ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
KHOA CÔNG NGHỆ THÔNG TIN

Đồ án 2: Wumpus World Agent

Môn học: Cơ Sở Trí Tuệ Nhân Tạo

Sinh viên thực hiện:
Huỳnh Ngọc Quốc - 23127112
Nguyễn Trần Thiên An - 23127315
Đinh Xuân Khương - 23127398
Nguyễn Đồng Thanh - 23127538

Giảng viên hướng dẫn:
Nguyễn Thanh Tình
Bùi Duy Đăng

Ngày 17 tháng 8 năm 2025

Mục lục

1 Tổng quan về đồ án

1.1 Giới thiệu

Những thập kỉ gần đây, Trí tuệ nhân tạo ngày càng tập trung vào việc tạo ra các agent thông minh có khả năng hoạt động trong môi trường không chắc chắn và phức tạp.
Bài toán Wumpus World là một ví dụ kinh điển và nổi tiếng về Agent dựa trên kiến thức. Trong đồ án lần này, chúng ta cần xây xưng một Agent thông minh. Trong đó Agent có thể cảm nhận (stench, breeze, glitter, bump, scream) để suy luận và hành động (di chuyển, quay, lấy, bắn, thoát). Mục tiêu là lấy vàng và thoát an toàn, tối ưu điểm số.
Nhưng có một thử thách mới, dự án mở rộng với nhiều Wumpus và các Wumpus này có khả chuyển động (yêu cầu nâng cao), thử thách Agent trong môi trường động và yêu cầu có những chiến lược thông minh hơn.

1.2 Mục tiêu

Mục tiêu chính là phát triển agent dựa trên kiến thức để lấy vàng và thoát với điểm số cao nhất trong cả môi trường tĩnh và động. Các mục tiêu cụ thể:

- Xây dựng không gian tri thức, có các thành phần như là breeze, stench, pit, wumpus, gold bằng logic mệnh đề, xử lý nhiều Wumpus và pit.
- Triển khai mô phỏng môi trường: Tạo bản đồ ngẫu nhiên, cung cấp cảm nhận, hiển thị một cách trực quan.
- Xây dựng engine suy luận để xác định ô an toàn/nguy hiểm.
- Module chiến lược tìm kiếm: Sử dụng A* hoặc Dijkstra để lập đường đi tối ưu, xem xét chi phí và rủi ro.
- Tích hợp agent hybrid: Kết hợp các module để agent hoạt động tự động, so sánh với agent ngẫu nhiên.
- Đánh giá: Thực nghiệm trên nhiều bản đồ, đo tỷ lệ thành công, điểm trung bình, hiệu quả quyết định, và ước lượng hoàn thành từng yêu cầu.

1.3 Công việc và hoàn thành

1.3.1 Phân chia công việc

MSSV	Họ và tên	Phân công	Hoàn thành
23127112	Huỳnh Ngọc Quốc	- Planning module - Hybrid Agent Intergration - Advanced setting - Report Writing	100%
23127315	Nguyễn Trần Thiên An	- Environment Simulator - Inference Engine - Report Writting - Video demo	100%
23127398	Đinh Xuân Khương	- Advanced setting - User-friendly function and Debug function - Report Writing and Proof-reading	100%
23127538	Nguyễn Đồng Thanh	- Random agent baseline - Experiment and conclusion - Report Writing and Proof-reading	100%

Bảng 1: Bảng phân chia công việc các thành viên trong nhóm

1.3.2 Hoàn thành

Công việc	Mức độ hoàn thành
Environment Simulator	100%
Inference Engine	100%
Planning Module	100%
Hybrid Agent Integration	100%
Random Agent Baseline	100%
Advanced Setting: Moving Wumpus Module	100%

Bảng 2: Bảng thống kê công việc và mức độ hoàn thành

2 Chi tiết thực hiện

2.1 Environment Simulator

Môi trường chứa Agent trong đồ án này được định nghĩa bởi một trục tọa độ Oxy với x, y là các số nguyên từ 1 đến N, với N là chiều dài x rộng của môi trường (mặc định là 8x8):
Ví dụ:

```
[#######] [#######] [#######] [#######] [#######] [#######] [#######] [#######]
[#######] [.......] [.......] [.......] [.......] [.......] [G,Gl   ] [#######]
[#######] [.......] [.......] [.......] [.......] [.......] [.......] [#######]
[#######] [.......] [.......] [.......] [.......] [.......] [.......] [#######]
[#######] [.......] [.......] [.......] [.......] [.......] [.......] [#######]
[#######] [.......] [.......] [.......] [.......] [.......] [.......] [#######]
[#######] [A>     ] [.......] [.......] [.......] [.......] [.......] [#######]
[#######] [#######] [#######] [#######] [#######] [#######] [#######] [#######]
```

Chú thích:

1. Trục thẳng đứng là trục y, trục ngang là trục x.
2. Vị trí của Agent (A): (y = 1, x = 1)
3. Các ô có kí tự '######' là tường, các ô có kí tự '**.....**' đại diện cho vị ô trống (không có Pit, Stench, hay các percepts khác)
4. Ô phía trên của Agent là (y = 2, x = 1) và cứ thế tăng dần theo y nếu Agent muốn di chuyển lên phía trên, ô bên phải của Agent là (y = 1, x = 2) và cứ thể tăng dần theo x nếu Agent muốn di chuyển về bên phải.

2.1.1 Mô tả Agent `Explorer`

`Explorer` là một tác nhân (*agent*) được kế thừa từ lớp `Agent`. Agent này được thiết kế để tham gia vào môi trường với nhiệm vụ khám phá và tìm vàng.

Thuộc tính (Attributes)

- **holding**: danh sách các vật thể mà agent đang giữ, mặc định là rỗng `[]`, khi Agent gặp một đối tượng là `Gold` thì Agent này sẽ bỏ vào trong **holding**.
- **has_arrow**: trạng thái cho biết agent có mũi tên hay không, mặc định là `True`.

- **killed_by**: ghi lại điều gì đã giết agent, mặc định là chuỗi rỗng.
- **direction**: hướng hiện tại của agent, mặc định là `Direction("right")`.
- **performance**: điểm số biểu diễn hiệu suất hoạt động, mặc định là `0`.
- **kb**: cơ sở tri thức (knowledge base) được agent sử dụng để suy luận.
- **visited**: tập hợp các ô đã được agent ghé qua, mặc định là $\{(1,1)\}$.
- **location**: vị trí hiện tại của agent trong môi trường, mặc định là $(1,1)$.

Hành vi của Agent

- Agent bắt đầu tại ô $(1,1)$, hướng về bên phải.
- Có khả năng suy luận dựa trên cơ sở tri thức (`kb`) để quyết định hành động.
- Có thể di chuyển, thay đổi hướng đi, nhặt vàng, bắn mũi tên, và bị tiêu diệt nếu gặp nguy hiểm, trèo và tẩu thoát.
- Hiệu suất (`performance`) thay đổi tùy theo hành động hoặc kết quả đạt được (ví dụ: nhặt vàng, bị giết, chi phí di chuyển v.v.).

