ĐẠI HỌC QUỐC GIA TP-HỒ CHÍ MINH
ĐẠI HỌC CÔNG NGHỆ THÔNG TIN
KHOA HỌC MÁY TÍNH

BÁO CÁO
MÔN TRÍ TUỆ NHÂN TẠO
ĐỀ TÀI: XÂY DỰNG CHƯƠNG TRÌNH CHƠI CỜ TƯỚNG

Giảng viên hướng dẫn: Ths Phạm Nguyễn Trường An

Thành viên nhóm: 17520144 Trần Kim Sen
17520180 Lê Thủy Triều
17520943 Trần Nguyễn Hồng Quân
17520964 Nguyễn Đình Quyết
17521180 Đặng Xuân Trường

Tp.Hồ Chí Minh, Chủ Nhật, 14 Tháng Bảy 2019

LỜI MỞ ĐẦU

Xin cảm ơn khoa Khoa học Máy tính và Th.s Phạm Nguyễn Trường An đã truyền tải những kiến thức nền tảng về Trí tuệ nhân tạo cũng như sự hướng dẫn tận tình của thầy, kết hợp với việc tìm tòi, học hỏi, nghiên cứu các kiến thức mới, nhóm đã hoàn thành "Game Cờ Tướng". Trong quá trình thực hiện, những sai sót là không thể tránh khỏi. Chính vì vậy, nhóm mong nhận được những ý kiến góp ý từ giảng viên để chương trình được hoàn thiện hơn.

Cờ Tướng là một trong những loại cờ phổ biến và lâu đời nhất thế giới. Lập trình Cờ Tướng rất quan trọng trong lĩnh vực Trí tuệ nhân tạo và hiện nay, sức mạnh của chương trình Cờ Tướng có thể sánh ngang con người. Vì vậy Cờ Tướng là một ví dụ điển hình để tìm hiểu về Trí tuệ nhân tạo. Trong báo cáo này, nhóm xin được giới thiệu một số kĩ thuật để phát triển một "Game Cờ Tướng".

MỤC LỤC

CHƯƠNG 1: TỔNG QUAN VỀ ĐỒ ÁN

1.1. Sơ lược về cờ tướng

1.1.1. Giới thiệu[1]

Cờ tướng là một trò chơi trí tuệ dành cho hai người. Đây là loại cờ phổ biến nhất tại Trung Quốc và Việt Nam, và nằm trong cùng một thể loại cờ với cờ vua, shogi, janggi.

Ván cờ được tiến hành giữa hai người, một người cầm quân Trắng (hay Đỏ), một người cầm quân Đen (hay Xanh). Mục đích của mỗi người chơi là tìm mọi cách đi quân trên bàn cờ theo đúng luật để bắt Tướng của đối phương hoặc làm đối phương hết nước đi hợp lệ.

1.1.2. Lịch sử

“Đây loại cờ có từ khoảng thế kỷ VII. Cờ tướng được bắt nguồn từ Saturanga, một loại cờ cổ được phát minh ở Ấn Độ từ thế kỷ V đến thế kỷ VI (trước cờ tướng khoảng 200 năm). Chính Saturanga được phát minh từ Ấn Độ, sau đó đi về phía tây, trở thành cờ vua và đi về phía Đông trở thành cờ tướng. Người Trung Quốc cũng đã thừa nhận điều này.

Cờ tướng cổ đại không có quân Pháo. Các nhà nghiên cứu đều thống nhất là quân Pháo được bổ sung từ thời Bắc Tống (sau năm 960), là quân cờ ra đời muộn nhất trong bàn cờ tướng, bởi cho tới thời đó, con người mới tìm ra vũ khí pháo để sử dụng trong chiến tranh.

Tuy nhiên, người Trung Hoa đã cải tiến bàn cờ Saturanga như sau:

Họ không dùng "ô", không dùng hai màu để phân biệt ô, mà họ chuyển sang dùng "đường" để đặt quân và đi quân. Chỉ với động tác này, họ đã tăng thêm số điểm đi quân từ 64 của Saturanga lên 81.

Đã là hai quốc gia đối kháng thì phải có biên giới rõ ràng, từ đó, họ đặt ra "hà", tức là sông. Khi "hà" xuất hiện trên bàn cờ, 18 điểm đặt quân nữa được tăng thêm. Như vậy,

bàn cờ tướng bây giờ đã là 90 điểm so với 64, đó là một sự mở rộng đáng kể. Tuy nhiên, diện tích chung của bàn cờ hầu như không tăng mấy (chỉ tăng thêm 8 ô) so với số điểm tăng lên tới một phần ba.

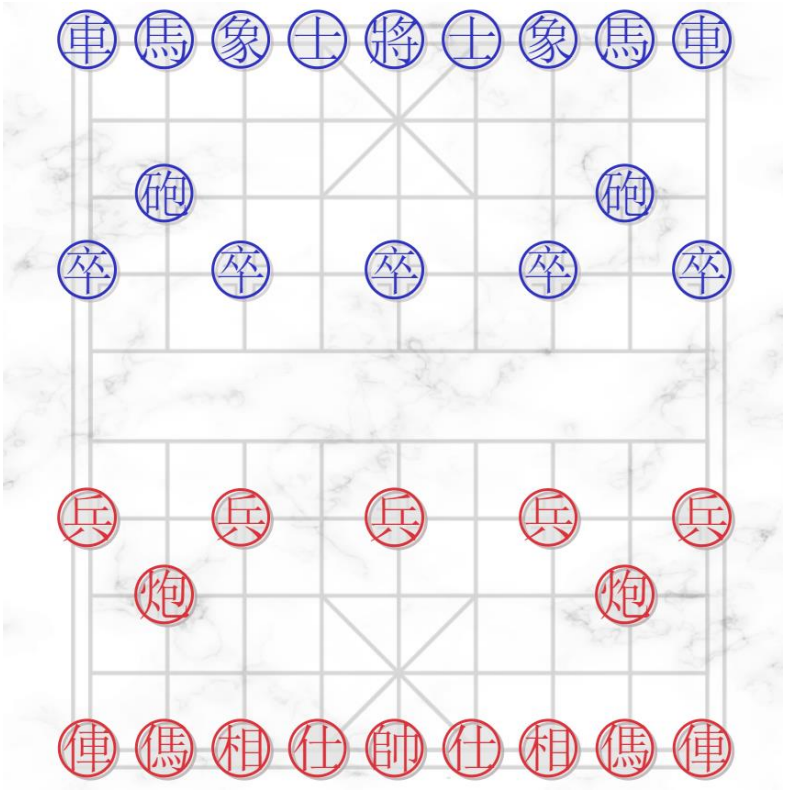
Đã là quốc gia thì phải có cung cấm (宮) và không thể đi khắp bàn cờ như kiểu trò chơi Saturanga được. Thế là "Cửu cung" đã được tạo ra. Điều này thể hiện tư duy phương Đông hết sức rõ ràng.

Bàn cờ Saturanga có hình dáng quân cờ là những hình khối, nhưng cờ Tướng thì quân nào trông cũng giống quân nào, chỉ có mỗi tên là khác nhau, lại được viết bằng chữ Hán. Đây có thể là lý do khiến cờ tướng không được phổ biến bằng cờ vua, chỉ cần liếc qua là có thể nhận ra đâu là Vua, đâu là Hoàng hậu, kỵ sĩ, v.v. Tuy nhiên, đối với người Trung Hoa thì việc thuộc mặt cờ này là không có vấn đề gì khó khăn. Có lẽ việc cải tiến này cũng một phần là do điều kiện kinh tế bấy giờ chưa sản xuất được bộ cờ có hình khối phức tạp như cờ vua. Cờ tướng không phải là một trò chơi sang trọng, muốn tạo ra một bàn cờ tướng cực kỳ đơn giản, chỉ cần lấy que vạch xuống nền đất cũng xong, còn cờ vua thì mất công hơn nhiều khi phải tạo ra các ô đen/trắng xen kẽ nhau.

Gần đây ngày càng có nhiều ý kiến đề nghị cải cách hình dáng các quân cờ tướng và trên thực tế người ta đã đưa những phác thảo của những bộ quân mới bằng hình tượng thay cho chữ viết, nhất là khi cờ tướng được chơi ở những nước không sử dụng tiếng Trung.

Với sự thay đổi bố cục bàn cờ, người Trung Hoa đã phải có những điều chỉnh để lấy lại sự cân bằng cho bàn cờ. Đó chính là những ngoại lệ mà người chơi phải tự nhớ. (Xem thêm phần Mã, Tướng).” [1]

1.1.3. Bàn cờ



1.1.4. Quân cờ và cách di chuyển

Quân	Số lượng	Cách di chuyển quân
Tướng	1	Đi ngang hoặc dọc từng bước 1 trong phạm vi cung tướng.
Sĩ	2	Đi chéo từng bước một trong Cửu cung.
Tượng	2	Đi theo đường chéo hình vuông 2x2. Nếu có quân ở giữa vị trí hiện tại và vị trí tới thì Tượng bị cản. Tượng không được qua sông.
Xe	2	Đi và ăn theo đường dọc hoặc ngang khắp bàn cờ miễn không bị cản.
Pháo	2	Đi giống quân xe nhưng nếu ăn quân phải có quân đứng làm "ngòi"
Mã	2	Đi theo hình chữ nhật 2x1. Nếu đường thẳng đi có quân thì Mã bị cản.
Tốt	5	Đi thẳng và ăn thẳng theo chiều dọc khi ở bên phần đất của bên mình. Khi Tốt qua được sông, chúng có thể đi và ăn theo chiều ngang

1.2. Giải vô địch cờ tướng thế giới World Xiangqi Championship [2][3]

1.2.1. Tên gọi và lịch sử

World Xiangqi Championship là giải đấu cờ tướng do World Xiangqi Federation (WXF) đứng ra tổ chức. Giải đấu được tổ chức lần đầu vào 1990 tại Singapore và được tổ chức 2 năm 1 lần từ năm 1991. Tính đến năm 2017, giải đấu đã được tổ chức 15 lần.

Tuy giải đấu được tổ chức từ năm 1990 tuy nhiên tới lần tổ chức thứ 3 tại Bắc Kinh, Trung Quốc WXF mới được thành lập (06.04.1993).

Khi tham gia giải đấu, mỗi đoàn tham dự được cử đi 3 kỳ thủ (2 nam, 1 nữ) tranh giải chính là ba giải: cá nhân nam, cá nhân nữ, đồng đội nam. Ngoài ra các đoàn có thể cử thêm kỳ thủ tranh giải dành cho kỳ thủ không phải gốc Hoa và gốc Việt (trước năm 1999, giải này dành cho kỳ thủ không phải gốc Hoa, từ năm 1999 thay đổi thành không phải gốc Hoa và gốc Việt). Nước chủ nhà cũng chỉ được cử một đoàn. Có một số trường hợp đặc biệt là Malaysia và Mỹ, mỗi quốc gia này được cử hai đoàn Đông và Tây.

1.2.2. Địa điểm tổ chức và người vô địch các lần tổ chức

	Year	Location	Men	Women	Non − Chinese
1	1990	Singapore	Lü Qin	Teo Sim Hua	Winston Williams
2	1991	Kunming	Zhao Guorong	Hu Ming	Winston Williams
3	1993	Beijing	Xu Tianhong	Hu Ming	Mai Thanh Minh
4	1995	Singapore	Lü Qin	Huang Yuying	Vo Van Hoang Tung
5	1997	Hong Kong	Lü Qin	Lin Ye	Mai Thanh Minh
					Non − Chinese/Non − Vietnamese
6	1999	Shanghai	Xu Yinchuan	Jin Haiying	Shoshi Kazuharu
7	2001	Macau	Lü Qin	Wang Linna	Kon Island
8	2003	Hong Kong	Xu Yinchuan	Guo Liping	Shoshi Kazuharu
9	2005	Paris	Lü Qin	Guo Liping	Kon Island
10	2007	Macau	Xu Yinchuan	Wu Xia	Shoshi Kazuharu
11	2009	Xintai	Zhao Xinxin	You Yingqin	Iwan Setiawan
12	2011	Jakarta	Jiang Chuan	Tang Dan	Kon Island
13	2013	Huizhou	Wang Tianyi	Tang Dan	Krishna Sankirta
14	2015	Munich	Zheng Weitong	Wang Linna	Shoshi Kazuharu
15	2017	Manila	Wang Tianyi	Tang Dan	Joep Nabuurs
16	2019	Canada			

1.2.3. Thể lệ thi đấu năm gần nhất (2017) [4]

1.2.3.1. Đối tượng tham gia

Mỗi đơn vị tham gia thường bao gồm 1 đội trưởng, 2 tuyển thủ nam, 1 tuyển thủ nữ. Tuy nhiên điều này không áp dụng cho tuyển thủ quốc tịch khác Trung Quốc và Việt Nam nên số lượng tham gia không giới hạn.

1.2.3.2. Các hạng mục của giải đấu

- Đồng đội nam
- Cá nhân nam
- Cá nhân nam (dành cho tuyển thủ không phải người Trung Quốc và Việt Nam)

- Cá nhân nữ
- Cá nhân nữ (dành cho tuyển thủ không phải người Trung Quốc và Việt Nam)

1.2.3.3. Cách thức thi đấu

Luật chơi cờ của giải đấu do WXF thông qua tương tự với luật thi đấu của Asian Xiangqi Federation [5].

Với hạng mục đấu cá nhân nam, các tuyển thủ sẽ đấu trong vòng 9 round sử dụng hệ thống thi đấu Swiss − system [6] với 1 game cho mỗi cặp đấu.

Với hạng mục đấu cá nhân nữ, hệ thống thi đấu và số round cũng như số game của mỗi cặp sẽ được quyết định dựa trên số lượng tuyển thủ nữ tham gia.

Kết quả của hạng mục đồng đội nam dựa trên kết quả cá nhân của 2 thành viên nam có kết quả tốt nhất đoàn.

Kết quả của hạng mục dành cho tuyển thủ không phải người Trung Quốc và Việt Nam sẽ được tính riêng.

1.3. Dự kiến các chức năng có thể được cài đặt*

1.3.1. Cờ tướng

Chế độ đánh cờ tướng với máy theo luật chơi truyền thống.

1.3.2. Giải cờ thế

Người chơi sẽ nhập thế cờ bằng 2 cách:

- Nhập theo chuẩn biên bản
- Kéo thả quân cờ vào các vị trí

Người chơi sẽ yêu cầu máy chọn 1 trong 2 phe và số nước giải (mặc định là vô hạn).

Máy tính giải thế cờ được nhập và xuất ra các nước đi.

1.3.3. Cờ úp

Ở chế độ cờ úp, các quân cờ ở mỗi bên trừ quân Tướng sẽ được lật úp xuống khi bắt đầu ván cờ, vị trí các quân cờ là ngẫu nhiên. Mỗi lần bắt đầu đi, quân cờ được chọn là trạng thái "úp" ta sẽ di chuyển theo luật đi của quân cờ úp đó, sau nước đi quân cờ sẽ được lật lên. Nếu quân cờ được chọn là trạng thái "lật" ta sẽ di chuyển quân cờ theo luật đi bình thường.

Một số quân cờ được thay đổi về luật đi để phù hợp với cờ úp như quân Sĩ, Tượng không giới hạn vị trí di chuyển.

1.3.4. Undo/ Redo

Người chơi có thể quay lại bước trước.Mỗi ván cờ được undo tối đa 10 lần.Sau khi undo thì người chơi có quyền redo một nước đi.

1.3.5. Reset

Khi người chơi cảm thấy không còn nước đi, hoặc muốn chơi ván mới, người chơi có quyền reset.

1.3.6. Ghi biên bản trận đấu

Biên bản được ghi theo chuẩn System2. Nước đi được xuất ra sau khi chơi.

