|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  | | | | |  |
|  | **ĐẠI HỌC BÁCH KHOA HÀ NỘI TRƯỜNG ĐIỆN – ĐIỆN TỬ**  **---****---**    **BÁO CÁO MÔN HỌC**  **CẤU TRÚC DỮ LIỆU VÀ GIẢI THUẬT**  **(Mã học phần: ET2100)**  ***ĐỀ TÀI: CHESS AI ENGINE***  ***Giảng viên: Phạm Doãn Tĩnh*** | | | | |  |
|  | | ***Thực hiện: Nhóm 8*** | | |  | |
| **STT** | **Họ và Tên** | **MSSV** |
| 1 | Nguyễn Trần Trí Dũng | 20213691 |
| 2 | Đỗ Thị Thúy Nga | 20213720 |
| 3 | Đỗ Trang Đức Duy | 20213692 |
| 4 | Kim Minh Quang | 20213726 |
| 5 | Nguyễn Thị Minh Tâm | 20210766 |
| ***Hà Nội, 07/2023*** | | | | | | |
|  |  | | | | |  |

MỤC LỤC

Contents

[LỜI MỞ ĐẦU 3](#_Toc140005304)

[A. NỘI DUNG ĐỀ TÀI 4](#_Toc140005305)

[I. Giới thiệu đề tài 4](#_Toc140005306)

[II. Mục tiêu hướng đến 4](#_Toc140005310)

.[B. QUÁ TRÌNH THỰC HIỆN 5](#_Toc140005315)

[I. Phân tích bài toàn, lựa chọn cấu trúc dữ liệu và giải thuật 5](#_Toc140005316)

[1.1Tổng quan 5](#_Toc140005317)

[1.2 Biểu diễn bàn cờ và quân cờ bằng các cấu trúc dữ liệu 5](#_Toc140005324)

[1.3 Thuật toàn tìm kiếm 6](#_Toc140005486)

[a)Thuật toán Minimax 6](#_Toc140005487)

[b)Cắt tỉa Alpha-Beta 6](#_Toc140005488)

[1.4 Giao diện người dùng 7](#_Toc140005489)

[II. Tiến hành cài đặt chương trình 8](#_Toc140005490)

[2.1 Khai báo thư viện 8](#_Toc140005491)

[2.2 Lớp quản lý trò chơi 8](#_Toc140005492)

[2.3 Phần của hàm Create() trong class GameManager 10](#_Toc140005493)

[Hàm này được sử dụng để gán các giá trị cần thiết cho danh sách các quân cờ trong trò chơi. 10](#_Toc140005494)

[2.4 Các nước đi quân cờ 11](#_Toc140005495)

[2.5 Khởi tạo hàm Play() và hàm IncreasePositive() 15](#_Toc140005496)

[2.6 Hàm undo() và move() 20](#_Toc140005497)

[2.7. Hàm đánh giá, giải thuật Minimax và cắt tỉa Alpha-Beta, hàm chọn nước đi 23](#_Toc140005498)

[C. Kiểm tra 32](#_Toc140005499)

[I. Giao diện, sự kiện click chuột 32](#_Toc140005500)

[1.1Giao diện 33](#_Toc140005501)

[1.2 Sự kiện click chuột, nút undo 33](#_Toc140005502)

[II. Tính hợp lệ các nước đi 33](#_Toc140005503)

[III: Tốc độ phản hồi và độ ‘thông minh’ của mô hình 34](#_Toc140005504)

[D. Kết luận 34](#_Toc140005505)

# LỜI MỞ ĐẦU

Cấu trúc dữ liệu và giải thuật là học phần rất quan trọng đối với người lập trình. Môn học này được xem như nền tảng của lập trình máy tính. Nó là cơ sở vững chắc để giải quyết một số bài toàn, đồng thời cung cấp cho chúng ta hiểu biết về các giải thuật tác động lên dữ liệu, cũng như cách tổ chức dữ liệu để giải quyết các bài toàn sao cho hiệu quả và tối ưu.

Thông qua quá trình làm bài tập lớn môn này, chúng em đã nắm bắt được những kỹ thuật quan trọng của việc xây dựng cấu trúc dữ liệu và các xây dựng một thuật toàn sao cho tối ưu nhất.

Chúng em xin chân thành cảm ơn thầy Phạm Doãn Tĩnh đã giúp đỡ chúng em thực hiện bài tập lớn này. Và chúng em cũng rất mong sự góp ý của thầy để bài làm được hoàn thiện hơn.

Nhóm chúng em xin chân thành cảm ơn!

Nhóm 8

# A. NỘI DUNG ĐỀ TÀI

## I. Giới thiệu đề tài

# Đứng dưới góc nhìn của những người thích chơi cờ vua, chúng em xây dựng một mô hình Chess AI đơn giản có khả năng triển khai các quân cờ cùng các thế cờ để chơi cùng.

# Chúng em sử dụng thư viện SFML để khởi tạo bàn cờ, quân cờ và hiển thị các ô nước đi của mỗi quân cờ bằng hình ảnh.

# 

## II. Mục tiêu hướng đến

# Mô hình có khả năng triển khai những nước đi của các quân cờ khác nhau trong bàn cờ.

# Mô hình có khả năng tính toán được nước đi tối ưu dựa vào thuật toán.

# Cung cấp giao diện tương tác đơn giản cho người dùng, cho phép họ chơi cờ với mô hình và nhận phản hồi trực tiếp.

# Xây dựng một codebase dễ dàng mở rộng và bảo trì. Điều này cho phép thêm tính năng mới, cải thiện hiệu suất và sửa lỗi trong tương lai.

# B. QUÁ TRÌNH THỰC HIỆN

## I. Phân tích bài toàn, lựa chọn cấu trúc dữ liệu và giải thuật

### 1.1Tổng quan

# Biểu diễn bàn cờ và quân cờ: chúng ta cần xây dựng một cấu trúc dữ liệu để biểu diễn bàn cờ và các quân cờ. Thông thường, một ma trận 2D được sử dụng để biểu diễn bàn cờ, trong đó mỗi ô trống được đại diện bởi giá trị không và mỗi quân cờ được đại diện bởi một ký tự hoặc một số.

# Di chuyển và kiểm tra hợp lệ: triển khai các quy tắc di chuyển và kiểm tra tính hợp lệ của các nước đi. Điều này bao gồm kiểm tra xem quân cờ có thể di chuyển đến vị trí mới, kiểm tra xem nước đi có bị chiếu, kiểm tra xem có quân cờ nào cản trở và xử lý các quy tắc đặc biệt như nước đi quân tốt hai ô.

# Thuật toán tìm kiếm: một phần quan trọng của engine cờ vua là thuật toán tìm kiếm để xác định nước đi tốt nhất. Một trong những thuật toán phổ biến là Alpha-Beta Pruning hoặc Negamax. Thuật toán này sẽ dựa trên việc tạo ra cây tìm kiếm của các nước đi có thể và sử dụng các phương pháp cắt tỉa để giảm số lượng nút cần phân tích.

# Cải tiến và tối ưu: một engine AI cờ vua liên tục được cải tiến và tối ưu hóa. Bạn có thể thử nghiệm và điều chỉnh các tham số của thuật toán tìm kiếm và hàm đánh giá để cải thiện hiệu suất và khả năng chơi của engine.

# Giao diện người dùng: một giao diện người dùng đơn giản và trực quan cho phép người dùng tương tác với engine. Điều này có thể bao gồm hiển thị bàn cờ, cho phép người chơi đi nước, hiển thị nước đi của engine, và cung cấp các tùy chọn khác như mức độ khó, undo, hay kiểm tra các trạng thái.

