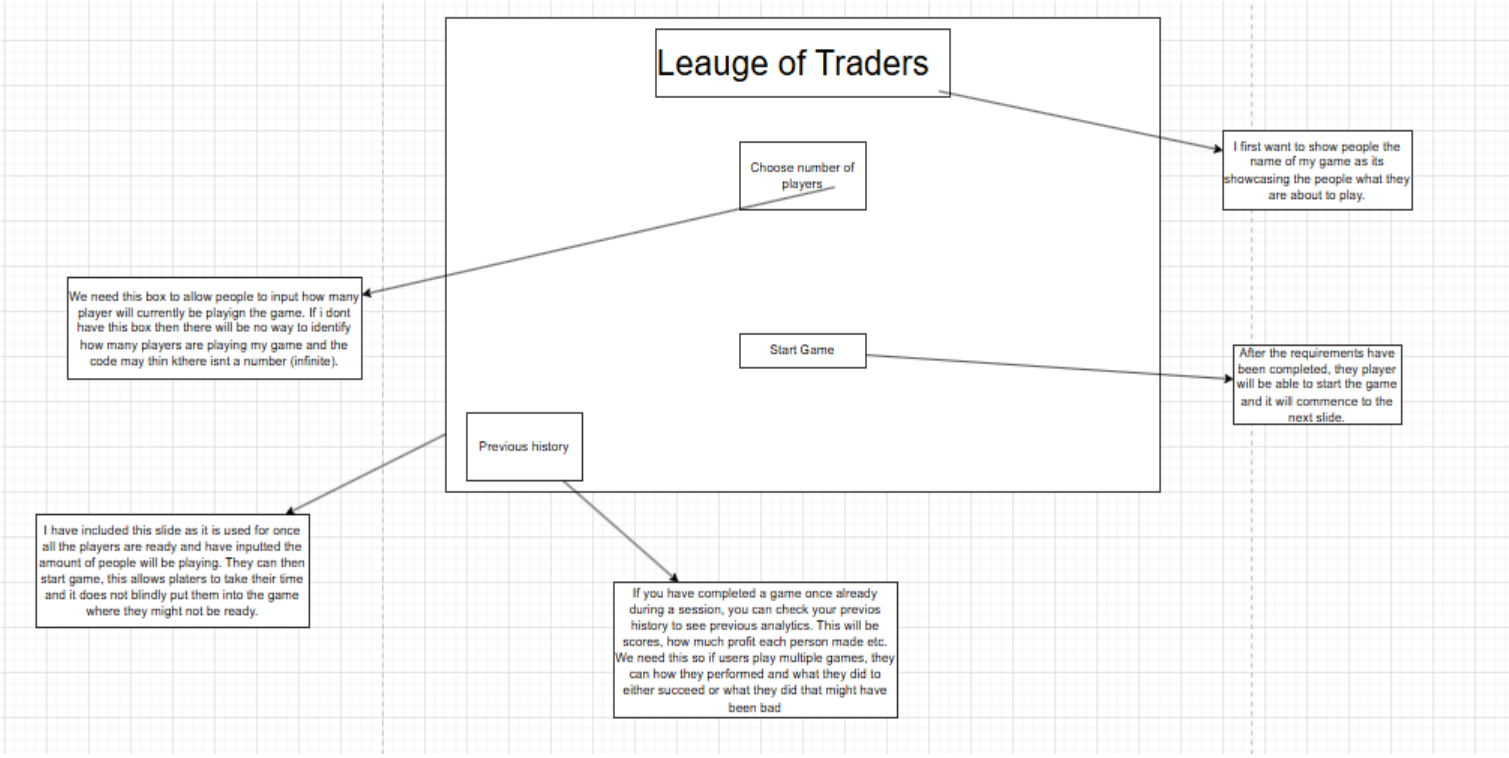
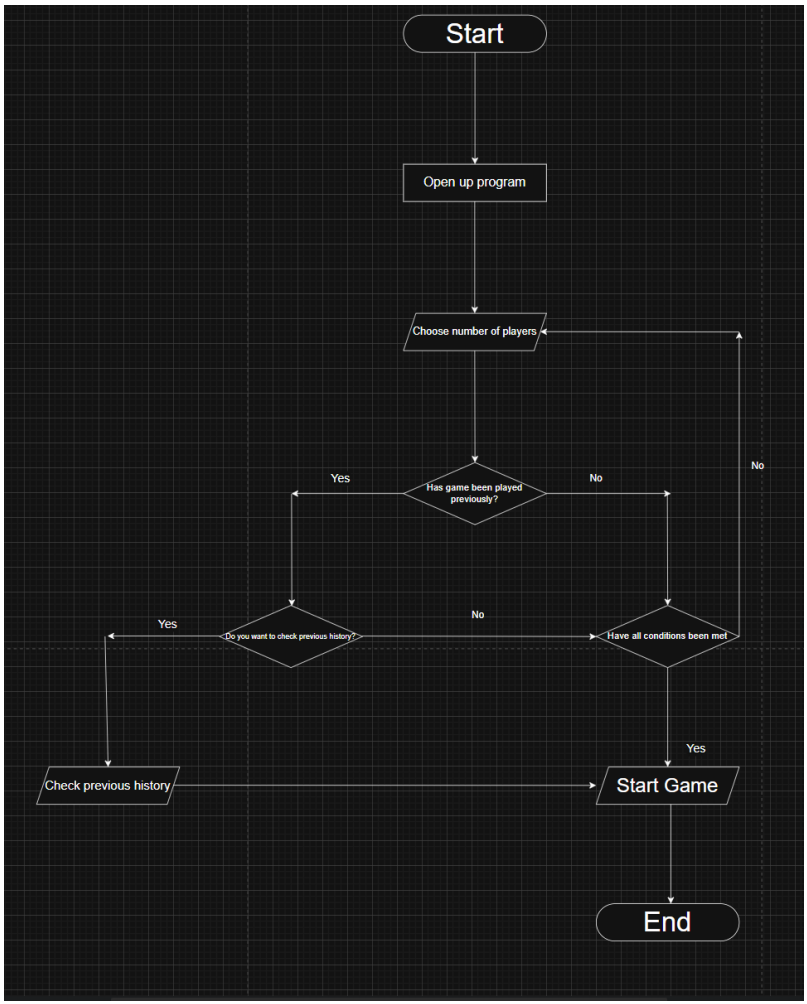
Iterative Development – Stage 1

