OTHELLO – An Othello Game Playing Program

© 2019 Valentín Albillo

Abstract

OTHELLO is a program written in 2002 for the SHARP PC-1350/1360 pocket computers to play Othello (aka Reversi) vs. the user. It displays the playing board in the graphics screen, uses a graphic cursor to select moves and includes menus and full prompting.

Keywords: *OTHELLO, Reversi, game, graphic board, SHARP PC-1350/1360, pocket computer, BASIC*

1. Introduction

OTHELLO is a medium-sized (112 lines) *BASIC* program that I wrote in 2002 for the *SHARP PC-1350/1360* pocket computers and compatibles to play *Othello* (also known as *Reversi*) vs. the user. It displays the playing board in the graphics screen, allows selection of moves via either a graphic cursor or entering coordinates and includes menus and full prompting.

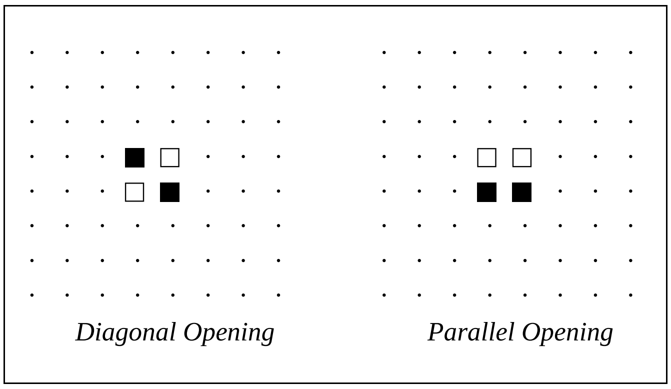
The program will check your moves for legality and will perform the tally and announce the winner when the game ends. It uses a simple strategy (either fixed or randomized) and will easily defeat a beginner but it's more intended to work as a non-trivial demo of the *PC-1350/1360* programming features and graphics capabilities.

The *SHARP PC-1360* is a 1987 pocket computer featuring a *SC61860 CMOS CPU* @ 768 Khz powered by two *CR-2032* cells, 138 Kb *ROM* and up to 64 Kb of *RAM* (2 slots for battery-backed 2 Kb-to-32 Kb *RAM* cards), a 4x24-char LCD display (150x32 fully addressable pixels) and 11-pin serial/15-pin printer connectors to interact with peripherals (mass storage, printers, plotters, etc.), in a small, light package (18.2x7.2x1.6 cm, 220 g.).

On the software side, it's got an extended version of *BASIC* with graphics and I/O commands. The *PC-1350* pictured above is very similar but with only one slot for *RAM* cards and lacking some extended commands.

2. Game Rules

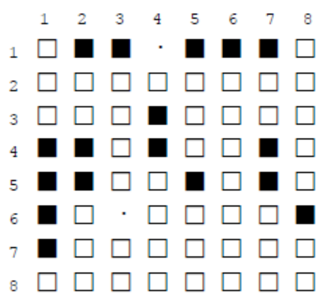
Othello is played on an 8x8 board. There are two standard openings, the *Diagonal Opening* and the *Parallel Opening*, as seen in the figure below:



The program plays the white pieces (☐) and the user plays the black pieces (■). To play a move, the player places one of his pieces in an *empty* location (represented by a dot) taking into account that:

- it must be adjacent to an enemy piece (either horizontally, vertically or diagonally)
- at least one enemy piece must be enclosed between the just placed piece and another piece of the same color, in any direction (horizontal, vertical or diagonal)
- all enemy pieces enclosed between them are flipped over and become of the capturer's color
- no empty locations can be enclosed, only a row of consecutive enemy pieces in any direction
- if more than one row is enclosed at the same time, all are flipped over
- the player can capture only when putting a new piece on the board, enemy pieces which are left enclosed because of other factors aren't captured

The following example will help to make it clear:



In this position, if White plays at ***14*** (row 1, col. 4) Black's pieces at ***12*** and ***13*** would be enclosed between the just played pìece and White's piece at ***11*** so they'd be flipped over and become White's. At the same time, Black pieces at ***15***, ***16*** and ***17*** would also be enclosed between the just played piece and White's piece at ***18***, so they'd be flipped over too and would become White's as well. Thus, five currently black pieces would become white.

