Project 1 – AI Algorithm for Gomoku

1 Description

Gomoku is a relatively simple board game. Its instrument is universal with Go, and it originated from one of the ancient Chinese black and white chess. Usually, the two sides use black and white chess pieces respectively, and at the intersection of the straight line and the horizontal line of the board, the next step is to form a five-member line to win.

In this assignment, we use the default board of size 15*15 board (administrators can modify the settings as needed). Students need to implement the AI algorithm of Gomoku according to the interface requirements and submit it to the system as required.

2 Evaluation Rule

The assessment is divided into 2 phases:

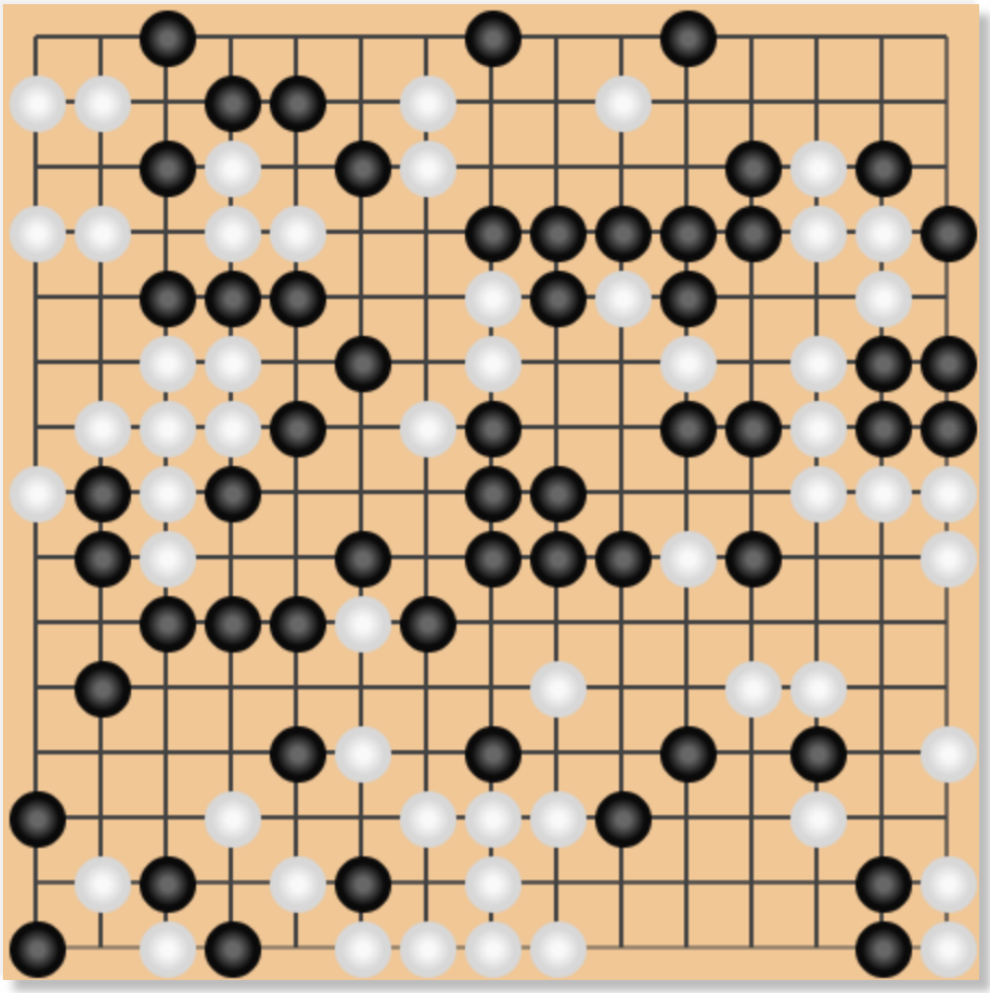
Usability testing: In this test, we will use some simple board test cases where students need to find the best place to drop. Only jobs that pass the usability test can pass.

Scoring stage: Students who pass the usability test can participate in the points game (积分赛). The specific competition rules are as follows: Students can submit their AI algorithm to the Gomoku battle platform (it is a successful submission if it passes the usability test). After a successful submission, select the player PK who ranks ahead of itself. The score is increased by 10 if it wins. The score remains unchanged if it is a tie. The score is reduced by 10 points if it loses. To avoid the first-hand advantage, each time the student should play against PK 2 innings with each one staring the game first. If they win twice, or win once and draw once, they win; if they win once and lose once, they draw; if they lose twice, or lose once and draw once, they lose.

