Project 2 - Minesweeper Init

CS 0447 — Computer Organization & Assembly Language

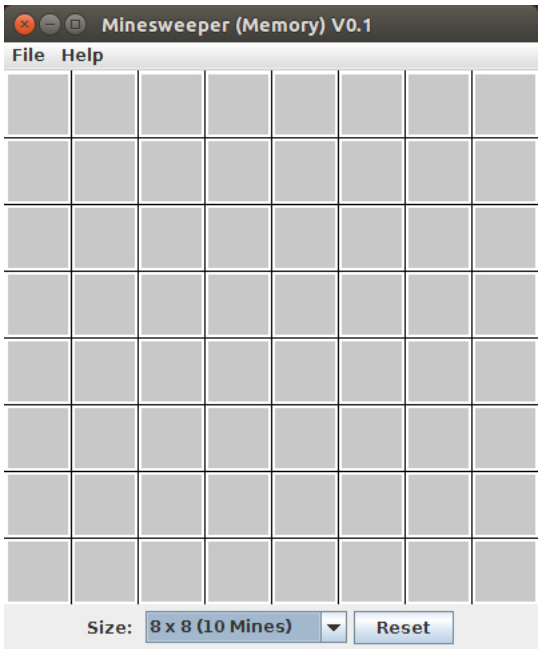
Check the Due Date on the CourseWeb

The purpose of this project is for you to practice writing MIPS assembly program with functions. Your goal is to initialize a Minesweeper game and read mouse input.

Introduction to MIPS' Minesweeper

From (Wikipedia), Microsoft Minesweeper (formerly Minesweeper) is a minesweeper computer game created by Curt Johnson, originally for OS/2, and ported to Microsoft Windows by Robert Donner, both Microsoft employees at the time. First officially released as part of the Microsoft Entertainment Pack 1 in 1990, it was included in the standard install of Windows 3.1 in 1992, replacing Reversi from Windows 3.0. Microsoft Minesweeper has been included without a major change in all subsequent Windows releases until Windows Vista, at which time an updated version by Oberon Media replaced it. In Windows 8 and later the game is not included, but Microsoft Studios published an updated version of it, developed by Arkadium, on Windows Store.

For this project, we will use our own Minesweeper hardware. This tool can be found in `minesweeper.zip` located in the CourseWeb under this project. Extract all files to your `[...]/mars4_5/mars/tools` directory. If you extract all files to the right directory, when you run the MARS program, you should see "Minesweeper (Memory) V0.1" under the "Tools" menu. Our MIPS' Minesweeper hardware is shown below:



For our Minesweeper hardware, there are three sizes:

- 8×8 with 10 mines,
- 12×12 with 15 mines, and
- 16×16 with 20 mines.

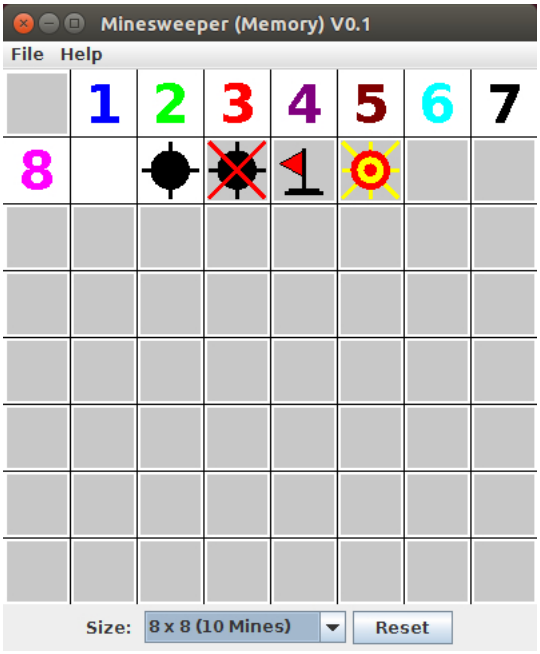
Note that this Minesweeper hardware is just a display/interface. Your program is the one who control its behavior. For example, if you press the "Reset" button. It does not actually do anything. It just sends a signal to your program that the "Reset" button is pressed. Your program needs to actually reset it.

