Code:

module subprograms

    implicit none

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

    integer :: columns,rows,current\_column,current\_row,x,y,set\_to\_make,input,total\_turn,computerIQ

    character :: tokken

    Logical :: game\_status,player !player is reffered as first player

    contains

    !change\_setting for game

    recursive subroutine change\_setting

        print \*,"enter computer IQ :"

        print\*,"press 0 for 2 player"

        read \*,computerIQ

        if(computerIQ<0 .or. columns>7) then

            print \*,"invalid"

            call change\_setting

        end if

        set\_to\_make=4

        print \*,"you have to make set of  ",set\_to\_make

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

        print \*,"for default(7x6), enter 1"

        print \*,"it should be greater than or equal to 4x4"

        read \*,input

        if (input==0) then

            print \*,"enter no of columns :"

            read \*,columns

            print \*,"enter no of rows :"

            read \*,rows

            if(rows<set\_to\_make .or. columns<set\_to\_make) then

                print \*,"invalid"

                call change\_setting

            end if

        else if(input==1) then

            columns=7

            rows=6

        else

            print \*,"invalid"

            call change\_setting

        end if

        end subroutine change\_setting

    !initialize game before start

    subroutine initialize

        game\_status=.TRUE.

        allocate(board(columns, rows));

        board = " "

        total\_turn=0

        end subroutine initialize

    !display screen

    subroutine display

        do y = rows, 1, -1

            do x = 1, columns

                print '(3a1,$)', '[', board(x, y), ']'

            end do

        print \*

        end do

        do x = 1, columns

            print "('[', I0, ']',$)", x

        end do

        print \*," "

        print \*,"------------------------------------------------------------------------------"

        end subroutine display

    !take input from user

    subroutine inputFromUser()

        REAL :: randomValue

        if(player) then

            tokken="\*"

            print \*,"its [\*] turn, enter move (enter 0 of exit)"

            read \*,input

            if (input==0) then

                print \*,"exit game"

                game\_status=.False.

            else if(input>columns .or. input<1) then

                print \*,"invalid move"

            else

                check1 : do y=rows,1,-1

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

                    print \*,"columns is already filled"

                    exit check1

                else if(board(input,y)/=" ") then

                    board(input,y+1)=tokken

                    current\_column=input

                    current\_row=y+1

                    total\_turn=total\_turn+1

                    player = .NOT. player

                    exit check1

                else if(y==1) then

                    board(input,y)=tokken

                    current\_column=input

                    current\_row=y

                    total\_turn=total\_turn+1

                    player = .NOT. player

                    end if

                end do check1

                call display

                end if

        else if(computerIQ==0) then

            tokken="o"

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

            read \*,input

            if (input==0) then

                print \*,"exit game"

                game\_status=.False.

            else if(input>columns .or. input<1) then

                print \*,"invalid move"

            else

                check2 : do y=rows,1,-1

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

                    print \*,"columns is already filled"

                    exit check2

                else if(board(input,y)/=" ") then

                    board(input,y+1)=tokken

                    current\_column=input

                    current\_row=y+1

                    total\_turn=total\_turn+1

                    player = .NOT. player

                    exit check2

                else if(y==1) then

                    board(input,y)=tokken

                    current\_column=input

                    current\_row=y

                    total\_turn=total\_turn+1

                    player = .NOT. player

                    end if

                end do check2

                call display

            end if

        else

            tokken="o"

            if(total\_turn<4) then

                CALL RANDOM\_NUMBER(randomValue)

                input = 1 + INT((columns - 2) \* randomValue)

                check3 : do y=rows,1,-1

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

                    print \*,"columns is already filled"

                    exit check3

                else if(board(input,y)/=" ") then

                    board(input,y+1)=tokken

                    current\_column=input

                    current\_row=y+1

                    total\_turn=total\_turn+1

                    player = .NOT. player

                    exit check3

                else if(y==1) then

                    board(input,y)=tokken

                    current\_column=input

                    current\_row=y

                    total\_turn=total\_turn+1

                    player = .NOT. player

                    end if

                end do check3

                call display()

            else

                call computer()

