Assignment2

Contents

Learning Outcomes and Objectives 2
Learning Outcomes 2
Assignment Learning Objective 2
Assignment Graduation Attributes Indicators 3
Instructions: 4
Assignment Structure and Milestones 5
Milestone Breakdown 5
Submission Requirements 5
Part One: Milestone1 Q1 (DFS) and Q2 (BFS) 6
Part Two: Milestone2 Q3 (cost function) and Q4 (A*) 9
Part Three: Reflection 10
Deliverables and Group Work 11

Assignment2

Learning Outcomes and Objectives

This assignment helps students achieve key course learning outcomes by engaging them in designing, implementing, testing, and evaluating core Artificial Intelligence search techniques in a realistic AI problem environment. Through the implementation of uninformed search algorithms (Depth-First Search and Breadth-First Search) and informed/cost-based search algorithms (Uniform Cost Search and A*), students directly **apply various search algorithms to solve AI toy problems** (CLO1). By running and comparing these algorithms across different maze configurations and cost functions, students develop the ability to **analyze and compare the applicability of AI methods based on problem characteristics**, such as path optimality, cost sensitivity, and heuristic guidance (CLO4). Additionally, the requirement to write pseudocode, implement algorithms in Python, and validate solutions with an **autograder** ensures that students gain hands-on experience implementing AI methods in **an AI programming language** (CLO5). Overall, this assignment strengthens students' algorithmic reasoning, problem-solving skills, and practical understanding of how foundational AI search methods behave in different contexts, preparing them for more advanced AI techniques later in the course.

Learning Outcomes

- **CLO1**: *Apply various search algorithms to find solutions to AI toy problems*
- **CLO4**: *Compare the applicability of AI methods based on the type of problem being solved*
- **CLO5:** *Implement AI methods using one of AI languages to solve Artificial Intelligence problems*.

Assignment Learning Objective

The learning objectives that can be achieved by solving the assignment tasks are as follows:

1. Formulate AI search problems using states, actions, costs, and goal conditions.
2. Design search algorithms using pseudo-code before implementation.
3. Implement uninformed search algorithms (DFS and BFS).
4. Implement cost-based and informed search algorithms (Uniform Cost Search and A*).
5. Apply heuristic functions to guide search efficiently.
6. Compare search algorithms based on correctness, optimality, and performance.
7. Test and validate AI algorithms using automated tools.
8. Use modern programming tools and professional development practices to solve AI problems.

Assignment2

Assignment Graduation Attributes Indicators

This assignment supports the achievement of several key **graduation attribute indicators** by engaging students in the structured design, implementation, testing, and analysis of Artificial Intelligence search algorithms. By formulating search problems in terms of states, actions, costs, and heuristics, and by implementing algorithms such as DFS, BFS, Uniform Cost Search, and A*, students **comprehend and apply first principles in specialized engineering sciences** (KB.4) and **develop algorithmic models from first principles** to solve complex problems (PA.2). The requirement to design pseudo-code, reason about correctness, and compare algorithm behavior across different maze configurations encourages students to **analyze complex engineering problems** (PA.3) and, at a foundational level, **identify and formulate problem variations** through cost functions and heuristics (PA.1). Through systematic testing, experimentation, and validation using multiple test cases and an **autograder**, students **conduct planned laboratory-style activities** (IN.1) and **interpret results to reach valid conclusions** about optimality, efficiency, and suitability of different search strategies (IN.2). Finally, the use of Python, command-line tools, and an established AI framework (Pacman project) ensures that students **apply appropriate techniques and modern engineering tools** in solving engineering problems (ET.2), reinforcing professional and industry-relevant problem-solving practices.

