AI PROJECT REPORT

MiniGo

Cao Nhu Dat - Nguyen Huy Hoang

11/7/2022

Table of contents

Introduction 1

2 Methodology

Results & Analysis 3

4 To-do

1

Introduction
Traditional Go and MiniGo

1. Introduction: Traditonal Go

Go is an abstract strategy board game for two players, the aim is to surround more territory than the opponent.

The game was invented in China more than 2,500 years ago and is believed to be the oldest board game continuously played to the present day.

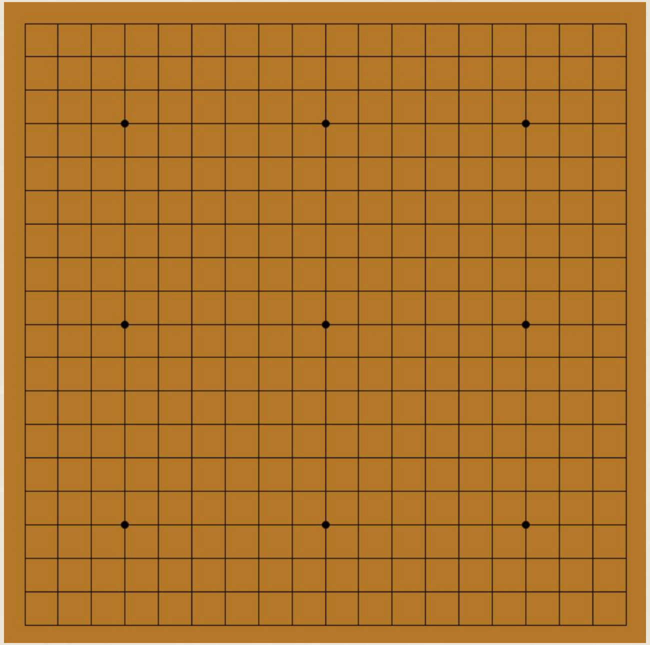
March 2016, *AlphaGo* by Google DeepMind using *Reinforcement Learning* defeated Lee Sedol in four of the five games.

1. Introduction: Traditonal Go

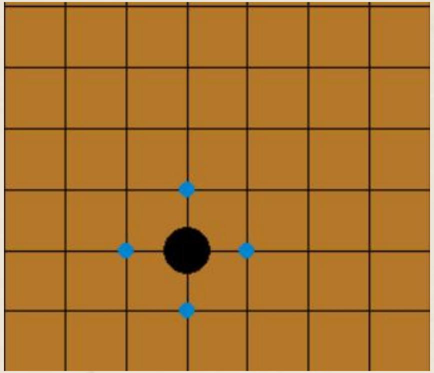
Standard Go board: 19×19 grid of lines, containing 361 points.

Beginners often play on smaller 9×9 and 13×13 boards.

Go board has 9 points called *"stars"* which help players to visualize the battle position (not affect tactics)



1. Introduction: Traditonal Go

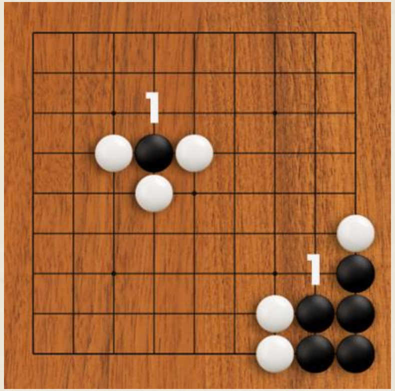


One player uses the white stones and the other, black and they take turns placing the stones on the vacant intersections. Player with black stones always go first.

