完全性, 秋季研究発表会アブストラクト集 2011, pp.370–371, 日本オペレーションズ・リサーチ学会, (2011).

[12] 富沢大介, 池田心, 橋本隼一: 関連性行列を用いたぷよぷよの定型連鎖構成法, ゲームプログラミングワークショップ 2011 論文集, Vol.2011, No,6, pp. 9–16, 情報処理学会, (2011).
http://id.nii.ac.jp/1001/00078248/

[13] 富沢 大介, 池田 心, Simon Viennot：落下型パズルゲームの定石形配置法とぷよぷよへの適用, 情報処理学会論文誌, Vol.53, No.11, pp.2560–2570, (2012).
http://id.nii.ac.jp/1001/00087056/

付録 A　ソースプログラム

本研究で作成したぷよぷよのプログラムのソースプログラムを以下に示す。

```
package puyopuyo;

/*********************************/
/* ぷよぷよ（ゲーム全体の制御） */
/*********************************/
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
import java.util.*;

import puyopuyo.Puyo;

public class Game2 extends Applet implements Runnable {
        int p_x = 0; // ぷよの x 座標
        int p_y = 0; // ぷよの y 座標
        int rot = 0; // 回転,0 なら横,1 なら縦
        int row = 12; // 行
        int col = 6; // 列
        boolean state = true; // スレッドの状態
        boolean game = true;
        boolean ok = true;
        Puyo py;
        int p[][] = new int[row][col]; // ぷよの位置
        int height[] = new int[6]; // ぷよの高さ
        int zentai[][] = new int[row][col]; // 盤面全体を格納する二次元変数
        Random rn; // 乱数
        Thread th;
        int color1; // ぷよの左の色
        int color2; // ぷよの右の色
        int sum; // ぷよの総数
        int speed = 500; // 落下速度
        int evaluationYoko1[] = new int[6]; // パターン 1 の評価
        int evaluationYoko2[] = new int[6]; // パターン 2 の評価
        int evaluationTate1[] = new int[7]; // パターン 3 の評価
        int evaluationTate2[] = new int[7]; // パターン 4 の評価
        int evaluationYoko1Max; // パターン 1 の最大連鎖数
        int evaluationYoko2Max; // パターン 2 の最大連鎖数
        int evaluationTate1Max; // パターン 3 の最大連鎖数
        int evaluationTate2Max; // パターン 4 の最大連鎖数
        int evaluationYoko1MaxPosition = 0; // パターン 1 の最大連鎖数の位置
        int evaluationYoko2MaxPosition = 0; // パターン 2 の最大連鎖数の位置
        int evaluationTate1MaxPosition = 0; // パターン 3 の最大連鎖数の位置
        int evaluationTate2MaxPosition = 0; // パターン 4 の最大連鎖数の位置
        int st = 0; // パターンの状態, 通常時 0,5,6 サポート時 1,2,3,4
        String evst; // パターンの識別
        int a = 0;
```

```
/************/
/* 初期設定 */
/************/
public void init() {
        // 背景色
        setBackground(new Color(238, 255, 238));
        // ゲームパネルの配置
        setLayout(null);
        py = new Puyo(this); // ピースの作成
        add(py);
        py.setSize(30 * col, 30 * row);
        py.setLocation(10, 10);
        py.setBackground(Color.white);

        rn = new Random();
        // ピースの選択
    select();
        // スレッドの生成
        th = new Thread(this);
        th.start();
}

/******************/
/* 左右上下の余裕 */
/******************/
public Insets getInsets() {
        return new Insets(10, 10, 10, 10);
}

/******************/
/* スレッドの停止 */
/******************/
public void stop() {
        state = false;
}

/******************/
/* スレッドの実行 */
/******************/
public void run() {
        int i1, i2, i3, i4, ct;
        int pp[][] = new int[row][col];
        while (state) {
                try {
                        th.sleep(speed); // 落下速度
                } catch (InterruptedException e) {
                }

                // ピースの落下

        ct = 0;
        if (rot == 0) {
                // 一番下まで落下した時
```

```
                if (p_y < row - 1) {
                        if (ok) {
                /*
                 * 下にぷよがない場合 p_x は左または上のピースの座標（横）
                 * p_y は左または上のピースの座標（縦）
        */
                if (p[p_y + 1][p_x] == 0 &&
                                p[p_y + 1][p_x + 1] == 0) {
                                        ct = 1;
                                        /*
                                         * 現在のぷよを一つ下に移す
                                         * +1 は一つ下を表す
                                         */
                                p[p_y + 1][p_x] = p[p_y][p_x];
                                p[p_y + 1][p_x + 1] = p[p_y][p_x + 1];
                                /* 元々あった場所を０にする */
                                p[p_y][p_x] = 0;
                                p[p_y][p_x + 1] = 0;
                                p_y++;
                        }
                                        // 下にぷよがある場合
                    else {
                            // 左のぷよの下にある場合
