

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



**BÁO CÁO ĐỒ ÁN 02  
Wumpus Game**

**Môn học: Cơ sở Trí tuệ Nhân tạo**

**Giảng viên hướng dẫn**

Bùi Duy Đăng  
Trần Quốc Huy  
Nguyễn Trần Duy Minh

**Nhóm 15**

**MSSV   Thành viên**

- 
- |          |                     |
|----------|---------------------|
| 21120291 | Nguyễn Đức Nam (NT) |
| 21120174 | Nguyễn Thị Gái      |
| 21120178 | Văn Bá Bảo Huy      |
| 21120180 | Nguyễn Bích Khuê    |

**Thành phố Hồ Chí Minh, 2023**

# Mục lục

	Trang
<b>Mục Lục .....</b>	<b>2</b>
<b>Danh sách Bảng.....</b>	<b>3</b>
<b>Danh sách Hình.....</b>	<b>4</b>
<b>CHAPTER I: Thông tin đồ án .....</b>	<b>5</b>
1. Mục tiêu đồ án .....	6
2. Bảng phân công công việc .....	6
3. Đánh giá mức độ hoàn thành đồ án .....	6
<b>CHAPTER II: Propositional Logic.....</b>	<b>7</b>
1. Knowledge Base sử dụng Resolution .....	7
1.1 Giới thiệu .. . . . .	7
1.2 Suy luận logic .. . . . .	7
1.3 Cài đặt .. . . . .	7
2. Knowledge Base dựa theo Backward .....	9
2.1 Giới thiệu .. . . . .	9
2.2 Suy diễn logic .. . . . .	9
2.3 Biểu diễn trong lập trình .. . . . .	10
3. Agent Logic.....	12
3.1 Giải thích .. . . . .	12
3.2 Quá trình suy luận agent .. . . . .	12
<b>CHAPTER III: Game.....</b>	<b>14</b>
1. Object .....	14
1.1 Cell .. . . . .	14
1.2 Map .. . . . .	15
1.3 Agent .. . . . .	15
2. Display .....	17

<b>CHAPTER IV: Kết quả .....</b>	<b>18</b>
<b>1. Xây dựng bản đồ và thực nghiệm.....</b>	<b>18</b>
1.1    Map 1 . . . . .	18
1.2    Map 2 . . . . .	18
1.3    Map 3 . . . . .	19
1.4    Map 4 . . . . .	19
1.5    Map 5 . . . . .	20
<b>2. Hạn chế.....</b>	<b>20</b>

# Danh sách bảng

I.1	Bảng phân công công việc . . . . .	6
I.2	Bảng đánh giá mức độ hoàn thành đồ án . . . . .	6

# Danh sách hình vẽ

III.1 Các trang trong giao diện chính của chương trình . . . . .	17
--	----

# Chapter I

## Thông tin đồ án

Cài đặt mô phỏng Wumpus World, trong đó sử dụng logic để điều khiển agent di chuyển trong Wumpus World.

Wumpus World bao gồm:

- Mạng lưới hang động (mảng 10x10) được kết nối với nhau.
- Nhứng căn phòng - room - 1 ô, có thể chứa các PIT (hố chết người), xung quanh các ô sẽ có gió (Breeze) để báo hiệu, hoặc Wumpus, xung quanh sẽ phát ra mùi hôi, và là vàng.
- Agent sẽ có mũi tên để bắn theo hướng chỉ định.
- Các di chuyển của agent như: tiến, lùi, trái phải 90°

Trong đồ án lần này, nhóm em sẽ chỉ sử dụng logic dạng logic mệnh đề (Propositional logic), và trình bày suy diễn logic theo 2 cách

- Resolution.
- Backward chaining.

Trong đồ án, đề yêu cầu khám phá Wumpus World, và lấy số điểm cao nhất với 3 điều kiện dừng:

- Agent chết.
- Agent giết toàn bộ Wumpus và lấy tất cả vàng trên bản đồ.
- Agent nhảy ra khỏi hang. (Hang tại vị trí bắt đầu).

Tuy nhiên trong bài này, nhóm em sẽ cố gắng hướng dẫn Wumpus, khám phá tất cả các ô có thể → giết hết quái, ăn hết vàng, đồng thời quay trở về ô ban đầu để nhảy ra khỏi hang.

