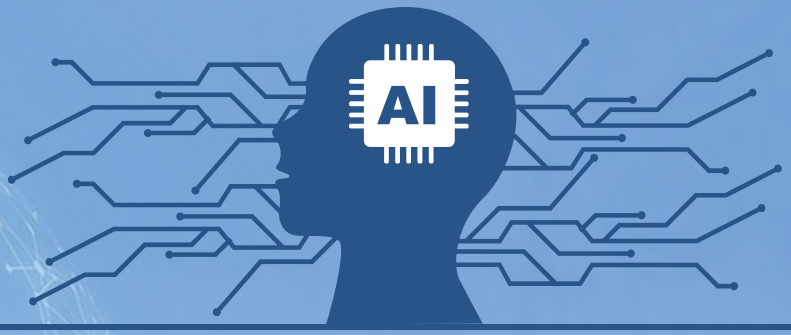
Artificial Intelligence



Homework #1
Pacman - Search

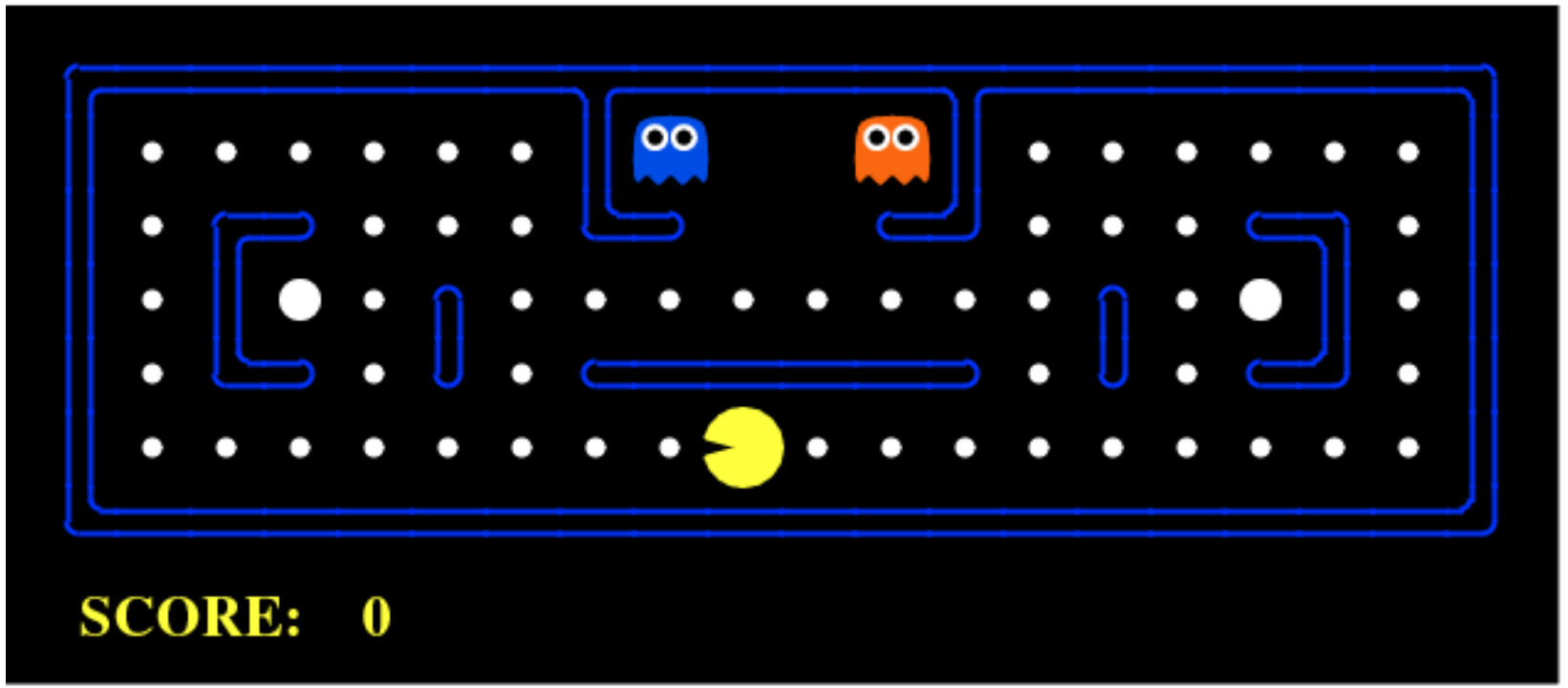
Wen-Huang Cheng (鄭文皇)

