# 知能プログラミング演習II 課題5

グループ8
29114142 湯浅範子

2019年12月9日

**提出物**　rep5(2914142.pdf),group08.zip

**グループ**　グループ8

**メンバー**

| 学生番号 | 氏名 | 貢献度比率 |
|---|---|---|
| 29114003 | 青山周平 | |
| 29114060 | 後藤拓也 | |
| 29114116 | 増田大輝 | |
| 29114142 | 湯浅範子 | |
| 29119016 | 小中祐希 | |

## 1　課題の説明

**必須課題 5-1** 目標集合を変えてみたときに，動作が正しくない場合があったかどうか，実行例を示して考察せよ．また，もしあったならその箇所を修正し，どのように修正したか記せ．

**必須課題 5-2** 教科書のプログラムでは，オペレータ間の競合解消戦略としてランダムなオペレータ選択を採用している．これを，効果的な競合解消戦略に改良すべく考察し，実装せよ．改良の結果，性能がどの程度向上したかを定量的に（つまり数字で）示すこと．

**必須課題 5-3** 上記のプランニングのプログラムでは，ブロックの属性（たとえば色や形など）を考えていないので，色や形などの属性を扱えるようにせよ．ルールとして表現すること．例えば色と形の両方を

扱えるようにする場合，A が青い三角形，B が黄色の四角形，C が緑の台形であったとする．その時，色と形を使ってもゴールを指定できるようにする（”green on blue” や”blue on box”のように）

**必須課題 5-4** 上記 5-2, 5-3 で改良したプランニングシステムの GUI を実装せよ．ブロック操作の過程をグラフィカルに可視化し，初期状態や目標状態を GUI 上で変更できることが望ましい．

**発展課題 5-5** ブロックワールド内における物理的制約条件をルールとして表現せよ．例えば，三角錐（pyramid）の上には他のブロックを乗せられない等，その世界における物理的な制約を実現せよ．

**発展課題 5-6** ユーザが自然言語（日本語や英語など）の命令文によってブロックを操作したり，初期状態／目標状態を変更したりできるようにせよ．なお，命令文の動詞や語尾を 1 つの表現に決め打ちするのではなく，多様な表現を許容できることが望ましい．

**発展課題 5-7** 3 次元空間 (実世界) の物理的な挙動を考慮したブロックワールドにおけるプランニングを実現せよ．なお，物理エンジン等を利用する場合，Java 以外の言語のフレームワークを使って実現しても構わない．

**発展課題 5-8** 教科書 3.3 節のプランニング手法を応用できそうなブロック操作以外のタスクをグループで話し合い，新たなプランニング課題を自由に設定せよ．さらに，もし可能であれば，その自己設定課題を解くプランニングシステムを実装せよ．

私は必須課題 5-4 の GUI 実装を行ったため，それについて記述する．

## 2 必須課題 5-4

> 上記 5-2, 5-3 で改良したプランニングシステムの GUI を実装せよ．ブロック操作の過程をグラフィカルに可視化し，初期状態や目標状態を GUI 上で変更できることが望ましい．

私の担当箇所は，得られた結果を基にした GUI 本体の実装である．

## 2.1 手法

GUI で求められる出力が行えるように，以下のような機能を加えた．

- ブロック操作の過程を示す
- ブロック操作の過程をグラフィカルに可視化する
- 初期状態と目標状態を GUI 上から画面入力で変更できる
- 属性の決定を GUI 上から行える
- 操作に必要なオペレータと属性のデータを表示する
- 禁止制約によってブロック操作が停止した場合にそれを表示する

これらのプログラムを実装するためのデータ受け渡し部分は青山君が作成してくれるため，ここでは受け取ったデータを基にしてどのような方法で表示を行うか考えながら実装を行った．実装は Swing によって行った．

また今回は初めての GUI 作成であったため，第一週では受け取るデータを既に取得したと仮定し，自ら作成した仮データを基にして GUI についての検討を行った．第二週では実際にデータを受け取り，そのデータを基に正しく描画やプランニングが行えるようにプログラムを改良した．

## 2.2 実装

作成した GUI プログラムの中の主部分についての実装を以下に示す．

まず，GUI プログラムの mein メソッドをソースコード 1 に示す．

ソースコード 1: main メソッド

```
1 public static void main(String[] args){
2         // 画像の定義
3         images = new String[5][4];
4         images[0][0] = "image/squBt.png";
5         …
6         images[4][3] = "image/daiDeft.png";
```

```
7
8           // 対応番号の指定 (色･形それぞれで定義)
9           imageMapC = new HashMap<>();
10          imageMapC.put("Blue", 0);
11          …
12          imageMapC.put("Default", 4);
13          imageMapS = new HashMap<>();
14          …
15
16          // 属性指定での入出力に対応 (default は予め弾く)
17          Attribution = new HashMap<>();
18          Attribution.put("Blue", 1);
19          …
20          Attribution.put("Yellow", 1);
21          Attribution.put("box", 2);
22          …
23          Attribution.put("trapezoid", 2);
24
25          PGUI frame = new PGUI();
26          frame.setDefaultCloseOperation(JFrame.
                EXIT_ON_CLOSE);
27          frame.setBounds(10, 10, 650, 450);
28          frame.setTitle("プランニングシステム");
29          frame.setVisible(true);
30  }
```

ここでフレームの大きさと表示位置・タイトルや，フレームを閉じると同時にプログラムを終了させるなどの動作を定めた．さらに，プログラム中で使用する変数であらかじめ値を設定しておくものの定義を行った．

次に，GUI プログラムのコンストラクタをソースコード 2 に示す．

ソースコード 2: コンストラクタ

```
1   PGUI(){
2           // プレゼンターとの連結
3           presenter = new Presenter();
4           // 結果の格納 (メソッド呼び出し)
5           ArrayList<String> result = presenter.getPlan();
6           results = new ArrayList<>(result);
7           // 初期状態の追加
8           results.add(0, "default position");
9           // 結果ステップデータの取得
```

```
10          pUR = presenter.getStepList();
11          // 入力デフォルト値の格納 (メソッド呼び出し)
12          initialState = presenter.getInitialState();
13          // 出力デフォルト値の格納 (メソッド呼び出し)
14          goalList = presenter.getGoalList();
15          // 初期状態の格納
16          String[] initialName = {"A", "B", "C"};
17          String[] initialAColor = {"Default", "Default", "
                Default"};
18          String[] initialAShape = {"default", "default", "
                default"};
19          for (int i = 0; i < initialName.length; i++) {
20                  modelName.addElement(new String(
                        initialName[i]));
21                  modelColor.addElement(new String(
                        initialAColor[i]));
22                  modelShape.addElement(new String(
                        initialAShape[i]));
23          }
24          // 禁止制約の格納
25          prohibitRules.add("box on pyramid");
26          ...
27          prohibitRules.add("trapezoid on ball");
28
29          // 2ページ目以降のカード作成用メソッド
30          createResultPage(pUR);
31          // ボタンの作成メソッド
32          createButton();
33          // 最終処理メソッド
34          finishData();
35  }
```

