Programming in C – Project 1

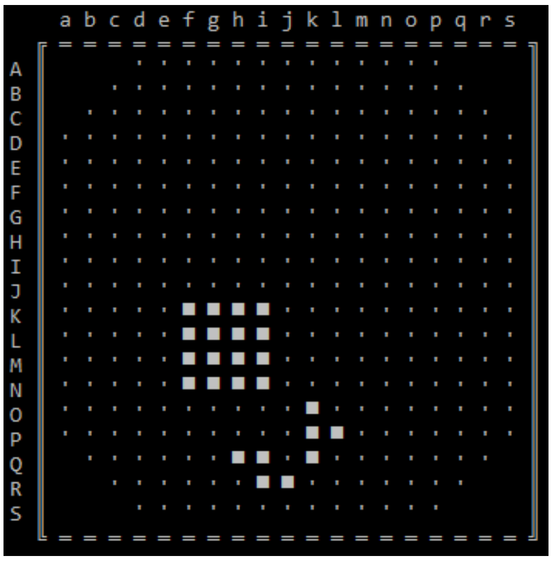
A Tetris-like Game

Description:

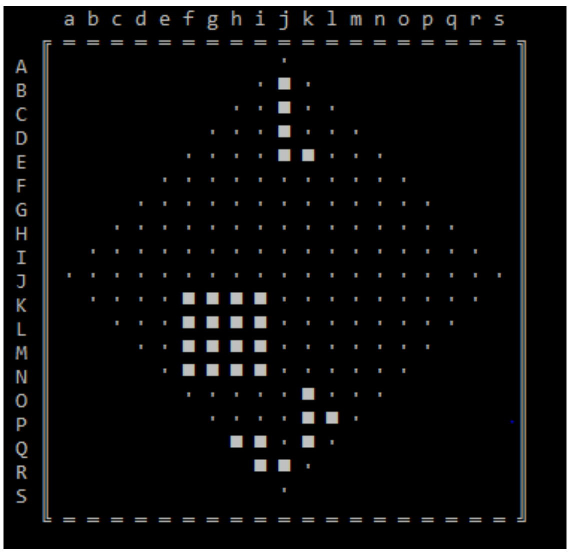
Tetris is a famous game where the board is in the form of a matrix and where blocks of different shapes must be placed so that the board is kept not completely full as long as possible. The idea is to place each block in the location that eliminates as many rows and / or columns as possible. These last are deleted automatically when they are full.

In this project, we want to take inspiration from *Tetris* and create a new version of this game.

The game is performed on a board represented by a 2-dimensional array a **minimum** of 21 rows and 21 columns on which a valid playing surface will be delimited. Lines are indexed by capital letters ('A', 'B', ...) while the columns are indexed by lowercase letters ('a', 'b', …). The valid playing surface can take three different forms: circle, diamond or triangle (see the three figures below).