A MIPS assembly program can control this Minesweeper hardware by writing data onto the main memory starting at the address 0xFFFF8000. Just imagine that this address (0xFFFF8000) contains a two-dimensional array of bytes. Each byte simply control a cell (row, column) of the Minesweeper game. For example, for the 8×8 game, cell (0, 0), row 0 column 0, is associated with the memory address 0xFFFF8000, cell (0, 1), row 0 column 1, is associated with the memory address 0xFFFF8001, cell (1, 0), row 1 column 0, is associated with the memory address 0xFFFF8008, and so on. Note that for a 16×16 game, cell (1, 0) is associated with the memory address 0xFFFF8010. In other word, the memory location associated with a cell (row, column) is based on the size of the game.

To change the appearance of each cell on the Minesweeper hardware, simply store a byte data into the address associated with the cell as follows:

- 0x00: Close
- 0x01: The number 1
- 0x02: The number 2
- 0x03: The number 3
- 0x04: The number 4
- 0x05: The number 5
- 0x06: The number 6
- 0x07: The number 7
- 0x08: The number 8
- 0x09: Blank (nothing is there)
- 0x0A: The mine
- 0x0B: The mine with red X
- 0x0C: The flag
- 0x0D: Exploaded mine

The following is an example of the 8×8 board where 0xFFFF8000 is set to 0, 0xFFFF8001 is set to 1, , 0xFFFF8002 is set to 2, and so on until 0xFFFF800D is set to 13.



The following program is used to create the output as shown above:

```
.text
      la   $s0, 0xffff8000    # Set $s0 to the address of the Minesweeper board
      add  $s1, $zero, $zero  # Counter
      addi $s2, $zero, 14     # Upper Bound
loop: beq  $s1, $s2, done     # Check whether the counter reaches the upper bound
      sb   $s1, 0($s0)        # Store counter (0 - 12) to the address
      addi $s1, $s1, 1        # Increase the counter by 1
      addi $s0, $s0, 1        # Increase the address by 1
      j    loop               # Go back to loop
done: addi $v0, $zero, 10     # Syscalll 10: Terminate the program
      syscall                 # Terminate the program
```

Your MIPS assembly program can also receive an input from the Minesweeper hardware as well. If a user click on the game, change the size, or click the "Reset" button, the value in the register `$t9` wil l be changed. Note that you must set `$t9` to 0 first. Otherwise, the Minesweeper will not change the value of `$t9`. In other words, the structure of your program should look like the following:

```
.text
    # Initialize the game
    :
    add  $t9, $zero, $zero         # Ready to receive an input, set $t9 to 0
wait:
    beq  $t9, $zero, wait          # Wait for $t9 to change to nonzero
    :
    # Process the user input
    :
```

```
add  $t9, $zero, $zero         # Ready to receive an input, set $t9 to 0
j    wait
```

The register `$t9` will contain a value associated with the user input as follows:

- Reset: 0x40MMNRNC where
  - MM is an 8-bit unsigned number representing the number of mines
  - NR is an 8-bit unsigned number representing the number of rows
  - NC is an 8-bit unsigned number representing the number of columns

  Note that the lower 24-bit in this format is also used in the size selector
- 8×8 (10 Mines): 0x200A0808
- 12×12 (15 Mines): 0x200F0C0C
- 16×16 (20 Mines): 0x20141010
- Left click: 0x8000RRCC where
  - RR is an 8-bit unsigned number representing the click row number
  - CC is an 8-bit unsigned number representing the click row number
- Right click: 0x8800RRCC where RR and CC are the same as in Left click

What To Do?

The main goal is to write the MIPS assembly program to drive this Minesweeper game so that it behaves like an actual Minesweeper game. The actual game play will be for the next project. So, for this one, we will only focus on the initializing part of the game.

