Computer and Information Sciences
University of Strathclyde

Week 6 – Game Playing with Min and Max, part 1 (worth 5% of your overall grade)

Aim

This lab builds on your previous week of developing a game tree for Nimsticks, 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. However, the minimax algorithm can become very slow, e.g. it already takes around one minute to compute the value of ([5,5,5],1). This can be significantly improved using pruning. This week, you will take your initial minimax tree and apply alpha-beta pruning and optimisation.

Note: The times in here are just guidelines, your times may vary!

Implementing α-β-pruning

Last week, you created a minimax function. In your week6 script, you should create a modified version of it:

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

The aim is to replace your max and min functions with two new functions, called by your new function.

```
def min_value_prune (state, alpha, beta):
     # computes the minimum value with pruning

def max_value_prune (state, alpha, beta):
     # computes the minimum value with pruning
```

- Your lecture notes from week 5 provide a detailed algorithm which you can follow. You will likely find that you have the basics of an algorithm from last week, and you just need to create a modified version

- To test your program, whenever pruning occurs your functions should display what kind of pruning, at what state and for which values for alpha and beta. For example:

  ```
  alpha pruning at ([2], 2) with alpha = -1 and beta = 1
  ```

- You can test your new function by modifying your timing function:
  ```
  output = (test_timing(([2,2],2)))

  alpha pruning at ([1, 1], 2) with alpha = -1 and beta = inf
  beta pruning at ([2], 1) with alpha = -inf and beta = 1
  beta pruning at ([1], 1) with alpha = -inf and beta = -1
  beta pruning at ([2, 1], 1) with alpha = -1 and beta = 1
  beta pruning at ([2], 1) with alpha = -inf and beta = 1

  Example Play: [([2, 2], 2), ([1, 2], 1), ([1], 2), ([], 1)]
  Time taken 0.003
  Value returned 1
  ```

- **Note that the pruning is highly dependent on the order you search the game tree, so you may not get the exact same pruning. However, your algorithms should be much faster!**

- You should now find that when you run the state [5,5,5], you get much improved performance. You can also disable the pruning print comments to speed up performance

Sample Minimax Pruning Examples

Some Sample pruned minimax outputs, pruning comments disabled. Try running these on your original algorithm to assess performance. We assume here that you have not optimised or removed duplicates.

```
output = (test_timing(([8,8],2)))

Example Play: [([8, 8], 2), ([5, 8], 1), ([4, 8], 2), ([1, 8],
1), ([8], 2), ([5], 1), ([4], 2), ([1], 1), ([], 2)]

Time taken 0.08441495895385742
Value returned -1
```

(around 6.4 seconds without pruning)

```
output = (test_timing(([6,4,2,3],1)))

Example Play: [([6, 4, 2, 3], 1), ([5, 4, 2, 3], 2), ([4, 4, 2,
3], 1), ([1, 4, 2, 3], 2), ([4, 2, 3], 1), ([1, 2, 3], 2), ([2,
3], 1), ([2, 2], 2), ([1, 2], 1), ([1], 2), ([], 1)]

Time taken 0.7134995460510254
Value returned 1
```

(around 356 seconds without pruning)

```
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 0.31122922897338867

Value returned -1
```

(around 69.63 seconds without pruning)

Computer and Information Sciences
University of Strathclyde

Minimax Optimisation

Although alpha and beta pruning optimised your algorithm and improved processing speed , this can still be a slow game when you have a lot of options. For example:

```
output = (test_timing(([8,8,8],1)))
Time taken 126.672
Value returned 1

output = (test_timing(([20,20],1)))
Took so long to run that I got bored!
```

- If you haven't done so already, you can optimise your solution to ensure you remove duplicates. For example, if the initial state is [2][2], then the possible states are [1][2], [2], [2][1], [2]. However, you should be able to see that [2] and [2] are duplicates, and [1][2], and [2][1] are identical. So rather than add 4 options to the agenda, you should just add [2] and [2][1].

  ```
  output = (test_timing(([8,8,8],1)))
  Time taken 34.42
  Value returned 1
  ```

- You can further optimise your algorithm to make it run a little faster by doing something similar to what you did in the final stages of the MIU practical. You can implement a dictionary to store the values of any nodes you have already explored.

- Note, that depending on your dictionary implementation, you may mess up your paths! You do not need to worry too much about this, as long as the value is correct, that is what matters.

- By doing this, you can reduce your time to process trees considerably:

  ```
  output = (test_timing(([8,8,8],1)))
  Time taken 0.06011080741882324
  Value returned 1

  output = (test_timing(([20,20],1)))
  Time taken 0.16210103034973145
  Value returned 1

  output = (test_timing(([10,10,10],2)))
  Time taken 0.48703622817993164
  Value returned -1

  output = (test_timing(([6,4,2,3,5,5,5],1)))
  Time taken 1.4752154350280762
  Value returned 1
  ```

Computer and Information Sciences
University of Strathclyde

Submission Instructions

- This practical is worth 5% of the mark for the class. You will submit online in the Week 6 lab submission on MyPlace. You should remove all the print statements from your code. Evaluation will look at the correct final value, and also the time taken to run. If your code is not sufficiently optimised, then it may not run at all (and will give you an error).

Modified February 2024
By Dr Andrew Abel, based on Dr Clemens Kupke