

COMP 1110/6710

Thu15b:

Yuqing Zhai (u6865190)

Yufan Zhou (u6213488)

Huhan Jiang (u6090688)





Assignment Objectives

- The dices have highway and railway connection
- The player has to roll four ordinary dices seven times
- Fill the board with the dices, and the connection have to be valid
- Each round of game can use only one special dice, and you can only use three times of the special dices in the game
- The goal is to establish a network of routes as long as possible and connect exits as more as possible
- The final score is decided by the longest railway/highway, how many exits it connects, and how many central grids it covers



Design Approach

1. Tile
2. Method: new string [add a tile string into placement string]
3. Give viable pieces method
4. Use recursive helping method to get all solutions

1. Tiles

```
139 // which exit can the four direction of tile connect - up,right,down,left
140 // highway->1, railway->2, no exit->0
141 static int S0[]={{1,1,2,1},{1,1,1,2},{2,1,1,1},{1,2,1,1},{1,1,2,1},{1,1,1,2},{2,1,1,1},{1,2,1,1}};
142 static int S1[]={{1,2,2,2},{2,1,2,2},{2,2,1,2},{2,2,2,1},{1,2,2,2},{2,1,2,2},{2,2,1,2},{2,2,2,1}};
143 static int S2[]={{1,1,1,1},{1,1,1,1},{1,1,1,1},{1,1,1,1},{1,1,1,1},{1,1,1,1},{1,1,1,1},{1,1,1,1}};
144 static int S3[]={{2,2,2,2},{2,2,2,2},{2,2,2,2},{2,2,2,2},{2,2,2,2},{2,2,2,2},{2,2,2,2},{2,2,2,2}};
145 static int S4[]={{1,2,2,1},{1,1,2,2},{2,1,1,2},{2,2,1,1},{1,2,1,2},{2,1,1,2},{2,2,1,1},{1,2,2,1}};
146 static int S5[]={{1,2,1,2},{2,1,2,1},{1,2,1,2},{2,1,2,1},{1,2,1,2},{2,1,2,1},{1,2,1,2},{2,1,2,1}};
147 static int A0[]={{2,0,0,2},{2,2,0,0},{0,2,2,0},{0,0,2,2},{2,2,0,0},{0,2,2,0},{0,0,2,2},{2,0,0,2}};
148 static int A1[]={{2,0,2,0},{0,2,0,2},{2,0,2,0},{0,2,0,2},{2,0,2,0},{0,2,0,2},{2,0,2,0},{0,2,0,2}};
149 static int A2[]={{2,2,2,0},{0,2,2,2},{2,0,2,2},{2,2,0,2},{2,0,2,2},{2,2,0,2},{2,2,2,0},{0,2,2,2}};
150 static int A3[]={{1,1,1,0},{0,1,1,1},{1,0,1,1},{1,1,0,1},{1,0,1,1},{1,1,0,1},{1,1,1,0},{0,1,1,1}};
151 static int A4[]={{1,0,1,0},{0,1,0,1},{1,0,1,0},{0,1,0,1},{1,0,1,0},{0,1,0,1},{1,0,1,0},{0,1,0,1}};
152 static int A5[]={{1,0,0,1},{1,1,0,0},{0,1,1,0},{0,0,1,1},{1,1,0,0},{0,1,1,0},{0,0,1,1},{1,0,0,1}};
153 static int B0[]={{1,0,2,0},{0,1,0,2},{2,0,1,0},{0,2,0,1},{1,0,2,0},{0,1,0,2},{2,0,1,0},{0,2,0,1}};
154 static int B1[]={{1,2,0,0},{0,1,2,0},{0,0,1,2},{2,0,0,1},{1,0,0,2},{2,1,0,0},{0,2,1,0},{0,0,2,1}};
155 static int B2[]={{1,2,1,2},{2,1,2,1},{1,2,1,2},{2,1,2,1},{1,2,1,2},{2,1,2,1},{1,2,1,2},{2,1,2,1}};
156
157 // To get the connection type
158 // Author: Yuguang Zhai
159 @ public static int getValue(String tilePlacementStringA,int x,int y)
160 {
161     int ans=0;
162     if(tilePlacementStringA.charAt(0)=='S'&&tilePlacementStringA.charAt(1)=='0') ans=S0[x][y];
163     if(tilePlacementStringA.charAt(0)=='S'&&tilePlacementStringA.charAt(1)=='1') ans=S1[x][y];
164     if(tilePlacementStringA.charAt(0)=='S'&&tilePlacementStringA.charAt(1)=='2') ans=S2[x][y];
165     if(tilePlacementStringA.charAt(0)=='S'&&tilePlacementStringA.charAt(1)=='3') ans=S3[x][y];
166     if(tilePlacementStringA.charAt(0)=='S'&&tilePlacementStringA.charAt(1)=='4') ans=S4[x][y];
167     if(tilePlacementStringA.charAt(0)=='S'&&tilePlacementStringA.charAt(1)=='5') ans=S5[x][y];
168     if(tilePlacementStringA.charAt(0)=='A'&&tilePlacementStringA.charAt(1)=='0') ans=A0[x][y];
169     if(tilePlacementStringA.charAt(0)=='A'&&tilePlacementStringA.charAt(1)=='1') ans=A1[x][y];
170     if(tilePlacementStringA.charAt(0)=='A'&&tilePlacementStringA.charAt(1)=='2') ans=A2[x][y];
171     if(tilePlacementStringA.charAt(0)=='A'&&tilePlacementStringA.charAt(1)=='3') ans=A3[x][y];
172     if(tilePlacementStringA.charAt(0)=='A'&&tilePlacementStringA.charAt(1)=='4') ans=A4[x][y];
173     if(tilePlacementStringA.charAt(0)=='A'&&tilePlacementStringA.charAt(1)=='5') ans=A5[x][y];
174     if(tilePlacementStringA.charAt(0)=='B'&&tilePlacementStringA.charAt(1)=='0') ans=B0[x][y];
175     if(tilePlacementStringA.charAt(0)=='B'&&tilePlacementStringA.charAt(1)=='1') ans=B1[x][y];
176     if(tilePlacementStringA.charAt(0)=='B'&&tilePlacementStringA.charAt(1)=='2') ans=B2[x][y];
177     return ans;
178 }
179
```

