COMPUTING LABORATORY ASSESSED EXERCISE:
Othello/Reversi

Autumn term 2001:

Author: **Andy Bennett** EEE
Username: **ajb01**
Grade: **XXXXX**
Marker: **XXXXX**

Project description

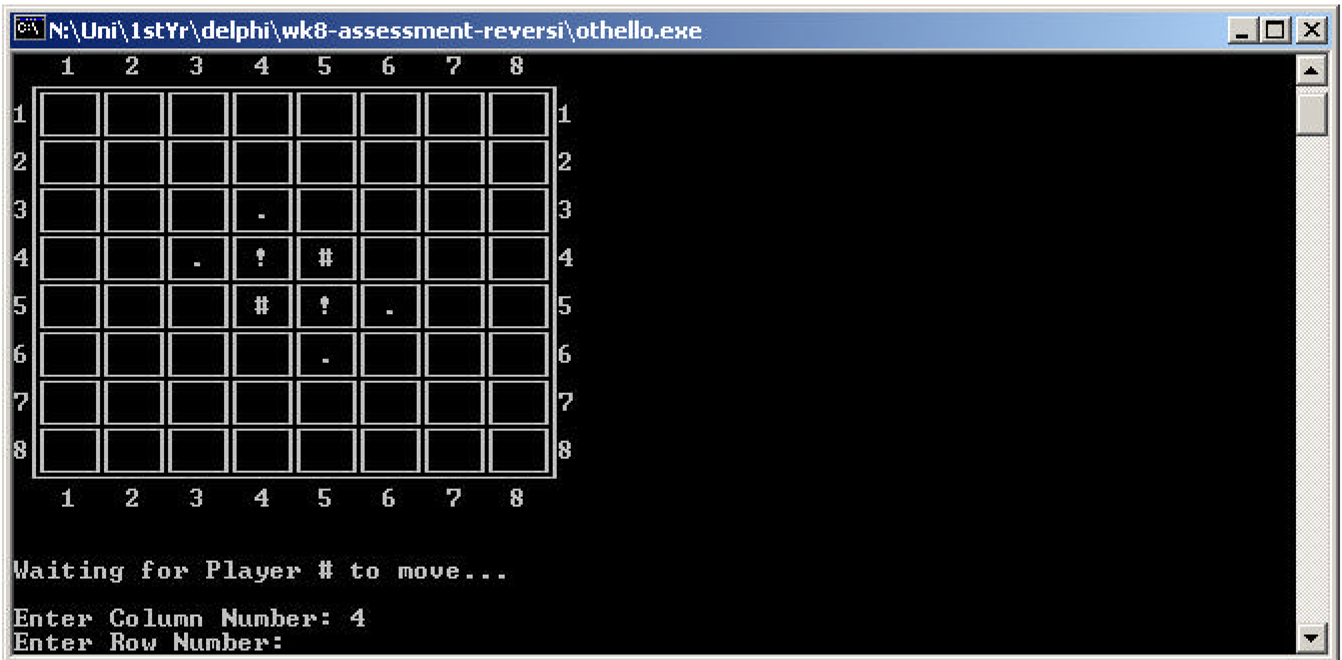
The program supervises a game of Othello/Reversi between two players, both of which are playing at the same console. On program startup the instructions can be requested and at the beginning of each game the players can choose whether the positions of valid moves are shown. For each player in turn, the program works out if there are any valid moves available. If so, the user is shown the playing board and prompted for a cell to occupy. The program then performs the necessary flipping and moves to the next player. If the player has no valid moves, they are told and game play automatically resumes with the next player. When there are no valid moves for either player, the number of cells occupied by each player is counted and the winner is announced. The game can then be played again or the program can end.

