ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN

——————— o0o ———————

BÁO CÁO ĐỒ ÁN
GAME TETRIS

MÔN HỌC: PHƯƠNG PHÁP LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG

Lớp : 22CLC10

Giảng viên : Bùi Tiến Lên
Trương Tấn Khoa
Lê Thanh Phong

Sinh viên : 22127147 - Đỗ Minh Huy
22127286 - Nguyễn Thanh Nam
22127400 - Thái Hữu Thọ
22127426 - Đinh Duy Triết

Mục lục

Chương 1

PHÂN CÔNG NHIỆM VỤ

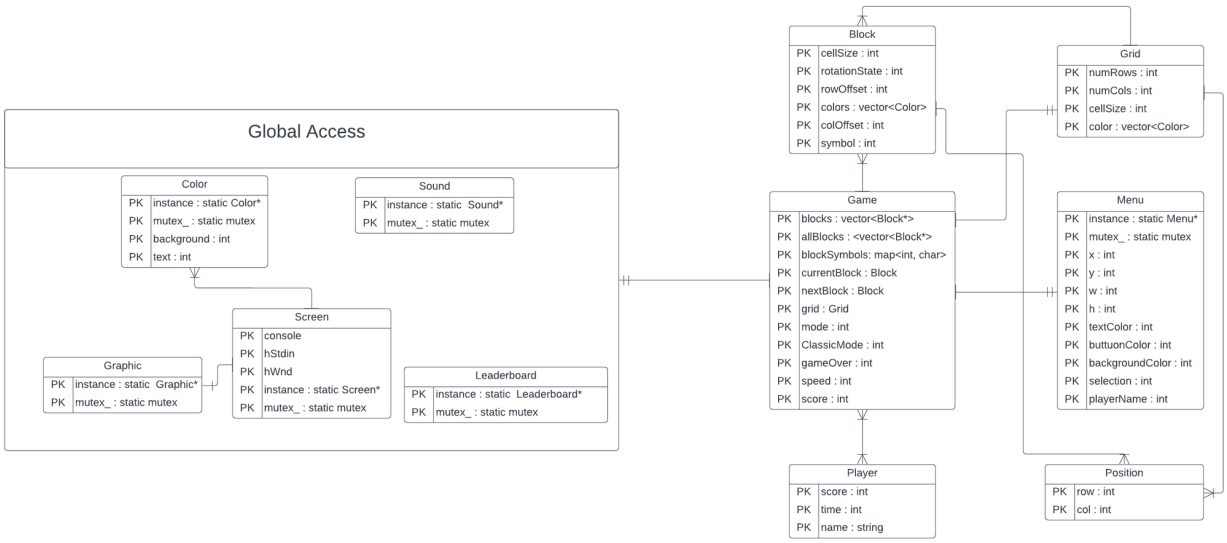
MSSV	Họ và tên	Nhiệm vụ	Mức độ hoàn thành
22127147	Đỗ Minh Huy	Vẽ ERD; Làm các lớp quản lý người chơi	100%
22127286	Nguyễn Thanh Nam	Làm các hàm xử lý logic game	100%
22127400	Thái Hữu Thọ	Viết báo cáo; Thêm nhạc trò chơi	100%
22127426	Đinh Duy Triết	Giao diện đồ hoạ; Kiểm tra báo cáo	100%

Chương 2

HƯỚNG DẪN TỔNG QUAN

- Cung cấp cái nhìn tổng quan ngắn gọn về các bước trong trò chơi.
- Các tiện ích mà người dùng có thể sử dụng trong quá trình trải nghiệm trò chơi.

2.1 SƠ ĐỒ ERD (ENTITY RELATIONSHIP DIAGRAM)



Hình 2.1: Sơ đồ ERD về cấu trúc của chương trình

- Phần mềm gồm có 11 file .cpp và 12 file .h. Mỗi file có 1 chức năng riêng để dễ dàng quản lý và người đọc cũng như người viết có thể dễ dàng chỉnh sửa. Một vài chức năng cơ bản của phần mềm là đồ hoạ, âm thanh, quản lý người chơi, các file logic của game. Âm thanh được lồng ghép 1 cách thú vị làm tăng cảm hứng của người chơi khi trải nghiệm.

