INTERNATIONAL UNIVERSITY

VIETNAM NATIONAL UNIVERSITY – HO CHI MINH CITY

School of Computer Science and Engineering

-----*-----**

PROJECT REPORT

Tetris

OBJECT-ORIENTED PROGRAMMING (IT069IU)

Semester 1 - Academic year 2024-2025

Course by: **Dr. Le Duy Tan**

MSc. Nguyen Trung Nghia

Table of Contents

CONTRIBUTION TABLE 4

ABSTRACT 5

Chapter 1: INTRODUCTION 6

A. Objectives 6

B. The tools used 6

Figure 1: Zalo platform **6**

Figure 2: Google Drive **6**

Chapter 2: GAME DESCRIPTION 7

A. Design 7

Figure 3: The designed items **8**

Figure 4: The Game Interface **8**

Figure 5: The Status Panel **9**

Figure 6: The Data Frame **9**

Figure 7: The Over Panel **9**

Figure 8: The Game Pause Interface **9**

Figure 9: Game Panel **10**

Figure 10: Game Panel **10**

Figure 11: Mino before rotation **10**

Figure 12: Mino after rotation **11**

B. Game Rule 11

Figure 13: Minor bar **12**

Figure 14: Minor bar **12**

Figure 15: After match, earn a bonus and sum to the total score **13**

Figure 20: After building a losing tower **14**

C. How the Core Game Works 14

D. UML Diagram 19

Figure 21: Body_Scence package **19**

Figure 22: Controls package **19**

2

Figure 23: Final_Scence package **20**

Figure 24: Guide package **20**

Figure 25: Setting package **21**

Figure 26: Start_Scence package **21**

Chapter 3: CONCLUSION..... 22

A. Summary..... **22**

B. Shortcoming..... **22**

C. Future Works **22**

Chapter 4: REFERENCE 23

3

CONTRIBUTION TABLE

No.	Full Name	Student's ID	Task	Contribution
1	Trần Lê Minh Quân	ITITWE23042	GUI, Plan Manager, Interface Design, Debugger	100%
2	Trịnh Minh Khoa	ITITIU21228	PowerPoint Slide Manager, Debugger	100%
3	Nguyễn Phước Thịnh	ITITWE22106	GUI(Guide), Guide, Design, UML Diagram	100%

4

ABSTRACT

Tetris is a **puzzle video game** created in **1985** by **Alexey Pajitnov, a Soviet software engineer**. It has been published by several companies on **more than 65 platform**, setting a *Guinness* world record for the most ported game.

The game's success can be attributed to its simple yet engaging puzzle which is thought easy in gameplay but needed a perfectly arrangement skill , where players rotate 7 kinds of colorful bricks to create a complete line and clear levels. The progression system, featuring a diverse range of levels with increasing difficulty, keeps players invested and motivated. In addition, the more player play the speed of these “falling bricks “ keep increasing . Moreover, finding a strategy to put the brick which unexpectedly into the right place making the gaming experience dynamic and enjoyable.

Furthermore, Tetris employs various psychological techniques to enhance player retention and satisfaction. The use of vibrant colors, cheerful graphics, and satisfying sound effects create a visually and auditorily pleasing experience. The introduction of challenges, in- game events keeps players motivated and invested over the long term. The immediate objective of the project is to give a brief introduction to Tetris, to redesign the game based on fundamentals of design patterns and to apply game rules.

Encapsulation, inheritance, and polymorphism are a few of the concepts related to object-oriented programming (OOP) that are covered in this project. The use of OOP has grown in the software real world due to the future expansion of the software business and the advancement of software engineering. Thus, the primary goal of this project is to become familiar with their enormous benefits in coding management, learn how to apply them to some related Tetris features, and notice the proper outcome.

5

Chapter 1: INTRODUCTION

A. Objectives

- This project aims to assist members revising and applying knowledge to form a Game completely.
- From this game, students understand and manipulate three fundamental concepts of Object-Oriented Programming (OOP).
- Developing more new features to make the Game more Interesting for Players.

B. The tools used

- Integrated Development Environments: Eclipse and VS Code
- Code version management: GitHub
- Project management: GitHub - Tetris.
- Storage material: Google Drive (**Figure 2**)
- Communication & Weekly Meetings: Zalo (**Figure 1**) , Directly.

