

Connect 4 With Machine Learning

<Date>

Table of Contents

[Analysis 4](#_Toc66457281)

[Introduction 4](#_Toc66457282)

[End User Identification 4](#_Toc66457283)

[Questionnaire 5](#_Toc66457284)

[Proposed Idea 6](#_Toc66457285)

[Identification of Programming Language 6](#_Toc66457286)

[Numbered Objectives 7](#_Toc66457287)

[Prototype 12](#_Toc66457288)

[Documented Design 18](#_Toc66457289)

[Program Overview 18](#_Toc66457290)

[Logging System 20](#_Toc66457291)

[Connect 4 Game 20](#_Toc66457292)

[Neural Network 21](#_Toc66457293)

[Input Format 21](#_Toc66457294)

[Output Format 22](#_Toc66457295)

[Network Structure 22](#_Toc66457296)

[Weights and Biases 22](#_Toc66457297)

[Feed Forward Algorithm 23](#_Toc66457298)

[Training Data 24](#_Toc66457299)

[Backpropagation 24](#_Toc66457300)

[File Structure 26](#_Toc66457301)

[Account Data File 27](#_Toc66457302)

[Previous Games File 28](#_Toc66457303)

[File Encryption 29](#_Toc66457304)

[File Checksum 30](#_Toc66457305)

[Graphical User Interface (GUI) 31](#_Toc66457306)

[State System 31](#_Toc66457307)

[Main Menu State 32](#_Toc66457308)

[Confirm Exit State 32](#_Toc66457309)

[Accounts State 33](#_Toc66457310)

[Create Account State 34](#_Toc66457311)

[Delete Account State 35](#_Toc66457312)

[Leader Board State 36](#_Toc66457313)

[Play Game State 38](#_Toc66457314)

[Connect 4 Game State 38](#_Toc66457315)

[Game Won State 40](#_Toc66457316)

[Game Draw State 40](#_Toc66457317)

[Object Oriented Programming 41](#_Toc66457318)

[Technical Solution 43](#_Toc66457319)

[Introduction 43](#_Toc66457320)

[Contents 43](#_Toc66457321)

[Premake 43](#_Toc66457322)

[Tools 45](#_Toc66457323)

[Entry Point 47](#_Toc66457324)

[Systems 48](#_Toc66457325)

[Program 59](#_Toc66457326)

[States 64](#_Toc66457327)

[Confirm Exit 65](#_Toc66457328)

[Main Menu 67](#_Toc66457329)

[Accounts 69](#_Toc66457330)

[Create Account State 76](#_Toc66457331)

[Delete Account State 79](#_Toc66457332)

[Play Game 83](#_Toc66457333)

[Connect 4 89](#_Toc66457334)

[Game State 89](#_Toc66457335)

[Accounts 94](#_Toc66457336)

[Game Over 99](#_Toc66457337)

[Leader Board 102](#_Toc66457338)

[Graphics 110](#_Toc66457339)

[Button 110](#_Toc66457340)

[Text 114](#_Toc66457341)

[Text Input 115](#_Toc66457342)

[Testing 120](#_Toc66457343)

[Evaluation 121](#_Toc66457344)

# Analysis

## Introduction

The problem area that I am interested in is Machine Learning[[1]](#footnote-1) which is a large topic in computer science research. I am planning on creating a machine learning algorithm which will train a neural network[[2]](#footnote-2) to act like a human at playing the board game, Connect 4. The main reason I chose this was because I watched a video on machine learning[[3]](#footnote-3) and was instantly mesmerized. I decided that I wanted to learn how to make something similar and chose Connect 4 as the game because it is quite simple and can demonstrate a learning AI quite well.

Machine learning algorithms have been around since the 1950s but has only recently taken off due to the rise of the internet and masses of data being transferred daily. Companies such as Google and Facebook are using machine learning algorithms daily to provide a better user experience, better than any human could do on their own. Many tools have been made to allow machine learning to be used by many people without much background in the topic itself. One of my inspirations is Google’s DeepMind[[4]](#footnote-4) AI which taught itself to beat a world-champion at the Chinese game of Go[[5]](#footnote-5).

Machine learning is based on mathematical ideas to adjust parameters such that a cost function is minimized. In my case of Connect 4, the cost function will be how different the AI’s choice is to the human’s choice so that when training, the AI will minimize this and start imitating the human better. I will derive all mathematics behind the algorithm as it is quite complex. The implementation is simple but, can get quite complex for efficient code due to a few tricks in organising the data in matrices and using matrix math to do calculations.

I am planning on only using a single Machine Learning algorithm for my final product as the number of tweaks to parameters I can make to the algorithm is massive. I should be able to adjust the algorithm so that I can get the most performance out of the single algorithm. If I finish the algorithm and realize that I there are not enough parameters to tweak, I can add another algorithm, but this will add a lot of complexity and take a lot of time as a large rework will have to be made. This is because I would have to allow a modular kind of program where the number of algorithms available is not hardcoded, but more can be added at any time.

## End User Identification

As this project is meant to be used by multiple people, I have asked both my friend, Max, and my Computer Science Teacher, Mr Willans, to be my end users. Both have studied Computer Science and understand what is required to make a program. Therefore, asking them to be my end user will allow them to choose the right features to be put into my project without over complicating certain tasks.

After speaking with both end users, I have found that they have had no interaction with Machine Learning before and would not know where to start if they wanted to. This will allow me to create a program to demonstrate the power of machine learning and test, if enough is demonstrated, on someone new to the topic.

### Questionnaire

|  |  |
| --- | --- |
| **Question** | |
| **Max’s Response** | **Mr Willan’s Response** |
| Who are you, what do you do and are you interested in machine learning? | |
| My name is Max, I am a student at Ousedale School, and I am interested in Machine Learning. | Martin Willans, Head of Computing and Digital Applications at Ousedale School. I am interested in machine learning because it demonstrates the development of artificial intelligence in computing |
| What do you plan on getting out of using this project? | |
| To understand how an AI can learn using Machine Learning to play Connect 4 | I would like to be able to use the project to demonstrate artificial intelligence to students and use it as a simple and fun game |
| What features are you looking for? | |
| An easy-to-understand representation of how the AI is progressing | I would like some visual demonstration of the data that is being updated to see the machine learning algorithm working |
| Should the program be interactive or passive (the user only plays the game, and everything works behind the scenes)? | |
| The program should be passive as I am only interested in seeing the AI develop over time | I think that the user should play the game and the rest of the process should happen I the background |
| How do you want to interact with the program? | |
| I would like to be able to interact through a graphical interface with buttons and text boxes | There should be a visual element so that the player can place the piece in the right column. You should also be able to log in and export your data to a file to be transferred to another computer |
| How should the program look? | |
| I am not interested in too much detail in the graphical interface. Just enough so that it looks good but can also express all the information needed | Only simple graphics are required to show the gameplay also, the machine learning is the most important aspect, so graphics are not that important |
| Do you want the program to run on multiple operating systems e.g., Windows and Mac? | |
| I only own a Windows PC so would not benefit from having the program run on Mac | The program only needs to be able to be run on a Windows PC |
| Should the user account be transferable such that they can use their account on another computer? If so, elaborate | |
| I am only interested in playing Connect 4 on my own computer but would be nice to be able to play against someone else’s AI without having to train from scratch | It would be good if the user data were stored in a file that could be exported and moved to another computer to be imported and played with on that computer |
| What quality of life additions would you like to see in the project? | |
| An easy-to-use interface, multiple accounts so I can play against different AI’s I have trained, and be able to watch an AI vs AI game | I would like to see some data being saved about the user such as number of games played, and games won |
| Any extra words? | |
| N/A | N/A |

## Proposed Idea

My final plan is to create a Connect 4 application that can demonstrate Machine Learning in an interactive way. I will do this by allowing each user to create an account and be given an AI to ‘teach’. As the user plays Connect 4 against another player or AI, the AI that the user ‘owns’ will learn from the game being played and eventually begin to imitate the user. This imitation is personal to each user and allows users to relate more to the machine learning application remain engrossed in the program.

## Identification of Programming Language

Research about programming languages came from Ignite Digital[[6]](#footnote-6) and Raygun[[7]](#footnote-7).

* **Python** – Python is a very user-friendly language with many libraries that help development. It does supports Object Oriented Programming and has a lot of Quality-of-Life features built into it however, it is not the most performant language when coding at a basic level. I have a small amount of experience in Python but will have to research how it handles object-oriented programming.
* **Java** – Java, like Python, has an abundance of open-source libraries useful for certain applications (including Machine Learning). Again, it is not the most performant language but, has much more control over lower-level ideas than Python does. I do not have much experience at all in Java, but the syntax is extremely like C/C++ meaning it will not take much to learn.
* **C/C++** – Both C and C++ are extremely low-level languages allowing for deep control of hardware to gain large performance boosts. C++ comes with its own libraries (STL[[8]](#footnote-8) – Standard Template Library) which are highly performant and almost never ignored for another library. Only C++ is Object Oriented out of the two meaning C will require a bit more work for certain areas. I am extremely familiar with C++ as I use this language for almost all my projects.
* **C#** – C# is an exceedingly popular language for development as it allows for low-level control like in C but also is much more user friendly like Python. C# is object oriented which is useful and certain programs such as Unity use C# as their language of choice meaning Graphics with C# is a lot easier than with other languages. I have almost no experience with C#, but the syntax is like C/C++ with simple pieces of code but changes a lot with function definitions and similar ideas.
* **Visual Basic** – Visual Basic is a simple and pseudocode like language meaning it is very user friendly with how it works. However, this causes some problems when it comes to consistency as some differences from other languages become apparent when using arrays as many times Visual Basic handles these differently and *off-by-one[[9]](#footnote-9)* errors occur are quite common. Visual Basic is the language we have been learning at School since year 10 meaning, I am quite familiar with it but due to its awkward Object-Oriented Programming it will require some learning.

In conclusion, after researching different programming languages, I have decided to use C++. This is because of its low-level control and great Object-Oriented Programming. The only problem with C++ is the use of Graphical Libraries, this is because as C++ is a language with a lot of control, the libraries it comes with also contain a lot of complexity meaning it is difficult to use them to do simple things. I would have chosen Python due to its user-friendly syntax however, I do not have a good enough understanding of how the Python compiler works in terms of multiple files to be able to create a large project with it.

## Numbered Objectives

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Main Objective | | Explanation | Success Criteria | | Sub-Success Criteria | |
|  | Create an AI that can learn from experience from human actions | A human will complete a task and the AI will try to replicate that task as best it can. As the human does more tasks the AI needs to learn to do all tasks like the human |  | A working neural network must be made |  | Be able to feed in an array of floating-point values and receive an array of floating-point values as an output |
|  | The neural network must have a runtime defined number of layers and number of neurons in each layer |
|  | Be able to apply different activation functions to different layers |
|  |  | Allow the initial values for weights and biases to be chosen with purpose |
|  | The neural network must be standalone meaning it has no relation to Connect 4. It is only used for that purpose |
|  | A machine learning algorithm must be in place to allow the neural network to adjust its weights and biases to learn from experience |  | Training data must be collected and stored so that the neural network can learn from it later |
|  | A cost function must be made to measure the deviation the neural network is from the human |
|  | A gradient descent algorithm must produce a change to all weights and biases from a single training example to minimise the cost function |
|  | An average over all training examples for changes to weights and biases must be calculated and applied to the neural network. |
|  | Training data must be stored in a file so that the neural network can minimise the cost function based on these training examples multiple times |  | A single training example consists of the inputs given to the network and the expected outputs |
|  | An entire Connect 4 game is built up from many training examples which all must be stored in a file |
|  | Multiple Connect 4 games must be stored to increase the number of training examples the neural network has to learn from. The number of Connect 4 games must be variable |
|  | The training data must be stored in a binary format so that it can be reconstructed and used even after the program has been closed |
|  | The neural network needs to be serialised and stored in a file so that it can be used when the program is run again |  | The number of layers and number of neurons in each layer must be stored in a file |
|  | The weight matrices and bias vectors must be stored in the same file as the size of the neural network |
|  | All information about which activation functions are being used must be stored in a file |
|  | The neural network must be able to be reconstructed from the serialised data and produce the same outputs as before being serialised |
|  | Create a working Connect 4 game | The Connect 4 game will take user input and display the current state of the board. It will also determine whether the game has been won and by who |  | The board needs to be displayed on the screen |  | The program needs to switch to a Connect 4 state which contains all instructions about the game |
|  | An empty Connect 4 board needs to be displayed as an image |
|  | Counters need to be displayed in the right positions on the board with the right colour |
|  | Both users’ names need to be displayed and which players turn it is currently |
|  | The users need to be able to place a counter in a specific column |  | Only the user whose turn it is currently should be able to place a counter |
|  | The user should be able to select a column by clicking on it and have a counter be placed in the lowest available place in that column |
|  | The game should check whether a user has won, or the game has been drawn |  | After each counter has been placed, a check should be done to see if four counters in a row have the same colour |
|  | If the game has been won, a game win state should be selected and displayed |
|  | A check should be done to see if the game has been drawn |
|  | If the game has been drawn, a game drawn state should be selected and displayed |
|  | Create a working account system | The user will have to create an account with details about them and this account will contain the AI that learns from that user |  | The user must be able to create an account |  | The user should input a username and password |
|  | If an account already exists with the inputted username then it should ask the user for a username again |
|  | The account should be created with a new AI and all details related to the user |
|  | The user should be able to delete their account |  | The user should be asked to input a username and check if that user exists |
|  | If the user exists, a password should be requested and if that exists then the account needs to be deleted |
|  | When deleting the account, the file containing the information needs to be removed |
|  | Each user should be able to access their account to play Connect 4 |  | When using an account both the username and password must match an already existing one |
|  | All data should be loaded into an object so that the AI can be utilised to either play the game or learn from one |
|  | The account should be able to be saved to a file |  | The username should be stored in the file and will be used as the name of the file itself |
|  | The password should be encrypted and stored in the same file as the username |
|  | All data about Connect 4 should be stored in the file such as number of Wins/Losses |
|  | The AI’s neural network needs to be stored in the file in a format such that it can be reconstructed |
|  | Create a pleasing graphical interface | Allow the user to interact with the program through a graphical interface using buttons and text boxes |  | Create a modular system where objects can be created and placed on a window |  | Allow the creation of an object such as a button or image and place it anywhere on the current screen |
|  | Allow the user to use the objects created to interact with the program such as change state to the game state |
|  | Be able to load in different resources such as textures to change the look of the program at runtime |  | Allow an image file to be changed and have the program reflect this change by using that image file |
|  | Load files from a folder which are used to describe how the program is to be laid out |
|  | Allow the user to view a leader board | Create a state where all users are displayed along with their total wins |  | Display all users in a list and allow them to be clicked on to view more information |  | Sort all users by a heuristic chosen by the user including alphabetical, number of wins, number of losses |
|  | When a user’s name is clicked, display all information about that user |
|  | When first using the leader board, all users must be loaded from their files and put into account objects |
|  | Leader board account objects should be stripped down as to save memory such as not including the AI itself |
|  | Allow the user to search for a username |  | Use algorithms such as Levenshtein Distance[[10]](#footnote-10) or Soundex[[11]](#footnote-11) to match similar usernames |
|  | Sort the list of users alphabetically when searching as usually the user will want to find one specific user |

## Prototype

I decided to prototype this project to make sure that it is simple enough to demonstrate Machine Learning quickly to keep the user entertained. I have chosen to remove a few features that are unnecessary for this task such as a neat graphical interface and leader board. This prototype will also be using an external library to speed up the process of development as it is just to allow the user to watch a learning algorithm take place.

The main take away from this prototype was the massive help that creating a state system had on development. This state system allowed me to create states and push them onto a stack which was used to determine where the user is in the program. This was also an easy way to deconstruct the program into sections and finish those programs before starting on others and see that part work through the state system. I will almost definitely be using a similar state system in the final project with some revisions to fix a few problems I had with it.



This is the entry point into my program which demonstrates how the state system should work. A new main menu state is created which is push onto the top of the state stack. Then a loop is started which runs until all states are removed. This would only happen if the ‘exit’ option were selected from the main menu state. If a state does exist on the stack, it is run which uses polymorphism[[12]](#footnote-12) to change what happens depending on what is currently the top state.



This is the definition of the ‘Neural Network’ class. Each user contains a single Neural Network[[13]](#footnote-13) and is only able to interface with it through its constructors and three methods: Feed Forward, Train and Serialize.



This Feed Forward method is a simple, vectorised implementation of forward propagation which is how data is manipulated in a neural network. The ‘Eigen’ keywords used here are to do with the Matrix library I am using for this prototype which allows easy Matrix-Math to be performed such as demonstrated with the line , which is a matrix-vector multiplication operation followed by a vector-vector addition operation.

I will be creating my own matrix math library in the final program to reduce the size of the executable because I only use a tiny amount of this library.



The Train method is where the updating of all weights and biases occurs depending on the output from the backpropagation method.



This is the program running on the Main Menu State. If the user inputs one of the options shown above, the Main Menu State will interpret this input and either push a new state depending on what was selected or pop the Main Menu State off if ‘exit’ was chosen. This causes no more states to be on the stack and the program to exit.



If ‘Play’ is selected on the Main Menu State, the user is asked to input details about who is playing the game including if they want the human or the AI to play.



The game is then repeated until someone wins. After the game is over, the AI will update using the previous game and execute the learning algorithm. This causes the AI to slowly change how it plays over time.

This prototype has shown me that the learning algorithm I am using requires a lot of training data to work effectively. In this prototype I am only using the previous game which is on average 24 moves per game[[14]](#footnote-14). This is nowhere near enough training data for the algorithm to learn quickly. After many games with the AI, I did find that it was changing very slightly but was mainly choosing the same column the entire game until it was full. This suggests that the neural network is not effectively using the input data it is given and just producing the same results each time.

Due to this problem in the learning algorithm, I have decided to save the past couple of games in the file along with the data about the player. Then when I come to train the algorithm, it can use the new game just played and the previous games to reinforce what it has already seen. In a perfect world, I would be able to train on all training examples every time however, without a custom compression algorithm, storing all past games would take up a lot of storage and I would prefer it if I could keep the data related to the user as small as possible.

Even though this is a prototype, just seeing that the AI changes what it does after each game is interesting and I believe that the final program can be extremely fun and engaging for new users.

# Documented Design

## Program Overview

As my program is quite large and complex, I will need multiple diagrams focused on different parts of this project. My overall program is amazingly simple and only contains a couple of features but the functionality that those features contain is exceedingly difficult to understand without diagrams. I will be making a Flow Chart[[15]](#footnote-15) for my program which shall show the overall idea of how the user will interact with it and what they will be expecting as a response.

There is too much detail within sub-problems in this program so the first Flow Chart I will be making will be a very abstracted version only showing the basics of what the final product will do. Items like implementation details and specific function calls have been omitted to fit the Flow Chart onto a single page.

I have used LucidChart[[16]](#footnote-16), an online diagram and visualisation creator, to create my Flow Charts as it is a powerful and simple editor meaning I can create complex Flow Diagrams in as little time as possible. The only problem with LucidChart is that the free version only allows for 60 objects to be created on a single document. I have run into problems with this before but usually this was because I was trying to fit everything into a single diagram.



This Flow Chart shows the entire program and how it should be used. Most individual process and input object will require their own Flow Chart as a description of how they will be implemented. This is most prominent in the “Play Connect 4” Process and “Current Player Place Counter in Column” decision as the player could either be a user playing or an AI which will need to be carefully taken care of when choosing the column to place a counter.

## Logging System

A common technique used to debug programs is a Logging System[[17]](#footnote-17). This is a system in which the programmer can print information about the program as it is running without affecting the program itself. Information can be removed from the final distribution build as it will slow the program down but during debugging it can be helpful to print any information, warnings and errors that appear in the program. I have decided to use a Logging System in my program so that development is sped up due to errors being caught quicker. I only need a simple Logging System which is why I have decided to create my own with simple logic. Libraries exist for many languages which contain a much more featureful Logging System than I can create however, I do not need to use all these features which is why I am not going to use a library.

The Logging System needs to be able to accept a string with a certain level. This level can include ‘Trace’, ‘Warn’ and ‘Error’ which determines if the program should print that message. If the Logging System is set to ‘Trace’ level, then any messages with level ‘Trace’ and above should be printed. I can add other levels to the system to allow for more detailed messages to appear which I may need to do if the program grows too large. The Logging System should also not be present at all in the distribution build of the program meaning it should not be included anywhere in the project. One way of doing this is a handy C++ feature called Pre-processor Statements[[18]](#footnote-18). Pre-processor statements are pieces of code which are evaluated at compile-time meaning they are changed before being compiled allowing code to be removed and not even seen by the compiler. I can use these pre-processor statements to include the Logging System only on debugging builds of the program and remove it in release builds.

