ASIA PACIFIC UNIVERSITY
OF TECHNOLOGY & INNOVATION

EE038-2-2-ESA

ENGINEERING SOFTWARE AND APPLICATIONS

TITLE	INDIVIDUAL ASSIGNMENT
STUDENT NAME	MOHAMMAD FAWZAN ALIM
STUDENT ID	TP064501
INTAKE	APD2F2206CE
LECTURER NAME	IR. EUR. ING. TS. DR. LAU CHEE YONG
SUBMISSION DATE	28 AUGUST 2022

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1. INTRODUCTION

MATLAB is a programming environment designed primarily for engineers and scientists to calculate and program complex systems. It has its own programming language, MATLAB language, which is a matrix-based language that facilitates the most natural expression of computational mathematics (Mathworks, n.d.). It is one of the most widely used software among the engineers, for its user friendliness while being such a powerful tool.

The Graphical User Interface, short for GUI, is a kind of user interface that enables users to interact with electronic devices using buttons and graphical icons (Computer Hope, 2021). GUI is the modernized version of the Command Line Interface (CLI) (Lithmee, 2018). GUI makes a program more intuitive, user-friendly, and more interactive. In MATLAB, GUI is designed using App Designer. It lets user drag and drop various elements such as image, buttons, slider, angle gauge, graph and so on (MathWorks, n.d.). User can easily move onto the functionality of the program rather worrying about the visual part.

Objective of this assignment is to apply the knowledge we learned from Engineering Software and Application module and conduct our own research to make a GUI application using MATLAB App Designer. The MATLAB application should simulate projectile motion based on different parameters such as speed, angle and environmental disturbances. These parameters should be taken input from the user and the output should be shown accordingly. To fulfill the requirements, we have to implement at least one innovative idea in the program.

2. DESIGN CONCEPT

The plan is to make a 2-D shooting game which will take the speed and shooting angle from the user as input. Then it will shoot the projectile at the target. If it succeeds to hit it, then the user will score points otherwise they will lose a life. Once they run out of all their lives, it is game over. The program will ask for their name at the end so it can record the high scores.

According to the initial Design, there would be a total of 4 windows in the program. They are the *Main* window, *Difficulty* window, *High Scores* window and finally *Game* window. The program will be designed on these concepts and improved later.

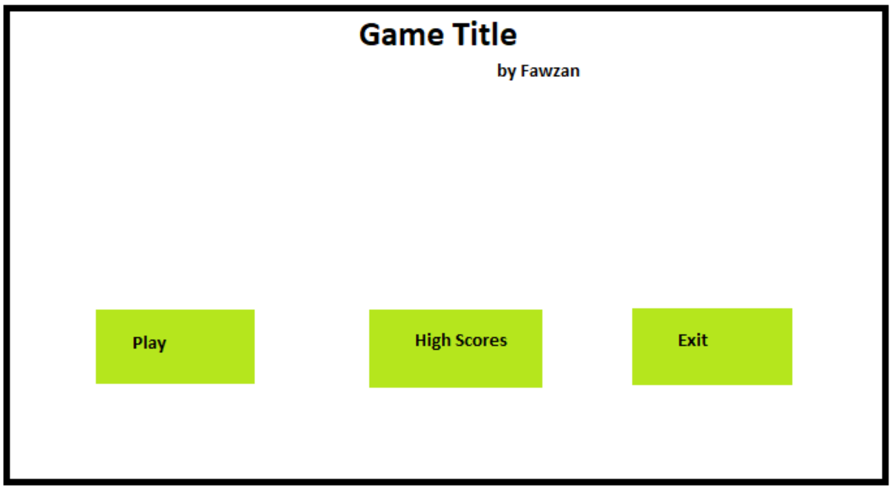


Figure 1: Design Concept of Main window

Main window will have three buttons, *Play* button to go to *Difficulty* window, *High Scores* button to go to *High Scores* window and lastly *Exit* button to Exit the program.

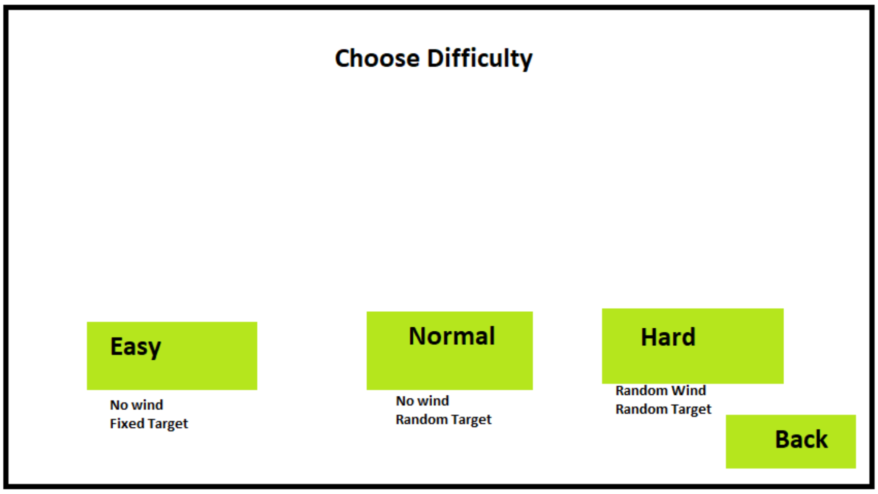


Figure 2: Design Concept of Difficulty Window

Difficulty window will have four buttons. Three buttons will let the user choose the difficulty of the game. All these three buttons will take the user to the *Game* window. Lastly the *Back* button will take them back to the *Main* window.

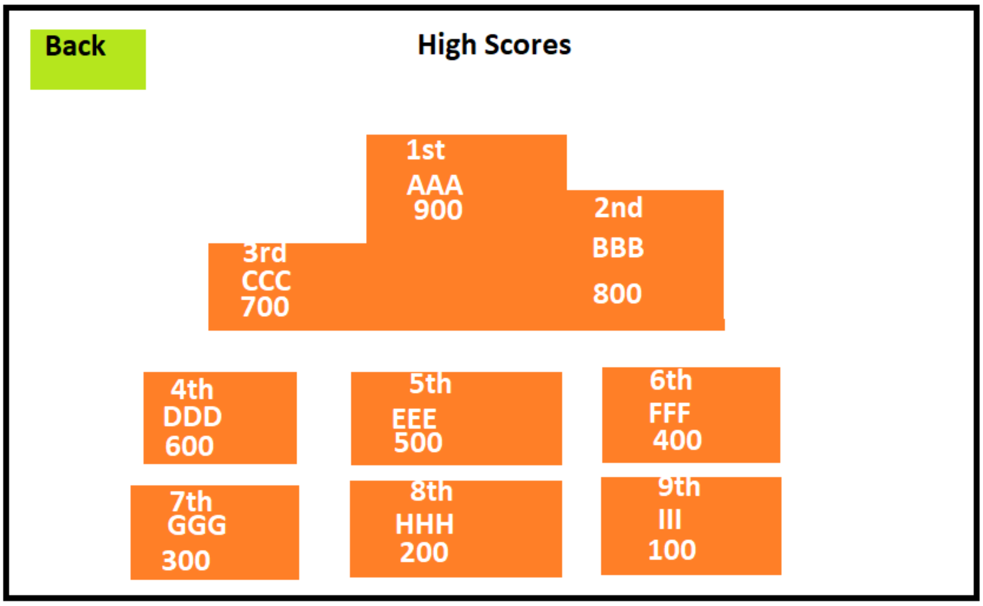


Figure 3: Design Concept of High Scores Window

In the *High Scores* window, there will be a podium for the top three scores, and the rest of them will be below the podium. By default, some random name and scores will be set as high scores. There will be a back button to take the user to *Main* window.

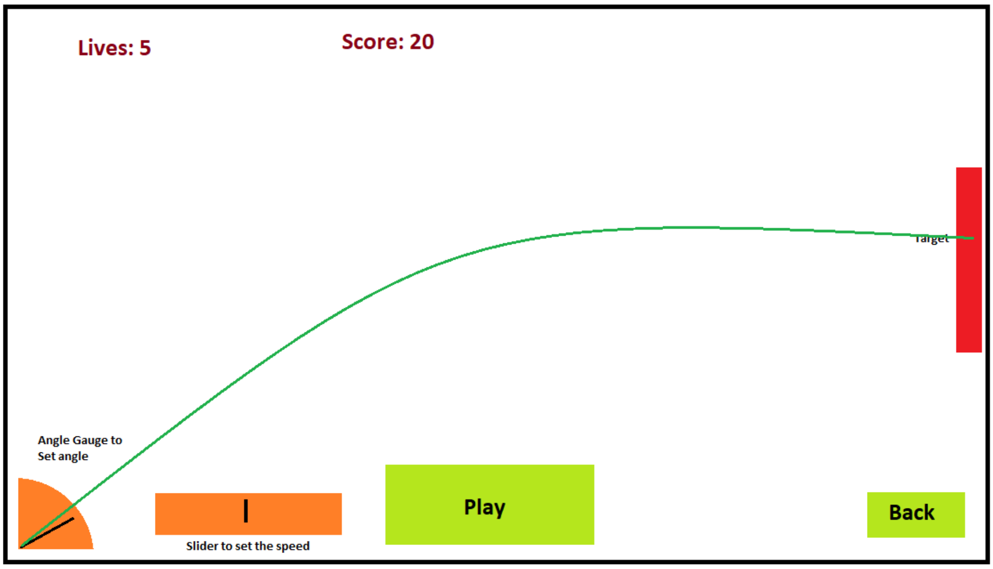


Figure 4: Design Concept of Game window

The *Game* window will be the main part of the program. There will be a *Play* button using which the user will set the angle and the speed of the projectile. Once set, it will animate the projectile. If the projectile hits the target, user will be rewarded 20 points and if it misses the target, that is going to cost the user one life. Once they run out of all lives, there will be a Game Over sign with a text field to write the user’s name for the high scores. A *Done* button will also appear with the text field which will take the user to the *High Scores* window.

3. PROJECTILE MOTION AND FORMULAE

A projectile is any object launched into space upon which only gravity acts. Gravity is the primary force acting on a projectile. This does not imply that no other forces act on it, only that their effect is negligible compared to gravity. A projectile's trajectory is known as its trajectory. (Turito Team, 2022).

Table 1: Parameters of Projectile Motion

v	Velocity
v_x	Horizontal Velocity
v_y	Vertical Velocity
u	Initial velocity
u_x	Initial Horizontal Velocity
u_y	Initial Vertical Velocity
θ	Initial Angle above horizontal axis
x	Horizontal Distance
y	Vertical Distance
t	Time
g	Acceleration due to gravity
V_a	Wind velocity in the positive horizontal direction

In order to plot the graph and to find the height of the projectile at any given displacement, we need an equation relating the x coordinates and y coordinates of the projectile. We have to derive to that equation from the basic ones as we have to take wind velocity into account.

In the horizontal axis:

$$x = (v_x + V_a)\ t$$

⇨ $x = (u_x + V_a)t$ [Since there is no horizontal acceleration]

⇨ $x = (u \cos \theta)t$

⇨ $t = \frac{x}{(u \cos \theta + V_a)}$ ----- eq (i)

In the vertical axis:

$$y = u_y t - \frac{1}{2} g t^2$$

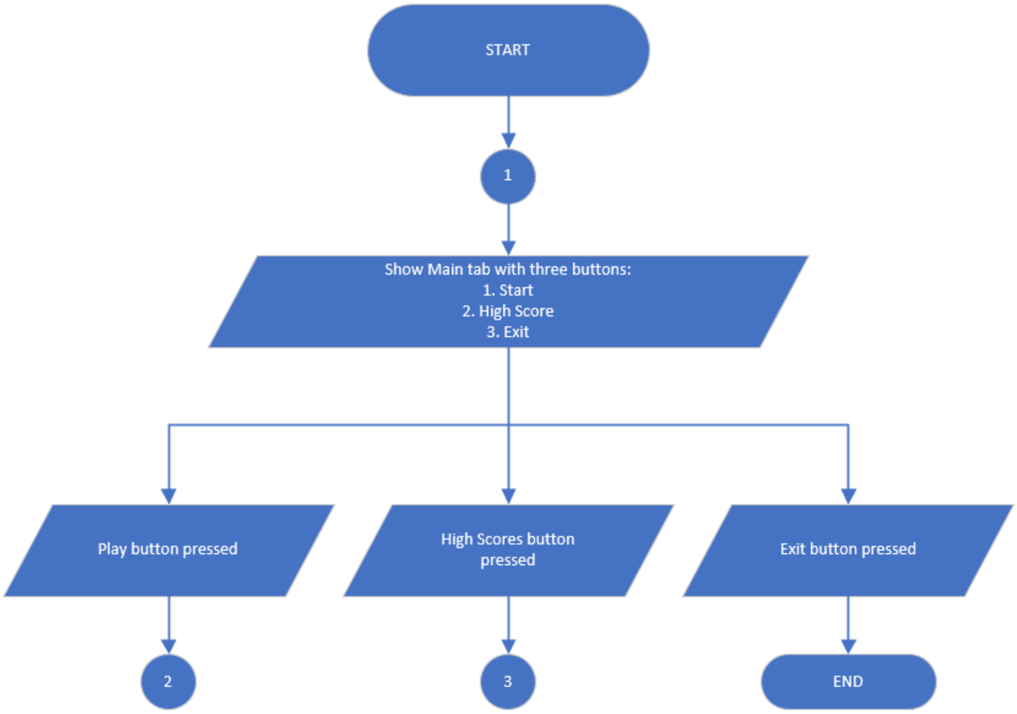
⇨ $y = u \sin \theta\ t - \frac{1}{2} g t^2$

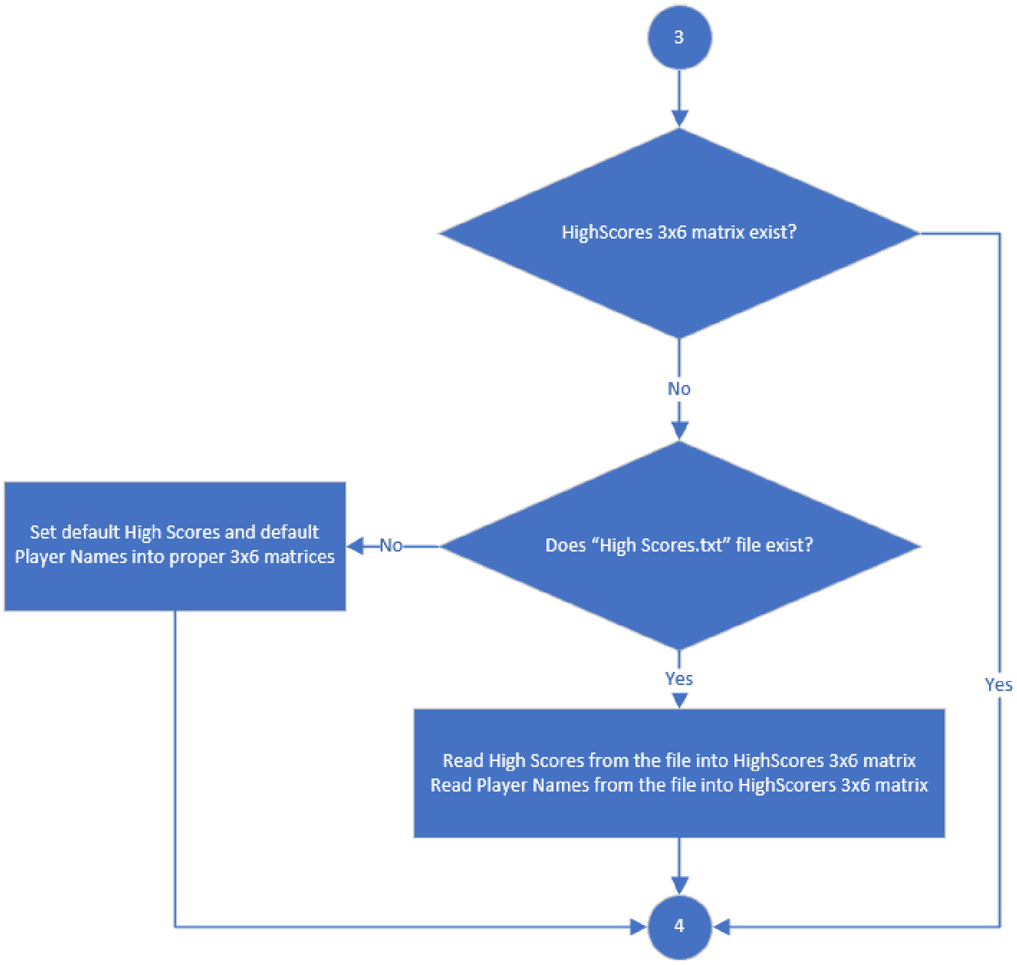
⇨ $y = u \sin \theta \times \frac{x}{(u \cos \theta + V_a)} - \frac{1}{2} g \left(\frac{x}{(u \cos \theta + V_a)} \right)^2$ [From eq (i)]

