# Contents - Debugging lab


**1 Setup** **2**
- 1.1 Turn-in instructions . . . 2
- 1.2 Setup . . . 2
  - 1.2.1 Using the starter code . . . 2
  - 1.2.2 About the starter code . . . 2

**2 Debugging lab** **4**
- 2.1 Task 1: Breakpoints and stepping over lines of code . . . 5
- 2.2 Step into a function and the call stack . . . 8
- 2.3 Step Over / Next Line with preprocessors . . . 11

**3 Additional handy commands** **13**

**4 Appendix: Starter code** **15**
- 4.1 main.cpp . . . 15
- 4.2 Game.hpp . . . 15
- 4.3 Game.cpp . . . 16
- 4.4 Map.hpp . . . 19
- 4.5 Map.cpp . . . 20
- 4.6 Utilities.hpp . . . 24
- 4.7 Utilities.cpp . . . 24

# Part 1

# Setup

## 1.1 Turn-in instructions

- For this lab, you won't be uploading any source code. You will be writing in a text file. Upload the .txt, .rtf, .docx, .odt, or .pdf file of your work.

## 1.2 Setup

### 1.2.1 Using the starter code

Some **starter code** is provided for you for this lab. You can download the starter file off the Canvas assignment, or type in the starter code (next section).

**Project setup:**
The starter code zip file contains all the files as well as the **Visual Studio project files** and the **Code::Blocks project files**.

### 1.2.2 About the starter code

This program contains the following files:

| | |
|---|---|
| main.cpp | Contains main() |
| Game.hpp<br>Game.cpp | Contains the game's loop and handles input. |
| Map.hpp<br>Map.cpp | Handles the location of rooms, the player, treasure, and enemy. |
| Utilities.hpp<br>Utilities.cpp | Utilities for getting user input and clearing the screen. |

# Part 2

# Debugging lab

```
TTTTT  RRR    EEEE   ZZZZZ   OOOO   RRR     OOOO
  T    R  R   E         Z    O  O   R  R    O  O
  T    RRR    EEEE     ZZ    O  O   RRR     O  O
  T    R  R   E       Z       O  O   R  R    O  O
  T    R   R  EEEE   ZZZZZ   OOOO   R   R   OOOO
-----------------------------------------
1. Play
2. Instructions
3. Quit
>>
```

The game is already complete and you will be using this project to practice using your IDE's various debugging tools. Instead of working on any code for this project, you will open a text document and log the information you find for each of the steps of this lab. Upload that text document to Canvas once done.

**But what if I'm not using Code::Blocks or Visual Studio?**
Any IDEs will have these same debugging tools, unless you're writing C++ in a text editor. Do a web search to look for how to use these features in your IDE.

**Make sure that you're running the program in debug mode!**

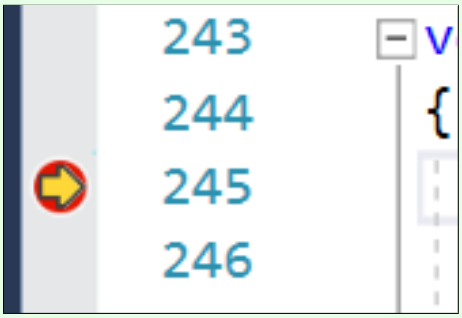
In Visual Studio, you will go to **Debug → Start Debugging (F5)**, and in Code::Blocks, go to **Debug → Start/Continue (F8)**, the button with the *red* arrow, not green.

## 2.1 Task 1: Breakpoints and stepping over lines of code

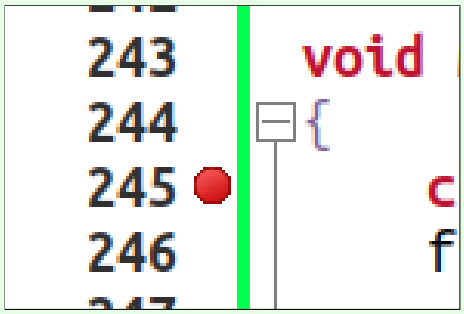
Set a breakpoint in file Map.cpp on line 132, the first line of the Map::MovePlayer function.

