Object Oriented Programming

Project: Tetris game

Lecturer: Tran Thanh Tung

and Pham Quoc Son Lam

TEAM INFORMATION

Hồ Thị Thu Hòa ITITIU19120

Hồ Tú Quyên ITITIU19196

Trần Ngọc Diễm Quyên ITITIU19197

Trần Anh Thi ITITIU19212

Table of contents

TABLE : Proposal of my team	
Week 1& 2	Choose a topic, discuss and set proposal
Week 3& 4	Write the game's introduction and rules
Week 5& 6	Create classes containing attributes for the project
Week 7& 8	Complete the code and run the demo
Week 9& 10	Finalize the report and powerpoint

- **The contribution of each person in the group:**

Hồ Thị Thu Hòa ITITIU19120: 25%

Hồ Tú Quyên ITITIU19196: 25%

Trần Ngọc Diễm Quyên ITITIU19197: 25%

Trần Anh Thi ITITIU19212: 25%

A. INTRODUCTION AND RULES OF TETRIS

Tetris is a tile-matching video game first designed and developed by Soviet computer scientist Alexey Pajitnov. The game was created on June 6th, 1984 while he was working at Dorodnicyn's Calculation center.

In Tetris, there are modes such as single player, multiplayer. At the start, player move pieces of diferent shapes that are dropped one after another on the playing field. The completed horizontal rows will disappear and the player will receive points. Players continue to fill in the gaps and the score will gradually increase. With the mode multiplayer, players must outlast their opponents to win.

Rules of Tetris

- Player can only move the blocks in the following directions left, right, and down.

- Each block is made up of 4 squares and has different forms and colors.

- There are 7 different forms: J, L, O, S, T, Z, I. The container has a fixed width and height to contain blocks.

- When a block moves down and collides with other blocks which is freeze or the bottom edge of the container, that block will be freeze and a new block will be created with random form at the top of the container.

- The player will get 1 points when 1 block is freeze.

- Newly created blocks will have a faster fall rate.

- The player can rotate the current block to change the current form of the block, if the player rotates 3 times, the fourth rotation will restore the original form.

- When each square of any blocks fills up any rows of the container, squares in these rows will be remove and the rest of the squares which is above of the removed rows will fall and stand on other blocks.

- The player will get 3 points when each row destroyed. If the player destroys many rows at the same time. The total score will be multiplied by the number of rows destroyed.

- The player can also pause the game when it is running and replay the game if they want.

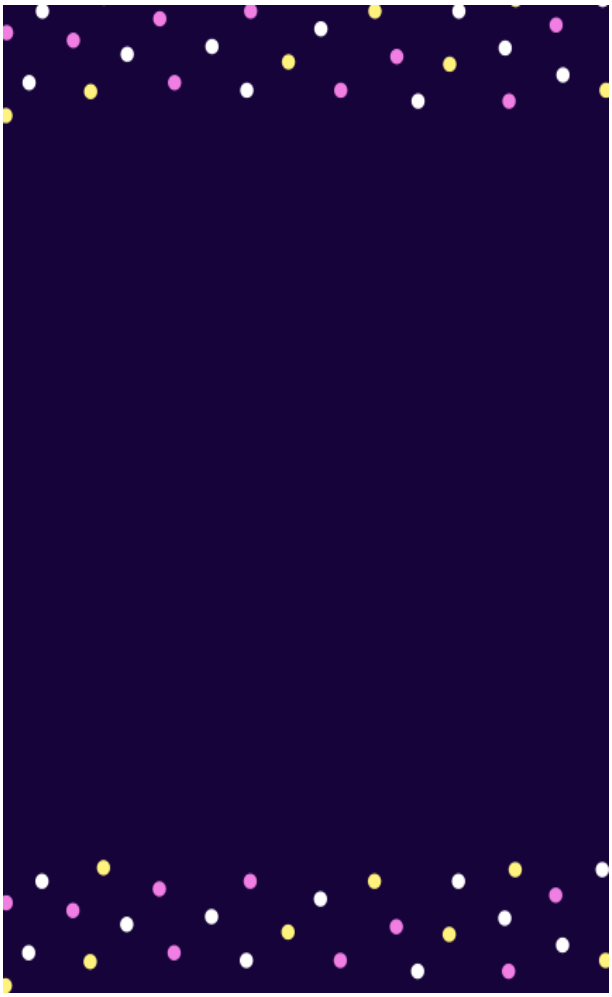
- If any squares of blocks collide the top edge of the container, the game will be over.

