

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
KỸ 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
KỸ 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

Mục lục

Lời cảm ơn	3
Mục lục	4
Danh sách hình	7
1. Hướng dẫn sử dụng	8
1.1. Hiển thị hình ảnh	8
1.2. Code có highlight (Paste code ít thôi, lấy cái nào quan trọng á)	8
2. Table	10
3. Tổng quan về trò chơi	11
3.1. Giới thiệu về trò chơi	12
3.1.1. Gomoku	12
3.1.2. Các yêu cầu về tính năng	12
3.1.3. Thông tin chung về trò chơi	12
3.2. Mô tả về các tính năng của game	12
3.2.1. Màn hình chính	12
3.2.2. Save/Load game đang chơi, replay game đã chơi xong	12
3.2.3. Xử lý, hiệu ứng thắng, thua, hòa	12
3.2.4. Giao diện màn hình khi chơi	13
3.2.5. Đa ngôn ngữ	13
3.2.6. Thay đổi Theme(Chủ đề)	13
3.2.7. Lưu thiết lập của người chơi	13
3.2.8. Chế độ chơi Thường	13
3.2.9. Chế độ chơi Rush	13
3.2.10. Đánh với máy	13
3.2.11. Đánh với người	13
3.2.12. Các hỗ trợ trong lúc chơi game	13
3.2.12.1. Gợi ý	13
3.2.12.2. Nổi bật nước mới đi	13
3.2.12.3. Cảnh báo nước 4	13
3.2.12.4. Hoàn tác nước đi	13

3.2.12.5. Di nháp	13
4. Chi tiết các chức năng	14
4.1. Logic	14
4.1.1. Chơi hiệu ứng, nhạc nền	14
4.1.1.1. Giải pháp để chơi các âm thanh của giao diện	14
4.1.1.2. Giải pháp để chơi nhạc nền	16
4.1.2. Điều hướng trong ứng dụng	20
4.1.3. Phương pháp lưu và tải game(save/load game)	23
4.1.3.1. Các hàm hỗ trợ khác	25
4.1.3.1.1. Các hàm hỗ trợ mở file	25
4.1.3.1.2. Hàm Ensure	26
4.1.3.1.3. Hàm Delete	26
4.1.4. Đếm giờ trong khi chơi game	27
4.1.5. Giải pháp cho đa ngôn ngữ	30
4.1.6. Cài đặt	32
4.1.7. Chủ đề	32
4.1.8. Hàm trung gian hỗ trợ vẽ giao diện	32
4.1.9. Nhận biết thắng thua	32
4.1.9.1. Hàm GetGameState	32
4.1.10. Các tương tác với bàn cờ	33
4.1.11. AI	34
4.1.11.1. Thuật toán Minimax	35
4.1.11.2. Đánh giá bàn cờ	35
4.1.11.3. Cải thiện tốc độ	35
4.1.11.3.1. Alpha-Beta pruning	35
4.1.11.3.2. Sắp xếp nước đi tìm kiếm	35
4.1.11.3.3. Transposition table (Bảng hoán vị)	35
4.1.11.3.4. So sánh tốc độ	35
4.1.11.4. Phân độ khó	35
4.1.11.5. Chức năng “Gợi ý”	35

4.1.11.6. Những mặt cần cải thiện	35
4.2. Giao diện	35
4.2.1. Màn hình chính	35
4.2.2. Cài đặt	35
4.2.3. Các màn hình lưu, tải game và replay	35
4.2.4. Màn hình trò chơi chính	35
4.2.5. Các màn hình khác	35
5. Đánh giá thành viên	36
6. Kết luận	37
6.1. Kết quả đạt được	37
6.1.1. Ưu điểm của trò chơi	37
6.1.2. Khuyết điểm của trò chơi	37
6.2. Các khó khăn gặp phải	37
6.3. Những gì đã học được	37
6.4. Các kinh nghiệm rút ra	38
6.5. Lí do hoàn thành mục tiêu	38
6.6. Hướng phát triển ứng dụng	38
Tài liệu tham khảo	38

Danh sách hình

Hình 1. Test fig	8
------------------------	---

Chương 1.

