

DEVELOPMENT

PROTOTYPE 1 - INITIAL LOGIN AND REGISTRATION PAGES

In this prototype, I will fully complete the functionality of logging in and registering.

ITERATION 1 - SETTING UP THE DATABASE AND USER INTERFACE

I started by importing the required libraries and setting up the database with the tables described in the DATABASE section. To learn how to use sqlite3 for these SQL commands, I used TutorialsPoint²³ to ensure I was formatting my SQL queries correctly. Below is the code for this:

1.1.a:

```

1 from tkinter import *
2 import tkinter as tk
3 import sqlite3
4
5 conn1 = sqlite3.connect("AlgorithmTeachingToolDB.db")
6 print("open successful")
7
8 conn1.execute('''CREATE TABLE IF NOT EXISTS Users
9             (Username      TEXT      NOT NULL      PRIMARY KEY,
10              Password      TEXT      NOT NULL,
11              Account_type  TEXT      NOT NULL ) ;'''')
12
13 conn1.execute('''CREATE TABLE IF NOT EXISTS StudentEnrolment
14             (StudentEnrolID INTEGER   PRIMARY KEY AUTOINCREMENT,
15              Username      TEXT      NOT NULL,
16              Algorithm     TEXT      NOT NULL,
17              CorrectScore  INT       NOT NULL,
18              IncorrectScore INT      NOT NULL,
19              FOREIGN KEY (Username) REFERENCES Users.Username ) ; ''')
20
21 conn1.execute('''CREATE TABLE IF NOT EXISTS TeacherEnrolment
22             (TeacherEnrolID INTEGER   PRIMARY KEY AUTOINCREMENT,
23              Username      TEXT      NOT NULL,
24              Algorithm     TEXT      NOT NULL,
25              FOREIGN KEY (Username) REFERENCES Users.Username ) ; ''')
26 conn1.commit()
27 conn1.close()
```

Then, I set up the Main class, which is the parent class for the LoginRegister subclass.

1.1.b:

```

30 class Main:
31     def __init__(self, master):
32         self.master = master # this is the window
33         self.title = None
34         # database connection:
35         self.conn = sqlite3.connect("AlgorithmTeachingToolDB.db") # actual database
36         self.cursor = self.conn.cursor() # for selecting
37
38     def closeConn (self):
39         self.conn.close()

```

Then I initialised the following attributes for the LoginRegister subclass:

1.1.c

```

43 class LoginRegister(Main):
44     def __init__(self, master):
45         super().__init__(master) # inherits all attributes of Main class
46
47         # login frame variables
48         self.loginFrame = None
49         self.username = StringVar()
50         self.password = StringVar()
51
52         # register frame variables
53         self.regFrame = None
54         self.newUsername = StringVar()
55         self.newPassword = StringVar()
56         self.newConfirmPassword = StringVar()
57         # algorithm selection variables - 1 is onvalue, 0 is offvalue
58         self.bubbleSortSelect = IntVar()
59         self.primSelect = IntVar()
60         self.dijkstrasSelect = IntVar()
61         self.simplexSelect = IntVar()
62         self.algorithmsChosen = [self.bubbleSortSelect, self.primSelect, self.dijkstrasSelect, self.simplexSelect]
63         # invalid message variables
64         self.usernameInvalid = None
65         self.passwordInvalid = None
66         self.passwordsMatchInvalid = None
67         self.invalidMessage = None
68         # account selection
69         self.accountType = StringVar()
70
71     self.loginRegisterWidgets()

```

All these attributes are required to store the different values entered in the form and so that widgets can be changed later on in the code if required. For the widgets method, which contains all the widgets used in the frames for the Login and Registration pages, I first wrote the following code for the Login frame:

1.1.d:

```

73     def loginRegisterWidgets(self):
74         self.title = Label(self.master, text="Login")
75         self.title.grid(row=0, column=0, columnspan=5)
76
77         # login frame widgets
78         self.loginFrame = Frame(self.master)
79         Label(self.loginFrame, text="Username:").grid(row=2, column=0, columnspan=2)
80         Label(self.loginFrame, text="Password:").grid(row=3, column=0, columnspan=2)
81         Entry(self.loginFrame, textvariable=self.username).grid(row=2, column=2, columnspan=3)
82         Entry(self.loginFrame, textvariable=self.password, show="*").grid(row=3, column=2, columnspan=3)
83         Label(self.loginFrame, text="").grid(row=5, column=0, columnspan=5) # unsuccessful login text
84         Button(self.loginFrame, text="Login", command=self.loginCheck).grid(row=6, column=1)
85         Button(self.loginFrame, text="Register", command=self.newRegistrationPage).grid(row=6, column=3)
86
87         self.loginFrame.grid(row=1, column=0, columnspan=5) # first frame accessed

```

And the following for the Register frame:

1.1.e:

```

89     # register frame widgets
90     self.regFrame = Frame(self.master)
91     Label(self.regFrame, text="Usernames should be longer than 6 characters and unique.\nPasswords should have at least 8 characters, uppercase and\\nlowercase").grid(row=2, column=0, columnspan=2)
92     Label(self.regFrame, text="Username:").grid(row=2, column=0, columnspan=2)
93     Label(self.regFrame, text="Password:").grid(row=3, column=0, columnspan=2)
94     Label(self.regFrame, text="Password confirmation:").grid(row=4, column=0, columnspan=2)
95     Label(self.regFrame, text="Select algorithms to study:").grid(row=5, column=0, columnspan=2)
96     Label(self.regFrame, text="Select account type:").grid(row=7, column=0, columnspan=2)
97     # entry fields:
98     Entry(self.regFrame, textvariable=self.newUsername).grid(row=2, column=2, columnspan=2)
99     Entry(self.regFrame, textvariable=self.newPassword).grid(row=3, column=2, columnspan=2)
100    Entry(self.regFrame, textvariable=self.newConfrimPassword).grid(row=4, column=2, columnspan=2)
101    # invalid messages:
102    self.usernameInvalid = Label(self.regFrame, text="").grid(row=2, column=4, columnspan=2)
103    self.passwordInvalid = Label(self.regFrame, text="").grid(row=3, column=4, columnspan=2)
104    self.passwordsMatchInvalid = Label(self.regFrame, text="").grid(row=4, column=4, columnspan=2)
105    self.invalidMessage = Label(self.regFrame, text="").grid(row=8, column=0, columnspan=4)
106    # algorithm checkboxes:
107    Checkbutton(self.regFrame, text="Bubble Sort", variable=self.bubbleSortSelect, onvalue=1, offvalue=0).grid(row=5, column=2, columnspan=2)
108    Checkbutton(self.regFrame, text="Prim's Algorithm", variable=self.primSelect, onvalue=1, offvalue=0).grid(row=5, column=4, columnspan=2)
109    Checkbutton(self.regFrame, text="Dijkstra's Algorithm", variable=self.dijkstraselect, onvalue=1, offvalue=0).grid(row=6, column=2, columnspan=2)
110    Checkbutton(self.regFrame, text="Simplex Algorithm", variable=self.simplexSelect, onvalue=1, offvalue=0).grid(row=6, column=4, columnspan=2)
111    # account selection radiobutton:
112    Radiobutton(self.regFrame, text="Student", variable=self.accountType, value="Student").grid(row=7, column=2, columnspan=2)
113    Radiobutton(self.regFrame, text="Teacher", variable=self.accountType, value="Teacher").grid(row=7, column=4, columnspan=2)
114    # buttons
115    Button(self.regFrame, text="Register", command=self.registerCheck).grid(row=8, column=4)
116    Button(self.regFrame, text="Back to Login", command=self.newLoginPage).grid(row=8, column=5)

```

The above code calls 4 different methods. As this first iteration focuses on creating the basic features of the GUI, I have added print messages to the validation methods (as these will be completed in the next iterations) so that I can check if pressing the different buttons causes the program to go to the correct method. However, I did complete the functionality for the methods where the frame is changed.

1.1.f:

```

118     def loginCheck(self):
119         # if pass - go to closeLoginOpenHome
120         print("in login check")
121
122     def registerCheck(self):
123         ...
124
125         check username is unique - if not, self.usernameInvalid["text"] = "Username unavailable"
126         check passwords match - if not, self.passwordsMatchInvalid["text"] = "Passwords do not match"
127         check password valid - if not, self.passwordInvalid["text"] = "Password invalid"
128         if pass - go to closeLoginOpenHome
129         ...
130
131     def newRegistrationPage(self):
132         self.title["text"] = "Create an account"
133         print("in new registration page")
134         self.loginFrame.grid_forget()
135         self.regFrame.grid(row=1, column=0, columnspan=6)
136
137
138     def newLoginPage(self):
139         self.title["text"] = "Login"
140         print("in new login page")
141         self.regFrame.grid_forget()
142         self.loginFrame.grid(row=1, column=0, columnspan=6)

```

Finally, I wrote the following code to set up the Tkinter window for the LoginRegister method:

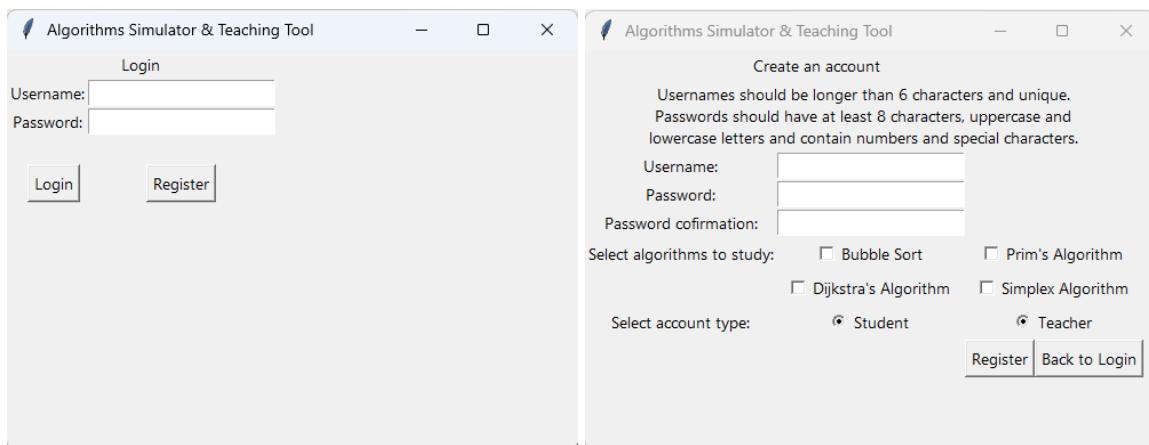
1.1.g:

```

154     if __name__ == "__main__":
155         loginRoot = tk.Tk()
156         loginRoot.geometry('500x350+300+300')
157         loginRoot.title("Algorithms Simulator & Teaching Tool")
158         running1 = LoginRegister(loginRoot)
159         loginRoot.mainloop()

```

Then I ran the code to check that the outputs are as expected. Below are the Login and Registration windows:



And these were the outputs to the Shell when each button was pressed - ‘Login’ and ‘Register’ on the Login page and ‘Register’ and ‘Back to Login’ on the Register page in that order. Furthermore, the ‘open successful’ output signifies that the database has been correctly created and opened, which is also visible in my DB Browser (shown below).

```
>>> %Run NEA_main_file.py
```

```
open successful
in login check
in new registration page
in register check
in new login page
```

The screenshot shows the DB Browser for SQLite interface. The top menu bar includes 'New Database', 'Open Database', 'Write Changes', 'Revert Changes', 'Open Project', 'Save Project', and 'Attach Database'. Below the menu is a toolbar with icons for 'Database Structure', 'Browse Data', 'Edit Pragmas', 'Execute SQL', 'Create Table', 'Create Index', and 'Print'. The main area is titled 'Name' and lists 'Tables (4)'. Under 'Tables (4)', there are four entries: 'StudentEnrolment', 'TeacherEnrolment', 'Users', and 'sqlite_sequence'. To the right of each table name is its corresponding CREATE TABLE SQL statement. The 'Indices (0)', 'Views (0)', and 'Triggers (0)' sections are listed below.

Name	Type	Schema
Tables (4)		
StudentEnrolment		CREATE TABLE StudentEnrolment (StudentEnrolID INTEGER PRIMARY KEY AUTOINCREMENT, Username TEXT NOT NULL, Password TEXT NOT NULL, Account_type TEXT NOT NULL);
TeacherEnrolment		CREATE TABLE TeacherEnrolment (TeacherEnrolID INTEGER PRIMARY KEY AUTOINCREMENT, Username TEXT NOT NULL, Password TEXT NOT NULL, Algorithm TEXT NOT NULL);
Users		CREATE TABLE Users (Username TEXT NOT NULL PRIMARY KEY, Password TEXT NOT NULL, Account_type TEXT NOT NULL);
sqlite_sequence		CREATE TABLE sqlite_sequence(name,seq);
Indices (0)		
Views (0)		
Triggers (0)		

As this is just an initial set up, the Iterative Test Data for the Initial Login and Registration pages have not been met as the functionality for checking the data for the different fields for login and registration has not been developed in this iteration. However, the Shell outputs signify that the code is functioning as intended at the current stage. Additionally the initial setup of the GUI is as expected but I will refine it in a later iteration so that it appears closer to how I designed each layout in the DESIGN sections.

ITERATION 1 - DEVELOPER REVIEW

The set up of the database and the initial GUI for the login and registration pages went very smoothly. Having created an entity relationship diagram and all the SQL queries helped me with the database. Considering that this was the first time I used Tkinter (although I have used other Python libraries for graphical user interfaces), I think development was quite straightforward.

ITERATION 2 - LOGIN AND REGISTRATION FUNCTIONALITY

I am going to begin by completing the functionality of the login system. To be able to test the login functionality throughout the development of this code, I inserted multiple values into the database using DB Browser for SQLite. Below you can see the state of the 3 tables with these values input:

The screenshot shows two tables in DB Browser for SQLite. On the left is the 'Users' table with columns 'Username', 'Password', and 'Account_type'. It contains 6 rows of data. On the right is the 'TeacherEnrolment' table with columns 'TeacherEnrolID', 'Username', and 'Algorithm'. It also contains 6 rows of data.

	Username	Password	Account_type
1	Test_accS	Testing1...	Student
2	Test_accT	Testing4...	Teacher
3	Billy123	1H087w...	Student
4	HelloWor...	1m@gin...	Student
5	123abc	cF2E/4.3U	Teacher
6	idkWhatT...	Mi[s]XcC...	Teacher

	TeacherEnrolID	Username	Algorithm
1	1	Test_accT	Prim's
2	2	Test_accT	Dijkstra's
3	3	123abc	Prim's
4	4	123abc	Simplex
5	5	idkWhatT...	Prim's
6	6	idkWhatT...	Dijkstra's

Table: StudentEnrolment

	StudentEnrolID	Username	Algorithm	CorrectScore	IncorrectScore
	Filter	Filter	Filter	Filter	Filter
1	1	Test_accS	Bubble Sort	3	2
2	2	Test_accS	Simplex	1	5
3	3	Billy123	Bubble Sort	0	0
4	4	HelloWor...	Bubble Sort	2	6
5	5	HelloWor...	Prim's	4	1
6	6	HelloWor...	Dijkstra's	5	0
7	7	HelloWor...	Simplex	0	3

Using these values I tested the login system using the white box testing table from the INITIAL LOGIN PAGE design section.

1.2.a:

```
119 def loginCheck(self):
120     # if pass - go to closeLoginOpenHome
121     print("in login check")
122     validated = False
123     while not validated:
124         validated = self.validateLoginInput()
125         closeOpen(self.master, "home")
```

Additionally, I need an attribute to allow me to display the error message so I updated the `_init_` and `loginRegisterWidgets` methods:

1.2.b:

```
84     self.loginInvalid = Label(self.loginFrame, text="").grid(row=5, column=0, columnspan=5) # unsuccessful login text
```

1.2.c:

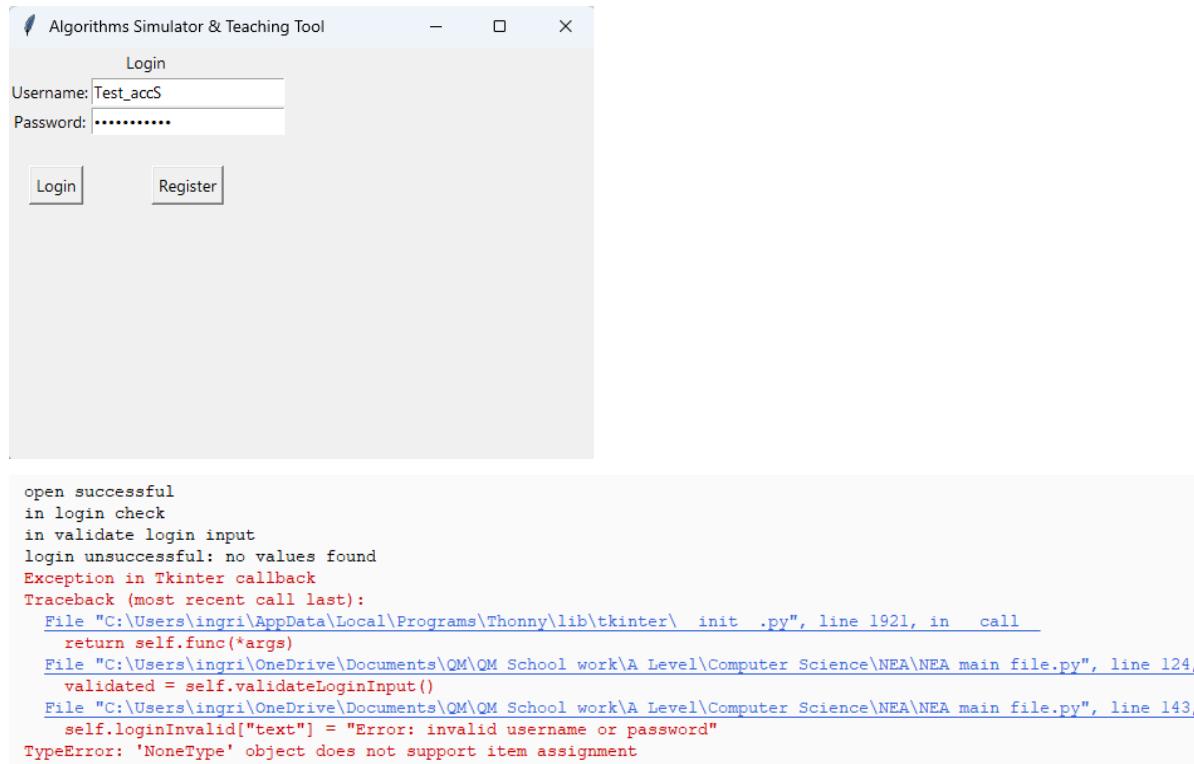
```
47     # login frame variables
48     self.loginFrame = None
49     self.username = StringVar()
50     self.password = StringVar()
51     self.loginInvalid = None
```

Then, I developed the following code for validating the login inputs using the pseudocode written in the DESIGN section. Furthermore, I ensured that I used a tuple in the SQL query in order to reduce risks from SQL injections.

1.2.d:

```
127 def validateLoginInput(self):
128     print("in validate login input")
129     username = self.username.get()
130     password = self.password.get()
131     # check for no username / password entered
132     if not username or not password:
133         self.loginInvalid["text"] = "Error: invalid username or password"
134     # select from database for entered username and password
135     try:
136         self.cursor.execute("SELECT * FROM Users WHERE Username = ? and Password = ?", (username, password))
137         user = self.cursor.fetchone() # returns values or None
138         if user: # not None
139             print("Login successful")
140             return True
141         else:
142             print("login unsuccessful: no values found")
143             self.loginInvalid["text"] = "Error: invalid username or password"
144             return False
145     except sqlite3.Error as e:
146         print(f"Database error: {e}")
147         self.loginInvalid["text"] = "Error: invalid username or password"
148         return False
```

When this code is run, the following is output:



From some research, I have identified that this error occurred because I was trying to assign a value to `self.loginInvalid["text"]` but this attribute had the value `None` due to the `grid` method in Tkinter. To fix this, I changed the `loginRegisterWidgets` method:

1.2.e:

```

93     self.loginInvalid = Label(self.loginFrame, text="") # unsuccessful login tex
94     self.loginInvalid.grid(row=5, column=0, columnspan=5)
    
```

Then I got the following error, which I identified was due to me not clearing the entry fields after an invalid input and not committing all the changes I had made to the database in the DB Browser:

```

in validate login input
login unsuccessful: no values found
in validate login input
login unsuccessful: no values found
in validate login input
login unsuccessful: no values found
in validate login input
login unsuccessful: no values found
in validate login input
    
```

To fix this, I added lines to clear the `username` and `password` attributes in each of the failed validation tests and committed the changes.

1.2.f:

```

self.username.set("")
self.password.set("")
    
```

Having input the details for the `Test_accS` account that I had already created, the shell outputs were as expected:

```
>>> %Run NEA_main_file.py
open successful
in login check
in validate login input
Login successful
login validated
```

The final functionality of the Login function is to close the current window and open the home page. To do this, I wrote the following function, which I will further develop later in development to allow it to be called for multiple functions:

1.2.g:

```
209 def closeOpen(root, newType):
210     print("in closeOpen")
211     root.destroy()
212     if newType == "home":
213         homeRoot = Tk()
214         homeRoot.geometry('500x350+300+300')
215         homeRoot.title("Algorithms Simulator and Teaching Tool")
216         running = HomeMain(homeRoot)
217         homeRoot.mainloop()
```

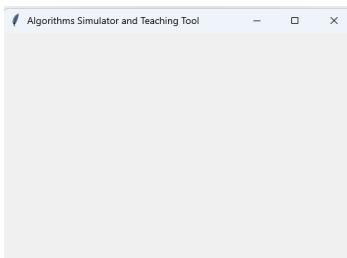
1.2.h:

```
194 class HomeMain(Main):
195     def __init__(self, master):
196         super().__init__(master)
```

After running the code, I get the following error:

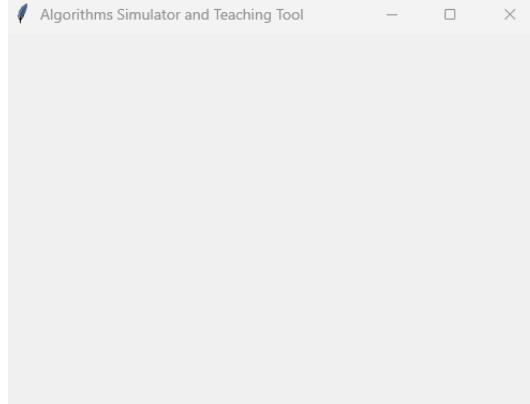
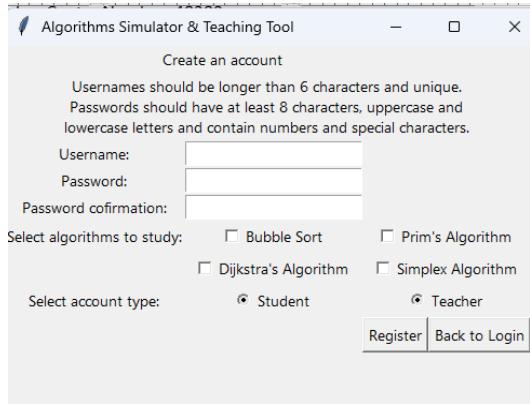
```
>>> %Run NEA_main_file.py
open successful
in login check
in validate login input
Login successful
login validated
Exception in Tkinter callback
Traceback (most recent call last):
  File "C:\Users\ingri\AppData\Local\Programs\Thonny\lib\tkinter\__init__.py", line 1921, in __call__
    return self.func(*args)
  File "C:\Users\ingri\OneDrive\Documents\QM\QM School work\A Level\Computer Science\NEA\NEA main file.py", line 127, in closeOpen
    self.master, "home")
NameError: name 'closeOpen' is not defined
```

I identified that I had to move the function to the beginning of the file (before the code for the Main class) in order for it to be called in the loginCheck method. After doing this and running the code, the output was as expected as the login window was closed and the following new window (the home window) was opened:



The GUI and functionality of the Home page will be developed in PROTOTYPE 2.

Now, I will test the code developed using the white box testing specified in the INITIAL LOGIN design section.

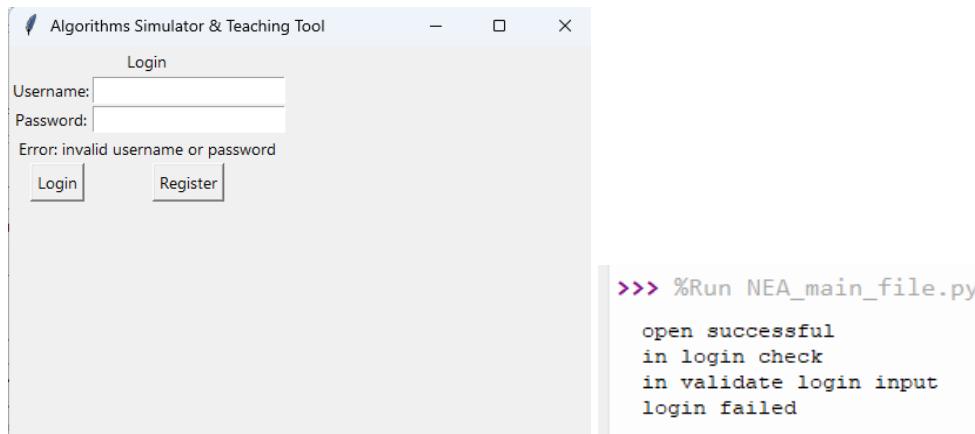
<p>Log in with a correct username and password. E.g. username: Test_accS password: Testing123!</p>	<p>The user should be logged in and redirected to the Home page.</p>	<p>This ensures that the user can log in if the form is completed correctly.</p>	<p>1.1.c 1.2.a 1.2.d 1.2.g 1.2.h</p>		<p>As expected</p>
<p>Press the Register button.</p>	<p>Should redirect the user to the Register page.</p>	<p>This ensures that the 'Register' button works correctly.</p>	<p>1.1.c 1.1.d 1.1.e 1.1.f 1.2.g</p>		<p>As expected</p>

The errors that I got when the details entered were incorrect (either not fully filled in or incorrect values) all stemmed from the same problem. This seemed to be the while loop that I used in the `loginCheck` method. To fix this, I changed the while loop to an if else statement because this would ensure that the program waits for a new input until checking fields again. Below is the code:

1.2.i:

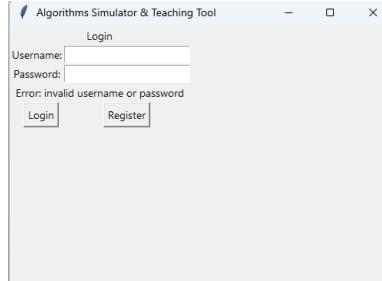
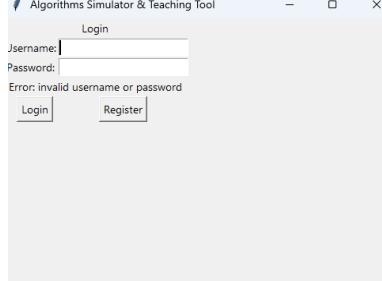
```
147     if self.validateLoginInput(): # is True
148         print("login validated")
149         closeOpen(self.master, "home")
150     else:
151         print("login failed")
```

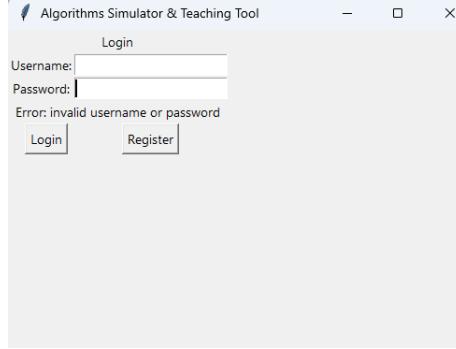
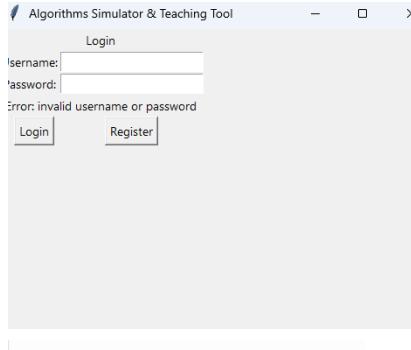
Now when I run the code without entering any details, the following is output:

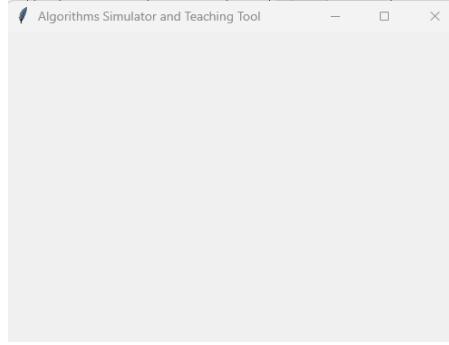
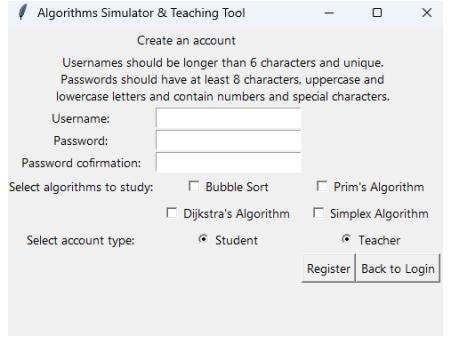


This is as expected so I will now retest this code with the white box testing table.

TEST DATA	EXPECTED RESULT	JUSTIFICATION	CODE	ACTUAL RESULT

Log in without filling in any details.	The user should be prompted with the ‘Unsuccessful login’ message.	This ensures that the user must complete the full form to log in.	1.1.c 1.1.d 1.2.a 1.2.d 1.2.i	 <pre data-bbox="1320 537 1641 664">>>> %Run NEA_main_file.py open successful in login check in validate login input login failed</pre>	Output as expected
Log in with a partially filled in form. E.g. username: Test_accT password blank	The user should be prompted with the ‘Unsuccessful login’ message.	This ensures that the user must complete the full form to log in.	1.1.c 1.1.d 1.2.a 1.2.d 1.2.i	 <pre data-bbox="1320 1014 1641 1141">>>> %Run NEA_main_file.py open successful in login check in validate login input login failed</pre>	Output as expected

<p>Log in with a fully filled in form but the username does not exist. E.g. username: Karu0807 password: Testing123!</p>	<p>The user should be prompted with the ‘Unsuccessful login’ message.</p>	<p>This ensures that the user enters the correct username and cannot log in otherwise.</p>	<p>1.1.c 1.1.d 1.2.a 1.2.d 1.2.i</p>	 <pre data-bbox="1320 605 1664 728">>>> %Run NEA_main_file.py open successful in login check in validate login input login unsuccessful: no values found login failed</pre> <p style="text-align: right;">Output as expected</p>
<p>Log in with a fully filled in form but the password is incorrect for the username. E.g. username: Test_accS password: Tere_talv7</p>	<p>The user should be prompted with the ‘Unsuccessful login’ message.</p>	<p>This ensures that the user must enter the correct password and cannot log in otherwise.</p>	<p>1.1.c 1.1.d 1.2.a 1.2.d 1.2.i</p>	 <pre data-bbox="1320 1140 1664 1262">>>> %Run NEA_main_file.py open successful in login check in validate login input login unsuccessful: no values found login failed</pre> <p style="text-align: right;">Output as expected</p>

<p>Log in with a correct username and password. E.g. username: Test_accS password: Testing123!</p>	<p>The user should be logged in and redirected to the Home page.</p>	<p>This ensures that the user can log in if the form is completed correctly.</p>	<p>1.1.c 1.1.d 1.2.a 1.2.d 1.2.g 1.2.h 1.2.i</p>	 <pre data-bbox="1320 605 1617 764">>>> %Run NEA_main_file.py open successful in login check in validate login input Login successful login validated in closeOpen</pre> <p style="text-align: right;">Output as expected</p>
<p>Press the Register button.</p>	<p>Should redirect the user to the Register page.</p>	<p>This ensures that the 'Register' button works correctly.</p>	<p>1.1.c 1.1.d 1.1.e 1.1.f 1.2.g</p>	 <pre data-bbox="1320 1173 1617 1252">>>> %Run NEA_main_file.py open successful in new registration page</pre> <p style="text-align: right;">Output as expected</p>

After fixing the errors that occurred, all the tests passed as expected so the functionality for the initial login section is fully complete.

Now I am going to complete the second part of the functionality for this PROTOTYPE, which is the Registration functionality. Firstly, following on from error that occurred with `self.loginInvalid["text"]`, I changed the widgets used to show error messages in the register frame so that this error will not occur again:

1.2.j:

```
117     # invalid messages:  
118     self.usernameInvalid = Label(self.regFrame, text="")  
119     self.usernameInvalid.grid(row=2, column=4, columnspan=2)  
120     self.passwordInvalid = Label(self.regFrame, text="")  
121     self.passwordInvalid.grid(row=3, column=4, columnspan=2)  
122     self.passwordsMatchInvalid = Label(self.regFrame, text="")  
123     self.passwordsMatchInvalid.grid(row=4, column=4, columnspan=2)  
124     self.invalidMessage = Label(self.regFrame, text="")  
125     self.invalidMessage.grid(row=8, column=0, columnspan=4)
```

Additionally, after considering how I will access the values for the algorithms chosen, I decided to change the `self.algorithmsChosen` list to a dictionary:

1.2.k:

```
74     self.algorithmsChosen = {"Bubble Sort": self.bubbleSortSelect,  
75                     "Prim's": self.primSelect,  
76                     "Dijkstra's": self.dijkstraSelect,  
77                     "Simplex": self.simplexSelect}
```

This is because a dictionary would allow me to search the values easily and to use the key values when inserting into the database easier as I would not have to program in the values of the indexes.

Then, I wrote the code for validating the username and password inputs:

1.2.l:

```
248     def validateUsernameReg(self):
249         print("in validate username")
250         usernameHolder = self.newUsername.get()
251         self.cursor.execute("SELECT Username FROM Users WHERE Username = ?", (usernameHolder,)) # comma to make tuple
252         usernameFound = self.cursor.fetchone()
253         if usernameFound:
254             return False
255         else:
256             return True
```

1.2.m:

```
258     def validatePasswordReg(self):
259         print("in validate password")
260         passwordHolder = self.newPassword.get()
261         # longer than 8 characters (inclusive)
262         if len(passwordHolder) >= 8:
263             # contains both uppercase and lowercase characters
264             if any(character.isupper() for character in passwordHolder) == True and any(character.islower() for character in passwordHolder) == True:
265                 # contains special chars - held in string.punctuation
266                 if any(character in string.punctuation for character in passwordHolder) == True:
267                     # contains number
268                     if any(character.isdigit() for character in passwordHolder) == True:
269                         return True # all tests passed
270         return False # at least 1 test failed
```

And the code for the registerCheck method:

1.2.n:

```
180 def registerCheck(self):
181     ...
182     check username is unique - if not, self.usernameInvalid["text"] = "Username unavailable"
183     check passwords match - if not, self.passwordsMatchInvalid["text"] = "Passwords do not match"
184     check password valid - if not, self.passwordInvalid["text"] = "Password invalid"
185     if pass - go to closeLoginOpenHome
186     ...
187     usernameHolder = self.newUsername.get()
188     passwordHolder = self.newPassword.get()
189     confirmPasswordHolder = self.newConfirmPassword.get()
190     print("in register check")
191     valid = False
192     while not valid:
193         validUsername = False
194         while not validUsername:
195             validUsername = self.validateUsernameReg()
196             if validUsername == False:
197                 self.usernameInvalid["text"] = "Username not available"
198                 self.newUsername.set("")
199
200         # moves on when validUsername is True
201         # validate password
202         validPassword = False
203         while not validPassword:
204             validPassword = self.validatePasswordReg()
205             if passwordHolder != confirmPasswordHolder and validPassword == False:
206                 self.passwordsMatchInvalid["text"] = "Passwords do not match"
207                 self.passwordInvalid["text"] = "Password invalid"
208             elif validPassword == False:
209                 self.passwordInvalid["text"] = "Password invalid"
210             elif passwordHolder != confirmPasswordHolder:
211                 self.passwordsMatchInvalid["text"] = "Passwords do not match"
212
213         # validate algorithm choice
214         algorithmValues = self.algorithmsChosen.values()
215         validAlgorithms = False
216         if any(value == 1 for value in algorithmValues) == True:
217             validAlgorithms = True
```

1.2.o:

```
219     # validate account type selection
220     accountHolder = self.accountType.get()
221     validAccount = False
222     if accountHolder: # not None
223         validAccount = True
224
225     if validUsername and validPassword and validAlgorithms and validAccount:
226         valid = True
227     else:
228         self.invalidMessageReg["text"] = "Unsuccessful registration - check valid username and password and that at least 1 algorithm and an account type have been selected"
229
230     # enter values into database
231     try:
232         self.cursor.execute("INSERT INTO Users (Username, Password, Account_type) VALUES (?, ?, ?)", (usernameHolder, passwordHolder, accountHolder))
233         self.conn.commit()
234
235         if accountHolder == "Student":
236             for name, numValue in self.algorithmsChosen.items():
237                 if numValue.get() == 1:
238                     self.cursor.execute("INSERT INTO StudentEnrolment (Username, Algorithm, CorrectScore, IncorrectScore) VALUES (?, ?, ?, ?)", (usernameHolder, name, 0, 0))
239                     self.conn.commit()
240         elif accountHolder == "Teacher":
241             for name, numValue in self.algorithmsChosen.items():
242                 if numValue.get() == 1:
243                     self.cursor.execute("INSERT INTO TeacherEnrolment (Username, Algorithm) VALUES (?, ?)", (usernameHolder, name))
244                     self.conn.commit()
245         print("Registration successful")
246
247     # clear inputs after successful registration:
248     self.newUsername.set("")
249     self.newPassword.set("")
250     self.newConfirmPassword.set("")
251     for value in algorithmValues:
252         value.set(0)
253     self.accountType.set("")
254     closeOpen(self.master, "home")
255
256 except sqlite3.Error as e:
257     print(f"Database error: {e}")
```

Running this code, I am able to enter the values in correctly however I get the following output to the Shell:

```
>>> %Run NEA_main_file.py

open successful
in new registration page
in register check
in validate username
in validate password
```

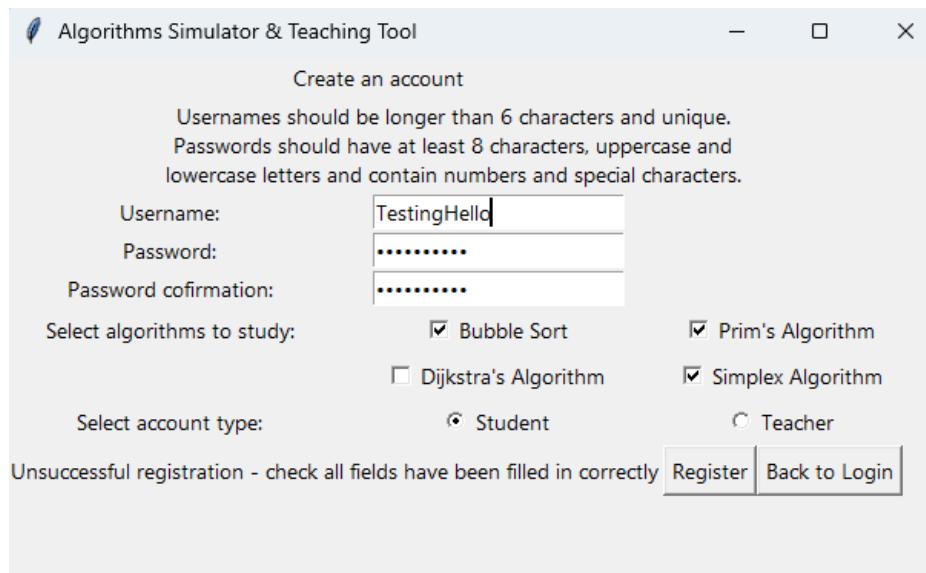
I have identified that this infinite loop is because the code does not break out of a loop if the username or password is invalid. To fix this, I replaced the while loops with if conditions so each is checked only once before moving on:

1.2.p:

```
180 def registerCheck(self):
181     """
182         check username is unique - if not, self.usernameInvalid["text"] = "Username unavailable"
183         check passwords match - if not, self.passwordsMatchInvalid["text"] = "Passwords do not match"
184         check password valid - if not, self.passwordInvalid["text"] = "Password invalid"
185         if pass - go to closeLoginOpenHome
186     """
187
188     usernameHolder = self.newUsername.get()
189     passwordHolder = self.newPassword.get()
190     confirmPasswordHolder = self.newConfrimPassword.get()
191     accountHolder = self.accountType.get()
192
193     print("in register check")
194
195     valid = False
196
197     if not self.validateUsernameReg():
198         self.usernameInvalid["text"] = "Username not available"
199         self.newUsername.set("")
200         return
201
202     # validate password
203     if not self.validatePasswordReg():
204         self.passwordInvalid["text"] = "Password invalid"
205         return
206
207     if passwordHolder != confirmPasswordHolder:
208         self.passwordsMatchInvalid["text"] = "Passwords do not match"
209         return
210
211     # validate algorithm choice
212     algorithmValues = self.algorithmsChosen.values()
213     if not any(value == 1 for value in algorithmValues):
214         self.invalidMessageReg["text"] = "Unsuccessful registration - check all fields have been filled in correctly"
215         return
```

1.2.q:

```
215     # validate account type selection
216     if not accountHolder: # not None
217         self.invalidMessageReg["text"] = "Unsuccessful registration - check all fields have been filled in correctly"
218
219
220     # enter values into database
221     try:
222         self.cursor.execute("INSERT INTO Users (Username, Password, Account_type) VALUES (?, ?, ?)", (usernameHolder, passwordHolder, accountHolder))
223         self.conn.commit()
224         print("inserted into Users")
225
226         if accountHolder == "Student":
227             for name, numValue in self.algorithmsChosen.items():
228                 if numValue.get() == 1:
229                     self.cursor.execute("INSERT INTO StudentEnrolment (Username, Algorithm, CorrectScore, IncorrectScore) VALUES (?, ?, ?, ?)", (usernameHolder, name, 0, 0))
230                     self.conn.commit()
231             print("inserted into StudentEnrolment")
232
233         elif accountHolder == "Teacher":
234             for name, numValue in self.algorithmsChosen.items():
235                 if numValue.get() == 1:
236                     self.cursor.execute("INSERT INTO TeacherEnrolment (Username, Algorithm) VALUES (?, ?)", (usernameHolder, name))
237                     self.conn.commit()
238             print("inserted into TeacherEnrolment")
239
240         print("Registration successful")
241
242         # clear inputs after successful registration:
243         self.newUsername.set("")
244         self.newPassword.set("")
245         self.newConfirmPassword.set("")
246         for value in self.algorithmsChosen.values():
247             value.set(0)
248         self.accountType.set("")
249         closeOpen(self.master, "home")
250     except sqlite3.Error as e:
251         print(f"Database error: {e}")
```



```
>>> %Run NEA_main_file.py
```

```
open successful
in new registration page
in register check
in validate username
in validate password
```

Despite inputting correct details, registration was unsuccessful. As I wasn't sure whether the algorithm selection or the account type selection was causing this error (due to them outputting the same message), I put in print statements to say when a test fails. This output the following when run:

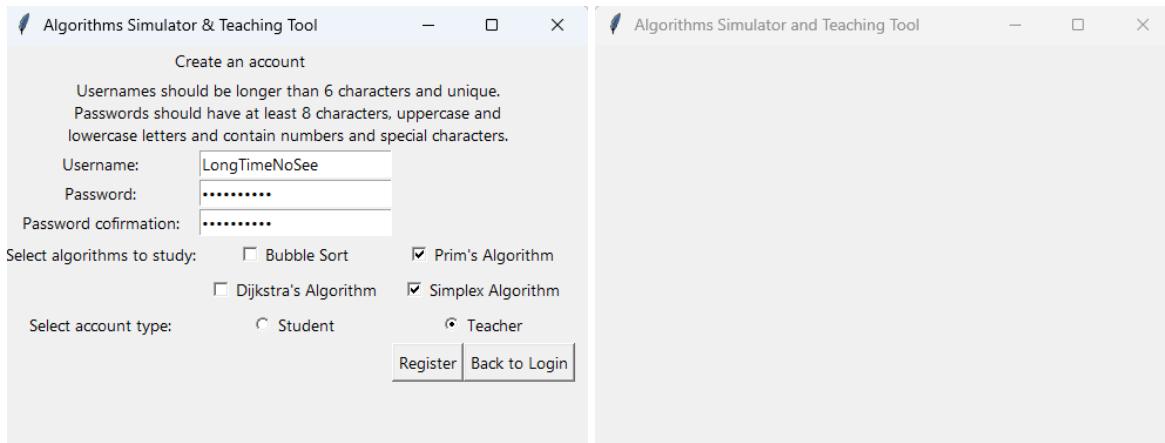
```
>>> %Run NEA_main_file.py
open successful
in new registration page
in register check
in validate username
in validate password
dict_values([<tkinter.IntVar object at 0x000001F9DB04E9B0>, <tkinter.IntVar object at 0x000001F9DB04E950>, <tkinter.IntVar object at 0x000001F9DB04E8F0>, <tkinter.IntVar object at 0x000001F9DB04E890>])
algorithms fails
```

This indicates that the error occurs at the checking of the algorithms selected as I have used the actual values of the Tkinter objects. To fix this, I changed the algorithmValues line to the following:

1.2.r:

```
210     algorithmValues = [var.get() for var in self.algorithmsChosen.values()]
```

This is because I have to get the value of each IntVar variable so that I can use the value in the code. When I now run this code, the following is output when I use the username LongTimeNoSee and password Tere_suvil8:



```
>>> %Run NEA_main_file.py
open successful
in new registration page
in register check
in validate username
in validate password
[0, 1, 0, 1]
inserted into Users
Registration successful
in closeOpen
```

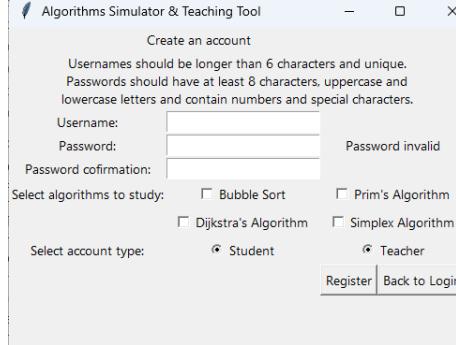
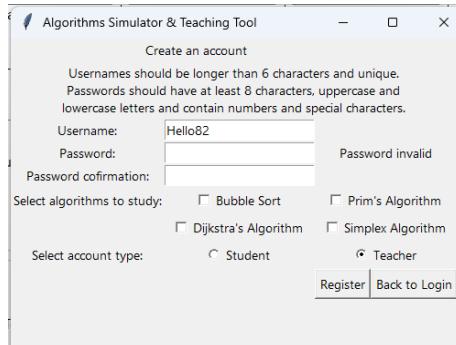
And when I check the DB Browser, it appears as expected:

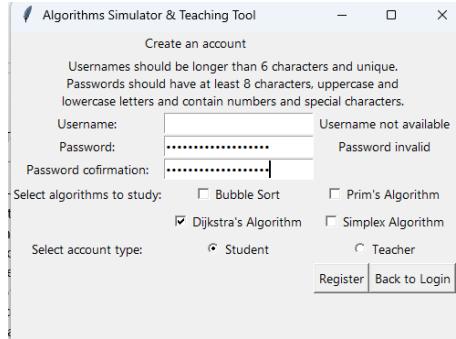
Table: Users			
	Username	Password	
	Account_type		
	Filter	Filter	
1	Test_accS	Testing123!	Student
2	Test_accT	Testing456!	Teacher
3	Billy123	1H087wow!	Student
4	HelloWord!!!	1m@gineThat	Student
5	123abc	cF2E/4.3U	Teacher
6	idkWhatToDo	Mi[s]XcCnj9	Teacher
7	TestHello	Tere_talv7	Student
8	LongTimeNoSee	Tere_suvi8	Teacher

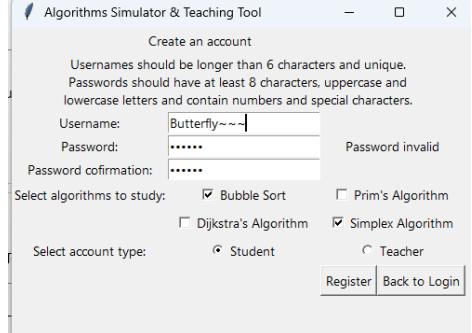
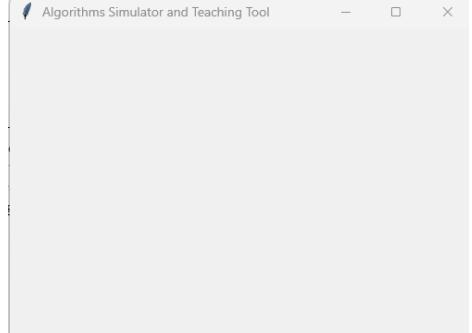
Table: TeacherEnrolment			
	TeacherEnrolID ^{▼1}	Username	Algorithm
	Filter	Filter	Filter
1	1	Test_accT	Prim's
2	2	Test_accT	Dijkstra's
3	3	123abc	Prim's
4	4	123abc	Simplex
5	5	idkWhatToDo	Prim's
6	6	idkWhatToDo	Dijkstra's
7	7	LongTimeNoSee	Prim's
8	8	LongTimeNoSee	Simplex

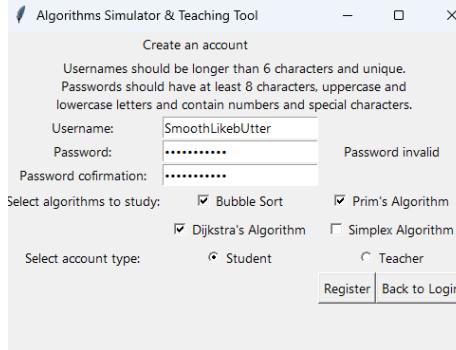
From running this code, it is indicated that the main functionality of checking all the requirements and inserting the values into the database is functioning as expected.

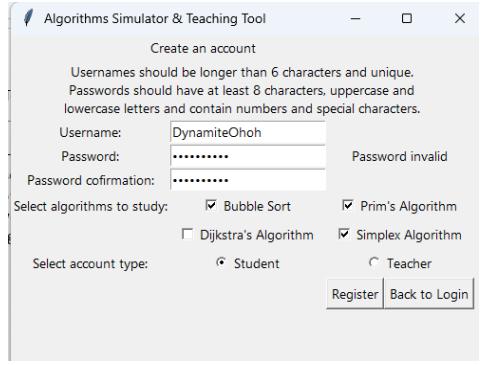
I will now test the code using the white box testing table specified in the REGISTRATION PAGE design section.

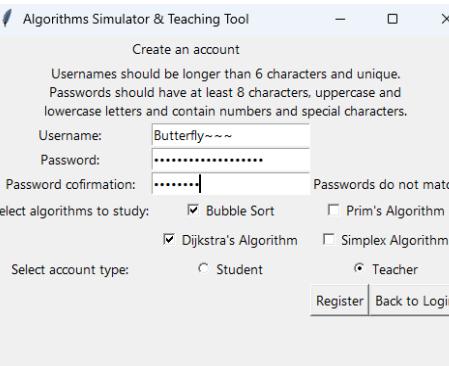
TEST DATA	EXPECTED RESULT	JUSTIFICATION	CODE	ACTUAL OUTPUT
Register without filling in any details.	The user should be prompted with the ‘Unsuccessful registration’ message.	This ensures that the form is filled in fully so that all necessary details are obtained.	1.1.c 1.1.e 1.1.f 1.2.j 1.2.k 1.2.l 1.2.m 1.2.p 1.2.q	 <pre>>>> %Run NEA_main_file.py open successful in new registration page in register check in validate username in validate password</pre> <p>Not as expected but registration is still unsuccessful</p>
Register with a partially filled out form.	Output depends on what has been filled in but should expect for the user to be prompted with the ‘Unsuccessful registration’ message.	This ensures that the form is filled in fully so that all necessary details are obtained.	1.1.c 1.1.e 1.1.f 1.2.j 1.2.k 1.2.l 1.2.m 1.2.p 1.2.q	

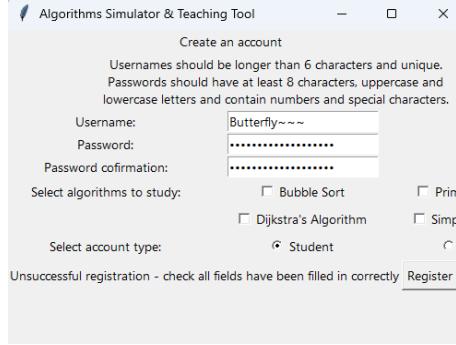
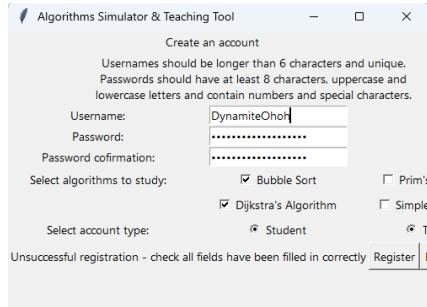
				<pre>>>> %Run NEA_main_file.py open successful in new registration page in register check in validate username in validate password</pre> <p>Not as expected but registration is still unsuccessful</p>
Register with an already used username.	The user should be prompted with the 'Username unavailable' message.	This ensures that the user has entered a unique username that can be used to query the database with.	1.1.c 1.1.e 1.1.f 1.2.j 1.2.k 1.2.l 1.2.m 1.2.p 1.2.q	 <p>The screenshot shows a 'Create an account' dialog box. It includes fields for 'Username' (containing 'testuser') and 'Password' (containing 'password'). Error messages 'Username not available' and 'Password invalid' are displayed next to their respective fields. Below the fields are checkboxes for selecting algorithms: 'Bubble Sort', 'Prim's Algorithm', 'Dijkstra's Algorithm' (which is checked), and 'Simplex Algorithm'. Under 'Select account type', there are radio buttons for 'Student' (which is selected) and 'Teacher'. At the bottom are 'Register' and 'Back to Login' buttons.</p> <pre>>>> %Run NEA_main_file.py open successful in new registration page in register check in validate username</pre> <p>As expected</p>

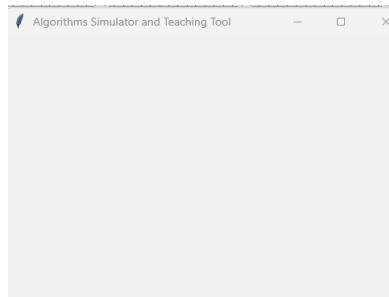
Register with a password that is less than 8 characters.	The user should be prompted with the 'Password invalid' message.	This ensures that the user enters a strong password that meets the requirements.	1.1.c 1.1.e 1.1.f 1.2.j 1.2.k 1.2.l 1.2.m 1.2.p 1.2.q	 <pre data-bbox="1320 601 1635 744">>>> %Run NEA_main_file.py open successful in new registration page in register check in validate username in validate password</pre> <p>As expected but could change to have both password fields cleared</p>
Register with a password that is 8 characters long (boundary test).	The user should be prompted with the 'Password invalid' message.	This ensures that the user enters a strong password that meets the requirements.	1.1.c 1.1.e 1.1.f 1.2.j 1.2.k 1.2.l 1.2.m 1.2.p 1.2.q	

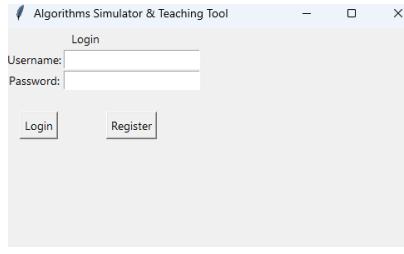
				<pre>>>> %Run NEA_main_file.py open successful in new registration page in register check in validate username in validate password [1, 0, 0, 1] inserted into Users Registration successful in closeOpen</pre> <p>As expected but could change to have both password fields cleared</p>
Register with a password that does not contain both uppercase and lowercase characters.	The user should be prompted with the 'Password invalid' message.	This ensures that the user enters a strong password that meets the requirements.	1.1.c 1.1.e 1.1.f 1.2.j 1.2.k 1.2.l 1.2.m 1.2.p 1.2.q	 <pre>>>> %Run NEA_main_file.py open successful in new registration page in register check in validate username in validate password [1, 0, 0, 1] inserted into Users Registration successful in closeOpen</pre> <p>As expected but could change to have both password fields cleared</p>

Register with a password that does not contain a number.	The user should be prompted with the 'Password invalid' message.	This ensures that the user enters a strong password that meets the requirements.	1.1.c 1.1.e 1.1.f 1.2.j 1.2.k 1.2.l 1.2.m 1.2.p 1.2.q	 <p>The screenshot shows the 'Create an account' window of the 'Algorithms Simulator & Teaching Tool'. The 'Username' field contains 'DynamiteOhoh', the 'Password' field contains '.....', and the 'Password confirmation' field also contains '.....'. A red error message 'Password invalid' is displayed next to the password fields. Below the fields, there are checkboxes for selecting algorithms to study: 'Bubble Sort' (checked), 'Prim's Algorithm' (checked), 'Dijkstra's Algorithm' (unchecked), and 'Simplex Algorithm' (checked). There are also radio buttons for 'Select account type': 'Student' (checked) and 'Teacher' (unchecked). At the bottom are 'Register' and 'Back to Login' buttons.</p>
Register with a password that does not contain a special character.	The user should be prompted with the 'Password invalid' message.	This ensures that the user enters a strong password that meets the requirements.	1.1.c 1.1.e 1.1.f 1.2.j 1.2.k 1.2.l 1.2.m 1.2.p 1.2.q	

				<pre>>>> %Run NEA_main_file.py open successful in new registration page in register check in validate username in validate password</pre> <p style="text-align: right;">As expected but could change to have both password fields cleared</p>
Register with the password and password confirmation fields not matching.	The user should be prompted with the 'Passwords do not match' message.	This ensures that the user must complete these fields correctly so they match and the user is sure of their password.	1.1.c 1.1.e 1.1.f 1.2.j 1.2.k 1.2.l 1.2.m 1.2.p 1.2.q	 <pre>>>> %Run NEA_main_file.py open successful in new registration page in register check in validate username in validate password</pre> <p style="text-align: right;">As expected but could change to have both password fields cleared</p>

Register without selecting an algorithm to study.	The user should be prompted with the ‘Unsuccessful registration’ message.	This ensures that at least one algorithm must be chosen.	1.1.c 1.1.e 1.1.f 1.2.j 1.2.k 1.2.l 1.2.m 1.2.p 1.2.q	 <pre data-bbox="1320 601 1641 807">>>> %Run NEA_main_file.py open successful in new registration page in register check in validate username in validate password [0, 0, 0, 0] algorithms fails</pre> <p data-bbox="1680 791 1837 823">As expected</p>
Register without choosing an account type.	The user should be prompted with the ‘Unsuccessful registration’ message.	This ensures that an account type must be selected.	1.1.c 1.1.e 1.1.f 1.2.j 1.2.k 1.2.l 1.2.m 1.2.p 1.2.q	

				<pre>>>> %Run NEA_main_file.py open successful in new registration page in register check in validate username in validate password [1, 0, 1, 0] account fails</pre> <p style="text-align: right;">As expected</p>
Register with fully correctly filled in details.	Registration should be successful so the registration window should be closed and the home page window should be opened.	This ensures that the user must enter fully correct details and cannot register otherwise.	1.1.c 1.1.e 1.1.f 1.2.j 1.2.k 1.2.l 1.2.m 1.2.p 1.2.q 1.2.g	 <pre>>>> %Run NEA_main_file.py open successful in new registration page in register check in validate username in validate password [1, 0, 1, 0]</pre> <pre>>>> %Run NEA_main_file.py open successful in new registration page in register check in validate username in validate password [1, 0, 1, 0] inserted into Users inserted into StudentEnrolment Registration successful in closeOpen</pre> <p style="text-align: right;">As expected</p>

Press the 'Back to Login' button.	Should redirect the user to the Login page.	This ensures the 'Back to Login' button works correctly.	1.1.f 1.1.d	 <pre>>>> %Run NEA_main_file.py open successful in new registration page in new login page</pre>
-----------------------------------	---	--	----------------	---

The two tests that resulted in an output that was not expected were the ones involving not all the fields being filled in. To fix this, I used code used in the validateLoginInput method where there was a check for if a value had been entered to fix this. See the new code below:

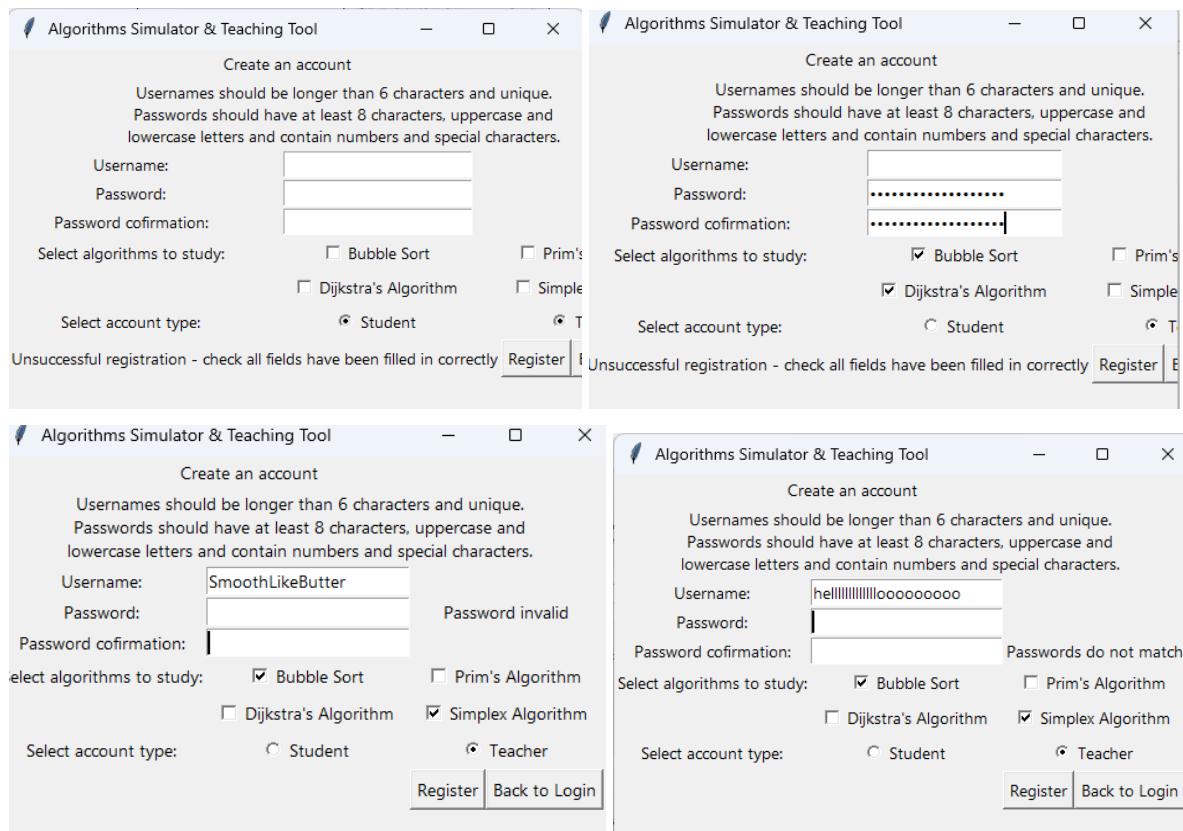
1.2.s:

```

203     if not usernameHolder or not passwordHolder or not confirmPasswordHolder:
204         self.invalidMessageReg["text"] = "Unsuccessful registration - check all
205         return
206
207     if not self.validateUsernameReg():
208         self.usernameInvalid["text"] = "Username not available"
209         self.newUsername.set("")
210         return
211
212     # validate password
213     if not self.validatePasswordReg():
214         self.passwordInvalid["text"] = "Password invalid"
215         self.newPassword.set("")
216         self.newConfirmPassword.set("")
217         return
218     if passwordHolder != confirmPasswordHolder:
219         self.passwordsMatchInvalid["text"] = "Passwords do not match"
220         self.newPassword.set("")
221         self.newConfirmPassword.set("")
222         return

```

Now when I test the code without completing the fields, partially completing the fields, inputting an incorrect password and inputting passwords that do not match, the following are output:



These outputs are all as expected so the functionality for the registration section is fully complete.

ITERATION 2 - DEVELOPER REVIEW

In this iteration, I encountered more issues than in the previous one as I focused on the functionality of validating user inputs. One issue was the infinite loop when validating the details. I overcame these issues by putting print statements to help me track the flow of the program, which helped me identify where the issues came from. Overall, all issues were resolved.

ITERATION 3 - REFINING THE GUI

As the functionality has been fully completed, this iteration of developing the Initial Login and Registration pages prototype will be focused on the system's GUI. I will be focused on refining the GUI so that it appears more user friendly and has the elements shown in the design section for the Login and Registration GUIs.

Firstly, I created variables for 3 fonts:

1.3.a

```
40 # fonts:  
41 fontLarge = ("Calibri", 16, "bold")  
42 fontMedium = ("Calibri", 12)  
43 fontSmall = ("Calibri", 10)
```

And then I focused on the colours and fonts of each widget. See below the changes made:

1.3.b

```

94
95
96 def loginRegisterWidgets(self):
97     self.title = Label(self.master, text="Login", font=fontLarge, fg="#1b263b", bg="#eaebed")
98     self.title.grid(row=0, column=0, columnspan=5)
99
100    # login frame widgets
101    self.loginFrame = Frame(self.master, bg="#eaebed")
102    Label(self.loginFrame, text="Username:", font=fontMedium, fg="#1b263b", bg="#eaebed").grid(row=2, column=0, columnspan=2)
103    Label(self.loginFrame, text="Password:", font=fontMedium, fg="#1b263b", bg="#eaebed").grid(row=3, column=0, columnspan=2)
104    Entry(self.loginFrame, textvariable=self.username).grid(row=2, column=2, columnspan=3)
105    Entry(self.loginFrame, textvariable=self.password, show="*").grid(row=3, column=2, columnspan=3)
106    self.loginInvalid = Label(self.loginFrame, text="", font=fontSmall, fg="#990000") # unsuccessful login text
107    self.loginInvalid.grid(row=5, column=0, columnspan=5)
108    Button(self.loginFrame, text="Login", command=self.loginCheck, activebackground="white", activeforeground="#1b263b", bg="#1b263b", fg="white", font=fontMedium).grid(row=6, column=0, columnspan=5)
109    Button(self.loginFrame, text="Register", command=self.newRegistrationPage, activebackground="white", activeforeground="#1b263b", bg="#1b263b", fg="white", font=fontMedium).grid(row=6, column=4, columnspan=2)

```

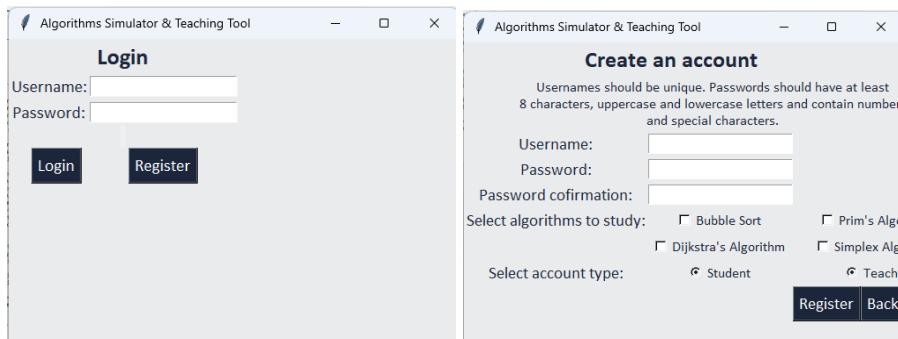
1.3.c

```

113
114    # register frame widgets
115    self.regFrame = Frame(self.master, bg="#eaebed")
116    Label(self.regFrame, text="Names should be unique. Passwords should have at least\n8 characters, uppercase and lowercase letters and contain numbers\nand special characters.", font=fontMedium, fg="#1b263b", bg="#eaebed").grid(row=2, column=0, columnspan=8)
117    Label(self.regFrame, text="Username:", font=fontMedium, fg="#1b263b", bg="#eaebed").grid(row=3, column=0, columnspan=2)
118    Label(self.regFrame, text="Password confirmation:", font=fontMedium, fg="#1b263b", bg="#eaebed").grid(row=4, column=0, columnspan=2)
119    Label(self.regFrame, text="Select algorithms to study:", font=fontMedium, fg="#1b263b", bg="#eaebed").grid(row=5, column=0, columnspan=2)
120    Label(self.regFrame, text="Select account type:", font=fontMedium, fg="#1b263b", bg="#eaebed").grid(row=7, column=0, columnspan=2)
121    # entry fields:
122    Entry(self.regFrame, textvariable=self.newUsername).grid(row=2, column=2, columnspan=2)
123    Entry(self.regFrame, textvariable=self.newPassword, show="*").grid(row=3, column=2, columnspan=2)
124    Entry(self.regFrame, textvariable=self.newConfirmPassword, show="*").grid(row=4, column=2, columnspan=2)
125    # invalid messages:
126    self.usernameInvalid = Label(self.regFrame, text="", font=fontSmall, fg="#990000", bg="#eaebed")
127    self.usernameInvalid.grid(row=2, column=4, columnspan=2)
128    self.passwordInvalid = Label(self.regFrame, text="", font=fontSmall, fg="#990000", bg="#eaebed")
129    self.passwordInvalid.grid(row=3, column=4, columnspan=2)
130    self.passwordsMatchInvalid = Label(self.regFrame, text="", font=fontSmall, fg="#990000", bg="#eaebed")
131    self.passwordsMatchInvalid.grid(row=4, column=4, columnspan=2)
132    self.invalidMessageReg = Label(self.regFrame, text="", font=fontSmall, fg="#990000", bg="#eaebed")
133    self.invalidMessageReg.grid(row=8, column=0, columnspan=4)
134    # algorithm checkboxes:
135    Checkbutton(self.regFrame, text="Bubble Sort", variable=self.bubbleSortSelect, onvalue=1, offvalue=0, font=fontSmall, fg="#1b263b", bg="#eaebed").grid(row=5, column=2, columnspan=2)
136    Checkbutton(self.regFrame, text="Prim's Algorithm", variable=self.primSelect, onvalue=1, offvalue=0, font=fontSmall, fg="#1b263b", bg="#eaebed").grid(row=5, column=4, columnspan=2)
137    Checkbutton(self.regFrame, text="Dijkstra's Algorithm", variable=self.dijkstraSelect, onvalue=1, offvalue=0, font=fontSmall, fg="#1b263b", bg="#eaebed").grid(row=6, column=2, columnspan=2)
138    Checkbutton(self.regFrame, text="Simplex Algorithm", variable=self.simplexSelect, onvalue=1, offvalue=0, font=fontSmall, fg="#1b263b", bg="#eaebed").grid(row=6, column=4, columnspan=2)
139    # account selection radiobutton:
140    Radiobutton(self.regFrame, text="Student", variable=self.accountType, value="Student", font=fontSmall, fg="#1b263b", bg="#eaebed").grid(row=7, column=2, columnspan=2)
141    Radiobutton(self.regFrame, text="Teacher", variable=self.accountType, value="Teacher", font=fontSmall, fg="#1b263b", bg="#eaebed").grid(row=7, column=4, columnspan=2)
142    # buttons
143    Button(self.regFrame, text="Register", command=self.registerCheck, activebackground="white", activeforeground="#1b263b", bg="#1b263b", fg="white", font=fontMedium).grid(row=8, column=2, columnspan=2)
144    Button(self.regFrame, text="Back to Login", command=self.newLoginPage, activebackground="white", activeforeground="#1b263b", bg="#1b263b", fg="white", font=fontMedium).grid(row=8, column=4, columnspan=2)

```

When this code is run, the following is output:



This was looking much more like the designs in the GUI section. However, the geometry for the window was such that the whole of the registration frame could not fit in it. Therefore, I changed the size of the window.

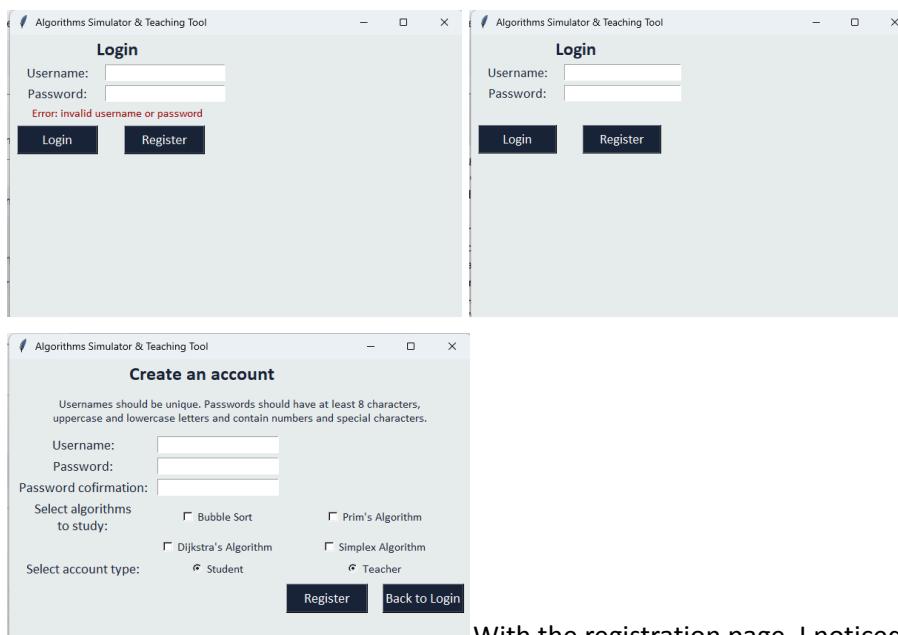
I also noticed that when I pressed the back to login button on the registration page, the invalid text did not clear. So to change this, I added the following lines of code to the sections of code where login or registration is validated and moves onto the home page and to the newRegistrationPage and newLoginPage methods:

1.3.d

```
self.newUsername.set("")
self.newPassword.set("")
self.newConfirmPassword.set("")
for value in self.algorithmsChosen.values():
    value.set(0)
self.accountType.set("")
self.usernameInvalid["text"] = ""
self.passwordInvalid["text"] = ""
self.passwordsMatchInvalid["text"] = ""
self.invalidMessageReg["text"] = ""

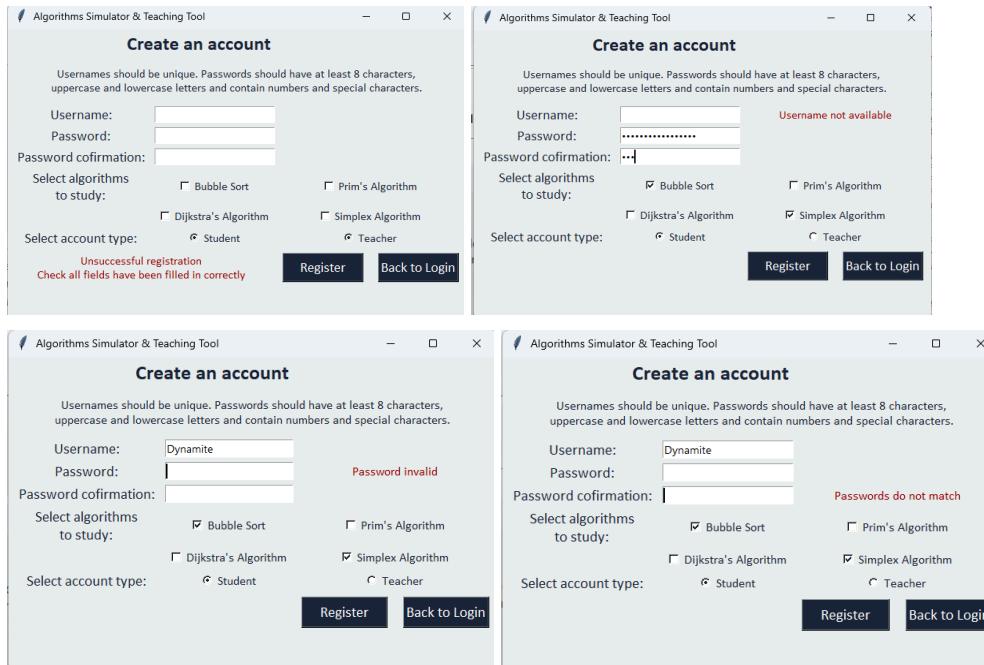
self.loginInvalid["text"] = ""
self.username.set("")
self.password.set("")
```

Then, I added some padding to the buttons and some of the text. Below examples of what the screen looks like after these changes:



With the registration page, I noticed that messages were not clearing if a test failed previously but now does not. Therefore, I added else statements to clear the

message that would otherwise have the invalid input message. This worked as expected so here are some screenshots of the output on the screen with different text that would error to test each stage:



ITERATION 3 - DEVELOPER REVIEW

I had no experience prior to this system with Tkinter GUIs so it took some time to familiarise myself with it. However, no issues were encountered. Development is now complete for PROTOTYPE 1 as all the usability and functionality features are complete. Therefore, I have now commented out all print messages from the code, which I had put in to help me check that the flow of the program was correct.

PROTOTYPE 1 - STAKEHOLDER REVIEW

Now that Prototype 1 has been fully developed and tested, I had a stakeholder review the progress. I asked about the Usability - how user friendly the system is - the Functionality - does it function as intended? - and Performance - how fast the system performs.

See below the feedback given by one of my Stakeholders:

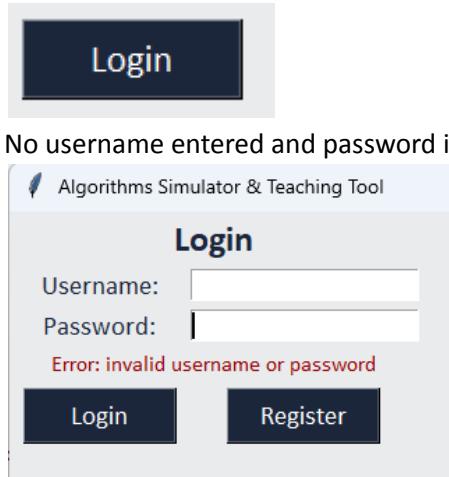
AREA OF CONCERN	FEEDBACK
Usability	Quite user friendly but there could be a button to show the password when typing it out.
Functionality	Everything works well.
Performance	Good performance and processing speed.

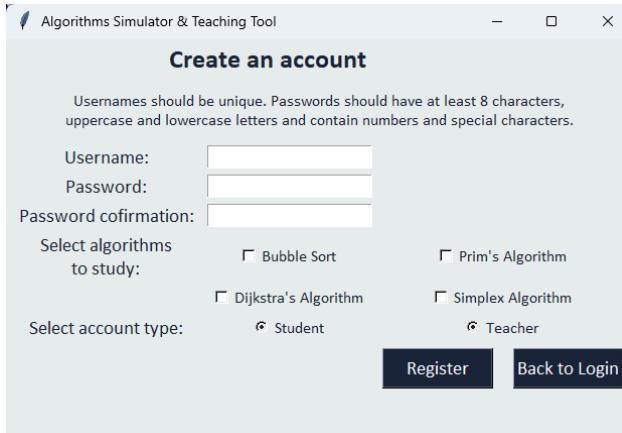
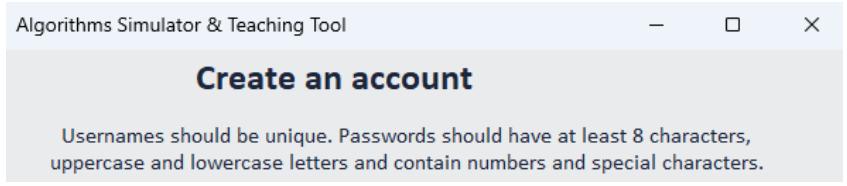
From the feedback given, the system generally works well and is user friendly. However, functionality to allow the user to view their password when they have typed it into the password or password confirmation fields could be included. I think that this is not something to be included right now and so may be something added in post-development.

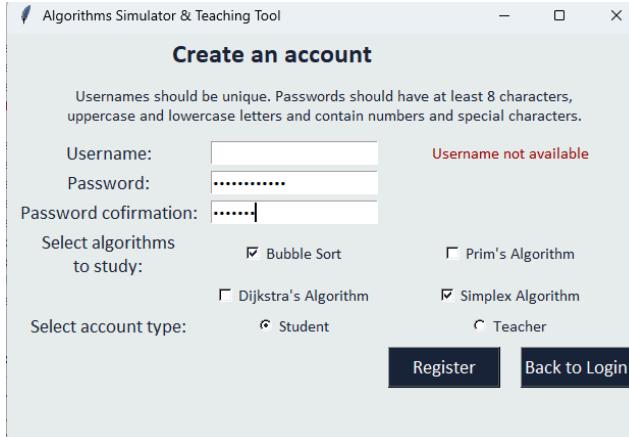
PROTOTYPE 1 - SUCCESS CRITERIA REVIEW

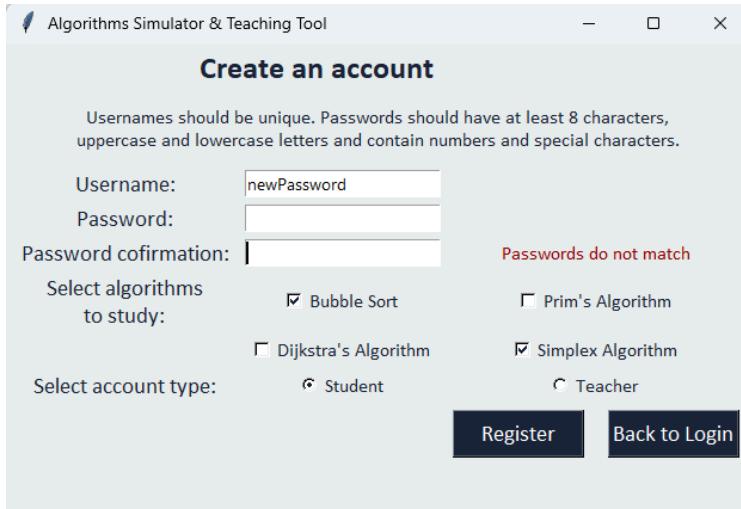
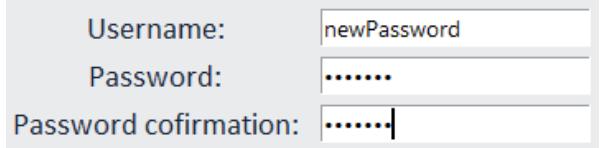
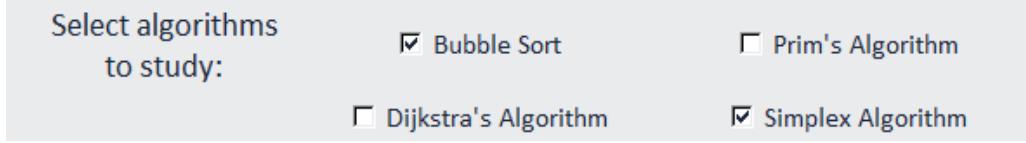
In order to check that all the success criteria outlined in the ANALYSIS section for the Initial Login and Registration pages have been met, below is a success criteria review for this prototype:

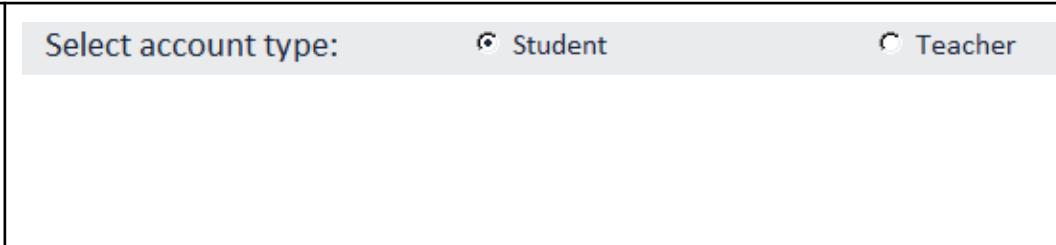
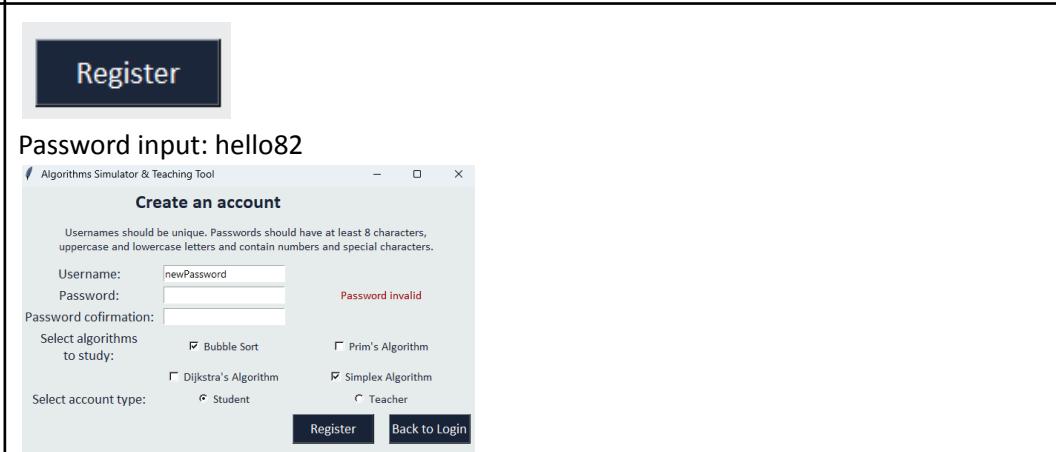
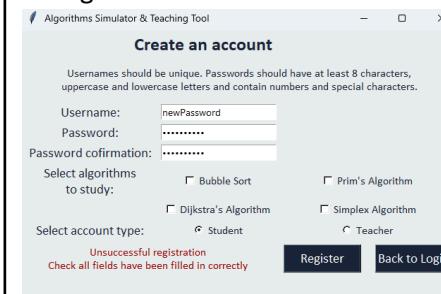
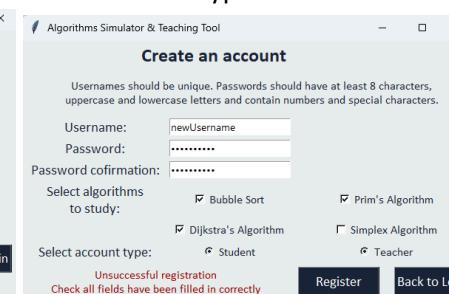
SUCCESS CRITERIA	OUTPUT & EVIDENCE	CODE	COMPLETE
[1][a] There will be text greeting the user with the title of the system 'Algorithms Simulator and Teaching Tool'. I will test this by checking the page and making sure this appears.		1.1.d 1.1.g 1.3.b	Y
[1][b] There will be two data input fields - Username and Password - that should appear below the title. I will test this by checking that they appear on the page and that text can be entered into them.		1.1.c 1.1.d 1.3.b	Y
[1][c] When data is entered into the Password field, black dots appear as placeholders. I will test these by entering different passwords into the Password field to make sure placeholders appear instead of the	 Input: HelloWorld34% Output: 	1.1.c 1.1.d 1.3.b	Y

text.			
[1][d] There will be a Sign In button below the two input fields. When it is pressed, the database is queried and the values are compared to make sure they are correct for the user to log in and continue to the Home Page. If incorrect details were entered, a message will pop up saying 'Login was unsuccessful. Check the username/password entered are correct. If you have not registered before, click the Register button.'. I will test this by entering correct and incorrect usernames and passwords and checking that the functionality of clicking the Sign In button is correct for all cases.	 <p>No username entered and password incorrect:</p> <p>Algorithms Simulator & Teaching Tool</p> <p>Login</p> <p>Username: <input type="text"/></p> <p>Password: <input type="password"/></p> <p>Error: invalid username or password</p> <p>Login Register</p>	1.1.c 1.1.d 1.1.f 1.2.a 1.2.b 1.2.d 1.2.i 1.3.b	Message has not been changed but otherwise Y
[1][e] There will be a Register button next to the Sign In button. When it is clicked, the user will be sent to the		1.1.f 1.1.d 1.1.e	Y

Registration page. I will test this by clicking the Register button and checking that this occurs.	When clicked: 	1.3.b 1.3.c	
[2][a] There will be text greeting the user saying 'Create an Account' and the following requirements: 'Usernames should be unique.' and 'Passwords should have at least 8 characters, uppercase and lowercase letters and contain numbers and special characters.' I will test this by checking that when the Registration page is accessed, this text is displayed.		1.1.e 1.3.c	Y
[2][b] A Username input field, Password input field and Password Confirmation input field appear below each other and below the text. I will		1.1.c 1.1.e 1.3.c	Y

test this by checking that when the Registration page is accessed, these fields are displayed and text can be entered into them.			
[2][c] When data is entered into the Username input field, text will say 'Username not available' if the entered username has already been used. I will test this by having premade accounts and checking that when I enter their usernames, this message displays and when I enter a unique username, it does not.		1.1.c 1.1.e 1.1.f 1.2.l 1.2.m 1.2.n 1.3.c	Y
[2][d] When data is entered into the Password Confirmation input field, text will display whether it matches the data in the Password field. I will test this by entering matching and different passwords into the fields and checking that only text is output if the passwords do not match.	Password: Hello82! Password confirmation: wow	1.1.c 1.1.e 1.1.f 1.2.l 1.2.m 1.2.n 1.3.c	Y

			
[2][e] The Password and Password Confirmation fields have black dots as placeholders. I will test this by entering different passwords into both fields to make sure placeholders appear instead of the text.		1.1.e 1.3.c	Y
[2][f] Check boxes will appear for the four different algorithms implemented. I will test this by checking that when the Registration page is accessed, these appear and algorithms can be checked.		1.1.c 1.1.e 1.2.k 1.3.c	Y

<p>[2][g] A student/teacher selection will appear for the type of account being created. I will test this by checking that when the Registration page is accessed, this appears and only one of these options can be chosen.</p>		1.1.c 1.1.e 1.3.c	Y
<p>[2][h] A Register button will appear below the student/teacher account choice. When pressed, the Username, Password and Password Confirmation fields should be validated against the rules detailed in criteria [2][a] and for the two password entries matching. Additionally, validation that at least one algorithm has been checked and that a type of account has been chosen. If all these checks have passed successfully, all of this data should be written to a database and continue to the Home page. If any of the validation checks is unsuccessful, a message will pop up saying ‘Registration unsuccessful. Check that username/password entries are valid and that at least one algorithm has been checked and a type of account has been selected.’ I will test this by checking that when the Registration page is reached, the Register button</p>	  <p>No algorithms selected:</p>  <p>No account type selected:</p> 	1.1.c 1.1.e 1.1.f 1.2.l 1.2.m 1.2.p 1.2.q 1.2.r 1.2.s 1.3.c	Y

appears. For testing functionality, I will enter details that fully/partially/don't meet the criteria and try pressing the Register button, checking that the functionality is correct.			
<p>[2][i] A Back button will appear at the bottom right corner of the Registration page. When clicked, this returns the user to the Initial Login page. I will test this by making sure that when the Registration page is reached, this button appears in the correct place and that pressing it successfully returns the user to the Initial Login page.</p>	<p>When pressed:</p>  <p>The screenshot shows a dark blue rectangular button with the white text "Back to Login". Below this, a larger window titled "Login" is displayed. It contains two input fields labeled "Username:" and "Password:", each with a small placeholder text "Placeholder". At the bottom of the window are two buttons: "Login" on the left and "Register" on the right, both in white text on dark backgrounds.</p>	1.1.d 1.1.e 1.1.f 1.3.b 1.3.d	Y

All success criteria have been met so development of this prototype is fully complete, all white box testing has been completed and all success criteria have been met.

PROTOTYPE 2 - HOME AND BUBBLE SORT VISUALISATION PAGES AND MENU

In this prototype, I will fully complete the functionality of the Home page, the Bubble Sort visualisation page and the Menu. For the Home page, this will require setting up the Visualisation pages for each of the 4 algorithm visualisations and the quiz page.

ITERATION 1 - HOME PAGE AND SETTING UP VISUALISATION AND QUIZ PAGES

In this iteration, I will fully complete the functionality and the fully refined GUI for the home page because the Home page is more sparse than the other pages. This will require setting up the other Visualisation pages and the Quiz page.

First, I added the following code to the openClose function:

2.1.a

```

48     elif newType == "bubble sort":
49         bubbleSortRoot = Tk()
50         bubbleSortRoot.geometry('500x350+300+300') # to change
51         bubbleSortRoot.title("Bubble Sort Visualisation")
52         running = BubbleSort(bubbleSortRoot)
53         bubbleSortRoot.mainloop()
54
55     elif newType == "prim":
56         primRoot = Tk()
57         primRoot.geometry('500x350+300+300') # to change
58         primRoot.title("Prim's Algorithm Visualisation")
59         running = Prim(primRoot)
60         primRoot.mainloop()
61
62     elif newType == "dijkstra":
63         dijkstraRoot = Tk()
64         dijkstraRoot.geometry('500x350+300+300') # to change
65         dijkstraRoot.title("Dijkstra's Algorithm Visualisation")
66         running = Dijkstra(dijkstraRoot)
67         dijkstraRoot.mainloop()
68
69     elif newType == "simplex":
70         simplexRoot = Tk()
71         simplexRoot.geometry('500x350+300+300') # to change
72         simplexRoot.title("Simplex Algorithm Visualisation")
73         running = Simplex(simplexRoot)
74         simplexRoot.mainloop()
75
76     elif newType == "quiz":
77         quizRoot = Tk()
78         quizRoot.geometry('500x350+300+300') # to change
79         quizRoot.title("Quiz")
80         running = Quiz(quizRoot)
81         quizRoot.mainloop()
```

And I created the following basic structures for the Home, BubbleSort, Prim, Dijkstra, Simplex and Quiz classes:

2.1.b

```

400 class HomeMain(Main):
401     def __init__(self, master):
402         super().__init__(master)
403
404         self.homeFrame = None
405
406
407     def homeWidgets(self):
408
409
410     def openBubbleSort(self):
411         closeOpen(self.master, "bubble sort")
412
413     def openPrim(self):
414         closeOpen(self.master, "prim")
415
416     def openDijkstra(self):
417         closeOpen(self.master, "dijkstra")
418
419     def openSimplex(self):
420         closeOpen(self.master, "simplex")
421
422     def openQuiz(self):
423         closeOpen(self.master, "quiz")
424
425     def logout(self):
426         closeOpen(self.master, "login")

```

2.1.c

```

429 class BubbleSort(HomeMain):
430     def __init__(self):
431         super().__init__(master)
432
433     def bubbleSortWidgets(self):
434
435
436     def openHome(self):
437         closeOpen(self.master, "home")
438
439
440 class Prim(HomeMain):
441     def __init__(self):
442         super().__init__(master)
443
444     def primWidgets(self):
445
446     def openHome(self):
447         closeOpen(self.master, "home")

```

2.1.e

```

470 class Quiz(HomeMain):
471     def __init__(self):
472         super().__init__(master)
473         # inherits algorithms chosen???
474
475     def quizWidgets(self):
476
477     def openHome(self):
478         closeOpen(self.master, "home")

```

2.1.d

```

450 class Dijkstra(HomeMain):
451     def __init__(self):
452         super().__init__(master)
453
454
455     def dijkstraWidgets(self):
456
457
458     def openHome(self):
459         closeOpen(self.master, "home")
460
461
462 class Simplex(HomeMain):
463     def __init__(self):
464         super().__init__(master)
465
466     def simplexWidgets(self):
467
468     def openHome(self):
469         closeOpen(self.master, "home")

```

I did this because having separate subclasses makes the code for each algorithm easier to manage and because the pages will all be different sizes so this would make it easier to keep track of the geometry for each page.

As this iteration focuses on the Home page, I then concentrated on the GUI and the Tkinter widgets for this page. Below is the code I wrote for the homeWidgets method:

2.1.f

```

427 def homeWidgets(self):
428     self.title = Label(self.master, text="Algorithm's Simulator and Teaching Tool")
429     self.title.grid(row=0, column=0, columnspan=5)
430
431     self.homeFrame = Frame(self.master)
432
433     buttonLocations = self.locations()
434     # the index of each item in the list is the column of it in the gui
435     if "Bubble Sort" in buttonLocations:
436         Button(self.homeFrame, text="Bubble Sort", command=self.openBubbleSort).grid(row=1, column=buttonLocations.index("Bubble Sort")) #bubble sort button
437     if "Prim's" in buttonLocations:
438         Button(self.homeFrame, text="Prim's Algorithm", command=self.openPrim).grid(row=1, column=buttonLocations.index("Prim's")) # prim's button
439     if "Dijkstra's" in buttonLocations:
440         Button(self.homeFrame, text="Dijkstra's Algorithm", command=self.openDijkstra).grid(row=1, column=buttonLocations.index("Dijkstra's")) # dijkstra button
441     if "Simplex" in buttonLocations:
442         Button(self.homeFrame, text="Simplex Algorithm", command=self.openSimplex).grid(row=1, column=buttonLocations.index("Simplex")) # simplex button
443     if "Quiz" in buttonLocations:
444         Button(self.homeFrame, text="Quiz", command=self.openQuiz).grid(row=1, column=buttonLocations.index("Quiz")) # quiz button
445
446     Button(self.homeFrame, text="Logout", command=self.logout).grid(row=5, column=4) # logout button
447
448     self.homeFrame.grid(row=1, column=0, columnspan=5)

```

This uses the locations method, which I wrote to be able to determine the columns of the buttons as users have a choice of algorithms. Below is the code for this:

2.1.g

```

451 def locations(self):
452     buttonLocations = []
453     # key = name of algorithm, value=1 or 0 (depending on if chosen or not)
454     for key, value in self.algorithmsChosen.items():
455         if value == 1:
456             buttonLocations.append(key)
457     if self.accountType == "Student":
458         buttonLocations.append("Quiz")
459
460     return buttonLocations

```

As seen in this code, I used the accountType and algorithmsChosen variables. Before, I had these as variables in the LoginRegister class but I moved these attributes (along with their associated attributes) to the Main class, which means that I would be able to use them as the Home class would inherit these attributes.

Then I focused on the code for the quiz statistics section of the home page. As I had decided to use the matplotlib library for plotting this bar chart, which was due to the styling benefits that matplotlib has over Tkinter, I had to research how the plotting²⁴ of bar charts works in matplotlib and how to embed²⁵ this in a Tkinter window. From this research, I identified that having lists for the x axis labels and the plotted data is the best way to approach this problem. Therefore, I created a method to get these values and attributes for them in the constructor method:

2.1.h

```
415 class HomeMain(Main):
416     def __init__(self, master):
417         super().__init__(master)
418         self.homeFrame = None
419         # holds algorithm names to go as labels on the x axis
420         self.algorithmNames = self.algorithmAxisValues("Name")
421         # hold the correct and incorrect scores to be plot as bars on the bar chart
422         self.correctScores = self.algorithmAxisValues("Correct")
423         self.incorrectScores = self.algorithmAxisValues("Incorrect")
424         self.homeWidgets()
425         self.plotStatistics()
```

2.1.i

```
461 def algorithmAxisValues(self, listType):
462     valuesList = []
463     try:
464         self.cursor.execute("SELECT Algorithm, CorrectScore, IncorrectScore FROM StudentEnrolment WHERE Username = ?", (self.currentAccUsername,))
465         results = self.cursor.fetchall()
466         if listType == "Name":
467             for row in results:
468                 valuesList.append(row[0])
469                 valuesList.append("Total")
470         elif listType == "Correct":
471             totalCorrect = 0
472             for row in results:
473                 valuesList.append(row[1])
474                 totalCorrect += row[1]
475             valuesList.append(totalCorrect)
476         elif listType == "Incorrect":
477             totalIncorrect = 0
478             for row in results:
479                 valuesList.append(row[2])
480                 totalIncorrect += row[2]
481             valuesList.append(totalIncorrect)
482     except sqlite3.Error as e:
483         print("Database error: {e}")
484
485     return valuesList
```

As I realised I would need to be able to use the username entered in order to perform SQL queries, I also created a new attribute in the Main class to hold the current accounts username. For this, I edited the loginCheck and registerCheck methods in the LoginRegister class to add the following line before clearing the fields:

2.1.j

```

342     # store entered username for sql queries:
343     self.currentAccUsername = self.newUsername.get()
344     # clear inputs after successful registration:
345     self.newUsername.set("")
346     self.newPassword.set("")
347     self.newConfirmPassword.set("")
348     for value in self.algorithmsChosen.values():
349         value.set(0)
350         self.accountType.set("")
351         self.usernameInvalid["text"] = ""
352         self.passwordInvalid["text"] = ""
353         self.passwordsMatchInvalid["text"] = ""
354         self.invalidMessageReg["text"] = ""
355         closeOpen(self.master, "home")

```

2.1.k

```

209     if self.validateLoginInput(): # is True
210         #print("login validated")
211         # store current username for sql queries:
212         self.currentAccUsername = self.username.get()
213         # clear input fields:
214         self.loginInvalid["text"] = ""
215         self.username.set("")
216         self.password.set("")
217         # open home page:
218         closeOpen(self.master, "home")

```

Additionally, see below the code for plotting the quiz statistics:

2.1.l

```

fig, ax = plt.subplots(figsize=(8,4))
x = range(len(self.algorithmNames))

# plot the bars
ax.bar(x, self.correctScores, width=0.3, label="Correct", align="center")
ax.bar([p + 0.3 for p in x], self.incorrectScores, width=0.3, label = "Incorrect", align = "center")

# labels, title, legend
ax.set_xlabel("Algorithms")
ax.set_ylabel("Score")
ax.set_title("Quiz Statistics")
ax.set_xticks([p + 0.2 for p in x])
ax.set_xticklabels(self.algorithmNames)
ax.legend()

# embed into Tkinter
canvas = FigureCanvasTkAgg(fig, self.master)
canvasWidget = canvas.get_tk_widget()
canvasWidget.grid(row=2, column=1, columnspan=3)

```

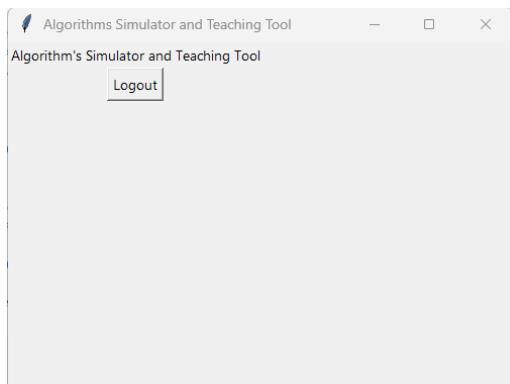
To test the code, I will be using the HelloWord!!! account that I had previously created in DB Browser. This is because it has values for the different algorithms already added so I am able to test the quiz statistics chart better. When I run this code (and have pressed the Login button), I get the following error:

```

>>> %Run NEA_main_file.py
in closeOpen
Database error: {e}
Exception in Tkinter callback
Traceback (most recent call last):
  File "C:\Users\ingri\AppData\Local\Programs\Thonny\lib\tkinter\_init_.py", line 1921, in _call
    return self._func(*args)
  File "C:\Users\ingri\OneDrive\Documents\QM\QM School work\A Level\Computer Science\NEA\NEA main file.py", line 218, in closeOpen
    self.master, "home")
  File "C:\Users\ingri\OneDrive\Documents\QM\QM School work\A Level\Computer Science\NEA\NEA main file.py", line 41, in running
    HomeMain(homeRoot)
  File "C:\Users\ingri\OneDrive\Documents\QM\QM School work\A Level\Computer Science\NEA\NEA main file.py", line 422, in self.correctScores = self.algoirthmAxisValues("Correct")
AttributeError: 'HomeMain' object has no attribute 'algoirthmAxisValues'

```

After trying to fix this by making the attributes None in the constructor method, changing them in the algorithmAxisValues method and calling the method in the constructor method, I get the following output:



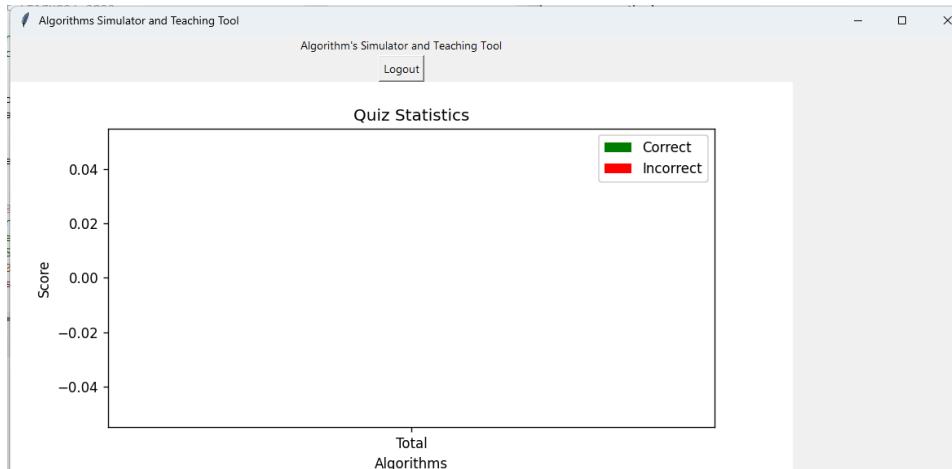
```
>>> %Run NEA_main_file.py
in closeOpen
Exception in Tkinter callback
Traceback (most recent call last):
  File "C:\Users\ingri\AppData\Local\Programs\Thonny\lib\tkinter\__init__.py", line 1921, in __call__
    return self.func(*args)
  File "C:\Users\ingri\OneDrive\Documents\QM\QM School work\A Level\Computer Science\NEA\NEA main file.py", line 218, in loginCheck
    closeOpen(self.master, "home")
  File "C:\Users\ingri\OneDrive\Documents\QM\QM School work\A Level\Computer Science\NEA\NEA main file.py", line 41, in closeOpen
    running = HomeMain(homeRoot)
  File "C:\Users\ingri\OneDrive\Documents\QM\QM School work\A Level\Computer Science\NEA\NEA main file.py", line 429, in __init__
    self.plotStatistics()
  File "C:\Users\ingri\OneDrive\Documents\QM\QM School work\A Level\Computer Science\NEA\NEA main file.py", line 500, in plotStatistics
    x = range(len(self.algorithmNames))
TypeError: object of type 'NoneType' has no len()
```

I fixed this error by fixing my typos and going back to the original code (calling the method for each attribute). Additionally, for debugging purposes, I added the following lines in the plotStatistics method:

```
500      print(f"Correct Scores: {self.correctScores}")
501      print(f"Incorrect Scores: {self.incorrectScores}")
```

When I now run the code, I get the following outputs (I resized the window to see the full bar chart):

```
>>> %Run NEA_main_file.py
in closeOpen
Correct Scores: [0]
Incorrect Scores: [0]
```



This output is not as expected as these are the values I had input for the HelloWorld!!! database entry and the buttons for the algorithms are not appearing as intended. Below are the values I had entered into the database:

4	4	HelloWord!!!	Bubble Sort	2	6
5	5	HelloWord!!!	Prim's	4	1
6	6	HelloWord!!!	Dijkstra's	5	0
7	7	HelloWord!!!	Simplex	0	3

As I was not fully sure how to fix this, I added a line to print the results from the SQL query. The following was returned:

I realised that despite the HomeMain class inheriting the attributes from the Main superclass, the actual values weren't in the scope so I changed the HomeMain class (and the algorithm and quiz classes) to take the username as a parameter and passed in the username in the closeOpen function. Additionally, I moved the attributes that I had previously moved to the Main superclass back to the LoginRegister class and moved the currentAccUsername to the HomeMain class.

I additionally encountered issues with the attributes being None so I ensured that the correct values were passed in as parameters in the loginCheck and registerCheck methods. See the code I wrote to fix these issues below:

2.1.m

```

34 def closeOpen(root, newType, username=None):
35     print("in closeOpen")
36     root.destroy()
37     if newType == "home":
38         homeRoot = Tk()
39         homeRoot.geometry('500x350+300+300') # to change
40         homeRoot.title("Algorithms Simulator and Teaching Tool")
41         running = HomeMain(homeRoot, username)
42         homeRoot.mainloop()
43
44 elif newType == "login":
45     loginRoot = Tk()
46     loginRoot.geometry('625x400')
47     loginRoot.title("Algorithms Simulator & Teaching Tool")
48     running = LoginRegister(loginRoot)
49     loginRoot.mainloop()

```

(Note that the username is passed in as a parameter in all the algorithm and quiz instances as well)

2.1.n

```

95 class Main:
96     def __init__(self, master):
97         self.master = master # this is the window
98         self.title = None
99         # database connection:
100        self.conn = sqlite3.connect("AlgorithmTeachingToolDB.db") # actual database
101        self.cursor = self.conn.cursor() # for selecting
102
103    def closeConn (self):
104        self.conn.close()

```

The edited code for the loginCheck method:

2.1.n

```

197     def loginCheck(self):
198         # if pass - go to closeLoginOpenHome
199         #print("in login check")
200         ...
201         validated = False
202         while not validated:
203             validated = self.validateLoginInput()
204             print("login validated")
205             ...
206             usernameHolder = self.username.get()
207             if self.validateLoginInput(): # is True
208                 #print("login validated")
209                 # clear input fields:
210                 self.loginInvalid["text"] = ""
211                 self.username.set("")
212                 self.password.set("")
213                 # open home page:
214                 closeOpen(self.master, "home", usernameHolder)

```

And for the registerCheck method:

2.1.o

```

340         self.newUsername.set("")
341         self.newPassword.set("")
342         self.newConfirmPassword.set("")
343         for value in self.algorithmsChosen.values():
344             value.set(0)
345         self.accountType.set("")
346         self.usernameInvalid["text"] = ""
347         self.passwordInvalid["text"] = ""
348         self.passwordsMatchInvalid["text"] = ""
349         self.invalidMessageReg["text"] = ""
350         closeOpen(self.master, "home", usernameHolder)
            ...

```

Below is the code for the constructor method and account type method in the HomeMain class:

2.1.p

```

410     class HomeMain(Main):
411         def __init__(self, master, pUsername):
412             super().__init__(master)
413             self.homeFrame = None
414
415             # current account username:
416             self.currentAccUsername = pUsername
417             self.accountType = self.accountTypeGet()
418
419             # holds algorithm names to go as labels on the x axis
420             self.algorithmNames = self.algorithmAxisValues("Name")
421             # hold the correct and incorrect scores to be plot as bars on the bar chart
422             self.correctScores = self.algorithmAxisValues("Correct")
423             self.incorrectScores = self.algorithmAxisValues("Incorrect")
424
425             self.homeWidgets()
426             #self.algorithmAxisValues()
427             self.plotStatistics()
428
429     def accountTypeGet(self):
430         usernameHolder = self.currentAccUsername
431         print(usernameHolder)
432         self.cursor.execute("SELECT Account_type FROM Users WHERE Username=?", (usernameHolder,))
433         accountType = self.cursor.fetchone()
434         print(accountType)
435         return accountType

```

2.1.q

```

461     def locations(self):
462         try:
463             buttonLocations = []
464             #self.cursor.execute("SELECT Account_type FROM Users WHERE Username=?", (self.currentAccUsername,))
465             #accountType = self.cursor.fetchone()
466             for name in self.algorithmNames:
467                 buttonLocations.append(name)
468             if self.accountType == "Student":
469                 buttonLocations.append("Quiz")
470             return buttonLocations
471         except sqlite3.Error as e:
472             print(f"Database error: {e}")

```

And the code for the algorithmAxisValues method:

2.1.r

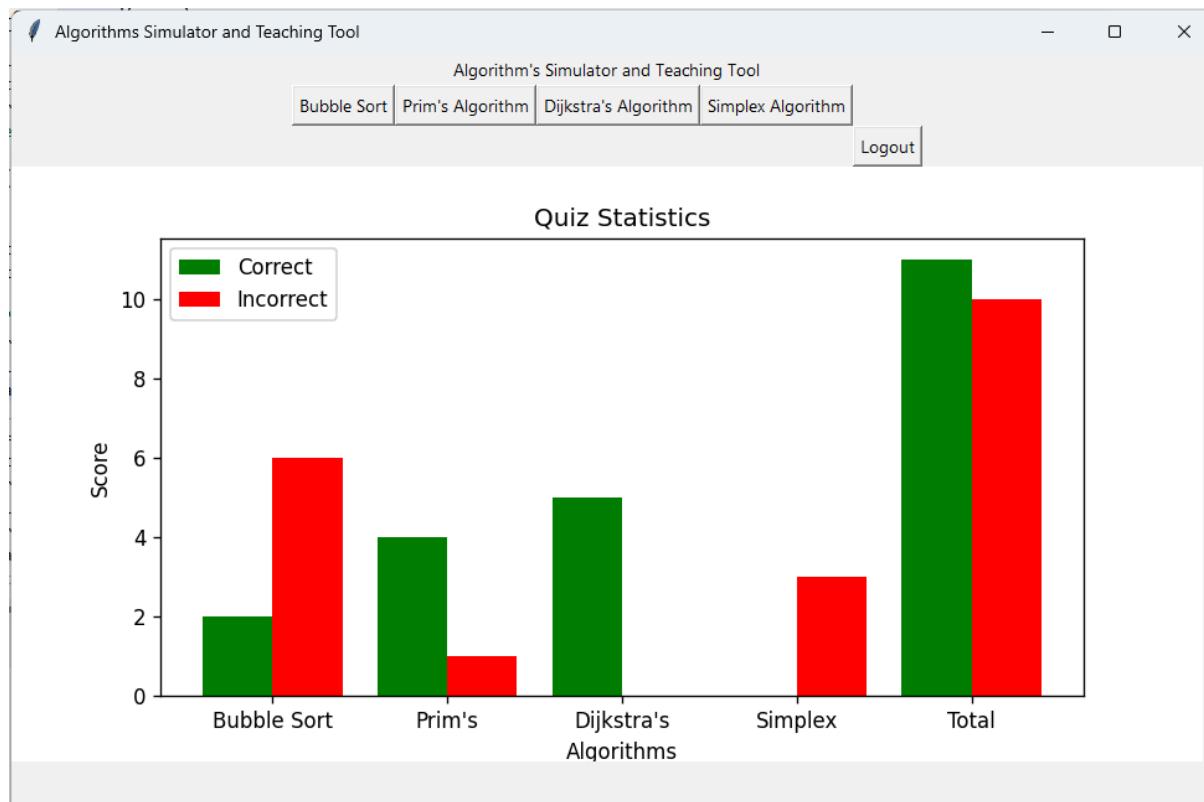
```

474     def algorithmAxisValues(self, listType):
475         valuesList = []
476         try:
477             self.cursor.execute("SELECT Algorithm, CorrectScore, IncorrectScore FROM StudentEnrolment WHERE Username = ?", (self.currentAccUsername,))
478             results = self.cursor.fetchall()
479             print(results)
480             if listType == "Name":
481                 for row in results:
482                     valuesList.append(row[0])
483                     valuesList.append("Total")
484                     #self.algorithmNames = valuesList
485             elif listType == "Correct":
486                 totalCorrect = 0
487                 for row in results:
488                     valuesList.append(row[1])
489                     totalCorrect += row[1]
490                     valuesList.append(totalCorrect)
491                     #self.correctScores = valuesList
492             elif listType == "Incorrect":
493                 totalIncorrect = 0
494                 for row in results:
495                     valuesList.append(row[2])
496                     totalIncorrect += row[2]
497                     valuesList.append(totalIncorrect)
498                     #self.incorrectScores = valuesList
499         except sqlite3.Error as e:
500             print(f"Database error: {e}")
501
502         return valuesList

```

When I now run this code, the following is output:

```
>>> %Run NEA_main_file.py
in closeOpen
HelloWord!!!
('Student',)
[('Bubble Sort', 2, 6), ("Prim's", 4, 1), ("Dijkstra's", 5, 0), ('Simplex', 0, 3)]
[('Bubble Sort', 2, 6), ("Prim's", 4, 1), ("Dijkstra's", 5, 0), ('Simplex', 0, 3)]
[('Bubble Sort', 2, 6), ("Prim's", 4, 1), ("Dijkstra's", 5, 0), ('Simplex', 0, 3)]
Correct Scores: [2, 4, 5, 0, 11]
Incorrect Scores: [6, 1, 0, 3, 10]
```



The main elements of this page are now displayed but the Logout button should be below the Quiz Statistics chart rather than above and the Quiz button does not appear.

The issue with the logout button being in the incorrect place was easily fixed by changing the master for the quiz statistics canvas to self.homeFrame:

2.1.s

```
527     # embed into Tkinter
528     canvas = FigureCanvasTkAgg(fig, self.homeFrame)
529     canvasWidget = canvas.get_tk_widget()
530     canvasWidget.grid(row=2, column=1, columnspan=3)
```

And for the Quiz button, I realised that the reason it wasn't working was because the accountType variable was holding a tuple (of one item). Therefore, I added the following code to make accountType just hold a string:

2.1.t

```

429     def accountTypeGet(self):
430         usernameHolder = self.currentAccUsername
431         print(usernameHolder)
432         self.cursor.execute("SELECT Account_type FROM Users WHERE Username=?", (usernameHolder,))
433         result = self.cursor.fetchone()
434         accountType = result[0]
435         print(accountType)
436         return accountType

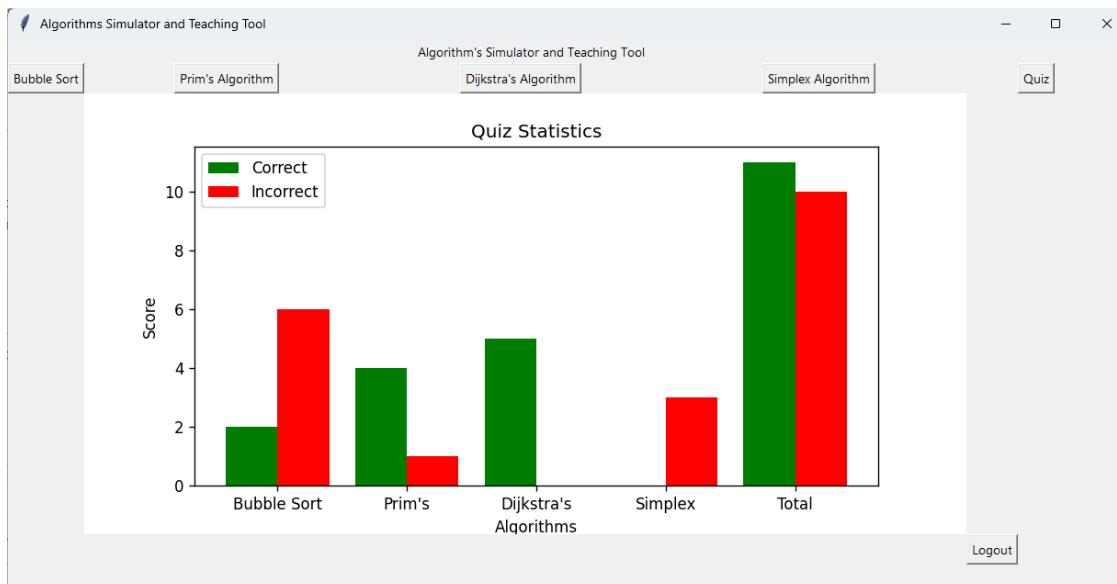
```

When this code is now run, the following is output:

```

>>> %Run NEA_main_file.py
in closeOpen
HelloWord!!!
Student
[('Bubble Sort', 2, 6), ("Prim's", 4, 1), ("Dijkstra's", 5, 0), ('Simplex', 0, 3)]
[('Bubble Sort', 2, 6), ("Prim's", 4, 1), ("Dijkstra's", 5, 0), ('Simplex', 0, 3)]
[('Bubble Sort', 2, 6), ("Prim's", 4, 1), ("Dijkstra's", 5, 0), ('Simplex', 0, 3)]
Correct Scores: [2, 4, 5, 0, 11]
Incorrect Scores: [6, 1, 0, 3, 10]

```



This is much closer to what I expected to be output.

