CSP Word Sudoku

Introduction

In Part 1, we implement an algorithm to solve a word sudoku puzzle as a constraint satisfaction problem. The word sudoku puzzle consists of a 9x9 grid of cells, with or without hints, that must be filled with words from a given word bank. The CSP is solved using a backtracking search.

CSP

Our formulation of the CSP involves the assignment of entire words to the grid. The variables are the individual words from the word bank, which have a position, which is the location of the first letter of the word in the grid, and an orientation. The domains of these values are the x and y coordinates of every cell in the grid for the position, and either horizontal or vertical for the orientation. The constraints are described below, along with the variables and domains.

Variables:	$((x_i, y_i), z_i)$
Domains:	$x_i, y_i \in [0,8], z_i \in \{horizontal, vertical\}$
Constraints:	$Each\ row, col, and\ 3x3\ block\ cannot\ have\ duplicate\ letters$
	$Words\ must\ fit\ inside\ the\ grid$
	$When\ words\ overlap\ (or\ overlap\ with\ hints), letters\ must\ be\ consistent$

This formulation of the CSP allows for a reasonable number of variables and possible values as compared to other methods, such as each variable representing a letter, rather than a full word. In this case, a backtracking search is appropriate, as opposed to other algorithms such as local search or hill-climbing search, because this CSP is more tightly constrained, and may have only a few or one solution.

Implementation

We implemented our backtracking search by performing a depth-first search with single-word assignments.To make our backtracking search more efficient, we implemented ordering heuristics and early detection of failure.

The most constrained variable heuristic allows us to choose the next variable to assign as the one with the greatest length, or equivalently, the longest word. Longer words are more tightly constrained because they fit in fewer positions and orientations, so they have fewer possible values and a smaller branching factor. Rather than using the most constraining variable heuristic as a tiebreaker between words of the same length, we implemented another most constrained variable heuristic to break ties. We choose the word with the most letters already assigned to the grid, because these assignments impose further constraints upon that word. We chose not to implement a least constraining value heuristics because we deemed as unwarranted the extra complexity of finding the least constraining value.

In addition, we implemented early detection of failure through forward checking. We keep track of all of the remaining legal values of each unassigned variable. After an assignment is deemed to satisfy all constraints, we check each unassigned variable to remove any illegal values, and terminate the search when any variable has no legal values. Forward checking allows us to reduce the space of possible assignments, so that we can detect failures earlier.

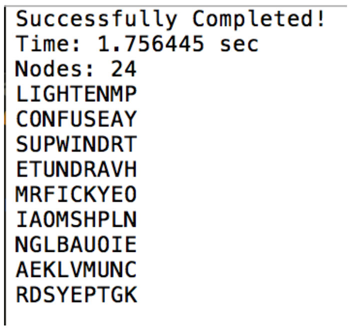
Results

Below are the results provided for each of the grids and word banks. We found that our algorithm performed very well in terms of time, especially with our implementation of forward checking. The number of nodes expanded is also very low, and very close to the number of variables, implying that our ordering heuristics and early detection of failure greatly reduced the amount of backtracking performed.

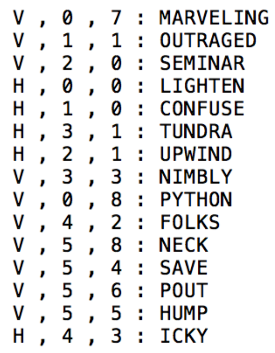
Each sudoku game shows:

1. Solution Grid
2. Sequence of assignment
3. Number of nodes expanded
4. Execution time

Grid 1 results:

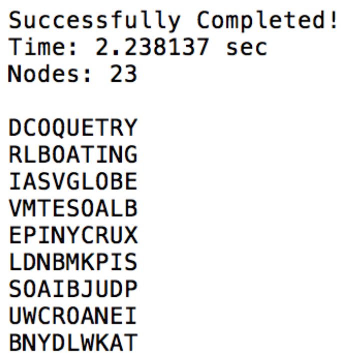


```
Successfully Completed!
Time: 1.756445 sec
Nodes: 24
LIGHTENMP
CONFUSEAY
SUPWINDRT
ETUNDRAVH
MRFICKYEO
IAOMSHPLN
NGLBAUOIE
AEKLVMUNC
RDSYEPTGK
```



