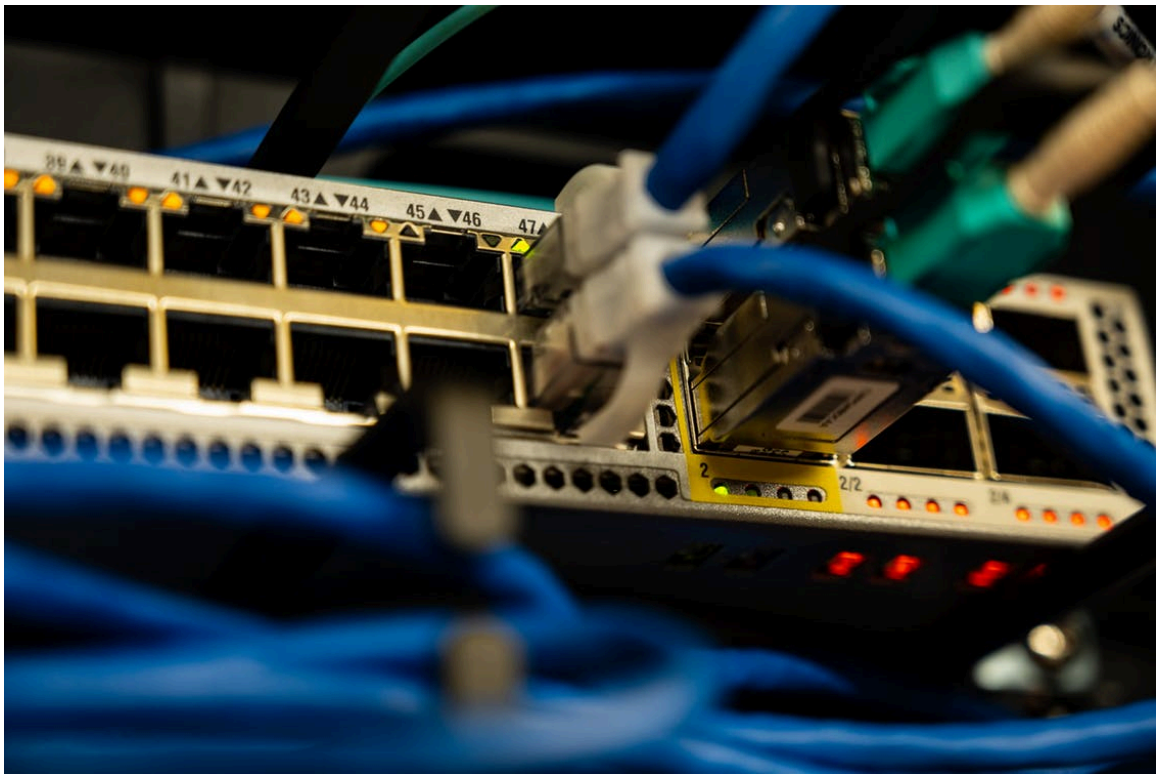


NETWORK GAMES

DESIGN DOCUMENTATION

JAMES WHYTE (u1913954), GAUTAM GOPINATHAN (u1931951)
& MURRAY GREEN (u1920853)
SOFTWARE DEVELOPMENT

1 TABLE OF CONTENTS

2 PLANNING 3

2.1 OVERVIEW 3

2.2 USER REQUIREMENTS 3

3 ANALYSIS 5

3.1 CHESS 5

3.1.1 THE BOARD 5

3.1.2 THE PIECES 5

3.1.3 MOVEMENT AND CAPTURE 6

3.1.4 THE START AND THE END – CHECK, STALEMATE AND CHECKMATE 7

3.1.5 SPECIAL RULES – CASTLING AND 'EN PASSANT' 7

3.1.6 CHESS NOTATION 7

3.1.7 CHESS SERVERS AND CLIENTS 8

3.2 BATTLESHIPS 9

3.2.1 GRID CONFIGURATION 9

3.2.2 THE SHIPS 10

3.2.3 GAMEPLAY 10

3.3 TIC-TAC-TOE 11

3.3.1 THE GRID 11

3.3.2 GAMEPLAY 11

3.4 NETWORK ..... **Error! Bookmark not defined.**

3.4.1 PORT NUMBER 12

3.4.2 METHODS 12

3.4.3 SET PASSWORD 12

3.4.4 ENCRYPT 12

3.4.5 DECRYPT 12

3.4.6 HOST 12

3.4.7 CONNECT 12

3.4.8 SEND 12

3.4.9 RECEIVE 12

3.4.10 CLOSE....13
3.5 Chat....**Error! Bookmark not defined.**
4 DESIGN....15
4.1 TECHNICAL REQUIREMENTS....16
4.2 DEPENDENCY DIAGRAM....22
4.3 PROJECT DEADLINES....23
5 TESTING....24
6 REVIEW....25
6.1 JAMES....25
6.2 GAUTAM....25
6.3 MURRAY....25
7 CONTRIBUTIONS....26
7.1 GAUTAM....26
7.2 JAMES....26
7.3 MURRAY....26

2 PLANNING

2.1 OVERVIEW

The program is required to be a suite of games, which can be played between two people over a network. The suite of games must include chess, tic-tac-toe and battleships. These games must be entirely based in a terminal, with no graphical component (preferably to create a "retro" theme), but ASCII art can and should be included. If extra games are created, they should be able to "slot" into the program, by creating some sort of framework for adding them into the system. Since the games are played over a network, the traffic between the two systems must be protected by encrypting the information as it travels through.