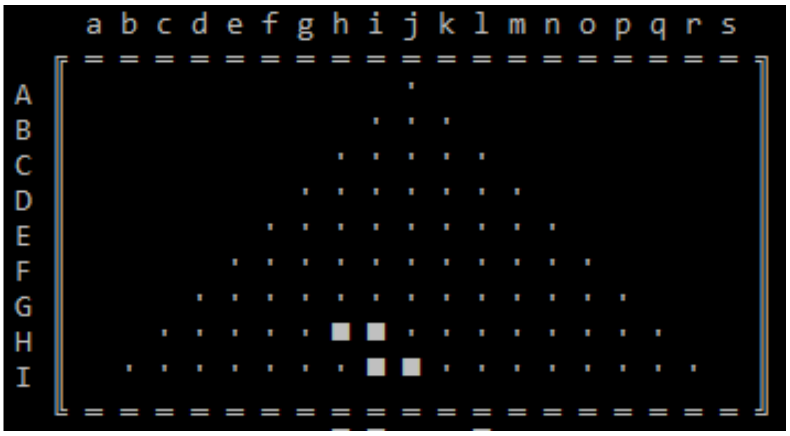


```
   a b c d e f g h i j k l m n o p q r s
  [= = = = = = = = = = = = = = = = = = =]
A        . . . . . . . . . . . . .
B      . . . . . . . . . . . . . . .
C    . . . . . . . . . . . . . . . . .
D  . . . . . . . . . . . . . . . . . . .
E  . . . . . . . . . . . . . . . . . . .
F  . . . . . . . . . . . . . . . . . . .
G  . . . . . . . . . . . . . . . . . . .
H  . . . . . . . . . . . . . . . . . . .
I  . . . . . . . . . . . . . . . . . . .
J  . . . . . . . . . . . . . . . . . . .
K  . . . . . ■ ■ ■ ■ . . . . . . . . . .
L  . . . . . ■ ■ ■ ■ . . . . . . . . . .
M  . . . . . ■ ■ ■ ■ . . . . . . . . . .
N  . . . . . ■ ■ ■ ■ . . . . . . . . . .
O  . . . . . . . . . . ■ . . . . . . . .
P  . . . . . . . . . . ■ ■ . . . . . . .
Q    . . . . . . ■ ■ . ■ . . . . . . .
R      . . . . . . ■ ■ . . . . . . .
S        . . . . . . . . . . . . .
  [= = = = = = = = = = = = = = = = = = =]
```



```
   a b c d e f g h i j k l m n o p q r s
  [= = = = = = = = = = = = = = = = = = =]
A                    .
B                  . ■ .
C                . . ■ . .
D              . . . ■ . . .
E            . . . . ■ ■ . . .
F          . . . . . . . . . . .
G        . . . . . . . . . . . . .
H      . . . . . . . . . . . . . . .
I    . . . . . . . . . . . . . . . . .
J  . . . . . . . . . . . . . . . . . . .
K    . . . . ■ ■ ■ ■ . . . . . . . . .
L      . . . ■ ■ ■ ■ . . . . . . . .
M        . . ■ ■ ■ ■ . . . . . . .
N          . ■ ■ ■ ■ . . . . . .
O            . . . . . ■ . . .
P              . . . . ■ ■ .
Q                ■ ■ . ■ .
R                  ■ ■ .
S                    .
  [= = = = = = = = = = = = = = = = = = =]
```

In this game, the user has a set of blocks which he/she will have to place in turn on the valid surface of the board by entering the coordinates of the place where he/she wants to insert them. Some blocks are common to the three forms, and some are more suitable to be dedicated to only one of them (see section 3).



```
    a b c d e f g h i j k l m n o p q r s
  [ = = = = = = = = = = = = = = = = = = = ]
A                     .
B                   . . .
C                 . . . . .
D               . . . . . . .
E             . . . . . . . . .
F           . . . . . . . . . . .
G         . . . . . . . . . . . . .
H       . . . . . ■ ■ . . . . . . . .
I     . . . . . . . ■ ■ . . . . . . . .
  [ = = = = = = = = = = = = = = = = = = = ]
```

Application to be developped:

When launching the game, a first screen should appear offering two options:

- Start the game

- Display the rules of the game.

The content of this screen not strict, and you can propose your own the design.

When the game starts, this screen should disappear and the user is invited to configure his game:

1. Yhe user must choose the dimension of the game board and its shape among:
   - Circle
   - Diamond
   - Triangle
2. Choose from two block suggestion policies:

   - Display at each game turn all the available blocks and the user selects one.

   - Display only 3 randomly selected blocks.

<u>**Note**</u>: the configuration made at this stage remains unchanged until the end of the game.

