**BỘ GIÁO DỤC VÀ ĐÀO TẠO**

**TRƯỜNG ĐẠI HỌC SƯ PHẠM THÀNH PHỐ HỒ CHÍ MINH**

**KHOA CÔNG NGHỆ THÔNG TIN**





**BÁO CÁO ĐỀ TÀI OOP**

**Chủ đề: Game 2048**

**Học phần : COMP101701 – Lập trình Hướng đối tượng**

**Nhóm sinh viên thực hiện:** Nhóm 5 anh em

**Họ và tên các thành viên:**

1. Đặng Trần Nhật Hoan – 48.01.103.026
2. Chế Thị Cẩm Loan – 48.01.103.045
3. Nguyễn Quang Lý – 48.01.103.047
4. Nguyễn Gia Thiện – 48.01.103.076
5. Dương Bảo Trân – 48.01.103.081

**Giảng viên:** Ths Lương Trần Ngọc Khiết

Thành phố Hồ Chí Minh, 06 tháng 08 năm 2024

**BẢNG PHÂN CÔNG**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **STT** | **Họ Và Tên** | **MSSV** | **Nhiệm Vụ** | **Phần trăm đóng góp** |
| **1** | Nguyễn Gia Thiện | 48.01.103.076 | Viết code | 100% |
| **2** | Chế Thị Cẩm Loan | 48.01.103.045 | Viết code | 100% |
| **3** | Nguyễn Quang Lý | 48.01.103.047 | Kiểm dữ liệu đầu ra | 100% |
| **4** | Dương Bảo Trân | 48.01.103.081 | Viết code | 100% |
| **5** | Đặng Trần Nhật Hoan | 48.01.103.026 | Viết báo cáo | 100% |

MỤC LỤC

[**GIỚI THIỆU ĐỀ TÀI** 4](#_Toc174199360)

[**MỤC TIÊU ĐỀ TÀI** 4](#_Toc174199361)

[**SƠ ĐỒ OOP** 5](#_Toc174199362)

[**MIÊU TẢ THUẬT TOÁN** 6](#_Toc174199363)

[***Lớp Tile và NumberTile*** 6](#_Toc174199364)

[Lớp Tile: 7](#_Toc174199365)

[Mô tả: 7](#_Toc174199366)

[Phương thức: 7](#_Toc174199367)

[Lớp NumberTile: 8](#_Toc174199368)

[Mô tả: 8](#_Toc174199369)

[Thuộc tính: 8](#_Toc174199370)

[Phương thức: 8](#_Toc174199371)

[Kế thừa và Đa hình: 8](#_Toc174199372)

[Minh họa 9](#_Toc174199373)

[***Lớp Board*** 9](#_Toc174199374)

[Lớp Board: 13](#_Toc174199375)

[Mô tả: 13](#_Toc174199376)

[Thuộc tính: 13](#_Toc174199377)

[Phương thức: 14](#_Toc174199378)

[Các lớp kế thừa Board: 14](#_Toc174199379)

[Tổng hợp: 15](#_Toc174199380)

[Minh họa 15](#_Toc174199381)

[Lớp Game 15](#_Toc174199382)

[*Lớp Game:* 18](#_Toc174199383)

[Mô tả: 18](#_Toc174199384)

[Thuộc tính: 18](#_Toc174199385)

[Phương thức: 18](#_Toc174199386)

[Tổng hợp: 19](#_Toc174199387)

[Minh họa 20](#_Toc174199388)

[**HƯỚNG DẪN CHƠI:** 20](#_Toc174199389)

[**KẾT LUẬN** 22](#_Toc174199390)

[**LINK GITHUB** 23](#_Toc174199391)

# **GIỚI THIỆU ĐỀ TÀI**

Game 2048 là một trò chơi trí tuệ đơn giản nhưng gây nghiện, được phát triển bởi Gabriele Cirulli vào năm 2014. Trò chơi nhanh chóng trở nên phổ biến nhờ vào lối chơi dễ hiểu nhưng đòi hỏi sự tính toán và chiến lược cao.

**ƯU ĐIỂM**

**Tính đơn giản và hấp dẫn**: 2048 có luật chơi đơn giản, dễ hiểu nhưng vẫn đầy thử thách, phù hợp với những sinh viên mới bắt đầu học lập trình.

**Khả năng mở rộng và sáng tạo**: Mặc dù game 2048 có luật chơi cố định, nhưng người lập trình có thể sáng tạo thêm nhiều tính năng mới như giao diện đồ họa, âm thanh, và các chế độ chơi khác nhau.

**Ứng dụng thực tiễn**: Thông qua việc phát triển game 2048, sinh viên có thể áp dụng kiến thức học được vào một dự án cụ thể, từ đó hiểu rõ hơn về quá trình phát triển phần mềm từ ý tưởng đến sản phẩm hoàn thiện.