1.3.7. Tính giờ

Mỗi lượt đi được giới hạn trong thời gian nhất định, nếu quá thời gian trên, bên chưa đi coi như thua.

1.3.8. Load trận đấu cũ

Load biên bản đã lưu trong file *.txt

các chức năng kể trên có thể bị thay đổi trong quá trình thực hiện

1.4. Công nghệ dự kiến sử dụng*

- Typescript
- Angular2
- ...

Áp dụng các thuật toán:

- Greedy
- Minimax
- Monte Carlo tree search
- ...

các công nghệ sử dụng có thể được thay đổi trong quá trình thực hiện

1.5. Mục tiêu đề tài

- Áp dụng các thuật toán tìm kiếm vào trò chơi.
- Xây dựng một trò chơi cờ tướng với các nước đi của máy có sử dụng trí tuệ nhân tạo là tối ưu nhất có thể.
- Tạo ra một khuôn mẫu chung giúp chơi các loại cờ khác nhau bằng cách chỉ cần thay đổi luật chơi và giao diện hiển thị sao cho thích hợp với loại cờ đó.

1.6. Bảng phân công nhiệm vụ

MSSV	Họ và tên	Nhiệm vụ
17521180	Đặng Xuân Trường	• Cài đặt khởi tạo và quản lí thời gian của ván đấu • Tạo giao diện thời gian: Thay đổi thời gian, hiển thị thời gian • Cài đặt luật cho các quân cờ tướng
17520180	Lê Thủy Triều	• Cài đặt luật chiếu tướng • Tạo giao diện chiếu tướng: hiển thị thông báo chiếu tướng nếu quân tướng bị chiếu sau khi đi nước cờ • Cài đặt và quản lý luồng dữ liệu từ client đến server • Hỗ trợ front end
17520964	Nguyễn Đình Quyết	• Cài đặt kiểm tra repeat cho server • Cài đặt thuật toán Alpha − Beta • Nghiên cứu và tổng hợp cơ sở lý thuyết • Tổng hợp và hoàn chỉnh báo cáo
17520144	Trần Kim Sen	• Cài đặt thuật toán Greedy • Cài đặt chế độ cờ úp và cờ thế • Nghiên cứu và sử dụng framework: Angular • Cài đặt chế độ switchturn, redo và undo
17520943	Trần Nguyễn Hồng Quân	• Cài đặt biểu đồ thống kê kết quả ván đầu và hiển thị biên bản. • Thiết kế bàn cờ và quân cờ. • Hỗ trợ front − end • Cài đặt thuật toán MCTS • Tìm hiểu luật giải đấu cờ tướng

Nhóm có hỗ trợ lẫn nhau để hoàn thành những chức năng chưa hoàn thiện như fix bug, test chức năng, đóng góp ý kiến và đưa ra lời khuyên, họp nhóm thường xuyên (khoảng 4 − 5 lần một tuần) để cùng nhau giải quyết vấn đề.

CHƯƠNG 2: CƠ SỞ LÝ THUYẾT

2.1. Bài toán tìm kiếm có đối thủ

2.1.1. Giới thiệu

Xét các trò chơi có hai người tham gia như cờ tướng hay cờ vua. Những trò chơi này có đặc điểm chung là hai người chơi thay phiên nhau ra các nước đi tuân theo luật đi riêng của trò chơi đó, luật này là như nhau cho cả hai người. Vấn đề chơi cờ có thể xem như là tìm kiếm nước đi, tại mỗi lần đến lượt mình người chơi phải tìm trong số rất nhiều nước đi hợp lệ nước đi tốt nhất có thể dẫn đến trạng thái kết thúc có lợi cho người chơi. Tuy nhiên vấn đề tìm kiếm ở đây là tìm kiếm có đối thủ. Người chơi không biết được đối thủ của mình sẽ đi nước nào trong các nước hợp lệ. Đối với các trò chơi này ta có thể xây dựng cây trò chơi để thể hiện tất cả các nước đi có thể có ở mỗi trạng thái, sau đó sử dụng các thuật toán như minimax, Alpha − Beta − pruning, … để tìm ra các nước đi mang lại kết quả có lợi cho người chơi.[8]

2.1.2. Biểu diễn bài toán dưới dạng cây trò chơi

Các trò chơi đối kháng có thể được biểu diễn dưới dạng cây trò chơi[7]:

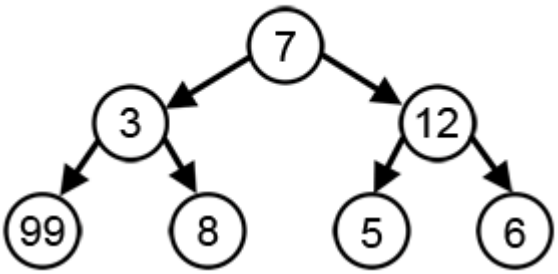
- Gốc là trạng thái ban đầu của trò chơi.
- Các node con thể hiện các nước đi hợp lệ từ một node cha nào đó, node cha đầu tiên là node gốc.
- Các nhánh nối giữa các node cho biết từ một trạng thái của trò chơi chuyển sang trạng thái khác thông qua nước đi nào.
- Các node lá thể hiện thời điểm kết thúc của trò chơi. Mỗi node lá này sẽ được gán một trọng số để xác định kết quả của trò chơi: thắng (1), thua (− 1), hòa (0).

2.2. Greedy Algorithm (Giải thuật tham lam)

Giải thuật tham lam là một thuật toán đơn giản được sử dụng trong các vấn đề tối ưu hóa. Nó giải quyết một bài toán theo kiểu metaheuristic để tìm kiếm lựa chọn tối ưu ở mỗi bước đi với hy vọng tìm được tối ưu toàn cục. Giải thuật tham lam khá thành công

ở một số vấn đề như là Mã hóa Huffman hay thuật toán Dijkstra. Tuy nhiên ở nhiều vấn đề, thuật giải này không là một giải pháp tối ưu.

Ví dụ: Ở hình bên dưới, thuật giải tham lam phải tìm đường đi với tổng giá trị các đỉnh là lớn nhất. Nó thực hiện bằng cách, ở mỗi bước sẽ lựa chọn đỉnh có giá trị lớn nhất kề với đỉnh hiện tại. Bởi vì giải thuật này ra quyết định chỉ dựa trên thông tin tại mỗi bước, không quan tâm đến tổng thể vấn đề nên nó đã sai trong bài toán này.



Hình 2. 1: Ví dụ tìm kiếm của giải thuật tham lam

Nếu cả hai tính chất bên dưới đúng thì giải thuật tham lam có thể dùng để giải quyết vấn đề:

- Greedy choice property: Một giải pháp tối ưu tổng thể có thể đạt được bằng các lựa chọn tối ưu tại mỗi bước.
- Optimal substructure: Một bài toán được gọi là có cấu trúc con tối ưu nếu một lời giải tối ưu của vấn đề tổng thể chứa lời giải tối ưu cho những vấn đề con.

Nói chung, giải thuật tham lam có năm thành phần [13]:

- Một tập hợp các ứng viên (candidate), để từ đó tạo ra lời giải.
- Một hàm lựa chọn, để theo đó lựa chọn ứng viên tốt nhất bổ sung vào lời giải.
- Một hàm khả thi (feasibility), dùng để quyết định nếu một ứng viên có thể được dùng để xây dựng lời giải.
- Một hàm mục tiêu gán giá trị cho lời giải hoặc lời giải chưa hoàn chỉnh
- Một hàm đánh giá, chỉ ra khi nào ta tìm ra một lời giải hoàn chỉnh.

2.3. Thuật toán Alpha − Beta − pruning

Thuật toán Alpha − Beta − pruning được xây dựng dựa trên thuật toán Minimax

2.3.1. Thuật toán Minimax

2.3.1.1. Ý tưởng

Hai đấu thủ trong trò chơi được gọi là MAX và MIN sẽ luân phiên đi nước cờ của mình nên mỗi mức trên cây được biểu diễn luân phiên là MAX và MIN. Mức MAX là mức mà tại đó đấu thủ MAX thực hiện nước đi, mức MIN là mức mà tại đó đấu thủ MIN thực hiện nước đi. Với mỗi nước đi trên bàn cờ tương ứng với các mức trên cây, giải thuật minimax sẽ định giá trị cho các node như sau [9]:

- Nếu node đang xét là node lá thì sẽ gán cho node đó một giá trị để phản ánh trạng thái thắng, thua hay hòa đối với đấu thủ thực hiện nước đi đầu tiên.
- Sử dụng giá trị các node lá để xác định giá trị của các node ở mức trên nó:
    - Node thuộc lớp MAX: gán cho nó giá trị lớn nhất trong các node con.
    - Node thuộc lớp MIN: gán cho nó giá trị nhỏ nhất trong các node con.

Giải thuật Minimax thể hiện việc tính toán nước đi tối ưu cho các đấu thủ thông qua giá trị được gán cho các node. Trong một nước cờ, khi đến lượt đấu thủ MAX thì nước đi được chọn sẽ là nước đi ứng với giá trị cao nhất trong các trạng thái con, còn đấu thủ MIN sẽ chọn một nước đi ứng với trạng thái có giá trị nhỏ nhất trong các trạng thái con.

2.3.1.2. Độ phức tạp của thuật toán

Thuật toán Minimax xét toàn bộ cây trò chơi bằng việc dùng chiến lược tìm kiếm theo chiều sâu. Nên độ phức tạp của thuật toán này tương ứng trực tiếp với kích thước không gian tìm kiếm, trong đó b là hệ số phân nhánh của cây hay chính là nước đi hợp pháp tại mỗi điểm, d là độ sâu tối đa của cây. Số lượng các node sẽ được xét là: $b(b^d - 1)(b - 1)$. Nhưng hàm lượng giá trị sẽ là phương thức chi phối hầu hết thời gian và chỉ làm trên các node lá, vì vậy việc xét các node không phải các node lá có thể bỏ qua. Do đó độ phức tạp thời gian là O(b^d). Bản chất của thuật toán là tìm kiếm theo chiều sâu, vì vậy việc đòi hỏi không gian bộ nhớ của nó chỉ tuyến tính với d và b. Vì thế độ phức tạp không gian là O(bd) [10].

2.3.1.3. Giải thuật

- Bước 1: Tạo toàn bộ cây trò chơi từ trạng thái bắt đầu của trò chơi cho đến trạng thái cuối. Xác định tầng nào cần chọn giá trị MAX, tầng nào cần chọn giá trị MIN.
- Bước 2: Lựa chọn hàm đánh giá phù hợp với trò chơi và xây dựng hàm để tính điểm mà các node đó có thể mang lại cho người chơi MAX, gán giá trị cho mỗi node lá.
- Bước 3: Từ các giá trị được gán ở bước 2 xét và gán giá trị cho các node ở tầng trên dựa theo yêu cầu tầng đó cần gán giá trị MIN hay MAX.
- Bước 4: Lặp lại hành động ở bước 3 với các node ở tầng kế tiếp cho đến khi gán được giá trị cho tầng cuối cùng gần node gốc nhất.
- Bước 5: Từ cây trò chơi đã xây dựng chọn đường đi dẫn đến trạng thái có lợi nhất cho người chơi MAX.

2.3.1.4. Mã giả [11]

```
function minimax(node, depth, maximizingPlayer) is
    if depth = 0 or node is a terminal node then
        return the heuristic value of node
    if maximizingPlayer then
        value := −∞
        for each child of node do
            value := max(value, minimax(child, depth −
            1, FALSE))
        return value
    else //minimizing player
        value := +∞
        for each child of node do
            value := min(value, minimax(child, depth −
            1, TRUE))
        return value
```

2.3.2. Thuật toán Alpha − Beta pruning

2.3.2.1. Ý tưởng

Dựa trên cây trò chơi đã xây dựng theo thuật toán Minimax, thuật toán này sẽ thêm vào cây trò chơi hai giá trị α và β với:

- α là giới hạn dưới tối đa (giá trị nhỏ nhất có thể) của các giải pháp.
- β là giới hạn trên tối thiểu (giá trị lớn nhất có thể) của các giải pháp.

Hai giá trị này được thêm vào để có thể loại bỏ các trường hợp không cần xét ra khỏi cây. Giá trị khởi tạo ban đầu của α là − infinity và β là +infinity. α là giá trị của lựa chọn tốt nhất hiện tại trong các node con đối với người chơi MAX, nếu có một giá trị nào đó trong các node con mà nhỏ hơn α thì nhánh đó sẽ bị loại bỏ ra khỏi cây trò chơi. Ngược lại β là giá trị lựa chọn hại nhất đối với người chơi MAX, nếu có một giá trị nào đó trong các node con mà lớn hơn β thì nhánh đó sẽ bị loại bỏ ra khỏi trò chơi. Giá trị α chỉ có thể được cập nhật khi đến lượt của người chơi MAX, và giá trị β cũng chỉ được cập nhật khi đến lượt của người chơi MIN. Tại mỗi node của cây trò chơi trong thuật toán này phải thỏa mãn điều kiện có $\alpha \leq \beta$, nếu không thỏa mãn điều kiện này thì node đó sẽ bị xóa khỏi cây trò chơi. Sau khi xóa đi các node không cần thiết ta được cây trò chơi mới có ít node cần xét hơn so với cây trò chơi được xây dựng theo thuật toán Minimax. Đây là ý tưởng cơ bản của thuật toán Alpha − Beta − pruning.

2.3.2.2. Giải thuật

- Bước 1: Xây dựng cây trò chơi dựa theo giải thuật Minimax.
- Bước 2: Khởi tạo α = − infinity và β = +infinity cho node đầu tiên của cây.
- Bước 3: Mang các giá trị α, β đến các node tiếp theo. Sau đó cập nhật giá trị α và β cho từng node của cây bắt đầu từ node gần lá nhất tính từ bên trái qua theo nguyên tắc:
    - Nếu node đang xét thuộc tầng MIN: so sánh giá trị được gán cho node bởi hàm đánh giá với giá trị β tại node đó nếu giá trị đó nhỏ hơn thì cập nhật giá trị này cho β. Ghi nhớ lại β (thu hẹp khoảng $[\alpha, \beta]$ bằng cách giảm giá trị β).

- Nếu node đang xét thuộc tầng MAX: so sánh giá trị được gán cho node bởi hàm đánh giá với giá trị α tại node đó nếu giá trị đó lớn hơn thì cập nhật giá trị này cho α. Ghi nhớ lại α (thu hẹp khoảng $[\alpha, \beta]$ bằng cách tăng giá trị α)

- Bước 4: Lặp lại bước 3 với các node kế tiếp cho đến khi gán được giá trị α, β cho node gốc. Trong quá trình cập nhật giá trị α và β cho mỗi node xóa đi các node không thỏa điều kiện.
- Bước 5: Từ cây trò chơi đã xây dựng chọn đường đi dẫn đến trạng thái có lợi nhất cho người chơi MAX.

2.3.2.3. Mã giả [12]

```
function Alpha Beta(node, depth, α, β, maximizingPlayer)
is
    if depth = 0 or node is a terminal node then
        return the heuristic value of node
    if maximizingPlayer then
        value := −∞
        for each child of node do
            value := max(value, Alpha Beta(child, depth
            − 1, α, β, FALSE))
            α := max(α, value)
            if α ≥ β then
                break (* β cut − off *)
        return value
    else
        value := +∞
        for each child of node do
            value := min(value, Alpha Beta(child, depth
            − 1, α, β, TRUE))
            β := min(β, value)
```

```
            if α ≥ β then
                break (* α cut – off *)
        return value
```

2.3.3. Đánh giá thuật toán

Hiệu quả của thuật toán Alpha − Beta phụ thuộc nhiều vào thứ tự các nước đi kế tiếp được thực hiện. Nếu các nước đi kế tiếp có thứ tự ngẫu nhiên thì tổng số node được thực hiện sẽ khoảng $O(b^{\frac{3d}{4}})$. Trong đó b là độ rộng của cây (hệ số phân nhánh trung bình của các con), d là độ sâu của cây.Trong điều kiện lí tưởng thuật toán chỉ phải xét số node theo công thức: $2b^{\frac{d}{2}} - 1$ (với d chẵn), $b^{d+\frac{1}{2}} + b^{\frac{d}{2}} - 1$ (với d lẻ). Trong trường hợp tốt nhất độ phức tạp thời gian của thuật toán là $O(b^{\frac{d}{2}})$.