2.2 TỔNG QUAN VỀ TRÒ CHƠI

- Khi người dùng bắt đầu trò chơi, tại menu chính sẽ có 5 sự lựa chọn:
  - **Play Game:** Vào trò chơi.
  - **High Scores:** Xem kết quả của 5 người chơi có số điểm cao nhất đã đạt được.
  - **Instruction:** Cách di chuyển, luật chơi và cách tính điểm trong trò chơi.
  - **Credits:** Thông tin về các thành viên trong nhóm dự án.
  - **Quit Game:** Thoát khỏi trò chơi.

- Sau khi người chơi chọn **Play Game**, họ sẽ được chuyển hướng đến giao diện đăng nhập và được yêu cầu nhập tên người dùng để có thể tiếp tục.
- Tiếp theo, người dùng sẽ được đưa đến giao diện chọn chế độ chơi **Choose Mode**. Tại đây, họ sẽ có 2 lựa chọn:
  - **Classic:** Tất cả các mảnh ghép đều sẽ rơi ra ngẫu nhiên và người chơi sẽ bắt đầu với 10 điểm có sẵn.
  - **Modern:** Sau khi rơi ra ngẫu nhiên đủ 7 loại hình dạng của mảnh ghép sẽ tiếp tục đợt tiếp theo và người chơi sẽ bắt đầu với 20 điểm có sẵn.
- Tiếp đó, người chơi sẽ tới với giao diện chọn độ khó **Choose Difficulty** với 4 lựa chọn:
  - **Easy**
  - **Medium**
  - **Hard**
  - **Extreme**
- Với các mức độ khó tăng dần, thời gian xuất hiện và rơi của các mảnh ghép cũng sẽ nhanh hơn. Đặc biệt, ở mức độ khó **Extreme**, các mảnh ghép sau khi rơi xuống sẽ tàng hình, đòi hỏi người chơi phải có khả năng ghi nhớ cao.
- Người dùng có thể sử dụng phím **ESC** để có thể trở về giao diện menu trước đó.

* **HƯỚNG DẪN ĐIỀU KHIỂN & LUẬT CHƠI:**

- Di chuyển sang Phải: d/D/Phím mũi tên phải.
- Di chuyển sang Trái: a/A/Phím mũi tên trái.
- Xoay phải: w/W/Phím mũi tên lên.
- Xoay trái: e/E/Phím Enter.
- Xuống: s/S/Phím mũi tên xuống.
- Rơi: x/X/Phím Space.

- Trong trò chơi Tetris, bạn sẽ đối mặt với một lưới động đầy các hình tetromino khác nhau (I, J, L, O, S, T, Z).
- Mục tiêu là điều khiển và đặt những khối rơi này để tạo ra các dòng ngang đầy đủ.
- Hoàn thành một dòng (hàng ngang) sẽ khiến nó biến mất, kiếm điểm và tạo ra không gian cho nhiều khối hơn. Tiếp tục làm như vậy để xóa càng nhiều dòng càng tốt.
- Nếu để cho các khối chồng lên đến đỉnh có nghĩa là bạn đã thua và trò chơi kết thúc.
- Bạn sẽ dành chiến thắng nếu đạt đủ số điểm quy định (500). Quy tắc tính điểm như sau:
• Mỗi dòng xóa: 10 điểm.
• Mỗi dòng xóa: 50 điểm.

