```cpp
//MazeSolver.cpp

#include "MazeSolver.h"
#include <iostream>
#include <fstream>
#include <stdio.h>
#include <string>

using namespace std;

//default constructor
MazeSolver::MazeSolver() {
    setMaze("Not Found");
}


MazeSolver::MazeSolver(string maze_data) {
    setMaze(maze_data);
}


//Gets the data from the txt file.
void MazeSolver::setMaze(string maze_data) {

    if (maze_data == "Not Found") {
        cout << "Please input valid txt file" << endl;

        return;
    }

    //open the file
    ifstream inFile(maze_data);
    string line;

    if (!inFile) {
        cout << "Could not open file" << endl;
        return;
    }

    inFile >> col;
    inFile >> row;

    //initialize the vector with the row and col size that is included in the file and set all data to char a
    vect.resize(row, vector<char>(col, 'a'));

    inFile.ignore();

    //count_row is used to locate the row value of mario
```

```cpp
    int count_row = 0;
    while (getline(inFile, line)) {

        //the loop is used to set each vector data to the character that is in the file
        //the string is placed in the variable line and then iterate through the string to get the character
        for (int i = 0; i < line.length(); i++) {

            vect[count_row][i] = line[i];

            //This is used to get the coordinates of mario and save it to his own variable
            if (line[i] == 'M') {
                mario_row = count_row;
                mario_col = i;
            }

            //This is used to get the coordinates of peach and save it to her own variable
            if (line[i] == 'P') {
                peach_row = count_row;
                peach_col = i;
            }
        }
        count_row++;
    }

    //print();
}

void MazeSolver::goNorth(int curr_row, int curr_col, bool& rescue) {

    //going up or down depends on the row value, to go up, you have to decrement row.
    //the conditional statement makes sures that the current position is not the border.
    if ((curr_row - 1) >= 0) {

        //the conditional statement checks if going up is peach, so we should set it to true
        if (vect[curr_row - 1][curr_col] == 'P' && rescue == false) {
            rescue = true;
        }

        //if it is not peach, we should move.
        else if (vect[curr_row - 1][curr_col] == ' ' && rescue == false) {
            curr_row = curr_row - 1;
            vect[curr_row][curr_col] = '?';
```

```
 91
 92             //going North is the initial movement, then east, then west.
 93             goNorth(curr_row, curr_col, rescue);
 94             if (rescue == false) {
 95                 goEast(curr_row, curr_col, rescue);
 96             }
 97             if (rescue == false) {
 98                 goWest(curr_row, curr_col, rescue);
 99             }
100             if (rescue == false) {
101                 //Once North is done, we cannot move south without going
                      back to the position that initiated this so set it to
                      '*' before going south
102                 vect[curr_row][curr_col] = '*';
103                 goSouth(curr_row, curr_col, rescue);
104             }
105         }
106     }
107     else {
108         rescue = false;
109     }
110 }
111
112 void MazeSolver::goSouth(int curr_row, int curr_col, bool& rescue) {
113
114     //going up or down depends on the row value, to go up, you have to
        increment row.
115     //the conditional statement makes sures that the current position is
        not the border.
116     if ((curr_row + 1) < row) {
117
118         //the conditional statement checks if going down is peach, so we
            should set it to true
119         if (vect[curr_row + 1][curr_col] == 'P' && rescue == false) {
120             rescue = true;
121         }
122
123         //if it is not peach, we should move.
124         else if (vect[curr_row + 1][curr_col] == ' ' && rescue == false) {
125             curr_row = curr_row + 1;
126             vect[curr_row][curr_col] = '?';
127
128             //going South is the initial movement, then east, then west.