PGUI コンストラクタで Presenter プログラムを起動し，プランニングを行った結果を取得する．また同時に初期状態を GUI で表示するため，これらの情報も格納した．

ここから，先に述べた各機能の実装について詳しく記述していく．
まず始めにブロック操作の過程を示すプログラムをソースコード 3 に示す．

ソースコード 3: 過程表示

```
1  JPanel tostring2 = new JPanel();
```

```
2 tostring2.setLayout(new BoxLayout(tostring2, BoxLayout.
    PAGE_AXIS));
3 tostring2.setBackground(Color.WHITE);
4 ArrayList<String> printResult = presenter.getPlan();
5 tostring2.add(new JLabel("***** This is a plan!
    *****"));
6 for (String printR : printResult) {
7         tostring2.add(new JLabel(printR));
8 }
9 JScrollPane scrollpane2 = new JScrollPane(tostring2);
10 scrollpane2.setPreferredSize(new Dimension(200, 310));
11 BevelBorder border2 = new BevelBorder(BevelBorder.LOWERED
    );
12 scrollpane2.setBorder(border2);
13 JPanel Plan = new JPanel();
14 Plan.setLayout(new BoxLayout(Plan, BoxLayout.PAGE_AXIS));
15 Plan.add(new JLabel("Plan "));
16 Plan.add(scrollpane2);
```

ブロック操作の過程を示すため，新しく作成したパネルに受け取ったリストデータをラベルとして加え，そのパネルをフレームに追加することで表示を行った．また経路が長くなったときを考え，必要に応じてスクロールバーを追加できるようにした．

次にブロック操作の過程をグラフィカルに可視化するためのプログラムをソースコード 4，5 に示す．このとき初期状態は別で取得するため，初期状態のみを描画するプログラムと，以降の過程を描画するプログラムを分けて作成した．

ソースコード 4: グラフィカル表示 1(一部抜粋)

```
1 // 変数決定,属性名の初期化,配置用･アーム用座標配列を定義
2 …
3 // 2ページ目の設定
4 JPanel page2 = new JPanel();
5 JLabel[][] p2Label = new JLabel[row][col];
6 GridLayout page2layout = new GridLayout();
7 page2layout.setRows(row); // 行数
8 page2layout.setColumns(col); // 列数
9 page2.setLayout(page2layout);
10 // テーブルの上に乗っているブロックの初期化
11 int next = 0;
```

```
12 for (String dataS : dataTable) {
13         String[] state = dataS.split(" ", 0);
14         for (int i = 0; i < blocks.size(); i++) {
15                 // 初めに入手したブロック名が何番目のものか
                       check
16                 if (state[1].equals(blocks.get(i))) {
17                         // 名称一致のとき
18                         iconX[i] = row - 1;
19                         iconY[i] = next;
20                         next++;
21                 }
22         }
23 }
24 // 他ブロックの上に乗っているブロックの初期化
25 for (String dataS : dataOn) {
26         String[] state = dataS.split(" ", 0);
27         for (int i = 0; i < blocks.size(); i++) {
28                 // それぞれの属性の番号を取得
29                 if (state[0].equals(blocks.get(i))) {
30                         ue = i;
31                 } else if (state[2].equals(blocks.get(i)))
                           {
32                         sita = i;
33                 }
34         }
35         // 上部分の座標の確定
36         iconX[ue] = iconX[sita] - 1;
37         iconY[ue] = iconY[sita];
38 }
39 // blockとアームの上書き
40 for (int i = 0; i < blocks.size(); i++) {
41         p2Label[iconX[i]][iconY[i]] = new JLabel(icon[i
               ]);
42         p2Label[iconX[i]][iconY[i]].setText(iconName[i]);
43 }
44 p2Label[armX][armY] = new JLabel(arm);
45 p2Label[armX][armY].setText(armname);
46 // アイコンの挿入
47 for (int i = 0; i < row; i++) {
48         for (int j = 0; j < col; j++) {
49                 page2.add(p2Label[i][j]);
50         }
```

```
51 }
```

ソースコード 5: グラフィカル表示 2(一部抜粋)

```
1  // 3ページ目以降
2  for (int i = 0; i < cardPage; i++) {
3          // 初期化 (2ページ目作成と同様の操作)
4          …
5          String hatenaX = pUR.get(i).getBindings().get("?x
               ");
6          String hatenaY = pUR.get(i).getBindings().get("?y
               ");
7          int hXz = blocks.indexOf(hatenaX);
8          int hYz = blocks.indexOf(hatenaY);
9          if (pUR.get(i).getName().equals("Place ?x on ?y
               ")) {
10             // xの操作
11             iconX[hXz] = iconX[hYz] - 1;
12             iconY[hXz] = iconY[hYz];
13             // アームの操作
14             armX = iconX[hXz] - 1;
15             armY = iconY[hYz];
16         } else if (pUR.get(i).getName().equals("remove ?x
                from on top ?y")) {
17             // xの操作
18             iconX[hXz] = iconX[hXz] - 1;
19             iconY[hXz] = iconY[hYz];
20             // アームの操作
21             armX = iconX[hXz] - 1;
22             armY = iconY[hYz];
23         } else if (pUR.get(i).getName().equals("pick up ?
               x from the table")) {
24             // xの操作
25             iconX[hXz] = iconX[hXz] - 1;
26             iconY[hXz] = iconY[hXz];
27             // アームの操作
28             armX = iconX[hXz] - 1;
29             armY = iconY[hXz];
30         } else if (pUR.get(i).getName().equals("put ?x
               down on the table")) {
31             // xの操作,アームの操作
32             iconX[hXz] = row - 1;
33             armX = iconX[hXz] - 1;
```

```
34                  boolean umu;
35                  for (int j = 0; j < col; j++) {
36                          umu = true;
37                          for (int k = 0; k < iconY.length;
                                k++) {
38                                  if (j == iconY[k]) {
39                                          umu = false;
40                                          break;
41                                  }
42                          }
43                          if (umu == true) {
44                                  iconY[hXz] = j;
45                                  armY = j;
46                                  break;
47                          }
48                  }
49          }
50          // block・アームの上書き，アイコンの挿入
51          …
52  }
```

