

Table of contents:

Report for Group 17..... 2

 Introduction: 2

 Approach..... 2

 Algorithm 3

 Result and output 4

 Analysis 13

 Summary and conclusion..... 13

 How to run and compile 13

 References 14

Report for Group 17.

Introduction:

Group 17 has made a Tic Tac Toe two player game. This game is usually a very simple game displayed on paper with a pen, in a black and white version. This version has been given improvements in the graphics department. It is played in 8-bit terminal colors which give a nicer feel when playing compared to the classic black and white version. Some terminals may have issues displaying colors, therefore a black and white version is also provided with a color check at startup to verify color support. Feel free to test out both modes. PS: Make sure the terminal window is big enough to show all the art.

The report is a little long because with 8 pages of output screenshots, and APA formatting (Only 4 written pages).

Approach

Three bigger issues were an obstacle to overcome during the creation of this game. Cross-Platform support, Terminal “rendering” and memory management for colors.

- Since the program has been created to support both Unix and Windows systems there is a lack of cross-platform tools available as “windows.h” header and bash calls for a unix environment, were not a possibility. This was solved using newlines to “clear” graphics before printing new ones instead of overwriting existing graphics. This global function is inlined in the “global.h” header file. It is a small function which is called often – therefore good function to inline to reduce the execution time of the program.
- The terminal output is limited to writing one line at a time, which means that with tic tac toe symbols X and O, of 9 times 7 character has to be printed one line at a time to be displayed correctly. This gets more complicated when you need 3 symbols in a row with a divider between each to display the board. In the Board::printGraphics() function this is solved iterating through each x row of the board, saving those values to a vector. Then proceeding to format the symbols in the correct manner:
prefix(playerArt), divider, symbol 1, divider, symbol 2, divider, symbol 3, suffix(playerArt).
After this a horizontal divider is printed with prefix and suffixes for each line. Then we

proceed to print the next 3 symbols and repeat until the terminal is displaying the 3x3 board.

- Colors are used widely throughout every part of the program. Therefore, a global structure AllColors storing all the values are present in the “global.h” headerfile. These colors are ANSI escape codes (Wikipedia contributors, 2019) are passed by references and pointers throughout the program to prevent value copying and increasing the memory usage. This file is initialized upon startup and right after a color check is displayed where the user has the choice to disable all colors, which will change all the values of the struct to an empty string. A GlobalBackgroundColors struct is also defined with pointers to the chosen background, border, divider, and visible text on the background values are stored in the AllColors struct. This makes it very easy to change the entire board layout upon initialization and could be a feature implemented later.

Instead of a classic select a number to place a symbol approach there has been implemented a struct Position selectionPosition in the Board class. This keeps track of the last coordinates on the board the marker was positioned at. This marker is moved using w, a, s, d, and x to place a symbol. Each key followed by enter.

The game is developed with header and source files to make it easier to get an overview of what each class' and function files are doing.

Algorithm

The Board::checkWonOrDraw() function is the only algorithm in game. It is following a very effective approach. Since the Board class keeps track of the last coordinate through the Position selectionPositon struct it can get the column and row of the last placed symbol. The x row, and y column is then iterated and checked if they match the type (X or O) of the last inputted symbol. If there is not a 3 in a row in the horizontal or vertical directions it proceeds to check if the last placed symbol was top right, bot left or middle. If it was it will check the diagonal from bottom left to top right and compare the types. At last if the last placed symbol is top left, bottom right, or middle it will check the anti-diagonal from top left to bottom right. All of this is checked with if else statements, and if 3 in a row are found the 3 values are getting stored in Position

wonMarkers[3] struct which will highlight the 3 winning symbols in the next Board::printGraphics().

Result and output

Here is a text version of an example run of the program.

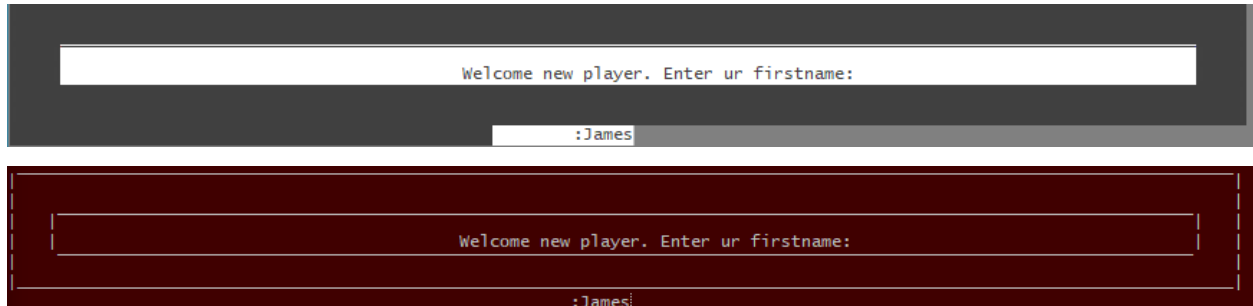
1. (Req. User input) First you are asked if you can see the colors on the screen: y/n.
2. 2-5: Repeats until 2 players are created:
3. (Req User input) Enter your first name: myName.
4. (Req User input) (If yes proceed, else skip to point 3) Select a color the play with: 1-5.
5. (Req User input) Select an avatar to displayed as prefix/suffix by the board: 1-4.
6. (When both players is ready) Display the board and the user to start is highlighted.
7. Play the game. Best of 2 format. The player to start swaps each round.
8. (Req User input) Prompted if you want to play another game.

Visual version of the game with comments above. Each “frame” is displayed after a clearScreen();

1. (Req. User input) First you are asked if you can see the colors on the screen: y/n.