Thuật toán Alpha − Beta giúp tiết kiệm nhiều thời gian so với Minimax và vẫn đảm bảo kết quả tìm kiếm vẫn chính xác. Tuy nhiên, số lượng các node bị cắt bỏ là không xác định đối với mỗi bài toán. Trong trường hợp xấu nhất là thuật toán không cắt bỏ được một nhánh nào và phải xét số node đúng bằng thuật toán Minimax.

2.4. Thuật toán Monte carlo tree search

2.4.1. Monte carlo methods

Monte carlo methods (tên khác là monte carlo experiments) là một lớp các thuật toán dựa vào việc lặp lại ngẫu nhiên một cách thức nào đó để lấy được kết quả số học cho một vấn đề. Phương pháp này thường được áp dụng cho các vấn đề vật lý, toán học và có hiệu quả nhất đối với những bài toán khó hoặc không thể giải quyết bằng những hướng tiếp cận khác. Monte carlo methods được dùng giải quyết các bài toán thuộc 1 trong các lớp: optimization (tìm giá trị tối ưu), numerical integration (tính giá trị xấp xỉ của hàm tích phân, vi phân) và sinh số dựa trên phân phối xác suất.

Monte carlo methods được phát triển bởi Stanislaw Ulam và John von Neumann vào cuối những năm 1940 và được đặt tên theo sòng bạc Monte Carlo ở Monte Carlo, Monaco.

2.4.2. Monte Carlo Tree Search

Bằng việc kết hợp ý tưởng của tìm kiếm minimax và mô hình dự đoán kết quả thông qua các hành động ngẫu nhiên của Monte Carlo methods, Monte Carlo Tree Search (MCTS) được giới thiệu trong công trình của Rémi Coulom năm 2006[16].

MCTS là một phương pháp tìm kiếm kết quả tối ưu trong một lĩnh vực bằng việc áp dụng những hành động ngẫu nhiên trong không gian trạng thái và xây dựng thành cây dựa trên kết quả thu được. MCTS đã tạo ra được những tác động lớn đối trong việc ứng dụng artificial intelligence (AI − trí tuệ nhân tạo) cho những vấn đề có thể biểu diễn thành cây dựa trên các hành động tuần tự, cụ thể trong planning problems và trong combinatorial games[17].

Những nghiên cứu quan tâm đến MCTS ngày càng tăng lên nhờ vào sự thành công trong việc tính toán các nước đi của cờ Go[18] và tiềm năng ứng dụng vào nhiều vấn đề khó. Trên lý thuyết, MCTS có thể áp dụng cho bất kỳ lĩnh vực nào có thể biểu diễn dưới dạng cặp {state, action} và có thể dự đoán được output.

Ý tưởng mà MCTS muốn thực hiện là thực hiện việc tìm kiếm ở phần có những nước đi có triển vọng của cây (chẳng hạn những nước đi dẫn tới có thể ăn xe của đối phương) chứ không mở rộng việc tìm kiếm ở toàn bộ cây. Nhưng vậy mục đích của MCTS sẽ tìm kiếm trên một cây bất đối xứng (Asymmetric Tree).

MCTS dựa trên 2 khái niệm nền tảng: giá trị ước lượng cho một state và giá trị này được dùng hiệu quả trong việc điều chỉnh quá trình duyệt cây để đạt được quyết định tốt nhất. Đối với trò chơi cờ tướng, với mỗi nước đi nhất định của trò chơi, thuật toán sẽ dần dần tạo một cây trò chơi nhờ vào kết quả của việc khám phá cây. Cây vừa được xây dựng sẽ được dùng để ước tính giá trị của nước đi đó và giá trị ước tính này sẽ được cải thiện độ chính xác khi cây dần được mở rộng.

2.4.3. Chi tiết thuật toán

Với MCTS, cây sẽ dần dần phình ra sau mỗi vòng lặp, vì vậy để có thể giới hạn chi phí tính toán, ta sẽ đặt một giới hạn cho việc tìm kiếm và sau khi đặt tới giới hạn, MCTS sẽ

trả về node chứa trạng thái được xem là "tốt nhất". Giới hạn này được gọi là computational budget và thường có thể là giới hạn thời gian, bộ nhớ hoặc số vòng lặp.

Bốn bước thực hiện trong mỗi vòng lặp tìm kiếm bao gồm [21]:

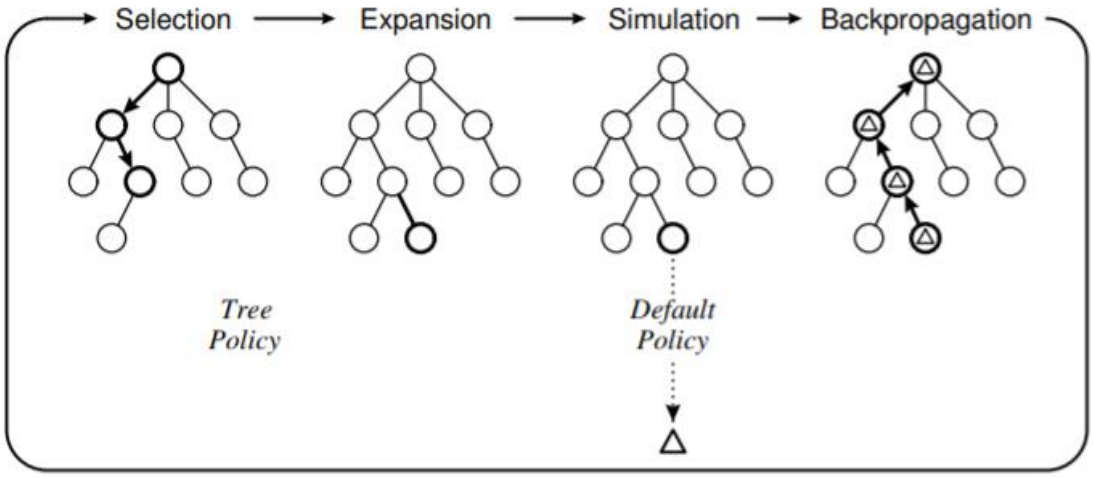


Fig. 2. One iteration of the general MCTS approach.

Hình 2. 2: Các bước của MCTS trong 1 vòng lặp. Nguồn: [21]

1. **Selection:** Trong quá trình này, MCTS bắt đầu duyệt cây từ root node dựa trên một chính sách (gọi là Tree Policy) đánh giá giá trị các child node của node đang xét và chọn ra child node xét tiếp theo có giá trị lớn nhất trong các child node. MCTS dùng công thức Upper Confidence Bound (UCB) để thực hiện việc đánh giá này:

$$UCB1 = \overline{X_j} + c \times \sqrt{\frac{ln(N)}{n_j}} \quad (2.1)$$

trong đó: $\overline{X_j}$: giá trị ước lượng trung bình của child node thứ j.

N : số lần parent node được duyệt.

n_j : số lần child node thứ j được duyệt.

c : là một tham số để điều chỉnh việc "exploration − exploitation trade − off".

Trong quá trình selection, child node trả về giá trị UCB lớn nhất sẽ được chọn (thường là các node chưa được duyệt lần nào, vì n_j = 0 dẫn đến UCB là ∞). Khi cây được duyệt tới node có thể mở rộng (expandable), MCTS chuyển sang bước expansion.

2. **Expansion:** Một node được coi là expandable nếu node đó không phải là non − terminal node (node đại diện trạng thái chưa phải là trạng thái kết thúc) và là leaf node hoặc là node vẫn còn các action chưa được thực hiện. Một hoặc nhiều nodes mới sẽ được thêm vào làm child nodes cho node hiện tại, dựa vào các action có thể thực hiện tại node đó.

3. **Simulation:** thực hiện quá trình rollout/simulated với một hoặc nhiều nodes vừa thêm dựa vào một chính sách khác gọi là Default Policy (hay Rollout Policy) để tới được terminal node (node đại diện trạng thái kết thúc). Việc rollout tùy thuộc vào bài toán và thường việc chọn các action để thực hiện là ngẫu nhiên.

4. **Backpropagation:** sau khi biết được giá trị cho node vừa mới rollout, quá trình backpropagation được tiến hành cập nhật từ node mới tới root node. Trong quá trình cập nhật, số lần simulation của node được tăng lên và đồng thời nếu kết quả của rollout tới được win state thì số lần thắng của nodes cũng được cập nhật. Với số lần thắng và số lần simulation tại 1 node, ta có thể biết được xác suất thắng của node đó.

Tree Policy và Default Policy có thể được hình dung là:

- Tree Policy: là cách ta chọn leaf node trong nodes đã có trong cây.
- Default Policy: là cách ta đi từ một node là non − terminal state tới terminal state để có thể có được 1 giá trị ước lượng nào đó.

2.4.4. Mã giả MCTS cơ bản

function $MctsSearch(state\ s_0)$**:**

    create root node v_0 with $state\ s_0$

    while within computational budget **do:**

        $v_l \leftarrow TreePolicy(v_0)$

$\Delta \leftarrow DefaultPolicy(state(v_l))$

$Backup(v_l, \Delta)$

return $action(BestChild(v_0))$

Trong mã giả trên, v_0 là node ứng với $state\ s_0$, v_l là node cuối cùng được chọn trong bước selection trước khi tiến hành mở rộng ứng với $state\ s_l$ và Δ là kết quả trả về sau khi simulated tới terminal state của $state\ s_l$. Kết quả tìm kiếm trả về sẽ là $action\ a$ mà với action này sẽ có được cách đi "tốt".

Fig 2 cho thấy cách chạy của thuật toán MCTS đơn giản. Bắt đầu từ root node v_0 dùng đệ quy để duyệt và chọn được child node dựa trên UCB, node được chọn cuối cùng sau khi không thể duyệt được nữa v_l. Với mỗi $action\ a$ có thể có của $state\ s_l$ (tương ứng node v_l) hoặc $action\ a$ mà $state\ s_l$ chưa thực hiện thì sẽ thêm một child node t_l tương ứng vào cây.

Từ node t_l mới, quá trình tìm ngẫu nhiên các action để có thể tới được terminal state được thực hiện và sẽ trả về một giá trị Δ dựa trên terminal state (giá trị này sẽ tùy thuộc vào bài toán để chọn, có thể là một giá trị rời rạc đại diện win / draw / loss hoặc một giá trị liên tục, hoặc có thể là 1 vector các giá trị đối với bài toán multi − agent). Sau khi đã có được Δ, tiến hành cập nhật số lần đã duyệt (số lần simulated) và số lần tới được win state (với Δ mang giá trị rời rạc) hoặc tổng (hoặc trung bình) các Δ tất cả lần simulated (với Δ mang giá trị liên tục) từ node v_l đi ngược lên tới root node v_0.

Tới khi việc tìm kiếm bị dừng hoặc tới giới hạn của computational budget, $action\ a$ được trả về mà thông qua action đó thì root node v_0 sẽ tìm được child node "tốt nhất". Tiêu chí "tốt nhất" sẽ tùy thuộc vào việc cài đặt cho mỗi bài toán (ví dụ: child node có Δ lớn nhất; child node được simulated nhiều nhất; child node có UCB cao nhất; v.v...).

2.4.5. Ưu điểm

Trong thực tế việc tìm kiếm trên cây, sẽ luôn có khả năng rằng action tốt nhất hiện tại không phải là action tối ưu nhất. Trong trường hợp như vậy, MCTS thường trở nên hữu dụng hơn do thuật toán luôn đánh giá các action khác bằng việc thử thực thi chúng thay

vì chỉ tập trung cho một action. Điều này được biết đến là "exploration − exploitation trade − off".

Exploration (khám phá) giúp cho việc tìm những phần không có trên cây (node chưa được duyệt) để nếu có sẽ tìm được đường đi tốt hơn. Thông thường làm như vậy sẽ mở rộng về chiều rộng của cây. Exploration đảm bảo việc MCTS không quá chú trọng vào một nhánh nào đó mà có thể bỏ qua các node có tiềm năng. Trong khi đó exploitation (khai thác) gắn với việc mở rộng theo chiều sâu cho một nhánh có giá trị tốt nhất. Nếu chỉ tập trung cho một trong hai sẽ khiến cây mất cân bằng và kém hiệu quả, đặc biệt là khi số vòng lặp tăng lên. Bằng việc dùng kết quả UCB để đánh giá node, MCTS đã cân bằng được cả hai yếu tố, giúp nó hiệu quả trong việc tìm kiếm quyết định tối ưu cho các bài toán áp dụng AI.

Do MCTS không yêu cần một cách duyệt cụ thể, kế hoạch chiến thuật của một lĩnh vực để tìm được quyết định hợp lý nên MCTS vẫn có thể hoạt động hiệu quả dù người cài đặt không có kiến thức rộng về trò chơi. MCTS chỉ cần biết các nước đi hợp lệ và điều kiện dừng trò chơi. Điều đó có nghĩa MCTS có thể được tận dụng cho nhiều trò chơi với chỉ một số điều chỉnh.

Thuật toán sẽ trả về kết quả được cho là tốt nhất một khi tới giới hạn của computational budget. Cây được xây dựng có thể được vứt bỏ hoặc tận dụng lại ở các nước đi tiếp theo.

Có thể song song hóa trên nhiều máy tính.

2.4.6. Hạn chế

MCTS yêu cầu việc simulation phải dễ thực hiện và nhanh chóng.

MCTS vẫn có thể không trả về được action hợp lý do kích thước tập các state hợp lệ đối với những bài toán phức tạp và các node có thể chưa được simulated đủ số lần để có được một ước lượng đáng tin. Việc tìm kiếm kết quả tối ưu trên bài toán như vậy cũng làm tăng thời gian của thuật toán.

CHƯƠNG 3: THIẾT KẾ GIAO DIỆN VÀ LUỒNG XỬ LÝ DỮ LIỆU TRONG SẢN PHẨM

3.1. Giới thiệu sơ lược về angular 2 [20]

- Angular 2 là một framework UI để xây dựng ứng dụng web trên desktop và mobile
- Nó được xây dựng dựa trên Javascript. Chúng ta có thể dùng nó để xây dựng một ứng dụng client side thú vị dùng HTML, CSS và Javascript
- Angular 2 có rất nhiều cải tiến so với Angular 1 để dễ dàng học và phát triển các ứng dụng có quy mô doanh nghiệp
- Với Angular 2 chúng ta dễ dàng xây dựng được 1 ứng dụng có thể dễ dàng mở rộng, bảo trì, kiểm nghiệm và chuẩn hóa ứng dụng của mình

3.2. Front − end

Sử dụng HTML, CSS, framework SemanticUI và các hàm được Angular2 hỗ trợ (ngif && ngfor) để thiết kế một giao diện bàn cờ mang màu sắc tươi sáng, nhẹ nhàng, thân thiện với người dùng.