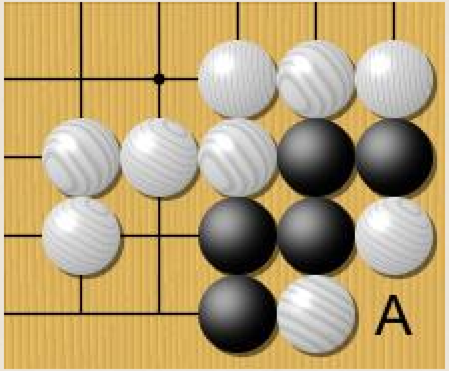
A stone when standing next to each other (horizontal, vertical) will combine to form a *"group of stones"*.

The empty intersections next to a stone or group of stones (top, bottom, left, right) are called *"liberty"*.

If the stone (or group of stones) doesn't have any liberty, in which case the stone or group is *"captured"* and is removed from the board.



1. Introduction: Traditonal Go



Suicidal move: A player may not place a stone such that it or its group immediately has no liberties, unless doing so immediately capture an enemy group

A player may “pass” a turn based on the game no longer has a chance of making a move to gain an advantage. The game ends when both players pass their turn, and then points are scored.

For each player, the number of captured stones is subtracted from the number of control points (surrounded) given in the "liberties" and the player with the higher score wins the match. Games can also be won by one player admitting defeat.

1. Introduction: MiniGo

The number of legal board positions in Go has been calculated to be approximately 2.1×10^170 => too large => **MiniGo**.

<u>Legal actions</u>

1. If there is one or more opponent's endangered groups, legal actions = those liberties.

2. Otherwise, if exist self endangered group, legal actions = those liberties.

3. Finally, if there are no endangered groups to both sides, legal actions = moves take opponent's liberties.

Note: suicidal moves will not be valid moves!

<u>Terminal condition</u>

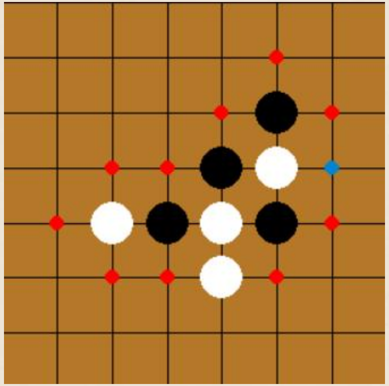
If there is a stone or group which is captured, or one player doesn't have any legal action.

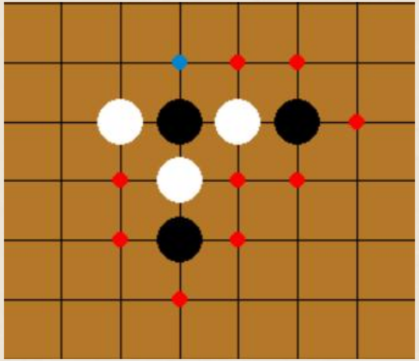
**Endangered group: a stone or group which has only 1 liberty.*