**How to set a breakpoint?**
In most IDEs there's a little region to the left of the code you can click on to set a breakpoint.

Visual Studio




Code::Blocks




You can also click on a line of code, and go to **Debug** > **Toggle breakpoint** in Visual Studio and Code::Blocks.

```
1 void Map::MovePlayer( string direction )
2 {    // < Breakpoint here
3      char firstLetter = direction[0];
4      firstLetter = tolower( firstLetter );
```

Then, run the program in debug mode (**Debug** → **Start Debugging (F5)** in Visual Studio, and **Debug** → **Start/Continue (F8)** in Code::Blocks). In the game, select "1" to start a new game.

A randomly generated dungeon will be created and there are three characters on the map: `@` is you, `$` is treasure, and `&` is a goblin.

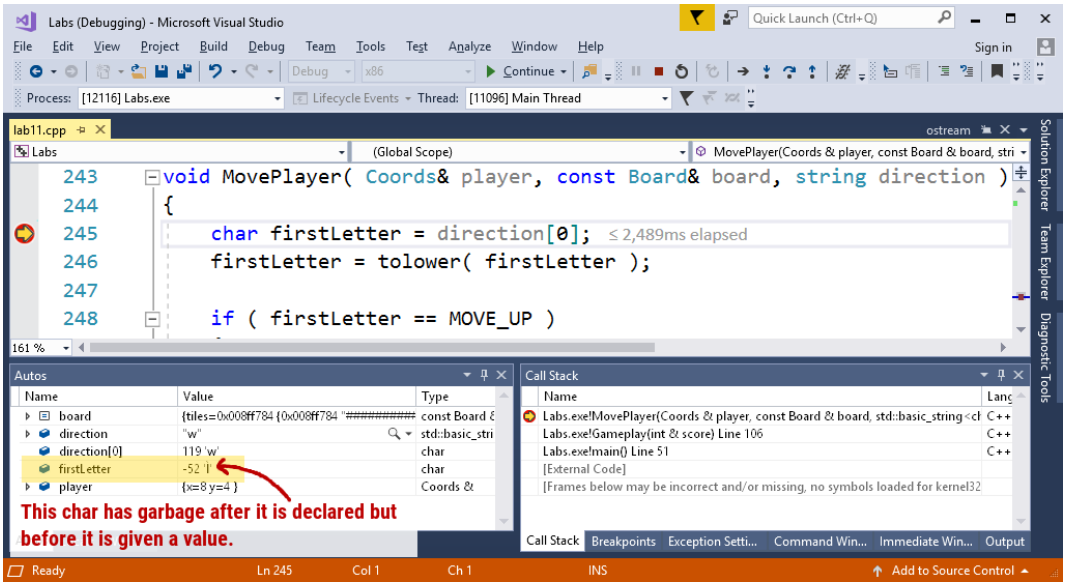
To move, type in a direction (WASD) and hit enter and thent he game screen will update with your new position.

**Is there an easier keypress command in C++?**
Not that I know of, without using a library to extend terminal programming functionality, like ncurses.

After you hit the ENTER button, your IDE will pause execution as it enters the `Map::MovePlayer` function. Leave your console window open and go to the IDE.

There should be a debug pane open that shows you local variables and their values, but if not you can open them up yourself.

**Visual Studio:** Go to **Debug** → **Windows** → and select **Autos** or **Locals**. Locals will show local function variables, and autos will automatically select variables in the current context to display.




