UNIVERSITY
OF HULL

Playing the Turing Game (Nag -21-266)

Final report

Submitted for the BSc in

Computer Science

April 2022

by

Callum Oliver Thomson Gray

Word count: 14118

Abstract

The Turing game is a thought experiment created by Alan Turing to test whether a computer or Artificial intelligence (AI) is capable of thinking or exhibiting intelligent behaviour comparable to that of a human. This is done by having a human, the Questioner, ask questions to an unknown third party, the Subject, this third party is either another human or it could be an AI. The job of the Questioner is to determine whether they are talking to an AI or to another human. If the Questioner cannot correctly identify the Subject or is not certain of their identity then the Subject is said to have passed the Turing Test. The developer has investigated creating a proof-of-concept application to implement the Turing game concepts into an educational tool for use in a classroom environment.

This Paper aims to show via a working-proof-of-concept network-based application how an educational tool may be implemented that allows the students to be paired with either a human or AI and have a conversation which can be monitored remotely by the teacher.

Keywords: Turing, AI, Artificial Intelligence, networking, client, server, chatbot

Contents

1 Introduction

1.1 Background to the project

The use of AI (Artificial Intelligence) In recent years has grown vastly and the chances are that most people will be using and interacting with AI without ever realising it. With this in mind teaching people about the concepts of AI is vitally important for them to understand this technology as it becomes more and more prevalent in everyday applications.

The Turing game is the name given to a thought experiment created by and named after Allan Turing. He devised this test as a means of evaluating whether an artificial intelligence (AI) could be considered conscious, or to be able to exhibit sufficient intelligence in order to compete with a human being.

In order to aid in teaching people about AI a network-based application could be used that allows students to connect with another random student or a chatbot. The student and partner would have a conversation at the end of which they would have to determine what they were talking to. The Teacher would be able to monitor the conversations that were happening remotely.

1.2 Aims and objectives

1.2.1 Objective 1 – Design and create an application that would act as the client connecting to the server application remotely

1. Understand the concepts behind network-based applications and distributed systems
2. Design a system that would allow for the sending and receiving of messages across a TCP connection asynchronously.
3. Design a graphical user interface to be utilised by the application
4. Implement an MVVM architecture pattern into the design for the User interface
5. Implement the design and test

The first part of Objective 1 was to research and understand how a network-based application could be produced which would allow multiple end users to be connected and send messages to each other.

The second part was to take what had been learnt in the research component in order to design a framework that incorporated these principles

The third and fourth parts involved researching how to implement a graphical user interface (GUI) into the design and implementing the MVVM structure to achieve this.

The final part is to build a solution based on the design produced and test it using a simple console application that it can connect to in order simulate the connection to a server.

1.2.2 Objective 2 – design and create an application to act as the server / Teacher's application

1. Design a system that would be able to allow the pairing up of two clients
2. Expand the design to allow the system to receive and redirect connected clients to the available rooms
3. Expand the design further in order to dynamically create rooms as required
4. Implement an MVVM architecture pattern for the GUI
5. Build and test the designed solution

The first part of Objective 2 is to design a concept for pairing the clients together in a "Room" and allowing messages to pass between them. The second part it to implement a design with a single room and a primary connection that could be used to receive the initial connection request and then redirect It to a room.

The third part involves expanding the solution design so that it can dynamically create and manage the "Rooms"

The fourth parts involved implementing the MVVM structure to the solution to create a GUI for the end user to work with.

The final part is to build a solution based on the design produced and test the application using the client application from Objective 1

1.2.3 Objective 3 - design and create an application that can host a chatbot to act as the AI in the Turing game scenario

1. Research chatbots and find some off the shelf solutions which could provide the chatbot functionality
2. Design a program that can be used to wrap the chatbot and extract the functionality but allow it to connect to the server/Controller in the same manner as any other client
3. Implement the design and test using the client and controller applications from the previous objectives.

1.2.4 Objective 4 - design a ruleset or scenario that uses the above-mentioned tools to create an interactive game for the students with scores etc.

1. Implement a score into the Client that keeps score of when they Guessed correctly as an Interviewer or managed to fool the interviewer as a Subject
2. Implement the ability for the Teacher to be able to specify the number of chatbots in the current session.
3. Implement the ability for the server to randomise and shuffle the rooms to mix up the Partners.

1.3 Report Structure

This Report will be structured in the following way. Section 2 Is made up of the Literature Review which discusses the background and current state of the fields with particular focus on the inception of the Turing game and network-based communication. Section 3 will cover the Requirements for the project which will lead on to section 4 of the document that shall cover the design and conception of the project solution. Section 5 shall cover the Implementation and testing of the project. Section 6 is the conclusion and critical evaluation of this report. Section 6 will also aim to identify any short coming in the project and areas that can be improved upon in future if the development process was allowed a greater time period.

2 Literature review

This section of the report will cover the current history and state of the domain that this project will be influenced by. This Project focuses on developing a series of applications that will allow multiple machines to connect and communicate to play the Turing Game. It is therefore important to understand What the Turing game is and the point behind it as well as the best way to achieve network communication between the applications.

2.1 Network Based Communication

This project relies heavily on network communication to allow the various applications that make up the solution to communicate. Two of the ways that communication can be achieved between applications are directly via network sockets or by using an Application Programming Interface (API) such as asp.net. The backgrounds and fundamentals of both approaches will be broken down in the following sections.

2.1.1 Transmission Control Protocols (TCP) / Internet Protocol (IP) Model

TCP is one of the fundamental standards for enabling applications and devices to communicate across networks. This is done by sending packets of data and confirming delivery of the packets upon receipt at the endpoint. Fortinet.com states the following.

"TCP is one of the basic standards that define the rules of the internet and is included within the standards defined by the Internet Engineering Task Force (IETF). It is one of the most commonly used protocols within digital network communications and ensures end-to-end data delivery." (Fortinet, 2022)

IP is the method for communicating data across the internet specifically. Every device has a unique IP address this allows the device to be identified and send and receive data across the internet. The TCP/IP Model Consists of four layers these are The Datalink Layer, The Internet Layer, The Transport Layer and the Application Layer.

2.1.1.1 The Data Link Layer

The Datalink Layer is used to handle the act of sending the data and receiving the data. It is a combination of the physical and datalink layers from the Open System interconnection model (OSI). The Datalink Layer also covers the physical medium that connects the network, such as Ethernet cables or Wi-Fi connections and how to interact with the target devices drivers.

2.1.1.2 The Internet Layer

The Internet Layer is the layer that holds responsibilities for sending and directing the packages of data between the devices in the network. It controls the movement of data across the networked devices and ensures the data reaches the endpoint. IP forms the main protocol for this layer as it gives the data the addresses to navigate to.

2.1.1.3 The Transport Layer

The Transport Layer is responsible for maintaining a reliable data connection between the current device and the target device. This is the Layer which handles the packaging up of data and sequencing the packets so that they can be reordered at the destination if for whatever reason they arrive out of sync. It uses the number of packets and the amount of data to ensure that the received data matches the sent data and that there have been no errors or missing packets. This happens in the form of an acknowledgement from the target device.

2.1.1.4 The Application Layer

The Application layer is the only layer of the TCP/IP Model that the end user will likely ever interact with as this is the layer that covers the applications/ interfaces that the user may use such as an email system or any other kind of application that communicates across a network.

2.1.2 Http Protocols/ Web API

A web API is an application programming interface that allows an application to extend its functionality to include web controls. This includes using Http Protocols to communicate remotely as well as using this to remotely call functions in a server or other device. Commonly used Web API's include Asp.net which is an API for C# programming.

HTTP stands for Hyper Text Transfer Protocol this is a client-server protocol that uses individual messages or requests as opposed to a persistent data stream. The requests are sent from the client to the server and then are fulfilled by sending a response from the server to the client. HTTP Sits in the Application layer of the TCP/IP model this is useful as the request would have to be programmed to route itself through all of the various proxies and devices between the source and destination machines such as routers and modems.

Asp.net and other API's extend functionality for the various Http request types these are.

HTTP Get this is a request to receive some data from a server usually the query for the data is included in the URI for this prefixed with a "?".

HTTP Post this is a request which provides data to a server usually to store it the data is usually included in the body of the HTTP request and is not visible in the URI

HTTP Put this is a request to update data on a server the new data is included in the body of the request and is hidden from the URI.

HTTP Delete this is a request to delete some data from the server the parameters for what to delete should be included in the request body or as a query in the URI prefixed by a "?".

2.1.3 Network sockets

A network socket is a structure at the software level that acts as the endpoint for a node or device within a computer network. There are various types of socket which can be used to relay data between applications. These are Datagram Sockets, stream sockets and Raw sockets.

2.1.3.1 Datagram Sockets

A datagram socket is a connection less socket that makes use of User Datagram Protocol (UDP). The socket is considered connection less because it does not maintain a connection to the target device it simply sends the datagram and then its job is done. This is done by giving every single packet an address and routing them individually this means that the data can be received in the wrong order or have packets missing. To receive data the receiving socket does not need to be bound to a specific address which can be advantageous if the receiving application needs to listen to several clients at the same time.

2.1.3.2 Stream Sockets

A stream socket is a connection-oriented socket that makes use of TCP. Because the connection is generally maintained when using a stream socket the data flow can be validated upon being sent meaning that the receiving socket will acknowledge receipt of the data and send a confirmation to the sending socket. This is useful as it ensures that data arrives in order and that no packets are lost along the way.

2.1.3.3 Raw Sockets

A raw socket is a network socket with no specified Transport layer protocols. Data can still be sent and received using these sockets however extra work is involved in developing protocols that the transmission will have to use. This is useful for utilities such as testing connection speed or whether an address is valid where the actual data sent is not important simply that the message is sent received and responded to.

2.2 Model View View-Model (MVVM)

Part of the Project is to develop a user interface with this in mind research into the MVVM programming pattern has been undertaken. This is a structure that abstracts the user interface, the display logic and the data from each other. This is done to allow the view model to be updated and changed with little to no change required to the business logic and the same for the data access layer. The structure is as shown below in figure 1.

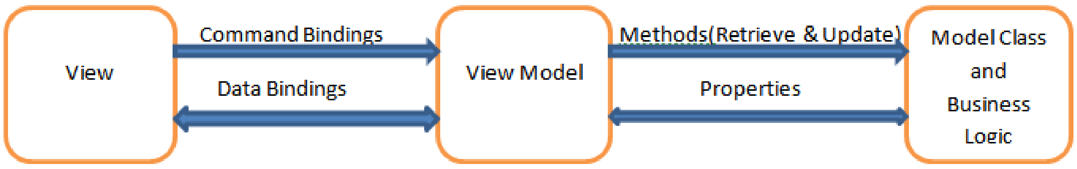


Figure 1 - MVVM Application Design (dotnetforall, 2022)

2.2.1 Model

This layer of the design should contain the business Logic and data models for the application, this layer should have no reference or dependence upon any class in the view or view model layer and should contain no visual code. This is the layer where database access would occur.

2.2.2 View

This layer of the design is the visual layer of the application. The code behind should not contain any logic that is not directly relevant to the view it is attached to for example animations. The view can hold a reference to the View-Model for the particular view via the data context. This will allow the view to bind to the properties of the view model and receive updates when these properties change causing the view to also update.

2.2.3 View-Model

The View-Model is the middle man between the model and the view it should not contain any direct visual logic and should be independent form any visual classes. Its job is to take the data from the business logic or model layer and format it into the appropriate format for a particular view to bind to. This is done by having public getter and setter properties that the view will bind to that implement allowing the view access to the data. As the data is only accessed via these bindings then any view which is made to access those binding can be used making updating the UI an almost completely separate process from the rest of the application.

2.3 The Turing Game

The Turing game originated as a thought experiment devised in part by and named after Alan Turing in response to the question on whether a machine could think, it is also known as the Turing test or the imitation game. An over view of the game is outlined here by Oppy, Graham and Dowe

"Turing (1950) describes the following kind of game. Suppose that we have a person, a machine, and an interrogator. The interrogator is in a room separated from the other person and the machine. The object of the game is for the interrogator to determine which of the other two is the person, and which is the machine." (Oppy, Graham, & Dowe, 2003)

The Question of whether a machine could think was not a concept that was created by Alan Turing. The first known reference to the pursuit of answering this question was Rene Descartes in the early 17th century however this was more as a philosophical pursuit rather than a scientific one. At the time the concept of a computer was still 200 years away with Charles Babbage inventing the first mechanical computer in 1822. Descartes used the term Automata to represent a self-operating machine that could respond to an interaction with a human ultimately Descartes concluded that no machine could ever respond meaningfully to a human with comparable intelligence. Alan Turing was a member of the Ratio Club This was a dining club which existed between 1949 and 1958 and its members consisted of Psychiatrists, Psychologists, Physiologists, Mathematicians and engineers which ponder the questions of cybernetics and machine intelligence. While Alan Turing was a member of this club he devised the original concept of the imitation game. Alan believed the following about his game:

"I believe that in about fifty years' time it will be possible to programme computers, with a storage capacity of about 109, to make them play the imitation game so well that an average interrogator will not have more than 70 percent chance of making the right identification after five minutes of questioning. … I believe that at the end of the century the use of words and general educated opinion will have altered so much that one will be able to speak of machines thinking without expecting to be contradicted."

3 Requirements

This project will use network sockets and an internet connection for communication between the various applications. This section of the document will cover the design decisions that were made and the rationale behind those choices over potential alternatives.

3.1 Product requirements

The software produced because of this project is aimed at providing a teaching aid for teachers who wish to teach their students about AI and the Turing game. The idea behind this is that it would be being used in a computer room at a school or college with multiple machines accessing the same network to allow communication between students with a teacher monitoring the results.

The software is designed to run on a machine running a windows 10 operating system or later.

3.2 Functional requirements

Functional requirements are features of a product that must be implemented by developers in order for an application to allow the end user to achieve their goals. They define the basic behaviour of the system and what conditions the system is in to achieve them.