B. CLASSES

1. Window class

- Graphical interface pack, display game

- Background of game screens



```java
public class Window {
    public static final int WIDTH = 445, HEIGHT = 629;
    private Board board;
    private Title title;
    private JFrame window;

    public Window() {
        window = new JFrame("Tetris");
        window.setSize(WIDTH, HEIGHT);
        window.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        window.setResizable(false);
        window.setLocationRelativeTo(null);

        board = new Board();
        title = new Title(this);
        window.addKeyListener(board);
        window.addMouseMotionListener(title);
        window.addMouseListener(title);
        window.add(title);
        window.setVisible(true);
    }
    public void startTetris(){
        window.remove(title);
        window.addMouseMotionListener(board);
        window.addMouseListener(board);
        window.add(board);
        board.startGame();
        window.revalidate();
    }
    public static void main(String[] args) {

        new Window();
    }
}
```

2. Title class

- Initialize the game and read image file and edit them.

- Identify items and design them on title screen.

```java
public class Title extends JPanel implements MouseListener, MouseMotionListener {

    private static final long serialVersionUID = 1L;
    private int mouseX, mouseY;
    private Rectangle bounds;
    private boolean leftClick = false;
    private BufferedImage title, play;
    private Window window;
    private BufferedImage[] playButton = new BufferedImage[2];
    private Timer timer;

    public Title(Window window){
    public void paintComponent(Graphics g){

    public void mouseClicked(MouseEvent e) {

    public void mousePressed(MouseEvent e) {

    public void mouseReleased(MouseEvent e) {
    public void mouseEntered(MouseEvent e) {

    public void mouseExited(MouseEvent e) {

    public void mouseDragged(MouseEvent e) {

    public void mouseMoved(MouseEvent e) {
}
```

3. Shape class

- Package of processing algorithms, handling use cases.

- The shapes and colors of the figure.

_In the constructor, we will create a shape with type, image, its container, and color.

We set the initial position at (4, 0) and the speed of it.

_In the **update** method, we will set moveX equals true so that we can move the shape horizontally.

Besides, if the shape collides, we will put the number of color into board and check for destroying row or not, and we will get 1 score.

After that, the new shape will be created.

_It is necessary to prevent the player from moving the shape out of the board horizontally and vertically.

_In the **render** method, we will draw the shape when it moves.

_In the **check line** method, we will create the size equals to the height of the board - 1. We store the score before and after to determine the number of lines destroy at the same time. In the for loop, if count equals to the width of the board, that means there are 1 row that needed to be destroyed, and other above shapes will fall. If the player destroys many rows at the same time, the player will get more scores.

```
private void checkLine(){
    int size = board.getBoard().length - 1;
    int scorebef = board.getScore();

    for(int i = board.getBoard().length - 1; i > 0; i--)
    {
        int count = 0;
        for(int j = 0; j < board.getBoard()[0].length; j++)
        {
            if(board.getBoard()[i][j] != 0) count++;

            if(count == 10) board.addScore(3);;

            board.getBoard()[size][j] = board.getBoard()[i][j];
        }
        if(count < board.getBoard()[0].length)
            size --;
    }
    int scoreaf = board.getScore();
    int linesDestroy = (scoreaf - scorebef)/3;
    board.addScore((scoreaf - scorebef) * linesDestroy - (scoreaf - scorebef));
    board.setNormal(board.getNormal() + linesDestroy * 5);

}
```

- ⇨ Get 3 points when 1 line was destroyed, 3*n*n points when n lines were destroyed at the same time.

_In the **transposeMatrix** method, we will return the new matrix with inverse row and column number.

```
private int[][] transposeMatrix(int[][] matrix){
    int[][] temp = new int[matrix[0].length][matrix.length];
    for (int i = 0; i < matrix.length; i++)
        for (int j = 0; j < matrix[0].length; j++)
            temp[j][i] = matrix[i][j];
    return temp;
}
```

_In the **reverseRows** method, we will swap the first and last rows in matrix so that the shape after rotating looks "real".

```
private int[][] reverseRows(int[][] matrix){

    int middle = matrix.length/2;


    for(int i = 0; i < middle; i++)
    {
        int[] temp = matrix[i];

        matrix[i] = matrix[matrix.length - i - 1];
        matrix[matrix.length - i - 1] = temp;
    }

    return matrix;

}
```

_In the **rotateShape** method, we will set the current matrix of shape to new matrix after rotating.

```
public void rotateShape()
{

    int[][] rotatedShape = null;

    rotatedShape = transposeMatrix(coords);

    rotatedShape = reverseRows(rotatedShape);

    if((x + rotatedShape[0].length > 10) || (y + rotatedShape.length > 20))
    {
        return;
    }

    for(int row = 0; row < rotatedShape.length; row++)
    {
        for(int col = 0; col < rotatedShape[row].length; col ++)
        {
            if(rotatedShape[row][col] != 0)
            {
                if(board.getBoard()[y + row][x + col] != 0)
                {
                    return;
                }
            }
        }
    }
    coords = rotatedShape;
}
```

Besides, it is essential to prevent the shape from rotating out of the board.

