CS121

Computer Programming I

FINAL PROJECT REPORT

Minesweeper

Contents:

- **Description and Features**
- **Assumptions**
- **Compilation and Running**
- **Design Overview**
- **Data Structures**
- **Functions**
- **Main Algorithms**
- **User Manual**
- **Sample Runs**

Team Members:

- **Abdelhakeem Osama [34]**
- **Abdelrahman Shams [40]**

Description and Features

This is a console-based implementation of the famous classic puzzle video game, Minesweeper.

As with any Minesweeper, the player is supposed to open all cells of a rectangular board containing mines, without detonating any of them (i.e.: without opening any cell that contains a mine.)

The first cell the player opens always has no mine. Cells that have no mines but are neighboring to cells with mines contain the number of adjacent cells with mines. This way the player can more easily guess where the mines actually are.

Along the way, the player can “flag” some cells as having mines to avoid opening them. Flagged cells can also make gameplay faster. For instance, the player can choose to open an open cell containing a certain number, in which case if the cell is surrounded by an equal number of flags then all neighboring cells are opened except those with flags (whether those neighboring cells contain mines or not), all in one move!

In addition, the player can mark any cell with a question mark, when unsure whether it contains a mine or not, to avoid opening it later. The player can unmark any cell previously marked with a flag or a question mark.

The player loses when a cell with a mine is opened. Otherwise, the player does not win unless all cells with no mines are opened. The player does not necessarily have to flag cells which contain mines.

The main features of our implementation include:

- Rectangular grid with custom dimensions, from 2x2 up to 30x30
- At any time, the game can be saved to a file which can be loaded later to continue the game from the same point
- Sorted table of winners with their names and scores

Assumptions

The game can only run in a POSIX-compliant environment. Therefore, for Windows, you need to compile and run the game from the Cygwin terminal.

The elapsed time starts counting just after the first move in case of a new game. In case of loading a previously-saved game, the elapsed time starts counting immediately, beginning from the point at which the game was saved.

Any valid move is considered in counting the number of moves done, including:

- Opening a cell.
- Flagging a cell.
- Marking a cell.
- Unmarking a cell.

A valid move is one which is done successfully without any errors.

Compilation and Running

- The following two binaries are found in `./bin`:
  - `minesweeper` (64-bit Linux)
  - `minesweeper.exe` (64-bit Windows)
- The project is supposed to be compiled and run on a POSIX-compliant operating system due to our use of input-polling functions (found in poll.h) and signal-handling functions (found in signal.h) from the C POSIX Library. Consequently, it should compile and run on Unix-like operating systems without issues.
  For Windows, we use Cygwin[1], where we invoke our compilation commands and run the executables directly from the Cygwin Terminal.
- Compilation Procedures:
  1. Open your terminal and change to the root directory of the project where the Makefile resides.
  2. Run “make minesweeper” (or simply “make” without specifying any target) to generate the necessary object files and link them into the final game executable under `./bin`

     The following targets are also available for testing game features separately:

     1. `gridtest`
     2. `actionstest`
     3. `gamelooptest`
     4. `savingtest`
     5. `scorestest`
     6. `tests` (make all the previous testing targets)

     All corresponding executables are generated under `./bin`

1 "Linux-like environment for Windows making it possible to port software running on POSIX systems (such as Linux, BSD, and Unix systems)" [https://www.cygwin.com]

Design Overview

1. Input Module

Handles validated input of various data types from the user.

2. Grid Module

Handles grid initialization and display.

3. Game Actions Module

Handles game actions, including:

- Opening a cell.
- Flagging a cell.
- Marking a cell with a question mark.
- Unmarking a cell.

4. Game Loop Module

Handles the main game loop till the game ends in either state: win or lose.

5. Saving Module

Handles saving and loading game state.

6. Scores Module

Handles storage and retrieval of scores of winners.

Data Structures

Cell	
Member	**Description**
`char status`	Represent the status of the cell 0: Closed 1: Open 2: Flagged 3: Marked
`char type`	Type of the cell ‘0’: Empty ‘V’: Empty visited cell (all adjacent cells are open) ‘1’~’8’: Number of adjacent cells with mines ‘*’: Mine ‘!’: Loss mine (mine that made the player lose) ‘M’: Missed mine ‘-’: Incorrect flag

Position	
Member	**Description**
`int x`	Row Number
`int y`	Column Number

ScoreEntry	
Member	**Description**
`char playerName[128]`	Player Name
`int score`	Score

Functions

1. Input Module

- `void flushstdin(void);`
  Reads from stdin until an ENTER character is encountered (which is also read).
  Also used to pause the game until the player presses ENTER.
- `void trimSpaces(char *str);`
  Trims leading and trailing spaces from a string.
- `char ab_getchar(void);`
- `int getint(void);`
- `void getstring(int size, char *str);`
  NOTE: Two characters at the end of the string are reserved for '\n' and '\0', so we actually read (size-2) characters.