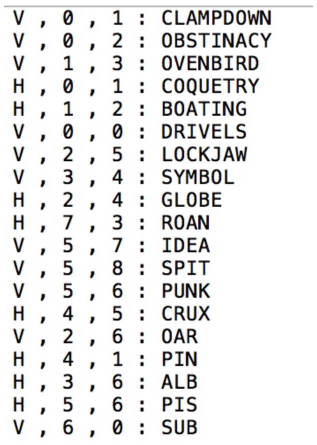
```
V , 0 , 7 : MARVELING
V , 1 , 1 : OUTRAGED
V , 2 , 0 : SEMINAR
H , 0 , 0 : LIGHTEN
H , 1 , 0 : CONFUSE
H , 3 , 1 : TUNDRA
H , 2 , 1 : UPWIND
V , 3 , 3 : NIMBLY
V , 0 , 8 : PYTHON
V , 4 , 2 : FOLKS
V , 5 , 8 : NECK
V , 5 , 4 : SAVE
V , 5 , 6 : POUT
V , 5 , 5 : HUMP
H , 4 , 3 : ICKY
```

Grid 2 results:



```
Successfully Completed!
Time: 2.238137 sec
Nodes: 23

DCOQUETRY
RLBOATING
IASVGLOBE
VMTESOALB
EPINYCRUX
LDNBMKPIS
SOAIBJUDP
UWCROANEI
BNYDLWKAT
```



```
V , 0 , 1 : CLAMPDOWN
V , 0 , 2 : OBSTINACY
V , 1 , 3 : OVENBIRD
H , 0 , 1 : COQUETRY
H , 1 , 2 : BOATING
V , 0 , 0 : DRIVELS
V , 2 , 5 : LOCKJAW
V , 3 , 4 : SYMBOL
H , 2 , 4 : GLOBE
H , 7 , 3 : ROAN
V , 5 , 7 : IDEA
V , 5 , 8 : SPIT
V , 5 , 6 : PUNK
H , 4 , 5 : CRUX
V , 2 , 6 : OAR
H , 4 , 1 : PIN
H , 3 , 6 : ALB
H , 5 , 6 : PIS
V , 6 , 0 : SUB
```

Game of Breakthrough

Introduction

Breakthrough is a strategy game, similar to chess, but played with pawns that can move forward and diagonally. In this part of the MP, we simulate this 2 player zero-sum game using two different types of searches and different heuristics.

Implementation

We initialized the board to be a 2D matrix of size 8x8, with empty spaces filled with the character '_'. We then create two separate dictionaries 'white' and 'black'. These dictionaries represent the positions of each white and black piece. They begin with 16 keys corresponding to values that are the piece's location on the board. Once we are done initializing these two dictionaries we simply assign those locations as 'W' and 'B' respectively in the 2D array.

Now we setup the end_case() function which basically checks if any of the black or white pieces have reached the other end or if there are any pieces left for the game to continue. If there are none or if it has hit the end then we simply state that the game is over.

An important function was to find out the next possible action for any/all given pieces in the board at a given state. This was done in possible_actions() function. It returns 2 values. One of them the set of all possible actions a given piece can take (i.e. 'Left', 'Up/Down', 'Right) for every single (either B/W) piece. So it is basically a tuple which is the coordinates inside another tuple (something like (w,(x,y)) where w is the key for each of the pieces). The other thing that it returns is 'possible_capture' which is basically an 'int' and it checks for any nearby enemy pieces and increments it if there is one. This is used further in our evaluation function and heuristic.

Now comes our 'goal_score()' and 'score_o()' and 'score_d()' which tells us a certain value that can be used for calculations in the evaluation function. The goal_score() function basically calls possible_actions() and finds out if the end goal (i.e. the finish line) is in sight for any of the given nodes. If it is then it returns a boolean 'T'. And if it's true it adds a certain constant 'value' to our scoring system. score_o() is the scoring system used by our offensive evaluation function and heuristic function. Similarly score_d() is for defensive.

