**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** 3](#_Toc174346892)

[ƯU ĐIỂM 3](#_Toc174346893)

[MỤC TIÊU ĐỀ TÀI 3](#_Toc174346894)

[PHƯƠNG PHÁP THỰC HIỆN 4](#_Toc174346895)

[**SƠ ĐỒ OOP** 5](#_Toc174346896)

[**MIÊU TẢ THUẬT TOÁN** 5](#_Toc174346897)

[***Lớp Tile và NumberTile*** 5](#_Toc174346898)

[Lớp Tile: 7](#_Toc174346899)

[Lớp NumberTile: 7](#_Toc174346900)

[***Lớp Board*** 9](#_Toc174346901)

[Lớp Board: 14](#_Toc174346902)

[***Lớp Game*** 16](#_Toc174346903)

[Lớp Game: 19](#_Toc174346904)

[**HƯỚNG DẪN CHƠI:** 20](#_Toc174346905)

[**KẾT LUẬN** 21](#_Toc174346906)

[**LINK GITHUB** 22](#_Toc174346907)

# **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. Trong đề tài này, chúng tôi xây dựng một phiên bản mở rộng của game 2048 bằng ngôn ngữ lập trình C++, với khả năng tùy chọn kích thước bảng và chế độ chơi (số hoặc chữ cái), giúp tăng cường trải nghiệm người dùng và khả năng sáng tạo trong phát triển phần mềm.

## ƯU ĐIỂM

**Tính đơn giản và hấp dẫn:** Trò chơi 2048 có luật chơi đơn giản, dễ hiểu, nhưng đầy thử thách, phù hợp với những sinh viên mới bắt đầu học lập trình. Người chơi chỉ cần sử dụng các phím mũi tên để di chuyển các ô trên bảng, kết hợp các ô có cùng giá trị để tạo ra ô mới với giá trị gấp đôi.

**Khả năng mở rộng và sáng tạo:** Với cấu trúc lập trình hướng đối tượng, trò chơi có thể dễ dàng mở rộng với nhiều tính năng mới. Chẳng hạn, người lập trình có thể thêm các chế độ chơi khác nhau (như chữ cái thay vì số) hoặc thay đổi kích thước bảng từ 4x4 đến 6x6. Điều này không chỉ giúp trò chơi trở nên đa dạng mà còn tạo điều kiện để người lập trình sáng tạo thêm các tính năng mới.

**Ứng dụng thực tiễn:** Phát triển game 2048 là một cơ hội tuyệt vời để sinh viên áp dụng các kiến thức lập trình vào một dự án cụ thể. Từ việc chọn kích thước bảng, chế độ chơi, đến việc xử lý các thao tác di chuyển, sinh viên sẽ hiểu rõ hơn về quá trình phát triển phần mềm từ ý tưởng đến sản phẩm hoàn thiện.

**Phát triển kỹ năng lập trình:** Việc viết code cho game 2048 giúp sinh viên thực hành lập trình hướng đối tượng (OOP). Các khái niệm như lớp, đối tượng, kế thừa, và đa hình được áp dụng triệt để trong trò chơi này. Sử dụng OOP không chỉ giúp tổ chức mã nguồn rõ ràng, dễ bảo trì và mở rộng, mà còn tăng cường khả năng tái sử dụng mã nguồn trong các dự án khác.

## MỤC TIÊU ĐỀ TÀI

Mục tiêu chính của đề tài này là xây dựng một phiên bản nâng cấp 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. Sinh viên sẽ học cách làm việc với các lớp và đối tượng để mô phỏng bảng trò chơi và các ô số trên đó.
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. Sinh viên sẽ học cách tạo và quản lý các lớp như Tile, Board, và Game để tổ chức mã nguồn một cách hiệu quả.
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. Sinh viên sẽ học cách triển khai các thuật toán để xử lý các thao tác di chuyển và kết hợp ô số một cách hiệu quả.
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. Việc kiểm tra tính hợp lệ của các bước di chuyển và tính năng kết thúc trò chơi là những kỹ năng quan trọng cần có trong quá trình phát triển phần mềm.
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. Điều này không chỉ giúp tăng cường khả năng tái sử dụng và bảo trì mã nguồn mà còn giúp sinh viên hiểu sâu hơn về cách áp dụng OOP trong các dự án thực tế.

