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

Nguyễn Quang Thông
Nguyễn Anh Vũ
Võ Nhật Phước
Hoàng Trung Nguyên

XÂY DỰNG GAME CỜ CARO

ĐỒ ÁN MÔN HỌC
KĨ THUẬT LẬP TRÌNH

Thành Phố Hồ Chí Minh, Tháng 04 năm 2023

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

ĐỒ ÁN MÔN HỌC
KĨ THUẬT LẬP TRÌNH
ĐỀ TÀI:
XÂY DỰNG GAME CỜ CARO

Nhóm 11

22127401 - Nguyễn Quang Thông

22127298 - Hoàng Trung Nguyên

22127339 - Võ Nhật Phước

22127463 - Nguyễn Anh Vũ

Giáo Viên Hướng Dẫn: Trương Toàn Thịnh

Thành Phố Hồ Chí Minh, Tháng 04 năm 2023

Lời cảm ơn

Lời đầu tiên, nhóm chúng em xin gửi lời cảm ơn sâu sắc nhất đến Thầy Trương Toàn Thịnh. Ngoài những cố gắng của các thành viên trong nhóm, nhóm chúng em sẽ không hoàn thiện đồ án này nếu không có sự chỉ bảo của Thầy. Thầy luôn giúp đỡ tận tình, cung cấp tài liệu để nhóm chúng em có thể hoàn thành đồ án trọn vẹn. Trong suốt quá trình làm đồ án, nhóm chúng em ắt hẳn sẽ có những sai sót kính mong thầy bỏ qua. Thông qua đồ án, nhóm chúng em đã học được một số kiến thức mới, có thêm kinh nghiệm để có thể áp dụng sau này.

Cuối cùng, nhóm chúng em kính chúc thầy nhiều sức khỏe, luôn thành công trên nhiều lĩnh vực để có thể cống hiến cho nền khoa học và giáo dục nước nhà.

Mục lục

Danh sách hình

Danh sách bảng

Chương 1.

Tổng quan về trò chơi

1.1. Giới thiệu về trò chơi

1.1.1. Trò chơi Cờ Caro (Gomoku)

Cờ Caro [1] hay còn gọi là Gomoku [2] là một trò chơi đối kháng có tính chiến thuật cao. Trò chơi được chơi trên một bàn cờ vuông với kích thước tiêu chuẩn là 15x15 ô (trong trò chơi này, do một vài giới hạn kĩ thuật nên bàn cờ sẽ có kích thước là 13x13).

Trò chơi có nhiều phiên bản khác nhau với các luật chơi khác nhau. Trong trò chơi này, chúng em đã sử dụng luật chơi như sau:

- Hai người chơi lần lượt đặt các quân cờ của mình trên bàn cờ.
- Người chơi nào đặt được 5 quân cờ liên tiếp theo chiều ngang, dọc hoặc chéo sẽ thắng cuộc.
- Nếu không có ai thắng khi không còn nước đi khả thi thì trò chơi kết thúc với kết quả hòa.

Ngoài ra, để tăng tính đa dạng và thêm phần thú vị cho trò chơi, chúng em còn bổ sung thêm chế độ chơi mới: tốc độ (Rush). Trong chế độ này, người chơi cần phải tập trung hơn, suy nghĩ nhanh hơn để chiến thắng. Luật chơi như sau:

- Người chơi sẽ có một khoảng thời gian nhất định để suy nghĩ và thực hiện nước đi của cả ván đấu.
- Nếu người chơi không đánh được nước đi trong thời gian giới hạn thì người chơi đó sẽ thua cuộc.
- Có ba khoảng thời gian khác nhau để người chơi lựa chọn: 1 phút, 5 phút và 15 phút.

1.1.2. Các yêu cầu về tính năng

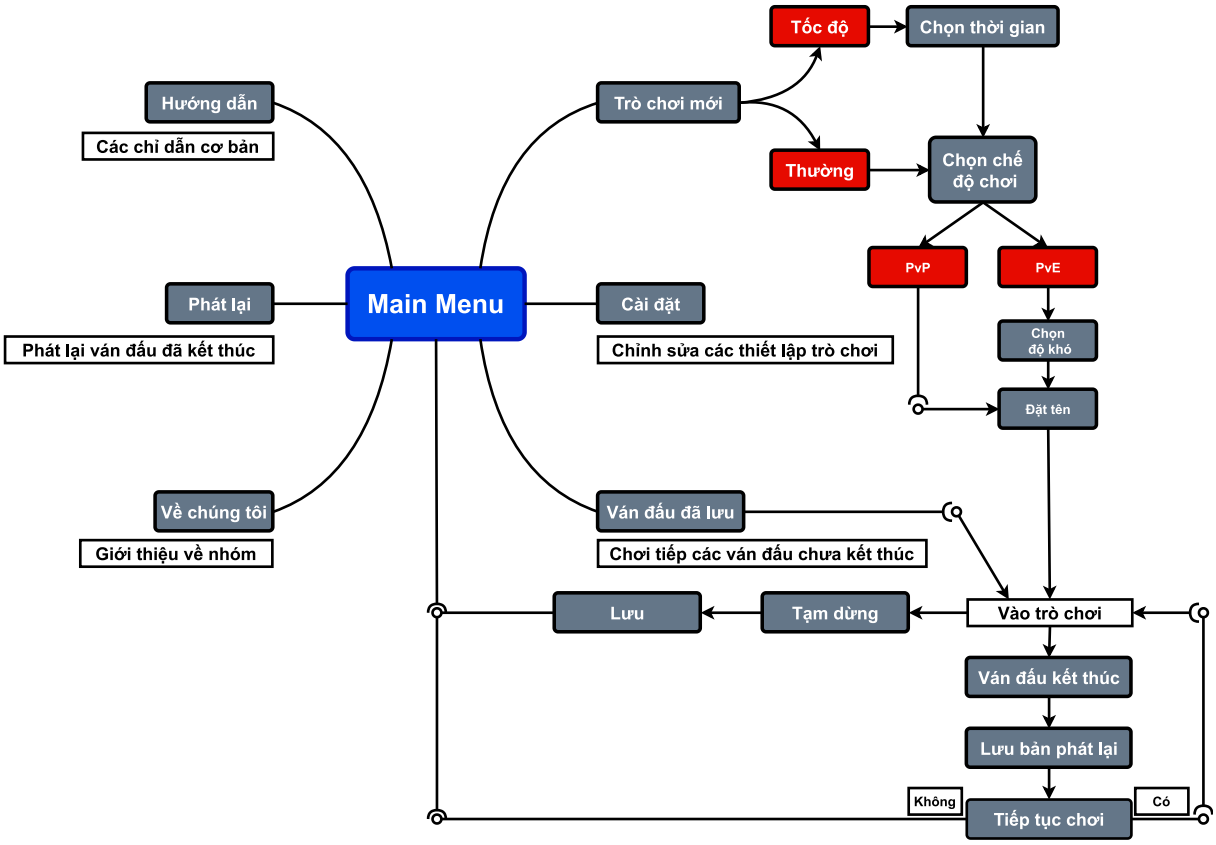
- Có thể lưu, tải trò chơi đang chơi
- Nhận biết được thắng, thua, hòa
- Xử lí hiệu ứng thắng, thua, hòa
- Xử lí giao diện màn hình khi chơi
- Xử lí màn hình chính
- Game có nhiều ngôn ngữ, người chơi có thể thêm được ngôn ngữ mới
- Có thể tải được các theme(chủ đề) bên ngoài
- Người chơi có thể bât/tắt các tính năng cơ bản, có thể lưu lại các thiết lập
- Có thể lưu và phát lại các trò chơi đã hoàn thành
- Có nhiều chế độ chơi, chơi với máy

1.1.3. Thông tin chung về trò chơi

Tên trò chơi	Cờ Caro
Môi trường phát triển và thử nghiệm	Visual Studio 2022
Khả dụng trên nền tảng	Windows
Source code	https://github.com/thng292/CaroGame

Bảng 1: Thông tin chung về trò chơi

1.1.4. Sơ đồ di chuyển của trò chơi



Hình 1: Sơ đồ di chuyển của trò chơi

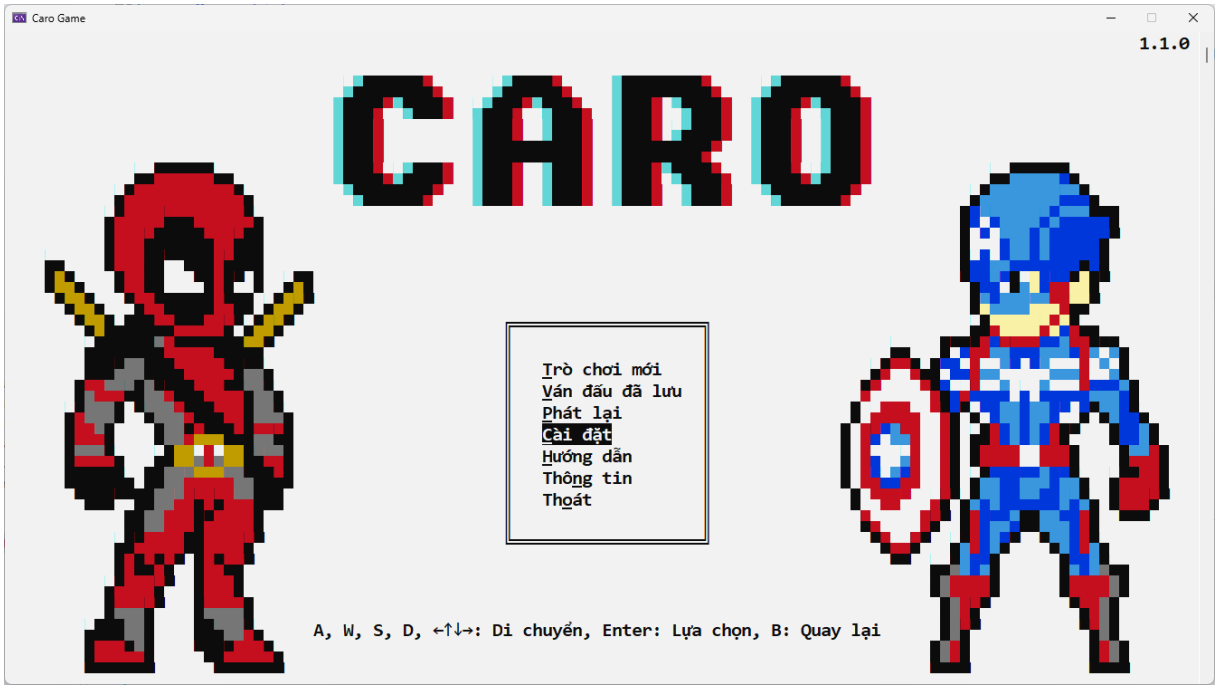
1.2. Mô tả về các tính năng của trò chơi

1.2.1. Menu chính

Đây là "Menu Chính" của trò chơi. Người chơi có thể chọn các chức năng khác nhau bằng cách chọn vào các mục tương ứng. Các mục chức năng bao gồm:

- Trò chơi mới: Bắt đầu một ván đấu mới.
- Ván đấu đã lưu: Tải lại một ván đấu đã lưu.
- Phát lại: Phát lại một ván đấu đã hoàn thành.
- Cài đặt: Tùy chỉnh các thiết lập của trò chơi.
- Hướng dẫn: Hướng dẫn cách chơi và cách tương tác với các thành phần giao diện của trò chơi.
- Thông tin: Thông tin về tác giả.
- Thoát: Thoát trò chơi.

Phía trên bên phải của "Menu Chính" là phiên bản của trò chơi.

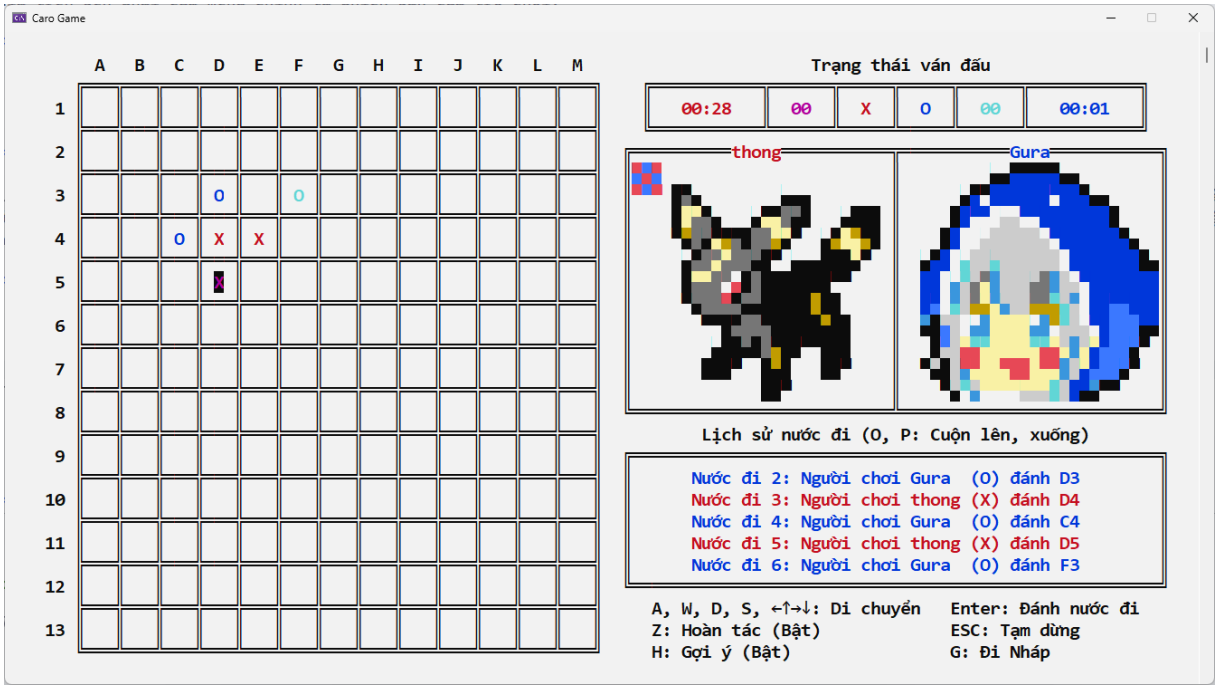


Hình 2: Menu chính

1.2.2. Giao diện và các tính năng đặc biệt khi chơi

1.2.2.1. Giao diện màn hình khi chơi

Giao diện màn hình khi chơi trò chơi là một phần rất quan trọng của trò chơi vì nó là nơi mà người chơi dành nhiều thời gian tương tác với trò chơi nhất. Khi thiết kế màn hình này, chúng em đã phải suy nghĩ rất kĩ, trên màn hình sẽ hiện thông tin nào, hiện ra sao và sắp xếp như thế nào. Cuối cùng, chúng em đã thiết kế nó như sau:



Hình 3: Giao diện màn hình khi chơi

Trong trò chơi này, giao diện màn hình khi chơi được chia thành 3 phần chính:

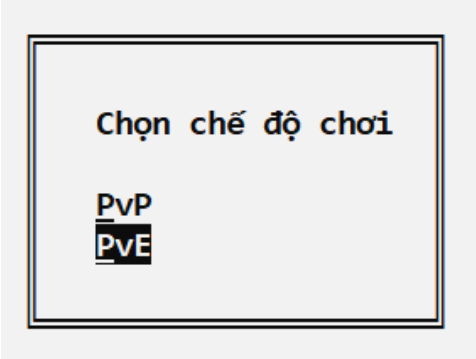
- Phần bàn cờ (bên trái): Khu vực chơi chính
- Phần trạng thái trận đấu (phía trên bên phải): là nơi hiển thị các thông tin về ván đấu như: thời gian đã chơi (chế độ thường), thời gian còn lại (chế độ tốc độ), số trận đấu đã thắng.
- Phần thông tin người chơi (dưới phần trạng thái trận đấu): là nơi hiển thị các thông tin về người chơi như: tên người chơi, ảnh đại diệnm lượt hiện tại của người nào.
- Phần lịch sử nước đi (dưới phần thông tin người chơi): là nơi hiển thị các nước đi đã được thực hiện trong trận đấu. Có thể cuộn lên, xuống để xem toàn bộ lịch sử nước đi.
- Phần hướng dẫn (phía dưới bên phải): là nơi hiển thị các nút để tương tác với trò chơi.

1.2.2.2. Các tính năng đặc biệt khi chơi

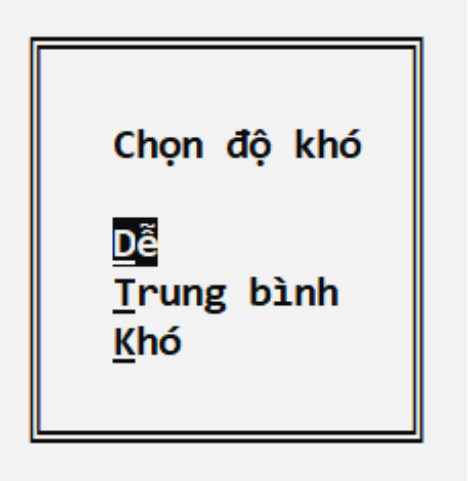
Để tăng thêm trải nghiệm cho người chơi, chúng em đã thêm vào một số tính năng đặc biệt.

1.2.2.2.1. Chơi với máy

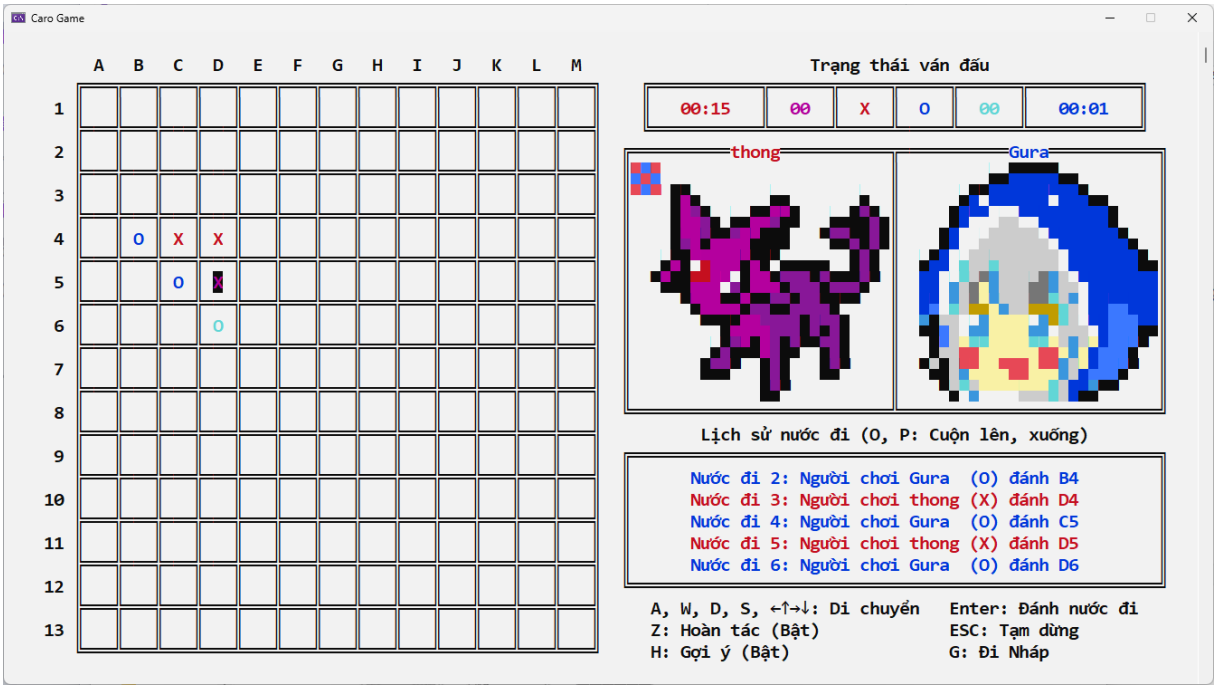
Chơi với máy là một tính năng vô cùng hữu ích khi người chơi không tìm được người chơi khác để chơi cùng. Có 3 độ khó: Dễ, Trung bình, Khó. Độ khó của máy được xác định bằng độ sâu tìm kiếm của thuật toán. Độ khó cao thì máy sẽ chơi tốt hơn, nhưng sẽ mất nhiều thời gian để tính toán hơn. Để chơi với máy, khi chọn chế độ chơi, người chơi cần chọn chế độ "PvE". Sau đó, người chơi chọn độ khó của máy. Máy tính sẽ có ảnh đại diện đặc biệt để phân biệt với người chơi.



