

|  |
| --- |
| Tic-Tac-Toe Report |
|  |
| August 5th  Samuel Pitchforth |

# Contents

|  |
| --- |
| [Contents 2](#_Toc110542295)  [Defining and understanding the problem 3](#_Toc110542296)  [Software description 3](#_Toc110542297)  [Functional Requirements 3](#_Toc110542298)  [Gantt chart 6](#_Toc110542299)  [Time Management Plan 7](#_Toc110542300)  [Context Diagram 8](#_Toc110542301)  [Planning and designing software solutions 9](#_Toc110542302)  [Dataflow Diagram 9](#_Toc110542303)  [Structure Chart 10](#_Toc110542304)  [IPO Chart 10](#_Toc110542305)  [Screen Designs 11](#_Toc110542306)  [Storyboard 12](#_Toc110542307)  [Pseudocode 13](#_Toc110542308)  [Flow Chart 14](#_Toc110542309)  [Implementing software solutions 14](#_Toc110542310)  [Data dictionary 14](#_Toc110542311)  [User documentation 17](#_Toc110542312) |

# Defining and understanding the problem

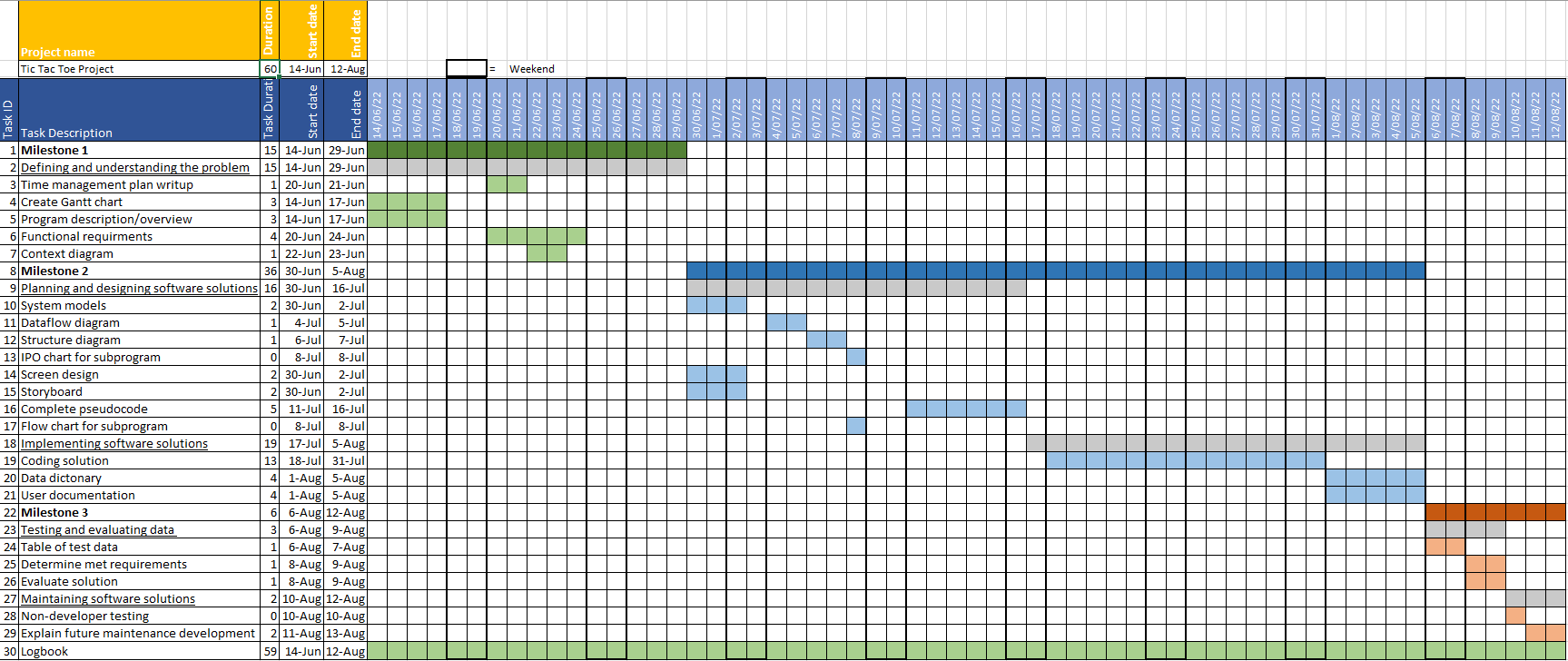
## Software description

The software required is a text-based game of Tic-Tac-Toe against a computer-controlled opponent. The game takes place on a 4\*4 size board and the players take turns placing their counters (“X” or “O”) on an empty square. The user always starts first and the first player to create an unbroken line 4 counter long wins. At this point the game ends and the winner is conveyed. Alternatively, if there is no space left on the board and neither player has won, it is a draw. This game must be password protected and include a main menu with options to play the game, read instructions, and exit the program. The only way to exit is from the main menu, every other option returns to the main menu when completed.

## Functional Requirements

* Start with a security screen
* Allow up to 4 password attempts
* If the correct password (1234) is entered open the main menu
* If the correct password is not entered, communicate what happened to the user, then exit the program
* The main menu has a welcome message/title
* The main menu has 3 choices: Player Vs Computer, Instructions, Exit
* By entering 1, 2 or 3, the user can choose the corresponding option