上のように，初期状態は別のメソッドから受け取り予め描画を行い，そのデータを基にしてブロック操作の過程を描画した．また描画は座標で管理を行った．

さらに，初期状態と目標状態・各ブロックの属性を GUI 上から変更，表示するためのプログラムをソースコード 6 と 7 に示す．

ソースコード 6: 初期状態と目標状態の変更

```
1  // 手動入力用パネル
2  JPanel natural = new JPanel();
3  natural.setLayout(new BoxLayout(natural, BoxLayout.
       PAGE_AXIS));
4  // 入力 (setInitialState)
5  JPanel sI = new JPanel();
6  iArea = new JTextArea(9, 20);
7  JScrollPane iScroll = new JScrollPane(iArea);
8  String ii = "";
9  for(String i : initialState) {
10         ii += i + "\n";
11 }
12 iArea.setText(ii);
```

```
13 sI.add(iScroll);
14 // 入力 (setGoal)
15 JPanel sG = new JPanel();
16 gArea = new JTextArea(4, 20);
17 JScrollPane gScroll = new JScrollPane(gArea);
18 String gg = "";
19 for(String g : goalList) {
20         gg += g + "\n";
21 }
22 gArea.setText(gg);
23 sG.add(gScroll);
24 natural.add(sI);
25 natural.add(sG);
```

JTextArea を用いて文字列の入力が GUI 上で行えるようにした．また，入力文が増えた場合のためスクロールバーを必要に応じて表示した．

ソースコード 7: 各ブロックの属性変更

```
1  JPanel allRadio = new JPanel();
2  // 色選択
3  JPanel p2 = new JPanel();
4  radio = new JRadioButton[4];
5  radio[0] = new JRadioButton("Blue");
6  ···
7  // ボタンのグループ化
8  ButtonGroup group = new ButtonGroup();
9  group.add(radio[0]);
10 ···
11 p2.add(new JLabel("Select Color"));
12 p2.add(radio[0]);
13 ···
14 // 形状選択 (色選択と同様の操作)
15 ···
16
17 // 属性編集用ボタンの作成
18 JPanel ADS = new JPanel();
19 ADS.add(new JLabel("new Name "));
20 newNameText = new JTextField(5);
21 ADS.add(newNameText);
22 JButton add = new JButton("追加");
23 add.addActionListener(this);
```

```
24 add.setActionCommand("addButton");
25 ADS.add(add);
26 ···
27 allRadio.add(ADS);
28 allRadio.add(p2);
29
30 // 属性入力用パネルの作成
31 JPanel attribution = new JPanel();
32 JPanel p4 = new JPanel();
33 // 属性の決定用パネル
34 JPanel nameD = new JPanel();
35 nameD.add(new JLabel("Determine Attribution "));
36 p4.add(nameD);
37 JPanel attribute = new JPanel();
38 JPanel NAME = new JPanel();
39 ···
40 // リストで実現
41 namelist = new JList(modelName);
42 JScrollPane namesp = new JScrollPane();
43 namesp.getViewport().setView(namelist);
44 NAME.add(namesp);
45 attribute.add(NAME);
46 // 色選択リストパネル
47 colorlist = new JList(modelColor);
48 JScrollPane colorsp = new JScrollPane();
49 colorsp.getViewport().setView(colorlist);
50 // リストを選択不可にする
51 colorlist.setEnabled(false);
52 colorsp.setBorder(borderC);
53 JPanel COLOR = new JPanel();
54 ···
55 COLOR.add(colorsp);
56 attribute.add(COLOR);
57 // 形状選択リストパネル (色選択と同様の操作)
58 ···
59 p4.add(attribute);
```

今回の実装では色と形状は任意に選択することが出来ないため，これらをラジオボタンによりユーザーが選択するようにした．さらに，これらのデータの追加・削除・編集も行えるよう，属性データは JList を用いて管理を行った．また，新しい属性名の追加のためテキストフィールドを作成した．

また，操作に必要なオペレータを表示するプログラムをソースコード 8 に示す.

ソースコード 8: オペレータ表示プログラム

```
JPanel tostring = new JPanel();
tostring.setLayout(new BoxLayout(tostring, BoxLayout.
    PAGE_AXIS));
for (Operator operator : operators) {
        tostring.add(new JLabel("●Operator" + i));
        tostring.add(new JLabel("NAME: " + operator.
            getName()));
        tostring.add(new JLabel("ADD: " + operator.
            getAddList()));
        tostring.add(new JLabel("DELETE: " + operator.
            getDeleteList()));
        }
}
JScrollPane scrollpane = new JScrollPane(tostring);
scrollpane.setBorder(border);
```

最後に，禁止制約によってブロック操作が停止した場合の処理を行うプログラムをソースコード 9 に示す.

ソースコード 9: ブロックの状態を表すプログラム

```
JPanel prohibit = new JPanel();
JPanel hosoku = new JPanel();
LineBorder inborder = new LineBorder(Color.red, 2);
TitledBorder border = new TitledBorder(inborder,"Warning
    !!", TitledBorder.LEFT, TitledBorder.TOP);
JLabel setumei = new JLabel(" This Goal is not allowed by
     ProhibitRules");
hosoku.add(setumei);
…
hosoku.setBorder(border);
JPanel kari = new JPanel();
kari.add(hosoku, BorderLayout.CENTER);
prohibit.add(kari);
// 禁止制約のパネルの表示
// 属性名
JPanel prohibit2_1 = new JPanel();
for (int i = 0; i < modelName.size(); i++) {
```

```
16          StringBuilder buf_1 = new StringBuilder();
17          buf_1.append((String)modelName.get(i));
18          buf_1.append(" is ");
19          …
20          buf_1.append((String)modelShape.get(i));
21          prohibit2_1.add( new JLabel( buf_1.toString()) );
22  }
23  JScrollPane scrollpane1 = new JScrollPane(prohibit2_1);
24  JPanel AN = new JPanel();
25  JPanel an = new JPanel();
26  an.add( new JLabel("Attribution ") );
27  …
28  AN.add(an);
29  AN.add(scrollpane1);
30  // 禁止制約と目標状態 (属性名と同様の操作を行う)
31  …
32  // 目標状態･禁止制約･属性名の設定
33  JPanel details = new JPanel();
34  details.add(AN);
35  details.add(PR);
36  details.add(GL);
37  prohibit.add(details);
```

目標状態が禁止制約に当たる場合は，目標状態の実現が不可能であるため，その目標状態が禁止制約に当たることを知らせる必要がある．今回はそのために新しくページを追加し，設定した属性名と禁止制約・目標状態を表示させ，この状態が禁止制約に当たることを知らせることでブロック操作が停止した場合の処理を行った．

## 2.3 実行例

作成したプログラムの実行結果を順に示す．

実行すると初めに初期状態・目標状態・属性の決定画面になる (図 1). デフォルトでは ABC の 3 つのブロックが存在し，その属性は全てデフォルト (黒色の四角形) で定義されている.