1. Introduction: MiniGo

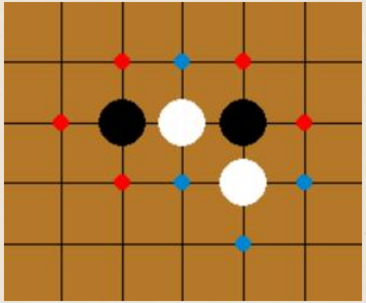
Examples: legal actions for Black

1

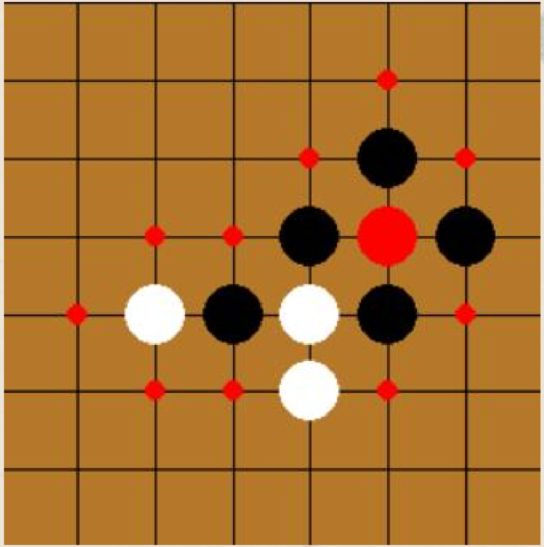




2



3

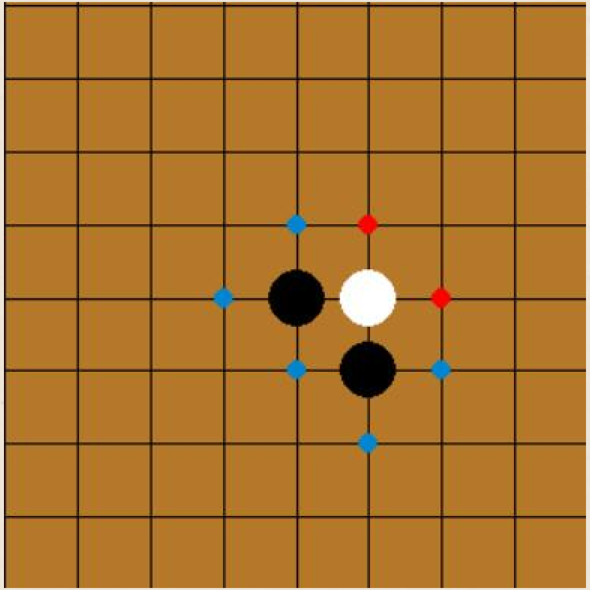


Example:Terminal state (Black win)

2

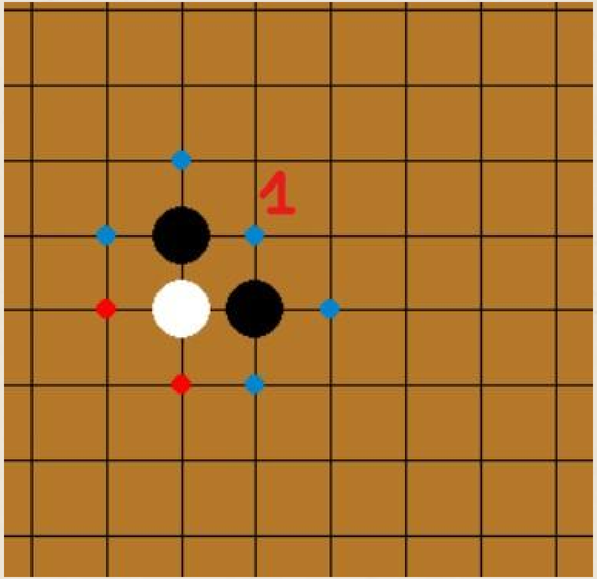
Methodology

2. Methodology: Random



Choose random action in *legal actions*. Not effective but create more game states

2. Methodology: Greedy



Get action that take the most number of opponent’s liberties

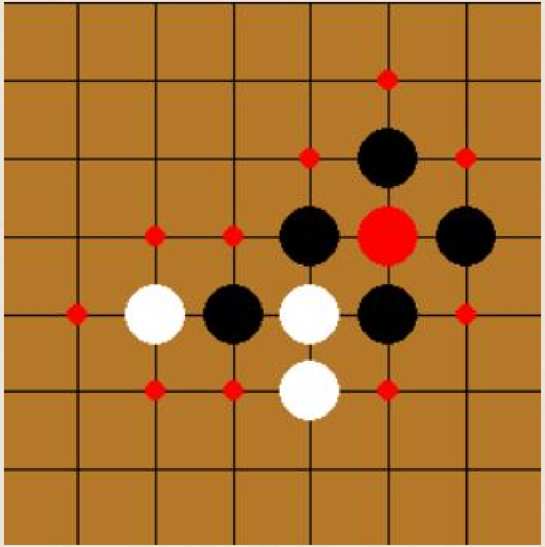
2. Methodology: Minimax and Alpha-beta prunning

