Goal:

The purpose of this lab is to understand the basic requirements for Assignment 1 and to get started working on the assignment by understanding the supplied starting code and writing code to draw the game map.

Part 1: Create Your Assignment Directory

1. Download the Project Template from Blackboard in the Assignment 1 section of the Introduction to C learning module.
2. Unzip the template after moving it to wherever you like in your Linux directories.

my-dir% **unzip robot_template.zip**

Part 2: Test the Program

1. Change to the project directory containing the source code for your project.

my-dir% **cd robot_template**

2. Build the application. The beginning of `game.c` contains the build instructions. Note, your final submission will be compiled with exactly this command:

robot_template% **gcc main.c game.c -ansi -g -Wall -Wshadow -lc -o r**

3. Run the application. Quit by typing 'Q' (capital-q).

robot_template% **./r**

Part 3: Understand the structure of the template:

View but do not modify `game.h`. The important function declaration to note is `void setup_board(int whichLevel);` which is not completely written for you. In a later section, you will comlete this function in `game.c`. The supplied implementation creates and empty board with one robot.

View but do not modify the `main` function in `main.c`.

```
int main(int argc, const char * argv[]){
	char c = ' ';

	/* initialize the random number generator */
	srand(time(0));

	/* setup game */
	open_game();
	setup_board(1);
	draw_game();

	while(c != 'Q'){
		c = mgetkey();
		if(0 < refresh_robots() && strchr(kValidGameCommands, c)){
			printf("command: %c\n", c);
			process_command(c);
			draw_game();
		}
	}
	close_game();

	mgetkey_reset();
	return 0;
}
```

The important things to observe are as follows:

1. The game is initially started at level 1.
2. There is a loop which gets input from the player.
3. The input loop stops if the player types `'Q'`.
4. The game handles each command in its `process_command` function.
5. `draw_game` is called to update the display; you should not have to call this function yourself.

Also examine the implementation of `draw_board`, `get_board`, and `set_board` in `main.c`.

Part 4: Initialise the game level:

In `game.c`, change the implementation of the `setup_board` function to create a line of robots and each different item that can occupy a square in the game. Moving north (up) should run the robots into the items.

Compile and test your code. You should see a map of the game with many items. Note if you press 'w' to move the robots then nothing interesting happens.

Part 5: Moving robots:

Implement the move function in game.c so that robots will move onto empty squares of the board. What happens if you move a horizontal line of robots left and right?

Discuss how to get robots moving in a line. You may want to convert robots you've moved into the `kMovedRobot` type so that you can tell which has been moved (there are other solutions as well). Note, the code in main.c converts `kMovedRobots` back to `kRobot` before each command is processed.

Part 6: Moving onto Exits and Fires:

Add features that allow robots to move onto fires and exits on the board. In both cases the robot should be removed from the board, but moving a robot onto an exit should also increment the numEscapedInLevel counter.

Print the current game level and number escaped so you can monitor the correct behavior.

Appendix: Standard C Library Functions:

The only external function you need to complete this lab are defined in main.h.