Tóm lại: `Explorer` là một agent khám phá môi trường, mang theo cơ sở tri thức để ghi nhớ và suy luận, có khả năng nhặt vàng, bắn mũi tên, và theo dõi hiệu suất của mình trong quá trình hoạt động.

2.1.2 Mô tả Agent `Wumpus`

`Wumpus` là một lớp kế thừa từ `Agent`, đại diện cho quái vật Wumpus trong môi trường. Thông thường, đây là một thực thể thụ động, không tự hành động mà chỉ phản ứng khi khi có tác nhân đi vào vị trí của nó. Trong **advanced setting** Wumpus có khả năng di chuyển 1 ô theo 1 trong 4 hướng (đông, tây, nam, và bắc) với tỷ lệ bằng nhau sau mỗi 5 hành động của Explorer.

Thuộc tính (Attributes)

- **screamed**: giá trị Boolean, mặc định là `False`. Thuộc tính này biểu thị việc Wumpus đã gào thét (*scream*) hay chưa. Khi Wumpus bị bắn trúng mũi tên, giá trị này sẽ được đặt thành `True`.

Hành vi của Wumpus

- Wumpus ẩn nấp trong một ô của môi trường, hoặc di chuyển (nếu trong advanced setting).
- Nếu `Explorer` đi vào ô có Wumpus, tác nhân sẽ bị giết.
- Nếu `Explorer` bắn trúng Wumpus bằng mũi tên, Wumpus sẽ chết và phát ra tiếng thét (`screamed = True`), đồng thời các mùi hôi (Stench) xung quanh sẽ biến mất.

Tóm lại: `Wumpus` là một thực thể thụ động trong môi trường (hoặc động trong advanced setting), đại diện cho mối nguy hiểm mà tác nhân cần tránh hoặc tiêu diệt. Nó có thể di chuyển (hoặc không) và ảnh hưởng trực tiếp đến sự sống còn của tác nhân trong quá trình khám phá.

2.1.3 Mô tả các hàm phương thức trong class `WumpusEnvironment`

Các phương thức chính của lớp `WumpusEnvironment`. Mỗi mục nêu: chữ ký (signature), mục đích, tham số, giá trị trả về (nếu có) và các hiệu ứng phụ quan trọng.

`__init__(self, N=8, K_wumpuses=2, pit_probability=0.2, advanced_setting=False)`

- Mục đích: Khởi tạo môi trường Wumpus (bảng, tường, vị trí pit, vị trí Wumpus, vàng, v.v.).
- Tham số:
  - `N`: kích thước lưới (chiều cao và chiều rộng).
  - `K_wumpuses`: số Wumpus trong môi trường.
  - `pit_probability`: xác suất một ô (không phải ô khởi đầu) có pit.
  - `advanced_setting`: cờ bật chế độ nâng cao (Wumpus di chuyển).
- Hành vi chính: tạo ma trận `board` với chỉ mục từ `0..N+1` để chứa tường ranh giới; gọi `add_wall`; sinh pit ngẫu nhiên theo xác suất; đặt Wumpus (và thêm `Stench` quanh chúng); đặt vàng ở ô ngẫu nhiên khác (1,1) và vị trí đã dùng; khởi tạo thuộc tính trạng thái (ví dụ `wumpus_pos`, `pit_pos`, `gold_taken`, `agents`, `action_counts`, `game_over`, `status`).
- Hiệu ứng phụ: sửa đổi `self.board`, `self.wumpus_pos`, `self.pit_pos`, v.v.

`add_wall(self)`

- Mục đích: thêm đối tượng `Wall` vào các ô biên (chỉ mục 0 và N+1) để tạo ranh giới.
- Tham số: không.

- Trả về: không.
- Hiệu ứng phụ: cập nhật `self.board` tại các ô biên.

`print_board(self)`

- Mục đích: in trạng thái hiện tại của bảng ra console (dùng để debug / hiển thị).
- Tham số: không.
- Trả về: không.
- Ghi chú: in thêm `action_counts` và duyệt các ô để hiển thị nội dung (Wumpus, Pit, Gold, v.v.).

`is_in_map(self, pos)`

- Mục đích: kiểm tra vị trí `pos=(y,x)` có nằm trong bản đồ hợp lệ hay không (không phải tường biên).
- Tham số: `pos`: tuple (y, x).
- Trả về: `True` nếu `1 <= y <= height` và `1 <= x <= width`, ngược lại `False`.

`update_stench(self)`

- Mục đích: làm mới (clear rồi tái tạo) tất cả `Stench` trên bảng dựa trên vị trí hiện tại của các Wumpus.
- Hành vi chính:
  - Xóa tất cả đối tượng `Stench` hiện có trong vùng chơi.
  - Với mỗi vị trí Wumpus, thêm `Stench` vào các ô kề (lên/xuống/trái/phải) nếu ô đó không chứa Pit/Wumpus và không phải vị trí Wumpus khác.
- Hiệu ứng phụ: cập nhật nội dung `self.board` (thêm/bớt `Stench`).

`exe_action(self, agent, pos, action)`

- Mục đích: thực thi một hành động của agent tại vị trí `pos` và trả về percepts tạo ra sau hành động.
- Tham số:
  - `agent`: đối tượng agent (thường là `Explorer`).
  - `pos`: vị trí hiện tại của agent (y, x).
  - `action`: chuỗi tên hành động (ví dụ `'MoveForward'`, `'TurnLeft'`, `'Grab'`, `'Climb'`, `'Shoot'`).
- Hành vi chính:
  - Quản lý quay mặt (`TurnLeft/TurnRight`) bằng cách cập nhật `agent.direction`.
  - Di chuyển (`MoveForward`): tính ô tiếp theo, kiểm tra va chạm với tường (gán `bump` nếu gặp tường), cập nhật vị trí agent nếu hợp lệ, kiểm tra chết do Pit/Wumpus.
  - Bắt vàng (`Grab`): nếu ô hiện tại chứa `Gold`, remove Gold/Glitter, cập nhật `gold_taken` và trạng thái agent.
  - Leo lên (`Climb`): nếu agent ở (1,1) thì kết thúc trò chơi / agent thoát (cập nhật `status`, `game_over`).
  - Bắn (`Shoot`): nếu agent có mũi tên, khởi hành viên đạn theo hướng hiện tại; đạn đi thẳng đến khi trúng tường hoặc trúng Wumpus; nếu trúng Wumpus, xóa Wumpus và thêm `Scream` vào ô trúng.
  - Tăng `action_counts` và (nếu chế độ nâng cao) định kỳ di chuyển Wumpus mỗi 5 hành động.
- Trả về: danh sách percepts hiện tại tại vị trí agent (các đối tượng như `Stench`, `Breeze`, `Glitter`, `Scream`, `Bump`).
- Hiệu ứng phụ: cập nhật `self.board`, trạng thái agent, `wumpus_pos`, `gold_taken`, `action_counts`, có thể gọi `wumpus_move()`.