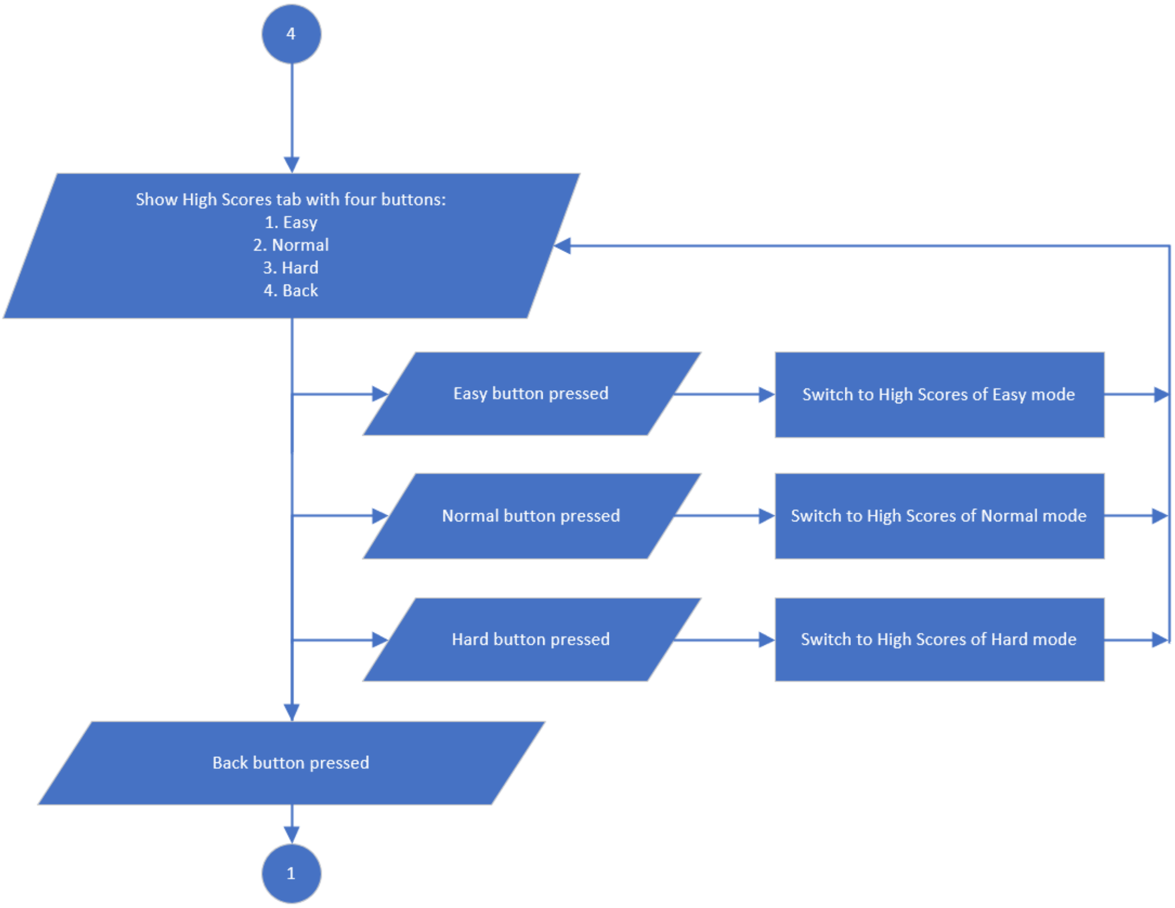
⇨ $y = \frac{ux \sin \theta}{(u \cos \theta + V_a)} - \frac{1}{2} \frac{g x^2}{(u \cos \theta + V_a)^2}$ ----- eq (ii)

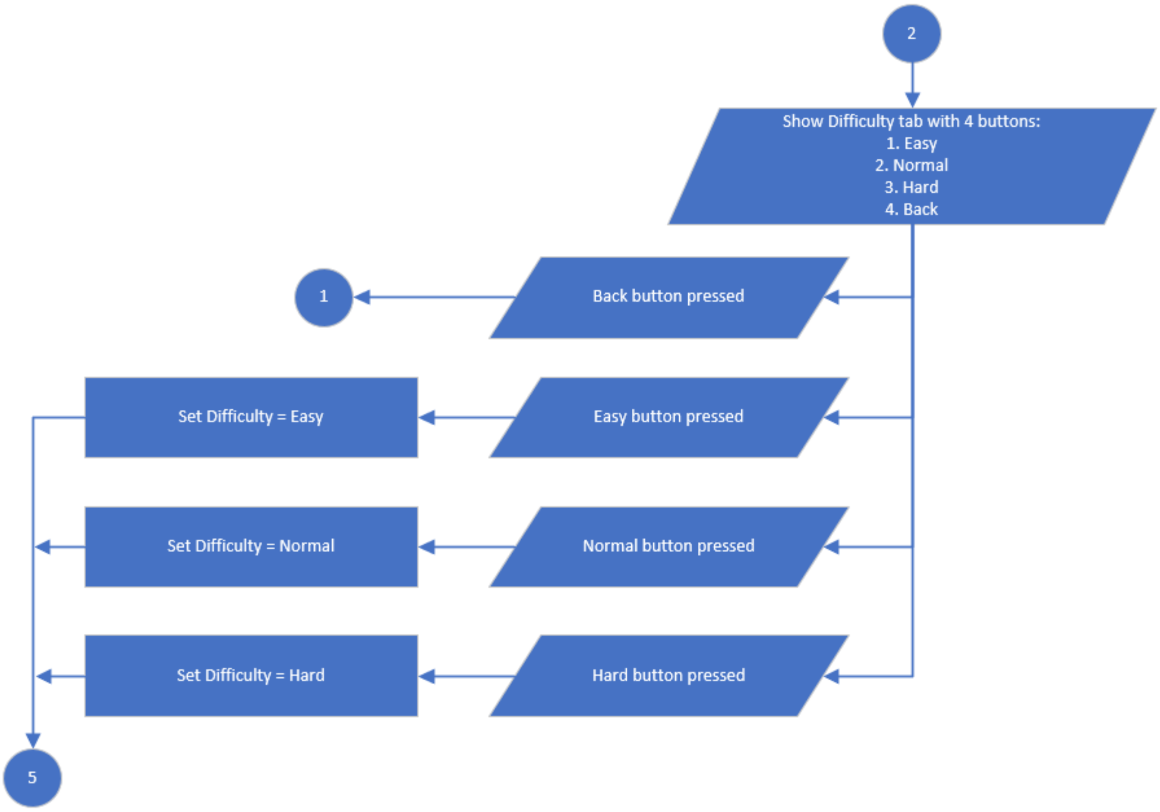
y-coordinate of the projectile at any given x-coordinate can be calculated using the equation (ii) above.

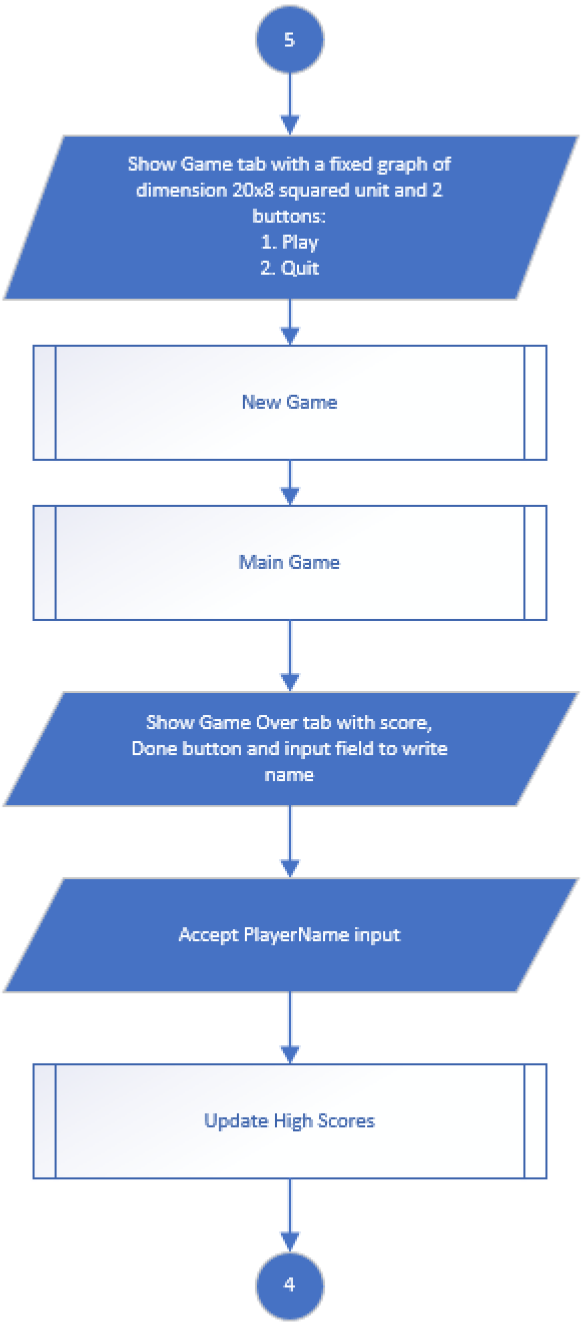
4. PROGRAM FLOW

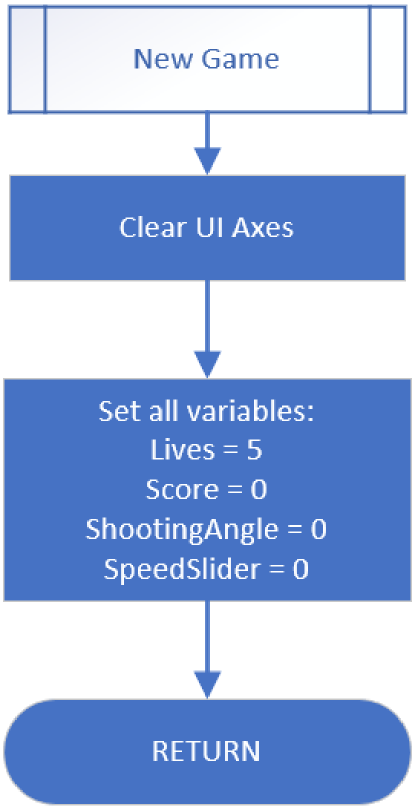


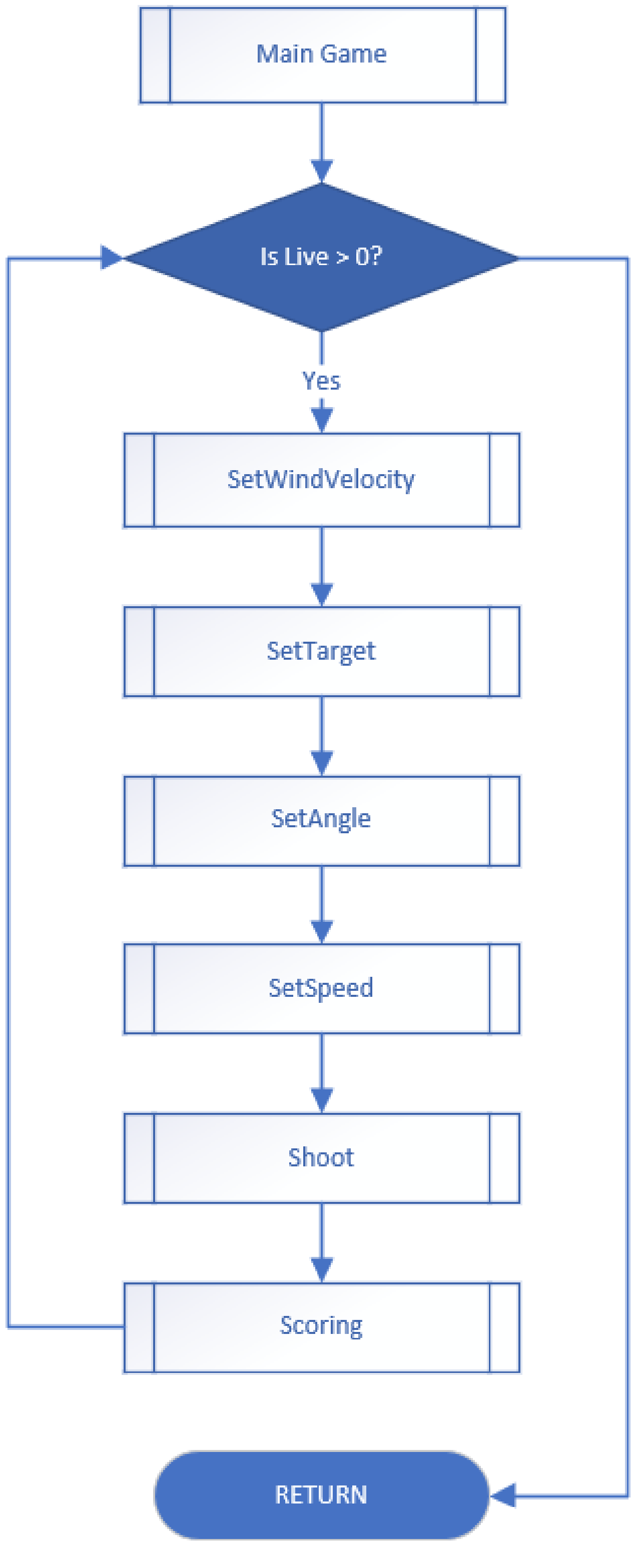

3
HighScores 3x6 matrix exist?
No
Does "High Scores.txt" file exist?
No
Set default High Scores and default Player Names into proper 3x6 matrices
Yes
Read High Scores from the file into HighScores 3x6 matrix
Read Player Names from the file into HighScorers 3x6 matrix
Yes
4

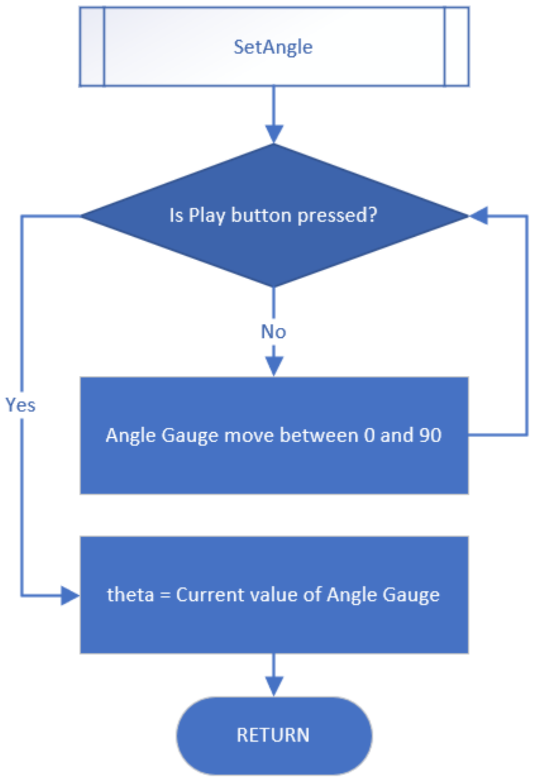

4
Show High Scores tab with four buttons:
1. Easy
2. Normal
3. Hard
4. Back
Easy button pressed
Switch to High Scores of Easy mode
Normal button pressed
Switch to High Scores of Normal mode
Hard button pressed
Switch to High Scores of Hard mode
Back button pressed
1