_And the rest of methods, we will use it in another class.

```java
public class Shape {

    private int color;
    private int x, y;
    private long time, lastTime;
    private double normal;
    private int fast = 50;
    private double delay;
    private final double speed = .77;
    private BufferedImage block;
    private int[][] coords;
    private int[][] reference;
    private int deltaX;
    private Board board;
    private boolean collision = false, moveX = false;

    public Shape(int[][] coords, BufferedImage block, Board board, int color){
    public void update(){
    public void render(Graphics g){

    private void checkLine(){
    public void rotateShape(){
    private int[][] transposeMatrix(int[][] matrix){
    private int[][] reverseRows(int[][] matrix){
    public int getColor(){

    public void setDeltaX(int deltaX){

    public void speedUp(){

    public void speedDown(){

    public BufferedImage getBlock(){

    public int[][] getCoords(){

    public int getX(){

    public int getY(){
}
```

4. ImageLoader class

- Read game data files

```java
public class ImageLoader {

    public static BufferedImage loadImage(String path){
        try {
            return ImageIO.read(ImageLoader.class.getResource(path));
        } catch (IOException e) {
            e.printStackTrace();
            System.exit(1);
        }
        return null;
    }
}
```

5. Board class

- Create Board class to display blocks.

+ Use a small size matrix, marking the boxes with the number 1 means containing blocks, the number 0 means that does not contain the block.

```java
// create shapes

shapes[0] = new Shape(new int[][]{
    {1, 1, 1, 1}   // I shape;
}, blocks.getSubimage(0, 0, blockSize, blockSize), this, 1);

shapes[1] = new Shape(new int[][]{
    {1, 1, 1},
    {0, 1, 0},   // T shape;
}, blocks.getSubimage(blockSize, 0, blockSize, blockSize), this, 2);
```

_In **paintComponent** method, we use **drawimage** to set the pause and refresh button

```java
if(stopBounds.contains(mouseX, mouseY))
    g.drawImage(pause.getScaledInstance(pause.getWidth() + 3, pause.getHeight() + 3, BufferedImage.SCALE_DEFAULT)
            , stopBounds.x + 3, stopBounds.y + 3, null);
else
    g.drawImage(pause, stopBounds.x, stopBounds.y, null);

if(refreshBounds.contains(mouseX, mouseY))
    g.drawImage(refresh.getScaledInstance(refresh.getWidth() + 3, refresh.getHeight() + 3,
            BufferedImage.SCALE_DEFAULT), refreshBounds.x + 3, refreshBound
else
    g.drawImage(refresh, refreshBounds.x, refreshBounds.y, null);
```

Open image in new tab

Save image as...

We use a random method to set the next shape

_In the Board applet, we use the **keyPressed** process to respond when the user presses one of the arrow keys. The applet includes **KeyEvent.VK_UP** to rotate the

shape, **KeyEvent.VK_RIGHT** and **KeyEvent.VK_LEFT** to move shape right and left and KeyEvent.VK_DOWN to increase move speed