`percept(self, pos)`

- Mục đích: lấy danh sách percepts hiện có tại vị trí `pos`.
- Tham số: `pos` = (y, x).

- Trả về: danh sách percept objects (lọc các instance của `Breeze`, `Stench`, `Glitter`, `Scream`, `Bump` từ ô hiện tại).
- Ghi chú: không thay đổi trạng thái môi trường.

`in_danger(self, agent)`

- Mục đích: kiểm tra agent có đang đứng trên ô nguy hiểm (Pit hoặc Wumpus) hay không; nếu có, "giết" agent.
- Tham số: `agent`.
- Trả về: `True` nếu agent bị giết (hoặc đang ở ô nguy hiểm), ngược lại `False`.
- Hiệu ứng phụ: nếu agent chết, cập nhật cờ `agent.alive=False`, `game_over`, và cập nhật `status` (ví dụ 'killed by pit' hoặc 'killed by wumpus').

`is_end(self)`

- Mục đích: kiểm tra xem trò chơi đã kết thúc hay chưa.
- Trả về: `True` nếu không còn Explorer sống hoặc Explorer đã leo ra khỏi hang tại (1,1); ngược lại `False`.
- Hiệu ứng phụ: có thể cập nhật `game_over` và `status`.

`wumpus_move(self)`

- Mục đích: (chế độ nâng cao) di chuyển các Wumpus theo một luật đơn giản (ngẫu nhiên sang ô kề).
- Hành vi chính:
  - Lặp qua bản sao của `wumpus_pos`; với mỗi Wumpus chọn hướng ngẫu nhiên trong {`"left"`,`"right"`,`"up"`,`"down"`}.
  - Nếu ô đích hợp lệ (trong bản đồ và không phải tường/pit/vị trí Wumpus khác), cập nhật vị trí Wumpus trong `board` và danh sách `wumpus_pos`.
  - Gọi `update_stench()` sau khi di chuyển để làm mới `Stench`.
- Hiệu ứng phụ: thay đổi `board` và `wumpus_pos`.

Ghi chú tổng quát về trạng thái và side effects

- Phần lớn phương thức thao tác trực tiếp lên `self.board` (thêm/xóa các đối tượng trong ô), các danh sách vị trí như `wumpus_pos`, `pit_pos` và cờ trạng thái như `gold_taken`, `game_over`, `status`, `action_counts`.
- Các hàm tương tác trực tiếp với agent (ví dụ `exe_action`, `in_danger`) có thể thay đổi thuộc tính của agent (vị trí, hướng, `alive`, inventory).
- Trong chế độ nâng cao, môi trường có trạng thái động (Wumpus có thể di chuyển), vì vậy agent cần cập nhật KB/percepts thường xuyên.

2.1.4 Knowledge Base (KB)

`KnowledgeBase` là một lớp dùng để quản lý cơ sở tri thức mà tác nhân (Explorer) sử dụng trong quá trình khám phá môi trường. Lớp này chịu trách nhiệm lưu trữ, cập nhật và suy luận dựa trên các thông tin thu nhận từ môi trường, chi tiết về Knowledge Base được nhóm định nghĩa như sau đây:

Cấu trúc và Thuộc tính

- **width, height**: kích thước của môi trường (mặc định $N = 8$).
- **visited**: tập hợp các ô đã được tác nhân ghé thăm (mặc định là $\{(1, 1)\}$).
- **clauses**: tập hợp các mệnh đề logic (các Object tạo từ các Class trong `logic.py`), được lưu trữ dưới CNF.
- **clause_formulas**: tập hợp (kiểu dữ liệu `set()`) chuỗi `string` công thức logic trong KB, nhằm theo dõi và tránh thêm trùng lặp mệnh đề.
- **symbols**: ánh xạ từ các đối tượng trong môi trường (Wumpus, Pit, Stench, Breeze, ...) sang các ký hiệu logic (`Symbol`).
  - *Lý do sử dụng:* Trong KB, các clause được lưu dưới dạng object của class con của class Sentence với định dạng là `'nameClass'_y_x`. Ví dụ: có Breeze tại vị trí $(2, 3)$ thì thuộc tính `clauses` chứa `Symbol('Breeze_2_3')` (tạm gán `a = Symbol('Breeze_2_3')` hay tổng quát là `Symbol(f'{obj.__clasStench__name}_{y}_{x}')` $\forall y, x$. Vậy nên, việc truy cập đến phần tử `a` (`Symbol('Breeze_2_3')`) trong `clauses` để tạo một clause khác liên quan rất khó khăn vì nó là 1 object - instance.

– Ví dụ: (Không sử dụng thuộc tính `symbols`) Giả sử, hiện tại trong `clauses` của KB đang có `Symbol('Breeze_1_1')` → đang percept Breeze tại vị trí $(1, 1)$, mình muốn thêm 1 Sentence **S** là `¬(Breeze_1_1) v Pit_1_2 v Pit_2_1`, tức:

```
Or(Not(Symbol('Breeze_1_1')), Symbol('Pit_1_2'), Symbol('Pit_2_1'))
```

Thì câu hỏi đặt ra là liệu **instance của `Symbol('Breeze_1_1')` có sẵn trong `clauses` với `Symbol('Breeze_1_1')` trong Sentence S có giống nhau?**
⇒ Câu trả lời là **KHÔNG** vì instance `Symbol('Breeze_1_1')` trong Sentence **S** sẽ được tạo mới khi được gọi, các `Symbol` này chỉ giống nhau về mặt hình thức nhưng bản chất KHÔNG như nhau. Vậy nên, sự KHÔNG liên kết này làm ảnh hưởng đến việc suy luận (inference) sau này.

- **last_shot**: lưu trạng thái (vị trí bắn, hướng bắn, bắn lúc nào - step mấy) khi Explorer thực hiện hành động bắn (nếu có). *Lưu ý:* chỉ bắn 1 lần duy nhất.
- **action_count**: đếm số lần thực hiện action.
- **is_advanced**: True/False để xem có phải đang ở chế độ Advanced Setting.

Ví dụ trong Knowledge Base được lưu trữ trong clause_formulas

- `¬(Pit_4_2)`: không có Pit ở vị trí ở tọa độ y = 4, x = 2.
- `Breeze_3_2`: có Breeze ở vị trí tọa độ y = 3, x = 2.
- `¬(Breeze_1_1) v Pit_1_2 v Pit_2_1`: không có Breeze tại $(1, 1)$ hoặc có Pit tại $(1, 2)$ hoặc có Pit tại $(2, 1)$. *Dễ hiểu*: có Breeze tại $(1, 1)$ thì có Pit tại $(1, 2)$ hoặc có Pit tại $(2, 1)$.

