COMP 1602 Computer Programming II
2018/2019 Semester 2
Assignment 4: Minesweeper

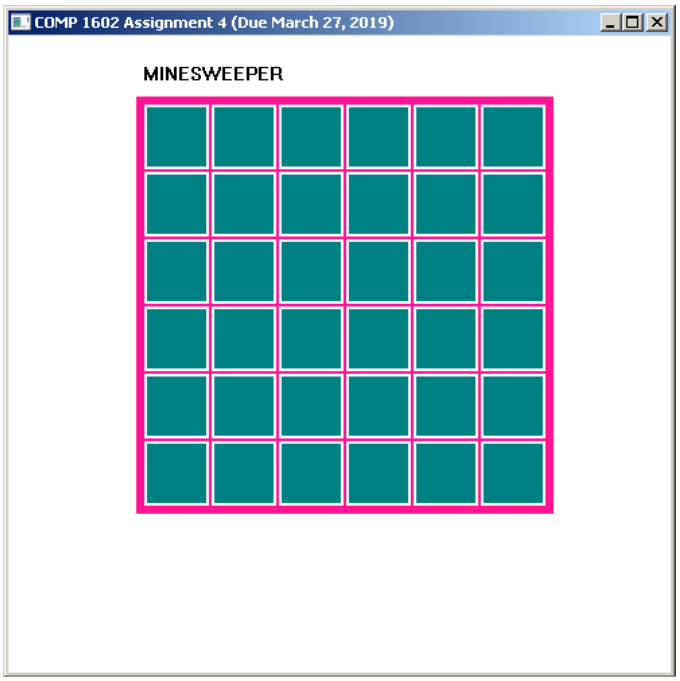
Date Due: *Wednesday March 27th, 2019 @ 11:55 pm*

Overview

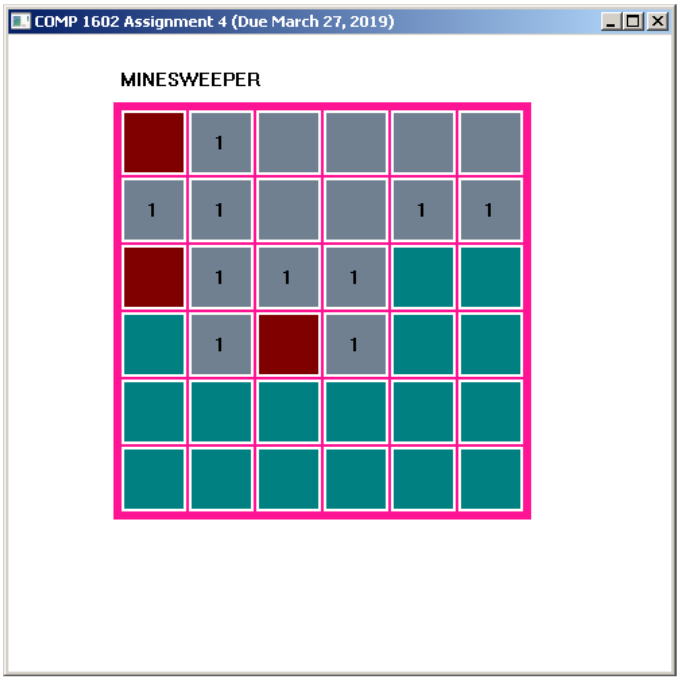
This assignment requires you to write part of the code for the game *Minesweeper*. Minesweeper is played using a 2D board with a certain amount of rows and columns. For example, the board can be of size 6 rows x 6 columns. Mines (or bombs) are randomly placed throughout the board. Each cell on the board that is not a mine contains a number indicating how many mines are adjacent to that cell. A cell could be blank indicating that there are no mines adjacent to that cell. Initially, all the cells are uncovered or hidden. The objective of the game is for the player to uncover all the cells not containing a mine. If the player uncovers a mine, he/she loses the game immediately.

