# AI Algorithm for Reversed Reversi

## 1 Overall Description

Reversed Reversi is a relatively simple board game. Players take turns placing disks on the board with their assigned color facing up. During a play, any disks of the opponent's color that are in a straight line and bounded by the disk just placed and another disk of the current player's color are turned over to the current player's color. The object of the game is to have the **fewest discs** turned to display your color when the last playable empty square is filled.

In this assignment, we use the default board of size 8*8 board (administrators can modify the settings as needed). Students need to implement the AI algorithm of Reversed Reversi according to the interface requirements and submit it to the system as required for usability testing and round-robin.

The Project is divided into two phases, each with one or two weeks for coding or improving your current coding.

| | Evaluation Rule | DeadLine |
|---|---|---|
| Lab1 | Score according to the number of test cases passed | 2022/07/05 Submit to platform |
| Lab2 | Score according to the ranking in the round robin | 2022/07/15 Submit to platform |
| Reversed Reversi AI report | | 2022/07/17 Submit to sakai |

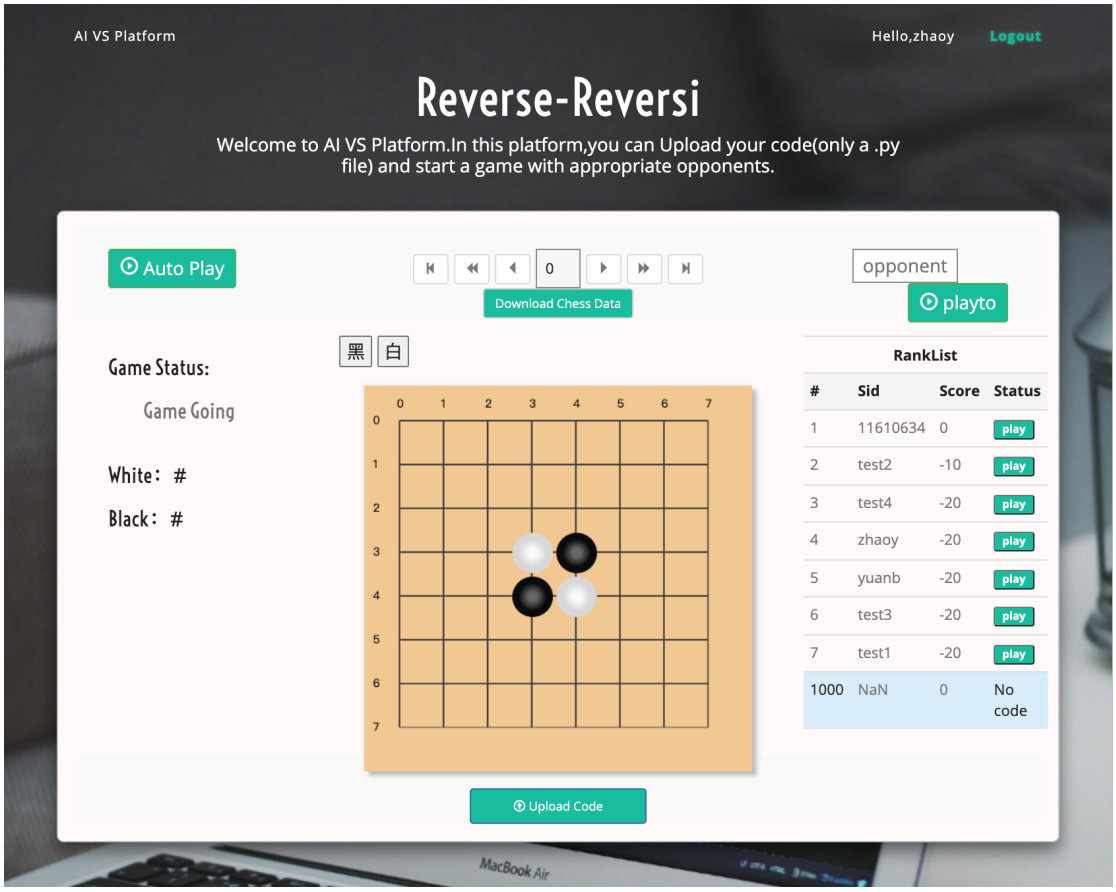
After Lab2, a carefully-written experiment report needs to be submitted.