**Phát triển kỹ năng lập trình**: Viết code cho game 2048 là một cơ hội tuyệt vời để sinh viên thực hành lập trình hướng đối tượng (OOP). OOP là một phương pháp lập trình mạnh mẽ và hiệu quả, giúp tổ chức mã nguồn rõ ràng, dễ bảo trì và mở rộng. Việc áp dụng các khái niệm OOP như lớp, đối tượng, kế thừa và đa hình trong phát triển game sẽ giúp sinh viên hiểu sâu hơn và vận dụng chúng linh hoạt trong các dự án sau này.

#### **MỤC TIÊU ĐỀ TÀI**

Mục tiêu chính của đề tài này là xây dựng một phiên bản đơn giản của game 2048 bằng ngôn ngữ lập trình C++. Qua đó, sinh viên sẽ đạt được những mục tiêu cụ thể sau:

1. **Hiểu rõ cấu trúc và logic của trò chơi**: Nắm vững cách trò chơi hoạt động, các quy tắc và chiến lược cần thiết để chiến thắng.
2. **Phát triển kỹ năng lập trình cơ bản**: Làm quen với các khái niệm lập trình căn bản trong C++ và cách áp dụng chúng vào dự án thực tế.
3. **Quản lý và xử lý dữ liệu**: Hiểu cách quản lý dữ liệu trò chơi, bao gồm việc lưu trữ và cập nhật trạng thái của bảng trò chơi.
4. **Kiểm tra và gỡ lỗi**: Phát triển kỹ năng kiểm tra và gỡ lỗi chương trình để đảm bảo trò chơi hoạt động chính xác và mượt mà.
5. **Áp dụng lập trình hướng đối tượng**: Tổ chức mã nguồn theo mô hình OOP, sử dụng các lớp và đối tượng để quản lý các thành phần của trò chơi như bảng, ô số, và các thao tác di chuyển, từ đó tăng cường khả năng tái sử dụng và bảo trì mã nguồn.

**PHƯƠNG PHÁP THỰC HIỆN**

Để đạt được các mục tiêu đề ra, phương pháp thực hiện sẽ gồm các bước chính sau:

1. **Nghiên cứu và phân tích**: Tìm hiểu về luật chơi 2048 và các thuật toán cần thiết để xử lý logic trò chơi.
2. **Thiết kế và lập trình**: Xây dựng các module chức năng của trò chơi bằng ngôn ngữ C++, bao gồm việc tạo giao diện người dùng đơn giản.
3. **Kiểm tra và hoàn thiện**: Thực hiện kiểm tra, phát hiện và khắc phục các lỗi, tối ưu hóa mã nguồn để trò chơi hoạt động hiệu quả nhất.

# **SƠ ĐỒ OOP**

# **MIÊU TẢ THUẬT TOÁN**

## ***Lớp Tile và NumberTile***

Code của tệp Tile.h:

#ifndef TILE\_H

#define TILE\_H

#include <iostream>

#include <sstream>

using namespace std;

class Tile {

public:

virtual ~Tile() {}

virtual bool isEmpty() const = 0;

virtual string getDisplayValue(bool useLetters) const = 0;

virtual void setValue(int newValue) = 0;

virtual int getValue() const = 0;

virtual bool canCombine(const Tile& other) const = 0;

virtual void combine(Tile& other) = 0;

};

class NumberTile : public Tile {

private:

int value;

string intToString(int value) const {

ostringstream oss;

oss << value;

return oss.str();

}

public:

NumberTile(int val = 0) : value(val) {}

bool isEmpty() const override {

return value == 0;

}

string getDisplayValue(bool useLetters) const override {

if (value == 0) return ".";

if (useLetters) {

switch (value) {

case 2: return "A";

case 4: return "B";

case 8: return "C";

case 16: return "D";

case 32: return "E";

case 64: return "F";

case 128: return "G";

case 256: return "H";

case 512: return "I";

case 1024: return "J";

case 2048: return "K";

default: return "X"; // Sử dụng 'X' thay cho '?'

}

}

else {

return intToString(value);

}

}

void setValue(int newValue) override {

value = newValue;

}

int getValue() const override {

return value;

}

bool canCombine(const Tile& other) const override {

return value == other.getValue() && value != 0;

}

void combine(Tile& other) override {

if (canCombine(other)) {

value \*= 2;

other.setValue(0);

}

}

};

#endif // TILE\_H

### Lớp Tile:

#### Mô tả:

Đại diện cho một ô số hoặc chữ cái trong trò chơi 2048, lưu trữ giá trị hiện tại của ô và quản lý việc kết hợp các ô.

