VIETNAM NATIONAL UNIVERSITY OF HOCHIMINH CITY

THE INTERNATIONAL UNIVERSITY

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

TETRIS GAME REPORT

Lecturer: Pham Quoc Son Lam

Tran Thanh Tung

MEMBERS:

Tạ Vĩ Khang – ITITIU20226

Nguyễn Bá Phúc – ITITIU20278

Lê Hoàng Thái Tuấn – ITITIU20340

Huỳnh Thanh Thảo – ITDSIU20126

Ho Chi Minh City, Vietnam 2022-2023

I. Outline of the Project

1. Itinerary and Goals

1.1 Inspiration

Tetris is a video game inspired by the classic puzzle game Pentomino. The goal of this game is to fill the row bar and try to earn as many points as possible. Tetris makes use of a number of cognitive processes, such as pattern recognition, memorization, and matching, which is why we decided to create this game. Additionally, it promotes restraint, judgment, and problem-solving. A complex web of thinking processes is simultaneously activated and developed by this simple game.

1.2 The game rules

The Tetromino will be randomly created, and the player will use the arrow keys from the keyboard to control them in suitable places on the ground. The player may continue by filling in the empty areas after the finished lines vanish and award them with points. When the uncleared lines reach the top of the playing field, the game is over. The player's score will increase the longer they can postpone this result.

There are four main control keys:

- Up arrow key: Rotate the block.
- Left arrow key: Move left.
- Right arrow key: Move right.
- Down arrow key: Speed up

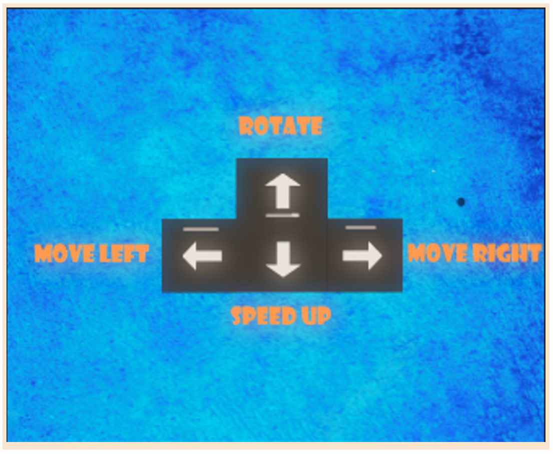


Figure 2. Control Key

Challenge mode:

By clicking "New Game," we can select different levels of difficulty, which affect the speed of the falling blocks and the number of points we receive.

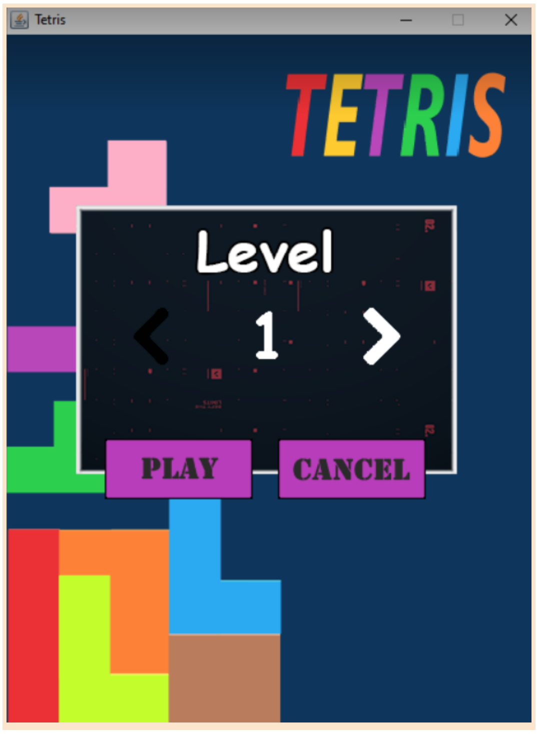


Figure 3. Challenge mode (Level)

1.3 Some Tetris Nomenclature

In this report, we have come up with a set of terms for the different things in the game program.

- **Board -** The board is made up of 10 x 20 spaces where the blocks fall and stack up.

Figure 4. Board

- **Box -** A box is a single filled-in square space on the board.

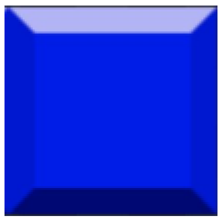


Figure 5. Box

- **Block -** The things that fall from the top of the board that the player can rotate and position. Each tile has a shape and is made up of 4 boxes.
- **Shape -** The shapes are the different types of blocks in the game. The names of the shapes are T, S, Z, J, L, I, and O.

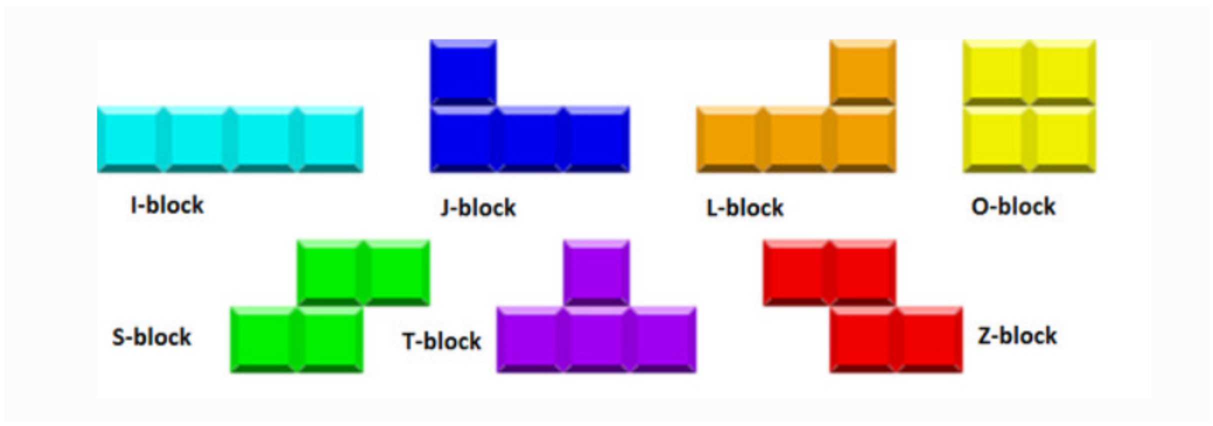


Figure 6. 7 Shapes

- **Template -** A list of shape data structures that represents all the possible rotations of a shape. These are stored in variables with names like ShapeT, ShapeS, ShapeZ, ShapeJ, ShapeL, ShapeI, and ShapeO.
- **Collision** – When a block has either reached the bottom of the board or is touching a box on the board, we say that the piece has landed. At that point, the next block should start falling

2. Goals of the project

2.1 Goals of the game

The aim of Tetris is simple; you bring blocks down from the top of the screen. You can move the blocks around, either left to right, and/or you can rotate them. The blocks fall at a specific rate, but you can make them fall faster if you're sure of your positioning. Your objective is to get all the blocks to fill all the empty space in a line at the bottom of the screen; whenever you do this, the blocks disappear and you get some points.

A goal gives us a reason to play the game. Tetris offers an incredibly simple reason to play—pitting your wits against the computerized block dropper in order to last as long as you can.

2.2 Goals of the game designer

Tetris helps players exercise their minds while relaxing while playing, resulting in better spatial and visual reasoning, training concentration and attention, improving memory, raising IQ, improving problem-solving ability, increased productivity, better collaboration and teamwork, an improved mood, and reduced stress.

II. Describes project

1. Use Case Diagram

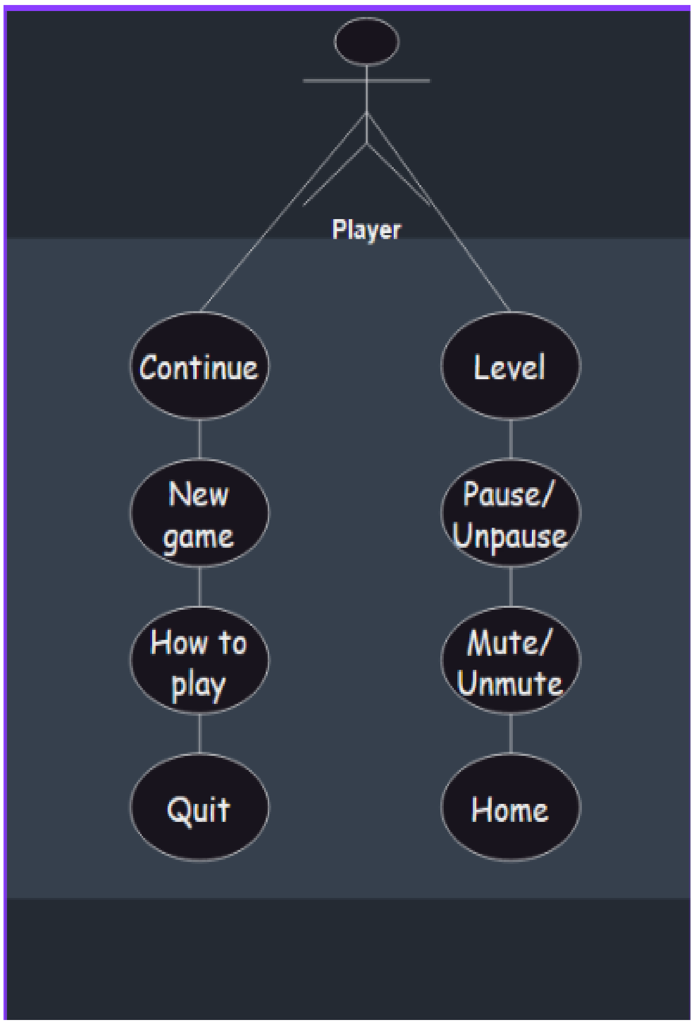


Figure 7. Use Case Diagram

- Continue: Continue the game you left unfinished.

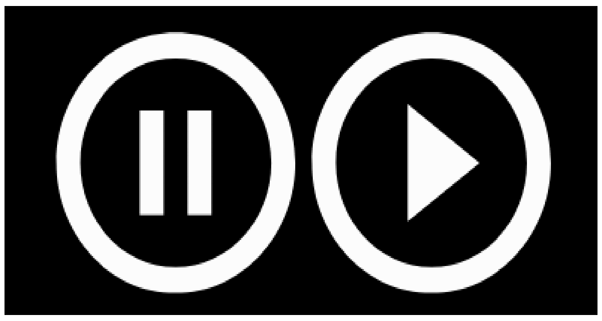
- New game: Start a new game.

- How to play: Displays the playing guide.

- Quit: Exit the current game.

- Level

- Pause/Unpause: Temporarily stop (Pause) and resume (Unpause) the currently playing game.



- Mute/Unmute: Turn on or off the game's soundtrack.

