TRƯỜNG ĐẠI HỌC GIAO THÔNG VÂN TẢI

KHOA CÔNG NGHỆ THÔNG TIN

---------------o0o---------------



**BÁO CÁO BÀI TẬP LỚN   
MÔN CÔNG NGHÊ JAVA**

**Đề Tài: Game 2048**

Giảng viên hướng dẫn: Vũ Huấn

|  |  |  |
| --- | --- | --- |
| Sinh viên thực hiện: | Nguyễn Văn Đức | 211211969 |
| Đỗ Xuân Hải | 211214265 |

**Hà Nội, tháng 4 năm 2023**

**LỜI NÓI ĐẦU**

Trò chơi 2048 là một trò chơi điện tử thú vị và gây nghiện. Được tạo ra bởi Gabriele Cirulli vào năm 2014, trò chơi này đã nhanh chóng trở thành một hiện tượng trong cộng đồng game thủ. 2048 kết hợp các yếu tố logic và chiến thuật để thách thức người chơi trong việc sắp xếp và kết hợp các ô số để đạt được con số 2048. Mục tiêu của trò chơi 2048 là kết hợp các ô số trong một ô vuông lưới 4x4 để tạo ra số 2048.

Trò chơi 2048 đòi hỏi người chơi phải có sự tư duy chiến thuật và logic để đạt được kết quả cao. Người chơi phải xem xét các bước di chuyển để không làm kẹt các ô số và cố gắng tạo ra các ô số lớn hơn thông qua việc kết hợp các ô có giá trị tương đồng. Đồng thời, người chơi cũng phải đối mặt với thách thức của không gian hạn chế trong lưới 4x4, khiến việc di chuyển và kết hợp trở nên phức tạp hơn.

Trò chơi 2048 đã gây tiếng vang lớn trong cộng đồng game thủ và trở thành một trò chơi rất phổ biến. Điểm mạnh của nó nằm ở tính gây nghiện và thú vị. Người chơi luôn muốn cố gắng vượt qua thành tích của mình và đạt điểm số cao hơn. Trong quá trình chơi, người chơi phải suy nghĩ và đưa ra quyết định để tạo ra các ô số lớn hơn và tiến xa hơn trong trò chơi. Điều này không chỉ giúp cải thiện khả năng tư duy và quyết định của người chơi, mà còn mang lại sự thú vị và niềm vui trong quá trình giải quyết các thách thức.

Tóm lại, trò chơi 2048 là một trò chơi đơn giản nhưng gây nghiện, kết hợp giữa yếu tố logic và chiến thuật.Với sự phát triển và ảnh hưởng của nó, 2048 đã trở thành một trong những trò chơi logic phổ biến và mang lại sự thú vị và thử thách cho người chơi

**Mục Lục**

[**I.** **Mở đầu** 4](#_Toc134039244)

[**1.** **Giới thiệu trò chơi** 4](#_Toc134039245)

[**2.** **Luật chơi** 4](#_Toc134039246)

[**II.** **Tổng quan về chương trình** 5](#_Toc134039247)

[**1.** **Thiết kế chương trình** 5](#_Toc134039248)

[**2.** **Các package và class** 6](#_Toc134039249)

[**III.** **Cấu trúc chương trình** 7](#_Toc134039250)

[**1.** **Package com.now2048.game** 7](#_Toc134039251)

[**2.** **Package com.now2048.gui** 29](#_Toc134039252)

[**IV.** **Hướng dẫn cài đặt người dùng** 38](#_Toc134039253)

[**V.** **Tài liệu tham khảo** 41](#_Toc134039254)

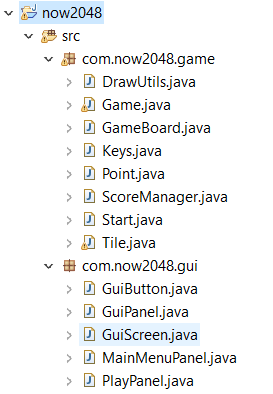
[**VI.** **Tổng kết** 41](#_Toc134039255)

1. **Mở đầu**
2. **Giới thiệu trò chơi**

* 2048 là một trò chơi giải đố, mục tiêu của trò chơi là trượt các khối vuông có mang số trên một lưới vuông để kết hợp chúng lại và tạo ra khối vuông có giá trị 2048.

1. **Luật chơi**

* Luật chơi: 2048 chơi trên một lưới vuông 4×4. Mỗi lần di chuyển là một lượt, người chơi sử dụng các phím mũi tên và các khối vuông sẽ trượt theo một trong bốn hướng tương ứng (lên, xuống, trái, phải). Mỗi lượt có một khối có giá trị 2 hoặc 4 sẽ xuất hiện ngẫu nhiên ở một ô trống trên lưới. Các khối vuông trượt theo hướng chỉ định cho đến khi chạm đến biên của lưới hoặc chạm vào khối vuông khác. Nếu hai khối vuông có cùng giá trị chạm vào nhau, chúng sẽ kết hợp lại thành một khối vuông có giá trị bằng tổng giá trị hai khối vuông đó (giá trị gấp đôi). Khối vuông kết quả không thể kết hợp với khối vuông khác một lần nữa trong một lượt di chuyển. Để dễ nhận biết thì các khối vuông giá trị khác nhau sẽ có màu sắc khác nhau.
* Bảng điểm ở góc trên bên trái cho biết điểm của người chơi. Ban đầu điểm bằng 0. Khi hai khối vuông kết hợp thì người chơi sẽ tăng điểm là giá trị khối vuông mới. Bên cạnh điểm hiện tại là kỉ lục điểm cao nhất người chơi từng đạt được.
* Khi người chơi tạo được ô vuông có giá trị 2048 thì thắng cuộc. Lúc này người chơi có thể lựa chọn tiếp tục chơi để đạt các giá trị cao hơn 2048. Khi không còn nước đi hợp lệ (không còn ô trống và các ô kề nhau đều khác giá trị) thì trò chơi kết thúc.

1. **Tổng quan về chương trình**
2. **Thiết kế chương trình**

* **Game 2048 này chia làm 2 package:**
* com.now2048.game:
* class Start
* class Game
* class GameBoard
* class Tile
* class Keys
* class Point
* class DrawUtils
* class ScoreManager
* com.now2048.gui:
* class GuiScreen
* class GuiPanel
* class GuiButton
* class MainMenuPanel
* class PlayPanel