# Kiểm thử và đánh giá: để đảm bảo tính đúng đắn và hiệu quả của engine, nên thực hiện kiểm thử kỹ lưỡng và đánh giá hiệu suất của nó. Bạn có thể kiểm tra engine với các trường hợp kiểm thử, so sánh với các engine cờ vua khác, và cải thiện nếu cần.

### 1.2 Biểu diễn bàn cờ và quân cờ bằng các cấu trúc dữ liệu

# Chúng em lựa chọn sử dụng mảng hai chiều để lưu trữ bàn cờ. Mảng sẽ khởi tạo vị trí ban đầu của các quân cờ bằng các chỉ số, những ô cờ còn lại sẽ mặc định giá trị bằng 0.

# Các quân cờ sẽ được đánh chỉ số tương ứng:

|  |  |  |
| --- | --- | --- |
| Quân cờ | Người chơi | Máy |
| Tốt | 6 | -6 |
| Xe | 1 | -1 |
| Tượng | 2 | -2 |
| Mã | 3 | -3 |
| Hậu | 4 | -4 |
| Vua | 5 | -5 |

# Các quân cờ cũng sẽ được định giá trị tương ứng:

|  |  |  |
| --- | --- | --- |
| Quân cờ | Người chơi | Máy |
| Tốt | 10 | -10 |
| Xe | 50 | -50 |
| Tượng | 30 | -30 |
| Mã | 30 | -30 |
| Hậu | 90 | -90 |
| Vua | 900 | -900 |

# Định nghĩa một cấu trúc để lưu trữ chỉ số, giá trị và Sprite (hình ảnh quân cờ) tương ứng, sau đó sử dụng một mảng gồm 33 phần tử để lưu kiểu dữ liệu này.

# Sử dụng một mảng Vector2f có trong thư viện SFML để lưu trữ các vị trí di chuyển có thể cho từng quân cờ, mảng này sẽ có 32 phần tử.

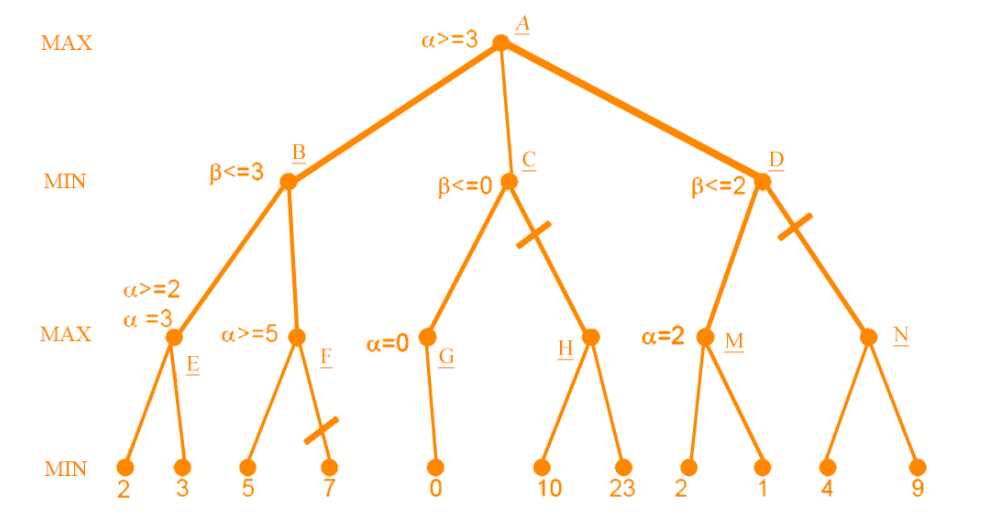
# Sử dụng Stack để lưu vị trí và chỉ số của quân cờ di chuyển, từ đó phục vụ cho tác vụ undo và thuật toàn tìm kiếm.

### 1.3 Thuật toàn tìm kiếm

# a)Thuật toán Minimax

* Minimax là một thuật toàn đệ quy lựa chọn bước đi kế tiếp trong một trò chơi bằng cách định giá trị cho các Node trên cây trò chơi sau đó tìm nút có giá trị phù hợp để đi bước tiếp, sử dụng tính chất **vét cạn**
* Tại sao chọn Minimax? Với game cờ vua thì khi chơi ta có thể triển khai hết không gian trạng thái nhưng khó khăn chủ yếu là phải tính toán được nước đi và phản ứng của đối thủ, vậy nên cách đơn giản là giả sử đối thủ cũng sử dụng kiến thức về không gian trạng thái giống mình. -> Giải thuật Minimax áp dụng giả thuyết này để tìm kiếm không gian trạng thái của trò chơi.
* Nói một cách đơn giản, đây là giải thuật tìm kiếm nước đi “tốt nhất” bằng cách đi ngược từ cuối trò chơi trở về đầu. Tại mỗi bước, nó sẽ ước định rằng người A đang cố gắng **tối đa hóa** cơ hội thắng của A khi đến lượt A, còn ở nước kế tiếp thì người B cố gắng **tối thiểu hóa** cơ hội thắng của người A.
* Áp dụng vào trò chơi, ta đi đến mức thấp nhất của cây, dùng hàm đánh giá tính điểm sau đó xác định điểm của các nhánh trên: với quân trắng là MAX, đen thì ngược lại.
* Dùng DFS (duyệt theo chiều sâu) để tìm, nhưng vì không gian trạng thái của trò chơi quá lớn nên ta sẽ giới hạn độ sâu của cây lại.

# b)Cắt tỉa Alpha-Beta

* Tuy đã giới hạn độ sâu nhưng không gian vẫn còn quá lớn nên ta sẽ cắt đi những nhánh không cần thiết, dùng để tối ưu hóa Minimax
* Nguyên tắc “Nếu biết là trường hợp xấu thì không cần phải xét thêm”
* Nút Max có một giá trị alpha (lớn hơn hoặc bằng alpha – luôn tăng), nút min có một giá trị beta (nhỏ hơn hoặc bằng beta – luôn giảm). Khi chưa có alpha và beta xác định thì thực hiện DFS để xác định được alpha, beta và truyền ngược lên nút cha
  + 
* VD:
  1. Ban đầu, chưa có các giá trị alpha, beta
  2. Xét từ trái qua phải, thấp nhất đến cao nhất
  3. Xét nút E,nút con đầu tiên là 2, vậy nên alpha >=2, nút tiếp theo là 3 -> alpha = 3
  4. Từ nút E -> beta của nút B <=3
  5. Xét nút F, nút con đầu tiên là 5 -> alpha >=5 mà có beta <=3 -> loại
  6. Tương tự với các nút khác

### 1.4 Giao diện người dùng

* Sử dụng thư viện SFML để triển khai giao diện người dùng.
* Khởi tạo cửa sổ bằng lớp sf:RenderWindow sẵn có.
* Sử dụng lớp sf:Event để xử lý các sự kiện click chuột từ người dùng.
* Khi người dùng chọn một nước đi trên bàn cờ, chúng ta có thể gửi nước đi đó cho engine AI của bạn để tính toán phản hồi. Sau đó, AI sẽ di chuyển quân cờ trên bàn cờ dựa trên nước đi đã được tính toán.
* Sau khi có các nước đi từ người chơi và AI, cập nhật trạng thái của bàn cờ và hiển thị lại bàn cờ trên cửa sổ sử dụng window.display() của SFML.
* Tạo một vòng lặp chính để duy trì game. Trong vòng lặp này có thể kiểm tra các sự kiện như nhấp chuột hoặc bấm phím để xử lý tương tác của người dùng. Đồng thời chúng ta cũng có thể cập nhật và hiển thị lại bàn cờ trong mỗi lần lặp.