#### Phương thức:

virtual ~Tile() {}: Destructor ảo để đảm bảo việc hủy đối tượng của lớp dẫn xuất được thực hiện đúng.

virtual bool isEmpty() const = 0;: Kiểm tra ô có rỗng hay không.

virtual string getDisplayValue(bool useLetters) const = 0;: Trả về giá trị hiển thị của ô (số hoặc chữ cái).

virtual void setValue(int newValue) = 0;: Đặt giá trị cho ô.

virtual int getValue() const = 0;: Lấy giá trị của ô.

virtual bool canCombine(const Tile& other) const = 0;: Kiểm tra ô hiện tại có thể kết hợp với ô khác hay không.

virtual void combine(Tile& other) = 0;: Thực hiện kết hợp hai ô.

### Lớp NumberTile:

#### Mô tả:

Lớp NumberTile kế thừa từ lớp Tile, đại diện cho các ô chứa số trong trò chơi. Lớp này có thuộc tính int value để lưu trữ giá trị số của ô.

#### Thuộc tính:

int value;: Lưu trữ giá trị số của ô.

#### Phương thức:

string intToString(int value) const: Chuyển giá trị số thành chuỗi.

NumberTile(int val = 0): Hàm dựng khởi tạo giá trị cho ô, mặc định là 0.

bool isEmpty() const override: Trả về true nếu giá trị của ô là 0, ngược lại trả về false.

string getDisplayValue(bool useLetters) const override: Trả về giá trị hiển thị của ô dưới dạng chữ cái hoặc số.

void setValue(int newValue) override: Đặt giá trị mới cho ô.

int getValue() const override: Trả về giá trị hiện tại của ô.

bool canCombine(const Tile& other) const override: Kiểm tra xem ô hiện tại có thể kết hợp với ô khác có cùng giá trị hay không.

void combine(Tile& other) override: Thực hiện kết hợp hai ô, giá trị của ô hiện tại sẽ được nhân đôi và ô kết hợp sẽ bị đặt về giá trị 0.

#### Kế thừa và Đa hình:

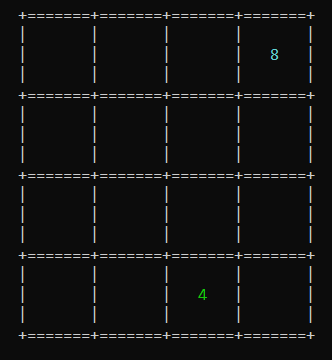
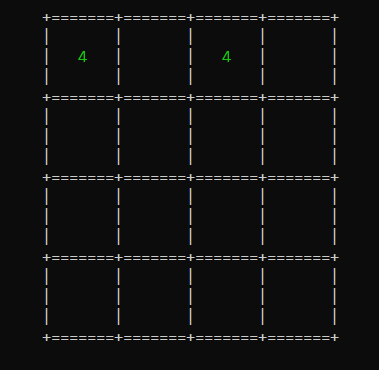
* 1. Kế thừa:

Lớp NumberTile kế thừa từ lớp cơ sở Tile, có nghĩa là nó phải cài đặt lại tất cả các phương thức thuần ảo của lớp Tile.

* 1. Đa hình:

Đa hình cho phép sử dụng các đối tượng NumberTile thông qua con trỏ hoặc tham chiếu đến lớp cơ sở Tile, giúp linh hoạt trong việc xử lý các đối tượng khác nhau kế thừa từ Tile.

#### Minh họa

 Hình 1 Hình 2

Nếu Tile A có giá trị 4 và Tile B có giá trị 4 như ở hình 1, chúng có thể kết hợp lại để tạo thành một Tile C có giá trị 8 như hình 2. Hàm combineWith sẽ thực hiện việc này và Tile B sẽ được đặt lại thành giá trị ban đầu (0 hoặc trống).

## ***Lớp Board***

Code của tệp Board.h:

#ifndef BOARD\_H

#define BOARD\_H

#include <vector>

#include <iostream>

#include <iomanip>

#include <cstdlib>

#include <ctime>

#include <windows.h>

#include "Tile.h"

using namespace std;

class Board {

protected:

int size;

vector<vector<Tile\*>> tiles;

bool useLetters; // Biến để lưu chế độ hiển thị (số hoặc chữ)

static void setColor(int color); // Static function

void rotateBoard();

public:

Board(int s, bool useLetters);

virtual ~Board();

virtual void addRandomTile();

virtual void display() const;

virtual bool moveLeft();

virtual bool moveRight();

virtual bool moveUp();

virtual bool moveDown();

virtual bool canMove() const;

};

class Board4x4 : public Board {

public:

Board4x4(bool useLetters) : Board(4, useLetters) {}

};

class Board5x5 : public Board {

public:

Board5x5(bool useLetters) : Board(5, useLetters) {}

};

class Board6x6 : public Board {

public:

Board6x6(bool useLetters) : Board(6, useLetters) {}

};

#endif // BOARD\_H

Code của tệp Board.cpp:

#include "Board.h"

