Ton Duc Thang University
Faculty of Information Technology

ĐỀ TÀI THUYẾT TRÌNH GIỮA KỲ

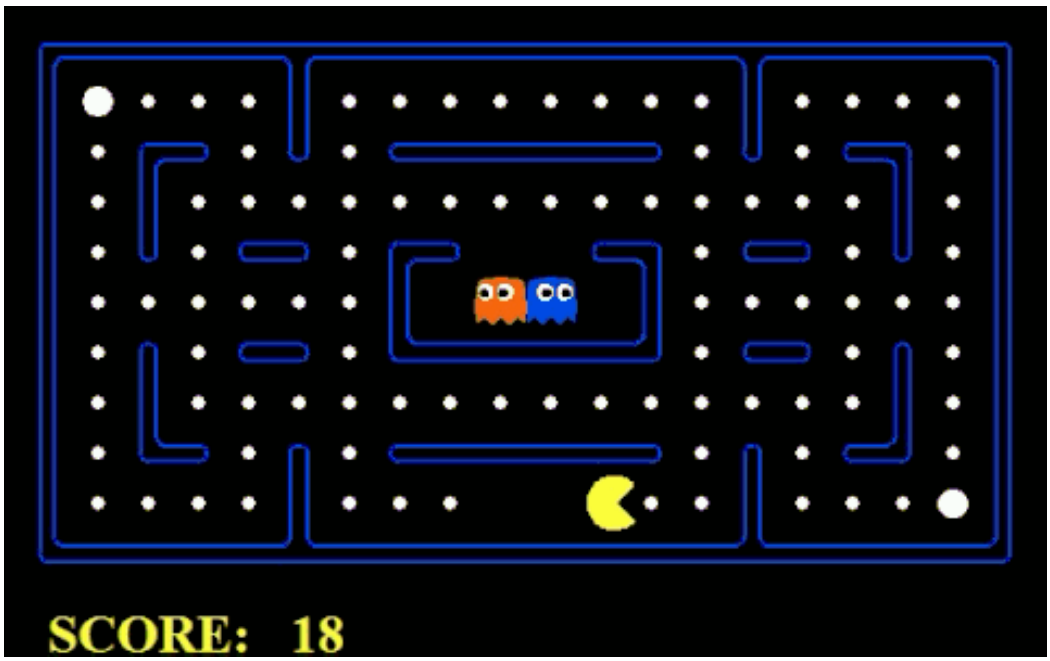
Môn: Nhập môn Trí tuệ Nhân tạo

Thời gian làm bài: 03 tuần

I. Hình thức

- Đề tài giữa kỳ được thực hiện theo nhóm **04** – **05** sinh viên.
- Nhóm sinh viên thực hiện các yêu cầu và nộp bài theo hướng dẫn chi tiết bên dưới.

II. Yêu cầu



Pacman game

Source: https://blog.sciencemuseum.org.uk/pac-man-turns-40/

a) Câu 1: Uninformed Search

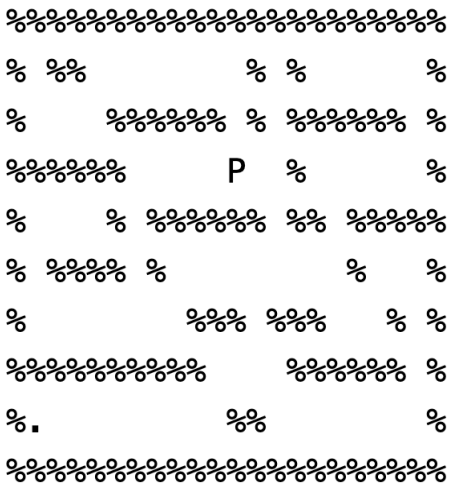
Trong câu này, sinh viên thực hiện cài đặt các giải thuật tìm kiếm mù (uninformed search) để giúp pacman di chuyển từ vị trí xuất phát (start) đến vị trí mồi (end).

