AQUILANO Arthur
PEREIRA Hugo

LP2A Project : Tetris Game

Within the framework of the LP2A course, we had the objective to code a Tetris game clone. This project was a great manner to use in a concrete situation all the knowledge we get during the semester.

Quick reminder of the subject: we had to recreate the Tetris game. The purpose of the game is to complete lines in an empty grid by putting in the best place and position blocks, named tetrominoes, which have different shapes, and which fall constantly. Of course, the game stops and the player loose when the grid is filled.

Summary:

I) Working Method

For this part of the report, we will see how we think about the subject, how we split the task, how we organized our work and how we face the encountered issues.

So firstly, we were two on the project, Hugo and Arthur. None of us ever touched java language before the begin of the semester however, we both know the concept of Tetris so we had some idea of how to code the main part of the game. The first thing we did was to take time to read together the project proposal, thus, we could orient our search and work more precisely. So, during the following week, Hugo looked on the internet some documents to find hint to create a good Tetris game. He used his discovery to have an idea of the technical stuffs we would have to implement. On the other hand, Arthur was trying to design a UML diagram. Despite knowing that it is always better to realize the UML diagram before starting coding, the first attempt was not the excepted result, so, we decided to keep it as a solid base and to improve it after we code the game.

II) Coding Classes

The second part of the game development is basically the coding of every class. As it has been said before, our UML diagram was not precise enough to know which functions each class would contain, however, we knew the number of classes we should write and the main interaction of each other. Hugo wrote the classes linked to the functional code of the game because he took time to learn and understand the main aspects of the Tetris. Arthur was concentrated on a way to code the GUI of the game. In fact, we never coded any graphical stuffs with other language (like C for instance) so it took him time to understand how to correctly use all the swing libraries. Thus, we split the work that way, and we often sent our advancement to the other and took time to explain what we did so no one was lost. We think our strategy paid because we succeed to produce a working project before deadlines.

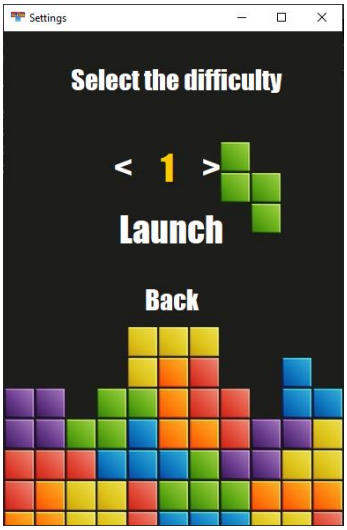
III) Code's explanations

For the third part of the report, we will enter a bit more in the code by explaining some part we think interesting to explain. (for instance, it is useless to speak 7th time about the tetrominoes' rotation). Our explanations will follow the progress of the game (so the menu first, then the settings windows, etc.).

So, the first window which shows up when we run the game is the Menu one. Two things are interesting to talk about: the way we arrange everything and the use of label instead of button.

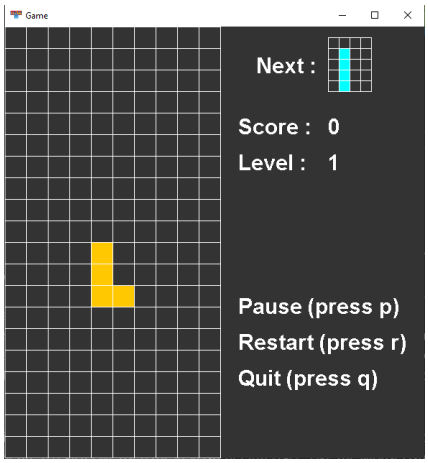
Even if it is a basic step of GUI, it's still interesting to speak about once. So, to create the window, we used a JFrame object with all default configurations (size, closing method, and adding all the panel). Then we used a JLabel placed in a JPanel to add a menu's wallpaper to our frame. The main difficulty for this step was to find an image which fit correctly in the window and the correct way to call it. The remains parts are all the text Label which are used as button. So, we basically create and configure then with a font, a colour, a size, etc. and added them in the panel. To use them as button we had to create an inner class MouseHandler class which implements a MouseListener. It allows us to get many precoded methods to detect mouse action or position. Thus, the mouse system detection checks the position of the mouse and which button is clicked and if user clicks on a label, it calls some specifics methods.

```
Menu(){

    pBackground = new JPanel();
    pBackground.setBorder(BorderFactory.createEmptyBorder(0, 0, 0, 0));
    pBackground.setLayout(null);
    pBackground.setOpaque(false);

    titre = new JLabel("TETRIS",SwingConstants.CENTER);
    titre.setBounds(95, 30, 200, 50);
    titre.setFont(new Font("Impact", Font.BOLD, 60));
    titre.setForeground(Color.white);
    pBackground.add(titre);

    start = new JLabel("Start",SwingConstants.CENTER);
    start.setBounds(90, 150, 200, 50);
    start.setFont(new Font("Impact", Font.PLAIN, 40));
    start.setForeground(Color.white);
    pBackground.add(start);

    quit = new JLabel("Quit",SwingConstants.CENTER);
    quit.setBounds(90, 250, 200, 50);
    quit.setFont(new Font("Impact", Font.PLAIN, 40));
    quit.setForeground(Color.white);
    pBackground.add(quit);

    lBackground = new JLabel();
    lBackground.setIcon(new ImageIcon("src/Images/bg.jpg"));
    lBackground.setBounds(0, 0, 400, 600);
    pBackground.add(lBackground);


    frame = new JFrame();
    frame.setTitle("Menu");
    frame.setSize(400,600);
    frame.setIconImage(Toolkit.getDefaultToolkit().getImage("src/Images/logo.png"));
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setLocationRelativeTo(null);
    frame.setVisible(true);
    frame.add(pBackground);
    frame.addMouseListener(mHandler);
}
```