Stage 1:



ALGORITHM:

Flow chart:



Iterative Development – Stage 1

The flow chart is used to show how exactly my menu will run. The start initialises the start of my program. Once that begins and you open the program, the first thing you should do is choose between the set number of players on the game, which is between 2 and 8. After that is done you then get sent to the next section which is if, if you have played a game before you will have the option to check your previous history. However, if its not done then you can start the game. Once you check history you can leave that section and get straight into the game.

Pseudocode:

Open window “New Game”

##Add details to the game

Set windowTitle = “League of Traders”

Set windowIcon = “app.png”

Set windowBackground = “Trading.png”

// Show game title

DISPLAY label "League of Traders"

Display label “Choose number of players”

// Input: number of players

CREATE dropBox “Choose number of players”

// Button to start game

CREATE push button "Start Game"

Iterative Development – Stage 1

```
// Button to view previous history
If previous history exist:
        Create pushbutton “Check History”


// Event: Start Game button clicked
IF Start Game button CLICKED THEN
   Print(“Game is starting with”, playerCount, “Players”)


// Event: Previous History button clicked
IF Previous History button CLICKED THEN
   RETRIEVE previous game data
   DISPLAY previous scores and analytics
```

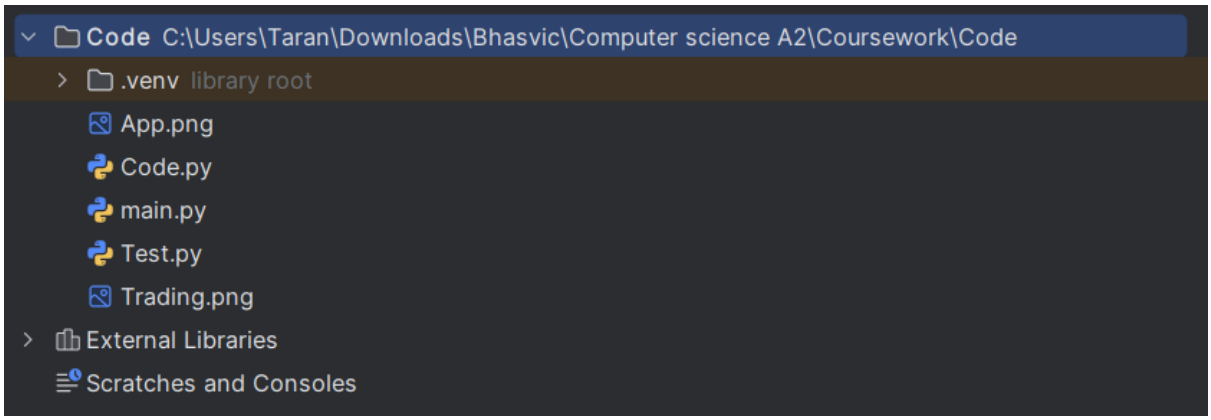
Iterative Development – Stage 1

Filed Name	Data Type	Data Format	Field Size	Description	Example
Field Name	**Data Type**	**Data Format**	**Field Size**	**Description**	**Example**
app	QApplication	Object instance	None	Initalises the start of your gui (game can start now)	app = QApplication(sys.argv)
window	MainWindow	Object instance	None	This the window that pops up in which the game is now starting	window = MainWindow()
MainWindow	class	Class Definition	None	This allows the tab to contain buttons, titles, inputs etc	class MainWindow(QMainWindow)
label_title	QLabel	String	1 - 20 characters	Piece of string that displays the title of my game	"League of Traders"
label_background	QLabel	QPixmap object	1 - 20 characters	This is the widget that gives my screen a dynamic background for my users to see	QPixmap("Trading.png")
button_start	QPushButton	String	1–20 chars	These are interactable buttons that perform an action, this one will allow you to go the next stage of the game	"Start Game"
button_history	QPushButton	String	1–20 chars	These are interactable buttons that perform an action, this one will allow you to check the previous histroy of your game, that is if you have played before	"View History"
combo_players	QComboBox	Integer or String options	1–8 options	This will act like a dropdown bar that gives you the ability to choose between 1 and 8 players	"4 Players"
icon_app	QIcon	File path	None	This is just a png that I can place for specific parts of my app such as its taskbar icon	"App.png"
background_image	QPixmap	PNG file	None	This will act as the background of my game	"Trading.png"
