TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
KHOA CÔNG NGHỆ THÔNG TIN

ĐỒ ÁN MÔN LẬP TRÌNH HƯỚNG ĐỐI TƯỢNG

Môn: Phương pháp lập trình hướng đối tượng.

Nhóm: 3

24127250	Phan Quang Tiến
24127446	Hà Như Lương
24127529	Nguyễn Chí Tài
24127545	Nguyễn Ngọc Thiên

Giảng viên hướng dẫn: Bùi Tiến Lên

1. Giới thiệu

- **Lý do chọn đề tài:** Nhóm em chọn đề tài game Tetris vì game khá vui, phù hợp và đơn giản cho những người bắt đầu học lập trình hướng đối tượng để thiết kế game.
- **Mục tiêu của đồ án:**
  + Để hiểu được nguyên tắc của **lập trình hướng đối tượng**, hiểu đươc việc xác định từng **đối tượng** và các nguyên tắc: **đóng gói, kế thừa, trừu tượng, đa hình** cùng với việc áp dụng vào để thiết kế game thay vì làm những bài tập quản lý nhỏ sẽ giúp nhóm em có thêm kiến thức, kinh nghiệm và trải nghiệm nhiều hơn.
  + Thông qua làm việc nhóm bằng đồ án mà nhóm em đã có nhiều kiến thức hơn thông qua việc chia sẻ, học hỏi cùng nhau, giúp đỡ nhau khi gặp khó khăn. Qua đó. nhóm em đã nhận ra được giá trị của việc làm đồ án bằng làm việc nhóm.
- **Công nghệ sử dụng:** C++, raylib, CMake.
- **Tính năng chính của game:**
  - Hiển thị lưới **20x10**, mỗi ô có kích thước 50px.
  - Sinh ngẫu nhiên **7 loại khối Tetris (I, J, L, O, S, T, Z).**
  - Cho phép **xoay, di chuyển trái/phải, thả nhanh** các khối.
  - **Phát hiện va chạm** và **cố định khối** khi chạm đáy hoặc khối khác.
  - **Xóa hàng đầy** và **dồn các hàng trên xuống**, cộng điểm tương ứng.
  - **Tính điểm và level**: điểm tăng khi xóa hàng, tốc độ rơi tăng theo level.
  - **Game Over** khi khối mới sinh ra không còn chỗ trống.
  - Có **âm thanh** khi xóa hàng và khi Game Over.
  - Giao diện đồ họa sử dụng **raylib**, hiển thị score, level và thông báo trạng thái.

2. Phân tích và Thiết kế

2.1. Phân tích yêu cầu

2.1.1. Yêu câu chức năng:

- **Khởi tạo trò chơi:** sinh lưới 20x10, khởi động âm thanh và giao diện.
- **Sinh khối ngẫu nhiên**: tạo 1 trong 7 loại khối Tetris (I, J, L, O, S, T, Z).
- **Điều khiển khối**:
    - Di chuyển sang trái/phải.
    - Xoay khối (theo chiều kim đồng hồ).
    - Thả nhanh xuống (soft drop).
- **Xử lý va chạm**: phát hiện khối chạm đáy hoặc khối khác.
- **Cố định khối**: khi khối chạm đáy thì "khóa" vào lưới.
- **Xóa hàng đầy**: khi một hàng đầy, tự động xóa và dồn các hàng trên xuống.
- **Tính điểm và level**: cộng điểm theo số hàng xóa, tăng level để tăng tốc độ rơi.
- **Game over**: khi khối mới không còn chỗ để sinh ra
- **Âm thanh và giao diện**: phát hiệu ứng khi xóa hàng, khi Game Over, hiển thị score, level, thông báo.

2.1.2. Yêu cầu phi chức năng

- **Hiệu năng**: game chạy mượt ở 60 FPS, không bị giật lag.
- **Tính dễ dùng**: điều khiển đơn giản bằng bàn phím (↑, ↓, ←, →, R).
- **Tính mở rộng**: dễ dàng bổ sung tính năng mới (hard drop, bảng xếp hạng, skin).
- **Đa nền tảng**: có thể build trên Windows và Linux nhờ CMake + raylib.