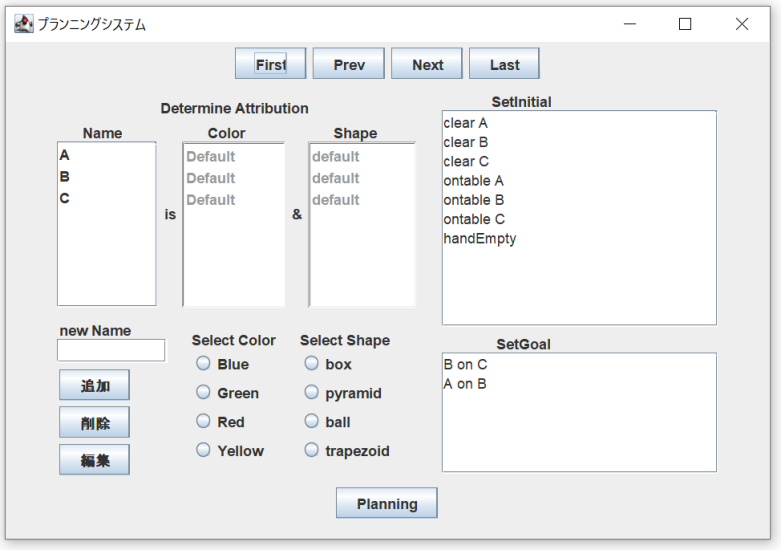


図 1: 初期状態と目標状態，属性の決定画面

初めに実行を行った段階で，デフォルト状態でのプランニングが行われているため，ページ上部の Next ボタンを押すと次の画面に遷移し，ブロック操作の過程の描画画面となる．Next ボタンを押すと次の状態へ，Prev ボタンを押すと一つ前の状態へ遷移する様子が確認できる．(図 2，図 3，図 4)．

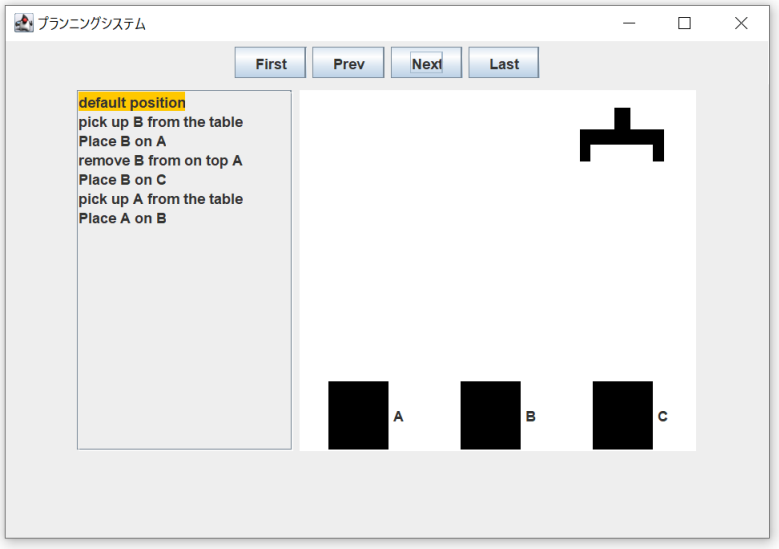


図 2: ブロック操作の過程 1

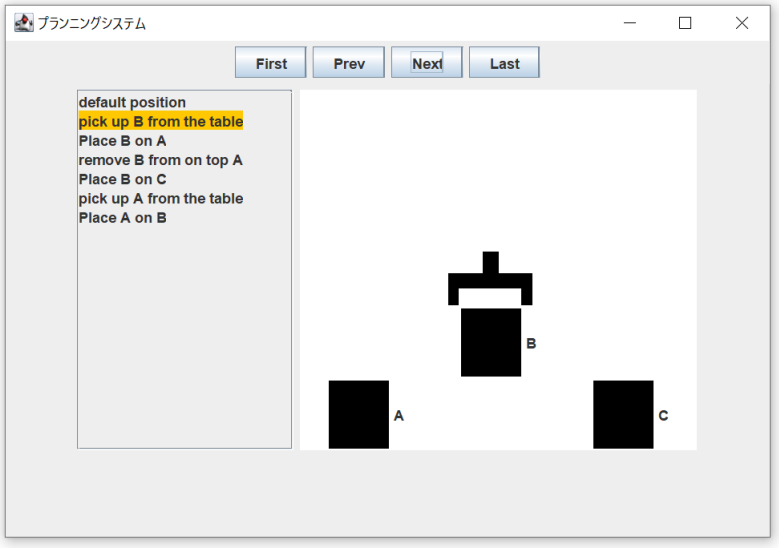


図 3: ブロック操作の過程 2

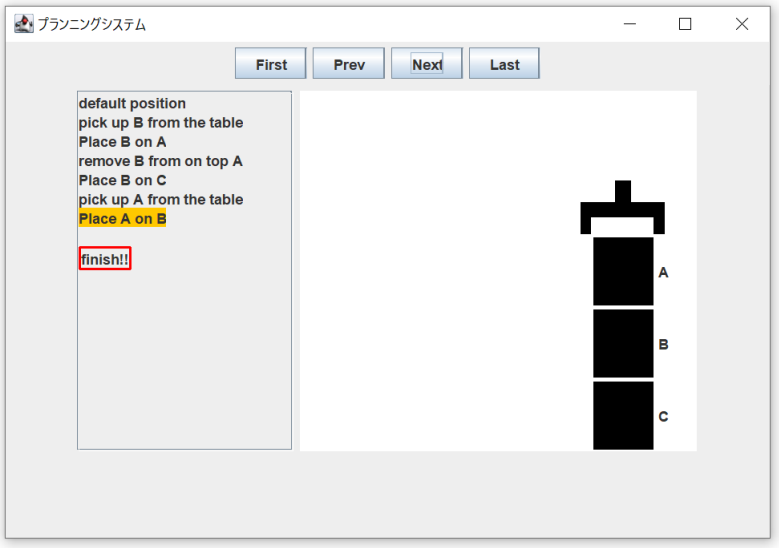


図 4: ブロック操作の過程 3

さらに，First ボタンを押すと一番初めの設定画面へ，Last ボタンを押すと一番最後の画面であるオペレータとプランニング結果を表す画面へと遷移する (図 5).

