

# 知能プログラミング演習II 課題6

グループ8

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

2019年1月6日

提出物 rep6

グループ グループ8

氏名	担当課題
青山周平	発展課題5-7の改良
後藤拓也	必須課題5-2の改良
増田大輝	発展課題5-5の改良
湯浅範子	必須課題5-4の改良
小中祐希	必須課題5-1の改良

## 1 課題の説明

必須課題6-1 課題5にやり残した発展課題があれば参考にして拡張しても良いし、全く新しい独自仕様を考案しても構わない。自由に拡張するか、あるいはもし残っていた問題点があれば完成度を高めよ。

## 1.1 課題 5 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つたかどうか, 実行例を示して考察せよ. また, もしあつた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なる（”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.

## 2 必須課題 6-1(必須課題 5-1 の改良)

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.

課題 5 にやり残した発展課題があれば参考にして拡張しても良いし, 全く新しい独自仕様を考案しても構わない. 自由に拡張するか, あるいはもし残っていた問題点があれば完成度を高めよ.

前レポートにおいて課題 5-1 では, 目標状態の順番が不適切である場合に適切な順番に直すというメソッドを実装した. 具体的にいって、例えば AonB,BonC のような順番でリストに入っている場合 AonB が先に対象となってしまい, 積木の上の部分が先に完成してしまい残りの推論がうまくいかなくなるということが起こるため, リスト内の順番をより下になるブロックを含む目標状態が先に来るよう並べ替えるというものであった. 前レポートではこのメソッドを実装した上でプログラムを実行したのだが, 他の箇所との連携等の問題から実際には機能していないにもかかわらず実行結果だけがうまく出てしまっていたことがわかった. また, メソッド単体での検証をしていなかったのでメソッド自体の有用性も確認できていない事になる.

そこで, 本レポートではメソッドの検証及び不具合の解消を行うこととする.

### 2.1 検証

まず, 前回作成したメソッドを単体で実行できるように別途プログラムを作成した.

目標状態を (A on B)(B on C)(C on D) で実行したところ, 以下のような実行結果が得られた.

---

### ソースコード 1: 修正前プログラム

---

```
1 head[0]A
2 tail[0]B
3 head[1]B
4 tail[1]C
5 head[2]C
6 tail[2]D
7 initGoalList = [A on B, B on C, C on D]
8 goalList = [A on B, B on C, C on D]
```

---

リスト head,tail はそれぞれ目標状態の先頭と末尾の文字を格納したリストであり,initGoalList はメソッドへの引数（変更前）,goalList はメソッドの返り値（変更後）である。

のことから、メソッドがうまく機能していないことがわかった。

## 2.2 手法

ハンドトレースをした限りではうまく機能する見込みがあったので、ひとまずアルゴリズムはそのままにしていわゆるバグ探しの作業を行った。原因となっている箇所を見つけ出すために番号を出力する print 文をプログラムの各重要箇所に配置して再度実行をかけたところ、先頭の文字と末尾の文字が一致するかという分岐のところで先に進んでおらず、全て if 文をスルーしてしまった結果一切入れ替えが行われていなかった。

結論としては、分岐の判定で等価演算子を使ってしまっていたことが原因出会った。よって、if 文の条件式を tail[i].equals(head[j]) と変更した。

## 2.3 実装

以下に、検証用のパート含め修正したメソッド全体を示す。

---

### ソースコード 2: 修正後プログラム

---

```
1 public static ArrayList<String> sortGoalList(ArrayList<
   String> goalList) {
2     ArrayList<String> sortedGoalList = new
   ArrayList<String>();
3
4     //sortedGoalList に目標状態を格納
```

```

5         for(String s : goalList) {
6             sortedGoalList.add(s);
7         }
8
9         for(int k = 0; k < sortedGoalList.size();
10            k++){
11             String[] head = new String[
12                 sortedGoalList.size()];
13             String[] tail = new String[
14                 sortedGoalList.size()];
15
16             //各目標状態の先頭と末尾の文字を配
17             //列に格納
18             for(int i = 0; i < sortedGoalList.
19                 size(); i++){
20                 head[i] = sortedGoalList.
21                     get(i).substring(0,1);
22                 tail[i] = sortedGoalList.
23                     get(i).substring(
24                         sortedGoalList.get(i).
25                         length()-1);
26             }
27
28             //head と tail を出力
29             for(int i = 0; i < head.length; i
30                 ++){
31                 System.out.println("head["++
32                     i+"] "+head[i]);
33                 System.out.println("tail["++
34                     i+"] "+tail[i]);
35                 System.out.println();
36             }
37
38             int flag = 0;
39             //System.out.println(0);
40             for(int i = 0; i < sortedGoalList.
41                 size(); i++){
42                 //System.out.println(1);
43                 for(int j = i; j <
44                     sortedGoalList.size()-i

```

```

33 ; j++) {
34     //System.out.
35     //println(2);
36     if(tail[i].equals(
37         head[j])){
38         //System.out
39         .println
40         (3);
41         sortedGoalList
42         .add(j
43             +1,
44             sortedGoalList
45             .get(i
46             ));
47         sortedGoalList
48         .remove(
49             i);
50         //System.out
51         .println
52         ("i="+i
53         +" j="+j
54         +" list=
55         "+sortedGoalList
56         );
57         flag += 1;
58         break;
59     }
60 }
61 }
62 if(flag == 1){
63     break;
64 }
65 }
66 if(flag == 0){
67     break;
68 }
69 }
70 }
71 return sortedGoalList;
72 }

```

---

## 2.4 実行例

修正の結果、以下のような結果が得られた。

ソースコード 3: 実行結果

---

```
1 head[0]A
2 tail[0]B
3
4 head[1]B
5 tail[1]C
6
7 head[2]C
8 tail[2]D
9
10 head[0]B
11 tail[0]C
12
13 head[1]A
14 tail[1]B
15
16 head[2]C
17 tail[2]D
18
19 head[0]A
20 tail[0]B
21
22 head[1]C
23 tail[1]D
24
25 head[2]B
26 tail[2]C
27
28 initGoalList = [A on B, B on C, C on D]
29 goalList = [C on D, B on C, A on B]
```

---

## 2.5 考察

ここでは、今回のようにメソッドが機能していないにもかかわらず、機能しているように見えてしまったことについて考察する。前回にレポートで作成した sortGoalList メソッドは、目標状態の順番により推論機能がうまく

く機能しないという現象を解消するために実装したものである。したがって、目標状態はどんな状態でも機能することを目標としており、目標状態は元々用意されていた initGoalList から取得するようにしてあった。しかしながら、本課題において私の班は GUI を実装する手段としてプレゼンターを用意しており、プレゼンターは直接 Planner.java のメソッドにアクセスする形になっている。その結果、他の課題を実装する過程で目標状態に関して直接 initGoalList から目標状態を取得するようになっており、作成した sortGoalList メソッドが機能していない状態になってしまっていたと考えられる。

### 3 必須課題 6-1(必須課題 5-2 の改良)

課題 5 にやり残した発展課題があれば参考にして拡張しても良いし、全く新しい独自仕様を考案しても構わない。自由に拡張するか、あるいはもし残っていた問題点があれば完成度を高めよ。

#### 3.1 前回の内容

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。

前回の課題 5-2 では、「Place A on A(存在しない制約)」や「Place B on A(属性を考慮した場合の禁止制約)」を改良させるために、属性クラスである Attributions クラスで定義した禁止制約の XonY 関係を HashMap に格納し、その条件と現状態の X' on Y' 関係を照らし合わせるという方法を用いた。オペレータ選択にはランダム選択を用いることで、どんなに適当な状態になっても禁止制約が発動されることを保証するようにした。

実行結果は以下のように失敗していた。

---

ソースコード 4: 失敗実行例その 1

```

1 *** GOALS ***[holding B]
2 **holding B
3 そのオペレータは実行できません
4 おすすめのオペレータを使います
5 **clear ?y3
6 [clear A, clear B, clear C, ontable A, ontable B, ontable
C, ontable pyramid, handEmpty]
7 *** GOALS ***[holding B]
8 **holding B
9 そのオペレータは実行できません
10 おすすめのオペレータを使います
11 **clear ?y3
12 [clear A, clear B, clear C, ontable A, ontable B, ontable
C, ontable pyramid, handEmpty]
13 *** GOALS ***[holding B]
14 **holding B
15 そのオペレータは実行できません
16 おすすめのオペレータを使います
17 **clear ?y3
18 ... (以下無限に続く)

```

---

問題点としては、離れた制約がうまく機能していなかったことである。「B on ?y1」という目標に対して、オペレータ「Place ?x2 on ?y2」によって、「?x2 = B」と「?y2 = ?y1」となるのはよいが、その後、「Clear A」と「Clear ?y2」のマッチングを行う際に、「?y2 = ?y1 = A」となるが、この処理において、禁止制約(X on Y関係)との関係がうまく取れていなかったのである。以下の図1を参照してほしい。

初期設定から与えられる禁止制約:prohibitと、現在の状態を表す:productKeyOnValueを照らし合わせて、積み方に制約をかけている。productKeyOnValueには、「X on Y関係」の目標が生じたときに、現在問題にしている「X on Y関係」の何がXで何がYかを保存する。これは、上記で述べたような離れた制約(B on ?y1の後に,Clear ?y2など)にも対応できるようになるためである。

これを実装するにあたり、replaceBufferのタイミングも変更した。以前のプログラムの内容では、「?x1 = A」と決まった後で、「?x2 = A」と制約を加える形をとっていた。それでは、?x2 = ?x1の対応が取れない。(実装で内容を詳しく説明)

また、木構造を用いてこれらの再帰構造の内容を表現すると図2のよう

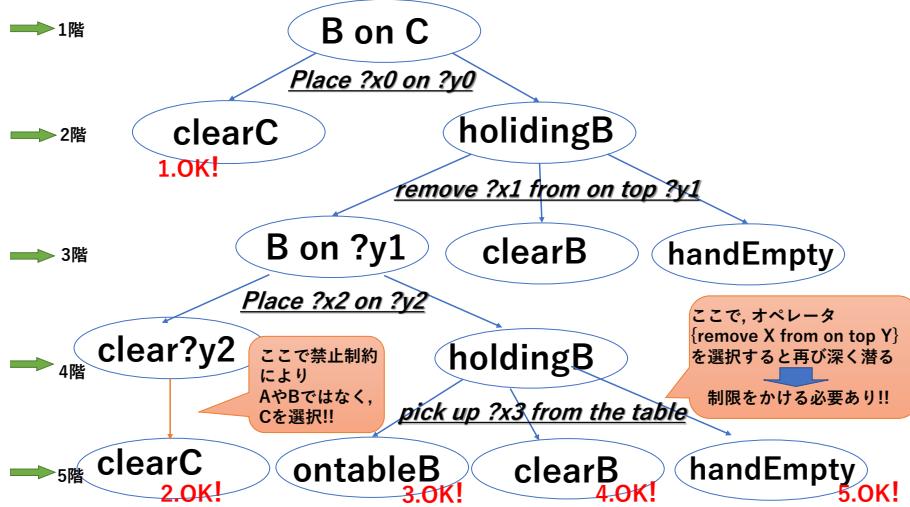


図 1: plannnig の再帰呼び出しの木構造表現

になる。図 2 は、目標「B on C, A on B」における「B on A」を単体目標としたときの再帰構造の様子である。1 階の「B on C」に対して、オペレータ「Place ?x on ?y」の add 部がマッチングし、その if 部が次の目標となり、1 つ深くなり、2 階目へ移行する。その後同様の処理を深さ優先探索として行われる。

ここでポイントになるのは、「X on Y 関係」において、1 階と 2 階の間で決定される「?x0 on ?y0 関係」に対して、3 階で定義される「B on ?y1」関係においては、4 階で決定されるのではなく、5 階で決定される。このように 3 階で定義されたものが次の階ではなく、5 階で定義される状態を”離れた制約”と呼んでいる。前回のプログラムではこの点に関してうまくいっていなかった。今回はこの点を修正した。実装方法は前回の内容を利用し、以下のように変更を加えた。

### 3.2 実装

Attribuite クラスで定義された禁止制約 (A on A など) を HashMap に格納し、それを Planner クラスで呼び、Unifier クラスとその情報を共有する。そして現在の「X on Y 関係」を保存する HashMap:productKeyOnValue を用いて禁止制約との照らし合わせをする。以下には前回の実装では失敗していた Unifier クラスでの”禁止制約との照らし合わせによるマッチ

ング”を示す。

ソースコード 5: Unifier クラスの varMatching メソッド

```
1 boolean varMatching(String vartoken, String token) {
2     System.out.println("vars="+vars);
3     System.out.println("vartoken="+vartoken);
4     System.out.println("token="+token);
5     if(vars.containsKey(vartoken)) {
6         if(token.equals(vars.get(vartoken))) {
7             return true;
8         }else{
9             return false;
10    }
11 }else{
12     /* ココじゃなくて...
13     replaceBuffer(vartoken, token);
14     if (vars.containsValue(vartoken)) {
15         replaceBindings(vartoken, token);
16     }
17 */
18     if(productKeyOnValue != null) {
19         //product_KeyValue の数
20         for(String str1 : productKeyOnValue.keySet()){
21             System.out.println("vars="+vars);
22             if(var(productKeyOnValue.get(str1))) {
23                 for(int num=0; num<prohibit.get(str1).size
24                     (); num++) {
25                     if(vars.containsKey(productKeyOnValue.
26                         get(str1))) {
27                         System.out.println(productKeyOnValue.
28                             get(str1));
29                         if(vars.get(productKeyOnValue.get(str1
30                             )).equals(vartoken) & prohibit.get
31                             (str1).get(num).equals(token)){
32                             System.out.println("禁止制約発動
33                             しました.");
34                             return false;
35                         }
36                     }
37                 }
38             }
39         }
40     }
41 }
```

```

34    }
35    //ココで行います!
36    replaceBuffer(vartoken, token);
37    if (vars.containsValue(vartoken)) {
38        replaceBindings(vartoken, token);
39    }
40    vars.put(vartoken, token);
41 }
42 return true;
43 }

