Othello 6: From Negamax to Alpha-beta

AI - Jan 2122

Most game playing software uses some form of Minimax, such as Negamax, to analyze the game state, in order to decide what the best move is going forward.

We have seen pseudo code for Negamax, for Othello purposes, along the lines of:

```
def negamax(brd, tkn):
  if no possible moves for tkn:       # if tkn can't move
    if no possible moves for enemy:   #   if game is over
      return [the score]
    nm = recurse on negamax           #   game not over; tkn passes
    return the right thing

  best = min(negamax(makeMove(brd, tkn, mv), enemy) + [mv]
             for mv in possible moves for tkn)
  return [-best[0]] + best[1:]
```

The difference between Minimax and Negamax is that Minimax always evaluates the board from a fixed point of view (such as positive being good for white in chess, and for 'x' (or black) in Othello), while Negamax attempts to maximize things for the player in question (`tkn` in the case above)

Alpha-beta is a way of implementing Minimax or Negamax to get even more speed out of it, which translates into being able to run it to a greater depth. It is an enhancement (ie. improvement) on Minimax/Negamax, rather than being distinct. The idea is to establish that the score going down a particular branch must be within certain bounds to be of use, and if the actual score falls outside the specified bounds, to conclude that further investigation of the branch is not needed. The pseudo code for Alpha-beta applied to Negamax could be written as:

```
def alphabeta(brd, tkn, lowerBnd, upperBnd):
  if no possible moves for tkn:       # if tkn can’t move
    if no possible moves for enemy:   #   if game is over
      return [the score]
    ab = recurse with alphabeta       # game not over
    return the right thing

  best = [lowerBnd-1]                 # guarantees best will be set
  for mv in possible moves for tkn:
    ab = alphabeta(makeMove(brd,tkn,mv), enemy, -upperBnd, -lowerBnd)
    score = -ab[0]                            # Score from tkn’s viewpt
    if score < lowerBnd: continue             # Not an improvement
    if score > upperBnd: return [score]       # Vile to the caller
    update best                               # Else it’s an improvement
    lowerBnd = score+1

  return best
```

This code has some analogy to the top level version of Negamax in that the comprehension in the main body has been unrolled to a loop. In particular, in the loop, as one gets back a score from `alphabeta()` going down a particular branch, after converting it from the enemy's point of view to that of `tkn`, there are three scenarios. The first scenario, which also holds in Negamax sans Alpha-beta, is that the score is not better than the best score from a prior branch. In other words, `tkn` has done better earlier; therefore, this new branch is not of interest to `tkn`.

The remaining scenarios, that the returned score from the current branch improves upon the prior best score, can be divided into two distinct parts. The usual case here, which also holds with Negamax, is when the improvement is moderate, within the bounds of "reasonableness". "Reasonableness" here is defined by `upperBnd`. In this case, the code updates what it intends to return (`best`) and updates the `lowerBnd` (that is to say, this branch is now the best branch), and continues processing.

However, in the remaining option, which only holds with Alpha-beta, there can be a case of too-much-of-a-good-thing (meaning that the score is greater than `upperBnd`). This is where the branch is so good that `tkn` would be overjoyed to have it, but the enemy has already seen a branch (higher up) that is better for it (and less good for `tkn`), precluding this *entire set* of branches. Specifically, all remaining branches that have not yet been tried by `tkn` are ignored (pruned). Note that `[score]` is returned sans moves. This is since we already know the caller will repudiate this branch (based on the score) and never use it.

Othello 6: From Negamax to Alpha-beta

This lab has several components and is not simply about writing the code, but also about exercising the code and reporting the results. Broadly speaking, this lab has three parts:

A: Upgrade your Othello 5 code to play a tournament if it receives no command line input
B: Upgrade your Othello 5 code to Othello 6 by implementing Alpha/Beta
C: Record stats (time, percent) as you push the limit (number of holes) of when Alpha/Beta kicks in

Part A)

In your script, on line 2, you should have your name. On line 3, you should have a global for when you will run your negamax routine. Your first three lines should look like:

```
import sys; args = sys.argv[1:]
# Jo Student
LIMIT_NM = 11
```

Make certain that your negamax() routine is called when the number of holes is less than LIMIT_NM and not less than or equal to it.

Part A2)

Ensure that your script is an extension of Othello 4. In other words, the control structure at this point should look like:

```
def main():
  if args:
    # Othello 3
    brd, tkn, moves = parseArgs()        # string, char, list of ints
    initialize any other variables

    for mv in mvs:
      if mv < 0: continue                # Ignore negative vals
      if not findMoves(brd, tkn): tkn = 'xo'[tkn== 'x']    # tkn can't move
      # show snapshot                    # Uncomment to see game (Othello3)
      brd = makeMove(brd, tkn, mv)
      tkn = 'xo'[tkn== 'x']              # Other side gets next turn

    # Othello 4
    if not findMoves(brd, tkn): tkn = 'xo'[tkn== 'x']      # tkn can't move
    show snapshot
    qm = quickMove(brd, tkn)
    print(f'My preferred move is {qm}')

    # Othello 5/6
    if number of holes < LIMIT_NM:
      call negamax routine
```

The Othello 3 part is there to help you debug. It is crucial, especially if you come to me for help, as you should be using your own code to help you diagnose issues you might be having. Othello 4 is there in case your negamax (or alpha-beta) times out. The script shown on the prior page could be used by the grader to run a tournament against Random. Note that `show snapshot` is commented out in the Othello 3 section. This is a matter of personal preference, but my thinking is that in the development of Othello 3, I cared about the choices that my tokens were making, but at this point, I only care about the choices that negamax is making. Keeping the snapshot (in the loop) would result in very lengthy output if I feed my Othello script a game transcript. A person might consider adding an additional optional argument (such as v, for verbose) to the command line to cause that commented out `show snapshot` to work.