2. Well Formed Placement

```
22 @ public static boolean isTilePlacementWellFormed(String tilePlacementString) {
23
24     // whether it contains only 5 characters
25     if(tilePlacementString.length()/5 !=1 || tilePlacementString.length()%5 !=0){
26         return false;
27     }
28
29     // whether the first character represents a die A or B, or a special tile S
30     if(tilePlacementString.charAt(0) != 'A' && tilePlacementString.charAt(0) != 'B' &&
31        tilePlacementString.charAt(0) != 'S'){
32         return false;
33     }
34
35     // when the first character is A or S, whether the second character between 0-5
36     if(tilePlacementString.charAt(0) == 'A' && tilePlacementString.charAt(1)>'5'){
37         return false;
38     }
39
40     if(tilePlacementString.charAt(0) == 'S' && tilePlacementString.charAt(1)>'5'){
41         return false;
42     }
43
44     // when the first character is B, whether the second character between 0-2
45     if(tilePlacementString.charAt(0) == 'B' && tilePlacementString.charAt(1)>'2'){
46         return false;
47     }
48
49     // whether the third character is between A and G
50     if(tilePlacementString.charAt(2) > 'G' || tilePlacementString.charAt(2) < 'A'){
51         return false;
52     }
53
54     // whether the fourth character is between 0 and 6
55     if(!(tilePlacementString.charAt(3) == '0' || tilePlacementString.charAt(3) == '1' ||
56         tilePlacementString.charAt(3) == '2' || tilePlacementString.charAt(3) == '3' ||
57         tilePlacementString.charAt(3) == '4' || tilePlacementString.charAt(3) == '5' ||
58         tilePlacementString.charAt(3) == '6')){
59         return false;
60     }
61
62     // whether the fifth character is between 0 and 7
63     if(tilePlacementString.charAt(4) > '7'){
64         return false;
65     }
66     // FIXME Task 2: determine whether a tile placement is well-formed
67     return true;
68 }
```