When I pressed the Bubble Sort button, I found that the exact same screen was displayed, which I realised was due to the BubbleSort class inheriting everything from the HomeMain class. To fix this, I changed the HomeMain class to have a pageStarter method, which has the code to display the widgets and the quiz statistics in it. Then, I used the OOP concept of polymorphism²⁶ in each of the visualisation and quiz subclasses. Each of these has its own pageStarter method but I added code to each to ensure that the HomeMain pageStarter method is overridden. See the code below:

2.1.u

```

410 class HomeMain(Main):
411     def __init__(self, master, pUsername):
412         super().__init__(master)
413         self.homeFrame = None
414
415         # current account username:
416         self.currentAccUsername = pUsername
417         self.accountType = self.accountTypeGet()
418
419         # holds algorithm names to go as labels on the x axis
420         self.algorithmNames = self.algorithmAxisValues("Name")
421         # hold the correct and incorrect scores to be plot as bars on the bar chart
422         self.correctScores = self.algorithmAxisValues("Correct")
423         self.incorrectScores = self.algorithmAxisValues("Incorrect")
424
425         self.pageStarter()
426
427     def pageStarter(self):
428         self.homeWidgets()
429         #self.algorithmAxisValues()
430         self.plotStatistics()

```

Note that all the other algorithm and quiz classes have the same code as the BubbleSort class, except for the text in the Label. See this code below:

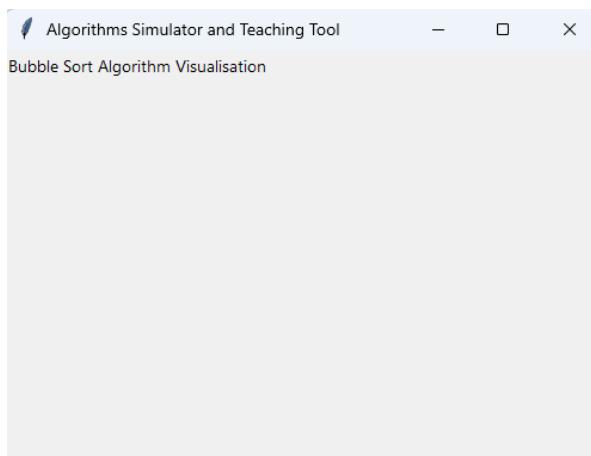
2.1.v

```

563 class BubbleSort(HomeMain):
564     def __init__(self, master, pUsername):
565         super().__init__(master, pUsername)
566
567         self.pageStarter()
568
569     def pageStarter(self):
570         print("in bubble sort page starter")
571         self.bubbleSortWidgets()
572
573     def bubbleSortWidgets(self):
574         self.title = Label(self.master, text="Bubble Sort Algorithm Visualisation")
575         self.title.grid(row=0, column=0, columnspan=5) # change columnspan
576
577     def openHome(self):
578         closeOpen(self.master, "home")

```

This worked as expected and a new window with a blank screen (except for the ‘Bubble Sort Visualisation’ title) is opened:



The main functionality of the Home page section has now been completed so I will now focus on enhancing the GUI of the Home page.

Firstly, I focused on the colours, widths and figure size for the Quiz statistics bar chart. See the code below:

2.1.w

```
# plot the bars
ax.bar(x, self.correctScores, width=0.3, label="Correct", align="center", color="#606c38")
ax.bar([p + 0.3 for p in x], self.incorrectScores, width=0.3, label = "Incorrect", align = "center", color = "#b26754")

# labels, title, legend
ax.set_xlabel("Algorithms", fontsize=11, fontfamily="Calibri", color="#1b263b")
ax.set_ylabel("Score", fontsize=11, fontfamily="Calibri", color="#1b263b")
ax.set_title("Quiz Statistics", fontsize=12, fontfamily="Calibri", color="#1b263b")
ax.set_xticks([p + 0.3 for p in x])
ax.set_xticklabels(self.algorithmNames, fontsize=10, fontfamily="Calibri", color="#1b263b")
ax.legend()

# embed into Tkinter
canvas = FigureCanvasTkAgg(fig, self.homeFrame)
canvasWidget = canvas.get_tk_widget()
canvasWidget.grid(row=2, column=1, columnspan=3)
canvasWidget.config(bg="#eaebed")
```

Then, I noticed that the quiz button was not inline with the Logout button, but rather seemed to be 1 grid space further along. To confirm this, I added a print statement for the buttonLocations. The output for this is shown below:

```
>>> %Run NEA_main_file.py

in closeOpen
HelloWord!!!
Student
[('Bubble Sort', 2, 6), ("Prim's", 4, 1), ("Dijkstra's", 5, 0), ('Simplex', 0, 3)]
[('Bubble Sort', 2, 6), ("Prim's", 4, 1), ("Dijkstra's", 5, 0), ('Simplex', 0, 3)]
[('Bubble Sort', 2, 6), ("Prim's", 4, 1), ("Dijkstra's", 5, 0), ('Simplex', 0, 3)]
['Bubble Sort', "Prim's", "Dijkstra's", 'Simplex', 'Total', 'Quiz']
Correct Scores: [2, 4, 5, 0, 11]
Incorrect Scores: [6, 1, 0, 3, 10]
```

The output to the Shell on the 7th line displays the buttonLocations. This includes a “Total” item. I first considered just adding a -1 to the grid column for the Quiz button to fix this. However, this is not very good practice and may make it more challenging for future development if more algorithms were to be added. Therefore, I changed the location method so that it does not include the “Total” item in the buttonLocations list.

2.1.x

```
470     def locations(self):
471         try:
472             buttonLocations = []
473             #self.cursor.execute("SELECT Account_type FROM Users WHERE Username=?",
474             #accountType = self.cursor.fetchone()
475             for counter in range (len(self.algorithmNames)-1):
476                 buttonLocations.append(self.algorithmNames[counter])
477             if self.accountType == "Student":
478                 buttonLocations.append("Quiz")
479             return buttonLocations
480         except sqlite3.Error as e:
481             print(f"Database error: {e}")
```

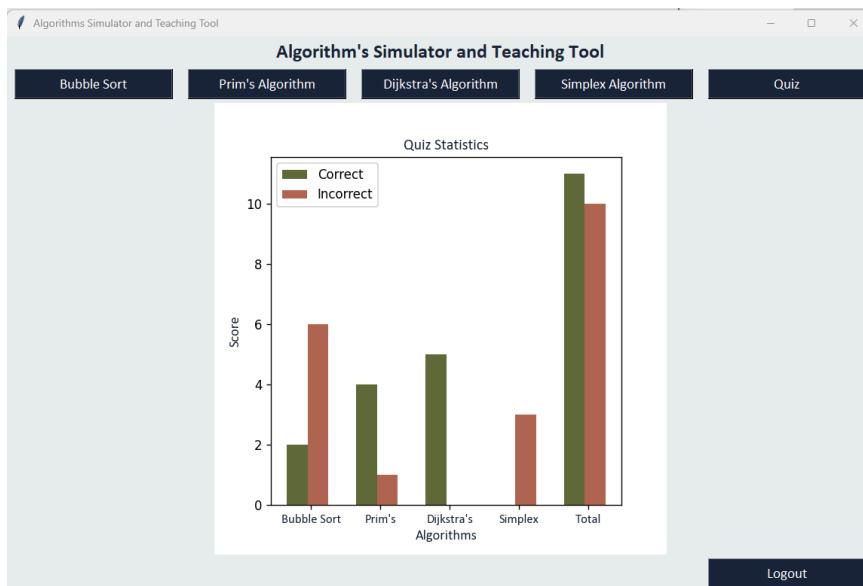
The for counter in range’ portion of the for loop means that the iteration stops 1 before the value of the range value. Subtracting 1 from the length means that the iteration will stop at the 2nd to last value in self.algorithmNames. This means that the “Total” item will not be added to buttonLocations. When I now run the code, the following are output:

```
>>> %Run NEA_main_file.py
in closeOpen
HelloWord!!!
Student
[('Bubble Sort', 2, 6), ("Prim's", 4, 1), ("Dijkstra's", 5, 0), ('Simplex', 0, 3)]
[('Bubble Sort', 2, 6), ("Prim's", 4, 1), ("Dijkstra's", 5, 0), ('Simplex', 0, 3)]
[('Bubble Sort', 2, 6), ("Prim's", 4, 1), ("Dijkstra's", 5, 0), ('Simplex', 0, 3)]
['Bubble Sort', "Prim's", "Dijkstra's", 'Simplex', 'Quiz']
Correct Scores: [2, 4, 5, 0, 11]
Incorrect Scores: [6, 1, 0, 3, 10]
```

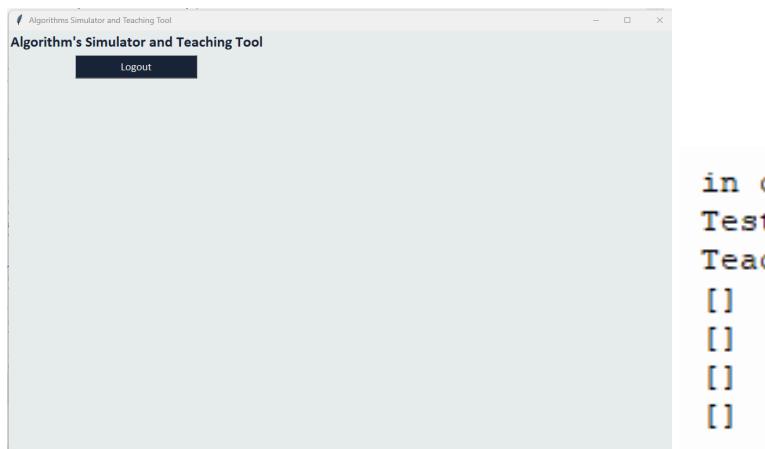


This is as expected. I then changed the geometry of the window to make sure that everything fit correctly and I changed the size of the bar chart a bit to make it clearer. To make these changes, I changed the figure size to width 5 and height 5 and I edited the geometry of the homeRoot so that the width was 1150 and the height was 740, as I determined that these fit the screen the best. Additionally, I noticed that the background colours did not quite match at the top so I configured the master attribute so the background colour was #eaebcd.

The following is output when the code is run:



Finally, I was thinking about what the home page would look like if I logged in with a Teacher account - which would not have a quiz statistics bar chart in the centre. For example, if I logged in with the Test_accT account that I had already created, which is a Teacher account and has Prim's and Dijkstra's algorithms selected, the following is output:



I realised that this output, which was not as expected, was because I had not set up a way to select the algorithms for a Teacher account (I had only set up a selection of algorithms and scores from the StudentEnrolment table). Therefore, I added the following functionality to select the algorithms from the TeacherEnrolment table and store them in the algorithmNames attribute:

2.1.y

```

if self.accountType == "Student":
    # holds algorithm names to go as labels on the x axis
    self.algorithmNames = self.algorithmAxisValues("Name")
    # hold the correct and incorrect scores to be plot as bars on the bar chart
    self.correctScores = self.algorithmAxisValues("Correct")
    self.incorrectScores = self.algorithmAxisValues("Incorrect")

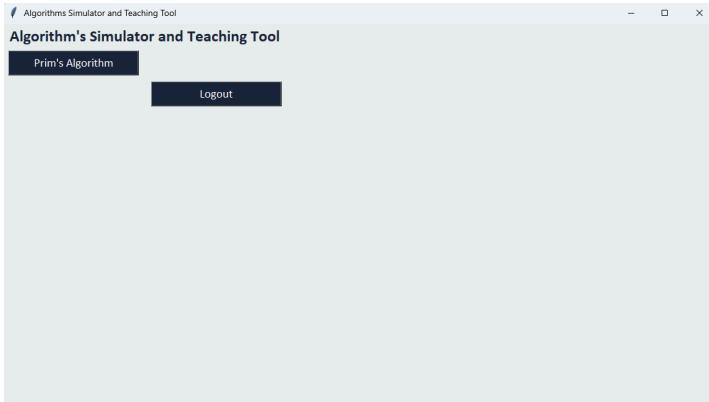
elif self.accountType == "Teacher":
    self.algorithmNames = self.teacherAlgorithmsGet()

```

2.1.z

```
def teacherAlgorithmsGet(self):
    usernameHolder = self.currentAccUsername
    algorithmList = []
    self.cursor.execute("SELECT Algorithm FROM TeacherEnrolment WHERE Username=?", (usernameHolder,))
    result = self.cursor.fetchall()
    for row in result:
        algorithmList.append(row[0])
    return algorithmList
```

When the code is now run, the following is output:



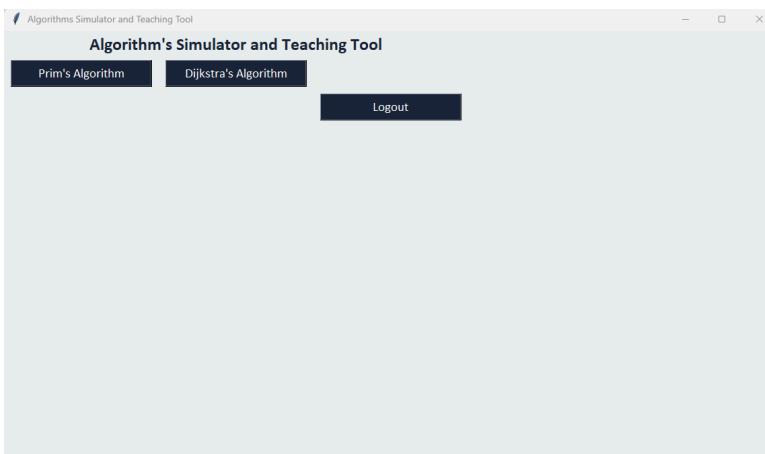
I realised that this error was because the locations method stops 1 before the end of the algorithmNames list because “Total” was included in it for Student accounts. To fix this problem, I added an if, elif statement to ensure that the correct algorithms are added:

2.1.A

```
def locations(self):
    try:
        buttonLocations = []
        #self.cursor.execute("SELECT Account_type FROM Users WHERE Username=?",
        #accountType = self.cursor.fetchone()

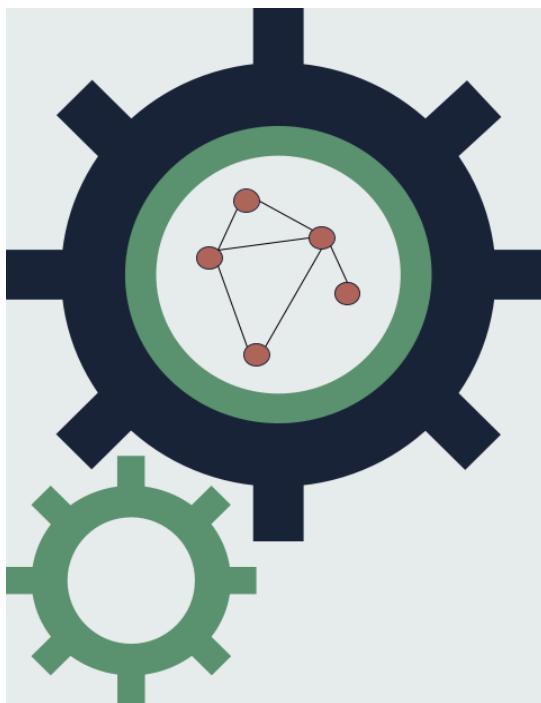
        if self.accountType == "Student":
            for counter in range(len(self.algorithmNames)-1):
                buttonLocations.append(self.algorithmNames[counter])
            buttonLocations.append("Quiz")
        elif self.accountType == "Teacher":
            buttonLocations = self.algorithmNames
        return buttonLocations
    except sqlite3.Error as e:
        print(f"Database error: {e}")
```

The output is now as expected at this:



Returning to thinking about the appearance of the GUI, from this screenshot, it is clear that the Home page looks very empty. So, I have decided to add a logo image in the centre of the page (in the

same grid space as the quiz statistics bar chart). Below is a logo design that I have created, which is only an initial design and will likely be changed if the system was developed further to be released to the public:



To display this image, I imported the `Image` and `ImageTk` modules from the `PIL` library, which would allow me to open the image, resize it and create a Tkinter widget with it.

2.1.B

```
1 from tkinter import *
2 import tkinter as tk
3 import sqlite3
4 import string # for checking for special chars
5 import matplotlib.pyplot as plt
6 from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
7 from PIL import Image, ImageTk # for logo image editing
```

Then I created a `displayLogo` method in the `HomeMain` class:

2.1.C

```
def displayLogo(self):
    # open the image
    image = Image.open("NEA_logo_image.png")
    # image = image.resize()
    logoImg = ImageTk.PhotoImage(image) # creates tkinter widget

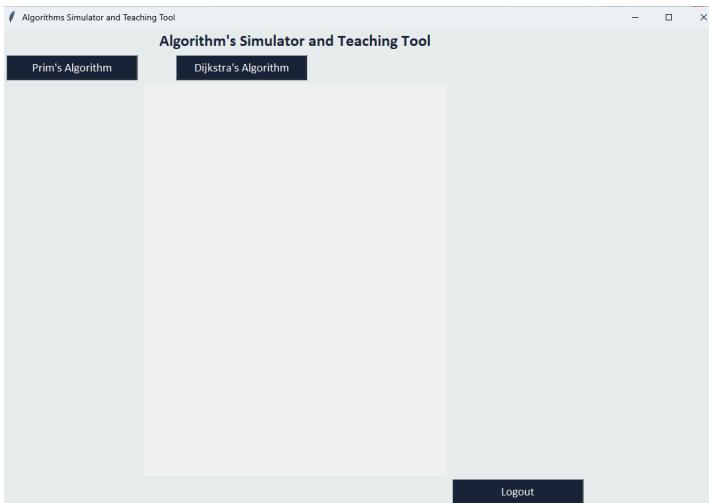
    # create a label and set the image
    imageLabel = Label(self.homeFrame, image=logoImg)
    imageLabel.grid(row=2, column=1, columnspan=3)
```

Then, I called this method in the `pageStarter` method:

2.1.D

```
def pageStarter(self):
    self.homeWidgets()
    if self.accountType == "Student":
        self.plotStatistics()
    elif self.accountType == "Teacher":
        self.displayLogo()
```

When the code is now run, the following is output:



I realised that this error is occurring because the image will not stay on displayed on the window if it is not an attribute of the class. Therefore, I modified the constructor method to create an attribute called logoImg and then edited the displayLogo method:

2.1.E

```
class HomeMain(Main):
    def __init__(self, master, pUsername):
        super().__init__(master)
        self.homeFrame = None

        # current account username:
        self.currentAccUsername = pUsername
        self.accountType = self.accountTypeGet()

        if self.accountType == "Student":
            # holds algorithm names to go as labels on the x axis
            self.algorithmNames = self.algorithmAxisValues("Name")
            # hold the correct and incorrect scores to be plot as bars on the y axis
            self.correctScores = self.algorithmAxisValues("Correct")
            self.incorrectScores = self.algorithmAxisValues("Incorrect")

        elif self.accountType == "Teacher":
            self.algorithmNames = self.teacherAlgorithmsGet()

        self.logoImg = None

        self.master.configure(bg="#eaebed")

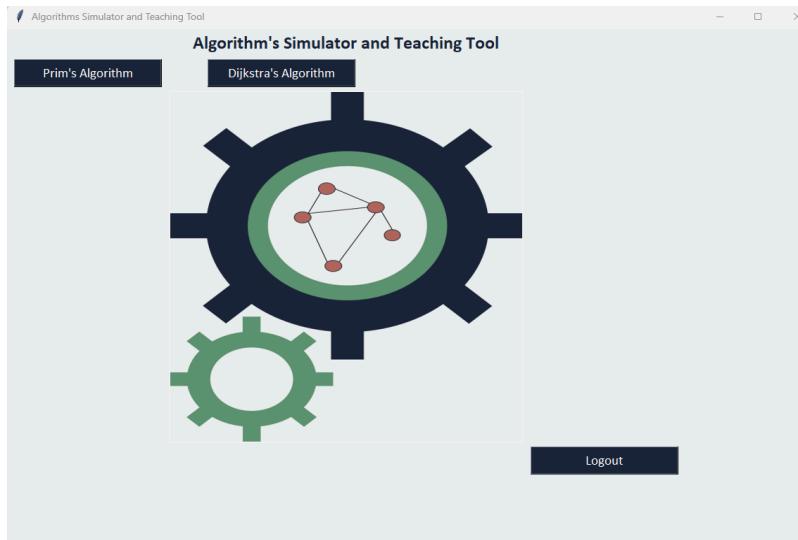
        self.pageStarter()
```

2.1.F

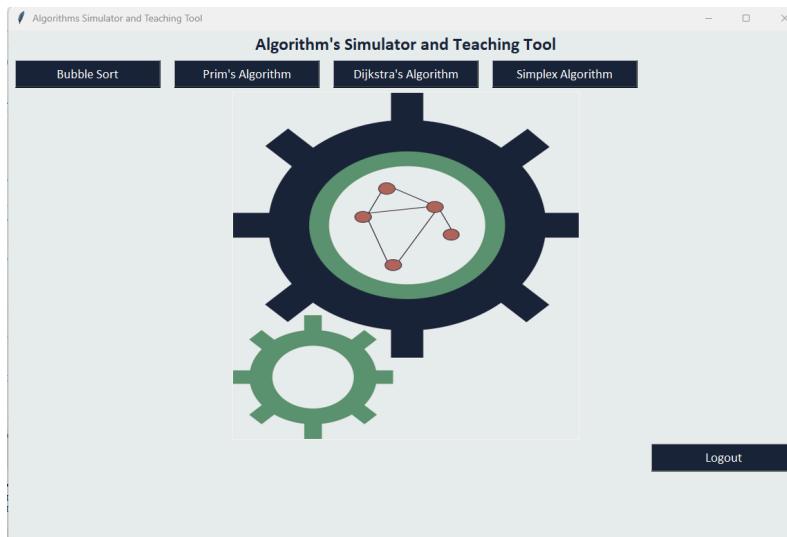
```
def displayLogo(self):
    # open the image
    image = Image.open("NEA_logo_image.png")
    image = image.resize((500,500))
    self.logoImg = ImageTk.PhotoImage(image) # creates tkinter widget

    # create a label and set the image
    imageLabel = Label(self.homeFrame, image=self.logoImg)
    imageLabel.grid(row=2, column=1, columnspan=3)
```

The following is now displayed:



This output is as expected. However, when I log in with a teacher account that selected all the algorithms, the following is output:



To modify this so that there is less empty space, I configured the size of the window, changed the size of the logo, and I moved the Logout button below the image for Teacher accounts as there is no Quiz button.

2.1.G

```

def displayLogo(self):
    # open the image
    image = Image.open("NEA_logo_image.png")
    image = image.resize((481,626))
    self.logoImg = ImageTk.PhotoImage(image) # creates tkinter widget

    # create a label and set the image
    imageLabel = Label(self.homeFrame, image=self.logoImg)
    imageLabel.grid(row=2, column=0, columnspan=4, rowspan=5)

```

For the Logout buttons, I changed the grid layouts so that Student and Teacher accounts have a button that fits correctly:

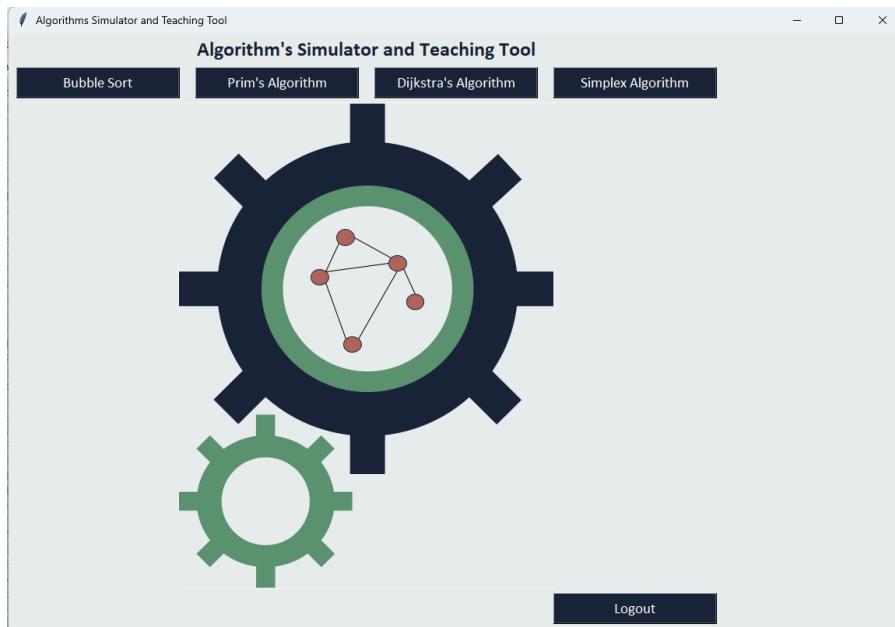
2.1.H

```

.grid(row=4, column=4, padx=10, pady=5) # logout button Student
.grid(row=7, column=3, padx=10, pady=5) # logout button Teacher

```

When the code is now run, the following is displayed:



This layout is much better than before. The final thing to do is configure the size of the window:

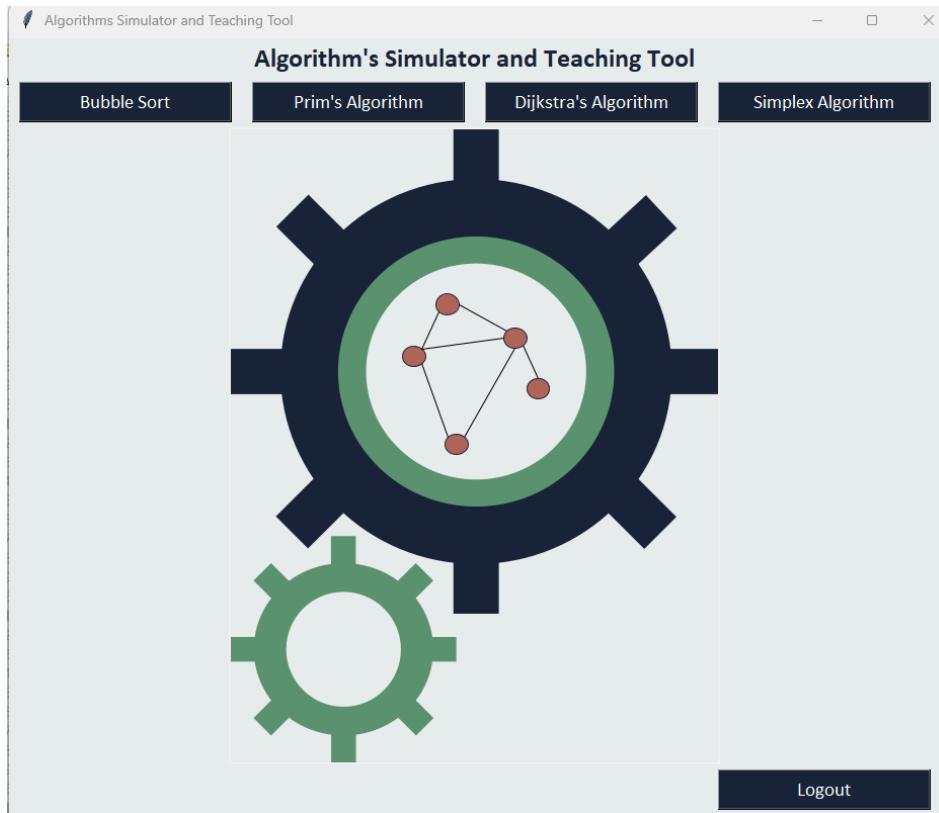
2.1.I

```

if self.accountType == "Student":
    # holds algorithm names to go as labels on the x axis
    self.algorithmNames = self.algorithmAxisValues("Name")
    # hold the correct and incorrect scores to be plot as bars on the
    self.correctScores = self.algorithmAxisValues("Correct")
    self.incorrectScores = self.algorithmAxisValues("Incorrect")
    self.master.geometry("1150x740")

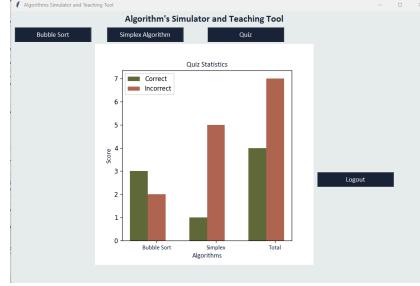
elif self.accountType == "Teacher":
    self.algorithmNames = self.teacherAlgorithmsGet()
    self.master.geometry("935x775")

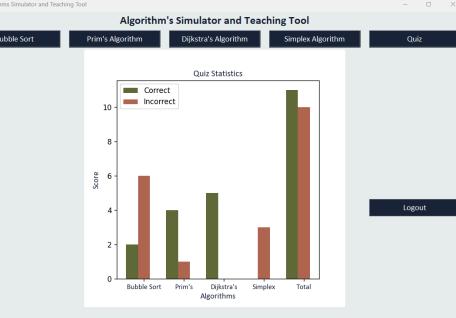
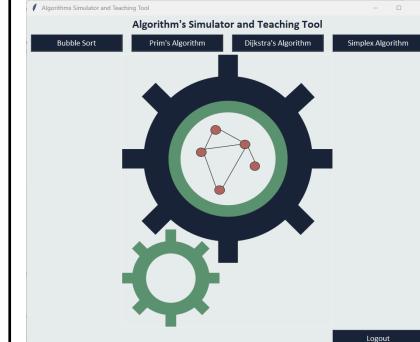
```

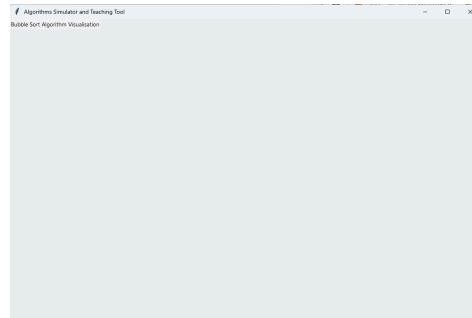


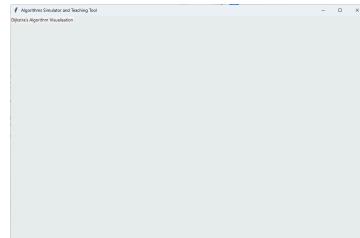
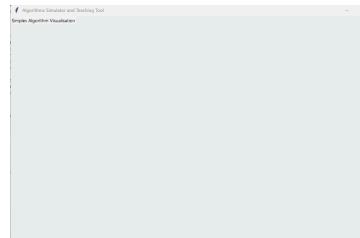
This layout is again much better. All the functionality and appearance is now complete and ready for testing.

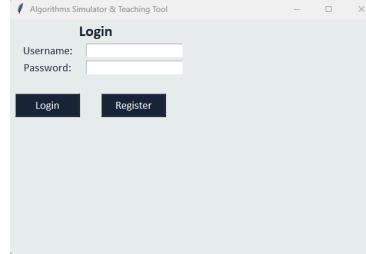
I will now test the code using the white box testing table specified in the HOME PAGE design section. I have also edited the expected results to include the logo being displayed:

TEST DATA	EXPECTED RESULT	JUSTIFICATION	CODE	ACTUAL OUTPUT
Log in with Student account	Home page GUI with selected algorithms buttons and quiz statistics chart displayed	This ensures that a student user has the correct algorithms that they registered with as options to visualise and quiz functionality, which includes a quiz statistics chart and a quiz button. Additionally, there should be a Logout button.	2.1.f 2.1.h 2.1.l 2.1.p 2.1.t 2.1.r 2.1.q	 <pre> in closeOpen Test_accS Student [("Bubble Sort", 3, 2), ('Simplex', 1, 5)] [("Bubble Sort", 3, 2), ('Simplex', 1, 5)] [("Bubble Sort", 3, 2), ('Simplex', 1, 5)] ['Bubble Sort', 'Simplex', 'Quiz'] Correct Scores: [3, 1, 4] Incorrect Scores: [2, 5, 7] </pre> <p>All elements are present but the Logout button should be lower.</p>
Log in with Teacher account	Home page GUI with selected algorithms buttons and logo image displayed	This ensures that a teacher user has the correct algorithms that they registered with as options to visualise and the logo in the middle of the page to enhance the appearance. Also, there should be a Logout button.	2.1.q 2.1.s 2.1.t 2.1.z 2.1.A 2.1.B 2.1.G 2.1.H 2.1.I	 <pre> in closeOpen Test_accT Teacher [("Prim's",), ("Dijkstra's",)] ["Prim's", "Dijkstra's"] ["Prim's", "Dijkstra's"] </pre> <p>As expected.</p>

Log in with multiple Student accounts with different algorithms choices and quiz scores already entered.	The correct algorithms are displayed as buttons and the quiz statistics are displayed correctly on the chart.	This ensures that the functionality is as expected for all student users for whatever choice of algorithms they had made when registering and whatever scores they have from the quiz section.	2.1.f 2.1.h 2.1.l 2.1.p 2.1.t 2.1.r 2.1.q	 <p>The chart displays quiz statistics with the following data:</p> <table border="1"> <thead> <tr> <th>Category</th> <th>Correct</th> <th>Incorrect</th> </tr> </thead> <tbody> <tr> <td>Bubble Sort</td> <td>2</td> <td>6</td> </tr> <tr> <td>Prim's</td> <td>1</td> <td>1</td> </tr> <tr> <td>Dijkstra's Algorithms</td> <td>5</td> <td>0</td> </tr> <tr> <td>Simplex</td> <td>3</td> <td>0</td> </tr> <tr> <td>Total</td> <td>11</td> <td>10</td> </tr> </tbody> </table> <pre> in closeOpen HelloWord!!! Student [("Bubble Sort", 2, 6), ("Prim's", 4, 1), ("Dijkstra's", 5, 0), ("Simplex", 0, 3)] [("Bubble Sort", 2, 6), ("Prim's", 4, 1), ("Dijkstra's", 5, 0), ("Simplex", 0, 3)] [("Bubble Sort", 2, 6), ("Prim's", 4, 1), ("Dijkstra's", 5, 0), ("Simplex", 0, 3)] ["Bubble Sort", "Prim's", "Dijkstra's", "Simplex", "Quiz"] Correct Scores: [2, 4, 5, 0, 11] Incorrect Scores: [6, 1, 0, 3, 10] </pre> <p>All elements are present but the Logout button should be lower.</p>	Category	Correct	Incorrect	Bubble Sort	2	6	Prim's	1	1	Dijkstra's Algorithms	5	0	Simplex	3	0	Total	11	10
Category	Correct	Incorrect																				
Bubble Sort	2	6																				
Prim's	1	1																				
Dijkstra's Algorithms	5	0																				
Simplex	3	0																				
Total	11	10																				
Log in with multiple Teacher accounts with different algorithms choices.	The correct algorithm buttons are displayed.	This ensures that the functionality is as expected for all teacher users for whatever choice of algorithms they had made when registering.	2.1.q 2.1.s 2.1.t 2.1.z 2.1.A 2.1.B 2.1.G 2.1.H 2.1.I																			

				<pre>in closeOpen FullTestT Teacher [('Bubble Sort',), ("Prim's",), ("Dijkstra's",), ('Simplex',)] ['Bubble Sort', "Prim's", "Dijkstra's", 'Simplex'] ['Bubble Sort', "Prim's", "Dijkstra's", 'Simplex']</pre> <p>As expected.</p>
Press the Bubble Sort button	The home page window is closed and the Bubble Sort window is opened.	This ensures that the Bubble Sort button works correctly.	2.1.a 2.1.b 2.1.c	 <pre>in bubble sort page starter in bubble sort page starter</pre> <p>As expected given that the Bubble Sort page has not been developed.</p>
Press the Prim's Algorithm button	The home page window is closed and the Prim's algorithm window is opened.	This ensures that the Prim's algorithm button works correctly.	2.1.a 2.1.b 2.1.c	 <pre>in prim's page starter in prim's page starter</pre> <p>As expected given that the Prim's Algorithm page has not been developed.</p>

Press the Dijkstra's Algorithm button	The home page window is closed and the Dijkstra's algorithm window is opened.	This ensures that the Dijkstra's algorithm button works correctly.	2.1.a 2.1.b 2.1.d	 in dijkstra page starter in dijkstra page starter	As expected given that the Dijkstra's Algorithm page has not been developed.
Press the Simplex Algorithm button	The home page window is closed and the Simplex algorithm window is opened.	This ensures that the Simplex algorithm button works correctly.	2.1.a 2.1.b 2.1.d	 in simplex page starter in simplex page starter	As expected given that the Simplex algorithm page has not been developed.
Press the Quiz button	The home page window is closed and the Quiz window is opened.	This ensures that the Quiz button works correctly.	2.1.a 2.1.b 2.1.e	 in quiz page starter in quiz page starter	As expected given that the Quiz page has not been developed.

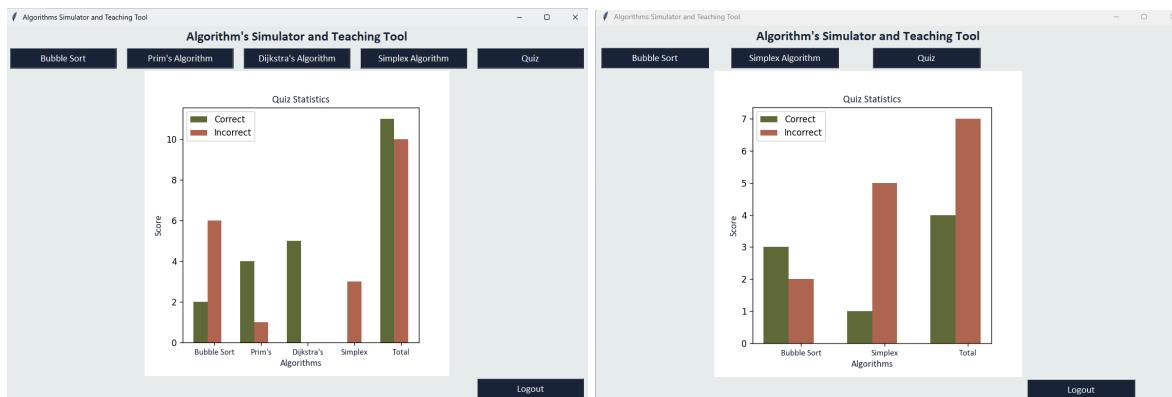
Press the Logout button	The home page window is closed and the Initial Login window is opened.	This ensures that the Logout button works correctly.	2.1.a 2.1.b 1.1.d 1.1.e 1.1.f 1.3.b 1.3.d		in closeOpen As expected.
-------------------------	--	--	---	---	------------------------------

Overall, all tests passed. The only issue that occurred was the location of the Logout button on Student accounts. I realised that I had changed the grid of the quiz statistics chart when changing the grid for the Image. Therefore, I changed it back so the code is now the following:

2.1.J

```
# embed into Tkinter
canvas = FigureCanvasTkAgg(fig, self.homeFrame)
canvasWidget = canvas.get_tk_widget()
canvasWidget.grid(row=2, column=1, columnspan=3)
canvasWidget.config(bg="#eaebed")
```

And the outputs are the following for the two Student login tests:



All tests have now passed so the usability and functionality features for the Home page have been fully developed.

ITERATION 1 - DEVELOPER REVIEW

During this iteration of the development of PROTOTYPE 2, which focused on developing the Home and Bubble Sort Visualisation pages and the Menu, I focused on the home page. This included the main Tkinter GUI for the page, the quiz statistics bar chart and the buttons to the different visualisation, quiz and login pages. When beginning this iteration, I thought that as there was less complex functionality to do with validating user inputs than the login and registration pages, development would be easier. However, I encountered more issues than expected as I had to try to find solutions to getting the username for the current account and then getting the values for the axes of the bar chart. I think that these issues occurred due to me not having extensive experience with OOP. Despite this, I managed to fix these issues by expanding the closeOpen function to include username as a parameter. The errors and challenges that I faced are a good learning experience and I believe will help me with my OOP skills, which will be vital to the development of the rest of the system. Enhancement of the GUI went more smoothly.

ITERATION 2 - BUBBLE SORT FUNCTIONALITY

As the Bubble Sort section of the system is the first algorithm visualisation section of the system that I designed and will now develop, I have split it into 2 iterations. The first of these is for the main functionality of the section and the second is for refining the GUI and adding the Menu.
I first set up the GUI, including all the entry fields and buttons. See the code I wrote below:

2.2.a

```

630
631     def pageStarter(self):
632         print("in bubble sort page starter")
633         self.bubbleSortWidgets()
634
635     def bubbleSortWidgets(self):
636         # TITLE / TOP SECTION:
637         self.title = Label(self.master, text="Bubble Sort Algorithm Visualisation")
638         self.title.grid(row=0, column=0, columnspan=17) # change columnspan
639
640         Button(self.master, text="Home", command=self.openHome).grid(row=0, column=17, columnspan=2)
641
642         # BUBBLE SORT FRAME:
643         self.bubbleSortFrame = Frame(self.master)
644         self.bubbleSortFrame.grid(row=1, column=0, columnspan=11, rowspan=19)
645
646         Label(self.bubbleSortFrame, text="1. Start at the first item in the list.\n2. Compare the current item with the next item.\n3. If the two items are in the wrong position, swap them.\n\n").grid(row=2, column=0, columnspan=11)
647
648         print(self.userEntries) # causing error???
649         # create 8 entry fields for user to enter numbers into
650         for i in range(8):
651             entry = Entry(self.bubbleSortFrame, width=3)
652             entry.grid(row=4 , column=(i+7))
653             self.userEntries.append(entry)
654
655         self.addButton = Button(self.bubbleSortFrame, text="Add Number", command=self.addEntry).grid(row=5, column=15, columnspan=4)

```

2.2.b

```

658     # ascending descending choice:
659     Radiobutton(self.bubbleSortFrame, text="Ascending", variable=self.sortOrder, value="Ascending").grid(row=10, column=0, columnspan=3)
660     Radiobutton(self.bubbleSortFrame, text="Descending", variable=self.sortOrder, value="Descending").grid(row=10, column=3, columnspan=3)
661
662     Button(self.bubbleSortFrame, text="Visualise", command=self.validate).grid(row=10, columns=11, columnspan=4)
663
664     self.invalidMessage = Label(self.bubbleSortFrame, text="")
665     self.invalidMessage.grid(row=11, columnspan=19)
666
667     # dataset plot
668     self.fig, self.ax = plt.subplots()
669     self.canvas = FigureCanvasTkAgg(self.fig, master=self.bubbleSortFrame)
670     self.canvas.get_tk_widget().grid(row=4, column=0, columnspan=7, rowspan=6)

```

2.2.c

```

673     def addEntry(self):
674         print("in add entry")
675         if len(self.userEntries) < 11:
676             entry = Entry(self.bubbleSortFrame, width=3)
677             entry.grid(row=4, column=(7+len(self.userEntries)))
678             self.userEntries.append(entry)
679         elif len(self.userEntries) == 11:
680             entry = Entry(self.bubbleSortFrame, width=3)
681             entry.grid(row=4, column=(7+len(self.userEntries)))
682             self.userEntries.append(entry)
683             self.addButton.destroy()

```

2.2.d

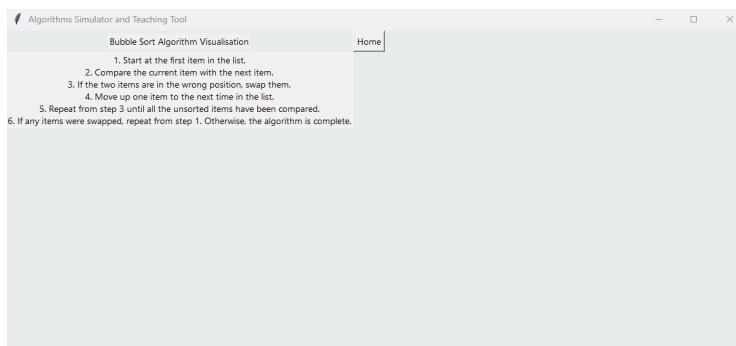
```

689     def validate(self):
690         print("in validate")
691         for entry in self.userEntries:
692             try:
693                 num = int(entry.get())
694                 if 1 <= num <= 100:
695                     self.numbers.append(num)
696                 else:
697                     raise ValueError
698             except ValueError:
699                 self.invalidMessage["text"] = "Invalid dataset: all values must be integers between 1 and 100"
700             return
701
702         if len(self.numbers) < 8 or len(self.numbers) > 12:
703             print("in length error")
704             self.invalidMessage["text"] = "Invalid dataset: the dataset must consist of between 8 and 12 numbers (inclusive)"
705             return
706
707         self.bubbleSortAnimate()

708
709
710     def bubbleSortAnimate(self):
711         print("in bubbleSortAnimate")
712         order = self.sortOrder.get()
713         n = len(self.numbers)
714         swapped = True
715         print(f"Order: {order}")
716         print(self.numbers)

```

This code does not include the actual implementation of the bubble sort algorithm. When this code is run, the following are output:



```

in bubble sort page starter
Exception in Tkinter callback
Traceback (most recent call last):
  File "C:\Users\ingri\AppData\Local\Programs\Thonny\lib\tkinter\_init_.py", line 1921, in _call_
    return self.func(*args)
  File "C:\Users\ingri\OneDrive\Documents\QM\QM School work\A Level\Computer Science\NEA\NEA main file.py", line 585, in openBubbleSort
    closeOpen(self.master, "bubble sort", usernameHolder)
  File "C:\Users\ingri\OneDrive\Documents\QM\QM School work\A Level\Computer Science\NEA\NEA main file.py", line 58, in closeOpen
    running = BubbleSort(bubbleSortRoot, username)
  File "C:\Users\ingri\OneDrive\Documents\QM\QM School work\A Level\Computer Science\NEA\NEA main file.py", line 612, in __init__
    super().__init__(master, pUsername)
  File "C:\Users\ingri\OneDrive\Documents\QM\QM School work\A Level\Computer Science\NEA\NEA main file.py", line 439, in __init__
    self.pageStarter()
  File "C:\Users\ingri\OneDrive\Documents\QM\QM School work\A Level\Computer Science\NEA\NEA main file.py", line 632, in pageStarter
    self.bubbleSortWidgets()
  File "C:\Users\ingri\OneDrive\Documents\QM\QM School work\A Level\Computer Science\NEA\NEA main file.py", line 649, in bubbleSortWidgets
    print(self.userEntries)
AttributeError: 'BubbleSort' object has no attribute 'userEntries'

```

I seem to be encountering an issue with the userEntries attribute, which I have created in the constructor method but I seem to be getting an error saying that it does not exist. I was not sure

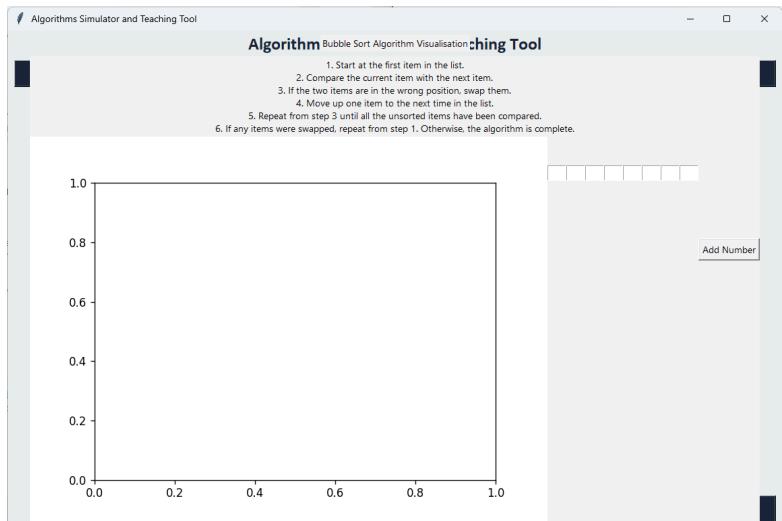
what was causing this error so I put in some print statements before and after the super() part of both the BubbleSort constructor and the HomeMain constructor. This was the output:

```

in closeOpen
inside HomeMain __init__
Back inside HomeMain __init__
in closeOpen
inside __init__ 1
inside HomeMain __init__
Back inside HomeMain __init__
in bubble sort page starter
Exception in Tkinter callback
Traceback (most recent call last):
  File "C:/Users/ingri/AppData/Local/Programs/Thonny/lib/tkinter/ init .py", line 1921, in
all
    return self.func(*args)
  File "C:/Users/ingri/OneDrive/Documents/QM/QM School work/A Level/Computer Science/NEA/NEA
n file.py", line 587, in openBubbleSort
    closeOpen(self.master, "bubble sort", usernameHolder)
  File "C:/Users/ingri/OneDrive/Documents/QM/QM School work/A Level/Computer Science/NEA/NEA
n file.py", line 58, in closeOpen
    running = BubbleSort(bubbleSortRoot, username)

```

This shows that the BubbleSort constructor is not reentered when the HomeMain constructor has been entered. After doing some research, I realised that this was due to both classes having a pageStarter method. To try and fix this, I changed the methods to have different names. The following are output:



```

in closeOpen
inside HomeMain __init__
Back inside HomeMain __init__
in closeOpen
inside __init__ 1
inside HomeMain __init__
Back inside HomeMain __init__
inside __init__ 2
[]
[]
in bubble sort page starter
numbers: []
userEntries: []
userEntries: []
userEntries: []
userEntries: [<tkinter.Entry object .!frame2.!entry>]
userEntries: [<tkinter.Entry object .!frame2.!entry>, <tkinter.Entry object .!frame2.!entry2>]
userEntries: [<tkinter.Entry object .!frame2.!entry>, <tkinter.Entry object .!frame2.!entry2>, <
tkinter.Entry object .!frame2.!entry3>]
userEntries: [<tkinter.Entry object .!frame2.!entry>, <tkinter.Entry object .!frame2.!entry2>, <
tkinter.Entry object .!frame2.!entry3>, <tkinter.Entry object .!frame2.!entry4>]
userEntries: [<tkinter.Entry object .!frame2.!entry>, <tkinter.Entry object .!frame2.!entry2>, <
tkinter.Entry object .!frame2.!entry3>, <tkinter.Entry object .!frame2.!entry4>, <tkinter.Entry
object .!frame2.!entry5>]
userEntries: [<tkinter.Entry object .!frame2.!entry>, <tkinter.Entry object .!frame2.!entry2>, <
tkinter.Entry object .!frame2.!entry3>, <tkinter.Entry object .!frame2.!entry4>, <tkinter.Entry
object .!frame2.!entry5>, <tkinter.Entry object .!frame2.!entry6>]
userEntries: [<tkinter.Entry object .!frame2.!entry>, <tkinter.Entry object .!frame2.!entry2>, <
tkinter.Entry object .!frame2.!entry3>, <tkinter.Entry object .!frame2.!entry4>, <tkinter.Entry
object .!frame2.!entry5>, <tkinter.Entry object .!frame2.!entry6>, <tkinter.Entry object .!frame
2.!entry7>]
userEntries: [<tkinter.Entry object .!frame2.!entry>, <tkinter.Entry object .!frame2.!entry2>, <
tkinter.Entry object .!frame2.!entry3>, <tkinter.Entry object .!frame2.!entry4>, <tkinter.Entry
object .!frame2.!entry5>, <tkinter.Entry object .!frame2.!entry6>, <tkinter.Entry object .!frame
2.!entry7>]

```

From the Shell, it is clear that the BubbleSort constructor is reentered so the issue with numbers and userEntries not being initialised has been resolved. However, from the GUI, it is clear that the widgets from the home page are still present. Therefore, I created a method in the HomeMain class to forget all the widgets. I then called this method in the pageStarterBubbleSort method.

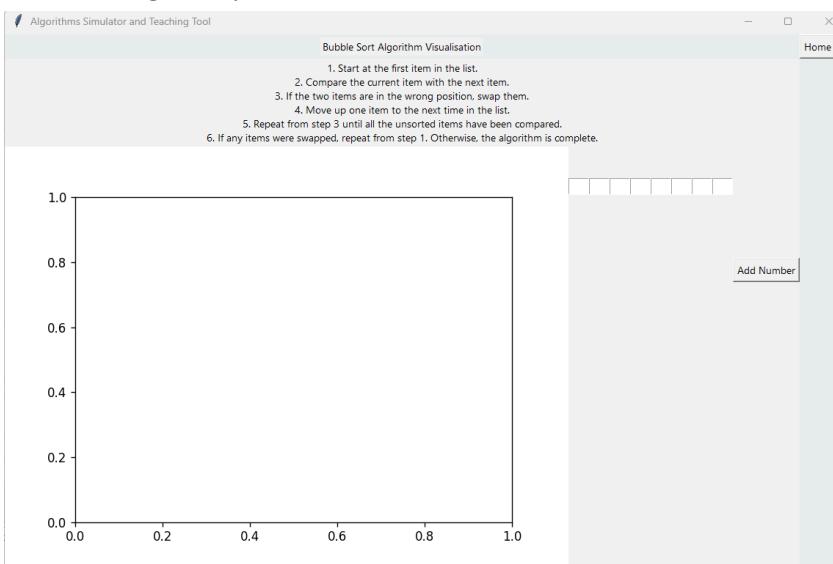
2.2.e

```
609     def hideHomeWidgets(self):
610         for widget in self.master.winfo_children():
611             widget.grid_forget()
```

2.2.f

```
642     def pageStarterBubbleSort(self):
643         print("in bubble sort page starter")
644         print(f"numbers: {self.numbers}")
645         print(f"userEntries: {self.userEntries}")
646         self.hideHomeWidgets()
647         self.bubbleSortWidgets()
```

The following is output:



```
in closeOpen
inside HomeMain __init__
Back inside HomeMain __init__
in closeOpen
inside __init__ 1
inside HomeMain __init__
Back inside HomeMain __init__
inside __init__ 2
[]
[]
in bubble sort page starter
numbers: []
userEntries: []
userEntries: []
userEntries: []
userEntries: [<tkinter.Entry object .!frame2.!entry>]
userEntries: [<tkinter.Entry object .!frame2.!entry>, <tkinter.Entry object .!frame2.!entry2>]
userEntries: [<tkinter.Entry object .!frame2.!entry>, <tkinter.Entry object .!frame2.!entry2>, <tkinter.Entry object .!frame2.!entry3>]
userEntries: [<tkinter.Entry object .!frame2.!entry>, <tkinter.Entry object .!frame2.!entry2>, <tkinter.Entry object .!frame2.!entry3>, <tkinter.Entry object .!frame2.!entry4>]
userEntries: [<tkinter.Entry object .!frame2.!entry>, <tkinter.Entry object .!frame2.!entry2>, <tkinter.Entry object .!frame2.!entry3>, <tkinter.Entry object .!frame2.!entry4>, <tkinter.Entry object .!frame2.!entry5>]
userEntries: [<tkinter.Entry object .!frame2.!entry>, <tkinter.Entry object .!frame2.!entry2>, <tkinter.Entry object .!frame2.!entry3>, <tkinter.Entry object .!frame2.!entry4>, <tkinter.Entry object .!frame2.!entry5>, <tkinter.Entry object .!frame2.!entry6>]
userEntries: [<tkinter.Entry object .!frame2.!entry>, <tkinter.Entry object .!frame2.!entry2>, <tkinter.Entry object .!frame2.!entry3>, <tkinter.Entry object .!frame2.!entry4>, <tkinter.Entry object .!frame2.!entry5>, <tkinter.Entry object .!frame2.!entry6>, <tkinter.Entry object .!frame2.!entry7>]
userEntries: [<tkinter.Entry object .!frame2.!entry>, <tkinter.Entry object .!frame2.!entry2>, <tkinter.Entry object .!frame2.!entry3>, <tkinter.Entry object .!frame2.!entry4>, <tkinter.Entry object .!frame2.!entry5>, <tkinter.Entry object .!frame2.!entry6>, <tkinter.Entry object .!frame2.!entry7>, <tkinter.Entry object .!frame2.!entry8>]
```

From this, it is clear that the home page is no longer visible on the Bubble Sort page. Therefore, I shall now remove all the print statements associated with the checks I have done.

I will now move onto the main functionality of the Bubble Sort page. For this, I created a method to update the chart and a method to animate bubble sort. See the code below:

2.2.g

```
def updateChart(self, numbers, swapIndices=None):
    # this is for animating the swapping of items on the bar chart
    print("in update chart")
    self.ax.clear()
    barColours = ["blue"] * len(numbers) # all bars blue
    if swapIndices: # not None
        for index in swapIndices:
            barColours[index] = "red" # swapping items in red
    self.ax.bar(range(len(numbers)), numbers, color=barColours)
    self.canvas.draw()
    self.bubbleSortFrame.update()
    time.sleep(0.3) # smooth animation
```

2.2.h

```
def bubbleSortAnimate(self):
    print("in bubbleSortAnimate")
    order = self.sortOrder.get()
    n = len(self.numbers)
    swapped = True
    print(f"Order: {order}")

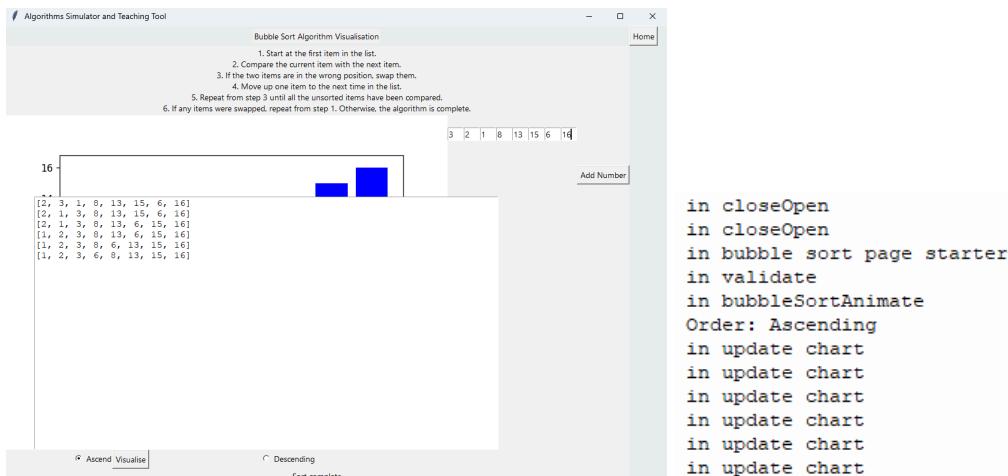
    if order == "Ascending":
        while swapped:
            swapped = False
            for j in range(n-1):
                if self.numbers[j] > self.numbers[j+1]:
                    temp = self.numbers[j]
                    self.numbers[j] = self.numbers[j+1]
                    self.numbers[j+1] = temp
                    swapped = True

                    self.visualiseText.insert(tk.END, f"{self.numbers}\n")
                    self.visualiseText.see(tk.END)
                    self.updateChart(self.numbers, [j, j+1])

    elif order == "Descending":
        while swapped:
            swapped = False
            for j in range(n-1):
                if self.numbers[j] < self.numbers[j+1]:
                    temp = self.numbers[j]
                    self.numbers[j] = self.numbers[j+1]
                    self.numbers[j+1] = temp
                    swapped = True

                    self.visualiseText.insert(tk.END, f"{self.numbers}\n")
                    self.visualiseText.see(tk.END)
                    self.updateChart(self.numbers, [j, j+1])
    self.invalidMessage["text"] = "Sort complete"
```

When the code is now run, the following are output:



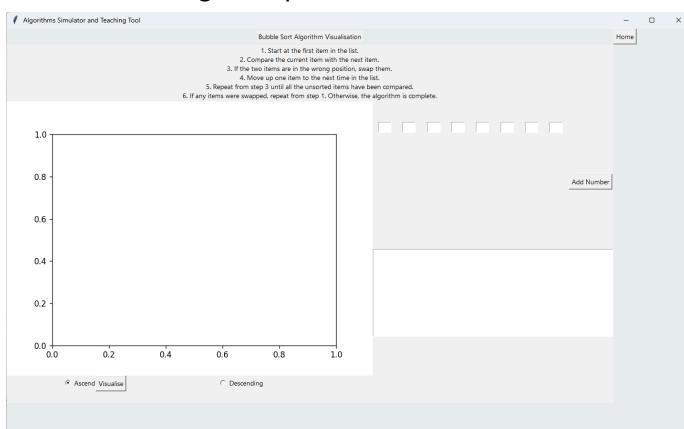
The output on the Text widget is correct however, it covers up the bar chart. To fix this, I changed the width, height and grid positioning of the visualiseText attribute:

2.2.i

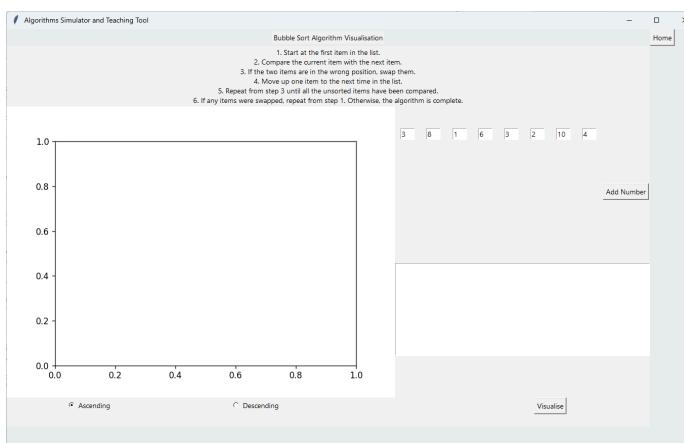
```

self.visualiseText = Text(self.bubbleSortFrame, width=50, height=10)
self.visualiseText.grid(row=6, column=7, colspan=12, rowspan=4)
    
```

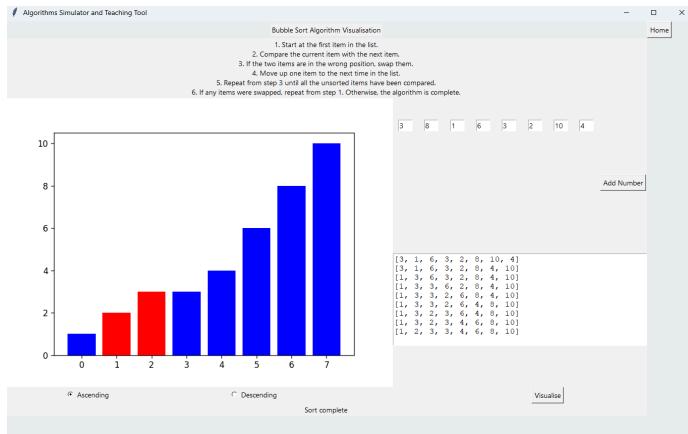
And the following is output:



Also, the visualise button is not in the intended position, which I realised was due to a typo. Once this was fixed, the following was output (with a dataset input by me):



When the Visualise button is now pressed, the following is output:



When watching the visualisation, everything seemed as expected. The screenshot above shows the final version.

When I try to run the system again, I encounter an issue with the 'Add Number' button not disappearing after 12 entry fields have been created. To fix this, move the widget part to the constructor method, along with the Frame widget for the bubbleSortFrame. This allowed me to grid_forget it (so I can use it again):

2.2.j

```
def addEntry(self):
    print("in add entry")
    if len(self.userEntries) < 11:
        entry = Entry(self.bubbleSortFrame, width=3)
        entry.grid(row=4, column=(7+len(self.userEntries)))
        self.userEntries.append(entry)
    elif len(self.userEntries) == 11:
        entry = Entry(self.bubbleSortFrame, width=3)
        entry.grid(row=4, column=(7+len(self.userEntries)))
        self.userEntries.append(entry)
    self.addButton.grid_forget()
```

2.2.k

```
class BubbleSort(HomeMain):
    def __init__(self, master, pUsername):
        #print("inside __init__ 1")
        super().__init__(master, pUsername)
        #print("inside __init__ 2")
        self.numbers = []
        self.userEntries = []

        self.bubbleSortFrame = Frame(self.master)

        self.addButton = Button(self.bubbleSortFrame, text="Add Number", command=self.addEntry)
        self.sortOrder = StringVar(value="Ascending")

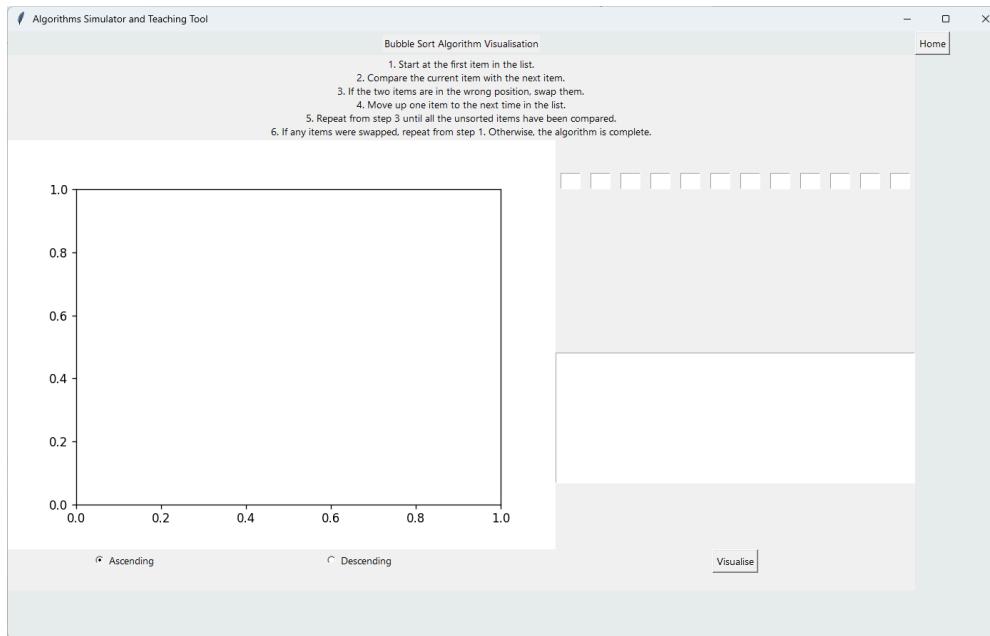
        self.invalidMessage = None

        self.fig = None
        self.ax = None
        self.canvas = None

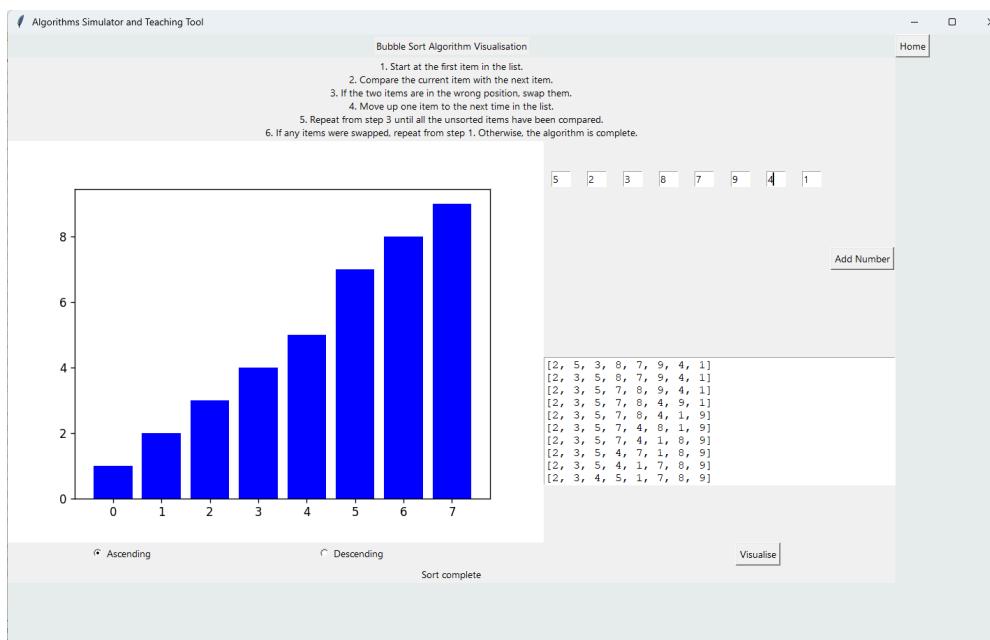
        self.visualiseText = None

        #print(self.numbers)
        #print(self.userEntries)
        self.pageStarterBubbleSort()
```

The output is the following when the 'Add Number' button has been clicked 4 times:



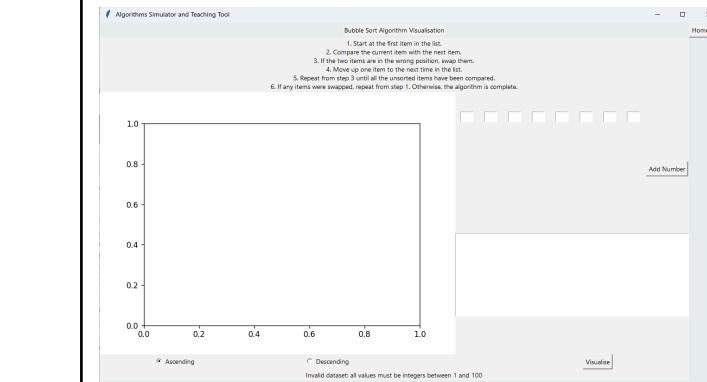
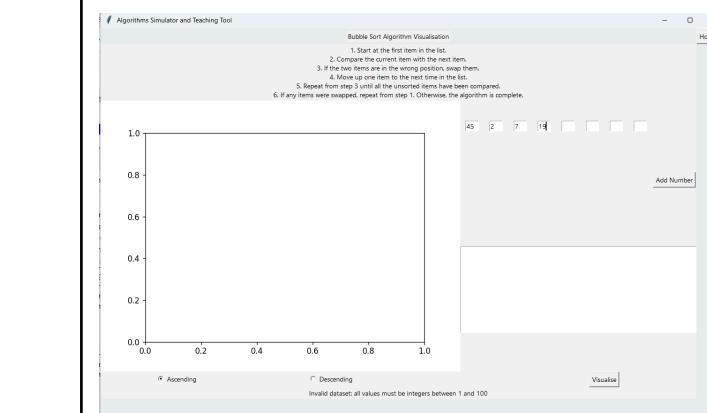
Now when I run the code, the following is output at the end:

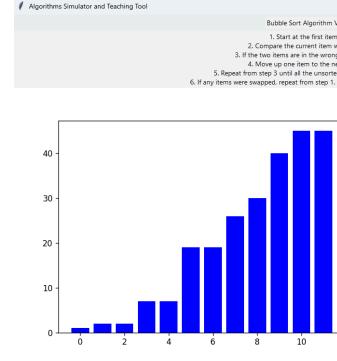
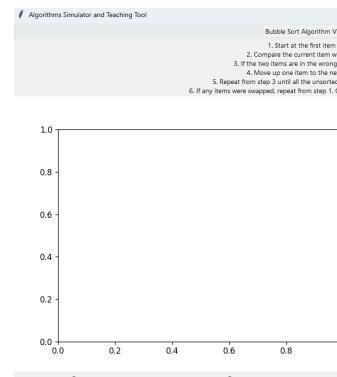


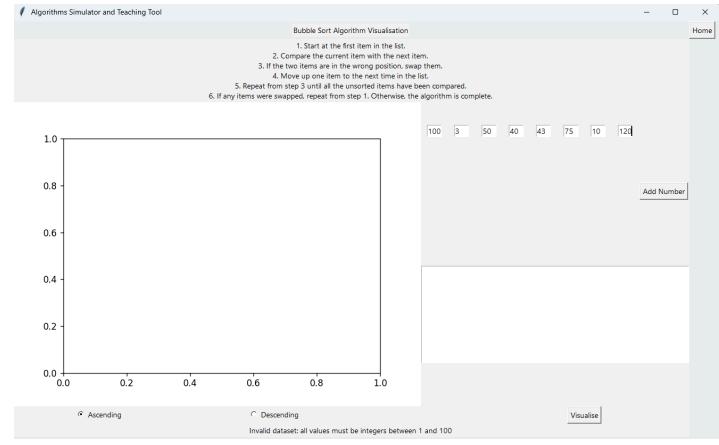
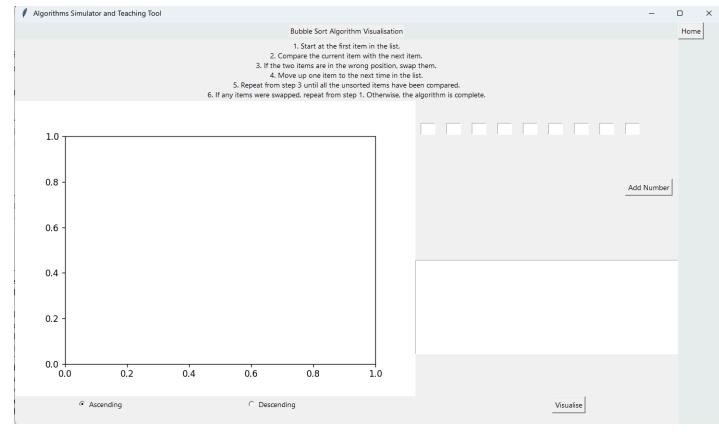
And all the visualisation, both in numbers and on the chart is correct. However, animations are not occurring in the text based version.

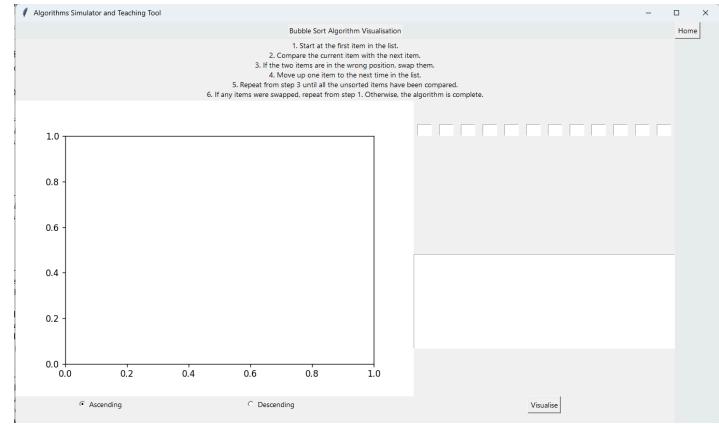
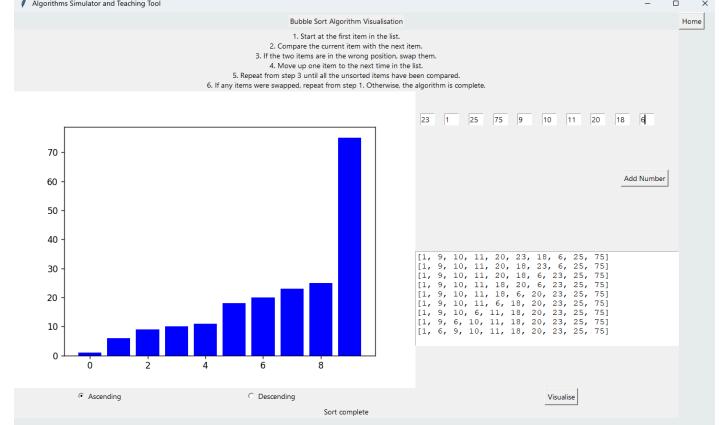
Considering the time constraints that I have and the other algorithms that I want to implement visualisations for, I think that the version I have with no animations on the text is satisfactory. Swaps are still shown and users are able to go through them by scrolling the Text section - which I adjusted to allow more lines to be shown.

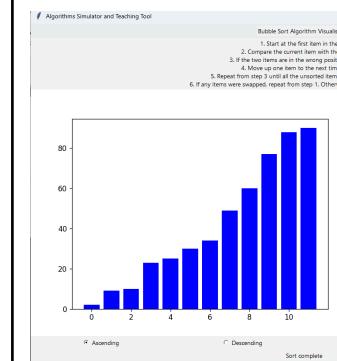
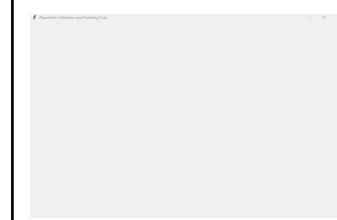
I will now test the system against the white box testing table specified in the BUBBLE SORT VISUALISATION PAGE design section.

TEST DATA	EXPECTED RESULT	JUSTIFICATION	CODE	ACTUAL OUTPUT
Visualise without entering a dataset	Invalid dataset message is displayed.	This ensures that the user must enter a set of numbers.	2.2.a 2.2.b 2.2.d	 <p>As expected</p>
Visualise with a dataset of length less than 8	Invalid dataset message is displayed.	This ensures that the testing of the length of the dataset is correct - so the dataset has >7 entries.	2.2.a 2.2.b 2.2.d	 <p>As expected</p>

Visualise with a dataset of length 8	Bubble Sort visualisation occurs correctly to sort the data both in text and on the bar chart.	This ensures that the testing of the length works correctly so that the minimum length is 8.	2.2.a 2.2.b 2.2.d 2.2.g 2.2.h	 <p>The screenshot shows a bar chart with x-axis values from 0 to 10 and y-axis values from 0 to 40. The bars represent the numbers 1 through 45 in ascending order. Below the chart, there are two radio buttons: 'Ascending' (selected) and 'Descending'. At the bottom right, a button labeled 'Sort complete' is visible.</p> <p>Bubble Sort Algorithm Visualisation</p> <ol style="list-style-type: none"> Start at the first item in the list. Compare the current item with the next item. If the two items are in the wrong order, swap them. Move up one item to the next time in the list. Repeat from step 3 until all the unordered items have been compared. If any items were swapped, repeat from step 1. Otherwise, the algorithm is complete. <p>45 2 7 19 30 26 1 45</p> <p>Add Number Visualise Sort complete</p>	<p>Output is correct but the Invalid message from the previous test did not disappear until it changed to 'Sort complete'. Additionally, the numbers on the x axis are rather confusing as they do not match the numbers to be sorted.</p>
Visualise with a dataset with values that are not integers	Invalid dataset message is displayed.	This ensures that the testing checks that all entries are integers.	2.2.a 2.2.b 2.2.d	 <p>The screenshot shows an empty bar chart with x-axis values from 0.0 to 1.0 and y-axis values from 0.0 to 1.0. Below the chart, there are two radio buttons: 'Ascending' and 'Descending'. At the bottom right, a button labeled 'Visualise' is visible. A message at the bottom states: "Invalid dataset: all values must be integers between 1 and 100".</p> <p>Bubble Sort Algorithm Visualisation</p> <ol style="list-style-type: none"> Start at the first item in the list. Compare the current item with the next item. If the two items are in the wrong order, swap them. Move up one item to the next time in the list. Repeat from step 3 until all the unordered items have been compared. If any items were swapped, repeat from step 1. Otherwise, the algorithm is complete. <p>A 34 1 40 30 23 c 9</p> <p>Add Number Visualise Invalid dataset: all values must be integers between 1 and 100</p>	<p>As expected</p>

Visualise with a dataset with values not in the range 1-100	Invalid dataset message is displayed.	This ensures that the testing checks that all entries are in the correct range of values.	2.2.a 2.2.b 2.2.d	 <p>The screenshot shows a window titled 'Algorithms Simulator and Teaching Tool' with a sub-section for 'Bubble Sort Algorithm Visualisation'. It displays a list of numbers: 100, 3, 50, 40, 43, 75, 10, 12. Below the list is a note: 'Invalid dataset: all values must be integers between 1 and 100'. At the bottom are buttons for 'Visualise' and 'Add Number'.</p> <p>As expected</p>
Press the 'Add Entry' button	An entry field is added on the line with the rest of the entry fields. If there are already 11 entry fields, the button disappears.	This ensures that the user can enter a dataset of length greater than 8.	2.2.j	 <p>The screenshot shows a window titled 'Algorithms Simulator and Teaching Tool' with a sub-section for 'Bubble Sort Algorithm Visualisation'. It displays a list of 11 empty entry fields. Below the list is a note: 'Invalid dataset: all values must be integers between 1 and 100'. At the bottom are buttons for 'Visualise' and 'Add Number'.</p>

				 <p>As expected when originally it had 8 numbers and when it originally had 11 numbers.</p>
Visualise with a dataset of length >8	Bubble Sort visualisation occurs correctly to sort the data both in text and on the bar chart.	This ensures that the testing of the length of the dataset is correct and allows lengths greater than 8 to be visualised.	2.2.a 2.2.b 2.2.d 2.2.g 2.2.h 2.2.j	 <p>As expected.</p>

Visualise with a dataset of length 12	Bubble Sort visualisation occurs correctly to sort the data both in text and on the bar chart.	This ensures that the maximum length of 12 is included in the range of valid values for the length of the dataset.	2.2.a 2.2.b 2.2.d 2.2.g 2.2.h 2.2.j	 <p>The screenshot shows a bar chart with data points at indices 0 to 11. The y-axis ranges from 0 to 80. The bars show values increasing from index 0 to 11. Below the chart is a list of 12 numbers: 19, 10, 25, 23, 30, 34, 49, 2, 60, 77, 89, 901. To the right of the list is a 'Visualise' button.</p> <p>As expected.</p>
Press the 'Home' button	The Bubble Sort Visualisation page is closed and the Home page is opened.	This ensures that the 'Home' button works correctly and that the user can go back to the Home page to either log out or go to the other visualisation/quiz pages.	2.2.l	 <pre data-bbox="1304 884 2023 1178"> in closeOpen Exception in Tkinter callback Traceback (most recent call last): File "C:\Users\lingri\AppData\Local\Programs\Thonny\lib\tkinter__init__.py", line 192 in __call__ return self.func(*args) File "C:\Users\lingri\OneDrive\Documents\QM\QM School work\A Level\Computer Science\NEA\NEA main file.py", line 77, in openHome closeOpen(self.master, "home") File "C:\Users\lingri\OneDrive\Documents\QM\QM School work\A Level\Computer Science\NEA\NEA main file.py", line 44, in closeOpen running = HomeMain(homeRoot, username) File "C:\Users\lingri\OneDrive\Documents\QM\QM School work\A Level\Computer Science\NEA\NEA main file.py", line 422, in __init__ self.accountType = self.accountTypeGet() File "C:\Users\lingri\OneDrive\Documents\QM\QM School work\A Level\Computer Science\NEA\NEA main file.py", line 457, in accountTypeGet accountType = result[0] TypeError: 'NoneType' object is not subscriptable </pre> <p>Error - haven't passed the username as a parameter into the closeOpen function.</p>

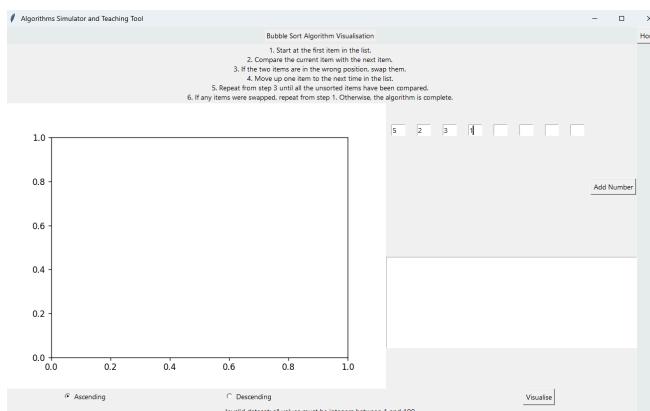
From these tests, I have identified 2 main issues to fix in this iteration. The first of these is the ‘Invalid dataset’ message not disappearing when I run the Visualisation after fixing the erroneous dataset input. I fixed this with the code below when I run the bubbleSortAnimate method:

2.2.k

```
def bubbleSortAnimate(self):
    print("in bubbleSortAnimate")
    order = self.sortOrder.get()
    n = len(self.numbers)
    swapped = True
    print(f"Order: {order}")

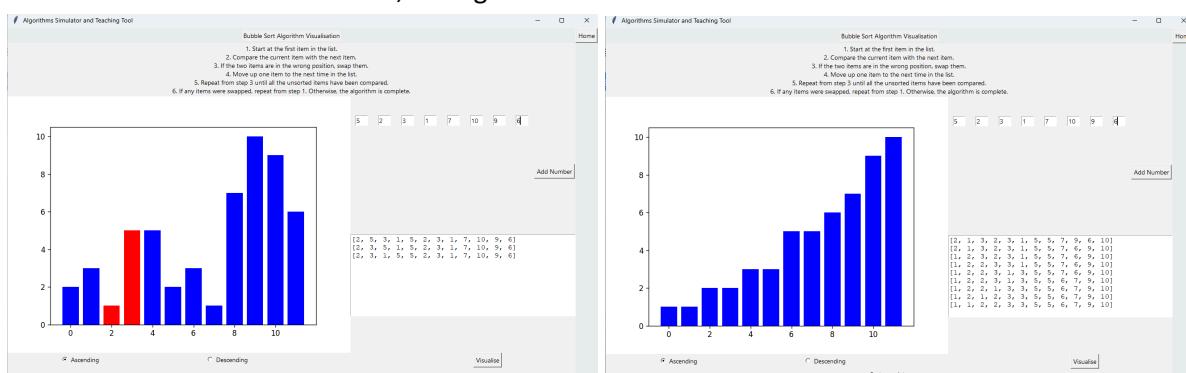
    self.invalidMessage["text"] = ""
```

When I now run the code, the following is the output from an erroneous dataset input:



And the following is the output when I run the Visualisation after fixing the errors:

- Show screenshot of error, during visualisation and after visualisation



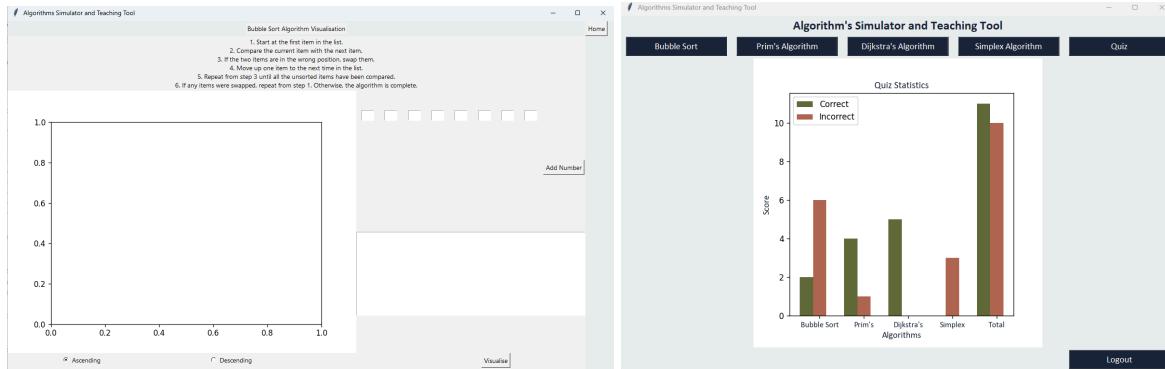
This has fixed the error so the ‘Visualise with a dataset of length 8’ test now passes - the issues with the x axis shall be fixed in the next iteration where the GUI is refined.

The second error I have identified from these tests is with pressing the ‘Home’ button. A NoneType error occurred, which was due to me not passing the username as a parameter when calling the closeOpen function. I fixed this with the following line of code:

2.2.l

```
def openHome(self):
    # inherited self.currentAccUsername from HomeMain
    closeOpen(self.master, "home", self.currentAccUsername)
```

And when I now run the code, the following are output:



The errors identified from this white box testing have now been fixed successfully.

ITERATION 2 - DEVELOPER REVIEW

Overall, development in this iteration went rather smoothly. The main issue that I encountered with the no attribute error challenged me with debugging my code. After thinking about how to solve the issue for some time, I put in print statements to help me work out the flow of the code. This helped me work out that the constructor method for the BubbleSort class was not being reentered correctly. The experience of solving these issues will be helpful in how I tackle the other algorithm visualisations as I will be sure to check that all methods are being entered correctly.

ITERATION 3 - REFINING THE BUBBLE SORT GUI AND DEVELOPING THE MENU

In this iteration, I will begin by developing the menu. As I want this code to be reusable (so that I can use it for every algorithm visualisation), I created a new class called Menu, which each algorithm visualisation shall inherit from:

2.3.a

```
616 class Menu:
617     def __init__(self, page, rowCoord, columnsLength):
618         self.visualisationPage = page
619         self.currentStep = 0
620         self.isPaused = True
621         self.steps = []
622         self.menuWidgets()
623         self.rowStart = rowCoord # this is the y coordinate of where the menu starts for the current page
624         self.menuLength = columnsLength # this is the rowspan of the menu based on the current page
625
626     def menuWidgets(self):
627         menuFrame = Frame(self.visualisationPage)
628         menuFrame.grid(row=self.rowStart, column = 0, columnspan = self.menuLength, rowspan=3)
629
630         Button(menuFrame, text="Skip Back", command=self.skipBack)
631         self.playPauseButton = Button(menuFrame, text="Pause", command=self.togglePlayPause)
632         Button(menuFrame, text="Skip Forward", command=self.skipForward)
633
634         # these are for the keyboard inputs:
635         self.visualisationPage.bind("<Left>", lambda e: self.stepBack())
636         self.visualisationPage.bind("<Right>", lambda e: self.stepForward())
```

And each Button will have its own method for its functionality:

2.3.b

```

639     def skipBack(self):
640         self.currentStep = 0
641         self.isPaused = True
642         if hasattr(self, "updateChart"): # note that methods count as attributes
643             self.updateChart(self.steps[self.currentStep])
644         # extend once implemented other algorithms
645
646     def skipForward(self):
647         self.currentStep = len(self.steps) - 1 # gets the last step index
648         self.isPaused = True
649         if hasattr(self, "updateChart"): # note that methods count as attributes
650             self.updateChart(self.steps[self.currentStep])
651         # extend once implemented other algorithms

```

2.3.c

```

653     def togglePlayPause(self):
654         self.isPaused = not self.isPaused # switches the state
655         self.playPauseButton.config(text="Pause" if not self.isPaused else "Play") # switches the text
656         if not self.isPaused: # ie displays Pause but the visualisation is playing
657             self.playAnimation()
658
659     def stepBack(self):
660         if self.isPaused and self.currentStep > 0:
661             self.currentStep -= 1
662             if hasattr(self, "updateChart"): # note that methods count as attributes
663                 self.updateChart(self.steps[self.currentStep])
664             # extend once implemented other algorithms
665
666     def stepForward(self):
667         if self.isPaused and self.currentStep < len(self.steps)-1:
668             self.currentStep += 1
669             if hasattr(self, "updateChart"): # note that methods count as attributes
670                 self.updateChart(self.steps[self.currentStep])
671             # extend once implemented other algorithms

```

2.3.d

And I added a method for the actual functionality of playing the animation of the visualisation:

```

673     def playAnimation(self):
674         while not self.isPaused and self.currentStep < len(self.steps)-1:
675             self.currentStep += 1
676             if hasattr(self, "updateChart"): # note that methods count as attributes
677                 self.updateChart(self.steps[self.currentStep])
678             # extend once implemented other algorithms
679             self.visualisationPage.update()
680             time.sleep(0.3)
681             self.isPaused = True
682             self.playPauseButton.config(text="Play")

```

From this code, it is clear that I have changed the way the visualisation is being animated and displayed from what I had developed for the BubbleSort class. I changed it so that each step of the visualisation is stored in the attribute steps, which allows the user to go back and forth between them. Therefore, I changed the functionality of the bubbleSortAnimate and updateChart methods in the BubbleSort class so that they worked with the way I have developed the menu:

2.3.e

```

687 class BubbleSort(HomeMain, Menu):
688     def __init__(self, master, pUsername):
689         #print("inside __init__ 1")
690         HomeMain.__init__(self, master, pUsername)
691         #print("inside __init__ 2")
692         Menu.__init__(self, master, 12, 19)
693         self.numbers = []
694         self.userEntries = []
695
696         self.bubbleSortFrame = Frame(self.master)
697
698         self.addButton = Button(self.bubbleSortFrame, text="Add Number", command=self.addEntry)
699         self.sortOrder = StringVar(value="Ascending")
700
701         self.invalidMessage = None
702
703         self.fig = None
704         self.ax = None
705         self.canvas = None
706
707         self.visualiseText = None
708
709         #print(self.numbers)
710         #print(self.userEntries)
711         self.pageStarterBubbleSort()

```

2.3.f

```

810     def bubbleSortAnimate(self):
811         print("in bubbleSortAnimate")
812         order = self.sortOrder.get()
813         n = len(self.numbers)
814         swapped = True
815         print(f"Order: {order}")
816
817         self.steps = [self.numbers[:]] # creates copy of current list of numbers
818
819         self.invalidMessage["text"] = ""
820
821         if order == "Ascending":
822             while swapped:
823                 swapped = False
824                 for j in range(n-1):
825                     if self.numbers[j] > self.numbers[j+1]:
826                         temp = self.numbers[j]
827                         self.numbers[j] = self.numbers[j+1]
828                         self.numbers[j+1] = temp
829                         swapped = True
830
831                         self.visualiseText.insert(tk.END, f"{self.numbers}\n")
832                         self.visualiseText.see(tk.END)
833                         self.updateChart(self.numbers, [j, j+1])
834
835         self.steps.append(self.numbers[:])

```

2.3.g

```

837     elif order == "Descending":
838         while swapped:
839             swapped = False
840             for j in range(n-1):
841                 if self.numbers[j] < self.numbers[j+1]:
842                     temp = self.numbers[j]
843                     self.numbers[j] = self.numbers[j+1]
844                     self.numbers[j+1] = temp
845                     swapped = True
846
847                     self.visualiseText.insert(tk.END, f"{self.numbers}\n")
848                     self.visualiseText.see(tk.END)
849                     self.updateChart(self.numbers, [j, j+1])
850
851             self.steps.append(self.numbers[:])
852
853         self.invalidMessage["text"] = "Sort complete"
854         self.updateChart(self.numbers)

```

2.3.h

```

775     def updateChart(self, numbers, swapIndices=None):
776         # this is for animating the swapping of items on the bar chart
777         print("in update chart")
778         self.ax.clear()
779         barColours = ["blue"] * len(numbers) # all bars blue
780         if swapIndices: # not None
781             for index in swapIndices:
782                 barColours[index] = "red" # swapping items in red
783             self.ax.bar(range(len(numbers)), numbers, color=barColours)
784             self.canvas.draw()
785             self.bubbleSortFrame.update()
786             #time.sleep(0.3) # smooth animation

```

The changes I made were to take out the sleep line in the updateChart method and to add lines to append the current set of numbers to the steps attribute. When the code is now run, the following are output:

```

in closeOpen
in closeOpen
Exception in Tkinter callback
Traceback (most recent call last):
  File "C:\Users\ingri\AppData\Local\Programs\Thonny\lib\tkinter\__init__.py", line 1921, in call
    return self.func(*args)
  File "C:\Users\ingri\OneDrive\Documents\QM\QM School work\A Level\Computer Science\NEA\NEA main file.py", line 588, in openBubbleSort
    closeOpen(self.master, "bubble sort", usernameHolder)
  File "C:\Users\ingri\OneDrive\Documents\QM\QM School work\A Level\Computer Science\NEA\NEA main file.py", line 58, in closeOpen
    running = BubbleSort(bubbleSortRoot, username)
  File "C:\Users\ingri\OneDrive\Documents\QM\QM School work\A Level\Computer Science\NEA\NEA main file.py", line 692, in __init__
    Menu.__init__(self, master, 12, 19)
  File "C:\Users\ingri\OneDrive\Documents\QM\QM School work\A Level\Computer Science\NEA\NEA main file.py", line 622, in __init__
    self.menuWidgets()
  File "C:\Users\ingri\OneDrive\Documents\QM\QM School work\A Level\Computer Science\NEA\NEA main file.py", line 629, in menuWidgets
    self.menuFrame.grid(row=self.rowStart, column = 0, columnspan = self.menuLength, rowspan=3)
AttributeError: 'BubbleSort' object has no attribute 'rowStart'

```

I realised that this was due to me placing the menuWidgets line before the lines initialising the other attributes:

2.3.i

```

def __init__(self, page, rowCoord, columnsLength):
    self.visualisationPage = page
    self.currentStep = 0
    self.isPaused = True
    self.steps = []
    self.rowStart = rowCoord # this is the y coordinate of where the menu starts for the current page
    self.menuLength = columnsLength # this is the rowspan of the menu based on the current page
    self.menuFrame = None

    self.menuWidgets()

```

Additionally, I realised that I hadn't given the Buttons grid locations as I was not sure where to place them. To fix this, I changed them to the following:

2.3.j

```

def menuWidgets(self):
    self.menuFrame = Frame(self.visualisationPage)
    self.menuFrame.grid(row=self.rowStart, column = 0, columnspan = self.menuLength, rowspan=3)

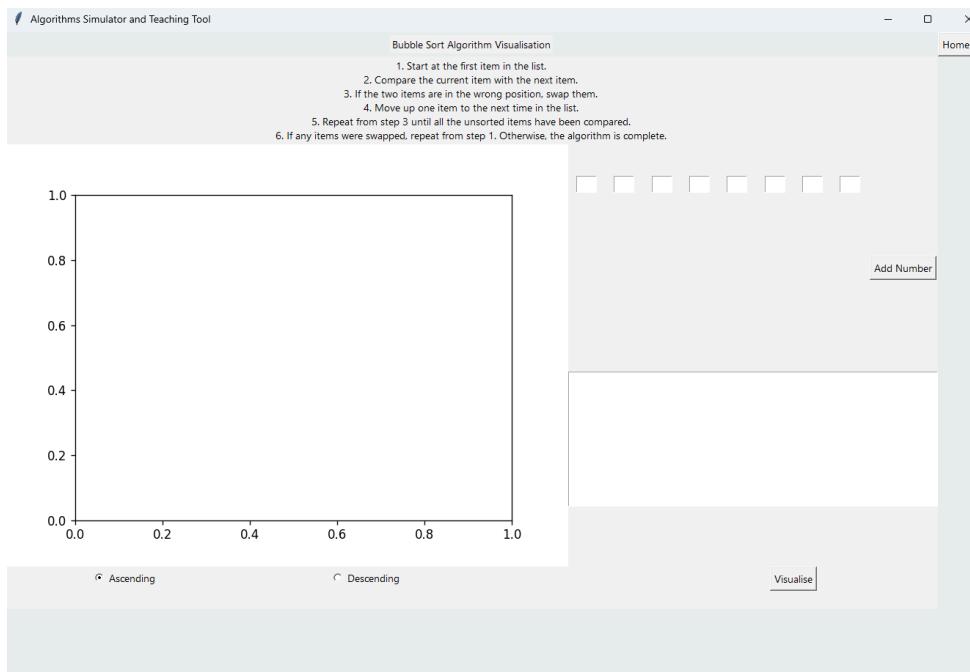
    Button(self.menuFrame, text="Skip Back", command=self.skipBack).grid(row=(self.rowStart+1), column= 1)
    self.playPauseButton = Button(self.menuFrame, text="Pause", command=self.togglePlayPause)
    self.playPauseButton.grid(row=(self.rowStart+1), column= 3)
    Button(self.menuFrame, text="Skip Forward", command=self.skipForward).grid(row=(self.rowStart+1), column=5)

    # add text describing arrow stuff

    # these are for the keyboard inputs:
    self.visualisationPage.bind("<Left>", lambda e: self.stepBack())
    self.visualisationPage.bind("<Right>", lambda e: self.stepForward())

```

When I now run the code, the following is output:



The buttons are not being output, which I think may be due to the grid spacing being incorrect or the menuWidgets method not being entered correctly. Therefore, I added the following code:

2.3.k

```
def bubbleSortWidgets(self):
    # TITLE / TOP SECTION:
    self.title = Label(self.master, text="Bubble Sort Algorithm Visualisation")
    self.title.grid(row=0, column=0, columnspan=17)

    Button(self.master, text="Home", command=self.openHome).grid(row=0, column=17, columnspan=2)

    # BUBBLE SORT FRAME:
    self.bubbleSortFrame.grid(row=1, column=0, columnspan=19, rowspan=11)
```

2.3.l

```
class Menu:
    def __init__(self, master, rowCoord, columnsLength):
        self.master = master
        self.currentStep = 0
        self.isPaused = True
        self.steps = []
        self.rowStart = rowCoord # this is the y coordinate of where the menu starts for the current page
        self.menuLength = columnsLength # this is the rowspan of the menu based on the current page
        self.menuFrame = None

        self.menuWidgets()

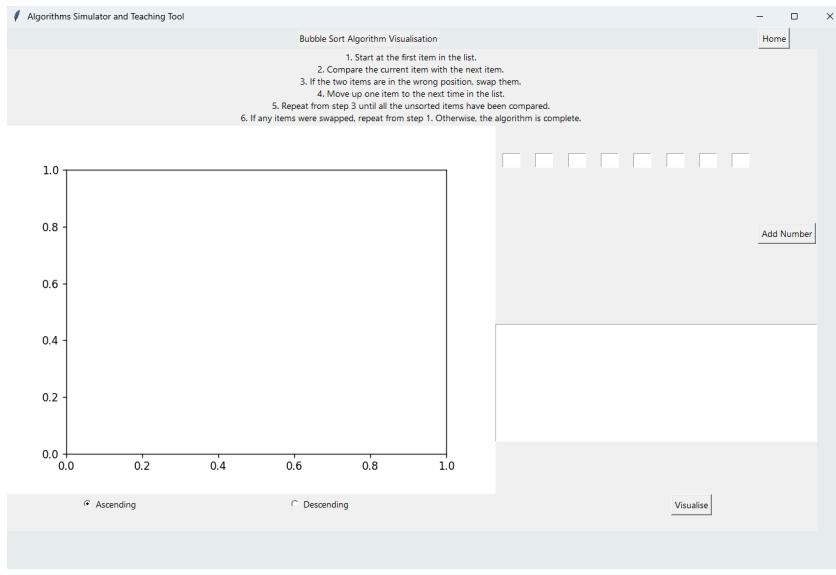
    def menuWidgets(self):
        print("in menuWidgets")
        self.menuFrame = Frame(self.master)
        self.menuFrame.grid(row=self.rowStart, column = 0, columnspan = self.menuLength, rowspan=3)

        Button(self.menuFrame, text="Skip Back", command=self.skipBack).grid(row=(self.rowStart+1), column= 1)
        self.playPauseButton = Button(self.menuFrame, text="Pause", command=self.togglePlayPause)
        self.playPauseButton.grid(row=(self.rowStart+1), column= 3)
        Button(self.menuFrame, text="Skip Forward", command=self.skipForward).grid(row=(self.rowStart+1), column=5)

        # add text describing arrow stuff

        # these are for the keyboard inputs:
        self.master.bind("<Left>", lambda e: self.stepBack())
        self.master.bind("<Right>", lambda e: self.stepForward())
```

Which displays the following when run:



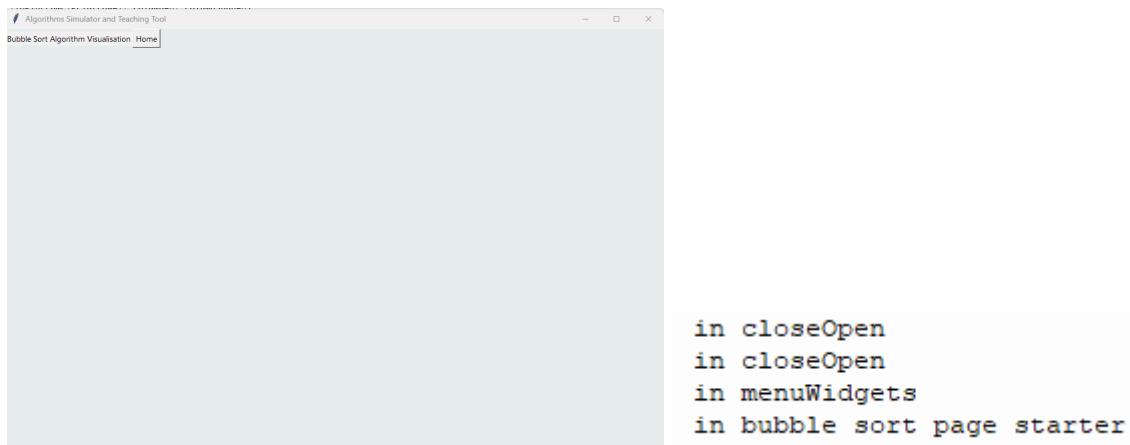
```
in closeOpen
in closeOpen
in menuWidgets
in bubble sort page starter
```

I tried to debug the code by changing the background colour of the menuFrame and by forgetting all the widgets in the bubbleSortFrame.

2.3.m

```
def pageStarterBubbleSort(self):
    print("in bubble sort page starter")
    #print(f"numbers: {self.numbers}")
    #print(f"userEntries: {self.userEntries}")
    self.hideHomeWidgets()
    self.bubbleSortWidgets()
    # testing for menu:
    for widget in self.bubbleSortFrame.winfo_children():
        widget.grid_forget()
    self.bubbleSortFrame.grid_forget()
```

However, this was output:

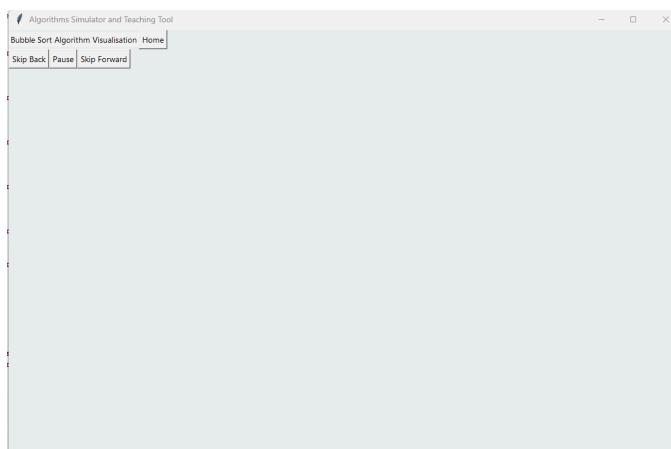


I was not sure where these issues were coming from. After looking through the code, I realised that the issues may have been arising from the `hideHomeWidgets` method because this was hiding all the widgets in master. Therefore, I changed the code in that method to the following:

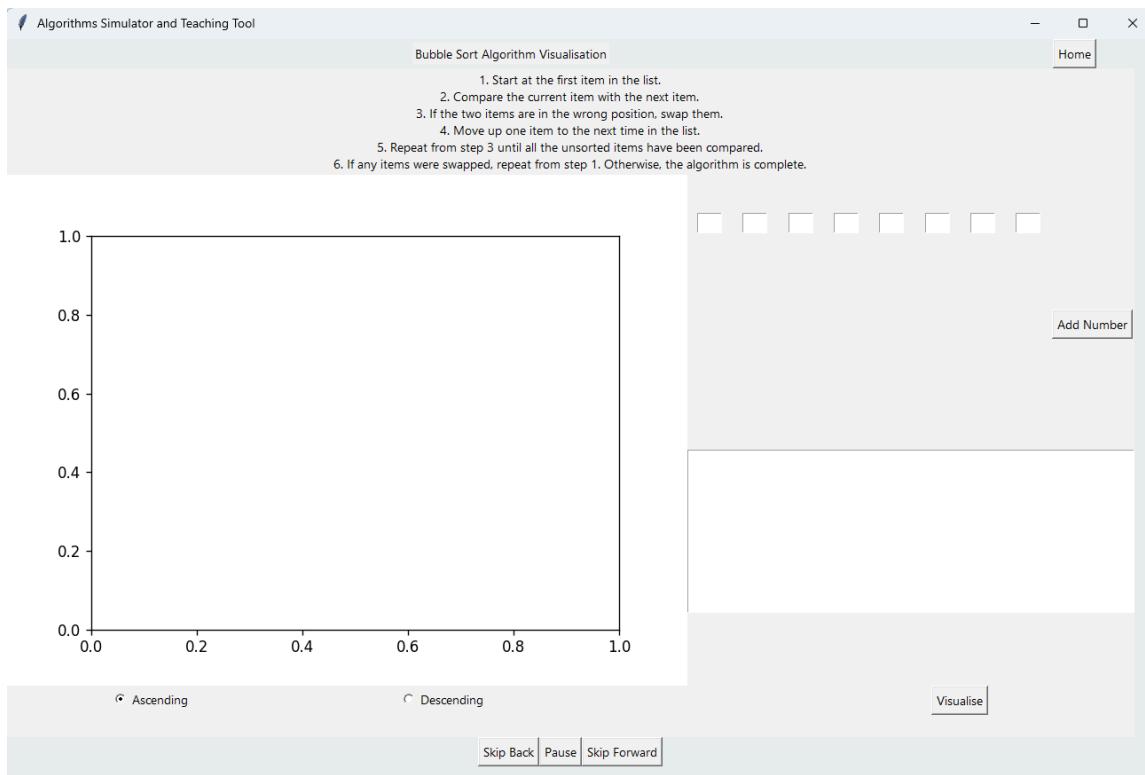
2.3.n

```
def hideHomeWidgets(self):
    for widget in self.homeFrame.winfo_children():
        widget.grid_forget()
    self.title.grid_forget()
```

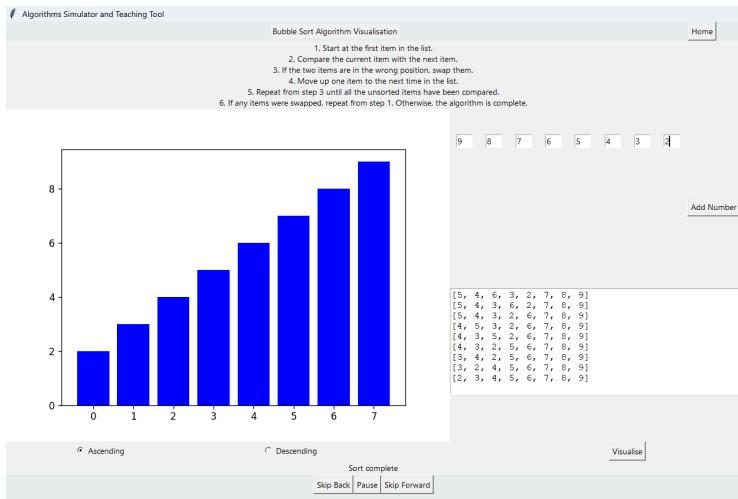
This displayed the following when run:



The buttons are now displaying so I removed the forgetting of the `bubbleSortFrame` widgets and I changed the grid back to what I had it before. When the code is now run, the following is output:

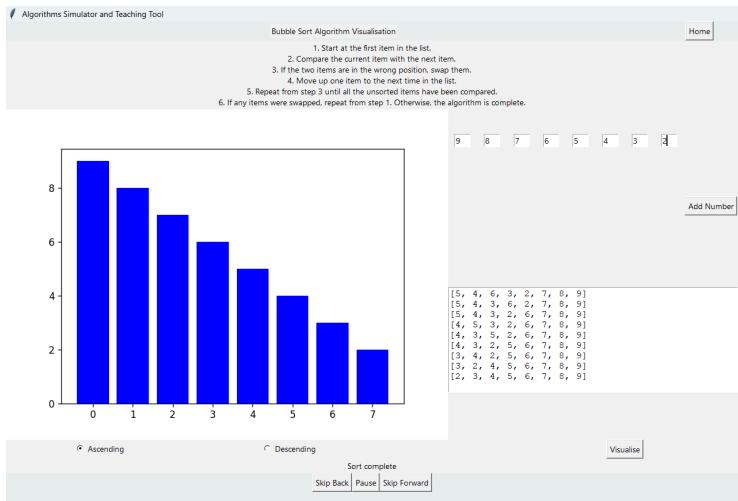


When I run the visualisation now, the following is output once the visualisation is complete:



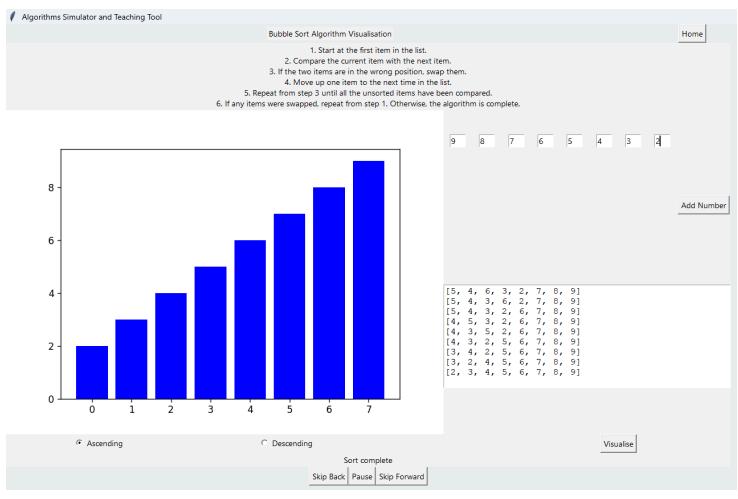
I noticed when the visualisation was

running that it was very fast. Therefore, I increased the time for when the code is in sleep so that I can easily test the Pause functionality. When I press the Skip Back button, the following is displayed:



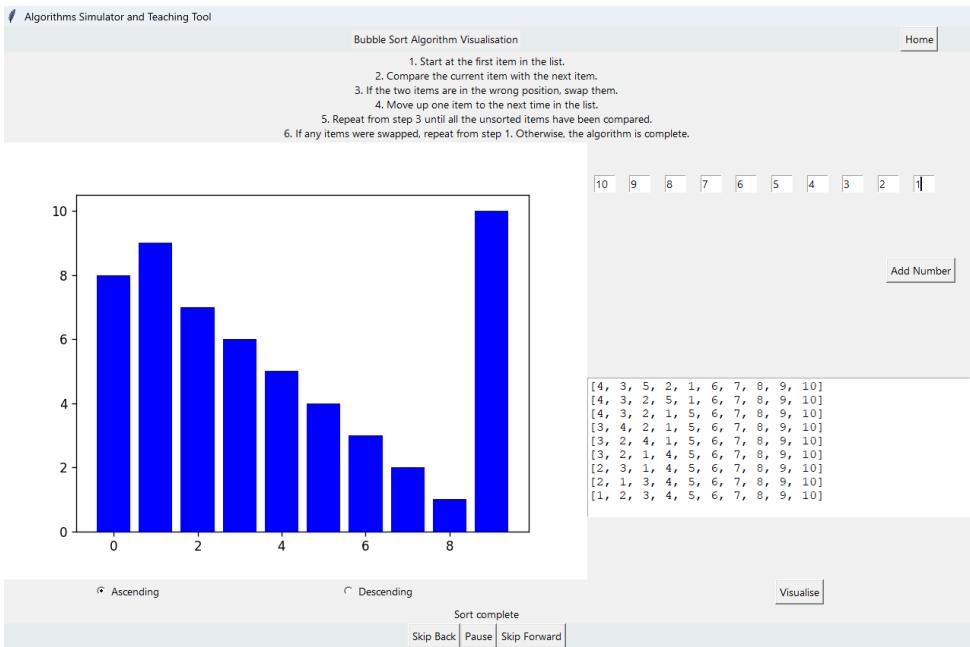
This is as expected but the Sort

complete message is still displayed. When I then press the Skip Forward button, the following is displayed:

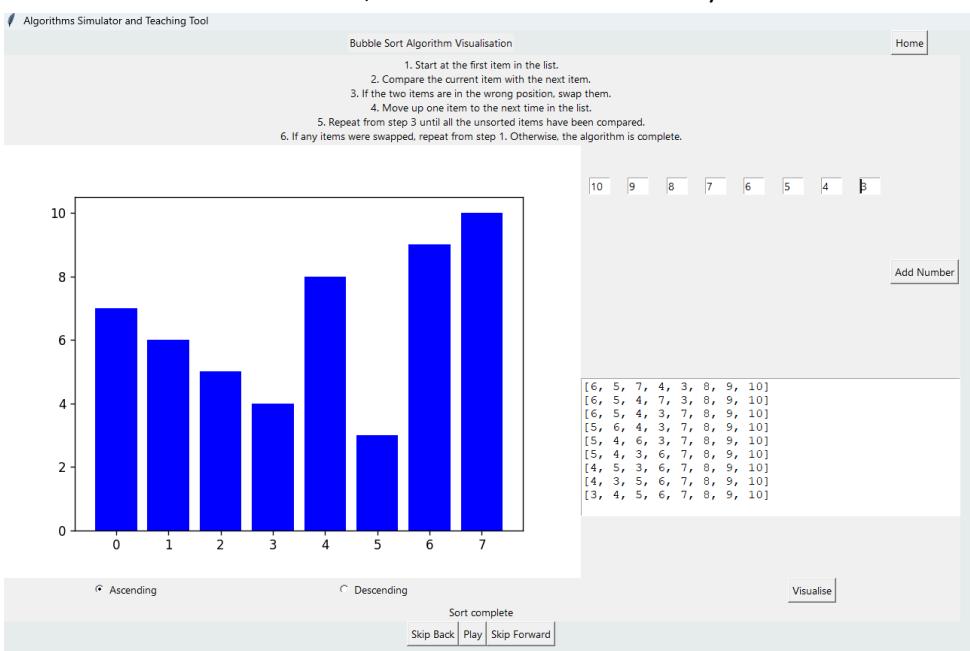


This is again as expected.

After increasing the sleep time, I tested the code by running a visualisation and pausing it.



I also tested the arrow buttons, which both worked correctly.



I noticed that the visualisation was still very fast and when I pressed the Pause button whilst it was running, the visualisation would go back to a step that it had already done - likely the step when I had paused it. Therefore, I slowed down the visualisation more. This slowing down of the visualisation made it so that when I press visualise, the algorithm runs through the whole visualisation very quickly, but, when it is paused and I press play, the visualisation is slow like expected. I realised that this may have been due to me removing the lines to sleep in the bubbleSortAnimate method, which I added again:

2.3.o

```

if order == "Ascending":
    while swapped:
        swapped = False
        for j in range(n-1):
            if self.numbers[j] > self.numbers[j+1]:
                temp = self.numbers[j]
                self.numbers[j] = self.numbers[j+1]
                self.numbers[j+1] = temp
                swapped = True

            self.visualiseText.insert(tk.END, f"{self.numbers}\n")
            self.visualiseText.see(tk.END)
            self.updateChart(self.numbers, [j, j+1])

        self.steps.append(self.numbers[:])
        time.sleep(0.75)

elif order == "Descending":
    while swapped:
        swapped = False
        for j in range(n-1):
            if self.numbers[j] < self.numbers[j+1]:
                temp = self.numbers[j]
                self.numbers[j] = self.numbers[j+1]
                self.numbers[j+1] = temp
                swapped = True

            self.visualiseText.insert(tk.END, f"{self.numbers}\n")
            self.visualiseText.see(tk.END)
            self.updateChart(self.numbers, [j, j+1])

        self.steps.append(self.numbers[:])
        time.sleep(0.75)

```

When I run the visualisation, it seems that there is some lag and that when I press the pause button, the visualisation is not paused and keeps running but I still have the ability to use my arrow keys.

After doing some research, I found out that the error from the visualisation kept going was due to an infinite loop stopping Tkinter from processing the events properly. To fix this, I changed the while loop to if statements and use after(). The code is now the following:

2.3.p

```

if not self.isPaused and self.currentStep < len(self.steps)-1:
    self.currentStep += 1
    if hasattr(self, "updateChart"):
        self.updateChart(self.steps[self.currentStep])
    # extend for other algorithms
    self.master.after(750, self.playAnimation) # call itself recursively after 750ms
else:
    self.isPaused = True
    self.pauseButton.config(text="Play")

```

It seems that as this code now checks the isPaused attribute first, this should cause the play/pause button to toggle as soon as it is pressed. However, when I run the code, the visualisation begins as normal and when I press the pause button, it changes correctly as soon as I press it. But, the visualisation keeps going.