The settings window uses the same principle as the Menu one.

Then we get into the Game frame. This time we did not use swing libraries but awt one (we know it's absolutely no advocate to mix them, but our lack of knowledge constrains us to opt to this option). Basically, the GUI is a bunch of Graphical objects. We set some text on the right who displays main information (like a score, a pause menu). Foremost, we initialize everything, then, by using the function repaint, we refresh the page. Regarding the grid, it has been created by adding white lines and in vertical traits. The specific aspect of the grid is that it is linked to an array (we will talk about it later). The last thing on this window is the previsualization of the next piece at the top right corner.




```
//draw the grid
g.setColor(Color.white);
for(int row = 0; row < GRID_HEIGHT + 1; row++) {
    g.drawLine(0, BLOCK_SIZE * row, BLOCK_SIZE * GRID_WIDTH, BLOCK_SIZE * row);
}
for(int col = 0; col < GRID_WIDTH + 1; col++) {
    g.drawLine(col * BLOCK_SIZE, 0, col * BLOCK_SIZE, BLOCK_SIZE * GRID_HEIGHT);
}

//draw the score
g.setColor(Color.white);
g.setFont(fonte);
g.drawString("Score :", 325, 150);
g.drawString(String.valueOf(score), 450, 150);

//draw the level
g.setColor(Color.white);
g.setFont(fonte);
g.drawString("Level :", 325, 200);
g.drawString(String.valueOf(level), 450, 200);

/draw the pause
g.setColor(Color.white);
g.setFont(fonte);
g.drawString("Pause (press p)", 325, 400);

//draw the restart
g.setColor(Color.white);
g.setFont(fonte);
g.drawString("Restart (press r)", 325, 450);

//draw the quit
g.setColor(Color.white);
g.setFont(fonte);
g.drawString("Quit (press q)", 325, 500);

nextBlock.render(g, 15, 31 , 2);

for(int row = 0; row < nextBackground.length; row++) {
    for(int col = 0; col < nextBackground[row].length; col++) {
        if(nextBackground[row][col] != null) {
            g.setColor(nextBackground[row][col]);
            g.fillRect(col * 15, row * 15, 15, 15);
        }
    }
```

Just before explaining all the code associated with the game window, we are going to quickly speak about the pause menu. To call it I we used the KeyListener interface. Like the MouseListener saw upper, the Keylistener permit us to call methods after some specific actions with the keyboard. It is codded with the same way as the game. Otherwise, this pause menu does not just pause the game, but it let the user the possibility to restart the game or to quit it.

```
public void keyPressed(KeyEvent e) {
    // TODO Auto-generated method stub
    if(e.getKeyCode() == KeyEvent.VK_DOWN) {
        isBlockFalling();
        score++ ;
    } else if(e.getKeyCode() == KeyEvent.VK_RIGHT) {
        movementRight();
    } else if(e.getKeyCode() == KeyEvent.VK_LEFT) {
        movementLeft();
    } else if(e.getKeyCode() == KeyEvent.VK_UP) {
        rotationBlock();
    } else if(e.getKeyCode() == KeyEvent.VK_P) {
        if(pause == true) {
            pause = false;
        } else if (pause == false) {
            pause = true;
        }
        repaint();
    } else if(e.getKeyCode() == KeyEvent.VK_R) {
        score = 0;
        resetBackground();
        selectRandomBlock();
    } else if(e.getKeyCode() == KeyEvent.VK_Q) {
        System.exit(0);
    }
}
```