- Folder component_board : gồm 2 file html và css
- html của file được thiết kế khá đơn giản nhờ việc gọi các attribute được thế kế sẵn từ class BoardComponent.
- Ở đây cần chú ý 2 phần :
    - Phần 1 : Xử lý sự kiện click vào 1 quân cờ, quân cờ này sẽ đưa ra các gợi ý nước đi tiếp nếu đi đc. Các ô gợi ý sẽ có style là class selected trong file css. Trong phần này ta kiểm tra:
        - selectedPiece có được selected hay chưa
        - `state.playingTeam ==1` đã tới turn đi hay chưa.
        - Nếu thỏa 2 đk trên thì bắt đầu xét tiếp

- Lấy tất cả các ô trên bàn cờ bằng hàm dummyPieces và sử dụng hàm để kiểm tra xem ô nào trong các ô trên hợp lệ bằng hàm isPossibleMove nếu thỏa mãn điều kiện này thì các ô sẽ có màu là style trong file css và lấy style =là ".possibleMove " nếu không thỏa sẽ không có màu. Tiếp theo set lại giá trị cho tọa độ bằng cách set thuộc tính [style.left] và [style.bottom] cho nút đó. Ở đây mỗi quân cờ nằm như là một thẻ button vị trí quân cờ là vị trí của thẻ button.

- Phần 2 : Hiển thị tất cả các quân cờ còn sống cả 2 quân. Khá đơn giản việc show các quân cờ này ta sử dụng ngfor và ngif để thực hiện cụ thể như sau

  - Đối với quân đỏ :

    - Lấy tất cả các quân cờ đỏ

```
state.redAgent.myPieces:

*ngFor="let piece of state.redAgent.myPieces"
```

    - Ứng với mỗi piece trong quân đỏ ta hiển thị nó lên nên web bằng việc set giá trị [style.left] và [style.bottom] và lấy sự kiện quân cờ bất kì nào đó có đc selected hay không cập nhật về trạng thái để tiến hành xử lý in ra các bước đi hợp lệ cho nó và thêm binding để set thuộc tính color cho quân cờ được chọn [class.selected]

  - Tương tự với quân đen

3.3. Mối liên hệ giữa client và server

Với mỗi bước đi, các quân cờ đỏ và đen luân phiên nhau di chuyển.

- Bước 1: Ngay khi bắt đầu chạy chương trình để mở giao diện web, file index.html ở folder public được gọi đến. Trong file này gọi thẻ <app> nhằm đưa đến class app ở file *app.component.main.ts* có định dạng các file html và css của nó. Trong

file app.component.main.html lại gọi thẻ <board> nhằm thể hiện bàn cờ trên web cùng các hàm xử lý (được diễn giải chi tiết ở chương cài đặt) thông qua file *board.ts* (thuộc *client/app/component_board*).

Ở bước này, các tham số ban đầu như trạng thái quân cờ, phe cờ hiện tại (đỏ, đen), trạng thái kết thúc cờ, trạng thái chiếu tướng… được khởi tạo.

- Bước 2: Mỗi bước di chuyển của các quân cờ sẽ gọi đến hàm **switchTurn()** (thuộc *client/app/component_board/board.ts*). Sau mỗi bước đi, giá trị về checkmate, playingTeam, thời gian … sẽ được cập nhật để phù hợp với 2 phe. Với quân đỏ (là quân của người chơi) các hàm được gọi thuộc các class ở các file trong folder client chủ yếu để bắt sự kiện người chơi di chuyển quân cờ và kiểm tra luật để hiển thị các nước đi hợp lệ trên bàn cờ. Với quân đen (quân cờ của địch) thì các hàm được gọi thuộc các class ở cá file trong folder server chủ yếu là xử lý 3 thuật toán để trả về nước đi hợp lệ và tối ưu nhất có thể dựa trên mỗi thuật toán ấy. Lúc này client sẽ gửi trạng thái bàn cờ hiện tại qua server truy xuất đến hàm **launchComputer(state)** để tìm nước đi tiếp theo hoặc **checkMate(state)** để kiểm tra trạng thái chiếu tướng (thuộc *client/app/service/service.computer.ts*) và thông qua phương thức PUT (Thay đổi tất cả các đại diện hiện tại của nguồn mục tiêu với nội dung được tải lên) để truy cập và xử lý dữ liệu trên server. Sử dụng **toPromise()** để xử lý bất đồng bộ, với cấu trúc.**then()** và.**catch()** Nếu thực hiện được việc Put dữ liệu thì sẽ dẫn tới then() còn không sẽ catch() quăng ra handleError.

Tại file *www.ts* (thuộc *server/bin*) server bắt đầu gọi đến các thuật toán để xử lý với tham số là trạng thái bàn cờ hiện tại được client gửi đến. Sau khi kết thúc việc tính toán, server sẽ trả về nước đi tiếp theo *next* và thời gian *t* (đối với hàm **launchComputer(state)**), trạng thái chiếu tướng (đối với hàm **checkMate(state)**). Sau khi client nhận được nước đi tiếp theo của server, nó có nhiệm vụ biểu diễn nước đi ấy ra bàn cờ, đồng thời cập nhật lại trạng thái bàn cờ hiện tại.

- Bước 3: Lặp lại bước 2 cho đến khi trạng thái kết thúc thỏa mãn điều kiện, dừng trận đấu.

CHƯƠNG 4: CHI TIẾT CÀI ĐẶT

4.1. Bàn cờ

- Tổng thế bàn cờ được cấu trúc như sau:
    - Một bàn cờ được biểu diễn là một state trong một state gồm có hai Agent để quản lý quân đỏ và quân đen. Đối với một agent có các quân cờ (piece), đối với một piece quản lý tên quân, vị trí quân đó, một số hàm bổ trợ khác.
    - Việc tổ chức và cấu trúc như trên ta dễ dàng chỉnh sửa và nâng cấp.
    - Ngoài ra luật chơi được cài đặt *Rule.ts* là một file riêng chỉ hỗ trợ cho state nên với bất kì loại cờ nào chúng ta chỉ cần chính trong file luật thì nó sẽ chạy được mọi loại cờ.

4.2. Quân cờ

- Một class piece đại diện cho quân cờ:

Class piece bao gồm:

- Name: kiểu string: tên quân cờ.
- Position[number,number]: mảng gồm 2 phần tử cho biết vị trí của quân cờ theo tạo độ.
- Reverse: kiểu boolean: là loại cờ úp hay không.
- Truthname: kiểu string: tên thật của quân cờ.
- IsMove: kiểu number: số lần di chuyển quân cờ.
- Gồm các hàm:
    - **Construct:** gán tên và vị trí hiện tại của quân cờ.
    - **moveTo(newPos):** di chuyển quân cờ tới vị trí newPos,mỗi lúc di chuyển đổi tên thành tên thật để cập nhật tên trong cờ úp, cập nhật luôn số nước di chuyển isMove = isMove +1 để tính toán số lần id chuyển của một nước hỗ trợ trong cờ úp.
    - **copy():** copy tất cả thuộc tính của quân cờ name, position, Revese, truthname, isMove

4.3. Khởi tạo các quân cờ

Việc khởi tạo các quân cờ được cài đặt trong file *client\app\ChineseChess\InitGame\init.ts* gồm 2 hàm **getRedPiece** và **getBlackPiece:**

Hàm **getRedPiece()** hàm này trả về mảng các quân cờ (piece), đối với cờ tướng thường thì tên sẽ xếp như bình thường và name và truthname (name và truthname này là biến trong class piece) như nhau, truthname hiện tại sẽ lấy giá trị của Tname, còn với cờ úp thì Tname được xáo lên theo việc random các số từ 0 − 14, Tname sẽ có các name mới, vì trong cờ úp sẽ có trường hợp một trên bàn cờ có thể có 4 còn xe 2 con đang úp với tên giả là xe, 2 con ngựa với tên thật là xe, cũng như đối với các quân khác, do đó khởi tạo các tên giả ta cần Newname các tên như j3, j4 cho con xe 3 và xe 4, xáo trộn các Tname lên với vị trí trong rand. Sau đó khởi tạo các quân như bình thường với truthname là Tname. Ngoại trừ quân tướng thì trong cờ úp hay thế ta vẫn giữ nguyên vị trí ban đầu. Tương tự đối với các quân đen − hàm **getBlackPiece().**

Hàm **RandomPosition** (là hàm hỗ trợ cho cờ úp) hàm này sinh trước 20 thế cờ để đỡ tốn time mỗi lần đi sinh ra các thế cờ, hàm sử dụng biến hằng sau đó random một thế cờ bất kì từ 0 −>19. Trả về một mảng các số 0 −>15 đây là các vị trí random mới của các quân cờ trong bàn cờ.

Ví dụ: new Piece('j1',RedTeam[0], reverse, Tname[0],0): quân piece này là một cờ class, class này gồm "tên quân , vị trí quân, loại cờ, tên thật, số nước di chuyển" lúc khởi tạo khởi tạo nhiều quân cờ đồng nghĩa sẽ tạo ra nhiều class piece khác nhau, RedTeam là một mảng chứa mặc định các vị trí của quân đỏ, còn đối với quân đen sẽ là BlackTeam

Còn đối với cờ thế chúng ta sẽ tự động khởi tạo lúc lấy được giá state của bàn cờ, ngay khi submit state đó. Được cài đặt trong *board.ts*, phần này sẽ được giải thích rõ hơn ở bên dưới.

4.4. Xây dựng luật chơi

4.4.1. Tổng quan

Xây dựng cách chơi cho server trước, luật được build trong cả client và server.

Sever sẽ coi quân của client là đối thủ và ngược lại client sẽ coi quân của server là đối thủ. Tuy nhiên cả sever và client đều được cài đặt tương tự nhau, chỉ cần set một thuộc tính trong boardstate quân hiện tại cần xét là quân nào, và quân đối thủ là quân nào.

Luật chơi được xây dựng trong folder *Client/app/ChineseChess/Rule/rule.ts* là luật cho cờ tướng và có xét cả luật của cờ úp.

Chức năng quan trọng của các hàm này ta sẽ trả về các nước đi hợp lệ của các quân cờ trên bàn cờ, và kiểm tra xem ván cờ đã kết thúc hay không (kiểm tra quân tướng còn sống hay đã chết).

Máy tính được đặt mặc định quân đen, người chơi được đặt là quân đỏ có flag là 1 và quân đen có flag là -1 (vì cài đặt trong client, còn đối với sever sẽ ngược lại và trong server có một số chức năng đặc biệt được nâng cấp để áp dụng cho các thuật toán minimax giả lập các trường hợp thay đổi quân đen và đỏ liên tục).

Để tiện cho việc cài đặt nhóm đã sử dụng thêm biến và object:

- static minRow =1
- maxRow = 10
- minCol = 1
- maxCol = 9.
- boardStates : là một object {a : b}
    - Trong đó a là 1 string chứa tọa độ x,y (sử dụng hàm convert mảng sang string trong typescript, vd : [1,2].toString() sẽ là "1,2")
    - b là 1 mảng gồm 2 phần tử phần tử đầu là tên quân cờ, phần tử thứ 2 là quân cờ team của quân cờ.
- Các chức năng cài đặt trong file :
    - hasPieceOnRows(col, minRow, maxRow, boardStates:{})
    - numPieceOnRows(col, minRow, maxRow, boardStates)
    - filterBoundedMoves(currRow, currCol, moves, boardStates)
    - movesOnSameLine(currRow, currCol, boardStates)
    - possibleMovesForJu(currRow, currCol, boardStates)

- possibleMovesForMa(currRow, currCol, boardStates)
- findFirstOpponentOnRow(row, startCol, states, team, incFn)
- possibleMovesForPao(currRow, currCol, boardStates, team)
- possibleMovesForShi(currRow, currCol, boardStates, isLowerTeam)
- possibleMovesForKing(currRow, currCol, boardStates)
- possibleMovesForXiang(currRow, currCol, boardStates, isLowerTeam)
- possibleMovesForZu(currRow, currCol, boardStates, isLowerTeam)
- possibleMoves = function(piece: Piece, boardStates: {}, isLowerTeam)
- allPossibleMoves = function(myPieces: Piece[], boardStates: {}, team)
- getGameEndState = function(agent)
- getGameEndStateByState = function(myPieces: Piece[], oppoPieces: Piece[], boardState, team)

4.4.2. Chi tiết cài đặt luật cho cờ tướng (file *rule.ts*)

- Hàm **hasPieceOnRows(col, minRow, maxRow, boadStates: {}):**

Kiểm tra xem có quân nào cùng hàng trên cùng một col từ giá trị row nhỏ nhất (minRow) tới giá trị row lớn nhất là (maxRow).

Việc kiểm tra khá đơn giản ta chạy i từ minRow tới maxRow xem giá trị trên ô [i,col] có trong boardStates[[i,col].toString()] hay không.

Hàm trả về kết quả có hay không.

- Hàm **numberpieceRowOnRow(col,minRow,maxRow,boardStates):**

Hàm đếm số lượng quân trên cùng cột (cột là biến col)

```
For (i = minRow, i<= maxRow,i++){
Nếu boardStates[[i,col].toString] có thì ta đếm giá trị
}
```

Hàm trả về số lượng của quân cờ cùng cột col.

- Hàm **movesOnSameLine(currRow, CurrCol, boardStates):**

CurrRow và CurrCol là vị trí hiện tại của quân cờ đang xét.

Hàm trả về các vị trí có thể đi từ vị trí hiện tại sang các hướng trái, phải, trên, dưới: di chuyển cho tới khi gặp 1 quân nếu là quân địch thì vị trí có thể di chuyển vào ô quân địch đang đứng (nghĩa là ăn quân đó) nếu là quân mình thì không thêm vào các ô có thể di chuyển.

Từ vị trí hiện tại currRow, currCol ta xét currRow + 1 chạy tới maxRow để xem các nước quân cờ đó có thể di chuyển tới đó hay không trên cột col:

```
For (i = currRow + 1, i<=maxRow ; i++ ){
     var k = [i,currCol].toString();
// convert mảng [i,currCol] sang string để lấy key trong object boardState
//Nếu k có xuất hiện trong boardStates thì ta sẽ kiểm tra quân cờ đó là quân mình hay quân địch
//Kiểm tra là quân nào sẽ là : boardState[k][1]
     if (k in boardStates) {
          if (!boardStates[k][1])
               moves.push([j, currCol]);
          break;
//Thoát khỏi vòng lặp vì đã chạm phải 1 quân nên không thể di chuyển được
     }
// Nếu chưa kết thúc nghĩa là nước này có thể đi ta sẽ thêm vào moves
     moves.push([i,col]);
}
```

Tương tự như trên từ vị trí hiện tại [CurrRow,CurrCol] xét các bước đi:

- Từ CurrRow − 1 xuống minRow.
- Từ CurrCol+1 tới maxCol.
- Từ CurrCol − 1 xuống minCol.

Lưu lại các bước có thể di chuyển. Hàm trả về moves là tất cả các trường hợp có thể di chuyển được từ vị trí [CurrRow,CurrCol]

- Hàm **possibleMovesForJu(currRow,currCol,boardState):** Hàm cài đặt các nước có thể đi của quân xe.

Quân xe được phép đi sang trái, sang phải, lên trên, xuống dưới. Nói cách khác nó sẽ di chuyển trên cùng dòng hoặc cùng cột cho tới khi gặp quân mình hoặc quân địch nếu là quân địch thì có thể ăn quân đó. Đây cũng chính là hàm **movesOnSameLine()** đã được cài đặt sẵn ở trên. Vì vậy để cài đặt ta chỉ cần return về hàm **movesOnSameLine(currRow, currCol, boardStates);**

- Hàm **possibleMovesForMa(currRow, currCol, boardStates):**

Quân mã di chuyển theo hình chữ L và không có quân cản trên đường di chuyển (luật di chuyển đã nêu ở trên).

