# CS1201: Computing II Midterm Assignment

Conall Tuohy

Student Number: 18320949

# Connect Four Documentation

## Task:

To make the board game Connect Four in ARM Assembly Language with support for playing against a person and also against a computer.

Overview:
2-Players: Using my GETS function to get input from the players, as each turn occurs the game piece is switched and thus the next player can make their turn. The program checks whether or not the move was a winning move, if so then print out the current game piece and whoever won.

```
while (!win){
	if(gamePiece== X && !playingAgainstRobot)
		gamePiece = Y;
	Else
		gamePiece = X;


	playerMove(input);
}
```

1- Player vs my Robot: First receives a move from the player. If the move was a winning one it ends the game and if not it then moves on to the computer. The computer individually checks both game pieces in each column and checks to see if they can make a winning move for either. If the computer can find a winning move for itself it will take it but if it cannot it will check for possible human player wins and then proceed to block them, failing both it will take piece of highest value and place its game piece in that column. Once the player or the computer has won the winner string is printed.

```
while(!win){
	gamePieceSet(gamePiece); //as shown above
	playerMove(input);
	if(gamePiece== X && playingAgainstRobot)
		gamePiece = R;
	Else
		gamePiece = X;
```

```
        gamePieceSet(gamePiece);
        computerMove(input)
}
```

## INITBOARD:

This subroutine initializes the game board from the array stored in BOARD to 0x40000000. This array is full of the Ascii character capital O as it allowed for easier testing when compared to the number 0. This uses the length of the array as a maxIndex to know how many tiles to place.

```
while(currentIndex < maxIndex){
        GameBoard[currentIndex]=Array[currentIndex];
        currentIndex++;
}
```

## Printboard:

Through the use of the puts subroutine, the Printboard subroutine will put the current gameboard into the console displaying the amount of rows through a constant ROWMUNBER declared at the start of the program. The subroutine will load in each value in the array (which is stored at 0x40000000) and then it uses the put subroutine to put the value to the console, after each value the subroutine will put the Ascii code for a space as having them all printed adjacent to one another makes playing slightly more inconvenient to view and having the space makes it easier to see where you want to place your piece.

```
while(currentIndex < maxIndex){
        System.out.print(GameBoard[currentIndex] + " ");
if(currentIndex % 7 == 0) { System.out.println(); }      //to go to the next line after seven
        currentIndex++;
}
```

## Place:

The Place subroutine takes in a column in the register R0 and whatever piece the game is currently on, the subroutine then begins at the bottom of the inputted column checking to see if the value in that position is equal to the ASCII code for O (which would mean that the space is empty), if the space is empty the game piece is inserted, if not then the subroutine moves the position directly up from the current one and repeats the check. Failing to place anything in the array will result in the program leaving the subroutine and printing that the row was full and asking the user to try again. At the end of this subroutine the system runs another subroutine to check if the player has won with the inputted position or not.

```
while(index > 0){
        if(gameBoard[index] == O)
                gameBoard[index] = gamePiece;
        Else
                index -= NUMBER_OF_ROWS;
}
```

## Robot Column Checker:

The computer subroutine operates by checking if any of the potential moves on the board it can make (in a free space in any row from 1 - 7), if any of them result in the computer winning then it immediately makes that move and ends the game. If that move cannot be made, it will get the move of the highest value, (judged by how close the place is from winning) and return that, but before it places that piece it is meant to check the Players potential pieces, checking for any move that could make the player win the game. If such a move was found it was meant to go there before it placed its move. Unfortunately despite me writing this PLAYERWINCHECKER seemingly correctly, ARM refused to run the program with it in it and because of a small lack of time I had to comment out this part so that the program never checks if it can block the player from winning, making the game extraordinarily easier but the intention to do it and method was there.

```
if(winningMove) {
place(winningMovePosition);
}
//Else if (playersWinningMove)
//{
//place(playersPlace);
//}
Else (PlaceInMostValuablePosition)
```

## checkIfWon Subroutine:

Winning move operates by checking 4 possible winning conditions, Horizontal win (----),Vertical win (|) ,Diagonal Right win (/) and Diagonal Left win (\).

The Horizontal win (-): Goes left and right from the placed piece and checks how many pieces of the same type there are, constantly counting up one if it finds another piece. The left and right checker can only move left and right to a maximum of three tiles and it also takes into account how close they are to the edge of the board (These are inputs needed in the subroutine and are made previously in Place). If the counter hits four, then four pieces in a row must have been played and it branches to the Game Won part of the main to finish the game. If it did not find three other pieces it moves on to the vertical checker.

```
while (win<4){
        if(pieceToTheRight = CurrentPiece)
                counter++;
        if(pieceToTheLeft = CurrentPiece)
                counter++;

}
```

The Vertical Win (|): This checker starts at the address of the last piece placed and just works down. Since no piece can be placed underneath another there is no reason to ever check in the

upwards direction. I had this checker check if the piece below was equal to the CurrentPlayerPiece and if it was to increase the counter but if not, to just stop.

```
while (counter<4){
if(piece at (currentCheckingAddress+amountForNextLine) ==
CurrentPlayerPiece){
        counter ++
} else
 goToDiagnalCounter;
}
```

Diagonal : Starts from the placed piece again and has four different directions to check (Up-Right, Up-Left, Down-Right and Down-Left). . Because the memory works up from 0x40000000 but that is the top of the array I needed to add a row if I wanted to move down and take away a row if I wanted to move up.

