TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN – ĐHQG TP.HCM
KHOA CÔNG NGHỆ THÔNG TIN

----------ଓ๑ଓ๑ଓ----------

BÁO CÁO ĐỒ ÁN THỰC HÀNH
CƠ SỞ TRÍ TUỆ NHÂN TẠO
PROJECT 02: LOGIC

- Nhóm thực hiện : 10
- Lớp : 21_4
- GVHD : Bùi Duy Đăng
  Trần Quốc Huy

MỤC LỤC

I. THÔNG TIN NHÓM

Thông tin các thành viên nhóm 10:

MSSV	Họ và tên
21120090	Mai Trần Phú Khương
21120123	Lê Thanh Thái Quảng
21120150	Nguyễn Song Toàn
21120169	Thái Chí Vỹ
21120411	Sần Dịch Anh

II. PHÂN CÔNG CÔNG VIỆC

MSSV	Công việc
21120090	- Xử lý in kết quả, trạng thái, suy luận ra màn hình - Làm báo cáo
21120123	- Xây dựng Knowledge base - Xử lí các trường hợp game - Làm báo cáo
21120150	- Xử lí kết thúc game - Xử lí các trường hợp game - Làm báo cáo
21120169	- Xử lý đồ họa cho game, vẽ các đối tượng. - Xử lý bắn tên. - Làm báo cáo.
21120411	- Xử lí lớp Agent - Làm báo cáo

III. ĐÁNH GIÁ MỨC ĐỘ HOÀN THÀNH

STT	Công việc	Điểm	Chi tiết
1	Hoàn thành mã nguồn	**45%** / 50%	Agent có thể di chuyển đến các ô an toàn nhờ suy luận dựa vào KB, tuy nhiên chưa tối ưu được cách bắn tên ăn quái (chỉ bắn theo hướng trước mặt khi gặp Stench).

2	Minh họa bằng đồ họa từng bước của quá trình chạy. Bạn có thể demo trong màn hình bảng điều khiển hoặc sử dụng bất kỳ thư viện đồ họa nào khác.	**10%** / 10%	Sử dụng pygame để minh họa từng bước di chuyển của Agent (Xem chi tiết trong phần V – Demo).
3	Tạo ra ít nhất 5 maps với cấu trúc khác nhau.	**20%** / 20%	Tạo ra 6 map với các trường hợp: - Map 1: Không ăn được hết vàng và không giết đươc hết quái. - Map 2, 5: Ăn hết vàng và giết hết quái. - Map 3, 4: Ăn hết vàng nhưng không giết hết quái. - Map 6: Bị chặn bởi quái
4	Báo cáo	**20%** / 20%	Hoàn thành báo cáo.
Total		**95%** / 100%	

IV. MÔ TẢ CHƯƠNG TRÌNH

1. Cơ sở lý thuyết về Logic

1.1. Logic vị từ (Propositional Logic)

1.1.1. Giới thiệu

Logic Vị Từ, hay Propositional Logic, là một phần quan trọng trong lĩnh vực Trí Tuệ Nhân Tạo (AI), tập trung vào việc xử lý các mệnh đề đơn lẻ và cách chúng kết hợp để tạo thành những mệnh đề phức tạp hơn. Điều này giúp máy tính hiểu và xử lý thông tin logic, làm nền tảng cho các hệ thống AI suy luận

1.1.2. Các thuật toán ứng dụng Suy luận logic của logic vị từ

- Biểu Diễn Mệnh Đề:

  Các biểu diễn trong logic vị từ sử dụng các biểu tượng và ký hiệu để đại diện cho các mệnh đề và các phép toán logic.

  Ví dụ: Biểu diễn mệnh đề A: "Ngày nay là thứ Ba."

- Phép Toán Logic Cơ Bản:
  Phủ định (¬): Biểu diễn mệnh đề phủ nhận.
  Và (∧): Biểu diễn mệnh đề hợp nhất, đúng khi cả hai mệnh đề đều đúng.
  Hoặc (∨): Biểu diễn mệnh đề lẻ tùy, đúng khi ít nhất một trong hai mệnh đề đúng.
  Nếu... thì... (→): Biểu diễn mệnh đề tuyến tính.
- Sự Kết Hợp Của Mệnh Đề:
  Các mệnh đề có thể kết hợp với nhau để tạo thành mệnh đề phức tạp bằng cách sử dụng các phép toán logic.
  Ví dụ: A∧(B∨¬C) biểu diễn mệnh đề phức tạp.
- Chân Lý Tổ Hợp (Truth Tables):
  Sử dụng bảng chân lý tổ hợp để mô tả giá trị đúng/sai của mệnh đề phức tạp dựa trên giá trị của các mệnh đề cơ bản.
- Suy Luận Logic:
  Logic vị từ cung cấp cơ sở cho suy luận logic, nơi có thể áp dụng các quy tắc suy luận để tìm ra các kết luận mới từ các mệnh đề đã biết