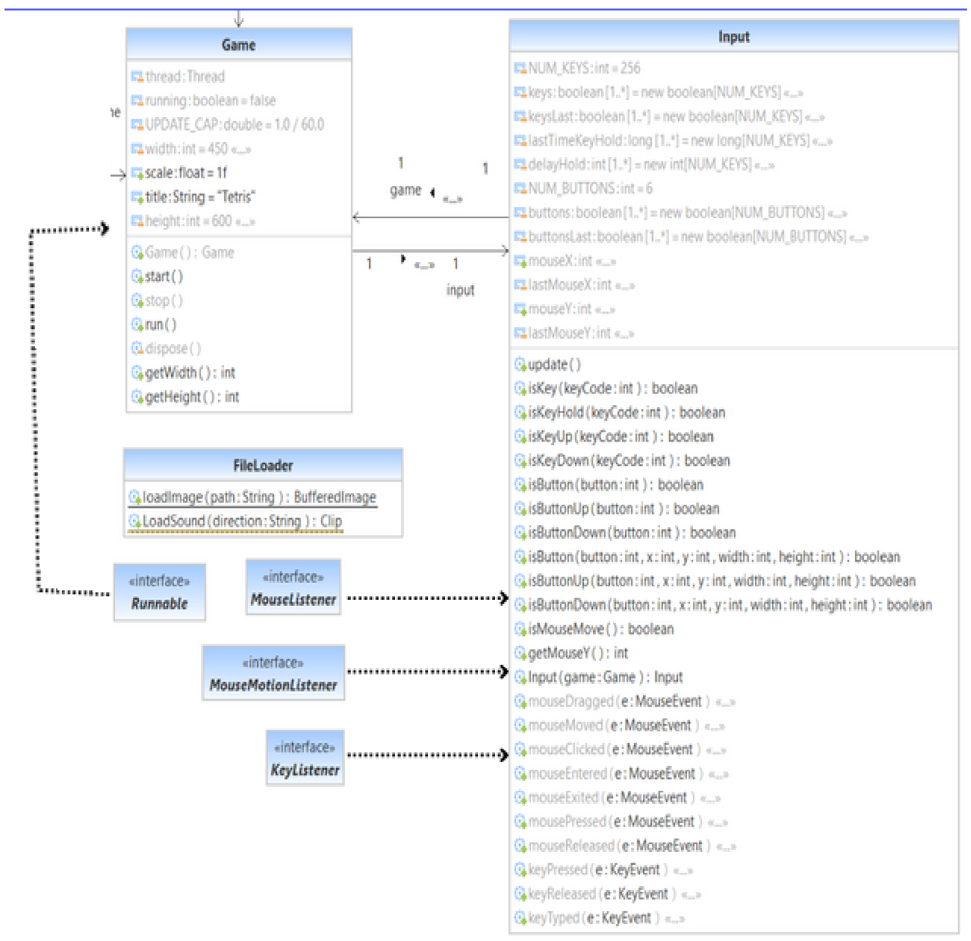
- Home: back to Menu screen.

2. Implementation Ideas

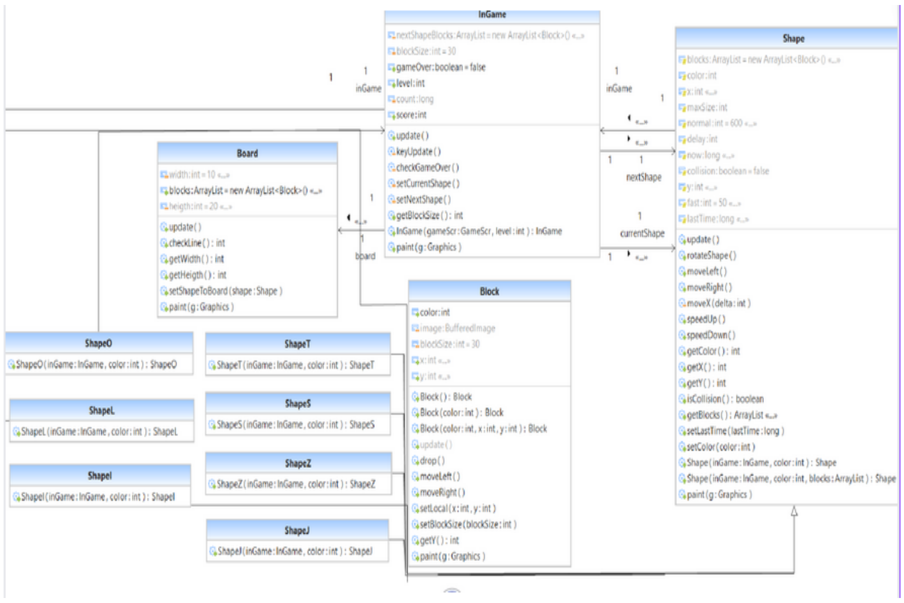
2.1 General class diagram

2.2 Class diagrams for each package

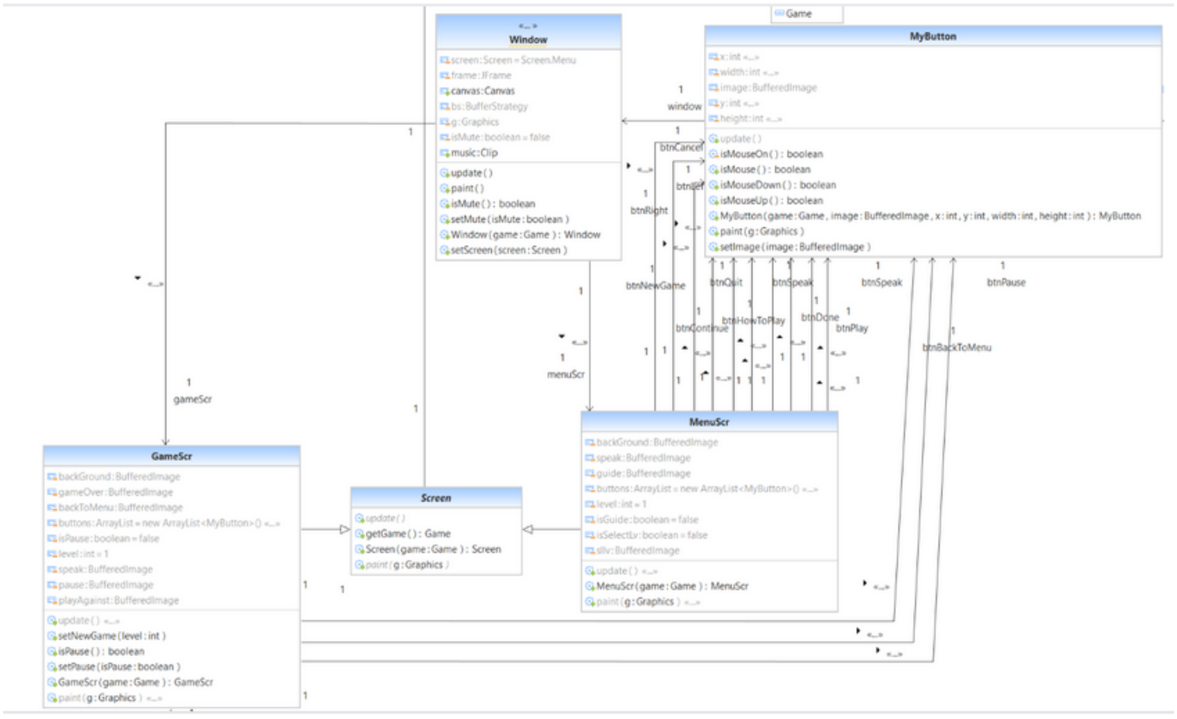
Engine package


Game
thread : Thread
running : boolean = false
UPDATE_CAP : double = 1.0 / 60.0
width : int = 450 «...»
scale : float = 1f
title : String = "Tetris"
height : int = 600 «...»
Game ( ) : Game
start ( )
stop ( )
run ( )
dispose ( )
getWidth ( ) : int
getHeight ( ) : int
1
game
«...»
1
1
«...»
1
input
FileLoader
loadImage (path : String) : BufferedImage
LoadSound (direction : String) : Clip
«interface»
Runnable
«interface»
MouseListener
«interface»
MouseMotionListener
«interface»
KeyListener
Input
NUM_KEYS : int = 256
keys : boolean [1..*] = new boolean[NUM_KEYS] «...»
keysLast : boolean [1..*] = new boolean[NUM_KEYS] «...»
lastTimeKeyHold : long [1..*] = new long[NUM_KEYS] «...»
delayHold : int [1..*] = new int[NUM_KEYS] «...»
NUM_BUTTONS : int = 6
buttons : boolean [1..*] = new boolean[NUM_BUTTONS] «...»
buttonsLast : boolean [1..*] = new boolean[NUM_BUTTONS] «...»
mouseX : int «...»
lastMouseX : int «...»
mouseY : int «...»
lastMouseY : int «...»
update ( )
isKey (keyCode : int) : boolean
isKeyHold (keyCode : int) : boolean
isKeyUp (keyCode : int) : boolean
isKeyDown (keyCode : int) : boolean
isButton (button : int) : boolean
isButtonUp (button : int) : boolean
isButtonDown (button : int) : boolean
isButton (button : int , x : int , y : int , width : int , height : int) : boolean
isButtonUp (button : int , x : int , y : int , width : int , height : int) : boolean
isButtonDown (button : int , x : int , y : int , width : int , height : int) : boolean
isMouseMove ( ) : boolean
getMouseY ( ) : int
Input (game : Game) : Input
mouseDragged (e : MouseEvent) «...»
mouseMoved (e : MouseEvent) «...»
mouseClicked (e : MouseEvent) «...»
mouseEntered (e : MouseEvent) «...»
mouseExited (e : MouseEvent) «...»
mousePressed (e : MouseEvent) «...»
mouseReleased (e : MouseEvent) «...»
keyPressed (e : KeyEvent) «...»
keyReleased (e : KeyEvent) «...»
keyTyped (e : KeyEvent) «...»

Algorithm package



Gui package



2.3 Engine

-Game

+This class initializes and starts the game loop

-Input

+Receive signals from the keyboard

+Read data from audio and video files.

2.4 Algorithm

-Block

+Declare the variable color, block size, and position of a single block.

-Board

+Create a grid array containing the moving shapes.

-InGame

+Calculate conditions and control objects in the game

-Shape

+Create different shapes of blocks. There are a total of 7 shapes in the game tetris.

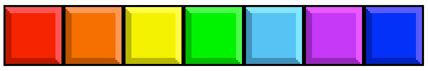
2.5 Graphical User Interface (GUI)

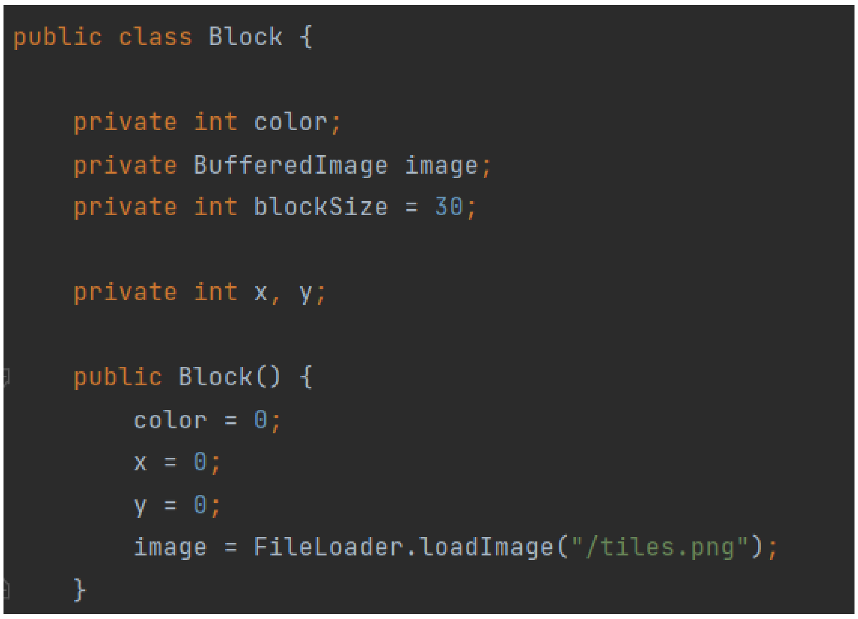
2.5.1 Game Display

- Class Window: Background of game screens.
- Class MyButton: command buttons in the interface.
- Class MenuScr: Menu screen.
- Class GameScr: Game screen

3. Conduct

a) **Create an image file consisting of 7 color blocks:** The image file size is 210x30, which means that the image consists of 7 blocks, and each block is 30x30 in size.