Evaluation Function

Now our evaluation function (i.e. score calculation of our state vs the enemy) basically takes the sum of the number of surviving pieces, the 'value' obtained from the goal_state(), checks and adds height of how many of the pieces have crossed the half line, as well as the maximum height reached by a friendly piece and an opponent piece. To these we then add the ‘possible_capture’ values and we give higher weight to the offensive scoring than the defensive one, so the offensive one is more greedy.

Heuristics

For our heuristic we simply took the results from the evaluation function/scoring function and weighted them such that our score is weighted higher than the enemy score and we subtracted them. This was for the offensive heuristic. Now, for defensive heuristic we weight them in the opposite way.

We observed that when we weight the possible_capture results in the scoring/evaluation function, a simple change drastically affects the behaviour of the gameplay. (i.e. if we weight one higher relative to the other then the play time increases as well as the number of nodes). They tend to play safer.

Minimax and Alpha-Beta Searches

Our minimax search is evaluated to a depth of 3. The player using minimax evaluates all of the possible different actions he can take, represented by board states, and uses a minimax search to choose his next action.

Our alpha-beta search is also evaluated to a depth of 3, and is similar to the minimax search except for that alpha-beta pruning takes place in order to avoid unnecessarily expanding nodes in branches that are known to be less than optimal. This search allows us to save on computation time.

Results

We find that the search depth does not have as great of an impact on the outcome of the game. However, this observation may be limited because of the narrow range of practical search depths (from one to three), and because even a depth of three is not enough to sufficiently foresee all future possibilities.

The type of evaluation function used greatly impacts the outcome, as even a small tweak or change in the heuristics and parameters produces a drastically different outcome. This may be due to the high sensitivity of these values in the minimax and alpha-beta search algorithms, as the smaller evaluation function values tend to have a more direct impact on the actions taken as opposed to the larger evaluation function values. Choosing the same evaluation function for both players generally led to longer games, with more nodes expanded.

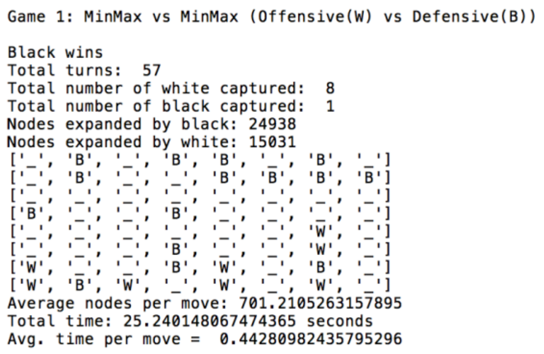
For players of equal strength, an advantage can be given by going first, and also by using a defensive evaluation function. However, we find that for our implementation, going first is a greater indication for the outcome of the game, possibly because of the difficulty of tuning the heuristics for the evaluation function appropriately.

Presented below are the 16 different simulations we ran using two different search strategies, as well as each with own offensive and defensive heuristic functions.

Each game simulation shows:

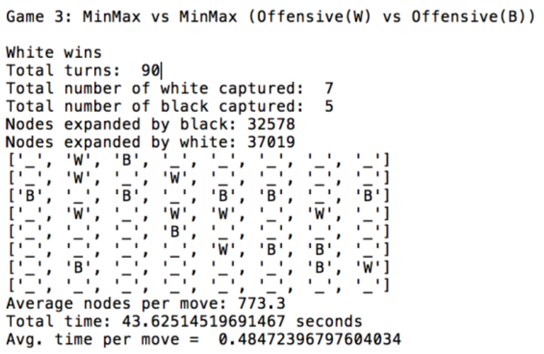
1. Final state of the board and the winning player
2. Total number of nodes expanded by each player in the game
3. Average number of nodes per move and average time per move in seconds
4. The number of workers captured by each players and total number of moves.

As a note, the first player listed is always white, and the second player listed is always black.