- **Thẩm mỹ**: giao diện đơn giản, rõ ràng, màu sắc trực quan.

2.2.Thiết kế hệ thống

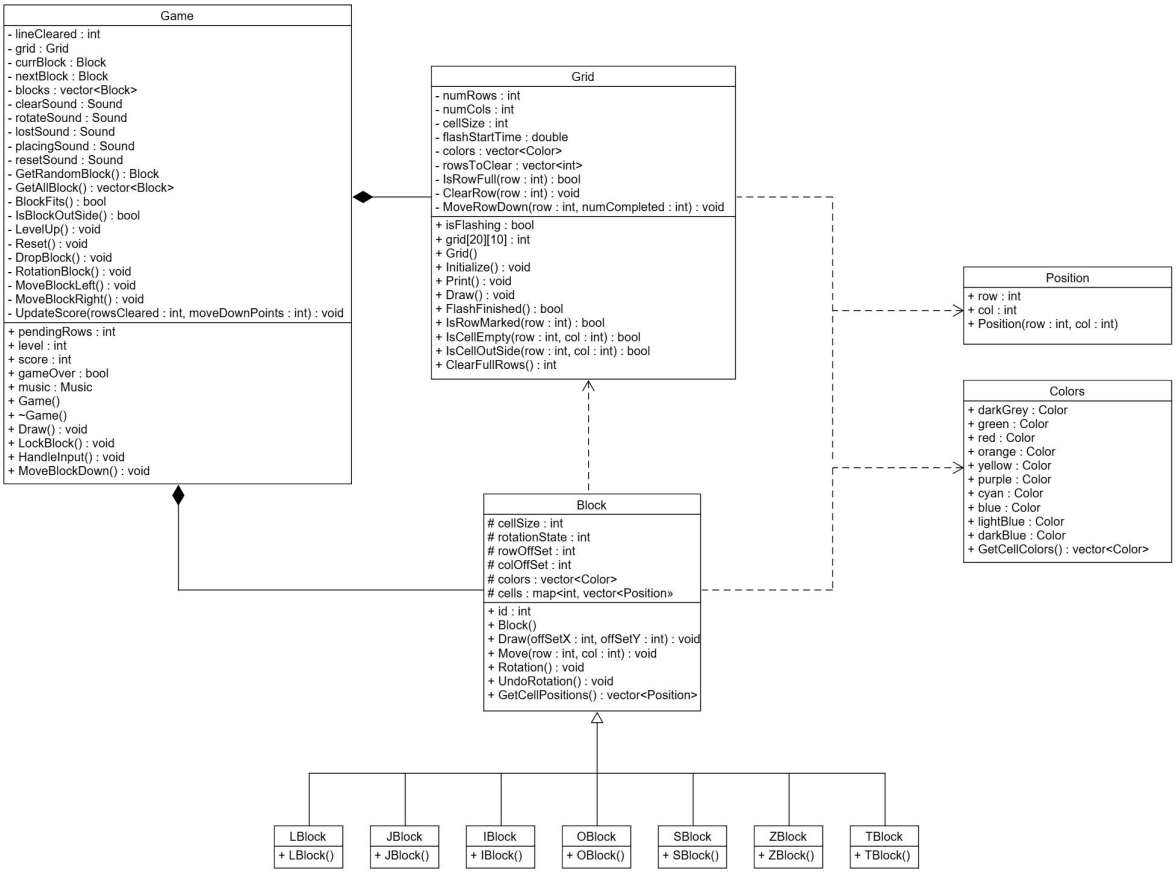
- Sơ đồ kiến trúc tổng quan.

  Hệ thống được tổ chức theo mô hình hướng đối tượng, chia thành nhiều lớp đảm nhận các trách nhiệm khác nhau. Mỗi lớp có vai trò rõ ràng, giúp chương trình dễ bảo trì và mở rộng.

