

---

# BACK TO THE CLIMATE

---

*Software combining 3D elements  
with statistical regression  
techniques to present predictions  
of changes in the climate  
worldwide*

Rishita Rallabandi (2024)  
Candidate Number: 9428  
Centre Number: 51411

# Table of Contents

<b><i>Table of Figures.....</i></b>	<b><i>6</i></b>
<b><i>Section 1: Analysis.....</i></b>	<b><i>7</i></b>
<b>Overview: Problem Definition.....</b>	<b>7</b>
<b>Intended Client &amp; Background .....</b>	<b>7</b>
Details .....	7
Need for Software .....	7
Background.....	8
User Identification .....	8
<b>Research: Similar Systems.....</b>	<b>9</b>
Description of system.....	9
Advantages of system.....	10
Disadvantages of System .....	11
<b>Research: Interview with Client .....</b>	<b>12</b>
General Questions.....	13
Specific Questions.....	14
Interview Summary/Interpretation .....	15
<b>Objectives.....</b>	<b>16</b>
<b>Proposed Solutions .....</b>	<b>18</b>
Comparison.....	18
Conclusion.....	19
<b>Research: Time series forecasting .....</b>	<b>19</b>
Autoregressive & Automated Methods .....	20
Deep Learning & Neural Networks .....	21
Comparison.....	21
<b>Programming Language(s) .....</b>	<b>22</b>
<b>System Requirements .....</b>	<b>23</b>
Software Requirements .....	23
Hardware Requirements .....	23
<b>Modelling: Formulae .....</b>	<b>24</b>
Linear Regression .....	24
Latitude and Longitude .....	25

Spherical Coordinate Transformation .....	25
DMS Coordinates to Decimal Degrees .....	25
Inverse Distance Weighting (IDW) Interpolation .....	26
Haversine formula .....	26
<b>Section 2: Design .....</b>	<b>28</b>
<b>Overview .....</b>	<b>28</b>
<b>Page Navigation .....</b>	<b>28</b>
<b>User Interface .....</b>	<b>29</b>
Sketches.....	29
Registration Data Table.....	31
<b>Class Diagram .....</b>	<b>35</b>
<b>Flowcharts of Proposed System .....</b>	<b>37</b>
Login .....	38
Coordinate Lookup Feature .....	39
Export Report .....	40
<b>Data Tables .....</b>	<b>41</b>
Employee Database (existing) .....	41
Climate Data .....	41
<b>File Structures .....</b>	<b>42</b>
All Cleaned Climate Data (CSV File).....	42
Linear Regression Results (CSV Files) .....	43
<b>Formulae Code .....</b>	<b>44</b>
Linear Regression Code.....	44
Degrees to Radians Conversion Code .....	45
Spherical Coordinate Conversion Code.....	45
Conversion of DMS coordinates to Decimal Degrees Code .....	45
Inverse Distance Weighting Code .....	46
Haversine Distance Calculation Code.....	46
<b>Tested Algorithm.....</b>	<b>47</b>
Weak Plots.....	49
Strong Linear Regression Plots.....	50
<b>Section 3: Technical Solution .....</b>	<b>51</b>
<b>Overview .....</b>	<b>51</b>

<b>Linear Regression.....</b>	<b>53</b>
Python Code .....	53
Code summary.....	55
<b>Skybox Rotation .....</b>	<b>56</b>
C# Code.....	56
Summary.....	56
<b>Exit Program .....</b>	<b>56</b>
<b>Switch Between Scenes .....</b>	<b>57</b>
Summary.....	58
<b>Login &amp; Registration System.....</b>	<b>59</b>
SQL for Database .....	59
Login Front-End.....	59
Register Front-End .....	60
Web Server Connection.....	62
Login php code (link to web server) .....	64
Register php code (link to web server) .....	66
<b>Earth Plots &amp; Climate Variables .....</b>	<b>68</b>
Summary of Code .....	74
<b>Coordinate Lookup System .....</b>	<b>75</b>
Decimal Latitude & Longitude System .....	75
Code Summary .....	76
Decimal, Minutes, Seconds System .....	77
<b>Export Report .....</b>	<b>79</b>
C# Code.....	79
Summary of Code .....	85
<b>Keyboard Controls .....</b>	<b>86</b>
Zoom In and Out on Earth .....	86
Rotate Earth.....	87
<b>Focused Option.....</b>	<b>87</b>
<b>Hovering Text .....</b>	<b>88</b>
<b>Settings Menu .....</b>	<b>88</b>
Summary of Code .....	90
<b><i>Section 4: Testing.....</i></b>	<b><i>92</i></b>

<b>Overview .....</b>	<b>92</b>
<b>Input &amp; Output Testing Plans .....</b>	<b>92</b>
Login and Registration Testing.....	93
Earth Interactivity Testing.....	97
Navigation of Program Testing.....	97
Earth Visualisation Testing.....	100
Coordinate Lookup Testing .....	100
Export Report Testing.....	103
<b><i>Section 5: Evaluation .....</i></b>	<b><i>105</i></b>
<b>Overview .....</b>	<b>105</b>
<b>Objective Evaluation.....</b>	<b>105</b>
<b>Client Feedback.....</b>	<b>107</b>
Ease of use.....	107
How it meets the requirements .....	108
Criticisms .....	108
Improvements .....	108
<b>Self Reflection .....</b>	<b>108</b>
Challenges I had to overcome.....	108
Possible Improvements.....	109
What I learned .....	109
<b><i>Appendix.....</i></b>	<b><i>110</i></b>
<b>References .....</b>	<b>110</b>
<b>Resources Used to Code .....</b>	<b>111</b>

# Table of Figures

Figure 1 NASA Climate Time Machine: Home Page .....	9
Figure 2 NASA Climate Time Machine: Summary.....	10
Figure 3 NASA Climate Time Machine: Mobile format .....	11
Figure 4 NASA Climate Time Machine: Rudimentary map .....	11
Figure 5 NASA Climate Time Machine: Slow Loading of Time Series Images .....	12
Figure 6: Page Navigation Diagram.....	28
Figure 7: UI Sketch - Opening Scene .....	29
Figure 8: UI Sketch - Login Page .....	29
Figure 9: UI Sketch - Register Screen I.....	30
Figure 10: UI Sketch - Register Screen II .....	30
Figure 11: UI Sketch - Main Screen.....	32
Figure 12: UI Sketch - Settings Screen .....	33
Figure 13: UI Sketch - Settings Screen II .....	34
Figure 14: UI Sketch - Loading Screen .....	34
Figure 15: UI Sketch - Help Screen .....	34
Figure 16 Class Diagram .....	35
Figure 17: Flowchart Shapes.....	37
Figure 18: Login System Flowchart .....	38
Figure 19: Coordinate Lookup Flowchart .....	39
Figure 20: Export Report Flowchart.....	40

# Section 1: Analysis

## Overview: Problem Definition

In this increasing global urgency of climate change, businesses and organisations are struggling to find solutions to challenges involving adapting to and mitigating the impacts of environmental shifts. Day by day, more and more companies are switching to more sustainable methods with those who are not, being left behind. Hence, environmental consulting firms such as Sustainable Solutions Ltd. are growing with the need to provide precise and actionable insights to their clients. Environmental consultants provide advice and feedback to their clients on how they can minimise their negative impact on the environment.

Additionally, many clients who have initially had an interest in hiring a consultant from Sustainable Solutions Ltd. have lost interest due to how long it takes them to perform a geospatial analysis and eventually give an assessment to their client, so as a result they are seeking out new methods to improve efficiency.

Current environment predictive software lacks precision, customisability and could be too difficult to interpret - traditional climate data presentation methods are often too complex for stakeholders who may not have a background in environmental science. Hence, the client – Sustainable Solutions Ltd. – requires a piece of software which will display various climate-related parameters such as temperature, precipitation levels and possibly sea levels and how each will change in the future.

## Intended Client & Background

### Details

*Client:* Sustainable Solutions Ltd.

*Contact Name:* Jenn Blakesee

*Job Title:* Chief Sustainability Officer

### Need for Software

The Chief of Sustainability is seeking out software which can enhance the depth and precision of the firm's environmental assessments for their diverse portfolio of clients. This will aid them with giving more beneficial and accurate information, thus increasing the value and reputation of the company resulting in a raise in profit.

## **Background**

Sustainable Solutions Ltd. is an environmental consulting firm with the aim to provide comprehensive solutions to sustainability for businesses and organisations around the world. They have a focus on harnessing modern technology to provide tailored, data-driven insights to their clients. The company works with a wide range of clients from the energy, real estate, transportation, and agriculture, devoted to incorporating sustainable practices into their operations are among their clientele.

## **User Identification**

The main user/administrator will be Mrs Blakesee, my contact, as she is overseeing the performance of the employees and whether they are using this software appropriately and relevantly to aid their work. The employees (environmental consultants) will be the primary users as they will use the software to produce predicted data which will form the basis for their environmental analysis. The clients of the firm will be the secondary users of the software as they will benefit from the technology used by the software through an intermediary.



## Research: Similar Systems

### Description of system

One of the similar systems (Boeck, Jackson, & Shaftel, 2022), similar to my program idea, is made by NASA (National Aeronautics and Space Administration) The website features an interactive visualization that shows how some of the key indicators of climate change, such as temperature, sea ice extent, and carbon dioxide concentrations, have changed over time. The interactive aspect is the timeline which dates back to years as far as 1955 and visually shows the progression of the abundance of the series of parameters mentioned above on a 3D model or 2D map. (<https://climate.nasa.gov/interactives/climate-time-machine/>).

This home page (Figure 1) presents the user with a menu of 6 choices (the different climate variables) and each leads to an image or 3D figure of the earth with the magnitudes of the above parameters in select locations. This is done by loading a collection of time-series, collated to form this smooth visualisation of climate changes.



FIGURE 1 NASA CLIMATE TIME MACHINE: HOME PAGE

## Advantages of system

- 1) **This website has a simple and user-friendly interface** which is crucial for interactive climate websites. It means that the layout, navigation, and overall design are intuitive and not overly complicated. Users should be able to easily find their way around the website, access distinctive features, and interact with visualizations without confusion. Complex concepts and data should be presented in a clear and comprehensible manner, using language that is understandable to a broad audience. It achieves this by having a concise menu displaying 6 main 'topics' relating to climate change and having a succinct summary in each section explaining what is happening in simple terms as shown in

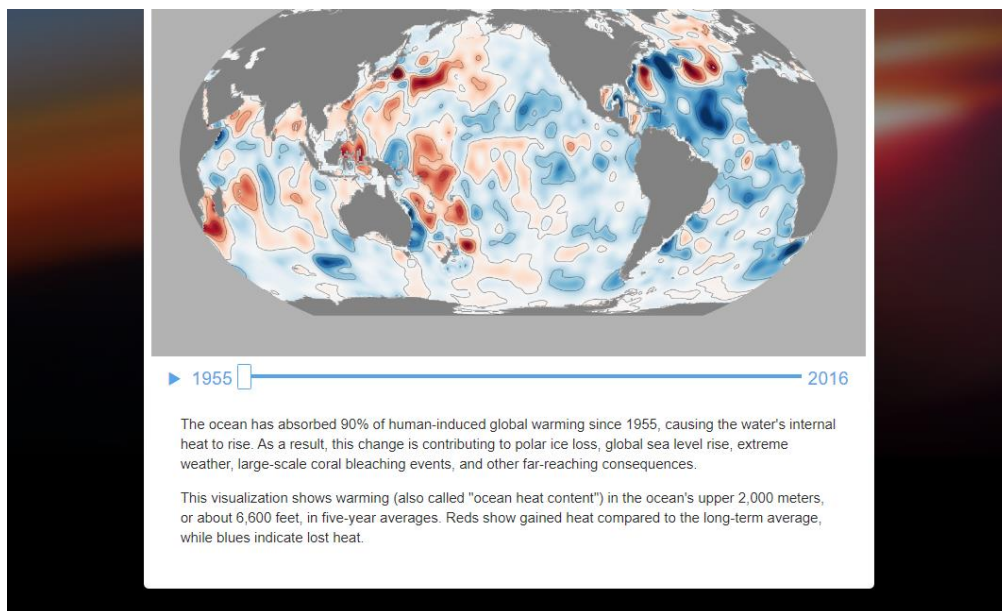


FIGURE 2 NASA CLIMATE TIME MACHINE: SUMMARY

Figure 2.

- 2) Using **accurate data** ensures that the information being displayed to users is trustworthy, contributing to the credibility of the website and the educational value it provides. Any data presented should be sourced from reputable and well-established climate research organizations.
- 3) **Accessible** by users on any device with access to the Internet as it doesn't require any downloads or add-ons. It is formatted in a user-friendly way not only for PC users but also for mobile users. (Figure 3)

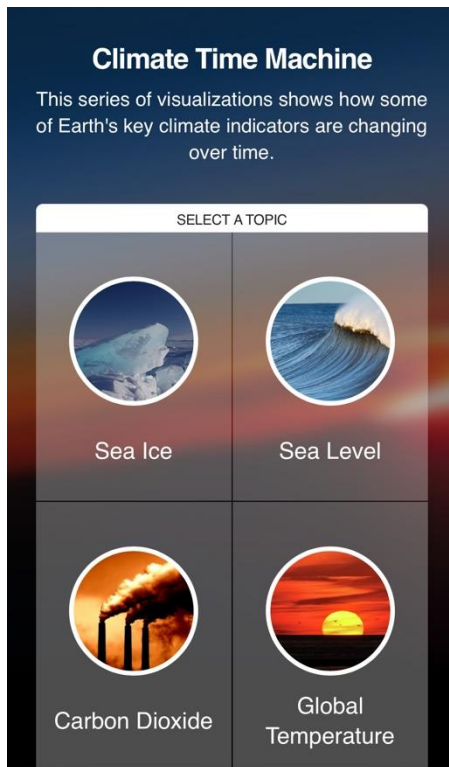


FIGURE 3 NASA CLIMATE TIME MACHINE: MOBILE FORMAT



This visualization shows the annual Arctic sea ice minimum since 1979. At the end of each summer, the sea ice cover reaches its minimum extent, leaving what is called the perennial ice cover. The area of the perennial ice has been steadily decreasing since the satellite record began in 1979.

Data source: Satellite observations.

Credit: [NASA Scientific Visualization Studio](#)

## Disadvantages of System

- 1) The **inability to pan** (move horizontally or vertically) **and zoom in** on the model could limit users' ability to explore the data in detail. Being able to zoom in allows users to focus on specific regions or time periods, enhancing their understanding of the data. Similarly, panning allows users to explore different areas of the Earth model, making the visualization more engaging and informative. As shown in Figure 5, certain maps are very rudimentary and lack detail.

- 2) If the website **restricts users to observing fixed locations** -- it limits the exploratory aspect of the interactive experience. Allowing users to choose and see various locations across the globe enhances their engagement

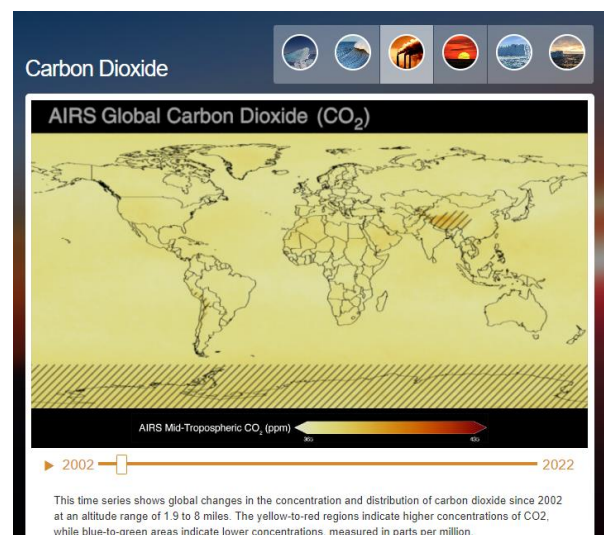


FIGURE 4 NASA CLIMATE TIME MACHINE: RUDIMENTARY MAP

and understanding. This is particularly relevant for climate change, where impacts can vary significantly by location.

- 3) This website **takes quite some time to load** in the 'time-series images' as shown in Figure 6. A slow-loading website can be frustrating for users and may discourage them from engaging with the content. This could be due to large file sizes, excessive use of graphics, or inefficient code. Optimizing the website's assets, compressing images, and minimizing unnecessary scripts can help improve loading times, enhancing the overall user experience.

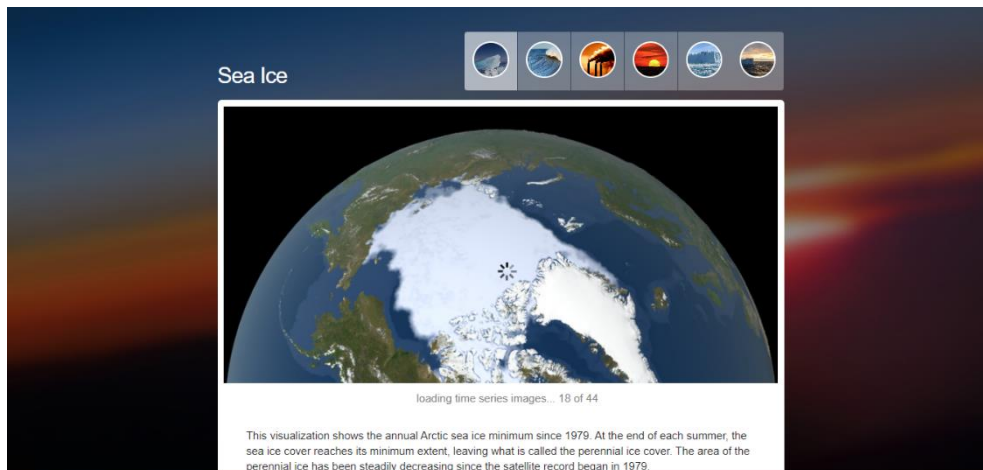


FIGURE 5 NASA CLIMATE TIME MACHINE: SLOW LOADING OF TIME SERIES IMAGES

- 4) Also, the 'Sea-level' feature only seems to show the effects of sea levels rising to 6 more metres. Though this is useful and intriguing, it doesn't fit the true purpose of a supposed 'time machine' and **doesn't give us a data based on time**. As shown below, the scale can only travel between 0 metres and 6 metres, not on a time scale. This means there is no information given as to when this sea level rise could occur.
- 5) **Doesn't predict future data** – this website only presents past data on the timeline, not future predicted data. Therefore, it doesn't fulfil the purpose of a Climate 'Time Machine'.

## **Research: Interview with Client**

I decided to ask Jenn Blakesee (COS) a few questions to derive exactly what features she and her firm require from this piece of software.

## General Questions

- 1) *What are the resources and software tools currently used by your firm for environmental consulting and climate analysis?*

We currently gather our data from the National Oceanic and Atmospheric Administration (NOAA). Recently, we have begun using Origin Lab to graph the data and visualise it.

- 2) *Does this current system complete all the tasks you require it to at the required standard?*

No it doesn't – as our firm is expanding we want to optimise the work load for our employees, improve efficiency, and reduce waste of time.

- 3) *What features about your current system do you find useful or beneficial?*

The data derived from NOAA is very reliable and detailed and for a couple of parameters like temperature and precipitation they even provide maps. Origin Lab is also a great open-source application with a range of complex functions and it can effectively present the data in great detail.

- 4) *What would you want to improve with your current system?*

Though Origin Lab is a very in-depth tool, it is very difficult to get a grasp of and I believe that slows our workflow down. The data from the NOAA is only restricted to the United States and as a firm we aim to expand on a global scale. So, I think we would prefer a simple, user-friendly interface which combines data collection with graphing.

- 5) *What are the most vital features you would like included?*

A minimum of 25 years of predicted climate data should be able to be displayed. Also, there must be feature where we could zoom in to specific coordinates by typing them in. It would be extremely useful to be able to export a report with all relevant data by entering a year and certain location.

- 6) *Which type of user-interface would be the most suitable for your firm? For example, a desktop application, web-page or API?*

For our firm, a desktop application would be the most effective as they could provide more privacy and security options and it would integrate better with our workflow.

### **Specific Questions**

*7) Would you like a form of user validation (e.g. username and password)?*

Yes, I want to ensure only employees of our firm would be able to access this program.

*8) Is there a way to uniquely identify each of your employees?*

Yes, we have an employee database which holds employee ID's for every single person currently employed at our firm.