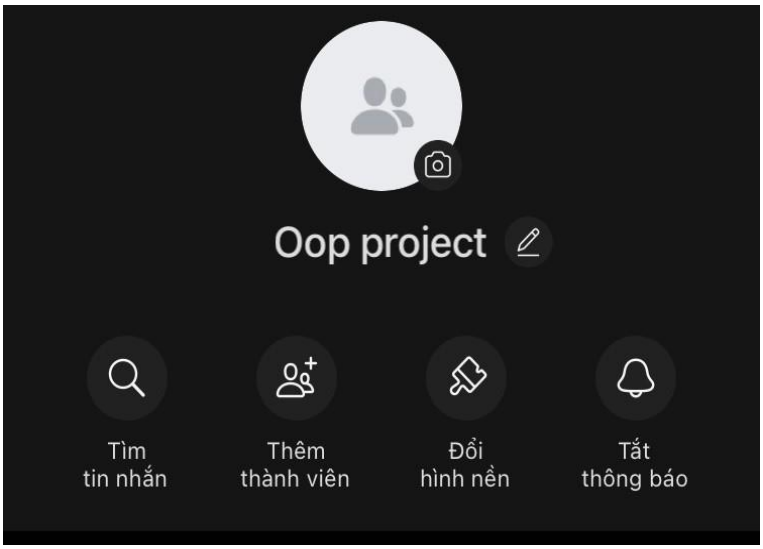


Figure 1: Zalo platform

Figure 2: Google Drive

Chapter 2: GAME DESCRIPTION

A. Design

- Game is created with an interface and seven other game elements entirely on our own. As seen in **Figure 3** below, the pieces are fixed designs that are connected to Mino class:

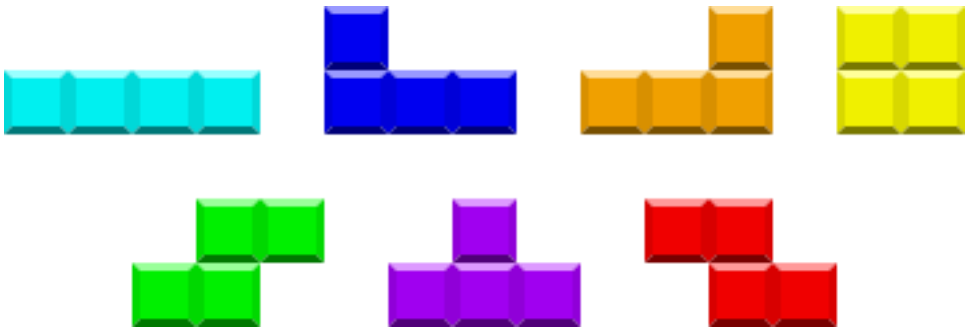


Figure 3: The designed items

a. **Mino_Bar**
b. **Mino_L1**
c. **Mino_L2**
d. **Mino_Square**
e. Mino_Z1
f. Mino_T
g. Mino_Z2

- The game has 1 primary interfaces, as seen in **Figures 4** below:

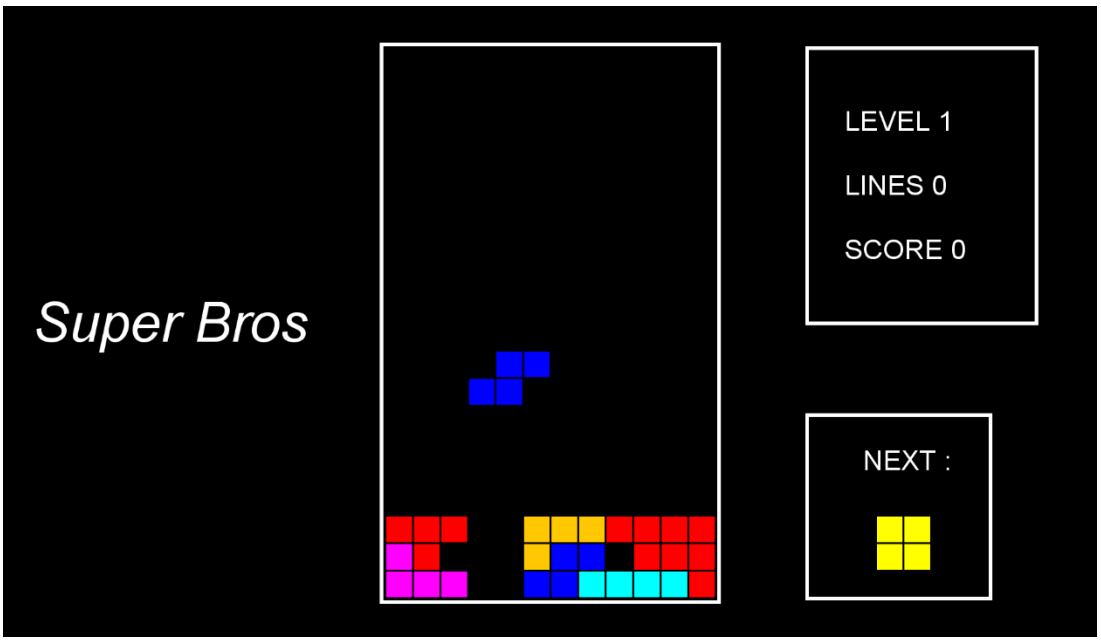


Figure 4: The Game Interface

- Tetris can be interacted by clicking the buttons which are customized om the keyboard, the Score Board (**figure 5**) located on the upper right side of the Game Interface (**Figure 4).** The Next Brick Panel (Figure 6) which is located on the down right side includes Data Frame (**Figure 6**) . Players can view the time, goal, the increased point total, and the score via Data Frame at the top right of the Score Board. Restart and Quit are the two buttons at the bottom of the Status Panel that allow you to restart or end the game. When the brick cannot go down anymore the game will be over , then the Game over Panel(**figure 7**) will pop up and stop the game , you can take a break in the middle of the game by pressed the key to pop up the Pause Panel(**figure 8)**