#include <iostream>

#include <iomanip>

#include <Windows.h> // Thư viện này để sử dụng SetConsoleTextAttribute và GetConsoleScreenBufferInfo

using namespace std;

Board::Board(int s, bool useLetters) : size(s), useLetters(useLetters), tiles(s, vector<Tile\*>(s)) {

srand(static\_cast<unsigned int>(time(0)));

for (int i = 0; i < size; ++i) {

for (int j = 0; j < size; ++j) {

tiles[i][j] = new NumberTile();

}

}

addRandomTile();

addRandomTile();

}

Board::~Board() {

for (int i = 0; i < size; ++i) {

for (int j = 0; j < size; ++j) {

delete tiles[i][j];

}

}

}

void Board::addRandomTile() {

vector<pair<int, int>> emptyPositions;

for (int i = 0; i < size; ++i) {

for (int j = 0; j < size; ++j) {

if (tiles[i][j]->isEmpty()) {

emptyPositions.push\_back({ i, j });

}

}

}

if (!emptyPositions.empty()) {

int idx = rand() % emptyPositions.size();

int value = (rand() % 2 + 1) \* 2;

tiles[emptyPositions[idx].first][emptyPositions[idx].second]->setValue(value);

}

}

void Board::setColor(int color) {

SetConsoleTextAttribute(GetStdHandle(STD\_OUTPUT\_HANDLE), color);

}

void Board::display() const {

system("cls");

// Get console size

CONSOLE\_SCREEN\_BUFFER\_INFO csbi;

GetConsoleScreenBufferInfo(GetStdHandle(STD\_OUTPUT\_HANDLE), &csbi);

int consoleWidth = csbi.srWindow.Right - csbi.srWindow.Left + 1;

// Calculate board width and left padding for centering

const int cellWidth = 7; // Cell width including padding

const int cellHeight = 3; // Cell height including padding

const int boardWidth = size \* (cellWidth + 1) + 1;

const int leftPadding = (consoleWidth - boardWidth) / 2;

// Adjust left padding if it's negative

int leftPaddingAdjusted;

if (leftPadding > 0) {

leftPaddingAdjusted = leftPadding;

}

else {

leftPaddingAdjusted = 0;

}

// Border characters

const string horizontalBorder = "=======";

const char verticalBorder = '|';

cout << "\n" << string(leftPaddingAdjusted, ' ') << "2048 Game\n\n";

// Print top border

cout << string(leftPaddingAdjusted, ' ') << "+";

for (int j = 0; j < size; ++j) {

cout << horizontalBorder << "+";

}

cout << "\n";

for (int i = 0; i < size; ++i) {

for (int k = 0; k < cellHeight; ++k) {

cout << string(leftPaddingAdjusted, ' ') << verticalBorder;

for (int j = 0; j < size; ++j) {

if (k == cellHeight / 2) {

// Print the value in the middle row of the cell

if (tiles[i][j]->isEmpty()) {

setColor(8); // Grey color for empty cells

cout << setw(cellWidth) << " ";

}

else {

switch (tiles[i][j]->getValue()) {

case 2: setColor(9); break; // Blue

case 4: setColor(10); break; // Green

case 8: setColor(11); break; // Cyan

case 16: setColor(12); break; // Red

case 32: setColor(13); break; // Magenta

case 64: setColor(14); break; // Yellow

case 128: setColor(15); break; // White

case 256: setColor(16); break; // Cyan

case 512: setColor(17); break; // Magenta

case 1024: setColor(18); break; // Yellow

case 2048: setColor(19); break; // Red

default: setColor(7); break; // Default color

}

string valueString = tiles[i][j]->getDisplayValue(useLetters);

int valueWidth = valueString.length();

int padding = (cellWidth - valueWidth) / 2;

cout << string(padding, ' ') << valueString << string(cellWidth - padding - valueWidth, ' ');

}

}

else {

// Print empty rows

cout << setw(cellWidth) << " ";

}

setColor(7); // Reset to default color

cout << verticalBorder;

}

cout << "\n";

}

// Print horizontal border

cout << string(leftPaddingAdjusted, ' ') << "+";

for (int j = 0; j < size; ++j) {

cout << horizontalBorder << "+";

}

cout << "\n";

}

}

bool Board::moveLeft() {

bool moved = false;

for (int i = 0; i < size; ++i) {

int lastMergePosition = -1;

for (int j = 1; j < size; ++j) {

if (!tiles[i][j]->isEmpty()) {

int k = j;

while (k > 0 && tiles[i][k - 1]->isEmpty()) {

tiles[i][k - 1]->setValue(tiles[i][k]->getValue());

tiles[i][k]->setValue(0);

k--;

moved = true;

}

if (k > 0 && tiles[i][k - 1]->canCombine(\*tiles[i][k]) && lastMergePosition != k - 1) {

tiles[i][k - 1]->combine(\*tiles[i][k]);

lastMergePosition = k - 1;

moved = true;

