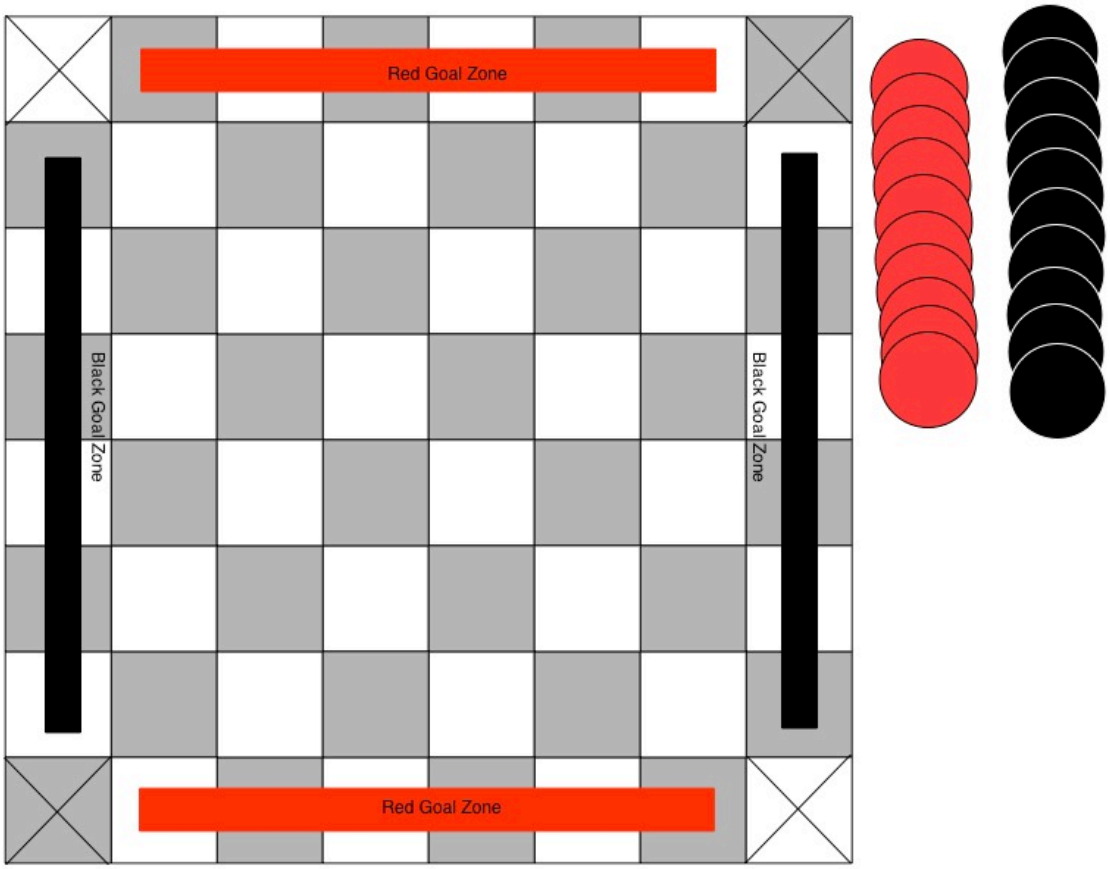
David Galles
Computer Science
Univerisity of San Francisco

Network (Due 5/13/2015)

For your fourth and final project, you will create a program that allows users to play the game Network, and build a simple AI that allows users to play against each other. We will pit each of your AIs against each other to find out who can design the championship Network player.

The Game of Network

Network is a game created by Sid Sackson, in his book "A Gamut of Games". It is played on an 8x8 checkerboard, with two players. The top and bottom rows of the checkerboard are the goal regions for player 1, and the left and right regions are the goal regions for player 2. Each player has ten pieces, and tries to make a "network" across the board