2. Grid Module

- `Position getRandomPos(int m, int n);`
  Returns a random position in a grid of (m X n) dimensions.
  NOTE: for this function to work, srand(time(NULL)) must be called at least once before using this function or any of its dependents to seed the random number generator.
- `char positionsEqual(Position a, Position b);`
  Returns 1 if Positions (a) and (b) have the same coordinates, 0 otherwise.
- `char arrayContains(Position *arr, int arrsize, Position key);`
  Returns 1 if (arr) contains the position (key), 0 otherwise.
- `void prepareGrid(int m, int n, Cell grid[m][n]);`
  Zeroes out (empties) all cells and sets them closed.
- `void initGrid(Position firstPos, int m, int n, Cell grid[m][n]);`
  Populates a grid of (m X n) dimensions with mines and numbers. (call prepareGrid first!)
  This function shall be called after the player makes their first move whose position (firstPos) must not contain a mine
  NOTE: Prior validation for (m) and (n) is required!
- `void printGrid(int m, int n, Cell grid[m][n], char *filename, char debug);`
  Prints a grid of (m X n) dimensions.
  To print to (stdout), simply pass an empty string for (filename)
  For debugging mode (i.e.: all cells open), pass a non-zero value for (debug)

3. Game Actions Module

- `void openEmpty(int m, int n, Cell grid[m][n], Position cellPos);`
  Opens an empty cell and all adjacent empty cells recursively.
- `char openCell(int m, int n, Cell grid[m][n], Position cellPos);`
  Opens the cell at (cellPos)
  If the cell is already open, there are two cases:
  1) If the cell contains a number, say (n), and (n) adjacent cells are marked with flags, then all adjacent cells without flags are opened, whether they contain mines or not.
  2) Otherwise, the function returns with error code 1
  If the opened cell is empty, all empty adjacent cells are opened.
  The value of any opened cell that contains a mine is replaced by an exclamation mark.
  Returns 0 on success. Otherwise, an error code is returned as follows:
  1: The cell is already open.
  2: One or more of the opened cells contain mines.
  3: The cell is flagged.
  4: The cell is marked with a question mark.
- `char flagCell(int m, int n, Cell grid[m][n], Position cellPos);`
  Flags the cell at (cellPos)
  Returns 0 on success. Otherwise, an error code is returned as follows:
  1: Cell is open.
  2: Cell is already flagged.
  3: Cell is marked with a question mark.
- `char markCell(int m, int n, Cell grid[m][n], Position cellPos);`
  Marks the cell at (cellPos) with a question mark.
  Returns 0 on success. Otherwise, an error code is returned as follows:
  1: Cell is open.
  2: Cell is flagged.
  3: Cell is already marked with a question mark.
- `char unmarkCell(int m, int n, Cell grid[m][n], Position cellPos);`
  Unmarks the cell at (cellPos)
  Returns an error code as follows:
  1: Flag removed successfully.
  2: Question mark removed successfully.
  3: The cell is not flagged nor marked.

4. Game Loop Module

- `long long calculateScore(int m, int n, int timePassed, int movesDone);`
  Calculates the score of the player after (timePassed) seconds have passed and (movesDone) moves have been done.
- `void printHeader(long long timePassed, int movesDone, int flagged, int marked, long long score);`
  Prints a header line showing the time passed in seconds, number of moves done, and current score, in addition to the number of flagged cells and number of cells marked with a question mark.
- `void getMove(int m, int n, Position *pos);`
  Gets the coordinates of the next move from player.
- `char hasWon(int m, int n, Cell grid[m][n]);`
  Checks the grid to determine if the player has won.
  Returns 1 if player has won, 0 otherwise
- `char startGame(int m, int n, Cell grid[m][n], int timePassed, int movesDone);`
  Starts the game with the following initial conditions:
  (timePassed): Time passed in seconds since the start of the current game
  (0 in case of starting a new game)
  (movesDone): Number of moves done since the start of the current game
  (0 in case of starting a new game)
  The player wins when all the cells that have no mines are opened.
  In case the player loses, incorrect flags should be replaced with '-' and missed mines should be replaced with 'M'
  Returns 0 on winning. Otherwise, an error code is returned as follows:
  1: Player lost.
  2: Player requested to save and exit to main menu.
  3: Player requested to exit to main menu without saving.