Cài đặt di chuyển cho quân mã khá đơn giản, đây là một ví dụ:

Nếu `([currRow + 1, currCol].toString())` không có trong boardStates) thì :

```
moves.push([currRow + 2, currCol + 1]);
moves.push([currRow + 2, currCol − 1]);
```

Vì quân mã không thể di chuyển khi có quân cản trước hướng nó muốn đi, trong trường hợp này là ô [currRow+1,currCol] kiểm tra xem có quân này trong state hay không nếu không thì ta có thể di chuyển sang ô [currRow+2,currCol + 1] và [currRow +2, currCol − 1].

Cài đặt tương tự với các hướng khác. Hàm trả về các ô quân mã có thể di chuyển từ vị trí [currRow,currCol] bằng cách return lại mảng moves.

- Hàm **findFirstOpponentOnCol(row, startCol, states, team, incFn):**

Hàm nhận vào vị trí hiện tại quân cờ và bàn cờ hiện tại. Hàm tìm quân địch đầu tiên trên cùng row. Nếu gặp quân cùng team sẽ ngừng tìm kiếm, nếu gặp quân khác team thì sẽ return tọa độ quân địch.

Code khá đơn giản: Tham số truyền vào hàng và cột hiện tại, có states là bàn cờ hiện tại và hàm inFn.

- Hàm inFn là một kiểu khá đặc biệt trong typescript, bên ngoài truyền inc = (x=>x+1) thì inFc sẽ là hàm (x=>x+1) tăng x lên một đơn vị, hoặc là (x=>x − 1) giảm x đi 1 đơn vị.
- Ở đây chỉ cần truyền vào 2 hàm là x=>x+1 và x=>x − 1 thì một hàm có thể duyệt cả phần tử nằm trên trái của vị trí hiện tại và bên phải vị trí hiện tại.
- Duyệt khi strartCol>=this.minRow và startCol <= this.maxRow.
- Lấy vị trí hiện tại k = [startRow, Col].toString() kiểm tra xem k có trong states hay k.
- Cập nhật startCol theo hàm inFn(startCol).

Hàm trả về quân địch đầu tiên trên cột tuỳ theo incFn mà trả về quân địch bên trái hay bên phải so với vị trí hiện tại của quân cờ.

- Hàm **findFirstOpponentOnRow(startRow, Col, states, team, incFn):**

Hàm này tương tự hàm trên và chỉ thay đổi cập nhập cho startRow thay vì col.Hàm trả về quân địch đầu tiên xuất hiện tuỳ vào incFn là quân địch bên trên hay dưới so với vị trí hiện tại.

- Hàm **possibleMovesForPao(currRow, currCol, boardStates, team):**

Tìm các nước di chuyển của pháo, ở đây truyền vào các tham số currRow, currCol, boardStates, team.

Định nghĩa trước hàm tăng và giảm, inc = (x=>x+1) và dec = (x=>x − 1) đây là một điều thú vị trong typescript.

Tại một vị trí cần tìm quân đang xuất hiện trong bàn cờ trước, giả sử ta duyệt về phía trước mặt, thì từ vị trí hiện tại duyệt từ row tới max row và col giữ nguyên, nếu phát hiện có quân đứng trước mặt thì gọi tiếp hàm để tìm quân địch xuất hiện đầu tiên sau đó trả về kết quả là ô có quân địch xuất hiện. Tương tự với các hướng dưới, trái, phải.

- Hàm **possibleMovesForShi(currRow, currCol, boardStates, isLowerTeam):** trả về các nước di chuyển hợp lệ của quân Sĩ

Tương tự như các hàm khác truyền vào các biến vị trí hàng hiện tại, cột hiện tại, bàn cờ, và ở đây truyền thêm team nào (isLowerTeam là biến xét xem quân bên này hay bên kia sông).

Ở đây ta xét tổng quát cho cả quân địch và quân mình mặc dù hàm chỉ viết cho quân đỏ nhưng đối với server thì ta cần xét cả hai, ở đây viết một lần nhưng dùng cho cả server và client.

Nếu con sĩ ở vị trí biên thì chỉ có thể đi vào trung tâm, nếu là đen thì đi vào ô [9,5], nếu là đỏ thì chỉ có thể đi vào [2,5].

Còn nếu ở giữa thì vị trí hiện tại đi chéo theo bốn hướng, cả 2 đều đi các hướng là:[currRow+1, currCol +1], [currRow+1, currCol − 1], [currRow − 1, currCol − 1], [currRow − 1, currCol +1].

- Hàm **possibleMovesForKing(currRow, currCol, boardStates):** trả về các nước di chuyển hợp lệ của quân tướng.

Code được tối ưu để có thể cài cho cả hai phía: client và sever.

Cho dù là quân tướng nào thì cũng chỉ di chuyển trong cột từ 4 → 6 do đó với vị trí hiện tại của hàng thì ta sẽ thêm các nước đi có col từ 4 tới 6 và hàng là hàng hiện tại.

Đối với quân có currRow <=5 thì là quân đỏ do đó cần push thêm các nước row chạy từ 1 → 3 với vị trí cột là currCol.

Tương tự push thêm các ô có tọa độ Row từ 8 → 10 đối với quân đen và vị trí hiện tại là cột currCol.

Lọc các bước có thể đi được bằng filter trong typescript, hàm lọc này tính khoảng cách từ vị trí hiện tại tới vị trí đi mới của vua nếu có bình phương khoảng cách < 2 thì là nước đi đúng còn lại đều sai.

- Hàm **possibleMovesForXiang(currRow, currCol, boardStates, isLowerTeam):** trả về các nước di chuyển hợp lệ của quân tượng.

Được cài đặt cho cả 2 loại quân đen và quân đỏ. Dùng biến để kiểm soát quân tượng có thể di chuyển lên hoặc xuống so với vị trí hiện tại.

Quân tượng bất kỳ ở cả 2 phía có thể đi về phía trước nếu có currRow<=3 hoặc là quân đen. Quân tượng có thể đi xuống nếu là quân đỏ và có currRow <=8.

- Hàm **possibleMovesForZu(currRow, currCol, boardStates, isLowerTeam):** trả về các bước di chuyển cho quân chốt.

Cài đặt cho cả quân đen và quân đỏ. Kiểm tra xem quân chốt đã qua sông hay chưa dựa trên loại quân và vị trí row hiện tại.

- beyond = true nếu quân đã qua sông, ngược lại là false.
- Nếu là true thì có thể di chuyển thêm 2 bước qua trái hoặc qua phải.
- Hàm trả về bước di chuyển hợp lệ.

Mục tiêu chính của *rule.ts* trả về các bước di chuyển hợp lệ của tất cả quân của 1 phe bất kỳ (đỏ hoặc đen). Hàm **possibleMoves = function(piece: Piece, boardStates: {}, isLowerTeam)** nhận vào quân cờ piece, bàn cờ hiện tại và quân của team nào (đen hoặc đỏ).

  - Quân cờ piece là class quân Piece lấy tên quân địch piece.name sẽ lấy được quân cờ và kèm theo số hiệu quân, các quân trong bàn cờ.

Ví dụ: chốt có 5 quân chốt thì name sẽ là z1, z2, z3, z4, z5 do đó tách tên sẽ gọi piece.name[0].

  - Tách lấy vị trí hiện tại piece.position sẽ là cặp toạ độ x, y.
  - Sau khi lấy được tên và tọa độ ta sẽ gọi hàm tương ứng để tìm các nước đi hợp lệ.
  - Tuy nhiên trong lúc tìm các nước đi hợp lệ ta vẫn chưa kiểm tra quân cờ có ra khỏi bàn cờ hay chưa nên ta sẽ filter các quân này lọc ra quân hợp lệ

di chuyển trong bàn cờ bằng cách gọi hàm filterBoundMoves(currRow, currCol,moves, boardStates).

- Hàm trả về nước di chuyển hợp lý của piece.

4.5. Điều khiển các chức năng bắt sự kiện

Trong các file html sử dụng để biểu diễn game cờ tướng trên nền web, thì file app.component.main.html như một khuôn mẫu tập hợp các thành phần từ các file html khác. File này được truy xuất từ class *app.component.main.ts*. Trong đó có một thẻ quan trọng thể hiện bàn cờ là <board>, thẻ này dẫn đến class board nằm trong file *board.ts* được biểu diễn trên nền web thông qua board.html và board.css.

Có 2 hàm phổ biến được sử dụng trong html do angular hỗ trợ, đó là *ngIf và *ngFor tương tự lệnh điều kiện và vòng lặp được sử dụng như sau:

- *ngIf : <thẻ_html *ngIf **điều kiện** > Thực thi nội dung thẻ </thẻ_html>

Một khi điều kiện được thỏa mãn thì *thẻ_html* mới thực thi nội dung. Hàm ngIf này cũng có thể gọi đến hàm của 1 class bất kì trong các file khác.

- *ngFor: <thẻ_html *ngFor let **a** of **b** > Thực thi nội dung thẻ </thẻ_html>

Với b là một mảng, a là một biến tương ứng với từng phần tử trong mảng b được chạy từ đầu mảng đến cuối mảng, tạo thành một vòng lặp cho *thẻ_html*. Hàm ngFor này cũng có thể gọi đến hàm của 1 class bất kì trong các file khác.

Ngoài ra còn có các cách bắt sự kiện khác như *click* để bắt sự kiện click chuột, *change* để bắt sự kiện nội dung một thành vừa được thay đổi để thực thi hàm mà nó truy xuất tới (thường được sử dụng trong select để bắt sự kiện chọn một nội dung mới của thanh lựa chọn), sử dụng các thẻ *sm − select* để hiển thị danh sách lựa chọn … Các thẻ và các thuộc tính bắt sự kiện ở trên quá quen thuộc trong html, tuy nhiên, thay vì phải viết 1 hàm script để sau khi bắt sự kiện sẽ thực hiện hàm đó thì angular hỗ trợ ta có thể tham chiếu thẳng tới một hàm nào đó thuộc một class bất kỳ trong các file khác, bởi vì angular được viết hoàn toàn bằng type script.

Một trong những ưu điểm nổi bật angular dễ dàng thay đổi thuộc tính của một đối tượng, hoặc thêm thuộc tính. Ở đây ta tập trung việc set các giá trị cho 1 quân cờ trên bàn cờ. Một quân cờ cần quan tâm việc vị trí tọa độ của nó trên bàn cờ. Angular có ngStyle chúng ta muốn thay đổi vị trí của 1 quân cờ trên bàn cờ dễ dàng :

[style.left] = x + 'px', [style.bottom] = y + 'px'.

Ở đây sẽ set lại vị trí của một đối tượng lại về tọa độ x,y theo đơn vị px.

Sản phẩm của nhóm có các biểu diễn như sau:

- App hỗ trợ các button như button chuyển loại cờ tướng sang cờ úp, button giải cờ thế sử dụng FormControl trong angular để bắt tín hiệu điều khiển và thực hiện các hàm theo mong muốn.
- Các button undo, redo, switchturn bắt sự kiện khi click vào button, và thực hiện hàm.
- ngForm() là kiểu form đặc biệt trong angular giúp tạo form bắt sự kiện khi bấm submit một thế cờ bất kì.
- Đồng hồ đo thời gian đơn vị tính đến mili giây sử dụng setinterval() để tính time, nếu hết time sẽ end game, khi switchturn thì thời gian 1 team sẽ dừng lại và tính thời gian team kia.
- Chọn các thuật toán để chơi với máy 0 − Greedy, 1 − ABpruning, 2 − MCTS, thanh chọn bắt được giá trị ô bằng cách lấy $event.target.value để lấy giá trị ô đã chọn và sử dụng hàm parse_agentType để lấy giá trị của loại chơi đó. vd parse_agentType(desc) { desc.split(‘ − ’)[0]}

4.6. Quản lý bàn cờ

- Quản lý bàn cờ bắt các sự kiện được cài đặt trong *client/app/compoent_board/board.ts.*
- **constructor(server: ComputeService):** khởi tạo bàn cờ, ta tạo một server để trao đổi dữ liệu giữa server và client.
- **ngOninit():**
    - initGame(): khởi tạo bàn cờ.

  - initDumypiece() đây là mặc định lúc chạy bàn cờ khởi tạo trước cờ tướng thường.

- ChangeMode: chuyển loại cờ từ cờ tướng sang cờ úp sử dụng hàm **changeMove(),** lúc bắt sự kiện thì ta sẽ thay đổi this.reverse = !this.reverse và khởi tạo lại bàn cờ. Mặc định là đánh cờ tướng thường nên reverse là false và đồng thời clear các kết quả các biến đã sử dụng trước bằng hàm clear_result().
- **newState(red,black)** hàm nhận vào một mảng Piece của red (quân đỏ) và black (quân đen) khởi tạo 2 Agent với tập hợp các quân red và black mặc định việc giải cờ thế là thuật toán Alpha Beta puring với độ sâu là 4.
- **SolveState(f:ngForm)** hàm nhận vào một ngForm là nội dung người dùng nhập vào form và submit thì ta sẽ chuyển nội dung submit ra thành thế cờ, nội dung submit người dùng là một chuỗi các quân được phân biệt bởi dấu ‘,’ và một quân gồm có các thuộc tính sau tên quân, vị trí, quân phe nào (1 là đỏ, − 1 là quân đen).

Ví dụ: “k 9 4 − 1,k 1 5 1,” ở đây ta khởi tạo tướng của đen và tướng quân đỏ. Yêu cầu việc nhập thế cờ phải nhập đúng theo cấu trúc và bắt buộc 2 phe đều có tướng. Việc nhập khá mất công sức vì hiện tại chưa hỗ trợ việc sắp xếp các quân cờ do thời gian quá hạn chế.

- ChangeType chuyển từ đánh cờ sang việc giải cờ thế, chỉ có giải cờ thế cho cờ tướng thường. Quản lý việc giải cờ thế: biến StateFlag = true đang giải cờ thế, flalse là đánh cờ thường, bàn cờ hiện tại sẽ là this.InputCurrentState khởi tạo bàn cờ với hai quân đỏ và quân đen (this.InputBlack và this.InputRed) bằng cách gọi **newState(this.InputBlack, this.InputRed).**
- Hàm **CheckTypeChess()** hỗ trợ việc kiểm tra loại cờ đang chơi, bổ trợ cho các hàm ngIf trong *compoent_main.html.*
- Hàm **swithTurn()** khá là đơn giản các công việc cần làm trong hàm như chuyển người chơi, **this.state.swithTurn(),** chuyển tính giờ cho quân khác, và dừng tính giờ cho quân k chơi, kiểm tra quân địch có bị chiếu tướng hay không, và cuối cùng giao tiếp với server chuyển trạng thái cho server để phân tích bước đi tiếp theo.