                        if (p[p_y + 1][p_x] == 0) {
                                ct = 1;
                                p[p_y + 1][p_x] = p[p_y][p_x];
                                p[p_y][p_x] = 0;
                                p_y++;
                            }
                            // 右のぷよの下にある場合
                        else if (p[p_y + 1][p_x + 1] == 0) {
                                ct = 1;
                                p[p_y + 1][p_x + 1] = p[p_y][p_x + 1];
                                p[p_y][p_x + 1] = 0;
                                p_y++;
                                        }
                                }
                            } else {
                                if (p[p_y + 1][p_x] == 0) {
                                        ct = 1;
                                        p[p_y + 1][p_x] = p[p_y][p_x];
                                        p[p_y][p_x] = 0;
                                        p_y++;
                                }
                            }
                }
        }
                // 回転する場合
else {
                if (p_y < row - 2 && p[p_y + 2][p_x] == 0) {
                        // a が 6 の場合ぷよを一つ右に移動させる
                        if (a == 6 && p_x < col - 1
                                        && p[p_y][p_x + 1] == 0
```

```
                                        && p[p_y + 1][p_x + 1] == 0) {
                                p[p_y][p_x + 1] = p[p_y][p_x];
                                p[p_y + 1][p_x + 1] = p[p_y + 1][p_x];
                                p[p_y][p_x] = 0;
                                p[p_y + 1][p_x] = 0;
                                p_x++;
                                repaint();
                        }
                                ct = 1;
                                p[p_y + 2][p_x] = p[p_y + 1][p_x];
                                p[p_y + 1][p_x] = p[p_y][p_x];
                                p[p_y][p_x] = 0;
                                p_y++;
                        } //else
                                //ok = false;
                }

                // 削除と次のピース
        if (ct == 0) {
                ct = 4;
                // 4 つ同じ色があれば削除する
                while (ct >= 4) {
                        ct = 0;
                        for (i1 = row - 1; i1 >= 0 && ct < 4; i1--) {
                                for (i2 = 0; i2 < col && ct < 4; i2++) {
                                        for (i3 = 0; i3 < row; i3++) {
                                                for (i4 = 0; i4 < col; i4++)
                                                        pp[i3][i4] = 0;
                                        }
                                        if (p[i1][i2] > 0) {
                                                pp[i1][i2] = 1;
                                                ct = search(pp, i1, i2, 1);
                                        }
                                }
                        }
                        if (ct >= 4) {
                                delete(p, pp);
                                // 連鎖カウントを加算していく
                                if (st == 1) { // パターン 1 の場合
                                 evaluationYoko1[p_x] += 1;
                                }
                                if (st == 2) { // パターン 2 の場合
                                 evaluationYoko2[p_x] += 1;
                                }
                                if (st == 3) { // パターン 3 の場合
                                 evaluationTate1[p_x] += 1;
                                }
                                if (st == 4) { // パターン 4 の場合
                                 evaluationTate2[p_x] += 1;
                                }
                        }
                }
                        // 連鎖なしは加算しない
```

```
  if (st == 1) { // パターン 1 の場合
    evaluationYoko1[p_x] += 0;
   }
   if (st == 2) { // パターン 2 の場合
    evaluationYoko2[p_x] += 0;
   }
   if (st == 3) { // パターン 3 の場合
    evaluationTate1[p_x] += 0;
   }
   if (st == 4) { // パターン 4 の場合
    evaluationTate2[p_x] += 0;
   }

   // ぷよの総数が 24 個以上の場合、
   //サポート機能の処理をする
   if (st == 0 && sum >= 24 ||
                   st == 6 && sum >= 24) {
           st = 1;