2.2 USER REQUIREMENTS

1. The program must open to a menu that allows a connection between two devices
    - 1.1. Must show the title of the program
    - 1.2. The menu must contain an option to quit
    - 1.3. The menu must contain a method to connect between the two devices over the internet
        - 1.3.1. The users must be able to specify the IP of the other device
        - 1.3.2. There must be a method to which the program can cancel an attempted connection
    - 1.4. The user must be able to specify the option they have selected by using a keyboard.
2. The connection between the two users must be encrypted
    - 2.1. The users must enter a unique password which will be used for the encryption
    - 2.2. The user input must be encrypted/decrypted and sent/received without user intervention
3. Once connected, one user must have the ability to choose the game that the two users play in another menu.
    - 3.1. The "choosing" user must be able to see a list of games which are able to be played
    - 3.2. The game that the users wish to play must be selectable via the menu
    - 3.3. The user who is not choosing must have some screen or information to show that the other user is choosing a game
4. Chess must be a playable game in the game menu
    - 4.1. The users must see an 8x8 board, set in the standard chess starting configuration
    - 4.2. It must be possible to reference every square on the chessboard via standard notation (a-h, 1-9)
    - 4.3. The rules must adhere to the international chess rules

5. Battleships must be a playable game in the game menu
    5.1. The users will be shown a 10x10 grid
    5.2. The users will set up their playing board by first inputting the row, followed by the column.
    5.3. The users will then play in turn using the same “row followed by column” notation
    5.4. If either player strikes 17 ships, the game will detect it and take appropriate action (begin the “You win” or “You lose” sequence)
6. Tic-Tac-Toe must be a playable game in the game menu
    6.1. The users will be shown a 3x3 grid
    6.2. The users will, in turn, select a grid square using the notation (1-3, 1-3)
    6.3. The game will detect if the winning condition has been met (3 in a row)
7. Once the chosen game ends, the program must return to the game menu
8. The users must be able to communicate using a text chat window
    8.1. They will be shown a separate window for chat
    8.2. They will be able to send messages during the game is being played

3 ANALYSIS

3.1 CHESS

Chess is 'a game for 2 players each of whom moves 16 pieces according to fixed rules across a checkerboard and tries to checkmate the opponent's king' (Merriam-Webster [Online], Chess entry)[1].

3.1.1 THE BOARD

The board itself consists of 64 squares in an 8x8 square, of alternating black and white colours. During setup, the players sit directly opposite each other and the board is placed between them such that there is a flat edge of the square towards each of them and a black square is in the left corner from the perspective of both players. The squares position on the board can be referenced by its *rank* and *file*. A rank is numbered from 1 to 8 and represents the 8 rows across the board (from the white player's perspective). The row closest to white is the first rank. A file uses letters a-h and represents the 8 columns down the board (from the white player's perspective). The leftmost column from the white player's perspective is a.

The use of file and rank notation allows the file-first method of identification for squares on the board (for example a1 is the closest left corner on the board for white. Notation is covered in more detail below)

3.1.2 THE PIECES

At the beginning of a game of chess, each player has 16 pieces on the board. One player has a set of 16 white pieces, while the other player has an identical set of pieces, except in black. There are 6 different types of chess piece:

- **Pawn** (8 per player)
- **Rook** (2 per player)
- **Bishop** (2 per player)
- **Knight** (2 per player)
- **Queen** (1 per player)
- **King** (1 per player)

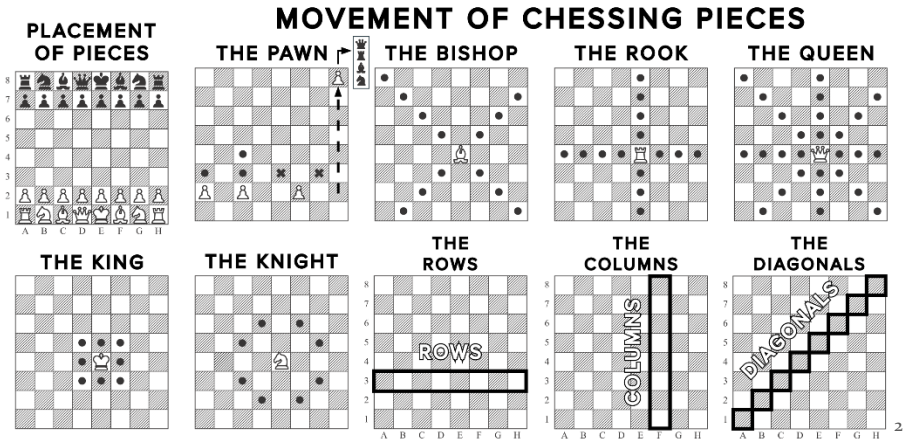
The most important pieces in the game are the kings. If a king is in checkmate or cannot move without being in check (described below), the game ends and the victor is decided.

[1] Merriam-Webster.com Dictionary, *Chess,* Merriam-Webster, Accessed 28 Apr. 2020, <https://www.merriam-webster.com/dictionary/chess/>

3.1.3 MOVEMENT AND CAPTURE

In chess when a piece is captured it is removed from the board. A piece is captured by an opponent's piece being moved onto the space it occupied (with exception to a capture that is done *en passant*, explained below). The movement of a piece must adhere to rules which are specific to their type (with exception to the pawn, the movement of a piece to capture and to move without capture have the same rules). Example movement is also found on the image below:

- **Pawn** – A pawn may move a single square toward the opponent's side of the board when unobstructed, or a single square diagonally toward the opponent's side of the board to capture. The pawn can also move two squares toward the opponent's side of the board if the pawn has not yet moved.
- **Rook** – A rook may move any number of squares in a straight (non-diagonal) line, until obstructed by a piece. An opponent's piece can instead be captured if it is in the path of the rook.
- **Bishop** - A bishop may move any number of squares in a straight diagonal line, until obstructed by a piece. An opponent's piece can instead be captured if it is in the path of the bishop.
- **Knight** – A knight moves to a square 2 ranks and 1 file, or 1 file and 2 ranks away, unless the square is already occupied by a piece. An opponent's piece can instead be captured, however.
- **Queen** - A queen may move any number of squares in a straight line, until obstructed by a piece. An opponent's piece can instead be captured if it is in the path of the queen.
- **King** – A king may move a single square in any direction, unless doing so would put the king in check (explained below).