*9) Which climate variables are important for your analyses?*

- Surface Temperature (Min and Max OR average)
- Precipitation levels
- Carbon dioxide concentrations (optional)
- Wind speed (optional)

*10) What temporal resolution is required for the predictive elements? e.g. hourly data, monthly or yearly?*

Yearly data would be preferred as we focus on the long term of climate change, so monthly data would be excessive and unnecessary.

*11) Would you like the model to show global, regional or local data?*

Preferably, the model should show all three types. Being able to move and zoom in to different areas on the Earth and also seeing it as a whole is essential.

*12) Should the timeline be intervals or continuous?*

Continuous would allow for more precision.

*13) Are there any accessibility requirements that could to be incorporated for your employees?*

Yes, this would be useful to add and any other features you can think of are welcome.

- Light and Dark modes to reduce eye strain

### **Interview Summary/Interpretation**

I gathered from this interview Mrs. Blakesee believes the current system is too time-consuming and requires an application that is essentially an 'all-in-one' package. They currently get data from the NOAA and reliable data is desirable. She also touched on having a user-friendly interface with accessibility features. They must be able to interact with the 3D model freely. There must be a feature to lookup coordinates and as a result the screen will zoom into that area on the earth. Additionally, an export report feature would add more value to the application as it would assist greatly in having not only visual data but also numerical data for further analysis.

## Objectives

### 1. New users must register and create an account to proceed to login

1. The user must be presented with a login screen with an option to register
2. The user must complete a registration form consisting of:
  1. Their first name
  2. Their surname
  3. Employee ID
  4. Password
3. The first, last name, and employee ID must be verified against an employee database, otherwise if not present in the employee database, registration does not proceed.
4. If an account has already been created with the same details, it must be flagged and user should be shown an error message
5. A unique and memorable username must be formed using the entered first name, surname and an ID
6. The user must enter a valid password with the following requirements:
  1. At least 8 characters long
  2. Capital and lowercase letters
  3. At least 1 special character e.g. @, !, ?, \$

### 2. Existing users must login before proceeding to the program

1. The login details entered by the user must be verified
2. Once login is successful, user is given immediate access to earth plotting program

### 3. The application must be easy to navigate and accessible

1. All buttons must be appropriately labelled with either a relevant symbol or text
2. When hovering over a symbol, text is shown to indicate the purpose for the button
3. A 'Help' screen must be available which displays all the controls and keyboard combinations
4. The toggles to activate the different parameters must be placed adjacent to each other
5. User should be able to adjust settings to ensure efficient and complete use of the program



1. User should be able to adjust the graphics quality in settings
2. User should be able to adjust the resolution through settings
3. User should be able to toggle full screen on and off through settings
6. In the main screens, the user should have the option to toggle a 'focused' mode where all elements on the screen except the necessary ones are hidden.

**4. The Earth model must be fully interactive for the user**

1. User must be able to zoom in and out
  1. Zoom in using Ctrl +
  2. Zoom out using Ctrl -
2. User must be able to rotate the earth model with arrow keys

**5. The levels of each climate variable must be presented visually on the 3D Earth model**

1. The climate variables/parameters that must be present are:
  1. Average annual surface temperature
  2. Average annual precipitation
  3. Annual snowfall
2. When presented on the model, the abundances should be indicated using a range of colours as points on the Earth - e.g. for temperature, red = high temperature, blue = low temperature
3. A legend should be shown indicating what colour a high or low level is and must be tailored to each climate parameter

**6. The user should be able to use the timeline to see not only past climate data but also future climate data**

1. A timeline or slider is to be used to enter the date in years
2. A least 25 years in the future should be shown
3. Future predictions must be feasible and formed from the extrapolation of reliable past data

**7. The user should be able to use a coordinate lookup feature to find the location on the Earth corresponding to the inputted coordinates**

1. User must enter a set of coordinates (either decimal degrees or DMS or other format)
2. The view of the Earth should change appropriately so that the location entered is zoomed in on

**8. The user must be able to export a report from this program**

1. The user must enter a year and a set of coordinates or latitude and longitude
2. A file must be produced consisting of:
  1. The year entered
  2. The location entered
  3. The average annual temperature, precipitation and snowfall
  4. The date and time of file generation

## ***Proposed Solutions***

I have arrived at 3 possible solutions to the defined problem and the requirements set by the client. I have listed them below with their positives and negatives to help me decide on one. All of the following solutions will build some type of machine learning model which will be integrated into Unity for its 3D aspects.

### **Comparison**

#	Proposed Solution Method	Advantages	Disadvantages
1	Supervised machine learning + Python	Relatively easy to train the model as algorithm knows what to look for	Very basic knowledge in python programming  Requires a lot of training time

2	TensorFlow + Python + time-series forecasting	High level of flexibility and scalability (Patel, 2018)  Comes with Tensor Board to visualise the machine learning	Steep learning curve for beginners (Rakshit, n.d.)  Time series forecasting is more computationally efficient and easier to implement.  Time series forecasting can be more challenging compared to others
3	ML.NET + C#	Will integrate seamlessly with the .NET ecosystem (Matec, 2023)  Already proficient in C# and C# is used in the Unity environment (and chosen for rest of A level)	Python is better suited to machine learning than C# (Beklemysheva, n.d.)  C# doesn't have as many libraries that can be used in conjunction

## Conclusion

I have settled on using Python with time series forecasting and a TensorFlow framework to produce a model used to predict future climate data. I have learned through many sources that Python tends to be a better programming language for creating machine learning models due to its consistency, simplicity and the range of libraries that can be used alongside it. A reason why I chose to use time series forecasting is that it isn't very computationally taxing and it is simply more appropriate for my type of program which focuses on predictions based on time.

## **Research: Time series forecasting**

Time series is a type of data where it measures the change in data over time. Time series forecasting (Lazzeri, 2020) is used by data scientists in a range of industries including manufacturing, finance and healthcare. This technique allows you to extrapolate insights such as trends and seasonality information for future operations.

## **Autoregressive & Automated Methods**

1. Autoregression:
  - Assumes that future observations depend linearly on past observations.
  - Each future data point is a linear combination of its own past values.
  - e.g. predicting tomorrow's temperature based on the temperatures from the past few days.
2. Moving Average:
  - Uses past forecast errors to predict future observations.
  - Considers the errors made in previous predictions to refine future predictions.
  - e.g. Adjusting a sales forecast based on how accurate or inaccurate recent forecasts have been.
3. Autoregressive Moving Average (ARMA):
  - Combines autoregression and moving average concepts.
  - Future observations depend on both past observations and past forecast errors.
  - e.g. Predicting stock prices using both the stock's past values and past prediction errors.
4. Autoregressive Integrated Moving Average (ARIMA):
  - Extends ARMA by incorporating differencing of observations.
  - Considers the differences between consecutive observations, making it suitable for non-stationary time series.
  - The model's goal is to predict by examining the differences between values in the series instead of actual values
  - e.g. Predicting monthly sales by taking into account the differences in sales from one month to the next.
5. Seasonal Autoregressive Integrated Moving Average with Exogenous Regressors (SARIMAX):
  - Extends ARIMA with seasonality and external factors.
  - It similarly uses past values but also considers any seasonality patterns
  - Considers both the trend and seasonality in the data, as well as the impact of external variables.

- e.g. Forecasting monthly sales considering not only historical patterns but also the effect of advertising spending.

6. Automated Machine Learning (Automated ML):

- Uses machine learning algorithms for time series forecasting with automated model selection and tuning.
- Iteratively tests various algorithms, tunes their parameters, and performs feature engineering to find the best model for the specific time series data.
- e.g. Automatically selecting and optimizing a model for predicting daily website traffic without manual intervention.

## **Deep Learning & Neural Networks**

Deep learning (Lazzeri, 2020) is a subset of machine learning which represents input data as vectors and transforms them utilising linear algebra in order to produce a given output. It is called 'deep' learning because it used artificial neural networks which consist of several layers including: inputs, outputs and hidden layers.

## **Comparison**

Autoregressive Methods	Deep Learning
Can use a small amount of data to form predictions	Requires a very large dataset
Doesn't need large amount of computational power – not many calculations or operations	Requires high-end computational power due to the abundance of matrix multiplication
Takes little time in comparison to deep learning	Takes a long time to train due to deep-learning algorithms having many layers
Can usually only output a numerical value or a classification	Can have a range of formats as outputs including sound, image and text

I have decided against using deep learning as I believe I do not have the appropriate resources or computational power to create neural networks. The only advantage deep learning has over autoregressive methods is that it can produce data not only in a numerical format but also sound and image, however I do not require this feature in my program. Not to mention, a study

(Reddy, Aneesh, Praneetha, & Vijay, 2021) was done on data science techniques and its precision with predicting global warming – it found that linear regression (ARIMA being a subset of it) obtains the 'highest precision for global warming and temperature'. Additionally, this project has a strict time-constraint, so taking that into account, autoregressive and automated methods (traditional methods of time series forecasting) would be more fitting.

As for the type of autoregressive and automated method, I have chosen Autoregressive Integrated Moving Average (ARIMA). This approach smooths the impact of outliers or transiently aberrant changes in the data, determines the significance of historical variations, and takes general trends into account. For this reason, ARIMA is ideal for capturing seasonality, unpredictability, and historical patterns – thus being well-suited to be used to predict climate data. (Lavanya, Ranjith, & Navaneetha Krishnan, 2022)

## ***Programming Language(s)***

The majority of my program will require me using C# as I am using Unity to not only create my 3D model but also create my user interface and the backbone of my program. C# is much easier to get a grasp of than languages like C++ and C# scripts are the code files storing behaviours in Unity (Lahoti, 2019). Additionally, I am already very familiar with C# through my A Level Computer Science course.

As for the machine learning aspect of my program, I would prefer using Python rather than C#. Though there are ways of developing machine learning models with C# such as using ML.NET,

## ***System Requirements***

These are the software and hardware requirements for me to develop this program

### **Software Requirements**

- Windows (7, 10 or 11), Linux or Mac Operating System
- Python interpreter (Python 3.5+)
- IDE: Visual Studio Code
- TensorFlow, Pandas, Keras Libraries
- Unity 3D C#

### **Hardware Requirements**

- Processor: Intel Core i5
- RAM capacity: 4GB+

## Modelling: Formulae

### Linear Regression

**Formula:**  $y = mx + c$

- ⇒ This is the equation that represents any straight line – this will be extrapolated in order to calculate future values as well as simplify the array of data

$$\textbf{Formula: } m = \frac{\Sigma(x - \bar{x})(y - \bar{y})}{\Sigma(x - \bar{x})^2}$$

- ⇒ 'm' represents the slope or gradient of the line
- ⇒ The gradient is calculated by taking the sum of the difference between the x values and their mean multiplied by the difference between the y values and their values. Then divide the sum by the sum of the squares of the difference between the x-value and the mean (Firdose, 2023).

$$\textbf{Formula: } r \text{ squared} = \frac{TSS - RSS}{TSS} = 1 - \frac{RSS}{TSS}$$

- ⇒ R-squared is the coefficient of determination and essentially it is a measure of how well the linear regression model fits the data (Enders, 2020).
- ⇒ The r-squared value in linear regression is used to assess how strong the correlation of the points on the graph are. Thus, it tells us how much the line fits our data from a range of 0 (no correlation) to 1 (fits data exactly).
- ⇒ In my program I will set a threshold of r-squared > 0.5 so that graphs with an r-squared value less than 0.5 are discarded as they are too weak.

$$RSS = \sum_{i=1}^n (y_i - \hat{y}_i)^2$$

- ⇒ RSS is the Residual Sum of Squares
- ⇒ This formula takes the sums of the squares of the difference between each y-value and the y- value according to the linear regression model.
- ⇒ N refers to the number of points on the graph,  $\hat{y}$  represents the predicted value according to the linear regression model



$$TSS = \sum_{i=1}^n (y_i - \bar{y}_i)^2$$

TSS is the Total Sum of Squares and it gives the total variation in y. TSS is calculated by summing the squared differences between each observation and the mean of all observed y-values.

### **Latitude and Longitude**

- ⇒ Theta  $\theta$  is the polar angle from the positive y-axis
- ⇒ Phi  $\phi$  is the azimuthal angle from the positive x-axis

$$\theta = \text{latitude} \times \frac{\pi}{180}$$

$$\phi = \text{longitude} \times \frac{\pi}{180}$$

- ⇒ These formulae simply convert latitude and longitude in degrees to radians to be used in the following formulae.

### **Spherical Coordinate Transformation**

Formulae are required to convert the latitude  $\phi$  in radians and longitude  $\theta$  in radians to XYZ cartesian coordinates to be plotted in Unity as points with coordinates (x, z, y). This transformation assumes that the Earth is a perfect sphere which is not entirely true as it is an ellipsoid, so there will be a few slight inaccuracies. However, it is a good approximation for the purpose of this program. (Nykamp, n.d.)

$$x = \text{radius} \times \cos \theta \times \cos \phi$$

$$y = \text{radius} \times \cos \theta \times \sin \phi$$

$$z = \text{radius} \times \sin \theta$$

### **DMS Coordinates to Decimal Degrees**

(Boddie, n.d.)

$$\text{decimal} = \text{degrees} + \frac{\text{minutes}}{60} + \frac{\text{seconds}}{3600}$$

## Inverse Distance Weighting (IDW) Interpolation

Inverse Distance Weighting is a type of multivariate interpolation which takes a set of scatter data point and assumes that closer points tend to be more similar and points further away are less similar – this assumption allows it to estimate the values of unknown values through interpolation. This is a popular interpolation algorithm for geospatial data which is what we are utilising in this program. (GIS Geography, 2023)

The weights assigned each of the points are determined by the inverse of the distance to the unknown or predicted point raised to the power of  $p$  (power parameter). The power parameter determines the significance of the known points – higher points highlight the significance of nearby points and are more detailed as well as less smooth. The default value of  $p$  is usually 2. The formula to determine the weights for each point  $i$  would be:

$$Weight_i = \frac{1}{distance(x, x_i)^p}$$

Calculating the distance between the two points will be done using the Haversine formula.

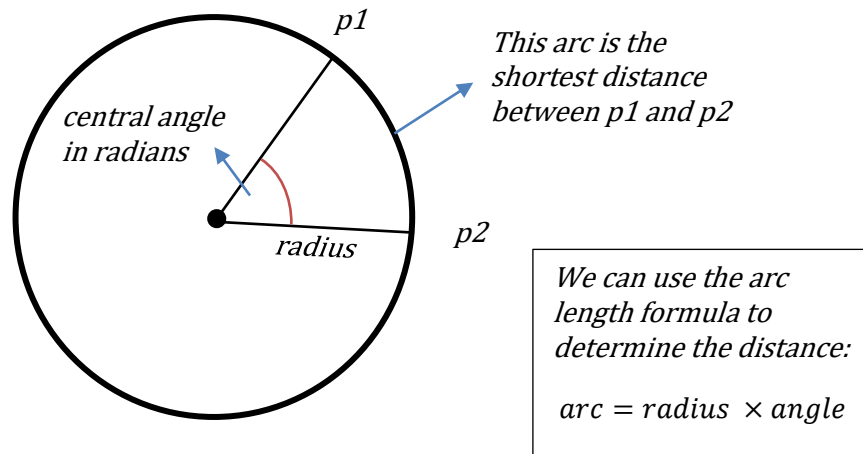
The interpolation formula involves taking a sum of these weights and the sum of the products of the weight of each point and the value (temperature, precipitation or snowfall) of each point. These sums are divided by each other to obtain the interpolated value.

$$interpolated\ value = \frac{\sum_{i=0}^n (weight_i \times value_i)}{\sum_{i=0}^n (weight_i)}$$

The  $n$  refers to the number of known scattered points and  $i$  refers to each of these points.

## Haversine formula

The Haversine formula is a way to calculate the shortest distance between two points on the surface of a sphere or more commonly, the Earth by taking the latitude and longitude values of each point (Kettle, 2017). The distance in metres can be calculated by modelling it as the arc of a sector. The diagram I created below can display the basis of this.



So this distance formula can be derived from this concept:

$$distance = 6378137 \times central\ angle$$

In order to calculate the central angle, we first use the haversine function to obtain the haversine. The haversine of an angle  $\theta$  would be:

$$hav(\theta) = \sin^2\left(\frac{\theta}{2}\right)$$

Let two points on the Earth with latitudes and longitudes be:  $(\phi_1, \lambda_1)$  and  $(\phi_2, \lambda_2)$  respectively. The haversine of the central angle  $\theta$  can be calculated directly from the latitudes and longitudes and can be written as this:

$$haversine(\theta) = haversine(\phi_2 - \phi_1) + \cos(\phi_1) \cos(\phi_2) haversine(\lambda_2 - \lambda_1)$$

Using the haversine function, we can expand this formula:

$$haversine(\theta) = \sin^2\left(\frac{\phi_2 - \phi_1}{2}\right) + \cos(\phi_1) \cos(\phi_2) \sin^2\left(\frac{\lambda_2 - \lambda_1}{2}\right)$$

The central angle can be calculated from this using the formula below where  $a$  represents the equation above:

$$\theta = 2 \times \arcsin(\sqrt{a})$$

## Section 2: Design

### Overview

When the program has been opened and loaded, the user will be presented with a 'title screen'. This system will be password-protected (requiring authentication) as the client requested that there should be measures to ensure only employees of their firm should have access to it. Thus, on the title screen, they will be presented with a button to login (or register). Once logged in and verified, they will be taken to the main screen. This will consist of the earth, timeline, a multitude of toggles for the different climate variables, general features (settings, information, profile management etc.) and the additional features requested by the client.

### Page Navigation

Using Microsoft Visio, I have created a page navigation diagram. This diagram aids me and the client in understanding how users will navigate through the application, helping to ensure a logical and intuitive user experience. By mapping out the relationships between different screens, features, and interactions, the page navigation diagram acts as a blueprint for me to implement seamless and efficient navigation within the software.

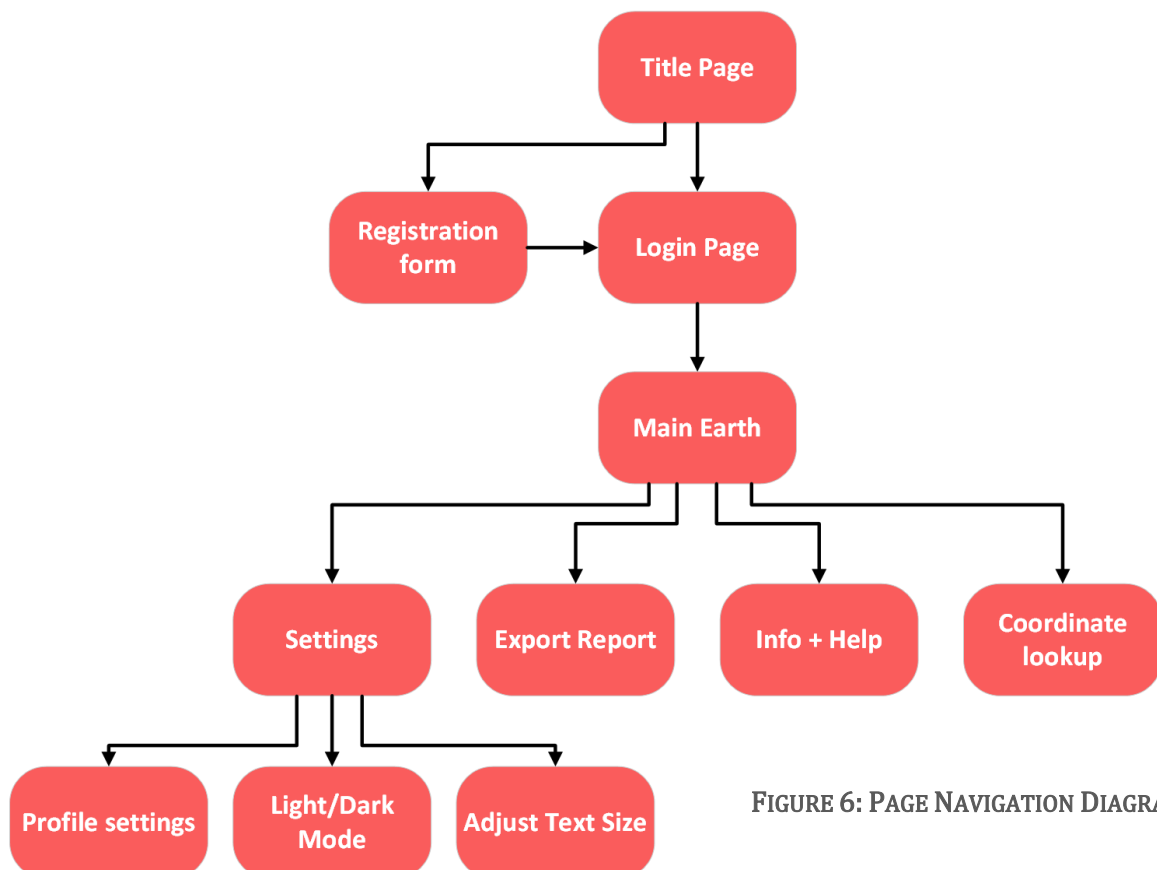


FIGURE 6: PAGE NAVIGATION DIAGRAM

## User Interface

### Sketches

I have created a few sketches of what I plan for the use interface of this system to look like. When the user opens the program, they will be presented with this title page below. This consists of: a background of the top of the revolving earth model; a label of the name of this program 'Back to the Climate'; a login button and register button (**Objective 1.1**). These buttons will lead to other pages.