## II. Tiến hành cài đặt chương trình

### 2.1 Khai báo thư viện

* SFML: sử dụng để khởi tạo giao diện người dùng.
* Math: sử dụng các hàm tính toán.
* Time: thư viện sử dụng để thao tác với date và time.
* Stack: sử dụng cấu trúc dữ liệu stack.
* Algorithm: sử dụng hàm min, max.

### Ảnh có chứa văn bản, ảnh chụp màn hình, Phông chữ, số Mô tả được tạo tự động2.2 Lớp quản lý trò chơi

* Class GameManager: Lớp này đóng vai trò quản lý trò chơi cờ vua, và chứa các phương thức và thuộc tính liên quan.
* Các thuộc tính:
  + QuanCo f[33]: Mảng lưu trữ thông tin về các quân cờ trong trò chơi.
  + Vector2f positiveMove[32]: Mảng lưu trữ các vị trí di chuyển có thể cho các quân cờ.
  + int positiveCount: Số lượng nước đi có thể thực hiện được.
  + stack<Vector2f> posS: Stack để lưu các vị trí nước đã đi.
  + stack<int> nS: Stack để lưu chỉ mục của các quân cờ đã đi, tương ứng với vị trí trên stack vị trí nước đi.
* Các phương thức:
  + move(int n, Vector2f oldPos, Vector2f newPos): Phương thức để thực hiện nước đi trong trò chơi.
  + Undo(): Phương thức để quay lại nước đi trước đó.
  + Create(): Phương thức để khởi tạo mảng quân cờ và các giá trị cần thiết.
  + Play(): Phương thức để xử lý tất cả các thao tác trong trò chơi.
  + PositiveXe(int n, int x, int y, int grid[9][9]): Phương thức để tìm các nước đi có thể cho quân Xe.
  + PositiveTuong(int n, int x, int y, int grid[9][9]): Phương thức để tìm các nước đi có thể cho quân Tượng.
  + PositiveMa(int n, int x, int y, int grid[9][9]): Phương thức để tìm các nước đi có thể cho quân Mã.
  + PositiveVua(int n, int x, int y, int grid[9][9]): Phương thức để tìm các nước đi có thể cho quân Vua.
  + PositiveTot(int n, int x, int y, int grid[9][9]): Phương thức để tìm các nước đi có thể cho quân Tốt.
  + IncreasePositive(int i, int j): Phương thức để tăng số lượng nước đi có thể ở vị trí (i, j).
  + PositiveMoving(int n): Phương thức để tìm các nước đi có thể tương ứng với chỉ mục n.
  + CostMove(): Phương thức để tính điểm cho một nước đi (hàm đánh giá).
  + Alpha\_Beta(int depth, bool luot, int alpha, int beta): Phương thức để thực hiện thuật toán cắt tỉa Alpha-Beta.
  + Vector2f getNextMove(bool luot): Phương thức để lấy nước đi tiếp theo tốt nhất dựa trên chiến lược.
* Hàm main(): Hàm chính của chương trình:
  + Tạo một đối tượng GameManager
  + Gọi phương thức Play() để bắt đầu trò chơi.

### 2.3 Phần của hàm Create() trong class GameManager

### Hàm này được sử dụng để gán các giá trị cần thiết cho danh sách các quân cờ trong trò chơi.

Ảnh có chứa văn bản, ảnh chụp màn hình, Phông chữ, số

Mô tả được tạo tự động

Hàm Create() được sử dụng để gán giá trị cho danh sách các quân cờ trong mảng f dựa trên các giá trị từ mảng board:

* `positiveCount=0;`: Đặt giá trị ban đầu của `positiveCount` là 0, biến này sẽ được sử dụng để đếm số nước đi có thể của các quân cờ.
* `int k=0;`: Khai báo và khởi tạo biến `k` với giá trị ban đầu là 0, biến này sẽ được sử dụng để duyệt qua các phần tử trong mảng `f`.
* Vòng lặp `for` lồng nhau để duyệt qua mỗi phần tử trong mảng `board` theo từng hàng và cột.
* Biến n: Biến này lưu giá trị của ô hiện tại trên bàn cờ. Nếu giá trị là 0, nghĩa là không có quân cờ tại ô đó và ta tiếp tục với ô tiếp theo.
* Biến x và y: Đây là các biến dùng để tính chỉ số của quân cờ trong mảng f (mảng lưu trữ thông tin về các quân cờ).
* `int n = board[i][j];`: Gán giá trị của phần tử tại vị trí `(i, j)` trong mảng `board` cho biến `n`. Đây là giá trị tượng trưng cho một quân cờ.
* `if (!n) continue;`: Kiểm tra nếu giá trị của `n` bằng 0, tức là không có quân cờ tại vị trí `(i, j)`, thì bỏ qua bước này và tiếp tục với phần tử tiếp theo trong vòng lặp.
* `int x = abs(n)-1;`: Tính toán giá trị `x` dựa trên giá trị tuyệt đối của `n`. Giá trị `x` sẽ được sử dụng để lựa chọn vị trí cắt ảnh từ tệp hình ảnh `figures.png` cho quân cờ tương ứng.
* `int y = n>0?1:0;`: Gán giá trị `y` dựa trên giá trị của `n`. Nếu `n` lớn hơn 0, tức là quân cờ thuộc phe đen, thì `y` được gán giá trị 1, ngược lại `y` được gán giá trị 0. Giá trị `y` sẽ được sử dụng để xác định phần của tệp hình ảnh `figures.png` chứa hình ảnh của quân cờ.
* `f[k].index=n;`: Gán giá trị `n` cho trường `index` của phần tử `f[k]` trong mảng `f`. Điều này lưu trữ giá trị của quân cờ.
* `f[k].s.setTextureRect(IntRect(size\*x,size\*y,size,size));`: Thiết lập vùng chọn cắt ảnh cho quân cờ `f[k]` dựa trên giá trị `x`, `y` và `size`. Điều này giúp hiển thị đúng hình ảnh của quân cờ.
* `f[k].s.setPosition(size\*j+offset.x,size\*i+offset.y);`: Thiết lập vị trí hiển thị của quân cờ `f[k]` trên bàn cờ. Vị trí này được tính dựa trên giá trị `j`, `i`, `size` và `offset`.
* Tiếp theo, các bước tính toán `v` và `g` để gán giá trị cho trường `cost` của quân cờ `f[k]`. Giá trị `v` đại diện cho giá trị của quân cờ và được tính toán dựa trên giá trị `g`. Cuối cùng, giá trị `cost` được tính bằng cách nhân `f[k].index/g` với `v`.
* Tăng giá trị của `k` sau khi khởi tạo một quân cờ để trỏ đến vị trí tiếp theo trong mảng `f`.

Tóm lại, hàm `Create()` sẽ duyệt qua các phần tử trong mảng `board`, khởi tạo giá trị và thuộc tính cho từng quân cờ trong mảng `f` dựa trên giá trị và vị trí của nó trong mảng `board`.