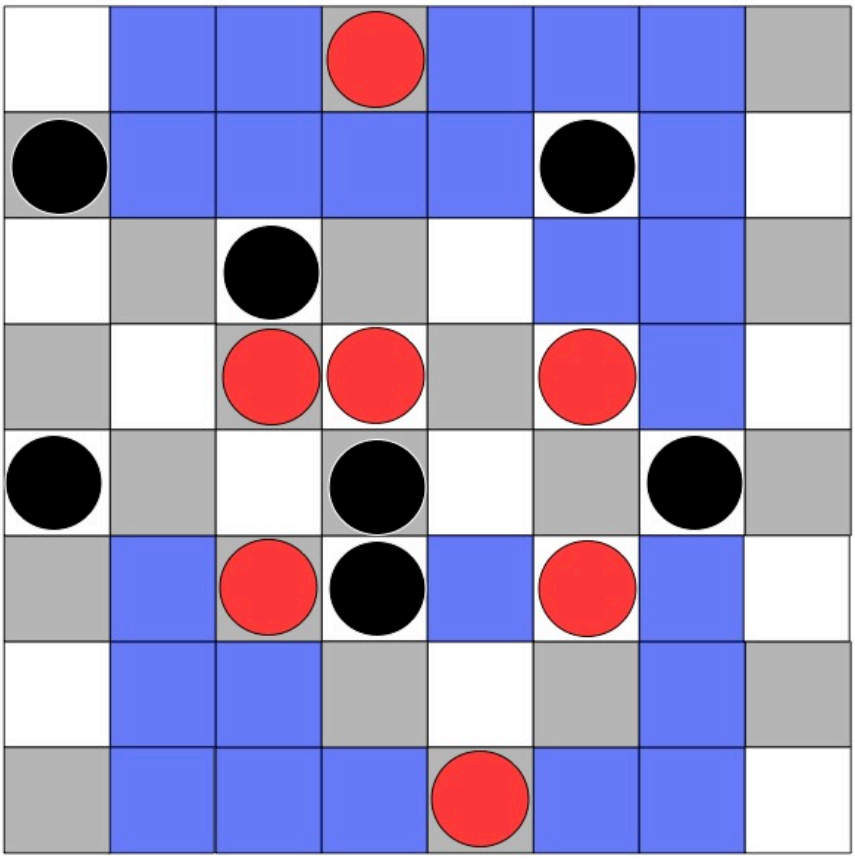


Playing the Game

Players take turns placing pices on the board, with the following restrictions:

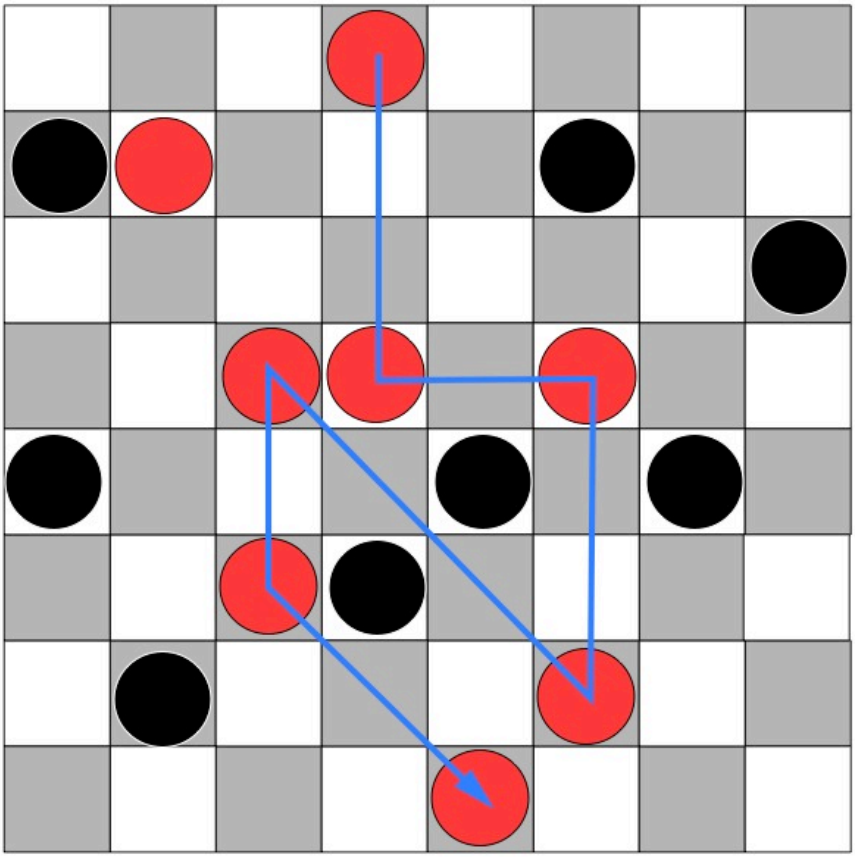
- No person can play in the 4 corners of the board
- No person can play in the other person's goal regions
- No person can play to form a clump of 3 or of their own colored pieces (connected horizontally, vertically, or diagonally)

In the board below, assuming it is red's move, the spaces that are *legal* to play in are marked in blue