FIGURE 7: UI SKETCH - OPENING SCENE



Below is the login page that is presented once the login button is pressed. This allows existing users to enter their given username and corresponding password to enter the system.

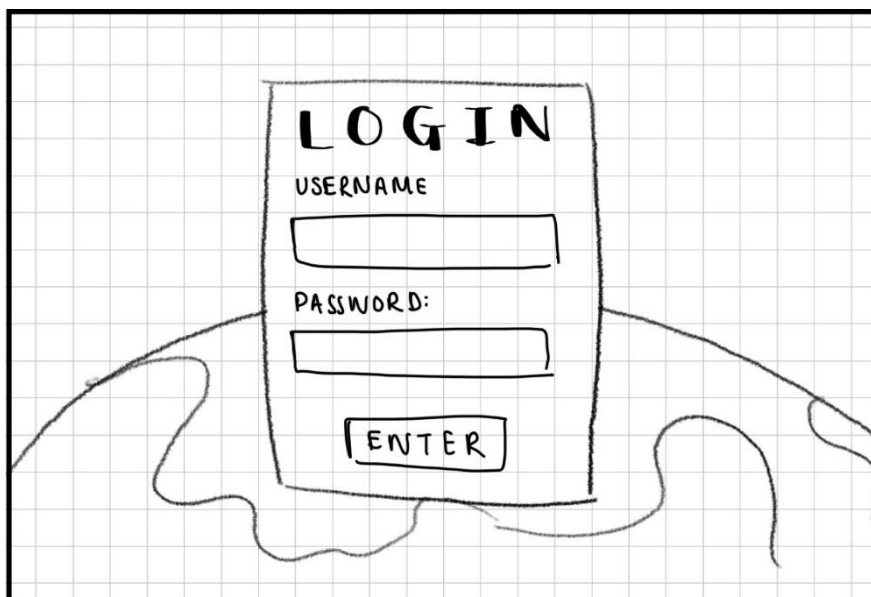
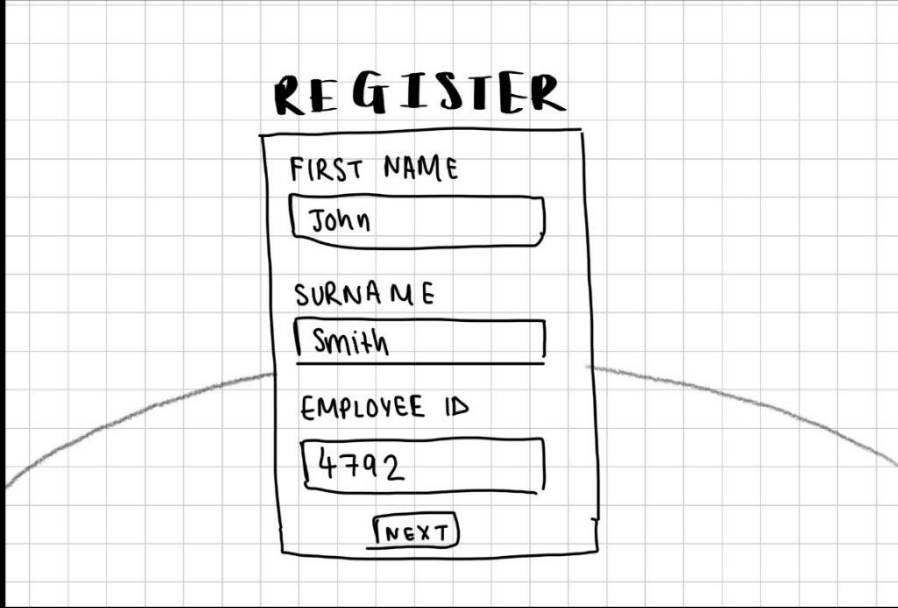


FIGURE 8: UI SKETCH - LOGIN PAGE

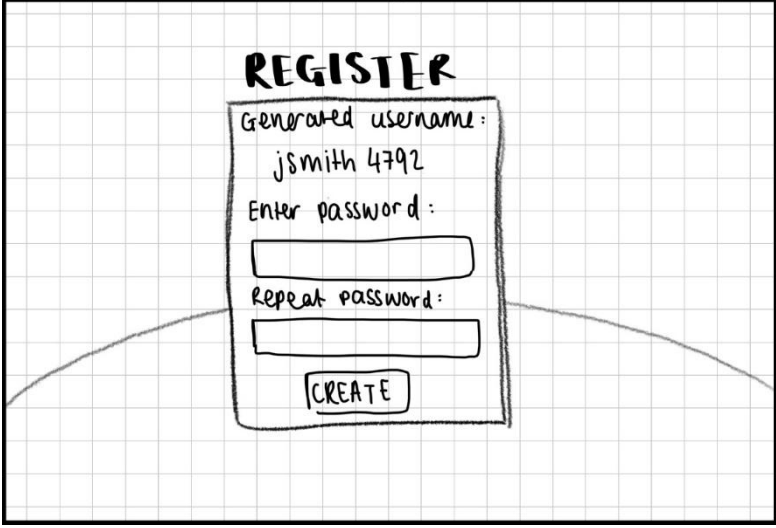
When creating an account for the first time, the user must first enter their name and employee ID. These details will then be verified against the employee database for the firm. This ensures that only employees of this firm will be able to access this software (Objective 1.2).



A hand-drawn UI sketch on a grid background. The title 'REGISTER' is at the top in bold, uppercase letters. Below it is a rectangular form containing three input fields. The first field is labeled 'FIRST NAME' and contains the text 'John'. The second field is labeled 'SURNAME' and contains the text 'Smith'. The third field is labeled 'EMPLOYEE ID' and contains the text '4792'. At the bottom of the form is a button labeled 'NEXT'.

FIGURE 9: UI SKETCH - REGISTER SCREEN I

Once the credentials have been verified against the employee database, a unique username will be generated from the inputted first name, last name and employee ID (Objective 1.5). Then the user can create a password for their account by entering it twice to ensure that no errors or typos were made. Once the account has been created the user can go back to the login screen to login and enter the program.



A hand-drawn UI sketch on a grid background. The title 'REGISTER' is at the top in bold, uppercase letters. Below it is a rectangular form. The first part of the form shows 'Generated username:' followed by the text 'jsmith 4792'. Below this are two input fields. The first is labeled 'Enter password:' and the second is labeled 'Repeat password:'. At the bottom of the form is a button labeled 'CREATE'.

FIGURE 10: UI SKETCH - REGISTER SCREEN II

### Registration Data Table

DATA ITEM	DATA TYPE	VALIDATIONS/RESTRICTIONS
First Name	string	<ul style="list-style-type: none"><li>• Required field</li><li>• Letters only</li></ul>
Last Name	string	<ul style="list-style-type: none"><li>• Required field</li><li>• Letters only</li></ul>
Employee ID	string	<ul style="list-style-type: none"><li>• Required field</li><li>• Numbers only</li><li>• No special characters</li><li>• 4 digits</li></ul>
Username	string	<ul style="list-style-type: none"><li>• Required field</li><li>• In format: first letter of first name, surname, employee ID</li></ul>
Password	string	<ul style="list-style-type: none"><li>• Minimum 8 characters</li><li>• Must include a special character</li><li>• Must include capital and lowercase characters (Objective 1.6)</li></ul>

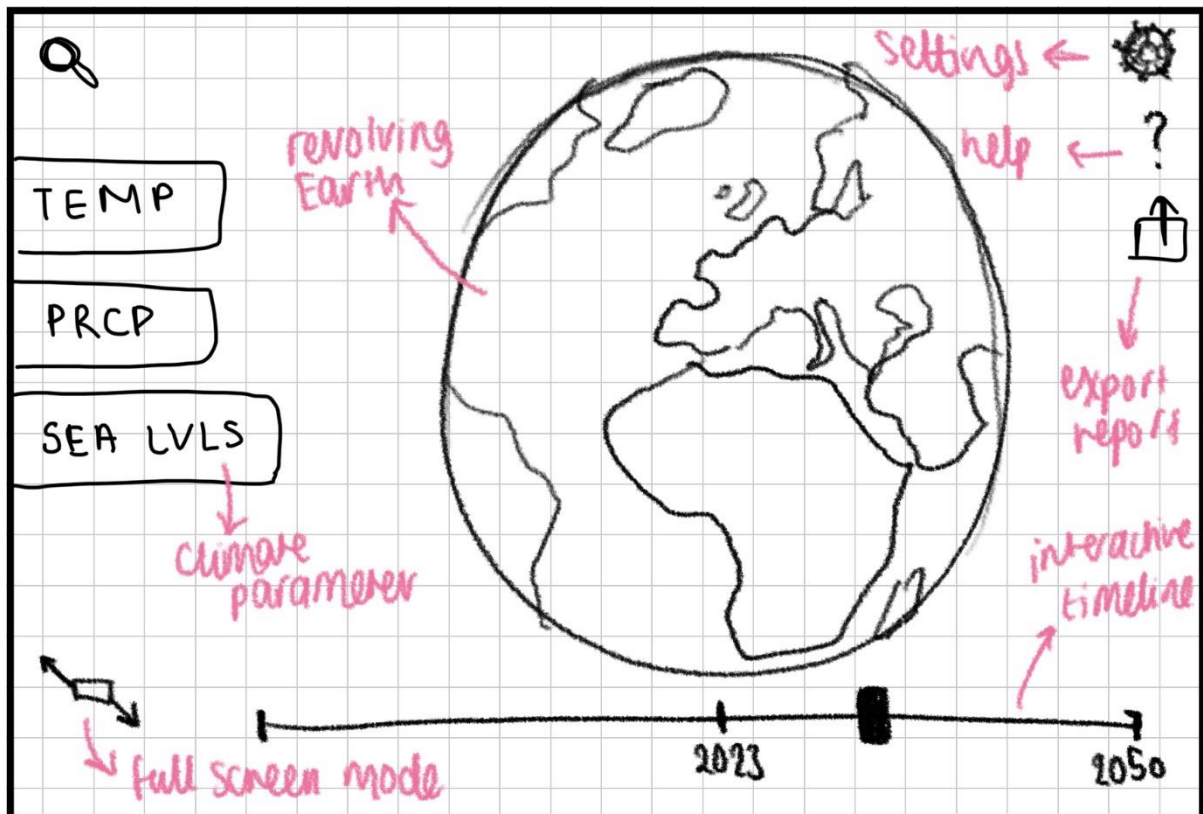


FIGURE 11: UI SKETCH - MAIN SCREEN

This is the main part of the program. It will contain all of the features which were the need for creating this software in the first place. At the centre, as shown by the annotated sketch, is a slowly revolving model of the earth. The revolving feature is simply an a design choice as it seems more appealing compared to a static, frozen model of the earth. I have laid out all of the additional features in a way to ensure that the program is easy to navigate (Objective 3.1). For example, placing all of the climate parameters close to one another (Objective 3.4) and general features like setting, help (Objective 3.3) and export adjacent to each other etc.

I expect the background to be similar to space – very dark with hints of blue and a few specks of white as stars. As shown, the timeline will have a bar that can be shifted left towards the past or right, towards the future all the way up to 2050 (27 years into the future). Adjacent to that, is a full-screen mode (Objective 3.7) button which hides all of the buttons including the parameters and general functions. This leaves the coordinate lookup button (Objective 4.3), an exit 'full screen mode' button the Earth and the timeline at the bottom.



What's shown below is a sketch of what the settings interface would look like – specifically the profile settings. This will allow users to change their password but not their username as that was specifically created by the system to be unique. This shows that the password can only be changed if the current password is entered. The other two sections shown are to switch between light and dark mode and to adjust text size.

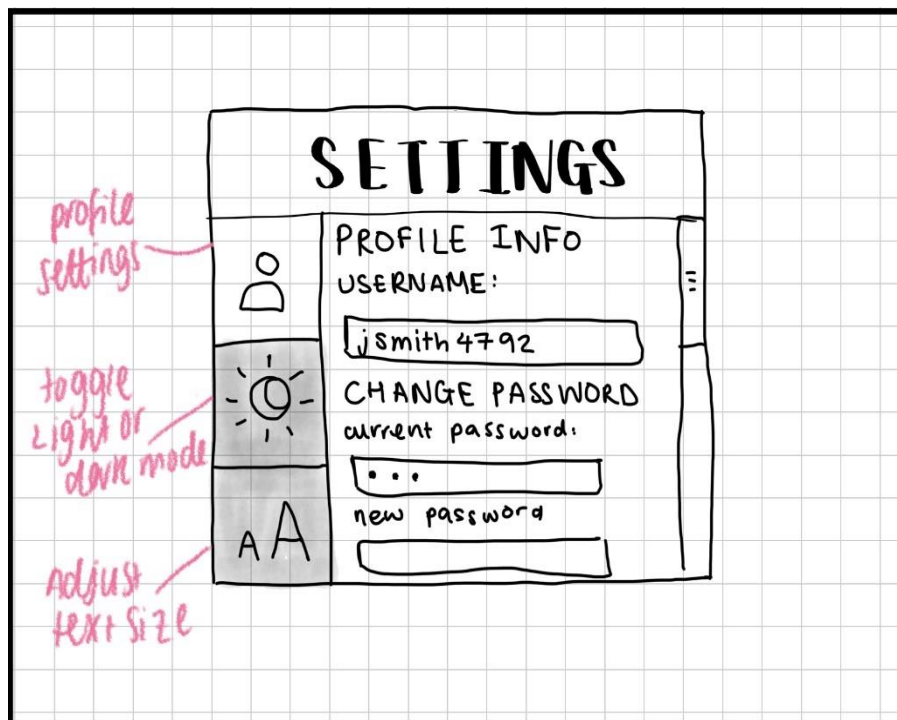


FIGURE 12: UI SKETCH - SETTINGS SCREEN

Here is a another sketch below of the setting page, however this shows the functionality to adjust the text sizes throughout the program as requested by the client. Here there is a slider with increments and a bar which can be moved left or right to decrease or increase the size, respectively.

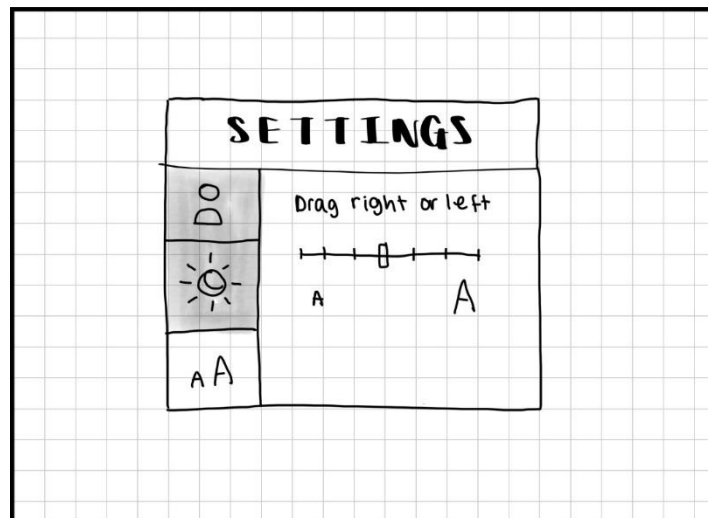


FIGURE 13: UI SKETCH - SETTINGS SCREEN II

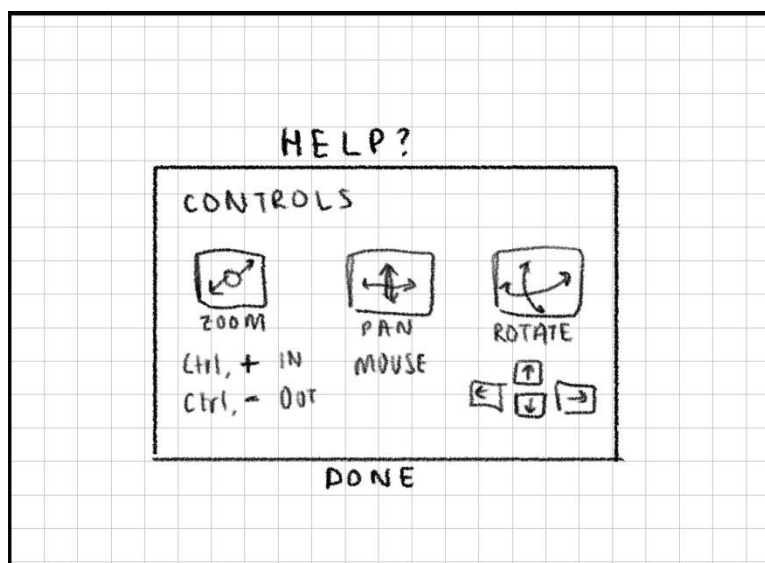


FIGURE 15: UI SKETCH - HELP SCREEN

On the left is what I imagine the help and information page will turn out to be. It will include instructions on how to interact with the earth model. This includes the key board combinations for each type of interaction: zooming in and out, panning and rotating the model.

Additionally, I sketched out a prototype for a loading screen which includes a miniature sized revolving earth. A loading screen enables the application to perform necessary initialization tasks in the background, ensuring that essential components are ready for use when the main interface is presented. Also, it helps maintain a consistent visual theme and layout, contributing to the overall design and aesthetics of the application

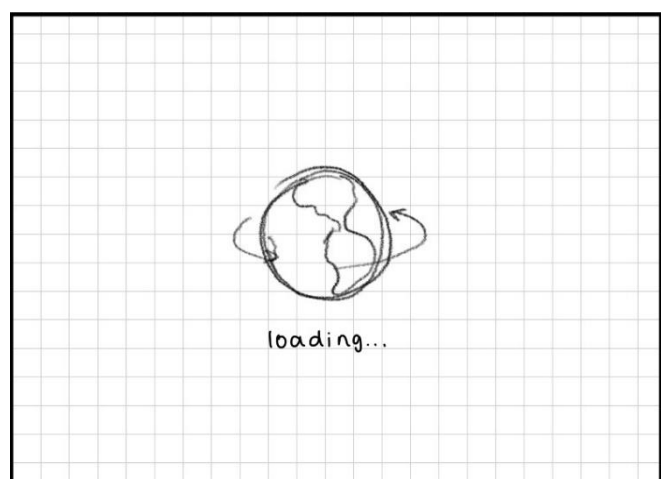


FIGURE 14: UI SKETCH - LOADING SCREEN

## Class Diagram

A class diagram is like a blueprint for a software system, using boxes to represent different types of things (classes) and lines to show how they're connected. It's a visual way to understand what pieces make up a program, what each piece does, and how they work together. This diagram helps software designers and developers communicate and make decisions about how to build a system, making the development process smoother and more organized.