Figure 5: The Status Panel

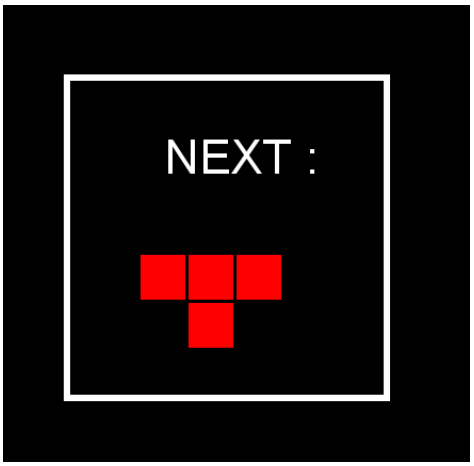


Figure 6: The Data Frame(Next Brick Panel)

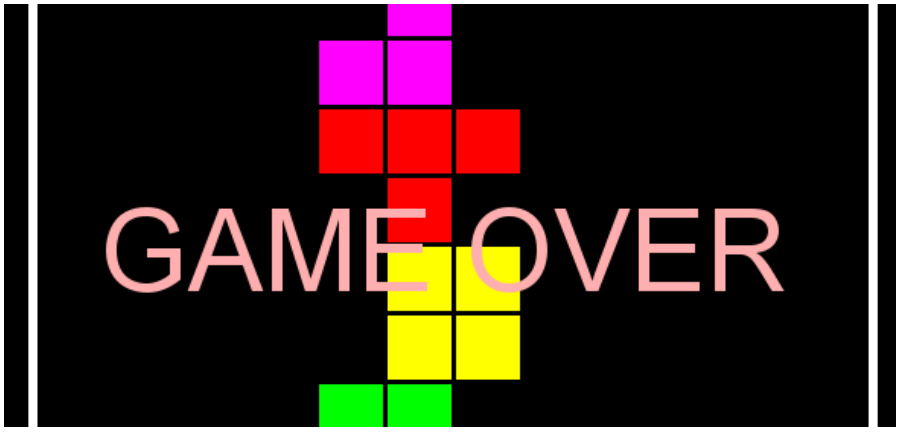


Figure 7: Game Over Panel

Figure 8: Game Pause Interface

B. Game Rule

- Here is all of the basic guide and rule of the game “Tetris”:
  - The rules for the foundational game are straightforward, emphasizing the arranging, putting 7 difference kind the brick in the right place to make the brick do not stack over .
  - The game starts with a plain panel in the middle and awaiting bricks start to fall.

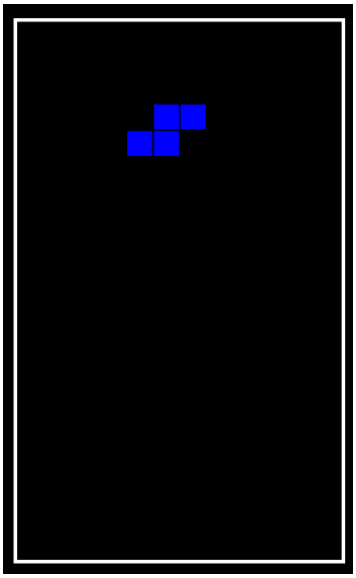


Figure 9: GamePanel
(at the beginning of the game)

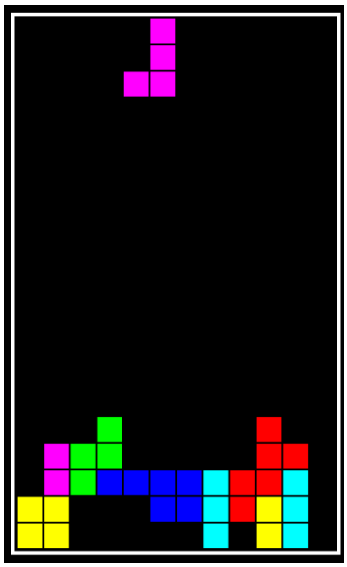


Figure 10: GamePanel
(after being filled)

- We also apply a code for it to slowly slide down the Mino for player can rotate the Mino or decide where it will be dropped.

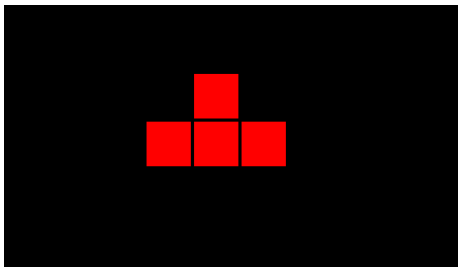


Figure 11: Mino before rotation

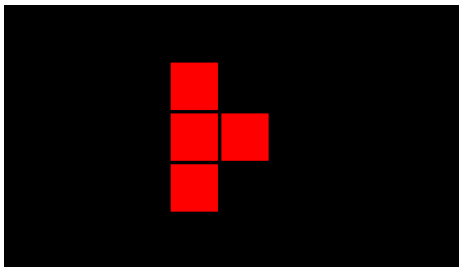
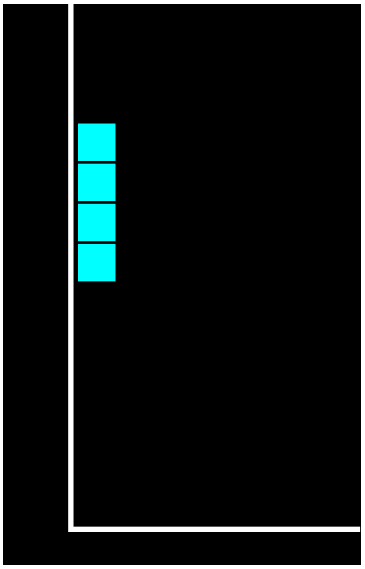