I again researched what may be causing this issue and found that the bubbleSortAnimate method may be causing it. Therefore, I changed the code from a while loop to use after() so that the steps are scheduled and all checks are complete between each swap:

2.3.q

```

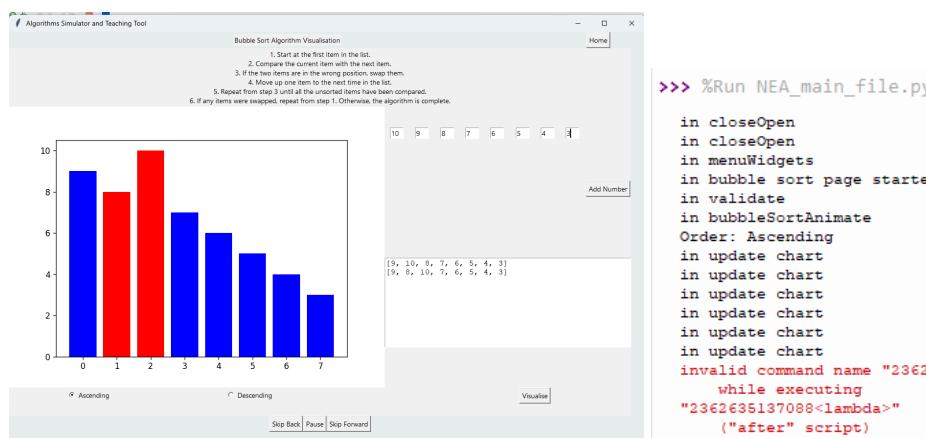
def bubbleSortAnimate(self):
    print("in bubbleSortAnimate")
    order = self.sortOrder.get()
    n = len(self.numbers)
    swapped = True
    print(f"Order: {order}")

    self.steps = [self.numbers[:]] # creates copy of current list of numbers
    self.invalidMessage["text"] = ""

    def sortStep(i, j):
        nonlocal swapped
        if i < n-1:
            if j < n-i-1:
                if order == "Ascending":
                    if self.numbers[j] > self.numbers[j+1]:
                        temp = self.numbers[j]
                        self.numbers[j] = self.numbers[j+1]
                        self.numbers[j+1] = temp
                        swapped = True
                    self.steps.append(self.numbers[:]) # append copy of list
                    self.updateChart(self.numbers, [j, j+1]) # highlights the swap
                    self.visualiseText.insert(tk.END, f"{self.numbers}\n")
                    self.visualiseText.see(tk.END)

                elif order == "Descending":
                    if self.numbers[j] < self.numbers[j+1]:
                        temp = self.numbers[j]
                        self.numbers[j] = self.numbers[j+1]
                        self.numbers[j+1] = temp
                        swapped = True
                    self.steps.append(self.numbers[:]) # append copy of list
                    self.visualiseText.insert(tk.END, f"{self.numbers}\n") # highlights the swap
                    self.visualiseText.see(tk.END)
                    self.updateChart(self.numbers, [j, j+1])
                # schedule the next step:
                self.master.after(750, lambda: sortStep(i, j+1))
            else:
                if swapped:
                    # move to next step
                    self.master.after(750, lambda: sortStep(i+1, 0))
                else:
                    # sort is complete
                    self.invalidMessage["text"] = "Sort Complete"
                    self.updateChart(self.numbers)
        sortStep(0,0)
    
```

When I run the code now:



The visualisation seems to stop at the second check and it does not continue.

I was not sure where these issues were coming from so I have decided to try and rework the logic of the menu and how I was approaching it. To do this, I created a new file that had only the Menu and

BubbleSort classes. Then, I copied some of the code for the widgets and the functionality for each class. As I have created a new file and am not focused on the entering of the dataset, I set the dataset to be the numbers 1 to 10 in descending order. Below is the code for the Menu that I had copied:

2.3.r

```

1 from tkinter import *
2 import tkinter as tk
3 import matplotlib.pyplot as plt
4 from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
5 import time
6
7 class Menu:
8     def __init__(self, master):
9         print("in menu constructor")
10        self.master = master
11        self.currentStep = 0
12        self.isPaused = False
13        #self.menuWidgets()
14
15    def menuWidgets(self):
16        self.menuFrame = Frame(self.master)
17        self.menuFrame.grid(row=10, column=0, columnspan=8)
18
19        Button(self.menuFrame, text="Skip Back", command=self.skipBack).grid(row=10, column=0, columnspan=2)
20        self.playPauseButton = Button(self.menuFrame, text="Pause", command=self.togglePlayPause)
21        self.playPauseButton.grid(row=10, column=3, columnspan=2)
22        Button(self.menuFrame, text="Skip Forward", command=self.skipForward).grid(row=10, column=7, columnspan=2)
23
24        self.master.bind("<Left>", lambda e: self.stepBack())
25        self.master.bind("<Right>", lambda e: self.stepForward())

```

2.3.s

```

27 def skipBack(self):
28     self.currentStep = 0
29     self.isPaused = True
30     if hasattr(self, "updateChart"): # note that methods count as attributes
31         self.updateChart(self.steps[self.currentStep])
32     # extend once implemented other algorithms
33     #self.invalidMessage["text"] = ""
34
35 def skipForward(self):
36     self.currentStep = len(self.steps) - 1
37     self.isPaused = True
38     if hasattr(self, "updateChart"): # note that methods count as attributes
39         self.updateChart(self.steps[self.currentStep])
40     # extend once implemented other algorithms
41     #self.invalidMessage["text"] = ""

```

2.3.t

```

43 def togglePlayPause(self):
44     self.isPaused = not self.isPaused
45     self.playPauseButton.config(text="Pause" if not self.isPaused else "Play")
46     if not self.isPaused:
47         if hasattr(self, "bubbleSortAnimate"):
48             self.bubbleSortAnimate()
49
50 def stepBack(self):
51     if self.isPaused and self.currentStep > 0:
52         self.currentStep -= 1
53         if hasattr(self, "updateChart"): # note that methods count as attributes
54             numbers = self.steps[self.currentStep]
55             indices = self.stepsDictionary[numbers]
56             self.updateChart(numbers, indices)
57
58 def stepForward(self):
59     if self.isPaused and self.currentStep < len(self.steps)-1:
60         self.currentStep += 1
61         if hasattr(self, "updateChart"): # note that method count as attributes
62             numbers = self.steps[self.currentStep]
63             indices = self.stepsDictionary[numbers]
64             self.updateChart(numbers, indices)

```

And below is the code for the BubbleSort class that I had copied:

2.3.u

```

68 class BubbleSort(Menu):
69     def __init__(self, master):
70         print("in bubble sort constructor")
71         super().__init__(master)
72         print("back in bubble sort constructor")
73         self.master = master
74         self.numbers = [10,9,8,7,6,5,4,3,2,1]
75         self.sortOrder = "Ascending"
76         self.stepsDictionary = self.bubbleSortSteps()
77         self.steps = list(self.stepsDictionary.keys())
78         self.fig = None
79         self.ax = None
80         self.canvas = None
81         self.visualiseText = None

```

2.3.v

```

87     def bubbleSortWidgets(self):
88         self.title = Label(self.master, text="Bubble Sort Algorithm Visualisation")
89         self.title.grid(row=0, column=0, columnspan=12)
90
91         self.bubbleSortFrame = Frame(self.master)
92         self.bubbleSortFrame.grid(row=1, column=0, columnspan=12, rowspan=8)
93
94         Label(self.bubbleSortFrame, text="[10, 9, 8, 7, 6, 5, 4, 3, 2, 1]).grid(row=1, column=0, columnspan=12)
95
96         self.visualiseText = Text(self.bubbleSortFrame, width=50, height=10)
97         self.visualiseText.grid(row=2, column=7, columnspan=5, rowspan=5)
98
99         self.fig, self.ax = plt.subplots()
100        self.canvas = FigureCanvasTkAgg(self.fig, master=self.bubbleSortFrame)
101        self.canvas.get_tk_widget().grid(row=2, column=0, columnspan=7, rowspan=5)
102
103        Button(self.bubbleSortFrame, text="Visualise", command=self.bubbleSortAnimate).grid(row=7, column=2, columnspan=2)

```

2.3.w

```

105     def updateChart(self, numbers, swapIndices=None):
106         self.ax.clear()
107         barColours = ["blue"] * len(numbers) # all bars blue
108         if swapIndices: # not None
109             for index in swapIndices:
110                 barColours[index] = "red" # swapping items in red
111             self.ax.bar(range(len(numbers)), numbers, color=barColours)
112             self.canvas.draw()
113             self.bubbleSortFrame.update()

```

From all the errors that I had encountered, I think that the issues were arising from when I was calling the playAnimation and bubbleSortAnimate methods. Additionally, I think that not having stored all the steps may be causing the issues with the Play/Pause button. Therefore, I firstly created a method to work out all the steps from the algorithm and store them, along with their indices (highlighted bars) in a dictionary. I chose a dictionary for this because it allows for easy searching for the indices as the relationships are created between the numbers and the indices. Before using the dictionary, however, I researched how to make sure I can use it with a list of numbers. Due to the nature of a list, which is mutable, I had to make sure that the type was a tuple, which is immutable, so that it can be accepted as a dictionary key. This can be seen in the code below.

2.3.x

```

115     def bubbleSortSteps(self):
116         print("in bubble sort steps")
117         n = len(self.numbers)
118         swapped = True
119         stepsDict = {}
120         stepsDict[tuple(self.numbers)] = [0]
121
122         if self.sortOrder == "Ascending":
123             while swapped:
124                 swapped = False
125                 for j in range(n-1):
126                     if self.numbers[j] > self.numbers[j+1]:
127                         temp= self.numbers[j]
128                         self.numbers[j] = self.numbers[j+1]
129                         self.numbers[j+1] = temp
130                         swapped = True
131                         #self.steps.append(self.numbers[:])
132                         # note must be tuple because lists are mutable so they are unhashable
133                         stepsDict[tuple(self.numbers)] = [j, j+1]
134         elif self.sortOrder == "Descending":
135             while swapped:
136                 swapped = False
137                 for j in range(n-1):
138                     if self.numbers[j] < self.numbers[j+1]:
139                         temp= self.numbers[j]
140                         self.numbers[j] = self.numbers[j+1]
141                         self.numbers[j+1] = temp
142                         swapped = True
143                         #self.steps.append(self.numbers[:])
144                         # note must be tuple because lists are mutable so they are unhashable
145                         stepsDict[tuple(self.numbers)] = [j, j+1]
146
147     return stepsDict

```

This returns stepsDict, which is then stored in the attribute self.stepsDictionary. The keys from this dictionary are then stored as a list in the attribute steps. As seen in the code before, these can then be passed easily into updateChart in the SkipBack, SkipForward, StepBack and StepForward methods. For the main functionality of the algorithm visualisation - the animation - I made use of the precalculated steps stored so that the values can be displayed. As I learnt from the while loops combined with sleep, the isPaused was not being checked regularly so I combined the functionality that I had come up with - recursively calling the animation method using after - with the outputting of the text numbers and the updating of the chart. See the code below:

2.3.y

```

149     def bubbleSortAnimate(self):
150         if not self.isPaused and self.currentStep < len(self.steps)-1:
151             numbers = self.steps[self.currentStep]
152             indices = self.stepsDictionary[numbers]
153             self.updateChart(numbers, indices)
154             self.visualiseText.insert(tk.END, f"{numbers}\n")
155             self.visualiseText.see(tk.END)
156
157             self.bubbleSortFrame.update()
158             self.currentStep += 1
159             self.master.after(750, self.bubbleSortAnimate)
160
161         elif not self.isPaused and self.currentStep == len(self.steps)-1:
162             numbers = self.steps[self.currentStep]
163             indices = self.stepsDictionary[numbers]
164             self.updateChart(numbers, indices)
165             self.visualiseText.insert(tk.END, f"{numbers}\n")
166             self.visualiseText.see(tk.END)
167             print("sort complete")
168             self.bubbleSortFrame.update()

```

In order to test the code, I added the following lines of code in the testing file:

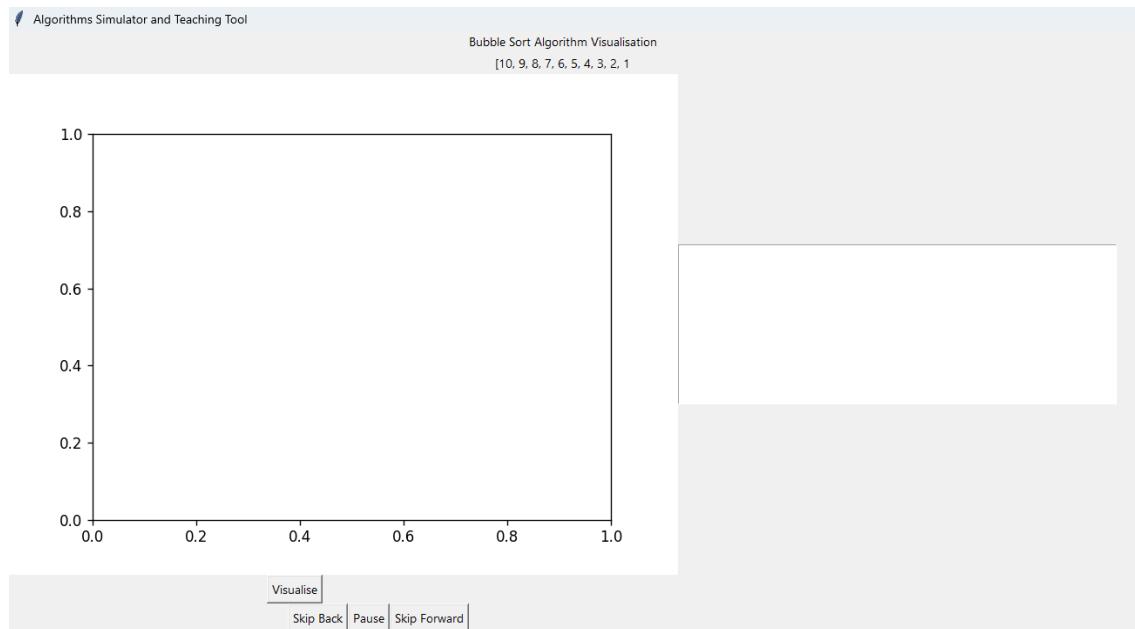
2.3.z

```

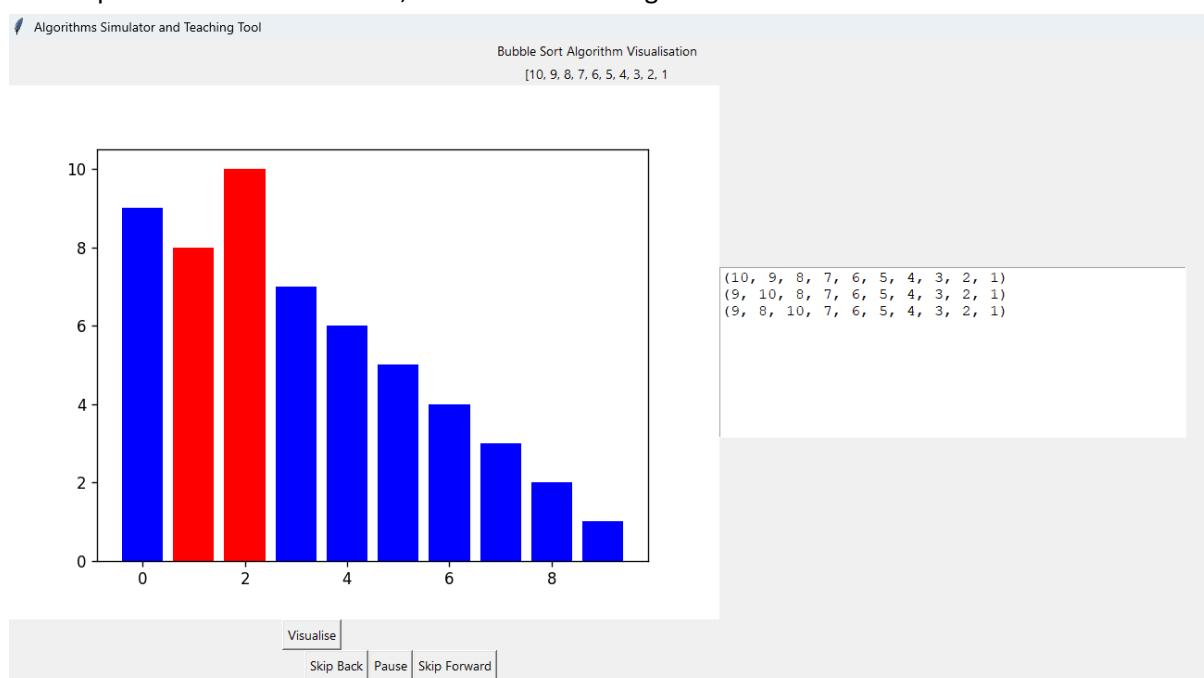
170 if __name__ == "__main__":
171     bubbleSortRoot = Tk()
172     bubbleSortRoot.geometry('1000x1000') # to change
173     bubbleSortRoot.title("Algorithms Simulator and Teaching Tool")
174     running = BubbleSort(bubbleSortRoot)
175     bubbleSortRoot.mainloop()

```

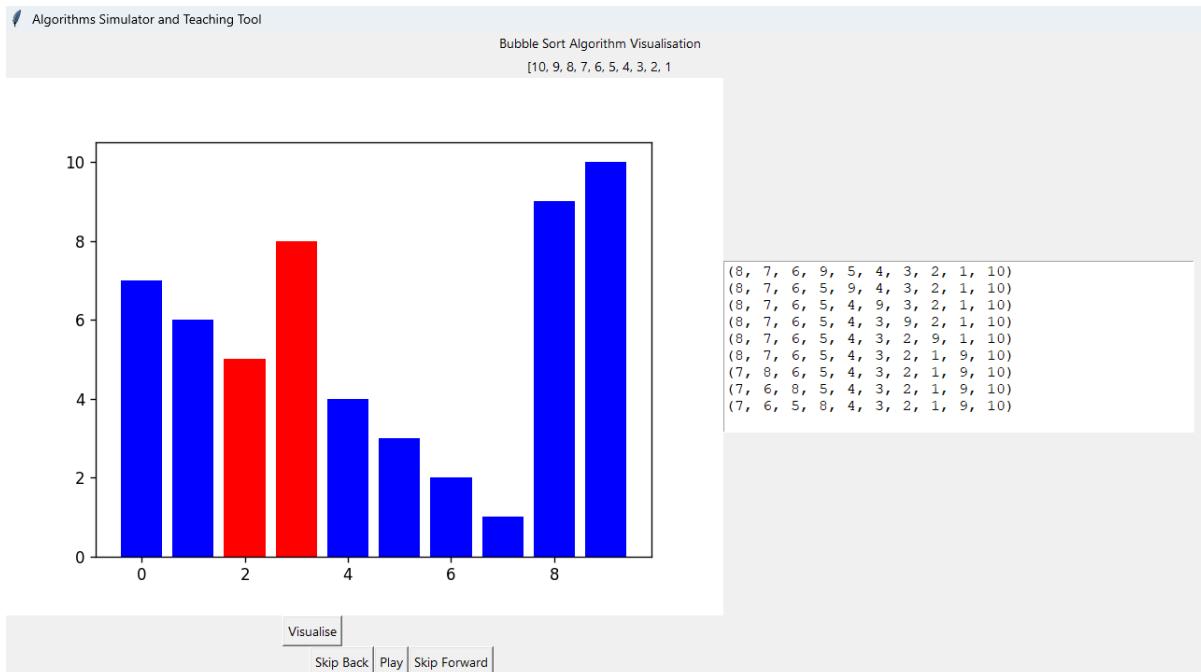
When I now run the code, the following is output initially:



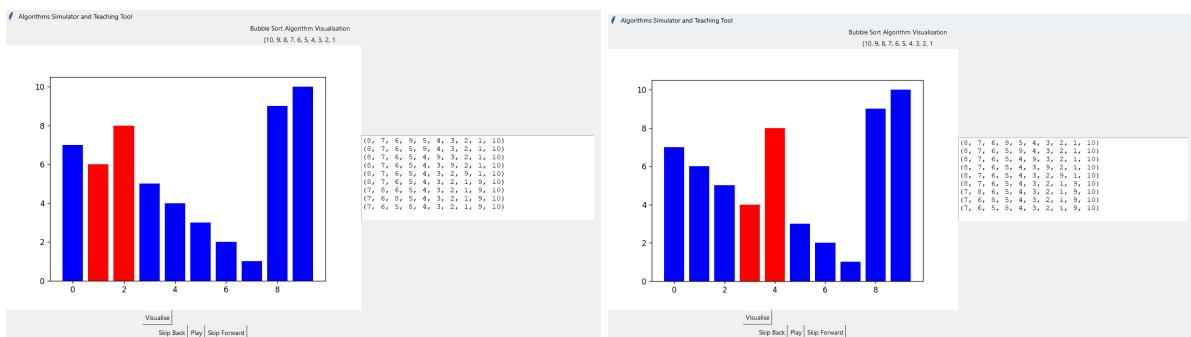
When I press the visualise button, the visualisation begins as intended:



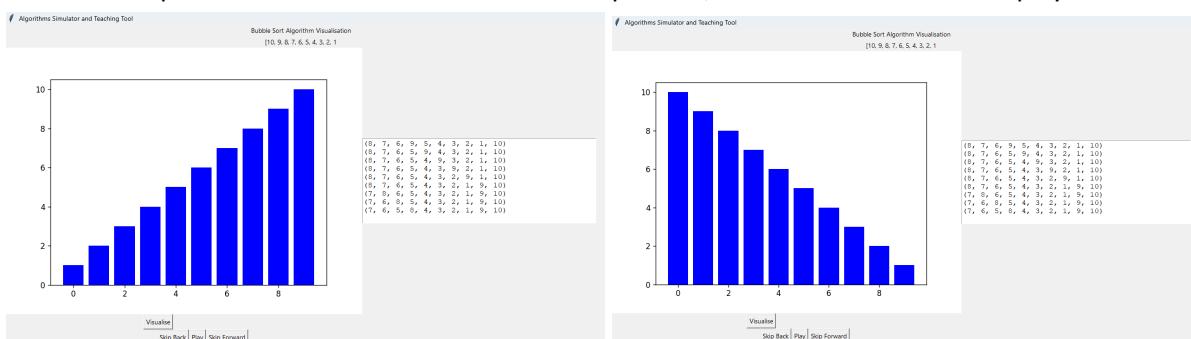
Pressing the pause button stops the algorithm as it should:



And when I press the forward and backward arrow keys, the correct values are displayed:



When the Skip Forward and Backward buttons are pressed, the correct values are displayed:



All functionality is as intended within this reworked version. Therefore, I shall implement this code into the main file. Below is the code from the BubbleSort class, which I had changed based on the code from the testing file:

2.3.A

```

class BubbleSort(HomeMain, Menu):
    def __init__(self, master, pUsername):
        print("inside __init__ 1")
        HomeMain.__init__(self, master, pUsername)
        print("inside __init__ 2")
        Menu.__init__(self, master, 12, 19)
        self.numbers = []
        self.userEntries = []

        self.bubbleSortFrame = Frame(self.master)

        self.addButton = Button(self.bubbleSortFrame, text="Add Number", command=self.addEntry)
        self.sortOrder = StringVar(value="Ascending")

        self.invalidMessage = None

        self.fig = None
        self.ax = None
        self.canvas = None

        self.visualiseText = None

        print(self.numbers)
        print(self.userEntries)
        self.hideHomeWidgets()
        self.bubbleSortWidgets()
        self.menuWidgets()

```

2.3.B

```

def bubbleSortSteps(self):
    print("in bubble sort steps")
    n = len(self.numbers)
    swapped = True
    stepsDict = {}
    stepsDict[tuple(self.numbers)] = [0]

    if self.sortOrder == "Ascending":
        while swapped:
            swapped = False
            for j in range(n-1):
                if self.numbers[j] > self.numbers[j+1]:
                    temp= self.numbers[j]
                    self.numbers[j] = self.numbers[j+1]
                    self.numbers[j+1] = temp
                    swapped = True
                #self.steps.append(self.numbers[:])
                # note must be tuple because lists are mutable so they are unhashable
                stepsDict[tuple(self.numbers)] = [j, j+1]
    elif self.sortOrder == "Descending":
        while swapped:
            swapped = False
            for j in range(n-1):
                if self.numbers[j] < self.numbers[j+1]:
                    temp= self.numbers[j]
                    self.numbers[j] = self.numbers[j+1]
                    self.numbers[j+1] = temp
                    swapped = True
                #self.steps.append(self.numbers[:])
                # note must be tuple because lists are mutable so they are unhashable
                stepsDict[tuple(self.numbers)] = [j, j+1]
    self.stepsDictionary = stepsDict
    self.steps = list(self.stepsDictionary.keys())

```

2.3.C

```

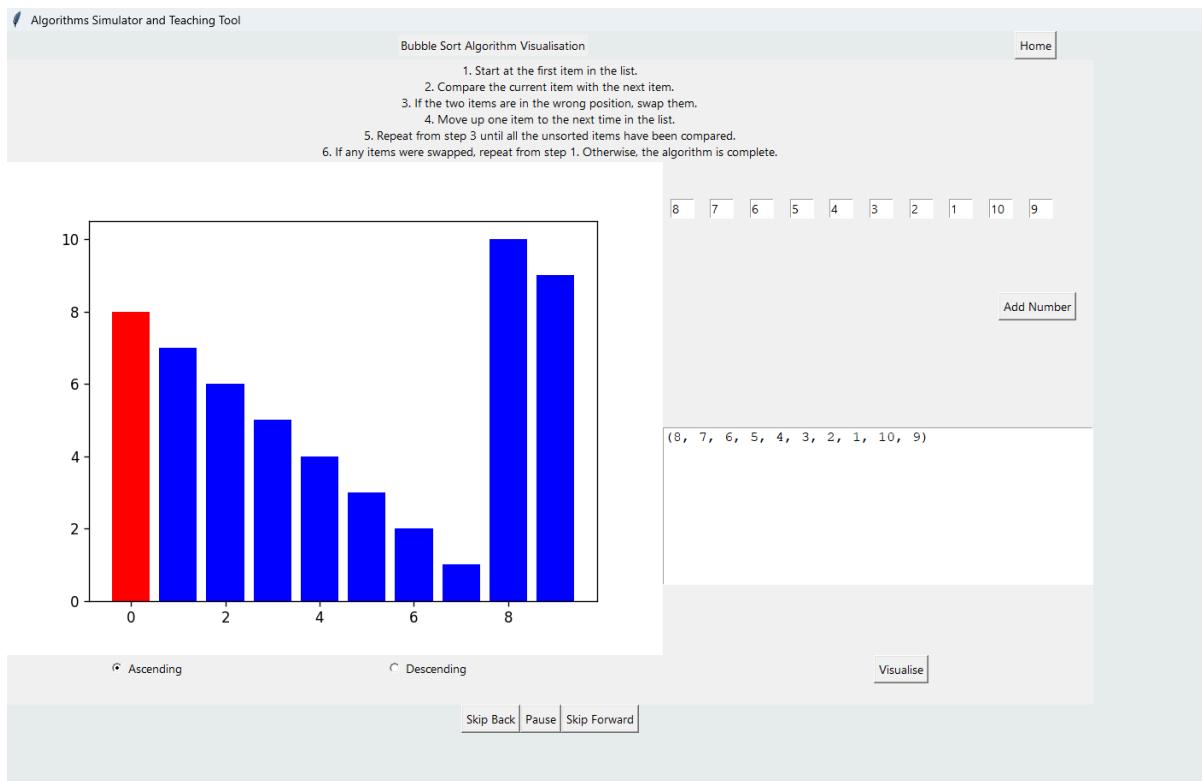
def bubbleSortAnimate(self):
    print("in bubble sort animate")
    if not self.isPaused and self.currentStep < len(self.steps)-1:
        numbers = self.steps[self.currentStep]
        indices = self.stepsDictionary[numbers]
        self.updateChart(numbers, indices)
        self.visualiseText.insert(tk.END, f"{numbers}\n")
        self.visualiseText.see(tk.END)

        self.bubbleSortFrame.update()
        self.currentStep += 1
        self.master.after(750, self.bubbleSortAnimate)

    elif not self.isPaused and self.currentStep == len(self.steps)-1:
        numbers = self.steps[self.currentStep]
        indices = self.stepsDictionary[numbers]
        self.updateChart(numbers, indices)
        self.visualiseText.insert(tk.END, f"{numbers}\n")
        self.visualiseText.see(tk.END)
        print("sort complete")
        self.bubbleSortFrame.update()

```

The rest of the code in the class stayed the same. I copied all the code from the Menu class into the main file as well. When I run the code, the following is output:



The visualisation seems to have stopped and not continue executing. Therefore, I added some print statements to check what values are in the dictionary and steps attributes.

2.3.D

```

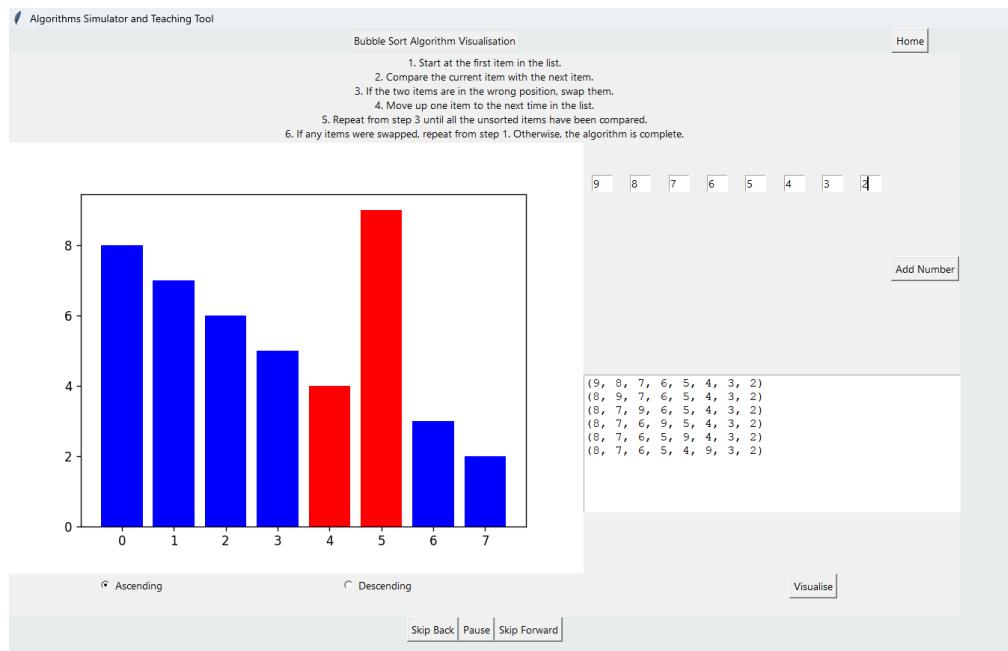
self.bubbleSortSteps()
print(self.steps)
print(self.stepsDictionary)
self.bubbleSortAnimate()
    
```

When I run the code, the following is the output to the Shell:

```

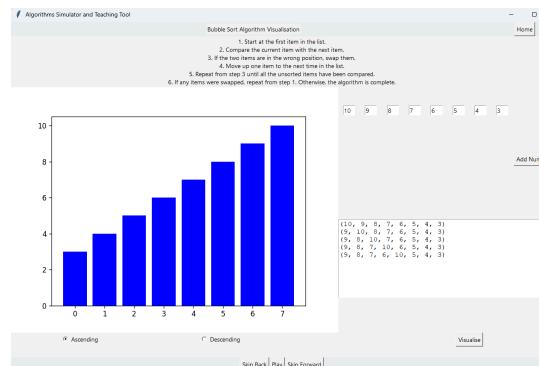
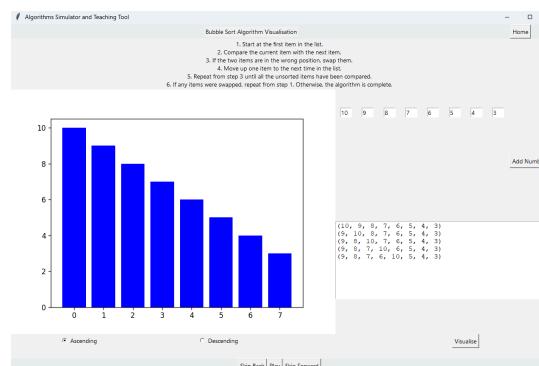
In __classOpen
In __classOpen
inside __init__ 1
inside __init__ 2
[]
[]
userEntries: []
userEntries: [<tkinter.Entry object .!frame2.!entry>]
userEntries: [<tkinter.Entry object .!frame2.!entry>, <tkinter.Entry object .!frame2.!entry2>]
userEntries: [<tkinter.Entry object .!frame2.!entry>, <tkinter.Entry object .!frame2.!entry2>, <tkinter.Entry object .!frame2.!entry3>]
userEntries: [<tkinter.Entry object .!frame2.!entry>, <tkinter.Entry object .!frame2.!entry2>, <tkinter.Entry object .!frame2.!entry3>, <tkinter.Entry object .!frame2.!entry4>]
userEntries: [<tkinter.Entry object .!frame2.!entry>, <tkinter.Entry object .!frame2.!entry2>, <tkinter.Entry object .!frame2.!entry3>, <tkinter.Entry object .!frame2.!entry4>, <tkinter.Entry object .!frame2.!entry5>]
userEntries: [<tkinter.Entry object .!frame2.!entry>, <tkinter.Entry object .!frame2.!entry2>, <tkinter.Entry object .!frame2.!entry3>, <tkinter.Entry object .!frame2.!entry4>, <tkinter.Entry object .!frame2.!entry5>, <tkinter.Entry object .!frame2.!entry6>]
userEntries: [<tkinter.Entry object .!frame2.!entry>, <tkinter.Entry object .!frame2.!entry2>, <tkinter.Entry object .!frame2.!entry3>, <tkinter.Entry object .!frame2.!entry4>, <tkinter.Entry object .!frame2.!entry5>, <tkinter.Entry object .!frame2.!entry6>, <tkinter.Entry object .!frame2.!entry7>]
userEntries: [<tkinter.Entry object .!frame2.!entry>, <tkinter.Entry object .!frame2.!entry2>, <tkinter.Entry object .!frame2.!entry3>, <tkinter.Entry object .!frame2.!entry4>, <tkinter.Entry object .!frame2.!entry5>, <tkinter.Entry object .!frame2.!entry6>, <tkinter.Entry object .!frame2.!entry7>, <tkinter.Entry object .!frame2.!entry8>]
in menuWidgets
in validate
in bubble sort steps
[(10, 9, 8, 7, 6, 5, 4, 3)]
[(10, 9, 8, 7, 6, 5, 4, 3): {}]
in bubble sort animate
in update chart
sort complete
    
```

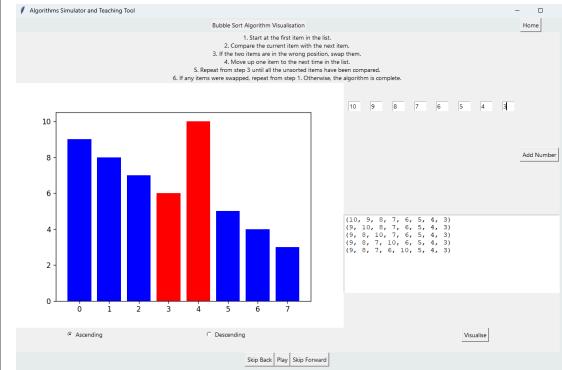
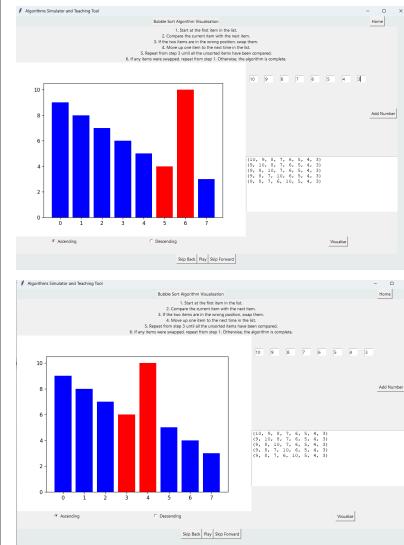
When the dictionary and steps are printed, only 1 line is printed, therefore, it seems that the if statement is not being entered after the sortOrder checks. To fix this, I changed the lines to use .get() because sortOrder is defined as a StringVar value.



All functionality as shown with the testing code is the same and functions correctly. Therefore, the functionality of the Menu is complete for the BubbleSort section of it. Additionally, I ensured that the Menu class is generalised and has the ability to be extended to allow for the other algorithms to use it, therefore making it a reusable piece of code.

I shall now test the Menu using the white box testing table specified in the MENU DESIGN section. However, I specifically only tested against the tests relating to the Bubble Sort visualisation as that is the only one that has been developed.

TEST DATA	EXPECTED RESULT	JUSTIFICATION	CODE	ACTUAL OUTPUT
Press the 'Skip Forward' button for the Bubble Sort visualisation	The text-based visualisation displays all the swaps and the bar chart updates to show the final sorted values.	Confirms that the 'Skip Forward' button works as intended for the Bubble Sort visualisation.	2.3.b 2.3.h 2.3.l	 <p>As expected</p>
Press the 'Skip Back' button for the Bubble Sort visualisation	The dataset remains the same but the initial, unsorted values are displayed as text and on the bar chart.	Confirms that the 'Skip Back' button works as intended for the Bubble Sort visualisation.	2.3.b 2.3.h 2.3.l	 <p>As expected</p>

Press the 'Pause' button for the Bubble Sort visualisation	The visualisation stops, the button changes from 'Pause' to 'Play' (or vice versa).	Confirms that the 'Pause' button works as intended for the Bubble Sort visualisation.	2.3.c 2.3.d 2.3.h 2.3.l	 <p>As expected.</p>
Use the right and left arrow keys for the Bubble Sort visualisation	The visualisation goes back/forward one step in the visualisation on both the bar chart and as text.	Confirms that the arrow key functionality works as intended for the Bubble Sort visualisation.	2.3.c 2.3.h 2.3.l	 <p>As expected</p>

All functionality tests have passed for the Menu section therefore all functionality intended to be developed in this iteration is complete.

Now that development and testing of the menu is complete, I will finish this iteration by refining the GUI. From Iteration 2 and the general design for all the GUI, my tasks for this are to:

- Remove the label on the x axis of the chart
- Increase the size of the Text section where the text-based visualisation is displayed
- Change the colours on the chart to match those specified in the Design section
- Change the colours of the buttons, text and background to match those specified in the Design section
- Change the fonts of the text on the page

See below the code for all of these changes:

2.3.E

```
class BubbleSort(HomeMain, Menu):
    def __init__(self, master, pUsername):
        print("inside __init__ 1")
        HomeMain.__init__(self, master, pUsername)
        print("inside __init__ 2")
        Menu.__init__(self, master, 12, 19)
        self.numbers = []
        self.userEntries = []

        self.master.configure(bg="#eaebed")
        self.master.geometry("1105x905") # required here as goes back into HomeMain first

        self.sortOrder = StringVar(value="Ascending")

        self.invalidMessage = None

        self.fig = None
        self.ax = None
        self.canvas = None

        self.visualiseText = None

        print(self.numbers)
        print(self.userEntries)

        self.hideHomeWidgets()
        self.bubbleSortWidgets()
        self.menuWidgets()
        #self.grid(row=0, column=0, sticky="nsew")
        ...
        self.columnconfigure(0, weight=1)
        self.columnconfigure(1, weight=1)
        self.columnconfigure(2, weight=1)...
```

2.3.F

```
def addEntry(self):
    print("in add entry")
    if len(self.userEntries) < 11:
        entry = Entry(self.entryFrame, width=3)
        entry.grid(row=0, column=(7+len(self.userEntries)))
        self.userEntries.append(entry)
    elif len(self.userEntries) == 11:
        entry = Entry(self.entryFrame, width=3)
        entry.grid(row=0, column=(7+len(self.userEntries)))
        self.userEntries.append(entry)
        self.addButton.grid_forget()
```

2.3.G

```

723     def bubbleSortWidgets(self):
724         # TITLE / TOP SECTION:
725         self.title = Label(self.master, text="Bubble Sort Algorithm Visualisation", font=fontLarge, fg="#1b263b", bg="#eaebcd")
726         self.title.grid(row=0, column=0, columnspan=17)
727
728         Button(self.master, text="Home", command=self.openHome, bg="#1b263b", fg="white", font=fontMedium, width=15).grid(row=0, column=17, columnspan=2)
729
730         # BUBBLE SORT FRAME:
731         self.bubbleSortFrame = Frame(self.master, bg="#eaebcd")
732         self.bubbleSortFrame.grid(row=1, column=0, columnspan=19, rowspan=11)
733
734         Label(self.bubbleSortFrame,
735             text= "1. Start at the first item in the list.\n"
736                 "2. Compare the current item with the next item.\n"
737                 "3. If the two items are in the wrong position, swap them.\n"
738                 "4. Move up one item to the next time in the list.\n"
739                 "5. Repeat from step 3 until all the unsorted items have been compared.\n"
740                 "6. If any items were swapped, repeat from step 1. Otherwise, the algorithm is complete.",
741             font=fontSmall,
742             fg="#1b263b",
743             bg="#eaebcd",
744             justify="left").grid(row=1, column=0, columnspan=19, sticky="w", padx=10, pady=5) # justify aligns the text to the left within the label
745
746         print(f"userEntries: {self.userEntries}")
747         print(self.userEntries) # causing error???
748         # create 8 entry fields for user to enter numbers into
749         self.entryFrame = Frame(self.bubbleSortFrame, bg="#eaebcd")
750         self.entryFrame.grid(row=4, column=7, columnspan=12, rowspan=2)
751         for i in range(8):
752             entry = Entry(self.entryFrame, width=3)
753             entry.grid(row=0 , column=(i+7), padx=2)
754             print(f"userEntries: {self.userEntries}")
755             self.userEntries.append(entry)
756         print(f"userEntries: {self.userEntries}")
757
758         self.addButton = Button(self.entryFrame, text="Add Number", command=self.addEntry, bg="#1b263b", fg="white", font=fontMedium, width=15)
759         self.addButton.grid(row=1, column=15, columnspan=4)

```

2.3.H

```

def menuWidgets(self):
    print("in menuWidgets")

    self.menuFrame = Frame(self.master, bg="#1b263b")
    #self.menuFrame.grid(row=self.rowStart, column = 0, columnspan = self.menuLength, rowspan=3)
    self.menuFrame.grid(row=self.rowStart, column=0, columnspan=self.menuLength, rowspan=3, sticky="ew")

    Button(self.menuFrame, text="Skip Back", command=self.skipBack, fg="#1b263b", bg="#eaebed", font=fontMedium, width=15).grid(row=1, column= 1, padx=10, pady=10)
    self.playPauseButton = Button(self.menuFrame, text="Pause", command=self.togglePlayPause, fg="#1b263b", bg="#eaebed", font=fontMedium, width=15)
    self.playPauseButton.grid(row=1, column=3, padx=5, pady=5)
    Button(self.menuFrame, text="Skip Forward", command=self.skipForward, bg="#eaebed", fg="#1b263b", font=fontMedium, width=15).grid(row=1, column=5, padx=10, pady=10)
    # add text describing arrow stuff
    Label(self.menuFrame, text="Use right/left arrow keys to step forward/back through the visualisation when paused.", font=fontSmall, fg="#eaebed", bg="#1b263b").grid(row=1, column=7, columnspan=12, padx=5)

    # these are for the keyboard inputs:
    self.master.bind("<Left>", lambda e: self.stepBack())
    self.master.bind("<Right>", lambda e: self.stepForward())

```

2.3.1

```

# ascending descending choice:
Radiobutton(self.bubbleSortFrame, text="Ascending", variable=self.sortOrder, value="Ascending", font=fontSmall, fg="#1b263b", bg="#eaebed").grid(row=10, column=0, columnspan=3)
Radiobutton(self.bubbleSortFrame, text="Descending", variable=self.sortOrder, value="Descending", font=fontSmall, fg="#1b263b", bg="#eaebed").grid(row=10, column=3, columnspan=3)

Button(self.bubbleSortFrame, text="Visualise", command=self.validate, bg="#1b263b", fg="white", font=fontMedium, width=20).grid(row=10, column=11, columnspan=4)

self.invalidMessage = Label(self.bubbleSortFrame, text="", font=fontSmall, fg="#1b263b", bg="#eaebed")
self.invalidMessage.grid(row=11, columnspan=19)

# dataset plot
self.fig, self.ax = plt.subplots(figsize=(5,5))
self.ax.set_xticks([])
self.canvas = FigureCanvasTkAgg(self.fig, master=self.bubbleSortFrame)
self.canvas.get_tk_widget().grid(row=4, column=0, columnspan=7, rowspan=5, pady=5)

self.visualiseText = Text(self.bubbleSortFrame, width=50, height=30)
self.visualiseText.grid(row=6, column=7, columnspan=12, rowspan=4)

```

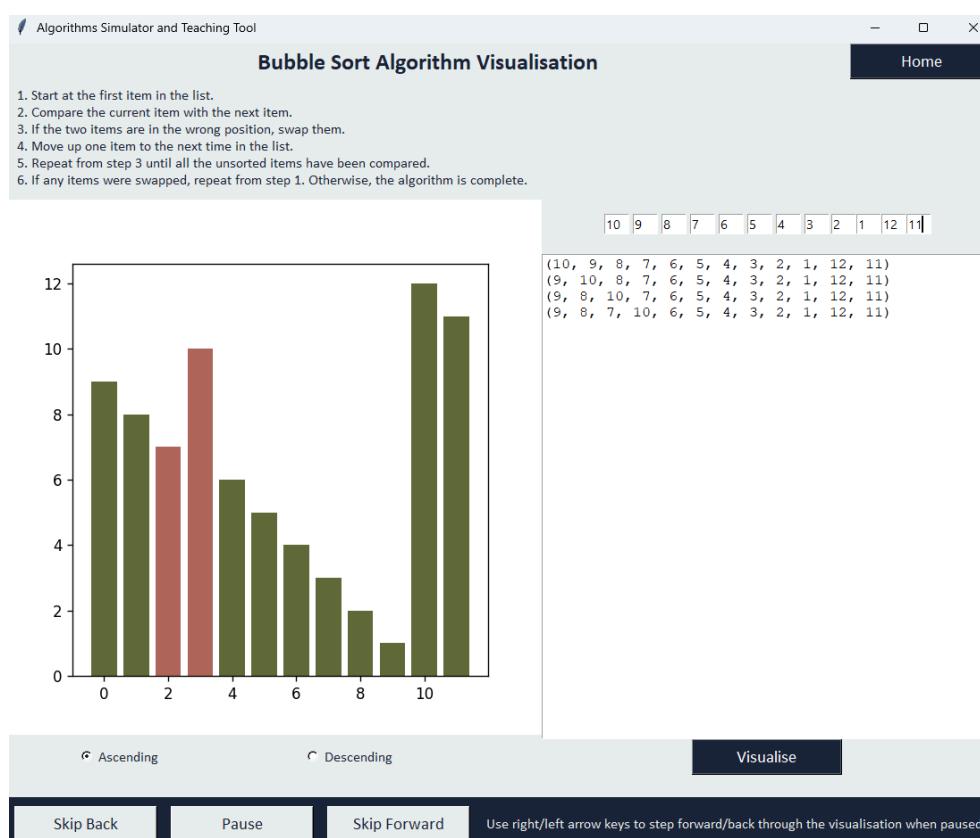
2.3.2

```

791
792     def updateChart(self, numbers, swapIndices=None):
793         # this is for animating the swapping of items on the bar chart
794         print("in update chart")
795         self.ax.clear()
796         barColours = ["#606c38"] * len(numbers) # all bars blue
797         if swapIndices: # not None
798             for index in swapIndices:
799                 barColours[index] = "#b2675e" # swapping items in red
800         self.ax.bar(range(len(numbers)), numbers, color=barColours)
801         self.canvas.draw()
802         self.bubbleSortFrame.update()
803         #time.sleep(0.3) # smooth animation

```

And when it is now run, the following are output:



Now that the GUI of the menu and Bubble Sort visualisation sections have been fully refined, development in Iteration 2 is fully complete as the functionality has already been developed and tested.

ITERATION 3 - DEVELOPER REVIEW

I think that I hadn't thought through the full logic of how the menu and algorithms should be combined. Therefore, this caused some of the issues that I encountered regarding the animation not pausing correctly and the visualisation stopping. However, now that the menu has been fully developed, this should make the development process of the other algorithms much simpler as I now have the structure in place for what should be used in the Menu class.

PROTOTYPE 2 - STAKEHOLDER REVIEW

Now that Prototype 2 has been fully developed and tested, I had a stakeholder review the progress. I asked about the Usability - how user friendly the system is - the Functionality - does it function as intended? - and Performance - how fast the system performs.

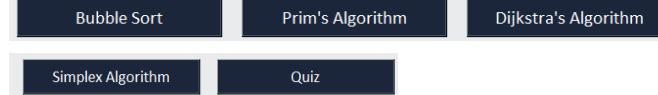
See below the feedback given by one of my Stakeholders:

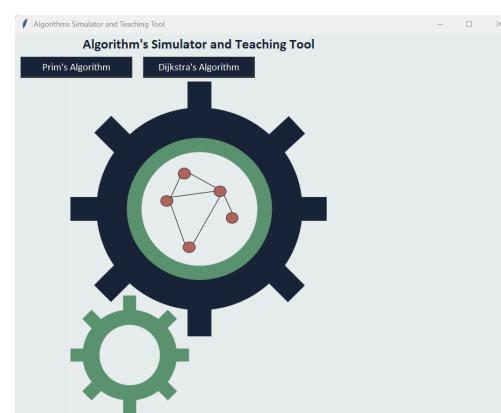
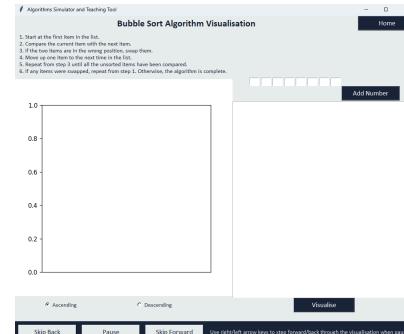
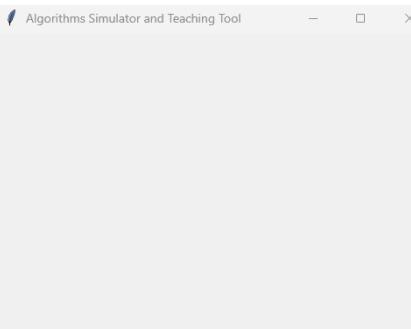
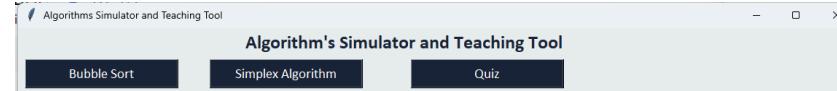
AREA OF CONCERN	FEEDBACK
Usability	Very good, but it would be good to have the ability to clear the visualisation so that it can be restarted.
Functionality	Functions well but it would be nice if functionality for decimal or float numbers could be included.
Performance	Very good performance speed.

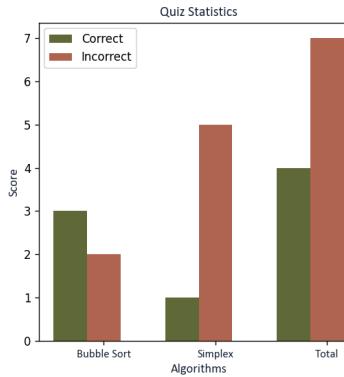
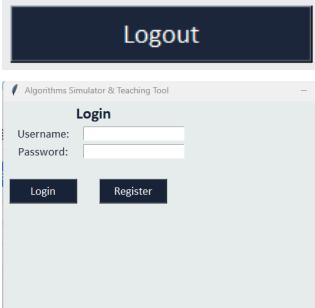
From the feedback given, it seems that, again, the system is very user friendly and performs well. However, two pieces of functionality that were pointed out as things to develop are the resetting of the visualisation so that the user does not have to go back to the home page and then back to the bubble sort page to visualise a new set of data. Secondly, functionality for decimal numbers (float or real numbers) could be developed, which is quite simple to implement. However, due to the time constraints on this project, these enhancements to the performance could be tasks to be completed in post development.

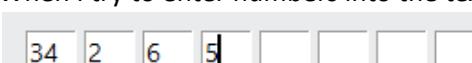
PROTOTYPE 2 - SUCCESS CRITERIA REVIEW

In order to check that all the success criteria outlined in the ANALYSIS section for the Home and Bubble Sort Visualisation pages and the Bubble Sort specific Menu criteria have been met, below is a success criteria review for this prototype:

SUCCESS CRITERIA	OUTPUT & EVIDENCE	CODE	COMPLETE
[3][a] There will be text greeting the user with the title of the system 'Algorithms Simulator and Teaching Tool'. I will test this by checking the page and making sure this appears.		2.1.A	Y
[3][b] There will be a button for each of the algorithm visualisation pages (out of the Bubble Sort Visualisation, Prim's Algorithm Visualisation, Dijkstra's Algorithm Visualisation and Simplex Algorithm Visualisation) that the user checked upon registration - see criteria [2][h]. When each is clicked, the user will be sent to the corresponding page. I will test this by creating accounts with different combinations of algorithm choices and making sure that when the button is clicked, the correct page is accessed.	<p>For a student account with all the algorithms selected:</p>  <p>For a student account with some algorithms selected:</p>  <p>For a teacher account with all the algorithms selected:</p>  <p>For a teacher account with some algorithms selected:</p>	2.1.a 2.1.b 2.1.c 2.1.d 2.1.e 2.1.q 2.1.s 2.1.t 2.1.z 2.1.A 2.1.B 2.1.E 2.1.G 2.1.H 2.1.I	Y

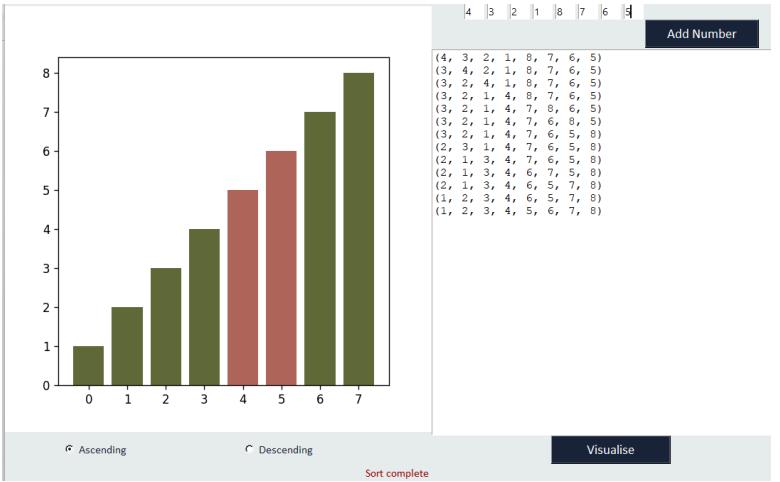
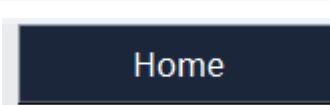
	 <p>When each is clicked, the following are output:</p>   <p>Note that for the Prim's, Dijkstra's, Simplex and Quiz pages, the same screen is output. This is because they have not been developed yet.</p>	
[3][c] If the user has a teacher account, there will be a Quiz button. When clicked, this should take the user to the Quiz page. It should appear below the algorithm visualisation page buttons. To test this, I will create student and teacher accounts and check that this	 <p>The placing has been changed to next to the algorithm visualisation buttons but otherwise this criteria has been met. See the screenshots</p>	2.1.A Y

button only appears for student accounts and that clicking it takes the user to the Quiz page successfully.	above as evidence for the Quiz button not being displayed for Teacher accounts.														
[3][d] If the user has a student account, there will be a bar chart of the user's quiz statistics, including the number of correctly and incorrectly answered questions of each type of algorithm they have completed. This data should be read from the database. I will test this by creating accounts with different numbers of correctly and incorrectly answered questions of each type and check that the data is output correctly on the bar chart on the Home page.	 <p>The chart displays quiz statistics for three categories: Bubble Sort, Simplex Algorithms, and Total. The Y-axis represents the score, ranging from 0 to 7. The X-axis lists the categories. For each category, there are two bars: a green bar for 'Correct' answers and a red bar for 'Incorrect' answers.</p> <table border="1"> <thead> <tr> <th>Category</th> <th>Correct (Green)</th> <th>Incorrect (Red)</th> </tr> </thead> <tbody> <tr> <td>Bubble Sort</td> <td>3</td> <td>2</td> </tr> <tr> <td>Simplex Algorithms</td> <td>1</td> <td>5</td> </tr> <tr> <td>Total</td> <td>4</td> <td>7</td> </tr> </tbody> </table> <p>This is for the testing account with incorrect and correct scores entered.</p>	Category	Correct (Green)	Incorrect (Red)	Bubble Sort	3	2	Simplex Algorithms	1	5	Total	4	7	2.1.h 2.1.i 2.1.p 2.1.q 2.1.r 2.1.t	Y
Category	Correct (Green)	Incorrect (Red)													
Bubble Sort	3	2													
Simplex Algorithms	1	5													
Total	4	7													
[3][e] There will be a Logout button in the bottom right corner of the page. When clicked, the user should be logged out of their account and return the Initial Login page. I will test this by checking that when the Home page is reached, this appears in the correct place and that clicking it has the correct functionality, as described.	 <p>When pressed:</p>	2.1.a 2.1.b 1.1.d 1.1.e 1.1.f 1.3.b 1.3.d	Y												
[4][a] There will be text greeting the user with the title of the page 'Bubble Sort Visualisation'. I will test this by checking the page and making sure this appears.	 <p>Bubble Sort Algorithm Visualisation</p>	2.3.G	Y												

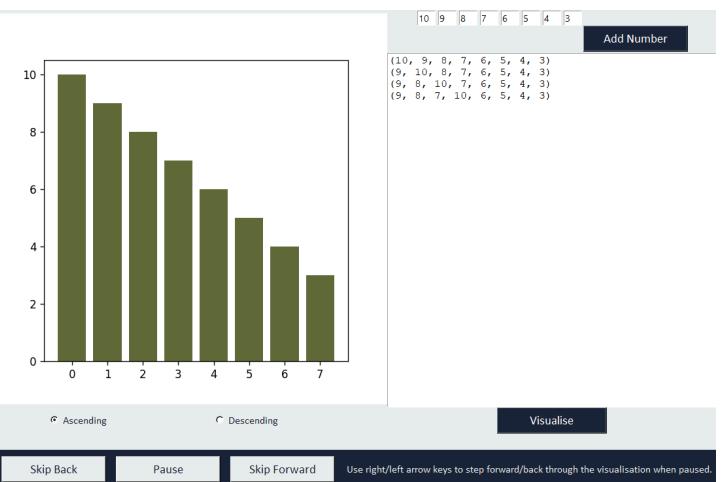
<p>[4][b] There will be text describing the Bubble Sort algorithm in steps in plain English and the requirements for the dataset. I will test this by checking the page and making sure this appears.</p>	<p>1. Start at the first item in the list. 2. Compare the current item with the next item. 3. If the two items are in the wrong position, swap them. 4. Move up one item to the next time in the list. 5. Repeat from step 3 until all the unsorted items have been compared. 6. If any items were swapped, repeat from step 1. Otherwise, the algorithm is complete.</p>	2.3.G	Y
<p>[4][c] There will be 8 text boxes on the right half of the screen that the user can enter numbers into. To test this, I will check that these appear and that numbers can be entered.</p>		2.3.G	Y
<p>[4][d] There will be a button in line with the text boxes to add 1 more text box. This will only appear until there are 12 text boxes. To test this, I will check that this button appears and that up to 4 text boxes can be added, the button disappears when there are 12 text boxes and that numbers can be entered into the text boxes.</p>	<p>Add Number</p> <p>When pressed:</p>  <p>The number of text boxes to be added has been changed due to the size of the screen so the maximum is 12.</p>  <p>When I try to enter numbers into the text boxes:</p> 	2.3.G 2.3.I	Y
<p>[4][e] There is an ascending/descending order selection that the user must make. To test this, I will check that on the Bubble Sort Visualisation page this is displayed and the selection functions as intended.</p>	<input checked="" type="radio"/> Ascending <input type="radio"/> Descending	2.3.G	Y

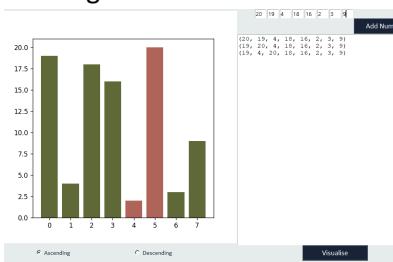
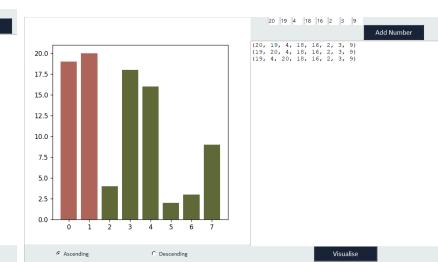
<p>[4][f] There is a Start Visualisation button below the text describing the algorithm. Pressing this will initially validate that the dataset is made up of eight numbers between 1 and 100 (inclusive). Additionally, it is checked that a choice of ascending or descending order is made. If this validation fails, a message will appear saying ‘Dataset not valid’ or ‘Order not chosen.’. If this validation is successful the entered dataset will appear on a bar chart, located on the left of the screen, and start the visual execution of the Bubble Sort algorithm. To test this, I will check that the button appears on the screen in the correct place and that the visualisation appears as it should. Additionally, to test the validation of the dataset, I will input different values that are/aren’t accepted and try not choosing an order to check that it functions correctly.</p>	 <p>The screenshot shows a bar chart for a dataset of length 8 (values 85, 1, 23, 4, 99, 56, 25, 79) in descending order. The bars are colored red (1), green (23, 4, 99, 56, 25), and dark green (79). Below the chart, the original dataset is listed: (85, 1, 23, 4, 99, 56, 25, 79). At the bottom, there are buttons for 'Ascending', 'Descending', 'Visualise', 'Skip Back', 'Play', and 'Skip Forward'. A note says: 'use right/left arrow keys to step forward/back through the visualisation when paused.'</p>	2.3.B 2.3.C 2.3.G 2.3.H 2.3.I 2.3.J	Y

	The functionality discussed can be seen more clearly with a video however, they all function as stated in the success criteria.		
[4][h] For each pass, the text for all the numbers should be on one line. To test this, I will make sure that when the visualisation has started, the passes are on one line in the text section.	<p>(85, 1, 23, 4, 98, 56, 25, 78) (1, 85, 23, 4, 98, 56, 25, 78) (1, 23, 85, 4, 98, 56, 25, 78)</p> <p>Instead of the passes, the result after each check is displayed. This was due to the time limitations I had preventing me from animating the text to be in passes. Therefore, they are displayed in lines.</p>	2.3.C 2.3.I	The way the text based results are shown has changed but otherwise SC has been met
[4][i] For each comparison, if there is a swap needed, there will be an animation of the two items swapping on both the bar chart and the text numbers	<p>(85, 1, 23, 4, 98, 56, 25, 78) (1, 85, 23, 4, 98, 56, 25, 78) (1, 23, 85, 4, 98, 56, 25, 78)</p> <p>Instead of the passes, the result after each check is displayed. This was due to the time limitations I had preventing me from animating the text to be in passes. Therefore, they are displayed in lines.</p>	2.3.C 2.3.I	N

<p>[4][j] When the visualisation is finished, a message will be displayed saying 'Sort Complete'. To test this, I will check that when the dataset has been sorted, this message is displayed.</p>	 <p>The screenshot shows a bar chart with 8 bars representing the sorted dataset [1, 2, 3, 4, 5, 6, 7, 8]. Below the chart are two radio buttons: 'Ascending' (selected) and 'Descending'. At the bottom right is a 'Visualise' button. Above the chart is a list of 16 numbers: (4, 3, 2, 1, 8, 7, 6, 5), (3, 4, 2, 1, 8, 7, 6, 5), (3, 2, 4, 1, 8, 7, 6, 5), (3, 2, 1, 4, 8, 7, 6, 5), (3, 2, 1, 4, 7, 8, 6, 5), (3, 2, 1, 4, 7, 6, 8, 5), (3, 2, 1, 4, 7, 6, 5, 8), (2, 3, 1, 4, 7, 6, 5, 8), (2, 1, 3, 4, 7, 6, 5, 8), (2, 1, 3, 4, 6, 7, 5, 8), (2, 1, 3, 4, 6, 5, 7, 8), (1, 2, 3, 4, 5, 6, 7, 8), (1, 2, 3, 4, 5, 6, 7, 8).</p>	<p>2.3.J</p>	<p>Y</p>
<p>[4][k] There is a Back button at the bottom right of the screen that returns the user to the Home page. To test this, I will check that the button appears in the correct place and functions as intended.</p>	 <p>Placement has been changed to the top right side of the screen. The text has also been changed to 'Home' so that the button's functionality is clearer.</p>	<p>2.3.G</p>	<p>Y with minor changes</p>

<p>[4][i] There is a menu - criteria [9] - that appears at the bottom of the screen. To test this, I will check that it is displayed in the correct place and functions as intended.</p>		2.3.H 2.3.I	Y
<p>[9][a] The Menu will be at the bottom of the screen for each visualisation of the algorithms. It will have 3 buttons - 'Skip Back', 'Pause' and 'Skip Forward' - along with text to describe the instructions for keyboard arrow inputs for stepping forward/back throughout algorithms. I will test that these all appear in the correct place for each algorithm's visualisation page and that the appearance is as designed.</p>		2.3.H 2.3.I	Y

<p>[9][b] The ‘Skip Forward’ button will skip forward through the visualisation to the final solution. For Bubble Sort, this will be all the text-based passes and a bar chart visualisation that is ordered. To test this, I will press the ‘Skip Forward’ button for each of the different algorithms, on different datasets/graphs/constraints, and check that the functionality is as intended.</p>	 <p>10 9 8 7 6 5 4 3 Add Number Visualise Ascending Descending Skip Back Pause Skip Forward Use right/left arrow keys to step forward/back through the visualisation when paused.</p>	2.3.b 2.3.h 2.3.i	Y
<p>[9][c] The ‘Skip Back’ button will return the screen to the initial state of the algorithm’s visualisation, which is where the original state of the dataset, graph of constraints will be displayed. To test this, I will press this button for all the algorithm visualisations, with a variety of datasets/graphs/constraints, and check that its functionality is as intended.</p>	 <p>10 9 8 7 6 5 4 3 Add Number Visualise Ascending Descending Skip Back Pause Skip Forward Use right/left arrow keys to step forward/back through the visualisation when paused.</p>	2.3.b 2.3.h 2.3.i	Y

<p>[9][d] The ‘Pause’ button will stop the execution of the visualisation and will retain the state of the different variables and graphics. When the visualisation is paused, the ‘Pause’ button will change to a ‘Play’ button, which will restart the visualisation from where it was left off. To test this, I will press this button - both when ‘Pause’ and ‘Play’ - at different points in the visualisation and check that it functions as intended for all the different algorithms, using a variety of datasets/graphs/constraints.</p>	 <p>20 19 4 15 16 2 3 9 Add Number Visualise Ascending Descending Skip Back Play Skip Forward Use right/left arrow keys to step forward/back through the visualisation when paused.</p>	<p>2.3.c 2.3.d 2.3.h 2.3.i 2.3.j</p>	<p>Y</p>
<p>[9][e] When the user presses their keyboards left or right arrow, the visualisation will stop executing at the set rate and will instead go onto the next/previous step when the user presses the right/left arrow key. To test this, I will press these keys at different points in the execution in the visualisations for each algorithm, using a variety of datasets/graphs/constraints, and check that they function correctly.</p>	<p>With the paused state shown above: Press right arrow:</p>  <p>20 19 4 15 16 2 3 9 Add Number Visualise Ascending Descending Skip Back Play Skip Forward Use right/left arrow keys to step forward/back through the visualisation when paused.</p> <p>Press left arrow:</p>  <p>20 19 4 15 16 2 3 9 Add Number Visualise Ascending Descending Skip Back Play Skip Forward Use right/left arrow keys to step forward/back through the visualisation when paused.</p>	<p>2.3.c 2.3.h 2.3.i</p>	<p>Y</p>

Most success criteria have been fully met. However, changes were made to how the Bubble Sort text is animated when I was developing the code.

Therefore, success criteria [4][h] and [4][i] have not been met. I will not be able to develop the features or functionality given in these features in the time frame but the prototype that I have created right now is sufficient. Therefore, development of Prototype 2 is fully complete.

PROTOTYPE 3 - PRIM'S AND DIJKSTRA'S VISUALISATION PAGES AND GRAPH INPUT

Before beginning the development within Prototype 3, I split up the code that I had in the NEA_main_file.py file so that navigating the different sections of code will be easier. I split it up so that each class is in a separate file. This became my new set up with the following being imported in each file:

NEA_main_file	login_register_page	home_page
<pre> 13 # imports specific to the initial main class 14 from tkinter import * 15 import tkinter as tk 16 import sqlite3 17 from login_register_page import LoginRegister 18 from home_page import HomeMain 19 from bubble_sort_page import BubbleSort 20 from prim_page import Prim 21 from dijkstra_page import Dijkstra 22 from simplex_page import Simplex 23 from quiz_page import Quiz </pre>	<pre> 1 # imports required specific to the LoginRegister class 2 from tkinter import * 3 import tkinter as tk 4 import sqlite3 5 from NEA_main_file import Main, closeOpen 6 import string </pre>	<pre> 1 # imports required specific to the HomeMain class 2 from tkinter import * 3 import tkinter as tk 4 import sqlite3 5 import matplotlib.pyplot as plt 6 from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg # allows matplotlib graph to 7 from PIL import Image, ImageTk # for logo image editing 8 from NEA_main_file import closeOpen, Main </pre>
bubble_sort_page	prim_page	dijkstra_page
simplex_page	quiz_page	menu_code

However, when I run the NEA_main_file, I get the following output:

```

>>> %Run NEA_main_file.py
Traceback (most recent call last):
  File "C:\Users\ingri\OneDrive\Documents\QM\QM School work\A Level\Computer Science\NEA\NEA_main_file.py", line 17, in <module>
    from login_register_page import LoginRegister
  File "C:\Users\ingri\OneDrive\Documents\QM\QM School work\A Level\Computer Science\NEA\login_register_page.py", line 5, in <module>
    from NEA_main_file import Main #closeOpen
  File "C:\Users\ingri\OneDrive\Documents\QM\QM School work\A Level\Computer Science\NEA\NEA_main_file.py", line 17, in <module>
    from login_register_page import LoginRegister
ImportError: cannot import name 'LoginRegister' from partially initialized module 'login_register_page' (most likely due to a circular import) (C:\Users\ingri\OneDrive\Documents\QM\QM School work\A Level\Computer Science\NEA\login_register_page.py)

```

The error is due to a circular import, which is an error that occurs when two modules mutually dependent on each other try to import before fully loading. To fix this, I moved the closeOpen function into a new file called utilities. I also created a new file specifically for the Main class. When I try running the code, however, I still get a circular import issue:

```
>>> %Run login_register_page.py
Traceback (most recent call last):
  File "C:\Users\ingri\OneDrive\Documents\QM\QM School work\A Level\Computer Science\NEA\login_register_page.py", line 8, in <module>
    from NEA_utilities import closeOpen, fontLarge, fontMedium, fontSmall
  File "C:\Users\ingri\OneDrive\Documents\QM\QM School work\A Level\Computer Science\NEA\NEA_utilities.py", line 3, in <module>
    from login_register_page import LoginRegister
  File "C:\Users\ingri\OneDrive\Documents\QM\QM School work\A Level\Computer Science\NEA\login_register_page.py", line 8, in <module>
    from NEA_utilities import closeOpen, fontLarge, fontMedium, fontSmall
ImportError: cannot import name 'closeOpen' from partially initialized module 'NEA_utilities' (most likely due to a circular import) (C:\Users\ingri\OneDrive\Documents\QM\QM School work\A Level\Computer Science\NEA\NEA_utilities.py)
```

As I have never dealt with this issue before, I researched methods of solving this issue. From my research, I identified two possible solutions:

- Use lazy imports in the closeOpen function
- Rearrange the imports and the classes

I decided that rearranging the imports and the classes may make the development of the system as a whole a bit more complicated for me as a single developer. Therefore, I decided to use lazy imports in the closeOpen function. The advantage of this approach is that it breaks the circular dependencies, improves the code organisation and prevents redundant imports. However, there is potential for debugging complexity and reduced clarity for other developers that could work on the system.

Furthermore, slight performance issues may occur. I implemented this with the following code, which has the same try, import, except structure for all the algorithm visualisation and quiz pages.

3.1.a

```
12  from tkinter import *
13  import tkinter as tk
14
15
16 def closeOpen(root, newType, username=None):
17     print("in closeOpen")
18     root.destroy()
19     if newType == "home":
20         try:
21             from home_page import HomeMain
22             homeRoot = Tk()
23             homeRoot.geometry('1150x740') # to change
24             homeRoot.title("Algorithms Simulator and Teaching Tool")
25             running = HomeMain(homeRoot, username)
26             homeRoot.mainloop()
27         except ImportError as e:
28             print(f"Failed import LoginRegister: {e}")
29             return
30
31     elif newType == "login":
32         try:
33             from login_register_page import LoginRegister
34             loginRoot = Tk()
35             loginRoot.geometry('625x400')
36             loginRoot.title("Algorithms Simulator & Teaching Tool")
37             running = LoginRegister(loginRoot)
38             loginRoot.mainloop()
39         except ImportError as e:
40             print(f"Failed import LoginRegister: {e}")
41             return
```

Now, when I run the NEA_main_file, no errors occur and the system functions as it did when Prototype 2 was completed. The structure of my NEA project is now the following:

 AlgorithmTeachingToolDB		19/01/2025 12:30	Data Base File	24 KB
 bubble_sort_page		08/02/2025 17:57	Python file	10 KB
 dijkstra_page		08/02/2025 17:57	Python file	1 KB
 home_page		08/02/2025 17:56	Python file	10 KB
 login_register_page		08/02/2025 17:57	Python file	16 KB
 main_class_code		08/02/2025 17:55	Python file	1 KB
 menu_code		09/02/2025 16:10	Python file	4 KB
 NEA_logo_image		19/01/2025 10:45	PNG File	30 KB
 NEA_main_file		08/02/2025 17:56	Python file	2 KB
 NEA_utilities		09/02/2025 16:09	Python file	4 KB
 prim_page		08/02/2025 17:57	Python file	1 KB
 quiz_page		08/02/2025 17:57	Python file	1 KB
 simplex_page		08/02/2025 17:57	Python file	1 KB

The advantages of splitting up a project like this into multiple smaller files are that the code organisation is improved because the modularity makes it easier to locate and manage different parts of the program, debugging is simplified, and scalability able to be implemented because new features can be added as new files without making any existing files too large or complex. However, as seen when I tried splitting up the code, issues with import dependencies can arise.

ITERATION 1 - FUNCTIONALITY OF GRAPH INPUT SECTION

In this iteration, I am going to fully develop the code for the Graph Input section and then I will set up the Prim's Visualisation page, with the Graph Input section implemented inside it. To begin, I wrote the code for the Graph Input section inside a tester file - note that the naming conventions I used in my main code are not being followed as this is tester code. In addition, after researching how to approach the problem, I decided to change the use of the Shift Drag that I had described in the Analysis and Design sections to single clicks on the two vertices being connected as the way to input an edge.

3.1.b

```

1 import tkinter as tk
2 from tkinter import simpledialog
3 import networkx as nx
4 import matplotlib.pyplot as plt
5 from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
6
7 class GraphInput:
8     def __init__(self, master):
9         self.master = master
10        self.graph = nx.Graph()
11
12        self.canvas_frame = tk.Frame(master)
13        self.canvas_frame.pack(side=tk.LEFT, fill=tk.BOTH, expand=True)
14
15        self.canvas = tk.Canvas(self.canvas_frame, width=500, height=500, bg='white')
16        self.canvas.pack()
17
18        self.fig, self.ax = plt.subplots(figsize=(5,5))
19        self.graph_canvas = FigureCanvasTkAgg(self.fig, master)
20        self.graph_canvas.get_tk_widget().pack(side=tk.RIGHT, fill=tk.BOTH, expand=True)
21
22        self.selected_vertex = None
23        #self.target_vertex = None
24        self.selected_edge = None
25
26        # set up the keyboard/user clicks/inputs:
27        self.canvas.bind("<Double-Button-1>", self.add_vertex)
28        self.canvas.bind("<Button-1>", self.select_or_create_edge)
29        self.canvas.bind("<Delete>", self.delete_element)

```

Then I set up the add, select and delete methods:

3.1.c

```

def add_vertex(self, event):
    print("in add vertex")
    if len(self.graph.nodes) < 10:
        vertex_id = simpledialog.askstring("Vertex Input", "Enter a unique vertex identifier (1 character):")
        if vertex_id and len(vertex_id) == 1 and vertex_id not in self.graph.nodes:
            x, y = event.x, event.y
            self.graph.add_node(vertex_id, pos=(x, y))

            self.canvas.create_oval(x-10, y-10, x+10, y+10, fill='blue', outline='black')
            self.canvas.create_text(x, y, text=vertex_id, fill='white')

    print(f"Added vertex: {vertex_id} at position ({x}, {y})")
    self.update_visualization()

```

With this implementation, I decided to use a dialogue message box to make the user input the identifier. This is because it makes validating the input easier. I did the same for inputting the edge in the code below:

3.1.d

```

def select_or_create_edge(self, event):
    print("in select or create edge")
    for node, (x, y) in nx.get_node_attributes(self.graph, 'pos').items():
        if abs(event.x - x) < 10 and abs(event.y - y) < 10:
            if self.selected_vertex is None:
                self.selected_vertex = node
                print(f"selected vertex: {node}")
            elif self.selected_vertex != node:
                self.target_vertex = node
                weight = simpledialog.askinteger("Edge Weight", "Enter a positive integer weight:")
                if weight and weight > 0:
                    self.graph.add_edge(self.selected_vertex, self.target_vertex, weight=weight)
                    # Draw the edge visually on the canvas
                    self.canvas.create_line(
                        self.graph.nodes[self.selected_vertex]['pos'][0],
                        self.graph.nodes[self.selected_vertex]['pos'][1],
                        self.graph.nodes[self.target_vertex]['pos'][0],
                        self.graph.nodes[self.target_vertex]['pos'][1],
                        fill="black", width=2, tags=(self.selected_vertex, self.target_vertex))
            )
            print(f"Created edge between {self.selected_vertex} and {self.target_vertex} with weight {weight}")
            self.update_visualization()
            self.selected_vertex = None
            self.target_vertex = None
    return

# Check if the click is on an edge
for edge in self.graph.edges:
    x1, y1 = self.graph.nodes[edge[0]]['pos']
    x2, y2 = self.graph.nodes[edge[1]]['pos']
    if min(x1, x2) <= event.x <= max(x1, x2) and min(y1, y2) <= event.y <= max(y1, y2):
        self.selected_edge = edge
        print(f"Selected edge: {self.selected_edge} for deletion")
        return

self.selected_vertex = None
self.target_vertex = None

```

3.1.e

```

def delete_element(self, event):
    print("in delete element")
    if self.selected_vertex is not None:
        print(f"Selected vertex: {self.selected_vertex} for deletion")
        for node, (x, y) in nx.get_node_attributes(self.graph, 'pos').items():
            if abs(event.x - x) < 10 and abs(event.y - y) < 10:
                self.graph.remove_node(node)
                # Remove the vertex from the canvas
                self.canvas.delete(node)

        print(f"Deleted vertex: {node}")
        self.update_visualization()
        self.selected_vertex = None
    return

```

```

if self.selected_edge is not None:
    print(f"Selected edge: {self.selected_edge} for deletion")
    for edge in list(self.graph.edges):
        x1, y1 = self.graph.nodes[edge[0]]['pos']
        x2, y2 = self.graph.nodes[edge[1]]['pos']
        if min(x1, x2) <= event.x <= max(x1, x2) and min(y1, y2) <= event.y <= max(y1, y2):
            self.graph.remove_edge(*self.selected_edge)
            # Remove the edge from the canvas
            edge_tag = self.selected_edge
            self.canvas.delete(edge_tag)
            #self.canvas.delete(edge)
            print(f"Deleted edge between {edge[0]} and {edge[1]}")
            self.update_visualization()
            self.selected_edge = None
    return

```

For all of these, I had put print statements in so that I could check that elements were being selected and added correctly. Below is the code that I wrote for updating the visualisation:

3.1.f

```

def update_visualization(self):
    print("in update visualization")
    self.ax.clear()
    pos = nx.get_node_attributes(self.graph, 'pos')
    nx.draw(self.graph, pos, with_labels=True, node_color='blue', edge_color='black', node_size=500, font_color='white', ax=self.ax)
    edge_labels = nx.get_edge_attributes(self.graph, 'weight')
    nx.draw_networkx_edge_labels(self.graph, pos, edge_labels=edge_labels, ax=self.ax)
    self.graph_canvas.draw()

```

In order to test this code, I added the following code:

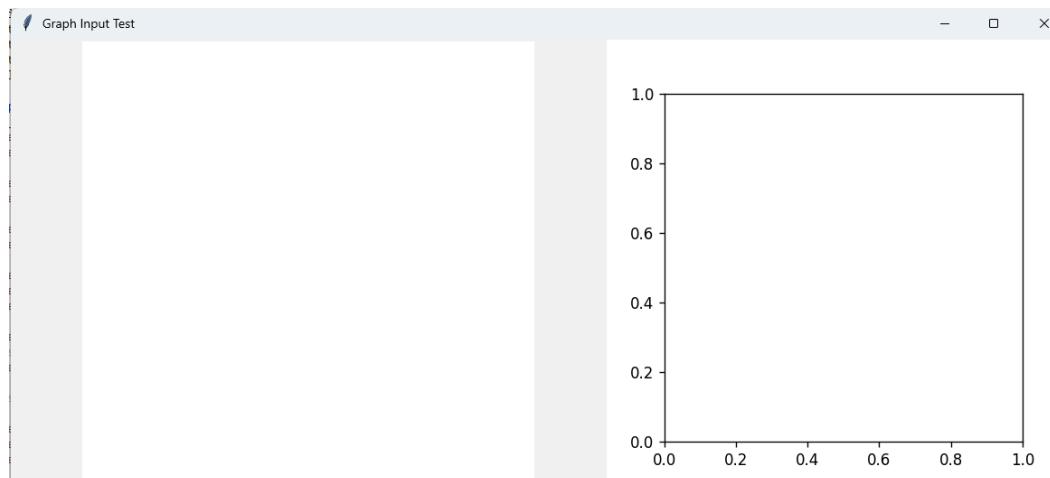
3.1.g

```

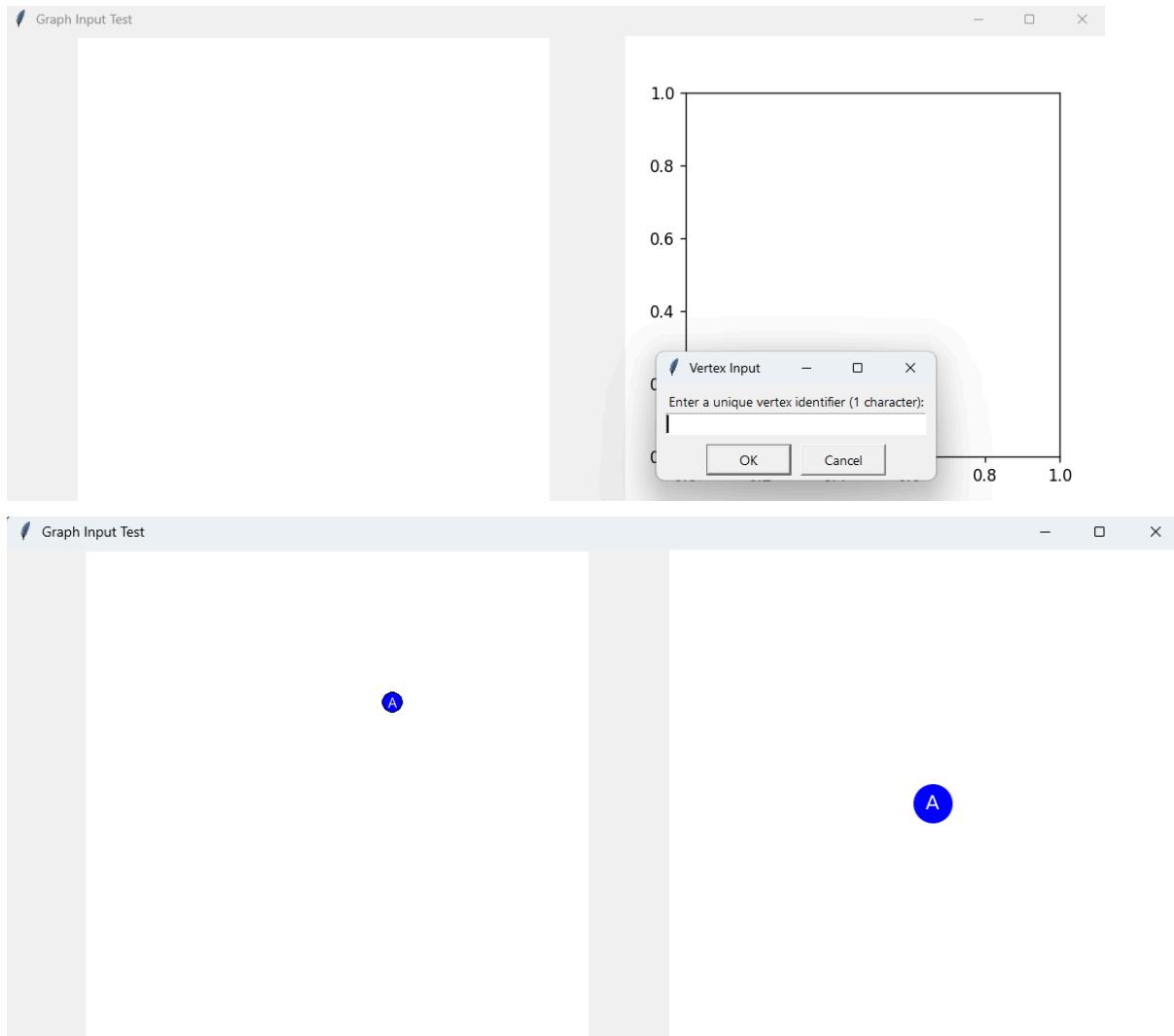
if __name__ == "__main__":
    root = tk.Tk()
    root.title("Graph Input Test")
    root.geometry("800x500")
    graph_input = GraphInput(root)
    root.mainloop()

```

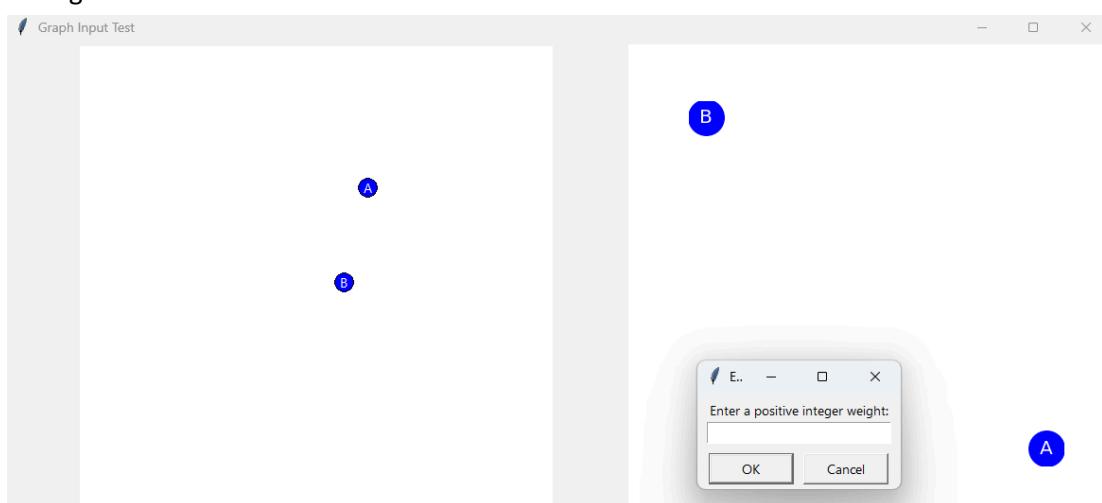
When I run this code, the following is output:

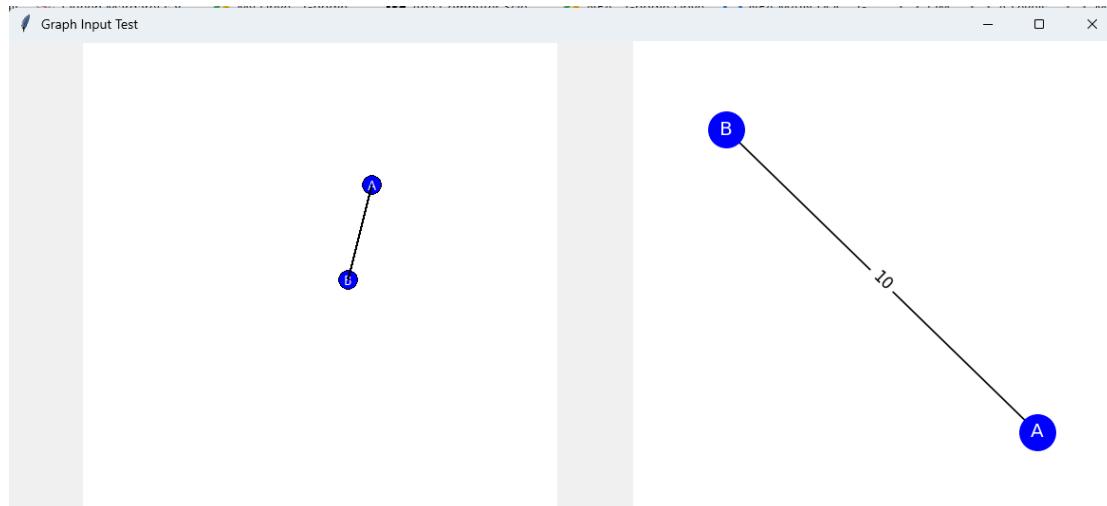


When I double click in the canvas section (on the left), the following is output:



After adding another vertex and clicking both the vertices, the following is output when I try to add an edge:

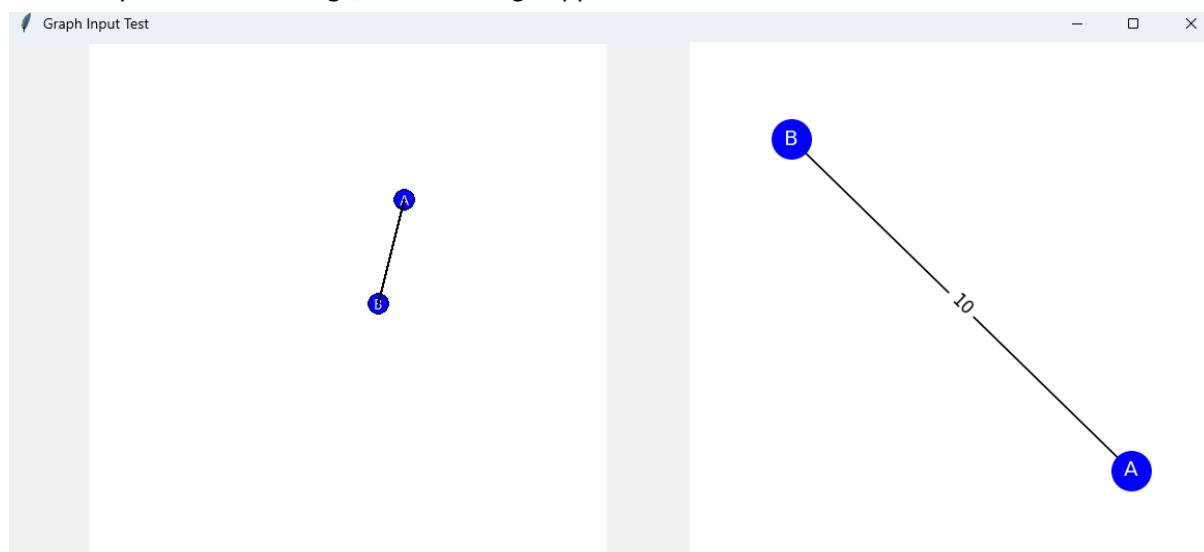




This is as expected on both the canvas and the NetworkX graph. For all of the above checks, the following is output to the shell:

```
in select or create edge
in add vertex
Added vertex: A at position (306, 151)
in update visualisation
in select or create edge
in add vertex
Added vertex: B at position (281, 251)
in update visualisation
in select or create edge
selected vertex: B
in select or create edge
in select or create edge
selected vertex: A
in select or create edge
Created edge between A and B with weight 10
in update visualisation
```

When I try to delete the edge, the following happens:



```

in select or create edge
in add vertex
Added vertex: A at position (306, 151)
in update visualisation
in select or create edge
in add vertex
Added vertex: B at position (281, 251)
in update visualisation
in select or create edge
selected vertex: B
in select or create edge
in select or create edge
selected vertex: A
in select or create edge
Created edge between A and B with weight 10
in update visualisation
in select or create edge
Selected edge: ('A', 'B') for deletion

```

The program is in the correct section however the edge is not deleted off the canvas or graph. I was not sure why this was happening so I researched the issue. I found that this may be due to an issue with the canvas not being focused. Therefore I added the following code:

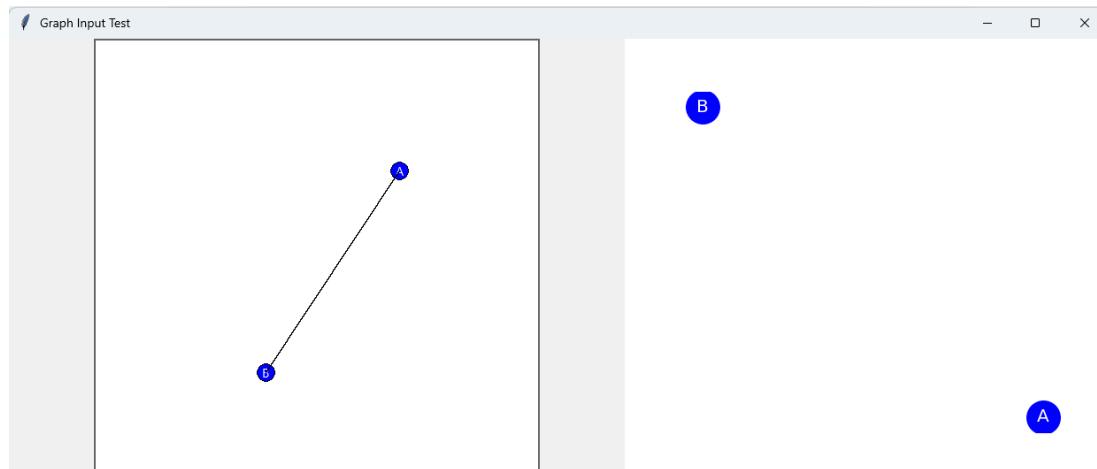
3.1.h

```

self.canvas.focus_set()
# set up the keyboard/user clicks/inputs:
self.canvas.bind("<Double-Button-1>", self.add_vertex)
self.canvas.bind("<Button-1>", self.select_or_create_edge)
self.canvas.bind("<Delete>", self.delete_element)

```

When I now run the code, the following happens:



```

in select or create edge
in add vertex
Added vertex: A at position (345, 149)
in update visualisation
in select or create edge
in add vertex
Added vertex: B at position (194, 377)
in update visualisation
in select or create edge
selected vertex: B
in select or create edge
Created edge between B and A with weight 10
in update visualisation
in select or create edge
Selected edge: ('A', 'B') for deletion
in delete element
Selected edge: ('A', 'B') for deletion
Deleted edge between A and B
in update visualisation

```

The edge is now correctly deleted from the graph and the Shell shows that it should also be deleted from the canvas. However it is still there. After looking at the code, I thought that the canvas should be updated, just as the graph is in the update visualisation method. Therefore, I wrote the following code and called it in the update_visualisation method:

3.1.i

```
def redraw_canvas(self):
    print("Redrawing canvas...")
    self.canvas.delete("all") # Clear the canvas

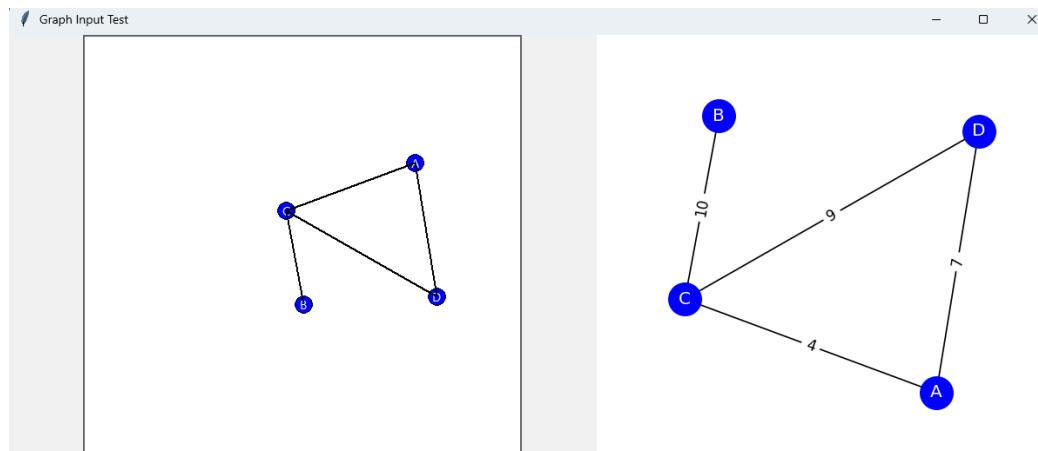
    # Redraw vertices
    for node, (x, y) in nx.get_node_attributes(self.graph, 'pos').items():
        self.canvas.create_oval(x-10, y-10, x+10, y+10, fill='blue', outline='black', tags=node)
        self.canvas.create_text(x, y, text=node, fill='white', tags=node)

    # Redraw edges
    for edge in self.graph.edges:
        x1, y1 = self.graph.nodes[edge[0]]['pos']
        x2, y2 = self.graph.nodes[edge[1]]['pos']
        edge_tag = (edge[0], edge[1])
        self.canvas.create_line(x1, y1, x2, y2, fill="black", width=2, tags=edge_tag)
```

3.1.j

```
def update_visualization(self):
    print("in update visualisation")
    self.ax.clear()
    pos = nx.get_node_attributes(self.graph, 'pos')
    nx.draw(self.graph, pos, with_labels=True, node_color='blue', edge_color='black', node_size=500, font_color='white', ax=self.ax)
    edge_labels = nx.get_edge_attributes(self.graph, 'weight')
    nx.draw_networkx_edge_labels(self.graph, pos, edge_labels=edge_labels, ax=self.ax)
    self.graph_canvas.draw()
    self.redraw_canvas()
```

I then ran the code and the following was output:

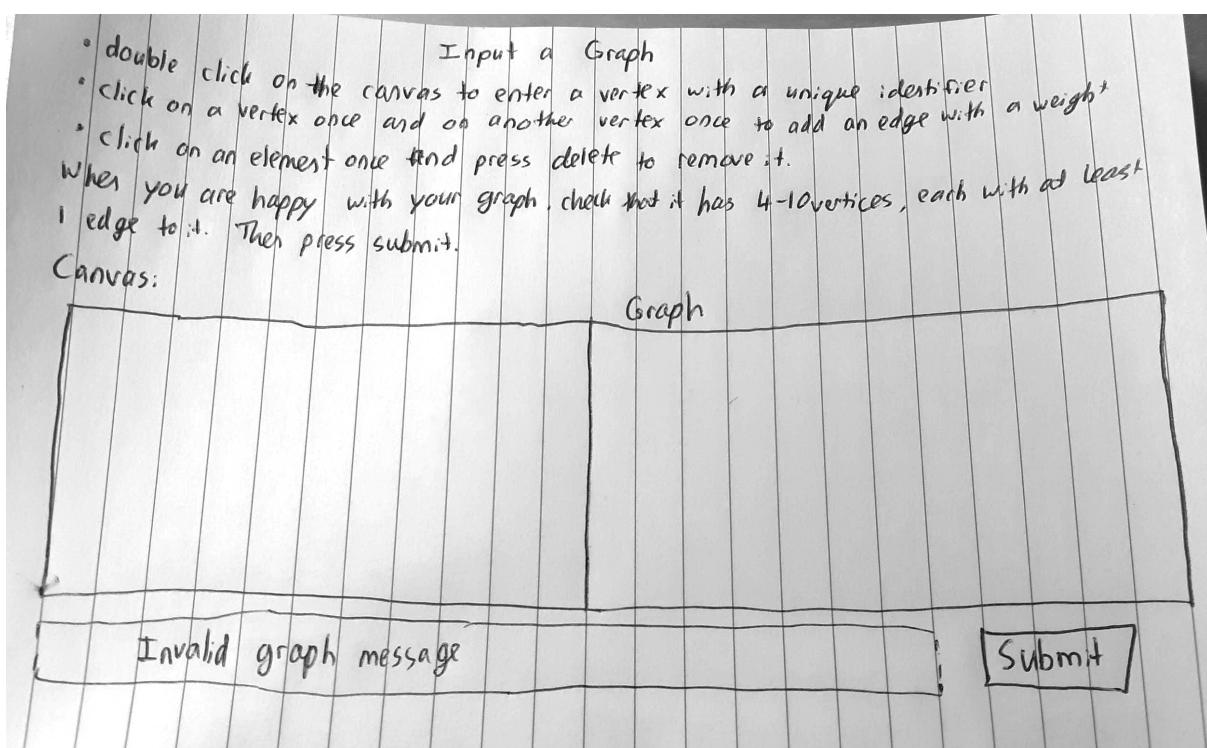


```
>>> %Run graph_input_test_code.py
in select or create edge
in add vertex
Added vertex: A at position (381, 147)
in update visualisation
Redrawing canvas...
in select or create edge
in add vertex
Added vertex: B at position (253, 310)
in update visualisation
Redrawing canvas...
in select or create edge
in select or create edge
in add vertex
Added vertex: C at position (233, 202)
in update visualisation
Redrawing canvas...
in select or create edge
in select or create edge
in add vertex
Added vertex: D at position (406, 301)
in update visualisation
Redrawing canvas...

in select or create edge
selected vertex: D
in select or create edge
Created edge between D and A with weight 7
in update visualisation
Redrawing canvas...
in select or create edge
selected vertex: C
in select or create edge
Created edge between C and D with weight 9
in update visualisation
Redrawing canvas...
in select or create edge
selected vertex: C
in select or create edge
Created edge between C and B with weight 10
in update visualisation
Redrawing canvas...
in select or create edge
selected vertex: A
in select or create edge
Created edge between A and C with weight 4
in update visualisation
Redrawing canvas...
```

Now that the graph input section functions as intended, I was considering how I should approach implementing this into the Prim's and Dijkstra's pages. As seen in the GUI design sections of both of these visualisation pages, there is only space for one graph, which I intended to be the NetworkX graph because the NetworkX library allows the weights of the edges to be displayed, therefore making it easier for the user to understand where the weights being added come from. However, when I developed the graph input section, a Tkinter canvas was required to be added because this would allow for the user to use their mouse to enter the vertices and edges. Therefore, I have decided that when the user wants to input a graph, a new window could be opened, where the canvas and NetworkX graph are. Then, when the user has finished inputting the graph and wants to return to the visualisation page, they can press a button to submit the graph. This causes validation checks on the graph to be completed. If they pass, the user will return to the visualisation page where the NetworkX graph will be displayed. If they do not pass, a message will be displayed saying that the graph is invalid.

As this is a new adjustment to the system, although it still has the same focus as the graph input design section originally had and moved some of the validation subsections from the Prim's and Dijkstra's pages into this new page, I have created a small diagram of what I expect his page to look like. All of the design features common to the other pages are the same, like the colour scheme and font.



Therefore, to complete the graph input page that I described above, I added the following code to make the page appear as shown above, with the GUI design of the other pages:

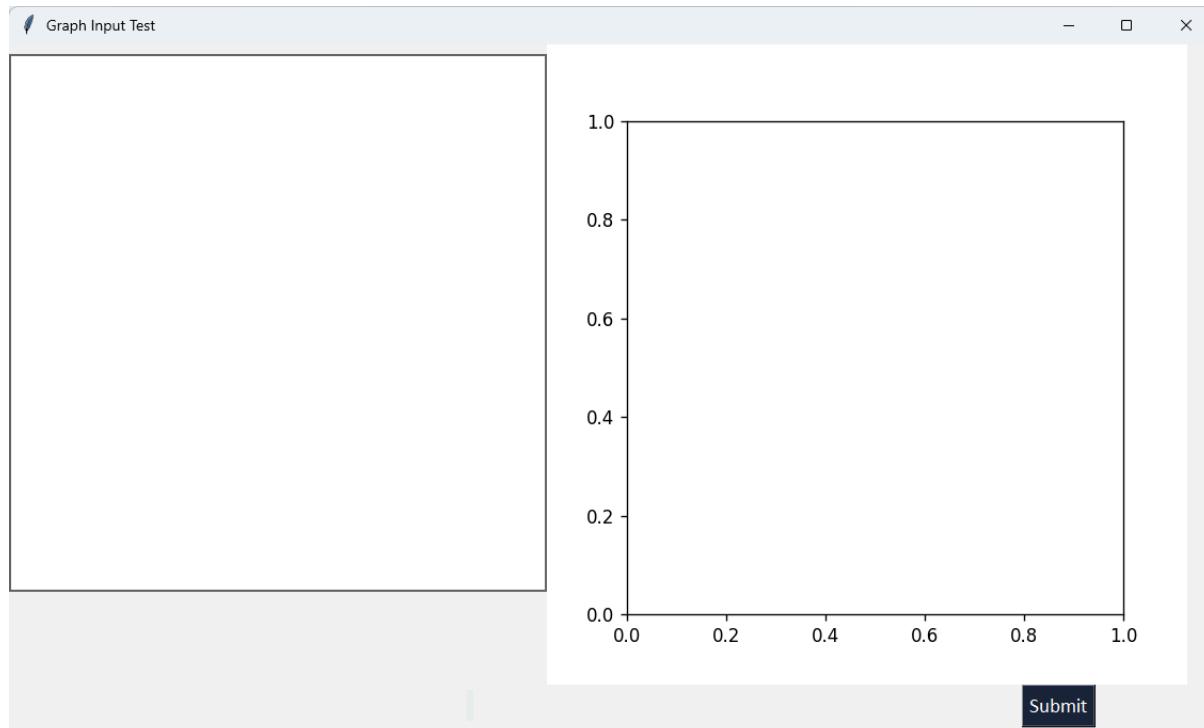
3.1.k

```

39 def graphInputPageWidgets(self):
40     self.graphInputFrame = Frame(self.master, bg="#eaebed")
41
42     Label(self.graphInputFrame, text="Input a Graph", font=fontLarge).grid(row=0, column=0, columnspan=10)
43
44     Label(self.graphInputFrame,
45         text=". Double click on the canvas to add a vertex. Enter a unique identifier.\n"
46         "• Click once on one vertex and once on another vertex to create an edge between them. Enter a weight greater than 0.\n"
47         "• Click on an element once and press delete to remove it."
48         "When you are happy with your graph, check that it has 4-10 vertices, each with at least 1 edge connected to it. Then press submit",
49         font = fontSmall,
50         fg="#1b263b",
51         bg="#eaebed",
52         justify="left").grid(row=1, column=0, columnspan=10)
53
54     Label(self.graphInputFrame, text="Canvas:", font=fontMedium).grid(row=2, column=0, columnspan=2)
55     Label(self.graphInputFrame, text="Graph:", font=fontMedium).grid(row=2, column=5, columnspan=2)
56
57     self.canvasFrame = Frame(self.master, bg="#eaebed")
58     self.canvasFrame.grid(row=3, column=0)
59
60     self.canvas = Canvas(self.canvasFrame, width=500, height=500, bg="white")
61     self.canvas.grid(row=3, column=0, columnspan=5, rowspan=5)
62
63     self.fig, self.ax = plt.subplots(figsize=(5,5))
64     self.graphCanvas = FigureCanvasTkAgg(self.fig, self.master)
65     self.graphCanvas.get_tk_widget().grid(row=3, column=5, columnspan=5, rowspan=5)
66
67     self.invalidMessage = Label(self.master, text="", font=fontMedium, bg="#eaebed").grid(row=8, column=0, columnspan=8)
68
69     Button(self.master, text="Submit", font=fontMedium, fg="white", bg="#1b263b", command=self.validateGraph).grid(row=8, column=8, columnspan=2)
70
71     self.canvas.focus_set()
72     # set up the keyboard/user clicks/inputs:
73     self.canvas.bind("<Double-Button-1>", self.add_vertex)
74     self.canvas.bind("<Button-1>", self.select_or_create_edge)
75     self.canvas.bind("<Delete>", self.delete_element)

```

When I run the code, the following is output:



The Submit button is displayed in the correct place but the other text is not where it should be. Therefore, I tried removing the graphInputFrame that I had created and changing it so that all the elements are just in the master frame. Also to check that the invalidMessage widget is where it should be, I added the text “...” to appear in it.

3.1.l

```

def graphInputPageWidgets(self):
    #self.graphInputFrame = Frame(self.master, bg="#eaebed")

    Label(self.master, text="Input a Graph", font=fontLarge).grid(row=0, column=0, columnspan=10)

    Label(self.master,
          text="• Double click on the canvas to add a vertex. Enter a unique identifier.\n"
               "• Click once on one vertex and once on another vertex to create an edge between them. Enter a weight greater than 0.\n"
               "• Click on an element once and press delete to remove it."
               "When you are happy with your graph, check that it has 4-10 vertices, each with at least 1 edge connected to it. Then press submit",
          font = fontSmall,
          fg="#1b263b",
          bg="#eaebed",
          justify="left").grid(row=1, column=0, columnspan=10)

    Label(self.master, text="Canvas:", font=fontMedium).grid(row=2, column=0, columnspan=2)
    Label(self.master, text="Graph:", font=fontMedium).grid(row=2, column=5, columnspan=2)

    self.canvasFrame = Frame(self.master, bg="#eaebed")
    self.canvasFrame.grid(row=3, column=0)

    self.canvas = Canvas(self.canvasFrame, width=500, height=500, bg="white")
    self.canvas.grid(row=3, column=0, columnspan=5, rowspan=5)

    self.fig, self.ax = plt.subplots(figsize=(5,5))
    self.graphCanvas = FigureCanvasTkAgg(self.fig, self.master)
    self.graphCanvas.get_tk_widget().grid(row=3, column=5, columnspan=5, rowspan=5)

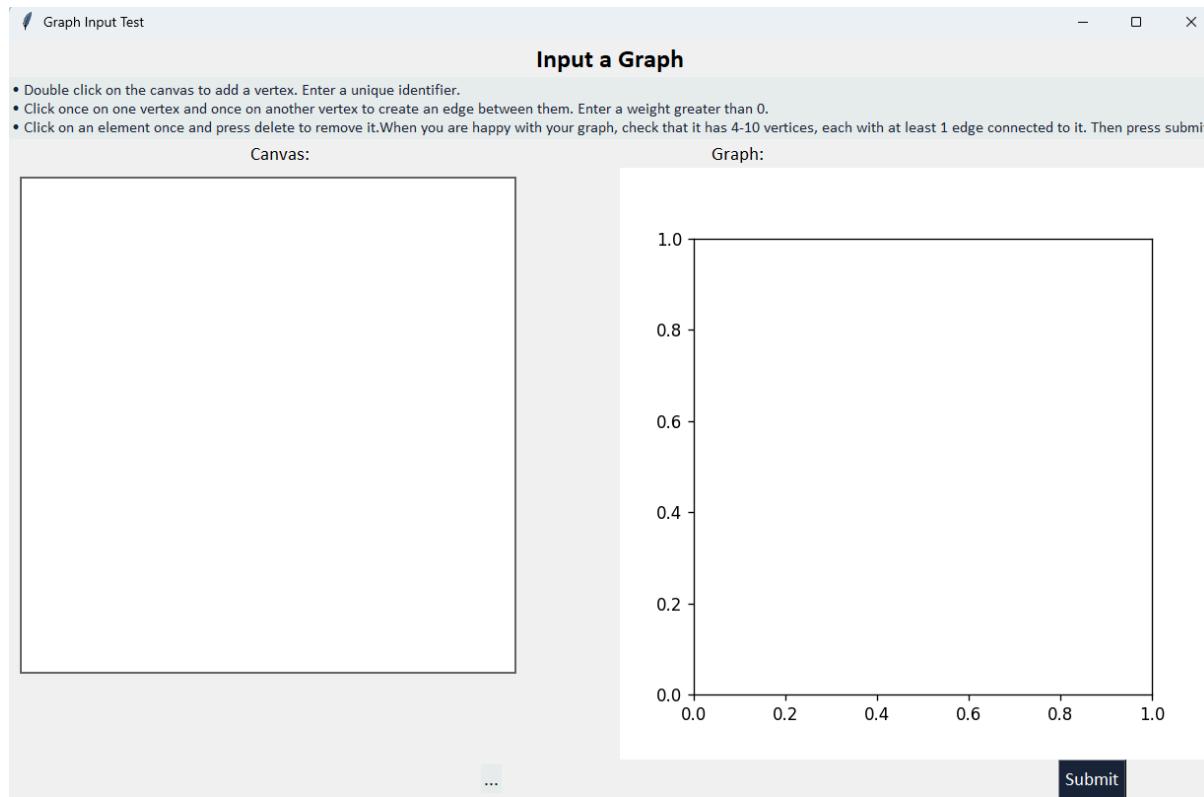
    self.invalidMessage = Label(self.master, text="...", font=fontMedium, bg="#eaebed").grid(row=8, column=0, columnspan=8)

    Button(self.master, text="Submit", font=fontMedium, fg="white", bg="#1b263b", command=self.validateGraph).grid(row=8, column=8, columnspan=2)

    self.canvas.focus_set()
    # set up the keyboard/user clicks/inputs:
    self.canvas.bind("<Double-Button-1>", self.add_vertex)
    self.canvas.bind("<Button-1>", self.select_or_create_edge)
    self.canvas.bind("<Delete>", self.delete_element)

```

This is what the page now looks like:



This is as expected. I then changed the master to have background colour #EAEBED, like the other sections. I also changed the lines in the text to make sure that there is not as much space between the canvas and the graph.