           // 落下速度
           speed = 10;
   }
   // パターン 1 が右端に到達した場合
if (a == 5 && st == 1) {
           a = 0;
           // パターンを 2 へ
           st = 2;
           evst = "evaluationYoko1Max";
           // パターン 1 の最大値を求める
           sequenceMaxValue(evaluationYoko1);
   }
   // パターン 2 が右端に到達した場合
if (a == 5 && st == 2) {
           a = 0;
           // パターンを 3 へ
           st = 3;
           evst = "evaluationYoko2Max";
           // パターン 2 の最大値を求める
           sequenceMaxValue(evaluationYoko2);
   }
   // パターン 3 が右端に到達した場合
if (a == 6 && st == 3) {
           a = 0;
           // パターンを 4 へ
           st = 4;
           evst = "evaluationTate1Max";
           // パターン 3 の最大値を求める
           sequenceMaxValue(evaluationTate1);
   }
   // パターン 4 が右端に到達した場合
if (a == 6 && st == 4) {
           a = 0;
           st = 5;
           evst = "evaluationTate2Max";
```

```
                    // パターン 4 の最大値を求める
                    sequenceMaxValue(evaluationTate2);

                    /* パターンそれぞれの最大値から
                     * 一番高い評価値を求める
                 */
                    int c[] = new int[4];
                    c[0] = evaluationYoko1Max;
                    c[1] = evaluationYoko2Max;
                    c[2] = evaluationTate1Max;
                    c[3] = evaluationTate2Max;
                    int max = c[0]; // 最大値の初期値
                    int num = 4; // 最大値の位置

                    // 最大値を求める処理
                    for (int i = 0; i < 3; i++) {
                            if (max <= c[i]) {
                                    max = c[i];
                                    num = i;
                            }
                    }
                    // 連鎖がない場合
               if (c[0] == 0 && c[1] == 0 &&
                                    c[2] == 0 && c[3] == 0) {
                            num = 4;
                    }

// パターン 1 の場合
if (num == 0) {
        int b = evaluationYoko1MaxPosition + 1;
        // 横に 6 列目はないので 5 列目とする
        if (b == 6) {
                System.out.println
                ("横そのまま：5 列に置いて下さい");
        } else {
                System.out.println
                ("横そのまま：" + b + "列に置いて下さい");
        }
        // 位置を初期化する
        evaluationYoko1MaxPosition = 0;
}
// パターン 2 の場合
else if (num == 1) {
        int b = evaluationYoko2MaxPosition + 1;
        // 横に 6 列目はないので 5 列目とする
        if (b == 6) {
                System.out.println
                ("横逆向きに：5 列に置いて下さい");
        } else {
                System.out.println
                ("横逆向きに：" + b + "列に置いて下さい");
        }
        // 位置を初期化する
```

```
                evaluationYoko2MaxPosition = 0;
        }
        // パターン 3 の場合
        else if (num == 2) {
                int b = evaluationTate1MaxPosition + 1;
                // 縦に 7 列目はないので 6 列目とする
                if (b == 7) {
                        System.out.println
                        ("縦そのまま：6 列に置いて下さい");
                } else {
                        System.out.println
                        ("縦そのまま：" + b + "列に置いて下さい");
                }
                // 位置を初期化する
                evaluationTate1MaxPosition = 0;
        }
        //パターン 4 の場合
else if (num == 3) {
                int b = evaluationTate2MaxPosition + 1;
                // 縦に 7 列目はないので 6 列目とする
                if (b == 7) {
                        System.out.println
                        ("縦逆向きに：6 列に置いて下さい");
                } else {
                        System.out.println
                        ("縦逆向きに：" + b + "列に置いて下さい");
                }
                // 位置を初期化する
                evaluationTate2MaxPosition = 0;
        }
        // 連鎖がない場合
  else if (num == 4) {
                System.out.println("連鎖はありません");
        }
                        // 配列の初期化
                    sequenceClear(evaluationYoko1);
                        sequenceClear(evaluationYoko2);
                        sequenceClear(evaluationTate1);
                        sequenceClear(evaluationTate2);
                        // 最大値の初期化
                  evaluationYoko1Max = 0;
                        evaluationYoko2Max = 0;
                        evaluationTate1Max = 0;
                        evaluationTate2Max = 0;
                        // 落下速度
