fit@hcmus
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN, ĐHQG - HCM
KHOA CÔNG NGHỆ THÔNG TIN

BÁO CÁO

Đồ án Caro

(báo cáo lần hai)

Lớp: 25CTT3.
Môn: Cơ sở lập trình.
GVHD: Thầy Trương Toàn Thịnh.
Nhóm: 10.
Các thành viên:

Họ và tên	*MSSV*
Hà Thiện Danh	25120173
Trần Thái Huy	25120194
Huỳnh Gia Long	25120205
Cù Văn An	25120158
Phạm Đăng Quang	25120226

Thành phố Hồ Chí Minh, ngày 24 tháng 11 năm 2025.

Mục lục

1. Xác định chủ đề và phạm vi đề tài

1.1. Phân tích và xác định yêu cầu bài toán

Trò chơi Caro (Gomoku) là một trò chơi trí tuệ phổ biến với luật chơi đơn giản nhưng đòi hỏi tư duy logic cao. Để số hóa trò chơi này thành một sản phẩm phần mềm hoàn chỉnh, nhóm chúng em xác định các yêu cầu sau:

- Mô phỏng luật chơi chuẩn: Hệ thống phải tuân thủ chính xác luật chơi Caro quốc tế (hoặc luật truyền thống tùy chọn): người chơi thắng khi đạt được chuỗi 5 quân liên tiếp (ngang, dọc, chéo).
- Tương tác người dùng: Phần mềm phải hỗ trợ tương tác trực quan qua chuột hoặc bàn phím, đảm bảo độ trễ thấp và phản hồi tức thì khi người chơi thực hiện nước đi.
- Chế độ chơi đa dạng: Hệ thống cần hỗ trợ tối thiểu hai chế độ:
  - Người đấu với Người (PvP).
  - Người đấu với Máy (PvE) - yêu cầu tích hợp thuật toán chơi cờ.
- Tính toàn vẹn của dữ liệu: Đảm bảo trạng thái bàn cờ được lưu giữ chính xác trong suốt quá trình chơi, không xảy ra lỗi mất quân hay sai lượt đi.

1.2. Mục tiêu xây dựng hệ thống

a) Xây dựng hệ thống giao diện

Màn hình chính: Thiết kế trang chào mừng với các tùy chọn rõ ràng: chơi mới, tiếp tục, hướng dẫn, thoát.

Giao diện Cài đặt: Cho phép người dùng tùy chỉnh trải nghiệm như: bật/tắt âm thanh, độ khó của máy (dễ/trung bình/khó).

Giao diện bàn chơi: Hiển thị bàn cờ lưới, thông tin lượt đi hiện tại, tỉ số và các nút chức năng phụ trợ (xin đi lại – undo,...).

b) Xây dựng logic trò chơi

Phát triển thuật toán kiểm tra điều kiện thắng thua chính xác sau mỗi nước đi; hực hiện chức năng chuyển đổi lượt chơi giữa hai bên một cách đồng bộ.

1.3. Các yêu cầu kỹ thuật cần lưu ý

Để đảm bảo ứng dụng hoạt động ổn định và hiệu quả, chúng em xác định các yếu tố kỹ thuật trọng yếu cần tuân thủ trong quá trình phát triển:

Cấu trúc dữ liệu: Sử dụng mảng hai chiều hoặc cấu trúc dữ liệu phù hợp để biểu diễn trạng thái bàn cờ, giúp tối ưu hóa việc truy xuất và kiểm tra tọa độ quân cờ.

Tối ưu hóa thuật toán duyệt: Thuật toán kiểm tra chiến thắng cần được tối ưu để không phải duyệt toàn bộ bàn cờ sau mỗi nước đi, mà chỉ kiểm tra vùng lân cận của nước đi mới nhất. Đối với AI, cần giới hạn độ sâu tìm kiếm để đảm bảo máy phản hồi trong thời gian chấp nhận được (thường dưới 3 giây), tránh tình trạng treo ứng dụng.

Xử lý sự kiện và ngoại lệ: Ngăn chặn người dùng đánh vào ô đã có quân cờ hoặc đánh ra ngoài bàn cờ.

Đồ họa và Hiển thị: Sử dụng các thư viện đồ họa tiêu chuẩn để vẽ lưới và quân cờ sắc nét, đảm bảo giao diện không bị vỡ hoặc sai lệch khi thay đổi kích thước cửa sổ (nếu cho phép).

1.4. Thư viện sử dụng

Nhóm chúng em thống nhất sử dụng thư viện hỗ trợ SFML để xây dựng phần đồ họa, giao diện, âm thanh và xử lí các tác vụ của người chơi. Đây là một thư viện mã nguồn mở, dễ học, dễ sử dụng, phù hợp với năng lực của các thành viên trong nhóm.

2. Cấu trúc của mã nguồn

Ở phần này, báo cáo của chúng em xin trình bày sơ lược về tổng quan cấu trúc của mã nguồn, sau đó trình bày mã nguồn của 3 file tiêu biểu là `game_window.cpp`, `prepare_window.cpp` và `win_check.cpp`.

2.1. Tổng quan cấu trúc của mã nguồn

Để quản lý một mã nguồn khổng lồ, gồm hàng nghìn dòng code, một cấu trúc chung là rất cần thiết. Cấu trúc chung ấy phải được thiết kế sao cho mỗi thành phần phải thực sự tách bạch, đơn lẻ, nhưng vẫn có thể kết hợp lại với nhau một cách dễ dàng. Hơn nữa, hạn chế việc phải lập trình các hàm chồng chéo, sử dụng quá nhiều biến toàn cục, gây khó khăn trong việc quản lí, duy trì và mở rộng mã nguồn.

Sau khi cân nhắc các yếu tố trên, mã nguồn của nhóm chúng em được hệ thống theo nguyên tắc: Mỗi một màn hình là một file. Cách hệ thống này giúp chúng em dễ dàng quản lý và phát triển các màn hình một cách riêng lẻ.

a) Các file đóng vai trò toàn cục

Trong cấu trúc thư mục của dự án, nhóm các file toàn cục đóng vai trò nền tảng cốt lõi, bao gồm `MAIN_APP`, `templates` và `config`. Khác với các mô-đun chức năng riêng lẻ, đây là những thành phần đặc biệt được thiết kế để xử lý các nhiệm vụ mang tính quản trị và điều phối chung. Chúng chịu trách nhiệm khởi tạo môi trường, lưu trữ cấu hình và định hình luồng xử lý chính, do đó mọi tác vụ thực hiện tại đây sẽ ảnh hưởng trực tiếp đến sự vận hành ổn định của toàn bộ hệ thống cũng như sự tương tác giữa tất cả các file thành phần khác.

- File `MAIN_APP.cpp` là file chứa hàm `main`, đóng vai trò là điểm khởi chạy của toàn bộ chương trình. Tập tin này không chứa các logic xử lí chi tiết của trò chơi hay giao diện người dùng cụ thể, mà thực hiện chức năng khởi tạo môi trường ban đầu và điều phối luồng hoạt động chính của ứng dụng thông qua việc gọi các hàm màn hình thành phần.
  - Ngay khi chương trình bắt đầu, hàm `main` thực hiện việc tạo ra đối tượng cửa sổ hiển thị `sf::RenderWindow` dựa trên các thông số chiều rộng và chiều cao được quy định trong tập tin cấu hình `config.h`. Đồng thời, một đối tượng cấu trúc `gameDataPackage` cũng được khai báo tại đây. Biến này đóng vai trò là vùng nhớ trung gian dùng để tiếp nhận dữ liệu cài đặt từ màn hình chuẩn bị và truyền tải dữ liệu đó sang màn hình chơi game khi cần thiết. Cấu trúc điều khiển chính của chương trình được xây dựng dựa trên một vòng lặp `while` duy trì liên tục cho đến khi biến điều kiện `is_loop` nhận giá trị sai. Bên trong vòng lặp này, chúng em sử dụng cấu trúc rẽ nhánh `switch-case` để kiểm tra giá trị của biến toàn cục `globalConfig::current_win`. Biến này đóng vai trò định danh cho màn hình hiện tại mà ứng dụng cần hiển thị. Cơ chế này hoạt động như một máy trạng thái, trong đó mỗi giá trị số nguyên tương ứng với một màn hình chức năng cụ thể.
  - Khi biến `current_win` mang giá trị 0, chương trình sẽ gán `is_loop` bằng sai để thoát vòng lặp và kết thúc ứng dụng. Khi giá trị là 1, chương trình gọi hàm `drawMenuWindow` để hiển thị màn hình menu chính.