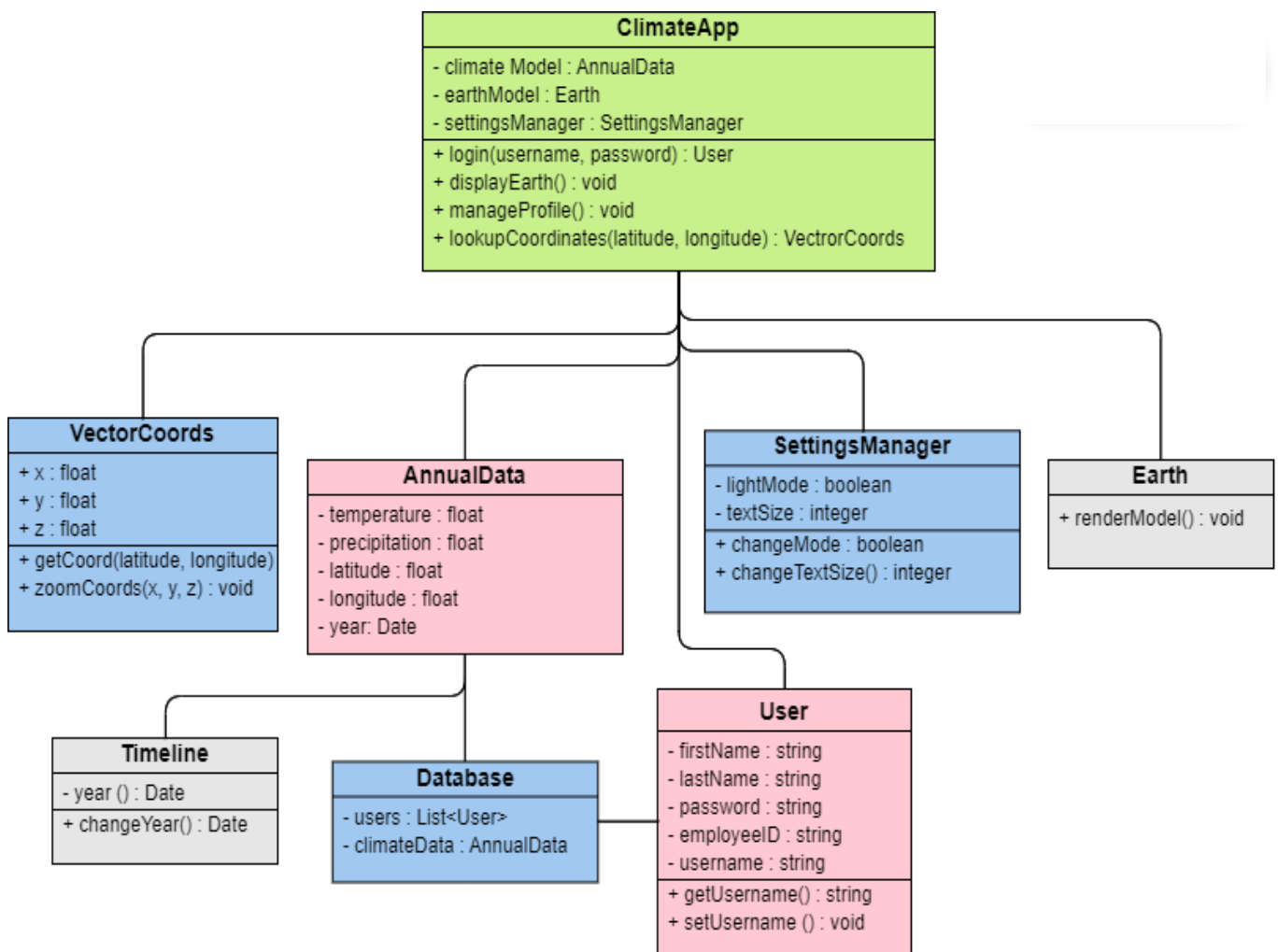


FIGURE 16 CLASS DIAGRAM

This class diagram I have created showcases the relationships and responsibilities of the core components and architecture of my proposed solution. At its core is the "ClimateApp" class, holding various components like the user database, climate model, earth model, and settings

manager. The "User" class encapsulates user information, linked to a "Database" class which implies user data storage. It also includes the methods to update user information – specifically the password associated with the account – as it is within the same class this also allows the database to be updated with the new password as the two classes are linked.

Settings management functionalities are delegated to the "SettingsManager" class which have the functionalities to switch between light and dark mode (I have marked this as a Boolean data type as there are two distinct values for this attribute) and changing text size. The "Earth" class handles the rendering of a 3D Earth model, associating with climate data. The precise methods of the relationships may further evolve based on the specific needs and design choices of the application, as this software development progresses.

## *Flowcharts of Proposed System*

I have created a series of flowcharts using Microsoft Visio Web to indicate the steps in my program. I have split the whole program into 'sub-systems' e.g. the login system. This will help break down the program into smaller steps which enable me to get a grasp of what I need to do easier.

Shapes I have used and what they indicate:



FIGURE 17: FLOWCHART SHAPES

## Login

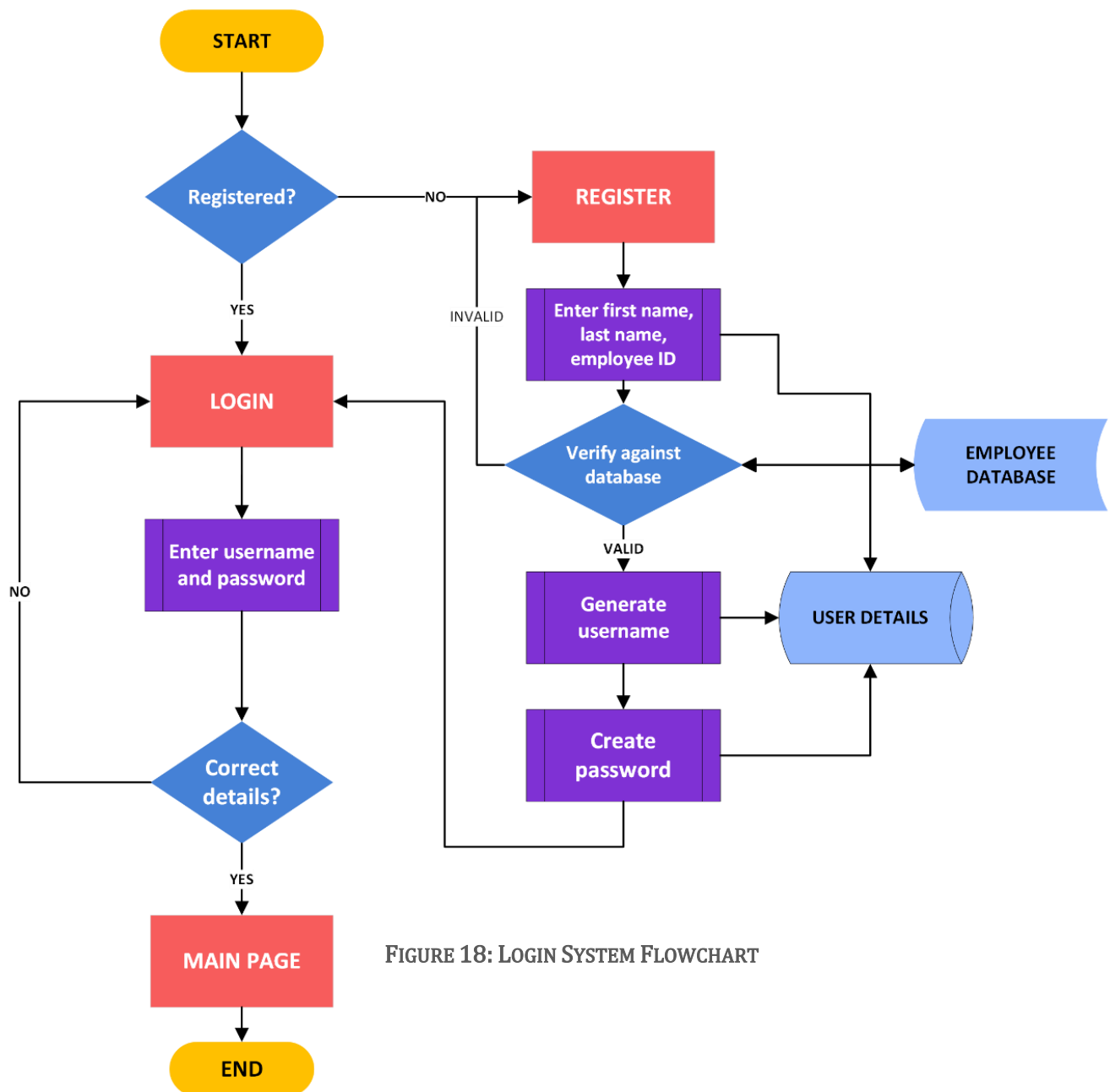
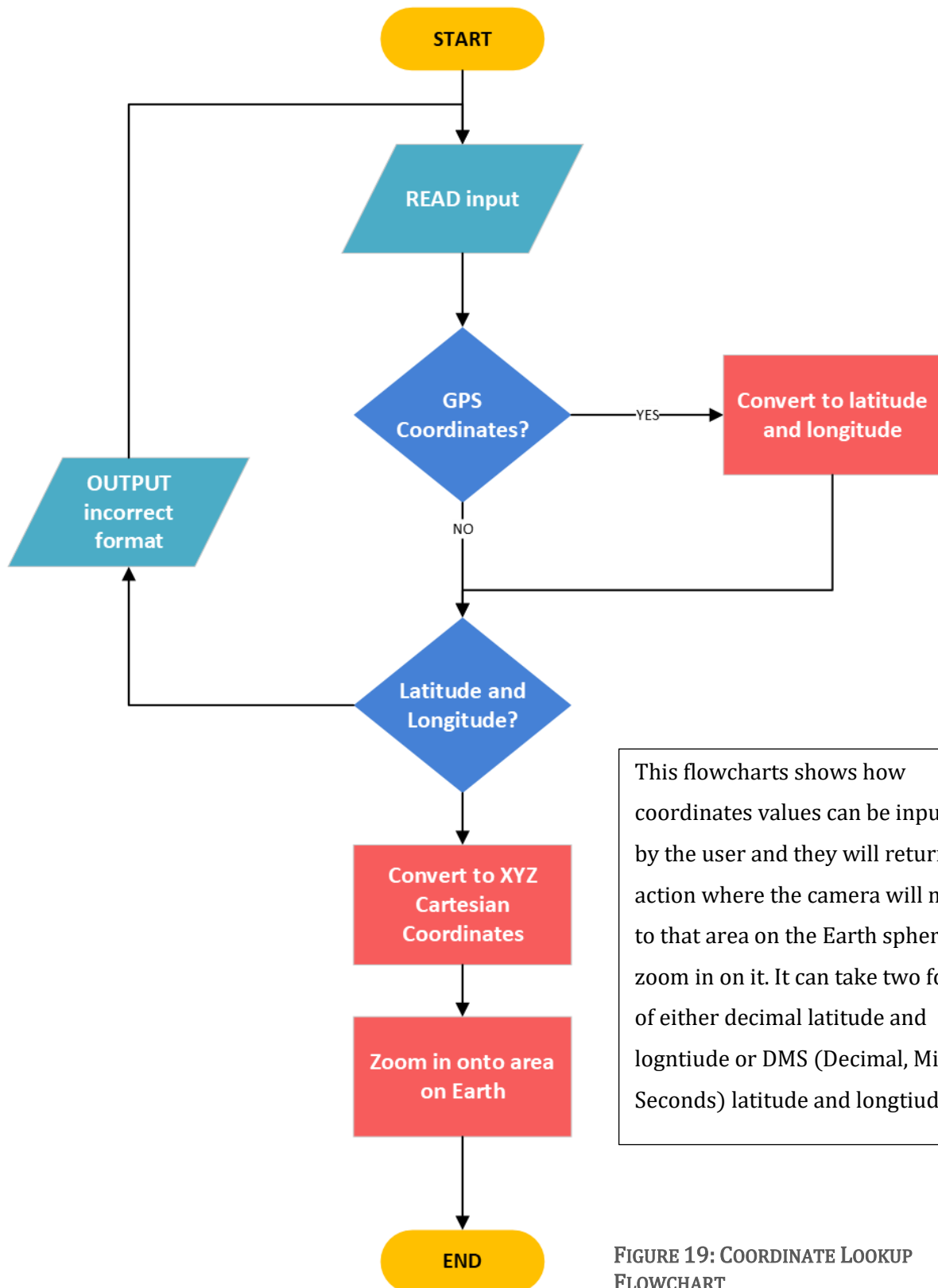


FIGURE 18: LOGIN SYSTEM FLOWCHART

### Coordinate Lookup Feature



This flowchart shows how coordinates values can be inputted by the user and they will return an action where the camera will move to that area on the Earth sphere and zoom in on it. It can take two formats of either decimal latitude and longitude or DMS (Decimal, Minutes, Seconds) latitude and longitude.

FIGURE 19: COORDINATE LOOKUP FLOWCHART

## Export Report

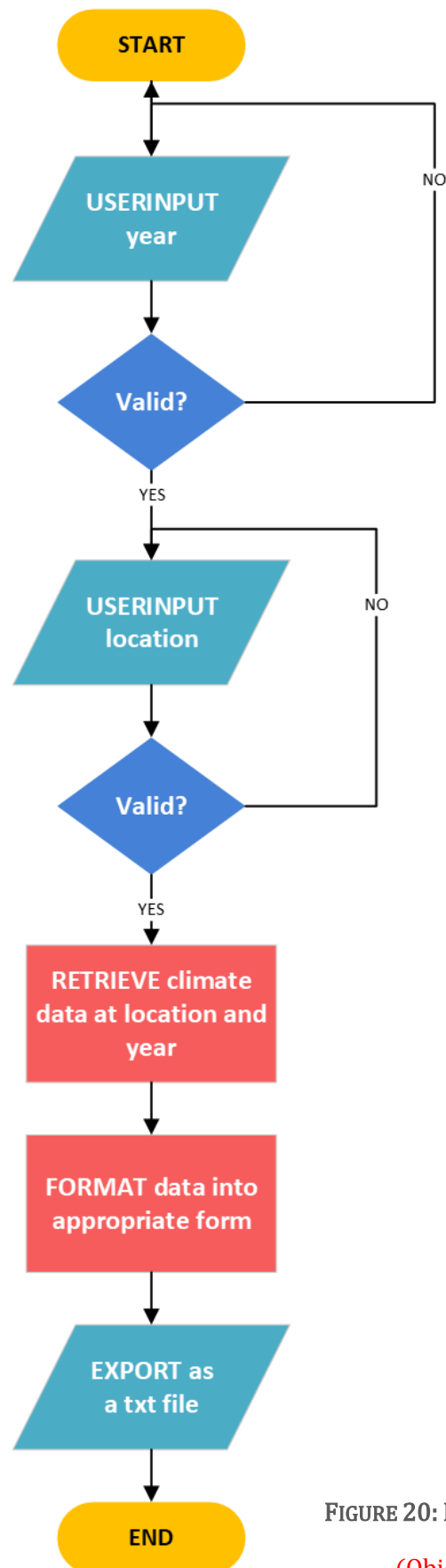


FIGURE 20: EXPORT REPORT FLOWCHART

(Objective 9.1, 9.2)



## Data Tables

### Employee Database (existing)

DATA ITEM	DATA TYPE	VALIDATIONS
First Name	String	Only Letters
Surname	String	Only Letters
Employee ID	string	4 digits Only Numbers
Email address	string	Must include @ symbol

### Climate Data

DATA ITEM	DATA TYPE	VALIDATIONS
Year	integer	Must be in range 1950 – 2023
Latitude	Float	In range from -90 to 90
Longitude	Float	In range from -180 to 180
Mean Annual temperature	Float	Units in Celsius degrees
Mean Annual Precipitation	Float	Units in millimetres (mm)
Annual Snow fall	Float	Units in mm

## **File Structures**

### **All Cleaned Climate Data (CSV File)**

This is a csv file which holds approximately 100,000 records of climate data. This data was obtained from the NOAA climate data archive which holds data for each coordinate in a separate file. From this I concatenated all of the files together in one and removed all unnecessary fields – around 50 fields – to be left with 10 fields.

Field name	Data Type	Detail
Station	String	It is the unique identifier for each station recording the climate data at its latitude and longitude e.g. AE000041196
Date	Integer	Date in years that the data was recorded
Latitude	Float	The latitude value in the NE (North, East) WGS64 System
Longitude	Float	The longitude value in the NE (North, East) WGS64 System
Name	String	The name of the station followed by initials of the country the station is in
TAVG	Float	Average annual temperature value in degrees Celsius
PRCP	Float	Average annual precipitation in millimetres
SNOW	Float	Average annual snowfall in millimetres

### Linear Regression Results (CSV Files)

In my Python code for obtaining linear regression values for climate data graphs, I export three CSV files – one for each parameter. Each of these files have the same fields and data types. Each record of these csv file would be holding values for each individual graph at unique latitude and longitude points.

Field name	Data Type	Detail
Slope	Float	This is simply the value for the gradient of the linear regression model
Intercept	Float	This is the Y-intercept of the linear regression model (the value of y when x is 0)
Latitude	Float	The latitude value in the NE (North, East) WGS64 System
Longitude	Float	The longitude value in the NE (North, East) WGS64 System
R Squared	Float	The r-squared value is calculated in the linear regression Python code and it represents how well the model fits the data ranging from 0 to 1

## Formulae Code

### Linear Regression Code

```
Python: ClimateVariable = (slope * date) + intercept
```

**Python:**

```
X_mean = sum(X)/len(x)
Y_mean = sum(Y)/len(Y)
Numerator = 0
Denominator = 0
For I in range(len(x)):
    Numerator += (X[i] - X_mean) * (Y[i] - Y_mean)
    Denominator += (X[i] - X_mean) ** 2
Slope = Numerator / Denominator
```

This python code follows the formula to calculate the gradient or slope of a set of scattered points which was stated in the Analysis section as:

**Formula:** 
$$m = \frac{\Sigma(x - \bar{x})(y - \bar{y})}{\Sigma(x - \bar{x})^2}$$

**Python Code:**

```
rss = 0
tss = 0

for i in range (len(x)):
    y_pred = slope * x[i] + intercept
    rss += (y[i] - y_pred)**2
    tss += (y[i] - mean_y)**2
r_squared = 1 - (rss/tss)
```

The python code to the left calculates the RSS (residual sum of squares) and the TSS value (total sum of squares) in order to determine the r-squared value for each graph. This r-squared value will then be used to get rid of unreliable or weak graphs, so data in the final program is reliable.

### Degrees to Radians Conversion Code

**C# Code:**

```
float phi = longitude * Mathf.PI / 180;  
float theta = latitude * Mathf.PI / 180;
```

**C# Code:**

```
float phi = longitude * Math.Deg2Rad;  
float theta = latitude * Math.Deg2Rad;
```

Both code snippets convert the latitude and longitude from degrees to radians but the first code block converts it according to the formula  $radians = degrees * \pi / 180$ . However, the second code snippet uses the Math library to convert them as it produces a value that is more precise with more decimal points.

### Spherical Coordinate Conversion Code

**C# Code:**

```
float Xcoord = radius * Mathf.Cos(theta) * Mathf.Cos(phi);  
float Ycoord = radius * Mathf.Cos(theta) * Mathf.Sin(phi);  
float Zcoord = radius * Mathf.Sin(theta);  
Vector3 cartCoords = new Vector3(Xcoord, Zcoord, Ycoord);
```

### Conversion of DMS coordinates to Decimal Degrees Code

**C# Code:**

```
float decimalValue = degrees + (minutes/60) + (seconds/3600);
```

### Inverse Distance Weighting Code

#### Calculate Weights Pseudocode:

```
p ← 2.0f;  
distance ← CalculateDistance(point1, point2);  
weight ← 1.0 / Power(distance, p);
```

#### Calculate Interpolated Value C# Code:

```
double weightSum = 0;  
double weightedVarSum = 0;  
foreach (point in points)  
{  
    double weight = CalcWeight(point, unknownPoint, p);  
    weightSum += weight;  
    weightedVarSum += weight * pointValue;  
}  
double interpolatedValue = weightedVarSum / weightSum;
```

### Haversine Distance Calculation Code

#### C# Code:

```
const int radius = 6378137;  
double latDiff = latitude2 * Math.Deg2Rad - latitude1 * Math.Deg2Rad;  
double lonDiff = longitude2 * Math.Deg2Rad - longitude1 * Math.Deg2Rad;  
  
double a = Math.Pow(Math.Sin(latDiff / 2.0), 2) + Math.Cos(latitude1) *  
Math.Cos(latitude2) * Math.Pow(Math.Sin(lonDiff / 2.0), 2);  
double c = 2.0 * Math.Asin(a);  
double distance = radius * c;
```

## Tested Algorithm

I obtained some climate data only for mean annual temperature and decided to test out a linear regression algorithm as it has been praised before in achieving a high level of precision in predicting climate change data. So, I wanted to identify, for myself, whether this would be a suitable regression algorithm or if I should turn to ARIMA which I had decided on previously.