On the other hand, if in this board position Black plays at ***63*** it would flip the nine white pieces at ***62***, ***53***, ***43***, ***33***, ***23***, ***64***, ***65***, ***66*** and ***67*** because there's another black piece at the end of each row of white pieces and none contain empty locations between the pieces.

3. Program Listing

Important note:

When entering the listing that follows, all text strings in messages and prompts must be keyed in exactly as shown, else serious misalignments and display misformatting may occur. For instance:

- *"Cannot pass.Select"* (*no* space between *"pass"*, *"."* and *"Select"*)
- *"I have no move.Pass"* (*no* space between *"move"*, *"."* and *"Pass"*)
- *"Setting up board..."* (no space between *"board"* and *"..."*, and exactly *three* dots)
- the statement `PRINT "I play";S;", flip";N` has *one* space between *","* and *"flip"*
- the similar statement `PRINT "You play";S;",flip";N` has *no* space between *","* and *"flip"*

In short, key all text strings *exactly* as shown. If misformatting occurs, you probably inserted o supressed a necessary space, comma, or dot. Check !

Also, the colon (**:**) after the line number must *not* be keyed in, it's automatically shown when listing lines.

3.1 Commented Program Listing

Specify no delay for `PRINT`, erase and initialize assorted variables, and call a subroutine to display the ***Options Menu*** and let the user toggle the available options as desired:

```
 1: REM *** OTHELLO - (c) Valentin Albillo, 2002
 5: "A" WAIT 0: CLEAR: Z=0, U=1, V=1, T=1, W=0, J=59*2: GOSUB 310
```

Upon return from the ***Options Menu***, show the program's banner and let the user know that *main initialization* will take place based on the selected options:

```
10: CLS: CURSOR 6: PRINT "* OTHELLO by VA *": USING "###"
15: LINE (36,0)-(149,7),X,BF: GOSUB 140: PRINT "Initializing ..."
```

Main initialization. Dimension and initialize the board array, the move offsets array, the strategy string array, and the small-font graphics numerals string array. If the user selected strategy randomization, calls a subroutine to do the randomization:

```
20: RESTORE: DIM B(99), M(7), C$(1)*60, D$(9)*8: C$(0)="": IF W GOSUB 400: GOTO 30
25: FOR I=1 TO 60: READ X: C$(0)=C$(0)+CHR$ X: NEXT I
30: FOR I=0 TO 7: READ M(I): NEXT I: FOR I=0 TO 9: READ D$(I): NEXT I
```

Sets up and draws the initial board and starting position, initializes the tally counters and either asks the user for a move or goes on to compute the program's move, depending on who plays first. If the user selected the option of cursor entry, calls a subroutine to allow that, else directly asks for the coordinates of the user's move, then goes to check them for legality:

```
35: GOSUB 240: P=2, Q=2: GOSUB 225: F=0, G=0: IF V=0 GOTO 145
40: GOSUB 140: IF T PRINT "Select your move": GOSUB 445: GOTO 50
45: S=0: INPUT "Move (11-88, 0)?";S: S=ABS INT S: GOTO 390
```

The user entered some move, go check it for legality:

```
50: IF S GOTO 80
```

The user didn't enter a move but selected to *pass* so a subroutine is called to check if the pass is legal (i.e.: if the user has no legal move indeed). If the user has any legal move, then passing is illegal, in which case if the user selected cursor entry a message is shown and the graphic cursor is placed at the first legal move found; if the user selected coordinates entry, a message is shown indicating the coordinates of the first legal move detected. In both cases the user will be required to enter another move:

```
 55: GOSUB 280: IF N=0 GOTO 70
 60: IF T GOSUB 140: WAIT J: PRINT "Cannot pass.Select": WAIT 0: H=S: GOSUB 455: GOTO 50
 65: GOSUB 140: WAIT J: PRINT "No, you can play";S: WAIT 0: GOTO 40
```

The user has no legal move so passing is legal. If previously the program had also passed, then neither player can play and the game is *ended* so it goes to a routine which shows a message to that effect, announces the winner and ends the game:

```
 70: IF F GOTO 185
```

Else, flags that the user has just passed and go on to compute the machine's move:

```
 75: F=1: GOTO 145
```