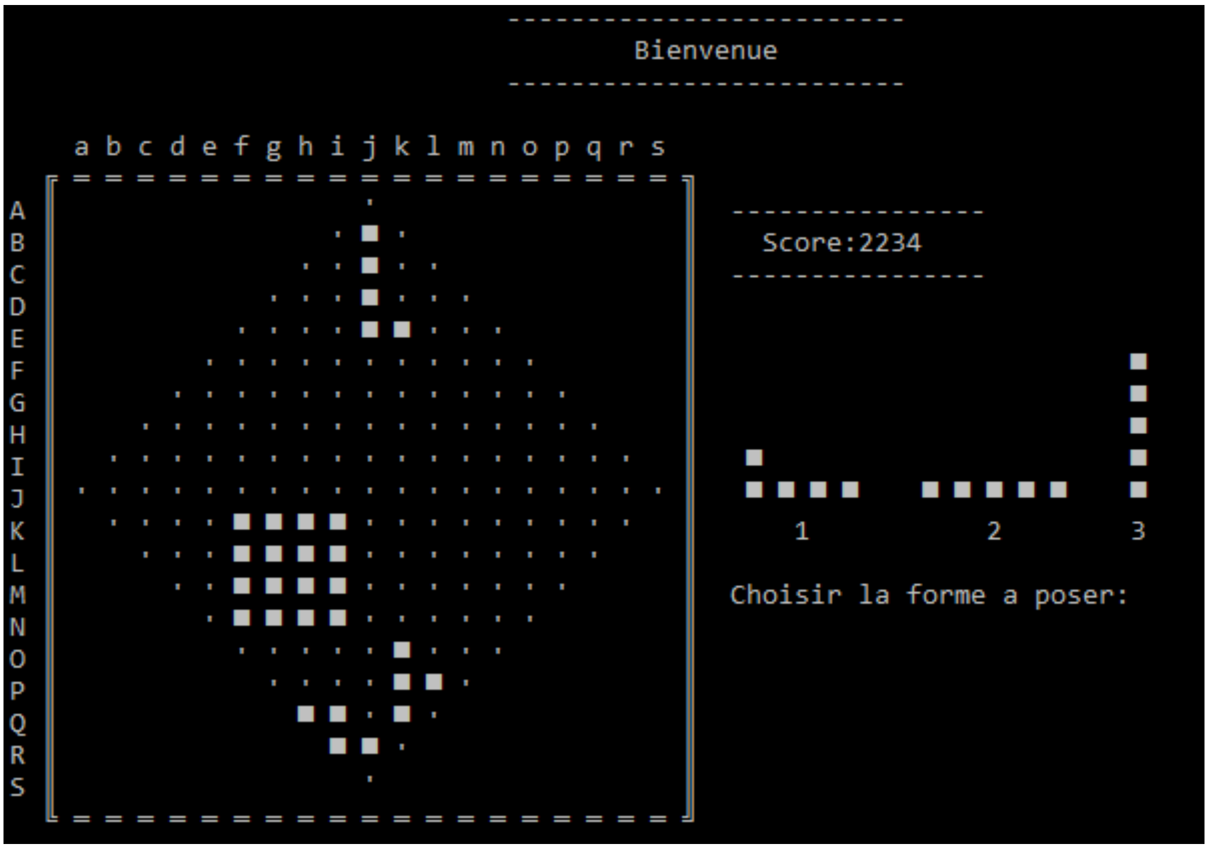
During the game, the program must be able to check if the coordinates entered by the user are valid:

- They are valid if the boxes of the chosen block can all be placed on empty boxes on the valid surface of the game.

- They are not valid if some - or all - of the boxes of the chosen block are outside the valid surface of the game or if the selected location does not have enough boxes to receive the chosen block.

Each time a block is inserted, a test is carried out to check whether there are full rows and / or columns. In this case, these must be eliminated (reset). In the case of line elimination, the full boxes found above must go down as far as possible at the bottom of the playing surface. However, when the column is eliminated, there is no shift to be performed.