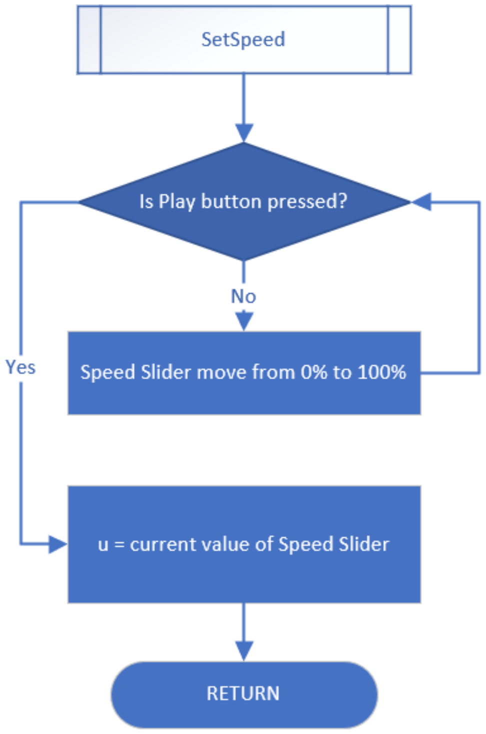

2
Show Difficulty tab with 4 buttons:
1. Easy
2. Normal
3. Hard
4. Back
1
Back button pressed
Set Difficulty = Easy
Easy button pressed
Set Difficulty = Normal
Normal button pressed
Set Difficulty = Hard
Hard button pressed
5

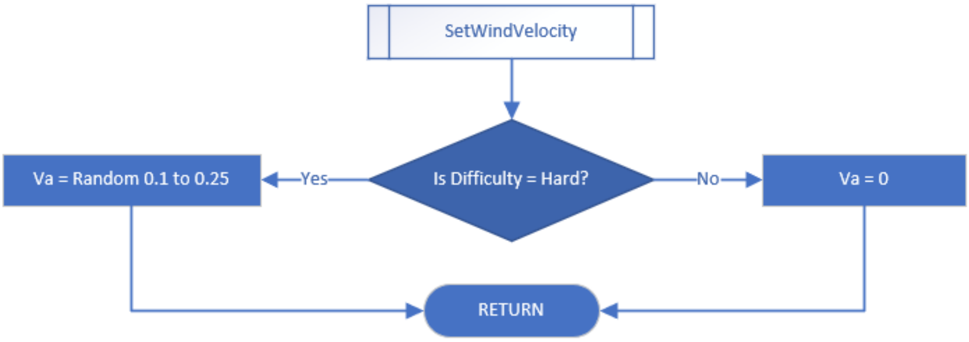

5
Show Game tab with a fixed graph of dimension 20x8 squared unit and 2 buttons:
1. Play
2. Quit
New Game
Main Game
Show Game Over tab with score, Done button and input field to write name
Accept PlayerName input
Update High Scores
4

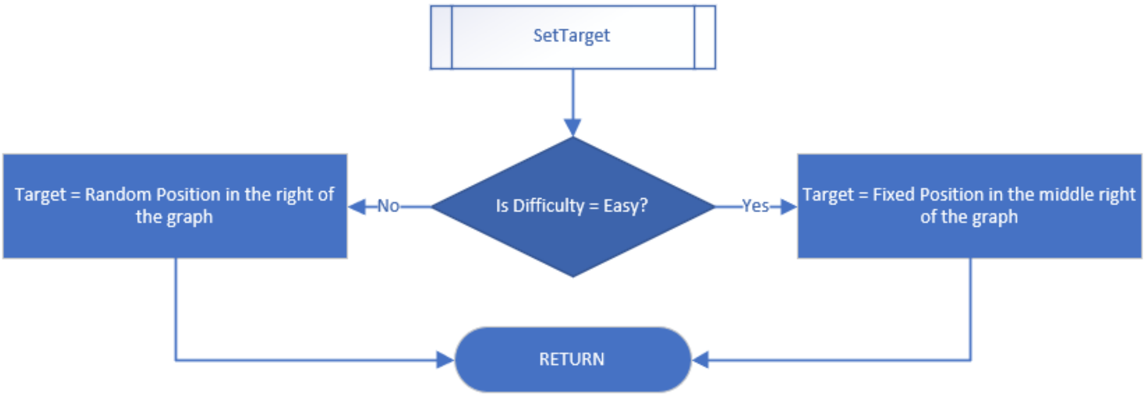

New Game
Clear UI Axes
Set all variables:
Lives = 5
Score = 0
ShootingAngle = 0
SpeedSlider = 0
RETURN

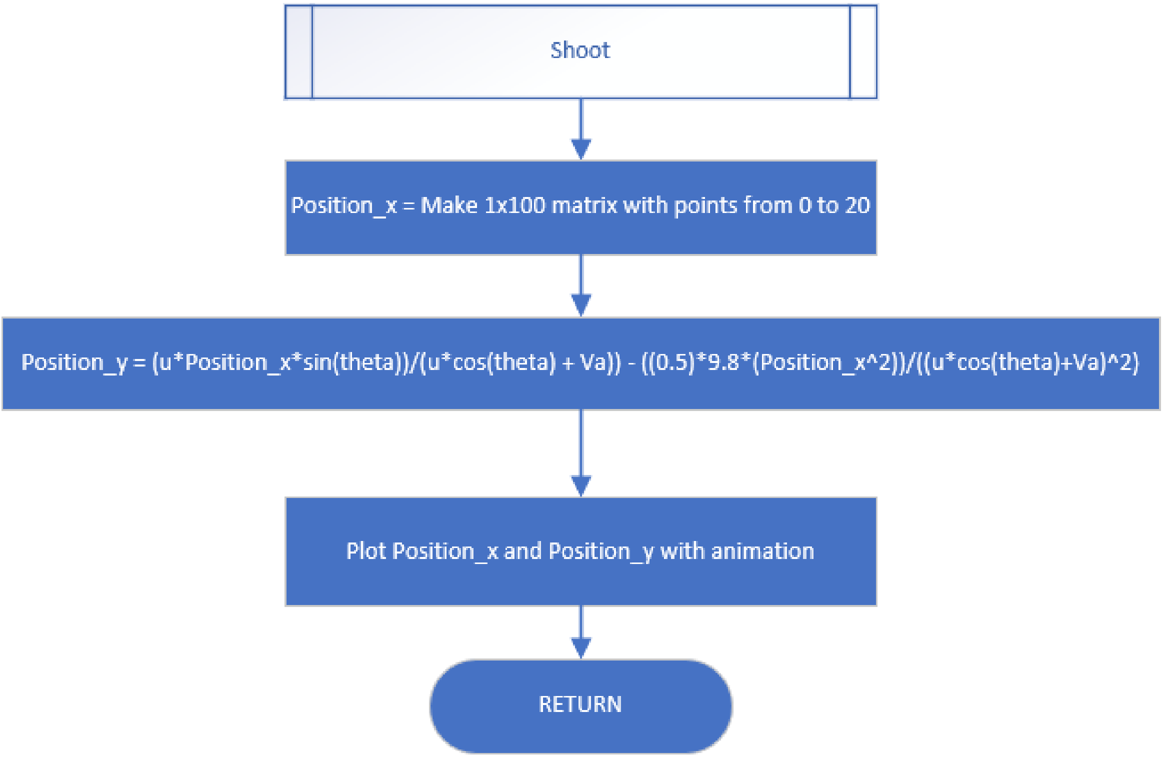

Main Game
Is Live > 0?
Yes
SetWindVelocity
SetTarget
SetAngle
SetSpeed
Shoot
Scoring
RETURN

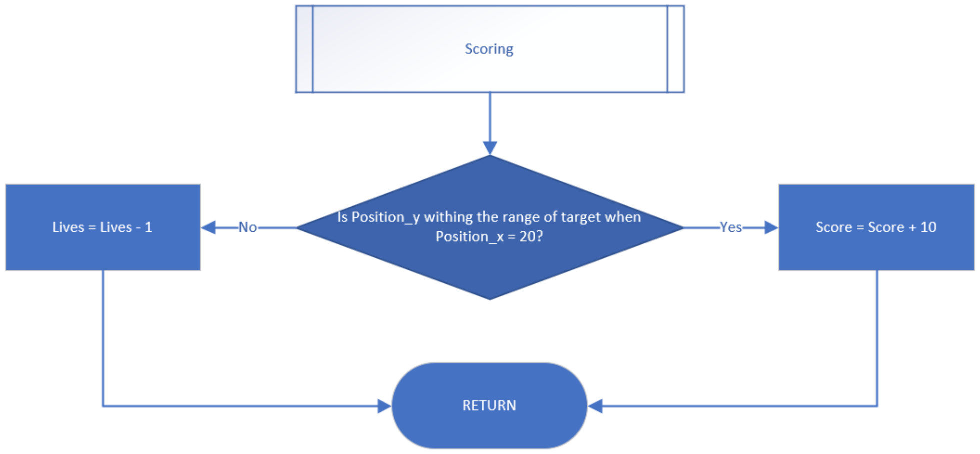

SetAngle
Is Play button pressed?
No
Yes
Angle Gauge move between 0 and 90
theta = Current value of Angle Gauge
RETURN

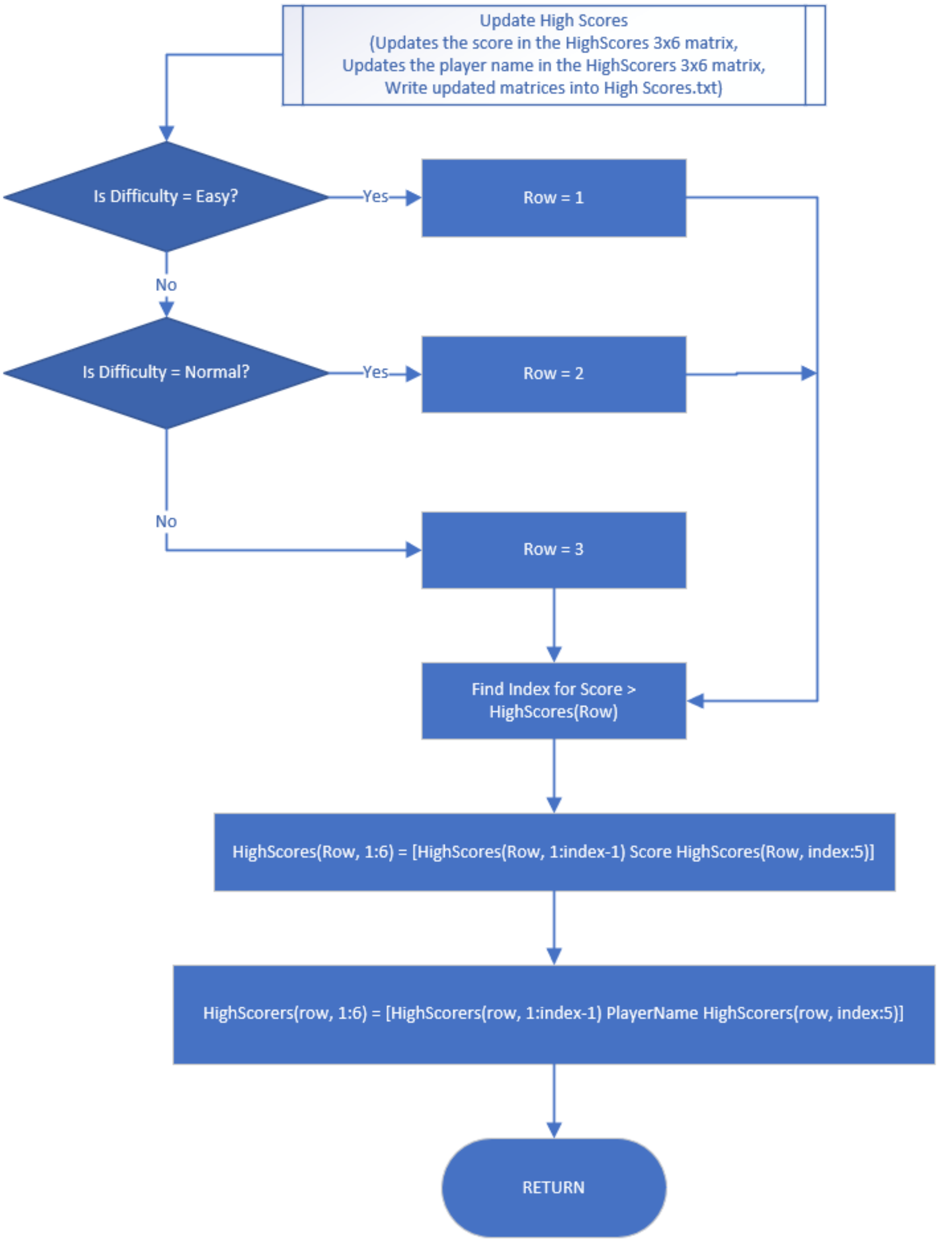

SetSpeed
Is Play button pressed?
No
Yes
Speed Slider move from 0% to 100%
u = current value of Speed Slider
RETURN


SetWindVelocity
Is Difficulty = Hard?
Yes
Va = Random 0.1 to 0.25
No
Va = 0
RETURN


SetTarget
Is Difficulty = Easy?
No
Target = Random Position in the right of the graph
Yes
Target = Fixed Position in the middle right of the graph
RETURN


Shoot
Position_x = Make 1x100 matrix with points from 0 to 20
Position_y = (u*Position_x*sin(theta))/(u*cos(theta) + Va)) - ((0.5)*9.8*(Position_x^2))/((u*cos(theta)+Va)^2)
Plot Position_x and Position_y with animation
RETURN


Scoring
Is Position_y withing the range of target when Position_x = 20?
No
Lives = Lives - 1
Yes
Score = Score + 10
RETURN


Update High Scores
(Updates the score in the HighScores 3x6 matrix,
Updates the player name in the HighScorers 3x6 matrix,
Write updated matrices into High Scores.txt)
Is Difficulty = Easy?
Yes
Row = 1
No
Is Difficulty = Normal?
Yes
Row = 2
No
Row = 3
Find Index for Score > HighScores(Row)
HighScores(Row, 1:6) = [HighScores(Row, 1:index-1) Score HighScores(Row, index:5)]
HighScorers(row, 1:6) = [HighScorers(row, 1:index-1) PlayerName HighScorers(row, index:5)]
RETURN

5. PROGRAM

5.1 FRONT END

The program is a 2D shooting game with Despicable Me and Minions movie series themes in mind to make it more interesting. (To learn more about these two movie series, check Appendix.) All the window from the user perspective have been explained in detail below.

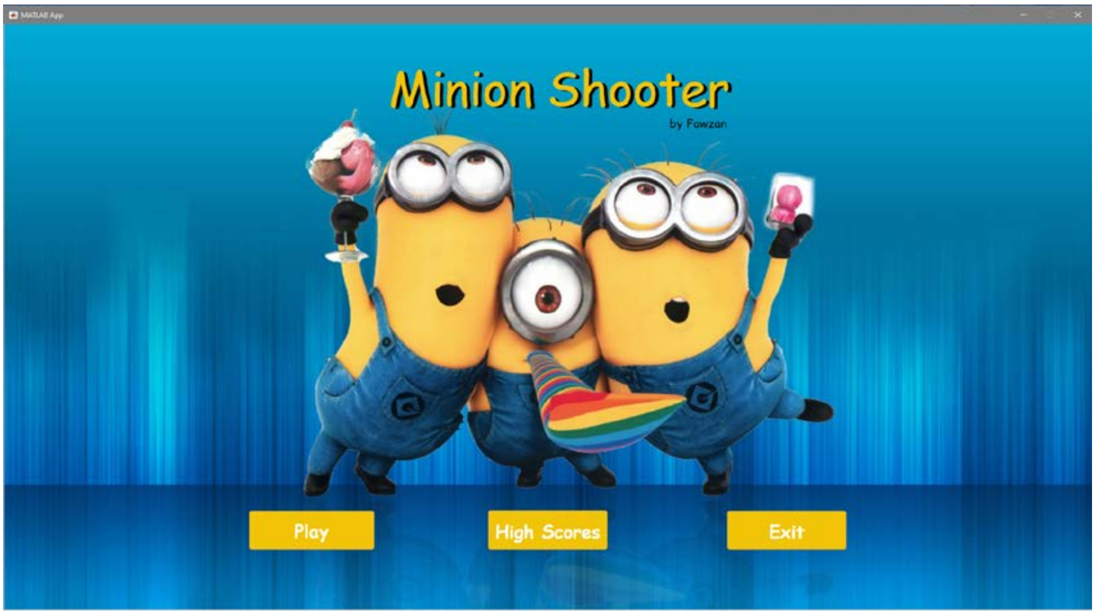


Figure 5: Main window of the program

As we can see in figure 5, *Main* window of the program contains three buttons. *Play*, *High Scores* and *Exit*. *Play* button takes the user to the *Difficulty* window, *High Scores* button takes to the *High Scores* window and *Exit* button exits the program.