}

}

}

}

return moved;

}

void Board::rotateBoard() {

vector<vector<Tile\*>> newTiles(size, vector<Tile\*>(size));

for (int i = 0; i < size; ++i) {

for (int j = 0; j < size; ++j) {

newTiles[j][size - i - 1] = tiles[i][j];

}

}

tiles = newTiles;

}

bool Board::moveRight() {

rotateBoard();

rotateBoard();

bool moved = moveLeft();

rotateBoard();

rotateBoard();

return moved;

}

bool Board::moveUp() {

rotateBoard();

rotateBoard();

rotateBoard();

bool moved = moveLeft();

rotateBoard();

return moved;

}

bool Board::moveDown() {

rotateBoard();

bool moved = moveLeft();

rotateBoard();

rotateBoard();

rotateBoard();

return moved;

}

bool Board::canMove() const {

for (int i = 0; i < size; ++i) {

for (int j = 0; j < size; ++j) {

if (tiles[i][j]->isEmpty()) return true;

if (j < size - 1 && tiles[i][j]->canCombine(\*tiles[i][j + 1])) return true;

if (i < size - 1 && tiles[i][j]->canCombine(\*tiles[i + 1][j])) return true;

}

}

return false;

}

### Lớp Board:

#### Mô tả:

Lớp Board quản lý bảng trò chơi và xử lý các thao tác di chuyển, hiển thị, và kiểm tra trạng thái của trò chơi.

#### Thuộc tính:

int size: Kích thước của bảng (ví dụ: 4x4, 5x5, 6x6).

vector<vector<Tile\*>> tiles: Ma trận 2D đại diện cho các ô trong bảng.

bool useLetters: Chế độ hiển thị (số hoặc chữ).

#### Phương thức:

Board(int s, bool useLetters): Khởi tạo bảng với kích thước s và chế độ hiển thị useLetters.

virtual ~Board(): Giải phóng bộ nhớ được cấp phát cho các ô.

virtual void addRandomTile(): Thêm một ô số ngẫu nhiên vào bảng.

virtual void display() const: Hiển thị bảng trò chơi trên console.

virtual bool moveLeft(), virtual bool moveRight(), virtual bool moveUp(), virtual bool moveDown(): Xử lý các thao tác di chuyển ô.

virtual bool canMove() const: Kiểm tra xem còn có thể di chuyển được không.

static void setColor(int color): Đặt màu sắc hiển thị trên console.

void rotateBoard(): Xoay bảng 90 độ theo chiều kim đồng hồ.

#### Các lớp kế thừa Board:

* Board4x4, Board5x5, Board6x6:

Kế thừa từ lớp Board.

Định nghĩa sẵn kích thước bảng là 4x4, 5x5, và 6x6 tương ứng.

* Hiển thị bảng:

Phương thức display() trong Board hiển thị bảng trên console với các ô được căn giữa và màu sắc khác nhau cho các giá trị ô khác nhau. Phương pháp này sử dụng thông tin về kích thước console và căn chỉnh bảng sao cho nằm giữa màn hình.

* Di chuyển:

Các phương thức di chuyển (moveLeft, moveRight, moveUp, moveDown) đều dựa trên việc xoay bảng và di chuyển sang trái (moveLeft). Việc di chuyển sang các hướng khác được thực hiện bằng cách xoay bảng nhiều lần và sử dụng lại logic di chuyển sang trái.

* Xoay bảng:

Phương thức rotateBoard() xoay bảng 90 độ theo chiều kim đồng hồ để hỗ trợ cho các thao tác di chuyển khác nhau.

* Kiểm tra khả năng di chuyển:

Phương thức canMove() kiểm tra xem có thể di chuyển ô nào không bằng cách kiểm tra các ô trống hoặc các ô có thể kết hợp với nhau theo chiều ngang và dọc.

#### Tổng hợp:

Lớp Board và các lớp con của nó cung cấp các chức năng cần thiết để quản lý và thao tác bảng trò chơi 2048, bao gồm các thao tác di chuyển, xoay, hiển thị và kiểm tra trạng thái của trò chơi. Các phương thức được thiết kế sao cho dễ dàng mở rộng và tái sử dụng logic di chuyển bằng cách xoay bảng.

#### Minh họa

Khi người chơi nhấn phím mũi tên phải, hàm moveRight() sẽ di chuyển tất cả các ô trên bảng sang phải. Nếu hai ô có giá trị giống nhau nằm cạnh nhau, chúng sẽ kết hợp thành một ô mới có giá trị gấp đôi. Các ô còn lại sẽ trượt sang phải để lấp đầy khoảng trống, và một ô mới sẽ được thêm vào vị trí ngẫu nhiên. (Xem hình 1 và hình 2).