Note that the Minesweeper hardware is just an interface/display. You need to maintain another two-dimensional array of bytes to store what would be at a specific cell. For simplicity of our discussion, let's call this array `board`. Note that inside a computer memory, a two-dimensional array is simply a one-dimensional array. So, since the largest Minesweeper hardware is 16×16, you should allocate this array `board` as follows:

```
.data
        board   .space   256
```

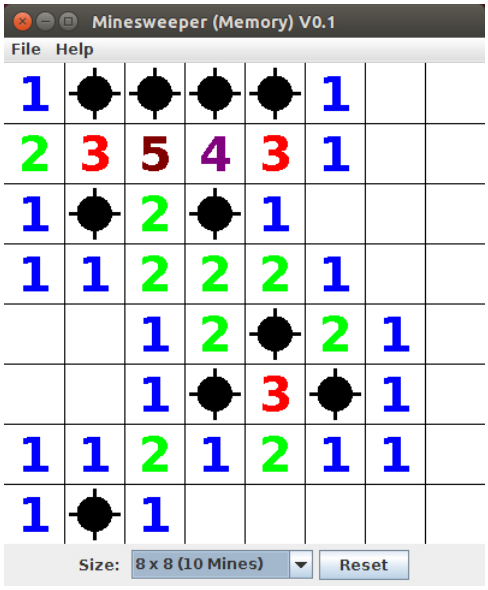
Now, you actually have two Minesweeper, at 0xFFFF8000 is the Minesweeper hardware to display/receive input and at the address referred by `board` to store value of each cell. For simplicity, you may need a function that copy everything from `board` to 0xFFFF8000 for debugging purpose.

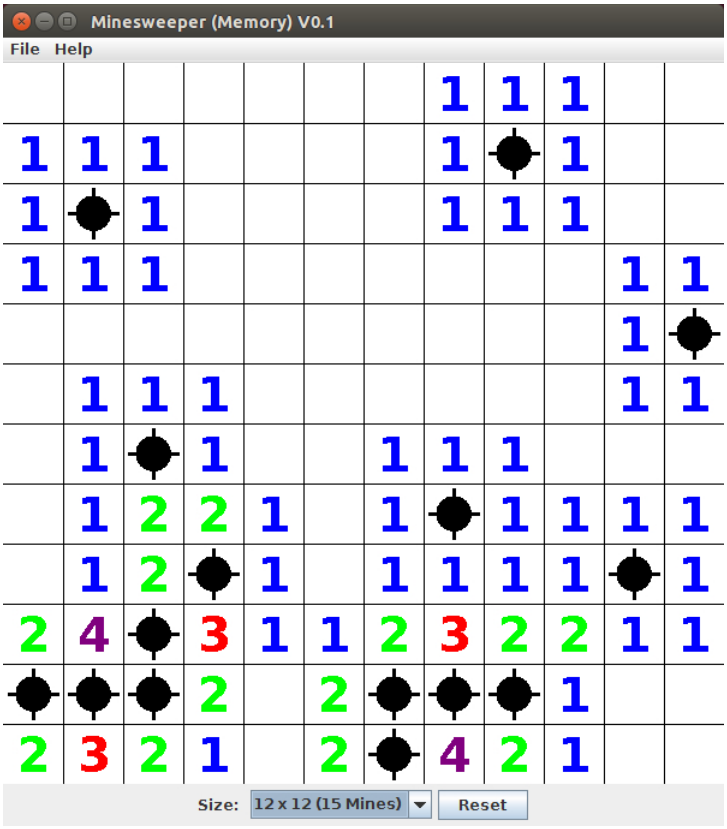
Part I: Randomly Place Mines (30 Points)

