ECE 150 Fall 2021

Project 3

Early Submission Due: 10:00pm on Friday, October 29, 2021

Due: 10:00pm on Thursday, November 4, 2021

Submission details

Project 3 will have 2 submission deadlines. You need to submit to **both**.

No late submission will be accepted.

Deadline #1 (Project #3 Early Submit)

- The functions tested in **Early Submit** are the following:
    1. `createBoard()`
    2. `hideBoard()`
    3. `cleanBoard()`
    4. `printBoard()`
    5. `mark()`
- The tests are out of 10, but we will mark it out of 5
    - ex: 7/10 will be marked as 7/5
- Which means:
    - Bonus: any grade above 5/10 will count as bonus toward the project total grade

Note: your project grade will still be capped at 100% (you will not get > 100%) on the overall project

Deadline #2 (Project #3)

- Your final grade will be
    - The sum of *Project #3 Early Submission* out of 5 AND *Project #3*

Introduction

Geese are part of Waterloo life! Everyone knows to not get too close to a goose, in particular a mother goose with her goslings.

In this project you will develop "GeeseSpotter", a simple, text-based, single-player game.

The player is first asked for the desired dimensions of the 2-dimensional playing board and then for the number of geese that should hide on the playing board. The playing board is made up of individual fields. A field can be occupied by a goose or not. Geese are messy, and the (up to) eight neighboring fields of a field that contains a goose contain goose droppings. An observant Waterloovian can tell how many geese are in neighboring fields by observing the amount of goose droppings.

Let's look at a simple four by four board. Two geese are hidden on the board, represented with the number `9` on the field. The other fields contain no geese, but instead contain the number of neighboring fields that contain a goose. That number can range from zero to eight. The x coordinate increases from left to right and starts at zero. The y coordinate increases from top to bottom and also starts at zero. So, the two geese are at coordinates (`x:2, y:0`) and (`x:1, y:2`) marked with a yellow background.

y \ x	0	1	2	3
0	0	1	9	1
1	1	2	2	1
2	1	9	1	0
3	1	1	1	0

The value of field (`x:2, y:1`) is determined by looking at the following 8 fields:

0	1	9	1
1	2	2	1
1	9	1	0
1	1	1	0

Two of those fields contain geese, so the value of field (`x:2, y:1`) is 2.

The University is asking us to determine all fields that contain a goose, without disturbing a goose. To make this easier, a field can be marked as containing a goose. A marked field is a reminder that a field might contain a goose. Only hidden fields can be marked and only marked fields can be unmarked. As the fields are initially hidden, there is no guarantee that a marked field contains a goose.

The player is presented with the current status of the playing board, where the content of all fields is initially unknown. The player is given the option to either reveal a field or to mark or unmark a field as likely occupied by a goose, by asking the player whether to reveal or mark/unmark a field and the two coordinates of the field.

If the player selects to reveal a field, there are two possible outcomes: (1) the field contains a goose, and the player upsets the mighty goose. The game is lost immediately. (2) the field does not contain a goose; the field is revealed. If the field and none of its neighbors contain a goose, the neighboring fields are also revealed, and the player learns how many geese are in the neighbors of each of the neighboring fields. This value can range from 0 to 8.

If the player selects to mark or unmark a field, the status of the field is changed from unknown to marked or from marked to unknown, depending on the current status of the field. Only unknown fields can be revealed or marked and only marked fields can be unmarked and thereby changed to an unknown field again.

The current status of the board is printed after every change to the fields. The player is free at any turn to choose to restart with a new game or to quit.

Let's take the earlier playing field. Initially, all fields are hidden and the player sees only stars:

As a first move, the player selects to reveal field (`x:0, y:0`). The value of that field is zero, so also the three neighboring fields are revealed. The new board is now:

Next, the player decides to reveal field (`x:2, y:1`). The value of that field is two, so no further fields are revealed. The new board is now:

The player now guesses that field (`x:1, y:2`) contains a goose and marks the field. The board is now:

The player next decides to reveal field (`x:2,y:0`). A goose inhabits that field and the game is lost.

The game is won when only (marked or unknown) fields with geese remain, that is, there are no marked or unknown fields without a goose left. Once a game is won or lost, it starts again by asking for the dimensions of the next playing board.

Example

Here is a sample interaction with the game:

```
Welcome to GeeseSpotter!
Please enter the x dimension: 5
Please enter the y dimension: 5
Please enter the number of geese: 2
*****
*****
*****
*****
*****
Please enter the action ([S]how; [M]ark; [R]estart; [Q]uit): S
```

```
Please enter the x location: 0
Please enter the y location: 0
00***
11***
*****
*****
*****
Please enter the action ([S]how; [M]ark; [R]estart; [Q]uit): S
Please enter the x location: 4
Please enter the y location: 0
00**1
11***
*****
*****
*****
Please enter the action ([S]how; [M]ark; [R]estart; [Q]uit): S
Please enter the x location: 0
Please enter the y location: 4
00**1
11***
*****
11***
00***
Please enter the action ([S]how; [M]ark; [R]estart; [Q]uit): S
Please enter the x location: 2
Please enter the y location: 4
00**1
11***
*****
1110*
0000*
```

Implementation

We provide you with the basic game skeleton and you are asked to implement the details.

