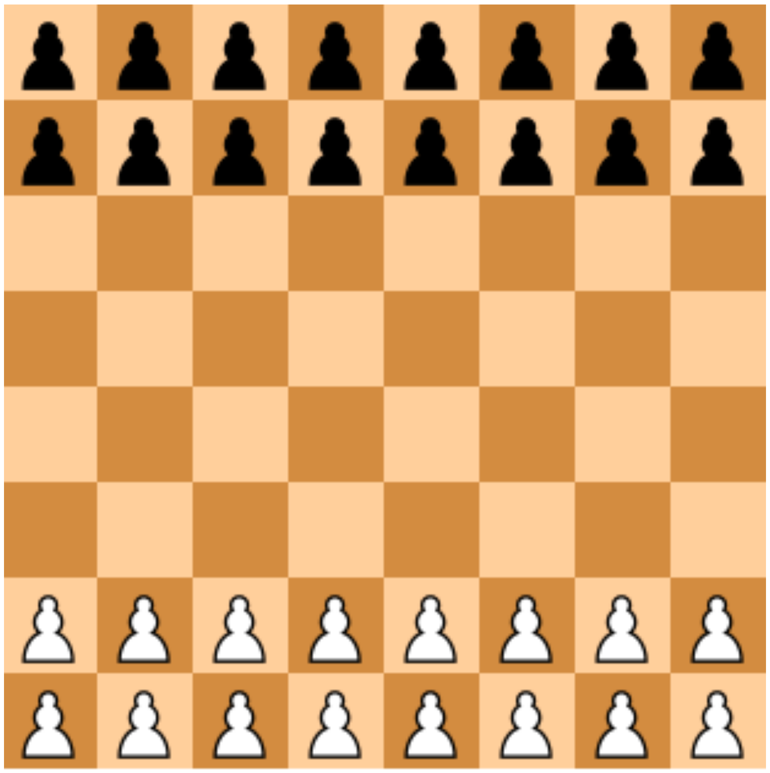

ASSIGNMENT 2: CONSTRAINT SATISFACTION PROBLEMS AND GAMES – 4 CREDIT

Word Sudoku and Game of Breakthrough

ABSTRACT

In the first part of this assignment, the Word Sudoku is solved as a constraint satisfaction problem through backtracking search. In the second part of this assignment, the Game of Breakthrough is simulated between agents of minimax and/or alpha-beta pruning search strategy with defensive or offensive evaluation functions.

Qixin Zhu

CS 440 / ECE 448

Table of Contents

1- Word Sudoku CSP

1.1 CSP Formulation

1.1.1 Variables

Each word in the word-bank is a variable.

1.1.2 Domains

For each variable, the domain is the location of that word, including both the row & column of the word's starting letter and the direction (horizontal or vertical) of the word.
D(word) = {$H_{i,j}$, $V_{i,j}$} in which H or V denotes the direction, i (0~8) denotes the row, j (0~8) denotes the column of the first letter in the word.

1.1.3 Constraints

The constraints consist the 4 follow parts:

(1) For each row, there is no duplicate letters.
(2) For each column, there is no duplicate letters.
(3) For each 3X3 sub-grid, there is no duplicate letters.
(4) The newly placed word does not violate the existing letters in the Sudoku board.
(5) Each word must be used exactly once.

1.2 Backtrack Search Implementation

1.2.1 Most Constrained Variable

Variables are stored in a priority queue so that the most constrained variable, aka **the word with least possible locations**, will be explored first. Before the next variable is chosen, all variables are recalculated based on the current Sudoku board to evaluate how many feasible locations there are, without violating any above mentioned constraints. Typically, the longer words will have smaller feasible domains and will be explored earlier.

1.2.2 Least Constraining Value

For a given variable, all possible values, aka feasible location assignments, are stored in a priority queue so that the least constraining value, aka the location that imposes least constraints to all remaining variables, will be explored first. After a certain variable is chosen, all feasible successor states have already been evaluated in the previous step, and the successor states are ordered according to the amount of cells that have been completed. Typically, **a word's location that overlaps with most number of existing cells** in Sudoku board and completes least number of new cells is the least constraining value.

1.2.3 Pruning Rule: Forward Checking

The 5th constraint mention in section 1.1.3 is used as the pruning rule to speed up the backtracking search. (This rule will have to be modified for 4th credit problem, to be discussed later in section 1.5.) Whenever a variable has zero choices in its feasible domain, all branches beyond that need not to be explored, and the search will immediately return the conclusion that this branch has failed. As noticed through running the program, **this pruning process is very efficient**.

1.3-Word Sudoku Solutions

1.3.1 Input 1: Start Grid with Hints

The number of nodes expanded is **19**, the execution time is **152ms**.

Original grid:

```
_ _ G _ _ _ _ _ _
_ _ _ _ _ _ _ _ _
_ _ P _ _ _ _ _ _
_ _ _ _ _ _ _ _ _
_ R _ _ _ _ _ _ _
_ _ _ _ S _ _ L _
N _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _
```

Finished grid:

```
L I G H T E N M P
C O N F U S E A Y
S U P W I N D R T
E T U N D R A V H
M R F I C K Y E O
I A O M S H P L N
N G L B A U O I E
A E K L V M U N C
R D S Y E P T G K
```

Sequence of assignments:

V,0,7: MARVELING
V,1,1: OUTRAGED
H,0,0: LIGHTEN
V,0,8: PYTHON
V,2,0: SEMINAR
H,3,1: TUNDRA
H,2,1: UPWIND
H,1,0: CONFUSE
V,3,3: NIMBLY
V,4,8: ONE
V,2,6: DAY
H,4,3: ICKY
V,4,2: FOLKS
(N/A): SEA
V,5,4: SAVE
V,5,5: HUMP
V,5,8: NECK
(N/A): SUP
V,5,6: POUT

1.3.2 Input 2: Empty Start Grid

The number of nodes expanded is **98**, the execution time is **1240ms**.