3.1.m

```

Label(self.master,
      text="• Double click on the canvas to add a vertex. Enter a unique identifier.\n"
           "• Click once on one vertex and once on another vertex to create an edge between them. Enter a weight greater than 0.\n"
           "• Click on an element once and press delete to remove it.\n"
           "When you are happy with your graph, check that it has 4-10 vertices, each with at least 1 edge connected to it. Then press submit",
      font = fontMedium,
      fg="#1b263b",
      bg="#eaebed",
      justify="left").grid(row=1, column=0, columnspan=10, sticky="w")

```

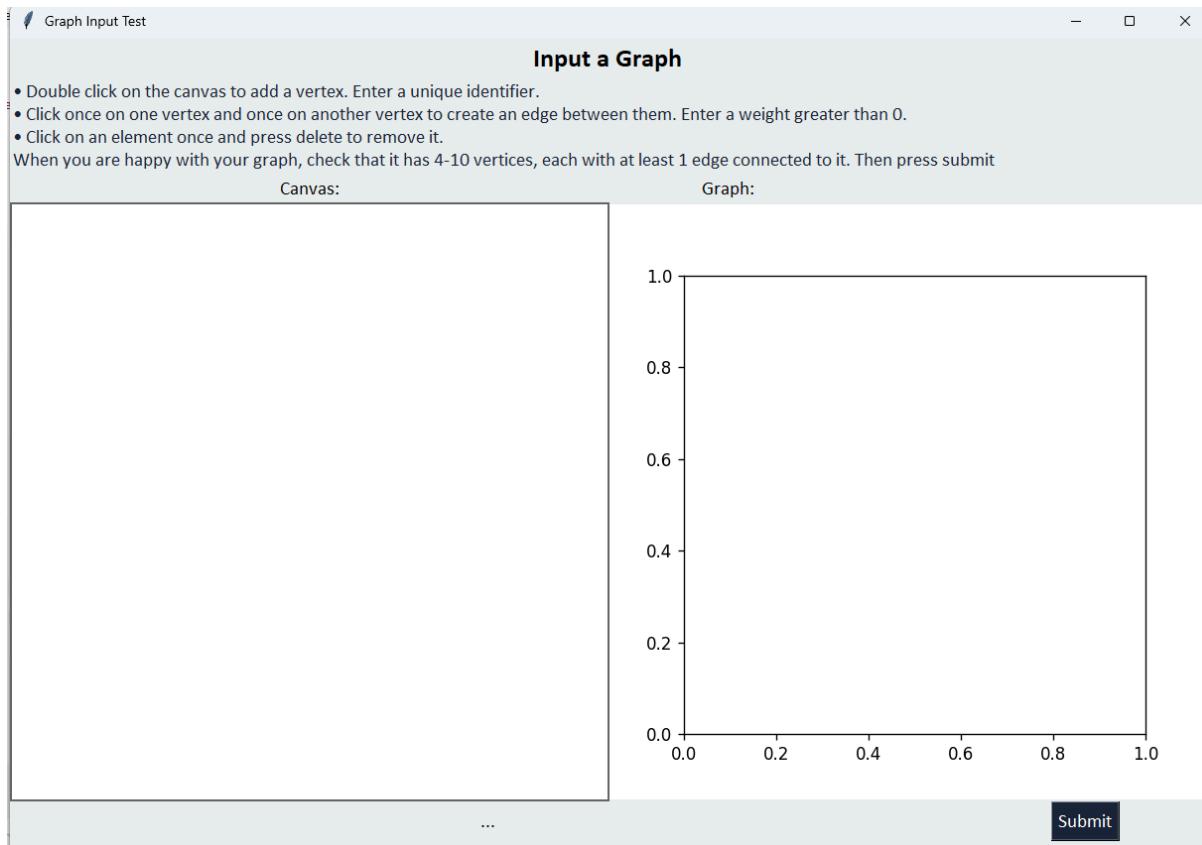
3.1.n

```

self.canvas = Canvas(self.canvasFrame, width=600, height=600, bg="white")
self.canvas.grid(row=3, column=0, columnspan=5, rowspan=5)

```

When the code is now run, the following is displayed:



Then, I configured the window to be the correct size:

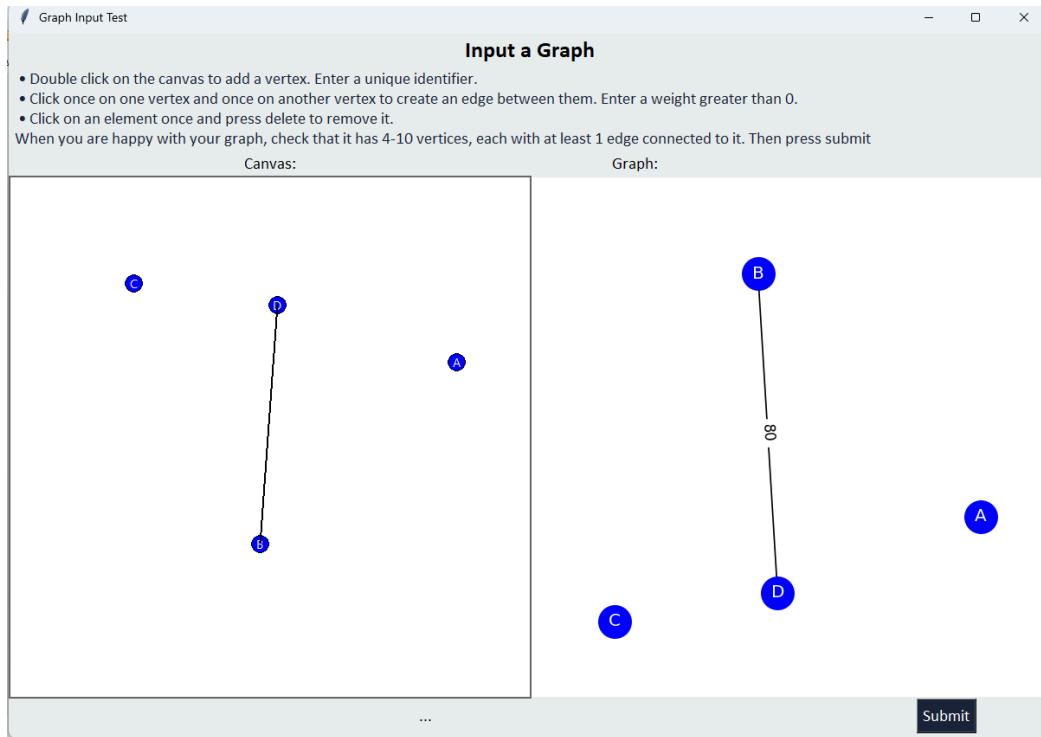
3.1.o

```

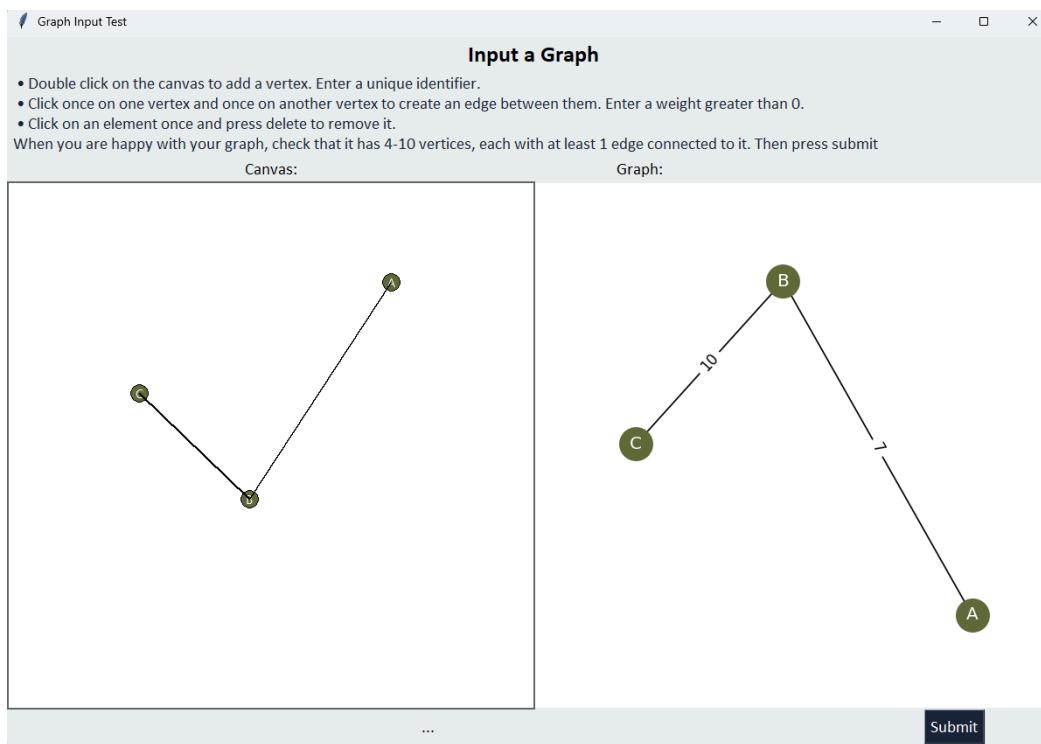
def graphInputPageWidgets(self):
    #self.graphInputFrame = Frame(self.master, bg="#eaebed")
    self.master.configure(bg="#eaebed")
    self.master.geometry("1200x815")

```

Now the window fits the widgets inside it well. Below is a screenshot with the graph functionality in it to demonstrate that all functionality has been maintained:



The final task to complete regarding the GUI is to ensure that the colour of the vertices is the green colour from the colour palette defined for this system:



The GUI has now been completed for the graph input page. The next task is to complete the validation method for the graph. I initially started by writing the following code:

3.1.p

```

def validateGraph(self):
    print("in validate graph")

    # check for 4 to 10 vertices:
    if len(self.graph.nodes) < 4 or len(self.graph.nodes) > 10:
        self.invalidMessage["text"] = "Invalid graph: Ensure that the graph has between 4 and 10 vertices."
        return False

    # check each vertex has at least 1 edge to it
    for vertex in self.graph.nodes:
        if len(list(self.graph.neighbors(vertex))) == 0:
            self.invalidMessage["text"] = "Invalid graph: Ensure that each vertex has at least 1 edge connected to it."
            return False

    # check weight edges greater than 0
    for start, end, data in self.graph.edges(data=True):
        if data.get("weight", 0) <= 0:
            self.invalidMessage["text"] = "Invalid graph: Ensure that the weight of each edge is greater than 0."
            return False

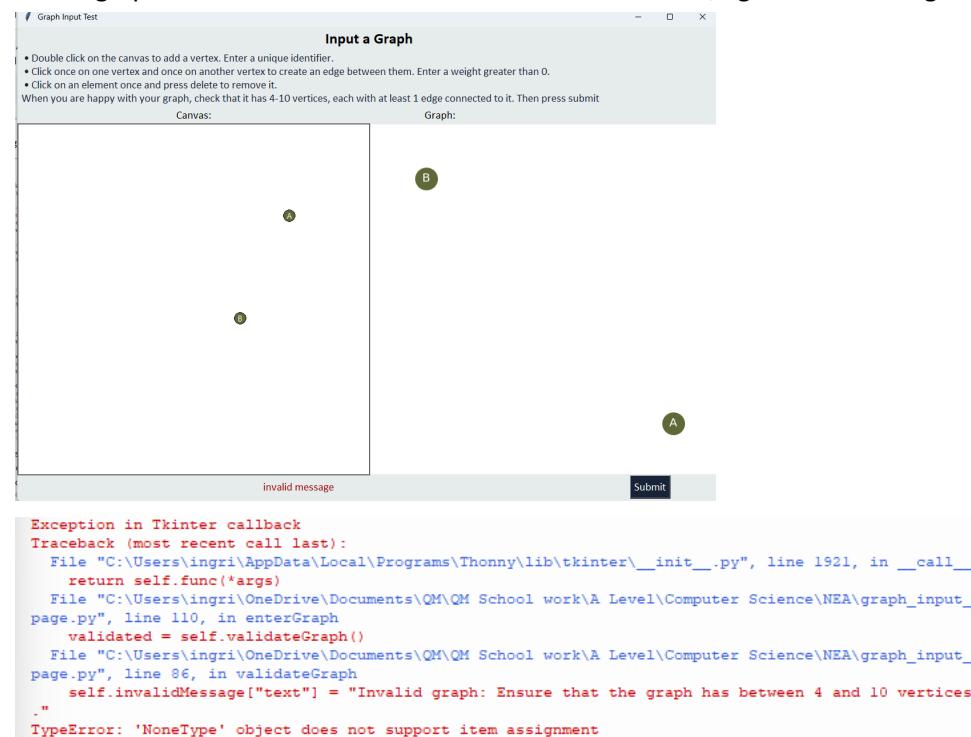
    self.invalidMessage["text"] = " "
    return True

def createAdjacencyMatrix(self):
    matrix = nx.to_numpy_array(self.graph, weight="weight")
    return matrix

def enterGraph(self):
    print("in enter Graph")
    validated = self.validateGraph()
    print(validated)
    if validated:
        matrix = self.createAdjacencyMatrix()
        print(matrix)
        #return matrix

```

I made use of the features of the NetworkX library to allow me to perform these checks easily. The creation of the adjacency matrix is to enable me to pass this into the Prim's or Dijkstra's pages to allow it to be used for the functionality of those algorithms. When I run the code and submit and invalid graph to test the first check of the number of nodes, I get the following output:

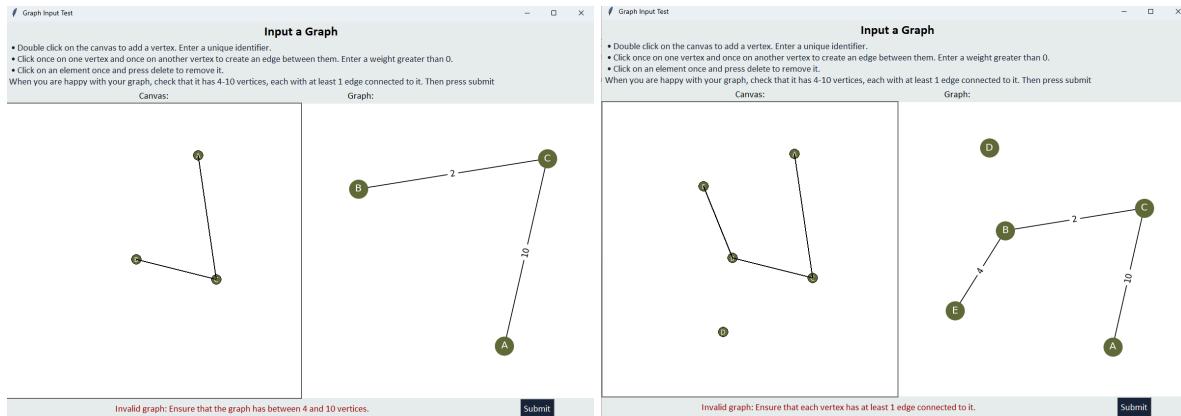


I realised that this error was due to me putting the grid placement of the invalidMessage widget on the same line as the creation of the widget. Therefore, I changed the code to the following in the widgets method:

3.1.q

```
self.invalidMessage = Label(self.master, text="invalid message", font=fontMedium, bg="#eaebed", fg="#990000")
self.invalidMessage.grid(row=8, column=0, columnspan=8)
```

When I now try to run the code, the following are output:



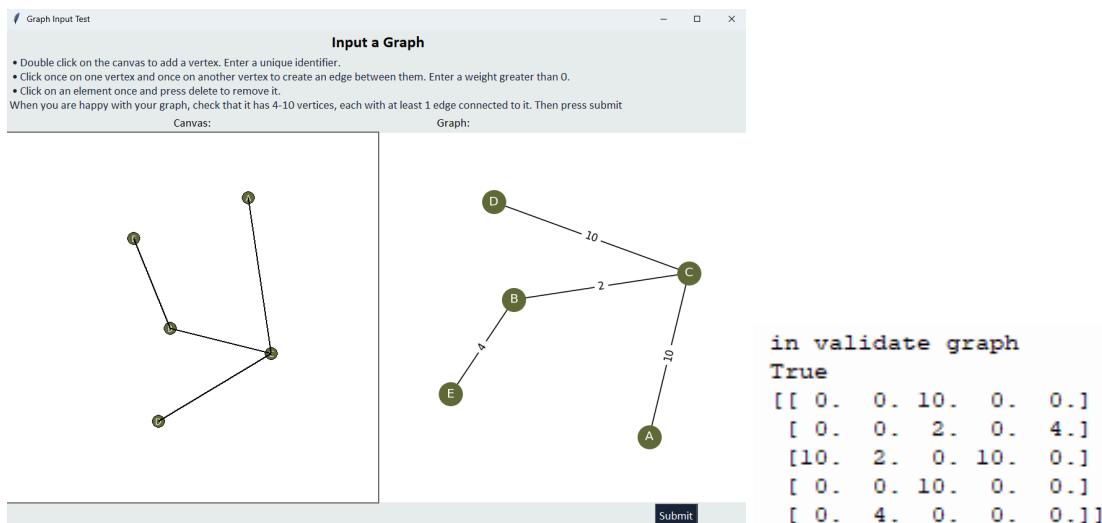
I realised when testing the third check of edges being greater than 0 that when I had written the code for creating edges, I had written the following code:

3.1.r

```
self.target_vertex = node
weight = simpledialog.askinteger("Edge Weight", "Enter a positive integer weight:")
if weight and weight > 0:
    self.graph.add_edge(self.selected_vertex, self.target_vertex, weight=weight)
```

This enforces the rule that the weight must be greater than 0 and stops the edge from being created if the weight is 0. Therefore, I shall take out the code in the validate method because it is redundant. The validateGraph method now becomes:

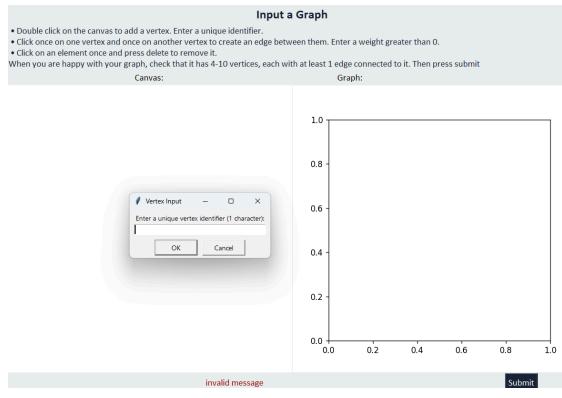
When I test to check that a correct graph is validated correctly, the following is output:

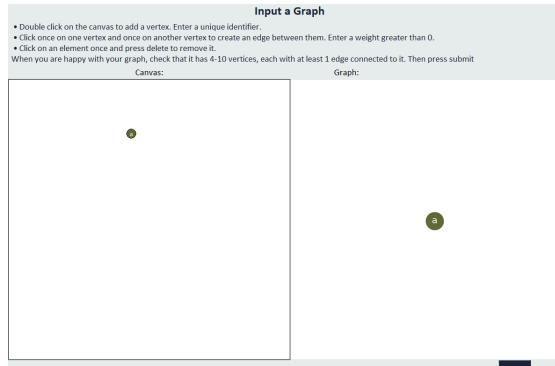
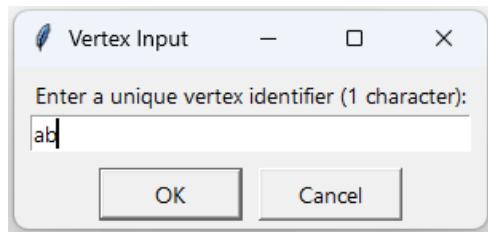
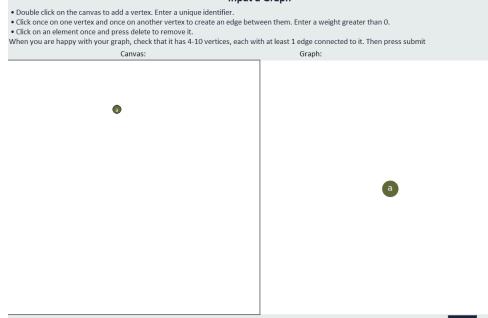


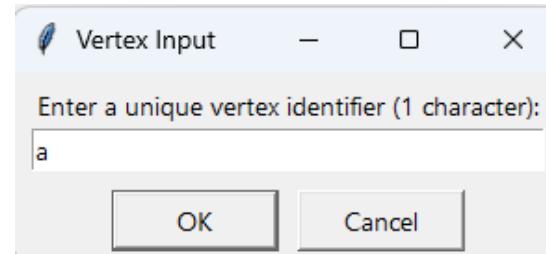
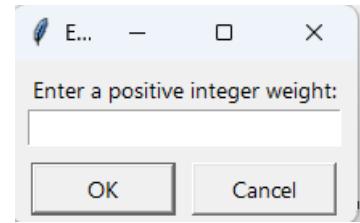
The shell output is the adjacency matrix of the graph, which is stored as a 2d array. For me to better understand how this can be used, I drew the following diagram, which will help me understand how to approach the solutions to Prim's and Dijkstra's algorithms.

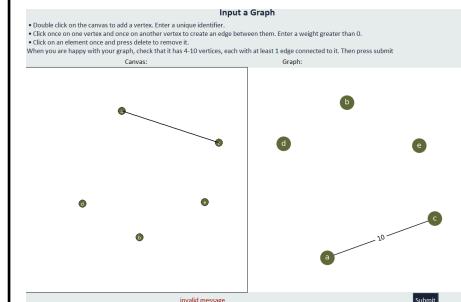
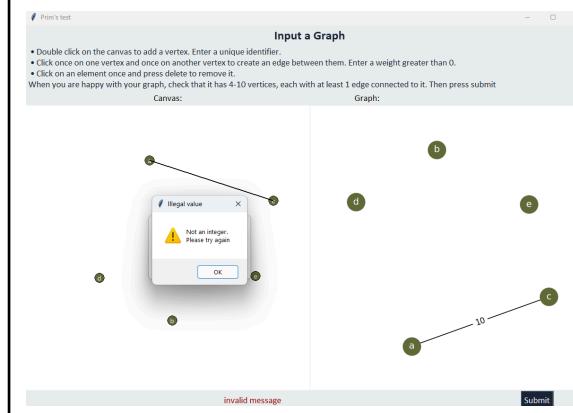
	A	B	C	D	E
A	0	0	10	0	0
B	0	0	2	0	4
C	10	2	0	10	0
D	0	0	10	0	0
E	0	4	0	0	0

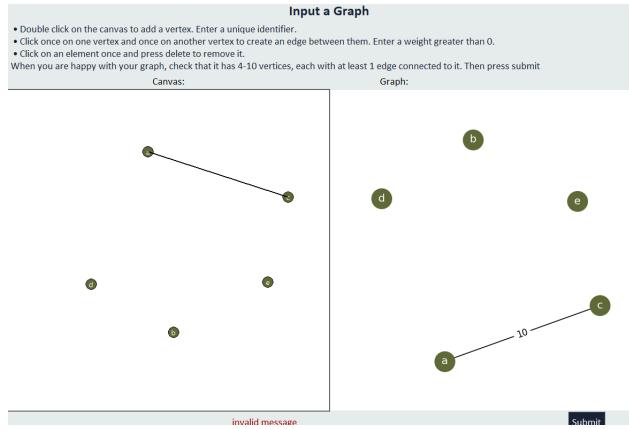
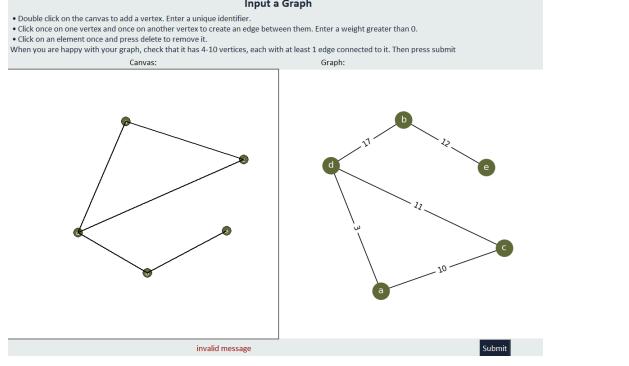
The enterGraph method will be further extended once I have decided how I am approaching the inputting of a graph in each algorithm visualisation section. Now, I will test the code developed using the white box testing table specified in the GRAPH INPUT design section. The validation of the graph input section will be tested in the next iteration in the Prim's Visualisation white box testing.

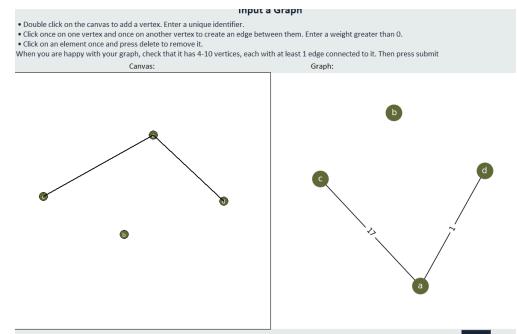
TEST DATA	EXPECTED RESULT	JUSTIFICATION	CODE	ACTUAL OUTPUT
Double click on the canvas	Pop up to enter the vertex ID appears.	Confirms that the functionality to add a vertex works as intended.	3.1.h 3.1.l 3.1.c	 <p>As expected</p>

Enter a single character as the ID	Vertex created and displayed on the graph.	Confirms that a valid input for the vertex ID allows a vertex to be added.	3.1.h 3.1.l 3.1.c 3.1.i 3.1.j	 <p>As expected</p>
Enter multiple characters as the ID	Error message so the vertex is not created.	Confirms that an invalid ID input does not allow a vertex to be created.	3.1.h 3.1.l 3.1.c	  <p>As expected</p>

Enter an already used ID as the vertex ID.	Error message so the vertex is not created.	Ensures that all vertices have unique IDs, therefore confirming that an invalid ID input does not allow a vertex to be created.	3.1.h 3.1.i 3.1.c	  <p>As expected</p>
Hold shift and drag between two vertices.	A pop up to enter the edge weight appears.	Confirms that the functionality to add an edge works as intended.	3.1.h 3.1.i 3.1.d	<p>Change to press 2 vertices as this is the functionality I have implemented for creating an edge:</p>  <p>As expected</p>

Enter a positive integer as the weight.	An edge between the two selected vertices is created.	Confirms that a valid weight input allows an edge to be displayed on the graph.	3.1.h 3.1.l 3.1.d 3.1.i 3.1.j	 <p>As expected</p>
Enter a float/real number for the weight.	An error message appears and the edge is not created between the two vertices.	Confirms that an invalid weight input prevents an edge from being created.	3.1.h 3.1.l 3.1.d	 <p>As expected</p>
Enter a negative number for the weight.	An error message appears and the edge is not created between the two vertices.	Confirms that an invalid weight input prevents an edge from being created.	3.1.h 3.1.l 3.1.d	Input: -9

				 <p>An error message doesn't appear however an edge is not displayed so it seems that this is not an issue.</p>
Select a vertex and press delete.	The vertex is removed from the graph, as well as any connected edges.	Confirms that the functionality to remove a vertex works as intended.	3.1.h 3.1.l 3.1.e 3.1.i 3.1.j	 <p>Not as expected as the vertex is not deleted from the graph.</p>

Select an edge and press delete.	The edge is removed from the graph.	Confirms that the functionality to delete an edge works as intended.	3.1.h 3.1.l 3.1.d 3.1.e 3.1.i 3.1.j	<p>input a graph</p> <p>• Double click on the canvas to add a vertex. Enter a unique identifier. • Click once on one vertex and once on another vertex to create an edge between them. Enter a weight greater than 0. • Click on an element once and press delete to remove it.</p> <p>When you are happy with your graph, check that it has 4-10 vertices, each with at least 1 edge connected to it. Then press submit</p> <p>Canvas: </p> <p>Graph:</p> <p>Deleted edge between b and c in update visualisation Redrawing canvas...</p> <p>As expected</p>
----------------------------------	-------------------------------------	--	--	---

As I'm not sure where the issues with the deleting are coming from, I added the following print statements in the vertex deletion section - as this is what I am focusing on for now:

3.1.s

```
def delete_element(self, event):
    print("in delete element")
    if self.selected_vertex is not None:
        print(f"Selected vertex: {self.selected_vertex} for deletion")
        for node, (x, y) in nx.get_node_attributes(self.graph, 'pos').items():
            print("in for")
            print(f"node to delete: {node}")
            print(f"node position : {(x,y)}")
            print(f"event position: ({event.x},{event.y})")
            print(f"Difference: ({abs(event.x - x)},{abs(event.y - y)})")
```

When I run the code to test this, the following is output:

```
in delete element
Selected vertex: a for deletion
in for
node to delete: a
node position : (497, 68)
event position: (679,190)
Difference: (182,122)
```

It is clear from these print statements that the <10 value is far too small for this size of graph. Therefore, I increased it to 500 because a user is unlikely to put elements too close together and because it would account for the margin of error on where they put their mouse. When I run the code now and press delete, the following is output:

The screenshot shows the 'Input a Graph' application interface. It has two main sections: 'Canvas' and 'Graph'. In the 'Canvas' section, there is a single vertex labeled 'b'. In the 'Graph' section, there is no visual representation of the graph. Below the sections is a 'Console' window showing the following output:

```
Input a Graph
• Double click on the canvas to add a vertex. Enter a unique identifier.
• Click once on one vertex and once on another vertex to create an edge between them. Enter a weight greater than 0.
• Click on an element once and press delete to remove it.
When you are happy with your graph, check that it has 4-10 vertices, each with at least 1 edge connected to it. Then press submit
Canvas: Graph:
b
```

Input a Graph

- Double click on the canvas to add a vertex. Enter a unique identifier.
- Click once on one vertex and once on another vertex to create an edge between them. Enter a weight greater than 0.
- Click on an element once and press delete to remove it.

When you are happy with your graph, check that it has 4-10 vertices, each with at least 1 edge connected to it. Then press submit

```
Canvas: Graph:
b
```

```
... -----
Selected vertex: b for deletion
in for
node to delete: b
node position : (409, 388)
event position: (473,422)
Difference: (64,34)
in if
Deleted vertex: b
in update visualisation
Redrawing canvas...
```

This testing now indicates that the issue has been resolved for deleting a vertex.

ITERATION 1 - DEVELOPER REVIEW

Completing the functionality of the graph input section was not very challenging and therefore was developed quickly. However, as discussed in the middle of Iteration 1, I have had to change the plan for how I implement this section in the visualisation pages because the NetworkX graph that I wanted to use does not allow for the user driven events like double clicking. Therefore, I had to add a Tkinter canvas to allow this functionality to be implemented. This has caused me to make some changes as to how I shall implement Prim's and Dijkstra's visualisation pages, although the inputting of a graph has been developed successfully.

ITERATION 2 - PRIM'S VISUALISATION PAGE FUNCTIONALITY AND GUI

In this iteration, I shall complete the Prim's Visualisation page, which will include the setting up of the GUI, the functionality of the algorithm, which will be specifically implemented to make use of the Menu, which was developed in Prototype 2 Iteration 3, and the implementation of the Graph Input window and placement of the submitted graph in the Prim's Visualisation window.

Firstly, as discussed in Iteration 1, I have created an adjacency matrix of the submitted graph, which is currently being held in the local variable matrix. Prim's algorithm on an adjacency matrix is the following, if it were to be carried out on an adjacency matrix on paper.

1. Choose a starting vertex and delete the row corresponding to that vertex.
2. Label the column corresponding to the start vertex with 1 and ring the smallest undeleted entry in that column. If there is a choice, pick any.
3. Delete the row corresponding to the entry just ringed.
4. Label the column corresponding to the entry just ringed with the next label number.
5. Ring the lowest undeleted entry in all labelled columns.
6. Repeat steps 3 - 5 until all rows are deleted. The ringed entries represent the edges in the MST.

To begin development in this Iteration, I shall start by setting up the GUI for the page. I started by taking out the inheritance from HomeMain because this is not required at this point. Then, I wrote the following code to set up the GUI:

3.2.a

```
10 class Prim(): #HomeMain
11     def __init__(self, master):
12         #super().__init__(master, pUsername)
13         self.master = master
14         self.primWidgets()
```

3.2.b

```
17 def primWidgets(self):
18     self.title = Label(self.master, text="Prim's Algorithm Visualisation")
19     self.title.grid(row=0, column=0, columnspan=5) # change columnspan
20
21     Button(self.master, text="Home", command=self.openHome, bg="#1b263b", fg="white", font=fontMedium, width=15).grid(row=0, column=10, columnspan=2)
22
23     self.primFrame = Frame(self.master, bg="#eaebcd")
24     self.primFrame.grid(row=1, column=0, columnspan=12, rowspan=12)
25
26     Label(self.primFrame,
27           text= "1. Choose any vertex to start the tree.\n"
28           "2. Choose the edge of least weight that joins a vertex that is already in the tree to a vertex that is not yet in the tree.\n"
29           "3. Repeat step 2 until all the vertices are connected to the tree.",
30           font=fontSmall,
31           fg="#1b263b",
32           bg="#eaebcd",
33           justify="left").grid(row=1, column=0, columnspan=12, rowspan=3, sticky="w")
34
35     Label(self.primFrame,
36           text="In order to input a graph, press the 'Input a graph' button, which will open the window to create the graph.\n"
37           "Then choose a start vertex from the list and visualise Prim's algorithm on the graph.",
38           font=fontSmall,
39           fg="#1b263b",
40           bg="#eaebcd",
41           justify="left").grid(row=4, column=0, columnspan=12, rowspan=2, sticky="w")
42
43     self.graphInputButton = Button(self.primFrame, text="Input a graph", bg="#1b263b", fg="white", font=fontMedium, width=20)
44     self.graphInputButton.grid(row=6, column=1, columnspan=2)
45
46     Label(self.primFrame, text="Start vertex:", bg="#eaebcd", fg="#1b263b", font=fontMedium).grid(row=6, column=8, columnspan=2)
47     #OptionsMenu(self.primFrame) # check how to do this once have the graph from the graph input
48
49     Label(self.primFrame, text="MST edges:", bg="#eaebcd", fg="#1b263b", font=fontMedium).grid(row=7, column=8, columnspan=4)
50     self.visualiseText = Text(self.primFrame, width=30, height=10)
51     self.visualiseText.grid(row=8, column=8, columnspan=4, rowspan=4)
52
53     Button(self.primFrame, text="Visualise", command=self.validate, bg="#1b263b", fg="white", font=fontMedium, width=20)
```

I additionally set up the following methods. The openHome method has already been completed. The graphInputWindow method is specifically for the functionality of displaying the graph in the Prim's Visualisation window. The validate method is for checking that a start vertex has been chosen and then beginning the animation. The other 3 methods are specifically for the functionality of the visualisation based on how the Menu is designed to work.

3.2.c

```
def openHome(self):
    closeOpen(self.master, "home")

def graphInputWindow(self):
    # open graph input window (but don't close current window)
    #destroy graphInputButton
    # attributes adjacency matrix
    # display network x graph in row=6, column=0, columnspan=7, rowspan=6
    print("in graph input window")

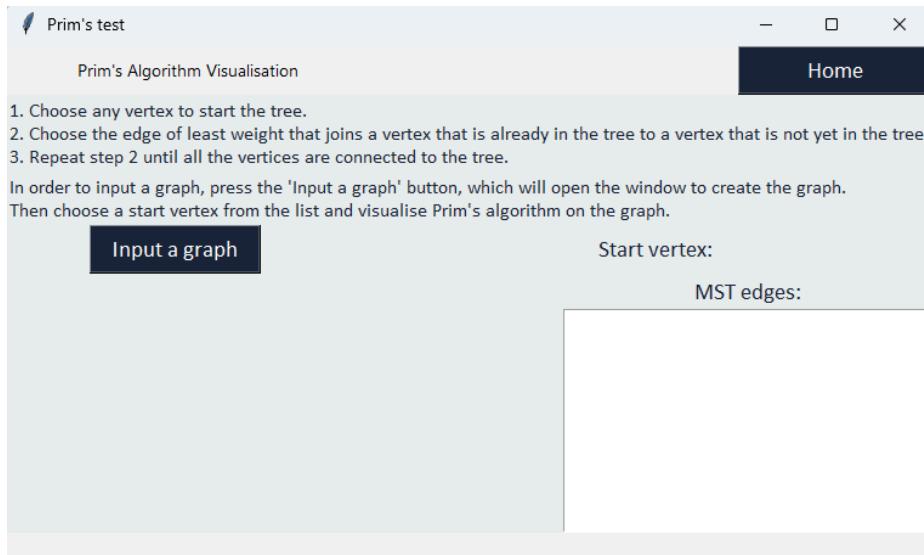
def validate(self):
    print("in validate prim's")
    # checks that a start vertex has been input
    # then starts finding steps

def updatePrimGraph(self):
    print("in update graph")
    # will change colours of current vertex and joined edges

def primSteps(self):
    # return steps in 2d array or dicitionary
    print("in prim steps")

def primAnimate(self):
    # animates at start when not paused
    print("in primAnimate")
```

When run, the GUI appears like so:



I shall now complete the linking of the GraphInput page and the Prim's Visualisation pages.

- ‘Input a graph’ button opens GraphInput window
- When ‘Submit’ is pressed, close the GraphInput window and display the NetworkX graph in place of the ‘Input a graph’ button.
- OptionsMenu made up of vertices in the graph

Firstly, I made the Prim class inherit from the GraphInput class because this would allow me to have access to all the attributes in the GraphInput class:

3.2.d

```

8  from graph_input_page import GraphInput
9
10 class Prim(GraphInput): #HomeMain
11     def __init__(self, master):
12         super().__init__(master, pUsername)
13         super().__init__(master)
14         self.master = master
15
16         self.primWidgets()

```

Then I wrote the following code for getting to the graphInput page:

3.2.e

```

def graphInputWindow(self):
    # open graph input window (but don't close current window)
    print("in graph input window method")
    # hide widgets on prim page
    self.hidePrimWidgets()
    # create graph page to open
    self.graphInputPageWidgets()

```

Then, within the GraphInput page class, I wrote the following code to allow the page to return to the prim page.

3.2.f

```

def backToPage(self):
    if self.validated == True:
        self.graphInputHideWidgets()
        if hasattr(self, "primWidgets"):
            self.primWidgets()
            self.primExtraWidgets()
            # get adjacency matrix to use
            self.adjacencyMatrix = self.createAdjacencyMatrix()
            # extend for dijkstra's

```

This requires some widgets to be changed because a NetworkX graph is now being displayed on the Prim page so I wrote the following code for that:

3.2.g

```

def primExtraWidgets(self):
    # destroy button so axes can go there
    self.graphInputButton.destroy()
    # create axes to plot graph on
    self.fig, self.ax = plt.subplots(figsize=(5,5))
    self.primGraphCanvas = FigureCanvasTkAgg(self.fig, self.master)
    self.primGraphCanvas.get_tk_widget().grid(row=6, column=0, columnspan=5, rowspan=5)
    # draw graph on axes
    pos = nx.get_node_attributes(self.graph, 'pos')
    nx.draw(self.graph, pos, with_labels=True, node_color='#606c38', edge_color='black', node_size=500, font_color='white', ax=self.ax)
    edge_labels = nx.get_edge_attributes(self.graph, 'weight')
    nx.draw_networkx_edge_labels(self.graph, pos, edge_labels=edge_labels, ax=self.ax)
    self.graphCanvas.draw()

```

All of these methods make use of the following methods, which allow the widgets to be hidden (not displayed):

3.2.h

3.2.i

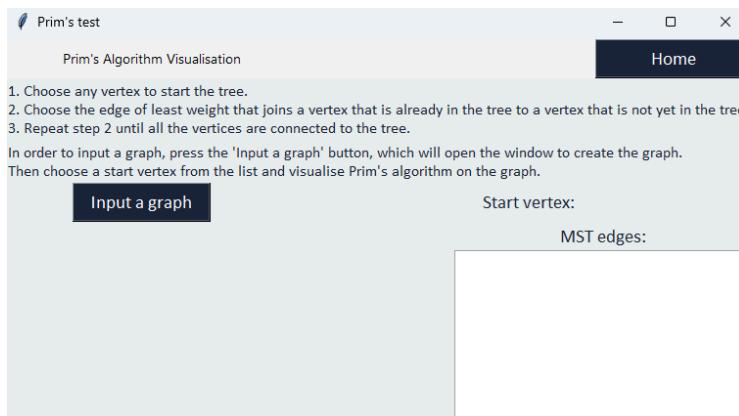
```

def hidePrimWidgets(self):
    for widget in self.primFrame:
        widget.grid_forget()
        self.title.grid_forget()

def graphInputHideWidgets(self):
    for widget in self.master:
        widget.grid_forget()

```

When I run the code for the Prim's page, the following is output:



Then, when I press the 'Input a graph' button, I get the following error:

```
in graph input window method
Exception in Tkinter callback
Traceback (most recent call last):
  File "C:\Users\ingri\AppData\Local\Programs\Thonny\lib\tkinter\__init__.py", line 1921, in __call__
    return self.func(*args)
  File "C:\Users\ingri\OneDrive\Documents\QM\QM School work\A Level\Computer Science\NEA\prim_page.py", line 66, in graphInputWindow
    self.hidePrimWidgets()
  File "C:\Users\ingri\OneDrive\Documents\QM\QM School work\A Level\Computer Science\NEA\prim_page.py", line 87, in hidePrimWidgets
    for widget in self.primFrame:
  File "C:\Users\ingri\AppData\Local\Programs\Thonny\lib\tkinter\__init__.py", line 1681, in cget
    return self.tk.call(self._w, 'cget', '-' + key)
TypeError: can only concatenate str (not "int") to str
```

I realised that this was due to the inability to iterate over a Frame, which I was trying to do in the hide methods. Therefore, I changed the code to the following:

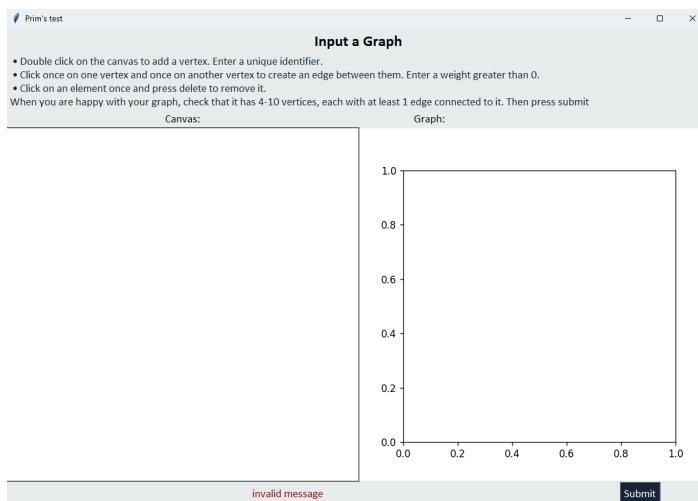
3.2.j

```
def hidePrimWidgets(self):
    for widget in self.primFrame.winfo_children():
        widget.grid_forget()
    self.title.grid_forget()
```

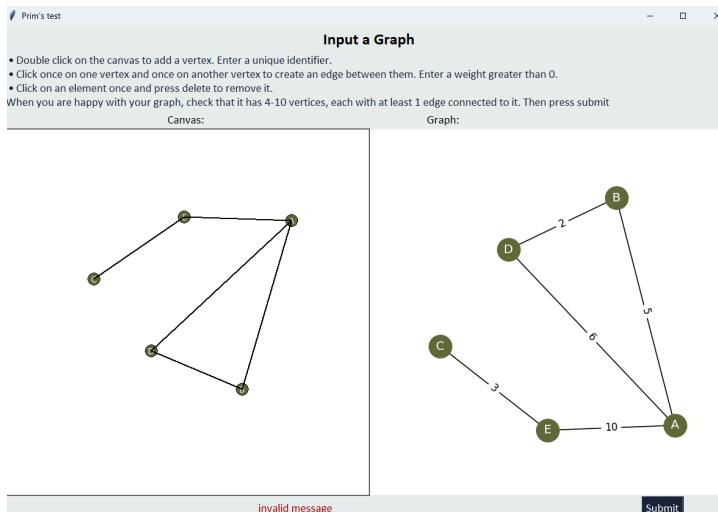
3.2.k

```
def graphInputHideWidgets(self):
    for widget in self.master.winfo_children():
        widget.grid_forget()
```

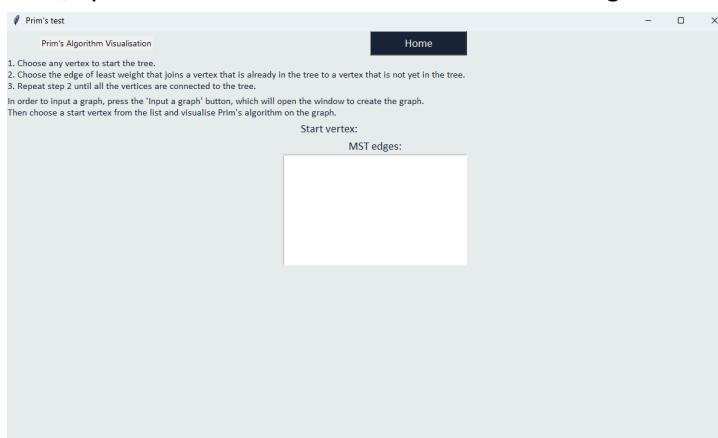
The .winfo_children() method returns a list of all the child widgets within a frame, therefore allowing me to call grid_forget() on each one. Now when I run the code, the following is output after pressing the 'Input a graph' button:



This is as intended so I input the following graph, which should be validated as correct:



Then, I pressed the submit button and the following was output:



```

in validate_graph
Exception in Tkinter callback
Traceback (most recent call last):
  File "C:\Users\ingri\AppData\Local\Programs\Thonny\lib\tkinter\_init_.py", line 1921, in _call_
    return self._func(*args)
  File "C:\Users\ingri\OneDrive\Documents\QM\QM School work\A Level\Computer Science\NEA\graph_input_page.py", line 104, in validateGraph
    self.backToPage()
  File "C:\Users\ingri\OneDrive\Documents\QM\QM School work\A Level\Computer Science\NEA\graph_input_page.py", line 111, in backToPage
    self.primExtraWidgets()
  File "C:\Users\ingri\OneDrive\Documents\QM\QM School work\A Level\Computer Science\NEA\prim_page.py", line 75, in primExtraWidgets
    self.fig, self.ax = plt.subplots(figsize=(5,5))
NameError: name 'plt' is not defined

```

The switching of pages seems to have been developed correctly. However, as the error message states, I have not defined plt. I realised that this was because I hadn't imported matplotlib in this file and because I usually define the library to be used as plt. Therefore I added the following line at the beginning of the code:

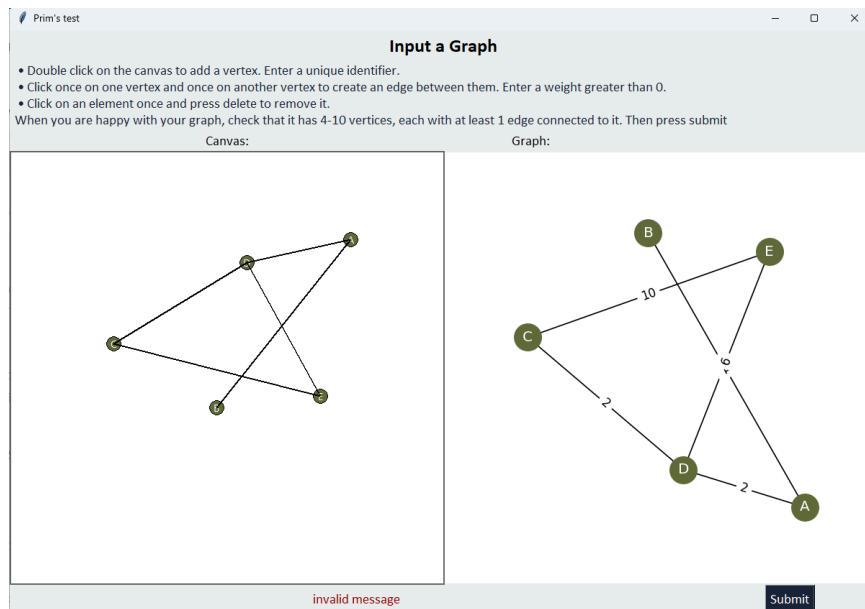
3.2.1

```

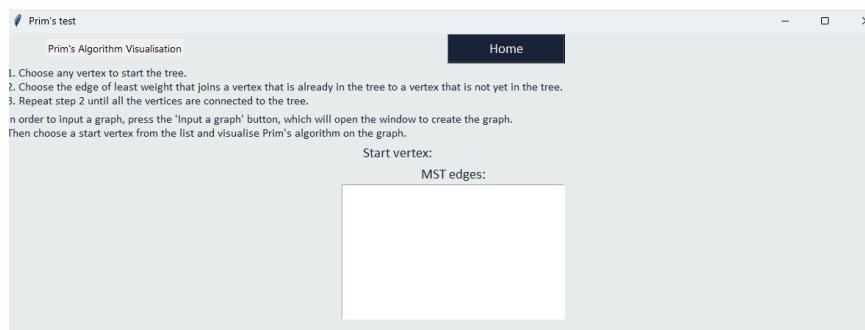
1 # imports specific to the Prim class
2 from tkinter import *
3 import tkinter as tk
4 #from home_page import HomeMain
5 #from NEA_main_file import closeOpen
6 from NEA_utilities import closeOpen, fontLarge, fontMedium, fontSmall
7
8 from graph_input_page import GraphInput
9 import matplotlib as plt

```

Then, I ran the code again and input the following graph:



When I now press the Submit button, the following is output:



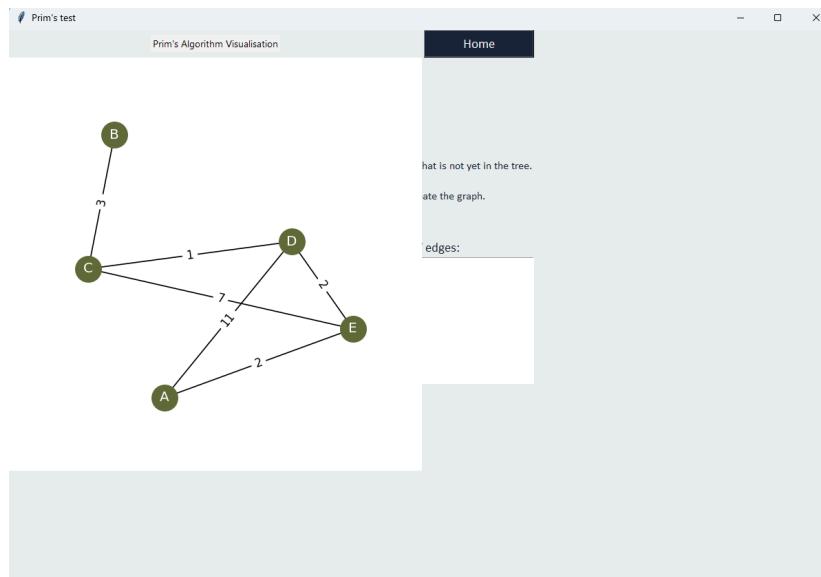
```
Exception in Tkinter callback
Traceback (most recent call last):
  File "C:\Users\ingri\AppData\Local\Programs\Thonny\lib\tkinter\__init__.py", line 1921, in __call__
    return self.func(*args)
  File "C:\Users\ingri\OneDrive\Documents\QM\QM School work\A Level\Computer Science\NEA\graph_input_page.py", line 104, in validateGraph
    self.backToPage()
  File "C:\Users\ingri\OneDrive\Documents\QM\QM School work\A Level\Computer Science\NEA\graph_input_page.py", line 111, in backToPage
    self.primExtraWidgets()
  File "C:\Users\ingri\OneDrive\Documents\QM\QM School work\A Level\Computer Science\NEA\prim_page.py", line 76, in primExtraWidgets
    self.fig, self.ax = plt.subplots(figsize=(5,5))
  File "C:\Users\ingri\AppData\Local\Programs\Thonny\lib\site-packages\matplotlib\api\__init__.py", line 218, in getattr
    raise AttributeError()
AttributeError: module 'matplotlib' has no attribute 'subplots'
```

This was again a quick fix because I hadn't imported matplotlib nor NetworkX correctly within the prim_page file:

3.2.m

```
1 # imports specific to the Prim class
2 from tkinter import *
3 import tkinter as tk
4 #from home_page import HomeMain
5 #from NEA_main_file import closeOpen
6 from NEA_utilities import closeOpen, fontLarge, fontMedium, fontSmall
7
8 from graph_input_page import GraphInput
9 import matplotlib.pyplot as plt
10 from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
11 import networkx as nx
```

Now when I run the code, the following is output after pressing the submit button:



The program flow is clearly correct as the graph that I had input is being displayed correctly on the Prim's Algorithm Visualisation page. The fixes that I now have to make are to ensure that the graph does not take up the whole page and doesn't cover up any of the other widgets. I did this with the following code:

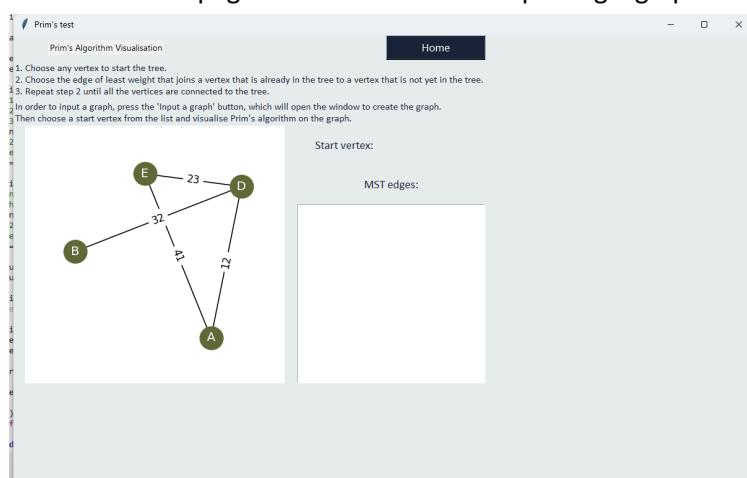
3.2.n

```
def primExtraWidgets(self):
    # destroy button so axes can go there
    self.graphInputButton.destroy()
    # create axes to plot graph on
    self.fig, self.ax = plt.subplots(figsize=(3.5,3.5))
    self.primGraphCanvas = FigureCanvasTkAgg(self.fig, self.primFrame)
    self.primGraphCanvas.get_tk_widget().grid(row=6, column=0, rowspan=6, columnspan=8)
```

3.2.o

```
Label(self.primFrame, text="MST edges:", bg="#eaebed", fg="#1b263b", font=fontMedium).grid(row=7, column=8, columnspan=4)
self.visualiseText = Text(self.primFrame, width=30, height=16)
self.visualiseText.grid(row=8, column=8, columnspan=4, rowspan=4)
```

This is what the page now looks like after inputting a graph.



Now, I am going to create a drop down menu of all the vertices in the graph. To do this, I set up a list called startVertexOptions and an attribute called startVertexChoice as values for the OptionMenu to take in order to be set up. Then I wrote the following code to set up the OptionMenu widget:

3.2.p

```
self.startVertexOptions = ["Select"] # initially Select - changes to the list of vertices once graph input
self.startVertexChoice = StringVar(self.primFrame)
self.startVertexChoice.set("Select")

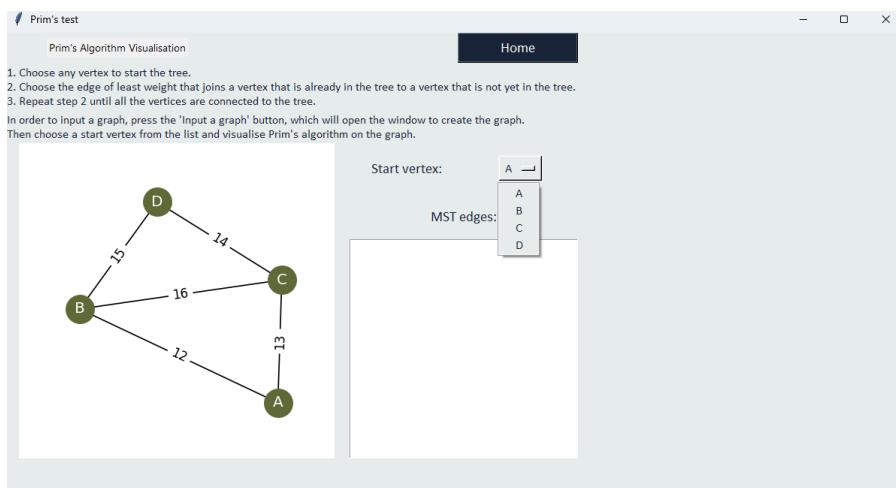
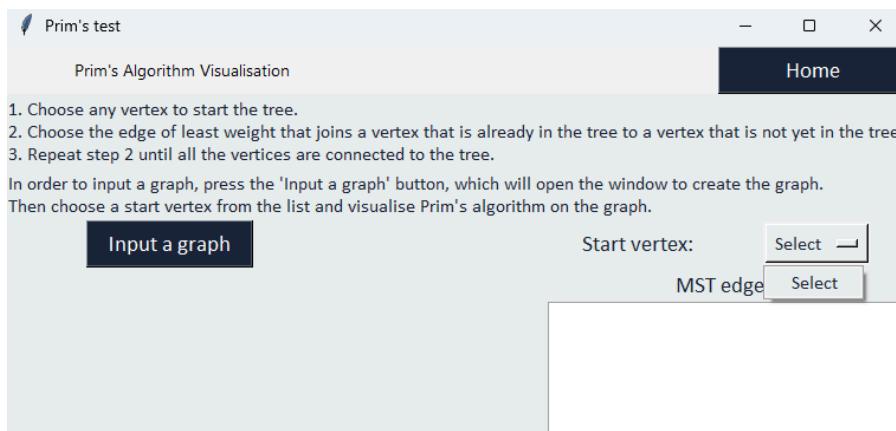
self.startVertexDropdown = OptionMenu(self.primFrame, self.startVertexChoice, *self.startVertexOptions)
self.startVertexDropdown.grid(row=6, column=10, columnspan=2)
self.startVertexDropdown.config(font=fontSmall, bg="#eaebed", fg="#1b263b")
self.startVertexDropdown["menu"].config(font=fontSmall, bg="#eaebed", fg="#1b263b")
```

These lines of code are in the primWidgets method. Then, in order to make sure that the vertices are displayed in the dropdown menu, I added the following lines of code:

3.2.q

```
def primExtraWidgets(self):
    self.startVertexOptions = list(self.graph.nodes)
    self.startVertexChoice.set(self.startVertexOptions[0] if self.startVertexOptions else "Select")
    self.startVertexDropdown["menu"].delete(0, "end")
    for vertex in self.startVertexOptions:
        self.startVertexDropdown["menu"].add_command(label=vertex, command=lambda v=vertex: self.startVertexChoice.set(v))
```

I put them in the primExtraWidgets method because this is the method accessed when the graph has been input by the user. Therefore, its functionality will occur right after a valid graph has been returned. When this code is run the following is output:



This is as expected.

Finally, in terms of the GUI set up, I created the Visualise button and invalidMessage widget:

3.2.r

```
Button(self.primFrame, text="Visualise", command=self.validate, bg="#1b263b", fg="white", font=fontMedium, width=20).grid(row=12, column=0, columnspan=3)

self.invalidMessage = Label(self.primFrame, text="invalid message", font=fontMedium, bg="#eaebed", fg="#990000")
self.invalidMessage.grid(row=12, column=3, columnspan=9)
```

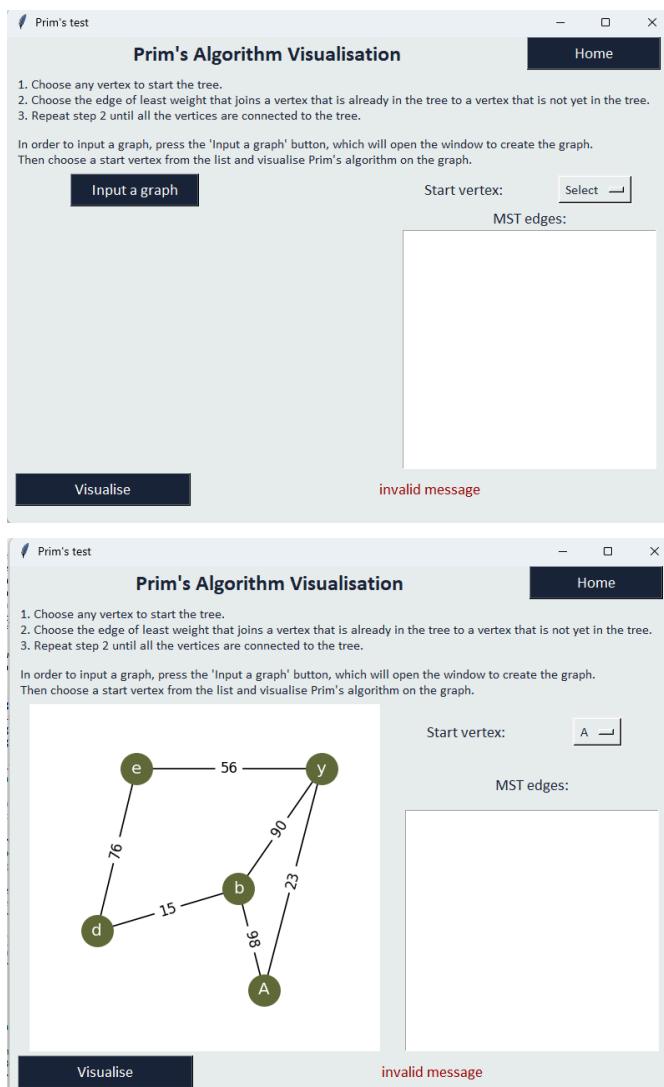
And I made sure that when the graph has been input, the size of the window returns to the original size with this code:

3.2.s

```
def primExtraWidgets(self):
    self.startVertexOptions = list(self.graph.nodes)
    self.startVertexChoice.set(self.startVertexOptions[0] if self.startVe
    self.startVertexDropdown["menu"].delete(0, "end")
    for vertex in self.startVertexOptions:
        self.startVertexDropdown["menu"].add_command(label=vertex, comman

    # destroy button so axes can go there
    self.graphInputButton.destroy()
    # change size of window
    self.master.geometry("750x550")
```

This is what is output now:



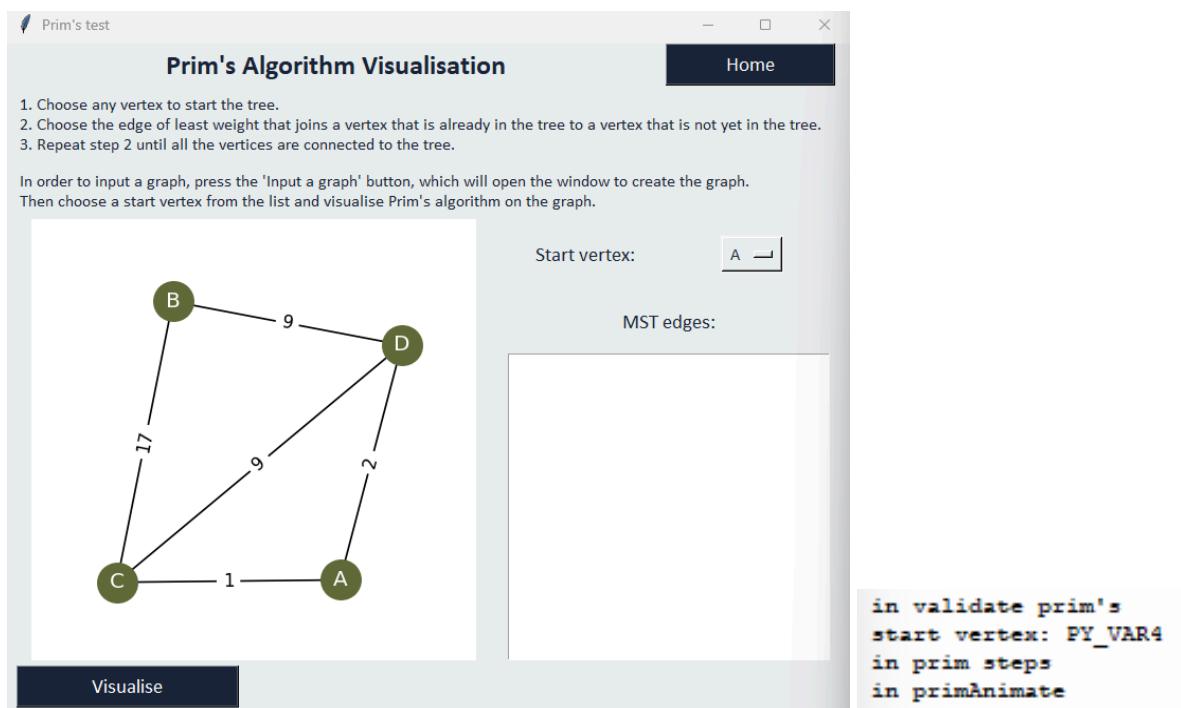
Now that the GUI has been set up, the next step is to complete the functionality of the Prim's algorithm. This requires the validate, updatePrimGraph, primSteps and primAnimate methods to be

completed in the Prim class. These need to then be added to the menu code. To begin, I completed the validate method, which checks that an edge has been selected:

3.2.t

```
def validate(self):
    print("in validate prim's")
    # note that graph has already been validated
    # checks that a start vertex has been input
    # then starts finding steps
    if self.startVertexChoice == "Select": # select is default option where there is no vertex selected
        # start vertex not chosen
        return
    self.primSteps()
    self.primAnimate()
```

This is the output on in the window and in the Shell after Visualise was pressed:



This is as expected. However, because I had enforced that the dropdown menu starts with the first value of the vertices, this check is unnecessary but I will still keep it because it enforces good practices and sets up a framework for other validation to occur in.

Now, I am going to develop the code for the other 3 sections of visualising Prim's algorithm. As I have already developed the visualisation for the Bubble Sort algorithm, I have a framework to follow in terms of how this should be implemented. See below the code that I have written:

3.2.u

```
def primSteps(self):
    # return steps in dicitionary
    print("in prim steps")

    self.steps = []
    self.mstEdges = []
    startVertex = self.startVertexChoice.get()
    visited = set() # stores mst edges in an unordered, unchangeable and unindexed format
    minHeap = [] # this is the queue for the edges connected to the current vertex

    visited.add(startVertex)

    for neighbour, weight in self.graph[startVertex].items():
        heapq.heappush(minHeap, (weight, startVertex, neighbour)) # enqueues all connected nodes and edges onto minHeap

    self.steps.append(tuple(self.mstEdges))

    while len(visited) < len(self.graph.nodes):
        if not minHeap:
            self.invalidMessage["text"] = "Graph is not connected. No MST found."
            return
        weight, u, v = heapq.heappop(minHeap) # heappop returns the smallest edge value element in minHeap
        if v not in visited:
            visited.add(v)
            self.mstEdges.append((u,v))
            self.steps.append(tuple(self.mstEdges))
            for neighbour, weight, in self.graph[v].items():
                if neighbour not in visited:
                    heapq.heappush(minHeap, (weight, v, neighbour))
```

3.2.v

```

def primAnimate(self):
    # animates at start when not paused
    print("In primAnimate")
    if not self.isPaused and self.currentStep < len(self.steps):
        mstStep = self.steps[self.currentStep] # current step#s edges
        self.updatePrimGraph(mstStep)
        # show all edges currently in the mst in the Text section
        self.visualiseText.insert(tk.END, "MST:" + " ".join([f"{u}-{v}" for u,v in step]) + "\n")
        self.visualiseText.see(tk.END)

        self.currentStep += 1
        self.master.after(750, self.primAnimate)

    elif not self.isPaused and self.currentStep == len(self.steps):
        step = self.steps[self.currentStep - 1]
        self.updatePrimGraph(step)
        self.visualiseText.insert(tk.END, "MST:" + " ".join([f"{u}-{v}" for u,v in step]) + "\n")
        self.visualiseText.see(tk.END)
        self.invalidMessage["text"] = "MST found"
    
```

Then, I extended the methods in the Menu class to include the Prim's algorithm implementation:

3.2.w

```

def skipBack(self):
    self.currentStep = 0
    self.isPaused = True
    if hasattr(self, "updateChart"): # note that methods count as attributes
        self.updateChart(self.steps[self.currentStep])
    elif hasattr(self, "updatePrimGraph"):
        self.updatePrimGraph(self.steps[self.currentStep])
    # extend once implemented other algorithms
    self.invalidMessage["text"] = ""

def skipForward(self):
    self.currentStep = len(self.steps) - 1 # gets the last step index
    self.isPaused = True
    if hasattr(self, "updateChart"): # note that methods count as attributes
        self.updateChart(self.steps[self.currentStep])
    elif hasattr(self, "updatePrimGraph"):
        self.updatePrimGraph(self.steps[self.currentStep])
    # extend once implemented other algorithms
    
```

3.2.x

```

def togglePlayPause(self):
    self.isPaused = not self.isPaused # switches the state
    self.playPauseButton.config(text="Pause" if not self.isPaused else "Play") # switches the text
    if not self.isPaused: # ie displays Pause but the visualisation is playing
        self.playAnimation()
    if hasattr(self, "bubbleSortAnimate"):
        self.bubbleSortAnimate()
    elif hasattr(self, "primAnimate"):
        self.primAnimate()
    
```

3.2.y

```

def stepBack(self):
    if self.isPaused and self.currentStep > 0:
        self.currentStep -= 1
    if hasattr(self, "updateChart"): # note that methods count as attributes
        numbers = self.steps[self.currentStep]
        indices = self.stepsDictionary[numbers]
        self.updateChart(numbers, indices)
    elif hasattr(self, "updatePrimGraph"):
        self.updatePrimGraph(self.steps[self.currentStep])
    # extend once implemented other algorithms

def stepForward(self):
    if self.isPaused and self.currentStep < len(self.steps)-1:
        self.currentStep += 1
    if hasattr(self, "updateChart"): # note that method count as attributes
        numbers = self.steps[self.currentStep]
        indices = self.stepsDictionary[numbers]
        self.updateChart(numbers, indices)
    elif hasattr(self, "updatePrimGraph"):
        self.updatePrimGraph(self.steps[self.currentStep])
    
```

Finally, I realised that going between the pages would mean that the Menu widgets would still be there on the Graph Input page. Therefore, I created a method to hide the Menu widgets.

3.2.z

```
def hideMenuWidgets(self):
    for widget in self.menuFrame.winfo_children():
        widget.grid_forget()
    self.title.grid_forget()
```

I also added lines to show and hide the widgets throughout the Prim class:

3.2.A

```
def graphInputWindow(self):
    # open graph input window (but don't close current window)
    print("in graph input window method")
    # hide widgets on prim page
    self.hidePrimWidgets()
    # hide menu widgets
    self.hideMenuWidgets()
    # create graph page to open
    self.graphInputPageWidgets()

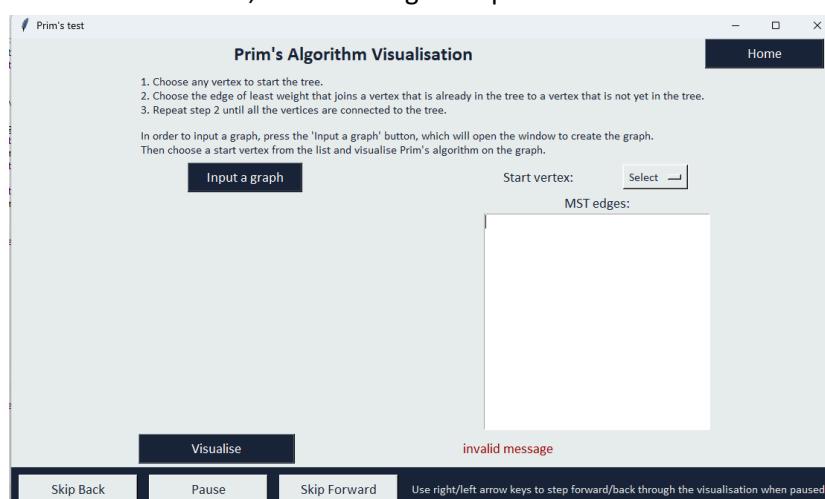
def primExtraWidgets(self):
    # show menu widgets
    self.menuWidgets()
```

3.2.B

```
class Prim(GraphInput, Menu): #HomeMain
    def __init__(self, master):
        #super().__init__(master, pUsername)
        self.master = master
        print("in prim __init__ 1")
        GraphInput.__init__(self, master)
        print("in prim __init__ after GraphInput")
        Menu.__init__(self, master, 13, 12)
        print("in prim __init__ after Menu")

        self.primWidgets()
        self.menuWidgets()
```

When I run the code, the following is output:



The screenshot shows a window titled "Input a Graph". It has two main sections: "Canvas" on the left and "Graph:" on the right. The "Canvas" section is a large empty area for drawing vertices. The "Graph:" section is a coordinate system with both x and y axes ranging from 0.0 to 1.0 in increments of 0.2. Below the graph area is a message box containing the text "invalid message". At the bottom right of the window is a "Submit" button.

The screenshot shows a window titled "Prim's Algorithm Visualisation". It displays a graph with 6 vertices labeled A through F. Vertex A is the starting vertex, indicated by a green border. Edges and their weights are: A-B (6), A-C (7), A-D (2), B-C (4), B-F (5), C-F (7), and C-E (9). To the right of the graph, there is a "Start vertex:" dropdown menu set to "A" and a "MST edges:" list which currently contains "invalid message". Below the graph are "Visualise" and "invalid message" buttons. At the bottom are "Skip Back", "Pause", "Skip Forward", and a note: "Use right/left arrow keys to step forward/back through the visualisation when paused."

```

in validate prim's
start vertex: PY_VAR4
in prim steps
Exception in Tkinter callback
Traceback (most recent call last):
  File "C:\Users\ingri\AppData\Local\Programs\Thonny\lib\tkinter\_init__.py", line 1921, in __call__
    return self.func(*args)
  File "C:\Users\ingri\OneDrive\Documents\QM\QM School work\A Level\Computer Science\NEA\prim_page.py", line 141, in validate
    self.primSteps()
  File "C:\Users\ingri\OneDrive\Documents\QM\QM School work\A Level\Computer Science\NEA\prim_page.py", line 175, in primSteps
    heapq.heappush(minHeap, (weight, startVertex, neighbour)) # enqueues all connected nodes and edges onto minHeap
TypeError: '<' not supported between instances of 'dict' and 'dict'
    
```

This error occurs because the items for `self.graph[v]` returns a dictionary of data on the edges connected to v, which can't be used when doing `heappush`. Therefore, I added a line of code to get the value of weight from the edge data:

3.2.C

```

for neighbour, edgeData in self.graph[startVertex].items():
    weight = edgeData["weight"]
    heapq.heappush(minHeap, (weight, startVertex, neighbour)) # enqueues

    self.steps.append(tuple(self.mstEdges))
    
```

As this code is also repeated later in the method, I changed the code to the following here as well:

3.2.D

```

while len(visited) < len(self.graph.nodes):
    if not minHeap:
        self.invalidMessage["text"] = "Graph is not connected. No MST found."
        return
    weight, u, v = heapq.heappop(minHeap) # heappop returns the smallest edge
    if v not in visited:
        visited.add(v)
        self.mstEdges.append((u,v))
        self.steps.append(tuple(self.mstEdges))
        for neighbour, edgeData in self.graph[v].items():
            weight = edgeData["weight"]
            if neighbour not in visited:
                heapq.heappush(minHeap, (weight, v, neighbour))

```

Now when I run the code and press Visualise, I get the following error:

```

in prim steps
in primAnimate
in update graph
Exception in Tkinter callback
Traceback (most recent call last):
  File "C:\Users\ingri\AppData\Local\Programs\Thonny\lib\tkinter\__init__.py", line 1921, in __call__
    return self._func(*args)
  File "C:\Users\ingri\OneDrive\Documents\QM\QM School work\A Level\Computer Science\NEA\prim_page.py", line 142,
    self.primAnimate()
  File "C:\Users\ingri\OneDrive\Documents\QM\QM School work\A Level\Computer Science\NEA\prim_page.py", line 203,
ate
    self.visualiseText.insert(tk.END, "MST:" + ".join([f"\{u}\{v}\," for u,v in step]) + "\n")
UnboundLocalError: local variable 'step' referenced before assignment

```

I think that there are two possible causes for this error. One of these is if self.steps doesn't have any values in it. To test this, I added the following code at the beginning of primAnimate:

3.2.E

```

# test steps:
if not self.steps:
    print("Error: no steps recorded in self.steps")
    return

```

The other issue may be that I have defined mstStep as the values of the current step but have written step in the text lines. Therefore, I changed the code to the following:

3.2.F

```
self.visualiseText.insert(tk.END, "MST:" + ".join([f"\{u}\{v}\," for u,v in mstStep]) + "\n")
```

When I now run the code, the following is output:

```

in prim steps
in primAnimate
in update graph
in primAnimate
in update graph
Exception in Tkinter callback
Traceback (most recent call last):
  File "C:\Users\ingri\AppData\Local\Programs\Thonny\lib\tkinter\__init__.py", line 1921, in __call__
    return self._func(*args)
  File "C:\Users\ingri\AppData\Local\Programs\Thonny\lib\tkinter\__init__.py", line 839, in callit
    func(*args)
  File "C:\Users\ingri\OneDrive\Documents\QM\QM School work\A Level\Computer Science\NEA\prim_page.py", line 207, in primAnimate
    self.updatePrimGraph(mstStep)
  File "C:\Users\ingri\OneDrive\Documents\QM\QM School work\A Level\Computer Science\NEA\prim_page.py", line 167, in updatePrimGraph
    nx.draw_networkx_edges(self.graph, pos, edgelist=[(u,v)], edge_color = colour, width=2, ax=self.ax)
NameError: name 'edgelist' is not defined

```

This may have been due to a syntax error in the draw_networkx_edges call. I fixed this by changing it to this:

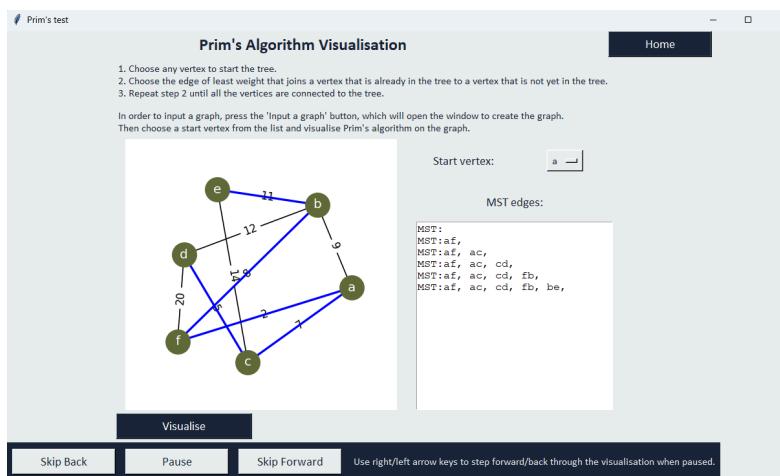
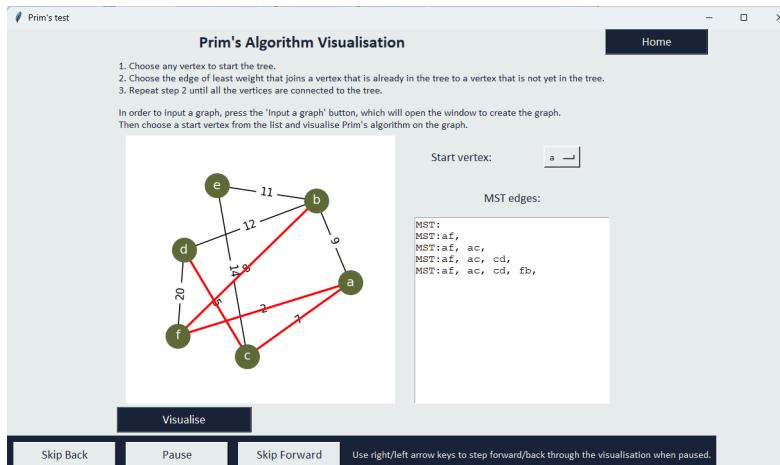
3.2.F

```

for u,v in step:
    colour = "blue" if step == self.steps[-1] else "red"
    nx.draw_networkx_edges(self.graph, pos, edgelist=[(u,v)], edge_color = colour, width=2, ax=self.ax)

```

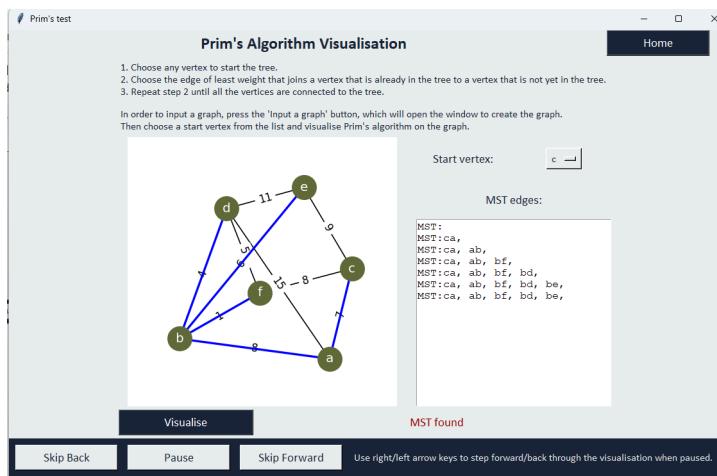
When I now run the code, the following is output:



```

Exception in Tkinter callback
Traceback (most recent call last):
  File "C:\Users\ingri\AppData\Local\Programs\Thonny\lib\tkinter\__init__.py", line 1921, in __call__
    return self.func(*args)
  File "C:\Users\ingri\AppData\Local\Programs\Thonny\lib\tkinter\__init__.py", line 839, in callit
    func(*args)
  File "C:\Users\ingri\OneDrive\Documents\QM\QM School work\A Level\Computer Science\NEA\prim page.py", line
218, in primAnimate
    self.visualiseText.insert(tk.END, "MST:" + " ".join([f"{u}{v}," for u,v in mstStep]) + "\n")
UnboundLocalError: local variable 'mstStep' referenced before assignment
    
```

This error was again due to a typo which was easily fixed. When I now run the visualisation:

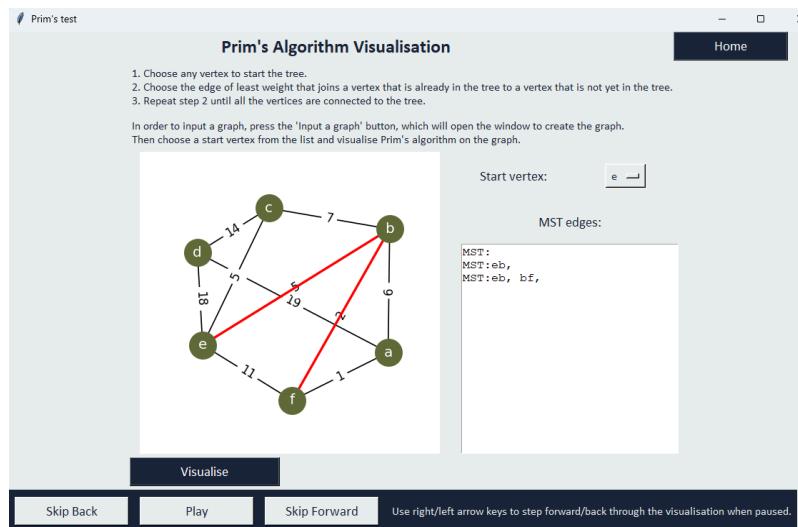


The functionality of the basic visualisation and animation seems to be working well but I realised from the final output that the MST edges are displaying again in the Text box. This is unnecessary so I removed the lines that inserted the text:

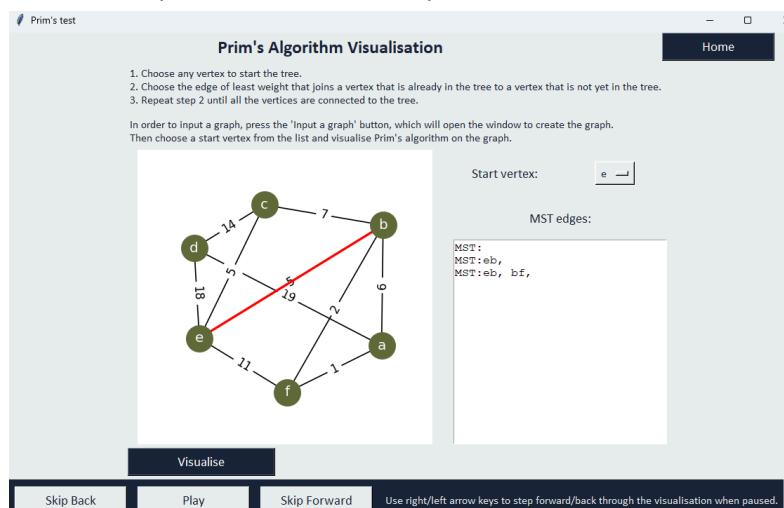
3.2.G

```
elif not self.isPaused and self.currentStep == len(self.steps):
    mstStep = self.steps[self.currentStep - 1]
    self.updatePrimGraph(mstStep)
    self.invalidMessage["text"] = "MST found"
```

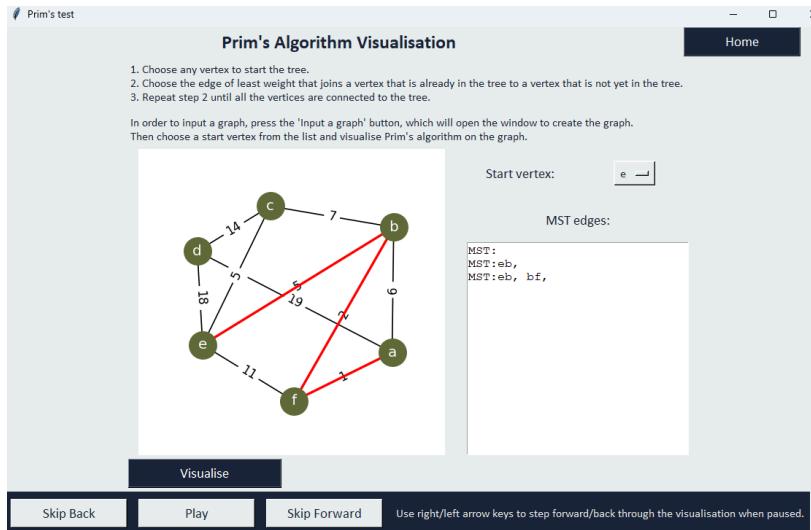
Now, I will test the Pause/Play, Skip Back/Forward and arrow key functionality as I have already extended their methods for the Prim's visualisation. First, when I pause the visualisation, the following is output:



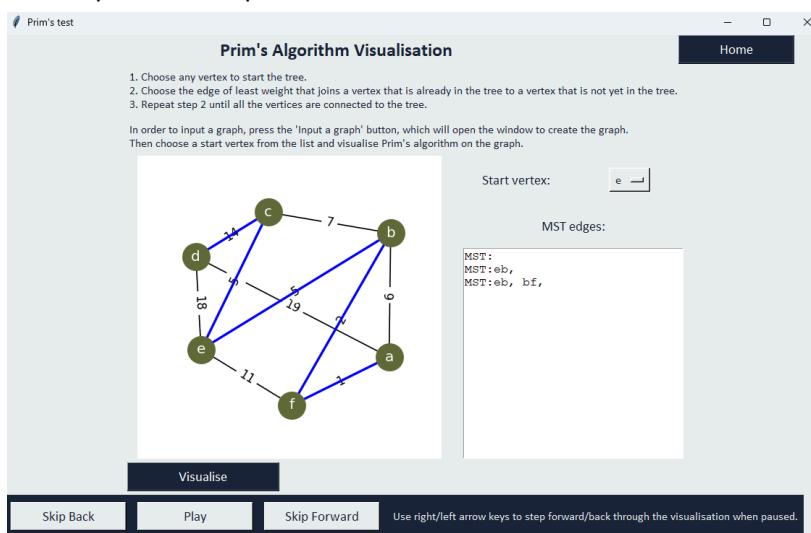
Then when I press the left arrow key:



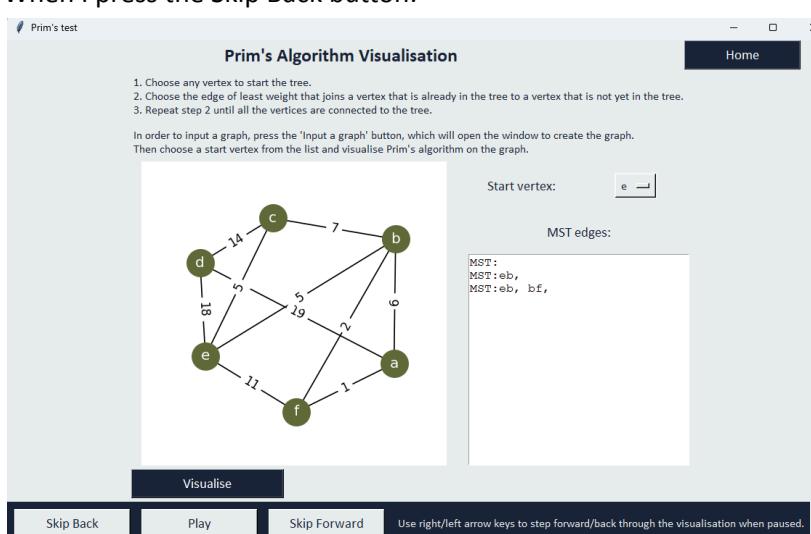
When I press the right arrow key:



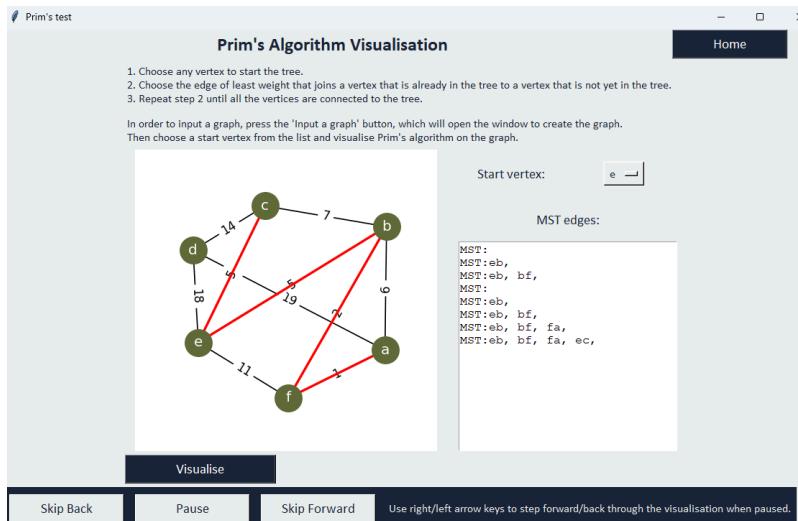
When I press the Skip Forward button:



When I press the Skip Back button:



When I press Play:



All this functionality seems to be working correctly except when I press the arrow keys, the Text for the MST edges doesn't seem to be updating. To fix this, I added the following code to the prim's portions of the menu code so that the mst edges can be shown in the Text box:

3.2.H

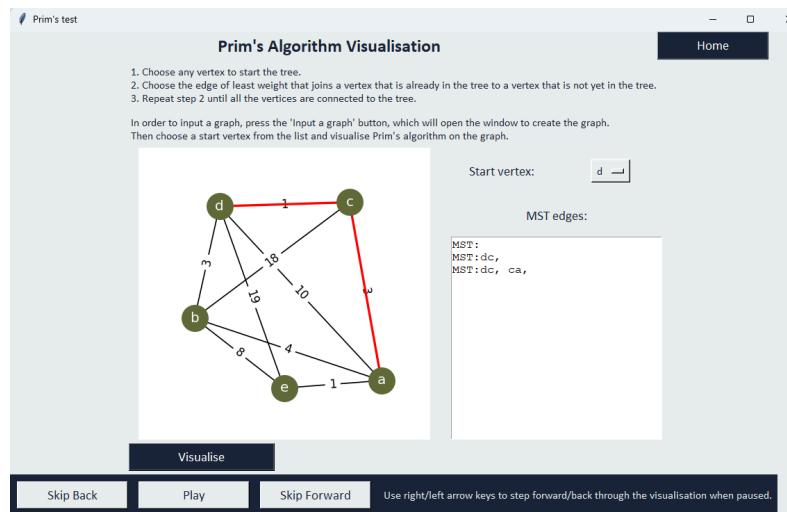
```

def stepBack(self):
    if self.isPaused and self.currentStep > 0:
        self.currentStep -= 1
        if hasattr(self, "updateChart"): # note that methods count as attributes
            numbers = self.steps[self.currentStep]
            indices = self.stepsDictionary[numbers]
            self.updateChart(numbers, indices)
    elif hasattr(self, "updatePrimGraph"):
        mstStep = self.steps[self.currentStep]
        self.updatePrimGraph(mstStep)
        self.visualiseText.insert(tk.END, "MST:" + ".join([f'{u}{v}', for u,v in mstStep]) + "\n")
        self.visualiseText.see(tk.END)
    # extend once implemented other algorithms

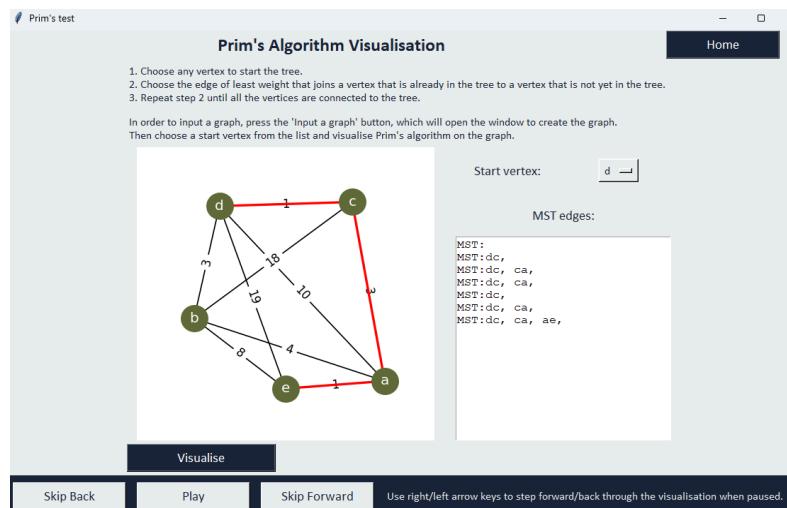
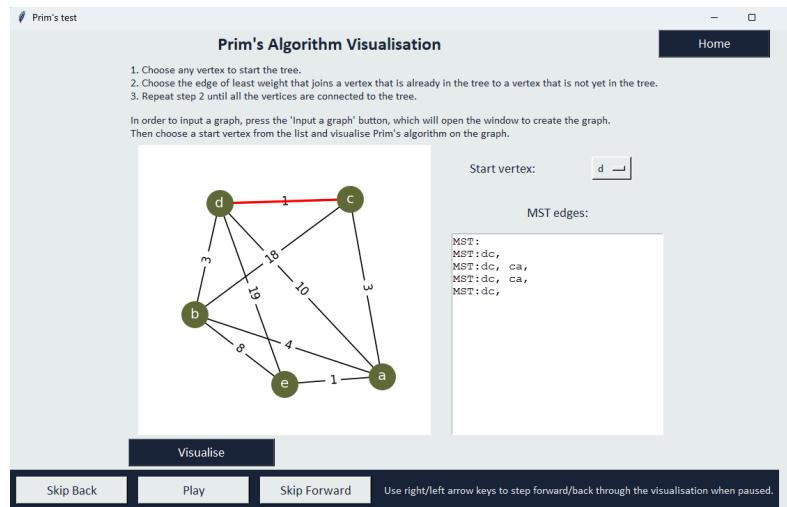
def stepForward(self):
    if self.isPaused and self.currentStep < len(self.steps)-1:
        self.currentStep += 1
        if hasattr(self, "updateChart"): # note that method count as attributes
            numbers = self.steps[self.currentStep]
            indices = self.stepsDictionary[numbers]
            self.updateChart(numbers, indices)
    elif hasattr(self, "updatePrimGraph"):
        mstStep = self.steps[self.currentStep]
        self.updatePrimGraph(mstStep)
        self.visualiseText.insert(tk.END, "MST:" + ".join([f'{u}{v}', for u,v in mstStep]) + "\n")
        self.visualiseText.see(tk.END)

```

When I now run the code and pause it the following is output:



And when I press the arrow keys, the following are output:



As can be seen, the MST edges Text box updates correctly when the arrow keys are pressed. Therefore, all functionality for the Prim's section is complete.

Now, I will focus on refining the Prim's page GUI. Firstly, I made sure that the menu is in line with the rest of the widgets of the page by adding a \n character in the text:

3.2.I

```
self.menuText = Label(self.menuFrame, text="Use right/left arrow keys to step forward/back through the visualisation when paused.", font="bold", fg="black", bg="white", width=40, height=1)  
self.menuText.grid(row=1, column=7, columnspan=12, padx=5)
```

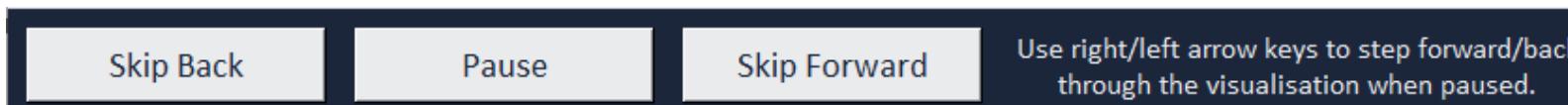
3.2.J

```
class Prim(GraphInput, Menu): #HomeMain  
    def __init__(self, master):  
        super().__init__(master, pUsername)  
        self.master = master  
        print("in prim __init__ 1")  
        GraphInput.__init__(self, master)  
        print("in prim __init__ after GraphInput")  
        Menu.__init__(self, master, 13, 12)  
        print("in prim __init__ after Menu")  
  
        self.primWidgets()  
        self.menuWidgets()  
        self.menuText["text"] = "Use right/left arrow keys to step forward/back\nthrough the visualisation when paused." # pos of \n so that it fits in the frame nicely
```

3.2.K

```
def primExtraWidgets(self):  
    # show menu widgets  
    self.menuWidgets()  
    self.menuText["text"] = "Use right/left arrow keys to step forward/back\nthrough the visualisation when paused."
```

When I run the code, the following is output:

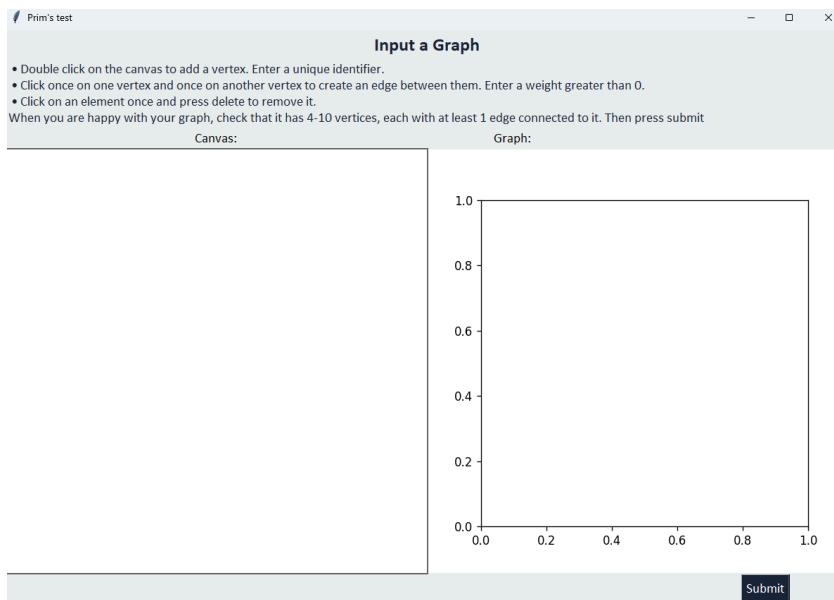


The next refinement I made was removing the dark blue background of the menu in the Graph Input page. I did this by configuring the menuFrame:

3.2.L

```
def hideMenuWidgets(self):
    for widget in self.menuFrame.winfo_children():
        widget.grid_forget()
    self.title.grid_forget()
    self.menuFrame.configure(bg="#eaebed")
```

When I run the code, the following is output:

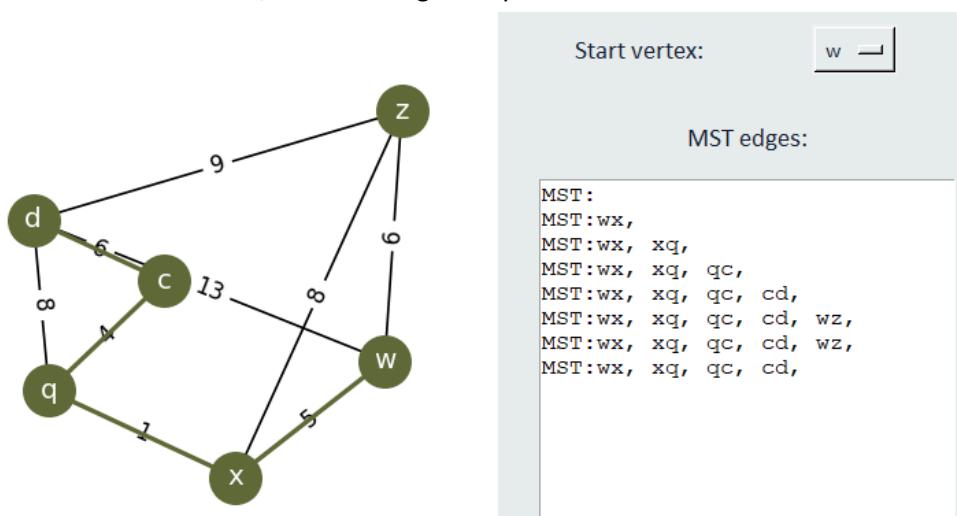


The next refinement I made was changing the colours of the edges in the graph:

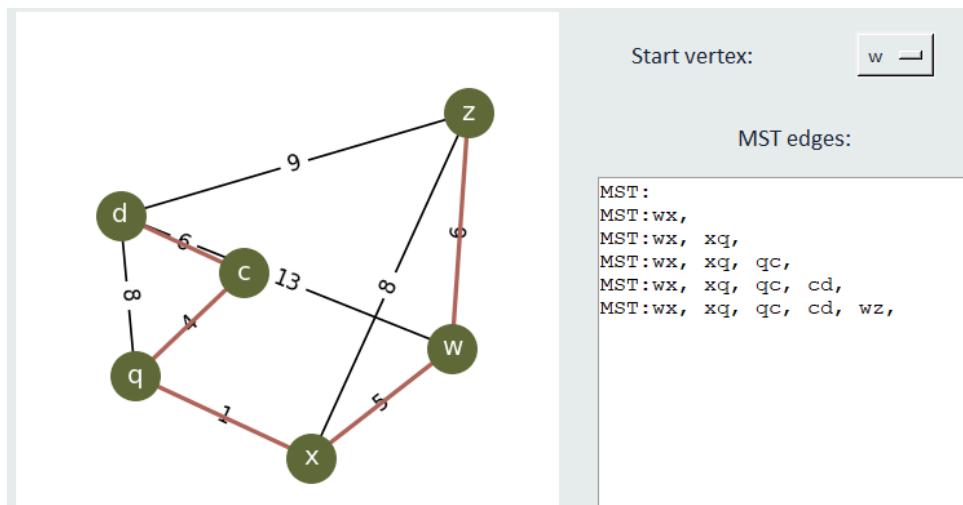
3.2.M

```
for u,v in step:
    colour = "#b2675e" if step == self.steps[-1] else "#606c38"
    nx.draw_networkx_edges(self.graph, pos, edgelist=[(u,v)], edge_color = colour, width=2, ax=self.ax)
```

When I run the visualisation, the following is output when the visualisation is still running:



And when the visualisation has finished:



Finally, I adjusted the size of the window for both when a graph has not yet been input and when a graph has been input:

3.2.N

```

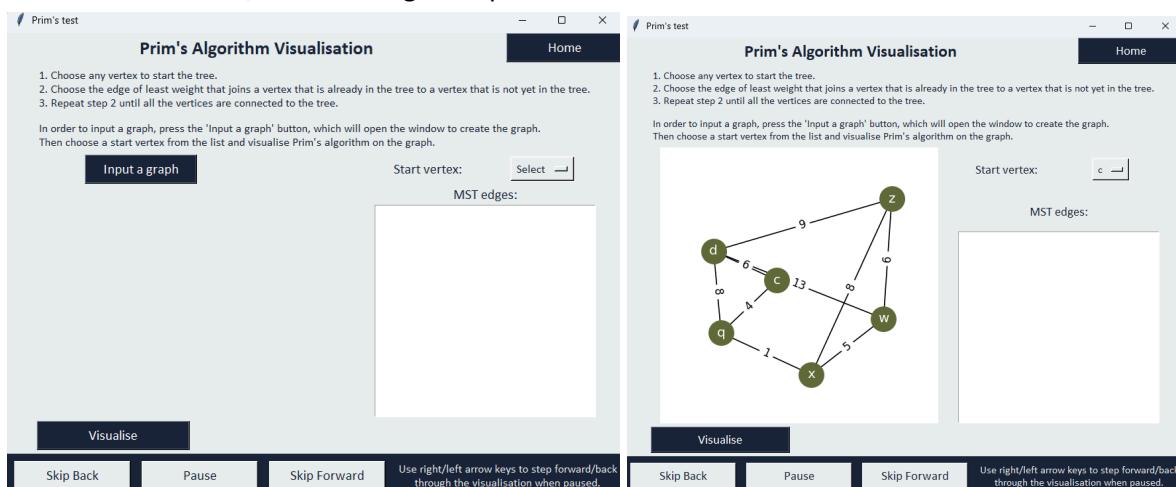
def primExtraWidgets(self):
    # show menu widgets
    self.menuWidgets()
    self.menuText["text"] = "Use right,
    # vertex dropdown menu
    self.startVertexOptions = list(self.vertices)
    self.startVertexChoice.set(self.startVertexOptions[0])
    self.startVertexDropdown["menu"].delete(0, "end")
    for vertex in self.startVertexOptions:
        self.startVertexDropdown["menu"].add_command(label=vertex, command=lambda v=vertex: self.startVertexChoice.set(v))
    # destroy button so axes can go through
    self.graphInputButton.destroy()
    # change size of window
    self.master.geometry("845x697")
    
```

3.2.O

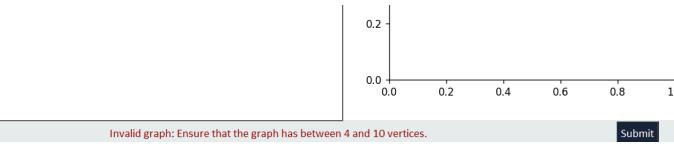
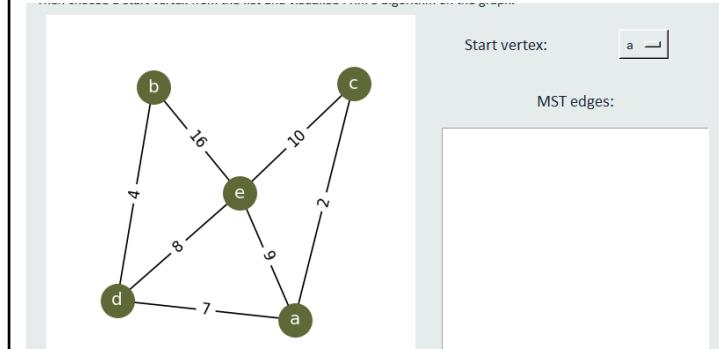
```

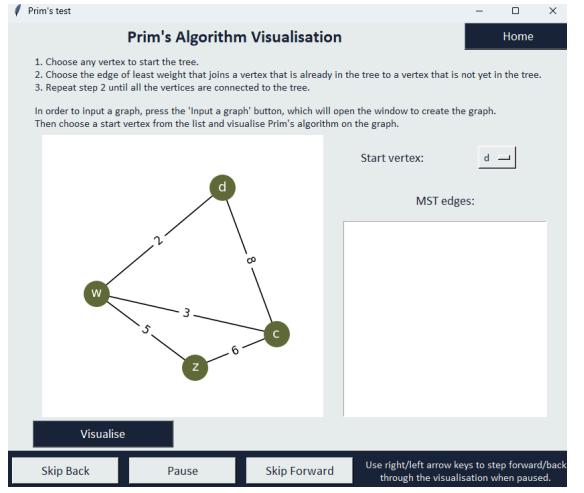
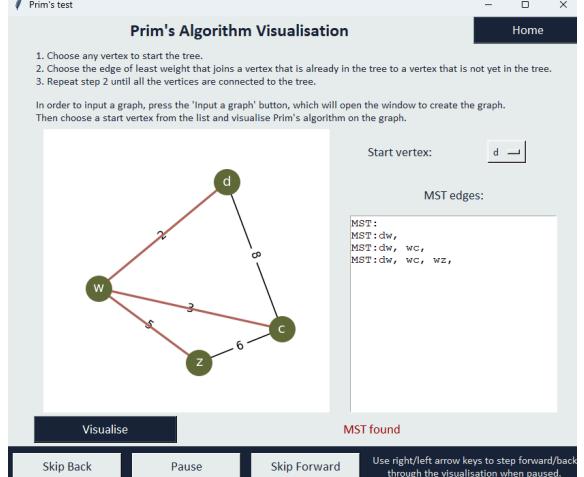
def primWidgets(self):
    self.master.configure(bg="#eaebed")
    self.master.geometry("845x640")
    
```

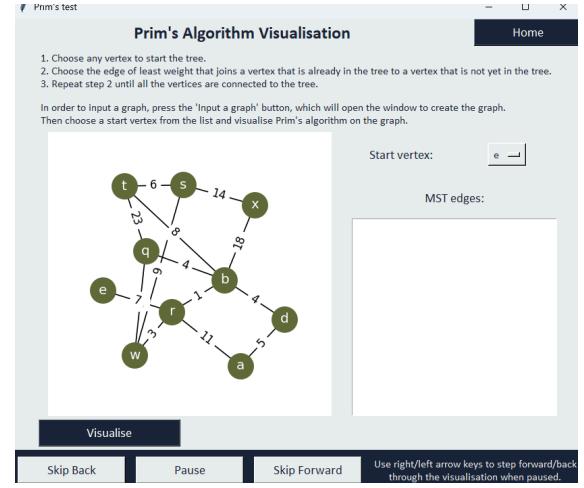
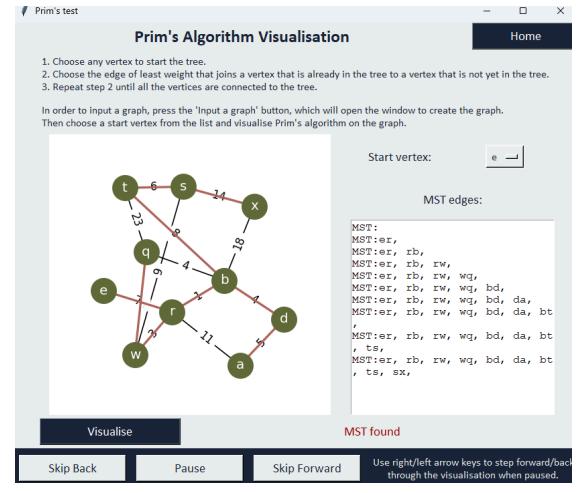
When I run the code, the following is output:



Now, I will test the code developed using the white box testing table specified in the PRIM'S VISUALISATION PAGE design section and the white box testing table specified in the MENU section that is relating to the Prim's visualisation.

TEST DATA	EXPECTED RESULT	JUSTIFICATION	CODE	ACTUAL OUTPUT
Press visualise without inputting a graph.	An 'Invalid graph' error message is displayed.	Confirms that a graph must be entered by the user for the visualisation to begin.	3.1.p	<p>As the functionality of the graph input has been changed to be in a new page, change this to press Submit without inputting a graph.</p>  <p>As expected</p>
Press visualise without entering a start vertex.	An 'Invalid - please enter a start vertex' error message is displayed.	Confirms that the start vertex must be entered for the algorithm to begin visualising.	3.2.b 3.2.t 3.2.A 3.2.N	<p>As the functionality of the select start vertex has been changed, this is no longer something to test:</p>  <p>As seen, the start vertex selection is always one of the vertices entered.</p>

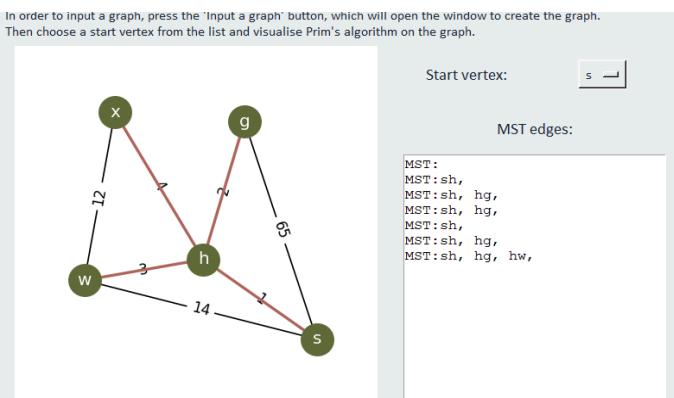
Visualise with a connected graph of 4 vertices and a start vertex chosen.	The visualisation of Prim's algorithm starts correctly to find the MST of the entered graph.	This is a boundary test on the minimum number of vertices that could be entered for the algorithm to visualise.	3.2.b 3.2.u 3.2.A 3.2.D 3.2.M 3.2.N	 <p>Prim's test</p> <p>Prim's Algorithm Visualisation</p> <ol style="list-style-type: none"> Choose any vertex to start the tree. Choose the edge of least weight that joins a vertex that is already in the tree to a vertex that is not yet in the tree. Repeat step 2 until all the vertices are connected to the tree. <p>In order to input a graph, press the 'Input a graph' button, which will open the window to create the graph. Then choose a start vertex from the list and visualise Prim's algorithm on the graph.</p> <p>Start vertex: <input type="text" value="d"/></p> <p>MST edges:</p> <p>Visualise Skip Back Pause Skip Forward Use right/left arrow keys to step forward/back through the visualisation when paused.</p>  <p>Prim's test</p> <p>Prim's Algorithm Visualisation</p> <ol style="list-style-type: none"> Choose any vertex to start the tree. Choose the edge of least weight that joins a vertex that is already in the tree to a vertex that is not yet in the tree. Repeat step 2 until all the vertices are connected to the tree. <p>In order to input a graph, press the 'Input a graph' button, which will open the window to create the graph. Then choose a start vertex from the list and visualise Prim's algorithm on the graph.</p> <p>Start vertex: <input type="text" value="d"/></p> <p>MST edges:</p> <p>MST: MST:dw, MST:dw, wc, MST:dw, wc, wz,</p> <p>MST found</p> <p>Visualise Skip Back Pause Skip Forward Use right/left arrow keys to step forward/back through the visualisation when paused.</p> <p>As expected.</p>
---	--	---	--	--

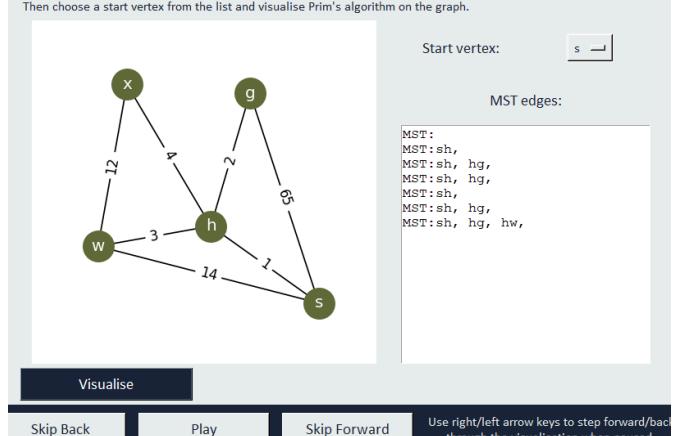
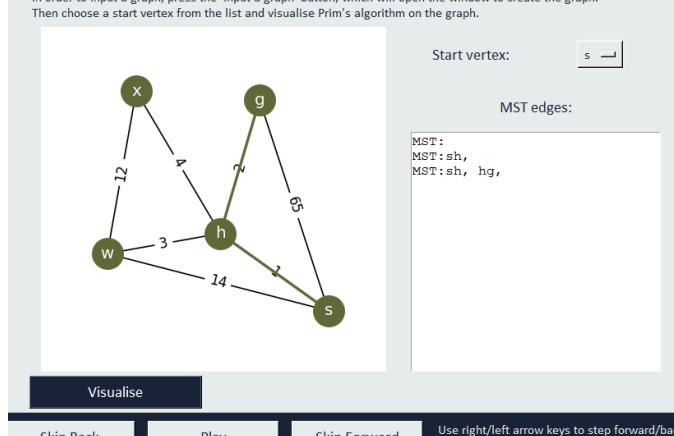
Visualise with a connected graph of 10 vertices and a start vertex chosen.	The visualisation of Prim's algorithm starts correctly to find the MST of the entered graph.	This is a boundary test on the maximum number of vertices that could be entered for the algorithm to visualise.	3.2.b 3.2.u 3.2.A 3.2.D 3.2.M 3.2.N	 <p>Prim's Algorithm Visualisation</p> <p>Start vertex: e</p> <p>MST edges:</p> <p>Visualise</p> <p>Skip Back Pause Skip Forward Use right/left arrow keys to step forward/back through the visualisation when paused.</p>  <p>Prim's Algorithm Visualisation</p> <p>Start vertex: e</p> <p>MST edges:</p> <pre> MST: MST:er, MST:er, rb, MST:er, rb, rw, MST:er, rb, rw, wq, MST:er, rb, rw, wq, bd, MST:er, rb, rw, wq, bd, da, MST:er, rb, rw, wq, bd, da, bt , MST:er, rb, rw, wq, bd, da, bt , ts, MST:er, rb, rw, wq, bd, da, bt , ts, sx, </pre> <p>MST found</p> <p>Visualise</p> <p>Skip Back Pause Skip Forward Use right/left arrow keys to step forward/back through the visualisation when paused.</p> <p>As expected.</p>
--	--	---	--	---

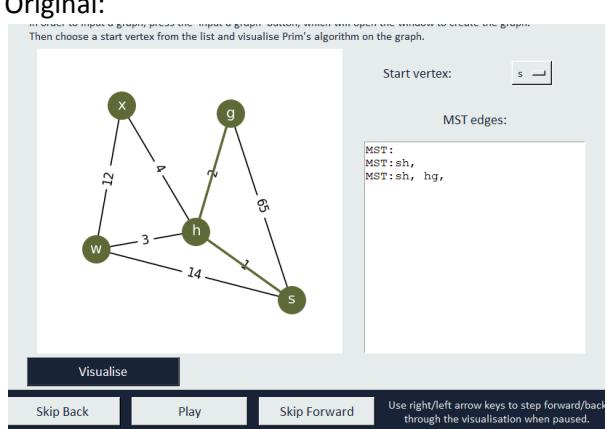
Visualise with a connected graph of 3 vertices and a start vertex chosen.	An ‘Invalid graph’ error message is displayed.	This is an erroneous test for if the number of vertices entered is less than the minimum.	3.1.p 3.1.l	<p>As the functionality of the graph input has been changed to be in a new page, change this to press Submit with a 3 vertex graph.</p> <div style="text-align: center; margin-top: 10px;"> <small>Invalid graph: Ensure that the graph has between 4 and 10 vertices.</small> Submit </div> <p>As expected.</p>
Visualise with a connected graph of 11 vertices and a start vertex chosen.	An ‘Invalid graph’ error message is displayed.	This is an erroneous test for if the number of vertices entered is more than the maximum.	3.1.c 3.1.l	<p>As the functionality of the graph input has been changed to be in a new page, change this to press Submit with an 11 vertex graph.</p> <p>On the input for the 11th vertex, when submit is pressed, the vertex does not appear on the graph. This is as expected.</p>

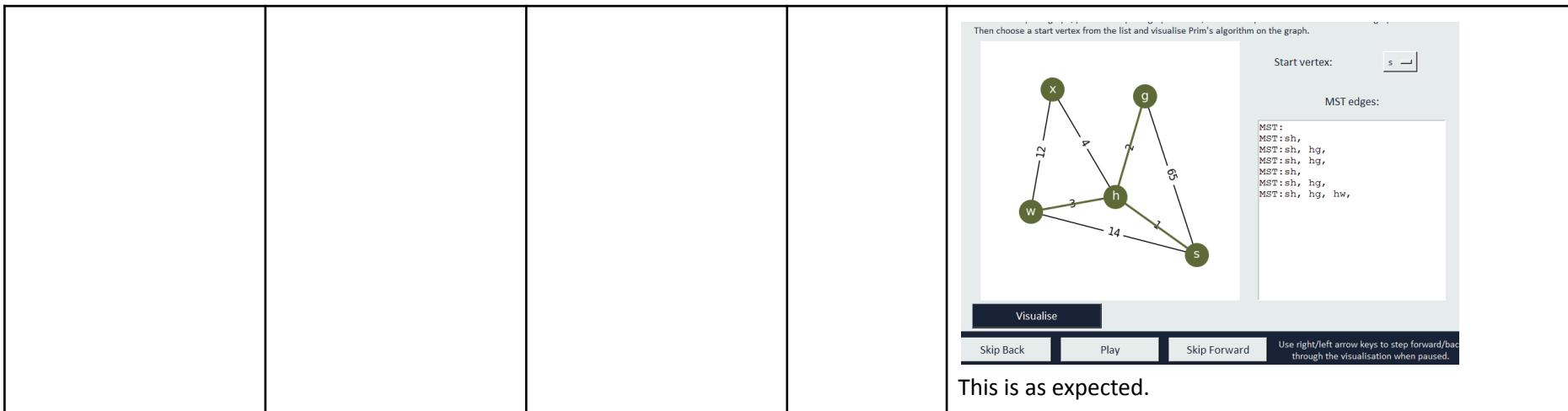
Visualise with an unconnected graph of 6 vertices and a start vertex chosen.	An 'Invalid graph' error message is displayed.	This is a test to ensure that all the vertices must have at least 1 edge connected to them, therefore making the visualisation worthwhile.	3.2.b 3.2.u 3.2.D 3.2.M 3.2.N	<p>As the functionality of the graph input has been changed to be in a new page, change this to press Submit with an unconnected graph.</p> <p>For a graph where vertices have no edges connecting them:</p> <p>This is as expected. For a graph that is not fully connected:</p> <p>Prim's Algorithm Visualisation</p> <p>In order to input a graph, press the 'Input a graph' button, which will open the window to create the graph. Then choose a start vertex from the list and visualise Prim's algorithm on the graph.</p> <p>Start vertex: <input type="text" value="c"/></p> <p>MST edges:</p> <p>When visualise is pressed for this:</p>
--	--	--	---	---

				<p>L. Choose any vertex to start the tree. 2. Choose the edge of least weight that joins a vertex that is already in the tree to a vertex that is not yet in the tree. 3. Repeat step 2 until all the vertices are connected to the tree.</p> <p>In order to input a graph, press the 'Input a graph' button, which will open the window to create the graph. Then choose a start vertex from the list and visualise Prim's algorithm on the graph.</p> <p>Start vertex: <input type="text" value="c"/></p> <p>MST edges: MST: MST: cd,</p> <p>Visualise Graph is not connected. No MST found.</p> <p>Skip Back Play Skip Forward Use right/left arrow keys to step forward/s Home</p> <p>Prim's Algorithm Visualisation</p> <p>1. Choose any vertex to start the tree. 2. Choose the edge of least weight that joins a vertex that is already in the tree to a vertex that is not yet in the tree. 3. Repeat step 2 until all the vertices are connected to the tree.</p> <p>In order to input a graph, press the 'Input a graph' button, which will open the window to create the graph. Then choose a start vertex from the list and visualise Prim's algorithm on the graph.</p> <p>Start vertex: <input type="text" value="c"/></p> <p>MST edges: MST: MST: cd, MST: cd, ce, MST: ce, ce, es,</p> <p>Visualise MST found</p> <p>Skip Back Pause Skip Forward Use right/left arrow keys to step forward/s Home</p> <p>Prim's Algorithm Visualisation</p> <p>Execution of Prim's algorithm should have stopped when the 'Graph not connected' message is displayed, however, it keeps going, which is erroneous.</p>
Press the Home button	The Prim's Visualisation	This ensures that the	3.2.c	After starting from the login page and clicking the Prim

	page is closed and the Home page is opened.	'Home' button works correctly and that the user can go back to the Home page to either log out or go to the other visualisation/quiz pages.		Algorithm button: <pre>Exception in Tkinter callback Traceback (most recent call last): File "C:\Users\lingri\AppData\Local\Programs\Thonny\lib\tkinter_.py", line 1921, in _call return self.func(*args) File "C:\Users\lingri\OneDrive\Documents\QM\QM School work\A Level\Computer Science\NEA\home_page.py", line 192, in openPrim running = Prim(prinRoot, username) TypeError: Prim.__init__() takes 2 positional arguments but 3 were given</pre> <p>This is an error.</p>
Press the 'Skip Forward' button for the Prim's visualisation	The text-based visualisation displays the final edges in the MST and the graph has the MST edges highlighted in #606C38	Confirms that the 'Skip Forward' button works as intended for the Prim's visualisation.	3.2.b 3.2.u 3.2.D 3.2.w 2.3.H 2.3.I	<p>In order to input a graph, press the 'Input a graph' button, which will open the window to create the graph. Then choose a start vertex from the list and visualise Prim's algorithm on the graph.</p> <p>Start vertex: <input type="text" value="s"/></p> <p>MST edges:</p> <pre>MST: MST:sh, MST:sh, hg, MST:sh, hg, MST:sh, hg, MST:sh, hg, hw,</pre>  <p>Visualise</p> <p>Skip Back Play Skip Forward Use right/left arrow keys to step forward/back through the visualisation when paused.</p> <p>As expected.</p>

Press the 'Skip Back' button for the Prim's visualisation	The graph remains the same but the highlighting on the graph is removed.	Confirms that the 'Skip Back' button works as intended for the Prim's visualisation.	3.2.b 3.2.u 3.2.D 3.2.w 2.3.H 2.3.I	 <p>Then choose a start vertex from the list and visualise Prim's algorithm on the graph.</p> <p>Start vertex: <input type="text" value="s"/></p> <p>MST edges:</p> <pre>MST: MST:sh, MST:sh, hg, MST:sh, hg, MST:sh, MST:sh, hg, MST:sh, hg, hw,</pre> <p>Visualise</p> <p>Skip Back Play Skip Forward</p> <p>Use right/left arrow keys to step forward/back through the visualisation when paused.</p> <p>As expected.</p>
Press the 'Pause' button for the Prim's visualisation	The visualisation stops, the button changes from 'Pause' to 'Play' (or vice versa).	Confirms that the 'Pause' button works as intended for the Prim's visualisation.	3.2.b 3.2.u 3.2.D 3.2.v 3.2.x 2.3.H 2.3.I	 <p>Then choose a start vertex from the list and visualise Prim's algorithm on the graph.</p> <p>Start vertex: <input type="text" value="s"/></p> <p>MST edges:</p> <pre>MST: MST:sh, MST:sh, hg,</pre> <p>Visualise</p> <p>Skip Back Play Skip Forward</p> <p>Use right/left arrow keys to step forward/back through the visualisation when paused.</p> <p>As expected.</p>

Use the right and left arrow keys for the Prim's visualisation	The visualisation goes back/forward one step in the visualisation on both the graph and text.	Confirms that the arrow key functionality works as intended for the Prim's visualisation.	3.2.b 3.2.u 3.2.D 3.2.H 2.3.H 2.3.I	 <p>Original:</p> <p>Then choose a start vertex from the list and visualise Prim's algorithm on the graph.</p> <p>Start vertex: <input type="text" value="s →"/></p> <p>MST edges: MST: MST:sh, MST:sh, hg,</p> <p>Visualise Skip Back Play Skip Forward Use right/left arrow keys to step forward/back through the visualisation when paused.</p> <p>Left arrow key pressed:</p> <p>In order to input a graph, press the Input a graph button, which will open the window to create the graph. Then choose a start vertex from the list and visualise Prim's algorithm on the graph.</p> <p>Start vertex: <input type="text" value="s →"/></p> <p>MST edges: MST: MST:sh, MST:sh, hg, MST:sh, hg, MST:sh,</p> <p>Visualise Skip Back Play Skip Forward Use right/left arrow keys to step forward/back through the visualisation when paused.</p> <p>Right arrow key pressed:</p>
--	---	---	--	--



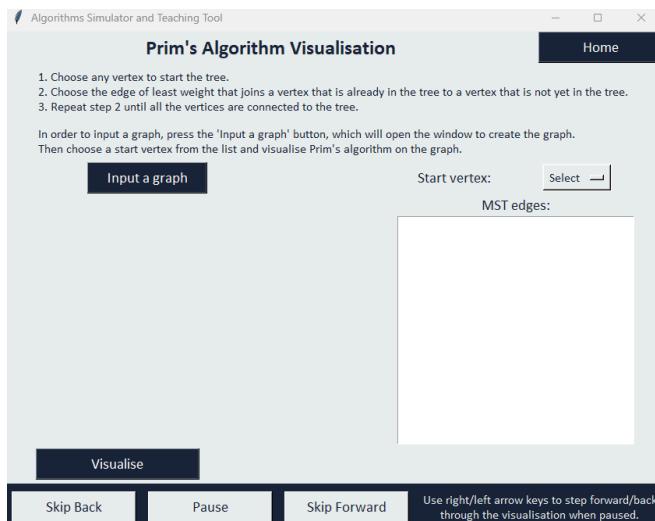
This is as expected.