Hình 4: Chọn chế độ chơi với máy



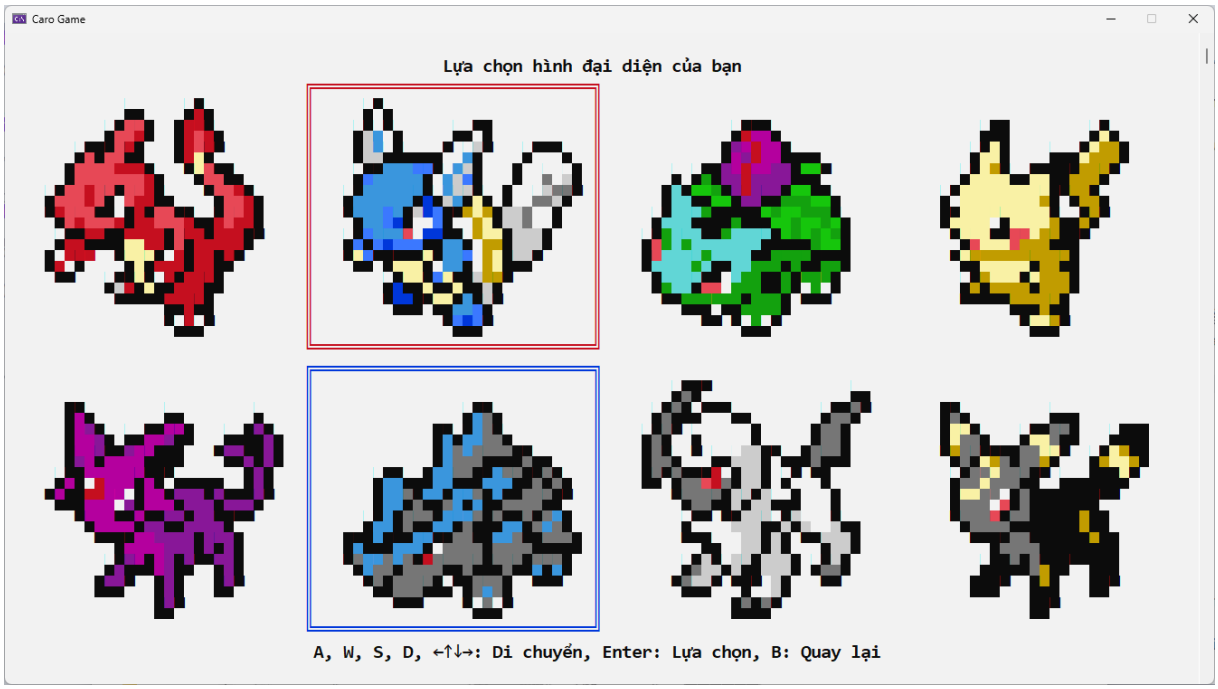
Hình 5: Chọn độ khó của máy



Hình 6: Giao diện màn hình khi chơi với máy

1.2.2.2.2. Chọn ảnh đại diện

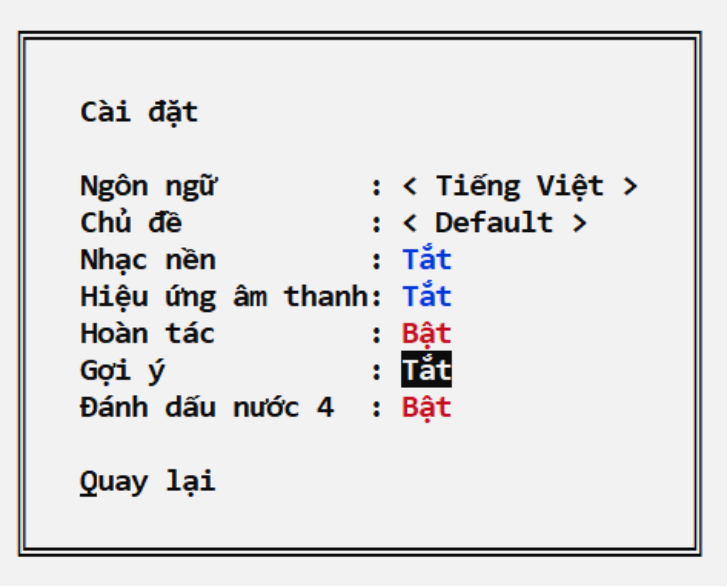
Người chơi có thể chọn ảnh đại diện của mình khi tạo trò chơi mới. Ảnh đại diện sẽ được hiển thị trên màn hình khi chơi, trong phần "thông tin người chơi". trò chơi có tổng cộng 8 ảnh đại diện cho người chơi chọn lựa và một ảnh đại diện đặc biệt dành cho máy. Các ảnh đại diện được lấy cảm hứng từ các nhân vật hoạt hình nổi tiếng để tạo cảm giác thân thuộc cho người chơi.



Hình 7: Chọn ảnh đại diện cho 2 người chơi

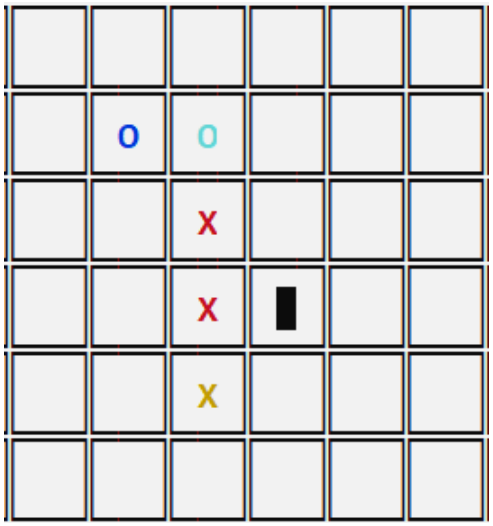
1.2.2.2.3. Gợi ý

Tính năng này sử dụng AI để gợi ý cho người chơi nước đi tốt tiếp theo. Người chơi có thể kiểm tra chức gợi ý có được bật không bằng cách kiểm tra góc dưới bên phải màn hình chơi xem phần hoàn tác đã bật hay chưa. Mặc định thì tính năng gợi ý sẽ được bật sẵn, nhưng người chơi có thể tắt nó đi trong phần cài đặt.



Hình 8: Tắt chức năng gợi ý

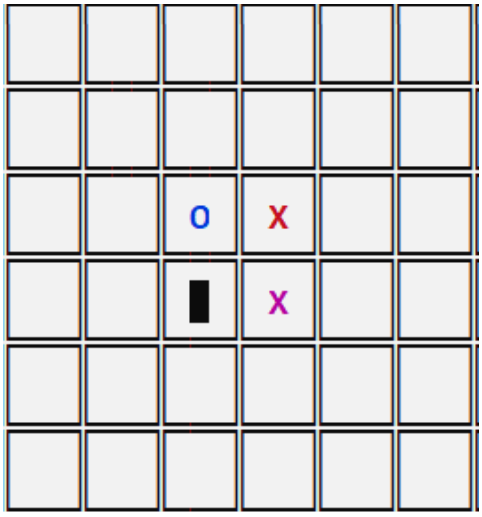
Khi cần, người chơi có thể nhấn phím `H` để sử dụng gợi ý. Nước đi gợi ý sẽ được làm nổi bật lên.



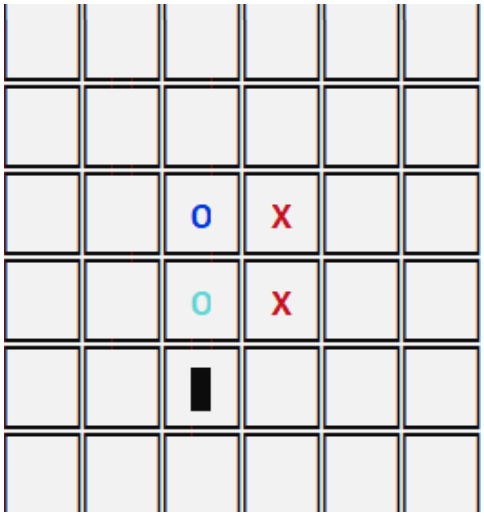
Hình 9: Nước đi gợi ý

1.2.2.2.4. Nổi bật nước mới đi

Nước mới đi sẽ được làm nổi bật lên giúp người chơi dễ dàng nhận biết và theo dõi.



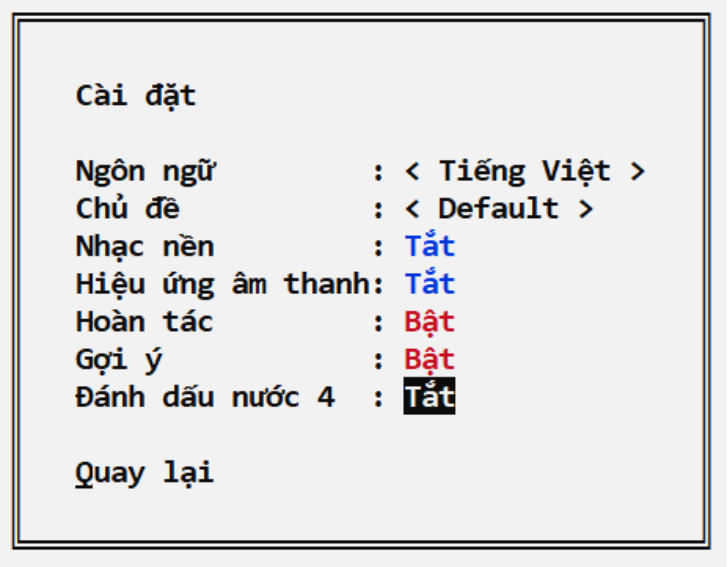
Hình 10: Nước mới đi bên X



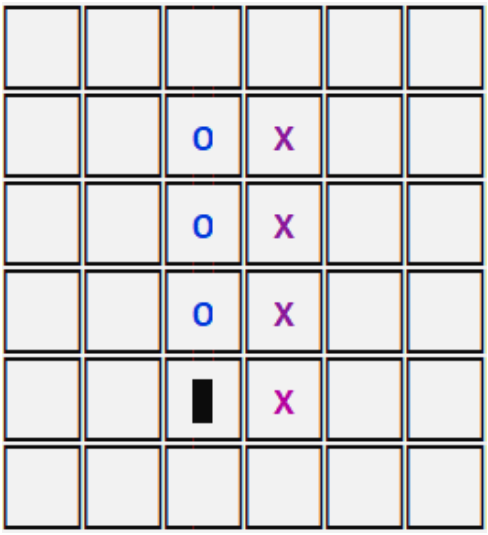
Hình 11: Nước mới đi bên O

1.2.2.2.5. Cảnh báo nước 4

Khi 4 quần cờ liên tiếp nối với nhau tạo thành một đường, các quân cờ đó sẽ được làm nổi bật bởi màu khác để người chơi có thể dễ dàng nhận biết từ đó đưa ra các nước cờ ngăn chặn đối phương. Mặc định tính năng này sẽ được bật sẵn, nhưng người chơi có thể tắt nó đi trong phần cài đặt.



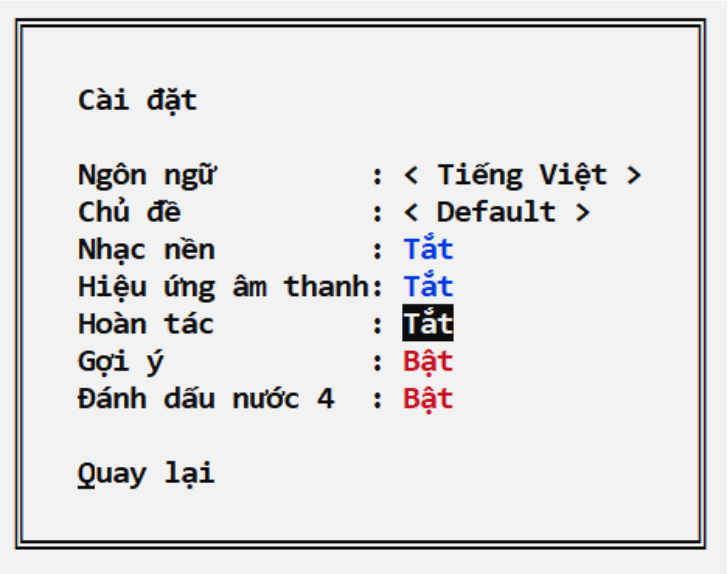
Hình 12: Cài đặt cảnh báo nước 4



Hình 13: Cảnh báo các quân cờ nước 4

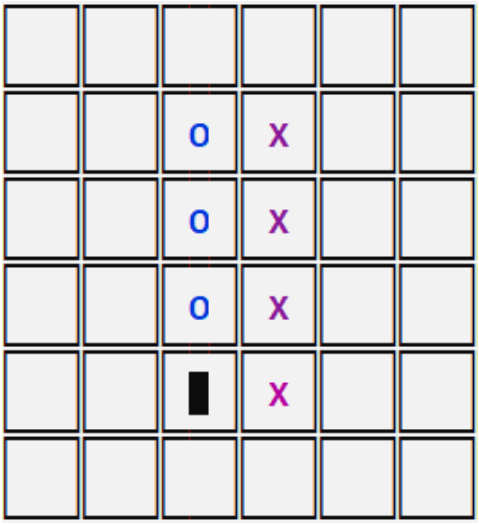
1.2.2.2.6. Hoàn tác nước đi

Đôi khi người chơi, vì nhiều lí do, lỡ ấn đánh nhầm nước và muốn hoàn tác lại nước đi vừa thực hiện. Khi đó tính năng hoàn tác nước đi sẽ trở nên rất hữu ích. Mặc định tính năng này sẽ được bật sẵn, nhưng người chơi có thể tắt nó đi trong phần cài đặt. Người chơi có thể kiểm tra chức năng hoàn tác nước đi có được bật không bằng cách kiểm tra góc dưới bên phải màn hình chơi xem tính năng hoàn tác có được bật hay chưa.

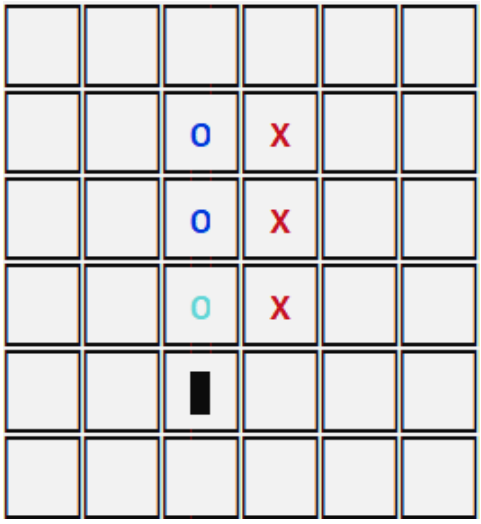


Hình 14: Cài đặt chức năng hoàn tác nước đi

Khi cần, người chơi có thể nhấn phím `Z` để hoàn tác nước đi. Nước đi sẽ được hoàn tác lại và quân cờ sẽ được đưa về trạng thái trước khi đánh nước đó.



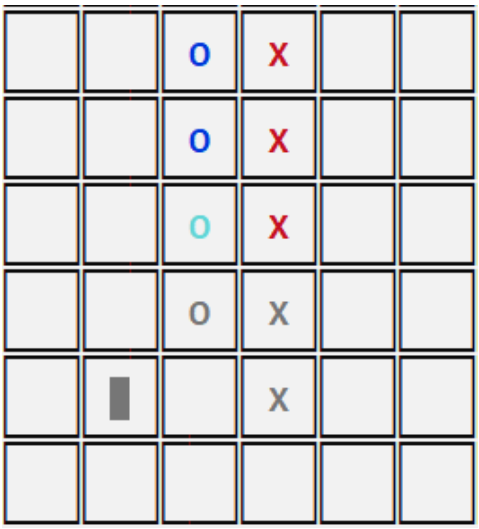
Hình 15: Trước khi hoàn tác nước đi



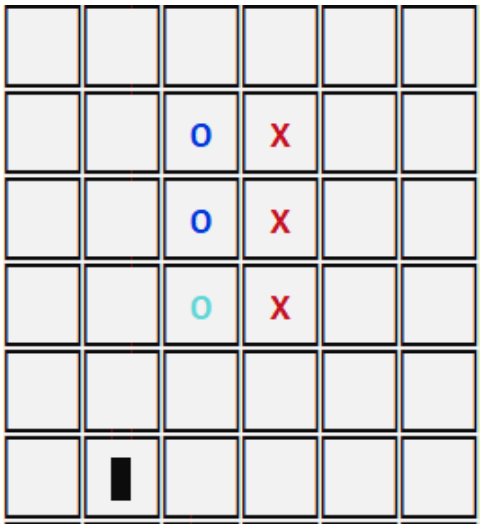
Hình 16: Sau khi hoàn tác nước đi

1.2.2.2.7. Đi nháp

Chức năng này cho phép người chơi đi các nước đi mà không ảnh hưởng tới diễn biến ván cờ, cho phép người chơi thoải mái tính toán các nước đi tiếp theo. Khi bật đi nháp, con trỏ sẽ chuyển màu, người chơi có thể tùy thích đi các nước đi. Các nước đi nháp sẽ có màu khác với màu của quân cờ hiện tại. Khi tắt đi nháp, các nước đi nháp sẽ bị xóa, trả lại ván cờ như lúc trước khi bật đi nháp.



Hình 17: Các nước đi nháp

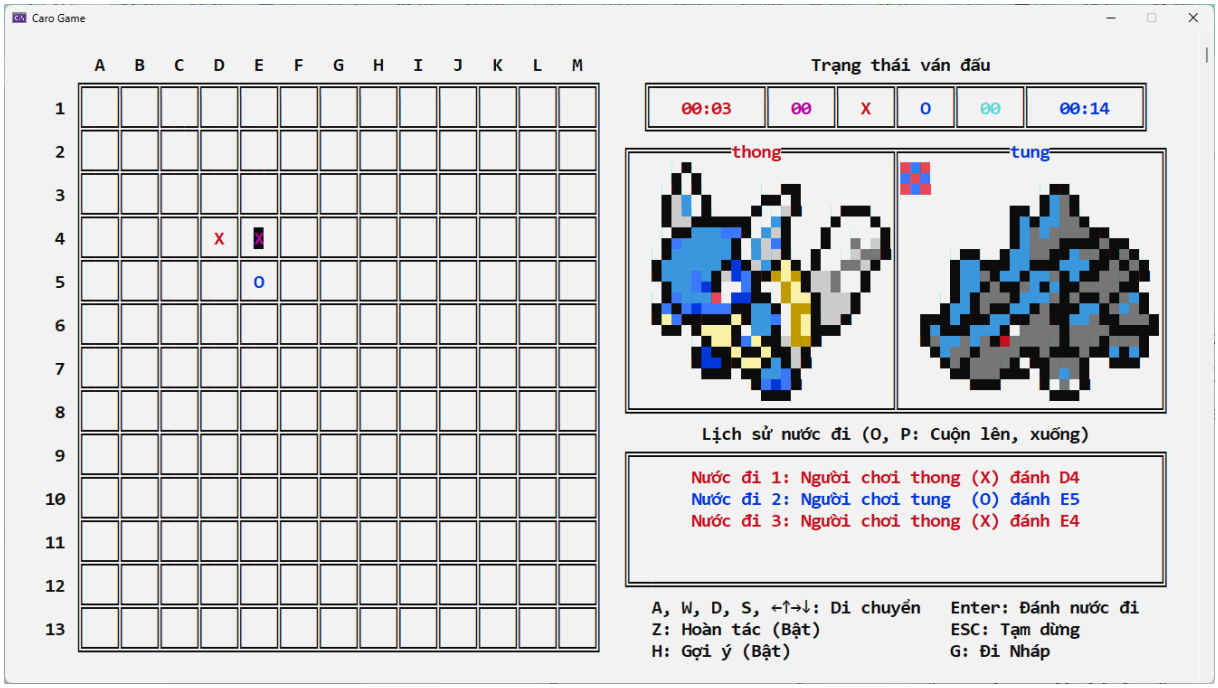


Hình 18: Khi tắt đi nháp

1.2.3. Các chế độ chơi

1.2.3.1. Chế độ chơi Thường

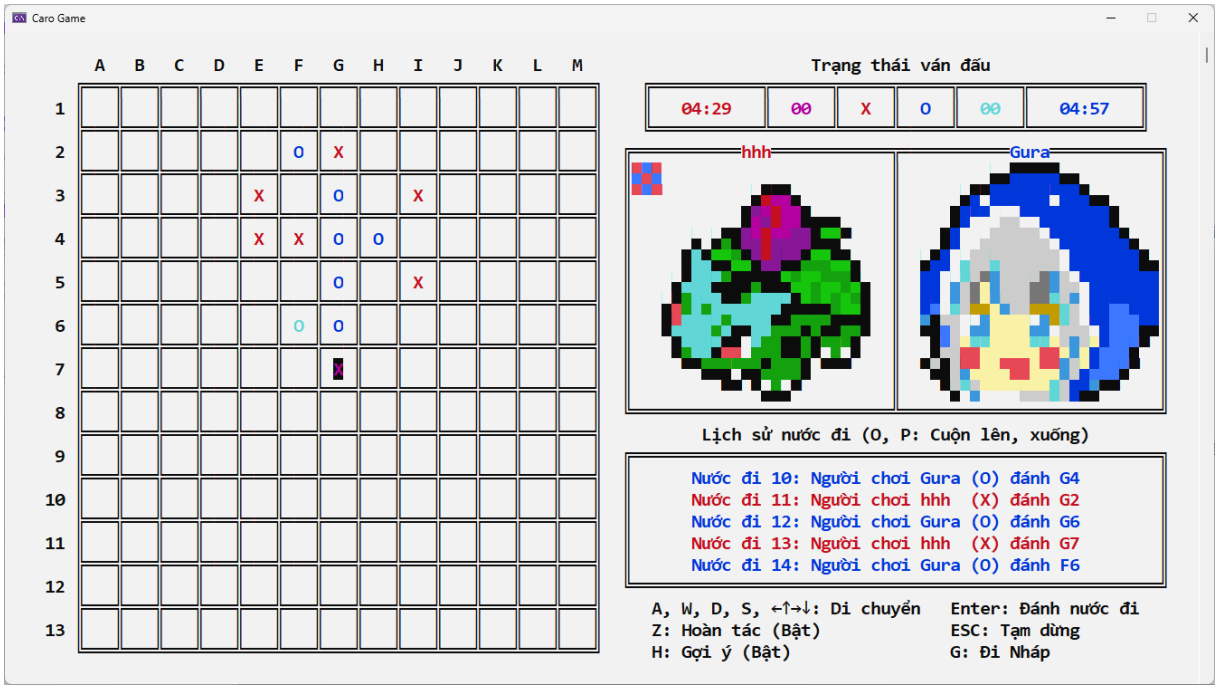
Chế độ không thể thiếu của trò chơi. Cổ điển và đơn giản. Ở chế độ này, trò chơi hiển thị thời gian mà người chơi sử dụng để suy nghĩ ở phần trạng thái ván đấu.



