COMP3211:
Fundamentals of Artificial Intelligence

Homework Assignment 3

Release Date: Nov. 16, 2024

The HKUST Academic Honor Code

Honesty and integrity are central to the academic work of HKUST. Students of the University must observe and uphold the highest standards of academic integrity and honesty in all the work they do throughout their program of study. As members of the University community, students have the responsibility to help maintain the academic reputation of HKUST in its academic endeavors.

1 Introduction

In this assignment, there are three problems corresponding to:

- Value Iteration and Policy Iteration Implementation (Problem 1).
- Minimax Game Tree (Problem 2).

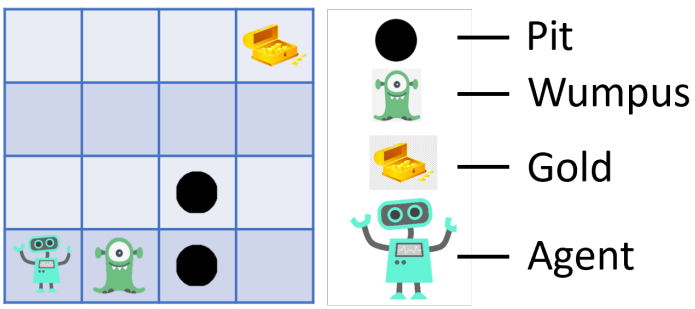
The deadline for submitting this homework is **23:59:59, Nov 30, 2024**. Please start as early as possible, and feel free to discuss with Jiabo Ma (jmabq@connect.ust.hk) for problem 1 and Xunguang Wang (xwanghm@cse.ust.hk) for problem 2 if you have any questions about your design and implementation.

This assignment should be solved individually. No collaboration, sharing of solutions, or exchange of models is allowed. Please do not directly copy other people's or previous solutions from anywhere. We will check assignments for duplicates. See below for more details about the homework.

2 Problem 1: Value Iteration and Policy Iteration in the Wumpus World (Total: 50 points)

Assignment Description

In this problem, we will delve into the application of Markov Decision Processes (MDPs) and the Value Iteration and Policy Iteration in the context of the Wumpus World. The Wumpus World is a well-known problem in the field of artificial intelligence, where the objective is to control an agent as it navigates through a grid-like environment in search of gold, all while avoiding deadly pits and the formidable Wumpus creature.



As you observed, the agent starts at the grid coordinate x = 0, y = 0, (x is the horizontal axes, y is the vertical axes) and its objectives are the following:

- Finding the gold, which provides a significant positive reward (+10).

- Avoiding the pits and the Wumpus, which are associated with negative penalties (-5 for each pit and -10 for the Wumpus).

- Minimizing the incurred movement penalty (-0.4 for each non-goal cell). Due to the noise of the control signal, the movements are stochastic: There is an 80% chance that the agent moves in the intended direction. To be more specific, there is a 10% chance that the agent moves in one of the orthogonal directions. For example, if the agent intends to move UP, there's an 80% chance it will move UP, a 10% chance it will move LEFT, and a 10% chance it will move RIGHT.

There are three user-defined parameters in the program, namely, 'gamma', 'eta' and 'max_iter'. The usages are the following:

- gamma: sets a discount factor of future rewards. It represents how much future rewards are valued compared to immediate rewards.

- eta: sets a threshold for the maximum value error between two adjacent iterations to assess algorithm convergence. If the maximum value error is less than this threshold, the iteration process is terminated.

- max_iter: sets the maximum number of iterations that the implemented algorithm will run.

