Sabancı University

Faculty of Engineering and Natural Sciences
CS204 Advanced Programming
Spring 2022

Homework 6 – Cell Ownership Operations on Board

Due: 24/05/2022, Tuesday, 21:00

PLEASE NOTE:

Your program should be a robust one such that you have to consider all relevant programmer mistakes and extreme cases; you are expected to take actions accordingly!

You can NOT collaborate with your friends and discuss solutions. You have to write down the code on your own. Plagiarism will not be tolerated!

Introduction

In this homework, you are asked to implement two classes using *object sharing* and *templated class* concepts of C++. The main program implementation, using classes, is given to you and you are expected to write two classes: *Board* class and *Player* class. We are going to explain these classes in more details in the following sections. Before that, we start with a general description of the program.

Overview of Cell Ownership Operations on a Board

In the program there are two players that operate over a two-dimensional board with `rows×columns` cells. Each cell in that board carries a value of a *templated* type and it has an integer label that indicates the current owner of that cell. During the execution of the program, any cell can be owned by any of the players given that it is not currently owned by the other player. Also, any player can unclaim his/her ownership of a certain cell or all cells that he/she previously owned. There are some other functions as well. Please read the specifications below.

Inputs and Input Checks

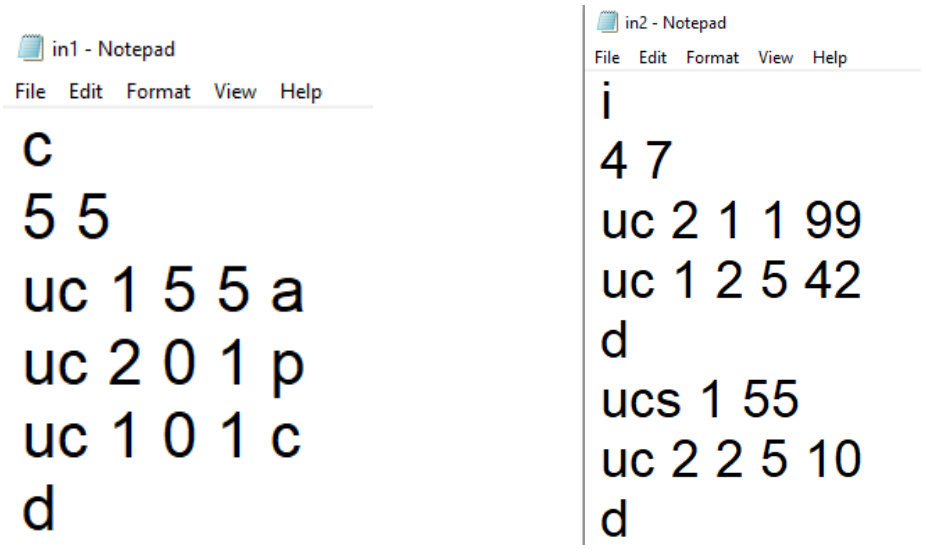
All the data will be read from a text file. First, the name of the input .txt file is entered by the user in the main function. The first line of this file indicates the data type of the values that the cells will carry: `i` for integer and `c` for character. Then, the second line in the file contains the number of rows and number of columns of the board. Using these dimensions, a board object is created in main.

The rest of the file contains commands, one command per line, along with their arguments, if any. These commands will be explained in detail in the subsequent sections.

You may assume that the input file is given to you in the correct format. That means, the first line contains either letter `i` or letter `c` and they are in lowercase. The second line contains two

positive integers where the first integer represents the number of rows, and the second integer represents the number of columns. The necessary parameter checks for the commands are done in main by us. Therefore, you do not need to make any input check for the content of the input file.

A partial screenshot from two sample input files are given below. The one on the left is for a character board with 5 rows and 5 columns, and the one on the right is for an integer board with 4 rows and 7 columns.



As you see in the samples above, the type of cell values of the board can be either `int` (screenshot on the right) or `char` (screenshot on the left). Your program should handle this without using different functions or classes for these different types. In other words, the classes that you will implement must be **templated**.

The row and column indexing starts with zero and upper-left element is at (0,0). In all of the commands that refer to a cell, the first index is for the row and the second one is for the column. You can assume that integers are used for row and column values. We already make the range checks in main, so you do not need to make the range check in the member functions that you will implement.