Khi giá trị là 2, hàm `drawPrepareScreen` được gọi để hiển thị màn hình chuẩn bị; đặc biệt, hàm này sau khi kết thúc sẽ trả về một gói dữ liệu `gameDataPackage` chứa các thông tin cài đặt của người dùng. Khi giá trị là 3, chương trình chuyển sang màn hình chơi game thông qua hàm `drawGameScreen`. Tại đây, một logic kiểm tra điều kiện được thực hiện dựa trên biến `previous_screen`: nếu người dùng quay lại từ màn hình cài đặt (giá trị 4), hàm sẽ được gọi với tham số khởi tạo là *false* để giữ nguyên trạng thái ván cờ; ngược lại, hàm sẽ được gọi với tham số *true* để bắt đầu ván mới với dữ liệu từ `package`. Các trường hợp còn lại tương ứng với màn hình cài đặt và các màn hình phụ khác.

- Sau mỗi lần thực thi một màn hình, biến `previous_screen` được cập nhật lại bằng giá trị của màn hình vừa hiển thị. Việc này nhằm mục đích lưu vết lịch sử điều hướng, hỗ trợ cho các chức năng quay lại ở các màn hình thành phần.

```
/* MODULE: MAIN_APP
 * Mục đích: Điểm khởi chạy chương trình, khởi tạo cửa sổ và điều phối luồng màn hình.
 * Cấu trúc: Vòng lặp chính và máy trạng thái (Switch-Case).
 */

#include "config.h"

// HÀM CHÍNH (ENTRY POINT)
FUNCTION main() {
    // 1. KHỞI TẠO MÔI TRƯỜNG
    // ------------------------------------------------
    // Tạo cửa sổ hiển thị dựa trên kích thước từ config
    VAR window = CreateRenderWindow(globalConfig::width, globalConfig::height);

    // Khai báo gói tin dữ liệu để truyền giữa các màn hình
    VAR package;

    // Biến kiểm soát vòng lặp chính
    VAR is_loop = true;

    // 2. VÒNG LẶP ĐIỀU PHỐI (MAIN LOOP)
    // ------------------------------------------------
    WHILE (is_loop) {

        // Kiểm tra trạng thái màn hình hiện tại
        SWITCH (globalConfig::current_win) {

            // CASE 0: THOÁT CHƯƠNG TRÌNH
            CASE 0:
                is_loop = false;
                BREAK;

            // CASE 1: MÀN HÌNH MENU CHÍNH
            CASE 1:
                drawMenuWindow(window);
                globalConfig::previous_win = 1; // Lưu vết
                BREAK;

            // CASE 2: MÀN HÌNH CHUẨN BỊ (LOAD/NEW GAME)
            CASE 2:
                // Gọi màn hình chuẩn bị và nhận lại gói dữ liệu cài đặt
                package = drawPrepareScreen(window);
                globalConfig::previous_win = 2;
                BREAK;

            // CASE 3: MÀN HÌNH CHƠI GAME (GAMEPLAY)
            CASE 3:
                // Kiểm tra nếu quay lại từ màn hình cài đặt (4) -> Không reset game
                IF (globalConfig::previous_win == 4) {
                    drawGameScreen(window, false, package);
                }
                // Ngược lại -> Khởi tạo game mới với dữ liệu từ package
                ELSE {
                    drawGameScreen(window, true, package);
                }
```

```
                globalConfig::previous_win = 3;
                BREAK;

            // CASE 4: MÀN HÌNH CÀI ĐẶT (SETTINGS)
            CASE 4:
                drawSettingsScreen(window);
                globalConfig::previous_win = 4;
                BREAK;

            // CASE 5: CÁC MÀN HÌNH KHÁC (INFO/HELP)
            CASE 5:
                drawADifferentWindow(window);
                globalConfig::previous_win = 5;
                BREAK;
        }
    }

    // Kết thúc chương trình
    RETURN 0;
}
```

Code 1. Mã giả trình bày tổng quát cấu trúc của file `MAIN_APP.cpp`.

• File `config` đóng vai trò "trung tâm điều phối" của toàn bộ dự án. Nó chịu trách nhiệm định nghĩa và lưu trữ các biến toàn cục (global variables) đại diện cho các thông số vận hành cốt lõi. Bất kỳ sự thay đổi nào tại đây đều sẽ tác động tức thời đến hành vi và giao diện của toàn bộ ứng dụng. Nó lưu trữ các thiết lập chung như kích thước cửa sổ, trạng thái âm thanh, chế độ hiển thị (giao diện sáng/tối), trạng thái ngôn ngữ (tiếng Việt/tiếng Anh),... Nó còn quản lý các biến trạng thái biểu thị màn hình hiện tại (current screen) và màn hình trước đó (previous screen), cho phép chương trình chuyển đổi mượt mà giữa các giao diện và hỗ trợ chức năng quay lại (nếu có).

```
#include ... // khai báo các thư viện
namespace globalConfig{
    extern unsigned int win_width, win_height;
    extern bool dark_mode, re_init, sound_on;
    extern int current_win, previous_win;
    extern short language;
};
```

Code 2. Cấu trúc của file `config.h`.

• File `templates` đóng vai trò như một lớp trừu tượng hóa và thư viện hỗ trợ, giúp tối ưu hóa quá trình xây dựng. Thay vì phải viết lại hàng chục dòng lệnh để khởi tạo một đối tượng đồ họa từ thư viện gốc (như sf::Text, sf::Sprite), file `templates` cung cấp các hàm dựng sẵn (helper functions). Các hàm này đóng gói logic phức tạp để tạo ra các thành phần giao diện phổ biến như nút bấm, ô nhập liệu,... Nhờ đó, nhóm phát triển chỉ cần gọi một dòng lệnh để tái sử dụng, giúp mã nguồn gọn gàng, dễ bảo trì và tiết kiệm đáng kể thời gian lập trình.

• Trong file `templates` còn định nghĩa cấu trúc dữ liệu (struct) `gameDataPackage`. Đây là một struct đặc biệt đóng vai trò như một "gói tin" trung gian. Nhiệm vụ của nó là đóng gói toàn bộ thông tin người dùng cài đặt tại *màn hình chuẩn bị* (tên người chơi, chế độ đấu, ai đi trước...) và truyền tải sang *màn hình chơi*.

```
struct gameDataPackage {
   bool is_new_game, is_multiplayer;
   int first_turn;
   std::string
         save_name,
         playerX_name,
         playerO_name;
   std::filesystem::path load_game_from;
};
```

Code 3. Cấu trúc `gameDataPackage`.

b) Các file đóng vai trò thành phần

Trong cấu trúc thư mục, bên cạnh phần lõi là nhóm các file định nghĩa cho từng màn hình. Khác với các file toàn cục, đây là các file chức năng riêng lẻ, được dùng để xây dựng giao diện cho từng phần cụ thể của trò chơi. Chúng chịu trách nhiệm vẽ hình ảnh và xử lý các thao tác của người chơi (như click chuột, bấm phím) tại màn hình đó, giúp trò chơi hoạt động đúng theo kịch bản của từng giai đoạn (như Menu hay lúc đang chơi).

Game được chia thành 4 màn hình: *màn hình chính* (`menu_window`), *màn hình cài đặt* (`settings_window`), *màn hình chuẩn bị* (`prepare_window`) và *màn hình chơi* (`game_window`).

• Mỗi file .cpp được cấu trúc thành các phần: phần khai báo các tài nguyên; phần định nghĩa các hàm xử lí các sự kiện (events); phần xử lí thuật toán (nếu có); phần định nghĩa quá trình vẽ (draw).