Figure 12: Mino after rotation

- To initiate the side of rotation for the block, we already have applied the keyboard input for the rotation also for the movement of the block.
    - Move to the left by Pressing A, the same for moving to the right by D and it will drop more immediately by Pressing S.
    - And if Rotation will follow the counterclockwise way to rotate by Pressing W.

- The movement and Rotation of the Mino will be stop when the collision happen , such as it will stop moving when you hit the floor(figure 16) or ability to rotate will also be disable when a part of a brick get over or get out of the “wall “ of the game(**figure 17**) .



- **Figure 13: Mino_Bar** cannot rotate when collide the wall

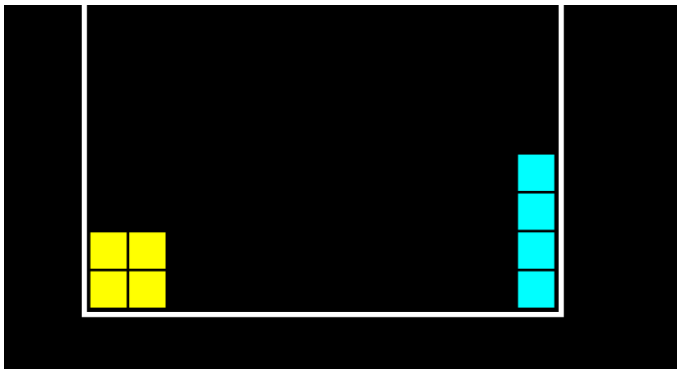


Figure 14: Mino_Bar cannot move

- And one more trick should be mention here is that when you hit the floor of the game , you wont just stop there , you also have a chance around 1 second to move your block to the place you want .
- Upon the line within fully 12 Blocks horizontal get in , it will be deleted to give more space for the next Mino show ups , and bring you back the score .

- After the perfect 12 Block in line disappear , you will receive for each 1 + line and the score will be summed for 10 points.

- When you reach the score of 100 , the level will be up from 1 to 2 and so on , then the Game starts getting harder by at this time the speed is no more slow it will get more faster , the more you play , the faster the Mino fall.

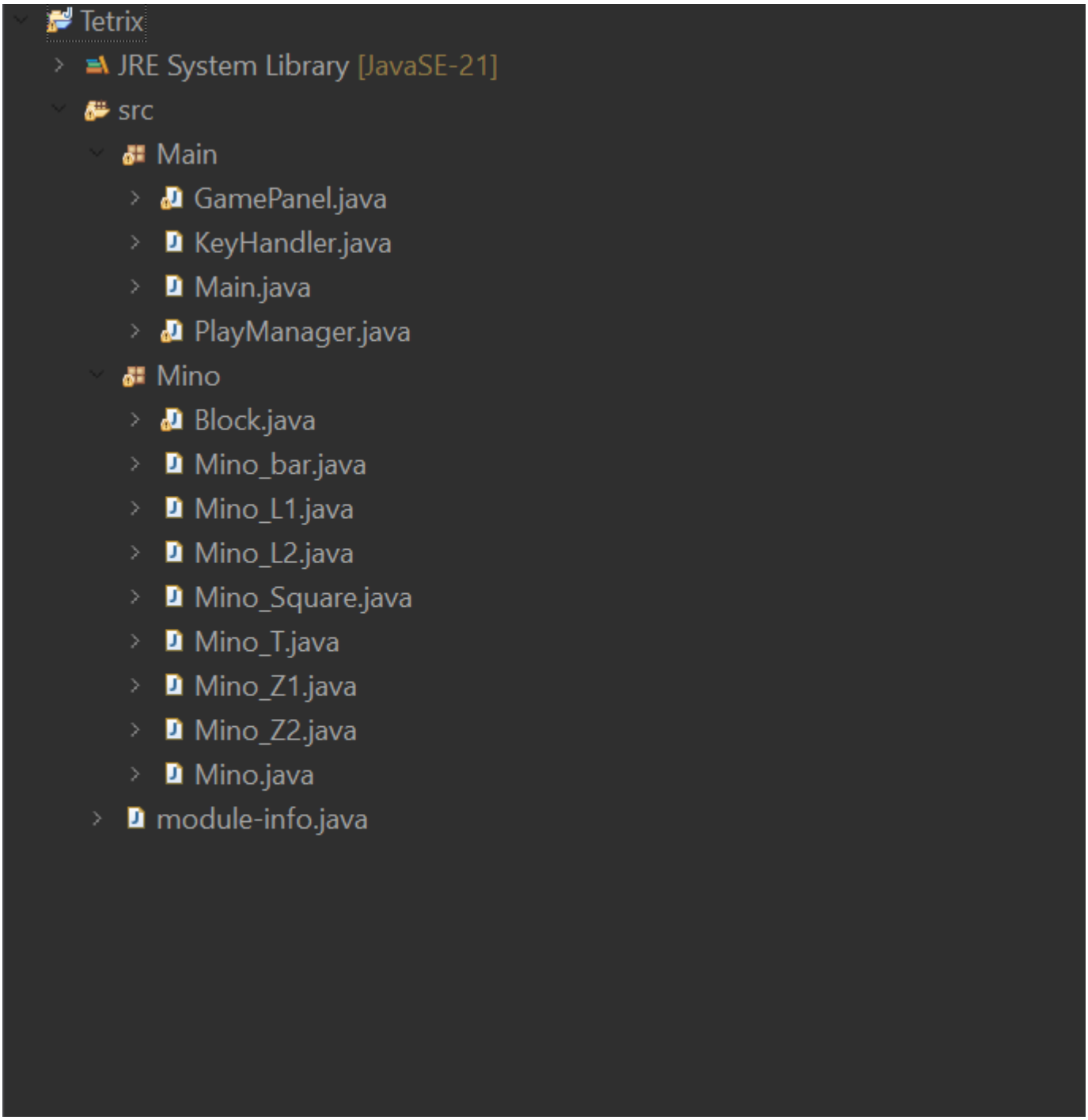
Figure 15: After match, earn a bonus and sum to the total score

