**Gomoku Implementation with Min-Max and Alpha-Beta algorithm**

**Author SheepCore**

**SID 20164586**

**I. Question Description**

AlphaGo now is on fire due to the issue of beating world-class go champion winner – Ke Jie. It was absolutely amazing for people to think about what AI can do and what AI can not do. And we perceive that the algorithm behind AlphaGo is a mix of neural network and with self-play mechanics. Everybody knows the story of AlphaGo:

AlphaGo is the first computer program to defeat a professional human Go player, the first program to defeat a Go world champion, and arguably the strongest Go player in history.

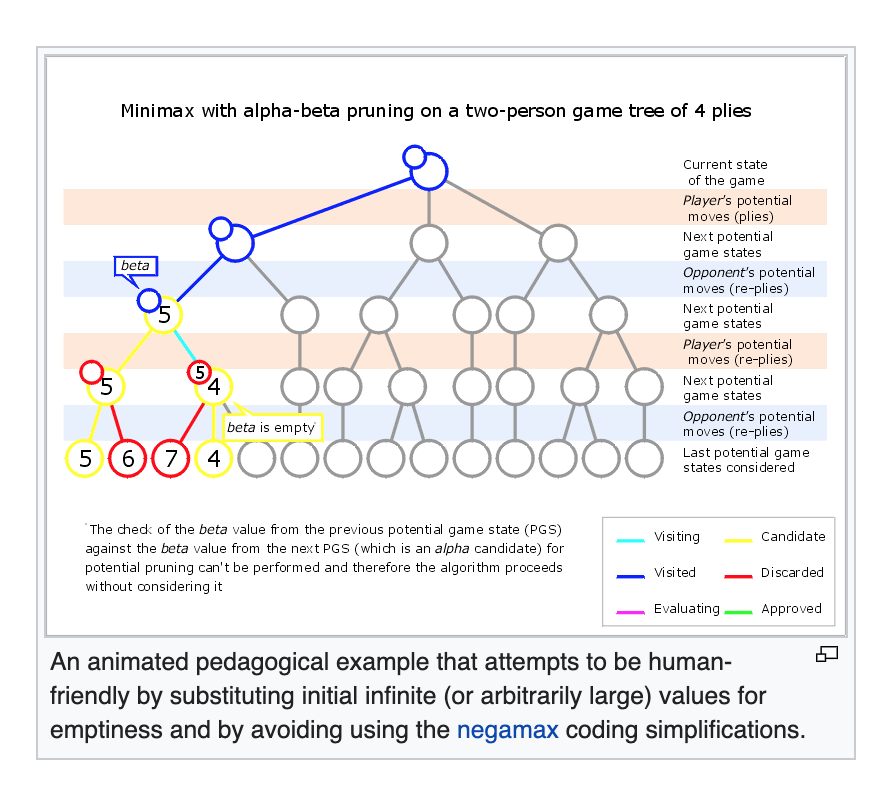
AlphaGo’s first formal match was against the reigning 3-times European Champion, Mr Fan Hui, in October 2015. Its 5-0 win was the first ever against a Go professional, and the results were published in full technical detail in the international journal, Nature. AlphaGo then went on to compete against legendary player Mr Lee Sedol, winner of 18 world titles and widely considered to be the greatest player of the past decade.

AlphaGo's 4-1 victory in Seoul, South Korea, in March 2016 was watched by over 200 million people worldwide. It was a landmark achievement that experts agreed was a decade ahead of its time, and earned AlphaGo a 9 dan professional ranking (the highest certification) - the first time a computer Go player had ever received the accolade.

**Alpha–beta pruning** is a search algorithm that seeks to decrease the number of nodes that are evaluated by the min-max algorithm in its [search tree](https://en.wikipedia.org/wiki/Game_tree). It is an adversarial search algorithm used commonly for machine playing of two-player games ([Tic-tac-toe](https://en.wikipedia.org/wiki/Tic-tac-toe), [Chess](https://en.wikipedia.org/wiki/Chess), [Go](https://en.wikipedia.org/wiki/Go_(board_game)), etc.). It stops evaluating a move when at least one possibility has been found that proves the move to be worse than a previously examined move. Such moves need not be evaluated further. When applied to a standard minimax tree, it returns the same move as minimax would, but prunes away branches that cannot possibly influence the final decision.

So why do we use alpha-beta pruning algorithm, the main goal is to elevate the efficiency of three search, otherwise we have to deeply search on some unnecessary branches which costs a lot. Normally speaking, The algorithm maintains two values, alpha and beta, which represent the minimum score that the maximizing player is assured of and the maximum score that the minimizing player is assured of respectively. Initially, alpha is negative infinity and beta is positive infinity, i.e. both players start with their worst possible score. Whenever the maximum score that the minimizing player (i.e. the "beta" player) is assured of becomes less than the minimum score that the maximizing player (i.e., the "alpha" player) is assured of (i.e. beta ≤ alpha), the maximizing player need not consider further descendants of this node, as they will never be reached in the actual play.

This is what alpha-beta looks like:



Here is my Pseudocode for minimax algorithm with alpha-beta pruning:

function minimax(node, depth, isMaximizingPlayer, alpha, beta):

if node is a leaf node :

return value of the node

if isMaximizingPlayer :

bestVal = -INFINITY

for each child node :

value = minimax(node, depth+1, false, alpha, beta)

bestVal = max( bestVal, value)

alpha = max( alpha, bestVal)

if beta <= alpha:

break

return bestVal

else :

bestVal = +INFINITY

for each child node :

value = minimax(node, depth+1, true, alpha, beta)

bestVal = min( bestVal, value)

beta = min( beta, bestVal)

if beta <= alpha:

break

return bestVal

// Calling the function for the first time.

minimax(0, 0, true, -INFINITY, +INFINITY)

**II. Implementation Process**

1. Evaluation for board state

Here is the score matrix for each board layout.

shape\_score = [(50, (0, 1, 1, 0, 0)),  
 (50, (0, 0, 1, 1, 0)),  
 (200, (1, 1, 0, 1, 0)),  
 (500, (0, 0, 1, 1, 1)),  
 (500, (1, 1, 1, 0, 0)),  
 (5000, (0, 1, 1, 1, 0)),  
 (5000, (0, 1, 0, 1, 1, 0)),  
 (5000, (0, 1, 1, 0, 1, 0)),  
 (5000, (1, 1, 1, 0, 1)),  
 (5000, (1, 1, 0, 1, 1)),  
 (5000, (1, 0, 1, 1, 1)),  
 (5000, (1, 1, 1, 1, 0)),  
 (5000, (0, 1, 1, 1, 1)),  
 (50000, (0, 1, 1, 1, 1, 0)),  
 (99999999, (1, 1, 1, 1, 1))]

1. Calculate evaluation value

Determine final evaluation value with heuristic algorithm.

def evaluation(is\_ai):  
 total\_score = 0  
 if is\_ai:  
 my\_list = list1  
 enemy\_list = list2  
 else:  
 my\_list = list2  
 enemy\_list = list1  
  
 # calculate evaluation value  
 score\_all\_arr = []   
 my\_score = 0  
 for pt in my\_list:  
 m = pt[0]  
 n = pt[1]  
 my\_score += cal\_score(m, n, 0, 1, enemy\_list, my\_list, score\_all\_arr)  
 my\_score += cal\_score(m, n, 1, 0, enemy\_list, my\_list, score\_all\_arr)  
 my\_score += cal\_score(m, n, 1, 1, enemy\_list, my\_list, score\_all\_arr)  
 my\_score += cal\_score(m, n, -1, 1, enemy\_list, my\_list, score\_all\_arr)  
  
 # calculate evaluation value of opponent  
 score\_all\_arr\_enemy = []  
 enemy\_score = 0  
 for pt in enemy\_list:  
 m = pt[0]  
 n = pt[1]  
 enemy\_score += cal\_score(m, n, 0, 1, my\_list, enemy\_list,  
 score\_all\_arr\_enemy)  
 enemy\_score += cal\_score(m, n, 1, 0, my\_list, enemy\_list,  
 score\_all\_arr\_enemy)  
 enemy\_score += cal\_score(m, n, 1, 1, my\_list, enemy\_list,  
 score\_all\_arr\_enemy)  
 enemy\_score += cal\_score(m, n, -1, 1, my\_list, enemy\_list,  
 score\_all\_arr\_enemy)  
  
 # determine final score  
 total\_score = my\_score - enemy\_score \* ratio \* 0.1  
 return total\_score

1. Gomoku Game Board Definition

# define Gomoku Window  
def gomokuWin():  
 win = GraphWin("Tiny Gomoku Game: ", GRID\_WIDTH \* COLUMN,  
 GRID\_WIDTH \* ROW)  
 win.setBackground("deeppink")  
  
 # horizontalLine verticalLine 分别绘制棋盘纵横线  
 horizontalLine = 0  
 while horizontalLine <= GRID\_WIDTH \* COLUMN:  
 l = Line(Point(horizontalLine, 0), Point(horizontalLine, GRID\_WIDTH \* COLUMN))  
 l.draw(win)  
 horizontalLine = horizontalLine + GRID\_WIDTH  
 verticalLine = 0  
 while verticalLine <= GRID\_WIDTH \* ROW:  
 l = Line(Point(0, verticalLine), Point(GRID\_WIDTH \* ROW, verticalLine))  
 l.draw(win)  
 verticalLine = verticalLine + GRID\_WIDTH  
 return win

1. Alpha-Beta Algorithm

def negamax(is\_ai, depth, alpha, beta):  
 # return evaluation value when the game reaches the end  
 if game\_win(list1) or game\_win(list2) or depth == 0:  
 return evaluation(is\_ai)  
  
 # list\_all - list3 = {blank\_list}  
 blank\_list = list(set(list\_all).difference(set(list3)))  
  
 order(blank\_list)   
  
 # traverse all blank space   
 for next\_step in blank\_list:  
  
 global search\_count  
 search\_count += 1  
  
 # no action if no neighbor around  
 if not has\_neighbor(next\_step):  
 continue  
 # check which is gonna go...  
 if is\_ai:  
 list1.append(next\_step)  
 else:  
 list2.append(next\_step)  
 list3.append(next\_step)  
  
 # switch to the opponent mode  
 value = -negamax(not is\_ai, depth - 1, -beta, -alpha)  
  
   
 if is\_ai:  
 list1.remove(next\_step)  
 else:  
 list2.remove(next\_step)  
 list3.remove(next\_step)  
  
 if value > alpha:  
 print(str(value) + " [alpha: " + str(alpha) + "beta:" + str(  
 beta) + ']')  
 print(list3)  
 # check if not step over max search depth  
 if depth == DEPTH:  
 next\_point[0] = next\_step[0]  
 next\_point[1] = next\_step[1]  
 # alpha-beta pruning   
 if value >= beta:  
 global cut\_count  
 cut\_count += 1  
 return beta  
 alpha = value  
  
 return alpha

**III. Parameter Adjusting**

Here are several super-parameters we need to adjust to satisfy our algorithm better:

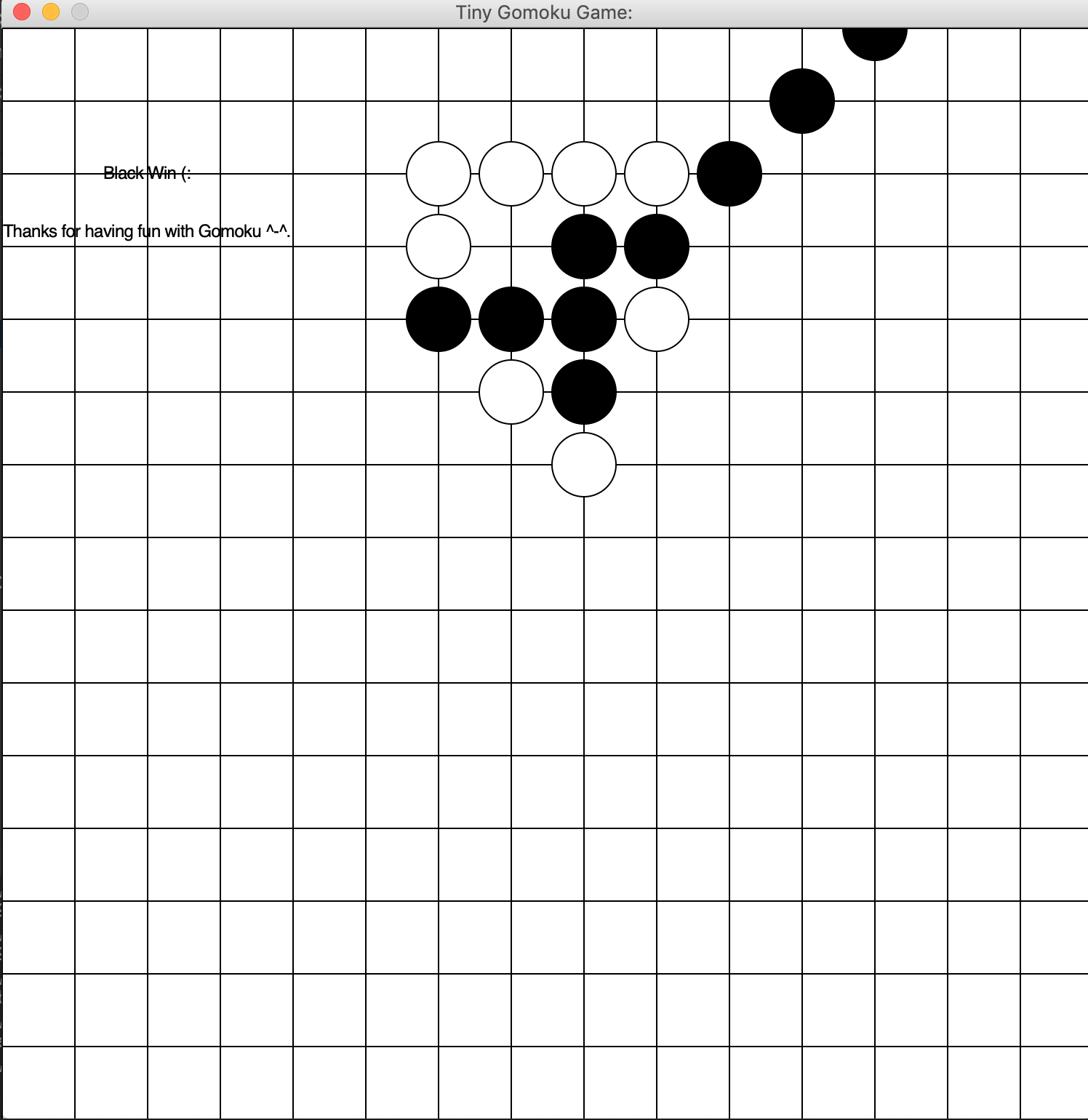
1. **MAX\_SEARCH\_DEPTH**
2. **Ratio**
3. **Who is the first run**
4. **Winning Board States**

Here is a simple analyzation based on those super-parameters I mentioned before.