Here is the code I had written:

```
import pandas as pd
import pymysql
#imports modules

#connects to the mysql server
connection = pymysql.connect(
    host='localhost',
    user='root',
    password='P@ssw0rd!',
    database='climatedata'
)

tempData = pd.read_sql('SELECT STATION, DATE, LATITUDE, LONGITUDE, TAVG FROM
my_table', con=connection)
#uses pandas module to read data from table from sql server
connection.close()

regression_results = {}

# Group the data by 'STATION' and perform linear regression for each group
for station, group in tempData.groupby('STATION'):
    x = group['DATE'].values.reshape(-1, 1)
# Reshape to 2D array for LinearRegression
    y = group['TAVG'].values
    model = LinearRegression()
    model.fit(x, y)
    regression_results[station] = {
        'slope': model.coef_[0],
        'intercept': model.intercept_,
        'latitude': group['LATITUDE'].values[0],
# Use the first value since it's the same for the group
        'longitude': group['LONGITUDE'].values[0],
    }

# plotting regression lines with latitude and longitude for each station
# iterates through each station, using results associated with them
for station, results in regression_results.items():
    plt.figure()
```

I had stored this data temporarily in a MySQL database due to the sheer size of the data. So I used this code to access it.

```
tempData['DATE'] = tempData['DATE'].astype(float)
results['slope'] = results['slope'].astype(float)

plt.scatter(tempData[tempData['STATION'] == station]['DATE'],
tempData[tempData['STATION'] == station]['TAVG'], label='Data')

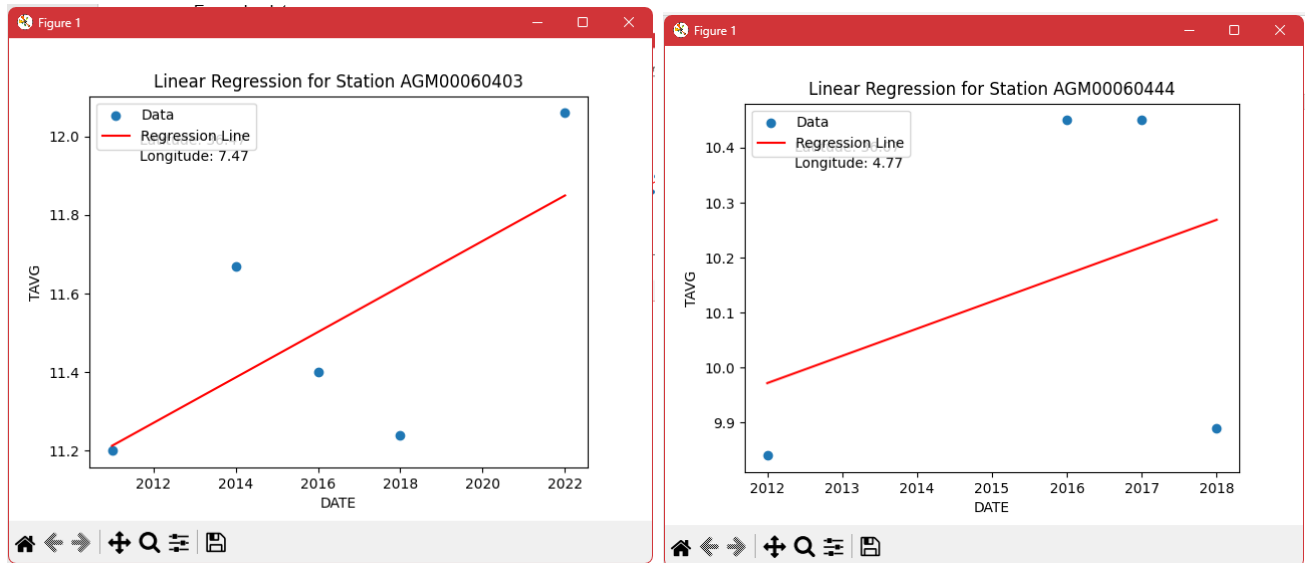
plt.plot(tempData[tempData['STATION'] == station]['DATE'],
results['slope'] * tempData[tempData['STATION'] == station]['DATE'] +
results['intercept'], color='red', label='Regression Line')

plt.title(f'Linear Regression for Station {station}')
plt.xlabel('DATE')
plt.ylabel('TAVG')
plt.text(0.1, 0.9, f'Latitude: {results["latitude"]:.2f}\nLongitude:
{results["longitude"]:.2f}', transform=plt.gca().transAxes, fontsize=10,
verticalalignment='top')

plt.legend()
plt.show()
```



## Weak Plots



The graphs shown above are very weak as they only have a few plot points and the line of best fit doesn't seem to even pass through a single one of these points. So, I needed to be able to distinguish between weak, inaccurate graphs and strong, precise graphs.

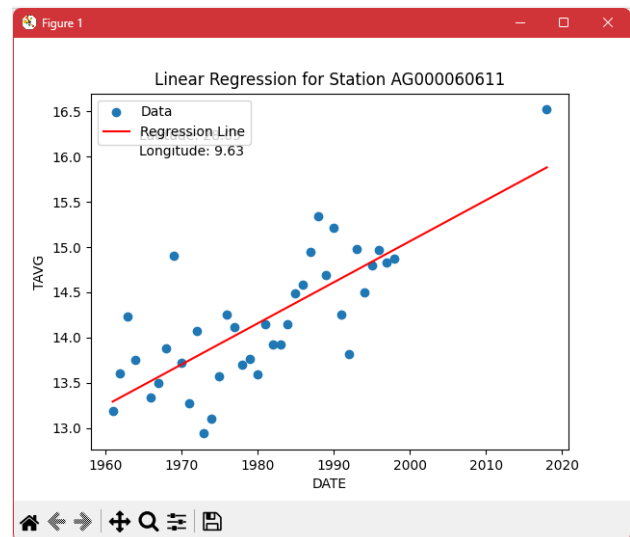
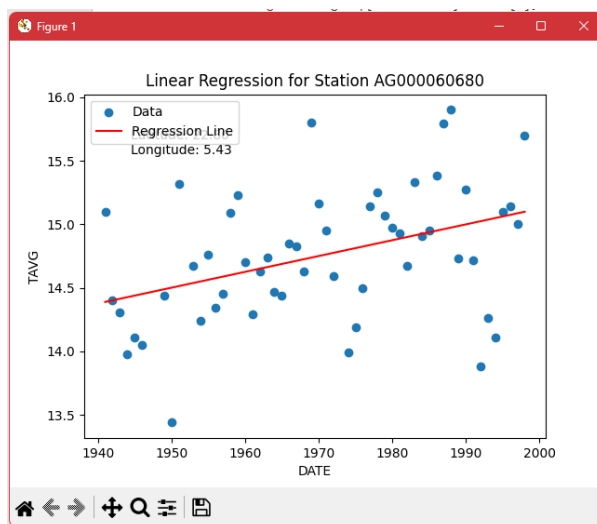
I have altered my code (highlighted) so I would calculate the r-squared value of each graph and only use that graph if the value is greater than a certain threshold.

```
threshold = 0.5
```

```
for station, group in tempData.groupby('STATION'):  
    x = group['DATE'].values.reshape(-1, 1)  
    y = group['TAVG'].values  
  
    # checks if there are at least 2 samples to prevent  
    "UndefinedMetricWarning: R^2 score is not well-defined with less than two  
    samples"  
    if len(x) < 2:  
        continue # Skip this station and move to the next  
  
    model = LinearRegression()  
    model.fit(x, y)  
    y_pred = model.predict(x) # Predicted values  
    r_squared = r2_score(y, y_pred) # Calculate R^2  
  
    if r_squared > threshold:  
        regression_results[station] = {  
            'slope': model.coef_[0],  
            'intercept': model.intercept_,  
            'latitude': group['LATITUDE'].values[0],
```

```
'longitude': group['LONGITUDE'].values[0],  
'r_squared': r_squared  
}
```

## Strong Linear Regression Plots



As you can see, these plots are heavily concentrated with data points which may suggest that these graphs have a moderate to strong correlation and thus must be reliable in predicting data. However, as you can see, especially in the left graph, the points are quite spread out and are very scattered. As for the second one, a correlation was found, despite the 20 year gap in data from 2000 to 2020. For these reasons, I have decided to go against the linear regression algorithm as it cannot capture seasonal trends in data, only a singular line. This, I believe, would not lead to accurate predictions, especially 25 years down the line where the data would be shown to exceed realistic values of temperature because the line is a constant increase.

## Section 3: Technical Solution

### Overview

In this section, I will showcase all of the pieces of code involved in creating this program, separated by appropriate headings for ease of understanding. I have also annotated my code by utilising comments within the code snippets ( they are typically highlighted in pink after either a # or //). Additionally, I have added separate text boxes to hold the objective that the code snippet relates to. Under a few code snippets I have given a short summary to sum up the purpose and functionality of the code.

Below I have listed the techniques used and their corresponding page numbers.

TECHNIQUES	PAGE NUMS.	YES/NO
OOP (Ignore page number)	-	YES
Classes – (class structure diagram)	Pg. 35	YES
Struct		NO
Enum	Pg. 68	YES
Inheritance	Pg. 68-69	YES
Polymorphism		NO
Procedures and Functions - used extensively to simplify the organisation and structure of your program (Ignore page number)	-	YES
Unity 3D or 2D simulation (Ignore page number)	-	YES
Arrays (multi-dimensional, or several single arrays used in an integrated way)	Pg. 53-54 Pg. 69-70 Pg. 82 Pg. 89	YES
Graphs/Tress/neural networks		NO
Queues		NO
Stacks		NO
Lists	Pg. 57 Pg. 69, Pg. 70	YES

	Pg. 79-80 Pg. 82 Pg. 89	
Sets	Pg. 62	YES
Vectors	Pg. 69 Pg. 71 Pg. 76	YES
Dictionary		NO
Records		NO
Sorts		NO
Read/write to file	Pg. 53 Pg. 70 Pg. 82 Pg. 84	YES
Exception handling	Pg. 63 Pg. 69-70	YES
Other(s): e.g. complex formulas (make a separate list if necessary)		
<ul style="list-style-type: none"> <li>• Conversion of latitude and longitude to 3D Cartesian Coordinates using azimuthal and polar angles (Pg. 71, 76)</li> <li>• Linear regression formulae (calculation of slope and intercept from a set of data) (Pg. 53-54)</li> <li>• R-squared calculation (Pg. 54)</li> <li>• Inverse Distance Weighting Interpolation formula (Pg. 82-83)</li> <li>• Haversine formula (Pg. 83-84)</li> </ul>		

## Linear Regression

### Python Code

```
#importing modules
import os
import matplotlib.pyplot as plt
import pandas as pd

#return the user profile to form file path
user_path = os.environ.get('USERPROFILE')
print(user_path)
csv_file = user_path + "\\Desktop\\cleanedData.csv"
print(csv_file)

# function to calculate the mean
def calculate_Mean (input):
    sum = 0
    # loops through each value in input and calculates sum of all values
    for i in range (len(input)):
        sum += input[i]
        i += 1
    # divides sum by number of values to obtain mean
    mean_value = sum/len(input)
    return mean_value

#subroutine to generate linear regression model for each climate variable
def linear_regression(climate_parameter):
    raw_variableData = pd.read_csv(csv_file, usecols=['STATION', 'DATE',
'LATITUDE', 'LONGITUDE', climate_parameter])

    #removes records where average temperature value is null
    if climate_parameter == 'TAVG':
        # drops record if there's no value for average temperature
        raw_variableData.dropna(subset= climate_parameter, inplace=True)
    else:
        raw_variableData.replace('', 0, inplace=True)
    print(raw_variableData)

    #holds results of linear regression
    regression_results = {}
    #states minimum level of accuracy for model
    r_squared_threshold = 0.5

    for station, group in (raw_variableData.groupby('STATION')):
        #declare x and y values of graph
        x = group['DATE'].values
```

OBJECTIVE 6.3: Future predictions must be feasible and formed from the extrapolation of reliable past data

```
y = group[climate_parameter].values

#skips station if number of data points is less than 2
if len(x) < 2:
    continue

#obtain means from subroutine
mean_x = calculate_Mean(x)
mean_y = calculate_Mean(y)

#use linear formulae to calculate slope and intercept
numerator_value = 0
denominator_value = 0

for i in range (len(x)):
    numerator_value += (x[i] - mean_x)*(y[i]-mean_y)
    denominator_value += (x[i]-mean_x)** 2
slope = numerator_value/denominator_value
intercept = mean_y - (slope * mean_x)
#print (f'slope = {slope}, intercept = {intercept}')
```

OBJECTIVE 6.3 – using the r squared value and eliminating weak plots improves reliability of data

```
#calculate r-squared values to eliminate insufficient models
rss = 0
tss = 0

for i in range (len(x)):
    y_pred = slope*x[i] + intercept
    rss += (y[i] - y_pred)**2
    tss += (y[i] - mean_y)**2
r_squared = 1 - (rss/tss)

if r_squared > r_squared_threshold:
    regression_results[station] = {
        'slope': slope,
        'intercept': intercept,
        'latitude': group['LATITUDE'].values[0], # Use the first
value since it's the same for the group
        'longitude': group['LONGITUDE'].values[0], # Use the first
value since it's the same for the group
        'r_squared': r_squared
    }

#plotting regression lines with latitude and longitude for each station
for station, results in regression_results.items():
    plt.figure()
    raw_variableData['DATE'] = raw_variableData['DATE'].astype(float)
    results['slope'] = results['slope'].astype(float)
```

```
plt.scatter(raw_variableData[raw_variableData['STATION'] ==
station]['DATE'], raw_variableData[raw_variableData['STATION'] ==
station][climate_parameter], label='Data')

plt.plot(raw_variableData[raw_variableData['STATION'] ==
station]['DATE'], results['slope'] *
raw_variableData[raw_variableData['STATION'] == station]['DATE'] +
results['intercept'], color='red', label='Regression Line')
plt.title(f'Linear Regression Model for Station {station}')

plt.xlabel('DATE')
plt.ylabel(climate_parameter)
plt.text(0.1, 0.9, f'Latitude: {results["latitude"]:.2f}\nLongitude:
{results["longitude"]:.2f}\nR²: {results["r_squared"]:.2f}',
transform=plt.gca().transAxes, fontsize=10, verticalalignment='top')
plt.legend()
plt.show()

return regression_results

def export_results(results, filename):
    df = pd.DataFrame(results).transpose()
    df.to_csv(filename, index_label='Station')
    print(f'Regression results saved to {filename}')

prcp_results = linear_regression('PRCP')
tavg_results = linear_regression('TAVG')
snow_results = linear_regression('SNOW')

export_results(prcp_results, 'prcp_regression_results.csv')
export_results(tavg_results, 'tavg_regression_results.csv')
export_results(snow_results, 'snow_regression_results.csv')
```

## **Code summary**

This Python script is used to carry out linear regression on climate data from different stations. This program reads the csv file consisting of all concatenated climate date, calculates the average of the data, and makes a model that shows how climate changes over time. The script deals with missing values, groups data by station, and calculates the slope, intercept, and R-squared value for each group. It removes models that have a lower R-squared value than a certain threshold of 0.5. It also displays the data points and regression lines for each station and exports the regression results to separate CSV files to be used by other scripts in Unity. The program linearly regresses three parameters to do with the climate: annual rainfall (in mm),

average annual temperature ( in degrees Celsius), annual snow fall (in mm). In short, this script is used to analyse climate data, perform linear regression, visualise the results, and export the data to be imported into the Unity Scene.

## *Skybox Rotation*

### C# Code

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class movingskybox : MonoBehaviour
{
    public float rotate_Speed = 1.3f;

    void Update()
    {
        RenderSettings.skybox.SetFloat("_Rotation", Time.time * rotate_Speed);
    }
}
```

### Summary

I have assigned this code to the main camera in the scene and essentially what the code does is that it accesses the shader render settings of the skybox I created (using images of space with stars) and it modifies the rotation of the skybox by 1.3 every second as the rotation speed is multiplied by the time (distance = speed x time). This results in a continuously changing rotation angle which makes the effect of a rotating skybox.

## *Exit Program*

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;

public class TitleScreen : MonoBehaviour
{
    public void ExitProgram()
    {
        Debug.Log("Exited program.");
        Application.Quit();
    }
}
```



}

## Switch Between Scenes

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;

public class SwitchScenes : MonoBehaviour
{
    public string nextScene;
    public void SwitchScene()
    {
        // Track the current scene before loading the new one
        SceneTracker.TrackScene(SceneManager.GetActiveScene().name);

        // Load the new scene
        SceneManager.LoadScene(nextScene, LoadSceneMode.Single);
    }
}

using UnityEngine;
using UnityEngine.SceneManagement;
using System.Collections.Generic;

public class SceneTracker : MonoBehaviour
{
    // list to stores the history of scenes accessed
    static List<string> sceneHistory = new List<string>();

    public static void TrackScene(string sceneName)
    {
        sceneHistory.Add(sceneName);
    }

    // called to load the previous scene
    public static void LoadPreviousScene()
    {
        if (sceneHistory.Count >= 2) // needs at least 2 scenes in the history
        to go back
        {
            // Get the scene at the second last position in the history list
            string sceneToLoad = sceneHistory[sceneHistory.Count - 2];
            // Remove the current scene from the history
            sceneHistory.RemoveAt(sceneHistory.Count - 1);
            // Load the previous scene
            SceneManager.LoadScene(sceneToLoad, LoadSceneMode.Single);
        }
    }
}
```

```
}  
}
```

## **Summary**

This code allows the program to switch between different scenes set up in Unity. For example, when the login details are authorised and the login button is pressed, this script can be triggered to switch to the main scene. The SceneTracker script is used to store history of scenes accessed in a list so that if there was a 'Go Back' button it would go to the previous scene. For example, the 'settings' scene can be accessed from the opening scene and main scene, so we cannot simply enter the name of the scene to go back to with the 'Go back' button. So, the history of scenes is used to ensure the user is returned to the correct scene.

## Login & Registration System

### SQL for Database

```
CREATE TABLE `users_database`.`user_logininfo` (`EmployeeID` INT NOT NULL ,  
`Username` VARCHAR(50) NOT NULL , `Password` VARCHAR(50) NOT NULL , PRIMARY  
KEY (`EmployeeID`)) ENGINE = InnoDB;
```

Creates a table within the users database for the login information. This includes the employee ID (which is verified against the employee database), username and password. The employee ID is set as the primary key as it is a unique identifier for each account.

```
CREATE TABLE `users_database`.`employees_info` (`EmployeeID` INT NOT NULL ,  
`FirstName` VARCHAR(20) NOT NULL , `LastName` VARCHAR(40) NOT NULL ,  
PRIMARY KEY (`EmployeeID`)) ENGINE = InnoDB;
```

Creates a table to hold employee information that is necessary for registration. This will hold records of all of the employees in the company or any other people who need to gain access to this software.