## Connect 4 Game

The game of Connect 4 is quite simple, it consists of two players taking turns placing coloured counters into 7 columns by 6 rows board. The counter can only be placed at the lowest available position in each column which is usually denoted by the board being upright and the counters falling down their respective columns due to gravity. I will be abstracting away the idea of gravity and implementing the “falling” idea as an algorithm as a gravity simulation would be too complicated and unnecessary for this project.

The main obstacle I will have to overcome in this Connect 4 game is that both players could be either a Human or AI. This means the way I will have to get that players column choice will differ depending on what they are. This is a prime example of when polymorphism[[19]](#footnote-19) is most useful and powerful. Polymorphism is the idea of using inheritance (with respect to object-oriented programming) to allow an object to take multiple forms. All objects that inherit the same base class will behave similarly with the same methods as in the base class. These inherited methods can be overridden to complete a different task but with the same function header.

In the case of Connect 4, I will be creating a *Connect 4 Player* class with the method *Get Column Choice*. This method will return the players choice of what column to place the counter in. This will differ depending on whether the user is a human or an AI which is why this method will be marked as *virtual[[20]](#footnote-20)* which is a C++ keyword declaring that a method is overridable. Two new classes will be created, *Connect 4 Player Human* and *Connect 4 Player AI*. Both will inherit the base class *Connect 4 Player* and will override the method *Get Column Choice* allowing each of these objects to perform differently when calling the same method.



This pseudocode shows the basics of how the connect 4 game will be created. It consists of alternating turns and placing a counter in each column until the game is over. This closely resembles how connect 4 is playing in person which is a good metric on how well the program will represent the game.

## Neural Network

A neural network is a representation of a real neurological network. This means that it can simulate actions close to what a biological neural network can. Its definition is simple which allows us to manipulate it and create efficient algorithms that can be performed on it such as learning algorithms. Neural networks are built up of neurons which can hold a single floating-point value which are manipulated to another floating-point value to be stored in another neuron. These neurons are placed in layers where the first layer is the input layer, and the last layer is the output layer. All layers in between are called hidden layers and are used to let the network compute more complex ideas.

The use of a neural network is just to turn input values into output values. Due to this, we can assume that a neural network is just a function. This function is how the program will use the network as the input we decide to give it will allow it to produce an output which we will interpret as a column choice for Connect 4. The input we will be giving it is the current state of the Connect 4 Board as this information is enough to produce a reasonable result for the column choice. The format in which we give the network this information is especially important as a badly optimized format will increase the time it takes to train producing a worse experience for the user.

### Input Format

One idea for the format of the input is to have each neuron represent a single position in the board and have its value represent the state of the position (empty, enemy counter, your counter) perhaps “-1, 0, 1” representing each of the three possible states. This is a simple format and makes sense to humans however the neural network will see this and assume that enemy counter is halfway in between empty and your counter as 0 is halfway in between -1 and 1. This does not make much sense which could cause the network to perform badly. A similar approach could be to arrange the states as such “-1: Enemy Counter, 0: Empty, 1: Your Counter”. This still has the advantage of being simple and makes a bit more sense than before but not completely.

A different approach would be to create three neurons for every position on the board where each group of three neurons only has a single neuron containing a value which represents the state of that position. This format would remove all uncertainty about what the values mean but this does triple the number of neurons in the input layer causing the network to perform worse and require more training data.

Overall, I will try both methods and see which one performs better using a heuristic produced by the network. I assume that the best will be the second however, this will only be if there is a lot of training data as the number of neurons has increased threefold.

### Output Format

The output of the neural network is much simpler as it follows the same idea as the input. We need to network to produce a column choice for the connect 4 game in an efficient way. One idea would be to just have it produce a single number where 1 corresponds to the first column, 2 corresponds to the second column etc. This is like the first method I suggested for the Input Format meaning there is a similar second approach. In the case of the output, the second approach will be much better, and we will have to distinguish between seven different choices and having seven output neurons each corresponding to a column choice would be the better option. In practice, we will take the outputs and find the position of the highest value which will correspond to the networks choice and that position will be chosen. If that position is invalid, such as the column is full, we will choose the second highest value and continue.

I have been following a YouTube series on this by 3Blue1Brown[[21]](#footnote-21) where he uses a similar format for the output of his neural network which is another reason why I chose to output values this way.

### Network Structure

The number of layers and number of neurons in each of those layers heavily impacts the performance of the network and choosing them correctly can improve the speed and accuracy of the network greatly. The restrictions we have are that the network has to have either 42 or 42×3 input neurons, depending on the format we choose, and 7 output neurons. The number of hidden layers and the neurons within those hidden layers are what we can change. I have decided to go with two hidden layers both with 16 neurons in each. This is a somewhat arbitrary choice that can be tweaked in the future if the network does not perform as expected.

Too many neurons in the hidden layers can cause a problem known as overfitting[[22]](#footnote-22) where the network learns to fit the training data too well causing it to perform badly on data it has not seen before.

### Weights and Biases

The weights and biases are the most important bit of the network as they are what determines what the network will do. Choosing these parameters manually is way too much work for anyone to do which is why we use an algorithm called Backpropagation. This algorithm adjusts these weights and biases to better fit some training data. Backpropagation does not do everything though. We need to choose good initial values for the weights and biases. The method I will be using the choose these initial values is mainly randomization with some bounds. These bounds will be determined by the number of previous neurons and number of next neurons.

The weights and biases will be randomly set to a value in the range [-ꜫ, ꜫ]

Where

### Feed Forward Algorithm

The Feed Forward Algorithm is the method of turning the input values into output values using the weights and biases within the network. This algorithm is extremely simple and even with this simplicity can produce quite complex outputs.

We can visualize the network as a collection of neurons, grouped into layers where every neuron in one layer is connected to every neuron in the next layer. These connections are our weights.



To calculate the value of a single neuron in one layer we take each neuron’s value in the previous layer and multiply it by the corresponding weight, sum these together and add the bias. This will give us a value which will be used in the Backpropagation algorithm. This value is not the new activation for the neuron, we need to use a normalization function to make sure the activation remains in the range between 0 and 1. For this we use the sigmoid function[[23]](#footnote-23) defined as:

Once we have applied the sigmoid function to our value we now have the activation for a single neuron in the next layer. We can repeat these steps for all neurons in the next layer so that we can repeat this same algorithm for the next layer and Feed Forward the input values until we reach the final layer, and the activations of that layer can be used as our output.

This algorithm would be slower than expected if we applied it as described. Luckily, we can simplify the algorithm into a series of matrix-vector multiplications. If we represent a layer’s activations as a vector, we can multiply these by a matrix containing all the weights so that the vector we get from this multiplication has the right dimensions for the next layer. We can then use element wise addition on another vector containing all the bias values which will in turn, give us a vector of values to which, we can apply the sigmoid function giving us a vector of activations for the next layer.

### Training Data

Training data is the set of data used by the backpropagation algorithm to adjust the weights and biases of a neural network to produce correct inputs on a set of input data. The training data I will be using are the previous games that have been played by the human. As I will be using supervised learning[[24]](#footnote-24), the training data needs to be labelled meaning that I need to provide the input along with what I would expect the network to output with that given input. Using this information, the backpropagation algorithm can tweak the weights and biases to produce the correct output more often on the training data. If the training data is good, then the network should perform better on situations it has not seen before giving the illusion that it is learning.

A problem that occurred in my Prototype was that the network was learning way too slow due to the lack of training data I had. Due to this, I have decided to store lots of previous games in a file which can be re-used as training data for the network to allow it to reinforce on what it has already learned.

The training data needs to be in the same format as what the network will usually accept however this does not mean it needs to be stored in the same way. We can store the training data as a simpler, compressed format and reconstruct the actual training data from this which will save time and space.

### Backpropagation

The backpropagation algorithm is a method to tweak the weights and biases within a neural network so that they produce a lower value for the cost function. The cost function is a function that uses the neural network along with the training data to produce a single number representative of how badly the network is performing. As the aim of backpropagation is to lower this cost, the network will perform better when this algorithm is run. To put in mathematical terms, the backpropagation algorithm calculates the derivative[[25]](#footnote-25) of the cost function with respect to each weight and bias individually. These deltas are what we use to adjust the weights and biases to allow the network to ‘learn’.

#### Cost Function

The most important thing we need for the backpropagation algorithm is a cost function. This cost function will use the current state of the neural network along with all the training data and produce a single real value which can be used as a measure of how well the network is performing. To produce this single value, we will sum across all training examples the cost of a single training example and then divide by the number of training examples (which I will denote with the letter ). To calculate the cost of a single training example we will need to calculate the difference between what the network outputted versus what we expected it to output – the training data’s output. To make sure that all values are positive we will take the squared difference between the output and expected output. As we have seven output neurons, we will sum across all of these to produce a single real value for the cost of a training example.

Put in mathematical terms:

In this formula represents the jth value of the ith training example’s output. The same idea can be used for where is the output activations.

#### Algorithm

If we take a simple network with a single neuron in each layer, we can describe the basics of the backpropagation algorithm and have it scale quite nicely to more neurons in each layer. If we also assume that we only have a single training example, then both the summations in the cost function are not needed reducing the complexity greatly.

We can expand the activation to get it in terms of its components:

If we want to figure out how much to change by to reduce the cost function, we take the derivative of the cost function with respect to that weight. Using the chain rule[[26]](#footnote-26), we can change this into a form we can work with.

If we work through each term and calculate the derivatives and multiply them together, we will get how much we should change by to increase the cost function. As we want to decrease the cost function, we just take the negative of this value. This is a similar process for calculating the derivative of the cost function with respect to and . Using this term we can keep iterating back through the network to calculate changes to other weights and biases in previous layers. This is where the backpropagation idea comes from.

These are the actual values for the derivate for weight, bias and activation used in the backpropagation algorithm.

Derivate of cost with respect to weight:

Derivate of cost with respect to bias:

Derivate of cost with respect to previous activation:

#### Layers with More Neurons

As the derivatives shown before are for a simplified version of a neural network with only a single neuron in each layer, when more neurons are introduced the derivatives change slightly. Derivatives with respect to the weights and biases do not change that much however, the derivative with respect to the activation in the previous layer does change as it can influence to cost function through multiple different paths. Due to this, the derivate needs to be a sum over all those possible paths.

This extra information surprisingly does not complicate the derivates that much even though it seems like it would.

#### Implementation

The entire algorithm shown above will create changes for all weights and biases in the network for all training examples. We need to make sure to not update the network after each training example as this would cause jittery motion in the algorithm. Instead, we store each iteration in a list of matrices and vectors to be used once all training examples have been used. After all training examples have been iterated over, we can finally update the weights and biases using these deltas.

When implementing this algorithm, many optimizations are made to simplify the process. The main one used is reducing all operations into matrix-vector math operations as seen before in the Feed Forward Algorithm. These simplified operations turn a complex algorithm into only a few simple matrix operations.

## File Structure

As my program will require data to be saved to a file and reconstructed when the program is run again, an easy to understand and efficient directory structure must be used. The only data that needs to be stored is data related to user accounts however this includes a lot of items. Therefore, I am planning on creating a folder for each user which holds all information about that user. This also allows for users to copy their folder and transfer between devices allowing their account to be used on multiple computers.



Figure - Directory Structure

Figure 1 shows the directory structure that I am planning on using for my final program. The “res” folder is just to hold any resources that the program will use such as textures or any other run time resources. The “saves” folder is where all data relating to the accounts will be stored. Each user will be contained within their own folder with the name of that folder being their username. This allows the program to easily look at all folders and navigate to the correct one without too much complication. The extensions of the files in the user’s folder are completely arbitrary as the program is only going to read them as binary files containing data and no other program will need to open these files.

The complexity of this file structure is finding a way to store all data relating to the user in the appropriate files in an efficient way. This means reducing the file size as much as possible and reducing the time required to read and write to these files.

### Account Data File

In relation to the account data file, the information that must be saved is the following: Username, Password, Date Created, Number of Human Wins, Number of Human Losses, Number of AI Wins, Number of AI Losses, and the Entire State of the Neural Network. A lot of this data has different types meaning that a custom file must be created. I will be using already existing file structures as a reference. The most prominent feature in files is the use of metadata and a header. The metadata is any information relating to how the file should be interpreted. In the case with an image file the metadata[[27]](#footnote-27) will contain information such as bit depth and resolution.

The information stored in my metadata section of the file will be structured as follows:



All parts will be stored using 4-byte unsigned integer values. This structure will allow for all remaining information to be read using the correct format.

During my prototype, I intertwined the metadata in with the header itself meaning that metadata related to the neural network was only read when reading the header for the neural network. The only problem with this was that it was hard to debug what was going wrong as not all data had been loaded when I was using breakpoints.

The information stored in the header section of the file will be as follows:



Most of this data is self-explanatory except for the neural network data. As the neural network can have an arbitrary size, we do not know how many bytes each part will take up. This is the reason for saving size data in metadata.

When writing to this file, the metadata will not have to change however, the other information such as Wins and Losses, and especially the neural network will have to. Due to this being most of the data, I have decided to wipe the file and rewrite all data back into it each time data is saved. This will not cause much of a performance cost but as almost all data would be written anyway, it is logical to be able to use the same function to save the data when it is created and when it is modified.

### Previous Games File

One of the largest problems I found with my program whilst making the prototype was that the AI was learning too slowly to be interesting. Therefore, I have decided to store the previous few games in a file and use these games to train the network on games that it has already seen a few times before. To save these games I will need to store them in a file in a very efficient way as I am looking to store around 500 previous games with each game lasting an average of 24 moves per game[[28]](#footnote-28). This comes to having to store around 500 \* 24 = 12000 moves.

I am planning on stored each game as a list of moves for each player. I will need to determine what player starts first which can be done with a single number, then I will store each column choice in which the game was played, alternating which player played. This will fully represent a whole connect 4 game allowing me to recreate the game and change it into a form which is useable by the backpropagation algorithm.

This file will most likely be a text file due to the ability to use newline characters ‘\n’[[29]](#footnote-29) to separate the games and having multiple C++ functions to manipulate these text files. When a game is finished, it will be saved to this file. If the file already contains more than the maximum number of games, then the eldest one should be deleted to make room for the new one. To check how many games are being stored in the file it would be too slow to load the entire file into an array and then count the size of the array. Due to this I will be storing the number of games in the file as well as all the games. This does add data redundancy[[30]](#footnote-30) but will speed the program up massively, especially when being run on a device with slow secondary storage.



This layout is how the previous games file will be structured. Each of the games will be on a single line with each game separated by a newline character. The number of games stored will be updated each time a new game is added to keep track of how many games are being stored currently. This file should allow as many past games as I decide to be stored as training data.

### File Encryption

One main problem with this file is that when saving text into a binary file, due to ASCII[[31]](#footnote-31) characters being represented with a single byte, opening a binary file as a text file will display all the bytes in their ASCII representation. Usually, this will make random characters appear but, in the case of strings which already contain single byte ASCII characters, can display the string itself. This is extremely dangerous with passwords as someone can just open the file and view the password. Therefore, I have decided to hide the password using a digest[[32]](#footnote-32). This digest will be a large number which will be used to check if an inputted password matches the stored one without having to store the password itself. The user can input their password and the hash algorithm can be applied to generate a digest. This digest can be compared against the stored digest now instead of checking the password string against the inputted string.

The algorithm I will be using to create this digest will most likely be SHA-1[[33]](#footnote-33). I chose this algorithm over the more common MD5 algorithm because it has been found to have vulnerabilities that cause security issues. SHA-1 is an algorithm that is becoming more widespread due to MD5 having these vulnerabilities. SHA-1 produces a 160-bit hash value, typically represented as 40 hexadecimal digits. As the algorithm is quite complicated and I do not need to know how it works, I will be using a C++ library which implements the SHA-1 algorithm for me. I will most likely find this library on GitHub[[34]](#footnote-34) meaning I can clone the source code and use the most up-to-date version whenever I want to build my project.

### File Checksum

To make sure that the file was not corrupted or edited in between use, I will be using SHA-1 again on the entire file to create a new digest which can be checked when the file is read to make sure that the file was not changed. I will need to make sure to not include the digest when using the SHA-1 algorithm as this will produce a different digest and will incorrectly identify the file as being changed. If the digests do not match, then the user should be presented with a message stating that the file has been corrupted or changed outside of the program.

Another method to check if a file has been corrupted or changed in between use is to use Hamming Codes[[35]](#footnote-35). This is a method of storing data in a format that allows for it to be changed and still fully represent the original message. A remarkably simple error detector would be bit parity which adds a parity bit to the end to determine whether the string of bits has an even or odd number of 1s. This can be used to determine if the number of 1s has changed but this does not tell you which one has been changed. Hamming codes on the other hand, do tell the program which bit has been changed allowing for that bit to be corrected and returned to its original state.

Hamming codes work by checking the parity of certain groups of bits which allows the program to reduce its search on which bit has been changed determined by the parity bits. This is sort of like how binary search works but with single bits and narrowing groups of bits. Hamming codes only work with a single bit that has been changed but an extra bit can be used to explicitly tell the program that more than one bit has been changed even if these bits cannot be corrected.

Implementing Hamming codes into my program would be a difficult task due to having to control bit level operations which is not explicitly possible in C++ without workarounds. Due to this I will most likely not be implementing Hamming codes but if I do have extra time and find that it is necessary then I will store the files as Hamming codes.

## Graphical User Interface (GUI)

I would like to be able to use the already existing menu system, that I created for the prototype, and move it to a Graphical User Interface[[36]](#footnote-36) using the SFML[[37]](#footnote-37) library for C++. With SFML I can create a render window and draw primitives to that window such as a rectangle or circle. Using the data that SFML collects such as mouse position, I can add functionality to these shapes to make objects such as a button or a text box.



Figure - Design for GUI

This minimalistic design is what I would like my GUI to look like when it is completed. Each of the purple boxes are buttons which, when clicked, take the user to another page of similar design with different ways of interacting. I will show and explain each of the states that will be present in my program along with the ways of interacting and desired outcome of this.

## State System

The way that my program is going to work is through a state system (which can be understood as a Finite State Machine[[38]](#footnote-38)). This state system will allow for different states to be created and used to allow my program to be broken down into easy to manage sections. I will be using polymorphism to easily render objects to the scene without having to know the exact state that is being used at that time.

These states will be contained within a stack somewhere in my program. The reason I chose a stack data type for this was that, whenever I want to move the user from one state to another, I can simply push a new state onto the top of the stack. Then when the program loop repeats, the topmost state can be used as the currently active one and its methods called – which is where polymorphism is especially useful.

Another benefit of using a stack as this data type is that old states are not removed when new states are added. Instead of overwriting an “active state” object, it is simply kept in the stack. Whenever a state finishes its task, or the exit button is pressed by the user, the state can pop itself off the stack and the previous state is revealed. This state now becomes the active state and is used by the program. This allows for an easy way to determine whether the program should close or not, as the Main Menu State will always be on the stack and the program only exiting if the user presses the exit button on this Main Menu State, we can test whether the stack is empty and if it is, we can determine that the user has pressed the exit button and the program should end.

Some roadblocks I ran into whilst making the Prototype where that when I wanted to push a state onto the stack whilst removing the current one (in case I just wanted to replace a state), popping had to occur before pushing, due to how a stack works. This caused any variables and information to be removed from memory when the state was deleted meaning I could not retain information when replacing a state. Knowing this, I have decided to pop any states after executing one frame of the state completely so that any variables in the object are not deleted when the state’s destructor[[39]](#footnote-39) is called. I will accomplish this by counting how many calls to the pop state function will occur and later on, pop that many states from the stack. A helpful tool that will be used to make sure that states are not deleted before they should be is to use the Logging System. I can print a message to say that a certain state has been deleted so that I can view any unwanted deletions.

### Main Menu State

The Main Menu State will be the first state that is pushed onto the stack state and because of this, the first state that will be viewed and interacted with by the user. An easy-to-understand design must be used to allow the user to easily navigate the program[[40]](#footnote-40) without getting lost within it. To keep the program simple, I will not be adding many extravagant details such as images or animations as these will take up a lot of time which could be spent on other more important features.

Buttons will be needed to allow the user to navigate to other states which includes Play Game State, Accounts State and Leader Board State. When clicking on any of these buttons, the Main Menu State will detect this and push a new state, corresponding to what button was pressed, on to the state stack. This will cause the program to now use the new state allowing the user to move between states.

Figure 1 already shows the design of what I would like the Main Menu State to look like. The buttons will be clicked, and the user transported to another state like this one but with different functionality.

### Confirm Exit State

I would prefer to only allow the user to be able to exit the program using the button found on the Main Menu State shown in Figure 1 because using the “x” at the top of the window can cause data to be lost if it is pressed in a case where the program is manipulating data.

One solution to this would be to disable the “x” button and only allow the user to stop running the application through the Main Menu State exit button. A big problem with this solution is that the user may get confused as to why they cannot quit the program by pressing the button if they happened to miss seeing the button when they first opened the program.