                        speed = 500;
                }

                // サポート後の処理
          if (a == 1 && st == 5) {
                        a = 0;
                        // 通常処理へ
                        st = 6;
```

```
                        }
                        // サポート時は undo 処理をする
                        if (st == 1 || st == 2 ||
                                st == 3 || st == 4 || st == 5) {
                                // copy した状態に戻す
                              undo();
                        }
                        // ぷよの高さと総数を求める
                 heightAndSum();
                        // コピー
                        copy();
                        // 次のピースを選択する
                    select();

                }

                // 再描画
                py.repaint();

        }
}

/*****************/
/* 中身をコピーする */
/*****************/
void copy() {
        for (int i = 0; i < 12; i++) {
                for (int j = 0; j < 5; j++) {
                        zentai[i][j] = p[i][j];
                        zentai[i][j + 1] = p[i][j + 1];
                }
        }
}

/*******************/
/* 中身を前の状態に戻す */
/*******************/
void undo() {
        for (int i = 0; i < 12; i++) {
                for (int j = 0; j < 5; j++) {
                        p[i][j] = zentai[i][j];
                        p[i][j + 1] = zentai[i][j + 1];
                }
        }
}

/*************************/
/* 各配列の評価の最大値を求める */
/*************************/
void sequenceMaxValue(int[] a) {
        // 初期値を入れる
        if (evst == "evaluationYoko1Max") {
                evaluationYoko1Max = a[0];
```

```
        } else if (evst == "evaluationYoko2Max") {
                evaluationYoko2Max = a[0];
        } else if (evst == "evaluationTate1Max") {
                evaluationTate1Max = a[0];
        } else if (evst == "evaluationTate2Max") {
                evaluationTate2Max = a[0];
        }
        // パターン 1 の場合
        if (evst == "evaluationYoko1Max") {
                for (int i = 0; i < a.length; i++) {
                        if (evaluationYoko1Max <= a[i]) {
                                evaluationYoko1Max = a[i];
                                evaluationYoko1MaxPosition = i;
                        }
                }
        }
        // パターン 2 の場合
        else if (evst == "evaluationYoko2Max") {
                for (int i = 0; i < a.length; i++) {
                        if (evaluationYoko2Max <= a[i]) {
                                evaluationYoko2Max = a[i];
                                evaluationYoko2MaxPosition = i;
                        }
                }
        }
        // パターン 3 の場合
        else if (evst == "evaluationTate1Max") {
                for (int i = 0; i < a.length; i++) {
                        if (evaluationTate1Max <= a[i]) {
                                evaluationTate1Max = a[i];
                                evaluationTate1MaxPosition = i;
                        }
                }
        }
        // パターン 4 の場合
        else if (evst == "evaluationTate2Max") {
                for (int i = 0; i < a.length; i++) {
                        if (evaluationTate2Max <= a[i]) {
                                evaluationTate2Max = a[i];
                                evaluationTate2MaxPosition = i;
                        }
                }
        }
}

/*****************/
/* 配列を初期化する */
/*****************/
void sequenceClear(int x[]) {
        for (int i = 0; i < x.length; i++) {
                x[i] = 0;
        }
}
```

```
/************************/
/* ピースの高さと総数を求める */
/************************/
void heightAndSum() {
        // 高さを求める
    for (int x = 0; x < 6; x++) {
                for (int y = 0; y < 12; y++) {
                        if (p[y][x] != 0) {
                                height[x] = 12 - y;
                                break;
                        }
                }
        }
        // 置かれているぷよの総数
        sum = 0;
        for (int y = 0; y < 6; y++) {
                sum += height[y];
        }
}

/******************/
/* ピースの位置を選択 */
/******************/
void position() {

        // サポート時の落下
 if (st == 1 || st == 2 ||
                st == 3 || st == 4 || st == 5) {
                p_x = a;
