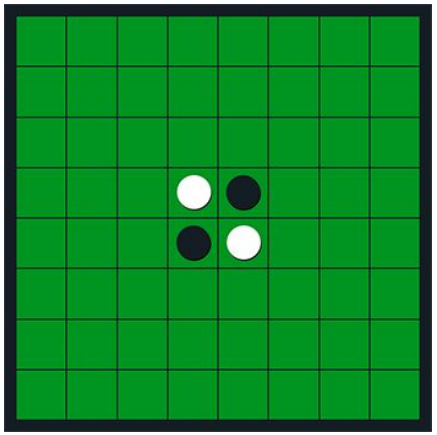
Othello: Minimax on Different Boards

Bryanna Geiger: bg2603

What is Othello?

- Board Game → derived from Reversi (original game)
- 8x8 square grid
  - My implementation: text output for the board
- Start state: 2 black pieces, 2 white pieces
  - occupying the middle 4 squares
- A disk (or row of disks) is surrounded by the opposing color
  - The surrounding disks will be flipped
  - How a player earns “points”



- Standard Initial Board State Above
  - My implementation: Board 1

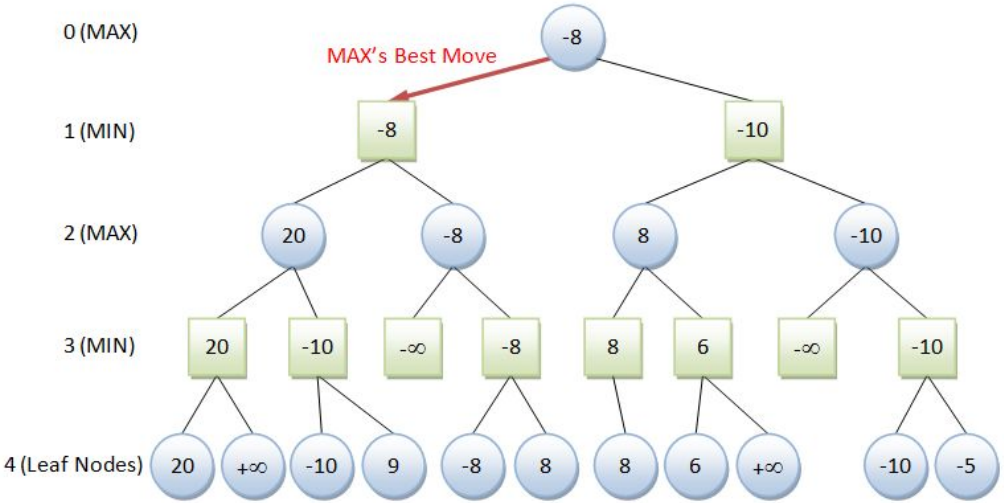
What is Othello continued:

- End Game:
  - When there are no valid moves for either player
    - A turn is skipped if player 1 cannot go, but player 2 still can go
  - My implementation
    - As soon as any player cannot go, game ends
- Winning:
  - The player with more of their colored disks wins (can end in a tie, although it is uncommon)

