Ton Duc Thang University
Faculty of Information Technology

MIDTERM PROJECT

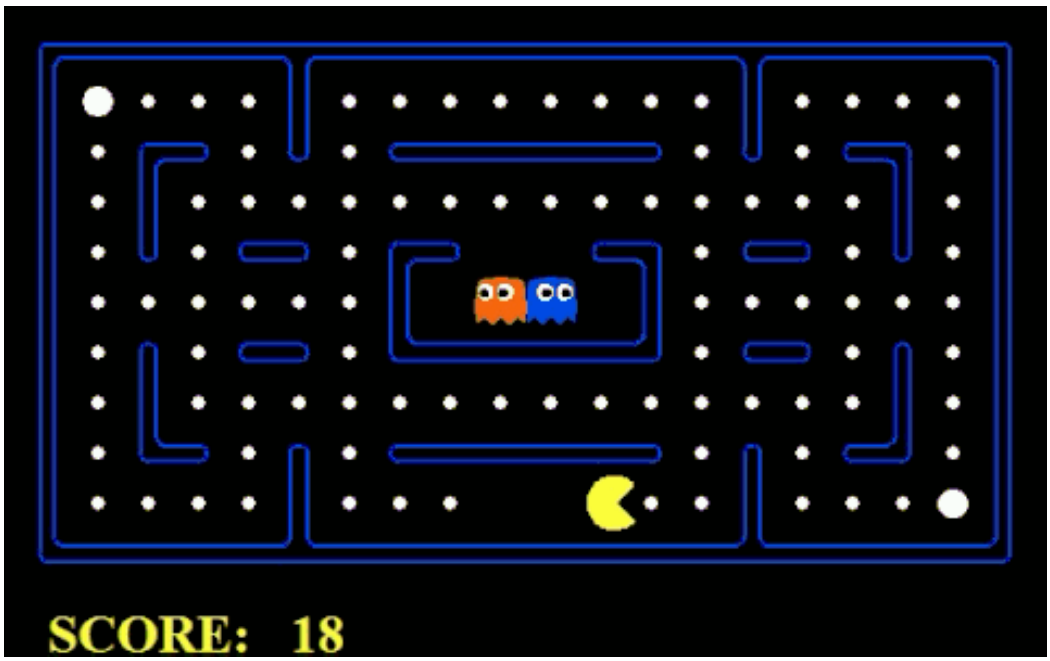
Course: Introduction to Artificial Intelligence

Duration: 03 weeks

I. Formation

- The midterm project is conducted in groups of 04 – 05 students.
- Student groups conduct required tasks and submit the project following instructions.

II. Requirements



Pacman game

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

a) Task 1: Uninformed Search

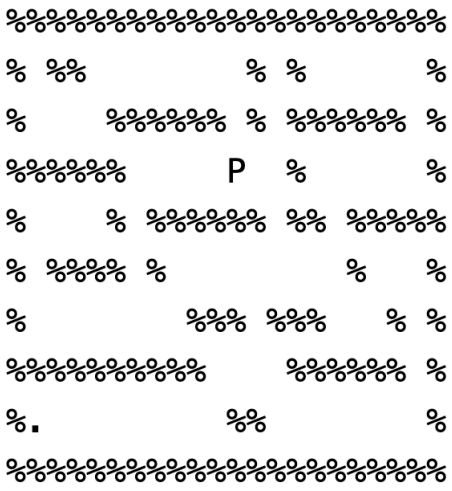
Students implements uninformed search strategies to help pacman to travel from the initial location (start) to the food location (end).

Maze layout is provided in a text file as below.

- **%** → obstacles, walls (not able to pass)
- **P** → initial location of pacman
- **.** → food location

- Bank cells are passable.

