Mini-Checkers Game
Chia-Hsien Lin (chl566)

Introduction

The game is played on a reduced game board size of 6x6 squares and the rules have been modified from the original checkers game. Implement the AI player primary by using the alpha-beta search algorithm. We will discuss the detail about the algorithm and how to run my program in this report later.

Environment Setup

- Language: C++
- Operating System: macOS High Sierra version 10.13.1
- IDE: Qt Creator 5.11.0
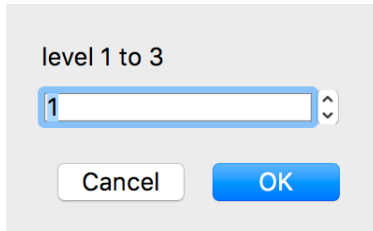- Qt/5.11.0/clang_64/bin/qmake

How to Build

- Install qt5-qmake
- In the project directory, execute command `/…/Qt/5.11.0/clang_64/bin/qmake .`
- You will see a Makefile, and execute command `make`
- You will see an application be generated
- Or you can download the executable file from https://github.com/chiahsienlin/CheckersGame/blob/master/Checker.zip
- If you download the executable file, decompress first
- Go to the System Preferences → Security & Privacy → Open anyway
- Now, you can run the game

Rule Description

- The checker board consists of 6 x 6 alternating light and dark squares. The board is placed so that the left corner square on each player's side is a dark square.
- There are two types of moves:
    - regular move: a piece can move forward diagonally to an adjacent square that is empty.
    - capture move: a piece can jump over and capture an opponent's piece and land on an empty square (landing on a square that is not empty is not allowed.) The jump must be in the forward diagonal direction and no consecutive jumps are allowed. In addition, every opportunity to jump must be taken. In the case where there are two or more possible jump moves, the player can choose which one to take.
    - No vertical, horizontal or backward moves areallowedforbothregularand capture moves.
- If a player has no legal move to take, his/her turn will be forfeited and the other player will make the next move.
- A player wins when he/she captures all of the other player's pieces. If both players do not have any legal move to take, the game will end and the player with the most number of pieces left wins; if the two players have the same number of pieces left, the game is a draw.

User Flow

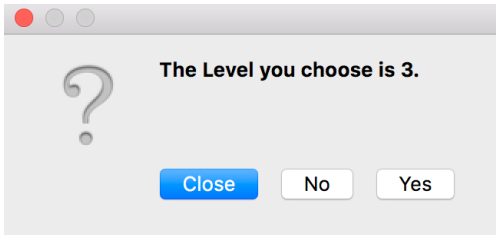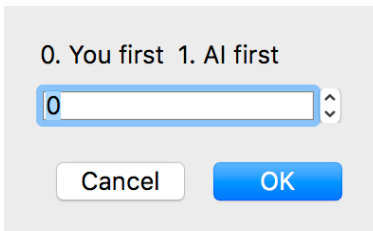At the begining, user can choose level from 1 - 3.

Then, the user can confirm whether it's the level he or she chose. If no, the user can re-input the level again. If yes, the user will have to choose who is going to move first.

Choose 0. You or 1. AI will move the first step.
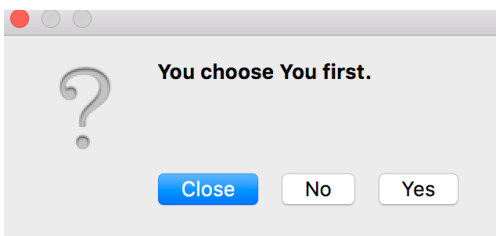

Just like above, the user can confirm if the choice he or she makes is correct.