```

上記のプログラムでは自分でも何をやっているのかわからなくなるので以下の図 2, 図 3 を参考にしながらプログラムを見ていただきたい。

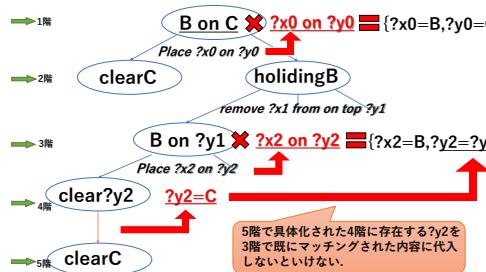


図 2: 階層的な視点

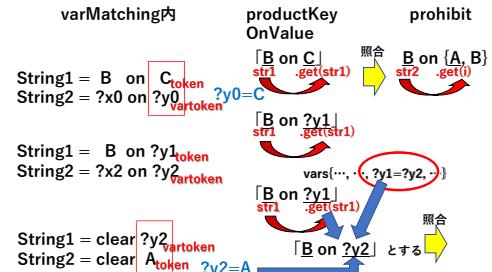


図 3: varMatching 内の処理内容

図 2, 3 ともに、「X on Y 関係」だけに焦点を絞っている。図 2 からは、「?x0 on ?y0」の場合に 2 階から 1 階へのマッチングに対し、「?x2 on ?y2」の場合は、5 階の「clearC」の C を使って、3 階のマッチングに利用されていることを示している。図 3 は、Unifier クラスの VarMatching メソッド内でのマッチングする 2 つの文、現状態 XonY 関係を示す HashMap:productKeyOnValue、禁止制約の HashMap:prohibit が示されている。

図 2 の各階のノードが決定された時点で、図 3 の productKeyOnValue にはそのノードの値が代入され、その後、選択されたオペレータの add 部 (X on Y に絞って言及中) と Matching をする。productKeyOnValue も prohibit も、HashMap なので、「X on Y」が「Key on Value」の関係で呼び出すことができる。1,2 階間でのマッチング処理は、varMatching メソッドでマッチングした?x0, ?y0 をそのまま prohibit の禁止制約と照らし合わせればよいが、3~5 階間でのマッチング処理は、図 2 のノードはあくま

で、「B on ?y1」なので, ?y2との関係は「B on ?y1」と「?x2 on ?y2」のマッチングが終わった後に確定される。その関係はハッシュマップ:varsに格納されるので,それを用いて,ノードの「B on ?y1」を「B on ?y2」とする。そして,4~5階で?y2=Cとマッチングすると,その情報を現状態のノード:productKeyOnValueに代入することで,「B on A」という関係を生成させる。それと禁止制約を照合することで,多階層のX on Y関係でも対応できるようにした。プログラムのfor文では,productKeyOnValueのKey値(str1)とVlaue値(.get(str1))を見て,そのKey(str1)に対応する禁止制約のprohibitのVlaue値(.get(str2))をもってきたのち,ifの条件文で,HashMap:varsの?y1=?y2関係と禁止制約:prohibitのvalue値を1つずつ照らし合わせる処理をしている。

そして以上のvarsの関係[?y1=?y2]の関係を見た後で,?y2=Cとし,?y1=Cとするところから,replaceBufferメソッドの呼び出しは,上記の内容を行った後で実行するようにした。先にreplaceBufferメソッド呼び出してしまうと,Key:?y1のValue値は?y2となってしまい,具体化されないまま終わってしまうからである。

### 3.3 実行例

今回の実行においては,上記の説明を実際に表示させるために,オペレータの選択を自分で選択し,逐一状況を確認していく。

ソースコード 6: 3~5階層のマッチング

---

```

1 Place ?x2 on ?y2
2 addList = [?x2 on ?y2, clear ?x2, handEmpty]
3 string1=B on ?y1
4 string2=?x2 on ?y2
5 productKeyOnValue 保存 = {B=?y1}
6 vars={?y0=C, ?x0=B, ?x1=B}
7 vartoken=?x2
8 token=B
9 productKeyOnValue.get()=?y1
10 vars={?y0=C, ?x0=B, ?x1=B}
11 vars={?y0=C, ?x0=B, ?x1=B, ?x2=B}
12 vartoken=?y1
13 token=?y2
14 productKeyOnValue.get()=?y1

```

```

15 vars={?y0=C, ?x0=B, ?x1=B, ?x2=B}
16 その 2
17 Key = B Value = ?y1
18 unify 成功
19 オペレータの具体化:Place B on ?y2
20 *** GOALS ***[clear ?y2, holding B]
21 **clear ?y2
22 string1=clear ?y2
23 string2=clear A
24 vars={?y0=C, ?x0=B, ?y1=?y2, ?x1=B, ?x2=B}
25 vartoken=?y2
26 token=A
27 productKeyOnValue.get()=?y1
28 vars={?y0=C, ?x0=B, ?y1=?y2, ?x1=B, ?x2=B}
29 禁止制約発動しました.
30 string1=clear ?y2
31 string2=clear B
32 vars={?y0=C, ?x0=B, ?y1=?y2, ?x1=B, ?x2=B}
33 vartoken=?y2
34 token=B
35 productKeyOnValue.get()=?y1
36 vars={?y0=C, ?x0=B, ?y1=?y2, ?x1=B, ?x2=B}
37 禁止制約発動しました.
38 string1=clear ?y2
39 string2=clear C
40 vars={?y0=C, ?x0=B, ?y1=?y2, ?x1=B, ?x2=B}
41 vartoken=?y2
42 token=C
43 productKeyOnValue.get()=?y1
44 vars={?y0=C, ?x0=B, ?y1=?y2, ?x1=B, ?x2=B}
45 theBinding{?y0=C, ?x0=B, ?y1=C, ?x1=B, ?y2=C, ?x2=B}

```

---

「?x2 on ?y2」関係から「clear C」に至るまでの処理を図2,3を参考にしながら見ていただけだと、よくわかると思う。「B on A」という禁止制約が発動され、A, Bではマッチングが行われず、Cのみで成功している。実際に、最後の Binding 情報では、?y1=C,...,?y2=Cと正しく行われていることが分かる。

### 3.4 考察

図1の木構造からも分かるように、問題に上がっていた繰り返しの処理(Place A on B, remove A from on top B の繰り返し)をなくすには、「pick up B from the table」を選択するしかないということが分かる。

ただ、以下の実行結果を見てもらいたい。最終結果のみを記す。

---

#### ソースコード 7: A on C が生じてしまう

---

```
1 ***** This is a plan! *****
2 pick up B from the table
3 Place B on C
4 remove B from on top C
5 Place B on C
6 pick up A from the table
7 Place A on C
8 remove A from on top C
9 Place A on B
```

---

確かに、「A on C」という禁止制約は存在しないが、既に「B on C」が存在している状況で、「A on C」とすることは…物理的に不可能である。そのため、実行中にも、禁止制約を随時追加していかなければならないことが分かる。ただ、「B on C」が確定したのちに「A on C」はダメという禁止制約、「remove B from on top C」を実行された後には、「A on C」はダメではないので禁止制約から取り除くといった処理を行うのは、また…大変であり、planningの実装に向けては、かなりゲームフィールの制限を強くするしかないことが分かった。”知識システム”的の授業で、新谷先生が「条件を限定した仮想空間を用いないとプログラムに書ききらないよ」と言っていた内容が良く分かった。

## 4 必須課題 5-2 のさらなる改良

### 4.1 序論

上記では、禁止制約を「X on Y 関係」のHashMapを用いることで解決するアプローチを試みた。その際に、図1で紹介したが、「AをとてBの上に置く」、「AをBから取り除く」、「AをとてBの上に置く」、「AをBから取り除く」といった無駄な処理をなくす方法として、オペレータ

の選択に制限をかける必要があると述べた。今回はそのプログラムを実装してみる。

## 4.2 手法

図1でもわかるように、2階で「holdingB」という目標が設定されていて、その後、4階でも同じ「holdingB」という目標が出てくる。4階で2階と同じオペレータ（ここでは、remove X from on top Y）を選択したらどんどんと木は深くなる。そのため、planningAGoal メソッドで設定された1つの目標（木構造におけるノード）と、そのノードに対して選択されたオペレータを保存しておき、繰り返しを防ぐという方法をとる。

## 4.3 実装

Planner クラスに、1つの目標（木構造におけるノード）とその時に選択されたオペレータの対応関係を保存する HashMap:goalOp を作成。Key には目標、Value にはオペレータのインデント [0~3] を保存する。（オペレータの選択とインデントの関係に関しては、課題5のレポートを参照してもらいたい。）

---

### ソースコード 8: 目標とオペレータの対応関係の保存

---

```
1 //Planner クラス
2   HashMap<String, Integer> goalOp; //Key: 単目標/Value: オ
      ペレータの番号
3 //コンストラクタ
4   goalOp = new HashMap<String, Integer>();
5
6 //planningAGoal メソッド内
7   //1.state リストとの照らし合わせ処理(省略)
8   //2. ノードをKeyとして保存(Value)は未定値
9   if(!goalOp.containsKey(theGoal)) {
10     goalOp.put(theGoal, -1);
11   }
12  //3.operator の選択(ランダム)
13  int randInt = Math.abs(rand.nextInt())\%operators.size
14  ();
15  Operator op = (Operator)operators.get(randInt);
16  cPoint = randInt;
```

```

16 //4-1.選択したオペレータの実行(一部省略)
17 Operator anOperator = rename((Operator) operators.get(
18     cPoint));
19 for (int j = 0; j < addList.size(); j++) {
20     Unifier unification = new Unifier();
21     if(unification.unify(theGoal, (String) addList.get(j
22         ), theBinding, attributions.keyValueProhibit,
23         p_productKeyOnValue)) {
24         //5.一度は選択されていたら,break
25         if(goalOp.get(theGoal) == cPoint) {
26             System.out.println("breakします");
27             break;
28         }
29         //6.breakしなかったら, 保存する
30         goalOp.put(theGoal, cPoint);
31         System.out.println("goalOp"+goalOp);
32         ...
33     //4-2.選択されたオペレータが実行されない場合,
34     for (int i = 0; i < operators.size(); i++) {
35         if(i != cPoint) {
36             anOperator = rename((Operator) operators.get(i));
37             addList = (ArrayList<String>) anOperator.getAddList
38                 ();
39             for (int j = 0; j < addList.size(); j++) {
40                 Unifier unification = new Unifier();
41                 if(unification.unify(theGoal, (String) addList.get
42                     (j), theBinding, attributions.keyValueProhibit
43                     , p_productKeyOnValue)) {
44                     //5.既にその目標に対してオペレータが選択されていたら,
45                     if(goalOp.get(theGoal) == i) {
46                         System.out.println("breakします");
47                         break;
48                     }
49                     //6.保存する
50                     goalOp.put(theGoal, i);
51                     System.out.println("goalOp"+goalOp);
52                     ...

```

---

選択したオペレータが実際に実行されるか、それとも別のオペレータが実行されるかに関しては、課題5のレポートを参考にしてもらいたい。

planningAGoal メソッドが実行されたら、まず、HashMap:goalOp にその目標が Key として設定されていない場合は、Value を-1 しておく。そ

れ以前にすでにその目標が Key として保存されていたら, -1 の初期化は行わない。その後, 実際に, オペレータの add 部と unify をしていき, 成功したら, そのオペレータのインデントを goalOp の Value に加える(上書き保存する)。すでにその目標に対して, 以前実行されたオペレータがあり, 保存されていたら(つまり-1ではなかったら), 別のオペレータを探すために for 文を break し, 下の”選択されたオペレータ以外のオペレータの実行”へ移動する。そこで, 一度も選択されていないオペレータだったら, そのオペレータを実行する。

#### 4.4 実行結果

---

##### ソースコード 9: goalOp によって制約される

---

```

1 [B on C, A on B]
2 *** GOALS ***[B on C, A on B]
3 **B on C
4 Place ?x0 on ?y0
5 goalOp{B on C=0}
6 *** GOALS ***[clear C, holding B]
7 **clear C
8 theBinding{?y0=C, ?x0=B}
9 *** GOALS ***[holding B]
10 **holding B
11 remove ?x1 from on top ?y1
12 goalOp{holding B=1, B on C=0}
13 オペレータの具体化:remove B from on top ?y1
14 *** GOALS ***[B on ?y1, clear B, handEmpty]
15 **B on ?y1
16 Place ?x2 on ?y2
17 goalOp{holding B=1, B on C=0, B on ?y1=0}
18 オペレータの具体化:Place B on ?y2
19 *** GOALS ***[clear ?y2, holding B]
20 **clear ?y2
21 theBinding{?y0=C, ?x0=B, ?y1=C, ?x1=B, ?y2=C, ?x2=B}
22 *** GOALS ***[holding B]
23 **holding B
24 Place ?x3 on ?y3
25 そのオペレータは実行できません
26 remove ?x4 from on top ?y4
27 break します

```

```

28 pick up ?x5 from the table
29 オペレータの具体化:pick up B from the table
30 *** GOALS ***[ontable B, clear B, handEmpty]
31 **ontable B
32 theBinding{?y0=C, ?x0=B, ?y1=C, ?x1=B, ?y2=C, ?x2=B, ?x4
    =B, ?x5=B}
33 *** GOALS ***[clear B, handEmpty]
34 **clear B
35 theBinding{?y0=C, ?x0=B, ?y1=C, ?x1=B, ?y2=C, ?x2=B, ?x4
    =B, ?x5=B}
36 *** GOALS ***[handEmpty]
37 **handEmpty
38 theBinding{?y0=C, ?x0=B, ?y1=C, ?x1=B, ?y2=C, ?x2=B, ?x4
    =B, ?x5=B}
39 Success !
40 Success !
41 Success !
42 ...