1. **Các package và class**

* **Package com.now2048.game**
* Class Start: Sử dụng để khởi tạo và hiển thị cửa sổ trò chơi.
* Class Game: Bao gồm một luồng chạy trò chơi riêng biệt để cập nhật và vẽ liên tục. Nó sử dụng một đối tượng GuiScreen để quản lý các panel trong trò chơi, như MainMenuPanel và PlayPanel. Bên trong phương thức run(), nó thực hiện vòng lặp chính của trò chơi, trong đó cập nhật và vẽ được thực hiện với tần suất 60 lần/giây. Lớp Game cũng triển khai các phương thức xử lý sự kiện bàn phím và chuột từ người dùng. Phương thức start() và stop() được sử dụng để bắt đầu và dừng chạy trò chơi. Khi start() được gọi, một luồng mới sẽ được tạo để chạy trò chơi. Khi stop() được gọi, chương trình sẽ thoát.
* Class GameBoard: Bao gồm các chức năng khởi tạo bảng trò chơi, xử lý việc tạo các ô vuông và giá trị ban đầu. Di chuyển các ô vuông trên bảng theo các hướng (trái, phải, lên, xuống) dựa trên sự tương tác của người dùng. Kiểm tra và xử lý việc kết hợp các ô vuông có cùng giá trị. Kiểm tra trạng thái của trò chơi (thắng, thua). Vẽ bảng trò chơi và các ô vuông lên màn hình. Xử lý các sự kiện như nhấn phím để di chuyển các ô vuông, bắt đầu trò chơi mới, lưu trạng thái trò chơi, v.v. Tính điểm và quản lý điểm số của người chơi.
* Class Tile: Bao gồm các chức năng khởi tạo các ô số có các giá trị, màu sắc, vị trí x và y được chỉ định, cập nhật hình ảnh và vẽ các ô số theo vị trí x và y đó. Ngoài ra còn có các chức năng để lấy giá trị hiện tại của ô getValue(), kiểm tra về khả năng kết hợp canCombine(), và nhiều hàm getter và setter để phục vụ cho các class khác.
* Class Keys: Có nhiệm vụ theo dõi trạng thái của các phím được nhấn trên bàn phím và cung cấp các phương thức để cập nhật và xử lý các sự kiện liên quan đến phím.
* Class Point: Có nhiệm vụ lưu thông tin về vị trí xét theo hàng và cột của các ô số trên bảng trò chơi nhằm phục vụ cho chức năng của class khác.
* Class DrawUtils: Cung cấp các phương thức tiện ích để vẽ và tính toán kích thước văn bản trên đồ họa 2D.
* Class ScoreManager: Quản lí thông tin về điểm số của trò chơi. Ngoài ra còn có khả năng truy/xuất dữ liệu từ tệp tin được ghi lại phục vụ cho các chức năng lưu tiến độ trò chơi, khởi động lại trò chơi theo tiến độ đã được lưu trước đó.
* **Package com.now2048.gui:**
* Class GuiScreen: Đại diện cho một màn hình giao diện người dùng (GUI) là một lớp trung tâm quản lý và điều hướng các thành phần giao diện trên màn hình. Nó đảm bảo rằng chỉ một thành phần giao diện được hiển thị và xử lý tại một thời điểm, cho phép cập nhật, vẽ và xử lý sự kiện chuột của thành phần giao diện đó.
* Class GuiPanel: Đại diện cho một bảng điều khiển giao diện người dùng (GUI Panel) quản lý và hiển thị các nút giao diện trên đó. Nó cho phép cập nhật, vẽ và xử lý sự kiện chuột của các đối tượng GuiButton trong danh sách.
* Class GuiButton: đại diện cho một các nút giao diện người dùng (GUI button) có khả năng hiển thị, xử lý sự kiện và quản lý trạng thái của nút trên giao diện.
* Class MainMenuPanel: Đại diện cho một bảng điều khiển (panel) trong giao diện chính của trò chơi 2048 khi bắt đầu nó chứa các nút giao diện và có khả năng vẽ và cập nhật giao diện. Khi người dùng nhấp vào các nút, nó sẽ thực hiện các hành động tương ứng như chuyển đổi sang màn chơi hoặc thoát khỏi trò chơi.
* Class PlayPanel: Đại diện cho bảng điều khiển giao diện chơi của trò chơi 2048 khi kết thúc. Nó chứa các nút giao diện và có khả năng vẽ và cập nhật giao diện. Khi người dùng nhấp vào các nút, nó sẽ thực hiện các hành động tương ứng như chơi lại hay quay về main menu.

1. **Cấu trúc chương trình**
2. **Package com.now2048.game**

* **Class Start:** Tạo ra và khởi chạy một khung hình JFrame chứa trò chơi với các chức năng cơ bản và là class chạy chính chạy trò chơi.

**package** com.now2048.game;

**import** javax.swing.JFrame;

**public** **class** Start {

**public** **static** **void** main(String[] args){

Game game = **new** Game();

JFrame window = **new** JFrame("2048");

window.setDefaultCloseOperation(JFrame.***EXIT\_ON\_CLOSE***);

window.setResizable(**false**);

window.add(game);

window.pack();

window.setLocationRelativeTo(**null**);

window.setVisible(**true**);

game.start();

}

}

* **Class Game:** Quy định chiều dài, chiều rộng, font chữ, màu sắc cửa sổ trò chơi cùng các hàm quản lí luồng game và các chức năng khác.

**package** com.now2048.game;

**import** java.awt.Color;

**import** java.awt.Dimension;

**import** java.awt.Font;

**import** java.awt.Graphics2D;

**import** java.awt.event.KeyEvent;

**import** java.awt.event.KeyListener;

**import** java.awt.event.MouseEvent;

**import** java.awt.event.MouseListener;

**import** java.awt.event.MouseMotionListener;

**import** java.awt.image.BufferedImage;

**import** javax.swing.JPanel;

**import** com.now2048.gui.GuiScreen;

**import** com.now2048.gui.MainMenuPanel;

**import** com.now2048.gui.PlayPanel;

**public** **class** Game **extends** JPanel **implements** Runnable, KeyListener, MouseListener, MouseMotionListener{

**private** **static** **final** **long** ***serialVersionUID*** = 1L;

**public** **static** **final** **int** ***WIDTH*** = GameBoard.*BOARD\_WIDTH* + 40;

**public** **static** **final** **int** ***HEIGHT*** = 630;

**public** **static** **final** Font ***main*** = **new** Font("Bebas Neue Regular", Font.***PLAIN***, 28);

**private** Thread game;

**private** **boolean** running;

**private** BufferedImage image = **new** BufferedImage(***WIDTH***, ***HEIGHT***, BufferedImage.***TYPE\_INT\_RGB***);

**private** GuiScreen screen;

**public** Game() {

setFocusable(**true**);

setPreferredSize(**new** Dimension(***WIDTH***, ***HEIGHT***));

addKeyListener(**this**);

addMouseListener(**this**);

addMouseMotionListener(**this**);

screen = GuiScreen.*getInstance*();

screen.add("Menu", **new** MainMenuPanel());

screen.add("Play", **new** PlayPanel());

screen.setCurrentPanel("Menu");

}

**private** **void** update() {

screen.update();

Keys.*update*();

}

**private** **void** render() {

Graphics2D g = (Graphics2D) image.getGraphics();

g.setColor(Color.***white***);

g.fillRect(0, 0, ***WIDTH***, ***HEIGHT***);

screen.render(g);

g.dispose();

Graphics2D g2d = (Graphics2D) getGraphics();

g2d.drawImage(image, 0, 0, **null**);

g2d.dispose();

}

@Override

**public** **void** run() {

**int** fps = 0, updates = 0;

**long** fpsTimer = System.*currentTimeMillis*();

**double** nsPerUpdate = 1000000000.0 / 60;

// last update time in nanoseconds

**double** then = System.*nanoTime*();

**double** unprocessed = 0;

**while** (running) {

**boolean** shouldRender = **false**;

**double** now = System.*nanoTime*();

unprocessed += (now - then) / nsPerUpdate;

then = now;

// Update queue

**while** (unprocessed >= 1) {

// update

updates++;

update();

unprocessed--;

shouldRender = **true**;

}

// Render

**if** (shouldRender) {

fps++;

render();

shouldRender = **false**;

}

**else** {

**try** {

Thread.*sleep*(1);

} **catch** (Exception e) {

e.printStackTrace();

}

}

// FPS timer

**if** (System.*currentTimeMillis*() - fpsTimer > 1000) {

System.***out***.println(fps + " fps" + " " +updates + " updates");

fps = 0;

updates = 0;

fpsTimer += 1000;

}

}

}

**public** **synchronized** **void** start() {

**if** (running) **return**;

running = **true**;

game = **new** Thread(**this**, "game");

game.start();

}

**public** **synchronized** **void** stop() {

**if** (!running) **return**;

running = **false**;

System.*exit*(0);

}

@Override

**public** **void** keyTyped(KeyEvent e) {

}

@Override

**public** **void** keyPressed(KeyEvent e) {

Keys.*keyPressed*(e);

}

@Override

**public** **void** keyReleased(KeyEvent e) {

Keys.*keyReleased*(e);

}

@Override

**public** **void** mouseClicked(MouseEvent e) {

}

@Override

**public** **void** mousePressed(MouseEvent e) {

screen.mousePressed(e);

}

@Override

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

screen.mouseReleased(e);

}

@Override

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

}

@Override

**public** **void** mouseExited(MouseEvent e) {

}

@Override

**public** **void** mouseDragged(MouseEvent e) {

screen.mouseDragged(e);

}

@Override

**public** **void** mouseMoved(MouseEvent e) {

screen.mouseMoved(e);

}

}

* **Class GameBoard:** Là class rất quan trọng của trò chơi chứa các phương thức di chuyển, tính điểm cũng như là thuật toán và logic kiểm tra trò chơi vẫn còn tiếp tục hay đã kết thúc.