Tóm lại: `KnowledgeBase` cung cấp cơ chế để biểu diễn tri thức dưới dạng logic mệnh đề, cho phép tác nhân lưu trữ thông tin từ môi trường, cập nhật dựa trên hành động và cảm nhận, và thực hiện suy luận để hỗ trợ ra quyết định trong quá trình khám phá.

2.2 Sự tạo thành cho các luật (Rules Formulation) trong KB

Trong bài toán Wumpus World, cơ sở tri thức (Knowledge Base – KB) được xây dựng dựa trên Logic mệnh đề (Propositional Logic) để mô tả các đối tượng và cảm nhận trong lưới kích thước $N \times N$. Phần này trình bày cách định nghĩa các ký hiệu logic và các luật tri thức liên quan đến `Wumpus`, `Pit`, `Breeze`, `Stench`, và `Gold`, cùng với cách mã hóa cho trường hợp có nhiều Wumpus và nhiều hố (Pit).

2.2.1 Định nghĩa ký hiệu

Mỗi ô (x, y) trong board được ánh xạ thành các ký hiệu logic, biểu thị sự hiện diện của các đối tượng hoặc cảm nhận:

- Wumpus_y_x: Có Wumpus tại ô (x, y).
- Pit_y_x: Có hố (Pit) tại ô (x, y).
- Stench_y_x: Có mùi hôi (Stench) tại ô (x, y).
- Breeze_y_x: Có gió (Breeze) tại ô (x, y).
- Gold_y_x: Có vàng (Gold) tại ô (x, y).
- Glitter_y_x: Có vàng (Gold) tại ô (x, y).

Các ký hiệu này được khởi tạo cho mọi ô (x, y) với $1 \leq x, y \leq N$, đảm bảo mô tả đầy đủ trạng thái của môi trường. Nhưng các ký hiệu trên là ký hiệu formula chuỗi string của các Symbol object. Nhưng để báo cáo nhìn gọn mắt và tiện lợi em xin phép ký hiệu trong báo cáo như sau:

- Wumpus_y_x $\rightarrow \mathrm{W}_{y,x}$.
- Pit_y_x $\rightarrow \mathrm{P}_{y,x}$.
- Stench_y_x $\rightarrow \mathrm{S}_{y,x}$.
- Breeze_y_x $\rightarrow \mathrm{Br}_{y,x}$.
- Gold_y_x $\rightarrow \mathrm{Gold}_{y,x}$.

2.2.2 Các luật tri thức cơ bản

Cơ sở tri thức được xây dựng dựa trên các quy tắc sau, phản ánh các ràng buộc và mối quan hệ trong môi trường Wumpus World:

1. **An toàn tại ô xuất phát:** Ô khởi đầu $(1, 1)$ được đảm bảo không chứa Wumpus hoặc hố:

$$\neg \mathrm{W}_{1,1} \wedge \neg \mathrm{P}_{1,1}$$

2. **Ràng buộc Wumpus và hố:** Một ô không thể chứa cả Wumpus và hố cùng lúc:

$$\forall x, y : \neg(\mathrm{W}_{y,x} \wedge \mathrm{P}_{y,x})$$

3. **Quy tắc về mùi hôi (Stench):** Mùi hôi xuất hiện tại ô (x, y) khi và chỉ khi có ít nhất một Wumpus ở ô kề bên (trên, dưới, trái, phải):

$$\mathrm{S}_{y,x} \leftrightarrow (\mathrm{W}_{y-1,x} \vee \mathrm{W}_{y+1,x} \vee \mathrm{W}_{y,x-1} \vee \mathrm{W}_{y,x+1})$$

hay,

$$\mathrm{S}_{y,x} \leftrightarrow \bigvee_{(j,i)\in \mathrm{adj}(y,x)} \mathrm{W}_{j,i}$$

Công thức này sử dụng phép `OR` để biểu thị rằng mùi hôi có thể do bất kỳ Wumpus nào ở các ô lân cận gây ra. Trong đó, $\mathrm{adj}(y, x)$ là tập hợp các ô kề bên $(y, x-1)$, $(y, x+1)$, $(y-1, x)$, $(y+1, x)$, với điều kiện các ô này nằm trong board.

4. **Quy tắc về gió (Breeze):** Gió xuất hiện tại (x, y) khi và chỉ khi có ít nhất một hố ở kề bên:

$$\mathrm{Br}_{y,x} \leftrightarrow (\mathrm{P}_{y-1,x} \vee \mathrm{P}_{y+1,x} \vee \mathrm{P}_{y,x-1} \vee \mathrm{P}_{y,x+1})$$

hay,

$$\mathrm{Br}_{y,x} \leftrightarrow \bigvee_{(j,i)\in \mathrm{adj}(y,x)} \mathrm{P}_{j,i}$$

Tương tự, phép `OR` cho phép nhiều hố góp phần tạo ra gió.

5. **Tính tổng quát:** môi trường có thể chứa nhiều Wumpus và nhiều hố. Các công thức trên sử dụng phép `OR` để cho phép nhiều nguồn (Wumpus hoặc hố) cùng tạo ra một cảm nhận (percept) hoặc ngược lại một cảm nhận (percept) có thể tạo ra nhiều nguồn. Điều này làm cho KB linh hoạt, có thể xử lý các môi trường với số lượng Wumpus và hố không xác định trước. ⇒ Mã hóa cho nhiều Wumpus và nhiều hố.

6. **Sự di chuyển của Explorer:** với sự di chuyển của Explorer, mỗi khi đến ô mới (y', x') mà Explorer không chết (chết vì Wumpus hoặc Pit) thì trong hàm `update_percept_sentence()`, chỉ cần truy xuất symbol $\mathrm{W}_{y',x'}$ và $\mathrm{P}_{y',x'}$ và thêm mệnh đề `Not` cho các symbol đó và thực hiện phép `And` và bỏ vào `clauses` của KB: $\neg\mathrm{W}_{y',x'} \wedge \neg\mathrm{P}_{y',x'}$. Đồng thời, vị trí này được đánh dấu là đã thăm và với tham số input `percepts` truyền vào - tham số này được truyền vào bởi kết quả trả về sau hàm phương thức `percept()` của class `WumpusEnvironment` trong file `environment.py`, KB sẽ được cập nhật các cảm nhận (percept: Bump/Glitter/Stench/Breeze) tại vị trí này có hay không.

- Ví dụ về quy trình nhận percept:

```
percepts = env.percept(agent.location)
agent.kb.update_percept_sentence(agent.location, percepts)
```

7. **Quay trái, phải:** Trong chế độ Normal, percept về Stench không thay đổi nên việc gọi hàm `update_percept_sentence()` cũng không ảnh hưởng KB vì luôn có điều kiện kiểm tra xem Sentence đó có trong KB hay chưa qua `if formula not in self.clause_formulas` trong `__iadd__()`.