Hướng dẫn sử dụng

1.1. Hiện thị hình ảnh

```
#figure(  
    image("HCMUS_Logo.png", width: 40%),  
    caption: [Test fig]  
)
```

Nhớ bỏ caption vô không là cái mục lục hình bị lỗi



Hình 1: Test fig

1.2. Code có highlight (Paste code ít thôi, lấy cái nào quan trọng á)


```
while (1) {
    auto tmp = InputHandle::Get();
    if (tmp == L"b" || tmp == L"B") {
        return NavHost.Back();
    }
    if (tmp == L"\r") {
        system(
            "rundll32 url.dll,FileProtocolHandler "
            "https://github.com/thng292/CaroGame"
        );
    }
}
```

Chương 2.

Table

Thành viên	Đánh giá
Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do.	Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do.

Chương 3.

Tổng quan về trò chơi

3.1. Giới thiệu về trò chơi

3.1.1. Gomoku

Nguyên

3.1.2. Các yêu cầu về tính năng

- Có thể save, load trò chơi
- Nhận biết được thắng, thua, hòa
- Xử lý hiệu ứng thắng, thua, hòa
- Xử lý giao diện màn hình khi chơi
- Xử lý màn hình chính
- Game có nhiều ngôn ngữ, người dùng có thể thêm được ngôn ngữ mới
- Có thể load được các theme(chủ đề) bên ngoài
- Lưu được các thiết lập của người chơi
- Có thể lưu và phát lại các game đã hoàn thành
- Có nhiều chế độ chơi
- Có thể chơi với máy, máy có nhiều mức độ
- Game có thể phát nhạc nền, hiệu ứng. Có thể bật tắt được

3.1.3. Thông tin chung về trò chơi

Nguyên Link source code, chạy trên nền tảng nào, ...

3.2. Mô tả về các tính năng của game

3.2.1. Màn hình chính

3.2.2. Save/Load game đang chơi, replay game đã chơi xong

3.2.3. Xử lý, hiệu ứng thắng, thua, hòa

3.2.4. Giao diện màn hình khi chơi

3.2.5. Đa ngôn ngữ

3.2.6. Thay đổi Theme(Chủ đề)

3.2.7. Lưu thiết lập của người chơi

3.2.8. Chế độ chơi Thường

3.2.9. Chế độ chơi Rush

3.2.10. Đánh với máy

3.2.11. Đánh với người

3.2.12. Các hỗ trợ trong lúc chơi game

3.2.12.1. Gợi ý

3.2.12.2. Nổi bật nước mới đi

3.2.12.3. Cảnh báo nước 4

3.2.12.4. Hoàn tác nước đi

3.2.12.5. Đi nháp

Chương 4.

Chi tiết các chức năng

4.1. Logic

4.1.1. 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"  
};
```

4.1.1.1. 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` [1]. 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` \Rightarrow phát âm thanh thành công
- `false` \Rightarrow 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 load file vào memory, đọc và phát
- Sau khi gọi xong thì không cần quan tâm đến nữa nên có độ linh hoạt cao

Nhược điểm:

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

- Khi chơi có độ delay cao (có thể được khắc phục bằng cách load 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` \Rightarrow phát âm thanh một cách đồng bộ (synchronous)
 - `false` \Rightarrow phát âm thanh một cách bất đồng bộ (asynchronous)

Return:

- `true` \Rightarrow phát âm thanh thành công
- `false` \Rightarrow phát âm thanh thất bại

Usage:

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

4.1.1.2. 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` [2]. 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` [3]. 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 load hết file vào memory
- 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` \Rightarrow chơi từ đầu
 - `false` \Rightarrow chơi tiếp tại vị trí con trỏ
- `repeat`:
 - `true` \Rightarrow lặp lại khi kết thúc

Return:

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

Usage:

```
{  
    Audio::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` \Rightarrow chơi từ đầu
 - `false` \Rightarrow chơi tiếp tại vị trí con trỏ
- `repeat`:
 - `true` \Rightarrow lặp lại khi kết thúc

Usage:

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

4.1.2. 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` [4] (thư viện điều hướng của Jetpack [5]). 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 thành 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