**package** com.now2048.game;

**import** java.awt.Color;

**import** java.awt.Graphics2D;

**import** java.awt.event.KeyEvent;

**import** java.awt.image.BufferedImage;

**import** java.util.Random;

**public** **class** GameBoard {

**public** **static** **final** **int** ***LEFT*** = 0;

**public** **static** **final** **int** ***RIGHT*** = 1;

**public** **static** **final** **int** ***UP*** = 2;

**public** **static** **final** **int** ***DOWN*** = 3;

**public** **static** **final** **int** ***ROWS*** = 4;

**public** **static** **final** **int** ***COLS*** = 4;

**private** **final** **int** startingTiles = 2;

**private** Tile[][] board;

**private** **boolean** dead;

**private** **boolean** won;

**private** BufferedImage gameBoard;

**private** **int** x;

**private** **int** y;

**private** **static** **int** *SPACING* = 10;

**public** **static** **int** *BOARD\_WIDTH* = (***COLS*** + 1) \* *SPACING* + ***COLS*** \* Tile.***WIDTH***;

**public** **static** **int** *BOARD\_HEIGHT* = (***ROWS*** + 1) \* *SPACING* + ***ROWS*** \* Tile.***HEIGHT***;

**private** **boolean** hasStarted;

**private** ScoreManager scores;

**private** **int** saveCount = 0;

**public** GameBoard(**int** x, **int** y) {

**this**.x = x;

**this**.y = y;

board = **new** Tile[***ROWS***][***COLS***];

gameBoard = **new** BufferedImage(*BOARD\_WIDTH*, *BOARD\_HEIGHT*, BufferedImage.***TYPE\_INT\_RGB***);

createBoardImage();

scores = **new** ScoreManager(**this**);

scores.loadGame();

**if**(scores.newGame()){

start();

scores.saveGame();

}

**else**{

**for**(**int** i = 0; i < scores.getBoard().length; i++){

**if**(scores.getBoard()[i] == 0) **continue**;

spawn(i / ***ROWS***, i % ***COLS***, scores.getBoard()[i]);

}

dead = checkDead();

won = checkWon();

}

}

**public** **void** reset(){

board = **new** Tile[***ROWS***][***COLS***];

start();

scores.saveGame();

dead = **false**;

won = **false**;

hasStarted = **false**;

saveCount = 0;

}

**private** **void** start() {

**for** (**int** i = 0; i < startingTiles; i++) {

spawnRandom();

}

}

/\*\* Debug method \*/

**private** **void** spawn(**int** row, **int** col, **int** value) {

board[row][col] = **new** Tile(value, getTileX(col), getTileY(row));

}

**private** **void** createBoardImage() {

Graphics2D g = (Graphics2D) gameBoard.getGraphics();

g.setColor(Color.***darkGray***);

g.fillRect(0, 0, *BOARD\_WIDTH*, *BOARD\_HEIGHT*);

g.setColor(Color.***lightGray***);

**for** (**int** row = 0; row < ***ROWS***; row++) {

**for** (**int** col = 0; col < ***COLS***; col++) {

**int** x = *SPACING* + *SPACING* \* col + Tile.***WIDTH*** \* col;

**int** y = *SPACING* + *SPACING* \* row + Tile.***HEIGHT*** \* row;

g.fillRoundRect(x, y, Tile.***WIDTH***, Tile.***HEIGHT***, Tile.***ARC\_WIDTH***, Tile.***ARC\_HEIGHT***);

}

}

}

**public** **void** update() {

saveCount++;

**if** (saveCount >= 120) {

saveCount = 0;

scores.saveGame();

}

checkKeys();

**if** (scores.getCurrentScore() > scores.getCurrentTopScore()) {

scores.setCurrentTopScore(scores.getCurrentScore());

}

**for** (**int** row = 0; row < ***ROWS***; row++) {

**for** (**int** col = 0; col < ***COLS***; col++) {

Tile current = board[row][col];

**if** (current == **null**) **continue**;

current.update();

resetPosition(current, row, col);

**if** (current.getValue() == 2048) {

setWon(**true**);

}

}

}

}

**public** **void** render(Graphics2D g) {

BufferedImage finalBoard = **new** BufferedImage(*BOARD\_WIDTH*, *BOARD\_HEIGHT*, BufferedImage.***TYPE\_INT\_ARGB***);

Graphics2D g2d = (Graphics2D) finalBoard.getGraphics();

g2d.setColor(**new** Color(0, 0, 0, 0));

g2d.fillRect(0, 0, *BOARD\_WIDTH*, *BOARD\_HEIGHT*);

g2d.drawImage(gameBoard, 0, 0, **null**);

**for** (**int** row = 0; row < ***ROWS***; row++) {

**for** (**int** col = 0; col < ***COLS***; col++) {

Tile current = board[row][col];

**if** (current == **null**) **continue**;

current.render(g2d);

}

}

g.drawImage(finalBoard, x, y, **null**);

g2d.dispose();

}

**private** **void** resetPosition(Tile tile, **int** row, **int** col) {

**if** (tile == **null**) **return**;

**int** x = getTileX(col);

**int** y = getTileY(row);

**int** distX = tile.getX() - x;

**int** distY = tile.getY() - y;

**if** (Math.*abs*(distX) < Tile.***SLIDE\_SPEED***) {

tile.setX(tile.getX() - distX);

}

**if** (Math.*abs*(distY) < Tile.***SLIDE\_SPEED***) {

tile.setY(tile.getY() - distY);

}

**if** (distX < 0) {

tile.setX(tile.getX() + Tile.***SLIDE\_SPEED***);

}

**if** (distY < 0) {

tile.setY(tile.getY() + Tile.***SLIDE\_SPEED***);

}

**if** (distX > 0) {

tile.setX(tile.getX() - Tile.***SLIDE\_SPEED***);

}

**if** (distY > 0) {

tile.setY(tile.getY() - Tile.***SLIDE\_SPEED***);

}

}

**public** **int** getTileX(**int** col) {

**return** *SPACING* + col \* Tile.***WIDTH*** + col \* *SPACING*;

}

**public** **int** getTileY(**int** row) {

**return** *SPACING* + row \* Tile.***HEIGHT*** + row \* *SPACING*;

}

**private** **boolean** checkOutOfBounds(**int** direction, **int** row, **int** col) {

**if** (direction == ***LEFT***) {

**return** col < 0;

}

**else** **if** (direction == ***RIGHT***) {

**return** col > ***COLS*** - 1;

}

**else** **if** (direction == ***UP***) {

**return** row < 0;

}

**else** **if** (direction == ***DOWN***) {

**return** row > ***ROWS*** - 1;

}

**return** **false**;

}

**private** **boolean** move(**int** row, **int** col, **int** horizontalDirection, **int** verticalDirection, **int** direction) {

**boolean** canMove = **false**;

Tile current = board[row][col];

**if** (current == **null**) **return** **false**;

**boolean** move = **true**;

**int** newCol = col;

**int** newRow = row;

**while** (move) {

newCol += horizontalDirection;

newRow += verticalDirection;

**if** (checkOutOfBounds(direction, newRow, newCol)) **break**;

**if** (board[newRow][newCol] == **null**) {

board[newRow][newCol] = current;

canMove = **true**;

board[newRow - verticalDirection][newCol - horizontalDirection] = **null**;

board[newRow][newCol].setSlideTo(**new** Point(newRow, newCol));

}

**else** **if** (board[newRow][newCol].getValue() == current.getValue() && board[newRow][newCol].canCombine()) {

board[newRow][newCol].setCanCombine(**false**);

board[newRow][newCol].setValue(board[newRow][newCol].getValue() \* 2);

canMove = **true**;

board[newRow - verticalDirection][newCol - horizontalDirection] = **null**;

board[newRow][newCol].setSlideTo(**new** Point(newRow, newCol));

scores.setCurrentScore(scores.getCurrentScore() + board[newRow][newCol].getValue());

}

**else** {

move = **false**;

}

}

**return** canMove;

}

**public** **void** moveTiles(**int** direction) {

**boolean** canMove = **false**;

**int** horizontalDirection = 0;

**int** verticalDirection = 0;

**if** (direction == ***LEFT***) {

horizontalDirection = -1;

**for** (**int** row = 0; row < ***ROWS***; row++) {

**for** (**int** col = 0; col < ***COLS***; col++) {

**if** (!canMove)

canMove = move(row, col, horizontalDirection, verticalDirection, direction);

**else** move(row, col, horizontalDirection, verticalDirection, direction);

}

}

}

**else** **if** (direction == ***RIGHT***) {

horizontalDirection = 1;

**for** (**int** row = 0; row < ***ROWS***; row++) {

**for** (**int** col = ***COLS*** - 1; col >= 0; col--) {

**if** (!canMove)

canMove = move(row, col, horizontalDirection, verticalDirection, direction);

**else** move(row, col, horizontalDirection, verticalDirection, direction);

}

}

}

**else** **if** (direction == ***UP***) {

verticalDirection = -1;