## 1. Mục tiêu đồ án

- Phát triển kỹ năng làm việc nhóm.
- Áp dụng được kiến thức lý thuyết vào một chương trình cụ thể.
- Giảng viên đánh giá được mức độ kiến thức của sinh viên qua cài đặt, báo cáo v,v,..

## 2. Bảng phân công công việc

STT	Công việc	Thành viên tham gia
1	Đồ họa	Văn Bá Bảo Huy Nguyễn Thị Gái
2	KB - Resolution	Văn Bá Bảo Huy Nguyễn Thị Gái Nguyễn Bích Khuê
3	KB - Backward	Nguyễn Thị Gái Nguyễn Đức Nam
4	Logic	Nguyễn Bích Khuê Nguyễn Đức Nam
5	Viết báo cáo	Cả 4 thành viên

Bảng I.1: Bảng phân công công việc

## 3. Đánh giá mức độ hoàn thành đồ án

STT	Mô tả	Hoàn thành
1	<b>KB-Resolution</b>	100%
2	<b>KB-Backward</b>	100%
3	<b>Logic</b>	100%
4	Trình bày đồ họa	100%
5	Thiết kế 5 bản đồ theo độ khó tăng dần	100%
6	Viết báo cáo	100%

Bảng I.2: Bảng đánh giá mức độ hoàn thành đồ án

# Chapter II

## Propositional Logic

### 1. Knowledge Base sử dụng Resolution

#### 1.1 Giới thiệu

Phần này tập trung vào cách tri thức được biểu diễn trong Knowledge Base (KB) và việc sử dụng phương pháp suy luận logic dựa trên ý tưởng của Resolution.

#### 1.2 Suy luận logic

Suy luận logic dựa trên Resolution trong Knowledge Base (KB) như sau:

- Nếu trong Knowledge Base (KB) đã có thông tin rằng ô đó không phải PIT hoặc wumpus, chúng ta kết luận rằng ô này không phải là PIT hoặc wumpus.
- Để suy luận xem ô đó có thể là PIT hoặc wumpus, chúng ta sử dụng giải thuật Resolution dựa trên các quy tắc logic đã cài đặt trong hàm **inferenceP** và **inferenceW** trong phần cài đặt.

Nếu trong quá trình giải quyết xung đột logic giữa các điều kiện, chúng ta tìm thấy một hoặc một số điều kiện mà khi giải quyết suy luận cho ra một điều kiện trống (empty clause), có nghĩa rằng hệ thống logic đã chứa mâu thuẫn và chúng ta có thể kết luận rằng ô đó có thể là PIT hoặc wumpus.

Tiếp theo, nếu không tìm thấy điều kiện trống, chúng ta tiếp tục giải quyết xung đột logic để xác định các điều kiện mới có thể suy luận được từ Knowledge Base.

#### 1.3 Cài đặt

Phương thức **inferenceP**:

```
def inferenceP(self, neg_alpha):
    clause_list = copy.deepcopy(self.KB_P)

    if neg_alpha not in clause_list:
```

```

clause_list.append(neg_alpha)

while True:
    new_clauses = []

    for (clause_i, clause_j) in itertools.combinations(clause_list, 2):
        resolvents = self.resolve(clause_i, clause_j)
        if [] in resolvents:
            new_clauses.append([])
            return True

        for resolvent in resolvents:
            if self.is_valid_clause(resolvent):
                break
            if resolvent not in clause_list and resolvent not in new_clauses:
                new_clauses.append(resolvent)

    if not new_clauses:
        return False

    clause_list += new_clauses

```

- Đầu vào chúng ta nhận vào một **neg\_alpha** được thêm vào danh sách các điều kiện logic **clause\_list**
- Sau đó tạo một danh sách **new\_clause** để lưu các điều kiện mới suy luận được.
- Sau đó, duyệt qua tất cả các cặp điều kiện trong **clause\_list** để thực hiện giải quyết xung đột logic (resolvents).
- Nếu tìm thấy một resolvent trống thì thêm một danh sách trống vào **new\_clauses** và trả về **True** ngụ ý rằng suy luận thành công.
- Nếu không tìm thấy resolvent trống, thực hiện kiểm tra từng resolvent trong danh sách resolvents:
  - Nếu resolvent hợp lệ thì thoát khỏi vòng lặp.
  - Nếu resolvent không có trong **clause\_list** và cũng không có trong **new\_clauses**, thì thêm vào **new\_clauses**.