In addition, the elimination of the rows / columns must result in the calculation of a score which is displayed next to the game board. A simple score function would consist to count the number of eliminated boxes. However, the proposal of any other function is accepted.



```
                         -------------------------
                                 Bienvenue
                         -------------------------

    a b c d e f g h i j k l m n o p q r s
  ╔ = = = = = = = = = = = = = = = = = = = ╗
A ║                   .                   ║
B ║                 . ■ .                 ║
C ║               . . ■ . .               ║
D ║             . . . ■ . . .             ║
E ║           . . . . ■ ■ . . .           ║
F ║         . . . . . . . . . . .         ║
G ║       . . . . . . . . . . . . .       ║
H ║     . . . . . . . . . . . . . . .     ║
I ║   . . . . . . . . . . . . . . . . .   ║
J ║ . . . . . . . . . . . . . . . . . . . ║
K ║   . . . . ■ ■ ■ ■ . . . . . . . . .   ║
L ║     . . . ■ ■ ■ ■ . . . . . . . .     ║
M ║       . . ■ ■ ■ ■ . . . . . . .       ║
N ║         . ■ ■ ■ ■ . . . . . .         ║
O ║           . . . . . ■ . . .           ║
P ║             . . . . ■ ■ .             ║
Q ║               ■ ■ . ■ .               ║
R ║                 ■ ■ .                 ║
S ║                   .                   ║
  ╚ = = = = = = = = = = = = = = = = = = = ╝

     ----------------
       Score:2234
     ----------------

                                 ■
                                 ■
                                 ■
     ■                           ■
     ■ ■ ■ ■       ■ ■ ■ ■ ■     ■
        1              2         3

     Choisir la forme a poser:
```

Stop condition:

To insert a block on the board, the user has three attempts to enter valid coordinates. After 3 unsuccessful successive attempt, the game stops.

At the end of the game, a message should appear on the screen displaying the obtained score.

Technical Aspects:

The realization of this project requires thinking carefully about the appropriate date structure for the storage of all the game elements. You should also think about a clever segmentation of your code into functions.

1. Date structures

- First, each block will be represented by a 2D matrix, which is created **dynamically, of dimension** NxM NxM (N = greatest height among all the blocks offered in the series, M = greatest width among all the blocks offered in the).

- Then, all the blocks to be used in the game are stored in a single array of size **nb_blocks**.

Example : Consider the following blocks:

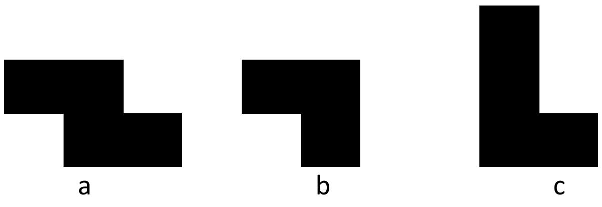
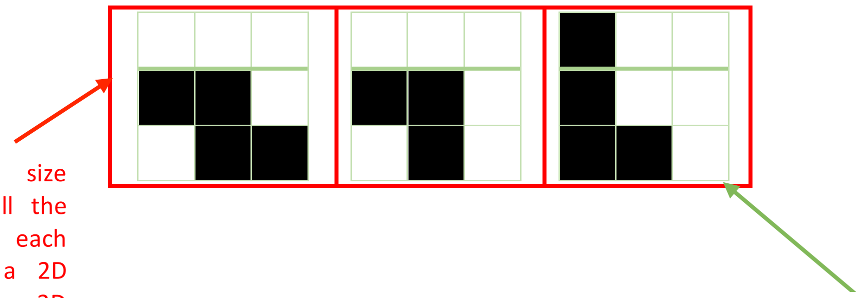


Figure 1

The size of the matrices where they will be stored will be 3x3 because the block in Figure 1.c has the largest height (= 3) and the block in Figure 1.a has the largest width (= 3).