To fix the issue with the connection between the Home page and the Prim page, I removed the username parameter from the class instantiation in closeOpen in NEA_utilities.py:

3.2.P

```
elif newType == "prim":
    try:
        from prim_page import Prim
        primRoot = Tk()
        primRoot.geometry('500x350') # to change
        primRoot.title("Algorithms Simulator and Teaching Tool")
        running = Prim(primRoot)
        primRoot.mainloop()
    except ImportError as e:
        print(f"Failed import Prim: {e}")
    return
```

When I navigate through the system and press the Prim Algorithm button on the Home page, the following is output



And when I press the Home button, the following is output:



I realised that for returning to the home page, the username needs to be passed back because otherwise there is no account. Therefore, I created an attribute in Prim called username and passed the username value into it:

3.2.Q

```

if newType == "prim":
    try:
        from prim_page import Prim
        primRoot = Tk()
        primRoot.geometry('500x350') # to change
        primRoot.title("Algorithms Simulator and Teaching Tool")
        running = Prim(primRoot, username)
        primRoot.mainloop()
    except ImportError as e:
        print(f"Failed import Prim: {e}")
    return

```

3.2.R

```

class Prim(GraphInput, Menu): #HomeMain
    def __init__(self, master, pusername):
        #super().__init__(master, pusername)
        self.master = master
        print("in prim __init__ 1")
        GraphInput.__init__(self, master)
        print("in prim __init__ after GraphInput")
        Menu.__init__(self, master, 13, 12)
        print("in prim__init__ after Menu")

        self.username = pusername

```

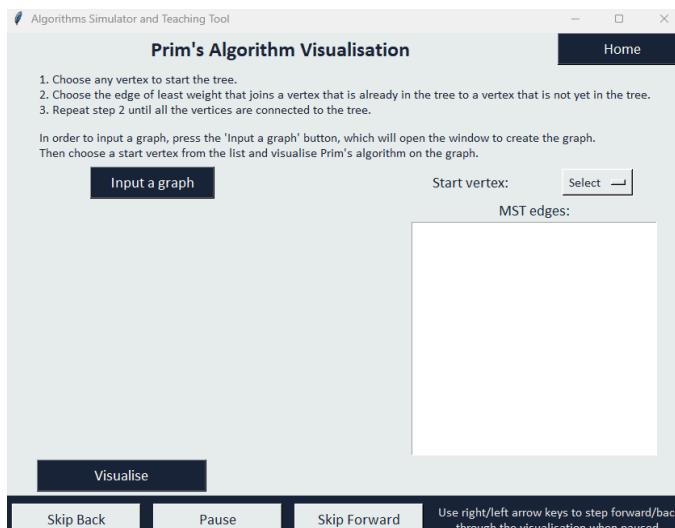
3.2.S

```

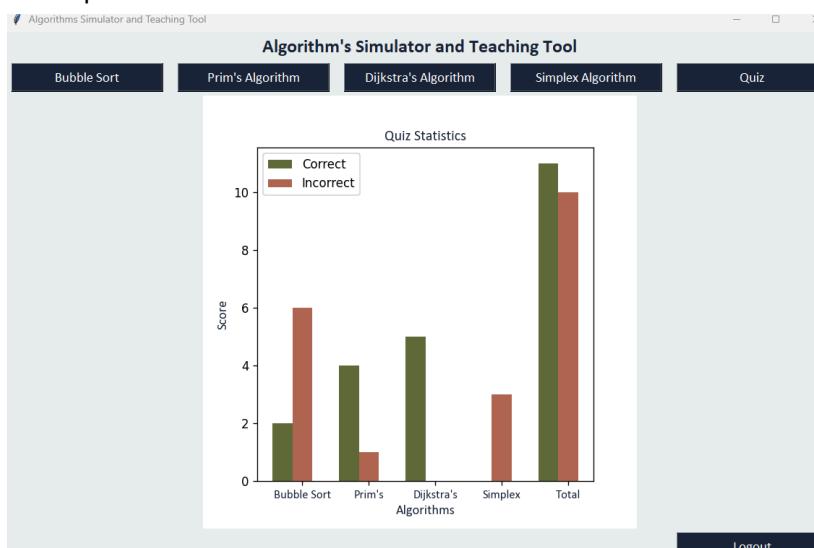
def openHome(self):
    closeOpen(self.master, "home", self.username)

```

When I now run the code and navigate to the prim page:



Then I press the home button:



This now works as expected so the error is fixed. Having now fixed this, I realise that this is a better solution to keeping the account, which I implemented in a more complicated way in the Bubble Sort class. Therefore, I am also going to implement a similar approach within this class.

3.2.T

```
class BubbleSort(Menu):
    def __init__(self, master, pUsername):
        #print("inside __init__ 1")
        #HomeMain.__init__(self, master, pUsername)
        #print("inside __init__ 2")
        Menu.__init__(self, master, 12, 19)
        self.numbers = []
        self.userEntries = []

        self.username = pUsername

        self.master.configure(bg="#eaebed")
        self.master.geometry("1105x905") # required here

        self.sortOrder = StringVar(value="Ascending")

        self.invalidMessage = None

        self.fig = None
        self.ax = None
        self.canvas = None

        self.visualiseText = None

        #print(self.numbers)
        #print(self.userEntries)

        #self.hideHomeWidgets()
        self.bubbleSortWidgets()
        self.menuWidgets()
```

3.2.U

```
def openHome(self):
    # inherited self.currentAccUsername from HomeMain
    closeOpen(self.master, "home", self.username)
```

3.2.V

```
elif newType == "bubble sort":
    try:
        from bubble_sort_page import BubbleSort
        bubbleSortRoot = Tk()
        bubbleSortRoot.geometry('400x1000') # to change
        bubbleSortRoot.title("Algorithms Simulator and Teaching Tool")
        running = BubbleSort(bubbleSortRoot, username)
        bubbleSortRoot.mainloop()
    except ImportError as e:
        print(f"Failed import BubbleSort: {e}")
    return
```

When I now navigate to the home page and click on the Bubble Sort button, the following is output:



The second issue I encountered in white box testing was that a graph that was not fully connected, e.g. has 2 loops but these are not connected, will still allow this to be submitted. Then, when the visualisation is run, the ‘Graph not fully connected’ message is displayed when this point is reached in execution. However, the algorithm will then continue executing and display an ‘MST found message’. I think this is because the if statement for when the last step is reached will be entered even when the graph is not connected. Therefore, I will add an extra condition to check how many edges are in the MST. Because of the nature of an MST, there will be $n-1$ edges if there are n vertices²⁷. Therefore, I added a check for how many edges are in the MST in each if statement.

3.2.W

```

def primAnimate(self):
    # animates at start when not paused
    print("in primAnimate")

    # test steps:
    if not self.steps:
        print("Error: no steps recorded in self.steps")
        return

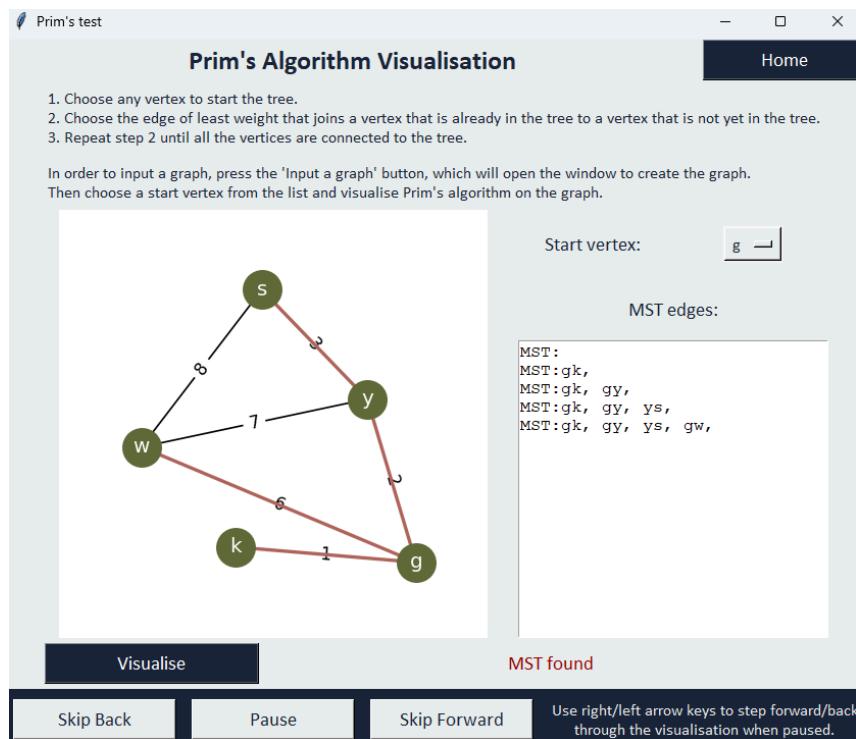
    if not self.isPaused and self.currentStep < len(self.steps):
        mstStep = self.steps[self.currentStep] # current step#s edges
        self.updatePrimGraph(mstStep)
        # show all edges currently in the mst in the Text section
        self.visualiseText.insert(tk.END, "MST:" + " ".join([f"{u}-{v}" for u,v in mstStep]) + "\n")
        self.visualiseText.see(tk.END)

        self.currentStep += 1
        self.master.after(750, self.primAnimate)

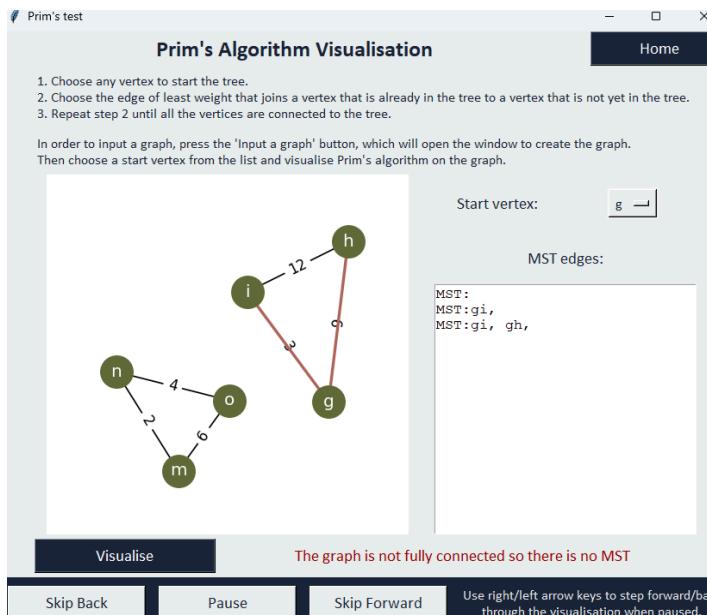
    elif not self.isPaused and self.currentStep == len(self.steps):
        mstStep = self.steps[self.currentStep - 1]
        print(mstStep)
        print(str(len(mstStep)))
        if len(mstStep) == len(self.graph)-1:
            self.updatePrimGraph(mstStep)
            self.invalidMessage["text"] = "MST found"
        else:
            self.invalidMessage["text"] = "The graph is not fully connected so there is no MST"

```

When I run the code now, on a valid connected graph, the following is output:



And on an unconnected graph, the following is output:



Both outputs are as expected so this error has now been resolved. All development, testing and debugging of the Prim's Algorithm Visualisation page is now complete. I shall now comment out the print statements and tidy up the code.

ITERATION 2 - DEVELOPER REVIEW

Overall, this iteration went quite smoothly as I already had a set plan and structure on how to implement this visualisation of this algorithm, having completed the development of the Bubble Sort visualisation page. The issues encountered during development were mostly to do with the nature of the matplotlib and NetworkX libraries as I have not used these before beginning the development of this system. In addition, issues found during the white box testing stage were easily fixed.

ITERATION 3 - DIJKSTRA'S VISUALISATION PAGE FUNCTIONALITY AND GUI

In this iteration, I shall complete the Dijkstra's Visualisation page, which will include the setting up of the GUI, the functionality of the algorithm, which will be specifically implemented to make use of the Menu, which was developed in Prototype 2 Iteration 3, and the implementation of the Graph Input window and placement of the submitted graph in the Dijkstra's Visualisation window.

Firstly, I set up the main structure of the Dijkstra class and the GUI with the following code:

3.3.a

```

1 # imports specific to the Dijkstra's page
2 from tkinter import *
3 import tkinter as tk
4 from NEA_utilities import closeOpen, fontLarge, fontMedium, fontSmall
5 from graph_input_page import GraphInput
6 import matplotlib.pyplot as plt
7 from matplotlib.backends.backend_tkagg import FigureCanvasTkAgg
8 import networkx as nx
9 from menu_code import Menu

```

3.3.b

```

class Dijkstra(GraphInput, Menu):
    def __init__(self, master, pUsername):
        self.master = master
        #print("in prim __init__ 1")
        GraphInput.__init__(self, master)
        #print("in prim __init__ after GraphInput")
        Menu.__init__(self, master, 14, 16)

        self.username = pUsername

        self.dijkstraWidgets()
        self.menuWidgets()
        # check menu text

```

3.3.c

```

def graphInputWindow(self):
    print("in start graph input window")
    self.hideDijkstraWidgets()
    self.hideMenuWidgets()
    self.graphInputPageWidgets()

def hideDijkstraWidgets(self):
    print("in hide dijkstra widgets")
    for widget in self.dijkstraFrame.winfo_children():
        widget.grid_forget()
    self.title.grid_forget()

def validate(self):
    print("in validate")
    # checks start and end vertex entered

```

3.3.d

```

def dijkstraExtraWidgets(self):
    print("in dijkstra extra widgets")
    # destroy Graph Input button
    # create graph
    # fill in vertex values in distance table
    # update start and end vertex values

def openHome(self):
    closeOpen(self.master, "home")

def updateDijkstraGraph(self):
    print("in update dijkstra graph")

def dijkstraSteps(self):
    print("in dijkstra steps")

def dijkstraAnimate(self):
    print("in dijkstra animate")

```

3.3.e

```
def dijkstraWidgets(self):
    self.master.configure(bg="#eaebed")
    self.master.geometry("845x640")

    self.title = Label(self.master, text="Dijkstra's Algorithm Visualisation")
    self.title.grid(row=0, column=0, columnspan=14)

    Button(self.master, text="Home", command=self.openHome, bg="#1b263b", fg="white", font=fontMedium, width=15).grid(row=0, column=14, columnspan=2)

    self.dijkstraFrame = Frame(self.master, bg="#eaebed")
    self.dijkstraFrame.grid(row=1, column=0, columnspan=16, rowspan=13)

    Label(self.dijkstraFrame, text=
        "1. Set the initial distance from the start values for all nodes (0 for the start node and infinity for all other nodes).\n"
        "2. Find the node with the shortest distance from the start that has not been visited and look at all the unvisited connected nodes.\n"
        "3. Calculate the distance from the start for each of the unvisited connected nodes. \n"
        "4. If the distance calculated is less than the current shortest distance from the start, set the previous node for that connected node to be the current node. \n"
        "5. Then set the current node as visited.\n"
        "6. Repeat steps 2 to 5 until all nodes are set to visited.\n"
        "To find the shortest path through backtracking:\n"
        "7. Start from the goal node.\n"
        "8. Add the 'previous node' to the start of a list.\n"
        "9. Repeat step 7 until the start node is reached.",
        font=fontSmall,
        fg="#1b263b",
        bg="#eaebed",
        justify="left").grid(row=1, column=0, rowspan=3, columnspan=16)

    Label(self.dijkstraFrame,
        text="In order to input a graph, press the 'Input a graph' button, which will open the window to create the graph.\n"
            "Then choose a start vertex from the list and visualise Prim's algorithm on the graph.", 
        font=fontSmall,
        fg="#1b263b",
        bg="#eaebed",
        justify="left").grid(row=4, column=0, rowspan=2, columnspan=16)
```

3.3.f

```
self.graphInputButton = Button(self.dijkstraFrame, ext="Input a graph", bg="#1b263b", fg="white", font=fontMedium, width=20, command=self.graphInputWindow)
self.graphInputButton.grid(row=6, column=1, columnspan=2)

# start vertex:
Label(self.dijkstraFrame, text="Start vertex:", bg="#eaebed", fg="#1b263b", font=fontMedium).grid(row=12, column=0, columnspan=2)

self.startVertexOptions = ["Select"] # initially Select - changes to the list of vertices once graph input
self.startVertexChoice = StringVar(self.dijkstraFrame)
self.startVertexChoice.set("Select")

self.startVertexDropdown = OptionMenu(self.dijkstraFrame, self.startVertexChoice, *self.startVertexOptions)
self.startVertexDropdown.grid(row=12, column=2, columnspan=2)
self.startVertexDropdown.config(font=fontSmall, bg="#eaebed", fg="#1b263b")
self.startVertexDropdown["menu"].config(font=fontSmall, bg="#eaebed", fg="#1b263b")

#end vertex:
Label(self.dijkstraFrame, text="End vertex:", bg="#eaebed", fg="#1b263b", font=fontMedium).grid(row=13, column=0, columnspan=2)

self.endVertexOptions = ["Select"] # initially Select - changes to the list of vertices once graph input
self.endVertexChoice = StringVar(self.dijkstraFrame)
self.endVertexChoice.set("Select")

self.endVertexDropdown = OptionMenu(self.dijkstraFrame, self.endVertexChoice, *self.endVertexOptions)
self.endVertexDropdown.grid(row=13, column=2, columnspan=2)
self.endVertexDropdown.config(font=fontSmall, bg="#eaebed", fg="#1b263b")
self.endVertexDropdown["menu"].config(font=fontSmall, bg="#eaebed", fg="#1b263b")

# distance table:

Button(self.dijkstraFrame, text="Visualise", command=self.validate, bg="#1b263b", fg="white", font=fontMedium, width=20).grid(row=12, column=5, columnspan=3, padx=10, pady=5)

self.invalidMessage = Label(self.dijkstraFrame, text="invalid message", font=fontMedium, bg="#eaebed", fg="#990000")
self.invalidMessage.grid(row=12, column=8, columnspan=8)
```

As I have already developed the validation method for Prim's, I copied the code and extended it to check the start and end vertices for Dijkstra's:

3.3.g

```
def validate(self):
    print("in validate")
    # checks start and end vertex entered
    if self.startVertexChoice == "Select": # select is default option where th
        self.invalidMessage["text"] = "Invalid: please enter a start vertex."
        # start vertex not chosen
        print("invalid start")
        return
    if self.endVertexChoice == "Select":
        print("invalid end")
        self.invalidMessage["text"] = "Invalid: please enter an end vertex."
        return
    print("valid")
    self.invalidMessage["text"] = ""
    self.dijkstraSteps()
    self.dijkstraAnimate()
```

Additionally, a check needs to be done to make sure that the start and end vertices are not the same because this would cause an error in the code:

3.3.h

```
if self.startVertexChoice == self.endVertexChoice:
    print("invalid same")
    self.invalidMessage["text"] = "Invalid: please enter a start and end vertex that are different."
    return
```

Then, I wrote the following code for the dijkstraExtraWidgets method. This is the method that is accessed once the graph has been validated. For this, I added functionality similar to the Prim version of this method to fill in the start and end vertex choices for the dropdown menu. I also destroyed the graphInput button as the graph replaces this:

3.3.i

```
def dijkstraExtraWidgets(self):
    print("in dijkstra extra widgets")
    self.menuWidgets()
    # destroy Graph Input button
    self.graphInputButton.destroy()

    # start vertex choices:
    self.startVertexOptions = list(self.graph.nodes)
    self.startVertexChoice.set(self.startVertexOptions[0] if self.startVertexOptions else "Select")
    self.startVertexDropdown["menu"].delete(0, "end")
    for vertex in self.startVertexOptions:
        self.startVertexDropdown["menu"].add_command(label=vertex, command=lambda v=vertex: self.startVertexChoice.set(v))

    # end vertex choices:
    self.endVertexOptions = list(self.graph.nodes)
    self.endVertexChoice.set(self.startVertexOptions[0] if self.startVertexOptions else "Select")
    self.endVertexDropdown["menu"].delete(0, "end")
    for vertex in self.endVertexOptions:
        self.endVertexDropdown["menu"].add_command(label=vertex, command=lambda v=vertex: self.startVertexChoice.set(v))

    # create axes to plot graph on
    self.fig, self.ax = plt.subplots(figsize=(3.5,3.5))
    self.dijkstraGraphCanvas = FigureCanvasTkAgg(self.fig, self.dijkstraFrame)
    self.dijkstraGraphCanvas.get_tk_widget().grid(row=6, column=0, rowspan=6, columnspan=6)
    # draw graph on axes
    pos = nx.get_node_attributes(self.graph, 'pos')
    nx.draw(self.graph, pos, with_labels=True, node_color="#606c38", edge_color='black', node_size=500, font_color='white', ax=self.ax)
    edge_labels = nx.get_edge_attributes(self.graph, 'weight')
    nx.draw_networkx_edge_labels(self.graph, pos, edge_labels=edge_labels, ax=self.ax)
    self.dijkstraGraphCanvas.draw()
```

Then, I wrote the following code to create the vertex/distance table. For this, I created a new frame because this allows the cells to be created and for everything to be contained together:

3.3.i

```

# create frame to hold the table
self.tableFrame = Frame(self.dijkstraFrame, bg="#eaebed")
self.tableFrame.grid(row=6, column=8, rowspan=8, columnspan=6)

headers = ["Vertex", "Shortest Distance", "Previous Vertex", "Visited"]
for column, text in enumerate(headers): # enumerate returns index and value
    label = Label(self.tableFrame, text=text, font=(fontSmall, "bold"), borderwidth=1, relief="solid", padx=5, pady=5)
    label.grid(row=6, column=(8+column), sticky="nsew")

# fill in vertex values in distance table
self.tableData = {}
for row, vertex in enumerate(self.graph.nodes(), start=1):
    self.tableData[vertex] = {}
    # vertex name:
    vLabel = Label(self.tableFrame, text=vertex, font=fontSmall, borderwidth=1, relief="solid", padx=5, pady=5)
    vLabel.grid(row=row, column=8, sticky="nsew")
    self.tableData[vertex]["vertex"] = vLabel
    # shortest distance: initialise infinity except start:
    sdLabel = Label(self.tableFrame, text="∞", font=fontSmall, borderwidth=1, relief="solid", padx=5, pady=5)
    sdLabel.grid(row=row, column=10, sticky="nsew")
    self.tableData[vertex]["shortest distance"] = sdLabel
    # previous vertex: initialise to -
    pvLabel = Label(self.tableFrame, text="-", font=fontSmall, borderwidth=1, relief="solid", padx=5, pady=5)
    pvLabel.grid(row=row, column=12, sticky="nsew")
    self.tableData[vertex]["previous vertex"] = pvLabel
    # visited: initialise empty
    visitedLabel = Label(self.tableFrame, text="", font=fontSmall, borderwidth=1, relief="solid", padx=5, pady=5)
    visitedLabel.grid(row=row, column=14, sticky="nsew")
    self.tableData[vertex]["visited"] = visitedLabel

```

In addition, I extended the backToPage method in the graphInput class to include the functionality for dijkstra's, therefore allowing this dijkstraExtraWidgets method to be accessed:

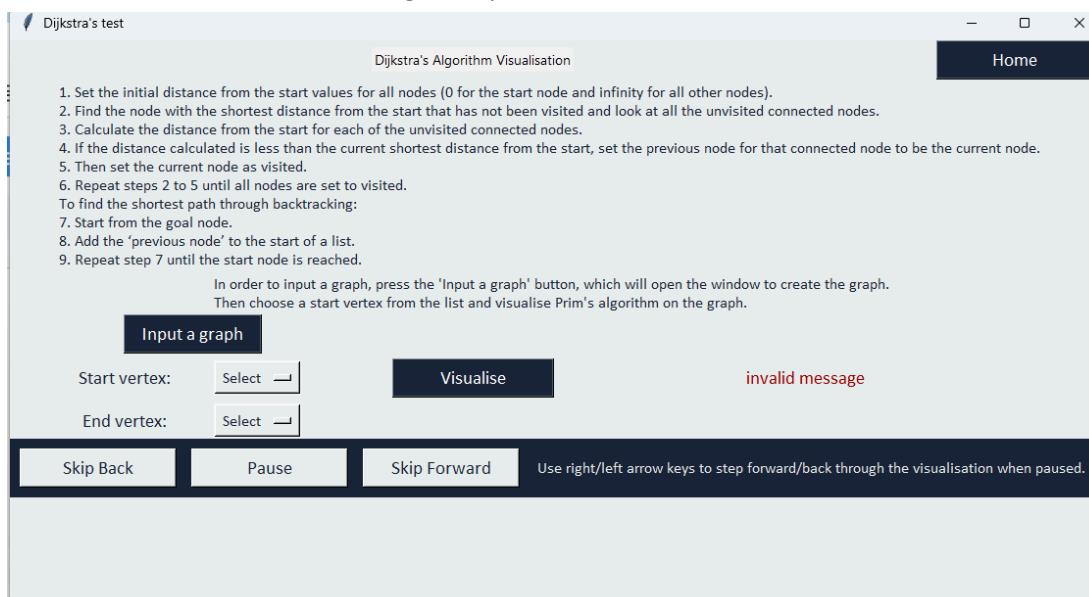
3.3.j

```

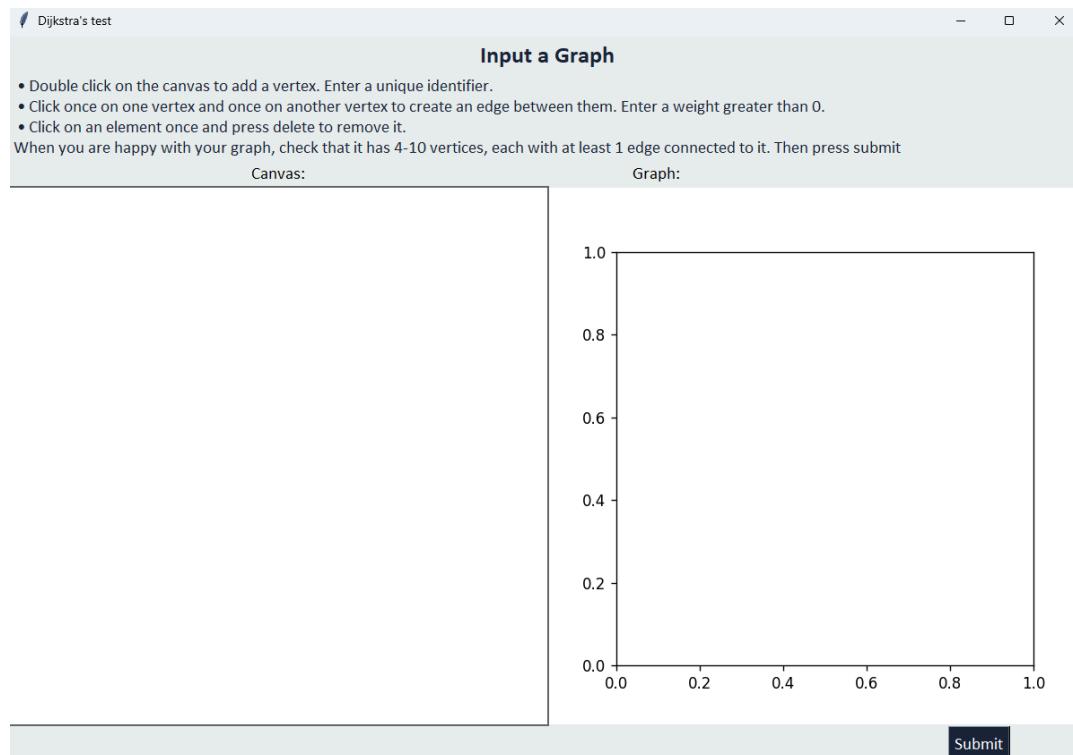
def backToPage(self):
    if self.validated == True:
        self.graphInputHideWidgets()
        if hasattr(self, "primWidgets"):
            self.primWidgets()
            self.primExtraWidgets()
        elif hasattr(self, "dijkstraWidgets"):
            self.dijkstraWidgets()
            self.dijkstraExtraWidgets()

```

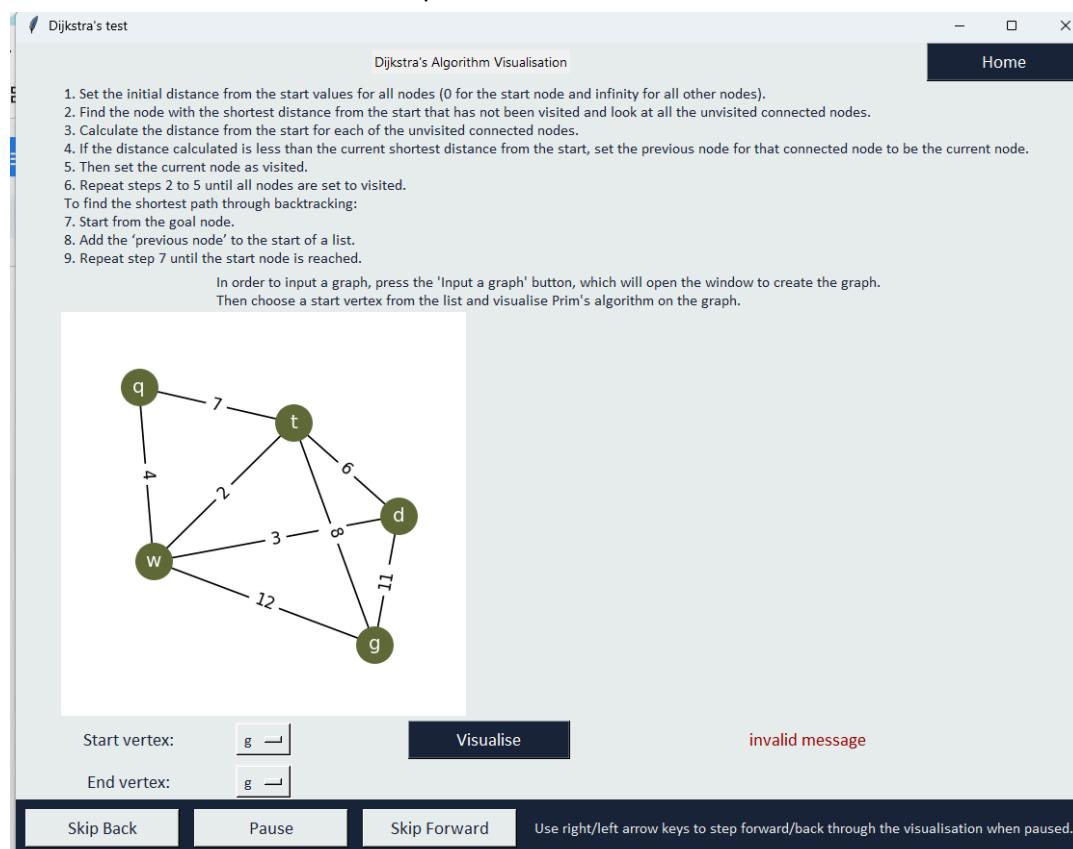
When I run the code, the following is output:

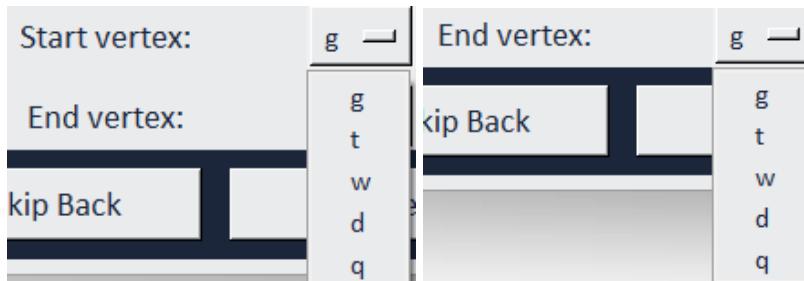


When the Graph Input button is pressed:



When the Submit button has been pressed:





```

in start graph input window
in hide dijkstra widgets
in dijkstra extra widgets
Exception in Tkinter callback
Traceback (most recent call last):
  File "C:\Users\ingri\AppData\Local\Programs\Thonny\lib\tkinter\__init__.py", line 19
    return self.func(*args)
  File "C:\Users\ingri\OneDrive\Documents\QM\QM School work\A Level\Computer Science\N
lidaGraph
    self.backToPage()
  File "C:\Users\ingri\OneDrive\Documents\QM\QM School work\A Level\Computer Science\N
ckToPage
    self.dijkstraExtraWidgets()
  File "C:\Users\ingri\OneDrive\Documents\QM\QM School work\A Level\Computer Science\N
straExtraWidgets
    for column, text in edumerate(headers): # enumerate returns index and value
NameError: name 'edumerate' is not defined

```

As is clear from the error message, the issue with the vertex table not showing was due to a spelling mistake. Otherwise the other functionality seems to be working correctly. To fix this issue, I replaced the code with the following:

3.3.k

```

for column, text in enumerate(headers): # enumerate returns index and value
    label = Label(self.tableFrame, text=text, font=(fontSmall, "bold"), bor
    label.grid(row=6, column=(8+column), sticky="nsew")

```

When I now run the code, navigate to the graph input page and then press submit, the following is output:

```

in start graph input window
in hide dijkstra widgets
in dijkstra extra widgets
Exception in Tkinter callback
Traceback (most recent call last):
  File "C:\Users\ingri\AppData\Local\Programs\Thonny\lib\tkinter\__init__.py", line 1921, in __call__
    return self.func(*args)
  File "C:\Users\ingri\OneDrive\Documents\QM\QM School work\A Level\Computer Science\NEA\graph_input_page.py", line 88, in va
lidaGraph
    self.backToPage()
  File "C:\Users\ingri\OneDrive\Documents\QM\QM School work\A Level\Computer Science\NEA\graph_input_page.py", line 98, in ba
ckToPage
    self.dijkstraExtraWidgets()
  File "C:\Users\ingri\OneDrive\Documents\QM\QM School work\A Level\Computer Science\NEA\dijkstra_page.py", line 168, in dijk
straExtraWidgets
    label = Label(self.tableFrame, text=text, font=(fontSmall, "bold"), borderwidth=1, relief="solid", padx=5, pady=5)
  File "C:\Users\ingri\AppData\Local\Programs\Thonny\lib\tkinter\__init__.py", line 3187, in __init__
    Widget.__init__(self, master, 'label', cnf, kw)
  File "C:\Users\ingri\AppData\Local\Programs\Thonny\lib\tkinter\__init__.py", line 2601, in __init__
    self.tk.call(
_tkinter.TclError: expected integer but got "bold"

```

Whilst I wanted to be able to make the table headers to be bold, the way I tried to implement this doesn't work correctly. For now, I will take this out and I may add some code later to make the headers bold. However, this will be when I refine the GUI once all functionality has been completed. After taking out the "bold", the following is output after inputting a graph:

Dijkstra's test

Dijkstra's Algorithm Visualisation

Home

- Set the initial distance from the start values for all nodes (0 for the start node and infinity for all other nodes).
- Find the node with the shortest distance from the start that has not been visited and look at all the unvisited connected nodes.
- Calculate the distance from the start for each of the unvisited connected nodes.
- If the distance calculated is less than the current shortest distance from the start, set the previous node for that connected node to be the current node.
- Then set the current node as visited.
- Repeat steps 2 to 5 until all nodes are set to visited.
- To find the shortest path through backtracking:
- Start from the goal node.
- Add the 'previous node' to the start of a list.
- Repeat step 7 until the start node is reached.

In order to input a graph, press the 'Input a graph' button, which will open the window to create the graph. Then choose a start vertex from the list and visualise Dijkstra's algorithm on the graph.

Vertex	Shortest Distance	Previous Vertex	Visited
g	∞		
q	∞		
x	∞		
a	∞		
r	∞		

Start vertex: Visualise invalid message

End vertex:

Skip Back | Pause | Skip Forward | Use right/left arrow keys to step forward/back through the visualisation when paused.

With the vertex table, the headers seem to be at the bottom of the table not the top and some fields are not showing up properly. To try and fix this, I changed the grid positions of the headers to make sure that they all have colspan=2 like the other cells and I made sure that they would be placed in the correct spot with this code:

3.3.i

```
# create frame to hold the table
self.tableFrame = Frame(self.dijkstraFrame, bg="#eaebed")
self.tableFrame.grid(row=6, column=8, rowspan=8, columnspan=6)

headers = ["Vertex", "Shortest Distance", "Previous Vertex", "Visited"]
for column, text in enumerate(headers): # enumerate returns index and value
    label = Label(self.tableFrame, text=text, font=fontSmall, borderwidth=1, relief="solid")
    if text == "Vertex":
        label.grid(row=6, column=8, columnspan=2)
    else:
        label.grid(row=6, column=(8+(2*column)), sticky="nsew", columnspan=2)
```

When I now run the code, the following is output after submitting a graph:

Dijkstra's test

Dijkstra's Algorithm Visualisation

Home

- Set the initial distance from the start values for all nodes (0 for the start node and infinity for all other nodes).
- Find the node with the shortest distance from the start that has not been visited and look at all the unvisited connected nodes.
- Calculate the distance from the start for each of the unvisited connected nodes.
- If the distance calculated is less than the current shortest distance from the start, set the previous node for that connected node to be the current node.
- Then set the current node as visited.
- Repeat steps 2 to 5 until all nodes are set to visited.
- To find the shortest path through backtracking:
- Start from the goal node.
- Add the 'previous node' to the start of a list.
- Repeat step 7 until the start node is reached.

In order to input a graph, press the 'Input a graph' button, which will open the window to create the graph.
Then choose a start vertex from the list and visualise Prim's algorithm on the graph.

Vertex	Shortest Distance	Previous Vertex	Visited
f	∞	-	
w	∞	-	
d	∞	-	
s	∞	-	
z	∞	-	
t	∞	-	

Start vertex: End vertex:

Visualise

invalid message

Skip Back | Pause | Skip Forward | Use right/left arrow keys to step forward/back through the visualisation when paused.

This is now as expected. I will further refine the GUI later but I am going to change the font to be the predefined fontMedium because this will make checking that values are updated correctly easier for me as I develop. Additionally, I changed the rowspan and colspan values around for the tableFrame because I realised that I had put them in the wrong order. When I now run the code, the following is output:

Dijkstra's test

Dijkstra's Algorithm Visualisation

Home

- Set the initial distance from the start values for all nodes (0 for the start node and infinity for all other nodes).
- Find the node with the shortest distance from the start that has not been visited and look at all the unvisited connected nodes.
- Calculate the distance from the start for each of the unvisited connected nodes.
- If the distance calculated is less than the current shortest distance from the start, set the previous node for that connected node to be the current node.
- Then set the current node as visited.
- Repeat steps 2 to 5 until all nodes are set to visited.
- To find the shortest path through backtracking:
- Start from the goal node.
- Add the 'previous node' to the start of a list.
- Repeat step 7 until the start node is reached.

In order to input a graph, press the 'Input a graph' button, which will open the window to create the graph.
Then choose a start vertex from the list and visualise Prim's algorithm on the graph.

Vertex	Shortest Distance	Previous Vertex	Visited
g	∞	-	
h	∞	-	
y	∞	-	
w	∞	-	
t	∞	-	

Start vertex: End vertex:

Visualise

invalid message

Skip Back | Pause | Skip Forward | Use right/left arrow keys to step forward/back through the visualisation when paused.

This is now as expected and the table values are a bit clearer to read.

Now I am going to test if the validation works correctly for when I press visualise without entering 2 different vertices:

Vertex	Shortest Distance	Previous Vertex	Visited
g	∞	-	
w	∞	-	
h	∞	-	
r	∞	-	

Start vertex: End vertex:

Visualise

Skip Back | Pause | Skip Forward | Use right/left arrow keys to step forward/back through the visualisation when paused.

```
in start graph input window
in hide dijkstra widgets
in dijkstra extra widgets
in validate
valid
in dijkstra steps
in dijkstra animate
```

This is not as expected because the values are the same but the checks seem to have been passed over. As I am not sure what is causing this issue, I added a .get() for each attribute being used because the values of them are what I want to be compared.

3.3.m

```
def validate(self):
    print("in validate")
    # checks start and end vertex entered
    if self.startVertexChoice.get() == "Select": # select is default option where there is
        self.invalidMessage["text"] = "Invalid: please enter a start vertex."
        # start vertex not chosen
        print("invalid start")
        return
    if self.endVertexChoice.get() == "Select":
        print("invalid end")
        self.invalidMessage["text"] = "Invalid: please enter an end vertex."
        return
    if self.startVertexChoice.get() == self.endVertexChoice.get():
        print("invalid same")
        self.invalidMessage["text"] = "Invalid: please enter a start and end vertex that are different"
        return
```

When I try press validate now, the following is output:

Dijkstra's test

Dijkstra's Algorithm Visualisation

Home

- Set the initial distance from the start values for all nodes (0 for the start node and infinity for all other nodes).
- Find the node with the shortest distance from the start that has not been visited and look at all the unvisited connected nodes.
- Calculate the distance from the start for each of the unvisited connected nodes.
- If the distance calculated is less than the current shortest distance from the start, set the previous node for that connected node to be the current node.
- Then set the current node as visited.
- Repeat steps 2 to 5 until all nodes are set to visited.
- To find the shortest path through backtracking:
- Start from the goal node.
- Add the 'previous node' to the start of a list.
- Repeat step 7 until the start node is reached.

In order to input a graph, press the 'Input a graph' button, which will open the window to create the graph. Then choose a start vertex from the list and visualise Dijkstra's algorithm on the graph.

Vertex	Shortest Distance	Previous Vertex	Visited
t	∞	-	
y	∞	-	
r	∞	-	
e	∞	-	

Start vertex: Visualise Invalid: please enter a start and end vertex that are different.

End vertex:

Skip Back Pause Skip Forward Use right/left arrow keys to step forward/back through the visualisation when paused.

```

in start graph input window
in hide dijkstra widgets
in dijkstra extra widgets
in validate
invalid same
    
```

This seems to have fixed the issue. Now all validation is occurring correctly because the graph input's validation has already been implemented and tested as this is a reusable component.

Now, I am going to complete the functionality of the `dijkstraSteps` method as this is the main functionality of the section and is focused on the actual algorithm for Dijkstra's. Firstly, I implemented the `dijkstraSteps` method. For this, I implemented the algorithm using a priority queue to store the connected edges and iterating through it and checking if the total distance is smaller than the current shortest distance to the node.

3.3.n

```

def dijkstraSteps(self):
    print("in dijkstra steps")
    start = self.startVertexChoice.get()
    end = self.endVertexChoice.get()

    self.distances = {node: float("inf") for node in self.graph.nodes}
    self.previous = {node: None for node in self.graph.nodes}
    self.distances[start] = 0

    self.steps = []
    queue = [(0, start)] # this is a priority queue for storing (distance, vertex)
    visited = set()
        
```

3.3.0

```
while queue:
    currentDistance, currentNode = heapq.heappop(queue)
    if currentNode in visited:
        # continue = go to process next vertex by going to the next iteration of while queue
        # this maintains Dijkstra's O((v+E)logV) complexity
        continue
    visited.add(currentNode)

    # store step for animation:
    checkingEdges = [(currentNode, neighbour) for neighbour in self.graph.neighbors(currentNode) if neighbour not in visited]
    self.steps.append((visited.copy(), currentNode, checkingEdges.copy(), currentDistance))

    # check for shortest connected edge/node
    for neighbour in self.graph.neighbors(currentNode):
        if neighbour in visited:
            continue
        edgeWeight = self.graph[currentNode][neighbour]["weight"]
        newDistance = currentDistance + edgeWeight
        if newDistance < self.distances[neighbour]:
            self.distances[neighbour] = newDistance
            self.previous[neighbour] = currentNode
            heapq.heappush(queue, (newDistance, neighbour))

# backtrack for shortest path:
path = []
current = end
while current:
    path.append(current)
    current = self.previous[current]
path.reverse()
self.steps.append(((“shortest path”), path))
```

After this, I implemented the updateDijkstraGraph method, which highlights the nodes and edges being checked and the shortest path. As this is very similar to the Prim’s version of updating the graph, a lot of the code could be copied. I made it so that the visited nodes are highlighted in red and the current node and its edges are highlighted in blue. These colours will be changed later on.

3.3.p

```

def updateDijkstraGraph(self, visitedNodes, currentNode=None, checkingEdges=[]):
    print("in update dijkstra graph")
    self.ax.clear()
    pos = nx.get_node_attributes(self.graph, "pos")

    # draw base graph:
    nx.draw(self.graph, pos, with_labels=True, node_color='#606c38', edge_color='black', node_size=500, font_color='white', ax=self.ax)
    edge_labels = nx.get_edge_attributes(self.graph, "weight")
    nx.draw_networkx_edge_labels(self.graph, pos, edge_labels=edge_labels, ax=self.ax)

    # highlighted visited nodes:
    nx.draw_networkx_nodes(self.graph, pos, nodelist=visitedNodes, node_color="red", ax=self.ax)

    # highlight current node:
    if currentNode:
        nx.draw_networkx_nodes(self.graph, pos, nodelist=[currentNode], node_color="blue", ax=self.ax)

    # highlight edges being checked:
    nx.draw_networkx_edges(self.graph, pos, edgelist=checkingEdges, edge_color="blue", width=2, ax=self.ax)

    self.dijkstraGraphCanvas.draw()

```

Finally, I implemented the dijkstraAnimate method, which calls the updateDijkstraGraph method and updates the table or outputs the shortest route.

3.3.q

```

def dijkstraAnimate(self, stepI=0):
    print("in dijkstra animate")
    if stepI < len(self.steps):
        step = self.steps[stepI]

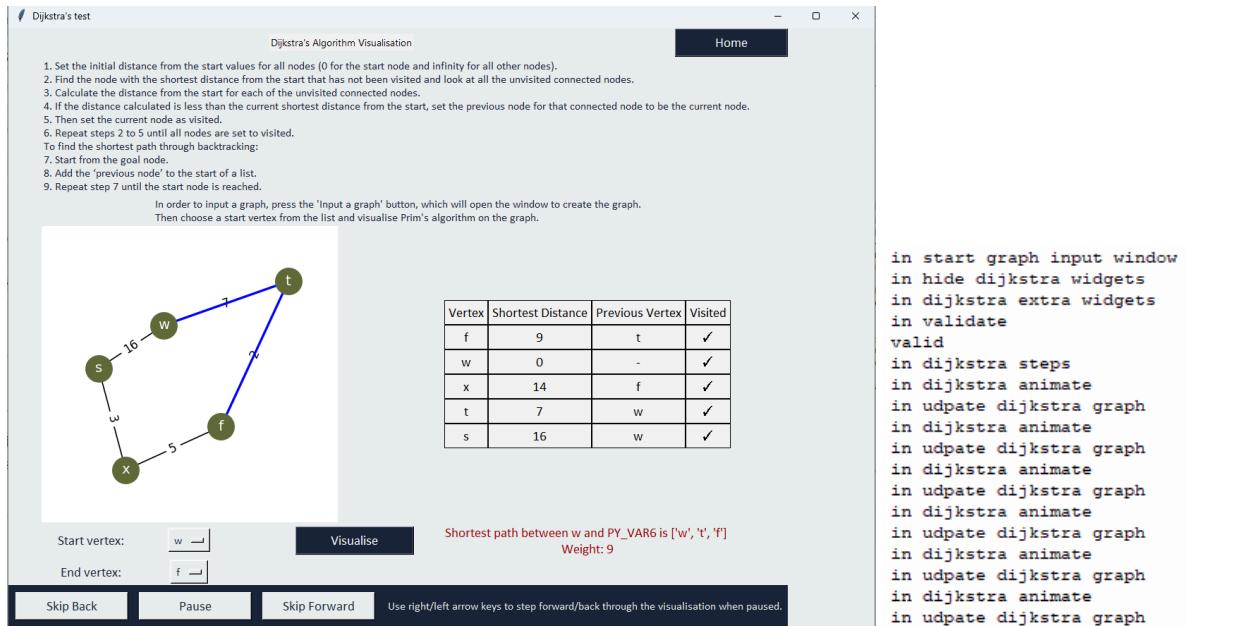
        if step[0] == "shortest path":
            # step[1] contains the shortest path as a list of vertices
            # therefore adding the combos of edges into the list path_edges
            # e.g. if step[1] = [A, B, C] then path_edges = [(A,B), {B,C}]
            pathEdges = [(step[1][i], step[1][i+1]) for i in range(len(step[1])-1)]
            self.updateDijkstraGraph([], checkingEdges=pathEdges)
            self.invalidMessage["text"] = f"Shortest path between {self.startVertexChoice.get()} and {self.endVertexChoice} is {step[1]} \nWeight: {self.distances[self.endVertexChoice.get()]}"
        else:
            visitedNodes, currentNode, checkingEdges, currentDistance = step
            self.updateDijkstraGraph(visitedNodes, currentNode, checkingEdges)

        # update table:
        self.tableData[currentNode]["shortest distance"]["text"] = str(currentDistance)
        self.tableData[currentNode]["previous vertex"]["text"] = self.previous[currentNode] if self.previous[currentNode] else "-"
        self.tableData[currentNode]["visited"]["text"] = "✓"

    self.master.after(750, self.dijkstraAnimate, stepI+1)

```

When I run the code, the following is output:



The functionality is as expected so I am now going to extend the methods in the menu section to allow the full functionality for this section to be completed. Below is the code I wrote to extend each method:

3.3.r

```

def togglePlayPause(self):
    self.isPaused = not self.isPaused # switches the state
    self.playPauseButton.config(text="Pause" if not self.isPaused else "Play") # switches the text
    if not self.isPaused: # ie displays Pause but the visualisation is playing
        #self.playAnimation()
        if hasattr(self, "bubbleSortAnimate"):
            self.bubbleSortAnimate()
        elif hasattr(self, "primAnimate"):
            self.primAnimate()
        elif hasattr(self, "dijkstraAnimate"):
            print("in dijkstra toggle play pause")
            self.dijkstraAnimate()

def skipBack(self):
    self.currentStep = 0
    self.isPaused = True
    if hasattr(self, "updateChart"): # note that methods count as attributes
        self.updateChart(self.steps[self.currentStep])
    elif hasattr(self, "updatePrimGraph"):
        self.updatePrimGraph(self.steps[self.currentStep])
    elif hasattr(self, "updateDijkstraGraph"):
        print("in dijkstra skip back")
        visitedNodes, currentNode, checkingEdges, currentDistance = self.steps[self.currentStep]
        self.updateDijkstraGraph(visitedNodes, currentNode, checkingEdges)
    # extend once implemented other algorithms
    self.invalidMessage["text"] = ""

```

3.3.s

```
def skipForward(self):
    self.currentStep = len(self.steps) - 1 # gets the last step index
    self.isPaused = True
    if hasattr(self, "updateChart"):
        self.updateChart(self.steps[self.currentStep])
    elif hasattr(self, "updatePrimGraph"):
        self.updatePrimGraph(self.steps[self.currentStep])
    elif hasattr(self, "updateDijkstraGraph"):
        print("in dijkstra skip forward")
        visitedNodes, currentNode, checkingEdges, currentDistance = self.steps[self.currentStep]
        self.updateDijkstraGraph(visitedNodes, currentNode, checkingEdges)
        self.invalidMessage["text"] = f"Shortest path between {self.startVertexChoice.get()} and {self.endVertexChoice} is {step[1]} \nWeight: {self.distances[self.endVertexChoice.get()]}"
    # extend once implemented other algorithms
```

3.3.t

```
def stepBack(self):
    if self.isPaused and self.currentStep > 0:
        self.currentStep -= 1
        if hasattr(self, "updateChart"):
            numbers = self.steps[self.currentStep]
            indices = self.stepsDictionary[numbers]
            self.updateChart(numbers, indices)
        elif hasattr(self, "updatePrimGraph"):
            mstStep = self.steps[self.currentStep]
            self.updatePrimGraph(mstStep)
            self.visualiseText.insert(tk.END, "MST:" + ".join([f'{u}-{v}', " for u,v in mstStep]) + "\n")
            self.visualiseText.see(tk.END)
        elif hasattr(self, "updateDijkstraGraph"):
            print("in dijkstra step back")
            visitedNodes, currentNode, checkingEdges, currentDistance = self.steps[self.currentStep]
            self.updateDijkstraGraph(visitedNodes, currentNode, checkingEdges)
            # update table:
            self.tableData[currentNode]["shortest distance"]["text"] = str(currentDistance)
            self.tableData[currentNode]["previous vertex"]["text"] = self.previous[currentNode] if self.previous[currentNode] else "-"
            self.tableData[currentNode]["visited"]["text"] = "✓"
    # extend once implemented other algorithms
```

3.3.u

```
def stepForward(self):
    if self.isPaused and self.currentStep < len(self.steps)-1:
        self.currentStep += 1
        if hasattr(self, "updateChart"): # note that method count as attributes
            numbers = self.steps[self.currentStep]
            indices = self.stepsDictionary[numbers]
            self.updateChart(numbers, indices)
        elif hasattr(self, "updatePrimGraph"):
            mstStep = self.steps[self.currentStep]
            self.updatePrimGraph(mstStep)
            self.visualiseText.insert(tk.END, "MST:" + " ".join([f"{u}-{v}" for u,v in mstStep]) + "\n")
            self.visualiseText.see(tk.END)
        elif hasattr(self, "updateDijkstraGraph"):
            print("in dijkstra step forward")
            visitedNodes, currentNode, checkingEdges, currentDistance = self.steps[self.currentStep]
            self.updateDijkstraGraph(visitedNodes, currentNode, checkingEdges)
            # update table:
            self.tableData[currentNode]["shortest distance"]["text"] = str(currentDistance)
            self.tableData[currentNode]["previous vertex"]["text"] = self.previous[currentNode] if self.previous[currentNode] else "-"
            self.tableData[currentNode]["visited"]["text"] = "✓"
    # extend once implemented other algorithms
```

When I run the code and press pause the following is output:

The screenshot shows a window titled "Dijkstra's test" with the sub-tittle "Dijkstra's Algorithm Visualisation". In the center is a graph with nodes r, y, w, g, and h. Node r is red, while others are green. Edges and weights are: r-w (20), r-y (4), w-y (5), w-h (8), and w-g (2). To the right is a table of shortest distances:

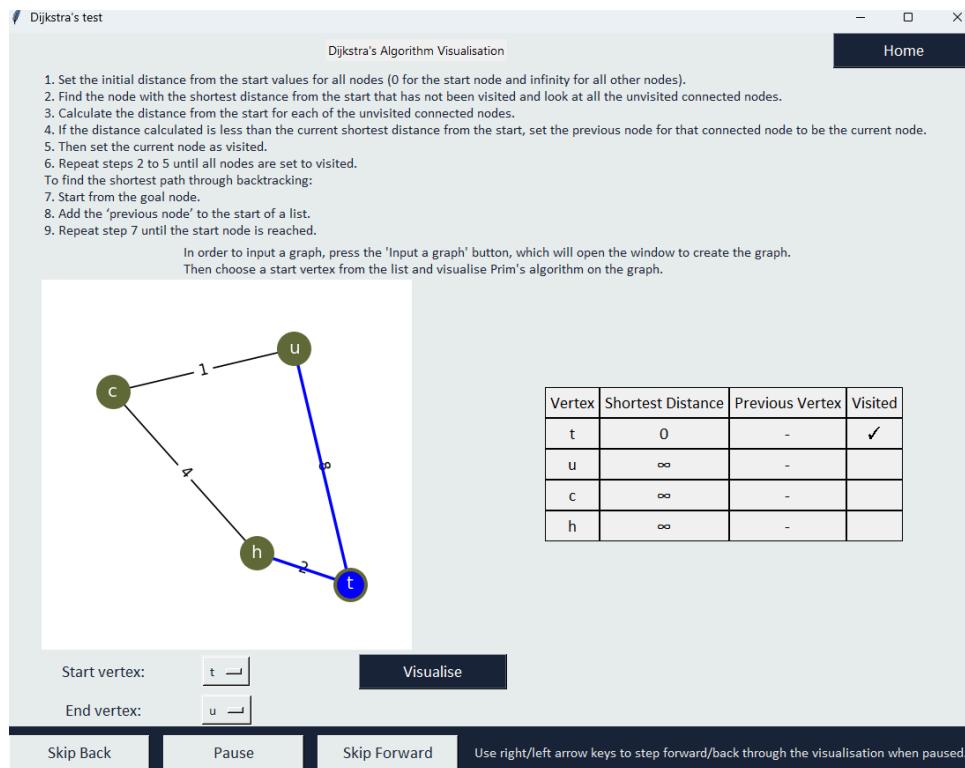
Vertex	Shortest Distance	Previous Vertex	Visited
g	∞	-	
y	4	r	✓
w	∞	-	
r	0	-	✓
h	∞	-	

Below the graph are input fields for "Start vertex" (r) and "End vertex" (h), and buttons for "Visualise", "Skip Back", "Pause", and "Skip Forward". A note says: "Use right/left arrow keys to step forward/back through the visualisation when paused."

```

in start graph input window
in hide dijkstra widgets
in dijkstra extra widgets
in validate
invalid same
in validate
valid
in dijkstra steps
in dijkstra animate
in update dijkstra graph
Exception in Tkinter callback
Traceback (most recent call last):
  File "C:\Users\ingri\AppData\Local\Programs\Thonny\lib\tkinter\__init__.py", line 1921, in __call__
    return self.func(*args)
  File "C:\Users\ingri\AppData\Local\Programs\Thonny\lib\tkinter\__init__.py", line 839, in callit
    func(*args)
TypeError: Dijkstra.dijkstraAnimate() takes 1 positional argument but 2 were given
    
```

As I wasn't sure where this issue was coming from, I ran the code without pressing pause to see what would happen and this was output:



```

in start graph input window
in hide dijkstra widgets
in dijkstra extra widgets
in validate
valid
in dijkstra steps
in dijkstra animate
in update dijkstra graph
Exception in Tkinter callback
Traceback (most recent call last):
  File "C:\Users\ingri\AppData\Local\Programs\Thonny\lib\tkinter\__init__.py", line 1921, in __call__
    return self.func(*args)
  File "C:\Users\ingri\AppData\Local\Programs\Thonny\lib\tkinter\__init__.py", line 839, in callit
    func(*args)
TypeError: Dijkstra.dijkstraAnimate() takes 1 positional argument but 2 were given

```

From the shell outputs, it seems that the updateDijkstraGraph is being entered and exited successfully. However, the issue may be arising at the after() step. For the Prim version, the line looked like this:

```

self.currentStep += 1
self.master.after(750, self.primAnimate)

```

Whereas I have written the following for the Dijkstra's version:

3.3.v

```

self.currentStep += 1
self.master.after(750, self.dijkstraAnimate, self.currentStep)

```

I think that it seems that the self.currentStep is being taken as a parameter when I should not have put it in. Therefore, I changed the line to the following:

3.3.w

```

self.currentStep += 1
self.master.after(750, self.dijkstraAnimate)

```

And when I run the code normally (no pausing), the following is output:

Dijkstra's test

Dijkstra's Algorithm Visualisation

Home

Start vertex: End vertex:

Visualise

In order to input a graph, press the 'Input a graph' button, which will open the window to create the graph. Then choose a start vertex from the list and visualise Prim's algorithm on the graph.

Graph:

Table:

Vertex	Shortest Distance	Previous Vertex	Visited
g	5	e	✓
r	20	g	✓
e	2	a	✓
t	8	a	✓
a	0	-	✓

Shortest path between a and PY_VAR6 is ['a', 'e', 'g']
Weight: 5

Skip Back | Pause | Skip Forward | Use right/left arrow keys to step forward/back through the visualisation when paused.

```
in start graph input window
in hide dijkstra widgets
in dijkstra extra widgets
in validate
valid
in dijkstra steps
in dijkstra animate
in update dijkstra graph
```

From these outputs, it seems that the normal algorithm visualisation works properly. Now when I try pausing it, the following is output:

Dijkstra's test

Dijkstra's Algorithm Visualisation

Home

Start vertex: End vertex:

Visualise

In order to input a graph, press the 'Input a graph' button, which will open the window to create the graph. Then choose a start vertex from the list and visualise Prim's algorithm on the graph.

Graph:

Table:

Vertex	Shortest Distance	Previous Vertex	Visited
g	6	f	✓
h	10	g	✓
f	0	-	✓
w	17	g	✓

Shortest path between f and PY_VAR6 is ['f', 'g', 'h']
Weight: 10

Skip Back | Play | Skip Forward | Use right/left arrow keys to step forward/back through the visualisation when paused.

```
in start graph input window
in hide dijkstra widgets
in dijkstra extra widgets
in validate
valid
in dijkstra steps
in dijkstra animate
in update dijkstra graph
```

When I press pause, the Pause button changes to Play but the visualisation just continues. To check this, I put print statements in the togglePlayPause method to check the flow of the program. To following is now output to the shell:

```
in start graph input window
in hide dijkstra widgets
in dijkstra extra widgets
in validate
valid
in dijkstra steps
in dijkstra animate
in update dijkstra graph
in dijkstra animate
in update dijkstra graph
in toggle play pause
in dijkstra animate
in update dijkstra graph
```

For some reason, it seems that the togglePlayPause method is being correctly entered and the program is not continuing to go through the if statement so this is not the issue. I compared my code

to that of the primAnimate method and I realised that the issue was that it doesn't check if the program is paused and that the current step is < len(self.steps) is not occurring. I therefore fixed the code to the following:

3.3.x

```
def dijkstraAnimate(self):
    print("in dijkstra animate")
    if not self.isPaused and self.currentStep == len(self.steps):
        step = self.steps[self.currentStep]
        if step[0] == "shortest path":
            # step[1] contains the shortest path as a list of vertices
            # therefore adding the combos of edges into the list path_edges
            # e.g. if step[1] = [A, B, C] then path_edges = [(A,B), {B,C}]
            pathEdges = [(step[1][i], step[1][i+1]) for i in range(len(step[1])-1)]
            self.updateDijkstraGraph([], checkingEdges=pathEdges)
            self.invalidMessage["text"] = f"Shortest path between {self.startVertexChoice.get()} and {se
                elif not self.isPaused and self.currentStep < len(self.steps):
                    step = self.steps[self.currentStep]
                    visitedNodes, currentNode, checkingEdges, currentDistance = step
                    self.updateDijkstraGraph(visitedNodes, currentNode, checkingEdges)

                    # update table:
                    self.tableData[currentNode]["shortest distance"]["text"] = str(currentDistance)
                    self.tableData[currentNode]["previous vertex"]["text"] = self.previous[currentNode] if self.prev
                    self.tableData[currentNode]["visited"]["text"] = "✓"

                    self.currentStep += 1
                    self.master.after(750, self.dijkstraAnimate)
```

When I now run the code, the following is output:

Dijkstra's test

Dijkstra's Algorithm Visualisation

Home

1. Set the initial distance from the start values for all nodes (0 for the start node and infinity for all other nodes).
2. Find the node with the shortest distance from the start that has not been visited and look at all the unvisited connected nodes.
3. Calculate the distance from the start for each of the unvisited connected nodes.
4. If the distance calculated is less than the current shortest distance from the start, set the previous node for that connected node to be the current node.
5. Then set the current node as visited.
6. Repeat steps 2 to 5 until all nodes are set to visited.
To find the shortest path through backtracking:
7. Start from the goal node.
8. Add the 'previous node' to the start of a list.
9. Repeat step 7 until the start node is reached.

In order to input a graph, press the 'Input a graph' button, which will open the window to create the graph. Then choose a start vertex from the list and visualise Dijkstra's algorithm on the graph.

Vertex	Shortest Distance	Previous Vertex	Visited
y	∞	-	
w	2	s	✓
z	∞	-	
s	0	-	✓
t	∞	-	

Start vertex: Visualise
End vertex:

Skip Back Play Skip Forward Use right/left arrow keys to step forward/back through the visualisation when paused.

```
in start graph input window
in hide dijkstra widgets
in dijkstra extra widgets
in validate
valid
in dijkstra steps
in dijkstra animate
in update dijkstra graph
in dijkstra animate
in update dijkstra graph
in toggle play pause
in dijkstra animate
```

Now the issue with pausing seems to have been resolved. When I press the right arrow, the following is output:

Dijkstra's test

Dijkstra's Algorithm Visualisation

Home

1. Set the initial distance from the start values for all nodes (0 for the start node and infinity for all other nodes).
 2. Find the node with the shortest distance from the start that has not been visited and look at all the unvisited connected nodes.
 3. Calculate the distance from the start for each of the unvisited connected nodes.
 4. If the distance calculated is less than the current shortest distance from the start, set the previous node for that connected node to be the current node.
 5. Then set the current node as visited.
 6. Repeat steps 2 to 5 until all nodes are set to visited.
 To find the shortest path through backtracking:
 7. Start from the goal node.
 8. Add the 'previous node' to the start of a list.
 9. Repeat step 7 until the start node is reached.

In order to input a graph, press the 'Input a graph' button, which will open the window to create the graph.
 Then choose a start vertex from the list and visualise Prim's algorithm on the graph.

Vertex	Shortest Distance	Previous Vertex	Visited
y	∞	-	
w	2	s	✓
z	∞	-	
s	0	-	✓
t	13	s	✓

Start vertex: End vertex:

Visualise

Skip Back Play Skip Forward Use right/left arrow keys to step forward/back through the visualisation when paused.

```
in dijkstra step forward
in update dijkstra graph
```

When I continue pressing the right arrow until the final step is reached, the following is output:

Dijkstra's test

Dijkstra's Algorithm Visualisation

Home

1. Set the initial distance from the start values for all nodes (0 for the start node and infinity for all other nodes).
 2. Find the node with the shortest distance from the start that has not been visited and look at all the unvisited connected nodes.
 3. Calculate the distance from the start for each of the unvisited connected nodes.
 4. If the distance calculated is less than the current shortest distance from the start, set the previous node for that connected node to be the current node.
 5. Then set the current node as visited.
 6. Repeat steps 2 to 5 until all nodes are set to visited.
 To find the shortest path through backtracking:
 7. Start from the goal node.
 8. Add the 'previous node' to the start of a list.
 9. Repeat step 7 until the start node is reached.

In order to input a graph, press the 'Input a graph' button, which will open the window to create the graph.
 Then choose a start vertex from the list and visualise Prim's algorithm on the graph.

Vertex	Shortest Distance	Previous Vertex	Visited
y	∞	-	
w	2	s	✓
z	17	t	✓
s	0	-	✓
t	13	s	✓

Start vertex: End vertex:

Visualise

Skip Back Play Skip Forward Use right/left arrow keys to step forward/back through the visualisation when paused.

```
in dijkstra step forward
in update dijkstra graph
in dijkstra step forward
in dijkstra step forward
Exception in Tkinter callback
Traceback (most recent call last):
  File "C:\Users\ingri\AppData\Local\Programs\Thonny\lib\tkinter\_init_.py", line 1921, in __call__
    return self.func(*args)
  File "C:\Users\ingri\OneDrive\Documents\QM\QM School work\A Level\Computer Science\NEA\menu_code.py", line 34, in <lambda>
    self.master.bind("<Right>", lambda e: self.stepForward())
  File "C:\Users\ingri\OneDrive\Documents\QM\QM School work\A Level\Computer Science\NEA\menu_code.py", line 116, in stepForward
    visitedNodes, currentNode, checkingEdges, currentDistance = self.steps[self.currentStep]
ValueError: not enough values to unpack (expected 4, got 2)
```

From the outputs, it seems that this issue occurs when the final step is reached for the stepping through the visualisation. To fix this, I changed the code in the stepForward method to the following:

3.3.y

```
elif hasattr(self, "updateDijkstraGraph"):
    print("in dijkstra step forward")
    step = self.steps[self.currentStep]
    if step[0] == "shortest path":
        pathEdges = [(step[1][i], step[1][i+1]) for i in range(len(step[1])-1)]
        self.updateDijkstraGraph([], checkingEdges=pathEdges)
        self.invalidMessage["text"] = f"Shortest path between {self.startVertexChoice.get()} and"
    else:
        visitedNodes, currentNode, checkingEdges, currentDistance = self.steps[self.currentStep]
        self.updateDijkstraGraph(visitedNodes, currentNode, checkingEdges)
        # update table:
        self.tableData[currentNode]["shortest distance"]["text"] = str(currentDistance)
        self.tableData[currentNode]["previous vertex"]["text"] = self.previous[currentNode] if se
        self.tableData[currentNode]["visited"]["text"] = "✓"
# extend once implemented other algorithms
```

This adds checks for if step is the final one and if it is, the code is the same as it is for the final step in dijkstraAnimate. In addition I changed the skipForward code to the following for the same reasons:

3.3.z

```
def skipForward(self):
    self.currentStep = len(self.steps) - 1 # gets the last step index
    self.isPaused = True
    if hasattr(self, "updateChart"): # note that methods count as attributes
        self.updateChart(self.steps[self.currentStep])
    elif hasattr(self, "updatePrimGraph"):
        self.updatePrimGraph(self.steps[self.currentStep])
    elif hasattr(self, "updateDijkstraGraph"):
        print("in dijkstra skip forward")
        pathEdges = [(step[1][i], step[1][i+1]) for i in range(len(step[1])-1)]
        self.updateDijkstraGraph([], checkingEdges=pathEdges)
        self.invalidMessage["text"] = f"Shortest path between {self.startVertexChoice.get()} and"
    # extend once implemented other algorithms
```

When I now press the pause button, the following is output:

The screenshot shows a window titled 'Dijkstra's test' with the sub-tittle 'Dijkstra's Algorithm Visualisation'. The window contains the following elements:

- Graph:** A graph with nodes s, e, w, f, r. Node s is the start node (green). Node f is the current node (red). Node r is the end node (blue). Edges and weights: s-e (8), s-f (8), f-w (16), f-r (3), e-w (15).
- Table:** A table showing the shortest distances, previous vertex, and visited status for each node.
- Inputs:** 'Start vertex:' dropdown set to 'f', 'End vertex:' dropdown set to 'w'.
- Buttons:** 'Visualise' button, 'Skip Back' button, 'Play' button, 'Skip Forward' button, and a note: 'Use right/left arrow keys to step forward/back through the visualisation when paused.'
- Text:** Instructions for using the tool and a note about inputting a graph.

Vertex	Shortest Distance	Previous Vertex	Visited
f	0	-	✓
w	∞	-	
e	∞	-	
r	3	f	✓
s	∞	-	

When I press the right arrow key until the final step is reached, the following is output:

Dijkstra's test

Dijkstra's Algorithm Visualisation

Home

- Set the initial distance from the start values for all nodes (0 for the start node and infinity for all other nodes).
- Find the node with the shortest distance from the start that has not been visited and look at all the unvisited connected nodes.
- Calculate the distance from the start for each of the unvisited connected nodes.
- If the distance calculated is less than the current shortest distance from the start, set the previous node for that connected node to be the current node.
- Then set the current node as visited.
- Repeat steps 2 to 5 until all nodes are set to visited.
- To find the shortest path through backtracking:
- Start from the goal node.
- Add the 'previous node' to the start of a list.
- Repeat step 7 until the start node is reached.

In order to input a graph, press the 'Input a graph' button, which will open the window to create the graph. Then choose a start vertex from the list and visualise Dijkstra's algorithm on the graph.

Vertex	Shortest Distance	Previous Vertex	Visited
f	0	-	✓
w	9	r	✓
e	11	s	✓
r	3	f	✓
s	∞	-	

Start vertex: End vertex: Visualise

Shortest path between f and PY_VAR6 is ['f', 'r', 'w'] Weight: 9

Skip Back Play Skip Forward Use right/left arrow keys to step forward/back through the visualisation when paused.

This is all as expected except for the 'PY_VAR6' in the text output, but this can be fixed later. To check that the other buttons work correctly, I press the Skip Back button, which outputs the following:

Dijkstra's test

Dijkstra's Algorithm Visualisation

Home

- Set the initial distance from the start values for all nodes (0 for the start node and infinity for all other nodes).
- Find the node with the shortest distance from the start that has not been visited and look at all the unvisited connected nodes.
- Calculate the distance from the start for each of the unvisited connected nodes.
- If the distance calculated is less than the current shortest distance from the start, set the previous node for that connected node to be the current node.
- Then set the current node as visited.
- Repeat steps 2 to 5 until all nodes are set to visited.
- To find the shortest path through backtracking:
- Start from the goal node.
- Add the 'previous node' to the start of a list.
- Repeat step 7 until the start node is reached.

In order to input a graph, press the 'Input a graph' button, which will open the window to create the graph. Then choose a start vertex from the list and visualise Dijkstra's algorithm on the graph.

Vertex	Shortest Distance	Previous Vertex	Visited
f	0	-	✓
w	9	r	✓
e	11	s	✓
r	3	f	✓
s	∞	-	

Start vertex: End vertex: Visualise

Skip Back Play Skip Forward Use right/left arrow keys to step forward/back through the visualisation when paused.

This is as expected. Now when I press the Skip Forward button, the following is output:

```
Exception in Tkinter callback
Traceback (most recent call last):
  File "C:\Users\ingri\AppData\Local\Programs\Thonny\lib\tkinter\_init__.py", line 1921, in __call__
    return self.func(*args)
  File "C:\Users\ingri\OneDrive\Documents\QM\QM School work\A Level\Computer Science\NEA\menu_code.py", line 59, in skipForward
rd
    pathEdges = [(step[1][i], step[1][i+1]) for i in range(len(step[1])-1)]
NameError: name 'step' is not defined
```

This is an easy fix as I added this line:

3.3.A

```

elif hasattr(self, "updateDijkstraGraph"):
    print("in dijkstra skip forward")
    step = self.steps[self.currentStep]
    pathEdges = [(step[1][i], step[1][i+1]) for i in range(len(step[1])) - 1)]

```

When I now run the code, pause it and press the Skip Forward button, the following is output:

Vertex	Shortest Distance	Previous Vertex	Visited
y	0	-	✓
h	∞	-	
j	∞	-	
x	4	y	✓
w	∞	-	

Start vertex: End vertex: Visualise Shortest path between y and PY_VAR6 is [y, x, h] Weight: 11

Skip Back Play Skip Forward Use right/left arrow keys to step forward/back through the visualisation when paused.

This is as expected. When I now press the step back button, the following is output:

Vertex	Shortest Distance	Previous Vertex	Visited
y	0	-	✓
h	∞	-	
j	∞	-	
x	4	y	✓
w	15	j	✓

Start vertex: End vertex: Visualise Shortest path between y and PY_VAR6 is [y, x, h] Weight: 11

Skip Back Play Skip Forward Use right/left arrow keys to step forward/back through the visualisation when paused.

And when I keep pressing it until the first step is reached, the following is output:

Dijkstra's test

Dijkstra's Algorithm Visualisation

Home

- Set the initial distance from the start values for all nodes (0 for the start node and infinity for all other nodes).
- Find the node with the shortest distance from the start that has not been visited and look at all the unvisited connected nodes.
- Calculate the distance from the start for each of the unvisited connected nodes.
- If the distance calculated is less than the current shortest distance from the start, set the previous node for that connected node to be the current node.
- Then set the current node as visited.
- Repeat steps 2 to 5 until all nodes are set to visited.
- To find the shortest path through backtracking:
- Start from the goal node.
- Add the 'previous node' to the start of a list.
- Repeat step 7 until the start node is reached.

In order to input a graph, press the 'Input a graph' button, which will open the window to create the graph. Then choose a start vertex from the list and visualise Dijkstra's algorithm on the graph.

Vertex	Shortest Distance	Previous Vertex	Visited
y	0	-	✓
h	11	x	✓
j	14	y	✓
x	4	y	✓
w	15	j	✓

Start vertex: End vertex: Visualise

Shortest path between y and h is [y, x, h]
Weight: 11

Skip Back Play Skip Forward Use right/left arrow keys to step forward/back through the visualisation when paused.

It seems that the graph is being updated accordingly but the vertex table is not being updated. After some thought about what is causing this issue, I realised that I was not updating the table for every vertex in the graph. This means that the state of the table does not revert back to the previous step. To fix this, I wrote the following code:

3.3.B

```
def stepBack(self):
    if self.isPaused and self.currentStep > 0:
        self.currentStep -= 1
        if hasattr(self, "updateChart"):
            numbers = self.steps[self.currentStep]
            indices = self.stepsDictionary[numbers]
            self.updateChart(numbers, indices)
        elif hasattr(self, "updatePrimGraph"):
            mstStep = self.steps[self.currentStep]
            self.updatePrimGraph(mstStep)
            self.visualiseText.insert(tk.END, "MST: " + " ".join([f"{u}-{v}" for u,v in mstStep]) + "\n")
            self.visualiseText.see(tk.END)
        elif hasattr(self, "updateDijkstraGraph"):
            print("In dijkstra step back")
            step = self.steps[self.currentStep]
            if isinstance(step[0], set): # check not shortest path
                visitedNodes, currentNode, checkingEdges, currentDistance = step
                for vertex in visitedNode:
                    previousDist = self.distances[vertex]
                    self.tableData[vertex]["shortest distance"]["text"] = (str(previousDist) if previousDist != float("inf") else "∞")
                    self.tableData[vertex]["previous vertex"]["text"] = (self.previous[vertex] if self.previous[vertex] else "-")
                    self.tableData[vertex]["visited"]["text"] = "✓"
                self.updateDijkstraGraph(visitedNodes, currentNode, checkingEdges)
```

When I run the code and press the arrow key to the first step, the following is output:

Dijkstra's test

Dijkstra's Algorithm Visualisation

Home

- Set the initial distance from the start values for all nodes (0 for the start node and infinity for all other nodes).
- Find the node with the shortest distance from the start that has not been visited and look at all the unvisited connected nodes.
- Calculate the distance from the start for each of the unvisited connected nodes.
- If the distance calculated is less than the current shortest distance from the start, set the previous node for that connected node to be the current node.
- Then set the current node as visited.
- Repeat steps 2 to 5 until all nodes are set to visited.

To find the shortest path through backtracking:

- Start from the goal node.
- Add the 'previous node' to the start of a list.
- Repeat step 7 until the start node is reached.

In order to input a graph, press the 'Input a graph' button, which will open the window to create the graph. Then choose a start vertex from the list and visualise Dijkstra's algorithm on the graph.

Vertex	Shortest Distance	Previous Vertex	Visited
h	31	w	✓
r	44	w	✓
w	26	z	✓
z	0	-	✓

Start vertex: Visualise

End vertex:

Skip Back Play Skip Forward Use right/left arrow keys to step forward/back through the visualisation when paused.

I then realised that the values in previous and distances store only the final values, which is why the vertex table is not changing correctly. Therefore, appending the values currently in distances and previous into steps will likely be the best way to fix the issue. Additionally, it seemed that I was writing the same code to update the vertex table multiple times so I am going to put this code in one method:

3.3.C

```
def updateDijkstraTable(self, step):
    visitedNodes, currentNode, checkingEdges, distances, previous = step
    for vertex in self.graph.nodes():
        # update shortest distance:
        # if the vertex already has a shortest distance, change to that, otherwise infinity
        self.tableData[vertex]["shortest distance"]["text"] = (str(distances[vertex]) if distances[vertex] != float("inf") else "∞")
        # update previous vertex
        # if the vertex already has a previous vertex, change to that, otherwise -
        self.tableData[vertex]["previous vertex"]["text"] = (previous[vertex] if previous[vertex] else "-")
        # visited nodes stay marked
        self.tableData[vertex]["visited"]["text"] = ("✓" if vertex in visitedNodes else "")
```

In dijkstraSteps:

3.3.D

```
# store step for animation:
checkingEdges = [(currentNode, neighbour) for neighbour in self.graph.neighbors(currentNode) if neighbour not in visited]
self.steps.append((visited.copy(), currentNode, checkingEdges.copy(), dict(self.distances), dict(self.previous)))
```

In dijkstraAnimate:

3.3.E

```
elif not self.isPaused and self.currentStep < len(self.steps):
    step = self.steps[self.currentStep]
    visitedNodes, currentNode, checkingEdges, *_ = step
    self.updateDijkstraGraph(visitedNodes, currentNode, checkingEdges)
    # update table
    self.updateDijkstraTable(step)
    self.currentStep += 1
    self.master.after(750, self.dijkstraAnimate)
```

In the menu class:

3.3.F

```
def skipBack(self):
    self.currentStep = 0
    self.isPaused = True
    if hasattr(self, "updateChart"): # note that methods count as attributes
        self.updateChart(self.steps[self.currentStep])
    elif hasattr(self, "updatePrimGraph"):
        self.updatePrimGraph(self.steps[self.currentStep])
    elif hasattr(self, "updateDijkstraGraph"):
        print("in dijkstra skip back")
        step = self.steps[self.currentStep]
        visitedNodes, currentNode, checkingEdges = step[0], step[1], step[2]
        self.updateDijkstraGraph(visitedNodes, currentNode, checkingEdges)
        self.updateDijkstraTable(step)
    # extend once implemented other algorithms
    self.invalidMessage["text"] = ""
```

3.3.G

```
def skipForward(self):
    self.currentStep = len(self.steps) - 1 # gets the last step index
    self.isPaused = True
    if hasattr(self, "updateChart"): # note that methods count as attributes
        self.updateChart(self.steps[self.currentStep])
    elif hasattr(self, "updatePrimGraph"):
        self.updatePrimGraph(self.steps[self.currentStep])
    elif hasattr(self, "updateDijkstraGraph"):
        print("in dijkstra skip forward")
        step = self.steps[self.currentStep]
        pathEdges = [(step[1][i], step[1][i+1]) for i in range(len(step[1]) - 1)]
        self.updateDijkstraGraph([], checkingEdges=pathEdges)
        self.invalidMessage["text"] = f"Shortest path between {self.startVertexChoice.get()} and {self.endVertexChoice.get()}"
        step1 = self.steps[self.currentStep-1] # so that the vertex table is actually filled in
        self.updateDijkstraTable(step1)
    # extend once implemented other algorithms
```

3.3.H

```
def stepBack(self):
    if self.isPaused and self.currentStep > 0:
        self.currentStep -= 1
        if hasattr(self, "updateChart"): # note that methods count as attributes
            numbers = self.steps[self.currentStep]
            indices = self.stepsDictionary[numbers]
            self.updateChart(numbers, indices)
        elif hasattr(self, "updatePrimGraph"):
            mstStep = self.steps[self.currentStep]
            self.updatePrimGraph(mstStep)
            self.visualiseText.insert(tk.END, "MST:" + ".join([f'{u}-{v}', for u,v in mstStep]) + "\n")
            self.visualiseText.see(tk.END)
        elif hasattr(self, "updateDijkstraGraph"):
            print("in dijkstra step back")
            step = self.steps[self.currentStep]
            if isinstance(step[0], set): # check not shortest path
                visitedNodes, currentNode, checkingEdges = step[0], step[1], step[2]
                self.updateDijkstraGraph(visitedNodes, currentNode, checkingEdges)
                self.updateDijkstraTable(step)
    # extend once implemented other algorithms
```

3.3.1

```

def stepForward(self):
    if self.isPaused and self.currentStep < len(self.steps)-1:
        self.currentStep += 1
    if hasattr(self, "updateChart"): # note that method count as attributes
        numbers = self.steps[self.currentStep]
        indices = self.stepsDictionary[numbers]
        self.updateChart(numbers, indices)
    elif hasattr(self, "updatePrimGraph"):
        mstStep = self.steps[self.currentStep]
        self.updatePrimGraph(mstStep)
        self.visualiseText.insert(tk.END, "MST: " + ".join([f"\u0337{u}\u0337{v}" for u,v in mstStep]) + "\n")
        self.visualiseText.see(tk.END)
    elif hasattr(self, "updateDijkstraGraph"):
        print("in dijkstra step forward")
        step = self.steps[self.currentStep]
        if step[0] == "shortest path":
            pathEdges = [(step[1][i], step[1][i+1]) for i in range(len(step[1])-1)]
            self.updateDijkstraGraph([], checkingEdges=pathEdges)
            self.invalidMessage["text"] = f"Shortest path between {self.startVertexChoice.get()} and
else:
    visitedNodes, currentNode, checkingEdges = step[0], step[1], step[2]
    self.updateDijkstraGraph(visitedNodes, currentNode, checkingEdges)
    self.updateDijkstraTable(step)
# extend once implemented other algorithms

```

When I run the code, the following is output:

The screenshot shows a window titled "Dijkstra's Algorithm Visualisation". Inside, there is a graph with nodes q, t, w, e, y, x. Node q is red, t is green, w is red, e is blue, y is green, and x is green. Edges and their weights are: q-t (63), q-w (14), w-t (13), w-e (50), w-y (6), e-y (8), e-x (2), and x-y (9). Below the graph is a table of shortest distances:

Vertex	Shortest Distance	Previous Vertex	Visited
e	29	y	✓
w	14	q	✓
t	33	y	
q	0	-	✓
x	50	q	
y	20	w	✓

At the bottom, there are input fields for "Start vertex" (q) and "End vertex" (x), a "Visualise" button, and control buttons for "Skip Back", "Play", and "Skip Forward". A note says: "Use right/left arrow keys to step forward/back through the visualisation when paused."

When Skip Forward is pressed:

Dijkstra's test

Dijkstra's Algorithm Visualisation

Home

- Set the initial distance from the start values for all nodes (0 for the start node and infinity for all other nodes).
- Find the node with the shortest distance from the start that has not been visited and look at all the unvisited connected nodes.
- Calculate the distance from the start for each of the unvisited connected nodes.
- If the distance calculated is less than the current shortest distance from the start, set the previous node for that connected node to be the current node.
- Then set the current node as visited.
- Repeat steps 2 to 5 until all nodes are set to visited.
- To find the shortest path through backtracking:
- Start from the goal node.
- Add the 'previous node' to the start of a list.
- Repeat step 7 until the start node is reached.

In order to input a graph, press the 'Input a graph' button, which will open the window to create the graph. Then choose a start vertex from the list and visualise Dijkstra's algorithm on the graph.

Vertex	Shortest Distance	Previous Vertex	Visited
y	6	r	✓
w	19	r	✓
x	34	w	✓
a	31	w	✓
r	0	-	✓
e	44	w	✓

Start vertex: End vertex: Visualise

Shortest path between r and a is [r, w, a]
Weight: 31

Skip Back Pause Skip Forward Use right/left arrow keys to step forward/back through the visualisation when paused.

When I press skip back, the following is output:

Dijkstra's test

Dijkstra's Algorithm Visualisation

Home

- Set the initial distance from the start values for all nodes (0 for the start node and infinity for all other nodes).
- Find the node with the shortest distance from the start that has not been visited and look at all the unvisited connected nodes.
- Calculate the distance from the start for each of the unvisited connected nodes.
- If the distance calculated is less than the current shortest distance from the start, set the previous node for that connected node to be the current node.
- Then set the current node as visited.
- Repeat steps 2 to 5 until all nodes are set to visited.
- To find the shortest path through backtracking:
- Start from the goal node.
- Add the 'previous node' to the start of a list.
- Repeat step 7 until the start node is reached.

In order to input a graph, press the 'Input a graph' button, which will open the window to create the graph. Then choose a start vertex from the list and visualise Dijkstra's algorithm on the graph.

Vertex	Shortest Distance	Previous Vertex	Visited
y	∞	-	
w	∞	-	
x	∞	-	
a	∞	-	
r	0	-	✓
e	∞	-	

Start vertex: End vertex: Visualise

Skip Back Pause Skip Forward Use right/left arrow keys to step forward/back through the visualisation when paused.

All functionality seems to be working correctly. Now, I need to check what would happen if I entered a graph that is not fully connected:

The screenshot shows a window titled "Dijkstra's test" with the sub-tittle "Dijkstra's Algorithm Visualisation". The window contains the following elements:

- Instructions:**
 - Set the initial distance from the start values for all nodes (0 for the start node and infinity for all other nodes).
 - Find the node with the shortest distance from the start that has not been visited and look at all the unvisited connected nodes.
 - Calculate the distance from the start for each of the unvisited connected nodes.
 - If the distance calculated is less than the current shortest distance from the start, set the previous node for that connected node to be the current node.
 - Then set the current node as visited.
 - Repeat steps 2 to 5 until all nodes are set to visited.
 - To find the shortest path through backtracking:
 - Start from the goal node.
 - Add the 'previous node' to the start of a list.
 - Repeat step 7 until the start node is reached.

In order to input a graph, press the 'Input a graph' button, which will open the window to create the graph. Then choose a start vertex from the list and visualise Prim's algorithm on the graph.
- Graph:** A directed graph with nodes n, v, b, r, w, h. Edges and weights: n to v (1), n to w (18), v to b (9), r to h (13), w to h (2).
- Table:** A table showing the state of vertices during the algorithm execution.

Vertex	Shortest Distance	Previous Vertex	Visited
h	0	-	✓
r	6	h	✓
w	2	h	✓
v	∞	-	
b	∞	-	
n	∞	-	

- Controls:**
 - Start vertex:
 - End vertex:
 - Visualise
 - Shortest path between h and PY_VAR6 is ['b']
Weight: inf
 - Skip Back
 - Pause
 - Skip Forward
 - Use right/left arrow keys to step forward/back through the visualisation when paused.

I wanted the shortest path between h and b but this caused an output of inf. As I wasn't sure how to fix this, I researched what the NetworkX library could offer for me to check this. I found 2 options:

- `is_connected`²⁸ checks if it is possible to get from every node in the graph to all the others
- `has_path`²⁹ checks if there is a possible way to get from a source node to a target node

I think that it would be better to implement `is_connected` into the graph validation method in the Graph Input class because this would eliminate the need for extra checks and because it ensures that the graph is also connected for Prim's. Therefore, I changed the code to the following:

3.3.J

```
# check connected:
if not nx.is_connected(self.graph):
    self.invalidMessage["text"] = "Invalid graph: Ensure that the graph is fully connected."
    self.validated = False
    return
```

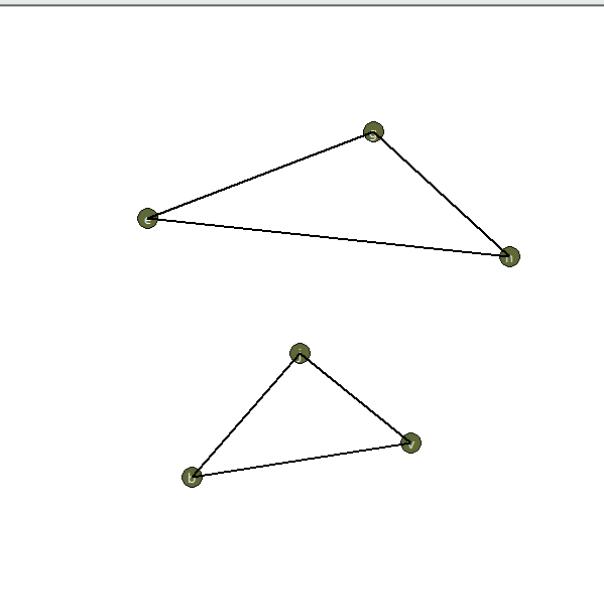
Now when I try to input a graph that is not fully connected, the following is output:

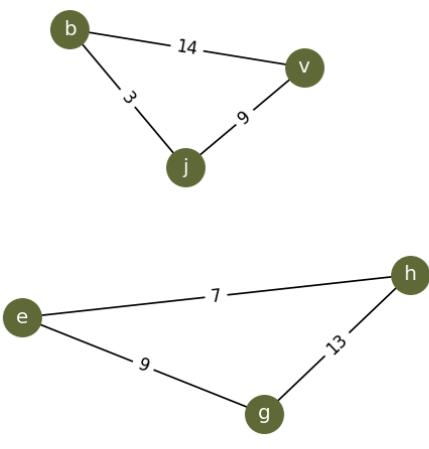
Dijkstra's test

Input a Graph

- Double click on the canvas to add a vertex. Enter a unique identifier.
- Click once on one vertex and once on another vertex to create an edge between them. Enter a weight greater than 0.
- Click on an element once and press delete to remove it.

When you are happy with your graph, check that it has 4-10 vertices, each with at least 1 edge connected to it. Then press submit

Canvas: 

Graph: 

Invalid graph: Ensure that the graph is fully connected.

Submit

This is as expected. Now all functionality for the Dijkstra's Visualisation page is complete.

Now I am going to finish off development for this iteration by refining the GUI for this page. For this, I have the following tasks to improve the GUI:

- Format the title
- Format the text in the menu
- Change the text in the invalidMessage so that the final path is shown clearly and so that the start and end vertices are displayed.
- Ensuring that the text is aligned
- Changing the colours for the highlighting on the visualisation
- Changing the size of the window for both before the graph is entered and after

I made these changes using the following code:

3.3.K

```
def dijkstraWidgets(self):
    self.master.configure(bg="#eaebed")
    self.master.geometry("900x505")

    self.title = Label(self.master, text="Dijkstra's Algorithm Visualisation", font=fontLarge, bg="#eaebed", fg="#1b263b")
    self.title.grid(row=0, column=0, columnspan=14)

    Button(self.master, text="Home", command=self.openHome, bg="#1b263b", fg="white", font=fontMedium, width=15).grid(row=0, column=14, columnspan=2)

    self.dijkstraFrame = Frame(self.master, bg="#eaebed")
    self.dijkstraFrame.grid(row=1, column=0, columnspan=16, rowspan=13)

    Label(self.dijkstraFrame, text=
        "1. Set the initial distance from the start values for all nodes (0 for the start node and infinity for all other nodes).\n"
        "2. Find the node with the shortest distance from the start that has not been visited and look at all the unvisited connected nodes.\n"
        "3. Calculate the distance from the start for each of the unvisited connected nodes. \n"
        "4. If the distance calculated is less than the current shortest distance from the start, set the previous node for that connected\nnode to be the current node. \n"
        "5. Then set the current node as visited.\n"
        "6. Repeat steps 2 to 5 until all nodes are set to visited.\n"
        "To find the shortest path through backtracking:\n"
        "7. Start from the goal node.\n"
        "8. Add the 'previous node' to the start of a list.\n"
        "9. Repeat step 7 until the start node is reached.",
        font=fontSmall,
        fg="#1b263b",
        bg="#eaebed",
        justify="left").grid(row=1, column=0, rowspan=3, columnspan=16, sticky="w")

    Label(self.dijkstraFrame,
        text="In order to input a graph, press the 'Input a graph' button, which will open the window to create the graph. Then choose a\nstart vertex from the list and visualise Prim's algorithm on the graph.\n",
        font=fontSmall,
        fg="#1b263b",
        bg="#eaebed",
        justify="left").grid(row=4, column=0, rowspan=2, columnspan=16, sticky="w")
```

3.3.L

```
def dijkstraExtraWidgets(self):
    print("in dijkstra extra widgets")
    self.master.geometry("900x880")

    self.menuWidgets()
    self.menuText["text"] = "Use right/left arrow keys to step forward/back through\nthe visualisation when paused."
    # destroy Graph Input button
    self.graphInputButton.destroy()
```

3.3.M

```
def updateDijkstraGraph(self, visitedNodes, currentNode=None, checkingEdges=[]):
    print("in update dijkstra graph")
    self.ax.clear()
    pos = nx.get_node_attributes(self.graph, "pos")

    # draw base graph:
    nx.draw(self.graph, pos, with_labels=True, node_color="#606c38", edge_color='black', node_size=500, font_color='white', ax=self.ax)
    edge_labels = nx.get_edge_attributes(self.graph, "weight")
    nx.draw_networkx_edge_labels(self.graph, pos, edge_labels=edge_labels, ax=self.ax)

    # highlighted visited nodes:
    nx.draw_networkx_nodes(self.graph, pos, nodelist=visitedNodes, node_color="#eca400", ax=self.ax)

    # highlight current node:
    if currentNode:
        nx.draw_networkx_nodes(self.graph, pos, nodelist=[currentNode], node_color="#b2675e", ax=self.ax)

    # highlight edges being checked:
    nx.draw_networkx_edges(self.graph, pos, edgelist=checkingEdges, edge_color="#b2675e", width=2, ax=self.ax)

    self.dijkstraGraphCanvas.draw()
```

3.3.N

```
pathEdges = [(step[1][i], step[1][i+1]) for i in range(len(step[1]) -1)]
self.updateDijkstraGraph([], checkingEdges=pathEdges)
shortestPathText = " -> ".join(step[1])
self.invalidMessage["text"] = f"Shortest path between {self.startVertexChoice.get()} and {self.endVertexChoice.get()} is {shortestPathText} \nWeight: {self.distances[self.endVertexChoice.get()]}"
```

When I run the code, the following are the outputs:

Dijkstra's test

Dijkstra's Algorithm Visualisation

Home

1. Set the initial distance from the start values for all nodes (0 for the start node and infinity for all other nodes).
 2. Find the node with the shortest distance from the start that has not been visited and look at all the unvisited connected nodes.
 3. Calculate the distance from the start for each of the unvisited connected nodes.
 4. If the distance calculated is less than the current shortest distance from the start, set the previous node for that connected node to be the current node.
 5. Then set the current node as visited.
 6. Repeat steps 2 to 5 until all nodes are set to visited.
 To find the shortest path through backtracking:
 7. Start from the goal node.
 8. Add the 'previous node' to the start of a list.
 9. Repeat step 7 until the start node is reached.

In order to input a graph, press the 'Input a graph' button, which will open the window to create the graph. Then choose a start vertex from the list and visualise Dijkstra's algorithm on the graph.

Vertex	Shortest Distance	Previous Vertex	Visited
v	∞	-	
x	∞	-	
q	∞	-	
c	∞	-	
s	∞	-	
a	∞	-	

Start vertex:

End vertex:

Skip Back Skip Forward Use right/left arrow keys to step forward/back through the visualisation when paused.

Dijkstra's test

Dijkstra's Algorithm Visualisation

Home

1. Set the initial distance from the start values for all nodes (0 for the start node and infinity for all other nodes).
 2. Find the node with the shortest distance from the start that has not been visited and look at all the unvisited connected nodes.
 3. Calculate the distance from the start for each of the unvisited connected nodes.
 4. If the distance calculated is less than the current shortest distance from the start, set the previous node for that connected node to be the current node.
 5. Then set the current node as visited.
 6. Repeat steps 2 to 5 until all nodes are set to visited.
 To find the shortest path through backtracking:
 7. Start from the goal node.
 8. Add the 'previous node' to the start of a list.
 9. Repeat step 7 until the start node is reached.

In order to input a graph, press the 'Input a graph' button, which will open the window to create the graph. Then choose a start vertex from the list and visualise Dijkstra's algorithm on the graph.

Start vertex:

End vertex:

Skip Back Skip Forward Use right/left arrow keys to step forward/back through the visualisation when paused.

During execution and when the shortest path have been found:

Dijkstra's Algorithm Visualisation

Home

- Set the initial distance from the start values for all nodes (0 for the start node and infinity for all other nodes).
- Find the node with the shortest distance from the start that has not been visited and look at all the unvisited connected nodes.
- Calculate the distance from the start for each of the unvisited connected nodes.
- If the distance calculated is less than the current shortest distance from the start, set the previous node for that connected node to be the current node.
- Then set the current node as visited.
- Repeat steps 2 to 5 until all nodes are set to visited.

To find the shortest path through backtracking:

- Start from the goal node.
- Add the 'previous node' to the start of a list.
- Repeat step 7 until the start node is reached.

In order to input a graph, press the 'Input a graph' button, which will open the window to create the graph. Then choose a start vertex from the list and visualise Dijkstra's algorithm on the graph.

Vertex	Shortest Distance	Previous Vertex	Visited
v	28	x	
x	14	c	✓
q	∞	-	
c	0	-	✓
s	26	x	
a	18	x	✓

Start vertex: Visualise

End vertex:

Skip Back Play Skip Forward Use right/left arrow keys to step forward/back through the visualisation when paused.

Dijkstra's Algorithm Visualisation

Home

- Set the initial distance from the start values for all nodes (0 for the start node and infinity for all other nodes).
- Find the node with the shortest distance from the start that has not been visited and look at all the unvisited connected nodes.
- Calculate the distance from the start for each of the unvisited connected nodes.
- If the distance calculated is less than the current shortest distance from the start, set the previous node for that connected node to be the current node.
- Then set the current node as visited.
- Repeat steps 2 to 5 until all nodes are set to visited.

To find the shortest path through backtracking:

- Start from the goal node.
- Add the 'previous node' to the start of a list.
- Repeat step 7 until the start node is reached.

In order to input a graph, press the 'Input a graph' button, which will open the window to create the graph. Then choose a start vertex from the list and visualise Dijkstra's algorithm on the graph.

Vertex	Shortest Distance	Previous Vertex	Visited
v	6	s	✓
w	9	s	✓
s	0	-	✓
q	13	w	✓

Start vertex: Visualise

End vertex:

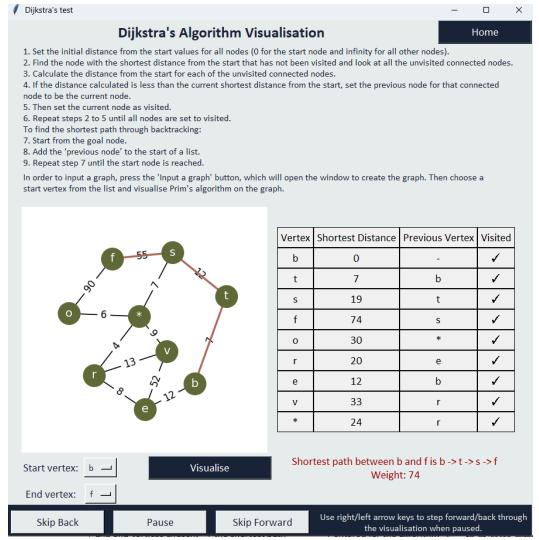
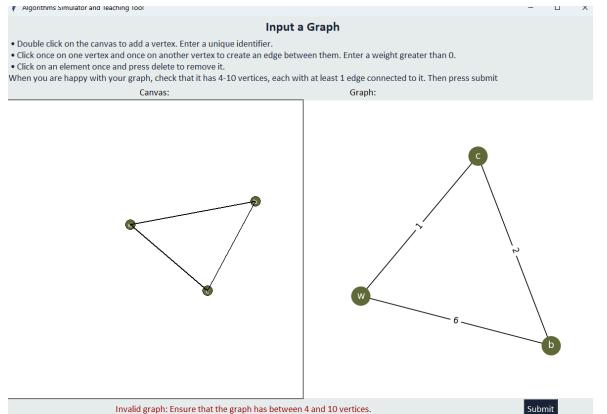
Shortest path between s and q is s -> w -> q
Weight: 13

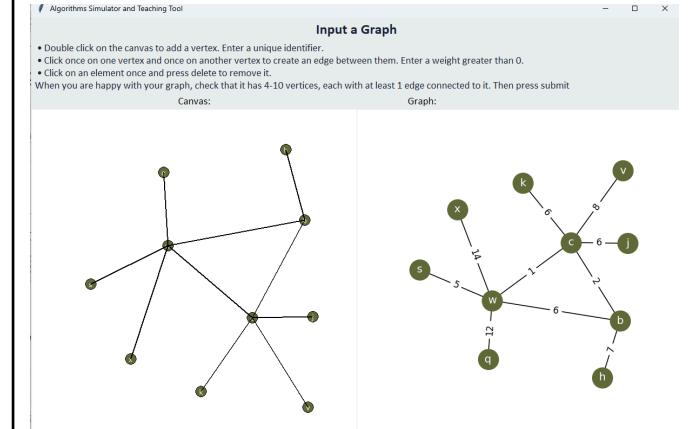
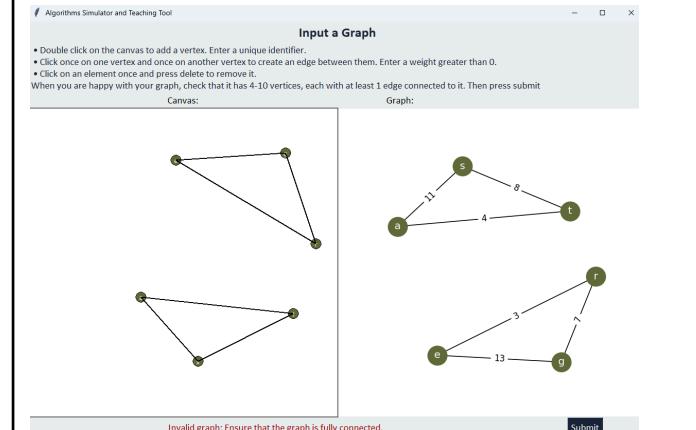
Skip Back Pause Skip Forward Use right/left arrow keys to step forward/back through the visualisation when paused.

Development of the functionality and refining the GUI is complete. Now, I will test the code developed using the white box testing table specified in the DIJKSTRA'S VISUALISATION PAGE design section and the white box testing table specified in the MENU section that is related to the Dijkstra's visualisation.

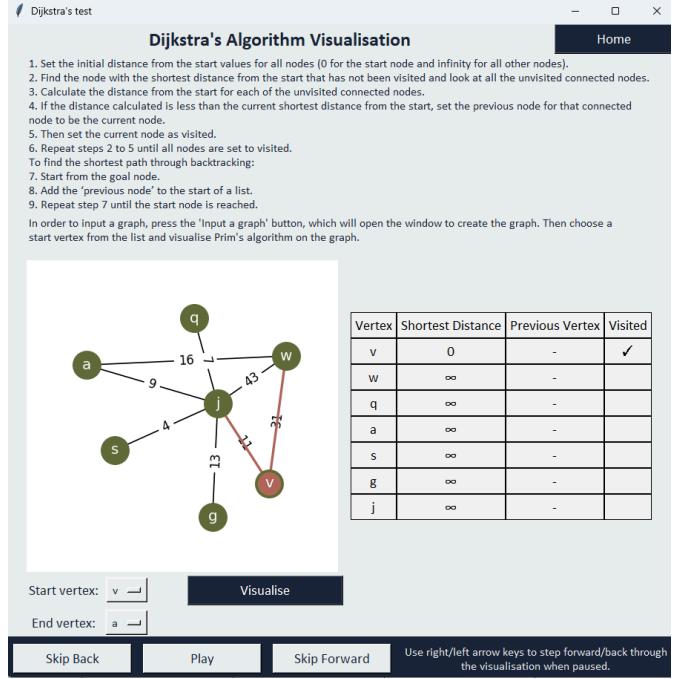
TEST DATA	EXPECTED RESULT	JUSTIFICATION	CODE	ACTUAL OUTPUT
Press visualise without inputting a graph.	An 'Invalid graph' error message is displayed.	Confirms that a graph must be entered by the user for the visualisation to begin.	3.1.p	<p>The screenshot shows a window titled 'Input a Graph'. It contains instructions: 'Double click on the canvas to add a vertex. Enter a unique identifier.', 'Click once on one vertex and once on another vertex to create an edge between them. Enter a weight greater than 0.', and 'Click on an element once and press delete to remove it.' Below these are two sections: 'Canvas:' and 'Graph:'. The 'Canvas:' section is a blank white square with axes from 0.0 to 1.0. The 'Graph:' section is also blank. At the bottom, there is an error message: 'Invalid graph: Ensure that the graph has between 4 and 10 vertices.' and a 'Submit' button.</p>
Press visualise without entering a start vertex.	An 'Invalid - please enter a start vertex' error message is displayed.	Confirms that the start vertex must be entered for the algorithm to begin visualising.	3.3.m	As the functionality of this system makes the first added vertex appear as the first item in the dropdown list and sets this to the start and end vertices, these tests can be merged into if the start and end vertices are the same:
Press visualise without entering an end vertex.	An 'Invalid - please enter a start vertex' error message is displayed.	Confirms that the end vertex must be entered for the algorithm to begin visualising.	3.3.m	

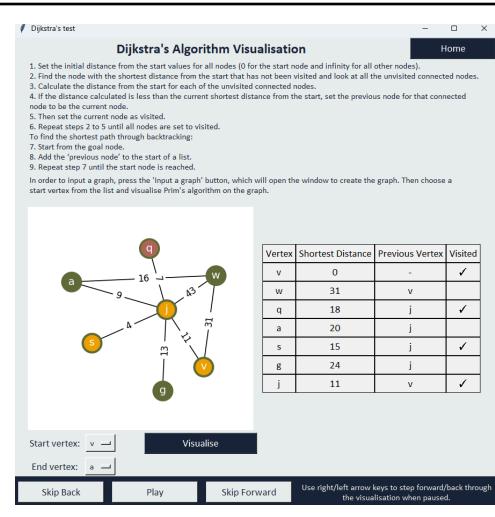
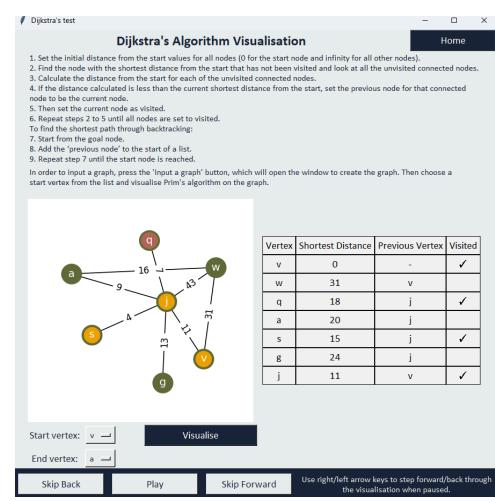
			<p>Dijkstra's Algorithm Visualisation</p> <p>Start vertex: f → End vertex: f →</p> <table border="1"> <thead> <tr> <th>Vertex</th> <th>Shortest Distance</th> <th>Previous Vertex</th> <th>Visited</th> </tr> </thead> <tbody> <tr> <td>f</td> <td>∞</td> <td>-</td> <td></td> </tr> <tr> <td>u</td> <td>∞</td> <td>-</td> <td></td> </tr> <tr> <td>c</td> <td>∞</td> <td>-</td> <td></td> </tr> <tr> <td>k</td> <td>∞</td> <td>-</td> <td></td> </tr> </tbody> </table> <p>Invalid: please enter a start and end vertex that are different.</p>	Vertex	Shortest Distance	Previous Vertex	Visited	f	∞	-		u	∞	-		c	∞	-		k	∞	-		This is as expected
Vertex	Shortest Distance	Previous Vertex	Visited																					
f	∞	-																						
u	∞	-																						
c	∞	-																						
k	∞	-																						
Visualise with a connected graph of 4 vertices and both start and end vertices chosen.	The visualisation of Dijkstra's algorithm begins correctly to find the shortest path between the start and end vertices for the entered graph.	This is a boundary test on the minimum number of vertices that could be entered for the algorithm to visualise.	<p>3.3.n 3.3.o 3.3.l 3.3.x 3.3.C 3.3.D 3.3.E 3.3.K 3.3.L 3.3.M 3.3.N</p> <p>Dijkstra's Algorithm Visualisation</p> <p>Start vertex: k → End vertex: u →</p> <table border="1"> <thead> <tr> <th>Vertex</th> <th>Shortest Distance</th> <th>Previous Vertex</th> <th>Visited</th> </tr> </thead> <tbody> <tr> <td>f</td> <td>30</td> <td>u</td> <td>✓</td> </tr> <tr> <td>u</td> <td>15</td> <td>c</td> <td>✓</td> </tr> <tr> <td>c</td> <td>7</td> <td>k</td> <td>✓</td> </tr> <tr> <td>k</td> <td>0</td> <td>-</td> <td>✓</td> </tr> </tbody> </table> <p>Shortest path between k and u is k > c > u Weight: 15</p>	Vertex	Shortest Distance	Previous Vertex	Visited	f	30	u	✓	u	15	c	✓	c	7	k	✓	k	0	-	✓	This is as expected
Vertex	Shortest Distance	Previous Vertex	Visited																					
f	30	u	✓																					
u	15	c	✓																					
c	7	k	✓																					
k	0	-	✓																					

Visualise with a connected graph of 10 vertices and both start and end vertices chosen.	The visualisation of Dijkstra's algorithm begins correctly to find the shortest path between the start and end vertices for the entered graph.	This is a boundary test on the maximum number of vertices that could be entered for the algorithm to visualise.	3.3.n 3.3.o 3.3.l 3.3.x 3.3.C 3.3.D 3.3.E 3.3.K 3.3.L 3.3.M 3.3.N	 <table border="1" data-bbox="1590 457 1837 672"> <thead> <tr> <th>Vertex</th> <th>Shortest Distance</th> <th>Previous Vertex</th> <th>Visited</th> </tr> </thead> <tbody> <tr><td>b</td><td>0</td><td>-</td><td>✓</td></tr> <tr><td>t</td><td>7</td><td>b</td><td>✓</td></tr> <tr><td>s</td><td>19</td><td>t</td><td>✓</td></tr> <tr><td>f</td><td>74</td><td>s</td><td>✓</td></tr> <tr><td>o</td><td>30</td><td>*</td><td>✓</td></tr> <tr><td>r</td><td>20</td><td>e</td><td>✓</td></tr> <tr><td>e</td><td>12</td><td>b</td><td>✓</td></tr> <tr><td>v</td><td>33</td><td>r</td><td>✓</td></tr> <tr><td>*</td><td>24</td><td>r</td><td>✓</td></tr> </tbody> </table> <p>Start vertex: b End vertex: f Shortest path between b and f is b -> t -> s -> f Weight: 74</p> <p>Skip Back Pause Skip Forward Use right/left arrow keys to step forward/back through the visualisation when paused.</p>	Vertex	Shortest Distance	Previous Vertex	Visited	b	0	-	✓	t	7	b	✓	s	19	t	✓	f	74	s	✓	o	30	*	✓	r	20	e	✓	e	12	b	✓	v	33	r	✓	*	24	r	✓	As expected
Vertex	Shortest Distance	Previous Vertex	Visited																																										
b	0	-	✓																																										
t	7	b	✓																																										
s	19	t	✓																																										
f	74	s	✓																																										
o	30	*	✓																																										
r	20	e	✓																																										
e	12	b	✓																																										
v	33	r	✓																																										
*	24	r	✓																																										
Visualise with a connected graph of 3 vertices and both start and end vertices chosen.	An 'Invalid graph' error message is displayed.	This is an erroneous test for if the number of vertices entered is less than the minimum.	3.1.p 3.1.l	 <p>Input a Graph</p> <ul style="list-style-type: none"> Double click on the canvas to add a vertex. Enter a unique identifier. Click once on one vertex and once on another vertex to create an edge between them. Enter a weight greater than 0. Click on an element once and press delete to remove it. <p>When you are happy with your graph, check that it has 4-10 vertices, each with at least 1 edge connected to it. Then press submit</p> <p>Canvas:</p> <p>Graph:</p> <p>Invalid graph: Ensure that the graph has between 4 and 10 vertices.</p> <p>Submit</p>	As expected.																																								

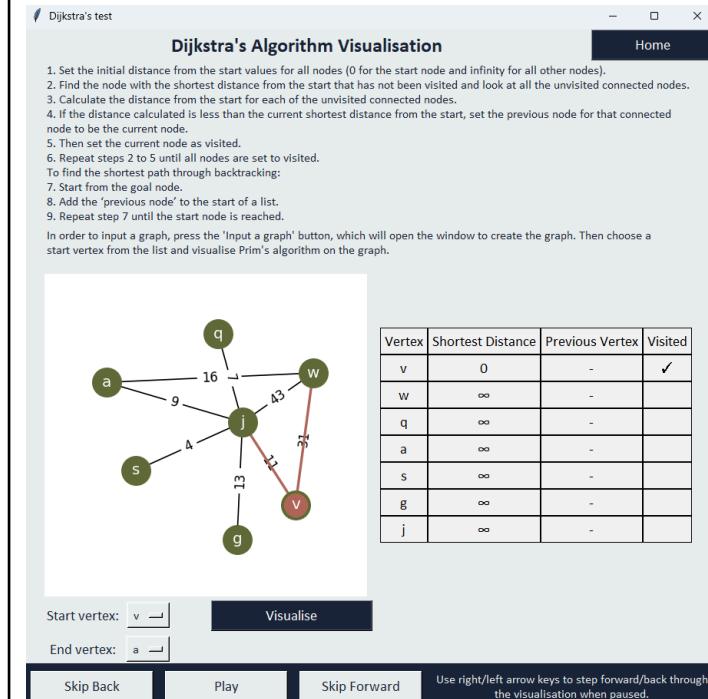
<p>Visualise with a connected graph of 11 vertices and both start and end vertices chosen.</p>	<p>An ‘Invalid graph’ error message is displayed.</p>	<p>This is an erroneous test for if the number of vertices entered is more than the maximum.</p>	<p>3.1.c 3.1.l</p>	 <p>When double click, the pop up to enter a vertex label does not appear. This is as expected.</p>
<p>Visualise with an unconnected graph of 6 vertices and both start and end vertices chosen.</p>	<p>An ‘Invalid graph’ error message is displayed.</p>	<p>This is a test to ensure that all the vertices must have at least 1 edge connected to them, therefore making the visualisation worthwhile.</p>	<p>3.3.J</p>	 <p>As expected.</p>

Press the Home button	The Dijkstra's Visualisation page is closed and the Home page is opened.	This ensures that the 'Home' button works correctly and that the user can go back to the Home page to either log out or go to the other visualisation/quiz pages.	3.3.d 2.1.h 2.1.l 2.1.p 2.1.q 2.1.r 2.1.t	<p>As expected.</p>
Press the 'Skip Forward' button for the Dijkstra's visualisation	The vertex table displays the final values in each of the rows and the graph has the shortest path highlighted in #B2675E.	Confirms that the 'Skip Forward' button works as intended for the Dijkstra's visualisation.	3.3.n 3.3.o 3.3.l 3.3.C 3.3.D 3.3.E 3.3.G 3.3.K 3.3.L 3.3.M 3.3.N 2.3.H 2.3.I	<p>As expected.</p>

Press the 'Skip Back' button for the Dijkstra's visualisation	The graph remains the same but the highlighting on the graph is removed. The vertex table returns to its original blank state.	Confirms that the 'Skip Back' button works as intended for the Dijkstra's visualisation.	3.3.n 3.3.o 3.3.l 3.3.C 3.3.D 3.3.E 3.3.F 3.3.K 3.3.L 3.3.M 3.3.N 2.3.H 2.3.I	 <table border="1" data-bbox="1657 553 1971 764"> <thead> <tr> <th>Vertex</th> <th>Shortest Distance</th> <th>Previous Vertex</th> <th>Visited</th> </tr> </thead> <tbody> <tr><td>v</td><td>0</td><td>-</td><td>✓</td></tr> <tr><td>w</td><td>∞</td><td>-</td><td></td></tr> <tr><td>q</td><td>∞</td><td>-</td><td></td></tr> <tr><td>a</td><td>∞</td><td>-</td><td></td></tr> <tr><td>s</td><td>∞</td><td>-</td><td></td></tr> <tr><td>g</td><td>∞</td><td>-</td><td></td></tr> <tr><td>j</td><td>∞</td><td>-</td><td></td></tr> </tbody> </table> <p>Start vertex: <input type="text" value="v"/> Visualise End vertex: <input type="text" value="a"/> Skip Back Play Skip Forward</p> <p>There is a tick in the row for the first node to be visited however this is inconsequential so this is nothing to be changed.</p>	Vertex	Shortest Distance	Previous Vertex	Visited	v	0	-	✓	w	∞	-		q	∞	-		a	∞	-		s	∞	-		g	∞	-		j	∞	-	
Vertex	Shortest Distance	Previous Vertex	Visited																																	
v	0	-	✓																																	
w	∞	-																																		
q	∞	-																																		
a	∞	-																																		
s	∞	-																																		
g	∞	-																																		
j	∞	-																																		

Press the 'Pause' button for the Dijkstra's visualisation	The visualisation stops, the button changes from 'Pause' to 'Play' (or vice versa).	Confirms that the 'Pause' button works as intended for the Dijkstra's visualisation.	3.3.n 3.3.o 3.3.l 3.3.C 3.3.D 3.3.E 3.3.K 3.3.L 3.3.M 3.3.N 2.3.H 2.3.I	 <table border="1" data-bbox="1560 977 1808 1144"> <thead> <tr> <th>Vertex</th> <th>Shortest Distance</th> <th>Previous Vertex</th> <th>Visited</th> </tr> </thead> <tbody> <tr><td>v</td><td>0</td><td>-</td><td>✓</td></tr> <tr><td>w</td><td>31</td><td>v</td><td></td></tr> <tr><td>q</td><td>18</td><td>j</td><td>✓</td></tr> <tr><td>a</td><td>20</td><td>j</td><td></td></tr> <tr><td>s</td><td>15</td><td>j</td><td>✓</td></tr> <tr><td>g</td><td>24</td><td>j</td><td></td></tr> <tr><td>y</td><td>11</td><td>v</td><td>✓</td></tr> <tr><td>j</td><td>11</td><td>v</td><td>✓</td></tr> </tbody> </table> <p>Start vertex: <input type="text" value="v"/> End vertex: <input type="text" value="a"/> Visualise Skip Back Play Skip Forward Use right/left arrow keys to step forward/back through the visualisation when paused.</p>	Vertex	Shortest Distance	Previous Vertex	Visited	v	0	-	✓	w	31	v		q	18	j	✓	a	20	j		s	15	j	✓	g	24	j		y	11	v	✓	j	11	v	✓	As expected.
Vertex	Shortest Distance	Previous Vertex	Visited																																						
v	0	-	✓																																						
w	31	v																																							
q	18	j	✓																																						
a	20	j																																							
s	15	j	✓																																						
g	24	j																																							
y	11	v	✓																																						
j	11	v	✓																																						
Use the right and left arrow keys for the Dijkstra's visualisation	The visualisation goes back/forward one step in the visualisation on both the graph and the vertex table.	Confirms that the arrow key functionality works as intended for the Dijkstra's visualisation.	3.3.n 3.3.o 3.3.l 3.3.C 3.3.D 3.3.E 3.3.H 3.3.I 3.3.K 3.3.L 3.3.M 3.3.N 2.3.H 2.3.I	 <table border="1" data-bbox="1560 977 1808 1144"> <thead> <tr> <th>Vertex</th> <th>Shortest Distance</th> <th>Previous Vertex</th> <th>Visited</th> </tr> </thead> <tbody> <tr><td>v</td><td>0</td><td>-</td><td>✓</td></tr> <tr><td>w</td><td>31</td><td>v</td><td></td></tr> <tr><td>q</td><td>18</td><td>j</td><td>✓</td></tr> <tr><td>a</td><td>20</td><td>j</td><td></td></tr> <tr><td>s</td><td>15</td><td>j</td><td>✓</td></tr> <tr><td>g</td><td>24</td><td>j</td><td></td></tr> <tr><td>y</td><td>11</td><td>v</td><td>✓</td></tr> <tr><td>j</td><td>11</td><td>v</td><td>✓</td></tr> </tbody> </table> <p>Start vertex: <input type="text" value="v"/> End vertex: <input type="text" value="a"/> Visualise Skip Back Play Skip Forward Use right/left arrow keys to step forward/back through the visualisation when paused.</p>	Vertex	Shortest Distance	Previous Vertex	Visited	v	0	-	✓	w	31	v		q	18	j	✓	a	20	j		s	15	j	✓	g	24	j		y	11	v	✓	j	11	v	✓	Pressing the right arrow is shown above. Pressing the left
Vertex	Shortest Distance	Previous Vertex	Visited																																						
v	0	-	✓																																						
w	31	v																																							
q	18	j	✓																																						
a	20	j																																							
s	15	j	✓																																						
g	24	j																																							
y	11	v	✓																																						
j	11	v	✓																																						

arrow is shown below:



The screenshot shows a window titled "Dijkstra's Algorithm Visualisation". Inside, a graph is displayed with nodes labeled a, w, q, j, s, v, g, f. Edges and their weights are: a to w (16), a to q (9), a to s (8), w to j (3), q to j (3), s to j (13), j to v (2), j to g (1), v to f (2), f to g (1). Node j is highlighted in green. A red arrow points from node j to node v. Below the graph, there is a table of shortest distances and a "Visualise" button.

