Báo cáo Đồ án môn học: Nhập môn Trí tuệ nhân tạo
ỨNG DỤNG CÁC KĨ THUẬT TRÍ TUỆ NHÂN TẠO TRONG CỜ VÂY - MINIGO

Nhóm sinh viên thực hiện: *Cao Như Đạt - 20194013, Nguyễn Huy Hoàng - 20194058*
Giảng viên hướng dẫn: *TS. Nguyễn Nhật Quang*

MỤC LỤC

Lời nói đầu

Cờ vây (tiếng anh là "go") có lịch sử rất lâu đời và khởi nguồn từ Trung Hoa cổ đại, là một trò chơi dạng chiến lược trừu tượng cho hai người chơi, trong đó mục tiêu là bao vây nhiều lãnh thổ hơn đối thủ.

Mục 1.1 sẽ nêu lên những luật chơi cơ bản của cờ vây thông thường, chúng em đã cài đặt và các phương pháp được trình bày trong phần phụ lục. Tuy nhiên, do sự hạn chế về thiết bị (khả năng tính toán của máy tính) , chúng em đề xuất một phiên bản đơn giản hơn của cờ vây, vẫn giữ được những đặc tính vốn có và chiến thuật cơ bản, tuy nhiên sẽ cắt giảm về không gian trò chơi - MiniGo. Các thuật toán chính được sử dụng sẽ là thuật toán tham lam, minimax và cắt tỉa alpha-beta.

Link google drive: AI - project - Google Drive

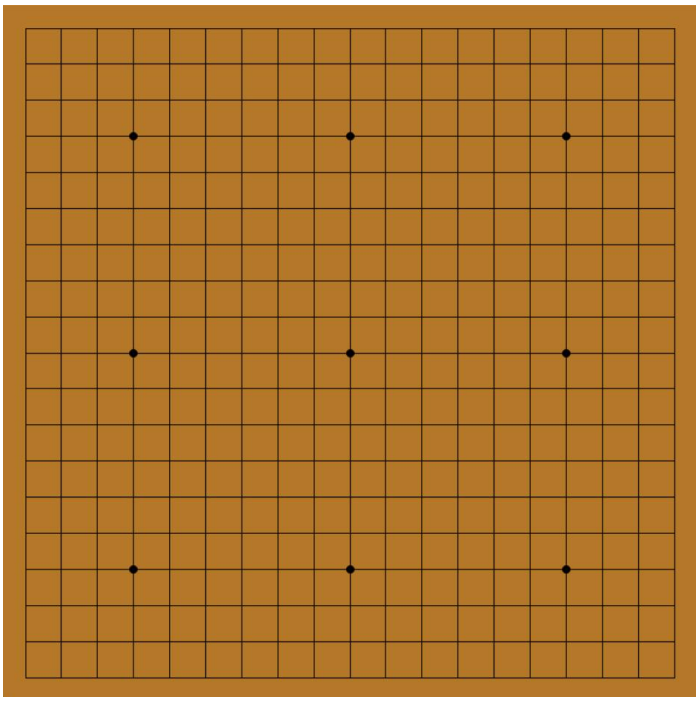
Link github: datcn1212/MINIGO-AI_PRJ: Project for AI subject - Game Go (github.com)

1. Giới thiệu bài toán: MiniGo

1.1 Luật chơi cờ vây

a, Bàn cờ

Bàn cờ vây hình vuông bao gồm 19 đường ngang dọc giao nhau tạo thành 361 điểm, trên bàn cờ có 9 chấm đen (gọi là *"sao"*) giúp người chơi dễ dàng hình dung và định vị bàn cờ, tuy nhiên không ảnh hưởng đến luật chơi. Cũng có một số bàn cờ với kích thước nhỏ hơn như 9x9, 13x13.

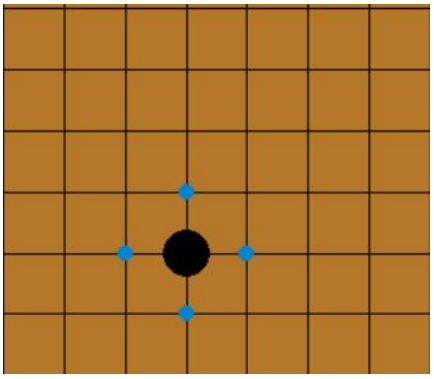


b, Quân cờ, nhóm quân và khí

Các quân cờ bao gồm 2 màu: đen và trắng, chúng được đặt xuống bàn cờ tại các giao điểm.