1.1.3. Các khái niệm cơ bản

- Thuật toán Hợp Unification và Hợp Giải (Unification and Resolution):
  Mô Tả: Kết hợp giữa Unification (thống nhất) và Resolution giúp giải quyết vấn đề của sự không chắc chắn (uncertainty) trong lĩnh vực Logic dựa trên mô hình Probabilistic Logic.
  Ứng Dụng: Trong các hệ thống AI đòi hỏi xử lý thông tin không chắc chắn, ví dụ như hệ thống chẩn đoán y tế dựa trên dữ liệu không đầy đủ.
- Thuật toán Giải Quyết Mệnh Đề (Resolution):
  Mô Tả: Resolution là một thuật toán giải quyết mệnh đề được sử dụng để suy luận về sự đúng/sai của một mệnh đề phức tạp dựa trên các mệnh đề đơn.
  Ứng Dụng: Trong hệ thống AI, Resolution được sử dụng để suy luận và giải quyết các vấn đề liên quan đến thông tin mà máy tính có thể rút ra từ các quy tắc và dữ liệu đầu vào.

1.2. Thuật toán hợp giải (Resolution):

1.2.1. Nguyên tắc hoạt động

Thuật toán hợp giải (Resolution) là một phương pháp quan trọng trong Logic Vị Từ để kiểm tra sự hợp lý của các mệnh đề và tìm ra các suy luận mới.

Dưới đây là nguyên tắc hoạt động cơ bản của thuật toán:

- Điều Kiện Thực Hiện: Để thực hiện quy tắc resolution, cần có ít nhất một cặp mệnh đề có một mệnh đề con và phủ định của nó.
- Quy Tắc Resolution: Resolution giúp kiểm tra tính đúng đắn của mệnh đề bằng cách loại bỏ một số mệnh đề dựa trên quy tắc:
  - Chọn Cặp Mệnh Đề: Chọn một cặp mệnh đề có ít nhất một mệnh đề con và phủ định của nó.
  - Loại Bỏ Mệnh Đề Con và Phủ Định: Loại bỏ cả mệnh đề con và phủ định của nó khỏi các mệnh đề ban đầu.
  - Thêm Mệnh Đề Mới: Thêm vào mệnh đề mới được tạo ra từ mệnh đề còn lại sau bước loại bỏ.

1.2.2. Ứng dụng vào Wumpus

- Bài toán Thế Giới Wumpus là một bài toán kinh điển trong lĩnh vực Trí Tuệ Nhân Tạo, nơi một tác nhân (Agent) cần khám phá một môi trường mà có thể chứa các chướng ngại vật như hố (pit), quái vật Wumpus và vàng. Sử dụng thuật toán resolution trong Logic Vị Từ có thể giúp tăng khả năng suy luận và ra quyết định thông minh cho tác nhân.
- Thuật toán resolution có thể được áp dụng trong bài toán Thế Giới Wumpus như sau:
- Biểu Diễn Môi Trường: Sử dụng logic vị từ để biểu diễn môi trường với các mệnh đề như "có hố tại ô (x, y)", "có Wumpus tại ô (x, y)", "có vàng tại ô (x, y)".
- Tri Thức của Agent: Tại mỗi bước, Agent thu thập thông tin về môi trường xung quanh và cập nhật tri thức của mình, tức là Knowledge Base, với các mệnh đề mới.
- Kiểm Tra Tính Đúng Đắn: Sử dụng resolution để kiểm tra tính đúng đắn của tri thức. Nếu hệ thống logic cho ra kết quả là True, thì tri thức được xem xét là hợp lý.

- Suy Luận Thêm Thông Tin: Nếu có thêm mệnh đề được suy luận từ quá trình resolution, thêm vào tri thức để tăng khả năng ra quyết định của Agent.

2. Môi trường và ngôn ngữ sử dụng

- Sử dụng ngôn ngữ Python.
- Interpreter Python 3.9 trở lên.

3. Ý tưởng chính

Chương trình gồm Agent và các chướng ngại vật ngăn cản bước đi của Agent, và tại các ô kề các chướng ngại vật sẽ có các dấu hiệu để nhận biết loại chướng ngại vật nào. Để Agent có thể tìm ra được nước đi tiếp theo cần phải xây dựng cho Agent một Knowledge Base (sẽ trình bày rõ cách cài đặt ở phần sau). Khi đi đến một vị trí trên bản đồ Agent sẽ có được thêm một dấu hiệu nhận biết về môi trường xung quanh (hay còn gọi là tri thức) và sẽ cho dấu hiệu đó vào Knowledge Base của Agent. Và Agent cũng dùng Knowledge Base đó để xác định được vị trí an toàn tiếp theo mà Agent có thể đi đến.

4. Mô tả chương trình

4.1. Mô tả tổng quan về chương trình

4.1.1. Những thư viện sử dụng

- pygame: thư viện dùng để vẽ đồ họa cho game, cũng như các game board.
- sys: để dừng chương trình khi game kết thúc.
- random: dùng để chọn nước đi ngẫu nhiên nếu bị chặn.

