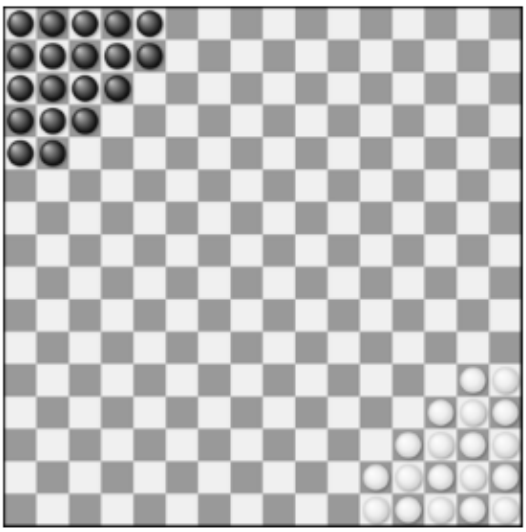
In this project, we will play the game of Halma, an adversarial game with some similarities to checkers. The game uses a 16x16 checkered gameboard. Each player starts with 19 game pieces clustered in diagonally opposite corners of the board. To win the game, a player needs to transfer all their pieces from their starting corner to the opposite corner, into the positions that were initially occupied by the opponent.

Setup for two players:

- Simple wooden pawn-style playing pieces, often called "Halma pawns."
- The board consists of a grid of 16×16 squares.
- Each player's camp consists of a cluster of adjacent squares in one corner of the board. These camps are delineated on the board.
- For two-player games, each player's camp is a cluster of 19 squares. The camps are in opposite corners.
- Each player has a set of pieces in a distinct color, of the same number as squares in each camp.
- The game starts with each player's camp filled by pieces of their own color.

The initial setup is shown below for black and white players.



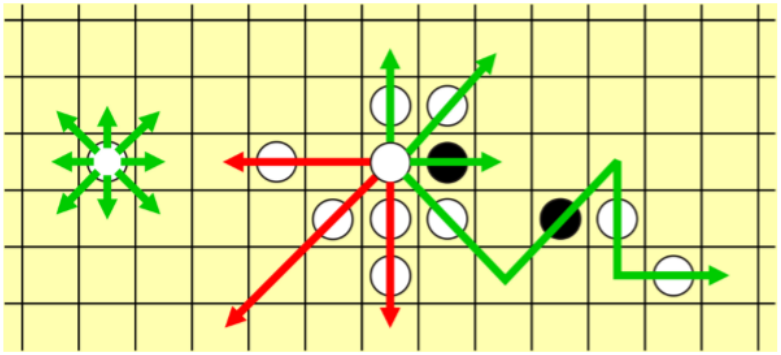
Play sequence:

We first describe the typical play for humans. We will then describe some minor modifications for how we will play this game with artificial agents.

- Create the initial board setup according to the above description.
- Players randomly determine who will move first.
- Pieces can move in eight possible directions (orthogonally and diagonally).
- Each player's turn consists of moving a single piece of one's own color in one of the following plays:
    - One move to an empty square:
        - Move the piece to an empty square that is adjacent to the piece's original position (with 8-adjacency).
        - This move ends the play for this player's turn.
    - One or more jumps over adjacent pieces:
        - An adjacent piece of any color can be jumped if there is an empty square on the directly opposite side of that piece.
        - Place the piece in the empty square on the opposite side of the jumped piece.
        - The piece that was jumped over is unaffected and remains on the board.

    - After any jump, one may make further jumps using the same piece, or end the play for this turn.
    - In a sequence of jumps, a piece may jump several times over the same other piece.
- Once a piece has reached the opposing camp, a play cannot result in that piece leaving the camp.
- If the current play results in having every square of the opposing camp that is not already occupied by the opponent to be occupied by one's own pieces, the acting player wins. Otherwise, play proceeds to the other player.

Below we show examples of valid moves (in green) and invalid moves (in red). At left, the isolated white piece can move to any of its empty 8 neighbors. At right, the central white piece can jump over one adjacent piece if there is an empty cell on the other side. After one jump is executed, possibly several other valid jumps can follow with the same piece and be combined in one move; this is shown in the sequence of jumps that start with a down-right jump for the central piece. Note that additional valid moves exist that are not shown (e.g., the central white piece could move to some adjacent empty location).



Note the invalid moves: red arrow going left: cannot jump over one or more empty spaces plus one or more pieces. Red arrow going left-down: cannot jump over one or more pieces plus one or more empty spaces. Red arrow going down: cannot jump over more than one piece.