## PHƯƠNG PHÁP THỰC HIỆN

Để đạt được các mục tiêu đề ra, quá trình thực hiện sẽ bao gồm các bước chính sau:

1. **Nghiên cứu và phân tích**: Tìm hiểu chi tiết về luật chơi 2048 cũng như các thuật toán cần thiết để xử lý logic trò chơi, bao gồm các thao tác di chuyển, kết hợp ô, và kiểm tra điều kiện kết thúc trò chơi.
2. **Thiết kế và lập trình**: Xây dựng các module chức năng của trò chơi bằng ngôn ngữ C++, tập trung vào việc triển khai các lớp như Tile, Board, và Game, đồng thời thiết kế giao diện hiển thị trò chơi trên console để người chơi có thể tương tác một cách dễ dàng.
3. **Kiểm tra và hoàn thiện**: Thực hiện quá trình kiểm tra để phát hiện và khắc phục các lỗi trong chương trình, đồng thời tối ưu hóa mã nguồn nhằm đảm bảo trò chơi hoạt động ổn định và 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:**

Lớp Tile là một lớp trừu tượng đại diện cho một ô số hoặc chữ cái trong trò chơi 2048. Nó lưu trữ giá trị hiện tại của ô và quản lý việc kết hợp các ô. Lớp này chỉ chứa các phương thức thuần ảo, yêu cầu các lớp con triển khai lại tất cả các phương thức đó.

**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:**

NumberTile kế thừa từ lớp Tile, đại diện cho các ô chứa số trong trò chơi. Nó có khả năng hiển thị giá trị dưới dạng số hoặc chữ cái tùy thuộc vào cài đặt.

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:**

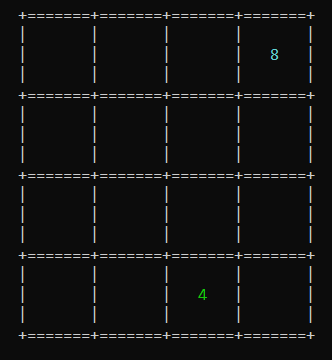
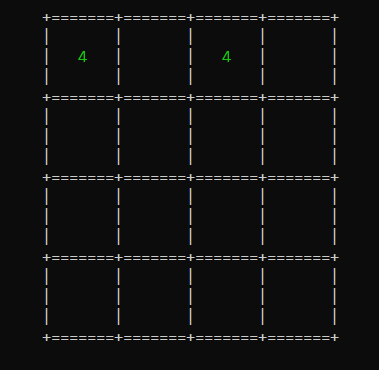
* 1. Kế thừa:

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. Trong trường hợp useLetters là true, kết quả hiển thị sẽ là "C" thay vì số 8. 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;

bool isWinningCondition() const;

vector<pair<int, int>> getEmptyPositions() const;

vector<vector<int>> getTileValues() const;

void setTileValue(int row, int col, int value);

bool has2048() 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");

// Lấy kích thước cửa sổ console

CONSOLE\_SCREEN\_BUFFER\_INFO csbi;

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

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

// Tính toán chiều rộng bảng và khoảng cách trái để căn giữa

const int cellWidth = 7; // Chiều rộng ô bao gồm cả padding

const int cellHeight = 3; // Chiều cao ô bao gồm cả padding

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

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

// Điều chỉnh khoảng cách trái nếu nó âm

int leftPaddingAdjusted;

if (leftPadding > 0) {

leftPaddingAdjusted = leftPadding;

}

else {

leftPaddingAdjusted = 0;

}

// Ký tự viền

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

const char verticalBorder = '|';

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

// In viền trên

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) {

// In giá trị ở hàng giữa của ô

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

setColor(8); // Màu xám cho ô trống

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

}

else {

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

case 2: setColor(9); break; // Xanh dương

case 4: setColor(10); break; // Xanh lá

case 8: setColor(11); break; // Xanh cyan

case 16: setColor(12); break; // Đỏ

case 32: setColor(13); break; // Tím

case 64: setColor(14); break; // Vàng

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

case 256: setColor(16); break; // Xanh cyan

case 512: setColor(17); break; // Tím

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

case 2048: setColor(19); break; // Đỏ

default: setColor(7); break; // Màu mặc định

}

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 {

// In hàng trống

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

}

setColor(7); // Đặt lại màu mặc định

cout << verticalBorder;