Commands and their arguments

1- **"uc":** (update cell) command receives 4 arguments: a player ID, a row index of a cell (`i`), a column index of a cell (`j`) and a given value. It should update the cell at the given indices with the given value, if that the cell is owned by the same player or is not currently owned by the other player. It also assigns the ownership of that cell to the player with the given ID.

2- **"ucs":** (update cells) command receives 2 arguments: a player ID and a value. It populates all the cells that the player ID owns with the provided value.

3- **"unc":** (unclaim cell) command receives 3 arguments: a player ID, a row index of a cell (`i`), and a column index of a cell (`j`). It unclaims the ownership of that cell (i.e., set it back

to zero), given that it is currently owned by the player with the specified ID. Note: the content (i.e., the value) that the cell contains will not be changed here.

4- **"uncs":** (unclaim cells) command receives one argument: a player ID. It unclaims the ownership of all the cells that this player owns (by setting to 0). Note again that the values of the cells will remain intact.

5- **"ko":** (know owner) command receives 2 arguments, a row index of a cell (`i`), and a column index of a cell (`j`). It should return the ID of the current owner of the cell at the specified indices.

6- **"d":** (display) command receives no arguments. It should display the board in a 2D matrix format (see the sample runs). It also displays the number of cells owned by each player as well.

The necessary member function calls that implement these commands and the related error messages have been incorporated in the `main` function, which is provided to you in main.cpp. What you have to do is to implement the `Board` and `Player` classes accordingly as detailed below.

`display` free function

In main.cpp file, there is a free function, named `display`, that is being called when the command **"d"** is used. We would like to grab your attention to the way that function is being called. It takes a board as a value parameter (not reference). This is done intentionally to enforce you to implement a deep copy constructor, because otherwise, your program would probably crash.

`Board` Class

The `Board` class will be used to create a board to which the players will access. A board is represented by a matrix which is a private data member of the class (pointer to pointer). The constructor of the `Board` class will create the matrix dynamically with the given row and column values as parameters. Moreover, the matrix will be designed such that it will be able to keep any type of value on the cells (such as int, char, etc.). Since you also need to keep the value on the cell and the owner of that cell (owner is of type integer), you may consider using a struct as the element type of the matrix; but here notice that such a struct should also be templated.

Now, we will give constructor and some member function explanations of the `Board` class. You are allowed to add other member functions depending on your choice of implementation.

Parametric Constructor: Constructor of the `Board` class takes two integer parameters: number of rows and number of columns of the board. You are going to construct the board matrix here using dynamic memory allocation; however, you will not read a file or take keyboard input for matrix initialization in the constructor. Actually, you will initialize only the owners of the cells to 0 (means no owner); the value fields of the cells will be left uninitialized (we later use another member function in main to set the values; see the main.cpp file).

Deep Copy Constructor: By definition a copy constructor takes a const-ref parameter of `Board` type and generates a **deep** copy of it.

Destructor: By definition, destructor deallocates all of the dynamically allocated memory of the class. It'd be safe to make the pointers `NULL` after deallocation.

`displayBoard:` The `displayBoard` function does not take any parameter. It only displays the current state of the board. This function should display the board in a 2D matrix format where each cell is shown as a tuple where the first element is the cell content and the second element is the owner of the cell. Please see the sample runs for example format.

`updateBoardCells`: The `updateBoardCells` function takes a player ID and a value as two parameters and populate all the cells owned by that player with that value.

`getPlayerCellsCount:` The `getPlayerCellsCount` takes one parameter, a player ID, and returns the number of cells that player owns.

`knowTheOwner:` the `knowTheOwner` function takes two parameters, the indices of a cell (row and column) and it returns the owner of that cell. It should return zero if the cell is not currently owned by any of the players; actually you do not need to make an extra control for this since the cells are initialized/reset to zero when not owned.

The above functions are explicitly used in program implementation given to you in main.cpp. Other than these member functions, you may need to add and implement some other functions to manage the board and their cells. This is due to the fact that you will implement another class for Player and this Player class will need to access board. Since the use of friend functions and friend classes are **not** allowed in this homework, in the implementation of the Player class, you will need to access board via some extra Board member functions.

Player Class

Player class will be used to create players of the program. There will be two players operating on the same board. Thus, player objects must share a board object using *object sharing* concept of C++ as we have seen in class. We have seen two different methods for object sharing in class; due to our main function implementation, which is provided to you, you must use the reference variable method.