* If any other input is given, an error message should be printed and the entire main menu should be reprinted including the input prompt
* After the Player Vs Computer or the Instructions mode have run and then ended, the main menu should be reprinted
* Selecting “Player Vs Computer” begins the game between the user and a computer-controlled opponent
* The player always goes first and is assigned the “X” counter
* As the game begins, the user will be prompted to enter their name which is stored for later use
* Whenever the turn is passed to the user, they will be prompted to make a move with their name
* The game starts with an empty board of the dimensions 4\*4
* The board state is stored as a 1D array
* Whenever it is the user’s turn, they are prompted to enter a move. A valid input consists of a number in the range 1-16 inclusive
* If an invalid input is entered, an appropriate error message is displayed and the user is prompted for a valid input
* Each input corresponds to a square on the board
* If the inputted square is already occupied by another counter that value is treated as an invalid input and a relevant reinput prompt is printed
* After the player makes their move the board is reprinted with the move.
* The computer is then passed the turn
* The computer generates moves randomly, and in the case of an invalid generation a new move is generated. An invalid generation need not be shared with the user
* After the computer makes a valid move, the board is reprinted with the new move
* At any time, a winning state (4 of the same counter in an unbroken line) should end the game and communicate to the user who won and then return them to the main menu
* If there are no squares left to place a counter the game should end, the draw communicated to the user and then return to the main menu
* Selecting “Instructions” should print the instructions, accessed from a separate text file named “intructions.txt”
* The file should describe the programs functionality clearly to the user
* The instructions text file must be read line by line
* Selecting “Exit” should exit the program
* The “Exit” option must not rely on subroutines or functions