## Phương thức inferenceW

```

def inferenceW(self, neg_alpha):
    clause_list = copy.deepcopy(self.KB_W)

    if neg_alpha not in clause_list:
        clause_list.append(neg_alpha)

    while True:
        new_clauses = []

        for (clause_i, clause_j) in itertools.combinations(clause_list, 2):
            resolvents = self.resolve(clause_i, clause_j)
            if [] in resolvents:
                new_clauses.append([])

            return True

        for resolvent in resolvents:
            if self.is_valid_clause(resolvent):
                break
            if resolvent not in clause_list and resolvent not in new_clauses:
                new_clauses.append(resolvent)

        if not new_clauses:
            return False

        clause_list += new_clauses
    
```

- Ở phương thức này cài đặt tương tự inferenceP.

## 2. Knowledge Base dựa theo Backward

### 2.1 Giới thiệu

Trong phần này, ta sẽ trình bày về cách trình bày tri thức (Knowledge Base), và cách suy diễn logic, dựa vào ý tưởng của backward chaining.

### 2.2 Suy diễn logic

Dựa vào dạng trình bày Knowledge Base (KB) như trên, ta có thể áp dụng Backward chaining để suy diễn như sau:

Để suy ra được một ô có phải là PIT hay không, nếu ô đó không phải PIT đã có trong KB → ô này không phải PIT. Ngược lại ta dùng 2 cách sau:

- Nếu tất cả các ô xung quanh ô này đều là gió → PIT. Thực tế, một số trường hợp các ô xung quanh đều là gió nhưng ô này không phải PIT (vì một số ô PIT xung quanh tác động), nhưng, vì xung quanh đều là gió nên chúng ta không thể vào ô này được → xem như là một ô PIT, và hiện lên.
- Nếu một ô xung quanh đó là gió (Breeze), và 3 ô xung quanh khác của ô đó không phải là PIT, thì ô này là PIT.

Và để kiểm tra một ô có phải không là PIT hay không, ta chỉ cần nếu một ô xung quanh không phải gió Breeze thì ô này không phải PIT.

Tương tự, ta áp dụng với Wumpus, thay PIT bằng wumpus, breeze bằng stench.

### 2.3 Biểu diễn trong lập trình

Ta sẽ có hàm

```
| def inference(self, cell_pos, self_logic, type):
```

Trong đó:

- cell\_pos: ô cần kiểm tra.
- self\_logic: PL → dùng hàm của lớp PL.
- type: loại mà ô này cần kiểm tra, PIT, noPIT, Wumpus, noWumpus.

Dầu tiên ta cần kiểm tra đối ngược với điều trên có nằm trong KB chưa, nếu rồi thì là False. Biến lưu: neg\_alpha

```
| if neg_alpha in clause_list:  
    return False
```

Ta lấy các ô xung quanh của ô này, sau đó bắt đầu kiểm tra điều kiện.

```
| adj_cell = cell_pos.get_adjCells(self_logic.map)
```

- Nếu 4 ô xung quanh đều là Breeze (phải đã nằm trong KB) → PIT.
- Nếu 1 ô xung quanh là gió, và cả 3 ô xung quanh còn lại ô xung quanh này không phải PIT (đã nằm trong KB) → PIT.

```

T = True
for adj in adj_cell:
    temp = [adj.get_Literal(s_entities_bre, '+')]
    if temp not in clause_list:
        T = False
    else: #adj is breeze
        #B9 ^ 3 thang xung quanh B9 ko la pit --> P10
        T2 = True
        sub_adj_cell = adj.get_adjCells(self_logic_map)
        sub_adj_cell.remove(cell_pos)
        for sub_adj in sub_adj_cell:
            temp = [sub_adj.get_Literal(s_entities坑, 'L')]
            if temp not in clause_list:
                T2 = False
        if T2 == True:
            T = True
            break

```

Và, để kiểm tra 1 ô đó là không phải PIT, chỉ cần trong KB, có một ô xung quanh không phải Breeze → no PIT.

```

T = True
for adj in adj_cell:
    temp = [adj.get_Literal(s_entities_bre, 'L')]
    if temp not in clause_list:
        T = False
    return T
return T

```

### 3. Agent Logic

#### 3.1 Giải thích

Trong phần này, sẽ trình bày về cách agent sử dụng 2 loại KB, để di chuyển và khám phá được Wumpus World Game.

