**Description**

Our project runs the Connect 4 game by first drawing the board which reflects the empty array of spaces on the grid. Our program then receives input from the user that indicates which column the user wants to drop the piece into. The program uses that value to find the corresponding column counter, which is then decremented by 1. The same process is done for the A.I., whose input is random. The game ends when 1 of the 4 win conditions are satisfied by either player, or when the board is completely full. The 4 win cons are:

1. A player has a backwards diagonal (\) strip of 4 pieces
2. A player has a forewards diagonal (/) strip of 4 pieces
3. A player has a horizontal strip of 4 pieces
4. A player has a vertical strip of 4 pieces

If a win condition is satisfied, the game ends. If a win condition is not satisfied, then the loop is reset to drawing the board.

**Challenges**

Some challenges that our team faced was figuring out how to set up the win conditions for the game. Drawing the ASCII board based on the array we set up in MIPS was relatively simple and only took us a day to figure out. Updating the array based on user input and random input was a bit more difficult but didn’t take us too long. When we came around to doing the win conditions, our member Thomas was able to pioneer the strategy for win conditions by coming up with a clever trick to save processing times and use cleaner computation.

YOUR CHALLENGES HERE

**What I’ve Learned**

**YOUR LEARNED HERE**

**Algorithms and Techniques**

**DrawBoard:** The algorithm we used to draw the ASCII board was to have the board start off as a string, and then loop through each char, printing a ‘|’ for each plus another one for whenever a null char marker is reached. Whenever the null marker is reached, a new line is formed and the process is repeated until the board is completely drawn. For subsequent drawings, the values in the array are used to draw the board. A description of the graphical version is at the bottom of this document

**AIChoice:** The algorithm we used for the A.I.’s turn was use Syscall 42, which prints out a random number between 0 and the value given -1, inclusive. The value is then sent to DropPiece.

**PlayerChoice:** The algorithm we used for the Player’s turn was to use Syscall 4, which takes in an int input from the keyboard, saves it to a register, make’s sure the value given is within the bounds, and then send it to DropPiece.

Both PlayerChoice and AIChoice verify that the column is not full before sending it to DropPiece.

**DropPiece:** The algorithm we used to update the values in the board was to first check who’s turn it was, so we know whether to use ‘X’ (red) or ‘O’ (yellow) for the update, and then calculate what position it needs to be added to based on the Row Counters, and then updates the respective Row Counter accordingly.

**WinCheck:** The algorithm we used to check for end game conditions could find where to start checks for wins, what to do to get to the next position to check, and a counter to check if there is four in a row. This is done by a counter starting at 1 of the only starting positions a certain strip could have (12 for forward diagonal, 12 for backward diagonal, 24 for horizontal, 24 for vertical), and adding a number (9 for forward diagonal, 7 for backward diagonal, 1 for horizontal, 8 for vertical) to the counters position, increasing the counter if the next piece in the jump is the same as the previous

**Row Counters:** The technique was used to easily update positions in the game array were row counters, where each column was assigned a counter that represented how many empty spaces were in the column. If all the counters were 0, then that means the game board is full and the game ends.

**Contributions**

YOUR EVAL HERE

Thomas – Helped tackle the win condition challenge and came up with a clever strategy to implement it

Connor – Helped optimize the code and worked on the extra credit portion of the project

Syed – Wrote the bulk of the code including DrawBoard, DropPiece and the AI/Player Choices

Wilfred – Wrote the Project Report, User Manual, and made the Video Clip demonstrating the program

**Extra Credit Graphics**

|  |  |  |  |
| --- | --- | --- | --- |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

To create graphics for connect 4, we had to find a way to draw objects in the MARS bitmap display. To do this, we used an X/Y grid system, where x is horizontal and y is vertical. Two vertically aligned pixels (at the resolution we chose), are 128 pixels offset from each other, so we sll’d the Y value by 7, before adding the x value to get the pixel number that needs to be written to. With this it was possible to draw any pixel we wanted, but we still needed to figure out how to draw circles.

We realized that a pixelated circle is actually a series of horizontal lines of pixels, and that we could draw a circle by drawing the length of those lines, from an offset position to align the centers, so for instance 2 pixel 1 pixel offset, 4 pixels 0 offset, 4 pixels 0 offset, and 2 pixel 1 offset, creates a really low resolution pixel. By using this method, we were able to hard code the values of a circle into an array of bytes in the data segment, and create a subroutine that takes the values of the left and top of a circle to draw a circle. Next we needed a way to change the color, so we added the color as another argument to our subroutine.

After that we needed a way to take the row and column values of our data to choose where the circles got drawn, so we did some math to figure out a function to take those values and convert them into X/Y coordinates for the top left, and created a function to take a row/column pair and draw a circle at the correct position. The next thing to do was to draw the board, which was to just to color every pixel from a few rows down till the last visible pixel blue, then to loop through all the locations a circle could need to be drawn to and draw a white circle there. With that the basics of the graphics were done.

After that we improved the graphics by adding smaller, slightly darker, circles to the centers of the tokens, but that was just a repeat of the hardcoded circles, just with fewer and smaller values, and some slight offsetting from the larger circles X/Y to get them in the right place. And finally we added a chevron at the top of the board to indicate the last piece played, by drawing a 4-pixel horizontal lines at +1 X values -1 Y values some times, and +1 X +1 Y values the same amount to create a v over the column.