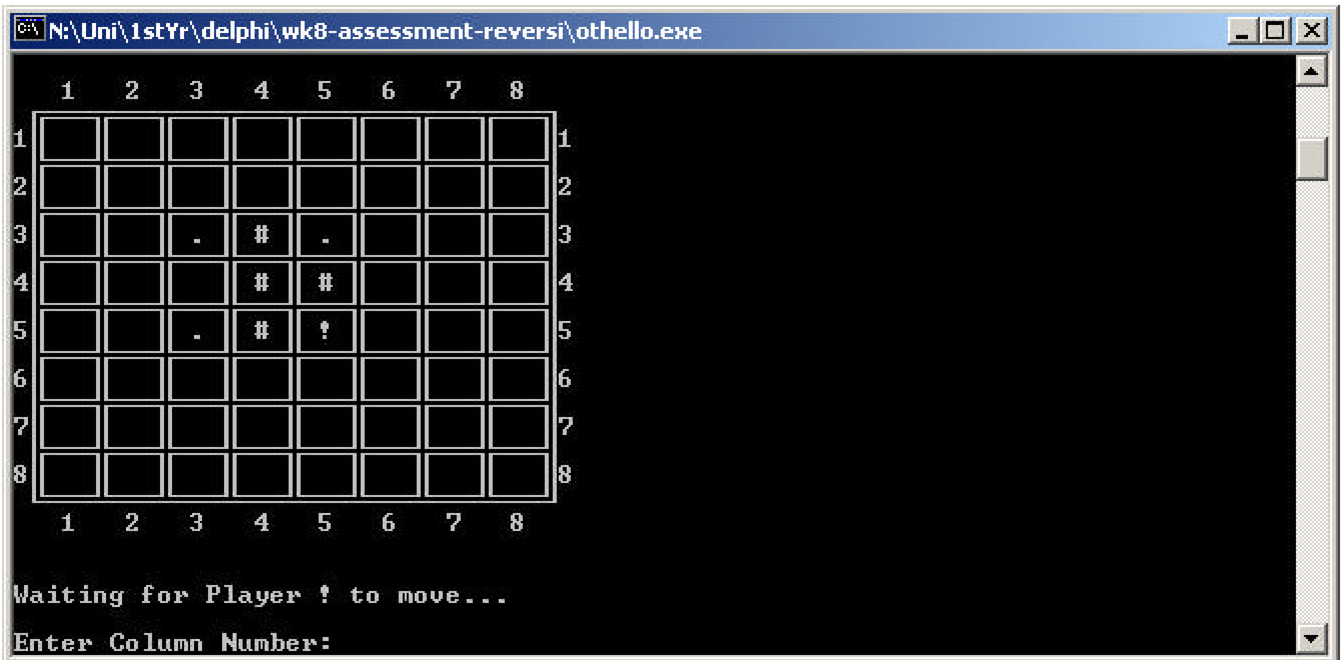
Screen dumps of program output

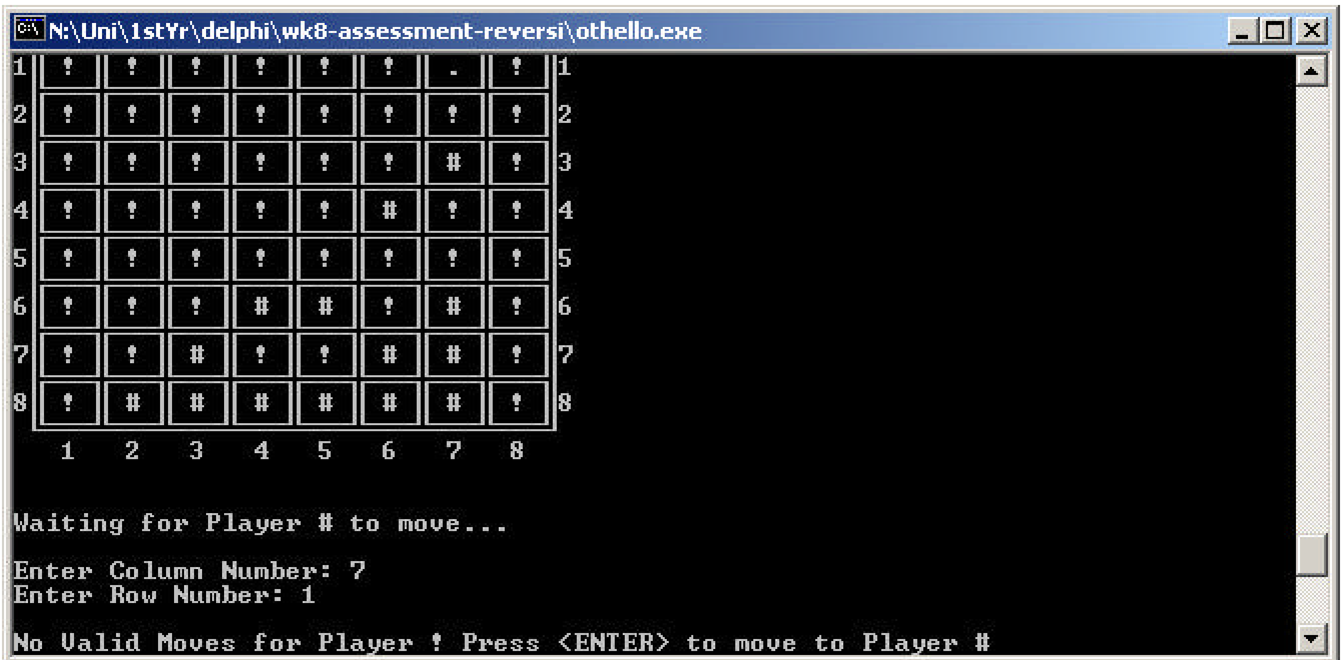
```
N:\Uni\1stYr\delphi\wk8-assessment-reversi\othello.exe
   1  2  3  4  5  6  7  8
1 [ ][ ][ ][ ][ ][ ][ ][ ] 1
2 [ ][ ][ ][ ][ ][ ][ ][ ] 2
3 [ ][ ][ ][ ][ ][ ][ ][ ] 3
4 [ ][ ][ ][!][#][ ][ ][ ] 4
5 [ ][ ][ ][#][!][ ][ ][ ] 5
6 [ ][ ][ ][ ][ ][ ][ ][ ] 6
7 [ ][ ][ ][ ][ ][ ][ ][ ] 7
8 [ ][ ][ ][ ][ ][ ][ ][ ] 8
   1  2  3  4  5  6  7  8

Waiting for Player # to move...

Enter Column Number: 4
Enter Row Number: _
```


N:\Uni\1stYr\delphi\wk8-assessment-reversi\othello.exe
1 2 3 4 5 6 7 8
Waiting for Player # to move...
Enter Column Number: 4
Enter Row Number:


N:\Uni\1stYr\delphi\wk8-assessment-reversi\othello.exe
1 2 3 4 5 6 7 8
Waiting for Player ! to move...
Enter Column Number:


N:\Uni\1stYr\delphi\wk8-assessment-reversi\othello.exe
1 2 3 4 5 6 7 8
Waiting for Player # to move...
Enter Column Number: 7
Enter Row Number: 1
No Valid Moves for Player ! Press <ENTER> to move to Player #

Data Dictionary

Identifier	*Type*	*Scope*	*Description*
BSIZE	Const	Global	Size of the playing board.
board	frame	Main Program	The playing board.
validmoves	frame	Main Program	The valid moves for the current player.
curplr	cell	Main Program	The current player.
move	amove	Main Program	The coordinates where the user moves.
ingame	boolean	Main Program	True until the end of the game.
showvalid	boolean	Main Program	True if the valid moves should be shown.
offset	array of amove	Global	The offsets for moving thru the board in any direction.
pos	amove	calcvalid makemove	A pair of temporary coordinates
foundvalid	boolean	calcvalid	True if there are valid moves for the current player.
a	integer	initboard drawscreen calcvalid getwinner mergeboard	Loop counter.
b	integer	initboard drawscreen calcvalid getwinner mergeboard	Loop counter.
c	integer	calcvalid makemove	Loop counter.
acnt	integer	getwinner	Counter.
bcnt	integer	getwinner	Counter.
ch	string	getmove	Gets input from the user.
ch	char	Main Program	Gets input from the user.
instructfile	textfile	showinstructions	File containing the instructions for the game.

Identifier	Type	Scope	Description
line	string	showinstructions	Temporary variable for reading from file.