### 2.4 Các nước đi quân cờ

grid[9][9] là ma trận 8x8. Lưu trữ index của các quân cờ, ví dụ tại tọa độ (3,4) không có quân cờ nào thì grid[3][4]=0, có xe trắng thì grid[3][4] = 1, xe đen thì grid[3][4] = -1.-

**a. Quân xe:**

* Lý thuyết:
  + Quân xe đi theo đường thẳng và ngang.
  + Chia ra 4 trường hợp: xe tiến, xe lùi, xe sang phải, xe sang trái.
    - Xét TH xe tiến: quân xe có thể đi được tất cả các ô trước mặt nó trên 1 đường thẳng cho đến khi gặp 1 quân cờ khác. Nếu quân cờ ấy cùng màu vs xe thì quân xe sẽ bị chặn tại ngay trước nó, còn nếu quân cờ khác màu thì xe có thể ăn quân cờ ấy.

A computer screen shot of a math problem

Description automatically generated

* + - i là trục dọc
    - Vòng for: i = x+1 là xét vị trị từ trước con xe đến cuối bàn cờ
    - Nếu 1 vị trí trước nó có quân cờ 🡪 kiểm tra quân cờ có cùng màu không thông qua phép nhân 2 grid, nếu nhỏ hơn 0, tức là 2 quân khác màu và vị trí tại quân cờ khác màu ấy vẫn là vị trí hợp lệ cho xe đi
    - Nếu vị trí trước nó không có quân cờ 🡪 vị trí hợp lệ cho xe đi.
    - Hàm xét vị trí hợp lệ: IncreasePositive(i,y): Hàm nhận vào 2 tham số i,j tương đương tọa độ dọc, ngang.
    - A close up of text

      Description automatically generatedpositiveMove được khai báo trước đó với kích cỡ là 32:

A screenshot of a computer code

Description automatically generatedpositiveCount coi như nước đi hợp lệ, giá trị chạy từ 0 🡪 31.

* i\*size, j\*size là tính giá trị pixel tại vị trí tọa độ i,j . Ý nghía là chuyển đổi từ vị trí tọa độ trên bàn cờ thành vị trí tọa độ trên màn hình.
* Size mình cho là 56
* Vector offset là (28,28)
* Ví dụ gọi hàm increasePositive lại vị trí i=2, j=3, positiveCount đang = 5 thì positiveMove[5]= (2\*56,3\*56) + (28,28) = (… , …) <lười tính>
* Khi đó (…, …) được thêm vào mảng positiveMove và positiveCount được tăng thành 6.

**b. Hàm tìm với tham số n:**

* Ý nghĩa hàm này: Tìm các nước đi hợp lệ, thông qua tham số n. Tham số n là giá trị index của quân cờ, ví dụ: xe trắng có index là 1; mã trắng có index là 2; xe đen có index là -1, mã đen có index là -2.
* Như vậy, hàm này sẽ dựa vào giá trị của index của từng quân cờ để chuyển về giá trị pixel và cuối cùng là chuyển về giá trị tọa độ.
* Sự chuyển đổi thông qua công thức:

f[k].s.setPosition(size\*j+offset.x,size\*i+offset.y); (\*)

A screenshot of a computer code

Description automatically generated

* Đầu tiên, khởi tạo 1 biến có định dạng vector2f, đặt tên biến đó là pos.
* Giá trị của pos được gán bởi f[n].s.getPosition() – offset; theo công thức (\*) ở trên kia thì sau bước này ta thu được (i\*size , j\*size);
* Bước tiếp theo chia kết quả cho size và gán lần lượt vào x,y tức là ta thu được 2 biến int x,y có giá trị là i,j tương ứng với tọa dộ quân cờ.
* Tiếp theo Ảnh có chứa văn bản, ảnh chụp màn hình, Phông chữ, tài liệu

  Description automatically generated
* Phần này là khởi tạo mảng grid[9][9] để lưu trữ tất cả các vị trí của quân cờ.
* Rồi 2 vòng lặp for để gán tất cả các vị trí sẽ có giá trị ban đầu là 0. (mục đích là để những ô không có quân cờ nào sẽ được gán giá trị là 0)
* Tiếp:

Ảnh có chứa văn bản, ảnh chụp màn hình, Phông chữ, tài liệu

Description automatically generated

* Vòng lặp này là duyệt trên tất cả 32 vị trí ô cờ, gắn vị trí của các ô có quân cờ vào biến index.

Ảnh có chứa văn bản, ảnh chụp màn hình, Phông chữ, tài liệu

Description automatically generated

* Khi có giá trị của các index thì sẽ gán với các quân cờ tương ứng như mặc định.
* Tại index = 4 tương đương với quân hậu, quân hậu sẽ tích hợp cách đi của cả xe và tượng.

### 2.5 Khởi tạo hàm Play() và hàm IncreasePositive()

A screenshot of a computer program

Description automatically generated

**a. Hàm Play():**  Hàm điều khiển trò chơi, bao gồm tạo cửa sổ, xử lý sự kiện, đổi lượt chơi giữa người chơi và máy tính, vẽ đồ họa và cập nhật hiển thị trò chơi.

* + `RenderWindow window(VideoMode(504, 504), "The Chess! Alpha Beta Pruning");`: Tạo cửa sổ hiển thị trò chơi với kích thước 504x504 và tiêu đề "The Chess! Alpha Beta Pruning".
  + `Texture t1, t2, t3;`: Khai báo ba biến kiểu Texture để chứa hình ảnh.
  + `t1.loadFromFile("figures.png"); t2.loadFromFile("board1.png"); t3.loadFromFile("no.png");`: Tải hình ảnh từ tệp tin "figures.png", "board1.png", "no.png" vào ba biến tương ứng.
  + `for(int i=0;i<32;i++) f[i].s.setTexture(t1);`: Gán hình ảnh từ biến t1 cho tất cả các Sprite trong mảng f.
  + `Sprite sBoard(t2);`: Khởi tạo Sprite sBoard với hình ảnh từ biến t2.
  + `Sprite sPositive(t3);`: Khởi tạo Sprite sPositive với hình ảnh từ biến t3.
  + `Create();`:Khởi tạo trạng thái ban đầu của trò chơi.
  + `bool LuotChoi=true;`: Khai báo biến boolean LuotChoi và gán giá trị true, đại diện cho lượt chơi của người chơi.
  + `Vector2f oldPos, newPos;`: Khai báo hai biến Vector2f để lưu vị trí click lần 1 và lần 2.
  + `int n=0, click=0, count=0;`: Khai báo các biến số nguyên n, click và count, ban đầu có giá trị 0. Biến n sẽ lưu chỉ số trong mảng f, biến click đếm số lần click chuột và biến count dùng trong việc đếm các nước đi hợp lệ.
  + `Vector2i pos;`: Khai báo biến Vector2i để lưu vị trí chuột khi click.
  + `while (window.isOpen())`: Bắt đầu vòng lặp chính của trò chơi, vòng lặp sẽ tiếp tục chạy miễn là cửa sổ window vẫn đang mở.
  + `Event e;`: Khai báo biến e kiểu Event để bắt sự kiện (sự kiện click chuột, đóng cửa sổ, vv).
  + `while (window.pollEvent(e))`: Vòng lặp này sẽ chạy để xử lý các sự kiện trong cửa sổ window.
  + `if (e.type == Event::Closed) window.close();`: Nếu sự kiện là đóng cửa sổ (sự kiện Event::Closed), thì đóng cửa sổ window.
  + `if (e.type == Event::KeyPressed)`: Kiểm tra nếu sự kiện là nhấn phím.
  + `if (e.key.code == Keyboard::BackSpace) { Undo(); }`: Nếu phím nhấn là Backspace, gọi phương thức Undo() để hoàn tác nước đi trước đó.
  + `if (e.type == Event::MouseButtonPressed)`: Kiểm tra nếu sự kiện là click chuột.
  + `if (e.key.code == Mouse::Left)`: Kiểm tra nếu là click chuột trái.
  + `pos = Mouse::getPosition(window) - Vector2i(offset);`: Lấy vị trí chuột khi click và trừ đi vector offset để tính toán vị trí chính xác trên cửa sổ game.
* A screenshot of a computer code