**Code::Blocks:** Go to **Debug** → **Debugging windows** → **Watches** to make the Watches window appear.

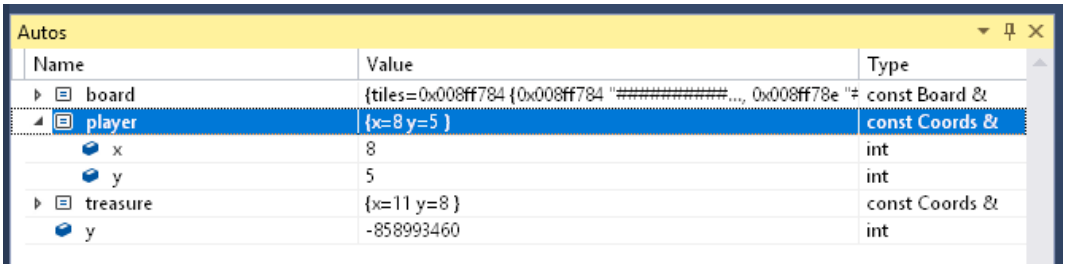
Locate the following variables and write in their values in your text document...

**Log 1:**

| Paused line | Variable | Value |
|---|---|---|
| 131 or 132 | `direction` | |
| 131 or 132 | `firstLetter` | |
| 131 or 132 | `m_player.x` | |
| 131 or 132 | `m_player.y` | |

If one of these variables doesn't show up (particularly in Code::Blocks), you can select one of the blank lines and type in the variable's name, and then it will begin keeping track of that variable's value.

The `m_player` variable is a struct object, so you can also expand this variable in the watches window to view its internal variables, x and y:




Expanding the player struct in Visual Studio


Expanding the player struct in Code::Blocks

(Note: These images are from a previous version of the game.
The variable is now `m_player`.)

Once you've recorded those values, you will use a command to step through the code, line-by-line, and watch the variables change their values.

**Go to the next line of code**
You can step through each line of code with one of the debug step commands. There are shortcuts in the toolbars, or you can access them through the **Debug** menu.

**Visual Studio:** Go to **Debug** → **Step Over (F10)**, or click on this button in the toolbar: 

**Code::Blocks:** Go to **Debug** → **Next line (F7)**, or click on this button in the toolbar: 

Watch the variable values for each line as you step to to line 164 in the function, the line that calls `MoveGoblin`. Before proceeding, log the updated values for the variables.

**Log 2:**

| Paused line | Variable | Value |
|---|---|---|
| 164 | `direction` | |
| 164 | `firstLetter` | |
| 164 | `m_player.x` | |
| 164 | `m_player.y` | |

## 2.2 Step into a function and the call stack

While still paused at line 164, we are going to use the **Step Into** debug feature to go inside this function call. If we used Step Over or Next Line, it would just hit the end of the MovePlayer function and go back to its caller.

Instead, make sure you're on the line calling `MoveGoblin();` and then use Step Into (see next page).

**Step Into a function call**
You can step through each line of code with one of the debug step commands. There are shortcuts in the toolbars, or you can access them through the **Debug** menu.

**Visual Studio:** Go to **Debug** → **Step Into (F11)**, or click on this button in the toolbar: 

**Code::Blocks:** Go to **Debug** → **Next into (Shift+F7)**, or click on this button in the toolbar: 

This function has some different local variables. At the start of the function, the variables have not been initialized and will be storing some sort of garbage values.

**Log 3:**

| Paused line | Variable | Value |
|---|---|---|
| 169 | `x` | |
| 169 | `y` | |
| 169 | `direction` | |
| 169 | `m_enemy.x` | |
| 169 | `m_enemy.y` | |
| 169 | `m_tiles[x][y]` | |

Use the **Step Over / Next Line** command to step down to line 195, looking at the variables update each time. Then log the variables at the end of the function.

**Log 4:**

| Paused line | Variable | Value |
|---|---|---|
| 195 | `x` | |
| 195 | `y` | |
| 195 | `direction` | |
| 195 | `m_enemy.x` | |
| 195 | `m_enemy.y` | |
| 195 | `m_tiles[x][y]` | |

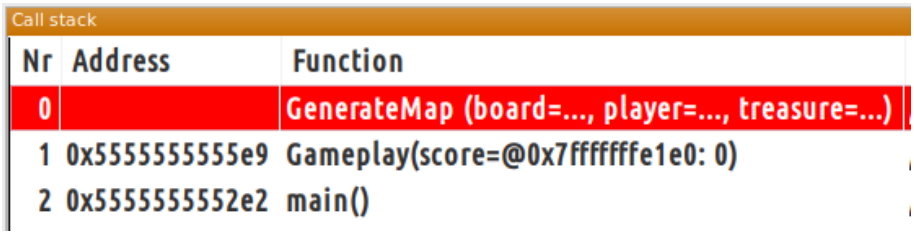
Before we leave this function, we also want to look at the **Call Stack**. The Call Stack shows you a list of all the functions that have been called leading up to where you're currently at (e.g., main() called Game.Run(), which called this or that...)