5. Saving Module

- `char saveGame(int m, int n, Cell grid[m][n], int timePassed, int movesDone, char *fileName);`
  Saves the status and grid of the current game to a file with the following format:

4 bytes:	**'ABMS' (for AB Team Minesweeper)**
int:	**Time passed since the start of the current game.**
int:	**Moves done since the start of the current game.**
int:	**Number of rows.**
int:	**Number of columns.**
(2*rows*columns) bytes:	**Grid array (elements are of Cell data structure)**

  Returns 0 on success, 1 otherwise.
- `char loadGame_info(int *m, int *n, int *timePassed, int *movesDone, char *fileName);`
  Loads the status of a previously saved game from a file into the variables pointed to by (m), (n), (timePassed), (movesDone)
  Returns 0 on success, 1 otherwise.
- `char loadGame_grid(int m, int n, Cell grid[m][n], char *fileName);`
  Loads the grid of a previously saved game from a file, given that its dimensions are (m X n)
  Returns 0 on success, 1 otherwise.

6. Scores Module

- `char strEqual(char *s1, char *s2);`
  Returns 1 if (s1) is identical to (s2), ignoring letter case. Otherwise, returns 0
- `char readScores(char *fileName, int numEntries, ScoreEntry *scores);`
  Reads table of scores from file and stores it in (scores)
  Return 1 on success, 0 otherwise
- `char addScore(char playerName[128], long long score, char *fileName);`
  Adds the score of the player to the table of scores file.
  Returns 1 on success, 0 otherwise.
- `char displayScores(char *fileName);`
  Reads table of scores from file.
  Returns 1 on success, 0 otherwise.

Scores File Format

4 bytes:	**'ABMS'**
int:	**Number of score entries**
int:	**Length of player name (n)**
(n) bytes:	**Player name**
long long:	**Player score**

.
.
.

Main Algorithms

1. Grid Initialization

```
function initGrid(firstMovePos)
	for each cell in grid
		cell.status = CELL_CLOSED
		cell.type = '0' // Empty
	end for each

	numMines ← 1 + (rows*columns)/10
	usedPositionCount ← 0
	usedPositions ← [] // Empty list

	while(usedPositionCount < numMines)
		generate random position → pos
		if(pos is NOT in usedPositions AND pos != firstMovePos)
			{cell at pos}.type = '*'
			add pos to usedPositions
			usedPositionCount++
			for each adjcell in {adjacent cells to cell at pos}
				if(adjcell.type != '*') // cell contains a number
					adjcell.type++
				end if
			end for each
		end if
	end while
end function
```

2. Opening a Cell

```
function openCell(cellPos)
	cell ← {cell at cellPos}
	switch(cell.status)
		case CELL_FLAGGED
			return CELL_IS_FLAGGED
		end case
		case CELL_MARKED
			return CELL_IS_MARKED
		end case
		case CELL_OPEN
			numFlags ← 0
			for each adjcell in {adjacent cells to cell at cellPos)
				if(adjcell.status == CELL_FLAGGED)
					numFlags++
				end if
			end for each
			if(numFlags == number value of cell.type)
				foundMine ← 0
				allAlreadyOpen ← 1
				for each adjcell in {adjacent cells to cell at cellPos}
					if(adjcell.status == CELL_CLOSED)
						allAlreadyOpen = 0
						adjpos ← {position of adjcell}
						if(openCell(adjpos) == TOUCHED_MINE)
							foundMine = 1
						end if
					end if
				end for each
				if(foundMine == 1)
					return TOUCHED_MINE
				else if(allAlreadyOpen == 1)
					return ALREADY_OPEN
				else
					return 0 // Success
				end if
			else
				return ALREADY_OPEN
			end if
		end case
	end switch
```

```
        cell.status = CELL_OPEN
        switch(cell.type)
                case '*'
                        cell.type = '!'
                        return TOUCHED_MINE
                end case
                case '0' // Empty
                        openEmpty(cellPos)
                end case
        end switch
        return 0 // Success
end function
```

3. Opening an Empty Cell

```
function openEmpty(cellPos)
        cell ← {cell at cellPos}
        cell.status = CELL_OPEN

        if(cell.type == '0') // Empty
                cell.type = 'V' // Empty Visited
                for each adjcell in {adjacent cells to cell}
                        openEmpty({position of adjcell})
                end for each
        end if
end function
```