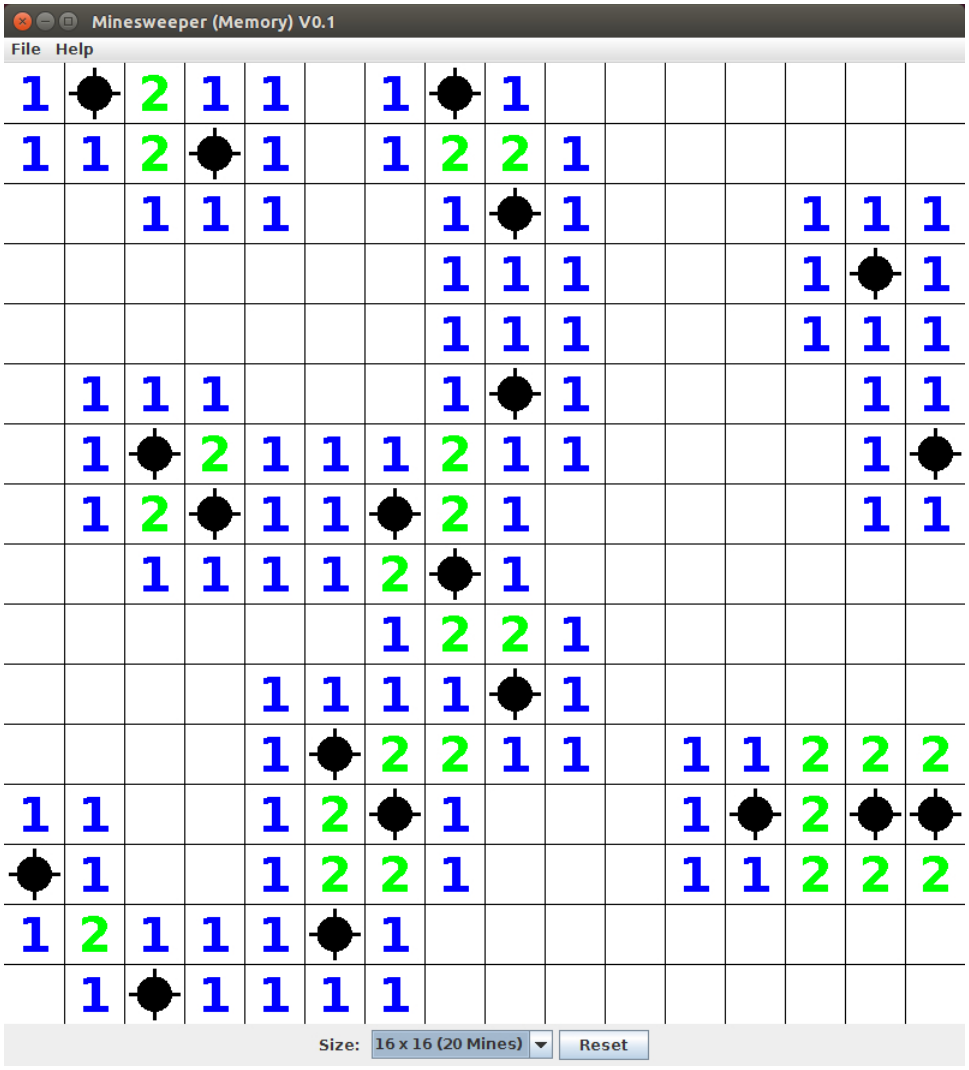
For this part, simply randomly place mines onto the `board` array. You can assume that the initial size is always 8×8 with 10 mines. If a cell at row `r` column `c` has no mine, simply set the value at `board[r][c]` to 0. Next, copy data from the array `board` to 0xFFFF8000 to see the result and wait for input from user (set `$t9` to 0 and wait until `$t9` is changed to non-zero value). If a user change the board size or press the "Reset" button, your program you starts over and randomly place mines on the array `board` and display (copy) again.

Part II: Initialize Cells (40 Points)

A number in the Minesweeper game represents the number of mines around that cell. For this part, open all cells of the Minesweeper game. If a cell contains a mine, set its associated memory location to 10. If a cell contains no mine and there is no mines around it, set its associated memory location to 9 (blank). If a cell contains no mine and there are one or more mines around it, set its associated memory location to the number of mines around that cell. Here are some examples:







Again, if a user click the "Reset" button or change the size, your program should randomly place mines, initialize all cells, and display the result on the Minesweeper hardware.

Part III: Mouse Click (30 Points)

For this part, we are going to check whether your program can read mouse inputs. From Part II, after display the whole board, wait for an input from a user. If a user left-click at row `r` column `c` on the board, your program should display the location `(r, c)` that the user click as well as what is at that cell. For example, if a user left-click at row 5 column 3 that contains a mine, your program should print the following string on the console screen:

```
Left-Click at row 5 column 3: Mine
```

If a user right-click at row 2 column 6, and it contains a number 3, your program should display the following string

```
Right-Click at row 2 column 6: 3
```

If a user left-click at row 0 column 5, and it contains a number nothing, your program should display the following string

```
Right-Click at row 0 column 5: Nothing
```

Submission

The due date of this project is stated on the CourseWeb. Late submissions will not be accepted. You should submit the file `minesweeper_init.asm` via CourseWeb.