If you don't see a Call Stack pane open, you will need to enable it through the debug menu.

**Visual Studio:** Go to **Debug** → **Windows** → **Call Stack**.


(Note that the screenshots are displaying info about the previous version of the game.)

**Code::Blocks:** Go to **Debug** → **Debugging windows** → **Call Stack**.



**Log 5:** Write down all of the functions listed in the Call Stack from the top down. Stop at the line that shows **main()**; anything after that isn't really important to us.

Any function listed was called prior to any of the functions above it, so for instance, `main()` started the program. At line 6 of main.cpp, the `main()` function called the `Game::Run()` function.

Double-click on each line in the Call Stack to see where the function was called from.

| | Function... | Was called by... | At line... |
|---|---|---|---|
| 1. | `Map::MoveGoblin()` | | 164 |
| 2. | | | |
| 3. | | | |
| 4. | `Game::Run()` | `main()` | 6 |
| 5. | `main()` | n/a | n/a |

At this point, we are done with this set of functions. Clear your current breakpoints and stop the game for now.

## 2.3 Step Over / Next Line with preprocessors

Go into the Utilities.cpp file and set another breakpoint at line 7, at the beginning of the `ClearScreen()` function.

Run the program **in debug mode** again. One of the first things it does is call the ClearScreen() function. This function has some special **preprocessor** code to determine what kind of operating system you're on, and it calls the correct "clear" command based on that.

```
1  void ClearScreen()
2  {
3      #if defined(WIN32) || defined(_WIN32) || defined(__WIN32) && !defined(
       __CYGWIN__)
4      // The terminal clear command on Windows is "cls"
5      system( "cls" );
6      #else
7      // The terminal clear command on Linux/Mac/Unix is "clear"
8      system( "clear" );
9      #endif
10 }
```

**Log 6:** Identify whether the program executes the `system( "cls" );` or `system( "clear" );` by using your breakpoint and the Step Over / Next Line command.

**System calls**

The `system()` function allows you to execute terminal commands. In Windows, this would be what you might think of as DOS (I'm not sure if they still call it that anymore), or in Linux or Mac it is terminal commands. You could also run these:

| | |
|---|---|
| `ls` in Linux/Mac, `dir` in Windows | Display all the folders/files in the directory. |
| `mkdir NAME` | Creates a folder called NAME from the current working directory. |
| `ping URL` | Sends some packets to the URL to see how long it takes. |
| `cp FILE1 FILE2` in Linux/Mac `copy FILE1 FILE2` in Windows | Copies FILE1 and pastes it as FILE2. |
| `mv FILE1 FILE2` in Linux/Mac `move FILE1 FILE2` in Windows | Moves/cuts FILE1 and pastes it as FILE2. Can be used as a rename functionality, too. |

These aren't important to the class, but fun little things you might play with later on your own.

# Part 3

# Additional handy commands

There are various shortcuts you can use to navigate code more easily. Some things of note are:

- You can auto-format the code to look cleaner.

  In **Visual Studio**, go to **Edit** → **Advanced** → **Format Document** or use CTRL+K then CTRL+D.

  In **Code::Blocks**, right-click in the code window and select **Format use AStyle**.

  > **Clean code**
  > Note, I will take off points for inconsistent indentation and otherwise bad formatting, so make sure to write clean code as you go, or auto-format before turning in work!

- You can set bookmarks in code (similar to setting breakpoints) to quickly return to a line of code.