[2] Unknown, Accessed 28 Apr. 2020, <https://www.pngfind.com/mpng/hJxwim_this-free-icons-png-design-of-chess-pieces/>

3.1.4 THE START AND THE END – CHECK, STALEMATE AND CHECKMATE

In chess, white always starts first. The two players alternate moving a single piece at a time, until there is checkmate or stalemate. Check occurs when a piece threatens the opponents king at the end of a player's turn (i.e. if the player could take a second turn, they could take the king). If it is not possible to stop the king from being taken, it is instead checkmate, and the player who could take the king wins the game. Stalemate occurs when a player cannot end a turn without putting themselves in check and ends the game in a draw.

3.1.5 SPECIAL RULES – CASTLING AND 'EN PASSANT'

Castling and 'en passant' are two rules that are less commonly known in chess. They allow for movement that is very different from the rest of the game:

- Castling allows for the movement of both the king and a rook in one turn, given the following conditions [3]:
    1. The king has not yet moved
    2. The king is not in check
    3. The king does not pass through check
    4. There are no pieces between the king and the rook

  The king moves two squares towards the rook, and the rook moves two spaces towards the king, causing one to 'jump' over the other.
- 'en passant' allows for the capturing of a pawn when it tries to move two spaces past an opponent's pawn. On the immediate turn after, the pawn can take the piece as if it moved only a single square.

3.1.6 CHESS NOTATION

There is a standard notation (named algebraic notation)[4] which is used to record chess games. The notation combines the file-first notation and letters which represent the type of piece (K: King, Q: Queen, R: Rook, B: Bishop, N: Knight – Pawn is represented by the lack of a letter) to denote the movement of a piece. For each pair of turns (i.e. both players move a single piece), the destination of the moved piece and its type is noted, along with the turn number, which starts at 1 and increments. For example:

1. Nc3 F5
2. E4 Fxe4

---

[3] CHESScom, Last updated 2019, *How to Castle in Chess*, CHESS.com, Accessed 28 Apr. 2020, <https://www.chess.com/article/view/how-to-castle-in-chess/>
[4] erik, Last updated 2019, *Chess Notation – The Language of the Game*, Accessed 29 Apr. 2020, <https://www.chess.com/article/view/chess-notation#special-chess-symbols>

Other special notation is used for notable actions [4]:

- x: Capture
- 0-0: Castle kingside
- 0-0-0: Castle queenside
- +: Check
- #: Checkmate
- !: Good Move (More can be used for emphasis)
- ?: Poor Move (More can be used for emphasis)

Since sometimes the multiple pieces of the same type can move to the specified space, when this occurs the file of the original space the piece was located on is also specified (for example 'Bab2' denotes the bishop that started in rank a moved to b2). When this is not enough, the rank or both file and rank can be used to ensure there is no ambiguity.

Portable Game Notation adds to algebraic notation with tags which allow more information. PGN is used to pass information between chess clients.

3.1.7 CHESS SERVERS AND CLIENTS

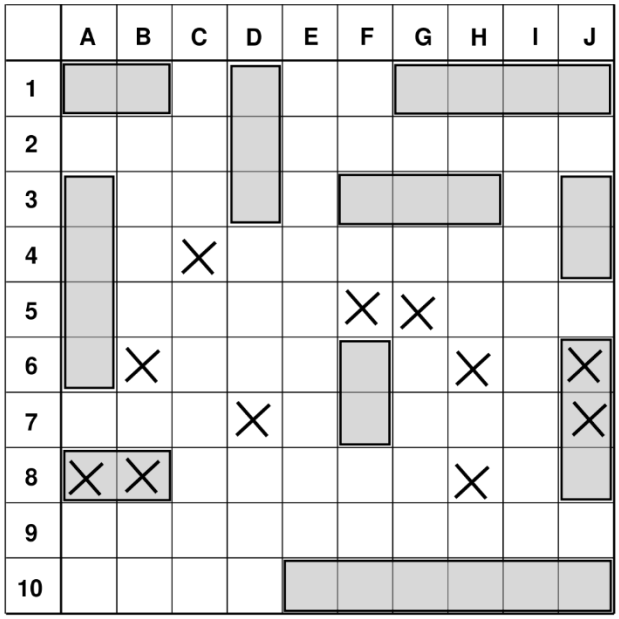
There are many internet chess servers currently available [5], including the Internet Chess Club (ICC), Free Internet Chess Server (FICS) and Chess.Net. Most of the servers allow for connection via both telnet and a web client. This allows users to create and use any number of chess clients, for example Eboard, Knights and PyChess[6], but also allows a standard text-based connection also. These servers allow chat, spectators and of course playing chess. The servers also allow playing variants of chess, such as bughouse and blitzkrieg.