history_exists	Boolean	True / False	1 bit	This will basically be used to track wheather a previous game data has been there, so users can see how previous games went	FALSE
setGeometry()	Method	Integers (x, y, width, height)	4 parameters	Its indiacates the size of the window and where it is placed on your screen,	(700, 300, 1000, 1000)
setWindowTitle()	Method	String	None	Will display the title of the game	"League of Traders"
setWindowIcon()	Method	QIcon(path)	None	Gives the game an icon that it will be presented by instead of something that is not linked to the game	QIcon("App.png")
setPixmap()	Method	QPixmap object	None	Loads and applies an image to a QLabel.	label_background.setPixmap(QPixmap("Trading.png"))
setScaledContents()	Method	Boolean	1	Allows images to be resized to fit the screen of the game so they are always 'true to size'	TRUE
clicked.connect()	Method	Function reference	None	Connects a button to a function that will happen once you click it	button_start.clicked.connect(start_game)
start_game()	Function	User-defined	None	Launches the main game and takes you to the next slide	def start_game(): ...
view_history()	Function	User-defined	None	If game data has been stored, it will show previous analytical data	def view_history(): ...

Iterative Development – Stage 1

Code

I have first set up my PyCharm program, using pyqt5, which will contain all the needed files to be able to code this project.



Each of the 3 python files have a crucial function and needs to be there. The main file is where I will hold the code of how I make a single window, I have done this as I believe it will be a recurring function that appears, and having a file showing that code, in my opinion, is handy.

I then have the code file which will be used to show all the main code of my game, this being the first stage which is just implementing a menu which will be expanded on in the second stage.

Finally, I have a test file which will individually test a concept I want to implement such as a push button so it can work.

Iterative Development – Stage 1

```python
import sys #Python module that provides access to specific parameters
from PyQt5.QtWidgets import QApplication, QMainWindow #Imports from PyQt5 so I can develop my GUI

class MainWindow(QMainWindow): #Code used to make the window pop up  1 usage
    def __init__(self):
        super().__init__()
        self.setGeometry(700, 300, 1000, 1000) #Sets the size of the window

def main(): #Subroutine that allows the tab to open and be in that state until users closes it  1 usage
    app = QApplication(sys.argv)
    window = MainWindow()
    window.show()
    sys.exit(app.exec_())

if __name__ == '__main__':
    main()
```

To begin with my code, I have imported the python system module, which will provide me access to parameters that are, system specific. This has been shown in my code by me passing command line arguments into the application. This is needed so sys.argv will be running with my PyQt5 application. This allows the program handle system level arguments and once at app is exited, it is closed without errors.