Các quân cờ khi đứng cạnh nhau (ngang, dọc) sẽ kết hợp với nhau tạo thành 1 nhóm quân.

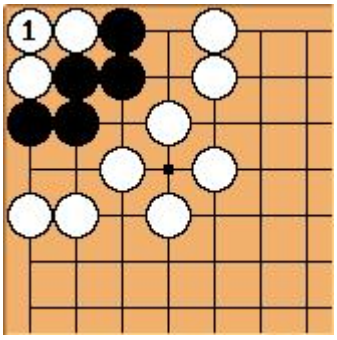
Các giao điểm trống cạnh một quân hay một nhóm quân (trên, dưới, trái, phải) được gọi là *"khí"* (4 chấm xanh ở hình dưới là các khí của quân đen). Các khí được bao vây bởi một nhóm quân được gọi là *"mắt"*.



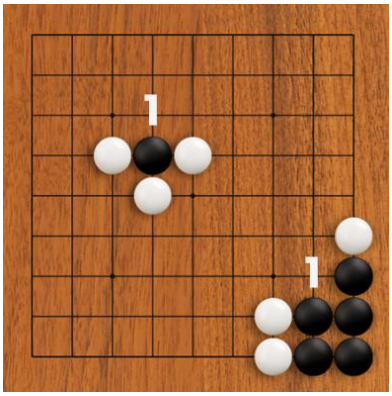
c, Các nguyên tắc cơ bản

Cờ vây chơi theo lượt, mỗi người đặt 1 quân. Lưu ý rằng người cầm quân đen luôn là người đi trước.

Ở mỗi lượt, người chơi cần đặt quân vào vị trí có khí và không được đặt quân tại vị trí làm mất hết khí của một nhóm quân của mình (*"tự sát"*, ở hình dưới, nếu quân trắng đánh vào vị trí 1 thì sẽ chặn hết khí của nhóm quân trắng ở góc, cho nên nước đi này không hợp lệ). Hơn nữa, các quân cờ trên bàn không bao giờ được lặp lại một vị trí của quân cờ đã có ngay trước đó. Các nước đi tạo ra kết quả như vậy bị cấm, và do đó chỉ các nước đi ở vị trí khác trên bàn cờ được cho phép trong lượt đi đó.



Một quân hay một nhóm quân sẽ bị rút ra khỏi bàn cờ nếu như chúng hết khí và trở thành tù binh. Như vậy, nếu muốn tồn tại trên bàn cờ, một quân hay một nhóm quân phải duy trì ít nhất một khí. Ở hình dưới, nếu quân trắng đánh vào một trong 2 vị trí mang số 1 thì quân (nhóm quân) đen tương ứng sẽ trở thành tù binh và bị đưa ra khỏi bàn cờ.



Một người chơi có thể bỏ lượt dựa trên việc xác định rằng ván cờ này không còn có cơ hội đi được nước cờ nào để giành lợi thế về bản thân. Ván cờ kết thúc khi cả hai người chơi đều bỏ lượt, và sau đó được tính điểm. Đối với mỗi người chơi, số lượng quân cờ bị bắt được trừ vào số điểm kiểm soát (bao quanh) được trong các *"khí"* hoặc *"mắt"* và người chơi có điểm số cao hơn sẽ thắng trận đấu. Các ván cờ cũng có thể thắng bằng việc một đấu thủ nhận thua.

1.2 Phiên bản cờ vây đơn giản: MiniGo

Như đã trình bày trong phần *"Lời nói đầu"*, mục 1.2 sẽ phát biểu luật chơi đơn giản của phiên bản rút gọn: MiniGo. Cụ thể, phần 1.1.a và 1.1.b (Bàn cờ, quân cờ, nhóm quân và khí) sẽ được giữ nguyên; phần 1.1.c sẽ thay đổi hai điều như sau:

- Tại mỗi lượt chơi, người chơi sẽ không đặt quân tại một vị trí bất kì tồn tại khí mà sẽ đặt vào những nước đi được gọi là *"nước đi hợp lệ" (legal action)*:

    +, Nếu có một hay nhiều quân (nhóm quân) đối phương chỉ còn một khí (gọi là *"nhóm nguy hiểm"*), nước đi hợp lệ sẽ gồm những khí đó, thực hiện nước đi để giành chiến thắng.