4.1.3. Phương pháp lưu và tải game(save/load game)

Để lưu game, chúng em đã chọn lưu trạng thái của game vào một file văn bản thuần và lưu trong một thư mục riêng. Để tải game, chúng em sẽ đọc file văn bản đó và khởi tạo lại trạng thái của game. Khi người dùng muốn tải game, chúng em muốn hiển thị một danh sách các file lưu game để người dùng 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 game vào một file văn bản thuần, khi cần tải, chúng em sẽ đọc file đó và hiển thị cho người dùng. 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 dùng không thể tải game nếu file đó bị xóa/lỗi
- Người dùng không thể load các file copy từ máy khác

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

- Không cần tạo ra file không cần thiết
- Người dùng có thể tải game từ máy khác

Để quét các file trong thư mục, chúng em đã sử dụng thư viện `filesystem` [6]. 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. Để việc sử dụng thư viện thuận tiện hơn, chúng em đã viết hàm `GetAllTextFileInDir`.

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);
    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
```

Note: Chi tiết về hàm `Ensure` nằm ở Mục 4.1.3.1.2

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 = FileHandle::GetAllTextFileInDir(
        "saves"
    );
    for (auto& file:files) {
        std::cout << file.filePath.filename() << '\n';
    }
}

```

4.1.3.1. Các hàm hỗ trợ khác

Các hàm sau nằm trong namespace `FileHandle`, file `FileHandle.h`, `FileHandle.cpp`

4.1.3.1.1. Các hàm hỗ trợ mở file

Hỗ trợ mở các file văn bản `utf-8`

Interface:

```

typedef std::filesystem::path fsPath;
std::wofstream OpenOutFile(const fsPath& filePath);
std::wifstream OpenInFile (const fsPath& filePath);

```

Parameters:

- `filePath`: đường dẫn đến file cần mở

Usage:

```

#include <string>
{
    std::wstring str = L"Tiếng Việt";
    auto outFile = FileHandle::OpenOutFile("test.txt");
    outFile << str;
    outFile.close();
    auto inFile = FileHandle::OpenInFile ("test.txt");
    inFile >> str;
}

```

4.1.3.1.2. Hàm Ensure

Dùng để đảm bảo đường dẫn đến file muốn mở có tồn tại, nếu không tồn tại, nếu không tồn tại thì tạo đường dẫn đó.

Interface:

```
void Ensure(const std::filesystem::path& Dir);
```

Parameters:

- `Dir`: đường dẫn muốn kiểm tra/tạo

Return:

- Các fstream tương ứng với thao tác In/Out

Usage:

```
{  
    // Đảm bảo đường dẫn tương đối "asset/language" tồn tại  
    FileHandle::Ensure("asset/language");  
}
```

4.1.3.1.3. Hàm Delete

Dùng để xóa file

Interface:

```
bool Delete(const std::filesystem::path& target)
```

Parameters:

- `target`: đường dẫn tới file cần xóa

Return:

- Trả về `true` nếu xóa thành công, `false` khi lỗi

Usage:

```

{
    // Xóa file tmp.cpp
    bool res = FileHandle::Delete("tmp.cpp");
    if (res) {
        std::cout << "Success";
    } else {
        std::cout << "Failed";
    }
}

```

4.1.4. Đếm giờ trong khi chơi game

Việc đếm và hiển thị thời gian trực tiếp trong lúc chơi khá phức tạp vì luồng chính trong game luôn phải chờ đợi và xử lý đầu vào của người dùng, nên việc sử dụng luồng chính để nó phụ thuộc vào việc người chơi có thực hiện các thao tác trong game hay không. Nếu người chơi không thực hiện các thao tác trong game 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` [7] 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` [8] 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);
            auto currPos = View::GetCursorPos();
            // Vẽ thời gian lên màn hình
            gameScreen.timerContainerOne.DrawToContainer(
                Utils::SecondToMMSS(curGameState.playerTimeOne),
                Theme::GetColor(ThemeColor::PLAYER_ONE_COLOR)
            );
            View::Goto(currPos.X, currPos.Y);
            if (curGameState.playerTimeOne == 0 && !endGame) {
                // Xu'ly khi hết thời gian
            }
        }
    }
);
}
```

4.1.5. Giải pháp cho đa ngôn ngữ

Các văn bản trong trò chơi thay vì được code cứng vào trò chơi thì sẽ được load từ một file riêng, điều này kiến cho phần ngôn ngữ trong game 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 load xong, các nhãn và văn bản sẽ được lưu vào một bảng để có thể dễ dàng truy xuất.

Ví dụ file ngôn ngữ:

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

Các nhãn và văn bản trong trò chơi được quản lí, truy xuất và load 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 {
    static Dict languageDict;

public:
    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");
}
```