3.2.1 Interfaces

TGF_REQ_UI_01: Accessibility

- Description: The UI shall provide comprehensive data in a form that is universally acceptable and must meet the accessibility standards such as those outlined in the W3C accessibility Guidelines
- Objective Reference: Objective 1 section 4 and Objective 2 section 4
- Dependencies: N/A

TGF_REQ_UI_02: Usability

- Description: The UI shall provide an intuitive and easy to pick up experience for the user.
- Objective Reference: Objective 1 section 4 and Objective 2 section 4
- Dependencies: N/A

3.2.2 Functional Capabilities

The functional requirements of the solution are broken down into individual functional requirements for each Application that is produced as part of the solutions these are as outlined below

3.2.2.1 Client Application

TGF_REQ_CA_01: Connection

- Description: The Solution shall provide the capability to connect another application on the same network.
- Objective Reference: Objective 1
- Dependencies: N/A

TGF_REQ_CA_02: Send Message

- Description: The Solution shall provide the capability to send data to a connected device across a network.
- Objective Reference: Objective 1
- Dependencies: N/A

TGF_REQ_CA_03: Receive Messages

- Description: The Solution shall provide the capability to receive data from a connected device across a network.
- Objective Reference: Objective 1
- Dependencies: N/A

TGF_REQ_CA_04: Submit Partner

- Description: The Solution shall provide the capability for the user to submit their choice as to what they are speaking to.
- Objective Reference: Objective 1
- Dependencies: N/A

TGF_REQ_CA_05: User Input

- Description: The Solution shall provide the capability to handle user input.
- Objective Reference: Objective 1
- Dependencies: N/A

3.2.2.2 Controller/Server Application

TGF_REQ_SA_01: Connection

- Description: The Solution shall provide the capability to connect to another device on the network.
- Objective Reference: Objective 2
- Dependencies: N/A

TGF_REQ_SA_02: Send Message

- Description: The Solution shall provide the capability to send data to a connected device across a network.
- Objective Reference: Objective 1
- Dependencies: N/A

TGF_REQ_SA_03: Receive Messages

- Description: The Solution shall provide the capability to receive data from a connected device across a network.
- Objective Reference: Objective 1
- Dependencies: N/A

TGF_REQ_SA_04: Partnering Clients

- Description: The Solution shall provide the capability to partner up connected clients to enable communications between them.
- Objective Reference: Objective 1
- Dependencies: N/A

TGF_REQ_SA_05: Launching the Chatbot

- Description: The Solution shall provide the capability to Instantiate a new instance of the chatbot Wrapper Application.
- Objective Reference: Objective 1
- Dependencies: N/A

TGF_REQ_SA_0: Creating new rooms as required

- Description: The Solution shall provide the capability to Instantiate a new Room with connections when a new connection attempt is made without a currently available room
- Objective Reference: Objective 1
- Dependencies: N/A

3.2.2.3 Chatbot Wrapper Application

TGF_REQ_CB_01: Connection

- Description: The Solution shall provide the capability to send messages to a connected device across a network.
- Objective Reference: Objective 3
- Dependencies: N/A

TGF_REQ_CB_02: Send Message

- Description: The Solution shall provide the capability to send data to a connected device across a network.
- Objective Reference: Objective 3
- Dependencies: N/A

TGF_REQ_CB_03: Receive Messages

- Description: The Solution shall provide the capability to receive data from a connected device across a network.
- Objective Reference: Objective 3
- Dependencies: N/A

TGF_REQ_CB_04: Host chatbot

- Description: The Solution shall provide the capability to launch a new instance of a provided chatbot.
- Objective Reference: Objective 3
- Dependencies: N/A

TGF_REQ_CB_04: Interact with chatbot

- Description: The Solution shall provide the capability to relay received messages to the chatbot and relay the responses back the connected device.
- Objective Reference: Objective 3
- Dependencies: N/A

3.2.3 Performance Requirements

TGF_REQ_PERF_01: Response Time

- Description: The response time between applications should be less than 2 seconds. This is defined as the time it takes for the user interface to acknowledge interaction from the user.
- Objective Reference: N/A
- Dependencies: N/A

3.2.4 Reliability

TGF_REQ_REL_01: Stability

- Description: The applications should be able to remain stable and connected for 24 hours without disconnecting or loosing any data sent between them.
- Objective Reference: N/A
- Dependencies: N/A

TGF_REQ_REL_02: User Disconnect

- Description: The server application should be able to handle if a user disconnects unexpectedly without crashing.
- Objective Reference: N/A
- Dependencies: N/A

TGF_REQ_REL_03: Server Disconnect

- Description: The client application should be able to handle if the server disconnects unexpectedly without crashing.
- Objective Reference: N/A
- Dependencies: N/A

3.2.5 Security/Privacy

No personal data is stored as part of the solution and users are kept completely anonymous unless the user chooses to reveal any data about themselves, this data however is simply passed on to the end point and is not stored in any form upon cessation of the session.

3.2.6 Quality

TGF_REQ_QUAL_01: User Testing

- Description: The system will be tested by several users and feedback given in the form of the feedback sheet provided as part of the Project initiation document in order to establish if the solution is of the expected quality.
- Objective Reference: N/A
- Dependencies: N/A

4 Design

4.1 Software design

This section of the report will cover the design of each of the three applications that make up the Turing Game proof of Concept the UML Diagrams are fairly large and have been scaled down however Full-Size Diagrams can be found in the Appendices of this document and will be referenced in their relevant section Below is a package diagram showing the relationships between the applications. The socket handler class in each application exposes the application to each other this is hosted by the Room class in the Controller as represented in the diagram below. Full-size versions of the Class diagrams in this section can be Found in Appendix B

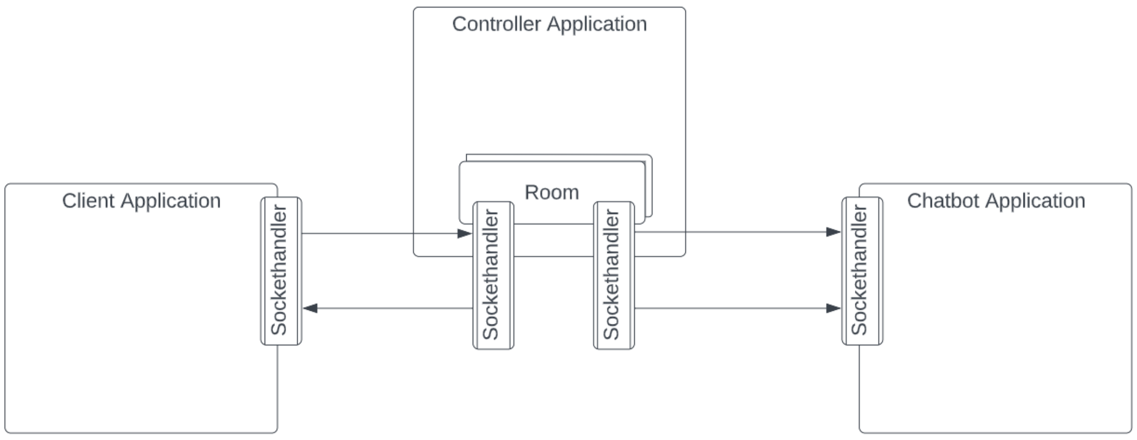


Figure 2 - TGF package Diagram

4.1.1 Client application Design

This section covers the design of the Client application, this section will include a UML (Diagram see Appendix B for the Full-size Diagram), A Breakdown of each of the Classes and their functionality as well as several sequence diagrams to demonstrate the concept of their operation.

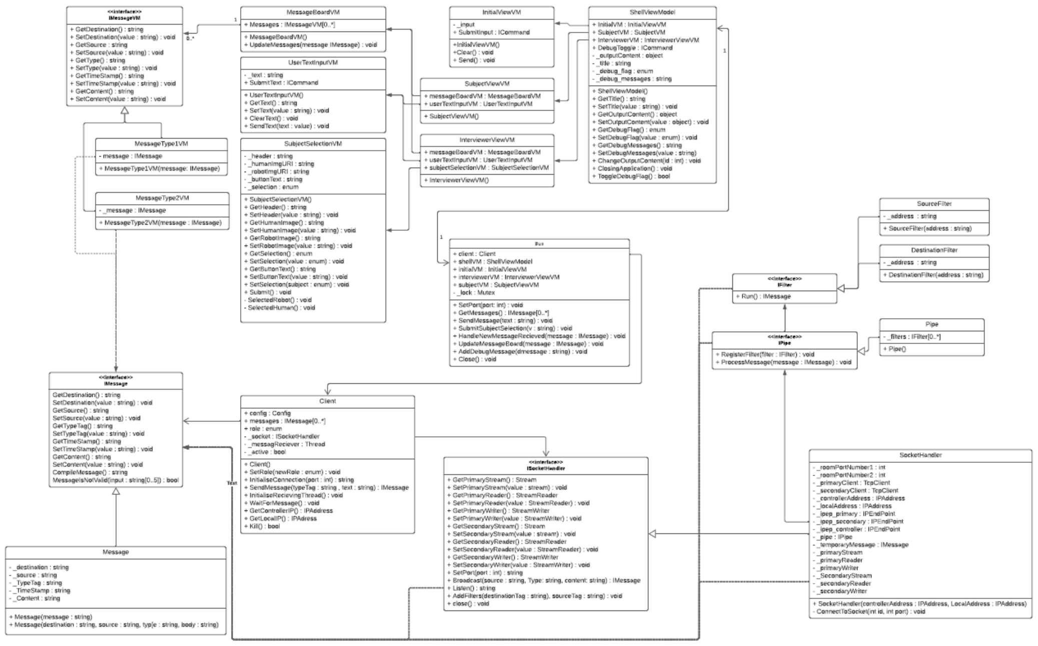


Figure 3 - TGF_Client Class Diagram

The main method of abstraction between the business logic and the View models is The Bus class which is a static class that is used to abstract the dependencies between the UI and the back end of the application. The front end is managed by a series of view model classes that have bind able properties which can be seen by the View and update the view when they are changed. The Business logic is built based upon an attempt at an Object-Oriented Approach. A breakdown of the back end is as follows.

4.1.1.1.1 Bus

This is a static class that has been designed for use as a data interface to allow the View models to interact with the business logic without becoming Tightly Bound to the solution and vice Versa.

Variables:

client – This is a public variable used to store the current instance of the Client Class.

shellVM – This is a public variable used to store the current instance of the ShellViewModel Class.

initialVM – This is a public variable used to store the current instance of the IntitalViewVM Class.

interviewerVM – This is a public variable used to store the current instance of the InterviewerViewVM Class, it is unused when the client is set to Subject Mode.

subjectVM – This is a public variable used to Store The current Instance of the SubjectViewVM Class, it is unused when the Client is set to Interviewer Mode.

_lock – This is a private Mutex variable that is used in order to promote thread safe data write access for the message board and other aspects of the solution with a multithreading component.

Functions:

SetPort(port : int) – This Function will be used to initiate the Digital handshake between the Server and the Client. The Parameter It takes is the port number of the Controller Host Port. it sets the next state of the Client application.

GetMessages() – This Function is used to access the current List of messages in an abstracted fashion It returns the Current List of IMessage's from the session.

SendMessage(text : string) – This Function is used to abstract the Functionality of sending the User input message to the server. It takes the user inputted message as the string parameter. It Calls the Client SendMessageFunction() passing the passed in string as a parameter.

SubmitSubjectSelection(v : string) – This Function is used to Send the Interviewers Selected Guess at who they are speaking to It has very similar functionality to the SendMessage Function however it filters the Message to have a specific Tag Type, It calls the Client SendMessage Function.

HandleNewMessageRecieved(message : IMessage) – This Function is used to abstract the Adding of a new Message to the Views message board. It Does this by taking in a IMessage derived Data Structure and calls the Update Message board function using the _lock to control thread access and prevent concurrency issues.

UpdateMessageBoard(message:IMessage) – This Function is used to update the MessageBoard for the Active session it takes an IMessage Derived Data Structure as a parameter and adds it to the active view models Message board.

AddDebugMessage(dmessage : string) - This function is used as part of the Debug/testing and adds messages with time stamps to the Debug window of the application.

Close() – This Function handles the Close events and triggers the deconstructors or close operations of the child classes of the current Class.

4.1.1.1.2 Client

This Class is used to host the Back-end functionality and Business Logic of the Client application

Variables:

config – This is a public variable that is used to store Config data as a **Dictionary<string,string>**.

messages – This is a public List IMessage Derived Classes.

role – this is an enum that is used to store the current role of the Client application (Interviewer or Subject).

_socket – This is a private variable that is used to store an instance of an **ISocketHandler** derived Class.

_messageReciever – This is a private variable used to store the instance of the thread that will be listening for messages.

_active – This is a private Boolean that is used to control the action loops within the Client code when false all code loops should fall out and cease operation.

Functions:

Client() – This is the Constructor for the Client Class it initialises the variables and sets up the environment for the **_messageReciever** Thread.

SetRole(newRole : enum) – This Function takes an enum as a parameter and is used to set the role variable of the Client Class.

InitialiseConnection(port : int) – This Function takes the controller port number as an int as a parameter and uses it to initiate connecting to the controller by calling the **SetPort()** function on the **_socket** variable. It Returns a string that is received from the controller after the Handshake is complete.

SendMessage(typeTag : string, text : string) – This function is used to send The clients messages to the server via the **_socket** variable it returns an **IMessage** derived class instance of the message that was sent.

InitialiseRecievingThread() – This Function initialises and starts the **_messageReciever** Thread.

WaitForMessage() – This Function is the task that is targeted by the **_messageReciever** Thread. It imply loops whilst checking the **_socket** variable for a new message, when a new message is found it calls the **HandleNewMessageRecieved()** Function in the Bus Class passing in the received message

GetControllerIP() – This Function returns the Currently stored IP address for the Controller from the **_config** variable.

