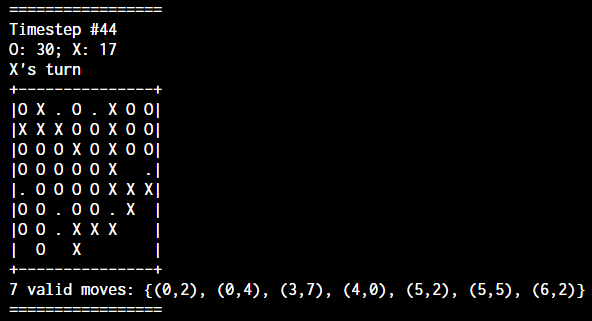
**README.md**

**Othello (Reversi) AI Template for I2P(2)\_2020\_SR**

We will use Othello AI in mini-project 3.

[](https://github.com/j3soon/OthelloAITemplate/blob/master/docs/imgs/preview.png)

Note that the . in the screenshot indicates the valid moves for the current player.

**Compile**

If you have make installed:

make

Otherwise: (Assuming g++ is installed)

g++ -Wall -Wextra --std=c++14 -o main main.cpp

g++ -Wall -Wextra --std=c++14 -o player\_random player\_random.cpp

g++ -Wall -Wextra --std=c++14 -o player\_infinite player\_infinite.cpp

g++ -Wall -Wextra --std=c++14 -o player\_partial player\_partial.cpp

g++ -Wall -Wextra --std=c++14 -o player\_invalid player\_invalid.cpp

Alternatively, you can compile them one by one through IDEs.

**Game Manager**

The code ([main.cpp](https://github.com/j3soon/OthelloAITemplate/blob/master/main.cpp)) contains the game logic of Othello. It should work in Linux, Windows, and MacOS.

You can change the constant timeout in the code to a larger number to increase the max calculation time for each action.

Note that this code isn't used by the TAs to grade your program; this is a simplified version for easy testing.

Syntax:

# In Linux / Mac

./main ./<player1> ./<player2>

# Windows

main <player1>.exe <player2>.exe

Example:

In Terminal / cmd:

# In Linux / Mac

./main ./player\_random ./player\_infinite

# In Windows

main player\_random.exe player\_infinite.exe

**Mac Only**:

Install coreutils with homebrew.

brew install coreutils

If you do not have homebrew installed, follow the [installation guide](https://docs.brew.sh/Installation).

**Players**

Take the random agent ([player\_random.cpp](https://github.com/j3soon/OthelloAITemplate/blob/master/player_random.cpp)) for the starting template. You can simply modify the function write\_valid\_spot to change the A.I.'s behavior.

* [player\_random.cpp](https://github.com/j3soon/OthelloAITemplate/blob/master/player_random.cpp) is a simple A.I. that randomly picks a action from all valid actions.
* [player\_infinite.cpp](https://github.com/j3soon/OthelloAITemplate/blob/master/player_infinite.cpp) shows that your program can keep output better actions by flushing the file stream.
* [player\_partial.cpp](https://github.com/j3soon/OthelloAITemplate/blob/master/player_partial.cpp) represents the situation that if your program is terminated during output, the last successful action is picked.
* [player\_invalid.cpp](https://github.com/j3soon/OthelloAITemplate/blob/master/player_invalid.cpp) shows that if you output an invalid action, or do not output an action, or your program crashes, you lose immediately.

**Example Input**

2

0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0

0 1 2 2 2 0 0 0

0 0 1 2 1 0 0 0

0 0 2 1 1 0 0 0

0 0 0 1 1 0 0 0

0 0 0 0 1 0 0 0

0 0 0 0 0 0 0 0

9

2 0

3 1

3 5

4 1

4 5

5 5

6 3

7 4

7 5

The first number indicates which side you are playing. (1 for black, 2 for white)

The first 8\*8 numbers represent the board state:

* 0: empty space
* 1: black
* 2: white

The next number indicates N, the number of next valid moves of your player.

The next N lines contain the coordinates of the valid moves.

**Example Output**

2 0

Output a coordinate where you want to put your disc at.