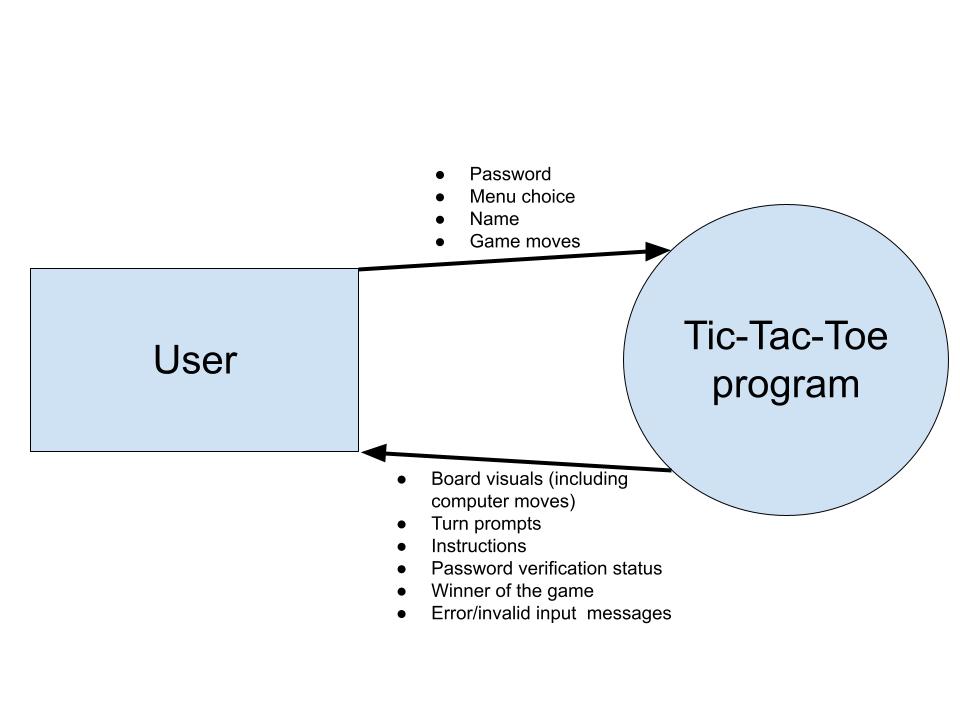
## Gantt chart



## Time Management Plan

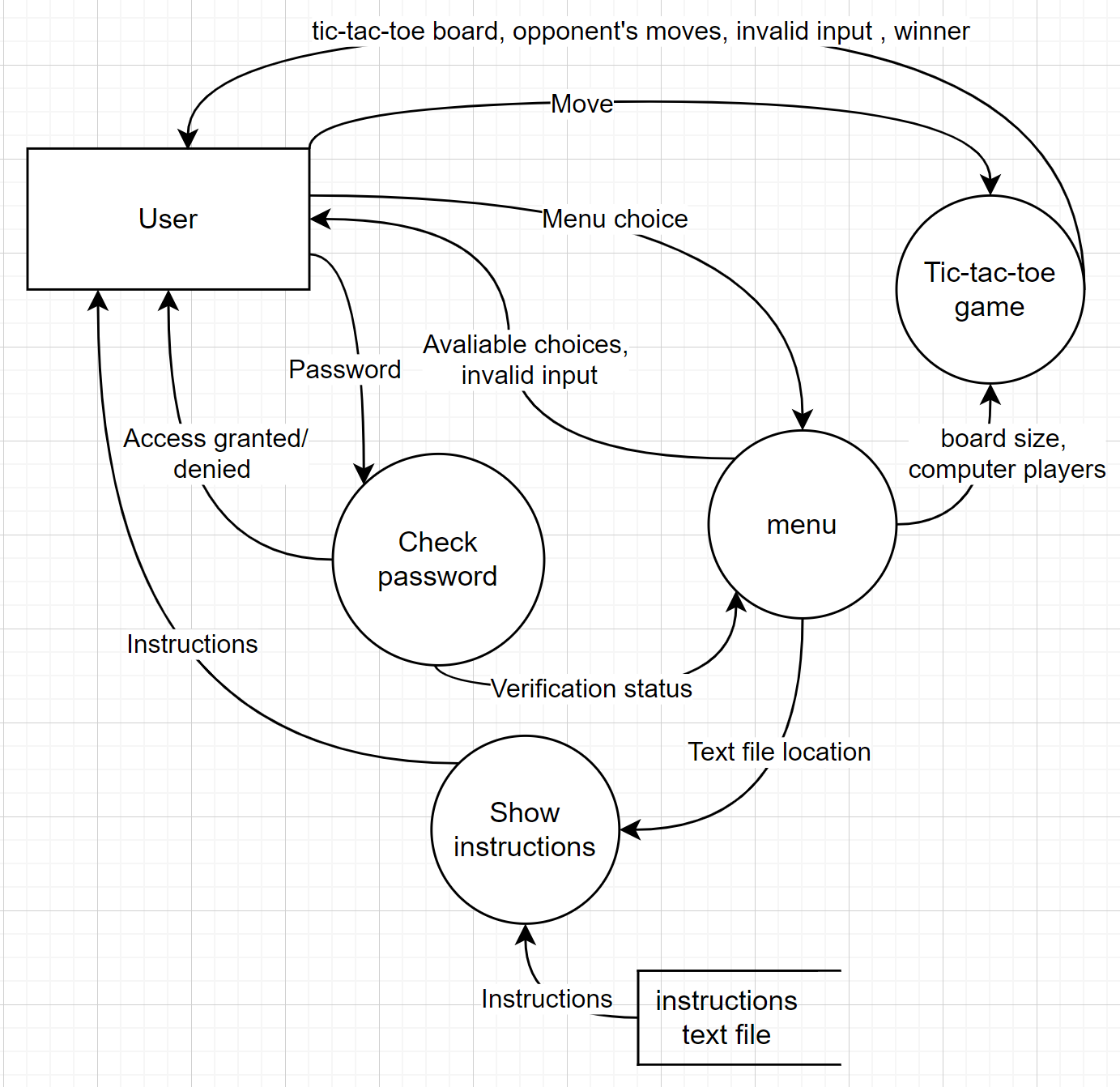
One of the most difficult factors of multistep projects such as this one is efficient management and use of time. In order to successfully create a software solution that fulfills the requirements, the Software Development Cycle will be used. This will be spread across the nearly 3-month project timespan. The Gantt chart above is essential in planning out dependencies, order and the time needed for each individual component. A logbook will be kept to allow clarity on how the project is progressing and keeping to the routine set forth by the initial time management strategy. This will also allow for reflection on difficulties encountered and what still needs to be completed to focus later work sessions. This breaking down of urgent tasks and focused work will maximize the time efficiency of the project. A version control will be kept in cloud storage. In the case of errors, it will be far easier to revert to a pre-error version, and there will backups of project