Checks the user's move for legality. If it isn't legal, goes to show a warning and request again a legal move. If it is, acknowledge the move and show the number of flipped pieces, then call a subroutine to update the scores and check for termination and if it returns (i.e.: the game's not yet finished), go to compute the program's move:

```
 80: GOSUB 140: PRINT "Checking your move": A=1, R=0: GOSUB 95: IF N=0 GOTO 235
 85: GOSUB 140: WAIT J: PRINT "You play";S;",flip";N: WAIT 0
 90: Q=Q+1, F=0: GOSUB 195: GOTO 145
```

Subroutine to check is a move is legal. If it is, the move is made for real, the board is updated and the number of flipped pieces is returned, unless we were just finding *any* legal move for the user to see if *passing* is legal, in which case the found move is discarded and it just returns the number of pieces which would be flipped:

```
 95: N=0: IF ABS B(S) RETURN
100: FOR I=0 TO 7: M=M(I), H=S+M: IF B(H)<>A GOTO 130
105: H=H+M: IF B(H)=A GOTO 105
110: IF B(H)<>-A GOTO 130
115: H=H-M: IF H=S GOTO 130
120: N=N+1: IF R LET I=7: NEXT I: RETURN
125: B(H)=-A: GOSUB 430: PSET (X-1,Y-2),X: GOTO 115
130: NEXT I: IF N LET A=-A, B(S)=A, H=S: GOSUB 435: A=-A
135: RETURN
```

Subroutine to clear the message line and reset the text cursor to the first position (note that there must be *exactly* 19 spaces between the quotes):

```
140: CURSOR 77: PRINT "                   ";: CURSOR 77: BEEP Z: RETURN
```

Computes the program's move using the strategy stored in `C$(0)`:

```
145: GOSUB 140: PRINT "My turn. Thinking..": K=1+12*(G<4)
150: S=ASC MID$(C$(0),K,1): IF ABS B(S) GOSUB 270: GOTO 175
155: A=-1, R=0: GOSUB 95: IF N=0 GOTO 170
```

A legal move has been found. The board and scores are updated and a check for termination is performed. If not yet finished, goes to ask the user for a move:

```
160: GOSUB 140: WAIT J: PRINT "I play";S;", flip";N: WAIT 0
165: P=P+1, F=0: GOSUB 195: GOSUB 270: GOTO 40
170: K=K+1
175: IF K<=LEN C$(0) GOTO 150
```

No legal move has been found so the program announces it *passes*. If the user also had no legal move, the program announces neither can play and goes to finish the game and announce the winner. Else it goes to ask the user for a move:

```
180: GOSUB 140: WAIT J: PRINT "I have no move.Pass": WAIT 0: BEEP 2*Z
185: IF F GOSUB 140: WAIT J: PRINT "Neither can play!": WAIT 0: GOTO 200
190: F=1: GOTO 40
```

Subroutine to update the scores and move number, and check for termination. If the game is indeed finished, it doesn't return but announces the winner (or a tie) and ends the program:

```
195: P=P-A*N, Q=Q+A*N, G=G+1: GOSUB 225: IF P+Q<>64 AND P*Q<>0 RETURN
200: GOSUB 140: WAIT
205: IF P>Q PRINT "* I WIN *"
210: IF P=Q PRINT "It is a tie!"
215: IF P<Q PRINT "You win ..."
220: CLS: CLEAR: END
```

Subroutine to update the scores' labels in the display:

```
225: GCURSOR (61,15): GPRINT D$(Q/10)+D$(Q-10*INT(Q/10))
230: GCURSOR (61,22): GPRINT D$(P/10)+D$(P-10*INT(P/10)): RETURN
```

Shows an *"Illegal move"* warning to the user and goes to ask the user for a legal move

```
235: GOSUB 140: WAIT J: PRINT "Illegal move!": WAIT 0: BEEP 2*Z: GOTO 40
```

Subroutine to set up the initial board. Draws all empty locations, places and draws the black and white pieces at their initial positions, and the small-font graphic labels *"You:"*, *"Me:"* :

```
240: GOSUB 140: PRINT "Setting up board...";
245: FOR X=3 TO 31 STEP 4: FOR Y=2 TO 30 STEP 4: PSET (X,Y): NEXT Y: NEXT X
250: M=45, N=55, B(44)=-1, B(54)=1, A=-1, H=44: GOSUB 435: IF U LET M=55, N=45
255: B(M)=-1, B(N)=1, H=M: GOSUB 435: A=1, H=54: GOSUB 435: H=N: GOSUB 435
260: GCURSOR (36,15): GPRINT "1C701C00705070007040700028282800787878780028"
265: GCURSOR (38,22): GPRINT "7C1830187C007C545C0028282800784848780028": RETURN
```