• Phần khai báo các tài nguyên là phần đầu của chương trình, gồm các biến toàn cục thuộc kiểu `sf::Font` hay `sf::Texture`, để khai báo các biến định nghĩa các phông chữ và các texture sẽ được sử dụng trong các hàm xuyên suốt chương trình.

• Phần xử lí thuật toán giúp xử lí các thuật toán có liên quan đến chương trình, bao gồm: xử lí và biểu diễn các thông tin đầu vào và đầu ra thành các biến với kiểu dữ liệu phù hợp; xử lí các tác dụng của người chơi lên trên các biến đó;... Không phải file nào cũng có phần này, mà đây là phần trọng tâm trong màn hình chơi, nơi mà các thuật toán kiểm tra đúng sai, hay sắp tới là thuật toán đánh cờ bằng máy sẽ được xử lí.

• Phần xử lí các sự kiện, đúng như tên gọi của nó, là nơi định nghĩa các hàm để xử lí các sự kiện mà người chơi thực hiện.

• Phần vẽ sẽ bao gồm vẽ lên giao diện của màn hình. Tương ứng với mỗi hàm sẽ vẽ nên một đối tượng, kết hợp với tiếp nhận sự kiện được truyền vào thông qua tham số `event`, kiểm tra sự kiện đó có liên quan đến đối tượng mà hàm đó định nghĩa hay không. Từ đó, các hàm phản ứng tương ứng bằng cách thay đổi cách xuất hiện của đối tượng, đồng thời thông báo cho các hàm tương ứng của phần xử lí thực hiện xử lí các sự kiện vừa được phát hiện.

• Một hàm vẽ `drawForEachLoop()` sẽ gọi tất cả các hàm của phần vẽ theo thứ tự: đối tượng nào cần được vẽ trước thì hàm tương ứng của nó sẽ được gọi trước. Đây là hàm thực hiện vẽ tất cả các đối tượng đã được định nghĩa trong phần vẽ trong một vòng lặp. Hàm vẽ này cũng nhận sự kiện được truyền vào từ hàm lặp thông qua tham số `event`, từ đó truyền sự kiện đó vào các hàm ở phần vẽ để thực hiện xử lí.

• Một hàm lặp `loop`***`Win`***`Screen()` sẽ thu thập tất cả các sự kiện của người chơi vào biến `event`, sau đó truyền sự kiện đó vào hàm vẽ, đồng thời gọi liên tục hàm vẽ `drawForEachLoop()` để vẽ màn hình *Win* liên tục cho từng vòng lặp. Trong mỗi vòng lặp, hàm sẽ kiểm tra xem liệu liệu

• Một hàm gọi `drawWinScreen()` sẽ là hàm đại diện duy nhất cho cả màn hình *Win*, được gọi tại `main` khi người chơi bước vào màn hình đó.

Các phần được đặt vào một không gian tên (namespace) riêng biệt, giúp phân biệt rõ vai trò của từng hàm, giúp chương trình trở nên tường minh, tránh nhầm lẫn.

Đoạn mã giả bên dưới trình bày tổng quát cấu trúc của file `prepare_win.cpp`, minh họa cho những gì vừa được trình bày ở trên.

```
1.  /* MODULE: PREPARE_WINDOW
2.   * Mục đích: Quản lý màn hình Menu chính, danh sách file save và thiết lập thông số cho ván game mới.
3.   * Nguyên tắc: Phân tách logic điều hướng, xử lý dữ liệu file và vẽ giao diện.
4.   */
5.  #include "config.h" // Import các biến cấu hình toàn cục
6.  // 1. QUẢN LÝ TÀI NGUYÊN & TRẠNG THÁI (MODEL)
```

```
// ------------------------------------------------------------------------
STRUCT SaveData {
    // Cấu trúc lưu trữ thông tin meta của một file save
    VAR name, path;        // Tên hiển thị và đường dẫn file
    VAR time;              // Thời gian chỉnh sửa cuối cùng
    VAR is_multiplayer;    // Chế độ chơi của file save đó
    FUNCTION IsNewerThan(other); // Hàm so sánh để sắp xếp theo thời gian
}

NAMESPACE internalStats {
    // Các biến trạng thái nội bộ của màn hình chuẩn bị
    VAR stage;             // 0: Menu chính, 1: Danh sách Save, 2: Tạo Game mới
    VAR list_of_saves;     // Vector chứa danh sách các file save tìm được
    VAR current_page;      // Trang hiện tại (để phân trang danh sách save)
}

NAMESPACE newGameData {
    // Lưu trữ tạm thời các cài đặt người dùng chọn trước khi bắt đầu game
    VAR is_new_game, is_multiplayer;
    VAR save_name;         // Tên file sẽ lưu
    VAR playerX, playerO;  // Tên hai người chơi
    VAR first_turn;        // Ai đi trước

    FUNCTION clear();              // Reset về mặc định
    FUNCTION convertToPackage();    // Đóng gói dữ liệu để gửi sang màn hình Game
}

// 2. XỬ LÝ DỮ LIỆU NỀN TẢNG (BACKEND LOGIC)
// ------------------------------------------------------------------------
NAMESPACE process {
    FUNCTION getFileModTime(filepath) {
        // Lấy thời gian sửa đổi file từ hệ điều hành
        // Chuyển đổi thành định dạng ngày/tháng/năm/giờ/phút
    }

    FUNCTION loadListOfGames() {
        // Quét toàn bộ thư mục "saves/"
        // Lọc các file hợp lệ (.txt cho Multiplayer, .txtx cho Singleplayer)
        // Đưa vào vector list_of_saves và gọi hàm sắp xếp (mới nhất lên đầu)
    }
}

// 3. XỬ LÝ SỰ KIỆN & TƯƠNG TÁC (CONTROLLER)
// ------------------------------------------------------------------------
NAMESPACE events {
    // Điều hướng giữa các giai đoạn (Stage)
    FUNCTION onLoadGameClicked() { internalStats::stage = 1; }
    FUNCTION onNewGameClicked()  { internalStats::stage = 2; }

    FUNCTION onBackClicked() {
        // Nếu đang ở Menu chính -> Thoát ra màn hình chờ
        // Nếu đang ở trang con (Load/New) -> Quay về Menu chính
    }

    // Xử lý phân trang danh sách save
    FUNCTION onNextPage() { internalStats::current_page++; }
    FUNCTION onPrevPage() { internalStats::current_page--; }
}

// 4. VẼ GIAO DIỆN (VIEW)
// ------------------------------------------------------------------------
NAMESPACE draw {
    FUNCTION drawBackground(window) {
        // Vẽ hình nền đã scale vừa kích thước cửa sổ
    }

    // --- GIAI ĐOẠN 0: MENU CHÍNH ---
    FUNCTION drawStageZero_Menu(window) {
        // Vẽ 2 nút lớn: "Load Game" và "New Game"
    }

    // --- GIAI ĐOẠN 1: DANH SÁCH SAVE ---
    FUNCTION drawStageOne_List(window) {
        // Gọi process::loadListOfGames() nếu cần cập nhật
```

```
        // Duyệt danh sách save theo trang hiện tại
        // Vẽ từng dòng thông tin (Tên, Ngày giờ, Chế độ)
        // Vẽ nút chuyển trang (Next/Prev)
    }

    // --- GIAI ĐOẠN 2: THIẾT LẬP GAME MỚI ---
    FUNCTION drawStageTwo_Setup(window) {
        // Vẽ nút chọn chế độ: 1 Người chơi vs 2 Người chơi

        IF (Chọn 2 Người chơi) {
            // Vẽ tiêu đề và các ô nhập liệu (Tên Save, Tên Player 1, Tên Player 2)
            // Vẽ nút gạt chọn ai đi trước (X hoặc O)
            // Vẽ nút "Play" để bắt đầu
        }
        ELSE {
            // Hiển thị thông báo "Tính năng đang phát triển"
        }
    }
}

// 5. VÒNG LẶP CHÍNH & API (MAIN LOOP)
// ------------------------------------------------------------------------
--

FUNCTION renderCurrentFrame(window) {
    window.clear();
    draw::drawBackground(window);
    draw::drawBackButton(window);

    // Vẽ nội dung tùy theo giai đoạn (Stage) hiện tại
    SWITCH (internalStats::stage) {
        CASE 0: draw::drawStageZero_Menu(window); BREAK;
        CASE 1: draw::drawStageOne_List(window);  BREAK;
        CASE 2: draw::drawStageTwo_Setup(window); BREAK;
    }
    window.display();
}

FUNCTION runPrepareScreen(window) {
    // Reset dữ liệu tạm
    newGameData::clear();
    internalStats::stage = 0;

    WHILE (window.isOpen) {
        // Xử lý sự kiện (Click chuột, Nhập phím...)
        WHILE (getEvent) {
            IF (Click Back) -> events::onBackClicked();
            IF (Click Load) -> events::onLoadGameClicked();
            IF (Click New)  -> events::onNewGameClicked();
            // ... Các sự kiện khác
        }

        // Vẽ màn hình
        renderCurrentFrame(window);

        // Nếu người dùng đã chốt xong cài đặt -> Thoát vòng lặp
        IF (GameReady) BREAK;
    }

    // Trả về gói tin cài đặt để khởi tạo Game
    RETURN newGameData::convertToPackage();
}
```

