

Hints for Task B of Game 1024

1. Name source code Board.cpp. Note that **board.cpp** is not the same as **Board.cpp** since Linux is case-sensitive.
2. Do not include main function in Board.cpp. You should enclose main function in TestBoard.cpp, but no need to submit TestBoard.cpp to gradescope.

Task B (20 points)

Submit **only** Board.cpp, with destructor, print, selectRandomCell and noAdjacentSameValue methods.

Define code to select a random cell with value 1. That is, define selectRandomCell method of Board.cpp. Students may have questions on the parameters in the header of
void selectRandomCell(int& row, int& col)

Parameters row and col are row- and column-index of the chosen random empty cell. Use int& to pass by reference since we need to return both row and column. In our program, the returned row- and column-index are not used. When we improve our code later on, say, set text color in the newly chosen random cell to be red, we will need such information.

- (2.1) **Comments all** constructors in the submission of Task A. But keep destructor ~Board() and method print and any helper function for print.
Reason: gradescope autograde scripts provide a correct version of constructors, in case your constructors are incorrect in Task A, methods selectRandomCell and noAdjacentSameValue in your Task B can still be graded separately.
- (2.2) Define method noAdjacentSameValue, which checks whether the game can be over or not. That is, if all cells are filled up (no empty cell, which represents zero, in the panel) and no two adjacent cells have the same value, return true, otherwise, return false.

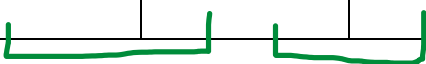
Note that, this method is called **after** finding every cell in the board is filled up. We can declare method noAdjacentSameValue as private, so the user of Board class cannot call this method directly, only selectRandomCell method can call method noAdjacentSameValue.

If we detect that every cell is filled up, we only need to check whether there are adjacent same values in horizontal and/or vertical direction and current max < target to conclude that game is over.

Hints: check whether there are adjacent same values in horizontal or vertical direction. That is,

- (1) In horizontal direction, for each row, check whether there are one pair of identical-value cells, if there is one, return false.

		...	



For a row with row index i ,

Column index runs from 0 to $\text{numCols} - 1$.

The first adjacent pair has column indices 0 and 1.

The second adjacent pair has column indices 1 and 2.

...

The last adjacent pair has column indices $\text{numCols} - 2$ and $\text{numCols} - 1$.

Since each adjacent pair (in horizontal direction) can be represented by left index and right index and these two indices are not independent (know one, know the other). That is, if the left index is j , then the right index is $j + 1$.

We only need to represent either left or right index, but not both. Without loss of generality, we choose left indices of each adjacent pair.

for a row with row index i ,

for ($\text{int } j = 0; j < \text{numCols} - 1; j++$)

Compare $\text{panel}[i][j]$ and $\text{panel}[i][j + 1]$ have the same value or not

Pseudocode to check adjacent pairs in every row.

for ($\text{int } i = 0; i < \text{numRows}; i++$)

for ($\text{int } j = 0; j < \text{numCols} - 1; j++$)

Compare $\text{panel}[i][j]$ and $\text{panel}[i][j + 1]$ have the same value or not.

if $\text{panel}[i][j]$ has the same value as of $\text{panel}[i][j + 1]$,

then there are adjacent same value cells, return false,
meaning the game is not over yet.

Otherwise ($\text{panel}[i][j]$ is not the same as $\text{panel}[i][j + 1]$),

There is no conclusion yet, we still need to check the rest cells,

$\text{panel}[i][j]$ does not equal to $\text{panel}[i][j + 1]$ only means

this specific adjacent pair do not share the same value,

but there might be same value adjacent pairs

somewhere else. Warning: do not return true in the else part.

- (2) In vertical direction, for each column, check whether there are one pair of identical-value cells, if there is one, return false. Note that two adjacent cells in vertical direction are represented by $\text{panel}[i][j]$ and $\text{panel}[i + 1][j]$, where i is row index ranging from 0 to $\text{numRows} - 2$ and j is column index ranging from 0 to $\text{numCols} - 1$.

