TEX
java タイルベースゲーム

北野弘雪

2025 年 10 月 15 日

目次

1　ゲーム内容

題名：ねこのもり
内容：ネコがランダムに動くマップの中で、プレイヤーはネズミを操作（十字キー）して左上のチーズを制限時間に取るゲーム。
アピールポイント
1. ネコは通常はランダムで動いているが、プレーのいる行にネコがいると、プレイヤーを等速直線運動でネコが追いかけてくる。
2. プレイヤーがチーズを獲得すると、ネコが口を開け閉めするアニメーションが開始されて、プレイヤーを祝福してくれる。
3. プログラム上ネコが追いかけてく速さや、ランダムで動く速さを調整できる。難しくしたい人は速くできる。
4. アニメーションタイマーで、ネコがなめらかに動く。

2　アルゴリズム

2.1　ネコがランダムに動くアルゴリズム

ネコは一定時間ごとに移動して、上下左右の 4 方向に移動できるようにする。ただし、壁 (B) には移動できない。そして、上下左右に数字を割り振り、上１、下２、左３、右４とする。1〜4 の中で移動可能な方向の中から、ランダムに数字を選んで、方向を確定させてネコをランダムに移動させる。

2.2　ネコがネズミと同じ行にきたら追いかけてくるアルゴリズム

まず、最初に、ネコとネズミ（プレイヤー）が同じ高さにいるかどうかを判断する。次に、ネズミがネコの右にいるか、左にいるかをはんだんする。最後に、右にいたら、右へ移動させる。左にいたら、左に移動する。ことで、ネコがネズミを追尾する機能を実装した。

3　プログラムの説明・（クラス構造）

最初に、物理運動・アニメーションの内容を確認しておく。

3.1　物理運動

物理運動は、ネコがランダムで動くときと、ネコがネズミと同じ行にきたら、等速直線運動で追いかけてくるようにした。

3.2 アニメーション

アニメーションは、ネズミが、一番左上のチーズをとったら、ネコが口をあけしめして、画像が切り変わって、パクパクしているようにアニメーションをした。また、ネコがランダムで動く際、アニメーションタイマーでアニメーションさせている。

3.3 game2 クラス構造

- 変数の説明

表 1　game2 クラス における変数

型	変数名	意味
int	initialSceneWidth	初期画面の幅 (900)
int	initialSceneHeight	初期画面の高さ (950)
MapAndChars	mapAndChars	マップとキャラクターの情報を管理する MapAndChars クラスのインスタンス
AnimationTimer	timer	ゲームのアニメーションを制御するタイマー
long	startTime	ゲーム開始時刻を記録する変数
long	trophyGetTime	トロフィー取得時刻を記録する変数
Stage	primaryStage	JavaFX アプリケーションのメインウィンドウ

- メソッドの説明

表 2　game2 におけるメソッド

修飾子・返戻値	メソッド名・引数	役割
public void	‘start‘ ‘Stage st‘	JavaFX アプリケーションの開始時に呼び出されるメソッド。
‘private void‘	‘startGame‘	ゲームを開始するメソッド。ゲーム画面を作成し、タイマーを開始する。
‘private‘ ‘void‘	‘showClearScene‘	クリア画面を表示するメソッド。
‘private‘ ‘void‘	‘showGameOverScene‘	ゲームオーバー画面を表示するメソッド。
‘private‘ ‘void‘	‘showStartScene‘	スタート画面を表示するメソッド。
‘public‘ ‘static void‘	‘main‘ ‘String[] a‘	アプリケーションのエントリポイント。
‘public‘ ‘void‘	‘keyPressed‘ ‘KeyEvent e‘	キーボード入力を受け付けるメソッド。

3.4 MapAndChars クラス

- 変数の説明

表 3　MapAndChars クラス

型	変数名	意味
char[][]	map	マップの情報を二次元配列で保持 ('B': ブロック, 'L': はしご, 'T': トロフィー, 'G': 織姫（ネコ）（ねこ）, 'M': 彦星（ねずみ）, ' ': 空白)
int	MX	マップの横方向のサイズ (18)
int	MY	マップの縦方向のサイズ (20)
String[]	initialMap	マップの初期配置を表す文字列配列
int	boyX	彦星（ねずみ）の X 座標
int	boyY	彦星（ねずみ）の Y 座標
ImageView	boyView	彦星（ねずみ）の画像を表示するための ImageView
double	boySize	彦星（ねずみ）の画像のサイズ
Line	erase1, erase2	彦星（ねずみ）を消去するための線
List¡Girl¿	girls	織姫（ネコ）（ねこ）のリスト
Image	girlImage01, girlImage02	織姫（ネコ）（ねこ）の画像
Trophy	trophy	トロフィー
boolean	hasTrophy	トロフィーを持っているかどうか
boolean	gameOver	ゲームオーバーかどうか
Text	gameOverText	ゲームオーバー時に表示するテキスト
Text	timeText	残り時間を表示するテキスト