Hình 19: Giao diện màn hình khi chơi chế độ Thường

1.2.3.2. Chế độ chơi Rush

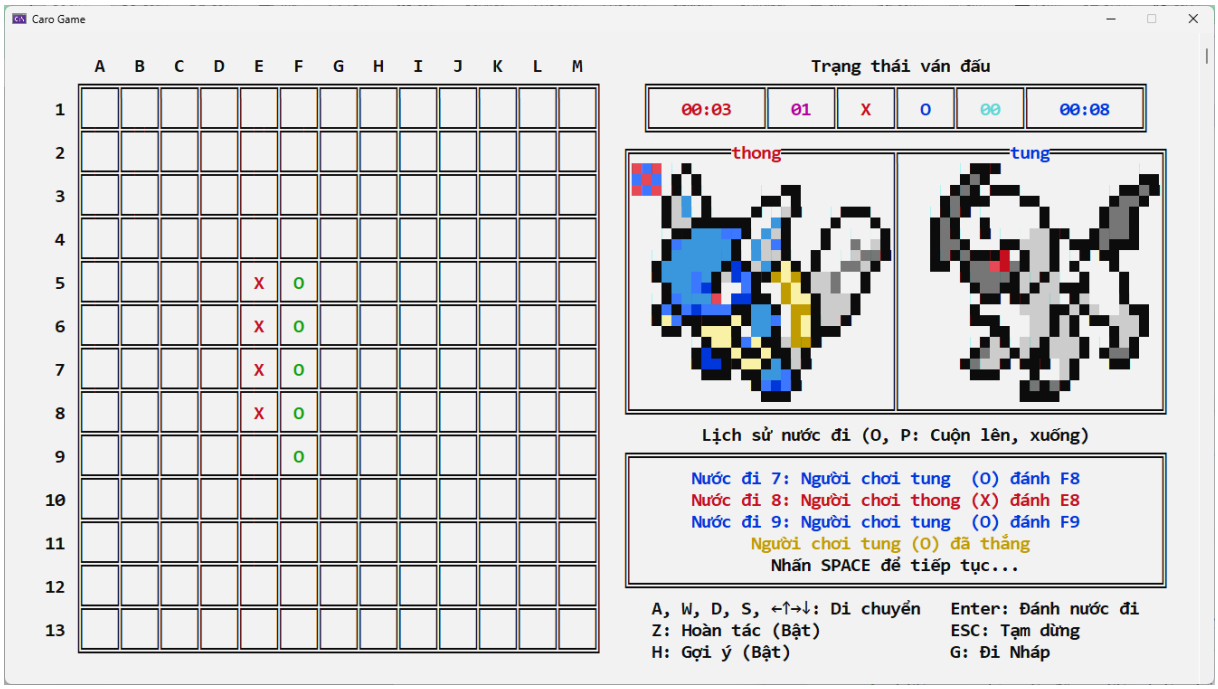
Để thêm phần thú vị cho trò chơi, chúng em đã thêm vào chế độ Rush. Ở chế độ này, trò chơi hiển thị thời gian còn lại của người chơi ở phần trạng thái ván đấu.



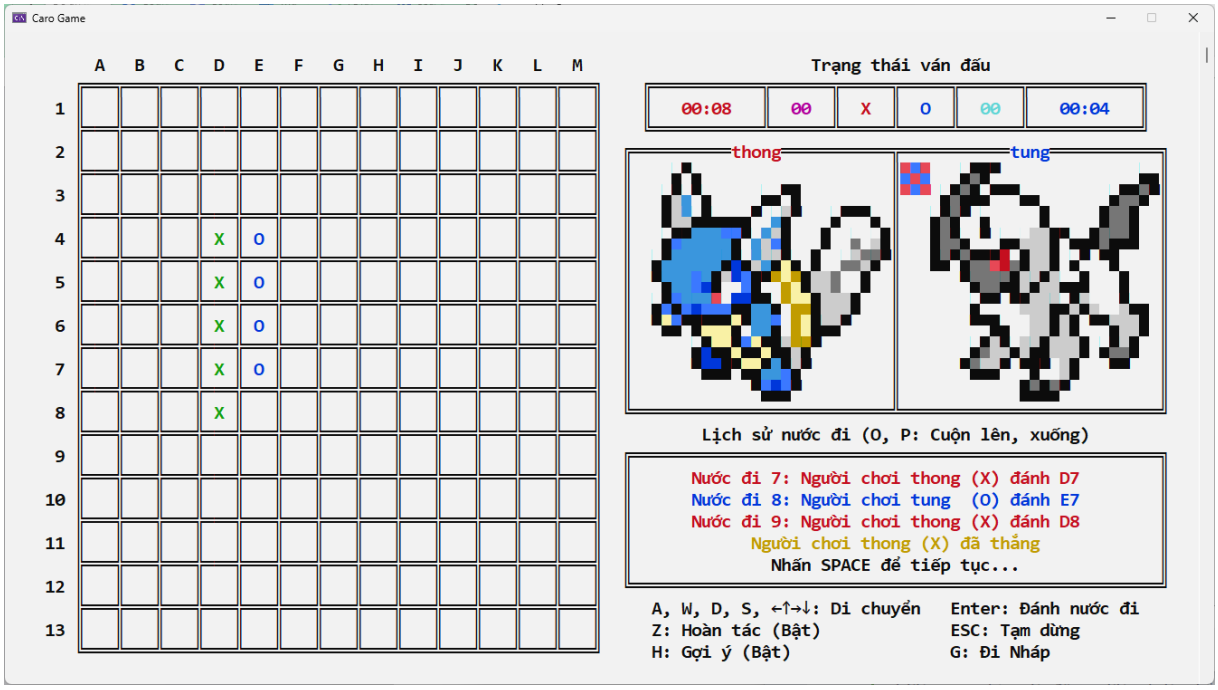
Hình 20: Giao diện màn hình khi chơi chế độ Rush

1.2.4. Xử lí, hiệu ứng thắng, thua, hòa

Nếu có người thắng ván đấu, trò chơi sẽ làm nổi bật lên chuỗi 5 quân cờ của người thắng, sau đó, trò chơi sẽ đợi người dùng nhấn phím `Space` để chuyển sang màn hình kết quả ván đấu.

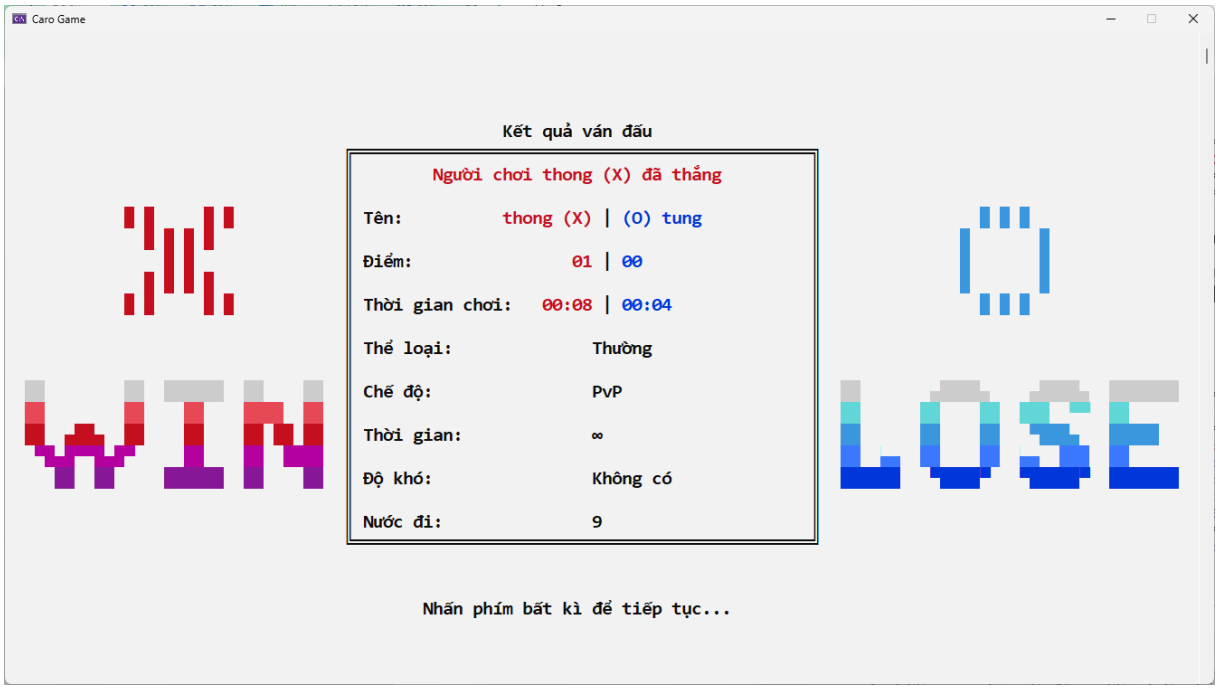


Hình 21: Giao diện màn hình khi O thắng



Hình 22: Giao diện màn hình khi X thắng

Sau khi nhấn `Space`, người chơi sẽ được chuyển sang màn hình kết quả ván đấu. Trong màn hình này, người chơi có thể xem lại thông tin trận đấu như: số nước đi, thời gian đã chơi, thời gian còn lại, người chơi đã thắng bao nhiêu trận.

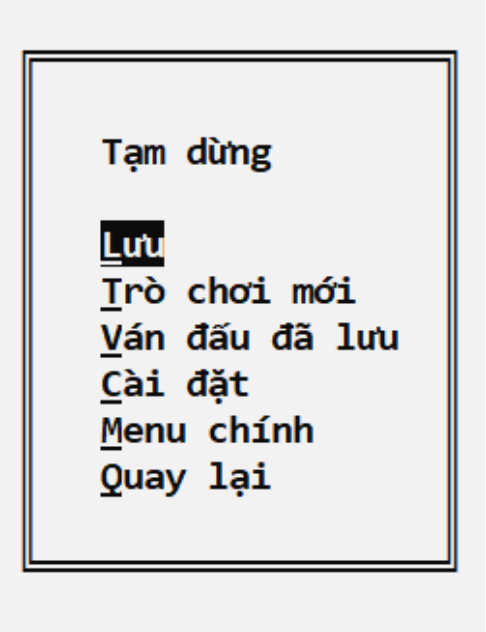


Hình 23: Giao diện màn hình kết quả ván đấu

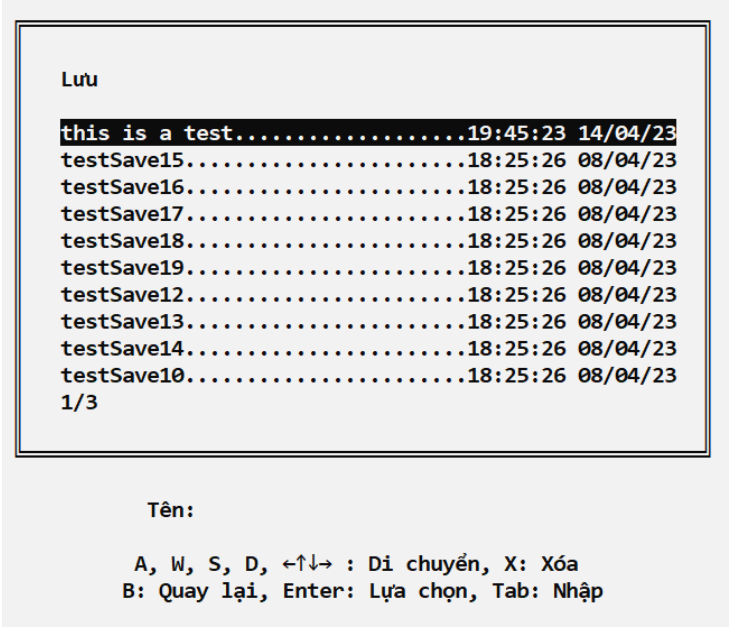
1.2.5. Lưu/tải ván đấu đang chơi, phát lại ván đấu đã chơi xong

1.2.5.1. Lưu ván đấu đang chơi

Người chơi có thể lưu lại ván đấu mình đang chơi bằng cách truy cập vào menu "Tạm dừng" (ấn phím `ESC`) và chọn mục "Lưu trò chơi" để mở menu "Lưu". Các bản lưu của người chơi cùng thời gian lưu sẽ hiện ra. Các bản lưu được sắp xếp theo thứ tự giảm dần của thời gian lưu, nếu người chơi có quá nhiều bản lưu thì trò chơi sẽ chia thành nhiều trang, người chơi có thể chuyển qua lại giữa các trang. Ở phía dưới các bản lưu là trang hiện hiên tại và tổng số trang. Phía dưới menu là nơi để người chơi nhập tên cho bản lưu, khi người chơi nhập tên, trò chơi sẽ đưa các bản lưu có tên liên quan tới đầu vào của người chơi. Sau khi nhấn lưu, trò chơi sẽ thông báo lưu thành công hay thất bại.

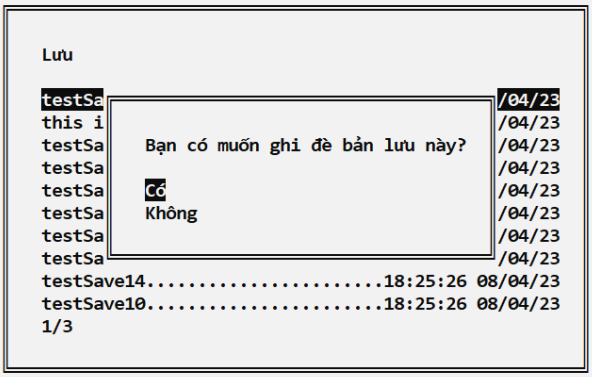


Hình 24: Menu "Tạm dừng"

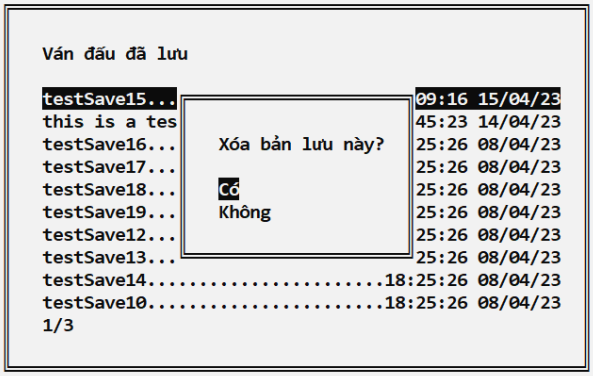


Hình 25: Menu "Lưu"

Trong trường hợp người chơi nhập tên bản lưu trùng với tên của một bản lưu khác, trò chơi sẽ thông báo cho người chơi và hỏi người chơi có muốn ghi đè lên bản lưu cũ hay không. Ngoài ra, người chơi cũng có thể xóa bản lưu.



Hình 26: Thông báo ghi đè bản lưu

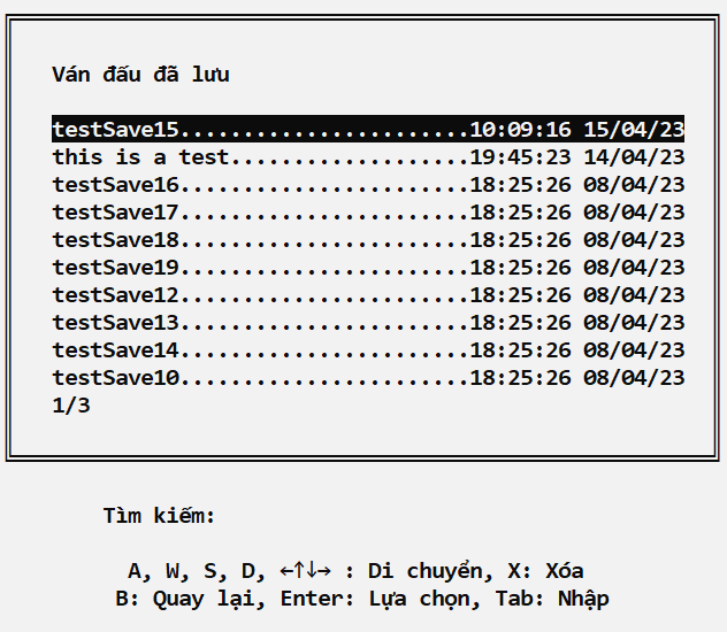


Hình 27: Thông báo xóa bản lưu

1.2.5.2. Tải trò chơi đã lưu

Người chơi có thể tải ván đấu đã lưu bằng cách truy cập vào "Menu Chính" và chọn mục "Ván đấu đã lưu". Giao diện của menu này tương tự như menu "Lưu", để tải bản lưu, người chơi tìm và chọn bản lưu đó. Để thuận tiện hơn cho người chơi, chúng em đã thêm vào ô tìm kiếm ở dưới menu, khi người chơi nhập vào ô tìm kiếm, trò chơi sẽ đưa các bản lưu có tên liên quan tới đầu vào của người chơi lên trên. Khi tải bản lưu, nếu việc tải bản lưu bị lỗi, trò chơi sẽ thông báo lỗi

và người chơi có thể chọn bản lưu khác. Nếu tải thành công, trò chơi sẽ chuyển sang màn hình chơi và người chơi có thể tiếp tục ván đấu. Ngoài ra người chơi cũng có thể xóa bản lưu như ở menu "Lưu".

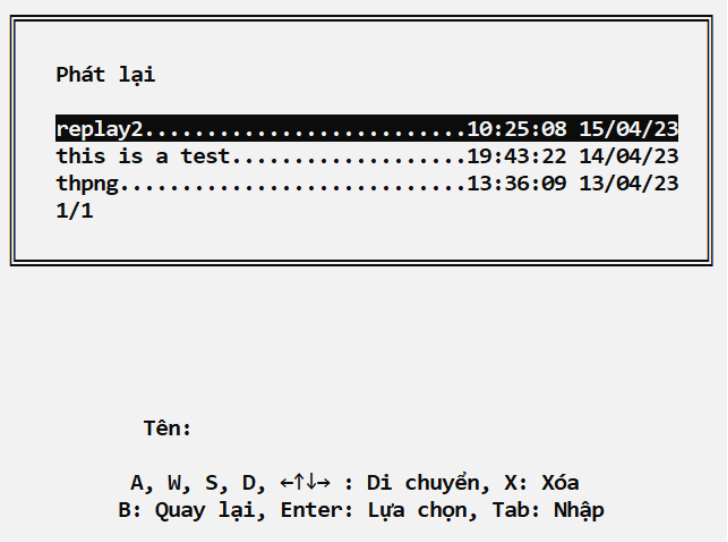


Hình 28: Menu "Ván đấu đã lưu"

1.2.5.3. Phát lại trò chơi đã chơi xong

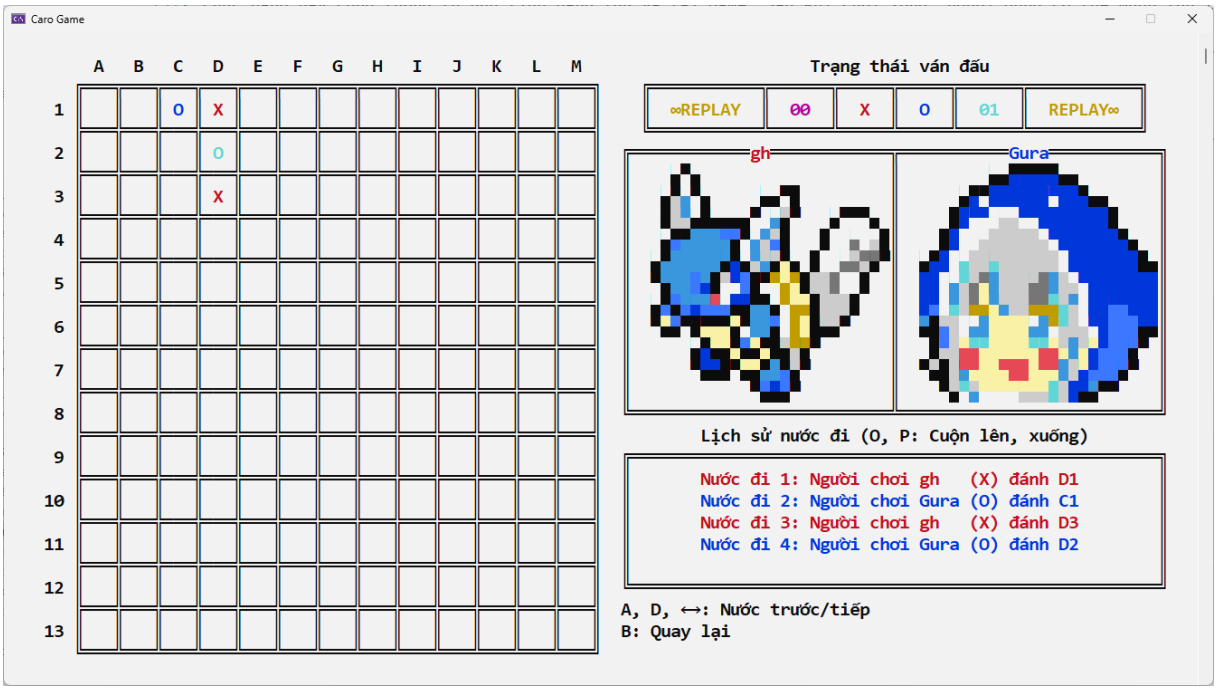
Chức năng này cũng tương tự như tính năng lưu và tải trò chơi. Sau khi chơi xong, người chơi có thể muốn lưu lại quá trình chơi của ván đấu để sau này phát lại, nên chúng em đã thêm bào tính năng lưu và phát lại các ván đấu đã chơi. Sau khi kết thúc ván đấu, trò chơi sẽ hỏi người chơi có muốn lưu bản phát lại hay không. Nếu chọn có, người chơi sẽ được dẫn tới menu "Phát lại", menu này tương tự như menu "Ván đấu đã lưu", người chơi có thể lưu, xóa các bản phát lại của mình.

Để phát bản phát lại, tại "Menu Chính", người chơi chọn vào mục "Phát lại" và chọn bản phát lại cần phát. Nếu tải bản phát lại thành công, trò chơi sẽ chuyển sang màn hình phát lại và người chơi có thể phát lại ván đấu, nếu thất bại sẽ có thông báo lỗi và người chơi có thể chọn bản phát lại khác.



Hình 29: Menu "Phát lại"

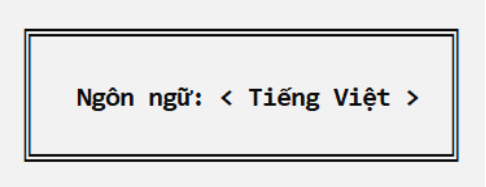
Tại đây, người chơi có thể xem lại trận đấu của mình.



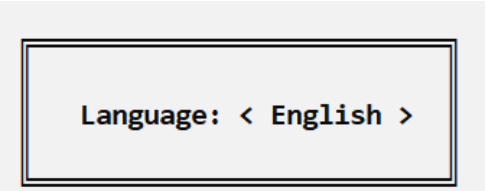
Hình 30: Màn hình chơi bản phát lại

1.2.6. Đa ngôn ngữ

Khi vào trò chơi lần đầu sẽ có bảng thông báo xuất hiện yêu cầu bạn lựa chọn ngôn ngữ cho trò chơi. Ngôn ngữ có sẵn trong trò chơi gồm tiếng Việt và tiếng Anh. Người chơi có thể thêm file ngôn ngữ khác ở mục asset/language để có thể lựa chọn thêm ngôn ngữ đó.