Evaluation function (Board, color):

Board: game board state
color: player has the next action

Set up *score_win* = 1000 - counter_move
=> The game prioritizes higher scores for games that end earlier

1. Game state at terminal condition:
return *score_win* if color win
-*score_win* if opponent color win



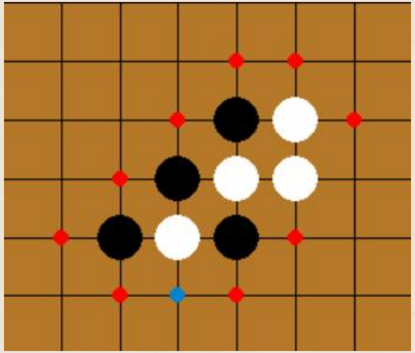
color = ‘WHITE’
return - (1000 - 9)

2. Methodology: Minimax and Alpha-beta prunning

Evaluation function (Board, color):

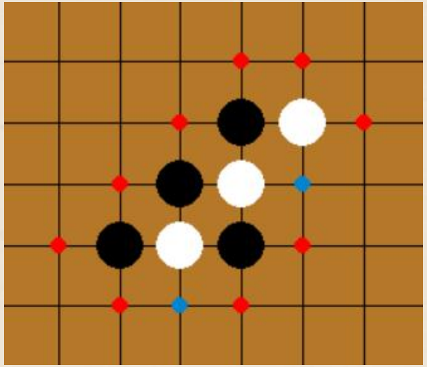
2. Game state has endangered groups

If opponent has at least 1 endangered group
Return: *score_win - 1*



color = ‘BLACK’

Else, and color (self) has more than 1 endangered group
Return: *-(score_win - 1)*



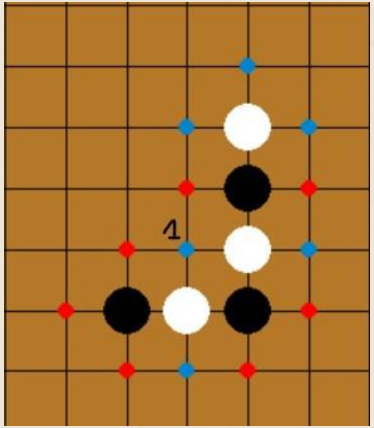
color = ‘WHITE’

2. Methodology: Minimax and Alpha-beta prunning

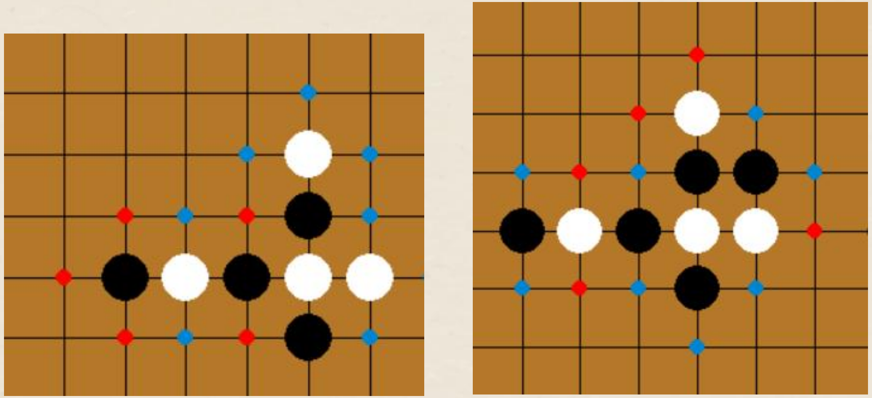
Evaluation function (Board, color):

3. Game state has endangered liberties

If opponent has at least 1 endangered liberty.Return: *score_win - 1*
Else, and color (self) has at least 1 endangered group. Return: *-(score_win - 1) if can't save*



color = 'BLACK'
white's endangered liberty is at '1'



color = 'BLACK'
example: *can save*

2. Methodology: Minimax and Alpha-beta prunning

Evaluation function (Board, color):