  Description automatically generated
  + `click++;`: Tăng giá trị biến click lên 1, đếm số lần click chuột.
  + `if (LuotChoi == true)`: Kiểm tra nếu là lượt chơi của người chơi.
  + `if (click == 1)`: Kiểm tra nếu là lần click đầu tiên.
  + `bool isMove = false;`: Khai báo biến boolean isMove và gán giá trị false, đại diện cho việc kiểm tra xem lần click đầu có hợp lệ hay không.
  + `for (int i = 16; i < 32; i++)`: Vòng lặp để kiểm tra tất cả các quân cờ của người chơi.
  + `if (f[i].s.getGlobalBounds().contains(pos.x + offset.x, pos.y + offset.y))`: Kiểm tra xem Sprite của quân cờ f[i] có chứa tọa độ vị trí chuột khi click không.
  + `isMove = true; n = i; f[n].s.setColor(Color::Green); oldPos = f[n].s.getPosition();`: Đánh dấu là click hợp lệ, gán giá trị biến n bằng i để lưu chỉ số của quân cờ, thay đổi màu quân cờ sang màu xanh lá cây và lưu vị trí hiện tại của quân cờ.
  + `if (!isMove) click = 0;`: Nếu không có nước đi hợp lệ, đặt lại giá trị biến click về 0.
  + `if (click == 2)`: Kiểm tra nếu là lần click thứ hai.
  + `f[n].s.setColor(Color::White);`: Đặt lại màu quân cờ về màu trắng.
  + `int x = pos.x / size; int y = pos.y / size;`: Tính toán vị trí x và y của ô cờ mà người chơi click.
  + `newPos = Vector2f(x \* size, y \* size) + offset;`: Tính toán vị trí mới của quân cờ dựa trên vị trí ô cờ được click.
  + `for (int i = 0; i < count; i++)`: Vòng lặp để kiểm tra các nước đi có thể của quân cờ.
  + `if (positiveMove[i] == newPos)`: Kiểm tra nếu nước đi mới là một trong các nước đi có thể.
  + `move(n, oldPos, newPos); LuotChoi = !LuotChoi;`: Di chuyển quân cờ từ vị trí cũ đến vị trí mới, đổi lượt chơi.
  + `count = 0; click = 0;`: Đặt lại giá trị count và click về 0.
* A computer code on a white background

  Description automatically generated
  + `else`: Trường hợp lượt chơi là của máy tính.
  + `newPos = getNextMove(LuotChoi);`: Lấy nước đi tiếp theo của máy tính dựa trên thuật toán AI.
  + `int c = nS.top(); nS.pop(); oldPos = posS.top(); posS.pop();`: Lấy giá trị từ stack nS và posS để di chuyển quân cờ từ vị trí cũ đến vị trí mới.
  + `move(c, oldPos, newPos); LuotChoi = !LuotChoi;`: Di chuyển quân cờ và đổi lượt chơi.
  + `window.draw(sBoard);`: Vẽ hình nền trò chơi.
  + `for (int i = 0; i < count; i++) window.draw(sPositive);`: Vẽ các nước đi có thể.
  + `for (int i = 0; i < 32; i++) window.draw(f[i].s);`: Vẽ tất cả các quân cờ.
  + `window.display();`: Hiển thị cửa sổ window sau khi vẽ.
  + Vòng lặp này sẽ tiếp tục chạy cho đến khi người chơi đóng cửa sổ hoặc trò chơi kết thúc.

**b. Hàm IncreasePositive():** được sử dụng để thêm một nước đi có thể vào danh sách `positiveMove` trong trò chơi cờ.

A black and red text

Description automatically generated

* `positiveMove[positiveCount]`: Truy cập vào phần tử thứ `positiveCount` của mảng `positiveMove`. Đây là vị trí mà nước đi mới sẽ được lưu trữ.
* `Vector2f(i\*size, j\*size) + offset`: Tạo một đối tượng `Vector2f` mới bằng cách nhân tọa độ `i` và `j` với kích thước `size` (giá trị này xác định vị trí ô cờ) và sau đó cộng thêm giá trị của `offset`. Điều này tính toán vị trí thực tế trên cửa sổ của nước đi có thể.
* `positiveCount++`: Tăng giá trị của `positiveCount` lên 1 để chỉ định rằng một nước đi mới đã được thêm vào danh sách `positiveMove`.

### 2.6 Hàm undo() và move()

**a. Hàm move()**

**- Tổng quan hàm Move:** *Hàm này sẽ thực hiện việc di chuyển quân cờ trong trò chơi:*

* *Di chuyển bình thường*
* *Di chuyển có ăn=> ẩn quân bị ăn*
* *Di chuyển phong hậu=> tốt->hậu, tốt bị ẩn*

**void GameManager::move(int n,Vector2f oldPos,Vector2f newPos)**

**{** //*Bước di chuyển thường(không ăn, không phong hậu của quân cờ)*

**posS.push(oldPos);** //*lưu vị trí cũ*

**posS.push(newPos);**//*lưu vị trí mới*

**nS.push(n);**//*lưu chỉ số*

//*Bước di chuyển khi phong hậu*

* *Kiểm tra xem có phong hậu hay không*
* *Phương pháp: (Như hình minh họa ở trên) Nếu quân cờ là quân tốt có :*
  + - *index=6 & hàng 0 (hàng đầu tiên của bàn cờ đối diện)*
    - *index=-6 và hàng 7 (hàng cuối của bàn cờ đối diện)* 
      * *quân tốt sẽ được nâng cấp thành hậu.*

*//xác định hàng của ô cờ mới để kiểm tra xem phong hậu không*

* *Công thức* ***(newPos-offset).y/size*** *tính toán khoảng cách theo trục y giữa vị trí mới newPos và vị trị gốc của bàn cờ rồi chia cho kích thước của ô cờ, ta sẽ được dòng của ô cờ mới.*

**int y=int((newPos-offset).y/size);**

//*Phong hậu cho tốt*

**if(y==0 && f[n].index==6){**

**nS.push(100);**//*đánh dấu một hành động cụ thể trong quá trình di chuyển của quân cờ để khi ta hoàn tác (gọi hàm Undo() ) giả trị 100 trong ngăn xếp sẽ cho chương trình biết rằng hành động phong hậu đã được thực hiện trước đó và cần bị xóa đi khi hoàn tác (xóa hậu). Giá trị 100 là 1 giá trị đặc biệt không liên quan đến quân cờ và vị trí cụ thể nên được dùng để đánh dấu*

*//cập nhập chỉ số, chi phí và hình ảnh của quân cờ chuyển từ tốt sang hậu*

**f[n].index=4;**

**f[n].cost=90;**

**f[n].s.setTextureRect(IntRect( 3\*size,size,size,size));**

**}**

**if(y==7 && f[n].index==-6){**

**nS.push(-100);**

**f[n].index=-4;**

**f[n].cost=-90;**

**f[n].s.setTextureRect(IntRect( 3\*size,0,size,size));**

**}**

//*kiểm tra và ẩn quân bị ăn*

**for(int i=0;i<32;i++){**

**if (f[i].s.getPosition()==newPos) {***//kiểm tra xem vị trí mới của quân hiện tại, có trùng với vị trí của quân cờ nào khác không, nếu trùng tức là quân cờ hiện tại di chuyển và chiếm vị trí của quân cờ i ( quân cờ i bị ăn)*