    +, Nếu không, nếu tồn tại nhóm quân đồng minh là nhóm nguy hiểm, nước đi hợp lệ là nước đi vào những khí đó.

    +, Cuối cùng, nếu không có nhóm quân nào nguy hiểm cho cả hai bên, nước đi hợp lệ sẽ là những nước đi chiếm khí của đối phương.

    Lưu ý: những nước đi *"tự sát"* sẽ không là nước đi hợp lệ!

- Trò chơi sẽ kết thúc khi có một quân (nhóm quân) bị chiếm làm tù binh hoặc một bên không còn nước đi hợp lệ nào, bên còn lại sẽ giành chiến thắng

Việc thay đổi một chút về luật chơi sẽ làm không gian tìm kiếm của trò chơi rút gọn lại, chiến thuật không thay đổi nhiều, tuy nhiên lại thuận tiện cho việc triển khai và đánh giá các phương pháp được sử dụng.

2. Các phương pháp sử dụng

Để giải quyết bài toán, chúng em đã sử dụng một số phương pháp như: *random (chọn ngẫu nhiên), greedy (tham lam), minimax và giải thuật cắt tỉa alpha-beta.*

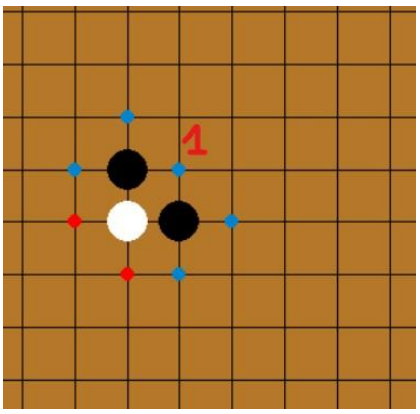
2.1 Random

Một phương pháp nghĩ đến đầu tiên trong các bài toán là *random*. Thuật toán cho phép chọn nước đi tiếp theo ngẫu nhiên trong số các nước đi hợp lệ *(legal actions)*. Tuy không có độ tốt về mặt giải thuật, tuy nhiên lại có ích trong việc tạo ra nhiều các trạng thái game khác nhau!

2.2 Greedy

Đúng với tên gọi, *thuật toán tham lam* sẽ chọn nước đi sao cho nó chiếm đi nhiều khí của đối phương nhất. Cụ thể, ta sẽ xét toàn bộ khí của các nhóm quân, duyệt qua chúng và tính xem vị trí

của khí nào là khí chung của nhiều nhóm quân nhất. Xét hình dưới, quân trắng sử dụng *greedy* sẽ đánh vào vị trí số 1 vì nó chiếm khí của chung 2 quân đen. Các nước còn lại chỉ chiếm khí của 1 quân đen.



2.3 Minimax & Cắt tỉa Alpha-beta

Khi nhắc đến các trò chơi đối kháng, chúng ta phải nhắc đến *Minimax* và phiên bản nâng cấp của nó là *Cắt tỉa Alpha-beta*. Tuy nhiên, trước khi bắt đầu chúng ta phải xác định được các yếu tố sau:

- Trạng thái bắt đầu *(initial state)* của trò chơi

- Hàm chuyển trạng thái *(successor function)* gồm các nước đi hợp lệ và trạng thái mới của trò chơi

- Điều kiện dừng *(terminal condition)* của trò chơi

- Hàm tiện tích hay hàm định giá *(evaluation function)* để đánh giá các trạng thái kết thúc

Trạng thái bắt đầu của trò chơi là khi bàn cờ trống và quân đen sẽ đi nước đầu tiên. Để tiện cho việc biểu diễn, trong mã nguồn chúng em đã cho quân đen đi nước đầu tiên luôn ở vị trí chính giữa của bàn cờ. Các nước đi hợp lệ và điều kiện dừng của trò chơi được trình bày trong mục 1.2.