4. Checking for Win

```
function hasWon()
        for each cell in grid
                if(cell.type != '*' AND cell.status == CELL_CLOSED)  Normal Cell
                        return FALSE
                end if
        end for each
        return TRUE
end function
```

5. Game Loop

```
function gameLoop(timePassed, movesDone)
        end ← GAME_IN_PROGRESS
        flaggedCells ← 0
        markedCells ← 0
        if(movesDone == 0) // First move (also timePassed is 0 then)
                print grid
                get first move from user → move
                initGrid(move)
                if({cell at move}.type = '0') // Empty
                        openEmpty(move)
                else // Numbered
                        open cell at move
                end if
                movesDone++
                get current time → start
                if(hasWon())
                        end = PLAYER_WON
                        timePassed++ // For proper score calculation (to avoid division by 0)
                end if
        else // if not first move (i.e.: loaded game)
                for each cell in grid
                        if(cell.status == CELL_FLAGGED)
                                flaggedCells++
                        else if(cell.status == CELL_MARKED)
                                markedCells++
                        end if
                end for each
                start ← {current time} - timePassed
        end if
        score ← rows^4 * columns^4 / (timePassed*movesDone)
        while(end == GAME_IN_PROGRESS)
                print info and grid
                get action from user → action
                change action to lowercase
```

```
switch(action)
	case ‘o’
		get move from user → move
		cell ← {cell at move}
		result ← openCell(move)
		switch(result)
			case ALREADY_OPEN
				print “The cell is already open!”
			end case
			case CELL_IS_FLAGGED
				print “The cell is flagged!”
			end case
			case CELL_IS_MARKED
				print “The cell is marked!”
			end case
			default
				movesDone++
				if(result == TOUCHED_MINE)
					end = PLAYER_LOST
				else if(hasWon())
					end = PLAYER_WON
				end if
			end case
		end switch
	end case
	case ‘f’
		get move from user → move
		cell ← {cell at move}
		switch(cell.status)
			case CELL_OPEN
				print “The cell is already open!”
			end case
			case CELL_FLAGGED
				print “The cell is already flagged!”
			end case
			case CELL_MARKED
				print “The cell is already marked!”
			end case
			default
				cell.status = CELL_FLAGGED // Flag cell
				flaggedCells++
				movesDone++
			end case
		end switch
	end case
	case ‘m’
		get move from user → move
		cell ← {cell at move}
		switch(cell.status)
			case CELL_OPEN
				print “The cell is already open!”
			end case
			case CELL_FLAGGED
				print “The cell is already flagged!”
			end case
			case CELL_MARKED
				print “The cell is already marked!”
			end case
			default
				cell.status = CELL_MARKED // Mark cell
				markedCells++
				movesDone++
			end case
		end switch
	end case
	case ‘u’
		get move from user → move
		cell ← {cell at move}
		switch(cell.status)
			case CELL_FLAGGED
				cell.status = CELL_CLOSED
				flaggedCells--
				movesDone++
			end case
			case CELL_MARKED
				cell.status = CELL_CLOSED
				markedCells--
```

```
                                        movesDone++
                                end case
                                default
                                        print “The cell is not flagged nor marked!”
                                end case
                        end switch
                end case
                case ‘s’
                        score ← rows^4 * columns^4 / (timePassed*movesDone)
                        timePassed = {current time} - start
                        timePassed → file
                        movesDone → file
                        rows → file
                        columns → file
                        write grid to file
                        print “Game saved successfully!”
                        exit to main menu
                end case
                case ‘q’
                        get confirmation from user → confirm
                        change confirm to lowercase
                        if(confirm == ‘y’)
                                exit to main menu
                        end if
                end case
                default
                        print “Unknown Action!”
                end case
        end switch
        score ← rows^4 * columns^4 / (timePassed*movesDone)
        timePassed = {current time} - start
end while
print info
switch(end)
        case PLAYER_WON
                for each cell in grid
                        if(cell.type == ‘*’)
                                cell.status = CELL_FLAGGED
                        end if
                end for each
                print grid
                print “Congratulations! You have won!”
                get player name → playerName
                playerFound ← FALSE
                for each currentScore in scores file
                        if(currentScore.playerName == playerName)
                                currentScore.scoreValue = score
                                playerFound = TRUE
                                break
                        end if
                end for each
                if(playerFound == FALSE)
                        add player score to file
                end if
        end case
        case PLAYER_LOST
                print “You’ve touched a mine! BOOM! :’(”
                for each cell in grid
                        if(cell.status == CELL_FLAGGED)
                                if(cell.type != ‘*’)
                                        cell.type = ‘-’
                                        cell.status = CELL_OPEN
                                end if
                        else // Not Flagged
                                if(cell.type == ‘*’)
                                        cell.type = ‘M’
                                        cell.status = CELL_OPEN
                                else // Not Mine
                                        cell.status = CELL_OPEN
                                end if
                        end if
                end for each
                print grid
        end case
end switch
end function
```