For example,



Maze layout with one food location

YC1-1: Create file **problems.py**, then implement **SingleFoodSearchProblem** class to formulate the problem in kind of Single-state problem. Details include

- State, Node, Initial state
- Successor function
- Goal-test function
- Path-cost function
- A method to read the maze layout from a text file
- A method to print the maze down on the screen

YC1-2: Implement, in **fringes.py**, necessary data structures utilized in search strategies, including

- Stack
- Queue
- PriorityQueue

YC1-3: Create **searchAgents.py** and the implement functions

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

These functions take in an object of **SingleFoodSearchProblem** and return a list of actions for pacman to travel from the initial location to food, including

- **N** → go up
- **S** → go down
- **W** → go to the left
- **E** → go to the right
- **Stop** → stop

An example returned value of bfs(): `['N', 'N', 'S', 'W', 'E', 'E', 'Stop']`.

YC1-4: Add to **SingleFoodSearchProblem** class the method

`animate(self, actions) → None`

in which actions is a sequence of moves resulted by search functions such as bfs, dfs, ucs. The function performs the procdure:

- Step 1: clear the screen
- Step 2: print the maze, current location of pacman, food locations. Then, wait for the user to press Enter
- Step 3: go to Step 1.

Each iteration step of the *animate* function extracts the next action and moves pacman to the corresponding location.

YC1-5: Add to the file **problems.py** the **MultiFoodSearchProblem** class to formulate a problem in type of Sing-state problem, in which pacman needs to **collect all food points** in the maze.

- State, Node, Initial state
- Successor function
- Goal-test function
- Path-cost function
- A method to read the maze from a text file
- A method to print the maze down on the screen
- The method `animate(self, actions)`

YC1-6: Modify function in **YC1-3** to be general enough to work correctly for problems of **SingleFoodSearchProblem** and **MultiFoodSearchProblem**.

Criteria	Score
YC1-1	**0.5 point(s)**
YC1-2	**1.5 point(s)**
YC1-3	**1.5 point(s)**
YC1-4	**0.5 point(s)**
YC1-5	**0.5 point(s)**
YC1-6	**0.5 point(s)**
Total	**5.0 point(s)**

b) Task 2: Best-Firest Search

YC2-1: Implement, in **searchAgents.py** at least 02 heuristic functions to estimate the cost from the current state to the goal state in **SingleFoodSearchProblem**.

- Parameters: state (current state)
- Returns: heuristic value (integer/floating-point number)

Discuss, in the presentation, ***admissibility*** and ***consistency*** of the two functions.

YC2-2: Implement, in **searchAgents.py**, at least 01 heuristic function to estimate the cost from the current state to the goal state in **MultiFoodSearchProblem**.

- Parameters: state (current state)
- Returns: heuristic value (integer/floating-point number)

YC2-3: Implement, in **searchAgents.py**, the function

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

- Parameters:
  - *problem (***SingleFoodSearchProblem)**
  - *fn_heuristic* → one heuristic function in **YC2-1**
- Returns: a list of actions for pacman to travel to the food location.

YC2-4: Modify the *astar* function in **YC2-3** to be general enough to work correctly for problems of **SingleFoodSearchProblem** and **MultiFoodSearchProblem**.

YC2-5: Implement, in **searchAgents.py**, the function

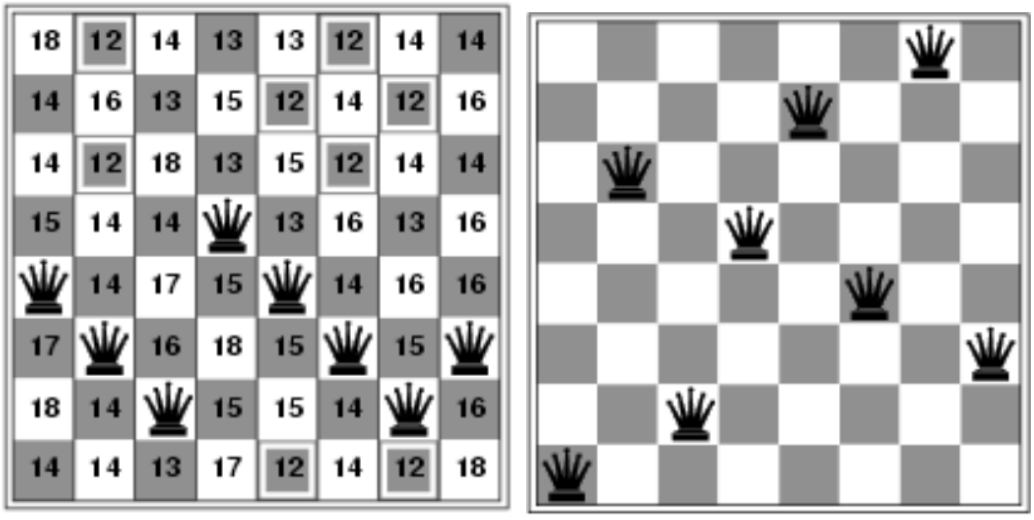
```
gbfs(problem, fn_heuristic) → list
```