```

---

「holdingB」が2回目の目標として設定された際に、「Place ?x3 on ?y3」のオペレータを選択しているが、add文には「holdingX」が存在しないので、次のオペレータを探す。すると、「remove ?x4 from on top ?y4」が選択されるが、goalOpにすでに holding B=1と保存されているので、インデント1のremoveオペレータは実行されない。その後、「pick up ?x5 from the table」が実行され、成功していることが分かる。

## 4.5 考察

このプログラムにより、オペレータの選択をランダムにしても、無駄な処理(AをBの上に置いて、AをBから取り除いて、またAをBの上において...など)はなくなった。完璧なプログラムかといえば、結局、「Place A on C」という前に述べた”物理的に不可能な置き方”が残ってしまったので、残念である。また、”机に置いてある3つの積み木(A, B, C)を順に上に積んでいく”という処理しかできなくなり、汎用性がなくなってしまった。本当は積み木が4つになって、初期状態がどうであれ、最終的な目標状態がどうであれ、適切なオペレータ選択ができればよかったです、なかなか難しい…

## 5 必須課題 6-1(必須課題 5-4 の改良)

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。

課題 5 にやり残した発展課題があれば参考にして拡張しても良いし、全く新しい独自仕様を考案しても構わない。自由に拡張するか、あるいはもし残っていた問題点があれば完成度を高めよ。

私の担当箇所は、前回課題で作成した GUI の改良である。

### 5.1 手法

前回の GUI に以下のような機能を加えた。

- ウィンドウサイズの変更時に起きたボタンのバグの解消
- 動作を動的に表示

今回の改良では、前回の課題で時間の都合などで出来なかった部分の機能の拡張を行った。

### 5.2 実装

作成した GUI プログラムの中の改良部分についての実装を以下に示す。

まず、プランニングボタンをクリックして経路導出を行ったときの再描画時の動作をソースコード 10 に示す。

ソースコード 10: 再描画時の動作

```
1 presenter.restart();
2 pUR = presenter.getStepList();
3 result = presenter.getPlan();
4 if (result != null) {
5     results = new ArrayList<>(result);
```

```

6         results.add(0, "default position");
7 } else {
8     // 禁止制約に引っかかった時
9     results = new ArrayList<>();
10    results.add("default position");
11 }
12 page1.removeAll();
13 cardPanel.removeAll();
14 btnPanel.removeAll();
15 for (JPanel c : card) {
16     c.removeAll();
17 }
18 createResultPage(pUR);
19 createButton();
20 finishData();

```

---

Presenter クラスの restart メソッドを用い、変更した状態でプログラムを動作させる。得られた結果に応じて描画内容を変更するが、このとき removeAll メソッドを用いることで、現在あった描画内容を一度全て削除することが出来る。これを行わずとも再描画を行うことは可能であるが、その場合はマウスの動作に応じて出力が二重になって現れるなど誤作動が起きる場合がある。そのため、一度記述内容を削除してから再描画を行うことで誤作動を無くした。

次に、出力結果を動的に表示するためのプログラムをソースコード 11 に示す。

#### ソースコード 11: 動的表示用

---

```

1 // 変数定義
2 Timer timer;
3 int time;
4 ...
5 // タイマーの設定
6 timer = new Timer(1100 , this);
7 timer.setActionCommand("timer");
8
9 // 動的操作開始クラス (マウスクリックに対応)
10 public class myListner extends MouseAdapter{
11     public void mousePressed(MouseEvent e){
12         layout.show(cardPanel, "label0");

```

```

13         time = 0;
14         timer.start();
15         firstButton.setEnabled(false);
16         prevButton.setEnabled(false);
17         nextButton.setEnabled(false);
18         lastButton.setEnabled(false);
19     }
20 }
21
22 // タイマーの動作設定
23 public void actionPerformed(ActionEvent e){
24     // ボタン選択アクション
25     String cmd = e.getActionCommand();
26     ...
27     else if (cmd.equals("timer")){
28         if (time < cardPage) {
29             layout.next(cardPanel);
30             time++;
31         } else if (time == cardPage) {
32             layout.next(cardPanel);
33             firstButton.setEnabled(true);
34             prevButton.setEnabled(true);
35             nextButton.setEnabled(true);
36             lastButton.setEnabled(true);
37             timer.stop();
38             time = 0;
39         }
40     }
41 }
```

---

前回の課題では、導出結果の結果を順に確認するためにはユーザーが毎回 Next ボタンをクリックする必要があった。この方法でも結果の確認を行えるが、一連の動作を流れで確認したい時には手間がかかる。そのため今回の改良では Move ボタンを新たに作成し、ユーザーが Move ボタンを一度クリックしただけで導出結果を順に全て確認できるようにした。

改良にあたり、タイマークラスを利用した。Move ボタンをクリックするとタイマーが動作を開始し、一定秒数ごとに経路を順に表示する。この動作中に他のボタンをクリックすると正しい動作が得られなくなってしまうため、経路の動的表示を行っている間は他のボタンを無効にする

ことで対処した。

### 5.3 実行例

初めに、ウィンドウサイズの変更時に起きたボタンのバグの解消結果を示す。

まず、ウィンドウサイズを変更した際に起きるバグは以下の画像から分かる(図4)。これはウィンドウ作成時には起きず、再描画後に表示されるウィンドウサイズをユーザーが変更した際に起きる。

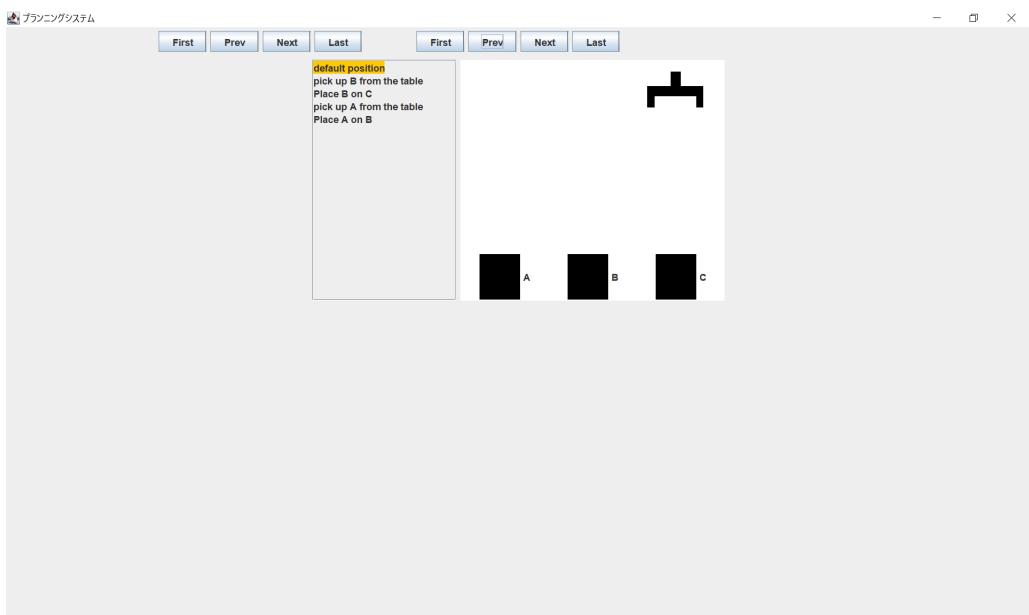


図4: ウィンドウサイズを変更した際に起きるバグ

画像からも分かるように上に表示されるボタンの数が二重に表示されている。このときどちらのボタンも正しく動作を行うが、ウィンドウサイズの大きさによってはボタンが重なって表示されてしまうこともある。

これを改良した後の状態が、以下のようになる(図5).

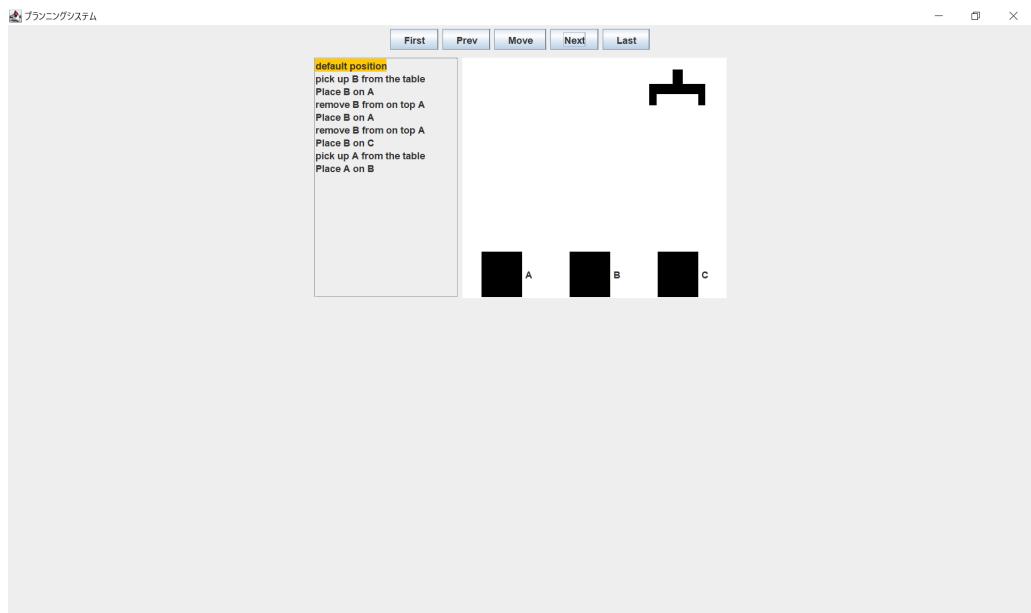


図5: 改良後のウィンドウ

ここからも分かるように、再描画後ウィンドウサイズを変更してもボタンが二重に表示されることが無くなった。

次に、導出結果をより見やすくするための Move ボタンの動作の実行結果を以下に示す。

実行した直後に現れる画面が以下のようになる(図 6)。

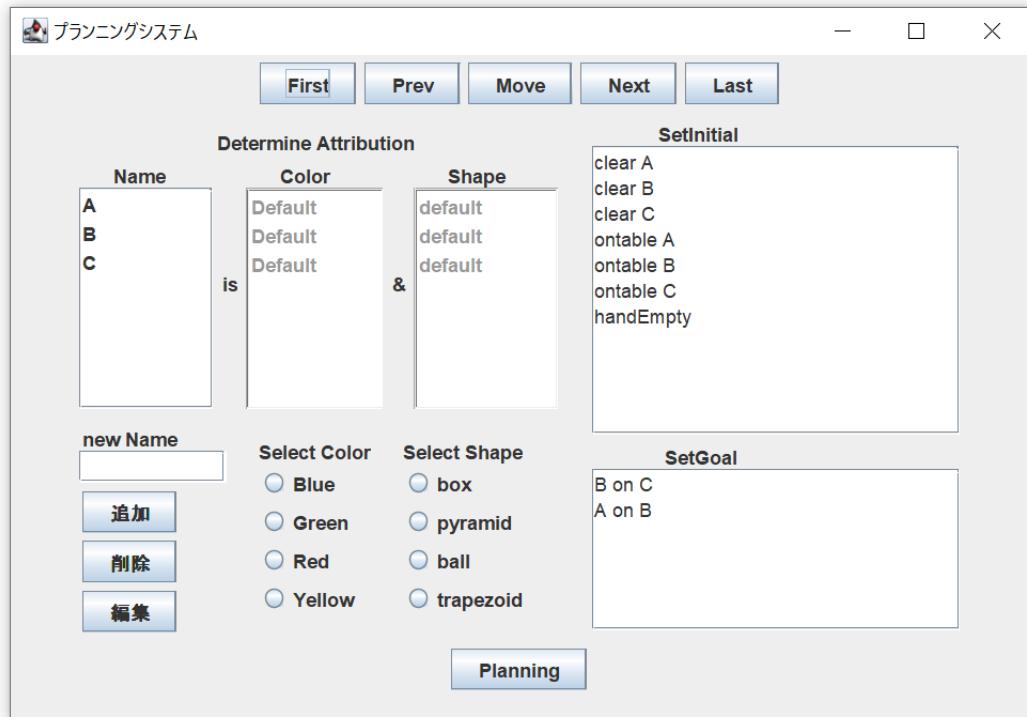


図 6: Move ボタン追加後の初期画面

前回の画面では、ウィンドウ上部にあるボタンが「First」「Prev」「Next」「Last」の四つであったが、今回の改良により新たに「Move」ボタンが追加されている。

Next ボタンを押すと、前回の課題同様に順に導出経路を確認することができる(図7)。

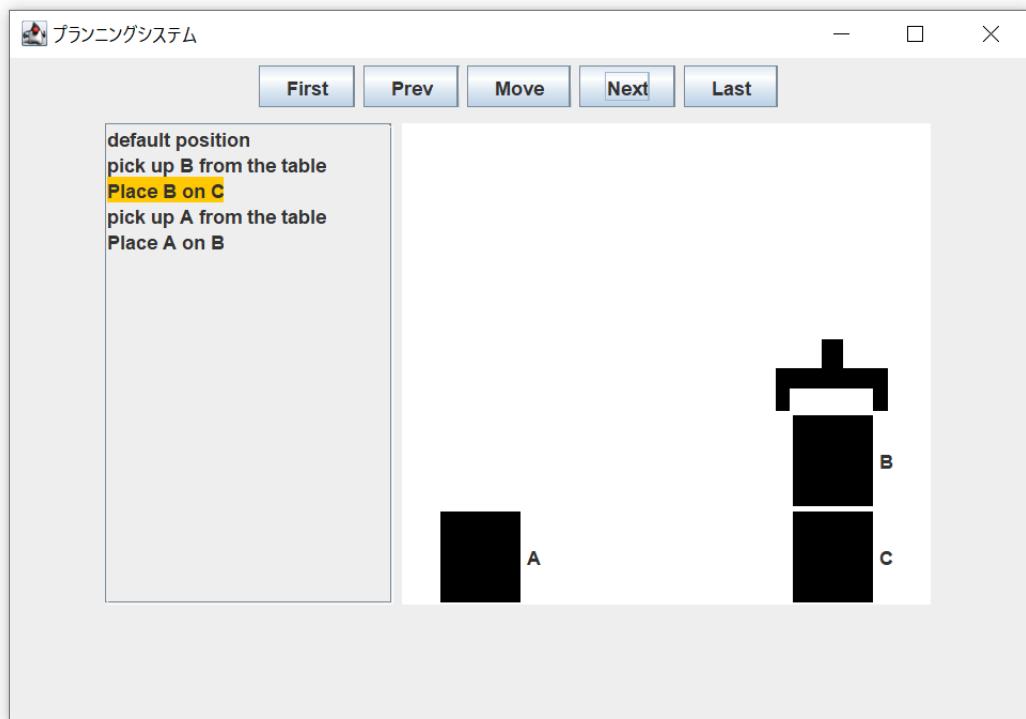


図 7: Next ボタンで経路を確認するとき

このとき、Next ボタンで状態が一つ進み、Prev ボタンで状態が一つ戻る。また First ボタンで初期画面に戻り、Last ボタンで最後の画面に遷移する。

これらの動作は前回作成した GUI と同じである。

Move ボタンを押した時の動作は以下のようになる(図8).