Code 4. Mã giả trình bày tổng quát cấu trúc của file `prepare_window.cpp`.

Ngoài ra, đồ án của chúng em còn có một file đặc biệt là `win_check`. File `win_check` giúp kiểm tra điều kiện thắng thua trong khi người chơi đang ở *màn hình chơi*. Nó nhận vào các thông số cho mỗi lần gọi (tương ứng với mỗi vòng lặp của trò chơi), và trả về kết quả thắng và chưa thắng. Nếu phát hiện thắng thì trả về các ô thắng tương ứng để *màn hình chơi* có thể gọi hàm vẽ đánh dấu vị trí thắng. File này được trình bày chi tiết ở phần 2.2.3.

2.2. Sơ lược về mã nguồn của một số file tiêu biểu

2.2.1. Mã nguồn màn hình chơi

Tương tự như các màn hình khác, *màn hình game* (`game_window.cpp`) là nơi diễn ra các hoạt động chính của trò chơi. Đây là màn hình phức tạp nhất, nơi người chơi tương tác trực tiếp với bàn cờ, thực hiện các nước đi và nhận kết quả thắng thua. Màn hình này nhận dữ liệu đầu vào (tên người chơi, chế độ, lượt đi...) từ gói tin `gameDataPackage` được gửi đến từ màn hình chuẩn bị, sau đó khởi tạo và quản lý toàn bộ vòng đời của một ván đấu.

Cấu trúc của file này cũng tuân thủ nguyên tắc phân chia thành các không gian tên để quản lý các nhóm chức năng riêng biệt:

• File bắt đầu bằng không gian tên `gameStat`. Thay vì dùng các biến toàn cục rải rác khó kiểm soát, chúng em gom nhóm toàn bộ dữ liệu trạng thái của ván cờ vào đây. Không gian tên này lưu trữ các thông tin biến đổi liên tục như: tên hai người chơi, tỉ số hiện tại, lượt đi (X hay O), danh sách các nước đã đi (để hỗ trợ chức năng undo) và trạng thái của lưới ô vuông 16×16. Các biến này hỗ trợ quản lí các nước đi, giúp các hàm sự kiện và các hàm vẽ có thể tương tác và thực hiện nhiệm vụ.

• Tiếp nối phần quản lý dữ liệu là khu vực khai báo và khởi tạo các tài nguyên đồ họa. Tại đây, chúng em định nghĩa các đối tượng bao gồm các bộ phông chữ (sf::Font) để hiển thị văn bản, các bộ hình ảnh bề mặt (sf::Texture) phục vụ cho cả hai chế độ Sáng và Tối, các đối tượng hình (sf::Sprite), các đối tượng văn bản (sf::Text) và một đối tượng đặc biệt là đối tượng góc nhìn (`sf::View`) `caroTableView` (chúng em sẽ trình bày riêng ở bên dưới). Trong các đối tượng đó bao gồm các đối tượng hiển thị cụ thể như bàn cờ, quân X, quân O, hay các nút bấm. Việc chuẩn bị trước các tài nguyên này ngay từ đầu giúp chương trình không phải tải lại hình ảnh liên tục trong quá trình chơi, từ đó tối ưu hóa hiệu năng và giúp các hàm vẽ ở phần sau chỉ cần tập trung vào nhiệm vụ hiển thị.

• Tiếp theo là không gian tên `process`, nơi đảm nhiệm các thuật toán xử lí dữ liệu từ các biến trong không gian tên gameStat. Tại đây chứa các hàm như `loadGame` để đọc dữ liệu từ file save cũ và khôi phục lại bàn cờ, hay `saveGame` để ghi trạng thái hiện tại của ván đấu vào bộ nhớ máy tính. Hàm `newGame` cũng nằm ở đây với nhiệm vụ xóa trắng dữ liệu cũ để bắt đầu một ván chơi mới sạch sẽ.

• Phần xử lý sự kiện nằm trong không gian tên `events`, đóng vai trò là "bộ điều khiển". Nó lắng nghe và phản hồi các thao tác của người dùng. Khi người chơi click chuột vào bàn cờ, hàm `addAMove` sẽ được gọi để tính toán tọa độ và đặt quân cờ. Các chức năng phụ trợ như `undoMove` (xin đi lại), `switchLightDarkMode` (đổi màu giao diện), hay xử lí việc nhập tên file khi lưu game cũng được định nghĩa tại đây.

• Cuối cùng là không gian tên `draw`, chịu trách nhiệm vẽ các đồ họa ra cửa sổ. Các hàm trong phần này sẽ lấy dữ liệu từ không gian tên `gameStat` để vẽ lên màn hình. Nó bao gồm việc vẽ lưới bàn cờ, vẽ các quân X, O tại đúng vị trí, hiển thị bảng điểm và các hiệu ứng trực quan như tô màu ô khi chuột lướt qua. Ngoài ra, không gian tên này còn xử lí việc vẽ các giao diện phức tạp hơn như các cửa sổ bật lên (popup) thông báo chiến thắng hoặc hộp thoại nhập tên để lưu game.

• Mọi hoạt động trên được điều phối bởi vòng lặp chính `loopGameScreen`. Hàm này hoạt động liên tục, kiểm tra các sự kiện đầu vào rồi gọi các hàm vẽ tương ứng để cập nhật màn hình sau mỗi khung hình.

• Để xử lý việc hiển thị bàn cờ, chúng em khai báo đối tượng `sf::View caroTableView`. Trong SFML, View hoạt động giống như một chiếc "camera 2D". Thay vì vẽ cố định bàn cờ lên cửa sổ, chúng em sử dụng View này để tạo ra một vùng hiển thị riêng biệt (viewport). Cửa sổ này hỗ trợ chúng em viết thuật toán xác định vị trí ô

tương ứng mà con trỏ chuột đang trỏ vào một cách dễ dàng hơn. Cụ thể, thông qua hàm chuyển đổi tọa độ tích hợp sẵn, chương trình có thể ánh xạ chính xác vị trí con trỏ chuột từ hệ tọa độ của màn hình sang hệ tọa độ của bàn cờ. Sau khi có được vị trí thực tế này, chúng em chỉ cần thực hiện phép chia nguyên cho kích thước của một ô cờ là có thể suy ra ngay chỉ số hàng và cột tương ứng trong mảng dữ liệu. Nhờ phương pháp này, chúng em đỡ phải thực hiện các bước tính toán bù trừ khoảng cách phức tạp, tránh sai sót, đảm bảo người chơi luôn đánh đúng vào ô mong muốn.