Chương 3

CẤU TRÚC CHƯƠNG TRÌNH

- **Cách sắp xếp các file trong dự án.**
- **Giải thích cách hoạt động của các hàm chức năng.**

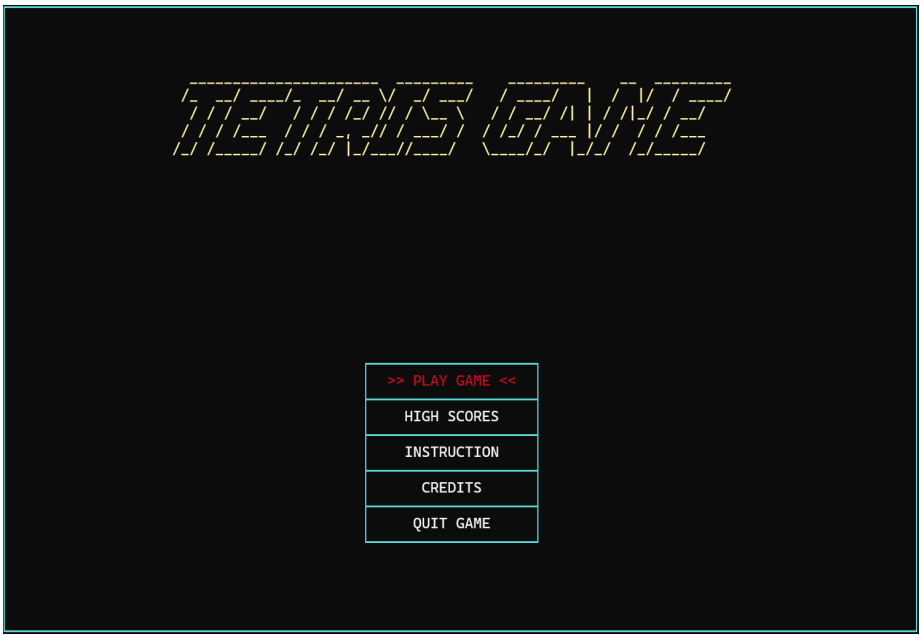
1/ Constants.h:

- File này định nghĩa mã phím sẽ được sử dụng để xử lý sự kiện phím từ người chơi và thực hiện các hành động tương ứng như di chuyển khối Tetris, xoay khối, và thả khối xuống. Các hằng số cung cấp thông tin cho trò chơi như kích thước của màn hình chơi và điểm số tối đa mà người chơi có thể đạt được.

2/ Menu.h và Menu.cpp:

- Tệp này chứa các chức năng tạo giao diện hoàn chỉnh và liên kết chúng với các tệp trò chơi trong dự án này.
- Chúng em đã cài đặt 1 vài hàm tiêu biểu như: **void MainMenu();**

→ Đây là chức năng chính sẽ xử lý toàn bộ giao diện menu chính. Nó bao gồm các chức năng in chữ hoạt hoạ và xử lý các tùy chọn do người dùng lựa chọn trong giao diện này.



Hình 3.1: Bắt đầu trò chơi

* Dưới đây chúng em sẽ giải thích chi tiết hơn về cấu trúc của hàm này:

• void printMainMenu();

→ Đây là chức năng phụ trong hàm **MainMenu()** dùng để định nghĩa một số chức năng khác để in văn bản hoạt hoạ và tô màu hình ảnh.

- Bước tiếp theo là triển khai phương pháp đánh dấu vị trí của tùy chọn khi người dùng sử dụng phím mũi tên để điều hướng trong menu.

→ Chúng em sử dụng vòng lặp để lấy ký tự đầu vào của người dùng và một biến số nguyên có tên là *"selection"* được khởi tạo với giá trị là 1. Khi người dùng nhấn phím 'W' (hoặc phím mũi tên lên trên), biến *"selection"* sẽ được tăng lên 1 và sẽ giảm đi 1 nếu người dùng nhấn phím 'S' (hoặc phím mũi tên xuống dưới). Vì menu chính chỉ có 5 tùy chọn nên khi biến *"selection"* là 0, nó sẽ được gán lại thành 5 và khi là 6, nó được đặt lại về 1.

