Báo cáo đồ án

**Cơ Sở Trí Tuệ Nhân Tạo**

**Logo

Description automatically generated**

Giảng viên hướng dẫn: Nguyễn Văn Quang Huy

Thực hiện: Trần Bảo Phú - 19127508

Trần Nguyên Trung - 19127607

Trần Khả Trí – 19127597

Lớp: 19CLC9

Đồ án: Cờ Lật

Khoa Công nghệ thông tin

Đại học Khoa học tự nhiên

Đại học quốc gia Thành phố Hồ Chí Minh

1. **GIỚI THIỆU**
   1. **Nội dung và yêu cầu đồ án**

Hãy viết chương trình AI tự động chơi Cờ lật biến thể sao cho chương trình càng khó bị đánh bại (bởi con người hoặc chương trình AI khác) càng tốt.

Các biến thể:

* Trước khi bắt đầu, trên bàn cờ sẽ có **5 ô** **đặc biệt** được chọn **ngẫu nhiên** tuỳ ý. Khi đó, ta gọi những ô này là ô chiến thắng.
* Người chơi nào chiếm được **toàn bộ** các **ô chiến thắng** thì sẽ giành **chiến thắng** ngay lập tức
* Nếu trò chơi kết thúc mà không ai chiếm được **toàn bộ các ô chiến thắng** thì ai chiếm **nhiều ô hơn** sẽ chiến thắng. Tuy nhiên, nếu cả hai chiếm được số ô bằng nhau thì ai chiếm được **nhiều ô chiến thắng hơn** sẽ chiến thắng, nếu không thoả thì được tính hoà
* Trò chơi không có khái niệm **bỏ lượt** mà chỉ có khái niệm đầu hàng.
  1. **Thông tin nhóm**

**1.2.1 Thông tin thành viên**

|  |  |  |  |
| --- | --- | --- | --- |
| **MSSV** | **Họ và tên** | **Email** | **Vai trò** |
| 19127508 | Trần Bảo Phú | 19127508@student.hcmus.edu.vn | PM, Developer, QC |
| 19127607 | Trần Nguyên Trung | 19127607@student.hcmus.edu.vn | Developer |
| 19127597 | Trần Khả Trí | 19127597@student.hcmus.edu.vn | Developer |

*Bảng thông tin thành viên nhóm*

**1.2.2 Phân công công việc**

|  |  |
| --- | --- |
| **Họ và tên** | **Công việc** |
| Trần Bảo Phú | Phân công các công việc, hướng dẫn. |
| Ý tưởng, cài đặt hàm heuristic |
|  |
| Trần Nguyên Trung | Thiết kế giao diện người dùng |
| Viết báo cáo |
|  |
| Trần Khả Trí | Ý tưởng, cài đặt hàm alpha-beta pruning |
| Ý tưởng, cài đặt hàm minimax |
|  |

*Bảng phân công các công việc cho các thành viên nhóm*

1. **Thiết kế đồ án**
   1. **Lý thuyết.**

* **Adversarial search** là tìm kiếm khi có một "kẻ thù" hoặc "đối thủ" thay đổi trạng thái của vấn đề từng bước theo hướng bạn không muốn.
* Đối thủ sẽ thay đổi trạng thái tiếp theo theo cách:

+ Không thể đoán trước

+ Gây bất lợi cho bạn (Thù địch)

* Bạn chỉ có thể giả sử trạng thái của đối thủ.

Ví dụ: Cờ vua, cờ tướng, cờ caro, …

Trong Adversarial search có 2 thuật toán tìm kiếm điển hình:

* Minimax:

+ Thuật toán Minimax cố gắng dự đoán hành vi của đối thủ.

+ Nó dự đoán đối thủ sẽ có hành động xấu nhất theo đánh giá.

+ Bạn là MAX - cố gắng tối đa hóa điểm số / chuyển sang trạng thái tốt nhất.

+ Đối thủ là MIN - cố gắng giảm thiểu điểm số của bạn / chuyển sang trạng thái tồi tệ nhất đối với bạn.

* Alpha-beta pruning:

+ Thuật toán alpha-beta pruniung là một cách cắt tỉa không gian.

+ Thay vì tìm kiếm toàn bộ không gian đến độ sâu cố định, chúng ta có thể loại bỏ các cành cây nếu chúng ta biết chúng sẽ không được chọn.

* 1. **Vận dụng.**

**- Pseudo code:**

+ Minimax decision:

Function: minimaxDecision() returns Move

moveList := moveGenerator(gameState)

for each move M in moveList do

value[M] := minimaxValue(applyMove(M, gameState), 1)

#with alpha-beta prunning algorithm use

# value[M] := minimaxValueAlphaBeta(applyMove(M, gameState), 1,MIN,MAX)

return M with the highest value[M]

+ MinimaxValue:

Function: minimaxValue(state, currentSearchDepth) returns a heuristic Value

if currentSearchDepth == maximumDepth or gameOver(state) then

return heuristic(state)

else

moveList := moveGenerator(state)

for each move M in moveList do

value[M] := minimaxValue(applyMove(M, state),

currentSearchDepth+1)

if whoseTurn == myTurn:

return max of value[]

else:

return min of value[]

+ minimaxValueAlphaBeta:

Function: minimaxValueAlphaBeta (state, originalTurn, depth, alpha, beta)

returns a heuristic Value

if currentSearchDepth == maximumDepth or gameOver(state) then

return heuristic(state)

else

moveList := moveGenerator(state)

if moveList == 0 then

tempState:=state

tempState.Turn:=state.Opponent()

return minimaxValueAlphaBeta(tempState, originalTurn,

searchPly + 1,alpha,beta);

else:

if originalTurn == state.Turn():

bestMoveVal = -99999

else:

bestMoveVal = 99999

for each move M in moveList do

tempState:=state

tempState.Turn:=state.Opponent()

val = minimaxValueAlphaBeta(tempState, originalTurn,

searchPly + 1,alpha,beta);

if originalTurn == state.Turn() then

if val > bestMoveVal then

bestMoveVal = val

if bestMoveVal >= alpha then

alpha = bestMoveVal

else:

if val < bestMoveVal then

bestMoveVal = val

if bestMoveVal <= beta then

beta = bestMoveVal

if alpha >= beta:

break

return bestMoveVal

**- Ý tưởng hàm heuristic:**

Hàm heuristic được tính toán dựa trên các tiêu chí xác định trạng thái của bảng, theo từng vị trí. Cụ thể xét mỗi ô chứa giá trị ‘W’ hoặc ‘B’ tại mỗi trạng thái, nếu:

+ Vị trí đó là gốc, điểm sẽ được tính là +20 : vì tại gốc sẽ không bị đối thủ tiêu diệt.

+ Vị trí đó là cạnh, điểm sẽ được tính +10 : vì cạnh chỉ có thể bị đối phương ăn từ 2 hướng.

+ Còn lại với mỗi một ô chiếm được, điểm sẽ là +1

+ Ngoài ra nếu ăn được vị trí victory cell, điểm sẽ là +20

Nếu bên nào ăn chiếm được tất cả 5 ô victory cell thì sẽ được điểm cao nhất: 99999 (giá trị mặc định)

Nếu lượt ‘W’ đi thì lấy (white\_score – black\_score) và ngược lại.

* 1. **Mở rộng**

**2.3.1 Ưu điểm của thuật toán hiện tại:**

**- Minimax:**

Tìm kiếm được mọi nước đi tiếp theo sau đó lựa chọn nước đi tốt nhất, vì giải thuật có tính chất vét cạn nên không bỏ soát trạng thái.

- **Alpha-beta prunning:**

Tối ưu giải thuật minimax, giảm thời gian chạy, tránh việc tìm kiếm những phần của cây mà không có bất kỳ giải pháp nào.

**2.3.2 Nhược điểm của thuật toán hiện tại:**

**- Minimax:**

Vấn đề lớn với thuật toán Minimax là nó có thể khám phá những phần không liên quan của cây, dẫn đến lãng phí tài nguyên.

**- Alpha-beta prunning:**

Vẫn còn phụ thuộc hoàn toàn vào độ sâu và hàm heuristic.

**2.3.3 Cải tiến:**

**- Thuật toán negascout:**

Nếu bạn có thứ tự nước đi tốt, bạn có thể tìm kiếm nước đi đầu tiên một cách

bình thường và sau đó chỉ cần dò tìm các nước đi còn lại của bạn.

**- Thuật toán NegaMax:**

Dựa trên nguyên tắc thực tế max(a,b) = -min(-a,-b) giúp đơn giản cho việc

thực hiện thuật toán MiniMax được dễ dàng hơn.

**Pseudo code:**

function minimax(node, depth, maximizingPlayer)

if depth = 0 or node is a terminal node

return the heuristic value of node

bestValue := -∞

foreach child of node

val := -negamax(child, other\_player)

bestValue := max( bestValue, val )

return bestValue

**- Reinforcement Learning: (hiện đại nhất)**

Là một loại học máy tập trung vào khả năng đưa ra một chuỗi các quyết định dựa trên các quan sát trong một môi trường nhất định. Không giống như các hình thức học máy khác được đào tạo dựa trên dữ liệu có sẵn, các thuật toán RL có thể điều hướng một không gian vấn đề bằng cách đánh giá các hành động tiềm năng. Do đó, RL có một lợi thế khác biệt khi chơi trò chơi vì mô hình có thể chọn nơi để tìm kiếm các cơ hội mới thay vì lấy ngẫu nhiên.

*\*Đó là những thuật toán, phương pháp mà nhóm mới tìm hiểu, chưa cài đặt, có thể sẽ tối ưu cho AI.*

1. **ĐÁNH GIÁ**
   1. **Đánh giá mức độ hoàn thành đồ án**

|  |  |
| --- | --- |
| **Yêu cầu** | **Mức độ hoàn thành** |
| Chạy được đồ án | 100% |
| Đánh cờ với bot online | 100% |
| Đánh cờ với AI khác | 100% |
| **TỔNG ĐỒ ÁN** | 100% |

*Bảng 3.1. Bảng đánh giá mức độ hành thành đồ án*

* 1. **Đánh giá thành viên trong nhóm**

|  |  |  |
| --- | --- | --- |
| **Họ và tên** | **Khối lượng công việc** | **Hoàn thành** |
| Trần Bảo Phú | 30% | 100% |
| Trần Nguyên Trung | 30% | 100% |
| Trần Khả Trí | 40% | 100% |

*Bảng 3.2. Bảng đánh giá thành viên trong* *nhóm*