Now, we will give member function explanations of player class. You are allowed to add other member functions depending on your choice of implementation.

Parametric Constructor: Constructor of the player class takes only one parameter, which is the reference to a board object that will be played on. In the constructor, you should also assign a value to the player ID starting with 1 (first player 1, second player 2, etc.); however, it is not a parameter. You should use the concept of static data member of a class to give incremental IDs for the players (see lecture notes; there is a very useful example there).

`updateCell:` The `updateCell` function takes three integer parameters, which are row and column indices of the cell on the board and the value that will be used to fill that cell's content if the cell is currently owned by the player calling the function or is currently not owned (i.e. if owner ID is 0). If the ownership condition is satisfied, then the cell is updated and the function returns true. Otherwise, no update is done and the function returns false.

`updateMyCells`: The `updateMyCells` function takes one parameter that indicates the value with which the player wants to populate all of their owned cells. The function updates all of the cell of the player calling the function with the parameter value.

`unclaimOwnership`: The `unclaimOwnership` function takes two parameters which are row and column indices of the cell on the board. This function unclaims the ownership of the player from that cell by setting the owner ID to zero, if it is currently owned by that player. In this case, the function should also return true. If the cell is not owned by the player calling the function, then no unclaiming operation is done and the function returns false.

`unclaimOwnerships`: The `unclaimOwnerships` function takes no parameters and it unclaims the ownership of all the cells owned by that (caller) player object.

Important Rule about using `friend`

In this homework, you are **not** allowed to use **`friend`** keyword, which means the use of friend functions and friend classes are prohibited. Our aim by this restriction is not to make your life miserable, but to enforce you to proper object oriented design and implementation.

Provided files

Together with this homework package, we provide some extra files to you.

main.cpp: This file contains the `main` function, which contain the program implementation by using related class functions. In this homework, our aim is to reinforce object oriented design capabilities; thus, we did not want you to deal with the class usage, but focus on their design and implementation. Please examine these functions to understand how the classes are used. We will test your codes with this main.cpp with different inputs. You are not allowed to make any modifications in this file other than adding your #include lines; you have to use other files for class definitions and implementations.

txt files: We also provide some sample input files.

Files to be Submitted and the Use of Templates

You will submit total of five (5) files. Two of them are the headers files; one for player class and the other is for board class (define cell struct here). Three files are cpp files. One is the main.cpp that will remain intact other than the #includes. One of the other cpp files is for the player class and the other is for the board class.

Since the classes will be templated, you will need to solve the *fundamental dilemma* mentioned in the lecture notes (5-templated.ppt, pages 24-25). You have to apply "Solution 2" mentioned there.

A General Remark about Object Oriented Design

You need to analyze the requirements carefully and make a good object oriented design for the classes that you will develop in this homework. In this context, you have to determine the data members and member functions of each class correctly. You have to apply the object sharing principles correctly and use the templates properly. We will evaluate your object oriented design as well. Moreover, you are not allowed to use friend class or friend functions in your design.

Please see the previous homework specifications for the other important rules and the submission guidelines

Sample Runs

Some sample runs are given below, but these are not comprehensive, therefore you have to consider **all possible cases** to get full mark. Especially try different, but valid, file contents (other than the ones given in the sample runs) and extreme cases. Inputs are shown in **bold** and *italic*.

Please do not try to understand the requirements by checking out the sample runs only. They may give you just an idea, but you have to read the requirements from the document and the main.cpp to fully understand what to do.