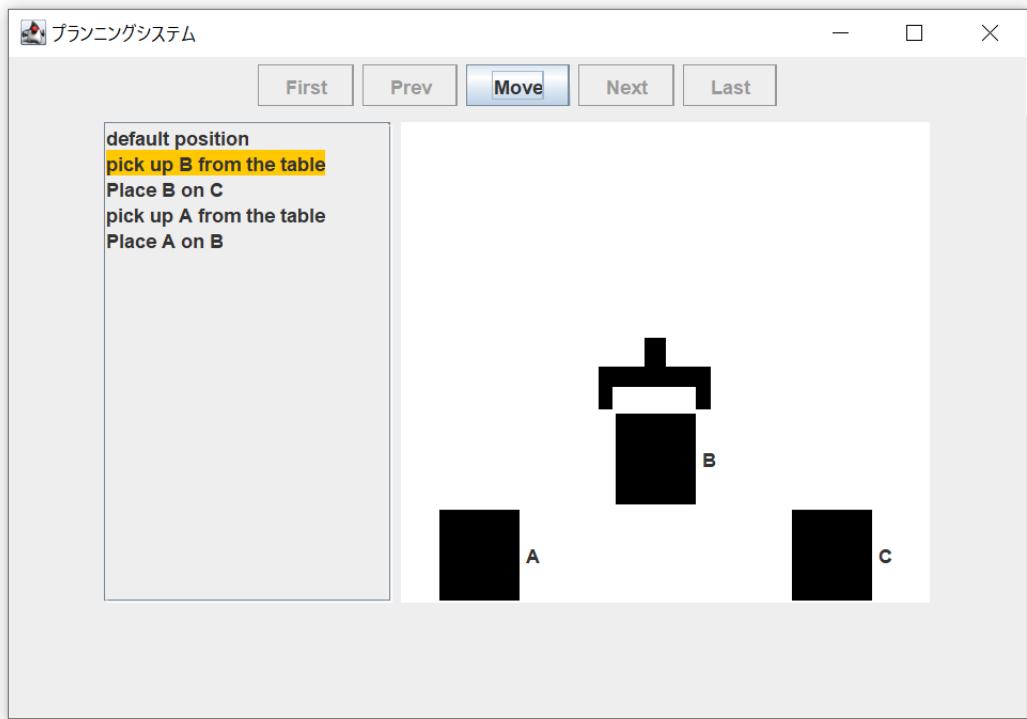


図 8: Next ボタンで経路を確認するとき

Move ボタンをクリックすると、その他のボタンが無効となり上図のようにクリック出来なくなる。さらにどの画面状態であっても初期状態の画面に遷移し、そこから一定時間ごとに導出経路が順に表示される。

最後に導出経路と使用したオペレータの出力画面が出力されると同時に無効になっていたボタンが有効になり、その他の操作が出来るようになる。

## 5.4 考察

今回の改良を行うにあたり実行結果を検証している際に GUI のウィンドウサイズを変えた際に起きてしまうバグを発見し、今回の課題としてまず改良を行った。

調べたところ removeAll メソッドを用いることでこの問題は解決できることが分かったが、このバグがなぜ起きるのかを調べたものの明確な答えを得ることが出来なかった。しかしこのバグが起きるのは再描画後であったため、再描画時に上書きする形で描画を行っていたことが原因ではないかと推測される。

前回課題ではウィンドウサイズの変更をしなくても良い GUI の作成を考えていたが、ユーザーがウィンドウサイズを変更する場合の可能性も考えてプログラムを作成する必要があると分かった。

さらに検証する中で、再描画時にウィンドウサイズを変更すると描画が一部消えてしまうことがあることも分かった。この場合も初めの描画時には消えることは無いため再描画に何か原因があると考えたが、これについては明確な原因とその対処方法が分からず、解決することが出来なかった。しかしマウスを近づけたりボタンをクリックすると消えた描画が元に戻るため、描画が出来ていないわけではないことは確認された。

この解決方法としては、再描画をしないようにプログラムを書き換える方法が考えられる。この場合私が使用している CardLayout での状態遷移の描画を、再描画の際も以前のものを消さずに Card を増やす形にする方法が最も現実的であると考えた。しかしこの場合は Next ボタンや Prev ボタンでの処理が少し複雑になることや、繰り返しプランニングを行うことで Card の枚数が増え最終的にメモリ不足によるエラーが発生してしまうという欠点がある。そもそも GUI が再描画に適していないとも考えられたが今回の場合は描画自体は正しくできていることも考え、前回までの実装をそのまま採用した。

また以前の課題ではブロックが五つ以上ある場合は描画がGUIのウィンドウサイズを超えてしまい、ウィンドウサイズを大きくしなければ図が確認できなくなってしまうという問題があった。これを修正するため描画画面に対してスクロールバーをつけ、範囲を超えた描画を行う場合はスクロールバーを移動させ、ウィンドウサイズを変更することなく表示させることを考え実装を行った。

以下にその際のソースコードを示す(ソースコード12)。

---

ソースコード12: スクロールバーの追加

---

```
1 // 変数定義
2 JPanel card = new JPanel();
3 JPanel newpage = new JPanel();
4
5 // 画面を超えた描画時にスクロールバーにする
6 JScrollPane scrollpane = new JScrollPane();
7 // 最大描画サイズの指定
8 scrollpane.setPreferredSize(new Dimension(400, 350));
9 JViewport view = scrollpane.getViewport();
10 view.setView(newpage);
11 JPanel panel = new JPanel();
12 panel.add(scrollpane);
13 // パネルへの追加
14 card.add(splist);
15 card.add(panel);
```

---

この方法を用いることで、描画サイズが最大範囲を超えた時にはスクロールバーが追加されるようになった。

この時の実行結果が以下のようになる(図9).

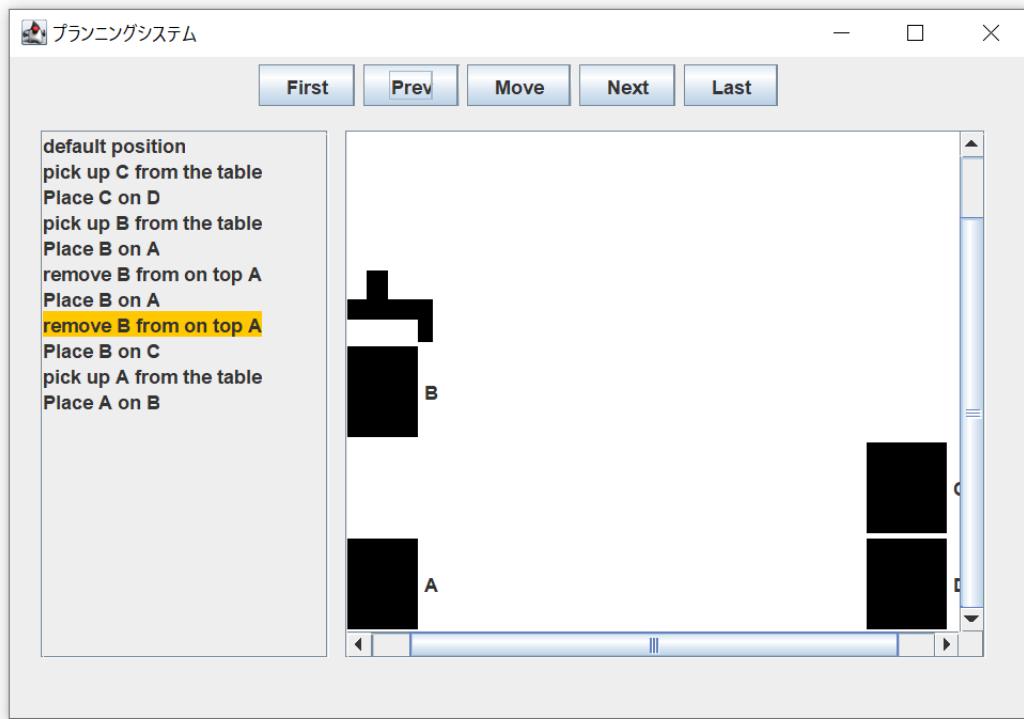


図9: スクロールバーを追加した場合

ここから分かるように、スクロールバーを追加した場合描画サイズが表示可能サイズよりも大きいため、スクロールバーが出力された場合は一度にすべての状態を確認することが出来なくなってしまう。これでは現在の状態がどうなっているのか確認するのにいちいちスクロールバーを操作しなければならず、ブロックの数が増えれば増えるほど確認に大きな負担がかかる。

さらにスクロールバーは指定しなければ常に同じ位置にあるため、状況によってはその時動かしたブロックがスクロールバーを動かさなければ確認できない位置に存在する場合も考えられる。これではGUIとして使い難いことは明らかであるため、今回はスクロールバーの追加はしないこととした。

この解決方法としては、描画方法を変えることが最も効果的と考えら

れる。今回は描画にあらかじめ用意した画像を使用したが、この方法では表示する画像のサイズを変更することが出来ない。

直接描画を行う方法であればサイズの調節が可能になるため、ウインドウサイズの変更をせず描画を行いたい場合は直接描画を行う方法が最良であると分かった。しかし前回の課題でも考察した通りその方法では座標設定を全て自身で行わなければならないため、座標管理が複雑になってしまうだけでなく、GUIの良さであるレイアウトマネージャーを活用出来なくなってしまう。のことから今回もこの方法での実装は行わないこととした。

Move ボタンの追加に関しては、動画のように滑らかな動きで経路を表現したいと考えたが、調べたところ滑らかな動きを実現するためにはレイアウトマネージャーを使用せずに座標で管理を行わなければならないことが分かった。前述の理由などから前回課題で作成したプログラムを活用して改良することを考え、今回は滑らかな動きをせず経路の過程を一回の操作(ボタンのクリック)で確認できるようにすることを目標とした。

導出経路の過程を表示するために用いたのが CardLayout であったため、これを一定時間ごと次の画面に遷移させることで過程の表示が動的に行えると考えた。どの状態にあっても Move ボタンをクリックすれば初期状態から目標状態までの過程を表示できるようにするために、Move ボタンをクリックした段階でまず初期状態の画面に遷移し、そこから導出されている経路の Card 枚数分だけ次のページへの遷移を繰り返す形とした。

このとき実行結果で述べたとおり他のボタンを動かせる状態にしておくと、Move ボタンによる繰り返しの回数と遷移状態がずれてしまうため表示がずれてしまう。そのため、Move ボタンで動作が行われている際はその他のボタンは無効にすることで正しい動作が行えるようにした。

またこの機能を実装する中で、ボタンの操作検出による動作中は繰り返し作業や更なるボタンの操作を行うことが出来ないことが分かった。試行錯誤の中で、前回まではボタンのクリック操作を ActionEvent で検知していたが、MouseEvent の mousePressed メソッドを用いればこれらの問題を解決できるのではないかと考え、mousePressed メソッドを使った

実装を行った(ソースコード11参照). 結果的にはMouseEventでも上記の動作は行うことが出来なかつたため, その他の方法を調べ, タイマークラスを用いる方法を実装して求めている動作を実現させた.

結果的にはタイマークラスを用いることで MouseEvent の mousePressed メソッドでなく ActionEvent での実装でも同様の結果が得られることが分かったが, 今回は様々な実装方法を学ぶという面から MouseEvent を用いてプログラムを作成した.

GUI の場合も, 同じ機能であっても様々な方法を用いて実装を行えることが分かった.

## 6 必須課題 6-1(発展課題 5-5 の改良)

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.

課題5にやり残した発展課題があれば参考にして拡張しても良いし, 全く新しい独自仕様を考案しても構わない. 自由に拡張するか, あるいはもし残っていた問題点があれば完成度を高めよ.

### 6.1 手法

私は, 前回課題の必須課題5-3と発展課題5-5を担当したので, 今回の課題ではこれらに応用的な機能を付与することにした. 特に, 課題5-5で実装した「禁止制約」機能を拡張し, 新たにルールの挿入/削除/編集機能を追加することとした. また, 私の担当範囲は内部処理に留まるため, 前回作成した属性クラス Attributions 内へのメソッド追加と Presenter クラスからの利用までを担当とする.

### 6.2 実装

まず, Attributions クラス内部に, 禁止制約への新たなルールの挿入/削除/編集を行う以下のメソッドを追加した.

**insertProhibitRules(ArrayList <String>targetRules)** 新たなルールを挿入するメソッド

**deleteProhibitRules(ArrayList <String>targetRules)** 指定したルールを削除するメソッド

**editProhibitRules(String beforeRule, String afterRule)** 指定したルールを編集するメソッド

以下に各メソッドについてソースコードと説明を示す。  
はじめに, 新たなルールを挿入するメソッドを示す.

ソースコード 13: insertProhibitRules メソッドの実装

---

```
1  public void insertProhibitRules(ArrayList<String>
2      targetRules) {
3          prohibitRules.addAll(targetRules);
4          prohibitBlockStates.clear();
5          prohibitBlockStates = editStatementList(
6              prohibitRules);
7      }
```

---

まず,挿入する対象のルールリストを引数で受け取り,2行目で既存の禁止制約リストに全て加える. さらに,3行目ではブロック名による禁止制約リストを一度全て削除し, 更新されたルールリストを元にして,4行目で作成し直している.