            end if

        end if

        end subroutine inputFromUser

    !computer input

    subroutine computer()

        integer :: leftpremove

        integer, dimension(columns) :: points

        character(1), dimension(columns,rows) :: a1

        integer :: poin,y1

        integer :: move=4

        integer :: inp

        Logical :: temp

        points=0

        a1=board

        leftpremove=computerIQ

        do inp=1,columns

            !check4 loop check the top empty place in column and put input

            check4 : do y1=rows,1,-1

                poin=0

                if(a1(inp,rows)/=" ") then

                    points(inp)=-1\*(columns\*\*columns)

                    exit check4

                else if(board(inp,y1)/=" ") then

                    a1(inp,y1+1)="o"

                    call winning(a1,temp,inp,y1+1,.True.)

                    if(temp) then

                        poin=columns\*\*leftpremove

                    else

                        call bestMove(poin,a1,.False.,leftpremove-1)

                    end if

                    points(inp)=poin

                    a1(inp,y1+1)=" "

                    exit check4

                else if(y1==1) then

                    a1(inp,y1)="o"

                    call winning(a1,temp,inp,y1,.True.)

                    if(temp) then

                        poin=columns\*\*leftpremove

                    else

                        call bestMove(poin,a1,.False.,leftpremove-1)

                    end if

                    points(inp)=poin

                    a1(inp,y1)=" "

                end if

            end do check4

        end do

        do inp=1,columns

            ! print \*,inp,points(inp)

            if(points(move)<points(inp)) then

                move=inp

            end if

        end do

        ! print \*,"move is ",move

        inp=move

        check5 : do y1=rows,1,-1

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

                exit check5

            else if(board(inp,y1)/=" ") then

                board(inp,y1+1)=tokken

                current\_column=inp

                current\_row=y1+1

                total\_turn=total\_turn+1

                player = .NOT. player

                exit check5

            else if(y1==1) then

                board(inp,y1)=tokken

                current\_column=inp

                current\_row=y1

                total\_turn=total\_turn+1

                player = .NOT. player

                end if

        end do check5

        call display

    end subroutine computer

    recursive subroutine bestMove(points,a2,tok,leftpremove)

        integer, intent(in) :: leftpremove

        character(1), dimension(1:columns,1:rows), intent(in) :: a2

        character(1), dimension(1:columns,1:rows) :: a1

        integer, intent(inout) :: points

        Logical ,intent(in) ::tok

        Logical :: temp

        integer :: inp

        integer :: y1=0

        a1=a2

        if(leftpremove<=0) then

            !print \*,"move selected ",points

            return

        else if(tok) then

            do inp=1,columns

            check3 : do y1=rows,1,-1

                if(a1(inp,rows)/=" ") then

                    ! points=points-100

                    exit check3

                else if(a1(inp,y1)/=" ") then

                    a1(inp,y1+1)="o"

                    call winning(a1,temp,inp,y1+1,tok)

                    if(temp) then

                        points=points+columns\*\*leftpremove

                    else

                        call bestMove(points,a1,.False.,leftpremove-1)

                    end if

                    a1(inp,y1+1)=" "

                    exit check3

                else if(y1==1) then

                    a1(inp,y1)="o"

                    call winning(a1,temp,inp,y1,tok)

                    if(temp) then

                        points=points+columns\*\*leftpremove

                    else

                        call bestMove(points,a1,.False.,leftpremove-1)

                    end if

                    a1(inp,y1)=" "

                    exit check3

                end if

            end do check3

            end do

        else

            do inp=1,columns

                check4 :do y1=rows,1,-1

                    if(a1(inp,rows)/=" ") then

                        ! points=points-100

                        exit check4

                    else if(a1(inp,y1)/=" ") then

                        a1(inp,y1+1)="\*"

                        call winning(a1,temp,inp,y1+1,tok)

                        if(temp) then

                            points=points-columns\*\*leftpremove

                        else

                            call bestMove(points,a1,.True.,leftpremove-1)

                        end if

                        a1(inp,y1+1)=" "

                        exit check4

                    else if(y1==1) then

                        a1(inp,y1)="\*"