National Taiwan University

wenhuang@csie.ntu.edu.tw

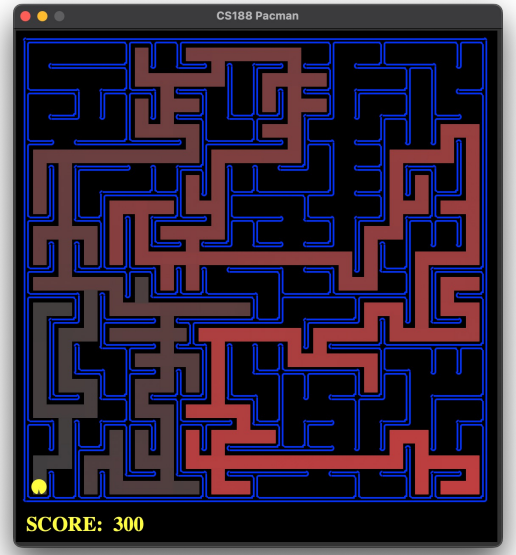
Objective

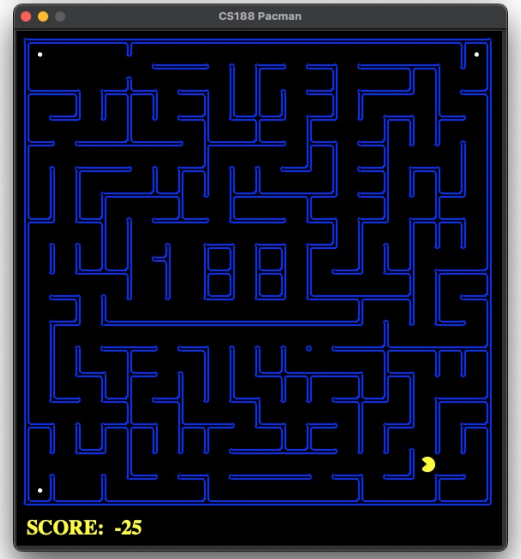
- **The goal of this assignment is to apply the search algorithm covered in class to successfully implement two problems in Pacman.**



Objective

1. **Position Search Problem:**
   - Q1. Depth First Search
   - Q2. Breadth First Search
   - Q3. Uniform Cost Search
   - Q4. A* Search (null heuristic)
2. **Corners Problem:**
   - Q5. Breadth Fist Search (Finding all the Corners)
   - Q6. A* Search (Corners Problem: Heuristic)





Important note: All of your search functions need to return a list of actions that will lead the agent from the start to the goal. These actions all have to be legal moves (valid directions, no moving through walls).

List of actions: [‘South’, ‘South’, ‘West’, ‘South’, ‘West’, ‘West’]

Objective

Files you'll edit:	
`search.py`	Where all of your search algorithms will reside.
`searchAgents.py`	Where all of your search-based agents will reside.
Files you might want to look at:	
`pacman.py`	The main file that runs Pacman games. This file describes a Pacman GameState type, which you use in this project.
`game.py`	The logic behind how the Pacman world works. This file describes several supporting types like AgentState, Agent, Direction, and Grid.
`util.py`	Useful data structures for implementing search algorithms.
Supporting files you can ignore:	
`graphicsDisplay.py`	Graphics for Pacman
`graphicsUtils.py`	Support for Pacman graphics
`textDisplay.py`	ASCII graphics for Pacman
`ghostAgents.py`	Agents to control ghosts
`keyboardAgents.py`	Keyboard interfaces to control Pacman
`layout.py`	Code for reading layout files and storing their contents
`autograder.py`	Project autograder
`testParser.py`	Parses autograder test and solution files
`testClasses.py`	General autograding test classes
`test_cases/`	Directory containing the test cases for each question
`searchTestClasses.py`	Project 1 specific autograding test classes