**f[i].s.setPosition(-100,-100);**//*ẩn quân cờ i khỏi màn hình hiển thị mà không xóa nó khỏi danh sách do phạm vi hiển thị màn hình trò chơi (0,0)->(27,27)*

*//Lưu thông tin quân cờ*

**posS.push(newPos);//***vị trí mới của quân cờ*

**posS.push(Vector2f(-100,-100));//***vị trí của quân cờ bị ẩn*

**nS.push(i);**

**break;**

**}**

**}**

**f[n].s.setPosition(newPos);**// *sau khi kiểm tra các điểu kiện và xử lý di chuyển thì quân cờ có chỉ số n sẽ được di chuyển đến vị trí mới newPos trên bàn cờ*

**}**

**b. Hàm undo():** *Hoàn tác hành động di chuyển của quân cờ trong từng trường hợp di chuyển trong hàm move*

**void GameManager::Undo()**

**{**

**int n= nS.top();//***gán n chỉ số của quân cờ đã được di chuyển*

**nS.pop();//*loại bỏ phần tử đỉnh của ngăn xếp sau khi đã gán vào biến n***

**Vector2f p=posS.top();//***gán cho biến p vị trị trước khi được undo của quân cờ, dùng để**kiểm tra xem co = (-100,-100) => đấy là con bị ăn*

**posS.pop();//***xóa*

**Vector2f q=posS.top();//***gán cho biến q vị trí mới(vị trí sẽ được trả về) của quân cờ*

**posS.pop();**

**//***Cách undo với th phong hậu*

**if(n==100) {//***kiểm tra xem nếu n =100, đánh dấu là quân cờ đã phong hậu trong quá trình trước đó*

**n=nS.top();//***lấy giá trị đầu của ngăn xếp nS, lúc này chính là chỉ số của quân cờ trước khi move, gán lại cho biến n*

**nS.pop();**

**//***Gán lại index, cost, hình ảnh của quân cờ thành các giá trị biểu thị quân tốt*

**f[n].index=6;**

**f[n].cost=10;**

**f[n].s.setTextureRect(IntRect( 5\*size,size,size,size));**

**}**

**//***tương tự với bên ngược lại*

**if(n==-100){**

**n=nS.top();**

**nS.pop();**

**f[n].index=-6;**

**f[n].cost=-10;**

**f[n].s.setTextureRect(IntRect( 5\*size,0,size,size));**

**}**

**f[n].s.setPosition(q);//** *trả về vị trí trước khi move của quân cờ trong trường hợp không phong hậu*

**if(p==Vector2f(-100,-100)) //***trong quá trình di chuyển quân cờ, khi quân cờ bị ăn, vị trí mới của nó sẽ được đặt là vector2f(-100,-100), tức là nó bị xóa khỏi bàn cờ. Điều kiện trên đúng tức là ta đang xử lý hoàn tác cho một quân cờ đã bị ăn*

**Undo();//** *sử dụng đệ quy để hoàn tác 1 cách chính xác và đúng thứ tự quân bị ăn*

**}**

### 2.7. Hàm đánh giá, giải thuật Minimax và cắt tỉa Alpha-Beta, hàm chọn nước đi

**a. Hàm đánh giá**

Đặt giá trị cho mỗi quân cờ, quân nào bị ăn mất thì không tính điểm quân đó

int GameManager::CostMove()// don gian con nao bi chet thi khong tinh diem cua con day

{

int s=0;

for(int i=0;i<32;i++)

{

if(f[i].s.getPosition()==Vector2f(-100,-100)) continue;//neu no da bi out-> ko tinh diem

s+=f[i].cost;

}

return s;

}

* “int GameManager::CostMove()”: Đây là một phương thức trong lớp GameManager, có kiểu trả về là int.
* “int s = 0;” : Khởi tạo biến ‘s’ với giá trị ban đầu bằng 0, biến này dùng để tính điểm các quân cờ có trên bàn.
* “for(int i = 0; i < 32; i++)”: Vòng lặp từ 0 đến 31 để duyệt qua danh sách các phần tử.
* “if(f[i].s.getPosition()==Vector2f(-100,-100)) continue;”: Kiểm tra nếu vị trí của phần tử thứ i (-100, -100), tức nghĩa là đã bị ăn mất thì sẽ bỏ qua nó.
* “s += f[i].cost;”: Tăng giá trị biến s bằng với số điểm của quân cờ thứ i.

**b. Cắt tỉa Alpha – Beta**

Trả lại điểm cho gốc muốn xét

int GameManager::Alpha\_Beta(int depth,bool luot,int alpha,int beta)

{

if(depth==0){

return CostMove();

}

Vector2f positiveMovetemp[32];//luu lai vi tri cac nuoc co the di

if(luot==true){

int bestMove=-10000;//gia cua bestMove ban dau

for(int j=16;j<32;j++)//cac quan cua nguoi choi

{

if(f[j].s.getPosition()==Vector2f(-100,-100)) continue;

PositiveMoving(j);

int coun=positiveCount;//ta khong the dung PositiveCount vi no thay doi lien tuc khi ta de quy

positiveCount=0;

for (int i = 0; i < coun; i++) positiveMovetemp[i]=positiveMove[i];

for(int i=0;i<coun;i++)

{

move(j,f[j].s.getPosition(),positiveMovetemp[i]);

bestMove=max(bestMove,Alpha\_Beta(depth1,!luot,alpha,beta));

//undo

Undo();

alpha=max(alpha,bestMove);

if(beta<=alpha) return bestMove;

}

}

return bestMove;

}

else {

int bestMove=10000;//gia cua bestMove ban dau

for(int j=0;j<16;j++)//quan cua may

{

if(f[j].s.getPosition()==Vector2f(-100,-100)) continue;

PositiveMoving(j);

int coun=positiveCount;//ta khong the dung PositiveCount vi no thay doi lien tuc khi ta de quy

positiveCount=0;

for (int i = 0; i < coun; i++) positiveMovetemp[i]=positiveMove[i];

for(int i=0;i<coun;i++)

{

move(j,f[j].s.getPosition(),positiveMovetemp[i]);

bestMove=min(bestMove,Alpha\_Beta(depth-1,!luot,alpha,beta));

//undo

Undo();

beta=min(beta,bestMove);

if(beta<=alpha) return bestMove;

}

}

return bestMove;