Subroutine for strategy bookkeeping:

```
270: L=LEN C$(0)-K, C$(1)=LEFT$(C$(0),K-1), C$(0)=RIGHT$(C$(0),L)
275: C$(0)=C$(1)+C$(0): RETURN
```

Subroutine to check if the user's *pass* is legal. It tries to quickly find *any* legal move for the user. Upon finding one, it returns with the number of pieces it would flip, which will be *0* only if it can't find any legal move:

```
280: GOSUB 140: PRINT "Checking your pass": K=1
285: S=ASC MID$(C$(0),K,1): IF ABS B(S) GOSUB 270: GOTO 300
290: A=1, R=1: GOSUB 95: IF N RETURN
```

```
295: K=K+1
300: IF K<=LEN C$(0) GOTO 285
305: RETURN
```

Subroutine to build anew and show the ***Options Menu***:

```
310: CLS: PRINT "# OTHELLO - Options Menu": LINE (0,0)-(149,7),X,BF
315: CURSOR 24: A$="N": IF Z LET A$="Y"
320: PRINT "1>Sounds :";A$;: A$="N": IF T LET A$="Y"
325: PRINT "  2>Cursor :";A$: A$="P": IF U LET A$="D"
330: PRINT "3>Opening:";A$;: A$="N": IF V LET A$="Y"
335: PRINT "  4>You 1st:";A$: A$="N": IF W LET A$="Y"
340: PRINT "5>Random :";A$;
```

Ask the user the number (1-5) of the option to toggle. If none selected, return (note that there must be *exactly* 11 spaces between the quotes):

```
345: CURSOR 85: PRINT "           ": CURSOR 85: INPUT "Option ?";A$: GOTO 355
350: RETURN
```

Toggle the selected option and redisplay the updated menu:

```
355: CURSOR 85: PAUSE "Changing..";
360: IF A$="1" LET Z=1-Z: GOTO 315
365: IF A$="2" LET T=1-T: GOTO 315
370: IF A$="3" LET U=1-U: GOTO 315
375: IF A$="4" LET V=1-V: GOTO 315
380: IF A$="5" LET W=1-W: GOTO 315
```

The user entered an illegal option number, show a warning and ask again:

```
385: CURSOR 85: PAUSE "Must be 1-5": GOTO 345
```

Extract the *X*, *Y* coordinates from the user input and check if they are legal *(1-8)*:

```
390: IF S LET X=INT(S/10), Y=S-10*X: IF X<1 OR X>8 OR Y<1 OR Y>8 GOTO 235
395: GOTO 50
```

Subroutine to randomize the strategy array. The resulting randomized strategy is left in `C$(0)`:

```
400: GOSUB 140: PRINT "Randomizing ...": FOR I=1 TO 60: READ B(I): NEXT I
405: RANDOM: N=4, X=12: GOSUB 420: X=56: GOSUB 420: N=8, X=4
410: GOSUB 420: X=16: GOSUB 420: GOSUB 420: GOSUB 420: GOSUB 420: GOSUB 420
415: FOR I=1 TO 60: C$(0)=C$(0)+CHR$ B(I), B(I)=0: NEXT I: RETURN
```

Subroutine to randomly exchange *N* specific entries in the strategy array:

```
420: FOR I=1 TO N: P=RND N+X, Q=RND N+X, M=B(P), B(P)=B(Q), B(Q)=M: NEXT I: X=X+N: RETURN
```

Subroutine to draw or erase the graphic cursor:

```
425: LINE (X,Y)-(X+4,Y+4),X,B: RETURN
```

Subroutine to extract the *X*,*Y* coordinates from variable `H`:

```
430: Y=INT(H/10)*4, X=4*H-10*Y: RETURN
```

Subroutine to draw either a white piece (□) or a black piece (■):

```
435: GOSUB 430: IF A=1 LINE (X,Y-1)-(X-2,Y-3),X,BF: RETURN
440: LINE (X,Y-1)-(X-2,Y-3),BF: RETURN
```