Then I now imported 2 key classes from PyQt5 library:

QApplication which manages my Gui

QMainWindow which will provide us the window and features that come with it such as widgets and menu bars

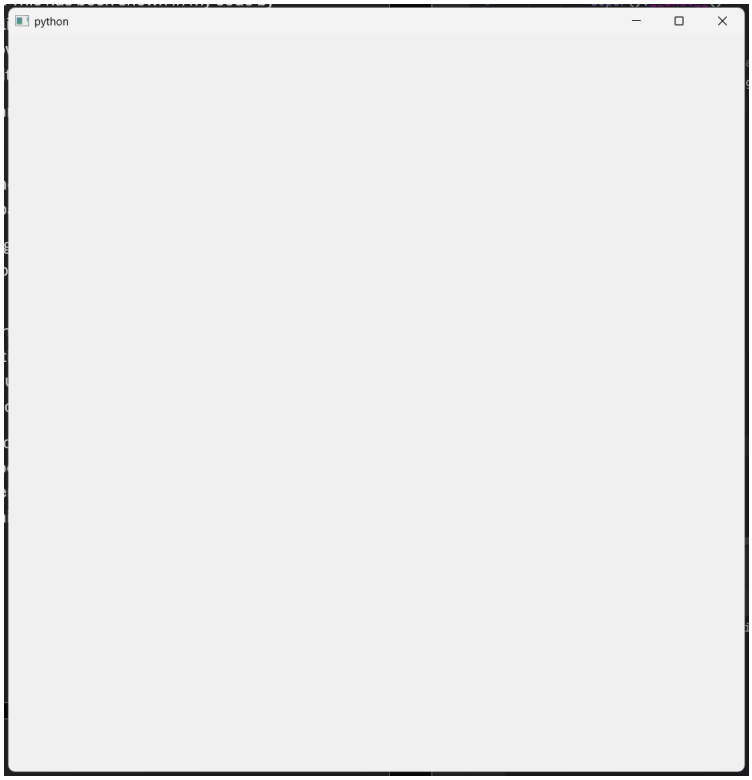
The reason behind using this is because I want my game to be developed with using PyQt5, I came to wanting to use this, because of how intricate you can make your tabs and its ease of use.

class MainWindow, inherits from QMainWindow. This will create my blueprint for my main window so it can allow me to start implementing other sections into my code. I have done this so the window can be shown to the user as it is a necessity. As I will be testing my code, I have set the window to be a size of 700 by 300 by 1000 by 1000

Finally, I created the main subroutine which will allow the program to ran. The code basically allows me to, once my main window is open, create a event loop so the actual GUI can stay active until it is eventually closed when the users want the program to end. Having this main function keeps the program organised and easy to reuse as it is a subroutine, for later parts of the code.

Iterative Development – Stage 1

With all this being programmed you will then be shown this once the program starts:



Next, after setting up my window, I wanted to start to create widgets for my game such as labels or buttons to interact with that have a purpose, however, before I could implement any of these interactive and dynamic elements, I first wanted and felt I needed to create a central widget and layout. This will be used to organise them in a proper manner. Furthermore, it ensures everything inside the window has a structured position that keeps it integrity:

```
import sys #Python module that provides access to specific parameters
from PyQt5.QtWidgets import QApplication, QMainWindow, QWidget, QVBoxLayout #Imports from PyQt5 so I can develop my GUI
from PyQt5.QtGui import QFont


class MainWindow(QMainWindow): #Code used to make the window pop up  1 usage
    def __init__(self):
        super().__init__()
        self.setGeometry(700, 300, 1000, 1000) #Sets the size of the window

        central_widget = QWidget() #Creates a way to contain all widgets in window
        self.setCentralWidget(central_widget) #Tells the program all widgets must go inside main area of window
        layout = QVBoxLayout(central_widget) #Makes layout automatically vertical so it appears top to bottem
        layout.setSpacing(25) #It will automatically have my widgets be spaced out by at least 25
        central_widget.setFont(QFont("Arial", 16, QFont.Bold)) #Give all my widgets the same font so i don't have to automatically change them
        central_widget.setStyleSheet("""
                    QPushButton,QComboBox, QLabel {
                        border: 2px solid black;
                        border-radius: 5px;
                        padding: 5px;
                    }
                """)
```