}

}

* `int GameManager::Alpha\_Beta(int depth, bool luot, int alpha, int beta)`: Đây là khai báo của phương thức `Alpha\_Beta`. Nó nhận vào các tham số: `depth` (độ sâu của cây tìm kiếm), `luot` (biến bool cho biết đến lượt nào), `alpha` và `beta` (giá trị tốt nhất cho lượt hiện tại và lượt tiếp theo).
* `if (depth == 0) { return CostMove(); }`: Đây là điều kiện dừng của đệ quy. Nếu `depth` bằng 0, nghĩa là đã đạt đến độ sâu tối đa được chỉ định, phương thức trả về giá trị được tính bằng hàm `CostMove()`.
* `Vector2f positiveMovetemp[32];`: Khai báo một mảng chứa tối đa 32 phần tử kiểu `Vector2f`. Mảng này được sử dụng để lưu lại vị trí của các nước có thể di chuyển.
* `if (luot == true) { ... }`: Nếu là lượt của người chơi (`luot` bằng `true`), thực hiện các lệnh bên trong khối này.
* `int bestMove = -10000;`: Khởi tạo giá trị ban đầu cho `bestMove` với một giá trị rất nhỏ. `bestMove` sẽ lưu trữ giá trị tốt nhất được tìm thấy cho lượt hiện tại.
* `for (int j = 16; j < 32; j++) { ... }`: Vòng lặp để duyệt qua các quân cờ của người chơi (từ 16 đến 31). Quân cờ của người chơi được đại diện bởi giá trị `j`.
* `if (f[j].s.getPosition() == Vector2f(-100, -100)) continue;`: Nếu vị trí của quân cờ `f[j]` là `Vector2f(-100, -100)`, tức là quân cờ không còn tồn tại trên bàn cờ, bỏ qua và chuyển sang quân cờ tiếp theo.
* `PositiveMoving(j);`: Gọi phương thức `PositiveMoving` với tham số là giá trị `j`. Phương thức này tính toán các nước đi hợp lệ cho quân cờ hiện tại và lưu kết quả vào mảng `positiveMove`.
* `int coun = positiveCount;`: Sao chép giá trị của biến `positiveCount` vào biến `coun`. `positiveCount` đếm số lượng nước đi hợp lệ được tính toán trong `PositiveMoving`.
* `positiveCount = 0;`: Đặt giá trị của `positiveCount` về 0 để sử dụng cho lần tính toán tiếp theo.
* `for (int i = 0; i < coun; i++) positiveMovetemp[i] = positiveMove[i];`: Sao chép các giá trị trong mảng `positiveMove` vào mảng `positiveMovetemp`.
* `for (int i = 0; i < coun; i++) { ... }`: Vòng lặp để duyệt qua các nước đi hợp lệ cho quân cờ hiện tại.
* `move(j, f[j].s.getPosition(), positiveMovetemp[i]);`: Di chuyển quân cờ `j` đến vị trí `positiveMovetemp[i]` bằng cách gọi phương thức `move`.
* `bestMove = max(bestMove, Alpha\_Beta(depth1, !luot, alpha, beta));`: Gọi đệ quy phương thức `Alpha\_Beta` để tìm giá trị tốt nhất cho lượt tiếp theo. Lưu giá trị tốt nhất vào `bestMove` bằng cách so sánh với giá trị hiện tại của `bestMove` và giá trị trả về từ đệ quy.
* `Undo();`: Hoàn tác nước đi trước đó bằng cách gọi phương thức `Undo`.
* `alpha = max(alpha, bestMove);`: Cập nhật giá trị của `alpha` thành giá trị lớn nhất giữa `alpha` và `bestMove`.
* `if (beta <= alpha) return bestMove;`: Kiểm tra điều kiện chặn cắt Alpha-Beta. Nếu `beta` nhỏ hơn hoặc bằng `alpha`, thoát khỏi vòng lặp và trả về giá trị `bestMove`.
* `return bestMove;`: Trả về giá trị `bestMove` là kết quả tốt nhất tìm thấy cho lượt hiện tại.
* `else { ... }`: Nếu không phải lượt của người chơi (`luot` bằng `false`), thực hiện các lệnh bên trong khối này tương tự như trên.
* `int bestMove = 10000;`: Khởi tạo giá trị ban đầu cho `bestMove` với một giá trị rất lớn. `bestMove` sẽ lưu trữ giá trị tốt nhất được tìm thấy cho lượt hiện tại.
* `for (int j = 0; j < 16; j++) { ... }`: Vòng lặp để duyệt qua các quân cờ của máy (từ 0 đến 15). Quân cờ của máy được đại diện bởi giá trị `j`.
* ‘if (f[j].s.getPosition() == Vector2f(-100, -100)) continue;`: Nếu vị trí của quân cờ `f[j]` là `Vector2f(-100, -100)`, tức là quân cờ không còn tồn tại trên bàn cờ, bỏ qua và chuyển sang quân cờ tiếp theo.
* `PositiveMoving(j);`: Gọi phương thức `PositiveMoving` với tham số là giá trị `j`. Phương thức này tính toán các nước đi hợp lệ cho quân cờ hiện tại và lưu kết quả vào mảng `positiveMove`.
* `int coun = positiveCount;`: Sao chép giá trị của biến `positiveCount` vào biến `coun`. `positiveCount` đếm số lượng nước đi hợp lệ được tính toán trong `PositiveMoving`.
* `positiveCount = 0;`: Đặt giá trị của `positiveCount` về 0 để sử dụng cho lần tính toán tiếp theo.
* `for (int i = 0; i < coun; i++) positiveMovetemp[i] = positiveMove[i];`: Sao chép các giá trị trong mảng `positiveMove` vào mảng `positiveMovetemp`.
* `for (int i = 0; i < coun; i++) { ... }`: Vòng lặp để duyệt qua các nước đi hợp lệ cho quân cờ hiện tại.
* `move(j, f[j].s.getPosition(), positiveMovetemp[i]);`: Di chuyển quân cờ `j` đến vị trí `positiveMovetemp[i]` bằng cách gọi phương thức `move`.
* `bestMove = min(bestMove, Alpha\_Beta(depth - 1, !luot, alpha, beta));`: Gọi đệ quy phương thức `Alpha\_Beta` để tìm giá trị tốt nhất cho lượt tiếp theo. Lưu giá trị tốt nhất vào `bestMove` bằng cách so sánh với giá trị hiện tại của `bestMove` và giá trị trả về từ đệ quy.
* `Undo();`: Hoàn tác nước đi trước đó bằng cách gọi phương thức `Undo`.
* `beta = min(beta, bestMove);`: Cập nhật giá trị của `beta` thành giá trị nhỏ nhất giữa `beta` và `bestMove`.
* `if (beta <= alpha) return bestMove;`: Kiểm tra điều kiện chặn cắt Alpha-Beta. Nếu `beta` nhỏ hơn hoặc bằng `alpha`, thoát khỏi vòng lặp và trả về giá trị `bestMove`.
* `return bestMove;`: Trả về giá trị `bestMove` là kết quả tốt nhất tìm thấy cho lượt hiện tại.

**c. Hàm chọn nước đi**

Dùng Alpha – Beta chọn ra nước đi tốt nhất

Vector2f GameManager::getNextMove(bool luot)

{

Vector2f oldPos,newPos,oldPostemp,newPostemp;// ta can tim vi tri co minimax nho nhat de ung voi may( quan den)

int minimaxtemp=10000,minimax=10000;

int count1,n;

Vector2f positiveMovetemp[32];

for(int i=0;i<16;i++)

{

if(f[i].s.getPosition()==Vector2f(-100,-100)) continue;

//////

PositiveMoving(i);

count1=positiveCount;//khong the dung PositiveCount vi no thay doi lien tuc khi ta de quy

positiveCount=0;

///set///

for (int k = 0; k < count1; k++) positiveMovetemp[k]=positiveMove[k];

//set oldPos va newPos tam thoi

oldPostemp=f[i].s.getPosition();

//newPostemp=positiveMove[0];

for(int j=0;j<count1;j++)

{

move(i,oldPostemp,positiveMovetemp[j]);

int alpha=-9999,beta=9999;

int temp=Alpha\_Beta(3,!luot,alpha,beta);

if(minimaxtemp>temp){

newPostemp=positiveMovetemp[j];

minimaxtemp=temp;

}

Undo();

}

if(minimax>minimaxtemp){

minimax=minimaxtemp;

oldPos=oldPostemp;

newPos=newPostemp;

n=i;

}

}

//lay cac thong tin nuoc di

posS.push(oldPos);//luu tam o trong stack ty ra ngoai xoa di

nS.push(n);

return newPos;