Requirement

We build a Wumpus World demo named `assignment3.py` for you. The states, actions, transition probabilities, and rewards are given in the code. You can use the following helper functions to help you finish the code:

- `get_reward(state, action, next_state)` # get the reward
- `get_transition_prob(state, action, next_state)` # get the transition probability

You need to implement the following functions to let the agent find gold:

- Task 1. Implement value iteration, the function is `MDP_value_iteration(self, gamma, eta, max_iter)`. You may find similar code on the Internet, but please follow the value iteration algorithm described in the lectures. (25 points)
- Task 2. Implement policy iteration, the function is `MDP_policy_iteration(self, gamma, eta, max_iter)`. You may find similar code on the Internet, but please follow the policy iteration algorithm described in the lectures. (25 points)

Once you have successfully implemented the two functions mentioned, you could use the following command to test your code ($\gamma = 0.9, \eta = 0.1, e = 10000$):

```
python assignment3.py --gamma 0.9 --eta 0.1 --e 10000
```

Submission Guidelines

- Submit your code implementation.
- Submit a document that contains the results (directly copy the output of your code) of running the following command to help us make sure that your code's output is the same as you expected. We will use other settings to test your code.

```
python assignment3.py --gamma 0.9 --eta 0.1 --e 10000
```

3 Problem 2: Games (50 points)

3.1 Problem Description: Tic-Tac-Toe

1. Tic-Tac-toe Rules. Tic-tac-toe is a two-player game played on a 3×3 grid. The rules are as follows:

- Players take turns placing their marks (O or X) in empty squares of the grid.
- The first player uses X marks, and the second player uses O marks.
- The goal is to place three of one's marks in a horizontal, vertical, or diagonal row.
- If all nine squares are filled and no player has three marks in a row, the game is a draw.
- Once a mark is placed, it cannot be moved to another position.
- The game ends when either:
  - A player wins by achieving three marks in a row.
  - All squares are filled (resulting in a draw).

2. Tic-Tac-Toe Endgame. Now, let's consider the following Tic-tac-toe game state:

O	b	X
a	X	O
O	c	X

In this game state:

- Player X moves first (using crosses).
- Player O moves second (using circles).
- Positions a, b, and c are currently empty.
- It is player X's turn to move.

3.2 Assignment Description

(1) Game Tree Construction (30 points). Construct a game tree for the above checkerboard following these specifications:

(a) Use upward-pointing triangles (Δ) to represent MAX nodes, i.e., the next action of this node is max.

(b) Use downward-pointing triangles (∇) to represent MIN nodes, i.e., the next action of this node is min.

(c) Arrange branches in alphabetical order (a, b, c from left to right).

(d) Label each edge with the corresponding move (a, b, or c).

(e) Use rectangles to represent the leaf (terminal) nodes and mark utility values at leaf nodes as follows:

- 1 for X's victory
- 0 for a draw
- -1 for O's victory

(f) Label each node with its minimax value.

(2) Alpha-Beta Pruning Analysis (20 points). Analyze the game tree from (1) for alpha-beta pruning:

(a) Determine if any branches can be pruned using the alpha-beta pruning algorithm. If not, please clarify the reasons.

(b) If pruning exists:

- Draw the pruned game tree.
- Use inequalities to indicate value bounds at pruned nodes if needed.

It is noted that alpha-beta pruning should be applied in a depth-first search way which always generates the leftmost child node first.

3.3 Requirements

- Draw the complete game tree (30 points).
- If pruning exists, draw the pruned game tree, otherwise, give the reasons (20 points).

4 Submission

Upon completion, students are required to **combine the solutions of problems 1 and 2 into one PDF file.** Required results should be clearly shown. You should name the combined PDF file as **StudentID_report.pdf**.

If there are special requirements for running your code, please also specify them in a separate ReadMe file for each problem as **StudentID_README_problem*.md**

Finally, the combined pdf report, one python file, and the optional readme files should be **packaged together in a zip file** as **StudentID_Assignment3.zip** to be uploaded for evaluation.

In summary, a student with student ID 20000000 should submit the following in the zip file 20000000_Assignment3.zip:

- 20000000_report.pdf – The combined PDF report file
- assignment3.py – Code implementation for problem 1

  (The following files are optional to submit)
- 20000000_README_problem1.md – ReadMe File for the code in problem 1

Late submission: 25 marks will be deducted for every 24 hours after the submission deadline.