All the blocks are stored in a single array as shown in the figure below

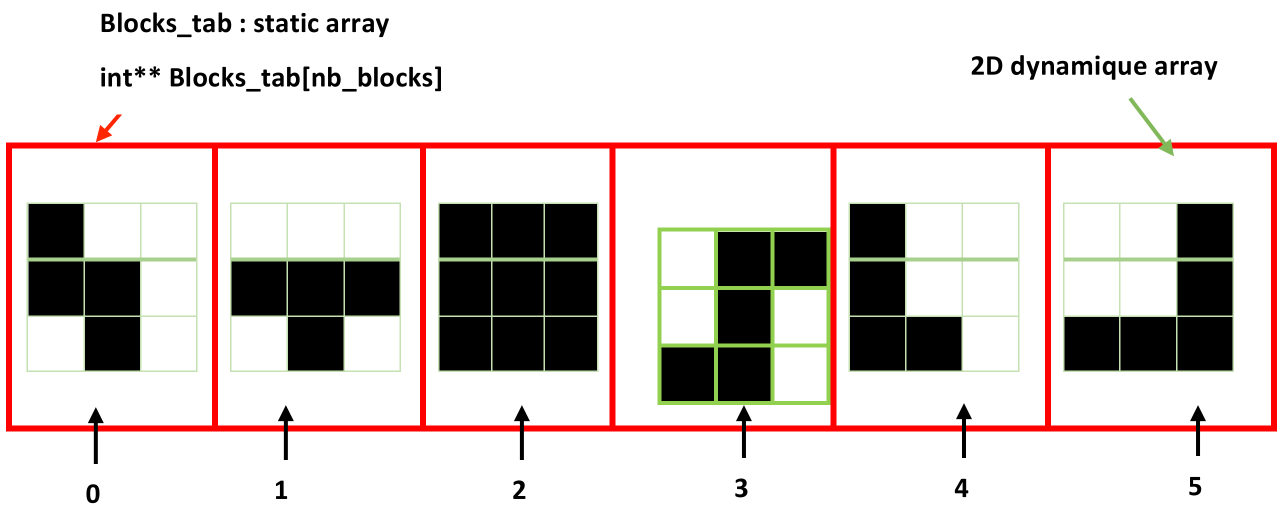


Blocks: Array of size *nb_blocs* storing all the blocks. Hence, each location contains a 2D array ➔ the whole is a 3D array.

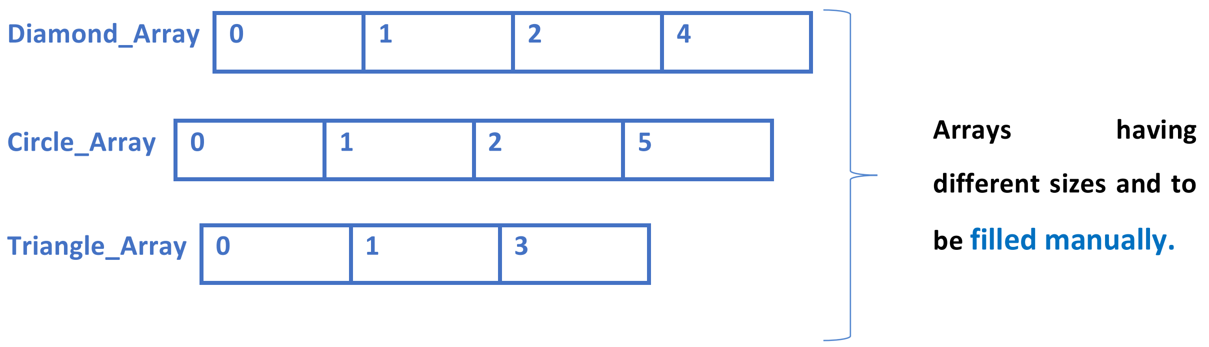
Block: A 2D array of size NxM. *Note: use 0 to represent empty boxes, and 1 for the filled ones.*

- Each shape of the board game is associated with a one dimension array containing the positions of the corresponding blocks in the global array (see section 3). Thus, in this game, there will therefore be three arrays of different sizes: **Diamond_Array**, **Circle_Array** and **Triangle_Array**.

<u>**Example**</u>: Assume that the array **Blocks_tab** contains the following blocks:



In this example, the three arrays associated to the three board shapes could be the following:



Thus, to access the block at position 5 for a board having the form of a circle, we use **Blocks_tab[Circle_Array[3]]** .