Như vậy, việc còn lại là xác định xem hàm định giá sẽ được tính toán như thế nào!

a, Xác định hàm định giá (evaluatation funtion)

Hàm định giá lấy tham số đầu vào là quân đi nước tiếp theo (ta giả sử là quân Đen) được xác định như sau:

Thiết lập biến *score_win = 1000 - số nước đã đi của cả hai bên*

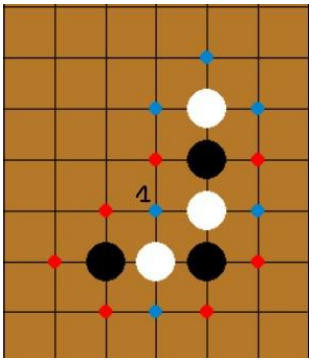
- Tại trạng thái game kết thúc: Đen thắng, trả về *score_win*, ngược lại trả về - *score_win* nếu Trắng thắng. Như vậy trò chơi ưu tiên số điểm cao hơn cho những ván cờ có kết thúc sớm hơn.

- Tiếp theo, ta xét đến trạng thái game bao gồm những nhóm quân nguy hiểm (những nhóm quân chỉ còn duy nhất một khí):

+, Nếu đối phương (Trắng) có ít nhất 1 nhóm quân nguy hiểm: Đen sẽ đánh vào khí tại đó để chiến thắng, hàm trả về giá trị *score_win - 1.*

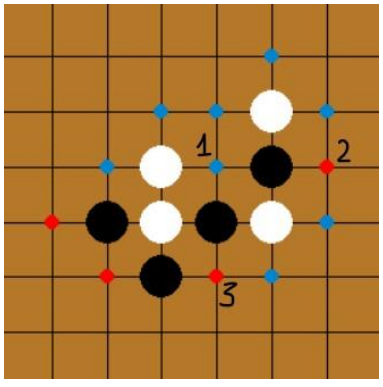
+, Nếu bên Trắng không có nhóm quân nguy hiểm, hơn nữa bên Đen có nhiều hơn 1 nhóm quân nguy hiểm, khi đó Đen chắc chắn thua ở nước đi tiếp theo của Trắng, hàm trả về *-(score_win - 1).*

- Nếu không rơi vào một trong hai trường hợp trên, ta xét đến trạng thái chứa khí nguy hiểm (khí nguy hiểm: khí chung của 2 nhóm quân, mỗi nhóm quân đang chứa đúng 1 khí - vị trí số 1 ở hình bên là khí chung của 2 quân Trắng, mỗi quân có đúng 2 khí, nếu Đen đánh vào thì tạo thành 2 nhóm quân nguy hiểm cho đối phương)



+, Nếu bên Trắng có chứa khí nguy hiểm, Đen sẽ thắng ở nước sau đó bằng cách di chuyển vào khí nguy hiểm, hàm trả về giá trị *score_win - 1*.

+, Nếu bên Trắng không có khí nguy hiểm, bên Đen có chứa khí nguy hiểm (chú ý giả sử Đen là người đi kế tiếp). Ta thực hiện duyệt qua tất cả khí nguy hiểm của bên Đen, chỉ cần 1 khí nào đó không "cứu được" thì hàm trả về *-(score_win - 1)*. Một khí nguy hiểm gọi là "cứu được" nếu như nó hoặc 2 khí còn lại của nhóm chứa nó là khí của một nhóm quân đối phương.



Xét hình bên, Đen đang đi, vị trí khí tại 1 là khí nguy hiểm, tuy nhiên vẫn cứu được vì Đen có thể di chuyển vào 1 (do 1 cũng là khí của một quân Trắng) và khi đó 1 không còn là khí nguy hiểm nữa.

+, Cuối cùng, nếu tất cả những khả năng trên đều không xảy ra, ta thực hiện tính điểm cho các nhóm quân và khí: điểm cho nhóm *(score_groups)* bằng hiệu của số nhóm đối phương chứa 2 khí và số nhóm đồng minh chứa 2 khí; điểm cho khí *(score_liberties)* bằng hiệu số khí

chung đối phương trừ đi số khí chung đồng minh (số khí chung = tổng số khí - tổng số nhóm). Cách tính như trên suy ra từ việc các nhóm đồng minh có 2 khí và các khí chung có khả năng tạo ra bất lợi (khí nguy hiểm). Điểm cuối cùng của trạng thái là:

*score_groups * normal(1, 0.1) + score_liberties * normal(1, 0.1)*

Trong đó *normal (1, 0.1)* sẽ sinh số ngẫu nhiên với phân phối chuẩn có kì vọng 1 và phương sai 0.1.

b, Cài đặt thuật toán minimax và cắt tỉa alpha-beta

Sau khi xác định được hàm định giá thì việc cài đặt thuật toán minimax và cắt tỉa alpha-beta còn lại khá đơn giản. Trước tiên là các bước cài đặt của minimax:

1. Hàm nhận vào màu của quân đang xét và giá trị *depth* (độ dài cây tìm kiếm sẽ là *depth* - độ dài đường đi từ nút gốc tới nút lá). Trả về giá trị của hàm định giá *(evaluation function)* nếu gặp trạng thái dừng hoặc đã đạt tới nút lá *(depth == self.depth).* Nếu không trả về hành động tương ứng với giá trị của hàm *max_value* tại trạng thái bàn cờ hiện tại và *depth = 0.*

```
def get_action(self, board):

    if len(board._get_legal_actions()) == 1:
        return board._get_legal_actions()[0]

    score, actions = self.max_value(board, 0)

    return actions[0] if len(actions) > 0 else None
```

2. Xây dựng hàm *max_value* với thông số đầu vào là trạng thái bàn cờ hiện tại và *depth.* Hàm này trả về hành động và ứng với số điểm cao nhất trong các *min_value* ứng với đầu vào là trạng thái con và *depth+1.*

```
def max_value(self, board, depth):

    if self.terminal_test(board) or depth == self.depth:
        return self.eval_func(board, self.color), []

    max_score = float("-inf")
    max_score_actions = []

    legal_actions = board.get_legal_actions()

    for action in legal_actions:
        score, actions = self.min_value(board.generate_successor_state(action), depth+1)
        if score > max_score:
            max_score = score
            max_score_actions = [action]

    return max_score, max_score_actions
```

3. Xây dựng hàm *min_value* tương tự hàm *max_value*, chỉ khác là lấy hành động và tương ứng với số điểm nhỏ nhất trong các *max_value*.

```
def min_value(self, board, depth):

    if self.terminal_test(board) or depth == self.depth:
        return self.eval_func(board, self.color), []

    min_score = float("inf")
    min_score_actions = []

    legal_actions = board.get_legal_actions()

    for action in legal_actions:
        score, actions = self.max_value(board.generate_successor_state(action), depth+1)
        if score < min_score:
            min_score = score
            min_score_actions = [action]

    return min_score, min_score_actions
```

Thuật toán cắt tỉa alpha-beta sẽ nhận thêm hai tham số đầu vào là alpha và beta tương ứng với giới hạn dưới và giới hạn trên của lời giải. Thực hiện việc cắt tỉa nếu tại hàm *max_value*, tồn tại giá trị *max_score > beta*, nếu không, cập nhật *alpha = max_score*. Ngược lại, đối với hàm *min_value*, cắt tỉa nếu *min_score < alpha*, nếu không cập nhật *beta = min_score*.

```
if max_score > beta:
    return max_score, max_score_actions  #prunning

if max_score > alpha:
    alpha = max_score
```

```
if min_score < alpha:
    return min_score, min_score_actions  #prunning

if min_score < beta:
    beta = min_score
```

Ưu điểm của thuật toán cắt tỉa alpha-beta là tốc độ nhanh hơn minimax do hạn chế không gian tìm kiếm. Cả minimax và cắt tỉa alpha-beta đều sử dụng thêm tri thức của bài toán: nếu tập các nước đi hợp lệ chỉ gồm 1 phần tử thì thực hiện di chuyển luôn mà không cần duyệt theo cây tìm kiếm.

3. Đánh giá các phương pháp sử dụng

Để đánh giá độ tốt của các phương pháp, chúng em đã thực hiện cho hai máy tự chơi với nhau trong 100 trận và không có GUI, mỗi máy sẽ là một agent. Bảng dưới hiển thị kết quả tỉ lệ thắng của *self_agent* và thời gian trung bình mỗi ván cờ , lưu ý rằng *self_agent* sẽ luôn là quân đen.