Original grid:

```
_ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _
```

Finished grid:

```
D C O Q U E T R Y
R L B O A T I N G
I A S V G L O B E
V M T E S O A L B
E P I N Y C R U X
L D N B M K P I S
S O A I B J U D P
U W C R O A N E I
B N Y D L W K A T
```

Sequence of assignments:

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

1.3.3 Input 3: Decoy Words

The number of nodes expanded is **108550**, the execution time is **266421ms**.

Original grid:

```
_ _ _ _ _ _ T _ _
_ _ _ T _ _ _ _ _
_ _ _ _ _ _ _ O _
_ _ _ _ _ _ _ _ T
_ _ _ _ _ _ _ _ _
_ T _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _
_ _ _ _ _ T _ _ _
_ _ _ _ _ _ _ T _
```

Finished grid:

```
S O R I E N T A L
N P A T C H I E R
I E G R Y P H O N
C R O S D W E L T
K A N U B G R I M
E T I L R J A G S
R I Z K A E B U C
A N E W I T L S O
M G D O N S Y T W
```

Sequence of assignments:

(N/A): ENLARGE
V,0,1: OPERATING
H,1,1: PATCHIER
H,0,1: ORIENTAL
(N/A): STRIDE
V,1,2: AGONIZED
H,2,2: GRYPHON
V,0,0: SNICKER
V,4,4: BRAIN
(N/A): AMBUSH
H,4,5: GRIM
H,3,4: DWELT
H,7,0: ANEW
V,5,5: JETS
(N/A): JAGS
V,5,7: GUST
V,5,6: ABLY
H,8,2: DON
V,3,3: SULK
V,5,8: SCOW
(N/A): WIT
V,6,0: RAM

As shown above in the sequence of assignments, 5 words are excluded from the Sudoku board, so **the fewest words solution has 17 words**.

1.4 Modifications for Decoy Words

The previous forward checking pruning rule in section 1.6 does not work for decoy words problem. In this problem, to ensure the fewest words solution can be found, the backtracking search algorithm **will not return immediately when finding first solution**. Instead, the search goes on to find all possible solutions, save them in the order of number of words used. Eventually, the fewest words solution, which has maximum number of "N/A" in its solution path, is selected from all feasible solutions.

Here, the pruning rule in section 1.6 is replaced by **checking the number of letters for remaining words**. If the number of letters is not enough to fill the remaining cells in the Sudoku board, it means too many words has been discarded due to zero feasible domain. Then the algorithm will stop exploring current branch and backtrack.

Another important character to facilitate the finding all solutions is that **the ordering of putting words into the solution does not matter**, only the words' location matters. With this thoughts, the N factorial possible permutations of a solution only need to be investigate once with a certain chosen order, where N is the number of words in the solution. The only thing to be enumerated here is the domain values of each variable, aka the feasible locations of each word. **Overall, the computational cost is in a reasonable and acceptable range.**

1.5 Tailored Algorithm (Bonus Points ?)

After enumerating all possible solutions, it can be observed that for this particular Sudoku board, the solution is unique – no matter how few words are used, the solution board always results in the same one. Using this property, **a faster algorithm can be designed** to solve such problem **with much fewer number of expanded nodes** and less time. However, the solution uniqueness of a Sudoku board is subject to the game designer, and cannot be known by the player before enumerating all solutions. Hence, **the limitation of this algorithm is that it only works for Sudoku with a unique solution board.** Fortunately, according to reference [6], a "proper puzzle" have a unique solution, so this algorithm will work for most of the game.

The algorithm's pseudocode, together with explanations, is given as the following:

```
// Part 1: find one valid solution, in a faster way
use the original backtracking search algorithm, with the same pruning rule in section 1.1.6, to
find a solution game board;
while (true)
    if solution is null (not found)
        increase the capacity of discard-set by 1;        // allow 1 more word to be unassigned
    else   // a valid solution is found
        reset the capacity of discard-set to 0;    // back to 1.1.6 pruning rule
        remove all words in discard-set from the word bank;
        remove all words in the assignment sequence with "N/A" from the word bank;
        if no words can be removed from the word bank, break while loop;
```

// At this point, a valid solution is found, words that cannot fit in Sudoku board and that are apparently redundant has been removed from the word bank, but this solution is not guaranteed to have fewest words

// **[Correctness Illustration]** Since the solution is unique, if there exists a solution with fewer words, the solutions board will still be the same. So the only possibility is: some words can be removed from word bank without leaving a blank in the solution board, aka, the valid solution can still be found.

```
// Part 2: try removing words one by one, until no valid solution can be found
put the filtered word bank in a queue Q;
while (Q is not empty)
    poll a word bank WB from Q;
    for each word W in WB       // try removing words one at a time
        remove W from WB, get a new word bank newWB;
        use the original backtracking algorithm to find a solution corresponding to newWB;
        if a solution exists, add newWB to Q for further reduction trial;
return the solution (has fewest words) that was last added to Q;
```

The result from this tailored algorithm is shown as the following:

The number of nodes expanded is **2010**, the execution time is **5610ms**.