                // 一列右へ移動させる
                a = a + 1;
        }

        // 通常時の落下の仕方、ランダムで決める
        if (st == 0 || st == 5 || st == 6) {
                p_x = (int) (rn.nextDouble() * (col - 1));
        }

        // パターン 3、パターン 4 は回転させた状態で落下させる
        if (st == 3 | st == 4) {
                rot = 1;
        }
}

/****************/
/* ピースの選択 */
/****************/
void select() {
        int beforeColor1 = 0; // 前に落としたピースの左の色
        int beforeColor2 = 0; // 前に落としたピースの右の色
        rot = 0; // 回転
        p_y = 0;
```

```
 position();

 // ピース作成時、右端を突き抜けないようにする為の処理
if (p_x > col - 2) {
        p_x = col - 2;
 }

 if (p[0][p_x] == 0 && p[0][p_x + 1] == 0) {
        // 通常時
        if (st == 0 || st == 6) {
                // ランダム生成
                color1 = (int) (rn.nextDouble() * 5) + 1;
        }
        beforeColor1 = color1;
        // サポート時とサポート終了直後の処理
     if (st == 1 || st == 2||
                st == 3 || st == 4 || st == 5) {
                color1 = beforeColor1;
        }
        if (color1 > 5)
                color1 = 5;
        /* 左の色 */
        if (st == 0 || st == 1 ||
                st == 3 || st == 5 || st == 6) {
                p[0][p_x] = color1;
        }
        if (st == 2) {
                p[0][p_x + 1] = color1;
        }
        if (st == 4) {
                p[1][p_x] = color1;
        }

        if (st == 0 || st == 6) {
                color2 = (int) (rn.nextDouble() * 5) + 1;
        }
        beforeColor2 = color2;
        if (st == 1 || st == 2 ||
                st == 3 || st == 4 || st == 5) {
                color2 = beforeColor2;
        }
        if (color2 > 5)
                color2 = 5;
        /* 右の色 */
        if (st == 0 || st == 1 ||
                st == 5 || st == 6) {
                p[0][p_x + 1] = color2;
        }
        if (st == 2 || st == 4) {
                p[0][p_x] = color2;
        }
        if (st == 3) {
```

```
                        p[1][p_x] = color2;
                }

        }

        else
                game = false;
}

/************************************/
/* 同じ色のピースを探す */
/* pp : 同じ色のピース位置 */
/* k1,k2 : 対象としているピース */
/* c1 : 同じ色のピースの数 */
/* return : 同じ色のピースの数 */
/************************************/
int search(int pp[][], int k1, int k2, int c1) {
        int ct = c1;

        // 同じ色のぷよを探しだし ct にカウントしていく
        if (k1 > 0 && p[k1 - 1][k2] ==
                        p[k1][k2] && pp[k1 - 1][k2] == 0) {
                pp[k1 - 1][k2] = 1;
                // 再帰呼び出し
          ct = search(pp, k1 - 1, k2, ct + 1);
        }
        if (k1 < row - 1 && p[k1 + 1][k2] ==
                        p[k1][k2] && pp[k1 + 1][k2] == 0) {
                pp[k1 + 1][k2] = 1;
                ct = search(pp, k1 + 1, k2, ct + 1);
        }
        if (k2 > 0 && p[k1][k2 - 1] ==
                        p[k1][k2] && pp[k1][k2 - 1] == 0) {
                pp[k1][k2 - 1] = 1;
                ct = search(pp, k1, k2 - 1, ct + 1);
        }
        if (k2 < col - 1 && p[k1][k2 + 1] ==
                        p[k1][k2] && pp[k1][k2 + 1] == 0) {
                pp[k1][k2 + 1] = 1;
                ct = search(pp, k1, k2 + 1, ct + 1);
        }

        return ct;
}

/********************************/
/* 同じ色のピースを削除 */
/* p : ピース位置 */
/* pp : 同じ色のピース位置 */
/********************************/
void delete(int p[][], int pp[][]) {
        int i1, i2, i3, k1, k2, k3;
        // 削除
```

```
                for (i1 = 0; i1 < row; i1++) {
                        for (i2 = 0; i2 < col; i2++) {
                                if (pp[i1][i2] > 0)
                                        p[i1][i2] = 0;
                        }
                }
                // 詰める
                for (i1 = 0; i1 < col; i1++) {
                        k1 = 1;
                        for (i2 = row - 1; i2 >0 &&
                                        k1 >= 0; i2--) {
                                if (p[i2][i1] == 0) {
                                        k1 = -1;
                                        for (i3 = i2 - 1; i3 >= 0 &&
                                                        k1 < 0; i3--) {
                                                if (p[i3][i1] > 0)
                                                        k1 = i3;