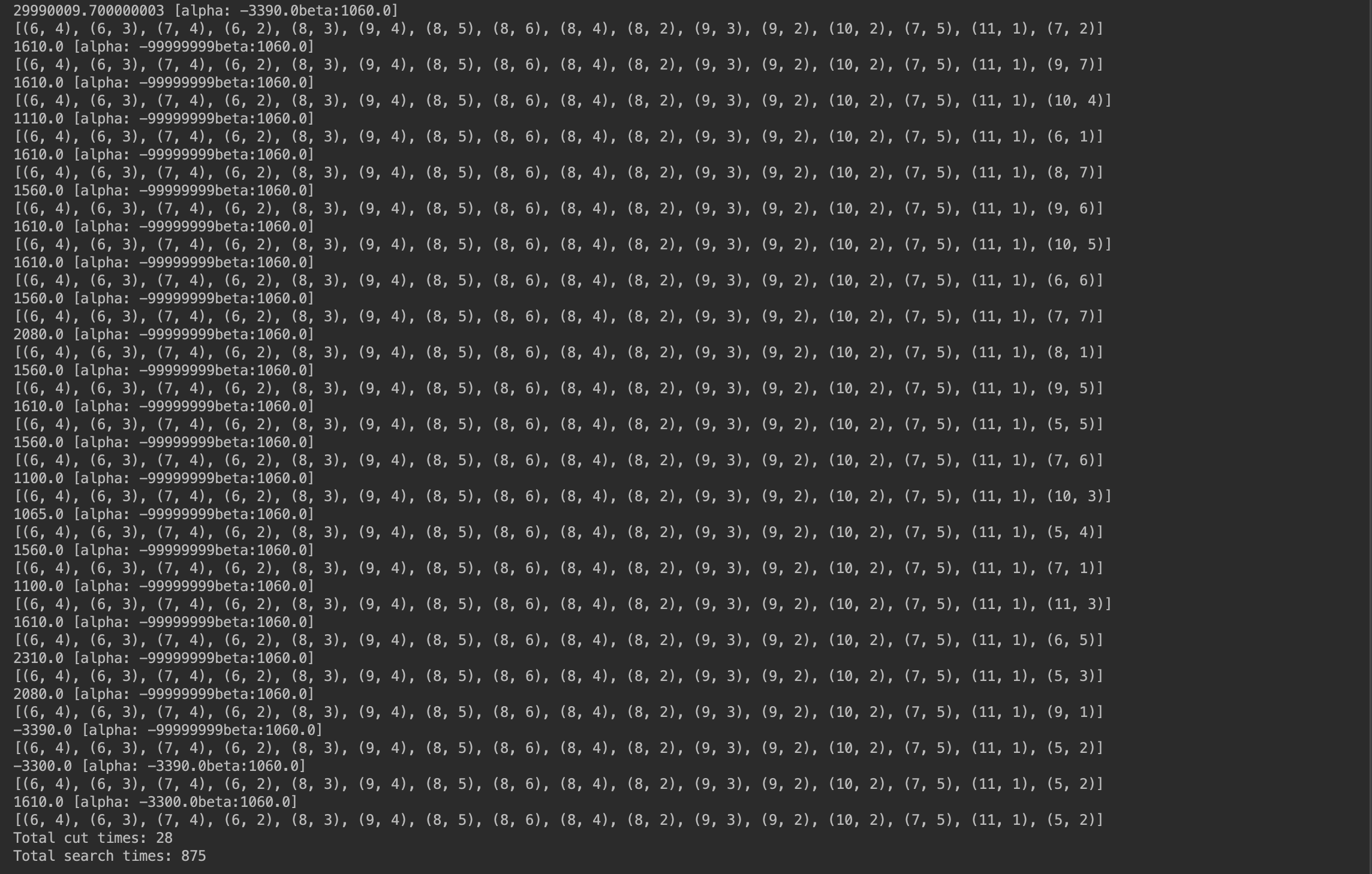
**Adjusting MAX\_SEARCH\_DEPTH:**

☞Set **MAX\_SEARCH\_DEPTH = 2:**

✔︎Performance:

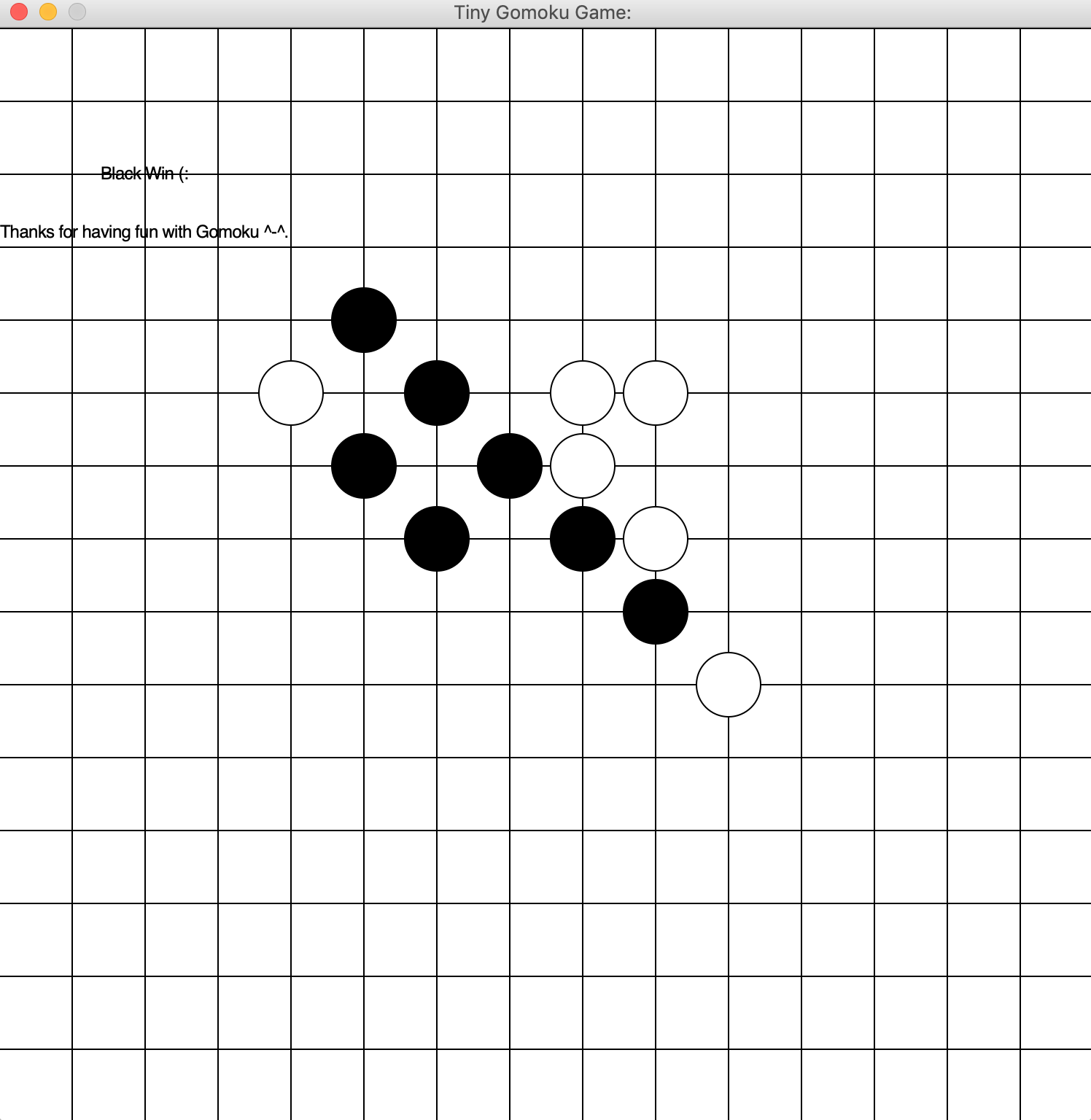


Statistical analysis:

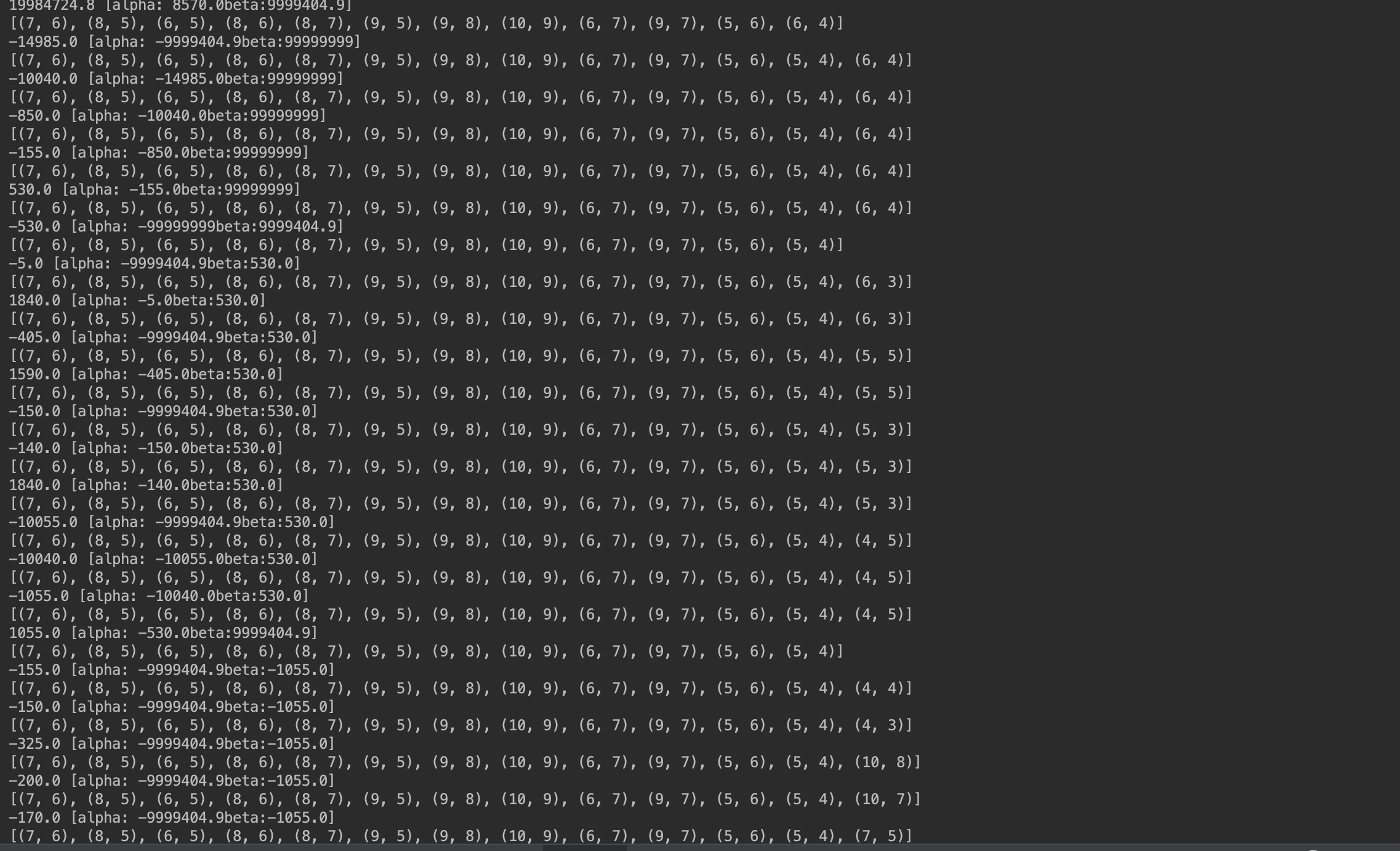


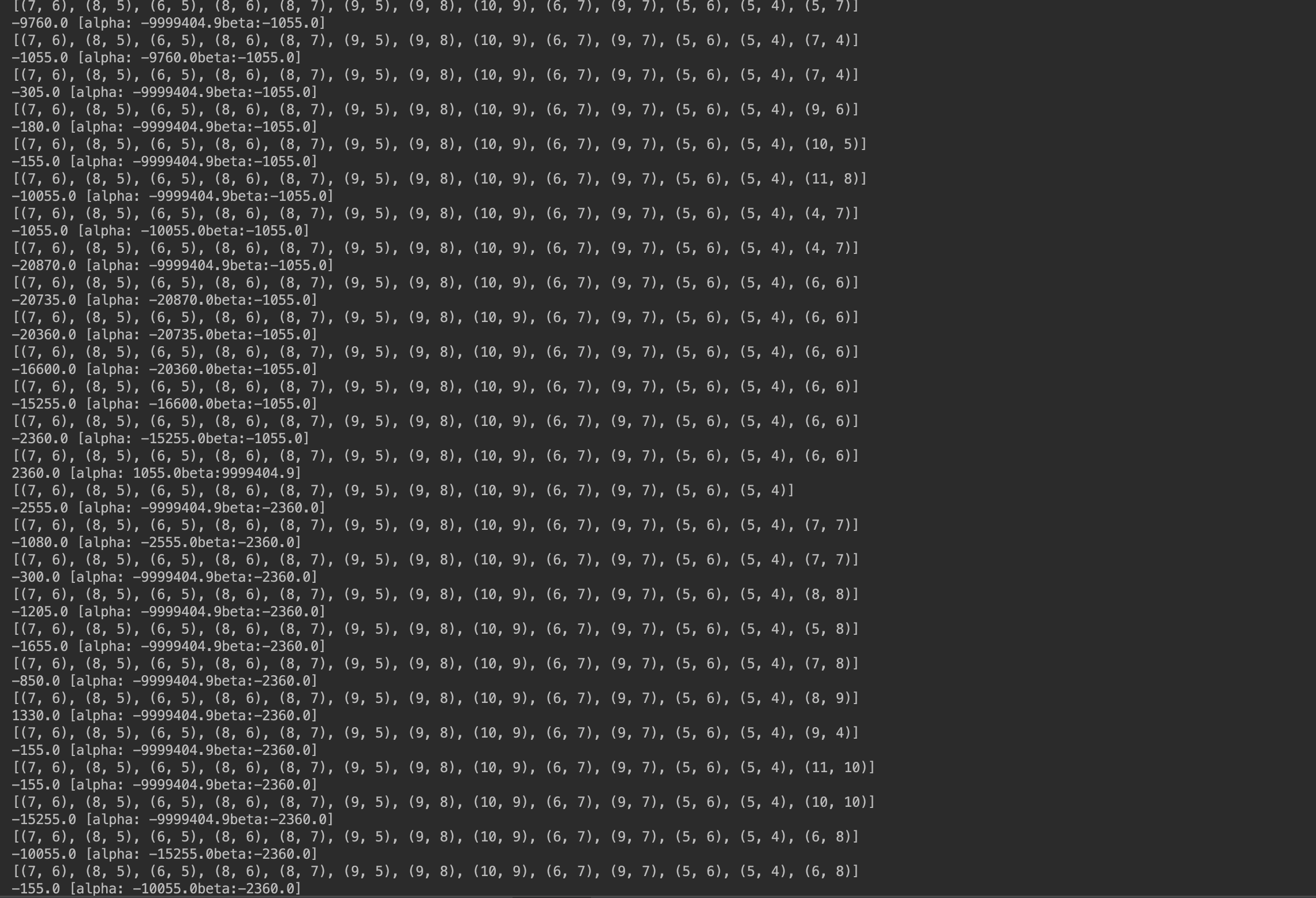
☞Set **MAX\_SEARCH\_DEPTH = 3:**

✔︎Performance:



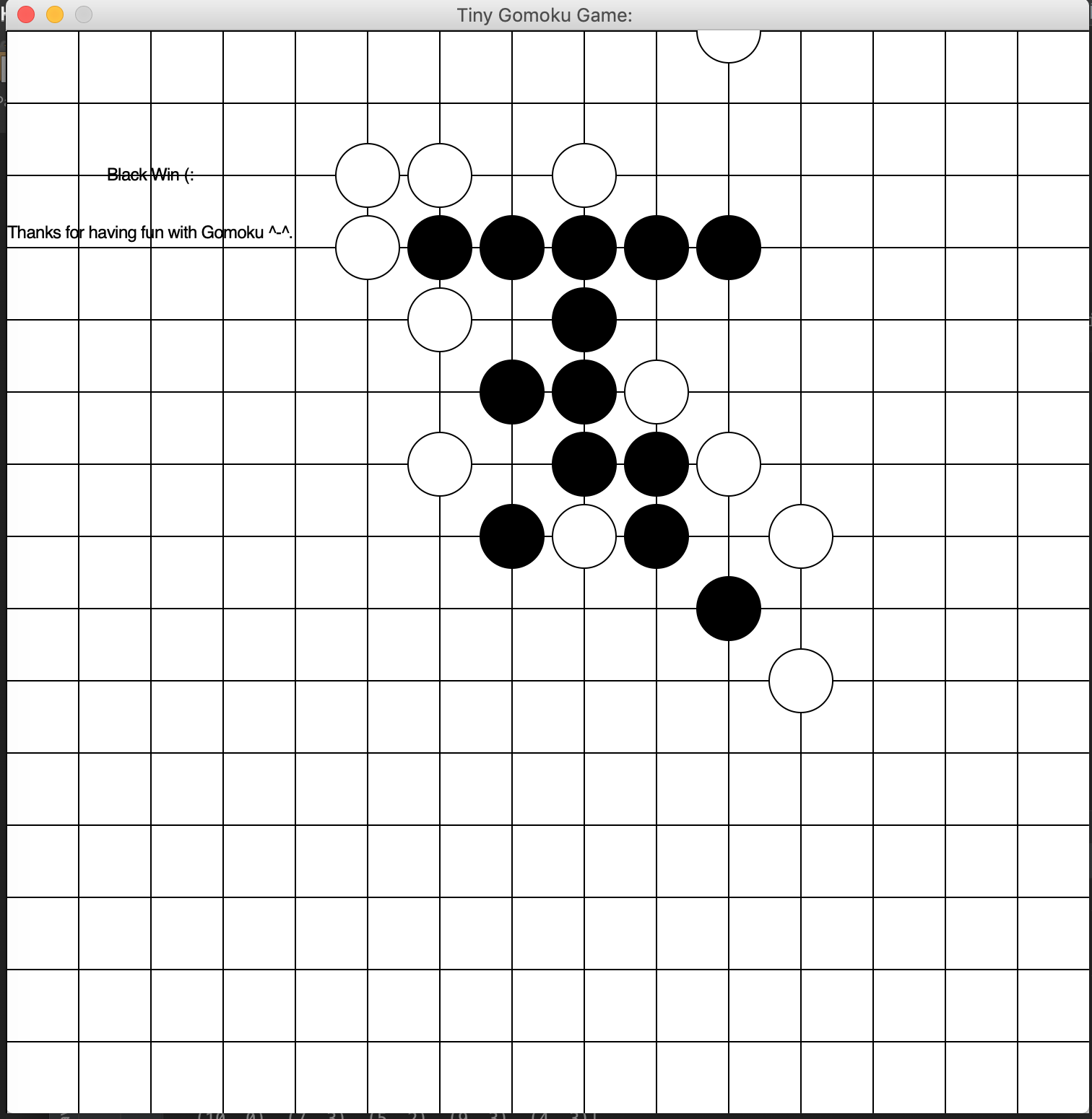
Statistical analysis:





☞Set **MAX\_SEARCH\_DEPTH = 4:**

✔︎Performance:



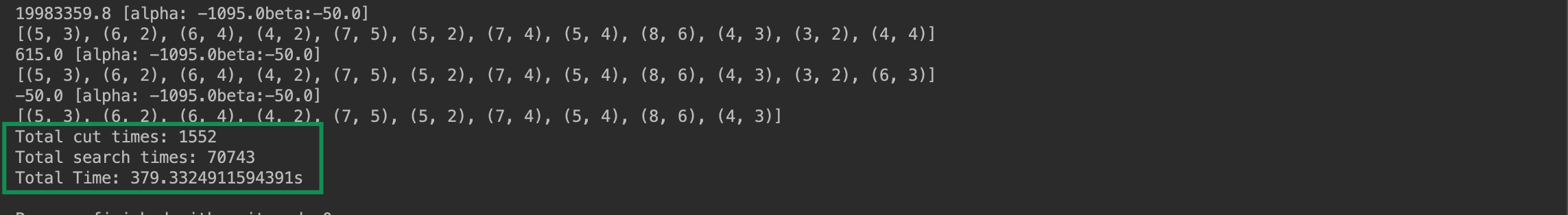
Cut times lasting for too long…

…

…

…

Statistical analysis:



**Conclusion for adjusting MAX\_SEARCH\_DEPTH:**

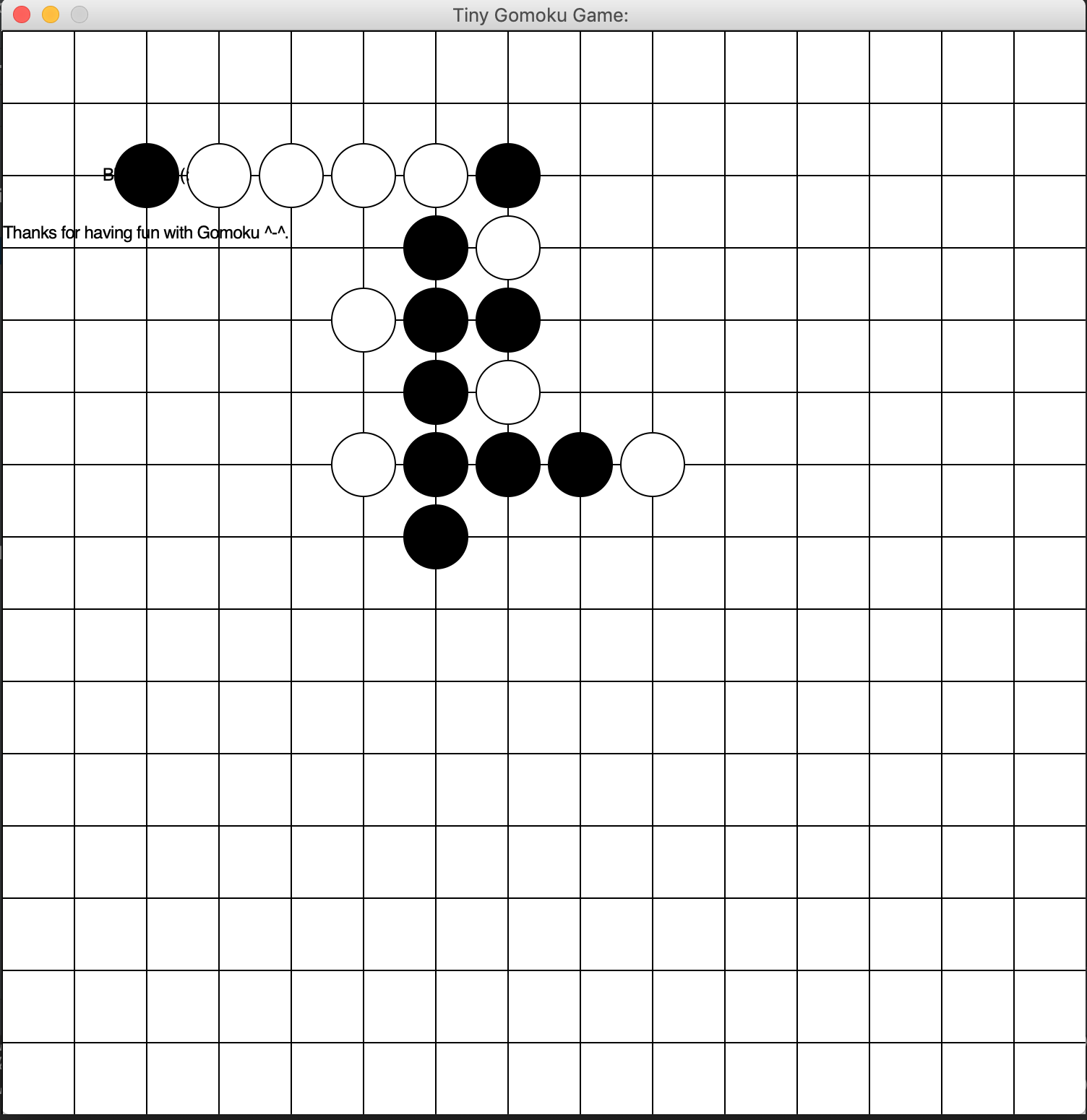
Let’s take a look what happen when adjust MAX\_SEARCH\_DEPTH, Here is the comparison on those performances:

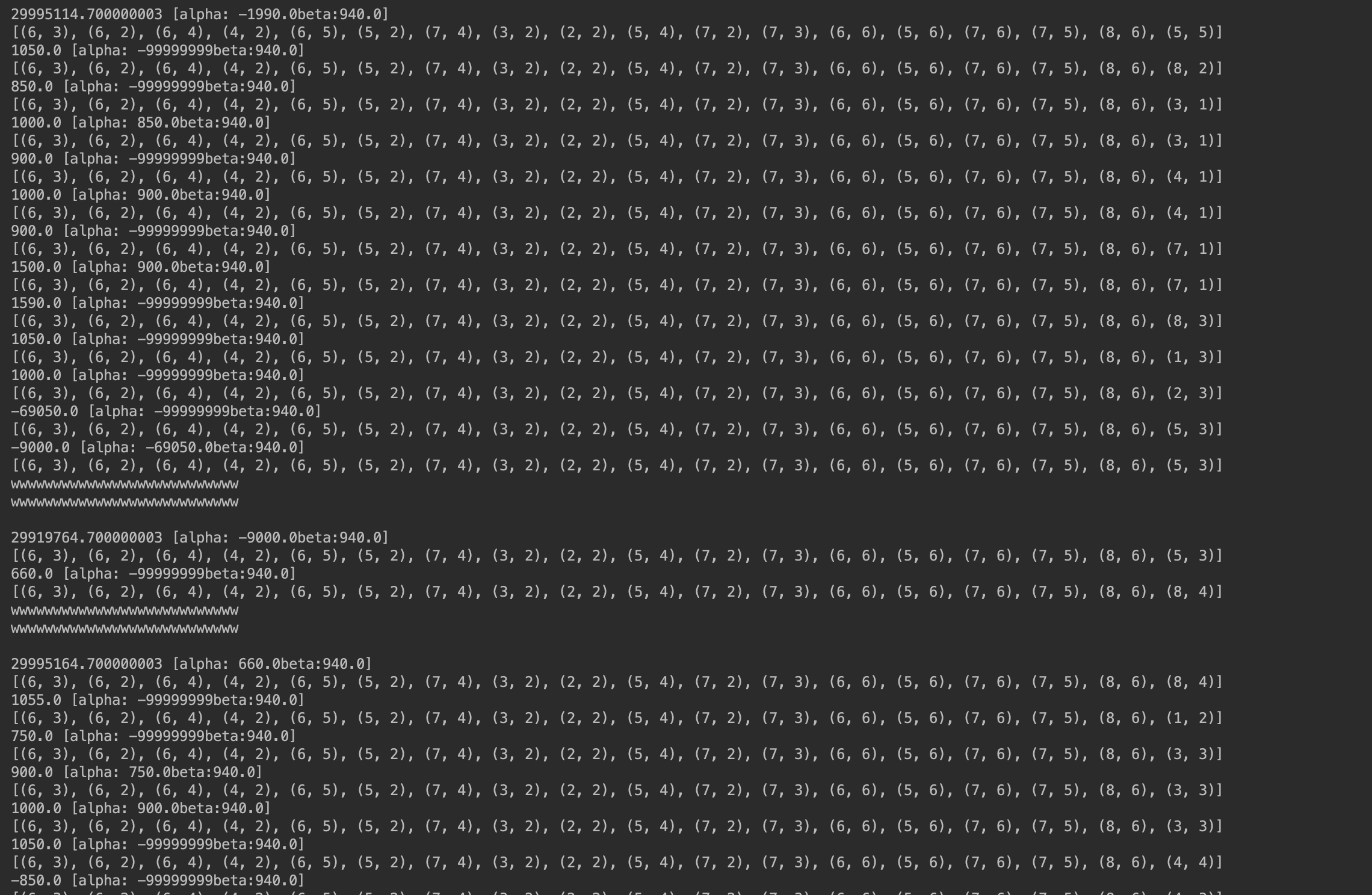
|  |  |  |  |
| --- | --- | --- | --- |
| Performance/  Max\_Search\_Depth | 2 | 3 | 4 |
| Total Cut Times | 28 | 266 | 1552 |
| Total Search Times | 785 | 2300 | 70743 |
| Total Time | 18.44s | 76.33s | 379.88s |

**Adjusting ratio:**

☞Set ratio = 1:

✔︎Performance:



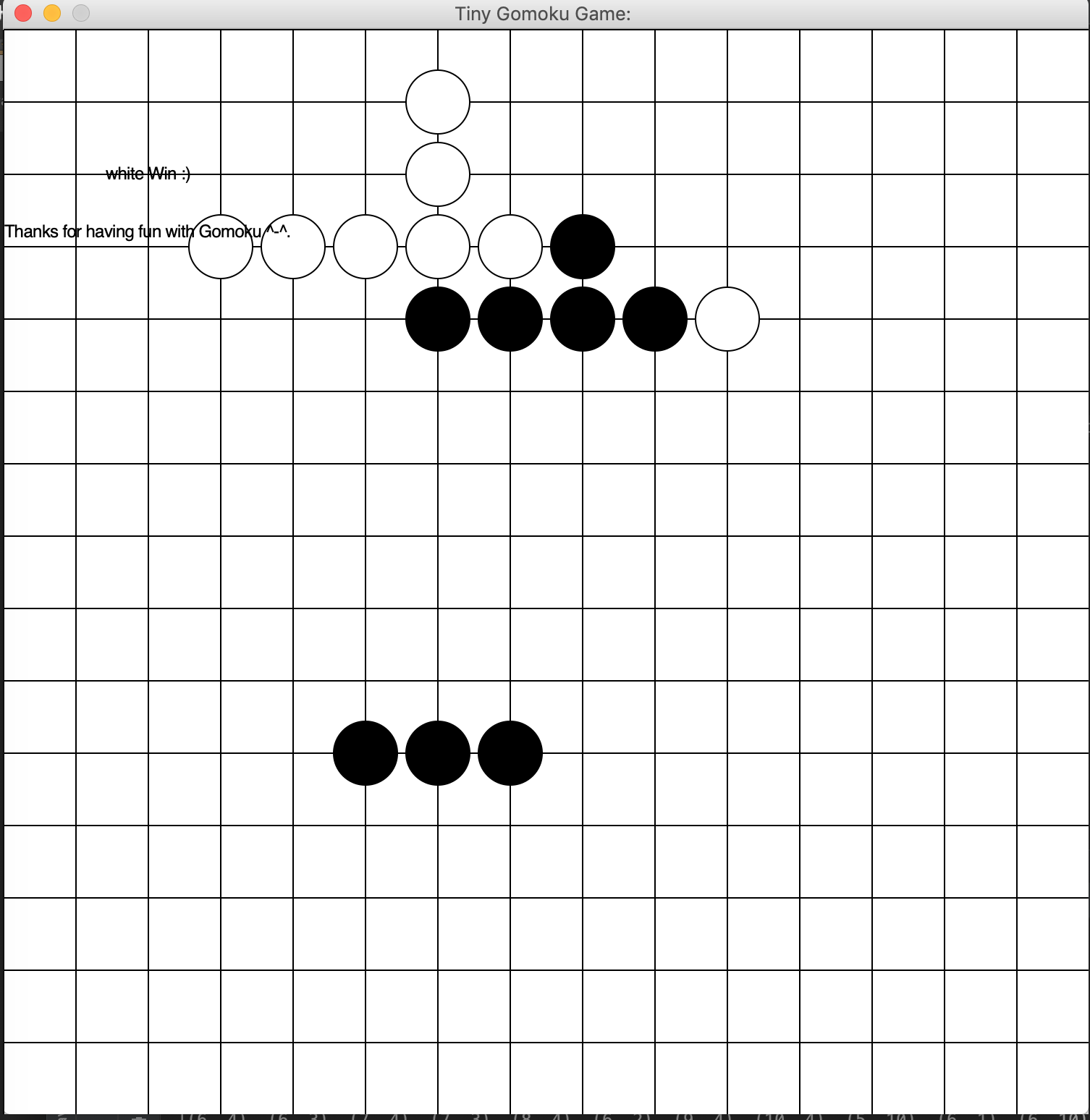


Statistical analysis:

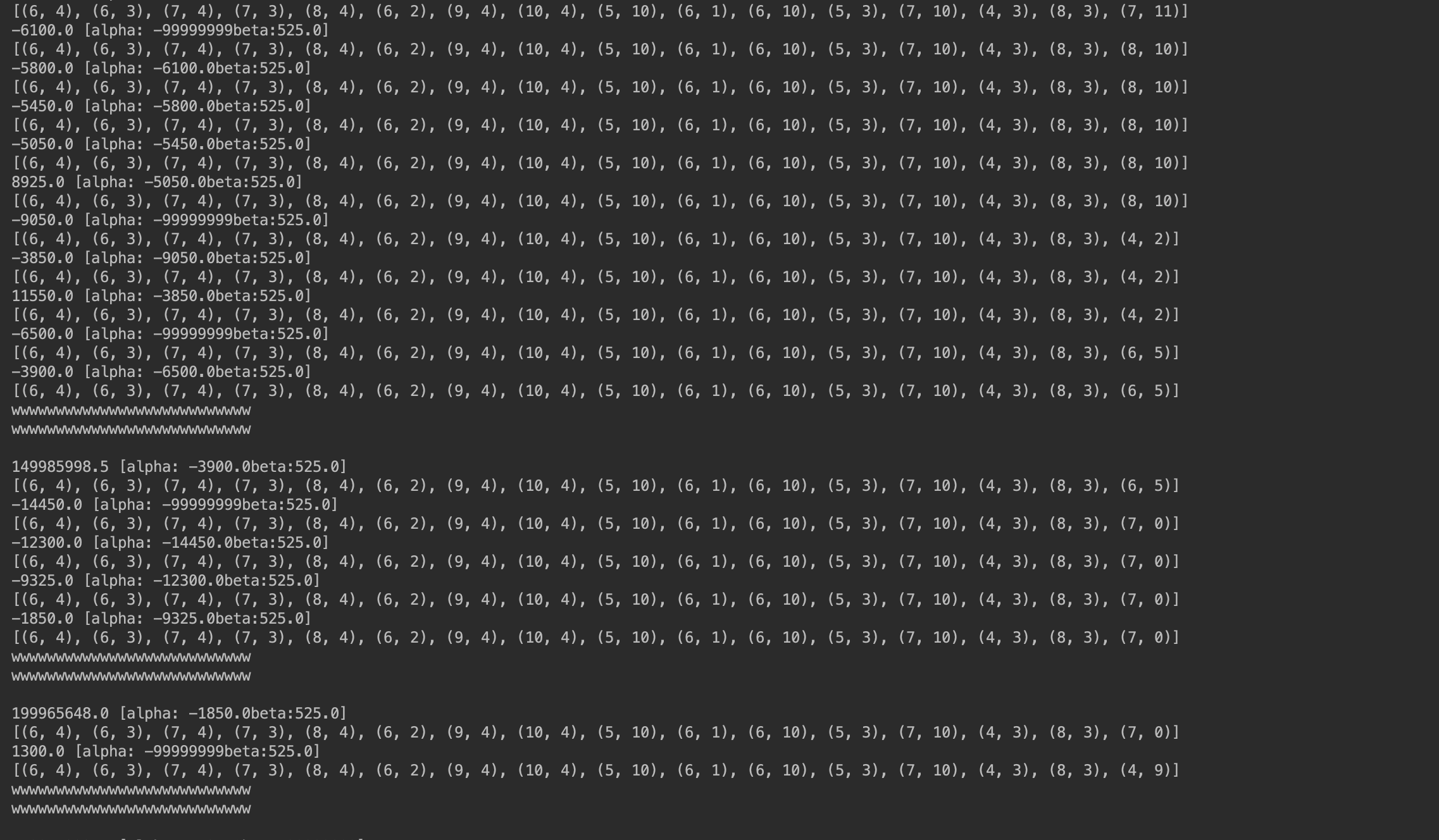


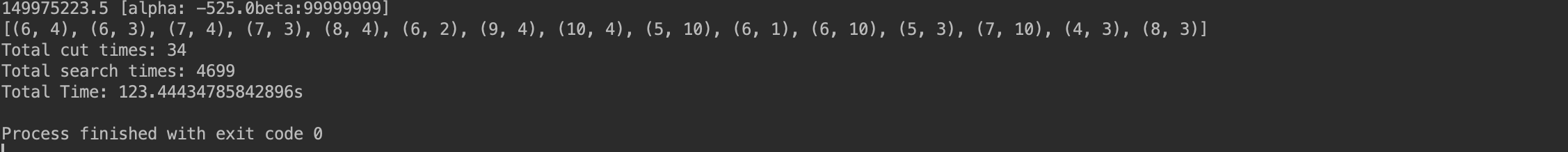
☞Set ratio = 5:

✔︎Performance:



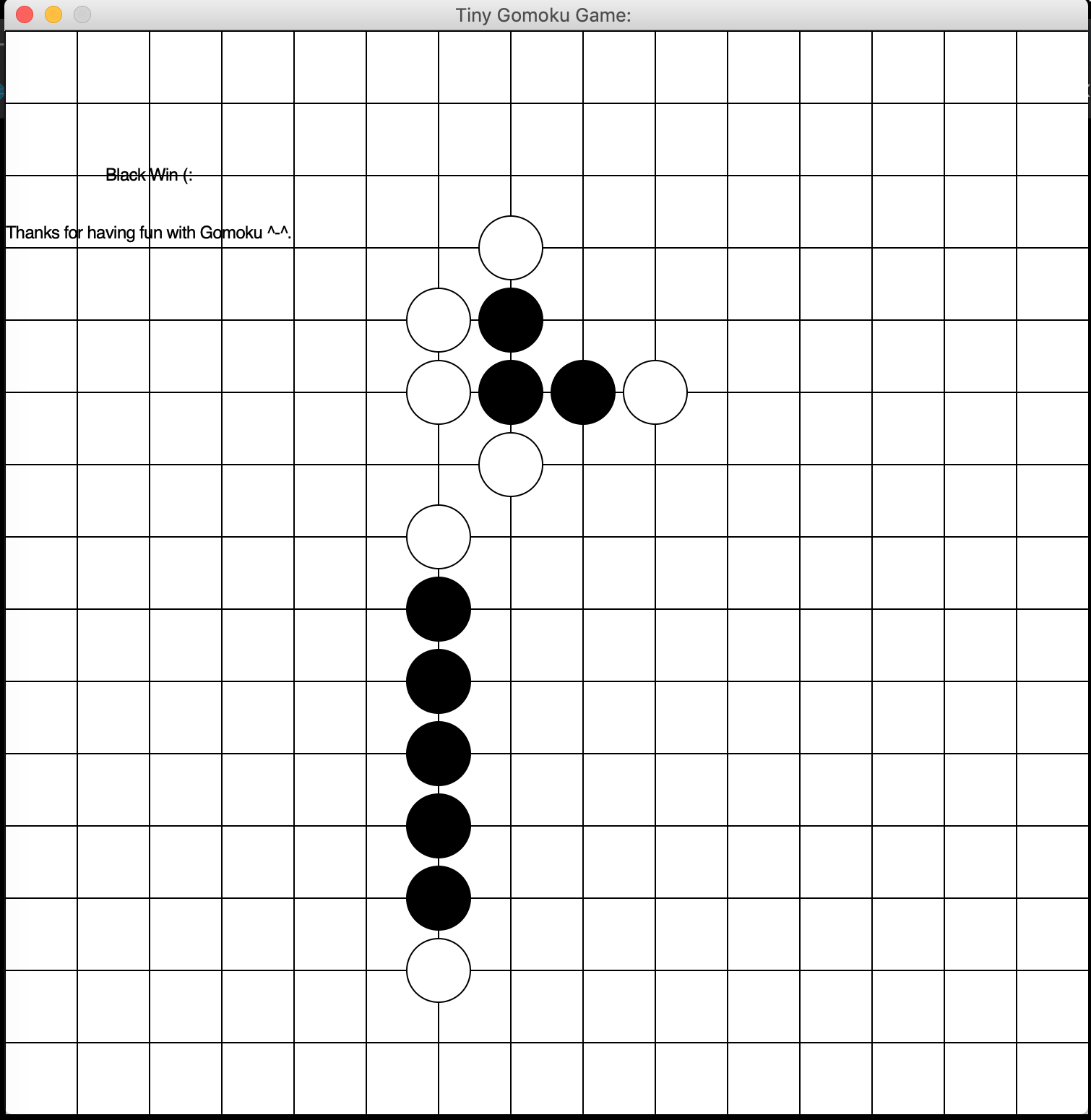
Statistical analysis:



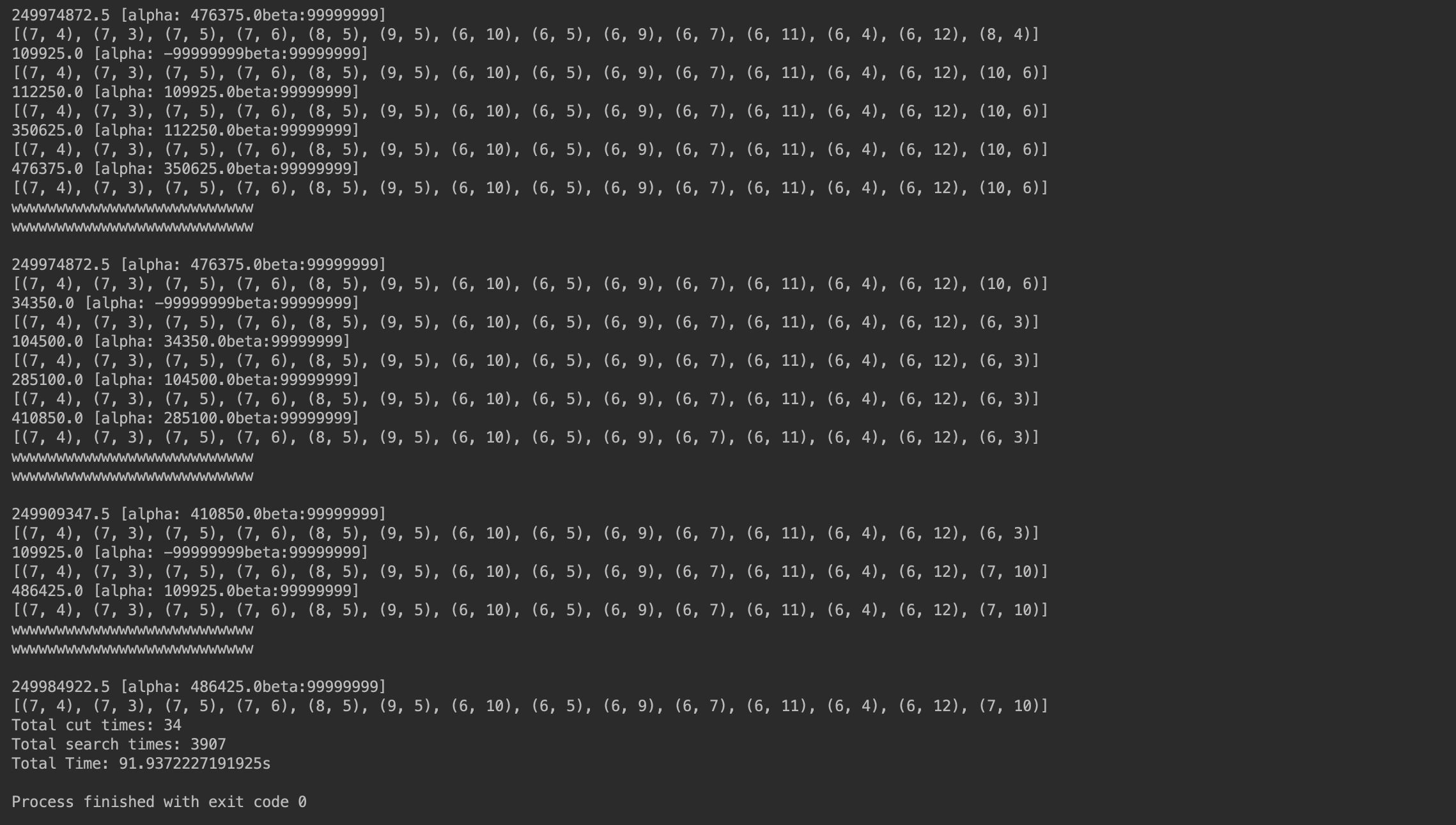


☞Set ratio = 25:

✔︎Performance:



Statistical analysis:



**Conclusion for adjusting Ratio:**

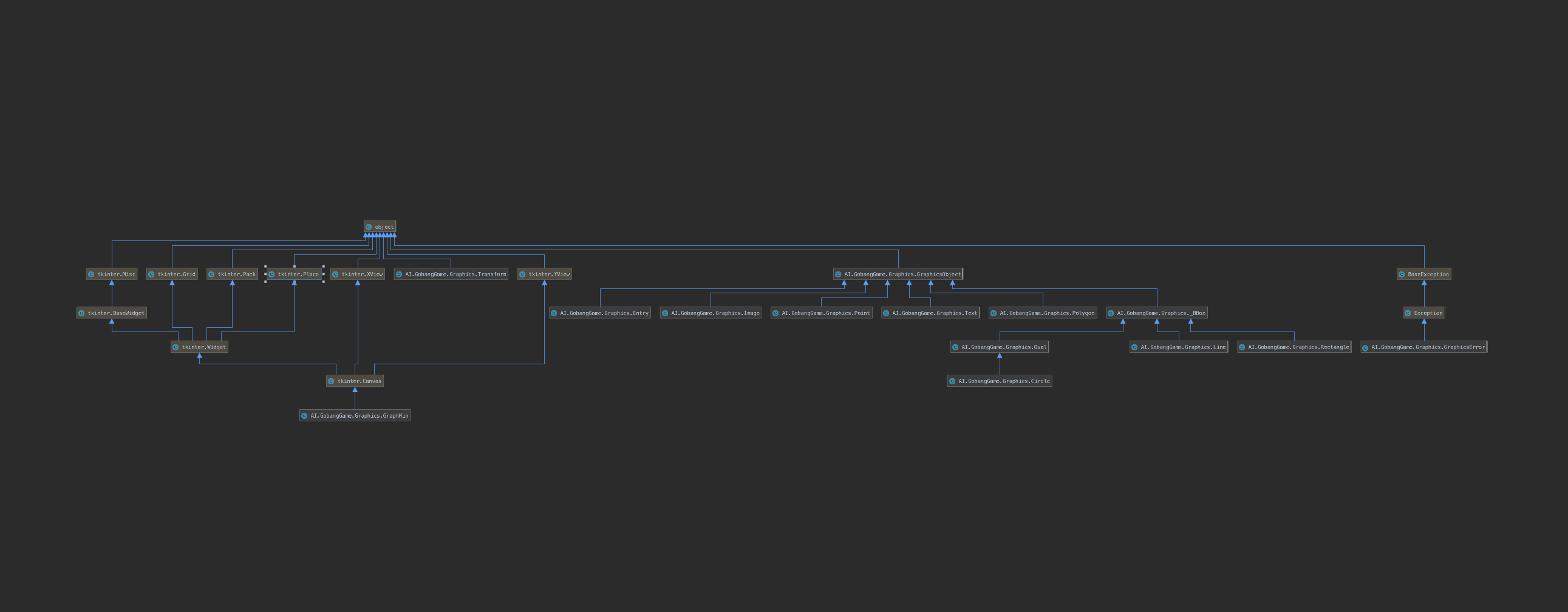
Let’s take a look what happen when adjust Ratio, Here is the comparison on those performances:

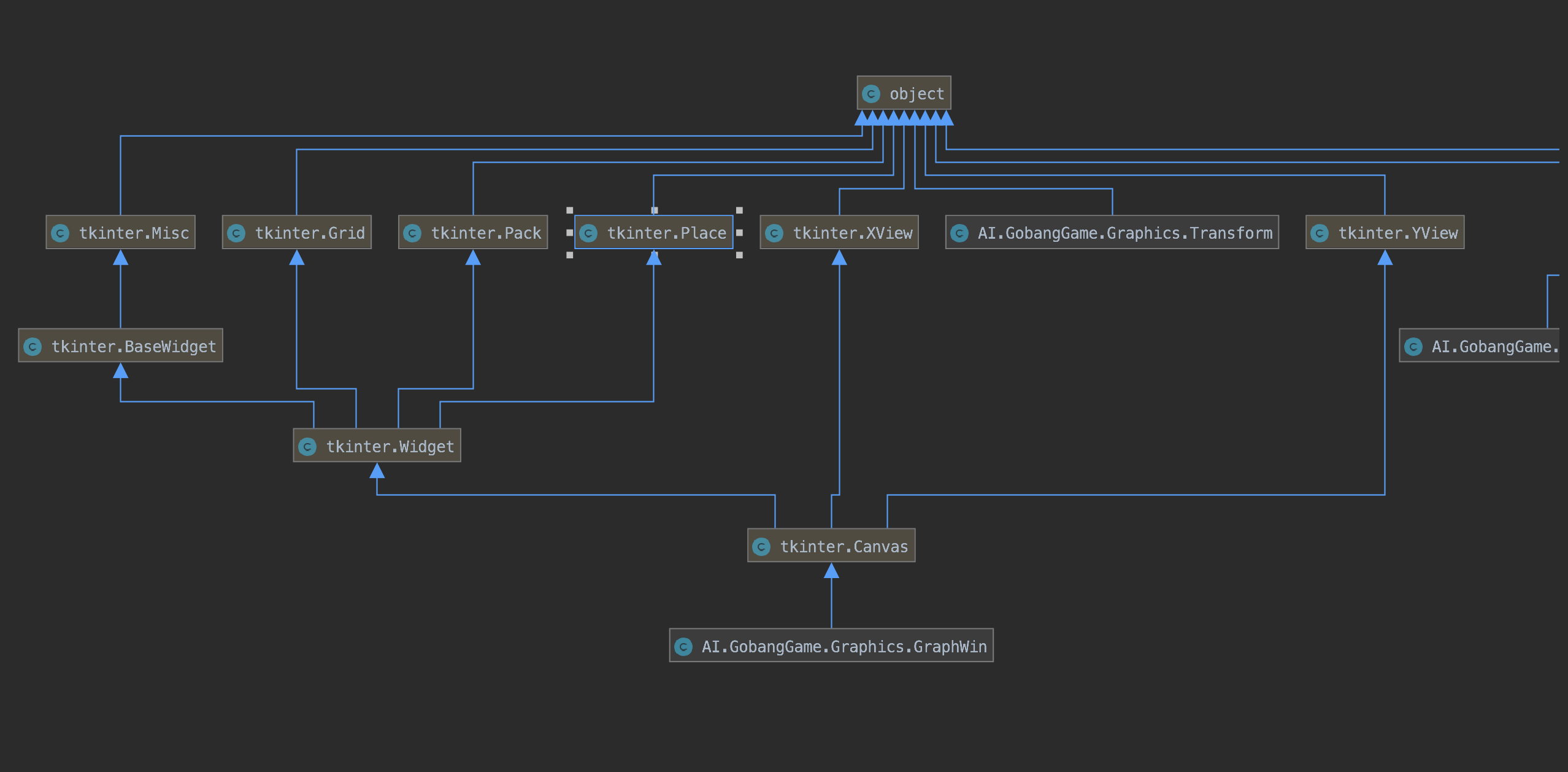
|  |  |  |  |
| --- | --- | --- | --- |
| Performance/  Ratio | 1 | 5 | 25 |
| Total Cut Times | 27 | 34 | 34 |
| Total Search Times | 2861 | 4699 | 3907 |
| Total Time | 125.16s | 123.44s | 91.93s |