User Manual

1. Running

Run the game by double-clicking on the executable “minesweeper” found under “./bin” or by invoking it from the terminal.
If you are using Windows, you will have to run the game from Cygwin terminal.

2. Main Menu

The first thing you see is the main menu:

```
Minesweeper Game


        1) New Game
        2) Load Game
        3) Hall of Fame
        4) Exit

Choice:
```

To perform an action, simply type the corresponding number and press ENTER.

3. New Game

Once you choose to start a new game, you are prompted for the dimensions of the grid. The allowed dimensions are from 2x2 up to 30x30. The grid doesn’t have to be a square.
You have to input each dimension separately, do not type both dimensions on one line.

```
Enter the dimensions of the grid
Rows: 5
Columns: 10
```

You are then asked to make your first move. Type the position of the cell you want to open on one line in the given format, then press ENTER.

```
   00 01 02 03 04 05 06 07 08 09
   -- -- -- -- -- -- -- -- -- --
00  X  X  X  X  X  X  X  X  X  X
01  X  X  X  X  X  X  X  X  X  X
02  X  X  X  X  X  X  X  X  X  X
03  X  X  X  X  X  X  X  X  X  X
04  X  X  X  X  X  X  X  X  X  X

Your first move:
Position ([X][SPACE][Y]): 0 9
```

The chosen cell is opened, along with the neighboring cells in case it turns out to be an empty cell. The game screen shows a header with information about the current game, the grid, and supported actions. The elapsed time since the start of the game is printed in seconds, and updated after every action or within 60 seconds if no action is taken. The number of flagged cells, the number of marked cells, and the current score are all printed in the header.

```
Time: 0s        Moves: 1        Flagged: 0     Marked: 0       Score: 0

   00 01 02 03 04 05 06 07 08 09
   -- -- -- -- -- -- -- -- -- --
00  X  X  X  X  X  2
01  X  X  X  X  X  2
02  X  X  X  X  X  2  1  1
03  X  X  X  X  X  X  X  1
04  X  X  X  X  X  X  X  1

       [O]pen Cell
       [F]lag Cell
       [M]ark Cell
       [U]nmark Cell
       [S]ave & Quit Game
       [Q]uit Game
Action:
```

Any valid move you make is counted on you and directly affects your score, including:
- Opening a cell.
- Flagging a cell.
- Marking a cell.
- Unmarking a cell.

The score is calculated according to the following formula:

$$\frac{rows^4 \times columns^4}{time\ passed\ (in\ seconds) \times moves\ done}$$

Every action can be invoked by entering the first letter of it. Actions are case-insensitive, so both `f` and `F` refer to the same thing.

The first four actions are the basic game actions, each one of them asks you for the position of the cell to perform the action on. Positions should be given in the same manner as you do in the first move. (i.e.: the coordinates are typed on one line separated by a space)

```
Time: 10s       Moves: 1        Flagged: 0     Marked: 0       Score: 625000

   00 01 02 03 04 05 06 07 08 09
   -- -- -- -- -- -- -- -- -- --
00
01     1  1  2  1  1  1  1  1
02  1  2  X  X  X  X  X  X  1
03  X  X  X  X  X  X  X  2  1
04  X  X  X  X  X  X  X  1

       [O]pen Cell
       [F]lag Cell
       [M]ark Cell
       [U]nmark Cell
       [S]ave & Quit Game
       [Q]uit Game
Action: f
Position ([X][SPACE][Y]): 2 7
```

The fifth action ([S]ave & Quit Game) saves the current state of the game and exits to the main menu.

The last action ([Q]uit Game) exits to the main menu without saving the current state of the game. You are prompted for confirmation before applying the action.

4. Load Game

This action continues the previously-saved game (if any) from the point at which it was saved.