Sample Run 1:

```
Please enter the file name: in1.txt
Invalid arguments!
Updated (0,1) cell by player 2 with value p
Error! You cannot update other player's cells.
Displaying the board: Each cell is shown as tuple (CURRENT OWNER ID, VALUE):
        0       1       2       3       4
0       (0,-)   (2,p)   (0,-)   (0,-)   (0,-)
1       (0,-)   (0,-)   (0,-)   (0,-)   (0,-)
2       (0,-)   (0,-)   (0,-)   (0,-)   (0,-)
3       (0,-)   (0,-)   (0,-)   (0,-)   (0,-)
4       (0,-)   (0,-)   (0,-)   (0,-)   (0,-)
The numbers of the owned cells by the two players are:
Player1: 0      Player2: 1

Updated all cells of player 1 with value b
Displaying the board: Each cell is shown as tuple (CURRENT OWNER ID, VALUE):
        0       1       2       3       4
0       (0,-)   (2,p)   (0,-)   (0,-)   (0,-)
1       (0,-)   (0,-)   (0,-)   (0,-)   (0,-)
2       (0,-)   (0,-)   (0,-)   (0,-)   (0,-)
3       (0,-)   (0,-)   (0,-)   (0,-)   (0,-)
4       (0,-)   (0,-)   (0,-)   (0,-)   (0,-)
The numbers of the owned cells by the two players are:
Player1: 0      Player2: 1

Updated (2,2) cell by player 2 with value z
Error! You can only unclaim one of your cells.
Displaying the board: Each cell is shown as tuple (CURRENT OWNER ID, VALUE):
        0       1       2       3       4
0       (0,-)   (2,p)   (0,-)   (0,-)   (0,-)
1       (0,-)   (0,-)   (0,-)   (0,-)   (0,-)
2       (0,-)   (0,-)   (2,z)   (0,-)   (0,-)
3       (0,-)   (0,-)   (0,-)   (0,-)   (0,-)
4       (0,-)   (0,-)   (0,-)   (0,-)   (0,-)
The numbers of the owned cells by the two players are:
Player1: 0      Player2: 2

Error! Wrong Player ID
```

```
Updated all cells of player 2 with value e
Displaying the board: Each cell is shown as tuple (CURRENT OWNER ID, VALUE):
        0       1       2       3       4
0       (0,-)   (2,e)   (0,-)   (0,-)   (0,-)
1       (0,-)   (0,-)   (0,-)   (0,-)   (0,-)
2       (0,-)   (0,-)   (2,e)   (0,-)   (0,-)
3       (0,-)   (0,-)   (0,-)   (0,-)   (0,-)
4       (0,-)   (0,-)   (0,-)   (0,-)   (0,-)
The numbers of the owned cells by the two players are:
Player1: 0      Player2: 2

Deallocating memory and ending program...
```