  - **Lớp Grid:**
    - Đại diện cho lưới 20x10 của trò chơi.
    - Quản lý trạng thái các ô (trống = 0, có block ≠ 0).
    - Cung cấp các hàm xử lý: kiểm tra va chạm, xoá hàng đầy, dồn khối xuống.
  - **Lớp Block** (lớp cơ sở) và các lớp kế thừa (**IBlock, JBlock, LBlock, OBlock, SBlock, TBlock, ZBlock**).
    - Mô tả hình dạng, màu sắc, và các trạng thái xoay của từng khối.
    - Cung cấp hành vi: xoay, di chuyển trái/phải, rơi xuống.
  - **Lớp Game:**
    - Đóng vai trò **Facade**, quản lý toàn bộ trò chơi.
    - Điều phối giữa **Grid, Block,** và các tài nguyên khác (âm thanh, font).
    - Xử lý input từ bàn phím, cập nhật logic, quản lý score/level, phát hiện Game Over.
    - Chứa vòng lặp chính (Game Loop).
  - **Lớp Color:**
    - Định nghĩa màu sắc cho các khối.
    - Giúp hiển thị khối trực quan hơn.
  - **Lớp Position:**
    - **Đại diện cho tọa độ một ô trong lưới (row, col).**

- **Hỗ trợ việc tính toán vị trí block trong Grid.**

- Sơ đồ UML Class (Game, Grid, Block, các khối con, …)



3. Cài đặt

- **Cấu trúc thư mục project:**

  include/ # Chứa các file header (.h)

  src/ # Chứa các file mã nguồn (.cpp)

  lib/ # Thư viện raylib đã biên dịch sẵn

  resource/ # Âm thanh, font chữ

  CMakeLists.txt

- **Các file chính:**
  - **main.cpp**: điểm vào chương trình, khởi tạo game và vòng lặp chính.
  - **game.h/ game.cpp:** lớp **Game** điều phối toàn bộ trò chơi (Facade Pattern).

- **grid.h/ grid.cpp:** quản lý lưới 20x10, kiểm tra va chạm, xóa hàng.
- **block.h/ block.cpp:** lớp cơ sở **Block** và các lớp con cho 7 khối Tetris.
- **color.h/ color.cpp:** định nghĩa và quản lý màu sắc các ô.
- **position.h/ position.cpp:** hỗ trợ lưu trữ và xử lý tọa độ (row, col).

- **Luồng xử lý chính (Game loop)**
  - **Input:** đọc phím (↑, ↓, ←, →, R).
  - **Update:**
    + Di chuyển/ xoay khối.
    + Kiểm tra va chạm.
    + Cố định khối khi chạm đáy.
    + Xóa hàng đầy, cộng điểm, tăng level.
    + Sinh khối mới.
  - **Render:**
    + Vẽ lưới và các khối cố định.
    + Vẽ khối đang rơi.
    + Vẽ thông tin Score, Level, Game Over.

- **Các hàm quan trọng:**
  - **Rotate():** xoay khối theo chiều kim đồng hồ, thay đổi trạng thái **state.**
  - **MoveLeft()/ MoveRight()/ SoftDrop():** dịch khối theo hướng tương ứng.
  - **IsBlockOutside():** kiểm tra khối có vượt ngoài lưới không.
  - **IsBlockColliding():** kiểm tra khối có đè lên ô đã có block không.
  - **ClearFullRows():** xoá hàng đầy và dồn các hàng phía trên xuống.
  - **GetRandomBlock():** sinh khối ngẫu nhiên từ 7 loại.
  - **UpdateScore():** cộng điểm dựa vào số hàng xóa, tăng level khi đạt mốc.
  - **PlaySound():** phát âm thanh khi xóa hàng hoặc Game Over.