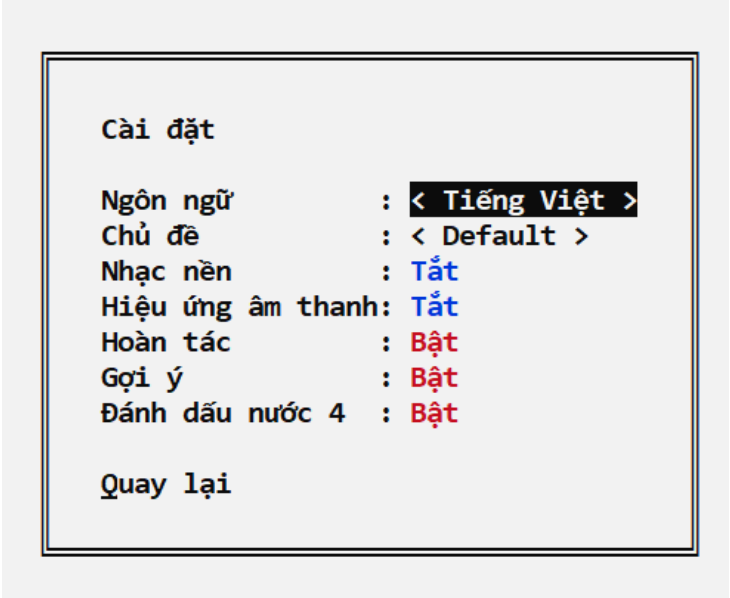


Hình 31: Lựa chọn ngôn ngữ tiếng Việt



Hình 32: Lựa chọn ngôn ngữ tiếng Anh

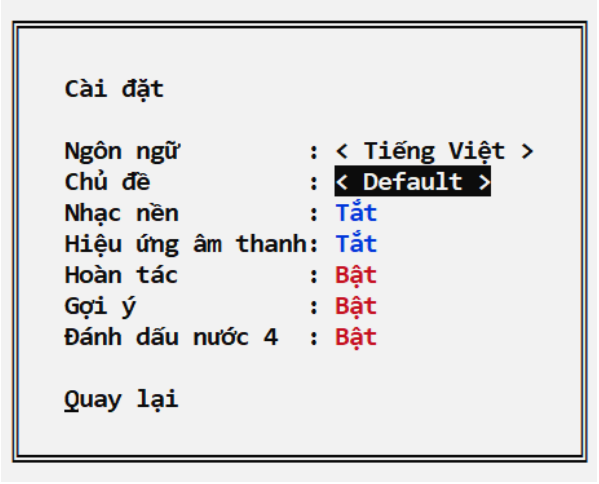
Ngoài ra người chơi còn có thể chỉnh ngôn ngữ trong phần cài đặt của trò chơi.



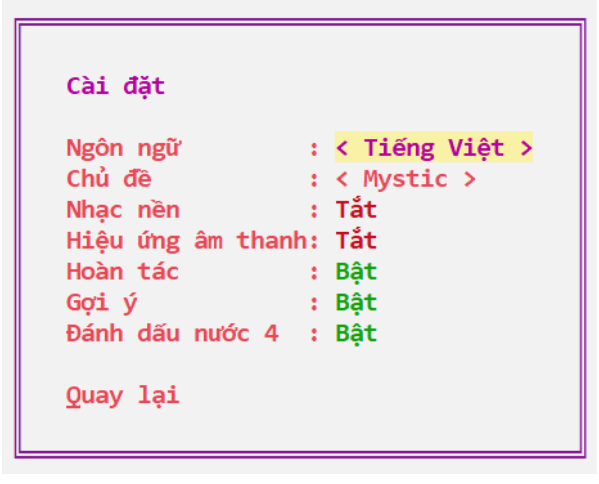
Hình 33: Lựa chọn ngôn ngữ ở phần cài đặt

1.2.7. Thay đổi Chủ đề

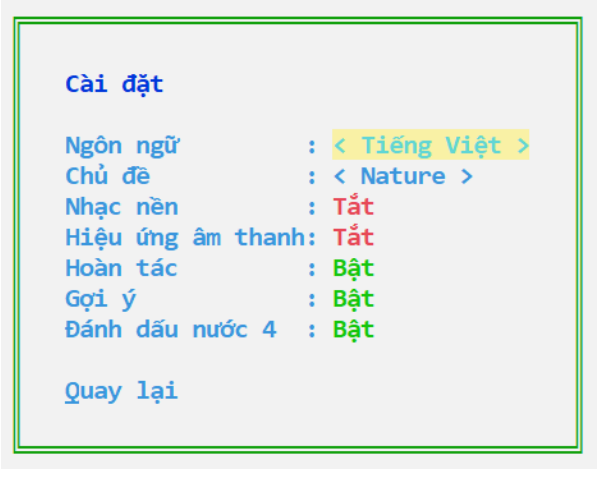
Để tăng tính cá nhân hóa cho trò chơi, chúng em đã thêm tính năng thay đổi chủ đề của trò chơi. Người chơi có thể thay đổi chủ đề của trò chơi trong phần cài đặt. Chủ đề có sẵn trong trò chơi gồm Default, Mystic, Nature, Mystery. Ngoài ra người chơi còn có thể thêm file chủ đề khác vào thư mục `themes`. Khi tải chủ đề lỗi, trò chơi sẽ áp dụng chủ để Default.



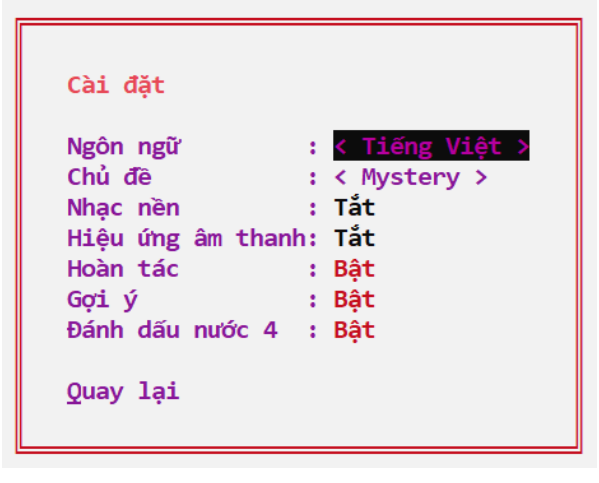
Hình 34: Default Theme



Hình 35: Mystic Theme



Hình 36: Nature Theme



Hình 37: Mystery Theme

Chương 2.

Chi tiết các chức năng

2.1. Logic

2.1.1. Chơi hiệu ứng, nhạc nền

Âm thanh là một phần không thể thiếu trong các trò chơi điện tử, nó khiến cho trò chơi thêm sinh động và chân thực, nâng cao trải nghiệm thi chơi. Dưới đây là những phương pháp mà chúng em đã áp dụng để chơi âm thanh và những khó khăn mà chúng em đã gặp phải.

Các file âm thanh được đặt trong thư mục asset/audio và có thể truy cập bằng các `enum`. Các `enum` được map sang một mảng chứa tên các file âm thanh. Các hàm và class sau đây nằm trong `namespace Audio`, file `Audio.h`, `Audio.cpp`

```
enum class Sound : char {
    NoSound = 0,
    OnKey,
    Draw,
    Win,
    Lose,
    MenuBGM,
    MenuMove,
    MenuSelect,
    GameBGM,
    WinSound,
    GamePlace,
    GameStart,
    Pause,
    WarningSound
};
```

```
constexpr std::array SoundName{
    L"",
    L"Key.wav",
    L"Draw.mp3",
    L"Win.mp3",
    L"Lose.mp3",
    L"MenuBGM.mp3",
    L"MenuMove.wav",
    L"MenuSelect.wav",
    L"GameBGM.mp3",
    L"WinSound.mp3",
    L"GamePlaceMove.mp3",
    L"GameStart.wav",
    L"Pause.wav",
    L"Warning.mp3"
};
```

2.1.1.1. Giải pháp để chơi các âm thanh của giao diện

Để chơi các âm thanh của giao diện, giải pháp của chúng em là sử dụng hàm `PlaySound` [3]. Hàm này có thể chơi các tài nguyên âm thanh thông qua tên

file, con trỏ đến tài nguyên âm thanh trong bộ nhớ hoặc chơi âm thanh của một sự kiện hệ thống

Interface:

```
BOOL PlaySound(
    LPCTSTR pszSound,
    HMODULE hmod,
    DWORD   fdwSound
);
```

Parameter:

- `pszSound`: Con trỏ đến tên file âm thanh, tài nguyên âm thanh trong bộ nhớ hoặc tên sự kiện hệ thống
- `hmod`: Handle của chương trình chứa tài nguyên âm thanh, nếu chơi bằng tài nguyên âm thanh trong bộ nhớ
- `fdwSound`: Các flag để chỉ định cách chơi âm thanh

Return:

- `true` ⟹ phát âm thanh thành công
- `false` ⟹ phát âm thanh thất bại

Ví dụ sử dụng hàm `PlaySound`:

```
    // Chơi âm thanh từ file "recycle.wav"
    PlaySound(TEXT("recycle.wav"), NULL, SND_FILENAME);
    // Chơi âm thanh của 1 sự kiện hệ thống
    PlaySound(TEXT("SystemStart"), NULL, SND_ALIAS);
```

Ưu điểm:

- Khi gọi hàm sẽ tự tải file vào bộ nhớ trong, đọc và phát
- Sau khi gọi xong thì không cần quan tâm đến nữa nên có độ linh hoạt cao

Nhược điểm:

- Phải tải cả file vào bộ nhớ khi chơi nên chỉ chơi những âm thanh ngắn, dung lượng nhỏ, vừa bộ nhớ trong

- Khi chơi có độ delay cao (có thể đươc khắc phục bằng cách tải trước file cần chơi)
- Chỉ có thể mở được file có định dạng `wav`.
- Không thể chơi cùng lúc nhiều âm thanh

Vì những âm thanh giao diện là những file ngắn, nhỏ, nên khắc phục được những điểm yếu của hàm và tận dụng tốt sự linh hoạt cao của hàm `PlaySound`. Nên chúng em đã chọn giải pháp này.

Để thuận tiện hơn trong việc sử dụng, chúng em đã viết hàm `PlayAndForget`.

Interface:

```
    bool PlayAndForget(Sound sound, bool wait)
```

Parameters:

- `Sound`: âm thanh cần chơi
- `wait`:
  - `true` ⟹ phát âm thanh một cách đồng bộ (synchronous)
  - `false` ⟹ phát âm thanh một cách bất đồng bộ (asynchronous)

Return:

- `true` ⟹ phát âm thanh thành công
- `false` ⟹ phát âm thanh thất bại

Usage:

```
    // Chơi âm thanh "MenuSelect.wav" bất đồng bộ
    PlayAndForget(Sound::MenuSelect);
```

2.1.1.2. Giải pháp để chơi nhạc nền

Với những nhược điểm trên thì hàm `PlaySound` không phù hợp để chơi những file dung lượng lớn như nhạc nền và những âm thanh yêu cầu độ trễ thấp. Vì vậy chúng em đã sử dụng một giải pháp khác là sử dụng `Media Control Interface` [4]. Media Control Interface là một chuẩn giao tiếp

giữa các ứng dụng và các thiết bị âm thanh, video, hình ảnh, v.v... Nó cho phép các ứng dụng giao tiếp với các thiết bị âm thanh, video, hình ảnh thông qua chuỗi lệnh đơn giản.

Để gửi lệnh đến thiết bị âm thanh, chúng em sử dụng hàm `mciSendString` [5]. Hàm này có thể gửi chuỗi lệnh đến thiết bị âm thanh và nhận kết quả trả về.

Interface:

```
    MCIERROR mciSendString(
        LPCTSTR lpszCommand,
        LPTSTR  lpszReturnString,
        UINT    cchReturn,
        HANDLE  hwndCallback
    );
```

Parameters:

- `lpszCommand`: Con trỏ đến chuỗi lệnh cần gửi đến thiết bị âm thanh
- `lpszReturnString`: Con trỏ đến mảng chứa chuỗi nhận thông tin trả về
- `cchReturn`: Độ dài của chuỗi trả về
- `hwndCallback`: Handle của cửa sổ sẽ nhận thông báo khi thiết bị âm thanh hoàn thành công việc

Return:

- Nếu gửi thành công sẽ trả về `0`, nếu lỗi trả về một giá trị biểu thị lỗi

Usage:

```
    // Mở file "song.mp3"
    mciSendString("open song.mp3 type mpegvideo alias song", NULL,
0, NULL);
    // Chơi file "song.mp3"
    mciSendString("play song from 0 repeat", NULL, 0, NULL);
    // Dừng file "song.mp3"
    mciSendString("stop song", NULL, 0, NULL);
    // Đóng file "song.mp3"
    mciSendString("close song", NULL, 0, NULL);
```

Ưu điểm:

- Chơi được nhiều định dạng âm thanh khác nhau
- Chơi được các file lớn
- Khi chơi ít bị delay do không cần tải hết file vào bộ nhớ trong
- Có thể chơi cùng lúc nhiều âm thanh

Nhược điểm:

- Phải sử dụng chuỗi để giao tiếp
- Phải tự quản lí các file âm thanh đã mở nên không có độ linh hoạt cao

Để khắc phục nhược điểm trên, chúng em đã tạo `class AudioPlayer` để việc sử dụng và quản lí tài nguyên thuận tiện, dễ dàng hơn

Interface:

```
class AudioPlayer {
    AudioPlayer();
    AudioPlayer(Sound song); // Khởi tạo và mở file

    // Mở file. Có thể dùng để đổi file cần chơi
    int Open(Sound song);
    Sound getCurrentSong() const;

    int Play(bool fromStart = 1, bool repeat = 0) const; // Chơi
    int Pause() const;       // Tạm dừng
    int Resume() const;      // Tiếp tục
    int Stop() const;        // Dừng chơi và trả con trỏ về đầu
    int Close();             // Đóng file đang mở

    ~AudioPlayer();          // Đóng file đang mở
}
```

Parameters:

- `song`: âm thanh cần chơi
- `fromStart`:
    - `true` ⟹ chơi từ đầu
    - `false` ⟹ chơi tiếp tại vị trí con trỏ
- `repeat`:
    - `true` ⟹ lặp lại khi kết thúc

Return:

- Các phương thức sẽ trả về `MCI code` của lệnh MCI tương ứng

Usage:

```
{
    AudioPlayer player(Audio::Sound::Draw);
    player.play(true, true);
}
```

`class AudioPlayer` có một nhược điểm lớn là tuổi thọ phụ thuộc vào thời gian sống của biến cục bộ và không thể được truy cập được từ các thành phần bên ngoài. Để khắc phục điểm yếu ấy chúng em đã tạo ra `class BackgroundAudioService` để tăng tuổi thọ của `class AudioPlayer`, đồng thời khiến cho nhạc nền có thể được điều khiển những nơi khác.

Interface:

```
class BackgroundAudioService {
    BackgroundAudioService() = delete;
    static Audio::Sound GetCurrentSong();

    static int ChangeSong(Audio::Sound song) // Đổi nhạc

    static int Play(bool fromStart = 0, bool repeat = 1); // Chơi
    static int Pause();                                   // Tạm dừng
    static int Resume();                                  // Tiếp tục
    // Dừng chơi, trả con trỏ về đầu
    static int Stop();
};
```

Parameters:

- `song`: âm thanh cần chơi
- `fromStart`:
    - `true` ⟹ chơi từ đầu
    - `false` ⟹ chơi tiếp tại vị trí con trỏ
- `repeat`:
    - `true` ⟹ lặp lại khi kết thúc

Usage:

```
{
    BackgroundAudioService::ChangeSong(Audio::Sound::MenuBGM);
    BackgroundAudioService::Play(true, true);
}
```

2.1.2. Điều hướng trong ứng dụng

Việc chuyển đổi giữa các màn hình khác nhau trong trò chơi là một thách thức lớn đối với chúng em, vì đây là lần đầu chúng em gặp phải vấn đề này. Để giải quyết vấn đề này, ban đầu chúng em gọi các hàm trực tiếp từ main, muốn chuyển tới màn hình nào thì gọi hàm của màn hình đó. Nhưng phương pháp này nhanh chóng để lộ nhiều điểm yếu:

- Cần phải biết chữ kí hàm của màn hình cần chuyển đến
- Khó quản lí các màn hình và các đích đến của chúng
- Có thể bị tràn stack khi chuyển màn hình nhiều lần
- Nếu muốn sửa lại code phải sửa ở nhiều nơi
- Khó mở rộng, dễ lỗi

Để khắc phục điểm yếu đó, chúng em đã tạo ra một hệ thống để quản lí các màn hình, lấy ý tưởng từ thư viện `navigation-compose` [6] (thư viện điều hướng của Jetpack [7]). Hệ thống này giúp chúng em có thể chuyển đổi giữa các màn hình một cách dễ dàng, linh hoạt và có thể mở rộng dễ dàng hơn.

Hệ thống này gồm 2 thanh phần:

- `class NavigationHost`: class trung tâm để quản lí các màn hình
- Các màn hình: là các hàm có chữ kí như sau:
  - `void ScreenName(NavigationHost& host)`

Mỗi màn hình sẽ được gán một nhãn độc nhất, nhãn này sẽ được sử dụng để chuyển đến màn hình đó. Các màn hình muốn chuyển đến màn hình khác sẽ gọi phương thức `Navigate("Nhãn")` để chuyển đến màn hình đó hoặc `NavigateStack("Nhãn")` để chuyển đến màn hình đó nhưng không xóa đi màn hình hiện tại. Ngoài ra, hệ thống còn có xử lí việc xóa màn hình trước khi

chuyển đến màn hình mới, lưu lịch sử di chuyển giữa các màn hình để có thể quay lại màn hình trước đó và cung cấp một số phương thức để hỗ trợ việc truyền dữ liệu giữa các màn hình.

Interface:

```
#define ViewFunc std::function<void(NavigationHost&)>
#define ViewFuncMap std::unordered_map<std::string, ViewFunc>

class NavigationHost {

    NavigationHost() = default;
    NavigationHost(
        const std::string& Start,
        const ViewFuncMap& links
    );

    // Các phương thức hỗ trợ truyền dữ liệu giữa các màn hình
    std::any& GetFromContext(const std::string& name);
    bool CheckContext(const std::string& name);
    void SetContext(
        const std::string& name,
        const std::any& data
    );
    void DeleteContext(const std::string& name);

    // Thêm màn hình khi đang chạy
    void Add(const std::string& path, const ViewFunc& view);

    // Các phương thức hỗ trợ điều hướng
    void NavigateStack(const std::string& path);
    void Navigate(const std::string& path);
    void Back();
    void BackToLastNotOverlay();
    void NavigateExit();

    ~NavigationHost();
};
```

Parameters:

- `Start`: nhãn của màn hình bắt đầu
- `links`: danh sách các màn hình có trong ứng dụng và nhãn của chúng

- `path`: nhãn của màn hình cần chuyển đến
- `view`: hàm của màn hình cần chuyển đến
- `name`: nhãn của dữ liệu cần truyền
- `data`: dữ liệu cần truyền

Usage:

```
#include <iostream>

void GameScreen(NavigationHost& NavHost) {
    // Lấy dữ liệu đã truyền
    int a = std::any_cast<int>(NavHost.GetFromContext("Context"));
    std::cout << a;
    // ...
    // Thoát chương trình
    return NavHost.NavigateExit();
}

void StartScreen(NavigationHost& NavHost) {
    // Truyền dữ liệu giữa các màn hình
    NavHost.SetContext("Context", 90);
    // ...
    return NavHost.Navigate("Game");
}

int main() {
    // Khởi tạo hệ thống điều hướng
    NavigationHost(
        "Start",
        {
            {"Start", StartScreen},
            {"Game", GameScreen},
        }
    );
    return 0;
}
```

Ưu điểm:

- Dễ mở rộng
- Không cần phải biết tên hàm để chuyển màn hình
- Khi thay đổi chỉ cần sửa ở một nơi

Nhược điểm:

- Khó chuyển dữ liệu giữa các màn hình
- Dễ đánh sai nhãn màn hình

2.1.3. Phương pháp lưu và tải trò chơi(save/load trò chơi)

Để lưu trò chơi, chúng em đã chọn lưu trạng thái của trò chơi vào một file văn bản thuần và lưu trong một thư mục riêng. Để tải trò chơi, chúng em sẽ đọc file văn bản đó và khởi tạo lại trạng thái của trò chơi. Khi người chơi muốn tải trò chơi, chúng em muốn hiển thị một danh sách các file lưu trò chơi để người chơi có thể chọn file cần tải. Có nhiều phương pháp để thực hiện việc này. Một trong những giải pháp mà chúng em đã cân nhắc là lưu tên file lưu trò chơi vào một file văn bản thuần, khi cần tải, chúng em sẽ đọc file đó và hiển thị cho người chơi. Tuy nhiên, chúng em đã quyết định không sử dụng phương pháp này do nó có nhiều khuyết điểm như:

- Tạo ra một file không cần thiết
- Người chơi không thể tải trò chơi nếu file đó bị xóa/lỗi
- Người chơi không thể tải các file copy từ máy khác

Một cách tiếp cận khác là mỗi khi người chơi muốn tải trò chơi thì sẽ duyệt qua các file trong thư mục lưu trò chơi và hiển thị cho người chơi. Điều này có nhiều ưu điểm như:

- Không cần tạo ra file không cần thiết
- Người chơi có thể tải trò chơi từ máy khác

Để quét các file trong thư mục, chúng em đã sử dụng thư viện `filesystem` [8]. Đây là một thư viện mới xuất hiện trong phiên bản `C++17`. Nó cung cấp các tiện ích để thực hiện các thao tác trên hệ thống tập tin và các thành phần của chúng, chẳng hạn như đường dẫn, tập tin thông thường và thư mục.