- You can usually collapse/expand code blocks in most modern IDEs. These usually look like a +/- square next to the beginning of a code block.

- You can right-click on a variable and click on something like "go to declaration" to see where the variable was declared.

- You can also right-click on a function and "go to declaration" to go to where the function declaration is (header only, ends with ;), and "go to definition" to go to where the function body is.

- In Code::Blocks, use CTRL+D to duplicate a line of code or the selection of code you have highlighted.
- You can highlight a bunch of code and comment it all out at once.

  In **Visual Studio**, go to **Edit** → **Advanced** → **Comment Selection** or use CTRL+K+C (CTRL+K+U to un-comment).

  In **Code::Blocks**, go to **Edit** → **Comment** or use CTRL+SHIFT+C (CTRL+SHIFT+X to un-comment).

# Part 4

# Appendix: Starter code

## 4.1 main.cpp

```cpp
#include "Game.hpp"

int main()
{
    Game game;
    game.Run();

    return 0;
}
```

## 4.2 Game.hpp

```cpp
#ifndef _GAME_HPP
#define _GAME_HPP

#include "Map.hpp"

enum GameState { MAIN_MENU, GAME, QUIT };

class Game
{
    public:
    Game();
    ~Game();

    void Run();

    private:
    GameState m_gameState;
    Map m_map;
    int m_score;

    GameState MainMenu();
    GameState Gameplay();
```

```
23      GameState YouWin();
24      GameState YouLose();
25      void Instructions();
26
27      void DrawHud();
28
29      void SaveGame();
30      void LoadGame();
31  };
32
33  #endif
```