}

cout << "\n";

}

// In viền ngang

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

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

cout << horizontalBorder << "+";

}

cout << "\n";

}

// In thông báo "You Win!" nếu điều kiện thắng được thỏa mãn

if (isWinningCondition()) {

cout << "\n" << string(leftPaddingAdjusted, ' ') << "YOU WIN!\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;

}

bool Board::isWinningCondition() const {

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

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

if (tiles[i][j]->getValue() == 2048) {

return true;

}

}

}

return false;

}

vector<pair<int, int>> Board::getEmptyPositions() const {

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 });

}

}

}

return emptyPositions;

}

vector<vector<int>> Board::getTileValues() const {

vector<vector<int>> values(size, vector<int>(size));

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

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

values[i][j] = tiles[i][j]->getValue();

}

}

return values;

}

void Board::setTileValue(int row, int col, int value) {

if (row >= 0 && row < size && col >= 0 && col < size) {

tiles[row][col]->setValue(value);

}

}

bool Board::has2048() const {

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

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

if (tiles[i][j]->getValue() == 2048) {

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. Lớp này sử dụng kế thừa để tạo ra các bảng với kích thước khác nhau như 4x4, 5x5, và 6x6.

Thuộc tính:

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

vector<vector<Tile\*>> tiles: Ma trận lưu trữ các ô trên bảng.

bool useLetters: Hiển thị giá trị của ô dưới dạng số hoặc chữ cái.

**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 cho các đối tượng Tile khi bảng bị hủy.

virtual void addRandomTile(): Thêm một ô có giá trị ngẫu nhiên (2 hoặc 4) vào một vị trí trống ngẫu nhiên trên bảng.

virtual void display() const: Hiển thị bảng trên console. Bảng được căn giữa trên màn hình và các ô có màu sắc tương ứng với giá trị.

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.

bool isWinningCondition() const: Kiểm tra xem có ô nào đạt giá trị 2048 không, từ đó xác định điều kiện thắng.

vector<pair<int, int>> getEmptyPositions() const: Trả về giá trị hiện tại của tất cả các ô trên bảng.

void setTileValue(int row, int col, int value): Đặt giá trị cho một ô cụ thể trên bảng.

bool has2048() const: Kiểm tra xem có ô nào có giá trị 2048 không.

**Các lớp con Board4x4, Board5x5, Board6x6:**

Các lớp này kế thừa từ Board và tạo ra các bảng với kích thước cố định là 4x4, 5x5, và 6x6. Chúng chỉ đơn giản là định nghĩa kích thước bảng cụ thể thông qua hàm khởi tạo.

**Kế thừa và đa hình:**

1. Kế thừa

Lớp Board là lớp cơ sở, được mở rộng bởi các lớp con Board4x4, Board5x5, Board6x6. Kế thừa giúp tái sử dụng mã nguồn và dễ dàng tạo ra các biến thể của bảng với kích thước khác nhau mà không cần viết lại toàn bộ logic.

1. Đa hình

Khi sử dụng các đối tượng của Board (hoặc các lớp con của nó) thông qua con trỏ hoặc tham chiếu đến lớp cơ sở Board, các phương thức như display(), moveLeft() sẽ hoạt động đúng đắn dựa trên lớp con thực tế mà đối tượng thuộc về, nhờ cơ chế đa hình.

**Kết luận:**

Đoạn mã trong Board.h và Board.cpp đã thiết kế một cách hiệu quả các lớp Board và các lớp con để quản lý bảng trò chơi 2048. Nó sử dụng tốt các khái niệm hướng đối tượng như kế thừa và đa hình để tạo ra các bảng với kích thước khác nhau và hỗ trợ hiển thị linh hoạt (số hoặc chữ cái). Các phương thức như moveLeft(), rotateBoard(), và canMove() được cài đặt một cách chi tiết để xử lý logic trò chơi và đảm bảo rằng mọi thao tác di chuyển đều được kiểm tra kỹ lưỡng.