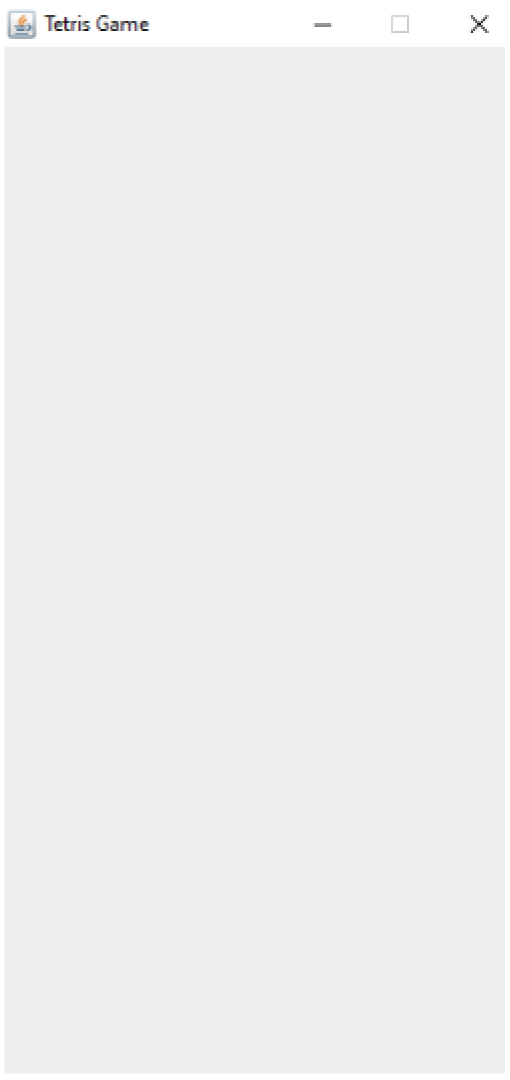


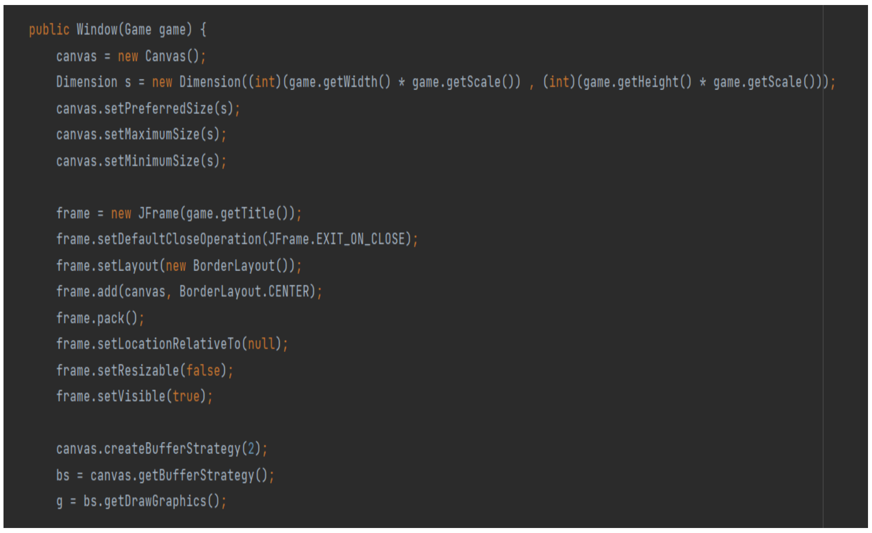
```
public class Block {

    private int color;
    private BufferedImage image;
    private int blockSize = 30;

    private int x, y;

    public Block() {
        color = 0;
        x = 0;
        y = 0;
        image = FileLoader.loadImage("/tiles.png");
    }
```

b) **Class division:** There are 3 main classes:

- class Window: create the main window, the upper horizontal bars, and the size of the window. Run the main window, and load the Board game.
- class Board: divide color image blocks, save them into 7 shapes, random shapes, and draw table lines.
- class Shape: set the properties of Shape, render Shape, update Shape after each down, and transform Shape.

c) **Setup the Window Game**

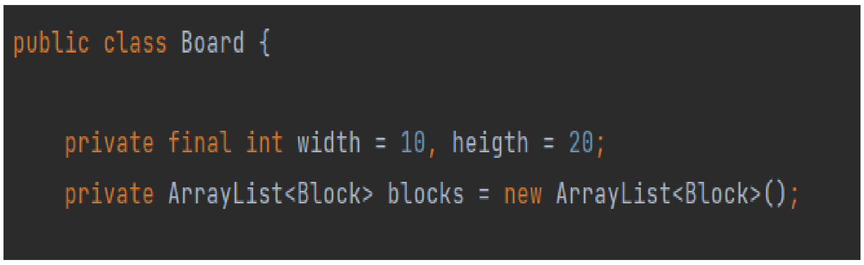




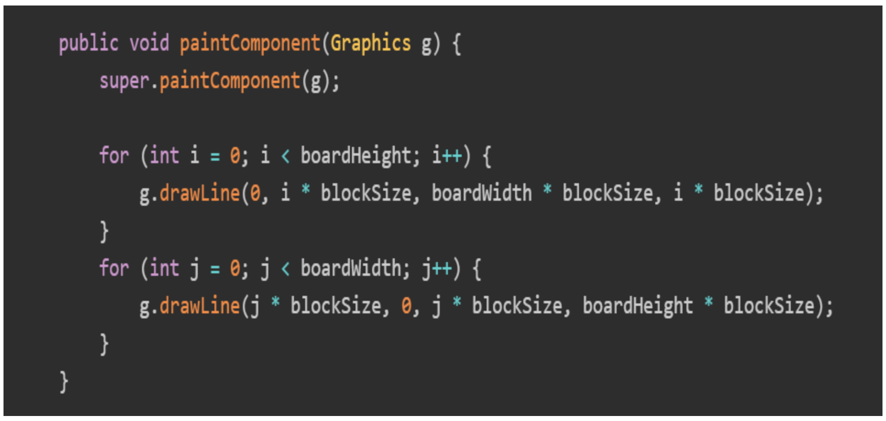
```java
public Window(Game game) {
    canvas = new Canvas();
    Dimension s = new Dimension((int)(game.getWidth() * game.getScale()) , (int)(game.getHeight() * game.getScale()));
    canvas.setPreferredSize(s);
    canvas.setMaximumSize(s);
    canvas.setMinimumSize(s);

    frame = new JFrame(game.getTitle());
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setLayout(new BorderLayout());
    frame.add(canvas, BorderLayout.CENTER);
    frame.pack();
    frame.setLocationRelativeTo(null);
    frame.setResizable(false);
    frame.setVisible(true);

    canvas.createBufferStrategy(2);
    bs = canvas.getBufferStrategy();
    g = bs.getDrawGraphics();
```

d) Draw a Board Game:

In the Board Game of size 300x600, we divide the width into 10 parts, and the length into 20 parts. Draw black lines that divide to form the Board.



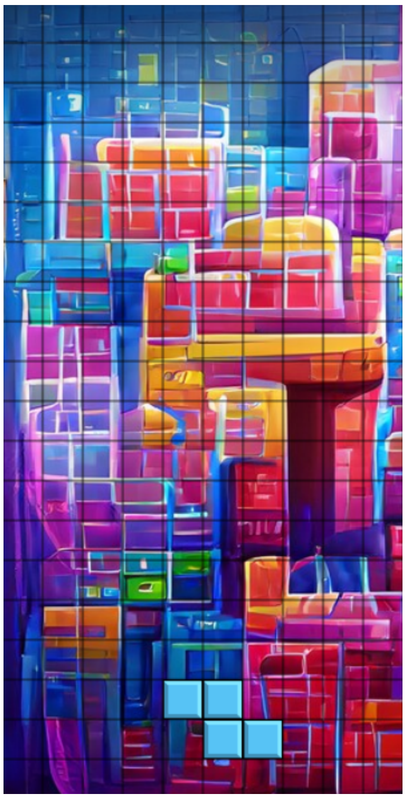
```java
public class Board {

    private final int width = 10, heigth = 20;
    private ArrayList<Block> blocks = new ArrayList<Block>();
```



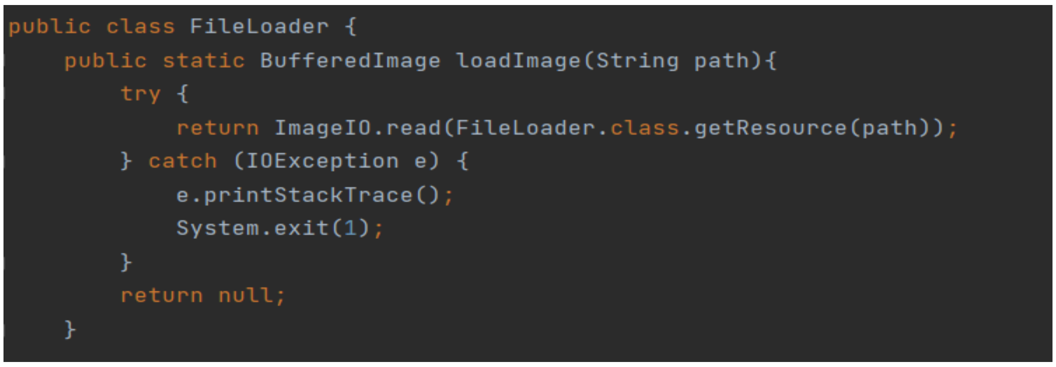
```java
public void paintComponent(Graphics g) {
    super.paintComponent(g);

    for (int i = 0; i < boardHeight; i++) {
        g.drawLine(0, i * blockSize, boardWidth * blockSize, i * blockSize);
    }
    for (int j = 0; j < boardWidth; j++) {
        g.drawLine(j * blockSize, 0, j * blockSize, boardHeight * blockSize);
    }
}
```

e) Load images, divide Blocks, and create Shape

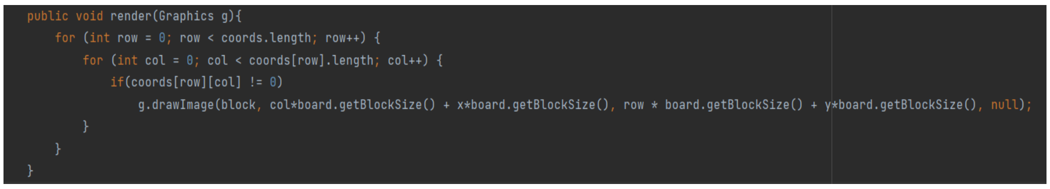
After having the Board game, we will load the tiles.png image containing the 7 color Blocks as shown above. Proceed to divide the image into 7 blocks with 7 colors of size 30x30.