GetLocalIP() – This function Finds the IP address of the Current Computer/ device that the Client application is running on.

Kill() – This Function is used to Terminate any active threads and break them out of any loops on application close.

4.1.1.1.3 Message

This class is used to parse and store the message data that the client both sends and receives. It inherits from the **IMessage** Interface.

Variables:

_destination – This is a private variable used to store the destination data of the current **Message** instance

_source – This is a private variable used to store the source data of the current **Message** instance

_typeTag – This is a private variable used to store the type data of the current **Message** instance

_timeStamp – This is a private variable used to store the time of creation for the current **Message** instance

_content – This is a private variable used to store the actual contents of the current **Message** instance

Functions:

Message(message : string) - This is a constructor for the Message Class that takes a single string with all of the message data separated by commas for a parameter. This is then split up and assigned to the appropriate variables.

Message(destination : string, source : string, type : string, body : string) – This is another constructor that takes in all of the individual values as separate string and assigns them to the corresponding variables generating a new time stamp for the **_timeStamp** variable.

4.1.1.1.4 Pipe

This class inherits from the **IPipe** interface. It is used to apply filters to messages before sending them and upon receiving them.

Variables:

_filters – This is a private variable that is used to store a list of the various filters which are to be applied to messages.

Functions:

Pipe() – This is a constructor that instantiates the Pipe class.

4.1.1.1.5 SourceFilter

This is a class that derives from the **IFilter** Interface. It is used to check the source address value of the message and correct it if necessary.

Variables:

_address – This is a private function that stores the address value to be applied to any messages which pass through the Filter.

Functions:

SourceFilter(address : string) – This is the constructor for the **SourceFilter** class.

4.1.1.1.6 DestinationFilter

This is a class that derives from the **IFilter** Interface. It is used to check the destination address value of the message and correct it if necessary.

Variables:

_address – This is a private function that stores the address value to be applied to any messages which pass through the Filter.

Functions:

DestinationFilter(address : string) – This is the constructor for the **DestinationFilter** class.

4.1.1.1.7 SocketHandler

This class inherits from the **ISocketHandler** Interface. This class Handles the logic and functionality for connecting to the Controller using network Sockets. It uses two client one for writing and one for reading in order to maintain complete asynchronous communication.

Variables:

_roomPortNumber1 – This is a private variable that stores the port number for the **_primaryClient** to connect to.

_roomPortNumber2 – This is a private variable that stores the port number for the **_secondaryClient** to connect to.

_primaryClient – This is a private variable that is used to store an instance of a **TCPClient** that will be used to send messages to the Controller.

_secondaryClient – This is a private variable that is used to store an instance of a **TCPClient** that will be used to Receive messages from the Controller.

_controllerAddress – This is a private variable that is used to store the IP Address of the Controller application.

_localAddress – This is a private variable that is used to store the IP Address of the Current computer or device that is running the client application.

_ipep_primary – This is a private variable that is used to store the primary IPEndpoint assigned to this client by the Controller. It is made up of the **_controllerAddress** and the **_roomPortNumber1**.

_ipep_secondary – This is a private variable that is used to store the secondary IPEndpoint assigned to this client by the Controller. It is made up of the **_controllerAddress** and the **_roomPortNumber2**.

_ipep_controller – This is a private variable that is used to store the IPEndpoint of the Controllers Main Port. It is made up of the **_controllerAddress** and the Controllers port number entered by the user.

_pipe – This is a private variable that is used to store an instance of an **IPipe** derived Class.

_primaryStream – This is a private variable used to store the instance of the DataStream that is used by the **_primaryClient.**

_primaryReader – this is a private variable used to store an instance of a stream reader that is used to read data from the **_primaryStream**.

_primaryWriter – this is a private variable used to store an instance of a stream writer that is used to write data to the **_primaryStream**.

_secondaryStream – This is a private variable used to store the instance of the DataStream that is used by the **_secondaryClient.**

_secondaryReader – this is a private variable used to store an instance of a stream reader that is used to read data from the **_secondaryStream**.

_secondaryWriter – this is a private variable used to store an instance of a stream writer that is used to write data to the **_secondaryStream**.

Functions:

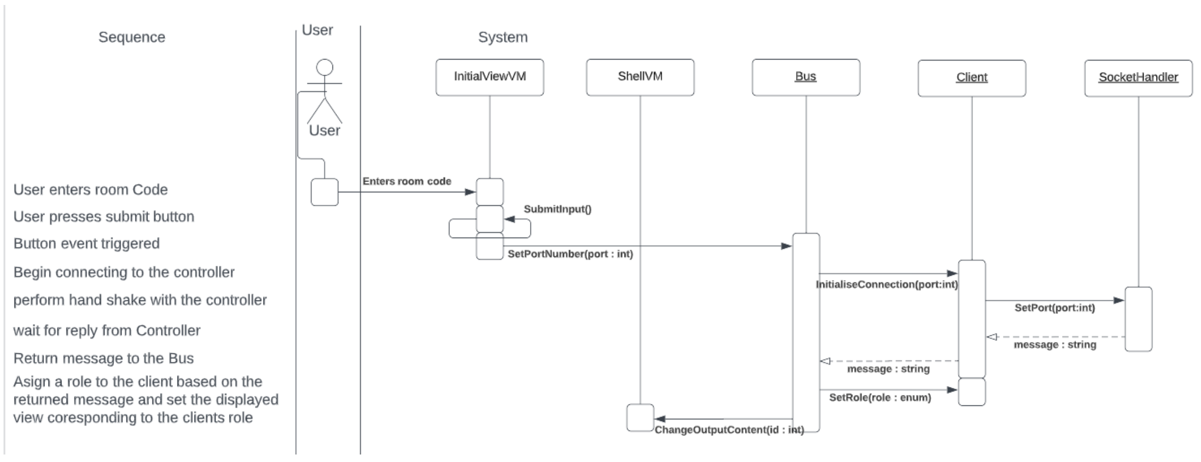
SocketHandler(controllerAddress : IPAddress, LocalAddress : IPAddress) – This is a Constructor for the class taking the IPAddress of the current machine and the controller as parameters.

ConnectToSocket(id : int, port : int) – This Function is Used to connect the primary and secondary clients to their respective ports the id parameter identifies which client variable is being connected and the port is the port number for that client to connect to.

4.1.2 Client Sequence Diagrams

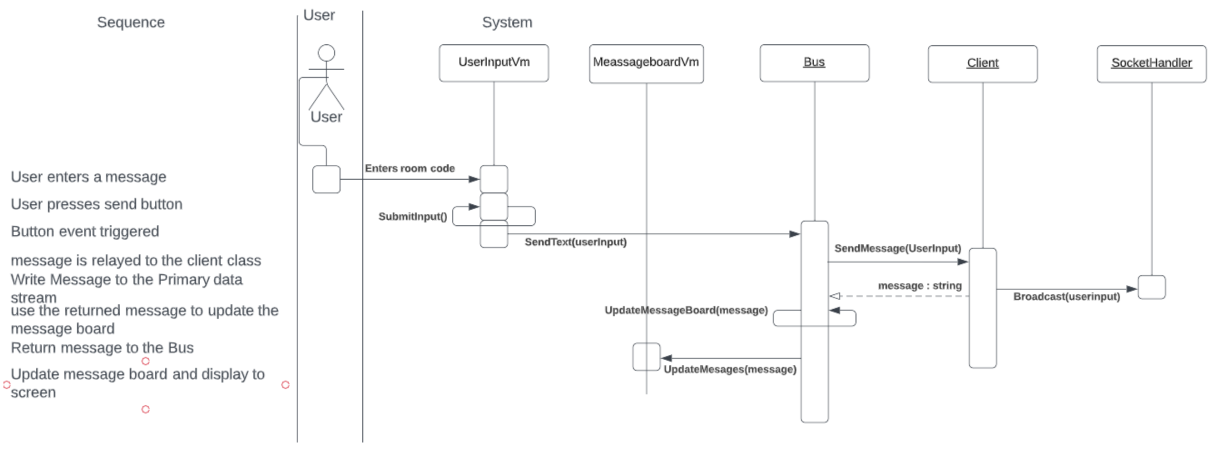
4.1.2.1 Client enters Session code

Below is a sequence diagram showing the sequence of events that occur when a user enters the session code into the Client. The image does not show the network communication stage only up until the communication is initiated.


Sequence
User
System
InitialViewVM
ShellVM
Bus
Client
SocketHandler
User
User enters room Code
User presses submit button
Button event triggered
Begin connecting to the controller
perform hand shake with the controller
wait for reply from Controller
Return message to the Bus
Asign a role to the client based on the returned message and set the displayed view coresponding to the clients role
Enters room code
SubmitInput()
SetPortNumber(port : int)
InitialiseConnection(port:int)
SetPort(port:int)
message : string
message : string
SetRole(role : enum)
ChangeOutputContent(id : int)

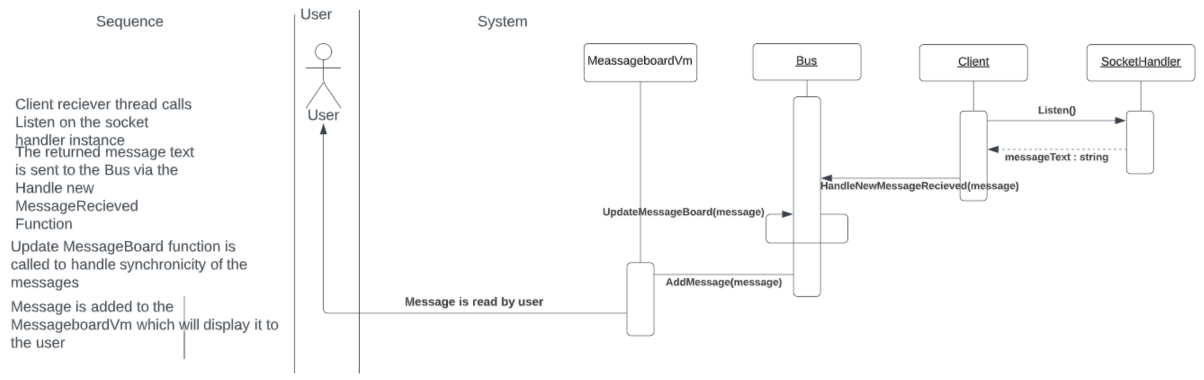
4.1.2.2 Client sends message

The below Diagram shows the sequence events and function calls required to send a user input message to the server. The image does not show the network communication stage only up until the communication is initiated.



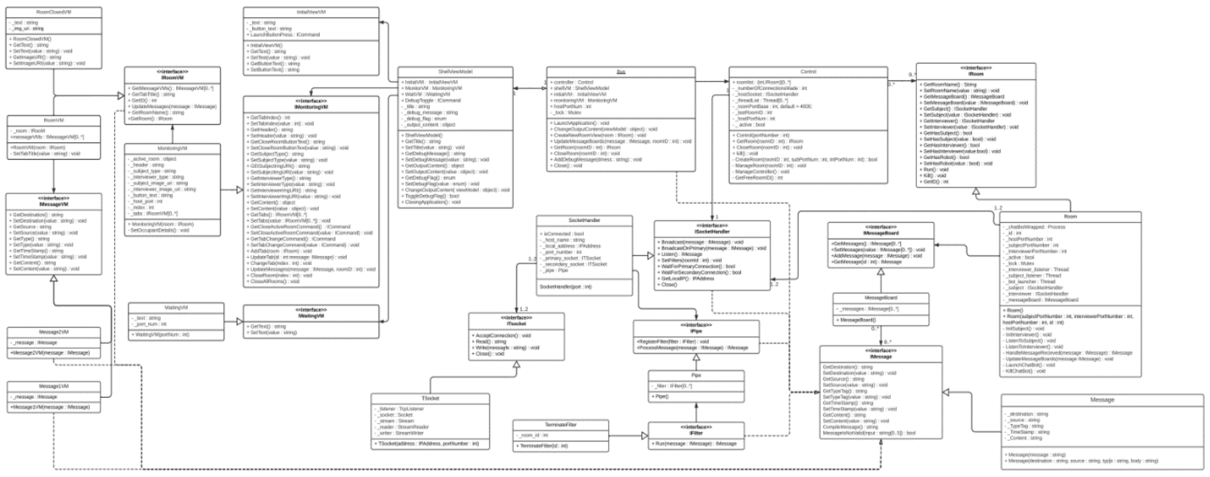
4.1.2.3 Client Receives message

The below Diagram shows the sequence events and function calls required to display a new message when a message is received from the server. The image does not show the network communication stage only up until the communication is initiated.


Sequence
User
System
MeassageboardVm
Bus
Client
SocketHandler
User
Client reciever thread calls Listen on the socket handler instance
The returned message text is sent to the Bus via the Handle new MessageRecieved Function
Update MessageBoard function is called to handle synchronicity of the messages
Message is added to the MessageboardVm which will display it to the user
Listen()
messageText : string
HandleNewMessageRecieved(message)
UpdateMessageBoard(message)
AddMessage(message)
Message is read by user

4.1.3 Controller application Design

This section covers the design of the Controller application, this section will include a UML (Diagram see Appendix B for the Full-size Diagram), A Breakdown of each of the Classes and their functionality as well as sequence diagrams to demonstrate the concepts of operation.



The main method of abstraction between the business logic and the View models is The Bus class Once again which is a static class that is used to abstract the dependencies between the UI and the back end of the application. The front end is managed by a series of view model classes that have bind able properties and are extended by interfaces to allow for mocking for testing purposes and easy change outs. This can be seen by the View and updates the view when the properties are changed. The Business logic is built based upon an attempt at an Object-Oriented Approach. A breakdown of the back end is as follows.

4.1.3.1.1 Bus

This is a static class that has been designed for use as a data interface to allow the View models to interact with the business logic without becoming Tightly Bound to the solution and vice Versa.