3. Is Board String Well Formed

```
77 @ public static boolean isBoardStringWellFormed(String boardString) {
78
79     // whether a board string is null or empty
80     if(boardString == null || boardString == ""){
81         return false;
82     }
83
84     // whether a board string has exactly N five-character tile placements, and not too long
85     if(boardString.length()%5 != 0) {
86         return false;
87     }else if(boardString.length()/5 > 31){
88         return false;
89     }
90
91     int count = 0;
92     for(int i=0; i<boardString.length(); i+=5){
93         // slice the board string into a single one
94         String newString = boardString.substring(i,i+4);
95         char margin = boardString.charAt(i+4);
96         newString = newString + margin;
97         // test whether it is well performed
98         if(!isTilePlacementWellFormed(newString)){
99             return false;
100         }
101         // count the number of special pieces
102         if(newString.charAt(0) == 'S'){
103             count++;
104         }
105     }
106
107     // if special pieces is more than 3 then it is not valid
108     if(count > 3){
109         return false;
110     }else{
111         return true;
112     }
113     // FIXME Task 3: determine whether a board string is well-formed
114 }
```

4. Are Connected Neighbors

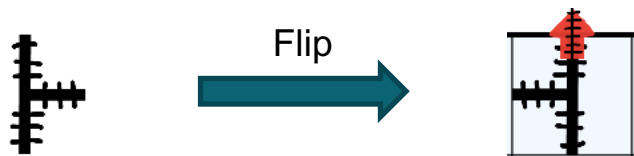
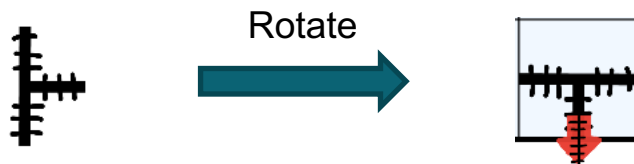
```
243 public static boolean areConnectedNeighbours(String tilePlacementStringA, String tilePlacementStringB) {  
244  
245     // first judge neighbour  
246     if (areNeighbours(tilePlacementStringA, tilePlacementStringB) == -1) {  
247         return false;  
248     }  
249     // If they are neighbours, whether they can successfully connect.  
250     if (tilePlacementStringA.charAt(2) - tilePlacementStringB.charAt(2) == 1) {  
251         if (areConnected(tilePlacementStringA, tilePlacementStringB, indexA: 0, indexB: 2))  
252             return true;  
253     }  
254  
255     if (tilePlacementStringA.charAt(2) - tilePlacementStringB.charAt(2) == -1) {  
256         if (areConnected(tilePlacementStringA, tilePlacementStringB, indexA: 2, indexB: 0))  
257             return true;  
258     }  
259  
260     if (tilePlacementStringA.charAt(3) - tilePlacementStringB.charAt(3) == 1) {  
261         if (areConnected(tilePlacementStringA, tilePlacementStringB, indexA: 3, indexB: 1))  
262             return true;  
263     }  
264  
265     if (tilePlacementStringA.charAt(3) - tilePlacementStringB.charAt(3) == -1) {  
266         if (areConnected(tilePlacementStringA, tilePlacementStringB, indexA: 1, indexB: 3))  
267             return true;  
268     }  
269     // FIXME Task 5: determine whether neighbouring placements are connected  
270     return false;  
271 }
```

5. Get solutions

```
485 public static int getBasicScore(String boardString) {
486     getGroups(boardString);
487
488     int ans = 0;
489     // Number of Exits connected to route | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | 12 |
490     // Points Awarded | 4 | 8 | 12 | 16 | 20 | 24 | 28 | 32 | 36 | 40 | 45 |
491     for(int i=0;i<group.size();i++){
492         int exits=0;
493         int error=0;
494         for(int j=0;j+4<group.get(i).length();j=j+5){
495             String temp=group.get(i).substring(j,j+5);
496             for(int k=0;k<4;k++) {
497                 if(mapNode.get(temp).direction[k]==-1) exits++;
498                 if(mapNode.get(temp).direction[k]==-2){
499                     error++;
500                 }
501             }
502         }
503         if(exits==2) ans+=4;
504         if(exits==3) ans+=8;
505         if(exits==4) ans+=12;
506         if(exits==5) ans+=16;
507         if(exits==6) ans+=20;
508         if(exits==7) ans+=24;
509         if(exits==8) ans+=28;
510         if(exits==9) ans+=32;
511         if(exits==10) ans+=36;
512         if(exits==11) ans+=40;
513         if(exits==12) ans+=45;
514
515         ans-=error;
516     }
517     //centre grid that are covered
518     ans += getCentreGrids(boardString);
519     // FIXME Task 8: compute the basic score
520     return ans;
521 }
```




Rotating and flipping



Solutions

StepsGame Viewer

Well done!