• void SelectionMenu1();

→ Nhiệm vụ của hàm **selectionMenu1()** là thay đổi màu văn bản của tùy chọn đầu tiên thành màu đỏ khi biến *"selection"* có giá trị là 1, trong khi vẫn giữ màu văn bản của các tùy chọn khác là màu trắng. Tương tự, nó sẽ thay đổi màu văn bản của tùy chọn thứ hai, thứ ba hoặc thứ tư thành màu đỏ khi biến *"selection"* có giá trị tương ứng là 2, 3 hoặc 4.

→ Điều này mô phỏng một con trỏ di chuyển dọc theo các tùy chọn menu dựa trên các phím định hướng mà người dùng nhập vào.

→ Dựa vào giá trị của biến lựa chọn, câu lệnh switch-case sẽ được sử dụng để xác định tùy chọn nào người dùng đã chọn khi nhấn phím "Enter". Mỗi trường hợp trong câu lệnh switch chứa các lệnh tương ứng để thực thi tùy chọn đã chọn của chương trình.

3/ Block.h và Block.cpp:

- Các file này định nghĩa lớp "Block" trong trò chơi, chứa các thuộc tính của 1 khối và xác định cách di chuyển, cách xoay của 1 khối bất kỳ.

• void rightRotate(): Quay theo chiều kim đồng hồ.

• void leftRotate(): Quay ngược chiều kim đồng hồ.

- Ngoài ra còn có 7 lớp "IBlock", "JBlock", "LBlock", "OBlock", "SBlock", "TBlock", "ZBlock" kế thừa từ lớp "Block", chứa thuộc tính khi xoay, id, tự động và cách xuất hiện.

4/ Color.h và Color.cpp:

- Các file này định nghĩa lớp "Color"trong trò chơi. Lớp này quản lý sắc màu của văn bản và nền trong giao diện trò chơi. Nó bao gồm các hàm tạo và phương thức để thiết lập và xuất sắc màu, cũng như các phương thức để thiết lập màu sắc trong giao diện console.

• Color::getInstance()->consoleColor: Thiết lập màu nền.

• Color::getInstance()->consoleTextColor: Thiết lập màu chữ.

5/ Graphic.h và Graphic.cpp:

- Các file này định nghĩa lớp "Graphic" trong trò chơi. Lớp này quản lý đồ họa trên console. Các hàm chủ yếu làm những công việc như đọc tệp tin và hiển thị nội dung với màu nền và màu chữ khác nhau. Mẫu Singleton được sử dụng để đảm bảo chỉ có một đối tượng Graphic, và có các hàm để lấy và xóa đối tượng đó.

• void artAtPosition(string fileName, int backgroundColor, int textColor, int x, int y)

→ Đọc nội dung từ tệp tin và hiển thị nó trên màn hình console tại một vị trí cụ thể với màu nền và màu chữ được xác định. Mô tả chi tiết về các tham số của hàm, ví dụ như fileName, x, y, backgroundColor, và textColor. Giải thích ý nghĩa của mỗi tham số và cách chúng ảnh hưởng đến hành vi của hàm. Mô tả cách hàm sử dụng một đối tượng

ifstream để mở tệp tin và sử dụng vòng lặp while để đọc từng dòng của tệp tin. Cho biết làm thế nào vòng lặp diễn ra và làm thế nào dòng văn bản được xử lý.

Hình 3.2: Đồ hoạ trong trò chơi

6/ Grid.h và Grid.cpp:

- Các file này định nghĩa lớp "Grid" trong trò chơi. Lớp này chứa các hàm kiểm tra các hàng và xử lý các hàng có đủ khối và tính điểm.

• **drawGrid():** Phương thức này được sử dụng để vẽ các ô trống trên màn hình.

• **Screen::getInstance()->resetConsoleColor():** Phương thức này sẽ đặt lại màu của console bằng cách gọi phương thức resetConsoleColor() từ lớp Screen.

* for (int i = 0; i < numRows; i++): Vòng lặp này sẽ lặp qua từng hàng của lưới.

* for (int j = 0; j < numCols; j++): Vòng lặp này sẽ lặp qua từng cột của lưới.

