DATA130008

Introduction to Artificial Intelligence

An $\alpha - \beta$ pruning approach to python-based Gomuku agent

Shao Yanjun 19307110036
Shao Yi 19307130113

2021-11-14

1 Introduction

1.1 Gomoku

Gomoku, also called Five in a Row, is an abstract strategy board game. It is traditionally played with Go pieces (black and white stones) on a Go board. It is played using a 15×15 board while in the past a 19×19 board was standard. Within the Gomoku competition, however, we are assigned to develop some basic-level Gomoku agents on a 20×20 board for the sake of simplicity.

Historically, researchers have been applying artificial intelligence techniques on playing gomoku for several decades. In 1994, L. Victor Allis raised the algorithm of proof-number search (pn-search) and dependency-based search (db-search), and proved that when starting from an empty 15×15 board, the first player has a winning strategy using these searching algorithms. This applies to both freestyle gomoku and standard gomoku without any opening rules. It seems very likely that black wins on larger boards too.

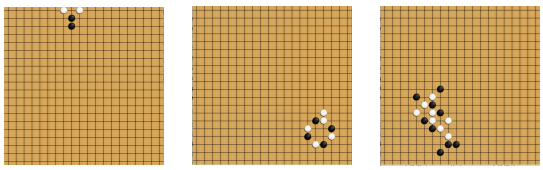

Figure 1: Openings Overview

Therefore, hosts for contests came up with extra rules including Renju, Caro, Omok and Tournament Opening Rules. Renju, which is slightly different from traditional Freestyle Gomoku, bans the usage of three and three, four and four, and overlines applied to Black only. But our Gomoku contest provides 3 openings for each pair of AI under the rule of Freestyle Gomoku and generates a rating for every single participant. All ratings are calculated using Bayesian Elo with eloAdvantage = 0, eloDraw = 0.01, and default prior.

1.2 $\alpha - \beta$ pruning

Alpha–beta pruning is a search algorithm that seeks to decrease the number of nodes that are evaluated by the minimax algorithm in its search tree.

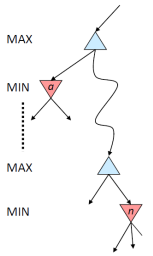

Figure 2: $\alpha - \beta$ pruning

Algorithm 1 $\alpha - \beta$ Pruning Search

```
function ALPHA-BETA-SEARCH(board, color) returns an action
    depth ← 0
    v, action ← MAX-VALUE(board, color, −∞, ∞, depth)
    return action

function MAX-VALUE(board, color, α, β, depth) returns v and action
    if TERMINAL-TEST(depth) then return UTILITY()
    v ← −∞
    for each (x, y) in frontier do
        MOVE(board, color, (x, y))
        move-v, move-action ← MIN-VALUE(board, opponent-color, α, β, depth+1)
        if move-v > v then v ← move-v, action ← a
        if v ≥ β then return v and action
        α ← MAX(α, v)
    return action

function MIN-VALUE(board, color, α, β, depth) returns v and action
    if TERMINAL-TEST(depth) then return UTILITY()
    v ← ∞
    for each (x, y) in frontier do
        MOVE(board, color, (x, y))
        move-v, move-action ← MAX-VALUE(board, opponent-color, α, β, depth+1)
        if move-v < v then v ← move-v, action ← a
        if v ≤ α then return v and action
        β ← MIN(β, v)
    return action
```

2 Main

In this section, we will have to delve deeper into the implementation of $\alpha - \beta$ pruning techniques. Some details about how to evaluate the board through Utility() and how to define the frontier of a min-max search is literally what we are going to discuss about.

2.1 Utility

xxxxx

2.1.1 First Subsubtask

xxxxx

2.2 Frontier

The frontier in a min-max tree directly determines the validity of the searching algorithm and the size of the searching tree. A trade-off being compelling, a good agent is obliged to select a bounded number of possible actions for the sake of high performance, while it is not supposed to leave out any of the optimal solutions.
Through careful spectating of several classical Gomoku games, we heuristically claim that majority of meaningful actions for a specific board state stays within 2 reaches.

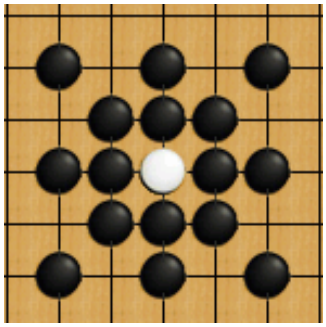

Figure 3: 2 reaches

However, if the agent is played on an aged CPU with unbearably high searching latency, we may make a small concession to half the frontier into 1 reach.

3 Performance

3.1 Implementation

Unfortunately, our $\alpha - \beta$ pruning agent can only afford min-max search for $depth = 1$. Two agents, *pydan1-1* and *pydan1-2*, namely depth-1 search with 1-reach frontier and 2-reach frontier, are implemented with the help of pyinstaller.

3.2 Tournament

We conducted a tournament between *pydan1-1*, *pydan1-2* and the benchmark agent *MUSHROOM*, and the results are as follows.

-	pydan1-1	pydan1-2	mushroom
pydan1-1	-	1 : 1	0 : 2
pydan1-2	1 : 1	-	0 : 2
mushroom	2 : 0	2 : 0	-
Total	3 : 1	3 : 1	0 : 4
Ratio	3.000	3.000	0.000
Points	4	4	0

4 Inspection of Future

4.1 AAA