```
ALTER TABLE user_logininfo  
ADD CONSTRAINT fk_EmployeeID  
FOREIGN KEY (EmployeeID)  
REFERENCES employees_info (EmployeeID);
```

OBJECTIVES 1.3 & 2.1: Creating an employee database to verify registration details against; Creating a users database to verify login details against

Alters the tables in the database to make the employee ID a foreign key across the login information table and the employee information table.

### Login Front-End

```
using System.Collections;  
using System.Collections.Generic;  
using UnityEngine;  
using UnityEngine.UI;  
using TMPro;  
  
public class LoginSystem : MonoBehaviour  
{  
    // declares the input fields to retrieve the user inputs  
    public TMP_InputField UsernameInput;  
    public TMP_InputField PasswordInput;  
    public Button LoginButton;  
  
    private SwitchScenes switchScenes;
```

```
void Start ()
{
    switchScenes = LoginButton.GetComponent<SwitchScenes>();
    // only runs code within lambda function once the button is pressed
    LoginButton.onClick.AddListener(() =>
    {
        // executes subroutine LoginFunction within WebRequest script with
        the parameters of the inputted username and password
        StartCoroutine(Main.Instance.WebRequest.LoginFunction(UsernameInput.
t.text, PasswordInput.text, success =>
        {
            if (success)
            {
                // if successful the login button will proceed to the main
screen

                switchScenes.nextScene = "main v3";
                switchScenes.SwitchScene();
            }
        }));
    });
}
```

OBJECTIVE 2.2: Immediately opens the main scene once the login is successful

## Register Front-End

```
using System;
using System.Collections;
using System.Collections.Generic;
using System.Text.RegularExpressions;
using UnityEngine;
using UnityEngine.UI;
using UnityEngine.SceneManagement;
using TMPro;

public class RegistrationSystem : MonoBehaviour
{
    // declare the input fields to retrieve user inputs
    public TMP_InputField employeeID;
    public TMP_InputField firstName;
    public TMP_InputField lastName;
    public TMP_InputField password;
    public TMP_InputField passwordRepeat;
    public Button RegisterButton;
    public TextMeshProUGUI RegisterMsgs;

    public GameObject Username_PopUp;
```

```
public TextMeshProUGUI UsernameMsg;

void Start ()
{
    // the code within the lambda function will be execute when the button
    is clicked
    RegisterButton.onClick.AddListener(() =>
    {
        // only executes code if the password is considered valid
        if (PasswordValid(password.text, passwordRepeat.text))
        {
            // converts user input (string) to an integer to verify
            against database
            int employeeID_int = Convert.ToInt32(employeeID.text);
            // create unique username using first name, surname and
            employee ID
            string username = (firstName.text.ToLower()).Substring(0, 1) +
            lastName.text.ToLower() + employeeID.text;
            // output the created username as text on the screen
            RegisterMsgs.text = username;
            // coroutine calls the RegisterUser function from the
            WebRequest class of the Main instance
            StartCoroutine(Main.Instance.WebRequest.RegisterUser(employeeI
            D_int, firstName.text, lastName.text, username, password.text, success =>
            {
                if (success)
                {
                    // pop-up screen becomes visible and displays username
                    and message
                    Username_PopUp.SetActive(true);
                    UsernameMsg.text = ("Your generated username is:
                    {username} \n Please use this and your password to login...");
                }
            }));
        }
    });
}

public bool PasswordValid(string passwordInput, string repeatedPass)
{
    bool valid = false;
    if (passwordInput == repeatedPass)
    {
        //regular expression only validates strings with length of 8-25
        upper and lowercase, a digit and a special character
        Regex passwordStrong = new Regex("(?=(?=.*?[A-Z])(?=.*?[a-
        z])(?=.*?[0-9])(?=.*?[#?!@$%^&*~]).{8,25}$");
    }
}
```

OBJECTIVE 1.5: Generates a unique and memorable username from the first name, surname and ID

```
        // IsMatch method checks if password follows regular expression
rules
    bool strong = passwordStrong.IsMatch(passwordInput);
    if (!strong)
    {
        RegisterMsgs.text = "Password must meet minimum requirements";
        // upper and lowercase, number & special character. \n Must
also have length from 8-25 characters.
    }
    else
    {
        valid = true;
    }
}
else
{
    RegisterMsgs.text = ("Passwords do not match");
}
return valid;
}
}
```

OBJECTIVE 1.6: Password must have at least 8 characters, a special character, both upper and lower case and at least one digit

## Web Server Connection

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Main : MonoBehaviour
{
    // create an instance with the class Main
    public static Main Instance;
    public WebRequest WebRequest;

    void Start()
    {
        Instance = this;
        WebRequest = GetComponent<WebRequest>();
    }
}

using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
```

```
using UnityEngine.Networking; // namespace for classes that are required for
network communication in Unity
using UnityEngine.UI;
using TMPro;

public class WebRequest : MonoBehaviour
{
    // declare text fields so messages can be outputted
    public TextMeshProUGUI LoginMsgs;
    public TextMeshProUGUI RegisterMsgs;

    // declares a coroutine named 'LoginFunction' that takes two string
    parameters: 'username' and 'password'
    public IEnumerator LoginFunction(string username, string password,
    Action<bool> callback)
    {
        // create form object for sending the username and password to the
server
        WWWForm form = new WWWForm();
        form.AddField("loginUser", username);
        form.AddField("loginPassword", password);

        // sends the form data created to the mysql server via HTTP POST
        UnityWebRequest www =
        UnityWebRequest.Post("http://localhost/back_to_the_climate/Login.php", form);
        // sends the web request and waits
        yield return www.SendWebRequest();

        if (www.result != UnityWebRequest.Result.Success)
        {
            // logs error message to the console log
            Debug.Log(www.error);
            LoginMsgs.text = "Error occured...Try again";
            callback(false);
        }
        else if (www.downloadHandler.text != "Login success")
        {
            LoginMsgs.text = (www.downloadHandler.text);
            callback(false);
        }
        else // if web request was successful
        {
            // sets the LoginMsgs text to the response message from the server
            LoginMsgs.text = (www.downloadHandler.text);
            callback(true);
        }
    }
}
```

```
public IEnumerator RegisterUser(int employeeID, string firstName, string
lastName, string username, string password, Action<bool> callback)
{
    // creates a form object for sending the employeeID, first name, last
name, username and password to the server
    WWWForm form = new WWWForm();
    form.AddField("loginID", employeeID);
    form.AddField("firstName", firstName.ToLower());
    form.AddField("lastName", lastName.ToLower());
    form.AddField("loginUser", username);
    form.AddField("loginPassword", password);

    // sends registration details to the server
    UnityWebRequest www =
UnityWebRequest.Post("http://localhost/back_to_the_climate/RegisterUser.php",
form);

    yield return www.SendWebRequest();

    if (www.result != UnityWebRequest.Result.Success)
    {
        // logs error message to the console
        Debug.Log(www.error);
        RegisterMsgs.text = "Error occured...Try again";
        // sets Boolean 'success' to false for coroutine in
RegistrationSystem.cs
        callback(false);
    }
    else if (www.downloadHandler.text != "New User created successfully!")
    {
        RegisterMsgs.text = www.downloadHandler.text;
        callback(false);
    }
    else
    {
        // sets out message text of RegisterMsgs to the response of server
        RegisterMsgs.text = www.downloadHandler.text;
        // sets Boolean 'success' to false for coroutine in
RegistrationSystem.cs
        callback(true);
    }
}
}
```

### Login PHP code (link to web server)

```
<?php
// declare mysqli server details
$servername = "localhost";
$username = "root";
```



```
$password = "";
$databaseName = "users_database";

// declare variables for user inputs
$loginUser = $_POST["loginUser"];
$loginPassword = $_POST["loginPassword"];

// connects to the mysqli server
$connection = new mysqli($servername, $username, $password, $databaseName);

// verifies the connection to the server
if ($connection->connect_error)
{
    // if an error crops up, the connection is terminated and message outputted
    die("Connection failed: " . $connection->connect_error);
}
$response = "Connected successfully";

// retrieves the corresponding password for unique username from server
$sql = "SELECT Password FROM user_logininfo WHERE Username = '". $loginUser .
'";
$result = $connection->query($sql);

// fetches each row of data through iteration
if ($result->num_rows > 0)
{
    while($row = $result->fetch_assoc())
    {
        if ($row["Password"] == $loginPassword)
        {
            $response = "Login success";
        }
        else
        {
            $response = "Username or password is incorrect";
        }
    }
}
else
{
    $response = "Username does not exist";
}

$connection->close();
// returns response messages
echo $response;
?>
```

### Register php code (link to web server)

```
<?php
// declares details for the mysql server
$servername = "localhost";
$username = "root";
$password = "";
$databaseName = "climateUsers";

// user inputs for registration details
$loginID = $_POST["loginID"];
$firstName = $_POST["firstName"];
$lastName = $_POST["lastName"];
$loginUser = $_POST["loginUser"];
$loginPassword = $_POST["loginPassword"];

// creates a connection to the mysqli server
$connection = new mysqli($servername, $username, $password, $databaseName);

// verifies the connection to server
if ($connection->connect_error)
{
    // terminates the connection if there is an error
    die("Connection failed: " . $connection->connect_error);
}
$response = "Connected successfully";

// retrieves password for unique username from server
$sql = "SELECT Username FROM user_logininfo WHERE Username = '". $loginUser .
''";
$result = $connection->query($sql);

//checks if credentials match employee database
$sql2 = "SELECT EmployeeID from employees_info WHERE EmployeeID = '" .
$loginID . "' AND FirstName = '" . $firstName . "' AND LastName = '" .
$lastName . "'";
$employeeValid = $connection->query($sql2);

// outputs each row of data through iteration
if ($result->num_rows > 0)
{
    // ensures that username isn't already in the database or account is already
    created with employee ID
    $response = "Username already exists";
}
// if number of rows returned with entered ID is zero, employee ID isn't
valid/part of employee database
elseif ($employeeValid->num_rows == 0)
```

OBJECTIVE 1.2: Takes in the required user inputs for a registration form: employee ID, first name, surname and password

OBJECTIVE 1.4: flags error if account is already created for that employee ID

```
{
    $response = "Invalid Employee ID";
}
else // if employee credentials inputted match the database
{
    $response = "Creating User...";
    // adds record to table with login information so a new account is created
    and can be verified for future logins
    $sql3 = "INSERT INTO user_logininfo (EmployeeID, Username, Password) VALUES
('" . $loginID . "', '" . $loginUser . "', '" . $loginPassword . "')";

    if ($connection->query($sql3))
    {
        $response = "New User created successfully!";
    }
    else
    {
        $response = "Error: " . $sql3 . "<br>" . $connection->error;
    }
}

$connection->close();
echo $response;
?>
```

## Earth Plots & Climate Variables

```
// define namespaces required
using System;
using System.IO;
using System.Linq;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
using TMPro;

public class LLConversion : MonoBehaviour
{
    // declare all ui elements that need to be accessed by this code
    public SphereCollider MyCollider;
    public RenderTexture HeatMap;
    public GameObject Earth;
    public GameObject DotPrefab;
    public Texture2D Gradient;
    public Slider Timeline;
    public TextMeshProUGUI YearOutput;
    public TextMeshProUGUI LegendMin;
    public TextMeshProUGUI LegendMax;
    public ToggleGroup Parameters;

    // create a class which stores values for each data point
    public class ClimateDataPoint
    {
        public GameObject Dot { get; set; }
        public float Latitude { get; set; }
        public float Longitude { get; set; }
        public float Slope { get; set; }
        public float Intercept { get; set; }
    }

    // enum is used to represent a group of constants, in this case: climate
    // variables
    public enum DataType
    {
        Temperature,
        Precipitation,
        Snowfall
    }

    // create a class for storing relevant data for each climate variable
    public class allParameterData
    {
```

OBJECTIVE 5.1: Temperature, precipitation and snowfall data must available to be potted on the Earth

```
    public DataType Parameters { get; set; }
    public List<ClimateDataPoint> VarDataPoints { get; set; }
    public string[][] FileContents { get; set; }
    public int MinValue { get; set; }
    public int MaxValue { get; set; }
}

// create list to hold three instances (temperature, precipitation and
snowfall) of the class allParameterData
private List<allParameterData> allData = new List<allParameterData>();

void Start()
{
    // sets rotation to default (zero)
    Earth.transform.rotation = Quaternion.identity;
    // gets collider of earth to obtain radius
    MyCollider = GetComponent<SphereCollider>();
    // loops through each type in the enum DataType
    foreach (DataType parameter in Enum.GetValues(typeof(DataType)))
    {
        // create an instance of the class and assign each value from
        // respective functions
        allParameterData data = new allParameterData();
        data.Parameters = parameter;
        // the file contents of the csv file (holding regression results)
        // for each parameter is stored
        data.FileContents = LoadData(parameter);
        Vector2 minMax = GetMinMax(parameter);
        data.MinValue = (int)minMax.x;
        data.MaxValue = (int)minMax.y;
        data.VarDataPoints = CreateDots(data);

        // add instance to a list holding all data of all three parameters
        allData.Add(data);
    }
}

string[][] LoadData(DataType parameter)
{
    string userProfilePath =
Environment.GetFolderPath(Environment.SpecialFolder.UserProfile);
    string filePath = "";
    // returns corresponding file path for each parameter
    switch (parameter)
    {
        case DataType.Temperature:
            filePath = Path.Combine(userProfilePath, @"unity
programs\final_unity\Assets\Resources\tavg_regression_results.csv");
    }
}
```

```
        break;
    case DataType.Precipitation:
        filePath = Path.Combine(userProfilePath, @"unity
programs\final_unity\Assets\Resources\prcp_regression_results.csv");
        break;
    case DataType.Snowfall:
        filePath = Path.Combine(userProfilePath, @"unity
programs\final_unity\Assets\Resources\snow_regression_results.csv");
        break;
    }

    using (var read = new StreamReader(filePath))
    {
        read.ReadLine();
        // holds each value separated by a comma in the array
        var lines = new List<string[]>();
        while (!read.EndOfStream)
        {
            var line = read.ReadLine();
            var values = line.Split(",");
            lines.Add(values);
        }
        return lines.ToArray();
    }
}

public List<ClimateDataPoint> CreateDots(allParameterData data)
{
    List<ClimateDataPoint> dataPoints = new List<ClimateDataPoint>();
    foreach (string[] record in data.FileContents)
    {
        // obtains necessary values from each record of the csv file
        float latitude = float.Parse(record[3]);
        float longitude = float.Parse(record[4]);
        float slope = float.Parse(record[1]);
        float intercept = float.Parse(record[2]);

        float pastVal = 1970 * slope + intercept;
        float futureVal = 2050 * slope + intercept;

        // doesn't instantiate points where the regressed value exceeds
the limits
        if (pastVal < data.MinValue || pastVal > data.MaxValue)
        {
            continue;
        }
        if (futureVal < data.MinValue || futureVal > data.MaxValue)
        {
            continue;
        }
    }
}
```

```
        continue;
    }

    // convert latitude and longitude to 3D spherical coordinates
    Vector3 position = ConvertTo_CartesianCoords(latitude, longitude);
    // creates new dot gameobject based on the prefab at (0, 0, 0)
coordinates
    GameObject dot = Instantiate(DotPrefab, Vector3.zero,
Quaternion.identity);
    // moves dot to 3D spherical coordinates calculated from latitude
and longitude
    dot.transform.position = position;
    // sets the dot as a child of the Earth sphere gameobject so dots
stay fixed on earth regardless of transformation
    dot.transform.SetParent(Earth.transform, true);

    dataPoints.Add(new ClimateDataPoint {Dot = dot, Latitude =
latitude, Longitude = longitude, Slope = slope, Intercept = intercept});
    }
    return dataPoints;
}

public Vector3 ConvertTo_CartesianCoords(float latitude, float longitude)
{
    // set radius of sphere the dots are plotted on
    float radius = 0.51f;
    // convert latitude and longitude degrees to radians
    float phi = longitude * Mathf.PI / 180;
    float theta = latitude * Mathf.PI / 180;

    // use formulae with polar angle (theta) and azimuthal angle (phi)
    float Xcoord = radius * Mathf.Cos(theta) * Mathf.Cos(phi);
    float Ycoord = radius * Mathf.Cos(theta) * Mathf.Sin(phi);
    float Zcoord = radius * Mathf.Sin(theta);

    // create 3d coordinates
    Vector3 coordinate = new Vector3 (Xcoord, Zcoord, Ycoord);
    // preserve direction of vector but change magnitude to the radius so
all points lie at the same altitude on the earth
    coordinate = coordinate.normalized * radius;
    return coordinate;
}

Vector2 GetMinMax(DataType parameter)
{
    Vector2 minMax = new Vector2();
    // return minimum and maximum values corresponding to the parameter
    switch (parameter)
```

```
{
    case DataType.Temperature:
        minMax.x = 0;
        minMax.y = 30;
        break;
    case DataType.Precipitation:
        minMax.x = 0;
        minMax.y = 20000;
        break;
    case DataType.Snowfall:
        minMax.x = 0;
        minMax.y = 10000;
        break;
}
return minMax;
}

void Update ()
{
    //gets selected toggle within toggle group
    Toggle selectedToggle = Parameters.ActiveToggles().FirstOrDefault();

    // if no toggles are selected, set all dots to be invisible
    if (selectedToggle == null)
    {
        foreach (var data in allData)
        {
            foreach (var point in data.VarDataPoints)
            {
                point.Dot.SetActive(false);
            }
        }
        return;
    }

    DataType selectedDataType;
    string units;
    switch (selectedToggle.gameObject.name)
    {
        case "TempToggle":
            selectedDataType = DataType.Temperature;
            units = " °C";
            break;
        case "PrcpToggle":
            selectedDataType = DataType.Precipitation;
            units = " mm";
            break;
        case "SnowToggle":
```



```
        selectedDataType = DataType.Snowfall;
        units = " mm";
        break;
    default:
        throw new Exception("Unknown Toggle Selected");
        // catches exception with inconsistencies with naming or other
errors
    }

    foreach (allParameterData parameter in allData)
    {
        if (selectedDataType == parameter.Parameters)
        {
            // displays the maximum and minimum values on the legend
            LegendMin.text = (parameter.MinValue).ToString() + units;
            LegendMax.text = (parameter.MaxValue).ToString() + units;
        }
    }

    // gets the year from the timeline input
    int year = (int)Timeline.value;
    // width and height of render texture
    int width = HeatMap.width;
    int height = HeatMap.height;
    Texture2D HeatMapTex = new Texture2D(width, height);

    foreach (var data in allData)
    {
        foreach (var point in data.VarDataPoints)
        {
            float value = (point.Slope * year) + point.Intercept;
            int normalisedValue = (int)((value/data.MaxValue) *
Gradient.width);
            // gets colour from red-green-blue gradient
            Color pixColour = Gradient.GetPixel(normalisedValue, 5);
            GameObject currentDot = point.Dot;
            currentDot.GetComponent<Renderer>().material.color =
pixColour;

            if (data.Parameters == selectedDataType)
            {
                // only makes data dots of the paramter selected visible
                currentDot.SetActive(true);
            }
            else
            {
                // makes data dots of other paramters not visible
                currentDot.SetActive(false);
            }
        }
    }
}
```

OBJECTIVE 5.3: The values on the legend must change depending on the paramtere chosen

OBJECTIVE 5.2: Colour of data point must be related to the value and where it lies on the scale between the minimum value and maximum value

```
    }

    //converts latitude and longitude to 2d xy coordinates
    int x = (int)((point.Longitude + 180) / 360) * width;
    int y = (int)((point.Latitude + 90) / 180) * height;
    Vector2 pixelCoordinates = new Vector2 (x, y);

    // plots points on texture
    HeatMapTex.SetPixel((int)(pixelCoordinates.x),
(int)(pixelCoordinates.y), pixColour);
    }
}
HeatMapTex.Apply();
Graphics.Blit(HeatMapTex, HeatMap);
// updates the text to show which year is selected on the timeline
YearOutput.text = $"{year}";
}
}
```

## Summary of Code

The climate data is represented by instances of the ClimateDataPoint class, which includes attributes for the data points: dot game object, latitude, longitude, slope, and intercept at the point. The data types (temperature, precipitation, and snowfall) are represented by an Enum named DataType.

The allParameterData class is used to store all of the compiled data formed from the data type , file contents and lists of the ClimateDataPoint, and the minimum and maximum values for each parameter. In the Start method, the script initializes the Earth model and loads the climate data. The LoadData method reads the data from csv files, and the CreateDots method instantiates the dots on the Earth model. The ConvertTo\_CartesianCoords method converts the geographical coordinates (latitude and longitude) to 3D Spherical Cartesian coordinates for placement on the 3D model. The Update method is called once per frame and updates the visualization based on the selected data type and the current year on the timeline. It also updates the texture and the year output text. Additionally, the minimum and maximum values on the legend are altered based on the climate variable that is toggled along with its units so the user is a way of which units are being used for that variable and there are no discrepancies.

## Coordinate Lookup System

### Decimal Latitude & Longitude System

```
using UnityEngine;
using UnityEngine.UI;
using System.Collections;
using System.Collections.Generic;
using System.Globalization;
using TMPro;
using System;

public class coordLookup : MonoBehaviour
{
    [SerializeField] private Transform earthTransform;
    [SerializeField] private TMP_InputField LatInput;
    [SerializeField] private TMP_InputField LongInput;
    [SerializeField] private float zoomDistance = 1.0f; // The distance from
the point to zoom in on
    [SerializeField] private TextMeshProUGUI ErrorMessage;

    public float searchLatitude;
    public float searchLongitude;

    public void ZoomToLatLong()
    {
        // checks if null to output correct error message
        if (LatInput.text != null && LongInput.text != null )
        {
            try
            {
                // retrieves latitude and longitude from input fields
                searchLatitude = float.Parse(LatInput.text);
                searchLongitude = float.Parse(LongInput.text);
                if (searchLatitude >= -90 && searchLatitude <= 90 &&
searchLongitude >= -180 && searchLongitude <= 180)
                {
                    ConvertAndTransform (searchLatitude, searchLongitude);
                }
                else
                {
                    throw new Exception ("Latitude and/or longitude is not
valid");
                    ErrorMessage.text = "Latitude and/or longitude are not
within range";
                }
            }
        }
    }
}
```

OBJECTIVE 7.1: a set of  
coordinates must be entered for  
the coordinate lookup feature

```
        catch (FormatException)
        {
            // outputs if the input isn't a number
            ErrorMessage.text = "Incorrect format";
            throw new Exception("Incorrect Decimal format");
        }
    }
    else
    {
        // outputs error message if field is blank
        ErrorMessage.text = "You cannot leave any field blank";
        throw new Exception("Field blank");
    }
}

public void ConvertAndTransform (float latitude, float longitude)
{
    // converts latitude and longitude to radians
    float theta = latitude * Mathf.Deg2Rad;
    float phi = longitude * Mathf.Deg2Rad;

    // calculate the position camera needs to zoom in on with formulae
    float x = Mathf.Cos(theta) * Mathf.Cos(phi);
    float y = Mathf.Cos(theta) * Mathf.Sin(phi);
    float z = Mathf.Sin(theta);

    // create 3D coordinates to move camera to
    Vector3 zoomPosition = new Vector3(x, z, y) * zoomDistance;
    // sets the earths rotation to zero
    earthTransform.rotation = Quaternion.identity;
    // moves the camera to the zoom position
    transform.position = earthTransform.position + zoomPosition;
    // rotates the camera to look at the Earth's center
    transform.LookAt(earthTransform.position);
}
}
```

OBJECTIVE 7.2: The position and rotation of the camera is altered to move to the location on the earth that was inputted

### Code Summary

This code takes in the values of the latitude and longitude in decimal degrees, checks that it is in the right format and then transforms the position and rotation of the scene camera so that it moves to that area and focuses on it.