**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

#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();

// Thêm thông báo để quay lại menu

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

// Kiểm tra nếu người chơi thắng

if (board->has2048()) {

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

int ch = \_getch();

if (ch == 'R' || ch == 'r') {

restartGame();

continue;

}

else if (ch == 'Q' || ch == 'q') {

break;

}

// Tiếp tục hiển thị thông báo cho đến khi nhấn 'R' hoặc 'Q'

while (true) {

ch = \_getch();

if (ch == 'R' || ch == 'r') {

restartGame();

break;

}

else if (ch == 'Q' || ch == 'q') {

return; // Thoát trò chơi

}

}

}

// Kiểm tra nếu trò chơi đã kết thúc

if (isGameOver()) {

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

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

int ch = \_getch();

if (ch == 'R' || ch == 'r') {

restartGame();

continue;

}

else if (ch == 'Q' || ch == 'q') {

break;

}

continue;

}

int ch = \_getch();

bool moved = false;

if (ch == 75) { // Phím mũi tên trái

moved = board->moveLeft();

}

else if (ch == 77) { // Phím mũi tên phải

moved = board->moveRight();

}

else if (ch == 72) { // Phím mũi tên lên

moved = board->moveUp();

}

else if (ch == 80) { // Phím mũi tên xuống

moved = board->moveDown();

}

else if (ch == 'M' || ch == 'm') { // 'M' để quay lại menu

selectGameMode();

selectBoardSize();

}

else if (ch == 'R' || ch == 'r') { // 'R' để khởi động lại trò chơi

restartGame();

continue;

}

else if (ch == 'Q' || ch == 'q') { // 'Q' để thoát trò chơi

break;

}

else if (ch == 'C' || ch == 'c') { // 'C' để gian lận và đặt ô thành 2048

auto emptyPositions = board->getEmptyPositions();

if (!emptyPositions.empty()) {

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

board->setTileValue(emptyPositions[idx].first, emptyPositions[idx].second, 2048);

}

}

if (moved) {

board->addRandomTile();

}

}

}

### Lớp Game:

**Mô tả:**

Lớp Game quản lý luồng chính của trò chơi 2048, bao gồm việc chọn chế độ chơi, kích thước bảng, xử lý thao tác của người chơi, và kiểm tra kết thúc trò 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:**

Game(): Khởi tạo giá trị mặc định cho board, currentSize, và useLetters. Gọi các hàm selectGameMode() và selectBoardSize() để người chơi chọn chế độ chơi và kích thước bảng.

~Game(): Giải phóng bộ nhớ đã cấp phát cho đối tượng Board.

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 hay chưa bằng cách kiểm tra khả năng di chuyển các ô trên bảng.

void play():

* Luồng chính của trò chơi, xử lý các thao tác của người chơi, bao gồm di chuyển ô, kiểm tra kết thúc trò chơi, khởi động lại trò chơi, quay lại menu chọn kích thước bảng, và thoát trò chơi.
* Hiển thị bảng và các thông báo tương ứng sau mỗi lượt di chuyển hoặc khi trò chơi kết thúc.
* Có hỗ trợ chức năng gian lận, cho phép đặt ô thành giá trị 2048 bằng cách nhấn phím 'C' (dành cho người lập trình game để kiểm tra dữ liệu đầu ra và các chức năng của trò chơi).

**Tổng hợp:**

* Lớp Game kết hợp với lớp Board để quản lý toàn bộ luồng trò chơi từ khi khởi tạo đến khi kết thúc.
* Cho phép người chơi lựa chọn giữa các chế độ chơi khác nhau và kích thước bảng khác nhau.
* Phương thức play() là trái tim của trò chơi, nơi tất cả các hành động của người chơi được xử lý.

**Minh họa**

Khi người chơi chọn chế độ chơi và kích thước bảng, Game tạo ra bảng tương ứng và bắt đầu trò chơi. Người chơi có thể di chuyển các ô, quay lại menu chọn kích thước bảng, khởi động lại trò chơi, hoặc thoát trò chơi bất cứ lúc nào bằng cách sử dụng các phím tắt tương ứng.