5. Hall of Fame

The scores of winners are displayed here sorted in descending order.

```
Hall of Fame

             Player                Score
             ------                -----
              Shams                437
              GHOST                346
        Abdelhakeem                229
```

When you win a game, you are asked to enter your name so that your score can be saved.

```
Time: 32s       Moves: 9        Flagged: 2     Marked: 0       Score: 1356

   00 01 02 03 04
   -- -- -- -- --
00     1  1  1
01     1  F  1
02     2  2  2
03  1  2  F  1
04  F  2  1  1

Congratulations! You have won!
Please enter your name: Another Ghost
```

Sample Runs

Below is a sample run which was executed and recorded using the `tee`[2] utility...

grid01.txt:

```
    00 01 02 03 04
    -- -- -- -- --
00   0  2  *  2  0
01   1  3  *  2  0
02   *  2  1  1  0
03   1  1  0  0  0
04   0  0  0  0  0
```

program output:

```
Minesweeper Game

       1) New Game
       2) Load Game
       3) Hall of Fame
       4) Exit

Choice: 1

Enter the dimensions of the grid
Rows: 5
Columns: 5

    00 01 02 03 04
    -- -- -- -- --
00   X  X  X  X  X
01   X  X  X  X  X
02   X  X  X  X  X
03   X  X  X  X  X
04   X  X  X  X  X

Your first move:
Position ([X][SPACE][Y]): 0 0

Time: 0s       Moves: 1       Flagged: 0     Marked: 0      Score: 0

    00 01 02 03 04
    -- -- -- -- --
00      2  X  X  X
01   1  3  X  X  X
02   X  X  X  X  X
03   X  X  X  X  X
04   X  X  X  X  X

       [O]pen Cell
       [F]lag Cell
       [M]ark Cell
       [U]nmark Cell
       [S]ave & Quit Game
       [Q]uit Game
Action:
```

2 "The `tee` command copies standard input to standard output and also to any files given as arguments." [https://www.gnu.org/software/coreutils/manual/html_node/tee-invocation.html]

```
Time: 60s      Moves: 1       Flagged: 0     Marked: 0      Score: 6510

    00 01 02 03 04
    -- -- -- -- --
00      2  X  X  X
01   1  3  X  X  X
02   X  X  X  X  X
03   X  X  X  X  X
04   X  X  X  X  X

       [O]pen Cell
       [F]lag Cell
       [M]ark Cell
       [U]nmark Cell
       [S]ave & Quit Game
       [Q]uit Game
Action: 4 0
Too much characters given!
Try again: o
Position ([X][SPACE][Y]): 4 0

Time: 69s      Moves: 2       Flagged: 0     Marked: 0      Score: 2830

    00 01 02 03 04
    -- -- -- -- --
00      2  F  2
01   1  3  F  2
02   F  2  1  1
03   1  1
04

Congratulations! You have won!
Please enter your name: Abdelhakeem
```

grid02.txt:

```
    00 01 02 03 04 05 06 07 08 09
    -- -- -- -- -- -- -- -- -- --
00   0  1  1  1  0  0  0  0  0  0
01   0  1  *  2  1  0  0  0  0  0
02   0  1  2  *  1  0  0  0  1  1
03   0  1  2  2  1  0  0  1  2  *
04   0  1  *  1  0  0  0  2  *  3
05   0  1  1  1  0  0  0  2  *  2
06   2  2  1  0  0  0  0  1  1  1
07   *  *  2  2  1  1  0  0  0  0
08   2  3  *  2  *  1  1  1  1  0
09   0  1  1  2  1  1  1  *  1  0
```