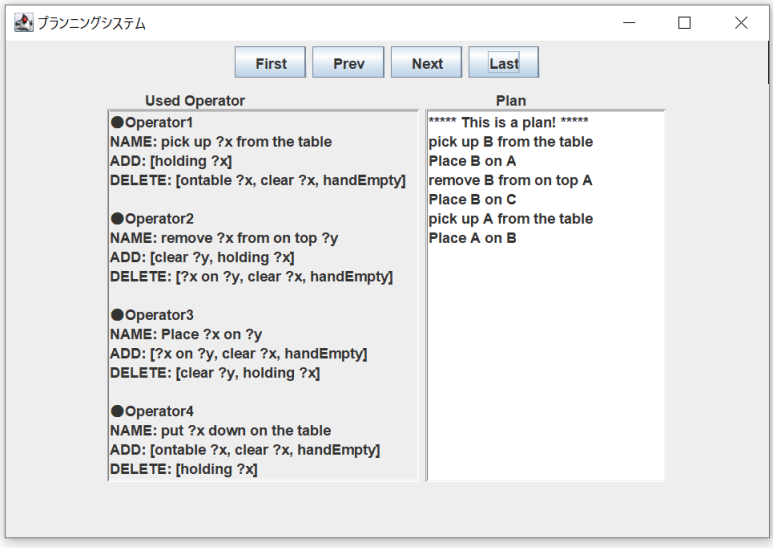


図 5: オペレータとプランニング結果の情報

次に，初期状態，目標状態，属性情報を変更してプランニングを行う．まず初期画面の左上に現在の属性情報があり，これを左下のラジオボタンを追加・削除・編集ボタンで変更する (図 6)．

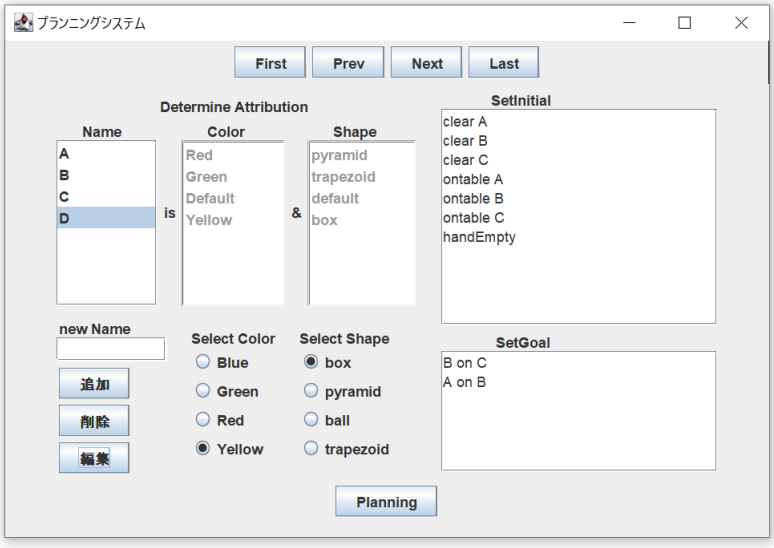


図 6: 属性情報の変更

また，ブロック追加時には左下にある’new Name’ 欄に新規ブロックの名前を入力する必要があるが，無記入で追加をした場合や既にある名前を入力した際にはエラーメッセージがダイアログウィンドウで表示されるようになっている (図 7，図 8)．編集・削除ボタンを押した際に属性リストの名前を選択していなかった場合にも同様にエラーメッセージが表示される．

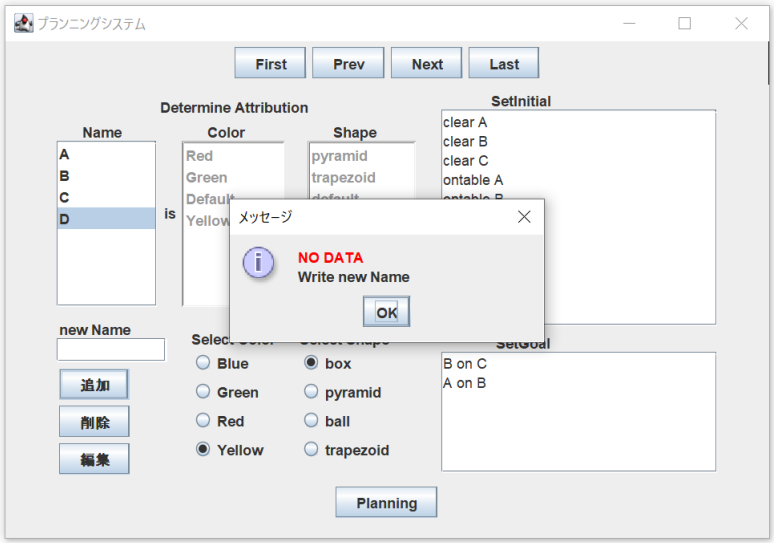


図 7: エラーメッセージの表示 1

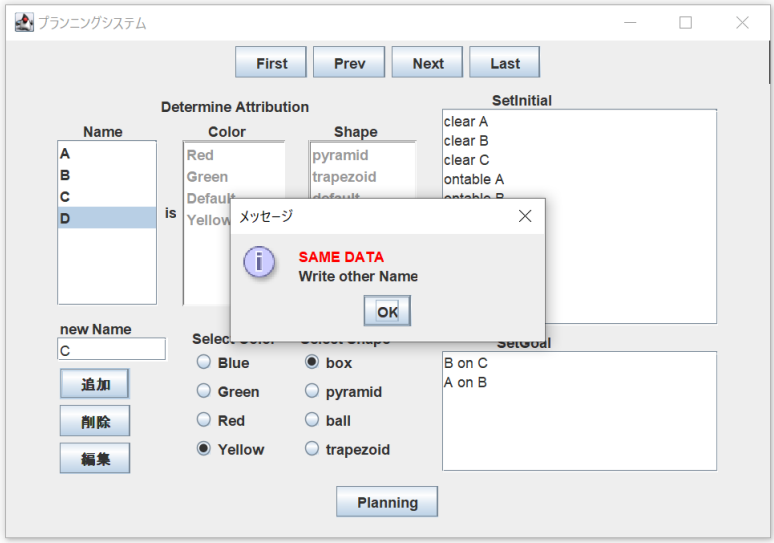


図 8: エラーメッセージの表示 2

初期状態と目標状態の変更は初期画面の右側のテキストボックスに記述する形で行う．このとき，範囲を超えた記述をする場合はスクロールバーが表示される (図 9)．また，属性情報変更時と同様に初期状態と目標状態に何も記入せず Planning ボタンを押すとエラーメッセージが表示される．