Sau đây là cách mà chúng em đã dùng để quét các file trong thư mục lưu trò chơi:

Implementation:

```
namespace FileHandle {
struct FileDetail {
    std::filesystem::path               filePath;
    std::filesystem::file_time_type     lastModified;
};

std::vector<FileHandle::FileDetail>
GetAllTextFileInDir(
    const std::filesystem::path& Dir
)
{
    std::vector<FileDetail> res;
    Ensure(Dir); // Đảm bảo đường dẫn tồn tại
    for (auto& file : std::filesystem::directory_iterator(Dir)) {
        if (file.is_regular_file()) {
            res.emplace_back(
                file.path(),
                file.last_write_time()
            );
        }
    }
    return res;
}
} // namespace FileHandle
```

Parameters:

- `Dir`: đường dẫn đến thư mục muốn tìm

Return:

- Trả về một `vector` chứa các thông tin của các file đã tìm được

Usage:

```
{
    // Tìm các file văn bản trong đường dẫn
    // tương đối "saves"
    auto files = GetAllTextFileInDir("saves");
    for (auto& file:files) {
        std::cout << file.filePath.filename() << '\n';
    }
}
```

Trạng thái của một ván đấu trong trò chơi bao gồm các thông tin sau:

- Thời gian chơi hoặc thời gian còn lại của mỗi người chơi, thời gian đã trôi qua
- Thông tin của 2 người chơi: tên, avatar, điểm số
- Chế độ, loại trò chơi, độ khó của AI, ai đi trước, kết cục của trò chơi
- Các nước đi đã được thực hiện trong ván đấu

Các trạng thái ấy được gộp chung vào một `struct` như sau:

```
struct GameState {
    std::wstring playerNameOne;
    short playerTimeOne = 0;
    short playerScoreOne = 0;
    short playerAvatarOne = -1;

    std::wstring playerNameTwo;
    short playerTimeTwo = 0;
    short playerScoreTwo = 0;
    short playerAvatarTwo = -1;

    short gameType = 0;
    short gameMode = 0;
    short aiDifficulty = 0;
    bool playerOneFirst = 1;
    short gameTime = 0;
    short gameEnd = 0;
    std::vector<std::pair<short, short>> moveList;
};
```

Với những thông tin ấy, chúng em đã chọn cách lưu ván đấu dưới dạng file văn bản do sự tiện dụng của cách lưu này. Tuy nhiên, khi lưu bằng cách này thì người chơi có thể dễ dàng can thiệp vào file lưu gây ra một số tình huống không mong muốn, một điều khó tránh khỏi kể cả khi lưu bằng file nhị phân.

Khi lưu ván đấu, hàm sẽ mở file với tên được người chơi nhập vào, sau đó ghi các thông tin của ván đấu vào file. Khi tải ván đấu, hàm sẽ mở file với tên được người chơi chọn, sau đó đọc các thông tin của ván đấu từ file.

Sau đây là chi tiết hàm lưu ván đấu:

Implementation:

```
bool SaveLoad::Save(
    const GameState& data,
    const std::wstring& name,
    const std::filesystem::path& dir
)
{
    auto file = FileHandle::OpenOutFile(dir.generic_wstring() +
name);
    if (file.fail()) {
        return false;
    }
    file << data.playerNameOne << '\n';
    file << data.playerScoreOne << '\n';
    file << data.playerTimeOne << '\n';
    file << data.playerAvatarOne << '\n';

    file << data.playerNameTwo << '\n';
    file << data.playerScoreTwo << '\n';
    file << data.playerTimeTwo << '\n';
    file << data.playerAvatarTwo << '\n';

    file << data.gameMode << '\n';
    file << data.aiDifficulty << '\n';
    file << data.playerOneFirst << '\n';
    file << data.gameTime << '\n';
    file << data.gameEnd << '\n';

    for (auto& i : data.moveList) {
        file << i.first << ' ' << i.second << '\n';
    }

    if (file.fail()) {
        std::filesystem::remove(dir.generic_wstring() + name);
    }

    return !file.fail();
}
```

Parameters:

- `data`: Trạng thái của ván đấu cần lưu
- `name`: Tên của file lưu

- `dir`: Đường dẫn của thư mục chứa file lưu

Return:

- `true` → Lưu thành công
- `false` → Lưu thất bại

Sau đây là chi tiết hàm tải ván đấu:

Implementation:

```
std::optional<GameState>
SaveLoad::Load(
    const std::filesystem::path& filePath
)
{
    auto file = FileHandle::OpenInFile(filePath);
    GameState data;
    file >> data.playerNameOne;
    file >> data.playerScoreOne;
    file >> data.playerTimeOne;
    file >> data.playerAvatarOne;

    file >> data.playerNameTwo;
    file >> data.playerScoreTwo;
    file >> data.playerTimeTwo;
    file >> data.playerAvatarTwo;

    file >> data.gameMode;
    file >> data.aiDifficulty;
    file >> data.playerOneFirst;
    file >> data.gameTime;
    file >> data.gameEnd;

    short a, b;
    while (!file.fail()) {
        file >> a >> b;
        data.moveList.emplace_back(a, b);
    }

    if (file.fail() && !file.eof()) {
        return std::nullopt;
    }
    return data;
}
```

Parameters:

- `filePath`: Đường dẫn của file lưu

Return:

- Trạng thái của ván đấu được tải

2.1.4. Lưu lại thiết lập của người chơi

Nếu mỗi khi mở trò chơi lên, người chơi lại phải chỉnh sửa các thiết lập theo ý muốn thì rất bất tiện và mất thời gian. Do đó, chúng em đã lưu lại các thiết lập của người chơi vào file để khi mở trò chơi lên, người chơi không cần phải chỉnh sửa lại các thiết lập. Các thiết lập được lưu dưới dạng file văn bản có đường dẫn là `config/user_config` và được lưu trữ trong một bảng để dễ dàng truy xuất và chỉnh sửa. Các hàm nằm trong `namespace Config` được định nghĩa trong file `Config.h` và `Config.cpp`.

Interface:

```
namespace Config {
    typedef std::unordered_map<std::wstring, std::wstring> Dict;
    extern Dict configDict;

    // Lưu, tải thiết lập của người chơi từ file
    bool LoadUserSetting();
    bool SaveUserSetting();

    // Xem và sửa thiết lập của người chơi
    std::wstring& GetConfig(const std::wstring& name);
    void SetConfig(
        const std::wstring& name,
        const std::wstring& data
    );
}; // namespace Config
```

Đây là chi tiết hàm lưu thiết lập của người chơi:

Implementation:

```
bool Config::SaveUserSetting()
{
    auto fout = FileHandle
        ::OpenOutFile(Constants::USERCONFIG_FILE_PATH);
    for (const auto& [key, val] : configDict) {
        fout << key << L'=' << val << '\n';
    }
    return !fout.fail();
}
```

Return:

- `true` ⟶ Lưu thành công
- `false` ⟶ Lưu thất bại

Đây là chi tiết hàm tải thiết lập của người chơi:

Implementation:

```
bool Config::LoadUserSetting()
{
    auto fin = FileHandle
        ::OpenInFile(Constants::USERCONFIG_FILE_PATH);
    if (fin.fail()) {
        return 0;
    }
    std::wstring buffer;
    while (!fin.eof()) {
        fin >> buffer;
        auto tmp = Utils::LineSplitter(buffer);
        Utils::trim(tmp.first); Utils::trim(tmp.second);
        configDict[tmp.first] = tmp.second;
    }
    return 1;
}
```

Rerturn:

- `true` ⟶ Tải thành công
- `false` ⟶ Tải thất bại

2.1.5. Đếm giờ trong khi chơi trò chơi

Việc đếm và hiển thị thời gian trực tiếp trong lúc chơi khá phức tạp vì luồng chính trong trò chơi luôn phải chờ đợi và xử lí đầu vào của người chơi, nên việc sử dụng luồng chính để nó phụ thuộc vào việc người chơi có thực hiện các thao tác trong trò chơi hay không. Nếu người chơi không thực hiện các thao tác trong trò chơi thì thời gian sẽ không được cập nhật lên màn hình. Để giải quyết vấn đề này, chúng em đã tạo thêm một luồng riêng để đếm thời gian và cập nhật lên màn hình. Sử dụng thư viện `thread` [9] để tạo luồng mới, chúng em đã có giải pháp để chạy một hàm sau một khoảng thời gian nhất định.

Implementation:

```
using namespace std::chrono;

struct TimerInternalState {
    std::function<void(void)> callback;
    milliseconds interval;
    bool running = false;
    bool pause = false;
};

class Timer {
    std::thread _thread;
    std::shared_ptr<TimerInternalState> _state {
        new TimerInternalState };

   public:
    Timer(
        std::function<void(void)> callback,
        const long& interval = 1000
    ) {
        _state->callback = callback;
        _state->interval = milliseconds{interval};
    }
```

```
    inline void Start()
    {
        _state->running = true;
        auto state = _state;
        _thread = std::thread([state] {
            while (state->running) {
                auto nextInterval = steady_clock::now();
                nextInterval += state->interval;
                if (!state->pause) { state->callback(); }
                std::this_thread::sleep_until(nextInterval);
            }
        });
        _thread.detach();
    }

    inline void Pause() { _state->pause = true; }
    inline void Continue() { _state->pause = false; }
    inline void Stop() { _state->running = false; }
    inline ~Timer() { Stop(); }
};
```

Parameters:

- `callback`: hàm sẽ được gọi sau mỗi khoảng thời gian `interval`
- `interval`: khoảng thời gian giữa các lần gọi hàm `callback` tính bằng mili giây

Việc lập trình đa luồng trong C++ khá phức tạp, và cũng là phần dễ gây lỗi nhất trong trò chơi, do việc vẽ lên màn hình phải được đồng bộ giữa các luồng với nhau. Nếu không đồng bộ thì có thể dẫn đến việc các phần tử trên màn hình bị vẽ sai vị trí. Để việc đó không xảy ra, chúng em đã sử dụng một khóa `mutex` [10] chung để đồng bộ các luồng với nhau.

Đoạn code sử dụng `Timer` và `mutex` trích từ trò chơi:

```
void GameScreenView::GameScreenView(NavigationHost& NavHost) {
std::mutex lock;
Timer timerPlayerOne(
    [&] {
        if (!endGame) {
            curGameState.playerTimeOne += timeAddition;
            std::lock_guard guard(lock);
            // Vẽ thời gian lên màn hình ...
            if (curGameState.playerTimeOne == 0 && !endGame) {
                // Xử lí khi hết thời gian
            }
        }
    }
);
}
```

2.1.6. Giải pháp cho đa ngôn ngữ

Các văn bản trong trò chơi thay vì được code cứng vào trò chơi thì sẽ được tải từ một file riêng, điều này kiến cho phần ngôn ngữ trong trò chơi có thể được chỉnh sửa một cách dễ dàng và khiến cho việc thêm các ngôn ngữ mới dễ dàng hơn.

File ngôn ngữ là một file văn bản thuần chứa các nhãn và phần văn bản ngăn cách bởi dấu "=", các nhãn có nằm bên trong cặp ngoặc `[]` là `meta` được dùng để chứa thông tin về file ngôn ngữ. Sau khi tải xong, các nhãn và văn bản sẽ được lưu vào một bảng để có thể dễ dàng truy xuất.

Ví dụ file ngôn ngữ:

```
[LANGUAGE]                  =    English
[LANG_SELECT]               =    Language
ABOUT_DESC                  =    LIST OF GROUP MEMBERS AND SOURCE
CODE LINK
ABOUT_SHORTCUT              =    A
ABOUT_TITLE                 =    About us
ABOUT_US_TITLE              =    About us
```

Các nhãn và văn bản trong trò chơi được quản lí, truy xuất và tải vào bộ nhớ thông qua class `Language` nằm trong file `Language.h` và `Language.cpp`

Interface:

```
typedef std::unordered_map<std::wstring, std::wstring> Dict;
typedef std::filesystem::path fsPath;

struct LanguageOption {
  Dict meta;
  fsPath path;
};

class Language {
  Language() = delete;

  // Chỉ đọc các phần thông tin về ngôn ngữ
  static Dict ExtractMetaFromFile(const fsPath& filePath);

  // Load file ngôn ngữ
  static void LoadLanguageFromFile(const fsPath& filePath);

  // Tìm các file ngôn ngữ
  static std::vector<LanguageOption>
  DiscoverLanguageFile(const fsPath& dirPath);

  // Truy xuất văn bản bằng nhãn
  static const std::wstring&
  GetString(const std::wstring& Label);

  static const std::wstring&
  GetMeta(const std::wstring& Label);
};
```

Parameters:

- `filePath`: đường dẫn tới file cần mở
- `dirPath`: đường dẫn tới thư mục cần tìm
- `Label`: nhãn của văn bản cần lấy

Usage:

```
{
    Language::LoadLanguageFromFile("asset/language/en.txt");
    std::cout << Language::GetMeta(L"[LANGUAGE]");
    std::cout << Language::GetMeta(L"ABOUT_TITLE");
}
```

2.1.7. Chủ đề

Nhằm tăng trải nghiệm khi chơi trò chơi và đáp ứng nhu cầu cá nhân hóa của người chơi, chúng em đã thêm chức năng thay đổi chủ đề màu sắc của trò chơi. Khi sử dụng tính năng này, người chơi có thể đôi màu sắc của một vài thành phần giao diện trong trò chơi theo ý mình. Các file chủ đề được lưu trong thư mục "theme", được quản lí và truy xuất thông qua class `Theme` nằm trong file `Theme.h` và `Theme.cpp`

File chứa chủ đề là một file văn bản thuần chứa các nhãn và màu ngăn cách bởi dấu "=", tên của chủ đề là tên của file. Sau khi tải xong, các màu sẽ được lưu vào một mảng để dễ dàng truy xuất. Việc tải và và truy xuất các màu sắc hoạt động tương tự như việc tải và truy xuất các văn bản trong trò chơi.

Ví dụ file chủ đề:

```
PLAYER_ONE_COLOR            =    1
PLAYER_ONE_HIGHLIGHT_COLOR  =    9
PLAYER_TWO_COLOR            =    4
PLAYER_TWO_HIGHLIGHT_COLOR  =    12
RESULT_TEXT_COLOR           =    6
SWITCH_OFF                  =    4
SWITCH_ON                   =    2
TEXT_COLOR                  =    12
TEXT_HIGHLIGHT_COLOR        =    13
TITLE_TEXT_COLOR            =    13
```

Các màu sắc trong trò chơi được đánh dấu bằng số từ 0 đến 15, mỗi số tương ứng với một màu nhất định.

```
enum class Color : char {
    BLACK = 0,
    BLUE = 1,
    GREEN = 2,
    CYAN = 3,
    RED = 4,
    MAGENTA = 5,
    YELLOW = 6,
    WHITE = 7,
    GRAY = 8,
    LIGHT_BLUE = 9,
    LIGHT_GREEN = 10,
    LIGHT_CYAN = 11,
    LIGHT_RED = 12,
    LIGHT_MAGENTA = 13,
    LIGHT_YELLOW = 14,
    BRIGHT_WHITE = 15
};
```

2.1.8. Các hàm hỗ trợ về giao diện

Các hàm hỗ trợ về giao diện được định nghĩa trong `namespace View` file `View.h` và `View.cpp`. Các hàm này được sử dụng để vẽ giao diện cho trò chơi. Nó cung cấp các hàm để vẽ các thành phần giao diện như: văn bản, hình chữ nhật, menu, ô nhập liêu, và các hàm để xóa màn hình. Các thành phần được vẽ bởi các hàm này sẽ có màu sắc mặc định tuân theo hệ thống chủ để mà không cần can thiệp từ người sử dụng. Ngoài ra nó còn được tối ưu để vẽ lên màn hình ít hơn, tạo cho người chơi cảm giác mượt mà cho người chơi. Dưới đây là một vài hàm sử dụng nhiều trong trò chơi.

2.1.8.1. Hàm DrawToView

Hỗ trợ vẽ văn bản lên màn hình. Hàm có 2 phiên bản, một phiên bản với tham số là chuỗi ký tự, và một phiên bản với tham số là ký tự.

Interface:

```
void WriteToView(
    short x, short y,
    const std::wstring& str,
    wchar_t shortcut,
    bool highlight,
    Color textColor,
    Color highlightColor,
    Color highlightTextColor,
    Color backgroundColor
);
void WriteToView(
    short x, short y,
    wchar_t str,
    bool highlight,
    Color textColor,
    Color highlightColor,
    Color highlightTextColor,
    Color backgroundColor
);
```

Parameters

- `x`, `y`: tọa độ x,y của văn bản
- `str`: văn bản cần vẽ
- `shortcut`: phím tắt của văn bản (được gạch chân)
- `highlight`: văn bản có được làm nổi bật hay không
- `textColor`: màu sắc của văn bản
- `highlightColor`: màu sắc của nền khi được làm nổi bật
- `highlightTextColor`: màu sắc của văn bản khi được làm nổi bật
- `backgroundColor`: màu sắc nền

Usage:

```
{ // Vẽ văn bản "Hello World" ở tọa độ 10, 10 lên màn hình
    // Các tham số còn lại đã được định nghĩa lúc khai báo hàm
    WriteToView(10, 10, "Hello World");
}
```

2.1.8.2. Hàm DrawMenu

Hỗ trợ vẽ menu lên màn hình. Hàm này còn có một phiên bản khác là `DrawMenuCenter` dùng để vẽ menu được căn ở giữa màn hình.

Interface:

```
struct Rect {
    short Top = 0, Left = 0, Right = 0, Bottom = 0;
};

struct Option {
    const std::wstring& option;
    const wchar_t underline;
};

Rect DrawMenu(
    DrawMenuPrevState& prevState,
    short x,
    short y,
    const std::wstring& title,
    const std::vector<Option>& optionsList,
    size_t selected,
    Color textColor,
    Color highlightColor,
    Color highlightTextColor,
    Color backgroundColor,
    Color titleColor
);

Rect DrawMenuCenter(
    DrawMenuPrevState& prevState,
    const std::wstring& title,
    const std::vector<Option>& optionsList,
    size_t selected,
    Color textColor,
    Color highlightColor,
    Color highlightTextColor,
    Color backgroundColor,
    Color titleColor
);
```

Parameters:

- `prevState`: trạng thái trước đó của menu (dùng cho việc tối ưu hóa)

- `x`, `y`: tọa độ x, y của menu
- `title`: tiêu đề của menu
- `optionsList`: danh sách các tùy chọn của menu
- `selected`: tùy chọn được chọn
- `textColor`: màu sắc của văn bản
- `highlightColor`: màu sắc của nền khi được làm nổi bật
- `highlightTextColor`: màu sắc của văn bản khi được làm nổi bật
- `backgroundColor`: màu sắc nền
- `titleColor`: màu sắc của tiêu đề

Return:

- Phần diện tích của menu đã được vẽ

Usage:

```
{ // Vẽ menu có 2 tùy chọn lên màn hình
    // Các tham số còn lại đã được định nghĩa lúc khai báo hàm
    DrawMenuPrevState prevState;
    DrawMenuCenter(prevState, "Menu", {
        { "Option 1", '1' },
        { "Option 2", '2' }
    }, 0);
}
```

2.1.8.3. Hàm Input

Hàm hỗ trợ lấy chuỗi mà người dùng nhập vào từ bàn phím. Đây là một thành phần giao diện đặc biệt và có cấu trúc phức tạp do bên trong hàm có một vòng lặp để đọc đầu vào từ người chơi. Khi người chơi chỉnh sửa nội dung, nó sẽ gọi hàm `onSearchValueChange` được đưa vào từ màn hình đã gọi nó, nhờ đó, người sử dụng hàm có thể can thiệp nhiều hơn vào đầu vào của người chơi. Từ đó, có thể xây dựng nhiều tính năng như giới hạn độ dài của đầu vào hay tìm kiếm ngay khi người dùng nhập vào một ký tự.

Interface:

```
wchar_t Input(
    short x,
    short y,
    const std::wstring& leadingText,
    std::wstring& inputText,
    bool hasFocus,
    const std::function<void(const std::wstring&)>& onValueChange,
    const std::function<bool(wchar_t)>& toogleFocus,
    const wchar_t delimiter,
    Color textColor,
    Color backgroundColor,
    Color focusTextColor,
    Color focusBackgroundColor
);
```

Parameters:

- `leadingText`: Chữ hiển thị trước ô nhập văn bản.
- `inputText`: Nội dung của ô nhập văn bản.
- `hasFocus`: Trạng thái focus của ô nhập văn bản.
- `onValueChange`: Hàm được gọi khi người dùng thay đổi nội dung của ô nhập văn bản.
- `toogleFocus`: Hàm được gọi mỗi khi người dùng thay đổi đầu vào, nếu hàm trả về `true` thì sẽ trả lại điều khiển cho màn hình đã gọi hàm.
- `delimiter`: Ký tự phân cách giữa chữ hiển thị trước ô nhập văn bản và ô nhập văn bản.
- `textColor`: Màu chữ của ô nhập văn bản.
- `backgroundColor`: Màu nền của ô nhập văn bản.
- `focusTextColor`: Màu chữ của ô nhập văn bản khi được focus.
- `focusBackgroundColor`: Màu nền của ô nhập văn bản khi được focus.

Return:

- `wchar_t`: Ký tự được nhập vào khiến cho hàm trả lại điều khiểu cho màn hình đã gọi hàm.

Usage:

```
{
    std::wstring searchValue;
    // Các tham số còn lại đã được định nghĩa lúc khai báo hàm
    Input(
        10, 10,
        L"Search",
        searchValue,
        hasFocus,
        [&](const std::wstring& value) {
            // Hàm này được gọi mỗi khi người dùng thay đổi nội
dung của ô nhập văn bản
            if (value.length() > 30) {
                // Giới hạn độ dài của đầu vào ở mức 30 kí tự
                return;
            }
            searchValue = value;
        }
    );
}
```

2.1.9. Nhận biết thắng thua

Việc nhận biết kết quả thắng, thua, và hòa của một ván đấu được thực hiện trong `namespace Logic` của chương trình. Các kết quả này là điều kiện để chương trình quyết định kết thúc ván đấu. Ngoài ra, việc biết được kết quả thắng, thua, và hòa sẽ giúp AI của trò chơi đưa ra đánh giá về trạng thái bàn cờ một cách đúng đắn.

2.1.9.1. Hàm GetGameState

Hàm `GetGameState` có vai trò đánh giá hiện trạng của ván đấu sau nước đi mới nhất. Cụ thể hơn, hàm xem xét nước đi mới nhất có dẫn đến một **kết quả thắng** hay **kết quả hòa**. Một nước đi sẽ dẫn đến kết quả thắng nếu nước đi đó tạo nên một chuỗi 5 nước đi liên tiếp đồng chất, và một nước đi sẽ dẫn đến kết quả hòa nếu nước đi đó không phải là nước đi thắng, đồng thời là nước đi hợp lệ cuối cùng của bàn đấu.