Original grid:

```
_ _ _ _ _ _ T _ _
_ _ _ T _ _ _ _ _
_ _ _ _ _ _ _ O _
_ _ _ _ _ _ _ _ T
_ _ _ _ _ _ _ _ _
_ T _ _ _ _ _ _ _
_ _ _ _ _ _ _ _ _
_ _ _ _ _ T _ _ _
_ _ _ _ _ _ _ T _
```

Finished grid:

```
S O R I E N T A L
N P A T C H I E R
I E G R Y P H O N
C R O S D W E L T
K A N U B G R I M
E T I L R J A G S
R I Z K A E B U C
A N E W I T L S O
M G D O N S Y T W
```

Sequence of assignments:

```
V,0,1: OPERATING
H,1,1: PATCHIER
H,0,1: ORIENTAL
V,1,2: AGONIZED
H,2,2: GRYPHON
V,0,0: SNICKER
V,4,4: BRAIN
H,3,4: DWELT
H,4,5: GRIM
V,5,5: JETS
H,7,0: ANEW
V,5,7: GUST
V,5,6: ABLY
V,3,3: SULK
V,6,0: RAM
H,8,2: DON
V,5,8: SCOW
(N/A): STRIDE
(N/A): JAGS
(N/A): AMBUSH
(N/A): WIT
(N/A): ENLARGE
```

2-Minimax and Alpha-beta Agents

2.1 Evaluation Function: Offensive VS. Defensive

At the very basic, the evaluation function should consider two factors: (1) Evaluate **the number of remaining workers**. The more workers remained for the player and the less workers remained for opponent, the better situation it is for the current player. (2) Evaluate **the distance to opponent's base**. The closer the furthest worker to opponent's base and the further the closest opponent's worker to player's base, the better situation it is for the current player. This rule will later be adjusted for the "3-worker-to-enemy-base win criterion".

As implied by the name, offensive evaluation function encourages moving forward and capturing opponent's workers. To do so, it gives more weight for reducing the number of opponent's remaining workers and for moving toward opponent's base. In contrast, defensive evaluation function focuses more on preventing losing workers and stopping opponent from getting close to defensive agent's base. To sum up, the evaluation functions are illustrated as below.

Let **n** be the **number** of remaining workers for the player, **m** be the **number** of remaining workers for the opponent. Let $\mathbf{d_i}$ be the **distance** from the player's i^{th} worker to the opponent's base, $\mathbf{d_i}'$ be the **distance** from the opponent's i^{th} worker to the player's base.

Additionally, let $\mathbf{f_i}$ be the **distance** from the player's **furthest** i^{th} worker to the opponent's base, $\mathbf{f_i}'$ be the **distance** from the opponent's **furthest** i^{th} worker to the player's base. Let **k** be the **number** of furthest workers counted in the current game, **k = 1** in the original rule and **k = 3** in the case of "3-worker-to-enemy-base win criterion".

$$Offensive = (n - O_{Factor} * m) + [O_{Factor} * (\sum_{i=0}^{n} \frac{1}{d_i} + \sum_{i=0}^{k} \frac{1}{f_i}) - (\sum_{i=0}^{n} \frac{1}{d_i'} + \sum_{i=0}^{k} \frac{1}{f_i'})]$$

$$Defensive = (D_{Factor} * n - m) + [(\sum_{i=0}^{n} \frac{1}{d_i} + \sum_{i=0}^{k} \frac{1}{f_i}) - D_{Factor} * (\sum_{i=0}^{n} \frac{1}{d_i'} + \sum_{i=0}^{k} \frac{1}{f_i'})]$$

In which O_{Factor} and D_{Factor} are taken to be **1.5** (user specified based on trials). The above formulas mean that an offensive evaluation function puts more weights on reducing the opponent's worker (through capturing) and advancing toward opponent's base than on maintaining the player's own workers and preventing opponent's moving forward. The defensive evaluation function does the opposite.

It is worth noting that, by taking the **reciprocal of $\mathbf{d_i}$**, the evaluation function naturally gives more weight on the top most worker. The extra 1 or 3 terms for distance measurements are included to encourage players moving forward when there is a chance. It is also easier to **adjust the evaluation function for different rules** in section 2.5.

2.2 Search Depth

The following table shows the approximate number of expanded game tree nodes per move for minimax agent and alpha-beta agent.

Number of expanded game tree nodes per move		
Search Depth	Alpha-beta Agent	Minimax Agent
3	3k	15k
4	30k	360k
5	200k	~7500k
6	2500k	N/A

(1) Alpha-beta agent expands less than 1/5 of nodes than Minimax agent with the same search depth;
(2) Minimax agent's number of expanded nodes goes up more quickly with the increase of search depth than Alpha-beta agent.
(3) To finish the game simulation in a decent amount of time, **the feasible search depth for Minimax agent is 4 and for Alpha-beta agent is 6**.