4.1.2. Class Agent

- Class Agent dùng để đại diện cho nhân vật chính trong game Wumpus World. Class này dùng để lưu trạng thái trước đó, trạng thái hiện tại, hướng mặt của Agent, các hành động đã thực hiện, các ô chưa khám phá, đường đi, điểm số của agent, số lượng mũi tên agent đang mang (vô hạn).

4.1.3. Class KnowledgeBase

- Class này dùng để giữ thông tin về Knowledge mà agent học được từ môi trường. Class này lưu các mệnh đề agent đã biết, các ô an toàn, các ô có hố, các ô có Wumpus, các ô có Vàng.

4.1.4. Class World

- Class World dùng để đọc map từ file.txt, lưu các attributes quan trọng của game và xử lý di chuyển cho agent, cũng như là các hành động như nhặt vàng, tiêu diệt Wumpus, ...

4.1.5. Class Tile

- Class Tile dùng để lưu trạng thái cho từng ô trên map như có tồn tại Wumpus, Pit, Gold hay các đặc điểm nhận biết.

4.1.6. Các Class Graphics

- Chứa các class con để tạo gameboard, tạo map và xử lý event game:
  - Class GameWindow: class chính dùng để tạo các gameboard của game, xử lý các event sự kiện trong game.
  - Class GameBoard và Button (kế thừa class GameBoard): dùng để vẽ các gameboard và button Step (cho biết các bước đi của agent).
  - Class World (kế thừa GameBoard): dành riêng cho gameboard chính, dùng để tạo map và các đối tượng trên map.

4.2. Class Agent

4.2.1. Thuộc tính (Attributes)

- *self.pre_state*: Lưu trạng thái trước đó của agent.
- *self.state*: Lưu trạng thái hiện tại của agent.
- *self.face*: Lưu hướng mặt của agent (R - Right, L - Left, U - Up, D - Down).
- *self.actions*: Danh sách lưu các hành động mà agent đã thực hiện.
- *self.unexplored*: Danh sách lưu các ô chưa được khám phá.
- *self.path*: Danh sách lưu đường đi của agent.
- *self.score*: Điểm số của agent.
- *self.arrows*: Số lượng mũi tên mà agent đang mang theo (vô hạn trong ngữ cảnh của đề bài).

4.2.2. Phương thức (Method)

- *get_neighbour(self, size)*: Trả về các ô lân cận của agent dựa trên kích thước map.

- *add_actions(self, new_state, size)*: Thêm hành động dựa trên trạng thái mới và kích thước map.
- *clear_agent(self)*: Reset thông tin của agent về trạng thái ban đầu.
- *print_agent(self)*: In thông tin về agent.
- *update_score(self, event)*: Cập nhật điểm số dựa trên sự kiện (ví dụ: nhặt vàng, bắn mũi tên, ...).

4.2.3. Phương thức: Hàm add_actions

- Input:
  - *new_state*: Trạng thái mới của agent, là một tuple đặc trưng cho vị trí mới của agent trong map.
  - *size*: Kích thước của map (được sử dụng để kiểm tra giới hạn vị trí).
- Output:
  - Không có giá trị trả về (hàm chỉ thực hiện thay đổi trạng thái của agent).
- Logic:
  - Xác định hướng mới của agent dựa trên sự thay đổi vị trí giữa *state* hiện tại và *new_state*.
  - Nếu hướng mới giống với hướng hiện tại (***face***), thêm một bước di chuyển mới với ***new_state*** và ***face*** vào danh sách ***actions***.
  - Nếu hướng mới khác với hướng hiện tại:
    - Thay đổi hướng của agent thành hướng mới.
    - Thêm một bước di chuyển mới từ *state* hiện tại đến *new_state* với hướng mới vào danh sách *actions*.
    - Thêm một bước di chuyển mới từ *new_state* đến *new_state* với hướng mới vào danh sách *actions*.

4.3. Class KnowledgeBase

4.3.1. Thuộc tính (Attributes)

- *self.sentence*: có kiểu dữ liệu là list, chứa các phần tử có kiểu dữ liệu là dict có *key=state* và value mô tả các ô có vị trí là state có thể là pit (value = 1), wumpus (value = 2), pit hoặc wumpus (value = 3)
- *self.safes*: có kiểu dữ liệu là list, chứa các cặp số (x, y) là vị trí của các ô an toàn mà Agent có thể di chuyển tới
- *self.pits*: có kiểu dữ liệu là list, chứa các cặp số (x, y) là vị trí của các pit

- *self.wumpus*: có kiểu dữ liệu là list, chứa các cặp số (x, y) là vị trí của các wumpus

4.3.2. Phương thức (Method)

- *self.add(sentence)*: là phương thức để thêm một sentence (dấu hiệu) mới vào *self.sentence*
- *self.inference()*: là phương thức để áp dụng thuật toán hợp giải, để suy diễn ra những tri thức mới rồi thêm vào *self.safes*, *self.pits*, *self.wumpus* sao cho thích hợp