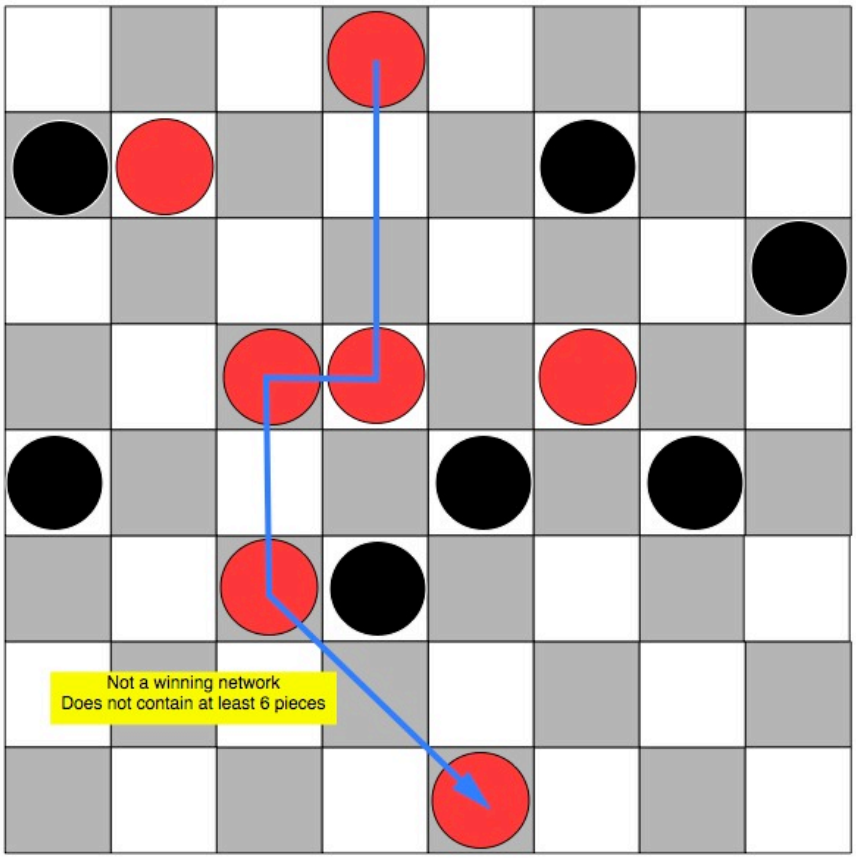
Winning Networks

Players are trying to create a network -- a series of 6 or more pieces that connect their goal areas. Two pieces can be connected if they are adjacent, or if they are separated only by empty spaces along a horziontal, vertical, or diagonal line. The following is a win for red:

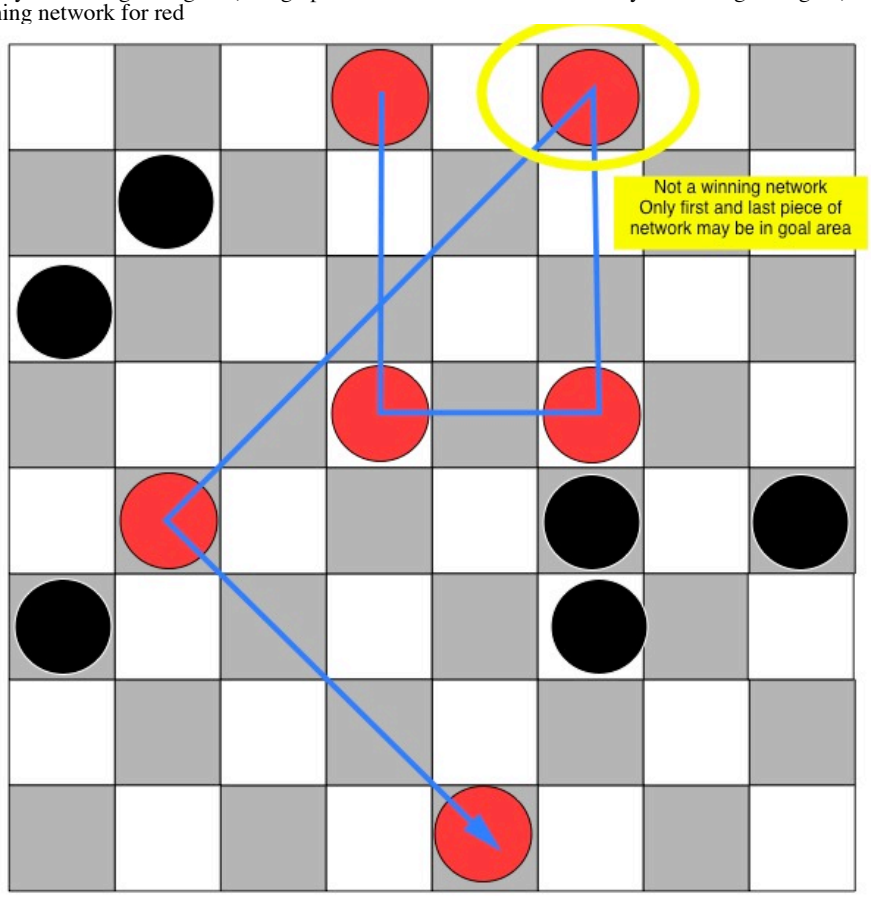


Winning networks have the following properties:

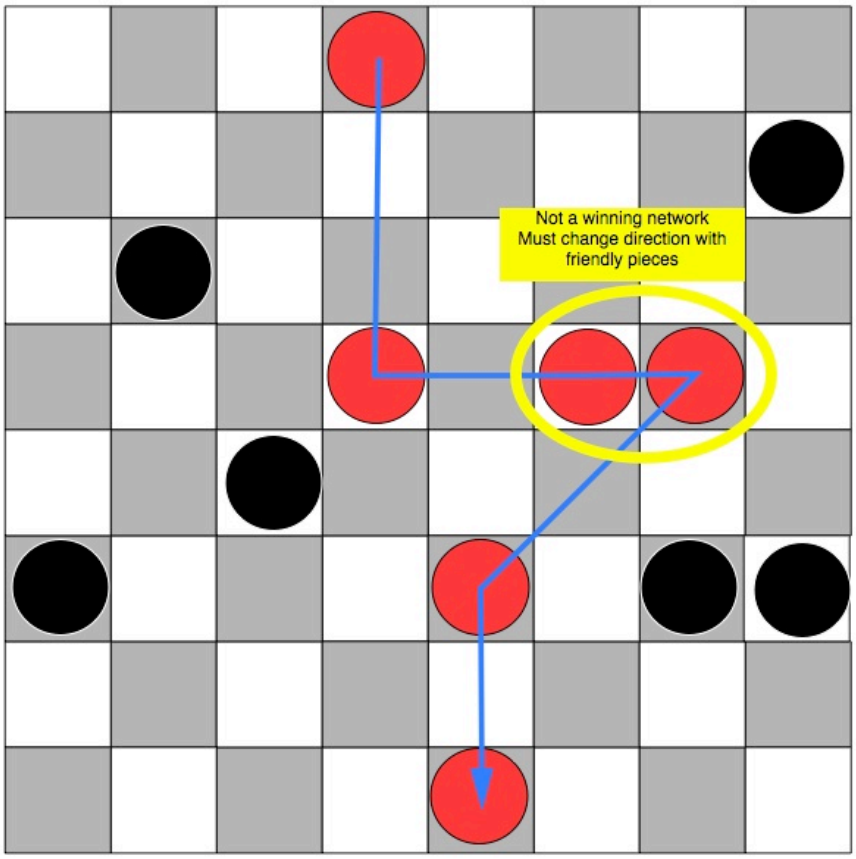
- A winning network must contain at least 6 pieces. More that 6 piece networks are acceptable, and not all player pieces on the board need to be a part of the network. So the following is not a winning network for red, since it only contains 4 pieces



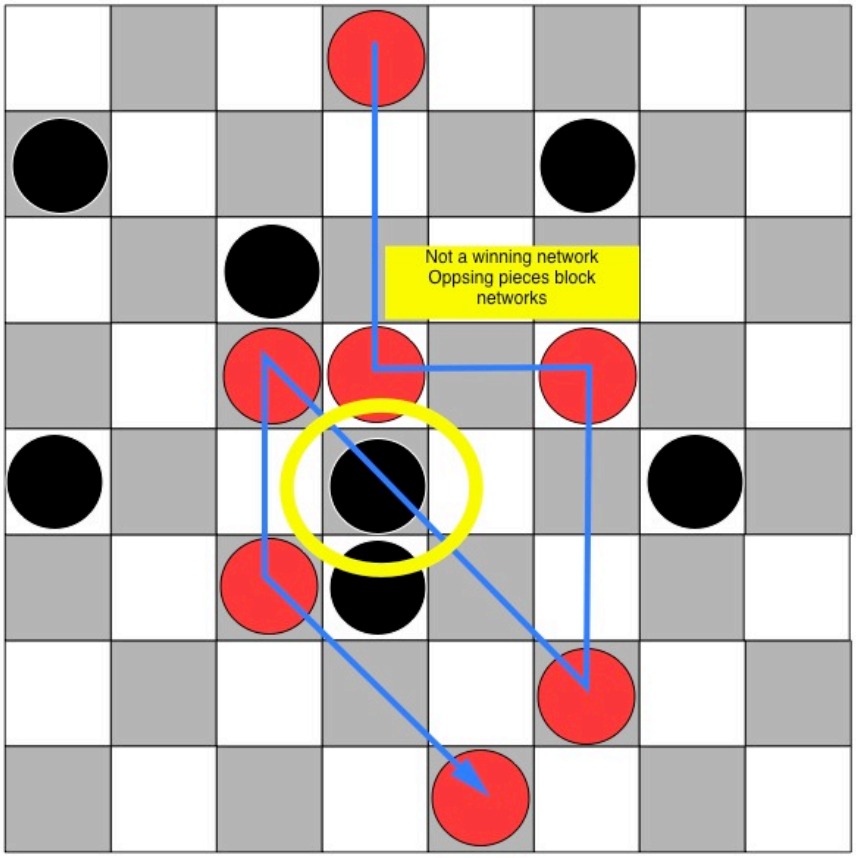
- A winning network must start in one goal region, and end in the other gool region. Only the start and end piece of a network may be in the goal region (though pieces not used in the network may be in the goal region) So the following is *not* a winning network for red