Now we can see in detail how the game works step by step. Firstly, we call the constructor GameArray(), which gets the level selected by the user, then it initializes a Random() and the loop. This attribute is like a for, in contrast of there is not any condition to end it. The next step is a first condition which check if the game is in pause or not. The next one verifies if the game is over or not, then the final one in this loop check if the block is falling or not. For the last, if the return is true, the piece goes down of one row, if it's false, we set the piece in the array with the blockIntoBackground() method, then we check if a line is complete, in case of a yes, the program removes it, a new tetrominoes is randomly picked and we repaint the whole grid.

```
GameArray(int level){

        this.level =level;

        rand = new Random();

        selectRandomBlock();

        loop = new Timer(1000/level, new ActionListener() {
            @Override
            public void actionPerformed(ActionEvent ae) {

                if(pause == false) {
                    if(isGameOver() == false) {
                        if(isBlockFalling()) {
                            repaint();
                        } else {
                            blockIntoBackground();
                            removeLine();
                            selectRandomBlock();
                            repaint();
                        }
                    }
                }
            }
        });
        loop.start();

}
```

We will see more precisely the running of all the methods linked with the comportment of a piece during the demonstration and the PowerPoint going with (it is more enjoyable to have direct representation of our talk).

However, it might currently be interesting to see deeper the manner we code the removeLine() and the blockIntoBackground() methods. The first one (as we can see on the screen) use two attributes, a boolean and an int. The boolean is obviously the condition for checking if a line is full or not, and the int counts how many lines are complete at the same moment so we can attribute more points. Thus, we enter in a double loop who check each case of every row until an empty case is found. If a line is complete, we change the value of the boolean to true to enter in the removing line part of the method. Really simple to understand, it sets all cases in colour null, then it levels down all the upper lines, just for the highest line we need to set apart the colour value to null. Next, for the second function, we put the tetrominoes into the background by simply changing the colour of the grid.

```
public void removeLine() {

    boolean comp;
    int count = 0;

    for(int r = GRID_HEIGHT - 1; r>=0; r--) {
        comp = true;
        for(int c = 0; c < GRID_WIDTH; c++) {
            if (background[r][c] == null) {
                comp = false;
                break;
            }
        }

        if(comp == true) {
            //clear the line complete
            for(int i = 0; i < GRID_WIDTH; i++) {
                background[r][i] = null;
            }

            //shiftDown all the line
            for(int r1 = r; r1 > 0; r1--) {
                for(int c1 = 0; c1 < GRID_WIDTH; c1++) {
                    background[r1][c1] = background[r1 - 1][c1];
                }
            }

            //clear the first line
            for(int i = 0; i < GRID_WIDTH; i++) {
                background[0][i] = null;
            }

            r++;
            count++;
        }
    }

    if(count <= 3) {
        score = score + count*100;
    } else if (count >= 4) {
        score = score + 800;
    }

    repaint();

}
```

```
public void blockIntoBackground() {
    for (int r = 0; r < block.getHeight(); r++) {
        for (int c = 0; c < block.getWidth(); c++) {
            if(block.pieceShape[r][c] == 1) {
                background[r + block.getY()][c + block.getX()] = block.getPieceColor();
            }
        }
    }
}
```

IV) Issues

The fourth part will deal with the issues we encountered during this project. There were a lot of minimal problems that we solved quite easily (with the help of documentation on internet). Nevertheless, three mains' parts took us time to solve. One is all the interface system. As we said earlier, we were not used with all the graphics packages, so we needed time to globally understand how to code GUI. Thus, this part was the last one of our work. Therefore, some methods were missing during the first runs, like the blockIntoBackground() one. As we see just higher the method is not tricky to write, but it is essential, and it takes some not predicted time.