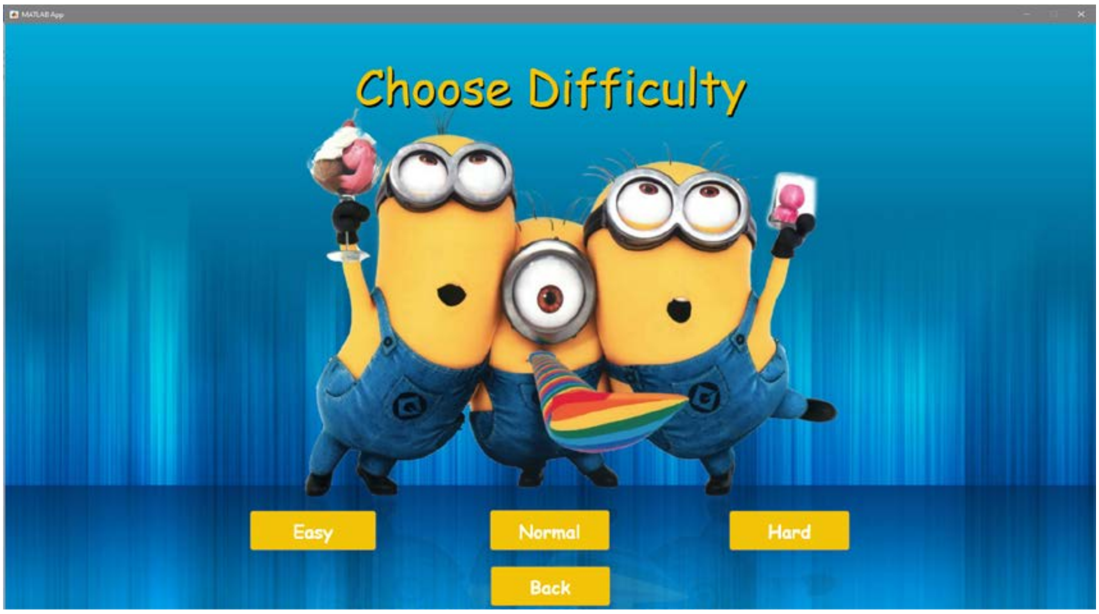
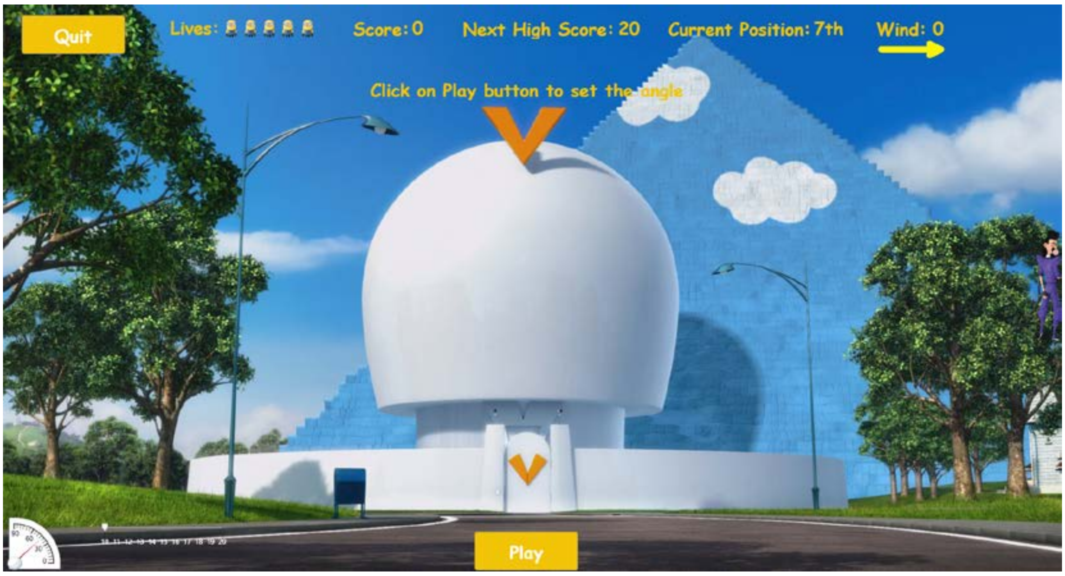


Figure 6: Difficulty window of the program

Difficulty window contains 4 buttons with the same design as *Main* window. *Easy*, *Normal* and *Hard* buttons take user to the *Game* window but with different situations. All these modes are explained later in the report. Lastly, *Back* button takes user back to the *Main* window.



Click on Play button to set the angle

Figure 7: Game window of the program

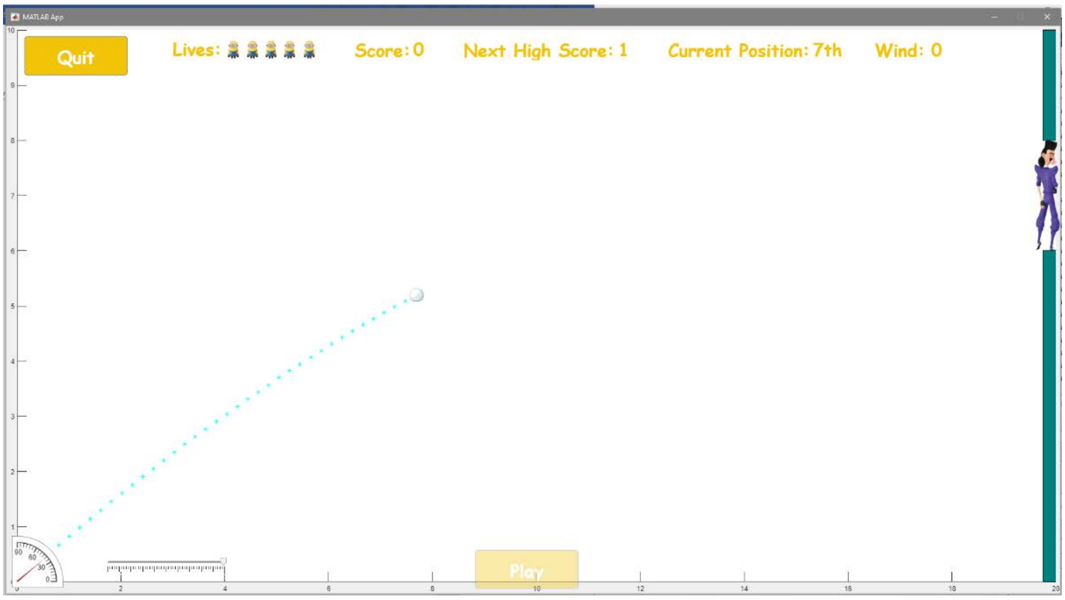


Figure 8: Behind the scene of Game window of the program

Game window will be a little bit different for each mode. But the elements and functionality are pretty much same. As we can see in figure 7 and 8, there are 5 *Lives* at the top left corner represented by little minions. Then we can see *Score* along with *Next High Score* to make it more challenging for the user. It also shows the current position among high scores. Then at the end, it shows the wind which flowing from left to write. For *Easy* and *Normal* mode, the wind velocity is zero. But for *Hard* mode there is a random wind flowing from left to right. At the bottom left, we can the snowball, which is the projectile to be shot at the target, purple colored character, on the right side of the window. In *Easy* mode, the target is set at the middle, and it is always fixed. But for *Normal* and *Hard* mode, it will randomly fluctuate up or down for each shot. In order to shoot the projectile, user need to click on the button at the bottom center of the window. Instructions are shown at the top of the window to make it more user friendly. The angle gauge at the bottom left corner will go up and down from zero to hundred and vice-versa until the user clicks on the *Play* button. Once the user clicks the button, the angle gauge will freeze, and the program will take the angle input. Then the speed slider right beside the angle gauge will go up and down until the user clicks *Play* button. With the angle and speed set, the program will shoot the snowball, taking the wind velocity into account, towards the target using the equation (ii) derived earlier. If the snowball hits the target, the *Score* will increase by 10. *Next High Score* and *Current Position* will be changed accordingly. Once the user runs out of all the lives, they will be taken to *Game Over* window. The user can also quit the game anytime they want using *Quit* button at the top left. It will also take them to *Game Over* window.

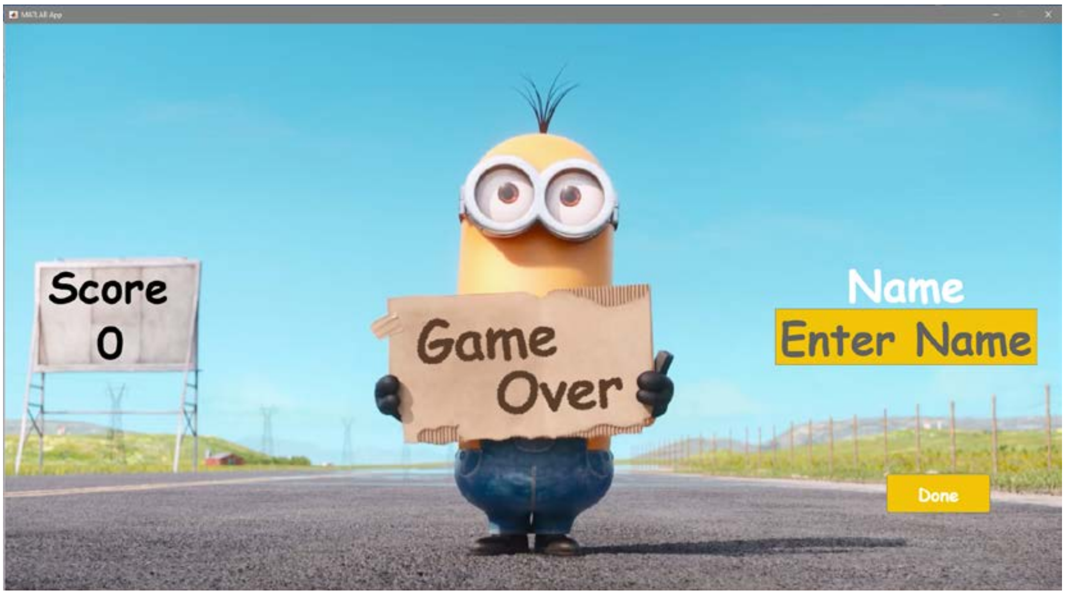


Figure 9: Game Over window of the program

Game Over window shows the *Score* on the left side of the window. On the right side, it asks for name input. If the user does not provide any name, then the program will take “FAW” as input. Maximum character of the name is three. If the user provides more than that, program will remove the rest. *Done* button will take the user to *High Scores* window.

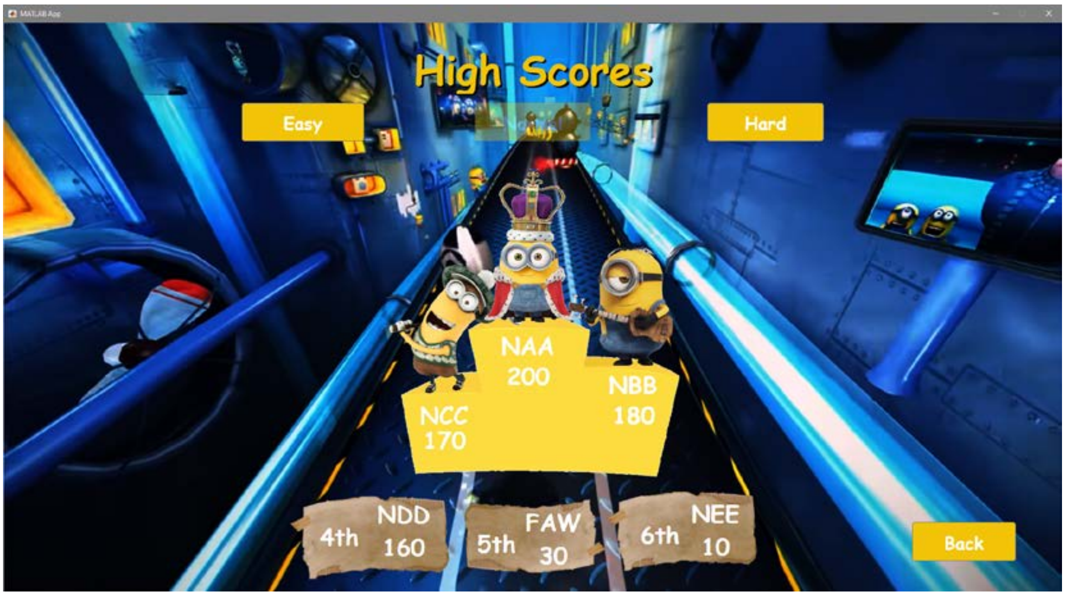


Figure 10: High Scores window of the program

This window has three buttons at the top, which let the user check the high scores for different modes. For example, figure 10 shows the high scores for normal mode, therefore *Normal* button is disabled. The program keeps record of top six scores and name of the scorers. *Back* button at the bottom right corner takes the user back to the *Main* menu.

5.2 BACK END

The program has callbacks, functions, and parameters. Callbacks are functions that execute in response to some user action, such as clicking on a button, providing input in text field and so on (MathWorks, n. d). There is total 16 callbacks in the program, which are listed in table 2 along with their description.

Table 2: Callbacks of the program in alphabetical order

CALLBACKS	**DESCRIPTION**
BackBtnDifficultyPushed	Executes when *Back* button is pushed in *Difficulty* window shown in figure 6
BackBtnHighScorePushed	Executes when *Back* button is pushed in *Game Over* window shown in figure 10
DifficultyBtnEasyPushed	Executes when *Easy* button is pushed in *Difficulty* window shown in figure 6
DifficultyBtnHardPushed	Executes when *Hard* button is pushed in *Difficulty* window shown in figure 6
DifficultyBtnNormalPushed	Executes when *Normal* button is pushed in *Difficulty* window shown in figure 6
DoneBtnGameOverPushed	Executes when *Done* button is pushed in *Game Over* window shown in figure 9
ExitBtnPushed	Executes when *Exit* button is pushed in *Main* window shown in figure 5
HighScoresBtnPushed	Executes when *High Scores* button is pushed in *Main* window shown in figure 5
NameFieldValueChanged	Executes when the value of the input text field in *Game Over* window shown in figure 9 is changed
PlayBtnGamePushed	Executes when the *Play* button is pushed in *Game* window shown in figure 7
PlayBtnMainPushed	Executes when the *Play* button is pushed in *Main* window shown in figure 5
QuitBtnPushed	Executes when the *Quit* button is pushed in *Game* window shown in figure 7
ScoreBtnEasyPushed	Executes when *Easy* button is pushed in *Game Over* window shown in figure 10
ScoreBtnHardPushed	Executes when *Hard* button is pushed in *Game Over* window shown in figure 10
ScoreBtnNormalPushed	Executes when *Normal* button is pushed in *Game Over* window shown in figure 10
startupFcn	Executes at the start of the program

Functions are a set of codes that were bundled together for their functionality. Benefit of having functions is that it adds modularity to the program, making it easier to understand the overall program. A function also allows us to reuse the same code without having to write them again or copying them. If there is a change to make, we only need to make changes in one place.

