```java
package clueGame;

import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.util.HashMap;
import java.util.HashSet;
import java.util.Map;
import java.util.Scanner;
import java.util.Set;

import clueGame.BoardCell;

public class Board
{
  private int numRows;
  private int numColumns;
  public int MAX_BOARD_SIZE = 50;
  private BoardCell board[][];
  private Map<Character, String> legend = new HashMap<Character, String>();
  private String legendConfig;
  private Map<BoardCell, Set<BoardCell>> adjMatrix = new HashMap<BoardCell, Set<BoardCell>>();
  private Set<BoardCell> targets = new HashSet<BoardCell>();
  private Set<BoardCell> visited = new HashSet<BoardCell>();

  private String boardConfigFile;
  private static Board theInstance = new Board();
  boolean firstRun = true;

  private Board() {}
  // this method returns the only Board
  public static Board getInstance()
  {
    return theInstance;
  }

  public void initialize()
  {
    try {
      loadRoomConfig();
    } catch (BadConfigFormatException e) {
      e.printStackTrace();
    }


    try {
      loadBoardConfig();
    } catch (BadConfigFormatException e) {
      e.printStackTrace();
    }
    calcAdjacencies();




  }
  public void loadRoomConfig() throws BadConfigFormatException
  {
    try {
      File file = new File(legendConfig);
```

```
60        Scanner scanner = new Scanner(file);
61
62        while(scanner.hasNextLine())
63        {
64          String line = scanner.nextLine();
65          String[] elements = line.split(",");
66
67          Character tempChar = elements[0].charAt(0);
68          String tempRoomName = elements[1];
69          String tempType = elements[2].trim();
70
71          if (!tempType.contentEquals("Card"))
72          {
73            if (!tempType.contentEquals("Other"))
74            {
75              throw new BadConfigFormatException();
76
77            }
78          }
79
80          legend.put(tempChar, tempRoomName.trim());
81
82        }
83        scanner.close();
84
85      } catch (FileNotFoundException e)
86      {
87        e.printStackTrace();
88
89      }
90    }
91    public void loadBoardConfig() throws BadConfigFormatException
92    {
93      Character walkwayKey = 'w';
94
95      try {
96        File file = new File(boardConfigFile);
97        Scanner scanner = new Scanner(file);
98
99        for (Map.Entry<Character, String> entry : legend.entrySet())
100       {
101         if(entry.getValue().equals("Walkway"))
102         {
103           walkwayKey = entry.getKey();
104         }
105       }
106
107       //Get the dimensions (this is a dumb way of doing this but quick
108       int oldColumns = 0;
109       boolean notFirstRun = false;
110       while (scanner.hasNextLine())
111       {
112         numRows++;
113         String line = scanner.nextLine();
114         String[] elements = line.split(",");
115         numColumns = elements.length;
116         if (numColumns != oldColumns && notFirstRun)
117         {
118           throw new BadConfigFormatException();
119         }
```

```
120         notFirstRun = true;
121
122       }
123       scanner.close();
124       scanner = new Scanner(file);
125       board = new BoardCell[numRows][numColumns];
126       int row = 0;
127
128       while(scanner.hasNextLine())
129       {
130         String line = scanner.nextLine();
131         String[] elements = line.split(",");
132         int column = 0;
133         while (column < elements.length)
134         {
135           BoardCell tempCell = new BoardCell(row, column);
136           tempCell.initial = elements[column].charAt(0);
137
138           if (elements[column].length() > 1)
139           {
140             // Multi Character spot (door)
141             tempCell.doorway = true;
142             switch (elements[column].charAt(1))
143             {
144             case 'U':
145               tempCell.doorDirection = DoorDirection.UP;
146               break;
147             case 'D':
148               tempCell.doorDirection = DoorDirection.DOWN;
149               break;
150             case 'L':
151               tempCell.doorDirection = DoorDirection.LEFT;
152               break;
153             case 'R':
154               tempCell.doorDirection = DoorDirection.RIGHT;
155               break;
156             case 'N':
157               tempCell.doorDirection = DoorDirection.NONE;
158               break;
159             };
160
161           }
162
163           else if(tempCell.initial == walkwayKey)
164           {
165             tempCell.walkway = true;
166           }
167           else
168           {
169             tempCell.room = true;
170           }
171
172           board[row][column] = tempCell;
173           column++;
174         }
175         row++;
176
177       }
178       scanner.close();
179
```

```
180     } catch (FileNotFoundException e)
181     {
182
183       e.printStackTrace();
184     }
185
186   }
187   public void calcAdjacencies()
188   {
189   //  System.out.println(board.length + " " + board[0].length);
190     for (int i = 0; i < board.length; i++)
191     {
192       for (int j = 0; j < board[0].length; j++)
193       {
194   //      System.out.print(board[i][j].initial + " ");
195         //BoardCell keyCell = getCell(grid[i][j]);
196         Set<BoardCell> adjacencies = new HashSet<BoardCell>();
197         System.out.println("GRID: " + board[i][j].row + " " + board[i]
[j].column);
198
199
200
201         if (board[i][j].isDoorway())
202         {
203           if (board[i][j].doorDirection == DoorDirection.UP)