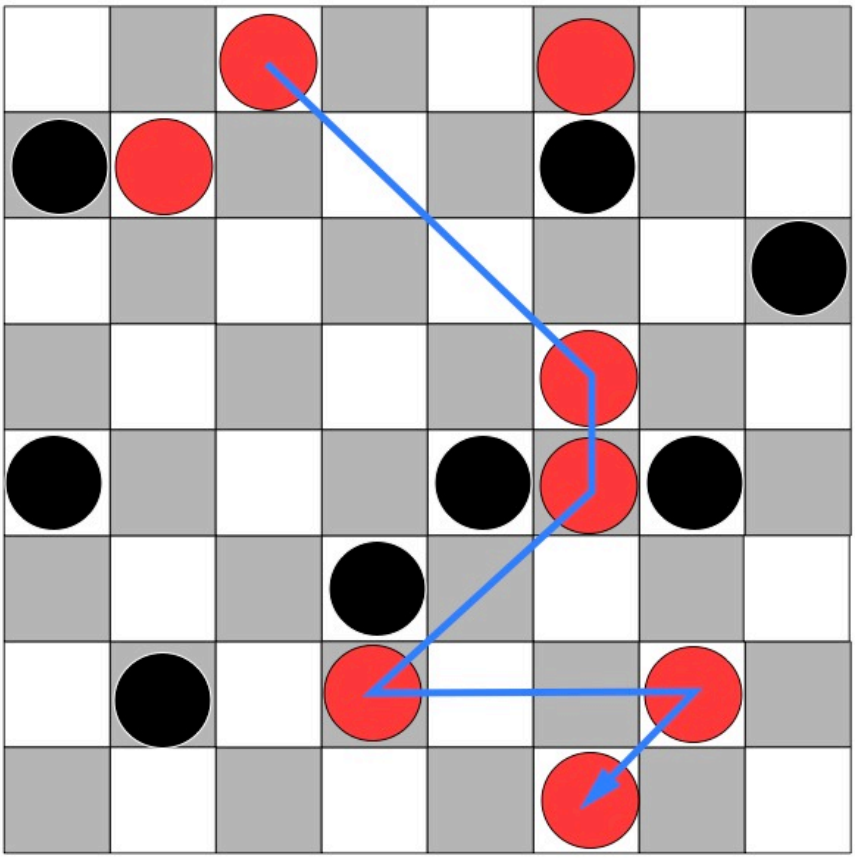
- The network must change direction with every friendly piece. So the following network is *not* a winning network for red:



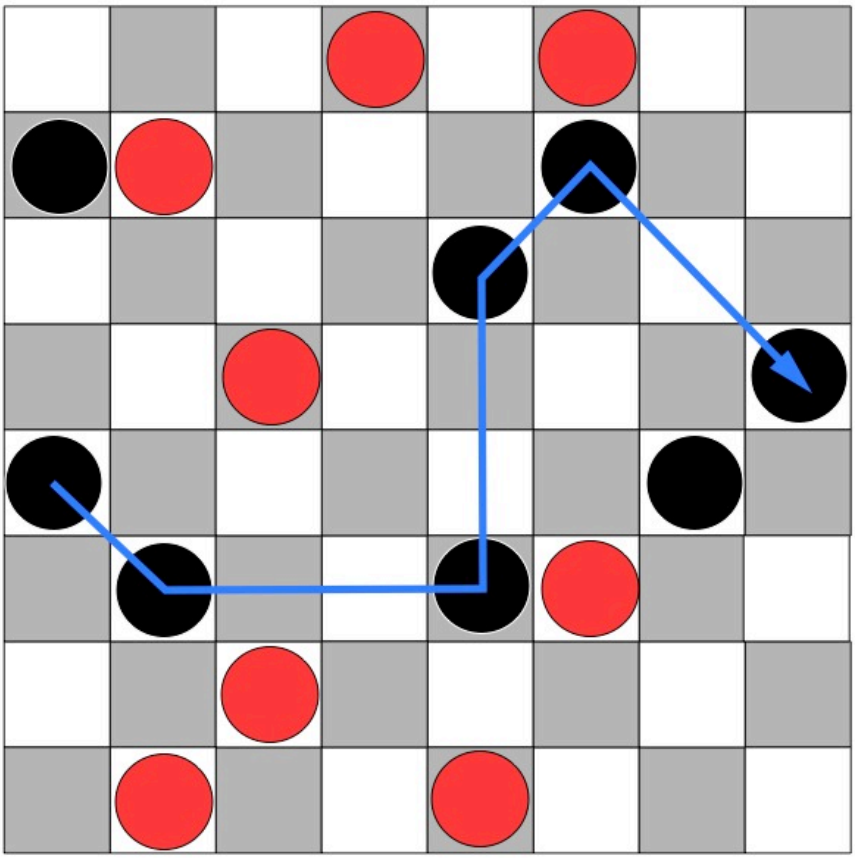
- Winning networks cannot be blocked by opposing pieces. So the following is *not* a winning network for red:



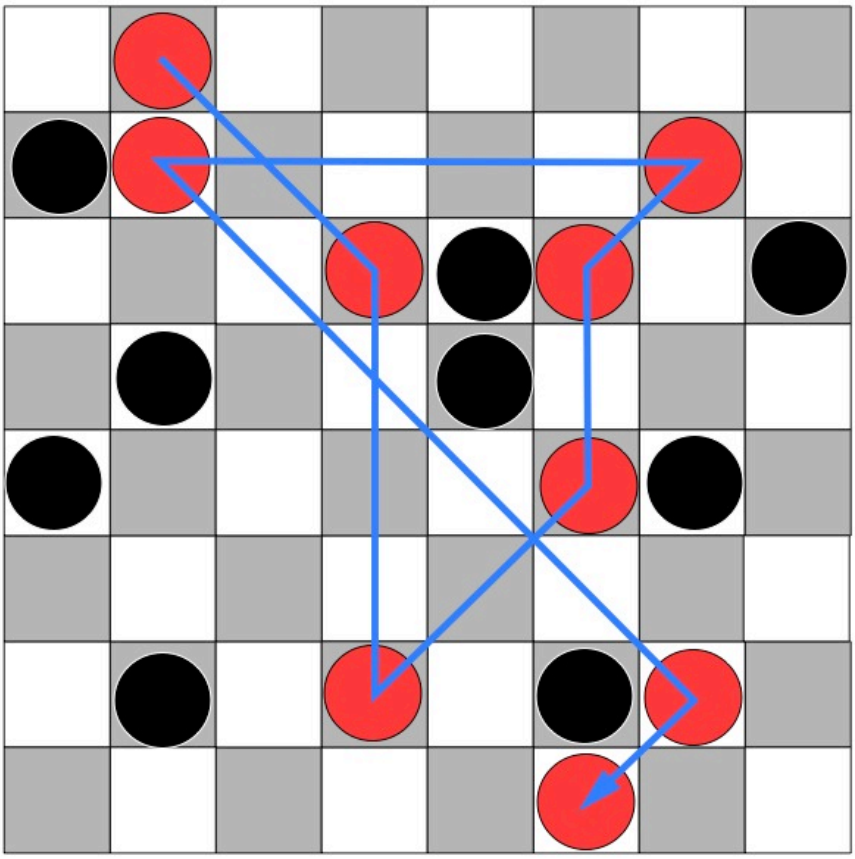
Some more example winning networks: Win for red



Win for black



Win for red



After all Pieces are Played

If all 10 pieces are played by both players without forming a winning network, then players take turns moving pieces. On each turn after all 10 pieces have been placed, a player moves one of her pieces to another location on the board, with the following restrictions:

- The place where the piece is moved to must follow all the same restrictions for placing pieces. (No clumps of 3 or more pieces, no playing in the corners, no playing in the opponent's goal areas)
- A player cannot move a piece in a way that results in a winning network for both players (which could happen if the moved piece unblocks an opponent's network, and creates a winning network at the same time)

Required Input/Output

In order to make our grading scripts work correctly on your projects, your program will need to conform to some standards. Failure to follow these standards will require extra time grading your project, and will lose you points!

