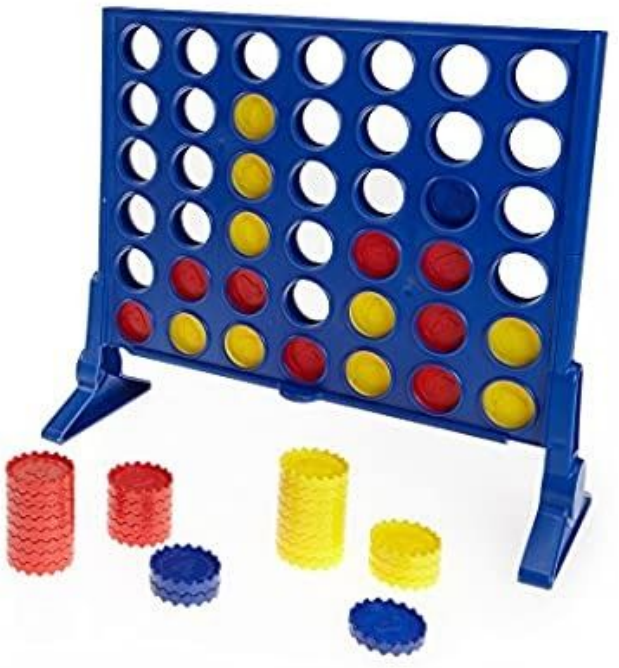
Connect-4 Implementation and Simulation in C

Xinken Zheng & John Paul Tran
CS-49C
San Jose State University



***Abstract*- This paper covers the creation of the Connect-4 game simulation. The simulation is built using C programming language. The paper will cover the libraries that are used for the implementation, the approach followed to complete the program, interesting C-language concepted learned from the project and a snapshot of the simulation.**

I. Libraries used.

- Stdio.h
  - This is a Standard header file for input and output, it is used so that compiler can get input from the user on the keyboard and produce output on the screen. For our project, we need the user to input their move for the game.
- Stdlib.h
  - This is the standard library header file used for the random function to generate a move by the computer. We need to use the random function when there are no more movesets for our AI.
- String.h
  - We use the String header file to manipulate character arrays for our game board.
- Time.h
  - This is the Time library header file. We use this to seed our random function in order to ensure different results each time.
- Board.h
  - This is a header file we created for our game. It contains two methods, printBoard(int gameBoard[7][6]) and showMove(int column, int int player, int gameBoard[7][6]).
- Gamerule.h
  - This is another header file we created for our program.
  - Game rule will contain four functions
    a) `int playerMove(int player, int turnCount, int gameBoard[7][6]);`
    b) `int winningMove(int row, int player, int col, int gameBoard[7][6]);`
    c) `int AImove(int gameBoard[7][6]);`
    d) `int winningComputer(int player, int gameBoard[7][6]);`

II. Approach followed to Complete the program

A. GameBoard

a. The game board for the connect 4 game is a 6 by 7 square, to implement this, we used a 2D array to initialize the gameboard to be 6 rows and 7 columns. `int gameBoard [7][6];`

b. We created two methods for the gameboard, int printBoard(int gameBoard[7][6]) and int showMove(int column, int value). printBoard(int gameBoard[7][6]) will be responsible to display the game board before the game starts, while showMove(int column, int int player, int gameBoard[7][6]) will update the value in the board after every move.

```
// function to print 7 by 6 game board
int printBoard(int gameBoard[7][6])
{
    int checkFull = 1;                      // int for whether or not the board is filled up
    printf(" 1 2 3 4 5 6 7 \n");            // column numbers at top of board
    printf(" -------------\n");
    for(int i = 5; i >= 0; i--)             // for loops to iterate through game board array
    {
        for(int j = 0; j < 7; j++)
        {
            printf(" %d",gameBoard[j][i]);  // print current array index
            if(gameBoard[j][i] == 0)        // if current index is zero
            {
                checkFull = 0;              // board is not full
            }
        }
        printf("\n");                       // print new line for next row
    }
    return checkFull;                       // return whether or not board is full
}
```

```
// function to show the player's move on game board
int showMove(int column, int player, int gameBoard[7][6])
{
    for(int i = 0; i < 6; i++)
    {
        if(gameBoard[column-1][i] == 0)             // if current index is empty
        {
            gameBoard[column-1][i] = player + 1;    // insert disc at current index
            // check if a player has won
            return winningMove(column - 1, player+1, i, gameBoard);
        }
    }
    return 0;
}
```