4.3.3. Phương thức: Hàm add(sentence)

- Input: là một dãy có kiểu dữ liệu là dict mô tả vị trí của pits hoặc wumpus, cụ thể: Nếu như đến một vị trí (x, y) và vị trí (x, y) có các vị trí xung quanh hợp lệ là (x1, y1), (x2, y2):
  - Nếu dấu hiệu tại (x, y) là 'B' thì thêm *sentence = {(x1, y1): 1, (x2, y2): 1}*.
  - Nếu dấu hiệu tại (x, y) là 'S' thì thêm *sentence = {(x1, y1): 2, (x2, y2): 2}*.
  - Nếu dấu hiệu tại (x, y) là 'B' thì thêm *sentence = {(x1, y1): 1, (x2, y2): 3}*.
- Output: không có trả về kết quả
- Mô tả hàm chi tiết
  - Kiểm tra sentence đã có ở trong *self.sentence* hay chưa.
  - Nếu chưa có trong *self.sentence* thì thêm sentence giống mô tả ở trên vào *self.sentence* bằng cách *self.sentence.append(sentence)*.

4.3.4. Phương thức: Hàm inference()

- Input: Không nhân giá trị đầu vào
- Output: Không trả về kết quả
- Mô tả hàm chi tiết
  - Với mỗi sentence trong *self.sentence* (mỗi sentence có state và value).
  - Nếu như state có trong *self.safes* (tức là vị trí đó là vị trí đã biết chắc là an toàn) thì xóa vị trí này trong sentence.
  - Nếu như state đó đang có giá trị là 3 mà state ở trong *self.pits* thì cập nhật lại giá trị của state đó là 1.

- Nếu như state đó đang có giá trị là 3 mà state ở trong *self.wumpus* thì cập nhật lại giá trị của state đó là 2.
- Nếu như sentence đó có độ dài là 1.
- Nếu giá trị của state đó bằng 1 và state chưa có trong *self.pits* thì thêm state đó vào *self.pits*.
- Nếu giá trị của state đó bằng 2 và state chưa có trong *self.wumpus* thì thêm state đó vào *self.wumpus*.

4.4. Class World

4.4.1. Thuộc tính

- *self.height* và *self.width*: có kiểu dữ liệu là int, lưu kích thước của map.
- *self.numGold*, *self.numPit* và *self.numWumpus*: có kiểu dữ liệu là int, lần lượt lưu số lượng còn lại của các đối tượng Gold, Pit và Wumpus.
- *self.listTiles*: có dạng ma trận, lưu tất cả các tiles cùng với trạng thái từng tile.
- *self.doorPos*: lưu tuple (x, y) là vị trí của cửa thoát.
- *self.agent* và *self.kb*: khởi tạo agent và knowledge base.
- *self.path* và *self.actions*: lưu đường đi và hành động của agent.
- *self.end_game*, *self.end_game_eaten*, *self.end_game_fall* và *self.end_game_full*: kiểu dữ liệu là boolean, dùng để xử lí trạng thái của game (thắng hoặc thua, thua do hết đường, rơi hố, wumpus ăn).
- *self.way_back*: kiểu dữ liệu là list, lưu đường đi ngắn nhất để trở về cửa thoát sau khi kết thúc game.

4.4.2. Phương thức:

- *self.neighbor(x, y)*: là phương thức trả về vị trí các ô lân cận của vị trí agent đang đứng ((x, y)).
- *self.direction(neighbor_state)*: là phương thức trả về hướng của agent đối với ô lân cận neighbor.
- *self.readMap(filename)*: là phương thức để đọc map với file map tên là filename.
- Các phương thức dành cho các tác vụ trong game:
  - *self.grabGold(i, j)*: nhặt vàng tại ô có tọa độ (i, j).
  - *self.killWumpus(i, j)*: bắn cung tiêu diệt Wumpus tại ô có tọa độ (i, j).

- *self.moveAgent(before_pos, after_pos)*: di chuyển agent từ vị trí before_pos đến vị trí aften_pos.

- Các phương thức kiểm tra end game:
  - *self.leftGold()*: kiểm tra trên map còn ô nào chứa vàng hay không.
  - *self.leftWumpus()*: kiểm tra trên map còn ô nào có Wumpus hay không.
- *self.findNextMove()*: dùng để xử lý di chuyển; làm các tác vụ như nhặt vàng, tiêu diệt Wumpus; và tìm bước tiếp theo.
- *self.findWayBack()*: dùng thuật toán GBFS để tìm đường về cửa thoát ngắn nhất, tương tự tìm nước tiếp theo trong *self.findNextMove()* ở phần tìm trong old_next

4.4.3. Phương thức: Hàm readMap(filename)

- Input: filename (string).
- Output: Không trả về giá trị.
- Mô tả hàm chi tiết:
  - Đọc file với chế độ read và dùng hàm **.splitlines()** để chia thành các dòng.
  - Bỏ dòng đầu (dòng lưu kích thước map), sau đó lưu kích thước cho 2 attribute **self.height** và **self.width**.
  - Chia nhỏ từng dòng với **.split('.')** và lưu giá trị vào ma trận **tiles**.
  - Khởi tạo từng *tile* cho ma trận **self.listTiles**.
  - Đọc giá trị từ ma trận **tiles** và lưu trạng thái cho từng *tile* trong ma trận **self.listTiles**.