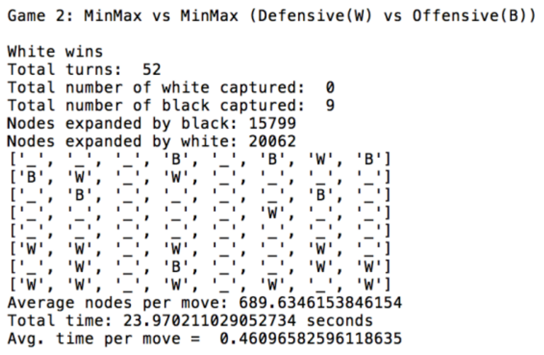
```
Game 1: MinMax vs MinMax (Offensive(W) vs Defensive(B))

Black wins
Total turns:  57
Total number of white captured:  8
Total number of black captured:  1
Nodes expanded by black: 24938
Nodes expanded by white: 15031
['_', 'B', '_', 'B', 'B', '_', 'B', '_']
['_', 'B', '_', '_', 'B', 'B', 'B', 'B']
['_', '_', '_', '_', '_', '_', '_', '_']
['B', '_', '_', 'B', '_', '_', '_', '_']
['_', '_', '_', '_', '_', '_', 'W', '_']
['_', '_', '_', 'B', '_', '_', 'W', '_']
['W', '_', '_', 'B', 'W', '_', 'B', '_']
['W', 'B', 'W', '_', 'W', '_', 'W', '_']
Average nodes per move: 701.2105263157895
Total time: 25.240148067474365 seconds
Avg. time per move =  0.44280982435795296
```