**for** (**int** row = 0; row < ***ROWS***; row++) {

**for** (**int** col = 0; col < ***COLS***; col++) {

**if** (!canMove)

canMove = move(row, col, horizontalDirection, verticalDirection, direction);

**else** move(row, col, horizontalDirection, verticalDirection, direction);

}

}

}

**else** **if** (direction == ***DOWN***) {

verticalDirection = 1;

**for** (**int** row = ***ROWS*** - 1; row >= 0; row--) {

**for** (**int** col = 0; col < ***COLS***; col++) {

**if** (!canMove)

canMove = move(row, col, horizontalDirection, verticalDirection, direction);

**else** move(row, col, horizontalDirection, verticalDirection, direction);

}

}

}

**else** {

System.***out***.println(direction + " is not a valid direction.");

}

**for** (**int** row = 0; row < ***ROWS***; row++) {

**for** (**int** col = 0; col < ***COLS***; col++) {

Tile current = board[row][col];

**if** (current == **null**) **continue**;

current.setCanCombine(**true**);

}

}

**if** (canMove) {

spawnRandom();

setDead(checkDead());

}

}

**private** **boolean** checkDead() {

**for** (**int** row = 0; row < ***ROWS***; row++) {

**for** (**int** col = 0; col < ***COLS***; col++) {

**if** (board[row][col] == **null**) **return** **false**;

**boolean** canCombine = checkSurroundingTiles(row, col, board[row][col]);

**if** (canCombine) {

**return** **false**;

}

}

}

**return** **true**;

}

**private** **boolean** checkWon() {

**for** (**int** row = 0; row < ***ROWS***; row++) {

**for** (**int** col = 0; col < ***COLS***; col++) {

**if**(board[row][col] == **null**) **continue**;

**if**(board[row][col].getValue() >= 2048) **return** **true**;

}

}

**return** **false**;

}

**private** **boolean** checkSurroundingTiles(**int** row, **int** col, Tile tile) {

**if** (row > 0) {

Tile check = board[row - 1][col];

**if** (check == **null**) **return** **true**;

**if** (tile.getValue() == check.getValue()) **return** **true**;

}

**if** (row < ***ROWS*** - 1) {

Tile check = board[row + 1][col];

**if** (check == **null**) **return** **true**;

**if** (tile.getValue() == check.getValue()) **return** **true**;

}

**if** (col > 0) {

Tile check = board[row][col - 1];

**if** (check == **null**) **return** **true**;

**if** (tile.getValue() == check.getValue()) **return** **true**;

}

**if** (col < ***COLS*** - 1) {

Tile check = board[row][col + 1];

**if** (check == **null**) **return** **true**;

**if** (tile.getValue() == check.getValue()) **return** **true**;

}

**return** **false**;

}

**private** **void** spawnRandom() {

Random random = **new** Random();

**boolean** notValid = **true**;

**while** (notValid) {

**int** location = random.nextInt(16);

**int** row = location / ***ROWS***;

**int** col = location % ***COLS***;

Tile current = board[row][col];

**if** (current == **null**) {

**int** value = random.nextInt(10) < 9 ? 2 : 4;

Tile tile = **new** Tile(value, getTileX(col), getTileY(row));

board[row][col] = tile;

notValid = **false**;

}

}

}

**private** **void** checkKeys() {

**if** (!Keys.*pressed*[KeyEvent.***VK\_LEFT***] && Keys.*prev*[KeyEvent.***VK\_LEFT***]) {

moveTiles(***LEFT***);

**if** (!hasStarted) hasStarted = !dead;

}

**if** (!Keys.*pressed*[KeyEvent.***VK\_RIGHT***] && Keys.*prev*[KeyEvent.***VK\_RIGHT***]) {

moveTiles(***RIGHT***);

**if** (!hasStarted) hasStarted = !dead;

}

**if** (!Keys.*pressed*[KeyEvent.***VK\_UP***] && Keys.*prev*[KeyEvent.***VK\_UP***]) {

moveTiles(***UP***);

**if** (!hasStarted) hasStarted = !dead;

}

**if** (!Keys.*pressed*[KeyEvent.***VK\_DOWN***] && Keys.*prev*[KeyEvent.***VK\_DOWN***]) {

moveTiles(***DOWN***);

**if** (!hasStarted) hasStarted = !dead;

}

**if** (!Keys.*pressed*[KeyEvent.***VK\_N***] && Keys.*prev*[KeyEvent.***VK\_N***]) {

reset();

getScores().reset();

}

}

**public** **boolean** isDead() {

**return** dead;

}

**public** **void** setDead(**boolean** dead) {

**this**.dead = dead;

}

**public** Tile[][] getBoard() {

**return** board;

}

**public** **void** setBoard(Tile[][] board) {

**this**.board = board;

}

**public** **int** getX() {

**return** x;

}

**public** **void** setX(**int** x) {

**this**.x = x;

}

**public** **int** getY() {

**return** y;

}

**public** **void** setY(**int** y) {

**this**.y = y;

}

**public** **boolean** isWon() {

**return** won;

}

**public** **void** setWon(**boolean** won) {

**this**.won = won;

}

**public** ScoreManager getScores(){

**return** scores;

}

}

* **Class Tile:** Class chịu trách nhiệm vẽ các ô số, thêm vào màu sắc, font chữ, tốc độ trượt, ngoài ra còn có những phương thức để lấy giá trị các ô số, hay xét khả năng kết hợp của hai ô số trong bảng với nhau.

**package** com.now2048.game;

**import** java.awt.Color;

**import** java.awt.Font;

**import** java.awt.Graphics2D;

**import** java.awt.RenderingHints;

**import** java.awt.image.BufferedImage;

**public** **class** Tile {

**public** **static** **final** **int** ***WIDTH*** = 100;

**public** **static** **final** **int** ***HEIGHT*** = 100;

**public** **static** **final** **int** ***SLIDE\_SPEED*** = 20;

**public** **static** **final** **int** ***ARC\_WIDTH*** = 15;

**public** **static** **final** **int** ***ARC\_HEIGHT*** =15;

**private** **int** value;

**private** BufferedImage tileImage;

**private** Color background;

**private** Color text;

**private** Font font;

**private** Point slideTo;

**private** **int** x;

**private** **int** y;

**private** BufferedImage beginningImage;

**private** **boolean** canCombine = **true**;

**public** Tile(**int** value, **int** x, **int** y) {

**this**.value = value;

**this**.x = x;

**this**.y = y;

slideTo = **new** Point(x, y);

tileImage = **new** BufferedImage(***WIDTH***, ***HEIGHT***, BufferedImage.***TYPE\_INT\_ARGB***);

beginningImage = **new** BufferedImage(***WIDTH***, ***HEIGHT***, BufferedImage.***TYPE\_INT\_ARGB***);

drawImage();

}

**public** **void** update() {

Graphics2D g2d = (Graphics2D) beginningImage.getGraphics();

g2d.setRenderingHint(RenderingHints.***KEY\_INTERPOLATION***, RenderingHints.***VALUE\_INTERPOLATION\_BICUBIC***);

g2d.setColor(**new** Color(0, 0, 0, 0));

g2d.fillRect(0, 0, ***WIDTH***, ***HEIGHT***);

g2d.dispose();

}

**public** **void** render(Graphics2D g){

g.drawImage(tileImage, x, y, **null**);

}

**private** **void** drawImage() {

Graphics2D g = (Graphics2D) tileImage.getGraphics();

**if** (value == 2) {

background = **new** Color(0xe9e9e9);

text = **new** Color(0x000000);

}

**else** **if** (value == 4) {

background = **new** Color(0xe6daab);

text = **new** Color(0x000000);

}

**else** **if** (value == 8) {

background = **new** Color(0xf79d3d);

text = **new** Color(0xffffff);

}

**else** **if** (value == 16) {

background = **new** Color(0xf28007);

text = **new** Color(0xffffff);

}

**else** **if** (value == 32) {

background = **new** Color(0xf55e3b);

text = **new** Color(0xffffff);

}

**else** **if** (value == 64) {

background = **new** Color(0xff0000);

text = **new** Color(0xffffff);

}

**else** **if** (value == 128) {

background = **new** Color(0xe9de84);

text = **new** Color(0xffffff);

}

**else** **if** (value == 256) {

background = **new** Color(0xf6e873);

text = **new** Color(0xffffff);

}

**else** **if** (value == 512) {

background = **new** Color(0xf5e455);

text = **new** Color(0xffffff);

}

**else** **if** (value == 1024) {

background = **new** Color(0xf7e12c);

text = **new** Color(0xffffff);