- Hàm **SuportSwitchTurn()** hàm hỗ trợ việc nhường cho đối thủ đi trước, hoặc là chấp đối thủ đi trước trong một vài nước tuỳ ý muốn người chơi. Hàm chỉ gọi **this.switchTurn().**
- Hàm **go2PreviousState()** hàm này chính là nút undo chuyển về trạng thái trước trạng thái đang đứng 2 lần. Nói cách khác lùi state về một nước của mình và địch. Lưu lại redo state hiện tại, cần cập nhật state mới bằng state cũ trước đó trong lastState, xoá lastState đi phần tử cuối.
- Hàm **redo()** đẩy bước di chuyển của redo vào laststate, vì laststate hiện tại sẽ là state khi redo xong, cập nhật state mới là this.state = this.redo, this.redo hiện tại sẽ là null, ở đây ta hỗ trợ việc redo khi chúng ta undo, nếu có undo nhiều lần thì redo cũng chỉ về 1 trạng thái cuối undo.
- **CheckReverse()** hàm trả về loại cờ đang chơi là cờ úp hay không để hỗ trợ cho *component_main.html*
- **CheckMove(current_piece)** kiểm tra một quân có di chuyển hay không để biết ta in ra tên thật hay tên giả, riêng với quân tướng thì sẽ in tên thật kể cả cờ úp hay cờ tướng.
- **TimeMode()** hàm này bật/tắt chế độ tính giờ, và khởi tạo lại bàn cờ mới nếu muốn chơi chế độ tính giờ.
- **hiddenTimer()** trả về true/flase để ẩn/hiện chế độ time nếu timemode sẽ hiện, ngược lại ẩn đi.
- **inputTime()** nhận vào thời gian nhờ ngForm để thay đổi time trong chế độ timemode, kiểm tra việc nhập time có đúng hay không, nếu nhập time là một số âm hoặc là 0 thì mặc định time là 10 phút.
- **StartTimer(team)** tính thời gian cho team hiện tại, nếu hết thời gian thì sẽ xuất ra kết quả thua cho team hết giờ.
- **PauseTimer(team)** dừng tính time đối với tên team.
- **initDummyButtons()** khởi tạo các quân giả trên bàn cờ, ta sẽ in ra nó nếu nước đi bất kì hợp lệ thì sẽ hiện quân giả lên, đây chính là một nước đi hợp lệ của quân ta đang selected. Khởi tạo DummyButtons là tất các các ô trên bàn cờ.

- **parse_agent_Type(desc):** hàm này nhận một chuỗi là 0 − Greedy, 1 − ABpruning hoặc 2 − MCTS. Chúng ta tách nó ra theo dấu ‘ − ’ và lấy phần tử đầu là số 0, 1 hoặc 2 để biết loại cờ người ta muốn chơi là gì.
- **chooseBlackAgent(desc)** nhận vào một chuỗi là 0 − Greedy, 1 − ABpruning hoặc 2 − MCTS sau đó gọi hàm parse_agent_Type(desc) để giải nén nó ra và xem đó là loại cờ gì, lưu lại loại cờ vào biến blackAgentType, sau đó khởi tạo bàn cờ hàm **initGame().**
- Hàm **parseInt()** là hàm có sẵn của typescript nhận vào một chuỗi và chuyển về số.
- **chooseBlackAgentDepth(desc)** nhận vào một chuỗi là số cho biết độ sâu(1 → 4) hoặc là số nút maximum khi duyệt MCTS(độ sâu từ 1000 → 100000) sử dụng hàm **parseInit(desc)** lưu lại độ sâu biến blackAgentDept.
- **HumanMove(piece: Piece)** nhận vào một quân cờ và di chuyển quân cờ này copy lại trạng thái để dùng sau, di chuyển quân cờ tạo nên state mới, khởi tạo redo là null, hàm humanMove là cập nhật bước di chuyển của người chơi do đó chỉ cập nhật **redAgent.movePieceTo(SelectedPiece,piece.position)** sau đó chuyển turn cho server chơi gọi hàm **this.switchTurn().**
- **end_game(end_state)** có 3 trạng thái end_state là − 1 thua, 1 thắng, 0 là hoà. Hàm sẽ cập nhật quân thắng, in kết quả, dừng tính giờ.
- **setCheckMate(value)** : kiểm tra xem Agent bất kì có bị chiếu hay không, sau đó đặt giá trị cho this.checkmate = value.

4.7. Cài đặt thuật toán Greedy

Mỗi quân cờ sẽ có một trọng số khác nhau. Quân “King” là quân quan trọng nhất và trọng số của nó lớn hơn rất nhiều so với tổng trọng số của tất cả các quân còn lại. Trọng số của một quân “Rook” tương đương với một “Horse” cộng với một “Cannon”. Trọng số của “Horse” tương đương với hai quân “Elephant”. Dựa vào heuristic trên, có được bảng giá trị sau (được thể hiện ở file *Evaluation.ts*):

Piece	King	Assistant	Elephant	Rook	Horse	Cannon	Pawn
Value	6000	120	120	600	270	285	30

Bảng 4.1: Trọng số của các quân cờ

Ta đi vào phần cài đặt thuật toán. Xét tất cả vị trí đi hợp lệ tiếp theo của các quân cờ ở trạng thái đang xét. Nếu tại vị trí nào có quân cờ (chỉ có thể là quân cờ của địch) thì ánh xạ trọng số của quân cờ đó, nếu không, ánh xạ giá trị 0.

```
var moves = this.get_moves_arr();
var values = moves.map(x =>
this.getValueOfGreedyMove(x[0], x[1]));
 – – – – – – – – – – – – –
getValueOfGreedyMove(pieceName, toPos) {
     var piece = this.boardState[toPos.toString()];
     if (piece) return Evaluation.pieceValue(piece[0]);
     return 0;
}
```

Nước đi tiếp theo sẽ là đi đến vị trí ánh xạ giá trị lớn nhất. Nếu tất cả vị trí đều ánh xạ giá trị 0, nước đi tiếp theo sẽ được đi một cách ngẫu nhiên đến một trong các vị trí hợp lệ theo hàm **random_move().**

4.8. Cài đặt thuật toán Alpha − Beta Pruning

Thuật toán này được cài đặt trong file *server\Strategy\ABPruning\ABPruning.ts*

Hàm **recurseEvaluation**: nhận vào trạng thái của bàn cờ, độ sâu được chọn, Alpha, Beta

- Đối với thuật toán AB − Pruning sẽ có một quân là Max và quân đối thủ là Min.

```
var isMax = state.playingTeam == state.redAgent.team;
```

sẽ xác định quân Max là quân đỏ.

- Để cài đặt thuật toán ta cần:
    - Xác định trạng thái hiện tại là của quân Max hay quân Min

```
var playingAgent: Agent = state.get_playing_agent();
```

    - Cập nhật bàn cờ

```
playingAgent.updateBoardState();
```

    - Kiểm tra có phải trạng thái kết thúc hay không

```
var endState = state.getEndState();
if (endState != 0) {
// return game score for red agent
    return [state.playingTeam * endState * Infinity,
null];
}
```

- Sau khi xác định được các yếu tố trên ta cần tạo mảng moves lấy tất cả các nước có thể di chuyển trên bàn cờ (trả về tên và vị trí của quân cờ). Mỗi nước di chuyển sẽ tạo ra các trạng thái khác nhau của bàn cờ. Như đã giới thiệu ở trên, thuật toán Alpha − Beta cần tạo cây trò chơi bắt đầu từ trạng thái đầu tiên của bàn cờ cho đến trạng thái kết thúc. Mỗi node của cây sẽ là các nước di chuyển hợp lệ của quân đỏ hoặc quân đen. Do đó sau khi có mảng moves ta cho vòng for chạy trong mảng moves để xét từng nước di chuyển của các quân trên bàn cờ, tạo cây trò chơi bằng cách đệ quy từng trạng thái kế tiếp từ nước đang xét với độ sâu giảm dần sau mỗi lần gọi.

```
var eval_result = [this.recurseEvaluation(nextState, depth
- 1, Alpha,  Beta)[0], [movePieceName, move]];
```

- Nếu deth == 0 thì sẽ `return [this.getValueOfState(state), null];`

Do khi độ sâu về 0 nghĩa là cây đã được xây dựng đến node lá nên ta cần tính giá trị tại mỗi trạng thái của node lá để xét xem nước cờ nào sẽ mang lại lợi thế cho sever (quân đen).

getValueOfState(state) là hàm sẽ trả về giá trị của trạng thái hiện tại cho quân đỏ, được cài đặt bằng cách lấy tổng trọng số của quân đỏ − tổng trọng số của quân đen. Trọng số của mỗi quân cờ được tính bằng tổng giá trị của quân cờ và giá trị của vị trí nơi quân cờ đang đứng. Các giá trị này được cài đặt trong file *Evaluation.ts.*

- Để lưu lại các nước di chuyển trong mỗi trạng thái ta sử dụng list next_evals được tổ chức theo cấu trúc [score, [movePieceName, toPos]]. Thêm các nước di chuyển này vào next_evals: next_evals.push(eval_result);
- Sau khi tạo cây tiến hành tính giá trị cho từng trạng thái của cây và cắt tỉa Alpha − Beta cho cây trò chơi. Nếu là tầng Max thì lấy giá trị max và cập nhật Alpha ngược lại thì lấy giá trị min và cập nhật Beta. Khi node có giá trị Alpha <= Beta thì cắt node đó.
- Hàm return về next_evals.

Hàm **computeNextMove**: nhận vào trạng thái hiện tại của bàn cờ

- Gọi hàm recurseEvaluation để tính nước đi tiếp theo cho sever.

```
var evalResult = this.recurseEvaluation(curr_state,
this.DEPTH, - Infinity, Infinity);
```

- Hàm return về tên quân cờ và vị trí di chuyển của quân cờ, đây chính là nước đi tiếp theo của quân đen. Nếu `evalResult[0] * this.team == - Infinity` hàm sẽ return về null.

4.9. Cài đặt thuật toán MCTS

Thuật toán này được cài đặt trong *server\Strategy\MCTS*

4.9.1. Ý tưởng ước lượng giá trị cho một node

Ở mục 2.4.3 có đề cập tới việc ước lượng một giá trị cho node *j* để áp dụng cho công thức UCB. Có 2 vấn đề nhóm xét tới khi đánh giá giá trị ước lượng:

- Thứ nhất: mỗi trạng thái bàn cờ chúng ta coi như là một node của một cây trò chơi. Bất kỳ thế nào chúng ta vẫn có thể đánh giá được trọng số của bàn cờ đó, nói cách khác chúng ta sử dụng một cách đánh giá heuristic để đánh trọng số cho thế cờ đó. Tuỳ vào mục đích của người cài đặt có thể thêm bớt để giảm mức độ quan trọng của một node hoặc tăng mức độ quan trọng của một node nào đó.
- Thứ hai: đối với 1 trạng thái bất kỳ của bàn cờ, quan sát dưới góc nhìn của một người biết chơi cờ thì đối với một thế cờ đánh giá tốt nếu các quân cờ ở vị trí "khai cục" thuận lợi giúp việc tấn công lẫn phòng thủ được tốt. Nhưng đối với góc nhìn là máy tính chúng ta sẽ cài đặt như thế nào để cân bằng việc tấn công lẫn phòng thủ?

Phân tích hàm heuristic cài đặt:

$$UCB\_value = \frac{sum\_score}{n} + c\sqrt{\frac{ln(N)}{\text{n}}} \quad (4.1)$$

- Theo công thức (2.1) thì $\overline{X_j}$ là phần quyết định cho việc khám phá một thế cờ. Mục đích của việc khám phá khá giống mở rộng cây theo depth first search, tuy nhiên như đã nói, MCTS chỉ muốn mở rộng ở những nước đi có triển vọng. Vì vậy đối với những node mà tại đó nếu là nước đi của quân đối phương đang có lợi thì ta cần tránh khu vực cây đó, ngược lại nếu nước đi của quân mình có lợi thì cần đi khám phá.
- Đối với một bàn cờ nhất định luôn có 2 phe đỏ và đen. Trên bàn cờ thì mỗi quân cờ bất kì có một trọng số nhất định và ở một vị trí bất kì thì chúng cũng có một trọng số [19]. Như vậy trọng số của một phe trên bàn cờ sẽ bằng tổng trọng số tất cả các quân hiện có. Trọng số của node hay state hiện tại sẽ tùy thuộc vào phe đang muốn xét trọng số là phe nào. Khi phe đang xét là phe đỏ thì trọng số của state bằng trọng số phe đỏ trừ trọng số phe đen, nếu là phe đen, chỉ cần lấy giá trị âm của giá trị trên. Vậy nếu một state đang có lợi cho phe đỏ thì trọng số đối với phe đỏ sẽ cao và phe đen sẽ thấp và ngược lại.
- sum_score của một node sẽ bằng tổng các trọng số của child node. Lấy sum_score chia cho số lần visit của node ta sẽ được $\overline{X_j}$.

Tuy nhiên còn nhiều hạn chế, do thời gian ít và chưa nghiên cứu được sâu nên chưa phát huy được thế mạnh thuật toán, trong tương lai có sẽ cân bằng việc phòng thủ lẫn tấn công thuật toán bằng việc điều chỉnh hàm heuristic cho hợp lý.

4.9.2. MCTS_State

Mỗi node xây dựng khi chạy thuật toán MCTS được biểu diễn là một MCTS_State thay vì chỉ là một state như thông thường do ta cần quản lý thêm một vài biến:

- sum_score: kiểu number, được khởi tạo bằng 0: tổng giá trị ước lượng khả năng thắng của tất cả child node đã được duyệt của một node.
- visits: kiểu number, được khởi tạo bằng 0: tổng số lần đã đi qua của một node.
- parent: trỏ tới MCTS_State cha của node đang xét.
- children: một mảng các MCTS_State, mỗi phần tử của mảng trỏ tới một MCTS_State con của node đang xét.
- state: trỏ tới State tương ứng của node.
- EXP_RATE = 12000: giá trị c.
- parentMove: mảng chứa nước đi mà node cha đi để có child node tương ứng.

Một số hàm cài đặt:

- Hàm **get_ave_score():** trả về giá trị $\underline{X_j}$ dựa trên sum_score và visits.
- Hàm **UCB_valule():** tính và trả về giá trị UCB theo (4.1)
- Hàm **generate_children():** tạo mảng children tương ứng của một node đang xét:
    - Xác định agent team nào đang thực hiện nước đi và cập nhật state:
    - Tính toán các nước đi hợp lệ, mỗi phần tử của mảng các nước đi trả về là một mảng có dạng [‘<piecename>’, [<newposision>]]:
    - Với mỗi nước đi trong mảng moves tạo một state mới. State mới đó sẽ là một child node cho node đang xét:

4.9.3. Cài đặt MCTS

Quá trình tìm kiếm trên cây của MCTS sẽ được giới hạn trong số lượng simulations thực hiện (biến N_SIMULATION). Với mỗi lần simulation, 3 bước được gọi:

- **Select**: bắt đầu từ root đi xuống. Nếu là một node chưa được duyệt (visit = 0) nên trả về node. Còn lại xét child node có UCB_value lớn nhất và gọi đệ quy Select xuống đối với chide node đó tới khi gặp node chưa được duyệt trước đó và trả về node.
- **Simulate**: từ node được trả về ở bước trên, thực hiện một bước di chuyển greedy để đẩy mạnh việc đi các nước ăn được quân cờ của địch. Tạo một node mới với nước đi thực hiện và thêm node vào cây. Ước lượng khả năng thắng sum_score của node đó.
- **Back_propagate**: cập nhật tất cả parent node của node vừa mới simulated bằng việc tăng số lần visits và cộng thêm sum_score.

Ba bước trên sẽ được gọi nhiều lần nhất có thể trọng phạm vi giới hạn của số lần simulations để có thể tìm được nước đi tốt nhất có thể. Khi tới giới hạn vòng lặp, nước đi tốt nhất được trả về dựa trên node có UCB_value lớn nhất. Khi cài đặt MCTS, nhóm đã chọn giá trị cho tham số *c* bằng 2 lần value của quân tướng.

Nhóm quan sát được rằng: với giá trị *c* càng lớn thì máy càng đẩy mạnh việc tìm kiếm các nước đi tấn công quân địch hơn là phòng thủ và ngược lại *c* nhỏ thì máy nghiêng về phòng thủ hơn. Như vậy có thể thấy với một giá trị *c* thích hợp thì sẽ cần bằng được việc công và thủ tốt hơn, tăng tỷ lệ thắng hơn.