Pseudo-code

```
1.0       procedure drawscreen(board)
1.1         write out a blank line
1.2.0       for loop: a = 1 to BSIZE
1.2.1         write out column numbers
1.3         write out carriage return
1.4         write out corner character
1.5.0       for loop: a = 1 to BSIZE-1
1.5.1         write out column headers for each cell
1.6         write out last cell on row with a corner character
1.7.0       for loop: a = 1 to BSIZE-1
1.7.1         write out the row number
1.7.2.0       for loop: b = 1 to BSIZE
1.7.2.1         write out column separator character then drawcell(board(b,a))
1.7.3         write out the end of line character and add a row number
1.7.4         write out row separator characters
1.7.5.0       for loop: b = 1 to BSIZE-1
1.7.5.1         write out row separator characters
1.7.6         write out the end of a line character
1.8         write out the row number for the last row
1.9.0       for loop: b = 1 to BSIZE
1.9.1         write out column separator character and then drawcell(board(b,BSIZE))
1.10        write out end of line character and row number
1.11        write out corner character
1.12.0      for loop: a = 1 to BSIZE-1
1.12.1        write out the column footers
1.13        write out last cell with corner character
1.14.0      for loop: a = 1 to BSIZE
1.14.1        write out column numbers
1.15        Fill the rest of the screen with blank lines – 3 lines needed.
1.16      end procedure


2.0                function calcvalid(board, validmoves byref, curplr)
2.1                  return = false
2.2.0                for loop: a = 1 to BSIZE
2.2.1                  foundvalid = false
2.2.2.0                if board[a,b] is EMPTY then
2.2.2.1.0                for loop: c = 1 to 8
2.2.2.1.1                  pos.x = a
2.2.2.1.2                  pos.y = b
2.2.2.1.3.0                if inbounds(pos) then
2.2.2.1.3.1.0                if board((a+offset[c].x), (b+offset[c].y)) belongs to next plr
2.2.2.1.3.1.1.0                while((NOT foundvalid) AND inbounds(pos) AND
board(pos+offset) <> EMPTY
2.2.2.1.3.1.1.1                  pos = pos + offset
2.2.2.1.3.1.1.2.0                if board(pos) belongs to curplr and inbounds(pos) then
2.2.2.1.3.1.1.2.1                  validmoves[a,b] = VALID
2.2.2.1.3.1.1.2.2                  foundvalid = TRUE
2.2.2.2.0              if foundvalid then return = true
2.2.2.2.1                else validmoves[a,b] = EMPTY
2.3                end function


3.0          Main Program
3.1.0          repeat
3.1.1            ask user if instructions are required...
3.1.2            read in response...
3.1.3          do it until user answers YES or NO
3.2            if user answered YES then showinstructions
3.3.0          repeat
3.3.1          board = initboard
3.3.2          curplr = Player A
3.3.3          initialise offsets for moving aroung playing board
```

```
3.3.4           ingame = true
3.3.5           showvalid = true
3.3.6.0         repeat
3.3.6.1           ask user if showing of valid moves required
3.3.6.2           read in response
3.3.6.3         do it until user answers YES or NO
3.3.7           if user answered YES then showvalid = true
3.3.8.0         while ingame
3.3.8.1.0         while calcvalid(board, validmoves, curplr)
3.3.8.1.1           if showvalud then drawscreen(mergeboard(board,validmoves)) else
drawscreen (board)
3.3.8.1.2           move = getmove(curplr)
3.3.8.1.3.0         if validatemove(validmoves, move) then
3.3.8.1.3.1           board = makemove(board, move, curplr)
3.3.8.1.3.2           curplr = nextplr(curplr)
3.3.8.1.4.0         else
3.3.8.1.4.1           write out a blank line
3.3.8.1.4.2           Tell the user invalid move was made and press <ENTER> to continue
3.3.8.1.4.3           wait for <ENTER>
3.3.8.2           write out a blank line
3.3.8.3           say that there no valid moves left for the current player
3.3.8.4           wait for <ENTER>
3.3.8.5           curplr = nextplr(curplr)
3.3.8.6           ingame = calcvalid(board, validmoves, curplr)

3.3.9             drawscreen(board)
3.3.10            write out a blank line
3.3.11            say that its the end of the game and announce the winner
3.3.12            press Q to quit, any other key to play again...
3.3.13            wait for user input
3.4             do it until user input is Q
3.5           end Main Program
```