- Board positions will be denoted by a character (A-H) representing the row, and a number (1-8) representing the column
- The first player (numbered "1") will have goal regions rows A and H
- The second player (numbered "2") will have goal regions in columns 1 and 8

Gameplay proceeds as follows:

- First, prompt the player for how many human players (0 or 1 or 2)
- For a zero-player game
  - Show the board after every move, and then show the winning network as a list of board locations (A3 D6 E6 G4 G7 H6)
- For a one-player game
  - Show the current board
  - Prompt the human player for a move (human player goes first)
  - Read in the next move for the human player: A letter followed by a number (for the row and column of the piece to move), or a letter followed by number followed by a letter followed by a number (after 10 pieces have been played, and a piece is to be moved)
  - If the move is invalid, state *why* that the move is invalid (a piece is alread in that location, placing a piece at that location would form a cluster of 3 or more pieces, the location is in opponent's goal, the location is in corner, move would create network for both players), and prompt the user for another move.
  - Make the move. If the human player wins, print out winning network as a list of board positions (i.e., A3 D6 E6 G4 G7 H6) and quit.
  - Otherwise, print the board after the player move and make the computer move. If the computer wins, print the board after the computer move, print winning network and quit
  - Otherwise, repeat with next player move
- For a two-player game
  - Show the board.
  - Read in the next move for the next player: A letter followed by a number (for the row and column of the piece to

move), or a letter followed by number followed by a letter followed by a number (after 10 pieces have been played, and a piece is to be moved)
    - If the move is invalid, state *why* the move is invalid, (a piece is alread in that location, placing a piece at that location would form a cluster of 3 or more pieces, the location is in opponent's goal, the location is in corner, move would create network for both players) and prompt the user for a different move
    - Make the move. If the human player wins, print out winning network as a list of board positions (i.e., A3 D6 E6 G4 G7 H6) and quit.
    - Repeat for the next player

Required Archtecture

You are mostly allowed to build this program as you see fit, with two exceptions:

- Your main class must be named "Network.java".
- You must implement a computer player named ComputerPlayer and a human player named HumanPlayer, each of which has a constructor that takes a single parameter (player number, either 1 or 2) and subclass the provided abstract class Player.java:

```
public abstract class Player
{
    protected int playerNum;

    public Player(int playerNum)
    {
        this.playerNum = playerNum;
    }

    public abstract Move getMove();
    public abstract void OpponentMove(Move m);

}
```

Where Move is defined by the class Move.java

```
/**
 * Representation of a Move in Network.  Each move is either a PLACE_PIECE move or a
 * MOVE_PIECE move.  If it is a PLACE_PIECE move, then the toRow and toCol fields are the
 * row and column to place the piece, and the fromRow and fromCol fields are ignored.
 * If this is a MOVE_PIECE move, then fromRow and fromCol are the row and column to
 * move the piece from, and toRow and toCol are the row and column to move the piece to
 */

public class Move
{
    public enum MoveType {PLACE_PIECE, MOVE_PIECE};

    public char toRow;
    public int toCol;

    public char fromRow;
    public int fromCol;

    public MoveType moveType;


    public Move(char row, int col)
    {
        moveType = MoveType.PLACE_PIECE;
        toRow = row;
        toCol = col;
    }

    public Move(char fromRow, int fromCol, char toRow, int toCol)
    {
        moveType = MoveType.MOVE_PIECE;
        this.fromRow = fromRow;
        this.fromCol = fromCol;
        this.toRow = toRow;
        this.toCol = toCol;
    }
 }
```

This part of the architecture is required so that we can pit your computer players against each other.

Computer Player

You will need to implement a computer player, which should use a min/max algorithm as described in class (preferably using alpha-beta pruning, though that is not required) and an evaluation function of your own design. You have a fair amount of leeway on what evaluation function to use -- though some ideas are:

- The evaluation function should return the largest positive value when one player wins, and the largest negative value when the other player wins

- It might be a good idea for your evaluation function to keep track of the number and / or length of partial networks

You will likely need to be able to make and unmake moves on your board to get this search to work correctly. Your are *strongly encouraged* to see me to get ideas, and to check if your ideas for the computer player are reasonable.

Network Tournament

We will be running a tournament running your programs against each other.

To make our lives easier, please rename your computer player as yourUsernameComputerPlayer (so mine would be gallesComputerPlayer), and rename all of the classes needed by your computer player (a board class, for instance) by prepending your username (so I would name my board class gallesBoard, for instance)

Note that the "correct" way to do this is with java packages, but this is the eaisest last-minute stopgap solution, since we are so close to the due date.