[5] T. Mann, *Internet Chess Servers*, Accessed 29 Apr. 2020, <https://www.tim-mann.org/ics.html/>
[6] M. Fioretti, 2011, *My four favourite graphical Chess clients for Linux*, Accessed 29 Apr. 2020, <https://www.techrepublic.com/blog/linux-and-open-source/my-four-favorite-graphical-chess-clients-for-linux/>

3.2 BATTLESHIPS

Battleships is a strategy type guessing game for two players. It is played on ruled grids (paper or board) on which each player's fleet of ships (including battleships) are marked. The locations of the fleets are concealed from the other player. Players alternate turns calling "shots" at the other player's ships, and the objective of the game is to destroy the opposing player's fleet.[7]

3.2.1 GRID CONFIGURATION



[7]

The game is played on four grids, two for each player. The grids are typically square – usually 10×10 – and the individual squares in the grid are identified by letter and number.[8] On one grid the player arranges ships and records the shots by the opponent. On the other grid, the player records their own shots.

Before play begins, each player secretly arranges their ships on their primary grid. Each ship occupies a number of consecutive squares on the grid, arranged either horizontally or vertically. The number of squares for each ship is determined by the type of the ship. The ships cannot overlap (i.e., only one ship can occupy any given square in the grid). The types and numbers of ships allowed are the same for each player. [7]

---

[7] Wikipedia, 2020. *Battleship (game).* [Online]
Available at: https://en.wikipedia.org/wiki/Battleship_(game)
[Accessed 29 April 2020].

[8] Brown, S., 2019. *The Spruce Crafts.* [Online]
Available at: https://www.thesprucecrafts.com/the-basic-rules-of-battleship-411069
[Accessed 29 April 2020].

3.2.2 THE SHIPS

Each player is given 17 ships, divided into 5 classes.

Class of Ship	Size
Carrier	5
Battleship	4
Destroyer	3
Submarine	3
Patrol Boat	2

[9]

3.2.3 GAMEPLAY

After the ships have been positioned, the game proceeds in a series of rounds. In each round, each player takes a turn to announce a target square in the opponent's grid which is to be shot at. The opponent announces whether the square is occupied by a ship. If it is a "hit", the player who is hit marks this on their own or "ocean" grid (with a red peg in the pegboard version). The attacking player marks the hit or miss on their own "tracking" or "target" grid with a pencil marking in the paper version of the game, or the appropriate colour peg in the pegboard version (red for "hit", white for "miss"), in order to build up a picture of the opponent's fleet.

When all the squares have been hit, the ship's owner announces the sinking of the Carrier, Submarine, Cruiser/Destroyer/Patrol Boat, or the titular Battleship. If all of a player's ships have been sunk, the game is over and their opponent wins. If all ships of both players are sunk by the end of the round, the game is a draw. [7]

[9] Milton Bradley Company, 1990. *BATTLESHIP.* [Online]
Available at: https://www.hasbro.com/common/instruct/battleship.pdf
[Accessed 14 May 2020].

3.3 TIC-TAC-TOE

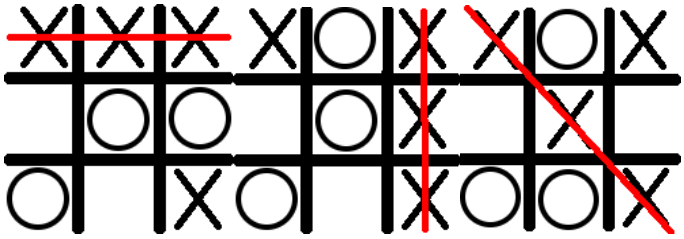
Tic-tac-toe or Noughts and crosses is a game for two players who take turns marking squares on a 3x3 grid. The winner is the player who places 3 marks in a horizontal, vertical or diagonal row.[10]

3.3.1 THE GRID

The game is played on a single 3x3 grid, which both players will see. The squares will be identified using x and y coordinates (0 – 3).

3.3.2 GAMEPLAY

Starting with x, players take turns marking a square on the grid. A player will win if they get 3 marks in a row, either horizontal, vertical, or diagonal. If the grid is full and no more cells can be filled, there is a draw.



In these examples, x wins by getting 3 in a horizontal, vertical and diagonal row.

10 Wikipedia Contributors (2019). Tic-tac-toe. [online] Wikipedia. Available at: https://en.wikipedia.org/wiki/Tic-tac-toe. [Accessed 25 May 2020]

3.4 NETWORK

To facilitate playing over a network, the program should be able to securely send data to the other player. The program will send the string inputted by the user and will be passed as an input to the game. The python socket module will be used to set up a connection between the two players.

3.4.1 PORT NUMBER

The port number to be used can be set when the network class is initialised, but if no port number is set the port number 6969 will be used as it is a lesser used port.

3.4.2 METHODS

The network class will contain 5 different methods that deal with network communications which can be called by the main program.

3.4.3 SET PASSWORD

This will ask the host for a password, which will be used to encrypt/decrypt the network traffic being sent/received. The password must be used by the other user to parse the network traffic correctly.

3.4.4 ENCRYPT

This will take a string as the input and encrypt it using the AES-256 cipher. (using the assigned password) The encrypted cipher text is then sent returned to be sent through the network.

3.4.5 DECRYPT

This will take the received ciphertext and decrypt it using the password entered by the user. This will return the string in its original form to the calling program.

3.4.6 HOST

This will listen for an incoming connection on the port specified when the class was initialised.

3.4.7 CONNECT

This is called when staring a connection, the target IP address is passed as an argument.

3.4.8 SEND

Takes a string and sends it to the other client.

3.4.9 RECEIVE

Waits for a message to be sent. A string will be returned.