Another better solution would be to allow the user to press the “x” button, but have the program detect this and push a new state onto the state stack which allows the user to confirm whether to leave the program or continue what they were doing, at the same time warning them that data may be lost if they are currently doing something. During this state, the program could also delay closing the window to save any currently in use data so that as little important data is lost as possible.



Figure - Confirm Exit State

Figure 2 shows the design for the state that should be displayed whenever the user presses the “x” button at the top of the window. This state will ask the user to confirm whether to leave the program or go back to what they were doing. If “Exit Program” is pressed, then all states should be popped of the state stack and the program will exit. If “Cancel” is pressed, this state should pop itself off the state stack revealing the previous state that was in use allowing the user to continue what they were just doing.

### Accounts State

If after, on the Main Menu State, the “Accounts” button is clicked, the user should be navigated to a place with the ability to create new accounts and delete old accounts. This will be done with a simple screen with two buttons labelled “Create Account” and “Delete Account”. A “back” button will also be shown allowing the user to return to the previous state by popping the current state off from the stack.



Figure - Accounts State

Figure 3 shows the design for the Accounts State which will act as an intermediary state between the main menu and either creating or deleting user accounts.

### Create Account State

If the user selects the “Create Account” button on the Accounts State, then they will be navigated to a screen where they are able to create an account using a username and password which they get to determine. This is also the point at which the AI’s neural network shall be created as the user account needs to be stored in a file with all information related to that user. The state will need to check that the username entered does not already exist so that multiple users cannot have the same name causing data to be lost. If a username already exists, then the user should be notified and prompted to enter a different username.

Overall, this state will receive input from the user for both their username and password, add these to an object which will store all information about the user and initialise all other details such as date created and the AI’s neural network itself. The next step that this state will complete is that it needs to save the object into a file so that it is possible to recreate the object when the program is run again. Once the user has created their account and it has been saved to a file, the program should return to the Accounts State allowing the user to choose between creating another account or deleting an account.

The user will also have the option to go back if they have changed their mind and do not want to create an account. This button will simply pop the current state returning the user to the Accounts State.



Figure - Create Account State

### Delete Account State

If the user clicks the Delete Account Button on the Accounts State, then they will be navigated to a screen where they can delete an already existing account. This will remove all information about the user so that they cannot login and will not show up on the leader board anymore. As all user information is stored in files, these files will need to be deleted making sure any extra files are removed as well leaving no trace that the user ever had an account.

This state will be like the Create Account State as in, the user will input a Username and a Password, and the state will determine whether this is a valid input and then delete the account if that is the case. Determining whether the input is valid is based on whether the user account does exist which is the opposite to the Create Account State. If the user does exist and the password matches that user’s password, the account should be deleted. If the password does not match, the account should not be deleted as it could be done by someone who does not own the account. The final case is when the username does not match any username meaning that the account trying to be deleted does not exist. In this case the user will be prompted to re-input the username or quit.

As the password is to be encrypted, the state cannot simply check whether the passwords match. The encryption algorithm must be run on the inputted password and the hashed version of the password must match giving almost certainty that the password inputted was the stored one.



Figure - Delete Account State

### Leader Board State

If the user decides to press the Leader Board Button on the Main Menu State, they will be navigated to a leader board where a few accounts will be displayed along with information about them. This will be so that users can see who has the most wins and who’s AI has the most wins. Any user will be able to view this information as it does not show passwords only statistics that are determined by playing the game.

If the user clicks on one of the accounts displayed in the list, they will be given all information related to that user such as wins and losses for both human and AI. This will allow specific information to be viewed instead of just a list of all users.

Initially, all accounts will be sorted by username in alphabetical order. This will be done by loading all usernames into an array and sorting that array using quick sort[[41]](#footnote-41). Quick sort is a sorting algorithm that executes in O(nlogn) time equivalent to that of merge sort[[42]](#footnote-42). Quick sort does have a worst-case time complexity of O(n2) however this rarely occurs due to optimizations in choosing a pivot value to partition the data. Quick sort works by selecting a pivot which is compared to every other value in the list. Values smaller than this pivot are put into a separate list and values greater are put into a different list. This divides the entire data in two parts where no values in either part need to be compared with each other as they are divided by this pivot. This is then repeated for these smaller lists until the entire data is sorted.

If this sorting algorithm happens to be too slow for the data, I can implement a radix sort algorithm[[43]](#footnote-43). This sorting algorithm has time complexity of O(n) which is extremely efficient however, it works best with numeric keys meaning using it for a string would require some tweaking namely, using the characters ASCII value instead of the character itself.

The user will also be allowed to choose what heuristic to sort the users by. This will include name, human wins, human losses, AI wins, AI losses and date created. Another feature of the leader board will be that the user can input a username to search for. This will display any users with a username like the one inputted. As the user will not be expected to perfectly remember all usernames, they should be given the benefit of the doubt and not only exact matches to the inputted username should be displayed but similar ones too. This will require an algorithm which can match similar strings together.

The two algorithms that I have found which can do this are Levenshtein Distance[[44]](#footnote-44) and Soundex[[45]](#footnote-45). These two algorithms work in different ways where the Levenshtein Distance works by comparing two string and returning a metric on how similar these two strings are. A value can then be used to remove any users whose name is too different to the one inputted by the user. Soundex on the other hand, works by, encoding a string as a different string representing its phonetic pronunciation. Strings that sound the same will be encoded by the same key and therefore will be detected as being the same string. Both algorithms have their benefits so perhaps, both can be used in conjunction with each other to produce a convincing result.



Figure - Leader Board State

As I do not know how many users will be stored in my program, using an array may begin to get slow and give the user a bad experience. I am considering using a Hash table[[46]](#footnote-46) to store all the users along with their information for constant look up times speeding up the process. The only problem with this is that it will be hard to implement sorting and searching whilst using a hash map. Therefore, I am most likely going to stick to a simple array but can move my program to use a hash map if it becomes necessary.

As shown in Figure 7, the user will also be able to press a back button which allows them to return to the Main Menu State if they want to.

### Play Game State

If the user presses the Play Button on the Main Menu State, they will be navigated to a state where they are about to play Connect 4. Before they play the game, they will need to be asked who is going to play in which they will input a username and password for the account they wish to use. This will be done twice, once for each player with the choice to select AI where the users AI will play for them instead. If AI is selected, then the user will not need to enter their password to allow for other users to play against other people’s AI.

This is a simple state in which a list of players will have to be loaded and checked if the inputted information (username and password) is valid. If it is not valid, the program should ask for the user to input the information again. Once both users have entered their information, a special type of account object will be created which uses polymorphism to allow both users to be used as arguments into a function irrespective of if the user is playing as an AI or a human. Once these objects are created, the user will be navigated to the Connect 4 Game State where they will play Connect 4 with the chosen account information.



Figure - Play Game State

Figure 8 shows the design for the Play Game State. As shown, the user will also be able to press the Back button which will allow them to return to the Main Menu State and select a different option if they please.

### Connect 4 Game State

This state will contain the Connect 4 Game where the two players decided in the Play Game State will take turns placing counters until a player has won the game or a draw occurs. This state is also where the training data for the neural network will be collected if a human is playing currently. As the neural network will only be trained on games that the human has played, the state will not need to save the game if an AI is playing, to save memory.

The users will use the mouse to choose a column and place a counter at the lowest available position however, if the user is an AI then they will need to choose a column using their neural network. This will be done by giving the neural network the current board positions and it returning a single number representing the column choice. The state will then check if the game has been won or drawn. As an optimization, the state only needs to check counters around the most recently placed counter as this is the only change to the game that can cause the game to be won. To check if the game has been won, we can count the number of same-coloured counters in a row around the most recent counter, if this number is four or greater than the game has been won. To check if the game has been drawn, we can simply check if there are any empty spaces in the top row. If there are then the game is not yet over but if all spaces in the top row are filled up, then we can know with confidence that all spaces have been used and the game is drawn.

As this is happening, the state will save all column choices in a list along with which player played first. This will allow the training data to be constructed for either user if necessary. If both players are AI, the state should not save any choices as training data will not need to be constructed allowing less memory to be used whilst playing the game.

After the game is over, the user object will use the current game and previous games stored in a file to train the neural network using backpropagation. After updating the neural network, the recently played game should be stored in the file along with all other previous games. Also, depending on whether the game has been won or drawn, a Game Won State or Game Draw State should be pushed onto the stack replacing the current Connect 4 Game State.



Figure - Connect 4 Game State

Figure 9 shows the design for the Connect 4 Game State. The Confirm button in the top right corner will be used to allow the human player the confirm that the AI’s choice has been registered and that after the game has been won/drawn, they can look at the board before being transported to the next state.

### Game Won State

If the game has been won in the Connect 4 Game State, this Game Won State will be used to congratulate the player that has won the game. It will be a simple state that shows the winning players name and how many wins they now have. It will also show the losing players name along with how many losses they now have. A message will also be displayed if either of the players were humans to say that their AIs have now been trained on that game just played so that they get confirmation that they are training their AI.

This state will also increase the number of wins/losses for the players which can be viewed in the Leader Board State.



Figure - Game Won State

Figure 10 shows the design for the Game Won State. The only interaction the user will be able to do on this state is that they can press the back button which will pop the state from the stack causing the user to return to the Main Menu State.

### Game Draw State

If the game has been drawn in the Connect 4 Game State, the user will be navigated to the Game Draw State where they will be greeted with a message saying that both players have drawn and that this will not affect their win/losses. A message will also be displayed saying that the AI has been trained if either player was a human.



Figure - Game Draw State

Figure 11 shows the design for the Figure 11. The only interaction the user will be able to do on this state is that they can press the back button which will pop the state from the stack causing the user to return to the Main Menu State.

## Object Oriented Programming

As my program will contain Object Oriented Programming, I need to know what classes I will need to create and what methods they will contain. I will be using class diagrams[[47]](#footnote-47) to show the classes in my program and how they relate to each other. As there are separate parts of my program that will not be related explicitly, I will use multiple class diagrams allowing more detail to be fit into a single diagram. The use of class diagrams is to plan interfaces of classes allowing an overview of the program to be visualised without needing to know implementation details. This is one of the principles of creating an object-oriented program[[48]](#footnote-48).

The tool I will be using to create my class diagrams is called Lucid Chart[[49]](#footnote-49) and it allows UML[[50]](#footnote-50) diagrams to be created easily and exported to file types such as PDF allowing easy viewing and sharing. UML is one of the most popular languages that allows for models to be created as it is simple, and many tools use it meaning it is easily accessible.

The sections in which I will be breaking down my class diagrams include: State System, Neural Network, Accounts and Graphics. These five sections are the main building blocks of my program all of which are extremely complicated in how they work but class diagrams should allow a much simpler representation of these sections. Many of these sections will be included in another such as Graphics will most definitely be used within the States however, I will still separate these as a single class diagram for my whole project will be too large and to make it fit on a single page would need to be too small to read or even understand.

# Technical Solution

## Introduction

The entire technical solution can be found on GitHub[[51]](#footnote-51) with build instructions located in the README.md file.

The code snippets shown in this section each have a caption which can be referenced in other explanations in the code. The caption also contains a file path shown in square brackets describing the exact location of this file relative to the project root also showing the filename on the end along with its extension.

## Contents

Premake 43

Tools 45

Entry Point 47

Systems 48

Program 59

States 64

Confirm Exit 65

Main Menu 67

Accounts 69

Create Account State 76

Delete Account State 79

Play Game 83

Connect 4 89

Game State 89

Accounts 94

Game Over 99

Leader Board 102

Graphics 110

Button 110

Text 114

Text Input 115

## Premake

workspace "Connect 4 With AI"

    configurations { "Debug", "Release" }

    architecture "x64"

os.execute [[vendor\CMake\bin\cmake -DBUILD\_SHARED\_LIBS=OFF -S "Connect 4\vendor\SFML" -B "Connect 4\vendor\SFML" -G "MinGW Makefiles"]]

externalproject "SFML"

    location "Connect 4/vendor/SFML"

    kind "StaticLib"

    language "C++"

Code Snippet – Premake Workspace and Submodules – [premake5.lua]

This Premake file contains the instructions on how to create build files used for compiling the code. This snippet describes the workspace (otherwise known as solution) and the external project SFML which is using CMake[[52]](#footnote-52) to generate those build files.

The *os.execute* command, runs a shell command to run CMake on the SFML CMake files to build project files.

project "Connect 4"

    kind "ConsoleApp"

    language "C++"

    location "%{prj.name}"

    targetname "Connect4"

    targetdir "bin/%{cfg.buildcfg}"

    objdir "bin/%{cfg.buildcfg}-obj"

    systemversion "latest"

    cppdialect "C++17"

    debugdir "%{prj.name}/"

    files

    {

        "%{prj.name}/src/\*\*.cpp"

    }

    includedirs

    {

        "%{prj.name}/src",

        "%{prj.name}/vendor/SFML/include",

        "%{prj.name}/vendor/Eigen",

        "%{prj.name}/vendor"

    }

    libdirs

    {

        "%{prj.name}/vendor/SFML/lib",

        "%{prj.name}/vendor/Freetype/release static/win64"

    }

    defines "SFML\_STATIC"

    links

    {

        "stdc++fs",

        "sfml-graphics-s",

        "sfml-window-s",

        "sfml-system-s",

        "opengl32",

        "freetype",

        "winmm",

        "gdi32"

    }

    filter "configurations:Debug"

        symbols "On"

        defines "DEBUG"

    filter "configurations:Release"

        optimize "On"

        defines "NDEBUG"

Code Snippet – Premake Project – [premake5.lua]

The rest of the Premake file contains information about the project itself. It describes that it is a C++17 project and where to build it. It also states what files should be included and what libraries it should link to, mainly, SFML.

At the bottom, there are some filters which turn on and off certain flags to change how the project is compiled during Debug builds and Release builds.

## Tools

#pragma once

#include <string>

#ifdef DEBUG

#define LOG\_TRACE(x) Connect::Logger::s\_Instance->PrintMessage(x, Connect::LOG\_LEVEL::TRACE)

#define LOG\_WARN(x) Connect::Logger::s\_Instance->PrintMessage(x, Connect::LOG\_LEVEL::WARN)

#define LOG\_ERROR(x) Connect::Logger::s\_Instance->PrintMessage(x, Connect::LOG\_LEVEL::ERROR)

#else

#define LOG\_TRACE(x)

#define LOG\_WARN(x)

#define LOG\_ERROR(x)

#endif

namespace Connect

{

    enum class LOG\_LEVEL

    {

        TRACE = 0,

        WARN  = 1,

        ERROR = 2

    };

    // Simple Logging System to print info, warnings and errors

    class Logger

    {

    public:

        Logger(std::string loggerName);

        virtual ~Logger();

    public:

        void PrintMessage(std::string message, LOG\_LEVEL level);

        inline void SetLogLevel(LOG\_LEVEL level) { m\_LogLevel = level; }

    private:

        LOG\_LEVEL m\_LogLevel;

        std::string m\_LoggerName;

        static const std::string LEVELS[];

    public:

        static Logger\* s\_Instance;

    };

}

Code Snippet – Logging System Declaration – [Connect 4/src/Tools/Logger.h]

This logger is a singleton[[53]](#footnote-53) class used to print out any information, warnings, or error messages whilst the program is running. The *#define* allows for the logging code to be completely removed when compiling in Release mode because I do not want any performance issues when releasing the application.

#include "Logger.h"

#include <iostream>

#include <chrono>

#include <ctime>

namespace Connect

{

    const std::string Logger::LEVELS[] = {"TRACE", "WARN ", "ERROR"};

    Logger\* Logger::s\_Instance = nullptr;

    Logger::Logger(std::string loggerName)

    {

        // Check that the logger hasn't already been created

        if (s\_Instance)

        {

            s\_Instance->PrintMessage(

"Attempt to create multiple logger objects", LOG\_LEVEL::WARN);

            return;

        }

        else

        {

            s\_Instance = this;

        }

        // Set default log level

        m\_LogLevel = LOG\_LEVEL::WARN;

        if (loggerName.empty())

            m\_LoggerName = "Default Logger";

        else

            m\_LoggerName = loggerName;

        return;

    }

    Logger::~Logger() {}

    void Logger::PrintMessage(std::string message, LOG\_LEVEL level)

    {

        // Only print messages above the current log level

        if (level < m\_LogLevel)

            return;

        // Print in "Level - [DateTime] - LoggerName: Message" Format

        std::time\_t currentTime = std::chrono::system\_clock::to\_time\_t(

std::chrono::system\_clock::now());

        printf("%s - [%.19s] - %s: %s\n", LEVELS[(int)level].c\_str(),

std::ctime(&currentTime), m\_LoggerName.c\_str(), message.c\_str());

    }

}

Code Snippet – Logging System Definition – [Connect 4/src/Tools/Logger.cpp]

The implementation of the logging system is quite simple as messages are printed as soon as they are issued. Only messages that reach a certain threshold are printed however meaning information messages can be turned off so that only warnings and errors are printed.

## Entry Point

#include <iostream>

#include <SFML/Window.hpp>

#include <SFML/Graphics.hpp>

#include "Tools/Logger.h"

#include "Program/Program.h"

#include "Program/States/MainMenuState.h"

int main()

{

    // Create logger object

    Connect::Logger logger("Connect 4");

    logger.SetLogLevel(Connect::LOG\_LEVEL::TRACE);

    // Create new program and use main menu as the inital state

    Connect::Program program;

    program.PushState(new Connect::MainMenuState());

    while (program.isRunning())

        program.ExecuteFrame();

    return 0;

}

Code Snippet – Entry Point of Program – [Connect 4/src/main.cpp]

The entry point of my program is quite simple as it is an object-oriented program. The only objects that are created are both the logger and the program itself. After these have been created the program is executed every frame while it is still running.

## Systems

#include <Eigen/Dense>

#include <vector>

#include <memory>

namespace Connect

{

    typedef std::pair<std::shared\_ptr<char[]>, size\_t> BinaryData;

    typedef std::pair<Eigen::VectorXf, Eigen::VectorXf> TrainingExample;

    class NeuralNetwork

    {

    public:

        // Random weights and biases

        NeuralNetwork(std::vector<size\_t> layerSizes);

        // Data from a file

        NeuralNetwork(std::vector<unsigned int>& layerSizes, std::shared\_ptr<char[]> data);

        virtual ~NeuralNetwork();

    public:

        Eigen::VectorXf FeedFoward(Eigen::VectorXf inputVector);

        BinaryData Serialize();

        std::vector<unsigned int> GetLayerSizes();

        // Returns the accuracy of the network, lower = better

        float Train(std::vector<TrainingExample>& trainingData, float eta);

    private:

        float Backpropagation(std::vector<Eigen::MatrixXf>& nablaWeights, std::vector<Eigen::VectorXf>& nablaBiases, TrainingExample& example);

        void RandomInitialize();

        // Activation Function

        Eigen::VectorXf activationFunctionVector(Eigen::VectorXf& vector);

        Eigen::VectorXf activationFunctionPrimeVector(Eigen::VectorXf& vector);

        static inline float activationFunction(float value);

        static inline float activationFunctionPrime(float value);

    private:

        std::vector<Eigen::MatrixXf> m\_Weights;

        std::vector<Eigen::VectorXf> m\_Biases;

    };

}

Code Snippet – Neural Network Class Declaration – [Connect 4/src/Program/Systems/NeuralNetwork.h]

This Neural Network class is a standalone class that has no relation to Connect 4 or any part of the program. The only methods that have any relation to other parts of the program are the serialization where the neural network can be saved and loaded from a file.