Self_agent	Random	Greedy	Alpha-beta (depth = 3)	Minimax (depth = 3)	Alpha-beta (depth = 4)	Minimax (depth=4)
Random	64%, 0.002s	34%, 0.002s	16%, 1.5s	17%, 4.6s	1%, 5.8s	0%, 19s
Greedy	65%, 0.001s	52%, 0.001s	58%, 0.7s	60%, 1.1s	5%, 1.3s	4%, 13.9s
Alpha-beta (depth = 3)	99%, 1.1s	80%, 3.3s	100%, 0.12s	100%, 0.14s	25%, 2.7s	27%, 13s

Từ kết quả bảng trên, ta có thể rút ra một số nhận xét như sau:

- Độ tốt của các thuật toán theo thứ tự tăng dần: *Random, Greedy, Minimax, Cắt tỉa Alpha-beta*

- *Random* không tốt trong kết quả nhưng lại gây khó khăn cho các ván cờ về mặt thời gian do chúng tạo ra nhiều phiên bản của trạng thái bàn cờ

- Đối với *Minimax* và *Cắt tỉa Alpha-beta*, khi tăng độ sâu thì hiệu quả sẽ tốt hơn.

- Cắt tỉa *Alpha-beta* có thời gian chạy nhanh hơn đáng kể so với *Minimax* do chúng có không gian tìm kiếm nhỏ hơn nhiều.

- Đối với cùng 1 thuật toán, bên đen sẽ có lợi thế hơn. Điều này đã được các chuyên gia cờ vây khẳng định!

4. Những khó khăn gặp phải và hướng giải quyết trong quá trình triển khai

4.1. Khó khăn về trò chơi

Mặc dù cờ vây đã có từ lâu đời nhưng để chơi thực tế thì lại không quá phổ biến ở Việt Nam. Không giống một số loại cờ như cờ vua, cờ tướng, cờ caro, … đã rất phổ biến, đây là lần đầu tiên chúng em tìm hiểu kĩ về luật chơi và nghiên cứu về cờ vây. Do đó rất khó để áp dụng nhiều kinh nghiệm sâu sắc về chơi cờ vào bài toán. Dù vậy, chúng em cũng vừa đọc tài liệu, vừa chơi thử để nâng cao kinh nghiệm và đồng thời lập trình để cải tiến song song.

4.2 Khó khăn về tài liệu

Tài liệu về lập trình cờ vây trên mạng khá ít ỏi. Khi khảo sát trên mạng cũng chỉ có vài mã nguồn về cờ vây cơ bản, tuy nhiên lại chỉ cung cấp giao diện và một số chức năng cơ bản và luật chơi cũng

không hoàn thiện. Ngoài ra không giống như cờ vua có thư viện chess, cờ vây cũng không hề có thư viện nào hỗ trợ. Đây là một điều khó khăn khi chúng em phải xây dựng và hoàn thiện chương trình với khá ít sự tham khảo.

4.2 Khó khăn về lập trình

Lập trình GUI và đảm bảo luật chơi: Do ít có nguồn tham khảo và hoàn chỉnh, lại là những người lần đầu lập trình pygame nên bọn em đã mất khá nhiều thời gian để tìm hiểu và hoàn thiện.

Lập trình cho các agent: Số nước đi của cờ vây rất là lớn, máy tính không đủ khả năng để tính toán nhanh chóng được. Bọn em đã thử lập trình với bàn cờ đầy đủ và luật chơi chặt chẽ nhưng máy tính không thể đáp ứng được và nó không có tính thực tiễn về mặt tính toán.

Giải quyết: Sau khi lập trình thử với bàn cờ hoàn chỉnh và luật chơi chặt chẽ, bọn em có đề xuất trò chơi MiniGo với luật chơi đơn giản hơn tuy nhiên vẫn giữ được một số tính chất của cờ vây và lại có tính thực tế hơn về tính toán.

5. Hướng phát triển trong tương lai

5.1. Cải thiện các phương pháp đã có

- Sử dụng một số kinh nghiệm và kĩ thuật để loại bỏ các nước đi ít giá trị.
- Tìm ra nhiều hàm lượng giá chất lượng khác cho các agent.
- Tối ưu hóa code để chương trình chạy nhanh hơn.
- Xây dựng các kĩ thuật phù hợp cho bài toán cờ vây hoàn chỉnh.

