Assignment 2

Introduction

This assignment includes two parts, programming assignment and writing assignment. The deadline of this assignment is **16 Mar (11:59 PM), 2024**.

Part 1: Programming assignment

Download the code for this assignment here and then unzip the archive.

As in assignment 1, this assignment uses python 3. Do not use python 2. You can work on the assignment using your favourite python editor. We recommend VSCode.

Post any questions or issues with this assignment to our discussion forum. Alternatively you may also contact your TA directly.

This assignment uses auto grading for Problems 1 and 3. For the other problems no auto grader is provided. The auto grader is provided for your own convenience and we won't use it to mark your submission. Instead, all submissions will be hand graded by our TA. Make sure that your code is readable and add appropriate comments, if necessary.

Important: Ensure that your code does not run too long. More than 5 min per test case (assuming at most 10 trials for each testcase) on our grading machine (M1 Mac) would be considered too long. For Problems 5 and 6 in particular it is required to search to reasonable depth which is only possible with a state representation that enables O(1) time queries. Problems 2, 4, 5 and 6 have a verbose option which is passed as a parameter into the search function. Only perform output computations when verbose is True to save valuable compute.

Problem 1: Random Pacman play against a single random Ghost (15 points)

In this part of the assignment you are going to implement

- a parser to read a pacman layout file in the file `parse.py`, and
- a game of pacman where pacman and ghost select moves randomly in the file `p1.py`.

Both these python files have already been created for. Do not change anything that has already been implemented. Our autograder relies on the existing code.

Start by implementing the `read_layout_problem()` in the file `parse.py`.

```
def read_layout_problem(file_path):
    #Your p1 code here
    problem = ''
    return problem
```

You can test your code with the first layout as follows.

```
python parse.py 1 1
```

This will supply the `test_cases/p1/1.prob` file as an argument to the function. The file has the following pacman layout.

```
seed: 8
%%%%
% W%
%  %
%  %
% .%
%P.%
%%%%
```

The first line is a random seed which will be used to initialize python's random number generator via the `random.seed()` function. This ensures that python generates a fixed sequence of random values. More on this later. The rest of the file is the pacman layout that you will have to parse. You can expect the following characters in the file.

```
'%': Wall
'W': Ghost
'P': Pacman
'.': Food
' ': empty Square
```

You may assume the layout is always surrounded by walls and that there is at least one ghost. For problem 1 there will be a single ghost only. Later Pacman we will have to deal with more ghosts (`'X'`, `'Y'`, `'Z'`). There will always be at least one food and there will always be exactly one pacman.

As in assignment 1, you can choose any data structure and return it from your `read_layout_problem` function. Once you are done with the parsing you can move on to the second part of this problem and implement the `random_play_single_ghost()` function in the file `p1.py`.

A correct implementation will return the following string for the first test case.

```
seed: 8
0
%%%%
% W%
%  %
%  %
% .%
%P.%
%%%%
1: P moving E
```

```
%%%%
% W%
%  %
%  %
% .%
% P%
%%%%
score: 9
2: W moving W
%%%%
%W %
%  %
%  %
% .%
% P%
%%%%
score: 9
3: P moving W
%%%%
%W %
%  %
%  %
% .%
%P %
%%%%
score: 8
4: W moving E
%%%%
% W%
%  %
%  %
% .%
%P %
%%%%
score: 8
5: P moving E
%%%%
% W%
%  %
%  %
% .%
% P%
%%%%
score: 7
6: W moving S
%%%%
%  %
% W%
```

```
%  %
% .%
% P%
%%%%
score: 7
7: P moving N
%%%%
%  %
% W%
%  %
% P%
%  %
%%%%
score: 516
WIN: Pacman
```

As you can see, Pacman and the Ghost make moves alternatively. Pacman starts by making a move east. This is determined using the random.choice() function on the available moves to pacman. In this case for the start state pacman can move East (E) for the food or North(N) for the empty square. Pacman (P) moves east. Let's reproduce that decision to understand the sequence of moves generated here.

```
(base) AICourse@AICourse a2 % python
>>> import random
>>> random.seed(8, version=1)
>>> random.choice(('E', 'N'))
'E'
```

Next, the Ghost (`W`) moves West (`W`). The available actions to the ghost are `W` and `S`.

```
(base) AICourse@AICourse a2 % python
>>> import random
>>> random.seed(8, version=1)
>>> random.choice(('E', 'N'))
'E'
>>> random.choice(('S', 'W'))
'W'
```

Important: You must ensure that the parameter to the random.choice() function is sorted alphabetically. Otherwise you will not be able to reproduce the exact result and you won't be able to pass the autograder.