# **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 chọn kích thước bảng:
* Nhấn '1' để chọn bảng 4x4.
* Nhấn '2' để chọn bảng 5x5.
* Nhấn '3' để chọn bảng 6x6.

Cách di chuyển các ô:

* Sử dụng các phím mũi tên để di chuyển các ô trên bảng:
* Mũi tên trái (←): Di chuyển các ô sang trái.
* Mũi tên phải (→): Di chuyển các ô sang phải.
* Mũi tên lên (↑): Di chuyển các ô lên trên.
* Mũi tên xuống (↓): Di chuyển các ô xuống dưới.
* Mục tiêu của bạn là kết hợp các ô có giá trị giống nhau để tạo ra ô có giá trị lớn hơn, cuối cùng đạt được ô có giá trị 2048.

Chức năng bổ sung:

* Nhấn 'R' để khởi động lại trò chơi với cùng kích thước bảng và chế độ chơi hiện tại.
* Nhấn 'M' để quay lại menu và chọn lại kích thước bảng hoặc chế độ chơi khác.
* Nhấn 'Q' để thoát trò chơi.
* Nhấn 'C' để gian lận bằng cách đặt một ô trên bảng thành giá trị 2048.

Kết thúc trò chơi:

* Trò chơi kết thúc khi không còn nước đi nào khả dĩ (không thể di chuyển hoặc kết hợp các ô).
* Khi trò chơi kết thúc, bạn có thể:
  + Nhấn 'R' để bắt đầu lại trò chơi.
  + Nhấn 'Q' để thoát khỏi trò chơi.

# **KẾT LUẬN**

Trong đồ án này, chúng tôi đã thành công trong việc xây dựng một phiên bản trò chơi 2048 đơn giản bằng ngôn ngữ lập trình C++. Trong quá trình thực hiện, nhóm đã hoàn thành các mục tiêu đề ra, bao gồm việc hiểu rõ cấu trúc và logic của trò chơi, cũng như áp dụng các khái niệm lập trình hướng đối tượng (OOP) vào quản lý và triển khai các chức năng của trò chơi.

**Kết quả đạt được:**

* **Xây dựng và tổ chức trò chơi:** Chúng tôi đã phát triển một trò chơi 2048 với các tính năng chính như lựa chọn chế độ chơi (số hoặc chữ cái), chọn kích thước bảng, và thực hiện các thao tác di chuyển ô trên bảng. Trò chơi hoạt động mượt mà, đáp ứng tốt các yêu cầu đề ra.
* **Áp dụng OOP:** Trò chơi được tổ chức theo mô hình lập trình hướng đối tượng với các lớp chính như Tile, Board, và Game. Các lớp này tương tác với nhau thông qua các phương thức được định nghĩa rõ ràng, giúp quản lý dữ liệu trò chơi hiệu quả và thực hiện các thao tác cần thiết. Việc sử dụng kế thừa và đa hình trong thiết kế hệ thống không chỉ giúp tăng cường khả năng mở rộng mà còn tạo điều kiện cho việc tái sử dụng mã nguồn trong các dự án tương lai.
* **Phát triển kỹ năng lập trình:** Quá trình thực hiện dự án đã giúp chúng tôi củng cố và nâng cao kỹ năng lập trình C++, đặc biệt là trong việc thiết kế và triển khai các thuật toán xử lý logic trò chơi, quản lý dữ liệu, và tối ưu hóa hiệu suất của trò chơi.

Tóm lại, đồ án này không chỉ giúp chúng tôi hiểu sâu hơn về cơ chế hoạt động của trò chơi 2048 mà còn mang đến cơ hội thực hành và áp dụng các kiến thức lập trình hướng đối tượng vào một sản phẩm cụ thể. Trò chơi 2048 mà chúng tôi phát triển là minh chứng cho sự kết hợp giữa lý thuyết và thực hành, đồng thời mở ra tiềm năng cho việc thực hiện các dự án lập trình phức tạp hơn trong tương lai. Đây là một bước tiến quan trọng trong quá trình học tập và phát triển kỹ năng lập trình của chúng tôi, góp phần tạo nền tảng vững chắc cho sự nghiệp lập trình sau này.

**LINK GITHUB**

<https://github.com/Hoandtn02/Demo_2048Game>