4.10. Cờ thế

- Giải cờ thế áp dụng thuật toán Alpha – Beta pruning với độ sâu mặc định là 4.
- Giải cờ thế ta nhận vào một thế cờ, ở đây thế cờ có dạng string mỗi ký tự cách nhau một dấu cách, giữa 2 quân cờ cách nhau dấu ‘,’ cuối string có dấu ','. Nhập quân cờ có mang tên quân cờ, vị trí quân cờ, quân cờ đen hay đỏ (− 1 hoặc 1).

Ví dụ: k 9 4 − 1,k 1 5 1

  - Thế này gồm 2 tướng của quân đen và đỏ
  - Tham khảo trong github có thế cờ cho sẵn
  - Hạn chế chưa hỗ trợ việc sắp xếp các quân cờ vì khả năng còn hạn chế và thời gian không cho phép.

- Giải cờ thế này do người và máy chơi tiếp thế cờ, công việc chủ yếu giải cờ thế là lấy được thế cờ từ người chơi sau đó tạo state mới từ thế cờ đó, cập nhập bàn cờ, chọn quân đi trước, sau đó đánh tiếp ván cờ. Trong tương lai sẽ phát triển được việc máy tự động đánh với nhau trọng việc giải cờ thế và từ đó có thể áp dụng một số thuật toán khác như TD learner để tối ưu cách chơi cờ.
- Button chuyển cờ thế bắt sự kiện chuyển thế cờ qua buton "Normal Chinese chess " hoặc "Reverse Chinese chess" bắt được sự kiện qua thư viện ngform trong angular, sau đó chuyển form này vào hàm **SloveState(f:ngform)** trong *board.ts* để giải nén thế cờ này.
- Bắt đầu một thế cờ ta đặt lại các giá trị chơ thời gian, sau đó giải thế cờ đã đưa vào, thế cờ đưa vào là một string dùng hàm split tách theo dấu ',' lấy được từng quân cờ, sau đó đưa các quân cờ đúng theo team của nó là đỏ hoặc đen, lưu vào InputBlack, InputRed và ta lưu lại cả bàn cờ là InputCurrentState.
- Khi người chơi submit thế cờ xong thì người chơi sẽ nhấn chuyển sang cờ thế, tại đây cập nhập các giá trị như reverse = false, StateFlag = !StateFlag biến này để đánh dấu chuyển qua dạng cờ thế, cập nhật boardstate = InputCurrentState, tạo một state mới bằng việc gọi hàm newState với tập hợp quân đen là InputBlack, quân đỏ là InputRed.
- Hàm newstate đã được nêu file *board.ts*, ở đây ta sẽ giải cờ thế với depth là 4 với thuật toán Alpha – Beta pruning.
- Tới đây ta đã tạo nên một state và từ đó chạy như thuật toán Alpha – Beta bình thường.

4.11. Cờ úp

- Cờ úp cài đặt trong file *server/strategy/Ultimate_algorthm.*
- Cờ cài đặt Alpha Beta pruning mặc định chạy với độ sâu 4.
- Ở cờ úp chỉ cần cài đặt thêm việc khoá các nước di chuyển làm lộ tên thật của quân cờ, nghĩa là quân cờ đang đứng chưa di chuyển bước nào thì ta chỉ cho nó di chuyển 1 lần, còn các quân cờ đã di chuyển ta sẽ cho nó di chuyển như thuật toán cho cờ tướng thường.
- Việc khoá các nước di chuyển này sử dụng 3 hàm chính:

    - AddElementToPastMove(name,count)
    - DeleteElement(name,count),
    - CheckFakeMove(name,count)
- Tất cả các hàm này được cài đặt trong *server/Strategy/Agent/Agent.ts*
- Tạo Past là một mảng, mỗi phần tử trong mảng là một mảng gồm 2 phần tử là quân cờ và số nước đã di chuyển.
- Hàm **AddElementToPastMove(name,count)** hàm nhận vào tên và số nước di chuyển, hàm thêm mảng [name,count] vào mảng Past
- Hàm **DeleteElement(name,count)** hàm xoá phần tử có tên là name và có số nước đi là count.
- Hàm **CheckFakeMove(name)** nhận vào tên name và kiểm tra xem phần tử [name,0] có trong Past hay không, nếu có thì quân name đã di chuyển 1 lần từ trạng thái chưa di chuyển lên trạng thái di chuyển và làm lộ tên thật, hàm kiểm tra có hay không phần tử [name,0].
- Cài đặt thuật toán tương tự như Alpha – Beta pruning trong file *server/Strategy/ABpruning* tuy nhiên ta sẽ kiểm tra mỗi quân trước khi di chuyển có hợp lệ hay không bằng hàm **CheckFakeMove(name)**, nếu kiểm tra có thể di chuyển thì ta thêm vào Past bằng hàm **AddElementToPastMove(name,count)**, và đệ quy để tìm kiếm tiếp phát triển các nước mới của bàn cờ, sau đó xoá phần tử có name và count ra PastMove, đây là một trong những bài đệ quy quay lui khá đơn giản và cơ bản trong lập trình.
- Và một điều quan trọng nữa là việc đếm số lần di chuyển của một quân, nếu như nước đi hợp lệ sau khi **CheckFakeMove(name)** thì sau đệ qui ta sẽ trừ 1 lần di chuyển đối với quân đó, dựa vào hàm **DecreaseMoveOfPiece(name)** nhận vào name và trừ đi 1 lần di chuyển đối với quân có tên name.

4.12. Ghi biên bản trận đấu và biểu diễn biểu đồ tỉ lệ thắng thua trận đấu

Biên bản trận đấu được ghi theo chuẩn System2 của WXF [1].

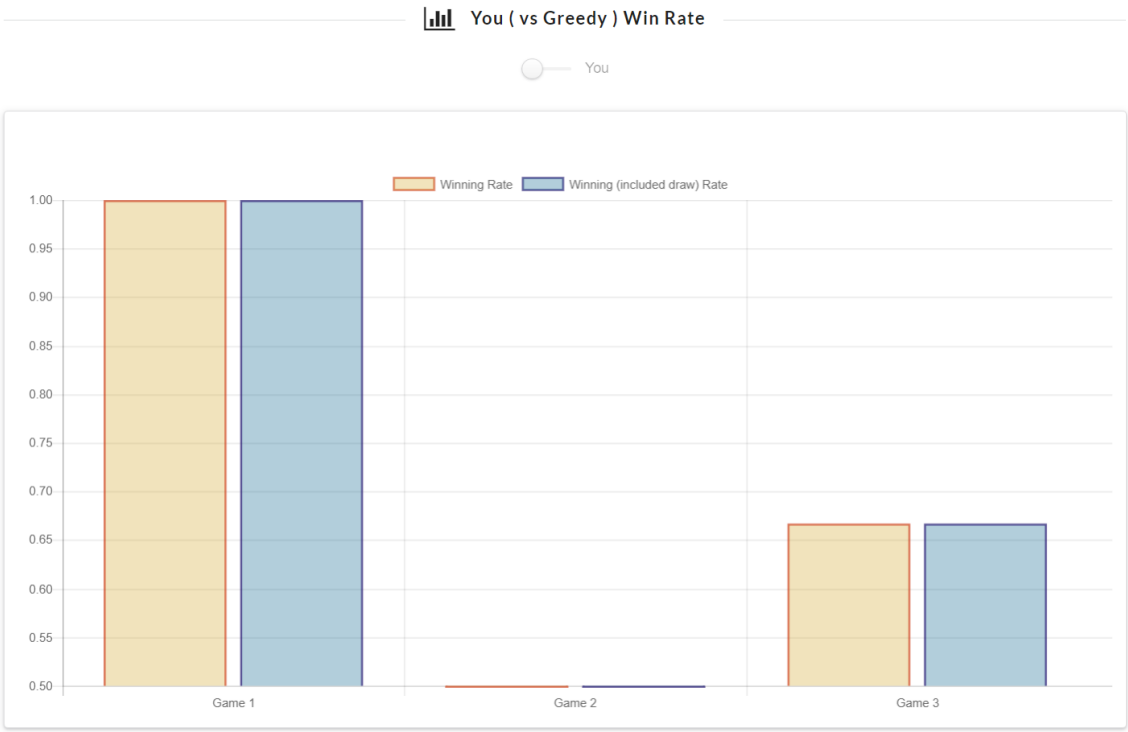
Một khi trận đấu được xác định là đã đi tới hồi kết, bên component board sẽ khởi tạo ra hai event bắt đầu việc hiển thị biên bản và biểu đồ. Sau khi event được khởi tạo, bên

html main sẽ thực hiện event được gọi trong tag <board> là **(onResultsUpdated)** cho phép hiện thị biểu đồ và **(onRecordsUpdated)** cho phép hiển thị biên bản.

Khi event **(onResultsUpdated)** được thực hiện, component main sẽ thực hiện hàm **update_result** với 2 tham số truyền vào là kết quả của ván đấu và mảng chứa thông tin [agent type, agent depth]. Hàm **update_result** sẽ thực hiện hàm **update** của component child là **WinRaterComponent**.

Để thể hiện biểu đồ cho đồ án, nhóm đã sử dụng hỗ trợ ChartsModule của ng2 − charts.

Biểu đồ thể hiện là một biểu đồ cột, thể hiện xác suất thắng của người chơi qua các ván đấu. Tên biểu đồ được thể hiện dựa trên mảng thông tin [agent type, agent depth] nhận được. Kết quả trận đấu nhận được sẽ được và xử lý để thể hiện thành 2 cột tỷ lệ thắng (không bao gồm ván hòa) và tỷ lệ thắng bao gồm ván hòa. Chức năng có phép xem tỷ lệ thắng của đối phương hữu ích khi muốn so sánh 2 agent máy đấu với nhau hoặc 2 người chơi với nhau trên nhiều ván đấu. Biểu đồ sẽ được cập nhật liên tục sau khi kết thúc mỗi ván đấu.



Tương tự với event **(onRecordsUpdated)**, component main sẽ thực hiện hàm **update_record** với 2 tham số nhận vào là 2 mảng chứa các nước đi đã thực hiện của 2 agent đỏ và xanh. Nước đi được kí hiệu theo system2:

[chữ cái viết tắt của quân cờ trong tiếng anh][cột hiện tại][dấu thể hiện hướng di chuyển][cột mới/ số hàng đi được (trong trường hợp di chuyển trên đường thẳng)]

Dấu " + " thể hiện hướng đi tiến về phía địch, dấu " − " thể hiện hướng đi lùi về phía quân mình và dấu ". " thể hiện hướng di chuyển theo hàng ngang.

Ví dụ: C2.5: quân pháo đi theo hàng ngang từ cột 2 sang cột 5.

H8+7: quân mã đi tiến từ cột 8 tới cột 7.

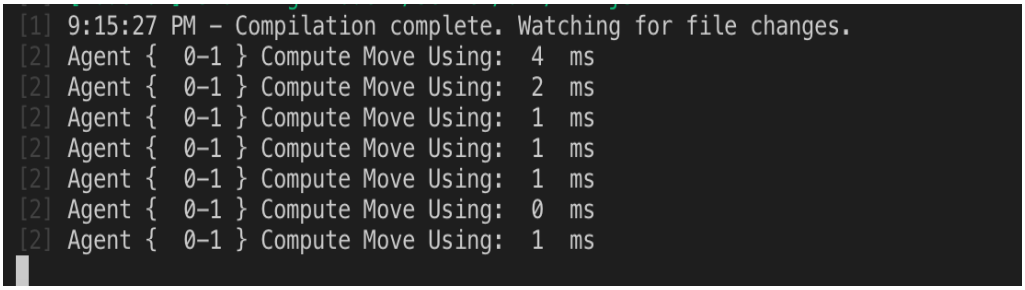
Đối với cờ úp, nếu là bước di chuyển quân chưa lật thì chữ cái viết tắt sẽ là tên vị trí quân cờ đang úp đứng nhưng sau khi lật lên, nước cờ sẽ ghi lại theo tên thật.

Ví dụ: nếu quân chưa được lật ở vị trí quân xe tại cột 1 hàng 1 đi tới hàng 2 thì ghi là R1+1. Sau khi được lật lên là quân mã, di chuyển quân tới cột 2 thì ghi lại theo tên thật là H1+2.

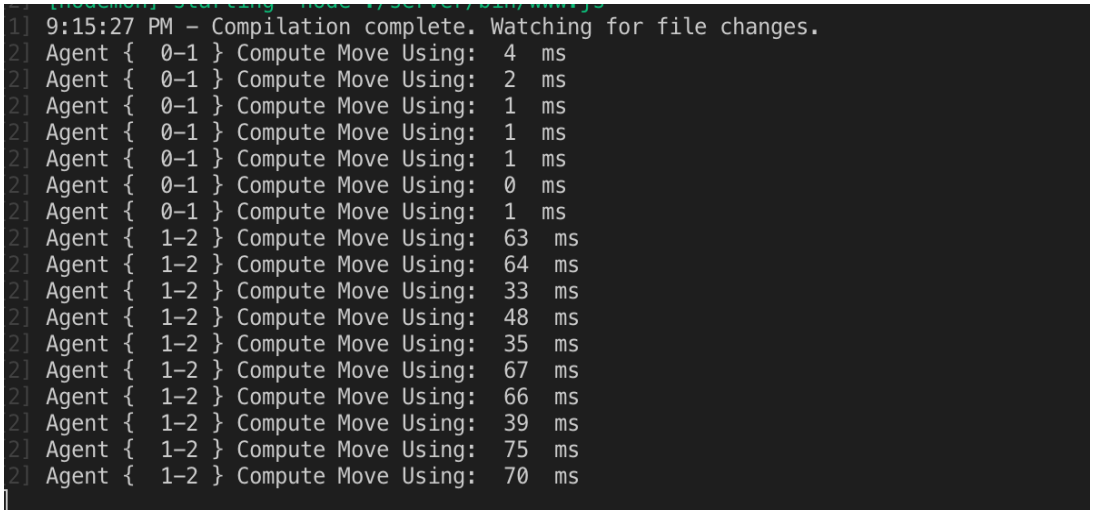
CHƯƠNG 5: KẾT QUẢ THỰC NGHIỆM

Thuật toán		Đánh giá	
Greedy		Đánh giá thuật toán chạy nhanh tuy nhiên đánh rất kém.	Hình 5.1
AB prunning	Depth = 2	Thuật toán chạy khá nhanh, tốc độ ổn khoảng 75ms biết bố trí được quan cờ hợp lý, di chuyển tránh né để không bị mất quân.	Hình 5.2
	Depth = 3	Thời gian chạy một nước cờ chưa tới 1s, tốc độ ổn định, tránh né được sự tấn công, biết bảo toàn được quân tuy nhiên đánh chưa tốt.	Hình 5.3
	Depth = 4	Khoảng thời gian chạy lâu khoảng 12ms đối với những nước đầu về sau quân số càng ít thì thời gian càng nhanh, tránh né và tấn công khá tốt tuy nhiên ở độ sâu này	Hình 5.4

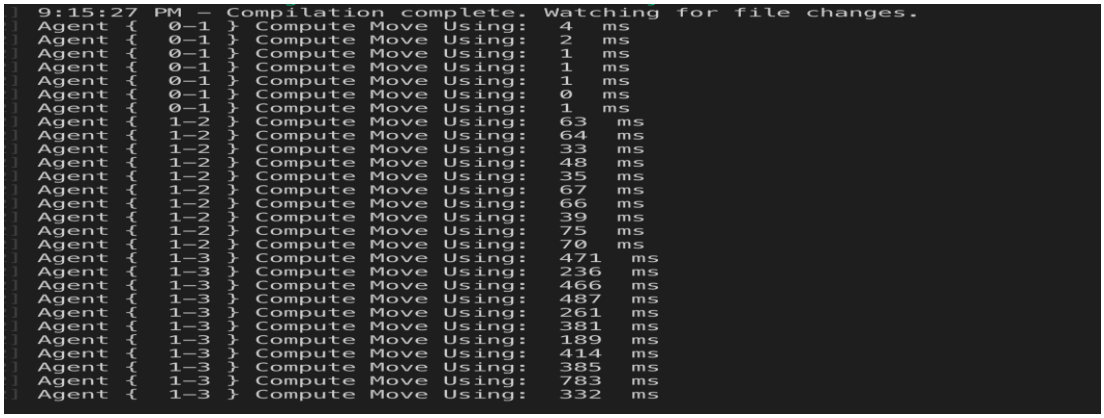
		thì chỉ chơi cờ ở mức trung bình.	
MCTS	Simulation 3000 node	Tốc độ khoảng 1s chạy nhanh, đánh chưa được tốt.	Hình 5.5
	Simulation 5000 node	Tốc độ ổn định chạy nhanh tuy nhiên mức độ chơi vẫn không bằng alpha beta pruning.	Hình 5.6