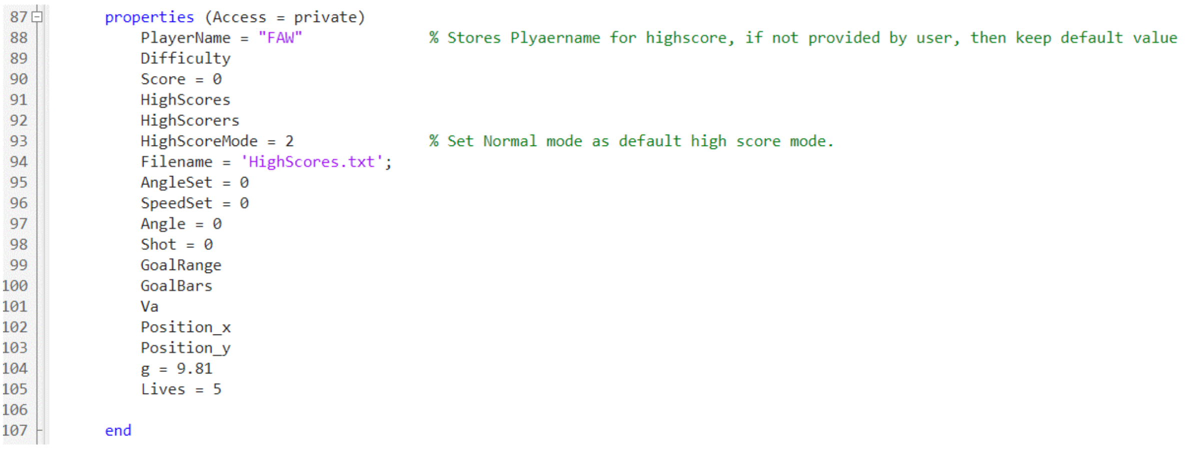
Table 3: Functions of the main program in alphabetical order

FUNCTIONS	**DESCRIPTION**
calculate(app)	Calculates the set of values for x and y coordinates of the projectile
mainGame(app)	Runs the main part of the game starting from setting angle till the game is over
newGame(app)	Starts a new game by resetting all variables
readAllHighScores(app)	Reads all the high scores from the file
scoring(app)	At the end of each shot, checks whether the projectile hit the target and updates *Score* as well as *Lives*
setAngle(app)	Moves the angle gauge up and down until it is set
setHighScores(app)	Sets the high score values and high scorers’ name based on the appropriate mode in the *High Scores* window shown in figure 5
setSpeed(app)	Moves the speed slider left and right until it is set
setTarget(app)	Sets the position of the target based on the appropriate mode
setWindVelocity(app)	Sets the wind velocity based on the appropriate mode
shoot(app)	Animates the projectile
updateHighScores(app)	Updates high scores at the end of each game and updates in the file if necessary
updateNextScore(app)	Updates *Next High Scores* in *Game* window shown in figure 7

Properties are global variables inside a program. Meaning it is a variable that every function inside an app can read and write. It is the best way to share data within an individual app (MathWorks, n.d.). There are 17 properties in this program. All these properties have been listed along with their data types and description in table 4 below.

Table 4: Properties of the program in alphabetical order

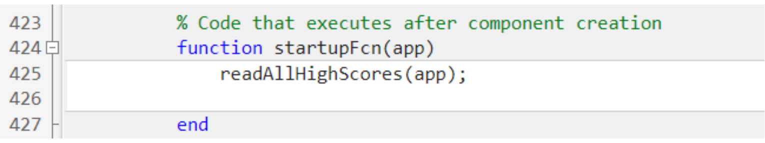
PROPERTIES	DATA TYPE	DESCRIPTION
Angle	Integer (0-90)	Stores the angle of the projectile. Value is in between 0 and 90.
AngleSet	Integer (0, 1)	Indicates if the angle is set or not. It acts as a Boolean. 1 represents true and 0 represents false
Difficulty	Integer (1, 2, 3)	Indicates the difficulty of the game. 1, 2 and 3 represents easy, normal, and hard respectively
FileName	String	Stores the name of the file for saving high scores
g	Double (9.81)	Stores the value of the acceleration due to gravity, which is 9.81
GoalBars	Array of 2 rectangles	Stores two rectangles, one for upper bar and one for lower bar of the target.
GoalRange	Array of 2 integers	Stores the upper point and the lower point of the target
HighScoreMode	Integer (1, 2, 3)	Indicates the mode of the high scores. 1, 2 and 3 represents easy, normal, and hard respectively
HighScorers	3x6 Matrix of integers	Stores high scores of 6 people for 3 different modes
HighScores	3x6 Matrix of strings	Stores the name of 6 people for 3 different modes
Lives	Integers (0-5)	Stores the quantity of lives in a game
PlayerName	String	Stores the name of the player at the end of each game. Default value is “FAW”.
Position_x	Array of 100 integers	Stores the x coordinates of the projectile
Position_y	Array of 100 integers	Stores the y coordinates of the projectile
Score	Integer	Stores the score of a game
SpeedSet	Integer (0, 1)	Indicates if the speed is set or not. It acts as a Boolean. 1 represents true and 0 represents false
Va	Double (0-0.25)	Stores the wind velocity



```
87      properties (Access = private)
88          PlayerName = "FAW"            % Stores Plyaername for highscore, if not provided by user, then keep default value
89          Difficulty
90          Score = 0
91          HighScores
92          HighScorers
93          HighScoreMode = 2             % Set Normal mode as default high score mode.
94          Filename = 'HighScores.txt';
95          AngleSet = 0
96          SpeedSet = 0
97          Angle = 0
98          Shot = 0
99          GoalRange
100         GoalBars
101         Va
102         Position_x
103         Position_y
104         g = 9.81
105         Lives = 5
106
107     end
```

Figure 11: Properties of the program

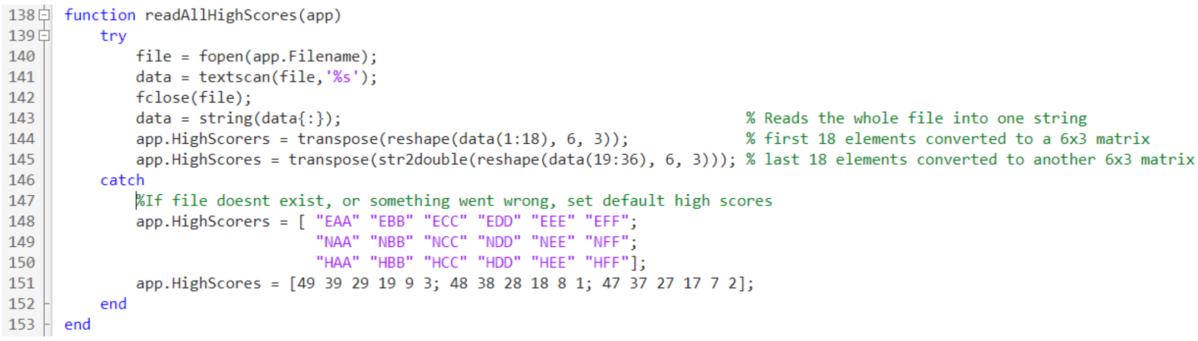
Figure 11 shows all the properties all the program. These properties have been described in table 4. Some of these properties have been initialized. For example: *HighScoreMode* is set to 1 in line 93, which represents *Normal* mode. Meaning, by default it will show the high scores for *Normal* mode. Also, we can see the *Filename* for high scores in line 94.



```
423         % Code that executes after component creation
424         function startupFcn(app)
425             readAllHighScores(app);
426
427         end
```

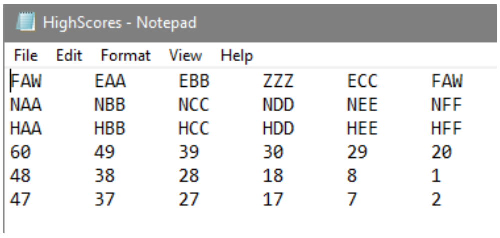
Figure 12: Callbacks of the program on startup

For the code, we will go along with the flow of the program to get a clearer idea instead of going from top to bottom. When the program starts, a startup function named *startupFcn* gets called which reads all the high scores from the file using *readAllHighScores* function in line 425. We read from the file once at the startup and keep updating it in the program as well as the file through the game.



```
function readAllHighScores(app)
    try
        file = fopen(app.Filename);
        data = textscan(file,'%s');
        fclose(file);
        data = string(data{:});                                    % Reads the whole file into one string
        app.HighScorers = transpose(reshape(data(1:18), 6, 3));           % first 18 elements converted to a 6x3 matrix
        app.HighScores = transpose(str2double(reshape(data(19:36), 6, 3))); % last 18 elements converted to another 6x3 matrix
    catch
        %If file doesnt exist, or something went wrong, set default high scores
        app.HighScorers = [ "EAA" "EBB" "ECC" "EDD" "EEE" "EFF";
                            "NAA" "NBB" "NCC" "NDD" "NEE" "NFF";
                            "HAA" "HBB" "HCC" "HDD" "HEE" "HFF"];
        app.HighScores = [49 39 29 19 9 3; 48 38 28 18 8 1; 47 37 27 17 7 2];
    end
end
```

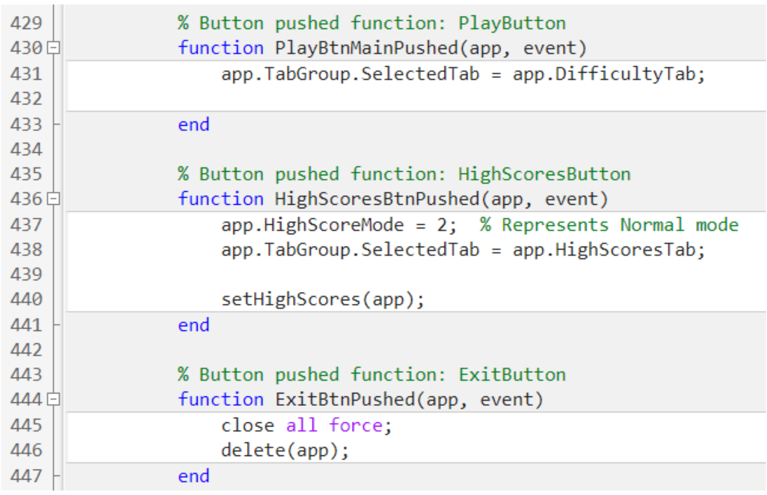
Figure 13: readAllHighScores function



```
FAW     EAA     EBB     ZZZ     ECC     FAW
NAA     NBB     NCC     NDD     NEE     NFF
HAA     HBB     HCC     HDD     HEE     HFF
60      49      39      30      29      20
48      38      28      18      8       1
47      37      27      17      7       2
```

Figure 14: High Scores text file

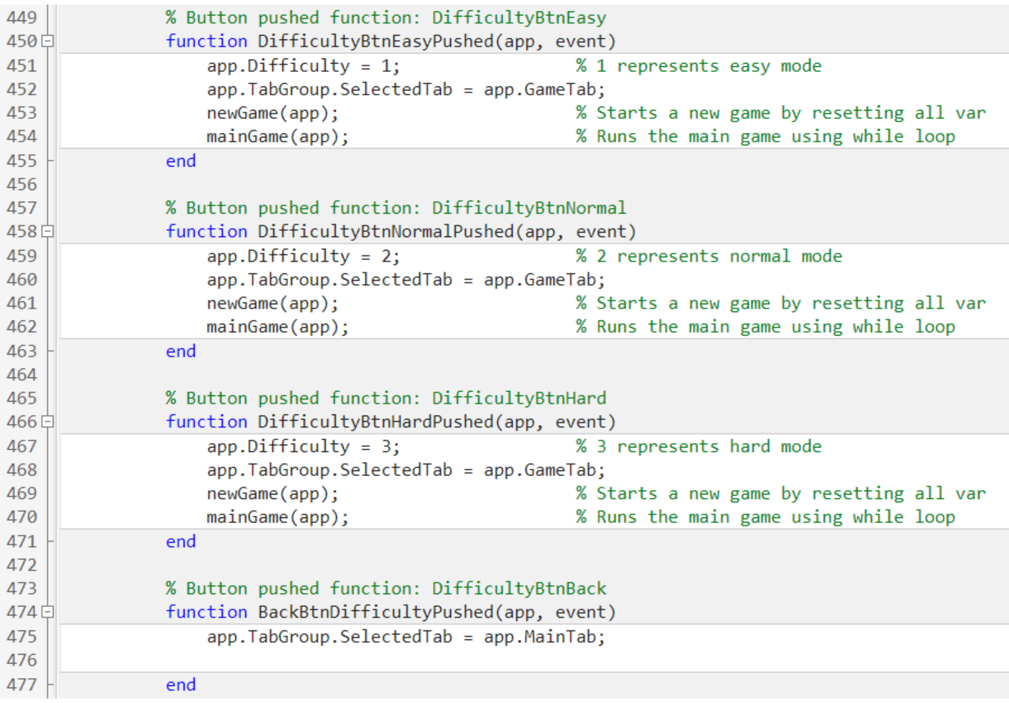
This functions first tries to read the high scores from the file. If the exists, it proceeds otherwise it sets some default value into two 3x6 matrix property named *HighScorers* and *HighScores* as seen in figure 13, line 148-151. In the file in figure 14, high scores are stored in 6 lines, where first 3 lines are the name of the top scorers from three different modes: easy, normal, and hard respectively. Similarly, the next 3 lines are the top scores of these people. The program reads the whole file into a single string in line 141 and then converts into an a 1x18 matrix in line 143. Then the first 18 elements are converted into a 3x6 matrix which holds the name of the top scorers inside *HighScorers* property of the program. Then the next 18 elements are converted and stored similarly into *HighScore* property.



```
429         % Button pushed function: PlayButton
430         function PlayBtnMainPushed(app, event)
431             app.TabGroup.SelectedTab = app.DifficultyTab;
432 
433         end
434 
435         % Button pushed function: HighScoresButton
436         function HighScoresBtnPushed(app, event)
437             app.HighScoreMode = 2;  % Represents Normal mode
438             app.TabGroup.SelectedTab = app.HighScoresTab;
439 
440             setHighScores(app);
441         end
442 
443         % Button pushed function: ExitButton
444         function ExitBtnPushed(app, event)
445             close all force;
446             delete(app);
447         end
```