4. None of above

```
score_groups = num_groups_2lbt_oppo - num_groups_2lbt_self
score_liberties = num_shared_liberties_oppo - num_shared_liberties_self

Return score_groups * normal(1, 0.1) + score_liberties * normal(1, 0.1)
```

num_groups_2lbt_self(oppo) : number of groups which have 2 liberties
num_shared_liberties_self(oppo) : number of liberties which belong to more than 1 group
normal(1, 0.1): gaussian distribution with mean = 1 and standard deviation = 0.1

2. Methodology: Minimax and Alpha-beta prunning

Deploy minimax:

```
def get_action(self, board):

    if len(board._get_legal_actions()) == 1:
        return board._get_legal_actions()[0]

    score, actions = self.max_value(board, 0)

    return actions[0] if len(actions) > 0 else None
```

2. Methodology: Minimax and Alpha-beta prunning

Deploy minimax:

```
def max_value(self, board, depth):

    if terminal_state or depth == self.depth:
        return evaluation(board, color), [ ]

    for action in legal_actions:
        score, actions = min_value(board.generate_successor_state(action), depth+1)
        if score > max_score:
            max_score = score
            max_score_actions = [action]

    return max_score, max_score_actions
```

2. Methodology: Minimax and Alpha-beta prunning

Deploy minimax:

```
def min_value(self, board, depth):

    if terminal_state or depth == self.depth:
        return evaluation(board, color), [ ]

    for action in legal_actions:
        score, actions = max_value(board.generate_successor_state(action), depth+1)
        if score < min_score:
            min_score = score
            min_score_actions = [action]

    return min_score, min_score_actions
```

2. Methodology: Minimax and Alpha-beta prunning

Deploy Alpha-beta prunning:

```
if max_score > beta:
    return max_score, max_score_actions
if max_score > alpha:
    alpha = max_score
```

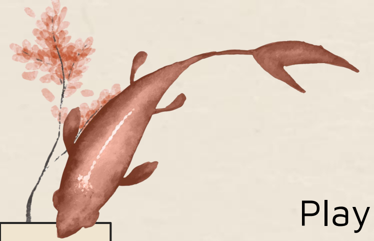
in *max_value*

in *min_value*

```
if min_score < alpha:
        return min_score, min_score_actions
if min_score < beta:
        beta = min_score
```

3

Results & Analysis



Benchmark

Play for 100 games, show *win_rate* of self_agent and *average time* per match

Self agent	Random	Greedy	M(3)	Ab(3)	M(4)	Ab(4)	Ab(6)
Random	64%	64%	17%	16%	0%	1%	0%
	0.002s	0.002s	4.6s	1.5s	19s	5.8s	128s
Greedy	65%	52%	60%	58%	4%	5%	0%
	0.001s	0.002s	1.1s	0.7s	13.9s	1.3s	108s
Ab(3)	99%	80%	100%	100%	27%	25%	0%
	1.1s	3.3s	0.14s	0.12s	13s	2.7s	16.8s

M(n): Minimax with depth = n
Ab(n): Alpha-beta prunning with depth = n

Analysis:

1. Algorithms' goodness in ascending order: Random, Greedy, Minimax, Alpha-beta prunning.

2. Random is not good in results but it creates multiple versions of the state

3. For Minimax and Alpha-beta prunning, when increasing the depth, the effect will be better. Even depth is naturally slightly better than odd depth, because the depth optimization gives more for “Max”.

4. Alpha-beta prunning has a significantly faster run time than Minimax with the same depth because it have a smaller search space.

5. For the same algorithm, Black will be better. It is shown by the expert!

To-do

1 Find out many other excellent evaluation functions for agents. Develop appropriate techniques for the complete Go.

2 Apply Machine Learning techniques and Reinforcement Learning (RL)

Thanks!

Does anyone have any questions?