Interface:

```
typedef std::vector<std::vector<short>> Board;
struct Point {
    int row, col;
}

short GetGameState(
    const GameAction::Board& board,
    const short& moveCount,
    const GameAction::Point& move,
    const short& playerValue,
    GameAction::Point& winPoint,
    bool getWinPoint
);
```

Parameters:

- `board`: Bàn đấu hiện tại.
- `moveCount`: Số nước đi đã thực hiện.
- `move`: Nước đi mới nhất.
- `playerValue`: Người chơi thực hiện nước đi.
- `winPoint`: Đầu mút của chuỗi thắng (nếu có).
- `getWinPoint`:
  - `true` ⟶ lấy đầu mút của chuỗi thắng (nếu có).
  - `false` ⟶ không lấy đầu mút của chuỗi thắng.

Return

- `Logic::WIN_VALUE`: Giá trị tượng trưng kết quả thắng (người thắng là playerValue).
- `Logic::DRAW_VALUE`: Giá trị tượng trưng kết quả hòa.
- `Logic::NULL_VALUE`: Giá trị tượng trưng kết quả vô định.

Usage:

```
{
    short gameState = Logic::GetGameState(
        gameBoard,
        moveCount,
        latestMove,
        currentPlayer,
        winPoint,
        true);
    if (gameState == Logic::WIN_VALUE) {
        std::cout << currentPlayer << " wins";
    }
    else if (gameState == Logic::DRAW_VALUE) {
        std::cout << "Game draw";
    }
}
```

2.1.9.2. Cách phát hiện nước đi thắng

Nước đi thắng là một nước đi dẫn đến một chuỗi 5 quân cờ liền kề đồng chất, dựa vào đặc điểm này, ta viết nên các hàm xử lý đáp ứng việc kiểm tra ấy. Cụ thể, bên trong hàm `GetGameState`, việc kiểm tra nước thắng sẽ được thực hiện qua bốn hàm `CheckVerticalWin`, `CheckHorizontalWin`, `CheckLeftDiagonalWin`, và `CheckRightDiagonalWin`. Các hàm ấy sẽ kiểm tra nước thắng cho 4 hướng tương ứng là: dọc, ngang, chéo từ trái sang phải, chéo từ phải sang trái. Tuy khác về hướng kiểm tra, nhưng bốn hàm đều có chung một phương pháp kiểm tra. Ví dụ, đối với hàm `CheckVerticalWin`, việc kiểm tra sẽ diễn ra như sau:

- Từ vị trí của nước đi mới nhất, ta cho kiểm tra nước ngay cạnh, theo hướng của chiều dọc, có **đồng chất** hay không:
    - **Nếu có** => tiếp tục kiểm tra nước ngay cạnh nước ấy.
    - **Nếu không** => ngưng kiểm tra.
- Nếu số nước đồng chất tính được bằng số nước đi thắng (trong trường hợp này là 5) thì trả về `true`, ngược lại trả về `false`.

Implementation:

```
bool CheckVerticalWin(
    const GameAction::Board& board,
    const GameAction::Point& move,
    const short& playerValue
)
{
    // Biến đếm số quân cờ đồng chất
    short sameValueCount = 1;

    // Kiểm tra theo chiều dọc trên
    for (
        short row = move.row + 1;
        row < Constants::BOARD_SIZE;
        ++row
    ) {
        if (board[row][move.col] == playerValue) {
            // Tăng số quân đồng chất tìm được
            sameValueCount++;
            // Nếu số quân bằng 5, trả về true
            if (sameValueCount == Constants::WIN_VALUE_COUNT) {
                return true;
            }
        } else {
            break;
        }
    }

    // Kiểm tra theo chiều dọc dưới
    for (short row = move.row - 1; row ≥ 0; --row) {
        if (board[row][move.col] == playerValue) {
            sameValueCount++;
            if (sameValueCount == Constants::WIN_VALUE_COUNT) {
                return true;
            }
        } else {
            break;
        }
    }

    return false;
}
```

Usage:

```
{
    /* Nếu một trong các phương kiểm tra trả về true
    thì nước đang xét là nước đi thắng*/
    if (CheckVerticalWin(board, move, playerValue)
    || CheckHorizontalWin(board, move, playerValue))
    || CheckLeftDiagonalWin(board, move, playerValue))
    || CheckRightDiagonalWin(board, move, playerValue))
    {
        return Logic::WIN_VALUE;
    }
}
```

2.1.9.3. Cách phát hiện nước đi hòa

Trận đấu sẽ đạt kết quả hòa nếu:

- Không có người chơi nào thắng.
- Không còn vị trí trống để thực hiện nước đi kế tiếp.

Từ đó, ngay sau việc kiểm tra nước thắng, nếu không có người thắng, ta sẽ có gọi hàm kiểm tra nước hòa thỏa mãn các điều kiện trên.

Implementation:

```
bool CheckDraw(const short& moveCount) {
    return (moveCount == BOARD_SIZE * BOARD_SIZE);
}
```

Usage:

```
{
    if(CheckDraw(moveCount) == true) {
        return Logic::DRAW_VALUE;
    }
}
```

2.1.10. Các tương tác với bàn cờ

Trạng thái bàn cờ có thể được thay đổi qua hai hình thức: **thực hiện nước đi** và **xóa nước đi**. Hàm `MakeMove` và `UndoMove` đảm nhiệm việc thực hiện hai chức năng ấy. Hai hàm này tuy đơn giản nhưng nắm vai trò quan trọng xuyên suốt

quá trình chơi, vì các thao tác người chơi chỉ thực sự được ghi lại trên bàn cờ khi hai hàm này được gọi đến. Việc xóa nước đi là để phục vụ cho chức năng **Hoàn tác** của trò chơi.

Interface:

```
 // Thực hiện nước đi lên bàn cờ
 void MakeMove(
    Board& board,
    short& moveCount,
    const Point& move,
    const short& playerValue
);

// Xóa nước đi khỏi bàn cờ
void UndoMove(
    Board& board,
    short& moveCount,
    const Point& move
);
```

Parameters:

- `board` : Bàn đấu hiện tại.
- `moveCount` : Số nước đi đã thực hiện.
- `move` : Nước đi được thực hiện.
- `playerValue` : Người chơi thực hiện nước đi.

Usage:

```
{
    // ...
    /* Gán vị trí hiện tại của con trỏ là nước đi mới nhất.*/
    latestMove = {row, col};
    GameAction::MakeMove(
        board,
        moveCount,
        latestMove,
        currentPlayer);
}
```

2.1.11. Chế độ đánh với máy

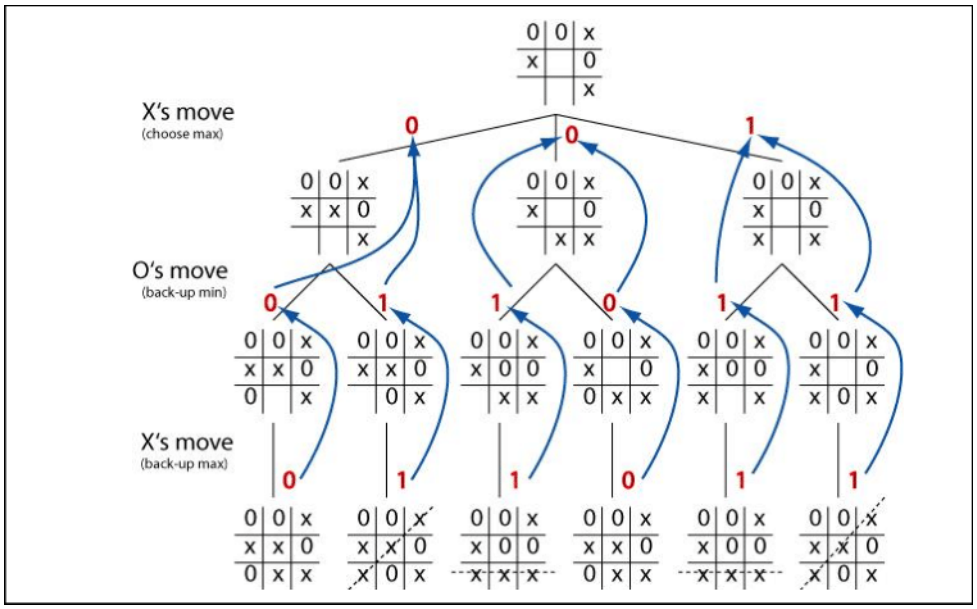
Việc thiết kết chương trình cho chế độ **"Đánh với máy"** là một trong những thách thức lớn nhất của đồ án. Khác với những tính năng khác của chương trình, tính năng này đòi hỏi những mảng kiến thức chuyên biệt về các thuật toán, kĩ thuật cụ thể. Ngoài ra, việc đánh giá độ đúng/sai của chương trình, hay nói cách khác là nước đi máy tính tìm được là tốt hay xấu, sẽ phần lớn phụ thuộc vào cảm tính và sự hiểu biết của người viết. Chính vì vậy, chương trình có thể đánh hay đối với người này, nhưng đánh không tốt đối với người khác. Phần tiếp theo sẽ trình bày những kĩ thuật mà nhóm đã sử dụng cho chức năng này.

2.1.11.1. Thuật toán Minimax

Đối với những trò chơi đối kháng hai người như cờ vua, cờ caro,... Để có thể tìm được một nước tốt qua code, ta không thể lập trình cứng để xét thế nào là nước đi tốt, thế nào là nước đi xấu được. Đơn thuần là vì có quá nhiều trường hợp để xét, và làm như thế chưa thể đảm bảo có thực sự đạt hiệu quả hay không. Thay vào đó, ta cần phải dựa vào thế mạnh của máy tính. Tuy không thể tư duy như con người, nhưng máy tính có khả năng thực hiện hàng chục nghìn phép tính trong một khoảng thời gian rất ngắn. Dựa vào đặc tính ấy, ta phát triển được phương pháp tìm một nước đi tốt cho cờ Caro.

Thuật toán Minimax[11] là một thuật toán phổ biến được áp dụng trong việc tìm kiếm một nước đi tốt trong các trò chơi đối kháng giữa hai người. Chính vì vậy, nhóm đã quyết định sử dụng thuật toán này để viết nên chương trình "AI" cho trò chơi. Giải thích một cách đơn giản, thuật toán sẽ tìm nước đi tốt nhất thông qua việc đánh giá tất cả các nước đi có thể trong mỗi lượt đi. Ví dụ, đối với cờ Caro, nếu hiện tại là lượt của người chơi O, thuật toán sẽ tìm mọi nước đi có thể của người chơi O. Sau khi đã thực hiện lượt chơi của O, thuật toán sẽ tìm mọi nước đi có thể của người chơi X. Quá trình này sẽ lặp lại đến một độ sâu nhất định, và khi đã đến độ sâu cuối cùng, một phép đánh giá tương đối sẽ được thực hiện để đánh giá "điểm" của bàn cờ. Người chơi **"tối đa hóa"** sẽ cố gắng

đạt được bàn cờ có điểm số cao nhất, ngược lại, người chơi **"tối thiểu hóa"** sẽ cố gắng đạt được bàn cờ có điểm số thấp nhất.



Hình 38: Sơ đồ tìm kiếm Minimax đối với trò chơi Tic-Tac-Toe

Ta sẽ áp dụng thuật toán này vào việc thiết kế chương trình AI của trò chơi. Mỗi khi người chơi hoàn thành thực hiện lượt chơi của mình, hàm `GetBestMove` sẽ được gọi để tìm nước đi tốt nhất cho lượt chơi của AI. Bên trong hàm `GetBestMove`, ta thực hiện việt tìm kiếm nước đi tốt nhất thông qua thuật toán Minimax.

Interface:

```
GameAction::Point GetBestMove(
    GameAction::Board& board,
    short& moveCount
);
```

Parameters:

- `board`: bàn cờ hiện tại
- `moveCount`: số nước đi đã thực hiện

Return

- Nước đi tốt nhất AI tìm được cho lượt đánh hiện tại

Usage:

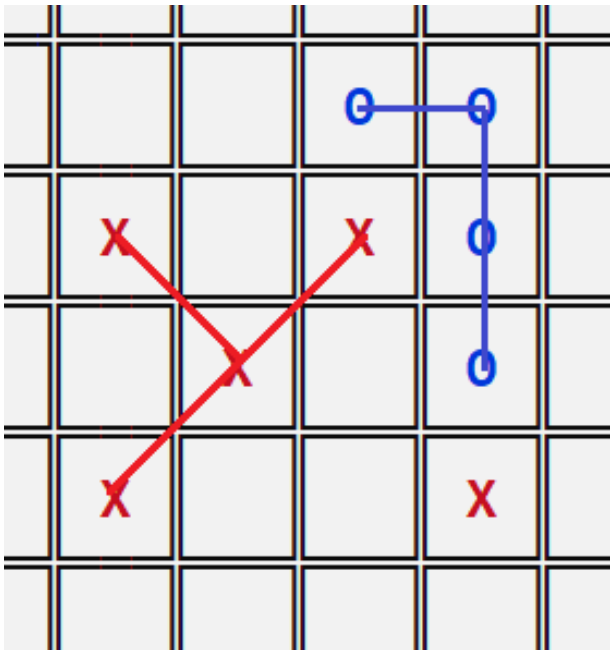
```
{
    // Tìm nước đi tốt nhất của AI
    GameAction::Point aiMove = AI::GetBestMove(board, moveCount);
    // Thực hiện nước đi AI
    GameAction::MakeMove(board, moveCount, aiMove);
}
```

2.1.11.2. Đánh giá bàn cờ

Thành phần quan trọng nhất trong thuật toán Minimax là **hàm đánh giá trạng thái**. Cần phải biết được trong một bàn cờ nhất định, lợi thế đang thuộc về người chơi nào. Trong cờ Caro, ta thấy rằng mục tiêu của mỗi nước đánh đều sẽ cố đạt được chuỗi 5 nước đồng chất liên tiếp. Ta gọi đó là chuỗi ấy là chuỗi **5 combo**. Để có thể đạt được 5 combo, ta phải có được chuỗi 4 nước đồng chất liên tiếp, gọi là chuỗi 4 combo. Và tương tự, muốn được 4 combo ta phải có 3 combo, muốn có 3 combo ta phải có nước 2 combo,... Nhìn chung, có thể thấy người chơi có được combo với độ dài càng gần với 5, họ sẽ có tỉ lệ thắng cao hơn. Ngoài ra, giả sử hai người chơi có **số lượng combo** 1 và combo 2 như nhau, thì người có số lượng combo 3 hay combo 4 lớn hơn sẽ có lợi thế cao hơn. Cuối cùng, một combo **bị chặn** (có một quân cờ của đối phương ở một hoặc cả hai đầu mút của combo) càng ít sẽ cho lợi thế càng cao. Vậy, dựa vào những tính chất ấy, ta xây dựng được thuật toán đánh giá như sau:

- Với một bàn cờ cho trước, ta duyệt qua tất cả những nước đi đã được thực hiện.
- Tại vị trí của mỗi nước đi, ta tính số điểm combo nước đó mang lại:
    - Ta gán cho mỗi combo một số điểm, combo có độ dài càng cao sẽ cho số điểm càng lớn.
        - Trong chương trình, combo 1, 2, 3, 4 không bị chặn có số điểm tương ứng là 1, 10, 50, 200.
    - Combo bị chặn một đầu sẽ bị giảm một nửa số điểm, bị chặn hai đầu sẽ không có điểm.

    - Combo sẽ được tính theo ba hướng: dọc, ngang và chéo. Ta lấy tổng số điểm của các hướng này trả lại.
- Nếu nước đi đang xét có cùng giá trị với người chơi đang xét, ta cộng số điểm combo tính được vào tổng điểm đánh giá. Ngược lại, nếu nước đi đang xét khác với người chơi đang xét, ta trừ giá trị điểm tính được khỏi tổng điểm đánh giá.
- Mỗi combo chỉ được xét một lần, tức nếu một nước thuộc một combo đã được duyệt từ trước, thì ta sẽ không xét combo cùng hướng tính từ nước này.



Hình 39: Minh họa combo mỗi người chơi qua đường nối liền

Implementation:

```
short Evaluation::GetComboEval(
    const GameAction::Board& board, const short& playerValue
)
{
    // Bảng đánh dấu combo
    GameAction::Board comboCheckBoard(
        Constants::BOARD_SIZE,
        std::vector<short>(Constants::BOARD_SIZE, 210)
    );

    short evalResult = 0;
    for (short row = 0; row < Constants::BOARD_SIZE; ++row) {
        for (short col = 0; col < Constants::BOARD_SIZE; ++col) {
            if (board[row][col]) {
                short evalValue = 0;
                // Combo chưa được đánh dấu sẽ được kiểm tra
                // Lấy số điểm có được từ combo theo phương ngang
                if (comboCheckBoard[row][col] % 2 == 0) {
                    short eval = GetHorizontalComboEval(
                        board, comboCheckBoard,
                        {row, col}, board[row][col]
                    );
                    // Cộng vào tổng điểm của nước đi này
                    evalValue += eval;
                }
                // Lấy số điểm có được từ combo theo phương dọc
                if (comboCheckBoard[row][col] % 3 == 0) {
                    short eval = GetVerticalComboEval(
                        board, comboCheckBoard,
                        {row, col}, board[row][col]
                    );
                    evalValue += eval;
                }
                /*Lấy số điểm có được từ combo theo phương chéo
                phải sang trái*/
                if (comboCheckBoard[row][col] % 5 == 0) {
                    short eval = GetDiagonalRightComboEval(
                        board, comboCheckBoard,
                        {row, col}, board[row][col]
                    );
                    evalValue += eval;
                }
```

```
                if (comboCheckBoard[row][col] % 7 == 0) {
                /*Lấy số điểm có được từ combo theo phương chéo
                trái sang phải*/
                    short eval = GetDiagonalLeftComboEval(
                        board, comboCheckBoard,
                        {row, col}, board[row][col]
                    );
                    evalValue += eval;
                }

                if (board[row][col] == playerValue)
                    // Cộng vào tổng điểm đánh giá của bàn cờ
                    evalResult += evalValue;
                else
                    // Trừ khỏi tổng điểm đánh giá của bàn cờ
                    evalResult -= evalValue;
            }
        }
    }
    return evalResult;
}
```

Usage:

```
{
    // Kết thúc một lần tìm kiếm của Minimax
    if (depth == 0) {
        return GetComboEval(board, playerValue);
    }
}
```

Khi áp dụng hàm đánh giá ấy vào thuật toán Minimax, với độ sâu tìm kiếm bằng 2, các nước đi chương trình tìm được đã thỏa yêu cầu ta đặt ra: các nước đi đều hướng đến việc tạo combo với độ dài càng cao càng tốt, và với số lượng càng nhiều càng tốt. Đồng thời, trong hàm đánh giá, ta cũng xét đến những combo mà đối thủ tạo nên, và trừ những số điểm đó đi khỏi kết quả đánh giá cuối cùng. Điều này giúp chương trình ngoài việc tìm những nước đi tạo combo mang lại lợi thế cao nhất, việc chặn combo của đối phương cũng được xem xét là những lựa chọn tốt.

2.1.11.3. Xử lí nước đi đầu tiên

Đối với trường hợp AI thực hiện nước đi đầu tiên, vì hiện không có bất kì nước đi nào trên bàn cờ để dựa trên mà đánh giá, ta sẽ cài đặt hàm `GetFirstMove` để xử lý việc này. Vì phần lớn các trận đấu Caro đều có nước đi nằm ở khoảng giữa của bàn cờ, vì khu vực này có nhiều khoảng không nhất, nên hàm `GetFirstMove` sẽ trả về một nước đi trong phạm vi đó.

Implementation:

```
inline GameAction::Point GetFirstMove()
{
    srand(time(NULL));
    // Tạo nước đi nằm ở khoảng giữa bàn cờ
    short row = Constants::BOARD_SIZE / 2 - 2 + (rand() % 3);
    short col = Constants::BOARD_SIZE / 2 - 2 + (rand() % 3);
    return {row, col};
}
```

Usage:

```
{
    // Lượt đầu tiên là của AI
    if (isAIFirst) {
        aiMove = GetFirstMove();
        /* ... */
    }
}
```

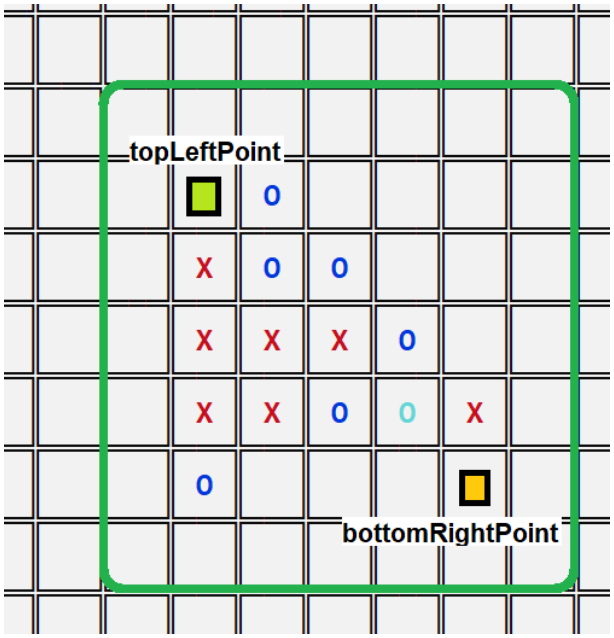
2.1.11.4. Cải thiện tốc độ

Một vấn đề có thể thấy rõ với AI hiện tại là thời gian tìm kiếm còn dài. Với độ sâu tìm kiếm bằng 3, trung bình mỗi lần tìm kiếm của AI mất khoảng 30 giây, với số lượng bàn cờ truy xét là hơn 100000 bàn cờ. Nếu giữ nguyên chương trình như vậy, người chơi sẽ dễ dàng cảm thấy chán nản khi phải đợi lượt đánh của AI. Ta cần phải có những biện pháp cải thiện tốc độ xử lý.