Figure 15: Callbacks of the Main window

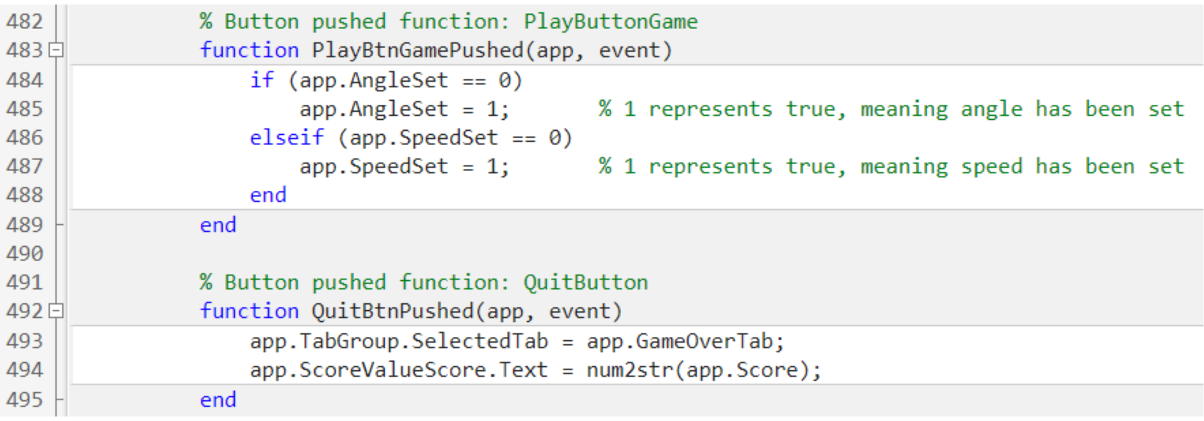
After that, *Main* window is shown from figure 5 as it is the first tab among 5 tabs. There are three buttons, each of them has their own callbacks. *Play* button just changes to *Difficulty* tab as seen in line 431 and *Exit* button closes everything and deletes the app. *High Score* button sets the high score mode and changes to *High Scores* tab as seen in line 437 and 438. Then calls the function *setHighScores* in line 440 to set the high scores of the predefined mode on the window.



```
        % Button pushed function: DifficultyBtnEasy
        function DifficultyBtnEasyPushed(app, event)
            app.Difficulty = 1;                 % 1 represents easy mode
            app.TabGroup.SelectedTab = app.GameTab;
            newGame(app);                       % Starts a new game by resetting all var
            mainGame(app);                      % Runs the main game using while loop
        end

        % Button pushed function: DifficultyBtnNormal
        function DifficultyBtnNormalPushed(app, event)
            app.Difficulty = 2;                 % 2 represents normal mode
            app.TabGroup.SelectedTab = app.GameTab;
            newGame(app);                       % Starts a new game by resetting all var
            mainGame(app);                      % Runs the main game using while loop
        end

        % Button pushed function: DifficultyBtnHard
        function DifficultyBtnHardPushed(app, event)
            app.Difficulty = 3;                 % 3 represents hard mode
            app.TabGroup.SelectedTab = app.GameTab;
            newGame(app);                       % Starts a new game by resetting all var
            mainGame(app);                      % Runs the main game using while loop
        end

        % Button pushed function: DifficultyBtnBack
        function BackBtnDifficultyPushed(app, event)
            app.TabGroup.SelectedTab = app.MainTab;

        end
```

Figure 16: Callbacks of the Difficulty window

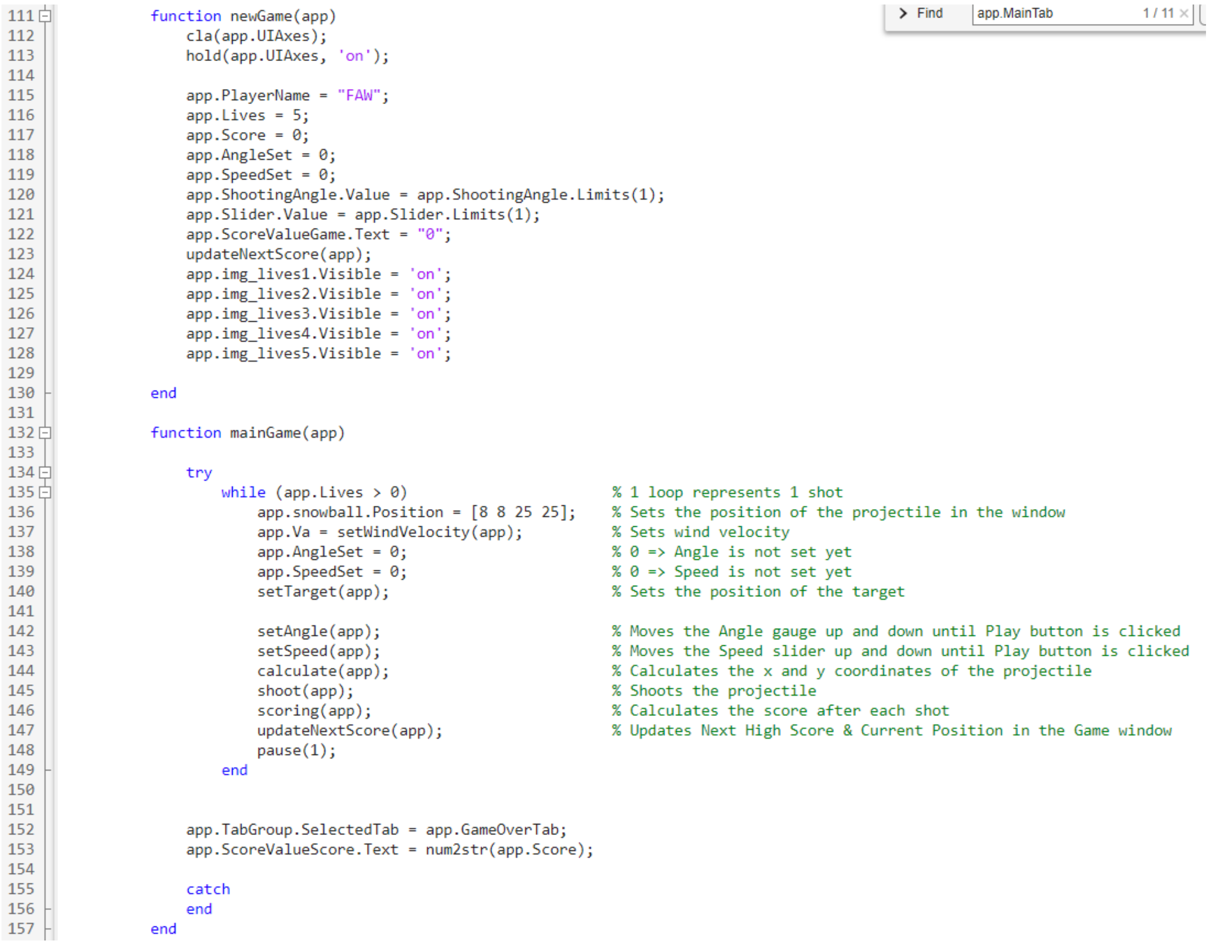
In the *Difficulty* window, there are four buttons, referring to figure 6. Clicking on *Easy*, *Normal* or *Hard* buttons sets the value of the *Difficulty* property to 1, 2 or 3 which represents *Easy*, *Normal* or *Hard* mode respectively. Then all three buttons switch from *Difficulty* to *Game* tab, calls two functions named *newGame* and *mainGame. newGame* starts a new game by clearing out old values and resetting all variables. *mainGame* runs the main game using a while loop until *Lives* property in the game is not zero. *Back* button takes the user back to the previous window, which is *Main* window in this case. At first, we will see the callbacks of the next window and then we will go into the functions called in figure 16.



```
        % Button pushed function: PlayButtonGame
        function PlayBtnGamePushed(app, event)
            if (app.AngleSet == 0)
                app.AngleSet = 1;       % 1 represents true, meaning angle has been set
            elseif (app.SpeedSet == 0)
                app.SpeedSet = 1;       % 1 represents true, meaning speed has been set
            end
        end

        % Button pushed function: QuitButton
        function QuitBtnPushed(app, event)
            app.TabGroup.SelectedTab = app.GameOverTab;
            app.ScoreValueScore.Text = num2str(app.Score);
        end
```

Figure 17: Callbacks of Game window

In the game window, there are two buttons as shown in figure 7. Clicking on *Play* button sets *AngleSet* to 1 and clicking again sets *SpeedSet* to 1. These two variables freeze the movement of angle gauge and speed slider in the game, which we will see later. *Quit* button lets the user quit the game anytime they want. It takes them to the *Game Over* tab.

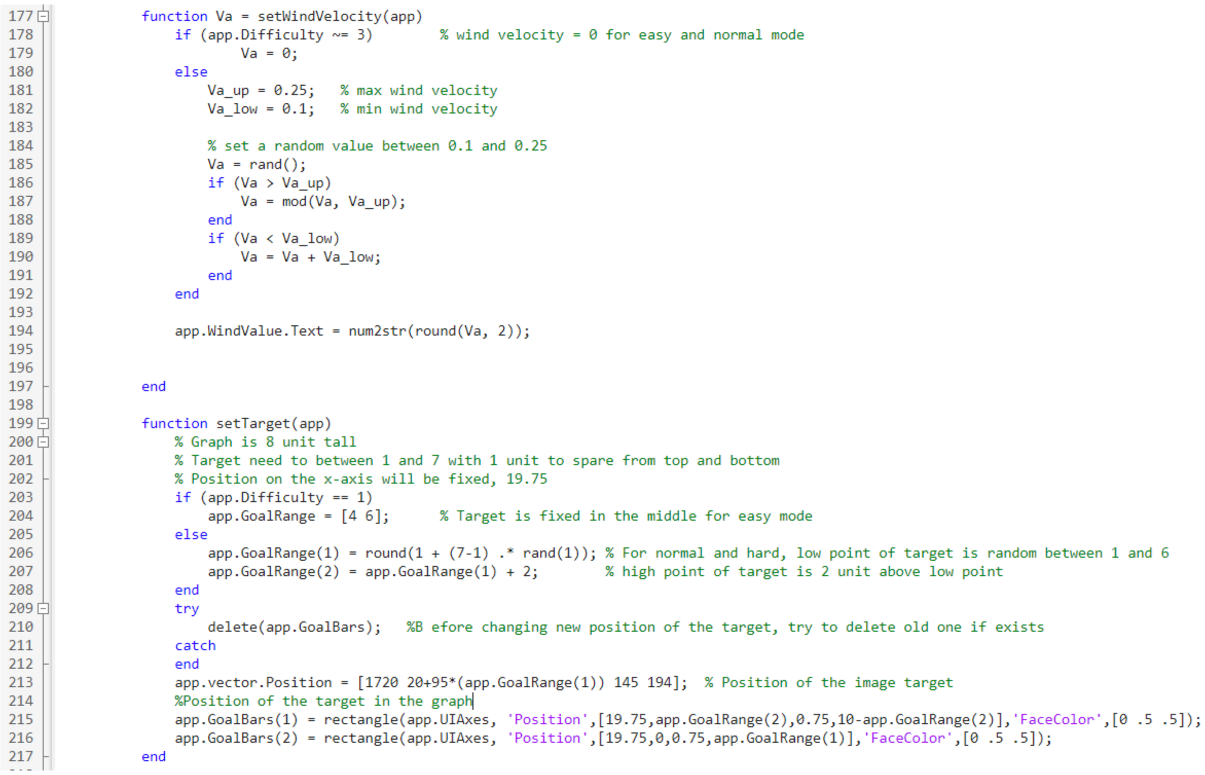


```
111    function newGame(app)
112        cla(app.UIAxes);
113        hold(app.UIAxes, 'on');
114
115        app.PlayerName = "FAW";
116        app.Lives = 5;
117        app.Score = 0;
118        app.AngleSet = 0;
119        app.SpeedSet = 0;
120        app.ShootingAngle.Value = app.ShootingAngle.Limits(1);
121        app.Slider.Value = app.Slider.Limits(1);
122        app.ScoreValueGame.Text = "0";
123        updateNextScore(app);
124        app.img_lives1.Visible = 'on';
125        app.img_lives2.Visible = 'on';
126        app.img_lives3.Visible = 'on';
127        app.img_lives4.Visible = 'on';
128        app.img_lives5.Visible = 'on';
129
130    end
131
132    function mainGame(app)
133
134        try
135            while (app.Lives > 0)                       % 1 loop represents 1 shot
136                app.snowball.Position = [8 8 25 25];    % Sets the position of the projectile in the window
137                app.Va = setWindVelocity(app);          % Sets wind velocity
138                app.AngleSet = 0;                       % 0 => Angle is not set yet
139                app.SpeedSet = 0;                       % 0 => Speed is not set yet
140                setTarget(app);                         % Sets the position of the target
141
142                setAngle(app);                          % Moves the Angle gauge up and down until Play button is clicked
143                setSpeed(app);                          % Moves the Speed slider up and down until Play button is clicked
144                calculate(app);                         % Calculates the x and y coordinates of the projectile
145                shoot(app);                             % Shoots the projectile
146                scoring(app);                           % Calculates the score after each shot
147                updateNextScore(app);                   % Updates Next High Score & Current Position in the Game window
148                pause(1);
149            end
150
151
152        app.TabGroup.SelectedTab = app.GameOverTab;
153        app.ScoreValueScore.Text = num2str(app.Score);
154
155        catch
156        end
157    end
```

Figure 18: newGame and mainGame function

Even though we cannot see the graph in the *Game* window in figure 7, there is a graph behind the image. Actual game is running on this graph. For each new game, the program is clearing out that graph in line 112 and holding all the plots as it is necessary to keep the projectile as well as the goal bars on the graph. Then it is resetting some properties from line 115-122. After that, it is calling *updateNextScore* function, which gets the next high score for the appropriate mode and shows on the screen for the user to make it more challenging. As shown in figure 7, five small images represent lives, which need to be made visible at the start of each game.

Main section of the *mainGame* function is wrapped inside a *try* block. It is necessary because when the user is exiting the game, there might be processes which are on hold in the background. These processes try to throw error but cannot because of this *try* block. The function is running a while loop until *Lives* property is above zero. At first, position of the snowball image is set on the window. Then *setWindVelocity* function gets called which returns the wind velocity and assigns it to the *Va* property. *AngleSet* and *SpeedSet* properties, which acts like a Boolean, are set to zero. Target on the window is set using the function named *setTarget.* It sets the target on the graph as well as the image on the window. Then *setAngle* function gets called, which moves the angle gauge up and down until the user hits the *Play* button seen in figure 7. As we have seen in figure17, this sets the *AngleSet* variable to 1, meaning the angle has been set. Therefore, program gets out of the *setAngle* function then goes into *setSpeed.* Like *setAngle* function, this function moves the speed slider up and down until the user clicks the *Play* button again. Once the angle and speed has been set, the program proceeds to execute the necessary calculations before shooting using *calculate* function. As soon as the calculation is complete, the program calls the *shoot* function which then shoots the projectile. It animates the snowball image and the plot in the background at the same time. The program then checks whether the projectile hit the target or missed it using *scoring* function and updates the next high score in the *Game* window using *updateNextScore.* The program pauses itself for a second before moving on. Once the user runs out of all the lives, It switches to *Game Over* tab and updates the score in that window.