As you can see on the user interface below, once the user presses the checkers piece that he or she wants to move, it will become orange. The two yellow blocks are the valid steps the particular checkers piece can choose. The timer is set for the AI because the searching time limitation is 15 seconds. The whiteboard on the right-hand side will record the AI status and the result of judge function which plays a judge in this game.
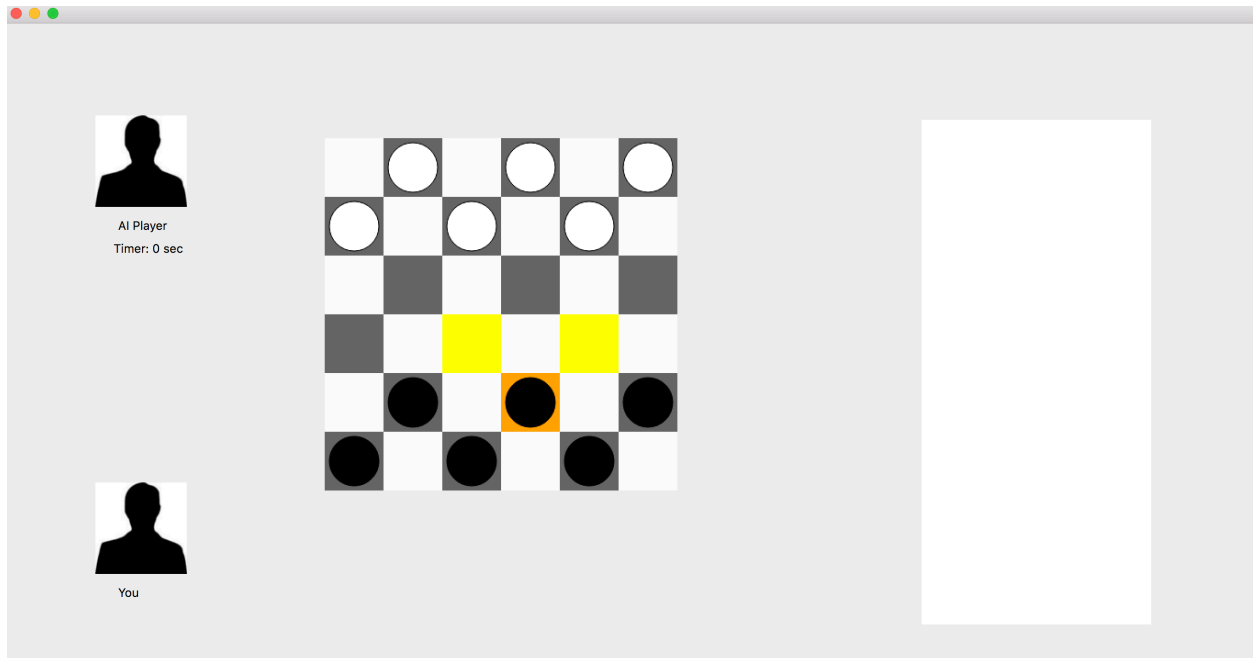
Once the user move the checkers piece the timer will count the time the AI spent since the user moves. The messege box will reveal the status of the AI player.


[AI] I'm done. It's your turn!
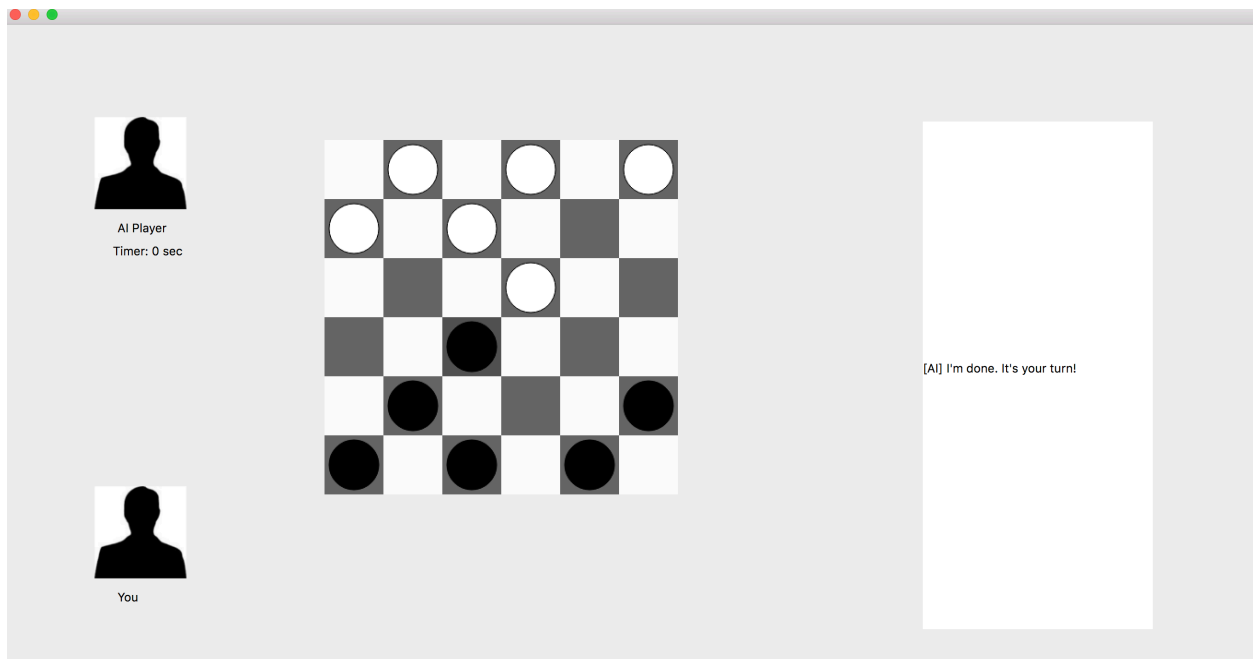
At the end of the game, the whiteboard will reveal the result of who is the winner.