- メソッド説明

表 4　MapAndChars クラス

修飾子・返戻値	メソッド名・引数	役割
MapAndChars	Group root, int initialSceneWidth, int initialSceneHeight	コンストラクタ。マップとキャラクターの初期化を行う。
public boolean	hasTrophy	トロフィーを持っているかどうかを返す。
public void	drawInitialMapAndChars Group root	マップとキャラクターを初期状態で描画する。
public void	drawScreen int t	画面の描画処理を行う。
public void	drawBoy	彦星（ねずみ）を描画する。
public void	boyMove int dir	彦星（ねずみ）を移動させる。
public void	drawTrophy	トロフィーを描画する。
public void	drawGirls	織姫（ネコ）（ねこ）たちを描画する。
public void	moveGirls	織姫（ネコ）（ねこ）たちを移動させる。
public void	displayTime int t	残り時間を表示する。
public void	getItem	トロフィーを取得する。
public boolean	isGameOver	ゲームオーバーかどうかを判定する。

3.5　Girl クラス

- 変数の説明

表 5　変数とその意味

型	変数名	意味
double	x	織姫（ネコ）（ねこ）の X 座標
double	y	織姫（ネコ）（ねこ）の Y 座標
ImageView	view	織姫（ネコ）（ねこ）の画像を表示するための ImageView
int	moveCounter	織姫（ネコ）（ねこ）の移動を制御するためのカウンター
int	direction	織姫（ネコ）（ねこ）の移動方向 (0: 右, 1: 上, 2: 左, 3: 下)
int	speed	織姫（ネコ）（ねこ）の移動速度
int	danceCounter	織姫（ネコ）（ねこ）のダンスアニメーションを制御するためのカウンター
double	targetX, targetY	織姫（ネコ）（ねこ）の目標座標
double	moveX, moveY	1 フレームあたりの移動量

- メソッドの説明

表 6　メソッド一覧と役割

修飾子・返戻値	メソッド名・引数	役割
Girl	double x, double y, ImageView view	コンストラクタ。織姫（ネコ）（ねこ）の初期化を行う。
public void	draw	織姫（ネコ）（ねこ）を描画する。
public void	move char[][]map int boyX、 int boyY	織姫（ネコ）（ねこ）を移動させる。
public void	dance	織姫（ネコ）（ねこ）を踊らせる。
public double	getX	織姫（ネコ）（ねこ）の X 座標を返す。
public double	getY	織姫（ネコ）（ねこ）の Y 座標を返す。

3.6　Tile クラス

- 変数の説明

表 7　変数とその意味

型	変数名	意味
Group	parts	タイルの構成要素をまとめる Group

- メソッドの説明

表 8　メソッド一覧と役割

修飾子・返戻値	メソッド名・引数	役割
abstract Tile	int x, int y	コンストラクタ。タイルの初期化を行う。
public Group	getParts	タイルのパーツを返す。
abstract void	construct	タイルを構築する。

3.7　Ladder クラス

- 変数の説明

表 9　変数とその意味

型	変数名	意味
特に無し		

- メソッドの説明

表 10　メソッド一覧と役割

修飾子・返戻値	メソッド名・引数	役割
Ladder	int x, int y	コンストラクタ。はしごの初期化を行う。
public void	construct	はしごを構築する。

3.8　Block クラス

- 変数の説明

表 11　変数とその意味

型	変数名	意味
(なし)	(なし)	特に固有の変数なし

3.9　Trophy クラス

- メソッドの説明

表 12　メソッド一覧と役割

修飾子・返戻値	メソッド名・引数	役割
Image	trophyImage	トロフィーの画像
Canvas	canvas	トロフィーを描画するための Canvas
GraphicsContext	gc	canvas に描画するための GraphicsContext

- メソッドの説明

表 13　メソッド一覧と役割

修飾子・返戻値	メソッド名・引数	役割
Trophy	int initX, int initY	コンストラクタ。トロフィーの初期化を行う。
public void	draw	トロフィーを描画する。
public void	clear	トロフィーを非表示にする。
public Canvas	getCanvas	トロフィーの Canvas を返す。

4　プログラムの説明・(制御構造)

最初に、このゲームプログラムは、授業資料のサンプルコードを改変したもであるので、主な改変箇所を明記して説明する。

1. 複数の織姫（ネコ）の移動を制御する moveGirls メソッドを追加。2. drawGirls メソッドで、複数の織姫（ネコ）の描画を制御させている。3. displayTime メソッドで、時間の表示に加えて、ゲームオーバーやクリア時の表示も制御している。4. getItem メソッドで、アイテムを取得したときの処理に加えて、彦星（ねずみ）の位置を調整する処理も追加している。5. isGameOver メソッドで、ゲームオーバー条件を満たしているかどうかを判定している。
上の５つに加えて
ネコの追尾する・ネコがランダムで動く機能の説明をする。

