```java
/*
 * Written by JJ Shepherd
 */
import java.util.Random;
public class MazeGame {
    public static final int MAZE_SIZE = 10;
    public static final int OBST_AMT = (MAZE_SIZE*MAZE_SIZE)/10;
    private char[][] maze;//index 0 = y, index 1 = x
    public static final char EMPTY = '_';
    public static final char OBST = 'X';
    public static final char PLAYER = 'O';
    public static final char PATH = '#';

    public static final String NORTH = "Go North";
    public static final String SOUTH = "Go South";
    public static final String WEST = "Go West";
    public static final String EAST = "Go East";
    public static final String BACK = "Go Back";

    private StackI<int[]> locations;
    private int[] currLoc;//index 0 = y, index 1 = x

    public MazeGame()
    {
        this.init();
    }
    private void init()
    {
        maze = new char[MAZE_SIZE][MAZE_SIZE];
        for(int i=0;i<maze.length;i++)
            for(int j=0;j<maze[i].length;j++)
                maze[i][j] = EMPTY;
        this.addObstacles();

        maze[0][0] = PLAYER;
        currLoc = new int[] {0,0};
        locations = new LLStack<int[]>();
    }
    private void addObstacles()
    {
        Random r = new Random();
        for(int i=0;i<OBST_AMT;i++)
        {
            int x = r.nextInt(MAZE_SIZE);
            int y = r.nextInt(MAZE_SIZE);
            if(maze[y][x] != EMPTY || (x==0 &&y==0))
                continue;
            maze[y][x] = OBST;
        }
    }
    public void printMoveOptions()
    {
        int currY = currLoc[0];
        int currX = currLoc[1];
        int[] prevLoc = locations.peek();
        //North
        if(isValid(currY-1) && maze[currY-1][currX] != OBST)
```

```
58          {
59              if(prevLoc != null && currY-1 == prevLoc[0] && currX == prevLoc[1])
60                  System.out.println(BACK);
61              else
62                  System.out.println(NORTH);
63          }
64          //South
65          if(isValid(currY+1) && maze[currY+1][currX] != OBST)
66          {
67              if(prevLoc != null && currY+1 == prevLoc[0] && currX == prevLoc[1])
68                  System.out.println(BACK);
69              else
70                  System.out.println(SOUTH);
71          }
72          //West
73          if(isValid(currX-1) && maze[currY][currX-1] != OBST)
74          {
75              if(prevLoc != null && currY == prevLoc[0] && currX-1 == prevLoc[1])
76                  System.out.println(BACK);
77              else
78                  System.out.println(WEST);
79          }
80          //East
81          if(isValid(currX+1) && maze[currY][currX+1] != OBST)
82          {
83              if(prevLoc != null && currY == prevLoc[0] && currX+1 == prevLoc[1])
84                  System.out.println(BACK);
85              else
86                  System.out.println(EAST);
87          }
88      }
89      public void move(String input)
90      {
91          maze[currLoc[0]][currLoc[1]] = EMPTY;
92          int currY = currLoc[0];
93          int currX = currLoc[1];
94          int[] copyLoc = {currLoc[0],currLoc[1]};
95          if(input.equalsIgnoreCase(NORTH))
96          {
97              if(isValid(currY-1) && maze[currY-1][currX] != OBST)
98              {
99                  locations.push(copyLoc);
100                 currLoc[0]--;
101             }
102             else
103             {
104                 System.out.println("Invalid Move");
105             }
106         }
107         else if(input.equalsIgnoreCase(SOUTH))
108         {
109             if(isValid(currY+1) && maze[currY+1][currX] != OBST)
110             {
111                 locations.push(copyLoc);
112                 currLoc[0]++;
113             }
114             else
```

```
115             {
116                 System.out.println("Invalid Move");
117             }
118         }
119         else if(input.equalsIgnoreCase(WEST))
120         {
121             if(isValid(currX-1) && maze[currY][currX-1] != OBST)
122             {
123                 locations.push(copyLoc);
124                 currLoc[1]--;
125             }
126             else
127             {
128                 System.out.println("Invalid Move");
129             }
130         }
131         else if(input.equalsIgnoreCase(EAST))
132         {
133             if(isValid(currX+1) && maze[currY][currX+1] != OBST)
134             {
135                 locations.push(copyLoc);
136                 currLoc[1]++;
137             }
138             else
139             {
140                 System.out.println("Invalid Move");
141             }
142         }
143         else if(input.equalsIgnoreCase(BACK))
144         {
145             if(locations.peek() != null)
146             {
147                 int[] prevLoc = locations.pop();
148                 currLoc[0] = prevLoc[0];
149                 currLoc[1] = prevLoc[1];
150             }
151             else
152             {
153                 System.out.println("Invalid Move");
154             }
155         }
156         else
157         {
158             System.out.println("Invalid Move");
159         }
160         maze[currLoc[0]][currLoc[1]] = PLAYER;
161     }
162     private boolean isValid(int index)
163     {
164         return index >= 0 && index < maze.length;
165     }
166     public boolean getWin()
167     {
168         return currLoc[0] == maze.length-1 && currLoc[1] == maze.length-1;
169     }
170     public void reset()
171     {
```

```
172         this.init();
173     }
174     public void printFullMaze()
175     {
176         for(int i=0;i<maze.length;i++)
177         {
178             for(int j=0;j<maze.length;j++)
179             {
180                 System.out.print(maze[i][j]);
181             }
182             System.out.println();
183         }
184     }
185     public void printFullMazeWithPath()
186     {
187         while(locations.peek() != null)
188         {
189             int[] temp = locations.pop();
190             maze[temp[0]][temp[1]] = PATH;
191         }
192         this.printFullMaze();
193     }
194 }
195
```