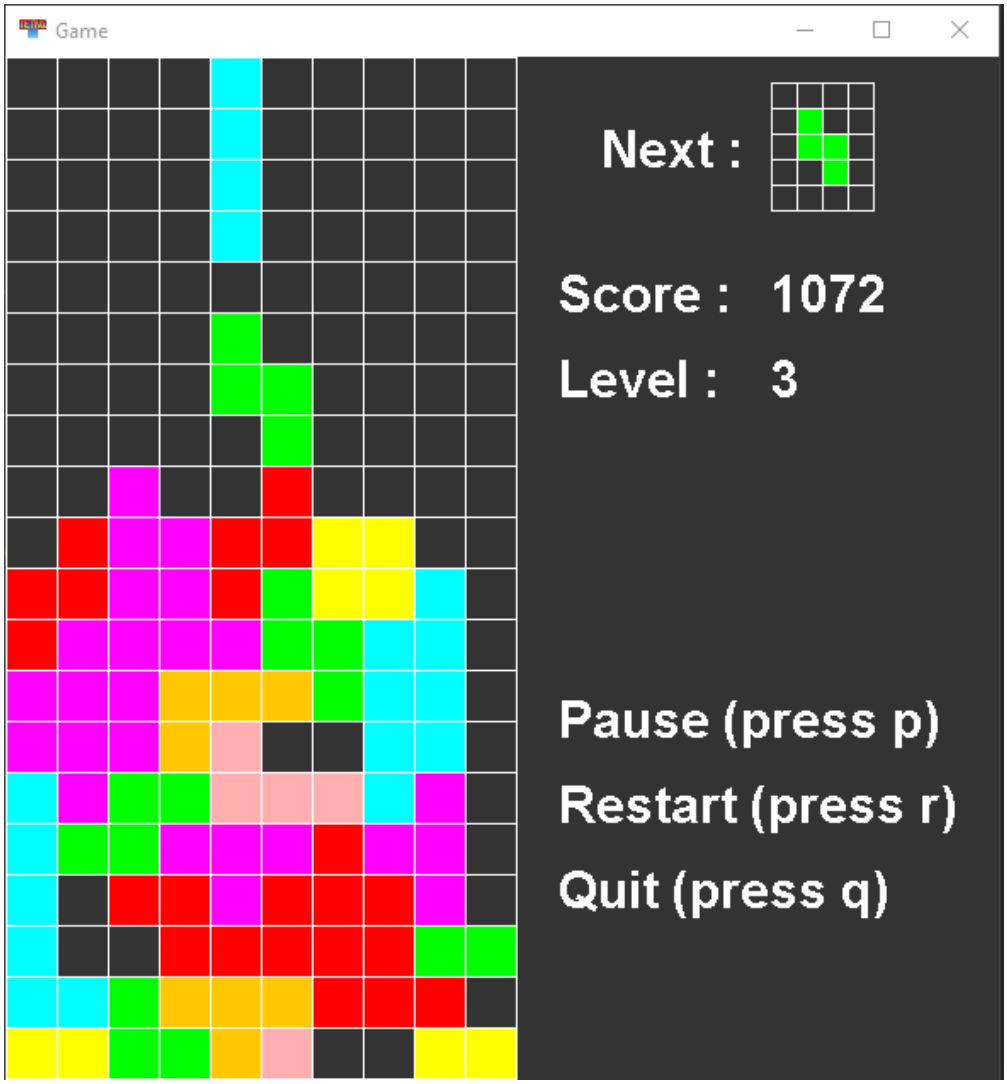
Furthermore, a point which blocked us was the rotation of the pieces. Same as before, it is not a difficult method in theory, but it requires a lot of unforeseen things around. For instance, a block must not go out of the grid, but in some cases, rotation leads to this issue. We had to code methods who check if rotation is available near grid's borders or other block, and which comportment the piece must adopt consequently.

The final main problem was linked to the spawning array of pieces. By putting its declaration in the top of the class, we encountered the situation of spawning each piece only once then nothing more happened. This problem also creates glitches with the removing line method. After reading all the code for hours, we simply changed the place of the array declaration in the selectRandomBLock() method, so all problems were solved.