Variables:

controller – This is a public variable used to store the current instance of the Controller Class.

shellVM – This is a public variable used to store the current instance of the ShellViewModel Class.

initialVM – This is a public variable used to store the current instance of the IntitalViewVM Class.

monitoringVM – This is a public variable used to store the current instance of the MonitoringVM Class

hostPortNum – This is a public integer that is the Public Port number for the Controller application.

_lock – This is a private Mutex variable that is used in order to promote thread safe data write access for the message board and other aspects of the solution with a multithreaded component.

Functions:

LaunchApplication() – This Function Is used to instantiate the various components of the Controller application and open the Public port to receive connections.

ChangeOutputcontent(viewModel:object) – This function is used to change out the current active view it takes a view model as a parameter represented by an object.

CreateNewRoomView(room:IRoom) – this Function creates a new RoomVM class based on the supplied IRoom derived data structure.

UpdateMessageBoards (message: IMessage, roomID: int) – This Function is used to update the Messageboard for the corresponding room it takes an IMessage Derived Data Structure as a parameter and adds it to Messageboard of the Supplied RoomId.

GetRoom(roomID : int) – This function returns the instance of the Room Class with the provided roomID

CloseRoom(roomID : int) – This function is used to call the Close command on the room with the corresponding roomID.

AddDebugMessage(dmessage : string) - This function is used as part of the Debug/testing and adds messages with time stamps to the Debug window of the application.

Close() – This Function handles the Close events and triggers the deconstructors/close operations on the child classes of the current Class.

4.1.3.1.2 Control

This Class is used to host the Back-end functionality and Business Logic of the Controller application

Variables:

roomList – This is a public dictionary of room objects keyed against an int which is used as the room ID

_numberOfConnectionsMade – This is a private variable that tracks the current number of connected clients

_hostSocket – This is a private variable which stores the Current instance of the host Socket handler class

_threadList – This is a private list of the active threads that are managing Rooms.

_roomPortBase – This is a private function that stores the current iteration of portnumber that the next room will use as a base the default value is 4000.

_lastRoomID – This is a private variable that stores the last id given to a room

_hostPortNum – This is a private variable to store the port number used by the **_hostSocket** variable.

_active – This is a private bool that is used to control the multithreaded elements.

Functions:

Control() – This is the Constructor for the Control Class it initialises the variables and sets up the environment for running the application

GetRoom(roomID : int) – This Function Returns the instance of the Room Class that is associated with the supplied roomID parameter.

CloseRoom(roomID : int – This Function calls the Close function on the Instance of the Room class that is associated with the supplied roomID.

Kill() – This Function is used to Terminate any active threads and break them out of any loops on application close.

CreateRoom(roomID : int, subPortNum : int, intPortNum : int) – This is a private Function that is used to create a new instance of the Room Class by supplying the included parameters in the Room Constructor.

ManageRoom(roomID : int) – This is a private Function that is used as the main Function for one of the Room Managing Threads it loops through the functionality of the room as long as the **_active** variable is true.

4.1.3.1.3 Room

This class hosts the two sets of connections for the Client applications to connect to and handles relaying messages between them and storing the communication for the current session.

Variables:

_chatBotWrapped – This is a private function that is used to store an instance of the Process that will host the Chatbot Wrapper application.

_id – This is a private function that stores the roomID

_hostPortNumber – This is a private variable that stores the Public Port number of the Controller Application.

_subjectPortNumber – This is a private variable that stores the private port number for the subject Client to connect to

_InterviewerPortNumber – This is a private variable that stores the private port number that the Interviewer Client will connect to.

_active – This is a private bool that is used to control the multithreaded elements.

_lock – This is a private Mutex variable that is used in order to promote thread safe data write access for the message board and other aspects of the solution with a multithreading component.

_interviewer_listener – This is a private Thread that will asynchronously Listen to the interviewer's primary data stream for new messages and add them to the **_messageBoard** as they arrive

_subject_listener – This is a private Thread that will asynchronously Listen to the Subject's primary data stream for new messages and add them to the **_messageBoard** as they arrive

_bot_launcher – This is a private Thread that will asynchronously launch the chat bot when required.

_subject – this is a private variable that will store the instance of the Sockethandler Class that the Subject Will Connect to.

_interviewer – this is a private variable that will store the instance of the Sockethandler Class that the Interviewer Will Connect to.

_messageBoard – This is a private variable that will store the current instance of the MessageBoard Class.

Functions:

Generic Getter and Setter Functions for the Private Variables.

Room() – This is a parameter less constructor for the Room Class

Room(subjectPortNumber : int, interviewerPortNumber : int, hostPortNumber : int, id : int) – This is a Constructor for the Room Class that takes the room id and relevant Port Numbers as parameters.

InitSubject() – This function initialises the Sockethandler Class that is being used to allow the Subject to connect.

InitInterviewer() – This function initialises the Sockethandler Class that is being used to allow the Interviewer to connect.

ListenToSubject() – This function is used as the Main function for the **_subject_listener** Thread it loops whilst checking for new messages on the Subject Sockethandler Class

ListenToInterviewer() () – This function is used as the Main function for the **_Interviewer_listener** Thread it loops whilst checking for new messages on the Interviewer Sockethandler Class

HandleMessageRecieved(message : IMessage) – This message is used to add a new message to the messageBoard class utilising the _lock to ensure there is no race condition.

UpdateMessageBoards (message: IMessage) – This function is used to notify the view model that the MessageBoard has new data and it needs to be updated.

LaunchChatBot () – This Function is used as the Main for the **_bot_launcher** Thread it sets up the Chatbot process and supplies the Command line arguments for it to be launched.

KillChatBot () – This function sends the command to close the Chat bot process.

4.1.3.1.4 SocketHandler

This class Handles the two socket connections used to connect to a client Application. As well as hosting functionality to send and receive data on the Created Data streams.

Variables:

isConnected – This is a public Boolean variable that reflects whether the Sockethandler is connected or not.

_host_name – This is a private variable that stores the name of the host machine.

_local_address – This is a private variable that stores the Ip Address of the local machine

_port_number – This is a private variable that stores the base port number for the two ports the Client will connect to.

_primary_socket – This is a private variable that stores an instance of the TSocket Class that represents the primary connection to the Client.

_secondary_socket – This is a private variable that stores an instance of the TSocket Class that represents the secondary connection to the Client.

Functions:

SocketHandler (port: int) – This is a constructor for the Socket handler Class which takes the base port number as the parameter.

BroadCastMessage (message: Message) – This function calls the **Write ()** function on the **_secondary_socket** variable passing the compiled message as a parameter.

Listen () – This function calls the **Read ()** function on the **_primary_socket** variable.

WaitForPrimaryConnection () – This Function will wait until the primary connection is established.

waitForSecondaryConnection () – This Function will wait until the secondary connection is established.

GetLocalIP () – This function returns the Local IP Address of the Host machine.

Close () – This function calls the Close function on the owned socket classes.

4.1.3.1.5 TSocket

This class extends the functionality of the TCPSocket class to incorporate the Data stream controls.

Variables:

- _listener – This is a private variable that stores the current instance of a TcpListener class.

_socket – This is a private variable that stores the current Socket Class instance.

_stream – This is a private variable that stores the current instance of the DataStream.

_reader – This is a private variable that stores the current instance of the Stream Reader.

_writer – This is a private variable that stores the current instance of the Stream Writer.

Functions:

TSocket (address: IPAddress, portNumber: int) – This is a constructor for the TSocket Class which takes the base port number and the local IP address as parameters.

AcceptConnection () – This Function is used to accept the current connection attempt made to the TCPListener

Read () – This Function is used to read any messages on the data stream.

Write (message: string) – This Function is used to write the passed in parameter to the Data Stream.

Close () – This function is used to terminate the active connection and close the DataStream.

4.1.3.1.6 MessageBoard

This class is used to store messages that are being sent through the room.

Variables:

- _messages – This is a private list that stores instances of the **Message** Class.

Functions:

MessageBoard () – This is a parameter less constructor which instantiates the MessageBoard class.

GetMessages () – This is a Getter function that returns the value of _**messages**.

SetMessages (value: IMessage [0 ..*]) – This is a setter function that sets the value of the **_messages** variable.

AddMessage (message: IMessage) – This Function is used to add a new message to the _**messages** variable.

GetMessage (id: int) – This function is used to retrieve a specified message from the _**messages** variable.

4.1.3.1.7 Message

This class is used to parse and store the message data that the controller both sends and receives.

Variables:

_destination – This is a private variable used to store the destination data of the current **Message** instance

_source – This is a private variable used to store the source data of the current **Message** instance

_typeTag – This is a private variable used to store the type data of the current **Message** instance

_timeStamp – This is a private variable used to store the time of creation for the current **Message** instance

_content – This is a private variable used to store the actual contents of the current **Message** instance

Functions:

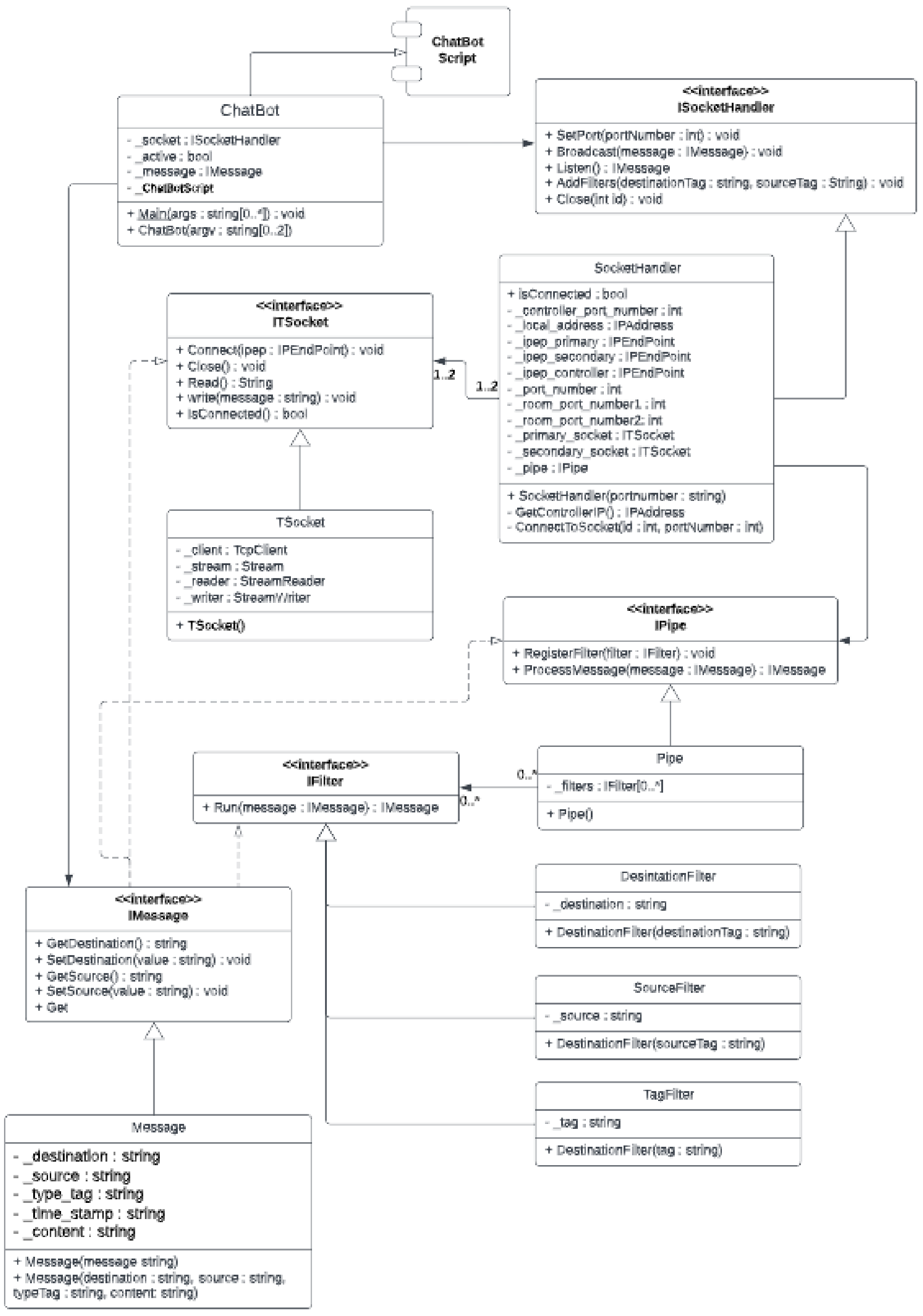
Generic Getter and Setter Functions To access the private Properties.

Message(message : string) - This is a constructor for the Message Class that takes a single string with all of the message data separated by commas for a parameter. This is then split up and assigned to thew appropriate variables.

Message(destination : string, source : string, type : string, body : string) – This is another constructor that takes in all of the individual values as separate string and assigns them to the corresponding variables generating a new time stamp for the **_timeStamp** variable.

4.1.4 Chatbot_Wrapper application Design

The chat bot Wrapper is designed in order to utilise the same connection functionality as the Client and the Controller but the messages are supplied by a chatbot executable instead of a human. The chat bot used is an opensource project created by (solo-rey, 2018) and is used to demonstrate the proof of concept for the Wrapper application and how it can be used to warp different chatbots. This is to prove the concept of supplying different chatbots depending upon the purpose. A breakdown of the classes that make up the Chatbot wrapper can be found below.