}

**else** **if** (value == 2048) {

background = **new** Color(0xffe400);

text = **new** Color(0xffffff);

}

**else** **if**(value == 0){

background = Color.***lightGray***;

text = Color.***black***;

}

**else**{

background = **new** Color(0x000000);

text = **new** Color(0xffffff);

}

g.setColor(**new** Color(0, 0, 0, 0));

g.fillRect(0, 0, ***WIDTH***, ***HEIGHT***);

g.setColor(background);

g.fillRoundRect(0, 0, ***WIDTH***, ***HEIGHT***, ***ARC\_WIDTH***, ***ARC\_HEIGHT***);

g.setColor(text);

**if** (value <= 64) {

font = Game.***main***.deriveFont(36f);

g.setFont(font);

}

**else** {

font = Game.***main***;

g.setFont(font);

}

**int** drawX = ***WIDTH*** / 2 - DrawUtils.*getMessageWidth*(""+ value, font, g)/2;

**int** drawY = ***HEIGHT*** / 2 + DrawUtils.*getMessageHeight*(""+ value,font,g)/2;

g.drawString("" + value, drawX, drawY);

g.dispose();

}

**public** **int** getValue() {

**return** value;

}

**public** **void** setValue(**int** value) {

**this**.value = value;

drawImage();

}

**public** Point getSlideTo() {

**return** slideTo;

}

**public** **void** setSlideTo(Point slideTo) {

**this**.slideTo = slideTo;

}

**public** **int** getX() {

**return** x;

}

**public** **void** setX(**int** x) {

**this**.x = x;

}

**public** **int** getY() {

**return** y;

}

**public** **void** setY(**int** y) {

**this**.y = y;

}

**public** **boolean** canCombine() {

**return** canCombine;

}

**public** **void** setCanCombine(**boolean** canCombine) {

**this**.canCombine = canCombine;

}

}

* **Class Keys:** Lớp này chứa các biến và phương thức để xử lý các sự kiện phím trong trò chơi.

**package** com.now2048.game;

**import** java.awt.event.KeyEvent;

**public** **class** Keys {

**public** **static** **boolean**[] *pressed* = **new** **boolean**[256];

**public** **static** **boolean**[] *prev* = **new** **boolean**[256];

**public** **static** **void** update(){

**for**(**int** i = 0; i < 5; i++){

**if**(i == 0) *prev*[KeyEvent.***VK\_LEFT***] = *pressed*[KeyEvent.***VK\_LEFT***];

**if**(i == 1) *prev*[KeyEvent.***VK\_RIGHT***] = *pressed*[KeyEvent.***VK\_RIGHT***];

**if**(i == 2) *prev*[KeyEvent.***VK\_UP***] = *pressed*[KeyEvent.***VK\_UP***];

**if**(i == 3) *prev*[KeyEvent.***VK\_DOWN***] = *pressed*[KeyEvent.***VK\_DOWN***];

**if**(i == 4) *prev*[KeyEvent.***VK\_N***] = *pressed*[KeyEvent.***VK\_N***];

}

}

**public** **static** **void** keyPressed(KeyEvent e){

*pressed*[e.getKeyCode()] = **true**;

}

**public** **static** **void** keyReleased(KeyEvent e){

*pressed*[e.getKeyCode()] = **false**;

}

}

* **Class DrawUtils:** Lớp này chứa các phương thức để hỗ trợ vẽ văn bản trong trò chơi. Quy định về chiều dài và chiều rộng của một đoạn văn bản nào đó xuất hiện trong trò chơi.

**package** com.now2048.game;

**import** java.awt.Font;

**import** java.awt.Graphics2D;

**import** java.awt.font.TextLayout;

**import** java.awt.geom.Rectangle2D;

**public** **class** DrawUtils {

**public** **static** **int** getMessageWidth(String message, Font font, Graphics2D g) {

g.setFont(font);

Rectangle2D bounds = g.getFontMetrics().getStringBounds(message, g);

**return** (**int**) bounds.getWidth();

}

**public** **static** **int** getMessageHeight(String message, Font font, Graphics2D g) {

g.setFont(font);

**if**(message.length() == 0) **return** 0;

TextLayout tl = **new** TextLayout(message, font, g.getFontRenderContext());

**return** (**int**) tl.getBounds().getHeight();

}

}

* **Class ScoreManager:** Lớp này chứa các phương thức để ghi lại dữ liệu của trò chơi cũng như các phương thức giúp đọc lại dữ liệu được ghi trước đó giúp người chơi có thể tiếp tục trò chơi ngay cả khi đã thoát ra.

**package** com.now2048.game;

**import** java.io.BufferedReader;

**import** java.io.BufferedWriter;

**import** java.io.File;

**import** java.io.FileInputStream;

**import** java.io.FileWriter;

**import** java.io.InputStreamReader;

**public** **class** ScoreManager {

// CURRENT SCORES

**private** **int** currentScore;

**private** **int** currentTopScore;

**private** **int**[] board = **new** **int**[16];

// File

**private** String filePath;

**private** String temp;

**private** GameBoard gBoard;

// Random booleans

**private** **boolean** newGame;

**public** ScoreManager(GameBoard gBoard) {

**try** {

filePath = **new** File("").getAbsolutePath();

} **catch** (Exception e) {

e.printStackTrace();

}

temp = "TEMP.tmp";

**this**.gBoard = gBoard;

}

**public** **void** reset() {

File f = **new** File(filePath, temp);

**if** (f.isFile()) {

f.delete();

}

newGame = **true**;

currentScore = 0;

}

**private** **void** createFile() {

FileWriter output = **null**;

newGame = **true**;

**try** {

File f = **new** File(filePath, temp);

output = **new** FileWriter(f);

BufferedWriter writer = **new** BufferedWriter(output);

writer.write("" + 0);

writer.newLine();

writer.write("" + 0);

writer.newLine();

**for** (**int** row = 0; row < GameBoard.***ROWS***; row++) {

**for** (**int** col = 0; col < GameBoard.***COLS***; col++) {

**if**(row == GameBoard.***ROWS*** - 1 && col == GameBoard.***COLS*** - 1){

writer.write("" + 0);

}

**else**{

writer.write(0 + "-");

}

}

}

writer.close();

} **catch** (Exception e) {

e.printStackTrace();

}

}

**public** **void** saveGame() {

FileWriter output = **null**;

**if** (newGame) newGame = **false**;

**try** {

File f = **new** File(filePath, temp);

output = **new** FileWriter(f);

BufferedWriter writer = **new** BufferedWriter(output);

writer.write("" + currentScore);

writer.newLine();

writer.write("" + currentTopScore);

writer.newLine();

**for** (**int** row = 0; row < GameBoard.***ROWS***; row++) {

**for** (**int** col = 0; col < GameBoard.***COLS***; col++) {

**this**.board[row \* GameBoard.***COLS*** + col] = gBoard.getBoard()[row][col] != **null** ? gBoard.getBoard()[row][col].getValue() : 0;

**if** (row == GameBoard.***ROWS*** - 1 && col == GameBoard.***COLS*** - 1)

writer.write("" + board[row \* GameBoard.***COLS*** + col]);

**else** writer.write(board[row \* GameBoard.***COLS*** + col] + "-");

}

}

writer.close();

} **catch** (Exception e) {

e.printStackTrace();

}

}

**public** **void** loadGame() {

**try** {

File f = **new** File(filePath, temp);

**if** (!f.isFile()) {

createFile();

}

BufferedReader reader = **new** BufferedReader(**new** InputStreamReader(**new** FileInputStream(f)));

currentScore = Integer.*parseInt*(reader.readLine());

currentTopScore = Integer.*parseInt*(reader.readLine());

String[] board = reader.readLine().split("-");

**for** (**int** i = 0; i < board.length; i++) {

**this**.board[i] = Integer.*parseInt*(board[i]);

}

reader.close();

} **catch** (Exception e) {

e.printStackTrace();

}

}

**public** **int** getCurrentScore() {

**return** currentScore;

}

**public** **void** setCurrentScore(**int** currentScore) {

**this**.currentScore = currentScore;

}

**public** **int** getCurrentTopScore() {

**return** currentTopScore;

}

**public** **void** setCurrentTopScore(**int** currentTopScore) {

**this**.currentTopScore = currentTopScore;

}

**public** **boolean** newGame() {

**return** newGame;

}

**public** **int**[] getBoard() {

**return** board;

}

}

* **Class Point:** Lớp này đại diện cho một điểm trên bảng trò chơi.