- Next, the game is being put in a loop that you do not win the game , you just keep playing until you cannot keep pace with the game speed and then that is the time for you to stop and replay the game , cause the “Game Over ” title will pop up right away when the space is no more to put more Mino.

Figure 20: After building a “losing tower”

- In case that you almost lose , do not worry just play it again next time for a better luck.
- Here is some tip and trick:
  - Thinking about the impact of each move on the entire board and your level objective
  - This is all the instructions of the Tetris for new comer .

C. How the Core Game Works



- To illustrate how the core game work that we will have to tell about 2 class the GamePanel and PlayManager which help a lot to make the game is runnable :

  1/ **GamePanel** class:

that is built easy to understand , it is extended of JPanel and implement Runnable, nable game loop functionality. This class manages the graphical user interface (GUI), game loop, and interactions with

```
1  package Main;
2
3  import java.awt.Color;
9
10 public class GamePanel extends JPanel implements Runnable {
11
12         public static final int WIDTH =1200;
13         public static final int HEIGHT= 720;
14         final int FPS=60;
15         Thread gameThread;
16         PlayManager pm;
17
18
19     public GamePanel () {
33 public void launchGame() {
34     gameThread=new Thread(this);
35     gameThread.start();
36 }
37
38
40     public void run() {
63         public void update() {
68
69         public void paintComponent(Graphics g) {
76
77     }
78
79
```

other game components.

+ All the attributes like :WIDTH and HEIGHT: These constants define the dimensions of the game panel ,FPS: This constant sets the target frame rate to 60 frames per secondand Thread It ensures the game runs independently of the main program thread.

- With the constructor GamePanel ()
  - Setting the preferred size to match the defined WIDTH and HEIGHT.
  - Configuring the background color to black to provide a clean contrast for the game elements.
  - Adding a KeyListener for user input via the KeyHandler class,and setFocusable to capture keyboard input effectively..

Methods

launchGame() : This method initializes and starts the game loop by creating a new thread (gameThread) and starting it.

run() : Implements the game loop using the following steps:

- **Timing Management**: Calculates the interval between frames to maintain a steady FPS.
- **Update and Redraw**: Calls the update() method for game logic and repaint() to redraw the screen when required.

update() : Updates the game state by:

- Checking if the game is not paused
- Delegating the update logic to the PlayManager instance.

paintComponent(Graphics g)

- Overrides the paintComponent() method from JPanel:
- Draws game elements on the panel using Graphics2D, leveraging the PlayManager's draw() method.

Responsibilities of GamePanel The GamePanel class is responsible for:

Managing the Game Loop, Handling User Input, Rendering Game Element and Providing a Container.

2/ **PlayManager class** :

The PlayManager class serves as the primary handler for gameplay mechanics, including managing the play area, controlling Tetris pieces (Mino), and updating the game state. It is responsible for drawing the game area, managing static blocks, handling gameplay actions, and providing visual effects and scoring.

WIDTH and HEIGHT: Define the size of the play area.

left_x, right_x, top_y, bottom_y: Specify the boundaries of the play area based on the screen dimensions.

Purpose: These variables establish the boundaries where Tetris gameplay occurs.

Tetris Pieces (Mino) Management

Current and Next Mino:

- currentMino: The active Tetris piece in play.
- nextMino: The upcoming Tetris piece displayed to the player.
- MINO_START_X, MINO_START_Y: The starting position for currentMino.
- NEXTMINO_X, NEXTMINO_Y: The position to display nextMino.