8. **Hành động bắn cung:** Khi rơi vào trường hợp không thể nhận diện các ô chưa thăm là an toàn hay không, Explorer cần bắn về hướng có thể có Wumpus, nơi có Stench. Khi này sẽ cần hàm `update_action_sentence()` lúc này sẽ có symbol mới thể hiện đã bắn cung với các thành tố: vị trí bắn, hướng, lúc bắn. Sau đó, khi thực hiện bắn thì hàm `update_percept_sentence()` sẽ cập nhật rằng CHẮC CHẮN ô kế hướng bắn sẽ là ô an toàn. Vì ô này dù có hay ko có Wumpus, sau khi bắn thì chắc chắn ô có thể đi được. Ví dụ hướng bắn sang phải tại vị trí (y, x):

$$\text{Shoot\_}y\text{\_}x\text{\_right\_step} \rightarrow \neg \text{W}_{y,x+1}$$

Sau đó, sẽ thêm percept $\text{Scream}_{y,x}$ - tức nghe tiếng Scream ở (y, x). Đồng thời, nếu có bất kỳ sự thay đổi percept nào ở nhưng ô đã thăm, ví dụ không còn Stench ở (y', x') nữa thì Explorer sẽ dùng hàm `remove_clauses()` để loại đi Sentence: $\text{S}_{y',x'}$ và thêm $\neg\text{S}_{y',x'}$ để phục vụ việc suy luận mấy lần sau không bị xung đột.

Lưu ý: Mỗi khi thêm bất kỳ Sentence, Symbol,... - gọi chung: Sentence, vào `clauses` của KB thì đều được gọi hàm `to_cnf()` để đổi sang định dạng `cnf` để dễ thực hiện tác vụ Suy luận (Inference) sau này.

2.2.3 Hàm gọi suy luận ask(query)

Hàm này sẽ gọi hàm `pl_resolution` trong file `logic.py` với câu query và trả với True/False. Quy tắc hỏi trong chương trình này luôn là:

```
ask(Not(self.symbols[('Pit', y, x)]))
```

và

```
ask(Not(self.symbols[('Wumpus', y, x)]))
```

Nếu cả 2 cùng trả về True, trong phần planning sẽ đánh dấu (y, x) đó là an toàn, còn trả về False không thêm (y, x) vào bất kỳ đâu. Tức là ô đó *"không chắc chắn an toàn hay không"* hoặc *"ô đó không an toàn"* đều sẽ trả về False.
Lưu ý: Quy luật chung ở chương trình này luôn chỉ hỏi `Not(self.symbols[('Pit', y, x)])` và `Not(self.symbols[('Wumpus', y, x)])` để tránh phức tạp hoá việc suy luận, khi Explorer chỉ cần biết ở đầu là an toàn để đi, không cần biết kiểu như "ô này chắc chắn có Wumpus/Pit".

2.2.4 Ở chế độ Advanced

Nhờ biến `action_counts`, KB của Explorer sẽ biết khi nào đã thực hiện số action chia hết cho 5 (`action_counts`%5). Vì vậy, khi thực hiện xong lần đó, trong hàm `update_action_sentence()` sẽ thực hiện loại bỏ tất cả các Tri thức (Sentence) ở các vị trí đã thăm về việc có:

$$\mathrm{S}_{y,x} \land \neg \mathrm{W}_{y,x} \ \forall (y, x) \in \text{visited}$$

Vì khi này, vị trí này có thể có Wumpus, nên không thể để Sentence $\neg \mathrm{W}_{y,x}$ tồn tại trong KB nữa, việc kiểm tra thêm $\mathrm{S}_{y,x}$ để KHÔNG phải loại bỏ các ô $\neg$W ĐÚNG. Vì các ô không có Stench ban đầu, thì sau khi Wumpus di chuyển sẽ không thể di chuyển được.
$\Rightarrow$ Việc loại bỏ các thông tin có thể gây nguy hiểm này cho agent Explorer là rất cần thiết khi nó có thể giúp việc suy luận (inference) của Explorer không bị sai lệch và vẫn dự tính rằng "ô này chưa chắc an toàn".

2.2.5 Kết luận

Cơ sở tri thức được xây dựng dựa trên các ký hiệu và luật logic chặt chẽ, cho phép mô tả chính xác trạng thái của môi trường Wumpus World. Việc sử dụng các phép toán logic như `OR` và `Biconditional` trong các quy tắc về `Stench` và `Breeze` đảm bảo rằng KB có thể xử lý trường hợp nhiều Wumpus và nhiều hố một cách hiệu quả. Cách tiếp cận này tạo nền tảng cho việc suy luận logic trong các phiên bản mở rộng của bài toán.

2.3 Inference Engine

Đây là phần cốt lõi của chương trình - một bộ não thực thụ, khi nhờ nó các quyết định rằng ô (y, x) $\forall x, y$ có an toàn hay không? Sẽ có 2 file python chính xử lý vấn đề này: `logic.py` và `knowledgeBase.py`.

2.3.1 Chế độ Normal

Để kiểm tra xem một mệnh đề logic, chẳng hạn như OK_y_x (ô (x, y) an toàn), có được suy ra từ cơ sở tri thức (KB) hay không, thuật toán độ phân giải (resolution) được sử dụng. Thuật toán này hoạt động trên các mệnh đề ở dạng chuẩn liên kết (Conjunctive Normal Form – CNF), tức là tập hợp các mệnh đề liên kết (conjunction) của các mệnh đề tách biệt (disjunction). Dưới đây là mô tả cách thức hoạt động của thuật toán để xác minh tính hợp lệ của các luật tri thức trong bài toán Wumpus World.

1. **Chuẩn hóa các mệnh đề trong KB:** Các luật tri thức trong KB, chẳng hạn như:

$$S_{y,x} \leftrightarrow (W_{y-1,x} \vee W_{y+1,x} \vee W_{y,x-1} \vee W_{y,x+1})$$

   được chuyển đổi sang dạng CNF. Quá trình này bao gồm:

   - Loại bỏ các phép toán hàm ý ($\Rightarrow$) và hai chiều ($\leftrightarrow$) bằng cách thay thế chúng bằng các phép `OR` và `AND`.
   - Đẩy các phép phủ định ($\neg$) vào trong bằng luật De Morgan, đảm bảo phủ định chỉ áp dụng cho các ký hiệu đơn. Ví dụ: $\neg(W_{y,x})$.
   - Phân phối phép `OR` qua `AND` để tạo ra các mệnh đề tách biệt.