ChatBot Script
ChatBot
- _socket : ISocketHandler
- _active : bool
- _message : IMessage
- _ChatBotScript
+ Main(args : string[0..*]) : void
+ ChatBot(argv : string[0..2])
<<interface>> ISocketHandler
+ SetPort(portNumber : int) : void
+ Broadcast(message : IMessage) : void
+ Listen() : IMessage
+ AddFilters(destinationTag : string, sourceTag : String) : void
+ Close(int id) : void
SocketHandler
+ isConnected : bool
- _controller_port_number : int
- _local_address : IPAddress
- _ipep_primary : IPEndPoint
- _ipep_secondary : IPEndPoint
- _ipep_controller : IPEndPoint
- _port_number : int
- _room_port_number1 : int
- _room_port_number2: int
- _primary_socket : ITSocket
- _secondary_socket : ITSocket
- _pipe : IPipe
+ SocketHandler(portnumber : string)
- GetControllerIP() : IPAddress
- ConnectToSocket(id : int, portNumber : int)
<<interface>> ITSocket
+ Connect(ipep : IPEndPoint) : void
+ Close() : void
+ Read() : String
+ write(message : string) : void
+ IsConnected() : bool
1..2
1..2
TSocket
- _client : TcpClient
- _stream : Stream
- _reader : StreamReader
- _writer : StreamWriter
+ TSocket()
<<interface>> IPipe
+ RegisterFilter(filter : IFilter) : void
+ ProcessMessage(message : IMessage) : IMessage
<<interface>> IFilter
+ Run(message : IMessage) : IMessage
0..*
0..*
Pipe
- _filters : IFilter[0..*]
+ Pipe()
<<interface>> IMessage
+ GetDestination() : string
+ SetDestination(value : string) : void
+ GetSource() : string
+ SetSource(value : string) : void
+ Get
DesintationFilter
- _destination : string
+ DestinationFilter(destinationTag : string)
SourceFilter
- _source : string
+ DestinationFilter(sourceTag : string)
TagFilter
- _tag : string
+ DestinationFilter(tag : string)
Message
- _destination : string
- _source : string
- _type_tag : string
- _time_stamp : string
- _content : string
+ Message(message string)
+ Message(destination : string, source : string, typeTag : string, content: string)

Figure 4 - Chatbot Wrapper Class Diagram

4.1.4.1.1 ChatBot

This is the main class that hosts the Chatbot_Wrapper applications functionality

Variables:

_socket – This is a Private variable that is used to store the current instance of the SocketHandler Class.

_active – This is a private variable that is used to store the Boolean flag for the thread loops in code.

_message – This is a private variable that stores the chatbot response as a message data structure.

_ChatBotScript – This is a private variable that stores the relative file path to the Chatbot executable.

Functions:

Main (args: string [0..*]) – This is the main function for the programme it launches the Constructor for the chatbot class and takes in the command line arguments supplied on launch of the application.

ChatBot (argv: string [1]) – This is the Constructor for the ChatBot class it takes a one element array of strings as an argument which contains the public port number of the Controller application it uses this to instantiate the _socket variable and also creates and hosts the process to launch the chatbot executable. It then overrides the input and output streams in order to pass received messages to the chatbot and read the chatbot responses from it. Once it has set up these variables it initiates the connection to the host and then loops through the send and receive methods until it is closed.

4.1.4.1.2 ChatBot

This class is responsible for handling the network socket connections for the application.

Variables:

isConnected – This is a public variable that stores the Boolean state of the connection, true for connected and false for not connected.

_localAddress – This is a private variable that is used to store the IP Address of the Current computer or device that is running the Chatbot application.

_ipep_primary – This is a private variable that is used to store the primary IPEndpoint assigned by the Controller.

_ipep_secondary – This is a private variable that is used to store the secondary IPEndpoint assigned by the Controller.

_ipep_controller – This is a private variable that is used to store the IPEndpoint of the Controllers Main Port. It is made up of the **_localAddress** and the Controllers port number passed into the constructor.

_port_number – This is a private variable used to store the public port number of the controller that is passed in to the Constructor.

_room_port_number1 – This is a private variable that is used to store the primary port number provided by the controller.

_room_port_number2 – This is a private variable that is used to store the secondary port number provided by the controller.

_primary_socket – This is a private variable used to store an instance of the **TSocket** Class that is used to Send messages to the server.

_secondary_socket – This is a private variable used to store an instance of the **TSocket** Class that is used to Receive messages from the server.

_pipe – This is a private variable that is used to store an instance of an **IPipe** derived Class.

Functions:

SocketHandler(portnumber : string) – This is the Constructor for the SocketHandler Class it takes the public portnumber of the Controller application and instantiates the **_localAddress, _pipe, _portNumber, _ipep_controller, _primary_socket** and **_secondary_socket** variables

GetControllerIP() – This function returns the IPAddress of the current device. As the Chatbot_Wrapper application is launched on the same device as the Controller application.

ConnectToSocket(id : int, portNumber : int) – This Function is Used to connect the primary and secondary clients to their respective ports the id parameter identifies which connection is being established and the port is the port number for that connection to connect to.

SetPort(portNumber : int) – This function initiates the communication between the current application and the controller initiating the Handshake. This is done by attempting to connect to the Controllers public Port once done it will receive a message with two new port number in the body. These are the primary and secondary port number of the room this entity has been placed in. The socket handler will now connect the primary and secondary sockets to the new port numbers and the handshake is complete.

Broadcast(message : IMessage) – This function calls the **Write()** function on the **_primary_socket** variable passing in the content of the message.

Listen() – This function calls the **Read()** function on the **_secondary_socket**. And returns the obtained string as a Message data structure.

AddFilters(destinationTag : string, sourceTag : String) – This instantiates new Filters for the Pipe to run and adds them to the **_pipe** variable.

Close(int id) – This function is used to deconstruct the **TSocket** Classes.

4.1.4.1.3 TSocket

This class extends the functionality of the TCPSocket class to incorporate the Data stream controls.

Variables:

_client – This is a private variable that stores the current instance of a TcpClient class.

_stream – This is a private variable that stores the current instance of the DataStream.

_reader – This is a private variable that stores the current instance of the Stream Reader.

_writer – This is a private variable that stores the current instance of the Stream Writer.

Functions:

TSocket () – This is a parameter less constructor to instantiate the TSocket Class.

Connect (ipep: IPEndpoint) – This Function attempts to make a connection to the supplied IPEndpoint.

Read () – This Function is used to read and return any messages on the data stream.

Write (message: string) – This Function is used to write the passed in parameter to the Data Stream.

Close () – This function is used to terminate the active connection and close the DataStream.

IsConnected () – This function returns a Boolean value that gives the current state of the connection with True meaning the Client is connected and False meaning the socket is not connected.

4.1.4.1.4 Pipe

This class allows for the application of filters to the messages that are sent and received by this application

Variables:

_filters – This is a private variable that stores the list of Filters to be applied to any messages passed through the pipe.

Functions:

Pipe () – This is a parameter less constructor for the Pipe Class that instantiates the **_filters** variable

RegisterFilter (filter: IFilter) – This Function is used to add a new Filter to the **_filters** variable

ProcessMessage (message: IMessage) – This function is used to run the passed in message through each of the filters stored in the **_filters** variable and then returns the resulting message.

4.1.4.1.5 DestinationFilter

This Class is used to filter the message and apply certain values to certain message fields that are passed into it.

Variables:

_destination – This is a private variable used to store the destination value to be applied by this filter.

Functions:

DestinationFilter (destinationTag: string) – This is a Constructor for the **DestinationFilter** Class that takes the destination value to be applied as a parameter.

Run (message : Message) – This Function takes the passed in message and applies the **_destination** value to it then returns the modified message.

4.1.4.1.6 SourceFilter

This Class is used to filter the message and apply certain values to certain message fields that are passed into it.

Variables:

_source – This is a private variable used to store the source value to be applied by this filter.

Functions:

SourceFilter (destinationTag: string) – This is a Constructor for the **SourceFilter** Class that takes the destination value to be applied as a parameter.

Run (message: Message) – This Function takes the passed in message and applies the **_source** value to it then returns the modified message.

4.1.4.1.7 TagFilter

This Class is used to filter the message and apply certain values to certain message fields that are passed into it.

Variables:

_tag – This is a private variable used to store the typeTag value to be applied by this filter.

Functions:

TagFilter (destinationTag: string) – This is a Constructor for the **TagFilter** Class that takes the destination value to be applied as a parameter.

Run (message: Message) – This Function takes the passed in message and applies the **_tag** value to it then returns the modified message.

4.1.4.1.8 Message

This class is used to parse and store the message data that the controller both sends and receives.

Variables:

_destination – This is a private variable used to store the destination data of the current **Message** instance

_source – This is a private variable used to store the source data of the current **Message** instance

_typeTag – This is a private variable used to store the type data of the current **Message** instance

_timeStamp – This is a private variable used to store the time of creation for the current **Message** instance

_content – This is a private variable used to store the actual contents of the current **Message** instance

Functions:

Generic Getter and Setter Functions To access the private Properties.

Message(message : string) - This is a constructor for the Message Class that takes a single string with all of the message data separated by commas for a parameter. This is then split up and assigned to thew appropriate variables.

Message(destination : string, source : string, type : string, body : string) – This is another constructor that takes in all of the individual values as separate string and assigns them to the corresponding variables generating a new time stamp for the **_timeStamp** variable

4.2 User Interface Design

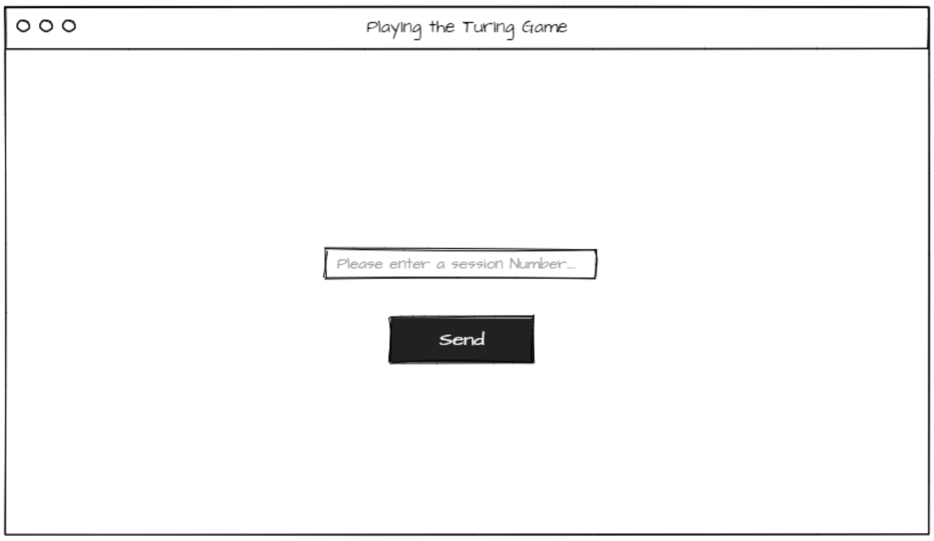
This section will cover the Wireframe designs for the application user interfaces and the final designs and the contrast between the two.

4.2.1 Client UI wireframe designs

This section will cover the original wireframe diagrams for the views that will be presented to the user of the client application.

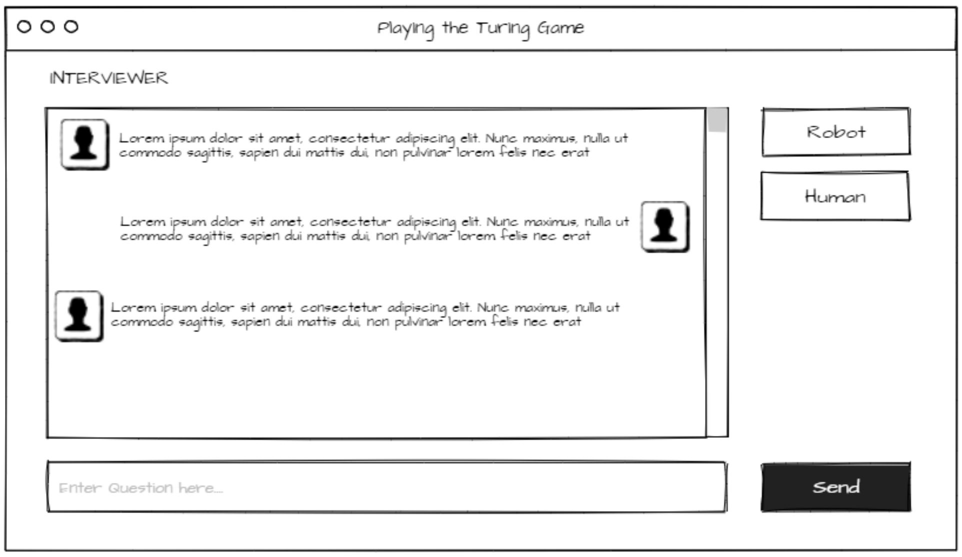
4.2.1.1 Initial view

The diagram below is the initial design for the view that is presented to the user of the client application upon start up. the box in the centre of the screen will allow the user to enter the session code provided by the Teacher and the Button below it will allow them to submit the entered text.



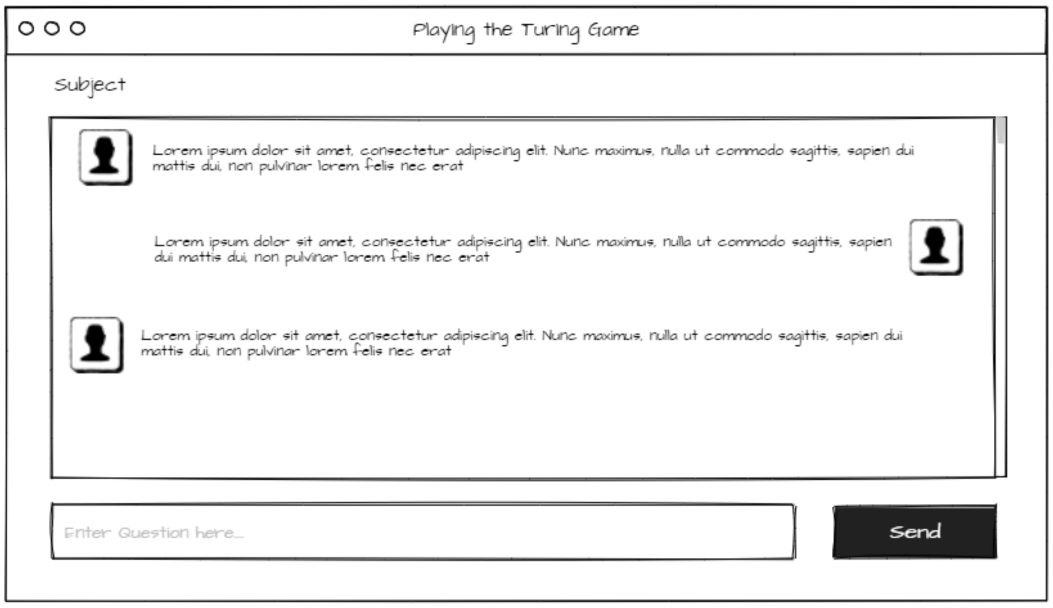
4.2.1.2 Interviewer View

The diagram below is the initial design for the view that will be presented to the Clients end user if they are assigned the Interviewer Role. The central box will be a scrollable window which will be used to display the messages that have been sent and received. The Box Below it is for users to enter their messages to send. The button to the right of this box is used to submit the entered text and send the message. To the right of the message box will be two buttons which can be used by the Interviewer to submit their guess as to who they're speaking to.