Figure 1: An example of the Othello board game

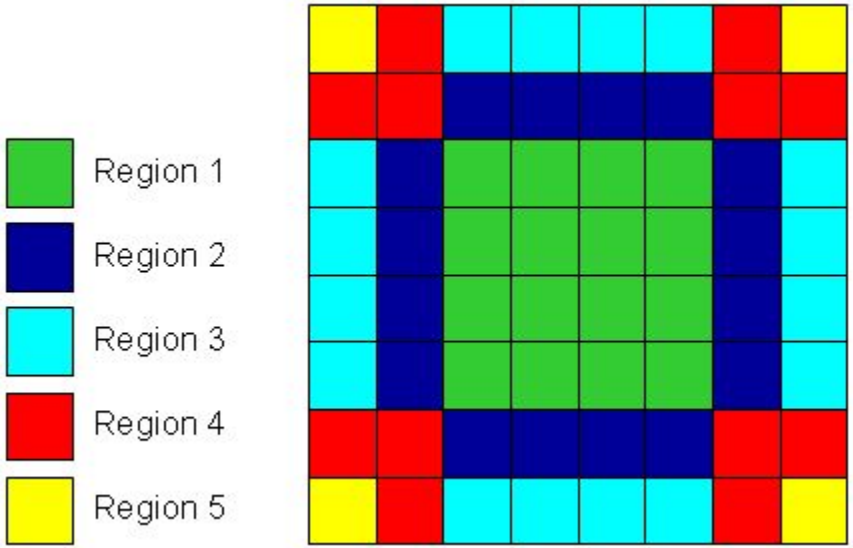
Minimax Algorithm

- Backtracking Algorithm ⟶ Recursive
- As a searching algorithm ⟶ commonly used in games
  - 2048, chess, othello, checkers, go, sudoku, ken ken
- Two players
  - “Maximizer” and “Minimizer”
  - Each player will either try to maximize value of a move while the other player will try to minimize



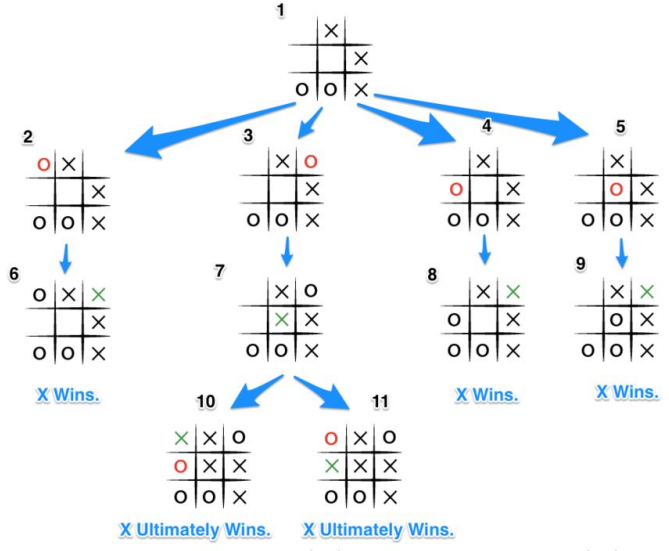
Minimax and Heuristics

- Different heuristics can be implemented
  - Will adjust values of the board
  - i.e. in 2048: a move that keeps the highest valued piece in a corner would have a higher value than moving it
  - i.e. in Othello, the corner spots are ideal and would have a higher value



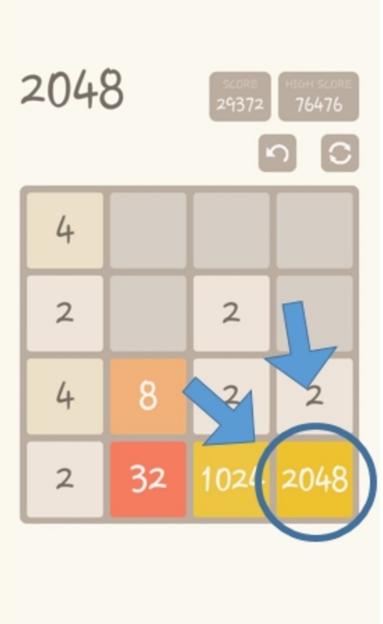
Approach and Background

- COMS 4701: Artificial Intelligence
  - Coded the minimax algorithm with alpha-beta pruning in Python
  - A matter of translating Python code to Haskell
  - Similar format and approach to the game algorithm itself
- Tic-Tac-Toe → Othello
  - Started with a scaled down version of a game similar in nature to Othello: Tic-Tac-Toe
  - Implemented minimax on tic-tac-toe
  - Applied to Othello: small scale → larger scale



Approach and Background continued:

- Various minimax implementations in different games
  - Tic-tac-toe, 2048, chess, go, etc.
  - Apply approaches to Othello
- Reference site:
  - For reference: http://www.pressibus.org/ataxx/autre/minimax/paper.html
  - Particularly helping in determining how to address applying minimax to different boards
- Goal:
  - Different boards to run Othello on
    - Apply minimax to different boards at different depths, rather than just the standard Othello game
  - Comparing the results across different boards
  - Parallelizing the minimax algorithm



Parallelization: Minimax

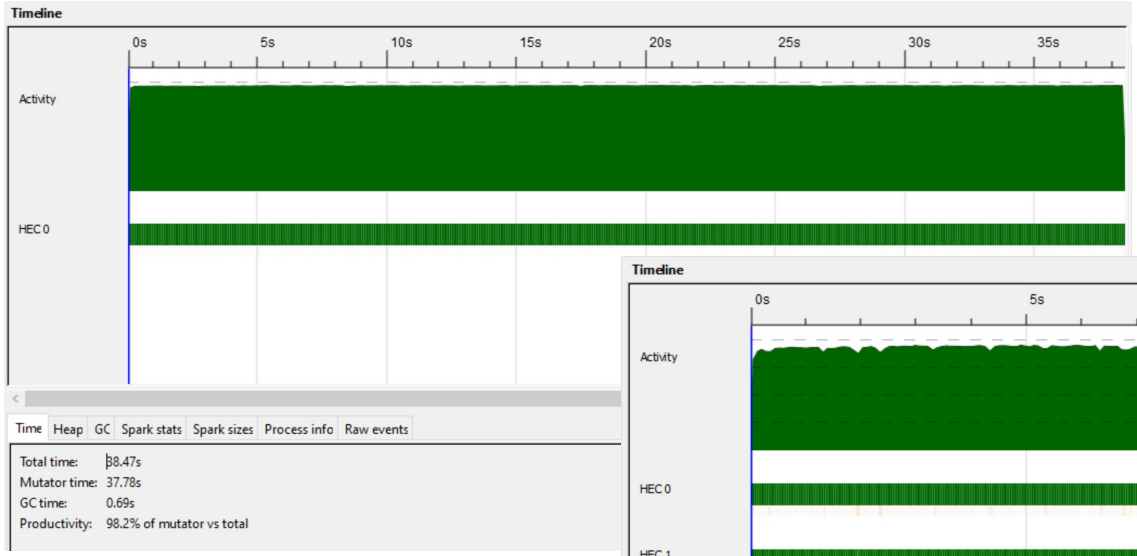
- ‘using’ parList rseq
- parList
  - Evaluates the list elements in parallel
- Straightforward in terms of implementation
  - Largely using lists ⟶ parList made sense to use
- Basis behind approach:
  - Sudoku
    - Game playing algorithms: a Sukoku version took a similar approach so I thought it would be applicable to Othello for the minimax algorithm implementation