   Kết quả là một tập hợp các mệnh đề tách biệt, từ ví dụ trên ta được trong KB:

$$\begin{aligned}
&\neg S_{y,x} \vee W_{y-1,x} \vee W_{y+1,x} \vee W_{y,x+1} \vee W_{y,x-1} \\
&\neg W_{y-1,x} \vee S_{y,x} \\
&\neg W_{y+1,x} \vee S_{y,x} \\
&\neg W_{y,x-1} \vee S_{y,x} \\
&\neg W_{y,x+1} \vee S_{y,x}
\end{aligned}$$

2. **Thêm phủ định của truy vấn:** Để kiểm tra xem KB có suy ra được truy vấn (query) như $(\neg W_{y,x})$ hay không, phủ định của truy vấn $W_{y,x}$ được thêm vào KB. Nếu KB và $W_{y,x}$ dẫn đến mâu thuẫn (contradiction), thì truy vấn được suy ra từ KB. Đây là chứng minh kiểu phản chứng trong toán học - một cách chứng minh tuyệt vời

3. **Unit Propagation:** Các mệnh đề đơn (unit clauses), tức là các "ký hiệu" hoặc "phủ định của ký hiệu" (ví dụ: $W_{1,1}$ hoặc $\neg P_{1,1}$), được sử dụng để đơn giản hóa các mệnh đề khác. Nếu một mệnh đề chứa một ký hiệu trái với mệnh đề đơn, mệnh đề đó được loại bỏ. Nếu chứa ký hiệu giống mệnh đề đơn, ký hiệu đó được xóa khỏi mệnh đề, có thể dẫn đến mệnh đề mới hoặc mâu thuẫn. Theo Wiki - Unit Propagation Có hai quy tắc:

- Mọi mệnh đề chứa ký hiệu của mệnh đề đơn (unit clause) được loại bỏ, vì mệnh đề đó đã được thỏa mãn.
- Trong mọi mệnh đề chứa phủ định của ký hiệu trong mệnh đề đơn, phủ định đó được xóa, vì nó không thể góp phần làm mệnh đề được thỏa mãn.

Ví dụ: Giả sử KB chứa tập hợp các mệnh đề sau:

$$\{\mathrm{W}_{1,1} \vee \mathrm{S}_{2,1}, \neg\mathrm{W}_{1,1} \vee \mathrm{P}_{1,2}, \neg\mathrm{P}_{1,2} \vee \mathrm{Br}_{2,2}, \mathrm{W}_{1,1}\}$$

Vì $\mathrm{W}_{1,1}$ là một mệnh đề đơn, ta áp dụng lan truyền đơn vị:

- Mệnh đề $\mathrm{W}_{1,1} \vee \mathrm{S}_{2,1}$ chứa $\mathrm{W}_{1,1}$, nên được loại bỏ (mệnh đề đã thỏa mãn).
- Mệnh đề $\neg\mathrm{W}_{1,1} \vee \mathrm{Pit}_{1,2}$ chứa $\neg\mathrm{W}_{1,1}$, nên $\neg\mathrm{W}_{1,1}$ được xóa, dẫn đến mệnh đề mới: $\mathrm{Pit}_{1,2}$.
- Mệnh đề $\neg\mathrm{P}_{1,2} \vee \mathrm{Br}_{2,2}$ không chứa $\mathrm{W}_{1,1}$ hoặc $\neg\mathrm{W}_{1,1}$, nên không thay đổi.
- Mệnh đề đơn $\mathrm{W}_{1,1}$ được giữ nguyên (hoặc có thể được lưu trong một mô hình cục bộ, tùy thuộc vào cách triển khai).

Kết quả là tập hợp mới:

$$\{\mathrm{P}_{1,2}, \neg\mathrm{P}_{1,2} \vee \mathrm{Br}_{2,2}, \mathrm{W}_{1,1}\}$$

Mệnh đề đơn mới $\mathrm{P}_{1,2}$ có thể được sử dụng để tiếp tục lan truyền đơn vị, ví dụ, xóa $\neg\mathrm{P}_{1,2}$ khỏi $\neg\mathrm{P}_{1,2} \vee \mathrm{Br}_{2,2}$, dẫn đến $\mathrm{Br}_{2,2}$.

4. **Resolution:** Thuật toán lặp lại quá trình độ phân giải giữa các cặp mệnh đề. Nếu hai mệnh đề chứa các ký hiệu trái ngược (ví dụ: $\mathrm{W}_{y,x}$ và $\neg\mathrm{W}_{y,x}$), chúng được kết hợp để tạo ra một mệnh đề mới bằng cách loại bỏ các ký hiệu trái ngược. Ví dụ:

$$(\mathrm{W}_{y,x} \vee \mathrm{S}_{y+1,x}) \wedge (\neg\mathrm{W}_{y,x} \vee \mathrm{P}_{y,x}) \rightarrow \mathrm{S}_{y+1,x} \vee \mathrm{P}_{y,x}$$

   Nếu kết quả là một mệnh đề rỗng (False), tức là mâu thuẫn, thì truy vấn được suy ra.

5. **Kiểm tra bao hàm (Subsumption):** Các mệnh đề mới được tạo ra được so sánh với các mệnh đề hiện có để loại bỏ các mệnh đề dư thừa (subsumed clauses). Điều này đảm bảo KB không chứa các mệnh đề trùng lặp hoặc không cần thiết.

6. **Kết thúc:** Thuật toán dừng khi tìm thấy mâu thuẫn (truy vấn được suy ra) hoặc khi không còn mệnh đề mới nào được tạo ra (truy vấn không được suy ra). Để tránh vòng lặp vô hạn, một giới hạn số lần lặp (ví dụ: 1000) được đặt ra.

Kết quả của quá trình này là xác định liệu các luật tri thức trong KB, chẳng hạn như $\neg W_{1,1} \wedge \neg P_{1,1}$, có đủ để suy ra ô $1,1$ có an toàn hay không. Cách tiếp cận này đảm bảo tính chặt chẽ trong việc xử lý các luật logic phức tạp, đặc biệt trong môi trường có nhiều Wumpus và nhiều hố (Pit).

2.3.2 Chế độ Advanced

Vì việc loại bỏ như đã nói ở 2.2.4 sẽ khiến KB của Agent sẽ không thể suy ra được những tín hiệu nhiễu như "thời điểm này có Stench tại (y, x)" mà "thời điểm này lại không có Stench tại (y, x)". Nên việc suy luận sẽ như bình thường ở chế độ Normal, khi đó Explorer chỉ suy luận được số ô an toàn ít hơn. Nếu không có ô an toàn nào thì sẽ di chuyển mạo hiểm như phần 4 của 2.5 đã đề cập.