Playing the Turing Game
INTERVIEWER
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nunc maximus, nulla ut commodo sagittis, sapien dui mattis dui, non pulvinar lorem felis nec erat
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nunc maximus, nulla ut commodo sagittis, sapien dui mattis dui, non pulvinar lorem felis nec erat
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nunc maximus, nulla ut commodo sagittis, sapien dui mattis dui, non pulvinar lorem felis nec erat
Robot
Human
Enter Question here...
Send

4.2.1.3 Subject View

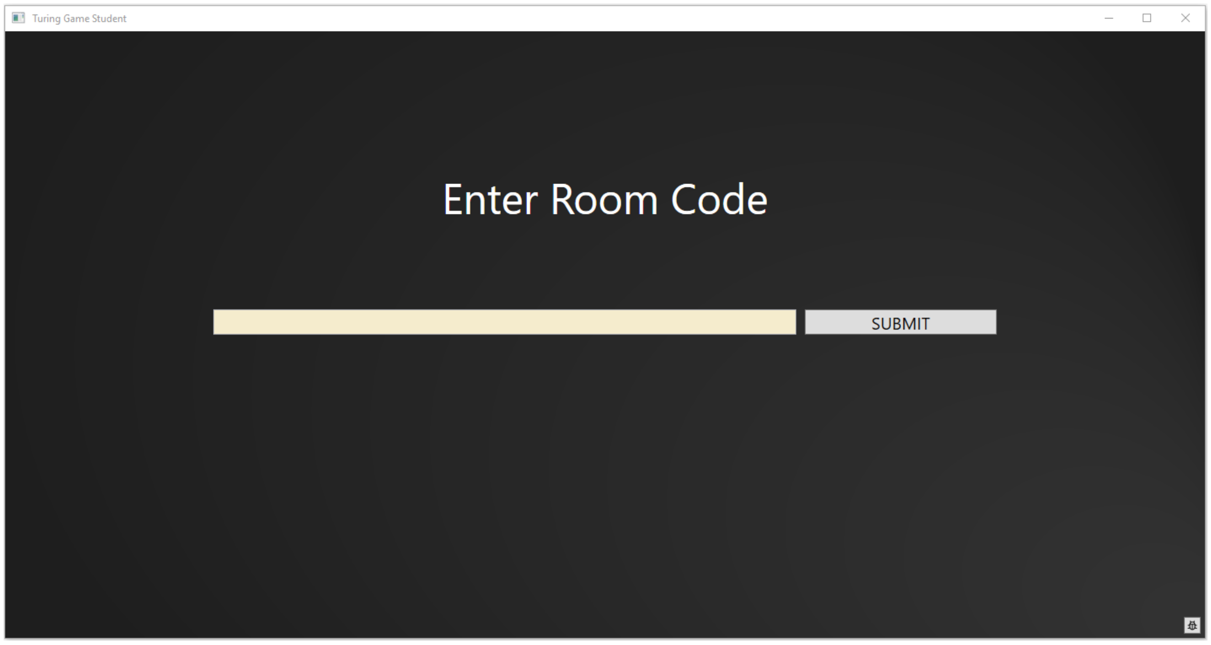
The diagram below is the view presented to the client if they are assigned the role of Subject. It is largely the same as the Interviewer view however does not have the Robot and human buttons on the right side of the screen.


Playing the Turing Game
Subject
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nunc maximus, nulla ut commodo sagittis, sapien dui mattis dui, non pulvinar lorem felis nec erat
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nunc maximus, nulla ut commodo sagittis, sapien dui mattis dui, non pulvinar lorem felis nec erat
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nunc maximus, nulla ut commodo sagittis, sapien dui mattis dui, non pulvinar lorem felis nec erat
Enter Question here...
Send

4.2.2 Client UI Final designs

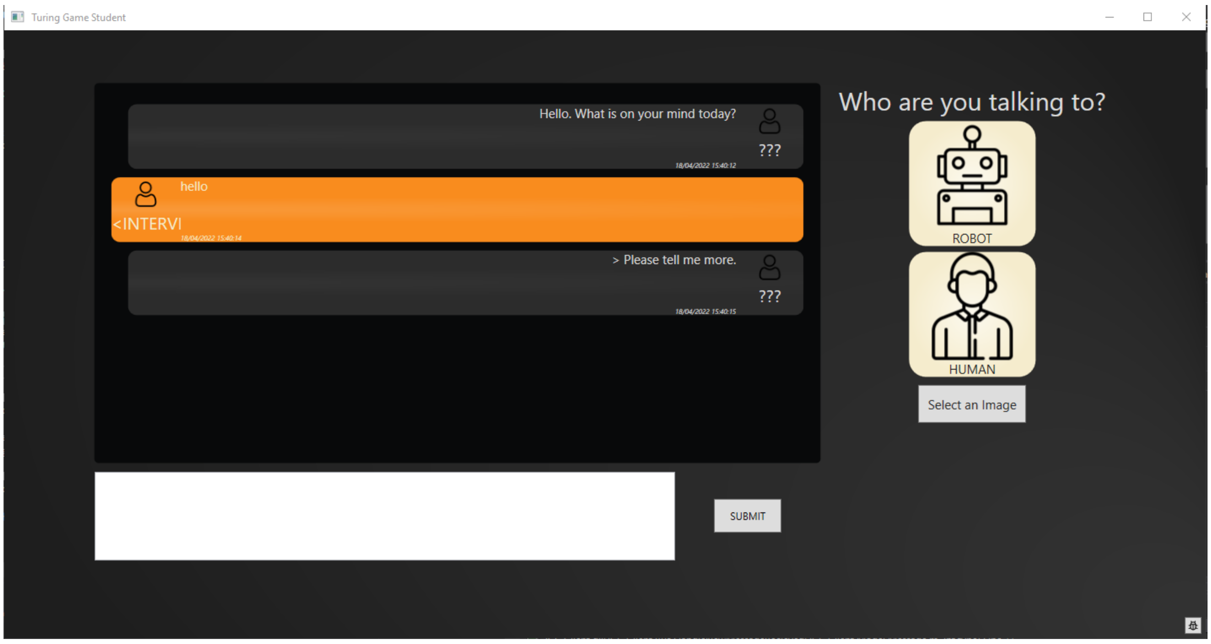
This section shows the final implemented designs of the user interface of the Client application.

4.2.2.1 Initial view



The above image is a screen shot of the final implementation of the client's initial view it remains largely consistent with the original design with only minor changes such as moving the submit button to be inline with the text entry and adding Text to tell the user what to do on this screen making it less ambiguous. A button has also been added in the bottom left which opens up a debug window at the bottom of the screen which displays debug messages and code events to the user if required.

4.2.2.2 Interviewer View



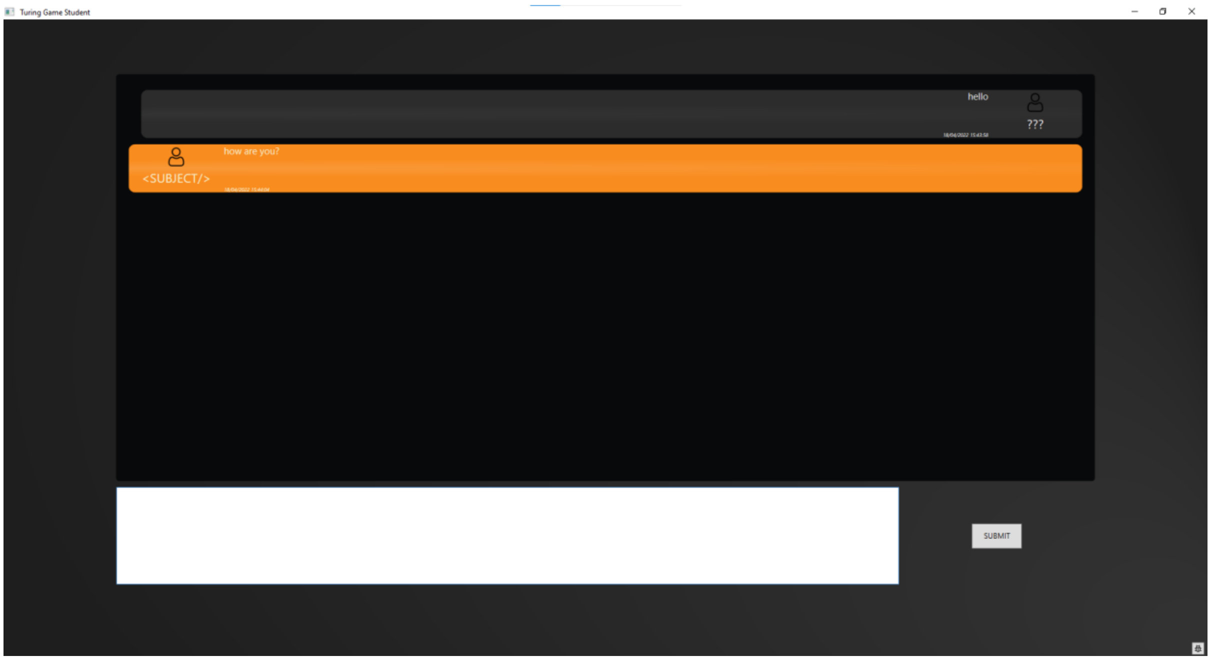
Hello. What is on your mind today?
???
18/04/2022 15:40:12

hello
<INTERVI
18/04/2022 15:40:14

> Please tell me more.
???
18/04/2022 15:40:15

The image above is the final implementation of the Interviewer view presented to the user the design has stayed mostly true to the original the main change occurring around the user selection element of the display. This has been expanded to include a pictographic representation for the options as well as the words to make it more accessible for people with reading difficulties. A button was also added below the two images to submit the choice so that the user can be sure they are submitting the answer they want to submit.

4.2.2.3 Subject View



hello
???
18/04/2022 15:43:58

how are you?
<SUBJECT/>
18/04/2022 15:44:04

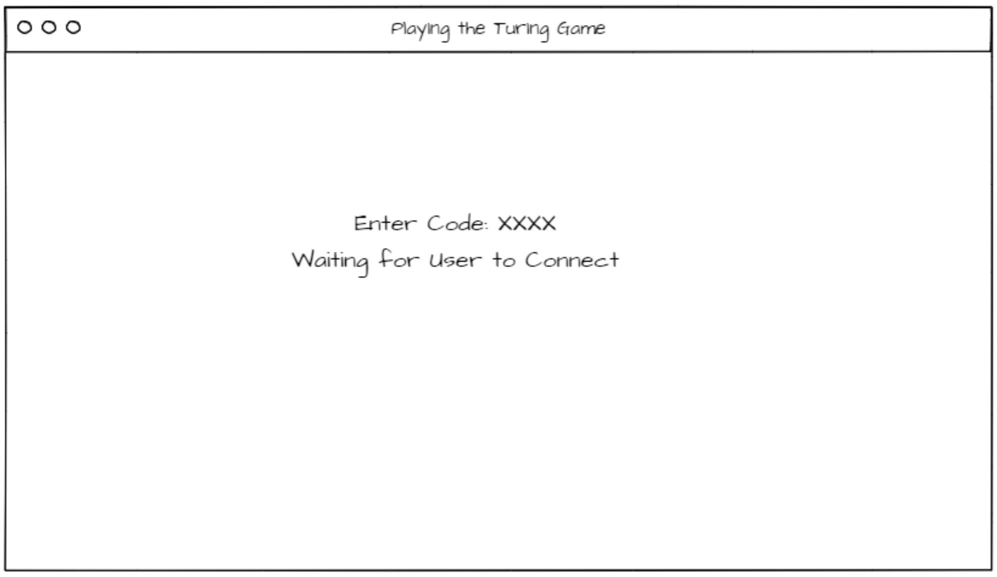
The above image is the final implementation of the Subject View there is very little change between the initial design and the final design the only real difference being the inclusion of the debug button in the bottom left which has been added for testing purposes.

4.2.3 Controller UI wireframe design

This section will cover the original wireframe diagrams for the views that will be presented to the user of the Controller application.