Sample Screen Shots

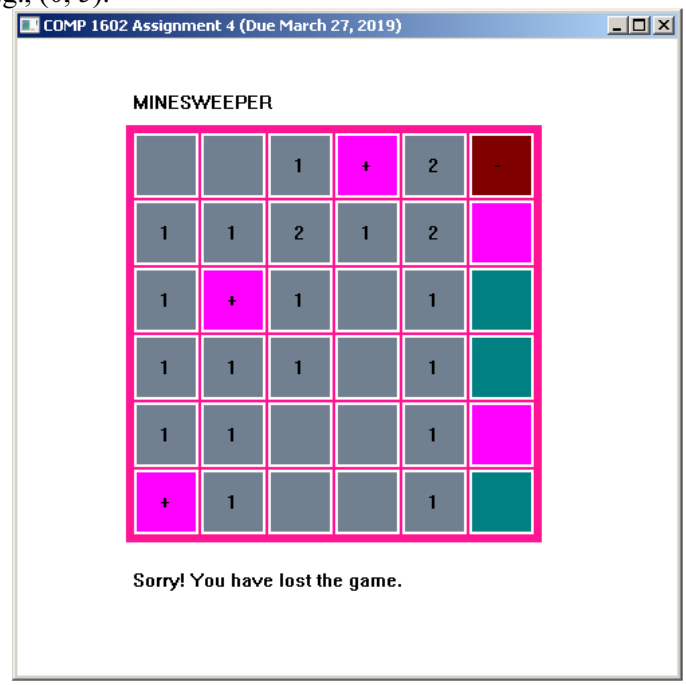
The following window shows the initial window generated for a Minesweeper game with 6 rows and 6 columns:



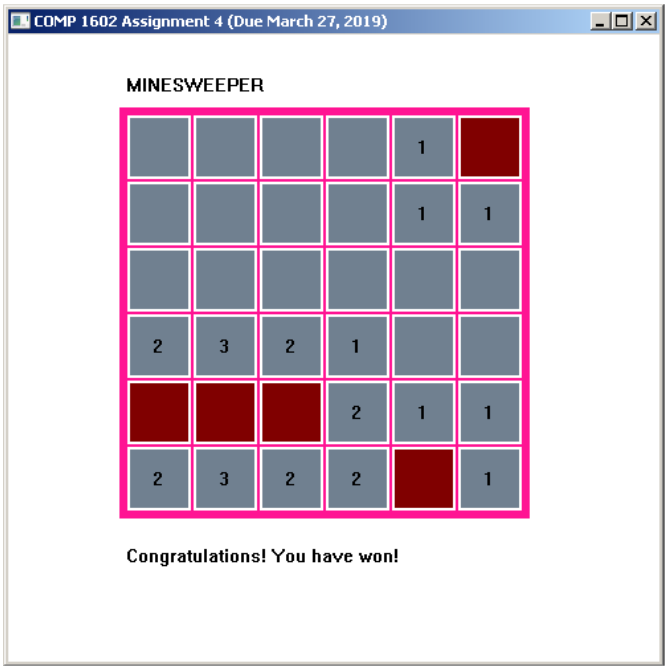
The following window shows when several cells have been exposed due to the player left-clicking on cells such as (0, 1), (1, 0), (1, 1), etc. Notice that the player has marked three cells (by right-clicking on (0, 0), (2, 0), and (3, 2)) to indicate that he/she believes that these cells contain a mine. These are coloured red by the graphics system.



The following window shows a lose situation where the player clicked on a mine by mistake. There are five mines in this game. When a player loses the game, all the cells containing a mine are coloured magenta. If the player had correctly marked a cell containing a mine, a plus symbol is shown on the cell. If the player had marked a cell which does not contain a mine, a minus symbol is shown, e.g., (0, 5).



The following window shows a win situation where the player has uncovered all the cells not containing a mine. There are five mines in this game.



Graphics Display

The output from your program is displayed graphically on a window using the Windows programming graphics library. You do **NOT** need to know or write any graphics code. Your task in this assignment is to write functions that initialize the game board and implement the behavior of the game. The graphics functions will call your functions to achieve the functionality of the game. For example, the graphics functions can determine if a mouse click (left or right) took place on a given cell (specified by row and column numbers). Given this information, you must determine if the player has won, if the player has lost, or how to progress with the game (e.g., uncover a cell).

Programming Guidelines

The *Graphics.cpp* file (available for download at the course Web site) contains the declarations and code stubs for **all** the functions you need to write for this program. All you have to do is write code to implement each function.