Program Requirements

- Correctly prints out board, inputs legal moves, records moves on board. 20 points.
- Program detects *all* illegal player moves (playing outside board range, playing in opponent's goal zone, forming clumps, moving a piece to cause networks to be created for both players), malformed input, etc, and prints out appropriate message to player. 30 points
- Detects and prints out wins correctly 30 points
- Computer player makes only legal moves 10 points
- Compter player wins more than 75% of games played against a completley random computer player 10 points

Supporting Files

- Player.java Absract superclass that your HumanPlayer and ComputerPlayer must subclass
- Move.javaMove class that your Human and Computer players will use
- SingletonScanner.java Standard input Scanner class that is more friently to piping a file into standard in
- NetworkTest1
- NetworkTest2
- NetworkTest3
- NetworkTest4
- NetworkTest5
- NetworkTest5

Sample Output

```
Enter the number of human players (0, 1 or 2): 2
  1 2 3 4 5 6 7 8
A:. . . . . . . .
B:. . . . . . . .
C:. . . . . . . .
D:. . . . . . . .
E:. . . . . . . .
F:. . . . . . . .
G:. . . . . . . .
H:. . . . . . . .
Player 1: Enter row and colum position to place piece:A 4
  1 2 3 4 5 6 7 8
A:. . . 1 . . . .
B:. . . . . . . .
C:. . . . . . . .
D:. . . . . . . .
E:. . . . . . . .
F:. . . . . . . .
G:. . . . . . . .
H:. . . . . . . .
Player 2: Enter row and colum position to place piece:D 3
  1 2 3 4 5 6 7 8
A:. . . 1 . . . .
B:. . . . . . . .
C:. . . . . . . .
D:. . 2 . . . . .
E:. . . . . . . .
F:. . . . . . . .
G:. . . . . . . .
H:. . . . . . . .
Player 1: Enter row and colum position to place piece:B 5
  1 2 3 4 5 6 7 8
A:. . . 1 . . . .
B:. . . . 1 . . .
C:. . . . . . . .
D:. . 2 . . . . .
E:. . . . . . . .
F:. . . . . . . .
G:. . . . . . . .
H:. . . . . . . .
```

```
Player 2: Enter row and colum position to place piece:D 1
  1 2 3 4 5 6 7 8
A:. . . 1 . . . .
B:. . . . 1 . . .
C:. . . . . . . .
D:2 . 2 . . . . .
E:. . . . . . . .
F:. . . . . . . .
G:. . . . . . . .
H:. . . . . . . .
Player 1: Enter row and colum position to place piece:A 6
Player 1: Bad move!
Position would form clump
Player 1: Enter row and column to place piece:B 7
  1 2 3 4 5 6 7 8
A:. . . 1 . . . .
B:. . . . 1 . 1 .
C:. . . . . . . .
D:2 . 2 . . . . .
E:. . . . . . . .
F:. . . . . . . .
G:. . . . . . . .
H:. . . . . . . .
Player 2: Enter row and colum position to place piece:B 1
  1 2 3 4 5 6 7 8
A:. . . 1 . . . .
B:2 . . . 1 . 1 .
C:. . . . . . . .
D:2 . 2 . . . . .
E:. . . . . . . .
F:. . . . . . . .
G:. . . . . . . .
H:. . . . . . . .
Player 1: Enter row and colum position to place piece:G 2
  1 2 3 4 5 6 7 8
A:. . . 1 . . . .
B:2 . . . 1 . 1 .
C:. . . . . . . .
D:2 . 2 . . . . .
E:. . . . . . . .
F:. . . . . . . .
G:. 1 . . . . . .
H:. . . . . . . .
Player 2: Enter row and colum position to place piece:A 3
Player 2: Bad move!
Position in opponent goal
Player 2: Enter row and column to place piece:B 4
  1 2 3 4 5 6 7 8
A:. . . 1 . . . .
B:2 . . 2 1 . 1 .
C:. . . . . . . .
D:2 . 2 . . . . .
E:. . . . . . . .
F:. . . . . . . .
G:. 1 . . . . . .
H:. . . . . . . .
Player 1: Enter row and colum position to place piece:D 2
  1 2 3 4 5 6 7 8
A:. . . 1 . . . .
B:2 . . 2 1 . 1 .
C:. . . . . . . .
D:2 1 2 . . . . .
E:. . . . . . . .
F:. . . . . . . .
G:. 1 . . . . . .
H:. . . . . . . .
Player 2: Enter row and colum position to place piece:E 8
  1 2 3 4 5 6 7 8
A:. . . 1 . . . .
B:2 . . 2 1 . 1 .
C:. . . . . . . .
D:2 1 2 . . . . .
E:. . . . . . . 2
F:. . . . . . . .
G:. 1 . . . . . .
H:. . . . . . . .
Player 1: Enter row and colum position to place piece:H 6
  1 2 3 4 5 6 7 8
A:. . . 1 . . . .
B:2 . . 2 1 . 1 .
C:. . . . . . . .
D:2 1 2 . . . . .
E:. . . . . . . 2
```

```
F:. . . . . .
G:. 1 . . . . .
H:. . . . 1 . .
Player 1 wins! A4 B5 B7 G2 D2 H6
```