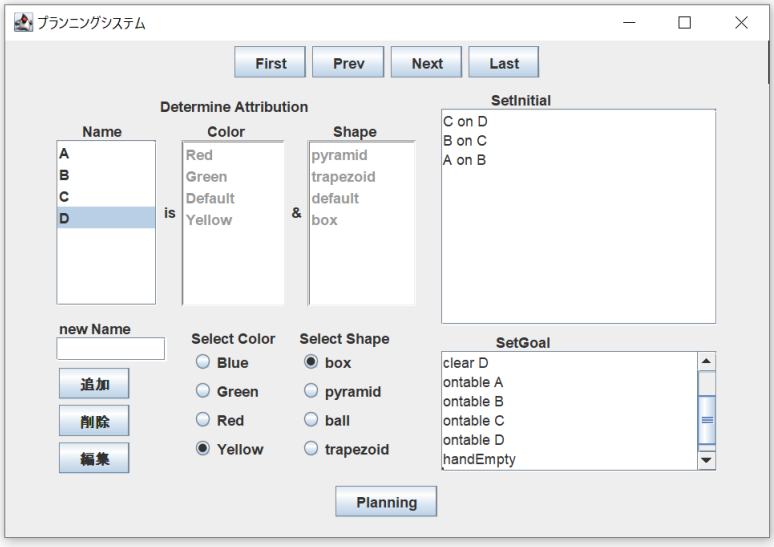


図 9: 初期状態と目標状態の変更

変更を行った状態で Planning ボタンを押すと，定義した状態を基にプランニングが行われる．すると，探索が完了したことを伝えるメッセージが表示され，ブロック操作の過程の画面が新しく得られた結果が表示される．今回は例としてブロックを 4 つに増やし，属性情報を変えた (図 10，図 11，図 12)．

