DATA0006 Assignment 1

June 20, 2022

Due Date

This assignment is due at 11:59PM on Sunday 10th July, 2022

Introduction

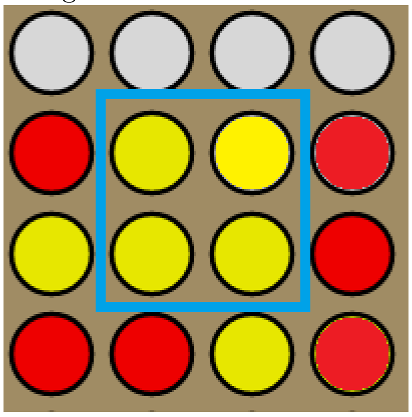
We often use computers to discover efficient solutions to problems. Artificial Intelligence (AI) provides a range of techniques for solving a wide variety of problems. In this project you will implement a number of component functions for solving an AI problem.

When developing and demonstrating AI techniques, we often use “toy worlds”. Toy Worlds are simplified problems for which a concise, exact definition is possible. This contrasts significantly to ”real world” problems, which are much more complex and messier.

The “toy world” scenario that we will use for this project is as follows.

Connect Square

The game we will consider is similar to the popular ’Connect Four’ game described here on Wikipedia, with one key difference. Rather than attempting to create a line of four discs of their own color, a player will instead win by creating a 2 x 2 square of four discs of their own color. For example, the diagram below shows a winning position for the yellow player brought about because of the four coins shown in the blue box:



As with the traditional Connect Four game, a player can place his coin in any of the columns, provided the topmost row is empty for that particular column. The coin will then slide down to the lowest unoccupied space in that column. We can represent the game space as a grid with (0,0) corresponding to the top left space. The diagram shown can then be represented as follows:

```
[[0,0,0,0],
 ['R','Y','Y','R'],
 ['Y','Y','Y','R'],
 ['R','R','Y','R']]
```

Note that the number 0 is used to represent empty spaces, while the characters 'R' and 'Y' are used to represent the red and yellow coins respectively. Note also that the grid may consist of an arbitrary number of rows and columns. A .ipynb template has been provided to help you get started.

Your Tasks

Task 1 (6 marks)

In order to begin playing this game, we require a function that can update the game state after a move has been made. Write a function `add_coin(board, coin, column)`. This function should take the following parameters:

- `board`: A list of lists representing the current state of the board
- `coin`: The character representing the current player's coin, i.e. 'R' or 'Y'
- `column`: An integer representing the column in which the player is inserting their coin, with 0 representing the left-most column in the grid. You may assume that a value greater than the maximum number of columns in the grid will never be entered.

Your function should return a list of lists representing the new state of the board after the coin has been played.

Task 2 (6 marks)

To ascertain whether the game is finished, we need a function to determine whether a particular player has won. Write a function `is_winner(board, coin)`. This function should take the following parameters:

- `board`: A list of lists representing the current state of the board
- `coin`: The character representing the current player's coin, i.e. 'R' or 'Y'

Your function should return `True` if the player using the `coin` coins has won the game in the current board position and `False` otherwise.

Task 3 (9 marks)

With these essential functions in place, we now wish to work towards building a competent AI to play our Connect Square game. The concept of a *heuristic* is central to building an AI for most strategy games. A heuristic is a function that maps a particular game state to a numeric value, indicating how desirable that state is to a particular player. For example, a game state in which the player is about to win could be assigned a very high heuristic value, while a game state in which the player is about to lose could be assigned a very low value.

There are numerous ways in which we can define a heuristic for any particular game, but we will adopt the following approach:

- We will consider each overlapping 2 x 2 square within the grid. For example, the points (0,0), (0, 1), (1, 0) and (1,1) will represent one square. (1,0), (1, 1), (2, 0) and (2,1) will represent a second square and so on.
- Consider the four points making up each square:
  - If any one of those points contains an opponent's coin, it will be impossible to win the game by filling this square with our own coins and the square will therefore be assigned a value of 0.
  - If one of the points contains our coin and the other three are empty the square will be assigned a value of 1.
  - If two of the points contain our coins and the other two points are empty then the square will be assigned a value of 10.
  - If three of the points contain our coins and the other one is empty then the square will be assigned a value of 100.
  - If all four of the points contain our coins then we have won the game and the square will be assigned a value of 1000.
- The heuristic value for this game state is the sum of the value of each square in the grid.

Note that there are some drawbacks to using this heuristic. In particular, we don't consider how close our opponent is to completing a square so an AI that uses this heuristic will not try to prevent an opponent from completing his square. You might like to consider how you could improve upon this heuristic, but for this task you should implement the heuristic as described. Write a function `heuristic(board, coin)` that returns the heuristic value of the `board` for the player using the `coin` coins.

Task 4 (9 marks)

We now wish to create an AI player capable of playing (and hopefully winning) our Connect Square game. Write a function `ai_move(board, coin)` to play the Connect Square game. Your function should generate a list of all possible moves from the current `board`. Your function should then generate the board states that arise from making each of these moves and select the move that results in the state with the highest heuristic value. If two states have the same heuristic value, then select the one that results from the using the numerically lowest column (e.g. column 1 is preferred over column 2 if the heuristic values are identical). Finally, your function should return the new board state after applying the best move.

Hint

You may find the copy library useful.

Marking

Your questions will be graded based on:

- The number of visible test cases and hidden test cases passed.
- The approach to the problem (i.e was your code too simple, overly-complicated, or excellent)
- PEP8 (make sure your output follows PEP8 standard and good variable names)
- Commenting (make sure you add comments and a docstring, you can visit Worksheet 10 for more tips)

Submission

A submission link will be made available on Canvas before the due date.