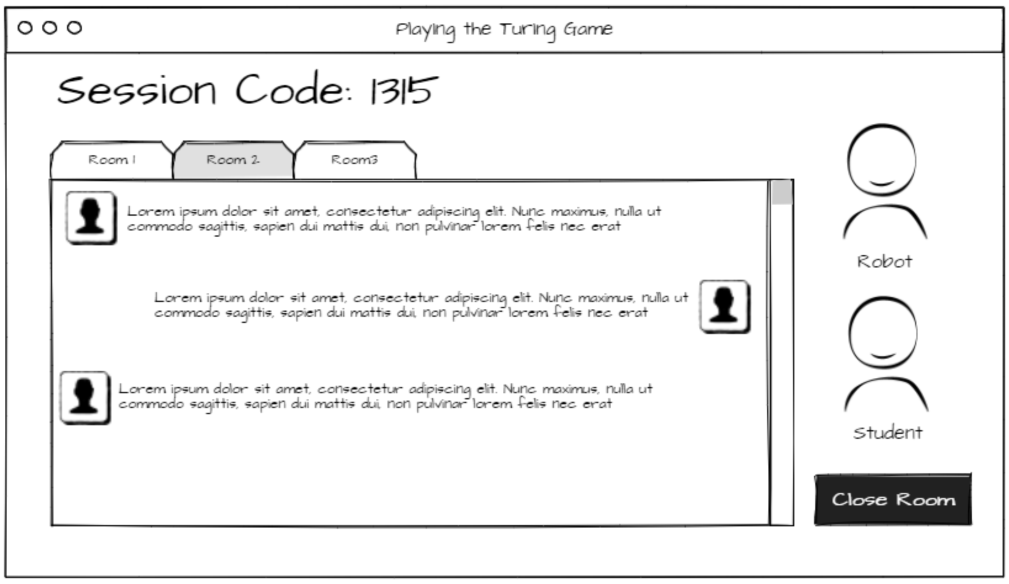
4.2.3.1 Initial View

Below is the initial wireframe design for the Controller application It will display the randomly generated Session code until a user connects to application.



4.2.3.2 Active View

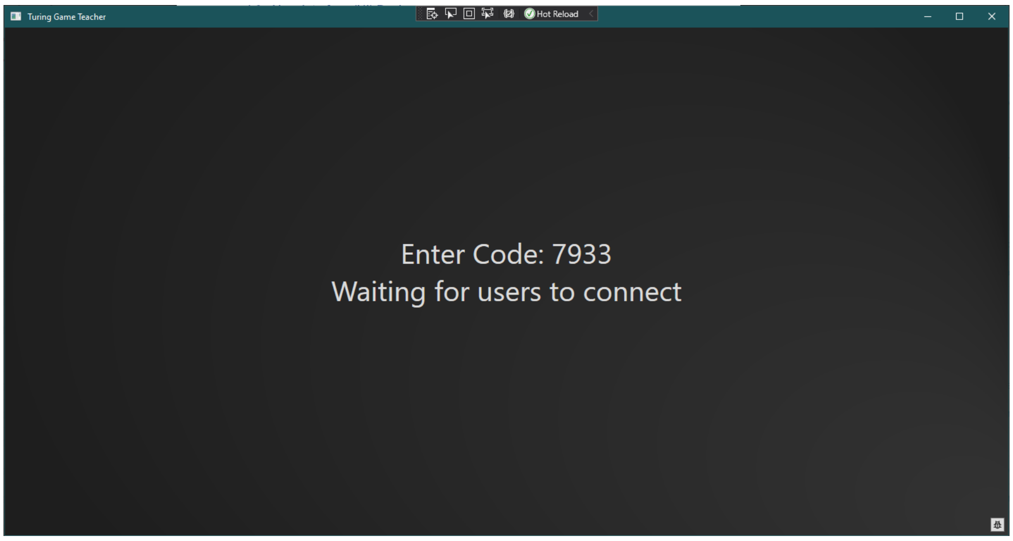
The diagram below is the initial design for the Controller application while in the active state. The central window will be a changeable view allowing the User to switch between the various active rooms and view the conversations in each one. The session code will continue to be displayed above the changeable window. On the right-hand side of the screen the user will be able to see what is in the room whether it be two humans or a robot and a human. The button in the bottom right will allow the User to close the room they are currently viewing.


Playing the Turing Game
Session Code: 1315
Room 1
Room 2
Room3
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Nunc maximus, nulla ut commodo sagittis, sapien dui mattis dui, non pulvinar lorem felis nec erat
Robot
Student
Close Room

4.2.4 Controller UI Final designs

This section shows the final implemented designs of the user interface of the Client application.

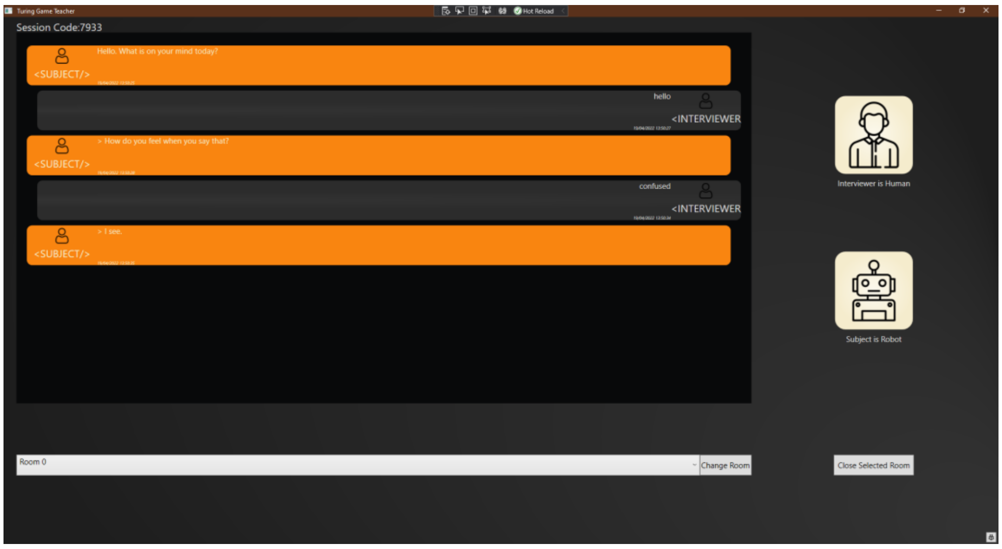
4.2.4.1 *Initial View*



Enter Code: 7933
Waiting for users to connect

The above image is the Final implementation of the initial view presented to the End-user upon start up of the Controller application. As can be seen the Design has stayed very much the same with minimal changes to the Text being displayed. This view once again has the debug button in the bottom right to display the debug window for testing purposes.

4.2.4.2 *Active View*



Session Code:7933

<SUBJECT/> Hello. What is on your mind today?

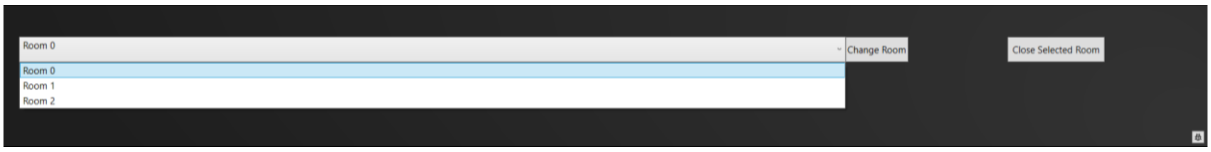
hello <INTERVIEWER

<SUBJECT/> > How do you feel when you say that?

confused <INTERVIEWER

<SUBJECT/> > I see.

The image above is the final implementation of the active view or monitoring view of the Controller application the design is very much in keeping with the initial wireframe. The Tabbed view in the wireframe has been swapped out and replaced with a drop-down selector to change views which is shown in the image below. This was done due to time constraints around researching how to achieve the same functionality with a tab view, as the drop down works to prove the concepts of swapping out the view.



5 Implementation and testing

Delete the red paragraphs and replace this one with your content (use the “Normal” paragraph style).

5.1 Constraints

There are several Constraints that have an impact upon this project These include Time as well as a knowledge Constraint. The Main Constraint on the Time of this Project is in the form of availability of the Developer. The developer is only able to put time into the Project during term times as during study leave and university holidays they are working full time. This is required by their employer who is sponsoring them and therefor is something that must be worked around.

The Knowledge Constraint of the developer is that they have very little experience in the field of network communication or user interface design in code therefore there is a steep learning curve as they attempt to research and apply the knowledge they gain throughout the Project. This is reflected in some of the designs and implementations which are not quite as polished as they could be but due to the time constraint there wasn’t sufficient time to go back and correct these designs.

5.2 Work Management

In the Project Initiation Document (PID) The projects workload was going to be managed using an Agile methodology with a two-week sprint window for work to be completed in and the effectiveness of the work rated to improve the process. This did not prove very effective with a single developer as performing the reviews every two weeks began to slip and performing them took a lot of time which was not conducive to the Strict time Constraints on the Project as set out in the previous section. In order to combat this the approach for managing the workload was changed to a Kanban style. This required very little change to the GitHub Project control as the task where already loaded in the board, it just meant treating them as a backlog and working on one task at a time and ticking it off which has a lot less overhead than the agile methodology as there was no need for a two-weekly review of the progress.

5.3 Time Management

The Time management of the project was established in the PID using a Gantt chart. This however proved to be ineffective due to a lack of experience in the subject area which led to over estimation of some tasks and under estimation of other tasks. This led to the schedule falling behind coupled with the periods where the developer was unavailable, as outlined in the Constraints section, meant that getting back into the roll of certain tasks that may have been left unfinished dragged certain tasks out. In Future projects time will be allocated to allow for these over runs and to focus on tasks to catch the work up to the schedule before continuing in the initial plan there was a small amount of time allowed for this however it did not match the underestimation and so a lot of work had to be put in in order to produce the project and some of the less crucial elements had to be dropped such as the additional documentation. Another oversight was the assumption that spending 24 hours a week on the project was realistic this quickly became unmaintainable due to burn out from working on the same thing as well as having to devote time to the other requirements of the developer such as coursework for units.

5.4 Implementation

The entire project has been developed using Visual studio 2019 in the C# Programming language using the .Net framework. Alternatives to this that were looked into where java and python however the developer was unfamiliar with these two languages and visual studios extremely intuitive and helpful tools to speed up the development process the decision was made to progress with C#

C# provides access to many functionalities which are useful for the project as it allows access to developing WPF applications and databinding as well as access to libraries such as systems.Net.

Systems.Net provides access to many classes that allows for the development of network-based applications these include:

The Socket class which handles the creation of a network sockets and allows for network communication. This is done by initialising the Socket to use a particular type of transport protocol (e.g UDP TCP or Raw) the Socket can then be told to listen for a connection attempt and accept it when it finds one.

The IPEndpoint Class which is a data structure which holds the complete address of the application that is being connected to, this is done by combining the IPv4 address and the port number thus creating a full address.

The TcpClient Class which allows an application to remotely connect to a network socket via a supplied IPEndpoint. This Class can be set to attempt to connect to the provided IP endpoint and once it is accepted it will create a Data Stream between the two making use of TCP transport Protocols.

In order to develop the User interface for the project WPF formed the main infrastructure that was used which is designed with MVVM in mind and allows for databinding and recycling of views. Windows Forms was looked into as an alternative however it is largely considered a legacy technology and will not receive updates from Microsoft. This led to the decision to go with WPF as this is considered the current technology and although it is initially harder to set up once the pattern is in place it is easy to follow.

In order to simulate an AI a chatbot was included in the project however the decision was made for the chatbot not to be a main focus and instead to make the chatbot a hosted executable by a wrapper class. This decision was made so that the solution could easily have the chatbot swapped out without affecting the functionality. The chatbot that is used is an opensource project (solo-rey, 2018) This has benefitted the project as it allowed more time to focus on areas where the developer as struggling or falling behind.

5.4.1 Implementation of client connection

The user can initiate connecting the client by entering the supplied public Port number in the text field provided upon initial start-up. upon first time set up the IPv4 address of the Controller device will need to be entered into the TGF_config.txt file in order for the client to construct the IpEndpoint for the Controller. This will start the Handshake between the two devices, upon completion of the handshake the client will have been supplied with a new secret port number as well as a role which will determine which view is displayed to the user and which room the client is connected to.

5.4.2 Implementation of client Sending and receiving data

In order to send and receive data asynchronously the solution uses two connections per client a primary and a secondary connection. The primary connection is used to send data to the Server the secondary is to receive data from the server this is set up as soon as the client receives the secret port number for the room it should connect to. For the user to send the data they simply need to enter a message into the text box on the Subject or interviewer Screen and hit send at which point the text is converted into a message object and sent to the server via the primary data stream. The client can receive messages whenever and as soon as they receive one on the secondary data stream it will be parsed into a message object and displayed to the screen in the message window.

5.4.3 Implementation of Interviewer Client submitting selection

The user of the Interviewer Client application can select their choice by clicking on either the robot or human image. The selection can then be submitted by clicking the button below the two images this will create a special message object which is hidden from the Message board and sent to the server which will handle the message it receives accordingly.

5.4.4 Implementation of the Client Server Handshake

The initial Connection between a Client application and the server application has been dubbed the Handshake. The first step is for the server to open up the public port this is the number that is displayed to the user and that the clients will enter on the initial screen. The second step is a client will try to connect to this public port, once it successfully connects the Controller will check for availability in any existing rooms, if there is a room available it will send a response back to the Client with the private port number of the available seat in the Room and the role that is available. If there is no room available the Controller will create a new room instance and send the private port number and role back to the client. Upon receipt of the response message from the Controller the Client will break down the message and disconnect from the Controllers public Port thus allowing the next client to connect. After disconnecting the Client will now use the private port number to connect to the room once this is complete it will change its view to match its assigned role and the handshake is complete.

5.4.5 Implementation of server Sending and receiving data

The clients are connected to the Server via a room each room has 4 sockets associated with it these are the primary and secondary connections for each client. This allows for a maintained connection while still allowing for asynchronous message sending. When the Server receives a message from a client it reads the message and adds it to the message board being displayed for the room and then forwards it onto the other client conversely to the clients the server reads from the primary connections and writes to the secondary connections this decision was made to simplify the explanation when describing the connections between server and client.

5.4.6 Implementation of the Chatbot wrapper

The Chatbot wrapper uses the same connection methods as a normal client would this allows the swapping out of the chatbot without a need to alter the Server at all. The Chatbot wrapper is an independent application that activates the chatbot executable as a process and redirects the input and output streams of the executable so that it can feed in the received messages and capture the chatbot responses. The chatbot responses are relayed to the server by writing to the primary stream between the Wrapper class and the server much like the normal Client. For the Purposes of this project an open source python script developed by (solo-rey, 2018) on GitHub has been used for the executable.