2.4 Planning Module

Ở dự án này, nhóm quyến định sử dụng chiến lược tìm kiếm có thông tin **Astar**.
Chiến lược A* trong Wumpus World sử dụng một hàm chi phí tổng quát $f(n) = g(n) + h(n)$, trong đó:

- $g(n)$: Chi phí thực tế từ điểm bắt đầu đến trạng thái hiện tại (số bước di chuyển).
- $h(n)$: Heuristic ước lượng chi phí từ trạng thái hiện tại đến mục tiêu, sử dụng khoảng cách Manhattan:

  $$h(n) = |x_1 - x_2| + |y_1 - y_2|$$

  với (x_1, y_1) là vị trí hiện tại và (x_2, y_2) là vị trí mục tiêu.

Hàm `find_path_astar(start, goal)` thực hiện thuật toán A* để tìm đường đi tối ưu từ vị trí xuất phát `start` đến vị trí đích `goal` trong môi trường Wumpus World. Hàm trả về một danh sách các hành động (`List[str]`) mô tả đường đi tối ưu. Thuật toán hoạt động theo các bước sau:

1. **Kiểm tra trạng thái ban đầu**:
   Nếu `start == goal` thì trả về danh sách rỗng vì không cần di chuyển.
2. **Khởi tạo dữ liệu**:
   - `open_set`: hàng đợi ưu tiên (priority queue) lưu các node đang chờ xét.
   - `closed_set`: tập các vị trí đã được xét.
   - `node_map`: ánh xạ vị trí → node tốt nhất hiện tại ở vị trí ấy.

   Khởi tạo `start_node` với chi phí $g = 0$ và heuristic h bằng khoảng cách Manhattan đến `goal`.
3. **Vòng lặp chính của thuật toán A***:

- Mỗi vòng lấy ra phần tử trong `open_set` có $f = g + h$ nhỏ nhất.
- Nếu vị trí hiện tại là đích (`goal`), tiến hành truy ngược node cha để tái cấu trúc đường đi:
  - Duyệt từ node hiện tại ngược về `start` thông qua thuộc tính `parent`.
  - Mỗi bước thêm hành động (`action`) vào danh sách đường đi.
  - Đảo ngược danh sách và trả về.

4. **Mở rộng node**:
   - Thêm vị trí hiện tại vào `closed_set`.
   - Với mỗi vị trí kề (`neighbor_pos`):
     - Bỏ qua nếu vị trí đã nằm trong `closed_set`.
     - Bỏ qua nếu vị trí không an toàn (`is_position_safe`).
     - Tính $g\_cost = g_{current} + 1$.
     - Tính h_cost bằng khoảng cách Manhattan từ `neighbor_pos` đến `goal`.
     - Nếu đây là đường đi tốt hơn (hoặc chưa từng xét), tạo node mới và cập nhật vào `open_set` và `node_map`.
5. **Kết luận**:
   Nếu `open_set` rỗng mà vẫn chưa tới được `goal` $\Rightarrow$ không tồn tại đường đi hợp lệ nên hàm trả về `[]` (danh sách rỗng).

Thuật toán sử dụng heuristic Manhattan đảm bảo tối ưu trong lưới ô chữ nhật, và việc chỉ xét các ô "an toàn" giúp tránh rơi vào hố hoặc gặp Wumpus.

2.5 Hybrid Agent Integration

Kết hợp với giữa thuật toán tìm kiếm có thông tin A*, và Inference Engine, ta có Chiến lược ưu tiên các hành động theo thứ tự:

1. Nếu phát hiện vàng (Glitter), nhặt vàng và quay về (1,1) để thoát ra (Climb).
2. Khám phá các ô an toàn chưa được thăm (safe unvisited).
3. Nếu có mũi tên, lập kế hoạch bắn Wumpus ở các vị trí có thể (dựa trên Stench).
4. Thực hiện hành động mạo hiểm đến các ô chưa thăm nhưng không chắc chắn an toàn.
5. Nếu không còn lựa chọn, quay về (1,1) và thoát ra.

2.5.1 Tích hợp với Wumpus World

Thuật toán A* được tích hợp với môi trường Wumpus World thông qua:

- **Cập nhật tri thức**: Hàm `update_world_knowledge` sử dụng cảm giác để cập nhật tập các ô an toàn (`known_safe`) và không an toàn (`known_unsafe`).
- **Cơ sở tri thức**: Nếu có, cơ sở tri thức được sử dụng để suy ra sự an toàn của các ô thông qua các biểu thức logic như `Not(Pit_y_x)` và `Not(Wumpus_y_x)`.
- **Heuristic một cách chủ quan**: Chỉ các ô liền kề ô đã thăm và không có tín hiệu nguy hiểm được coi là an toàn nếu không có suy luận logic.
- **Quản lý hành động**: Các hành động như `MoveForward`, `TurnLeft`, `TurnRight`, `Shoot`, `Grab`, và `Climb` được lập kế hoạch dựa trên trạng thái hiện tại và mục tiêu.

2.5.2 Kết luận

Chiến lược A* trong Wumpus World kết hợp hiệu quả giữa tìm kiếm dựa trên heuristic và suy luận logic, đảm bảo tác nhân di chuyển an toàn khi có thể và thực hiện các hành động mạo hiểm có tính toán khi cần thiết. Việc ưu tiên các ô an toàn, bắn Wumpus, và quay về điểm xuất phát giúp tối ưu hóa hiệu suất và tăng khả năng thành công trong môi trường nguy hiểm.

2.6 Random Agent Baseline

Random Agent là một tác nhân cơ sở (baseline) trong môi trường Wumpus World, hoạt động dựa trên các hành động hợp lệ tại thời điểm hiện tại và lựa chọn ngẫu nhiên, không suy luận dựa trên percepts. Agent này không xây dựng bản đồ hay suy luận nguy hiểm, nhưng nó vẫn tuân theo một số ưu tiên đơn giản sau:

- Luôn nhặt vàng (**Grab**) ngay khi phát hiện tại vị trí hiện tại.
- Nếu đã nhặt được vàng và đang ở vị trí xuất phát $(1, 1)$ thì ưu tiên hành động **Climb** để thoát khỏi hang.
- Trong các hành động hợp lệ, agent ưu tiên di chuyển (**MoveForward**) với xác suất cao (60%) để tăng khả năng khám phá bản đồ.
- Nếu không thể đi thẳng, agent sẽ thực hiện xoay trái hoặc xoay phải một cách ngẫu nhiên (**TurnLeft**, **TurnRight**).

- Agent chỉ bắn mũi tên (**Shoot**) khi vẫn còn mũi tên (has_arrow = True), tuy nhiên hành động này cũng được chọn ngẫu nhiên (nếu không di chuyển tới phía trước thì sẽ ngẫu nhiên chọn các hành động có thể thực hiện, bao gồm cả bắn tên).
- Nếu bị kẹt hoặc đã thực hiện quá nhiều hành động (quá `max_actions`), agent cố gắng đi ngẫu nhiên để quay lại vị trí $(1, 1)$. Nếu đang ở $(1, 1)$ thì tự động leo ra.