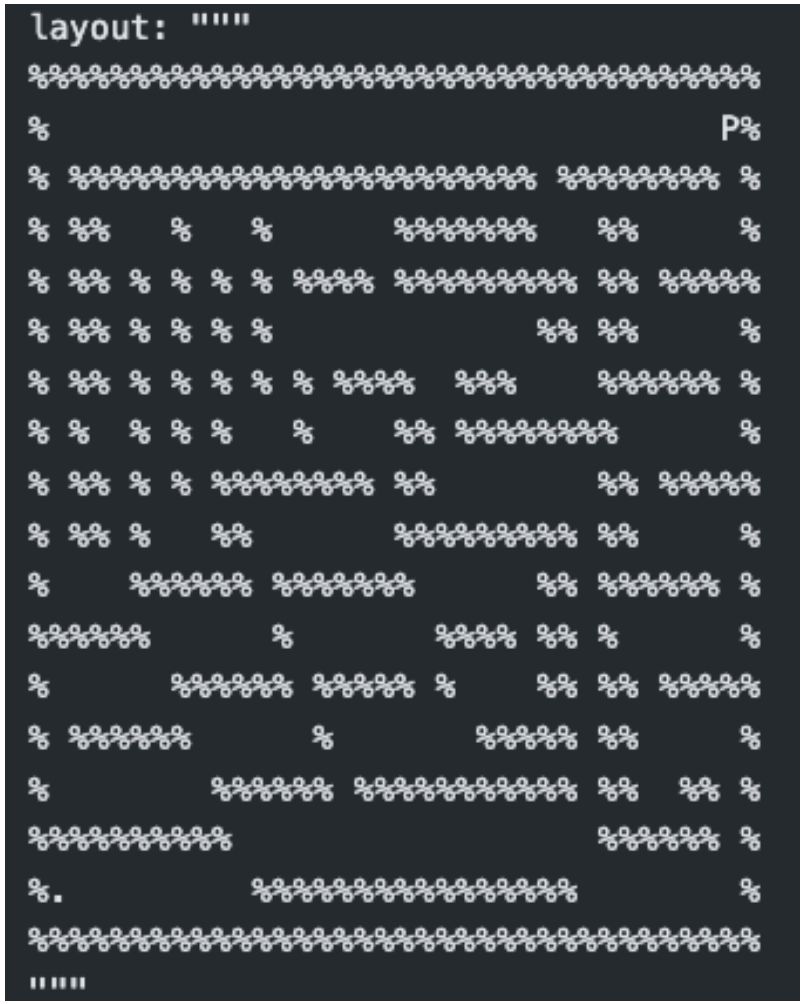
The code for this project consists of several Python files, **some of which you will need to read and understand in order to complete the assignment**, and some of which you can ignore.

In **searchAgents.py**, you'll find a fully implemented **SearchAgent**, which plans out a path through Pacman's world and then executes that path step-by-step. The search algorithms for formulating a plan are not implemented – that's your job.

Important note: Make sure to use the Stack, Queue and PriorityQueue data structures provided to you in util.py! These data structure implementations have particular properties which are required for compatibility with the autograder.

Maze Explain

- **Start Position (P)**:
  (34, 16)
- **Start's Successors**:
  [((34, 15), 'South', 1), ((33, 16), 'West', 1)]
- **End Position (.)**:
  (1, 1)
- **Wall (%)**



Pacman

Q1. Depth First Search

- Implement the depth-first search (DFS) algorithm in the depthFirstSearch function in search.py. To make your algorithm complete, write the graph search version of DFS, which avoids expanding any already visited states.
- *Grading*: python autograder.py –q q1

Q2. Breadth First Search

- Implement the breadth-first search (BFS) algorithm in the breadthFirstSearch function in search.py. Again, write a graph search algorithm that avoids expanding any already visited states.
- *Grading*: python autograder.py –q q2

Detailed information is listed in README.md inside AI2024-hw1.zip.