Chúng em phải thừa nhận rằng, vì là thành phần được xây dựng đầu tiên, file này chưa áp dụng cấu trúc mô-đun hóa chuẩn mực được mô tả tại mục 2.1b. Hệ quả là dù đóng vai trò then chốt, sự thiếu đồng bộ trong tổ chức mã nguồn đã làm giảm khả năng đọc hiểu mã nguồn, đồng thời gây khó khăn cho việc duy trì và phát triển các tính năng mới.

```
/* MODULE: GAME_WINDOW
 * Mục đích: Quản lý toàn bộ vòng đời, logic và giao diện của màn hình chơi game.
 */
#include "config.h" // Import các biến cấu hình toàn cục
// 1. QUẢN LÝ TÀI NGUYÊN & TRẠNG THÁI (MODEL & RESOURCES)
// ------------------------------------------------------------------------
NAMESPACE gameStat {
    // Lưu trữ toàn bộ dữ liệu biến đổi của ván đấu tại một nơi duy nhất
    VAR name, player1_name, player2_name;   // Thông tin định danh
    VAR is_win, is_save, is_multiplayer;    // Các cờ kiểm soát trạng thái
    VAR score_x, score_y;                  // Điểm số tích lũy
    VAR first_turn;                        // Lượt đi hiện tại

    // Cấu trúc dữ liệu bàn cờ
    VAR moves_history;       // Vector lưu lịch sử nước đi (để Undo/Redo)
    VAR cells_matrix[16][16];// Ma trận logic 2 chiều ánh xạ bàn cờ
}

// Khai báo các đối tượng đồ họa tĩnh (tránh load lại nhiều lần)
VAR caroTableView; // sf::View: Camera 2D để quản lý vùng nhìn bàn cờ riêng biệt
VAR Textures;      // Các ảnh: Bàn cờ, Quân X/O, Nút bấm (phiên bản Sáng/Tối)
VAR Shapes;        // Các khối hình: Bảng điểm, Popup thông báo
VAR Texts;         // Các dòng chữ: Tên lượt, Điểm số, Thông báo thắng/thua

// 2. XỬ LÝ DỮ LIỆU NỀN TẢNG (BACKEND LOGIC)
// ------------------------------------------------------------------------
NAMESPACE process {
    FUNCTION loadGame(path) {
        // Đọc dữ liệu từ file save .txt
        // Khôi phục lại lịch sử moves_history và ma trận cells_matrix
    }

    FUNCTION newGame(settings) {
        // Reset toàn bộ biến trong gameStat về 0 hoặc rỗng
        // Chuẩn bị cho ván đấu mới
    }

    FUNCTION saveGame() {
        // Kiểm tra thư mục, tạo file mới
        // Ghi toàn bộ trạng thái hiện tại từ gameStat xuống ổ cứng
    }
}

// 3. XỬ LÝ SỰ KIỆN & TƯƠNG TÁC (CONTROLLER)
// ------------------------------------------------------------------------
NAMESPACE events {
    FUNCTION addAMove(mouse_position) {
        // Chuyển đổi tọa độ chuột -> tọa độ ô cờ (thông qua caroTableView)
        // Nếu ô hợp lệ và trống -> Thêm vào moves_history, cập nhật cells_matrix
        // Kiểm tra điều kiện thắng thua ngay sau nước đi
    }

    FUNCTION undoMove() {
        // Loại bỏ phần tử cuối cùng khỏi moves_history
        // Cập nhật lại trạng thái bàn cờ
    }
```

```
57.
58.     FUNCTION switchLightDarkMode() {
59.         // Đảo ngược biến globalConfig::dark_mode
60.         // Cập nhật lại texture cho các nút bấm
61.     }
62.
63.     FUNCTION readGameName(keyboard_event) {
64.         // Xử lý nhập liệu từ bàn phím khi người chơi đặt tên file save
65.         // Chặn các ký tự đặc biệt không hợp lệ
66.     }
67.
68.     // Các hàm xử lý sự kiện phụ: updateScore, exitButton, continueOption...
69. }
70.
71. // 4. VẼ GIAO DIỆN (VIEW)
72. // ------------------------------------------------------------------------
73. NAMESPACE draw {
74.     FUNCTION drawAllMoves(window) {
75.         // Duyệt vector moves_history
76.         // Vẽ quân X hoặc O tại tọa độ tương ứng lên màn hình
77.     }
78.
79.     FUNCTION drawLastMoveMarker(window) {
80.         // Vẽ viền đỏ bao quanh nước đi gần nhất để người chơi dễ nhìn
81.     }
82.
83.     FUNCTION drawScoreboard(window) {
84.         // Cập nhật text điểm số từ gameStat và vẽ bảng điểm
85.     }
86.
87.     FUNCTION drawSavePromptPopup(window) {
88.         // Vẽ hộp thoại nhập tên file khi người chơi muốn thoát/lưu game
89.         // Hiển thị cảnh báo nếu tên không hợp lệ
90.     }
91.
92.     FUNCTION drawWinningIndication(window) {
93.         // Nếu có người thắng -> Gọi thuật toán tìm 5 ô liên tiếp
94.         // Tô màu nổi bật các ô đó
95.     }
96.
97.     // Các hàm vẽ thành phần UI khác: UndoButton, SettingsButton, TurnIndicator...
98. }
99.
100. // 5. VÒNG LẶP CHÍNH & API (MAIN LOOP)
101. // ------------------------------------------------------------------------
102.
103. FUNCTION initGameScreen(start_data) {
104.     // Thiết lập Viewport cho camera bàn cờ
105.     // Load texture, font chữ, cài đặt vị trí ban đầu cho các nút
106.     // Nếu là game cũ -> gọi process::loadGame()
107.     // Nếu là game mới -> gọi process::newGame()
108. }
109.
110. FUNCTION loopGameScreen(window) {
111.     WHILE (window.isOpen) {
112.         // A. Xử lý sự kiện (Polling Events)
113.         WHILE (getEvent) {
114.             IF (Click vào bàn cờ) -> events::addAMove();
115.             IF (Click Undo)       -> events::undoMove();
116.             IF (Click Save)       -> Bật cờ is_save -> Hiện Popup;
117.             IF (Nhập phím)        -> events::readGameName();
118.         }
119.
120.         // B. Vẽ ra màn hình (Rendering)
121.         window.clear();
122.
123.         draw::drawBackground();      // Vẽ nền
124.         draw::drawScoreboard();      // Vẽ bảng điểm
125.         draw::drawUIButtons();       // Vẽ các nút chức năng
126.
127.         window.setView(caroTableView); // Chuyển Camera vào vùng bàn cờ
128.         draw::drawAllMoves();          // Vẽ các quân cờ
129.         window.setView(Default);       // Trả Camera về mặc định
130.
```

```
        IF (is_save) draw::drawSavePromptPopup(); // Vẽ popup nếu cần

        window.display();
    }
}

// hàm công khai được gọi từ hàm Main
FUNCTION drawGameScreen(window, package) {
    // Nhận gói tin cài đặt (package) từ màn hình Menu
    initGameScreen(package);
    loopGameScreen(window); // Bắt đầu vòng lặp trò chơi
}
```

Code 5. Mã giả trình bày tổng quát cấu trúc của file `game_window.cpp`.

2.2.2. Mã nguồn màn hình chuẩn bị

File `prepare_window.cpp` đảm nhận chức năng quản lý màn hình chuẩn bị. Chức năng chính của màn hình này là cung cấp giao diện để người dùng thực hiện các thao tác khởi tạo trước khi vào trò chơi, bao gồm việc tải dữ liệu từ các ván cờ đã lưu hoặc thiết lập các thông số cho một ván cờ mới. Cấu trúc mã nguồn của file này tuân thủ nguyên tắc phân chia thành các không gian tên để tách biệt các nhóm chức năng xử lý dữ liệu, xử lý sự kiện và vẽ giao diện.

Về mặt tổ chức và quản lý dữ liệu, file định nghĩa một cấu trúc dữ liệu tên là `Saves_data` và hai không gian tên lưu trữ biến là `internalStats` và `newGameData`.

- Cấu trúc `Saves_data` được sử dụng để lưu trữ các thuộc tính của một file lưu trữ, bao gồm tên file, đường dẫn tuyệt đối, thời gian chỉnh sửa cuối cùng và chế độ chơi (một hoặc hai người chơi).