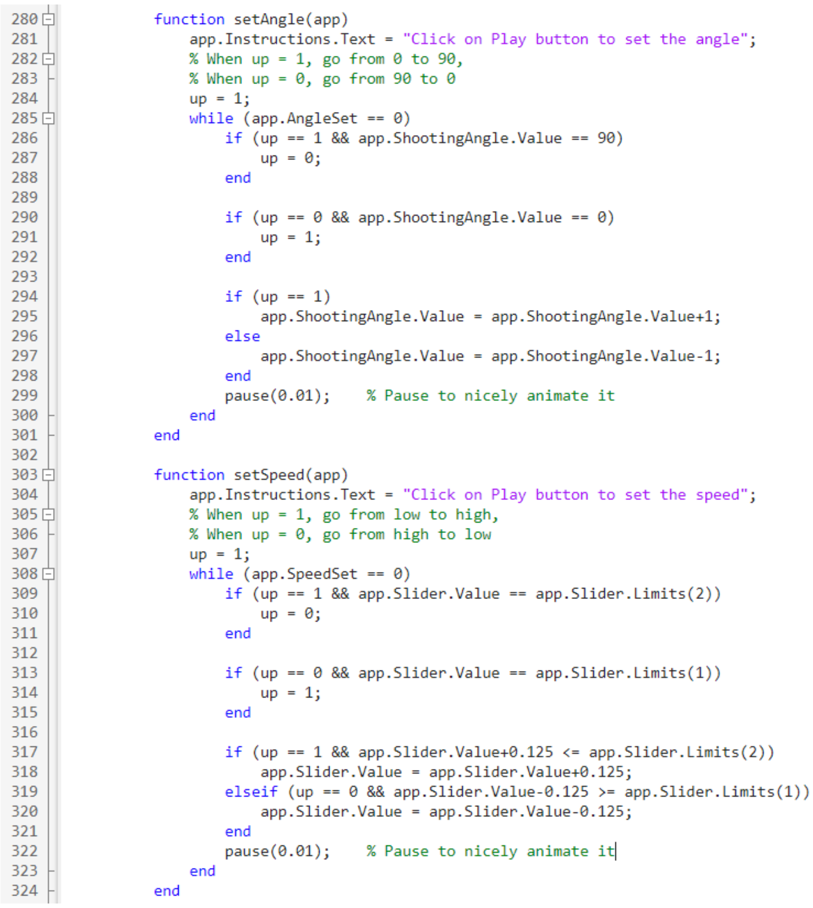


```
177     function Va = setWindVelocity(app)
178         if (app.Difficulty ~= 3)        % wind velocity = 0 for easy and normal mode
179                 Va = 0;
180         else
181             Va_up = 0.25;   % max wind velocity
182             Va_low = 0.1;   % min wind velocity
183 
184             % set a random value between 0.1 and 0.25
185             Va = rand();
186             if (Va > Va_up)
187                 Va = mod(Va, Va_up);
188             end
189             if (Va < Va_low)
190                 Va = Va + Va_low;
191             end
192         end
193 
194         app.WindValue.Text = num2str(round(Va, 2));
195 
196 
197     end
198 
199     function setTarget(app)
200         % Graph is 8 unit tall
201         % Target need to between 1 and 7 with 1 unit to spare from top and bottom
202         % Position on the x-axis will be fixed, 19.75
203         if (app.Difficulty == 1)
204             app.GoalRange = [4 6];      % Target is fixed in the middle for easy mode
205         else
206             app.GoalRange(1) = round(1 + (7-1) .* rand(1)); % For normal and hard, low point of target is random between 1 and 6
207             app.GoalRange(2) = app.GoalRange(1) + 2;        % high point of target is 2 unit above low point
208         end
209         try
210             delete(app.GoalBars);   %B efore changing new position of the target, try to delete old one if exists
211         catch
212         end
213         app.vector.Position = [1720 20+95*(app.GoalRange(1)) 145 194];  % Position of the image target
214         %Position of the target in the graph
215         app.GoalBars(1) = rectangle(app.UIAxes, 'Position',[19.75,app.GoalRange(2),0.75,10-app.GoalRange(2)],'FaceColor',[0 .5 .5]);
216         app.GoalBars(2) = rectangle(app.UIAxes, 'Position',[19.75,0,0.75,app.GoalRange(1)],'FaceColor',[0 .5 .5]);
217     end
```

Figure 19: setWindVelocity and setTarget functions

In the *setWindVelocity* function, wind velocity is zero for easy and normal mode. For hard mode it is a random value between 0.1 and 0.25. *rand* function used in line 185 generates a random double between 0 and 1. If it is smaller than 0.1, than 0.1 is added to get wind velocity and if its greater than 0.25 then remainder is taken from dividing generated number by 0.25. Finally wind velocity is returned after updating in the *Game* window.

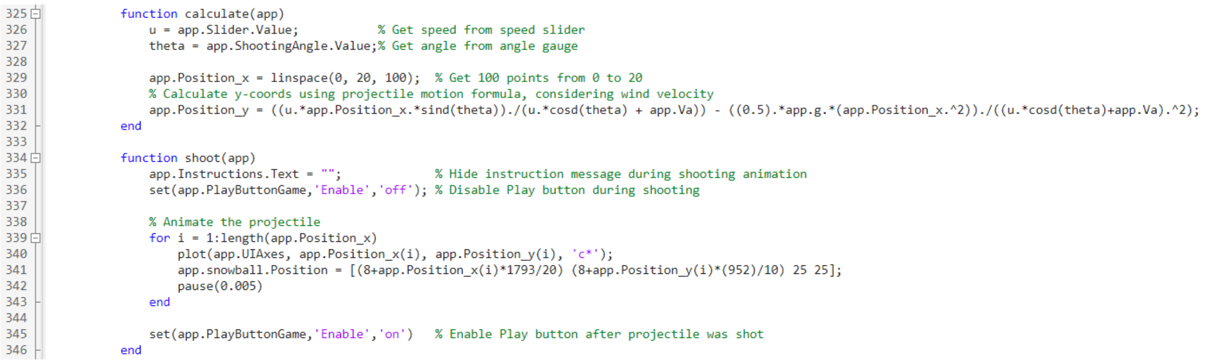
As mentioned before, the actual game is happening in the plot behind the image in *Game* window shown in figure 7 and 8. The plot is 8 unit tall, meaning that the target needs to be within 8 units. The target is two unit tall here. For the easy mode, that 2 unit is from 4 to 6. But for normal and hard mode, a random number is generated for low point between 1 and 6, then 2 is added to get the high point of the target in line 206 and 207. For each shot, a new target will appear. Before drawing the target, old target on the plot needs to be removed as the hold is on for the graph shown in figure 19 line 119. The program tries to delete the target in line 210. A try block was necessary, because for the first shot, there is no old target to delete. Once the target position is calculated, the image is set in that position in line 213. For the coordinates of the image, the target is converted from plot coordinates to window coordinates using approximate calculation and trial-error. For the target in the graph, two rectangles are drawn in line 215 and 216.



```
        function setAngle(app)
            app.Instructions.Text = "Click on Play button to set the angle";
            % When up = 1, go from 0 to 90,
            % When up = 0, go from 90 to 0
            up = 1;
            while (app.AngleSet == 0)
                if (up == 1 && app.ShootingAngle.Value == 90)
                    up = 0;
                end

                if (up == 0 && app.ShootingAngle.Value == 0)
                    up = 1;
                end

                if (up == 1)
                    app.ShootingAngle.Value = app.ShootingAngle.Value+1;
                else
                    app.ShootingAngle.Value = app.ShootingAngle.Value-1;
                end
                pause(0.01);    % Pause to nicely animate it
            end
        end

        function setSpeed(app)
            app.Instructions.Text = "Click on Play button to set the speed";
            % When up = 1, go from low to high,
            % When up = 0, go from high to low
            up = 1;
            while (app.SpeedSet == 0)
                if (up == 1 && app.Slider.Value == app.Slider.Limits(2))
                    up = 0;
                end

                if (up == 0 && app.Slider.Value == app.Slider.Limits(1))
                    up = 1;
                end

                if (up == 1 && app.Slider.Value+0.125 <= app.Slider.Limits(2))
                    app.Slider.Value = app.Slider.Value+0.125;
                elseif (up == 0 && app.Slider.Value-0.125 >= app.Slider.Limits(1))
                    app.Slider.Value = app.Slider.Value-0.125;
                end
                pause(0.01);    % Pause to nicely animate it
            end
        end
```

Figure 20: setAngle and setSpeed function

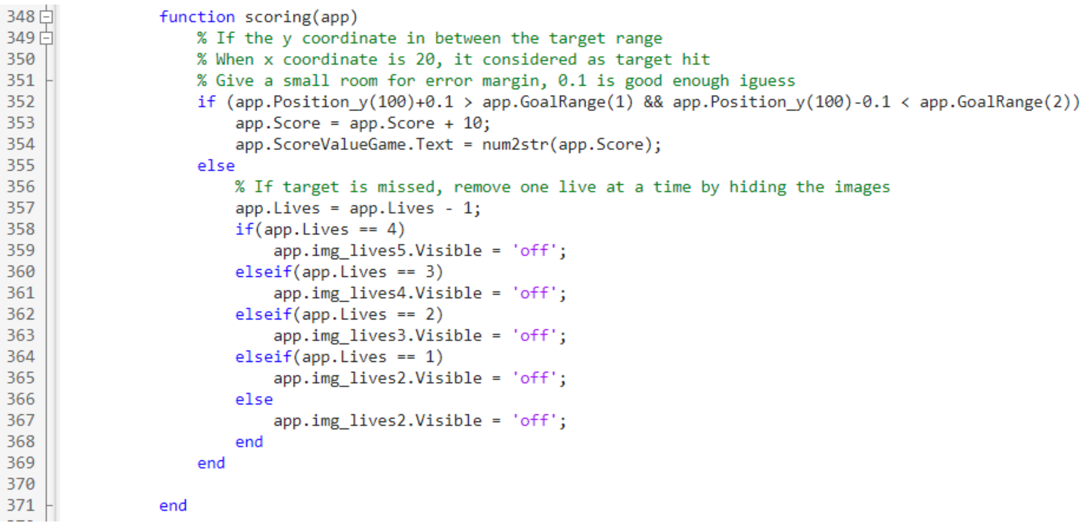
These two functions just move the angle gauge and speed slider up and down until the *Play* button is clicked. There is a local variable named *up*. When it is zero, the value of the angle gauge is increase on every loop by 1 degree and vice-versa when it is one. It pauses for a small moment to animate it. For *setSpeed* function it is similar. Instead of going from 0 to 90, it goes from lower limit of the speed slider, which is ten, to upper limit, which is twenty. And increment and decrement is 0.125 with same amount of pause for the animation.



```
        function calculate(app)
            u = app.Slider.Value;           % Get speed from speed slider
            theta = app.ShootingAngle.Value;% Get angle from angle gauge

            app.Position_x = linspace(0, 20, 100);  % Get 100 points from 0 to 20
            % Calculate y-coords using projectile motion formula, considering wind velocity
            app.Position_y = ((u.*app.Position_x.*sind(theta))./(u.*cosd(theta) + app.Va)) - ((0.5).*app.g.*(app.Position_x.^2))./((u.*cosd(theta)+app.Va).^2);
        end

        function shoot(app)
            app.Instructions.Text = "";              % Hide instruction message during shooting animation
            set(app.PlayButtonGame,'Enable','off'); % Disable Play button during shooting

            % Animate the projectile
            for i = 1:length(app.Position_x)
                plot(app.UIAxes, app.Position_x(i), app.Position_y(i), 'c*');
                app.snowball.Position = [(8+app.Position_x(i)*1793/20) (8+app.Position_y(i)*(952)/10) 25 25];
                pause(0.005)
            end

            set(app.PlayButtonGame,'Enable','on')   % Enable Play button after projectile was shot
        end
```

Figure 21: calculate and shoot function

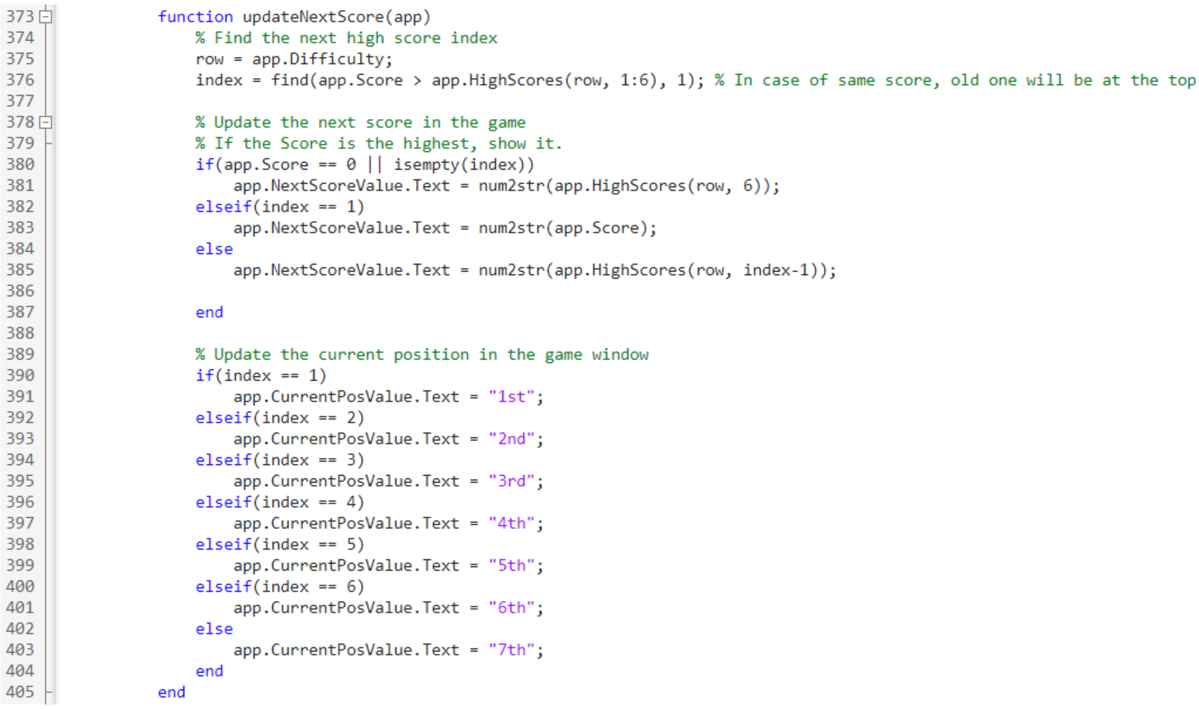
In the first function, the program reads the speed and angle from speed slider and angle gauge respectively. Then calculates y-coordinates using the equation (ii) derived in section 3. *shoot* function first disables the *Play* button to stop user from spam clicking on the button. Then runs a for loop with a small amount of pause in each loop to animate the whole process. In each loop, it plots a new point in the plot seen in figure 8 and moves the snowball image to that point. But before updating the position of the snowball, the graph coordinates are converted to window coordinates.



```
        function scoring(app)
            % If the y coordinate in between the target range
            % When x coordinate is 20, it considered as target hit
            % Give a small room for error margin, 0.1 is good enough iguess
            if (app.Position_y(100)+0.1 > app.GoalRange(1) && app.Position_y(100)-0.1 < app.GoalRange(2))
                app.Score = app.Score + 10;
                app.ScoreValueGame.Text = num2str(app.Score);
            else
                % If target is missed, remove one live at a time by hiding the images
                app.Lives = app.Lives - 1;
                if(app.Lives == 4)
                    app.img_lives5.Visible = 'off';
                elseif(app.Lives == 3)
                    app.img_lives4.Visible = 'off';
                elseif(app.Lives == 2)
                    app.img_lives3.Visible = 'off';
                elseif(app.Lives == 1)
                    app.img_lives2.Visible = 'off';
                else
                    app.img_lives2.Visible = 'off';
                end
            end

        end
```

Figure 22: scoring function

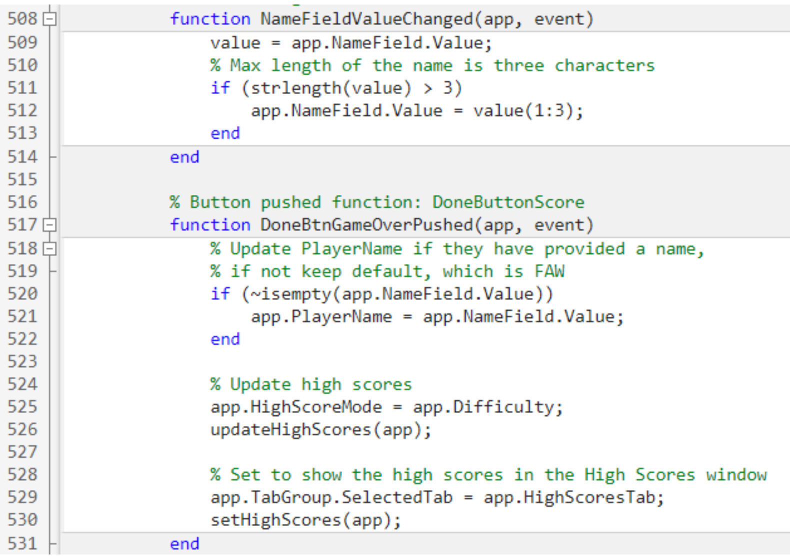
In this function, program checks whether the projectile hit the target. It allows a 0.1-unit error margin. If the projectile hits the target, it increases the score by ten, otherwise removes one life by hiding an image.



```
        function updateNextScore(app)
            % Find the next high score index
            row = app.Difficulty;
            index = find(app.Score > app.HighScores(row, 1:6), 1); % In case of same score, old one will be at the top

            % Update the next score in the game
            % If the Score is the highest, show it.
            if(app.Score == 0 || isempty(index))
                app.NextScoreValue.Text = num2str(app.HighScores(row, 6));
            elseif(index == 1)
                app.NextScoreValue.Text = num2str(app.Score);
            else
                app.NextScoreValue.Text = num2str(app.HighScores(row, index-1));

            end

            % Update the current position in the game window
            if(index == 1)
                app.CurrentPosValue.Text = "1st";
            elseif(index == 2)
                app.CurrentPosValue.Text = "2nd";
            elseif(index == 3)
                app.CurrentPosValue.Text = "3rd";
            elseif(index == 4)
                app.CurrentPosValue.Text = "4th";
            elseif(index == 5)
                app.CurrentPosValue.Text = "5th";
            elseif(index == 6)
                app.CurrentPosValue.Text = "6th";
            else
                app.CurrentPosValue.Text = "7th";
            end
        end
```