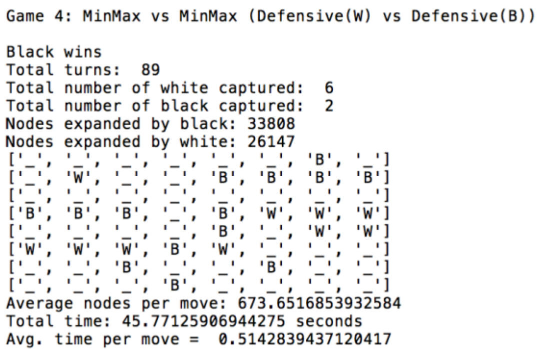
```
Game 3: MinMax vs MinMax (Offensive(W) vs Offensive(B))

White wins
Total turns:  90
Total number of white captured:  7
Total number of black captured:  5
Nodes expanded by black: 32578
Nodes expanded by white: 37019
['_', 'W', 'B', '_', '_', '_', '_', '_']
['_', 'W', '_', 'W', '_', '_', '_', '_']
['B', '_', 'B', '_', 'B', 'B', '_', 'B']
['_', 'W', '_', 'W', 'W', '_', 'W', '_']
['_', '_', '_', 'B', '_', '_', '_', '_']
['_', '_', '_', '_', 'W', 'B', 'B', '_']
['_', 'B', '_', '_', '_', '_', 'B', 'W']
['_', '_', '_', '_', '_', '_', '_', '_']
Average nodes per move: 773.3
Total time: 43.62514519691467 seconds
Avg. time per move =  0.48472396797604034
```



```
Game 2: MinMax vs MinMax (Defensive(W) vs Offensive(B))

White wins
Total turns:  52
Total number of white captured:  0
Total number of black captured:  9
Nodes expanded by black: 15799
Nodes expanded by white: 20062
['_', '_', '_', 'B', '_', 'B', 'W', 'B']
['B', 'W', '_', 'W', '_', '_', '_', '_']
['_', 'B', '_', '_', '_', '_', 'B', '_']
['_', '_', '_', '_', '_', 'W', '_', '_']
['_', '_', '_', '_', '_', '_', '_', '_']
['W', 'W', '_', 'W', '_', '_', 'W', '_']
['_', 'W', '_', 'B', '_', '_', 'W', 'W']
['W', 'W', '_', 'W', '_', 'W', '_', 'W']
Average nodes per move: 689.6346153846154
Total time: 23.970211029052734 seconds
Avg. time per move =  0.46096582596118635
```



```
Game 4: MinMax vs MinMax (Defensive(W) vs Defensive(B))

Black wins
Total turns:  89
Total number of white captured:  6
Total number of black captured:  2
Nodes expanded by black: 33808
Nodes expanded by white: 26147
['_', '_', '_', '_', '_', '_', 'B', '_']
['_', 'W', '_', '_', 'B', 'B', 'B', 'B']
['_', '_', '_', '_', '_', '_', '_', '_']
['B', 'B', 'B', '_', 'B', 'W', 'W', 'W']
['_', '_', '_', '_', 'B', '_', 'W', 'W']
['W', 'W', 'W', 'B', 'W', '_', '_', '_']
['_', '_', 'B', '_', '_', 'B', '_', '_']
['_', '_', '_', 'B', '_', '_', '_', '_']
Average nodes per move: 673.6516853932584
Total time: 45.77125906944275 seconds
Avg. time per move =  0.5142839437120417
```

```
Game 5: AlphaBeta vs AlphaBeta (Offensive(W) vs Defensive(B))

White wins
Total turns:  96
Total number of white captured:  11
Total number of black captured:  6
Nodes expanded by black: 29598
Nodes expanded by white: 26615
['W', '_', '_', '_', '_', '_', '_', '_']
['_', '_', '_', '_', '_', '_', 'B', 'B']
['_', '_', 'B', 'B', '_', 'B', '_', '_']
['_', '_', '_', '_', '_', '_', '_', '_']
['B', '_', '_', '_', 'B', '_', '_', '_']
['_', 'B', '_', '_', '_', '_', '_', '_']
['B', 'B', '_', '_', '_', '_', '_', '_']
['_', '_', '_', '_', 'W', 'W', 'W', 'W']
Average nodes per move: 585.5520833333334
Total time: 37.10763597488403 seconds
Avg. time per move =  0.3865382596850395
```

```
Game 6: AlphaBeta vs AlphaBeta (Defensive(W) vs Offensive(B))

White wins
Total turns:  54
Total number of white captured:  1
Total number of black captured:  5
Nodes expanded by black: 14784
Nodes expanded by white: 20880
['_', 'W', '_', '_', '_', 'B', 'B', 'B']
['_', 'W', '_', 'W', 'B', '_', 'B', 'B']
['_', '_', 'W', 'B', 'B', 'B', '_', '_']
['_', 'B', '_', '_', 'W', '_', '_', '_']
['_', '_', '_', '_', 'W', '_', '_', '_']
['_', '_', '_', '_', '_', '_', '_', '_']
['W', 'B', '_', '_', '_', '_', 'W', 'W']
['_', 'W', '_', 'W', 'W', 'W', 'W', 'W']
Average nodes per move: 660.4444444444445
Total time: 26.26862597465515 seconds
Avg. time per move =  0.48645631472269696