204           {
205             adjacencies.add(board[i -1][j]);
206
207           }
208
209
210           else if (board[i][j].doorDirection == DoorDirection.LEFT)
211           {
212             adjacencies.add(board[i][j-1]);
213
214           }
215           else if (board[i][j].doorDirection == DoorDirection.RIGHT)
216           {
217             adjacencies.add(board[i][j+1]);
218           }
219           else if (board[i][j].doorDirection == DoorDirection.DOWN)
220           {
221             adjacencies.add(board[i + 1][j]);
222
223           }
224           adjMatrix.put(board[i][j], adjacencies);
225           for (BoardCell adjacent : adjacencies)
226           {
227             System.out.print("Adj: " + adjacent.row + " " + adjacent.column +
" ");
228
229
230           }
231           System.out.println();
232         }
233         ////////////////////////////////////Not
Doorway//////////////////////////////////////////////
234         else if(!board[i][j].isRoom())
235         {
236
```

```
237
238
239           if (board[i][j].row != 0)
240           {
241             //BoardCell tempCell =  getCell(grid[i][j].column -1, grid[i]
    [j].row)
242             //Handle the in room adjacencies ignore
243             if (board[i-1][j].isDoorway())
244             {
245               // Need to check that its enetered from correct direction
246               if (board[i-1][j].doorDirection == DoorDirection.DOWN)
247               {
248                 adjacencies.add(board[i -1][j]);
249
250               }
251             }
252
253             else if (!board[i-1][j].isRoom())
254             {
255               adjacencies.add(board[i -1][j]);
256             }
257           }
258
259           if (board[i][j].column != 0)
260           {
261
262             if (board[i][j-1].isDoorway())
263             {
264               if (board[i][j-1].doorDirection == DoorDirection.RIGHT)
265               {
266                 adjacencies.add(board[i][j-1]);
267
268               }
269             }
270             else if (!board[i][j-1].isRoom())
271             {
272               adjacencies.add(board[i][j-1]);
273             }
274
275           }
276
277           if (board[i][j].column != board[0].length -1)
278           {
279
280             if (board[i][j+1].isDoorway())
281             {
282               if (board[i][j+1].doorDirection == DoorDirection.LEFT)
283               {
284                 adjacencies.add(board[i][j+1]);
285
286               }
287             }
288             else if(!board[i][j+1].isRoom())
289             {
290               adjacencies.add(board[i][j+1]);
291             }
292           }
293
294           if (board[i][j].row != board.length -1)
295           {
```

```
296             if (board[i+1][j].isDoorway())
297             {
298               if ((board[i+1][j].doorDirection == DoorDirection.UP))
299               {
300                 adjacencies.add(board[i + 1][j]);
301
302               }
303             }
304             else if (!board[i+1][j].isRoom())
305             {
306               adjacencies.add(board[i + 1][j]);
307             }
308           }
309           adjMatrix.put(board[i][j], adjacencies);
310           for (BoardCell adjacent : adjacencies)
311           {
312             System.out.print("Adj: " + adjacent.row + " " + adjacent.column +
    " ");
313
314
315           }
316           System.out.println();
317         }
318
319
320       }
321     }
322   }
323
324
325
326   public void calcTargets(int row, int column, int pathLength)
327   {
328     if (firstRun)
329     {
330       targets.clear();
331       visited.clear();
332       firstRun = false;
333
334     }
335     visited.add(board[row][column]);
336     Set<BoardCell> adjCells = getAdjList(row, column);
337
338     for(BoardCell eachCell : adjCells) {
339
340       if(!visited.contains(eachCell))
341       {
342
343         visited.add(eachCell);
344         if(pathLength == 1 || eachCell.isDoorway())
345         {
346           targets.add(eachCell);
347         }
348         else
349         {
350           calcTargets(eachCell.row, eachCell.column, pathLength -1);
351         }
352         visited.remove(eachCell);
353       }
354
```

```
355     }
356 //    Leave me for future debugging
357 //    System.out.print("Cell Path "+ pathLength +" " + row + " " + column +
    "| " + "Lenght " + targets.size());
358 //    for (BoardCell target : targets)
359 //    {
360 //      System.out.print("( " + target.row + " " + target.column + " )");
361 //
362 //    }
363 //    System.out.println();
364
365   }
366   public Set<BoardCell> getAdjList(int row, int column)
367   {
368     Set<BoardCell> adjCells = adjMatrix.get(board[row][column]);
369     if (adjCells == null)
370     {
371       return new HashSet<BoardCell>();
372     }
373     return adjCells;
374   }
375   public Set<BoardCell> getTargets()
376   {
377     firstRun = true;
378     return targets;
379   }
380
381   public void setConfigFiles(String boardConfig, String legendConfig)
382   {
383     this.boardConfigFile = boardConfig;
384     this.legendConfig = legendConfig;
385
386   }
387   public Map<Character, String> getLegend()
388   {
389     return legend;
390   }
391   public BoardCell getCellAt(int row, int column)
392   {
393     return board[row][column];
394   }
395
396   public int getNumRows()
397   {
398     return numRows;
399   }
400   public int getNumColumns()
401   {
402     return numColumns;
403   }
404
405 }
406
```