Tic Tac Toe – Documentation

Terminology Used

i. **board:** the 3×3 matrix in which the 'X's and 'O's are marked

ii. **cell:** one of the 9 positions on the board

iii. **cell index:** an index number in the range 1-9 referring to each cell on the board (as shown in figure 1)

1	**2**	**3**
4	**5**	**6**
7	**8**	**9**

Figure 1

iv. **field:** one of the three rows (0, 1, 2), three columns (3, 4, 5) or two diagonals (6, 7) on completion of which the game ends in victory

v. **field index:** an index number in the range 0-7 referring to the eight fields defined

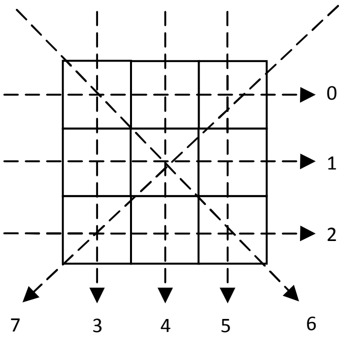


Figure 2

vi. **status:** the status of ownership of a field can be empty (no one occupying it), player (only player occupying it), computer (only computer occupying it) or none (both occupying it and hence belonging to no one)

Macros

- **MAINDIAG:** refers to the field index of the main diagonal (6)
- **BACKDIAG:** refers to the field index of the reverse diagonal (7)
- **CELL(cellind):** extracts the cell from the cell index
- **ROW(cellind):** extracts the row from the cell index

- **CELLIND(row,col):** inputs the row and column and outputs cell index

Structures and External Variables

- **char board[3][3]:** the 3×3 matrix which represents the game board
- **struct cellinfo:** contains two 4-bit variables that store the cell row and column information
- **enum control status[8]:** the status of the 8 fields of the game; may have any of 4 values (EMPTY, NONE, COMPUTER, PLAYER)
- **diff:** the difficulty level (ranging from 0-4)

Source Files

- **tictac.c:** main source file
- **ai.h:** contains the AI script for the computer's moves
- **display.h:** contains the display procedures
- **humint.h:** contains the human interaction procedures for getting input from the player
- **victory.h:** contains procedures for updating field status and deciding victory