## 4.3 Game.cpp

```
1   #include "Game.hpp"
2
3   #include "Utilities.hpp"
4
5   #include <cstdlib>
6   #include <ctime>
7   #include <iostream>
8   #include <string>
9   #include <iomanip>
10  #include <fstream>
11  using namespace std;
12
13  Game::Game()
14  {
15      // Seed the random number generator
16      srand( time( NULL ) );
17
18      // Set the starting program state
19      m_gameState = MAIN_MENU;
20
21      m_score = 0;
22
23      LoadGame();
24  }
25
26  Game::~Game()
27  {
28      SaveGame();
29  }
30
31  void Game::Run()
32  {
33      GameState nextState = m_gameState;
34
35      while ( nextState != QUIT )
36      {
37          if      ( nextState == MAIN_MENU )  { nextState = MainMenu(); }
38          else if ( nextState == GAME )       { nextState = Gameplay(); }
39      }
40
41      cout << "Game saved." << endl;
42  }
43
44  GameState Game::MainMenu()
```

```
45  {
46      ClearScreen();
47
48      cout << "TTTTT RRR   EEEE  ZZZZZ  OOOO  RRR    OOOO" << endl;
49      cout << "  T   R  R  E        Z   O  O  R  R   O  O" << endl;
50      cout << "  T   RRR   EEEE    ZZ    O  O  RRR    O  O" << endl;
51      cout << "  T   R  R  E      Z      O  O  R  R   O  O" << endl;
52      cout << "  T   R   R EEEE  ZZZZZ  OOOO  R   R  OOOO" << endl;
53      cout << "----------------------------------------" << endl;
54      cout << "1. Play" << endl;
55      cout << "2. Instructions" << endl;
56      cout << "3. Quit" << endl;
57      int choice = GetIntInput( 1, 3 );
58
59      if ( choice == 1 )
60      {
61          return GAME;
62      }
63      else if ( choice == 3 )
64      {
65          return QUIT;
66      }
67      else
68      {
69          Instructions();
70          return MAIN_MENU;
71      }
72  }
73
74  GameState Game::Gameplay()
75  {
76      bool done = false;
77      m_map.GenerateMap();
78
79      while ( !done )
80      {
81          ClearScreen();
82          m_map.Draw();
83          DrawHud();
84
85          string choice = GetStringInput();
86          m_map.MovePlayer( choice );
87
88          if ( choice == "QUIT" )
89          {
90              done = true;
91          }
92
93          if ( m_map.PlayerCollectTreasure() )
94          {
95              m_score++;
96              return YouWin();
97          }
98          else if ( m_map.GoblinGetPlayer() )
99          {
100             return YouLose();
101         }
102     }
103
104     return MAIN_MENU;
105 }
106
```

```
107 GameState Game::YouWin()
108 {
109     ClearScreen();
110     cout << "-------------------------------------------------" << endl;
111     cout << "-              Y O U       W I N !              -" << endl;
112     cout << "-                                               -" << endl;
113     cout << "- You have found " << m_score << " treasures. \t\t\t-" << endl;
114     cout << "-------------------------------------------------" << endl;
115 
116     cout << " Continue to the next level? (y/n): ";
117     string input = GetStringInput();
118     if ( input == "n" )
119     {
120         return MAIN_MENU;
121     }
122     else
123     {
124         return GAME;
125     }
126 }
127 
128 GameState Game::YouLose()
129 {
130     ClearScreen();
131     cout << "-------------------------------------------------" << endl;
132     cout << "-                   O H   N O !                 -" << endl;
133     cout << "-                                               -" << endl;
134 
135     if ( m_score == 0 )
136     {
137         cout << "- The goblin got you, but you have no treasure! -" << endl;
138     }
139     else
140     {
141         m_score--;
142         cout << "- The goblin stole one of your treasures!       -" << endl;
143     }
144 
145     cout << "-------------------------------------------------" << endl;
146 
147     cout << " Continue to the next level? (y/n): ";
148     string input = GetStringInput();
149     if ( input == "n" )
150     {
151         return MAIN_MENU;
152     }
153     else
154     {
155         return GAME;
156     }
157 }
158 
159 void Game::Instructions()
160 {
161     cout << "Use the [W, A, S, D] keys to move around the labyrinth." <<
    endl;
162     cout << "Find treasure and collect it to win." << endl;
163     cout << endl;
164     cout << "Go back? (y/n): ";
165     GetStringInput();
166 }
167 
```

```
168 void Game::DrawHud()
169 {
170     cout << endl;
171     cout << "------------------------------------------------" << endl;
172     cout << "- GOAL: Move yourself (@) to           SCORE:   -" << endl;
173     cout << "-       collect the treasure ($)!      " << m_score << "\t-" <<
         endl;
174     cout << "-       Avoid the goblin (&)!                   -" << endl;
175     cout << "-                                               -" << endl;
176     cout << "-                " << m_map.MoveUpKey() << ": Move NORTH
                -" << endl;
177     cout << "- " << m_map.MoveLeftKey() << ": Move WEST   " << m_map.
        MoveDownKey() << ": Move SOUTH   " << m_map.MoveRightKey() << ": Move
        EAST   -" << endl;
178     cout << "-                                               -" << endl;
179     cout << "- Or type QUIT to quit.                         -" << endl;
180     cout << "------------------------------------------------" << endl;
181 }
182
183
184 void Game::SaveGame()
185 {
186     ofstream output( "save.txt" );
187     output << m_score;
188 }
189
190 void Game::LoadGame()
191 {
192     ifstream input( "save.txt" );
193     input >> m_score;
194 }
```

## 4.4 Map.hpp

```
1  #ifndef _MAP_HPP
2  #define _MAP_HPP
3
4  #include <string>
5  using namespace std;
6
7  struct Coords
8  {
9      int x, y;
10 };
11
12 class Map
13 {
14     public:
15     Map();
16     void GenerateMap();
17     void Draw();
18     void MovePlayer( string direction );
19     void MoveGoblin();
20     bool PlayerCollectTreasure();
21     bool GoblinGetPlayer();
22
23     char MoveUpKey();
24     char MoveDownKey();
```

```
25      char MoveLeftKey();
26      char MoveRightKey();
27
28      private:
29      const int MAP_WIDTH;
30      const int MAP_HEIGHT;
31      const char EMPTYROOM;
32      const char WALL;
33      const char MOVE_UP;
34      const char MOVE_DOWN;
35      const char MOVE_LEFT;
36      const char MOVE_RIGHT;
37
38      char m_tiles[20][10];
39      Coords m_player;
40      Coords m_treasure;
41      Coords m_enemy;
42  };
43
44  #endif
```