program output:

```
Minesweeper Game

       1) New Game
       2) Load Game
       3) Hall of Fame
       4) Exit

Choice: 1

Enter the dimensions of the grid
Rows: 10
Columns: 10

    00 01 02 03 04 05 06 07 08 09
    -- -- -- -- -- -- -- -- -- --
00   X  X  X  X  X  X  X  X  X  X
01   X  X  X  X  X  X  X  X  X  X
02   X  X  X  X  X  X  X  X  X  X
03   X  X  X  X  X  X  X  X  X  X
04   X  X  X  X  X  X  X  X  X  X
05   X  X  X  X  X  X  X  X  X  X
06   X  X  X  X  X  X  X  X  X  X
07   X  X  X  X  X  X  X  X  X  X
08   X  X  X  X  X  X  X  X  X  X
09   X  X  X  X  X  X  X  X  X  X
```

```
Your first move:
Position ([X][SPACE][Y]): 5 5

Time: 0s       Moves: 1       Flagged: 0     Marked: 0      Score: 0

    00 01 02 03 04 05 06 07 08 09
    -- -- -- -- -- -- -- -- -- --
00   X  X  X  1
01   X  X  X  2  1
02   X  X  X  X  1           1  1
03   X  X  X  2  1        1  2  X
04   X  X  X  1           2  X  X
05   X  X  1  1           2  X  X
06   X  X  1              1  1  1
07   X  X  2  2  1  1
08   X  X  X  X  X  1  1  1  1
09   X  X  X  X  X  X  X  X  1

        [O]pen Cell
        [F]lag Cell
        [M]ark Cell
        [U]nmark Cell
        [S]ave & Quit Game
        [Q]uit Game
Action: f
Position ([X][SPACE][Y]): 8 5
The cell is already open!

Time: 9s       Moves: 1       Flagged: 0     Marked: 0      Score: 11111111

    00 01 02 03 04 05 06 07 08 09
    -- -- -- -- -- -- -- -- -- --
00   X  X  X  1
01   X  X  X  2  1
02   X  X  X  X  1           1  1
03   X  X  X  2  1        1  2  X
04   X  X  X  1           2  X  X
05   X  X  1  1           2  X  X
06   X  X  1              1  1  1
07   X  X  2  2  1  1
08   X  X  X  X  X  1  1  1  1
09   X  X  X  X  X  X  X  X  1

        [O]pen Cell
        [F]lag Cell
        [M]ark Cell
        [U]nmark Cell
        [S]ave & Quit Game
        [Q]uit Game
Action: f
Position ([X][SPACE][Y]): 8 4

Time: 13s      Moves: 2       Flagged: 1     Marked: 0      Score: 3846153

    00 01 02 03 04 05 06 07 08 09
    -- -- -- -- -- -- -- -- -- --
00   X  X  X  1
01   X  X  X  2  1
02   X  X  X  X  1           1  1
03   X  X  X  2  1        1  2  X
04   X  X  X  1           2  X  X
05   X  X  1  1           2  X  X
06   X  X  1              1  1  1
07   X  X  2  2  1  1
08   X  X  X  X  F  1  1  1  1
09   X  X  X  X  X  X  X  X  1

        [O]pen Cell
        [F]lag Cell
        [M]ark Cell
        [U]nmark Cell
        [S]ave & Quit Game
        [Q]uit Game
Action: o
Position ([X][SPACE][Y]): 8 5
```

```
Time: 16s      Moves: 3       Flagged: 1     Marked: 0      Score: 2083333

    00 01 02 03 04 05 06 07 08 09
    -- -- -- -- -- -- -- -- -- --
00   X  X  X  1
01   X  X  X  2  1
02   X  X  X  X  1           1  1
03   X  X  X  2  1        1  2  X
04   X  X  X  1           2  X  X
05   X  X  1  1           2  X  X
06   X  X  1              1  1  1
07   X  X  2  2  1  1
08   X  X  X  X  F  1  1  1  1
09   X  X  X  X  1  1  1  X  1

       [O]pen Cell
       [F]lag Cell
       [M]ark Cell
       [U]nmark Cell
       [S]ave & Quit Game
       [Q]uit Game
Action: s
Game saved successfully!

Minesweeper Game

       1) New Game
       2) Load Game
       3) Hall of Fame
       4) Exit

Choice: 2


Time: 27s      Moves: 3       Flagged: 1     Marked: 0      Score: 1234567

    00 01 02 03 04 05 06 07 08 09
    -- -- -- -- -- -- -- -- -- --
00   X  X  X  1
01   X  X  X  2  1
02   X  X  X  X  1           1  1
03   X  X  X  2  1        1  2  X
04   X  X  X  1           2  X  X
05   X  X  1  1           2  X  X
06   X  X  1              1  1  1
07   X  X  2  2  1  1
08   X  X  X  X  F  1  1  1  1
09   X  X  X  X  1  1  1  X  1

       [O]pen Cell
       [F]lag Cell
       [M]ark Cell
       [U]nmark Cell
       [S]ave & Quit Game
       [Q]uit Game
Action: o
Position ([X][SPACE][Y]): 5 2
The cell is already open!

Time: 36s      Moves: 3       Flagged: 1     Marked: 0      Score: 925925

    00 01 02 03 04 05 06 07 08 09
    -- -- -- -- -- -- -- -- -- --
00   X  X  X  1
01   X  X  X  2  1
02   X  X  X  X  1           1  1
03   X  X  X  2  1        1  2  X
04   X  X  X  1           2  X  X
05   X  X  1  1           2  X  X
06   X  X  1              1  1  1
07   X  X  2  2  1  1
08   X  X  X  X  F  1  1  1  1
09   X  X  X  X  1  1  1  X  1
```

```
       [O]pen Cell
       [F]lag Cell
       [M]ark Cell
       [U]nmark Cell
       [S]ave & Quit Game
       [Q]uit Game
Action: m
Position ([X][SPACE][Y]): 4 8

Time: 46s      Moves: 4       Flagged: 1     Marked: 1      Score: 543478

    00 01 02 03 04 05 06 07 08 09
    -- -- -- -- -- -- -- -- -- --
00   X  X  X  1
01   X  X  X  2  1
02   X  X  X  X  1           1  1
03   X  X  X  2  1        1  2  X
04   X  X  X  1           2  ?  X
05   X  X  1  1           2  X  X
06   X  X  1              1  1  1
07   X  X  2  2  1  1
08   X  X  X  X  F  1  1  1  1
09   X  X  X  X  1  1  1  X  1

       [O]pen Cell
       [F]lag Cell
       [M]ark Cell
       [U]nmark Cell
       [S]ave & Quit Game
       [Q]uit Game
Action: o
Position ([X][SPACE][Y]): 4 9

Time: 50s      Moves: 5       Flagged: 1     Marked: 1      Score: 400000

    00 01 02 03 04 05 06 07 08 09
    -- -- -- -- -- -- -- -- -- --
00   X  X  X  1
01   X  X  X  2  1
02   X  X  X  X  1           1  1
03   X  X  X  2  1        1  2  X
04   X  X  X  1           2  ?  3
05   X  X  1  1           2  X  X
06   X  X  1              1  1  1
07   X  X  2  2  1  1
08   X  X  X  X  F  1  1  1  1
09   X  X  X  X  1  1  1  X  1

       [O]pen Cell
       [F]lag Cell
       [M]ark Cell
       [U]nmark Cell
       [S]ave & Quit Game
       [Q]uit Game
Action: o
Position ([X][SPACE][Y]): 3 9

Time: 54s      Moves: 6       Flagged: 1     Marked: 1      Score: 308641

You've touched a mine! BOOM! :'(

    00 01 02 03 04 05 06 07 08 09
    -- -- -- -- -- -- -- -- -- --
00      1  1  1
01      1  M  2  1
02      1  2  M  1           1  1
03      1  2  2  1        1  2  !
04      1  M  1           2  M  3
05      1  1  1           2  M  2
06   2  2  1              1  1  1
07   M  M  2  2  1  1
08   2  3  M  2  F  1  1  1  1
09      1  1  2  1  1  1  M  1
```