また, タイトル画面やクリアー画面、ゲームオーバー画面は、ゲーム事体のコードに関わってないので省略する。

4.1 1.2 の girl クラス

Listing 1 synchronized boolean reserv

```
1  class Girl {
2    private double x, y;
3    private ImageView view;
4    private int moveCounter = 0;
5    private int direction = 0; // 0: right, 1: up, 2: left, 3: down
6    private int speed = 12; 数字を小さくすると速くなる//
7    private int danceCounter = 0;
8    private double targetX, targetY; // 目標座標
9    private double moveX, moveY; // フレームあたりの移動量 1
10
11   public Girl(double x, double y, ImageView view) {
12       this.x = x;
13       this.y = y;
14       this.view = view;
15       this.targetX = x;
16       this.targetY = y;
17       draw();
18   }
19
20   public void draw() {
21       view.setX(40 * x + 31);
22       view.setY(40 * y + 20);
23       view.toFront();
24   }
25
26   public void move(char[][] map, int boyX, int boyY) {
27       moveCounter++;
28       if (moveCounter % speed == 0) {
29           Random rand = new Random();
30           boolean moved = false;
```

```
31
32              if (boyY == (int) y) {
33                  if (boyX > x && map[(int) y][(int) x + 1] != 'B') {
34                      targetX = x + 1;
35                      moved = true;
36                  } else if (boyX < x && map[(int) y][(int) x - 1] != 'B') {
37                      targetX = x - 1;
38                      moved = true;
39                  }
40              }
41              if (!moved) {
42                  List<Integer> possibleDirections = new ArrayList<>(); // 移動可能な方向
                        を格納するリスト
43
44                  // 上下左右の方向をチェックし、移動可能な方向をリストに追加
45                  if (map[(int) y][(int) x + 1] != 'B') possibleDirections.add(0); // 右
46                  if (map[(int) y][(int) x - 1] != 'B') possibleDirections.add(2); // 左
47                  if (map[(int) y - 1][(int) x] != 'B') possibleDirections.add(1); // 上
48                  if (map[(int) y + 1][(int) x] != 'B') possibleDirections.add(3); // 下
49
50                  if (!possibleDirections.isEmpty()) {
51                      // 移動可能な方向がある場合、ランダムに方向を選択
52                      direction = possibleDirections.get(rand.nextInt(possibleDirections.
                            size()));
53
54                      // 選択した方向に移動
55                      switch (direction) {
56                          case 0: // right
57                              targetX = x + 1;
58                              break;
59                          case 1: // up
60                              targetY = y - 1;
61                              break;
62                          case 2: // left
63                              targetX = x - 1;
64                              break;
65                          case 3: // down
66                              targetY = y + 1;
67                              break;
68                      }
69                  }
70              }
71
72              // 目標座標への移動量を計算
73              moveX = (targetX - x) / speed;
74              moveY = (targetY - y) / speed;
```

```
75          }
76
77          x += moveX;
78          y += moveY;
79
80          if (map[(int) y][(int) x] == ' ' && map[(int) y + 1][(int) x] == ' ') {
81              targetY = y + 1;
82              moveY = (targetY - y) / speed;
83          }
84      }
85
86      public void dance() {
87          danceCounter = (danceCounter + 1) % 120;
88          switch (danceCounter) {
89              case 0:
90                  view.setImage(new Image("./image/paku1.png")); //
91                  break;
92              case 18:
93                  view.setImage(new Image("./image/paku2.png")); //
94                  break;
95              case 60:
96                  view.setImage(new Image("./image/paku1.png")); //
97                  break;
98              case 66:
99                  view.setImage(new Image("./image/paku2.png")); //
100                 break;
101             case 72:
102                 view.setImage(new Image("./image/paku1.png")); //
103                 break;
104             case 78:
105                 view.setImage(new Image("./image/paku2.png")); //
106                 break;
107             case 84:
108                 view.setImage(new Image("./image/paku1.png")); //
109                 break;
110             case 102:
111                 view.setImage(new Image("./image/paku2.png")); //
112                 break;
113             case 110:
114                 view.setImage(new Image("./image/paku1.png")); //
115                 break;
116             case 119:
117                 view.setImage(new Image("./image/paku2.png")); //
118                 break;
119
120         }
```

```
121     }
122
123     public double getX() {
124         return x;
125     }
126
127     public double getY() {
128         return y;
129     }
130 }
```

(orihime) はネコのキャラクターとしている。
Girl クラスは、ゲーム内の orihime のキャラクターを表現するためのクラスである。
各 orihime は、ゲームのマップ内を移動し、プレイヤーがトロフィーを持っていない場合はプレイヤーを追跡する。
プレイヤーがトロフィーを持っている場合は、orihime はダンスをする。ダンスは、画像を切り替えて、アニメーションとしている
1.move メソッドは、orihime（ネコ）の移動を制御するメソッドである。
このメソッドは、ゲームのマップ、彦星（ねずみ）の X 座標 (boyX)、彦星（ねずみ）の Y 座標 (boyY) を引数として受け取り、以下の手順で orihime の動きを制御する。
１. 移動カウンターをインクリメントする。
２．移動速度に応じて、移動処理を実行する。
３．彦星（ねずみ）が同じ高さにいる場合は、彦星（ねずみ）の方向に移動させる。
４．そうでない場合は、ランダムに移動方向を決定した。（switch 文使用）アルゴリズムでは、1,2,3,4 といったが、プログラムでは、0,1,2,3
５．目標座標に向かって移動させる。
６．orihime（ネコ）が落下する場合は、落下処理を実行した。
dance メソッドの詳細
orihime は配列で管理して、複数の orihime がそれぞれの動きができるようにしている。dance メソッドは、orihime のダンスアニメーションを制御するメソッドである。
このメソッドは、ダンスカウンターをインクリメントし、カウンターの値に応じて orihime の画像を切り替えることで、ダンスアニメーションを表現する。