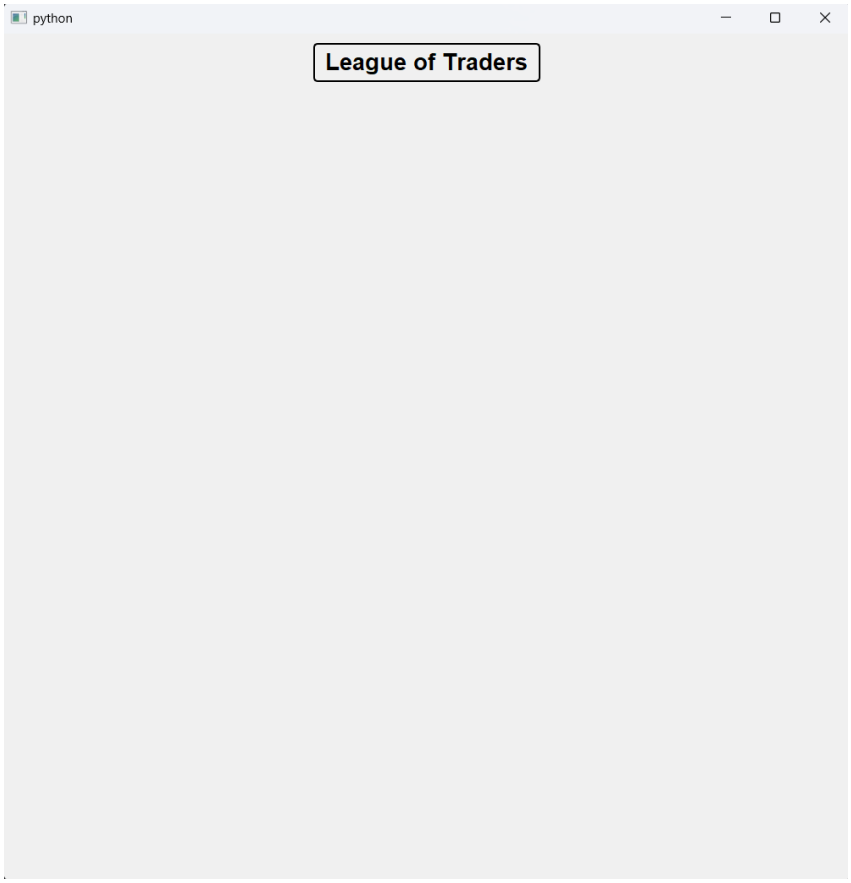
Iterative Development – Stage 1

To begin with, I have new imported another 2 functions from PyQt5.QtWidgets, being QWidget and QVBoxLayout. These functions will be used to later implement my widgets and for my widgets to automatically be assorted in an order such as vertical or horizontal.

I then added the Qfont function from PyQt5.QtGui which is a module that main aspect is to contain functions that assist in the design for a GUI such as a widget's font.

Then I begin to code the central widget. The code in that box is used to make somewhat of a container for all my widgets to be in. Then the latter part of a code is used to automatically give properties to these widgets so that I don't have to do them individually for each one. Such as their font, all being bold and all my widgets having a box around them. Also the spacing between each one being set to a minimum of 25 which can be changed if I want it to

Due to me not having anything to test it on, now I will be placing this code in the test section and creating a label to test it with. After putting in the needed detail I was able to then input a label shown here:



Iterative Development – Stage 1

After finding out that the code was a success, I am now able to start adding my other widgets, such as labels, push buttons, drop down buttons etc:

```
QLabel, QComboBox, QPushButton
```

I first added the 3 necessary functions to produce these widgets, which are labels, combo box (dropdown box) and pushbuttons.

```
from PyQt5.QtCore import Qt
```

Then I added the module PyQt5.QtCore which is currently holding Qt, which will help with being able to use widgets,

```
    title_label = QLabel("League of Traders") #Titles game league of traders
    layout.addWidget(title_label, alignment=Qt.AlignTop | Qt.AlignHCenter) # Align game to fit where i want it to be

    player_label = QLabel("Choose Number of Players:") #Some string to tell the player how many other members are in the game
    layout.addWidget(player_label, alignment=Qt.AlignHCenter) #Alligns it with where i want it to be

    self.player_dropdown = QComboBox() #Inital code for dropdown box
    for i in range(2,9): #For statement to give the ran of vales
        if i == 2: #If it's the smallest value
            text = "2 Player" # Output 2 players
        else:
            text = str(i) + " Players" #Otherewise do the output someone chose plus the string 'Players'
        self.player_dropdown.addItem(text) #Adds each item to the dropdown
    layout.addWidget(self.player_dropdown, alignment=Qt.AlignHCenter) #Alligns the dropdown box just under its string


    start_button = QPushButton("Start Game") #Startgame button
    layout.addWidget(start_button, alignment=Qt.AlignHCenter) #Allign start game

    history_button = QPushButton("View History") #View History button
    layout.addWidget(history_button, alignment=Qt.AlignHCenter) #Allign history button

    start_button.clicked.connect(self.start_game) #Allows the button to be push
    history_button.clicked.connect(self.view_history) #Allowes the button to be pushed

def start_game(self):  1 usage
    print("Game started with", self.player_dropdown.currentText()) #Text once you click the button

def view_history(self):  1 usage
    print("Viewing game history...") #Text once you click the button
```