2.3 Game Quality Improvements (Bonus Points)

For **alpha-beta pruning** algorithm, the evaluating order matters and will affect the amount of pruning significantly. To maximize the pruning, in each evaluation round, **the possible moves of the top most worker is evaluated first**, which gives more potentially good behavior than the last row of workers staying in the player's base. **Before implementing this ordering strategy**, the program simply goes through the game board row by row from small to large, which is naturally the good order for one player and bad order for the other. It is noticed that the number of nodes expanded for **the two players differs in about 5 times!**

An important character of **alpha-beta pruning** is that, though not affect results of the evaluated state, the results of sub-states that are pruned may be incorrect, thus may be misleading when there are multiple sub-states of the same value. The algorithm will always pick the first "best" move, but there may be equivalent good moves that have been pruned for being "no better than the existing best move". To avoid this subtle imperfection, **this alpha-beta agent evaluates all next moves** and start pruning in the second step, without pruning any potential equivalently good moves. However, the **tradeoff is the speed** – this is taking more time and expanding more nodes than the "neglecting equivalently good moves" strategy. It could be the explanation why the current implementation is able to work with depth 4 for Minimax but is not able to double the search depth to 8 for Alpha-beta.

So far, the alpha-beta agent saves all equivalent good moves, so does the minimax agent by its natural. **The tie breaking strategy for both agent is random**. A list of equivalently good moves is saved during one search round, then a random move is chosen from them as the agent's decision. This tie breaking strategy will result in different final state for each run of the program, and gives some interesting results, such as a player's workers are all captured or a player's only two remaining workers arrive the opponent's base and has no valid moves at all.

2.4 Game Matchup Results

For each matchup required, results of the following 4 games are shown:

(1) Game 1: Offensive (Player X) vs Defensive (Player O);
(2) Game 2: Defensive (Player X) vs Offensive (Player O);
(3) Game 3: Offensive (Player X) vs Offensive (Player O);
(4) Game 4: Defensive (Player X) vs Defensive (Player O).

Due to the random tie breaking strategy describe in section 2.3, the results are different for each run. To be able to duplicate results, random seeds are set in the program. To evaluate the effect of playing order and evaluation function, multiple simulations may be run and saved, but only one representative result is shown in the following sections.

The **search depth** used in the following results is 4 for minimax agent and 6 for alpha-beta agent. For implementation reason, **the player that takes the first move is named as player X**.

***Discussion**: TA's demo in Oct-20's lecture shows that offensive player tends to form a worker wall 3 steps away from the opponent's base, which is **not observed in my simulations**. I believe this is due to using the linear distance in evaluation function, instead the reciprocal (see section 2.1). Linear distance measure gives the same reward for moving a worker from row 3 to row 4 as for moving a work from row 4 to row 5. In contrast, reciprocal distance measure favors the move for the further worker as illustrated in the simple math below (opponent's base is at row 7):

$$\frac{1}{(7-5)} - \frac{1}{(7-4)} = \frac{1}{6} > \frac{1}{12} = \frac{1}{(7-4)} - \frac{1}{(7-3)}$$

2.4.1 Minimax vs. Minimax

Game 1		Game 2		Game 3		Game 4	
Winner: **Player O**		Winner: **Player O**		Winner: **Player O**		Winner: **Player X**	
Total Moves: 34		Total Moves: 33		Total Moves: 31		Total Moves: 28	
Player X	Player O	Player X	Player O	Player X	Player O	Player X	Player O
Captured Workers		Captured Workers		Captured Workers		Captured Workers	
7	5	5	7	7	7	4	5
# of Nodes Expanded		# of Nodes Expanded		# of Nodes Expanded		# of Nodes Expanded	
16195028	15860597	17861077	17625546	14923091	15189478	20100007	19567350
# Nodes per Move		# Nodes per Move		# Nodes per Move		# Nodes per Move	
476324	466488	541244	534107	481390	489983	717857	724716
time (ms) per move		time (ms) per move		time (ms) per move		time (ms) per move	
386	374	403	403	343	354	531	517

```
0 X   X       X | X     0 X     X | X 0 X X   X X   | X         X X X
X         X   0 |   X       X   X | X               |   X   X X
        X     X |             0 X |         X       |   X 0     0
  X   X   X     |   X       0     |             0   |     X         0
        X       |         X       | X         X     |   0           X
  0   0 0       |   0   0         |       0   0   0 |
0       0     0 | 0 0           0 |           0     | 0       0   0
              0 |     0   0     0 | 0 0 0           | 0 0 0 0 X     0
```

2.4.2 Alpha-beta vs. Alpha-beta

Game 1		Game 2		Game 3		Game 4	
Winner: **Player X**		Winner: **Player O**		Winner: **Player X**		Winner: **Player X**	
Total Moves: 40		Total Moves: 37		Total Moves: 35		Total Moves: 42	
Player X	Player O	Player X	Player O	Player X	Player O	Player X	Player O
Captured Workers		Captured Workers		Captured Workers		Captured Workers	
10	9	6	5	7	7	5	7
# of Nodes Expanded		# of Nodes Expanded		# of Nodes Expanded		# of Nodes Expanded	
60422457	44731538	111060921	124155317	56527876	51703515	99604319	73349036
# Nodes per Move		# Nodes per Move		# Nodes per Move		# Nodes per Move	
1510561	1146962	3001646	3355549	1615082	1520691	2371531	1789000
time (ms) per move		time (ms) per move		time (ms) per move		time (ms) per move	
2792	1993	5214	6615	2668	2865	4573	3427