4. Nguyên lí OOP và Design Patterns

4.1. Nguyên lí OOP áp dụng.

- **Encapsulation (Đóng gói)**
  - Lớp **Grid** đóng gói dữ liệu lưới 20x10 và các hàm xử lý (**IsCellEmpty**, **ClearFullRows**).
  - Lớp **Block** và các lớp con quản lý thông tin khối (tọa độ, trạng thái xoay) và hành vi (di chuyển, xoay).
  - Lớp **Game** bao bọc toàn bộ logic: input, update, render, điểm số, âm thanh.
- **Inheritance (Kế thừa)**
  - 7 loại tetromino (**IBlock, JBlock, LBlock, OBlock, SBlock, TBlock, ZBlock**) kế thừa từ lớp cơ sở **Block**.
- **Polymorphism (Đa hình)**
  - Các lớp con override hàm khởi tạo hình dạng khối, nhưng có thể sử dụng chung hàm **Rotate(), Draw(), Move()**.
- **Abstraction (Trừu tượng)**
  - Lớp **Block** được thiết kế như một lớp trừu tượng khái quát, các lớp con chỉ triển khai đặc thù cho từng loại khối.

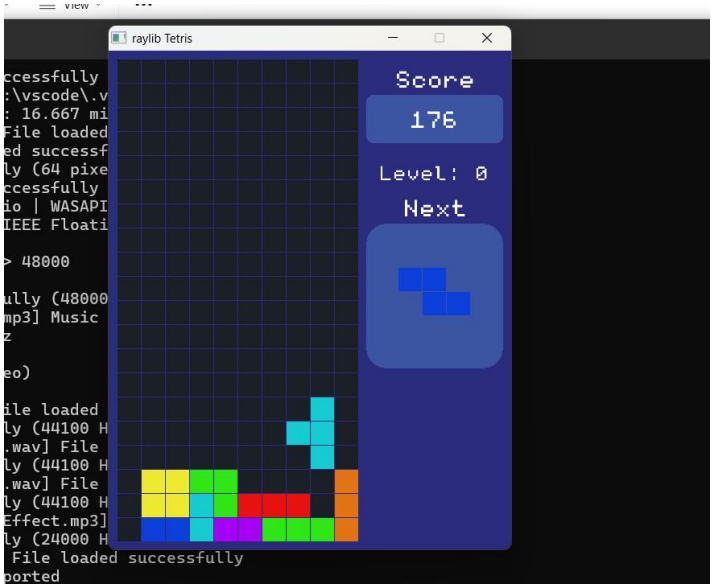
4.2. Design Patterns sử dụng

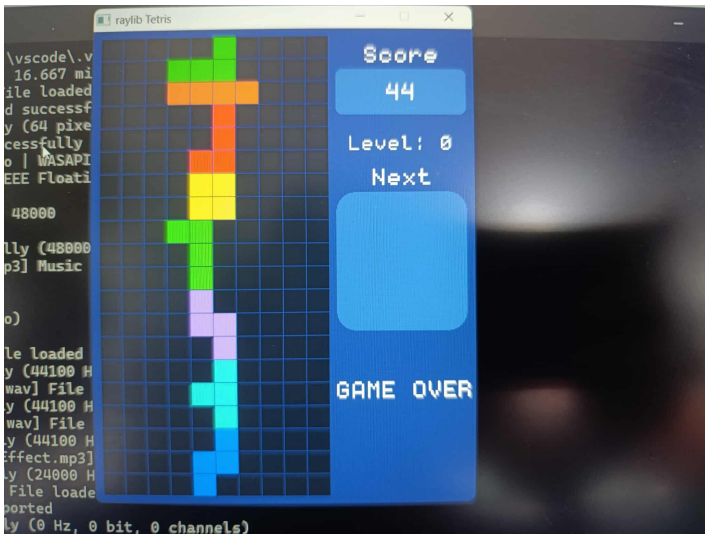
- **Prototype:**
  - Ý tưởng: mỗi loại khối được xem như một "mẫu" có sẵn về hình dạng.
  - Khi tạo khối mới, game dựa trên "mẫu" và nhân bản ra một instance mới.
  - Ví dụ: trong Game::**GetRandomBlock(),** chương trình chọn 1 mẫu khối và sinh bản sao để đưa vào game.
- **Template Method:**
  - Trong vòng lặp game, trình tự các bước (Input -> Update -> Render) được cố định như một "khuôn mẫu".
  - Hàm **Game::Run()** định nghĩa khung xử lý, còn chi tiết các bước (nhập phím, xoay khối, tính điểm) được triển khai riêng.