2.1.11.4.1. Giới hạn phạm vi tìm kiếm

Hiện giờ, thuật toán Minimax đang xét tất cả nước đi có thể thực hiện của toàn bộ bàn cờ, nhưng việc làm này rất tốn kém và mất thời gian. Trong cờ Caro, các

nước đi thường sẽ nằm liền kề nhau, tạo nên một phạm vi mà phần lớn các quân cờ đều nằm bên trong. Lí do là vì những nước đi tách biệt quá xa khỏi phạm vi ấy thường là những nước đi không tốt, không mang lại lợi thế cho người chơi. Dựa vào việc này, ta sẽ giới hạn phạm vi tìm kiếm của thuật toán Minimax để có thể giảm thời gian xử lí. Gọi `topLeftPoint` là vị trí có row bằng row của quân cờ cao nhất, col bằng col của quân cờ trái cùng nhất, `bottomRightPoint` là vị trí có row bằng row của quân cờ thấp nhất, col bằng col của quân cờ phải cùng nhất. Khi ấy, phạm vi tìm kiếm của chúng ta sẽ là một hình chữ nhật như **hình 4**. Lưu ý, `topLeftPoint` và `bottomRightPoint` không cố định, mà sẽ biến đổi trong quá trình tìm kiếm của Minimax. Tức nếu nước đi thử của Minimax nằm ngoài khu vực tìm kiếm hiện có, thì hai giá trị trên sẽ được cập nhật để mở rộng khu vực tìm kiếm.

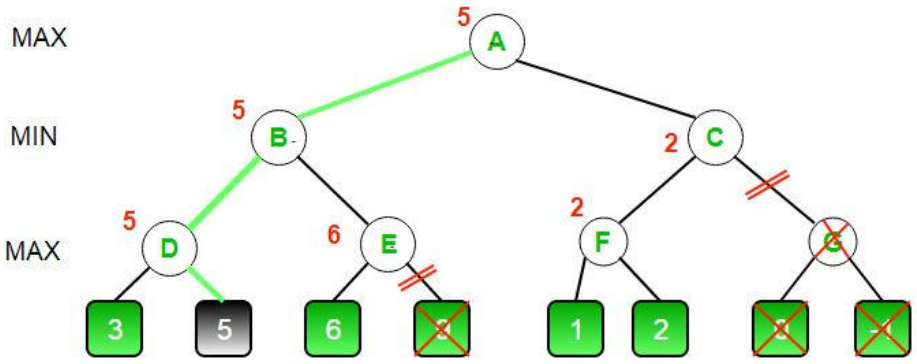


Hình 40: Giới hạn tìm kiếm minh họa qua khung màu xanh

Với sự cải thiện này, trong một trận đấu mà các quân cờ nằm gần nhau, thời gian xử lý sẽ được rút ngắn đi. Tuy nhiên, sự rút ngắn ấy không đáng kể, chưa kể người chơi có thể thực hiện hai nước đi ở hai góc đối của bàn cờ, từ đó khiến cho việc giới hạn phạm vi trở nên vô ích.

2.1.11.4.2. Alpha-Beta pruning

Một phương pháp hiệu quả để tăng tốc thuật toán Minimax là kĩ thuật **Alpha-Beta pruning**[12]. Ý tưởng là nếu nước đi đang xét có thể được chứng minh là tệ hơn một nước đi đã tìm được trước đó, thì ta sẽ ngừng truy xét nước đi này. Hay nói cách khác, ta sẽ "tỉa" những đoạn kiểm tra không cần thiết khỏi quá trình truy xét. Việc này giúp giảm số lượng nước đi phải kiểm tra, từ đó tăng tốc độ xử lý của thuật toán.



Hình 41: Quá trình cắt tỉa thông qua Alpha-Beta pruning

2.1.11.4.3. Sắp xếp nước đi tìm kiếm

Phương pháp Alpha-Beta pruning chỉ thực sự phát huy hiệu quả khi ta kiểm tra những nước đi tốt trước[13]. Hiện giờ, thuật toán chỉ kiểm tra từng nước đi theo thứ tự tuần tự trên bàn cờ (từ trái sang phải, từ trên xuống dưới). Ta cần truy xét những nước đi theo một trật tự sao cho nước đi tốt được xét trước và nước đi xấu được xét sau. Thế nhưng, làm thế nào để biết một nước đi là nước đi tốt? Không phải chúng ta thực hiện thuật toán Minimax cũng là để tìm nước đi đó hay sao? Trong cờ vua, những nước đi như cho một quân có giá trị thấp ăn một quân có giá trị cao hơn có thể nói là một nước đi tốt, mặc dù ta không biết nó có ảnh hưởng lâu dài đến lúc sau hay không. Dựa vào ý tưởng đó, ta có thể viết một hàm phỏng đoán một nước đi có phải là nước đi tốt hay không. Sau đó, ta sắp xếp các nước đi có thể thực hiện vào một danh sách theo thứ tự **nước đi tốt nhất đến nước đi xấu nhất**, và cho thuật toán Minimax truy xét danh sách ấy. Hàm đảm nhiệm việc lập nên danh sách nước đi thỏa yêu cầu trên là hàm

`GetMoveList`. Ý tưởng đánh giá một nước đi tốt trong cờ Caro có thể được miêu tả như sau:

- Xét vị trí của một nước đi, ta kiểm tra xem trên một phương nhất định, còn thiếu bao nhiêu quân cờ để tạo nên một chuỗi 5 nước đồng chất.
- Nếu số quân cờ thiếu càng ít, thì số điểm gán cho nước đi đang xét sẽ càng cao, và ngược lại.
- Ta thực hiện việc kiểm tra đối với cả 8 phía xung quanh nước đi ấy, và lấy tổng của từng kết quả.
- Tiếp đến, ta kiểm tra trên một phía nhất định có bao nhiêu quân cờ của đối phương.
- Nếu số quân cờ của đối phương càng nhiều, số thì điểm gán cho nước đang xét sẽ càng cao, và ngược lại.
- Ta thực hiện việc kiểm tra đối với cả 8 phía xung quanh nước đi ấy, và lấy tổng của từng kết quả.
- Kết quả cuối cùng gán cho nước đi sẽ là tổng của việc xét quân cờ đồng chất và xét quân cờ đối phương.

Lí do tại sao không đánh giá điểm của nước đi theo cách đánh giá bàn cờ trong hàm `GetComboEval` là vì một nước đi tốt không nhất thiết phải là nước tạo nên chuỗi quân cờ đồng chất dài nhất. Nếu đối phương có 4 quân cờ liền kề nhau bị chặn một đầu, thì việc ta chặn đầu còn lại của chuỗi quân cờ ấy là một nước đi rất tốt.

Interface:

```
MoveQueue GetMoveList(
    short rowLowerLimit,
    short rowUpperLimit,
    short colLowerLimit,
    short colUpperLimit,
    short moveCount,
    GameAction::Board& board,
    short playerValue
);
```

Parameters:

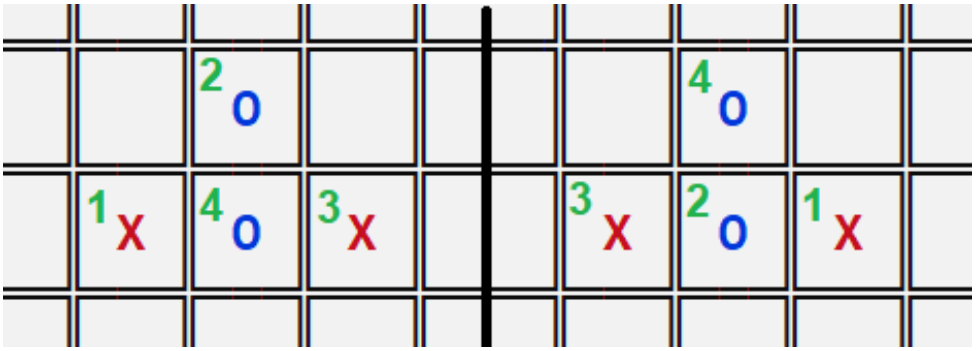
- `rowLowerLimit` : giới hạn tìm kiếm dưới của dòng
- `rowUpperLimit` : giới hạn tìm kiếm trên của dòng
- `colLowerLimit` : giới hạn tìm kiếm trái của cột
- `colUpperLimit` : giới hạn tìm kiếm phải của cột
- `moveCount` : số nước đi đã thực hiện
- `board` : bàn cờ hiện tại
- `playerValue` : giá trị người chơi đang Xét

Usage:

```
{
    MoveQueue moveList = GetMoveList(
        rowLowerLimit,
        rowUpperLimit,
        colLowerLimit,
        colUpperLimit,
        moveCount, board,
        currentPlayer);
    while (!moveList.empty()) {
        GameAction::Point move = moveList.top();
        /* Thực hiện Minimax với move ... */
    }
}
```

2.1.11.4.4. Transposition table (Bảng hoán vị)

Trong quá trình tìm kiếm của thuật toán Minimax, sẽ có nhiều trường hợp một trạng thái bàn cờ bị lặp lại theo thứ tự nước đi khác nhau. Việc này sẽ gây lãng phí thời gian truy xét, vì ta đang thực hiện lại phép tính đã có kết quả từ trước.



Hình 42: Hai thứ tự nước đi khác nhau cùng đạt một bàn cờ

Để khắc phục được việc này, ta cần một bảng lưu trữ những kết quả đánh giá có được của các bàn cờ đã xét, để khi gặp lại những bàn cờ ấy, ta trả về giá trị lưu trong bảng, từ đó tránh việc phải lặp lại phép tính. Một bảng lưu trữ như vậy được gọi là **bảng hoán vị** (Transposition table)[14]. Để thực hiện yêu cầu ấy, ta sử dụng cấu trúc dữ liệu `unordered_map`[15] trong C++. Cấu trúc dữ liệu này lưu trữ dữ liệu theo hình thức **key-value pair**, với mỗi key là một giá trị đơn nhất. Dựa vào tính chất ấy, giả sử key trong trường hợp này là một bàn cờ nhất định, thì value lúc này sẽ là kết quả đánh giá của bàn cờ ấy. Tuy nhiên, bàn cờ trong chương trình được lưu dưới dạng một mảng hai chiều, `vector<vector<short>>`, `unordered_map` không hỗ trợ key với cấu trúc dữ liệu ấy. Vì vậy, ta cần mã hóa bàn cờ thành một giá trị mà có thể sử dụng để làm key. Việc này có thể được thực hiện thông qua kĩ thuật **Zobrist Hashing**[16]. Kĩ thuật này giúp chúng ta chuyển hóa một bàn cờ hai chiều thành một con số đơn nhất, từ đó có thể sử dụng con số ấy làm key cho bảng lưu. Những con bot cờ vua, cờ vây cũng sử dụng kĩ thuật này để mã hóa bàn cờ[17].

2.1.11.4.5. So sánh tốc độ

Sau khi áp dụng những phương pháp trên, chương trình hiện giờ đã có thể truy xét với độ sâu bằng 3 trong một khoảng thời gian rất ngắn. Tuy việc tìm hiểu và cài đặt những phương pháp này đã mất rất nhiều thời gian, nhưng so với tốc độ ban đầu của chương trình, thì sự đầu tư này thực sự rất thỏa đáng.

Phương pháp	**1**	**1 + 2**	**1 + 2 + 3**	**1 + 2 + 3 + 4**
Thời gian trung bình	30 giây	15 giây	3 giây	Gần như tức thì
Số bàn cờ truy xét	100000	30000-60000	5000-10000	1000-3000

Bảng 2: Bảng thống kê tốc độ (độ sâu 3)

Chú thích:

- **1**: Phương pháp giới hạn bàn cờ.
- **2**: Phương pháp Alpha-Beta pruning.
- **3**: Phương pháp sắp xếp thứ tự tìm kiếm.
- **4**: Phương pháp bảng hoán vị.

2.1.11.5. Phân độ khó

Sau khi áp dụng những kĩ thuật để tối ưu hóa thuật toán, độ sâu tối đa mà chương trình có thể thực hiện trong một khoảng thời gian hợp lý là 3. Vì vậy, ta có thể phân độ khó của chế độ đánh với máy với những độ khó:

- **Dễ**: chiều sâu 1
- **Trung bình**: chiều sâu 2
- **Khó**: chiều sâu 3

Với độ sâu bằng 1, thuật toán chỉ kiểm tra tất cả nước đi của 1 lượt duy nhất, vì vậy, tuy các nước đi vẫn sẽ cố gắng tạo nên các combo dẫn đến chuỗi thắng, máy không thể nhìn trước được những đòn tấn công do đối phương gây nên. Với độ sâu bằng 2, thuật toán sẽ có thể kiểm tra thêm những nước đi của đối phương, vì vậy, máy có thể ngăn chặn những mối nguy do đối phương gây nên, như là những nước tạo nên combo 3, combo 4. Với độ sâu bằng 3, không chỉ gây khó dễ cho đối phương qua việc phòng thủ, vì 2 trong 3 lượt kiểm trà là lượt của máy, nên thuật toán sẽ tìm được nhiều nước đi tấn công hơn, và trong cờ Caro, người chơi có thế chủ động thường sẽ có lợi thế cao hơn.

2.1.11.6. Chức năng "Gợi ý"

Ngoài chế độ đánh với máy, ta có thể tận dụng tốc độ xử lý nhanh của AI vào một chức năng khác của trò chơi, đó là chức năng **"Gợi ý"**. Thay vì cố định giá trị người chơi như trong chế độ đánh với máy (người là người chơi X, máy là người chơi O), mỗi khi đến lượt đánh của một người chơi nào đó, khi sử dụng chức năng "Gợi ý", ta sẽ gán giá trị người chơi AI là người chơi hiện tại, từ đó, tìm ra một nước đi tốt cho người chơi ấy.

Implementation:

```
GameAction::Point GameScreenAction::GetHintMove(
    GameAction::Board& board,
    short moveCount,
    bool isPlayerOneTurn,
    AI ai
)
{
    if (isPlayerOneTurn) {
        ai.PLAYER_AI = Constants::PLAYER_ONE.value;
        ai.PLAYER_HUMAN = Constants::PLAYER_TWO.value;
    } else {
        ai.PLAYER_AI = Constants::PLAYER_TWO.value;
        ai.PLAYER_HUMAN = Constants::PLAYER_ONE.value;
    }
    ai.SetDifficulty(AI::AI_DIFFICULTY_HARD);
    if (moveCount == 0) return ai.GetFirstMove();
    return ai.GetBestMove(board, moveCount);
}
```

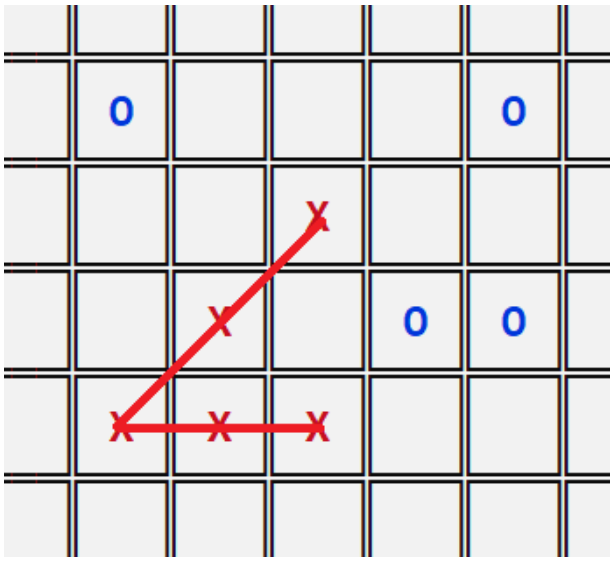
Usage:

```
{
    // Thực hiện chức năng Gợi ý
    auto hintMove = GameScreenAction::GetHintMove(
        board,
        moveCount,
        isPlayerOneTurn,
        ai
    );
}
```

2.1.11.7. Những mặt cần cải thiện

Tuy chương trình hiện tại đã có thể đánh tương đối như một người chơi bình thường, vẫn còn nhiều mặt ta có thể cải thiện. Trong đó, việc tìm ra một hàm đánh giá để đưa ra được những đánh giá chính xác hơn của trạng thái bàn cờ là điều quan trọng nhất. Hàm đánh giá hiện tại có một khuyết điểm lớn là không đưa ra đánh giá đúng đắn đối với những trạng thái "đặc biệt" như **hình 7**. Các trạng thái này mang đến lợi thế rất lớn cho người chơi, nói thẳng hơn là họ gần như đã giành được chiến thắng. Hàm đánh giá hiện có chỉ coi những trạng

thái trên như những combo tách biệt, từ đó đưa ra những đánh giá không chính xác. Ngoài ra, việc cài đặt các hàm trong chương trình còn rườm rà, thiếu hiệu quả, chỉ mang tính chất hoạt động tức thời, không được tính toán kĩ càng. Ngoài những phương pháp cải thiện tốc độ nêu trên, một điều người lập trình cần chú ý là việc cài đặt chương trình sao cho mọi thứ hoạt động nhanh và hiệu quả nhất.



Hình 43: Lợi thế thắng của đỏ qua hai combo 3 nối liền

2.2. Giao diện

2.2.1. Các màn hình lưu, tải và phát lại ván đấu

Các màn hình này đều có cấu trúc tương tự nhau, gồm:

- Danh sách các file: hiển thị danh sách các file đã lưu, trang hiện tại và tổng số trang.
- Ô nhập văn bản: cho phép người chơi nhập tên file cần lưu, tải hoặc phát lại hoặc tìm kiếm trong danh sách các file đã lưu.
- Các hướng dẫn: hiện các nút và chức năng tương ứng.

Vì sự tương tự này, em đã tạo một class chung là `SaveLoadScreenViewModel` bao gồm các trạng thái và chức năng chung của các màn hình này. class nằm trong `namespace Common` trong file `Common.h` và `Common.cpp`.

Interface(Đã loại bỏ một số hàm nhỏ):

```
using namespace std::filesystem::path;
typedef std::vector<std::pair<std::wstring, path>> OptionList;

class SaveLoadScreenViewModel {
    // Các trạng thái chung
    path allOptionsDir;
    int selected = 0;
    int currentPage = 0;
    bool isSearching = 0;
    int maxPage = 0;

    // Các file đã tìm được trong thư mục
    OptionList allOptions;
    // Dữ liệu để hiển thị lên màn hình
    std::vector<View::Option> options;
    std::wstring pageIndicator; // Hiện số trang
    std::wstring searchInput; // Nội dung của ô tìm kiếm

    SaveLoadScreenViewModel(path dir)
    {
        allOptionsDir = dir;
    }

    // Tải lại danh sách các file
    void ReloadAllOptions()
    {
        LoadAllOptions(allOptionsDir);
        UpdatePage(currentPage);
    }

    // Đọc tất cả các file trong thư mục
    void LoadAllOptions(path dir);
    // Tìm kiếm các file có tên chứa chuỗi searchInput
    bool Search();

    // Tạo ra một hàm để chạy
    // khi người chơi thay đổi nội dung ô tìm kiếm
    std::function<void(const std::wstring&)> onSearchValueChange(
        const std::function<void(void)>& callback
    );
};
```

Usage:

```
// Sử dụng class trong màn hình lưu
class SaveScreenState : public Common::SaveLoadScreenViewModel {
  public:
    bool Save(const GameState& currentGameState);
};
```

2.2.1.1. Hiển thị danh sách các file đã lưu

Để hiển thị danh sách các file đã lưu, trò chơi cần tải danh sách các file đã lưu trong một thư mục nhất định, sau đó sắp xếp tăng dần theo thời gian rồi chỉnh sửa một chút để hiển thị lên phần "Danh sách các file".

Implementation:

```
void Common::SaveLoadScreenViewModel::LoadAllOptions(path dir)
{
    auto availableLoadFiles
        = SaveLoad::DiscoverSaveFiles(dir);
    std::sort(
        availableLoadFiles.begin(),
        availableLoadFiles.end(),
        [](const FileHandle::FileDetail& a,
           const FileHandle::FileDetail& b) {
            return a.lastModified > b.lastModified;
        }
    );

    allOptions.clear();
    allOptions.reserve(availableLoadFiles.size());
    for (auto& file : availableLoadFiles) {
        allOptions.emplace_back(
            Utils::CatStringSpaceBetween(
                50, file.filePath.filename(),
                Utils::filesystem_time_to_wstr_local
                                (file.lastModified)
            ), file.filePath
        );
    }
    maxPage = allOptions.size() / 10
            + bool(allOptions.size() % 10);
}
```

2.2.1.2. Ô tìm kiếm hoặc nhập tên

Ô tìm kiếm sử dụng thành phần giao diện Input để đọc đầu vào của người dùng. Mỗi khi người dùng thay đổi nội dung ô tìm kiếm, trò chơi sẽ sắp xếp các file có tên chứa chuỗi mà người dùng đã nhập lên đầu và cập nhật lại danh sách các file hiển thị lên màn hình. Để làm được điều đó, em đã tạo một hàm để tìm kiếm các file có tên chứa chuỗi mà người dùng đã nhập, hàm sẽ duyệt qua danh sách các file đã lưu, rồi sắp xếp đưa các file phù hợp lên đầu danh sách rồi cập nhật lên màn hình. Việc tìm kiếm mỗi lần người dùng nhập một ký tự sẽ làm chậm trò chơi do phải liên tục vẽ lại danh sách lên màn hình, để giảm thiểu điều này, trước khi thực hiện việc sắp xếp, kiểm tra xem danh sách đã được sắp xếp hay chưa, nếu chưa thì mới sắp xếp.

Implementation:

```
struct SortTemporary {
    size_t foundIndex = 0;
    std::wstring name;
    size_t mapIndex = 0;
}

bool Common::SaveLoadScreenViewModel::Search()
{
    static auto cmp = [](
        const SortTemporary& a,
        const SortTemporary& b
    ) {
        if (a.foundIndex ≠ size_t(-1)
            && b.foundIndex ≠ size_t(-1))
        {
            return a.name > b.name;
        }

        if (a.foundIndex == b.foundIndex) {
            return a.name > b.name;
        }

        return a.foundIndex < b.foundIndex;
    };
```

```
    size_t n = allOptions.size();
    std::vector<SortTemporary> tmp;
    tmp.resize(n);

    for (size_t i = 0; i < n; i++) {
        auto ttt = allOptions[i].second.filename().wstring();
        tmp[i] = {ttt.find(searchInput), std::move(ttt), i};
    }
    // Kiểm tra xem danh sách có cần sắp xếp hay không
    if (std::is_sorted(tmp.begin(), tmp.end(), cmp)) {
        return 0;
    }

    std::sort(tmp.begin(), tmp.end(), cmp);

    OptionList vtmp;
    vtmp.resize(n);
    for (size_t i = 0; i < n; i++) {
        vtmp[i] = allOptions[tmp[i].mapIndex];
    }
    allOptions = vtmp;
    return 1;
}
```

2.2.2. Màn hình game chính

Trong tất cả các màn hình được cài đặt trong chương trình, màn hình game chính là màn hình có cấu trúc phức tạp nhất. Ngoài việc xử lý giao diện của các nước đi và các tính năng bổ trợ (cảnh báo nước 4, nháp, gợi ý) trên bàn cờ, còn phải chú tâm đến các thành phần khác như khung trạng thái (chứa thời gian, số trận thắng của người chơi), khung avatar và khung lịch sử nước đi. Ngoài ra, vì trong quá trình chơi, người chơi có thể tạm ngừng ván đấu, và tiếp tục ngay sau đó, nên việc lưu trữ trạng thái và hiển thị bàn cờ hiện tại cũng trở thành một vấn đề phải đề cập đến.