Minimax Psuedocode

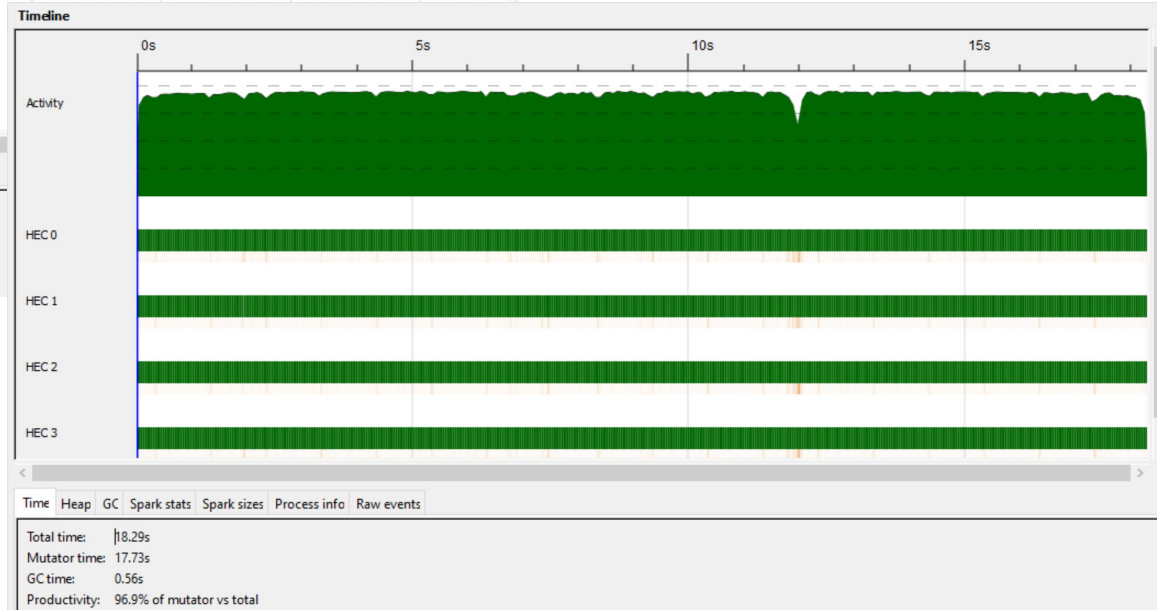
```
minimax(in game board, in int depth, in int max_depth,
        out score chosen_score, out score chosen_move)
begin
    if (depth = max_depth) then
        chosen_score = evaluation(board);
    else
        moves_list = generate_moves(board);
        if (moves_list = NULL) then
            chosen_score = evaluation(board);
        else
            for (i = 1 to moves_list.length) do
                best_score = infinity;
                new_board = board;
                apply_move(new_board, moves_list[i]);
                minimax(new_board, depth+1, max_depth, the_score, the_move);
                if (better(the_score, best_score)) then
                    best_score = the_score;
                    best_move = the_move;
                endif
            enddo
            chosen_score = best_score;
            chosen_move = best_move;
        endif
    endif
end.
```

```haskell
minimax :: Int -> Othello -> Board -> Int
minimax dpth col b
    | endGame = if (adv col b) > 0
      then 100000
      else -100000
    | dpth <= 0 = adv col b
    | otherwise = if ( moves (changeColor col) b ) /= []
      then -maxPt
      else maxPt
    where
      endGame = null (moves col b) && null (moves (changeColor col) b)
      clrUp = if ( moves (changeColor col) b ) /= []
          then changeColor col
          else col
      nm = if clrUp /= col
          then ( moves (changeColor col) b )
          else ( moves col b)
      maxPt = maximum (map (minimax (dpth - 1) clrUp . move clrUp b) nm `using` parList rseq)
```

Threadscope Comparison



- Total Run Time
  - 1 Core: 38.47 s
  - 4 Cores: 18.29 s



- Top Left Image:
  - Board 1, Depth 4, 1 Core
- Right Image:
  - Board 1, Depth 4, 4 Cores

Runtime comparison by depths (2 and 4)

Total Run Time	Depth of 2	Depth of 4
Board 1	0.82 seconds	18.29 seconds
Board 2	0.578 seconds	6.84 seconds
Board 3	0.984 seconds	36.65 seconds

- Note that the default depth size is set to 4, although this can be altered when depth is declared in the code
- Run with 4 Cores at depths of 2 and 4

Runtime comparison across boards (1-3)