5.2. Sử dụng các kĩ thuật khác

- *Reinforcement Learning:* RL là một kĩ thuật rất phổ biến trong lập trình trò chơi và có khả năng áp dụng rất lớn vào bài toán, chúng em vẫn đang tiếp tục phát triển bài toán với việc sử dụng RL.

- *Machine Learning:* Trong tương lai khi có thể thu thập được nhiều dữ liệu hoàn toàn có thể áp dụng các kĩ thuật của Học máy để xây dựng lời giải cho bài toán.

6. Tài liệu tham khảo & tri thức có sẵn được khai thác

1. Các nguồn tìm hiểu về luật chơi cờ vây

Cờ vây – Wikipedia tiếng Việt

How to Play | British Go Association (britgo.org)

Hướng dẫn, luật chơi cờ vây cơ bản - Ziga (zigavn.com)

2. Các nguồn tìm hiểu về thuật toán sử dụng

Bài giảng Nhập môn Trí tuệ nhân tạo (IT3160) - Nguyễn Nhật Quang

Deep learning and the game of go.pdf - Google Drive

thesis_erikvanderwerf.pdf (tengen.nl)

lxucs/go-game-easy: A simplified version of Go game in Python, with AI agents built-in and GUI to play. (github.com)

3. Các nguồn tìm hiểu về cách cài đặt và triển khai mã nguồn

lxucs/go-game-easy: A simplified version of Go game in Python, with AI agents built-in and GUI to play. (github.com)

gogame/gogame.py at master · thomashikaru/gogame (github.com)

7. Phụ lục: Các phương pháp triển khai trong luật cờ vây hoàn chỉnh

7.1. Phiên bản cờ vây hoàn thiện

Đây là phiên bản gogam được bọn em xây dựng ban đầu với bàn cờ và luật chơi hoàn chỉnh.
- Sử dụng kinh nghiệm của bài toán trong tìm kiếm các nước đi hợp lệ:
- Ngoài việc nước đi phải thỏa mãn như phải đi vào ô trống, không đi vào ô mà không có khí hoặc không đi vào ô vừa bị ăn trước đó, bọn em thêm một điều kiện là không đi vào vùng bị bao vây. Khi một quân ở trong vùng bị quân khác bao vây mặc dù, quân đó vẫn còn khí nhưng chỉ cần đối phương đặt quân siết lại là quân đó sẽ bị ăn.

7.2 Random agent

Thuật toán random chọn ngẫu nhiên các nước đi trong tập các nước đi hợp lệ.

7.3 Greedy agent

Xuất phát từ mục đích của cờ vây là mở rộng tầm ảnh hưởng của quân mình và hạn chế tầm ảnh hưởng của quân đối phương, ta có 2 hàm lượng giá sau:

a, Tổng số khí của quân ta / Tổng số khí quân địch [1]: Giá trị hàm càng lớn chứng tỏ quân ta ảnh hưởng lớn và quân địch bị hạn chế

b, Xuất phát từ việc quân càng gần các nhóm của quân địch thì có tính uy hiếp càng cao nên ta có hàm sau:

$$Val(point) = \sum_{group\_i} len(group\_i)/dist(point, group\_i) \ [2]$$

group_i là nhóm quân thứ i của quân địch
len(group_i) là số quân trong nhóm quân thứ i
dist là khoảng cách gần nhất từ điểm point đến các quân ở trong *group_i*

Ở hàm trên ta có thể thấy: nhân với *len(group_i)* để thể hiện nhóm càng nhiều quân thì tầm ảnh hưởng càng nhiều; chia cho *dist* để thể hiện quân càng gần ảnh hưởng càng nhiều, hàm tính khoảng cách giữa 2 quân ta áp dụng khoảng cách Manhattan do tính chất khí của quân chỉ tính theo chiều ngang hoặc chiều dọc. Kết quả chạy thử với 20 ván cờ cũng cho thấy tỉ lệ thắng 19/20 của greedy dùng hàm lượng giá này với hàm phía trước.

7.4 Minimax_reduce

Chúng em cũng đã thử dùng minimax đầy đủ nhưng chạy mất quá nhiều thời gian và không có tính thực tế. Chúng em có đề xuất minimax_reduce khi mà chỉ xét những điểm hợp lệ và phải là khí của quân địch.
Chi tiết có trong *"gogame.rar"*