2.2 Modules and Algorithms

Main Screen: This module will initialise the user interface. It will load the layout of the interface, the required input devices and the graph on which the projectiles path will be plotted. I will need to set up the matplotlib graph (used to show the projectiles path) to work in conjunction with Tkinter in order to ensure that everything is in the correct position.

Button to reset the simulation: The majority of the input/output devices will be set up using the Tkinter library in python. This button press will call a subroutine that resets the simulation to its default state, with no initial variable entries and an empty graph.

Button press to fire projectile: This button will be set up using Tkinter such that it interacts with the subroutine required to calculate the position of the projectile over time which in turn calls a subroutine to plot its position over time.

Calculating the projectile's position over time: This subroutine will carry out the main calculations that the program must perform. The projectiles position over time will be calculated using the Euler-Richardson Algorithm. This is an iterative algorithm used to solve the differential equations relating to an objects motion, so it can definitely be implemented in this situation (projectile motion calculations). The standard Euler algorithm approximates the velocity (for example) of an object at the end of a small time interval 'dt'. However, the Euler-Richardson algorithm would approximate the velocity of the object at the middle of the small time interval. Although twice as many calculations must be performed per step (since the velocity must be calculated in the middle of the time interval, and then used to calculate the velocity at the end of the interval) this algorithm is better than the Euler algorithm as a larger time step ('dt') can be chosen that still yields an improved accuracy over the Euler algorithm.

PSEUDOCODE ALGORITHMS:

A function to calculate frequently used values – called by the 'Fire' button press:

```
FUCTION velocityCalc(angle, v_initial):
        global v_ix //defining v_ix and v_iy as global variables so they can be used elsewhere
        global v_iy
        v_ix = v_initial * cos(angle) //calculating the x and y components of the velocity
        v_iy = v_initial * sin(angle)

ENDFUCTION
```

Dry Run:

- Inputs: angle = 30.0 degrees, v_initial = 10.00 m/s

Variables after the subroutine has been run:

- v_iy = 10sin(30) = 5.00 m/s (2 d.p) – correct output
- v_ix = 10cos(30) = 8.66 m/s (2 d.p) – correct output

```
        DISPLAY "Range of Projectile:" & range
        DISPLAY "Final Position of projectile: (" & 0 & "," & range & ")"

ENDFUCTION
```

Dry Run:
Example Inputs:

- Angle = 30.0 degrees
- v_initial = 10.00 m/s
- v_ix = 5.00 m/s
- v_iy = 8.66 m/s
- y_initial = 2.00 m
- g = 9.81 m/s^2

Variable values after the subroutine has been run:

I checked the values my pseudocode gave against other online calculators/ simulators to see if they were correct or not.
- height = 1.27 m (2 d.p) – correct output
- x_height = 4.41 m (2 d.p) – correct output
- y_height = 3.27 m (2 d.p) – correct output
- range = 11.49 m (2 d.p) – correct output

Validation

I will need to validate all of my inputs (the initial variables) to ensure that they are of the correct data type and are of the correct size. Before the projectile can be fired, the program will check that all of the required initial variables have been entered. Below is a list of the required validation for each initial variable:

1. Initial Height: Must be a floating point value. Entered to 2 decimal places. Cannot be negative. Default value: 0.00 m

2. Initial Velocity: Must be a floating point value. Entered to 2 decimal places. Cannot be negative. Default value in input box: 5.00 m/s. Minimum slider value: 0 m/s. Maximum slider value: 30 m/s. This is appropriate as most exam questions don't have particularly large velocities. If the maximum slider value was too large then the slider increments would be too small and the slider would be more difficult to control.

3. Acceleration Due to Gravity ('g'): Must be a floating point value. Entered to 2 decimal places. Must not be negative since the program assumes this value is positive. Must be greater than 0. If $g = 0$ then the program would carry on running forever since the projectile would never return to the ground. Default value: 9.81 m/s^2

4. Test co-ordinates to see if the projectile passes through them: These must be entered as floating point values. Entered to 3 decimal places. Must be positive values since the projectile's co-ordinates are always positive. Cannot be rounded as otherwise the program may return false positives. This will be told to the user, as it is not something that the program can validate. Empty by default. These inputs aren't required for the program to run.

5. Release Angle: Needs to be a real value. Must be between 0 and 90 degrees (cannot be negative since the projectile must launch into the positive x and y directions). Entered to 1 decimal place. Default value: 30.0°

The majority of this validation will be handled using Tkinter. For example, the initial velocity can be entered by either a slider or an input box. The slider will be configured such that it only allows numbers to be entered within the specified range so that the scale of the slider is not too large. The input boxes will be configured such that they only permit input of certain data types (real values).

EXAMPLE VALIDATION PSEUDOCODE ALGORITHM:

```
FUNCTION validateAngle(angle):
    IF angle is not a floating point value THEN
        RETURN an error message
    ELSE
        IF angle < 0 OR angle > 90 THEN
            RETURN an error message
```

```
            ENDIF
        ENDIF
ENDFUNCTION
```

Key Variables List

Below is a list of the variables that I have used in the above algorithms. They are all floating point values. This is because they need to be stored to a high degree of accuracy so that the calculations are accurate – rounding the values to integers may cause incorrect outputs. The number of decimal places that each variable will be entered to is based on my experience with exam questions. For example, angles are nearly always given as whole numbers, so entering them to 1 d.p. is adequate. In contrast, velocity is often given to a higher accuracy, so entering it to 2 d.p. is more appropriate.

Variable Name	Data Type	How used?
v_initial	Floating Point	INPUT– to 2 d.p.
y_inital	Floating Point	INPUT– to 2 d.p.
angle	Floating Point	INPUT– to 1 d.p.
g	Floating Point	INPUT– to 2 d.p.
x_try	Floating Point	INPUT– to 3 d.p.
y_try	Floating Point	INPUT– to 3 d.p.
v_ix	Floating Point	CALCULATION
v_iy	Floating Point	CALCULATION
t	Floating Point	CALCULATION
dt	Floating Point	CALCULATION
x_0	Floating Point	CALCULATION
y	Floating Point	CALCULATION
x	Floating Point	CALCULATION
y_mid	Floating Point	CALCULATION
v_ymid	Floating Point	CALCULATION
a_y	Floating Point	CALCULATION
y_check	Floating Point	CALCULATION
v_x	Floating Point	OUTPUT – rounded to 2 d.p.
v_y	Floating Point	OUTPUT – rounded to 2 d.p.
time	Floating Point	OUTPUT – rounded to 2 d.p.
height	Floating Point	OUTPUT – rounded to 2 d.p.
x_height	Floating Point	OUTPUT – rounded to 2 d.p.
y_height	Floating Point	OUTPUT – rounded to 2 d.p.
range	Floating Point	OUTPUT – rounded to 2 d.p.

Arrays:

These two arrays are required to store the co-ordinates of the projectile over time. The data can then be plotted using the matplotlib library in python directly from these arrays.

- x_list[] – Stores floating point values
- y_list[] – Stores floating point values