Part A3)

You should have some idea of how your script is behaving. Of course, you should first test your script on a concrete board or two. Do this before moving on to Part A3. Once you have the notion that your script might be working, you'd like to know how well it's working, and to that end, you will have it run a tournament of 100 games against Random if the script receives no command line input. For every other game in the tournament, your script should alternate the token that it uses (ie. x in one game, o in the next, then back to x …).

In particular, you should note each game's result (ie. the score - your token count less the enemy token count), displaying 10 on a line. Then, omit a line and on the following line show how many tokens you captured and sum of the total number of tokens on the board over all the games. On the next line, give the Score (in percent) this implies. On the following "four" lines, show the worst two games as
Game *#* **as** *tkn* **=>** *Score:*
CondensedGameTranscript
where *tkn* indicates your token for the game. The first game number in the tournament is 1 and the last game is 100. Finally, display the total elapsed time, in seconds, with a single decimal point (not forgetting the s). See part C for a concrete example of how it should look.

Run a tournament with LIMIT_NM set to 11 and note the total time taken and the reported score as you will be submitting this as your baseline. If you determine that the time to run the entire tournament is unreasonable on your machine, then decrease LIMIT_NM. On my laptop, a tournament might take around 600 seconds.

Part B)

Make a copy of your code and now implement alphabeta() to replace negamax(). alphabeta() is meant to take a board, token, lowerLimit (alpha) and upperLimit (beta) as arguments. Alpha and beta are meant to be the lower and upper bound of scores that would be accepted as improvements over the current score. To be clear, both alpha and beta would be accepted as improvements in the score, but alpha-1 and beta+1 would not be accepted.

Also, update the constant that you have on line 3 to LIMIT_AB, making sure your alpha-beta routine is called when the number of holes is less than LIMIT_AB and not less than or equal to it. Test your new Othello 6 script on the same boards that you tested your Othello 5 script on, and it should be faster.

Part C)

You have already recorded the results of a 100 game tournament against Random with your Othello 5 script in part A3. You will now record a sequence of tournaments using your Othello 6 script. Specifically, you will set `LIMIT_AB = 1` in your code and run a 100 game tournament against Random, recording (timeInSeconds, scoreAsPercent). The time taken should have a single decimal point. `LIMIT_AB = 1` amounts to Othello 4. Now, set `LIMIT_AB = 2` and do the same thing. Keep incrementing `LIMIT_AB` until your tournament takes over 300 seconds, noting the results.

Recheck that the first three lines in your code look exactly like:

```
import sys; args = sys.argv[1:]
# Jo Student
LIMIT_AB = #
```

The number in the third line should be the largest integer that still resulted in a tournament of less than 300 seconds. I will adjust this as necessary on my back end, but you should try to get it right.

Examples)

Here are two examples with a `LIMIT_AB` of 1 (ie. Othello 4) and 12:

```
D:\Tanitas\TJ\AI\Othello\Web6>ab2021.py
 40  12  62  32  42  30  54  14  18  36
 22  16  36  36  28  22  30  44  12  34
 32  38  34  48  42  18  41  48  38  32
 34  28  38  14  42  28  28  26  28  38
 26  20  40  32  36  42  18   4  16  16
 28  46  34  12  26  53  42  22  34  52
 44  52  30  30  40  14  10  28  38  40
 40  28   4  46  28  30  36  46  10  18
 48  42  28  34  28  28  46  14  48  34
 36  18  38  46  48  20  16  36  52  34

My tokens: 4799; Total tokens: 6398
Score: 75.0%
NM/AB LIMIT: 1
Game 47 as x => 4:
1918371126331743102134_1123844_942242049_045_2_5132522_356_4_6_8164132574840583052534623295
46314_76062153147553959615051
Game 72 as o => 4:
444519523729536130202113382622314234181112_94341511425594633_7_6_5_315_42316_21739475554506
3_058_8_12410324048604957566 2
Elapsed time: 0.4s
```

```
D:\Tanitas\TJ\AI\Othello\Web6>ab2021.py
 46  42  46  54  50  50  58  48  42  42
 52  52  38  46  46  50  52  54  60  54
 40  52  42  52  36  46  44  60  49  58
 40  42  52  46  38  54  42  56  48  61
 45  49  42  46  42  24  54  48  52  50
 48  58  58  48  52  32  46  48  30  56
 46  34  61  44  52  46  61  32  58  44
 58  62  58  50  32  46  54  50  32  56
 62  54  54  30  44  48  44  43  42  56
 50  60  52  54  64  52  40  32  44  46

My tokens: 5603; Total tokens: 6391
Score: 87.7%
NM/AB LIMIT: 13
Game 45 as x => 24:
191826342511434112173313293210525O_24245443020214638224716593714_753_6_9_04956_81548_12340_
55724315158_339606162555463-1_4
Game 58 as o => 30:
37291945222130104418435150233849534241615633573158605952624020483213243411_246542516_82617_
912_3_4_0_5_6_114_71539476355
Elapsed time: 130.3s
```