- KB.4 - Comprehend and apply first principles and concepts in specialized engineering sciences
- PA. 1 - Identify and formulate complex engineering problems
- PA. 2 - Develop models from first principles to solve complex engineering problems
- PA. 3 - Analyze complex engineering problems
- IN. 1 - Conduct planned activities (literature review, experiments, measurements, laboratories, etc.)
- IN. 2 - Interpret results and reaches valid conclusions regarding complex problems
- ET. 2 - Use appropriate techniques, resources and modern engineering tools in engineering activities

Assignment2

Instructions:

- The assignment can be completed **in groups (maximum of three members per group, recommended)**.
- All group members should work together, and they will receive the same mark.
    - Any partial grade will be given case-by-case.
- This workshop is worth 5% of the total course grade, and it will be graded out of 100 marks and evaluated through your written submission, as well as the lab demo, as follows:
    - 100 marks (5% of the total course grade)
        - 60%: Blackboard submission
        - 40%: Lab demo during the lab session
- Please submit the submission file(s) through Blackboard. **Only one person from the group needs to submit; only the last submission will be graded**.
- During the lab demo, group members are ***randomly*** selected to explain the submitted solution.
- **Group members who do not present during the lab demo will lose the demo mark.**
- You must submit all your answers and screenshots inside pdf file.
- You must submit all of your Python code files inside an archive file (zip).

Assignment2

Assignment Structure and Milestones

Assignment2 is worth **5%** of your final course grade. This assignment is divided into two milestones to help you manage your workload effectively and complete the assignment.

Milestone Breakdown

- **Milestone 1:** 2.5% of the final grade
- **Milestone 2:** 2.5% of the final grade

Each milestone is graded **independently** and must be completed, demonstrated, and submitted separately. Also, it will be graded out of 100 and will follow the 60% and 40% instructions above. You **must complete Milestone 1 before moving on to Milestone 2**.

Submission Requirements

There are two separate submission links (boxes) on Blackboard:

- One link/box for Milestone 1
- A different link for Milestone 2

You must submit each milestone to its corresponding box. **Submitting work to the wrong box may result in your work not being graded.**
For **each milestone**, you must follow this process:

1. **Complete the milestone**
2. **Demonstrate the milestone during your lab session**
3. **Submit the milestone on Blackboard to the correct link/box**

You must **fully complete and demonstrate Milestone 1 before submitting it**, and only then proceed to Milestone 2.

Note: Think of this assignment as one assignment divided into two smaller assignments. The milestone structure is intentional and is meant to:

- Help you start early
- Divide the work into manageable parts
- Improve your understanding and overall performance
- Do not leave both milestones until the last minute.

Assignment2

Part One: Milestone1 Q1 (DFS) and Q2 (BFS)

In this milestone, you will implement uninformed search algorithms. You will be using code from

- Project 1: https://inst.eecs.berkeley.edu/~cs188/sp25/projects/proj1/

Step1. At this stage, to stay organized and avoid confusion when working on assignments, it is important to create a dedicated workspace on your computer. Such as

```
SEA400
└── Assignments
    ├── Assignment1
    ├── Assignment2
    └── Assignment3
        …
    └── Assignment6
```

Step2. Download **search.zip** from this link
   a. A zip archive located here: tutorial.zip
   b. Move the **search.zip** file to the directory you designated for assignment2 (see above)
   c. Unzip/extract the **search.zip** file inside the assignment directory.

Step3. At this point, you have two options for running the commands provided in the tutorial:
   a. **Local Environment**: Use your own computer's terminal (Command Prompt, PowerShell, or Terminal on macOS/Linux).
   b. **Google Colab:** If you prefer a cloud-based solution, you can use Google Colab.