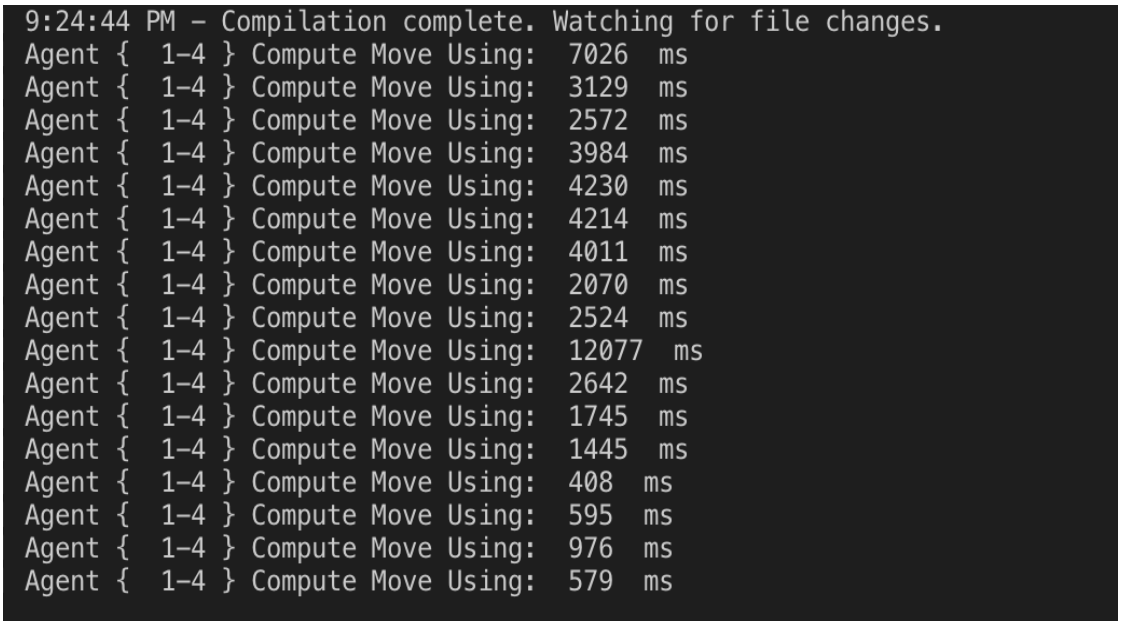
Hình 5. 1



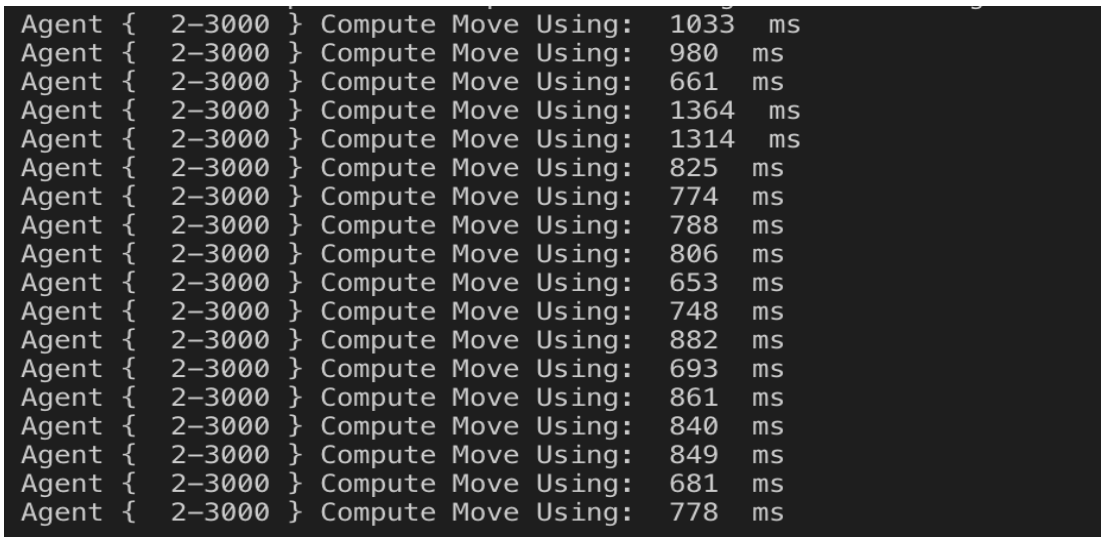
Hình 5. 2



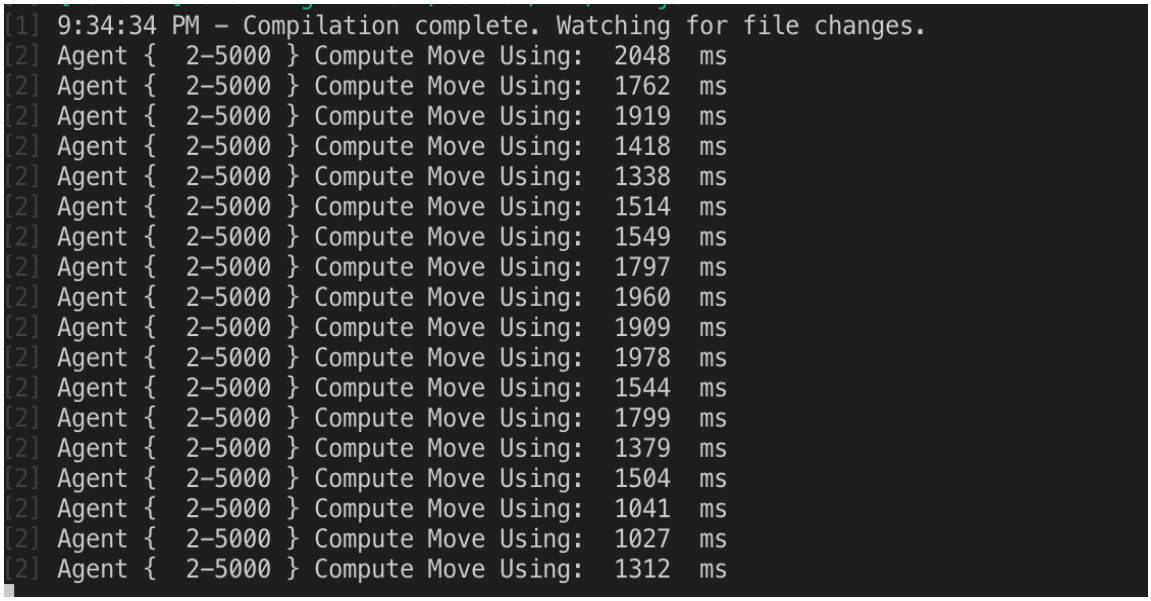
Hình 5. 3



Hình 5. 4



Hình 5. 5


[1] 9:34:34 PM - Compilation complete. Watching for file changes.
[2] Agent { 2-5000 } Compute Move Using: 2048 ms
[2] Agent { 2-5000 } Compute Move Using: 1762 ms
[2] Agent { 2-5000 } Compute Move Using: 1919 ms
[2] Agent { 2-5000 } Compute Move Using: 1418 ms
[2] Agent { 2-5000 } Compute Move Using: 1338 ms
[2] Agent { 2-5000 } Compute Move Using: 1514 ms
[2] Agent { 2-5000 } Compute Move Using: 1549 ms
[2] Agent { 2-5000 } Compute Move Using: 1797 ms
[2] Agent { 2-5000 } Compute Move Using: 1960 ms
[2] Agent { 2-5000 } Compute Move Using: 1909 ms
[2] Agent { 2-5000 } Compute Move Using: 1978 ms
[2] Agent { 2-5000 } Compute Move Using: 1544 ms
[2] Agent { 2-5000 } Compute Move Using: 1799 ms
[2] Agent { 2-5000 } Compute Move Using: 1379 ms
[2] Agent { 2-5000 } Compute Move Using: 1504 ms
[2] Agent { 2-5000 } Compute Move Using: 1041 ms
[2] Agent { 2-5000 } Compute Move Using: 1027 ms
[2] Agent { 2-5000 } Compute Move Using: 1312 ms

Hình 5. 6

CHƯƠNG 6: TỔNG KẾT

6.1. Thuận lợi và khó khăn trong quá trình tạo nên sản phẩm "trò chơi đánh cờ tướng"

6.1.1. Thuận lợi

- Toàn bộ các thành viên trong nhóm có tinh thần trách nhiệm cao, luôn hoàn thành tốt nhiệm vụ được giao và đúng hẹn.
- Framework Angular hỗ trợ nhiều phương thức hiệu quả và đơn giản để sử dụng.
- Luật cờ tướng khá đơn giản để tự tìm hiểu và xây dựng bộ luật cho chúng.
- Cờ tướng là tựa game kinh điển thế nên cũng có rất nhiều source code để tham khảo về thuật toán cũng như về giao diện hiển thị.
- Angular có sự hỗ trợ nodemon giúp vừa code vừa chạy kiểm tra được xem code có lỗi hay không.

6.1.2. Khó khăn

- App được viết bằng angular là một framework mới của google nên khá ít tài liệu.
- Typescript cú pháp ổn định cấu trúc chặt chẽ tuy nhiên tốc độ khá chậm so với các ngôn ngữ khác, ảnh hưởng tới độ sâu khi duyệt nên thuật toán chỉ chạy được ở mức trung bình.
- Cần phải tìm hiểu nhiều về kiến thức web, việc trao đổi dữ liệu trên web.
- Debug khó khăn không được hỗ trợ nhiều như các ngôn ngữ khác.
- Quản lý team gặp khó khăn, mọi người chưa sử dụng thành thạo github, chưa tiếp xúc với framework angular, chưa làm việc chuẩn gitflow.
- Chưa xử lý được trường hợp khó gặp như việc 3 con chốt cùng trên một cột nên ghi biên bản cho trường hợp còn sai sót.

6.2. Nhận xét

Mục tiêu muốn đạt được khoảng 80%. Sản phẩm có khả năng các loại cờ tướng, cờ úp, giải cờ thế (người đánh với máy). Về cơ bản có các chức năng như undo, redo, nhập thế cờ, bật tắt thời gian, chọn thời gian đánh cờ.

Áp dụng được các thuật toán như greedy, alpha beta pruning, monte carlo tree search, có thể chọn được độ sâu của alpha beta pruning và chọn được số node của monte carlo tree search.

Tuy nhiên còn thiếu một số chức năng như chưa hỗ trợ được tự động giải cờ thế, tốc độ chương trình còn hạn chế cần giảm bớt những thành phần không cần thiết và tối ưu code.

Chức năng submit một thế cờ chưa hoàn thiện và gây bất tiện cho người chơi trong tương lai có thể cải tiến để các con cờ có thể di chuyển kéo vào vị trí thích hợp cho người dùng.

Chưa phát huy được sức mạnh của MCTS, chỉ cài đặt ở mức trung bình cần nghiên cứu kĩ đưa ra giải pháp tốt hơn.

Về cơ bản đã hoàn thiện được cấu trúc xây dựng được bàn cờ và có thể code được nhiều loại cờ dựa trên cấu trúc này chỉ cần thay đổi và code được luật chơi của loại cờ mình muốn và thay đổi UI loại cờ mình muốn.

Trong tương lai có thể áp dụng thử một số thuật toán khác như TD learner, alpha beta move reordering.

Nhìn chung đồ án hoàn thành tốt, các thành viên đã tiếp xúc sử dụng thành thạo github, làm quen với gitflow, làm quen với sử dụng một framework để tạo nên sản phẩm, biết được khó khăn trong làm việc team quản lý source code.

6.3. Hướng phát triển sản phẩm

- Cố gắng hoàn thiện sản phẩm, khắc phục những lỗi chưa sửa được.
- Cải thiện độ thông minh cho thuật toán của máy.
- Tìm hiểu thêm về các thuật toán khác sản phẩm có thể phù hợp với những người chơi cờ có trình độ cao.
- Tìm hiểu và áp dụng công thức, thuật toán tốt hơn với chế độ cờ úp.
- Thêm các chức năng mới như tự giải cờ thế, chế độ 2 người chơi

CHƯƠNG 7: TÀI LIỆU THAM KHẢO

[1] https://en.wikipedia.org/wiki/Xiangqi

[2] https://en.wikipedia.org/wiki/World_Xiangqi_Championship

[3] http://wxf.ca/wxf/index.php/xiangqi − news/news − from − asia/470 − regulations − of − the − 15th − wold − xiangqi − championship

[4] http://wxf.ca/wxf/index.php/xiangqi − news/news − from − asia/470 − regulations − of − the − 15th − wold − xiangqi − championship

[5] https://www.asianxiangqi.org/English/AXFrulesEng.htm

[6] https://en.wikipedia.org/wiki/Swiss − system_tournament

[7] http://doc.edu.vn/tai − lieu/luan − van − giai − thuat − tim − kiem − minimax − va − ung − dung − trong − cac − tro − choi − co − tong − bang − khong − 22012/

[8] https://123doc.org/document/4043555 − tim − kiem − co − doi − thu.htm

[9] https://vanbadesigner.blogspot.com/2017/10/ung − dung − giai − thuat − minimax − trong − tro.html

[10]http://doc.edu.vn/tai − lieu/luan − van − giai − thuat − tim − kiem − minimax − va − ung − dung − trong − cac − tro − choi − co − tong − bang − khong − 22012/

[11] https://en.wikipedia.org/wiki/Minimax

[12] https://en.wikipedia.org/wiki/Alpha%E2%80%93 Beta_pruning

[14] https://brilliant.org/wiki/greedy − algorithm/

[13] https://vi.wikipedia.org/wiki/Gi%E1%BA%A3i_thu%E1%BA%ADt_tham_lam

[15] https://angular − 2 − training − book.rangle.io/handout/directives/ng_if_directive.html

[16] COULOM, Rémi. Efficient selectivity and backup operators in Monte − Carlo tree search. In: International conference on computers and games. Springer, Berlin, Heidelberg, 2006. p. 72 − 83.

[17] FRAENKEL, Aviezri. Combinatorial games: selected bibliography with a succinct gourmet introduction. The Electronic Journal of Combinatorics, 2012, 1000: 2 − 9, 2012.

[18] SILVER, David, et al. Mastering the game of go without human knowledge. Nature, 2017, 550.7676: 354.

[19] Yen, S.J., Chen Jr, C., Yang, T.N. and Hsu, S.C., 2004. Computer chinese chess. ICGA Journal, 27(1), pp.3 − 18.

[20] https://viblo.asia/p/gioi − thieu − ve − angular − 2 − APqzearpzVe

[21] Browne CB, Powley E, Whitehouse D, Lucas SM, Cowling PI, Rohlfshagen P, Tavener S, Perez D, Samothrakis S, Colton S. A survey of monte carlo tree search methods. IEEE Transactions on Computational Intelligence and AI in games. 2012 Mar;4(1):1-10.