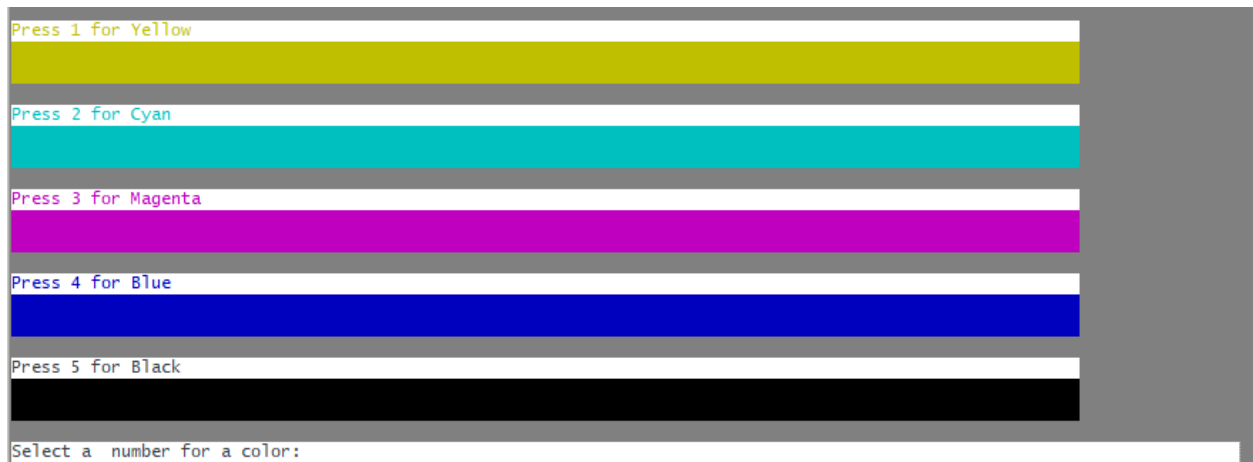


2. 3-5: Repeats until 2 players are created:
3. (Req User input) Enter your first name: myName.



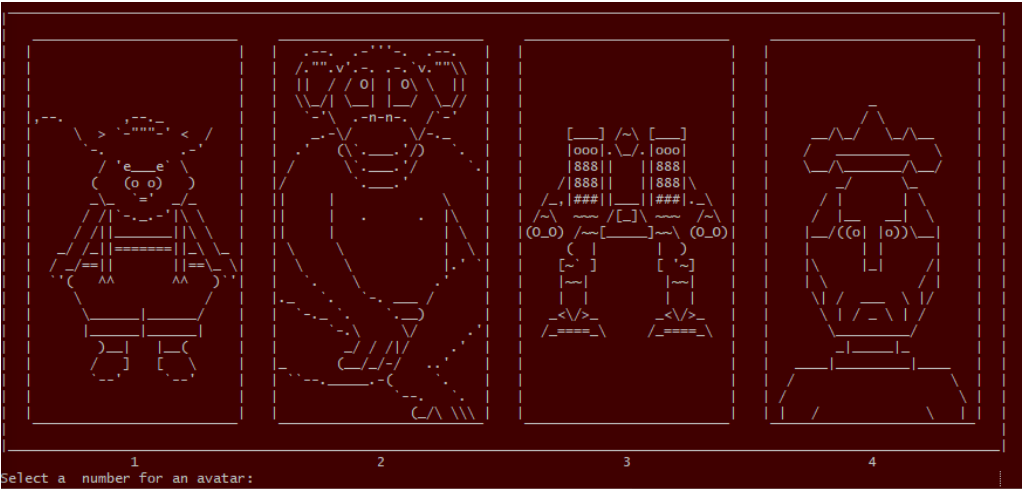
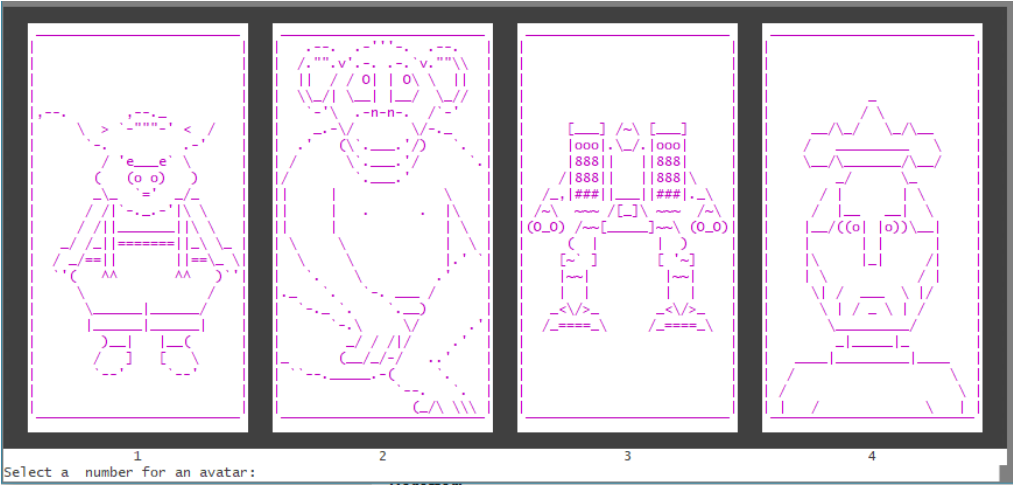
The image shows two screenshots of a terminal window. The top screenshot has a dark gray background and shows the text "Welcome new player. Enter ur firstname:" in a white input field. Below it, the text ":James" is entered in a smaller white input field. The bottom screenshot has a dark red background and shows the same text "Welcome new player. Enter ur firstname:" in a white input field. Below it, the text ":James" is entered in a smaller white input field.