                                        }
                                        if (k1 >= 0) {
                                                k2 = i2;
                                                k3 = k2 - k1;
                                                while (k1 >= 0) {
                                                        p[k2][i1] = p[k1][i1];
                                                        k1--;
                                                        k2--;
                                                }
                                                k1++;
                                                for (i3 = 0; i3 < k3; i3++)
                                                        p[i3][i1] = 0;
                                        }
                                }
                        }
                }
        }
}

/****************/
/* ゲームパネル */
/****************/
class Puyo extends Panel {
        Game2 gm;

        /******************/
        /* コンストラクタ */
        /******************/
        Puyo(Game2 gm_t) {
                gm = gm_t;
                addKeyListener(new Key_e());
        }

        /********/
        /* 描画 */
        /********/
```

```
public void paint(Graphics g) {
        int i1, i2;
        // ゲーム中
        if (gm.game) {
                for (i1 = 0; i1 < gm.row; i1++) {
                        for (i2 = 0; i2 < gm.col; i2++) {
                                if (gm.p[i1][i2] > 0) {
                                        switch (gm.p[i1][i2]) {
                                        case 1:
                                                g.setColor(Color.red);
                                                break;
                                        case 2:
                                                g.setColor(Color.yellow);
                                                break;
                                        case 3:
                                                g.setColor(Color.green);
                                                break;
                                        case 4:
                                                g.setColor(Color.blue);
                                                break;
                                        case 5:
                                                g.setColor(Color.pink);
                                                break;
                                        }
                                        g.fillRect(i2 * 30, i1 * 30, 30, 30);
                                }
                        }
                }
        }
        // ゲームオーバー
        else {
                Font f = new Font("TimesRoman", Font.BOLD, 40);
                g.setFont(f);
                g.drawString("Game", 20, 200);
                g.drawString("Over!", 25, 250);
                gm.state = false;
        }
}

/************************/
/* キーイベントの有効化 */
/************************/
public boolean isFocusable() {
        return true;
}

/**********************/
/* キーイベントの処理 */
/**********************/
class Key_e extends KeyAdapter {
        public void keyPressed(KeyEvent e) {
                int k;
                boolean end = false;
```

```
        if (gm.ok) {

                if (e.getKeyCode() == KeyEvent.VK_LEFT) { // 左矢印（左移動）
                        if (gm.p_x > 0) {
                                if (gm.rot == 0 && gm.p[gm.p_y][gm.p_x - 1] == 0) {
                                        gm.p[gm.p_y][gm.p_x - 1] = gm.p[gm.p_y][gm.p_x];
                                        gm.p[gm.p_y][gm.p_x] = gm.p[gm.p_y][gm.p_x + 1];
                                        gm.p[gm.p_y][gm.p_x + 1] = 0;
                                        gm.p_x--;
                                        repaint();
                                } else if (gm.p[gm.p_y][gm.p_x - 1] == 0
                                        && gm.p[gm.p_y + 1][gm.p_x - 1] == 0) {
                                        gm.p[gm.p_y][gm.p_x - 1] =
                                                        gm.p[gm.p_y][gm.p_x];
                                        gm.p[gm.p_y + 1][gm.p_x - 1] =
                                                        gm.p[gm.p_y + 1][gm.p_x];