                        call winning(a1,temp,inp,y1,tok)

                        if(temp) then

                            points=points-columns\*\*leftpremove

                        else

                            call bestMove(points,a1,.True.,leftpremove-1)

                        end if

                        a1(inp,y1)=" "

                        exit check4

                    end if

                end do check4

            end do

        end if

    end subroutine bestMove

    subroutine winning(board,winn\_check,column\_,row\_,tok)

        character(1), dimension(columns,rows), intent(in) :: board

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

        Logical,intent(out) :: winn\_check

        Logical, intent(in) :: tok

        character(1) :: tokken1

        integer :: count

        count=0

        winn\_check = .False.

        if(tok) then

            tokken1="o"

        else

            tokken1="\*"

        end if

        !verticle

        do y=row\_,1,-1

            if(board(column\_,y)==tokken1) then

                count=count+1

                if(count>=set\_to\_make) then

                    winn\_check=.TRUE.

                end if

            else

                count=0

            end if

        end do

        !horizontle

        count=0

        do x=1,columns

            if(board(x,row\_)==tokken1) then

                count=count+1

                if(count>=set\_to\_make) then

                    winn\_check=.TRUE.

                end if

            else

                count=0

            end if

        end do

        ! /slash

        count=0

        x=column\_-set\_to\_make

        do y=row\_-set\_to\_make,row\_+set\_to\_make

            if((x>=1 .and. x<=columns) .and. (y>=1 .and. y<=rows)) then

                if(board(x,y)==tokken1) then

                    count=count+1

                    if(count>=set\_to\_make) then

                        winn\_check=.TRUE.

                    end if

                else

                    count=0

                end if

            end if

            x=x+1

        end do

        ! backslash

        count=0

        x=column\_-set\_to\_make

        slash2 : do y=row\_+set\_to\_make,row\_-set\_to\_make,-1

            if((x>=1 .and. x<=columns) .and. (y>=1 .and. y<=rows)) then

                if(board(x,y)==tokken1) then

                    count=count+1

                    if(count>=set\_to\_make) then

                        winn\_check=.TRUE.

                    end if

                else

                    count=0

                end if

            end if

            x=x+1

        end do slash2

        !draw check

        if(total\_turn>=columns\*rows) then

            print \*,"game draw";

            game\_status=.False.

        end if

    end subroutine winning

    end module subprograms

program show

    use subprograms

    integer :: t

    call change\_setting

    call initialize

    call display

    player = .TRUE.

    kloop: do while(game\_status)

        call inputFromUser()

        if(winn\_check(current\_column,current\_row)) then

            print \*,"Winner ",tokken

            exit kloop

        end if

        end do kloop

    contains

    Logical function winn\_check(column\_,row\_)

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

        Logical :: outp

        integer :: count

        count=0

        winn\_check = .False.

        !verticle

        do y=row\_,row\_-set\_to\_make,-1

            if(board(column\_,y)==tokken) then

                count=count+1

                if(count>=set\_to\_make) then

                    winn\_check=.TRUE.

                end if

            else

                count=0

            end if

        end do

        !horizontle

        count=0

        do x=1,columns

            if(board(x,row\_)==tokken) then

                count=count+1

                if(count>=set\_to\_make) then

                    winn\_check=.TRUE.

                end if

            else

                count=0

            end if

        end do

        ! slash

        count=0

        x=column\_-set\_to\_make

        do y=row\_-set\_to\_make,row\_+set\_to\_make

            if(board(x,y)==tokken) then

                count=count+1

                if(count>=set\_to\_make) then

                    winn\_check=.TRUE.

                end if

            else

                count=0

            end if

            x=x+1

        end do

        ! backslash

        count=0

        x=column\_-set\_to\_make

        do y=row\_+set\_to\_make,row\_-set\_to\_make,-1

            if(board(x,y)==tokken) then

                count=count+1

                if(count>=set\_to\_make) then

                    winn\_check=.TRUE.

                end if

            else

                count=0

            end if

            x=x+1

        end do

        !draw check

        if(total\_turn>=columns\*rows) then

            print \*,"game draw";

            game\_status=.False.

        end if

        end function winn\_check

end program show

**Explain:**

**Module subprograms:**

This module contains all the subroutines and functions necessary for the Connect Four game.