**package** com.now2048.game;

**public** **class** Point {

**public** **int** row;

**public** **int** col;

**public** Point(**int** row, **int** col){

**this**.row = row;

**this**.col = col;

}

}

1. **Package com.now2048.gui**

* **Class Guiscreen:** Là lớp quản lý các giao diện đồ họa trong trò chơi. Chứa các phương thức thêm, cập nhật và vẽ các giao diện, và xử lý các sự kiện chuột trên giao diện đang hiển thị.

**package** com.now2048.gui;

**import** java.awt.Graphics2D;

**import** java.awt.event.MouseEvent;

**import** java.util.HashMap;

**public** **class** GuiScreen {

**private** **static** GuiScreen *screen*;

**private** HashMap<String, GuiPanel> panels;

**private** String currentPanel = "";

**private** GuiScreen() {

panels = **new** HashMap<String, GuiPanel>();

}

**public** **static** GuiScreen getInstance() {

**if** (*screen* == **null**) {

*screen* = **new** GuiScreen();

}

**return** *screen*;

}

**public** **void** update() {

**if** (panels.get(currentPanel) != **null**) {

panels.get(currentPanel).update();

}

}

**public** **void** render(Graphics2D g) {

**if** (panels.get(currentPanel) != **null**) {

panels.get(currentPanel).render(g);

}

}

**public** **void** add(String panelName, GuiPanel panel) {

panels.put(panelName, panel);

}

**public** **void** setCurrentPanel(String panelName) {

currentPanel = panelName;

}

**public** **void** mousePressed(MouseEvent e) {

**if** (panels.get(currentPanel) != **null**) {

panels.get(currentPanel).mousePressed(e);

}

}

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

**if** (panels.get(currentPanel) != **null**) {

panels.get(currentPanel).mouseReleased(e);

}

}

**public** **void** mouseDragged(MouseEvent e) {

**if** (panels.get(currentPanel) != **null**) {

panels.get(currentPanel).mouseDragged(e);

}

}

**public** **void** mouseMoved(MouseEvent e) {

**if** (panels.get(currentPanel) != **null**) {

panels.get(currentPanel).mouseMoved(e);

}

}

}

* **Class GuiPanel:** Đại diện cho một bảng điều khiển các giao diện trong trò chơi.

**package** com.now2048.gui;

**import** java.awt.Graphics2D;

**import** java.awt.event.MouseEvent;

**import** java.util.ArrayList;

**public** **class** GuiPanel {

**private** ArrayList<GuiButton> buttons;

**public** GuiPanel(){

buttons = **new** ArrayList<GuiButton>();

}

**public** **void** update(){

**for**(GuiButton b : buttons){

b.update();

}

}

**public** **void** render(Graphics2D g){

**for**(GuiButton b : buttons){

b.render(g);

}

}

**public** **void** add(GuiButton button){

buttons.add(button);

}

**public** **void** remove(GuiButton button){

buttons.remove(button);

}

**public** **void** mousePressed(MouseEvent e){

**for**(GuiButton b : buttons){

b.mousePressed(e);

}

}

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

**for**(GuiButton b : buttons){

b.mouseReleased(e);

}

}

**public** **void** mouseDragged(MouseEvent e){

**for**(GuiButton b : buttons){

b.mouseDragged(e);

}

}

**public** **void** mouseMoved(MouseEvent e){

**for**(GuiButton b : buttons){

b.mouseMoved(e);

}

}

}

* **Class GuiButton:** Class quy định các thông tin của nút bấm sẽ được gọi ra trong trò chơi như kiểu hình, màu sắc, font chữ, cỡ chữ, hay khả năng thay đổi màu sắc nút bấm khi thực hiện các sự kiện chuột.

**package** com.now2048.gui;

**import** java.awt.Color;

**import** java.awt.Font;

**import** java.awt.Graphics2D;

**import** java.awt.Rectangle;

**import** java.awt.event.ActionListener;

**import** java.awt.event.MouseEvent;

**import** java.util.ArrayList;

**import** com.now2048.game.DrawUtils;

**import** com.now2048.game.Game;

**public** **class** GuiButton {

**private** State currentState = State.***RELEASED***;

**private** Rectangle clickBox;

**private** ArrayList<ActionListener> actionListeners;

**private** String text = "";

**private** Color main;

**private** Color hover;

**private** Color pressed;

**private** Font font = Game.***main***.deriveFont(22f);

**public** GuiButton(**int** x, **int** y, **int** width, **int** height){

clickBox = **new** Rectangle(x, y, width, height);

actionListeners = **new** ArrayList<ActionListener>();

main = **new** Color(173, 177, 179);

hover = **new** Color(150, 156, 158);

pressed = **new** Color(111, 116, 117);

}

**public** **void** update(){

}

**public** **void** render(Graphics2D g){

**if**(currentState == State.***RELEASED***){

g.setColor(main);

g.fill(clickBox);

}

**else** **if**(currentState == State.***PRESSED***){

g.setColor(pressed);

g.fill(clickBox);

}

**else**{

g.setColor(hover);

g.fill(clickBox);

}

g.setColor(Color.***white***);

g.setFont(font);

g.drawString(text, clickBox.x + clickBox.width / 2 - DrawUtils.*getMessageWidth*(text, font, g) / 2, clickBox.y + clickBox.height / 2 + DrawUtils.*getMessageHeight*(text, font, g) / 2);

}

**public** **void** addActionListener(ActionListener listener){

actionListeners.add(listener);

}

**public** **void** mousePressed(MouseEvent e) {

**if**(clickBox.contains(e.getPoint())){

currentState = State.***PRESSED***;

}

}

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

**if**(clickBox.contains(e.getPoint())){

**for**(ActionListener al : actionListeners){

al.actionPerformed(**null**);

}

}

currentState = State.***RELEASED***;

}

**public** **void** mouseDragged(MouseEvent e) {

**if**(clickBox.contains(e.getPoint())){

currentState = State.***PRESSED***;

}

**else**{

currentState = State.***RELEASED***;

}

}

**public** **void** mouseMoved(MouseEvent e) {

**if**(clickBox.contains(e.getPoint())){

currentState = State.***HOVER***;

}

**else**{

currentState = State.***RELEASED***;

}

}

**public** **int** getX(){

**return** clickBox.x;

}

**public** **int** getY(){

**return** clickBox.y;

}

**public** **int** getWidth(){

**return** clickBox.width;

}

**public** **int** getHeight(){

**return** clickBox.height;

}

**public** **void** setText(String text){

**this**.text = text;

}

**private** **enum** State{

***HOVER***, ***RELEASED***, ***PRESSED***

}

}

* **Class MainMenuPanel:** Class chứa giao diện của trò chơi khi trò chơi được bắt đầu.

**package** com.now2048.gui;

**import** java.awt.Color;

**import** java.awt.Font;

**import** java.awt.Graphics2D;

**import** java.awt.event.ActionEvent;

**import** java.awt.event.ActionListener;

**import** com.now2048.game.DrawUtils;

**import** com.now2048.game.Game;

**public** **class** MainMenuPanel **extends** GuiPanel {

**private** Font titleFont = Game.***main***.deriveFont(100f);

**private** String title = "2048";

**private** **int** buttonWidth = 220;

**public** MainMenuPanel() {

GuiButton playButton = **new** GuiButton(Game.***WIDTH*** / 2 - buttonWidth / 2, 280, buttonWidth, 60);

playButton.addActionListener(**new** ActionListener() {

@Override

**public** **void** actionPerformed(ActionEvent e) {

GuiScreen.*getInstance*().setCurrentPanel("Play");

}

});

playButton.setText("Play");

add(playButton);

GuiButton quitButton = **new** GuiButton(Game.***WIDTH*** / 2 - buttonWidth / 2, 400, buttonWidth, 60);

quitButton.addActionListener(**new** ActionListener() {

@Override

**public** **void** actionPerformed(ActionEvent e) {

System.*exit*(0);

}

});

quitButton.setText("Quit");

add(quitButton);

}

@Override

**public** **void** render(Graphics2D g){

**super**.render(g);

g.setFont(titleFont);

g.setColor(Color.***black***);

g.drawString(title, Game.***WIDTH*** / 2 - DrawUtils.*getMessageWidth*(title, titleFont, g) / 2, 150);

}

}

* **Class PlayPanel:** Class này chứa giao diện của trò chơi khi trò chơi đã kết thúc.