grid03.txt:

```
    00 01 02
    -- -- --
00   0  0  0
01   1  1  1
02   1  *  1
```

program output:

```
Minesweeper Game

       1) New Game
       2) Load Game
       3) Hall of Fame
       4) Exit

Choice: 1

Enter the dimensions of the grid
Rows: 3
Columns: 3

    00 01 02
    -- -- --
00   X  X  X
01   X  X  X
02   X  X  X

Your first move:
Position ([X][SPACE][Y]): 0 0

Time: 0s       Moves: 1       Flagged: 0     Marked: 0      Score: 0

    00 01 02
    -- -- --
00
01   1  1  1
02   X  X  X

       [O]pen Cell
       [F]lag Cell
       [M]ark Cell
       [U]nmark Cell
       [S]ave & Quit Game
       [Q]uit Game
Action: o
Position ([X][SPACE][Y]): 2 0

Time: 15s      Moves: 2       Flagged: 0     Marked: 0      Score: 218

    00 01 02
    -- -- --
00
01   1  1  1
02   1  X  X

       [O]pen Cell
       [F]lag Cell
       [M]ark Cell
       [U]nmark Cell
       [S]ave & Quit Game
       [Q]uit Game
Action: o
Position ([X][SPACE][Y]): 2 2

Time: 16s      Moves: 3       Flagged: 0     Marked: 0      Score: 136

    00 01 02
    -- -- --
00
01   1  1  1
02   1  F  1

Congratulations! You have won!
Please enter your name: Shams
```

```
Minesweeper Game

        1) New Game
        2) Load Game
        3) Hall of Fame
        4) Exit

Choice: 3

Hall of Fame

           Player              Score
           ------              -----
      Abdelhakeem              2830
            Shams              136


Minesweeper Game

        1) New Game
        2) Load Game
        3) Hall of Fame
        4) Exit

Choice: 4
```