Dữ liệu về bố cục mê cung được đọc lên từ một text file với cấu trúc như sau:

- **%** → vật cản, tường, không thể qua
- **P** → vị trí ban đầu của pacman
- **.** → vị trí điểm mồi

- Các ô trống còn lại pacman có thể đi qua.

Ví dụ:



File dữ liệu mê cung với một điểm mồi

YC1-1: Tạo tệp **problems.py**, sau đó cài đặt lớp **SingleFoodSearchProblem** để mô tả bài toán theo dạng Single-state problem. Trong đó mô tả chi tiết

- State, Node, Initial state
- Successor function
- Goal-test function
- Path-cost function
- Phương thức đọc mê cung từ tệp
- Phương thức in mê cung ra màn hình

YC1-2: Cài đặt, trong tệp **fringes.py**, các cấu trúc dữ liệu thường dùng để cài đặt thuật toán tìm kiếm, gồm:

- Stack
- Queue
- PriorityQueue

YC1-3: Tạo tệp **searchAgents.py** và cài đặt các hàm sau

- `bfs(problem) → list`
- `dfs(problem) → list`
- `ucs(problem) → list`

Các hàm trên nhận vào một tham số kiểu **SingleFoodSearchProblem** và trả về một list các bước di chuyển để pacman đi tới điểm mồi, gồm:

- **N** → đi lên trên
- **S** → đi xuống dưới
- **W** → đi sang trái
- **E** → đi sang phải
- **Stop** → đứng yên

Ví dụ giá trị trả về của hàm bfs() là `['N', 'N', 'S', 'W', 'E', 'E', 'Stop']`.

YC1-4: thêm vào lớp **SingleFoodSearchProblem** phương thức

```
animate(self, actions) → None
```

trong đó, actions là chuỗi bước đi chuyển có được từ các hàm tìm kiếm như bfs, dfs, ucs. Hàm này thực hiện thao tác:

- B1: xoá màn hình
- B2: in ra mê cung, vị trí hiện tại của pacman và điểm mồi, chờ người dùng nhấn Enter
- B3: quay về bước 1.

Mỗi bước lặp của hàm animate sẽ lấy ra action tiếp theo và di chuyển pacman tới vị trí tương ứng.

YC1-5: Thêm vào tệp **problems.py** một lớp **MultiFoodSearchProblem** để mô tả bài toán dạng Sing-state problem trong đó pacman cần phải **ăn hết các điểm mồi** trong mê cung.

- State, Node, Initial state
- Successor function
- Goal-test function
- Path-cost function
- Phương thức đọc mê cung từ tệp
- Phương thức in mê cung ra màn hình
- Phương thức `animate(self, actions)`

YC1-6: Chỉnh sửa các hàm ở **YC1-3** lên mức tổng quát để có thể hoạt động đúng với problem là **SingleFoodSearchProblem** hoặc **MultiFoodSearchProblem**.

Tiêu chí	Điểm
YC1-1	**0.5 điểm**
YC1-2	**1.5 điểm**
YC1-3	**1.5 điểm**
YC1-4	**0.5 điểm**
YC1-5	**0.5 điểm**
YC1-6	**0.5 điểm**
Tổng	**5.0 điểm**

b) Câu 2: Best-Firest Search

YC2-1: Cài đặt, trong tệp **searchAgents.py** ít nhất 02 hàm heuristic để ước lượng chi phí từ trạng thái hiện tại tới trạng thái đích cho bài toán **SingleFoodSearchProblem**.

- Tham số: state (trạng thái hiện tại)
- Trả về: giá trị heuristic (số nguyên/thực)

Giải thích trong bài thuyết trình về tính ***admissible*** và ***consistent*** của hai hàm heuristic trên.

YC2-2: Cài đặt, trong tệp **searchAgents.py**, ít nhất 01 hàm heuristic để ước lượng chi phí từ trạng thái hiện tại tới trạng thái đích cho bài toán **MultiFoodSearchProblem**.