4.4.4. Phương thức: Hàm findNextMove()

- Input: Không nhân giá trị đầu vào
- Output: Không trả về giá trị
- Mô tả hàm chỉ tiết
  - Gọi hàm *self.handleEndGame()*, nếu không kết thúc game
  - Khởi tạo *pre_state = self.agent.pre_state* để lưu vị trí trước của agent, khởi tạo *current_state = self.agent.next_state* để lưu vị trí hiện tại của agent.
  - Nếu *current_state* có giá trị (tức là đã tìm được nước đi khi gọi hàm) thì:

+ Thêm *current_state* vào actions của agent.
+ Cập nhật *self.agent.state = current_state.*
+ Đánh dấu *self.listTile* tại vị trí hiện tại *current_state* đã đi qua.

- Nếu *current_state* chưa có giá trị thì gán giá trị là *self.doorPos*, tức là từ cửa đi ra tại vị trí cửa
- Nếu như vị trí hiện tại *current_state* có dấu hiệu là Stench thì bắn mũi tên theo hướng trước mặt, nếu nước mặt là tường thì bắn theo một hướng khác
- Gọi hàm *self.moveAgent(pre_state, current_state)* để di chuyển agent đến vị trí hiện tại.
- Khởi tạo *neighbours* chính là các vị trí trên, dưới, trái, phải hợp lệ của vị trí hiện tại.
- Nếu *current_state* không có trong *self.kb.safes* thì thêm current_state vào và gọi hàm *self.kb.inference()* để suy luận khi có thêm tri thức mới.
- Nếu vị trí hiện tại không có dấu hiệu của pits hay wumpus thì thêm neighbours vào trong *self.kb.safes* (nếu các vị trí của neighbour chưa có trong *self.kb.safes*) đồng thời thêm các vị trí vào *self.agent.unexplored* (nếu chưa có) và gọi hàm *self.kb.inference()* để suy luận khi có thêm tri thức mới.
- Nếu vị trí hiện tại chỉ có vàng (chỉ có dấu hiệu 'G') thì gọi hàm *self.grabGold()* để nhặt vằng và thêm neighbours vào *self.kb.safes* (nếu các vị trí của neighbour chưa có trong *self.kb.safes*) và gọi hàm *self.kb.inference()* để suy luận khi có thêm tri thức mới.
- Nếu vị trí hiện tại có dấu hiệu có vàng, và có thêm các dấu hiệu khác thì chỉ gọi hàm *self.grabGold()* để nhặt vàng.
- Nếu vị trí hiện tại xuất hiện dấu hiệu 'BS', thì thêm sentence với key là vị trí trong *neighbours* và value = 3 vào *self.kb.sentence* bằng cách gọi hàm *self.kb.add*.
- Nếu vị trí hiện tại xuất hiện dấu hiệu 'B', thì thêm sentence với key là vị trí trong *neighbours* và value = 1 vào *self.kb.sentence* bằng cách gọi hàm *self.kb.add*.

- Nếu vị trí hiện tại xuất hiện dấu hiệu 'S', thì thêm sentence với key là vị trí trong *neighbours* và value = 2 vào *self.kb.sentence* bằng cách gọi hàm *self.kb.add*.
- Tạo *new_next* là các vị trí tiếp theo mà an toàn và chưa có trong đường đi của agent.
- Tạo *old_next* là các vị trí tiếp theo mà an toàn và có trong đường đi của agent.
- Chọn vị trí tiếp theo (*next_state*) chính là vị trí có trong *new_next* và cùng hướng quay mặt của agent, nếu không chọn được trong *new_next* thì chọn trong trong *old_next*.
- Chọn vị trí tiếp theo trong *old_next* bằng cách dùng thuật toán GBFS, tìm ra đường đến vị trí có trong *self.agent.unexplored* (các ô an toàn nhưng chưa khám phá) gần nhất
- Cập nhật *self.agent.pre_state=current_state*, *self.agent.next_state = next_state*.
- Nếu *self.handleEndGame()* trả về kết quả kết thúc thúc thì
- Nếu *self.end_game_full*, tức là kết thúc game khi đã ăn hết vàng và giết hết wumpus thì in kết quả và gọi hàm *self.findWayBack()* để tìm đường về cửa
- Nếu *self.end_game_eaten*, tức là bị quái ăn thì in ra kết quả
- Nếu *self.end_game_fall*, tức là bị roi vào hố thì in kết quả

4.5. Các class về graphics

4.5.1. Các class con

4.5.1.1. GameBoard: Class GameBoard dùng để vẽ các gameboard.

4.5.1.2. Button: kế thừa class GameBoard, dùng để tạo button Step, khi click vào sẽ di chuyển Agent và in ra bước đi và trạng thái của nó.