**package** com.now2048.gui;

**import** java.awt.Color;

**import** java.awt.Font;

**import** java.awt.Graphics2D;

**import** java.awt.event.ActionEvent;

**import** java.awt.event.ActionListener;

**import** java.awt.image.BufferedImage;

**import** com.now2048.game.DrawUtils;

**import** com.now2048.game.Game;

**import** com.now2048.game.GameBoard;

**import** com.now2048.game.ScoreManager;

**public** **class** PlayPanel **extends** GuiPanel {

**private** GameBoard board;

**private** BufferedImage info;

**private** ScoreManager scores;

**private** Font scoreFont;

// Game Over

**private** GuiButton tryAgain;

**private** GuiButton mainMenu;

**private** **int** spacing = 20;

**private** **int** ButtonWidth = 160 \* 2 + spacing;

**private** **int** buttonHeight = 50;

**private** **boolean** added;

**private** **int** alpha = 0;

**private** Font gameOverFont;

**public** PlayPanel() {

scoreFont = Game.***main***.deriveFont(24f);

gameOverFont = Game.***main***.deriveFont(70f);

board = **new** GameBoard(Game.***WIDTH*** / 2 - GameBoard.*BOARD\_WIDTH* / 2, Game.***HEIGHT*** - GameBoard.*BOARD\_HEIGHT* - 20);

scores = board.getScores();

info = **new** BufferedImage(Game.***WIDTH***, 200, BufferedImage.***TYPE\_INT\_RGB***);

mainMenu = **new** GuiButton(Game.***WIDTH*** / 2 - ButtonWidth / 2, 450, ButtonWidth, buttonHeight);

tryAgain = **new** GuiButton(mainMenu.getX(), mainMenu.getY() - spacing - buttonHeight, ButtonWidth, buttonHeight);

tryAgain.setText("Try Again");

mainMenu.setText("Back to Main Menu");

tryAgain.addActionListener(**new** ActionListener() {

**public** **void** actionPerformed(ActionEvent e) {

board.getScores().reset();

board.reset();

alpha = 0;

remove(tryAgain);

remove(mainMenu);

added = **false**;

}

});

mainMenu.addActionListener(**new** ActionListener() {

**public** **void** actionPerformed(ActionEvent e) {

GuiScreen.*getInstance*().setCurrentPanel("Menu");

}

});

}

**private** **void** drawGui(Graphics2D g) {

// Draw it

Graphics2D g2d = (Graphics2D) info.getGraphics();

g2d.setColor(Color.***white***);

g2d.fillRect(0, 0, info.getWidth(), info.getHeight());

g2d.setColor(Color.***lightGray***);

g2d.setFont(scoreFont);

g2d.drawString("" + scores.getCurrentScore(), 30, 40);

g2d.setColor(Color.***red***);

g2d.drawString("Best: " + scores.getCurrentTopScore(), Game.***WIDTH*** - DrawUtils.*getMessageWidth*("Best: " + scores.getCurrentTopScore(), scoreFont, g2d) - 20, 40);

g2d.dispose();

g.drawImage(info, 0, 0, **null**);

}

**public** **void** drawGameOver(Graphics2D g) {

g.setColor(**new** Color(222, 222, 222, alpha));

g.fillRect(0, 0, Game.***WIDTH***, Game.***HEIGHT***);

g.setColor(Color.***red***);

g.drawString("Game Over!", Game.***WIDTH*** / 2 - DrawUtils.*getMessageWidth*("Game Over!", gameOverFont, g) / 2, 250);

}

@Override

**public** **void** update() {

board.update();

**if** (board.isDead()) {

alpha++;

**if** (alpha > 170) alpha = 170;

}

}

@Override

**public** **void** render(Graphics2D g) {

drawGui(g);

board.render(g);

**if** (board.isDead()) {

**if** (!added) {

added = **true**;

add(mainMenu);

add(tryAgain);

}

drawGameOver(g);

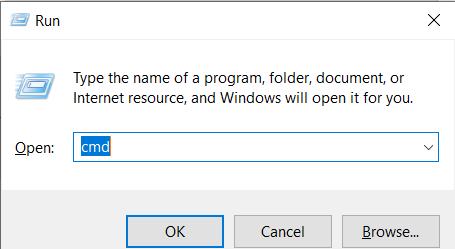
}

**super**.render(g);

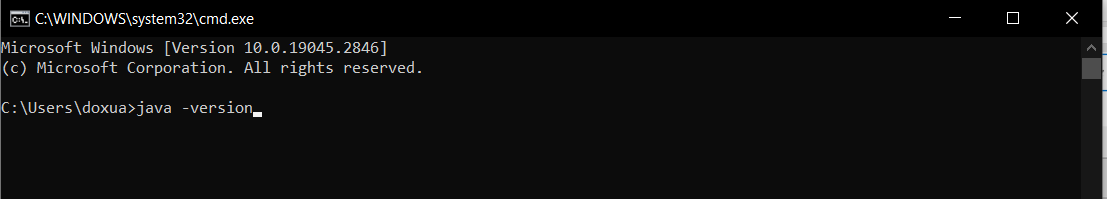
}

1. **Hướng dẫn cài đặt người dùng**

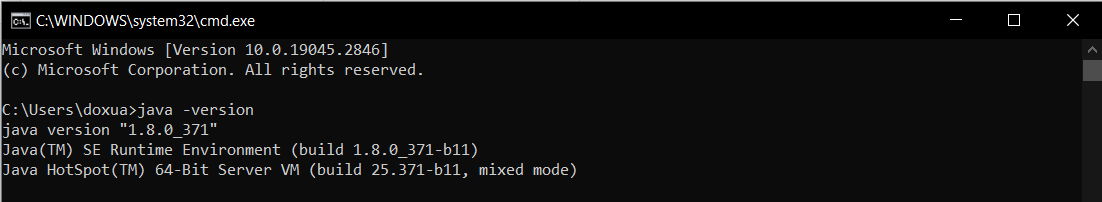
**Bước 1:** Kiểm tra xem trong máy đã được cài đặt Java hay chưa?

Nhấn tổ hợp phím Window+R, trong cửa sổ Run gõ “cmd” sau đó chọn OK.

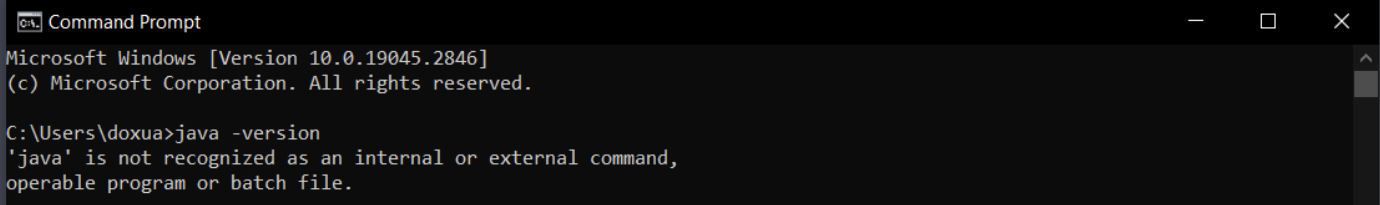
Sau khi chọn OK, cửa sổ Command Prompt sẽ hiện ra gõ vào đó lệnh java –verison để kiểm tra java trong máy đã được cài đặt hay chưa.