`X`
`    X       X`
`  X`
`               `
`    X   0   0`
`    0     X`
`  0       0 0`
`          X`
		`      X       0`
		`    X X   X 0`
		`    X   X X 0`
		`X   0`
		`  X`
		`  X 0       X 0`
		`  0`
		`  0 0 0`
				`X         X   X`
				`  X X`
				`        X`
				`0 0`
				`          0`
				`      X 0 X`
				`  0           0`
				`  0     0   X 0`
						`    X`
						`    X`
						`X     0 0   X`
						`X 0     X   0 0`
						`  X 0   0`
						`  0           X`
						`  0         0 0`
						`    X`	

2.4.3 Minimax vs. Alpha-beta

Game 1		Game 2		Game 3		Game 4	
Winner: **Player O**		Winner: **Player O**		Winner: **Player O**		Winner: **Player O**	
Total Moves: 32		Total Moves: 29		Total Moves: 43		Total Moves: 32	
Player X	Player O	Player X	Player O	Player X	Player O	Player X	Player O
Captured Workers		Captured Workers		Captured Workers		Captured Workers	
4	6	5	6	8	12	4	2
# of Nodes Expanded		# of Nodes Expanded		# of Nodes Expanded		# of Nodes Expanded	
16967234	57514018	13071935	54926563	15307902	58636061	25737426	103368601
# Nodes per Move		# Nodes per Move		# Nodes per Move		# Nodes per Move	
530226	1797313	450756	1894019	355997	1363629	804294	3230268
time (ms) per move		time (ms) per move		time (ms) per move		time (ms) per move	
404	3916	326	3686	251	2607	663	6424

`          X   0`
`X X   X   X X`
`              X`
`X       0 0   X`
`  0`
`0 0     0`
`    0       X 0`
`      0   0 0`
		`X   0     X X`
		`            X`
		`X   X     X`
		`  X           X`
		`  0 0   0   X`
		`            0 0`
		`        0`
		`0         0 0 0`
				`              0`
				`  X       X`
				`X 0`
				`    0`
				`0     X     0`
				`               `
				`              0`
				`  0           0`
						`          0 X X`
						`    X   X     0`
						`X X X`
						`  X   X X`
						`X     X     0 X`
						`  0   0 0`
						`  0   0 0 0   X`
						`0       0`	

2.4.4 Alpha-beta vs. Minimax

Game 1		Game 2		Game 3		Game 4	
Winner: **Player O**		Winner: **Player X**		Winner: **Player X**		Winner: **Player O**	
Total Moves: 41		Total Moves: 21		Total Moves: 44		Total Moves: 46	
Player X	Player O	Player X	Player O	Player X	Player O	Player X	Player O
Captured Workers		Captured Workers		Captured Workers		Captured Workers	
6	9	2	2	10	8	8	10
# of Nodes Expanded		# of Nodes Expanded		# of Nodes Expanded		# of Nodes Expanded	
54564518	12701900	42673070	12325999	47515486	13393122	108482252	19316249
# Nodes per Move		# Nodes per Move		# Nodes per Move		# Nodes per Move	
1330841	309802	2032050	616299	1079897	311467	2358309	419918
time (ms) per move		time (ms) per move		time (ms) per move		time (ms) per move	
2236	220	3944	427	2073	219	4354	317

```
X     0     X     X X X     X X                                      0
  X                 X     X   X X      0     X X          X
    X     0             0 X   0 X            X          X
                        0 X                X   0   X    X   0   0
X   0 0       0               X                    0    0 X X   0 X   0
      X                             0              X      0   0
0     0 0 X 0         0     0 0 0     0 X       0   0
0                   0 X 0 0   0 0 0             X
```

2.4.5 Summary

<u>Search Depth > Evaluation Function (Offensive win) ~ Play Order (First player win)</u>

In which ">" means dominate, "~" means roughly the same effect

(1) Search Depth:
Search depth is the most important factor in winning this game. As shown in section 2.4.3 and 2.4.4, the Alpha-beta agent with search depth of 6 dominated the game by winning 6 out of 8.

(2) Evaluation Function:
To be fair, when analyzing the effects of playing first or second and playing defensively or offensively, only those matchups with equal search depth (section 2.4.1 and 2.4.2) should be considered. To gain more samples, 8 games are simulated with the same condition as each game of section 2.4.1 and 2.4.2, giving a total of 64 equal search depth games. (Results are not shown in the report, but text files are included with source code.)
Among the 64 equal search depth games, agents with offensive evaluation function won 36, while agents with defensive evaluation function won 28. So the conclusion is that **offensive strategy performs better in this game**.

(3) Playing Order:
Among the above mentioned 64 games, the first player won 35 and the second player won 29, it seems the first player has some advantages. To eliminate the effects from evaluation function, consider only the 32 games in A3, A4, B3 and B4. The first player won 19 out of 32, the second player only won 13. So the conclusion is that **the first player has a better winning chance**.

The winning percentage for offensive strategy is in part (2) is 56%, and for first player in part (3) is 59%. It is **not clear which factor will dominate the other**, since the minor difference can easily be affected by the simulation randomness.

2.5 Extended Rules

2.5.1 Modifications

No modification is required for the Long Rectangular Board problem (simply create the game board to be 5X10), so only modifications for the “3 workers rule” are discussed in this section.

(1) Game Termination:
The game is over when either player has 3 workers arriving at the opponent’s base or either player has less than 3 workers.

(2) Evaluation Function:
As illustrated in section 2.1, the original evaluation function can be easily adjusted for “3 worker rule” by specifying the number of furthest workers (**k**) counted in the evaluation function to be 3.

