AI Algorithm for Reversed Reversi

October 16, 2025

Abstract

Reversed Reversi is a relatively simple board game. Players take turns placing chess pieces on the board with their assigned color facing up. During a play, any pieces of the opponent's color that are in a straight line and bounded by the piece just placed and another piece 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 pieces** 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 platform as required for the **usability tests** and **round-robin**.

For detailed rules and an interactive demo of standard Reversi, see https://cardgames.io/reversi/. Note: this project is "Reversed Reversi" — the **objective is inverted** (fewer discs of your color at the end is better), while the **move legality and flipping rules** are the same as standard Reversi.

1 Schedule

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

	Evaluation Rule	DeadLine
Phase 1	Score according to the number of test cases passed.	**2025/11/02**
Phase 2	Score according to the rank in a round-robin.	**2025/11/16**

Score = Phase1+Phase2 (Parse1:Parse2=1:1)

2 How to Use The Platform

The Reversed Reversi platform is logged in with your student ID and password. The default password is your student ID. The platform will ask you to change your password when you first log in. After logging in successfully, you can see the page below.

Now, you can submit your AI algorithm to the Reversed Reversi battle platform. The platform will first do the usability tests after submission. The submission is successful only if

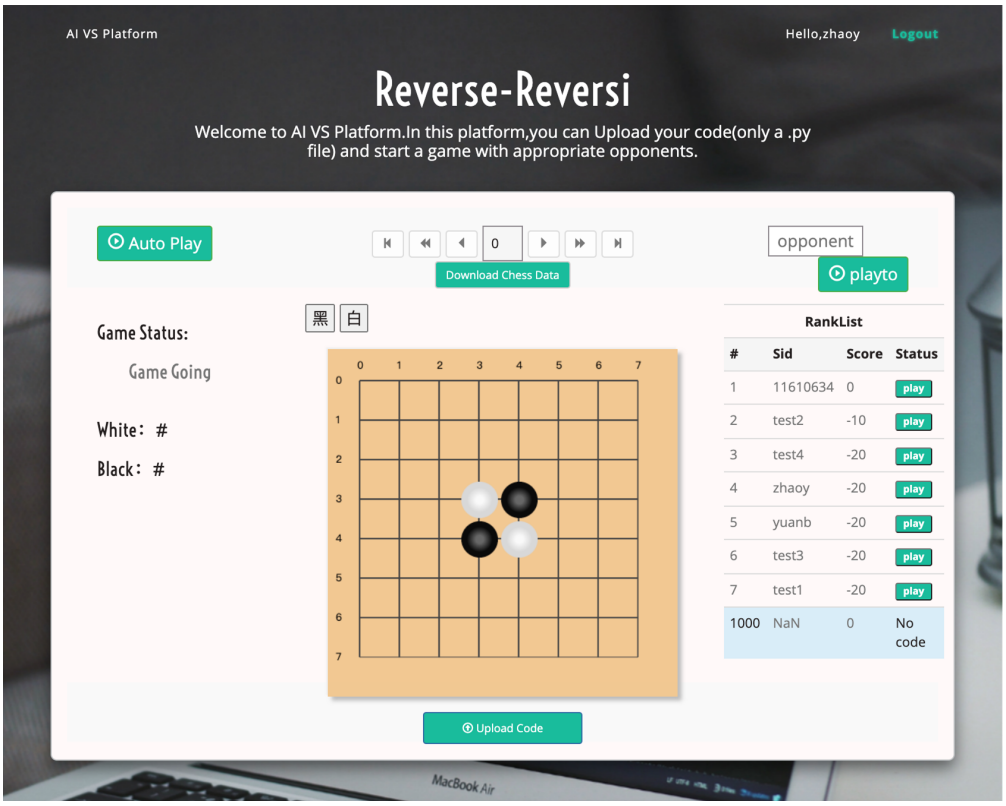

Figure 1: Home page

your AI algorithm passes the usability tests. After submitting your AI algorithm successfully, you can click on "Auto Play" to challenge the players who rank ahead of you for a PK. 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, his (or her) score is increased by 5 points. If one of you loses, then his (or her) score is reduced by 5 points. If a draw occurs, both of your 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 download the game data as a text file to debug or review their code and algorithms. On the right side, you can see the leaderboards, showing only the top ten and their points and 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's Student ID you want to challenge and click "playto", then you can play a pre-PK with this opponent. **This pre-PK will 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 Phase 2 DDL, the round-robin is started. Every student will play a PK with each other students. The score is accumulated, and each student ranks according to the final points.

3 Code Requirements

1. Version Details
   (a) Python version: 3.7.6
   (b) Numpy: 1.21.0
   (c) Numba: 0.56.4
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 to add your decision to your candidate_list.
    # The system will get the end of your candidate_list as your decision.
    self.candidate_list = []

# The input is the current chessboard. Chessboard is a numpy array.
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 on the chess board is empty.
    # If not, the system will return error.
    # Add your decision into candidate_list, Records the chessboard
    # You need to add all the positions which are valid
```

```
    # candidate_list example: [(3,3),(4,4)]
    # You need append your decision at the end of the candidate_list,
    # candidate_list example: [(3,3),(4,4),(4,4)]
    # we will pick the last element of the candidate_list
    # as the position you choose.
    # In above example, we will pick (4,4) as your decision.
    # If there is no valid position, you must return an empty list.
```

3. Time measurement

```
start = time.time()
# Your algorithm
run_time = (time.time() - start)
```

4. Note: `import os` is not allowed to use.

5. The usage of memory cannot go more than 100M, the time to find a place to drop cannot be longer than 5s, the whole battle cannot be longer than 180s.

4 Requirements for Each Phase

4.1 Phase 1

Usability tests: In this test, we will use simple board cases. Your AI must return the *complete set of legal moves* in `candidate_list` as `(row, col)` tuples. Elements must be tuples, and if there is no legal move, return an empty `candidate_list`. The platform uses the last element of `candidate_list` as your actual move only during live matches. This prevents submissions with illegal operations, infinite loops, or random drops. The usability tests include: the `os` package cannot be imported to prevent student code from affecting the platform.

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 Phase 2

Students who pass the usability tests 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 tests). After a successful submission, you can click "Auto Play" to challenge the player who ranks ahead of you for a PK. To avoid the first-hand advantage, the PK is played for two rounds, with each one starting the game first. In each round, if one of you wins, his (or her) score is increased by 5 points. If one of you loses, his (or her) score is reduced by 5 points. If a draw occurs, both of your scores remain unchanged.

Students can submit and update their AI algorithm to the Reversed Reversi battle platform before Phase 2 DDL. After the Phase 2 DDL, the round-robin is started. Every student

will play a PK with each other. The score is accumulated, and the ranking of each student is given according to the final points.