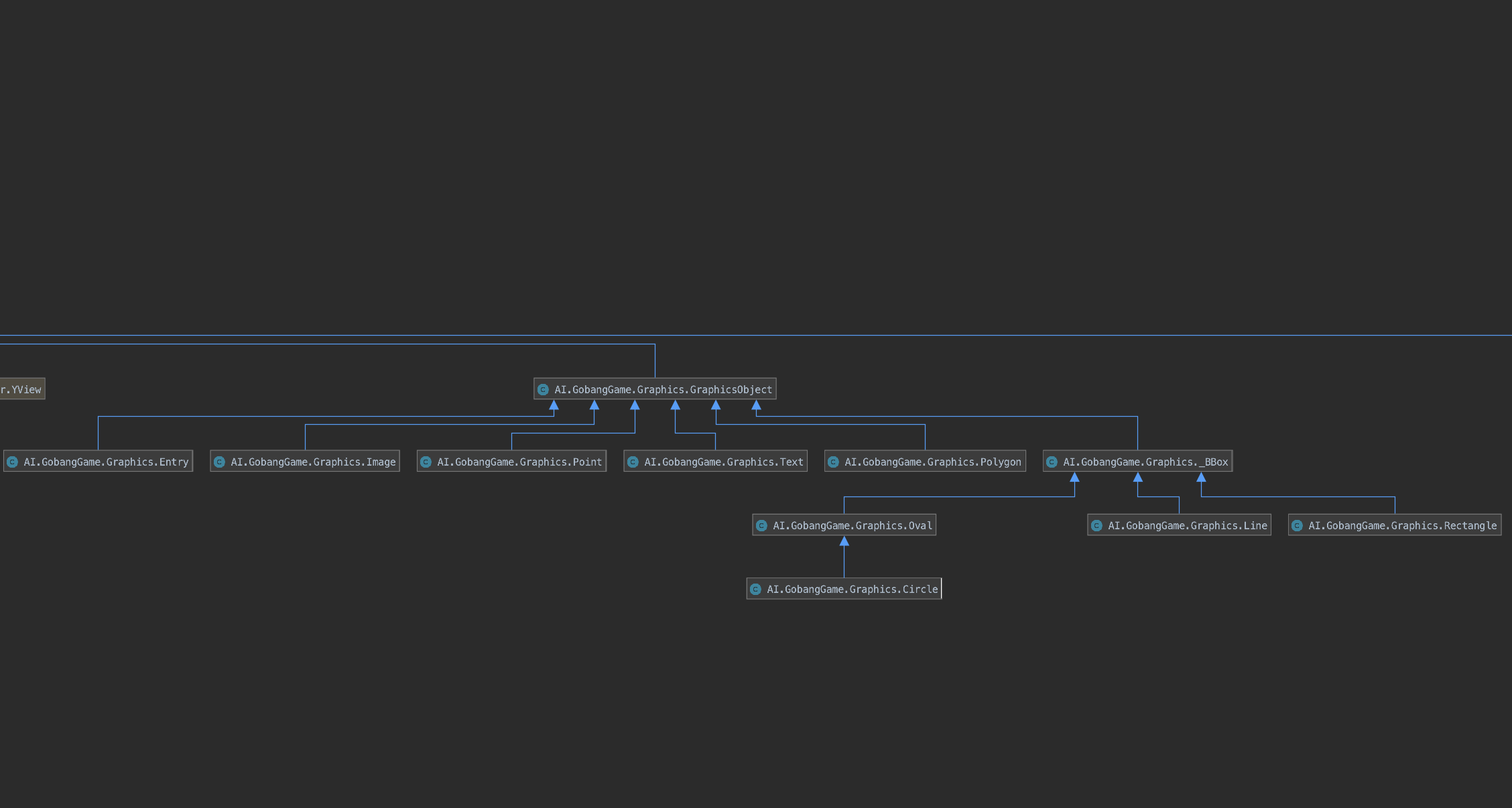
**IV. Debug Guidance**

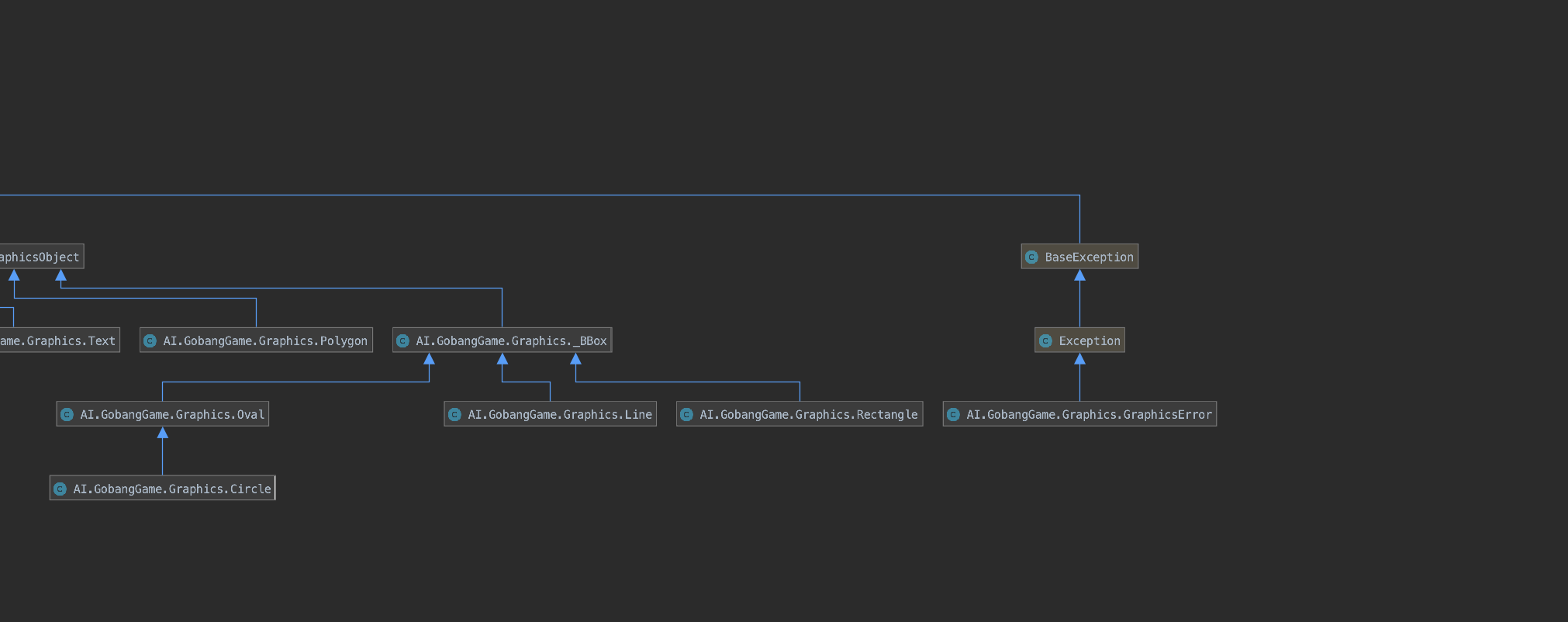
Here is what I have done in this paper and all the work is based on the Ai-Studio Platform and I’ve already reformat the structure of the code style. Even though I was hoping to implement Gomoku-Game with Coevolutionary Neural Network (CNN), but you know time is limited, I have to spent more time on Graduate Unified Examination at the end of the year 2019.

Here is the class hierarchy:









**V. Summary**

**Learning AI is great opportunity for us recognize the new model and digital world, which got me feeling like it’s the crazy thing in the world. In the early period of last AI Course, I’ve learned “Dive into deep learning” written by Mu Li, that was a great book which brought me into the Artificial Intelligence World. And I feel grateful to re-implement some model by myself.**

**由于自己考研原因，其实很想用MXnet 深度学习框架再实现一个 AI 五子棋版本，时间确实比较紧张，而且由于我的macbook-pro 没有 Nvidia GPU，所以如果训练15 \* 15，通过self-play 至少要4000 局才能达到80%胜率。训练时间耗时挺长的。所以只做了Minimax Algorithm with alpha-beta-pruning 版本。**

**Reference:**

<https://github.com/junxiaosong/AlphaZero_Gomoku>

<https://aistudio.baidu.com/user/56655/65661/notebooks/65661.ipynb?redirects=1>

**VI. Source Code**

**Alpha-Beta-Algorithm:**

Gomoku.py

from math import \*

import numpy as np

from AI.GobangGame.Graphics import \*

# global variables here

GRID\_WIDTH = 50

COLUMN = 15

ROW = 15

BeanSize = 22

# Gomoku board states

list1 = [] # AI

list2 = [] # human

list3 = [] # all

list\_all = [] # all board states

next\_point = [0, 0] # next step for AI

ratio = 25 # offense-defense-ratio: offense if ratio > 0 else defense < 0

MAX\_SEARCH\_DEPTH = 2 # MAX\_SEARCH\_DEPTH of search

# give scores according to board layout

shape\_score = [(50, (0, 1, 1, 0, 0)),

(50, (0, 0, 1, 1, 0)),

(200, (1, 1, 0, 1, 0)),

(500, (0, 0, 1, 1, 1)),

(500, (1, 1, 1, 0, 0)),

(5000, (0, 1, 1, 1, 0)),

(5000, (0, 1, 0, 1, 1, 0)),

(5000, (0, 1, 1, 0, 1, 0)),

(5000, (1, 1, 1, 0, 1)),

(5000, (1, 1, 0, 1, 1)),

(5000, (1, 0, 1, 1, 1)),

(5000, (1, 1, 1, 1, 0)),

(5000, (0, 1, 1, 1, 1)),

(50000, (0, 1, 1, 1, 1, 0)),

(99999999, (1, 1, 1, 1, 1))]

def ai():

global cut\_count # total cut num for ai

cut\_count = 0

global search\_count # total search times for ai

search\_count = 0

# negative-max algorithm

negamax(True, MAX\_SEARCH\_DEPTH, -99999999, 99999999)

print("Total cut times: " + str(cut\_count))

print("Total search times: " + str(search\_count))

return next\_point[0], next\_point[1]

def negamax(is\_ai, MAX\_SEARCH\_DEPTH, alpha, beta):

# return evaluation value when the game reaches the end

if game\_win(list1) or game\_win(list2) or MAX\_SEARCH\_DEPTH == 0:

return evaluation(is\_ai)

# list\_all - list3 = {blank\_list}

blank\_list = list(set(list\_all).difference(set(list3)))

order(blank\_list)

# traverse all blank space

for next\_step in blank\_list:

global search\_count

search\_count += 1

# no action if no neighbor around

if not has\_neighbor(next\_step):

continue

# check which is gonna go...

if is\_ai:

list1.append(next\_step)

else:

list2.append(next\_step)

list3.append(next\_step)

# switch to the opponent mode

value = -negamax(not is\_ai, MAX\_SEARCH\_DEPTH - 1, -beta, -alpha)

if is\_ai:

list1.remove(next\_step)

else:

list2.remove(next\_step)

list3.remove(next\_step)

if value > alpha:

print(str(value) + " [alpha: " + str(alpha) + "beta:" + str(

beta) + ']')

print(list3)

# check if not step over max search MAX\_SEARCH\_DEPTH

if MAX\_SEARCH\_DEPTH == MAX\_SEARCH\_DEPTH:

next\_point[0] = next\_step[0]

next\_point[1] = next\_step[1]

# alpha-beta pruning

if value >= beta:

global cut\_count

cut\_count += 1

return beta

alpha = value

return alpha

# order and make a priority for all blank\_list to speed-up search efficiency

def order(blank\_list):

last\_pt = list3[-1]

for item in blank\_list:

for i in range(-1, 2):

for j in range(-1, 2):

if i == 0 and j == 0:

continue

if (last\_pt[0] + i, last\_pt[1] + j) in blank\_list:

blank\_list.remove((last\_pt[0] + i, last\_pt[1] + j))

blank\_list.insert(0, (last\_pt[0] + i, last\_pt[1] + j))

def has\_neighbor(pt):

for i in range(-1, 2):

for j in range(-1, 2):

if i == 0 and j == 0:

continue

if (pt[0] + i, pt[1] + j) in list3:

return True

return False

def evaluation(is\_ai):

total\_score = 0

if is\_ai:

my\_list = list1

enemy\_list = list2

else:

my\_list = list2

enemy\_list = list1

# calculate evaluation value

score\_all\_arr = []

my\_score = 0

for pt in my\_list:

m = pt[0]

n = pt[1]

my\_score += cal\_score(m, n, 0, 1, enemy\_list, my\_list, score\_all\_arr)

my\_score += cal\_score(m, n, 1, 0, enemy\_list, my\_list, score\_all\_arr)

my\_score += cal\_score(m, n, 1, 1, enemy\_list, my\_list, score\_all\_arr)

my\_score += cal\_score(m, n, -1, 1, enemy\_list, my\_list, score\_all\_arr)

# calculate evaluation value of opponent

score\_all\_arr\_enemy = []

enemy\_score = 0

for pt in enemy\_list:

m = pt[0]

n = pt[1]

enemy\_score += cal\_score(m, n, 0, 1, my\_list, enemy\_list,

score\_all\_arr\_enemy)

enemy\_score += cal\_score(m, n, 1, 0, my\_list, enemy\_list,

score\_all\_arr\_enemy)

enemy\_score += cal\_score(m, n, 1, 1, my\_list, enemy\_list,

score\_all\_arr\_enemy)

enemy\_score += cal\_score(m, n, -1, 1, my\_list, enemy\_list,

score\_all\_arr\_enemy)