Vertex	Shortest Distance	Previous Vertex	Visited
v	0	-	✓
w	∞	-	
q	∞	-	
a	∞	-	
s	∞	-	
g	∞	-	
j	∞	-	

Start vertex: End vertex: Visualise

Skip Back Play Skip Forward Use right/left arrow keys to step forward/back through the visualisation when paused.

As expected

All white box tests have passed successfully so development and testing in Prototype 3 Iteration 3 is complete.

ITERATION 3 - DEVELOPER REVIEW

Development went very successfully through this Iteration thanks to the structure developed from the previous sections. However, issues were encountered regarding the menu and how the vertex table can be updated alongside the graph. These issues were solved through reworking my existing methods so that the values in the vertex table, the values needed for updating the graph and the values stored in the steps can be joined correctly.

PROTOTYPE 3 - STAKEHOLDER REVIEW

Now that Prototype 3 has been fully developed and tested, I had a stakeholder review the progress. I asked about the Usability - how user friendly the system is - the Functionality - does it function as intended? - and Performance - how fast the system performs.

See below the feedback given by one of my Stakeholders:

AREA OF CONCERN	FEEDBACK
Usability	Everything is very clear and makes sense.
Functionality	When inputting a graph, I tried to use quotation marks “ for a vertex label and this broke the graph input section. Maybe this could be changed. Other than that, the system functions well.
Performance	Everything performed well, especially going between steps with the arrow keys.

From the stakeholder feedback, it seems that the system is user friendly but the only feedback that I will act on is regarding the graph input section. I hadn't considered the different values that could be input for the vertices. As it seemed that the quotations marks broke the system, I will add a check to ensure that they are not entered:

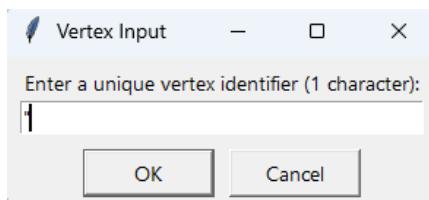
3.3.O

```
if vertex_id and len(vertex_id) == 1 and vertex_id not in self.graph.nodes and vertex_id != '':
    x, y = event.x, event.y
    self.graph.add_node(vertex_id, pos=(x, y))

    self.canvas.create_oval(x-10, y-10, x+10, y+10, fill='blue', outline='black')
    self.canvas.create_text(x, y, text=vertex_id, fill='white')

    #print(f"Added vertex: {vertex_id} at position ({x}, {y})")
    self.update_visualization()
```

To check that this works correctly, I went to the graph input section and tried to enter a “ for a vertex:



After pressing OK, the following is output:

Dijkstra's test

Input a Graph

- Double click on the canvas to add a vertex. Enter a unique identifier.
- Click once on one vertex and once on another vertex to create an edge between them. Enter a weight greater than 0.
- Click on an element once and press delete to remove it.

When you are happy with your graph, check that it has 4-10 vertices, each with at least 1 edge connected to it. Then press submit

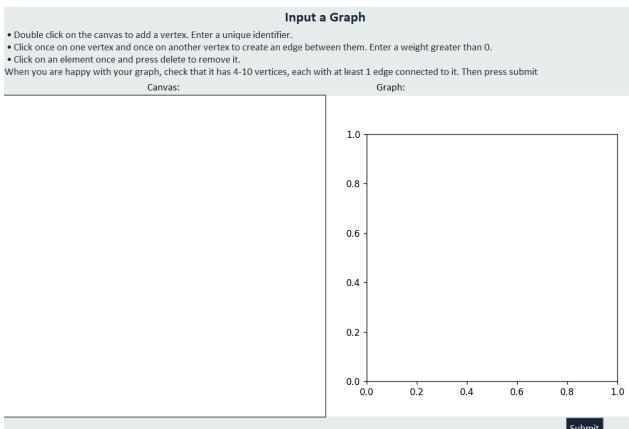
Canvas: Graph:

Submit

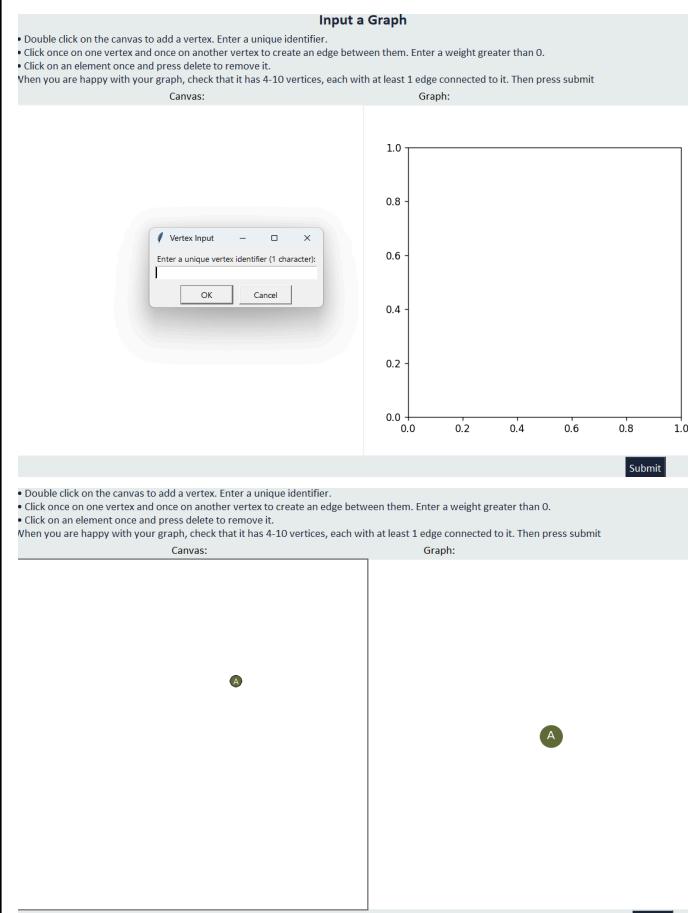
The vertex has not been added to the canvas nor the graph so this functions correctly and the stakeholder feedback has been used to improve the system.

PROTOTYPE 3 - SUCCESS CRITERIA REVIEW

In order to check that all the success criteria outlined in the ANALYSIS section for the Graph Input, Prim's and Dijkstra's Visualisation pages have been met, below is a success criteria review for this prototype:

SUCCESS CRITERIA	OUTPUT & EVIDENCE	CODE	COMPLETE
[10][a] A graph input space will be on the right side of the screen for Prim's and Dijkstra's algorithms. The following functionality will only work when they occur with the mouse in this space. To test this, I will try to perform the different functions both in and out of the space and check that the correct output appears on the screen, along with the programmed version of the graph being correct.	 <p>Input a Graph</p> <ul style="list-style-type: none"> Double click on the canvas to add a vertex. Enter a unique identifier. Click once on one vertex and once on another vertex to create an edge between them. Enter a weight greater than 0. Click on an element once and press delete to remove it. <p>When you are happy with your graph, check that it has 4-10 vertices, each with at least 1 edge connected to it. Then press submit</p> <p>Canvas: Graph: Submit</p> <p>Due to limitations on how I could use the NetworkX library, I had to make use of a Tkinter canvas for the user to do their mouse action on. Therefore, I decided to change this to be in a new window so that the GUI on the Prim's and Dijkstra's pages is clear and organised, whilst still allowing the user to input the graph in a clear and intuitive space. Therefore, on the Prim's and Dijkstra's pages, the following button is displayed:</p> <p>Input a graph</p>	3.1.l 3.1.m 3.1.n	Partially met due to changes in approach

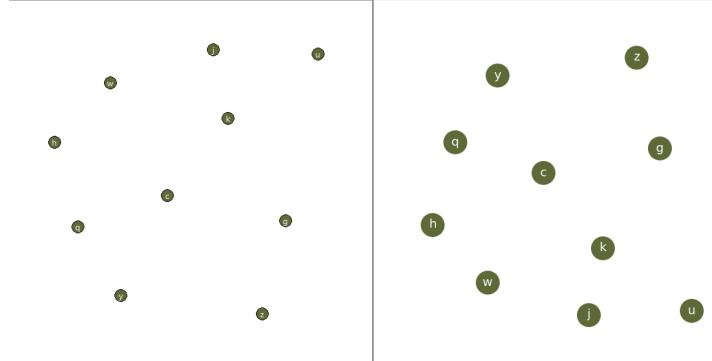
[10][b] When the user double clicks in the graph input space, a vertex will be added. This vertex will have a text box that the user must enter a unique letter into. If a letter is entered that has already been used, the outline of the vertex will turn red. Once a unique letter has been entered, the user can click off the vertex or drag the vertex around the graph input space - connected edges will stay connected to the vertex. To test these functionalities, I will try to double click to create vertices in different parts of the graph input space, try entering different characters (with some being more than 1 letter, some being non-unique characters and some being valid characters) and try dragging vertices around the graph input space. With all these tests, I will check that the function is as intended.



When I try to drag a vertex, it doesn't move because this functionality has not been implemented due to the limitations of the NetworkX library and the time constraints on development.

3.1.h
3.1.i
3.1.c
3.1.i
3.1.j

Partially met due to changes in approach

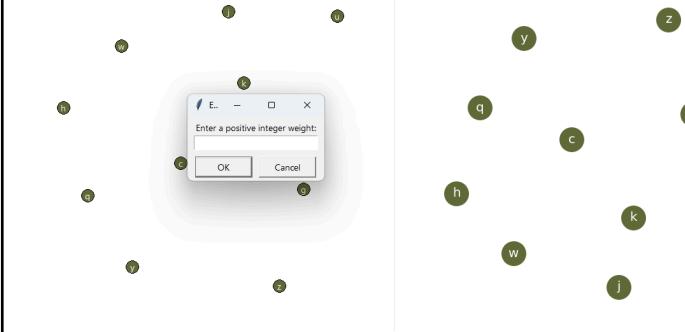
<p>[10][c] When 10 vertices have been added in the graph input space, double-clicking will no longer add any more vertices. I have limited graphs to this size due to the time complexity of the different algorithms. Additionally, the graph must have at least 4 vertices, which will be checked when the 'Start Visualisation' button is pressed for either visualisation. To test these, I will try creating graphs of different sizes and check whether the corresponding functionality and outputs are correct.</p>	<p>For trying to submit a graph of less than 4 vertices:</p> <p>When you are happy with your graph, check that it has 4-10 vertices, each with at least 1 edge connected to it. Then press submit</p> <p>Canvas: Graph:</p>  <p>Invalid graph: Ensure that the graph has between 4 and 10 vertices.</p> <p>For trying to add an 11th vertex to the graph:</p> <p>Click on an element once and press delete to remove it.</p> <p>When you are happy with your graph, check that it has 4-10 vertices, each with at least 1 edge connected to it. Then press submit</p> <p>Canvas: Graph:</p>  <p>Double clicking does not cause the pop up to enter the vertex id to display, enforcing that the user can't submit a graph of more than 10 vertices.</p>	<p>3.1.p 3.1.i 3.3.J</p>	<p>Y</p>
--	--	----------------------------------	----------

[10][d] To add an edge, the user will need to hold shift on their keyboard and drag between two vertices. When this has been done, a text box will appear, connected to the edge, that will make the user input a weight for the edge. Validation will occur for a number being entered for each weight, with the edge turning red if an invalid entry has been made. To test this, I will try to connect edges between different vertices, ensuring that they are only added when intended and appear correctly, and that entering different values for the weight (valid numbers and invalid text) will result in the correct output.

Holding shift and drag was changed to clicking on the start and end vertex. When this is done on the above graph between vertices w and k, the following is displayed:

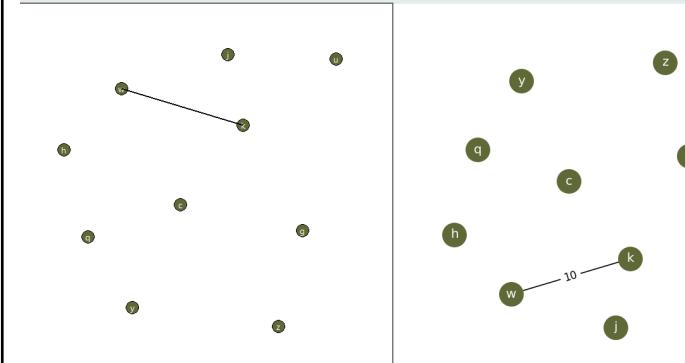
- Double click on the canvas to add a vertex. Enter a unique identifier.
 - Click once on one vertex and once on another vertex to create an edge between them. Enter a weight greater than 0.
 - Click on an element once and press delete to remove it.
- When you are happy with your graph, check that it has 4-10 vertices, each with at least 1 edge connected to it. Then press submit

Canvas: Graph:



- Double click on the canvas to add a vertex. Enter a unique identifier.
 - Click once on one vertex and once on another vertex to create an edge between them. Enter a weight greater than 0.
 - Click on an element once and press delete to remove it.
- When you are happy with your graph, check that it has 4-10 vertices, each with at least 1 edge connected to it. Then press submit

Canvas: Graph:

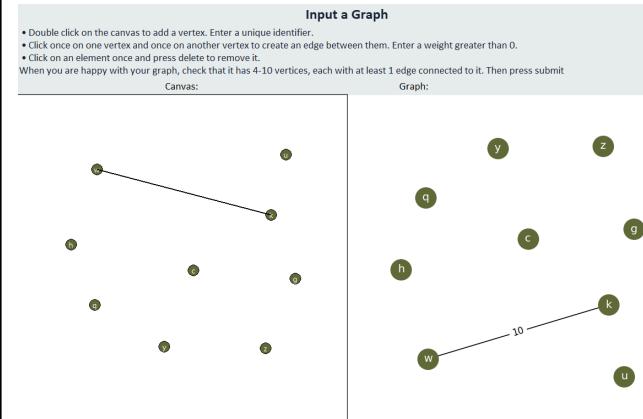


- 3.1.h
3.1.i
3.1.d
3.1.i
3.1.j

Changes were made in approach but edges are able to be added

[10][e] To delete any element (vertex or edge), the user will need to click on the element, which should appear on the screen as highlighted, and press delete on their keyboard. To test this, I will generate different graphs and try to delete different edges and vertices, ensuring that this occurs correctly and that it is updated in the graph in the code correctly.

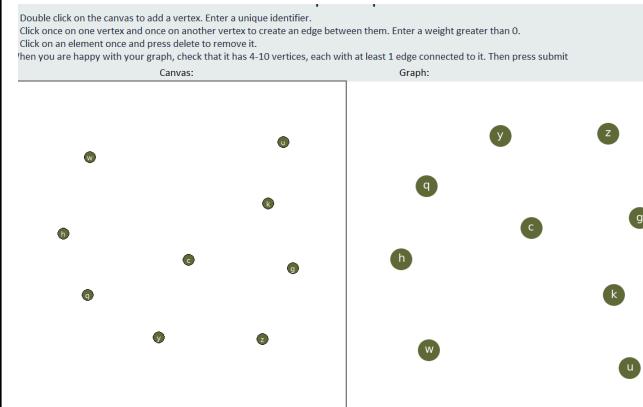
For trying to delete vertex c:



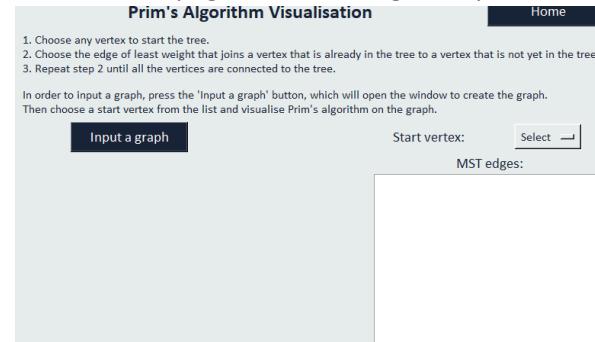
3.1.h
3.1.i
3.1.d
3.1.e
3.1.i
3.1.j

Partially met

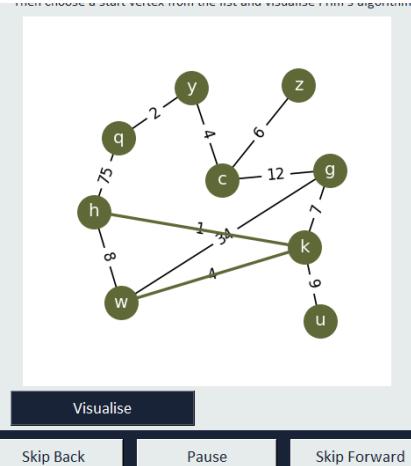
Vertex j was deleted instead. For trying to delete the edge:

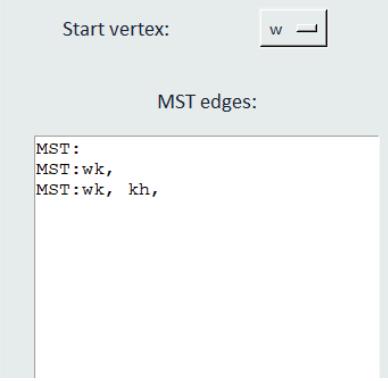
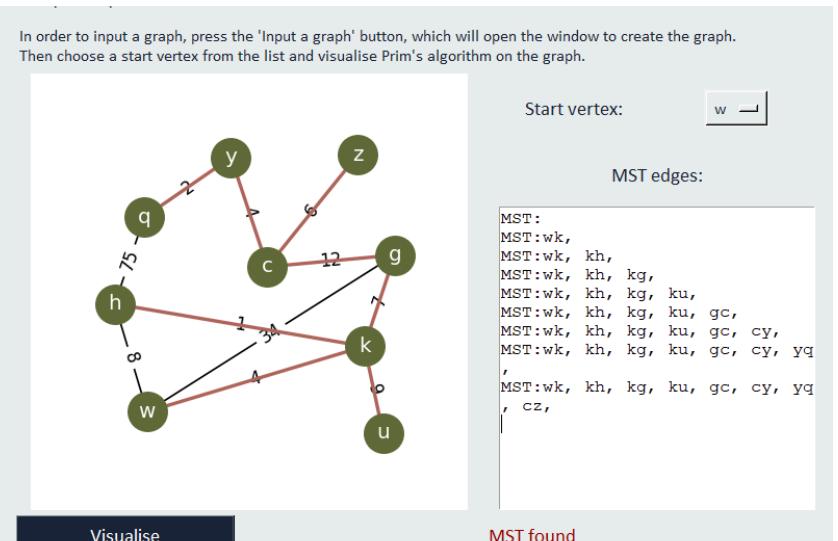


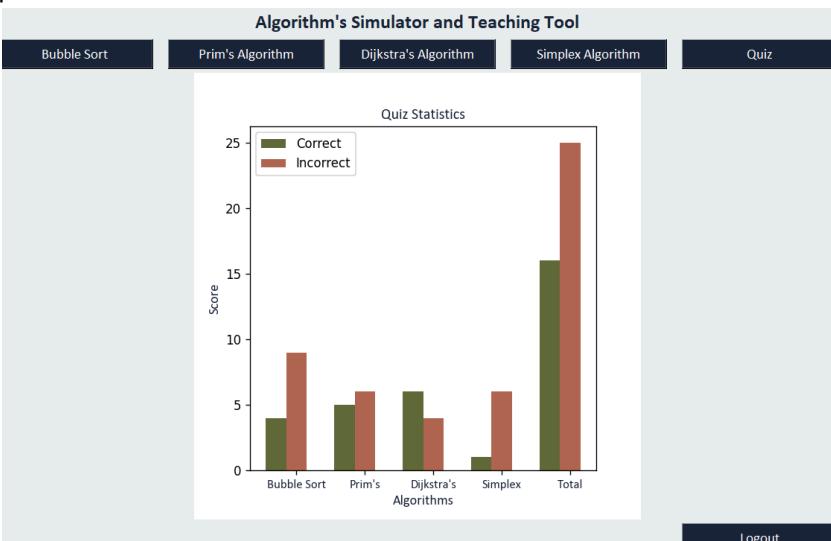
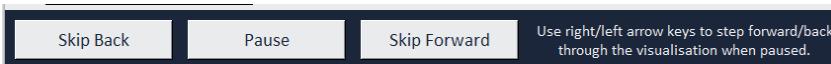
Deleting an edge works correctly. However, there seems to be an issue with deleting a specific vertex. I think that this may be due to how the code determines the position of the mouse's click.

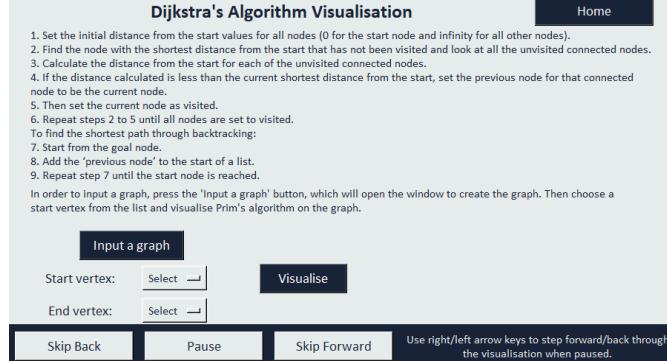
[5][a] There will be text greeting the user with the title of the system 'Prim's Algorithm Visualisation'. I will test this by checking the page and making sure this appears.	<h2>Prim's Algorithm Visualisation</h2>	3.2.b	Y
[5][b] There will be text describing Prim's algorithm in steps in plain English. I will test this by checking the page and making sure this appears.	<p>1. Choose any vertex to start the tree. 2. Choose the edge of least weight that joins a vertex that is already in the tree to a vertex that is not yet in the tree. 3. Repeat step 2 until all the vertices are connected to the tree.</p>	3.2.b	Y
[5][c] Below the text describing Prim's algorithm, there is text describing how to input a graph - see criteria [10]. I will test this by checking the page and making sure this appears.	<p>In order to input a graph, press the 'Input a graph' button, which will open the window to create the graph. Then choose a start vertex from the list and visualise Prim's algorithm on the graph.</p>	3.2.b	Y
[5][d] On the left of the screen, there will be a graph input section - see criteria [10]. To test this, I will check that it is displayed on the correct side of the screen and functions as intended.	<p>For the initial page, the following is output:</p>  <p>When a graph has been input:</p>	3.2.b 3.2.n 3.2.s	Y

	<p>Prim's Algorithm Visualisation</p> <p>Home</p> <p>1. Choose any vertex to start the tree. 2. Choose the edge of least weight that joins a vertex that is already in the tree to a vertex that is not yet in the tree. 3. Repeat step 2 until all the vertices are connected to the tree.</p> <p>In order to input a graph, press the 'Input a graph' button, which will open the window to create the graph. Then choose a start vertex from the list and visualise Prim's algorithm on the graph.</p> <p>Start vertex: <input type="text" value="w"/></p> <p>MST edges:</p> <p>Visualise</p> <p>Skip Back Pause Skip Forward Use right/left arrow keys to step forward/back through the visualisation when paused.</p>		
[5][e] On the right side of the screen, there will be a start vertex drop-down menu to choose a vertex from the input graph. I will test this by checking that it is displayed in the correct place on the screen and that all the vertices in an entered graph are in the drop-down menu and can be selected. I will do this by inputting different valid graphs and ensuring that this is correct for all of them.	<p>Start vertex: <input type="text" value="w"/></p> <p>MST edges:</p>	3.2.q	Y

<p>[5][f] There is a Start Visualisation button below the text describing the graph inputs. Pressing this will initially check if a valid graph has been entered and that a start vertex has been chosen. If this validation is unsuccessful, a message will be displayed saying either 'Graph not valid' or 'Choose start vertex'. If validation is successful, the visualisation of Prim's algorithm will begin. To test this, I will try pressing the Start Visualisation button, having input different valid and invalid graphs, with or without a start vertex entered. This is to check that the correct output is displayed for each case.</p>	 <p>The validation of the graph has been moved to the graph input page so this part of the criteria has been completed elsewhere.</p>	3.2.b	Y
<p>[5][g] As the visualisation of Prim's algorithm is executing, the edges being checked should be highlighted in red. To test this, I will input different graphs and run the visualisation, ensuring that the correct edges are being checked.</p>	 <p>The colour has been changed to green but otherwise this works correctly.</p>	3.2.u 3.2.v 3.2.C 3.2.D 3.2.W	Y

<p>[5][h] As the visualisation of Prim's algorithm is executing, the edges added to the MST and their weights should be displayed as text on the right side of the screen. I will test this by inputting different graphs and running the visualisation, ensuring that the correct edges and values are displayed, which I will work out by hand.</p>		<p>3.2.u 3.2.v 3.2.C 3.2.D 3.2.F</p>	<p>Y</p>
<p>[5][i] When the visualisation is finished, a message will appear saying 'MST found' and the edges in the MST should be highlighted in red on the input graph. To test this, I will enter different graphs and check that running the visualisation completes with this step.</p>	 <p>The colours have again been changed but otherwise this has been met.</p>	<p>3.2.u 3.2.v 3.2.C 3.2.D 3.2.F 3.2.G 3.2.W</p>	<p>Y</p>

<p>[5][j] There is a Home button at the bottom right of the screen that returns the user to the Home page. To test this, I will check that the button appears in the correct place and functions as intended.</p>	 <p>The placement has been changed to the top right of the screen. When pressed:</p>  <table border="1"> <caption>Quiz Statistics</caption> <thead> <tr> <th>Algorithm</th> <th>Correct</th> <th>Incorrect</th> </tr> </thead> <tbody> <tr> <td>Bubble Sort</td> <td>4</td> <td>9</td> </tr> <tr> <td>Prim's</td> <td>5</td> <td>6</td> </tr> <tr> <td>Dijkstra's Algorithms</td> <td>6</td> <td>4</td> </tr> <tr> <td>Simplex</td> <td>1</td> <td>6</td> </tr> <tr> <td>Total</td> <td>16</td> <td>25</td> </tr> </tbody> </table> <p>This is as expected.</p>	Algorithm	Correct	Incorrect	Bubble Sort	4	9	Prim's	5	6	Dijkstra's Algorithms	6	4	Simplex	1	6	Total	16	25	<p>3.2.Q 3.2.U</p>	<p>Y</p>
Algorithm	Correct	Incorrect																			
Bubble Sort	4	9																			
Prim's	5	6																			
Dijkstra's Algorithms	6	4																			
Simplex	1	6																			
Total	16	25																			
<p>[5][k] There is a menu - criteria [9] - that appears at the bottom of the screen. To test this, I will check that it is displayed in the correct place and functions as intended.</p>	 <p>Skip Back Pause Skip Forward Use right/left arrow keys to step forward/back through the visualisation when paused.</p>	<p>2.3.H 2.3.I</p>	<p>Y</p>																		

[6][a] There will be text greeting the user with the title of the system 'Dijkstra's Algorithm Visualisation'. I will test this by checking the page and making sure this appears.	<h2>Dijkstra's Algorithm Visualisation</h2>	3.3.K	Y
[6][b] There will be text describing Dijkstra's algorithm in steps in plain English. I will test this by checking the page and making sure this appears.	<p>1. Set the initial distance from the start values for all nodes (0 for the start node and infinity for all other nodes). 2. Find the node with the shortest distance from the start that has not been visited and look at all the unvisited connected nodes. 3. Calculate the distance from the start for each of the unvisited connected nodes. 4. If the distance calculated is less than the current shortest distance from the start, set the previous node for that connected node to be the current node. 5. Then set the current node as visited. 6. Repeat steps 2 to 5 until all nodes are set to visited. To find the shortest path through backtracking: 7. Start from the goal node. 8. Add the 'previous node' to the start of a list. 9. Repeat step 7 until the start node is reached.</p>	3.3.K	Y
[6][c] Below the text describing Dijkstra's algorithm, there is text describing how to input a graph - see criteria [10]. I will test this by checking the page and making sure this appears.	<p>In order to input a graph, press the 'Input a graph' button, which will open the window to create the graph. Then choose a start vertex from the list and visualise Prim's algorithm on the graph.</p> <p>There is a mistake as the text says to visualise Prim's algorithm. This is a quick fix in the text for the Label.</p>	3.3.K	N
[6][d] On the left of the screen, there will be a graph input section - see criteria [10]. To test this, I will check that it is displayed on the correct side of the screen and functions as intended.	<p>The following is displayed initially:</p>  <p>And when a valid graph has been input:</p>	3.3.K 3.3.f 3.3.L	Y

1. Set the initial distance from the start values for all nodes (0 for the start node and infinity for all other nodes).
2. Find the node with the shortest distance from the start that has not been visited and look at all the unvisited connected nodes.
3. Calculate the distance from the start for each of the unvisited connected nodes.
4. If the distance calculated is less than the current shortest distance from the start, set the previous node for that connected node to be the current node.
5. Then set the current node as visited.
6. Repeat steps 2 to 5 until all nodes are set to visited.

To find the shortest path through backtracking:

7. Start from the goal node.
8. Add the 'previous node' to the start of a list.
9. Repeat step 7 until the start node is reached.

In order to input a graph, press the 'Input a graph' button, which will open the window to create the graph. Then choose a start vertex from the list and visualise Prim's algorithm on the graph.

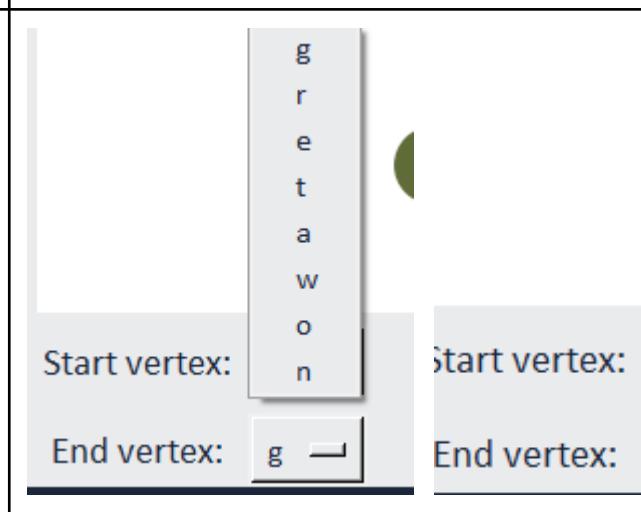
```

graph LR
    n((n)) ---|17| e((e))
    n((n)) ---|9| t((t))
    e((e)) ---|84| r1((r))
    e((e)) ---|3| t((t))
    t((t)) ---|14| g((g))
    t((t)) ---|14| a((a))
    r1((r)) ---|5| r2((r))
    r1((r)) ---|63| w((w))
    g((g)) ---|14| w((w))
    r2((r)) ---|o| o((o))
  
```

Vertex	Shortest Distance	Previous Vertex	Visited
g	∞	-	
r	∞	-	
e	∞	-	
t	∞	-	
a	∞	-	
w	∞	-	
o	∞	-	
n	∞	-	

Start vertex: Visualise

End vertex:

<p>[6][e] On the right side of the screen, there will be a vertex table including columns for vertex, distance from start vertex, visited (T/F) and path. To test this, I will check that when the Dijkstra's visualisation page is accessed, this table is displayed in the correct place.</p>	<table border="1"> <thead> <tr> <th>Vertex</th><th>Shortest Distance</th><th>Previous Vertex</th><th>Visited</th></tr> </thead> <tbody> <tr><td>g</td><td>∞</td><td>-</td><td></td></tr> <tr><td>r</td><td>∞</td><td>-</td><td></td></tr> <tr><td>e</td><td>∞</td><td>-</td><td></td></tr> <tr><td>t</td><td>∞</td><td>-</td><td></td></tr> <tr><td>a</td><td>∞</td><td>-</td><td></td></tr> <tr><td>w</td><td>∞</td><td>-</td><td></td></tr> <tr><td>o</td><td>∞</td><td>-</td><td></td></tr> <tr><td>n</td><td>∞</td><td>-</td><td></td></tr> </tbody> </table>	Vertex	Shortest Distance	Previous Vertex	Visited	g	∞	-		r	∞	-		e	∞	-		t	∞	-		a	∞	-		w	∞	-		o	∞	-		n	∞	-			3.3.i 3.3.k 3.3.l Y
Vertex	Shortest Distance	Previous Vertex	Visited																																				
g	∞	-																																					
r	∞	-																																					
e	∞	-																																					
t	∞	-																																					
a	∞	-																																					
w	∞	-																																					
o	∞	-																																					
n	∞	-																																					
<p>[6][f] Below the table, there will be a start vertex drop-down menu and an end vertex drop-down menu to choose a vertex from the input graph. The vertices chosen should be different. I will test this by checking that it is displayed in the correct place on the screen and that all the vertices in an entered graph are in the drop-down menu and can be selected. I will do this by inputting different valid graphs and ensuring that this is correct for all of them.</p>			3.3.e 3.3.f 3.3.m Y																																				

[6][g] Next to the start vertex drop-down menu, there will be a Start Visualisation button. Pressing this will initially check if a valid graph has been entered and that different start and end vertices have been chosen. If this validation is unsuccessful, a message will be displayed saying either ‘Graph not valid’ or ‘Choose different start and end vertices. If validation is successful, the vertex table is filled in with the vertices and the start vertex’s distance from the start value is set to 0. Then, the visualisation will begin. To test this, I will try pressing the Start Visualisation button, having input different valid and invalid graphs, with or without a start vertex entered. This is to ensure that the vertex table is filled in correctly and to check that the correct output is displayed for each case.

[6][h] As vertices are visited in Dijkstra’s algorithm, they will be highlighted on the graph in red and the cells in the vertex table will be highlighted as they are updated. This is because colour-coding makes it clear to the user why certain values are being updated in the table. To test this, I will work out by hand the order the vertices should be visited and the state of the vertex table at each stage for a number of different valid graphs in order to check that the highlighting is correct at all stages.

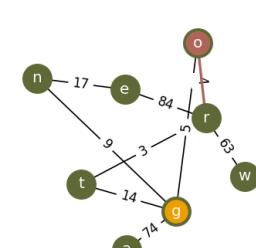
Visualise

The validation of the graph has been moved to the graph input page so this part of the criteria has been completed elsewhere. When the start and end vertices are the same:

Start vertex: <input type="text" value="g"/>	<input type="button" value="Visualise"/>	Invalid: please enter a start and end vertex that are different.
End vertex: <input type="text" value="g"/>		

3.3.e
3.3.f
3.3.K
3.3.m

Y



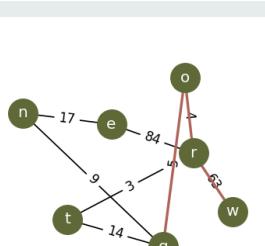
Vertex	Shortest Distance	Previous Vertex	Visited
g	0	-	✓
r	∞	-	
e	∞	-	
t	14	g	
a	74	g	
w	∞	-	
o	5	g	✓
n	9	g	

Start vertex: <input type="text" value="g"/>	<input type="button" value="Visualise"/>		
End vertex: <input type="text" value="w"/>			
<input type="button" value="Skip Back"/>	<input type="button" value="Pause"/>	<input type="button" value="Skip Forward"/>	Use right/left arrow keys to step forward/back through the visualisation when paused.

3.3.C
3.3.D
3.3.E
3.3.M

Y

[6][i] For the backtracking portion of Dijkstra's algorithm, the edges being added to the path should be highlighted on the graph as they are added (i.e. from end vertex to start vertex). To test this, I will run the visualisation on different valid graphs and ensure that the order of backtracking and highlighting is correct and match up.



Vertex	Shortest Distance	Previous Vertex	Visited
g	0	-	✓
r	9	o	✓
e	26	n	✓
t	12	r	✓
a	74	g	✓
w	72	r	✓
o	5	g	✓
n	9	g	✓

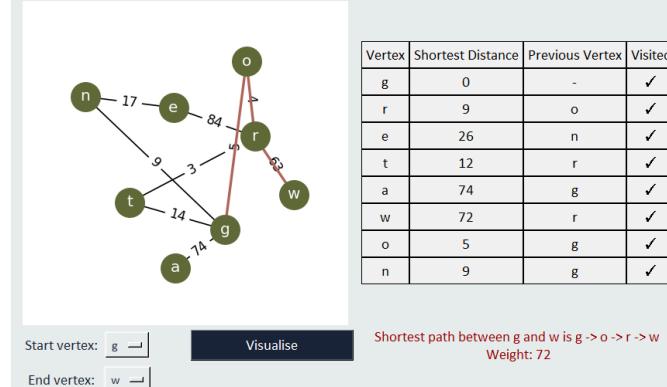
Highlighting is not done in order of backtracking as this complicates the code further.

3.3.N
3.3.M

N

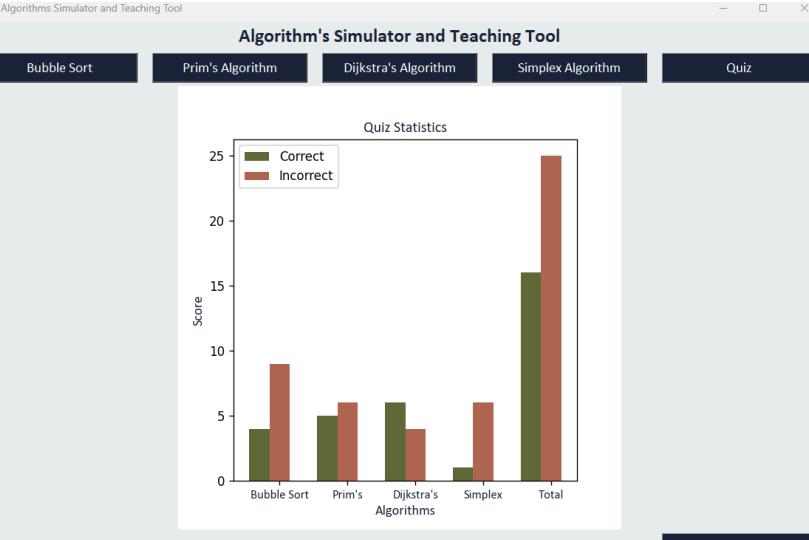
[6][j] When the visualisation is finished, text should appear stating the shortest route from the start to the end vertex and the path should remain highlighted on the graph. To test this, I will run the visualisation on different valid graphs and ensure that the correct shortest route is displayed.

1. Set the initial distance from the start values for all nodes (0 for the start node and infinity for all other nodes).
 2. Find the node with the shortest distance from the start that has not been visited and look at all the unvisited connected nodes.
 3. Calculate the distance from the start for each of the unvisited connected nodes.
 4. If the distance calculated is less than the current shortest distance from the start, set the previous node for that connected node to be the current node.
 5. Then set the current node as visited.
 6. Repeat steps 2 to 5 until all nodes are set to visited.
 To find the shortest path through backtracking:
 7. Start from the goal node.
 8. Add the 'previous node' to the start of a list.
 9. Repeat step 7 until the start node is reached.
 In order to input a graph, press the 'Input a graph' button, which will open the window to create the graph. Then choose a start vertex from the list and visualise Prim's algorithm on the graph.



3.3.q
3.3.K
3.3.N

Y

<p>[6][k] There is a Back button at the bottom right of the screen that returns the user to the Home page. To test this, I will check that the button appears in the correct place and functions as intended.</p>	 <p>When pressed:</p>  <table border="1"> <thead> <tr> <th>Algorithm</th> <th>Correct Score</th> <th>Incorrect Score</th> </tr> </thead> <tbody> <tr> <td>Bubble Sort</td> <td>4</td> <td>9</td> </tr> <tr> <td>Prim's</td> <td>5</td> <td>6</td> </tr> <tr> <td>Dijkstra's</td> <td>6</td> <td>4</td> </tr> <tr> <td>Simplex</td> <td>1</td> <td>6</td> </tr> <tr> <td>Total</td> <td>16</td> <td>25</td> </tr> </tbody> </table>	Algorithm	Correct Score	Incorrect Score	Bubble Sort	4	9	Prim's	5	6	Dijkstra's	6	4	Simplex	1	6	Total	16	25	<p>3.3.K 3.3.d</p>	<p>Y</p>
Algorithm	Correct Score	Incorrect Score																			
Bubble Sort	4	9																			
Prim's	5	6																			
Dijkstra's	6	4																			
Simplex	1	6																			
Total	16	25																			
<p>[6][l] There is a menu - criteria [9] - that appears at the bottom of the screen. To test this, I will check that it is displayed in the correct place and functions as intended.</p>	 <p>Skip Back Pause Skip Forward Use right/left arrow keys to step forward/back through the visualisation when paused.</p>	<p>2.3.H 2.3.I</p>	<p>Y</p>																		

For the error in the text describing how to input a graph on the Dijkstra's visualisation page, I changed it to the following in the Label:

Then choose a start vertex from the list and visualise Dijkstra's algorithm on the graph.\n'

Now the text appears like this, meaning that the issue has been fixed:

In order to input a graph, press the 'Input a graph' button, which will open the window to create the graph. Then choose a start vertex from the list and visualise Dijkstra's algorithm on the graph.

There were many modifications made to how the graph input section would be implemented due to the limitations of Tkinter and NetworkX. This means that the criteria [10] all had changes made, meaning that they were only partially or not met. Additionally, the criteria for highlighting the edges in the order of backtracking for the Dijkstra's visualisation was not met due to the added complexity of the code. Taking the time to develop this would put more pressure on the time constraints for the development of this system so it will become a feature to develop in post-development. The issues this Success Criteria found, however, were regarding the deleting of a vertex. It seemed that the code was not correctly identifying which vertex was being selected, which may be due to the bugs in how Tkinter binds actions to keyboard inputs. I think that this is not a major issue currently so fixing this bug will become a post-development task.

PROTOTYPE 4 - SIMPLEX VISUALISATION AND QUIZ PAGES

In this prototype, I will fully complete the functionality of the Simplex algorithm visualisation page and the Quiz page. For the Quiz page, it is required that all the other algorithms are fully completed, which is why it is the last section to be developed.

ITERATION 1 - SIMPLEX VISUALISATION PAGE FUNCTIONALITY AND GUI

In this iteration, I shall complete the Simplex Visualisation page, which will include the setting up of the GUI and the functionality of the algorithm, which will be specifically implemented to make use of the Menu, which was developed in Prototype 2 Iteration 3.

As I have done with all the other algorithm visualisation pages, I first set up the main structure of the Dijkstra class and the GUI with the following code

4.1.a

```
class Simplex(Menu):
    def __init__(self, master, pUsername):
        Menu.__init__(self, master, 12, 14)
        self.username = pUsername

    def simplexWidgets(self):
        self.master.configure(bg="#eaebed")

    def openHome(self):
        closeOpen(self.master, "home", self.username)
```

4.1.b

```
def validate(self):
    print("in validate")

def updateSimplexGraph(self):
    print("in update simplex graph")

def simplexSteps(self):
    print("in simplex steps")

def simplexAnimate(self):
    print("in simplex animate")
```

4.1.c

```
# imports specific to the Simplex page
from tkinter import *
import tkinter as tk
#from NEA_main_file import closeOpen
from NEA_utilities import closeOpen, fontLarge, fontMedium, fontSmall
from menu_code import Menu
```

4.1.d

```
if __name__ == "__main__":
    root = tk.Tk()
    root.title("Simplex test")
    #root.geometry("800x595")
    graph_input = Simplex(root, "HelloWord!!!") #HelloWord!!! username for testing purposes
    root.mainloop()
```

Then, I focused on developing the GUI for the Simplex page in the simplexWidgets method:

4.1.e

```
def simplexWidgets(self):
    self.master.configure(bg="#eaebed")
    self.title = Label(self.master, text="Simplex Algorithm Visualisation", font=fontLarge, bg="#eaebed", fg="#1b263b")
    self.title.grid(row=0, column=0, columnspan=12) # change columnspan
    Button(self.master, text="Home", command=self.openHome, bg="#1b263b", fg="white", font=fontMedium, width=15).grid(row=0, column=12, columnspan=2)

    self.simplexFrames = Frame(self.master, bg="#eaebed")
    self.simplexFrames.grid(row=1, column=0, columnspan=14, rowspan=11)

    Label(self.simplexFrames, text=
        "1. Turn the inequalities into equalities by adding a slack variable. Rearrange the objective function\n\tto be equal to 0. Input these equations into the simplex tableau.\n"
        "2. Choose the most negative value in the objective row to become the basis.\n"
        "3. Work out the 0-values for each row. Using the smallest (but not negative) value, select the pivot.\n"
        "4. Divide the pivot row by the value of the pivot.\n"
        "5. Add or subtract multiple of the pivot row from the other rows so that the basis variable is 0.\n"
        "6. Repeat until the objective row contains no negative entries.\n"
        "7. Read off the optimal solution from the tableau.",
        font=fontSmall,
        fg="#1b263b",
        bg="#eaebed",
        justify="left").grid(row=1, column=0, rowspan=3, columnspan=14, sticky="w")

    # visualise:
    Button(self.simplexFrames, text="Visualise", command=self.validate, bg="#1b263b", fg="white", font=fontMedium, width=15).grid(row=9, column=10, columnspan=3)
    # invalid:
    self.invalidMessage = Label(self.simplexFrames, text="invalid message", font=fontMedium, bg="#eaebed", fg="#990000")
    self.invalidMessage.grid(row=10, column=8, columnspan=6, rowspan=2)
```

For creating the constraints, as I need to create a method to add the third constraint, in order to reduce code redundancy, I will just call this method twice in the simplexWidgets method, passing in the value for which row the constraint should go on:

4.1.f

```

def addConstraint(self, row):
    print("in add constraint")
    if row == 6:
        self.constraintButton.destroy()
    Label(self.simplexFrme, text="Constraint 3: ", font=fontSmall, bg="#eaebed", fg="#eca400").grid(row=6, column=8)

    xEntry = Entry(self.simplexFrme, width=5)
    xEntry.grid(row=row, column=9)
    Label(self.simplexFrme, text=" x + ", font=fontSmall, bg="#eaebed", fg="#606c38").grid(row=row, column=10)
    yEntry = Entry(self.simplexFrme, width=5)
    yEntry.grid(row=row, column=11)
    Label(self.simplexFrme, text=" y ≤ ", font=fontSmall, bg="#eaebed", fg="#606c38").grid(row=row, column=12)
    rhsEntry = Entry(self.simplexFrme, width=5)
    rhsEntry.grid(row=row, column=13)

```

4.1.g

```

# constraint 1:
Label(self.simplexFrme, text="Constraint 1:", font=fontSmall, bg="#eaebed", fg="#606c38").grid(row=4, column=8)
self.addConstraint(4)

# constraint 2:
Label(self.simplexFrme, text="Constraint 2:", font=fontSmall, bg="#eaebed", fg="#b2675e").grid(row=5, column=8)
self.addConstraint(5)

# constraint 3 button:
self.constraintButton = Button(self.simplexFrme, text="Add constraint", command=lambda: self.addConstraint(6), bg="#1b263b", fg="white", font=fontMedium, width=15)
self.constraintButton.grid(row=6, column=9, columnspan=3)

# objective:
Label(self.simplexFrme, text="Objective function:", font=fontMedium, bg="#eaebed", fg="#1b263b").grid(row=7, column=8, columnspan=4)
Label(self.simplexFrme, text="P =", font=fontSmall, bg="#eaebed", fg="#1b263b").grid(row=8, column=8)
xEntry = Entry(self.simplexFrme, width=5)
xEntry.grid(row=8, column=9)
Label(self.simplexFrme, text=" x + ", font=fontSmall, bg="#eaebed", fg="#1b263b").grid(row=8, column=10)
yEntry = Entry(self.simplexFrme, width=5)
yEntry.grid(row=8, column=11)
Label(self.simplexFrme, text=" y", font=fontSmall, bg="#eaebed", fg="#1b263b").grid(row=8, column=12)

```

In order to be able to use the values entered, I created a list called `textConstraints`, which stores the widgets for the values entered. In the validate method and when creating the simplex tableau, I will use `.get()` to get the values entered.

4.1.h

```
class Simplex(Menu):
    def __init__(self, master, pUsername):
        Menu.__init__(self, master, 12, 14)
        self.username = pUsername

        self.textConstraints = [] # for widgets
```

In `addConstraint`:

4.1.i

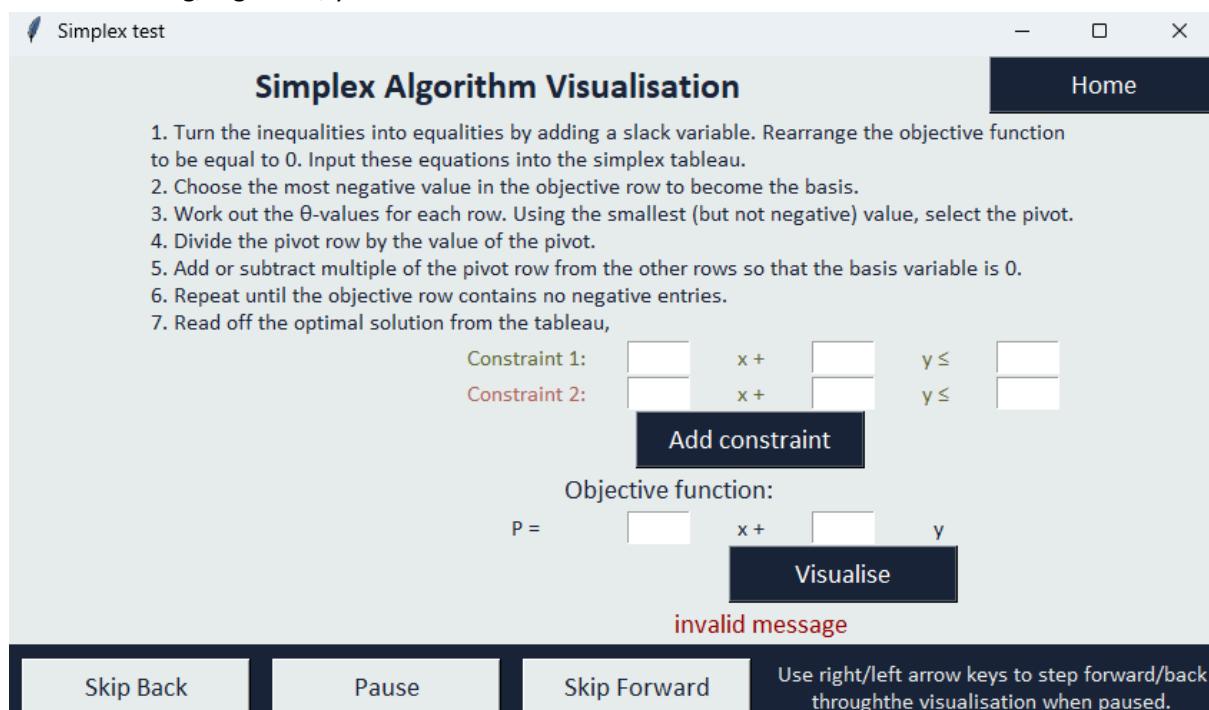
```
self.textConstraints.append({
    "x": xEntry,
    "y": yEntry,
    "rhs": rhsEntry})
```

As this is also needed for the objective function, I wrote the following code for that:

4.1.j

```
self.objectiveText = {"x": xEntry,
                     "y": yEntry}
```

The reason I used dictionaries was because this would allow me to use the key to determine which value I am using, e.g. the x, y or rhs value.



When the Add Constraint button is pressed, the following is output:

The screenshot shows a window titled "Simplex Algorithm Visualisation". The interface includes a "Home" button in the top right. Below it is a list of 7 steps for solving a simplex tableau:

- Turn the inequalities into equalities by adding a slack variable. Rearrange the objective function to be equal to 0. Input these equations into the simplex tableau.
- Choose the most negative value in the objective row to become the basis.
- Work out the θ -values for each row. Using the smallest (but not negative) value, select the pivot.
- Divide the pivot row by the value of the pivot.
- Add or subtract multiple of the pivot row from the other rows so that the basis variable is 0.
- Repeat until the objective row contains no negative entries.
- Read off the optimal solution from the tableau,

Below these instructions is a simplex tableau with 3 rows (Constraints 1, 2, 3) and 4 columns (x+, y, y ≤, empty). The objective function P = is shown below, with a "Visualise" button. A red "invalid message" is displayed at the bottom.

Skip Back Pause Skip Forward Use right/left arrow keys to step forward/back through the visualisation when paused.

These outputs are as expected so I am now going to develop the validate method. For this the requirements are that x and y are not both equal to 0 and that all numbers entered are between -10 and 10. Additionally, for simplicity, I will cast the values to be integers, although they could be float values.

4.1.k

```
def validate(self):
    print("in validate")
    # validate constraints
    for constraint in self.textConstraints:
        try:
            x = int(constraint["x"].get())
            y = int(constraint["y"].get())
            rhs = int(constraint["rhs"].get())
            if x == 0 and y == 0:
                raise ValueError
            if x < -10 or x > 10:
                raise ValueError
            if y < -10 or y > 10:
                raise ValueError
            if rhs < -10 or rhs > 10:
                raise ValueError
        except ValueError:
            self.invalidMessage["text"] = "Invalid constraint entry: please ensure that all numbers entered are integers"
            return

    # validate objective
    try:
        x = int(self.objectiveText["x"].get())
        y = int(self.objectiveText["y"].get())
        if x == 0 and y == 0:
            raise ValueError
        if x < -10 or x > 10:
            raise ValueError
        if y < -10 or y > 10:
            raise ValueError
    except ValueError:
        self.invalidMessage["text"] = "Invalid objective entry: please ensure that all numbers entered are integers"
        return
```

I made sure that the invalidMessage text is specific about what part is causing the error because this will make it clearer to the user what they need to fix. Now, to check that this validation works

correctly, I put print statements in to check that each section is validated correctly. For a fully valid input, the following is output:

The screenshot shows the software interface with the following data:

- Constraints:**
 - Constraint 1: $x + 0y \leq -10$
 - Constraint 2: $3x + 4y \leq 7$
- Objective Function:** $P = -2x + -7y$
- Buttons:** Add constraint, Visualise, Skip Back, Pause, Skip Forward.
- Text on the right:** in add constraint
in add constraint
in validate
constraints valid
objective valid
fully valid

In order to check that numbers must be integers, I entered the following and the following was output:

The screenshot shows the software interface with the following data:

- Constraints:**
 - Constraint 1: $1.5x + 3.4y \leq 2$
 - Constraint 2: $10x + -5.9y \leq 2$
- Objective Function:** $P = 6x + 6y$
- Buttons:** Add constraint, Visualise, Skip Back, Pause, Skip Forward.
- Text on the right:** in add constraint
in add constraint
in validate

A red message at the bottom left says: Invalid constraint entry: please ensure that all numbers entered are between -10 and 10, with the x and y coefficients not both being 0.

This is as expected. Now to check that the range conditions are correctly checked, the following is output:

The screenshot shows the software interface with the following data:

- Constraints:**
 - Constraint 1: $-11x + 13y \leq 56$
 - Constraint 2: $2x + 6y \leq 0$
- Objective Function:** $P = 14x + 17y$
- Buttons:** Add constraint, Visualise, Skip Back, Pause, Skip Forward.
- Text on the right:** in add constraint
in add constraint
in validate

A red message at the bottom left says: Invalid constraint entry: please ensure that all numbers entered are between -10 and 10, with the x and y coefficients not both being 0.

This is again as expected. Finally, to check that x and y can't both be 0:

The screenshot shows a window titled "Simplex test" with the title bar "Simplex Algorithm Visualisation" and a "Home" button. Below the title bar is a list of 7 steps for solving a linear programming problem using the simplex method. The main area contains a simplex tableau and an objective function input field.

Simplex Tableau:

Constraint 1:	0	x +	0	y ≤	3
Constraint 2:	5	x +	6	y ≤	0

Objective Function:

$$P = 10 \quad x + \quad -10 \quad y$$

Buttons:

- Add constraint
- Visualise
- Skip Back
- Pause
- Skip Forward
- Use right/left arrow keys to step forward/back through the visualisation when paused.

Text on the right:

```
in add constraint
in add constraint
in validate
```

All these test outputs are as expected for the constraint testing. As the tests for the objective function are the same, I will not test this now.

Next, as simplex algorithm uses a tableau, with rows for each of the constraints and the objective function and columns for the x, y, slack variable and rhs values. So, I am going to develop a method called `createTableau` that will do this. However, for this, the number values for each of the x,y,rhs entries needs to be used. For easier programming, I modified the `validate` method to append these values to attributes called `valueConstraints` and `valueObjective`, which hold the validated values entered:

4.1.l

```
if x == 0 and y == 0:
    raise ValueError
if x < -10 or x > 10:
    raise ValueError
if y < -10 or y > 10:
    raise ValueError
if rhs < -10 or rhs > 10:
    raise ValueError
self.valueConstraints.append([x,y,rhs])
```

4.1.m

```
if x == 0 and y == 0:
    raise ValueError
if x < -10 or x > 10:
    raise ValueError
if y < -10 or y > 10:
    raise ValueError
print("objective valid")
self.valueObjective = [x,y]
```

Then, for the `createTableau` method, I wrote the following code to add the values in `valueConstraints` and `valueObjective` into the 2D array that is the tableau.

4.1.n

```
def createTableau(self):
    print("in create tableau")
    numSlackVars = len(self.valueConstraints)

    for i, constraint in enumerate(self.valueConstraints):
        row = constraint[:-1] # take all coefficients except rhs
        row += [1 if j==i else 0 for j in range(numSlackVars)] # add slack variable
        row.append(constraint[-1]) # append rhs value
        self.tableau.append(row)

    # add objective with -ve coefficients as maximising
    objectiveRow = [-c for c in self.valueObjective] + [0]*(numSlackVars +1)
    self.tableau.append(objectiveRow)
```

This method is then called at the end of `validate()`:

4.1.o

```

print("fully valid")
# all validation passed
self.createTableau()

```

To test the code, I added the following lines of code to print the tableau:

```

# check
for line in self.tableau:
    print(line)

```

When I run the code, the following is output:

The screenshot shows a window titled "Simplex Algorithm Visualisation". It contains a "Simplex test" section with instructions for solving linear programming problems using the simplex method. Below this is a "Constraint" section showing two rows of equations:

Constraint 1:	3	x +	6	y ≤	-3
Constraint 2:	-10	x +	1	y ≤	1

Below the constraints is an "Add constraint" button. Further down is an "Objective function" section with the equation:

$$P = -4x + 2y$$

Below the objective function is a "Visualise" button. At the bottom of the window are buttons for "Skip Back", "Pause", "Skip Forward", and a note: "Use right/left arrow keys to step forward/back through the visualisation when paused."

```

in add constraint
in add constraint
in validate
in create tableau
[3, 6, 1, 0, 3]
[-10, 1, 0, 1, -10]
[4, 4, 0, 0, 0]

```

The Shell output indicates that the tableau has been created correctly. Now, I am going to create the matplotlib graph that has the entered constraints plotted on it.

4.1.p

```

def plotConstraints(self):
    colours = ["#606c38", "#b2675e", "#eca400"]
    xVals = np.linspace(0, 10, 200) # used to plot constraint lines on graph

    fig, ax = plt.subplots(figsize=(5,5))
    for i, (a,b,c) in enumerate(self.valueConstraints):
        if b != 0: # is a line in the form by = ax + c
            yVals = (c - a*xVals) / b
        else: # in the form ax = c
            xVals = np.full_like(xVals, c / a)
            yVals = np.linspace(0, 10, 200)
        ax.plot(xVals, yVals, label=f"Constraint{i+1}", color=colours[i])

    ax.set_xlim(0,10)
    ax.set_ylim(0,10)
    ax.axhline(0, color="black", linewidth=1)
    ax.axvline(0, color="black", linewidth=1)
    ax.set_xlabel("x")
    ax.set_ylabel("y")
    ax.legend()
    ax.grid()

    self.canvas = FigureCanvasTkAgg(fig, master=self.simplexFrm)
    self.canvas.grid(row=4, column=0, rowspan=8, columnspan=8)
    self.canvas.draw()

```

I call this method at the end of validate. When I run the code, the following is output to the Shell:

```

in add constraint
in add constraint
in validate
constraints valid
objective valid
fully valid
in create tableau
[-8, 2, 1, 0, -8]
[5, 1, 0, 1, 5]
[7, 7, 0, 0, 0]
Exception in Tkinter callback
Traceback (most recent call last):
  File "C:\Users\ingri\AppData\Local\Programs\Thonny\lib\tkinter\_init_.py", line 1921, in __call__
    return self.func(*args)
  File "C:\Users\ingri\OneDrive\Documents\QM\QM School work\A Level\Computer Science\NEA\simplex_page.py", line
ate
    self.plotConstraints()
  File "C:\Users\ingri\OneDrive\Documents\QM\QM School work\A Level\Computer Science\NEA\simplex_page.py", line
onstraints
    self.canvas.grid(row=4, column=0, rowspan=8, columnspan=8)
AttributeError: 'FigureCanvasTkAgg' object has no attribute 'grid'

```

The error is caused by not using `.get_widget()`, which is a quick fix:

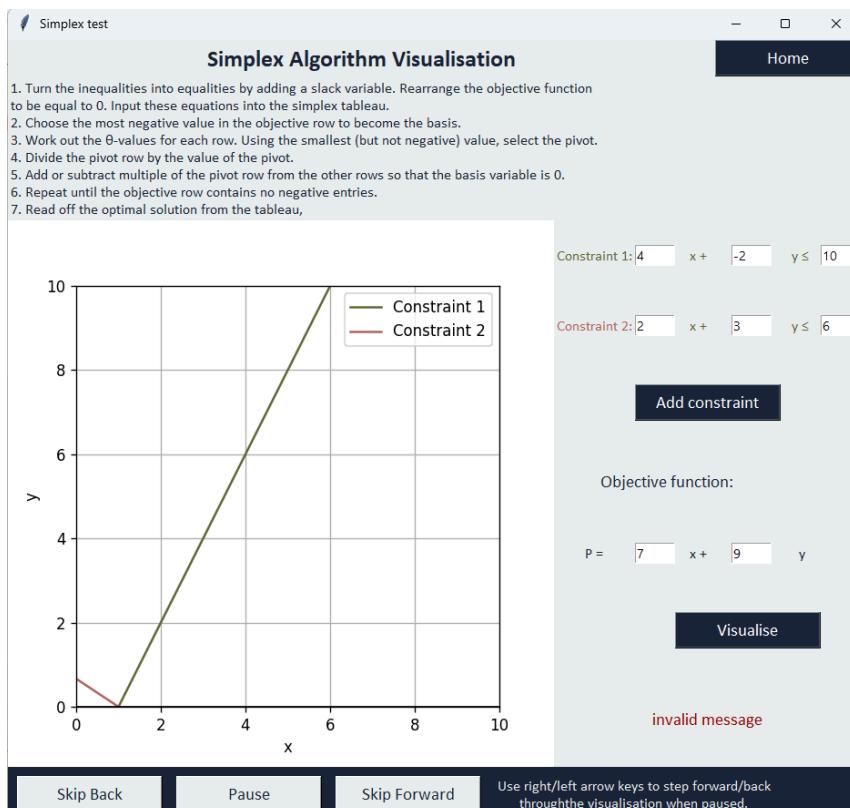
4.1.q

```

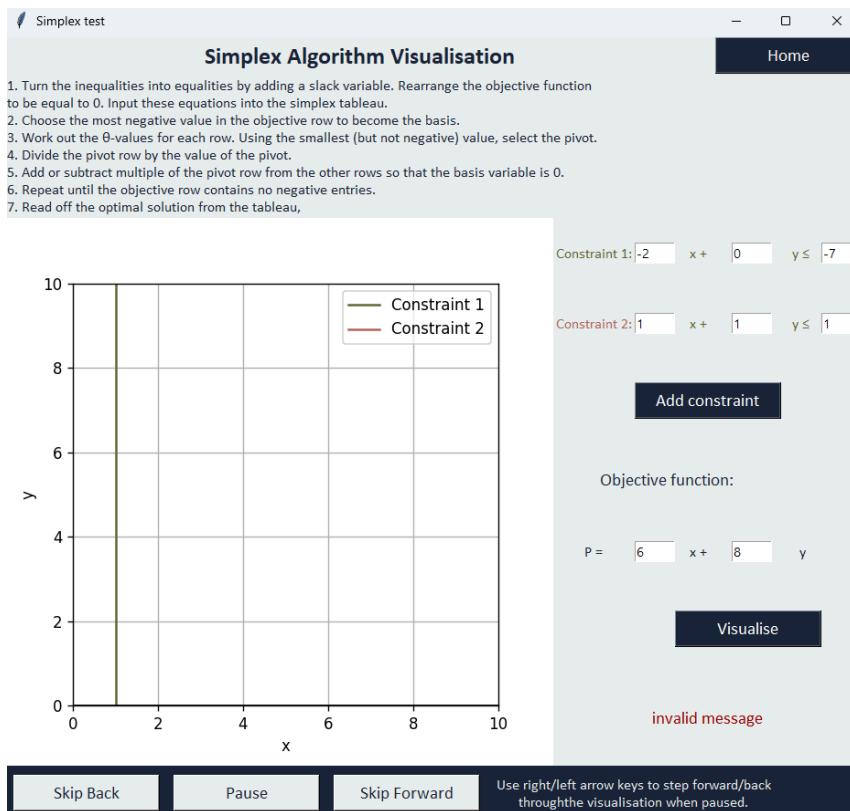
self.canvas = FigureCanvasTkAgg(fig, master=self.simplexFrm)
self.canvas.get_tk_widget().grid(row=4, column=0, rowspan=8, columnspan=8)
self.canvas.draw()

```

When I now run the code, the following is output:



For a different input to check that the vertical line works correctly, I get the following output:



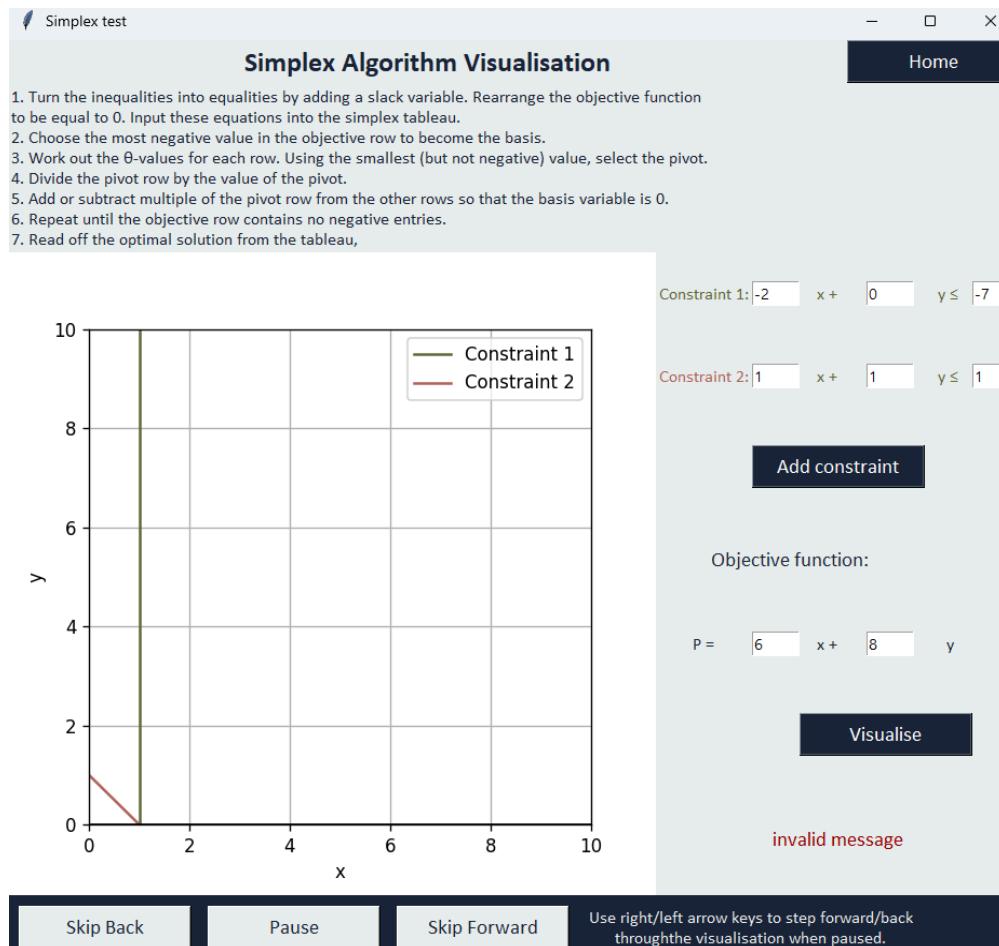
On this one, it seems that only one of the constraints is plotted on the graph. The Constraint 1, which is the one testing the vertical line aspect, is plotted correctly, however. To try and fix the plotting of the lines so that both constraints are plotted correctly, I had the idea to move the line that creates the x values for the line to be plotted on into the for loop so that each constraint has its own values of x to be plotted on.

4.1.r

```
def plotConstraints(self):
    colours = ["#606c38", "#b2675e", "#eca400"]

    fig, ax = plt.subplots(figsize=(5,5))
    for i, (a,b,c) in enumerate(self.valueConstraints):
        xVals = np.linspace(0, 10, 200) # used to plot constraint lines on graph
        if b != 0: # is a line in the form ax + by = c
            yVals = (c - a*xVals) / b
        else: # in the form ax = c
            xVals = np.full_like(xVals, c / a)
            yVals = np.linspace(0, 10, 200)
        ax.plot(xVals, yVals, label=f"Constraint {i+1}", color=colours[i])
```

When I now run the code, with the same lines, the following is output:

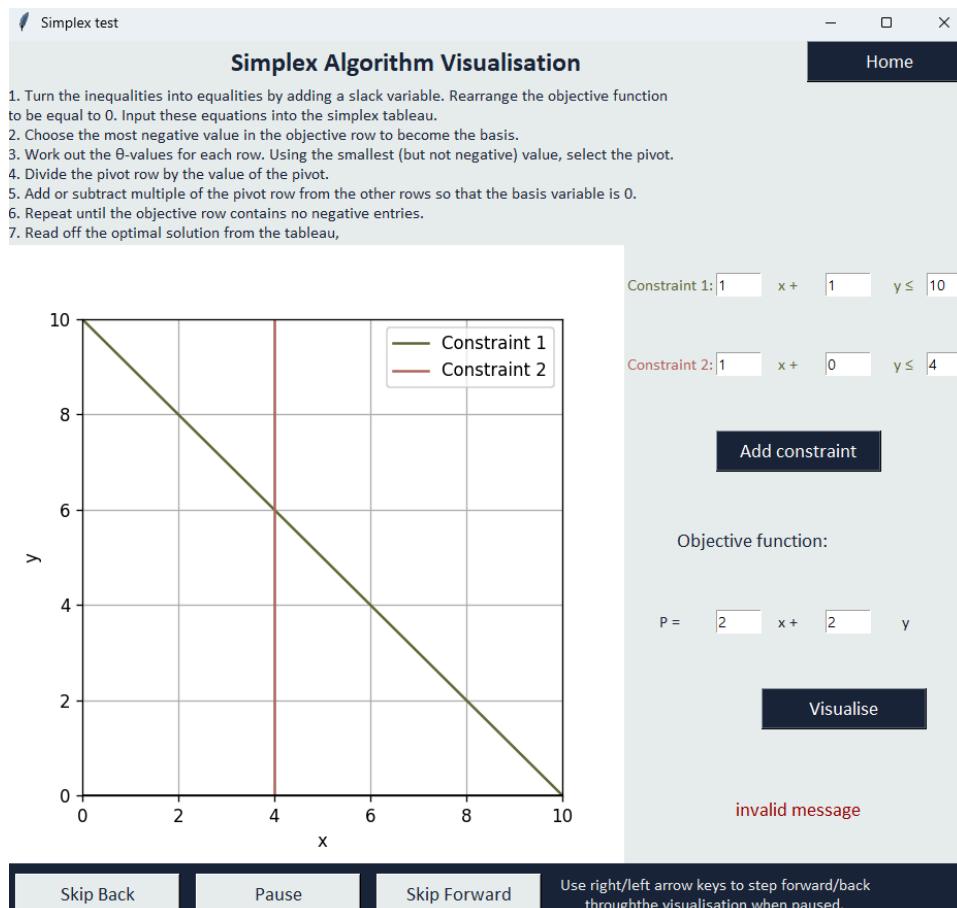


This is again not as expected. In order to try and fix it, I looked at my code again and realised that I had written “x” instead of “rhs” to get the value of rhs. To fix this, I changed the code to the following:

4.1.s

```
x = int(constraint["x"].get())
y = int(constraint["y"].get())
rhs = int(constraint["rhs"].get())
```

Now when the code is run, I get the following output:



This is as expected.

Now I am going to focus on developing the implementation of the simplex algorithm in my code. Firstly, I developed the following methods which will be used multiple times throughout the execution of finding the steps. Below is the code for it.

4.1.t

```
def isOptimal(self):
    print("in is optimal")
    # checks all values in objective row except rhs are non-negative
    return np.all(self.tableau[-1, :-1] >= 0)

def findPivotColumn(self):
    # find most negative coef in objective row
    return np.argmin(self.tableau[-1, :-1])

def findPivotRow(self, pivotColumn):
    # find row by doing rhs/pivot column value - least positive
    ratios = []
    for i in range(self.numConstraints):
        if self.tableau[i, pivotColumn] > 0:
            ratios.append(self.tableau[i, -1] / self.tableau[i, pivotColumn])
        else:
            ratios.append(float("inf")) # ignores negative or 0 values
    return np.argmin(ratios)
```

For this, it is clear that I have been using the numpy library. This is because it has built in functionality that allows calculations like the ones shown above to be done without me having to code them.

Then, as I am using Gaussian elimination style row reduction, which is the method taught in A-Level Further Maths, below is the code for this:

4.1.u

```
def pivot(self, row, column):
    # perform row op to change values by Gaussian elimination style row reduction
    pivotValue = self.tableau[row, column]

    # 1: make pivot element 1
    self.tableau[row] /= pivotValue

    # 2: row reduction
    for i in range(len(self.tableau)):
        if i != row:
            multiplier = self.tableau[i, col]
            self.tableau[i] -= multiplier * self.tableau[row]
```

The advantage of using this method is that it reduces the need for recomputation and it makes it easier to read off the final solution. Then, in terms of putting these methods together in simplexSteps, I wrote the following code:

4.1.v

```
def simplexSteps(self):
    print("in simplex steps")
    self.steps = []
    while not self.isOptimal():
        pivotColumn = self.findPivotColumn()
        pivotRow = self.findPivotRow(pivotColumn)

        # check for unbounded:
        if self.tableau[pivotRow, pivotColumn] <= 0 :
            self.invalidMessage["text"] = "Unbounded solution."
            break

        # steps append ...
        x = self.getVariableValue(0) # x is in tableau column 0
        y = self.getVariableValue(1) # y is in column index 1
        P = self.tableau[-1][-1]
        self.steps.append((x,y,P))

        self.pivot(pivotRow, pivotColumn)
```

For the visualisation of Simplex, I used the same method of appending values into an attribute called steps. In terms of how I want it to be visualised, I want the points in the feasible region currently being visited to be shown on the graph clearly, with a path joining them. Additionally, the value of P needs to be added so that the final value - the optimal value - can be displayed at the end. For this, the values of x and y need to be obtained from the tableau. For this I wrote the following method for getting the value:

4.1.w

```

def getVariableValue(self, varIndex):
    # find row where variable is basic
    column = self.tableau[:, varIndex] # selects all rows in varIndex column
    basicRow = -1

    for i in range(len(column) - 1): # ignores last row as this is the objective function
        if column[i] == 1:
            # ensure all other coefficients in row are 0
            if np.count_nonzero(column)==1:
                basicRow = i
                break

    print("basic row is:", basicRow)
    # variable is basic returns value in rightmost column
    # variable not basic returns 0
    return self.tableau[basicRow, -1] if basicRow != -1 else 0

```

For using these values in the visualisation, I wrote the following method to update the graph:

4.1.x

```

def updateSimplexGraph(self):
    print("in update simplex graph")
    self.ax.clear()

    # replot constraints
    self.plotConstraints()

    # draw path of visited points
    if len(self.steps) > 1:
        xVals, yVals = zip(*self.steps[: self.currentStep +1])
        self.ax.plot(xVals, yVals, "r-", linewidth=3)

    # highlight current point
    if self.currentStep < len(self.steps):
        self.ax.plot(self.steps[self.currentStep][0], self.steps[self.currentStep][1], "ro", markersize=8)

    self.canvas.draw()

```

Finally, I wrote the following method to animate the visualisation:

4.1.z

```

def simplexAnimate(self):
    print("in simplex animate")
    if self.currentStep < len(self.steps)-1:
        self.updateSimplexGraph()
        self.currentStep += 1
        self.master.after(750, self.simplexAnimate)
    elif self.currentStep == len(self.steps)-1:
        self.updateSimplexGraph()
        self.invalidMessage["text"] = f"Optimal solution P = {self.steps[-1][2]}\n{x = {self.steps[-1][0]}, y = {self.steps[-1][1]}}"

```

The code in this method is pretty similar in structure to the code for the animation of the other algorithms already developed. Now, when I run the code, the following is output:

```

Exception in Tkinter callback
Traceback (most recent call last):
  File "C:\Users\ingri\AppData\Local\Programs\Thonny\lib\tkinter\__init__.py", line 1921, in __call__
    return self.func(*args)
  File "C:\Users\ingri\OneDrive\Documents\QM\QM School work\A Level\Computer Science\NEA\simplex_page.py", line 151, in update
    self.simplexSteps()
  File "C:\Users\ingri\OneDrive\Documents\QM\QM School work\A Level\Computer Science\NEA\simplex_page.py", line 275, in simplexSteps
    while not self.isOptimal():
  File "C:\Users\ingri\OneDrive\Documents\QM\QM School work\A Level\Computer Science\NEA\simplex_page.py", line 206, in isOptimal
    return np.all(self.tableau[-1, :-1] >= 0)
TypeError: list indices must be integers or slices, not tuple

```

It seems that the issue occurs because I have just created tableau as an array, not a numpy array. Therefore the operations I want to be carried out on it can't be due to a type error. I fixed this with the following lines of code:

4.1.A

```
class Simplex(Menu):
    def __init__(self, master, pUsername):
        Menu.__init__(self, master, 12, 14)
        self.username = pUsername

        self.textConstraints = [] # for widgets
        self.valueConstraints = []
        self.valueObjective = []
        self.tableau = np.array([], dtype=float) # empty numpy array
```

When I now run the visualisation, the following is output:

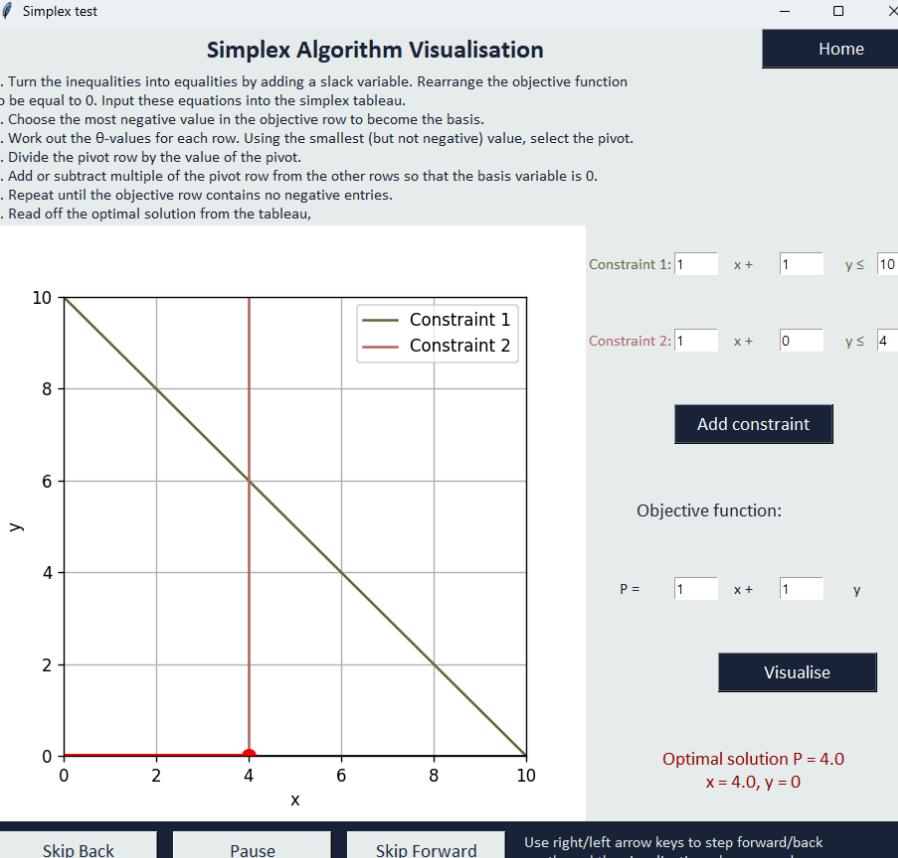
```
Exception in Tkinter callback
Traceback (most recent call last):
  File "C:\Users\ingri\AppData\Local\Programs\Thonny\lib\tkinter\_init_.py", line 1921, in __call__
    return self.func(*args)
  File "C:\Users\ingri\OneDrive\Documents\QM\QM School work\A Level\Computer Science\NEA\simplex_page.py", line 151, in validate
    self.simplexSteps()
  File "C:\Users\ingri\OneDrive\Documents\QM\QM School work\A Level\Computer Science\NEA\simplex_page.py", line 293, in simplexSteps
    self.pivot(pivotRow, pivotColumn)
  File "C:\Users\ingri\OneDrive\Documents\QM\QM School work\A Level\Computer Science\NEA\simplex_page.py", line 233, in pivot
    self.tableau[row] /= pivotValue
numpy.core._exceptions._UFuncOutputCastingError: Cannot cast ufunc 'divide' output from dtype('float64') to dtype('int64') without casting rule 'same_kind'
```

This error seems to be because I have cast the values to be integers instead of floats when validating them. I didn't realise at the time that this would be an issue but I will now change them to the following:

4.1.B

```
x = float(constraint["x"].get())
y = float(constraint["y"].get())
rhs = float(constraint["rhs"].get())
.
```

Now to test the code, I input the constraints $x + y \leq 10$ and $x \leq 4$. The objective function is $P=x+y$. This should result in the output $P=10$ $x=4$ and $y=6$. Additionally, to test the flow of the program, I put print statements in to check what the values of the pivot row/column, x y and P and the tableau are. When I run the code, the following is output:



Simplex Algorithm Visualisation

1. Turn the inequalities into equalities by adding a slack variable. Rearrange the objective function to be equal to 0. Input these equations into the simplex tableau.
 2. Choose the most negative value in the objective row to become the basis.
 3. Work out the θ -values for each row. Using the smallest (but not negative) value, select the pivot.
 4. Divide the pivot row by the value of the pivot.
 5. Add or subtract multiple of the pivot row from the other rows so that the basis variable is 0.
 6. Repeat until the objective row contains no negative entries.
 7. Read off the optimal solution from the tableau,

Constraint 1: $1 \quad x + \quad 1 \quad y \leq 10$

Constraint 2: $1 \quad x + \quad 0 \quad y \leq 4$

Add constraint

Objective function:

$P = 1 \quad x + 1 \quad y$

Visualise

Optimal solution $P = 4.0$
 $x = 4.0, y = 0$

```

in add constraint
in add constraint
in validate
1.0 1.0 10.0
1.0 0.0 4.0
constraints valid
objective valid
1.0 1.0
fully valid
in create tableau
[ 1. 1. 1. 0. 10. ]
[ 1. 0. 0. 1. 4. ]
[ -1. -1. 0. 0. 0. ]
1.0 1.0 10.0
1.0 0.0 4.0
in simplex steps
in is optimal
pivot column: 1
pivot row: 0
basic row is: -1
basic row is: -1
x, y, P: 0 0 4.0
[[ 0. 1. 1. -1. 6. ]
 [ 1. 0. 0. 1. 4. ]
 [ 0. -1. 0. 1. 4. ]]
in is optimal
in simplex animate
in update simplex graph
1.0 1.0 10.0
1.0 0.0 4.0
in simplex animate
in update simplex graph
1.0 1.0 10.0
1.0 0.0 4.0
    
```

This is not as expected.

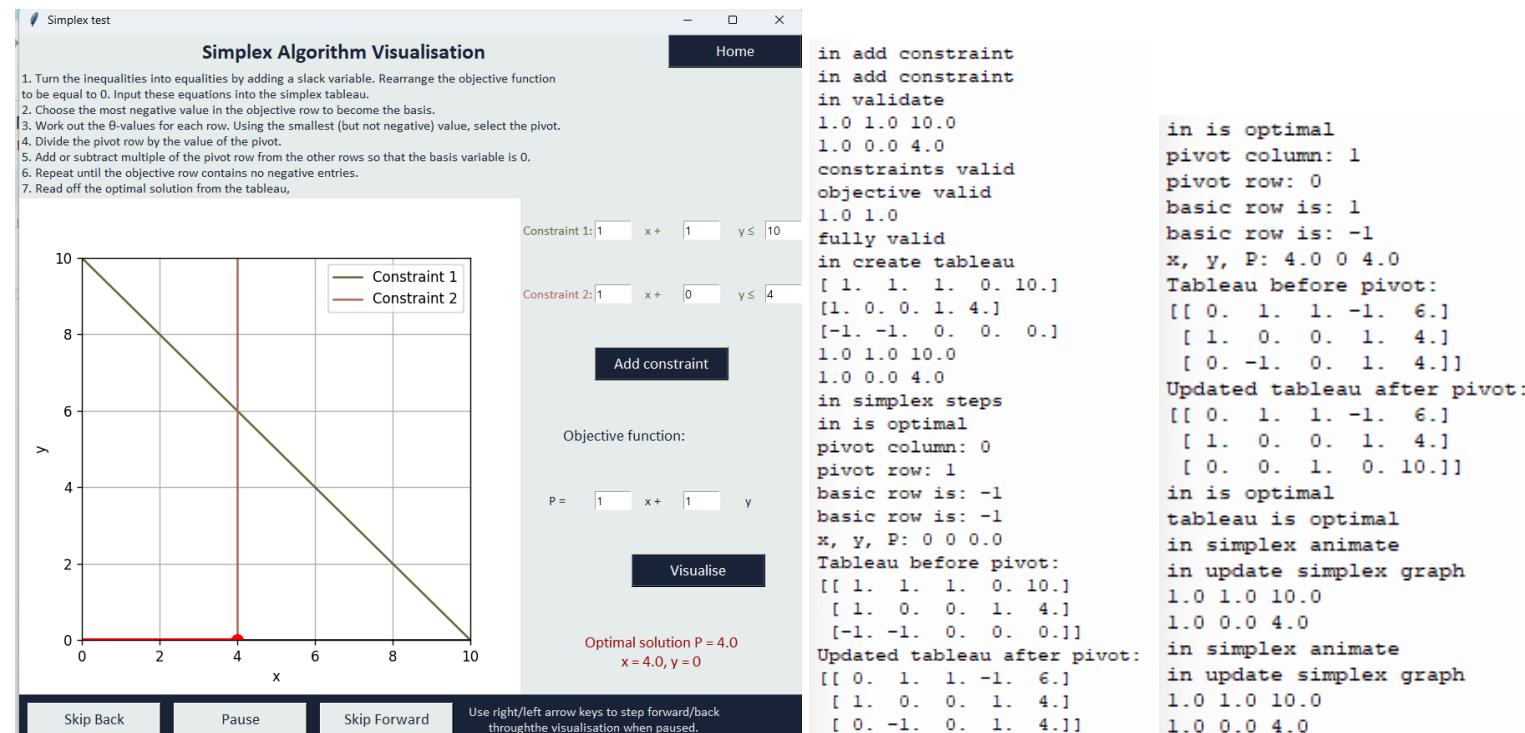
432

I was not really sure what was causing the error but because it stopped at the stage before the final optimal solution stage, I think the issue may be that the tableau is being determined as being optimal before it actually is fully optimal. I researched what may be causing this issue and I think that because of float values being used, the `isOptimal` method doesn't quite identify the value as being negative. So I changed it to the following:

4.1.C

```
def isOptimal(self):
    print("in is optimal")
    # checks all values in objective row except rhs are non-negative
    return np.all(self.tableau[-1, :-1] >= -1e-9)
```

Additionally, to check that the tableau is pivoting correctly, I added print statements to check its stage. Below is the new output:



From this, it can be seen that the final state of the tableau is correct, as the x and y are both basic variables and the values in the final column are both correct. Therefore, the issue is occurring in the getVariableValues method.

4.1.D

```
def getVariableValue(self, varIndex):
    # find row where variable is basic
    column = self.tableau[:, varIndex] # selects all rows in varIndex column
    basicRow = -1

    for i in range(len(column) - 1): # ignores last row as this is the objective function
        if abs(column[i]-1) < 1e-9: # avoids rounding error of 0.999999999
            # ensure all other coefficients in row are 0
            if np.count_nonzero(column)==1:
                basicRow = i
                break

    print("basic row is:", basicRow)
    # variable is basic returns value in rightmost column
    # variable not basic returns 0
    return self.tableau[basicRow, -1] if basicRow != -1 else 0
```

Additionally, I think that there may be an issue with at what point in the execution I am appending the values to steps. So I changed simplexSteps to the following:

4.1.E

```
def simplexSteps(self):
    print("in simplex steps")
    self.steps = []
    while not self.isOptimal():
        pivotColumn = self.findPivotColumn()
        print("pivot column:", pivotColumn)
        pivotRow = self.findPivotRow(pivotColumn)
        print("pivot row:", pivotRow)

        # check for unbounded:
        if self.tableau[pivotRow, pivotColumn] <= 0 :
            self.invalidMessage["text"] = "Unbounded solution."
            break

        print("Tableau before pivot:")
        print(self.tableau)

        self.pivot(pivotRow, pivotColumn)

        # steps append ...
        x = self.getVariableValue(0) # x is in tableau column 0
        y = self.getVariableValue(1) # y is in column index 1
        P = self.tableau[-1][-1]
        print("x, y, P:", x, y, P)
        self.steps.append((x,y,P))

        print("tableau is optimal")
        self.steps.append((x,y,P))
        print("steps:")
        print(self.steps)
```

When I now run the code, the following is output:

Simplex Algorithm Visualisation

1. Turn the inequalities into equalities by adding a slack variable. Rearrange the objective function to be equal to 0. Input these equations into the simplex tableau.
 2. Choose the most negative value in the objective row to become the basis.
 3. Work out the θ -values for each row. Using the smallest (but not negative) value, select the pivot.
 4. Divide the pivot row by the value of the pivot.
 5. Add or subtract multiple of the pivot row from the other rows so that the basis variable is 0.
 6. Repeat until the objective row contains no negative entries.
 7. Read off the optimal solution from the tableau,

- □ ×

Home

Constraint 1:

+

y ≤

Constraint 2:

+

y ≤

Add constraint

Visualise

Optimal solution P = 10.0
 $x = 4.0, y = 6.0$

Skip Back
Pause
Skip Forward

Use right/left arrow keys to step forward/back through the visualisation when paused.

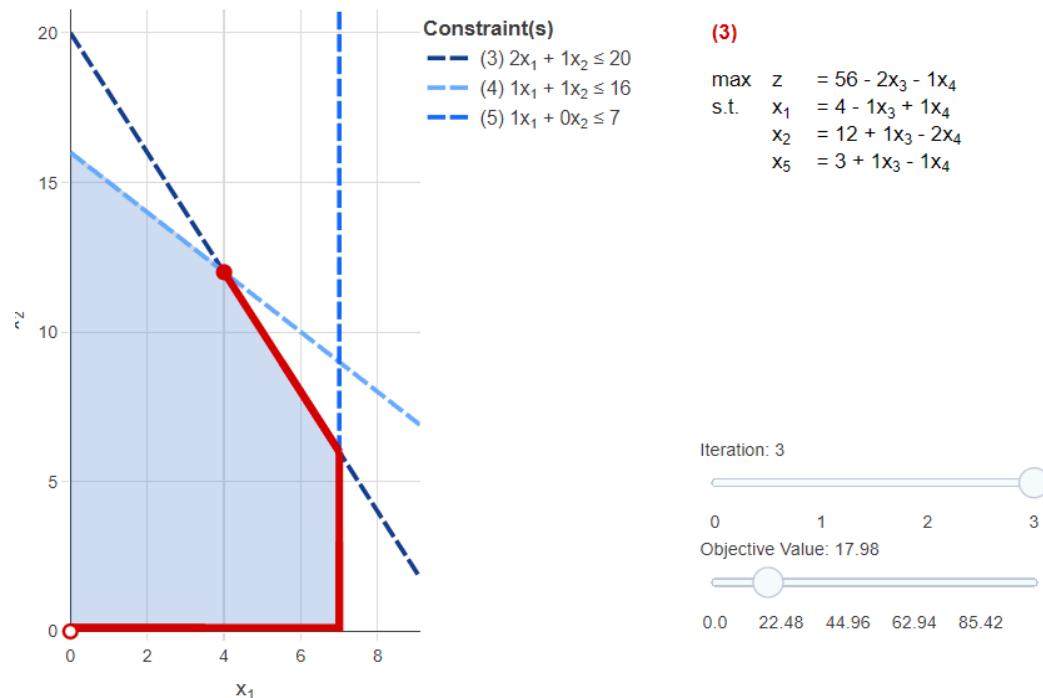
```

in add constraint
in add constraint
in validate
1.0 1.0 10.0
1.0 0.0 4.0
constraints valid
objective valid
1.0 1.0
fully valid
in create tableau
[[ 1. 1. 1. 0. 10.]
 [1. 0. 0. 1. 4.]
 [-1. -1. 0. 0. 0.]]
1.0 1.0 10.0
1.0 0.0 4.0
in simplex steps
in is optimal
pivot column: 0
pivot row: 1
Tableau before pivot:
[[ 1. 1. 1. 0. 10.]
 [1. 0. 0. 1. 4.]
 [-1. -1. 0. 0. 0.]]
Updated tableau after pivot:
[[ 0. 1. 1. -1. 6.]
 [1. 0. 0. 1. 4.]
 [0. 0. 1. 0. 10.]]
basic row is: 1
basic row is: 0
x, y, P: 4.0 6.0 10.0
in is optimal
tableau is optimal
steps:
[(np.float64(4.0), 0, np.float64(4.0)), (np.float64(4.0), np.float64(6.0), np.float64(10.0)), (np.float64(4.0), np.float64(6.0), np.float64(10.0))]
in simplex animate
in update simplex graph
1.0 1.0 10.0
1.0 0.0 4.0
in simplex animate
in update simplex graph
1.0 1.0 10.0
1.0 0.0 4.0
in simplex animate
in update simplex graph
1.0 1.0 10.0
1.0 0.0 4.0
in simplex animate
in update simplex graph
1.0 1.0 10.0
1.0 0.0 4.0

```

435

From the outputs, it is clear that the optimal solution has now been correctly calculated. Now, I am going to try and change the way updateSimplexGraph to follow the path along the edges of the feasible region, starting at (0,0). This is similar to the example from the existing solution of GILP:



To make the changes, I think it would be best if I make an attribute called simplexPath, which stores the x and y coordinates being visited. The code now looks like this:

4.1.F

```
self.simplexPath = [(0,0)]

# steps append ...
x = self.getVariableValue(0) # x is in tableau column 0
y = self.getVariableValue(1) # y is in column index 1
P = self.tableau[-1][-1]
print("x, y, P:", x, y, P)
self.steps.append((x,y,P))
self.simplexPath.append((x,y))
```

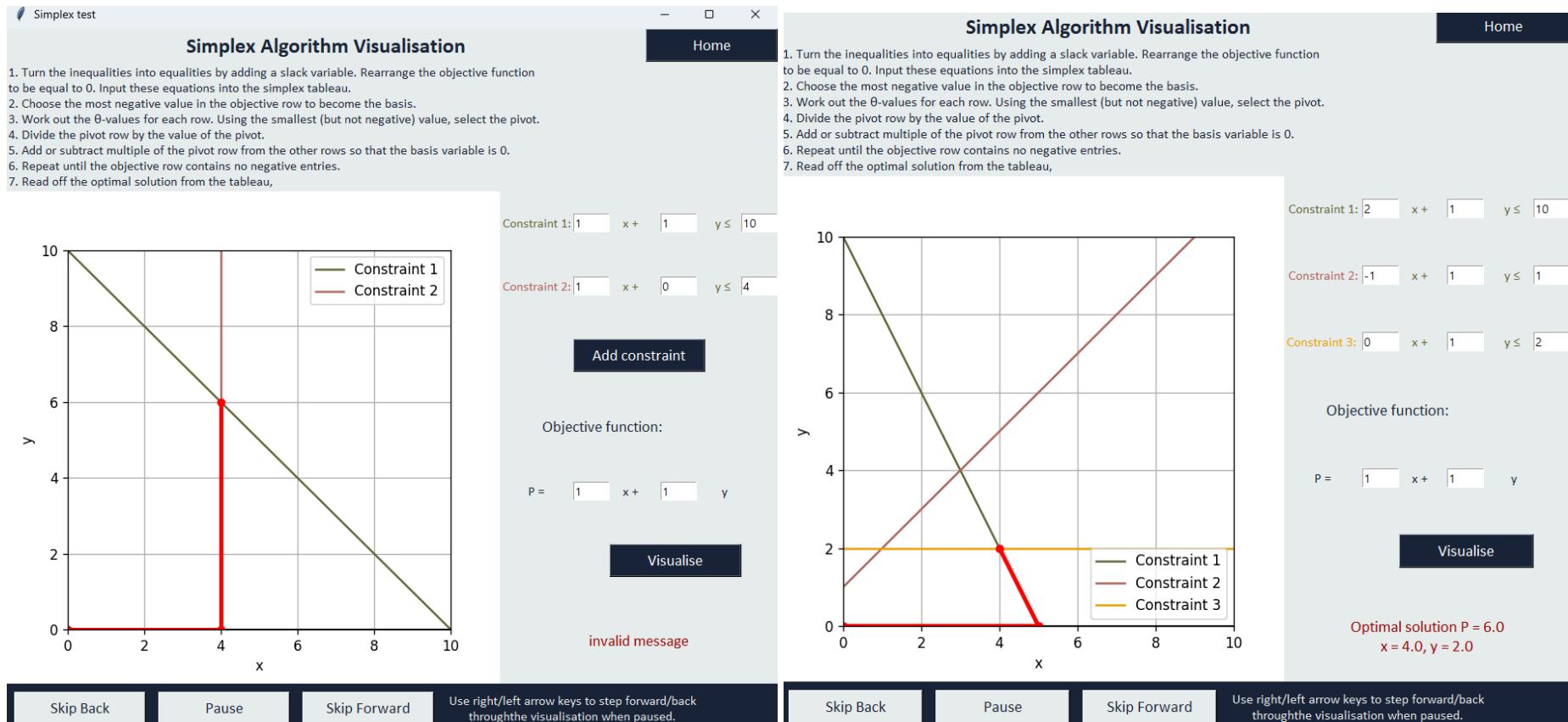
4.1.G

```
def updateSimplexGraph(self):
    print("in update simplex graph")
    self.ax.clear()

    # replot constraints
    self.plotConstraints()

    # draw path of visited points
    if self.currentStep > 0:
        xPath, yPath = zip(*self.simplexPath[:self.currentStep +1])
        self.ax.plot(xPath, yPath, "ro-", linewidth=3, markersize=5)
    self.canvas.draw()
```

When I run the code, the following is output:



Both outputs are as expected. Therefore, the main functionality of the Simplex algorithm visualisation is complete.

Now, I am going to focus on extending the Menu class to include the functionality for the Simplex algorithm. I did this with the following code:

4.1.H

```
def togglePlayPause(self):
    print("in toggle play pause")
    self.isPaused = not self.isPaused # switches the state
    self.playPauseButton.config(text="Pause" if not self.isPaused else "Play")
    if not self.isPaused: # ie displays Pause but the visualisation is playing
        print("in not self.isPaused")
        #self.playAnimation()
        if hasattr(self, "bubbleSortAnimate"):
            self.bubbleSortAnimate()

    elif hasattr(self, "primAnimate"):
        self.primAnimate()

    elif hasattr(self, "dijkstraAnimate"):
        #print("in dijkstra toggle play pause")
        self.dijkstraAnimate()

    elif hasattr(self, "simplexAnimate"):
        self.simplexAnimate()
```

4.1.I

```
def skipForward(self):
    self.currentStep = len(self.steps) - 1 # gets the last step index
    self.isPaused = True
    if hasattr(self, "updateChart"): # note that methods count as attributes
        self.updateChart(self.steps[self.currentStep])

    elif hasattr(self, "updatePrimGraph"):
        self.updatePrimGraph(self.steps[self.currentStep])

    elif hasattr(self, "updateDijkstraGraph"):
        print("in dijkstra skip forward")
        step = self.steps[self.currentStep]
        pathEdges = [(step[1][i], step[1][i+1]) for i in range(len(step[1]) - 1)]
        self.updateDijkstraGraph([], checkingEdges=pathEdges)
        shortestPathText = " -> ".join(step[1])
        self.invalidMessage["text"] = f"Shortest path between {self.startVertexChoice.get()} and {self.endVertexChoice.get()}: {shortestPathText}"
        step1 = self.steps[self.currentStep-1] # so that the vertex table is actually filled in
        self.updateDijkstraTable(step1)

    elif hasattr(self, "updateSimplexGraph"):
        self.updateSimplexGraph()
```

4.1.J

```
def skipBack(self):
    self.currentStep = 0
    self.isPaused = True

    if hasattr(self, "updateChart"): # note that methods count as attributes
        self.updateChart(self.steps[self.currentStep])

    elif hasattr(self, "updatePrimGraph"):
        self.updatePrimGraph(self.steps[self.currentStep])

    elif hasattr(self, "updateDijkstraGraph"):
        print("in dijkstra skip back")
        step = self.steps[self.currentStep]
        visitedNodes, currentNode, checkingEdges = step[0], step[1], step[2]
        self.updateDijkstraGraph(visitedNodes, currentNode, checkingEdges)
        self.updateDijkstraTable(step)

    elif hasattr(self, "updateSimplexGraph"):
        self.updateSimplexGraph()

    self.invalidMessage["text"] = ""
```

4.1.K

```

def stepBack(self):
    if self.isPaused and self.currentStep > 0:
        self.currentStep -= 1
        if hasattr(self, "updateChart"):
            numbers = self.steps[self.currentStep]
            indices = self.stepsDictionary[numbers]
            self.updateChart(numbers, indices)

    elif hasattr(self, "updatePrimGraph"):
        mstStep = self.steps[self.currentStep]
        self.updatePrimGraph(mstStep)
        self.visualiseText.insert(tk.END, "MST:" + ".join([f'{u}{v}'," for u,v in mstStep]) + self.visualiseText.see(tk.END)

    elif hasattr(self, "updateDijkstraGraph"):
        print("in dijkstra step back")
        step = self.steps[self.currentStep]
        if isinstance(step[0], set): # check not shortest path
            visitedNodes, currentNode, checkingEdges = step[0], step[1], step[2]
            self.updateDijkstraGraph(visitedNodes, currentNode, checkingEdges)
            self.updateDijkstraTable(step)

    elif hasattr(self, "updateSimplexGraph"):
        self.updateSimplexGraph()

```

4.1.L

```

def stepForward(self):
    if self.isPaused and self.currentStep < len(self.steps)-1:
        self.currentStep += 1
        if hasattr(self, "updateChart"):
            numbers = self.steps[self.currentStep]
            indices = self.stepsDictionary[numbers]
            self.updateChart(numbers, indices)

    elif hasattr(self, "updatePrimGraph"):
        mstStep = self.steps[self.currentStep]
        self.updatePrimGraph(mstStep)
        self.visualiseText.insert(tk.END, "MST:" + ".join([f'{u}{v}'," for u,v in mstStep]) + "\n self.visualiseText.see(tk.END)

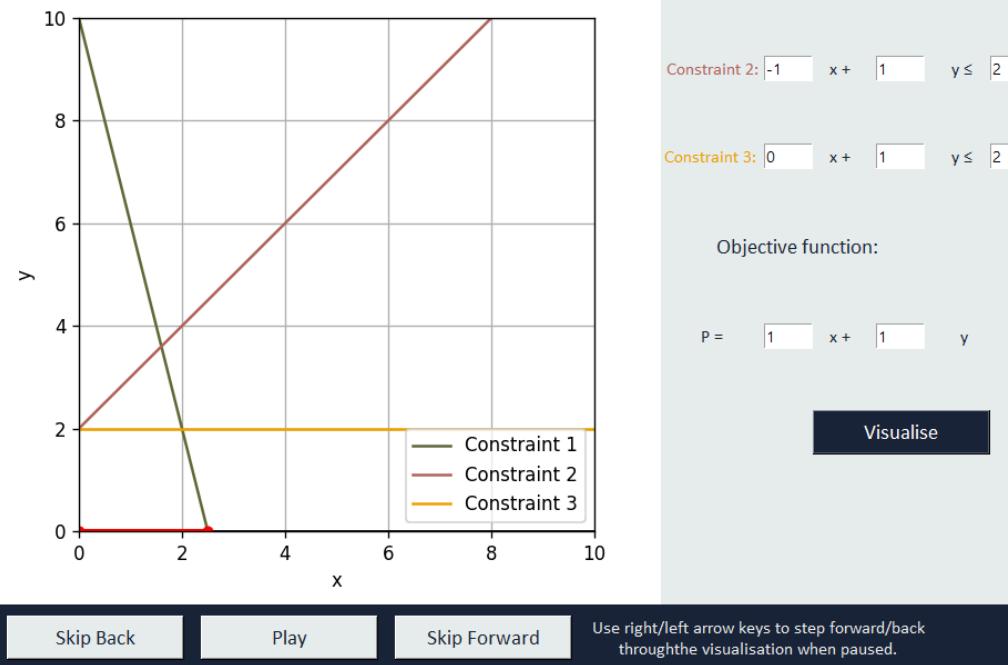
    elif hasattr(self, "updateDijkstraGraph"):
        print("in dijkstra step forward")
        step = self.steps[self.currentStep]
        if step[0] == "shortest path":
            pathEdges = [(step[1][i], step[1][i+1]) for i in range(len(step[1])-1)]
            self.updateDijkstraGraph([], checkingEdges=pathEdges)
            shortestPathText = " -> ".join(step[1])
            self.invalidMessage["text"] = f"Shortest path between {self.startVertexChoice.get()} a
else:
            visitedNodes, currentNode, checkingEdges = step[0], step[1], step[2]
            self.updateDijkstraGraph(visitedNodes, currentNode, checkingEdges)
            self.updateDijkstraTable(step)

    elif hasattr(self, "updateSimplexGraph"):
        step = self.steps[self.currentStep]
        self.updateSimplexGraph()

```

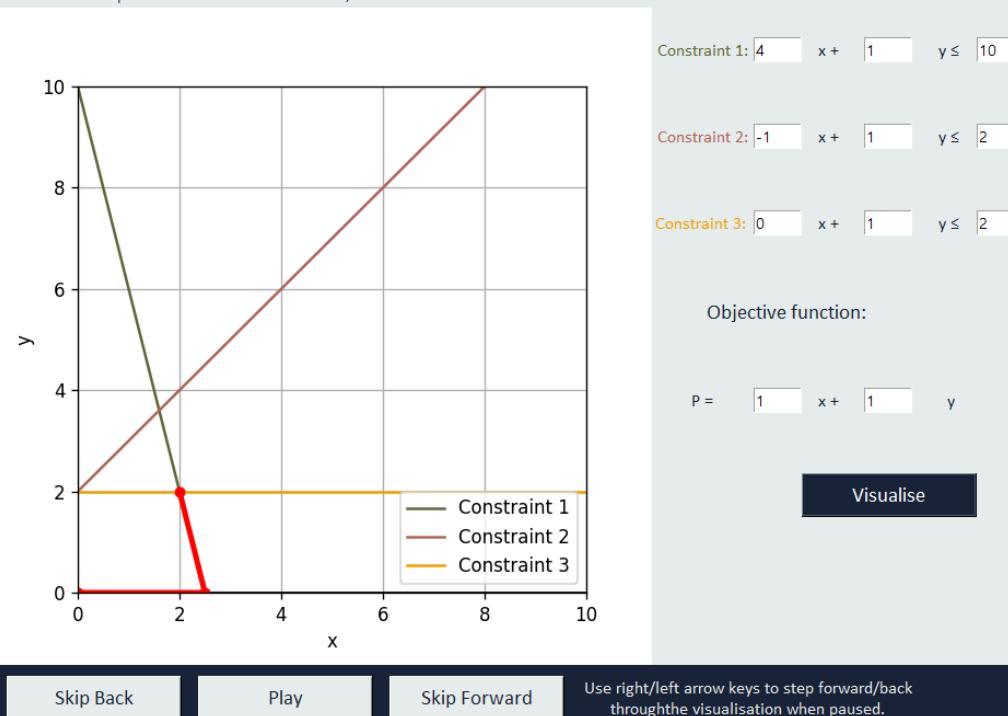
When I press Pause, the following is output:

4. Divide the pivot row by the value of the pivot.
5. Add or subtract multiple of the pivot row from the other rows so that the basis variable is 0.
6. Repeat until the objective row contains no negative entries.
7. Read off the optimal solution from the tableau,



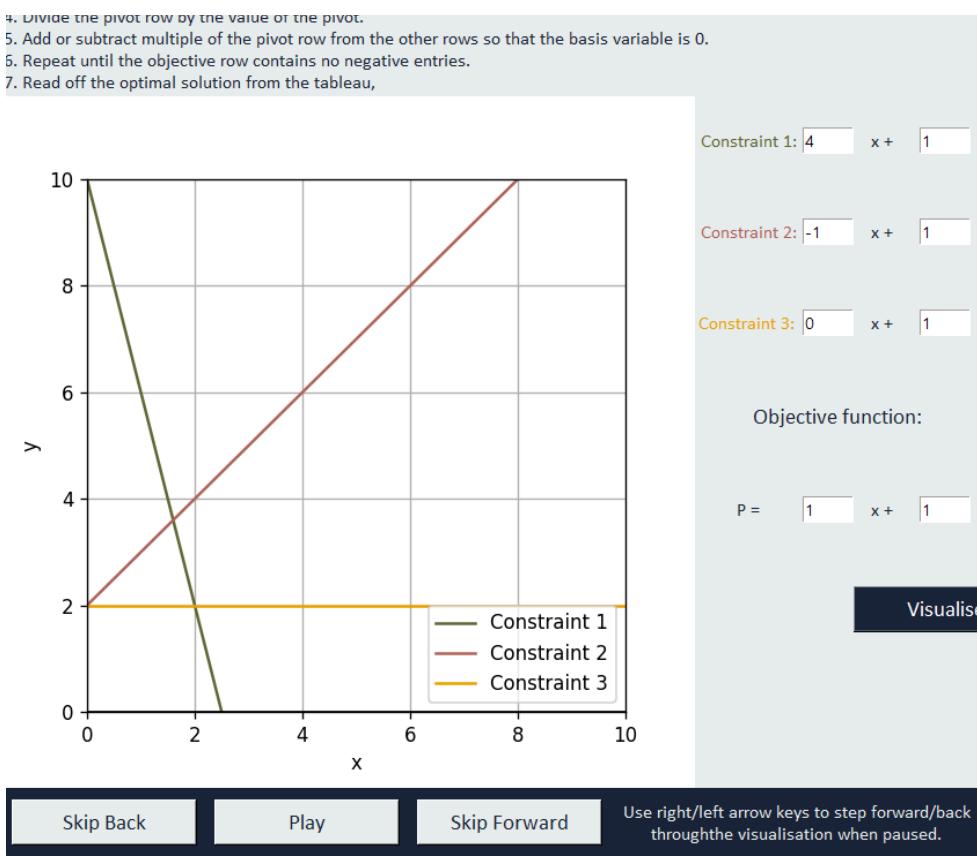
When I press the left and right arrow keys, the following are output:

5. Add or subtract multiple of the pivot row from the other rows so that the basis variable is 0.
6. Repeat until the objective row contains no negative entries.
7. Read off the optimal solution from the tableau,

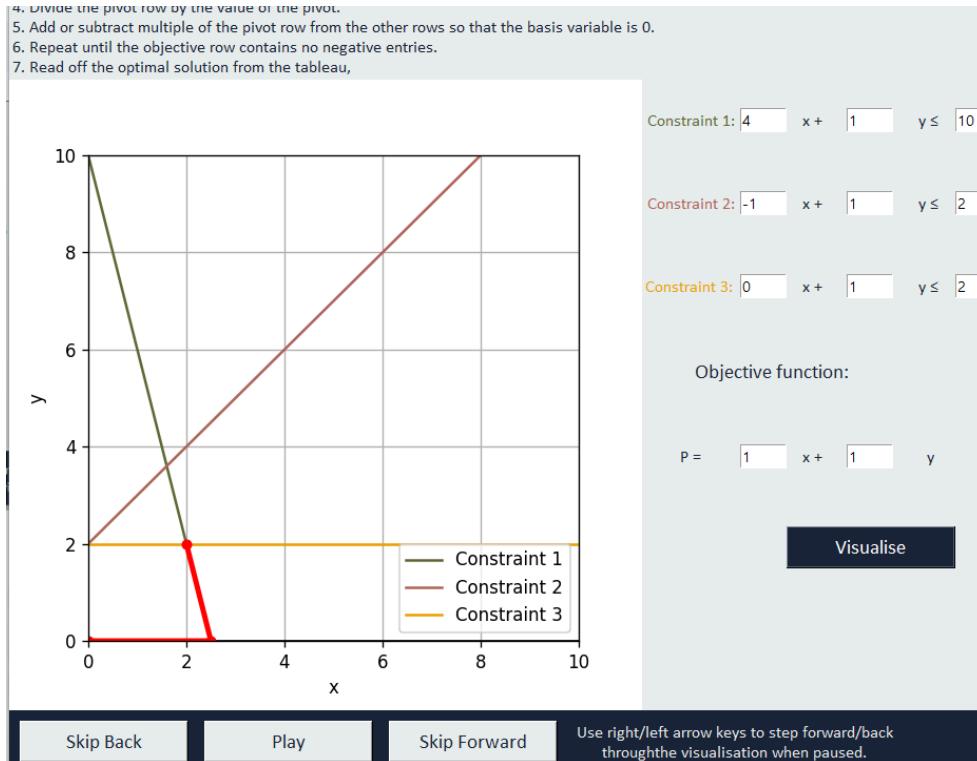


When I press Skip Back, the following is output:

4. Divide the pivot row by the value of the pivot.
5. Add or subtract multiple of the pivot row from the other rows so that the basis variable is 0.
6. Repeat until the objective row contains no negative entries.
7. Read off the optimal solution from the tableau,



And when I press Skip Forward, the following is output:



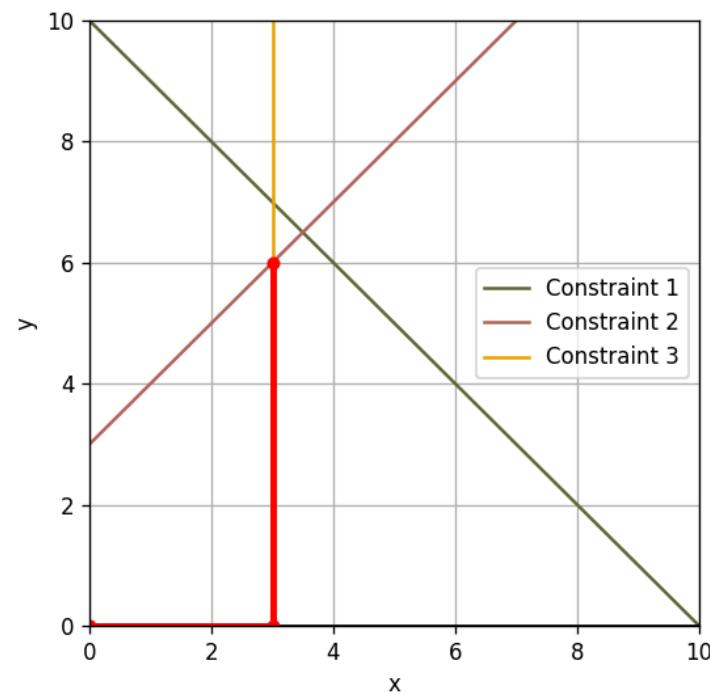
It seems that the final values are not being output as text on the graph when the final step is reached both when pressing Skip Forward and when pressing the right arrow key. To fix this, I added the following code:

4.1.M

```
elif hasattr(self, "updateSimplexGraph"):
    step = self.steps[self.currentStep]
    self.updateSimplexGraph()
    if step == self.steps[-1]:
        self.invalidMessage["text"] = f"Optimal solution P = {self.steps[-1][2]}\nx = {self.ste
```

Now when I pause it and press the right arrow key, until the final stage is reached, the following is output:

4. Divide the pivot row by the value of the pivot.
5. Add or subtract multiple of the pivot row from the other rows so that the basis variable is 0.
6. Repeat until the objective row contains no negative entries.
7. Read off the optimal solution from the tableau,



When I press Skip Back and then Skip Forward, the same is output.