## Decimal, Minutes, Seconds System

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
using TMPro;
using System;
```

```
public class DMStoLL : MonoBehaviour
{
```

```
    public GameObject Camera;
    public TMP_InputField Lat_Degrees;
    public TMP_InputField Lat_Minutes;
    public TMP_InputField Lat_Seconds;

    public TMP_InputField Lon_Degrees;
    public TMP_InputField Lon_Minutes;
    public TMP_InputField Lon_Seconds;
```

```
    [SerializeField] private TextMeshProUGUI ErrorMessage;
```

```
    private coordLookup coordLookup;
```

```
    public void Convert_DMStoLL ()
    {
        coordLookup = Camera.GetComponent<coordLookup>();

        if (Lat_Degrees!=null && Lat_Minutes!=null && Lat_Degrees!= null &&
Lon_Degrees!=null && Lon_Minutes!= null && Lon_Seconds!= null)
        {
            // will try block of code and catches any format exception to
            output to screen
            try
            {
                // parses value from input field to integer and float data
                types

                int degreesLat = Int32.Parse(Lat_Degrees.text);
                int minsLat = Int32.Parse(Lat_Minutes.text);
                float secsLat = float.Parse (Lat_Seconds.text);

                int degreesLon = Int32.Parse(Lon_Degrees.text);
                int minsLon = Int32.Parse(Lon_Minutes.text);
                float secsLon = float.Parse (Lon_Seconds.text);

                // calls the subroutine to calculate the decimal value for
                latitude and longtiude
                float latitude = DMS2Dec (degreesLat, minsLat, secsLat);
```

OBJECTIVE 7.1 The user must enter a set of coordinates (DMS coordinates) to find a location on the earth

```
        float longitude = DMS2Dec (degreesLon, minsLon, secsLon);

        // calls the subroutine from coordLookup to transform camera
        coordLookup.ConvertAndTransform(latitude, longitude);
    }
    catch (FormatException)
    {
        // if format exception is thrown because values cannot be
        converted to Int32 or Float, error message is shown
        ErrorMessage.text = "Degrees and Minutes must be integers";
        throw new Exception("DMS coordinates are in the wrong
format");
    }
}
else
{
    ErrorMessage.text = "A field is blank";
}
}

private float DMS2Dec (int degrees, int minutes, float seconds)
{
    // uses DMS to decimal formula to calculate the latitude and longitude
    decimal values
    float decimalValue = degrees + (minutes/60) + (seconds/3600);
    return decimalValue;
}
}
```

## Export Report

### C# Code

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using TMPro;
using System;
using UnityEngine.UI;
using System.IO;

public class reportV2 : MonoBehaviour
{
    [SerializeField] private TMP_InputField YearInput;
    [SerializeField] private TMP_InputField DecLat;
    [SerializeField] private TMP_InputField DecLon;
    [SerializeField] private TMP_InputField DegLat;
    [SerializeField] private TMP_InputField MinLat;
    [SerializeField] private TMP_InputField SecLat;
    [SerializeField] private TMP_InputField DegLon;
    [SerializeField] private TMP_InputField MinLon;
    [SerializeField] private TMP_InputField SecLon;
    [SerializeField] private TextMeshProUGUI OutputMsgDec;
    [SerializeField] private TextMeshProUGUI OutputMsgDms;
    [SerializeField] private GameObject DecSystem;

    // create class to hold attributes of each plot point
    class Plot
    {
        public double Latitude { get; set; }
        public double Longitude { get; set; }
        public double VariableData { get; set; }

        public Plot (double latitude, double longitude, double varData)
        {
            Latitude = latitude;
            Longitude = longitude;
            VariableData = varData;
        }
    }

    // create a new list with the type Plot to hold the plot points for each
    parameter
    List<Plot> temperatureData = new List<Plot>();
    List<Plot> precipitationData = new List<Plot>();
    List<Plot> snowfallData = new List<Plot>();
```

```
public int year;
public double inputLat;
public double inputLon;

public void Main()
{
    // checks if the decimal system was selected or the DMS system
    if (DecSystem.activeInHierarchy)
    {
        try
        {
            inputLat = double.Parse(DecLat.text);
            inputLon = double.Parse(DecLon.text);
        }
        catch (FormatException)
        {
            OutputMsgDec.text = "Fields are in incorrect format or blank";
            throw new Exception ("Incorrect format or blank");
        }
        if (inputLat < -90 | inputLat > 90 || inputLon < -180 || inputLon
> 180)
        {
            OutputMsgDec.text = "Latitude and/or longitude out of range";
            throw new Exception ("Latitude and longitude out of range");
        }
    }
    else
    {
        try
        {
            // converts the input values from DMS (degrees, minutes,
seconds) to decimal
            inputLat = DMS2Dec(int.Parse(DegLat.text),
int.Parse(MinLat.text), float.Parse(SecLat.text));
            inputLon = DMS2Dec(int.Parse(DegLon.text),
int.Parse(MinLon.text), float.Parse(SecLon.text));
        }
        catch (FormatException)
        {
            OutputMsgDms.text = "Fields are in incorrect format or blank";
            throw new Exception ("Incorrect format or blank");
        }
        if (inputLat < -90 | inputLat > 90 || inputLon < -180 || inputLon
> 180)
        {
            OutputMsgDms.text = "Values out of range";
            throw new Exception ("Latitude and longitude out of range");
        }
    }
}
```

OBJECTIVE 8.1: User must enter the year and a set of coordinates to export a report of climate data



```
    }  
}  
  
    // retrieves year inputted from input field and parses to integer  
    year = int.Parse(YearInput.text);  
  
    // loads data from csv and formats it into the list created for each  
parameter  
    temperatureData = LoadData(@"unity  
programs\final_unity\Assets\Resources\tavg_regression_results.csv");  
    precipitationData = LoadData(@"unity  
programs\final_unity\Assets\Resources\prcp_regression_results.csv");  
    snowfallData = LoadData(@"unity  
programs\final_unity\Assets\Resources\snow_regression_results.csv");  
  
    // creates plot point for inputted latitude and longitude values  
    Plot unknown = new Plot (inputLat, inputLon, 0.0);  
  
    // power used when calculating weight  
    double pValue = 2.0;  
    // retrieves interpolated temp, prcp and snowfall values from  
Interpolate function  
    double interTemp = Interpolate(temperatureData, unknown, pValue);  
    double interPrcp = Interpolate(precipitationData, unknown, pValue);  
    double interSnow = Interpolate(snowfallData, unknown, pValue);  
  
    // use a ternary function to set precipitation to 0 if negative  
    interPrcp = (interPrcp < 0) ? 0 : interPrcp;  
    // use ternary to set snowfall to 0 if value is negative  
    interSnow = (interSnow < 0) ? 0 : interSnow;  
  
    ExportTXT(interTemp, interPrcp, interSnow);  
}  
  
    // subroutine to convert from DMS format to decimals  
    double DMS2Dec(int degrees, int minutes, float seconds)  
    {  
        double decimalValue = degrees + (minutes/60) + (seconds/3600);  
        return decimalValue;  
    }  
  
    List<Plot> LoadData(string path)  
    {  
        string userProfilePath =  
Environment.GetFolderPath(Environment.SpecialFolder.UserProfile);  
        // combines the user profile path and path inputted
```

```
string filePath = Path.Combine(userProfilePath, path);
var lines = new List<string[]>();

using (var read = new StreamReader(filePath))
{
    read.ReadLine();
    // holds each value separated by a comma in the array
    while (!read.EndOfStream)
    {
        var line = read.ReadLine();
        var values = line.Split(",");
        lines.Add(values);
    }
    // converts list to array
    lines.ToArray();
}

List<Plot> contentList = new List<Plot>();
foreach (string[] record in lines)
{
    float slope = float.Parse(record[1]);
    float intercept = float.Parse(record[2]);
    float latitude = float.Parse(record[3]);
    float longitude = float.Parse(record[4]);

    // creates new plot point for each record in the 2D array
    Plot newPoint = new Plot(latitude, longitude,
((slope*year)+intercept));
    // adds each plot point to the list
    contentList.Add(newPoint);
}

return contentList;
}

double Interpolate(List<Plot> points, Plot unknownPoint, double pValue)
{
    double weightSum = 0;
    double weightedVarSum = 0;

    // loops through each plot point in the list of points
    foreach (Plot point in points)
    {
        // retrieve value of weight for each point by calling CalcWeight
function
        double weight = CalcWeight(point, unknownPoint, pValue);
        weightSum += weight;
    }
}
```

```
        // finds sum of weighted temperatures or precipitation values or
snowfall levels
        weightedVarSum += weight * point.VariableData;
    }

    // prevents dividing by zero and error
    if (weightSum == 0)
    {
        return double.NaN;
    }

    // inverse distance weighting interpolation formula
    double interpolatedValue = weightedVarSum / weightSum;
    return interpolatedValue;
}

// function to calculate weights
double CalcWeight (Plot point1, Plot point2, double pValue)
{
    double distance = CalcDistance(point1, point2);

    if (distance == 0)
    {
        return double.MaxValue;
    }

    double weight = 1.0 / Math.Pow (distance, pValue);
    return weight;
}

// calculating distance between two points on a sphere in metres (latitude
and longitude)
double CalcDistance (Plot p1, Plot p2)
{
    // radius of the earth in metres
    const int radius = 6378137;

    // converts values from degrees to radians
    double lati1 = p1.Latitude * Mathf.Deg2Rad;
    double long1 = p1.Longitude * Mathf.Deg2Rad;
    double lati2 = p2.Latitude * Mathf.Deg2Rad;
    double long2 = p2.Longitude * Mathf.Deg2Rad;

    // find difference between latitudes and longitudes
    double latDiff = lati2 - lati1;
    double lonDiff = long2 - long1;
}
```

```
// Using Haversine formulae to calculate distance between two points
on a sphere
double a = Math.Pow(Math.Sin(latDiff / 2.0), 2) + Math.Cos(lati1) *
Math.Cos(lati2) * Math.Sin(lonDiff / 2.0) * Math.Sin(lonDiff / 2.0);
double c = 2.0 * Math.Atan2(Math.Sqrt(a), Math.Sqrt(1.0 - a));

// IDW formula
double distance = radius * c;
return distance;
}

void ExportTXT (double temp, double prcp, double snow)
{
    // gets current time and date
    DateTime currentDateTime = DateTime.Now;
    // finds userprofile path and downloads folder in path
    string downloadsFolder =
Environment.GetFolderPath(Environment.SpecialFolder.UserProfile) +
"\Downloads";
    // formats the date and time
    string now = currentDateTime.ToString("dd-MM-yyyy HH:mm:ss");

    // create file name based on latitude and longitude and year inputted
    string fileName = $"{Math.Round(inputLat, 2)}N {Math.Round(inputLon,
2)}W ClimateReport{year}.txt";
    // obtain final file path by combining download folder path and the
new file name
    string filePath = Path.Combine(downloadsFolder, fileName);
    // create an instance of the StreamWriter class
    var writer = new StreamWriter(filePath, true);

    writer.WriteLine($"Report Created : [{now}]");
    writer.WriteLine($"Year: {year}");
    writer.WriteLine($"Coordinates: {inputLat}, {inputLon}");
    writer.WriteLine($"Annual Mean Temperature: {temp} °C");
    writer.WriteLine($"Annual Precipitation: {prcp} mm");
    writer.WriteLine($"Annual Snowfall: {snow} mm");

    writer.Close();

    if (DecSystem.activeInHierarchy)
    {
        OutputMsgDec.text = "File successfully generated in downloads
folder";
    }
    else
    {

```

```
        OutputMsgDms.text = "File successfully generated in downloads  
folder";  
    }  
    Debug.Log("Exported Text file!");  
}  
}
```

OBJECTIVE 8.2: A report file is created which includes: the year entered, the location entered, the average annual temperature, precipitation and snowfall as well as the time and date of file generation

### Summary of Code

The overall purpose of this C# script is to produce a report where a latitude, longitude (in the format of either decimal or degrees, minutes and seconds) and a date in years is inputted by the user. Then the report produced from this code outputs a text file in the user's Downloads folder – this file would include the average annual temperature (°C), mean annual precipitation (mm) and annual snowfall (mm) at the inputted coordinates, during the inputted year. However, the data from the csv file that holds these data points does not include every possible latitude and longitude on the Earth. So, I used an interpolation algorithm commonly used for geospatial data called Inverse Distancing Weighting to obtain estimated values for each climate parameter regardless of the coordinates. This formula incorporates the distance between the points and the unknown point – finding the distance between points on a sphere was done by using the Haversine formulae.

## Keyboard Controls

### Zoom In and Out on Earth

OBJECTIVES 4.1.1 & 4.1.2: User should be able to zoom in using Ctrl + and zoom out with Ctrl - keys

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class ZoomEarth : MonoBehaviour
{
    [SerializeField] private float zoomLimit = 10f;
    [SerializeField] private float zoomSpeed = 5.0f;

    void Update()
    {
        // gets the current field of view from the camera object
        float fieldOfView = Camera.main.fieldOfView;
        // if the user presses ctrl+ and the field of view is larger than the
        zoom limit
        if (Input.GetKey(KeyCode.LeftControl) && Input.GetKey(KeyCode.Equals)
        && fieldOfView > zoomLimit)
        {
            // decreases the field of view by the zoom speed every second
            fieldOfView -= zoomSpeed * Time.deltaTime;
        }

        // if the user presses ctrl- and the field of view is smaller than 180
        if (Input.GetKey(KeyCode.LeftControl) && Input.GetKey(KeyCode.Minus)
        && fieldOfView < 180)
        {
            // increases the field of view by the zoom speed every second
            fieldOfView += zoomSpeed * Time.deltaTime;
        }
        // clamps the field of view between the zoom limit and 180
        fieldOfView = Mathf.Clamp(fieldOfView, zoomLimit, 180);
        // assign the new field of view to the camera
        Camera.main.fieldOfView = fieldOfView;
    }
}
```

## Rotate Earth

```
using UnityEngine;

public class RotateEarth : MonoBehaviour
{
    // sets default rotation speed
    public float rotateSpeed = 50f;

    void Update()
    {
        // gets keyboard input of left and right arrows
        float X_Axis = Input.GetAxis("Horizontal") * rotateSpeed *
Time.deltaTime;
        // gets keyboard input of up and down arrows
        float Y_Axis = Input.GetAxis("Vertical") * rotateSpeed *
Time.deltaTime;

        transform.Rotate(Y_Axis, X_Axis, 0);
    }
}
```

OBJECTIVE 4.2: User should be able to rotate the earth using the up, down, left and right arrow keys

## *Focused Option*

(Hides all options/buttons on the screen apart from the timeline, legend and the earth)

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class hideShowOptions : MonoBehaviour
{
    public CanvasGroup ToHide;

    public void HideUI()
    {
        // Set the alpha value to 0 to make the UI invisible
        ToHide.alpha = 0f;
        ToHide.blocksRaycasts = false;
        canvasGroup.interactable = false;
    }

    public void ShowUI()
    {
        // Set the alpha value to 1 to make the UI visible
        ToHide.alpha = 1f;
        ToHide.blocksRaycasts = true;
        ToHide.interactable = true;
    }
}
```

OBJECTIVE 3.6: User must have focused option to remove unnecessary UI elements from the screen

```
}  
}
```

## Hovering Text

```
using System.Collections;  
using System.Collections.Generic;  
using UnityEngine;  
  
public class hoverText : MonoBehaviour  
{  
    public GameObject textBox;  
    public GameObject textBG;  
    //called when PointerEnter event is triggered  
    public void ShowText()  
    {  
        textBox.SetActive(true);  
        textBG.SetActive(true);  
    }  
    //called when PointerExit event is triggered  
    public void HideText()  
    {  
        textBox.SetActive(false);  
        textBG.SetActive(false);  
    }  
}
```

OBJECTIVE 3.2: When hovering over a button with a symbol, text must be shown to indicate its purpose

## Settings Menu

```
using System.Collections;  
using System.Collections.Generic;  
using UnityEngine;  
using UnityEngine.UI;  
using TMPro;  
  
public class SettingsMenu : MonoBehaviour  
{  
    public Transform settingsIcon;  
    public TMP_Dropdown GraphicsDropdown;  
    public TMP_Dropdown ResolutionDropdown;  
    public Toggle toggle;  
    public Image toggleOn;  
    public Image toggleOff;  
  
    Resolution[] resolutions;
```



```
void Start()
{
    // gets all possible resolutions for the screen
    resolutions = Screen.resolutions;
    // remove all options from drop down
    ResolutionDropdown.ClearOptions();
    // create a new list to hold the new options for the dropdown
    List<string> newOptions = new List<string>();
    int currentResIndex = 0;

    for (int i = 0; i < resolutions.Length; i++ )
    {
        string option = $"{resolutions[i].width} x
{resolutions[i].height}";
        newOptions.Add(option);

        //checks for resolution that matches current resolutions
        if (resolutions[i].width == Screen.currentResolution.width &&
resolutions[i].height == Screen.currentResolution.height)
        {
            // obtains index for the current resolution set
            currentResIndex = i;
        }
    }

    // adds options in list to options part of the dropdown
    ResolutionDropdown.AddOptions(newOptions);
    ResolutionDropdown.value = currentResIndex;
    // refreshes the options in the dropdown
    ResolutionDropdown.RefreshShownValue();
}

public void SettingResolution(int resIndex)
{
    // gets index of option selected in dropdown and resolution
    Resolution setResolution = resolutions[resIndex];
    Screen.SetResolution(setResolution.width, setResolution.height,
Screen.fullScreen);
}

public void SetQuality(int qualityIndex)
{
    // sets the quality in project settings
    QualitySettings.SetQualityLevel(qualityIndex);

    // sets the background of the colour depending on quality selected
    switch (qualityIndex)
    {
```

OBJECTIVE 3.5.2 : User should  
be able to adjust resolution in  
settings

OBJECTIVE 3.5.1: User should be  
able to adjust graphics quality in  
settings

```
        case 0:
            GraphicsDropdown.image.color = new Color (0.8773585f,
0.2589695f, 0.2524475f, 1f); //red
            break;
        case 1:
            GraphicsDropdown.image.color = new Color (0.8773585f,
0.5677743f, 0.2524475f, 1f); //orange
            break;
        case 2:
            GraphicsDropdown.image.color = new Color (0.8153692f,
0.8773585f, 0.2524475f, 1f); //yellow
            break;
        case 3:
            GraphicsDropdown.image.color = new Color (0.4454046f,
0.8773585f, 0.2524475f, 1f); //green
            break;
        case 4:
            GraphicsDropdown.image.color = new Color (0.505162f,
0.7594199f, 0.8301887f, 1f); //blue
            break;
    }
}

void Update ()
{
    // rotates the cog image in the bacground constantly in the z axis by
10 degrees per second
    settingsIcon.transform.Rotate (0 , 0, 10 * Time.deltaTime);
}

public void SetFullscreen(bool fullScreenOn)
{
    // changes true false value to float
    float value = (fullScreenOn) ? 1f:0f;
    // sets alpha value toggle graphic depending on value of toggle
    toggleOn.color = new Color (0, 0, 0, value);
    toggleOff.color = new Color (0, 0, 0, 1-value);
    // sets screen to fullscreen
    Screen.fullScreen = fullScreenOn;
}
}
```

OBJECTIVE 3.5.3: User should be able to toggle fullscreen on and off in settings

## Summary of Code

There are three possible aspects of the program that can be altered in the settings menu by the user : the graphics quality, the resolution and fullscreen mode. The user is able to access these through dropdowns and toggles which trigger their respective subroutine in this script. The

Page | 90

graphics quality is altered by accessing the quality settings within the project settings. The possible resolutions for the device are retrieved in the Start() function and the user can choose between these in the dropdown. As for the fullscreen, the Screen.fullScreen attribute is set to true when the toggle is on and false when the toggle is off.