Pascal source code

```pascal
program othello;
{$APPTYPE CONSOLE}
uses
  SysUtils;

const
  BSIZE = 8;

type
  cell = (EMPTY, PLRA, PLRB, VALID);
  frame = array[1..BSIZE, 1..BSIZE] of cell;

  amove = record
   x: integer;
   y: integer;
  end;

var
  offset: array[1..8] of amove;


function initboard: frame;
  var
    a, b: integer;            // Loop Counters

  begin
    for a := 1 to BSIZE do
      for b := 1 to BSIZE do
        result[a,b] := EMPTY;

    result[(BSIZE div 2)+1,(BSIZE div 2)] := PLRA;
    result[(BSIZE div 2),(BSIZE div 2)+1] := PLRA;
    result[(BSIZE div 2),(BSIZE div 2)] := PLRB;
    result[(BSIZE div 2)+1,(BSIZE div 2)+1] := PLRB;
```

```
end; //initgame: frame


function drawcell(curcell: cell): string;
  begin
    case curcell of
      EMPTY: result := '   ';
      PLRA: result := ' # ';
      PLRB: result := ' ! ';
      VALID: result := ' . ';
    end; //case of

end; //drawcell(curcell): string


procedure drawscreen(board: frame);
  var
    a,b: integer;           // Loop Counters

  begin
    writeln;
    for a := 1 to BSIZE do                              // Column Numbers
      write('   ' + inttostr(a));                       //
    writeln;                                            //

    write(' ' + chr(201));
    for a :=1 to (BSIZE-1) do                           // Row Header
      write(chr(205)+chr(205)+chr(205)+chr(203));       //
    writeln(chr(205)+chr(205)+chr(205)+chr(187));       //   End of line

    for a := 1 to (BSIZE-1) do begin                    // Normal Rows
      write(a);                                         // Row Number
      for b := 1 to (BSIZE) do                          //
        write(chr(186)+ drawcell(board[b,a]));          //
      writeln(chr(186)+inttostr(a));                    //   End of line & Row Number

      write(' ' + chr(204));
      for b :=1 to (BSIZE-1) do                         // Row Spacer
        write(chr(205)+chr(205)+chr(205)+chr(206));     //
      writeln(chr(205)+chr(205)+chr(205)+chr(185));     //   End of line
    end;

    write(BSIZE);
    for b := 1 to (BSIZE) do                            // Last Row
        write(chr(186)+ drawcell(board[b,BSIZE]));      //
    writeln(chr(186)+inttostr(a));                      //   End of line & Row Number

    write(' ' + chr(200));
    for a :=1 to (BSIZE-1) do                           // Row Footer
      write(chr(205)+chr(205)+chr(205)+chr(202));       //
    writeln(chr(205)+chr(205)+chr(205)+chr(188));       //   End of line

    for a := 1 to BSIZE do                              // Column Numbers
      write('   ' + inttostr(a));                       //

    writeln;                                            // Fill the rest of the screen
    writeln;
    writeln;

end; //drawscreen(board)


procedure showinstructions;
var
  instructfile: TextFile;
  line: string;

begin
```

```
  AssignFile(instructfile, 'instructions.txt');
  Reset(instructfile);
  while NOT(EOF(instructfile)) do begin
    readln(instructfile, line);
    writeln(line);
  end;
  close(instructfile);
  writeln;
  write('Press <Enter> to begin the game...');
  readln;

end; // showinstructions()


function getmove(curplr: cell): amove;
var
  ch: string;               // Temporary Variable

begin
  writeln('Waiting for Player' + drawcell(curplr) + 'to move...');
  writeln;
  write('Enter Column Number: ');
    readln(ch);
    result.x := StrToIntDef(ch, 0);
  write('Enter Row Number: ');
    readln(ch);
    result.y := StrToIntDef(ch, 0);
end; //getmove(curplr): amove


function nextplr(curplr: cell): cell;           // Return the next / other player
begin
  case curplr of
    PLRA: result := PLRB;
    PLRB: result := PLRA;
  else
    result := PLRA;
  end; //case of

end; //nextplr(curplr): cell