#include "NeuralNetwork.h"

#include "Tools/Logger.h"

namespace Connect

{

    NeuralNetwork::NeuralNetwork(std::vector<size\_t> layerSizes)

    {

        // Resize array to correct size

        m\_Weights.reserve(layerSizes.size() - 1);

        m\_Biases.reserve(layerSizes.size() - 1);

        for (int i = 1; i < layerSizes.size(); i++)

        {

            m\_Weights.emplace\_back(Eigen::MatrixXf(layerSizes[i], layerSizes[i - 1]));

            m\_Biases.emplace\_back(Eigen::VectorXf(layerSizes[i]));

        }

        LOG\_TRACE("Neural Network object created");

        RandomInitialize();

    }

    NeuralNetwork::NeuralNetwork(std::vector<unsigned int> &layerSizes, std::shared\_ptr<char[]> data)

    {

        char\* pointer = data.get();

        // Reserve enough space for all layers

        m\_Weights.reserve(layerSizes.size() - 1);

        m\_Biases.reserve(layerSizes.size() - 1);

        for (int i = 1; i < layerSizes.size(); i++)

        {

            m\_Weights.emplace\_back(Eigen::MatrixXf(layerSizes[i], layerSizes[i - 1]));

            m\_Biases.emplace\_back(Eigen::VectorXf(layerSizes[i]));

            for (int j = 0; j < layerSizes[i - 1]; j++)

            {

                // Copy each column of data into matrix

                memcpy(&m\_Weights[i - 1].col(j)[0], pointer, layerSizes[i] \* sizeof(float));

                pointer += layerSizes[i] \* sizeof(float);

            }

            // Copy entire bias vector

            memcpy(&m\_Biases[i - 1][0], pointer, layerSizes[i] \* sizeof(float));

            pointer += layerSizes[i] \* sizeof(float);

        }

        LOG\_TRACE("Constructed neural network from serialized data");

    }

    NeuralNetwork::~NeuralNetwork()

    {

    }

    Eigen::VectorXf NeuralNetwork::FeedFoward(Eigen::VectorXf inputVector)

    {

        // Multiply input vector by weights and add biase for each layer

        for (int i = 0; i < m\_Weights.size(); i++)

        {

        #ifdef DEBUG

            if (m\_Weights[i].cols() != inputVector.rows())

            {

                LOG\_ERROR("Incorrect dimensions of matrix vector multiplication");

                return inputVector;

            }

        #endif

            inputVector = m\_Weights[i] \* inputVector + m\_Biases[i];

            inputVector = activationFunctionVector(inputVector);

        }

        LOG\_TRACE("Completed feed forward algorithm");

        return inputVector;

    }

    BinaryData NeuralNetwork::Serialize()

    {

        // Serializes the neural network weights and biases

        // into a byte array to be stored in accountData.bin

        size\_t numberOfBytes = 0;

        // For each layer count number of bytes used

        for (int i = 0; i < m\_Weights.size(); i++)

        {

            numberOfBytes += sizeof(float) \* m\_Weights[i].rows() \* m\_Weights[i].cols();

            numberOfBytes += sizeof(float) \* m\_Biases[i].rows();

        }

        std::shared\_ptr<char[]> data(new char[numberOfBytes]);

        char\* pointer = data.get();;

        // For each layer save weights and biases in data array

        for (int i = 0; i < m\_Weights.size(); i++)

        {

            int previousNeurons = m\_Weights[i].cols();

            int currentNeurons = m\_Weights[i].rows();

            // For each column read every row into array

            for (int j = 0; j < previousNeurons; j++)

            {

                memcpy(pointer, &m\_Weights[i].col(j)[0], currentNeurons \* sizeof(float));

                pointer += currentNeurons \* sizeof(float);

            }

            // Save bias vector in data array

            memcpy(pointer, &m\_Biases[i][0], currentNeurons \* sizeof(float));

            pointer += currentNeurons \* sizeof(float);

        }

        LOG\_TRACE("Serialized neural network data");

        return BinaryData(data, numberOfBytes);

    }

    std::vector<unsigned int> NeuralNetwork::GetLayerSizes()

    {

        // Count neurons in each layer and return

        std::vector<unsigned int> layerSizes(m\_Weights.size() + 1);

        for (int i = 0; i < m\_Weights.size(); i++)

            layerSizes[i + 1] = m\_Weights[i].rows();

        layerSizes[0] = m\_Weights[0].cols();

        return layerSizes;

    }

    float NeuralNetwork::Train(std::vector<TrainingExample>& trainingData, float eta)

    {

        // This is slow so only do it when compiling for debug

        #ifdef DEBUG

        for (int i = 0; i < trainingData.size(); i++)

        {

            // Check example input and output has same neurons as network

            if (trainingData[i].first.rows() != m\_Weights[0].cols() || trainingData[i].second.rows() != m\_Weights.back().rows())

            {

                LOG\_ERROR("Incorrect matrix dimensions in training of neural network");

                return NAN;

            }

        }

        #endif

        // Changes to all weights and biases

        std::vector<Eigen::MatrixXf> nablaWeights(m\_Weights.size());

        std::vector<Eigen::VectorXf> nablaBiases(m\_Biases.size());

        // Initialize all nablas to 0

        for (int i = 0; i < m\_Weights.size(); i++)

        {

            nablaWeights[i] = Eigen::MatrixXf::Zero(m\_Weights[i].rows(), m\_Weights[i].cols());

            nablaBiases[i] = Eigen::VectorXf::Zero(m\_Biases[i].rows());

        }

        float cost = 0.0f;

        // Backpropagation happens here (updating nabla weights and nabla biases)

        for (TrainingExample& example : trainingData)

            cost += Backpropagation(nablaWeights, nablaBiases, example);

        // Update weights and biases based on calculated nablas

        float scalar = eta / trainingData.size();

        for (int i = 0; i < m\_Weights.size(); i++)

        {

            m\_Weights[i] -= scalar \* nablaWeights[i];

            m\_Biases[i] -= scalar \* nablaBiases[i];

        }

        cost /= (2 \* trainingData.size());

        LOG\_TRACE("Trained neural network on training data");

        LOG\_TRACE("Training neural network produced a cost of: " + std::to\_string(cost));

        return cost;

    }

    float NeuralNetwork::Backpropagation(std::vector<Eigen::MatrixXf>& nablaWeights, std::vector<Eigen::VectorXf>& nablaBiases, TrainingExample& example)

    {

        // Temporary changes to all weights and biases

        std::vector<Eigen::MatrixXf> deltaWeights(m\_Weights.size());

        std::vector<Eigen::VectorXf> deltaBiases(m\_Biases.size());

        // Initialize all deltas to 0

        for (int i = 0; i < m\_Weights.size(); i++)

        {

            deltaWeights[i] = Eigen::MatrixXf::Zero(m\_Weights[i].rows(), m\_Weights[i].cols());

            deltaBiases[i] = Eigen::VectorXf::Zero(m\_Biases[i].rows());

        }

        // Feed forward and save all activation and z values

        std::vector<Eigen::VectorXf> activations(m\_Weights.size() + 1);

        std::vector<Eigen::VectorXf> zvalues(m\_Weights.size() + 1);

        activations[0] = example.first;

        for (int i = 0; i < m\_Weights.size(); i++)

        {

            zvalues[i + 1] = m\_Weights[i] \* activations[i] + m\_Biases[i];

            activations[i + 1] = activationFunctionVector(zvalues[i + 1]);

            zvalues[i + 1] = activationFunctionPrimeVector(zvalues[i + 1]);

        }

        // Backpropagate through the network calculating the changes to the weights and biases

        Eigen::VectorXf costVector = (activations.back() - example.second);

        float cost = pow(costVector.sum(), 2);

        costVector = costVector.array() \* zvalues.back().array();

        deltaBiases[deltaBiases.size() - 1] = costVector;

        deltaWeights[deltaWeights.size() - 1] = costVector \* activations[activations.size() - 2].transpose();

        for (int i = m\_Weights.size() - 2; i >= 0; i--)

        {

            costVector = m\_Weights[i + 1].transpose() \* costVector;

            costVector = costVector.array() \* zvalues[i + 1].array();

            deltaBiases[i] = costVector;

            deltaWeights[i] = costVector \* activations[i].transpose();

        }

        // Update nabla weights and biases for total change to weights and biases

        for (int i = 0; i < deltaWeights.size(); i++)

        {

            nablaWeights[i] += deltaWeights[i];

            nablaBiases[i] += deltaBiases[i];

        }

        return cost;

    }

    void NeuralNetwork::RandomInitialize()

    {

        // Set all weights and biases to random values in a range

        for (int i = 0; i < m\_Weights.size(); i++)

        {

            // Initialize between -epsilon and +epsilon

            float epsilon = sqrt(6.0f / (m\_Weights[i].cols() + m\_Weights[i].rows()));

            // Set random is in the range [-1, 1]

            m\_Weights[i].setRandom();

            m\_Biases[i].setRandom();

            // Multiply by epsilon to get range [-eps, eps]

            m\_Weights[i] = epsilon \* m\_Weights[i];

            m\_Biases[i] = epsilon \* m\_Biases[i];

        }

        LOG\_TRACE("Neural network randomly initialized");

    }

    Eigen::VectorXf NeuralNetwork::activationFunctionVector(Eigen::VectorXf& vector)

    {

        // Apply activation function to an entire vector

        return vector.unaryExpr(std::function(activationFunction));

    }

    Eigen::VectorXf NeuralNetwork::activationFunctionPrimeVector(Eigen::VectorXf& vector)

    {

        // Apply activation function derivative to an entire vector

        return vector.unaryExpr(std::function(activationFunctionPrime));

    }

    float NeuralNetwork::activationFunction(float value)

    {

        // Sigmoid Activation Function

        // 1 / (1 + e^-x)

        return 1.0f / (1.0f + exp(-value));

    }

    float NeuralNetwork::activationFunctionPrime(float value)

    {

        // Derivative of the activation function

        float sigmoid = activationFunction(value);

        return sigmoid \* (1 - sigmoid);

    }

}

Code Snippet – Neural Network Class Definition – [Connect 4/src/Program/Systems/NeuralNetwork.cpp]

#pragma once

#include "Tools/Logger.h"

#include <vector>

#include <cstring>

#define BOARD\_WIDTH 7

#define BOARD\_HEIGHT 6

namespace Connect

{

    struct SavedGame

    {

        std::vector<int> columnChoices;

    };

    enum class BoardSpace : char

    {

        PLAYER1 = -1,

        EMPTY = 0,

        PLAYER2 = 1

    };

    enum class Outcome

    {

        DRAW,

        PLAYER1WIN,

        PLAYER2WIN

    };

    class Connect4

    {

    public:

        Connect4();

        virtual ~Connect4();

    public:

        bool isValidPosition(int column);

        void PlaceCounter(int column);

        // Game state checks

        inline bool GameOver() { return GameWon() || GameDrawn(); }

        bool GameWon();

        bool GameDrawn();

        Outcome GetOutcome();

        inline const auto& GetBoard() { return m\_Board; }

        inline bool isPlayer1Turn() { return m\_CurrentPlayer == BoardSpace::PLAYER1; }

    private:

        // Current State of the board

        BoardSpace m\_Board[BOARD\_WIDTH][BOARD\_HEIGHT];

        // Most recent positions to allow for optimizations

        int m\_RecentColumn;

        int m\_RecentRow;

        BoardSpace m\_CurrentPlayer;

    };

}

Code Snippet – Connect 4 Class Declaration – [Connect 4/src/Program/Systems/Connect4.h]

Like the Neural Network class, the Connect 4 class has no relation to the rest of the program allowing it to be re-used in a different program without any tweaks.

#include "Connect4.h"

namespace Connect

{

    Connect4::Connect4()

        : m\_RecentColumn(-1), m\_RecentRow(-1), m\_CurrentPlayer(BoardSpace::PLAYER1)

    {

        // Initialize Board to empty

        memset(m\_Board, 0, BOARD\_HEIGHT \* BOARD\_WIDTH);

    }

    Connect4::~Connect4() {}

    bool Connect4::isValidPosition(int column)

    {

        // Check whether the chosen column is valid

        // If the top space in that column is empty then it is valid

        return m\_Board[column][0] == BoardSpace::EMPTY;

    }

    void Connect4::PlaceCounter(int column)

    {

        int chosenRow = BOARD\_HEIGHT - 1;

        // Places counter in chosen column

        // Loop downwards from top to simulate gravity

        // We can assume that the top space is open and can loop from i = 1

        for (int i = 1; i < BOARD\_HEIGHT; i++)

        {

            // If its not empty then can chose the space above it

            if (m\_Board[column][i] != BoardSpace::EMPTY)

            {

                chosenRow = i - 1;

                break;

            }

        }

        // Place counter

        m\_Board[column][chosenRow] = m\_CurrentPlayer;

        m\_RecentColumn = column;

        m\_RecentRow = chosenRow;

        // Swap players

        m\_CurrentPlayer = (m\_CurrentPlayer == BoardSpace::PLAYER1) ? BoardSpace::PLAYER2 : BoardSpace::PLAYER1;

        return;

    }

    bool Connect4::GameWon()

    {

        BoardSpace recentPlayer = (m\_CurrentPlayer == BoardSpace::PLAYER1) ? BoardSpace::PLAYER2 : BoardSpace::PLAYER1;

        // Check in all possible lines (up-down, left-right, both diagonals)

        int dx[] = {0, -1, -1, -1};

        int dy[] = {-1, -1, 0, 1};

        // Loop through all directions

        for (int i = 0; i < 4; i++)

        {

            // The most recent one is already 1 in a row

            int inARow = 1;

            // Check all counters in a direction for 3 steps

            for (int direction = -1; direction < 2; direction += 2)

            {

                for (int step = 1; step < 4; step++)

                {

                    // Position to check

                    int x = m\_RecentColumn + dx[i] \* step \* direction;

                    int y = m\_RecentRow + dy[i] \* step \* direction;

                    // If out of bounds

                    if (x < 0 || x >= BOARD\_WIDTH || y < 0 || y >= BOARD\_HEIGHT)

                        break;

                    if (m\_Board[x][y] != recentPlayer)

                        break;

                    inARow++;

                }

            }

            // If 4 or more counters were found then the game has been won

            if (inARow >= 4)

                return true;

        }

        return false;

    }

    bool Connect4::GameDrawn()

    {

        // If all spaces are filled then the game is drawn

        for (int i = 0; i < BOARD\_WIDTH; i++)

            if (m\_Board[i][0] == BoardSpace::EMPTY)

                return false;

        return true;

    }

    Outcome Connect4::GetOutcome()

    {

        // Return the outcome of the game

        if (GameWon())

        {

            if (m\_CurrentPlayer == BoardSpace::PLAYER1)

                return Outcome::PLAYER2WIN;

            else

                return Outcome::PLAYER1WIN;

        }

        else

        {

            return Outcome::DRAW;

        }

    }

}

Code Snippet – Connect 4 Class Definition – [Connect 4/src/Program/Systems/Connect4.cpp]

## Program

#pragma once

#include "State.h"

#include <stack>

#include <queue>

#include <SFML/Window.hpp>

#include <SFML/Graphics.hpp>

namespace Connect

{

    class Program

    {

    public:

        Program();

        virtual ~Program();

    public:

        // Add or remove states from the stack

        void PushState(State\* newState);

        void PopState();

        void RemoveAllStates();

        // Useful methods

        inline State\* Top() { return m\_StateStack.top(); }

        inline bool isRunning() { return !m\_StateStack.empty() || !m\_StatesToPush.empty(); }

        // Will be run every frame to compute and draw

        // Program will be locked at 60fps so 1/60 is the time between frames

        void ExecuteFrame();

        inline bool WasMousePressed() { return m\_WasMousePressed; }

    private:

        void ExecuteStateChanges();

    public:

        static Program\* s\_Instance;

        // Global Font for all objects to use

        static sf::Font\* s\_Font;

        static sf::Keyboard::Key s\_KeyPressed;

    private:

        // Holds all currently active states

        std::stack<State\*> m\_StateStack;

        // Window to draw objects to

        sf::RenderWindow\* m\_Window;

        bool m\_WasMousePressed = false;

        // Number of times PopState was called

        // Used after to keep variables in memory

        int m\_StatesToPop = 0;

        // States pushed during the current frame

        std::queue<State\*> m\_StatesToPush;

    };

}

Code Snippet – Program Class Declaration – [Connect 4/src/Program/Program.h]

The program class is a singleton class which contains all the states and the window that the current program is using.

#include "Program.h"

#include "Tools/Logger.h"

#include "States/ConfirmExitState.h"

namespace Connect

{

    Program\* Program::s\_Instance = nullptr;

    sf::Font\* Program::s\_Font = nullptr;

    sf::Keyboard::Key Program::s\_KeyPressed = sf::Keyboard::Unknown;

    Program::Program()

    {

        // Check for singleton

        if (s\_Instance)

        {

            LOG\_WARN("Program object created multiple times");

            return;

        }

        else

            s\_Instance = this;

        // Create Window and give pointer to it for all states

        LOG\_TRACE("New Window created");

        m\_Window = new sf::RenderWindow(sf::VideoMode(1280, 720), "Connect 4 With AI", sf::Style::Titlebar | sf::Style::Close);

        if (!m\_Window)

        {

            LOG\_ERROR("SFML Window could not be created");

        }

        else

        {

            LOG\_TRACE("Window framerate changed to 60");

            m\_Window->setFramerateLimit(60);

            // Load global font

            s\_Font = new sf::Font();

            LOG\_TRACE("Loading Font From File");

            if (!s\_Font->loadFromFile("res/font/Calibri-Regular.ttf"))

                LOG\_ERROR("Error whilst loading font from file");

        }

    }

    Program::~Program()

    {

        // Delete any states left in stack

        while (!m\_StateStack.empty())

            PopState();

        m\_Window->close();

        LOG\_TRACE("Removed all states from stack");

        LOG\_TRACE("Called Program destructor");

    }

    void Program::PushState(State\* newState)

    {

        m\_StatesToPush.push(newState);

    }

    void Program::PopState()

    {

        m\_StatesToPop++;

    }

    void Program::RemoveAllStates()

    {

        // -1 represents pop all states

        m\_StatesToPop = -1;

    }

    void Program::ExecuteFrame()

    {

        ExecuteStateChanges();

        if (m\_StateStack.empty())

            return;

        m\_StateStack.top()->Execute();

        m\_WasMousePressed = false;

        s\_KeyPressed = sf::Keyboard::Key::Unknown;

        // Handle Input

        sf::Event e;

        while (m\_Window->pollEvent(e))

        {

            switch (e.type)

            {

            case sf::Event::Closed:

                LOG\_TRACE("Window Close callback called");

                if (dynamic\_cast<ConfirmExitState\*>(m\_StateStack.top()) == NULL)

                    PushState(new ConfirmExitState());

                break;

            case sf::Event::MouseButtonPressed:

                LOG\_TRACE("Mouse Button Pressed Event Occurred");

                m\_WasMousePressed = true;

                break;

            case sf::Event::KeyPressed:

                LOG\_TRACE("Key Press Event Occurred");

                s\_KeyPressed = e.key.code;

                break;

            }

        }

        // Clear, Draw, Display cycle

        m\_Window->clear(sf::Color(0x121212FF));

        if (!m\_StateStack.empty())

            m\_StateStack.top()->Draw();

        m\_Window->display();

    }

    void Program::ExecuteStateChanges()

    {

        // Pop required states

        if (m\_StatesToPop == -1)

        {

            LOG\_TRACE("Removing all States");

            m\_StatesToPop = m\_StateStack.size();

        }

        for (int i = 0; i < m\_StatesToPop; i++)

        {

            LOG\_TRACE("Deleting current state and removing from stack");

            delete m\_StateStack.top();

            m\_StateStack.pop();

        }

        m\_StatesToPop = 0;

        // Push new states

        while (!m\_StatesToPush.empty())

        {

            State\* newState = m\_StatesToPush.front();

            newState->SetWindow(m\_Window);

            LOG\_TRACE("Added new state to top of state stack");

            m\_StateStack.push(newState);

            LOG\_TRACE("Initialized new state");

            m\_StateStack.top()->Initialize();

            m\_StatesToPush.pop();

        }

    }

}

Code Snippet – Program Class Definition – [Connect 4/src/Program/Program.cpp]

### States

#pragma once

#include <SFML/Graphics.hpp>

namespace Connect

{

    class Button;

    // Abstract State class used to allow for polymorphism

    // to store all states in a single data structure easily

    class State

    {

    public:

        State() {};

        virtual ~State() {};

    public:

        // Used to initialize any values that need to be created a single time

        // Cannot take parameters so the construcutor of the base class must be used

        virtual void Initialize() = 0;

        // Used to update any values that may need to change based on the user

        // Perhaps an optimization can be made to only run this when something has actually changed

        virtual void Execute() = 0;

        // Used to draw all the objects in the state will be run every frame

        // Unless there is an optimization but this will be hard to communicate with main.cpp

        virtual void Draw() = 0;

        // Used to give this state the handle to the window to draw stuff

        inline void SetWindow(sf::RenderWindow\* window) { m\_Window = window; }

    // Common Button Functions

    protected:

        void PopState();

        void RemoveAllStates();

        void PushConfirmExitState();

    protected:

        static sf::RenderWindow\* m\_Window;

    public:

        Button\* recentButton = nullptr;

    };

}

Code Snippet – State Class Declaration – [Connect 4/src/Program/State.h]

The State class is an abstract base class which the other states inherit allowing polymorphism to be used within the Program class. The base state also contains some common button functions which can be used to give functionality to buttons such as a back button popping the most recent state without having to write that code for each button.