4.5.1.3. World: kế thừa class GameBoard, dùng để vẽ map chơi.

4.5.1.4. GameWindow: class chính dùng để tạo các gameboard của game, xử lý các event sự kiện trong game.

4.5.2. Các phương thức chính

4.5.2.1. Phương thức *World.generateWorld()*: dùng để vẽ map trên Main Gameboard.

- Input: Không nhân giá trị đầu vào.

- Output: Không trả về giá trị.
- Mô tả hàm chi tiết:
  - Đọc và kiểm tra trạng thái của từng ô, vẽ đối tượng tương ứng với trạng thái bằng hàm **self.draw_image(image_path, rect)**.
  - Sau khi vẽ xong các đối tượng, ta vẽ phủ lên các ô chưa được khám phá bằng các *CEIL* (gọi là trần nhà).

4.5.2.2. Phương thức *GameWindow.run_game()*: dùng để xử lý các event diễn ra trong game.

- Input: Không nhân giá trị đầu vào.
- Output: Không trả về giá trị.
- Mô tả hàm chi tiết:
  - Khởi tạo biến *clock* để điều chỉnh tốc độ render khung hình.
  - Xử lý event: Nếu button Step được nhấn, ta gọi hàm **findNextMove()** để di chuyển agent tới vị trí tiếp theo.
  - Vẽ các Gameboard lên cửa sổ game. Sau đó cập nhật lại game.

4.6. Class tile

4.6.1. Thuộc tính

- self.isPit: Cho biết một tile có là hố không
- self.isBreeze: Cho biết một tile có breeze không
- Self.isWumpus: Cho biết wumpus có trên tile không
- Self.numStench: Cho biết số stench trong tile
- Self.isGold: Cho biết tile có gold không
- Self.isAgent: Cho biết agent hiện đang ở tile đó không
- Self.isExplored: Cho biết tile đó đã được khám phá hay chưa

4.6.2. Phương thức

Các getter:

- getPit: Trả về giá trị của isPit
- getBreeze: Trả về giá trị của isBreeze
- getWumpus: Trả về giá trị của isWumpus
- getStench: Trả về false nếu số stench=0; hoặc true nếu số stench > 0
- getGold: Trả về giá trị của isGold
- getAgent: Trả về giá trị của isAgent

- getExplored: Trả về giá trị của isExplored

⇒ Phương thức cho phép truy cập từ bên ngoài của class Tile

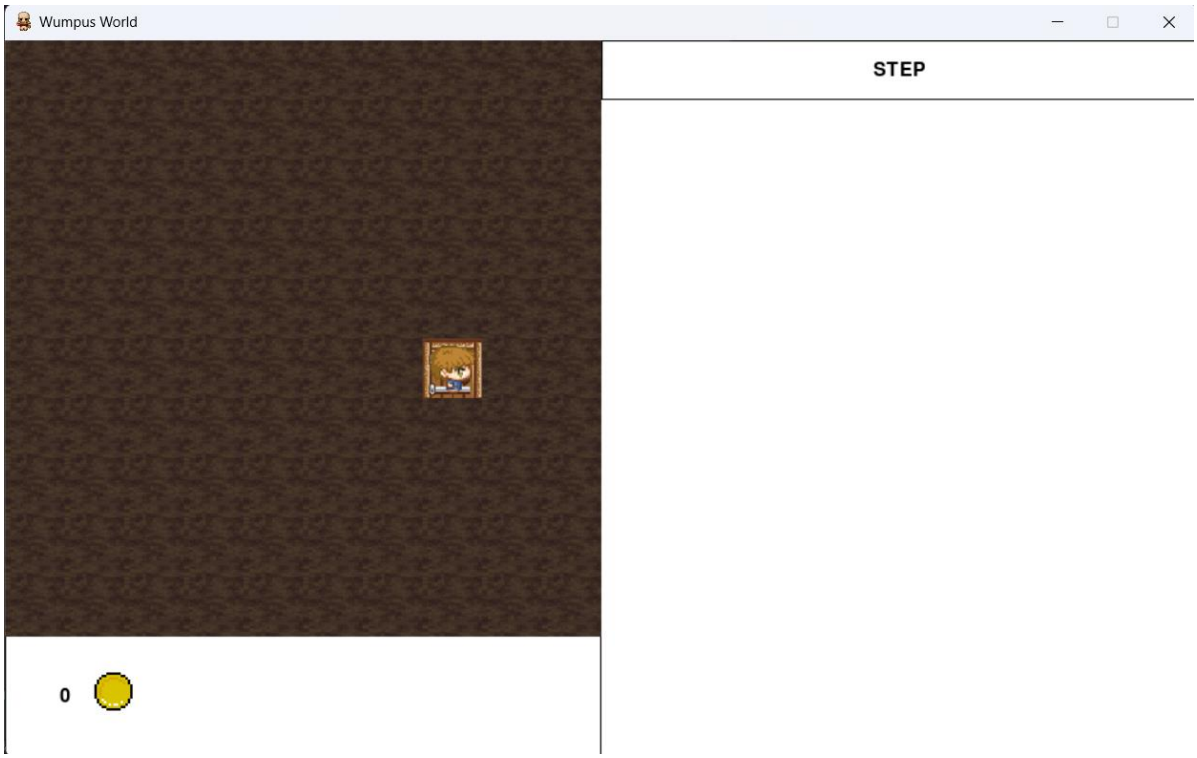
Các setter:

- setPit:Đặt isPit là true
- setBreeze: Đặt isBreeze là true
- setWumpus: Đặt isWumpus là true
- setStench: Tăng biến isStench thêm 1
- setGold: Đặt isGold là true.
- setAgent: Đặt isAgent là true
- setExplored: Đặt isExplored là true

⇒ Cung cấp cơ chế cập nhật trạng thái của tile từ bên ngoài class.