function inbounds(pos: amove): boolean;          // Checks position is inside grid
begin
  result := ((pos.x > 0) AND (pos.y > 0) AND (pos.x <> BSIZE+1) AND (pos.y <> BSIZE+1));
end;


function calcvalid(board: frame; var validmoves: frame; curplr: cell): boolean;
var
  a,b,c : integer;    // Loop Counter
  pos: amove;         // Position
  foundvalid: boolean;


begin
  result := FALSE;                        // No valid moves found yet...
  for a := 1 to BSIZE do
    for b := 1 to BSIZE do begin
      foundvalid := FALSE;                        // No valid moves found yet...
      if board[a,b] = EMPTY then begin
        for c := 1 to 8 do begin
          pos.x := a;
          pos.y := b;
          if inbounds(pos) then
            if board[(a + offset[c].x), (b + offset[c].y)] = nextplr(curplr)  then begin
              while ((NOT foundvalid) AND inbounds(pos) AND (board[(pos.x + offset[c].x),
(pos.y + offset[c].y)] <> EMPTY)) do begin
                pos.x := pos.x + offset[c].x;
```

```
                pos.y := pos.y + offset[c].y;
                if ((board[pos.x, pos.y] = curplr) AND inbounds(pos)) then begin
                  validmoves[a,b] := VALID;
                  foundvalid := TRUE;
                end; //if ((board[pos.x, pos.y] = curplr) AND inbounds(pos))
              end; //while
            end; //board[(a + offset[c].x), (b + offset[c].y)] = nextplr(curplr)
        end; //for c

      end; //if board[a,b] = EMPTY

      if foundvalid then result := TRUE
        else
          validmoves[a,b] := EMPTY;   // Set if a valid move was found

    end; //for b

end; //calcvalid(board, var validmoves, curplr): boolean


function validatemove(validmoves: frame; move: amove): boolean;
begin
  if validmoves[move.x, move.y] = VALID then result := TRUE
    else result := FALSE;

end; //validatemove(validmoves, move): boolean


function makemove(board: frame; move: amove; curplr: cell): frame;
var
  c : integer;    // Loop Counter
  pos: amove;     // Position
  foundvalid: boolean;


begin
  // Simply just place the move in the board, but don't flip anything yet...
  board[move.x, move.y] := curplr;

  // Flip Counters
        for c := 1 to 8 do begin
          foundvalid := FALSE;
          pos.x := move.x;
          pos.y := move.y;
          if board[(move.x + offset[c].x), (move.y + offset[c].y)] = nextplr(curplr)  then
begin
            while ((NOT foundvalid) AND inbounds(pos) AND (board[(pos.x + offset[c].x),
(pos.y + offset[c].y)] <> EMPTY)) do begin
              pos.x := pos.x + offset[c].x;
              pos.y := pos.y + offset[c].y;
              if inbounds(pos) then
                if board[pos.x, pos.y] = curplr then begin
                  while ((pos.x <> move.x) OR (pos.y <> move.y)) do begin          // OK,
so we reset the new cell: who cares?
                    pos.x := pos.x - offset[c].x;
                    pos.y := pos.y - offset[c].y;
                    board[pos.x, pos.y] := curplr;
                  end; // while
                  foundvalid := TRUE;
                end; //if board[pos.x, pos.y] = curplr
            end; //while
          end; //if board[(move.x + offset[c].x), (move.y + offset[c].y)] =
nextplr(curplr)
        end; //for c

  result := board;

end; // makemove(board, move, curplr): frame
```

```
function getwinner(board: frame): cell;          // Count pieces owned by each playe and
return the winner.
var
  a,b: integer;           // Loop counters
  acnt, bcnt: integer;    // Number of pieces

begin
  acnt := 0;
  bcnt := 0;
  for a := 1 to BSIZE do
    for b := 1 to BSIZE do
      case board[a,b] of
        PLRA: acnt := acnt + 1;
        PLRB: bcnt := bcnt + 1;
      end; //case of

  if acnt > bcnt then result := PLRA;
  if acnt < bcnt then result := PLRB;
  if acnt = bcnt then result := EMPTY;

end; //getwinner(board): cell