- Không gian tên `internalStats` chứa các biến quản lý trạng thái nội bộ của màn hình, cụ thể là biến `stage` dùng để định danh trạng thái hiển thị hiện tại (menu chính, menu load game, hoặc menu new game), biến `status` dùng để xác định trạng thái của màn hình (chưa tải ván chơi, đã tải ván cũ hoặc đã tạo ván mới), và một vector `list_of_saves` dùng để lưu trữ danh sách các đối tượng `Saves_data` là tên các ván chơi đã được lưu trữ trong máy.

- Không gian tên `newGameData` đóng vai trò là vùng đệm dữ liệu tạm thời, lưu trữ các giá trị mà người dùng nhập vào trong quá trình thiết lập game mới, bao gồm tên ván đấu, tên hai người chơi, chế độ chơi và quyền đi trước.

- Không gian tên `process` chịu trách nhiệm thực thi các thuật toán xử lí liên quan đến hệ thống tập tin. Hàm quan trọng nhất trong không gian tên này là `loadListOfGames`. Hàm này thực hiện việc duyệt qua thư mục chứa dữ liệu lưu trữ bằng đối tượng `std::filesystem`. Trong quá trình duyệt, hàm sẽ kiểm tra phần mở rộng của từng tập tin để phân loại dữ liệu (ví dụ định dạng `.txt` cho chế độ hai người chơi và `.txtx` cho chế độ một người chơi). Sau khi lọc ra các tập tin hợp lệ, hàm sẽ truy xuất thông tin thời gian chỉnh sửa cuối cùng của tập tin và lưu vào cấu trúc dữ liệu. Cuối cùng, danh sách các tập tin này sẽ được sắp xếp lại theo thứ tự thời gian giảm dần thông qua một thuật toán so sánh, đảm bảo các tập tin mới nhất được hiển thị ở vị trí đầu tiên.

- Không gian tên `events` chứa các hàm xử lý sự kiện đầu vào từ người dùng. Các hàm này hoạt động bằng cách thay đổi giá trị của biến trạng thái `internalStats::stage`. Khi người dùng tác động lên các nút chức năng như "Load Game" hoặc "New Game", hàm tương ứng sẽ cập nhật biến `stage`, từ đó điều hướng chương trình sang màn hình con phù hợp. Ngoài ra, không gian tên này còn chứa các logic xử lí phân trang cho danh sách tập tin (tăng hoặc giảm chỉ số trang) và logic xử lí sự kiện nhập văn bản để cập nhật dữ liệu vào các biến trong `newGameData` khi người dùng gõ phím.

• Không gian tên `draw` đảm nhiệm chức năng hiển thị các đối tượng đồ họa lên màn hình. Hàm vẽ chính sẽ kiểm tra giá trị của biến `stage` để quyết định gọi các hàm vẽ thành phần tương ứng. Nếu `stage` ở trạng thái menu, hệ thống vẽ các nút lựa chọn chức năng. Nếu `stage` ở trạng thái *load game* (tải ván chơi cũ), hệ thống vẽ danh sách các tập tin đã được tải trong `internalStats::list_of_saves` cùng với các nút điều hướng trang. Nếu `stage` ở trạng thái *new game* (tạo ván chơi mới), hệ thống vẽ các ô nhập liệu và các nút tùy chọn. Các hàm vẽ này sử dụng lại các hàm tiện ích đã được định nghĩa trong file `templates` để khởi tạo các đối tượng nút bấm và ô nhập liệu.

• Hàm khởi chạy chính của màn hình này là `drawPrepareScreen`. Hàm này gọi hàm `loopPrepareScreen` để thực hiện vòng lặp chính để duy trì màn hình, liên tục gọi các hàm xử lý sự kiện và hàm vẽ. Khi người dùng hoàn tất thao tác và chuyển sang màn hình chơi game, hàm này sẽ đóng gói toàn bộ dữ liệu từ `newGameData` vào cấu trúc `gameDataPackage` và trả về kết quả cho chương trình chính.

```
/* MODULE: PREPARE_WINDOW
 * Mục đích: Quản lý màn hình Menu, danh sách file save và cài đặt game mới.
 * Cấu trúc: Phân tách dữ liệu, xử lý file, sự kiện và vẽ.
 */

#include "config.h"

// 1. DỮ LIỆU VÀ TRẠNG THÁI (MODEL)
// ---------------------------------------------------------------------------
STRUCT SaveData {
    VAR name, path;        // Tên và đường dẫn file
    VAR time;              // Thời gian sửa đổi
    VAR is_multiplayer;    // Chế độ chơi
    FUNCTION IsNewerThan(other); // So sánh thời gian
}

NAMESPACE internalStats {
    VAR stage;             // Biến xác định màn hình con (0, 1, 2)
    VAR list_of_saves;     // Danh sách file save
    VAR current_page;      // Số trang hiện tại
}

NAMESPACE newGameData {
    VAR is_new_game, is_multiplayer;
    VAR save_name;         // Tên file save mới
    VAR playerX, playerO;  // Tên người chơi
    VAR first_turn;        // Lượt đi đầu

    FUNCTION clear();              // Xóa dữ liệu
    FUNCTION convertToPackage();    // Đóng gói dữ liệu
}

// 2. XỬ LÝ LOGIC (BACKEND)
// ---------------------------------------------------------------------------
NAMESPACE process {
    FUNCTION getFileModTime(filepath) {
        // Lấy thời gian sửa đổi file
    }

    FUNCTION loadListOfGames() {
        // Quét thư mục "saves"
        // Lọc file và thêm vào list_of_saves
        // Sắp xếp danh sách
    }
}

// 3. XỬ LÝ SỰ KIỆN (CONTROLLER)
// ---------------------------------------------------------------------------
NAMESPACE events {
    FUNCTION onLoadGameClicked() { internalStats::stage = 1; }
    FUNCTION onNewGameClicked()  { internalStats::stage = 2; }

    FUNCTION onBackClicked() {
```

```
54.         // Quay lại màn hình trước hoặc thoát
55.     }
56.
57.     FUNCTION onNextPage() { internalStats::current_page++; }
58.     FUNCTION onPrevPage() { internalStats::current_page--; }
59. }
60.
61. // 4. VẼ GIAO DIỆN (VIEW)
62. // ------------------------------------------------------------------------
63. NAMESPACE draw {
64.     FUNCTION drawBackground(window) {
65.         // Vẽ nền
66.     }
67.
68.     // --- GIAI ĐOẠN 0: MENU ---
69.     FUNCTION drawStageZero_Menu(window) {
70.         // Vẽ nút Load và New Game
71.     }
72.
73.     // --- GIAI ĐOẠN 1: DANH SÁCH SAVE ---
74.     FUNCTION drawStageOne_List(window) {
75.         // Vẽ danh sách file save
76.         // Vẽ phân trang
77.     }
78.
79.     // --- GIAI ĐOẠN 2: CÀI ĐẶT ---
80.     FUNCTION drawStageTwo_Setup(window) {
81.         // Vẽ tùy chọn chế độ chơi
82.         // Vẽ ô nhập liệu và nút Play
83.     }
84. }
85.
86. // 5. VÒNG LẶP CHÍNH (MAIN LOOP)
87. // ------------------------------------------------------------------------
88.
89. FUNCTION renderCurrentFrame(window) {
90.     window.clear();
91.     draw::drawBackground(window);
92.     draw::drawBackButton(window);
93.
94.     SWITCH (internalStats::stage) {
95.         CASE 0: draw::drawStageZero_Menu(window); BREAK;
96.         CASE 1: draw::drawStageOne_List(window);  BREAK;
97.         CASE 2: draw::drawStageTwo_Setup(window); BREAK;
98.     }
99.     window.display();
100. }
101.
102. FUNCTION runPrepareScreen(window) {
103.     newGameData::clear();
104.     internalStats::stage = 0;
105.
106.     WHILE (window.isOpen) {
107.         WHILE (getEvent) {
108.             // Gọi các hàm trong namespace events dựa trên sự kiện
109.         }
110.
111.         renderCurrentFrame(window);
112.
113.         IF (GameReady) BREAK;
114.     }
115.
116.     RETURN newGameData::convertToPackage();
117. }
```

Code 6. Mã giả trình bày tổng quát cấu trúc của file `prepare_window.cpp`.