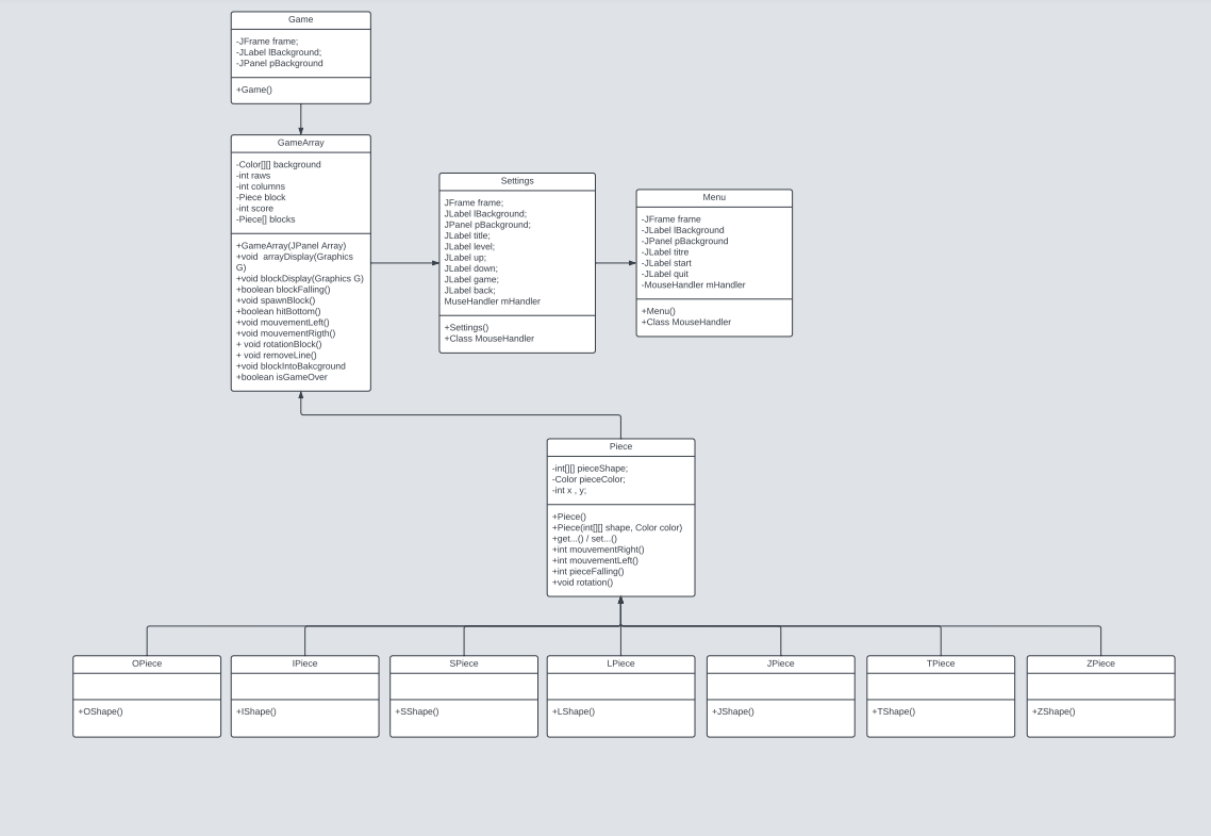
V) Conclusion

To conclude, we achieve to create a functional Tetris game in the allotted time. This project brings us a lot of experience with java language and OOP in general. We are now more familiar with classes, interfaces, and GUI.

With a personal touch, we found the project funny and playful, not too hard to globally understand but enough difficult to be exciting.



Annex:


Game
-JFrame frame;
-JLabel lBackground;
-JPanel pBackground
+Game()
GameArray
-Color[][] background
-int raws
-int columns
-Piece block
-int score
-Piece[] blocks
+GameArray(JPanel Array)
+void arrayDisplay(Graphics G)
+void blockDisplay(Graphics G)
+boolean blockFalling()
+void spawnBlock()
+boolean hitBottom()
+void mouvementLeft()
+void mouvementRigth()
+ void rotationBlock()
+ void removeLine()
+void blockIntoBakcground
+boolean isGameOver
Settings
JFrame frame;
JLabel lBackground;
JPanel pBackground;
JLabel title;
JLabel level;
JLabel up;
JLabel down;
JLabel game;
JLabel back;
MuseHandler mHandler
+Settings()
+Class MouseHandler
Menu
-JFrame frame
-JLabel lBackground
-JPanel pBackground
-JLabel titre
-JLabel start
-JLabel quit
-MouseHandler mHandler
+Menu()
+Class MouseHandler
Piece
-int[][] pieceShape;
-Color pieceColor;
-int x , y;
+Piece()
+Piece(int[][] shape, Color color)
+get...() / set...()
+int mouvementRight()
+int mouvementLeft()
+int pieceFalling()
+void rotation()
OPiece
+OShape()
IPiece
+IShape()
SPiece
+SShape()
LPiece
+LShape()
JPiece
+JShape()
TPiece
+TShape()
ZPiece
+ZShape()