#include "State.h"

#include "Program.h"

#include "States/ConfirmExitState.h"

sf::RenderWindow\* Connect::State::m\_Window = nullptr;

void Connect::State::PopState()

{

    Program::s\_Instance->PopState();

}

void Connect::State::RemoveAllStates()

{

    Program::s\_Instance->RemoveAllStates();

}

void Connect::State::PushConfirmExitState()

{

    Program::s\_Instance->PushState(new Connect::ConfirmExitState());

}

Code Snippet – State Class Definition – [Connect 4/src/Program/State.cpp]

### Confirm Exit

#pragma once

#include "Tools/Logger.h"

#include "../State.h"

#include "Graphics/Button.h"

#include "Graphics/Text.h"

namespace Connect

{

    class ConfirmExitState : public State

    {

    public:

        ConfirmExitState()

            : m\_ExitProgramButton(State::RemoveAllStates, this),

              m\_CancelButton(State::PopState, this)

        {

            LOG\_TRACE("Created Confirm Exit State");

        }

        virtual ~ConfirmExitState() { LOG\_TRACE("Deleting Confirm Exit State"); }

    public:

        void Initialize() override

        {

            m\_ExitProgramButton.SetPosition(sf::Vector2f(300.0f, 340.0f));

            m\_ExitProgramButton.SetSize(sf::Vector2f(220.0f, 100.0f));

            m\_ExitProgramButton.SetText("Exit Program");

            m\_CancelButton.SetPosition(sf::Vector2f(760.0f, 340.0f));

            m\_CancelButton.SetSize(sf::Vector2f(220.0f, 100.0f));

            m\_CancelButton.SetText("Cancel");

            m\_Title.SetPosition(sf::Vector2f(470.0f, 50.0f));

            m\_Title.SetCharacterSize(48);

            m\_Title.SetText(" Are You Sure You\nWould Like to Exit\n    the Program?");

        }

        void Execute() override

        {

            m\_ExitProgramButton.OnMouseUpdate(m\_Window);

            m\_CancelButton.OnMouseUpdate(m\_Window);

        }

        void Draw() override

        {

            m\_Window->draw(m\_ExitProgramButton);

            m\_Window->draw(m\_CancelButton);

            m\_Window->draw(m\_Title);

        }

    private:

        Button m\_ExitProgramButton;

        Button m\_CancelButton;

        Text m\_Title;

    };

}

Code Snippet – Confirm Exit State Class – [Connect 4/src/Program/States/ConfirmExitState.h]

The Confirm Exit State is a state that will be pushed either when the window close button is pressed, or the exit button is pressed on the main menu state. This simple state will allow the user to either return to what they were doing or exit the program.

### Main Menu

#pragma once

#include "Tools/Logger.h"

#include "../State.h"

#include "Graphics/Button.h"

#include "Graphics/Text.h"

#include "PlayState.h"

#include "AccountsState.h"

#include "LeaderboardState.h"

namespace Connect

{

    class MainMenuState : public State

    {

    public:

        MainMenuState()

            : m\_ExitButton(State::PushConfirmExitState, this),

              m\_PlayButton(onClickFunction(PlayButtonFunction), this),

              m\_AccountsButton(onClickFunction(AccountsButtonFunction), this),

              m\_LeaderboardButton(onClickFunction(LeaderboardButtonFunction), this)

        {

            LOG\_TRACE("Created Main Menu State");

        }

        virtual ~MainMenuState() { LOG\_TRACE("Deleting Main Menu State"); }

    public:

        void Initialize() override

        {

            m\_ExitButton.SetPosition(sf::Vector2f(1280.0f - 150.0f, 720.0f - 150.0f));

            m\_ExitButton.SetSize(sf::Vector2f(125.0f, 125.0f));

            m\_ExitButton.SetText("Exit");

            m\_PlayButton.SetPosition(sf::Vector2f(440.0f, 300.0f));

            m\_PlayButton.SetSize(sf::Vector2f(400.0f, 75.0f));

            m\_PlayButton.SetText("Play");

            m\_AccountsButton.SetPosition(sf::Vector2f(440.0f, 400.0f));

            m\_AccountsButton.SetSize(sf::Vector2f(400.0f, 75.0f));

            m\_AccountsButton.SetText("Accounts");

            m\_LeaderboardButton.SetPosition(sf::Vector2f(440.0f, 500.0f));

            m\_LeaderboardButton.SetSize(sf::Vector2f(400.0f, 75.0f));

            m\_LeaderboardButton.SetText("Leaderboard");

            m\_Title.SetPosition(sf::Vector2f(470.0f, 50.0f));

            m\_Title.SetCharacterSize(48);

            m\_Title.SetText("  Connect 4 with\nMachine Learning");

        }

        void Execute() override

        {

            m\_ExitButton.OnMouseUpdate(m\_Window);

            m\_PlayButton.OnMouseUpdate(m\_Window);

            m\_AccountsButton.OnMouseUpdate(m\_Window);

            m\_LeaderboardButton.OnMouseUpdate(m\_Window);

        }

        void Draw() override

        {

            m\_Window->draw(m\_ExitButton);

            m\_Window->draw(m\_PlayButton);

            m\_Window->draw(m\_AccountsButton);

            m\_Window->draw(m\_LeaderboardButton);

            m\_Window->draw(m\_Title);

        }

    // Button Functions

    private:

        void PlayButtonFunction() { LOG\_TRACE("Play Button Pressed"); Program::s\_Instance->PushState(new PlayState()); }

        void AccountsButtonFunction() { LOG\_TRACE("Accounts Button Pressed"); Program::s\_Instance->PushState(new AccountsState()); }

        void LeaderboardButtonFunction() { LOG\_TRACE("Leaderboard Button Pressed"); Program::s\_Instance->PushState(new LeaderboardState()); }

    private:

        Button m\_ExitButton;

        Button m\_PlayButton;

        Button m\_AccountsButton;

        Button m\_LeaderboardButton;

        Text m\_Title;

    };

}

Code Snippet – Main Menu State Class – [Connect 4/src/Program/States/MainMenuState.h]

The Main Menu State is the default state that the program uses. This state allows the user to move to different states in the program.

## Accounts

#pragma once

#include "Tools/Logger.h"

#include "Systems/NeuralNetwork.h"

#include <string>

namespace Connect

{

    struct WinLosses

    {

        int wins = 0;

        int losses = 0;

    };

    class Account

    {

    public:

        Account(std::string name, std::string password);

        Account(std::string filename);

        virtual ~Account();

    public:

        // Doesn't have to load NN if not necessary

        void LoadNeuralNetwork();

        void SaveToFile();

        bool PasswordMatches(std::string testPassword);

        virtual inline const std::string& GetName() const { return m\_Name; }

    public:

        WinLosses HumanStats;

        WinLosses AIStats;

        time\_t DateCreated;

    private:

        std::string m\_Name;

        std::string m\_Password;

        // Used to index navigate straight to where the NN is stored

        unsigned int m\_NNOffset;

    protected:

        NeuralNetwork\* m\_NeuralNetwork;

    };

}

Code Snippet – Account Class Declaration – [Connect 4/src/Program/Account.h]

The account class is used for the leader board and used as a base class for the Connect 4 game as it contains all the information necessary to play the game and update any information about the game.

#include "Account.h"

#include "SHA-1/sha1.hpp"

#include <fstream>

#include <filesystem>

#define NEURAL\_NETWORK\_LAYER\_SIZES {42, 30, 16, 8, 7}

namespace Connect

{

    Account::Account(std::string name, std::string password)

        : m\_Name(name)

    {

        // Hash and store password

        SHA1 checksum;

        checksum.update(password);

        m\_Password = checksum.final();

        // Create new neural network with random initialization

        m\_NeuralNetwork = new NeuralNetwork(NEURAL\_NETWORK\_LAYER\_SIZES);

        DateCreated = time(nullptr);

    }

    Account::Account(std::string filename)

        : m\_NeuralNetwork(nullptr)

    {

        // Load the account from the file

        std::ifstream file(filename, std::ios::binary);

        if (!file.is\_open())

        {

            LOG\_ERROR("Cannot open file for creating account object");

            return;

        }

        // Read bytes in metadata

        int bytesInMetadata;

        file.read((char\*)&bytesInMetadata, 4);

        // Read username and password size

        int usernameSize;

        file.read((char\*)&usernameSize, 4);

        int passwordSize;

        file.read((char\*)&passwordSize, 4);

        // Read username and password string

        file.seekg(bytesInMetadata);

        m\_Name.resize(usernameSize);

        file.read(&m\_Name[0], usernameSize);

        m\_Password.resize(passwordSize);

        file.read(&m\_Password[0], passwordSize);

        // Read date/time created

        file.read((char\*)&DateCreated, sizeof(time\_t));

        // Read human/ai stats

        file.read((char\*)&HumanStats, sizeof(WinLosses));

        file.read((char\*)&AIStats, sizeof(WinLosses));

        m\_NNOffset = file.tellg();

        file.close();

    }

    Account::~Account()

    {

        // Clean up memory

        if (m\_NeuralNetwork)

            delete m\_NeuralNetwork;

    }

    void Account::LoadNeuralNetwork()

    {

        // Load the account from the file

        std::ifstream file("saves/" + m\_Name + "/accountData.bin", std::ios::binary);

        if (!file.is\_open())

        {

            LOG\_ERROR("Cannot open file for loading neural network data");

            return;

        }

        // Navigate to neural network metadata

        file.seekg(3 \* sizeof(unsigned int));

        // Read number of layers

        unsigned int numberOfLayers;

        file.read((char\*)&numberOfLayers, sizeof(unsigned int));

        // Read and save each layer

        std::vector<unsigned int> layerSizes;

        layerSizes.resize(numberOfLayers);

        for (int i = 0; i < numberOfLayers; i++)

            file.read((char\*)&layerSizes[i], sizeof(unsigned int));

        // Get size of file and navigate to neural network data

        file.seekg(0, std::ios::end);

        unsigned int sizeOfFile = file.tellg();

        file.seekg(m\_NNOffset);

        // Read rest of file containing neural network data

        std::shared\_ptr<char[]> data(new char[sizeOfFile - m\_NNOffset]);

        file.read(data.get(), sizeOfFile - m\_NNOffset);

        file.close();

        // Load neural network object

        if (m\_NeuralNetwork)

            delete m\_NeuralNetwork;

        m\_NeuralNetwork = new NeuralNetwork(layerSizes, data);

    }

    void Account::SaveToFile()

    {

        // Create saves directory

        if (!std::filesystem::exists("saves/" + m\_Name))

            std::filesystem::create\_directories("saves/" + m\_Name);

        bool alreadyExists = std::filesystem::exists("saves/" + m\_Name + "/accountData.bin");

        std::fstream file;

        if (alreadyExists)

            file.open("saves/" + m\_Name + "/accountData.bin", std::ios::binary | std::ios::out | std::ios::in);

        else

            file.open("saves/" + m\_Name + "/accountData.bin", std::ios::binary | std::ios::out);

        if (!file.is\_open())

        {

            LOG\_ERROR("Cannot open file to save data to");

            return;

        }

        if (!alreadyExists)

        {

            // Problem! To write to a binary file and keep 4 byte long numberse

            // Must save number as a lvalue and then write that

            // Can't find any workaround for this

            // ----------- Write Metadata -----------

            std::vector<unsigned int> layerSizes = m\_NeuralNetwork->GetLayerSizes();

            int sizeOfMetadata = 16 + layerSizes.size() \* sizeof(unsigned int);

            file.write(reinterpret\_cast<const char\*>(&sizeOfMetadata), 4);

            int sizeOfUsername = m\_Name.size();

            file.write(reinterpret\_cast<const char\*>(&sizeOfUsername), 4);

            int sizeOfPassword = m\_Password.size();

            file.write(reinterpret\_cast<const char\*>(&sizeOfPassword), 4);

            int numberOfLayers = layerSizes.size();

            file.write(reinterpret\_cast<const char\*>(&numberOfLayers), 4);

            for (int i = 0; i < numberOfLayers; i++)

                file.write(reinterpret\_cast<const char\*>(&layerSizes[i]), 4);

            // ----------- Write header -----------

            file.write(m\_Name.c\_str(), sizeOfUsername);

            file.write(m\_Password.c\_str(), sizeOfPassword);

            file.write(reinterpret\_cast<const char \*>(&DateCreated), sizeof(time\_t));

        }

        else

        {

            // Seek to where the WinLosses are located

            file.seekp(m\_NNOffset - 2 \* sizeof(WinLosses));

        }

        file.write(reinterpret\_cast<const char\*>(&HumanStats), sizeof(WinLosses));

        file.write(reinterpret\_cast<const char\*>(&AIStats), sizeof(WinLosses));

        // Write neural network data

        BinaryData neuralNetworkData = m\_NeuralNetwork->Serialize();

        file.write(neuralNetworkData.first.get(), neuralNetworkData.second);

        file.close();

    }

    bool Account::PasswordMatches(std::string testPassword)

    {

        SHA1 checksum;

        checksum.update(testPassword);

        return strcmp(m\_Password.c\_str(), checksum.final().c\_str()) == 0;

    }

}

Code Snippet – Account Class Definition – [Connect 4/src/Program/Account.cpp]

#pragma once

#include "Tools/Logger.h"

#include "../State.h"

#include "Graphics/Button.h"

#include "Graphics/Text.h"

#include "CreateAccountState.h"

#include "DeleteAccountState.h"

namespace Connect

{

    class AccountsState : public State

    {

    public:

        AccountsState()

            : m\_BackButton(State::PopState, this),

              m\_CreateAccountButton(onClickFunction(CreateAccountButtonFunction), this),

              m\_DeleteAccountButton(onClickFunction(DeleteAccountButtonFunction), this)

        {

            LOG\_TRACE("Created Accounts State");

        }

        virtual ~AccountsState() { LOG\_TRACE("Deleting Accounts State"); }

    public:

        void Initialize() override

        {

            m\_BackButton.SetPosition(sf::Vector2f(1280.0f - 150.0f, 720.0f - 150.0f));

            m\_BackButton.SetSize(sf::Vector2f(125.0f, 125.0f));

            m\_BackButton.SetText("Back");

            m\_CreateAccountButton.SetPosition(sf::Vector2f(300.0f, 340.0f));

            m\_CreateAccountButton.SetSize(sf::Vector2f(220.0f, 100.0f));

            m\_CreateAccountButton.SetText("Create Account");

            m\_DeleteAccountButton.SetPosition(sf::Vector2f(760.0f, 340.0f));

            m\_DeleteAccountButton.SetSize(sf::Vector2f(220.0f, 100.0f));

            m\_DeleteAccountButton.SetText("Delete Account");

            m\_Title.SetPosition(sf::Vector2f(540.0f, 50.0f));

            m\_Title.SetCharacterSize(48);

            m\_Title.SetText("Accounts");

            m\_SubTitle.SetPosition(sf::Vector2f(500.0f, 130.0f));

            m\_SubTitle.SetCharacterSize(36);

            m\_SubTitle.SetText("What Would You\n      Like To Do?");

        }

        void Execute() override

        {

            m\_BackButton.OnMouseUpdate(m\_Window);

            m\_CreateAccountButton.OnMouseUpdate(m\_Window);

            m\_DeleteAccountButton.OnMouseUpdate(m\_Window);

        }

        void Draw() override

        {

            m\_Window->draw(m\_BackButton);

            m\_Window->draw(m\_CreateAccountButton);

            m\_Window->draw(m\_DeleteAccountButton);

            m\_Window->draw(m\_Title);

            m\_Window->draw(m\_SubTitle);

        }

    // Button Functions

    private:

        void CreateAccountButtonFunction() { LOG\_TRACE("Create Account Button Pressed"); Program::s\_Instance->PushState(new CreateAccountState()); }

        void DeleteAccountButtonFunction() { LOG\_TRACE("Delete Account Button Pressed"); Program::s\_Instance->PushState(new DeleteAccountState()); }

    private:

        Button m\_BackButton;

        Button m\_CreateAccountButton;

        Button m\_DeleteAccountButton;

        Text m\_Title;

        Text m\_SubTitle;

    };

}

Code Snippet – Accounts State – [Connect 4/src/Program/States/AccountsState.h]

The Accounts State is used as a transition state to get the user to either the Create Account or Delete Account State. This state has no functionality as it only contains three buttons to move to different states.

### Create Account State

#pragma once

#include "Tools/Logger.h"

#include "../State.h"

#include "Graphics/Button.h"

#include "Graphics/Text.h"

#include "Graphics/TextInput.h"

namespace Connect

{

    class CreateAccountState : public State

    {

    public:

        CreateAccountState()

            : m\_BackButton(State::PopState, this),

              m\_CreateAccountButton(onClickFunction(CreateAccountButtonFunction), this),

              m\_UsernameInput(this, false),

              m\_PasswordInput(this, true)

        {

            LOG\_TRACE("Created Create Account State");

        }

        virtual ~CreateAccountState() { LOG\_TRACE("Deleting Create Account State"); }

    public:

        void Initialize() override

        {

            m\_BackButton.SetPosition(sf::Vector2f(1130.0f, 570.0f));

            m\_BackButton.SetSize(sf::Vector2f(125.0f, 125.0f));

            m\_BackButton.SetText("Back");

            m\_CreateAccountButton.SetPosition(sf::Vector2f(440.0f, 540.0f));

            m\_CreateAccountButton.SetSize(sf::Vector2f(400.0f, 125.0f));

            m\_CreateAccountButton.SetText("Create Account");

            m\_UsernameInput.SetPosition(sf::Vector2f(400.0f, 150.0f));

            m\_UsernameInput.SetSize(sf::Vector2f(480.0f, 60.0f));

            m\_UsernameInput.SetText("Username: ");

            m\_PasswordInput.SetPosition(sf::Vector2f(400.0f, 240.0f));

            m\_PasswordInput.SetSize(sf::Vector2f(480.0f, 60.0f));

            m\_PasswordInput.SetText("Password: ");

            m\_Title.SetPosition(sf::Vector2f(460.0f, 50.0f));

            m\_Title.SetCharacterSize(48);

            m\_Title.SetText("Create an Account");

            m\_Messages.SetPosition(sf::Vector2f(50.0f, 500.0f));

            m\_Messages.SetCharacterSize(18);

            m\_Messages.SetText("");

        }

        void Execute() override

        {

            m\_BackButton.OnMouseUpdate(m\_Window);

            m\_CreateAccountButton.OnMouseUpdate(m\_Window);

            m\_UsernameInput.OnMouseUpdate(m\_Window);

            m\_UsernameInput.KeyboardEvent();

            m\_PasswordInput.OnMouseUpdate(m\_Window);

            m\_PasswordInput.KeyboardEvent();

        }

        void Draw() override

        {

            m\_Window->draw(m\_BackButton);

            m\_Window->draw(m\_CreateAccountButton);

            m\_Window->draw(m\_UsernameInput);

            m\_Window->draw(m\_PasswordInput);

            m\_Window->draw(m\_Title);

            m\_Window->draw(m\_Messages);

        }

    // Button Functions

    private:

        void CreateAccountButtonFunction();

        void EnterDataIntoFile();

        std::string CheckInputsAreValid();

    private:

        Button m\_BackButton;

        Button m\_CreateAccountButton;

        TextInput m\_UsernameInput;

        TextInput m\_PasswordInput;

        Text m\_Title;

        Text m\_Messages;

    };

}

Code Snippet – Create Account State Class Declaration – [Connect 4/src/Program/States/CreateAccountState.h]

The Create Account State contains all the functions required to create an account and make sure that all information is stored in the appropriate files.