2 0

3 1

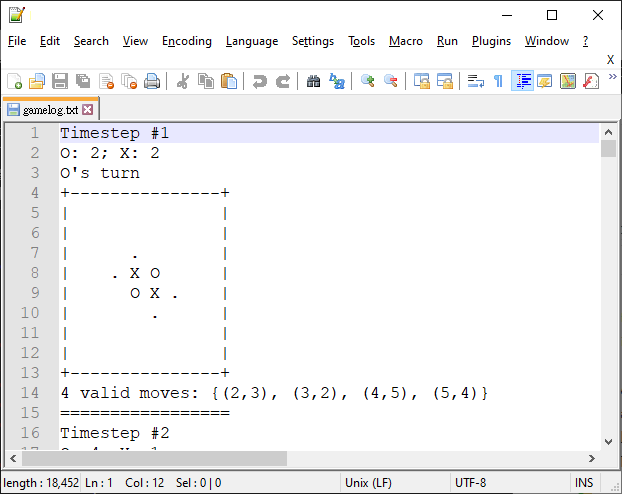
2

You can keep updating the new coordinate until your process is killed. Make sure the coordinates are flushed to the disk through ofstream.flush().

In the above sample, the last successful coordinate is (3,1).

**Log File**

After running the game manager, gamelog.txt is created.

[](https://github.com/j3soon/OthelloAITemplate/blob/master/docs/imgs/gamelog.png)

As shown as the screenshot above, you can disable WordWrap (View > Word wrap) in [Notepad++](https://notepad-plus-plus.org/) and set the height of the window to show about 16 lines. Then, you can inspect the game process by pressing PAGEUP/PAGEDOWN. (By setting the page height to 16 lines, moving through pages can cause the effect of Persistence of vision. So the game process looks like a simple animation with the ability to fast-forward/rewind.)

**FAQ**

1. Error occurred on Windows: ERROR: The process "./player\_random" not found.

This is because in Windows, launching a process main.exe in the same directory only requires main or main.exe, but not ./main.

You can refer to the commands below:

# In Linux / Mac

./main ./<player1> ./<player2>

# Windows

main <player1>.exe <player2>.exe

For WSL, although not tested, I think you should run the same command used in Linux.

1. Error occurred on Windows: ERROR: The process "player\_random.exe" not found.

This is because there is a bug in main.cpp on Windows platform, please re-download the project package. Sorry for the inconvenience.

1. What should I hand in?

You can see this project as an Online Judge problem. The input is the current board and valid moves, while the output is the predicted best move. So, you can simply copy player\_random.cpp or any other template, modify it to something like <Student\_ID>\_project3.cpp, and improve the AI based on the template. Then you can simply hand-in that file.

1. What is the Project Structure?

Since your code can be seen as a simple Online Judge program, the GameManager (main.cpp) stores the board state and other information and calls the 2 AI programs to receive the next move. So, you can simply modify the player\_random.cpp and ignore main.cpp. But I strongly encourage you to read main.cpp since it contains a lot of details on how this project works.

Calling 2 programs in turns instead of compiling the 2 programs together may seem unnecessary. However, when Runtime Error occurs, it is hard to tell which side is responsible for the crash if compiled together. Furthermore, to enable multiple outputs within a certain time and limit the memory usage requires threading (which requires different API on Windows/Mac/Linux, unless using an external library) or other techniques. So, we decided to simply separate the programs, and make it more like the Online Judge problem that you are familiar with.

1. How are the scores calculated?
   * AI Baselines: There are multiple hidden baselines. Your code will be compiled and tested on TA's work station, and the score depends on how many baselines you can defeat.
   * Tree Search: Refer to the Hackathon recording and search online for how to implement MiniMax or other methods.
   * State Value Function Design: Read the current board and output how good the board is by a number.
   * Alpha-beta Pruning: You can refer to Hackathon's recording or Wikipedia.
   * Version Control (Bonus): Try out GitHub, you can refer to the spec of this project.
   * Class Ranking (Bonus): We'll pick the student's code, which can defeat all AI baselines. Then make them compete with each other. The ranking is determined by how many opponents your AI can defeat.
2. Error occurred on Windows: Assertion failed: argc == 3

Either you compile the codes through g++ or IDE, you should put all executables in a same directory. Then you can launch the command below:

# In Linux / Mac

./main ./<player1> ./<player2>

# Windows

main <player1>.exe <player2>.exe

You should not rely on the run/debug feature of your IDE since they do not give arguments to main, which expects 2 arguments.

If you are using IDEs, you can compile/build your code and take the executable file located in the project directory. (The location of the executable file varies depending on which IDE you are using) Alternatively, you can run/debug your code and ignore the error message, then similarly take the compiled executable.