relevant files in the case of file loss. The version control can save huge amounts of time in these scenarios. It also acts as a verification process for the logbook, that is every logbook entry can be checked against the relevant version control files to detect errors and inconsistencies. This increases the accuracy of the logbook, and thus its efficacy. Each of these components helps manage the development cycle of this project, and complete work efficiently in a useful manner.

## Context Diagram

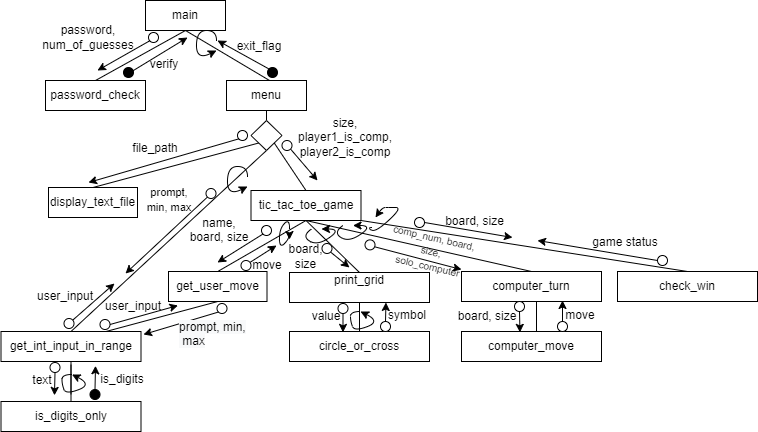


# Planning and designing software solutions

## Dataflow Diagram

The dataflow diagram (DFD) above shows the flow of data through the program. DFD’s like this one are useful for mapping out the solution, understanding how data is processed into information, and how different processes interact with each other and external entities. In this solution, the menu is the main user interface where the user selects their preferred options. These details are given to the relevant submodules which then perform their task, often interacting with the user directly.