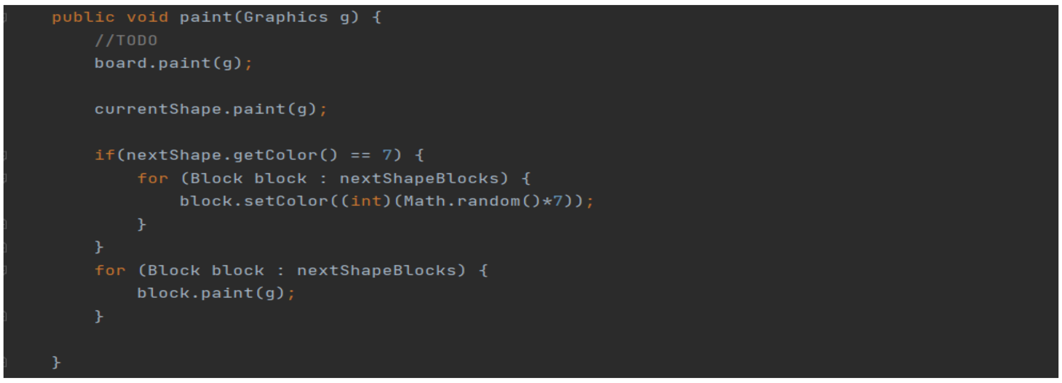
Next, we create a Shape that stores some current information, such as a color block, and position. Then create a render function to draw Blocks of the same color on the Board game to create Shape.



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

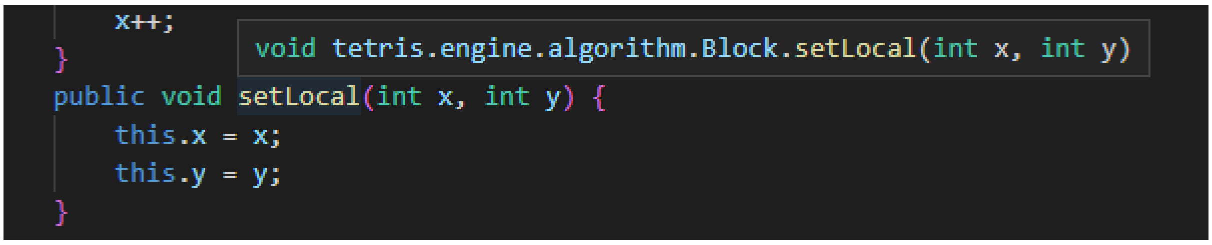


```java
public void render(Graphics g){
    for (int row = 0; row < coords.length; row++) {
        for (int col = 0; col < coords[row].length; col++) {
            if(coords[row][col] != 0)
                g.drawImage(block, col*board.getBlockSize() + x*board.getBlockSize(), row * board.getBlockSize() + y*board.getBlockSize(), null);
        }
    }
}
```



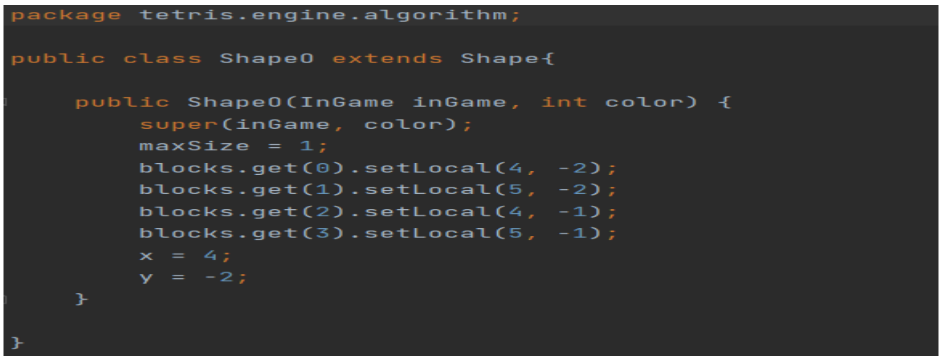
```java
    public void paint(Graphics g) {
        //TODO
        board.paint(g);

        currentShape.paint(g);

        if(nextShape.getColor() == 7) {
            for (Block block : nextShapeBlocks) {
                block.setColor((int)(Math.random()*7));
            }
        }
        for (Block block : nextShapeBlocks) {
            block.paint(g);
        }

    }
```

Shape includes 7 shapes, including O, I, S, Z, L, J, and T shapes. The division is that we use a coordinate x,y to display the shapes of Shape. Then combine the coordinates to get the shape of the Shape.



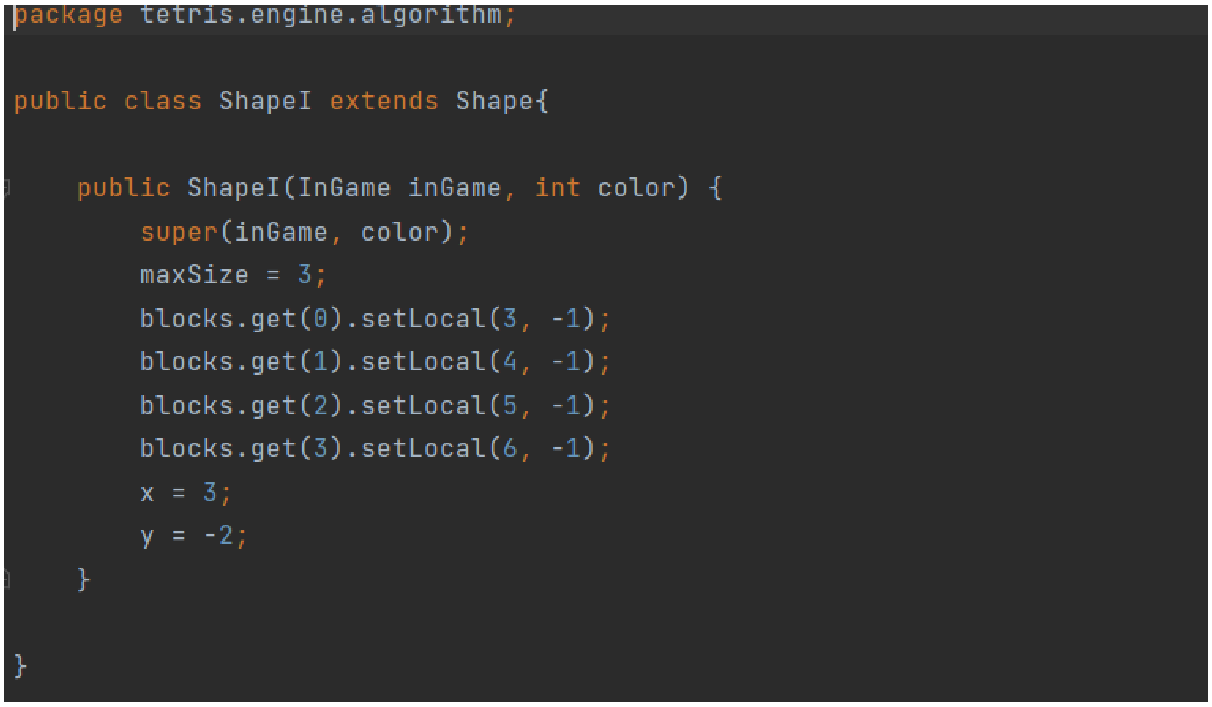
```java
        x++;
    }
    public void setLocal(int x, int y) {
        this.x = x;
        this.y = y;
    }
```

- **ShapeO**



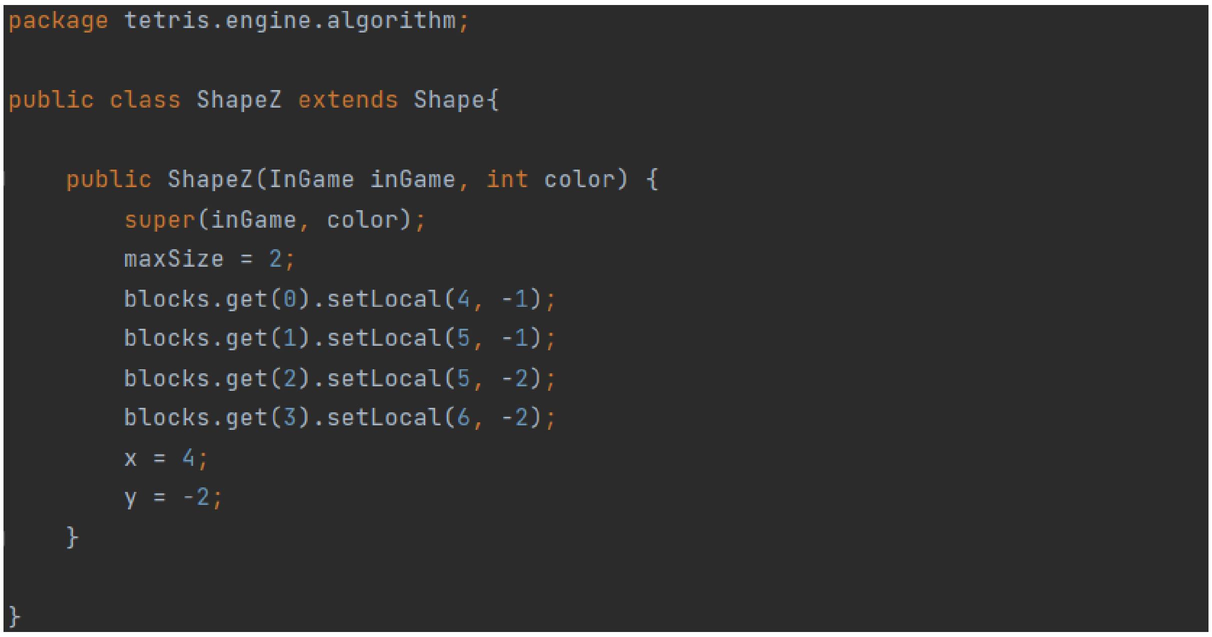
```java
package tetris.engine.algorithm;

public class ShapeO extends Shape{

    public ShapeO(InGame inGame, int color) {
        super(inGame, color);
        maxSize = 1;
        blocks.get(0).setLocal(4, -2);
        blocks.get(1).setLocal(5, -2);
        blocks.get(2).setLocal(4, -1);
        blocks.get(3).setLocal(5, -1);
        x = 4;
        y = -2;
    }

}
```