#include "CreateAccountState.h"

#include "Program/Account.h"

#include <filesystem>

#include <fstream>

#include <cstring>

void Connect::CreateAccountState::CreateAccountButtonFunction()

{

    LOG\_TRACE("Create Account Button Pressed");

    m\_Messages.SetText("");

    // Check user doesn't already exist and then create account

    if (!std::filesystem::is\_directory("saves"))

        std::filesystem::create\_directories("saves");

    std::string error\_message = CheckInputsAreValid();

    if (strcmp(error\_message.c\_str(), "") == 0)

    {

        // Create folder and account data file

        EnterDataIntoFile();

        LOG\_TRACE("Created account and stored data");

        m\_Messages.SetText("Successfully created user account " + m\_UsernameInput.GetInput());

    }

    else

    {

        // Print error messages if they exist

        LOG\_WARN("Problem in creating user account");

        m\_Messages.SetText("Error: " + error\_message);

    }

}

void Connect::CreateAccountState::EnterDataIntoFile()

{

    Account account(m\_UsernameInput.GetInput(), m\_PasswordInput.GetInput());

    account.SaveToFile();

}

std::string Connect::CreateAccountState::CheckInputsAreValid()

{

    std::string error\_message = "";

    // Check if username is valid and doesn't exist

    if (strcmp(m\_UsernameInput.GetInput().c\_str(), "") == 0)

        error\_message += "Please enter a username\n";

    else if (std::filesystem::is\_directory("saves/" + m\_UsernameInput.GetInput()))

        error\_message += "Username already exists\n";

    // Check if password is valid and exists

    if (strcmp(m\_PasswordInput.GetInput().c\_str(), "") == 0)

        error\_message += "Please enter a password\n";

    return error\_message;

}

Code Snippet – Create Account State Class Definition – [Connect 4/src/Program/States/CreateAccountState.cpp]

### Delete Account State

#pragma once

#include "Tools/Logger.h"

#include "../State.h"

#include "Graphics/Button.h"

#include "Graphics/Text.h"

#include "Graphics/TextInput.h"

namespace Connect

{

    class DeleteAccountState : public State

    {

    public:

        DeleteAccountState()

            : m\_BackButton(State::PopState, this),

              m\_DeleteAccountButton(onClickFunction(DeleteAccountButtonFunction), this),

              m\_UsernameInput(this, false),

              m\_PasswordInput(this, true)

        {

            LOG\_TRACE("Created Delete Account State");

        }

        virtual ~DeleteAccountState() { LOG\_TRACE("Deleting Delete Account State"); }

    public:

        void Initialize() override

        {

            m\_BackButton.SetPosition(sf::Vector2f(1130.0f, 570.0f));

            m\_BackButton.SetSize(sf::Vector2f(125.0f, 125.0f));

            m\_BackButton.SetText("Back");

            m\_DeleteAccountButton.SetPosition(sf::Vector2f(440.0f, 540.0f));

            m\_DeleteAccountButton.SetSize(sf::Vector2f(400.0f, 125.0f));

            m\_DeleteAccountButton.SetText("Delete Account");

            m\_UsernameInput.SetPosition(sf::Vector2f(400.0f, 150.0f));

            m\_UsernameInput.SetSize(sf::Vector2f(480.0f, 60.0f));

            m\_UsernameInput.SetText("Username: ");

            m\_PasswordInput.SetPosition(sf::Vector2f(400.0f, 240.0f));

            m\_PasswordInput.SetSize(sf::Vector2f(480.0f, 60.0f));

            m\_PasswordInput.SetText("Password: ");

            m\_Title.SetPosition(sf::Vector2f(460.0f, 50.0f));

            m\_Title.SetCharacterSize(48);

            m\_Title.SetText("Delete an Account");

            m\_Messages.SetPosition(sf::Vector2f(50.0f, 500.0f));

            m\_Messages.SetCharacterSize(18);

            m\_Messages.SetText("");

        }

        void Execute() override

        {

            m\_BackButton.OnMouseUpdate(m\_Window);

            m\_DeleteAccountButton.OnMouseUpdate(m\_Window);

            m\_UsernameInput.OnMouseUpdate(m\_Window);

            m\_UsernameInput.KeyboardEvent();

            m\_PasswordInput.OnMouseUpdate(m\_Window);

            m\_PasswordInput.KeyboardEvent();

        }

        void Draw() override

        {

            m\_Window->draw(m\_BackButton);

            m\_Window->draw(m\_DeleteAccountButton);

            m\_Window->draw(m\_UsernameInput);

            m\_Window->draw(m\_PasswordInput);

            m\_Window->draw(m\_Title);

            m\_Window->draw(m\_Messages);

        }

        // Button Functions

    private:

        void DeleteAccountButtonFunction();

        std::string CheckInputsAreValid();

    private:

        Button m\_BackButton;

        Button m\_DeleteAccountButton;

        TextInput m\_UsernameInput;

        TextInput m\_PasswordInput;

        Text m\_Title;

        Text m\_Messages;

    };

}

Code Snippet – Delete Account State Class Declaration – [Connect 4/src/Program/DeleteAccountState.h]

The Delete Account State allows the user to delete any accounts that already exist given that they know the username and password to stop users deleting accounts that they do not own.

#include "DeleteAccountState.h"

#include "Program/Account.h"

#include <filesystem>

#include <fstream>

#include <cstring>

void Connect::DeleteAccountState::DeleteAccountButtonFunction()

{

    LOG\_TRACE("Delete Account Button Pressed");

    m\_Messages.SetText("");

    std::string error\_message = CheckInputsAreValid();

    if (strcmp(error\_message.c\_str(), "") == 0)

    {

        // No errors occurred so delete the specified user

        LOG\_TRACE("Deleting user account");

        std::filesystem::remove\_all("saves/" + m\_UsernameInput.GetInput());

        m\_Messages.SetText("Succesfully deleted user account " + m\_UsernameInput.GetInput());

    }

    else

    {

        // Errors did occur, print them to the user

        LOG\_WARN("Problem in deleting user account");

        m\_Messages.SetText("Error: " + error\_message);

    }

}

std::string Connect::DeleteAccountState::CheckInputsAreValid()

{

    std::string error\_message = "";

    // Check username is valid and exists

    if (strcmp(m\_UsernameInput.GetInput().c\_str(), "") == 0)

        error\_message += "Please enter a username\n";

    else if (!std::filesystem::is\_directory("saves/" + m\_UsernameInput.GetInput()))

        error\_message += "User does not exist\n";

    if (!error\_message.empty())

        return error\_message;

    // Check for valid password

    if (strcmp(m\_PasswordInput.GetInput().c\_str(), "") == 0)

        error\_message += "Please enter a password";

    else

    {

        // Check if password matches

        Account account("saves/" + m\_UsernameInput.GetInput() + "/accountData.bin");

        if (!account.PasswordMatches(m\_PasswordInput.GetInput() + '\0'))

            error\_message += "Incorrect password\n";

    }

    return error\_message;

}

Code Snippet – Delete Account State Class Definition – [Connect 4/src/Program/States/DeleteAccountState.cpp]

## Play Game

#pragma once

#include "Tools/Logger.h"

#include "../State.h"

#include "Graphics/Button.h"

#include "Graphics/Text.h"

#include "Graphics/TextInput.h"

namespace Connect

{

    class PlayState : public State

    {

    public:

        PlayState()

            : m\_BackButton(State::PopState, this),

              m\_PlayGameButton(onClickFunction(PlayGameButtonFunction), this),

              m\_IsPlayer1AIButton(onClickFunction(AIButtonFunction), this),

              m\_Player1Username(this),

              m\_Player1Password(this, true),

              m\_IsPlayer2AIButton(onClickFunction(AIButtonFunction), this),

              m\_Player2Username(this),

              m\_Player2Password(this, true)

        {

            LOG\_TRACE("Created Leaderboard State");

        }

        virtual ~PlayState() { LOG\_TRACE("Deleting Leaderboard State"); }

    public:

        void Initialize() override

        {

            m\_BackButton.SetPosition(sf::Vector2f(1280.0f - 150.0f, 720.0f - 150.0f));

            m\_BackButton.SetSize(sf::Vector2f(125.0f, 125.0f));

            m\_BackButton.SetText("Back");

            m\_PlayGameButton.SetPosition(sf::Vector2f(440.0f, 540.0f));

            m\_PlayGameButton.SetSize(sf::Vector2f(400.0f, 125.0f));

            m\_PlayGameButton.SetText("Play Game");

            m\_Title.SetPosition(sf::Vector2f(480.0f, 50.0f));

            m\_Title.SetCharacterSize(48);

            m\_Title.SetText("Play Connect 4");

            // Player 1

            m\_Player1Text.SetPosition(sf::Vector2f(50.0f, 120.0f));

            m\_Player1Text.SetCharacterSize(36);

            m\_Player1Text.SetText("Player 1");

            m\_IsPlayer1AIButton.SetPosition(sf::Vector2f(50.0f, 200.0f));

            m\_IsPlayer1AIButton.SetSize(sf::Vector2f(320.0f, 60.0f));

            m\_IsPlayer1AIButton.SetText("AI: no", true);

            m\_IsPlayer1AIButton.id = 1;

            m\_IsPlayer1AI = false;

            m\_Player1Username.SetPosition(sf::Vector2f(50.0f, 300.0f));

            m\_Player1Username.SetSize(sf::Vector2f(550.0f, 60.0f));

            m\_Player1Username.SetText("Username: ");

            m\_Player1Password.SetPosition(sf::Vector2f(50.0f, 400.0f));

            m\_Player1Password.SetSize(sf::Vector2f(550.0f, 60.0f));

            m\_Player1Password.SetText("Password: ");

            m\_Player1ErrorMessages.SetPosition(sf::Vector2f(50.0f, 500.0f));

            m\_Player1ErrorMessages.SetCharacterSize(18);

            m\_Player1ErrorMessages.SetText("");

            // Player 2

            m\_Player2Text.SetPosition(sf::Vector2f(1100.0f, 120.0f));

            m\_Player2Text.SetCharacterSize(36);

            m\_Player2Text.SetText("Player 2");

            m\_IsPlayer2AIButton.SetPosition(sf::Vector2f(910.0f, 200.0f));

            m\_IsPlayer2AIButton.SetSize(sf::Vector2f(320.0f, 60.0f));

            m\_IsPlayer2AIButton.SetText("AI: no", true);

            m\_IsPlayer2AIButton.id = 2;

            m\_IsPlayer2AI = false;

            m\_Player2Username.SetPosition(sf::Vector2f(680.0f, 300.0f));

            m\_Player2Username.SetSize(sf::Vector2f(550.0f, 60.0f));

            m\_Player2Username.SetText("Username: ");

            m\_Player2Password.SetPosition(sf::Vector2f(680.0f, 400.0f));

            m\_Player2Password.SetSize(sf::Vector2f(550.0f, 60.0f));

            m\_Player2Password.SetText("Password: ");

            m\_Player2ErrorMessages.SetPosition(sf::Vector2f(850.0f, 500.0f));

            m\_Player2ErrorMessages.SetCharacterSize(18);

            m\_Player2ErrorMessages.SetText("");

        }

        void Execute() override

        {

            m\_BackButton.OnMouseUpdate(m\_Window);

            m\_PlayGameButton.OnMouseUpdate(m\_Window);

            // Player 1

            m\_IsPlayer1AIButton.OnMouseUpdate(m\_Window);

            m\_Player1Username.OnMouseUpdate(m\_Window);

            m\_Player1Username.KeyboardEvent();

            if (!m\_IsPlayer1AI)

            {

                m\_Player1Password.OnMouseUpdate(m\_Window);

                m\_Player1Password.KeyboardEvent();

            }

            // Player 2

            m\_IsPlayer2AIButton.OnMouseUpdate(m\_Window);

            m\_Player2Username.OnMouseUpdate(m\_Window);

            m\_Player2Username.KeyboardEvent();

            if (!m\_IsPlayer2AI)

            {

                m\_Player2Password.OnMouseUpdate(m\_Window);

                m\_Player2Password.KeyboardEvent();

            }

        }

        void Draw() override

        {

            m\_Window->draw(m\_BackButton);

            m\_Window->draw(m\_PlayGameButton);

            m\_Window->draw(m\_Title);

            // Player 1

            m\_Window->draw(m\_Player1Text);

            m\_Window->draw(m\_IsPlayer1AIButton);

            m\_Window->draw(m\_Player1Username);

            if (!m\_IsPlayer1AI) m\_Window->draw(m\_Player1Password);

            m\_Window->draw(m\_Player1ErrorMessages);

            // Player 2

            m\_Window->draw(m\_Player2Text);

            m\_Window->draw(m\_IsPlayer2AIButton);

            m\_Window->draw(m\_Player2Username);

            if (!m\_IsPlayer2AI)

                m\_Window->draw(m\_Player2Password);

            m\_Window->draw(m\_Player2ErrorMessages);

        }

    // Button Functions

    private:

    void AIButtonFunction();

    void PlayGameButtonFunction();

    std::string CheckInputsAreValid(TextInput &username, TextInput &password, bool AI);

    private:

        Button m\_BackButton;

        Button m\_PlayGameButton;

        TextInput m\_Player1Username;

        TextInput m\_Player1Password;

        Button m\_IsPlayer1AIButton;

        bool m\_IsPlayer1AI;

        TextInput m\_Player2Username;

        TextInput m\_Player2Password;

        Button m\_IsPlayer2AIButton;

        bool m\_IsPlayer2AI;

        Text m\_Title;

        Text m\_Player1Text;

        Text m\_Player1ErrorMessages;

        Text m\_Player2Text;

        Text m\_Player2ErrorMessages;

    };

}

Code Snippet – Play Game State Class Declaration – [Connect 4/src/Program/States/PlayState.h]

The Play State is used when the user presses the Play button on the Main Menu State allowing them to choose the users that will play the game and whether any of them are using an AI.

#include "PlayState.h"

#include "Program/Account.h"

#include "Program/Connect4Account.h"

#include "Program/Connect4AI.h"

#include "Program/Program.h"

#include "ConnectGameState.h"

#include <filesystem>

namespace Connect

{

    void PlayState::AIButtonFunction()

    {

        // Swap between AI and Human

        LOG\_TRACE("AI choice button pressed");

        if (recentButton->id == 1)

        {

            m\_IsPlayer1AI = !m\_IsPlayer1AI;

            if (m\_IsPlayer1AI)

                m\_IsPlayer1AIButton.SetText("AI: yes", true);

            else

                m\_IsPlayer1AIButton.SetText("AI: no", true);

        }

        else

        {

            m\_IsPlayer2AI = !m\_IsPlayer2AI;

            if (m\_IsPlayer2AI)

                m\_IsPlayer2AIButton.SetText("AI: yes", true);

            else

                m\_IsPlayer2AIButton.SetText("AI: no", true);

        }

    }

    void PlayState::PlayGameButtonFunction()

    {

        LOG\_TRACE("Play Game button pressed");

        // Reset error messages

        m\_Player1ErrorMessages.SetText("");

        m\_Player2ErrorMessages.SetText("");

        // Check inputs are valid for both players

        bool player1Valid = true;

        bool player2Valid = true;

        // Check player 1 inputs correct

        std::string error\_message = CheckInputsAreValid(m\_Player1Username, m\_Player1Password, m\_IsPlayer1AI);

        if (!error\_message.empty())

        {

            player1Valid = false;

            m\_Player1ErrorMessages.SetText("Error: " + error\_message);

            LOG\_WARN("Error selecting player 1");

        }

        // Check player 2 inputs correct

        error\_message = CheckInputsAreValid(m\_Player2Username, m\_Player2Password, m\_IsPlayer2AI);

        if (!error\_message.empty())

        {

            player2Valid = false;

            m\_Player2ErrorMessages.SetText("Error: " + error\_message);

            LOG\_WARN("Error selecting player 2");

        }

        // Check both players are valid

        if (player1Valid && player2Valid)

        {

            LOG\_TRACE("Both Players are valid");

            // Create either AI or Human players for the game

            Connect4Account \*player1 = m\_IsPlayer1AI ? new Connect4AI(m\_Player1Username.GetInput())

                                                     : new Connect4Account(m\_Player1Username.GetInput());

            Connect4Account \*player2 = m\_IsPlayer2AI ? new Connect4AI(m\_Player1Username.GetInput())

                                                     : new Connect4Account(m\_Player1Username.GetInput());

            // Replace this state with Game state

            Program::s\_Instance->PopState();

            Program::s\_Instance->PushState(new ConnectGameState(player1, m\_IsPlayer1AI, player2, m\_IsPlayer2AI));

        }

    }

    std::string PlayState::CheckInputsAreValid(TextInput& username, TextInput& password, bool AI)

    {

        // Check for empty text input

        if (username.GetInput() == "")

            return "Please enter a username";

        // Check that user exists for both AI and Human

        if (!std::filesystem::exists("saves/" + username.GetInput()))

            return "User doesn't exist";

        // Only check password is correct for AI

        if (!AI)

        {

            Account user("saves/" + username.GetInput() + "/accountData.bin");

            if (!user.PasswordMatches(password.GetInput()))

                return "Password is incorrect";

        }

        return "";

    }

}

Code Snippet – Play Game State Class Definition – [Connect 4/src/Program/States/PlayState.cpp]

## Connect 4

### Game State

#pragma once

#include "Tools/Logger.h"

#include "../State.h"

#include "Program/Connect4Account.h"

#include "Program/Systems/Connect4.h"

#include "Graphics/Button.h"

#include "Graphics/Text.h"

#include "Graphics/TextInput.h"

#include <SFML/Graphics.hpp>

namespace Connect

{

    class ConnectGameState : public State

    {

    public:

        ConnectGameState(Connect4Account \*player1, bool player1AI, Connect4Account \*player2, bool player2AI)

            : m\_GameOver(false), m\_WaitTimer(0),

              m\_ContinueButton(onClickFunction(ContinueButtonFunction), this)

        {

            // Create column choice buttons

            m\_ColumnChoiceButtons.reserve(BOARD\_WIDTH);

            for (int i = 0; i < BOARD\_WIDTH; i++)

            {

                m\_ColumnChoiceButtons.emplace\_back(Button(onClickFunction(ColumnChoiceButtonFunction), this, std::to\_string(i + 1)));

                float xcoord = 255.0f + i \* 115.0f;

                m\_ColumnChoiceButtons.back().SetPosition(sf::Vector2f(xcoord, 25.0f));

                m\_ColumnChoiceButtons.back().SetSize(sf::Vector2f(75.0f, 75.0f));

                m\_ColumnChoiceButtons.back().id = i;

            }

            // Save players and whether they are AI or not

            m\_Player1 = player1;

            m\_Player2 = player2;

            m\_IsPlayer1AI = player1AI;

            m\_IsPlayer2AI = player2AI;

            LOG\_TRACE("Created Connect 4 Game state");

        }

        virtual ~ConnectGameState() { LOG\_TRACE("Deleting Connect 4 Game state"); }

    public:

        void Initialize() override

        {

            m\_ContinueButton.SetPosition(sf::Vector2f(1075.0f, 25.0f));

            m\_ContinueButton.SetSize(sf::Vector2f(150.0f, 100.0f));

            m\_ContinueButton.SetText("Continue");

            if (!m\_BoardTexture.loadFromFile("res/images/Connect4Board.png"))

            {

                LOG\_ERROR("Unable to load image from file");

                return;

            }

            m\_BoardTexture.setSmooth(true);

            m\_Board.setPosition(227.0f, 100.0f);

            m\_Board.setScale(1.29f, 1.29f);

            m\_Board.setTexture(m\_BoardTexture);

        }

        void Execute() override

        {

            if (!m\_GameOver)

            {

                PlayAITurn();

                // Execute buttons if they have been pressed

                for (Button& b : m\_ColumnChoiceButtons)

                    b.OnMouseUpdate(m\_Window);

            }

            else

            {

                // Only use continue button when the game is over

                m\_ContinueButton.OnMouseUpdate(m\_Window);

            }

        }

        void Draw() override

        {

            m\_Window->draw(m\_Board);

            DrawCounters();

            if (m\_GameOver)

                m\_Window->draw(m\_ContinueButton);

            else

                for (Button &b : m\_ColumnChoiceButtons)

                    m\_Window->draw(b);

        }

    private:

        void PlayAITurn();

        void DrawCounters();

    // Button Functions

    private:

        void ColumnChoiceButtonFunction();

        void ContinueButtonFunction();

    private:

        Connect4Account\* m\_Player1;

        Connect4Account\* m\_Player2;

        bool m\_IsPlayer1AI;

        bool m\_IsPlayer2AI;

        unsigned int m\_WaitTimer;

        bool m\_GameOver;

        Connect4 m\_Game;

        SavedGame m\_SavedGame;

    private:

        std::vector<Button> m\_ColumnChoiceButtons;

        Button m\_ContinueButton;

        sf::Sprite m\_Board;

        sf::Texture m\_BoardTexture;

    };

}

Code Snippet – Connect 4 Game State Class Declaration – [Connect 4/src/Program/States/ConnectGameState.h]

The Connect Game State is where the actual game of Connect 4 is played. This state displays all the counters and the board in their correct position along with any other information relating to the game.