## Structure Chart

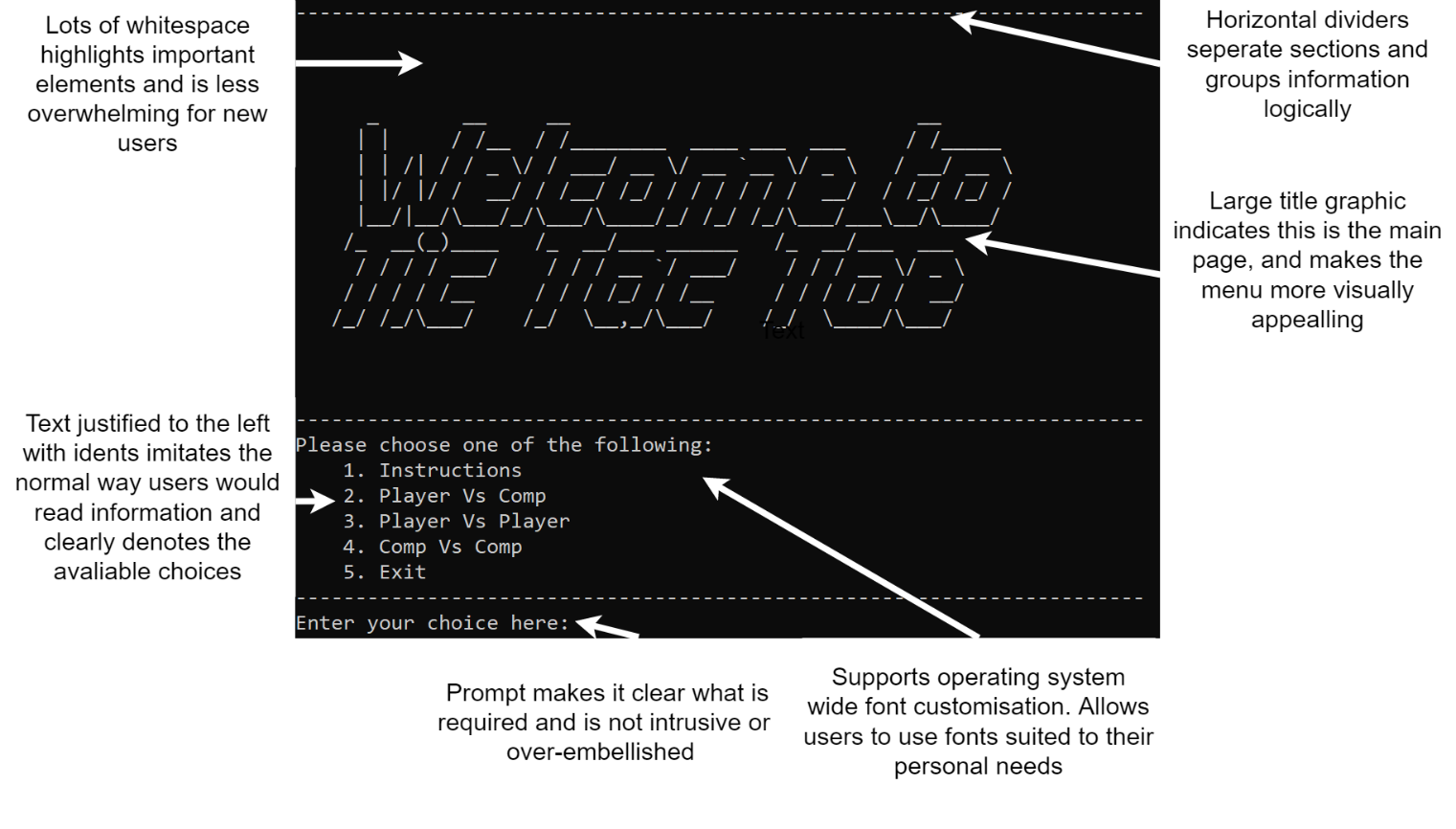


This structure chart shows the hierarchy of subroutines within the solution and the parameters and return values they communicate with.