The Diagonal Right win (/): This used both the Up-Right checker and Down-Left from the placed pieces address to check whether the combination of these two directions counters could add up to four and be a winning move.

Up-Right: I moved the address up by taking away a row and right by just adding #4 to make it move the next address.
Down-Left: I moved the address down by adding a row and left by just subtracting #4 to make it move the next address.

After both of these if the counter didn't equal four it moved on to the Diagonal Left.

The Diagonal Left win (\): Is a direct mirror of the Diagonal Right win portion. This used both the Up-Left checker and Down-Right from the placed pieces address to check whether the combination of these two directions counters could add up to four and be a winning move.

Up-Left: I moved the address up by taking away a row and right by just subtracting #4 to make it move the next address.
Down-Right: I moved the address down by adding a row and left by just adding #4 to make it move the next address.

After both of these if the counter didn't equal four it finished the subroutine.

CHECKROBOSPOTVALUE Subroutine:
This is a copy of the checkIfWon subroutine that has a small difference at the end that saves the max value of each spot. This subroutine was created to cycle through all available spots on the board to see the best one for the Robot to make.
The program functions as stated before in the CheckIfWon Subroutine.

GETS Subroutine:
We designed this in a previous lab and is just a subroutine that can potentially take a string of inputs rather than just one letter (Though I only use it for one letter at a time). The reason I use

this is because I wrote put into this naturally so instead of having to type get and put separately I can just type GETS and it will work.

isBoardFull subroutine:
Checks whether the very top row of the board is full or not every turn. If it is the game ends in a draw and if it isn't nothing occurs. Only the top row is checked because the pieces all fall to the bottom so as long as the top row is empty there are always places for the pieces to go and for the game to continue.

Subroutine testing:

PRINTBOARD and INITBOARD:
Method:
For these subroutines I entered random ASCII characters into my initializing array in order to ensure that they were displayed correctly, I tested these together as I needed to initialize the board before I could print it and printing the board is a good visual indicator that the initialize subroutine is working and did not interfere with my results allowing for streamlining of my testing process.

Test: When provided the array in image 1 the INITBOARD subroutine initialized the array correctly and Print Board output image 2 to the console.
The L(76) is there to prove it prints what is put in the array

Image 1:

```
189
190
191  BOARD    DCD 79,79,79,79,79,79,76
192           DCD 79,79,79,79,79,79,79
193           DCD 79,79,79,79,79,79,79
194           DCD 79,79,79,79,79,79,79
195           DCD 79,79,79,79,79,79,79
196           DCD 79,79,79,79,79,79,79
197
198
```

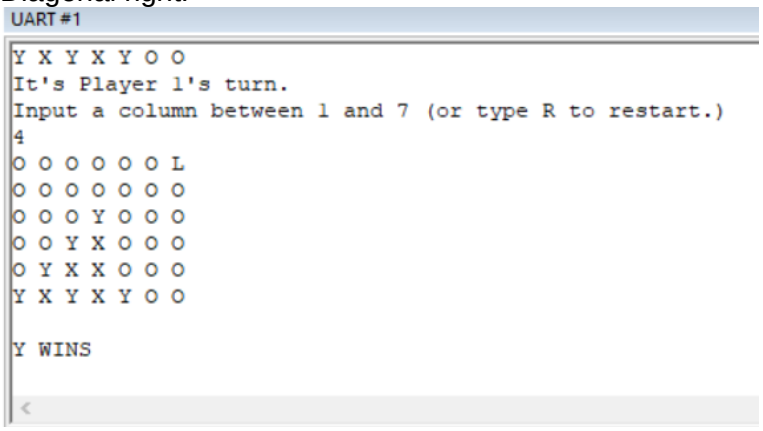
Image 2:

```
UART #1

Would you like to play against a robot (type R) or another player (type P)
PLet's play Connect4!!

O O O O O O L
O O O O O O O
O O O O O O O
O O O O O O O
O O O O O O O
O O O O O O O
It's Player 1's turn.
Input a column between 1 and 7 (or type R to restart.)
```

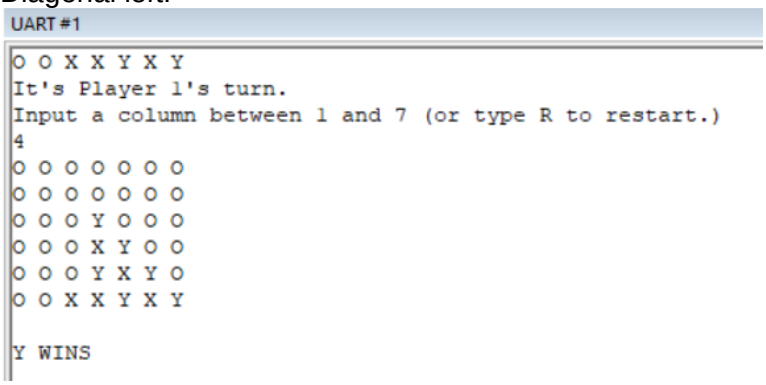
WINNINGMOVE:

Methodology: I used the two player game mode for this subroutine to test every different win condition.