Nếu máy đã được cài đặt java sẽ xuất hiện thông báo như sau:



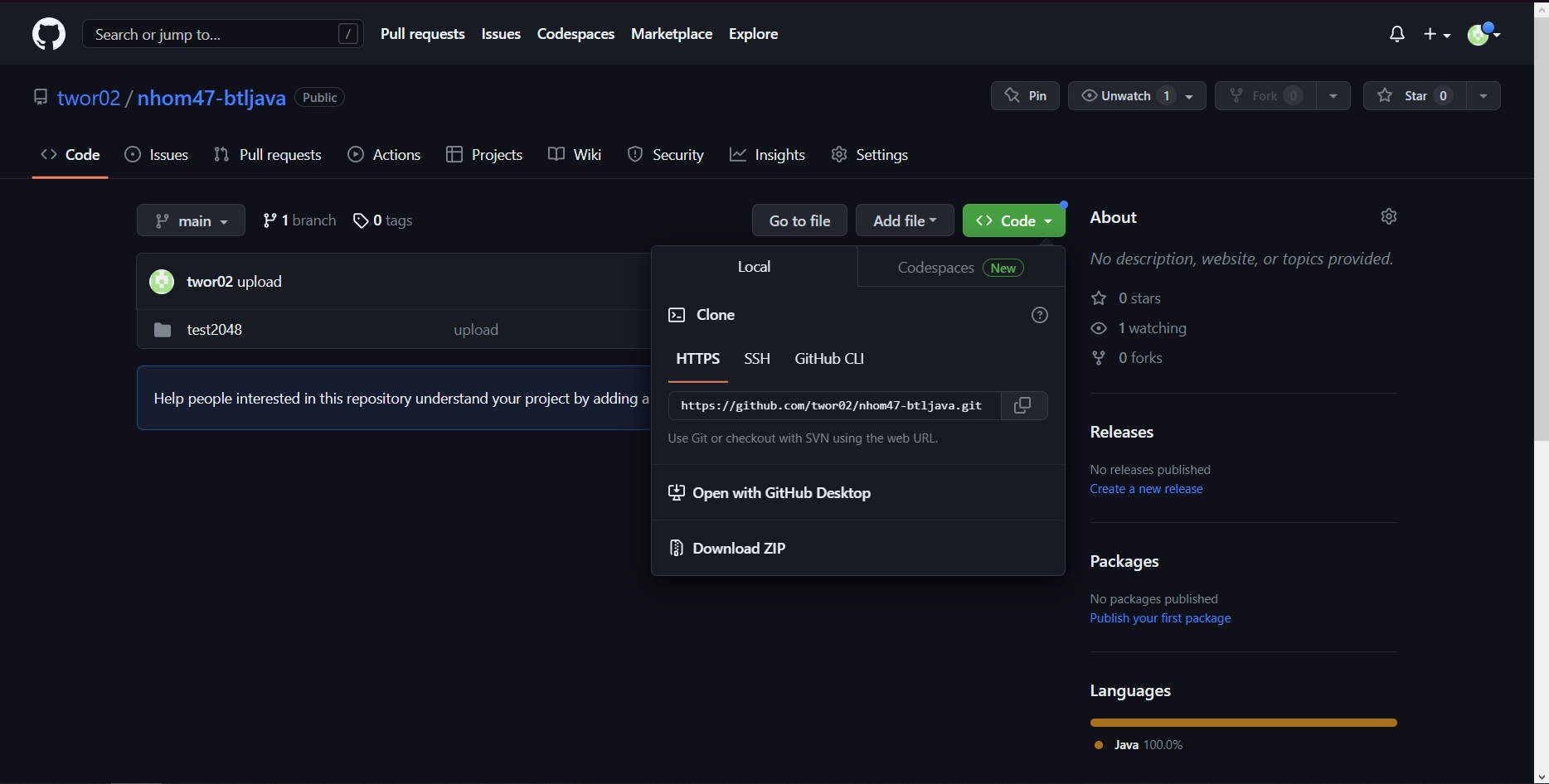
Nếu không xuất hiện được thông báo như trên, mà thay vào đó hiển thị ra thông báo sau:



Nghĩa là máy bạn chưa được cài đặt java. Hãy truy cập vào: <https://www.java.com/en/download/>

Sau đó chọn “Download Java” và tải về máy, nếu tải thành công thì máy của bạn đã được cài đặt java.

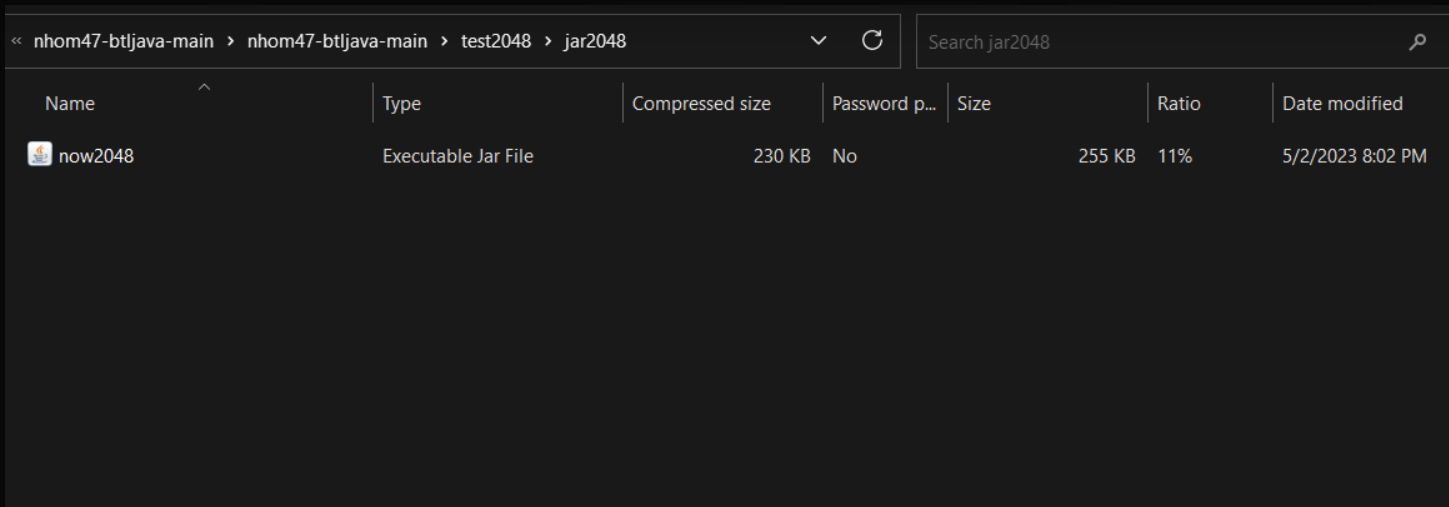
**Bước 2:** Sau khi đã cài đặt java thành công, hãy truy cập đường link: <https://github.com/twor02/nhom47-btljava>, chọn nút Code màu xanh lá như hình và chọn “Download ZIP” để tải folder code về.



**Bước 3:** Sau khi tải folder về thành công, truy cập theo dường dẫn:

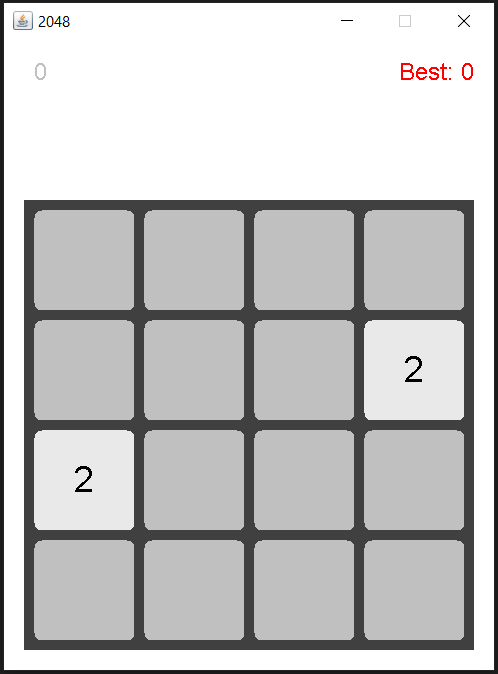
nhom47-btljava-main.zip\nhom47-btljava-main\test2048\jar2048

Sẽ xuất hiện một màn hình như sau:



Chỉ cần mở file now2048 như trên hình là bạn sẽ khởi chạy được trò chơi.

**Bước 4:** Sau khi mở trò chơi thành công sẽ xuất hiện cửa sổ:



Lúc này bạn đã có thể trải nghiệm trò chơi. Chúc bạn may mắn!

1. **Tài liệu tham khảo**

* Kênh youtube Fatal Cubez:

<https://www.youtube.com/watch?v=GlXWdQSfddw&list=PLig6-gM-fHMGH6jmCpsxW6YbagHgCS3Jd>

* Kênh youtube Luka Uranic:

<https://www.youtube.com/watch?v=iVRL9jwAwh4&list=PLBU53O4QuJBma22q0v5pXeJxLx66yKLI9>

1. **Tổng kết**

Trong thời gian nghiên cứu và vận dụng những gì đã học ở trường và tìm hiểu thêm trên Internet. Nhóm em đã viết được game 2048 nhằm mục đích giải trí lành mạnh và nghiên cứu môn học. Trong quá trình thực hiện đề tài, chúng em đã nhận được sự giúp đỡ của Thầy Vũ Huấn và các bạn giúp nhóm em hoàn thiện đề tài. Vì kinh nghiệm còn ít nên không tránh khỏi các sai sót, rất mong nhận được sự góp ý của thầy và các bạn.