B. Game Rule

a. int winningMove(int row, int player, int col, int gameBoard[7][6]);

- This function will check whether or not there are four discs in a row on the game board. This would signify a player or computer has won the game. The function would have to iterate through the 2-D array horizontally, vertically, and diagonally to check for a four in a row.

```
// function to check for winning move (4 in a row)
int winningMove(int row, int player, int col, int gameBoard[7][6])
{
    int count = 0;                          // int for counting 4 in a row
    for(int i = 0; i < 6; i++)              // loop to check horizontally
    {
        if(gameBoard[row][i] == player)     // if current index is filled by
        {
            count++;                        // increment count
        }
        else
        {
            count = 0;                      // reset count
        }
        if(count == 4)
        {
            return 1;                       // return 1 for winning move
        }
    }
    count = 0;                              // reset count for next condition

    for(int i = 0; i < 7; i++)              // loop to check vertically
    {
        if(gameBoard[i][col] == player)     // if current index is filled by
        {
            count++;                        // increment count
            if(count == 4)
            {
                return 1;                   // return 1 for winning move
            }
        }
        else
        {
            count = 0;                      // reset count
        }
    }
    count = 0;                              // reset the count for next condi
```

```
for(int i = 0; i < 7 * 2; i++)              // loop to check diagonally
{
    for(int j = 0; j <= i; j++)             // loop to check diagonally up
    {
        int k = i - j;                      // correct diagonal index
        if(k < 7 && j < 6)                  // if they are within the index
        {
            if(gameBoard[k][j] == player)   // if current index is filled by p
            {
                count++;                    // increment count
                if(count == 4)
                {
                    return 1;               // return 1 for winning move
                }
            }
            else
            {
                count = 0;                  // reset count
            }
        }
    }
    count = 0;                              // reset count for next condition

    for(int j = 0; j <= i; j++)             // loop to check diagonally down
    {
        int k = i - j;                      // correct diagonal index
        int x = 6 - k;                      // correct diagonal index
        if(k < 7 && j < 6)                  // if they are within the index
        {
            if(gameBoard[x][j] == player)   // if current index is filled by p
            {
                count++;                    // increment count
                if(count == 4)
                {
                    return 1;               // return 1 for winning move
                }
            }
            else
            {
                count = 0;                  // reset count
            }
        }
    }
    count = 0;                              // reset count for next condition
}
```

b. AI vs Player Mode

- This game mode allows the user to play against the computer in the game. Instead of making the AI make random moves in the game, we implemented a moveset function for the AI called int AImove(int gameBoard[7][6]). This moveset function will tell the AI to always put a move in the middle first if possible, and one to the right if not. It will also check the winning condition after the AI's move. This strategy will make it easier to win for the AI. The AI will always do one of the following.
    - Always go for a winning move if available
    - If not, block an opponent's winning move
    - A moveset move designed by us
    - Random move outside of move set move.

```
// function for AI computer player's next move
int AImove(int gameBoard[7][6])
{
    int move = 9;                                   // initialize mo

    // AI first wants to win
    int checkWin= winningComputer(1, gameBoard);    // check for win
    if(checkWin != 9)                               // if there is a
    {
        return checkWin;                            // return winnin
    }

    // second option is to block opponent from winning
    int checkBlock= winningComputer(0, gameBoard);  // check for mov
    if(checkBlock != 9)                             // if there is a
    {
        return checkBlock;                          // return blocki
    }

    // basic opening movesets we researched and implemented
    if(gameBoard[3][0] == 0)
    {
        return 4;
    }
    else if(gameBoard[3][1] == 0)
    {
        return 4;
    }
    else if(gameBoard[3][2] == 0 && gameBoard[3][1] == 2)
```

```
    else if(gameBoard[4][0] != 0 && gameBoard[5][0] == 0)
    {
        return 6;
    }
    else if(gameBoard[2][0] != 0 && gameBoard[1][0] == 0)
    {
        return 2;
    }
    else if(gameBoard[1][1] != 0 && gameBoard[0][0] == 0)
    {
        return 1;
    }
    else if(gameBoard[5][0] != 0 && gameBoard[6][0] == 0)
    {
        return 7;
    }
    // if all the above moves are not possible
    else
    {
        srand(time(NULL));      // seed for random number
        move = 1 + rand() % 7;  // randomize a column number betweer
        while(gameBoard[move][5] != 0)  // if the column is full
        {
            move = 1 + rand() % 7;      // randomize another column
        }
    }
    return move;                        // return move
```

```
// function for computer to check for possible winning moves
int winningComputer(int player, int gameBoard[7][6])
{
    int tempDisc = 0;               // temporary disc to be inserted into gameboard
    int move = 9;                   // initialize move with nothing value
    for(int i = 1; i < 8; i++)      // loop for each column
    {
        // if current index is not filled
        if(gameBoard[i - 1][5] != 1 && gameBoard[i - 1][5] != 2)
        {
            tempDisc = showMove(i, player, gameBoard);   // insert temporary disc to board
            if(tempDisc == 1)           // if temporary disc is a winning move
            {
                move = i;               // save winning column number
            }
            for(int j = 5; j >= 0; j--) // loop to delete temporary insert
            {
                if(gameBoard[i - 1][j] == player + 1)
                {
                    gameBoard[i - 1][j] = 0;    // reset current index back to 0 (deletes disc)
                    break;                      // break out of loop
                }
            }
        }
```