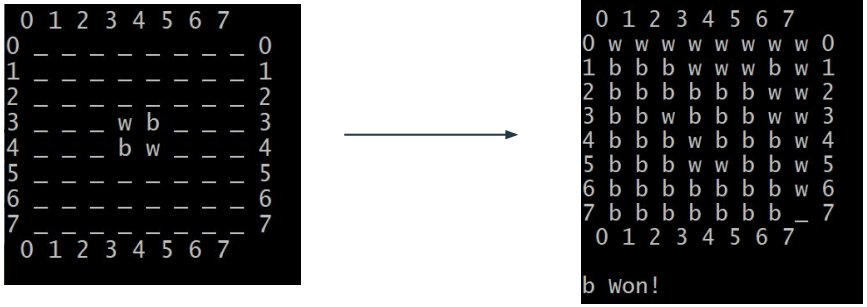
Total Run Time (seconds)	With 1 Core	With 4 Cores	Difference
Board 1	38.47 seconds	18.29 seconds	20.17 s (47.54%)
Board 2	17.24 seconds	6.84 seconds	10.4 s (39.68%)
Board 3	93.46 seconds	36.65 seconds	56.81 s (39.21%)

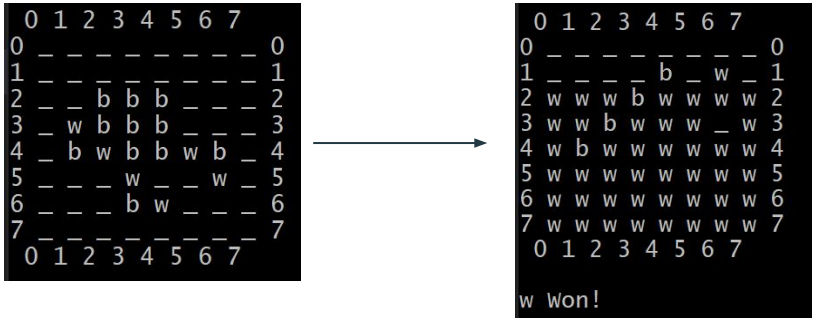
- Runtime comparison across board 1, board 2, and board 3
- Run with the default depth of 4
- Interesting note
  - Board 3 is currently at the longest runtime, despite being midway through the game as the initial board state

Demo

- Pre-recorded video demo
  - Change video quality if blurry (https://www.youtube.com/watch?v=aPAZCjxofkk)
  - Run on board 1, board 2, and board 3
  - Depth of 4 is used in the demo
  - 4 cores are used in the demo video
  - Commands used:
    - ./test board1 +RTS -N4 -s
    - ./test board2 +RTS -N4 -s
    - ./test board3 +RTS -N4 -s
    - It can be run without an argument, however, it will default to running board 1
- Boards:
  - Board 1: default board, the way an actual othello game would start
    - 2 b, 2 w pieces
  - Board 2: mid-game
    - 10 b, 6 w
  - Board 3: mid-game
    - 7 b, 11 w