2. Functions

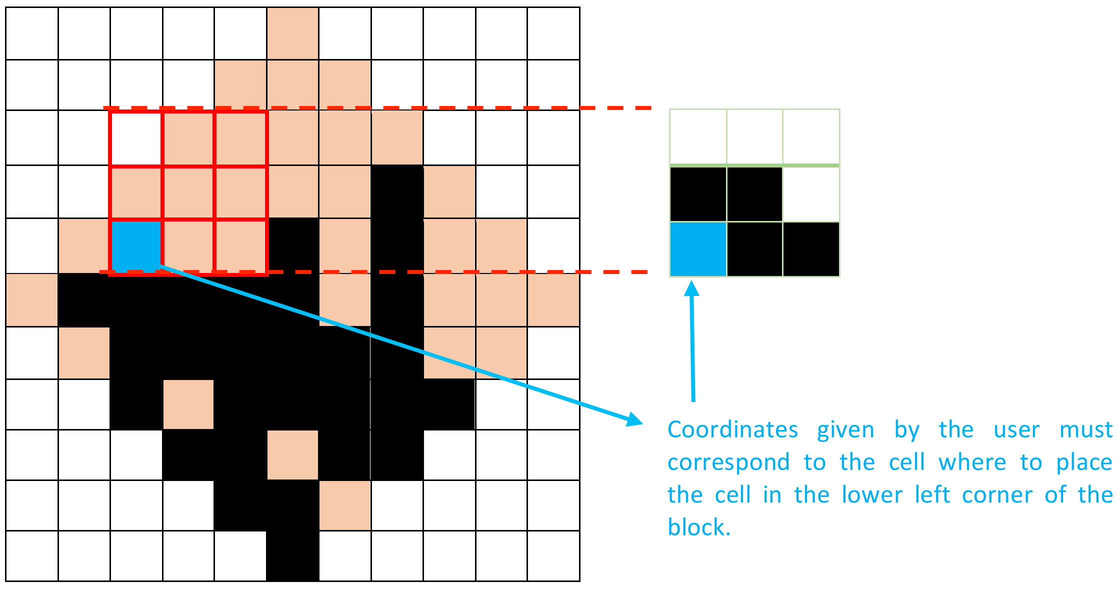
- Write a function **create_2D_dyn** to create a dynamic 2D array. Note that the game board and the different blocks must be created dynamically.
- For each shape **"sh"**, write a function **create_block_sh** that takes the dimensions of a given block as parameters, creates a 2D array dynamically and fills it with '1's and '0's depending on the desired shape. This function can be used for the creation of the game board, but also for the creation of the different blocks.

- Write a function fill_location_tab which takes in parameters a 2D array and an index i and which allows, according to the index i, to call one of the previous functions **create_block_sh** to create the corresponding block (you could use **switch case** statement on the index i to allow the creation of any kind of block).
- Write a function **display_board** which takes in parameters a 2D array and its dimension, and which displays it on the screen. The display will be made using the ASCII codes of the various symbols and characters. The choice of the latter yours, the only condition is that the rendering on screen is clear enough.
- Write a function **display_blocks** that displays the list of the blocks associated with a given form of the game board.
- Write a function **select_blocks** that selects the blocks to be suggested to the user according to one of the two strategies mentioned above.
- Write a function **check_validity** that takes as parameters a block and coordinates on the board, and that checks if it is possible to place the block at these coordinates. If yes, the function returns 1, else it returns 0.

  Rule for placing a block on the board:

  - We assume that the insertion of a block on the board will be done by the superposition of the corresponding matrix at the place where we want to insert it. To do this, the cell in the lower left corner of the block must coincide with the coordinates entered by the user.

    <u>**Example**</u>:

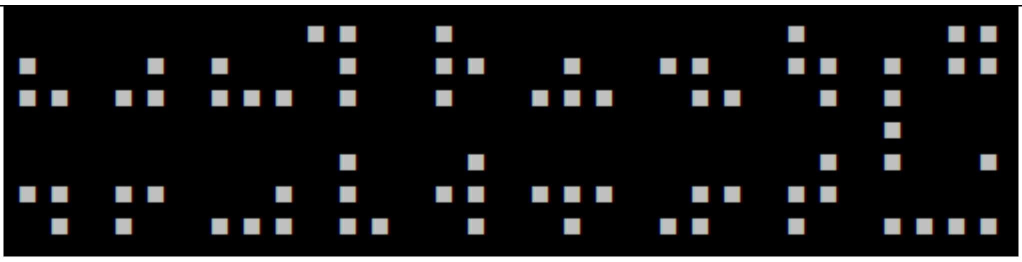
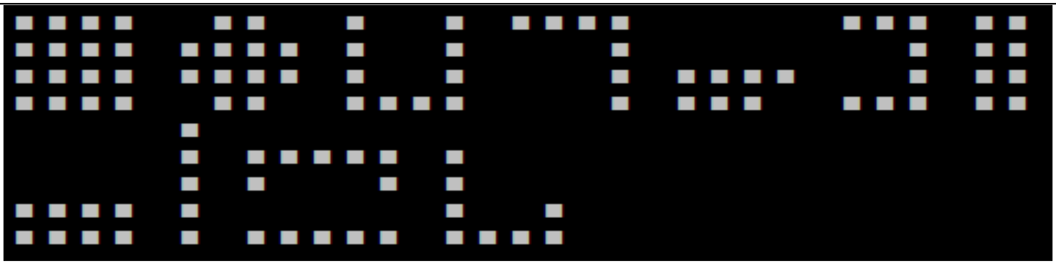
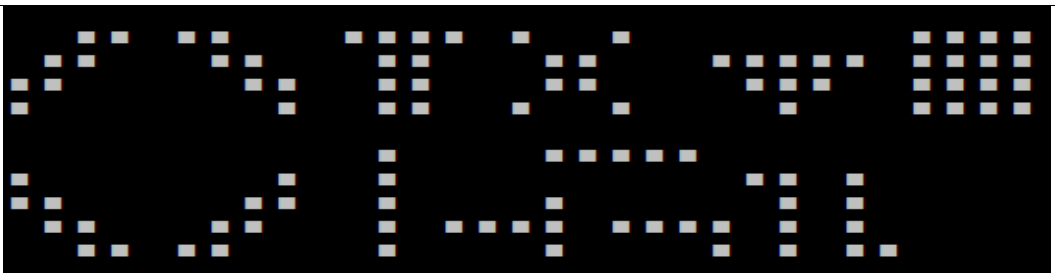
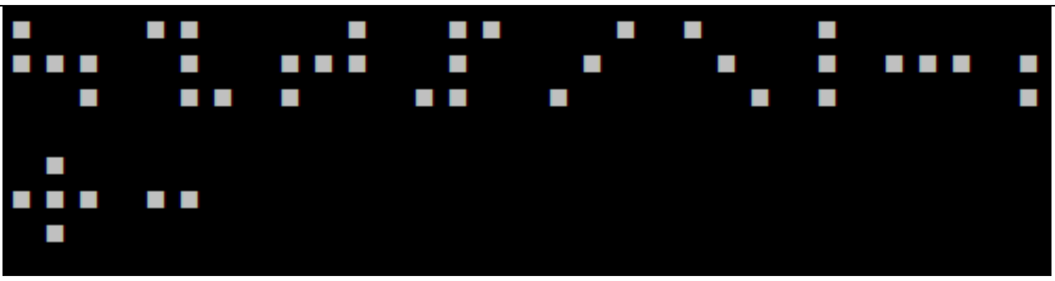

Coordinates given by the user must correspond to the cell where to place the cell in the lower left corner of the block.

- Write a function **place_block** that place a block on the board at position (i, j). The cells occupied by the block can take the value 2 so that they are distinguished from the other cells of the board (0 and 1).
- Write a function **state_line** that returns 1 when a given line is full, and returns 0 otherwise.
- Write a function **state_column** that returns 1 when a given column is full, and returns 0 otherwise.
- Write a function **eliminate_line** that eliminate a given line from the board (reset it to become empty).
- Write a function **eliminate_column** that eliminate a given column from the board (reset it to become empty).
- Write a function **shift_lines**, which, starting from a given line, shifts downwards all the possible cells. The cells, whose shifting put them outside the valid playing area, must not change position.
- Write a function **compute_score** that updates the score each time a line or column has been eliminated.
- Write the main program simulating the game using the previous functions (and others if necessary).

