CSSA Programming Contest:
Votey Checkers

1 The Rules of the Game

This game is played on an ordinary 8×8 checkerboard, with 12 white checkers, and 12 black checkers. Each square on the board will be labeled by a two digit number, the tens place denoting the row, and the ones place denoting the column, as shown in Figure 1. The pieces are placed on the board as shown in Figure 2: the black pieces on squares 12, 13, 14, 15, 16, 17, 82, 83, 84, 85, 86, 87, and the white pieces on squares 21, 28, 31, 38, 41, 48, 51, 58, 61, 68, 71, and 78. Like in checkers, players alternate turns, moving one piece per turn. Black moves first. Unlike ordinary checkers, pieces can move along a horizontal, vertical, or diagonal line. The unusual feature of this game is that the number of squares that a piece moves equals the total number of pieces that sit along that particular line on the board. For example, in the opening position in Figure 2, the black piece on square 14 can move to either squares 32, 34, or 36, because each one of those lines contains two pieces. When moving a piece it is allowed to jump over any piece of its own color; it cannot, however, jump over any of its opponents pieces. Conversely, a piece is not allowed to land on a piece of its own color, but it can land on a square that contains an opposing piece. In the latter case, the opposing piece is "captured," and removed from the board for the duration of the game. The winner of the game is the first play who is able to arrange his or her pieces so that they form a single *connected arrangement*. This means that it is possible to traverse through all of ones pieces, without hitting any opposing pieces or blank squares, by a sequence of unit horizontal, vertical, and diagonal displacements.

11	12	13	14	15	16	17	18
21	22	23	24	25	26	27	28
31	32	33	34	35	36	37	38
41	42	43	44	45	46	47	48
51	52	53	54	55	56	57	58
61	62	63	64	65	66	67	68
71	72	73	74	75	76	77	78
81	82	83	84	85	86	87	88

Figure 1: Each square is represented by a two-digit integer. The most significant digit indicated the row; the least, the column.

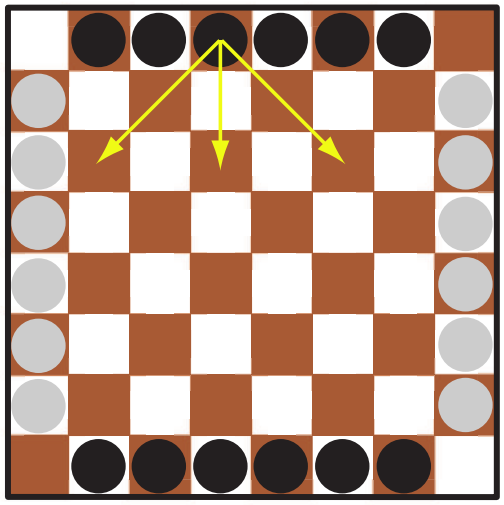

Figure 2: The initial configuration of the 24 pieces. The three yellow arrows indicate the possible moves for the piece on square 14.

An example end game position is shown in Figure 3. If it's black's turn then the piece at position

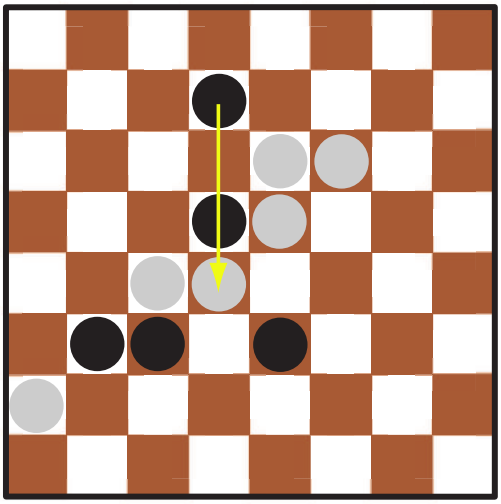

Figure 3: An endgame position in Votey checkers. If it's black's turn, then moving the piece from 24 to 54 results in a winning connected arrangement, after the white piece at 54 is captured. If it's white's turn, then the move from 36 to 34 is a good defensive move.

24 can move one square left to 23, one square right to 25, one square diagonally to 15 or 33, or two squares vertically to 54. Note that it cannot move diagonally to 46 as this would require a jump over an opposing piece, which is always illegal. The vertical move to 54 is legal because one can always jump over a piece of one's own color, and can always land on an opposing piece. In this case, the move to 54 results in a victory for black, as all of that players pieces (62, 63, 54, 65, and 44) form a connected arrangement. If on the other hand, it is white's move, then moving the piece from 36 to 34 would prevent black from making an immediate winning move.

If a player creates a connected arrangement for its pieces, and simultaneously (by a capture) creates a connected arrangement for those of its opponent, then the player making the move is declared the winner. Thus, ties cannot occur. Note also that a single piece constitutes a connected arrangement.

2 Program Guidelines

Each team should create a software agent that will play Votey Checkers against either a human player, or another software agent. A tournament will be organized that will pit each agent against the others. Although it is not clear if black or white has the advantage, during the tournament each agent will play approximately an equal number of games with each color.

Your agent should be an executable program that runs on tyr. You may name your executable as you please, but it should accept one command line argument, which will be 1 if your agent is to play black, and 2 if your agent is to play white. For example, let's say your agent is named sunny, and it is about to play a game against blinky. The referee determines that sunny will play black, and blinky will play white. The command

```
tyr> sunny 1
```

will be invoked in one shell, and the command

```
tyr> blinky 2
```

will be invoked in the other.

If your agent is black, it should select an opening move. All moves will be described by a four digit integer: the two most significant digits denote the original location of the piece that is being moved, and the least two significant digits denote the destination. Thus the opening move from square 14 to square 34 is described by the single number `1434`. In order to register a move, your agent should invoke the integer valued function

```
int play(int move);
```

where the argument *move* should equal the value of a legal move, e.g. 1434 in the opening. This function will then return the value of the resulting move of the opposing agent. (Function play will return -1 if the input value corresponds to an illegal move, or if too much time was taken. See below.) For development, this function can simply print the move to `stdout`, and retrieve a move from `stdin`. For example, in C++,

```
int play(int myMove) {
    int yourMove;
    cout << myMove << endl;
    cin  >> yourMove;
    return yourMove;
}
```

If your agent is playing white, then it must receive an opening move from black before it can proceed. To accomplish this, an initial call of `play(0)` will return black's first move.

A player who plays a winning move must declare it as such by negating the integer value of the move. Thus, in the example described in Figure 3, black's winning move should be expressed by the function call `play(-2454);`.

If an agent makes an illegal move, incorrectly declares a winning move, or neglects to negate a winning move, then it will be disqualified, and the opposing agent will be declared the winner. Finally, each move must be completed within one minute of CPU time.

Good Luck! And may the best agent win!