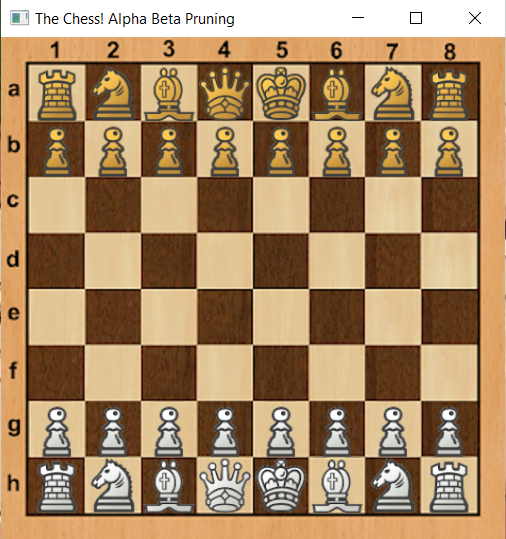
}

* `Vector2f GameManager::getNextMove(bool luot)`: Đây là khai báo hàm `getNextMove` của lớp `GameManager`. Hàm này nhận một tham số `luot` kiểu boolean và trả về một đối tượng `Vector2f`.
* `Vector2f oldPos, newPos, oldPostemp, newPostemp;`: Khai báo bốn biến `oldPos`, `newPos`, `oldPostemp`, `newPostemp` kiểu `Vector2f`.
* `int minimaxtemp = 10000, minimax = 10000;`: Khai báo hai biến `minimaxtemp` và `minimax` kiểu `int` và gán giá trị khởi tạo là 10000.
* `int count1, n;`: Khai báo hai biến `count1` và `n` kiểu `int`.
* `Vector2f positiveMovetemp[32];`: Khai báo một mảng `positiveMovetemp` kiểu `Vector2f` với kích thước 32.
* `for(int i=0;i<16;i++)`: Bắt đầu vòng lặp for với biến đếm `i` được khởi tạo là 0 và điều kiện dừng là `i<16`.
* `if(f[i].s.getPosition()==Vector2f(-100,-100)) continue;`: Kiểm tra nếu vị trí của `f[i].s` bằng `Vector2f(-100, -100)` thì tiếp tục với lần lặp tiếp theo của vòng lặp.
* `PositiveMoving(i);`: Gọi hàm `PositiveMoving` với tham số `i`.
* `count1 = positiveCount;`: Gán giá trị của biến `positiveCount` cho `count1`.
* `for (int k = 0; k < count1; k++) positiveMovetemp[k] = positiveMove[k];`: Sao chép các phần tử từ mảng `positiveMove` vào mảng `positiveMovetemp`.
* `oldPostemp = f[i].s.getPosition();`: Gán giá trị vị trí của `f[i].s` cho `oldPostemp`.
* `for(int j=0;j<count1;j++)`: Bắt đầu vòng lặp for với biến đếm `j` được khởi tạo là 0 và điều kiện dừng là `j<count1`.
* `move(i, oldPostemp, positiveMovetemp[j]);`: Gọi hàm `move` với các tham số `i`, `oldPostemp`, và `positiveMovetemp[j]`.
* `int alpha = -9999, beta = 9999;`: Khai báo hai biến `alpha` và `beta` kiểu `int` và gán giá trị khởi tạo là -9999 và 9999 tương ứng.
* `int temp = Alpha\_Beta(3, !luot, alpha, beta);`: Gọi hàm `Alpha\_Beta` với các tham số `3`, `!luot`, `alpha`, và `beta`, và gán giá trị trả về cho biến `temp`.
* `if(minimaxtemp > temp)`: Kiểm tra nếu `minimaxtemp` lớn hơn `temp`.
* `newPostemp = positiveMovetemp[j];`: Gán giá trị của `positiveMovetemp[j]` cho `newPostemp`.
* `minimaxtemp = temp;`: Gán giá trị của `temp` cho `minimaxtemp`.
* `Undo();`: Gọi hàm `Undo()`.
* `if(minimax > minimaxtemp)`: Kiểm tra nếu `minimax` lớn hơn `minimaxtemp`.
* `minimax = minimaxtemp;`: Gán giá trị của `minimaxtemp` cho `minimax`.
* `oldPos = oldPostemp;`: Gán giá trị của `oldPostemp` cho `oldPos`.
* `newPos = newPostemp;`: Gán giá trị của `newPostemp` cho `newPos`.
* `n = i;`: Gán giá trị của `i` cho `n`.
* `posS.push(oldPos);`: Đưa `oldPos` vào ngăn xếp `posS`.
* `nS.push(n);`: Đưa `n` vào ngăn xếp `nS`.
* `return newPos;`: Trả về `newPos` là kết quả của hàm `getNextMove`.

# C. Kiểm tra

## I. Giao diện, sự kiện click chuột

### 1.1Giao diện

****

### 1.2 Sự kiện click chuột, nút undo

* Link video: <https://drive.google.com/file/d/1cRWDxcQRGZi2EHRTgb4l9NFJKVaehpBU/view?usp=sharing>

<https://drive.google.com/file/d/1d8l0tABbqdXIom86JZArrtrqUmbHETJy/view?usp=sharing>

* Khi click các quân cờ đều hiển thị nước đi.
* Nút undo hoạt động đúng với yêu cầu.
* Các nút cửa sổ hoạt động đúng với yêu cầu.

## II. Tính hợp lệ các nước đi

* Link video: <https://drive.google.com/file/d/1DIiPdV_0RpDjbWPvUDpRRel1W0Rd5MFT/view?usp=sharing>
* Các quân cờ hiển thị nước đi hợp lệ.
* Nếu có quân cùng màu chắn ở trước, quân cờ không thể di chuyển đến vị trí đó.
* Nếu có quân khác màu, quân cờ có thể ăn quân đó.

## III: Tốc độ phản hồi và độ ‘thông minh’ của mô hình

* Link video: <https://drive.google.com/file/d/1OHnAs8noSc3LnMCCTJwSpNUz79L2ypVB/view?usp=sharing>
* Tốc độ phản hồi của mô hình gần như là lập tức sau khi người chơi di chuyển quân cờ, chỉ có những nước đi khó hoặc đột biến tới từ người chơi thì máy sẽ mất thời gian hơn.
* Vì bị giới hạn về độ sâu của thuật toán, những nước đi đầu tiên của mô hình gần như không có nhiều khác biệt, thậm chí là chỉ di chuyển quân qua lại.
* Ở những nước đi sau và tàn cờ, vì hàm giá trị của thuật toán là tổng số giá trị của các quân cờ, mô hình có xu hướng di chuyển các quân cờ liên kết với nhau và đánh từ từ, không có những quyết định trade-off hay nước đi mang tính mạo hiểm cao.
* Dù có những hạn chế nhất định nhưng mô hình vẫn thể hiện tốt khả năng tính toán và di chuyển quân cờ của mình ở mức độ tạm chấp nhận được.
* Để có thể cải thiện độ ‘thông minh’ của mô hình, ta có thể thêm bảng nước đi khai cuộc có sẵn cho mô hình, sử dụng các mô hình học máy và học sâu, học tang cường,…

# D. Kết luận

* Chess AI engine đã thực hiện được các nước đi cơ bản của trò chơi một cách chính xác.
* Tuy nhiên, phần mềm vẫn còn những hạn chế như chưa có chức năng nhập thành, các mức độ khó khác nhau, chưa hiển thị quân bị ăn, chưa hiển thị thời gian, …
* Mong nhận được sự nhận xét và chỉ dẫn từ thầy về sản phẩm của chúng em để chúng em có thể hoàn thiện sản phậm tốt nhất trong tương lai