We provide the following eight functions:

1. `int main();`
   The main entry point for the game.
2. `bool game();`
   One complete game.
3. `void spreadGeese(char *board, std::size_t xdim, std::size_t ydim, unsigned int numgeese);`
   Randomly spreads numgeese geese over the board.
4. `std::size_t readSizeT();`
   A helper function to reliably read a `std::size_t` value from `std::cin`.
5. `std::size_t xdim_max();` and `std::size_t ydim_max();`
   Helpers to determine maximum x and y dimensions.
6. `char markedBit();`
   A field with bit `0x10` set is a marked field.
7. `char hiddenBit();`
   A field with bit `0x20` set is a hidden field.
8. `char valueMask();`
   The mask `0x0F` can be used to easily remove the marked and hidden bits from a field value.

Fields are represented as chars. We use the lower four bits of a char to represent the value of goose neighbors and the value 9 to represent a goose. We set markedBit() to mark a field and we set hiddenBit() to hide a field. Combining a field value with the valueMask() always determines whether a field contains a goose or how many geese neighbors there are. As an example, consider a field that has two geese neighbors. Its value is `0x02` initially. When the field is hidden, its value is changed to `0x22`, by setting the hidden bit `0x20`. If the player then decides to mark that field, its value is changed to `0x32` (both the marked bit `0x10` and the hidden bit `0x20` are set to true). The value can then be unmarked, going back to `0x22`. And finally, the field can be revealed, changing it to `0x02`.

The board is represented as a one-dimensional array of chars, in row-major order. For our example board:

0	1	9	1
1	2	2	1
1	9	1	0
1	1	1	0

The char array will contain the following values:

Index	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Value	0	1	9	1	1	2	2	1	1	9	1	0	1	1	1	0

Your Task

Implement the following functions:

1. `char *createBoard(std::size_t xdim, std::size_t ydim);`
   Allocate a char array with xdim * ydim elements and initialize each element with zero.
2. `void computeNeighbors(char *board, std::size_t xdim, std::size_t ydim);`
   For a board that contains only the values 0 or 9, compute the number of goose-neighbored fields.
3. `void hideBoard(char *board, std::size_t xdim, std::size_t ydim);`
   Hide all the field values.
4. `void cleanBoard(char *board);`
   To deallocate the given board.
5. `void printBoard(char *board, std::size_t xdim, std::size_t ydim);`
   Print the content of the board to `std::cout`.A hidden field uses '*', a marked field uses 'M', and otherwise the field value is displayed. Note that the only time a field with a goose is displayed is at the end of the game (either won or lost).
6. `int reveal(char *board, std::size_t xdim, std::size_t ydim, std::size_t xloc, std::size_t yloc);`
   Reveal a hidden field. Return 1 if the field is marked. Return 2 if the field is already revealed. Return 9 if the field contains a goose. Return 0 otherwise.
   If an empty field is revealed (it contains the value zero), also reveal the values of the neighbors that don't contain geese.
7. `int mark(char *board, std::size_t xdim, std::size_t ydim, std::size_t xloc, std::size_t yloc);`
   Mark the given field. Return 2 if the field is already revealed. Return 0 otherwise.
8. `bool isGameWon(char *board, std::size_t xdim, std::size_t ydim);`
   Determine whether the board is in a won state. For example, by creating a 5 by 5 board with

25 geese on it, you would expect that you immediately win as every field contains a goose. However, you first need to mark any field before the program realizes that the game is won. Review the usage of function isGameWon() in function game() and determine the necessary changes to the logic.

Visual Studio Code Project Setup

1. Put **"geesespotter_lib.h"**, **"geesespotter_lib.cpp"**, and **"geesespotter.h"** in a folder you create.
2. Create a file called **"geesespotter.cpp"** inside the same folder. There should now be 4 files in your folder. At the top of your **geesespotter.cpp** file, make sure to write the following statement:

```
#include "geesespotter_lib.h"
```

3. Start by creating the skeleton definitions of every functions in your **geesespotter.cpp** file as stated in Your Tasks.
4. To test your **geesespotter.cpp** file, type the following command into terminal while in the folder directory.

```
g++ -o main geesespotter.cpp geesespotter_lib.cpp -std=c++11
```

Run main to run the test file.

5. The **"geesespotter.cpp"** file will be the only file you need to submit on Marmoset. Submit this file with the starter function definitions and make sure it compiles on Marmoset!

Note: Your geesespotter.cpp file does NOT require a main function or function declarations. Only function definitions are required.

Marmoset Submission Details

Project name: Project #3

Submission filename: geesespotter.cpp

Note that as you are not submitting a `main()` function, you will not require `#ifndef` `MARMOSET_TESTING` pre-processor directives.

Plagiarism detection software

We analyze all submissions with automated plagiarism detection software. Please consult the syllabus for more information.