● 3. の (ネコがランダムに動く) の具体的な処理
1.moveCounter をインクリメントし、speed で設定された速度に応じて移動処理を実行する。
2. 彦星が同じ Y 座標にいる場合、彦星の X 座標と織姫の X 座標を比較し、彦星が右にいるなら右へ、左にいるなら左へ移動するように targetX を設定する。
3. 彦星が同じ Y 座標にいない場合は、移動可能な方向（上下左右）をリスト possibleDirections に追加する。
4.possibleDirections が空でない場合、ランダムに方向を選択し、targetX または targetY を設定する。
5.moveX、moveY を計算し、織姫の現在位置 x、y を更新する。

6. 最後に、織姫が落下する状況であれば、落下処理を実行する。

● 4. の（ネコが追尾してくる）具体的な処理
1. 移動可能な方向をリストに追加: 織姫（ネコ）の現在位置から、上下左右の 4 方向について、移動可能かどうかをチェックする。移動可能な方向は、リスト possibleDirections に追加される。
・map[(int) y][(int) x + 1] != 'B' のように、マップのデータを参照して、壁 ('B') がないかを確認している。
2. ランダムに方向を選択: possibleDirections が空でない場合、つまり移動可能な方向がある場合は、rand.nextInt(possibleDirections.size()) を使って、リストからランダムに 1 つの方向を選択する。
3. 選択した方向に移動: 選択した方向に応じて、targetX または targetY を更新する。
・targetX と targetY は、織姫（ネコ）が次に移動する目標となる座標である。

4.2 displayTime

Listing 2 表示の改変、できない場合も表示

```
1  public void displayTime(int t) {
2    if (gameOver) {
3        timeText.setFill(Color.RED);
4        timeText.setText("Game Over");
5    } else if (hasTrophy) {
6        timeText.setFill(Color.YELLOW);
7        timeText.setText("Clear!");
8    } else {
9        timeText.setFill(Color.ORANGE);
10       timeText.setText("Time: " + t);
11       if (t == 0) gameOver = true;
12   }
13 }
```

displayTime メソッドのコード Java

public void displayTime(int t) if (gameOver) timeText.setFill(Color.RED); timeText.setText(”Game Over”); else if (hasTrophy) timeText.setFill(Color.YELLOW); timeText.setText(”Clear!”); else timeText.setFill(Color.ORANGE); timeText.setText(”Time: ” + t); if (t == 0) gameOver = true; コードの説明このメソッドは、ゲームの残り時間、ゲームオーバー、クリアの状態を表示するためのメソッドである。

引数は t: ゲームの残り時間
処理の流れ
if (gameOver): ゲームオーバーフラグ (gameOver) が true の場合
timeText.setFill(Color.RED): 表示するテキストの色を赤色に設定する。
timeText.setText(”Game Over”): テキストを”Game Over”に設定する。
else if (hasTrophy): ゲームオーバーフラグが false で、トロフィー取得フラグ (hasTrophy) が true の場合
timeText.setFill(Color.YELLOW): 表示するテキストの色を黄色に設定する。
timeText.setText(”Clear!”): テキストを”Clear!”に設定します。

else: ゲームオーバーフラグとトロフィー取得フラグが共に false の場合
timeText.setFill(Color.ORANGE): 表示するテキストの色をオレンジ色に設定する。
timeText.setText(”Time: ” + t): テキストを”Time: ”と残り時間を連結した文字列に設定する。
if (t == 0) gameOver = true: 残り時間が 0 になった場合、ゲームオーバーフラグを true に設定する。

4.3 メソッド getItem

Listing 3 getItem

```
1  public void getItem() {
2    hasTrophy = true;
3    boyView.setX(40 * boyX + 31);
4    boyView.setY(40 * boyY + 20);
5  }
```

このメソッドは、彦星（ねずみ）がトロフィーを取得したときに呼ばれるメソッドである。

1. hasTrophy = true;: トロフィー取得フラグ (hasTrophy) を true に設定する。

●これで、ゲーム内でトロフィーを取得した状態になる。
●このフラグは、moveGirls メソッドなどで、orihime（ネコ）の行動を変化させるために使われる。
2. boyView.setX(40 * boyX + 31);: 彦星（ねずみ）の画像の X 座標を調整します。
● boyX は彦星の X 座標（タイル座標）である。
● 40 * boyX でタイル座標をピクセル座標に変換し、+ 31 で画像の位置を微調整している。
3. boyView.setY(40 * boyY + 20);: 彦星（ねずみ）の画像の Y 座標を調整している。
● boyY は彦星の Y 座標（タイル座標）である。
● 40 * boyY でタイル座標をピクセル座標に変換し、+ 20 で画像の位置を微調整している。