[Judge] Game Over. AI WIN, you LOSE.
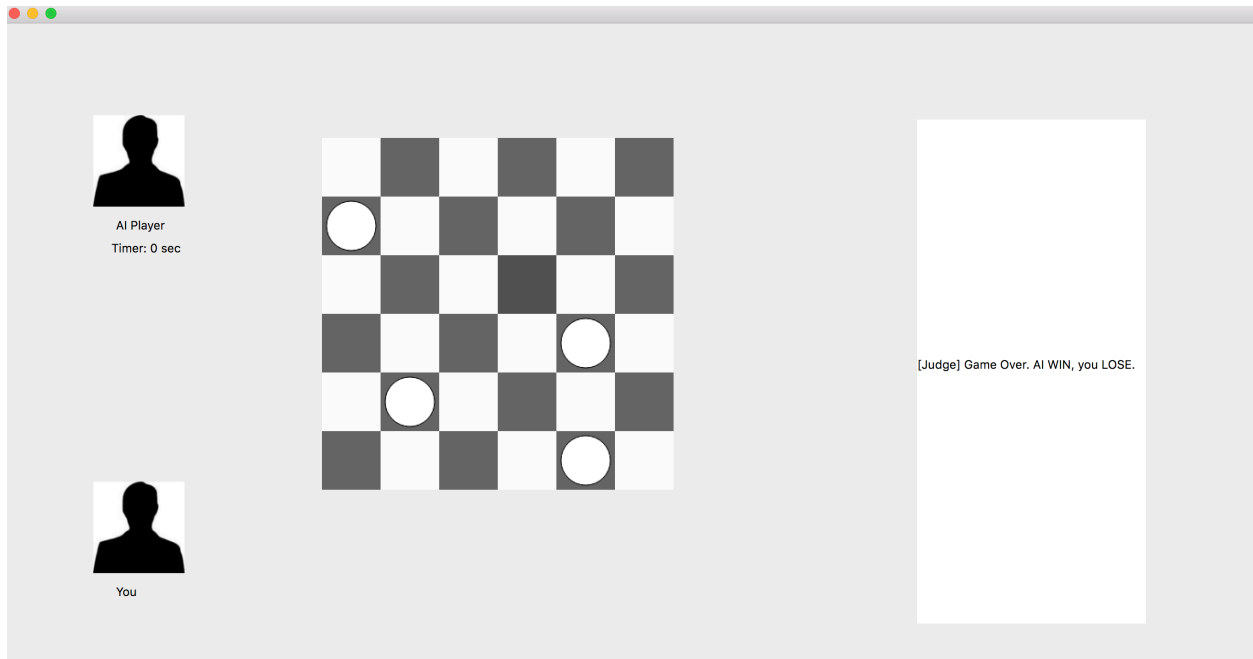
Evaluation Function

First of all, I scan whole 6x6 checkers board once to find the white boundary and the black boundary which is the last checkers piece behind all its pieces with the same color. You can see the example below to illustrate the boundary.