Pseudocode to check adjacent pairs in every row.

for ($\text{int } j = 0; j < \text{numCols}; j++$)

for ($\text{int } i = 0; i < \text{numRows} - 1; i++$)

Compare $\text{panel}[i][j]$ and $\text{panel}[i + 1][j]$ have the same value or not.

if $\text{panel}[i][j]$ has the same value as of $\text{panel}[i + 1][j]$,

then there are adjacent same value cells, return false,

meaning the game is not over yet.

Otherwise (panel[i][j] is not the same as panel[i+1][j]),

There is no conclusion yet, we still need to wait to check the rest cells,

panel[i][j] does not equal to panel[i+1][j] only means

this specific adjacent pair do not share the same value,

but there might be same value adjacent pairs

somewhere else. Warning: do not return true in the else part.

- (3) If there is no return in either (1) or (2), then it means there is no adjacent same value pairs in any row and any column, return true.

(2.3) Define method selectRandomCell, which do the following

- (a) Record the locations – represented by row and column -- of empty cells (cells with value zero) in the panel in a vector. The locations can be represented by a class called **RowCol** with of row and col (for column) information.

```
class RowCol {  
public:  
    int row;  
    int col;  
};
```

- (b) If there is at least one empty cell, put the cell's location, represented by row and column, into a dynamically allocated array or a vector (will be introduced later) of **RowCol**.

Use a dynamic allocated array of **RowCol** (a type) to record row and column information of each empty cell. You can use vector of **RowCol** as well to avoid dynamic allocated memory processing (new, then delete and handle dangling pointer problem).

- (c) Select a random element from the above array or vector, set the corresponding cell by 1.
- (d) Print the updated panel with the added cell.
- (e) If there is no empty cell, or with the added cell and every cell is filled up (that is, the size of vector was either 0 or 1), if no adjacent cells with same value and max is smaller than target, print "Game over. Try again." with a new line character and exit the program using exit(0); statement.

In general, exit statement terminates the program and is stronger than return statement, while return statement returns to the caller of the function where return statement locates.

Only when in main function, statement exit and return have the same effect. For simplicity, we may think main function return or exit to operating system.

In this task, only submit Board.cpp to gradescope for grading. However, you should keep Board.hpp and Board.cpp in the same directory of local computer for compilation.

After finishing this task, your Board.cpp does not run yet, use `g++ -std=c++11 -c Board.cpp` to check compilation errors.

Pseudocode of `selectRandomCell` using class `RowCol` to record row and column information

```
//define class RowCol to record row and column information of a cell before
//method selectRandomCell
class RowCol
{
    int row;
    int col;
};

void Board::selectRandomCell(int& row, int& col)
begin
    Declare a vector of RowCol called zeros.
    Declare a variable of RowCol called cell.
    for each i in valid row index
    begin
        for each j in valid column index
        begin
            if (panel[i][j] is zero)
            begin
                set cell's row to be i
                set cell's col to be j
                push back cell to vector zeros
            end
        end
    end
end

if the size of vector zeros is larger than 0
begin
    set index to be a random index – an integer in between 0 (included) and size (not
included) -- in this vector
    row = zeros[index].row
    col = ... //your fill the code here
    ... //set the corresponding element in panel with row index row and column index col
to be 1
```

```

        call print method
    end

    if the size of vector zeros was either 0 or 1,
    //this means now all elements in panel is filled
    begin
        if (noAdjacentSameValue())
        begin
            if (max < target)
                print "Game over. Try again.", followed by a new line.
            else print "Congratulations!", followed by a new line.

            exit(0) //end the project
        end
        else return //some adjacent items have same value, the game can continue.
            //Return to caller of selectRandomCell, that is, press arrow key methods.
        end
    end
end

```

*Note that, if we check whether max equals target or not **before** calling selectRandomCell, we do not need to re-check whether max < target or not in method selectRandomCell.