Methods:

- pickMino(): Randomly selects one of the predefined Mino types (Mino_L1, Mino_Z1, etc.) for the next piece.
- update(): Updates the position of the currentMino and handles its transition to a static block if it becomes inactive.

Static Blocks Management

- **staticBlocks:** An ArrayList storing blocks that are no longer active.
- **Purpose:** Tracks blocks that have settled at the bottom of the play area or stacked on other blocks.

Methods:

- checkdelete(): Checks for completed lines, removes them, and shifts blocks downward to maintain the gameplay area.

Scoring and Level Progression

Variables:

- score, level, lines: Track the player's current score, level, and the number of completed lines.
- dropInterval: Determines the speed of the falling Tetris pieces, which decreases as the level increases.

Scoring Mechanism:

- Points are awarded based on the number of lines cleared and the current level.
- The game becomes faster as more lines are cleared.

Variables:

- gameOver: Indicates whether the game has ended.
- eco and effectY: Manage visual effects when a line is cleared.

Game Over Check:

- The game ends if a new currentMino cannot be placed at the starting position due to existing static blocks.

draw(Graphics2D g2):

- Draws the main play area, the currentMino, nextMino, and static blocks.
- Displays score, level, and game over/pause messages.
- Implements visual effects for cleared lines.

3/ The Block class: extends Rectangle, represents an individual square unit that makes up a Tetris piece (Mino) or forms part of the static blocks on the playfield. It is a fundamental building block of the game.

```java
package Mino;

import java.awt.Color;

public class Block extends Rectangle  {


    public int x,y;
    public static final int SIZE =30;//30x30 block
    public Color c ;

    public Block(Color c) {
        this.c=c;
    }
    public void draw (Graphics2D g2 ) {
        int margine =1;
        g2.setColor(c);
        g2.fillRect(x+margine, y+margine, SIZE-(2*margine), SIZE-(2*margine));
    }
    }
```

x, y: The coordinates of the block on the game grid.

SIZE: A constant defining the size of each block as 30x30 pixels.

c: The color of the block, which visually distinguishes different types of Mino.

- **Constructor**
    - Block(Color c): Initializes a block with a specific color. The color corresponds to the type of Mino it belongs to.
- **Methods**
    - draw(Graphics2D g2): Renders the block on the screen.
    - A small margin is applied to give each block a border effect, enhancing the visual appearance.
- **Interaction with Other Classes**
    - **Mino:** Each Mino is composed of multiple Block objects arranged in specific shapes.
    - **PlayManager:** Manages the position and rendering of Block objects, either as part of active Mino pieces or static blocks on the playfield.
    - **Conclusion**The Block class is a foundational component of the Tetris game, providing the basic unit for constructing Tetris pieces and maintaining the static block structure on the playfield. Its simplicity ensures efficient rendering and interaction within the game.

4/Mino class: The Mino class represents a Tetris piece in the game, consisting of four blocks (Block objects) that form specific shapes. It handles the piece's movement, rotation, collision detection, and deactivation.

```java
package Mino;

import java.awt.Color;

public class Mino {


    public Block b[]=new Block[4];
    public Block tempB[]= new Block[4];
    int autodrop =0;
    public int direction =1 ;
    boolean leftCollision, rightCollision , bottomCollision;
    public boolean active =true ;
    public boolean deactivating;
    int deactivatingCounter =0;



    public void create(Color c ) {
    public void setXY(int x , int y ) {}
    public void updateXY(int direction) {

    public void getDirection1() {}
    public void getDirection2() {}
    public void getDirection3() {}
    public void getDirection4() {}
    public void checkMovementCollision() {
    public void checkRotationCollision() {

    public void checkStaticBlockCollision() {
    public void update(){
    private void deactivating() {
    public void draw(Graphics g2) {


        int margin=1;
        g2.setColor(b[0].c);
        g2.fillRect(b[0].x+ margin, b[0].y+margin, Block.SIZE-(margin*2), Block.SIZE-(margin*2));
        g2.fillRect(b[1].x+ margin, b[1].y+margin, Block.SIZE-(margin*2), Block.SIZE-(margin*2));
        g2.fillRect(b[2].x+ margin, b[2].y+margin, Block.SIZE-(margin*2), Block.SIZE-(margin*2));
        g2.fillRect(b[3].x+ margin, b[3].y+margin, Block.SIZE-(margin*2), Block.SIZE-(margin*2));
```

- **Blocks:**
  - b[]: The primary array of four blocks representing the active Mino.
  - tempB[]: A temporary array used during rotation to test for collisions.
- **Movement and State:**
  - autodrop: Counter for automatic downward movement based on the drop interval.
  - direction: Indicates the current orientation of the Mino (1 to 4).
  - active: Specifies whether the Mino is currently in play.
  - deactivating: Tracks whether the Mino is in the process of being deactivated.
  - deactivatingCounter: Counts frames to determine when to deactivate.
- **Collision Flags:**
  - leftCollision, rightCollision, bottomCollision: Flags indicating if the Mino is colliding with walls, floor, or other blocks.