- Parameters:
  - *problem* (**SingleFoodSearchProblem**/**MultiFoodSearchProblem**)
  - *fn_heuristic* → one heuristic function in **YC2-1** or **YC2-2**
- Returns: a list of actions for pacman to reach the goal state.

Criteria	Score
YC2-1	**1.0 point(s)**
YC2-2	**0.5 point(s)**
YC2-3	**0.5 point(s)**
YC2-4	**0.5 point(s)**
YC2-5	**0.5 point(s)**
Total	**3.0 point(s)**

c) Task 3: Local Search

Give an 8x8 chess board in which there are 8 queens in arbitrary cells but there is exactly one queen in a column. For example,



The chess board with heuristic values (left) and a successor state (right)

Suppose *h(state)* is a heuristic function taking in a chess board state and an integer which is the number of queen couples that are able to attack mutually.

In the figure (left), each number is a value evaluated by h() when placing the queen of the column to that cell. For example, to calculate the value of cell (0, 0), fix 7 queens in columns 1 – 7, then place the queen of column 0 in cell (0, 0) and call h() to compute.

The initial state of the chess board is given in a text file as below.

- 8 lines
- 8 characters, separated by spaces, for each line
- **0** → blank cell
- **Q** → cell with a queen

```
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
```

Example input text file

YC3-1: Implement, in **problems.py**, class **EightQueenProblem** as below

- A method to read data from a text file
- A method to print the chess board on the screen
- The function *h(state)* as above

YC3-2: Implement the method

```
hill_climbing_search(self)
```

in the **EightQueenProblem** class as below

- Parameters: none (except *self*)
- Returns: a chess board at the "best" state (local maximum)

- For each step in the algorithm, move the queen, in each column, to the cell with the minimum value in the corresponding column.

Criteria	Score
YC3-1	**0.5 point(s)**
YC3-2	**0.5 point(s)**
Total	**1.0 point(s)**

d) Task 7 (1.0 point): Presentation

- Student groups compose a presentation to report your work.
- **THERE IS NO PRESENTATION TEMPLATES. STUDENTS ARANGE CONTENTS IN A LOGICAL LAYOUT BY YOURSELVES.**
- The presentation must include below contents
  - Student list: Student ID, Full name, Email, Assigned tasks, Complete percentage.
  - Briefly present approaches to solve tasks, should make use of pseudo code/diagrams.
  - AVOID EMBEDDING RAW SOURCE CODE IN THE PRESENTATION.
  - Study topics are introduced briefly with practical examples.
  - Advantages versus disadvantages
  - A table of complete percentages for each task.
  - References are presented in IEEE format.
- **Format requirements:** slide ratio of 4x3, avoid using dark background/colorful shapes because of projector quality, students ensure contents are clear enough when printing the presentation in grayscale.
- Presentation duration is **10 minutes**.

III. Submission Instructions

- Create a folder whose name is as

<Student ID 1>_< Student ID 2>_< Student ID 3>_< Student ID 4>

- Content:

- **source/** → source code folder (containing .py files)
- **presentation.pdf** → presentation.

- Compress the folder to a zip file and submit by the deadline.

IV. Policy

- **Student groups submitting late get 0.0 points for each member.**
- **Wrong student IDs in the submission filename cause 0.0 points for the corresponding students.**
- **Missing required materials in the submission loses at least 50% points of the presentation.**
- **Copying source code on the internet/other students, sharing your work with other groups, etc. cause 0.0 points for all related groups.**
- **If there exist any signs of illegal copying or sharing of the assignment, then extra interviews are conducted to verify student groups' work.**

-- THE END --