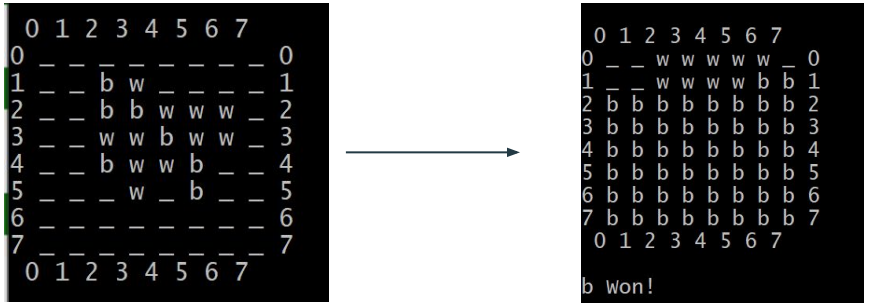
Board 1: Start and end states



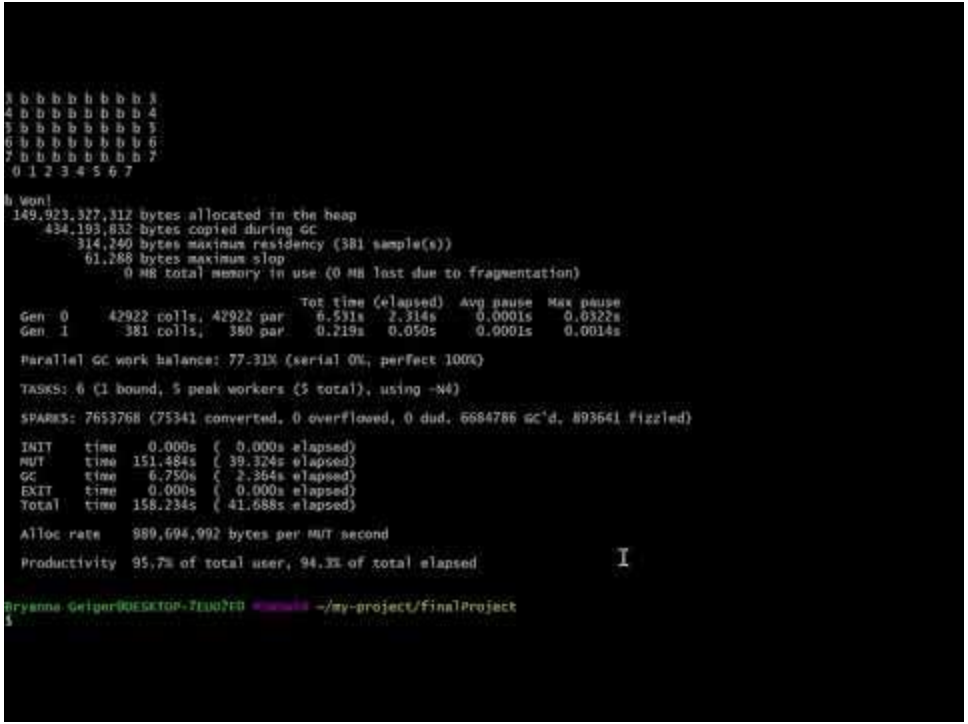


Board 2: Start and end states

Board 3: Start and end states



Demo


b won!
149,923,327,312 bytes allocated in the heap
434,193,832 bytes copied during GC
314,240 bytes maximum residency (381 sample(s))
61,288 bytes maximum slop
0 MB total memory in use (0 MB lost due to fragmentation)
Tot time (elapsed) Avg pause Max pause
Gen 0 42922 colls, 42922 par 6.531s 2.314s 0.0001s 0.0322s
Gen 1 381 colls, 380 par 0.219s 0.050s 0.0001s 0.0014s
Parallel GC work balance: 77.31% (serial 0%, perfect 100%)
TASKS: 6 (1 bound, 5 peak workers (5 total), using -N4)
SPARKS: 765376B (75341 converted, 0 overflowed, 0 dud, 6684786 GC'd, 893641 fizzled)
INIT time 0.000s (0.000s elapsed)
MUT time 151.484s (39.324s elapsed)
GC time 6.750s (2.364s elapsed)
EXIT time 0.000s (0.000s elapsed)
Total time 158.234s (41.688s elapsed)
Alloc rate 989,694,992 bytes per MUT second
Productivity 95.7% of total user, 94.3% of total elapsed
~/my-project/finalProject

Going Forward

- To implement:
  - Alpha-beta pruning
  - User Interaction
    - Option for the user to actually play the Othello game themselves
    - Keep options to run on different boards
  - Visualization
    - Othello board that is not just a text output
    - Being able to click where to move would be an interesting application
  - Parallelization
    - Explore different strategies, approaches, and places where code can be optimized
  - Heuristics
    - Implementing heuristics for Othello
      - If a corner is an option, the player should make that move, etc.
  - Troubleshooting
    - Comparing runtimes of different boards
    - Board 3 is the longest at ~40 seconds running on 4 cores vs Board 1 at ~20 seconds on 4 cores

References

Google Doc Link With Full References Sheet:

https://docs.google.com/document/d/1KhxJaxueMmcWSAnAHaaKYi5vADCF26gKcKw655Q4KOs/edit?usp=sharing