• **Screen::getInstance()->moveCursor(j * cellSize + 10, i * cellSize + 10):** Phương thức *moveCursor()* của lớp Screen được gọi để di chuyển con trỏ console đến vị trí cụ thể trên màn hình. Vị trí này được tính toán dựa trên kích thước của mỗi ô trong lưới (cellSize) và chỉ số của hàng và cột hiện tại.

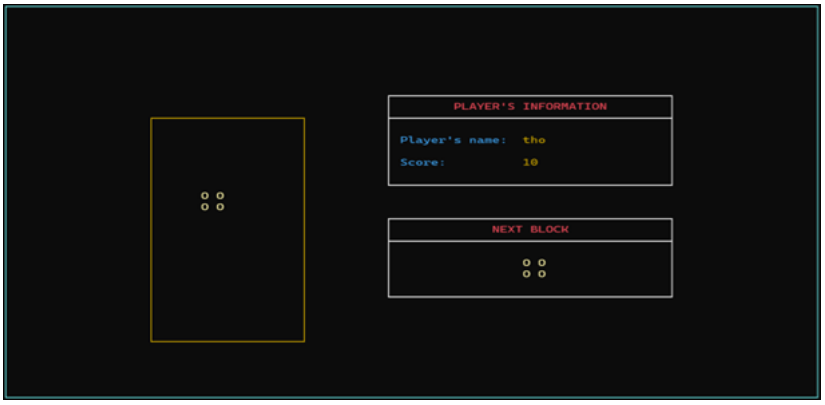
- Trong tetris, lưới này có thể được sử dụng để hiển thị các khối rơi xuống. Mỗi ô trong lưới tương ứng với một vị trí có thể trên bảng điểm, và dấu chấm được in ra có thể đại diện cho một ô trống. Khi một khối rơi xuống và điền vào một ô, dấu chấm có thể được thay thế bằng một ký tự hoặc một màu khác để biểu thị khối đó. - Hàm **moveRowDown** trong class Grid được sử dụng để di chuyển một dòng xuống dưới, để lấp đầy những ô trống sau khi một dòng đã được xóa.

* for (int column = 0; column < numCols; column++): Duyệt qua các cột của dòng cần di chuyển. Vòng lặp for duyệt qua từng ô trong dòng cần di chuyển (dòng row), từ cột 0 đến numCols - 1.

• **grid[row + numRows][column] = grid[row][column]:** Di chuyển giá trị từ dòng hiện tại xuống dòng mới. Mỗi ô trong dòng row được di chuyển xuống dưới numRows hàng. Cụ thể, giá trị của ô tại dòng row và cột column được sao chép xuống dòng row + numRows và cột column.

• **grid[row][column] = 0:** Đặt giá trị của ô cũ về 0: Sau khi giá trị đã được sao chép, giá trị của ô tại dòng row và cột column được đặt về 0. Điều này giúp giữ cho dòng row trở nên trống sau khi đã di chuyển giá trị xuống dưới.

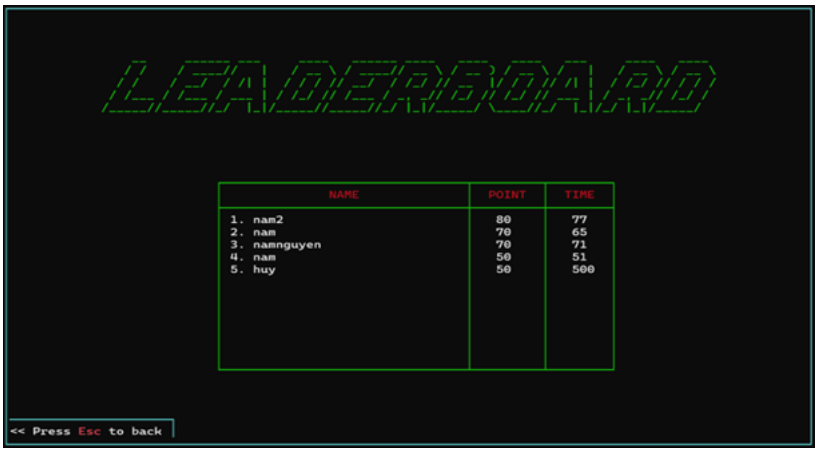
- Cuối cùng, khi hàm moveRowDown được gọi, nó sẽ di chuyển toàn bộ dòng xuống dưới, để lấp đầy các ô trống do việc xóa dòng ở trên.