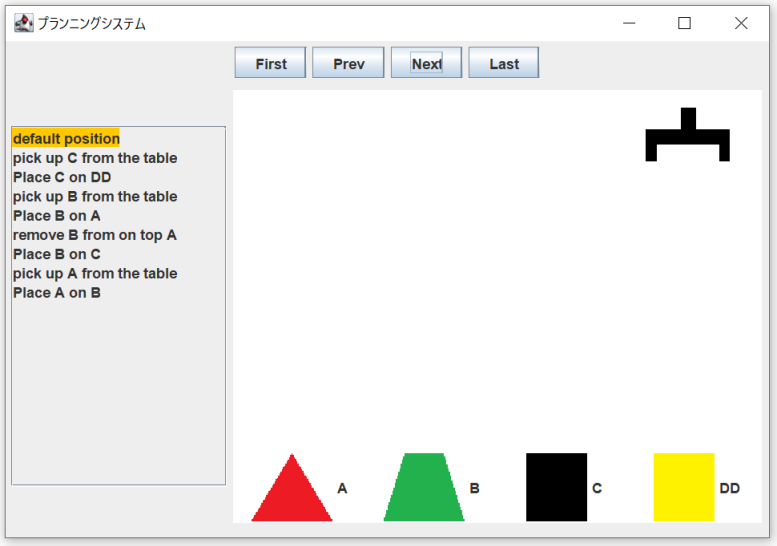


図 10: 再実行時のブロック操作の過程 1

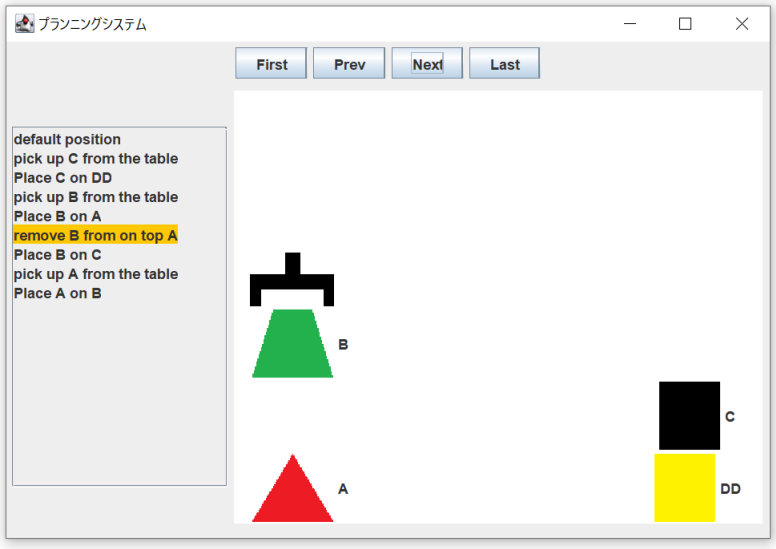


図 11: 再実行時のブロック操作の過程 2

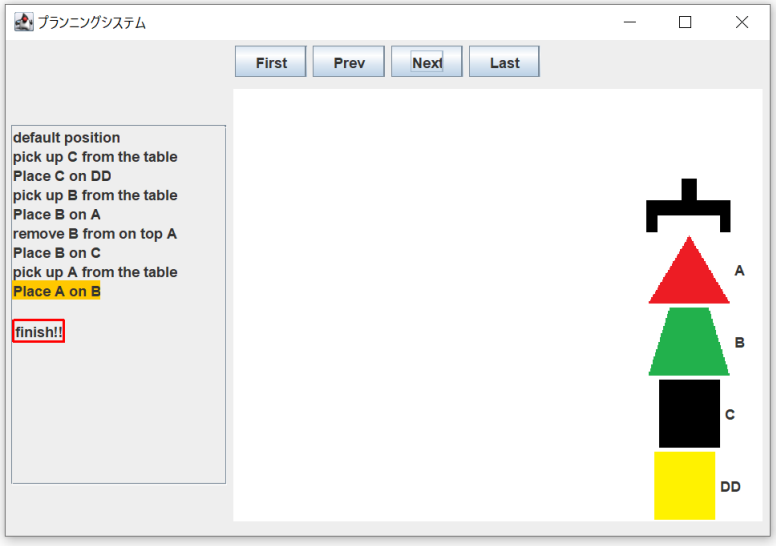


図 12: 再実行時のブロック操作の過程 3

またブロックを積み上げる動作だけでなく，ブロックをおろす場合もグラフィカルな可視化を行うことが出来る (図 13，図 14)．

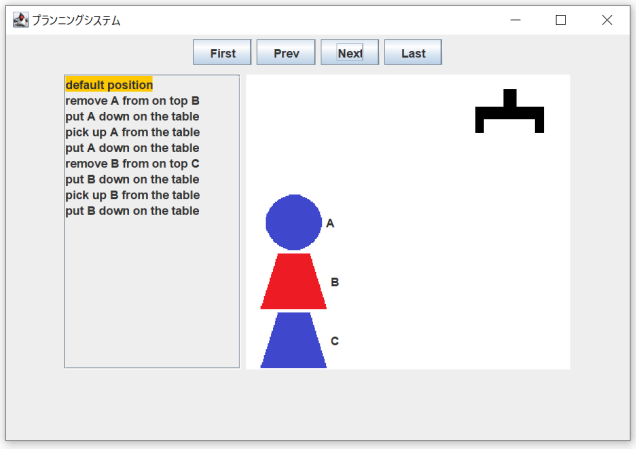


図 13: 積み下ろしのブロック操作の過程 1

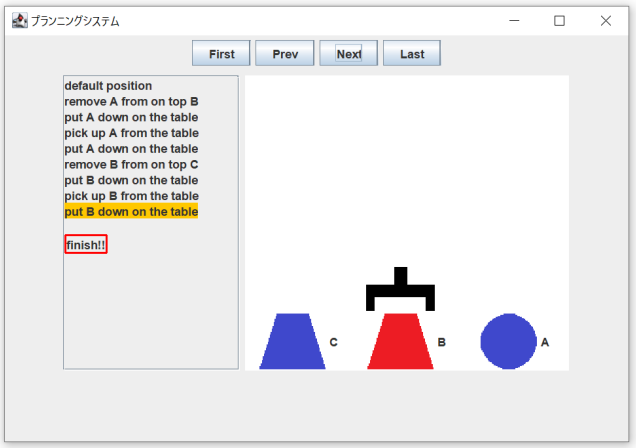


図 14: 積み下ろしのブロック操作の過程 2

次に，初期状態と目標状態が同一であった場合の動作を示す．初期状態を示す画面の後に以下の表示が現れ，既に目標状態にあることが示される (図 15)．

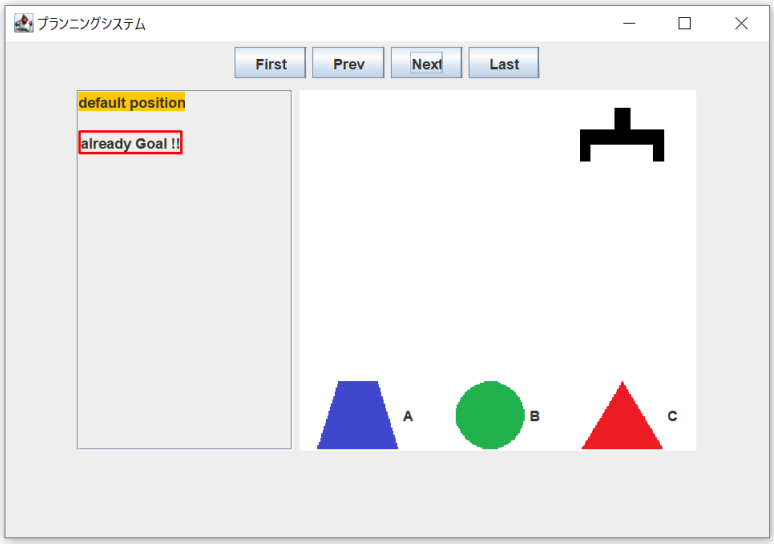


図 15: 初期状態と目標状態が同一の場合

さらに，禁止制約によってプランニングが行えなかった場合の動作を示す．Planning ボタンを押すと，プランニングが行えなかったことを示すエラーメッセージが表示され，ユーザーが定めた属性情報・禁止制約・ユーザーが定めた目標状態が表示される (図 16，図 17)．