* **Variables**:
  + **board**: a 2D array representing the game board.
  + **columns**, **rows**: integers defining the dimensions of the game board.
  + **current\_column**, **current\_row**: integers tracking the current position of the player's move.
  + **x**, **y**: loop indices.
  + **set\_to\_make**: integer specifying the number of tokens needed to make a set.
  + **input**: integer used for user input.
  + **total\_turn**: integer tracking the total number of turns in the game.
  + **computerIQ**: integer determining the intelligence level of the computer player.
  + **tokken**: character representing the token used by players.
  + **game\_status**: logical variable indicating whether the game is ongoing.
  + **player**: logical variable representing the first player.

**Subroutines and Functions:**

1. **change\_setting**:
   * Allows users to set up game parameters such as the computer's IQ level and the dimensions of the game board.
   * Uses recursion to handle invalid input and ensure proper configuration.
2. **initialize**:
   * Initializes the game by allocating memory for the game board and setting the game status to true.
3. **display**:
   * Displays the current state of the game board, including tokens placed by players.
4. **inputFromUser**:
   * Handles user input for placing tokens on the game board.
   * Implements logic for both single-player and two-player modes.
   * Handles computer moves based on the selected intelligence level.
5. **Computer:**

* Handle computer moves based on the selected intelligence.
* First few moves are random but after 4 moves, algorthm start.

1. **bestmove:**

* recursively check for best move and up to computerIQ moves ahead which make it approx. impossible to win.

1. **Other Subroutines**:
   * Additional subroutines handle various aspects of the game, such as determining the best computer move, checking for win conditions, etc.

**Main Program show:**

* Initiates the game by calling the **change\_setting**, **initialize**, and **display** subroutines.
* Implements a loop for game turns, where players take turns placing tokens until a win condition is met or the game is terminated.
* The loop also checks for win conditions and ends the game if necessary.

**Connect Four Game Summary:**

1. Customizable Board Size:

* Players have the flexibility to adjust the dimensions of the game board according to their preferences.
* The **change\_setting** subroutine allows users to specify the number of rows and columns, catering to different gameplay experiences.
* Customizable board sizes add variability and enable players to explore diverse strategies.

2. Flexible Computer Intelligence Level:

* Users can set the intelligence level of the computer opponent using the **computerIQ** parameter.
* The game offers a range of intelligence levels, allowing players to select the desired level of challenge.
* Higher **computerIQ** settings result in more strategic and challenging gameplay, making it increasingly difficult for players to win against the computer.

3. Advanced Computer Algorithm:

* The computer player employs sophisticated algorithms, including backtracking and tree algorithms, to analyze potential moves.
* Recursive functions are utilized to explore multiple future move possibilities and select the most optimal course of action.
* The advanced algorithm enhances the computer's decision-making capabilities, ensuring competitive and engaging gameplay.

4. Strategic Point System:

* A strategic point-based scoring system is integrated into the computer player's decision-making process.
* Points are assigned to each potential move based on factors such as column potential and remaining moves (leftpremove).
* The reinforcement method used to assign points enables the computer to prioritize moves with higher strategic value, increasing its chances of winning.

5. Challenging Gameplay:

* With higher **computerIQ** settings, the computer becomes exceptionally challenging to defeat.
* Players face formidable opponents capable of anticipating and countering their moves effectively.
* The combination of advanced algorithms, strategic scoring, and customizable intelligence levels ensures engaging and competitive gameplay for players of all skill levels.

**Conclusion:**

The Connect Four game implementation presented in the Fortran code offers a comprehensive and immersive gaming experience. With customizable settings, advanced computer intelligence, and strategic gameplay mechanics, players can enjoy dynamic and challenging matches. Whether playing against friends or testing their skills against the computer, users can explore diverse strategies and engage in captivating gameplay sessions. The combination of flexibility, intelligence, and strategic depth makes this Connect Four game implementation a compelling choice for players seeking an immersive gaming experience.