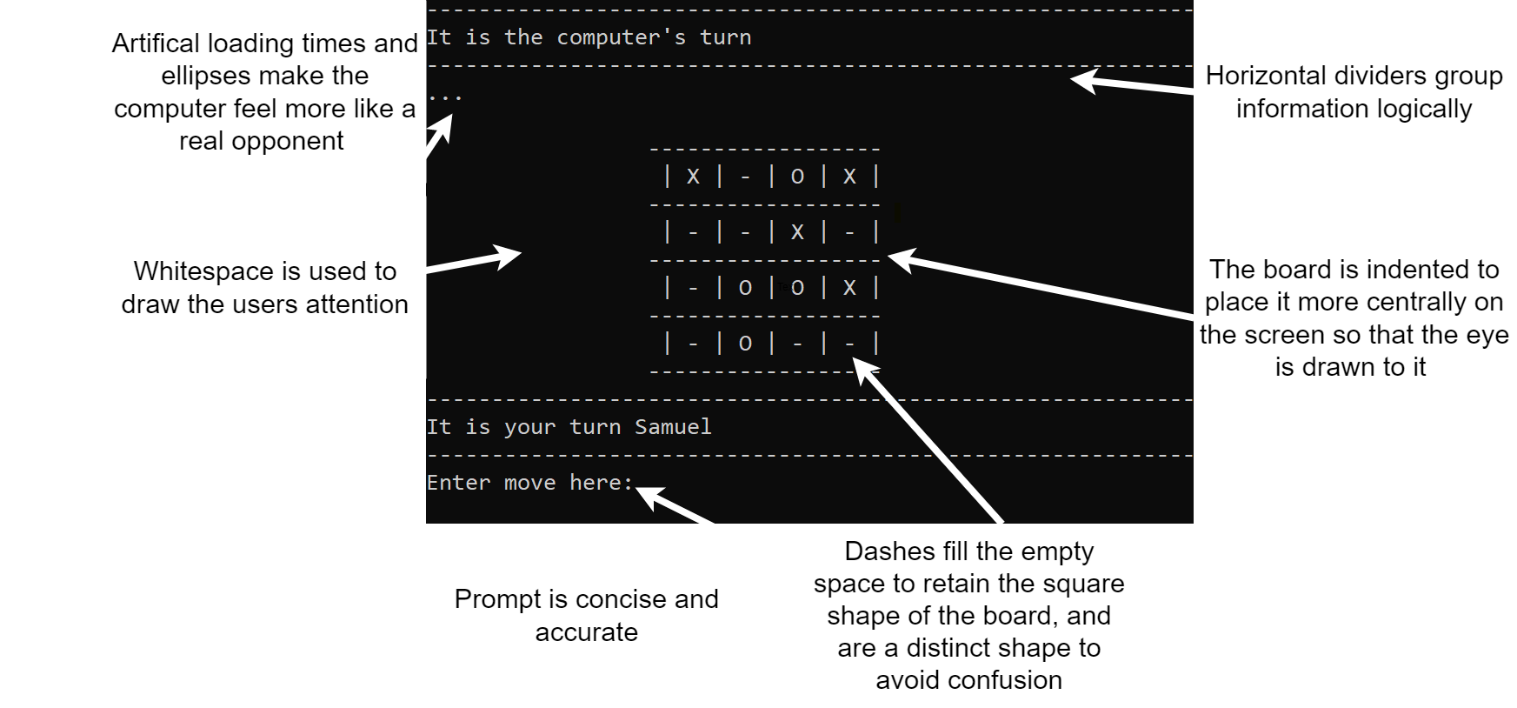
## IPO Chart

|  |  |  |
| --- | --- | --- |
| INPUT | PROCESSING | OUTPUT |
| User input | Check if it is an integer  Check if it is within the specified range | “That is not a valid input. Please try again”  None (if the input is correct then it is passed to the caller with no output to user) |