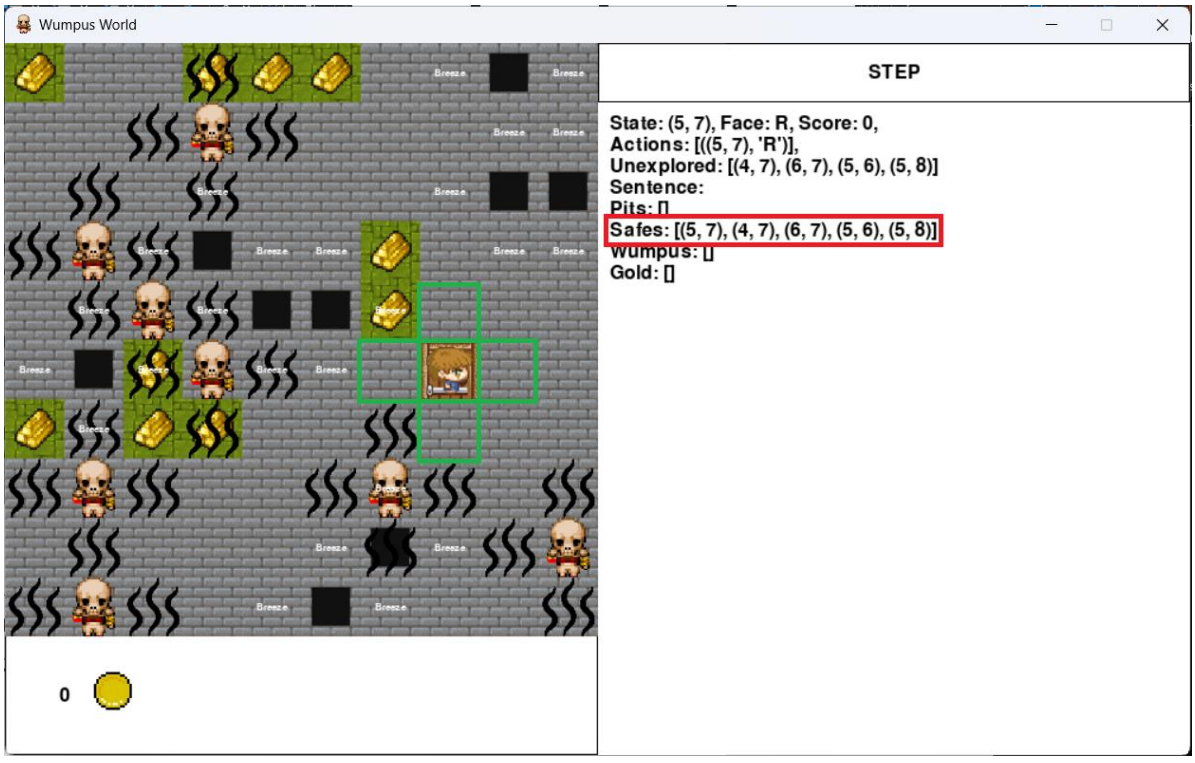
V. DEMO CHƯƠNG TRÌNH

- Giao diện của chương trình khi chọn map 4:



- Để thấy rõ hơn quá trình sử dụng các thuật toán thì sẽ mở hết các ô như sau:

- Tại một vị trí đứng của Agent, nếu như không có dấu hiệu gì thì Agent sẽ biết được ô đang đứng và các ô xung quanh là an toàn và sẽ đưa các vị trí an toàn vào một list để lưu