## Lớp Game

Code của tệp Game.h:  
#ifndef GAME\_H

#define GAME\_H

#include "Board.h"

class Game {

private:

Board\* board;

int currentSize; // Lưu kích thước hiện tại

bool useLetters; // Lưu chế độ chơi (số hoặc chữ)

void selectBoardSize();

void selectGameMode(); // Thêm khai báo hàm selectGameMode

void restartGame(); // Thêm khai báo hàm restartGame

bool isGameOver() const;

public:

Game();

~Game();

void play();

};

#endif // GAME\_H

Code của tệp Game.cpp:  
#include "Game.h"

#include <conio.h>

#include <iostream>

using namespace std;

Game::Game() : board(nullptr), currentSize(4), useLetters(false) {

selectGameMode();

selectBoardSize();

}

Game::~Game() {

delete board;

}

void Game::selectGameMode() {

cout << "Select game mode by pressing the corresponding key:\n";

cout << "1. Numbers\n";

cout << "2. Letters\n";

cout << "Press the corresponding key (1/2): ";

while (true) {

int ch = \_getch();

switch (ch) {

case '1':

useLetters = false;

return;

case '2':

useLetters = true;

return;

default:

cout << "Invalid choice. Please press 1 or 2: ";

break;

}

}

}

void Game::selectBoardSize() {

if (board != nullptr) {

delete board;

}

cout << "Select board size by pressing the corresponding key:\n";

cout << "1. 4x4\n";

cout << "2. 5x5\n";

cout << "3. 6x6\n";

cout << "Press the corresponding key (1/2/3): ";

while (true) {

int ch = \_getch();

switch (ch) {

case '1':

currentSize = 4;

board = new Board4x4(useLetters);

return;

case '2':

currentSize = 5;

board = new Board5x5(useLetters);

return;

case '3':

currentSize = 6;

board = new Board6x6(useLetters);

return;

default:

cout << "Invalid choice. Please press 1, 2, or 3: ";

break;

}

}

}

bool Game::isGameOver() const {

return !board->canMove();

}

void Game::restartGame() {

delete board; // Xóa bảng cũ

switch (currentSize) {

case 4:

board = new Board4x4(useLetters);

break;

case 5:

board = new Board5x5(useLetters);

break;

case 6:

board = new Board6x6(useLetters);

break;

default:

board = new Board4x4(useLetters); // Default to 4x4 if size is unknown

break;

}

}

void Game::play() {

while (true) {

board->display();

// Check if game is over

if (isGameOver()) {

cout << "\nGame Over!\n";

cout << "Press 'R' to restart or 'Q' to quit.\n";

}

else {

cout << "Press 'M' to return to menu.\n";

}

int ch = \_getch();

bool moved = false;

if (ch == 75) { // Left arrow key

moved = board->moveLeft();

}

else if (ch == 77) { // Right arrow key

moved = board->moveRight();

}

else if (ch == 72) { // Up arrow key

moved = board->moveUp();

}

else if (ch == 80) { // Down arrow key

moved = board->moveDown();

}

else if (ch == 'M' || ch == 'm') { // 'M' to return to menu

selectGameMode(); // Chọn chế độ chơi

selectBoardSize();

}

else if (ch == 'R' || ch == 'r') { // 'R' to restart game

restartGame(); // Khởi động lại trò chơi với kích thước hiện tại

continue; // Bắt đầu trò chơi mới

}

else if (ch == 'Q' || ch == 'q') { // 'Q' to quit game

break;

}

if (moved) {

board->addRandomTile();

}

}

}

### *Lớp Game:*

#### Mô tả:

Quản lý toàn bộ luồng chơi của trò chơi 2048, bao gồm việc lựa chọn chế độ chơi, kích thước bảng, khởi động lại trò chơi, kiểm tra kết thúc trò chơi, và xử lý các sự kiện trong quá trình chơi.

#### Thuộc tính:

Board\* board: Con trỏ đến đối tượng bảng trò chơi (Board).

int currentSize: Lưu kích thước hiện tại của bảng.

bool useLetters: Lưu chế độ chơi hiện tại (số hoặc chữ cái).

#### Phương thức:

void selectBoardSize(): Chọn kích thước bảng trò chơi (4x4, 5x5, 6x6).

void selectGameMode(): Chọn chế độ chơi (số hoặc chữ cái).

void restartGame(): Khởi động lại trò chơi với kích thước và chế độ hiện tại.

bool isGameOver() const: Kiểm tra xem trò chơi đã kết thúc chưa (không còn nước đi).

void play(): Điều khiển luồng chính của trò chơi, xử lý các sự kiện như di chuyển, khởi động lại, và thoát.

* Khởi tạo và Huỷ đối tượng Game:

Game::Game(): Khởi tạo trò chơi, chọn chế độ chơi và kích thước bảng.