4.4 isGameOver メソッド

Listing 4 isGameOver

```
1  public boolean isGameOver() {
2      for (Girl girl : girls) {
3          if (boyX == (int) girl.getX() && boyY == (int) girl.getY()) {
4              gameOver = true;
5              return true; // ゲームオーバーを検知したらすぐにtrue を返す
6          }
7
8      }
9      return false;
10 }
```

このメソッドは、ゲームオーバー条件を満たしているかどうかを判定するメソッドである。具体的には、彦星（ねずみ）と orihime（ネコ）が同じ位置にいるかどうかをチェックする。処理の流れ 1. for (Girl girl : girls): すべての orihime（ネコ） (Girl オブジェクト) について繰り返し処理を行う。
2. if (boyX == (int) girl.getX() & & boyY == (int) girl.getY()): 彦星（ねずみ）の X 座標 (boyX) と orihime（ネコ）の X 座標 (girl.getX()), 彦星（ねずみ）の Y 座標 (boyY) と orihime（ネコ）の Y 座標 (girl.getY()) を比較する。
●どちらも等しい場合、彦星と orihime（ネコ）が同じ位置にいると判定される。
3. gameOver = true;: ゲームオーバーフラグ (gameOver) を true に設定した。
・このフラグは、displayTime メソッドなどで、ゲームオーバーの表示を制御するために使用された。
4. return true;: true を返して、ゲームオーバー条件を満たしていることを示す。
・同じ位置にいる orihime（ネコ）が見つかった時点で、ループを終了し、すぐに true を返す。
5. return false;: すべての orihime（ネコ）をチェックしても、彦星と同じ位置にいる orihime（ネコ）が見つからなかった場合、false を返して、ゲームオーバー条件を満たしていないことを示す。