2.2.3. Mã nguồn kiểm tra thắng thua

File `win_check.cpp` là một file mã nguồn phụ trợ nhưng đóng vai trò thiết yếu trong việc vận hành logic của trò chơi. Chức năng duy nhất của tập tin này là thực hiện các phép toán kiểm tra xem sau nước đi vừa thực hiện,

người chơi đã đạt đủ điều kiện để chiến thắng hay chưa. Mã nguồn trong tập tin này hoạt động độc lập với giao diện đồ họa, chỉ tiếp nhận dữ liệu đầu vào là trạng thái bàn cờ và trả về kết quả tính toán.

Về mặt cấu trúc chi tiết, tập tin bao gồm hai hàm chức năng là `inside` và `checkForWin`. Hàm inside là một hàm phụ trợ đơn giản, nhận vào một số nguyên và trả về giá trị đúng hoặc sai. Nhiệm vụ của hàm này là kiểm tra xem một chỉ số tọa độ có nằm trong phạm vi hợp lệ của bàn cờ hay không (từ 0 đến 15). Việc kiểm tra này là bắt buộc để đảm bảo chương trình không truy cập vào các vùng nhớ không xác định bên ngoài mảng dữ liệu, tránh gây ra lỗi dừng chương trình đột ngột trong quá trình duyệt các ô cờ ở biên.

Hàm `checkForWin` là hàm xử lý chính. Hàm này nhận vào ba tham số: tọa độ của nước đi cuối cùng (`last_move`), ma trận trạng thái toàn bộ bàn cờ (`cells`), và một biến tham chiếu trạng thái (`status`) để cập nhật kết quả thắng thua. Thuật toán được sử dụng trong hàm này là thuật toán duyệt loang từ tâm. Để tránh việc viết lại mã lệnh lặp đi lặp lại, nhóm chúng em sử dụng một biểu thức lambda tên là `check` để định nghĩa quy trình duyệt chung cho các hướng.

Quy trình kiểm tra được thực hiện lần lượt trên bốn trục: trục ngang, trục dọc, trục chéo chính và trục chéo phụ. Tại mỗi trục, thuật toán sử dụng một vòng lặp để mở rộng phạm vi kiểm tra dần ra xa vị trí nước đi cuối cùng theo cả hai phía đối xứng. Trong mỗi lần lặp, thuật toán gọi hàm `inside` để xác định tính hợp lệ của ô cờ, sau đó so sánh giá trị của ô đang xét với giá trị của quân cờ vừa đánh. Nếu các ô liên tiếp có cùng giá trị, biến đếm số lượng sẽ được tăng lên và tọa độ của ô đó sẽ được thêm vào danh sách kết quả. Ngược lại, nếu gặp ô trống, ô khác màu hoặc đi ra ngoài biên bàn cờ, vòng lặp sẽ dừng lại. Nếu biến đếm đạt giá trị từ 5 trở lên, thuật toán sẽ cập nhật biến `status` thành giá trị đúng, báo hiệu đã có người chiến thắng. Cuối cùng, hàm trả về một vector chứa danh sách tọa độ các quân cờ tạo thành đường chiến thắng để phục vụ cho việc vẽ hiệu ứng nổi bật trên giao diện.

```
1.  bool inside(short n){
2.      return (n >= 0 && n < 16);
3.  };

4.  std::vector<sf::Vector2i> checkForWin(
5.      sf::Vector2i& last_move,
6.      std::vector<std::vector<char> >& cells,
7.      bool& status
8.  ){
9.      std::vector<sf::Vector2i> result;
10.     result.push_back(last_move);
11.     bool status2 = true;
12.     auto check = [&last_move, &cells, &status, &status2, &result](short _x, short
    _y){
13.         int count = 1, i = 0;
14.         bool end_loop = false;
15.         while(!end_loop){
16.             end_loop = true;
17.             ++i;
18.             if(
19.                 (inside(last_move.x + (i * _x))) && (inside(last_move.y + (i * _y)))
    && (cells[last_move.x + (i * _x)][last_move.y + (i * _y)] ==
    cells[last_move.x][last_move.y])
20.             ){
21.                 ++count;
22.                 result.push_back(sf::Vector2i(last_move.x + (i * _x), last_move.y +
    (i * _y)));
23.                 end_loop = false;
24.             };
25.             if(
26.                 (inside(last_move.x - (i * _x))) && (inside(last_move.y - (i * _y)))
    && cells[last_move.x - (i * _x)][last_move.y - (i * _y)] ==
    cells[last_move.x][last_move.y]
27.             ){
28.                 ++count;
```

```
29.                 result.push_back(sf::Vector2i(last_move.x - (i * _x), last_move.y -
    (i * _y)));
30.                 end_loop = false;
31.             };
32.             if(count >= 5) status2 *= false;
33.             else status2 *= true;
34.         };
35.     };
36.
37.     check(1, 0);
38.     check(0, 1);
39.     check(1, 1);
40.     check(-1, 1);

41.     status = !status2;

42.     return result;
43. };
```

Code 7. Mã nguồn file `win_check.cpp`.

3. Nhận xét về sản phẩm

3.1. Tự đánh giá sản phẩm

a) Những điều mà nhóm đã làm được

Về cơ bản, nhóm đã xây dựng thành công phần mềm trò chơi Caro hoạt động ổn định trên nền tảng Windows. Về mặt chức năng, chương trình đã đáp ứng đầy đủ các yêu cầu cơ bản được đặt ra trong đề cương ban đầu. Người dùng có thể thực hiện trọn vẹn một quy trình chơi game, từ việc khởi tạo ván đấu mới với các tùy chọn cài đặt tên và lượt đi, đến việc lưu trữ trạng thái ván cờ xuống ổ cứng và tải lại để tiếp tục chơi sau này. Các tính năng bổ trợ như xin đi lại (undo) hoạt động chính xác, giúp người chơi sửa lỗi sai trong quá trình tư duy. Bên cạnh đó, hệ thống giao diện cũng được hoàn thiện với hai chế độ hiển thị sáng và tối, cũng như hỗ trợ chuyển đổi giữa hai ngôn ngữ là tiếng Anh và tiếng Việt, từ đó mang lại trải nghiệm người dùng linh hoạt hơn. Cơ chế kiểm tra thắng thua hoạt động chính xác theo luật Caro tiêu chuẩn, đảm bảo tính công bằng và logic của trò chơi.

Về mặt kỹ thuật, đồ án đã bước đầu áp dụng thành công phương pháp tổ chức mã nguồn theo hướng mô-đun hóa. Việc phân chia chương trình thành các file riêng biệt tương ứng với từng màn hình (như `MAIN_APP`, `game_window`, `prepare_window`,...) kết hợp với việc sử dụng không gian tên đã giúp nhóm quản lý được khối lượng mã nguồn tương đối lớn, giảm thiểu xung đột tên biến và tăng khả năng đọc hiểu mã lệnh. Nhóm cũng đã làm chủ được các kỹ thuật cơ bản của thư viện SFML để xử lý đồ họa 2D, sự kiện chuột, bàn phím và quản lý cửa sổ. Hệ thống lưu trữ file save được xây dựng dựa trên đối tượng `std::filesystem`, cho phép thực hiện các thao tác đọc ghi dữ liệu và sắp xếp danh sách file save theo thời gian thực.

b) Những hạn chế cần khắc phục

Bên cạnh những kết quả khả quan đã đạt được, nhóm nhận thấy hệ thống vẫn còn tồn tại một số hạn chế về mặt thuật toán và cấu trúc mã nguồn cần được khắc phục.