In the test case 1 above, Pacman won because he cleared all the food. Pacman and Ghost may move to any square except a wall. A ghost may move on top of a food but it won't eat the food. If Pacman is eaten, the game is over and Pacman loses. Here is another run where that happens. This is test case 2.

```
seed: 42
0
%%%%
%.W%
%  %
%  %
% .%
%P.%
%%%%
1: P moving E
%%%%
%.W%
%  %
%  %
% .%
% P%
%%%%
score: 9
2: W moving S
%%%%
%. %
% W%
%  %
% .%
% P%
%%%%
score: 9
3: P moving W
%%%%
%. %
% W%
%  %
% .%
%P %
%%%%
score: 8
4: W moving N
%%%%
%.W%
%  %
%  %
% .%
%P %
%%%%
score: 8
5: P moving E
%%%%
%.W%
```

```
%  %
%  %
% .%
% P%
%%%%
score: 7
6: W moving S
%%%%
%. %
% W%
%  %
% .%
% P%
%%%%
score: 7
7: P moving N
%%%%
%. %
% W%
%  %
% P%
%  %
%%%%
score: 16
8: W moving W
%%%%
%. %
%W %
%  %
% P%
%  %
%%%%
score: 16
9: P moving W
%%%%
%. %
%W %
%  %
%P %
%  %
%%%%
score: 15
10: W moving S
%%%%
%. %
%  %
%W %
%P %
```

```
%  %
%%%%
score: 15
11: P moving E
%%%%
%. %
%  %
%W %
% P%
%  %
%%%%
score: 14
12: W moving S
%%%%
%. %
%  %
%  %
%WP%
%  %
%%%%
score: 14
13: P moving S
%%%%
%. %
%  %
%  %
%W %
% P%
%%%%
score: 13
14: W moving E
%%%%
%. %
%  %
%  %
% W%
% P%
%%%%
score: 13
15: P moving N
%%%%
%. %
%  %
%  %
% W%
%  %
%%%%
score: -488
```

```
WIN: Ghost
```

Scoring is done as follows.

```
EAT_FOOD_SCORE = 10
PACMAN_EATEN_SCORE = -500
PACMAN_WIN_SCORE = 500
PACMAN_MOVING_SCORE = -1
```

You may define any number of your own functions to solve the problem.
Once you are done you can check if you pass all the test cases for Problem 1.

```
(base) AICourse@AICourse a2 % python p1.py -6
Grading Problem 1 :
----------> Test case 1 PASSED <----------
----------> Test case 2 PASSED <----------
----------> Test case 3 PASSED <----------
----------> Test case 4 PASSED <----------
----------> Test case 5 PASSED <----------
----------> Test case 6 PASSED <----------
```

You may import anything from the Python Standard Library. Do not import packages that are not included in Python such as numpy.
Make sure that you pass all provided test cases before moving on to the next question. Note that we may use novel test cases for marking. You can also design your own new test cases for testing.

Problem 2: Pacman play against a single random Ghost (10 points)

Next you will write a simple reflex agent that does not move randomly. Instead it will move to a better position based on a simple evaluation function, designed by you, that takes a state-action pair and returns the best action. Hint: You may consider evaluating the distance to the closest ghost and combine somehow with the distance to the closest food.

You may reuse some of the code that you wrote in Problem 1 and implement just the evaluation function and action selection by completing the code in the function `better_play_single_ghosts()`. This function should return two values, the gameplay as usual and the winner which should be `'Pacman'` if Pacman wins. You can see how the `better_play_single_ghosts()` is used in the main function of the script `p2.py`.