Subroutine to accept a move entered via the graphic cursor. First, select the first *empty* location as the initial position for the graphic cursor (not necessarily *legal* to move to):

```
445: FOR I=1 TO 8: FOR K=1 TO 8: H=10*I+K: IF B(H)=0 LET K=8, I=8
450: NEXT K: NEXT I
```

Draw the graphic cursor at the chosen position and wait for the user to press the direction keys in order to move it to the desired location:

```
455: GOSUB 430: X=X-3, Y=Y-4
460: GOSUB 425
465: A$=INKEY$: IF A$="" GOTO 465
```

Check if the direction key pressed is legal *(0-9)*. If not, do nothing and go wait for another key.

```
470: W=ASC A$-48: IF W<0 OR W>9 GOTO 465
```

Update the graphic cursor and if the user pressed *'0'* (no move, *pass*) return immediately:

```
475: GOSUB 425: IF W=0 LET S=0: RETURN
```

If the user pressed *'5'* (done, *play here*), compute the selected location from the cursor's coordinates and return:

```
480: IF W=5 LET S=INT(Y/4)*10+INT(X/4)+11: RETURN
```

The user pressed a direction key *(1-4, 6-9)*, call the appropriate subroutine to update the *X, Y* coordinates of the graphic cursor and go to redraw it and wait for another direction key to be pressed:

```
490: GOSUB W+490: GOTO 460
```

Subroutines to update the *X*,*Y* coordinates of the cursor, depending on the direction key pressed:

```
491: X=X-4*(X>3)
492: Y=Y+4*(Y<25): RETURN
493: X=X+4*(X<26), Y=Y+4*(Y<25): RETURN
494: X=X-4*(X>3): RETURN
496: X=X+4*(X<26): RETURN
497: X=X-4*(X>3)
498: Y=Y-4*(Y>3): RETURN
499: X=X+4*(X<26), Y=Y-4*(Y>3): RETURN
```

Data for the initial (non-randomized) strategy:

```
900: DATA 81,88,11,18,83,86,61,68,31,38,13,16,63,66,33,36,84,85,51,58
905: DATA 41,48,14,15,64,65,53,56,43,46,34,35,74,75,52,57,42,47,24,25
```

```
910: DATA 73,76,62,67,32,37,23,26,82,87,71,78,21,28,12,17,72,77,22,27
```

Data for the move offsets array:

```
915: DATA 1,9,10,11,-1,-9,-10,-11
```

Data for the small graphic numbers used in the score *("0","1",..."9")*:

```
920: DATA "7C447C00","487C4000","74545C00","44547C00","1C107C00"
925: DATA "5C547400","7C547400","04741C00","7C547C00","5C547C00"
```

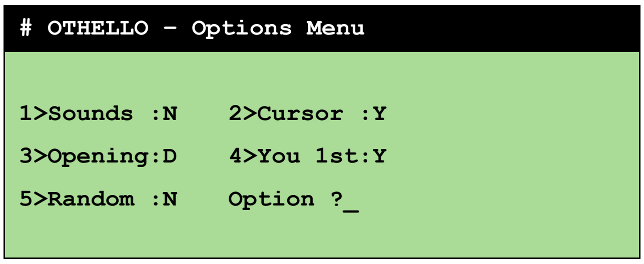
4. Variables Used

The program uses the variables listed and described in the following table:

Variables	Description
`A`	flags *User Moves* vs. *Program Moves*, also *User Pieces* vs. *Program Pieces*
`B(99)`	board array (100 board positions, including *out-of-board* borders)
`C$(1)*60`	strategy array (60 possible moves, plus 60 scratch elements)
`D$(9)*8`	binary definition for small-font graphic numerals *("0" to "9")*
`F`	flags if the previous move was a pass
`G`	scratch for move counting
`H`	scratch for move checking, also general purpose scratch
`I`	index for loops
`J`	delay for messages (default = 2 seconds)
`K`	index for strategy, also scratch for strategy bookkeeping
`L`	scratch for strategy bookkeeping
`M`	scratch for move offset, also general purpose scratch
`M(7)`	move offsets array (8 possible directions)
`N`	number of flipped pieces, also general purpose scratch
`P`	tally user's pieces, also used as scratch when initializing
`Q`	tally program's pieces, also used as scratch when initializing
`R`	flags checking and performing move vs. only checking move (test pass legality)
`S`	scratch for move entry, also general purpose scratch
`T`	toggle menu option *Cursor entry* vs. *Coordinates entry*
`U`	toggle menu option *Parallel opening* vs. *Diagonal opening*
`V`	toggle menu option *You move 1st* vs. *Program moves 1st*
`W`	toggle menu option *Randomized strategy* vs. *Static strategy*, also scratch
`X`	*X* coordinate, also scratch for data initialization
`Y`	*Y* coordinate
`Z`	toggle menu option *Sound* vs. *Silent*