4.1.6. Cài đặt

Thông

4.1.7. Chủ đề

Thông

4.1.8. Hàm trung gian hỗ trợ vẽ giao diện

Thông

4.1.9. Nhận biết thắng thua

Việc nhận biết kết quả thắng, thua, và hòa của một ván đấu được thực hiện trong Cpp 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.

4.1.9.1. 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:


```

short GetGameState(
    const GameAction::Board& board,
    const short& moveCount,
    const GameAction::Point& move,
    const short& playerValue,
    GameAction::Point& winPoint = temp,
    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` \Rightarrow lấy đầu mút của chuỗi thắng (nếu có).
 - `false` \Rightarrow không lấy đầu mút của chuỗi thắng.

Returns:

- `Logic::WIN_VALUE` : Giá trị tượng trưng kết quả thắng.
- `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);

```

4.1.10. 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, và là một phần không thể thiếu đối với thuật toán **Minimax** được sử dụng cho AI 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:

```
GameAction::MakeMove(board, moveCount, latestMove, currentPlayer);
GameAction::UndoMove(board, moveCount, latestMove, currentPlayer);
```

4.1.11. AI

Việc thiết kế 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, kỹ thuật lập trình 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 kỹ thuật mà nhóm đã sử dụng cho chương trình AI.

4.1.11.1. Thuật toán Minimax

4.1.11.2. Đánh giá bàn cờ

4.1.11.3. Cải thiện tốc độ

4.1.11.3.1. Alpha-Beta pruning

4.1.11.3.2. Sắp xếp nước đi tìm kiếm

4.1.11.3.3. Transposition table (Bảng hoán vị)

4.1.11.3.4. So sánh tốc độ

4.1.11.4. Phân độ khó

4.1.11.5. Chức năng “Gợi ý”

4.1.11.6. Những mặt cần cải thiện

4.2. Giao diện

4.2.1. Màn hình chính

4.2.2. Cài đặt

Thông

4.2.3. Các màn hình lưu, tải game và replay

Thông

4.2.4. Màn hình trò chơi chính

Vũ

4.2.5. Các màn hình khác

Chương 5.

Đánh giá thành viên

Chương 6.

Kết luận

6.1. Kết quả đạt được

6.1.1. 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 game
- 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

6.1.2. Khuyết điểm của trò chơi

-

6.2. 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

6.3. 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 tối đa 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

6.4. 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 kỹ, 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

6.5. Lí do hoàn thành mục tiêu

Nguyên

6.6. 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 game lên nhiều nền tảng khác

Tài liệu tham khảo

- [1] “Playsound function.” [https://learn.microsoft.com/en-us/previous-versions/dd743680\(v=vs.85\)](https://learn.microsoft.com/en-us/previous-versions/dd743680(v=vs.85))
- [2] “Mci.” <https://learn.microsoft.com/vi-vn/windows/win32/multimedia/mci>
- [3] “Mcisendstring function.” [https://learn.microsoft.com/en-us/previous-versions/dd757161\(v=vs.85\)](https://learn.microsoft.com/en-us/previous-versions/dd757161(v=vs.85))

- [4] “Navigating with compose.” <https://developer.android.com/jetpack/compose/navigation>
- [5] “Android jetpack.” <https://developer.android.com/jetpack>
- [6] “Filesystem library.” <https://en.cppreference.com/w/cpp/filesystem>
- [7] “Std::thread.” <https://en.cppreference.com/w/cpp/thread/thread>
- [8] “Std::mutex.” <https://en.cppreference.com/w/cpp/thread/mutex>