5　実行結果の評価

5.1　タイトル画面

図 1　タイトル画面

5.2　ゲーム画面

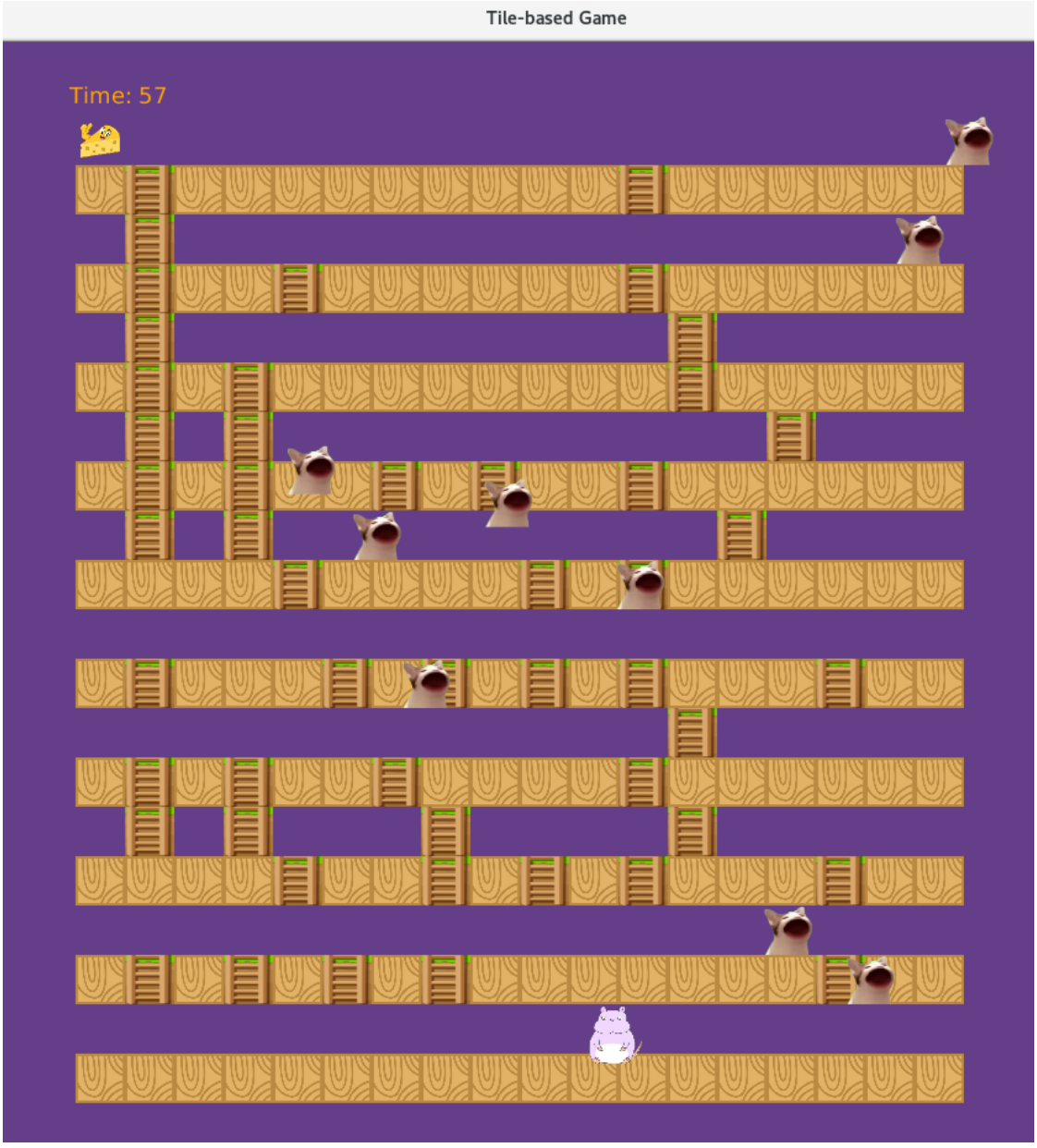

図 2　ゲーム画面

5.3　ゲームオーバー画面

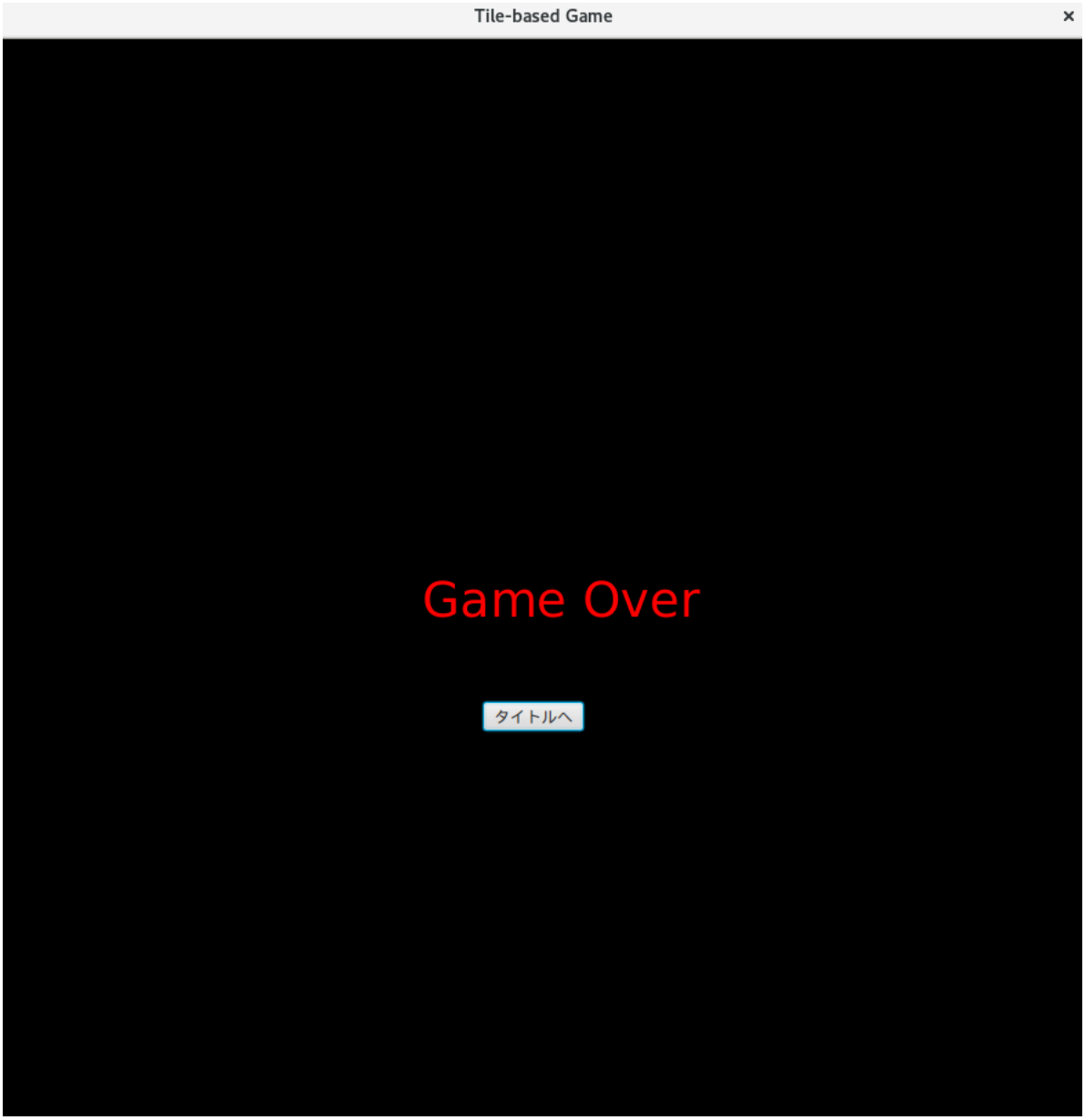

図 3　ゲームオーバー画面

5.4　チーズを取った後

5.4.1　パクパクする過程１

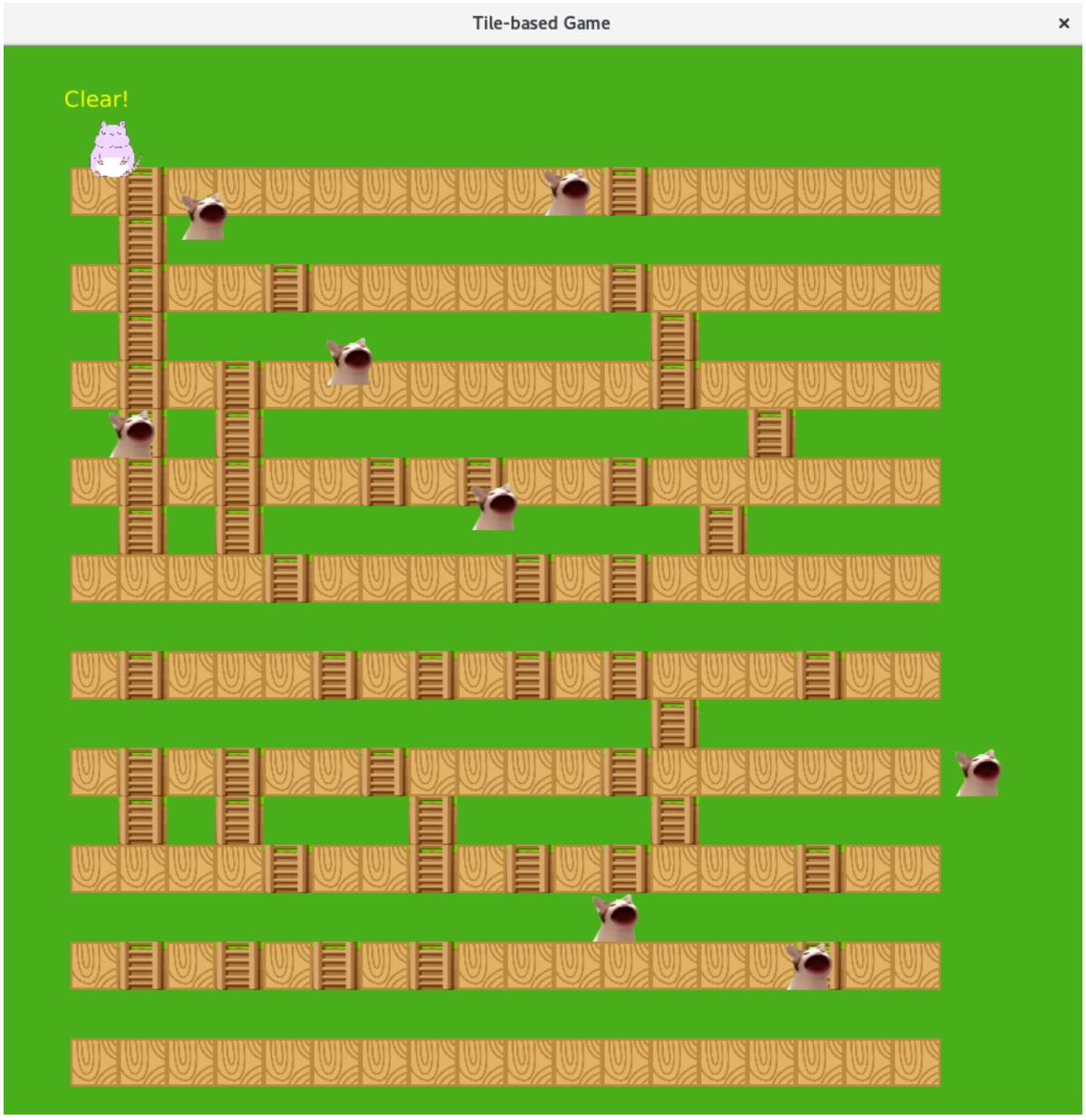

図 4　獲得後

5.4.2　パクパクする過程２

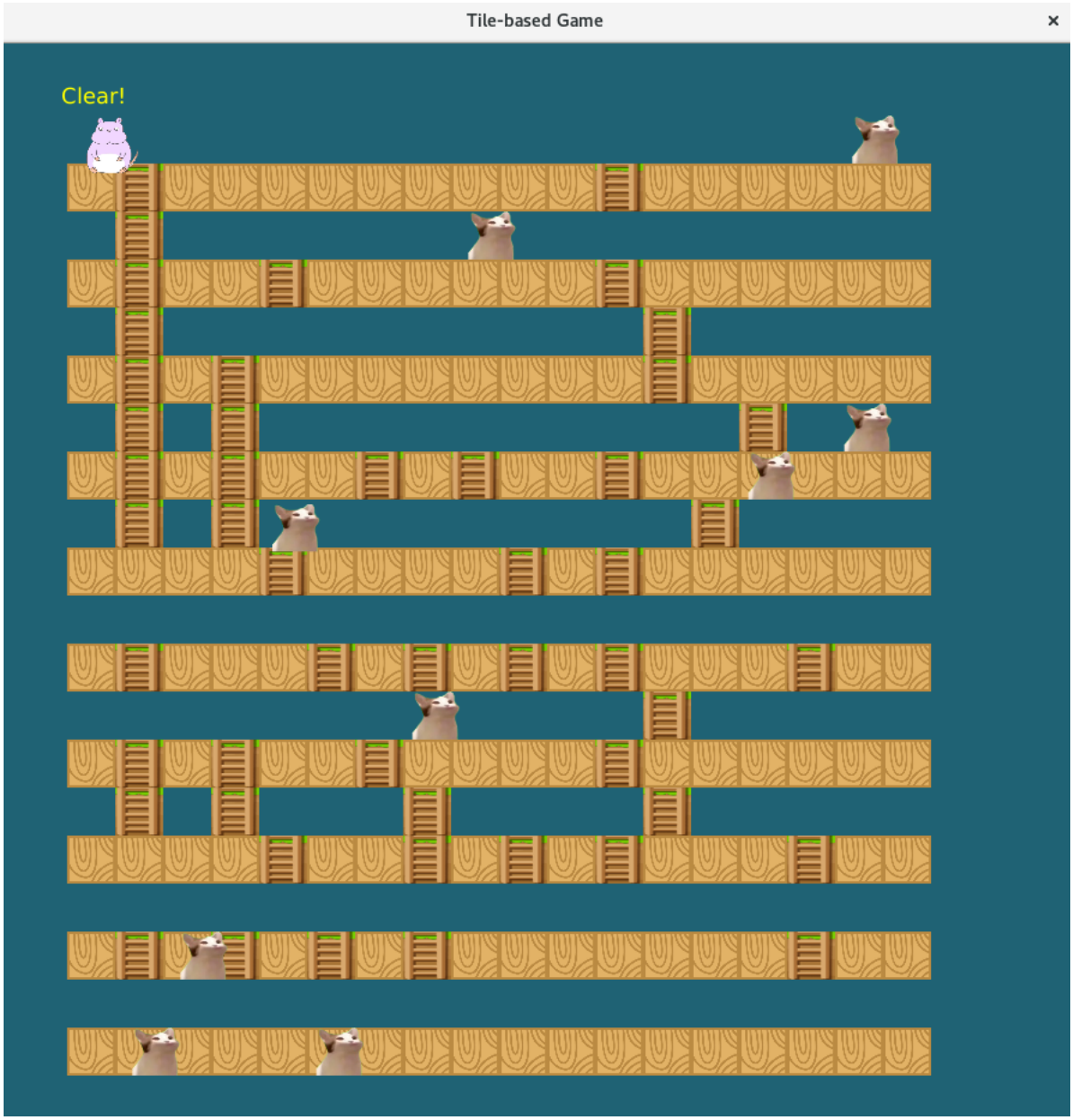

図５　獲得後

5.5　ゲームクリア画面

図６　ゲームクリア画面

5.6　評価

5.6.1　ゲーム

　ゲームの画面は４画面で、ゲームの概要と説明のタイトル画面とゲームをする画面そして、ゲームのクリアー画面、そしてゲームオーバー画面である。