- **ShapeI**



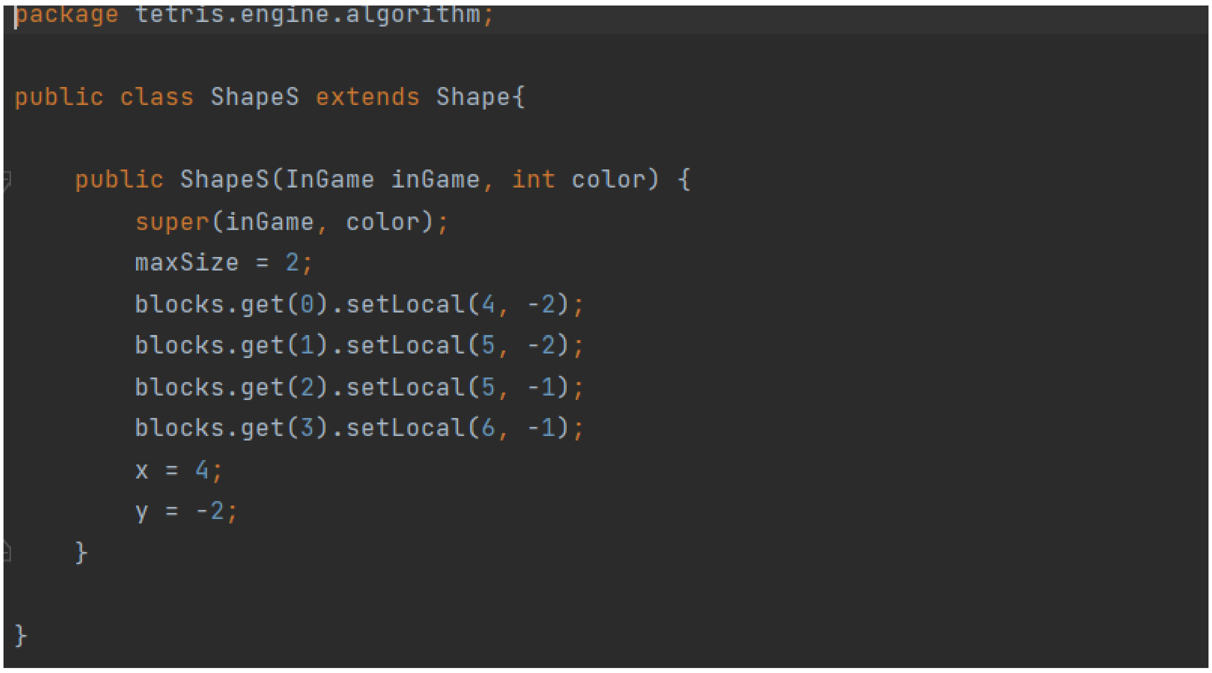
```java
package tetris.engine.algorithm;

public class ShapeI extends Shape{

    public ShapeI(InGame inGame, int color) {
        super(inGame, color);
        maxSize = 3;
        blocks.get(0).setLocal(3, -1);
        blocks.get(1).setLocal(4, -1);
        blocks.get(2).setLocal(5, -1);
        blocks.get(3).setLocal(6, -1);
        x = 3;
        y = -2;
    }

}
```

- **ShapeZ**



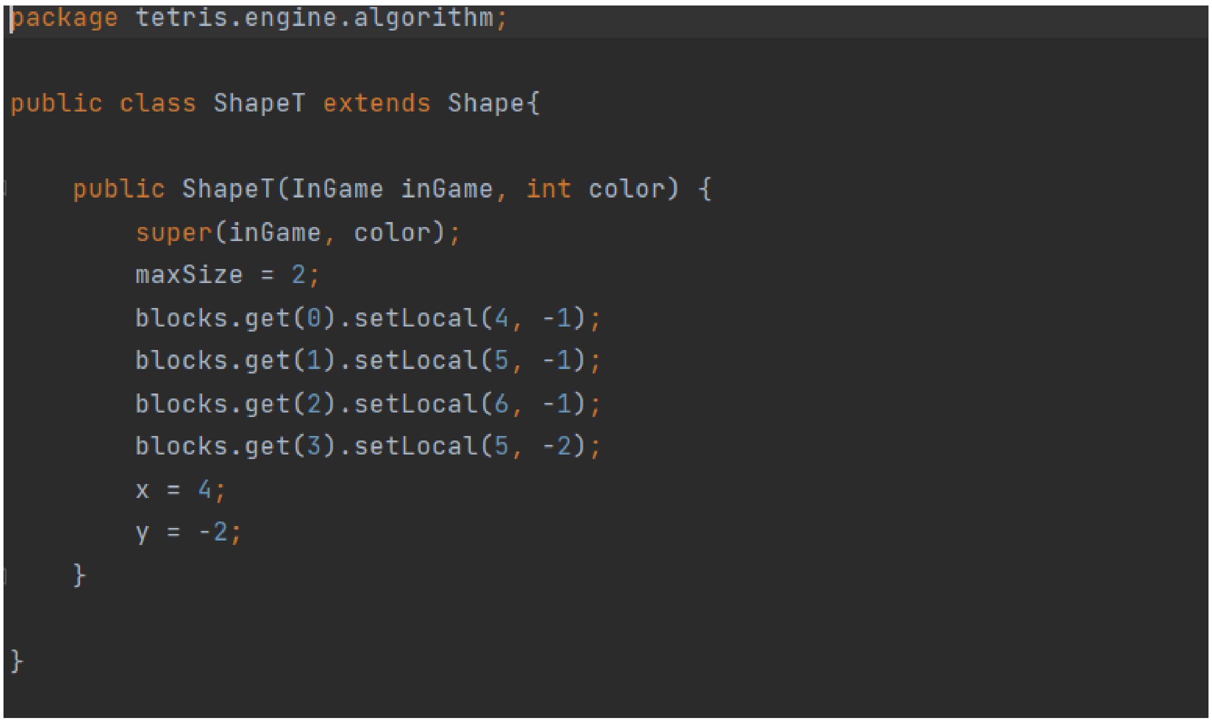
```java
package tetris.engine.algorithm;

public class ShapeZ extends Shape{

    public ShapeZ(InGame inGame, int color) {
        super(inGame, color);
        maxSize = 2;
        blocks.get(0).setLocal(4, -1);
        blocks.get(1).setLocal(5, -1);
        blocks.get(2).setLocal(5, -2);
        blocks.get(3).setLocal(6, -2);
        x = 4;
        y = -2;
    }

}
```

- **ShapeS**



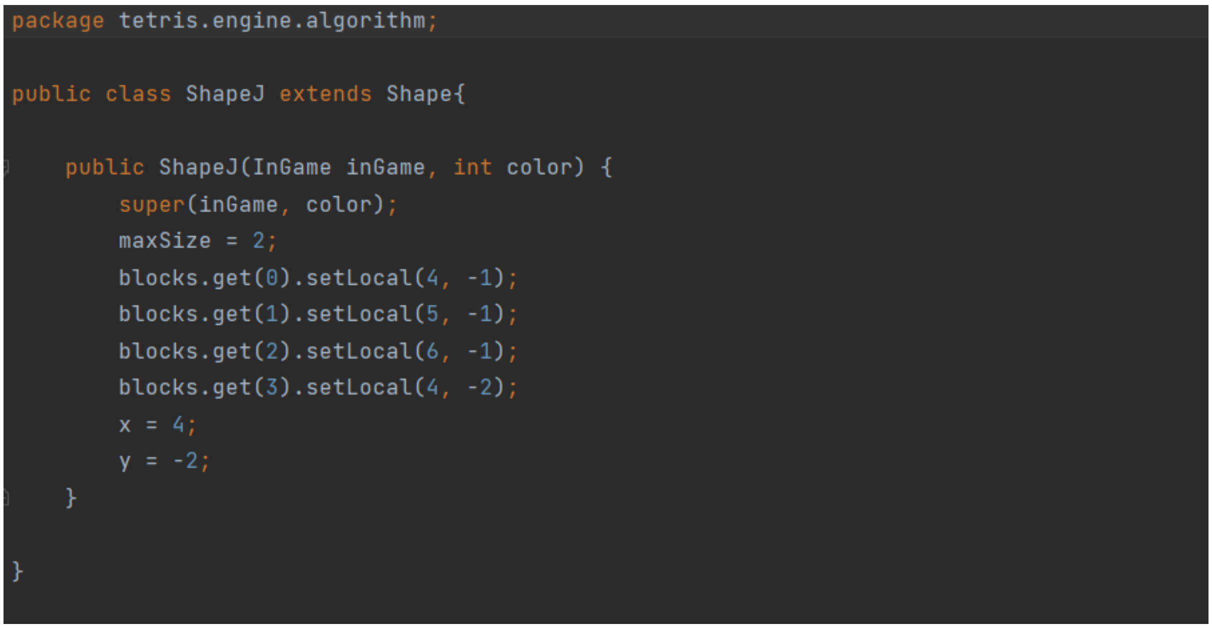
```java
package tetris.engine.algorithm;

public class ShapeS extends Shape{

    public ShapeS(InGame inGame, int color) {
        super(inGame, color);
        maxSize = 2;
        blocks.get(0).setLocal(4, -2);
        blocks.get(1).setLocal(5, -2);
        blocks.get(2).setLocal(5, -1);
        blocks.get(3).setLocal(6, -1);
        x = 4;
        y = -2;
    }

}
```

- **ShapeT**



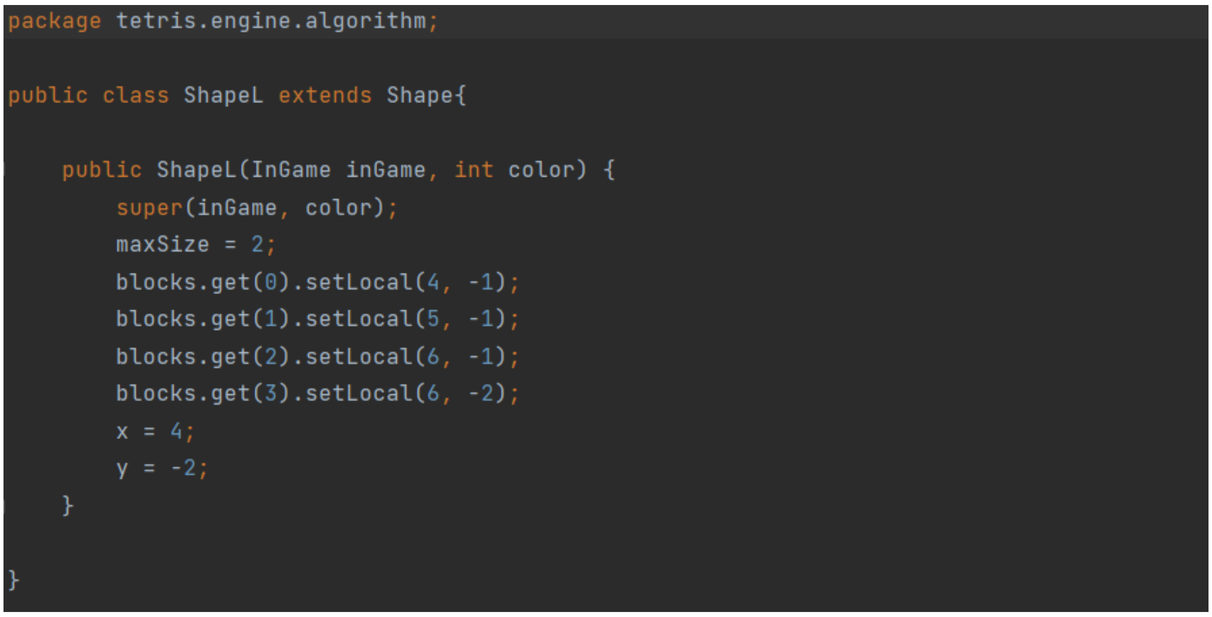
```java
package tetris.engine.algorithm;

public class ShapeT extends Shape{

    public ShapeT(InGame inGame, int color) {
        super(inGame, color);
        maxSize = 2;
        blocks.get(0).setLocal(4, -1);
        blocks.get(1).setLocal(5, -1);
        blocks.get(2).setLocal(6, -1);
        blocks.get(3).setLocal(5, -2);
        x = 4;
        y = -2;
    }

}
```