## 4.5 Map.cpp

```
1   #include "Map.hpp"
2
3   #include <iostream>
4   #include <vector>
5   using namespace std;
6
7   Map::Map()
8       : MAP_WIDTH(20), MAP_HEIGHT(10), EMPTYROOM(' '), WALL('+'),
9           MOVE_UP('w'), MOVE_DOWN('s'), MOVE_LEFT('a'), MOVE_RIGHT('d')
10  {
11  }
12
13  void Map::GenerateMap()
14  {
15      // Fill in all the rooms
16      for ( int y = 0; y < MAP_HEIGHT; y++ )
17      {
18          for ( int x = 0; x < MAP_WIDTH; x++ )
19          {
20              m_tiles[x][y] = WALL;
21          }
22      }
23
24      vector<Coords> roomCoords;
25
26      // Choose random rooms
27      int roomCount = rand() % 5 + 5;
28      for ( int i = 0; i < roomCount; i++ )
29      {
30          Coords c;
31          c.x = rand() % 18 + 1;
32          c.y = rand() % 8 + 1;
33          m_tiles[c.x][c.y] = EMPTYROOM;
34          roomCoords.push_back( c );
35      }
```

```
36
37      // Connect all the rooms
38      for ( unsigned int i = 0; i < roomCoords.size() - 1; i++ )
39      {
40          int x = roomCoords[i].x;
41          int y = roomCoords[i].y;
42          int targetX = roomCoords[i+1].x;
43          int targetY = roomCoords[i+1].y;
44
45          while ( x != targetX )
46          {
47              m_tiles[x][y] = EMPTYROOM;
48
49              if ( x < targetX ) { x++; }
50              else { x--; }
51          }
52
53          while ( y != targetY )
54          {
55              m_tiles[x][y] = EMPTYROOM;
56
57              if ( y < targetY ) { y++; }
58              else { y--; }
59          }
60      }
61
62      // Place the player and treasure
63      int randX = rand() % MAP_WIDTH;
64      int randY = rand() % MAP_HEIGHT;
65
66      while ( m_tiles[randX][randY] != EMPTYROOM )
67      {
68          randX = rand() % MAP_WIDTH;
69          randY = rand() % MAP_HEIGHT;
70      }
71
72      m_player.x = randX;
73      m_player.y = randY;
74
75      randX = rand() % MAP_WIDTH;
76      randY = rand() % MAP_HEIGHT;
77      while ( !( m_tiles[randX][randY] == EMPTYROOM
78              && randX != m_treasure.x && randY != m_treasure.y ) )
79      {
80          randX = rand() % MAP_WIDTH;
81          randY = rand() % MAP_HEIGHT;
82      }
83
84      m_treasure.x = randX;
85      m_treasure.y = randY;
86
87      randX = rand() % MAP_WIDTH;
88      randY = rand() % MAP_HEIGHT;
89      while ( !( m_tiles[randX][randY] == EMPTYROOM
90                  && randX != m_treasure.x && randY != m_treasure.y
91                  && randX != m_player.x && randY != m_player.y ) )
92      {
93          randX = rand() % MAP_WIDTH;
94          randY = rand() % MAP_HEIGHT;
95      }
96
97      m_enemy.x = randX;
```

```
98      m_enemy.y = randY;
99  }
100
101 void Map::Draw()
102 {
103     cout << endl;
104     for ( int y = 0; y < MAP_HEIGHT; y++ )
105     {
106         cout << " ";
107         for ( int x = 0; x < MAP_WIDTH; x++ )
108         {
109             if ( x == m_player.x && y == m_player.y )
110             {
111                 cout << "@";
112             }
113             else if ( x == m_treasure.x && y == m_treasure.y )
114             {
115                 cout << "$";
116             }
117             else if ( x == m_enemy.x && y == m_enemy.y )
118             {
119                 cout << "&";
120             }
121             else
122             {
123                 cout << m_tiles[x][y];
124             }
125         }
126         cout << endl;
127     }
128 }
129
130 void Map::MovePlayer( string direction )
131 {
132     char firstLetter = direction[0];
133     firstLetter = tolower( firstLetter );
134
135     if ( firstLetter == MOVE_UP )
136     {
137         if ( m_player.y - 1 >= 0 && m_tiles[m_player.x][m_player.y - 1] ==
    EMPTYROOM )
138         {
139             m_player.y--;
140         }
141     }
142     else if ( firstLetter == MOVE_DOWN )
143     {
144         if ( m_player.y + 1 < 10 && m_tiles[m_player.x][m_player.y + 1] ==
    EMPTYROOM )
145         {
146             m_player.y++;
147         }
148     }
149     else if ( firstLetter == MOVE_RIGHT )
150     {
151         if ( m_player.x + 1 < 20 && m_tiles[m_player.x + 1][m_player.y] ==
    EMPTYROOM )
152         {
153             m_player.x++;
154         }
155     }
156     else if ( firstLetter == MOVE_LEFT )
```

```
157     {
158         if ( m_player.x - 1 >= 0 && m_tiles[m_player.x - 1][m_player.y] ==
    EMPTYROOM )
159         {
160             m_player.x--;
161         }
162     }
163 
164     MoveGoblin();
165 }
166 
167 void Map::MoveGoblin()
168 {
169     int x = m_enemy.x;
170     int y = m_enemy.y;
171 
172     int direction = rand() % 4;
173 
174     if ( direction == 0 )
175     {
176         x += 1;
177     }
178     else if ( direction == 1 )
179     {
180         x -= 1;
181     }
182     else if ( direction == 2 )
183     {
184         y += 1;
185     }
186     else
187     {
188         y -= 1;
189     }
190 
191     if ( m_tiles[x][y] == EMPTYROOM )
192     {
193         m_enemy.x = x;
194         m_enemy.y = y;
195     }
196 }
197 
198 bool Map::PlayerCollectTreasure()
199 {
200     return ( m_player.x == m_treasure.x && m_player.y == m_treasure.y );
201 }
202 
203 bool Map::GoblinGetPlayer()
204 {
205     return ( m_player.x == m_enemy.x && m_player.y == m_enemy.y );
206 }
207 
208 char Map::MoveUpKey()
209 {
210     return MOVE_UP;
211 }
212 
213 char Map::MoveDownKey()
214 {
215     return MOVE_DOWN;
216 }
217 
```

```
218 char Map::MoveLeftKey()
219 {
220     return MOVE_LEFT;
221 }
222
223 char Map::MoveRightKey()
224 {
225     return MOVE_RIGHT;
226 }
```