In the middle of the game, there will be some checkers pieces reach the opponent's boundary which means those pieces are safe because it is not possible eaten by opponent's pieces. I use the variable named safeWhite and safeBlack to count those safe pieces.
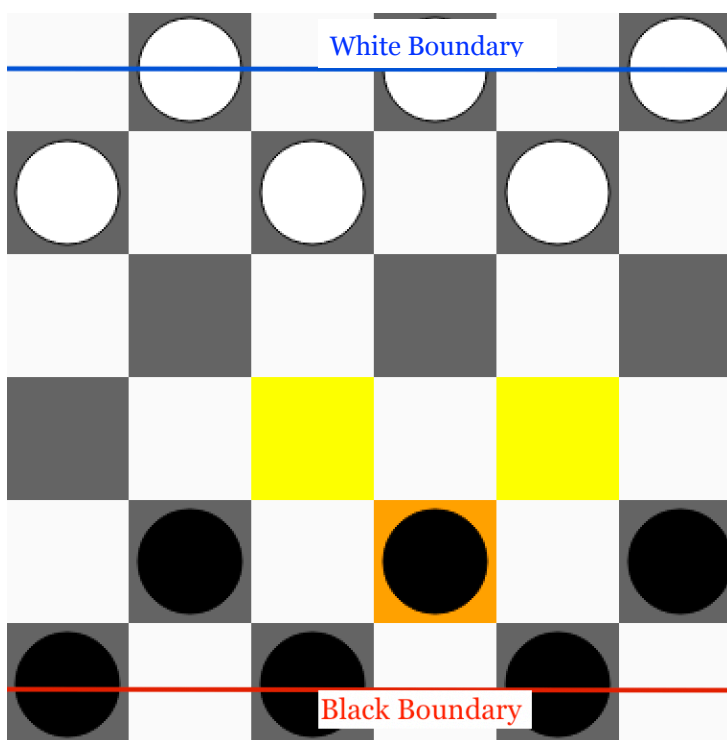
The checkers pieces on the four boundary of the board are also safe. Therefore, I also count those pieces with borderWhite and borderBlack variables.
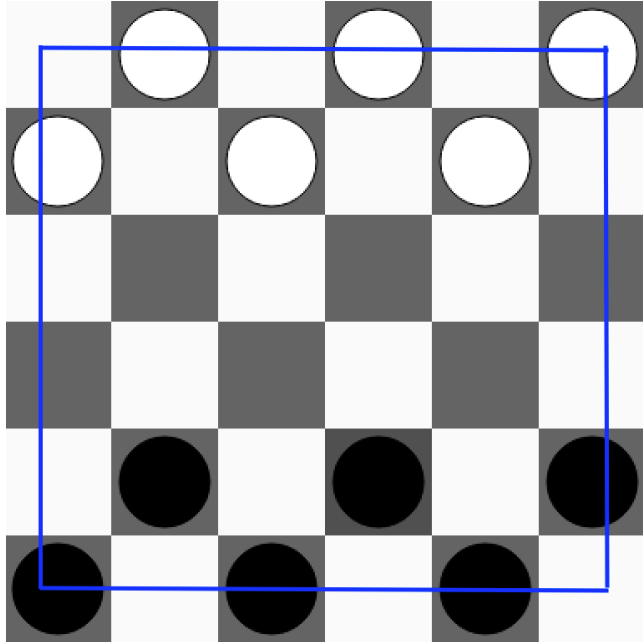
For the AI's checkers piece in the middle, it is safe in the scenario below. Therefore, I count them to safeWhite and safeBlack as well.
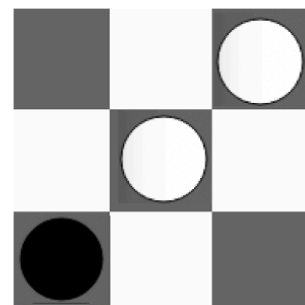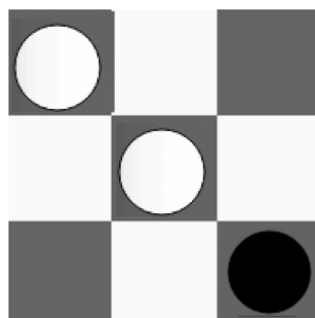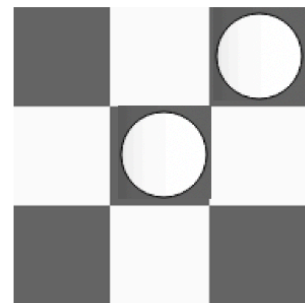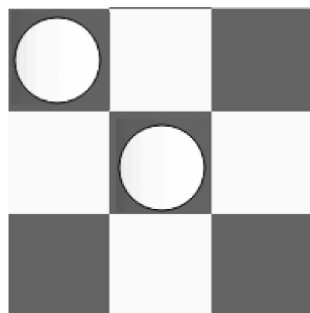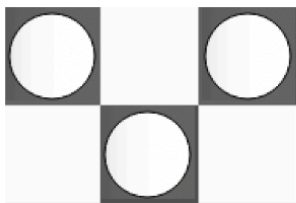
As you can see, the four scenarios below, the AI's checkers piece in the middle is safe and is able to eat the black checkers piece. I use the variable SafeEatBlack and SafeEatWhite to count these checkers pieces in the safe-to-eat positions.
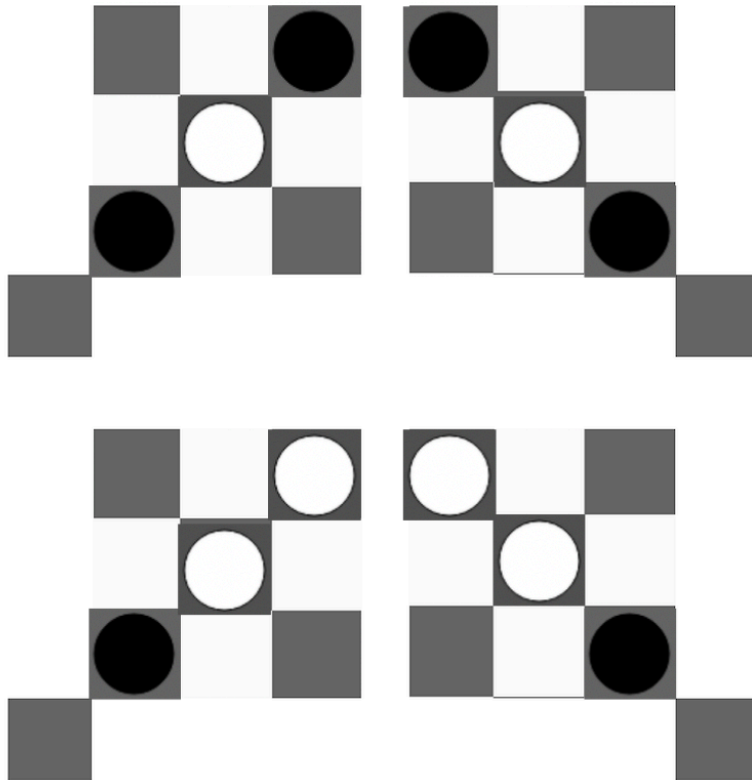
Besides, I use the variable named cntWhite and cntBlack count all the checkers pieces white or black on the checkers board.