Tiles

Special Tiles

SCORE

2

Placement:

New Game Refresh Generate VS Computer AI Move

Single Player



StepsGame Viewer

A 7x7 grid with a 3x3 area in the center highlighted in light red. Red arrows are placed on the perimeter of the grid, pointing outwards from the edges. The arrows are located at the following positions: (1,2), (1,4), (1,6), (2,1), (2,7), (3,1), (3,7), (4,1), (4,7), (5,1), (5,7), (6,1), (6,7), (7,2), (7,4), (7,6).

Special Tiles

A vertical list of seven special tile patterns, each consisting of a cross shape with various extensions and dashed lines.

Placement: New Game Refresh Generate VS Computer

Multi-player

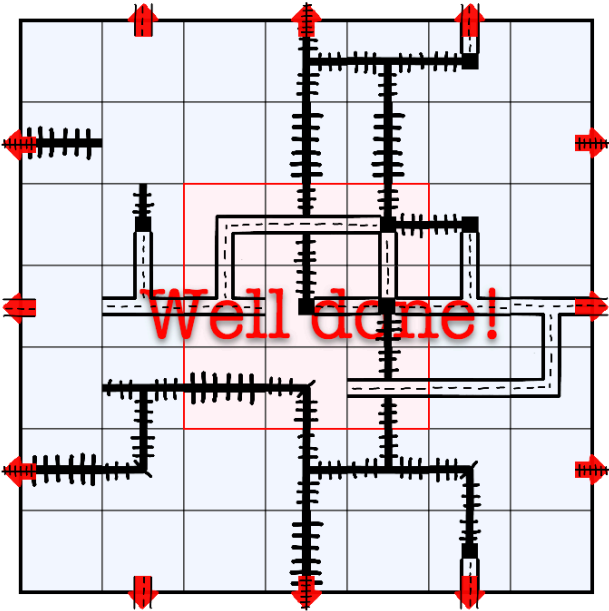
The screenshot shows a Java IDE with the following components:

- Project Explorer:** Shows a project named 'RailroadLink.java' with a 'Viewer' sub-project.
- Code Editor:** Contains Java code for the 'StepsGame Viewer' and 'Computer Player' classes. The code includes comments and method definitions for setting up the game grid and handling moves.
- StepsGame Viewer:** A 10x10 grid with a red 3x3 area in the center. It features various black and red tiles, including straight, L-shaped, and T-shaped tiles, and red arrows indicating movement directions.
- Computer Player:** A similar 10x10 grid with a red 3x3 area in the center, showing the state of the game as played by the computer.
- Run Console:** Shows the output of the program, including the path of the Java executable and the version of the Java runtime.



Interesting part

StepsGame Viewer



Well done!

Tiles

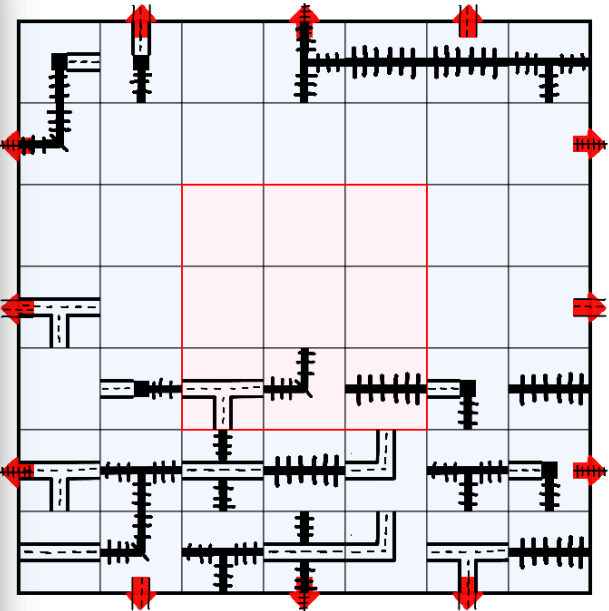
Special Tiles

SCORE

16

Placement: New Game Refresh Generate VS Computer AI Move

Computer Player



SCORE

-494



Thank you

Any questions?