Sample Run 2:

```
Please enter the file name: in2.txt
Updated (1,1) cell by player 2 with value 99
Displaying the board: Each cell is shown as tuple (CURRENT OWNER ID, VALUE):
        0       1       2       3       4       5       6
0       (0,-1)  (0,-1)  (0,-1)  (0,-1)  (0,-1)  (0,-1)  (0,-1)
1       (0,-1)  (2,99)  (0,-1)  (0,-1)  (0,-1)  (0,-1)  (0,-1)
2       (0,-1)  (0,-1)  (0,-1)  (0,-1)  (0,-1)  (0,-1)  (0,-1)
3       (0,-1)  (0,-1)  (0,-1)  (0,-1)  (0,-1)  (0,-1)  (0,-1)
The numbers of the owned cells by the two players are:
Player1: 0      Player2: 1

Updated (2,5) cell by player 1 with value 42
Displaying the board: Each cell is shown as tuple (CURRENT OWNER ID, VALUE):
        0       1       2       3       4       5       6
0       (0,-1)  (0,-1)  (0,-1)  (0,-1)  (0,-1)  (0,-1)  (0,-1)
1       (0,-1)  (2,99)  (0,-1)  (0,-1)  (0,-1)  (0,-1)  (0,-1)
2       (0,-1)  (0,-1)  (0,-1)  (0,-1)  (0,-1)  (1,42)  (0,-1)
3       (0,-1)  (0,-1)  (0,-1)  (0,-1)  (0,-1)  (0,-1)  (0,-1)
The numbers of the owned cells by the two players are:
Player1: 1      Player2: 1

Updated all cells of player 1 with value 55
Error! You cannot update other player's cells.
Displaying the board: Each cell is shown as tuple (CURRENT OWNER ID, VALUE):
        0       1       2       3       4       5       6
0       (0,-1)  (0,-1)  (0,-1)  (0,-1)  (0,-1)  (0,-1)  (0,-1)
1       (0,-1)  (2,99)  (0,-1)  (0,-1)  (0,-1)  (0,-1)  (0,-1)
2       (0,-1)  (0,-1)  (0,-1)  (0,-1)  (0,-1)  (1,55)  (0,-1)
3       (0,-1)  (0,-1)  (0,-1)  (0,-1)  (0,-1)  (0,-1)  (0,-1)
The numbers of the owned cells by the two players are:
Player1: 1      Player2: 1

Error! You can only unclaim one of your cells.
Displaying the board: Each cell is shown as tuple (CURRENT OWNER ID, VALUE):
        0       1       2       3       4       5       6
0       (0,-1)  (0,-1)  (0,-1)  (0,-1)  (0,-1)  (0,-1)  (0,-1)
1       (0,-1)  (2,99)  (0,-1)  (0,-1)  (0,-1)  (0,-1)  (0,-1)
2       (0,-1)  (0,-1)  (0,-1)  (0,-1)  (0,-1)  (1,55)  (0,-1)
3       (0,-1)  (0,-1)  (0,-1)  (0,-1)  (0,-1)  (0,-1)  (0,-1)
The numbers of the owned cells by the two players are:
Player1: 1      Player2: 1

Player 1 has unclaimed his/her (2,5) cell
Displaying the board: Each cell is shown as tuple (CURRENT OWNER ID, VALUE):
```

```
        0       1       2       3       4       5       6
0       (0,-1)  (0,-1)  (0,-1)  (0,-1)  (0,-1)  (0,-1)  (0,-1)
1       (0,-1)  (2,99)  (0,-1)  (0,-1)  (0,-1)  (0,-1)  (0,-1)
2       (0,-1)  (0,-1)  (0,-1)  (0,-1)  (0,-1)  (0,55)  (0,-1)
3       (0,-1)  (0,-1)  (0,-1)  (0,-1)  (0,-1)  (0,-1)  (0,-1)
The numbers of the owned cells by the two players are:
Player1: 0      Player2: 1

Updated (2,5) cell by player 2 with value 10
Displaying the board: Each cell is shown as tuple (CURRENT OWNER ID, VALUE):
        0       1       2       3       4       5       6
0       (0,-1)  (0,-1)  (0,-1)  (0,-1)  (0,-1)  (0,-1)  (0,-1)
1       (0,-1)  (2,99)  (0,-1)  (0,-1)  (0,-1)  (0,-1)  (0,-1)
2       (0,-1)  (0,-1)  (0,-1)  (0,-1)  (0,-1)  (2,10)  (0,-1)
3       (0,-1)  (0,-1)  (0,-1)  (0,-1)  (0,-1)  (0,-1)  (0,-1)
The numbers of the owned cells by the two players are:
Player1: 0      Player2: 2

Error! You can only unclaim one of your cells.
Updated (3,3) cell by player 2 with value 15
Displaying the board: Each cell is shown as tuple (CURRENT OWNER ID, VALUE):
        0       1       2       3       4       5       6
0       (0,-1)  (0,-1)  (0,-1)  (0,-1)  (0,-1)  (0,-1)  (0,-1)
1       (0,-1)  (2,99)  (0,-1)  (0,-1)  (0,-1)  (0,-1)  (0,-1)
2       (0,-1)  (0,-1)  (0,-1)  (0,-1)  (0,-1)  (2,10)  (0,-1)
3       (0,-1)  (0,-1)  (0,-1)  (2,15)  (0,-1)  (0,-1)  (0,-1)
The numbers of the owned cells by the two players are:
Player1: 0      Player2: 3

Player 2 has unclaimed all of their cells
Displaying the board: Each cell is shown as tuple (CURRENT OWNER ID, VALUE):
        0       1       2       3       4       5       6
0       (0,-1)  (0,-1)  (0,-1)  (0,-1)  (0,-1)  (0,-1)  (0,-1)
1       (0,-1)  (0,99)  (0,-1)  (0,-1)  (0,-1)  (0,-1)  (0,-1)
2       (0,-1)  (0,-1)  (0,-1)  (0,-1)  (0,-1)  (0,10)  (0,-1)
3       (0,-1)  (0,-1)  (0,-1)  (0,15)  (0,-1)  (0,-1)  (0,-1)
The numbers of the owned cells by the two players are:
Player1: 0      Player2: 0

Deallocating memory and ending program...
```