Ý tưởng chính: Ta sẽ liên tục dùng KB hiện tại, để suy ra được những ô xung quanh, ô nào là an toàn bằng phép suy diễn (phát hiện PIT, noPIT, Wumpus, no Wumpus), sau đó bắt đầu đi theo từng ô, tại vị trí tiếp theo lại tiếp tục suy các ô xung quanh, đi tiếp, cho đến khi không còn ô xung quanh nào an toàn để đi nữa, ta sẽ quay lại theo các ô đã đi trước đó (previous\_cell).

#### 3.2 Quá trình suy luận agent

Đầu tiên tại ô ban đầu, ta check các điều kiện như:

- Nếu vào trúng PIT, hoặc Wumpus → Agent chết.
- Nếu trong ô có vàng → lấy vàng.
- Nếu ô này chưa được khám phá (chưa đi trước đó) → đánh giá đã khám phá và thêm KB dựa vào ô này. (KB về PIT, noPIT, Wumpus, noWumpus - sẽ suy được KB của các ô xung quanh).

Lần lượt lấy các ô xung quanh, sau đó loại các ô đã khám phá (không đi lại những ô này), đồng thời kiểm tra nếu ô này, có nguy hiểm (là Stench, hoặc Breeze), thì suy luận các ô xung quanh như sau:

Nếu là Breeze:

- Kiểm tra nếu ô đó là PIT →, thêm vào ô đã khám phá (PIT thì không đi vào được), và thêm KB ô này là PIT.
- Kiểm tra nếu không là PIT, chỉ thêm vào ô này không phải PIT (vào được - chưa khám phá).
- Ngược lại, ô này là ô nguy hiểm (nghi vấn vì chưa đủ thông tin).

Tương tự đối với Wumpus:

- Kiểm tra nếu ô đó là Wumpus →, dùng cung tên bắn vào ô đó, và loại các stench xung quanh.

- Kiểm tra nếu không là Wumpus, chỉ thêm vào ô này không phải Wumpus (vào được - chưa khám phá).
- Ngược lại, ô này là ô nguy hiểm (nghi vấn vì chưa đủ thông tin).

Tuy nhiên, sau khi chạy, nhóm nhận thấy nếu chỉ khi nào đủ thông tin phát hiện Wumpus, thì mới bắn, thì nó sẽ rất lâu, nên nhóm chọn theo cách, nếu đã xét rồi, không đủ thông tin, thì ta sẽ bắt đầu bắn theo mọi hướng. (Tức đã kiểm tra Wumpus phía trên rồi, sau đó thấy vẫn còn Stench) → bắn đại, tương tự trúng sẽ bắt đầu loại bỏ các stench xung quanh.

Cuối cùng, ta lấy được các ô an toàn xung quanh, ta sẽ bắt đầu di chuyển vào một ô trước (nếu được đánh dấu chưa khám phá), lặp lại quá trình trên, cho đến khi không còn ô an toàn xung quanh → quay lui lại theo previous\_cell.

Tuy nhiên, trong quá trình chạy, nhóm nhận thấy, nếu chỉ đi theo previous\_cell, thì sẽ bị lặp, ví dụ như đi từ  $(0,0) \rightarrow (0,9)$ , lên  $(1,9) \rightarrow (1,0)$ , nếu đi theo quay lui thì sẽ đi hoàn toàn lại 20 bước, nhưng ta hoàn toàn có thể đi tắt theo từ  $(1,0) \rightarrow (0,0)$ , vì những ô trung gian đi sẽ không còn ý nghĩa.

Chú ý theo cách trên, nếu những ô trung gian, có những ô xung quanh mà chưa khám phá → đi quay lui theo những ô trung gian này → để khám phá hết tất cả các ô.