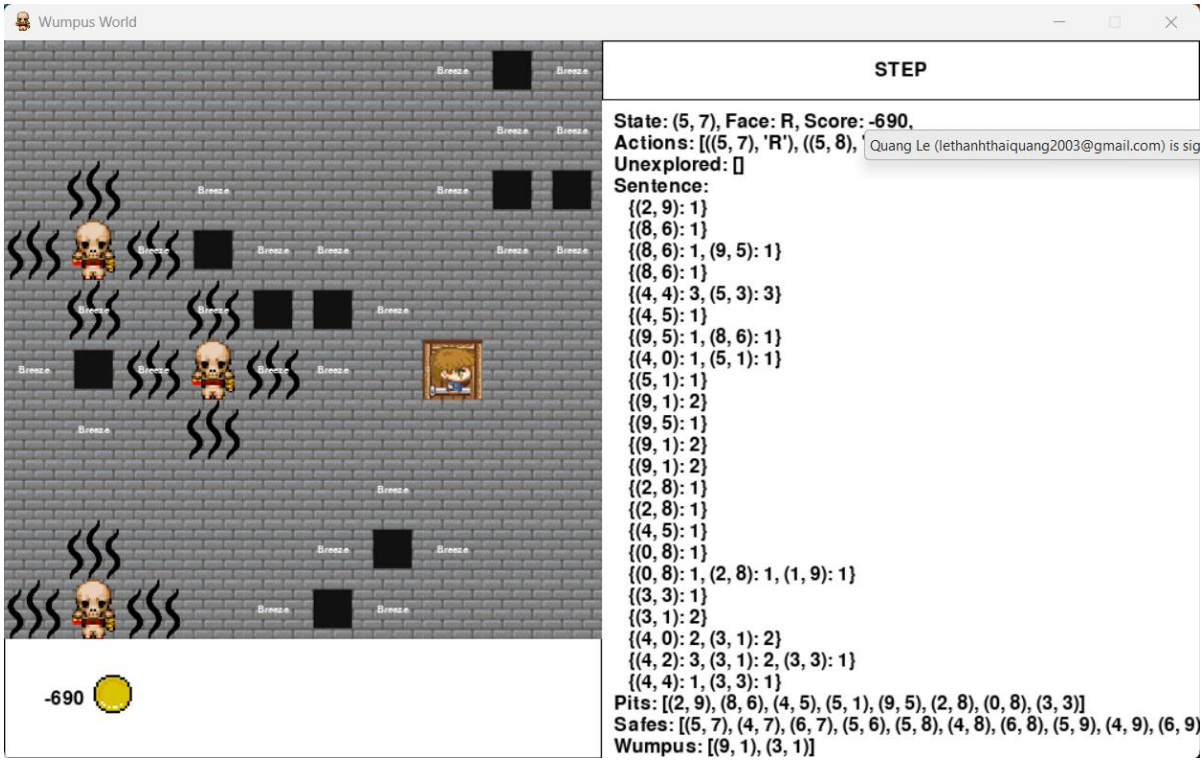
- Sau đó Agent sẽ lựa chọn vị trí an toàn tiếp theo để đi đến

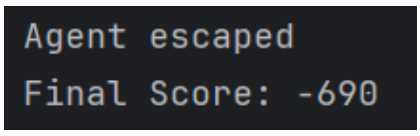
- Tại một vị trí đứng của Agent, nếu như có dấu hiệu là "Breeze" thì Agent sẽ đưa các vị trí nghi ngờ có Pit vào trong sentence. Trong trường hợp này, các vị trí nghi ngờ có Pit sẽ là (2, 9), (3, 8), (4, 9), tuy nhiên vị vị trí (4, 9) đã nằm trong danh sách an toàn nên sẽ loại bỏ (4, 9) khỏi nghi ngờ có Pit

- Tại vị trí đứng của Agent, nếu như có vàng thì sẽ nhặt vàng và thêm các dấu hiệu (nếu có), các ô an toàn suy luận được (nếu có) và thêm vị trí vàng vào danh sách

- Tại vị trí đứng của Agent, nếu như có dấu hiệu Stench, thì Agent sẽ bắn mũi tên theo hướng trước mặt, sau đó thêm các dấu hiệu. Trong trường hợp này, tiêu diệt được Wumpus nên vị trí Stench trở thành vị trí an toàn

- Khi Agent đi hết đường, thì quay về door và kết thúc





VI. CÁCH TỔ CHỨC VÀ THỰC THI CHƯƠNG TRÌNH

- Cây thư mục của project

```
./project
    |_/document
        |_/report.pdf
    |_/assets
        |_/agent_down.png
        |_/agent_up.png
        |_/agent_left.png
        |_/agent_right.png
        |_/agent_walk.png
        |_/arrow_down.png
        |_/arrow_up.png
        |_/arrow_left.png
        |_/arrow_right.png
        |_/ceil.png
        |_/wumpus.png
        |_/stench.png
        |_/pit.png
        |_/floor.png
        |_/floor_gold.png
        |_/gold.png
        |_/gold-icon.png
        |_/score_icon.png
        |_/terrain.png
        |_/door.png
    |_/input
        |_/map1.txt
        |_/map2.txt
        |_/map3.txt
        |_/map4.txt
        |_/map5.txt
        |_/map6.txt
    |_/source
        |_/agent.py
        |_/knowledgebase.py
        |_/tile.py
        |_/world.py
        |_/graphics.py
        |_/main.py
```

- Chạy chương trình (cần cài đặt trước thư viên pygame):

- Chạy file main.py.
- Chọn map (từ map1 đến map6).
- Click Step để di chuyển Agent.
- Khi trò chơi kết thúc xem kết quả cuối cùng ở Terminal

VII. TÀI LIỆU THAM KHẢO

[1] Các tài liệu tham khảo trên hệ thống Moodle môn học được cung cấp
[2] Github tham khảo