Finally, I return the utility value with some priority weight

utility value = (cntWhite - cntBlack)*1000 + (safeWhite - safeBlack)*3000 + (SafeEatBlack - SafeEatWhite)*2000 + (borderWhite - borderBlack)*500;
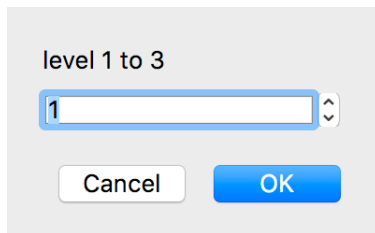
Utility Function

When the state is the terminal state, I will scan through whole checkers board and count the checkers pieces for each color and compare which color has the more checkers pieces. If it's the white checkers pieces more than the black ones and return the INT_MAX which is positive infinity. On the other hand, it will return the INT_MIN which is negative infinity.

Terminal States

I scan through whole the board once to check whether the number of black or white checkers piece is zero. Besides, for each step, we will check if the AI player or the human being is out of actions which means both of the players cannot move at all. If so, then I return it is terminate state. Then the JudgeFunction will decide the final winner depends on the number of the checkers pieces.

Levels

The user can choose the level of difficulty from 1-3 before the game starts. The difficulty of the game depends on the searching level I define. I define the difficulty = level*5+2. For instance, if the user chooses the level 3, and the depth of the searching limitation will be 17.

level 1 to 3

1

Cancel   OK