- Tham số: state (trạng thái hiện tại)
- Trả về: giá trị heuristic (số nguyên/thực)

YC2-3: Cài đặt, trong tệp **searchAgents.py**, hàm

```
astar(problem, fn_heuristic) → list
```

- Tham số:
  - problem kiểu **SingleFoodSearchProblem**
  - fn_heuristic → một hàm heuristic ở **YC2-1**
- Trả về: list các action để pacman đi tới điểm mồi.

YC2-4: Chỉnh sửa hàm astar ở **YC2-3** lên mức tổng quát để có thể hoạt động tốt với cả **SingleFoodSearchProblem** và **MultiFoodSearchProblem**.

YC2-5: Cài đặt, trong tệp **searchAgents.py**, hàm

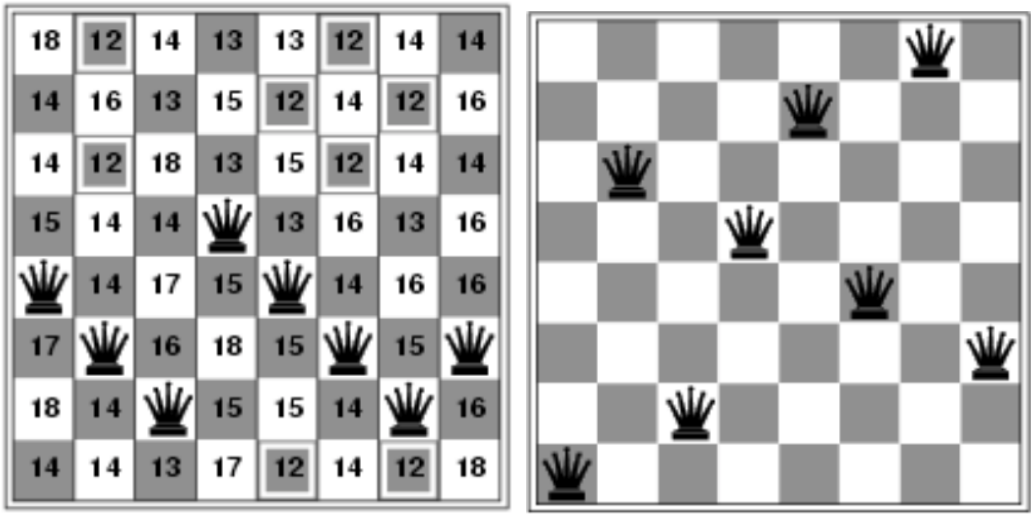
`gbfs(problem, fn_heuristic) → list`

- Tham số:
  - problem kiểu **SingleFoodSearchProblem/MultiFoodSearchProblem**
  - fn_heuristic → một hàm heuristic ở **YC2-1** hoặc **YC2-2**
- Trả về: list các action để pacman đi tới trạng thái đích.

Tiêu chí	Điểm
YC2-1	**1.0 điểm**
YC2-2	**0.5 điểm**
YC2-3	**0.5 điểm**
YC2-4	**0.5 điểm**
YC2-5	**0.5 điểm**
Tổng	**3.0 điểm**

c) Câu 3: Local Search

Cho một bàn cờ vua 8x8, trên đó đặt sẵn 8 quân hậu ở 8 vị trí bất kỳ với ràng buộc mỗi cột có ít nhất một quân hậu. Ví dụ như hình bên dưới.



Bàn cờ với các giá trị heuristic (trái) và successor state (phải)

Gọi h(state) là hàm heuristic nhận vào một trạng thái bàn cờ và trả ra một số nguyên là số cặp hậu có thể tấn công lẫn nhau.

Ở hình trên, bên trái, những con số trên mỗi ô là giá trị hàm h() khi thử đặt quân hậu trên cột tương ứng vào ô đó. Ví dụ để tính giá trị ô (0, 0) ta cố định 7 quận hậu trên cột 1 – 7, sau đó đặt quân hậu ở cột 0 vào ô (0, 0) rồi gọi hàm h() để tính.