# Chapter III

## Game

### 1. Object

#### 1.1 Cell

Xây dựng lớp Cell để thể hiện cho từng vị trí (tầng phòng) ở trong thế giới Wumpus.

```
class Cell:  
    def __init__(self, pos_matrix, size, entities):  
        self.pos_matrix = pos_matrix  
        self.pos_coor = self.matrix_to_coordinate(pos_matrix, size)  
        self.map_size = size  
  
        self.explored = False  
        self.entities = self.set_cell_entites(entities)  
  
        self.previous = None  
        self.next_list = []
```

Lớp Cell được xây dựng bao gồm các thuộc tính quan trọng:

- pos\_matrix: Vị trí của ô trong ma trận.
- pos\_coor: Tọa độ của ô trong thế giới Wumpus (room).
- explored: Trạng thái của ô, đã được khám phá hay chưa.
- entities: Danh sách các thực thể trong ô (Gold, Pit, Wumpus, Breeze, Stench) được thể hiện bằng một dictionary với giá trị True và False.
- next\_list: Danh sách các ô kế tiếp (an toàn) có thể được di chuyển đến từ ô hiện tại được tìm kiếm thông qua quá trình xử lí.

Các phương thức chính của lớp:

- get\_Room: Trả về tọa độ trong thế giới Wumpus (room).

- set\_cell\_entites: Thiết lập danh sách thực thể trong ô dựa trên danh sách đầu vào.
- Phương thức kiểm tra các thực thể cụ thể như checkGold, checkPit, checkWumpus, checkBreeze, checkStench.
- setexploredCell: Đánh dấu ô là đã được khám phá.
- isSafe: Kiểm tra xem ô có an toàn để di chuyển không (không chứa Breeze hoặc Stench).
- get\_adjCells: Trả về danh sách các ô lân cận.
- get\_Literal: Tạo biểu diễn chữ cái của một thực thể trong ô dưới dạng chuỗi.
- Bao gồm cả hai phương thức xử lí trong quá trình di chuyển của Agent giữa các phòng bao gồm take\_Gold và kill\_Wumpus.

## 1.2 Map

Xây dựng lớp Map để chứa các thành phần tạo nên thế giới Wumpus hay là chứa các Cell thể hiện cho từng vị trí (từng phòng) trong thế giới Wumpus.

```
class Map:
    def __init__(self):
        self.map_size = 10
        self.cell_size = 60

        self.cell_unexplore = pygame.image.load(s_map_ele_unexplored)
        self.cell_explore = pygame.image.load(s_map_ele_explored)
        self.cell_pit = pygame.image.load(s_map_ele_pit)
        self.cell_exploreList = {key:None for key in list(s_map_ele_exploredList.keys())}
        self.loadExplored()

        self.MapCell = [[None]*self.map_size for _ in range(self.map_size)]

    self.initAgentPos = (9, 0)
    self.agentInit = None
```

Lớp Map bao gồm các thuộc tính được xác định mặc định bao gồm kích thước bản đồ, kích thước mỗi Cell và vị trí ban đầu của. Lớp Map bao gồm một ma trận các Cell thể hiện cho từng phòng trong thế giới Wumpus. Ngoài ra, lớp Map bao gồm hàm để xác định đồ họa hiển thị cho từng Cell dựa trên các thực thể tồn tại ở Cell đó.

## 1.3 Agent

Cài đặt lớp Agent:

```

class Agent(pygame.sprite.Sprite):
    def __init__(self, initPos):
        pygame.sprite.Sprite.__init__(self)
        #Graphic
        self.image = pygame.image.load(s_agent_right)
        self.imageList = [pygame.image.load(s_agent_up), ...]
        self.arrow = [pygame.image.load(s_agent_arrow_up), ...]

        self.position = initPos
        self.facing = s_agent_direction_right #[0-up, 1-down, 2-left, 3-right]

```

Lớp Agent được xây dựng để thể hiện cho người chơi trong thế giới Wumpus thực hiện các nhiệm vụ di chuyển và thực hiện các thao tác trong bản đồ thông qua các suy luận logic.

Lớp Agent được xây dựng kế thừa từ lớp pygame.sprite.Sprite của thư viện Pygame nhằm thuận tiện hơn cho việc thể hiện các đối tượng lên màn hình. Lớp được xây dựng bao gồm các thuộc tính chính thể hiện hình ảnh đồ họa của Agent, vị trí và các thông tin về hướng hiện tại.

Bao gồm các phương thức:

- agent\_appear: hiển thị Agent tại vị trí Cell (room) hiện tại lên giao diện.
- Các thao tác của agent bao gồm: turn (xoay), move (di chuyển) và shoot (bắn cung).