function mergeboard(boarda: frame; boardb: frame): frame;  // Merge 2 boards
var
  a,b: integer;               // Loop counters

begin
  for a := 1 to BSIZE do
    for b := 1 to BSIZE do
      if boarda[a,b] <> EMPTY then result[a,b] := boarda[a,b]
        else result[a,b] := boardb[a,b];

end; //mergeboard(boarda, boardb): frame



// Main program

var
  board: frame;
  validmoves: frame;
  curplr: cell;
  move: amove;
  ch: char;
  ingame: boolean;
  showvalid: boolean;

begin
  // Instructions
  repeat
    write('Instructions? [Y/N] ');
    readln(ch);
  until (UpperCase(ch) = 'Y') OR (UpperCase(ch) = 'N');
  if UpperCase(ch) = 'Y' then showinstructions;

  repeat
    // Initialisation
    board := initboard;
    curplr := PLRA;
    offset[1].x := 0; offset[1].y :=-1; // North
    offset[2].x :=+1; offset[2].y :=-1; // North East
    offset[3].x :=+1; offset[3].y := 0; // East
    offset[4].x :=+1; offset[4].y :=+1; // South East
    offset[5].x := 0; offset[5].y :=+1; // South
    offset[6].x :=-1; offset[6].y :=+1; // South West
    offset[7].x :=-1; offset[7].y := 0; // West
```

```
    offset[8].x :=-1; offset[8].y :=-1; // North West
    ingame := TRUE;
    showvalid := FALSE;


    // Play game
    repeat
      write('Show Valid Moves? [Y/N] ');
      readln(ch);
    until (UpperCase(ch) = 'Y') OR (UpperCase(ch) = 'N');
    if UpperCase(ch) = 'Y' then showvalid := TRUE;

    while ingame do begin
      while calcvalid(board, validmoves, curplr) do begin          // Check that there are
some moves left
        if showvalid then drawscreen(mergeboard(board,validmoves)) // Draw the screen
          else drawscreen(board);
        move := getmove(curplr);                                   // Get a move from the
current player
        if validatemove(validmoves, move) then begin               // Check that a valid
move was entered
          board := makemove(board, move, curplr);                  // Make the move: place
counter & flip others
          curplr := nextplr(curplr);                               // Get the next player
        end
          else begin                                               // An invalid move was
made
            writeln;                                               // Let the current
player try again
            writeln('NOT a Valid Move: Press <ENTER> to try again.');
            readln;
          end; //else
      end; //while calcvalid(board, curplr)
      writeln;
      writeln('No Valid Moves for Player' + drawcell(curplr) + 'Press <ENTER> to move to
Player' + drawcell(nextplr(curplr)));
      readln;
      curplr := nextplr(curplr);
      ingame := calcvalid(board, validmoves, curplr);
    end; //while ingame


    // End Game
    drawscreen(board);
    writeln;
    writeln('End of Game! - Player' + drawcell(getwinner(board)) + 'won.');

    write('Press Q to quit, any other key to play again...');
    read(ch);
  until UpperCase(ch) = 'Q';


end.


{ Contents of instructions.txt

Reversi/Othello - Instructions
------------------------------

Reversi/Othello is a game for two players.

Playing the game
----------------

The players take turns at placing counters onto the playing board in an attempt
to eliminate the other player's counters from the board.
Players must place their counters in such a way that they sandwich a number of
their opponent's counters between two of their own in any direction. Their
opponent's counters are then "flipped over" to become the counters of the
```

current player.
If a player cannot move then they forfeit their move and control of the game
passes to the other player.

At the end of the game

The game ends when there are only counters belonging to one player on the board
or when the board is full. In the case where the board is full, the winner is
the player with the most counters.

)