Methods

- **Creation and Initialization:**
  - create(Color c): Initializes the Mino with four blocks of a specified color.
  - setXY(int x, int y): Sets the initial position of the Mino.
- **Movement and Rotation:**
  - updateXY(int direction): Updates the Mino's position during rotation, checking for collisions before applying changes.
  - getDirection1(), getDirection2(), getDirection3(), getDirection4(): Define the four possible orientations of the Mino.
- **Collision Detection:**
  - checkMovementCollision(): Checks for collisions with walls, the floor, or static blocks when the Mino moves.
  - checkRotationCollision(): Verifies if the Mino can rotate without colliding.
  - checkStaticBlockCollision(): Checks for collisions with static blocks on the playfield.
- **Game Logic:**
  - update(): Handles the Mino's movement (left, right, down), rotation, and automatic drop based on user input and collision states.
  - deactivating(): Manages the deactivation process, ensuring the Mino becomes inactive after hitting the bottom or static blocks for a specified duration.
- **Rendering:**
  - draw(Graphics g2): Draws the Mino on the screen by rendering its blocks with the appropriate color and position.

5/ All of the 7 Mino:

Which were built the same and it need to take 1 block out to be the static block and all other 3 Blocks around it will be made base on the data of that static Block.

Take MinoL1 class for instance :

O

O

O O

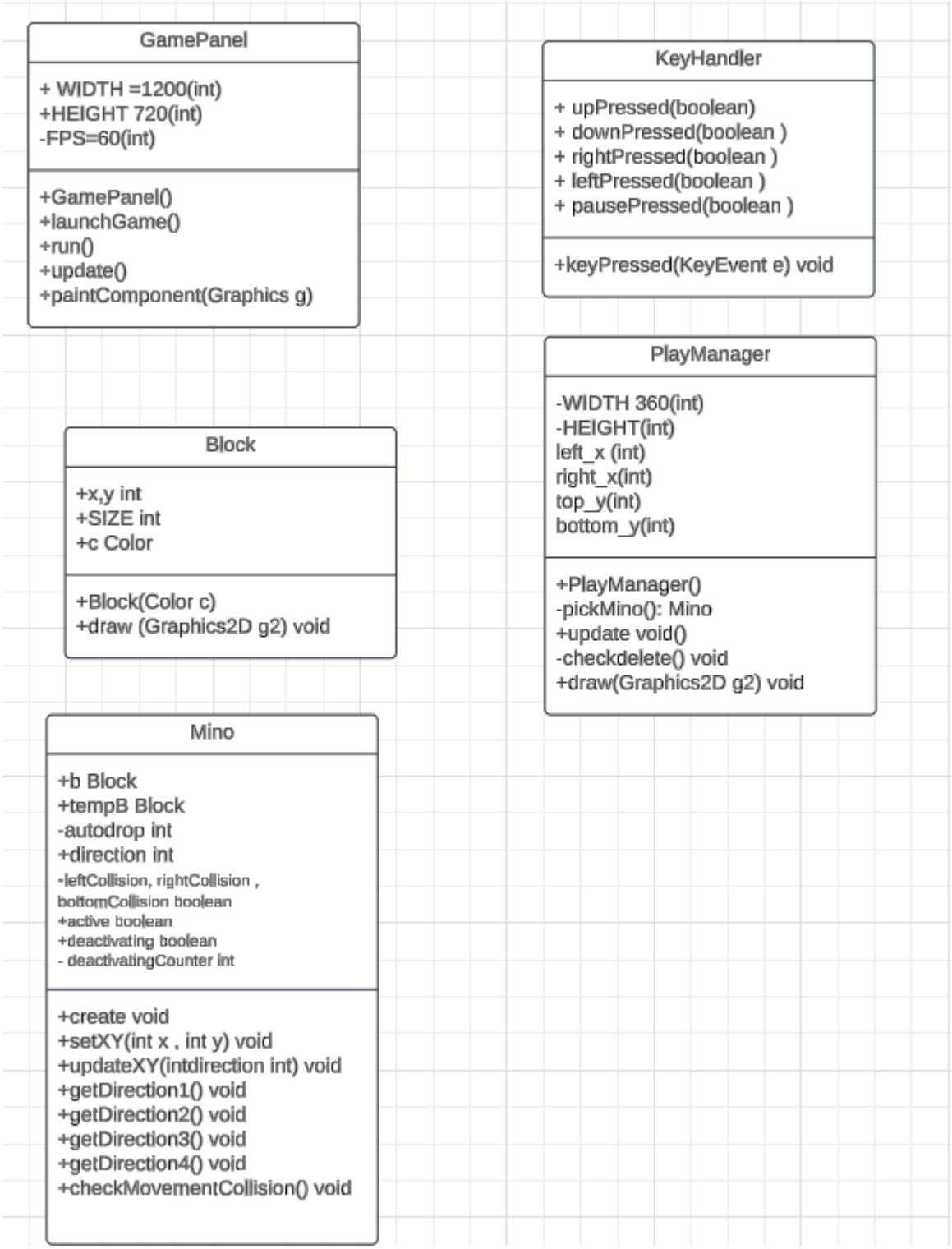
This is the original L1 Mino will drop when the game start , and I make the O to be the static Block and all other 3 are made base on its

```
public void getDirection1() {

    //o 1
    //o 0
    //o o 2,3
    tempB[0].x= b[0].x;
    tempB[0].y= b[0].y;
    tempB[1].x= b[0].x;
    tempB[1].y= b[0].y - Block.SIZE;
    tempB[2].x= b[0].x;
    tempB[2].y= b[0].y + Block.SIZE;
    tempB[3].x= b[0].x + Block.SIZE;
    tempB[3].y= b[0].y + Block.SIZE;


        updateXY(1);
}
public void getDirection2() {
    //
    // 0 0 0
    // 0

            tempB[0].x=b[0].x;
            tempB[0].y=b[0].y;
            tempB[1].x= b[0].x+ Block.SIZE;
            tempB[1].y= b[0].y ;
            tempB[2].x= b[0].x- Block.SIZE;
            tempB[2].y= b[0].y ;
            tempB[3].x= b[0].x - Block.SIZE;
            tempB[3].y= b[0].y + Block.SIZE;

            updateXY(2);
```