4. (Req User input) (If yes proceed, else skip to next step) Select a color the play with: 1-5.

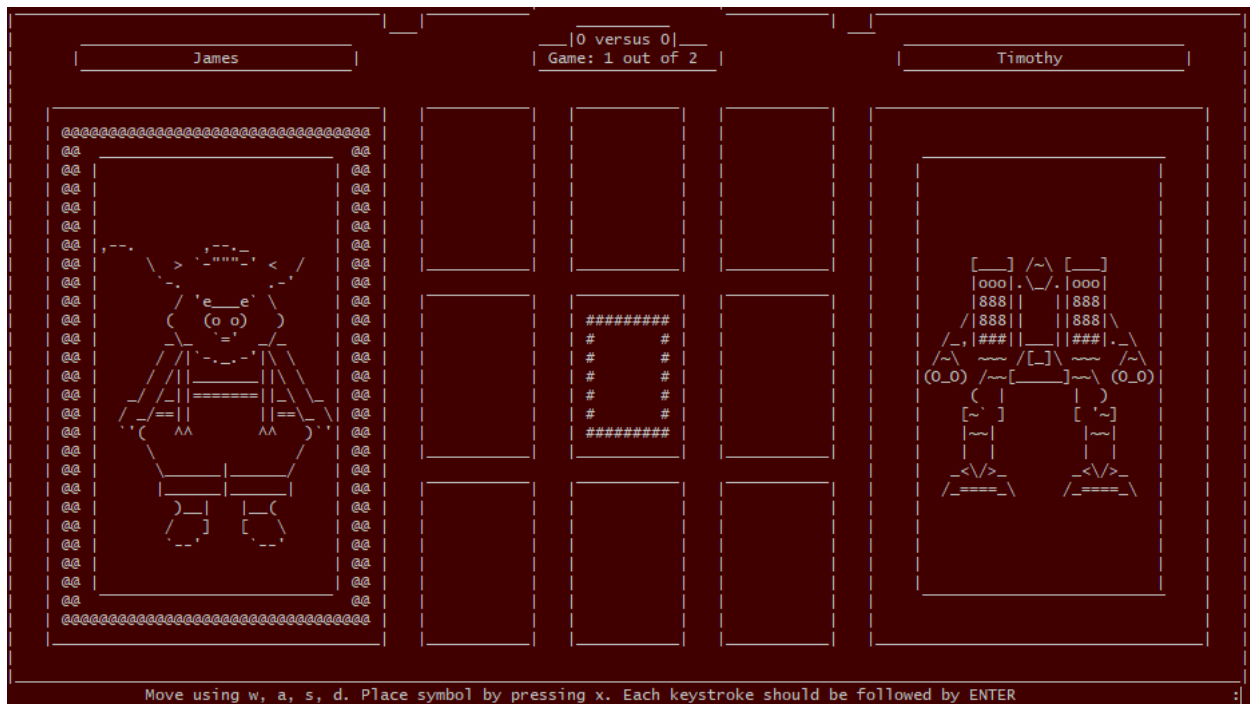
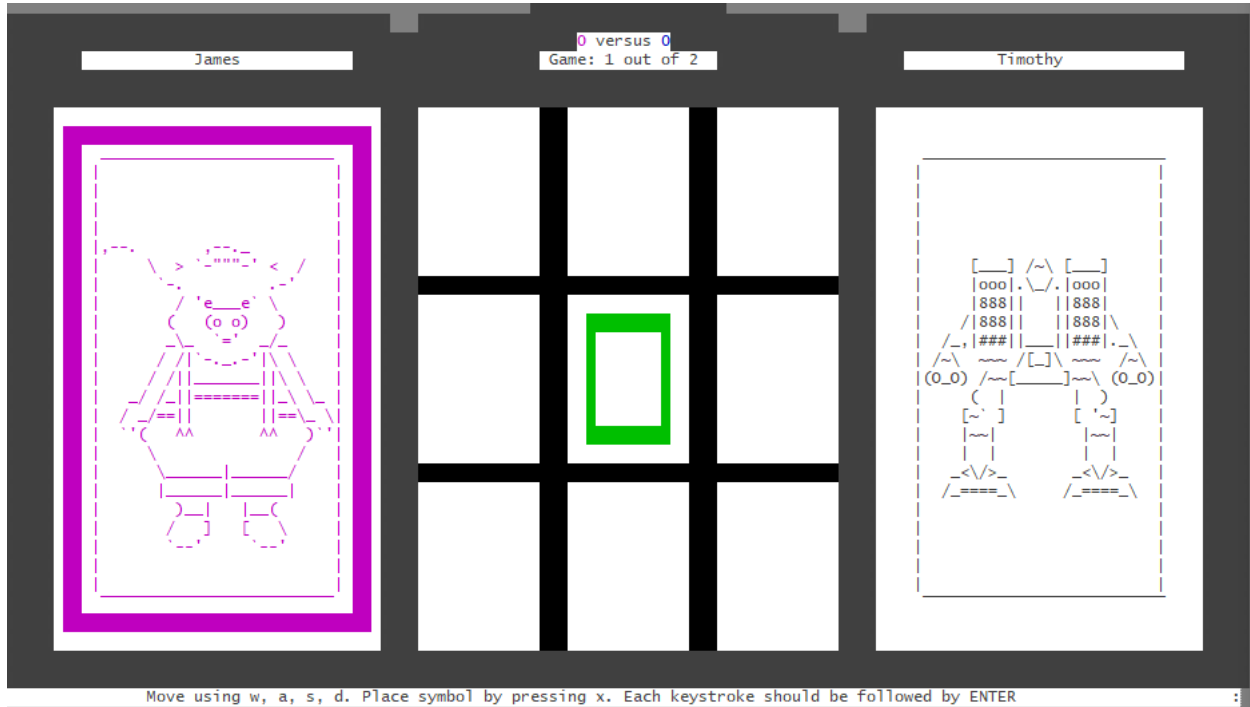