## 2. Display



Hình III.1: Các trang trong giao diện chính của chương trình

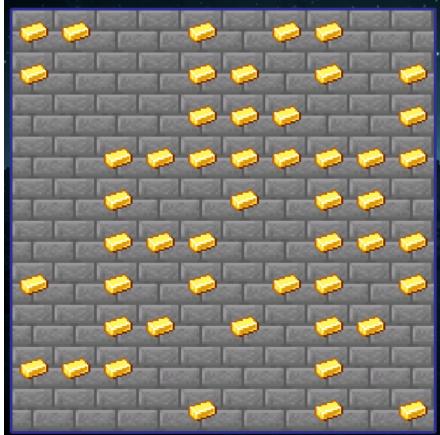
Trò chơi được xây dựng với 5 bản đồ (thế giới Wumpus) và 2 thuật toán giải Logic khác nhau cho người dùng lựa chọn.

# Chapter IV

## Kết quả

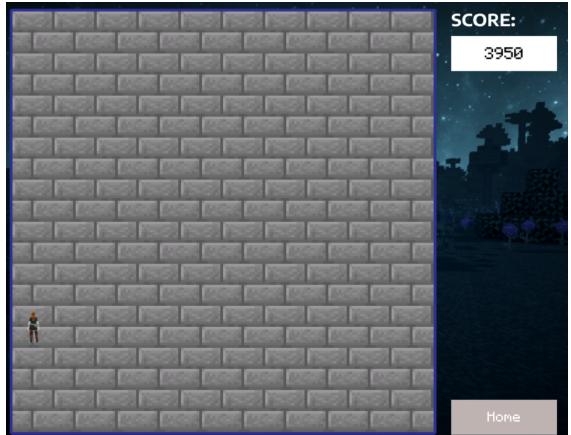
### 1. Xây dựng bản đồ và thực nghiệm

#### 1.1 Map 1



Map:

Kết quả chạy:



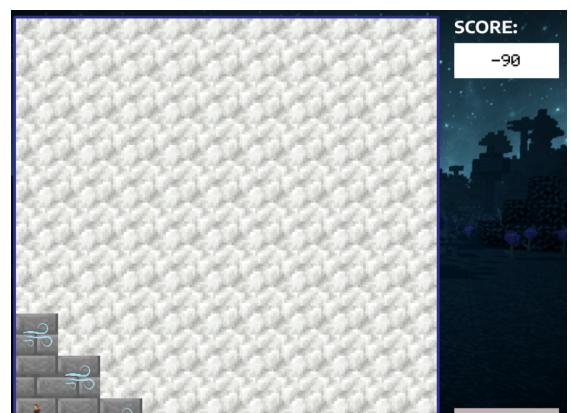
- Tác nhân thực hiện thu thập tất cả vàng.
- Sau khi khám phá tất cả các ô, tác nhân quay trở về vị trí xuất phát và leo ra khỏi hang.

#### 1.2 Map 2



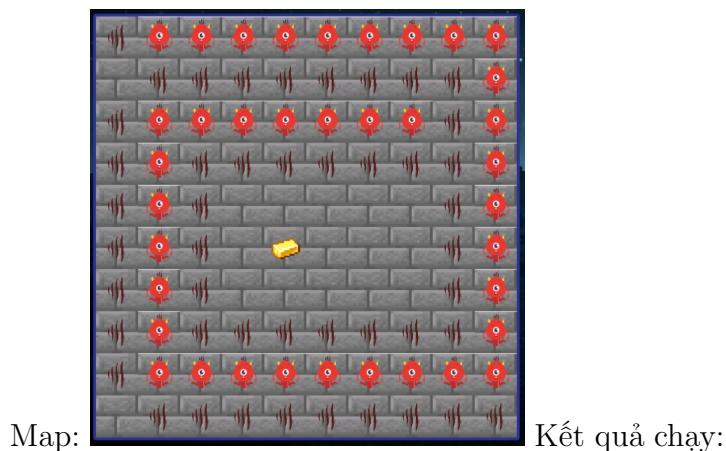
Map:

Kết quả chạy:



- Tác nhân thực hiện khám phá tất cả những ô an toàn.
- Khi không còn ô an toàn, tác nhân lựa chọn quay trở về và leo ra khỏi hang thay vì thử nghiệm nguy hiểm.