Here comes the code of my section which is used to be able to be able to access these types of widgets. I first wanted to implement the easiest widget, which is label. It's the easiest as it does not need you to give it any other function but to just stay on the screen. The label was just going to show the game, league of traders, as I need to show the title of my game. This also goes for the next label which was a choose number of player's label.

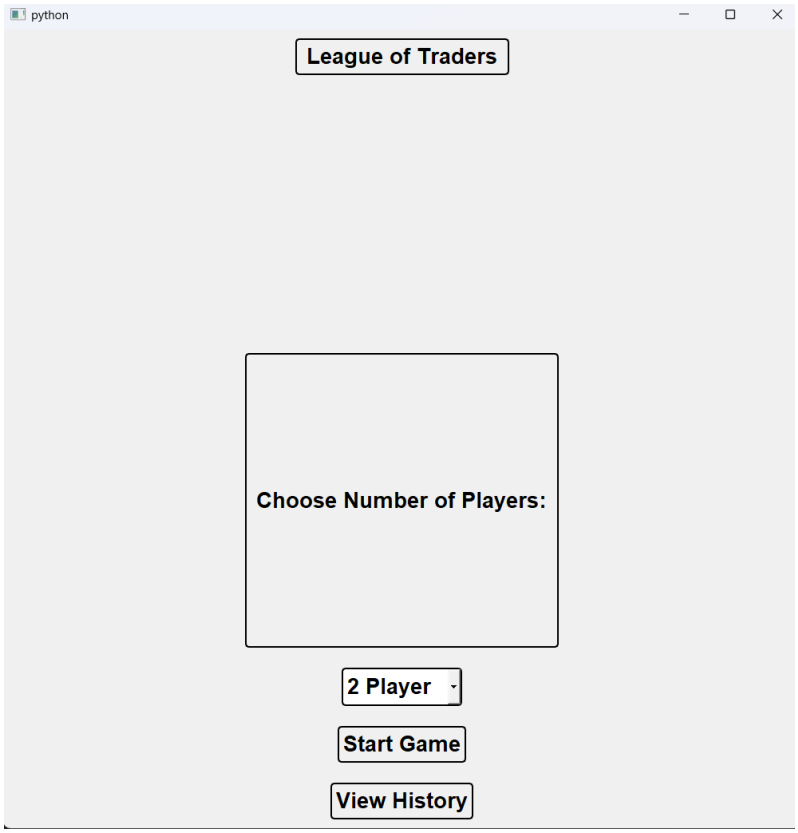
The next widget I added was my combo as it is slightly harder as now, I have to add some python. To add a dropdown box, I first initiated player dropdown making QComboBox now be known as that. After doing so I then went to create a for loop to state the range of players ill be choosing to be presented within the button. I made the range between 2 and 9 and made a if statement staying if it was 2 then you would see 2

Iterative Development – Stage 1

players else present the other players then the final piece of code for that section would add each item to the list and align the box to fit in the window.

Finally, I then added 2 push buttons, start game and history. Once creating them if you clicked start game it would out put game starting then with how many players are playing and if you pushed view history, it would allow you to view the history and say history is being viewed.

Once I tested this code, while it did work, I came across a couple issues. The choose player label box was very big and took over the space. Furthermore, I felt as if the title was not as 'out there' as I would like and wanted it to be slightly bolder than the rest. Finally, it was hard to tell whether my push buttons have been pushed unless you looked at the code, which most people would not do. This is what it currently looks like:

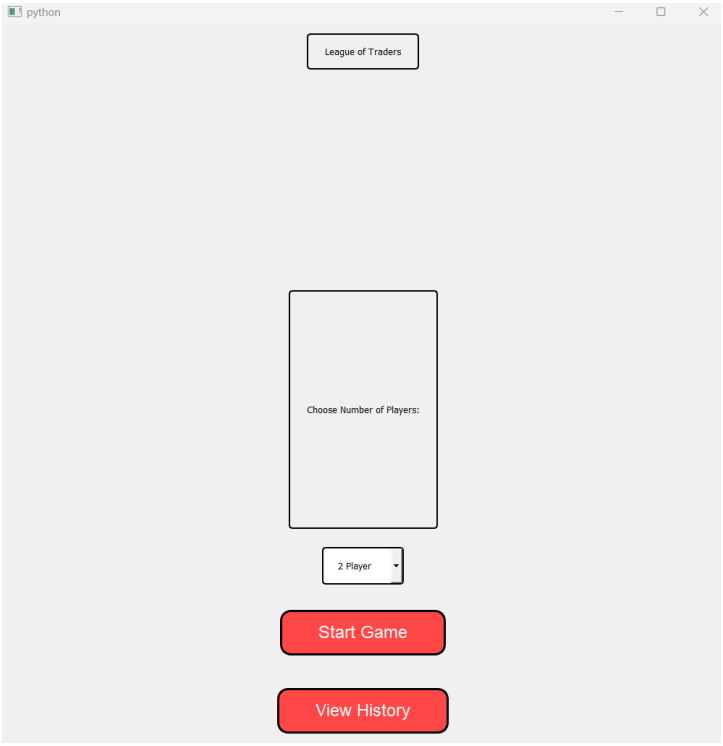


I will now be fixing these mistakes and showing the code, upon completion:

Iterative Development – Stage 1

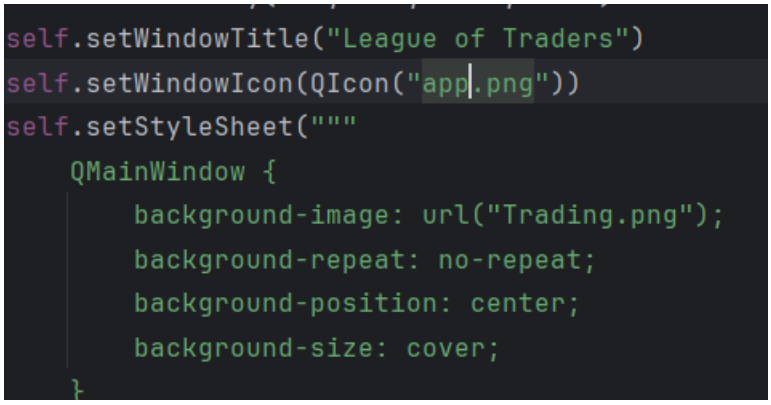
```
startButton = QPushButton("Start Game") #Startgame button
startButton.setObjectName("gameButton") #Assign object name for styling
layout.addWidget(startButton, alignment=Qt.AlignHCenter) #Allign start game

historyButton = QPushButton("View History") #View History button
historyButton.setObjectName("gameButton") #Assign object name for styling
layout.addWidget(historyButton, alignment=Qt.AlignHCenter) #Allign history button

# Apply custom style for all buttons with objectName "gameButton"
self.setStyleSheet(self.styleSheet() + """
    QPushButton#gameButton {
        font-size: 24px;
        font-family: Arial;
        padding: 15px 50px;
        margin: 10px;
        border: 3px solid black;
        border-radius: 15px;
        background-color: hsl(0, 100%, 64%);
        color: white;
    }
    QPushButton#gameButton:hover {
        background-color: hsl(0, 100%, 84%);
    }
""")
```

To begin with, I first wanted to assign both my pushbuttons as gamebutton so once I have to assign them in their style sheet I would not have to do it twice. Then I started to style them in their style sheet and when ever you hover on it, it will change different colours, so the user knows when they are on the button. With all this, my window finally looks like this:



Iterative Development – Stage 1

Finally, to make the game look slightly better, I am now going to be adding a background and icon for my stage one.