The image shows a terminal window with a gray background. It displays a color selection menu with five options, each preceded by a prompt: "Press 1 for Yellow", "Press 2 for Cyan", "Press 3 for Magenta", "Press 4 for Blue", and "Press 5 for Black". Each prompt is followed by a horizontal bar of the corresponding color. At the bottom, there is a white input field with the text "Select a number for a color:" and a cursor.

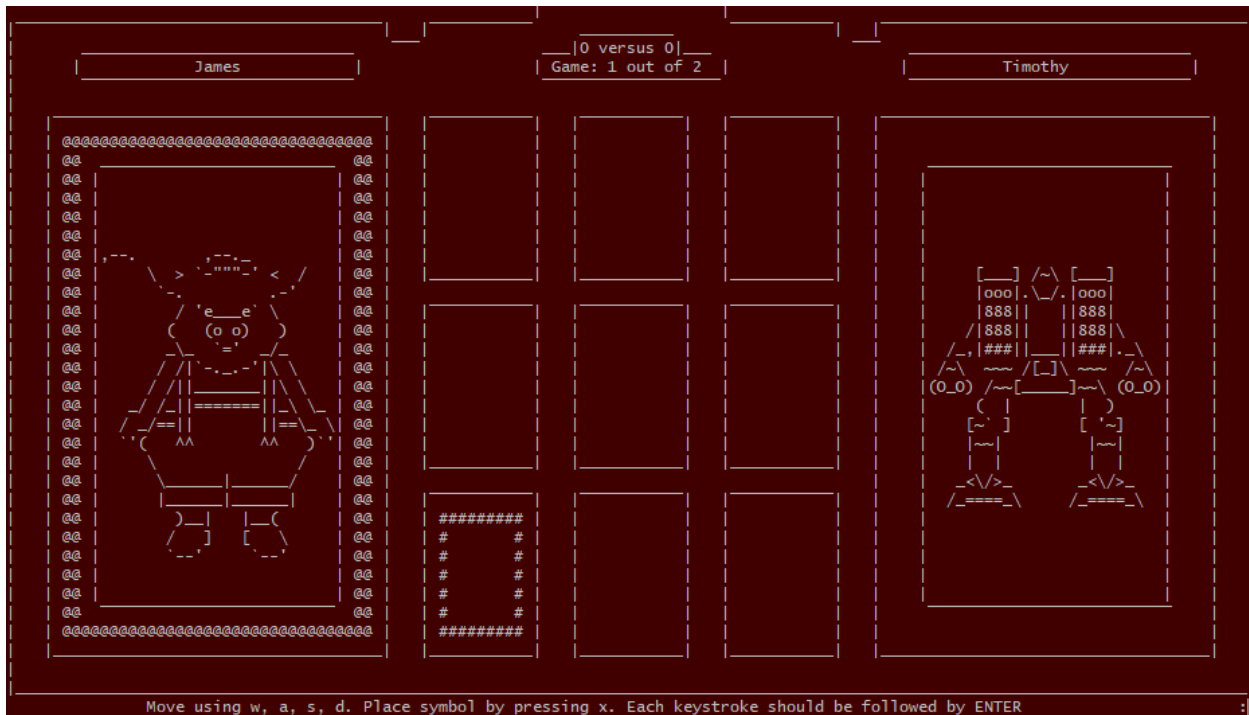
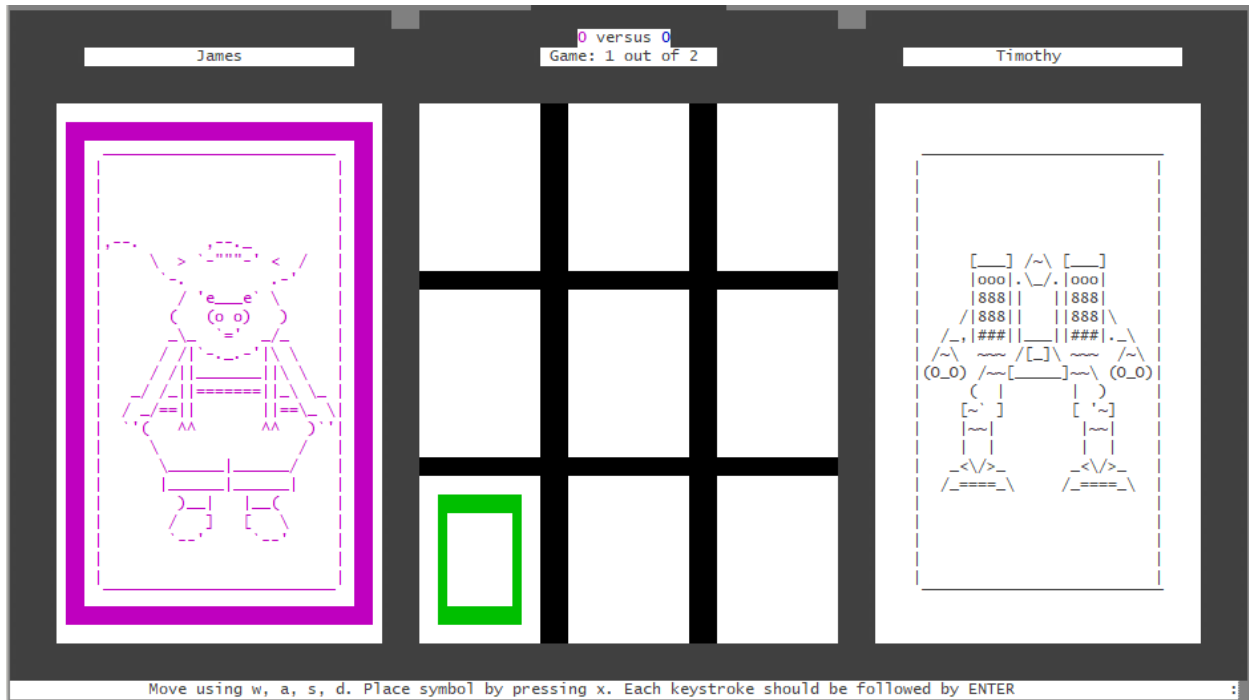
5. (Req User input) Select an avatar to displayed as prefix/suffix: 1-4.



