Code

module datahandling

    implicit none

    character(1), dimension(:, :), allocatable :: board

    integer, dimension(10,2) :: undoMove

    integer :: undoindex=0

    integer :: columns,rows

    integer :: curcol,currow

    integer :: temp\_column, temp\_row

    integer :: difficulty=4

    integer :: user\_command

    character :: tokken

    Logical :: is\_on,firstPlayer

    contains

    !settings for game

    recursive subroutine settings

        print \*,"for setting, enter no to make win change ,(default = 4)"

        print \*,"enter in range(3 to 6), out of range consider as default"

        read \*,user\_command

        if(user\_command>=3 .and. user\_command<=6) then

            difficulty=user\_command

        else

            difficulty=4

        end if

        print \*,"difficulty is ",difficulty

        print \*,"for dimention change ,Enter 0"

        print \*,"for default(7x6), enter any other number"

        read \*,user\_command

        if (user\_command==0) then

            print \*,"enter no of columns u want"

            read \*,columns

            print \*,"enter no of rows u want"

            read \*,rows

            if(rows<difficulty .or. columns<difficulty) then

                print \*,"rows and column must be greater then 4"

                print \*,"invalid"

                call settings

            end if

        else

            columns=7

            rows=6

        end if

        end subroutine settings

    !initiate game before start

    subroutine initiate

        is\_on=.TRUE.

        allocate(board(columns, rows));

        board = " "

        end subroutine initiate

    !display screen

    subroutine display

        do temp\_row = rows, 1, -1

            do temp\_column = 1, columns

                print '(3a1,$)', '[', board(temp\_column, temp\_row), ']'

            end do

        print \*

        end do

        do temp\_column = 1, columns

            print "('[', I0, ']',$)", temp\_column

        end do

        print \*

        end subroutine display

    subroutine insertinto(value)

        integer ,intent(in) :: value

        loop1 : do temp\_row=rows,1,-1

                if(board(value,rows)/=" ") then

                    print \*,"already filled"

                    exit loop1

                else if(board(value,temp\_row)/=" ") then

                    board(value,temp\_row+1)=tokken

                    curcol=value

                    currow=temp\_row+1

                    undoindex=undoindex+1

                    undoMove(undoindex,1)=curcol

                    undoMove(undoindex,2)=currow

                    firstPlayer = .NOT. firstPlayer

                    exit loop1

                else if(temp\_row==1) then

                    board(value,temp\_row)=tokken

                    curcol=value

                    currow=temp\_row

                    undoindex=undoindex+1

                    undoMove(undoindex,1)=curcol

                    undoMove(undoindex,2)=currow

                    firstPlayer = .NOT. firstPlayer

                    end if

                end do loop1

    end subroutine insertinto

    subroutine userInput()

        if(firstPlayer) then

            tokken="o"

        else

            tokken="\*"

            end if

        print \*,"its ", tokken," turn, enter move (enter -1 for undo) (enter 0 of exit)"

        read \*,user\_command

        if(user\_command==-1) then

            if(undoindex>=1) then

                print \*,"Undo"

                board(undoMove(undoindex,1),undoMove(undoindex,2))=" "

                undoindex=undoindex-1

                firstPlayer = .NOT. firstPlayer

            else

                print \*,"undo not possible"

            end if

            call display

        !vailid user\_command loop1

        else if (user\_command==0) then

            print \*,"exit game"

            is\_on=.False.

        else if(user\_command>columns .or. user\_command<1) then

            print \*,"invalid move"

        else

            call insertinto(user\_command)

            call display

            end if

        end subroutine userInput

    end module datahandling

program show

    use datahandling

    integer :: t

    call settings

    call initiate

    call display

    firstPlayer = .TRUE.

    kloop: do while(is\_on)

        call userInput()

        if(is\_on) then

            if(func(curcol,currow)) then

                print \*,"Win ",tokken

                exit kloop

            end if

        else

            print \*,"thankyou"

        end if

    end do kloop

    contains

    Logical function func(column\_,row\_)

        integer, intent(in) :: column\_,row\_

        Logical :: outp

        integer :: count

        count=0

        func = .False.