- **ShapeJ**



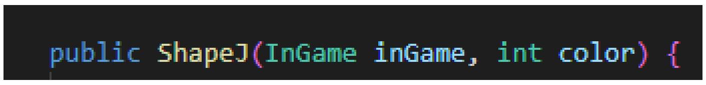
```java
package tetris.engine.algorithm;

public class ShapeJ extends Shape{

    public ShapeJ(InGame inGame, int color) {
        super(inGame, color);
        maxSize = 2;
        blocks.get(0).setLocal(4, -1);
        blocks.get(1).setLocal(5, -1);
        blocks.get(2).setLocal(6, -1);
        blocks.get(3).setLocal(4, -2);
        x = 4;
        y = -2;
    }

}
```

- **ShapeL**



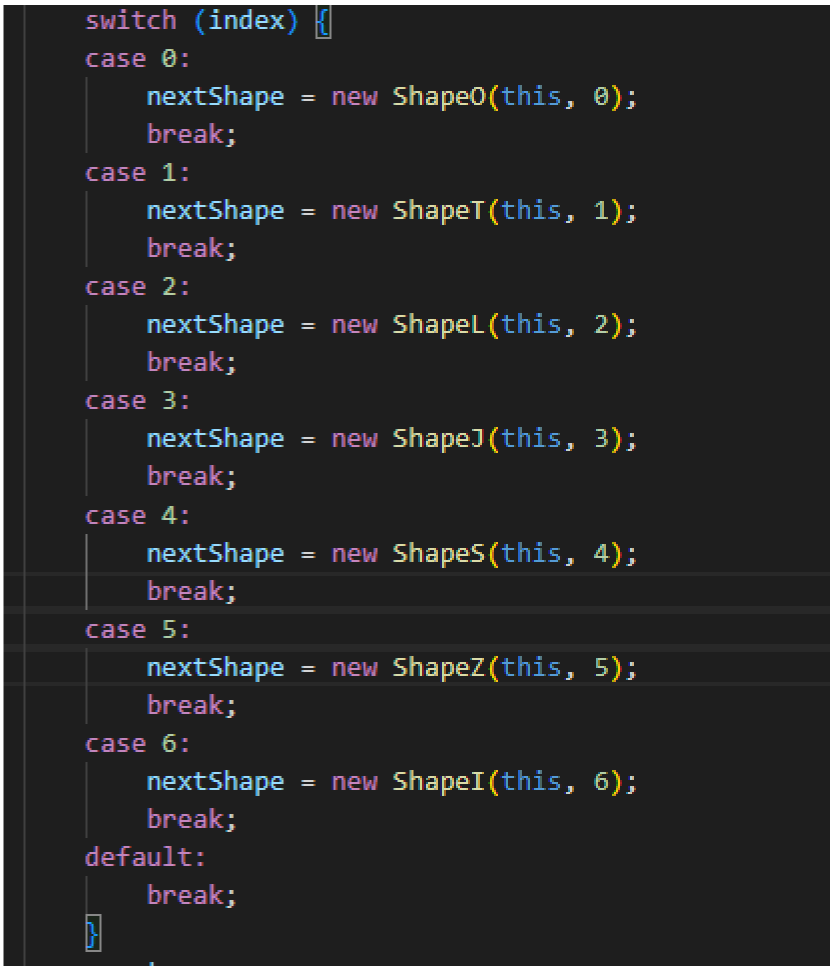
```java
package tetris.engine.algorithm;

public class ShapeL extends Shape{

    public ShapeL(InGame inGame, int color) {
        super(inGame, color);
        maxSize = 2;
        blocks.get(0).setLocal(4, -1);
        blocks.get(1).setLocal(5, -1);
        blocks.get(2).setLocal(6, -1);
        blocks.get(3).setLocal(6, -2);
        x = 4;
        y = -2;
    }

}
```

f) Draw for the Shape



```java
public ShapeJ(InGame inGame, int color) {
```

From the constructor of ShapeJ for example

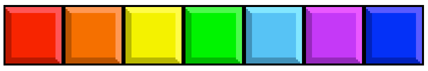


```
switch (index) {
case 0:
    nextShape = new ShapeO(this, 0);
    break;
case 1:
    nextShape = new ShapeT(this, 1);
    break;
case 2:
    nextShape = new ShapeL(this, 2);
    break;
case 3:
    nextShape = new ShapeJ(this, 3);
    break;
case 4:
    nextShape = new ShapeS(this, 4);
    break;
case 5:
    nextShape = new ShapeZ(this, 5);
    break;
case 6:
    nextShape = new ShapeI(this, 6);
    break;
default:
    break;
}
```

The color of each Shape based on these code lines

and the number 0 to 6 represent for the color left to right of

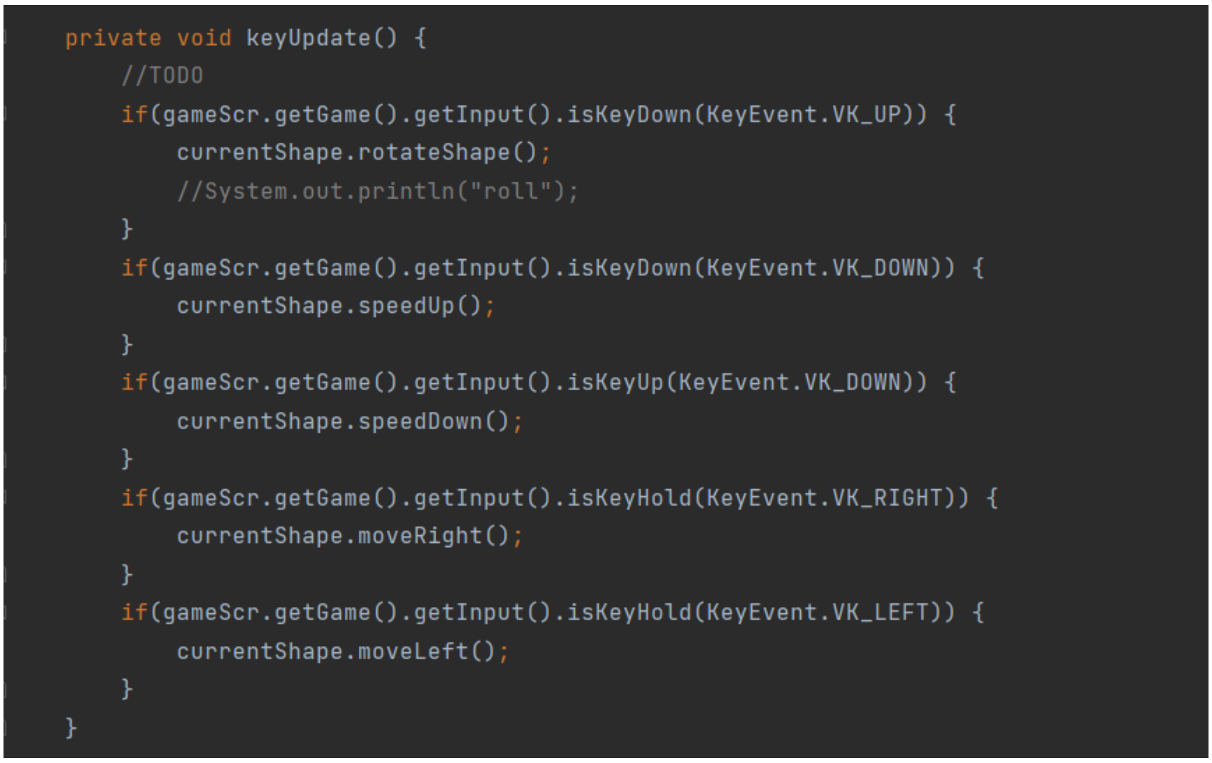
this figure:



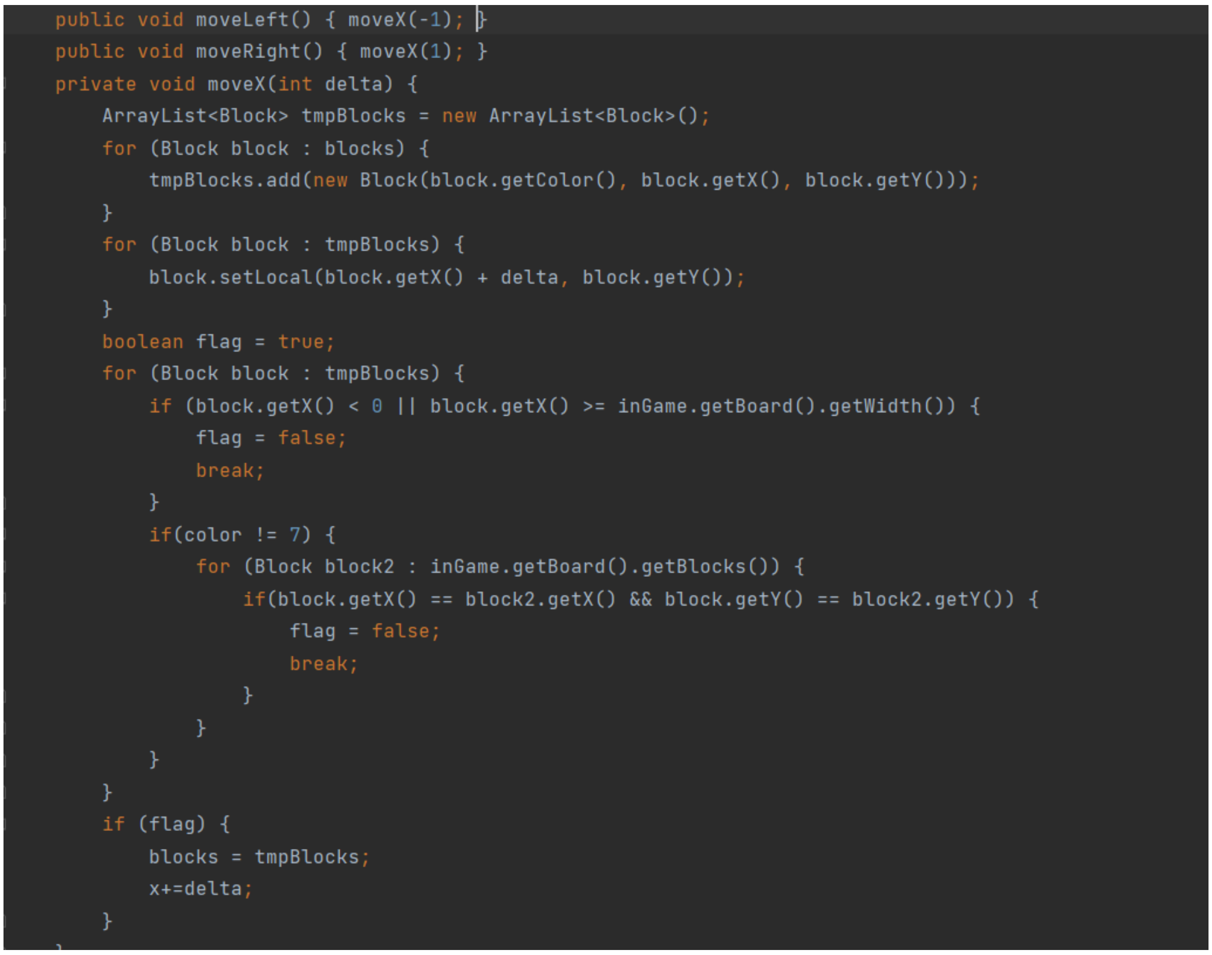
g) Create function for Shape

Create manipulations with Shape and handle collisions with Board borders.