This time, we will have no autograder and ghosts should move randomly without a seed. You may inspect the test cases and note that the seed is -1, which indicates that there will be no seed. This allows us to conduct multiple trials.

```
(base) AICourse@AICourse a2 % python p2.py 1 10 0
test_case_id: 1
```

```
num_trials: 10
verbose: False
time:  0.09921669960021973
win % 90.0
```

The three parameters control the test case id, number of trials and a verbose option that will output the actual gameplay if it is set to `1` instead of `0`.

```
base) AICourse@AICourse a2 % python p2.py 1 1 1
test_case_id: 1
num_trials: 1
verbose: True
seed: -1
0
%%%%%
% . %
%.W.%
% . %
%. .%
%   %
%  .%
%   %
%P .%
%%%%%
…
123: P moving W
%%%%%
%   %
%   %
%   %
%P  %
%   %
%   %
%   %
%W  %
%%%%%
score: 518
WIN: Pacman
time:  0.008844137191772461
win % 100.0
```

With a reasonable evaluation function you should be able to win half of the games easily. No need to spend too much time on this question. You will implement expectimax soon and play better games.

Problem 3: Random Pacman play against up to 4 random Ghost (10 points)

This problem is similar to problem 1 except that you will implement a game against multiple ghosts.

Ghosts cannot move on top of each other. If a Ghost is stuck no move will be done. Pacman will start followed by `W`, `X`, `Y`, `Z`. Note that the last three ghosts are optional. So the following combinations are possible:

- 1 Ghost: `W`
- 2 Ghosts: `W, X`
- 3 Ghosts: `W, X, Y`
- 4 Ghosts: `W, X, Y, Z`

Here is a game with 2 ghosts.

```
seed: 42
0
%%%%
%.X%
%W %
%P %
%%%%
1: P moving E
%%%%
%.X%
%W %
% P%
%%%%
score: -1
2: W moving E
%%%%
%.X%
% W%
% P%
%%%%
score: -1
3: X moving W
%%%%
%X %
% W%
% P%
%%%%
score: -1
4: P moving N
%%%%
%X %
% W%
%  %
%%%%
score: -502
```

```
WIN: Ghost
```

Note that ghosts move in alphabetical order, i.e., `W` first followed by `X` etc.

Problem 4: Pacman play against up to 4 random Ghost (15 points)

This problem is similar to P2 except that we have multiple ghosts as in P3. Again, no autograding is provided and you may check the performance of your agent as follows.

```
(base) AICourse@AICourse a2 % python p4.py 1 100 0
test_case_id: 1
num_trials: 100
verbose: False
time:  0.3293917179107666
win % 80.0
```

Don't worry too much if the performance of your agent is much worse compared to P2. This is to be expected considering the problem difficulty has increased with multiple ghosts.

Problem 5: Minimax Pacman play against up to 4 minimax Ghosts (15 points)

In this problem, you will implement a minimax agent and minimax ghosts. Your minimax should search until a ply depth k (i.e., k moves by everyone) and then use an evaluation function to determine the value of the state. The depth is provided as a parameter to the function. You can test the performance of your game playing agent as follows.

```
(base) AICourse@AICourse a2 % python p5.py 1 3 1 0
test_case_id: 1
k: 3
num_trials: 1
verbose: False
time:  0.93696603775024414
win % 0.0
```

With minimax pacman and agent moves could be deterministic, depending on your evaluation function. So a single trial would be sufficient then. However, you may add randomness to this problem by selecting the best move randomly among equally good best moves.

Problem 6: Expectimax Pacman play against up to 4 random Ghosts (10 points)

Finally, you will implement an expectimax agent against random ghosts. Your expectimax should search until a depth k and then use an evaluation function to determine the value of the state. The depth is provided as a parameter to the function. You can test the performance of your game playing agent as follows.

```
(base) AICourse@AICourse a2 % python p6.py 1 2 10 0
test_case_id: 1
k: 2
num_trials: 10
verbose: False
time:  0.2952871322631836
win % 90.0
```

Part 2: Writing assignment

Answer the questions below and save your answers as "report.pdf".

1. **(11 points)** You are in charge of scheduling for computer science classes that meet on the same day. There are 6 classes that meet on these days and 4 professors who will be teaching these classes. You are constrained by the fact that each professor can only teach one class at a time.