c. Player vs Player Mode

- Player versus player mode will be self-explanatory. The program will prompt each player one time to enter a move, and player1's move will be represented by 1 while player2''s move will be represented by 2. After every move, we will check the board if there is a winner with the function int playerMove `(int player, int turnCount, int gameBoard[7][6])`. If the game board is full, the game results in a draw.

C. Function Main for Execution

- For the most part, function main will be printing out useful information for the user of the program, and prompting messages for players and the computers' move. All of the functions we implemented will be used in function main. The user can also play another game by simply pressing any key or enter 3 to exit the game.

III. Interesting C-Concept learned from the project

- Overall, this project can be implemented using the knowledge we gain through lectures this semester. However, we did learn how to create and use our own header files to make the code cleaner. The header files we created for this project can be found under "libraries used" in the report.

IV. Final Product

- Here we will showcase our final product for the game. The game will first print the board then ask what mode you would like to play.

```
Welcome to John & Ken's Connect Four Game

 1 2 3 4 5 6 7
 -------------
 0 0 0 0 0 0 0
 0 0 0 0 0 0 0
 0 0 0 0 0 0 0
 0 0 0 0 0 0 0
 0 0 0 0 0 0 0
 0 0 0 0 0 0 0

Choose game mode:
1. Player versus Computer
2. Player 1 versus Player 2
Your Choice: 1
```

- Then once you choose your game mode, you can pick a column to insert your disc into.
- After each turn, the program will print what the board looks like and ask for your column preference.
- Once someone wins, the game is over and the winner is printed.
- You can play again or exit.

```
Turn # 10
Computer's turn
 1 2 3 4 5 6 7
 -------------
 0 0 0 0 0 0 0
 0 0 0 2 0 0 0
 0 0 1 1 0 0 0
 2 1 1 1 0 0 0
 1 2 2 1 2 0 0
 2 2 1 2 2 2 1

Computer chose column 5

Turn # 10
Player 1 's turn
Please choose a number from 1 to 7: 5
 1 2 3 4 5 6 7
 -------------
 0 0 0 0 0 0 0
 0 0 0 2 0 0 0
 0 0 1 1 0 0 0
 2 1 1 1 1 0 0
 1 2 2 1 2 0 0
 2 2 1 2 2 2 1

Player 1 has won!
Enter 3 to exit or any other number to play again: 3
```

```
Turn # 3
Player 1 's turn
Please choose a number from 1 to 7: 7
 1 2 3 4 5 6 7
 -------------
 0 0 0 0 0 0 0
 0 0 0 0 0 0 0
 0 0 0 0 0 0 0
 0 0 0 0 0 0 0
 0 0 0 1 0 0 0
 0 0 1 2 2 2 1

Turn # 4
Computer's turn
 1 2 3 4 5 6 7
 -------------
 0 0 0 0 0 0 0
 0 0 0 0 0 0 0
 0 0 0 0 0 0 0
 0 0 0 0 0 0 0
 0 0 0 1 0 0 0
 0 2 1 2 2 2 1

Computer chose column 2

Turn # 4
Player 1 's turn
Please choose a number from 1 to 7:
```

- Overall, we had a fun time recreating our favorite childhood game and we can now play it whenever we want to.