Playing with agents:

The game has two scenarios:

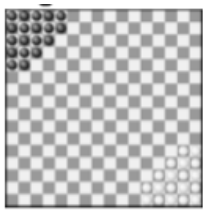
1) **Single move:** The agent will be given in input.txt a board configuration, a color to play, and some number of seconds of allowed time to play one move. The agent should return in output.txt the chosen move(s) before the given play time has expired. Play time is measured as total CPU time used by your agent on all CPU threads it may spawn (so, parallelizing your agent will not get you any free time).
2) **Play against reference agent:** There will be a limited total amount of play time available to your agent for the whole game (e.g., 100 seconds), so the agent should think about how to best use it throughout the game. This total amount of time will vary from game to game. In case of a draw, the agent with more remaining play time wins

Input and output file formats:

Input: The file input.txt in the current directory of your program will be formatted as follows:

First line: A string SINGLE or GAME to let you know whether you are playing a single move (and can use all of the available time for it) of playing a full game with potentially many moves (in which case you should strategically decide how to best allocate your time across moves).

Second line: A string BLACK or WHITE indicating which color you play. The colors will always be organized on the board as follows:

 (black starts in the top-left corner and white in the bottom-right).

Third line: A strictly positive floating-point number indicating the amount of total play time remaining for your agent.

Next 16 lines: Description of the game board, with 16 lines of 16 symbols each:

- W for a grid cell occupied by a white piece
- B for a grid cell occupied by a black piece
- . (a dot) for an empty grid cell

Output: The file output.txt which your program creates in the current directory should be formatted as follows:
1 or more lines: Describing your move(s). There are two possible types of moves (see above):

E FROM_X, FROM_Y TO_X, TO_Y – The agent moves one of your pieces from location FROM_X, FROM_Y to adjacent empty location TO_X, TO_Y. We will again use zero-based, horizontal-first, start at the top-left indexing in the board, as in homework 1. So, location 0,0 is the top-left corner of the board; location 15,0 of the top-right corner; location 0,15 is the bottom-left corner, and location 15,15 the bottom-right corner. As explained above, TO_X, TO_Y should be adjacent to FROM_X, FROM_Y (8-connected) and should be empty. If you make such a move, you can only make one per turn.

J FROM_X, FROM_Y TO_X, TO_Y – The agent moves one of your pieces from location FROM_X,FROM_Y to empty location TO_X,TO_Y by jumping over a piece in between. You can make several such jumps using the same piece, as explained above, and should write out one jump per line in output.txt.

1. Moving pieces

a) Players cannot make a move that starts outside their own camp and causes one of their pieces to end up in their own camp.
b) If a player has at least one piece left in their own camp, they have to
• Move a piece out of their camp (i.e. at the end of the whole move the piece ends up outside of their camp).
• If that is not possible, move a piece in their camp further away from the corner of their own camp ([0,0] or [15,15] respectively).

Only if the player does not have any pieces left in their camp or none of the two alternatives above are possible are they free to move pieces outside of their camp.

Note: To move “further away”, you should simply move so that you either move further away horizontally (while not moving closer vertically), or vertically (while not moving closer horizontally), or both.

2. Winning the game (no real change here, just a clarification)
A player wins if they are the first to fill out all the space in the opposite camp that’s not occupied by the opposite player’s pieces using at least one of their pieces.

Example:

SINGLE
WHITE
100.0
BBBBB.....
BBBBB.....
BBBB.....
BBB.....
BB.....
.....
.....
.....
.....
.....
.....
.....WW
.....WWW
.....WWWW
.....WWWWW
.....WWWWW

For example, **output.txt** may contain:

E 11,15 10,15

This moves the leftmost white piece on the bottom row of the board to the left. The resulting board would look like this, given the above input.txt:

BBBBB.....
BBBBB.....
BBBB.....
BBB.....
BB.....
.....
.....
.....
.....
.....
.....
.....WW
.....WWW
.....WWWW
.....WWWWW
.....W.WWWW

Or it could contain:

J 12,15 10,13

which is a jump along the left-up diagonal for a bottom piece. The board would look like this after this jump, given the above input.txt:

BBBBB.....
BBBBB.....
BBBB.....
BBB.....
BB.....
.....
.....
.....
.....
.....
.....
.....WW
.....WWW
.....W.WWWW
.....WWWWW
.....W.WWW

Examples:

Example 1:

For this input.txt:

SINGLE
WHITE
100.0
WWWWW.....
WWWWW.....
WWW.....
WWW.W.....
WW.....
.....
.....
.....
.....
.....
.....
.....B...BB
.....BBB
.....BBB
.....BBBBB
.....BBBBB

one possible correct output.txt is:

E 4,3 3,2

Example 2:

For this input.txt:

SINGLE
WHITE
6.6
WWWWW.....
WW.WW.....
WWWW.....
WWW.W.....
WW.....
.....
.....
.....
.....
.....
.....
.....B.BB
.....B.BBB
.....B.B
.....BBBBB
.....BBBBB

one possible correct output.txt is:

J 4,3 2,1

Example 3:

For this input.txt:

SINGLE
BLACK
23.33
WWWWW.....
W.W.W.....
WWW.....
WW...W.....
WW....W.....
.....
.....W.....
.....
.....
.....BW.....
.....B....
.....BB
.....BBBB
.....B.BB
.....BBB
.....BBBBB

one possible correct output.txt is:

J 9,9 11,9
J 11,9 11,11
J 11,11 13,13