次に, 指定されたルールを削除するメソッドを示す.

ソースコード 14: deleteProhibitRules メソッドの実装

---

```
1  public void deleteProhibitRules(ArrayList<String>
2      targetRules) {
3          for(String targetRule: targetRules) {
4              prohibitRules.remove(targetRule);
5          }
6          prohibitBlockStates.clear();
7          prohibitBlockStates = editStatementList(
8              prohibitRules);
9      }
```

---

まず, 削除する対象のルールリストを引数で受け取り,2行目で既存の禁止制約リストから削除する処理を行なっている. 5,6行目では先ほど同様

にブロック名による表現に反映している。

続いて、指定されたルールを編集するメソッドを示す。

ソースコード 15: editProhibitRules メソッドの実装

---

```
1  public void editProhibitRules(String beforeRule,
2                                  String afterRule) {
3          prohibitRules.remove(beforeRule);
4          prohibitRules.add(afterRule);
5          prohibitBlockStates.clear();
6          prohibitBlockStates = editStatementList(
7              prohibitRules);
}
```

---

まず、編集する対象のルールリストを引数で受け取り、2行目で既存の禁止制約リストから編集前ルールを削除し、3行目で編集後ルールを挿入する処理を行なっている。4,5行目では先ほど同様にブロック名による表現に反映している。

また、Presenter クラスにおいてもこれらを利用する同名メソッドを定義した。

なお、これらのメソッドでは、引数で受け取るリストの要素数によって、GUIへの反映処理を切り分けられる実装となっている。具体的には、要素数が 0 の場合や null といった場合には、処理を行うことができないように、GUIへとフィードバックを行うように変更した。

### 6.3 実行例

以下に、今回追加したメソッドをテストした結果を示す。

ソースコード 16: editProhibitRules メソッドの実装

---

```
1  ~/Programming2/individual_report/attributions
2  ●java Test
3  **** Attribution rules ****
4  A is blue
5  A is ball
6  B is green
7  B is trapezoid
8  C is red
```

---

```

9      C is box
10     ##### Add prohibitRule #####
11     ***** ProhibitRule:ball on ball *****
12     ***** ProhibitRule:trapezoid on ball *****
13     ***** ProhibitRule:trapezoid on trapezoid *****
14     ***** ProhibitRule:box on ball *****
15     ***** ProhibitRule:box on box *****
16     ***** ProhibitRule:pyramid on ball *****
17     ***** ProhibitRule:ball on pyramid *****
18     ***** ProhibitRule:box on pyramid *****
19     ***** ProhibitRule:trapezoid on pyramid *****
20     ***** ProhibitRule:pyramid on pyramid *****
21     ++++++ EditStatement ++++++
22     ball on ball =====> A on A
23     trapezoid on ball =====> B on A
24     trapezoid on trapezoid =====> B on B
25     box on ball =====> C on A
26     box on box =====> C on C
27     pyramid on ball =====> pyramid on A
28     ball on pyramid =====> A on pyramid
29     box on pyramid =====> C on pyramid
30     trapezoid on pyramid =====> B on pyramid
31     pyramid on pyramid =====> pyramid on pyramid
32
33     ---- insert rules ----
34     blue on red
35     green on blue
36     ++++++ EditStatement ++++++
37     ball on ball =====> A on A
38     trapezoid on ball =====> B on A
39     trapezoid on trapezoid =====> B on B
40     box on ball =====> C on A
41     box on box =====> C on C
42     pyramid on ball =====> pyramid on A
43     ball on pyramid =====> A on pyramid
44     box on pyramid =====> C on pyramid
45     trapezoid on pyramid =====> B on pyramid
46     pyramid on pyramid =====> pyramid on pyramid
47     blue on red =====> A on C
48     green on blue =====> B on A
49
50     ---- delete rules ----

```

```

51    blue on red
52    ++++++ EditStatement ++++++
53    ball on ball =====> A on A
54    trapezoid on ball =====> B on A
55    trapezoid on trapezoid =====> B on B
56    box on ball =====> C on A
57    box on box =====> C on C
58    pyramid on ball =====> pyramid on A
59    ball on pyramid =====> A on pyramid
60    box on pyramid =====> C on pyramid
61    trapezoid on pyramid =====> B on pyramid
62    pyramid on pyramid =====> pyramid on pyramid
63    green on blue =====> B on A
64
65    変更前のルール:green on blue
66    変更後のルール:yellow on blue
67    ++++++ EditStatement ++++++
68    ball on ball =====> A on A
69    trapezoid on ball =====> B on A
70    trapezoid on trapezoid =====> B on B
71    box on ball =====> C on A
72    box on box =====> C on C
73    pyramid on ball =====> pyramid on A
74    ball on pyramid =====> A on pyramid
75    box on pyramid =====> C on pyramid
76    trapezoid on pyramid =====> B on pyramid
77    pyramid on pyramid =====> pyramid on pyramid
78    yellow on blue =====> yellow on A

```

---

331行目で属性ルールおよび禁止制約の追加と, 禁止制約をブロック名による表現に変換する処理の結果を示している. ここまでは, 前回課題での実装範囲と同じである.

3348行目においては, 禁止制約挿入処理とブロック名での表現への変換処理の結果を示している. 特に注目すべきは, 47・48行目で, これらは新たに挿入された禁止制約の影響を受けた部分である. 先ほどは存在しなかった

```

blue on red=====>A on C
green on blue=====>B on A

```

という表現が現れている.

50 63 行目においては、禁止制約削除処理とブロック名での表現への変換処理の結果を示している。特に注目すべきは、62 行目と 63 行目の間で、先ほどは存在した

```
blue on red=====>A on C
```

が削除されてなくなっている。

65 78 行目においては、禁止制約編集処理とブロック名での表現への変換処理の結果を示している。特に注目すべきは、78 行目で、先ほどは

```
green on blue=====>B on A
```

となっていた部分が、

```
yellow on blue=====>yellow on A
```

として編集されている。

## 6.4 考察

今回、禁止制約を自由に挿入/削除/編集することができる機能を追加したことによって、前回課題では物理法則に則ったもののみであったが、ユーザー独自のカスタマイズを行うことが可能となった。特に、色などの属性による制約を加えることによって、さらに複雑な問題設定が可能となる。プランニングなど、知識ベース型の人工知能システムは、十分な知識を取得可能な場面においては、データ駆動型のシステムに比べて低コストで高精度の予測を行えることもあり、現在でも非常に有効な手段である。むしろ、このような場合には、知識ベース型のシステムを選択すべきであり、いたずらにデータ駆動型の機械学習を導入すべきではないとも考えられる。双方の特徴を知り、長所・短所を正しく認識した上で社会への応用が可能となることを胸に刻む必要があると感じた。また、ただ理論やアルゴリズムを組み上げるのみではなく、「多くの人に使ってもらえるような UI/UX」を考える必要もある。情報プロダクトにおいても、エンジニアの自己満足ではなくユーザーの側に立ってプロダクトを作ることができて初めて社会的価値があると言える。

さらに、本演習前半の「Java による知識システム実装」では、知識システムプログラムの改良と GUI 実装を行なうことを通して、グループワーク

の難しさも体験した。個人それぞれの事情や性格が違うので、なかなか上手くまとまらない場面もあり、毎回レポートやプログラムの完成が提出期限間際となっていた。私個人としては、早めに課題を終わらせておく性格であるため、そうでない人と協調して作業を行うことは非常に難しく感じた。しかし、GUIなど特定の個人が興味を抱いている分野に関しては率先して取り組む姿が見られ、担当者に頼もしさを感じることもあった。実際に社会に出た際に求められるスキルとして、グループ開発は必須となると考えられるので、現段階から少しでもそれを体験できたのは非常に良かったと感じている。

## 7 必須課題 6-1(発展課題 5-7 の改良)

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。

課題5にやり残した発展課題があれば参考にして拡張しても良いし、全く新しい独自仕様を考案しても構わない。自由に拡張するか、あるいはもし残っていた問題点があれば完成度を高めよ。

私の担当箇所は、発展課題5-7に対してUnityを用いて実装したプログラムの、GUI改善やプランニングへの機能追加である。

### 7.1 手法

発展課題5-7では、ブロックワールドにおけるプランニングを実現するための過程として、以下のような実装を行った。

1. 空間やプランに関するオブジェクトの生成。
2. プランニングを行うための、オブジェクトの動作等に関するスクリプトの作成。

これに引き続いて今回は、ブロックワールドにおけるプランニングを実現するために、以下のような方針を立てた。

1. プランに用いるブロックをより正確に生成できるようにする.
  2. ブロックの管理を容易にする.
  3. プランニングの情報を可視化する.
1. に関して, GUI を画面上に表示することで, ブロックを名前や色, 形を指定した上で生成できるような仕様とした.
  2. に関して, オブジェクトの情報を一覧で表示して生成したオブジェクトの管理を視覚的に行えるような仕様とした. また, フォーカスしているブロックを視覚的に確認できるように, 選択中のオブジェクトのアウトラインが表示されるような仕様とした.
  3. に関して, 2. で作成する一覧と連動して, 選択したブロックの情報が画面左下のステータスバーで確認できるようにした.

## 7.2 実装

発展課題 5-7 で作ったオブジェクトは以下のとおりである.

**Main Camera** 主カメラに関するオブジェクト. Room 全体をやや見下ろし気味に映す.

**Directional Light** オブジェクト全体を照らす照明.

**Master** スクリプトをアタッチするための空オブジェクト.

**Room** 6 個の Plane オブジェクトを子に持つ, 立方体の部屋を構成するオブジェクト.

**Cube** 直方体のブロックを生成するプレハブ.

**Sphere** 球のブロックを生成するプレハブ.

**Torus** 円環体のブロックを生成するプレハブ.

新しく作った主なオブジェクトは以下のとおりである.

**EventSystem** GUI において Button や InputField を機能させるための, フォーカス情報等を掌るオブジェクト.

**Canvas** Generator, Preparator, Stater を子オブジェクトを持つ, GUI の大元となるオブジェクト.

**Generator** オブジェクトを生成するためのパネル. 子オブジェクトに InputFieldName, DropdownColor, DropdownShape, ButtonGen を持つ.

**Preparator** 生成したオブジェクトの一覧を管理するパネル. 子オブジェクトに ScrollView, ButtonRm, ButtonPlanning を持つ.

**Stater** オブジェクトの情報等を表示するためのパネル. 子オブジェクトに TextStatus, Scrollbar を持つ.

**ListObj** Preparator における ScrollView の要素となるプレハブ.

C#スクリプトでは以下のものが実装されている.

**Clicked** クリックされたオブジェクトにフォーカスを当てるスクリプト. Master にアタッチされる.

**Operationg** Clicked でフォーカスされたオブジェクトにキーボード入力を反映するスクリプト. Master にアタッチされる.

**Generator** Generator オブジェクトで得た情報に合わせて, ブロックを生成するスクリプト. ButtonGen にアタッチされる.

**Destroyer** Preparator オブジェクトで選択されたブロックを削除するためのスクリプト. ButtonRm にアタッチされる.

**Manager** ListObj プレハブにアタッチされる, 各インスタンスが 1 対 1 で対応するブロックを保持するためのスクリプト.

**SelectOnList** ListObj プレハブにアタッチされる, インスタンスが自身にアウトラインや削除のフォーカスを当てるためのスクリプト.

**Starter** GUI の一部を非表示にし, プランニングを開始するためのスクリプト. ButtonPlanning にアタッチされる.

**CollisionGetter** 自身と衝突中のブロックを保持するスクリプト. 各ブロックにアタッチされる.

**StateGetter** SelectOnList でフォーカスされたブロックと衝突中のブロックを Stater に表示するスクリプト. Master にアタッチされる.

### 7.2.1 プランに用いるオブジェクトをより正確に生成できるようにする.

Unity[8]においてGUIを作成するために、Unityの標準機能に含まれるuGUI[9]を用いた。

まず、GUI制作をUnity上で開始するために、Canvasオブジェクトを生成した。Canvasオブジェクトは実行環境によって、そのサイズが動的に変化するため、シーンビューにおける大きさは意味を持たない。その仕様を理解するまでは扱いに戸惑った。

次に、ブロック生成のためのGUIであるGeneratorオブジェクト（図10）を作るために、GeneratorオブジェクトにはPanelを、名前を入力するためのフォーム(InputFieldName)にはInputFieldを、色や形の選択(DropdownColor,DropdownShape)にはDropdownを、生成ボタン(ButtonGen)にはButtonをそれぞれ用いた。これらUIに関するパーツはいずれもRectTransformコンポーネントを用いて配置される。

RectTransformの特徴として、アンカーとピボットが挙げられる。これらはブロック等のオブジェクトの座標指定で用いられるTransformコンポーネントには含まれない。今回ピボットは用いなかったが、アンカーはGUIパーツの配置を行う上では欠かせないものであることが分かった。

アンカーは、指定した位置を基準として、オブジェクトの相対座標を指定するものである。図10に示すように、Canvasの右上部である位置に表示されている4つの三角がアンカーの基準点を示すものであり、パネルはアンカーから左下部に相対座標で配置されている。これにより、実行環境により画面サイズが変化しても、図11や図12で示すようにレイアウトを崩さずに表示されることが確認できる。