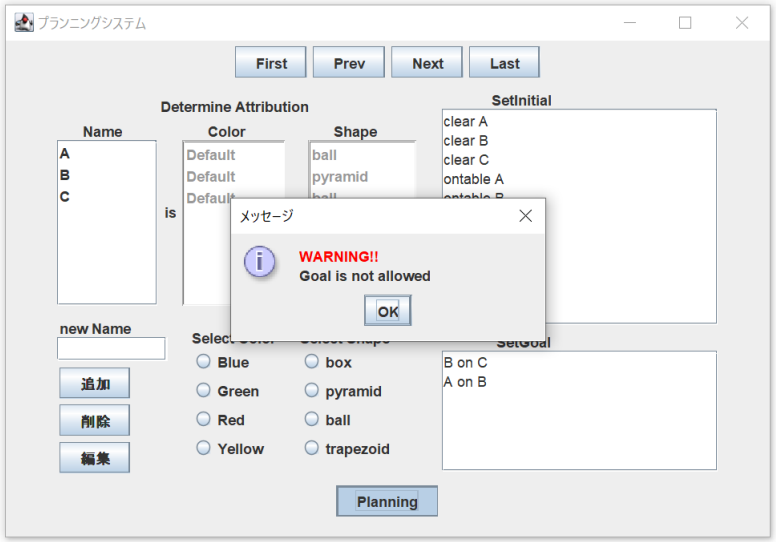


図 16: 禁止制約を示すエラーメッセージの表示

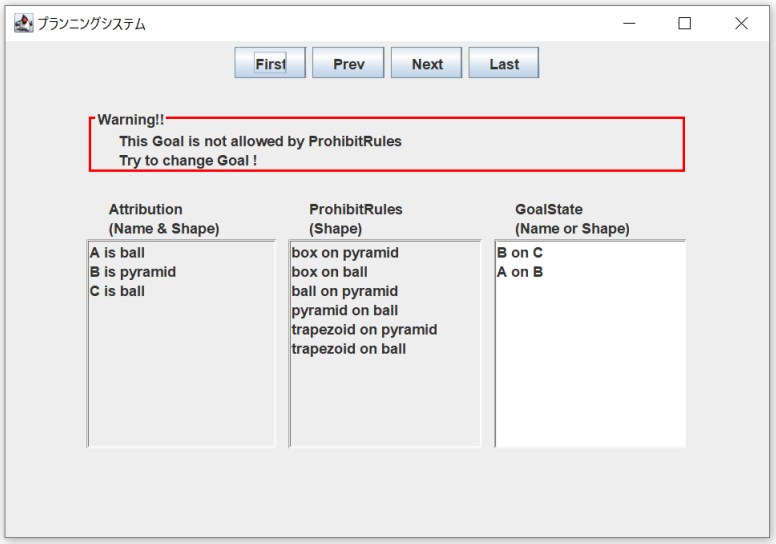


図 17: 属性情報・禁止制約・目標状態の表示

## 2.4 考察

今回は新しい知識について学ぶ必要があったため，まず初めにプログラムを二週間という期間の中でどのように実装するかの計画を出来るだけ細かく立てた．これによっていつまでに何を終わらせておけば良いかが明確になり，スケジュール調整がし易くなったため，初めに分かる範囲で詳細な計画を立てることが重要であると分かった．

今回は状態遷移をグラフィカルに可視化するため，どのようにそれを描画するかがまず大きな課題であった．初めは一枚のフレームに状態遷移を全て表示することを考えていたが，それでは状態遷移が何回行われるか分からない今回の課題では表示方法に問題がある．そこで異なる方法を検討した．GUI について調べる中で，CardLayout を用いればカードの枚数を増やしていくことで状態の遷移回数の変化にも対応できると分かり，この実装でプログラムを考えた．

次に描画をどのように行うかについて考えた．GUI 上での図形の描画は座標指定で行う必要がある．しかしこれではレイアウトマネージャーを無効にする必要がある．この場合描画を全て自身の座標管理で行わなければならないため，描画の難度が高いと考えた．そこで今回は図形を画像として予め作成しておき，これを ImageIcon として取り込み，BoxLayout で等分に分割した座標空間に対して割り当てていくことで実装を行うことを考えた．BoxLayout を用いることで，空間の行と列を指定した数に等分に分割し，それぞれに画像を割り当てることが出来る．この方法を用いることで，座標指定をすることなく図形を配置でき，比較的容易に状態遷移を描画出来ると考えた．

しかしこの方法を用いると，取り込んだ図に対して操作を行うことは出来ないことが分かった．そのため，画像の上に重なるように図形の名称 (A,B,C 等) を入れることや，取り込んだ画像の色を上書きすることが出来ないため，今回は図形の横に名称を記載し，画像は色違いのものをはじめから用意しておくことでこの問題を解決した．

予め図形を作成しておくことから，図形や色の任意指定も出来なくなってしまった．しかし座標指定を行う場合も図形の任意指定はできないと考えられるため，図形の任意指定を行えるようにするためには異なる方法を検討する必要があると感じた．

また，表示範囲は図形の大きさと個数から定めることで，ブロックの個数の変化に対応できるようにした．この方法を用いることで，表示する個数が変化しても描画内容に大きな変化が起きないようになった．しかし初めに設定したフレームの大きさの関係で，5 個以上のブロックの描画を行うと規定のフレームサイズを描画が超えてしまう．これにより，フレームをユーザー自身が大きくする必要が出てしまった．これについてはスクロールバーをつけることで対応しようと考えたが，実装が難しく実現できなかった．そのため描画は正しく行えたものの，表示方法に課題が残る結果となった．同時に，初めに作成するフレームのサイズも考えて決める必要があると分かった．

さらに属性情報の更新方法をどのようにするか考えた．属性情報では，名前・色・形を連動させ変更しなければならない．加えて任意での色や形の設定が出来ないため，これをラジオボタンで操作することとした．ここで項目を選択して編集を行えるとより使いやすくなると考え，JList を用いることを考えた．さらに名前・色・形をそれぞれ別の JList で表示することで，見やすい表示を心掛けた．

しかし名前・色・形をそれぞれ JList で管理すると，それぞれの項目を選択出来てしまうため，ユーザーが現在どの項目を編集しようとしているのかが分かりにくくなってしまう．これを解決するため色と形の JList は，プログラム側で編集を行う時以外は無効にすることで，ユーザーが操作できないようにすることを考えた．

また今回はユーザー視点の操作について考えた実装を心掛けた．

まず，ユーザーが任意で記述できる初期状態や目標状態・属性情報は記述スペースを超えた場合はスクロールバーが表示されるようにした．また出力される経路についても，表示画面よりも長くなってしまった場合はスクロールバーでの表示が出来るようにした．さらに状態遷移の描画を行う際の経路の表示では，自身がどの状態にあるのかを分かりやすくするため，現在の状態を表す一文に背景色としてオレンジ色を付けた．

経路が長くなってしまった場合はスクロールバーが表示されるものの，そのままでは常にスクロールバーが一番上にある状態でページが作成される．これでは表示範囲を超えた状態遷移のページでは，自身の現在の状態を確認するためにはスクロールバーを毎回下げる必要がある．これではユーザーにとって使い難い GUI だと考え，常に現在の状態が見える

位置に来るように表示する座標位置を順に下げ，ユーザーがスクロールバーを動かさずに現在の状態を確認できるよう実装を行った．

さらに禁止されている入力を行った場合などはメッセージを表示することで，何が問題でどのようにすれば良いのかを分かりやすく示せるようにした．

# 3 感想

GUI は初めて作成したため慣れない部分も多く大変だったが，今回の実装を通して理解を深められた．

今回のプログラムでは，実装期間の短さから学びながらの実装であったため，理解できたことから試してプログラムに加えていくという形になってしまった．そのため，プログラムが冗長的になってしまい，後から見返すと不必要になっている部分や重複している部分が多く見られた．さらに部分的に修正を行っていったため，一部分の機能が複雑になってしまった．これでは他の人が見た時に分かりやすいプログラムとは言えないため，メソッドに分けるなど機能ごとに細分化をする必要があったと考える．しかし，コメントアウトを多用することで，各機能についての内容は比較的分かりやすく記述することが出来たのではないかと感じた．

# 参考文献

[1] 新谷虎松『Java による知能プログラミング入門』コロナ社，2002 年．

[2] Let's プログラミング　Swing を使ってみよう，https://www.javadrive.jp/tutorial/ （2019 年 12 月 8 日アクセス）．