# determine final score

total\_score = my\_score - enemy\_score \* ratio \* 0.1

return total\_score

#

def cal\_score(m, n, x\_direction, y\_direction, enemy\_list, my\_list, score\_all\_arr):

add\_score = 0

max\_score\_shape = (0, None)

for item in score\_all\_arr:

for pt in item[1]:

if m == pt[0] and n == pt[1] and x\_direction == item[2][0] \

and y\_direction == item[2][1]:

return 0

# check scores around last move

for offset in range(-5, 1):

# offset = -2

pos = []

for i in range(0, 6):

if (m + (i + offset) \* x\_direction, n + (i + offset) \* y\_direction) in\

enemy\_list:

pos.append(2)

elif (m + (i + offset) \* x\_direction,

n + (i + offset) \* y\_direction) in my\_list:

pos.append(1)

else:

pos.append(0)

tmp\_shap5 = (pos[0], pos[1], pos[2], pos[3], pos[4])

tmp\_shap6 = (pos[0], pos[1], pos[2], pos[3], pos[4], pos[5])

for (score, shape) in shape\_score:

if tmp\_shap5 == shape or tmp\_shap6 == shape:

if tmp\_shap5 == (1, 1, 1, 1, 1):

print('wwwwwwwwwwwwwwwwwwwwwwwwwww')

print('wwwwwwwwwwwwwwwwwwwwwwwwwww')

print()

if score > max\_score\_shape[0]:

max\_score\_shape = (score, (

(m + (0 + offset) \* x\_direction, n + (0 + offset) \* y\_direction),

(m + (1 + offset) \* x\_direction, n + (1 + offset) \* y\_direction),

(m + (2 + offset) \* x\_direction, n + (2 + offset) \* y\_direction),

(m + (3 + offset) \* x\_direction, n + (3 + offset) \* y\_direction),

(m + (4 + offset) \* x\_direction, n + (4 + offset) \* y\_direction)),

(x\_direction, y\_direction))

# combine two kind of scores in horizontal and vertical direction

if max\_score\_shape[1] is not None:

for item in score\_all\_arr:

for pt1 in item[1]:

for pt2 in max\_score\_shape[1]:

if pt1 == pt2 and max\_score\_shape[0] > 10 and item[0] > 10:

add\_score += item[0] + max\_score\_shape[0]

score\_all\_arr.append(max\_score\_shape)

return add\_score + max\_score\_shape[0]

# winning board states

def game\_win(list):

for m in range(COLUMN):

for n in range(ROW):

if n < ROW - 4 and (m, n) in list and (m, n + 1) in list and (

m, n + 2) in list and (

m, n + 3) in list and (m, n + 4) in list:

return True

elif m < ROW - 4 and (m, n) in list and (m + 1, n) in list and (

m + 2, n) in list and (

m + 3, n) in list and (m + 4, n) in list:

return True

elif m < ROW - 4 and n < ROW - 4 and (m, n) in list and (

m + 1, n + 1) in list and (

m + 2, n + 2) in list and (m + 3, n + 3) in list and (

m + 4, n + 4) in list:

return True

elif m < ROW - 4 and n > 3 and (m, n) in list and (

m + 1, n - 1) in list and (

m + 2, n - 2) in list and (m + 3, n - 3) in list and (

m + 4, n - 4) in list:

return True

return False

# define Gomoku Window

def gomokuWin():

win = GraphWin("Tiny Gomoku Game: ", GRID\_WIDTH \* COLUMN,

GRID\_WIDTH \* ROW)

win.setBackground("white")

# horizontalLine verticalLine

horizontalLine = 0

while horizontalLine <= GRID\_WIDTH \* COLUMN:

l = Line(Point(horizontalLine, 0), Point(horizontalLine, GRID\_WIDTH \* COLUMN))

l.draw(win)

horizontalLine = horizontalLine + GRID\_WIDTH

verticalLine = 0

while verticalLine <= GRID\_WIDTH \* ROW:

l = Line(Point(0, verticalLine), Point(GRID\_WIDTH \* ROW, verticalLine))

l.draw(win)

verticalLine = verticalLine + GRID\_WIDTH

return win

def main():

startTime = time.time()

try:

win = gomokuWin()

for i in range(COLUMN + 1):

for j in range(ROW + 1):

list\_all.append((i, j))

change = 0

GAMEOVER = 0

m = 0

n = 0

# GAMEOVER == 1 game over

while GAMEOVER == 0:

# step alternatively

if change % 2 == 1:

# ai turn

pos = ai()

# duplicate position

if pos in list3:

message = Text(Point(200, 200),

"dessert position" + str(pos[0]) + "," + str(

pos[1]))

message.draw(win)

GAMEOVER = 1

list1.append(pos)

list3.append(pos)

''''

draw white node

'''

piece = Circle(Point(GRID\_WIDTH \* pos[0], GRID\_WIDTH \* pos[1]), BeanSize)

piece.setFill('white')

piece.draw(win)

if game\_win(list1):

message = Text(Point(100, 100), "white Win :)")

message.draw(win)

GAMEOVER = 1

change = change + 1

else:

# get the position of current user

p2 = win.getMouse()

if not ((round((p2.getX()) / GRID\_WIDTH),

round((p2.getY()) / GRID\_WIDTH)) in list3):

# get user move pos

a2 = round((p2.getX()) / GRID\_WIDTH)

b2 = round((p2.getY()) / GRID\_WIDTH)

list2.append((a2, b2))

list3.append((a2, b2))

piece = Circle(Point(GRID\_WIDTH \* a2, GRID\_WIDTH \* b2), BeanSize)

piece.setFill('black')

piece.draw(win)

if game\_win(list2):

message = Text(Point(100, 100), "Black Win (:")

message.draw(win)

GAMEOVER = 1

change = change + 1

message = Text(Point(100, 140), "Thanks for having fun with Gomoku ^-^.")

message.draw(win)

win.getMouse()

win.close()

except KeyboardInterrupt:

print("Bye")

endTime = time.time()

print("Total Time: {}s".format(endTime - startTime))

main()

**Graphics.py:**

# graphics.py

\_\_version\_\_ = "5.0"

import time

import os

import sys

try: # import as appropriate for 2.x vs. 3.x

import tkinter as tk

except:

import Tkinter as tk

##########################################################################

# Module Exceptions

class GraphicsError(Exception):

"""Generic error class for graphics module exceptions."""

pass

OBJ\_ALREADY\_DRAWN = "Object currently drawn"

UNSUPPORTED\_METHOD = "Object doesn't support operation"

BAD\_OPTION = "Illegal option value"

##########################################################################

# global variables and functions

\_root = tk.Tk()

\_root.withdraw()

\_update\_lasttime = time.time()

def update(rate=None):

global \_update\_lasttime

if rate:

now = time.time()

pauseLength = 1 / rate - (now - \_update\_lasttime)

if pauseLength > 0:

time.sleep(pauseLength)

\_update\_lasttime = now + pauseLength

else:

\_update\_lasttime = now

\_root.update()

############################################################################

# Graphics classes start here

class GraphWin(tk.Canvas):

"""A GraphWin is a toplevel window for displaying graphics."""

def \_\_init\_\_(self, title="Graphics Window",

width=200, height=200, autoflush=True):

assert type(title) == type(""), "Title must be a string"

master = tk.Toplevel(\_root)

master.protocol("WM\_DELETE\_WINDOW", self.close)

tk.Canvas.\_\_init\_\_(self, master, width=width, height=height,

highlightthickness=0, bd=0)

self.master.title(title)

self.pack()

master.resizable(0, 0)

self.foreground = "black"

self.items = []

self.mouseX = None

self.mouseY = None

self.bind("<Button-1>", self.\_onClick)

self.bind\_all("<Key>", self.\_onKey)

self.height = int(height)

self.width = int(width)

self.autoflush = autoflush

self.\_mouseCallback = None

self.trans = None

self.closed = False

master.lift()

self.lastKey = ""

if autoflush: \_root.update()

def \_\_repr\_\_(self):

if self.isClosed():

return "<Closed GraphWin>"

else:

return "GraphWin('{}', {}, {})".format(self.master.title(),

self.getWidth(),

self.getHeight())

def \_\_str\_\_(self):

return repr(self)

def \_\_checkOpen(self):

if self.closed:

raise GraphicsError("window is closed")

def \_onKey(self, evnt):

self.lastKey = evnt.keysym

def setBackground(self, color):

"""Set background color of the window"""

self.\_\_checkOpen()

self.config(bg=color)

self.\_\_autoflush()

def setCoords(self, x1, y1, x2, y2):

"""Set coordinates of window to run from (x1,y1) in the

lower-left corner to (x2,y2) in the upper-right corner."""

self.trans = Transform(self.width, self.height, x1, y1, x2, y2)

self.redraw()

def close(self):

"""Close the window"""

if self.closed: return

self.closed = True

self.master.destroy()

self.\_\_autoflush()

def isClosed(self):

return self.closed

def isOpen(self):

return not self.closed

def \_\_autoflush(self):

if self.autoflush:

\_root.update()

def plot(self, x, y, color="black"):

"""Set pixel (x,y) to the given color"""

self.\_\_checkOpen()

xs, ys = self.toScreen(x, y)

self.create\_line(xs, ys, xs + 1, ys, fill=color)

self.\_\_autoflush()

def plotPixel(self, x, y, color="black"):

"""Set pixel raw (independent of window coordinates) pixel

(x,y) to color"""

self.\_\_checkOpen()

self.create\_line(x, y, x + 1, y, fill=color)

self.\_\_autoflush()

def flush(self):

"""Update drawing to the window"""

self.\_\_checkOpen()

self.update\_idletasks()

def getMouse(self):

"""Wait for mouse click and return Point object representing

the click"""

self.update() # flush any prior clicks

self.mouseX = None

self.mouseY = None

while self.mouseX == None or self.mouseY == None:

self.update()

if self.isClosed(): raise GraphicsError("getMouse in closed window")

time.sleep(.1) # give up thread

x, y = self.toWorld(self.mouseX, self.mouseY)

self.mouseX = None

self.mouseY = None

return Point(x, y)

def checkMouse(self):

"""Return last mouse click or None if mouse has

not been clicked since last call"""

if self.isClosed():

raise GraphicsError("checkMouse in closed window")

self.update()

if self.mouseX != None and self.mouseY != None:

x, y = self.toWorld(self.mouseX, self.mouseY)

self.mouseX = None

self.mouseY = None

return Point(x, y)

else:

return None

def getKey(self):

"""Wait for user to press a key and return it as a string."""

self.lastKey = ""

while self.lastKey == "":

self.update()

if self.isClosed(): raise GraphicsError("getKey in closed window")

time.sleep(.1) # give up thread

key = self.lastKey

self.lastKey = ""

return key

def checkKey(self):

"""Return last key pressed or None if no key pressed since last call"""

if self.isClosed():

raise GraphicsError("checkKey in closed window")

self.update()

key = self.lastKey

self.lastKey = ""

return key

def getHeight(self):

"""Return the height of the window"""

return self.height

def getWidth(self):

"""Return the width of the window"""

return self.width

def toScreen(self, x, y):

trans = self.trans

if trans:

return self.trans.screen(x, y)

else:

return x, y

def toWorld(self, x, y):

trans = self.trans

if trans:

return self.trans.world(x, y)

else:

return x, y

def setMouseHandler(self, func):

self.\_mouseCallback = func

def \_onClick(self, e):

self.mouseX = e.x

self.mouseY = e.y

if self.\_mouseCallback:

self.\_mouseCallback(Point(e.x, e.y))

def addItem(self, item):

self.items.append(item)

def delItem(self, item):

self.items.remove(item)

def redraw(self):

for item in self.items[:]:

item.undraw()

item.draw(self)

self.update()

class Transform:

"""Internal class for 2-D coordinate transformations"""

def \_\_init\_\_(self, w, h, xlow, ylow, xhigh, yhigh):

# w, h are width and height of window

# (xlow,ylow) coordinates of lower-left [raw (0,h-1)]

# (xhigh,yhigh) coordinates of upper-right [raw (w-1,0)]

xspan = (xhigh - xlow)

yspan = (yhigh - ylow)

self.xbase = xlow

self.ybase = yhigh

self.xscale = xspan / float(w - 1)

self.yscale = yspan / float(h - 1)

def screen(self, x, y):

# Returns x,y in screen (actually window) coordinates

xs = (x - self.xbase) / self.xscale

ys = (self.ybase - y) / self.yscale

return int(xs + 0.5), int(ys + 0.5)

def world(self, xs, ys):

# Returns xs,ys in world coordinates

x = xs \* self.xscale + self.xbase

y = self.ybase - ys \* self.yscale

return x, y

# Default values for various item configuration options. Only a subset of

# keys may be present in the configuration dictionary for a given item

DEFAULT\_CONFIG = {"fill": "",

"outline": "black",

"width": "1",

"arrow": "none",

"text": "",

"justify": "center",

"font": ("helvetica", 12, "normal")}

class GraphicsObject:

"""Generic base class for all of the drawable objects"""

# A subclass of GraphicsObject should override \_draw and

# and \_move methods.

def \_\_init\_\_(self, options):

# options is a list of strings indicating which options are

# legal for this object.

# When an object is drawn, canvas is set to the GraphWin(canvas)

# object where it is drawn and id is the TK identifier of the

# drawn shape.

self.canvas = None

self.id = None

# config is the dictionary of configuration options for the widget.

config = {}

for option in options:

config[option] = DEFAULT\_CONFIG[option]

self.config = config

def setFill(self, color):