## 4.6 Utilities.hpp

```
1  #ifndef _UTILITIES_HPP
2  #define _UTILITIES_HPP
3
4  #include <string>
5  using namespace std;
6
7  void ClearScreen();
8  int GetIntInput( int minimum, int maximum );
9  string GetStringInput();
10
11 #endif
```

## 4.7 Utilities.cpp

```
1  #include "Utilities.hpp"
2
3  #include <iostream>
4  using namespace std;
5
6  void ClearScreen()
7  {
8      #if defined(WIN32) || defined(_WIN32) || defined(__WIN32) && !defined(
       __CYGWIN__)
9      // The terminal clear command on Windows is "cls"
10     system( "cls" );
11     #else
12     // The terminal clear command on Linux/Mac/Unix is "clear"
13     system( "clear" );
14     #endif
15 }
16
17 int GetIntInput( int minimum, int maximum )
18 {
19     cout << ">> ";
20     int val;
21     cin >> val;
22     while ( val < minimum || val > maximum )
23     {
24         cout << "Invalid choice." << endl;
25         cout << ">> ";
26         cin >> val;
```

```
27      }
28      return val;
29  }
30
31  string GetStringInput()
32  {
33      cout << ">> ";
34      string val;
35      cin >> val;
36      return val;
37  }
```