Pacman (Cont.)

Q3. Uniform Cost Search

- Implement the uniform-cost graph search algorithm in the uniformCostSearch function in search.py. We encourage you to look through util.py for some data structures that may be useful in your implementation.
- *Grading*: python autograder.py –q q3

Q4. A* Search (null heuristic)

- Implement A* graph search in the empty function aStarSearch in search.py. A* takes a heuristic function as an argument. Heuristics take two arguments: a state in the search problem (the main argument), and the problem itself (for reference information). The nullHeuristic heuristic function in search.py is a trivial example.
- *Grading*: python autograder.py –q q4

Detailed information is listed in README.md inside AI2024-hw1.zip.

Pacman (Cont.)

Q5. Breadth First Search (Finding all the Corners)

- The real power of A* will only be apparent with a more challenging search problem. Now, it's time to formulate a new problem and design a heuristic for it.
- Implement the CornersProblem search problem in searchAgents.py. You will need to choose a state representation that encodes all the information necessary to detect whether all four corners have been reached.
- Hint 1: The only parts of the game state you need to reference in your implementation are the starting Pacman position and the location of the four corners.
- Hint 2: When coding up getSuccessors, make sure to add children to your successors list with a cost of 1.
- *Grading*: python autograder.py –q q5

Detailed information is listed in README.md inside AI2024-hw1.zip.

Pacman (Cont.)

Q6. A* Search (Corners Problem: Heuristic)

- Note: Make sure to complete Question 4 before working on Question 6, because Question 6 builds upon your answer for Question 4.
- Implement a non-trivial, consistent heuristic for the CornersProblem in cornersHeuristic.
- Grading: Your heuristic must be a non-trivial non-negative consistent heuristic to receive any points. Make sure that your heuristic returns 0 at every goal state and never returns a negative value. Depending on how few nodes your heuristic expands, you'll be graded:
- python autograder.py –q q6

Number of nodes expanded	More than 2000	At most 2000	At most 1600	At most 1200
Grade	0/9	3/9	6/9	9/9

Detailed information is listed in README.md inside AI2024-hw1.zip.

Grading

- **Pacman (90%)**
  - Q1. Depth First Search (valid: 5%, test: 5%)
  - Q2. Breadth First Search (valid: 5%, test: 5%)
  - Q3. Uniform Cost Search (valid: 10%, test: 5%)
  - Q4. A* Search (null Heuristic) (valid: 15%, test: 10%)
  - Q5. Breadth Fist Search (Finding all the Corners) (valid: 5%, test: 5%)
  - Q6. A* Search (Corners Problem: Heuristic) (valid: 9%, test: 11%)
- **Report (10%)**
  - PDF Format only.
  - Score of each problem is listed in p10.

Evaluation

- **Pacman**
  - Your code will be autograded for technical correctness. Please do not change the names of any provided functions or classes within the code, or you will wreak havoc on the autograder. However, the correctness of your implementation – not the autograder's judgements – will be the final judge of your score.
  - Please run the below command to see if your implementation passes all the autograder test cases (You'll need to pass all the autograder on single question to get full credit):
    - python autograder.py (grade all questions at once)
    - python autograder.py –q q1 (grade one particular question)

Report

- **Show your autograder results and describe each algorithm:**
  - Q1. Depth First Search (1%)
  - Q2. Breadth First Search (1%)
  - Q3. Uniform Cost Search (1%)
  - Q4. A* Search (null Heuristic) (1%)
  - Q5. Breadth First Search (Finding all the Corners) (1%)
  - Q6. A* Search (Corners Problem: Heuristic) (1%)
- **Describe the difference between Uniform Cost Search and A* Contours (2%)**
- **Describe the idea of Admissibility Heuristic (2%)**

Submission

- Deadline: 2024/03/27 (Wed.) 23:59
- Zip all files as **hw1_<student_id>.zip**
- Submit to NTU COOL
- Your submission should include the following files:
  - hw1_<student_id>.pdf
  - All codes


Any
Question
ai.ta.2024.spring@gmail.com