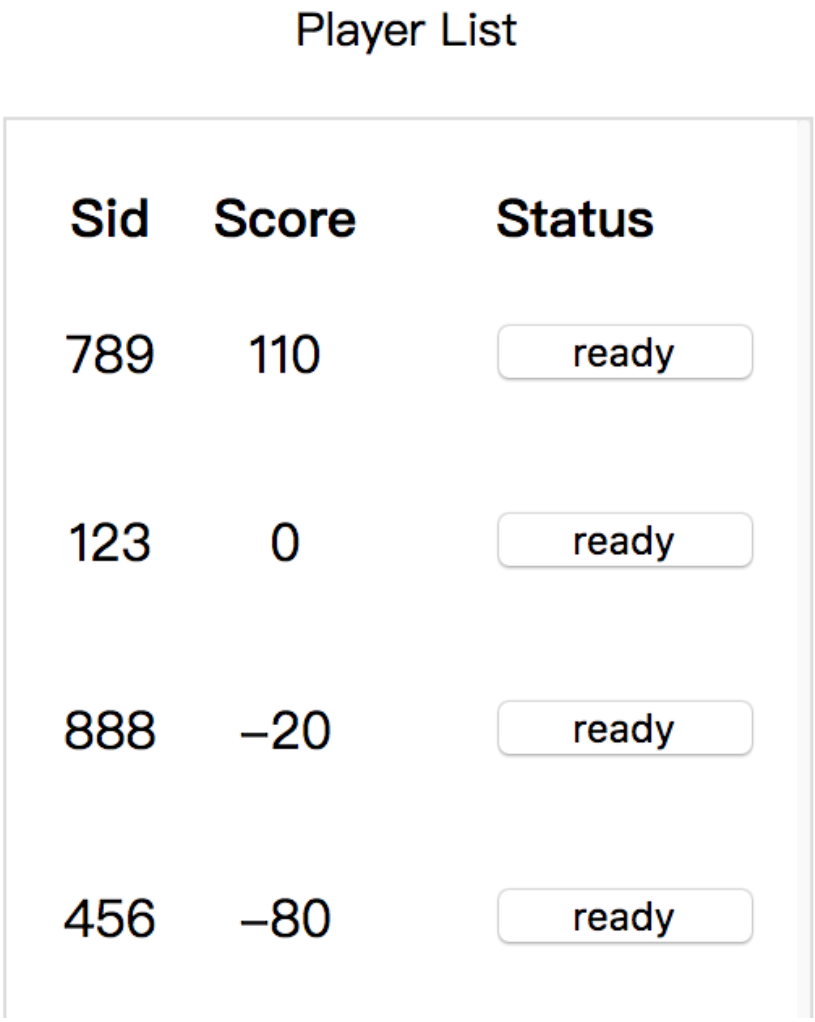
The final score is given according to the points.

The following game gives an example of the battle between two AI algorithms. Of course, this example is very simple and is the simulation result of two random AI algorithms.

Status： 游戏结束，123获胜！ Go Play



The score list is as follows



The Gomoku battle platform is still in the phase of developing internal tests, and there is still much room for improvement in all aspects. After the system is stable, we will open it to all students. At present, we are recruiting outstanding students to participate in development and testing. If you are interested in it, please send an email to zhaoy6@sustc.edu.cn

3 Code requirements

1、Python version: 3.6

2、Code template:

```
import numpy as np
import random
import time

COLOR_BLACK=-1
COLOR_WHITE=1
COLOR_NONE=0
random.seed(0)
#don't change the class name
class AI(object):
```

```
    #chessboard_size, color, time_out passed from agent
    def __init__(self, chessboard_size, color, time_out):
        self.chessboard_size = chessboard_size
        #You are white or black
        self.color = color
        #the max time you should use, your algorithm's run time must not exceed the time limit.
        self.time_out = time_out
        # You need add your decision into your candidate_list. System will get the end of your candidate_list as your decision .
        self.candidate_list = []

    # If your are the first, this function will be used.
    def first_chess(self):
        assert self.color == COLOR_BLACK
        self.candidate_list.clear()
        #==============================================================
        #Here you can put your first piece
        #for example, you can put your piece on sun（天元）
        self.candidate_list.append((self.chessboard_size//2,self.chessboard_size//2))
        self.chessboard[self.candidate_list[-1][0], self.candidate_list[-1][0]] = self.color

    # The input is current chessboard.
    def go(self, chessboard):
        # Clear candidate_list
        self.candidate_list.clear()
        #==============================================================
        #To write your algorithm here
        #Here is the simplest sample:Random decision
        idx = np.where(chessboard == 0)
        idx = list(zip(idx[0], idx[1]))
        pos_idx = random.randint(0, len(idx)-1)
        new_pos = idx[pos_idx]
        #==============Find new pos========================================
        # Make sure that the position of your decision in chess board is empty.
        #If not, return error.
        assert chessboard[new_pos[0],new_pos[1]]==0
        #Add your decision into candidate_list, Records the chess board
        self.candidate_list.append(new_pos)
```

3、Time measurement

start = time.time()

… algorithm…

run_time = (time.time() - start)