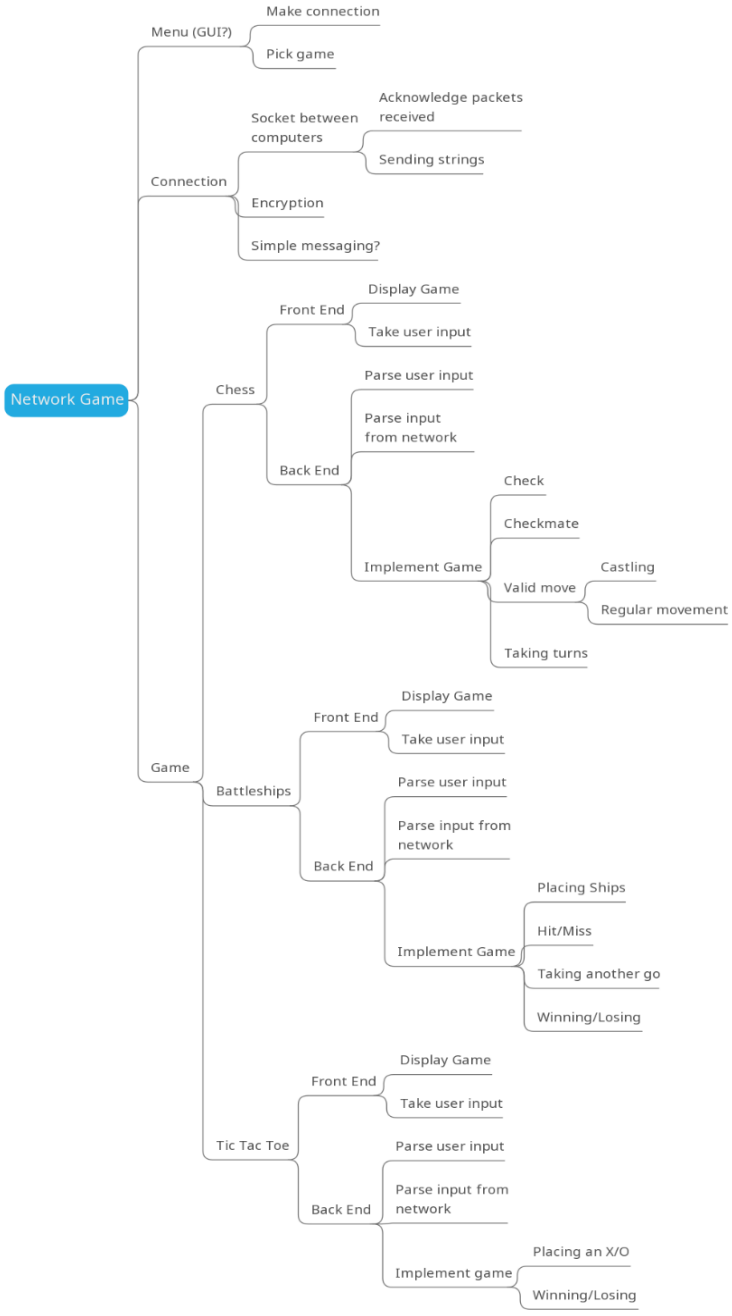
3.4.10 CLOSE

Closes the connection. This should be called before the main program ends.

3.5 CHAT

The users should have the ability to send text messages during the games. To achieve this, a second window must be opened in which messages can be send without interrupting the game. Previous messages will be displayed in order and text can be entered into a box. The chat window will open before the game starts and messages can be sent throughout the game. The network class will be used to send the messages. To open and manage a second window, the tkinter module will be used. If one user disconnects, the message "Disconnected..." will be printed to the chat window.

4 DESIGN


Network Game
Menu (GUI?)
Make connection
Pick game
Connection
Socket between computers
Acknowledge packets received
Sending strings
Encryption
Simple messaging?
Game
Chess
Front End
Display Game
Take user input
Back End
Parse user input
Parse input from network
Implement Game
Check
Checkmate
Valid move
Castling
Regular movement
Taking turns
Battleships
Front End
Display Game
Take user input
Back End
Parse user input
Parse input from network
Implement Game
Placing Ships
Hit/Miss
Taking another go
Winning/Losing
Tic Tac Toe
Front End
Display Game
Take user input
Back End
Parse user input
Parse input from network
Implement game
Placing an X/O
Winning/Losing

4.1 TECHNICAL REQUIREMENTS

1. Menu
    1.1. Main Function
        1.1.1. Input: None
        1.1.2. Return: None
        1.1.3. Use: Provides the user with the option of closing the program, connecting to a remote connection or opening a connection. Allows the creator to pick a game to play. The function then opens the connection and creates the object for the chosen game. Once the players are connected, each one passes strings to the game object. The main function checks what is returned, using the same user for input until a 1 is returned, and continuing the loop until -1 is returned.
    1.2. Clear Function
        1.2.1. Input: None
        1.2.2. Return: None
        1.2.3. Use: Clears the screen
    1.3. Typewriter Function
        1.3.1. Input: Message (String)
        1.3.2. Return: None
        1.3.3. Use: Write each character with a delay from string *Message*
2. Network
    2.1. Network sockets Class
        2.1.1. Set password function
            2.1.1.1. Input: Password (string) (Default: None)
            2.1.1.2. Return: Normal completion or error (integer)
            2.1.1.3. Use: Used to set the unique password used by the encrypt/decrypt function
        2.1.2. Encryption function
            2.1.2.1. Input: Message (string)
            2.1.2.2. Return: Ciphertext (string)
            2.1.2.3. Use: Encrypts the input with a fixed key and IV and returns the encrypted message to be sent over the network
        2.1.3. Decryption function
            2.1.3.1. Input: Ciphertext (string)
            2.1.3.2. Return: Message (string)
            2.1.3.3. Use: Decrypts the input with a fixed key and IV and returns the decrypted message to be received and used by the program
        2.1.4. __Init__ function
            2.1.4.1. Input: Port number (Integer)
            2.1.4.2. Return: None
            2.1.4.3. Use: Creates the socket and defines which port should be used.