                                        gm.p[gm.p_y][gm.p_x] = 0;
                                        gm.p[gm.p_y + 1][gm.p_x] = 0;
                                        gm.p_x--;
                                        repaint();
                                }
                        }
                } else if (e.getKeyCode() ==
                        KeyEvent.VK_RIGHT) { // 右矢印（右移動）
                                if (gm.rot == 0) {
                                        if (gm.p_x < gm.col - 2
                                                && gm.p[gm.p_y][gm.p_x + 2] == 0) {
                                                gm.p[gm.p_y][gm.p_x + 2] =
                                                                gm.p[gm.p_y][gm.p_x + 1];
                                                gm.p[gm.p_y][gm.p_x + 1] =
                                                                gm.p[gm.p_y][gm.p_x];
                                                gm.p[gm.p_y][gm.p_x] = 0;
                                                gm.p_x++;
                                                repaint();
                                        }
                                } else {
                                        if (gm.p_x < gm.col - 1
                                                && gm.p[gm.p_y][gm.p_x + 1] == 0
                                                && gm.p[gm.p_y + 1][gm.p_x + 1] == 0) {
                                                gm.p[gm.p_y][gm.p_x + 1] =
                                                                gm.p[gm.p_y][gm.p_x];
                                                gm.p[gm.p_y + 1][gm.p_x + 1] =
                                                                gm.p[gm.p_y + 1][gm.p_x];
                                                gm.p[gm.p_y][gm.p_x] = 0;
                                                gm.p[gm.p_y + 1][gm.p_x] = 0;
                                                gm.p_x++;
                                                repaint();
                                        }
                                }
                        } else if (e.getKeyCode() ==
                                        KeyEvent.VK_UP) { // 上矢印（上下または左右
入れ替え）
                                        if (gm.rot == 0) {
```

```
                                        k = gm.p[gm.p_y][gm.p_x];
                                        gm.p[gm.p_y][gm.p_x] =
                                                gm.p[gm.p_y][gm.p_x + 1];
                                        gm.p[gm.p_y][gm.p_x + 1] = k;
                                } else {
                                        k = gm.p[gm.p_y][gm.p_x];
                                        gm.p[gm.p_y][gm.p_x] =
                                                gm.p[gm.p_y + 1][gm.p_x];
                                        gm.p[gm.p_y + 1][gm.p_x] = k;
                                }
                                repaint();
                        } else if (e.getKeyCode() == KeyEvent.VK_DOWN) {
                                // 下矢印 (90 度または-90 度回転)
                        if (gm.rot == 0 &&
                                gm.p[gm.p_y + 1][gm.p_x] == 0) {
                                if (gm.p_y < gm.row - 1) {
                                gm.p[gm.p_y + 1][gm.p_x] =
                                gm.p[gm.p_y][gm.p_x + 1];
                                gm.p[gm.p_y][gm.p_x + 1] = 0;
                                gm.rot = 1;
                                repaint();
                                }
                        } else {
                                if (gm.p_x < gm.col - 1
                                        && gm.p[gm.p_y][gm.p_x + 1] == 0) {
                                        gm.p[gm.p_y][gm.p_x + 1] =
                                        gm.p[gm.p_y + 1][gm.p_x];
                                        gm.p[gm.p_y + 1][gm.p_x] = 0;
                                        gm.rot = 0;
                                        repaint();
                                  }
                                }
                        }
                    }
                }
        }
}
```