Game::~Game(): Giải phóng bộ nhớ cho đối tượng bảng khi trò chơi kết thúc.

* Chọn chế độ chơi (selectGameMode) và kích thước bảng (selectBoardSize):

selectGameMode(): Hiển thị menu cho phép người chơi chọn giữa chế độ chơi số hoặc chữ cái.

selectBoardSize(): Hiển thị menu cho phép người chơi chọn kích thước bảng. Sau khi chọn, tạo đối tượng bảng tương ứng với kích thước đã chọn.

* Khởi động lại trò chơi (restartGame):

Xóa bảng hiện tại và khởi tạo lại bảng với kích thước và chế độ hiện tại.

* Kiểm tra kết thúc trò chơi (isGameOver):

Sử dụng phương thức canMove() của đối tượng bảng để kiểm tra nếu không còn nước đi nào, trả về true nếu trò chơi kết thúc.

* Chơi trò chơi (play):

Vòng lặp chính của trò chơi kiểm tra các thao tác từ người chơi, bao gồm di chuyển các ô, chọn lại chế độ chơi, khởi động lại trò chơi, hoặc thoát trò chơi.

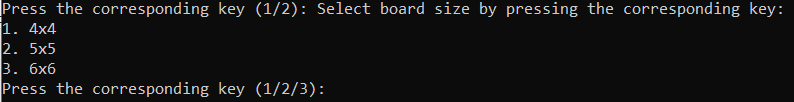
Nếu người chơi chọn chế độ khởi động lại (R), trò chơi sẽ khởi động lại với các thiết lập hiện tại.

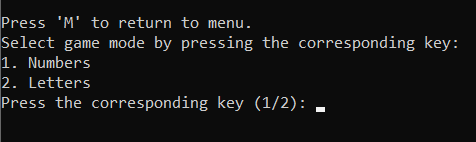
Nếu người chơi chọn chế độ quay lại menu (M), trò chơi sẽ quay lại menu chọn chế độ chơi và kích thước bảng.

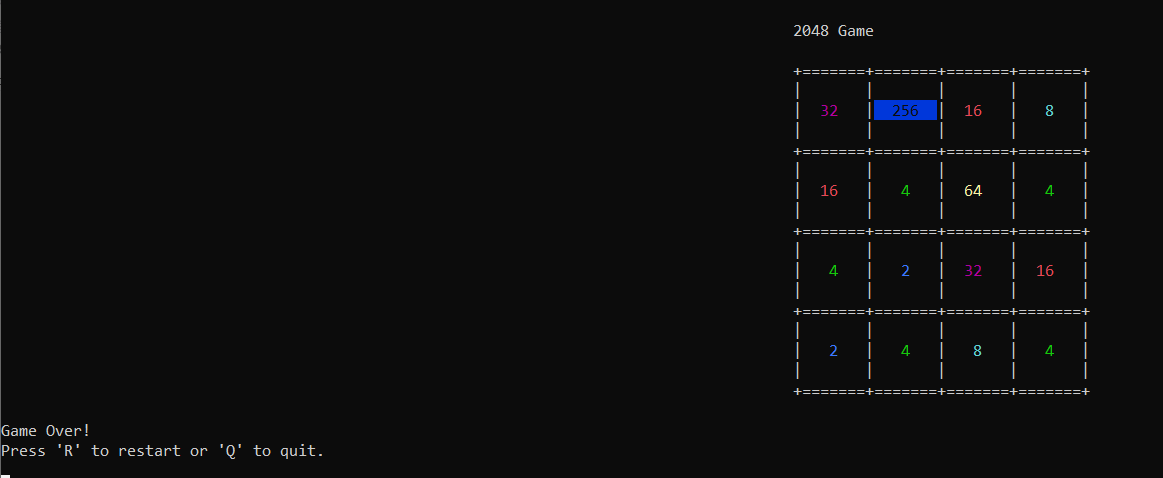
#### Tổng hợp:

Lớp Game quản lý toàn bộ luồng chơi của trò chơi 2048, từ việc khởi tạo bảng, lựa chọn chế độ chơi và kích thước, đến việc xử lý các thao tác trong quá trình chơi và kiểm tra kết thúc trò chơi. Các phương thức trong lớp này được tổ chức chặt chẽ để đảm bảo trò chơi hoạt động mượt mà và người chơi có thể dễ dàng chọn lại chế độ hoặc kích thước bảng khi cần.