Hình 3.3: Giao diện màn chơi

7/ Leaderboard.h và Leaderboard.cpp:

- Các file này định nghĩa lớp "Leaderboard" trong trò chơi. Lớp này quản lý điểm của các người chơi và lập ra bảng điểm và so sánh điểm của các người chơi.



Hình 3.4: Bảng điểm

8/ Player.h và Player.cpp:

- Các file này định nghĩa lớp "Player" trong trò chơi. Lớp này chứa các thuộc tính như tên, điểm số và thời gian của người chơi, cùng với các phương thức để truy cập và thiết lập các giá trị này.

9/ Position.h và Position.cpp:

- Tệp này định nghĩa lớp "Vị trí"trong trò chơi Tetris chứa các thuộc tính (x, y). Nó cung cấp các phương thức để xuất ra và thiết lập giá trị của vị trí.

10/ Game.h và Game.cpp:

- Các file này định nghĩa lớp "Game" trong trò chơi, thực hiện các chức năng quan trọng của 1 trò chơi. Tạo trò chơi, xử lý đầu vào của người chơi các phím điều khiển và xoay khối. Tạo khối, xử lý hoạt động của khối (chuyển, xoay và xử lý va chạm), tính năng của người chơi, cài đặt chế độ và tốc độ trò chơi.

• bool Trò chơi::verticalCollision()

→ Mục tiêu của hàm này là xử lý va chạm theo chiều dọc (va chạm dọc) giữa khối Tetris hiện tại và lưới (lưới). Nếu có ít nhất một ô của khối Tetris nằm ngoài lưới, hàm sẽ trả về

true, ngược lại sẽ trả về false. Dưới đây là giải thích chi tiết về cách hoạt động của chức năng này:
• **vector<Position> blockCells = currentBlock.getCellsPositions()**
→ Sử dụng hàm getCellsPositions() của đối tượng currentBlock để lấy vectơ chứa vị trí của tất cả các ô trong khối Tetris hiện tại.
* for (Position cell : blockCells): Sử dụng vòng lặp để duyệt từng ô trong khối Tetris.

- Kiểm tra ô xem có nằm ngoài mạng hay không:

• **if (grid.isCellOutside(cell.getRow(), cell.getCol())) return true;** : Sử dụng lưới đối tượng để kiểm tra vị trí xem của ô (được xác định bởi cell.getRow() và cell.getCol()) có nằm ngoài mạng không. Nếu có, hàm sẽ trả về true, có xung đột theo chiều dọc. Vòng lặp kết thúc mặc dù không có xung đột.
- Tương tự với hàm bool **Game::horizontalCollision()**: kiểm tra xung đột theo chiều ngang.
• **vector<Position> blockCells = currentBlock.getCellsPositions()**
- Hàm **currentBlock.getCellsPositions()** trả về một vectơ chứa tất cả các ô (cell) của khối hiện tại, được biểu thị bằng đối tượng Position (vị trí). Đặt giá trị của các ô hiện tại vào lưới (grid).

11/ Screen.h và Screen.cpp:

- Tệp này định nghĩa lớp "Screen" trong trò chơi, chứa các hàm để điều chỉnh và vẽ các phần tử trên bảng điều khiển màn hình. Các phương thức bao gồm thiết lập cửa sổ kích thước, vô hiệu hóa việc thay đổi kích thước cửa sổ, hiển thị/ẩn với đèn nháy con trỏ và xóa nội dung màn hình bảng điều khiển.

12/ Thư mục static: Chứa các văn bản hiển thị trong trò chơi cũng như các ascii mã hóa hình ảnh được sử dụng.