And when it rotate , that one static Block still be stable whenever direction it is.

D. UML Diagram

- These three classes are responsible for showing and playing the Game Area and Counting Score Area.


GamePanel
+ WIDTH =1200(int)
+HEIGHT 720(int)
-FPS=60(int)
+GamePanel()
+launchGame()
+run()
+update()
+paintComponent(Graphics g)
KeyHandler
+ upPressed(boolean)
+ downPressed(boolean)
+ rightPressed(boolean)
+ leftPressed(boolean)
+ pausePressed(boolean)
+keyPressed(KeyEvent e) void
PlayManager
-WIDTH 360(int)
-HEIGHT(int)
left_x (int)
right_x(int)
top_y(int)
bottom_y(int)
+PlayManager()
-pickMino(): Mino
+update void()
-checkdelete() void
+draw(Graphics2D g2) void
Block
+x,y int
+SIZE int
+c Color
+Block(Color c)
+draw (Graphics2D g2) void
Mino
+b Block
+tempB Block
-autodrop int
+direction int
-leftCollision, rightCollision ,
bottomCollision boolean
+active boolean
+deactivating boolean
- deactivatingCounter int
+create void
+setXY(int x , int y) void
+updateXY(intdirection int) void
+getDirection1() void
+getDirection2() void
+getDirection3() void
+getDirection4() void
+checkMovementCollision() void

Chapter 3: CONCLUSION

A. Summary

In conclusion, the project has been built on OOP features. Through the process of implementing this game project, we have strengthened and practiced coding skills while improving other skills such as teamwork skills and planning for a project. However, the game project has not yet been completed and needs to be supplemented in the future.

B. Shortcoming

Because of the limited time, the game has not been completed yet. When the game is launched, only the ‘lose scene’ appears, while the ‘win scene’ has a problem not being displayed and must be changed by restarting the countdown timer as well as the game board. In addition, the game still lacks some features, such as saving game progress (players must start from the beginning), the ability have the sound ,and finally, the code still does not strictly follow the SOLID principles, causing difficulties in developing.

C. Future Works

In the future, team will improve upon these weaknesses by incorporating the following features:

- Replay : which help the player can replay the game when the game over.
- SOLID code: The code will be modified to adhere to the SOLID principles more closely in order to make it easier to extend and maintain.
- Setting: Active two switches which allow players to turn on/off Background Sound and Matched Sound in the game.
- Music & Sound Effect : it need the to be added in to make the game more fascinating.

Chapter 4: REFERENCE

1) FaTal Cubez. (2014, September 9). *2048 Tutorial Series*. YouTube. https://www.youtube.com/watch?v=GlXWdQSfddw&list=PLig6-gM-fHMGH6jmCpsxW6YbagHgCS3Jd&index=1
2) Freepik. (n.d.). Flat Tet Instagram Posts Collection. Retrieved from https://www.freepik.com/free-vector/flat-tet-instagram-posts-collection_21530587.htm?fbclid=IwAR0slxlDFsed8UTER4FeVvhrC81_xiHaiZZh6AGYxiciL076ckb-XN2zOzc#query=t%E1%BA%BFt%20vi%E1%BB%87t%20collection&position=27&from_view=search&track=ais&uuid=5a61856f-9304-456c-96a0-55ae7107b2d5
3) RyiSnow. (2021, April 14). [Java Code Sample] Create timer (normal/countdown/two digits). https://www.ryisnow.online/2021/04/java-beginner-code-sample-create-timer.html
4) freepik. (n.d.). *Free vector: Flat tet instagram posts collection*. Freepik. https://www.freepik.com/free-vector/flat-tet-instagram-posts-collection_21530587.htm?fbclid=IwAR260N5-DTMebfjNORBDb9l2AfcQ3XofEMkP64pBlacvxTdsOPzjf-6LpNg#query=t%E1%BA%BFt%20vi%E1%BB%87t%20collection&position=27&from_view=search&track=ais&uuid=5a61856f-9304-456c-96a0-55ae7107b2d5
5) King. (2018, June 19). Video Game Music. https://downloads.khinsider.com/game-soundtracks/album/candy-crush-saga-gamerip