All functionality for the Simplex algorithm visualisation page is now complete. As I have already implemented the refined GUI (because I have already used all the features before), all development is now complete in this iteration.

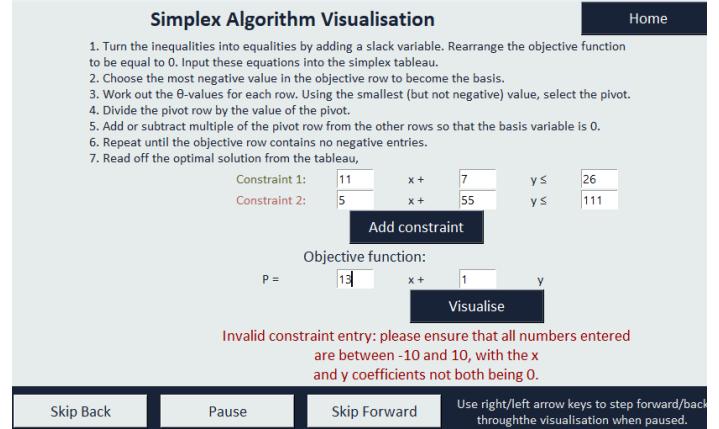
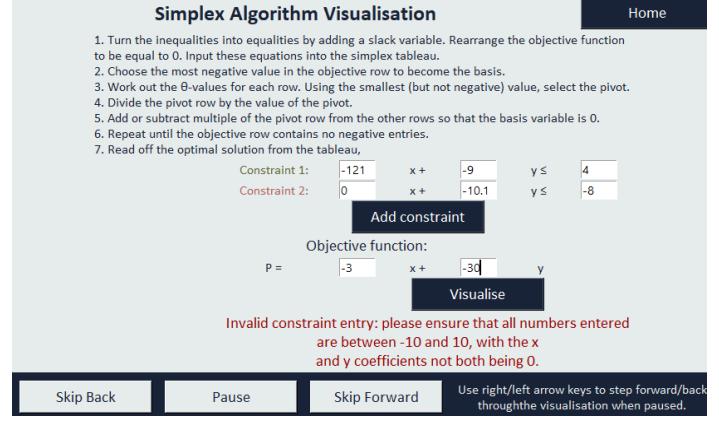
Now, I will test the code developed using the white box testing table specified in the SIMPLEX VISUALISATION PAGE design section and the white box testing table specified in the MENU section that is related to the Simplex visualisation.

TEST DATA	EXPECTED RESULT	JUSTIFICATION	CODE	ACTUAL OUTPUT
Press visualise without entering any constraints.	A 'Constraint invalid' error message is displayed.	Confirms that constraints must be entered by the user for the visualisation to begin.	4.1.e 4.1.g 4.1.k 4.1.l 4.1.m	<p>As expected.</p>
Press visualise without entering an objective function.	A 'Objective function invalid' error message is displayed.	Confirms that an objective function must be entered by the user for the visualisation to begin.	4.1.e 4.1.g	<p>As expected.</p>

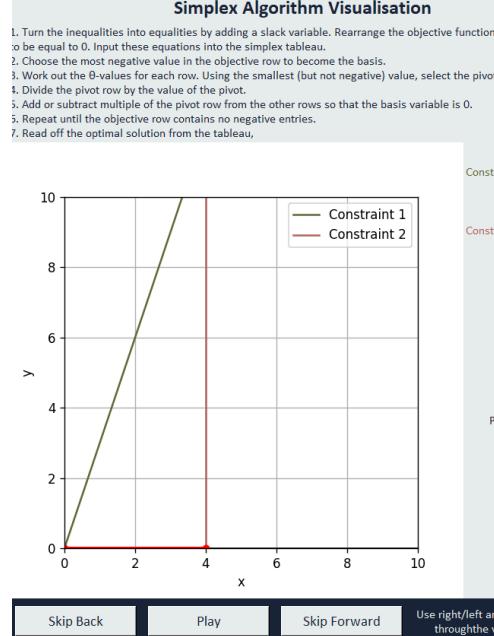
Press the Add Constraint button	A constraint entry field is displayed below the other 2 entry fields. The button is no longer displayed.	Confirms that the Add Constraint button works as intended and allows an extra constraint to be entered by the user.	4.1.e 4.1.g 4.1.f	<p>Simplex Algorithm Visualisation</p> <p>1. Turn the inequalities into equalities by adding a slack variable. Rearrange the objective function to be equal to 0. Input these equations into the simplex tableau. 2. Choose the most negative value in the objective row to become the basis. 3. Work out the θ-values for each row. Using the smallest (but not negative) value, select the pivot. 4. Divide the pivot row by the value of the pivot. 5. Add or subtract multiple of the pivot row from the other rows so that the basis variable is 0. 6. Repeat until the objective row contains no negative entries. 7. Read off the optimal solution from the tableau,</p> <table border="1" style="margin-left: auto; margin-right: auto;"> <tr> <td>Constraint 1:</td> <td>1</td> <td>x +</td> <td>3</td> <td>y ≤</td> <td>5</td> </tr> <tr> <td>Constraint 2:</td> <td>2</td> <td>x +</td> <td>-9</td> <td>y ≤</td> <td>4</td> </tr> <tr> <td>Constraint 3:</td> <td></td> <td>x +</td> <td></td> <td>y ≤</td> <td></td> </tr> </table> <p>Objective function: $P =$ <input type="text"/> x + <input type="text"/> y</p> <p>Visualise</p> <p>Invalid objective entry: please ensure that all numbers entered are between -10 and 0, with the x and y coefficients not both being 0.</p> <p>Skip Back Pause Skip Forward Use right/left arrow keys to step forward/back through the visualisation when paused.</p>	Constraint 1:	1	x +	3	y ≤	5	Constraint 2:	2	x +	-9	y ≤	4	Constraint 3:		x +		y ≤	
Constraint 1:	1	x +	3	y ≤	5																	
Constraint 2:	2	x +	-9	y ≤	4																	
Constraint 3:		x +		y ≤																		
Visualise with 1 valid constraint entered.	A 'Constraint invalid' error message is displayed.	Confirms that the user must enter at least 2 constraints for the visualisation to begin.	4.1.e 4.1.g	<p>Simplex Algorithm Visualisation</p> <p>1. Turn the inequalities into equalities by adding a slack variable. Rearrange the objective function to be equal to 0. Input these equations into the simplex tableau. 2. Choose the most negative value in the objective row to become the basis. 3. Work out the θ-values for each row. Using the smallest (but not negative) value, select the pivot. 4. Divide the pivot row by the value of the pivot. 5. Add or subtract multiple of the pivot row from the other rows so that the basis variable is 0. 6. Repeat until the objective row contains no negative entries. 7. Read off the optimal solution from the tableau,</p> <table border="1" style="margin-left: auto; margin-right: auto;"> <tr> <td>Constraint 1:</td> <td>4</td> <td>x +</td> <td>1</td> <td>y ≤</td> <td>8</td> </tr> <tr> <td>Constraint 2:</td> <td></td> <td>x +</td> <td></td> <td>y ≤</td> <td></td> </tr> </table> <p>Add constraint</p> <p>Objective function: $P =$ <input type="text"/> x + <input type="text"/> y</p> <p>Visualise</p> <p>Invalid constraint entry: please ensure that all numbers entered are between -10 and 10, with the x and y coefficients not both being 0.</p> <p>Skip Back Pause Skip Forward Use right/left arrow keys to step forward/back through the visualisation when paused.</p>	Constraint 1:	4	x +	1	y ≤	8	Constraint 2:		x +		y ≤							
Constraint 1:	4	x +	1	y ≤	8																	
Constraint 2:		x +		y ≤																		

Visualise with 2 valid constraints entered.	The visualisation of the Simplex algorithm begins correctly on the entered constraints.	Confirms that a valid input allows the visualisation to begin.	4.1.e 4.1.g 4.1.n 4.1.p 4.1.x 4.1.z 4.1.B 4.1.D 4.1.E 4.1.F 4.1.G	<p>Simplex Algorithm Visualisation</p> <p>Home</p> <p>1. Turn the inequalities into equalities by adding a slack variable. Rearrange the objective function to be equal to 0. Input these equations into the simplex tableau. 2. Choose the most negative value in the objective row to become the basis. 3. Work out the θ-values for each row. Using the smallest (but not negative) value, select the pivot. 4. Divide the pivot row by the value of the pivot. 5. Add or subtract multiple of the pivot row from the other rows so that the basis variable is 0. 6. Repeat until the objective row contains no negative entries. 7. Read off the optimal solution from the tableau.</p> <p>Constraint 1: $4x + y \leq 8$</p> <p>Constraint 2: $-1x + y \leq 0$</p> <p>Add constraint</p> <p>Objective function:</p> <p>$P = 2x + 2y$</p> <p>Visualise</p> <p>Optimal solution $P = 6.4$ $x = 1.6, y = 1.6$</p> <p>Skip Back Pause Skip Forward Use right/left arrow keys to step forward/back through the visualisation when paused.</p> <p>As expected.</p>
---	---	--	---	---

Visualise with 3 valid constraints entered.	The visualisation of the Simplex algorithm begins correctly on the entered constraints.	Confirms that a valid input allows the visualisation to begin.	4.1.e 4.1.g 4.1.n 4.1.p 4.1.x 4.1.z 4.1.B 4.1.D 4.1.E 4.1.F 4.1.G	Simplex Algorithm Visualisation Home 1. Turn the inequalities into equalities by adding a slack variable. Rearrange the objective function to be equal to 0. Input these equations into the simplex tableau. 2. Choose the most negative value in the objective row to become the basis. 3. Work out the θ -values for each row. Using the smallest (but not negative) value, select the pivot. 4. Divide the pivot row by the value of the pivot. 5. Add or subtract multiple of the pivot row from the other rows so that the basis variable is 0. 6. Repeat until the objective row contains no negative entries. 7. Read off the optimal solution from the tableau, Constraint 1: $-1 \leq x + 2y \leq 8$ Constraint 2: $3 \leq x + 0y \leq 9$ Constraint 3: $0 \leq x + 1y \leq 3$ Objective function: $P = 1x + 1y$ Visualise Optimal solution $P = 6.0$ $x = 3.0, y = 3.0$ Skip Back Pause Skip Forward Use right/left arrow keys to step forward/back through the visualisation when paused.
---	---	--	---	---

Visualise with coefficients being greater than 10.	A 'Constraint invalid' error message is displayed.	Confirms that the coefficients are checked to be within the range -10 to 10.	4.1.e 4.1.g	 <p>Simplex Algorithm Visualisation</p> <p>1. Turn the inequalities into equalities by adding a slack variable. Rearrange the objective function to be equal to 0. Input these equations into the simplex tableau. 2. Choose the most negative value in the objective row to become the basis. 3. Work out the θ-values for each row. Using the smallest (but not negative) value, select the pivot. 4. Divide the pivot row by the value of the pivot. 5. Add or subtract multiple of the pivot row from the other rows so that the basis variable is 0. 6. Repeat until the objective row contains no negative entries. 7. Read off the optimal solution from the tableau.</p> <p>Constraint 1: $11 \quad x + \quad 7 \quad y \leq 26$ Constraint 2: $5 \quad x + \quad 55 \quad y \leq 111$</p> <p>Add constraint</p> <p>Objective function: $P = 13 \quad x + \quad 1 \quad y$</p> <p>Visualise</p> <p>Invalid constraint entry: please ensure that all numbers entered are between -10 and 10, with the x and y coefficients not both being 0.</p> <p>Skip Back Pause Skip Forward Use right/left arrow keys to step forward/back through the visualisation when paused.</p>
Visualise with coefficients being less than -10.	A 'Constraint invalid' error message is displayed.	Confirms that the coefficients are checked to be within the range -10 to 10.	4.1.e 4.1.g	 <p>Simplex Algorithm Visualisation</p> <p>1. Turn the inequalities into equalities by adding a slack variable. Rearrange the objective function to be equal to 0. Input these equations into the simplex tableau. 2. Choose the most negative value in the objective row to become the basis. 3. Work out the θ-values for each row. Using the smallest (but not negative) value, select the pivot. 4. Divide the pivot row by the value of the pivot. 5. Add or subtract multiple of the pivot row from the other rows so that the basis variable is 0. 6. Repeat until the objective row contains no negative entries. 7. Read off the optimal solution from the tableau.</p> <p>Constraint 1: $-121 \quad x + \quad -9 \quad y \leq 4$ Constraint 2: $0 \quad x + \quad -10.1 \quad y \leq -8$</p> <p>Add constraint</p> <p>Objective function: $P = -3 \quad x + \quad -30 \quad y$</p> <p>Visualise</p> <p>Invalid constraint entry: please ensure that all numbers entered are between -10 and 10, with the x and y coefficients not both being 0.</p> <p>Skip Back Pause Skip Forward Use right/left arrow keys to step forward/back through the visualisation when paused.</p>

Visualise with both the x and y coefficients being 0.	A 'Constraint invalid' error message is displayed.	Confirms that the x and y coefficient checks occur correctly to ensure that the visualisation does not error.	4.1.e 4.1.g	<p>Simplex Algorithm Visualisation</p> <p>1. Turn the inequalities into equalities by adding a slack variable. Rearrange the objective function to be equal to 0. Input these equations into the simplex tableau. 2. Choose the most negative value in the objective row to become the basis. 3. Work out the θ-values for each row. Using the smallest (but not negative) value, select the pivot. 4. Divide the pivot row by the value of the pivot. 5. Add or subtract multiple of the pivot row from the other rows so that the basis variable is 0. 6. Repeat until the objective row contains no negative entries. 7. Read off the optimal solution from the tableau.</p> <p>Constraint 1: $4 \quad x + \quad -9 \quad y \leq \quad 4$ Constraint 2: $0 \quad x + \quad 0 \quad y \leq \quad 9$</p> <p>Add constraint</p> <p>Objective function: $P = 1 \quad x + \quad 1 \quad y$</p> <p>Visualise</p> <p>Invalid constraint entry: please ensure that all numbers entered are between -10 and 10, with the x and y coefficients not both being 0.</p> <p>Skip Back Pause Skip Forward Visualise Use right/left arrow keys to step forward/back through the visualisation when paused.</p>																		
Press the Home button.	The Simplex Visualisation page is closed and the Home page is opened.	This ensures that the 'Home' button works correctly and that the user can go back to the Home page to either log out or go to the other visualisation/quiz pages.	4.1.a	<p>Algorithm's Simulator and Teaching Tool</p> <p>Bubble Sort Prim's Algorithm Dijkstra's Algorithm Simplex Algorithm Quiz</p> <p>Quiz Statistics</p> <table border="1"> <thead> <tr> <th>Algorithm</th> <th>Correct</th> <th>Incorrect</th> </tr> </thead> <tbody> <tr> <td>Bubble Sort</td> <td>~2</td> <td>~6</td> </tr> <tr> <td>Prim's</td> <td>~1</td> <td>~1</td> </tr> <tr> <td>Dijkstra's Algorithms</td> <td>~5</td> <td>~0</td> </tr> <tr> <td>Simplex</td> <td>~3</td> <td>~0</td> </tr> <tr> <td>Total</td> <td>~11</td> <td>~10</td> </tr> </tbody> </table> <p>Logout</p>	Algorithm	Correct	Incorrect	Bubble Sort	~2	~6	Prim's	~1	~1	Dijkstra's Algorithms	~5	~0	Simplex	~3	~0	Total	~11	~10
Algorithm	Correct	Incorrect																				
Bubble Sort	~2	~6																				
Prim's	~1	~1																				
Dijkstra's Algorithms	~5	~0																				
Simplex	~3	~0																				
Total	~11	~10																				

Press the 'Pause' button for the Simplex visualisation	The visualisation stops, the button changes from 'Pause' to 'Play' (or vice versa).	Confirms that the 'Pause' button works as intended for the Simplex visualisation.	4.1.e 4.1.g 4.1.t 4.1.u 4.1.x 4.1.B 4.1.D 4.1.E 4.1.F 4.1.G 4.1.H	 <p>Simplex Algorithm Visualisation</p> <p>1. Turn the inequalities into equalities by adding a slack variable. Rearrange the objective function to be equal to 0. Input these equations into the simplex tableau. 2. Choose the most negative value in the objective row to become the basis. 3. Work out the 0-values for each row. Using the smallest (but not negative) value, select the pivot. 4. Divide the pivot row by the value of the pivot. 5. Add or subtract multiple of the pivot row from the other rows so that the basis variable is 0. 6. Repeat until the objective row contains no negative entries. 7. Read off the optimal solution from the tableau.</p> <p>Constraint 1: $-3x + y \leq 0$</p> <p>Constraint 2: $x + 0y \leq 4$</p> <p>Add constraint</p> <p>Objective function:</p> <p>$P = 1x + 1y$</p> <p>Visualise</p> <p>Skip Back Play Skip Forward Use right/left arrow keys to step forward/back through the visualisation when paused.</p> <p>As expected.</p>
--	---	---	---	--

Press the 'Skip Forward' button for the Simplex visualisation	The final path from the origin to the optimal vertex is highlighted on the graph and the values of x y and P are displayed as text.	Confirms that the 'Skip Forward' button works as intended for the Simplex visualisation.	4.1.e 4.1.g 4.1.t 4.1.u 4.1.x 4.1.B 4.1.D 4.1.E 4.1.F 4.1.G 4.1.I 4.1.M	<div style="background-color: #f0f0f0; padding: 10px;"> <p>Simplex Algorithm Visualisation</p> <p>Home</p> <p>Turn the inequalities into equalities by adding a slack variable. Rearrange the objective function P to be equal to 0. Input these equations into the simplex tableau.</p> <p>Choose the most negative value in the objective row to become the basis.</p> <ol style="list-style-type: none"> Work out the θ-values for each row. Using the smallest (but not negative) value, select the pivot. Divide the pivot row by the value of the pivot. Add or subtract multiple of the pivot row from the other rows so that the basis variable is 0. Repeat until the objective row contains no negative entries. Read off the optimal solution from the tableau. <p>Constraint 1: $-3x + y \leq 0$</p> <p>Constraint 2: $x + 0y \leq 4$</p> <p>Add constraint</p> <p>Objective function:</p> <p>$P = 1x + 1y$</p> <p>Visualise</p> <p>Optimal solution $P = 16.0$ $x = 4.0, y = 12.0$</p> <p>Skip Back Play Skip Forward Use right/left arrow keys to step forward/back through the visualisation when paused.</p> <p>As expected.</p> </div>
---	---	--	--	--

<p>Press the 'Skip Back' button for the Simplex visualisation</p>	<p>The blank graph is displayed with just the constraints shown. The text is blank.</p>	<p>Confirms that the 'Skip Back' button works as intended for the Simplex visualisation.</p>	<p>4.1.e 4.1.g 4.1.t 4.1.u 4.1.x 4.1.B 4.1.D 4.1.E 4.1.F 4.1.G 4.1.J</p>	<p>Simplex Algorithm Visualisation</p> <p>Home</p> <p>1. Turn the inequalities into equalities by adding a slack variable. Rearrange the objective function to be equal to 0. Input these equations into the simplex tableau. 2. Choose the most negative value in the objective row to become the basis. 3. Work out the θ-values for each row. Using the smallest (but not negative) value, select the pivot. 4. Divide the pivot row by the value of the pivot. 5. Add or subtract multiple of the pivot row from the other rows so that the basis variable is 0. 6. Repeat until the objective row contains no negative entries. 7. Read off the optimal solution from the tableau.</p> <p>Constraint 1: $-3x + y \leq 0$</p> <p>Constraint 2: $x + 0y \leq 4$</p> <p>Add constraint</p> <p>Objective function:</p> <p>$P = 1x + 1y$</p> <p>Visualise</p> <p>Skip Back Play Skip Forward</p> <p>Use right/left arrow keys to step forward/back through the visualisation when paused.</p> <p>As expected.</p>
<p>Use the right and left arrow keys for the Simplex visualisation</p>	<p>The visualisation goes back/forward one step in the visualisation on the graph. If the final stage is reached, the values of P x and y are displayed as text.</p>	<p>Confirms that the arrow key functionality works as intended for the Simplex visualisation.</p>	<p>4.1.e 4.1.g 4.1.t 4.1.u 4.1.x 4.1.B 4.1.D 4.1.E 4.1.F</p>	<p>When starting at the stage shown above - the blank graph:</p>

			4.1.G 4.1.K 4.1.L	<p>Simplex Algorithm Visualisation</p> <p>Turn the inequalities into equalities by adding a slack variable. Rearrange the objective function to be equal to 0. Input these equations into the simplex tableau.</p> <p>Choose the most negative value in the objective row to become the basis.</p> <p>Work out the B-values for each row. Using the smallest (but not negative) value, select the pivot.</p> <p>Divide the pivot row by the value of the pivot.</p> <p>Add or subtract multiple of the pivot row from the other rows so that the basis variable is 0.</p> <p>Repeat until the objective row contains no negative entries.</p> <p>Read off the optimal solution from the tableau.</p> <p>Constraint 1: $-3x + 1y \leq 0$</p> <p>Constraint 2: $1x + 0y \leq 4$</p> <p>Add constraint</p> <p>Objective function:</p> <p>$P = 1x + 1y$</p> <p>Visualise</p> <p>Skip Back Play Skip Forward</p> <p>Use right/left arrow keys to step forward/back through the visualisation when paused.</p> <p>As expected.</p>
--	--	--	--	--

All white box tests have passed successfully. Therefore, development and testing in Prototype 4 Iteration 1 is complete.

ITERATION 1 - DEVELOPER REVIEW

I think that this iteration went quite well. Having already completed the visualisation pages for the previous 3 algorithms made setting up the GUI and using the matplotlib library much easier. However, I found the Simplex algorithm quite complicated to program as it is quite different to the other algorithms, due to it being focused on optimisation and using Gaussian elimination row reduction - despite having already learnt how to apply it by hand. Therefore, debugging became quite tricky when the visualisation was stopping before reaching the final optimal stage. However, I have now reached the milestone of finishing the development of all the algorithm visualisations in this system.

ITERATION 2 - QUIZ PAGE: BUBBLE SORT AND SIMPLEX

In this iteration, I will set up the GUI for the initial quiz page and complete the functionality of the Bubble Sort and Simplex questions. To begin, I set up the following structure for the Quiz class and I wrote the code to create the widgets that will remain on the window for every question - the title and buttons.

4.2.a

```
# imports specific to the Quiz class
from tkinter import *
import tkinter as tk
import sqlite3 # for writing scores to the database
from NEA_utilities import closeOpen, fontLarge, fontMedium, fontSmall

class Quiz():
    def __init__(self, master, pUsername):
        self.username = pUsername
        self.currentQType = None

    def initialQuizWidgets(self):
        self.master.configure(bg="#eaebed")
        self.title = Label(self.master, text="Quiz". font=fontLarge, bg="#eaebed", fg="#1b263b")
        self.title.grid(row=0, column=0, columnspan=10)
        Button(self.master, text="Home", command=self.openHome, bg="#1b263b", fg="white", font=fontMedium, width=15).grid(row=0, column=10, columnspan=2)

        self.bubbleSortButton = Button(self.master, text="Bubble Sort", command=lambda: self.newType("Bubble Sort"), bg="#1b263b", fg="white", font=fontMedium)
        self.bubbleSortButton.grid(row=1, column=0, columnspan=3)

        self.primButton = Button(self.master, text="Prim's", command=lambda: self.newType("Prim"), bg="#1b263b", fg="white", font=fontMedium)
        self.primButton.grid(row=1, column=3, columnspan=3)

        self.dijkstraButton = Button(self.master, text="Dijkstra's", command=lambda: self.newType("Dijkstra"), bg="#1b263b", fg="white", font=fontMedium)
        self.dijkstraButton.grid(row=1, column=6, columnspan=3)

        self.simplexButton = Button(self.master, text="Simplex", command=lambda: self.newType("Simplex"), bg="#1b263b", fg="white", font=fontMedium)
        self.simplexButton.grid(row=1, column=9, columnspan=3)
```

4.2.b

4.2.c

```

def bubbleSortQuizWidgets(self):
    # widgets for bubbleSort frame
    print("in bubbleSort quiz widgets")

def primQuizWidgets(self):
    # widgets for prim frame
    print("in prim quiz widgets")

def dijkstraQuizWidgets(self):
    # widgets for dijkstra frame
    print("in dijkstra quiz widgets")

def simplexQuizWidgets(self):
    # widgets for simplex frame
    print("in simplex quiz widgets")

def bubbleSortDataset(self):
    # 10 random integers between 1 and 50 inclusive
    print("in create bubbleSort dataset")

def randomGraph(self):
    # connected graph of 6-8 vertices labelled A, B, C...
    # weights between 1 and 50
    print("in create random graph")

def simplexConstraints(self):
    # 3 constraints and objective function
    # float coefficients between -10 and 10, x and y not both 0
    print("in create simplex constraints")

def check(self):
    # calls updateScore
    print("in check")

def newQuestion(self, qType):
    # forget current widgets, create new ones for qType
    print(f"in new question for: {qType}")

def newType(self, newQType):
    # qType = question type ---> bubble Sort, prim, dijkstra, simplex
    # forget currentQType ---> None initially
    print(f"current question type: {self.currentQType}, new question type: {newQType}")

def updateScore(self, answer):
    # answer = Correct or Incorrect
    print("in update score")

def openHome(self):
    closeOpen(self.master, "home")

```

Then I added the following lines to allow me to test the Quiz page on its own and to allow the initial widgets to be created:

4.2.d

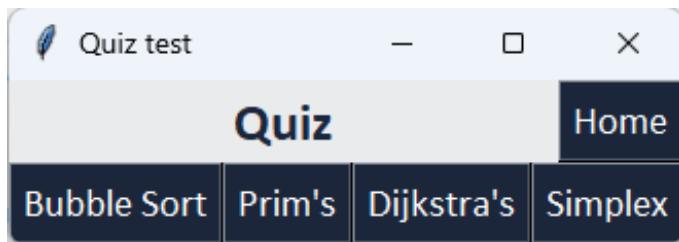
```

self.initialQuizWidgets()

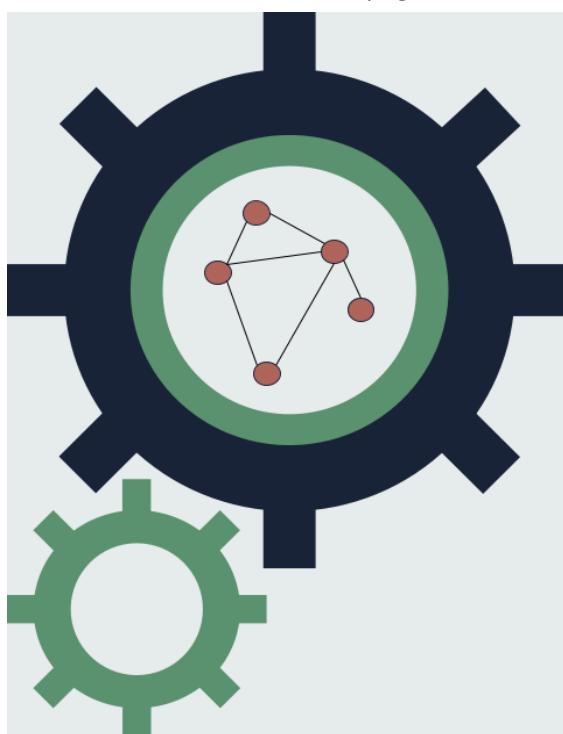
if __name__ == "__main__":
    root = tk.Tk()
    root.title("Quiz test")
    #root.geometry("800x595")
    graph_input = Quiz(root, "HelloWord!!!") #HelloWord!!! username for testing purposes
    root.mainloop()

```

When I run the code, the following is output:



As discussed in the Quiz Design section, the initial page is quite plain as it only has these buttons. A possibility to make this page more visually appealing could be to also display the logo that I created for the Teacher account home page:



Another idea would be to display the quiz statistics again - as done on the Home page. However, this is much more complicated to implement than just displaying the logo image. Therefore, displaying the logo image below the buttons for each of the algorithm questions is the best idea. For this, I wrote the following code in the initial widgets method:

4.2.e

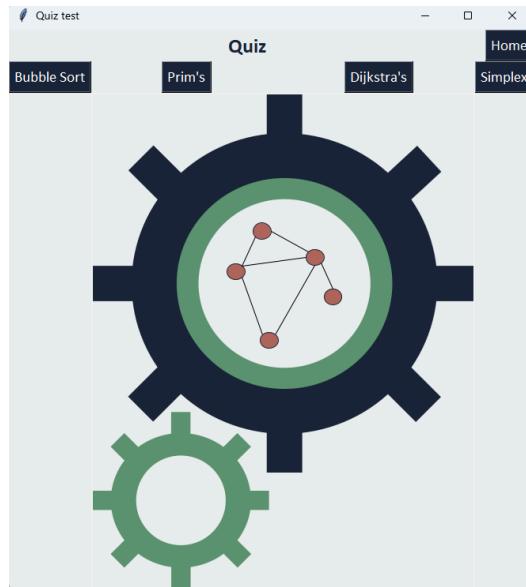
```
image = Image.open("NEA_logo_image.png")
image = image.resize((481, 626))
self.logoImg = ImageTk.PhotoImage(image) # creates tkinter widget
self.imageLabel = Label(self.master, image=self.logoImg)
self.imageLabel.grid(row=2, column=3, columnspan=6, rowspan=8)
```

And I added the following line to import PIL. This is because it would allow me to resize the image:

4.2.f

```
from PIL import Image, ImageTk # for logo image editing
```

Now when the code is run, the following is output:



This is much more visually appealing but a final edit I would make is to add padding and increase the width of the buttons so that there is less empty space at the top. For the home button, I edited the widget to the following:

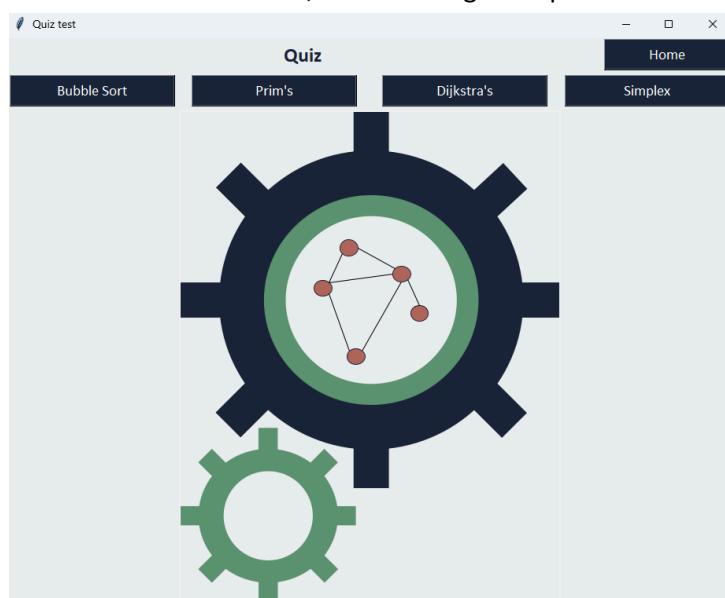
4.2.g

```
bg="#1b263b", fg="white", font=fontMedium, width=15  
padx=5, pady=1)
```

For the other buttons, I edited them to all have the same changes to the width and padding:

```
bg="#1b263b", fg="white", font=fontMedium, width=20)  
padx=5, pady=5)
```

When I now run the code, the following is output:



I think that this is much better than before. Now, I am going to implement the methods for the Bubble Sort quiz questions. Firstly, the Bubble Sort section requires an unsorted dataset. Therefore, I will first implement the bubbleSortDataset method:

4.2.h

```
def bubbleSortDataset(self):
    # 10 random integers between 1 and 50 inclusive
    print("in create bubbleSort dataset")
    self.numbers = []
    for i in range(10):
        number = randint(1, 50)
        self.numbers.append(number)
```

Then, I wrote the following code in the newType method to display the widgets of the Bubble Sort method and create the dataset:

4.2.i

```
def newType(self, newQType):
    # qType = question type ---> bubble Sort, prim, dijkstra, simplex
    # forget currentQType ---> None initially
    print(f"current question type: {self.currentQType}, new question type: {newQType}")
    self.imageLabel.grid_forget()
    # change current q type button background

    if newQType == "Bubble Sort":
        self.currentQType = "Bubble Sort"
        self.bubbleSortDataset()
        self.bubbleSortQuizWidgets()
```

Note that I will change the currentQType background once I have completed the functionality for Bubble Sort quiz questions. Then, I wrote the following code in the bubbleSortQuizWidgets method to create all the widgets needed for Bubble Sort questions:

4.2.j

```
def bubbleSortQuizWidgets(self):
    # widgets for bubbleSort frame
    print("in bubbleSort quiz widgets")

    self.bubbleSortButton["bg"] = "#b2675e"

    self.bubbleSortQuizFrame = Frame(self.master, bg="#eaebed")
    self.bubbleSortQuizFrame.grid(row=2, column=0, columnspan=12, rowspan=8)

    Label(self.bubbleSortQuizFrame,
          text="1. Start at the first item in the list.\n"
               "2. Compare the current item with the next item.\n"
               "3. If the two items are in the wrong position, swap them.\n"
               "4. Move up one item to the next time in the list.\n"
               "5. Repeat from step 3 until all the unsorted items have been compared.\n"
               "6. If any items were swapped, repeat from step 1. Otherwise, the algorithm is complete.",
          font=fontSmall,
          fg="#1b263b",
          bg="#eaebed",
          justify="left").grid(row=2, column=0, columnspan=12, sticky="w", padx=10, pady=5)

    Label(self.bubbleSortQuizFrame, text="Please sort the following unsorted dataset in ascending order and input the number of passes it took:",
          font=fontMedium, fg="#1b263b", bg="#eaebed").grid(row=4, column=0, columnspan=12, padx=2)

    self.numChoices = []
    iCounter = 1 # column index counter
    for number in self.numbers:
        numText= str(number)
        Label(self.bubbleSortQuizFrame, text=numText, font=fontMedium, fg="#1b263b", bg="#eaebed").grid(row=5, column=iCounter, padx=1, pady=1)

        var = IntVar()
        var.set(0) # default value
        self.numChoices.append(var)

        dropdown = OptionMenu(self.bubbleSortQuizFrame, var, *self.numbers)
        dropdown.grid(row=6, column=iCounter, padx=1, pady=1)

    iCounter += 1
```

```

Label(self.bubbleSortQuizFrame, text="Number of passes = ", font=fontMedium, fg="#1b263b", bg="#eaebcd").grid(row=7, column=0, columnSpan=3)

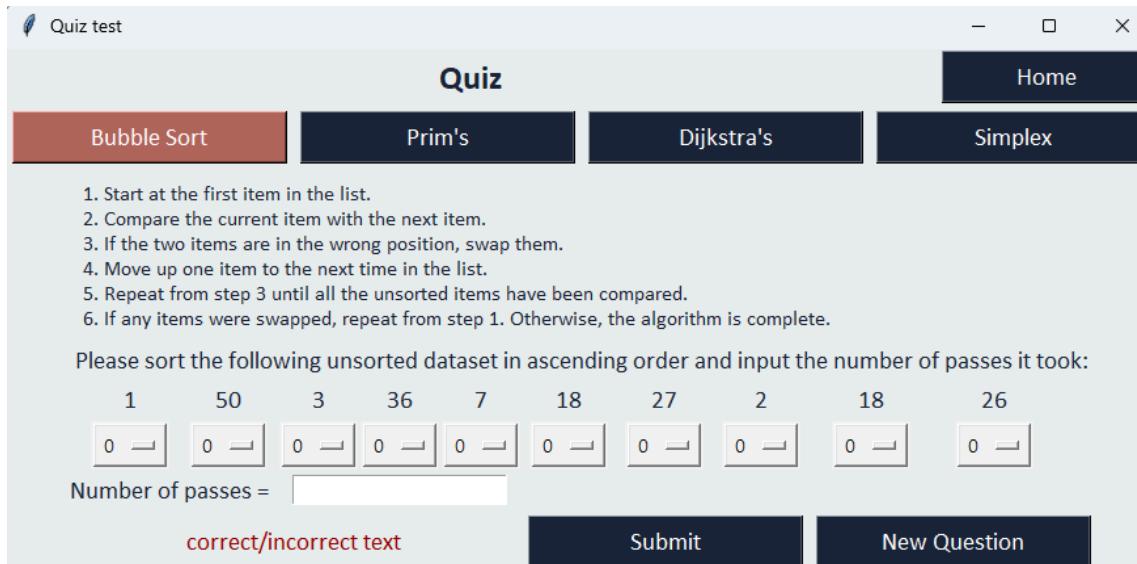
self.passesInput = StringVar()
Entry(self.bubbleSortQuizFrame, textvariable=self.passesInput).grid(row=7, column=3, columnSpan=3, padx=1, pady=1)

self.answerText = Label(self.bubbleSortQuizFrame, text="correct/incorrect text", font=fontMedium, fg="#990000", bg="#eaebcd")
self.answerText.grid(row=9, column=0, columnSpan=6)

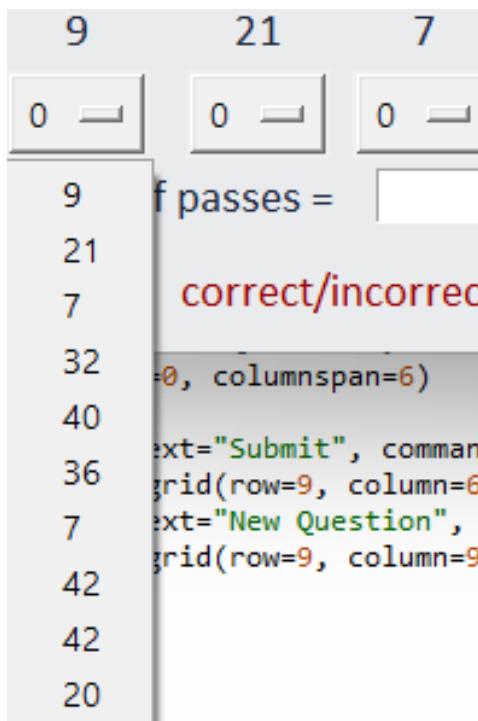
Button(self.bubbleSortQuizFrame, text="Submit", command=self.check, bg="#1b263b", fg="white",
       font=fontMedium, width=20).grid(row=9, column=6, columnSpan=3, padx=5, pady=5)
Button(self.bubbleSortQuizFrame, text="New Question", command=lambda: self.newQuestion("Bubble Sort"), bg="#1b263b", fg="white",
       font=fontMedium, width=20).grid(row=9, column=9, columnSpan=3, padx=5, pady=5)

```

For the dropdown menus, I opted to set var to be 0 initially because all the integers in numbers are between 1 and 50. Therefore, this would allow me to check if the user has actually answered the question. When I run the code and press the Bubble Sort button, the following is output:



When I press the dropdown menu for a different dataset, the following is output:



This shows that the functionality for creating a new dataset works correctly and that the dropdown menus work correctly as well. Now I am going to implement the check method. Because I have already developed the methods to work out the sorted dataset, MST, shortest path and optimal values, having Quiz inherit from all the classes was my initial thought - and the one I described in the Design section. However, now that I have coded all these classes, I realised that there may be an issue as each of the constructors for the classes will display the widgets for each of the visualisation pages. I don't want this in my Quiz class as I already have specific designs and widgets for this page. I researched how I could do this and I found out that I could use the concept of composition³⁰ instead of inheritance.

Composition is the concept of a class using the functionality of another class but without being directly dependent on its structure. In this system, would be where Quiz is able to access the methods of another class, like BubbleSort's steps method, without inheriting unwanted attributes and methods like the GUI components. The advantage of this is that it reduces tight coupling - where classes are too dependent on each other's internal details, which can lead to unnecessary complexity.

Therefore, I changed the Quiz constructor method to the following:

4.2.k

```
from bubble_sort_page import BubbleSort
from prim_page import Prim
from dijkstra_page import Dijkstra
from simplex_page import Simplex

class Quiz(BubbleSort, Prim, Dijkstra, Simplex):
    def __init__(self, master, pUsername):
        self.master = master
        self.username = pUsername
        self.currentQType = None

        # instances of algorithm classes - composition:
        self.bubbleSort = BubbleSort(master, pUsername)
        self.prim = Prim(master, pUsername)
        self.dijkstra = Dijkstra(master, pUsername)
        self.simplex = Simplex(master, pUsername)

        # hide algorithm frames to avoid interference
        self.bubbleSort.bubbleSortFrame.grid_forget()
        self.prim.primFrame.grid_forget()
        self.dijkstra.dijkstraFrame.grid_forget()
        self.simplex.simplexFrames.grid_forget()

    self.initialQuizWidgets()
```

And for the check method, I wrote the following code:

4.2.l

```

def check(self):
    # calls updateScore
    print("in check")
    if self.currentQType == "Bubble Sort":
        self.bubbleSort.numbers = self.quizNumbers[:] # copy
        self.bubbleSort.bubbleSortSteps()
        correctOrder = list(self.bubbleSort.steps[-1]) # final sorted order
        correctPasses = self.bubbleSort.numberPasses

        userOrder = [var.get() for var in self.numChoices]
        userPasses = int(self.passesInput.get()) if self.passesInput.get().isdigit() else -1

        if userOrder == correctOrder and userPasses == correctPasses:
            self.answerText["text"] = "Correct :)"
            self.answerText["fg"] = "#008000"
            self.updateScore("Correct")
        elif userOrder != correctOrder:
            self.answerText["text"] = "Incorrect order :("
            self.answerText["fg"] = "#990000"
            self.updateScore("Incorrect")
        else: # passes wrong
            self.answerText["text"] = f"Incorrect number of passes. Correct answer: {correctPasses}"
            self.answerText["fg"] = "#990000"
            self.updateScore("Incorrect")
    
```

As seen above, I added an extra attribute to BubbleSort for the number of passes. For this, I initialised numberPasses as 0 in the BubbleSort constructor method:

```
self.numberPasses = 0
```

And in the bubbleSortSteps method, I added the following code:

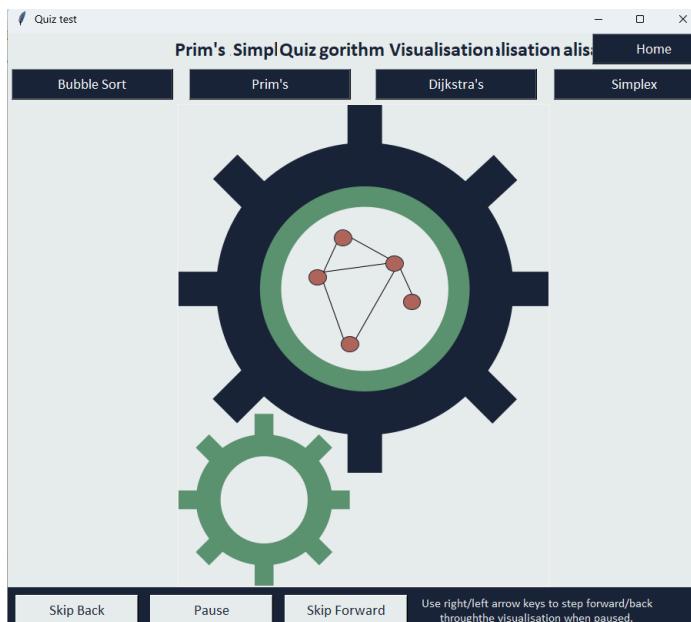
4.2.m

```

if self.sortOrder.get() == "Ascending":
    while swapped:
        swapped = False
        for j in range(n-1):
            if self.numbers[j] > self.numbers[j+1]:
                temp= self.numbers[j]
                self.numbers[j] = self.numbers[j+1]
                self.numbers[j+1] = temp
                swapped = True
            #self.steps.append(self.numbers[:])
            # note must be tuple because lists are mutable so they are unhashable
            stepsDict[tuple(self.numbers)] = [j, j+1]
    if swapped:
        self.numberPasses += 1

```

When I run the code, the following is output:



From this, it is clear that although the main widgets each of the classes is not visible, the titles for each of the visualisation pages and the menu are still visible. In order to fix this, I decided to have

each class have a flag for whether or not to display the menu and I will change any widgets in the master for each class to be in its specific frame as this allows `grid_forget()` to be applied to them. The code in each of the constructor classes will have these modification made (note that the following screenshot just shows in for BubbleSort but it is the same for each of the others):

4.2.n

```
class BubbleSort(Menu):
    def __init__(self, master, pUsername, show_menu=True)
        Menu.__init__(self, master, 12, 19)
        self.numbers = []
        self.userEntries = []

        self.username = pUsername

        self.master.configure(bg="#eaebed")
        self.master.geometry("1105x905") # required here

        self.sortOrder = StringVar(value="Ascending")

        self.invalidMessage = None

        self.fig = None
        self.ax = None
        self.canvas = None

        self.visualiseText = None

        self.numberPasses = 0

        self.bubbleSortWidgets()
        if show_menu:
            self.menuWidgets()
```

And in the widgets methods, I changed each to the following (but specific to the algorithm):

4.2.o

```
# BUBBLE SORT FRAME:
self.bubbleSortFrame = Frame(self.master, bg="#eaebed")
self.bubbleSortFrame.grid(row=0, column=0, columnspan=19, rowspan=12)

# TITLE / TOP SECTION:
self.title = Label(self.bubbleSortFrame, text="Bubble Sort Algorithm V:")
self.title.grid(row=0, column=0, columnspan=17)

Button(self.bubbleSortFrame, text="Home", command=self.openHome, bg="#:
```

When I check that the BubbleSort page still works, the following is output:



This is as expected so the changes made still allow it to work correctly.

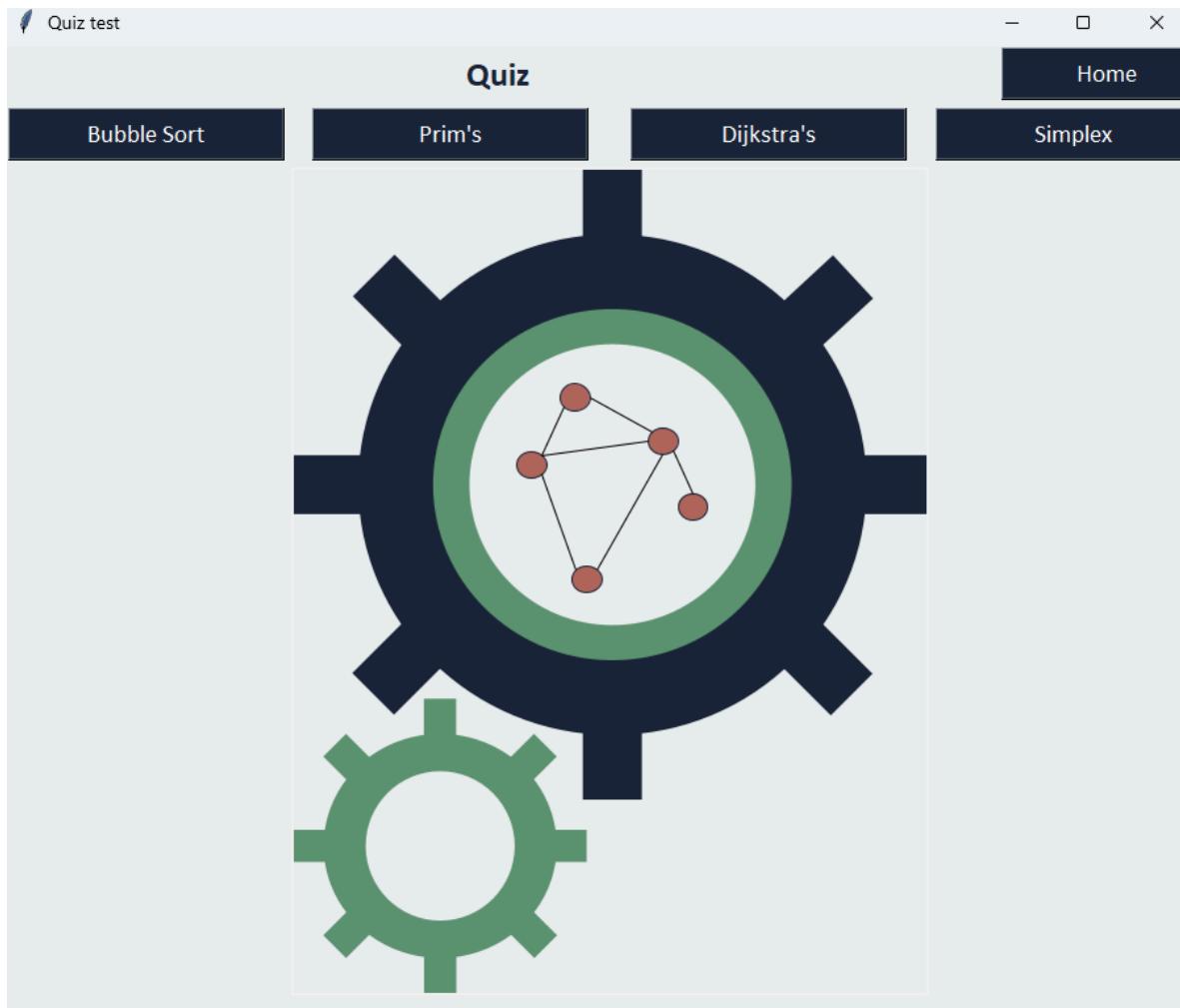
Then, I modified the Quiz constructor to set this flag to False so that the Menus are not displayed:

4.2.p

```
class Quiz(BubbleSort, Prim, Dijkstra, Simplex):
    def __init__(self, master, pUsername):
        self.master = master
        self.username = pUsername
        self.currentQType = None

        # instances of algorithm classes - composition:
        self.bubbleSort = BubbleSort(master, pUsername, show_menu=False)
        self.prim = Prim(master, pUsername, show_menu=False)
        self.dijkstra = Dijkstra(master, pUsername, show_menu=False)
        self.simplex = Simplex(master, pUsername, show_menu=False)
```

When I now run the Quiz page, the following is output:



When the bubble sort button is pressed:

 A screenshot of the same quiz application window. The "Bubble Sort" tab is now highlighted in orange. Below the tabs, a list of steps is displayed:

1. Start at the first item in the list.
2. Compare the current item with the next item.
3. If the two items are in the wrong position, swap them.
4. Move up one item to the next time in the list.
5. Repeat from step 3 until all the unsorted items have been compared.
6. If any items were swapped, repeat from step 1. Otherwise, the algorithm is complete.

 Below the steps, a message reads: "Please sort the following unsorted dataset in ascending order and input the number of passes it took:". A row of ten numbers is provided: 45, 10, 38, 49, 7, 44, 10, 17, 30, 6. Next to each number is a small input field containing "0 →". Below the numbers is a text input field labeled "Number of passes =". At the bottom of the window are three buttons: "correct/incorrect text" (in red), "Submit", and "New Question".

Both of these are as expected.

Now when I try to answer the question given correctly, the following is output:

The screenshot shows a quiz interface titled "Quiz". The navigation bar includes "Home", "Bubble Sort" (which is highlighted in red), "Prim's", "Dijkstra's", and "Simplex". The main content area contains the following text:

1. Start at the first item in the list.
2. Compare the current item with the next item.
3. If the two items are in the wrong position, swap them.
4. Move up one item to the next time in the list.
5. Repeat from step 3 until all the unsorted items have been compared.
6. If any items were swapped, repeat from step 1. Otherwise, the algorithm is complete.

Please sort the following unsorted dataset in ascending order and input the number of passes it took:

48 10 32 31 49 38 19 43 7 29

Number of passes =

Correct :)

This is as expected. Now when I try to sort it incorrectly, the following is output:

The screenshot shows a quiz interface titled "Quiz". The navigation bar includes "Home", "Bubble Sort" (highlighted in red), "Prim's", "Dijkstra's", and "Simplex". The main content area contains the same instructions as the previous screenshot.

Please sort the following unsorted dataset in ascending order and input the number of passes it took:

27 13 46 42 6 18 29 26 42 21

Number of passes =

Incorrect order :(

And for an incorrect number of passes but a correct order:

The screenshot shows a quiz interface titled "Quiz". The navigation bar includes "Home", "Bubble Sort" (highlighted in red), "Prim's", "Dijkstra's", and "Simplex". The main content area contains the same instructions as the previous screenshots.

Please sort the following unsorted dataset in ascending order and input the number of passes it took:

18 47 49 6 14 17 37 32 46 24

Number of passes =

Incorrect number of passes. Correct answer: 5

All of these are as expected so the Bubble Sort checking functionality works correctly. Now the final section for the Bubble Sort quiz is creating a new question. In order to do this, I modified

bubbleSortQuizWidgets to store the Label widgets and dropdown menus in lists so that I can change them in newQuestion:

4.2.q

```
self.numLabels = []
self.numChoices = []
self.dropdowns = []
iCounter = 1 # column index counter
for number in self.quizNumbers:
    numText= str(number)
    label = Label(self.bubbleSortQuizFrame, text=numText, font=fontMedium, fg="#1b263b"
    label.grid(row=5, column=iCounter, padx=1, pady=1)
    self.numLabels.append(label)

var = IntVar()
var.set(0) # default value
self.numChoices.append(var)

dropdown = OptionMenu(self.bubbleSortQuizFrame, var, *self.quizNumbers)
dropdown.grid(row=6, column=iCounter, padx=1, pady=1)
self.dropdowns.append(dropdown)

iCounter += 1
```

Then, for newQuestion, I wrote the following code:

4.2.r

```
def newQuestion(self, qType):
    # forget current widgets, create new ones for qType
    #print(f"in new question for: {qType}")
    if qType == "Bubble Sort":
        self.bubbleSortDataset()
        # reset text labels:
        for label, number in zip(self.numLabels, self.quizNumbers):
            label.config(text=str(number))
        # reset dropdowns:
        for var, dropdown in zip(self.numChoices, self.dropdowns):
            var.set(0)
            dropdown["menu"].delete(0, "end")
            for num in self.quizNumbers:
                dropdown["menu"].add_command(label=num, command=lambda value=num, var=var: var.set(value))
        # clear text box:
        self.passesInput.set("")
        self.answerText["text"] = ""
```

When I run the code, I input the following initially and press Submit:

The screenshot shows a window titled "Quiz test" with a tab bar at the top. The "Quiz" tab is active. Below the tabs, there are four buttons: "Bubble Sort" (highlighted in red), "Prim's", "Dijkstra's", and "Simplex". The main area contains a list of steps for bubble sort and a sorting interface.

1. Start at the first item in the list.
2. Compare the current item with the next item.
3. If the two items are in the wrong position, swap them.
4. Move up one item to the next time in the list.
5. Repeat from step 3 until all the unsorted items have been compared.
6. If any items were swapped, repeat from step 1. Otherwise, the algorithm is complete.

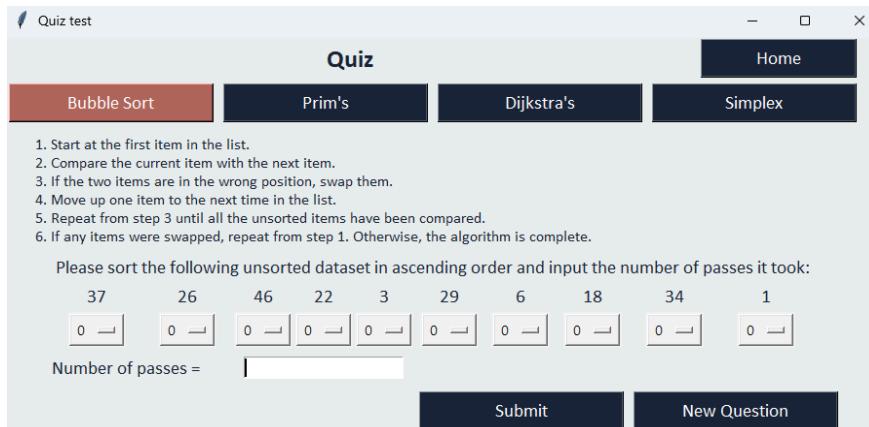
Please sort the following unsorted dataset in ascending order and input the number of passes it took:

9	39	26	10	18	12	14	44	29	27
<input type="button" value="9"/>	<input type="button" value="39"/>	<input type="button" value="26"/>	<input type="button" value="10"/>	<input type="button" value="18"/>	<input type="button" value="12"/>	<input type="button" value="14"/>	<input type="button" value="44"/>	<input type="button" value="29"/>	<input type="button" value="27"/>

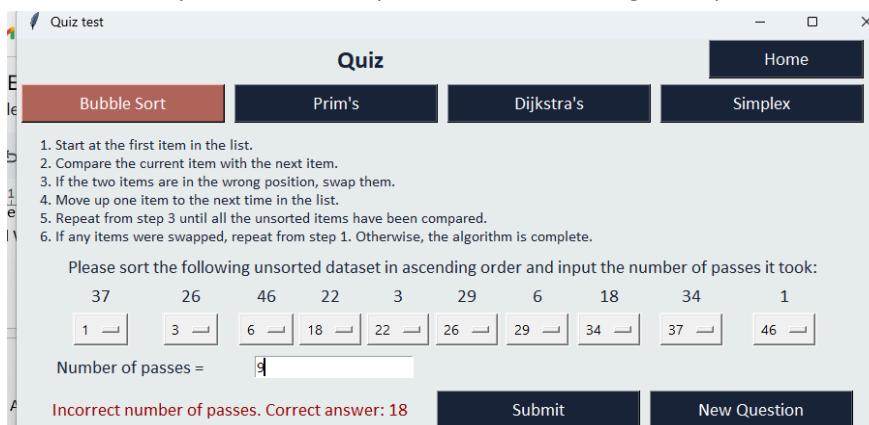
Number of passes =

Incorrect number of passes. Correct answer: 3

Then when I press the New Question button, the following is output:



Then when I try to answer the question, the following is output:

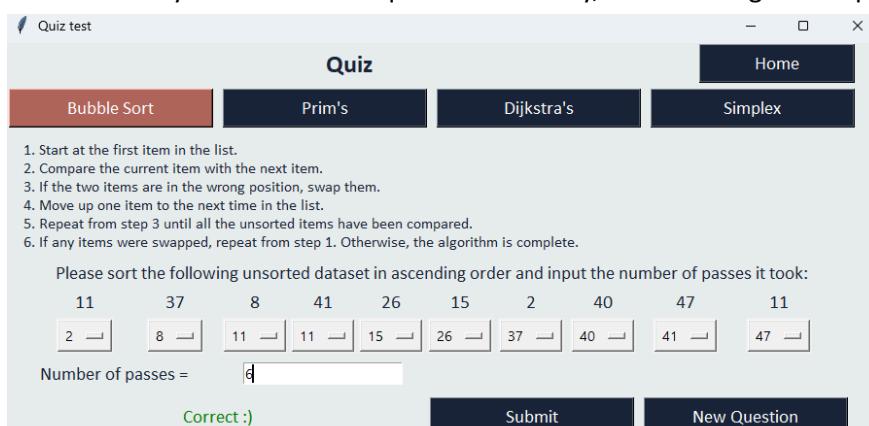


The correct number of passes should not be 18. Whilst I'm not sure what is causing this, I think that I should set the attribute `numberPasses` to 0 for each newQuestion.

4.2.s

```
# openwindow menu j.add_command(
    # clear text box:
    self.passesInput.set("")
    self.answerText["text"] = ""
    self.bubbleSort.numberPasses = 0
```

Now when I try to answer each question correctly, the following are output:



The screenshot shows a window titled "Quiz test" with a "Quiz" header. Below the header are four tabs: "Bubble Sort" (highlighted in red), "Prim's", "Dijkstra's", and "Simplex". The main content area contains the following text:

1. Start at the first item in the list.
2. Compare the current item with the next item.
3. If the two items are in the wrong position, swap them.
4. Move up one item to the next time in the list.
5. Repeat from step 3 until all the unsorted items have been compared.
6. If any items were swapped, repeat from step 1. Otherwise, the algorithm is complete.

Please sort the following unsorted dataset in ascending order and input the number of passes it took:

17	32	17	21	46	50	10	12	42	40
10	12	17	17	21	32	40	42	46	50

Number of passes =

Correct :) **Submit** **New Question**

By chance, both happened to have the same number of passes but passes weren't being added, which means that the issue has been fixed. All functionality for Bubble Sort Quiz questions has now been completed.

Now I am going to implement the sections needed for Simplex quiz questions. I am going to begin by setting up the generate constraints and objective function method:

4.2.t

```
def simplexConstraints(self):
    # 3 constraints and objective function
    # integers coefficients between -10 and 10, x and y not both 0
    print("in create simplex constraints")
    self.quizConstraints = []
    while len(self.quizConstraints) < 3:
        xCoef = randint(-10, 10)
        yCoef = randint(-10, 10)
        rhs = randint(-10, 10)
        if yCoef != 0 or xCoef != 0: # allows for one to be 0 using or
            self.quizConstraints.append([xCoef, yCoef, rhs])
    self.quizObjective = []
    while len(self.quizObjective) < 1:
        xCoef = randint(-10, 10)
        yCoef = randint(-10, 10)
        if yCoef != 0 or xCoef != 0: # allows for one to be 0 using or
            self.quizObjective.append(xCoef)
            self.quizObjective.append(yCoef)
```

The reason why I used while loops was because this would allow the code to iterate until the xCoef and yCoef are not both 0.

Then, I wrote the following code for the GUI for Simplex algorithm questions:

4.2.u

```

def simplexQuizWidgets(self):
    # widgets for simplex frame
    print("in simplex quiz widgets")

    self.simplexButton["bg"] = "#b2675e"
    self.simplexQuizFrame = Frame(self.master, bg="#eaebed")
    self.simplexQuizFrame.grid(row=2, column=0, columnspan=12, rowspan=8)

    Label(self.simplexQuizFrame, text=
        "1. Turn the inequalities into equalities by adding a slack variable. Rearrange the objective function\n to be equal to zero.\n"
        "2. Choose the most negative value in the objective row to become the basis.\n"
        "3. Work out the 0-values for each row. Using the smallest (but not negative) value, select the pivot.\n"
        "4. Divide the pivot row by the value of the pivot.\n"
        "5. Add or subtract multiple of the pivot row from the other rows so that the basis variable is 0.\n"
        "6. Repeat until the objective row contains no negative entries.\n"
        "7. Read off the optimal solution from the tableau.",
        font=fontSmall,
        fg="#1b263b",
        bg="#eaebed",
        justify="left").grid(row=2, column=0, rowspan=2, columnspan=12, sticky="w", padx=5, pady=5)

    self.constraintTexts = []
    for constraint in self.quizConstraints:
        a,b,c = constraint
        text = f"{a}x + {b}y ≤ {c}"
        self.constraintTexts.append(text)

    a,b, = self.quizObjective
    self.objectiveText = f"P = {a}x + {b}y"

    Label(self.simplexQuizFrame, text=
        "Constraints:\n" + "\n".join(self.constraintTexts) + "\n"
        "Objective: " + self.objectiveText,
        font=fontMedium,
        fg="#1b263b",
        bg="#eaebed",
        justify="left").grid(row=4, column=7, rowspan=2, columnspan=5, sticky="w", padx=5, pady=5)

    Label(self.simplexQuizFrame, text="x = ", font=fontMedium, fg="#1b263b", bg="#eaebed").grid(row=6, column=6, columnspan=2, padx=5)
    self.xInput = StringVar()
    Entry(self.simplexQuizFrame, textvariable=self.xInput).grid(row=6, column=8, columnspan=2)

    Label(self.simplexQuizFrame, text="y = ", font=fontMedium, fg="#1b263b", bg="#eaebed").grid(row=7, column=6, columnspan=2, padx=5)
    self.yInput = StringVar()
    Entry(self.simplexQuizFrame, textvariable=self.yInput).grid(row=7, column=8, columnspan=2)

    Label(self.simplexQuizFrame, text="P = ", font=fontMedium, fg="#1b263b", bg="#eaebed").grid(row=8, column=6, columnspan=2, padx=5)
    self.pInput = StringVar()
    Entry(self.simplexQuizFrame, textvariable=self.pInput).grid(row=8, column=8, columnspan=2)

    self.answerText = Label(self.simplexQuizFrame, text="correct/incorrect text", font=fontMedium, fg="#990000", bg="#eaebed")
    self.answerText.grid(row=9, column=0, columnspan=6)

    Button(self.simplexQuizFrame, text="Submit", command=self.check, bg="#1b263b", fg="white",
           font=fontMedium, width=20).grid(row=9, column=6, columnspan=3, padx=5, pady=5)
    Button(self.simplexQuizFrame, text="New Question", command=lambda: self.newQuestion("Simplex"), bg="#1b263b", fg="white",
           font=fontMedium, width=20).grid(row=9, column=9, columnspan=3, padx=5, pady=5)

```

4.2.v

```

# plot on graph:
colours = ["#606c38", "#b2675e", "#eca400"]

self.fig, self.ax = plt.subplots(figsize=(4,4))
for constraint in self.quizConstraints:
    a, b, c = constraint
    #print(a, b, c)
    i = self.quizConstraints.index(constraint)
    xVals = np.linspace(0, 10, 200) # used to plot constraint lines on graph
    if b != 0: # is a line in the form ax + by = c
        yVals = (c - a*xVals) / b
    else: # in the form ax = c
        xVals = np.full_like(xVals, c / a)
        yVals = np.linspace(0, 10, 200)
    self.ax.plot(xVals, yVals, label=f"Constraint {i+1}", color=colours[i])

self.ax.set_xlim(0,10)
self.ax.set_ylim(0,10)
self.ax.axhline(0, color="black", linewidth=1)
self.ax.axvline(0, color="black", linewidth=1)
self.ax.set_xlabel("x")
self.ax.set_ylabel("y")
self.ax.legend()
self.ax.grid()

self.canvas = FigureCanvasTkAgg(self.fig, master=self.simplexQuizFrame)
self.canvas.get_tk_widget().grid(row=4, column=0, rowspan=5, columnspan=6)
self.canvas.draw()

```

And I extended the newType method to include Simplex:

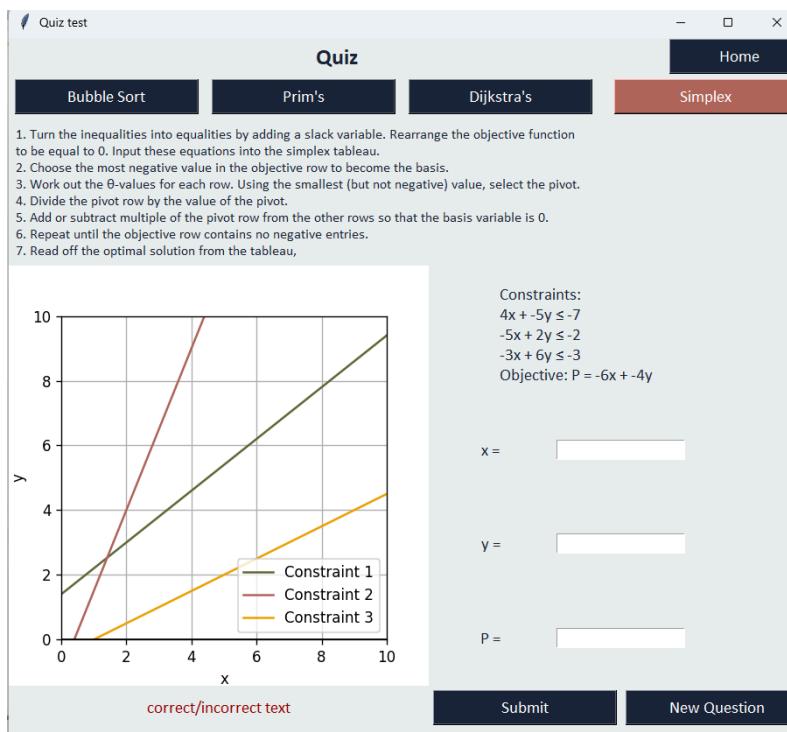
4.2.w

```

elif newQType == "Simplex":
    self.currentQType = "Simplex"
    self.simplexConstraints()
    self.simplexQuizWidgets()

```

When the code is run the following is output:



This is as expected.

Then, I extended the check method for the Simplex algorithm:

4.2.x

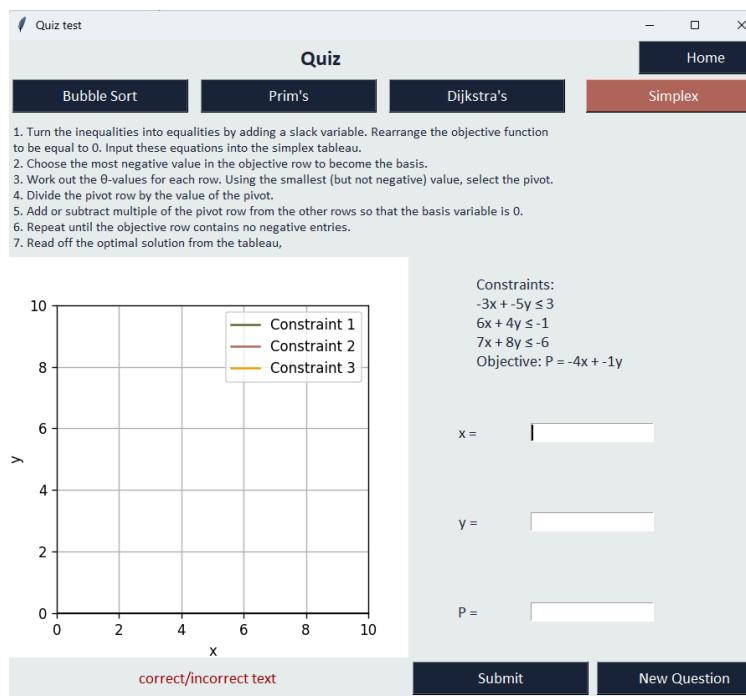
```

        elif self.currentQType == "Simplex":
            self.simplex.valueConstraints = self.quizConstraints[:]
            self.simplex.valueObjective = self.quizObjective[:]
            self.simplex.createTableau()
            self.simplex.simplexSteps()
            correctX = self.simplex.steps[-1][0]
            correctY = self.simplex.steps[-1][1]
            correctP = self.simplex.steps[-1][2]

            userX = self.xInput.get()
            userY = self.yInput.get()
            userP = self.pInput.get()

            if userX == correctX and userY == correctY and userP == correctP:
                self.answerText["text"] = "Correct :)"
                self.answerText["fg"] = "#008000"
                self.updateScore("Correct")
            else:
                self.answerText["text"] = "Incorrect\nTry a new question."
                self.answerText["fg"] = "#990000"
                self.updateScore("Incorrect")
    
```

Now, the following is displayed:



```

in add constraint
in add constraint
current question type: None, new question type: Simplex
in create simplex constraints
in simplex quiz widgets
in check
[-3 -5 1 0 0 3]
[ 6 4 0 1 0 -1]
[ 7 8 0 0 1 -6]
[4 1 0 0 0 0]
Exception in Tkinter callback
Traceback (most recent call last):
  File "C:\Users\ingri\AppData\Local\Programs\Thonny\lib\tkinter\__init__.py", line 1921, in __call__
    return self.func(*args)
  File "C:\Users\ingri\OneDrive\Documents\QM\QM School work\A Level\Computer Science\NEA\quiz page.py", line 291, in check
    self.simplex.simplexSteps()
  File "C:\Users\ingri\OneDrive\Documents\QM\QM School work\A Level\Computer Science\NEA\simplex page.py", line 312, in simplexSteps
    self.steps.append((x,y,P))
UnboundLocalError: local variable 'x' referenced before assignment

```

Based on the randomised constraints, it seems that there is no real solution to the set of constraints given. To try and handle this, finding the solution to the set of constraints randomised may be the best idea. Therefore, I modified simplexConstraints to the following:

4.2.y

```

def simplexConstraints(self):
    # 3 constraints and objective function
    # integers coefficients between -10 and 10, x and y not both 0
    print("in create simplex constraints")
    while True:
        self.quizConstraints = []
        while len(self.quizConstraints) < 3:
            xCoef = randint(-10, 10)
            yCoef = randint(-10, 10)
            rhs = randint(-10, 10)
            if yCoef != 0 or xCoef != 0: # allows for one to be 0 using or
                self.quizConstraints.append([xCoef, yCoef, rhs])
        self.quizObjective = []
        while len(self.quizObjective) < 1:
            xCoef = randint(-10, 10)
            yCoef = randint(-10, 10)
            if yCoef != 0 or xCoef != 0: # allows for one to be 0 using or
                self.quizObjective.append(xCoef)
                self.quizObjective.append(yCoef)
        # check if feasible:
        self.simplex.valueConstraints = self.quizConstraints[:]
        self.simplex.valueObjective = self.quizObjective[:]
        self.simplex.createTableau()
        self.simplex.simplexSteps()
        if self.simplex.steps[-1][0] is not None and self.simplex.steps[-1][1] is not None:
            break
        else:
            print("Generated infeasible problem")
    print("generated feasible problem")

```

And to fix the error seen in the screenshot of the Shell, I changed simplexSteps to the following:

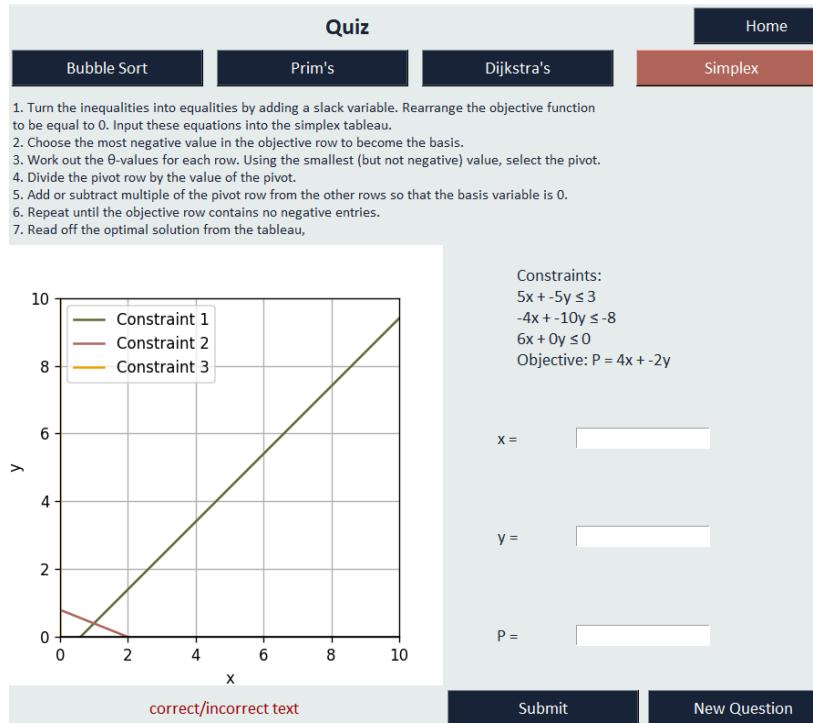
4.2.z

```

def simplexSteps(self):
    #print("in simplex steps")
    self.steps = [(0,0,0)]
    x,y,P = 0,0,0
    while not self.isOptimal():
        pivotColumn = self.findPivotColumn()
        #print("pivot column:", pivotColumn)
        pivotRow = self.findPivotRow(pivotColumn)
        #print("pivot row:", pivotRow)

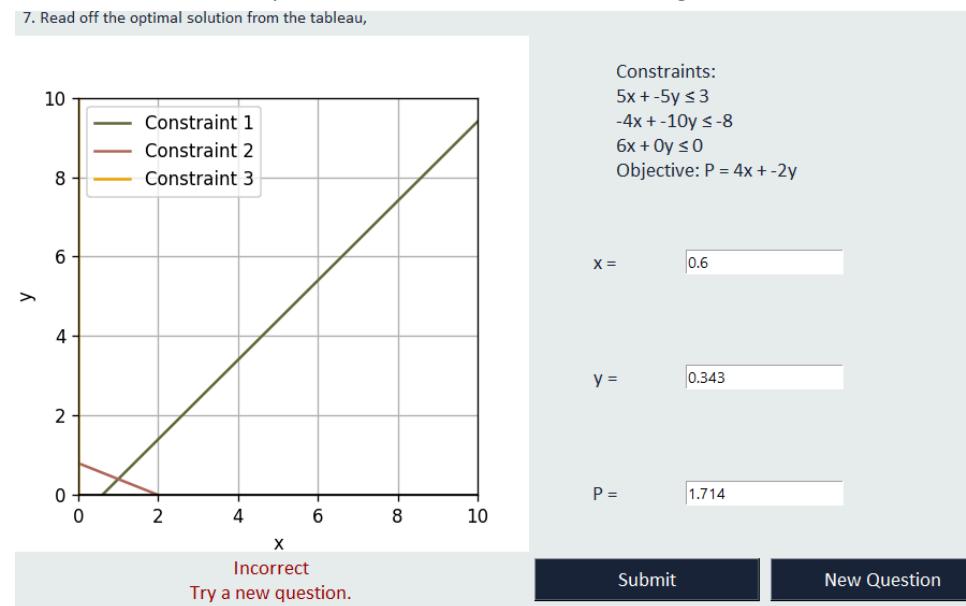
```

When I now run the code, the following is output:



From the graph, it is clear that the problem is feasible. Therefore, to check that the check functionality works correctly for a correct input. When I solve the simplex problem, the results I get

are $x=3/5$ $y=12/35$ and $P=12/7$. The issue with this is that the inputs can't handle fractions at the moment. So when I input the answers as floats, rounding issues occur:



I think that allowing fractions to be input would make it easier for users because they don't have to convert their answer to decimals and then worry about rounding errors. So, I changed it to the following:

4.2.A

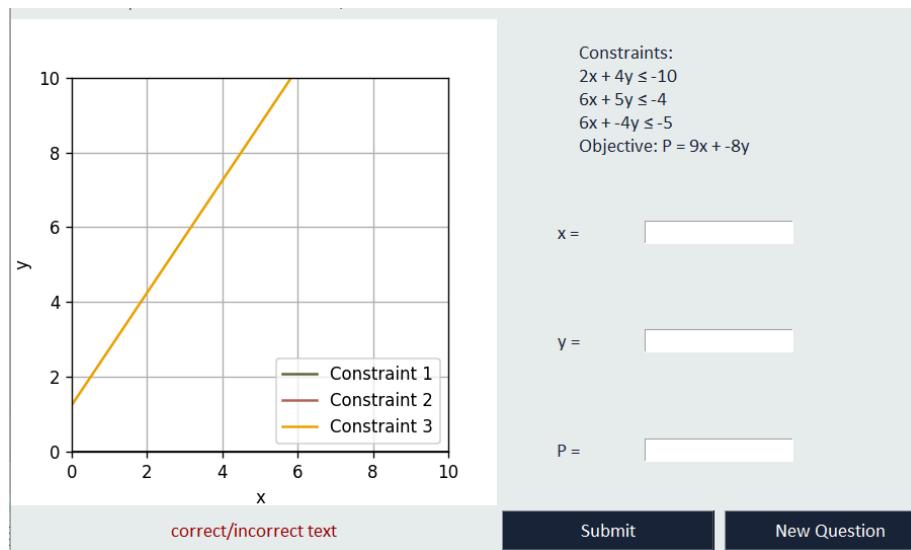
```
from fractions import Fraction

elif self.currentQType == "Simplex":
    self.simplex.valueConstraints = self.quizConstraints[:]
    self.simplex.valueObjective = self.quizObjective[:]
    self.simplex.createTableau()
    self.simplex.simplexSteps()
    correctX = Fraction(self.simplex.steps[-1][0]).limit_denominator()
    correctY = Fraction(self.simplex.steps[-1][1]).limit_denominator()
    correctP = Fraction(self.simplex.steps[-1][2]).limit_denominator()

    try:
        userX = Fraction(self.xInput.get())
        userY = Fraction(self.yInput.get())
        userP = Fraction(self.pInput.get())
    except ValueError:
        self.answerText["text"] = "Please enter numbers of fractions (e.g. 3/5)"
        self.answerText["fg"] = "#990000"
        return

    if userX == correctX and userY == correctY and userP == correctP:
        self.answerText["text"] = "Correct :)"
        self.answerText["fg"] = "#008000"
        self.updateScore("Correct")
    else:
        self.answerText["text"] = "Incorrect\nTry a new question."
        self.answerText["fg"] = "#990000"
        self.updateScore("Incorrect")
```

I then ran the code again. This time the following was output:



This is again not a feasible problem, despite tests saying it is. I wasn't sure why unfeasible problems were still being displayed. So, I looked into the requirements for simplex and I found out that the rhs value must be greater than 0³¹ for a feasible solution to be able to be found. This is a quick fix as I can just change the range of values for the rhs.

4.2.B

```
self.quizConstraints = []
while len(self.quizConstraints) < 3:
    xCoef = randint(-10, 10)
    yCoef = randint(-10, 10)
    rhs = randint(0, 10)
    if yCoef != 0 or xCoef != 0: # allows for one to be 0 using or
        self.quizConstraints.append([xCoef, yCoef, rhs])
```

Then, I realised that there may be an issue in the check method because I am calculating the optimal values of x y and P again but not changing the tableau to be blank. Whilst this is not necessarily causing the issues, because the tableau will just be determined as optimal, reducing redundant code, especially for the Simplex algorithm is useful:

4.2.C

```
elif self.currentQType == "Simplex":
    correctX = Fraction(self.simplex.steps[-1][0]).limit_denominator()
    correctY = Fraction(self.simplex.steps[-1][1]).limit_denominator()
    correctP = Fraction(self.simplex.steps[-1][2]).limit_denominator()
    print(f"correct: x={correctX}, y={correctY}, P={correctP}")

try:
    userX = Fraction(self.xInput.get())
    userY = Fraction(self.yInput.get())
    userP = Fraction(self.pInput.get())
except ValueError:
    self.answerText["text"] = "Please enter numbers of fractions (e.g. 3/5)"
    self.answerText["fg"] = "#990000"
    return

if userX == correctX and userY == correctY and userP == correctP:
    self.answerText["text"] = "Correct :)"
    self.answerText["fg"] = "#008000"
    self.updateScore("Correct")
else:
    self.answerText["text"] = "Incorrect\nTry a new question."
    self.answerText["fg"] = "#990000"
    self.updateScore("Incorrect")
```

When I run the code now, the following is output:

Quiz test

Quiz

[Home](#)

[Bubble Sort](#) [Prim's](#) [Dijkstra's](#) [Simplex](#)

1. Turn the inequalities into equalities by adding a slack variable. Rearrange the objective function to be equal to 0. Input these equations into the simplex tableau.

2. Choose the most negative value in the objective row to become the basis.

3. Work out the θ -values for each row. Using the smallest (but not negative) value, select the pivot.

4. Divide the pivot row by the value of the pivot.

5. Add or subtract multiple of the pivot row from the other rows so that the basis variable is 0.

6. Repeat until the objective row contains no negative entries.

7. Read off the optimal solution from the tableau,

Constraints:

$0x + 8y \leq 8$
 $1x + 5y \leq 9$
 $-9x + -1y \leq 6$

Objective: $P = 6x + -8y$

x =

y =

P =

Correct :)

[Submit](#) [New Question](#)

```

in add constraint
in add constraint
current question type: None, new question type: Simplex
in create simplex constraints
[-10 -3 1 0 0 4]
[ 5 -2 0 1 0 8]
[-5 -1 0 0 1 4]
[ 5 -7 0 0 0 0]
steps:
(0, 0, 0)
(0, 0, 0)
Generated infeasible problem
[10 7 1 0 0 4]
[-8 -4 0 1 0 3]
[ 4 -10 0 0 1 5]
[-1 -2 0 0 0 0]
[[ 10 7 1 0 0 4]
 [ -8 -4 0 1 0 3]
 [ 4 -10 0 0 1 5]
 [ -1 -2 0 0 0 0]]
[[ 1 1 0 0 0 0]
 [-4 0 0 1 0 3]
 [14 0 0 0 1 5]
 [ 1 0 0 0 0 0]]
x, y, P: 0 0 0
steps:
(0, 0, 0)
(0, np.int64(0), np.int64(0))
(0, np.int64(0), np.int64(0))
Generated infeasible problem

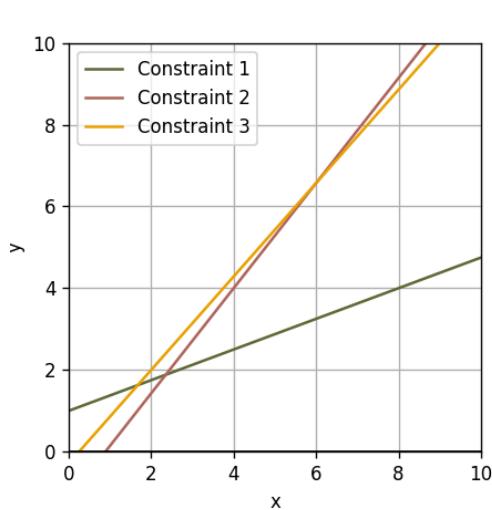
```

```

[-7 4 1 0 0 6]
[ -7 -10 0 1 0 1]
[ -8 0 0 0 1 8]
[ -3 7 0 0 0 0]
steps:
(0, 0, 0)
(0, 0, 0)
Generated infeasible problem
[0 8 1 0 0 8]
[1 5 0 1 0 9]
[-9 -1 0 0 1 6]
[-6 8 0 0 0 0]
[[ 0 8 1 0 0 8]
 [ 1 5 0 1 0 9]
 [-9 -1 0 0 1 6]
 [-6 8 0 0 0 0]]
[[ 0 8 1 0 0 8]
 [ 1 5 0 1 0 9]
 [ 0 44 0 9 1 87]
 [ 0 38 0 6 0 54]]
x, y, P: 9 0 54
steps:
(0, 0, 0)
(np.int64(9), 0, np.int64(54))
(np.int64(9), 0, np.int64(54))
generated feasible problem
in simplex quiz widgets
in check
correct: x=9, y=0, P=54
in update score

```

This is all as expected. Whilst I can't determine every single possible set of constraints, it seems that the constraints I am now obtaining do seem to give feasible solutions. Some examples of randomise constraints include the following:

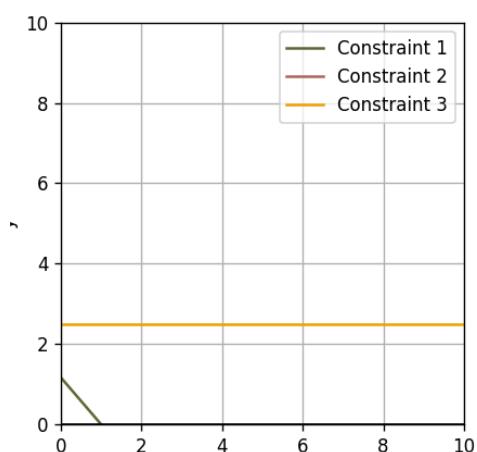


Constraints:
 $-3x + 8y \leq 8$
 $9x + -7y \leq 8$
 $8x + -7y \leq 2$
 Objective: $P = 1x + 7y$

x =

y =

P =

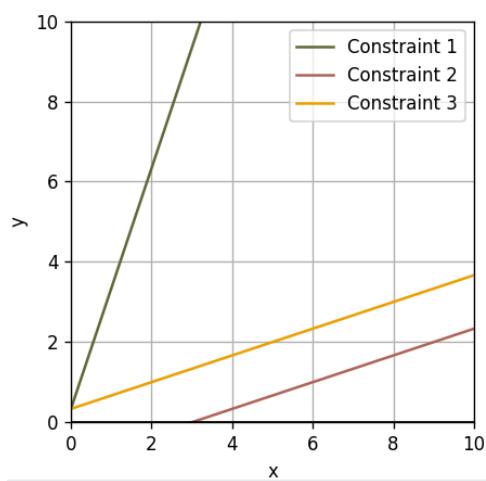


Constraints:
 $7x + 6y \leq 7$
 $-2x + -8y \leq 1$
 $0x + 2y \leq 5$
 Objective: $P = -10x + 4y$

x =

y =

P =

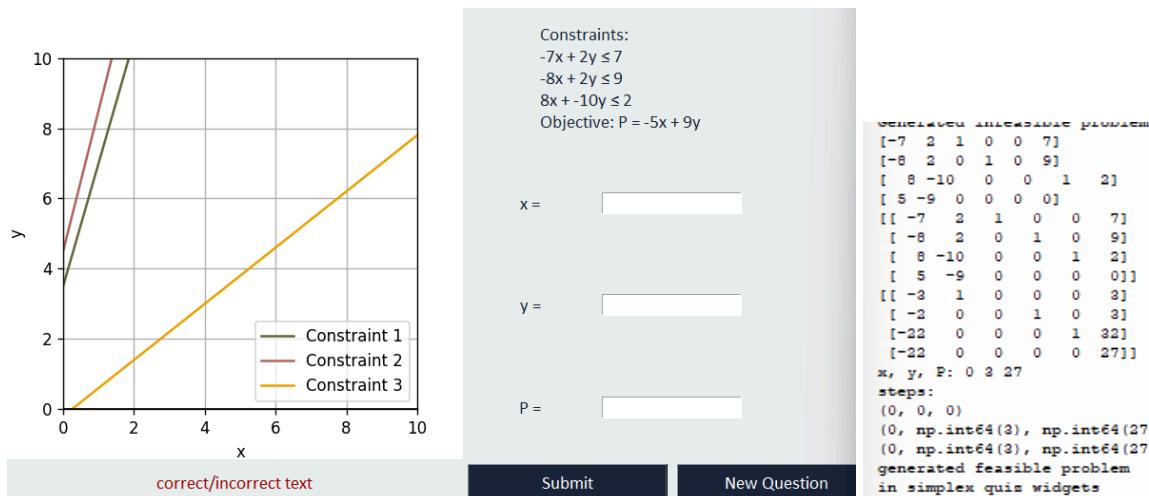


Constraints:
 $-9x + 3y \leq 1$
 $1x + -3y \leq 3$
 $-1x + 3y \leq 1$
 Objective: $P = 7x + 1y$

x = 3

y = 0

P = 21



This is not as expected because the constraints should not give a feasible solution. To try and fix this, I decided to modify the Simplex class to have an attribute called invalid, which will be set to True if the Simplex algorithm determines an unbounded feasible region:

4.2.D

```

class Simplex(Menu):
    def __init__(self, master, pUsername, show_menu=True):
        Menu.__init__(self, master, 12, 14)
        self.username = pUsername

        self.textConstraints = [] # for widgets
        self.valueConstraints = []
        self.valueObjective = []
        self.tableau = np.array([], dtype=float) # empty nu

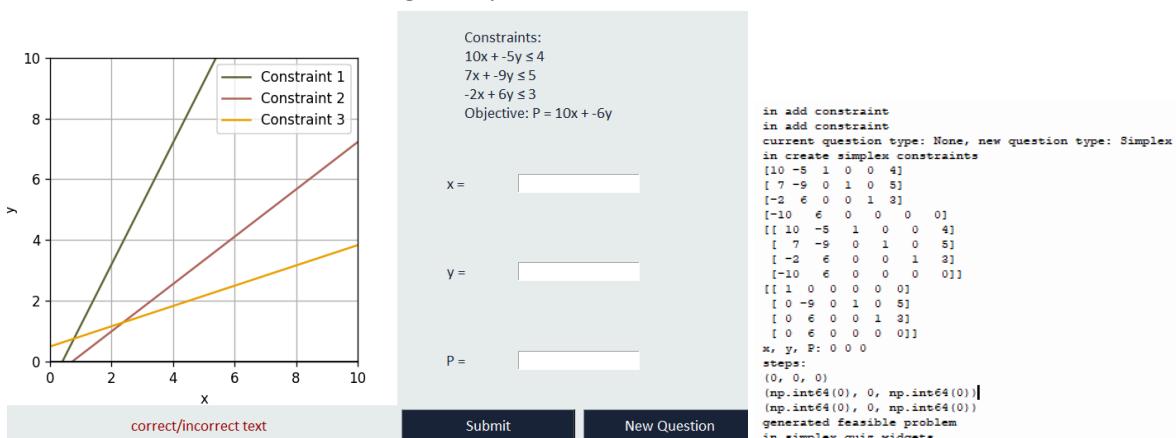
        self.canvas = None

        self.currentStep = 0

        self.simplexPath = [(0,0)]

        self.simplexWidgets() # check for unbounded:
        if show_menu:
            if self.tableau[pivotRow, pivotColumn] <= 0 :
                self.invalidMessage["text"] = "Unbounded solution."
                self.invalid = True
                return
            self.invalid = False
    
```

When I run the code, the following is output:



For other randomised constraints:

Constraints:
 $7x + 10y \leq 6$
 $9x - y \leq 6$
 $0x + 3y \leq 4$
Objective: $P = -9x + -10y$

x =

y =

P =

Correct :)

Submit

New Question

Constraints:
 $2x + 7y \leq 3$
 $-8x + 9y \leq 7$
 $-10x + -1y \leq 1$
Objective: $P = -3x + 0y$

x = 0

y = 0

P = 0

Correct :)

Submit

New Question

Home

QUIZ

Bubble Sort

Prim's

Dijkstra's

Simplex

```
in add constraint
in add constraint
current question type: None, new qu
in create simplex constraints
[-8 -3  1  0  0  4]
[10 -7  0  1  0  0]
[10 -5  0  0  1  10]
[ 9 -8  0  0  0  0]

steps:
(0, 0, 0)
Generated infeasible problem
[[ 3  4  1  0  0  8]
 [ 8  9  0  1  0  6]
 [-1  0  0  0  1  6]
 [10 -6  0  0  0  0]
 [[ 3  4  1  0  0  8]
 [ 8  9  0  1  0  6]
 [-1  0  0  0  1  6]
 [10 -6  0  0  0  0]]
 [[ 3  0  1  0  0  8]
 [ 0  1  0  0  0  0]
 [-1  0  0  0  1  6]
 [10  0  0  0  0  0]]
x, y, P: 0 0 0
steps:
(0, 0, 0)
(0, np.int64(0), np.int64(0))
(0, np.int64(0), np.int64(0))
generated feasible problem
in simplex quiz widgets
```

Constraints:
 $3x + 4y \leq 8$
 $8x + 9y \leq 6$
 $-1x + 0y \leq 6$
Objective: $P = -10x + 6y$

x =

y =

P =

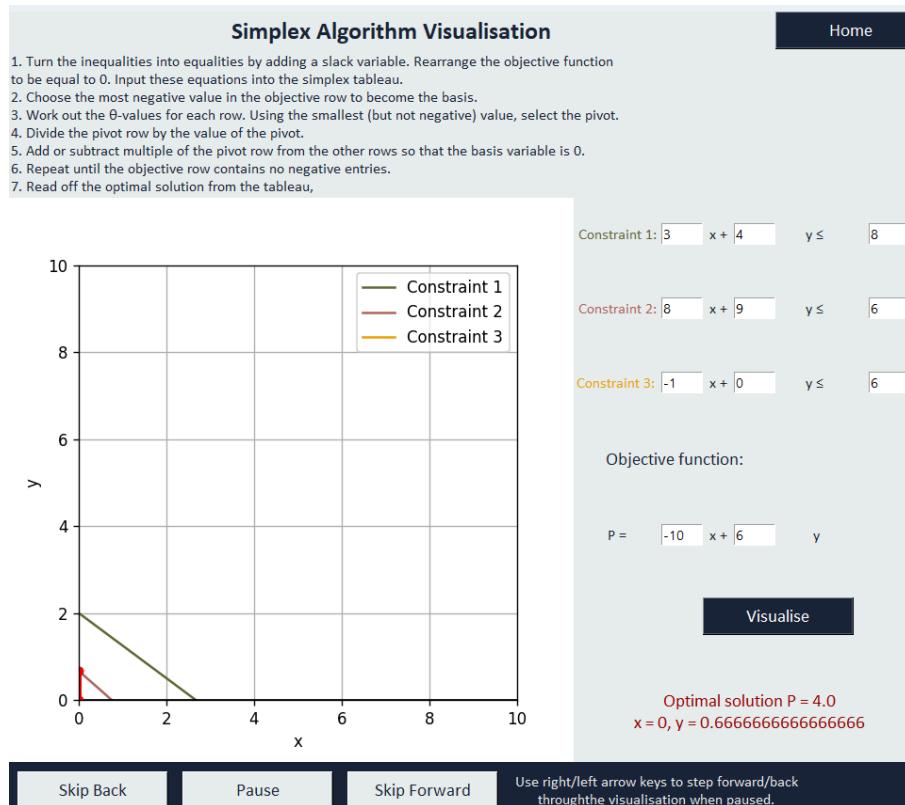
correct/incorrect text

Submit

New Question

The solution to this should not be $x=0 y=0$ as it should be at one of the points at which constraint 2 crosses the axes (this is because constraint 2 is the boundary of the feasible region). To check what

solution I should get, I put the constraints and objective function in my visualisation that I programmed for Simplex and this was output:



From these issues, I think I have determined 2 issues with my code. Firstly, I think that there is an issue regarding the objective function. I think that having negative values as coefficients may be complicating my code, although it should not be an issue. However, to sort this out, I am simply going to restrict the values of the x and y coefficients in the objective function to be between 0 and 10 (but both can't be 0).

4.2.E

```
self.quizObjective = []
while len(self.quizObjective) < 1:
    xCoef = randint(0, 10)
    yCoef = randint(0, 10)
    if yCoef != 0 or xCoef != 0: # allows for one to be 0 using or
        self.quizObjective.append(float(xCoef))
        self.quizObjective.append(float(yCoef))
```

Secondly, from the Shell output for the quiz question above, the values in the tableau all seem to be integers. I think that this may be causing issues with rounding. So, when I put the values of the constraints into the valueConstraints and valueObjective attributes for Simplex, I am going to put in float values. Additionally, this will bring about issues with how the user can input their answer as decimals will not allow for accurate answers to be input. Therefore, I am also going to make sure that fractional values are being stored in steps and the answers input are fractions:

4.2.F

```

self.pivot(pivotRow, pivotColumn)

# steps append ...
x = Fraction(self.getVariableValue(0)).limit_denominator() # x is in tableau column 0
y = Fraction(self.getVariableValue(1)).limit_denominator() # y is in column index 1
P = Fraction(self.tableau[-1][-1]).limit_denominator()
print("At this step:")
print("x, y, P:", x, y, P)
self.steps.append((x,y,P))
self.simplexPath.append((x,y))
    
```

4.2.G

```

print(f"state of invalid: {self.simplex.invalid}")
if self.simplex.invalid == False:
    break
else:
    print("Generated infeasible problem")
    self.simplex.steps = [(0,0,0)]
    self.simplex.tableau = np.array([], dtype=float) # empty numpy array ''
    self.simplex.valueConstraints = []
    self.simplex.valueObjective = []

print("generated feasible problem")
correctX = Fraction(self.simplex.steps[-1][0]).limit_denominator()
correctY = Fraction(self.simplex.steps[-1][1]).limit_denominator()
correctP = Fraction(self.simplex.steps[-1][2]).limit_denominator()

#correctX, correctY, correctP = self.simplex.steps[-1]
print(f"correct: x={correctX}, y={correctY}, P={correctP}")
    
```

4.2.H

```

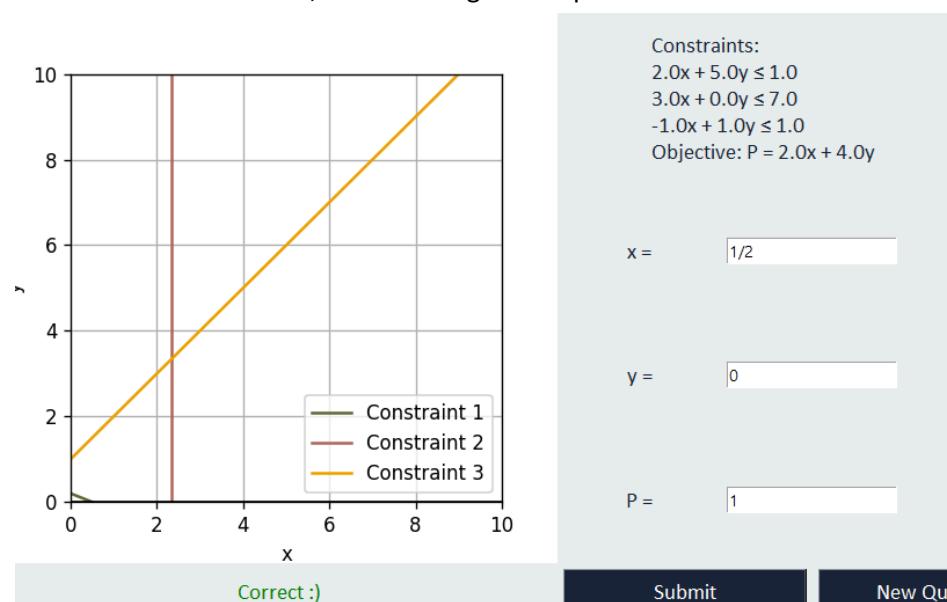
elif seit.currentQType == "Simplex":

    correctX = Fraction(self.simplex.steps[-1][0]).limit_denominator()
    correctY = Fraction(self.simplex.steps[-1][1]).limit_denominator()
    correctP = Fraction(self.simplex.steps[-1][2]).limit_denominator()

    #correctX, correctY, correctP = self.simplex.steps[-1]
    print(f"correct: x={correctX}, y={correctY}, P={correctP}")

    try:
        userX = Fraction(self.xInput.get())
        userY = Fraction(self.yInput.get())
        userP = Fraction(self.pInput.get())
    except ValueError:
        self.answerText["text"] = "Please enter numbers of fractions (e.g. 3/5)"
        self.answerText["fg"] = "#990000"
    return
    
```

When I run the code now, the following are output:

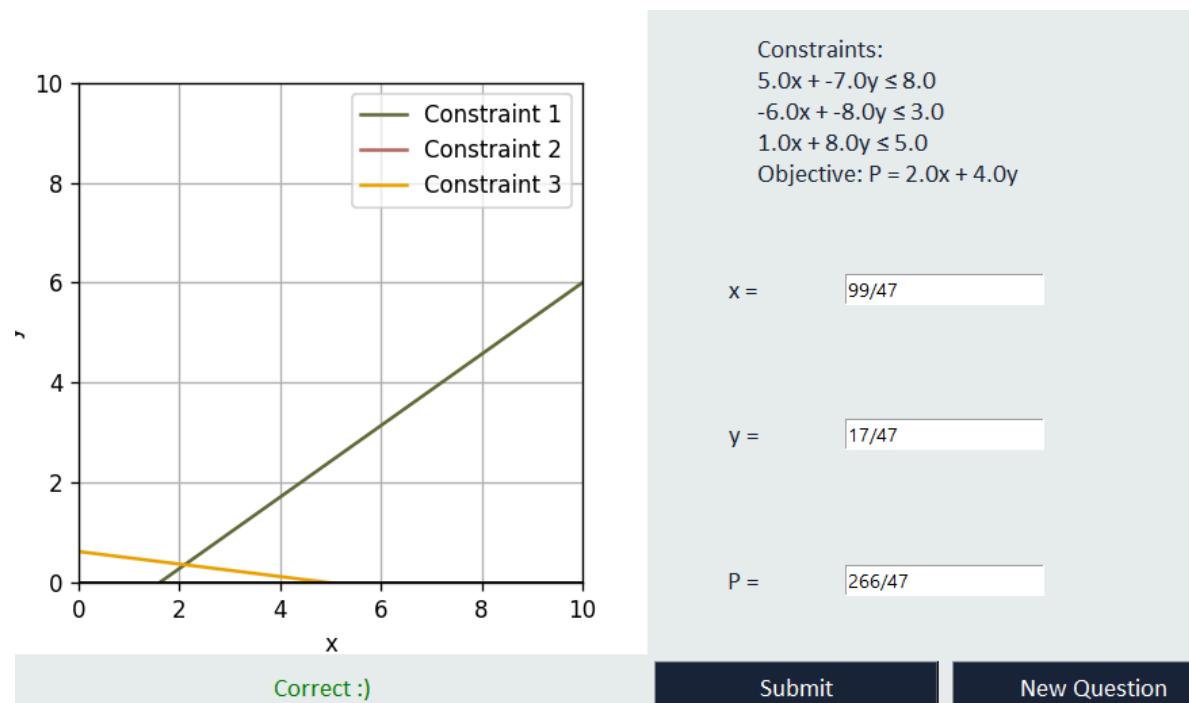


```

in add constraint
in add constraint
current question type: None, new question type: Simplex
in create simplex constraints
simplex valueConstraints: [[2.0, 5.0, 1.0], [3.0, 0.0, 7.0], [-1.0, 1.0, 1.0]]
simplex valueObjective: [2.0, 4.0]
initial tableau:
[[ 2.  5.  1.  0.  0.  1.]
 [ 3.  0.  0.  1.  0.  7.]
 [-1.  1.  0.  0.  1.  1.]
 [-2. -4.  0.  0.  0.  0.]]
in new step loop
pivot column: 1
pivot row: 0
Tableau after pivot
[[ 0.4  1.   0.2  0.   0.   0.2]
 [ 3.   0.   0.   1.   0.   7. ]
 [-1.4  0.   -0.2  0.   1.   0.8]
 [-0.4  0.   0.8  0.   0.   0.8]]
At this step:
x, y, P: 0 0.2 0.8
in new step loop
pivot column: 0
pivot row: 0
Tableau after pivot
[[ 1.   2.5  0.5  0.   0.   0.5]
 [ 0.   -7.5 -1.5  1.   0.   5.5]
 [ 0.   3.5  0.5  0.   1.   1.5]
 [ 0.   1.   1.   0.   0.   1. ]]
At this step:
x, y, P: 0.5 0 1.0
steps:
(0, 0, 0)
(0, np.float64(0.2), np.float64(0.8))
(np.float64(0.5), 0, np.float64(1.0))
(np.float64(0.5), 0, np.float64(1.0))
state of invalid: False
generated feasible problem
correct: x=1/2, y=0, P=1
in simplex quiz widgets
in check
correct: x=1/2, y=0, P=1
in update score

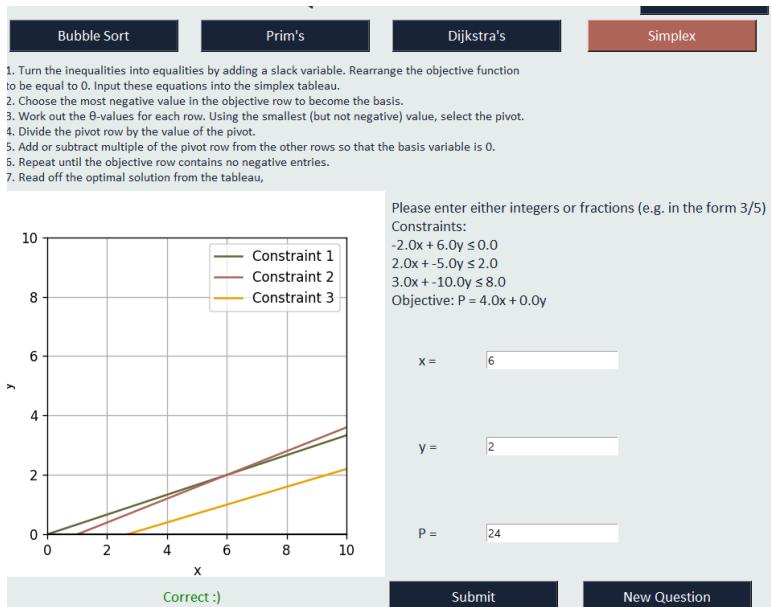
```

Then after putting the values in Simplex to be fractions:



```

in add constraint
in add constraint
current question type: None, new question type: Simplex
in create simplex constraints
simplex valueConstraints: [[5.0, -7.0, 8.0], [-6.0, -8.0, 3.0], [1.0, 8.0, 5.0]]
simplex valueObjective: [2.0, 4.0]
initial tableau:
[[ 5. -7. 1. 0. 0. 8.]
 [-6. -8. 0. 1. 0. 3.]
 [ 1. 8. 0. 0. 1. 5.]
 [-2. -4. 0. 0. 0. 0.]]
in new step loop
pivot column: 1
pivot row: 2
Tableau after pivot
[[ 5.875 0. 1. 0. 0.875 12.375]
 [-5. 0. 0. 1. 1. 8. ]
 [ 0.125 1. 0. 0. 0.125 0.625]
 [-1.5 0. 0. 0. 0.5 2.5 ]]
At this step:
x, y, P: 0 5/8 5/2
in new step loop
pivot column: 0
pivot row: 0
Tableau after pivot
[[ 1. 0. 0.17021277 0. 0.14893617 2.10638298]
 [ 0. 0. 0.85106383 1. 1.74468085 18.53151489]
 [ 0. 1. -0.0212766 0. 0.10638298 0.36170213]
 [ 0. 0. 0.25581915 0. 0.72340426 5.65957447]]
At this step:
x, y, P: 99/47 17/47 266/47
steps:
(0, 0)
(Fraction(0, 1), Fraction(5, 8), Fraction(5, 2))
(Fraction(99, 47), Fraction(17, 47), Fraction(266, 47))
(Fraction(99, 47), Fraction(17, 47), Fraction(266, 47))
state of invalid: False
generated feasible problem
correct: x=99/47, y=17/47, P=266/47
in simplex quiz widgets
    
```



All of this seems to be as expected. Although I can't test every single set of constraints, user inputs of fractions seem to be working and the optimal results don't seem to be just $x=0$ $y=0$ $P=0$. Now, I am going to complete the functionality for generating new questions for the Simplex algorithm. See the first part of the code in the method on the next page.

4.2.1

```

self.ax.set_xlim(0,10)
self.ax.set_ylim(0,10)
self.ax.axhline(0, color="black", linewidth=1)
self.ax.axvline(0, color="black", linewidth=1)
self.ax.set_xlabel("x")
self.ax.set_ylabel("y")
self.ax.legend()
self.ax.grid()

self.canvas.draw()
    
```

4.2.J

```

elif qType == "Simplex":
    self.simplexConstraints()
    self.answerText["text"] = ""

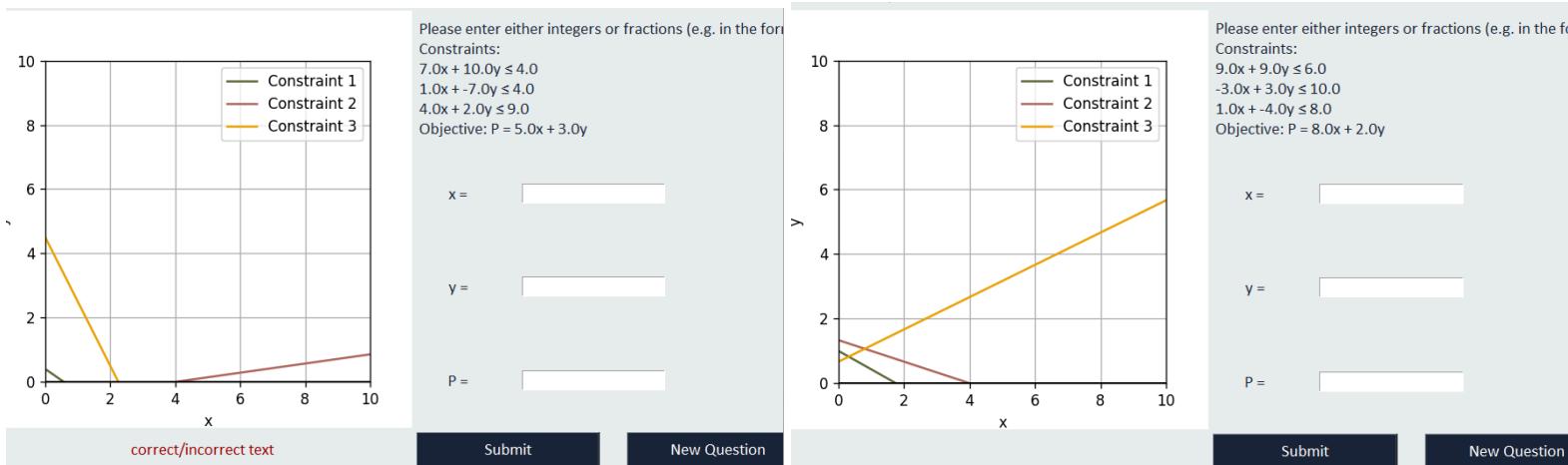
    self.constraintsTexts = [ f"{a}x + {b}y ≤ {c}" for a,b,c in self.quizConstraints]
    self.objectiveText = f"P = {self.quizObjective[0]}x + {self.quizObjective[1]}y"
    self.textSayingConstraints["text"] = "Please enter either integers or fractions (e.g. in the form 3/5)\n" + "Constraints:\n" + "\n".join(self.constraintTexts) + "\n" + "Objective: " + self.objectiveText

    self.xInput.set("")
    self.yInput.set("")
    self.pInput.set("")

    self.ax.clear()
    for constraint in self.quizConstraints:
        a, b, c = constraint
        #print(a, b, c)
        i = self.quizConstraints.index(constraint)
        xVals = np.linspace(0, 10, 200) # used to plot constraint lines on graph
        if b != 0: # is a line in the form ax + by = c
            yVals = (c - a*xVals) / b
        else: # in the form ax = c
            xVals = np.full_like(xVals, c / a)
            yVals = np.linspace(0, 10, 200)
        self.ax.plot(xVals, yVals, label=f"Constraint {i+1}", color=self.colours[i])

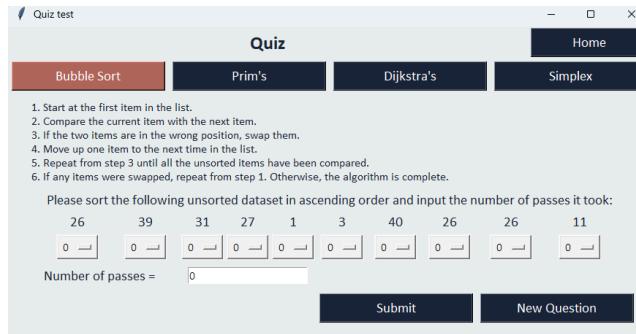
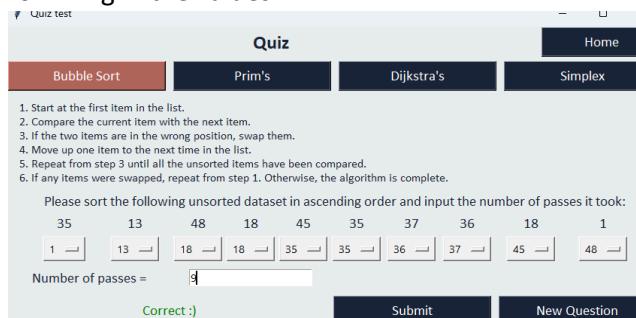
```

When I run the code and press the New Question button:

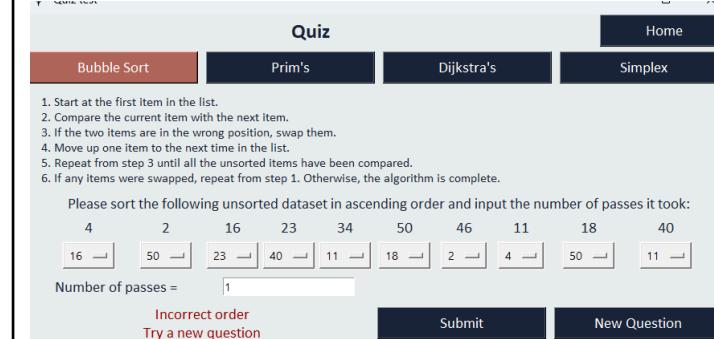
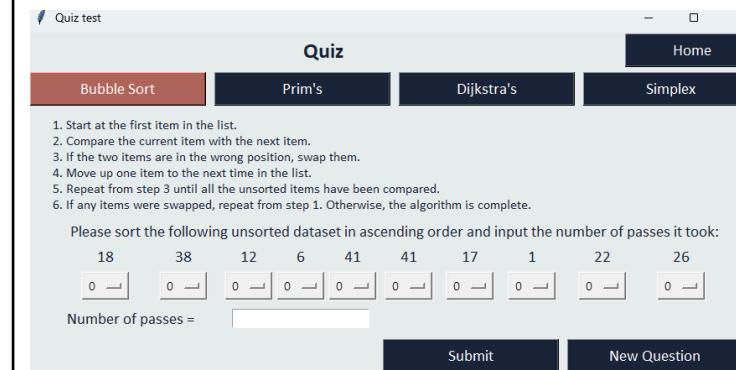


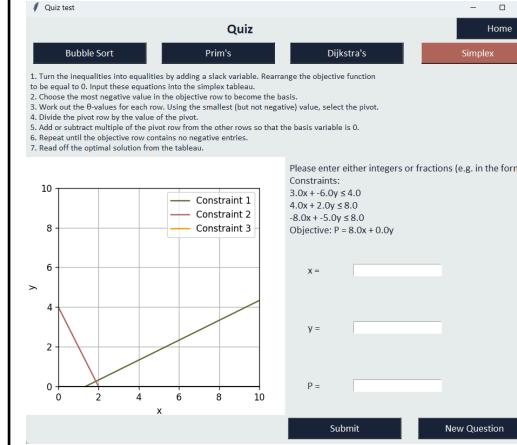
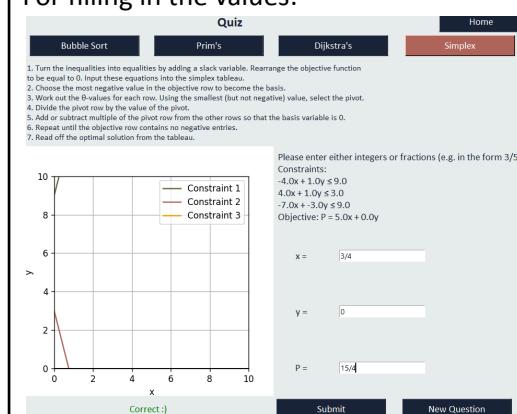
This is as expected. The final thing needed to complete these sections is to complete the updateScore method. However, I have decided to complete this in Iteration 3 once all the quiz question types have been completed.