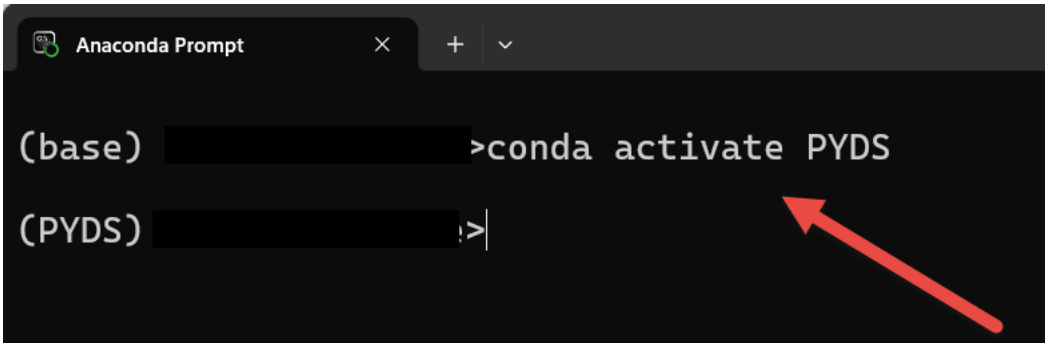

Figure 1

Step4. Open your terminal window and navigate to the **search** directory

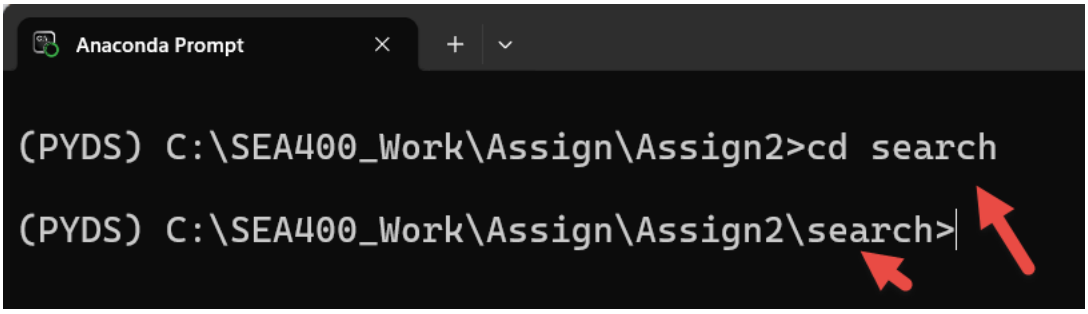

Figure 2

Step5. Keep this terminal open and run code here. For example, you can check the programming environment by running the following command.

```
python pacman.py
```

Step6. Follow the instructions to solve and test questions 1 to 2 from the link (Project 1).
For each question, it is recommended to develop and write your algorithm (pseudo-code) that you plan to implement, for example:

```
function BREADTH-FIRST-SEARCH(problem) returns a solution node or failure
  node ← NODE(problem.INITIAL)
  if problem.IS-GOAL(node.STATE) then return node
  frontier ← a FIFO queue, with node as an element
  reached ← {problem.INITIAL}
  while not IS-EMPTY(frontier) do
    node ← POP(frontier)
    for each child in EXPAND(problem, node) do
      s ← child.STATE
      if problem.IS-GOAL(s) then return child
      if s is not in reached then
        add s to reached
        add child to frontier
  return failure
```

Assignment2

Step7. After you write your pseudo-code on a piece of paper, it is time to implement your algorithm and add comments to each step (or add sufficient comments to your code).

Step8. Implement the **depth-first search (DFS)** algorithm in the **depthFirstSearch** function in **search.py**.

Step9. Your code should quickly find a solution for:

```
python pacman.py -l tinyMaze -p SearchAgent
python pacman.py -l mediumMaze -p SearchAgent
python pacman.py -l bigMaze -z .5 -p SearchAgent
```

Step10. Please run the command below to see if your implementation passes all the **autograder** test cases.

```
python autograder.py -q q1
```

Step11. Implement the **breadth-first search (BFS)** algorithm in the **breadthFirstSearch** function in **search.py.**

Step12. Test your code the same way you did for depth-first search.

```
python pacman.py -l mediumMaze -p SearchAgent -a fn=bfs
python pacman.py -l bigMaze -p SearchAgent -a fn=bfs -z .5
```

Step13. Please run the command below to see if your implementation passes all the **autograder** test cases.

```
python autograder.py -q q2
```

Step14. If you do want to see any warning messages the use the following command

Windows:

```
python -W "ignore::SyntaxWarning" autograder.py -q q1
```

Linux (Terminal):

```
python -W "ignore::SyntaxWarning" autograder.py -q q1
```

Step15. You are required to take screenshots to demonstrate the successful execution of your code for the above steps. These screenshots serve as proof of completion and proper identification.

a. Use any document editor (e.g., Word, Google Docs) to compile all your screenshots into a single document.
b. Make sure you follow the assignment instructions for the file naming convention

Step16. You need to submit your **search.py** file, which contains your implementation for Q1 and Q2.