2.5.2 Three Workers Matchup Results

The same settings as section 2.4 are used.

Minimax vs. Minimax

Game 1		Game 2		Game 3		Game 4	
Winner: **Player X**		Winner: **Player O**		Winner: **Player O**		Winner: **Player O**	
Total Moves: 52		Total Moves: 49		Total Moves: 47		Total Moves: 55	
Player X	Player O	Player X	Player O	Player X	Player O	Player X	Player O
Captured Workers		Captured Workers		Captured Workers		Captured Workers	
10	8	9	11	8	9	8	8
# of Nodes Expanded		# of Nodes Expanded		# of Nodes Expanded		# of Nodes Expanded	
16845026	16516432	18731567	18600709	14304184	14306106	27106722	26876180
# Nodes per Move		# Nodes per Move		# Nodes per Move		# Nodes per Move	
323942	323851	382276	379606	304344	304385	492849	488657
time (ms) per move		time (ms) per move		time (ms) per move		time (ms) per move	
260	253	304	307	243	242	482	479

```
0         0 X     X X   0 0 0     0         0 0     X     0   0   0 
  0                       0         X         X           X         
  X X     X                   X                           0       X 
      X                               X   X           X             
              0                   X                     X           
          0                   X                 0     0 0         0 
                                    0 0   0                     X   
    X   X X   0     0   0   X 0       X 0     X           X X   0   
```

Alpha-beta vs. Alpha-beta

Game 1		Game 2		Game 3		Game 4	
Winner: **Player X**		Winner: **Player O**		Winner: **Player X**		Winner: **Player O**	
Total Moves: 57		Total Moves: 60		Total Moves: 53		Total Moves: 63	
Player X	Player O	Player X	Player O	Player X	Player O	Player X	Player O
Captured Workers		Captured Workers		Captured Workers		Captured Workers	
12	12	10	8	12	12	7	9
# of Nodes Expanded		# of Nodes Expanded		# of Nodes Expanded		# of Nodes Expanded	
62780381	47090585	119540638	129208097	54556960	50470093	117279339	90747311
# Nodes per Move		# Nodes per Move		# Nodes per Move		# Nodes per Move	
1101410	840903	1992343	2153468	1029376	970578	1861576	1440433
time (ms) per move		time (ms) per move		time (ms) per move		time (ms) per move	
2278	1655	4136	4957	1918	1810	6715	3446

```
      0   0                 0 0 0         0                 0   0 0
                            X       0   X               X 0
                      0     X                   0       0     0
0             X   X                                               0
          0       0 0                                 0 X X       X
                        X             0                 0
                        X X                           X
  X X       X       X         X       X     X     X         X       X
```

Minimax vs. Alpha-beta

Game 1		Game 2		Game 3		Game 4	
Winner: **Player O**		Winner: **Player O**		Winner: **Player O**		Winner: **Player O**	
Total Moves: 54		Total Moves: 42		Total Moves: 60		Total Moves: 52	
Player X	Player O	Player X	Player O	Player X	Player O	Player X	Player O
Captured Workers		Captured Workers		Captured Workers		Captured Workers	
10	13	7	9	9	10	4	12
# of Nodes Expanded		# of Nodes Expanded		# of Nodes Expanded		# of Nodes Expanded	
16540826	69158856	14180618	61389875	14020595	57958196	16075549	69764184
# Nodes per Move		# Nodes per Move		# Nodes per Move		# Nodes per Move	
306311	1280719	337633	1461663	233676	965969	309145	1341618
time (ms) per move		time (ms) per move		time (ms) per move		time (ms) per move	
279	2760	316	3141	214	1991	286	2846

```
    0     0   0   0   0 X   0 X           0 0 0       0 0       0
                  X X     X     0   X             0     X
        0                                 X 0     X           0   0 X
  X         0                             0               0
                                      0               X           0 0
X                                                     X 0 0
                            X         X       X
0   X             0   X   0 0 0 0       X                 0   0
```

Alpha-beta vs. Minimax

Game 1		Game 2		Game 3		Game 4	
Winner: **Player O**		Winner: **Player X**		Winner: **Player X**		Winner: **Player X**	
Total Moves: 54		Total Moves: 54		Total Moves: 53		Total Moves: 56	
Player X	Player O	Player X	Player O	Player X	Player O	Player X	Player O
Captured Workers		Captured Workers		Captured Workers		Captured Workers	
11	12	13	6	9	9	6	6
# of Nodes Expanded		# of Nodes Expanded		# of Nodes Expanded		# of Nodes Expanded	
58017591	12315323	81179112	17831542	75166148	14327489	146776365	24897810
# Nodes per Move		# Nodes per Move		# Nodes per Move		# Nodes per Move	
1074399	228061	1503316	336444	1418229	275528	2621006	452687
time (ms) per move		time (ms) per move		time (ms) per move		time (ms) per move	
2053	209	3221	311	3026	251	5336	425

```
      0 0 0     |                 |             0 0 | 0 0            
X     X         |                 |           X     |               X
                |   0   X       X | X   X           | X     0       X
                |   X     X       |                 |     0       X 0
    0           |   0 X     X     |   0         0   | X              
                |                 |     0           | 0 0       X 0 0
  X     0       | X               |       X         |   0     X      
            X   | 0 X   X       X |   X 0 0 X   X   |   X     X X    
```

Summary:

Winning Chance: using the same approach as in section 2.4.5, except the total number of simulations is 32. In this new rule, the second player won 23 out of 32, thus dominates the advantage of offensive strategy, which won 18 out of 32.
Search Depth > Play Order (Second player win) > Evaluation Function (Offensive win)

Total Moves: "3 Worker Rule" requires more steps to end the game, which make sense, since game continues when 1 worker arrives opponent's base.