#### Minh họa

Hình 3

Hình 4

Hình 5

Hình 6

Khi bắt đầu trò chơi, người chơi sẽ được chọn giữa chế độ chơi với số hoặc chữ cái (selectGameMode) bằng cách nhập 1 hoặc 2 và sau đó chọn kích thước bảng (selectBoardSize) bằng cách nhập giống như chọn chế độ chơi. Trò chơi sẽ bắt đầu và người chơi có thể di chuyển các ô bằng các phím mũi tên như ở minh họa của lớp Tile và lớp Board. Nếu người chơi muốn khởi động lại trò chơi với cùng kích thước bảng, họ có thể nhấn phím 'R'. Nếu muốn quay lại menu để chọn chế độ chơi hoặc kích thước bảng mới, họ có thể nhấn phím 'M'. Khi không còn nước đi nào có thể thực hiện, trò chơi sẽ kết thúc và hiển thị thông báo "Game Over".

# **HƯỚNG DẪN CHƠI:**

Chọn chế độ chơi:

* Khi khởi động trò chơi, bạn sẽ được yêu cầu chọn chế độ chơi:
  + Nhấn 1 để chơi với số.
  + Nhấn 2 để chơi với chữ cái.

Chọn kích thước bảng:

* Sau khi chọn chế độ chơi, bạn sẽ được yêu cầu chọn kích thước bảng:
* Nhấn 1 để chơi với bảng 4x4.
* Nhấn 2 để chơi với bảng 5x5.
* Nhấn 3 để chơi với bảng 6x6.

Cách di chuyển các ô:

* Sử dụng các phím mũi tên để di chuyển các ô trên bảng:
* Mũi tên trái (←) để di chuyển các ô sang trái.
* Mũi tên phải (→) để di chuyển các ô sang phải.
* Mũi tên lên (↑) để di chuyển các ô lên trên.
* Mũi tên xuống (↓) để di chuyển các ô xuống dưới.

Khi hai ô có cùng giá trị nằm cạnh nhau theo hướng di chuyển, chúng sẽ kết hợp lại thành một ô mới với giá trị gấp đôi.

Khởi động lại trò chơi:

* Nếu bạn muốn khởi động lại trò chơi với kích thước bảng hiện tại:
* Nhấn R hoặc r để khởi động lại trò chơi.

Quay lại menu:

* Nếu bạn muốn quay lại menu để chọn lại chế độ chơi hoặc kích thước bảng:
* Nhấn M hoặc m để trở về menu chính.

Thoát khỏi trò chơi:

* Nếu bạn muốn thoát khỏi trò chơi:
* Nhấn Q hoặc q để thoát.

Kết thúc trò chơi:

Trò chơi sẽ kết thúc khi không còn bất kỳ nước đi nào có thể thực hiện được. Khi đó, màn hình sẽ hiển thị thông báo "Game Over". Bạn có thể khởi động lại hoặc thoát khỏi trò chơi bằng cách nhấn các phím tương ứng.

# **KẾT LUẬN**

Trong đồ án này, nhóm chúng tôi đã thành công trong việc xây dựng một phiên bản đơn giản của trò chơi 2048 bằng ngôn ngữ lập trình C++. Qua quá trình thực hiện, chúng tôi đã đạt được các mục tiêu đề ra, từ việc hiểu rõ cấu trúc và logic của trò chơi đến việc áp dụng các khái niệm lập trình hướng đối tượng (OOP) trong việc quản lý.

**Kết quả đạt được:**

* **Xây dựng và tổ chức trò chơi:** Chúng tôi đã xây dựng được một trò chơi 2048 hoạt động mượt mà với các chế độ chơi khác nhau (số và chữ cái), cho phép người chơi lựa chọn kích thước bảng và thực hiện các thao tác di chuyển ô trên bảng.
* **Áp dụng OOP:** Trò chơi được tổ chức theo mô hình lập trình hướng đối tượng với các lớp như Tile, Board, và Game. Các lớp này tương tác với nhau một cách hợp lý, giúp quản lý dữ liệu trò chơi và thực hiện các thao tác cần thiết. Việc sử dụng kế thừa và đa hình đã giúp tăng cường khả năng mở rộng và tái sử dụng mã nguồn.
* **Phát triển kỹ năng lập trình:** Thông qua việc thực hiện đề tài, chúng tôi đã củng cố và phát triển kỹ năng lập trình C++, đặc biệt là trong việc thiết kế và triển khai các thuật toán để xử lý logic trò chơi, quản lý dữ liệu.

Tóm lại, dự án này không chỉ giúp chúng tôi hiểu rõ hơn về trò chơi 2048 mà còn cung cấp một nền tảng vững chắc trong việc áp dụng lập trình hướng đối tượng vào các dự án thực tế. Đây là một cơ hội tuyệt vời để sinh viên thực hành và cải thiện kỹ năng lập trình, đồng thời tạo ra một sản phẩm cụ thể, hữu ích và thú vị. Trò chơi 2048 mà chúng tôi phát triển không chỉ là một bài tập học tập mà còn có thể là bước đầu cho những dự án lập trình lớn hơn và phức tạp hơn trong tương lai.

# **LINK GITHUB**

**https://github.com/Hoandtn02/Demo\_2048Game**