The latter tournament took 130 seconds which means that Alpha-beta did its thing in under 1.3 seconds when presented with 12 holes. To see how much of a difference Alpha/beta can make, let's take the worst game (#47) from the first tournament example (with a `LIMIT_AB` of 1), where x wound up winning by 4, and see whether that corner choice indicated by _7 was reasonable. That is, feed the sequence up to that point into the Alpha-beta code (part 2A):

```
D:\Tanitas\TJ\AI\Othello\Web6>ab2021.py
19183711263317431021 34_1123844_942242049_045_2_5132522_356_4_6_816413257484058305253462329546
314
xxxxxxx*
xxoooooo*
xxxoxooo
xxoxoxo*
xoxoxxo*
oooxxxo*
xo**xox*
xxx..**x

xxxxxxx.xxoooooo.xxxoxoooxxoxoxo.xoxoxxo.oooxxxo.xo..xox.xxx....x  31/21
Possible moves for x: 7, 15, 31, 39, 47, 50, 51, 55, 61, 62
My corner move is 7
Min score is 22; rev. move sequence: [51, 62, 59, 50, 61, 60, 55, 39, 47, 31, 15, 7]
Min score is 24; rev. move sequence: [39, 47, 31, 62, 51, 59, 60, 61, 55, 50, 7, 15]
Min score is 26; rev. move sequence: [61, 62, 51, 60, 7, 15, 59, 47, 31, 39, 55, 50]
Min score is 32; rev. move sequence: [39, 31, 47, 55, 61, 60, 15, 7, 59, 50, 62, 51]

Min score: 32; move sequence: [39, 31, 47, 55, 61, 60, 15, 7, 59, 50, 62, 51]
Elapsed time: 3.3s
```

Above, the input to Othello 6 was a compact move sequence. "My corner move is 7" is Othello 4, which resulted in the poor showing in the tournament (helped along by future poor choices on the part of x in the tournament). It makes sense to try our presumed best move first with Alpha/Beta, and we see it guarantees a minimum score of 22. Already this would be a huge improvement over what happened in the original showing. As Othello 6 tests other branches, it sees the improvements walk up to a score of 32.

Grading)

You will submit the following to a google form.
Baseline results from your negamax tournament in part A3 (LIMIT_NM, time taken, score).
The sequence of results from your tournaments in part C.
Othello 6 code, making certain that the first 3 lines conform to what is shown at the end of part C

When I look at your submission in the google form, if the first three lines don't match what I'm expecting, I go no further, and it is as if you hadn't submitted. If your code conforms, then I will give it a few board/token combinations to make sure that your code doesn't topple over, and outputs a correct min score and reverse sequence. At that point, I'll run a tournament with your code.

All items entered, script conforms, and does not topple over on the initial board/tkn tests: 55%
Script outputs a snapshot, preferred move prior to starting alpha beta: +5%
Script accepts a sequence of moves: +5%
Script accepts a condensed sequence of moves: +5%
Statistics look reasonable: +5%
Code runs a tournament of 100 games in the absence of input: +5%

The remaining points are from the performance of your script on the tournament. Your code gets 300 seconds to run, and it will finish in some time T. If it exhausts my patience and does not finish within 300 seconds, it gets nothing, and I'll try with a smaller LIMIT_AB. If it runs really quickly, it will get the max points allowed for that LIMIT_AB. If T is within the range of [20*(LIMIT_AB-8), 300], then the scoring is linear within the max points / min points shown below

LIMIT_AB	Max points	Time interval	Min points
9	2.5	[20, 300]	0
10	5	[40, 300]	2.5
11	7.5	[60, 300]	5
12	12.5	[80, 300]	7.5
13	17.5	[100, 300]	12.5
14	20	[120, 300]	17.5
15	20 points		

9 through 11 are just Negamax, while 12 and 13 are most likely for what alpha-beta will achieve without significant tweaking.

Two important notes:
1) I will want all the fields (except comments) filled out each time you submit. If you make tweaks which are minor in the grand scheme of things, you do not have to rerun the tournaments. In that case, you can resubmit with the same tournament information – so make sure to save that info. I will not look at your prior submission for this info, so make sure to retain these numbers in case you need to reuse them. If you make a significant change, you should, of course, rerun the tournaments.

2) If alpha-beta has not yet worked out for you, I still want a submission from you. In that case your negmax will stand in for alpha-beta. That means that your tournament results will be from negamax and your final entry in the list will match the Negamax vals.