```

```
Game 7: AlphaBeta vs AlphaBeta (Offensive(W) vs Offensive(B))

White wins
Total turns:  54
Total number of white captured:  3
Total number of black captured:  4
Nodes expanded by black: 17379
Nodes expanded by white: 21430
['_', 'W', '_', '_', 'B', 'B', 'B', 'B']
['W', '_', '_', '_', 'B', 'B', 'W', 'B']
['_', '_', '_', '_', '_', '_', '_', '_']
['_', '_', '_', '_', 'B', '_', '_', '_']
['_', 'B', '_', 'B', '_', '_', '_', '_']
['_', '_', 'W', '_', '_', '_', '_', '_']
['B', '_', 'B', '_', '_', '_', 'W', 'W']
['_', 'W', 'W', 'W', 'W', 'W', 'W', 'W']
Average nodes per move: 718.6851851851852
Total time: 28.50815510749817 seconds
Avg. time per move =  0.5279290013843112
```

```
Game 8: AlphaBeta vs AlphaBeta (Defensive(W) vs Defensive(B))

Black wins
Total turns:  57
Total number of white captured:  0
Total number of black captured:  0
Nodes expanded by black: 17312
Nodes expanded by white: 29459
['_', '_', '_', 'B', 'B', '_', '_', 'B']
['_', '_', 'B', 'B', 'B', 'B', 'B', 'B']
['B', '_', '_', '_', '_', '_', 'B', '_']
['W', 'B', 'W', 'W', 'W', '_', 'W', '_']
['_', 'W', '_', '_', 'W', '_', 'W', 'B']
['_', 'W', 'W', '_', '_', '_', '_', 'W']
['B', '_', '_', 'B', 'W', '_', '_', '_']
['_', 'B', '_', 'W', '_', 'W', 'W', 'W']
Average nodes per move: 820.5438596491229
Total time: 37.66683387756348 seconds
Avg. time per move =  0.6608218561139023
```

```
Game 9: MinMax vs AlphaBeta (Offensive(W) vs Defensive(B))

White wins
Total turns:  68
Total number of white captured:  5
Total number of black captured:  3
Nodes expanded by black: 19544
Nodes expanded by white: 20180
['_', '_', '_', 'W', '_', '_', '_', 'B']
['_', '_', 'B', '_', 'B', 'B', 'B', 'B']
['_', 'B', '_', '_', 'B', '_', '_', '_']
['B', '_', 'B', '_', '_', '_', '_', '_']
['_', '_', '_', '_', '_', '_', '_', '_']
['B', '_', '_', '_', '_', '_', 'W', '_']
['W', 'B', 'B', 'W', 'W', 'W', '_', 'W']
['W', '_', '_', '_', 'W', 'W', '_', 'W']
Average nodes per move: 584.1764705882352
Total time: 27.796682834625244 seconds
Avg. time per move =  0.4087749404065749
```

```
Game 10: MinMax vs AlphaBeta (Defensive(W) vs Offensive(B))

White wins
Total turns:  52
Total number of white captured:  1
Total number of black captured:  7
Nodes expanded by black: 11197
Nodes expanded by white: 24575
['_', '_', 'W', 'B', 'B', 'B', 'B', 'B']
['W', 'W', '_', '_', 'B', '_', 'B', 'B']
['_', '_', '_', '_', '_', '_', '_', '_']
['_', '_', '_', '_', '_', '_', '_', '_']
['W', 'W', '_', 'W', '_', '_', 'B', '_']
['_', '_', '_', '_', '_', '_', '_', '_']
['_', '_', '_', 'W', 'W', '_', 'W', 'W']
['_', 'W', '_', '_', 'W', 'W', 'W', 'W']
Average nodes per move: 687.9230769230769
Total time: 25.299731969833374 seconds
Avg. time per move =  0.4865335409457867
```

```
Game 11: MinMax vs AlphaBeta (Offensive(W) vs Offensive(B))

Black wins
Total turns:  87
Total number of white captured:  2
Total number of black captured:  3
Nodes expanded by black: 32655
Nodes expanded by white: 39867
['_', '_', '_', '_', '_', '_', '_', '_']
['_', 'W', '_', 'W', 'B', '_', '_', '_']
['_', '_', '_', 'W', 'B', 'B', 'B', 'B']
['_', 'B', '_', 'W', '_', 'W', 'W', 'W']
['W', 'B', '_', 'B', 'B', 'B', '_', '_']
['W', '_', 'W', '_', 'W', 'W', 'W', '_']
['B', 'B', '_', 'W', '_', '_', '_', '_']
['_', '_', 'B', '_', '_', '_', '_', '_']
Average nodes per move: 833.5862068965517
Total time: 50.63875985145569 seconds
Avg. time per move =  0.5820548726224352