129             goSouth(curr_row, curr_col, rescue);
130             if (rescue == false) {
131                 goEast(curr_row, curr_col, rescue);
132             }
133             if (rescue == false) {
134                 goWest(curr_row, curr_col, rescue);
```

```
135             }
136             if (rescue == false) {
137 
138                 //Once South is done, we cannot move north without going back to the position that initiated this so set it to '*' before going north
139                 vect[curr_row][curr_col] = '*';
140                 goNorth(curr_row, curr_col, rescue);
141             }
142         }
143     }
144     else {
145         rescue = false;
146     }
147 }
148 
149 void MazeSolver::goWest(int curr_row, int curr_col, bool& rescue) {
150 
151     //going left or right depends on the col value, to go left, you have to decrement col.
152     //the conditional statement makes sures that the current position is not the border.
153     if ((curr_col - 1) >= 0) {
154 
155         //the conditional statement checks if going left is peach, so we should set it to true
156         if (vect[curr_row][curr_col - 1] == 'P' && rescue == false) {
157             rescue = true;
158         }
159 
160         //if it is not peach, we should move.
161         else if (vect[curr_row][curr_col - 1] == ' ' && rescue == false) {
162             curr_col = curr_col - 1;
163             vect[curr_row][curr_col] = '?';
164 
165             //going West is the initial movement, then south, then north.
166             goWest(curr_row, curr_col, rescue);
167             if (rescue == false) {
168                 goSouth(curr_row, curr_col, rescue);
169             }
170             if (rescue == false) {
171                 goNorth(curr_row, curr_col, rescue);
172             }
173             if (rescue == false) {
174 
175                 //Once West is done, we cannot move east without going back to the position that initiated this so set it to '*' before going east
176                 vect[curr_row][curr_col] = '*';
```

```
177                 goEast(curr_row, curr_col, rescue);
178             }
179         }
180     }
181     else {
182         rescue = false;
183     }
184 }
185
186 void MazeSolver::goEast(int curr_row, int curr_col, bool& rescue) {
187
188     //going left or right depends on the col value, to go left, you have to increment col.
189     //the conditional statement makes sures that the current position is not the border.
190     if ((curr_col + 1) < col) {
191
192         //the conditional statement checks if going right is peach, so we should set it to true
193         if (vect[curr_row][curr_col + 1] == 'P' && rescue == false) {
194             rescue = true;
195         }
196
197         //if it is not peach, we should move.
198         else if (vect[curr_row][curr_col + 1] == ' ' && rescue == false) {
199             curr_col = curr_col + 1;
200             vect[curr_row][curr_col] = '?';
201
202             //going East is the initial movement, then south, then north.
203             goEast(curr_row, curr_col, rescue);
204             if (rescue == false) {
205                 goSouth(curr_row, curr_col, rescue);
206             }
207             if (rescue == false) {
208                 goNorth(curr_row, curr_col, rescue);
209             }
210             if (rescue == false) {
211
212                 //Once East is done, we cannot move west without going back to the position that initiated this so set it to '*' before going west
213                 vect[curr_row][curr_col] = '*';
214                 goWest(curr_row, curr_col, rescue);
215             }
216         }
217     }
218     else {
219         rescue = false;
220     }
```

```
221 }
222
223 void MazeSolver::searchForPath() {
224     bool rescued = false;
225     int init_mario_row = mario_row, init_mario_col = mario_col;
226
227     //Initialize a movement for mario for each scenario.
228     goNorth(mario_row, mario_col, rescued);
229     goSouth(mario_row, mario_col, rescued);
230     goEast(mario_row, mario_col, rescued);
231     goWest(mario_row, mario_col, rescued);
232
233     print();
234
235     if (rescued) {
236         cout << "Yay! The Princess Has Been Rescued! \\(o^.^)/" << endl;
237     }
238     else {
239         cout << "No Path Found To Rescue The Princess! (T^T)\\" << endl;
240     }
241 }
242
243
244 void MazeSolver::print() {
245
246     //prints the maze
247     for (int i = 0; i < vect.size(); i++)
248     {
249         for (int j = 0; j < vect[i].size(); j++)
250         {
251             cout << vect[i][j];
252         }
253         cout << endl;
254
255     }
256
257 }
```