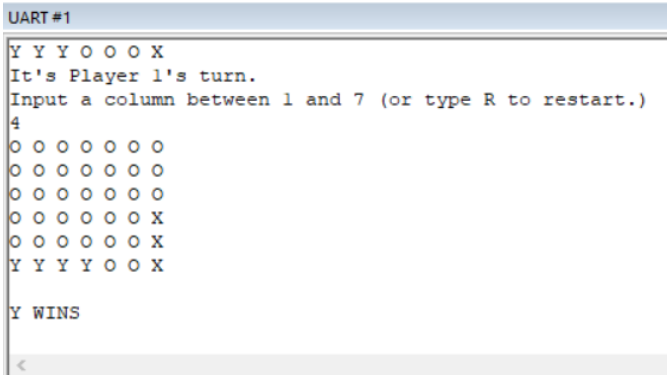
Diagonal right:



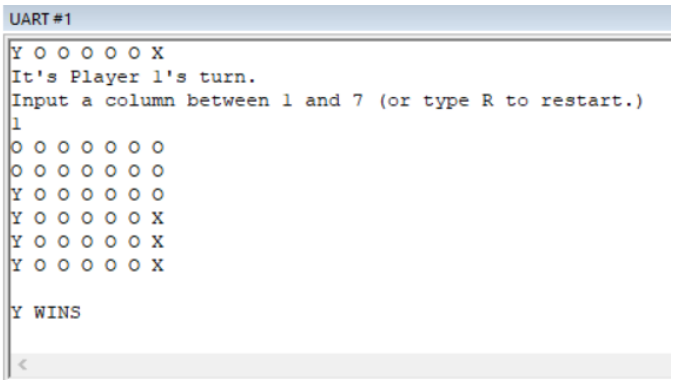

Diagonal left:




Horizontal:



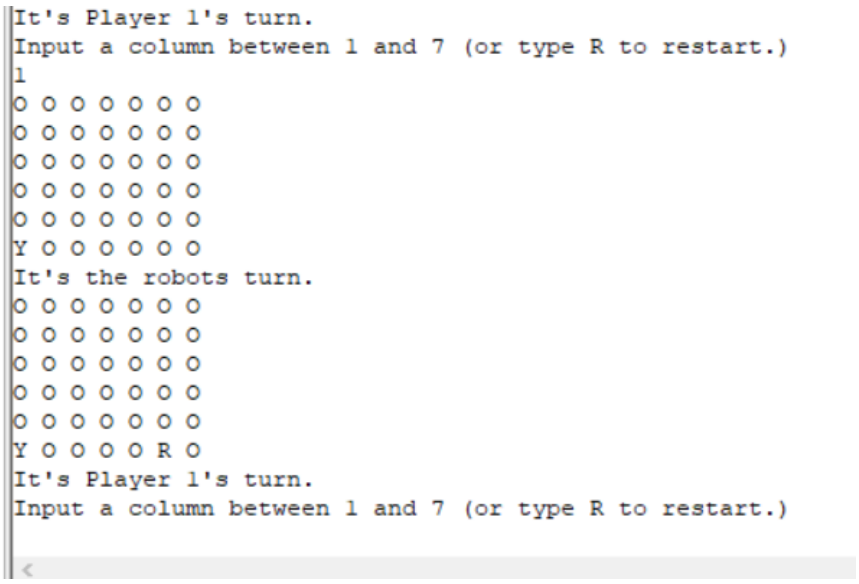

Vertical:




CheckIfWon:
This subroutine can be proven to work by the above pictures as it found all the different win conditions, printed who won and then ended the game.

Computer:
This will show that the computer works as I intended.

So the Computer moves automatically after the players move as seen here:




To Show that the Robot can win I let it in this picture:

```
It's the robots turn.

O O O O O O O
O O O O O O O
O O O O O R O
O O O O O R O
Y Y O O O R O
Y Y O O O R O

R WINS
```

PLACE:
The Place Subroutine has been used in all of the previous pictures in conjunction with the Print command. To show that the place function is printing the right things into memory, here is the memory of the last photo where the Robot won in a straight line up. The memory is formatted in a seven by six grid to make it easier to find where the pieces should be.

For reference the ASCII character 0x59 is Y, the ASCII character 0x4F is O and the ASCII character 0x52 is R.