The functions you have to write are described in the following table:

Function	**Description**
select()	This function is called when the user left-clicks on the mouse on a particular cell. It is possible that the cell is blank, has a number assigned to it, or contains a mine. If there are no more mines to be discovered, it results in a "win" situation. It returns 0 to indicate that the game is still in progress, -1 to indicate that the game has been lost, or 1 to indicate that the game has been won
mark()	This function is called when a user right-clicks on the mouse to indicate that a cell should be "marked" as having a mine. If a cell is already marked, it should be un-marked.
hasWon()	This function determines if the current situation on the board is a "win" situation. A "win" situation occurs when all the hidden cells are mines.
setNumbers()	This function sets the number to be displayed in all the cells on the board (which is not a mine). It calls the *getNumber* function to get the number for a particular cell.
getNumber()	This function finds the number that should be assigned to a given cell on the board (which is not a mine). This number is calculated based on the number of adjacent mines.
isValidLocation()	Given a particular (row, col) location, this function determines if the location is valid for the given board. A location is valid if it is within the range of the rows and columns of the board.
placeMinesOnBoard()	This function randomly places a certain amount of mines (specified as a parameter, *numMines*) on the 2D board.
initGameBoard()	This function initializes all the cells in the 2D minesweeper board.
createCell()	This function creates and initializes a *Cell* struct to default values.

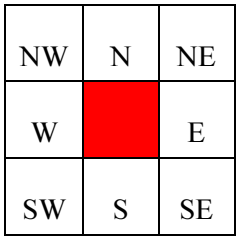
Note that you do **not** have to write the *clearBlanks()* function. However, it must be called when the player left-clicks on a cell that is blank.

Structs

The program uses two structs to store information on the game, *Cell* and *MouseEvent*. The *Cell* struct stores data for each cell on the 2D board as follows:

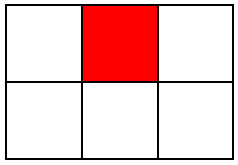
```
struct Cell {
   bool hasMine;      // does the cell have a mine?
   bool isMarked;     // has the player marked the cell by right-clicking on it?
   bool isHidden;     // is the cell still hidden from the player?
   int number;        // a number representing the amount of adjacent mines
};
```

The number for each cell is calculated in the *getNumber* function based on the amount of mines that are adjacent to the given cell. Consider the cell shaded in red below:



The cells to the north, south, east, and west of the shaded cell are adjacent to the shaded cell. However, the cells to the north-east, south-east, south-west, and north-west are also adjacent to the shaded cell. So, there are eight cells that are adjacent to the shaded cell.

In searching for the cells that are adjacent to a given cell, it is important to only consider cells that are on the board. For example, suppose that the shaded cell is on the topmost row of cells:



In this case, there will be only five adjacent cells. The *getValidLocation* function can be used to ensure that cell locations outside the range of the board are not considered when generating the numbers for each cell.

The *MouseEvent* struct stores information when the mouse is clicked on a cell:

```
struct MouseEvent {
   bool leftButtonPressed;   // was the left mouse button pressed?
   bool rightButtonPressed;  // was the right mouse button pressed?
   int row;                  // row number where mouse button was pressed
   int col;                  // column number where mouse button was pressed
};
```

It is used internally by the graphics code to deal with mouse clicks on a cell.

What You are Given

A zip file, *Assignment4.zip*, is available at the course Web site containing all the files you need for this assignment:

```
Assignment4.dev
Graphics.cpp
ApplauseClip.wav
GameOverClip.wav
```

Create a folder for Assignment 4 on your computer and unzip all the files to this folder. Double click on *Assignment4.dev* and Dev C++ opens. On the left side of the window, you will see the following files listed:

```
Assignment4.cpp
Graphics.cpp
```

Click on *Graphics.cpp* to edit this file. You must write all the code for this assignment in the file *Graphics.cpp*. A stub for each function is already given. The *Graphics.cpp* file also contains the code to generate the graphics. This code should not be modified; if you choose to modify the *Graphics.cpp* file, you do so at your own risk

Whenever you wish to compile your program, choose "Compile" or "Rebuild All" from the Execute menu.

What to Submit

Upload *Graphics.cpp* to myElearning by the deadline date. There is no need to upload the other files. Closer to the deadline date, an executable version of the program you are expected to write will be made available at the course Web site.