- Giúp đảm bảo luồng game nhất quán, dễ mở rộng hoặc thay đổi chi tiết.

- **Facade:**
  - Lớp **Game** đóng vai trò như 1 **Facade,** cung cấp giao diện đơn giản để điều phối toàn bộ hệ thống.
  - **main.cpp** chỉ cần gọi **game.Run()**, còn chi tiết xử lý được ẩn đi (quản lý **Grid**, khối, điểm, âm thanh, vẽ).
  - Nhờ vậy, code ở **main.cpp** rất gọn và dễ hiểu.

5. Kết quả và Demo





6. Đánh giá và Hạn chế

- **Những kết quả đạt được:**
    - Cài đặt thành công trò chơi **Tetris** với đầy đủ chức năng cơ bản: sinh khối, di chuyển, xoay, thả nhanh, phát hiện va chạm, khóa khối, xóa hàng đầy.
    - Áp dụng được các **nguyên lý OOP** (Encapsulation, Inheritance, Polymorphism, Abstraction).
    - Triển khai được một số **Design Patterns** (Prototype, Template Method, Facade) trong thực tế.
    - Tổ chức **cấu trúc project rõ ràng** (chia file header/source, thư mục resource, lib).
    - Game chạy ổn định, hiệu suất tốt (~60 FPS) và hỗ trợ build trên nhiều nền tảng (Windows/Linux).
    - Tích hợp **âm thanh và font chữ** để tăng trải nghiệm người chơi.
- **Hạn chế:**
    - Giao diện còn đơn giản, chưa có hiệu ứng đồ họa đẹp mắt.
    - Âm thanh mới dừng lại ở mức cơ bản (clear line, game over), chưa đa dạng.
    - Chưa có chế độ **hard drop** (thả khối tức thì).
    - Chưa có **bảng xếp hạng (leaderboard)** để lưu điểm cao.
    - Chưa có chế độ nhiều người chơi hoặc AI tự động.
    - Các thông số (kích thước ô, tốc độ tăng level, điểm cộng) còn cố định, chưa tùy chỉnh linh hoạt.
- **Hướng phát triển:**
    - Cải tiến giao diện (màu sắc, hiệu ứng, animation khi xóa hàng).
    - Bổ sung tính năng hard drop, preview nhiều khối tiếp theo.
    - Lưu trữ điểm số cao vào file, hiển thị leaderboard.
    - Tùy chỉnh mức độ khó, tốc độ rơi theo cấu hình ban đầu.

    - Xây dựng chế độ **multiplayer** hoặc đấu online.
    - Phát triển AI để tự động chơi (hỗ trợ nghiên cứu thuật toán).

7. Kết luận

Qua quá trình thực hiện đồ án **Tetris Game**, nhóm đã:

- Hiểu và vận dụng được các **nguyên lý lập trình hướng đối tượng (Encapsulation, Inheritance, Polymorphism, Abstraction)** vào dự án thực tế.
- Áp dụng được các **Design Patterns (Prototype, Template Method, Facade)** giúp cấu trúc code rõ ràng, dễ bảo trì và mở rộng.
- Hoàn thiện một trò chơi **Tetris hoạt động đầy đủ**, có giao diện đồ họa, âm thanh, tính điểm và tăng level.
- Cải thiện kỹ năng làm việc nhóm, phân chia công việc, sử dụng công cụ quản lý code (CMake, raylib).

Mặc dù còn một số hạn chế như giao diện đơn giản, thiếu các tính năng nâng cao (hard drop, leaderboard, multiplayer), nhưng đồ án đã đạt mục tiêu ban đầu: **xây dựng một ứng dụng game hoàn chỉnh, có tính ứng dụng và đúng tinh thần OOP.**

Đây là nền tảng để nhóm tiếp tục phát triển, nghiên cứu thêm các tính năng nâng cao và áp dụng vào các dự án lớn hơn trong tương lai.

8. Tài liệu tham khảo

- raylib docs
- CMake docs
- Slide/giáo trình môn OOP