- Thứ nhất, thuật toán kiểm tra điều kiện chiến thắng (`win_check`) tuy đã hoạt động nhưng chưa đạt độ tối ưu tuyệt đối; trong một số trường hợp cụ thể, thuật toán trả về kết quả bao gồm cả các ô dư thừa không thuộc chuỗi 5 ô thắng, gây ảnh hưởng nhỏ đến hiệu ứng hiển thị kết quả cuối ván. Hơn nữa, thuật toán load game sẽ bị lỗi nếu gặp phải các tập tin lưu trữ không được cấu trúc đúng.

• Thứ hai, về mặt chức năng, đồ án hiện tại mới chỉ hoàn thiện chế độ hai người chơi và chưa tích hợp được chế độ chơi đơn với máy (một người chơi) do nhóm chưa kịp xây dựng thuật toán tự động đánh cờ và giao diện chuẩn bị cho tính năng một người chơi.

• Đồng thời, hệ thống âm thanh vẫn chưa được hiện thực hóa; trò chơi hiện tại hoàn toàn thiếu vắng các hiệu ứng âm thanh tương tác (như âm thanh khi đặt quân, âm thanh thông báo chiến thắng) cũng như nhạc nền, làm giảm phần nào tính sinh động của trải nghiệm người dùng.

• Về khía cạnh kỹ thuật, tập tin `game_window.cpp` do được phát triển trong giai đoạn đầu nên cấu trúc tổ chức mã lệnh còn thiếu tính hệ thống và chưa được phân tách rõ ràng như các thành phần khác. Sự thiếu đồng bộ trong mã nguồn này đôi khi dẫn đến một số lỗi hiển thị nhỏ trên giao diện.

• Ngoài ra, tính linh hoạt của phần mềm còn hạn chế do thiếu tính năng cài đặt trong thời gian thực (in-game settings); người chơi hiện tại chưa thể thay đổi các thông tin của ván đấu (như tên người chơi, màu sắc) ngay trong quá trình chơi mà buộc phải khởi tạo lại ván mới từ đầu.

3.2. Định hướng cải thiện trong lần nộp sau

Từ những đánh giá về kết quả và hạn chế của phiên bản hiện tại, nhóm đề xuất các định hướng cụ thể để nâng cấp và hoàn thiện đồ án trong các giai đoạn phát triển tiếp theo như sau:

• Hoàn thiện thuật toán và logic trò chơi:

  ▫ Tập trung nghiên cứu và cài đặt *thuật toán Minimax* kết hợp với *kỹ thuật cắt tỉa Alpha-Beta* để xây dựng chế độ chơi đơn (một người chơi). Mục tiêu là tạo ra một đối thủ máy có khả năng đánh chặn và tấn công thông minh theo ba mức độ dễ/trung bình/khó, mang lại thử thách thực sự cho người chơi.

  ▫ Cải tiến thuật toán kiểm tra thắng hiện tại để xác định chính xác chuỗi 5 quân liên tiếp mà không bao gồm các ô thừa.

  ▫ Xây dựng một thuật toán kiểm tra và báo lỗi trong trường hợp tập tin load game là một tập tin không đúng cấu trúc.

• Tái cấu trúc và chuẩn hóa mã nguồn:

  ▫ Thực hiện tái cấu trúc toàn bộ file `game_window.cpp` theo hướng tách biệt hoàn toàn giữa giao diện và logic. Áp dụng triệt để các hàm tiện ích từ thư viện templates để đồng bộ hóa phong cách lập trình với các màn hình khác.

  ▫ Bổ sung thêm các hàm tiện ích khác vào thư viện templates để tăng tốc quá trình xây dựng trò chơi hơn nữa.

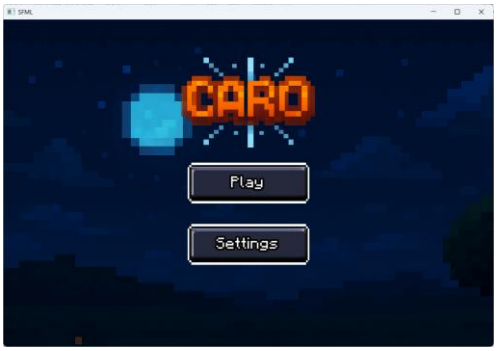
• Cải thiện, bổ sung giao diện:

  ▫ Tích hợp đối tượng `sf::Audio` của SFML để thêm nhạc nền và các hiệu ứng âm thanh tương tác (khi đặt quân, khi bấm nút, khi chiến thắng), giúp trò chơi trở nên sinh động hơn.

  ▫ Phát triển tính năng In-game Settings, xây dựng màn hình cài đặt nổi (Overlay) ngay trong ván đấu, cho phép người chơi thay đổi thông tin hoặc cấu hình mà không cần thoát ra menu chính.

Báo cáo của nhóm xin hết. Chúng em cảm ơn Thầy đã đọc.

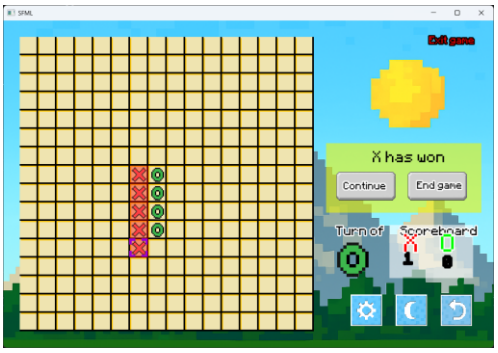
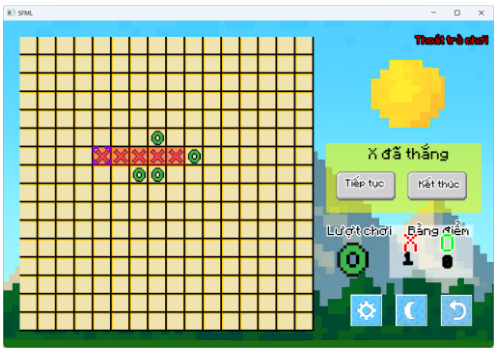
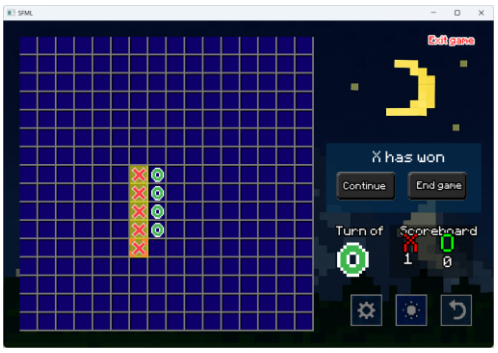
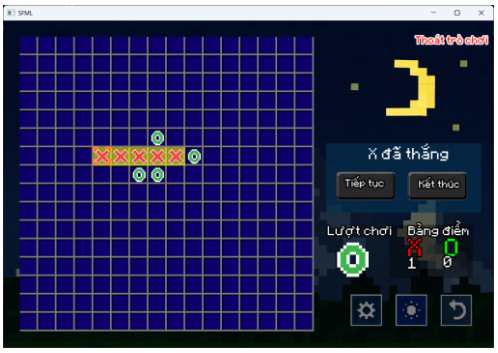
4. Phụ lục: Một số hình ảnh

	Tiếng Anh	Tiếng Việt
Sáng		
Tối		

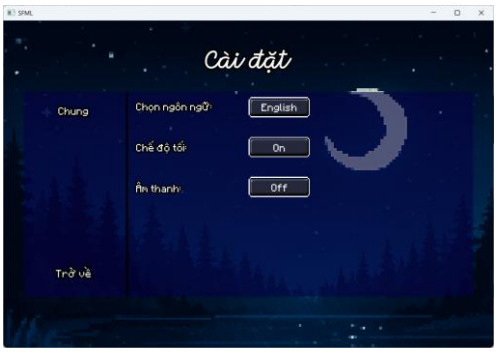
Hình 1. Màn hình chính.

	Tiếng Anh	Tiếng Việt
Sáng		
Tối		

Hình 2. Một màn trong màn hình chuẩn bị.

	Tiếng Anh	Tiếng Việt
Sáng		
Tối		

Hình 3. Màn hình game.

	Tiếng Anh	Tiếng Việt
Sáng		
Tối		

Hình 4. Màn hình cài đặt.