2.1.5. Host function
2.1.5.1. Input: None
2.1.5.2. Return: None
2.1.5.3. Use: Listens for a connection on the port specified in __init__.
2.1.6. Connect function
2.1.6.1. Input: Ip address (string)
2.1.6.2. Return: None
2.1.6.3. Use: Connects to a server at the specified IP address.
2.1.7. Send function
2.1.7.1. Input: Message (string)
2.1.7.2. Return: None
2.1.7.3. Use: Sends a string to the other client.
2.1.8. Receive function
2.1.8.1. Input: None
2.1.8.2. Return: Message (string)
2.1.8.3. Use: Waits for a message to be sent and returns it as a string.
2.1.9. Close function
2.1.9.1. Input: None
2.1.9.2. Return: None
2.1.9.3. Use: Closes the connection.

3. Grid
3.1. Init Function
3.1.1. Input: (x, y) (integer) and fill
3.1.2. Return: None
3.1.3. Use: Initialise board of size (x, y) and fill the board with value fill
3.2. Set Cell function
3.2.1. Input: (x, y) (integer) and value
3.2.2. Return: None
3.2.3. Use: Set value at index (x, y) to value
3.3. Get Cell function
3.3.1. Input: (x, y)
3.3.2. Return: Piece value
3.3.3. Use: Retrieve value at index (x, y)
3.4. Set x strip function
3.4.1. Input: x (integer), value and set independent (Boolean)
3.4.2. Return: None
3.4.3. Use: Set all spaces with the same x value. Allows for either a single value to fill all spaces or a different value for each space, using set independent
3.5. Set y strip function
3.5.1. Input: y (integer), value and set independent (Boolean)
3.5.2. Return: None

3.5.3. Use: Set all spaces with the same y value. Allows for either a single value to fill all spaces or a different value for each space, using set independent

3.6. Find piece function

3.6.1. Input: value

3.6.2. Return: (x, y) (integer)

3.6.3. Use: Return the first found co-ordinate with value 'value'

3.7. Swap function

3.7.1. Input: (x1, y1) (x2, y2) (integers)

3.7.2. Return: None

3.7.3. Use: Swap the value at (x1, y1) with the value at position (x2, y2)

4. Chess

4.1. Distance Function

4.1.1. Input: (x1, y1) and (x2, y2) (Integer co-ordinate pairs)

4.1.2. Return: Distance (Integer)

4.1.3. Use: Takes two co-ordinates from a square game board: (x1, y1) & (x2, y2) and finds the distance between them. The distance is based upon the number of cells between the two given cells, including the destination cell. If the two cells are not in a straight line, -1 is returned. If they are in a straight line, the distance is returned

4.2. Direction Function

4.2.1. Input: (x1, y1) and (x2, y2) (Integer co-ordinate pairs)

4.2.2. Return: Direction (String)

4.2.3. Use: This function takes two co-ordinates from a square game board: (x1,y1) & (x2,y2) and finds the cardinal direction between them relative to (x1, y1). The direction is returned

4.3. Chess Class

4.3.1. __init__ Function

4.3.1.1. Input: Fill (String)

4.3.1.2. Return: None

4.3.1.3. Use: Container for Chess game. Creates the board; sets the pieces and sets the first turn

4.3.2. Print Board Function

4.3.2.1. Input: None

4.3.2.2. Return: None

4.3.2.3. Use: Print the board. a-h on top row. 1-8 on down the side.

4.3.3. Input Function

4.3.3.1. Input: Input (String)

4.3.3.2. Return: Change (Integer)

4.3.3.3. Use: Take a string input and compute it as a turn. Uses private functions to check that the input is valid, and then move the piece. The turn is then changed to the other user

4.3.4. Intercept Piece Function
4.3.4.1. Input: (x, y) (Integer) and Line (String)
4.3.4.2. Return: Distance Away (Integer) and Piece (Integer)
4.3.4.3. Use: Take a co-ordinate and a cardinal direction and return the closest piece in that direction, as well as the distance.
4.3.5. Valid Move Function
4.3.5.1. Input: (x1, y1), (x2, y2) Pair of co-ordinates (Integer)
4.3.5.2. Return: Valid (Integer)
4.3.5.3. Use: Find whether the move from (x1, y1) to (x2,y2) would be valid, and return 1 if true and 0 if false
4.3.6. Piece Number Function
4.3.6.1. Input: (x, y) (Integer)
4.3.6.2. Return: Piece Number (Integer)
4.3.6.3. Use: Returns the Unicode value of the piece in location (x, y)
4.3.7. Check for Check Function
4.3.7.1. Input: None
4.3.7.2. Return: White Check, Black Check (Integers)
4.3.7.3. Use: Checks the board to find whether the white king or black king is in check
4.3.8. Pawn Promote Function
4.3.8.1. Input: None
4.3.8.2. Return: None
4.3.8.3. Use: Check for any pawns at the opponents end rank. Promote them to queen
4.3.9. Castling Check Function
4.3.9.1. Input: None
4.3.9.2. Return: Left Check, Right Check (Integer)
4.3.9.3. Use: Return whether the current user can castle on either side
4.3.10. Game Over Function
4.3.10.1. Input: None
4.3.10.2. Return: None
4.3.10.3. Use: Prints which user has won

5. Battleships
5.1. Battleships class
5.1.1. __init__ function
5.1.1.1. Input: User number (integer)
5.1.1.2. Return: None
5.1.1.3. Use: Initializes the user number to facilitate other functions based on the value passed from the main function
5.1.2. Conversion function
5.1.2.1. Input: Board (list), Row (integer), Column (integer)
5.1.2.2. Return: Ship type (character)