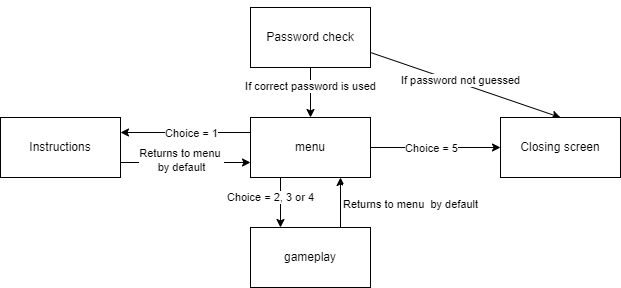
## Screen Designs

Menu: 

Gameplay:



## Storyboard



This storyboard shows the different screens that the user sees and how the link to each other. Despite the far more complex subroutine interactions, there are only 5 key screens that the user will see. A storyboard such as this helps when designing the interface in a way that can fluidly move between these different “scenes” without being jarring to the user. Understanding how the user will see the program also helps with making an accessible and intuitive solution.

## Pseudocode

Pseudocode is located in the attached text file

## Flow Chart

# Implementing software solutions

## Data dictionary

main()

|  |  |  |
| --- | --- | --- |
| Data item | Data type | Description |
| break\_line | String | Stores a set length horizontal divider that is used for readability and consistency |
| exit\_flag | Boolean | Flag that controls the exit of the program |

menu()

|  |  |  |
| --- | --- | --- |
| Data item | Data type | Description |
| break\_line | String | Stores a set length horizontal divider that is used to ensure readability and consistency |
| exit\_flag | Boolean | Flag returned to the caller |
| answer | Integer | The user’s choice in the range 1 - 5 |
| size | Integer | The chosen board size in the range 3 - 30 |
| player\_1\_is\_comp | Boolean | If player 1 should be computer controlled |
| player\_2\_is\_comp | Boolean | If player 2 should be computer controlled |

print\_grid ()

|  |  |  |
| --- | --- | --- |
| Data item | Data type | Description |
| space | String | Stores a string of empty characters used for formatting |
| size | Integer | The length of one side of the board |

circle\_or\_cross()

|  |  |  |
| --- | --- | --- |
| Data item | Data type | Description |
| value | Can be any | The value to be converted into a symbol |

password\_check()

|  |  |  |
| --- | --- | --- |
| Data item | Data type | Description |
| break\_line | String | Stores a set length horizontal divider that is used to ensure readability and consistency |
| attempts | String | Used to ensure correct pluralization |
| verify | Boolean | Flag to be returned. Conveys whether the user has used the correct password. |
| guess | String | Stores the user’s most recent guess |
| num\_of\_guesses | Integer | How many guesses the user has to get the correct password |
| password | String | Contains correct password to compare guesses to |

tic\_tac\_toe\_game()

|  |  |  |
| --- | --- | --- |
| Data item | Data type | Description |
| break\_line | String | Stores a set length horizontal divider that is used to ensure readability and consistency |
| size | Integer | The length of one side of the board |
| board | Array(Integer) | An array that stores the grid that the game is played on |
| game\_over | Boolean | Controls gameplay loop. Ends when it becomes True |
| player\_1\_is\_comp | Boolean | If player 1 should be computer controlled |
| player\_2\_is\_comp | Boolean | If player 2 should be computer controlled |
| namep1 | String | Name of player 1 |
| namep2 | String | Name of player 2 |
| solo\_computer | Boolean | If there is exactly one computer-controlled player |
| winner | Integer | An integer that represents the winning player. 1 represents player 1, -1 for player 2, and 999 for a draw |