13/ Thư mục sounds: Chứa các đoạn âm thanh ngắn được sử dụng trong trò chơi.

14/ main.cpp:

- Đây là phần chính của chương trình. Nó sẽ chạy toàn bộ chương trình.

* *Các Thư viện đặc biệt được sử dụng trong Tetris gồm:*

- **<Windows.h>** là một tệp tiêu đề trong hệ điều hành Windows. Nó cung cấp các khai báo và định nghĩa liên quan đến API của Windows, cho phép ứng dụng tương tác với các tính năng hệ thống như giao diện đồ họa, tệp tin, mạng, âm thanh và nhiều tính năng khác.

- **<mutex>** ngăn chặn các xung đột giữa các luồng khi chúng cố gắng truy cập vào và chỉnh sửa dữ liệu được chia sẻ. Nếu không sử dụng mutex trong môi trường đa luồng, có thể xảy ra các vấn đề liên quan đến đồng thời và không an toàn.

- **<future>** thường được sử dụng để quản lý và kiểm soát các tác vụ chạy không đồng bộ, cho phép lập trình viên kiểm soát và truy xuất kết quả của chúng một cách an toàn. Cụ thể, <future> cung cấp các lớp như std::future và hàm như std::async.

- **<memory>** cung cấp các con trỏ thông minh như std::unique_ptr để quản lý bộ nhớ một cách an toàn và tự động. std::unique_ptr được sử dụng để quản lý đối tượng của lớp Trò chơi, Trình phát và Menu, giúp giảm nguy cơ rò rỉ bộ nhớ và quản lý kết quả tài nguyên.

- **<chrono>** được sử dụng để kiểm soát và đo lường thời gian, giúp làm cho game chạy mượt mà và dễ dàng điều chỉnh các tham số như tốc độ, thời gian thực hiện các hành động,

và các khía cạnh thời gian khác trong game Tetris.

- **<thread>** được sử dụng để tạo và quản lý luồng (threads) trong chương trình. std::thread được sử dụng để thực hiện các công việc trong các luồng riêng biệt, không làm tạm dừng hoặc chặn luồng chính (main thread). Ngoài ra, std::async và std::future được sử dụng để bắt đầu một nhiệm vụ trong một luồng riêng biệt, trong khi std::this_thread::sleep_for giúp kiểm soát tốc độ cập nhật của trò chơi bằng cách chờ một khoảng thời gian cụ thể. Sử dụng luồng giúp tăng hiệu suất và đồng thời trong các tác vụ đa nhiệm.

- **<mmsystem.h>** là một thư viện trong Windows API được sử dụng để làm việc với hệ thống âm thanh và đa phương tiện.

Hình 3.5: Giao diện khi thoát trò chơi

Chương 4

CÁCH CHẠY CHƯƠNG TRÌNH

- Mã nguồn sử dụng thư viện <Windows.h> sẽ chỉ tương thích với các máy sử dụng hệ điều hành Windows.
- Trình cài đặt môi trường: Windows 11.
- Mã nguồn xem tại: github

4.1 SỬ DỤNG VISUAL STUDIO

- Mở tệp chọn thư mục TetrisGame.sln sau đó nhấn Ctrl + F5 để sử dụng chương trình.

4.2 CHẠY TRỰC TIẾP

- Tại thư mục chứa tệp chọn chương trình mã nguồn main.exe để sử dụng chương trình ngay lập tức.

Chương 5

CÔNG CỤ HỖ TRỢ & TÀI LIỆU THAM KHẢO

5.1 CÔNG CỤ HỖ TRỢ

- Không gian làm việc: https://github.com/mareZ-noob/Tetris-Game-OOP
- Chuyển đổi hình ảnh thành kí tự ASCII: https://www.patorjk.com/
- Trình biên dịch (IDE): Visual Studio.

5.2 TÀI LIỆU THAM KHẢO

- How do you clear the console screen in C?
- WASD Key movement c++
- C++ Binary File I/O