5.1.2.3. Use: Converts the numbers used in the display function to characters

5.1.3. Display function

5.1.3.1. Input: Board (list)

5.1.3.2. Return: None

5.1.3.3. Use: Displays the board with the required formatting

5.1.4. Ships placement function

5.1.4.1. Input: Row (integer), Column (integer), Orientation (character)

5.1.4.2. Return: Normal completion or completion with errors (integer)

5.1.4.3. Use: Places ships in the specified location from the user's input (front-end of ships placing)

5.1.5. Placement function

5.1.5.1. Input: Board (list), Row (integer), Column (integer), Orientation (character), Ship (character)

5.1.5.2. Return: Normal completion or completion with errors (integer)

5.1.5.3. Use: Facilitates the ships placement function (back-end of ships placing)

5.1.6. Attack function

5.1.6.1. Input: Row (integer), Column (integer)

5.1.6.2. Return: Hit/Repeat (integer) or Miss (integer)

5.1.6.3. Use: Used to strike ships with co-ordinates from the user

5.1.7. Game over function

5.1.7.1. Input: Resignation value (character, default: None)

5.1.7.2. Return: Game completion (integer)

5.1.7.3. Use: Used to check if 17 ships are down (player wins) or if the game is forfeited and ends the game

5.1.8. Check ships down function

5.1.8.1. Input: Board (list), Play (list)

5.1.8.2. Return: Ships down (Integer, 5 options)

5.1.8.3. Use: Checks if a class of ships are down

5.1.9. Display ships down function

5.1.9.1. Input: Value from shipsdown() (Integer)

5.1.9.2. Return: None

5.1.9.3. Use: Displays the classes of ships down using values from shipsdown()

5.1.10. Input to Game function

5.1.10.1. Input: userInput (string)

5.1.10.2. Return: Play again, do not play again, or terminate game (Integer)

5.1.10.3. Use: Facilitates all the game code's functions by taking input from the user and redirecting as necessary

6. Tic-Tac-Toe

6.1. NoughtsAndCrosses class

6.1.1. __init__ function

6.1.1.1. Input: None
6.1.1.2. Return: None
6.1.1.3. Use: Creates the grid and sets the character that will be used.

6.1.2. Check_full function
6.1.2.1. Input: None
6.1.2.2. Return: True or False
6.1.2.3. Use: Checks if the board is full, returns false if there are any empty cells.

6.1.3. Draw_grid function
6.1.3.1. Input: None
6.1.3.2. Return: None
6.1.3.3. Use: Prints the contents of the grid to the screen.

6.1.4. InputToGame function
6.1.4.1. Input: input_string (string)
6.1.4.2. Return: integer
6.1.4.3. Use: Takes string input for each turn, returns 1 if the turn is valid, 0 if the turn is not valid and –1 if the game is over.

7. Chat

7.1. Chat class

7.1.1. __init__ function
7.1.1.1. Input: name (string), usr (integer), ip_addr (string)
7.1.1.2. Return: None
7.1.1.3. Use: Creates the chat window and connects the clients

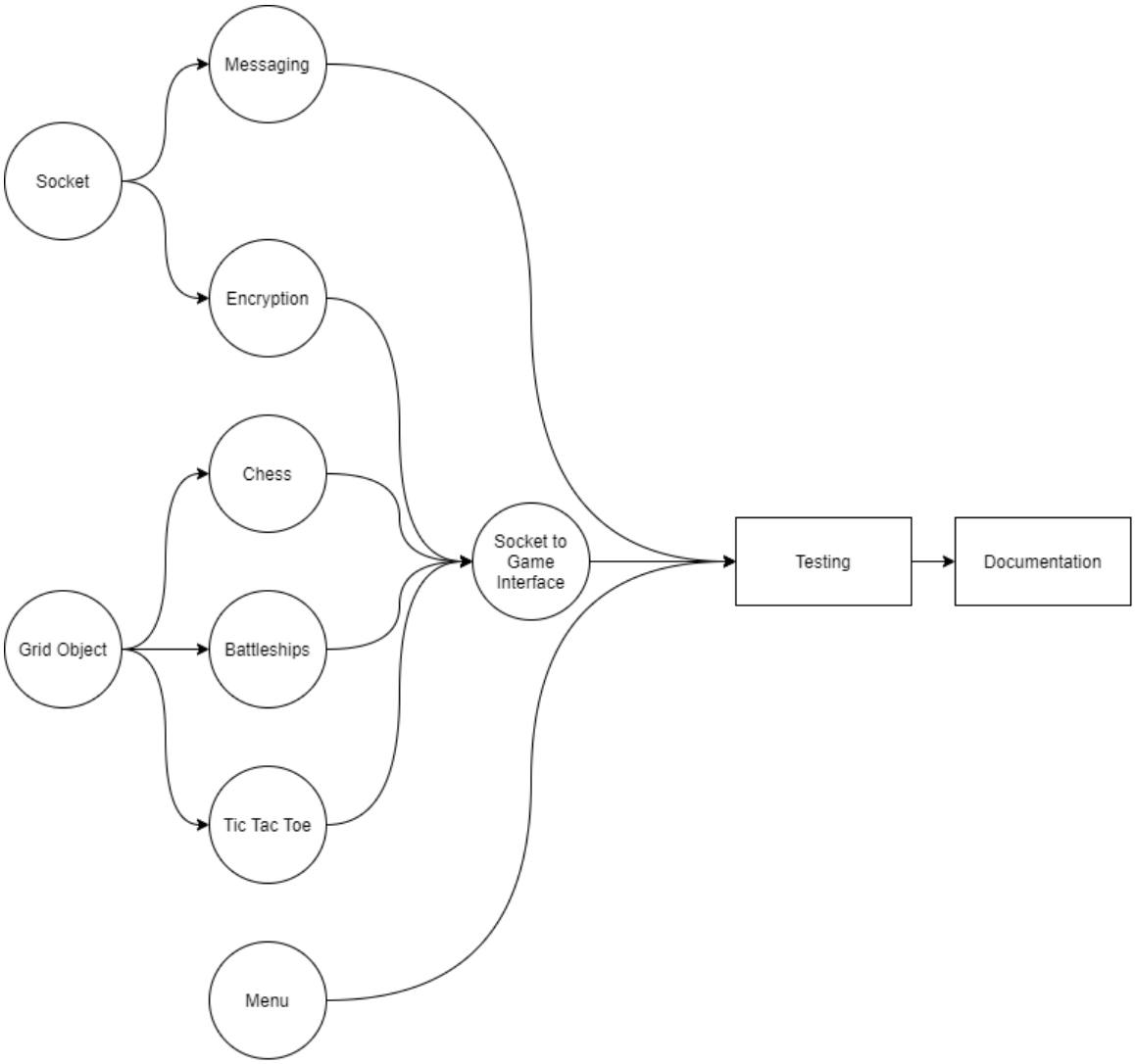
7.1.2. Send_message function
7.1.2.1. Input: None
7.1.2.2. Return: None
7.1.2.3. Use: Gets the text input from the GUI and sends it to the network send function.