"""Set interior color to color"""

self.\_reconfig("fill", color)

def setOutline(self, color):

"""Set outline color to color"""

self.\_reconfig("outline", color)

def setWidth(self, width):

"""Set line weight to width"""

self.\_reconfig("width", width)

def draw(self, graphwin):

"""Draw the object in graphwin, which should be a GraphWin

object. A GraphicsObject may only be drawn into one

window. Raises an error if attempt made to draw an object that

is already visible."""

if self.canvas and not self.canvas.isClosed(): raise GraphicsError(

OBJ\_ALREADY\_DRAWN)

if graphwin.isClosed(): raise GraphicsError(

"Can't draw to closed window")

self.canvas = graphwin

self.id = self.\_draw(graphwin, self.config)

graphwin.addItem(self)

if graphwin.autoflush:

\_root.update()

return self

def undraw(self):

"""Undraw the object (i.e. hide it). Returns silently if the

object is not currently drawn."""

if not self.canvas: return

if not self.canvas.isClosed():

self.canvas.delete(self.id)

self.canvas.delItem(self)

if self.canvas.autoflush:

\_root.update()

self.canvas = None

self.id = None

def move(self, dx, dy):

"""move object dx units in x direction and dy units in y

direction"""

self.\_move(dx, dy)

canvas = self.canvas

if canvas and not canvas.isClosed():

trans = canvas.trans

if trans:

x = dx / trans.xscale

y = -dy / trans.yscale

else:

x = dx

y = dy

self.canvas.move(self.id, x, y)

if canvas.autoflush:

\_root.update()

def \_reconfig(self, option, setting):

# Internal method for changing configuration of the object

# Raises an error if the option does not exist in the config

# dictionary for this object

if option not in self.config:

raise GraphicsError(UNSUPPORTED\_METHOD)

options = self.config

options[option] = setting

if self.canvas and not self.canvas.isClosed():

self.canvas.itemconfig(self.id, options)

if self.canvas.autoflush:

\_root.update()

def \_draw(self, canvas, options):

"""draws appropriate figure on canvas with options provided

Returns Tk id of item drawn"""

pass # must override in subclass

def \_move(self, dx, dy):

"""updates internal state of object to move it dx,dy units"""

pass # must override in subclass

class Point(GraphicsObject):

def \_\_init\_\_(self, x, y):

GraphicsObject.\_\_init\_\_(self, ["outline", "fill"])

self.setFill = self.setOutline

self.x = float(x)

self.y = float(y)

def \_\_repr\_\_(self):

return "Point({}, {})".format(self.x, self.y)

def \_draw(self, canvas, options):

x, y = canvas.toScreen(self.x, self.y)

return canvas.create\_rectangle(x, y, x + 1, y + 1, options)

def \_move(self, dx, dy):

self.x = self.x + dx

self.y = self.y + dy

def clone(self):

other = Point(self.x, self.y)

other.config = self.config.copy()

return other

def getX(self): return self.x

def getY(self): return self.y

class \_BBox(GraphicsObject):

# Internal base class for objects represented by bounding box

# (opposite corners) Line segment is a degenerate case.

def \_\_init\_\_(self, p1, p2, options=["outline", "width", "fill"]):

GraphicsObject.\_\_init\_\_(self, options)

self.p1 = p1.clone()

self.p2 = p2.clone()

def \_move(self, dx, dy):

self.p1.x = self.p1.x + dx

self.p1.y = self.p1.y + dy

self.p2.x = self.p2.x + dx

self.p2.y = self.p2.y + dy

def getP1(self): return self.p1.clone()

def getP2(self): return self.p2.clone()

def getCenter(self):

p1 = self.p1

p2 = self.p2

return Point((p1.x + p2.x) / 2.0, (p1.y + p2.y) / 2.0)

class Rectangle(\_BBox):

def \_\_init\_\_(self, p1, p2):

\_BBox.\_\_init\_\_(self, p1, p2)

def \_\_repr\_\_(self):

return "Rectangle({}, {})".format(str(self.p1), str(self.p2))

def \_draw(self, canvas, options):

p1 = self.p1

p2 = self.p2

x1, y1 = canvas.toScreen(p1.x, p1.y)

x2, y2 = canvas.toScreen(p2.x, p2.y)

return canvas.create\_rectangle(x1, y1, x2, y2, options)

def clone(self):

other = Rectangle(self.p1, self.p2)

other.config = self.config.copy()

return other

class Oval(\_BBox):

def \_\_init\_\_(self, p1, p2):

\_BBox.\_\_init\_\_(self, p1, p2)

def \_\_repr\_\_(self):

return "Oval({}, {})".format(str(self.p1), str(self.p2))

def clone(self):

other = Oval(self.p1, self.p2)

other.config = self.config.copy()

return other

def \_draw(self, canvas, options):

p1 = self.p1

p2 = self.p2

x1, y1 = canvas.toScreen(p1.x, p1.y)

x2, y2 = canvas.toScreen(p2.x, p2.y)

return canvas.create\_oval(x1, y1, x2, y2, options)

class Circle(Oval):

def \_\_init\_\_(self, center, radius):

p1 = Point(center.x - radius, center.y - radius)

p2 = Point(center.x + radius, center.y + radius)

Oval.\_\_init\_\_(self, p1, p2)

self.radius = radius

def \_\_repr\_\_(self):

return "Circle({}, {})".format(str(self.getCenter()), str(self.radius))

def clone(self):

other = Circle(self.getCenter(), self.radius)

other.config = self.config.copy()

return other

def getRadius(self):

return self.radius

class Line(\_BBox):

def \_\_init\_\_(self, p1, p2):

\_BBox.\_\_init\_\_(self, p1, p2, ["arrow", "fill", "width"])

self.setFill(DEFAULT\_CONFIG['outline'])

self.setOutline = self.setFill

def \_\_repr\_\_(self):

return "Line({}, {})".format(str(self.p1), str(self.p2))

def clone(self):

other = Line(self.p1, self.p2)

other.config = self.config.copy()

return other

def \_draw(self, canvas, options):

p1 = self.p1

p2 = self.p2

x1, y1 = canvas.toScreen(p1.x, p1.y)

x2, y2 = canvas.toScreen(p2.x, p2.y)

return canvas.create\_line(x1, y1, x2, y2, options)

def setArrow(self, option):

if not option in ["first", "last", "both", "none"]:

raise GraphicsError(BAD\_OPTION)

self.\_reconfig("arrow", option)

class Polygon(GraphicsObject):

def \_\_init\_\_(self, \*points):

# if points passed as a list, extract it

if len(points) == 1 and type(points[0]) == type([]):

points = points[0]

self.points = list(map(Point.clone, points))

GraphicsObject.\_\_init\_\_(self, ["outline", "width", "fill"])

def \_\_repr\_\_(self):

return "Polygon" + str(tuple(p for p in self.points))

def clone(self):

other = Polygon(\*self.points)

other.config = self.config.copy()

return other

def getPoints(self):

return list(map(Point.clone, self.points))

def \_move(self, dx, dy):

for p in self.points:

p.move(dx, dy)

def \_draw(self, canvas, options):

args = [canvas]

for p in self.points:

x, y = canvas.toScreen(p.x, p.y)

args.append(x)

args.append(y)

args.append(options)

return GraphWin.create\_polygon(\*args)

class Text(GraphicsObject):

def \_\_init\_\_(self, p, text):

GraphicsObject.\_\_init\_\_(self, ["justify", "fill", "text", "font"])

self.setText(text)

self.anchor = p.clone()

self.setFill(DEFAULT\_CONFIG['outline'])

self.setOutline = self.setFill

def \_\_repr\_\_(self):

return "Text({}, '{}')".format(self.anchor, self.getText())

def \_draw(self, canvas, options):

p = self.anchor

x, y = canvas.toScreen(p.x, p.y)

return canvas.create\_text(x, y, options)

def \_move(self, dx, dy):

self.anchor.move(dx, dy)

def clone(self):

other = Text(self.anchor, self.config['text'])

other.config = self.config.copy()

return other

def setText(self, text):

self.\_reconfig("text", text)

def getText(self):

return self.config["text"]

def getAnchor(self):

return self.anchor.clone()

def setFace(self, face):

if face in ['helvetica', 'arial', 'courier', 'times roman']:

f, s, b = self.config['font']

self.\_reconfig("font", (face, s, b))

else:

raise GraphicsError(BAD\_OPTION)

def setSize(self, size):

if 5 <= size <= 36:

f, s, b = self.config['font']

self.\_reconfig("font", (f, size, b))

else:

raise GraphicsError(BAD\_OPTION)

def setStyle(self, style):

if style in ['bold', 'normal', 'italic', 'bold italic']:

f, s, b = self.config['font']

self.\_reconfig("font", (f, s, style))

else:

raise GraphicsError(BAD\_OPTION)

def setTextColor(self, color):

self.setFill(color)

class Entry(GraphicsObject):

def \_\_init\_\_(self, p, width):

GraphicsObject.\_\_init\_\_(self, [])

self.anchor = p.clone()

# print self.anchor

self.width = width

self.text = tk.StringVar(\_root)

self.text.set("")

self.fill = "gray"

self.color = "black"

self.font = DEFAULT\_CONFIG['font']

self.entry = None

def \_\_repr\_\_(self):

return "Entry({}, {})".format(self.anchor, self.width)

def \_draw(self, canvas, options):

p = self.anchor

x, y = canvas.toScreen(p.x, p.y)

frm = tk.Frame(canvas.master)

self.entry = tk.Entry(frm,

width=self.width,

textvariable=self.text,

bg=self.fill,

fg=self.color,

font=self.font)

self.entry.pack()

# self.setFill(self.fill)

self.entry.focus\_set()

return canvas.create\_window(x, y, window=frm)

def getText(self):

return self.text.get()

def \_move(self, dx, dy):

self.anchor.move(dx, dy)

def getAnchor(self):

return self.anchor.clone()

def clone(self):

other = Entry(self.anchor, self.width)

other.config = self.config.copy()

other.text = tk.StringVar()

other.text.set(self.text.get())

other.fill = self.fill

return other

def setText(self, t):

self.text.set(t)

def setFill(self, color):

self.fill = color

if self.entry:

self.entry.config(bg=color)

def \_setFontComponent(self, which, value):

font = list(self.font)

font[which] = value

self.font = tuple(font)

if self.entry:

self.entry.config(font=self.font)

def setFace(self, face):

if face in ['helvetica', 'arial', 'courier', 'times roman']:

self.\_setFontComponent(0, face)

else:

raise GraphicsError(BAD\_OPTION)

def setSize(self, size):

if 5 <= size <= 36:

self.\_setFontComponent(1, size)

else:

raise GraphicsError(BAD\_OPTION)

def setStyle(self, style):

if style in ['bold', 'normal', 'italic', 'bold italic']:

self.\_setFontComponent(2, style)

else:

raise GraphicsError(BAD\_OPTION)

def setTextColor(self, color):

self.color = color

if self.entry:

self.entry.config(fg=color)

class Image(GraphicsObject):

idCount = 0

imageCache = {} # tk photoimages go here to avoid GC while drawn

def \_\_init\_\_(self, p, \*pixmap):

GraphicsObject.\_\_init\_\_(self, [])

self.anchor = p.clone()

self.imageId = Image.idCount

Image.idCount = Image.idCount + 1

if len(pixmap) == 1: # file name provided

self.img = tk.PhotoImage(file=pixmap[0], master=\_root)

else: # width and height provided

width, height = pixmap

self.img = tk.PhotoImage(master=\_root, width=width, height=height)

def \_\_repr\_\_(self):

return "Image({}, {}, {})".format(self.anchor, self.getWidth(),

self.getHeight())

def \_draw(self, canvas, options):

p = self.anchor

x, y = canvas.toScreen(p.x, p.y)

self.imageCache[self.imageId] = self.img # save a reference

return canvas.create\_image(x, y, image=self.img)

def \_move(self, dx, dy):

self.anchor.move(dx, dy)

def undraw(self):

try:

del self.imageCache[self.imageId] # allow gc of tk photoimage

except KeyError:

pass

GraphicsObject.undraw(self)

def getAnchor(self):

return self.anchor.clone()

def clone(self):

other = Image(Point(0, 0), 0, 0)

other.img = self.img.copy()

other.anchor = self.anchor.clone()

other.config = self.config.copy()

return other

def getWidth(self):

"""Returns the width of the image in pixels"""

return self.img.width()

def getHeight(self):

"""Returns the height of the image in pixels"""

return self.img.height()

def getPixel(self, x, y):

"""Returns a list [r,g,b] with the RGB color values for pixel (x,y)

r,g,b are in range(256)

"""

value = self.img.get(x, y)

if type(value) == type(0):

return [value, value, value]

elif type(value) == type((0, 0, 0)):

return list(value)

else:

return list(map(int, value.split()))

def setPixel(self, x, y, color):

"""Sets pixel (x,y) to the given color

"""

self.img.put("{" + color + "}", (x, y))

def save(self, filename):

"""Saves the pixmap image to filename.

The format for the save image is determined from the filname extension.

"""

path, name = os.path.split(filename)

ext = name.split(".")[-1]

self.img.write(filename, format=ext)

def color\_rgb(r, g, b):

"""r,g,b are intensities of red, green, and blue in range(256)

Returns color specifier string for the resulting color"""

return "#%02x%02x%02x" % (r, g, b)

# MacOS fix 2

# tk.Toplevel(\_root).destroy()

# MacOS fix 1

update()