Sample Run 3:

```
Please enter the file name: in3.txt
Updated all cells of player 2 with value t
Invalid arguments!
Displaying the board: Each cell is shown as tuple (CURRENT OWNER ID, VALUE):
        0       1       2       3       4       5
0       (0,-)   (0,-)   (0,-)   (0,-)   (0,-)   (0,-)
1       (0,-)   (0,-)   (0,-)   (0,-)   (0,-)   (0,-)
2       (0,-)   (0,-)   (0,-)   (0,-)   (0,-)   (0,-)
3       (0,-)   (0,-)   (0,-)   (0,-)   (0,-)   (0,-)
4       (0,-)   (0,-)   (0,-)   (0,-)   (0,-)   (0,-)
5       (0,-)   (0,-)   (0,-)   (0,-)   (0,-)   (0,-)
6       (0,-)   (0,-)   (0,-)   (0,-)   (0,-)   (0,-)
7       (0,-)   (0,-)   (0,-)   (0,-)   (0,-)   (0,-)
8       (0,-)   (0,-)   (0,-)   (0,-)   (0,-)   (0,-)
The numbers of the owned cells by the two players are:
```

```
Player1: 0      Player2: 0

Updated (1,2) cell by player 1 with value g
Updated (0,4) cell by player 1 with value k
Updated (3,4) cell by player 2 with value m
Displaying the board: Each cell is shown as tuple (CURRENT OWNER ID, VALUE):
        0       1       2       3       4       5
0       (0,-)   (0,-)   (0,-)   (0,-)   (1,k)   (0,-)
1       (0,-)   (0,-)   (1,g)   (0,-)   (0,-)   (0,-)
2       (0,-)   (0,-)   (0,-)   (0,-)   (0,-)   (0,-)
3       (0,-)   (0,-)   (0,-)   (0,-)   (2,m)   (0,-)
4       (0,-)   (0,-)   (0,-)   (0,-)   (0,-)   (0,-)
5       (0,-)   (0,-)   (0,-)   (0,-)   (0,-)   (0,-)
6       (0,-)   (0,-)   (0,-)   (0,-)   (0,-)   (0,-)
7       (0,-)   (0,-)   (0,-)   (0,-)   (0,-)   (0,-)
8       (0,-)   (0,-)   (0,-)   (0,-)   (0,-)   (0,-)
The numbers of the owned cells by the two players are:
Player1: 2      Player2: 1

Updated (7,4) cell by player 2 with value b
Displaying the board: Each cell is shown as tuple (CURRENT OWNER ID, VALUE):
        0       1       2       3       4       5
0       (0,-)   (0,-)   (0,-)   (0,-)   (1,k)   (0,-)
1       (0,-)   (0,-)   (1,g)   (0,-)   (0,-)   (0,-)
2       (0,-)   (0,-)   (0,-)   (0,-)   (0,-)   (0,-)
3       (0,-)   (0,-)   (0,-)   (0,-)   (2,m)   (0,-)
4       (0,-)   (0,-)   (0,-)   (0,-)   (0,-)   (0,-)
5       (0,-)   (0,-)   (0,-)   (0,-)   (0,-)   (0,-)
6       (0,-)   (0,-)   (0,-)   (0,-)   (0,-)   (0,-)
7       (0,-)   (0,-)   (0,-)   (0,-)   (2,b)   (0,-)
8       (0,-)   (0,-)   (0,-)   (0,-)   (0,-)   (0,-)
The numbers of the owned cells by the two players are:
Player1: 2      Player2: 2

Player 1 has unclaimed all of their cells
Displaying the board: Each cell is shown as tuple (CURRENT OWNER ID, VALUE):
        0       1       2       3       4       5
0       (0,-)   (0,-)   (0,-)   (0,-)   (0,k)   (0,-)
1       (0,-)   (0,-)   (0,g)   (0,-)   (0,-)   (0,-)
2       (0,-)   (0,-)   (0,-)   (0,-)   (0,-)   (0,-)
3       (0,-)   (0,-)   (0,-)   (0,-)   (2,m)   (0,-)
4       (0,-)   (0,-)   (0,-)   (0,-)   (0,-)   (0,-)
5       (0,-)   (0,-)   (0,-)   (0,-)   (0,-)   (0,-)
6       (0,-)   (0,-)   (0,-)   (0,-)   (0,-)   (0,-)
7       (0,-)   (0,-)   (0,-)   (0,-)   (2,b)   (0,-)
8       (0,-)   (0,-)   (0,-)   (0,-)   (0,-)   (0,-)
The numbers of the owned cells by the two players are:
Player1: 0      Player2: 2

Deallocating memory and ending program...
```

Good Luck!
Albert Levi, Ahmed Salem