5. Usage Instructions

In **RUN** Mode, proceed as follows to play a game. Press: **DEF** **A** → the *Options Menu* appears ...



```
# OTHELLO – Options Menu

1>Sounds :N    2>Cursor :Y
3>Opening:D    4>You 1st:Y
5>Random :N    Option ?_
```

... with the following options:

1. ***Sounds***: If ***Y*** then a beep will signal all messages. Default is ***N*** for no sounds.
2. ***Cursor***: If ***Y*** then your moves are entered by moving a graphic cursor over the location to move to. If ***N*** then you'll be prompted to enter the coords of the location to move to. Default is ***Y.***
3. ***Opening***: if ***D*** then your pieces are initially placed diagonally to each other. If ***P*** then they're placed in parallel. Default is ***D***, diagonal opening.
4. ***You 1st***: If ***Y*** you move first. If ***N*** the program makes the first move. Defaul is ***Y***, you move first.
5. ***Random***: if ***Y*** the program's strategy is somewhat randomized to avoid repeating games exactly. If ***N*** the same game is repeated if you play the same moves. Default is ***N***.

- If you want to change some option (1 to 5), press:

  1 - **5** **ENTER** → *Changing.., {the menu is updated to reflect the change}* → *Option ?_*

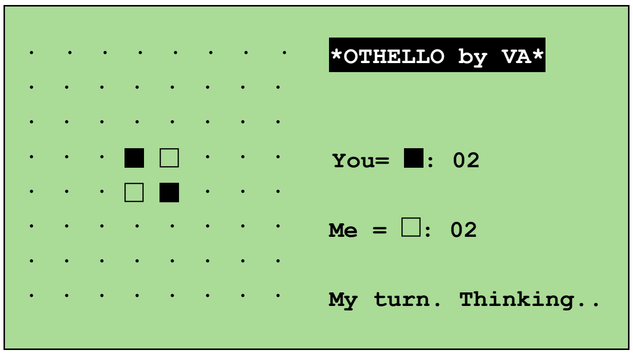
  (if you press something other than **1** - **5** then *"Changing.."* appears and "*Must be 1-5*" is displayed for 1 sec., then *"Option ?_"* is displayed again)

- If you want to change some other options, press **1** - **5** **ENTER** again. Once done with the changes press:

  ENTER → *"Initializing ..."* → *"Setting up board..."*

  and the playing board is displayed.

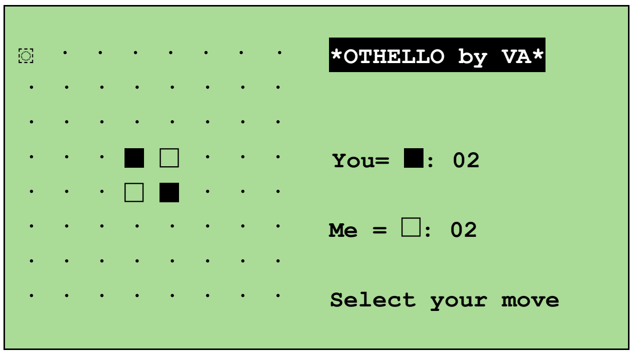
- If you selected the program to play first, it will immediately do so after displaying the board, like this:



```
. . . . . . . .    *OTHELLO by VA*
. . . . . . . .
. . . . . . . .
. . . ■ □ . . .    You= ■: 02
. . . □ ■ . . .
. . . . . . . .    Me = □: 02
. . . . . . . .
. . . . . . . .    My turn. Thinking..
```