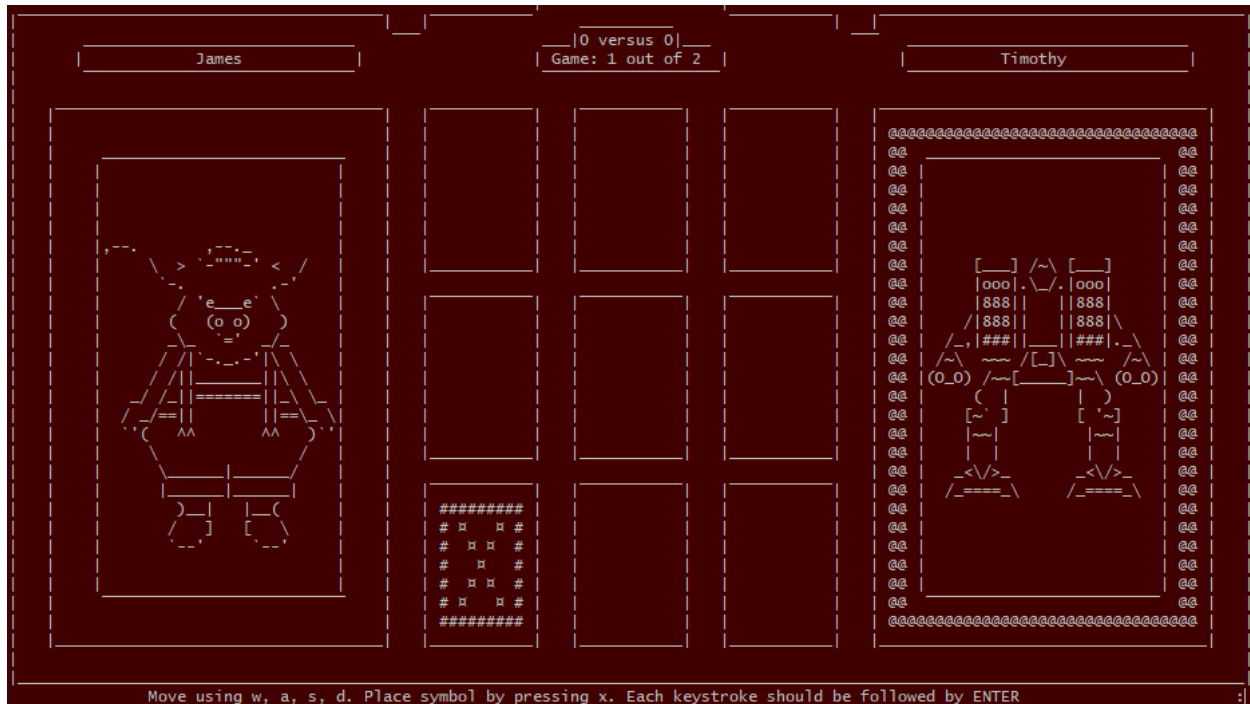
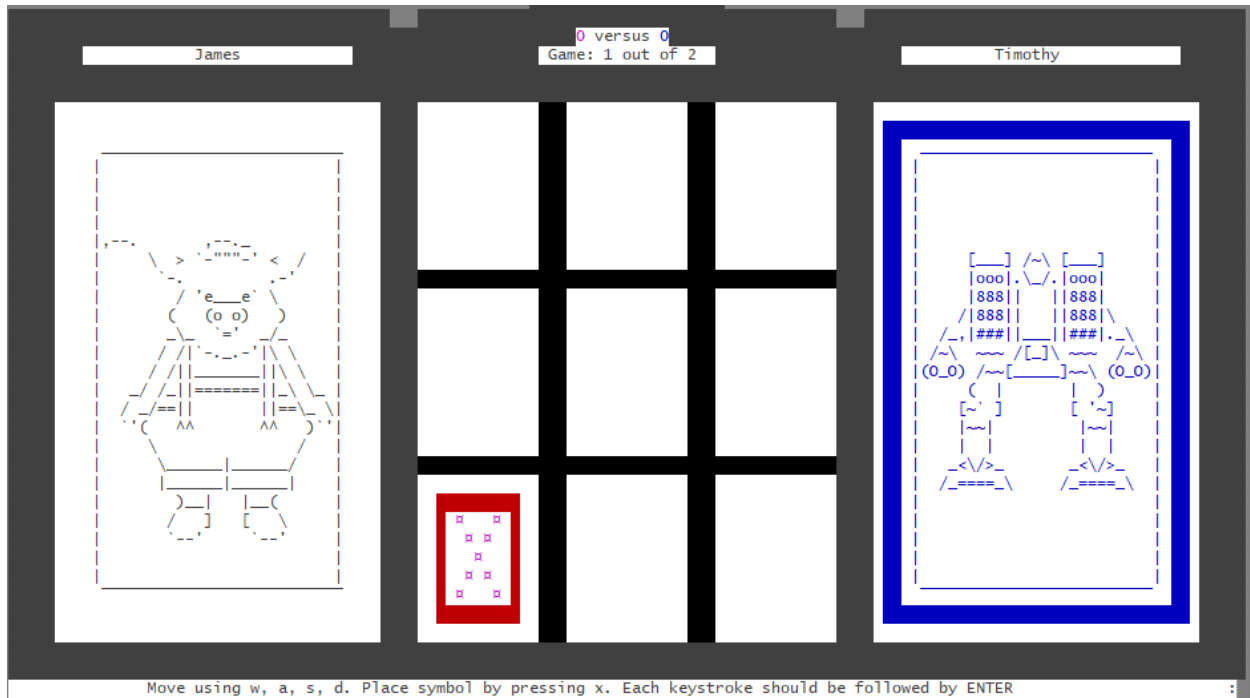
6. (Req User input) (When both players is ready) Round 1: Display the board and the user to start is highlighted. Use w, a, s, d followed by ENTER to move the selection square (in the middle of the board):