IMPORTANT!!

1. Your source code must be separated into three files: **project1c.h**, **project1c.c** and **main.c**
2. In all your functions, you must secure the input regarding the type of the expected values and the their validity regarding the valid playing area.

3. Blocks associated with each form of the game board

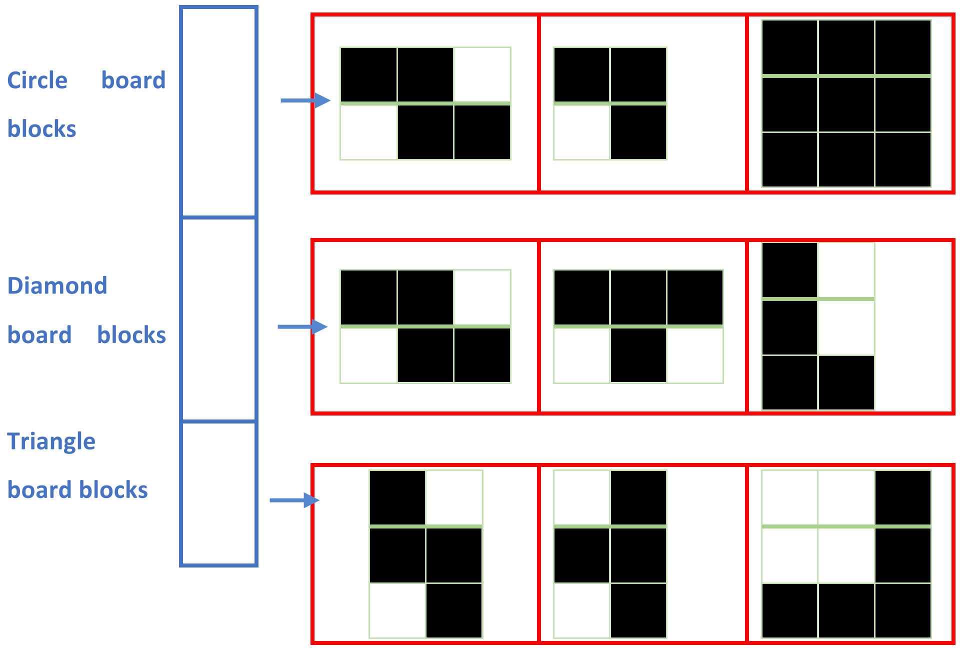
Board	List of blocks
All (Blocks shared by all board forms)	
Circle	
Diamond	
Triangle	

Optional

Below are some additional features that the most motivated can integrate to have a more advanced version of the game:

- Depending on the value of the score, add difficulty to the game by suggesting blocks that are not always easy to insert.
- In order to generalize the game to different boards and different shapes of blocks, implement a date structure where all the arrays are created dynamically. Also, the sizes of the arrays associated with the different blocks have not necessary the same size.

-



- Add a third policy to suggest blocks that allows to display at each turn of the game only 3 blocks after checking that they can be placed on the board.
- Change the condition of validity of the coordinates entered by the user such that: the entered coordinates are only accepted if there is a path that the block can take from the top of the board to the desired position. To place this block, simulate its movement through the previously calculated path.

General instructions

1) The project is to be done in pairs. A single group of three students could be authorized if the number of students in the group is odd.

2) You will have to upload on the moodle the following:

- A .zip file containing all the developed C files (.h and .c files)

- A .pdf file (10 to 15 pages) describing your solution and data structure choice (use algorithm language to describe the main functionalities).

3) The uploaded file must be named as follows: NAME1_NAME2 (where NAME1 and NAM2 are the names of the two students of the group).

4) The deadline to upload your work is April 23 at 23:59

Planning

- Sunday March 23 : Subject is available on the moodle

- Week of March 30: Lab session dedicated to the project
- Week of April 20: Project defence

Evaluation

The final mark will be composed as follows:

1) Functionalities and source code (/12)

2) The report (/4)

3) The defence (/4)