４つしっかりうつっており、チーズをとった後は、ネコたちが、図４と図５のネコの画像が交互に動き、ネコ

がパクパクしているように見える。
また、ゲームをするときの背景は、色がランダムで出力されるようになっている。

6　考察

今回は、2 次元マップを使った、タイルベースゲームであり、かつ、物理法則とアニメーションを加える条件つきであったため、授業資料のコードを参考にして、シンプルなおっかけっこゲームを作った。最初は、ゲームの青鬼が著作権もフリーで鬼ごっこのよなゲームであったので最初は青鬼を作ろうとしたが、逃げるプレイヤーをずっと追尾するアルゴリズムが思いつかなかったので、同じ行にきたら追尾できるよに実装した。また、ネコの追いかけて来る速さを調整できるようにプログラムし、ネコの追いかけて来る速さをランダムにしてプログラムしたら面白いと思ったが、ランダムにすると、ネコが物凄いスピードで追いかけてきて、ゲームとして成り立たなくなるので、ネコが追いかけて来る速さは固定とした。また、ゲームをしていると、ネコが、B のブロックの中をすすんできてしまう事になってしまった。B のブロックは、ネコも、進めないように、プログラムしているつもりだったが、Bのブロックの中やブロックをすり抜けていくバグのような仕様になってしまった。考えられるのは、Lのはしごのプログラムにおいて、はしごを通る時に、ネコが、進む方向をランダムで割り当てられるのではしご通過中に、ブロックの方向に動こうとするとブロックの中に入って動けるよになったのではと考えている。最後に、今回は、java で学校の環境でゲームを作ったのでそれが影響して exe ファイルの作成が困難でもあった。

7　ソースコード

game1 ファイルの中：
game1.java（プログラムソース）
image　　（画像ファイル）