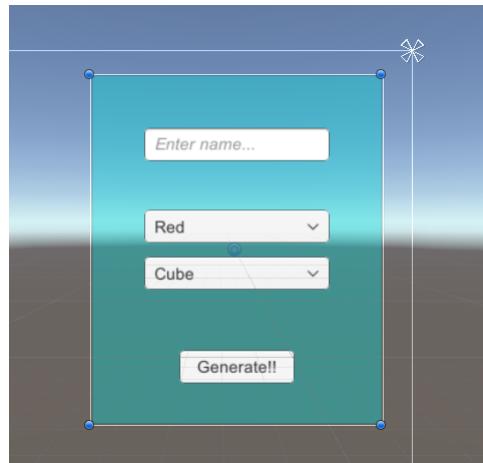


図 10: Generator オブジェクト

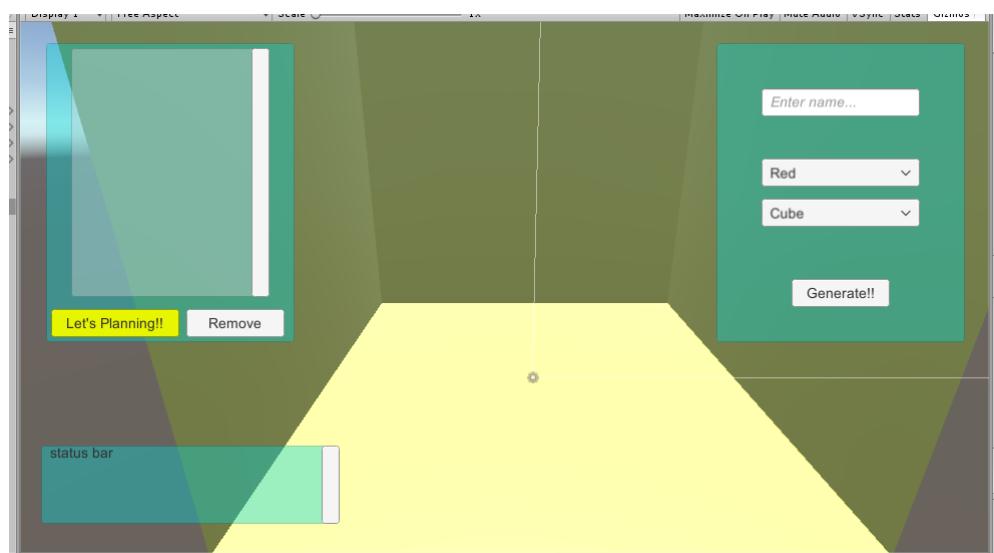


図 11: 横長の画面における表示

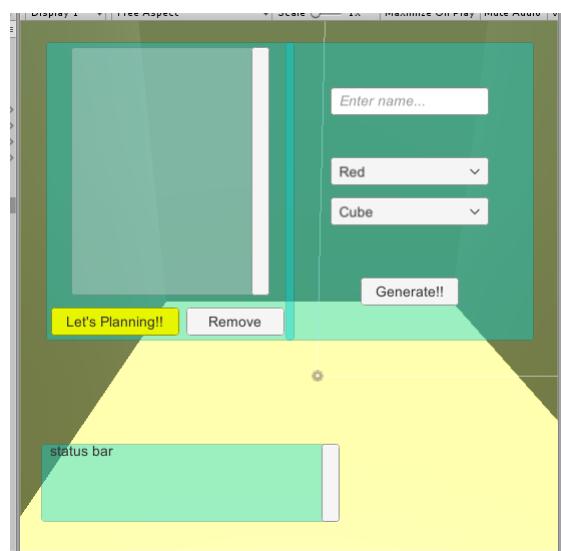


図 12: 縦長の画面における表示

次に、配置した Generator 内のオブジェクトを用いて、ブロックを生成するためのスクリプト (Generator.cs) を生成する。ブロックはソースコード 17 で示すように生成される。

ソースコード 17: Generator クラス

```
1 public class Generator : MonoBehaviour
2 {
3     ...
4     public void Generate()
5     {
6         ...
7         GameObject obj = (GameObject)Resources.Load("Cube
8             ");
9         switch (ddShape.value)
10        {
11            case 0:
12                obj = (GameObject)Resources.Load("Cube");
13                break;
14            case 1:
15                obj = (GameObject)Resources.Load("Sphere
16                    ");
17                break;
18            case 2:
19                obj = (GameObject)Resources.Load("Torus");
20                break;
21        }
22        GameObject target = Instantiate(obj, obj.transform
23             .position, Quaternion.identity);
24        target.name = name;
25        target.GetComponent<Renderer>().material.color =
26            getColor(ddColor.value).color;
27
28
29        ...
30
31        Material getColor(int num)
32        {
```

```
33     Material color = matR;
34     switch(num)
35     {
36         case 0:
37             color = matR;
38             break;
39         case 1:
40             color = matG;
41             break;
42         case 2:
43             color = matB;
44             break;
45     }
46     return color;
47 }
48 }
```

---

ddShape.value を介して DropdownShape で選択された形を選択し, Resource.Load メソッドでプレハブを取得している。取得したプレハブを Instantiate メソッドで生成し, 生成したインスタンスの名前を InputFieldName で入力した名前に書き換え, 色を ddColor.value を介して DropdownShape で選択された色に書き換えている。

このスクリプトを ButtonGen にアタッチすることで, ボタンをクリックしたときにこのスクリプトは呼び出され, 実行される。

### 7.2.2 ブロックの管理を容易にする.

ブロックを管理するための GUI である Preparator オブジェクト (図 13) を作るために, Preparator オブジェクトには Panel を, ブロックを表示するリスト (ScrollView) には ScrollView を, リスト内の要素 (ListObj) には Button をプレハブ化したもの, リスト内の要素削除ボタン (ButtonRm) とプランニング開始ボタン (ButtonPlanning) には Button をそれぞれ用いた.

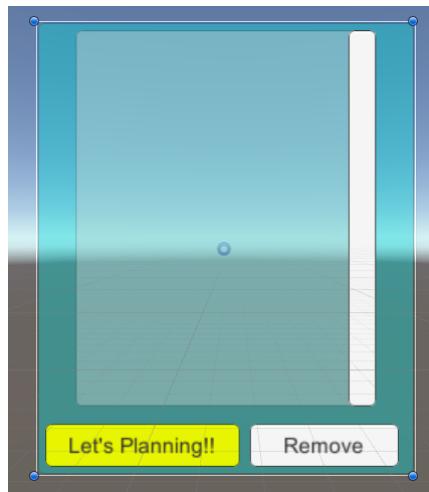


図 13: Preparator オブジェクト

ScrollView を作成する [10] にあたり, ListObj は要素の選択を可能にするために, Button を用いて実装した.

Generator スクリプト (ソースコード 17) の変数 `_text` によって, ListObj からインスタンスが生成されている. この際, スクロールバーに表示するためには適切な位置の子オブジェクトとして格納する必要がある. これは, `Instantiate` の引数に親オブジェクトの `transform` を渡すことで実現した. また, ここで生成したインスタンスに, 先程生成したブロックのインスタンスへの参照も, Manager スクリプトの変数を介して渡している. これにより, ListObj のインスタンスから対応するブロックへのアクセスを可能としている.

次に, ScrollView から選択した要素にアウトラインを付けるためのスクリプト `SelectOnList`, `StateGetter` を実装する.

まず、アウトラインはUnityの標準機能に備わっていないため、アセットQuickOutline[11]を導入し、各ブロックに非アクティブ状態でアタッチした。

その上で、引数で渡されたブロックのアウトラインをアクティブにするStateGetterクラス内のFocusOutlineメソッドを、ソースコード18に示す。

---

ソースコード18: StateGetterクラス内のFocusOutlineメソッド

---

```
1     public GameObject target;
2
3     public void FocusOutline (GameObject newObj) {
4         if (target != null) {
5             target.GetComponent<Outline> ().enabled =
6                 false;
7         }
8         if (newObj != null) {
9             newObj.GetComponent<Outline> ().enabled = true
10            ;
11        }
12        target = newObj;
13    }
```

---

アウトラインをアクティブにしたブロックをtargetに保持することで、フォーカスが外れた際の非アクティビ化も同時に見えるようになっている。StateGetterはMasterに1つだけアタッチされることで、フォーカスの管理を一元的に行うこととした。

SelectOnListスクリプトは各ListObjインスタンスにアタッチされることで、フォーカスが当てられたときにSelectOnListスクリプトからFocusOutlineメソッドを呼び出すことで、自身にアウトラインを付けることができる。(図14,15)

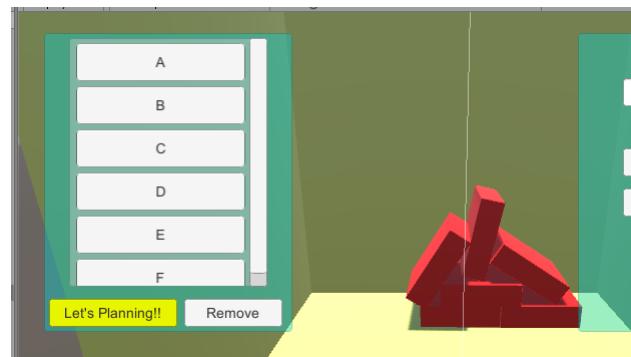


図 14: 非フォーカス時

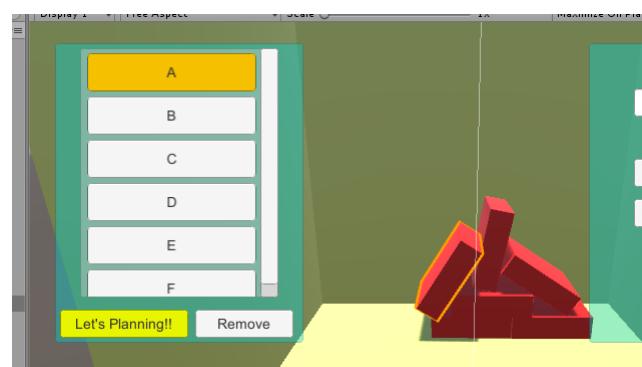


図 15: フォーカス時

### 7.2.3 プランニングの情報を可視化する.

プランニングの情報表示のための GUI である Stater オブジェクト（図 16）を作るために、Stater オブジェクトには Panel を、情報を表示するテキスト (TextStatus) に Text を用い、一度に全部を表示できないときのために Scrollbar を作成した [12].

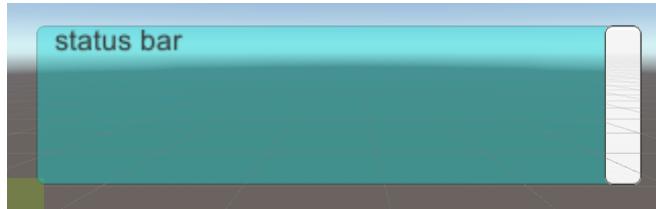


図 16: Stater オブジェクト

フォーカスされたオブジェクトの衝突情報を取得するために、CollisionGetter スクリプトを作成し、各ブロックにアタッチした。それをソースコード 19 に示す。

ソースコード 19: CollisionGetter クラス

---

```
1 public class CollisionGetter : MonoBehaviour {
2
3     public List<GameObject> colList;
4
5     void Awake () {
6         colList = new List<GameObject> ();
7     }
8
9     void OnCollisionStay (Collision collision) {
10         if (!colList.Contains(collision.gameObject)) {
11             colList.Add (collision.gameObject);
12         }
13     }
14     void OnCollisionExit (Collision collision) {
15         colList.Remove (collision.gameObject);
16     }
17 }
```

---

衝突相手のブロック名が colList に 1 つだけ格納される。この際、同じブロックに対して複数の衝突判定を持つときに colList に複数同じ要素が

含まれないようにするために、要素の追加は OnCollisionEnter メソッドではなく OnCollisionStay メソッドを用いて常に監視することで、colList の要素が常に正確になるような実装をした。

このリストを StateGetter から取得し、Stater に反映することで、フォーカスしたブロックの衝突判定を、常に可視化することを実現した。

### 7.3 実行例

task5-7/build/Block World Planning.exe を起動したところ、図 17 のような画面が得られる。

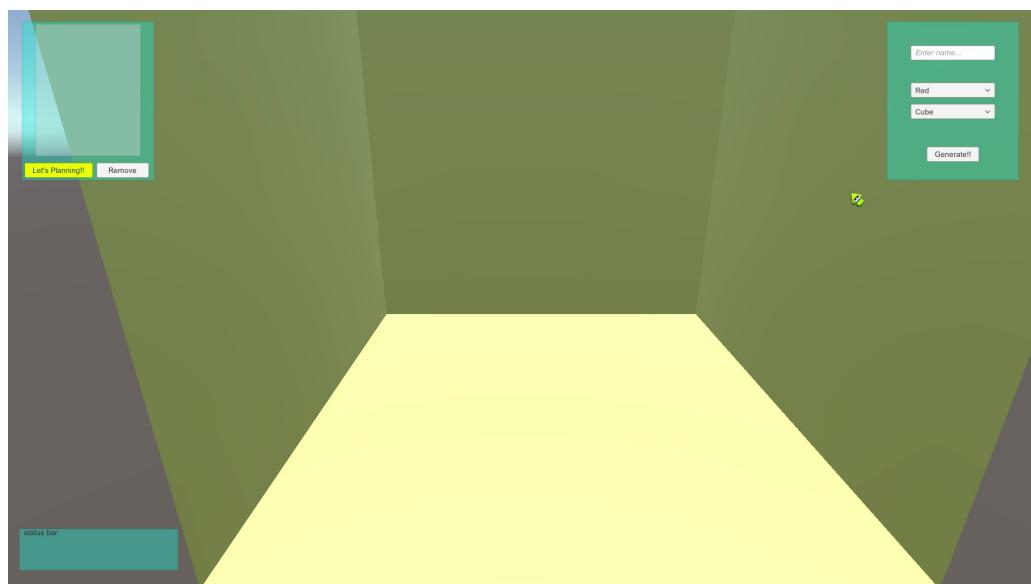


図 17: 起動時の画面

Generator を用いてブロックを生成すると、Preparator にも反映されることが分かる(図 18,19).