Do Random Agent không suy luận dựa trên percepts (stench, breeze), hiệu quả không cao nhưng đóng vai trò là baseline để so sánh với tác nhân thông minh hơn như hybrid Agent.

3 Kết quả và kết luận

Để chạy thí nghiệm và lấy số liệu, trước hết chúng em sẽ chạy và lấy 5 map 8×8, 5 map 6×6, sao cho 10 map đó Hybrid agent đều giải được. Quá trình tìm kiếm các map phù hợp được thực hiện bằng cách:

1. Sinh ngẫu nhiên các môi trường Wumpus World với 1 Wumpus và xác suất hố 0.2
2. Kiểm tra xem Hybrid agent có thể giải được map hay không.
3. Chỉ giữ lại những map mà Agent có thể giải được.
4. Lưu trữ seed của 10 map được chọn để đảm bảo tính tái lập.

Sau khi có được 10 map, chúng em thực hiện chạy cả 2 agent (Hybrid agent và Random Agent) để lấy các chỉ số thống kê sau cho 2 loại map:

1. **Performance (Điểm số):**
   - Điểm trung bình
   - Điểm cao nhất
   - Điểm thấp nhất
2. **Khả năng giải bài toán:**
   - Tỷ lệ tìm thấy vàng
   - Tỷ lệ sống sót

3.1 Kết quả của yêu cầu nâng cao

Bảng 3: Kết quả thống kê trên 5 map 8×8

Chỉ số	**Hybrid advanced agent**
Điểm trung bình	388
Điểm cao nhất	**1004**
Điểm thấp nhất	-1054 (bị wumpus giết), 8 (lụm vàng nhưng không kiếm được đường về)
Tỷ lệ tìm thấy vàng	**80%**
Tỷ lệ sống sót	80%

Bảng 4: Kết quả thống kê trên 5 map 6×6

Chỉ số	**Hybrid advanced agent**
Điểm trung bình	587.4
Điểm cao nhất	**1004**
Điểm thấp nhất	-1036 (bị pit giết)
Tỷ lệ tìm thấy vàng	**80%**
Tỷ lệ sống sót	80%

3.2 So sánh Hybrid agent và random agent

3.2.1 Kết quả thực nghiệm trên Map 8×8

Bảng 5: Kết quả thống kê trên 5 map 8×8

Chỉ số	**Hybrid agent**	**Random Agent**
Điểm trung bình	984.6	-604.60
Điểm cao nhất	**994**	**984**
Điểm thấp nhất	950	-1019
Tỷ lệ tìm thấy vàng	**100%**	80%
Tỷ lệ sống sót	100%	20%

3.2.2 Kết quả thực nghiệm trên Map 6×6

Bảng 6: Kết quả thống kê trên 5 map 6×6

Chỉ số	**Hybrid agent**	**Random Agent**
Điểm trung bình	994	-1013.60
Điểm cao nhất	1004	-995
Điểm thấp nhất	985	-1021
Tỷ lệ tìm thấy vàng	**100%**	**100%**
Tỷ lệ sống sót	100%	0%

3.2.3 Nhận xét, so sánh giữa 2 agent

Từ kết quả thống kê trên, chúng em rút ra các nhận xét sau:

1. **Về hiệu suất tìm vàng:** Hybrid agent cho thấy khả năng vượt trội với tỷ lệ tìm thấy vàng 100% trên cả hai loại map, trong khi Random Agent chỉ đạt 80% trên map 8×8. Điều này chứng tỏ thuật toán A* và knowledge base giúp agent đưa ra quyết định chính xác hơn.

2. **Về điểm số performance:** Trên map 8×8, Hybrid agent đạt điểm cao nhất là 1004 (thành công thoát ra với vàng), cao hơn Random Agent (984 điểm). Hybrid agent luôn đảm bảo khi tìm được vàng là sẽ chắc chắn sẽ tìm được đường ra và chiến thắng.

3. **Về độ ổn định:** Hybrid agent có độ biến thiên điểm số thấp hơn Random Agent, cho thấy tính ổn định và dự đoán được của thuật toán logic so với việc hành động ngẫu nhiên. Điều này đặc biệt quan trọng trong các ứng dụng thực tế đòi hỏi độ tin cậy cao.

Bảng 7: Tổng kết so sánh hiệu suất giữa Hybrid agent và Random Agent

Chỉ số	**Hybrid agent**		**Random Agent**		**Nhận xét**
	8×8	**6×6**	**8×8**	**6×6**	
Điểm TB	984.6	994	-604.60	-1013.60	Hybrid agent ổn định hơn
Điểm cao nhất	**1004**	1004	**984**	-995	Hybrid agent luôn tối ưu hơn
Điểm thấp nhất	950	985	-1019	-1021	Map 8x8 có sự biến thiên cao
Tỷ lệ tìm vàng	**100%**	**100%**	80%	**100%**	Tỷ lệ Random tìm thấy vàng khá cao
Tỷ lệ sống sót	100%	100%	20%	0%	Random vẫn có cơ hội giải được

3.3 Đánh giá và nhận xét

- Hybrid agent: Thể hiện tốt và chạy tốt, đã đảm bảo được khi tìm thấy vàng chắc chắn sẽ trở về an toàn và chiến thắng.

- Random agent: Chọn action ngẫu nhiên, do đó độ hiểu quả không thể so được với hybrid agent.

- Advanced setting Agent: Khi Hybrid agent giải được trong trường hợp số bước thực hiện thấp và ít wumpus thì khả năng cao Advanced Agent vẫn giải được.

4 Video

Video demo youtube của chúng em: Video

5 Tài liệu tham khảo

1. geeksforgeeks, The Wumpus World Description - AI, url, [8/8/2025]
2. humbertobg, Wumpus World, url, [9/8/2025]
3. aima.eecs.berkeley.edu, Logical agents, url, [9/8/2025]
4. Mahesh Huddar, Wumpus World Example in AI, url, [7/8/2025]
5. AIMA code, Logic, url, [30/07/2025]
6. AIMA code, Agent, url, [28/07/2025]
7. Havard CS50, Knowledge example, url, [27/07/2025]
8. Havard CS50, Knowledge Minesweeper, url, [27/07/2025]

6 Lời cảm ơn

Nhóm em xin chân thành gửi lời cảm ơn tới thầy *Nguyễn Thanh Tình* vì đã chỉ dẫn và giải đáp thắc mắc trong quá trình hoàn thiện dự án. Bên cạnh đó, em xin cảm ơn thầy *Bùi Duy Đăng* vì đã cho nhóm em gợi ý khi xử lý yêu cầu nâng cao.
Lời cuối, đồ án này có sự giúp đỡ của ChatGPT, Grok, Copilot đã giúp đỡ trong quá trình viết báo cáo, tra cứu thông tin, và đề ra ý tưởng.