2.2.2.1. GameScreenView

Phần chương trình đảm nhiệm cho giao diện của màn hình game nằm trong view `GameScreenView`. Đoạn code nằm trong đây có thể được chia làm 3 phần:

- Phần 1: khai báo các biến cần thiết, tiền xử lý và khởi động các thao tác trước khi bắt đầu vào vòng lặp chính.

- Phần 2: vòng lặp chính, mọi sự tương tác của người chơi đều được xử lý trong đây.
- Phần 3: xử lý các nghiệp vụ sau khi ván đấu kết thúc.

Implementation:

```
void GameScreenView::GameScreenView(NavigationHost& NavHost) {
    // Phần 1
    NavHost.SetContext(
        Constants::NEXT_VIEW,
        Constants::NULL_VIEW
    );
    NavHost.SetContext(
        Constants::IS_SAVED,
        false
    );
    NavHost.SetContext(
        Constants::CURRENT_BGM,
        Audio::Sound::GameBGM
    );
    /* ... */
    GameAction::Board gameBoard(
        Constants::BOARD_SIZE,
        std::vector<short>(Constants::BOARD_SIZE, 0)
    );
    /* ... */
    GameScreen gameScreen(7, 2);
    gameScreen.DrawGameScreen();
    gameScreen.DrawToElements(curGameState);
    /* ... */

    // Phần 2
    while (!endGame) /* Vòng lặp chính */

    // Phần 3
    /* ... */
    curGameState.gameEnd = endGame;
    NavHost.SetContext(Constants::FINISHED_GAME, curGameState);
    return NavHost.Navigate("GameEndView");
}
```

Ngoài ra, mọi thao tác được xử dụng trong `GameScreenView` như xử lý nước đi của người chơi, thực hiện các tính năng bổ trợ, xử lý kết thúc ván đấu,... đều

nằm trong `namespace GameScreenAction`. Việc tách ra như vậy sẽ giúp chương trình dễ dàng được kiểm soát hơn. Nếu gộp hết tất cả vào một nơi, thì phần code cho đoạn này sẽ dài hơn 1000 dòng.

Interface:

```
namespace GameScreenAction {
    // Cập nhật cả frontend và backend của ván đấu
    void UpdateGame(
        GameScreen gameScreen,
        GameAction::Board& board,
        short& moveCount,
        const GameAction::Point& move,
        const Constants::Player& player,
        GameState& gameState,
        bool loadFromSave = false
    );

    // Nổi bật con trỏ di chuyển
    void HighLightCursor(
        GameScreen& gameScreen,
        const GameAction::Board& gameBoard,
        const GameAction::Point& curPos,
        const ColorMatrix& colorMatrix,
        std::mutex& lock,
        bool isGhostMode
    );
    /* ... */
}
```

2.2.2.1.1. Class GameScreen

Giao diện của trò chơi được dựng nên và xử lý qua `class GameScreen`, và được cấu thành bởi 2 thành phần là: bàn cờ và các khung bổ trợ. Tương ứng, ta có `class BoardContainer` và `class Container`.

2.2.2.1.2. Class Container

Đối với thành phần sau, các "container" đơn thuần là những khung hình chữ nhật được vẽ qua hàm `View::DrawRect` tại một tọa độ, với chiều dài và chiều rộng lúc khai báo, và có vai trò "chứa" những thông tin của bàn cờ. Để có thể điền vào những container này, ta gọi method `DrawToContainer`, với tham số là

giá trị muốn được hiển thị. Ví dụ, container `timerContainerOne` sẽ hiển thị thời gian hiện tại của người chơi 1, còn container `winCountContainerOne` sẽ hiển thị số trận thắng của người chơi 1. Mỗi container thường chỉ hiển thị một giá trị nhất định, nhưng đối với `logContainer`, vì số lượng thông tin hiển thị nhiều và phức tạp hơn, ta cần một method riêng tên `DrawToLogContainer`.

Interface:

```
class Container {
    // Vẽ khung của container
    void DrawContainer();

    // Vẽ giá trị bên trong container
    void DrawToContainer(
        std::wstring value,
        View::Color color =
            Theme::GetColor(ThemeColor::TEXT_COLOR)
    );

    // Vẽ giá trị bên trong container lịch sử nước đi
    void DrawToLogContainer(
        const std::vector<std::pair<short, short>>& moveList,
        const std::wstring& playerNameOne,
        const std::wstring& playerNameTwo,
        bool playerOneFirst,
        short winMethod = 0,
        bool isReplay = false,
        short goBack = 0
    );
};
```

Parameter

- `value`: giá trị hiển thị trong container
- `color`: màu sắc vẽ giá trị
- `moveList`: danh sách nước đi hiện tại
- `playerNameOne`, `playerNameTwo`: tên người chơi 1, người chơi 2
- `playerOneFirst`: xác nhận lượt đầu tiên là của người chơi 1
- `winMethod`: kết quả ván đấu
- `isReplay`: xác nhận đang vẽ trong màn hình phát lại

- `goBack`: số lần cuộn lên của container

Usage:

```
{
    Container timerPlayerOne, logContainer;
    /*Thực hiện gán giá trị xCoord, yCoord, ... cho container ... */

    // Vẽ thời gian người chơi 1
    timerPlayerOne.DrawToContainer(
        L"05:00",
        Theme::GetColor(ThemeColor::PLAYER_ONE_COLOR));

    // Vẽ danh sách các nước đi đã thực hiện
    logContainer.DrawToLogContainer(
        moveList,
        L"Adam",
        L"Bob",
        true,
        Constants::ENd_GAME_WIN_ONE);
}
```

2.2.2.1.3. Class BoardContainer

Tương tự như các container, bàn cờ cũng sẽ có hai phần là: vẽ giao diện của bàn cờ và hiển thị giá trị của quân cờ. Vì các quân cờ đều có những vị trí khác nhau, nên ta cần thực hiện phép tính để quân cờ hiện vào đúng vị trí tương ứng trên bàn cờ.

Interface:

```
class BoardContainer {
    // Vẽ giao diện bàn cờ
    void DrawBoardContainer();
    void DrawBoardRow();
    void DrawBoardCol();
    // Vẽ các label theo chiều ngang bàn cờ
    void DrawBoardHorizontalLabels();
    // Vẽ các label theo chiều dọc bàn cờ
    void DrawBoardVerticalLabels();
```

```
    // Vẽ quân cờ vào ô cờ
    void DrawToBoardContainerCell(
        short row,
        short col,
        std::wstring value,
        View::Color color,
        bool highlight,
        bool isGhostMode
    );
};
```

2.2.2.2. Xử lý nước đi

Mọi tương tác của người chơi đều sẽ được xử lý trong vòng lặp chính của `GameScreenView`. Cụ thể hơn, khi người chơi ấn một phím **lệnh** (phím di chuyển, phím chức năng, phím tạm ngừng,...) thì các thao tác tương ứng sẽ được thực hiện trong `GameScreenAction`. Đặc biệt, khi người chơi thực hiện nước đi (phím Enter), hàm `HandlePlayerMove` sẽ được gọi, đây là hàm xử lý nước đi của người chơi. Mỗi khi hàm trên được sử dụng, các hàm xử lý con như `HighLightMove`, `UnhighlightMove`, `FlipTurn` sẽ được kích hoạt để xử lý giao diện cho nước đi và logic của ván đấu. Nhưng quan trọng nhất là hai hàm con `UpdateGame` và `HandleState`:

- `UpdateGame`: đây là nơi duy nhất có quyền sử dụng đến hàm `MakeMove` được đề cập ở những mục trên. Hay nói cách khác, chỉ khi hàm này được gọi thì nước đi của người chơi mới được lưu lại trên bàn cờ. Ngoài ra, hàm cũng lưu nước đi đó vào danh sách nước đi hiện tại của bàn cờ, nhằm phục vụ việc lưu trữ sau này.
- `HandleState`: mỗi khi một nước đi được thực hiện, hàm `HandleState` sẽ có vai trò kiểm tra nước đi đó có phải là nước đi kết thúc ván đấu hay không (nước đi thắng hoặc nước đi hòa). Nếu có, hàm sẽ thực hiện những thao tác kết thúc ván đấu, từ đó chuyển `GameScreenView` sang phần hậu xử lý.

Nếu không có hai hàm này, tuy vẫn có giao diện, trò chơi sẽ không thể hoạt động.

Bên trong hàm HandleState

```
    /* ... */
    // Lấy trạng thái ván đấu
    short state = Logic::GetGameState(
        board, moveCount, move, player.value, winPoint, true
    );
    switch (state) {
        // Xử lý các trạng thái tương ứng
        case Logic::WIN_VALUE:
            if (isPlayerOneTurn) {
                curGameState.playerScoreOne++;
                endGame = Constants::END_GAME_WIN_ONE;
            } else {
                curGameState.playerScoreTwo++;
                endGame = Constants::END_GAME_WIN_TWO;
            }
            HightLightWin(
                move, winPoint,
                player.symbol, gameScreen
            );
            break;
        case Logic::DRAW_VALUE:
            endGame = Constants::END_GAME_DRAW;
            break;
    }
    /* ... */
```

Bên trong hàm UpdateGame

```
// Thực hiện nước đi
GameAction::MakeMove(board, moveCount, move, player.value);
if (!loadFromSave) {
    // Lưu nước đi vào danh sách nước đi
    if (move.row != -1) {
        gameState.moveList.push_back({move.row, move.col});
    }
    gameScreen.logContainer.DrawToLogContainer(
        gameState.moveList,
        gameState.playerNameOne,
        gameState.playerNameTwo,
        gameState.playerOneFirst
    );
}
```

2.2.2.3. Lưu và load trạng thái ván đấu

Trong quá trình chơi, sẽ có lúc người chơi tạm ngưng ván đấu, và quay lại một lúc sau đó. Khi ấy, màn hình game sẽ chuyển sang màn hình tạm ngưng, tức đã thoát khỏi scope của `GameScreenView`, và khi người chơi quay lại, một `GameScreenView` mới sẽ được tạo nên, với dữ liệu khác dữ liệu ván đấu đang diễn ra. Vì vậy, cần phải xử lý việc lưu trữ và load ván đấu hiện tại. Đối với việc lưu, ngay khi người chơi tạm ngưng, trạng thái ván đấu hiện tại sẽ được lưu qua đoạn code bên dưới:

```
// Tạm ngưng ván đấu
if (tmp == L"ESC") {
    // Lưu trạng thái ván đấu vào context tương ứng
    NavHost.SetContext(Constants::CURRENT_GAME, curGameState);
    // Chuyển sang màn hình tạm ngưng
    return NavHost.Navigate("PauseMenuView");
}
```

Từ đó, khi qua một `GameScreenView` mới, ta chỉ cần lấy trạng thái ván đấu qua context `Constants::CURRENT_GAME`

```
// Bên trong phần 1 của GameScreenView
{
    /* ... */
    GameState curGameState =
    std::any_cast<GameState>
    (NavHost.GetFromContext(Constants::CURRENT_GAME));
    /* ... */
}
```

Sau khi đã có trạng thái của ván đấu, việc tiếp theo là hiển thị các nước đi đã thực hiện lên bàn cờ, và cập nhật những biến cần thiết. Việc này được thực hiện trước khi vào vòng lặp chính của `GameScreenView`, qua hàm `LoadGameToView`

Interface:

```
void GameScreenAction::LoadGameToView(
    GameScreen& gameScreen,
    GameAction::Board& board,
    short& moveCount,
    GameState& gameState,
    AI& ai,
    std::vector<GameAction::Point>& warningPointList,
    ColorMatrix& colorMatrix,
    std::mutex& lock
);
```

Parameters:

- `gameScreen`: màn hình game.
- `board`: bàn cờ hiện tại.
- `moveCount`: số nước đi đã thực hiện.
- `gameState`: trạng trái ván đấu hiện tại.
- `ai`: AI sử dụng cho ván đấu.
- `warningPointList`: danh sách các nước cảnh báo 4.
- `colorMatrix`: bảng màu sắc của bàn cờ.
- `lock`: khóa mutex.

Usage(lấy từ hàm `GameScreenView`):

```
    /* Sau khi có được trạng thái ván đấut*/
    // Vẽ màn hình game
    gameScreen(7, 2);
    gameScreen.DrawGameScreen();
    gameScreen.DrawToElements(curGameState);
    // Load trạng thái game hiện tại vào các biến và màn hình game
    GameScreenAction::LoadGameToView(
        gameScreen,
        gameBoard,
        moveCount,
        curGameState, myAI,
        warningPointList,
        colorMatrix,lock
    );
    /* ... */
```

2.2.3. Các màn hình khác

Các màn hình còn lại có cấu trúc rất tương tự với nhau. Gồm có 3 phần:

- Lấy các dữ liệu cần thiết để vẽ màn hình
- Vẽ màn hình
- Chạy vòng lặp để lấy input từ người dùng
- Xử lý input người dùng.

Do đó chúng em sẽ chỉ nêu chi tiết màn hình "Menu Chính".

Màn hình "Menu Chính" gồm có một menu ở giữa hiện lên các tùy chọn khả dụng cho người chơi, một phần chỉ dẫn các nút ở phía dưới và các hình trang trí. Người chơi có thể sử dụng bàn phím để di chuyển con trỏ chọn, ngoài ra, để tăng sự tiện dụng cho người chơi, chúng em thêm vào các phím tắt, các tùy chọn khác nhau sẽ có phím tắt khác nha. Khi nhấn phím tắt, trò chơi sẽ chọn và thực hiện tùy chọn tương ứng. **Impplementation:**

```
void MainMenu::MainMenuScreen(NavigationHost& NavHost)
{
    // Kiểm tra thiết lập và chơi nhạc nền
    NavHost.SetContext(
        Constants::CURRENT_BGM,
        Audio::Sound::MenuBGM
    );
    if (Config::GetConfig(Config::BGMusic)
        == Config::Value_True)
    {
        if (BackgroundAudioService::GetCurrentSong()
            != Audio::Sound::MenuBGM)
        {
            BackgroundAudioService::
                ChangeSong(Audio::Sound::MenuBGM);
            BackgroundAudioService::Play(true, true);
        }
    }
```

```
    // Vẽ các phần không thay đổi
    Common::DrawHintsLess();
    Caro(32, 1);
    Logo_Deadpool(3, 5);
    Logo_Captain(79 + 4, 5);
    View::WriteToView(
        119 - Constants::version.size(), 0,
        Constants::version
    );
    static short selectedOption = 0;
    static const short maxOption = 7;
    auto& soundEffect = Config::GetConfig(L"SoundEffect");
    View::DrawMenuPrevState menuPrevState;

    // Các phím tắt của các tùy chọn
    std::vector<std::wstring> shortcuts = {
        Language::GetString(L"NEW_GAME_SHORTCUT"),
        Language::GetString(L"LOAD_SHORTCUT"),
        Language::GetString(L"REPLAY_SHORTCUT"),
        Language::GetString(L"SETTINGS_SHORTCUT"),
        Language::GetString(L"TUTORIAL_SHORTCUT"),
        Language::GetString(L"ABOUT_SHORTCUT"),
        Language::GetString(L"EXIT_SHORTCUT")

    };
    // Các tùy chọn của menu
    std::vector<View::Option> options = {
        {Language::GetString(L"NEW_GAME_TITLE"), shortcuts[0][0]},
        {Language::GetString(L"LOAD_TITLE"),     shortcuts[1][0]},
        {Language::GetString(L"REPLAY_TITLE"),   shortcuts[2][0]},
        {Language::GetString(L"SETTINGS_TITLE"), shortcuts[3][0]},
        {Language::GetString(L"TUTORIAL_TITLE"), shortcuts[4][0]},
        {Language::GetString(L"ABOUT_TITLE"),    shortcuts[5][0]},
        {Language::GetString(L"EXIT_TITLE"),     shortcuts[6][0]},
    };
    while (1) {
        // Vẽ menu
        View::DrawMenu(
            menuPrevState, 50, 13, L"",
            options, selectedOption
        );
```

```
        auto tmp = InputHandle::Get();  // Lấy input từ người dùng
        // Kiểm tra và chơi hiệu ứng
        if (soundEffect == Config::Value_True) {
            if (tmp == L"\r") {
                Utils::PlaySpecialKeySound();
            } else {
                Utils::PlayKeyPressSound();
            }
        }
        // Xử lý input
        if (Utils::keyMeanUp(tmp)) {
            selectedOption = Utils::modCycle(
                selectedOption - 1,
                options.size()
            );
        }
        if (Utils::keyMeanDown(tmp)) {
            selectedOption = Utils::modCycle(
                selectedOption + 1,
                options.size()
            );
        }
        if (Utils::ShortcutCompare(tmp, shortcuts[0])) {
            return NavHost.Navigate("GameModeTypeView");
        }
        //  ...
        if (tmp == L"\r") {
            switch (selectedOption) {
                case 0:
                    return NavHost.Navigate("GameModeTypeView");
                case 1:
                     //  ...
            }
        }
    }
}
```

Chương 3.

Kết luận

3.1. Kết quả đạt được

3.1.1. Ưu điểm của trò chơi

- Có thể thêm nhiều ngôn ngữ và theme vào trò chơi
- Có nhạc hay, hiệu ứng sống động
- Có chế độ tính thời gian
- AI chạy tương đối tốt, đánh nhanh
- Lối chơi đa dạng
- Có nhiều nhân vật ngộ nghĩnh
- Có nhiều tính năng hỗ trợ khi chơi trò chơi
- Có thể xem lại trận đấu đã chơi
- Màn hình save/load có khả năng tương tác tốt
- Hướng dẫn dễ hiểu, có gợi ý ở mỗi màn hình

3.1.2. Khuyết điểm của trò chơi

- Giao diện chưa được bắt mắt
- AI chưa được tối ưu, còn chặn luồng chính khi chạy
- Không thể tùy chỉnh kích thước màn hình
- Không chỉnh được thời gian trong chế độ tính thời gian
- Phần tìm kiếm làm cho danh sách các file bị lag nếu người dùng gõ quá nhanh

3.2. Các khó khăn gặp phải

- Một vài thành viên không có kinh nghiệm sử dụng git và GitHub
- Khó khăn khi lập trình đa luồng do không có kinh nghiệm
- Màn hình terminal vẽ các kí tự chậm

- Khó khăn trong việc thiết kế AI do có nhiều kiến thức mới, lạ
- Chưa có kinh nghiệm trong việc thiết kế giao diện, viết ứng dụng có giao diện

3.3. Những gì đã học được

- Cách làm việc nhóm với git và GitHub
- Cách sử dụng các tính năng mới của C++ 20
- Cách sử dụng các tính năng liên quan tới đo hiệu năng, format code và debug trong Visual Studio
- Cách làm việc nhóm hiệu quả
- Cách lên kế hoạch, phân chia công việc
- Cách lập trình hướng đối tượng, lập trình đa luồng cơ bản

3.4. Các kinh nghiệm rút ra

- Khi code, nên tách nhỏ các hàm ra, không nên viết hàm dài
- Code xong một hàm phải kiểm tra kĩ, tránh phát sinh lỗi về sau
- Không nên viết code mà không thiết kế trước
- Nên viết code theo một quy chuẩn nhất định và đồng bộ trong 1 dự án
- Nên viết code có tính tái sử dụng cao
- Cần có kế hoạch và phân chia công việc rõ ràng
- Code xong phải có người review lại

3.5. Lí do hoàn thành mục tiêu

- Phân chia công việc hợp lí
- Tận dụng được thế mạnh của từng thành viên
- Thường xuyên gặp nhau để trao đổi, thảo luận về dự án

3.6. Hướng phát triển ứng dụng

- Có thể chơi 2 người qua mạng lan
- Thêm nhiều ngôn ngữ mới
- Thêm nhiều chủ đề hơn
- Hiện lợi thế của 2 bên

- Đưa trò chơi lên nhiều nền tảng khác

Tài liệu tham khảo

[1] "Cờ ca-rô." https://vi.wikipedia.org/wiki/C%E1%BB%9D_ca-r%C3%B4

[2] "Gomoku." https://en.wikipedia.org/wiki/Gomoku

[3] "Playsound function." https://learn.microsoft.com/en-us/previous-versions/dd743680(v=vs.85)

[4] "Mci." https://learn.microsoft.com/vi-vn/windows/win32/multimedia/mci

[5] "Mcisendstring function." https://learn.microsoft.com/en-us/previous-versions//dd757161(v=vs.85)

[6] "Navigating with compose." https://developer.android.com/jetpack/compose/navigation

[7] "Android jetpack." https://developer.android.com/jetpack

[8] "Filesystem library." https://en.cppreference.com/w/cpp/filesystem

[9] "Std::thread." https://en.cppreference.com/w/cpp/thread/thread

[10] "Std::mutex." https://en.cppreference.com/w/cpp/thread/mutex

[11] "Minimax." https://vi.wikipedia.org/wiki/Minimax

[12] "Alpha-beta pruning." https://en.wikipedia.org/wiki/Alpha%E2%80%93beta_pruning

[13] "Move ordering." https://www.chessprogramming.org/Move_Ordering

[14] "Tranposition table." https://en.wikipedia.org/wiki/Transposition_table

[15] "Unordered map." https://en.cppreference.com/w/cpp/container/unordered_map

[16] "Zobrist hashing." https://en.wikipedia.org/wiki/Zobrist_hashing

[17] "Zobrist hashing in chess." https://www.chessprogramming.org/Zobrist_Hashing