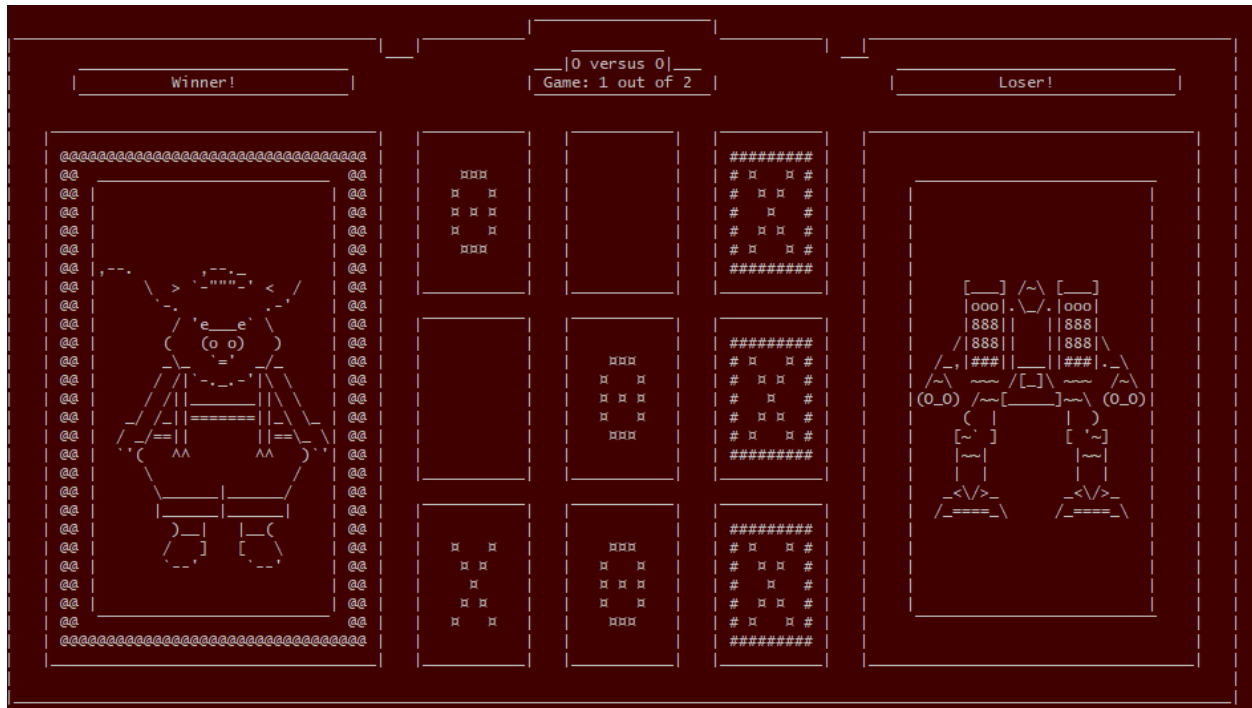
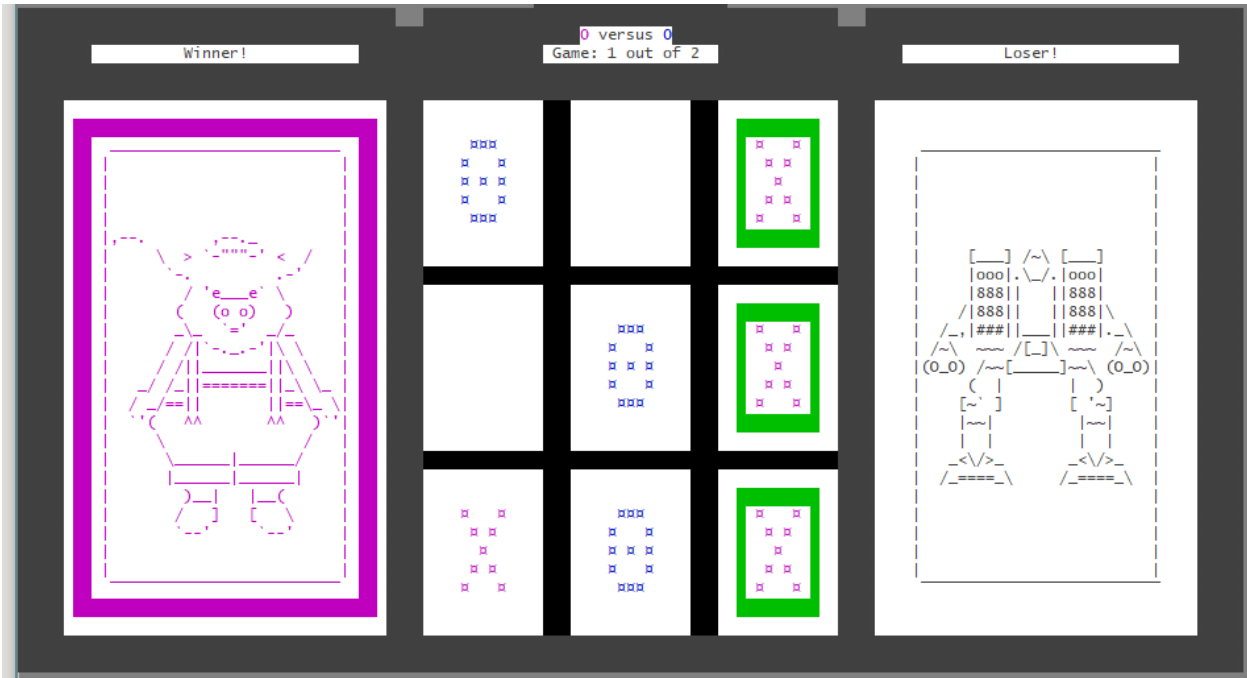
7. (Req User input) Move to desired square:



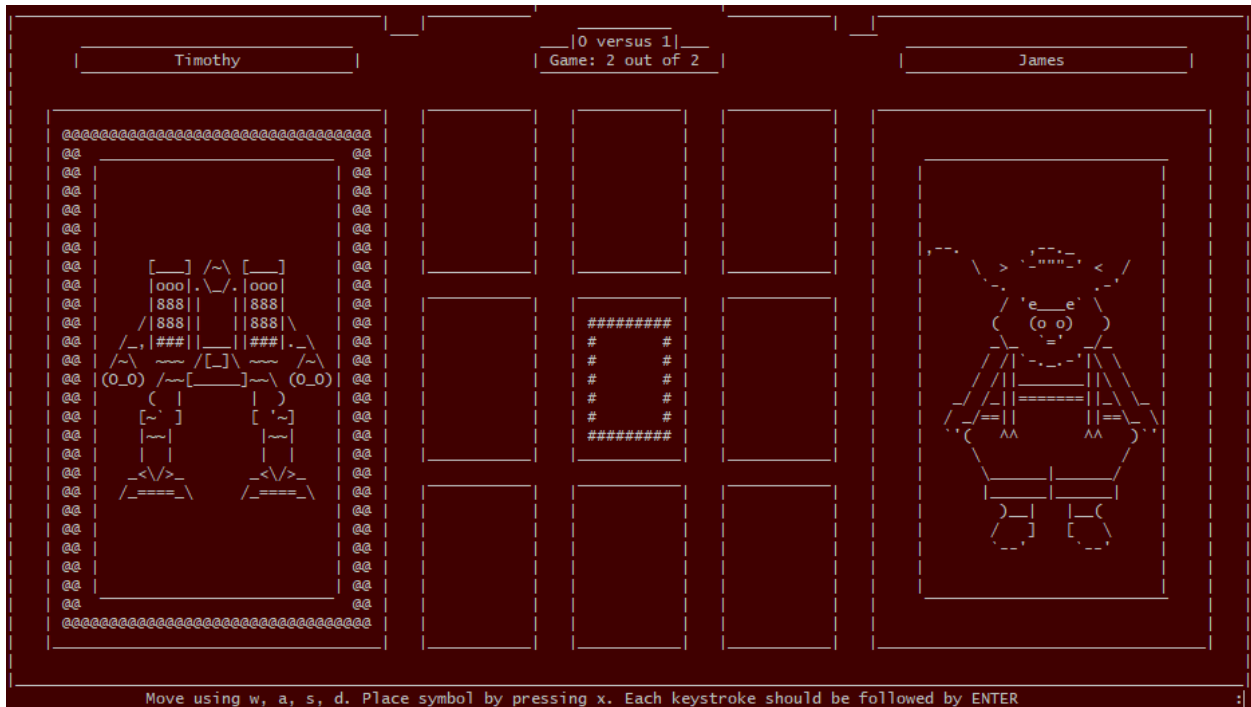
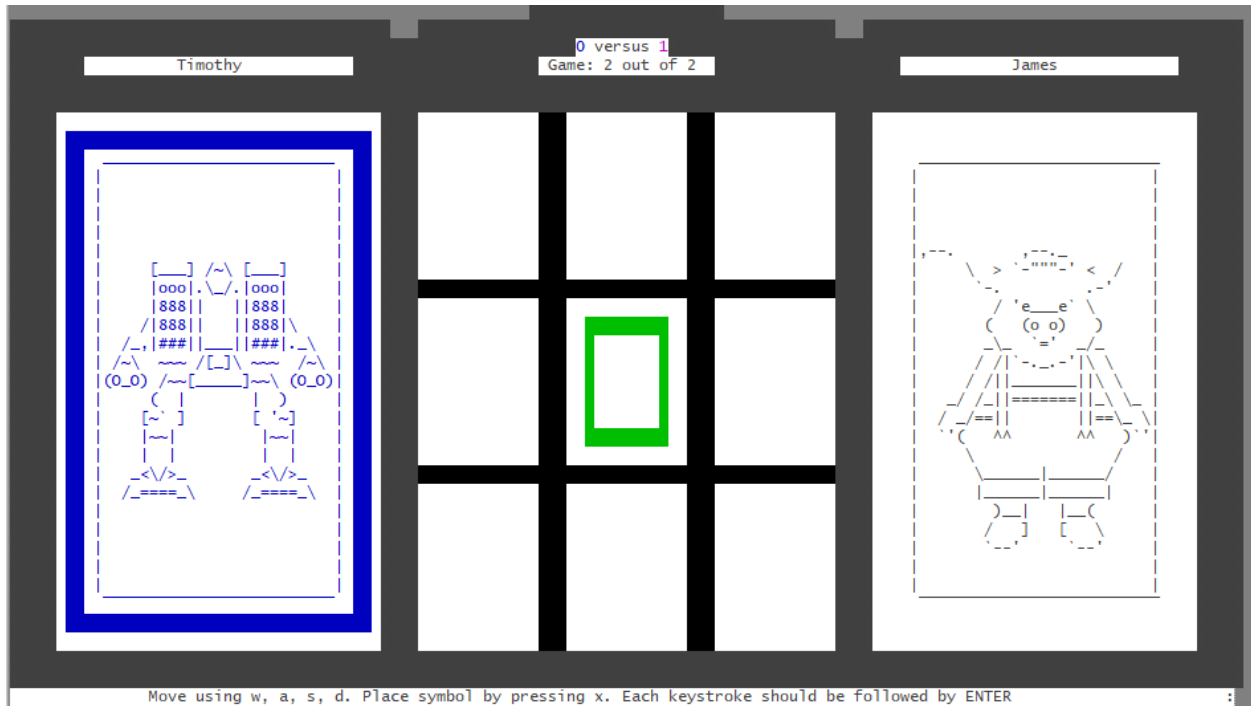
8. (Req User input) Press x to enter symbol (followed by ENTER):