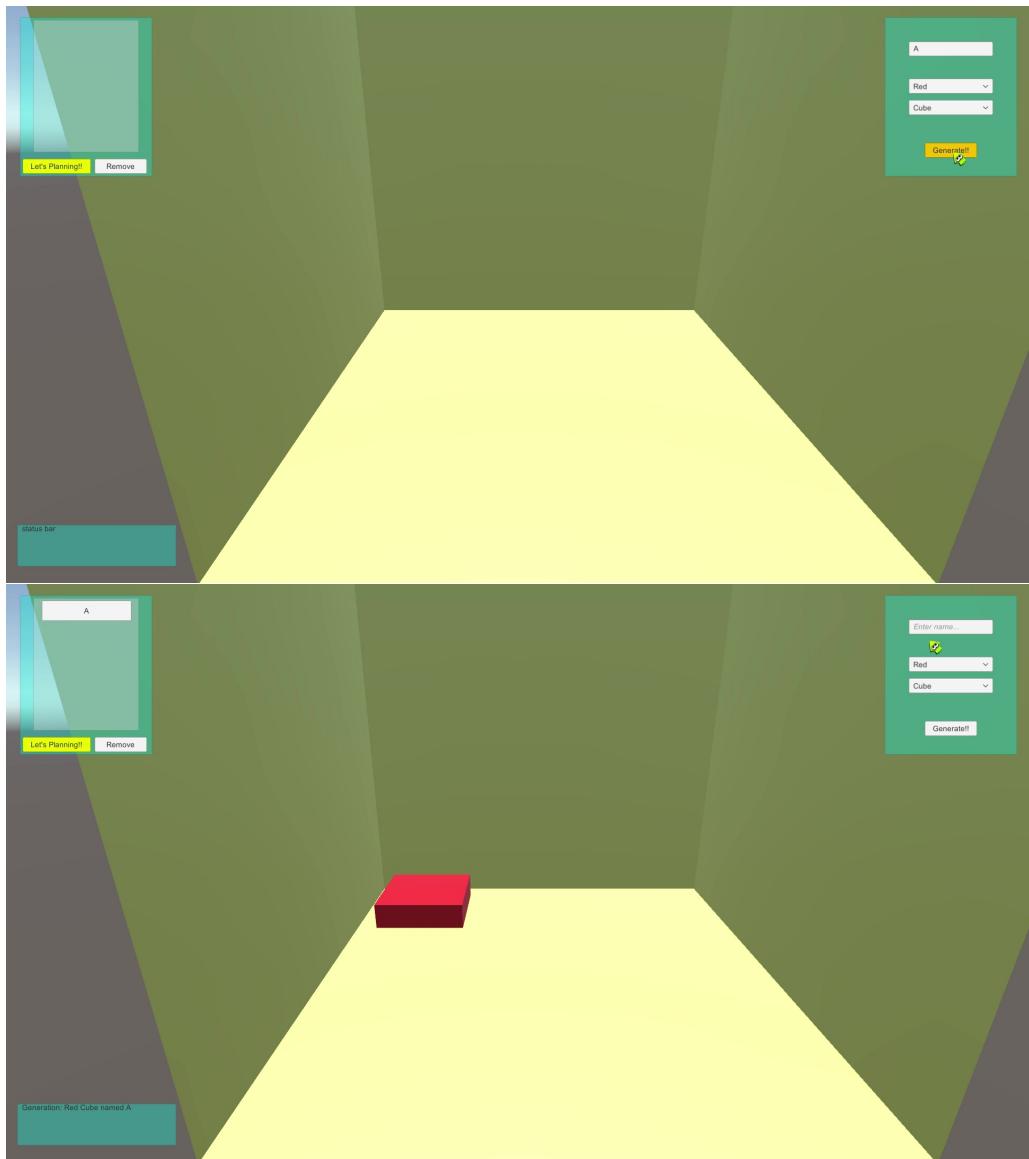


図 18: 赤い球を A という名前で生成

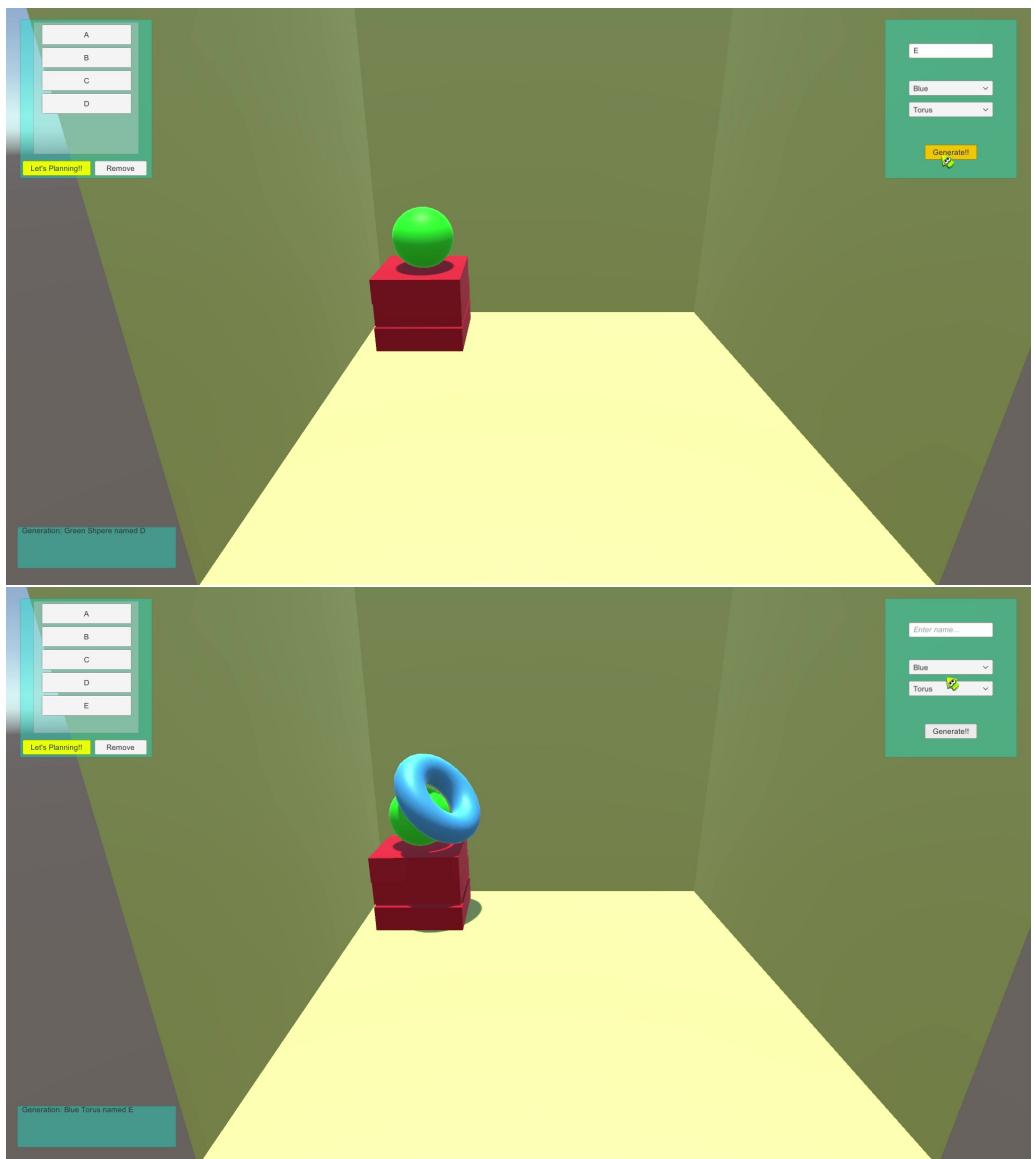


図 19: 青い円環体を E という名前で生成

Preparator で要素を選択すると、対応するブロックのアウトラインが表示される(図 20).

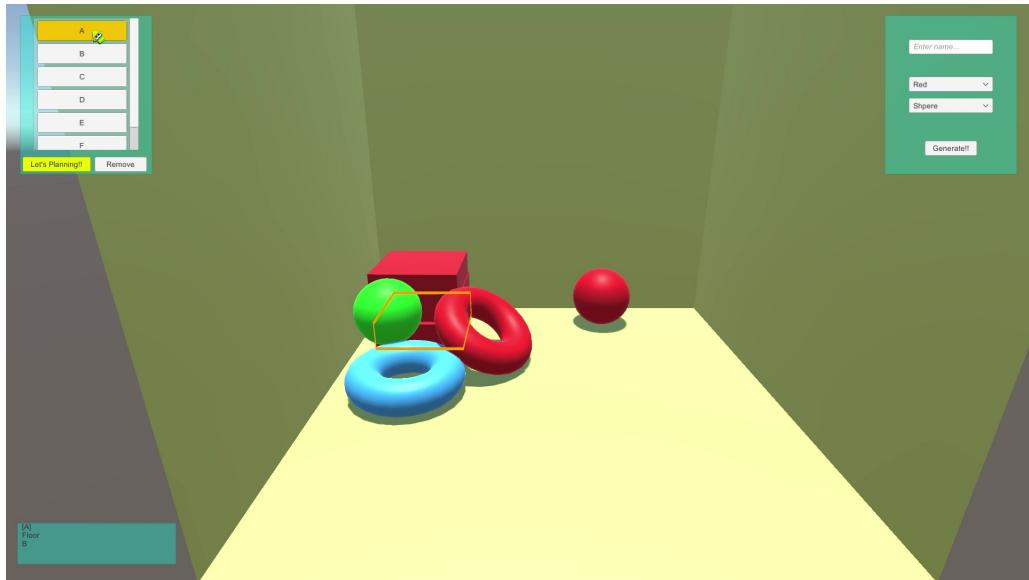


図 20: A にフォーカス

フォーカスしたブロックは Remove ボタンで削除できる (図 21).

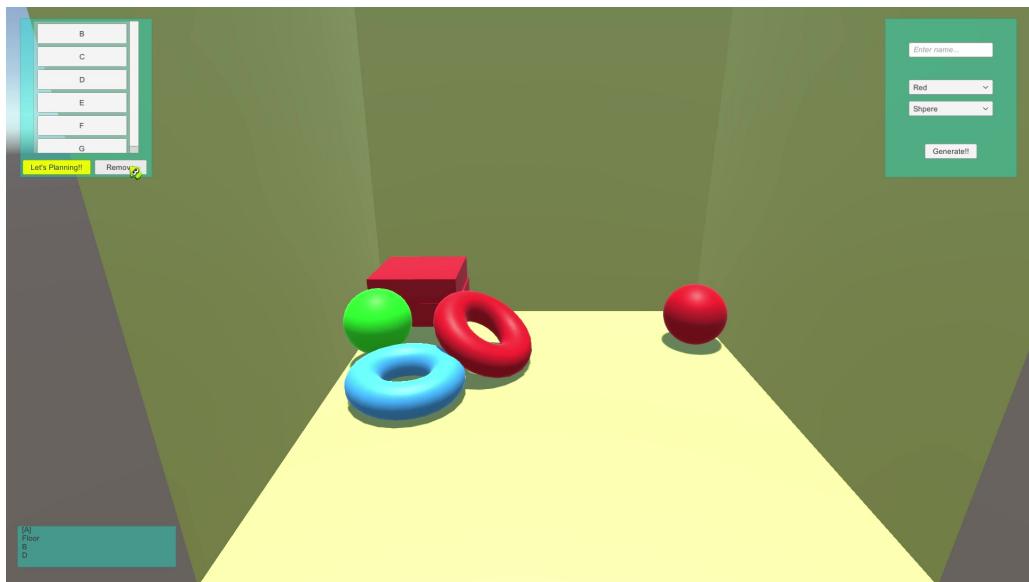


図 21: A を削除

ブロックの生成や削除が完了したら、Let's Planning ボタンでプランニングを開始する(図 22).

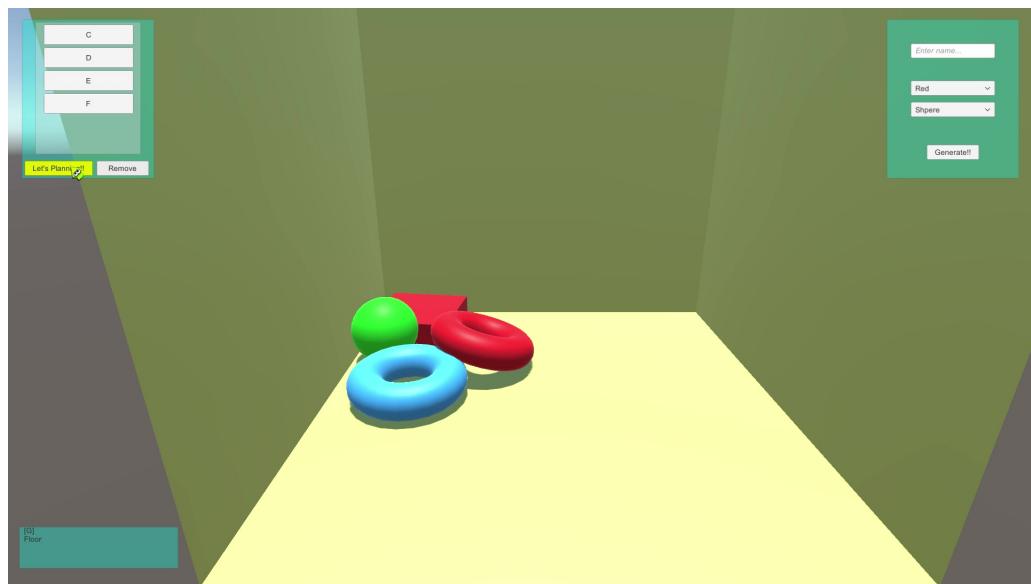


図 22: プランニング初期状態

フォーカスしたブロックと衝突しているブロックの情報がStaterに表示される(図23,24).