### 1.3 Map 3

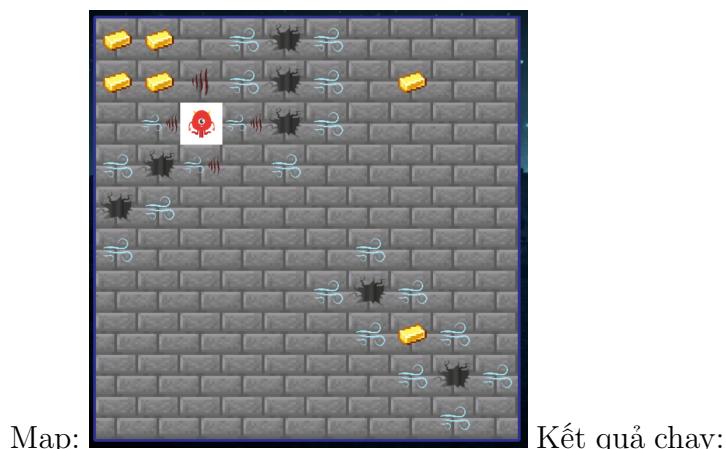


Kết quả chạy:

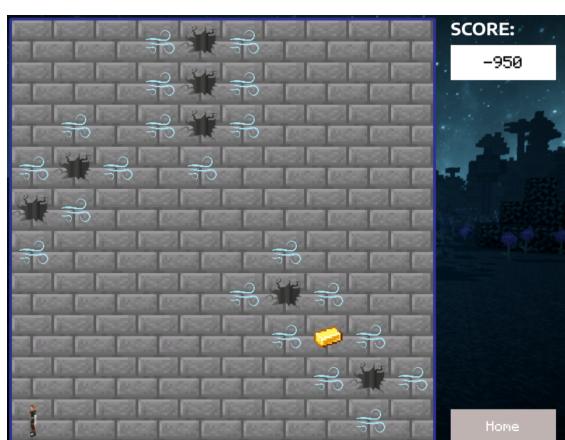


- Khi phát hiện mùi hôi thối tác nhân thực hiện suy luận và tìm ra Wumpus để thực hiện bắn. Tuy nhiên nếu chưa đủ kiến thức để suy luận, tác nhân sẽ thực hiện bắn tất cả những ô vẫn vắng để tiếp tục.
- Đối với map Wumpus thì tác nhân thực hiện giết hết Wumpus và thu thập hết vàng sau đó quay về và leo ra khỏi hang.

### 1.4 Map 4



Kết quả chạy:



- Đối với bản đồ này, thì ở vị trí của Wumpus chúng ta có thể thấy xung quanh nó bao gồm cả mùi hôi thối và gió. Tuy nhiên nếu bắn được Wumpus thành công thì theo quy luật (Wumpus và hố không thể ở cùng 1 ô) nên chúng ta có thể xác định ô là an toàn.

Từ đó có thể giải được và thu thập vàng ở góc trên cùng bên trái.

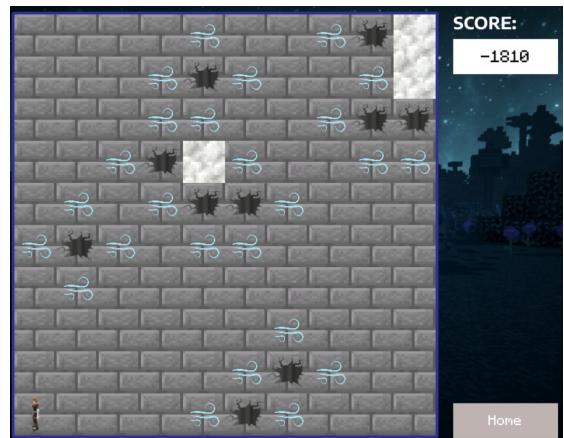
- Đối với vàng ở góc dưới bên phải, chúng ta có thể thấy nếu thay vị trí đó là 1 cái hố thì vẫn hợp lý nên khi chưa có cơ sở chắc chắn, tác nhân lựa chọn quay về chứ không khám phá thêm.

## 1.5 Map 5

Map:



Kết quả chạy:



- Tác nhân thực hiện khám phá tất cả các ô an toàn, thu thập vàng và giết Wumpus sau đó quay về vị trí ban đầu và leo ra khỏi hang.

## 2. Hạn chế

- Chưa áp dụng First Order Logic vào hướng dẫn agent.