Dữ liệu về trạng thái ban đầu của bàn cờ được đọc lên từ tệp có cấu trúc như sau:

- 8 dòng dữ liệu
- Mỗi dòng chứa 8 ký tự, cách nhau bởi khoảng trắng.
- **0** → ô trống
- **Q** → ô chứa hậu

```
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0
0 0 0 Q 0 0 0 0
Q 0 0 0 Q 0 0 0
0 Q 0 0 0 Q 0 Q
0 0 Q 0 0 0 Q 0
0 0 0 0 0 0 0 0
```

Cấu trúc tệp dữ liệu bài 8-hậu

YC3-1: Cài đặt, trong tệp **problems.py**, lớp **EightQueenProblem** với các thông tin

- Phương thức đọc bàn cờ từ tệp
- Phương thức in bàn cờ ra màn hình
- Hàm h(state) theo mô tả ở trên

YC3-2: Cài đặt phương thức

```
hill_climbing_search(self)
```

trong lớp **EightQueenProblem** như sau

- Tham số: không (ngoại trừ self)
- Trả về: bàn cờ ở trạng thái "tốt nhất" tìm được
- Mỗi bước thuật toán, di chuyển các quân hậu trên từng cột về ô với giá trị heuristic nhỏ nhất của cột tương ứng.

Tiêu chí	Điểm
YC3-1	**0.5 điểm**
YC3-2	**0.5 điểm**
Tổng	**1.0 điểm**

d) Câu 4 (1.0 điểm): Thuyết trình

- Sinh viên viết báo cáo kết quả đề tài theo hình thức thuyết trình. **KHÔNG CÓ MẪU THUYẾT TRÌNH, NHÓM SINH VIÊN TỰ TỔ CHỨC NỘI DUNG**.
- Các thông tin tối thiểu cần có.
    - Danh sách sinh viên: MSSV, Họ tên, Email, Phân công công việc, Mức độ hoàn thành.
    - Tóm tắt cách xử lý từng yêu cầu, nên diễn đạt bằng mã giả/sơ đồ.
    - HẠN CHẾ TỐI ĐA NHÚNG MÃ NGUỒN THÔ VÀO BÀI THUYẾT TRÌNH.
    - Các nội dụng tìm hiểu cần trình bày cô đọng, có ví dụ trực quan.
    - Thuận lợi và khó khăn trong đề tài.
    - Bảng tự đánh giá mức độ hoàn thành các yêu cầu.
    - Tài liệu trích dẫn ghi theo định dạng IEEE.
- Yêu cầu về định dạng: tỷ lệ slide 4x3, hạn chế dùng nền tối/màu sắc vì máy chiếu mờ, đảm bảo khi in bài thuyết trình dạng trắng đen thì các nội dung vẫn rõ ràng.
- Thời lượng tối đa cho phần thuyết trình là **10 phút**.

III. Hướng dẫn nộp bài

- Tạo thư mục với tên theo cú pháp

  <MSSV1>_<MSSV2>_<MSSV3>_<MSSV4>_<MSSV5>

  trong đó gồm:
    - source/ thư mục mã nguồn chứa các tệp .py
    - presentation.pdf bài thuyết trình.
- Nén thư mục thành tệp zip và nộp theo deadline.

IV. Quy định

- **Nhóm sinh viên nộp trễ hạn bị 0.0 điểm toàn nhóm.**
- **Sai sót mã số sinh viên nào trong tên tệp nộp bài thì sinh viên tương ứng bị 0.0 điểm.**
- **Thiếu sót các tài liệu được yêu cầu trong tệp nộp bài sẽ bị trừ tối thiểu 50% điểm phần thuyết trình.**
- **Mọi hành vi sao chép code trên mạng, chép bài bạn hoặc cho bạn chép bài nếu bị phát hiện đều sẽ bị điểm 0.0.**
- **Nếu bài làm của sinh viên có dấu hiệu sao chép trên mạng hoặc sao chép nhau, sinh viên sẽ được gọi lên phỏng vấn code riêng để chứng minh bài làm là của mình.**

-- HẾT --