```

```
Game 12: MinMax vs AlphaBeta (Defensive(W) vs Defensive(B))

Black wins
Total turns:  79
Total number of white captured:  3
Total number of black captured:  5
Nodes expanded by black: 19853
Nodes expanded by white: 29728
['_', '_', '_', '_', '_', '_', 'B', 'B']
['W', 'W', '_', '_', 'B', 'B', 'B', 'B']
['B', '_', '_', '_', '_', '_', '_', '_']
['_', '_', 'W', '_', 'B', 'W', 'W', '_']
['W', '_', '_', '_', 'W', '_', '_', '_']
['_', '_', '_', 'W', 'W', '_', 'W', '_']
['_', '_', 'B', 'B', '_', 'W', '_', 'W']
['_', 'B', '_', '_', '_', '_', '_', 'W']
Average nodes per move: 627.6075949367089
Total time: 35.90165901184082 seconds
Avg. time per move =  0.45445153079455414
```

```
Game 13: AlphaBeta vs MinMax (Offensive(W) vs Defensive(B))

White wins
Total turns:  12
Total number of white captured:  0
Total number of black captured:  2
Nodes expanded by black: 3146
Nodes expanded by white: 3642
['W', 'B', '_', 'B', 'B', 'B', 'B', 'B']
['_', '_', 'B', 'B', 'B', 'B', 'B', 'B']
['B', '_', '_', '_', '_', '_', '_', '_']
['_', '_', '_', 'B', '_', '_', '_', '_']
['_', '_', '_', '_', '_', '_', '_', '_']
['_', '_', '_', '_', '_', '_', '_', '_']
['_', 'W', 'W', 'W', 'W', 'W', 'W', 'W']
['W', 'W', 'W', 'W', 'W', 'W', 'W', 'W']
Average nodes per move: 565.6666666666666
Total time: 5.349469900131226 seconds
Avg. time per move =  0.4457903305689494
```

```
Game 14: AlphaBeta vs MinMax (Defensive(W) vs Offensive(B))

White wins
Total turns:  56
Total number of white captured:  2
Total number of black captured:  3
Nodes expanded by black: 18582
Nodes expanded by white: 20733
['B', 'W', 'B', 'B', 'B', 'B', 'B', 'B']
['_', 'W', 'W', '_', '_', '_', 'B', '_']
['_', '_', '_', '_', 'B', '_', '_', '_']
['_', '_', '_', 'W', '_', 'B', '_', 'B']
['_', '_', '_', '_', 'W', '_', '_', '_']
['W', 'W', '_', '_', '_', '_', '_', '_']
['_', '_', 'B', 'B', 'W', 'W', 'W', 'W']
['_', '_', '_', '_', '_', 'W', 'W', 'W']
Average nodes per move: 702.0535714285714
Total time: 27.41042995452881 seconds
Avg. time per move =  0.4894721806049347
```

```
Game 15: AlphaBeta vs MinMax (Offensive(W) vs Offensive(B))

Black wins
Total turns:  59
Total number of white captured:  4
Total number of black captured:  4
Nodes expanded by black: 23403
Nodes expanded by white: 24577
['B', 'B', 'B', '_', '_', '_', 'B', '_']
['_', 'W', '_', '_', '_', 'W', '_', '_']
['B', '_', 'B', '_', 'B', '_', 'B', '_']
['_', 'W', '_', 'W', '_', 'W', 'B', '_']
['_', '_', '_', '_', '_', 'W', '_', '_']
['_', '_', '_', '_', 'W', '_', '_', '_']
['_', 'B', '_', '_', '_', '_', 'B', '_']
['W', '_', 'W', 'B', '_', 'W', 'W', 'W']
Average nodes per move: 813.2203389830509
Total time: 33.13905906677246 seconds
Avg. time per move =  0.5616791692830748
```

```
Game 16: AlphaBeta vs MinMax (Defensive(W) vs Defensive(B))

White wins
Total turns:  78
Total number of white captured:  2
Total number of black captured:  0
Nodes expanded by black: 25180
Nodes expanded by white: 26668
['_', 'W', '_', '_', '_', '_', '_', '_']
['W', '_', 'W', '_', 'B', 'B', '_', '_']
['_', '_', '_', 'B', 'B', 'B', 'B', 'B']
['B', 'B', 'B', 'B', 'B', '_', 'B', '_']
['_', '_', '_', '_', '_', '_', 'W', 'B']
['_', '_', 'W', 'W', 'W', 'W', '_', 'W']
['B', 'B', '_', 'W', '_', 'W', '_', 'W']
['_', '_', '_', '_', '_', '_', 'W', 'W']
Average nodes per move: 664.7179487179487
Total time: 38.036494970321655 seconds
Avg. time per move =  0.4876475242468027
```