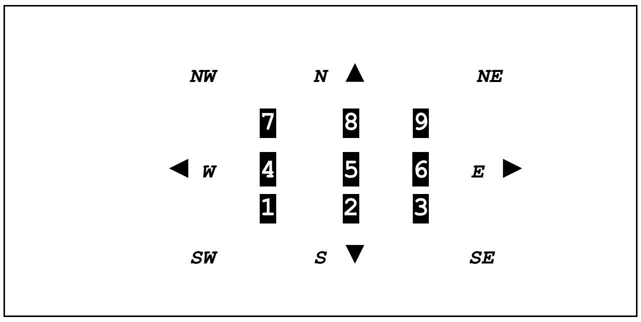
→ *"My turn. Thinking.."* → *"I play 65, flip 1"* → *"Select your move"*

- If you selected to play first, it will prompt for your move after displaying the board, like this:



```
◌ . . . . . . .     *OTHELLO by VA*
. . . . . . . .
. . . . . . . .
. . . ■ □ . . .     You= ■: 02
. . . □ ■ . . .
. . . . . . . .     Me = □: 02
. . . . . . . .
. . . . . . . .     Select your move
```

and the *graphic cursor* (◌) is at location ***11*** (these are the coordinates for row *8*, column *1*). You can use the numeric pad keys to move the cursor around and play your move:



- If you can't play a legal move press 0 to indicate that you ***pass***, in which case the program will display "*Checking your pass"* and if it finds *any* legal move for you it will display *"Cannot pass.Select"* and the cursor will be placed at a location where you *can* actually play a legal move (*any* legal move, not necessarily good or bad). You can then play that move by pressing 5 or else move the cursor to another legal location.

  If the pass is indeed warranted (indeed no legal moves for you) then the program will display *"My turn. Thinking.."* and will then proceed to compute its move or, if previously to your pass the program had *also* passed then none of the players can play a legal move, the game is ended and the program will proceed to do the tally and declare the winner (***"* I WIN *"***, ***"You win ...***" or ***"It is a tie!"***).

To select the current cursor's location to play your move simply press 5. The program will display *"Checking your move"* and will check your move for legality.

- if illegal, it will display: *"Illegal move!"* → *"Select your move"* , and you'll have to select another move or else *pass* (see above).

- if legal, it will display your move's coordinates and how many pieces flipped. e.g.: *"You play 46, flip 1"*

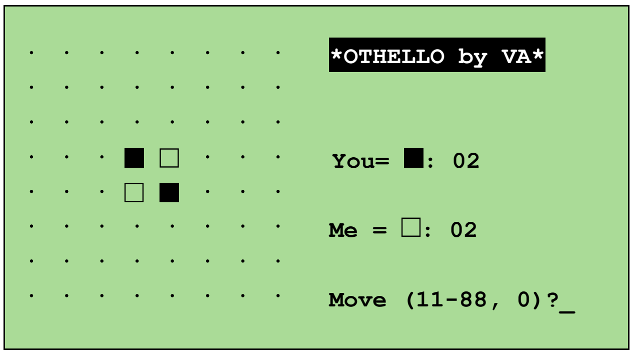
then the board and scores are updated and the program computes and displays its move (if it has any legal move):

→ *"My turn. Thinking.."* → *"I play 56, flip 2"*

- If the program has no legal moves it will display *“"I have no move.Pass"”* and will ask for your move except if in your previous move you had passed too, in which case the game is *ended* and the program will proceed to do the tally and declare the winner (or a tie).

The game proceeds in this fashion until either the board is full or neither player can play a legal move. Either way, the game is *ended* and the program will proceed to do the tally and declare the winner or a tie.

- If the **Cursor** option is set to ***N***, then the *“Select your move”* prompt is replaced by *“Move (11-88,0)?_”*



```
. . . . . . . .    *OTHELLO by VA*
. . . . . . . .
. . . . . . . .
. . . ■ □ . . .    You= ■: 02
. . . □ ■ . . .
. . . . . . . .    Me = □: 02
. . . . . . . .
. . . . . . . .    Move (11-88, 0)?_
```

and you must key in the row and column coordinates of the location where you want to move to, then proceed to press **ENTER** to actually play the move.

- The coordinates go from ***1*** *(leftmost column)* to ***8*** *(rightmost column)* for the columns, and likewise from ***1*** *(upper row)* to ***8*** *(bottom row)* for the rows, e.g.: the upper-left location is entered as ***11*** (i.e.: row ***1***, column ***1***) and the lower-right location is entered as ***88*** (row ***8***, column ***8***).