        !verticle

        do temp\_column=1,columns

            count=0

            do temp\_row=1,rows

                if(board(temp\_column,temp\_row)==tokken) then

                    count=count+1

                    if(count>=difficulty) then

                        func=.TRUE.

                    end if

                else

                    count=0

                end if

            end do

        end do

        !horizontle

        do temp\_row=1,rows

            count = 0

            do temp\_column=1,columns

                if(board(temp\_column,temp\_row)==tokken) then

                    count=count+1

                    if(count>=difficulty) then

                        func=.TRUE.

                    end if

                else

                    count=0

                end if

            end do

        end do

        ! slash

        count=0

        temp\_column=column\_-difficulty

        do temp\_row=row\_-difficulty,row\_+difficulty

            if(board(temp\_column,temp\_row)==tokken) then

                count=count+1

                if(count>=difficulty) then

                    func=.TRUE.

                end if

            else

                count=0

            end if

            temp\_column=temp\_column+1

        end do

        ! backslash

        count=0

        temp\_column=column\_-difficulty

        do temp\_row=row\_+difficulty,row\_-difficulty,-1

            if(board(temp\_column,temp\_row)==tokken) then

                count=count+1

                if(count>=difficulty) then

                    func=.TRUE.

                end if

            else

                count=0

            end if

            temp\_column=temp\_column+1

        end do

        !is draw?

        is\_on=.False.

        do temp\_column=1,columns

            if(board(temp\_column,rows)==" ") then

                is\_on=.true.

            end if

        end do

        if((.not. is\_on) .and. (.not. func)) then

            print \*,"game end, its board draw"

        end if

        end function func

end program show

Explanation

The provided Fortran code implements a simple game where players take turns placing tokens on a game board, aiming to connect a specified number of tokens in a row horizontally, vertically, or diagonally. Let's go through the key components and functionalities:

**Module datahandling:**

* **Variables**:
  + **board**: A character array representing the game board.
  + **undoMove**: An array storing the coordinates of moves for undo functionality.
  + **undoindex**: An integer tracking the index of the last move for undo functionality.
  + **columns**, **rows**: Integers representing the dimensions of the game board.
  + **curcol**, **currow**: Integers representing the current position of the token.
  + **temp\_column**, **temp\_row**: Temporary integers used for iterating through the board.
  + **difficulty**: An integer specifying the number of tokens needed to win.
  + **user\_command**: An integer storing user input for moves and settings.
  + **tokken**: A character representing the current player's token.
  + **is\_on**: A logical variable indicating if the game is ongoing.
  + **firstPlayer**: A logical variable indicating the current player.
* **Subroutines**:
  + **settings**: Sets up the initial game settings such as difficulty and board dimensions.
  + **initiate**: Initializes the game board.
  + **display**: Displays the current state of the game board.
  + **insertinto**: Inserts a token into the specified column.
  + **userInput**: Handles user input for moves and undo functionality.

**Main Program show:**

* **Execution**:
  + Calls the **settings** subroutine to set up the game.
  + Calls the **initiate** subroutine to initialize the game board.
  + Executes a loop to allow players to take turns until the game ends.
  + Calls the **userInput** subroutine to handle player input.
  + Calls the **func** function to check for winning conditions after each move.
* **Function**:
  + **func**: Checks for winning conditions (vertical, horizontal, diagonal) and draws.

**Explanation:**

* Players can set the game difficulty and board dimensions at the beginning.
* Players take turns entering their moves, and they can undo their last move if needed.
* The game continues until a player wins, the board is filled (draw), or the player decides to exit.
* The **func** function checks for winning conditions by examining rows, columns, and diagonals.
* If no winning condition is met and the board is full, the game ends in a draw.

Overall, the code provides a basic framework for a two-player board game with undo functionality and adjustable difficulty. It utilizes simple logical and looping structures to manage the game flow and conditions.