5.5 Testing

This section will cover the various tests that are carried out on the applications that make up this project in order to satisfy the requirements. The below table is the Test script for the client application used to test against the requirements of the solution.

step	Operator action	Outcome	Requirement
1	**Test ability to connect to another application**		TGF_REQ_CA_01: Connection
1.1	Start-up Client Application	The client application opens up	
1.2	Start-up Controller Spoof Application	A console window opens up	
1.3	Start-up room spoof application	A Console window opens up	
1.4	Enter 5000 into the Client application	Controller_spoof prints to the console "a device is attempting a connection"	
1.5	Enter continue into the Controller spoof console	Controller_spoof prints to the console "Forwarding Connection to room" Room Spoof prints to the console ""a device is attempting a connection"	
2	**Test ability to send and receive data from client**		TGF_REQ_CA_01: Send Message TGF_REQ_CA_03: Receive Messages TGF_REQ_CA_05: User Input
2.1	Setup: repeat the steps from Test 1	Same as Test 1	
2.2	Enter a message into the Text box of the Client application	N/A	
2.3	Hit the send button on the UI	The text box clears and the message appears in the message board on screen The room console window displays the message as a string.	
2.4	Enter a response message into the console window of the Room_spoof application and hit enter	The message written in the console window is displayed in the message board of the Client application.	
3	**Test ability to submit user input**		TGF_REQ_CA_04: Submit Partner
3.1	Set up: repeat steps from Test 1	Same as Test 1	
3.2	Click on the Robot image on the Client application	The button text on the Client application changes to say "Robot"	
3.3	Click the button under the images to submit selection	The Room_spoof console displays "user selected Robot"	
3.4	Click on the Human image on the Client application	The button text on the Client application changes to say "Human"	

3.5	Click the button under the images to submit selection	The Room_spoof console displays ”user selected Human”	
4	**Test ability to receive connections**		TGF_REQ_SA_01: Connection
4.1	Launch the Controller application	Controller Application will open and the user is presented with a button to press to start the session	TGF_REQ_SA_01: Connection
4.2	Launch an instance of the Client application	The Client application opens and presents the initial view to the user	
4.3	Click Launch on the Controller application	The screen changes to display a session code along with text that reads “waiting for users to connect”	
4.4	Enter the Session code into the Client application and click Submit	The Client application screen will change to be the Interviewer View The Controller Application will change displays to display the Active View.	
5	**Test ability to send and receive data between Clients via the server**		TGF_REQ_SA_02: Send Message TGF_REQ_SA_03: Receive Messages
5.1	Repeat the steps from Test 4	Outcome is the same as Test 4	
5.2	Open a second Client application	The Client application opens and presents the initial view to the user	
5.3	Enter the session code into the Second client application	The second Client application presents the Subject View to the Client No change to the Controller application.	
5.4	Enter a message on either Client and hit send	The message should appear in both the clients Message board screens The message should also appear on the Server Message board screen	
5.5	Enter a response message on the opposite Client and hit send	The message should appear in both the clients Message board screens The message should also appear on the Server Message board screen	
6	**Testing the ability to create new chatbot instances**		
6.1	Repeat the steps from Test 4	Outcome is the same as Test 4	

6.2	Click the debug button in the bottom right of the Controller application	The debug window will state “Room:0 has obtained interviewer”	TGF_REQ_SA_05: Launching the Chatbot TGF_REQ_SA_02: Send Message TGF_REQ_SA_03: Receive Messages
6.3	Wait 10 seconds	The debug window will state “Room:0 has entered bot launch” a message should appear in the chat window that says “Hello. What is on your mind today” on both the controller and client application The controller application will be updated and show that the interviewer is human and the Subject is a Robot	
7	**Testing the ability to manage new connections as required.**		TGF_REQ_SA_0: Creating new rooms as required
7.1	Repeat the steps from Test 4	Outcome is the same as Test 4	
7.2	Wait 10 seconds for the chat bot to Launch	Outcome the same as test 6	
7.3	Check the drop-down list of available rooms on the Controller application	Should only have Room 0 available	
7.4	Launch a new instance of the Client application and connect it to the server	The client should change to display the Interviewer display The Controller drop-down should have a new option Room 1	
7.5	Repeat step 7.4 3 more times	Same outcome but with the appropriate room number after each new option.	
8	**Testing the stability of the connection**		TGF_REQ_REL_01: Stability
8.1	Repeat the steps from Test4	Outcome same as Test 4	
8.2	Leave applications and devices running for 24 hours	The applications are still running correctly	
8.3	Send a message from the client	The sent message appears on both the Client and Controller message boards. The Chatbot sends a response to the message that was sent.	
9	**Testing the reliability of the Client**		TGF_REQ_REL_03: Server Disconnect
9.1	Repeat the steps from Test 4	Outcome matches the Outcome from Test 4	
9.2	Close the Controller application after the Client receives a message from the Chatbot	The client displays a received message stating “Connection Terminated”	

9.3	Close the Client Application	Client closes immediately	
9.4	Repeat the steps from Test 4	Outcome matches the Outcome from Test 4	
9.5	Click Close room button on Controller application	The client displays a received message stating “Connection has been terminated by remote host”	
9.6	Close the Client Application	Client closes immediately	
10	**Testing the reliability of the Controller**		TGF_REQ_REL_02: User Disconnect
9.1	Repeat the steps from Test 4	Outcome matches the Outcome from Test 4	
9.2	Close the Client Application	The Controller continues to run without freezing or crashing	
9.3	Close the Controller Application	Controller closes immediately	

6 Evaluation and discussion of results

The overarching goal of this project was to design and develop proof of concept for a tool that can be used in schools to aid in teaching of AI concepts by providing a series of applications that allow students to take part in The Turing game. Each of the requirements in this document is linked back to an initial aim or objective and the tests in the Test script are designed to test that the code satisfies each of the Requirements.

6.1 Achieved Objectives

After research had been carried out into network communication and the MVVM project structure Two basic console applications were also developed in order to Test the cross-network communication of the Client application in the initial stages. These console applications are referred to as Controller_spoof and Room_Spoof for the purposes of this document. The Spoof applications would print out everything that was sent to them in order to verify the message content was being received correctly and would convert any entered text into a message to be sent back.

6.1.1 Objective 1 – Design and create an application that would act as the client connecting to the server application remotely

The first part of this Objective was met by carrying out thorough research into network-based applications and attending Lectures for the Distributed Systems Course at university which covers network-based principles and best practices in order to gain the required understanding of network communication to implement it into the design for the various applications in this Project.

The Second part of this objective involved understanding how to incorporate asynchronous and parallel tasks into the Design so as to allow for seamless communication between clients using a TCP Type Connection. This is necessary as attempting to read from the Data stream of the TCP client locks out the thread that is attempting it until it receives a message This is an issue on a single threaded application as it would mean messages must be sent and received like a zip one after the other and the client that was waiting to receive would become unresponsive until it had received the next message. One approach for this was to make the action of reading a message an async task that could be called after a message is sent. This would keep the application responsive but would also enforce the single message call and response pattern as the data stream would be locked out for being read from in the application meaning no more messages could be sent till the async task completed. In order to combat this the application design was modified to incorporate a dual stream or two channel system into the application with one Socket dedicated to Writing data and another dedicated to Reading. This meant that the Read and Write tasks could run completely separate to each other allowing for multiple messages to be sent or received in any order.

The next 2 parts of this objective were to develop the actual application implementing the MVVM Structure that had been researched at the beginning of this project however this was not strictly followed as there was some confusion by the developer about how to pass data back from the view model to the business logic and so to keep this abstracted a Static class is used which masks the business dependencies from the View models which ultimately conforms to the philosophy but not in the Truest sense.

The final part of this objective was to test the developed application the method for this is outlined in the Test script in the Implementation Section of this Document.

6.1.2 Objective 2 – design and create an application to act as the server / Teacher's application

To achieve the First part of this objective it was necessary to design a data structure for the Controller application that would allow two clients to connect to it and communicate. In Order to do this the Room Class was developed which had two open connections one for each client. Each connection Reflects the Dual channel design implemented in the client and so has a primary and secondary port number associated with them which is determined upon creation of the room.

The second part of this objective involved developing the "Handshake" Sequence. This involves the Controller providing a public Port number which is referred to as the Session code and sending the private port numbers for available rooms to any client that connects. This process allows clients to connect and then be redirected to the available rooms as per the stated objective.

The third part of this Objective was to design an extra step into the handshake sequence which will allow the controller to create new rooms when required. This was done so that there is only ever the minimum number of rooms running at a time to limit resource costs of the application. This was achieved by planning how the port numbers for the Rooms would be incremented to avoid clashes and also add a limit to the number of Rooms which can be created every time a room is created the private port number for the room is incremented by 100 each time. Then a new thread is created to manage that room.

For the final two parts of this objective the application has been built and tested using the Test script detailed in the Implementation section of this Document. Again, much Like with Objective 1 the MVVM structure was not strictly adhered to and If more time had been available for the Project then further redesign and development work would be put into correcting this.

6.1.3 Objective 3 - design and create an application that can host a chatbot to act as the AI in the Turing game scenario

To achieve the first part of this Objective Research was carried out into available chatbots and what the most common way for chat bots to handle input and output of data in order to incorporate this into the design this research identified that most data input/output for chatbots can be captured by overriding the input and output streams for the executable. The off the shelf opensource solution provided by (solo-rey, 2018) was perfect for testing the capabilities of the proposed Chatbot Wrapper application.

The application was designed to interact with the server as if it is another Client and so the connection methods are copied from the Client application this worked as expected. This was an important step as it meant that the chatbot could be developed independently of the server and the possibility of supplying different chatbot options in future become available for future development.

6.2 Failed Objectives

This section covers any Objectives that were not captured or met as part of this Project

6.2.1 Objective 4 - design a ruleset or scenario that uses the above-mentioned tools to create an interactive game for the students with scores etc.

This objective was not met due to the strict time constraints upon the Developer and due to unforeseen shortcomings in the effectiveness of the project plan. The concepts for future development where designed but Unfortunately, they were not implemented into the Final project

These concepts included:

Having a score on the Client application that recorded how many times The Client has correctly Identified what they are talking to as an Interviewer, or how many times They managed to Trick the Interviewer as a subject. This could have been used as a basis to further develop a potential True AI chatbot that could learn to trick the Interviewers more effectively by using the Score as the success condition in an adversarial system.

Allowing the Teacher to specify how many chatbots are present in the current session This was not successfully implemented and so for testing purposes and demonstration purposes the Chatbot is launched after 10 seconds of an Interviewer being in a room on their own. And there can be as many chatbots as users.

With additional time a room shuffle functionality would have been added to the Server which allows the server to shuffle the rooms up and mix up the pairs so that users would be in a different room with potentially a different role. This would develop on the game concept of the score keeping aspect to make it a longer running game.

7 Conclusion

In this section you should evaluate the *project* as a whole, and draw conclusions from the work you have done. Ask yourself what the project has achieved – what is its contribution? Has it met its initial aims and objectives? If not, why? How does the work you have done enhance the field in general? What has been learned from the project? If you have a well defined research question, has it been answered? What do the results mean?

You should also use this section to reflect on the *process* by which you undertook the project. Was your methodology appropriate (and did you stick to it)? Was your time planning good? Did you complete the primary and secondary objectives, and if not then why? What have you learned from the process? What would you do better/differently if you had more time?

Sometimes, it's appropriate to include a subsection on 'Further work', making suggestions of how to proceed and what could be done to enhance the project in future.

Delete the red paragraphs and replace this one with your content (use the "Normal" paragraph style).

8 References

dotnetforall. (2022, December 13). *WPF MVVM Practical Data Application*. Retrieved from dotnetforall.com: https://www.dotnetforall.com/wpf-mvvm-practical-data-application-example/

Fortinet. (2022, March 03). *What is a Transmission Control Protocol TCP/IP Model?* Retrieved from fortinet.com: https://www.fortinet.com/resources/cyberglossary/tcp-ip

Lutkevich, B. (2022, November 6). *Transmission Control Protocol (TCP)*. Retrieved from techtarget.com: https://www.techtarget.com/searchnetworking/definition/TCP

mdn web docs. (2016, July 04). *An overview of HTTP*. Retrieved 11 07, 2022, from developer.mozilla.org: https://developer.mozilla.org/en-US/docs/Web/HTTP/Overview

Oppy, Graham, & Dowe, D. (2003, April 09). *The Turing Test*. Retrieved from Stanford Encyclopedia of Philosophy: https://plato.stanford.edu/entries/turing-test/

solo-rey. (2018, February 2). *command-line-chatbot*. Retrieved from Github: https://github.com/solo-rey/command-line-chatbot

Turing, A. (1950). Computing Machinery and Intelligence,. *Mind*, 433-460.

Yadav, r. (2008, March 11). *DevMentor.org.* Retrieved from semanticscholar.org: https://web.archive.org/web/20170409201335/https://pdfs.semanticscholar.org/0858/d27611d7a90c221beb5106e63721aeaed810.pdf

Appendix A – Interesting but not vital material

Appendices are used to include information which may be of interest but is not necessary for the reader. You do not have to include appendices if there is no need for them.

You might, for example, want to include some details of a particular piece of software (an API, perhaps) or hardware which your project uses. This might be something that a reader might wish to consult, but you wouldn't want to include in the main body of the report. You could also put raw data from experiments in an appendix, or perhaps survey results. It should still be information of relevance, but nothing that everyone would be expected to read.

If you wish to refer to elements of your PID, you could include them in appendices.

Delete the red paragraphs and replace this one with your content (use the "Normal" paragraph style).

Appendix B – Links to Full Size Class diagrams

Double Click on the image to view the full-size image