Total # of Expanded Nodes & # of Captured Workers: these two values increase with the number of total moves, since the time for games with "3 Worker Rule" is longer than before.

Average # of Expanded Nodes: In "3 Worker Rule" games, the average number of expanded nodes per move is smaller than original games, potentially because both players capture more opponent's workers as the game takes more moves to finish. When there is less workers left in the game, the search space for both agents reduces, resulting in the smaller average nodes per move.

2.5.3 Long Rectangular Board Matchup Results

The same settings as section 2.4 are used.

Minimax vs. Minimax

Game 1		Game 2		Game 3		Game 4	
Winner: **Player X**		Winner: **Player X**		Winner: **Player X**		Winner: **Player X**	
Total Moves: 18		Total Moves: 18		Total Moves: 8		Total Moves: 12	
Player X	Player O	Player X	Player O	Player X	Player O	Player X	Player O
Captured Workers		Captured Workers		Captured Workers		Captured Workers	
6	6	6	6	3	2	6	5
# of Nodes Expanded		# of Nodes Expanded		# of Nodes Expanded		# of Nodes Expanded	
6552592	6255490	6552592	6255490	3864298	3556663	5662930	5427664
# Nodes per Move		# Nodes per Move		# Nodes per Move		# Nodes per Move	
364032	367970	364032	367970	483037	508094	471910	493424
time (ms) per move		time (ms) per move		time (ms) per move		time (ms) per move	
252	251	239	227	325	296	299	290
X X X X X X X X 0 X X 0 0 0 0 X X 0 X 0 0 0 0 0 0 X 0 0		X X X X X X X X 0 X X 0 0 0 0 X X 0 X 0 0 0 0 0 0 X 0 0		X X X X X X X X X X X X X X X X 0 0 0 X 0 0 0 0 0 0 0 0 0 0 0 0 0 X 0		X X X X X X X X X X X X X 0 X 0 0 0 0 0 0 0 0 0 0 0 0 X 0	

Alpha-beta vs. Alpha-beta

Game 1		Game 2		Game 3		Game 4	
Winner: **Player O**		Winner: **Player X**		Winner: **Player O**		Winner: **Player X**	
Total Moves: 21		Total Moves: 21		Total Moves: 15		Total Moves: 10	
Player X	Player O	Player X	Player O	Player X	Player O	Player X	Player O
Captured Workers		Captured Workers		Captured Workers		Captured Workers	
6	10	10	6	4	6	4	4
# of Nodes Expanded		# of Nodes Expanded		# of Nodes Expanded		# of Nodes Expanded	
7807448	8238988	4416948	5761904	3968327	3942146	3955269	2032373
# Nodes per Move		# Nodes per Move		# Nodes per Move		# Nodes per Move	
371783	392332	210330	288095	264555	262809	395526	225819
time (ms) per move		time (ms) per move		time (ms) per move		time (ms) per move	
747	919	472	614	555	619	912	457
0 X X X 0 X X 0 X X X 0 X X 0 0 0 0 0 0 0 0 0 0		X X X X X X X X X X 0 0 0 X 0 0 X X 0 X 0 0 0 0		0 X X X X X X X X X X X X 0 0 X X 0 0 0 0 0 0 0 0 0 0 0 0 0		X X X X X X X X X X X 0 X X X 0 0 X 0 0 0 0 0 X 0 0 0 0 0 0 0 0	

Minimax vs. Alpha-beta

Game 1		Game 2		Game 3		Game 4	
Winner: **Player X**		Winner: **Player O**		Winner: **Player O**		Winner: **Player O**	
Total Moves: 14		Total Moves: 21		Total Moves: 13		Total Moves: 18	
Player X	Player O	Player X	Player O	Player X	Player O	Player X	Player O
Captured Workers		Captured Workers		Captured Workers		Captured Workers	
6	5	7	12	6	7	8	9
# of Nodes Expanded		# of Nodes Expanded		# of Nodes Expanded		# of Nodes Expanded	
6700201	9461730	8061561	12102758	5793542	10183542	7451057	11249841
# Nodes per Move		# Nodes per Move		# Nodes per Move		# Nodes per Move	
478585	727825	383883	576321	445657	783349	413947	624991
time (ms) per move		time (ms) per move		time (ms) per move		time (ms) per move	
312	1525	238	1236	282	1650	267	1420

```
X     X   X X X       X     0 X                   0 X X X X   X     X     X 0 X X       
X   X X X X X   X X   X     X       X   0   X   X   X X   X X X           X   X   X X 
X         0   0       0 0     X X   X           X           0   0   X     X 0   0 0     
0 0     0   0   0 0     0     0 0 0   0                 0 0   0           0   0       0 X 
0 0 0 0 0 0   X           0     0 0       0   0 0   0 0 0 0   0 0     0 0   0       0 0 
```