   The classes are:

   - Class 1 - Intro to Programming: meets from 8:00-9:00am
   - Class 2 - Intro to Artificial Intelligence: meets from 8:30-9:30am
   - Class 3 - Natural Language Processing: meets from 9:00-10:00am
   - Class 4 - Computer Vision: meets from 9:00-10:00am
   - Class 5 - Machine Learning: meets from 9:30-10:30am
   - Class 6 - Computer Graphics: meets from 9:30-10:30am

   The professors are:

   - Professor A, who is available to teach Classes 3 and 6
   - Professor B, who is available to teach Classes 2, 3, 4, and 5
   - Professor C, who is available to teach Classes 3, 4, and 5
   - Professor D, who is available to teach Classes 1, 2, 3, and 6

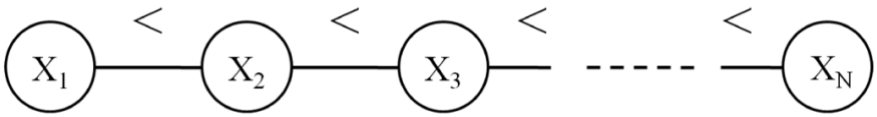
(1) Formulate this problem as a CSP problem in which there is one variable per class, stating the domains, and constraints. Constraints should be specified formally and precisely, but may be implicit rather than explicit. **(3 points)**

(2) Draw the constraint graph associated with your CSP. **(3 points)**

(3) Show the domains of the variables after running arc-consistency on this initial graph (after having already enforced any unary constraints). **(3 points)**

(4) Give all solutions to this CSP. **(2 points)**

2. **(7 points)** Consider the general less-than chain CSP below. Each of the N variables X_i has the domain $\{1 \dots M\}$. The constraints between adjacent variables X_i and X_{i+1} require that $X_i < X_{i+1}$.



For now, assume $N = M = 6$.

(1) How many solutions does the CSP have? **(1 point)**

(2) What will the domain of X_1 be after enforcing the consistency of only the arc $X_1 \rightarrow X_2$? **(1 point)**

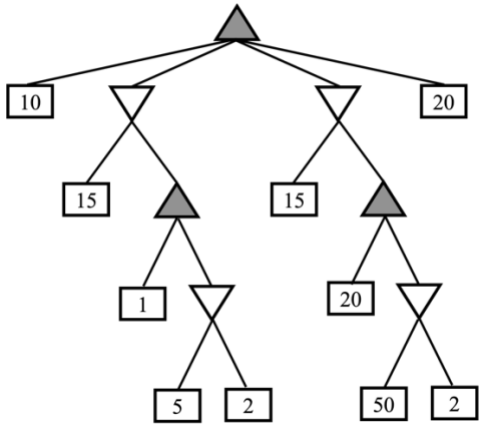
(3) What will the domain of X_1 be after enforcing the consistency of only the arcs $X_3 \rightarrow X_4$ then $X_2 \rightarrow X_3$ and then $X_1 \rightarrow X_2$? **(2 points)**

(4) What will the domain of X_1 be after fully enforcing arc consistency? **(1 point)**

Now consider the general case for arbitrary N and M.

(5) What is the minimum number of arcs (big-O is ok) which must be processed by AC-3 (the algorithm which enforces arc consistency) on this graph before arc consistency is established? **(2 points)**

3. **(7 points)** Consider the following minimax tree.



(1) What is the minimax value for the root? **(2 points)**

(2) Draw an "X" through any nodes which will not be visited by alpha-beta pruning, assuming children are visited in left-to-right order. **(3 points)**

(3) Is there another ordering for the children of the root for which more pruning would result? If so, state the order. **(2 points)**

Submission

To submit your assignment to Moodle, `*.zip` the following files ONLY:

- `p1.py`
- `p2.py`
- `p3.py`
- `p4.py`
- `p5.py`
- `p6.py`
- `parse.py`
- `report.pdf`

Do not zip any other files. Use the `*.zip` file format. Name the file UID.zip, where UID is your university number, like 3030661123.zip. Make sure that you have submitted the correct files and named all files correctly. We will deduct up to 5% for files with incorrectly file names.