computer\_turn()

|  |  |  |
| --- | --- | --- |
| Data item | Data type | Description |
| break\_line | String | Stores a set length horizontal divider that is used to ensure readability and consistency |
| comp\_num | Integer | Which player the computer is taking control of. Changes counters used and certain texts shown to the user. |
| size | Integer | The length of one side of the board |
| board | Array(Integer) | An array that stores the grid that the game is played on |
| solo\_computer | Boolean | If there is exactly one computer-controlled player |
| counter | Integer | Placed into the square that the computer chooses for its move |

computer\_move()

|  |  |  |
| --- | --- | --- |
| Data item | Data type | Description |
| size | Integer | The length of one side of the board |
| board | Array(Integer) | An array that stores the grid that the game is played on |
| move | Integer | The index referring to the place on the board the computer chose |

get\_user\_move()

|  |  |  |
| --- | --- | --- |
| Data item | Data type | Description |
| break\_line | String | Stores a set length horizontal divider that is used to ensure readability and consistency |
| size | Integer | The length of one side of the board |
| board | Array(Integer) | An array that stores the grid that the game is played on |
| move | Integer | The index referring to the place on the board the computer chose |
| name | String | User’s name used to prompt them |

check\_win()

|  |  |  |
| --- | --- | --- |
| Data item | Data type | Description |
| board\_full | Boolean | If the board has no empty squares |
| game\_status | Integer | An integer that represents the current game status. 1 represents player 1 has won, -1 for player 2, 999 for a draw and 0 for a game that is still in play |
| size | Integer | The length of one side of the board |
| board | Array(Integer) | An array that stores the grid that the game is played on |
| diagonal\_total | Integer | Stores the total of the diagonal from the top left to the bottom right of the board |
| other\_diagonal\_total | Integer | Stores the total of the diagonal from the top right to the bottom left of the board |
| row\_total | Integer | The total value of a row on the board |
| column\_total | Integer | The total value of a column on the board |

display\_text\_file()

|  |  |  |
| --- | --- | --- |
| Data item | Data type | Description |
| file\_path | String | The path of the file to be read |
| f | Text file | The opened text file |
| line | String | The most recent line read from the text file |

get\_int\_input\_in\_range()

|  |  |  |
| --- | --- | --- |
| Data item | Data type | Description |
| break\_line | String | Stores a set length horizontal divider that is used for readability and consistency |
| valid\_input | Boolean | If the input is considered valid |
| prompt | String | The prompt used to get the input |
| min | Integer | The lower bound of the range |
| max | Integer | The upper bound of the range |
| first\_guess | Boolean | If this is the first time the user has tried to enter an input |

Is\_digits\_only()

|  |  |  |
| --- | --- | --- |
| Data item | Data type | Description |
| text | String | The text to be checked |
| is\_digits | Boolean | If each character in text is a digit |

## User documentation

The tic-tac-toe game requires a password before play can begin. The default password is 1234. After entering the main menu, there are a number of available options. An option can be chosen be entering the corresponding number. For example, entering ‘5’ will select the “Exit” option. Choosing “Instructions” will print instructions on how to play the game, entering 2, 3 or 4 will start the corresponding type of the game, and “Exit” will close the program. After beginning one of the game types, you will be prompted for a board size. This should be entered as a number between 3 and 30. This size is the length of one side of the board. For example, choosing 3 would create a 3\*3 board with 9 squares. Afterwards, a name will be requested. Enter a name and the game will begin. Once your turn begins, enter a number referring to the square you would like to place your tile in. Counting starts in the top left, so on a 3\*3 board enter “5” to place a counter on the central square. Counters can’t be placed on squares with counters belonging to either player already there. Once your opponent makes a move, the turn will pass back to you. This sequence will continue until a player makes an unbroken straight line of tiles across the board in which case they win, or there are no spaces left, in which case it is a draw. Every option except exiting will return back to the menu where a new choice can be made.