#include "ConnectGameState.h"

#include "Program/Program.h"

#include "GameOverState.h"

namespace Connect

{

    void ConnectGameState::PlayAITurn()

    {

        // Only get column choice if it is an AI player's turn

        if (m\_Game.isPlayer1Turn() && m\_IsPlayer1AI)

        {

            // Allow time for the user to look at the board

            if (m\_WaitTimer < 60)

            {

                m\_WaitTimer++;

            }

            else

            {

                m\_WaitTimer = 0;

                int columnChoice = m\_Player1->ColumnChoice(m\_Game, true);

                m\_Game.PlaceCounter(columnChoice);

                m\_SavedGame.columnChoices.push\_back(columnChoice);

            }

        }

        // Repeat for player 2

        if (!m\_Game.isPlayer1Turn() && m\_IsPlayer2AI)

        {

            if (m\_WaitTimer < 60)

            {

                m\_WaitTimer++;

            }

            else

            {

                m\_WaitTimer = 0;

                int columnChoice = m\_Player2->ColumnChoice(m\_Game, false);

                m\_Game.PlaceCounter(columnChoice);

                m\_SavedGame.columnChoices.push\_back(columnChoice);

            }

        }

        // Check if game has finished

        if (m\_Game.GameOver())

        {

            m\_GameOver = true;

        }

    }

    void ConnectGameState::DrawCounters()

    {

        // Draw counters in the correct positions on the board

        auto board = m\_Game.GetBoard();

        sf::CircleShape redCounter(45);

        sf::CircleShape yellowCounter(45);

        redCounter.setFillColor(sf::Color::Red);

        yellowCounter.setFillColor(sf::Color::Yellow);

        int leftSide = 245.0f;

        int topSide = 108.0f;

        // Loop through all positions and draw counters where they exist

        for (int i = 0; i < BOARD\_WIDTH; i++)

        {

            for (int j = 0; j < BOARD\_HEIGHT; j++)

            {

                if (board[i][j] == BoardSpace::PLAYER1)

                {

                    redCounter.setPosition(leftSide + i \* 116.0f, topSide + j \* 103.0f);

                    m\_Window->draw(redCounter);

                }

                else if (board[i][j] == BoardSpace::PLAYER2)

                {

                    yellowCounter.setPosition(leftSide + i \* 116.0f, topSide + j \* 103.0f);

                    m\_Window->draw(yellowCounter);

                }

            }

        }

    }

    void ConnectGameState::ColumnChoiceButtonFunction()

    {

        LOG\_TRACE("Column choice button pressed");

        // Make sure that it is a humans turn

        if (m\_Game.isPlayer1Turn() && m\_IsPlayer1AI)

            return;

        if (!m\_Game.isPlayer1Turn() && m\_IsPlayer2AI)

            return;

        if (m\_Game.isValidPosition(recentButton->id))

        {

            m\_Game.PlaceCounter(recentButton->id);

            m\_SavedGame.columnChoices.push\_back(recentButton->id);

        }

    }

    void ConnectGameState::ContinueButtonFunction()

    {

        LOG\_TRACE("Continue Button Pressed");

        Program::s\_Instance->PushState(new GameOverState(m\_Player1, m\_Player2, m\_SavedGame, m\_Game.GetOutcome()));

        Program::s\_Instance->PopState();

    }

}

Code Snippet – Connect 4 Game State Class Definition – [Connect 4/src/Program/States/ConnectGameState.cpp]

### Accounts

#pragma once

#include "Tools/Logger.h"

#include "Account.h"

#include "Program/Systems/Connect4.h"

#include <fstream>

namespace Connect

{

    class Connect4Account : public Account

    {

    public:

        Connect4Account(std::string username)

            : Account("saves/" + username + "/accountData.bin")

        {

            // Make sure that the Neural network is loaded for playing the game/training

            LoadNeuralNetwork();

        }

        virtual ~Connect4Account() {}

    public:

        virtual int ColumnChoice(Connect4& board, bool player1) { return -1; };

        virtual void TrainOnGames()

        {

            LOG\_TRACE("Training on previous games");

            std::vector<TrainingExample> gamesToTrainOn;

            // Load games from file

            std::fstream file("saves/" + GetName() + "/previousGames.gam");

            if (!file.is\_open())

            {

                LOG\_ERROR("Cannot open previous games file");

                return;

            }

            std::vector<std::pair<SavedGame, bool>> previousGames;

            // Read all games in the file

            while (!file.eof())

            {

                std::string str;

                file >> str;

                // Navigate to start of game

                if (str != "{")

                    continue;

                previousGames.push\_back(std::pair<SavedGame, bool>());

                // Read bool for is that game was played as player 1

                file >> str;

                previousGames.back().second = std::stoi(str);

                // Read in all moves until the end

                file >> str;

                while (str != "}")

                {

                    previousGames.back().first.columnChoices.push\_back(std::stoi(str));

                    file >> str;

                }

            }

            // Convert all games into correct format for neural network

            for (std::pair<SavedGame, bool> p : previousGames)

            {

                int firstMove = p.second ? 0 : 1;

                for (int i = firstMove; i < p.first.columnChoices.size() - 2; i += 2)

                {

                    Connect4 simulatedGame = SimulateGame(p.first, i);

                    auto input = ConvertBoard(simulatedGame, p.second);

                    auto output = ConvertOutput(p.first.columnChoices[i + 2]);

                    gamesToTrainOn.push\_back({ input, output });

                }

            }

            previousGames.clear();

            // Train on games multiple times

            int repeats = 1000 / gamesToTrainOn.size();

            for (int i = 0; i < repeats; i++)

                m\_NeuralNetwork->Train(gamesToTrainOn, 1.0f);

            // Save updated Neural Network

            SaveToFile();

        }

        virtual void SaveGame(SavedGame& previousGame, bool player1)

        {

            // Open file

            std::ofstream file("saves/" + GetName() + "/previousGames.gam", std::ios::app);

            if (!file.is\_open())

            {

                LOG\_ERROR("Unable to open previous games file");

                return;

            }

            // Write new game to file

            file << "{ ";

            file << player1 << ' ';

            for (int i : previousGame.columnChoices)

                file << i << ' ';

            file << "}\n";

            file.close();

        }

    protected:

        // Convert board to Neural Network input format

        Eigen::VectorXf ConvertBoard(Connect4& board, bool player1)

        {

            auto boardArray = board.GetBoard();

            // Make sure -1 corresponds to the right player

            int multiplier = player1 ? 1 : -1;

            Eigen::VectorXf output(BOARD\_WIDTH \* BOARD\_HEIGHT, 1);

            for (int i = 0; i < BOARD\_WIDTH; i++)

            {

                for (int j = 0; j < BOARD\_HEIGHT; j++)

                    output[j \* BOARD\_WIDTH + i] = (float)boardArray[i][j] \* multiplier;

            }

            return output;

        }

        // Convert column choice to Neural Network output

        Eigen::VectorXf ConvertOutput(int columnChoice)

        {

            Eigen::VectorXf output = Eigen::VectorXf::Zero(BOARD\_WIDTH);

            output[columnChoice] = 1.0f;

            return output;

        }

        // Run game and return list of moves up to a certain number

        Connect4 SimulateGame(SavedGame game, int moveNumber)

        {

            Connect4 simulation;

            for (int i = 0; i < moveNumber; i++)

                simulation.PlaceCounter(game.columnChoices[i]);

            return simulation;

        }

    };

}

Code Snippet – Connect 4 Account Class – [Connect 4/src/Program/Connect4Account.h]

The Connect 4 Account class is used to play the Connect 4 game and is also used to save the most recent game to a file. Another use of this class is to use it as base class for the AI class allowing polymorphism to be used to get the player’s column choice.

#pragma once

#include "Tools/Logger.h"

#include "Connect4Account.h"

#include "Program/Systems/Connect4.h"

namespace Connect

{

    class Connect4AI : public Connect4Account

    {

    public:

        Connect4AI(std::string username)

            : Connect4Account(username)

        {}

    public:

        int ColumnChoice(Connect4& board, bool player1) override

        {

            LOG\_TRACE("AI Made a choice");

            // Turn the board into a different type for the NN to use

            Eigen::VectorXf boardLayout = ConvertBoard(board, player1);

            auto choices = m\_NeuralNetwork->FeedFoward(boardLayout);

            // Get highest ranking valid choice

            float max = 2.0f;

            int choice = -1;

            while (true)

            {

                float currentBest = 0.0f;

                for (int i = 0; i < BOARD\_WIDTH; i++)

                {

                    if (choices[i] < max)

                    {

                        if (choices[i] > currentBest)

                        {

                            currentBest = choices[i];

                            choice = i;

                        }

                    }

                }

                // If the positions chosen is valid then use it

                // Otherwise find the next highest ranked one

                if (board.isValidPosition(choice))

                    return choice;

                else

                    max = currentBest;

            }

        }

        void TrainOnGames() override

        {

            // No need to train when the AI just played

            LOG\_WARN("Trying to train on AI");

            return;

        }

        void SaveGame(SavedGame &previousGame, bool player1) override

        {

            // No need to save the game if the AI played

            LOG\_WARN("Trying to save game using AI account");

            return;

        }

    };

}

Code Snippet – Connect 4 AI Account Class – [Connect 4/src/Program/Connect4AI.h]

### Game Over

#pragma once

#include "Tools/Logger.h"

#include "../State.h"

#include "ConnectGameState.h"

#include "Program/Connect4Account.h"

#include "Program/Systems/Connect4.h"

#include "Graphics/Button.h"

#include "Graphics/Text.h"

#include "Graphics/TextInput.h"

namespace Connect

{

    class GameOverState : public State

    {

    public:

        GameOverState(Connect4Account\* player1, Connect4Account\* player2, SavedGame previousGame, Outcome results)

            : m\_BackButton(onClickFunction(BackButtonFunction), this),

              m\_RematchButton(onClickFunction(RematchButtonFunction), this),

              m\_Player1(player1), m\_Player2(player2), m\_PreviousGame(previousGame), m\_GameOutcome(results)

        {

            LOG\_TRACE("Game Over State created");

        }

        virtual ~GameOverState() { LOG\_TRACE("Game Over State Deleted"); }

    public:

        void Initialize()

        {

            TrainAI();

            m\_BackButton.SetPosition(sf::Vector2f(1280.0f - 150.0f, 720.0f - 150.0f));

            m\_BackButton.SetSize(sf::Vector2f(125.0f, 125.0f));

            m\_BackButton.SetText("Back");

            m\_RematchButton.SetPosition(sf::Vector2f(1280.0f / 2.0f - 100.0f, 720.0f - 175.0f));

            m\_RematchButton.SetSize(sf::Vector2f(200.0f, 125.0f));

            m\_RematchButton.SetText("Rematch");

            m\_Title.SetPosition(sf::Vector2f(50.0f, 50.0f));

            m\_Title.SetCharacterSize(48);

            if (m\_GameOutcome == Outcome::DRAW)

                m\_Title.SetText("Game Drawn");

            else if (m\_GameOutcome == Outcome::PLAYER1WIN)

                m\_Title.SetText("Congratulations\n" + m\_Player1->GetName());

            else

                m\_Title.SetText("Congratulations\n" + m\_Player2->GetName());

            m\_GameInfo.SetPosition(sf::Vector2f(350.0f, 200.0f));

            m\_GameInfo.SetCharacterSize(36);

            m\_GameInfo.SetText("Any human player's AIs have been trained\n       on this and previous games");

        }

        void Execute()

        {

            m\_BackButton.OnMouseUpdate(m\_Window);

            m\_RematchButton.OnMouseUpdate(m\_Window);

        }

        void Draw()

        {

            m\_Window->draw(m\_Title);

            m\_Window->draw(m\_GameInfo);

            m\_Window->draw(m\_BackButton);

            m\_Window->draw(m\_RematchButton);

        }

    // Button Functions

    private:

        void BackButtonFunction();

        void RematchButtonFunction();

    private:

        void TrainAI();

    private:

        Connect4Account\* m\_Player1;

        Connect4Account\* m\_Player2;

        SavedGame m\_PreviousGame;

        Outcome m\_GameOutcome;

    private:

        Text m\_Title;

        Text m\_GameInfo;

        Button m\_BackButton;

        Button m\_RematchButton;

    };

}

Code Snippet – Game Over State Class Declaration – [Connect 4/src/Program/States/GameOverState.h]

#include "GameOverState.h"

#include "Program/Connect4AI.h"

#include "Program/Program.h"

namespace Connect

{

    void GameOverState::TrainAI()

    {

        // Train human player's AI on this game and also any saved games

        bool isPlayer1AI = dynamic\_cast<Connect4AI\*>(m\_Player1);

        bool isPlayer2AI = dynamic\_cast<Connect4AI\*>(m\_Player2);

        // Save this game to a file

        m\_Player1->SaveGame(m\_PreviousGame, true);

        m\_Player2->SaveGame(m\_PreviousGame, false);

        // These may take a while so multi-threading may need to be used

        m\_Player1->TrainOnGames();

        m\_Player2->TrainOnGames();

    }

    void GameOverState::BackButtonFunction()

    {

        // Free memory holding accounts

        delete m\_Player1;

        delete m\_Player2;

        Program::s\_Instance->PopState();

    }

    void GameOverState::RematchButtonFunction()

    {

        // Create Connect Game State with same players again

        bool isPlayer1AI = dynamic\_cast<Connect4AI\*>(m\_Player1);

        bool isPlayer2AI = dynamic\_cast<Connect4AI\*>(m\_Player2);

        m\_Player1->LoadNeuralNetwork();

        m\_Player2->LoadNeuralNetwork();

        Program::s\_Instance->PushState(new ConnectGameState(m\_Player1, isPlayer1AI, m\_Player2, isPlayer2AI));

        Program::s\_Instance->PopState();

    }

}

Code Snippet – Game Over State Class Definition – [Connect 4/src/Program/States/GameOverState.cpp]

## Leader Board

#pragma once

#include "Tools/Logger.h"

#include "../State.h"

#include "Graphics/Button.h"

#include "Graphics/Text.h"

#include "Graphics/TextInput.h"

#include "Program/Account.h"

#include <vector>

namespace Connect

{

    #define NUMBER\_OF\_SORTING\_HEURISTICS 6

    enum SortHeuristic

    {

        NAME = 0,

        HUMAN\_WINS,

        HUMAN\_LOSSES,

        AI\_WINS,

        AI\_LOSSES,

        DATE\_CREATED,

    };

    const std::string sortingHeuristics[NUMBER\_OF\_SORTING\_HEURISTICS] =

    {

        "Name",

        "Human Wins",

        "Human Losses",

        "AI Wins",

        "AI Losses",

        "Date Created"

    };

    class LeaderboardState : public State

    {

    public:

        LeaderboardState()

            : m\_BackButton(State::PopState, this),

              m\_SearchBar(this),

              m\_SortByButton(onClickFunction(SortByButtonFunction), this)

        {

            LOG\_TRACE("Created Leaderboard State");

            LoadUsers();

        }

        virtual ~LeaderboardState() { LOG\_TRACE("Deleting Leaderboard State"); }

    public:

        void Initialize() override

        {

            m\_BackButton.SetPosition(sf::Vector2f(25.0f, 720.0f - 150.0f));

            m\_BackButton.SetSize(sf::Vector2f(125.0f, 125.0f));

            m\_BackButton.SetText("Back");

            m\_SearchBar.SetPosition(sf::Vector2f(700.0f, 25.0f));

            m\_SearchBar.SetSize(sf::Vector2f(550.0f, 80.0f));

            m\_SearchBar.SetText("Search: ");

            m\_SortByButton.SetPosition(sf::Vector2f(700.0f, 125.0f));

            m\_SortByButton.SetSize(sf::Vector2f(550.0f, 80.0f));

            m\_SortByButton.SetText("Sort By: Name Ascending", true);

            m\_Title.SetPosition(sf::Vector2f(80.0f, 50.0f));

            m\_Title.SetCharacterSize(48);

            m\_Title.SetText("Leader Board");

            m\_UserInfo.SetPosition(sf::Vector2f(260.0f, 300.0f));

            m\_UserInfo.SetCharacterSize(24);

            m\_UserInfo.SetText("");

        }

        void Execute() override

        {

            m\_SearchBar.KeyboardEvent(onClickFunction(KeyboardEventCallback));

            m\_SearchBar.OnMouseUpdate(m\_Window);

            m\_SortByButton.OnMouseUpdate(m\_Window);

            m\_BackButton.OnMouseUpdate(m\_Window);

            for (Button &b : m\_UserButtons)

                b.OnMouseUpdate(m\_Window);

        }

        void Draw() override

        {

            m\_Window->draw(m\_Title);

            m\_Window->draw(m\_UserInfo);

            m\_Window->draw(m\_SearchBar);

            m\_Window->draw(m\_SortByButton);

            m\_Window->draw(m\_BackButton);

            for (Button &b : m\_UserButtons)

                m\_Window->draw(b);

        }

    private:

        void LoadUsers();

        void SortUsers();

        void UpdateUserButtons();

        void KeyboardEventCallback();

    // Button Functions

    private:

        void SortByButtonFunction();

        void UserButtonFunction();

    private:

        Text m\_Title;

        Button m\_BackButton;

        Text m\_UserInfo;

        TextInput m\_SearchBar;

        Button m\_SortByButton;

        std::vector<Button> m\_UserButtons;

        std::vector<Account> m\_Users;

        SortHeuristic m\_CurrentHeuristic = SortHeuristic::NAME;

        bool m\_Ascending = true;

    };

    int LevenshteinDistance(std::string left, std::string right);

    std::string SoundexConversion(std::string string);

}

Code Snippet – Leader Board State Class Declaration – [Connect 4/src/Program/States/LeaderboardState.h]

The Leader Board State is used to display all the users in an order chosen by the user. The user can change how the users are sorted and the top 5 users who match those restrictions are shown.

#include "LeaderboardState.h"

#include <filesystem>

#include <ctime>

#include <map>

namespace Connect

{

    void LeaderboardState::LoadUsers()

    {

        LOG\_TRACE("Loading users from file");

        // Create directory if doesn't exist

        if (!std::filesystem::exists("saves/"))

            std::filesystem::create\_directories("saves/");

        for (auto& p : std::filesystem::directory\_iterator("saves/"))

            // Add user to list of all users

            m\_Users.push\_back((Account(p.path().generic\_string() + "/accountData.bin")));

        SortUsers();

    }

    // Calculate the Levenshtein Distance between two strings

    int LevenshteinDistance(std::string left, std::string right)

    {

        // Matrix to store all distances for lengths of left and right

        std::vector<std::vector<int>> distanceMatrix(left.size() + 1);

        for (std::vector<int>& v : distanceMatrix)

            v.resize(right.size() + 1);

        // Distance from empty string to any string is just its length

        for (int i = 0; i < left.size() + 1; i++)

            distanceMatrix[i][0] = i;

        for (int i = 0; i < right.size() + 1; i++)

            distanceMatrix[0][i] = i;

        // Calculate distance using dynamic programming

        for (int i = 1, n = left.size() + 1; i < n; i++)

        {

            for (int j = 1, m = right.size() + 1; j < m; j++)

            {

                int substitutionCost = 0;

                if (left[i - 1] != right[j - 1])

                    substitutionCost = 4;

                distanceMatrix[i][j] = std::min(

                    {

                        distanceMatrix[i-1][j] + 1,

                        distanceMatrix[i][j-1] + 1,

                        distanceMatrix[i-1][j-1] + substitutionCost

                    });

            }

        }

        // Return distance of both full strings

        return distanceMatrix.back().back();

    }

    std::string SoundexConversion(std::string string)

    {

        char firstCharacter = string[0];

        static std::map<char, char> characterMap = {

            {'a', '0'}, {'e', '0'}, {'i', '0'}, {'o', '0'}, {'u', '0'},

            {'b', '1'}, {'f', '1'}, {'p', '1'}, {'v', '1'},

            {'c', '2'}, {'g', '2'}, {'j', '2'}, {'k', '2'}, {'q', '2'}, {'s', '2'}, {'x', '2'}, {'z', '2'},

            {'d', '3'}, {'t', '3'},

            {'l', '4'},

            {'m', '5'}, {'n', '5'},

            {'r', '6'}

        };

        // Replace all characters with their respective digit

        for (int i = 1; i < string.size(); i++)

            string[i] = characterMap[string[i]];

        // Remove repeated adjacent characters and 0s

        char lastCharacter = string[0];

        std::string newString(1, lastCharacter);

        for (int i = 1; i < string.size(); i++)

        {

            // If character is different from previous character

            if (string[i] != lastCharacter && string[i] != '0')

                newString.push\_back(string[i]);

            lastCharacter = string[i];

        }

        // There are extra steps to the algorithm but the Levenshtein distance will fix most of them

        return newString;

    }

    int GetStringHeuristic(std::string string, std::string reference)

    {

        // Convert both string using Soundex then return the distance between those two strings

        return LevenshteinDistance(SoundexConversion(string), SoundexConversion(reference));

    }

    void LeaderboardState::SortUsers()

    {

        LOG\_TRACE("Sorting users");

        std::string searchText = m\_SearchBar.GetInput();

        // Function to compare two accounts to each other

        // If not using the search bar then just use default sorting method

        std::function<bool(Account &, Account &)> comparisonFunction;

        // Determine sorting function based on heuristic

        switch (m\_CurrentHeuristic)

        {

        case SortHeuristic::NAME:

            comparisonFunction = [](Account& left, Account& right){ return strcmp(left.GetName().c\_str(), right.GetName().c\_str()) != 1; };

            break;

        case SortHeuristic::HUMAN\_WINS:

            comparisonFunction = [](Account &left, Account &right) { return left.HumanStats.wins < right.HumanStats.wins; };

            break;

        case SortHeuristic::HUMAN\_LOSSES:

            comparisonFunction = [](Account &left, Account &right) { return left.HumanStats.losses < right.HumanStats.losses; };

            break;

        case SortHeuristic::AI\_WINS:

            comparisonFunction = [](Account &left, Account &right) { return left.AIStats.wins < right.AIStats.wins; };

            break;

        case SortHeuristic::AI\_LOSSES:

            comparisonFunction = [](Account &left, Account &right) { return left.AIStats.losses < right.AIStats.losses; };

            break;

        case SortHeuristic::DATE\_CREATED:

            comparisonFunction = [](Account &left, Account &right) { return left.DateCreated < right.DateCreated; };

            break;

        default:

            LOG\_ERROR("Invalid sorting heuristic used");

            return;

            break;

        }

        std::function<bool(Account &, Account &)> comparisonWithHeuristic = [comparisonFunction](Account &left, Account &right) { return comparisonFunction(left, right); };

        // If using search bar then compare by Levenshtein distance of Soundex form then sort by default method

        if (!searchText.empty())

        {

            comparisonWithHeuristic = [&](Account &left, Account &right) {

                // Recomputing heuristics multiple times possible optimization

                int leftHeuristic = GetStringHeuristic(left.GetName(), searchText);

                int rightHeuristic = GetStringHeuristic(right.GetName(), searchText);

                if (leftHeuristic == rightHeuristic)

                    return comparisonFunction(left, right);

                return leftHeuristic < rightHeuristic;

            };

        }

        std::sort(m\_Users.begin(), m\_Users.end(), comparisonWithHeuristic);

        if (!m\_Ascending)

            std::reverse(m\_Users.begin(), m\_Users.end());

        LOG\_TRACE("Finished sorting users");

        UpdateUserButtons();

    }

    void LeaderboardState::UpdateUserButtons()

    {

        // Display top 5 users

        for (int i = 0; i < 5 && i < m\_Users.size(); i++)

        {

            // Check whether the button at that position already exists

            if (i >= m\_UserButtons.size())

            {

                m\_UserButtons.push\_back(Button(onClickFunction(UserButtonFunction), this));

                m\_UserButtons[i].SetPosition(sf::Vector2f(825.0f, 225.0f + i \* 100.0f));

                m\_UserButtons[i].SetSize(sf::Vector2f(425.0f, 80.0f));

            }

            m\_UserButtons[i].SetText(m\_Users[i].GetName(), true);

            m\_UserButtons[i].id = i;

        }

    }

    void LeaderboardState::KeyboardEventCallback()

    {

        // When the user types in the search box

        SortUsers();

        UpdateUserButtons();

    }

    void LeaderboardState::SortByButtonFunction()

    {

        // Change sorting method

        if (!m\_Ascending)

        {

            m\_Ascending = true;

            m\_CurrentHeuristic = static\_cast<SortHeuristic>((m\_CurrentHeuristic + 1) % NUMBER\_OF\_SORTING\_HEURISTICS);

        }

        else

            m\_Ascending = false;

        m\_SortByButton.SetText("Sort By: " + sortingHeuristics[m\_CurrentHeuristic] + (m\_Ascending ? " Ascending" : " Descending"), true);

        LOG\_TRACE("Changed Sorting heuristic");

        SortUsers();

    }

    void LeaderboardState::UserButtonFunction()

    {

        LOG\_TRACE("User clicked");

        // Display All Info about clicked user

        Account& user = m\_Users[recentButton->id];

        std::string info = user.GetName();

        info += "\n";

        info += "\nHuman Wins: " + std::to\_string(user.HumanStats.wins);

        info += "\nHuman Losses: " + std::to\_string(user.HumanStats.losses);

        info += "\nAI Wins: " + std::to\_string(user.AIStats.wins);

        info += "\nAI Losses: " + std::to\_string(user.AIStats.losses);

        info += "\n\n";

        char buffer[30];

        strftime(buffer, sizeof(buffer), "%d/%m/%Y %H:%M:%S", std::localtime(&user.DateCreated));

        info += "Date Created: " + std::string(buffer);

        m\_UserInfo.SetText(info);

    }

}

Code Snippet – Leader Board State Class Definition – [Connect 4/src/Program/States/LeaderboardState.cpp]

## Graphics

### Button

#pragma once

#include <string>

#include <SFML/Graphics.hpp>

#include <SFML/System.hpp>

#include "Program/State.h"

namespace Connect

{

    typedef void (State::\*onClickFunction)();

    class Button : public sf::Drawable

    {

    public:

        Button(onClickFunction function, State\* parentState, std::string text = "Default Text");

        virtual ~Button();

    public:

        void SetPosition(sf::Vector2f position);

        void SetSize(sf::Vector2f size);

        void SetText(std::string text, bool centerLeft = false);

        void OnMouseUpdate(sf::RenderWindow\* window);

    private:

        // Override draw method to draw both rect and text

        void draw(sf::RenderTarget& target, sf::RenderStates states) const;

        // Recenter Text

        void CenterText();

        void CenterTextLeft();

    private:

        // Drawable objects

        sf::RectangleShape m\_Rect;

        sf::Text m\_Text;

    public:

        onClickFunction m\_ClickFunction = nullptr;

        State\* m\_ParentState = nullptr;

        // Used to know which button was pressed

        int id = -1;

    };

}

Code Snippet – Button Class Declaration – [Connect 4/src/Graphics/Button.h]

The Button Class inherits from a SFML drawable object allowing it to be drawn to the window easily by overriding the draw method.

#include "Button.h"

#include <functional>

#include "Tools/Logger.h"

#include "Program/Program.h"

namespace Connect

{

    Button::Button(onClickFunction function, State \*parentState, std::string text)

        : m\_ClickFunction(function), m\_ParentState(parentState)

    {

        // Set default style and position

        m\_Rect.setFillColor(sf::Color(0x5E1F8DFF));

        m\_Rect.setPosition(sf::Vector2f(0.0f, 0.0f));

        m\_Rect.setSize(sf::Vector2f(100.0f, 100.0f));

        m\_Text.setFont(\*Program::s\_Font);

        m\_Text.setString(text);

        m\_Text.setCharacterSize(24);

        m\_Text.setFillColor(sf::Color(0xD9D9D9FF));

        LOG\_TRACE("Button object created");

        CenterText();

    }

    Button::~Button()

    {

    }

    void Button::SetPosition(sf::Vector2f position)

    {

        m\_Rect.setPosition(position);

        CenterText();

    }

    void Button::SetSize(sf::Vector2f size)

    {

        m\_Rect.setSize(size);

        CenterText();

    }

    void Button::SetText(std::string text, bool centerLeft)

    {

        m\_Text.setString(text);

        if (centerLeft)

            CenterTextLeft();

        else

            CenterText();

    }

    void Button::OnMouseUpdate(sf::RenderWindow\* window)

    {

        // Reset to default colour

        m\_Rect.setFillColor(sf::Color(0x5E1F8DFF));

        // Check mouse is inside rectangle

        bool mouseInside = m\_Rect.getGlobalBounds().contains(sf::Vector2f(sf::Mouse::getPosition(\*window)));

        if (!mouseInside)

            return;

        // Mouse hover colour

        m\_Rect.setFillColor(sf::Color(0x6E2F9DFF));

        // Change state depending on what the button should do

        if (Program::s\_Instance->WasMousePressed())

        {

            // Call whatever function this button is linked to

            m\_ParentState->recentButton = this;

            std::invoke(m\_ClickFunction, \*m\_ParentState);

        }

    }

    void Button::draw(sf::RenderTarget &target, sf::RenderStates states) const

    {

        target.draw(m\_Rect, states);

        target.draw(m\_Text, states);

    }

    void Button::CenterText()

    {

        // Center Text in rectangle due to position being top left corner of object

        sf::Vector2f textPosition = m\_Rect.getPosition() + 0.5f \* (m\_Rect.getSize() - m\_Text.getGlobalBounds().getSize());

        textPosition.y -= m\_Text.getCharacterSize() >> 2;

        m\_Text.setPosition(textPosition);

    }

    void Button::CenterTextLeft()

    {

        // Center Text in rectangle due to position being top left corner of object

        sf::Vector2f textPosition = m\_Rect.getPosition();

        textPosition.x += 5.0f;

        textPosition.y += 0.5f \* (m\_Rect.getSize().y - m\_Text.getGlobalBounds().getSize().y) - (m\_Text.getCharacterSize() >> 2);

        m\_Text.setPosition(textPosition);

    }

}

Code Snippet – Button Class Definition – [Connect 4/src/Graphics/Button.cpp]

### Text

#pragma once

#include <string>

#include <SFML/Graphics.hpp>

namespace Connect

{

    class Text : public sf::Drawable

    {

    public:

        Text(sf::Vector2f position = sf::Vector2f(0.0f, 0.0f), std::string text = "Default Text");

        virtual ~Text();

    public:

        void SetPosition(sf::Vector2f position);

        void SetCharacterSize(unsigned int size);

        void SetText(std::string text);

    private:

        // Override draw method to draw both rect and text

        void draw(sf::RenderTarget &target, sf::RenderStates states) const;

    private:

        sf::Text m\_Text;

    };

}

Code Snippet – Text Class Declaration – [Connect 4/src/Graphics/Text.h]

The Text class inherits from an SFML drawable object allowing it to easily be drawn to the screen by overriding the SFML draw method.

#include "Text.h"

#include "Tools/Logger.h"

#include "Program/Program.h"

namespace Connect

{

    Text::Text(sf::Vector2f position, std::string text)

    {

        m\_Text.setFont(\*Program::s\_Font);

        SetPosition(position);

        SetCharacterSize(24);

        SetText(text);

        LOG\_TRACE("Title object created");

    }

    Text::~Text() {}

    void Text::SetPosition(sf::Vector2f position)

    {

        m\_Text.setPosition(position);

    }

    void Text::SetCharacterSize(unsigned int size)

    {

        m\_Text.setCharacterSize(size);

    }

    void Text::SetText(std::string text)

    {

        m\_Text.setString(text);

    }

    void Text::draw(sf::RenderTarget &target, sf::RenderStates states) const

    {

        target.draw(m\_Text);

    }

}

Code Snippet – Text Class Definition – [Connect 4/src/Graphics/Text.cpp]

### Text Input

#pragma once

#include <string>

#include <SFML/Graphics.hpp>

#include <SFML/System.hpp>

#include "Program/State.h"

namespace Connect

{

    class TextInput : public sf::Drawable

    {

    public:

        TextInput(State \*parentState, bool passwordField = false, std::string text = "Default Text");

        virtual ~TextInput();

    public:

        void SetPosition(sf::Vector2f position);

        void SetSize(sf::Vector2f size);

        void SetText(std::string text);

        void KeyboardEvent(void (State::\*functionPointer)() = nullptr);

        void OnMouseUpdate(sf::RenderWindow \*window);

        inline const std::string& GetInput() { return m\_UserInput; }

    private:

        // Override draw method to draw both rect and text

        void draw(sf::RenderTarget &target, sf::RenderStates states) const;

        // Recenter Text

        void CenterTextLeft();

    private:

        sf::RectangleShape m\_Rect;

        sf::Text m\_Text;

        std::string m\_UserInput;

        std::string m\_DefaultText;

        bool m\_Active = false;

        bool m\_PasswordInputField = false;

        State\* m\_ParentState = nullptr;

    };

}

Code Snippet – Text Input Class Declaration – [Connect 4/src/Graphics/TextInput.h]

SFML does not provide any object to input text into so I had to implement my own using the information SFML does collect. This object is almost the same as a Text object except when it is selected, any keyboard inputs will be directed to it.

#include "TextInput.h"

#include <functional>

#include "Tools/Logger.h"

#include "Program/Program.h"

namespace Connect

{

    TextInput::TextInput(State \*parentState, bool passwordField, std::string text)

        : m\_ParentState(parentState), m\_PasswordInputField(passwordField), m\_UserInput(""), m\_DefaultText(text)

    {

        // Set default style and position

        m\_Rect.setFillColor(sf::Color(0x5E1F8DFF));

        m\_Rect.setPosition(sf::Vector2f(0.0f, 0.0f));

        m\_Rect.setSize(sf::Vector2f(100.0f, 100.0f));

        m\_Text.setFont(\*Program::s\_Font);

        m\_Text.setString(m\_DefaultText);

        m\_Text.setCharacterSize(24);

        m\_Text.setFillColor(sf::Color(0xD9D9D9FF));

        LOG\_TRACE("Text Input Field object created");

        CenterTextLeft();

    }

    TextInput::~TextInput()

    {

    }

    void TextInput::SetPosition(sf::Vector2f position)

    {

        m\_Rect.setPosition(position);

        CenterTextLeft();

    }

    void TextInput::SetSize(sf::Vector2f size)

    {

        m\_Rect.setSize(size);

        CenterTextLeft();

    }

    void TextInput::SetText(std::string text)

    {

        // Default text cannot be deleted

        m\_DefaultText = text;

        m\_Text.setString(m\_DefaultText);

        CenterTextLeft();

    }

    void TextInput::KeyboardEvent(void (State::\*functionPointer)())

    {

        // Make sure the key pressed is available

        if (!m\_Active || Program::s\_KeyPressed == sf::Keyboard::Unknown)

            return;

        // Make sure that the key pressed was alphabetic

        if (Program::s\_KeyPressed >= sf::Keyboard::A && Program::s\_KeyPressed <= sf::Keyboard::Z)

            m\_UserInput += 'a' + Program::s\_KeyPressed;

        // If backspace was pressed removed last character

        if (Program::s\_KeyPressed == sf::Keyboard::BackSpace && m\_UserInput.size() > 0)

            m\_UserInput.pop\_back();

        // Display inputted text

        if (m\_PasswordInputField)

            m\_Text.setString(m\_DefaultText + std::string(m\_UserInput.size(), '\*'));

        else

            m\_Text.setString(m\_DefaultText + m\_UserInput);

        CenterTextLeft();

        // Call function whenever text has changed

        if (functionPointer)

            std::invoke(functionPointer, \*m\_ParentState);

    }

    void TextInput::OnMouseUpdate(sf::RenderWindow \*window)

    {

        // Check mouse is inside rectangle

        bool mouseInside = m\_Rect.getGlobalBounds().contains(sf::Vector2f(sf::Mouse::getPosition(\*window)));

        // Nice highlighting for on hover

        if (mouseInside || m\_Active)

            m\_Rect.setFillColor(sf::Color(0x6E2F9DFF));

        else

            m\_Rect.setFillColor(sf::Color(0x5E1F8DFF));

        // Change state depending on what the button should do

        if (Program::s\_Instance->WasMousePressed())

        {

            if (mouseInside)

            {

                m\_Active = true;

                LOG\_TRACE("Text Input Field is active");

                m\_Rect.setFillColor(sf::Color(0x6E2F9DFF));

            }

            else if (m\_Active)

            {

                m\_Active = false;

                LOG\_TRACE("Text Input Field is deactivated");

                m\_Rect.setFillColor(sf::Color(0x5E1F8DFF));

            }

        }

    }

    void TextInput::draw(sf::RenderTarget &target, sf::RenderStates states) const

    {

        target.draw(m\_Rect, states);

        target.draw(m\_Text, states);

    }

    void TextInput::CenterTextLeft()

    {

        // Center Text in rectangle due to position being top left corner of object

        sf::Vector2f textPosition = m\_Rect.getPosition();

        textPosition.x += 5.0f;

        textPosition.y += 0.5f \* (m\_Rect.getSize().y - m\_Text.getGlobalBounds().getSize().y) - (m\_Text.getCharacterSize() >> 2);

        m\_Text.setPosition(textPosition);

    }

}

Code Snippet – Text Input Class Definition – [Connect 4/src/Graphics/TextInput.cpp]

# Testing

# Evaluation

1. <https://en.wikipedia.org/wiki/Machine_learning> [↑](#footnote-ref-1)
2. <https://en.wikipedia.org/wiki/Neural_network> [↑](#footnote-ref-2)
3. <https://youtu.be/gn4nRCC9TwQ> [↑](#footnote-ref-3)
4. [https://deepmind.com](file:///C:\Users\josep\AppData\Local\Packages\Microsoft.Office.Desktop_8wekyb3d8bbwe\LocalCache\Roaming\Microsoft\Word\.git) [↑](#footnote-ref-4)
5. <https://en.wikipedia.org/wiki/Go_(game)> [↑](#footnote-ref-5)
6. <https://www.ignite.digital/10-best-programming-languages-to-learn-in-2020/> [↑](#footnote-ref-6)
7. <https://raygun.com/blog/programming-languages/> [↑](#footnote-ref-7)
8. <https://en.wikipedia.org/wiki/Standard_Template_Library> [↑](#footnote-ref-8)
9. <https://en.wikipedia.org/wiki/Off-by-one_error> [↑](#footnote-ref-9)
10. <https://en.wikipedia.org/wiki/Levenshtein_distance> [↑](#footnote-ref-10)
11. <https://en.wikipedia.org/wiki/Soundex> [↑](#footnote-ref-11)
12. <https://www.tutorialspoint.com/java/java_polymorphism> [↑](#footnote-ref-12)
13. <https://www.investopedia.com/terms/n/neuralnetwork.asp> [↑](#footnote-ref-13)
14. <https://www.reddit.com/r/math/comments/1lo4od/how_many_games_of_connect4_there_are/> [↑](#footnote-ref-14)
15. <https://en.wikipedia.org/wiki/Flowchart> [↑](#footnote-ref-15)
16. <https://www.lucidchart.com/> [↑](#footnote-ref-16)
17. <https://en.wikipedia.org/wiki/Log_file> [↑](#footnote-ref-17)
18. <http://www.cplusplus.com/doc/tutorial/preprocessor/> [↑](#footnote-ref-18)
19. <https://www.tutorialspoint.com/java/java_polymorphism> [↑](#footnote-ref-19)
20. <https://en.cppreference.com/w/cpp/language/virtual> [↑](#footnote-ref-20)
21. <https://youtu.be/aircAruvnKk> [↑](#footnote-ref-21)
22. <https://www.investopedia.com/terms/o/overfitting.asp> [↑](#footnote-ref-22)
23. <https://en.wikipedia.org/wiki/Sigmoid_function> [↑](#footnote-ref-23)
24. <https://en.wikipedia.org/wiki/Supervised_learning> [↑](#footnote-ref-24)
25. <https://en.wikipedia.org/wiki/Derivative> [↑](#footnote-ref-25)
26. <https://en.wikipedia.org/wiki/Chain_rule> [↑](#footnote-ref-26)
27. <https://dev.exiv2.org/projects/exiv2/wiki/The_Metadata_in_PNG_files> [↑](#footnote-ref-27)
28. <https://www.reddit.com/r/math/comments/1lo4od/how_many_games_of_connect4_there_are/> [↑](#footnote-ref-28)
29. <https://www.loginradius.com/blog/async/eol-end-of-line-or-newline-characters/> [↑](#footnote-ref-29)
30. <https://www.youtube.com/watch?v=sd3Lg5qgTzw> [↑](#footnote-ref-30)
31. <https://en.wikipedia.org/wiki/ASCII> [↑](#footnote-ref-31)
32. <https://en.wikipedia.org/wiki/Cryptographic_hash_function> [↑](#footnote-ref-32)
33. <https://en.wikipedia.org/wiki/SHA-1> [↑](#footnote-ref-33)
34. <https://github.com/> [↑](#footnote-ref-34)
35. <https://youtu.be/X8jsijhllIA> [↑](#footnote-ref-35)
36. <https://www.omnisci.com/technical-glossary/graphical-user-interface> [↑](#footnote-ref-36)
37. <https://www.sfml-dev.org/> [↑](#footnote-ref-37)
38. <https://en.wikipedia.org/wiki/Finite-state_machine> [↑](#footnote-ref-38)
39. <https://en.cppreference.com/w/cpp/language/destructor> [↑](#footnote-ref-39)
40. <https://www.usability.gov/what-and-why/user-interface-design.html> [↑](#footnote-ref-40)
41. <https://www.geeksforgeeks.org/quick-sort/> [↑](#footnote-ref-41)
42. <https://www.geeksforgeeks.org/merge-sort/> [↑](#footnote-ref-42)
43. <https://en.wikipedia.org/wiki/Radix_sort> [↑](#footnote-ref-43)
44. <https://en.wikipedia.org/wiki/Levenshtein_distance> [↑](#footnote-ref-44)
45. <https://en.wikipedia.org/wiki/Soundex> [↑](#footnote-ref-45)
46. <https://en.wikipedia.org/wiki/Hash_table> [↑](#footnote-ref-46)
47. <https://www.visual-paradigm.com/guide/uml-unified-modeling-language/what-is-class-diagram/> [↑](#footnote-ref-47)
48. <https://medium.com/better-programming/code-against-interfaces-not-implementations-37b30e7ab992> [↑](#footnote-ref-48)
49. <https://www.lucidchart.com/> [↑](#footnote-ref-49)
50. <https://en.wikipedia.org/wiki/Unified_Modeling_Language> [↑](#footnote-ref-50)
51. <https://github.com/SirDisco/NEA-Coursework> [↑](#footnote-ref-51)
52. <https://cmake.org/> [↑](#footnote-ref-52)
53. <https://en.wikipedia.org/wiki/Singleton_pattern> [↑](#footnote-ref-53)