Alpha-beta vs. Minimax

Game 1		Game 2		Game 3		Game 4	
Winner: **Player X**		Winner: **Player O**		Winner: **Player X**		Winner: **Player X**	
Total Moves: 18		Total Moves: 25		Total Moves: 11		Total Moves: 21	
Player X	Player O	Player X	Player O	Player X	Player O	Player X	Player O
Captured Workers		Captured Workers		Captured Workers		Captured Workers	
8	7	9	13	4	3	9	6
# of Nodes Expanded		# of Nodes Expanded		# of Nodes Expanded		# of Nodes Expanded	
14694316	7272678	12258592	8485127	5997608	4454533	8100787	7299817
# Nodes per Move		# Nodes per Move		# Nodes per Move		# Nodes per Move	
816350	427804	490343	339405	545237	445453	385751	364990
time (ms) per move		time (ms) per move		time (ms) per move		time (ms) per move	
1536	255	993	213	1119	261	801	221

```
    X X       X   X   X     X   0           X X X X X X X     X         X     X X     
    0   X   X X X X   0       0     X   X   X   X X X     X         X   X X     X X   X 
X X     X               0   0     X X 0 0   0 0 0     0 0 X   X   0 X 0 0     X X   0 
0       0 0     0 0     0     X                 0   0   0   X 0   0     0     X 0 0   
X 0 0 0   0     0 0       0   0       0     0 0 0 0 X 0 0   0             X 0 0     0 
```

Summary:

Winning Chance: using the same approach as in section 2.4.5, except the total number of simulations is 32. In this new rule, probably **due to the shallow depth of the game board**, the first player won 25 out of 32, thus dominates the advantage of defensive strategy, which won 19 out of 32.
Search Depth > Play Order (First player win) > Evaluation Function (Defensive win)

Total Moves & Number of Expanded Nodes: 5x10 game board requires less steps and less number of expanded nodes to complete the game.

2.6 Greedy Agent (Bonus Points)

A 1-depth greedy agent is implemented and games are simulated between greedy agent and 4-depth minimax agent. In fact, the 1-depth greedy is simply a 1-depth minimax agent, without the second search level taking minimum of its branches. As expected, **the greedy agent has no chance to win** over minimax agent, since the search depth will dominate the winning chance of this game.

Minimax vs. Greedy

Game 1		Game 2		Game 3		Game 4	
Winner: **Player X**		Winner: **Player X**		Winner: **Player X**		Winner: **Player X**	
Total Moves: 7		Total Moves: 7		Total Moves: 8		Total Moves: 20	
Player X	Player O	Player X	Player O	Player X	Player O	Player X	Player O
Captured Workers		Captured Workers		Captured Workers		Captured Workers	
3	0	3	0	2	0	7	1
# of Nodes Expanded		# of Nodes Expanded		# of Nodes Expanded		# of Nodes Expanded	
2681446	142	2681446	142	3694938	173	10450990	443
# Nodes per Move		# Nodes per Move		# Nodes per Move		# Nodes per Move	
383063	23	383063	23	461867	24	522549	23
time (ms) per move		time (ms) per move		time (ms) per move		time (ms) per move	
389	0	314	0	374	0	430	0
X X X X X X X X		X X X X X X X X		X X X X X X X X		X X X X X X X X	
X X X X X X		X X X X X X		X X X X X X		X X	
X		X					
				X		X	
0		0		0 0		0 X X X	
0		0		0		0	
0 0 0 0 0 0		0 0 0 0 0 0		0 0 0 0		0 0 0 0 0 0	
0 X 0 0 0 0		0 X 0 0 0 0		0 0 0 0 0 X 0 0		X 0	

Greedy vs. Minimax

Game 1		Game 2		Game 3		Game 4	
Winner: **Player O**		Winner: **Player O**		Winner: **Player O**		Winner: **Player O**	
Total Moves: 13		Total Moves: 14		Total Moves: 19		Total Moves: 16	
Player X	Player O	Player X	Player O	Player X	Player O	Player X	Player O
Captured Workers		Captured Workers		Captured Workers		Captured Workers	
0	5	0	1	0	2	0	4
# of Nodes Expanded		# of Nodes Expanded		# of Nodes Expanded		# of Nodes Expanded	
327	7119054	345	6657308	443	8048441	400	9438138
# Nodes per Move		# Nodes per Move		# Nodes per Move		# Nodes per Move	
25	547619	24	475522	23	423602	25	589883
time (ms) per move		time (ms) per move		time (ms) per move		time (ms) per move	
0	486	0	487	0	425	0	612
X 0 X		X 0 X X X		0		X X 0 X	
X X X X X X		X X X X X X		X X X X X X		X X X X X	
X X X		X 0 X		X X X X X X X X		X X 0	
0		0 X X		0 0		X X	
0		X		0 0		0 0	
0 0				0		0 0	
0 0 0		0 0 0 0 0 0		0 0 0 0		0 0	
0 0 0 0 0 0 0 0		0 0 0 0 0 0 0		0 0 0 0 0 0		0 0 0 0 0 0 0 0	

3-References

[1] https://en.wikipedia.org/wiki/Minimax

[2] https://en.wikipedia.org/wiki/Alpha%E2%80%93beta_pruning

[3] https://courses.edx.org/courses/BerkeleyX/CS188x_1/1T2013/info

[4] https://en.wikipedia.org/wiki/Breakthrough_(board_game)

[5] https://en.wikipedia.org/wiki/Evaluation_function

[6] https://en.wikipedia.org/wiki/Mathematics_of_Sudoku