Normally, Shape will automatically move from top to bottom. The operation key includes the left and right keys to move the Shape left or right. The down key is used to speed up Shape. Normally, deltaX = 0 and deltaX = -1 will move the shape left, and deltaX = 1 will move the shape right.

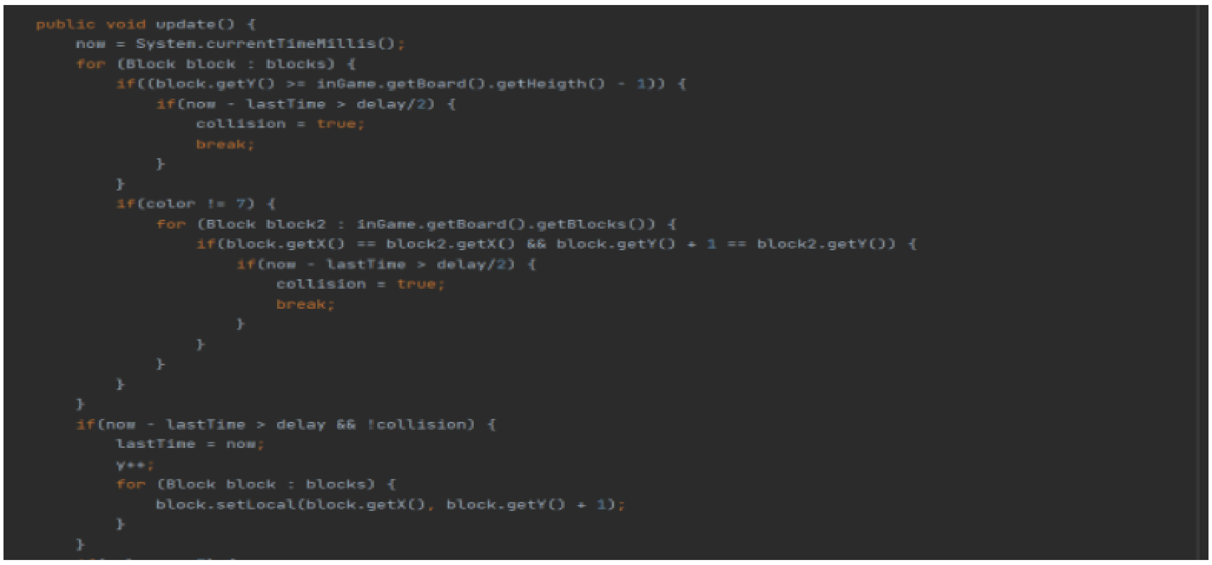


```
private void keyUpdate() {
    //TODO
    if(gameScr.getGame().getInput().isKeyDown(KeyEvent.VK_UP)) {
        currentShape.rotateShape();
        //System.out.println("roll");
    }
    if(gameScr.getGame().getInput().isKeyDown(KeyEvent.VK_DOWN)) {
        currentShape.speedUp();
    }
    if(gameScr.getGame().getInput().isKeyUp(KeyEvent.VK_DOWN)) {
        currentShape.speedDown();
    }
    if(gameScr.getGame().getInput().isKeyHold(KeyEvent.VK_RIGHT)) {
        currentShape.moveRight();
    }
    if(gameScr.getGame().getInput().isKeyHold(KeyEvent.VK_LEFT)) {
        currentShape.moveLeft();
    }
}
```



```java
    public void moveLeft() { moveX(-1); }
    public void moveRight() { moveX(1); }
    private void moveX(int delta) {
        ArrayList<Block> tmpBlocks = new ArrayList<Block>();
        for (Block block : blocks) {
            tmpBlocks.add(new Block(block.getColor(), block.getX(), block.getY()));
        }
        for (Block block : tmpBlocks) {
            block.setLocal(block.getX() + delta, block.getY());
        }
        boolean flag = true;
        for (Block block : tmpBlocks) {
            if (block.getX() < 0 || block.getX() >= inGame.getBoard().getWidth()) {
                flag = false;
                break;
            }
            if(color != 7) {
                for (Block block2 : inGame.getBoard().getBlocks()) {
                    if(block.getX() == block2.getX() && block.getY() == block2.getY()) {
                        flag = false;
                        break;
                    }
                }
            }
        }
        if (flag) {
            blocks = tmpBlocks;
            x+=delta;
        }
```

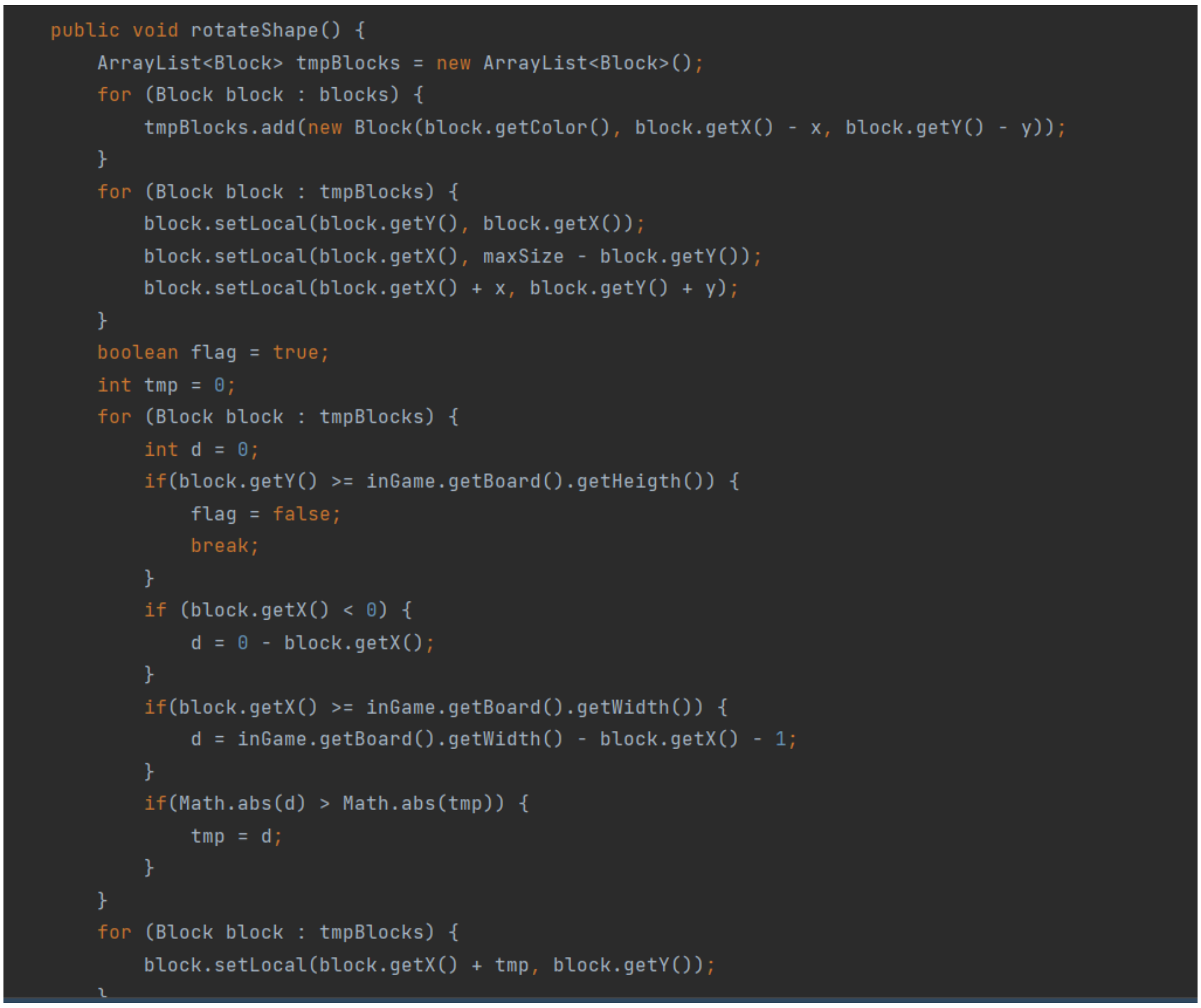
When we encounter the left and right borders, we will cancel the Control Key. The blocks will come to a halt if they come into contact with the ground or any of the objects on the field.



```java
    public void update() {
        now = System.currentTimeMillis();
        for (Block block : blocks) {
            if((block.getY() >= inGame.getBoard().getHeigth() - 1)) {
                if(now - lastTime > delay/2) {
                    collision = true;
                    break;
                }
            }
            if(color != 7) {
                for (Block block2 : inGame.getBoard().getBlocks()) {
                    if(block.getX() == block2.getX() && block.getY() + 1 == block2.getY()) {
                        if(now - lastTime > delay/2) {
                            collision = true;
                            break;
                        }
                    }
                }
            }
        }
        if(now - lastTime > delay && !collision) {
            lastTime = now;
            y++;
            for (Block block : blocks) {
                block.setLocal(block.getX(), block.getY() + 1);
            }
        }
```

h) Rotate Shape

Create an up key operation to rotate the Shape. As above, each Shape will be defined as a vector consisting of parts x and y. therefore, the Shape will be rotated according the formula: change the x and y values and size.



```
public void rotateShape() {
    ArrayList<Block> tmpBlocks = new ArrayList<Block>();
    for (Block block : blocks) {
        tmpBlocks.add(new Block(block.getColor(), block.getX() - x, block.getY() - y));
    }
    for (Block block : tmpBlocks) {
        block.setLocal(block.getY(), block.getX());
        block.setLocal(block.getX(), maxSize - block.getY());
        block.setLocal(block.getX() + x, block.getY() + y);
    }
    boolean flag = true;
    int tmp = 0;
    for (Block block : tmpBlocks) {
        int d = 0;
        if(block.getY() >= inGame.getBoard().getHeigth()) {
            flag = false;
            break;
        }
        if (block.getX() < 0) {
            d = 0 - block.getX();
        }
        if(block.getX() >= inGame.getBoard().getWidth()) {
            d = inGame.getBoard().getWidth() - block.getX() - 1;
        }
        if(Math.abs(d) > Math.abs(tmp)) {
            tmp = d;
        }
    }
    for (Block block : tmpBlocks) {
        block.setLocal(block.getX() + tmp, block.getY());
    for (Block block : tmpBlocks) {
        block.setLocal(block.getX() + tmp, block.getY());
    }
    x += tmp;
    if(color != 7) {
        for (Block block : tmpBlocks) {
            for (Block block2 : inGame.getBoard().getBlocks()) {
                if(block.getX() == block2.getX() && block.getY() == block2.getY()) {
                    flag = false;
                    break;
                }
            }
        }
    }
    if (flag) {
        blocks = tmpBlocks;
    }
}
```

