UNIVERSITY OF THE WITWATERSRAND, JOHANNESBURG
**School of Computer Science & Applied Mathematics**

# Advanced Analysis of Algorithms
# 2021 Assignment: Part 6

## 1 Introduction

At this point in the project, you have now implemented two AIs to play the game: a minimax with simple evaluation, and alpha-beta pruning with slightly more complex evaluation. The final step is now to play games to see how well they do. **Please note the very last sentence of this document: you must submit your code, or else you will receive 0 for this part.**

## 2 To-Do

If you are working in groups, do not forget that you must still implement two additional modifications to your algorithms. These can be anything from faster move generation (bitboards) to iterative deepening search, to quiescence search, to better move ordering, etc. This means that if you are working individually, you will have two AIs to compare (unless you implemented something more, which is great but not strictly necessary), but if you worked in groups, you will have three: minimax, alpha-beta pruning and your final one with all the modifications.

This section will involve running games amongst the various agents. Each agent should play every other agent twice — once as white and once as black. The section below describes the rules that should be used when running the game.

## 3 Rules

The starting position of the game is given by the FEN string `2ele1z/ppppppp/7/7/7/PPPPPPP/2ELE1Z w 4`. As normal, one player wins when they capture the opposing lion. If each side has played 100 moves and the game has yet to end, declare the game a draw.

## 4 Extracting Moves

Our AI has so far only returned the score at the current position, but this does not help us play the game! We need to know what the best move is so we can actually play it. There are various ways to extract the move (we can even extract all the best future moves using an approach like this: `https://www.chessprogramming.org/Principal_Variation`), and below is some pseudocode for doing just that. The main idea here is that we have a boolean that tells us whether we are at the root node and, if so, records the best move (passed by reference):

```
/// pass bestMove by pointer or reference (since the function itself returns an int)
function minimax(state, depth, bestMove, first):
    if state.gameOver() or depth <= 0:
        return board.evaluate()
    value = -INF
    for each move in state.generateMoves():
        nextState = state.makeMove(state)
        eval = -minimax(nextState, depth-1, bestMove, false)
        if eval > value:
            value = eval
            if first:
                bestMove = move
    return value
```

This would then be called with `first=True`. A similar approach can be taken for alpha-beta pruning, where the best move is recorded when the evaluation returned is greater than alpha.

## 5 Running Games

You may decide how best to run games, which will vary depending on how you have structured your code. For example, you may have a shared `Board` object that you provide to the two agents playing a game, and every time a player makes a move, this board is updated. Below is some pseudocode for one approach to running a game, where we assume white is a minimax agent, and black is an alpha-beta agent.

```
function playGame(board):
    sideToPlay = 'w'
    moveCount = 0
    while moveCount < 100:
        position = board.toFen()
        if sideToPlay == 'w':
            move = minimax(position, depth=3)
        else:
            move = alphabeta(position, depth=6, alpha=-INF, beta=+INF)
            moveCount++ // update moves after black plays

        board.makeMove(move)
        if board.gameOver():
            return sideToPlay  // side to play just won

        //swap sides
        if sideToPlay == 'w':
            sideToPlay = 'b'
        else:
            sideToPlay = 'w'
    // we played 100 moves with no winner
    return "DRAW"
```

The kind of game you should play depends on how you have implemented everything. If your AI can search to a fixed depth only, then you should select a depth (greater than 1) for each agent such that they take roughly the same amount of time to return a move (e.g minimax to depth 2 and alpha-beta to depth 4). However, if your agent's are always able to return a move after a fixed amount of time (e.g. you're using iterative deepening), then play your games with fixed time controls (e.g. each agent is allowed X seconds per move).

# 6 Report

The LaTeX file below provides the format of the report and the details that need to be filled in. I recommend opening it in Overleaf or another LaTeX editor and replacing the content of the sections with your own.

The report is worth $10\%$ for individuals, and $20\%$ for groups, and will be marked on the following basis:

**For individuals:**

- Description of alpha-beta pruning and the evaluation function that demonstrates understanding of the methods — 5%
- Results table produced and shown — 5%

**For groups:**

Groups must implement additional modifications, otherwise no marks will be awarded.

- Description of alpha-beta pruning and the evaluation function that demonstrates understanding of the methods — 5%
- Description of two additional modifications that demonstrates understanding of the methods — 10%
- Results table produced and shown — 5%

# 7 Submission

Submit to Moodle the following files:

- A PDF of the report. Be sure to include your names and student numbers. If in groups, only one member needs to submit
- All of your code, including your AI algorithms and code used to generate the results. The code may be spread across multiple files, in which case attach it as a zip file. **A submission without code will be given 0!**