Coding_Score =（Lab1+Lab2）/2

Project1_Score = Coding_Score *0.7+ Report_Score *0.3

## 2 The Use of the Platform

The Reversed Reversi platform http://10.20.18.54:8080/ is logged in with the student ID and password. The default password is the student ID. At the first login, the system will remind you to change the password. After logging in successfully, you can see the below interface.




Now, you can submit your own AI algorithm to the Reversed Reversi battle platform. The platform will first do usability test after submission. The submission is successful only if your AI algorithm pass the usability test. After your AI algorithm is submitted successfully, you can click on “Auto Play” to challenge the player who ranks ahead of you for a PK. In order to avoid the first-hand advantage, the PK is played for 2 rounds with each other starting the game first. In each round, if one of your wins, then its score is increased by 5 points. If one of you loses, then its score is reduced by 5 points. If it is a draw, the scores remain unchanged.

The whole battle process will be recorded for each match. You can play back the game you just experienced by playing and backing buttons, or you can download the game data as a text file so that you can debug or review their code and algorithms. On the right side you can see the leaderboards, showing only the top ten and their points, as well as your ranking and points. Click on “play” to see the live broadcast for the battle of the top-ten players. If you want to evaluate your newly uploaded algorithm, please use the function of “playto”. Just input the opponent you want to challenge and

click “playto”, then you can play a pre-PK with this opponent. The pre-PK does not affect your rankings.

**Your ranking on the platform does not affect your final score. The platform is just a tool to help you improve your algorithm.**

After the Lab2 DDL, the round-robin is started. Every student will play a PK with each other student. The score is accumulated and the ranking for each student is given according to the final points.

# 3 Code Requirements

1、Python version: 3.7.6

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 = []


    # The input is current chessboard.
    def go(self, chessboard):
        # Clear candidate_list, must do this step
        self.candidate_list.clear()
        #==============================================================
        #Write your algorithm here
        #Here is the simplest sample:Random decision
        idx = np.where(chessboard == COLOR_NONE)
        idx = list(zip(idx[0], idx[1]))
        #==============Find new pos=======================================
        # Make sure that the position of your decision in chess board is empty.
        # If not, the system will return error.
        # Add your decision into candidate_list, Records the chess board
        # You need add all the positions which is valid
        # candidate_list example: [(3,3),(4,4)]
```

```
37          # You need append your decision at the end of the candidate_list,
38          # we will pickthe last element of the candidate_list as the position you choose
39           # If there is no valid position, you must return an empty list.
40
```

1、Time measurement

start = time.time()

… algorithm…

run_time = (time.time() - start)

2、Note：**import os** not allowed to use

3、The use of memory cannot go beyond 100M, the time to find a place to drop cannot be longer than 5s, the whole battle cannot be longer than 180s

# 4 Requirements in Each Phase

## 4.1 Lab1

**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. This is to prevent students from submitting code including illegal operations, infinite loop or random drop. The usability testing includes: the os package cannot be imported in the code file to prevent the student code from destroying the operation of the system.

A total of 10 test cases were prepared, each with 10 points, and the scores were determined by the number of test cases passed.

## 4.2 Lab2

Students who pass the usability test can use the platform to improve the algorithm. The specific competition rules of the platform are as follows: Students can submit their AI algorithm to the Reversed Reversi battle platform (it is a successful submission if it passes the usability test). After a successful submission, you can click on “Auto Play” to challenge the player who ranks ahead of you for a PK. In order to avoid the first-hand advantage, the PK is played for 2 rounds with each other starting the game first.

In each round, if one of you wins, then its score is increased by 5 points. If one of you loses, then its score is reduced by 5 points. If it is a draw, the scores remain unchanged.

Students can submit and update their AI algorithm to the Reversed Reversi battle platform before Lab 2 DDL. After the Lab2 DDL, the round-robin is started. Every student will play a PK with each other student. The score is accumulated and the ranking for each student is given according to the final points.