## Section 4: Testing

### *Overview*

In this section, I will be assessing whether my program has met the objectives I have formed initially in the Analysis section through a series of tests. These will be documented in a testing video and a testing plan tables will act as the framework for the tests done in the video. In my testing video, I have added text captions along with a voiceover to explain what is happening. With the text captions, yellow text represents objectives that are being met with the features I am showing. Blue text represents the test in the testing table that is being referenced. Green or red text shows whether the test was successful or not. I will also make a short demonstration video which swiftly moves through the program, showing its main features.

### *Input & Output Testing Plans*

A tabular form of results will assist to ensure that the system I have created handles inputs and outputs correctly and navigates correctly through the sections of the program.

I have formed this table to display the various tests I will carry out and the success of their outcomes with three types of testing inputs – typical, erroneous, and extreme data. Typical data should give the correct response; erroneous data should throw an exception and output a message on the screen; extreme data is less expected data that could appear on the boundary in ranges. The colour of the row in the table dictates whether the test was successful with green as success and red being failure.

## Login and Registration Testing

Test No.	Test Description	Test Type (TEX)	Objectives	Input Data	Expected Result	Outcome
1	Login button leads to login form	Typical	Obj, 1.1	Press on 'Login' button	Navigates to login form page	As Expected
2	Register button leads to registration form	Typical	Obj. 1.1	Press on 'Register' button	Moves to the registration form page	As Expected
3	Valid credentials for registration form (not registered already but also an employee)	Typical	Obj. 1.3, Obj. 1.4	First name: Joanne Surname: Welt Employee ID: 7777 Password: Climate@999	Output message that user was created successfully	As Expected
4	First name and surname that match database but incorrect employee ID in registration.	Erroneous	1.3, 1.4	First name: Joanne Surname: Welt Employee ID: 9238 Password: Climate@999	Outputs "Invalid Employee ID" as a message on the screen	As Expected
5	Details of account that is already created when registering	Erroneous	Obj. 1.4	First name: Jordan Surname: Carter Employee ID: 1629 Password: Su\$tainable2024	Outputs "Account already exists"	As Expected

Test No.	Test Description	Test Type (TEX)	Objectives	Input Data	Expected Result	Outcome
6	Creates username with successful registration	Typical	Obj. 1.5	First name: Joanne Surname: Welt Employee ID: 7777 Password: Climate@999	Creates username and outputs to screen: "jwelt7777"	As Expected
7	Must have strong password to register (at least 8 characters, at least one digit, special character and both cases of letters)	Typical	Obj. 1.6	Password: "Climate@999"	Proceeds to next part of registration validation	As Expected
8	Non-matching password and repeat password fields	Erroneous	Obj. 1.6	Password: "climate" Repeat password: "earth"	Output "Passwords do not match"	As Expected
9	Weak password inputted in registration form	Erroneous	Obj. 1.6	Password: "WeakPass123" (no special char)	Output "Password must meet requirements"	As Expected
				Password: "W3ak!" (too short)		As Expected
				Password: "weakPass!" (no digits)		As Expected

Test No.	Test Description	Test Type (TEX)	Objectives	Input Data	Expected Result	Outcome
				Password: "weak@999" (no upper case)		As Expected
10	Password on the boundary of requirements: 8 characters long and meeting all other requirements exactly once.	Extreme	Obj. 1.6	First name: Denise Surname: Acosta Employee ID: 2837 Password: "Str0ng9!"	Proceed to next part of registration validation	As Expected
11	Valid login details in login form	Typical	Obj. 2.1	Username: "jwelt7777" Password: "Climate@999"	Proceeds to main program	As Expected
12	Username that has not been created is entered in the login form	Erroneous	Obj. 2.1	Username: "paul555" Password: "Climate@999"	Output "Username does not exist"	As Expected

Test No.	Test Description	Test Type (TEX)	Objectives	Input Data	Expected Result	Outcome
13	Existing username with incorrect password is entered in the login form	Erroneous	Obj. 2.1	Username: "jwelt7777" Password: "Inc0rrect\$47"	Output: "Username or password is incorrect"	As Expected



### Earth Interactivity Testing

Test No.	Test Description	Test Type (TEX)	Objectives	Input Data	Expected Result	Outcome
14	'Ctrl +' should be used to zoom into Earth model	Typical	Obj, 4.1.1	Press Ctrl + keys	Zoom into the earth	As Expected
15	'Ctrl -' should zoom out from the earth	Typical	Obj. 4.1.2	Press Ctrl - keys	Zooms out from the Earth	As Expected
16	Use of arrow keys for rotation of the earth	Typical	Obj. 4.2	Press Up arrow key Press Down arrow key Press Left arrow key Press Right arrow key	Up and down keys rotate the earth towards and away from the user. Left and right rotate the earth on its axis.	As Expected

### Navigation of Program Testing

Test No.	Test Description	Test Type (TEX)	Objectives	Input Data	Expected Result	Outcome
17	Hovering over a button with a symbol reveals text to indicate the purpose of the button	Typical	Obj. 3.2	Hover over '?' button in registration form	Text displayed "Click to see password requirements"	As Expected

Test No.	Test Description	Test Type (TEX)	Objectives	Input Data	Expected Result	Outcome
18	A help button should be available to show information about the key controls in the program	Typical	Obj. 3.3	Press "HELP" button	Display screen showing the purpose of the arrow keys and other controls	As Expected
19	Set Full screen mode on and off	Typical	Obj. 3.5.3	Set 'Fullscreen' toggle ON in settings	Program is put in fullscreen	As Expected
				Set 'Fullscreen' toggle OFF in settings	Program exits fullscreen	As Expected
20	Adjust the Graphics quality in settings	Typical	Obj. 3.5.1	Set quality to 'Low' in dropdown	Quality drops (and quality is set to Low in project settings)	As Expected
				Set quality to 'Medium' in dropdown	Graphics quality is set to medium in the project settings	As Expected

Test No.	Test Description	Test Type (TEX)	Objectives	Input Data	Expected Result	Outcome
				Set quality to 'Very High' in dropdown	High quality is visible in contrast to other settings (and quality settings set to Very High in project settings)	As Expected
21	Adjust the resolution in settings	Typical	Obj. 3.5.2	Select a variety of resolution options – these will vary depending on the device.	Aspect ratios of the screen should change when resolutions are changed – also a difference in quality may be found	As Expected

### Earth Visualisation Testing

Test No.	Test Description	Test Type (TEX)	Objectives	Input Data	Expected Result	Outcome
22	Check if the dot points on the earth and their colours change appropriately with a different climate parameter selected	Typical	Obj. 5.1	Switch on each of the parameter toggles in turn (temperature, precipitation and snowfall)	Change in abundance of dots and colours of dots	As Expected
23	Check if the dot points on the earth and their colours change appropriately with a change in the date inputted through the date slider	Typical	Obj. 6.1 Obj. 6.2	Move the slider from 1970 to 2050 for each parameter	Change in the colour of dots in one direction (individual dots getting colder or warmer)	As Expected

### Coordinate Lookup Testing

Test No.	Test Description	Test Type (TEX)	Objectives	Input Data	Expected Result	Outcome
24	Enter decimal degrees coordinates of	Typical	Obj. 7.1 Obj. 7.2	Latitude: 51.509865 Longitude: -0.118092	View shifts to London, England area	As Expected

Test No.	Test Description	Test Type (TEX)	Objectives	Input Data	Expected Result	Outcome
	known cities or countries			Latitude: 40.730610 Longitude: -73.935242	View shifts to New York, USA	As Expected
				Latitude: 19.076090 Longitude: 72.877426	View shifts to Mumbai, India	As Expected
25	Enter DMS coordinates of known cities or countries	Typical	Obj. 7.1	Latitude: D: 51; M: 30; S: 30.7 Longitude: D: 0; M: 7; S: 32.7	View shifts to London, England area	As Expected
				Latitude: D: 40; M: 43; S: 50.1960 Longitude: D: 73; M: 53'; S: 6.8712"	View shifts to New York, USA	As Expected
				Latitude: D: 19; M: 4; S: 33.9240 Longitude: D: 72; M: 52'; S: 38.7336	View shifts to Mumbai, India	As Expected

Test No.	Test Description	Test Type (TEX)	Objectives	Input Data	Expected Result	Outcome
26	Enter latitude and longitude values out of their range	Erroneous	Obj. 7.1	Latitude = 100.123 Longitude = -217.93	Should output error message as "Latitude and/or longitude out of range"	As Expected
27	Leave latitude and longitude fields blank	Extreme	Obj. 7.1	Latitude = "" Longitude = ""	Outputs "Incorrect Format" message to screen	As Expected
28	Enter boundary values for the latitude and longitude	Extreme	Obj. 7.1	Latitude = 90 Longitude = 180	Moves view to the North Pole	As Expected
29	Enter decimal values for degrees and minutes of the DMS inputs	Erroneous	Obj. 7.1	Latitude = D: 10.1 M: 99.73 S: 89.23 Longitude = D: 56.13 M: 79.27 S: 10.62	Output message "Degrees and Minutes must be integers"	As Expected

### Export Report Testing

Test No.	Test Description	Test Type (TEX)	Objectives	Input Data	Expected Result	Outcome
30	Enter valid values to export a report	Typical	Obj. 8.1 Obj. 8.2	Latitude: 40.730610 Longitude: -73.935242 Year: 2020	Outputs text file in downloads folder with all climate parameter details (year entered, location entered, annual mean temperature, annual mean precipitation and annual snowfall) and time and date of file generation	As Expected
31	Enter invalid values for latitude and longitude	Erroneous	Obj. 8.1	Latitude: hello9823 Longitude: 100283	Outputs error message as "Incorrect format"	As Expected
32	Enter out of logical range values for latitude and longitude	Erroneous	Obj. 8.1	Latitude: 839 Longitude: 200	Outputs message "Latitude and/or longitude is out of range"	As Expected

Test No.	Test Description	Test Type (TEX)	Objectives	Input Data	Expected Result	Outcome
33	Leave all fields blank	Extreme	Obj. 8.1	Blank	Outputs "Incorrect Format"	As Expected
34	Enter valid DMS values and year to export a report of parameters at those coordinates and during that year	Typical	Obj. 8.2	Latitude: D: 51; M: 30; S: 30.7 Longitude: D: 0; M: 7; S: 32.7	Outputs text file in downloads folder of the user which includes values of all of the climate variables and the date/time of the file generation.	As Expected



## Section 5: Evaluation

### Overview

I will go through each objective and declare whether I have or have not met an objective. Also, I have given the installation files to my client, Ms Blakesee, and gave a few general user instructions to navigate through the program. I have asked her to feedback on the 'ease of use' of the program, whether her requirements were met, any 'criticisms' and 'potential improvements' I could make - she replied with her notes on each of these prompts.

### Objective Evaluation

This is a table of all of my original objectives and how they have been applied in my program. The colour of the right column cell corresponds whether that objective has been met or not by the following key:

MET OBJECTIVE	NOT MET OBJECTIVE
---------------	-------------------

Obj. No.	Original Objective	Completed Application in System
1	New users must register and create an account to proceed to login	When user enters the program, they are greeted with a login screen and a registration option. The registration form has the fields: first name, surname, employee ID and password (and a 'repeat password'). Only people with the employee ID and name that matches that of the employee database are allowed to register. Once registered, a unique username is created for the user.

2	Existing users must be able to login before proceeding to the main part of the program	The login screen has the fields: username and password and only valid credentials of a registered account allow the user through.
3	The application must be easy to navigate and accessible	I have labelled all of my buttons in my program appropriately so the user is able to navigate the software easily. When I use a symbol to label a button I have added a feature so when you hover over the button, text shows the purpose of the button. A help screen was created to show the keyboard controls. Also, a settings menu was created to allow the user to adjust various properties of the application.
4	The Earth model must be fully interactive for the user	In order to rotate the earth model, the user can use the arrow keys. To zoom in and zoom out Ctrl + and Ctrl – key combinations can be used respectively.
5	The levels of each climate variable must be presented visually on the 3D Earth model	The user is able to toggle one of the three parameters: temperature, precipitation and snowfall. When toggled, the relevant climate data is plotted on the earth with each plot's colour corresponding to the value of the climate variable. A legend is also shown showing the maximum and minimum values for each parameter.

6	The user should be able to use the timeline to see not only past climate data but also future climate data	The timeline goes from 1970 up to 2050 (which is 26 years into the future). The timeline is a slider which allows the user to move a handle to input a year for the climate data. This climate data plotted on the Earth is then updated immediately for the inputted year. Past data is extrapolated using linear regression (a reliable supervise machine learning model).
7	The user should be able to use a coordinate lookup feature to find the location on the Earth corresponding to the inputted coordinates	The user can either enter latitude and longitude of their desired location in the form of decimal degrees or degrees, minutes and seconds (DMS). Once a valid set of coordinates is entered, the view of the camera is moved towards that location.
8	The user must be able to export a report from this program	I added this feature so that the user can enter a set of coordinates (latitude and longitude in the form of decimal degrees or DMS) and a date in years and this will return a text file to the user's downloads folder consisting of the all climate data at the inputted coordinates during the inputted year.

## Client Feedback

### Ease of use

*"I think the user interface of your software is very simple and easy to use and understand. The controls are well defined and there are no discrepancies with the purpose of any buttons, toggles and drop down features. Many of our employees are under a lot of time constraint, so I am glad there is no steep learning curve to handle this software. The fact that it is straightforward and simple really benefits the employees. Also, I enjoy your selected colour scheme."*

My client seems very satisfied with the 'ease of use' of the program as she found it 'easy' and 'simple' to navigate the user interface of the software.

### **How it meets the requirements**

*"I believe this program fulfils all of the features I stated during our original interview. The program predicts future data using past climate-related data; it also shows this information on a 3D model of the earth and we can create a report with relevant climate-related data for any point on the earth. Also, it ensures that only employees of this firm are able to access this program through its login and registration system."*

### **Criticisms**

*"My main criticism is that the majority of data is crowded in the North America region – especially USA. This is a global company we would prefer if the more data was present in other areas of the world. Also, the data for snowfall is extremely scarce with data only in USA."*

I see where my client is coming from as I can also see how the USA is densely populated with points but other areas such as Africa and Asia are significantly lacking in plot points. Next time, I should find data from other sources that are more global other than the NOAA and concatenate all the data to provide more precision and accuracy to the program.

### **Improvements**

*"In relation to my criticism, this could be improved if data from various sources that are based around the world is used so we can have global data. In the interview we had previously, I mentioned possibly adding a feature to have a light and dark mode to reduce eye strain. This wasn't implemented but the user interface implemented has a dark theme so it is not too harsh on the eyes. Next time this feature could be added."*

## **Self Reflection**

### **Challenges I had to overcome**

What I found very challenging in the early stages of development was sourcing my data. It was very difficult to find reliable sources that provide annual climate data for free in the form of CSV files. Once I found a source, merging all 85000 CSV files together by column name was also time and resource intensive. In the end, I overcame this by using a bash script which was left running for a couple of hours and it outputted what I needed. Another challenge I encountered in this coursework is figuring out how to optimise my earth plotting code and remove redundant,

repetitive code as there are many subroutines with different parameters and return values.

Also, I had initially written the code only for temperature, so I had rewritten and reorganised my code several times to arrive at a solution. I needed data structures that can hold my data that can be accessed, updated and instantiated throughout my code. To overcome this, I had to create two classes – one to hold values of each individual data point and one to hold a list of data points for each climate variable.

### **Possible Improvements**

As my client identified, I need to source my data more diversely as my data points are heavily and densely crowded in the Northern America region, so I need to find sources which provide global data.

Also, if I had the processing power and more time, I think the program would be enhanced if I had a heatmap over the earth rather than data points. The heatmap would be a blended overlay on the earth and would more effectively display changes in climate data values. This, would require a lot more resources and time as mastering shaders in Unity can be difficult and complex, also the GPU of my laptop could not handle this program with this feature.

### **What I learned**

I have not only learnt techniques directly relevant to programming and software development but also general project management techniques and research techniques. I have developed my programming techniques in both C# and Python and even learnt a new language to communicate with my login database – PHP. Through research, I have gained knowledge of many types of machine learning algorithms and machine learning models and their functionality. Additionally, I have become very familiar with Unity 3D and its plethora of features and intricacies. Overcoming its steep learning curve was difficult but I can now apply this skill of using the Unity framework to future projects in creating software.

# Appendix

## References

- Beklemysheva, A. (n.d.). *Why Use Python for AI and Machine Learning?* Retrieved from STEEL KIWI: <https://steelkiwi.com/blog/python-for-ai-and-machine-learning/#:~:text=Benefits%20that%20make%20Python%20the%20best%20fit%20for,add%20to%20the%20overall%20popularity%20of%20the%20language.>
- Boddie, K. (n.d.). *How to Convert Degrees, Minutes & Seconds to Decimal Degrees.* Retrieved from Study.com: <https://study.com/skill/learn/how-to-convert-degrees-minutes-seconds-to-decimal-degrees-explanation.html>
- Boeck, M., Jackson, R., & Shaftel, H. (2022, November 12). *Climate Time Machine.* Retrieved from NASA: <https://climate.nasa.gov/interactives/climate-time-machine>
- Enders, F. B. (2020, July 6). *Coefficient of determination.* Retrieved from Encyclopedia Britannica: <https://www.britannica.com/science/coefficient-of-determination>
- Firdose, T. (2023, June 7). *Linear Regression — Derivation of slope and Intercept using Ordinary Least Square.* Retrieved from Medium: <https://tahera-firdose.medium.com/linear-regression-derivation-of-slope-and-intercept-using-ordinary-least-square-971534ec6b77#:~:text=The%20estimated%20slope%20is%20obtained,its%20mean%20across%20all%20observations.>
- GIS Geography. (2023, October 29). *Inverse Distance Weighting (IDW) Interpolation.* Retrieved from GIS Geography: <https://gisgeography.com/inverse-distance-weighting-idw-interpolation/v>
- Kettle, S. (2017, May 10). *Distance on a sphere: The Haversine Formula.* Retrieved from ESRI Community: <https://community.esri.com/t5/coordinate-reference-systems-blog/distance-on-a-sphere-the-haversine-formula/ba-p/902128#:~:text=All%20of%20these%20can%20be,longitude%20of%20the%20two%20points.>
- Lahoti, S. (2019, December 16). *Harrison Ferrone explains why C# is the preferred programming language for building games in Unity.* Retrieved from Packt hub:

<https://hub.packtpub.com/harrison-ferrone-why-c-preferred-programming-language-building-games-unity/>

Lavanya, B., Ranjith, R., & Navaneetha Krishnan, M. (2022). *Climate Change Prediction Using ARIMA Model*. IJRASET.

Lazzeri, F. (2020). *Machine Learning for Time Series Forecasting with Python*. Wiley.

Matec, I. (2023, May 17). *ML.NET – Introduction to Machine Learning With C#*. Retrieved from CodeMaze: <https://code-maze.com/csharp-mlnet-machine-learning-introduction/>

Nykamp, D. Q. (n.d.). *Spherical Coordinates*. Retrieved from Math Insight: [https://mathinsight.org/spherical\\_coordinates](https://mathinsight.org/spherical_coordinates)

Patel, H. (2018, July 6). *TensorFlow Pros and Cons — The Bright and the Dark Sides*. Retrieved from Medium: <https://medium.com/@patelharshali136/tensorflow-pros-and-cons-the-bright-and-the-dark-sides-5fefbb11fb96>

Rakshit. (n.d.). *Linear Regression Using Tensorflow*. Retrieved from GeeksforGeeks: <https://www.geeksforgeeks.org/linear-regression-using-tensorflow/>

Reddy, M. P., Aneesh, A., Praneetha, K., & Vijay, S. (2021). Global Warming Analysis and Prediction Using Data Science. *Fifth International Conference on I-SMAC (IoT in Social, Mobile, Analytics and Cloud)*, 1055-1059. doi:10.1109/I-SMAC52330.2021.9640944.

## Resources Used to Code

1. Udemy Course – A Beginner’s Guide to Machine Learning
2. Udemy Course – A Beginner’s Guide to Machine Learning in Unity
3. ‘Brackeys’ YouTube Channel
4. StackOverflow forums
5. Unity forums
6. Unity Official Beginner Guides