  You can also enter ***0*** to indicate you can’t play a legal move and so you must ***pass***. The pass is checked for legality and if illegal (you actually *do* have some legal move) it will display the message *“No,you can play XY”* where *XY* are the coordinates of some legal move for you, and you’ll have to enter either the suggested move or another legal one.

- Your move is checked for legality and everything proceeds as described above for cursor entry.

6. Examples

The following sample game can be useful to check that the program is correctly entered.

6.1 Sample game

In **RUN** Mode, proceed as follows to play the sample game. First of all, you must change some options:

DEF **A**	*{ the **Options Menu** is displayed}*
2 **ENTER**	*{change option to **Cursor: N** to input moves using row/col coordinates}*
4 **ENTER**	*{change option to **You 1st: N** to have the program make the first move}*

The remaining options should remain at their default values, namely: ***Sounds: N, Opening: D, Random: N***

Now exit the ***Options Menu*** and the program will initialize everything and proceed to play the first move:

ENTER → *“Initializing ...”* → *“Setting up board...”* → *“My turn. Thinking..”*

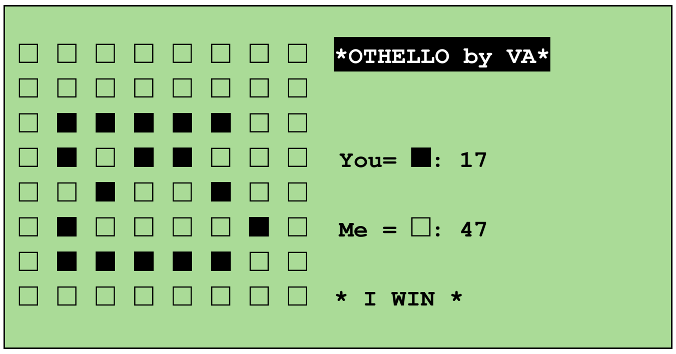
These are the coordinates of the sample game’s moves made by both the user and the program, in ***(user’s move, program’s move)*** format, with some pertinent comments:

`(--, 65)` *{the program moves first, at location* ***65****, i.e.: row 6, column 5}*

```
(46,33),(64,63),(43,66),(72,53),(67,81),(42,68),(75,36),(35,84),(86,51),(31,56)
(27,18),(57,85),(83,58),(76,41),(61,34),(62,74),(24,13),(25,16),(26,52),(32,47)
(23,14),(15,73),(17,37),(38,48),(78,82),(71,87),(12,11),
```

`( 0,21)` *{the user has no legal move and passes,* ***0****, then the program moves at location* ***21****}*

`(77,88),(22,28)` *{the game’s ended and the program adjudicates the winner based on the final score}*



```
□ □ □ □ □ □ □ □   *OTHELLO by VA*
□ □ □ □ □ □ □ □
□ ■ ■ ■ ■ ■ □ □
□ ■ □ ■ ■ □ □ □   You= ■: 17
□ □ ■ □ □ ■ □ □
□ ■ □ □ □ □ ■ □   Me = □: 47
□ ■ ■ ■ ■ ■ □ □
□ □ □ □ □ □ □ □   * I WIN *
```

Notes

1. The program uses a very simple positional strategy, either fixed or randomized. The fixed strategy will always play the same moves if you do as well so the same game would be repeated, while the randomized one will ensure different games each time. The sample game above should be run with option ***Random: N*** to duplicate the given moves and check that the program is correctly loaded but for actual play using the randomized strategy (***Random: Y***) is recommended.

2. The computing time for the program’s moves is ~ 15-20 sec. at most on a physical machine (not an emulator) for the first moves but decreases appreciably as the game progresses and the board gets fuller, taking just a few seconds near the end.

3. Depending on the model used and the available *RAM* an optional *RAM* card might be needed to run this program.

References

Martin Gardner	*New Mathematical Diversions*
Valentin Albillo (1980)	*Othello – A Computer Game for the 41C (PPC Technical Notes V1N2 pp.44-50)*
Valentin Albillo (1981)	*Reversi - Hewlett-Packard HP-41C User’s Library program #11019*
Valentin Albillo (2005)	*HP Article VA019 - 25 years of Othello*

Copyrights