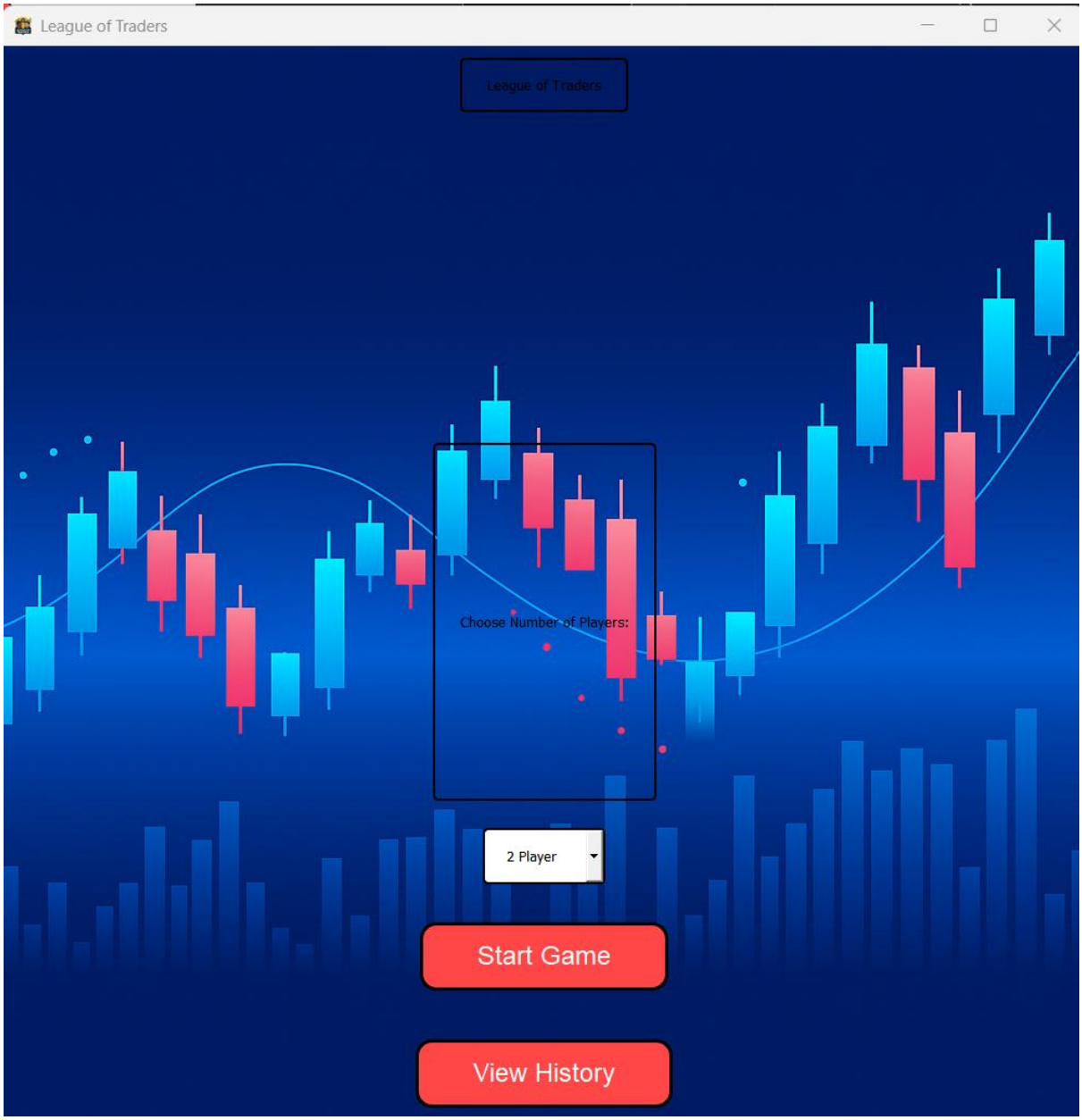


```
self.setWindowTitle("League of Traders")
self.setWindowIcon(QIcon("app.png"))
self.setStyleSheet("""
    QMainWindow {
        background-image: url("Trading.png");
        background-repeat: no-repeat;
        background-position: center;
        background-size: cover;
    }
```

With this code im able to change the name of the app to the name of my game, the icon for the game is no one that I like and now I have a dynamic background behind the game.

This is the finished product of the first stage of the gamne:

Iterative Development – Stage 1



Through the second stage I will be adding a multitude of different things and remaking so of my concepts in the first stage to make them look better, such as some of the sections of this being unreadable, having the icon be shown on the taskbar, the input buttons doing something etc.

Iterative Development – Stage 1

Stage 1 review:

My aims for this stage where:

1) Create a menu screen
2) Crete widgets such as buttons and labels
3) Let you have a previous history screen that can only be clicked if data is found
4) Have the widow data to be readable
5) Once clicking on start game, a new, blank, window will replace the existing one

Score – 4/10

Comments: This stage is seriously underdeveloped. As I will be trying to complete more of it while I go to stage 2, this stage was more of a learning curve for me to overcome as this was the first real experience of using PyQt5 in a large product. So while I am very happy with the results I was able to output while getting over this curve and going on with this experience, I do believe this stage, I should have been able to do more.

Stage 2: Basics

Stage 2 will be expanding on this work and going onto the next slide (GUI design framework) of my game. I will most likely during this stage, cover the next 2 slides of my game as they will be quick to do and they both are linked to each other until the slide 4 (stage 3) which will be a very large stage and take me the longest, other than implementing the game.