Code Documentation

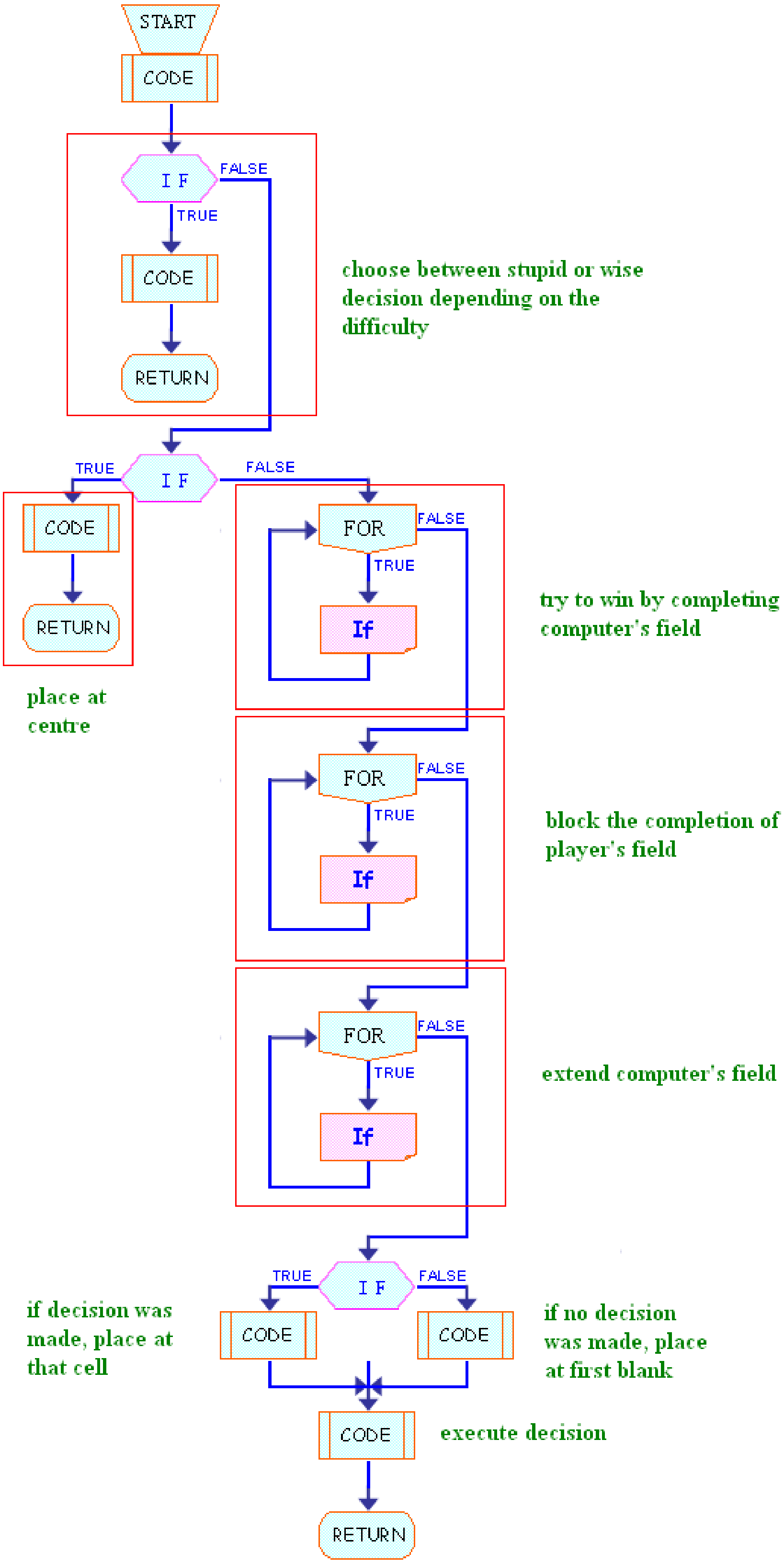
ai.h:

char masterbrain()

- *Purpose:* the AI function which decides which cell to mark
- *Arguments:* none
- *Return Value:* a namesake return value with no significance; char datatype is returned due to its minimal size

Algorithm:

- Choose between stupid or wise decision depending on the difficulty level.
- If the centre cell is blank, occupy it.
- If the computer can win in the next move, do it.
- If the player can win in the next move, block him/her.
- If computer cannot win or block, extend its own field.
- When none of the decisions mentioned above is made, place at the first blank cell.



choose between stupid or wise decision depending on the difficulty

place at centre

try to win by completing computer's field

block the completion of player's field

extend computer's field

if decision was made, place at that cell

if no decision was made, place at first blank

execute decision

int centreblank()

- *Purpose:* finds if the centre cell is blank
- *Arguments:* none
- *Return Value:* TRUE if centre cell is blank, FALSE if centre cell is occupied

int firstblank()

- *Purpose:* finds the first blank cell on the board
- *Arguments:* none
- *Return Value:* the cell index of the first blank cell encountered searching in the cell index order 1-9; if no blank cell is found, it returns the cell index of the cell (3,3) which is a hypothetical quantity.

void execution(struct cellinfo cell)

- *Purpose:* executes the decision of the AI script
- *Arguments:* struct cellinfo cell (representing the decision of the AI)
- *Return Value:* none

Algorithm:

- Mark appropriate cell.
- Update field control status.
- Update screen.

display.h:

void updatescreen()

- *Purpose:* refreshes the game display screen
- *Arguments:* none
- *Return Value:* none

Algorithm:

- Clear the game screen.
- Print game details such as game title and difficulty level.
- Print the board.

void printgamedetails()

- *Purpose:* prints the game details such as game title and difficulty level
- *Arguments:* none
- *Return Value:* none

Algorithm:

- Print game title.
- Print player and computer markers.
- Print difficulty level.

void printboard()

- *Purpose:* prints the current state of the board
- *Arguments:* none
- *Return Value:* none

Algorithm:

- Read the board and print the characters in the appropriate colors as given below:
    - ‘X’ – Red
    - ‘O’ – Blue
    - ‘X’ + 1 – Red blinking X
    - ‘O’ + 1 – Blue blinking O
    - Null – the cell index of the cell in light cyan
- Set textcolor back to light gray.

humint.h:

void awaitplayerresponse()

- *Purpose:* receives user input and pass it on for validation
- *Arguments:* none
- *Return Value:* none

Algorithm:

- Print message prompting user for cell index no. to cross.
- Scan the input from the user.
- Clear stdin.
- Validate the user input for errors.

void validateinput(int input)

- *Purpose:* validates user input for errors
- *Arguments:* int input(representing the user input)
- *Return Value:* none

Algorithm:

- If the cell index is outside the range 1-9 or the cell is already marked,
    - Print appropriate error message.
    - Increment stupiditycount.
    - If stupiditycount is equal to 3, raise stupidity alarm. After alarm, revert stupiditycount back to 0.
    - Get user input again.
- If no errors are found,
    - Mark appropriate cell.
    - Update field control status.
    - Update user screen.

void stupidityalarm()

- *Purpose:* displays the stupidity alarm messages
- *Arguments:* none
- *Return Value:* none

victory.h:

void updatecontrol(struct cellinfo cell, int wru)

- *Purpose:* calls the updatestatus function with properly processed arguments
- *Arguments:* struct cellinfo cell (representing the cell marked), int wru (saying whether the cell was marked by the player or the computer)
- *Return Value:* none

Algorithm:

- Update the status of the row field the cell belongs to.
- Update the status of the column field the cell belongs to.
- If the cell lies on the main diagonal, update its status.
- If the cell lies on the reverse diagonal, update its status.

void updatestatus(int fieldind, int wru)

- *Purpose:* updates the status of a field
- *Arguments:* int fieldind (the field index of the field whose status is to be updated), int wru (saying whether the cell was marked by the player or the computer)
- *Return Value:* none

Algorithm:

- If the field was previously empty, hand over ownership to wru.
- If the field was previously owned by the other player, change ownership to NONE.

void checkforvictory()

- *Purpose:* checks whether the player or computer has won, prints the appropriate message on the screen and exits execution of the program
- *Arguments:* none
- *Return Value:* none

Algorithm:

- If the status of a field is either COMPUTER or PLAYER and if the no. of nulls in that field is 0, then victory has been achieved by either the computer or the player respectively.
- If victory has been achieved, take the following action:
    - Add blink effect to the winning fields.
    - Update screen.
    - Print appropriate message.
    - Exit execution of the program.

int fieldfunc(int fieldind, char mode)

- *Purpose:* interacts with the fields and does three different operations (counting nulls, finding blanks, adding blink effect) depending on the mode
- *Arguments:* int fieldind (the field index of the field to interact with), char mode (the mode representing the operation to perform)
- *Return Value:* none

Algorithm:

- If mode is ‘c’, return the no. of nulls in the field.
- If mode is ‘b’, return the cell index of a blank cell in the field.
- If mode is ‘v’, add blink effect to the cells of the winning field.
- The default return(63) has no significance and is just to suppress the unreachable code warning.

Difficulty Levels:

In levels 0 to 3, the player has 20%, 40%, 60%, 80% chance of winning respectively.

The level 4 is invincible, as far as our testing has shown. If you do beat it, please do inform us at

theroarofthedragon@gmail.com