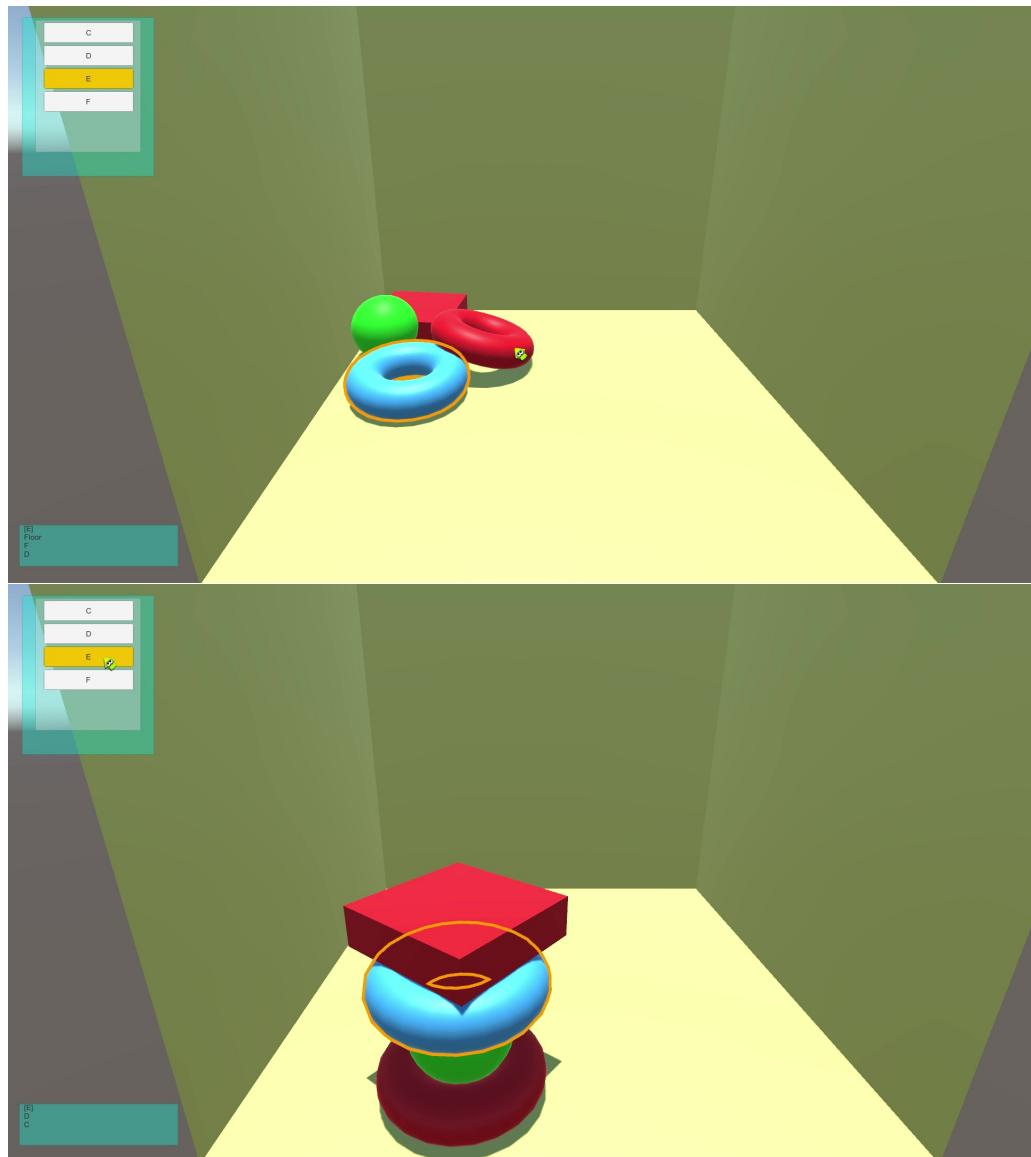


図23: Eにフォーカスを当て、プランニング実行

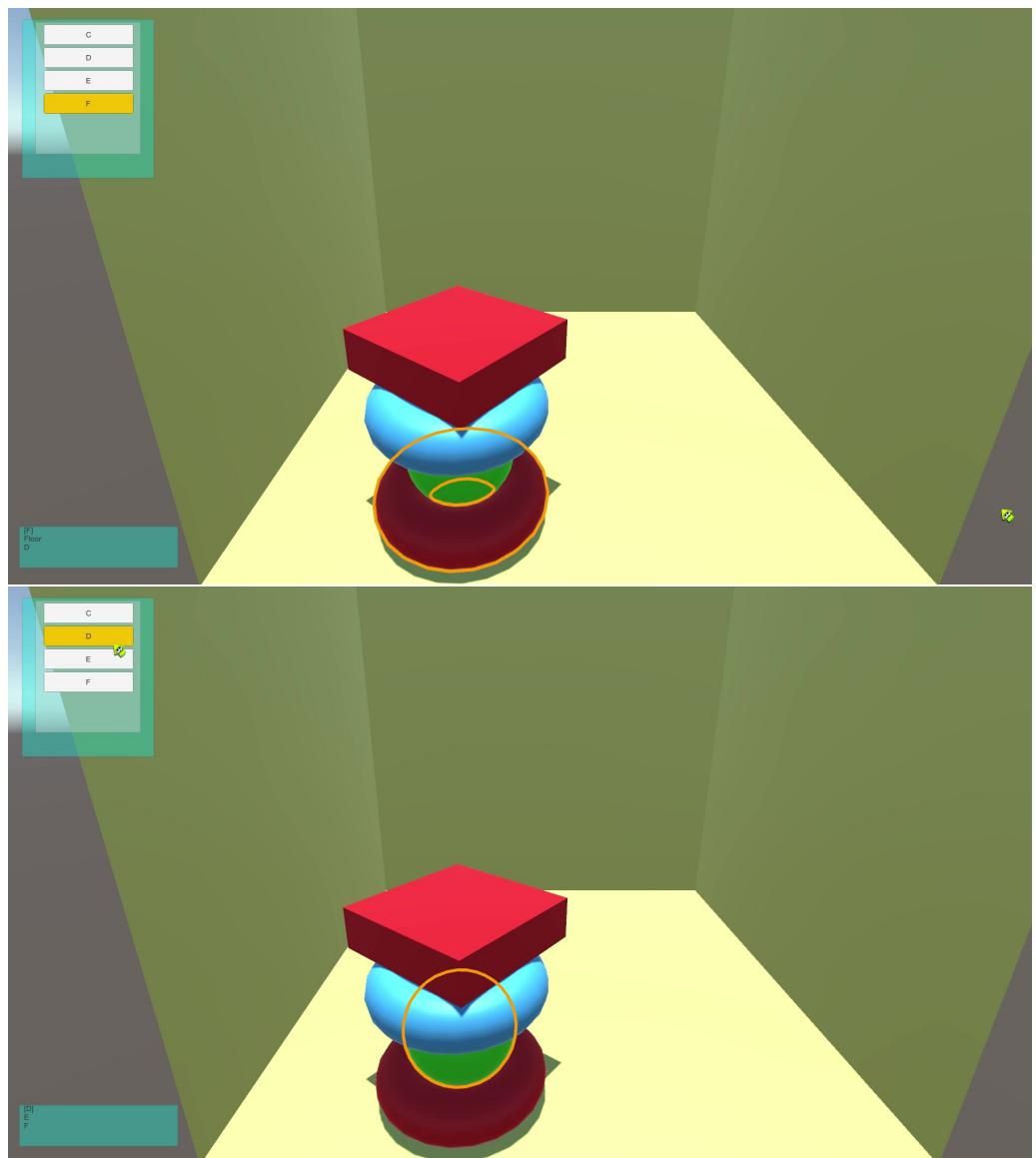


図 24: プランニング完了時の各ブロックの衝突情報

## 7.4 考察

Generator スクリプト (ソースコード 17) にて, switch を用いているが, ここで用いる変数 obj の宣言の際, 初期化しないとコンパイルエラーになることは C# の特徴として挙げられると考えられる.

GUI を実装し始めた際, 自動で追加された EventSystem というオブジェクトを誤って消してしまい, 実行しても Button や InputField が全く反応しないという事態に陥った. EventSystem を追加し直すことで修正できたが, このように自動で用意されるものに甘んじて各種機能を使うことは, 便利な反面で, 何かトラブルが起こったときに簡単には自力で直せないという事態が起こり得ることに繋がるため, 自分が利用している機能はどのようなものに基づいてどのように動いているのかを探求することは, 今後それを用いてより高度なことをするときのために大切なことだと感じた.

uGUI での GUI 実装と Swing での GUI の実装と比較してみると, Swing よりも並列動作は扱いやすく, Swing よりも変数の管理は面倒だと感じた. Unity での GUI の部品となる各オブジェクトは, 常時独立して動き続けているため, 並列的な動作を行ったときにどのような挙動をとるかをオブジェクト毎に確認できるため挙動の管理がしやすいのだと考えられる. 一方で, 各オブジェクトが独立しているということは, オブジェクト間の変数はスクリプトを用意して共有する必要があるということなので, 並列動作の分かりやすさと変数の管理の楽さはトレードオフの関係にあることが考えられる.

また, uGUI と Swing のいずれも, フォーカスの操作はとっつき辛いと感じた. しかし, uGUI では GUI 全体のフォーカスが 1 つの EventSystem によって明確に行われているため, その点では uGUI の方が把握しやすいと感じた.

しかし, GUI の一番の難点は, やはり実行する環境によって表示のされ方が異なってくることだと考えられる. その点では, Swing も標準の機能としてレイアウトが用意されてはいるが, uGUI でのアンカーを用いた配置の方が, 斷然とっつきやすく, 調整もしやすいと感じた.

以上のことから, 今後 GUI を作る際は, 基本的に Swing よりも uGUI で良いと考えられる. しかし, 軽量化という点において uGUI は Swing に優ることができるか, または優る必要があるかということは今後の課題である.

Unity を使って一番悩まされたことが、プレハブにアタッチしたコンポーネントの変数が public でも、その変数に他のオブジェクトを直接アタッチして参照させることができなかったことである。そのため、GameObject.Find メソッドで毎度目的のオブジェクトを探して参照を作っているが、直接アタッチできる方が効率的だと考えられる。前回も、通常のオブジェクトだと上手くいくのにプレハブだとうまくいかないようなこと(Torus の衝突判定) があったように、プレハブの特性や注意点で理解できていないことがまだ多々あることが考えられる。Unity を使っていく上でプレハブは欠かせない機能なので、今後も積極的に使って学んでゆきたい。

また、Unity ではオブジェクトを消去するという機能があるため、この点も注意して使っていかなければいけないと考えられる。Unity に限らず並列処理を行うようなプログラムでは、先程まで参照していたものが突然なくなるということが往々にして起こり得るため、その点も留意した上でプログラム実装を行ってゆく必要があると考えられる。

また、Unityにおいて、変数をオブジェクト間で共有するために、いちいちスクリプトを作成する必要があると述べたが、共有する変数ごとにスクリプトを作成するのではなく、共有すべき変数をまとめて管理するようなスクリプトと、そのためのオブジェクトを作成して、そこを共有変数の仲介場所とすると改善できることが考えられる。例えば、今回のプログラムでは Stater がその仲介場所として適していたと考えられる。更に、Stater を仲介場所にすればステータスの表示もより自在に行えるようになることが考えられ、一石二鳥である。また、このような方法で共有する変数を管理したほうが、プログラムの保守性にも良いということが考えられる。

今回の実装について、開始時点を決めれるようにしたことで、初期状態の明確化ができるようになった。

最終的に自動でのプランニングを実装するには、その前段階として実装する必要のあるものが多くあったため、その過程の一部を実装しようと思い今回作ったような実装となった。ブロックの属性付与や生成したブロックの管理、衝突情報の管理が行えるようにしたことで、プランニング実装に向けて次に必要なのは、衝突情報を突き詰め、どちらのブロックが上に乗っているかといった相対的な位置関係の取得・管理をするこ

とと、それに基づきどの物体を目標状態に向けてどう動かすかを選択することの実装だと考えられる。しかし、3次元空間における物体間の衝突の仕方は多様であり、どちらのブロックが上に乗っているかを厳密に判別することは困難な場合が多く存在することが考えられる。また、それを解決したところで、物体を動かすことによる状態変化も不定であり、目標状態に向けた行動の選択を一意的に決めるることは難しいと考えられる。このように、実装上の難しさだけでなく、そもそも3次元空間上においてどのようにプランニングを定義するかという実世界的な難しさが数多くあることを改めて痛感した。

## 8 感想

グループみんなで協力して、最後までやりきれて良かった。  
8班サイコー!!

## 参考文献

- [1] Javaによる知能プログラミング入門 -著：新谷 虎松
- [2] 知識システムの実装基礎 -著：新谷 虎松
- [3] 人工知能の基礎知識 -著：太原 育夫
- [4] LATEX2 $\epsilon$  美文書作成入門 -著：奥村 晴彦
- [5] 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.
- [6] Let's プログラミング Swing を使ってみよう, <https://www.javadrive.jp/tutorial/> (2019年1月3日アクセス) .
- [7] Javaによる知能プログラミング入門 -著：新谷 虎松
- [8] Unity Technologies.: 『Unity - Manual: Unity User Manual (2019.2)』 <https://docs.unity3d.com/Manual/index.html> (2020/01/06 アクセス)

- [9] 竹内 大五郎 : 『uGUI チュートリアル - Metal Brage』  
<http://www.metalbrage.com/UnityTutorials/uGUI/index.html>  
(2020/01/06 アクセス)
- [10] shimoaraiso : 『Unity の ScrollView で一覧表示を作成する』  
[https://tech.pjin.jp/blog/2016/08/30/unity\\_skill\\_3/](https://tech.pjin.jp/blog/2016/08/30/unity_skill_3/) (2020/01/06 アクセス)
- [11] chrisnolet : 『chrisnolet/QuickOutline: Unity asset for adding outlines to game objects』 <https://github.com/chrisnolet/QuickOutline>  
(2020/01/06 アクセス)
- [12] hiyotama : 『【Unity 開発】uGUI の Scrollbar の使い方 【ひよこエッセンス】 - Unity(C#) 初心者・入門者向けチュートリアル ひよこのたまご』 <https://hiyotama.hatenablog.com/entry/2015/07/04/090000>  
(2020/01/06 アクセス)
- [13] PukiWiki Developers Team. : 『【LaTeX 入門/相互参照とリンク - TeX Wiki】 <https://texwiki.texjp.org/?LaTeX?LaTeX%2F%20相互参照とリンク#h84e81eb>