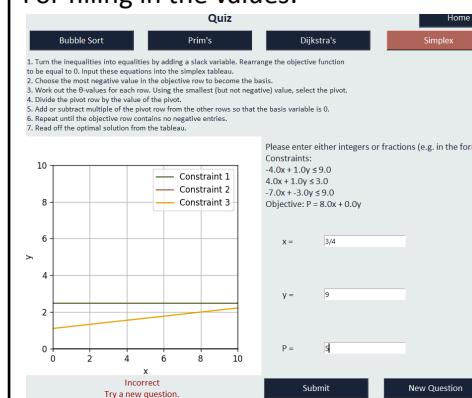
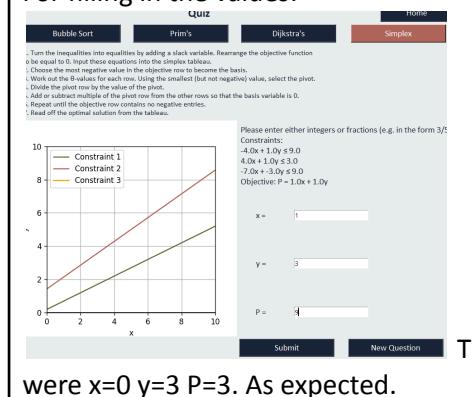
Now I will test the code developed so far using the white box testing table specified in the QUIZ PAGE design section. As only the Bubble Sort and Simplex sections have been developed so far, these are the only criteria that I will test against.

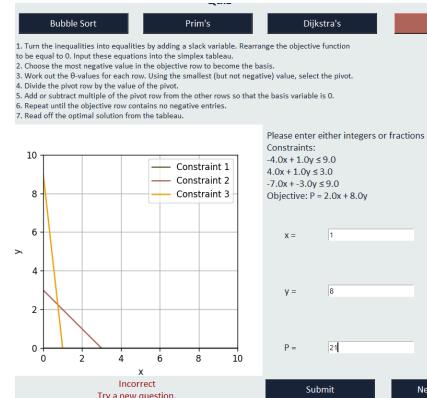
TEST DATA	EXPECTED RESULT	JUSTIFICATION	CODE	ACTUAL OUTPUT
Press the Bubble Sort button.	The Bubble Sort button is highlighted in pink and text describing the algorithm is below this. Dropdown menus for the unordered dataset are displayed below it and there is an entry field for the number of passes.	This ensures that the correct widgets required for the user to complete a Bubble Sort question are displayed and that the randomised dataset functionality works correctly.	4.2.j 4.2.h 4.2.i 4.2.j 4.2.q	 <p>As expected.</p>
Submit a Bubble Sort question with both the dropdown menus and number of passes correctly filled in.	Text will appear saying 'Correct' and the correct score in the database is updated.	This ensures that the program can validate a correct answer as being correct.	4.2.j 4.2.h 4.2.i 4.2.j 4.2.q 4.2.l	<p>The functionality for updating the database has not been completed in this iteration so that part cannot be tested.</p> <p>For filling in the values:</p>  <p>As expected.</p>

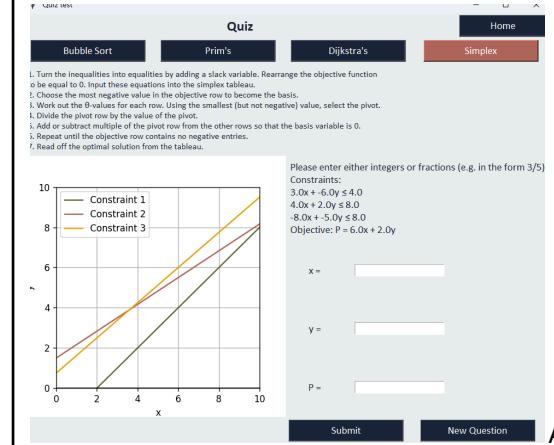
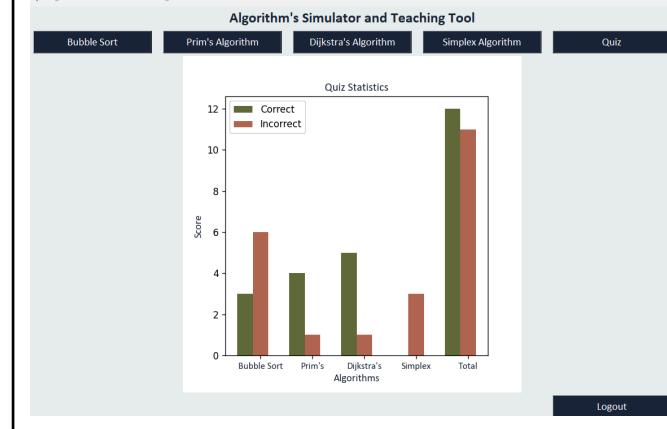
Submit a Bubble Sort question with only the dropdown menus filled in correctly.	Text will appear saying 'Incorrect number of passes' and the incorrect score in the database is updated.	This ensures that the program can validate an incorrect answer as being incorrect.	4.2.j 4.2.h 4.2.i 4.2.j 4.2.q 4.2.l	<p>The functionality for updating the database has not been completed in this Iteration so that part cannot be tested.</p> <p>For filling in the values:</p> <p>Quiz test</p> <p>Quiz</p> <p>Bubble Sort Prim's Dijkstra's Simplex</p> <p>1. Start at the first item in the list. 2. Compare the current item with the next item. 3. If the two items are in the wrong position, swap them. 4. Move up one item to the next time in the list. 5. Repeat from step 3 until all the unsorted items have been compared. 6. If any items were swapped, repeat from step 1. Otherwise, the algorithm is complete.</p> <p>Please sort the following unsorted dataset in ascending order and input the number of passes it took:</p> <table border="1"> <tr> <td>1</td> <td>42</td> <td>47</td> <td>16</td> <td>25</td> <td>49</td> <td>7</td> <td>16</td> <td>45</td> <td>19</td> </tr> <tr> <td>1 —</td> <td>7 —</td> <td>16 —</td> <td>16 —</td> <td>19 —</td> <td>25 —</td> <td>42 —</td> <td>45 —</td> <td>47 —</td> <td>49 —</td> </tr> </table> <p>Number of passes = <input type="text" value="6"/></p> <p>Incorrect number of passes. Correct answer: 5.</p> <p>Try a new question Submit New Question</p>	1	42	47	16	25	49	7	16	45	19	1 —	7 —	16 —	16 —	19 —	25 —	42 —	45 —	47 —	49 —
1	42	47	16	25	49	7	16	45	19															
1 —	7 —	16 —	16 —	19 —	25 —	42 —	45 —	47 —	49 —															
Submit a Bubble Sort question with only the number of passes filled in correctly.	Text will appear saying 'Incorrect order' and the incorrect score in the database is updated.	This ensures that the program can validate an incorrect answer as being incorrect.	4.2.j 4.2.h 4.2.i 4.2.j 4.2.q 4.2.l	<p>The functionality for updating the database has not been completed in this Iteration so that part cannot be tested.</p> <p>For filling in the values:</p> <p>Quiz</p> <p>Bubble Sort Prim's Dijkstra's Simplex</p> <p>1. Start at the first item in the list. 2. Compare the current item with the next item. 3. If the two items are in the wrong position, swap them. 4. Move up one item to the next time in the list. 5. Repeat from step 3 until all the unsorted items have been compared. 6. If any items were swapped, repeat from step 1. Otherwise, the algorithm is complete.</p> <p>Please sort the following unsorted dataset in ascending order and input the number of passes it took:</p> <table border="1"> <tr> <td>14</td> <td>20</td> <td>8</td> <td>35</td> <td>11</td> <td>36</td> <td>13</td> <td>46</td> <td>45</td> <td>9</td> </tr> <tr> <td>8 —</td> <td>11 —</td> <td>9 —</td> <td>14 —</td> <td>13 —</td> <td>20 —</td> <td>45 —</td> <td>36 —</td> <td>35 —</td> <td>46 —</td> </tr> </table> <p>Number of passes = <input type="text" value="8"/></p> <p>Incorrect order</p> <p>Try a new question Submit New Question</p>	14	20	8	35	11	36	13	46	45	9	8 —	11 —	9 —	14 —	13 —	20 —	45 —	36 —	35 —	46 —
14	20	8	35	11	36	13	46	45	9															
8 —	11 —	9 —	14 —	13 —	20 —	45 —	36 —	35 —	46 —															

Submit a Bubble Sort question with the dropdown menus and number of passes incorrectly filled in.	Text will appear saying 'Incorrect' and the incorrect score in the database is updated.	This ensures that the program can validate an incorrect answer as being incorrect.	4.2.j 4.2.h 4.2.i 4.2.q 4.2.l	<p>The functionality for updating the database has not been completed in this Iteration so that part cannot be tested.</p> <p>For filling in the values:</p>  <p>As expected.</p>
Press the New Question button on the Bubble Sort page.	A new dataset is randomised and displayed. The dropdown menus are cleared and the new dataset becomes the options. The number of passes entry field is cleared.	This ensures that the user can generate a new question of the Bubble Sort algorithm type when they press this button.	4.2.j 4.2.h 4.2.i 4.2.q 4.2.r	 <p>As expected.</p>

Press the Simplex button.	The Simplex button is highlighted in pink and text describing the algorithm is below this. Constraints are displayed as text and on a set of xy axes. An objective function is displayed as text. There are text boxes for the user to enter the optimal values of x y and P.	This ensures that the correct widgets required for the user to complete a Simplex algorithm question are displayed and that the randomised constraints and objective functionality work correctly.	4.2.h 4.2.i 4.2.j 4.2.q 4.2.y 4.2.v	 <p>Please enter either integers or fractions (e.g. in the form 3/5)</p> <p>Constraints: $3.0x + -6.0y \leq 4.0$ $4.0x + 2.0y \leq 8.0$ $-8.0x + -5.0y \leq 8.0$</p> <p>Objective: $P = 8.0x + 0.0y$</p> <p>x = <input type="text"/> y = <input type="text"/> P = <input type="text"/></p> <p>Submit New Question</p>	As expected.
Submit a Simplex algorithm question with the x y and P values correctly filled in.	Text will appear saying 'Correct' and the correct score in the database is updated.	This ensures that the program can validate a correct answer as being correct.	4.2.j 4.2.h 4.2.i 4.2.q 4.2.C	<p>The functionality for updating the database has not been completed in this iteration so that part cannot be tested.</p> <p>For filling in the values:</p>  <p>Please enter either integers or fractions (e.g. in the form 3/5)</p> <p>Constraints: $4.0x + -6.0y \leq 3.0$ $4.0x + 1.0y \leq 3.0$ $-7.0x + -3.0y \leq 5.0$</p> <p>Objective: $P = 5.0x + 0.0y$</p> <p>x = <input type="text"/> 3/4 y = <input type="text"/> 0 P = <input type="text"/> 15/4</p> <p>Submit New Question</p>	As expected.

Submit a Simplex algorithm question with only the x field filled in correctly.	Text will appear saying 'Incorrect' and the incorrect score in the database is updated.	This ensures that the program can validate an incorrect answer as being incorrect.	4.2.j 4.2.h 4.2.i 4.2.q 4.2.C	<p>The functionality for updating the database has not been completed in this iteration so that part cannot be tested.</p> <p>For filling in the values:</p>  <p>Please enter either integers or fractions (e.g. in the form 3/5)</p> <p>Constraints:</p> <ul style="list-style-type: none"> -4.0x + 1.0y ≤ 9.0 4.0x + 1.0y ≤ 3.0 -7.0x + -3.0y ≤ 9.0 Objective: P = 8.0x + 0.0y <p>x = <input type="text" value="0"/> y = <input type="text" value="0"/> P = <input type="text" value="0"/></p> <p>Incorrect</p> <p>Try a new question.</p> <p>Submit New Question</p>
Submit a Simplex algorithm question with only the y field filled in correctly.	Text will appear saying 'Incorrect' and the incorrect score in the database is updated.	This ensures that the program can validate an incorrect answer as being incorrect.	4.2.j 4.2.h 4.2.i 4.2.q 4.2.C	<p>The functionality for updating the database has not been completed in this iteration so that part cannot be tested.</p> <p>For filling in the values:</p>  <p>Please enter either integers or fractions (e.g. in the form 3/5)</p> <p>Constraints:</p> <ul style="list-style-type: none"> -4.0x + 1.0y ≤ 9.0 4.0x + 1.0y ≤ 3.0 -7.0x + -3.0y ≤ 9.0 Objective: P = -1.0x + 1.0y <p>x = <input type="text" value="0"/> y = <input type="text" value="0"/> P = <input type="text" value="0"/></p> <p>Incorrect</p> <p>Try a new question.</p> <p>Submit New Question</p>

Submit a Simplex algorithm question with only the P field filled in correctly.	Text will appear saying 'Incorrect' and the incorrect score in the database is updated.	This ensures that the program can validate an incorrect answer as being incorrect.	4.2.j 4.2.h 4.2.i 4.2.q 4.2.C	<p>The functionality for updating the database has not been completed in this Iteration so that part cannot be tested.</p> <p>For filling in the values:</p>  <p>The correct values were $x=0$ $y=3$ $P=6$. As expected.</p>
Submit a Simplex algorithm question with the x y and P fields incorrectly filled in.	Text will appear saying 'Incorrect' and the incorrect score in the database is updated.	This ensures that the program can validate an incorrect answer as being incorrect.	4.2.j 4.2.h 4.2.i 4.2.q 4.2.C	<p>The functionality for updating the database has not been completed in this Iteration so that part cannot be tested.</p> <p>For filling in the values:</p>  <p>The correct values were $x=0$ $y=3$ $P=24$. As expected.</p>

Press the New Question button on the Simplex page.	A new set of constraints and objective function are randomised and displayed. The x y and P entry fields are cleared.	This ensures that the user can generate a new question of the Simplex algorithm type when they press this button.	4.2.j 4.2.h 4.2.i 4.2.q 4.2.J	 <p>As expected.</p>
Press the Home button.	The Quiz page is closed and the Home page is opened.	This ensures that the 'Home' button works correctly and that the user can go back to the Home page to either log out or go to the visualisation pages.	4.2.c	 <p>As expected.</p>

All white box tests have passed successfully so development and testing in Prototype 3 Iteration 3 is complete.

ITERATION 2 - DEVELOPER REVIEW

This iteration started off quite simple with the quiz section for Bubble Sort, with no major issues arising during development. However, the quiz section for the Simplex algorithm was very challenging due to issues with ensuring that the randomised constraints had a feasible region and then also making sure that the tableau worked correctly with the integer values, which I had to cast the floats. All these challenges were quite likely due to the complexity of the Simplex algorithm, which also made it difficult to test as I had to use other systems to find the optimal x y and P values as calculating it by hand would take too much time, which I don't have due to the time constraints on this project.

ITERATION 3 - QUIZ PAGE: PRIM'S AND DIJKSTRA'S

In this iteration, I will set up the GUI for the Prim's and Dijkstra's questions and complete the functionality for them. I will also complete the functionality for updating scores in the database. To begin, I wrote the code for the randomGraph method, which is based off the Prim's algorithm method of creating an MST and then adding extra edges:

4.3.a

```
def randomGraph(self):
    # connected graph of 6-8 vertices labelled A, B, C...
    # weights between 1 and 50
    print("in create random graph")

    numVertices = randint(6,8)
    vertices = [chr(65+i) for i in range(numVertices)]
    edges = []
    startVertex = vertices[0]
    visited = set([startVertex])
    queue = []

    # all edges from start vertex to queue
    for v in vertices:
        if v != startVertex:
            weight = randint(1,50)
            heapq.heappush(queue, (weight, startVertex, v))

    while queue:
        weight, u, v = heapq.heappop(queue)
        # has been visited - add to MST
        if v not in visited:
            visited.add(v)
            edges.append((u, v, weight))
            for w in vertices:
                if w != v and w not in visited:
                    weight = randint(1,50)
                    heapq.heappush(queue, (weight, v, w))
        if len(visited) == numVertices:
            break

    # add random extra edges:
    extraEdges = randint(4, 10)
    allPossEdges = [(u,v) for u in vertices for v in vertices if u<v]
    shuffle(allPossEdges)
    for u, v in allPossEdges:
        if len(edges) >= extraEdges:
            break
        if (u,v) not in edges and (v,u) not in edges:
            weight = randint(1,50)
            edges.append((u,v,weight))

    print(f"Vertices generated: {vertices}")
    print(f"Edges generated: {edges}")

    self.graph = nx.Graph()
    self.graph.add_weighted_edges_from(edges)

    print(f"self.graph nodes: {self.graph.nodes}")
    print(f"self.graph edges: {self.graph.edges}")
```

Then, I set up the widgets for the Prim quiz page:

4.3.b

```
def primQuizWidgets(self):
    # widgets for prim frame
    print("in prim quiz widgets")

    self.primButton["bg"] = "#b2675e"

    self.primQuizFrame = Frame(self.master, bg="#eaebcd")
    self.primQuizFrame.grid(row=2, column=0, columnspan=12, rowspan=8)

    Label(self.primQuizFrame,
          text="1. Choose any vertex to start the tree.\n"
               "2. Choose the edge of least weight that joins a vertex that is already in the tree to a vertex that is not yet in the tree.\n"
               "3. Repeat step 2 until all the vertices are connected to the tree.",
          font=fontSmall,
          fg="#1b263b",
          bg="#eaebcd",
          justify="left").grid(row=2, column=0, columnspan=12, sticky="w", padx=10, pady=5)

    Label(self.primQuizFrame,
          text="Start at vertex A.\n Input the MST edges below in the form 'AB, BC' etc:",
          font=fontMedium,
          fg="#1b263b",
          bg="#eaebcd",
          justify="left").grid(row=4, column=6, columnspan=6, rowspan=2)

    self.mstEntry = StringVar()
    Entry(self.primQuizFrame, textvariable=self.mstEntry).grid(row=6, column=6, columnspan=6, rowspan=2)

    Label(self.primQuizFrame, text="MST weight = ", font=fontMedium, fg="#1b263b", bg="#eaebcd").grid(row=8, column=6, columnspan=3)
    self.weightEntry = IntVar()
    Entry(self.primQuizFrame, textvariable=self.weightEntry).grid(row=8, column=9, columnspan=3)

    self.answerText = Label(self.primQuizFrame, text="correct/incorrect text", font=fontMedium, fg="#990000", bg="#eaebcd")
    self.answerText.grid(row=9, column=0, columnspan=6)
```

4.3.c

```

Button(self.primQuizFrame, text="Submit", command=self.check, bg="#1b263b", fg="white",
       font=fontMedium, width=20).grid(row=9, column=6, columnspan=3, padx=5, pady=5)
Button(self.primQuizFrame, text="New Question", command=lambda: self.newQuestion("Prim"), bg="#1b263b", fg="white",
       font=fontMedium, width=20).grid(row=9, column=9, columnspan=3, padx=5, pady=5)

print(f"graph: {self.graph}")
# create axes to plot graph on
self.fig, self.ax = plt.subplots(figsize=(4,4))
self.primQuizGraphCanvas = FigureCanvasTkAgg(self.fig, self.primQuizFrame)
self.primQuizGraphCanvas.get_tk_widget().grid(row=4, column=0, rowspan=6, columnspan=6)
# draw graph on axes
pos = nx.spring_layout(self.graph)
nx.draw(self.graph, pos, with_labels=True, node_color='#606c38', edge_color='black', node_size=500, font_color='white', ax=self.ax)
edge_labels = nx.get_edge_attributes(self.graph, 'weight')
nx.draw_networkx_edge_labels(self.graph, pos, edge_labels=edge_labels, ax=self.ax)
self.primQuizGraphCanvas.draw()

```

Note that I put a `spring_layout` as the layout for the graph initially. I may change this if I don't like it. Then I extended the `newType` method to include Prim's:

4.3.d

```

def newType(self, newQType):
    # qType = question type ---> bubble Sort, prim, dijkstra, simplex
    # forget currentQType ---> None initially
    print(f"current question type: {self.currentQType}, new question type: {newQType}")
    self.imageLabel.grid_forget()
    # change current q type button background

    if newQType == "Bubble Sort":
        self.currentQType = "Bubble Sort"
        self.bubbleSortDataset()
        self.bubbleSortQuizWidgets()

    elif newQType == "Prim":
        self.currentQType = "Prim"
        self.randomGraph()
        self.primQuizWidgets()

```

When I run the code, the following is output:

Quiz

Home

Bubble Sort Prim's Dijkstra's Simplex

1. Choose any vertex to start the tree.
 2. Choose the edge of least weight that joins a vertex that is already in the tree to a vertex that is not yet in the tree.
 3. Repeat step 2 until all the vertices are connected to the tree.

Start at vertex A.
 Input the MST edges below in the form 'AB, BC' etc:

MST weight =

Submit **New Question**

```
in add constraint
in add constraint
current question type: None, new question type: Prim
in create_random_graph
Vertices generated: ['A', 'B', 'C', 'D', 'E', 'F']
Edges generated: [('A', 'D', 2), ('A', 'B', 5), ('B', 'F', 5), ('A', 'E', 14), ('E', 'C', 19), ('B', 'C', 37)]
self.graph.nodes: ['A', 'D', 'B', 'F', 'E', 'C']
self.graph.edges: [('A', 'D'), ('A', 'B'), ('A', 'E'), ('B', 'F'), ('B', 'C'), ('E', 'C')]
in prim_quiz_widgets
graph: Graph with 6 nodes and 6 edges
```

The answerText is not visible. To fix this, I corrected the graph grid placement. Now, I am going to complete the functionality for the check method, specific to the Prim's algorithm quiz:

4.3.e

```
elif self.currentQType == "Prim":
    print("prim")
    self.prim.graph = self.quizGraph.copy()
    #self.prim.startVertexChoice = "A"
    self.prim.primSteps(start="A")
    correctMST = set(tuple(sorted(edge)) for edge in self.prim.steps[-1]) # so edges are in alphabetical order AB instead of BA
    correctWeight = sum(self.quizGraph[u][v]["weight"] for u,v in correctMST)

    print(f"Correct MST: {correctMST}")
    print(f"Correct weight: {correctWeight}")
    # remove spaces, split by commas
    userMSTInput = self.mstEntry.get().replace(" ", "").split(",")
    userMST = set()
    for edge in userMSTInput:
        if len(edge) == 2:
            userMST.add(tuple(sorted(edge))) # store as sorted tuple - AB same as BA
    try:
        userWeight = int(self.weightEntry.get())
    except ValueError:
        userWeight = -1 # invalid input

    print(f"User MST: {userMST}")
    print(f"User weight: {userWeight}")
    # check correctness:
    if userMST == correctMST and userWeight == correctWeight:
        self.answerText["text"] = "Correct :)"
        self.answerText["fg"] = "#008000"
        self.updateScore("Correct")
    elif userMST != correctMST:
        self.answerText["text"] = "Incorrect MST edges\nTry a new question"
        self.answerText["fg"] = "#990000"
        self.updateScore("Incorrect")
    else:
        self.answerText["text"] = f"Incorrect MST weight. Correct answer: {correctWeight}"
        self.answerText["fg"] = "#990000"
        self.updateScore("Incorrect")
```

For this, I adjusted the primSteps method to take start as a flag:

4.3.f

```
def primSteps(self, start=None):
    #print("in prim steps")

    self.steps = []
    self.mstEdges = []
    if start:
        startVertex = start
    else:
        startVertex = self.startVertexChoice.get()
    visited = set() # stores mst edges in an unordered, unchangeable and unindexed format
    minHeap = [] # this is the queue for the edges connected to the current vertex

    visited.add(startVertex)
```

Now when I run the code and input the correct answer, the following is output:

```
in add constraint
in add constraint
current question type: None, new question type: Prim
in create random graph
in prim quiz widgets
graph: Graph with 8 nodes and 15 edges
in check
prim
Correct MST: {('B', 'D'), ('A', 'H'), ('E', 'H'), ('B', 'G'), ('C', 'D'), ('B', 'F'), ('E', 'G')}
Correct weight: 65
User MST: {('B', 'D'), ('A', 'H'), ('E', 'H'), ('B', 'G'), ('C', 'D'), ('B', 'F'), ('E', 'G')}
User weight: 65
in update score
```

Quiz Home

Bubble Sort Prim's Dijkstra's Simplex

1. Choose any vertex to start the tree.
2. Choose the edge of least weight that joins a vertex that is already in the tree to a vertex that is not yet in the tree.
3. Repeat step 2 until all the vertices are connected to the tree.

Start at vertex A.
Input the MST edges below in the form 'AB, BC' etc:

H, HE, EG, GB, BD, DC, BF

MST weight = 65

Correct :) Submit New Question

This is all as expected so it seems that the code functions correctly. I then adjusted the entry field length and now the following is output:

Quiz Home

Bubble Sort Prim's Dijkstra's Simplex

1. Choose any vertex to start the tree.
2. Choose the edge of least weight that joins a vertex that is already in the tree to a vertex that is not yet in the tree.
3. Repeat step 2 until all the vertices are connected to the tree.

Start at vertex A.
Input the MST edges below in the form 'AB, BC' etc:

correct/incorrect text

MST weight = 0

Submit New Question

Now, the final section to develop for Prim's quiz is the New Question method:

4.3.g

```

elif qType == "Prim":
    print("prim")
    self.randomGraph()

    self.ax.clear()
    pos = nx.spring_layout(self.quizGraph)
    nx.draw(self.quizGraph, pos, with_labels=True, node_color="#606c38", edge_color='black', node_size=200, font_color='white', font_size=11, ax=self.ax)
    edge_labels = nx.get_edge_attributes(self.quizGraph, 'weight')
    nx.draw_networkx_edge_labels(self.quizGraph, pos, edge_labels=edge_labels, ax=self.ax)

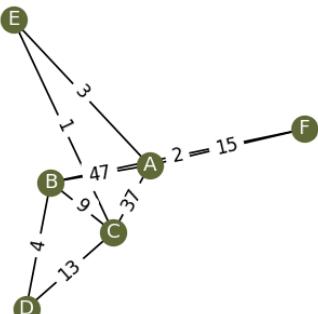
    self.mstEntry.set("")
    self.weightEntry.set(0)

```

When I run the code, the following is output before and after pressing the New Question button:

Bubble Sort	Prim's	Dijkstra's	Simplex
-------------	--------	------------	---------

1. Choose any vertex to start the tree.
 2. Choose the edge of least weight that joins a vertex that is already in the tree to a vertex that is not yet in the tree.
 3. Repeat step 2 until all the vertices are connected to the tree.

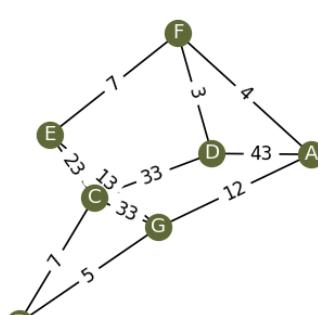


Start at vertex A.
 Input the MST edges below in the form 'AB, BC' etc:

MST weight =

correct/incorrect text	Submit	New Question	
Bubble Sort	Prim's	Dijkstra's	Simplex

1. Choose any vertex to start the tree.
 2. Choose the edge of least weight that joins a vertex that is already in the tree to a vertex that is not yet in the tree.
 3. Repeat step 2 until all the vertices are connected to the tree.



Start at vertex A.
 Input the MST edges below in the form 'AB, BC' etc:

MST weight =

These outputs show that it functions correctly.

Now, I am going to complete the functionality for the Dijkstra quiz section. First, I completed the GUI:

4.3.h

```
def dijkstraQuizWidgets(self):
    # widgets for dijkstra frame
    print("in dijkstra quiz widgets")
    self.dijkstraButton["bg"] = "#b2675e"

    self.dijkstraQuizFrame = Frame(self.master, bg="#eaebed")
    self.dijkstraQuizFrame.grid(row=2, column=0, columnspan=12, rowspan=8)

    Label(self.dijkstraQuizFrame, text=
        "1. Set the initial distance from the start values for all nodes (0 for the start node and infinity for all other nodes).\n"
        "2. Find the node with the shortest distance from the start that has not been visited and look at all the unvisited connected nodes.\n"
        "3. Calculate the distance from the start for each of the unvisited connected nodes. \n"
        "4. If the distance calculated is less than the current shortest distance from the start, set the previous node for that connected\nnode to be the current node. \n"
        "5. Then set the current node as visited.\n"
        "6. Repeat steps 2 to 5 until all nodes are set to visited.\n"
        "To find the shortest path through backtracking:\n"
        "7. Start from the goal node.\n"
        "8. Add the 'previous node' to the start of a list.\n"
        "9. Repeat step 7 until the start node is reached.",
        font=fontSmall,
        fg="#1b263b",
        bg="#eaebed",
        justify="left").grid(row=2, column=0, columnspan=12, sticky="w", padx=10, pady=5)

    Label(self.dijkstraQuizFrame,
        text=f"Start at vertex A. End vertex {self.endVertex}.\nInput the shortest path in the form 'ABC...':",
        font=fontMedium,
        fg="#1b263b",
        bg="#eaebed",
        justify="left").grid(row=4, column=6, columnspan=6, rowspan=2)

    self.shortestPathEntry = StringVar()
    Entry(self.dijkstraQuizFrame, textvariable=self.shortestPathEntry).grid(row=6, column=6, columnspan=6, rowspan=2)

    Label(self.dijkstraQuizFrame, text="Shortest path = ", font=fontMedium, fg="#1b263b", bg="#eaebed").grid(row=8, column=6, columnspan=3)
    self.weightEntry = IntVar()
    Entry(self.dijkstraQuizFrame, textvariable=self.weightEntry).grid(row=8, column=9, columnspan=3)

    self.answerText = Label(self.dijkstraQuizFrame, text="correct/incorrect text", font=fontMedium, fg="#990000", bg="#eaebed")
    self.answerText.grid(row=9, column=0, columnspan=6)
```

4.3.i

```

Button(self.dijkstraQuizFrame, text="Submit", command=self.check, bg="#1b263b", fg="white",
       font=fontMedium, width=20).grid(row=9, column=6, columnspan=3, padx=5, pady=5)
Button(self.dijkstraQuizFrame, text="New Question", command=lambda: self.newQuestion("Dijkstra"), bg="#1b263b", fg="white",
       font=fontMedium, width=20).grid(row=9, column=9, columnspan=3, padx=5, pady=5)

#print(f"graph: {self.graph}")
# create axes to plot graph on
self.fig, self.ax = plt.subplots(figsize=(3.5,3.5))
self.dijkstraQuizGraphCanvas = FigureCanvasTkAgg(self.fig, self.dijkstraQuizFrame)
self.dijkstraQuizGraphCanvas.get_tk_widget().grid(row=4, column=0, rowspan=5, columnspan=6)
# draw graph on axes
pos = nx.spring_layout(self.quizGraph)
nx.draw(self.quizGraph, pos, with_labels=True, node_color='#606c38', edge_color='black', node_size=200, font_color='white', font_size=11, ax=self.ax)
edge_labels = nx.get_edge_attributes(self.quizGraph, 'weight')
nx.draw_networkx_edge_labels(self.quizGraph, pos, edge_labels=edge_labels, ax=self.ax)
self.dijkstraQuizGraphCanvas.draw()

```

And for the newType method:

4.3.j

```

elif newQType == "Dijkstra":
    self.currentQType = "Dijkstra"
    self.randomGraph()
    self.dijkstraQuizWidgets()

```

For checking the input, I again added a flag for the dijkstraSteps method:

4.3.k

```

def dijkstraSteps(self, startV=None, endV=None):
    print("in dijkstra steps")
    if startV:
        start = startV
    else:
        start = self.startVertexChoice.get()

    if endV:
        end = endV
    else:
        end = self.endVertexChoice.get()

    self.distances = {node: float("inf") for node in self.graph.nodes}
    self.previous = {node: None for node in self.graph.nodes}
    self.distances[start] = 0

```

Then for the check method:

4.3.1

```
elif self.currentQType == "Dijkstra":
    print("dijkstra")
    self.dijkstra.graph = self.quizGraph.copy()
    self.dijkstra.dijkstraSteps(startV="A", endV=self.endVertex)
    correctPath = "".join(self.dijkstra.steps[-1][1]) # last tuple, 2nd part
    correctWeight = self.dijkstra.distances[self.endVertex]

    userPath = self.shortestPathEntry.get().strip().upper()
    try:
        userWeight = int(self.weightEntry.get())
    except ValueError:
        userWeight = -1

    # check correctness:
    if userPath == correctPath and userWeight == correctWeight:
        self.answerText["text"] = "Correct :)"
        self.answerText["fg"] = "#008000"
        self.updateScore("Correct")
    elif userPath != correctPath:
        self.answerText["text"] = "Incorrect path\nTry a new question"
        self.answerText["fg"] = "#990000"
        self.updateScore("Incorrect")
    else:
        self.answerText["text"] = f"Incorrect path weight. Correct answer: {correctWeight}"
        self.answerText["fg"] = "#990000"
        self.updateScore("Incorrect")
```

When I run the code the following is output:

1. Set the initial distance from the start values for all nodes (0 for the start node and infinity for all other nodes).
 2. Find the node with the shortest distance from the start that has not been visited and look at all the unvisited connected nodes.
 3. Calculate the distance from the start for each of the unvisited connected nodes.
 4. If the distance calculated is less than the current shortest distance from the start, set the previous node for that connected node to be the current node.
 5. Then set the current node as visited.
 6. Repeat steps 2 to 5 until all nodes are set to visited.
 To find the shortest path through backtracking:
 7. Start from the goal node.
 8. Add the 'previous node' to the start of a list.
 9. Repeat step 7 until the start node is reached.

From the above output, it is clear that the checking functionality works correctly. Now for the newQuestion functionality:

4.3.m

```

elif qType == "Dijkstra":
    print("dijkstra")
    self.randomGraph()

    self.ax.clear()
    pos = nx.spring_layout(self.quizGraph)
    nx.draw(self.quizGraph, pos, with_labels=True, node_color="#606c38", edge_color='black', node_size=200, font_color='white', font_size=11, ax=self.ax)
    edge_labels = nx.get_edge_attributes(self.quizGraph, 'weight')
    nx.draw_networkx_edge_labels(self.quizGraph, pos, edge_labels=edge_labels, ax=self.ax)
    self.dijkstraQuizGraphCanvas.draw()

    self.shortestPathEntry.set("")
    self.weightEntry.set(0)

```

When I run the code and then press the New Question button the following is output:

Bubble Sort Prim's **Dijkstra's** Simplex

1. Set the initial distance from the start values for all nodes (0 for the start node and infinity for all other nodes).
 2. Find the node with the shortest distance from the start that has not been visited and look at all the unvisited connected nodes.
 3. Calculate the distance from the start for each of the unvisited connected nodes.
 4. If the distance calculated is less than the current shortest distance from the start, set the previous node for that connected node to be the current node.
 5. Then set the current node as visited.
 6. Repeat steps 2 to 5 until all nodes are set to visited.
 To find the shortest path through backtracking:
 7. Start from the goal node.
 8. Add the 'previous node' to the start of a list.
 9. Repeat step 7 until the start node is reached.

Start at vertex A. End vertex G.
 Input the shortest path in the form 'ABC...':

Shortest path =

correct/incorrect text **Submit** **New Question**

Now all functionality for Dijkstra's quiz works correctly. The final functionality needed to be developed for the Quiz section is updating the scores in the database. For this I need to implement the updateScore method and include the SQL statements described in the Design section.

4.3.n

```

def updateScore(self, answer):
    # answer = Correct or Incorrect
    #print("in update score")
    usernameHolder = self.username
    algorithmHolder = self.currentQType
    if answer == "Correct":
        self.cursor.execute("UPDATE StudentEnrolment SET CorrectScore = CorrectScore+1 WHERE Username = ? AND Algorithm=?;", (usernameHolder, algorithmHolder))
        self.conn.commit()
    elif answer == "Incorrect":
        self.cursor.execute("UPDATE StudentEnrolment SET IncorrectScore = IncorrectScore+1 WHERE Username = ? AND Algorithm=?;", (usernameHolder, algorithmHolder))
        self.conn.commit()
    #print("score updated")

```

This outputs the following:

Quiz Home

Bubble Sort Prim's **Dijkstra's** Simplex

1. Start at the first item in the list.
 2. Compare the current item with the next item.
 3. If the two items are in the wrong position, swap them.
 4. Move up one item to the next time in the list.
 5. Repeat from step 3 until all the unsorted items have been compared.
 6. If any items were swapped, repeat from step 1. Otherwise, the algorithm is complete.

Please sort the following unsorted dataset in ascending order and input the number of passes it took:

50	50	3	37	1	48	10	39	21	38
1 ↗	3 ↗	10 ↗	21 ↗	37 ↗	38 ↗	39 ↗	48 ↗	50 ↗	50 ↗

Number of passes =

Correct :) **Submit** **New Question**

in add constraint
in add constraint
in update score
score updated

To check that the score has been correctly updated, when I open the database in DB Browser for SQLite, the following is output:

	StudentEnrolID	Username	Algorithm	CorrectScore	IncorrectScore
	Filter	Filter	Filter	Filter	Filter
1	1	Test_accS	Bubble Sort	3	2
2	2	Test_accS	Simplex	1	5
3	3	Billy123	Bubble Sort	0	0
4	4	HelloWord!!!	Bubble Sort	3	6
5	5	HelloWord!!!	Prim's	4	1
6	6	HelloWord!!!	Dijkstra's	5	0
7	7	HelloWord!!!	Simplex	0	3

This means that the score has been correctly updated because this was the previous state:

Table: StudentEnrolment					
	StudentEnrolID	Username	Algorithm	CorrectScore	IncorrectScore
	Filter	Filter	Filter	Filter	Filter
1	1	Test_accS	Bubble Sort	3	2
2	2	Test_accS	Simplex	1	5
3	3	Billy123	Bubble Sort	0	0
4	4	HelloWor...	Bubble Sort	2	6
5	5	HelloWor...	Prim's	4	1
6	6	HelloWor...	Dijkstra's	5	0
7	7	HelloWor...	Simplex	0	3

Now to check than an incorrect input results in the database being updated:

Quiz

Home

Bubble Sort Prim's Dijkstra's Simplex

1. Set the initial distance from the start values for all nodes (0 for the start node and infinity for all other nodes).
 2. Find the node with the shortest distance from the start that has not been visited and look at all the unvisited connected nodes.
 3. Calculate the distance from the start for each of the unvisited connected nodes.
 4. If the distance calculated is less than the current shortest distance from the start, set the previous node for that connected node to be the current node.
 5. Then set the current node as visited.
 6. Repeat steps 2 to 5 until all nodes are set to visited.
 To find the shortest path through backtracking:
 7. Start from the goal node.
 8. Add the 'previous node' to the start of a list.
 9. Repeat step 7 until the start node is reached.

Start at vertex A. End vertex G.
 Input the shortest path in the form 'ABC...':

ABCD

Shortest path = 10

Incorrect path
Try a new question

Submit New Question

	StudentEnrolID	Username	Algorithm	CorrectScore	IncorrectScore
	Filter	Filter	Filter	Filter	Filter
1	1	Test_accS	Bubble Sort	3	2
2	2	Test_accS	Simplex	1	5
3	3	Billy123	Bubble Sort	0	0
4	4	HelloWord!!!	Bubble Sort	3	6
5	5	HelloWord!!!	Prim's	4	1
6	6	HelloWord!!!	Dijkstra's	5	1
7	7	HelloWord!!!	Simplex	0	3

As seen in the database before, for Dijkstra's the IncorrectScore was 0 but now is 1, meaning that the updating of the scores works correctly.

Now, I am going to make sure that only the selected algorithm buttons are displayed on the screen for the Quiz section:

4.3.o

```
def accountAlgorithms(self):
    try:
        usernameHolder = self.username
        self.cursor.execute("SELECT Algorithm FROM StudentEnrolment WHERE Username=?;", (usernameHolder,))
        result = self.cursor.fetchall()
        print(f"sql result: {result}")
        algorithmsList = [row[0] for row in result]
        print(f"algorithms list: {algorithmsList}")
        return algorithmsList
    except sqlite3.Error as e:
        print(f"Database error: {e}")
```

In the constructor method:

```
self.userAlgorithmLocations = self.accountAlgorithms()

self.initialQuizWidgets()
```

And in the initial GUI set up method:

4.3.p

```
if "Bubble Sort" in self.userAlgorithmLocations:
    self.bubbleSortButton = Button(self.master, text="Bubble Sort", command=lambda: self.newType("Bubble Sort"), bg="#1b263b", fg="white", font=fontMedium, width=20)
    self.bubbleSortButton.grid(row=1, column=0, columnspan=3, padx=5, pady=5)

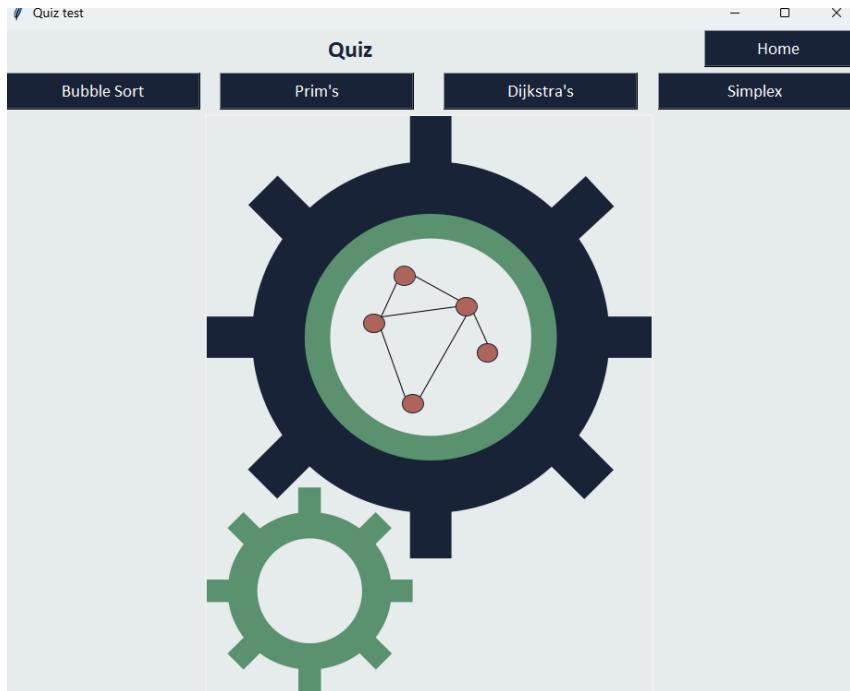
if "Prim's" in self.userAlgorithmLocations:
    primColumn = self.userAlgorithmLocations.index("Prim's") * 3
    self.primButton = Button(self.master, text="Prim's", command=lambda: self.newType("Prim's"), bg="#1b263b", fg="white", font=fontMedium, width=20)
    self.primButton.grid(row=1, column=primColumn, columnspan=3, padx=5, pady=5)

if "Dijkstra's" in self.userAlgorithmLocations:
    dijkstraColumn = self.userAlgorithmLocations.index("Dijkstra's") * 3
    self.dijkstraButton = Button(self.master, text="Dijkstra's", command=lambda: self.newType("Dijkstra's"), bg="#1b263b", fg="white", font=fontMedium, width=20)
    self.dijkstraButton.grid(row=1, column=dijkstraColumn, columnspan=3, padx=5, pady=5)

if "Simplex" in self.userAlgorithmLocations:
    simplexColumn = self.userAlgorithmLocations.index("Simplex") * 3
    self.simplexButton = Button(self.master, text="Simplex", command=lambda: self.newType("Simplex"), bg="#1b263b", fg="white", font=fontMedium, width=20)
    self.simplexButton.grid(row=1, column=simplexColumn, columnspan=3, padx=5, pady=5)
```

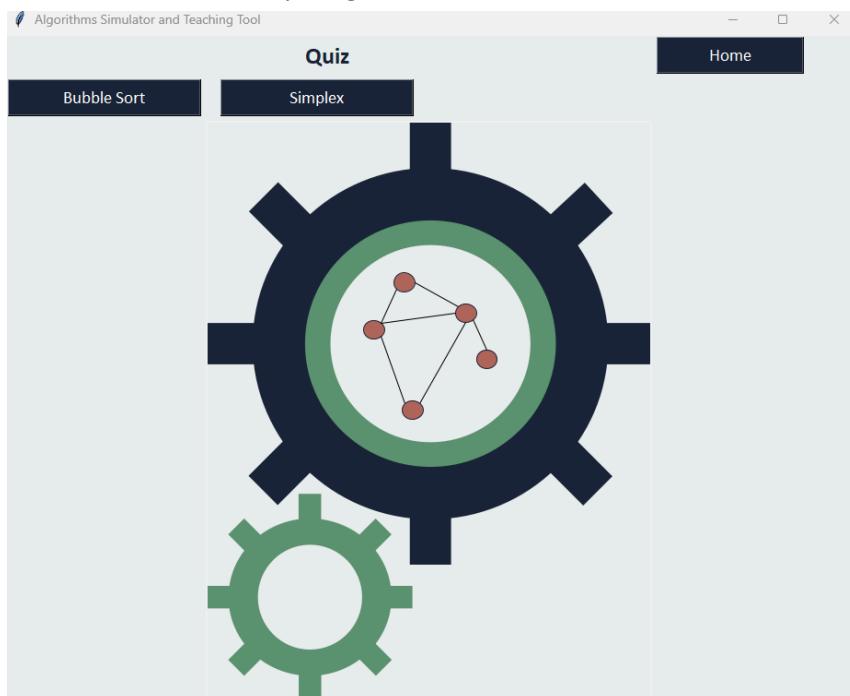
Having the if statements means that only the buttons for the algorithms chosen by the user are displayed, meaning that only that functionality is able to be accessed.

For an account with all 4 algorithms chosen:



```
sql result: [('Bubble Sort',), ("Prim's",), ("Dijkstra's",), ('Simplex',)]
algorithms list: ['Bubble Sort', "Prim's", "Dijkstra's", 'Simplex']
```

For an account with only 2 algorithms chosen:



```
sql result: [('Bubble Sort',), ('Simplex',)]
algorithms list: ['Bubble Sort', 'Simplex']
```

This is all as expected.

Finally, I am going to configure the size of the window for each question type so that the GUI looks as refined and user friendly as possible:

4.3.q

```
def newType(self, newQType):
    # qType = question type --> bubble Sort, prim, dijkstra, simplex
    # forget currentQType --> None initially
    #printf("current question type: {self.currentQType}, new question type: {newQType}")
    self.imageLabel.grid_forget()

    # change current button colour to blue, forget current question widgets
    if self.currentQType == "Bubble Sort":
        self.bubbleSortButton["bg"] = "#1b263b"
        for widget in self.bubbleSortQuizFrame.winfo_children():
            widget.grid_forget()

    elif self.currentQType == "Prim's":
        self.primButton["bg"] = "#1b263b"
        for widget in self.primQuizFrame.winfo_children():
            widget.grid_forget()

    elif self.currentQType == "Dijkstra's":
        self.dijkstraButton["bg"] = "#1b263b"
        for widget in self.dijkstraQuizFrame.winfo_children():
            widget.grid_forget()

    elif self.currentQType == "Simplex":
        self.simplexButton["bg"] = "#1b263b"
        for widget in self.simplexQuizFrame.winfo_children():
            widget.grid_forget()
```

4.3.r

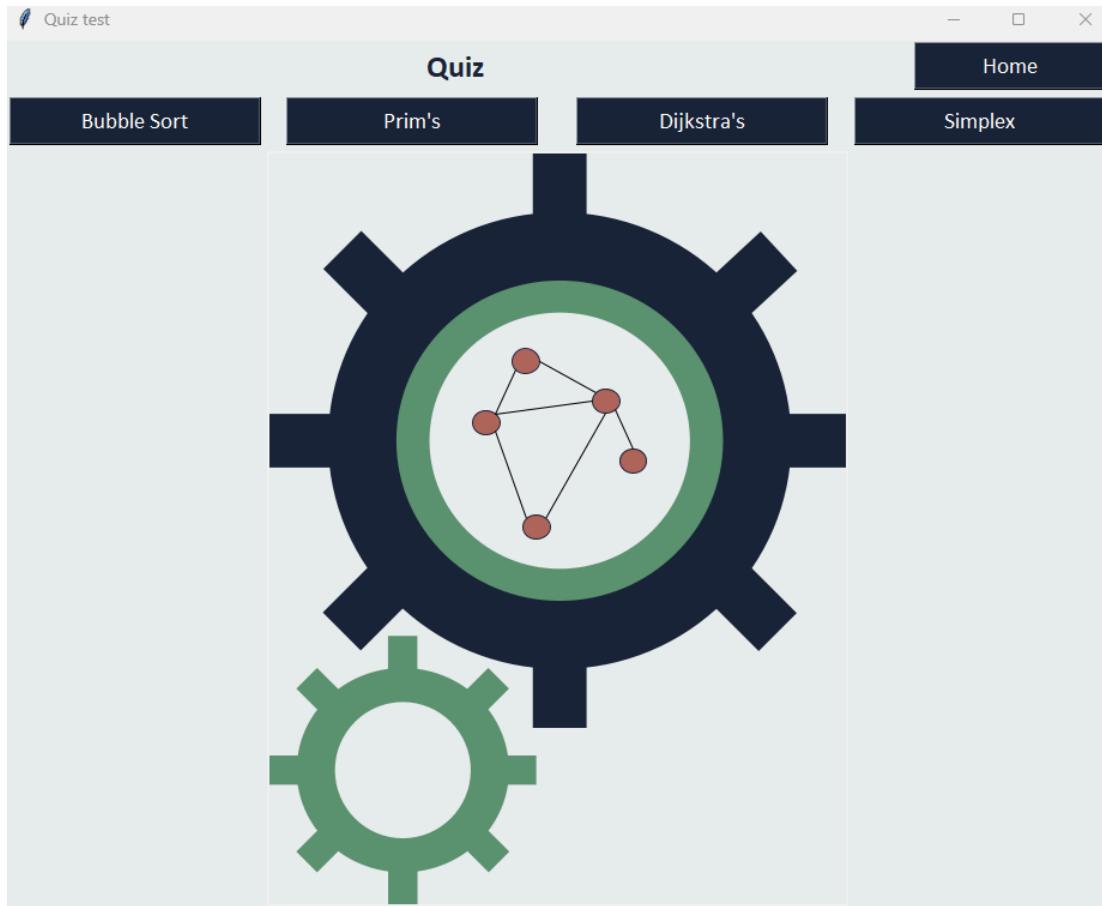
```
# new question widgets:
if newQType == "Bubble Sort":
    self.currentQType = "Bubble Sort"
    self.bubbleSortDataset()
    self.bubbleSortQuizWidgets()
    self.master.geometry("880x440")

elif newQType == "Prim's":
    self.currentQType = "Prim's"
    self.randomGraph()
    self.primQuizWidgets()
    self.master.geometry("880x650")

elif newQType == "Dijkstra's":
    self.currentQType = "Dijkstra's"
    self.randomGraph()
    self.dijkstraQuizWidgets()
    self.master.geometry("880x795")

elif newQType == "Simplex":
    self.currentQType = "Simplex"
    self.simplexConstraints()
    self.simplexQuizWidgets()
    self.master.geometry("965x795")
```

When I run the code, the following is output:

A screenshot of the same "Quiz test" window, but the "Bubble Sort" tab is now highlighted in red. The main content area contains the following text:

1. Start at the first item in the list.
2. Compare the current item with the next item.
3. If the two items are in the wrong position, swap them.
4. Move up one item to the next time in the list.
5. Repeat from step 3 until all the unsorted items have been compared.
6. If any items were swapped, repeat from step 1. Otherwise, the algorithm is complete.

Please sort the following unsorted dataset in ascending order and input the number of passes it took:

32	6	5	1	19	12	8	47	49	13
0 ↗	0 ↗	0 ↗	0 ↗	0 ↗	0 ↗	0 ↗	0 ↗	0 ↗	0 ↗

Number of passes =

Submit **New Question**

Quiz test

Quiz

Home

[Bubble Sort](#) [Prim's](#) [Dijkstra's](#) [Simplex](#)

1. Choose any vertex to start the tree.
 2. Choose the edge of least weight that joins a vertex that is already in the tree to a vertex that is not yet in the tree.
 3. Repeat step 2 until all the vertices are connected to the tree.

The graph consists of 8 nodes labeled A through H. The edges and their weights are: A-E (12), A-H (7), E-G (1), E-H (4), G-F (5), G-H (3), H-B (5), H-C (7), B-C (4), D-B (6), D-C (5), and F-C (3).

Start at vertex A.
 Input the MST edges below in the form 'AB, BC' etc:

MST weight =

[Submit](#) [New Question](#)

Quiz test

Quiz

Home

[Bubble Sort](#) [Prim's](#) [Dijkstra's](#) [Simplex](#)

1. Set the initial distance from the start values for all nodes (0 for the start node and infinity for all other nodes).
 2. Find the node with the shortest distance from the start that has not been visited and look at all the unvisited connected nodes.
 3. Calculate the distance from the start for each of the unvisited connected nodes.
 4. If the distance calculated is less than the current shortest distance from the start, set the previous node for that connected node to be the current node.
 5. Then set the current node as visited.
 6. Repeat steps 2 to 5 until all nodes are set to visited.
 To find the shortest path through backtracking:
 7. Start from the goal node.
 8. Add the 'previous node' to the start of a list.
 9. Repeat step 7 until the start node is reached.

The graph consists of 8 nodes labeled A through H. The edges and their weights are: A-B (1), A-C (5), A-D (9), A-E (1), A-F (1), B-C (2), B-E (10), C-D (3), C-F (38), C-G (4), D-G (23), E-F (34), E-G (2), F-G (17), F-H (5), and G-H (1).

Start at vertex A. End vertex H.
 Input the shortest path in the form 'ABC...':

Shortest path =

[Submit](#) [New Question](#)

Quiz test

Quiz

[Home](#)

[Bubble Sort](#) [Prim's](#) [Dijkstra's](#) [Simplex](#)

1. Turn the inequalities into equalities by adding a slack variable. Rearrange the objective function to be equal to 0. Input these equations into the simplex tableau.
 2. Choose the most negative value in the objective row to become the basis.
 3. Work out the θ -values for each row. Using the smallest (but not negative) value, select the pivot.
 4. Divide the pivot row by the value of the pivot.
 5. Add or subtract multiple of the pivot row from the other rows so that the basis variable is 0.
 6. Repeat until the objective row contains no negative entries.
 7. Read off the optimal solution from the tableau.

Please enter either integers or fractions (e.g. in the form 3/5)
 Constraints:
 $2.0x + 9.0y \leq 1.0$
 $5.0x + -7.0y \leq 8.0$
 $7.0x + -3.0y \leq 10.0$
 Objective: $P = 10.0x + 1.0y$

x =

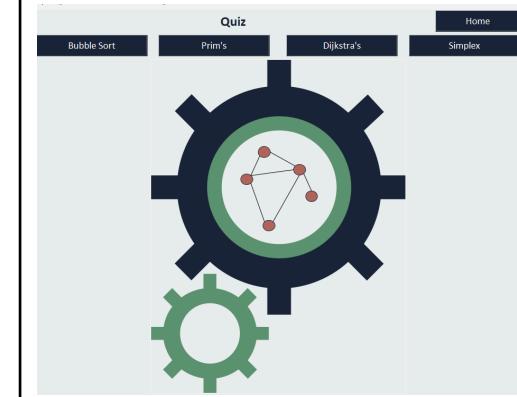
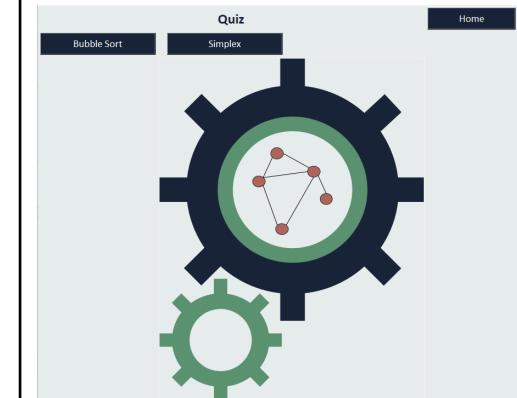
y =

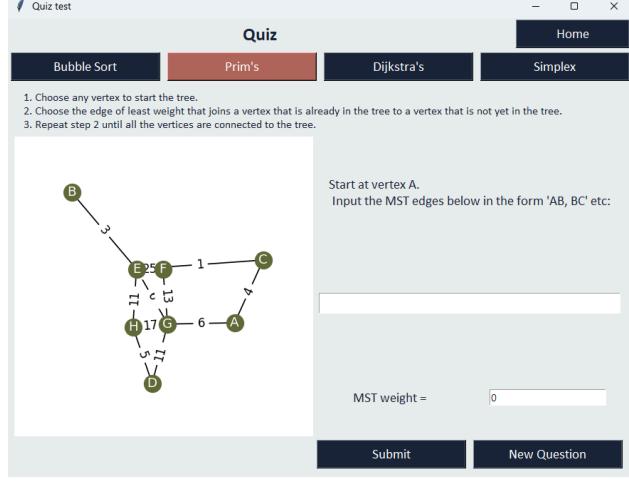
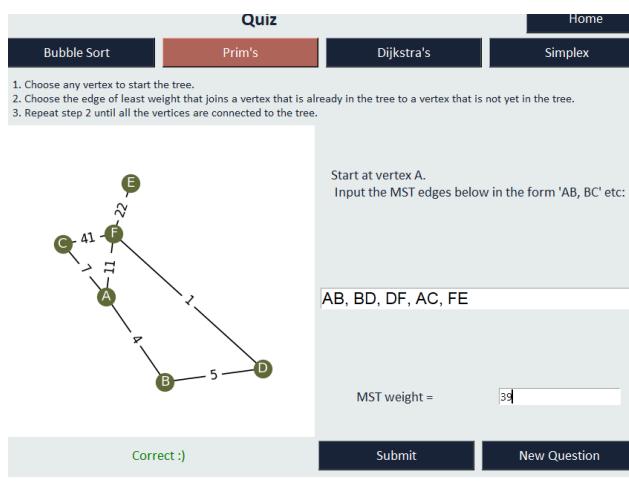
P =

[Submit](#) [New Question](#)

All the GUIs now look good so all development in this iteration is now complete.

Now I will test the code developed so far using the white box testing table specified in the QUIZ PAGE design section. Having already tested the Bubble Sort and Simplex quiz functionality, I will now test against the rest of the criteria.

TEST DATA	EXPECTED RESULT	JUSTIFICATION	CODE	ACTUAL OUTPUT
Open the quiz page for an account with different algorithms chosen.	Only buttons for the selected algorithms during registration are displayed next to each other at the top of the page.	This ensures that the user's algorithm choices are followed on the quiz page, whatever the user's original choice was.	4.3.o 4.3.p	<p>For an account with all algorithms chosen:</p>  <p>For an account with only 2 of the algorithms chosen:</p>  <p>As expected.</p>

Press the Prim's button.	The Prim's button is highlighted in pink and text describing the algorithm is below this. A graph is displayed on the left side of the screen and there are text boxes for the MST edges and weight on the right.	This ensures that the correct widgets required for the user to complete a Prim's algorithm question are displayed and that the randomised graph functionality works correctly.	4.3.a 4.3.b 4.3.c 4.3.d	 <p>As expected.</p>
Submit a Prim's algorithm question with both the MST and weight fields correctly filled in.	Text will appear saying 'Correct' and the correct score in the database is updated.	This ensures that the program can validate a correct answer as being correct.	4.3.b 4.3.c 4.3.f 4.3.e 4.3.n	 <p>In DB Browser for SQLite, the original state was this:</p>

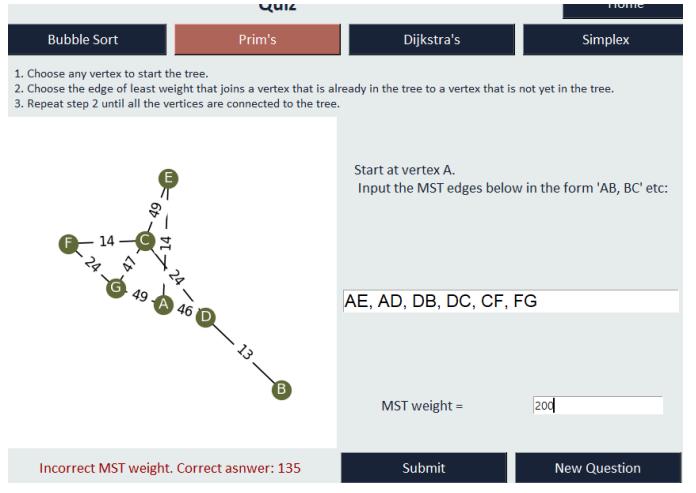
StudentEnrolID	Username	Algorithm	CorrectScore	IncorrectScore
Filter	Filter	Filter	Filter	Filter
1	Test_accS	Bubble Sort	3	2
2	Test_accS	Simplex	1	5
3	Billy123	Bubble Sort	0	0
4	HelloWord!!!	Bubble Sort	3	6
5	HelloWord!!!	Prim's	4	1
6	HelloWord!!!	Dijkstra's	5	1
7	HelloWord!!!	Simplex	0	3

After submitting the answer, the state is this:

StudentEnrolID	Username	Algorithm	CorrectScore	IncorrectScore
Iter	Filter	Filter	Filter	Filter
1	Test_accS	Bubble Sort	3	2
2	Test_accS	Simplex	1	5
3	Billy123	Bubble Sort	0	0
4	HelloWord!!!	Bubble Sort	4	9
5	HelloWord!!!	Prim's	5	2
6	HelloWord!!!	Dijkstra's	5	1
7	HelloWord!!!	Simplex	1	6

As expected.

<p>Submit a Prim's algorithm question with only the weight filled in correctly.</p>	<p>Text will appear saying 'Incorrect MST edges' and the incorrect score in the database is updated.</p>	<p>This ensures that the program can validate an incorrect answer as being incorrect.</p>	<p>4.3.b 4.3.c 4.3.e 4.3.n</p>	<p>The screenshot shows a web-based application for algorithm simulation. At the top, there are tabs for Bubble Sort, Prim's (which is selected), Dijkstra's, and Simplex. Below the tabs, instructions for Prim's algorithm are listed: 1. Choose any vertex to start the tree. 2. Choose the edge of least weight that joins a vertex that is already in the tree to a vertex that is not yet in the tree. 3. Repeat step 2 until all the vertices are connected to the tree. The graph below shows vertices A through H with their connections and weights. Vertex A is the starting point. The user has inputted MST edges AD, BC, DF. The calculated MST weight is 82. A message box indicates an error: "Incorrect MST edges Try a new question".</p> <table border="1" data-bbox="1343 728 1994 966"> <tbody> <tr> <td>4</td> <td>HelloWord!!!</td> <td>Bubble Sort</td> <td>4</td> <td>9</td> </tr> <tr> <td>5</td> <td>HelloWord!!!</td> <td>Prim's</td> <td>5</td> <td>3</td> </tr> <tr> <td>6</td> <td>HelloWord!!!</td> <td>Dijkstra's</td> <td>5</td> <td>1</td> </tr> <tr> <td>7</td> <td>HelloWord!!!</td> <td>Simplex</td> <td>1</td> <td>6</td> </tr> </tbody> </table> <p>From the same state of database shown above, the new state is:</p> <p>As expected.</p>	4	HelloWord!!!	Bubble Sort	4	9	5	HelloWord!!!	Prim's	5	3	6	HelloWord!!!	Dijkstra's	5	1	7	HelloWord!!!	Simplex	1	6
4	HelloWord!!!	Bubble Sort	4	9																				
5	HelloWord!!!	Prim's	5	3																				
6	HelloWord!!!	Dijkstra's	5	1																				
7	HelloWord!!!	Simplex	1	6																				

<p>Submit a Prim's algorithm question with only the MST filled in correctly.</p>	<p>Text will appear saying 'Incorrect weight' and the incorrect score in the database is updated.</p>	<p>This ensures that the program can validate an incorrect answer as being incorrect.</p>	<p>4.3.b 4.3.c 4.3.e 4.3.n</p>	 <p>Start at vertex A. Input the MST edges below in the form 'AB, BC' etc:</p> <p>AE, AD, DB, DC, CF, FG</p> <p>MST weight = <input type="text" value="200"/></p> <p>Incorrect MST weight. Correct answer: 135</p> <p>Submit New Question</p>
--	---	---	--	---

Below is the new state of the database - note that one incorrect answer had been input since the previous state shown before:

4	HelloWord!!!	Bubble Sort	4	9
5	HelloWord!!!	Prim's	5	5
6	HelloWord!!!	Dijkstra's	5	1
7	HelloWord!!!	Simplex	1	6

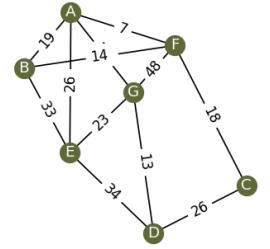
As expected.

Submit a Prim's algorithm question with the MST and weight incorrectly filled in.	Text will appear saying 'Incorrect' and the incorrect score in the database is updated.	This ensures that the program can validate an incorrect answer as being incorrect.	4.3.b 4.3.c 4.3.e 4.3.n	<div style="border: 1px solid #ccc; padding: 10px;"> <p>Quiz</p> <p>Bubble Sort Prim's Dijkstra's Simplex</p> <p>1. Choose any vertex to start the tree. 2. Choose the edge of least weight that joins a vertex that is already in the tree to a vertex that is not yet in the tree. 3. Repeat step 2 until all the vertices are connected to the tree.</p> <p>Start at vertex A. Input the MST edges below in the form 'AB, BC' etc:</p> <p>DF, AD, GE</p> <p>MST weight = <input type="text" value="10"/></p> <p>Incorrect MST edges Try a new question</p> <p>Submit New Question</p> <p>The state of the database:</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <tbody> <tr> <td style="padding: 5px;">4</td> <td style="padding: 5px;">HelloWord!!!</td> <td style="padding: 5px;">Bubble Sort</td> <td style="padding: 5px;">4</td> </tr> <tr> <td style="padding: 5px;">5</td> <td style="padding: 5px;">HelloWord!!!</td> <td style="padding: 5px;">Prim's</td> <td style="padding: 5px;">5</td> </tr> <tr> <td style="padding: 5px;">6</td> <td style="padding: 5px;">HelloWord!!!</td> <td style="padding: 5px;">Dijkstra's</td> <td style="padding: 5px;">5</td> </tr> <tr> <td style="padding: 5px;">7</td> <td style="padding: 5px;">HelloWord!!!</td> <td style="padding: 5px;">Simplex</td> <td style="padding: 5px;">1</td> </tr> </tbody> </table> <p>As expected.</p> </div>	4	HelloWord!!!	Bubble Sort	4	5	HelloWord!!!	Prim's	5	6	HelloWord!!!	Dijkstra's	5	7	HelloWord!!!	Simplex	1
4	HelloWord!!!	Bubble Sort	4																	
5	HelloWord!!!	Prim's	5																	
6	HelloWord!!!	Dijkstra's	5																	
7	HelloWord!!!	Simplex	1																	

<p>Press the New Question button on the Prim's page.</p>	<p>A new graph is randomised and displayed. The MST and weight entry fields are cleared.</p>	<p>This ensures that the user can generate a new question of the Prim's algorithm type when they press this button.</p>	<p>4.3.b 4.3.c 4.3.g</p>	<p>Quiz</p> <p>Bubble Sort Prim's Dijkstra's Simplex</p> <p>Start at vertex A. Input the MST edges below in the form 'AB, BC' etc:</p> <p>MST weight = <input type="text" value="0"/></p> <p>Submit New Question</p>
<p>Press the Dijkstra's button.</p>	<p>The Dijkstra's button is highlighted in pink and text describing the algorithm is below this. A graph is displayed on the left side of the screen and there are text boxes for the shortest path and weight on the right.</p>	<p>This ensures that the correct widgets required for the user to complete a Dijkstra's algorithm question are displayed and that the randomised graph functionality works correctly.</p>	<p>4.3.h 4.3.i 4.3.j</p>	<p>Quiz</p> <p>Bubble Sort Prim's Dijkstra's Simplex</p> <p>Start at vertex A. End vertex H. Input the shortest path in the form 'ABC...':</p> <p>Shortest path = <input type="text" value="0"/></p> <p>Submit New Question</p>

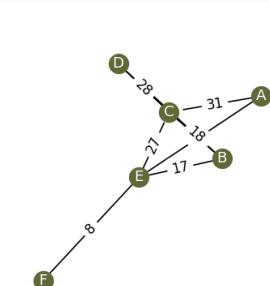
As expected.

<p>Submit a Dijkstra's algorithm question with both the shortest path and weight fields correctly filled in.</p>	<p>Text will appear saying 'Correct' and the correct score in the database is updated.</p>	<p>This ensures that the program can validate a correct answer as being correct.</p>	<p>4.3.k 4.3.l 4.3.n</p> <div data-bbox="1320 235 2039 887"> <p>Quiz</p> <p>Bubble Sort Prim's Dijkstra's Simplex Home</p> <p>1. Set the initial distance from the start values for all nodes (0 for the start node and infinity for all other nodes). 2. Find the node with the shortest distance from the start that has not been visited and look at all the unvisited connected nodes. 3. Calculate the distance from the start for each of the unvisited connected nodes. 4. If the distance calculated is less than the current shortest distance from the start, set the previous node for that connected node to be the current node. 5. Then set the current node as visited. 6. Repeat steps 2 to 5 until all nodes are set to visited. To find the shortest path through backtracking: 7. Start from the goal node. 8. Add the 'previous node' to the start of a list. 9. Repeat step 7 until the start node is reached.</p> <p>Start at vertex A. End vertex G. Input the shortest path in the form 'ABC...':</p> <p>ABDG</p> <p>Shortest path = <input type="text" value="17"/></p> <p>Correct :) Submit New Question</p> </div> <p>The new state of the database:</p> <table border="1"> <tbody> <tr> <td>4</td> <td>HelloWord!!!</td> <td>Bubble Sort</td> <td>4</td> <td>9</td> </tr> <tr> <td>5</td> <td>HelloWord!!!</td> <td>Prim's</td> <td>5</td> <td>6</td> </tr> <tr> <td>6</td> <td>HelloWord!!!</td> <td>Dijkstra's</td> <td>6</td> <td>1</td> </tr> <tr> <td>7</td> <td>HelloWord!!!</td> <td>Simplex</td> <td>1</td> <td>6</td> </tr> </tbody> </table> <p>As expected.</p>	4	HelloWord!!!	Bubble Sort	4	9	5	HelloWord!!!	Prim's	5	6	6	HelloWord!!!	Dijkstra's	6	1	7	HelloWord!!!	Simplex	1	6
4	HelloWord!!!	Bubble Sort	4	9																			
5	HelloWord!!!	Prim's	5	6																			
6	HelloWord!!!	Dijkstra's	6	1																			
7	HelloWord!!!	Simplex	1	6																			

Submit a Dijkstra's algorithm question with only the shortest path filled in correctly.	Text will appear saying 'Incorrect weight' and the incorrect score in the database is updated.	This ensures that the program can validate an incorrect answer as being incorrect.	4.3.k 4.3.l 4.3.n	<div style="background-color: #f0f0f0; padding: 10px;"> <p>Quiz</p> <p>Bubble Sort Prim's Dijkstra's Simplex</p> <p>1. Set the initial distance from the start values for all nodes (0 for the start node and infinity for all other nodes). 2. Find the node with the shortest distance from the start that has not been visited and look at all the unvisited connected nodes. 3. Calculate the distance from the start for each of the unvisited connected nodes. 4. If the distance calculated is less than the current shortest distance from the start, set the previous node for that connected node to be the current node. 5. Then set the current node as visited. 6. Repeat steps 2 to 5 until all nodes are set to visited. To find the shortest path through backtracking: 7. Start from the goal node. 8. Add the 'previous node' to the start of a list. 9. Repeat step 7 until the start node is reached.</p> <div style="border: 1px solid #ccc; padding: 10px; margin-top: 10px;">  <p>Start at vertex A. End vertex G. Input the shortest path in the form 'ABC...':</p> <input data-bbox="1776 674 1918 696" type="text" value="AG"/> <p>Shortest path = <input data-bbox="1866 785 2007 807" type="text" value="81"/></p> <p>Incorrect path weight. Correct answer: 22</p> <p>Submit New Question</p> </div> <p>The state of the database:</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <tbody> <tr> <td style="padding: 5px;">4</td><td style="padding: 5px;">HelloWord!!!</td><td style="padding: 5px;">Bubble Sort</td><td style="padding: 5px;">4</td><td style="padding: 5px;">9</td></tr> <tr> <td style="padding: 5px;">5</td><td style="padding: 5px;">HelloWord!!!</td><td style="padding: 5px;">Prim's</td><td style="padding: 5px;">5</td><td style="padding: 5px;">6</td></tr> <tr> <td style="padding: 5px;">6</td><td style="padding: 5px;">HelloWord!!!</td><td style="padding: 5px;">Dijkstra's</td><td style="padding: 5px;">6</td><td style="padding: 5px;">2</td></tr> <tr> <td style="padding: 5px;">7</td><td style="padding: 5px;">HelloWord!!!</td><td style="padding: 5px;">Simplex</td><td style="padding: 5px;">1</td><td style="padding: 5px;">6</td></tr> </tbody> </table> <p>As expected.</p> </div>	4	HelloWord!!!	Bubble Sort	4	9	5	HelloWord!!!	Prim's	5	6	6	HelloWord!!!	Dijkstra's	6	2	7	HelloWord!!!	Simplex	1	6
4	HelloWord!!!	Bubble Sort	4	9																				
5	HelloWord!!!	Prim's	5	6																				
6	HelloWord!!!	Dijkstra's	6	2																				
7	HelloWord!!!	Simplex	1	6																				

<p>Submit a Dijkstra's algorithm question with only the weight filled in correctly.</p>	<p>Text will appear saying 'Incorrect shortest path' and the incorrect score in the database is updated.</p>	<p>This ensures that the program can validate an incorrect answer as being incorrect.</p>	<p>4.3.k 4.3.l 4.3.n</p> <div data-bbox="1313 230 1942 628"> <p>To find the shortest path through backtracking: 7. Start from the goal node. 8. Add the 'previous node' to the start of a list. 9. Repeat step 7 until the start node is reached.</p> <p>Start at vertex A. End vertex G. Input the shortest path in the form 'ABC...':</p> <input type="text"/> <p>Shortest path = <input type="text"/> 0</p> <p>Submit New Question</p> </div> <p>For this question the end vertex is stated as G but it shouldn't be. Therefore, I need to add some code to allow this to be updated when New Question is pressed. For a question that does have the correct text:</p> <div data-bbox="1313 787 2021 1264"> <p>5. Then set the current node as visited. 6. Repeat steps 2 to 5 until all nodes are set to visited. To find the shortest path through backtracking: 7. Start from the goal node. 8. Add the 'previous node' to the start of a list. 9. Repeat step 7 until the start node is reached.</p> <p>Start at vertex A. End vertex F. Input the shortest path in the form 'ABC...':</p> <input type="text"/> ACBF <p>Shortest path = <input type="text"/> 12</p> <p>Submit New Question</p> </div> <p>The state of the database:</p>
---	--	---	---

				<table border="1"> <tbody> <tr> <td>4</td><td>HelloWord!!!</td><td>Bubble Sort</td><td>4</td></tr> <tr> <td>5</td><td>HelloWord!!!</td><td>Prim's</td><td>5</td></tr> <tr> <td>6</td><td>HelloWord!!!</td><td>Dijkstra's</td><td>6</td></tr> <tr> <td>7</td><td>HelloWord!!!</td><td>Simplex</td><td>1</td></tr> </tbody> </table> <p>As expected for this input.</p>	4	HelloWord!!!	Bubble Sort	4	5	HelloWord!!!	Prim's	5	6	HelloWord!!!	Dijkstra's	6	7	HelloWord!!!	Simplex	1
4	HelloWord!!!	Bubble Sort	4																	
5	HelloWord!!!	Prim's	5																	
6	HelloWord!!!	Dijkstra's	6																	
7	HelloWord!!!	Simplex	1																	
Submit a Dijkstra's algorithm question with the shortest path and weight incorrectly filled in.	Text will appear saying 'Incorrect' and the incorrect score in the database is updated.	This ensures that the program can validate an incorrect answer as being incorrect.	4.3.k 4.3.l 4.3.n	<div style="background-color: #f0f0f0; padding: 10px;"> <p style="text-align: center;">Quiz</p> <p style="text-align: right;">home</p> <p style="text-align: center;">Bubble Sort Prim's Dijkstra's Simplex</p> <p>1. Set the initial distance from the start values for all nodes (0 for the start node and infinity for all other nodes). 2. Find the node with the shortest distance from the start that has not been visited and look at all the unvisited connected nodes. 3. Calculate the distance from the start for each of the unvisited connected nodes. 4. If the distance calculated is less than the current shortest distance from the start, set the previous node for that connected node to be the current node. 5. Then set the current node as visited. 6. Repeat steps 2 to 5 until all nodes are set to visited. To find the shortest path through backtracking: 7. Start from the goal node. 8. Add the 'previous node' to the start of a list. 9. Repeat step 7 until the start node is reached.</p> <div style="border: 1px solid #ccc; padding: 10px; margin-top: 10px;"> <p>Start at vertex A. End vertex G. Input the shortest path in the form 'ABC...':</p> <input style="width: 200px; margin-bottom: 10px;" type="text" value="ABFG"/> <p>Shortest path = <input style="width: 100px;" type="text" value="1000"/></p> <p style="text-align: right;">Submit New Question</p> </div> <p style="color: red; font-size: small; margin-top: 10px;">Incorrect path Try a new question</p> </div> <p>The state of the database:</p>																

				<table border="1"> <tbody> <tr> <td>4</td><td>HelloWord!!!</td><td>Bubble Sort</td><td>4</td><td>9</td></tr> <tr> <td>5</td><td>HelloWord!!!</td><td>Prim's</td><td>5</td><td>6</td></tr> <tr> <td>6</td><td>HelloWord!!!</td><td>Dijkstra's</td><td>6</td><td>4</td></tr> <tr> <td>7</td><td>HelloWord!!!</td><td>Simplex</td><td>1</td><td>6</td></tr> </tbody> </table> <p>As expected.</p>	4	HelloWord!!!	Bubble Sort	4	9	5	HelloWord!!!	Prim's	5	6	6	HelloWord!!!	Dijkstra's	6	4	7	HelloWord!!!	Simplex	1	6
4	HelloWord!!!	Bubble Sort	4	9																				
5	HelloWord!!!	Prim's	5	6																				
6	HelloWord!!!	Dijkstra's	6	4																				
7	HelloWord!!!	Simplex	1	6																				
Press the New Question button on the Dijkstra's page.	A new graph is randomised and displayed. The shortest path and weight entry fields are cleared.	This ensures that the user can generate a new question of the Dijkstra's algorithm type when they press this button.	4.3.m	<div style="background-color: #f0f0f0; padding: 10px;"> <p style="text-align: right;">Quiz Home</p> <p style="text-align: center;">Bubble Sort Prim's Dijkstra's Simplex</p> <p>1. Set the initial distance from the start values for all nodes (0 for the start node and infinity for all other nodes). 2. Find the node with the shortest distance from the start that has not been visited and look at all the unvisited connected nodes. 3. Calculate the distance from the start for each of the unvisited connected nodes. 4. If the distance calculated is less than the current shortest distance from the start, set the previous node for that connected node to be the current node. 5. Then set the current node as visited. 6. Repeat steps 2 to 5 until all nodes are set to visited. To find the shortest path through backtracking: 7. Start from the goal node. 8. Add the 'previous node' to the start of a list. 9. Repeat step 7 until the start node is reached.</p> <div style="border: 1px solid #ccc; padding: 10px; margin-top: 10px;">  <p>Start at vertex A. End vertex H. Input the shortest path in the form 'ABC...': <input type="text"/></p> <p>Shortest path = <input type="text"/> 0</p> <p style="text-align: right;">Submit New Question</p> </div> </div> <p>As expected.</p>																				

During white box testing, I found one issue that needs to be fixed. When pressing the New Question button, I found that the text saying the end vertex does not update to be the last vertex alphabetically in the graph. To fix this, I made the text widget an attribute:

4.3.s

```
self.startEndText = Label(self.dijkstraQuizFrame,
    text=f"Start at vertex A. End vertex {self.endVertex}.\nInput the s
    font=fontMedium,
    fg="#1b263b",
    bg="#eaebed",
    justify="left")
self.startEndText.grid(row=4, column=6, columnspan=6, rowspan=2)
```

And I updated the text in this widget in the newQuestion method:

4.3.t

```
elif qType == "Dijkstra's":
    self.randomGraph()

    self.ax.clear()
    pos = nx.spring_layout(self.quizGraph)
    nx.draw(self.quizGraph, pos, with_labels=True, node_color='#606c38', edge_color='black', node_size=200, font_color='white', font_size=10)
    edge_labels = nx.get_edge_attributes(self.quizGraph, 'weight')
    nx.draw_networkx_edge_labels(self.quizGraph, pos, edge_labels=edge_labels, ax=self.ax)
    self.dijkstraQuizGraphCanvas.draw()

    self.shortestPathEntry.set("")
    self.weightEntry.set(0)
    self.answerText["text"] = ""

    # add code to update the text saying the end vertex
    self.startEndText["text"] = f"Start at vertex A. End vertex {self.endVertex}.\nInput the shortest path in the form 'ABC...':"
```

Now when I run the code and press the New Question button, the following is output:

QUIZ		Home	
Bubble Sort	Prim's	Dijkstra's	Simplex
<p>1. Set the initial distance from the start values for all nodes (0 for the start node and infinity for all other nodes). 2. Find the node with the shortest distance from the start that has not been visited and look at all the unvisited connected nodes. 3. Calculate the distance from the start for each of the unvisited connected nodes. 4. If the distance calculated is less than the current shortest distance from the start, set the previous node for that connected node to be the current node. 5. Then set the current node as visited. 6. Repeat steps 2 to 5 until all nodes are set to visited. To find the shortest path through backtracking: 7. Start from the goal node. 8. Add the 'previous node' to the start of a list. 9. Repeat step 7 until the start node is reached.</p>			
<p>1. Set the initial distance from the start values for all nodes (0 for the start node and infinity for all other nodes). 2. Find the node with the shortest distance from the start that has not been visited and look at all the unvisited connected nodes. 3. Calculate the distance from the start for each of the unvisited connected nodes. 4. If the distance calculated is less than the current shortest distance from the start, set the previous node for that connected node to be the current node. 5. Then set the current node as visited. 6. Repeat steps 2 to 5 until all nodes are set to visited. To find the shortest path through backtracking: 7. Start from the goal node. 8. Add the 'previous node' to the start of a list. 9. Repeat step 7 until the start node is reached.</p>			
<p>Start at vertex A. End vertex F. Input the shortest path in the form 'ABC...':</p> <input style="width: 100px; margin-left: 100px;" type="text"/> <p>Shortest path weight = <input style="width: 50px; margin-left: 100px; value='0'" type="text"/></p>		<p>Start at vertex A. End vertex G. Input the shortest path in the form 'ABC...':</p> <input style="width: 100px; margin-left: 100px;" type="text"/> <p>Shortest path weight = <input style="width: 50px; margin-left: 100px; value='0'" type="text"/></p>	
<p>Submit</p>		<p>New Question</p>	

This is how I expected the output to be so the issue has been fixed. Now all development and testing in Prototype 4 Iteration 3 is complete. All development in this system is now finished.

ITERATION 3 - DEVELOPER REVIEW

I would say that development in this iteration went much smoother than in Iteration 2 because Prim's and Dijkstra's algorithms both had to have a random graph generator, which was quite simple to develop thanks to me using a Prim's algorithm approach to coding it, meaning that I was quite confident about the approach. The SQL queries to update the scores in the database also were quite easy to develop thanks to me defining the SQL statements early in the Design section. Only one minor issue regarding updating the GUI was found during white box testing but this was fixed easily.

PROTOTYPE 4 - STAKEHOLDER REVIEW

Now that Prototype 4 has been fully developed and tested, I had a stakeholder review the progress. I asked about the Usability - how user friendly the system is - the Functionality - does it function as intended? - and Performance - how fast the system performs.

See below the feedback given by one of my Stakeholders:

AREA OF CONCERN	FEEDBACK
Usability	It seemed quite simple and easy to understand what was going on.
Functionality	It worked well and when I tried to put values in that could have caused the program to break, it didn't.
Performance	Everything performed well but the Simplex algorithm visualisation could have been a bit faster.

From this feedback, the system seems to be functioning well and be user friendly. The only area that could be improved was the speed of the Simplex algorithm visualisation. I had initially set it to be quite slow at 1500 milliseconds. To make the visualisation seem smoother, I will reduce this to 1000 milliseconds because this still lets the user understand where the current vertex in the feasible region is but changes appear smoother:

4.3.u

```
def simplexAnimate(self):
    #print("in simplex animate")
    if self.currentStep < len(self.steps)-1 and not self.isPaused:
        self.updateSimplexGraph()
        self.currentStep += 1
        self.master.after(1000, self.simplexAnimate) # change time
```

PROTOTYPE 4 - SUCCESS CRITERIA REVIEW

In order to check that all the success criteria outlined in the ANALYSIS section for the Simplex Visualisation page and the Quiz page have been met, below is a success criteria review for this prototype:

SUCCESS CRITERIA	OUTPUT & EVIDENCE	CODE	COMPLETE
[7][a] There will be text greeting the user with the title of the system 'Simplex Algorithm Visualisation'. I will test this by checking the page and making sure this appears.	Simplex Algorithm Visualisation	4.1.e	Y
[7][b] There will be text describing the Simplex algorithm in steps in plain English. I will test this by checking the page and making sure this appears.	<ol style="list-style-type: none"> 1. Turn the inequalities into equalities by adding a slack variable. Rearrange the objective function to be equal to 0. Input these equations into the simplex tableau. 2. Choose the most negative value in the objective row to become the basis. 3. Work out the θ-values for each row. Using the smallest (but not negative) value, select the pivot. 4. Divide the pivot row by the value of the pivot. 5. Add or subtract multiple of the pivot row from the other rows so that the basis variable is 0. 6. Repeat until the objective row contains no negative entries. 7. Read off the optimal solution from the tableau, 	4.1.e	Y
[7][c] On the right side of the screen, there are two constraint entry fields. This includes a text box for the x coefficient, a text box for the y coefficient and a text box for the constant. These will display as the following $x + y \geq$ with the $_$ signifying a text box in this case. I will test this by checking that these are displayed when the page is accessed and that numbers can be entered into the text boxes.	<p>Constraint 1: <input type="text"/> $x +$ <input type="text"/> $y \leq$ <input type="text"/></p> <p>Constraint 2: <input type="text"/> $x +$ <input type="text"/> $y \leq$ <input type="text"/></p>	4.1.f 4.1.g	Y

<p>[7][d] Below the 2 constraint entry fields, there will be an 'Add constraint' button. When pressed, this will display a constraint entry field - as described in [7][c] - below the 2 existing entry fields, with the button disappearing. I will test this by checking that when the page is accessed, the button is displayed and that clicking it will make the button disappear and a constraint entry field appear, which will be tested as done in [7][c].</p>	<p>Add constraint</p> <p>When pressed:</p> <table border="1"> <tr> <td>Constraint 1:</td> <td><input type="text"/></td> <td>x +</td> <td><input type="text"/></td> <td>y ≤</td> </tr> <tr> <td>Constraint 2:</td> <td><input type="text"/></td> <td>x +</td> <td><input type="text"/></td> <td>y ≤</td> </tr> <tr> <td>Constraint 3:</td> <td><input type="text"/></td> <td>x +</td> <td><input type="text"/></td> <td>y ≤</td> </tr> </table>	Constraint 1:	<input type="text"/>	x +	<input type="text"/>	y ≤	Constraint 2:	<input type="text"/>	x +	<input type="text"/>	y ≤	Constraint 3:	<input type="text"/>	x +	<input type="text"/>	y ≤	<p>4.1.f 4.1.g</p>	<p>Y</p>
Constraint 1:	<input type="text"/>	x +	<input type="text"/>	y ≤														
Constraint 2:	<input type="text"/>	x +	<input type="text"/>	y ≤														
Constraint 3:	<input type="text"/>	x +	<input type="text"/>	y ≤														
<p>[7][e] Below the 'Add constraint' button, there will be text box entry fields for the x and y coefficients of the objective function. This will be displayed in the form $P = _x + _y$. I will test this by checking that this is displayed when the page is accessed and that numbers can be entered into the text boxes.</p>	<p>Objective function:</p> <p>$P =$ <input type="text"/> x + <input type="text"/> y</p>	<p>4.1.g</p>	<p>Y</p>															
<p>[7][f] Below the objective function entry field, there will be a 'Visualise' button. When the user presses this, the entered constraints and objective function will be validated, checking that the numbers have been entered and that they are between -10 and 10, with the x and y coefficients not both being 0. If this fails, a message will appear saying 'Constraints/objective function not valid'. If validation is successful, the constraints will be displayed on the x,y graph on the screen and the visualisation of the Simplex algorithm will begin. To test this, I will enter different valid and invalid data into the text boxes and check that the correct output is displayed.</p>	<p>Visualise</p> <p>When pressed for nothing entered:</p> <p>Invalid constraint entry: please ensure that all numbers entered are between -10 and 10, with the x and y coefficients not both being 0.</p> <p>For coefficients being less than -10 or greater than 10:</p>	<p>4.1.e 4.1.g</p>	<p>Y</p>															

	<p>Constraint 1: <input type="text" value="-11"/> $x +$ <input type="text" value="17"/> $y \leq$ <input type="text" value="0"/></p> <p>Constraint 2: <input type="text" value="35"/> $x +$ <input type="text" value="6"/> $y \leq$ <input type="text" value="-90"/></p> <p>Constraint 3: <input type="text" value="-10.1"/> $x +$ <input type="text" value="9"/> $y \leq$ <input type="text" value="3"/></p> <p>Objective function: $P =$ <input type="text" value="1"/> $x +$ <input type="text" value="1"/> y</p> <p>Visualise</p> <p>Invalid constraint entry: please ensure that all numbers entered are between -10 and 10, with the x and y coefficients not both being 0.</p> <p>For x and y coefficients both being 0:</p> <p>Constraint 1: <input type="text" value="-3"/> $x +$ <input type="text" value="1"/> $y \leq$ <input type="text" value="0"/></p> <p>Constraint 2: <input type="text" value="0"/> $x +$ <input type="text" value="0"/> $y \leq$ <input type="text" value="-90"/></p> <p>Constraint 3: <input type="text" value="0"/> $x +$ <input type="text" value="9"/> $y \leq$ <input type="text" value="3"/></p> <p>Objective function: $P =$ <input type="text" value="1"/> $x +$ <input type="text" value="1"/> y</p> <p>Visualise</p> <p>Invalid constraint entry: please ensure that all numbers entered are between -10 and 10, with the x and y coefficients not both being 0.</p>	
[7][g] On the right side of the page, below the text describing the algorithm, there will be a graph of the positive x and y axes. On the graph, the entered constraints will be plotted as straight lines and colour coded with red, green and blue. To test this, I will enter different valid constraints and check that they are plotted correctly on the axes.	<p>When valid constraints have not been submitted, the following is output:</p> <p>6. Repeat until the objective row contains no negative entries. 7. Read off the optimal solution from the tableau,</p> <p>Constraint 1: <input type="text" value="-3"/> $x +$ <input type="text" value="1"/> $y \leq$ <input type="text" value="0"/></p> <p>Constraint 2: <input type="text" value="0"/> $x +$ <input type="text" value="0"/> $y \leq$ <input type="text" value="-90"/></p> <p>Constraint 3: <input type="text" value="0"/> $x +$ <input type="text" value="9"/> $y \leq$ <input type="text" value="3"/></p> <p>Objective function: $P =$ <input type="text" value="1"/> $x +$ <input type="text" value="1"/> y</p> <p>Visualise</p> <p>Invalid constraint entry: please ensure that all numbers entered are between -10 and 10, with the x and y coefficients not both being 0.</p> <p>When valid constraints have been submitted:</p>	4.1.e 4.1.g 4.1.n 4.1.p 4.1.x 4.1.z 4.1.B 4.1.D 4.1.E 4.1.F

	<div style="border: 1px solid black; padding: 5px;"> <p>Constraint 1: $-3x + 1 \leq y \leq 0$</p> <p>Constraint 2: $0.5x + 0 \leq y \leq 2$</p> <p>Constraint 3: $0 \leq x + 9 \leq 3$</p> <p>Objective function:</p> <p>$P = 1x + 1y$</p> <p>Visualise</p> </div>	4.1.G	
[7][h] As the iterations of the Simplex algorithm occur, a highlighted bold black line should appear on the axes showing which vertices of the feasible region are visited with each iteration. To test this, I will run the visualisation with different valid constraints and check whether or not the order of visiting the vertices and the final maximising vertex is correct by comparing it to a calculation done by hand using the simplex tableau.		4.1.e 4.1.g 4.1.n 4.1.p 4.1.x 4.1.z 4.1.B 4.1.D 4.1.E 4.1.F 4.1.G	Y

<p>[7][i] When the visualisation is finished, a message should be output saying ‘Maximise P to _ when x = _ and y = _’, with x and y being the coordinate values and P being the value calculated using these x and y values. To test this, I will enter different constraints and objective functions and check that the correct values are output in this message.</p>	<p>Constraint 1: $-3x + y \leq 0$ Constraint 2: $0.5x + y \leq 2$ Constraint 3: $x + y \leq 3$</p> <p>Objective function: $P = 1x + 1y$</p> <p>Visualise</p> <p>Optimal solution $P = 13/3$ $x = 4, y = 1/3$</p>	4.1.e 4.1.g 4.1.n 4.1.p 4.1.x 4.1.z 4.1.B 4.1.D 4.1.E 4.1.F 4.1.G	Y																		
<p>[7][j] There is a Back button at the bottom right of the screen that returns the user to the Home page. To test this, I will check that the button appears in the correct place and functions as intended.</p>	<table border="1"> <caption>Quiz Statistics</caption> <thead> <tr> <th>Category</th> <th>Correct</th> <th>Incorrect</th> </tr> </thead> <tbody> <tr> <td>Bubble Sort</td> <td>~4</td> <td>~9</td> </tr> <tr> <td>Prim's</td> <td>~5</td> <td>~6</td> </tr> <tr> <td>Dijkstra's Algorithms</td> <td>~6</td> <td>~4</td> </tr> <tr> <td>Simplex</td> <td>~1</td> <td>~6</td> </tr> <tr> <td>Total</td> <td>~16</td> <td>~25</td> </tr> </tbody> </table>	Category	Correct	Incorrect	Bubble Sort	~4	~9	Prim's	~5	~6	Dijkstra's Algorithms	~6	~4	Simplex	~1	~6	Total	~16	~25	4.1.a	Y
Category	Correct	Incorrect																			
Bubble Sort	~4	~9																			
Prim's	~5	~6																			
Dijkstra's Algorithms	~6	~4																			
Simplex	~1	~6																			
Total	~16	~25																			

<p>[7][k] There is a menu - criteria [9] - that appears at the bottom of the screen. To test this, I will check that it is displayed in the correct place and functions as intended.</p>		2.3.H 2.3.I	Y
<p>[8][a] There will be text greeting the user with the title of the system 'Quiz'. I will test this by checking the page and making sure this appears.</p>		4.3.o 4.3.p	Y
<p>[8][b] There will be a button for each of the algorithms (out of the Bubble Sort Visualisation, Prim's Algorithm Visualisation, Dijkstra's Algorithm Visualisation and Simplex Algorithm Visualisation) that the user checked upon registration - see criteria [2][h]. When each is</p>	 <p>When the Bubble Sort button is pressed:</p>	4.3.o 4.3.p 4.2.j 4.2.h 4.2.o	Y

clicked, the page will change to a quiz question of that algorithm. I will test this by creating accounts with different algorithms selected and check that the correct ones appear on the screen and that they generate the correct type of question - see criteria [8][c-k].

Quiz

Bubble Sort Prim's Dijkstra's Simplex Home

- Start at the first item in the list.
- Compare the current item with the next item.
- If the two items are in the wrong position, swap them.
- Move up one item to the next time in the list.
- Repeat from step 3 until all the unsorted items have been compared.
- If any items were swapped, repeat from step 1. Otherwise, the algorithm is complete.

Please sort the following unsorted dataset in ascending order and input the number of passes it took:

36	40	29	16	2	11	10	19	35	25
<input type="text" value="0"/>									

Number of passes =

Submit **New Question**

4.2.q
4.2.r
4.2.J
4.3.b
4.3.c
4.3.g
4.3.m

When the Prim's button is pressed:

Quiz

Bubble Sort Prim's Dijkstra's Simplex Home

- Choose any vertex to start the tree.
- Choose the edge of least weight that joins a vertex that is already in the tree to a vertex that is not yet in the tree.
- Repeat step 2 until all the vertices are connected to the tree.

Start at vertex A.
Input the MST edges below in the form 'AB, BC' etc:

MST weight =

Submit **New Question**

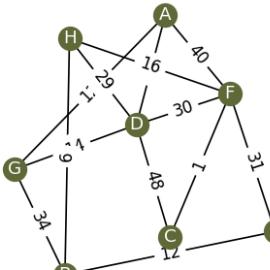
When the Dijkstra's button is pressed:

Quiz

Home

[Bubble Sort](#) [Prim's](#) **Dijkstra's** [Simplex](#)

1. Set the initial distance from the start values for all nodes (0 for the start node and infinity for all other nodes).
2. Find the node with the shortest distance from the start that has not been visited and look at all the unvisited connected nodes.
3. Calculate the distance from the start for each of the unvisited connected nodes.
4. If the distance calculated is less than the current shortest distance from the start, set the previous node for that connected node to be the current node.
5. Then set the current node as visited.
6. Repeat steps 2 to 5 until all nodes are set to visited.
To find the shortest path through backtracking:
7. Start from the goal node.
8. Add the 'previous node' to the start of a list.
9. Repeat step 7 until the start node is reached.

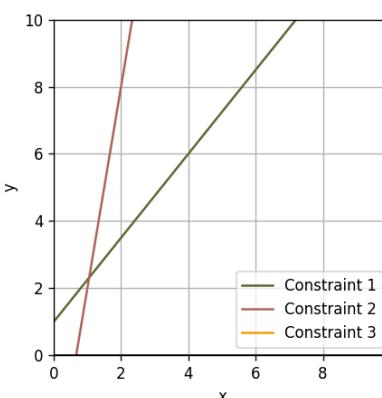


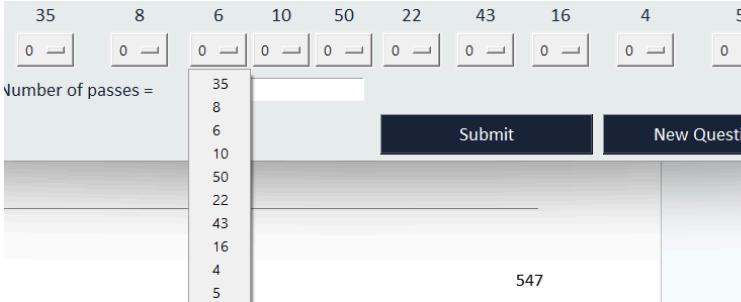
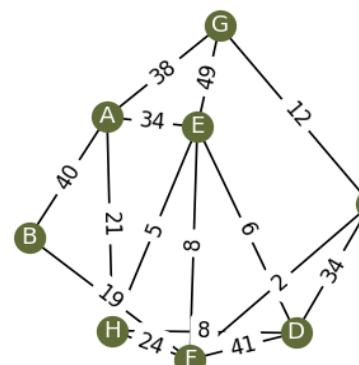
Start at vertex A. End vertex H.
Input the shortest path in the form 'ABC...':

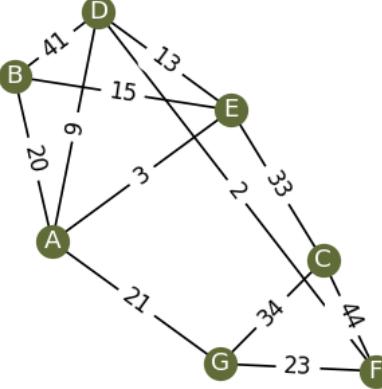
Shortest path weight =

[Submit](#) [New Question](#)

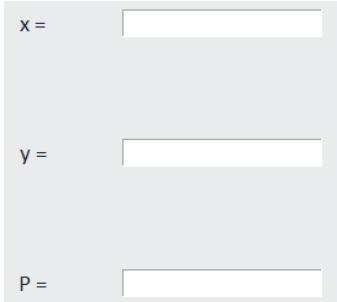
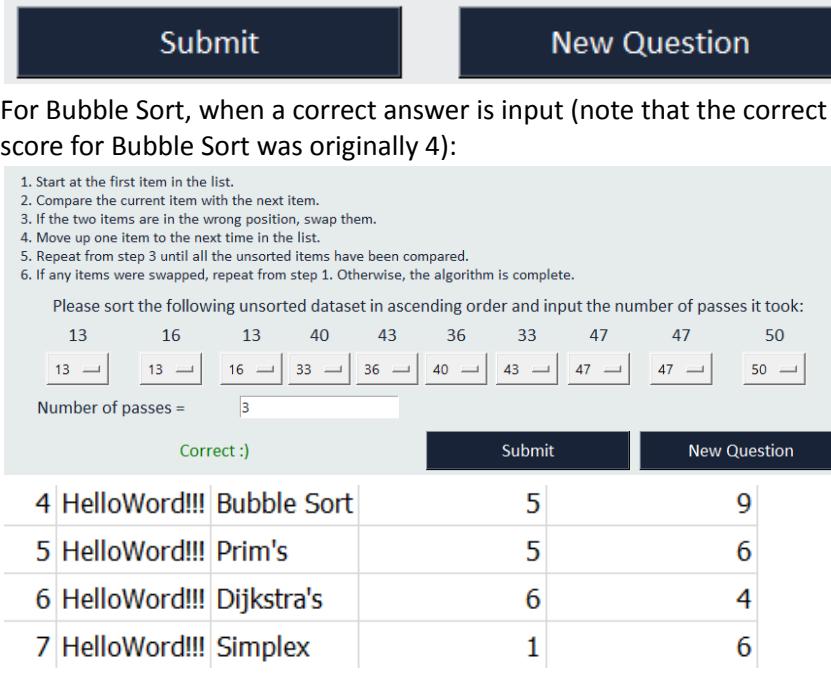
When the Simplex button is pressed:

	<div style="background-color: #f0f0f0; padding: 10px;"> <p style="text-align: center;">Quiz</p> <p style="text-align: center;">Bubble Sort Prim's Dijkstra's Home Simplex</p> <p>1. Turn the inequalities into equalities by adding a slack variable. Rearrange the objective function to be equal to 0. Input these equations into the simplex tableau. 2. Choose the most negative value in the objective row to become the basis. 3. Work out the θ-values for each row. Using the smallest (but not negative) value, select the pivot. 4. Divide the pivot row by the value of the pivot. 5. Add or subtract multiple of the pivot row from the other rows so that the basis variable is 0. 6. Repeat until the objective row contains no negative entries. 7. Read off the optimal solution from the tableau.</p>  <p>Please enter either integers or fractions (e.g. in the form 3/5) Constraints: $-10.0x + 8.0y \leq 8.0$ $6.0x + -1.0y \leq 4.0$ $-4.0x + -1.0y \leq 10.0$ Objective: $P = 1.0x + 6.0y$</p> <p>x = <input type="text"/></p> <p>y = <input type="text"/></p> <p>P = <input type="text"/></p> <p style="text-align: center;">Submit New Question</p> </div>												
[8][c] For the Bubble Sort algorithm question, a random dataset of ten values between 1 and 50 is generated and displayed on the right side of the screen as a list.	<p>Please sort the following unsorted dataset in ascending order and input the number of passes it took:</p> <table style="width: 100%; text-align: center;"> <tr> <td>35</td> <td>8</td> <td>6</td> <td>10</td> <td>50</td> <td>22</td> <td>43</td> <td>16</td> <td>4</td> <td>5</td> </tr> </table>	35	8	6	10	50	22	43	16	4	5	4.2.j 4.2.h 4.2.i 4.2.q	Y
35	8	6	10	50	22	43	16	4	5				

<p>[8][d] Below the generated dataset are 10 drop down menus of the values in the dataset. This is because this eliminates the issues from users typing the wrong number. The user should enter the final sorted outcome of the Bubble Sort algorithm being performed on the given dataset. I will test this by checking generating multiple questions and checking that the datasets are random and that the correct values appear in the drop down menus.</p>		4.2.j 4.2.h 4.2.i 4.2.q	Y
<p>[8][e] Below the drop down entry fields will be a text box for the user to enter the number of passes that occurred into. To test this, I will randomise multiple questions and check that this appears correctly.</p>	<p>Number of passes = <input type="text" value="0"/></p>	4.2.j 4.2.h 4.2.i 4.2.q	Y
<p>[8][f] For Prim's algorithm, a random connected graph of 6-8 vertices with 7+ edges will be generated as a graph for the user to perform Prim's algorithm on. Each of the vertices will be labelled with the letters A-H. This graph will be on the left side of the screen. The start vertex will also be listed as A. I will test this by choosing this type of question multiple times and checking that the graph displayed is random.</p>	 <p>There are slight issues in that some edge weights may cover other ones, therefore making it more difficult to answer the question.</p>	4.3.a 4.3.b 4.3.c 4.3.d	Y

<p>[8][g] On the right side of the screen for Prim's algorithm will be a text box for the user to enter the edges in the MST into like this: "AB, BC, CD". Below this text box will be another one to enter the value of the weight of the MST into. I will test this by randomising multiple questions and checking that these appear as expected.</p>	<p>Start at vertex A. Input the MST edges below in the form 'AB, BC' etc:</p> <input type="text"/> <p>MST weight = <input type="text" value="0"/></p>	<p>4.3.a 4.3.b 4.3.c 4.3.d</p>	<p>Y</p>
<p>[8][h] For Dijkstra's algorithm, a random connected graph of 6-8 vertices with 7+ edges will be generated as a graph for the user to perform Dijkstra's algorithm on. Each of the vertices will be labelled with the letters A-H. This graph will be on the left side of the screen. The start and end vertices will be listed as A and the last alphabetical letter vertex in the graph. I will test this by choosing this type of question multiple times and checking that the graph displayed is random.</p>	 <p>There are slight issues in that some edge weights may cover other ones, therefore making it more difficult to answer the question.</p>	<p>4.3.a 4.3.b 4.3.c 4.3.d</p>	<p>Y</p>

<p>[8][i] On the right side of the screen for quiz questions of Dijkstra's algorithm will be a text box that the user can enter the shortest path into in the format "ABCD". Below this will be a text box that the user can enter the value of the weight of the shortest path into. I will test this by selecting this type of questions and checking that these text boxes are displayed in the correct place.</p>	<p>Start at vertex A. End vertex G. Input the shortest path in the form 'ABC...':</p> <input type="text"/> <p>Shortest path weight = <input type="text" value="0"/></p>	4.3.a 4.3.b 4.3.c 4.3.d	Y
<p>[8][j] For the Simplex algorithm, 3 constraints and an objective function will be randomised, with coefficients being between -10 and 10, but not both 0 for x and y. These will be displayed on the right side of the screen and plotted on a graph on the left side of the screen. I will test this by selecting this type of question and checking that the correct constraints, with coefficients in the correct range, are displayed and plotted.</p>	<p>Please enter either integers or fractions (e.g. in the form 3/5) Constraints: $0.0x + -1.0y \leq 7.0$ $-10.0x + -9.0y \leq 3.0$ $-7.0x + 5.0y \leq 5.0$ Objective: $P = 7.0x + 2.0y$</p>	4.2.j 4.2.h 4.2.i 4.2.q 4.2.y 4.2.v	Y

<p>[8][k] Below the objective function on the right side of the screen for the Simplex algorithm will be a text box for P, a text box for x and a text box for y. The user must enter the values of P, x and y that maximise the objective function. I will test this by selecting this type of question and checking that these text boxes are displayed in the correct place and that numbers can be entered into them.</p>		4.2.j 4.2.h 4.2.i 4.2.q 4.2.y 4.2.v	Y																																								
<p>[8][l] Below the question displays will be a Check button that the user can press. This will check whether the answer entered is correct by running the algorithm on the randomised data. Then, text will display whether the user got the answer correct or not. The user's quiz statistics will also be updated in the database. I will test this by getting questions right and wrong (by working them out by hand) and checking that the output is correct.</p>	 <p>For Bubble Sort, when a correct answer is input (note that the correct score for Bubble Sort was originally 4):</p> <ol style="list-style-type: none"> 1. Start at the first item in the list. 2. Compare the current item with the next item. 3. If the two items are in the wrong position, swap them. 4. Move up one item to the next time in the list. 5. Repeat from step 3 until all the unsorted items have been compared. 6. If any items were swapped, repeat from step 1. Otherwise, the algorithm is complete. <p>Please sort the following unsorted dataset in ascending order and input the number of passes it took:</p> <table border="1"> <tr> <td>13</td> <td>16</td> <td>13</td> <td>40</td> <td>43</td> <td>36</td> <td>33</td> <td>47</td> <td>47</td> <td>50</td> </tr> <tr> <td>13</td> <td>13</td> <td>16</td> <td>33</td> <td>36</td> <td>40</td> <td>43</td> <td>47</td> <td>47</td> <td>50</td> </tr> </table> <p>Number of passes = <input type="text" value="3"/></p> <p>Correct :)</p> <p>Submit New Question</p> <table border="1"> <tr> <td>4</td> <td>HelloWord!!!</td> <td>Bubble Sort</td> <td>5</td> <td>9</td> </tr> <tr> <td>5</td> <td>HelloWord!!!</td> <td>Prim's</td> <td>5</td> <td>6</td> </tr> <tr> <td>6</td> <td>HelloWord!!!</td> <td>Dijkstra's</td> <td>6</td> <td>4</td> </tr> <tr> <td>7</td> <td>HelloWord!!!</td> <td>Simplex</td> <td>1</td> <td>6</td> </tr> </table> <p>For an incorrect answer:</p>	13	16	13	40	43	36	33	47	47	50	13	13	16	33	36	40	43	47	47	50	4	HelloWord!!!	Bubble Sort	5	9	5	HelloWord!!!	Prim's	5	6	6	HelloWord!!!	Dijkstra's	6	4	7	HelloWord!!!	Simplex	1	6	4.2.l 4.2.h 4.2.i 4.2.j 4.2.q 4.2.C 4.3.b 4.3.c 4.3.e 4.3.k 4.3.l 4.3.n	Y
13	16	13	40	43	36	33	47	47	50																																		
13	13	16	33	36	40	43	47	47	50																																		
4	HelloWord!!!	Bubble Sort	5	9																																							
5	HelloWord!!!	Prim's	5	6																																							
6	HelloWord!!!	Dijkstra's	6	4																																							
7	HelloWord!!!	Simplex	1	6																																							

1. Start at the first item in the list.
2. Compare the current item with the next item.
3. If the two items are in the wrong position, swap them.
4. Move up one item to the next time in the list.
5. Repeat from step 3 until all the unsorted items have been compared.
6. If any items were swapped, repeat from step 1. Otherwise, the algorithm is complete.

Please sort the following unsorted dataset in ascending order and input the number of passes it took:

16	3	23	22	35	18	17	18	13	6
<input type="button" value="16 ↴"/>	<input type="button" value="22 ↴"/>	<input type="button" value="35 ↴"/>	<input type="button" value="18 ↴"/>	<input type="button" value="6 ↴"/>	<input type="button" value="3 ↴"/>	<input type="button" value="17 ↴"/>	<input type="button" value="18 ↴"/>	<input type="button" value="3 ↴"/>	<input type="button" value="16 ↴"/>

Number of passes =

[Incorrect order](#) [Try a new question](#) [Submit](#) [New Question](#)

4	HelloWord!!! Bubble Sort	5	10
5	HelloWord!!! Prim's	5	6
6	HelloWord!!! Dijkstra's	6	4
7	HelloWord!!! Simplex	1	6

For Prim's, when a correct answer is input:

1. Choose any vertex to start the tree.
2. Choose the edge of least weight that joins a vertex that is already in the tree to a vertex that is not yet in the tree.
3. Repeat step 2 until all the vertices are connected to the tree.

The graph consists of 7 vertices labeled A through G. Vertex A is at the top right, B is at the top left, C is at the bottom right, D is at the bottom left, E is at the middle right, F is at the middle left, and G is at the bottom center. Edges and their weights are: AB (14), AC (28), AD (5), AE (30), AF (15), BG (11), BF (6), CG (23), and CF (2). Vertex A is highlighted with a green circle.

Correct :)

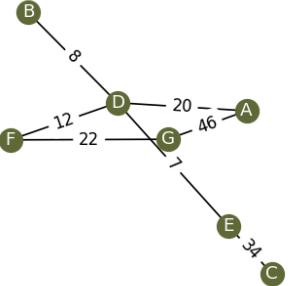
Start at vertex A.
Input the MST edges below in the form 'AB, BC' etc:

MST weight =

[Submit](#) [New Question](#)

4	HelloWord!!!	Bubble Sort	5	10
5	HelloWord!!!	Prim's	6	7
6	HelloWord!!!	Dijkstra's	6	4
7	HelloWord!!!	Simplex	1	6

For an incorrect answer:



Start at vertex A.
Input the MST edges below in the form 'AB, BC' etc:

MST weight =

Incorrect MST edges
Try a new question

Submit
New Question

4	HelloWord!!!	Bubble Sort	5	10
5	HelloWord!!!	Prim's	6	8
6	HelloWord!!!	Dijkstra's	6	4
7	HelloWord!!!	Simplex	1	6

For Dijkstra's, when a correct answer is input:

Start at vertex A. End vertex F.
Input the shortest path in the form 'ABC...':

Shortest path weight =

Correct :)
Submit
New Question

4	HelloWord!!!	Bubble Sort	5	10
5	HelloWord!!!	Prim's	6	8
6	HelloWord!!!	Dijkstra's	7	4
7	HelloWord!!!	Simplex	1	6

For an incorrect answer:

Incorrect path
Try a new question

Start at vertex A. End vertex G.
Input the shortest path in the form 'ABC...':

Shortest path weight =

Submit
New Question

	<table border="1" style="width: 100%; border-collapse: collapse;"> <tr><td>4</td><td>HelloWord!!!</td><td>Bubble Sort</td><td>5</td><td>10</td></tr> <tr><td>5</td><td>HelloWord!!!</td><td>Prim's</td><td>6</td><td>8</td></tr> <tr><td>6</td><td>HelloWord!!!</td><td>Dijkstra's</td><td>7</td><td>5</td></tr> <tr><td>7</td><td>HelloWord!!!</td><td>Simplex</td><td>1</td><td>6</td></tr> </table> <p>For Simplex, when a correct answer is input:</p> <div style="display: flex; align-items: flex-start;"> <div style="flex: 1;"> <p>x = <input type="text" value="0"/></p> <p>y = <input type="text" value="2/7"/></p> <p>P = <input type="text" value="2"/></p> </div> <div style="flex: 1; padding-left: 20px;"> Constraints: $8.0x + 7.0y \leq 2.0$ $6.0x + 1.0y \leq 7.0$ $-9.0x + -6.0y \leq 0.0$ Objective: $P = 1.0x + 7.0y$ </div> </div> <p style="text-align: center;">Correct :) Submit New Question</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <tr><td>4</td><td>HelloWord!!!</td><td>Bubble Sort</td><td>5</td><td>10</td></tr> <tr><td>5</td><td>HelloWord!!!</td><td>Prim's</td><td>6</td><td>8</td></tr> <tr><td>6</td><td>HelloWord!!!</td><td>Dijkstra's</td><td>7</td><td>5</td></tr> <tr><td>7</td><td>HelloWord!!!</td><td>Simplex</td><td>2</td><td>6</td></tr> </table> <p>For an incorrect answer:</p>	4	HelloWord!!!	Bubble Sort	5	10	5	HelloWord!!!	Prim's	6	8	6	HelloWord!!!	Dijkstra's	7	5	7	HelloWord!!!	Simplex	1	6	4	HelloWord!!!	Bubble Sort	5	10	5	HelloWord!!!	Prim's	6	8	6	HelloWord!!!	Dijkstra's	7	5	7	HelloWord!!!	Simplex	2	6	
4	HelloWord!!!	Bubble Sort	5	10																																						
5	HelloWord!!!	Prim's	6	8																																						
6	HelloWord!!!	Dijkstra's	7	5																																						
7	HelloWord!!!	Simplex	1	6																																						
4	HelloWord!!!	Bubble Sort	5	10																																						
5	HelloWord!!!	Prim's	6	8																																						
6	HelloWord!!!	Dijkstra's	7	5																																						
7	HelloWord!!!	Simplex	2	6																																						

	<p>Please enter either integers or fractions (e.g. in the form Constraints: $8.0x + 7.0y \leq 2.0$ $6.0x + 1.0y \leq 7.0$ $-9.0x + -6.0y \leq 0.0$ Objective: $P = 1.0x + 6.0y$</p> <p>x = <input type="text" value="0"/> y = <input type="text" value="5"/> P = <input type="text" value="13"/></p> <p>Incorrect Try a new question.</p> <table border="1"> <tbody> <tr> <td>4</td> <td>HelloWord!!!</td> <td>Bubble Sort</td> <td>5</td> <td>10</td> </tr> <tr> <td>5</td> <td>HelloWord!!!</td> <td>Prim's</td> <td>6</td> <td>8</td> </tr> <tr> <td>6</td> <td>HelloWord!!!</td> <td>Dijkstra's</td> <td>7</td> <td>5</td> </tr> <tr> <td>7</td> <td>HelloWord!!!</td> <td>Simplex</td> <td>2</td> <td>7</td> </tr> </tbody> </table>	4	HelloWord!!!	Bubble Sort	5	10	5	HelloWord!!!	Prim's	6	8	6	HelloWord!!!	Dijkstra's	7	5	7	HelloWord!!!	Simplex	2	7	
4	HelloWord!!!	Bubble Sort	5	10																		
5	HelloWord!!!	Prim's	6	8																		
6	HelloWord!!!	Dijkstra's	7	5																		
7	HelloWord!!!	Simplex	2	7																		
[8][m] There is a Back button at the bottom right of the screen that returns the user to the Home page. To test this, I will check that the button appears in the correct place and functions as intended.	<p>Home</p> <p>This is in the top right corner of the window.</p>	4.2.c	Y																			

All Success Criteria have now been met so development and testing in Prototype 4 is now complete.