Computer and Information Sciences
University of Strathclyde

Week 5 – Game Playing with Min and Max, part 1 (worth 5% of your overall grade)

Aim

This lab is a completely new lab, leaving the previous material behind. In this practical, you are going to work with game trees, to carry out minimax, alpha-beta pruning, and other AI work. This lab is the first part, and you will implement the minimax algorithm for the game of Nimsticks

The Game of Nimsticks

Nimsticks is a turn-taking zero-sum game for two players. The game state consists of one or more piles of sticks. On their turn, a player may take 1, 2 or 3 sticks from any **one** pile. The player who takes the last stick is the loser of the game. You are recommended to work it out on paper first so you understand what the tree will look like!

Example game:

At the start of the game, we have a pile of 3 sticks and a pile of 2 sticks [3,2]. The players are Alice and Bob.

Alice takes 1 stick from the pile of 3. State is now [2, 2].
Bob takes 1 stick from a pile of 2. The state is now [2, 1].
Alice takes 2 sticks from the pile of 2. The state is now [1].
Bob takes the final stick. Alice wins!

Implementing the Minimax Algorithm– `minimax_value(state)`

You need to implement the following function:

```
def minimax_value(state):
# return the minimax value of a state
```

Your lecture notes from week 4 will provide a good guide to the algorithm.

- The state will be a tuple containing two parts: the current state of the piles, and who's turn it is (1 = MAX, 2 = MIN). For example, if there is a pile of 3 sticks and 2 sticks, and it is Max's turn to play, then the input should look like ([3, 2], 1)

  ```
  minimax_value(([3,2],1))
  ```

- It would make sense to divide your code into several functions, all called from minimax_value. It would make sense to have at the minimum separate max and min functions

- There are 2 terminal states, ([], 1) – a win for MAX, so the value is +1, and ([], 2) – a win for MIN, so the value is −1. You should check for these states, as per the algorithm

- You will find similarities with your previous labs. For example, in the algorithm, identifying successors, is similar to defining next states

- Your function should also display a game that is played according to the players' optimal strategies. You may (or may not!) need to define a global variable within your minimax

function, or return path AND value from min and max, depending on whether or not you are creating separate functions for min and max player.

- The output should be a value, either -1 or +1, and your minimax function should print an example optimal game path. Your path does not have to be exactly the same as in the examples!

Sample Examples

Some Sample outputs, the path is printed within the minimax function, and the value is returned as a string.

- A single pile of 4 sticks, max to go first, resulting in max winning (output 1).

```
print(minimax_value(([4],1)))
```

```
Example Play:  [([4], 1), ([1], 2), ([], 1)]
1
```

- Two piles of sticks, 2 and 3 in each, max to go first, resulting in max winning.

```
print(minimax_value(([2,3],1)))
```

```
Example Play: [([2, 3], 1)([2, 2], 2)([1, 2], 1)([1], 2)([], 1)]
1
```

- Three piles of 5 sticks, max to go first, resulting in min winning.

```
print(minimax_value(([5,5,5],1)))
```

```
Example Play: [([5, 5, 5], 1)([4, 5, 5], 2)([1, 5, 5], 1)([5, 5],
2)([4, 5], 1)([3, 5], 2)([5], 1)([4], 2)([1], 1)([], 2)]
-1
```

- Two piles of 1 and 2 sticks, min to go first, resulting in min winning.

```
print(minimax_value(([1,2],2)))
```

```
Example Play [([1, 2], 2)([1], 1)([], 2)]
-1
```

Submission Instructions

- This practical is worth 5% of the mark for the class. Please make sure you have tested your cases (as well as some of your own), and submit your code to the week 5 online quiz.

Implementing timing

One way of investigating algorithm performance is to time how long it takes to run the game tree. We have provided you some code on MyPlace that will allow you to do this. This code assumes that you have a minimax function that returns a value (-1 or 1). Use the sample function on MyPlace to test your project. Note, you do not need to submit anything for this. This is to test your code:

```
output = (test_timing(([5,5],2)))
print("Time taken", output[0])
print("Value returned", output[1])
```

You should get something along the lines of:

```
Example Play: [([5, 5], 2), ([4, 5], 1), ([3, 5], 2), ([5], 1),
([4], 2), ([1], 1), ([], 2)]
Time taken 0.015095949172973633
Value returned -1
```

Your exact times will be different, but generally, as the piles become more complex, you should expect the performance to be slower.

Sample Minimax Timing Examples

Note, you will not get exactly the same times, you may not even get the same paths, but you should get the same value returned. The times are just an indication, based on my laptop.

```
output = (test_timing(([4],1)))

Example Play: [([4], 1), ([1], 2), ([], 1)]
Time taken 0.0
Value returned 1
```

```
output = (test_timing(([2,3],1)))

Example Play: [([2, 3], 1), ([2, 2], 2), ([1, 2], 1), ([1], 2),
([], 1)]
Time taken 0.0005006790161132812
Value returned 1
```

```
output = (test_timing(([9,9],1)))

Example Play: [([9, 9], 1), ([8, 9], 2), ([7, 9], 1), ([4, 9],
2), ([3, 9], 1), ([9], 2), ([8], 1), ([5], 2), ([4], 1), ([1],
2), ([], 1)]
Time taken 60.8878288269043
Value returned 1
```

```
output = (test_timing(([5,5,5],1)))

Example Play: [([5, 5, 5], 1), ([4, 5, 5], 2), ([1, 5, 5], 1),
([5, 5], 2), ([4, 5], 1), ([3, 5], 2), ([5], 1), ([4], 2), ([1],
1), ([], 2)]

Time taken 47.00967264175415
Value returned -1
```