Einführung in die Programmierung für Physiker WiSe 2017/18 - Final Project

```
##   ## # #    # ######  ####  #    # ###### ###### #####  ###### #####
# # # # # ##   # #      #      #    # #      #      #    # #      #    #
#  #  # # # #  # #####   ####  #    # #####  #####  #    # #####  #    #
#     # # #  # # #           # # ## # #      #      #####  #      #####
#     # # #   ## #      #    # ##  ## #      #      #      #      #   #
#     # # #    # ######  ####  #    # ###### ###### #      ###### #    #
```

1 Introduction

This project contains a console implementation of the well-known game 'Minesweeper'. It is written in C and tested in a ubuntu terminal The function to clear the screen used in this program: system("clear") needs to be changed to: system("cls") for Windows. Please note that especially colors are dependant of the operating system and color scheme of your terminal. Therefore they might not appear the way they should.

2 How to use the program

To start the program, run the executable "Minesweeper". You have the option of adding arguments for the number of rows, columns and mines as well as the additional option of playing an extended version of the game with periodic boundary conditions.

To compile to programm using gcc, type: ''' gcc -o minesweeper text.c game_logic.c board.c main.c ''' In that order. To run the executable, use for example ''' ./minesweeper -r 10 -c 10 -m 10 ''' This will run the game with a board consisting of 10 rows, columns and mines and no boundary conditions. If you want to play the extended version, add -b to invoke the periodic boundary conditions. The arguments are optional. If you don't provide any command line arguments, the default game with 8 rows, 8 columns and 7 mines and no periodic boundary conditions will start. Every argument you privide overwrites the respective default parameter.

3 Rules and functionalities of the game

The basic rules of the game are simple. Hidden somewhere underneath the tiles is a certain amount of mines. Your task is to open all safe tiles without opening a mine tile. If you do open a mine tile, you lose instantly. Every safe tile has a number on it that indicates how many of the adjacent tiles have mines on them. If you think you know which of the hidden tiles has a mine underneath, you can mark that tile as armed. Of course you can also disarm a tile if you change your mind. However you cannot mark an already revealed tile as armed, because that doesn't make any sense. If you reveal a tile that doesn't have any mines surrounding it (an empty tile), all surrounding tiles will automatically be revealed as well. This will set off a chain reaction if one of the surrounding tiles is empty as well. You also have the option of "digging" underneath an already revealed tile, meaning you can re-reveal an open tile and thereby reveal all adjacent tiles. This option is only available if the number of marked-as-armed tiles adjacent to the chosen tile is at least as high as the number on the tile. The marked tiles are safe, which means that you are safe as long as you've marked the right tiles. If you've made a mistake, you'll lose. The second you open the last safe tile on the board, you win the game.

If you've lost, the board will be printed another time in the state it was when the game ended but with all mines highlighted in one shade of red, and the mine that made you lose in another.

If you've won, the board will be printed with mines highlighted in green color.

3.1 Periodic boundary conditions

Having periodic boundary conditions in this game means that all tiles have the same number of adjacent tiles. Therefore borders don't actually exist, a border tile connects to another border tile on the other side of the board. Instead of a board like this:

```
|2|x|2| | | | |
|2|x|3|1| | | |
|1|2|x|1| | | |
| |1|1|2|1|2|1|1|
| | | |1|x|2|x|1|
| |1|1|2|1|2|1|1|
|1|2|x|1| | | |
|1|x|2|1| | | |
```

You actually have a board like this:

```
| |1|x|2|1| | | | |1|
| |2|x|2| | | | | |2|
| |2|x|3|1| | | | |2|
| |1|2|x|1| | | | |1|
|1| |1|1|2|1|2|1|1| |
|1| | | |1|x|2|x|1| |
|1| |1|1|2|1|2|1|1| |
| |1|2|x|1| | | | |1|
| |1|x|2|1| | | | |1|
| |2|x|2| | | | | |2|
```

The top row is below the bottom row, the right border is left to the left border and vice versa. Therefore you should be careful and take the mines on the other side of the field into account. Revealing adjacent tiles, the chain reaction for empty tiles and the digging option all cross the borders.

4 Running Tests

Wrong user input gets automatically stored into a log file. This way, the error message doesn't interrupt the game, but if something goes wrong, it is easy to keep track of what happened.

The functions themselves can be tested by running the testing-executable. To compile, use

```
gcc -o test text.c game_logic.c board.c test_functions.c test_main.c
```

The output should look something like this:

```
create_mines:                                       OK
create_mask:                                        OK
number_fields (no boundary-condition):              OK
number_fields (boundary-condition):                 OK
reveal_adjacent_tiles (no boundary-condition):      OK
dig_under_open_tile (boundary-condition):           OK
dig_under_open_tile (no boundary-condition):        OK
```

```
-------------------------------------------------------------------------------
Tests run: 7,  failed 0, passed: 7
```