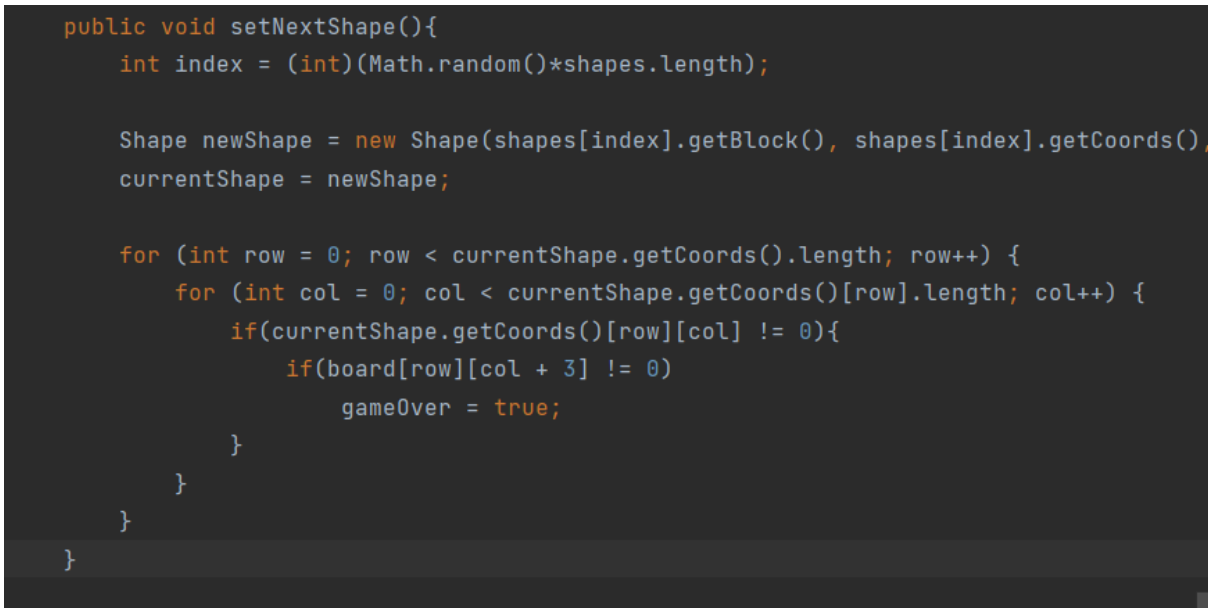
i) Get Score

When a row is filled with all the empty cells, it will automatically disappear and move the rows from top to bottom.

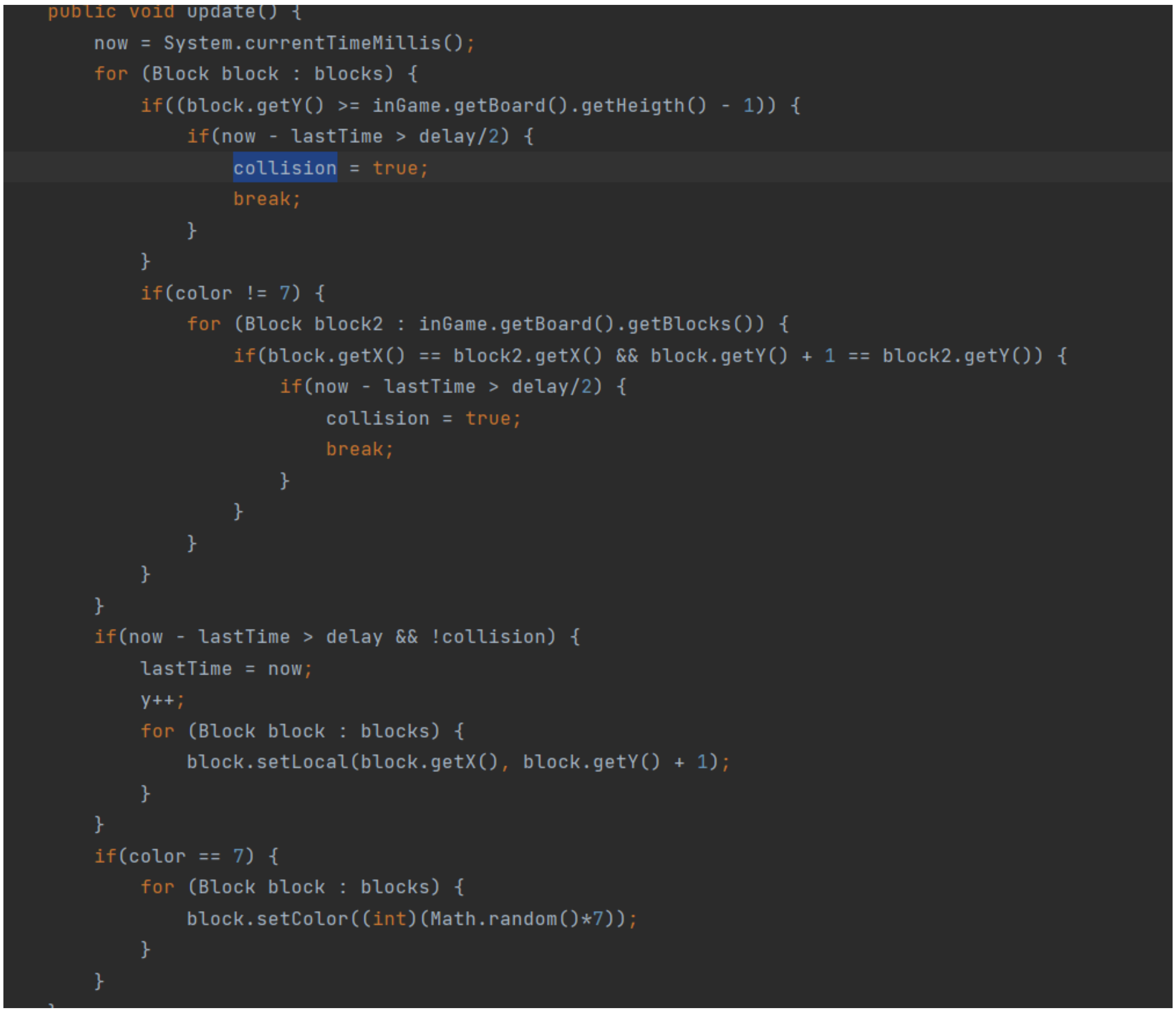
```
private int score;

public InGame(GameScr gameScr, int level) {

    this.gameScr = gameScr;
    this.level = level;

    //TODO
    count = 0;
    score = 0;
    board = new Board();

    setNextShape();
    setCurrentShape();
}

public void update() {
    //TODO
    currentShape.update();
    if(currentShape.isCollision()) {
        board.setShapeToBoard(currentShape);
        score += board.checkLine() * (8 + level * 2);
        checkGameOver();
        if(!gameOver) setCurrentShape();
    }
    keyUpdate();
}
```

j) Create a new Shape and handle collision

After we have manipulated a Shape. Next we will create a new Shape after the old Shape collides with the bottom or collides with another Shape. Each Shape is generated using the random function and is generated right in the middle of of the top row.



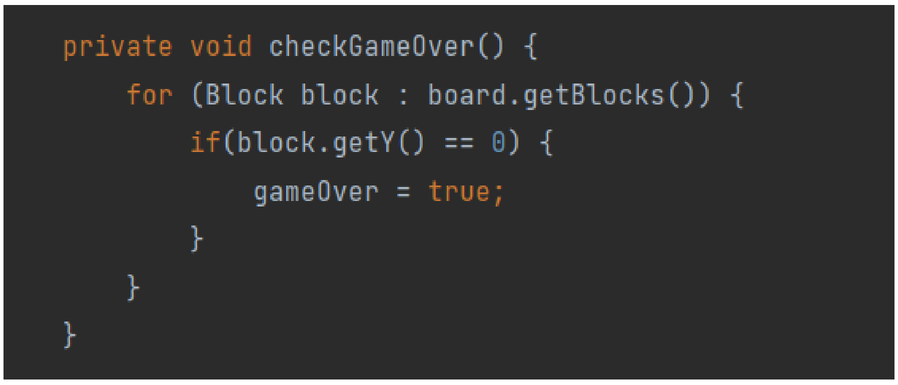
```
public void setNextShape(){
    int index = (int)(Math.random()*shapes.length);

    Shape newShape = new Shape(shapes[index].getBlock(), shapes[index].getCoords(),
    currentShape = newShape;

    for (int row = 0; row < currentShape.getCoords().length; row++) {
        for (int col = 0; col < currentShape.getCoords()[row].length; col++) {
            if(currentShape.getCoords()[row][col] != 0){
                if(board[row][col + 3] != 0)
                    gameOver = true;
            }
        }
    }
}
```



```
public void update() {
    now = System.currentTimeMillis();
    for (Block block : blocks) {
        if((block.getY() >= inGame.getBoard().getHeigth() - 1)) {
            if(now - lastTime > delay/2) {
                collision = true;
                break;
            }
        }
        if(color != 7) {
            for (Block block2 : inGame.getBoard().getBlocks()) {
                if(block.getX() == block2.getX() && block.getY() + 1 == block2.getY()) {
                    if(now - lastTime > delay/2) {
                        collision = true;
                        break;
                    }
                }
            }
        }
    }
    if(now - lastTime > delay && !collision) {
        lastTime = now;
        y++;
        for (Block block : blocks) {
            block.setLocal(block.getX(), block.getY() + 1);
        }
    }
    if(color == 7) {
        for (Block block : blocks) {
            block.setColor((int)(Math.random()*7));
        }
    }
```

k) Game over

The check Over will show down here to check the player will they continue to play or not:



```
private void checkGameOver() {
    for (Block block : board.getBlocks()) {
        if(block.getY() == 0) {
            gameOver = true;
        }
    }
}
```

The Blocks will be alive when getY from range 1 to 20 which is the height of the field. So when the block has nowhere to fall , that means it is out of the range and the game will stop immediately.

III. Restriction and Development

1. Restriction:

- Have not to implement Save Game, Edit Game function for players
- When on a single machine, the game can only be played by one person, the function of playing two people has not been implemented on the computer: one plays the keyboard, and the other plays the mouse.
- The player only can use arrow-down to pull the shape faster but not immediately

2. Development:

- Create hard-drop to let the shape fall into the field immediately.
- Create a store to save the block that is not useful at that time and re-use it in the suitable situation.
- People can share a keyboard to play together. Moreover, they may play online in the future.

IV. Contribution

Name	ID	Email	Personal Email	Participation	%

Nguyễn Bá Phúc	ITITIU20278	ITITIU20278@student.hcmiu.edu.vn		● Conduct Algorithm ● Test Game	25
Tạ Vĩ Khang	ITITIU20226	ITITIU20226@student.hcmiu.edu.vn	Khang9a22016@gmail.com	● Conduct Engine ● Create loop game ● Manage team member ● Manage github ● Implement UML	25
Lê Hoàng Thái Tuấn	ITITIU20340	ITITIU20340@student.hcmiu.edu.vn		● Design gameplay, assets ● Support other parts	
Huỳnh Thanh Thảo	ITDSIU20126	ITDSIU20126@student.hcmiu.edu.vn		● Main report writer ● Conduct GUI	