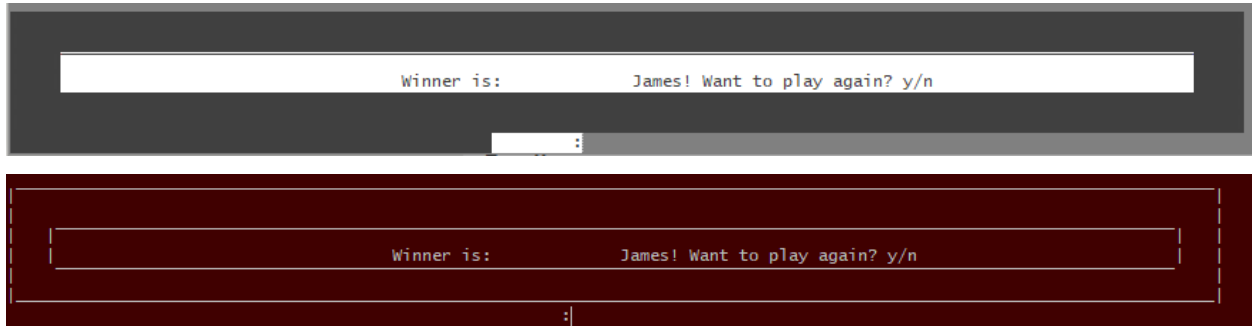
9. (Req User input) Play until a winner or draw, Winner /Loser is displayed for 2.5 seconds before moving on:



10. Round 2: The starter is swapped. Game x out of y now updated. and points updated.



11. (Req User input) After another round the winner or a draw is announced and prompted to play again. y/n



12. If 'y' jump to step 6. Else finished.

Analysis

There is a lack of smooth movement controls since each input has to be followed by an ENTER throughout the program. It would be a great addition to later add a feature which accepts the input keys and then just proceeds (And to loop if no viable character was inputted). There was not enough time to find and implement a cross platform solution to this.

The game has an unusual approach including selection of a color and an avatar. The selector square is also new and would have probably been more intuitive as a click on square to select. These features are not necessary to make a good Tic Tac Toe game. However, they give something extra to the game which gives it a little more depth.

Summary and conclusion

In hindsight a game engine would be a great addition to make this program better, and even more intuitive. Place a symbol on a click or user input without verification would make the program a little easier and make even more sense as a computer game instead of a paper game. It would also make it easier to display all the needed graphics for the visualization of this program. The game engine would have been a very time-consuming task to learn and implement while focusing on the efficiency of the program in the given timeframe. And as the first edition of this program it is safe to say it fulfills its purpose.

How to run and compile

This game was developed in Debian 9 “Jessie”. And is compiled using the GNU C++ compiler: g++. It is tested and working in both Windows and Linux environments. It has not been tested on an OS X system but since it is based on UNIX it will most likely run there as well.

For Linux any terminal should do the job.

For Windows I have tested and successfully executed the program using CYGWIN(Cygwin, 2019) and is my recommended (and proved to work, no other Windows terminals are tested) terminal.

A compiled a.exe file is included. However, if a recompile is needed this command will do the trick:

(Make sure no other source/header files are in the folder).

```
g++ *.h *.cpp
```

References

Cygwin (n.d) *This is the home of Cygwin project*. Retrieved April 8, 2019 from
<https://www.cygwin.com/>

Lafore, R. (2002) *Object Oriented Programming in C++* (4th edition) Indianapolis, Indiana,
USA: Sams publishing

Wikipedia contributors. (2019, April 3). *ANSI escape code*. In *Wikipedia, The Free Encyclopedia*. Retrieved 10:12, April 8, 2019, from https://en.wikipedia.org/w/index.php?title=ANSI_escape_code&oldid=890844283