7.1.3. Recv_thread function
7.1.3.1. Input: None
7.1.3.2. Return: None
7.1.3.3. Use: Listens for message and prints it to the chat window

7.2. Open chat function
7.2.1. Input: name (string), usr (int), ip_addr (string)
7.2.2. Return: None
7.2.3. Use: Initialises the chat class and opens the chat window.

4.2 DEPENDENCY DIAGRAM

This diagram shows the dependency between tasks. For a task to be started, it is necessary to complete the tasks that it is dependent on. The notable exceptions are Testing and Documentation, both of which cannot be completed without the prior task's completion, but are started at the beginning of the project (i.e. they do not need to rely on the completion of prior tasks to start)



4.3 PROJECT DEADLINES

TASK NO.	NAME	DEADLINE	ASSIGNED TO	COMPLETION DATE
1	Socket	11/4/20	Murray	30/03/20
2	Encryption	17/4/20	Gautam	30/03/20
3	Messaging	24/5/20	Murray	28/05/20
4	Chess: Front End	24/5/20	James	18/05/20
5	Chess: Back End	24/5/20	James	18/05/20
6	Menu	24/5/20	James	21/05/20
7	Battleships: Front End	10/5/20	Gautam	27/05/20
8	Battleships: Back End	10/5/20	Gautam	27/05/20
9	Tic Tac Toe: Front End	10/5/20	Murray	25/05/20
10	Tic Tac Toe: Back End	10/5/20	Murray	25/05/20
11	Testing	24/5/20	Team	28/05/20
12	Grid Object	2/4/20	James	27/03/20
13	Socket to Game Interface	24/5/20	James	21/05/20
14	Documentation	31/5/20	Team	31/05/20

Changes

All dates have been moved back due to COVID-19.

5 TESTING

Testing can be found in the second PDF.

6 REVIEW

6.1 JAMES

Overall, the project was a success. The program does what was set out for it to do, and it successfully passes all tests. It does have to be said, however, that the execution of this project was not ideal. After the loss of time and a member, the original scale of the program had to be reduced. For example, in future iterations of the program it would be best if the chess program could identify checkmate, and a simpler system to add new games would be a welcome addition. Additionally, it should be mentioned that the original lack of any focused leadership led to little work occurring at the start of the timeline, but after realising this fact we were able to get back on track (albeit closer to the wire).

6.2 GAUTAM

Relatively happy with how the project has turned out, considering the initial few hurdles of losing a member and having to come up with a backup plan within a rather short timeframe. Despite our lack of communication in the initial stages, the final product was fairly cohesive and well put-together. Some features had to be dropped and/or rethought as a result. (the battleships code saved the opponent's board natively, which is not ideal and the input function in the main program could be more featured) However, all things considered, the project was successful as all the pieces came together in the end to produce a very usable and enjoyable program.

6.3 MURRAY

Overall, I would consider the project to be successful. Despite losing a member the project was completed on time and fulfils all the requirements. At first there were problems with lack of communication and leadership, but these were solved as time went on. Unfortunately, these issues caused some parts of the program to be rewritten in order to be compatible with other modules, but this was only a minor problem. The final program is fully functional and provides a satisfying user experience.

7 CONTRIBUTIONS

7.1 GAUTAM

- Programs
  - Battleships Program (battleships.py)
  - Portions of the Menu & Socket-to-Game interface (main.py)
  - Portions of the Network socket interface (network.py)
- Documentation
  - Battleships analysis
  - User Requirements (2 & 5)
  - Technical Requirements (2.1.1, 2.1.2, 2.1.3, 5)
- Testing
  - Any tests by Gautam

7.2 JAMES

- Programs
  - Chess Program (chess.py)
  - Menu & Socket-to-Game interface (main.py)
  - Grid Class (grid.py)
- Documentation
  - Document Layout
  - User Requirements (except 1.3, 1.5, 1.6)
  - Technical Requirements (1, 3 & 4)
  - Chess Analysis
  - Project Deadlines
  - Dependency Diagram
  - Design Mind-Map
  - Review - James
- Testing
  - Document Layout
  - Any tests by James

7.3 MURRAY

- Programs
  - Tic-Tac-Toe (noughtsandcrosses.py)
  - Network socket interface (network.py)
  - Chat Module (chat.py)
- Documentation
  - Tic-Tac-Toe Analysis
  - User Requirements (6 & 8)
  - Technical Requirements (6 & 7)

- Testing
    - Any tests by Murray