Assignment2

Part Two: Milestone2 Q3 (cost function) and Q4 (A*)

Step17. Follow the instructions to solve and test questions 3 to 4 from the link (Project 1).

For each question, it is recommended to develop and write your algorithm (pseudo-code) that you plan to implement.

Step18. After you write your pseudo-code on a piece of paper, it is time to implement your algorithm and add comments to each step (or add sufficient comments to your code).

Step19. Implement the **uniform-cost graph** search algorithm in the **uniformCostSearch** function in **search.py**.

Step20. You should now observe successful behavior in all three of the following layouts, where the agents below are all **UCS** agents that differ only in the cost function they use (the agents and cost functions are written for you):

```
python pacman.py -l mediumMaze -p SearchAgent -a fn=ucs
python pacman.py -l mediumDottedMaze -p StayEastSearchAgent
python pacman.py -l mediumScaryMaze -p StayWestSearchAgent
```

Step21. Implement **A*** search in the empty function **aStarSearch** in **search.py**.

Step22. You can test your **A*** implementation on the original problem of finding a path through a maze to a fixed position using the Manhattan distance heuristic (implemented already as **manhattanHeuristic** in **searchAgents.py**).

```
python pacman.py -l bigMaze -z .5 -p SearchAgent -a
fn=astar,heuristic=manhattanHeuristic
```

Step23. Please run the command below to see if your implementation passes all the **autograder** test cases.

```
python autograder.py -q q4
```

Step24. If you do want to see any warning messages the use the following command
Windows:

```
python -W "ignore::SyntaxWarning" autograder.py -q q3
```

Linux (Terminal):

```
python -W "ignore::SyntaxWarning" autograder.py -q q3
```

Step25. You are required to take screenshots to demonstrate the successful execution of your code for the above steps. These screenshots serve as proof of completion and proper identification.

c. Use any document editor (e.g., Word, Google Docs) to compile all your screenshots into a single document.
d. Make sure you follow the assignment instructions for the file naming convention

Step26. You need to submit your **search.py** file, which contains your implementation for Q1 and Q2.

Part Three: Reflection

Reflecting on your approach and techniques will enhance your understanding of the concepts covered in this Assignment. This reflection will help solidify your learning and identify areas for further study. Ensure you have a firm grasp of the concepts and techniques applied in this lab. After this review, answer the following questions.

Question 1. In your own words, what is the key difference between **depth-first search (DFS)** and **breadth-first search (BFS)**?

Question 2. How did you ensure that your search algorithms avoided revisiting the same states repeatedly?

Question 3. How does **Uniform Cost Search (UCS)** differ conceptually from BFS?

Question 4. What additional information does A* use that UCS does not?

Question 5. Rank **DFS**, **BFS**, **UCS**, and **A*** in terms of:

- Optimality
- Completeness
- Time efficiency

And then explain your rank.

Assignment2

Deliverables and Group Work

Create workshop report with the following name format

group_<number>_assign_<assignment number>_report.pdf

For example, if **group26** created a report for **workshop20,** then the report name should be

group_26_assign_20_report.pdf

The workshop report should include:

(b) Complete this declaration by adding your names:

We, ---------- (mention your names), declare that the attached assignment is our own work in accordance with the Seneca Academic Policy. We have not copied any part of this assignment, manually or electronically, from any other source, including websites, unless specified as references. We have not distributed our work to other students.

(c) Specify what each member has done towards the completion of this work:

	Name	Task(s)
1		
2		
3		

(d) Read the assignment steps and questions carefully. If the assignment questions (or part of the question) are asked for output or response, then you should include the output images inside an assignment report as a PDF file and write a response to answer some of the assignment questions (or part of the question).

(e) As part of your assignment submission, you are required to take screenshots to demonstrate the successful execution of your code. These screenshots serve as proof of completion and proper identification.
   a. Use any document editor (e.g., Word, Google Docs) to compile all your screenshots into a single document.
   b. Make sure you follow the assignment instructions for the file naming convention

(f) Submit file (s)
   a. group_26_assign_20_report.pdf
   b. search.py