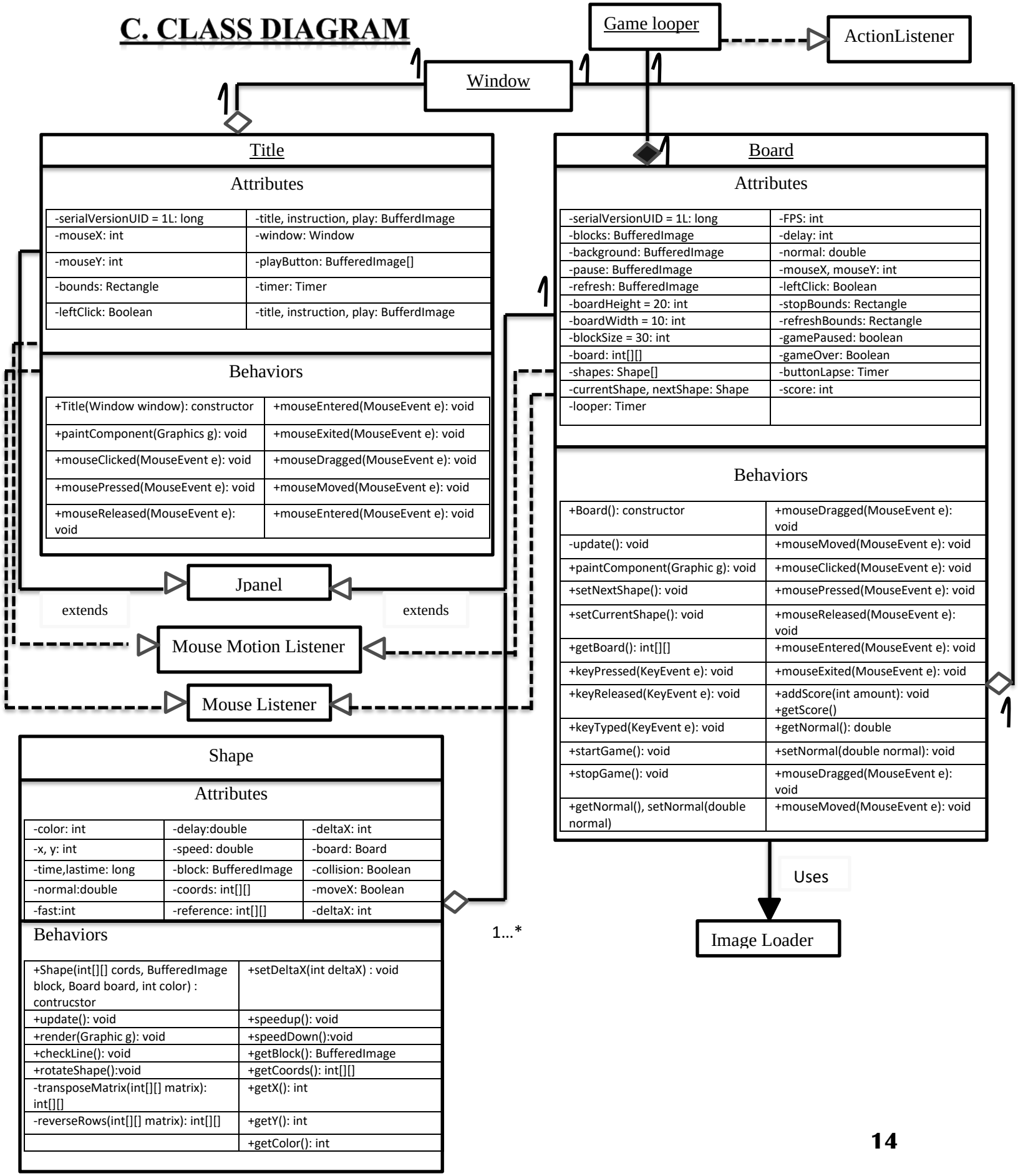
```
@Override
public void keyPressed(KeyEvent e) {
    if(e.getKeyCode() == KeyEvent.VK_UP)
        currentShape.rotateShape();
    if(e.getKeyCode() == KeyEvent.VK_RIGHT)
        currentShape.setDeltaX(1);
    if(e.getKeyCode() == KeyEvent.VK_LEFT)
        currentShape.setDeltaX(-1);
    if(e.getKeyCode() == KeyEvent.VK_DOWN)
        currentShape.speedUp();
}
```

```
86⊕     public Board(){
143
144⊕    private void update(){
160
161
162⊕    public void paintComponent(Graphics g){
242
243⊕    public void setNextShape(){
247
248⊕    public void setCurrentShape(){
265
266
267⊕    public int[][] getBoard(){
270
272⊕    public void keyPressed(KeyEvent e) {
283⊕    public void keyReleased(KeyEvent e) {
287
289⊕    public void keyTyped(KeyEvent e) {
292
293⊕    public void startGame(){
301⊕    public void stopGame(){
314
315⊕    class GameLooper implements ActionListener{
324
326⊕    public void mouseDragged(MouseEvent e) {
330
332⊕    public void mouseMoved(MouseEvent e) {
336
338⊕    public void mouseClicked(MouseEvent e) {
341
343⊕    public void mousePressed(MouseEvent e) {
347
349⊕    public void mouseReleased(MouseEvent e) {
353
355⊕    public void mouseEntered(MouseEvent e) {
358
360⊕    public void mouseExited(MouseEvent e) {
```

```java
    public void addScore(int amount){
        score += amount;
    }

    public int getScore() {
        return score;
    }

    public double getNormal() {
        return this.normal;
    }

    public void setNormal(double normal) {
        this.normal = normal;
    }

}
```

⇨ Set the speed of the new shapes when the previous shape collides.

C. CLASS DIAGRAM



D. CLASS STAGE DIAGRAM