Figure 23: updateNextScore function

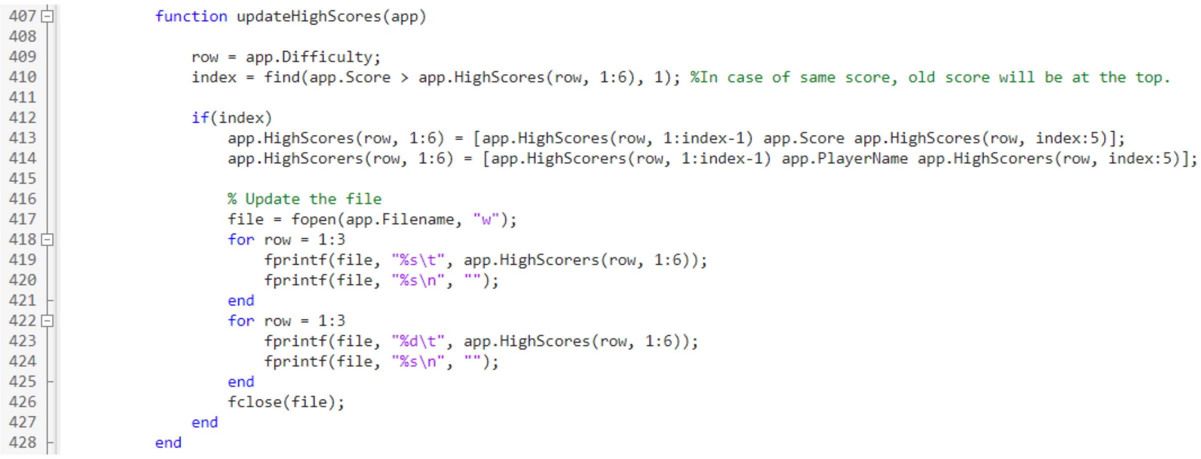
This function goes through the current high scores for the chosen mode and finds the next high score for the user to break. Then shows it in the *Game* window along with the current position among the top scorers.



```
        function NameFieldValueChanged(app, event)
            value = app.NameField.Value;
            % Max length of the name is three characters
            if (strlength(value) > 3)
                app.NameField.Value = value(1:3);
            end
        end

        % Button pushed function: DoneButtonScore
        function DoneBtnGameOverPushed(app, event)
            % Update PlayerName if they have provided a name,
            % if not keep default, which is FAW
            if (~isempty(app.NameField.Value))
                app.PlayerName = app.NameField.Value;
            end

            % Update high scores
            app.HighScoreMode = app.Difficulty;
            updateHighScores(app);

            % Set to show the high scores in the High Scores window
            app.TabGroup.SelectedTab = app.HighScoresTab;
            setHighScores(app);
        end
```

Figure 24: Callbacks of the Game Over window

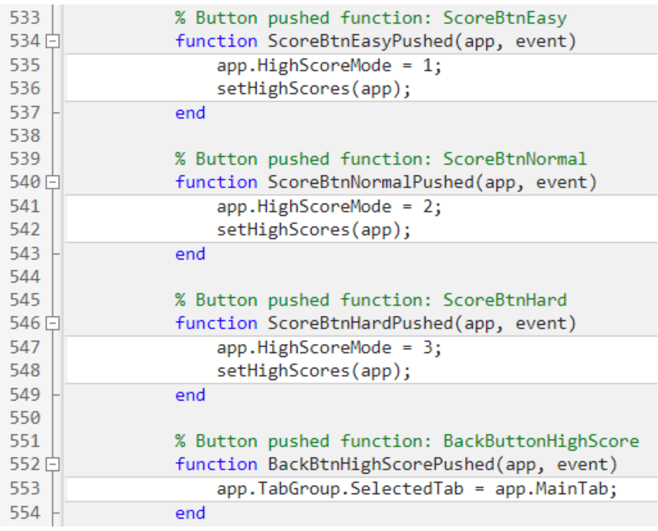
In the game over window, there is an input text field for player name and a *Done* button. When the input text field is updated, the program calls *NameFieldValueChanged* callback and it checks if the length of the name provided in the text field is more than three or not. If it is, then it removes the characters after three. The maximum length of the name is set to three because there is not enough room for larger name in the *High Scores* window shown in figure 10. After providing the name, when the user clicks on the *Done* button, it updates the high scores by calling the *updateHighScores* function. Then it switches to *High Scores* tab and sets the high scores in that window using *setHighScores* functon.



```
        function updateHighScores(app)

            row = app.Difficulty;
            index = find(app.Score > app.HighScores(row, 1:6), 1); %In case of same score, old score will be at the top.

            if(index)
                app.HighScores(row, 1:6) = [app.HighScores(row, 1:index-1) app.Score app.HighScores(row, index:5)];
                app.HighScorers(row, 1:6) = [app.HighScorers(row, 1:index-1) app.PlayerName app.HighScorers(row, index:5)];

                % Update the file
                file = fopen(app.Filename, "w");
                for row = 1:3
                    fprintf(file, "%s\t", app.HighScorers(row, 1:6));
                    fprintf(file, "%s\n", "");
                end
                for row = 1:3
                    fprintf(file, "%d\t", app.HighScores(row, 1:6));
                    fprintf(file, "%s\n", "");
                end
                fclose(file);
            end
        end
```

Figure 25: updateHighScores function

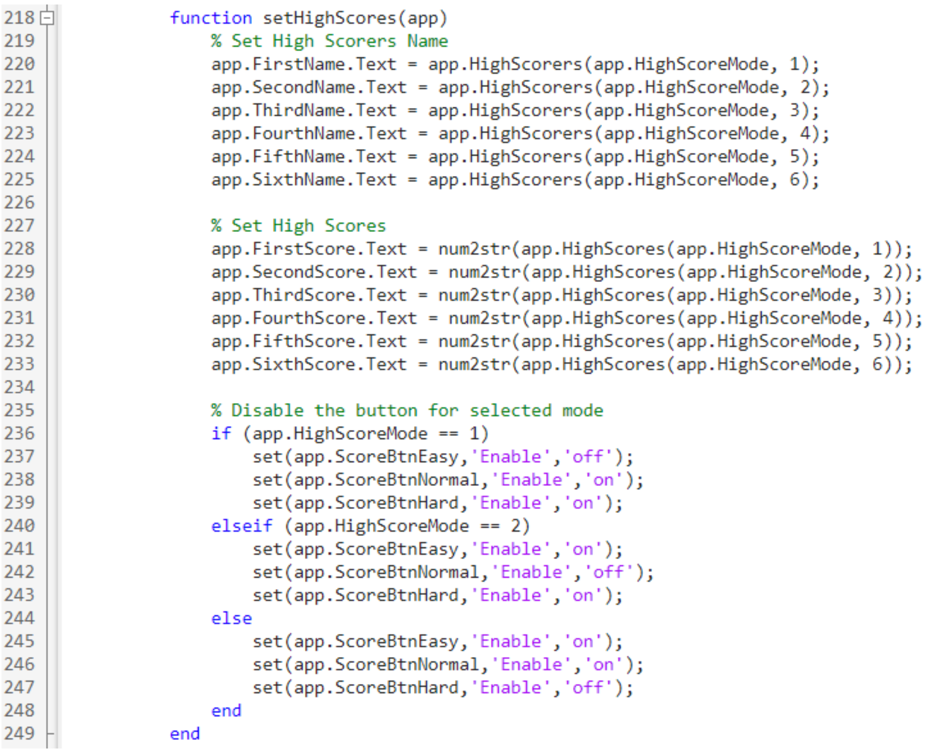
This function goes through the current high scores at the end of each game and checks if the score is higher than any of them. If it is, then it updates the *HighScores* and *HighScorers* array with the score and player name. Once the high scores are updated, the program writes the updated data into the file.



```
        % Button pushed function: ScoreBtnEasy
        function ScoreBtnEasyPushed(app, event)
            app.HighScoreMode = 1;
            setHighScores(app);
        end

        % Button pushed function: ScoreBtnNormal
        function ScoreBtnNormalPushed(app, event)
            app.HighScoreMode = 2;
            setHighScores(app);
        end

        % Button pushed function: ScoreBtnHard
        function ScoreBtnHardPushed(app, event)
            app.HighScoreMode = 3;
            setHighScores(app);
        end

        % Button pushed function: BackButtonHighScore
        function BackBtnHighScorePushed(app, event)
            app.TabGroup.SelectedTab = app.MainTab;
        end
```

Figure 26: Callbacks of the High Scores window

High Scores window has four buttons, six name labels and six score labels. Clicking on any of the *Easy*, *Normal* or *Hard* buttons shown in figure 10 will update the name labels and score labels to show the high scores of that mode by calling the *setHighScores* function. Lastly *Back* button will take the users back to *Main* window.



```
        function setHighScores(app)
            % Set High Scorers Name
            app.FirstName.Text = app.HighScorers(app.HighScoreMode, 1);
            app.SecondName.Text = app.HighScorers(app.HighScoreMode, 2);
            app.ThirdName.Text = app.HighScorers(app.HighScoreMode, 3);
            app.FourthName.Text = app.HighScorers(app.HighScoreMode, 4);
            app.FifthName.Text = app.HighScorers(app.HighScoreMode, 5);
            app.SixthName.Text = app.HighScorers(app.HighScoreMode, 6);

            % Set High Scores
            app.FirstScore.Text = num2str(app.HighScores(app.HighScoreMode, 1));
            app.SecondScore.Text = num2str(app.HighScores(app.HighScoreMode, 2));
            app.ThirdScore.Text = num2str(app.HighScores(app.HighScoreMode, 3));
            app.FourthScore.Text = num2str(app.HighScores(app.HighScoreMode, 4));
            app.FifthScore.Text = num2str(app.HighScores(app.HighScoreMode, 5));
            app.SixthScore.Text = num2str(app.HighScores(app.HighScoreMode, 6));

            % Disable the button for selected mode
            if (app.HighScoreMode == 1)
                set(app.ScoreBtnEasy,'Enable','off');
                set(app.ScoreBtnNormal,'Enable','on');
                set(app.ScoreBtnHard,'Enable','on');
            elseif (app.HighScoreMode == 2)
                set(app.ScoreBtnEasy,'Enable','on');
                set(app.ScoreBtnNormal,'Enable','off');
                set(app.ScoreBtnHard,'Enable','on');
            else
                set(app.ScoreBtnEasy,'Enable','on');
                set(app.ScoreBtnNormal,'Enable','on');
                set(app.ScoreBtnHard,'Enable','off');
            end
        end
```

Figure 27: setHighScores function

This function is pretty basic, it just sets the label text accordingly, then disables the button for selected high scores mode.

6. RESULTS AND ANALYSIS

Main functionality of the program is the projectile motion part. We are using a formula we derived for ourselves.

$$y = \frac{ux\sin\theta}{(u\cos\theta + V_a)} - \frac{1}{2}\frac{gx^2}{(u\cos\theta + V_a)^2}$$

The formula has 6 parameters in it as shown in table 5. g here is fixed, which is 9.8. x-coordinate is an array of numbers in the x axis, for this program it is 100 numbers between 0 and 20. y-coordinate is the output. That leaves only three parameters which work as input to the program. Two of them, initial speed and angle of projection are user input. But the wind velocity is zero for easy and normal mode and for hard mode it is a random value. Now we can analyze results when we change the values of these three parameters.

Table 5: Parameters of the equation

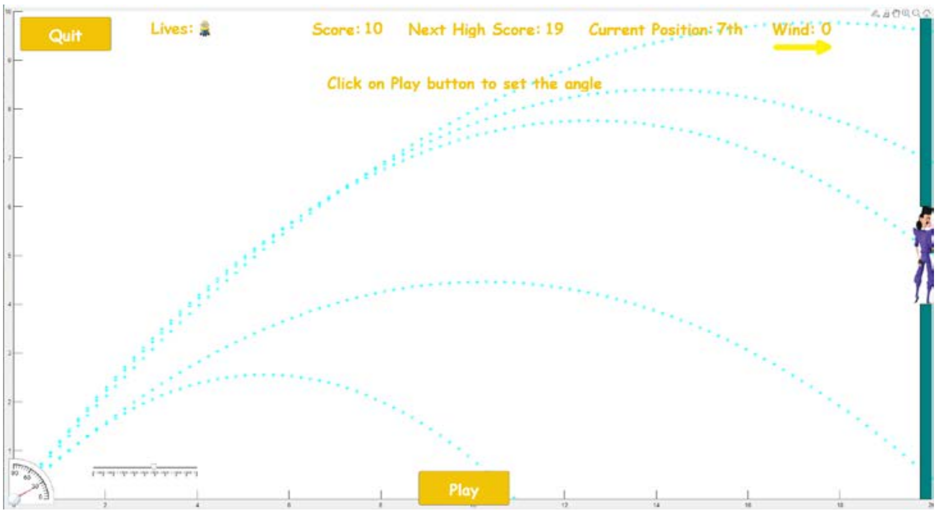
Parameters	Description
x	x-coordinate of the projectile
y	y-coordinate of the projectile
u	Initial speed
θ	Angle of projection
V_a	Wind velocity from left to right
g	Acceleration due to gravity

Among three parameters, we will analyze the initial speed first. While doing that, our angle of projection and wind velocity will be constants. We will analyze it for easy mode where we can understand the whole process better. Value of the x-coordinate will be 20 as we want to know what the height of the projectile is when it reaches the end of the plot. As for the target, it is between $y = 4$ and $y = 6$ for easy mode. We also have to remember that there is a room for error of 0.1 unit when considering if the projectile hit the target. As for angle of projection, we will set it to 45° as it is the angle that produces maximum horizontal range and also it is the angle in the middle.

Table 6: Results for different values of initial speed

u	θ	g	V_a	x	y	**Outcome**
10	45°	9.81	0	20	-19.20	Miss
12					-7.22	Miss
14					0.00	Miss
15					2.58	Miss
15.5					3.68	Miss
15.6					3.89	Miss
15.65					3.99	Hit
15.75					4.20	Hit
16					4.69	Hit
16.5					5.60	Hit
16.75					6.03	Hit
17					6.44	Miss
19					9.14	Miss
20					10.2	Miss

As we can see in table 6, when the angle is 45°, the initial velocity can be between 15.65 and 16.75 to hit the target and score. For all the other values outside of this range will cause the projectile to miss the target. In figure 28, we can see that when the speed is too low, it never reaches the end of the graph. On the other hand, if the speed is too high, then it goes way above the target. There is a small sweet spot that can make the projectile hit the target, which is 15.65 to 16.75 for the projection angle 45°.



Lives:

Score: 10

Next High Score: 19

Current Position: 7th

Wind: 0

Click on Play button to set the angle

Figure 28: Results for different values of initial speed

Next parameter we can analyze is the projection angle θ. Just as before, our output is y-coordinate of the projectile. Angle of projection will be the variable here and the initial speed will be fixed. We have chosen the speed somewhere close the middle, which will give us a general idea. Since we are analyzing it in the easy mode, the wind velocity will be zero. The target is between $y = 4$ and $y = 6$ and it includes 0.1 unit margin error when checking whether the projectile hit the target or missed it.

Table 7: Results for different values of angle of projection

θ	u	g	V_a	x	y	**Outcome**
$10°$	17	9.81	0	20	-3.47	Miss
$20°$					-0.40	Miss
$30°$					2.50	Miss
$35°$					3.90	Hit
$40°$					5.22	Hit
$43°$					5.97	Hit
$45°$					6.43	Miss
$50°$					7.42	Miss
$60°$					7.51	Miss
$70°$					-3.03	Miss
$80°$					-111.49	Miss
$90°$					∞	Miss

As we can see in table 7, when the speed is 17 unit, the projectile can hit the target for only certain values of the angle of projection. That range is 35° to 43°. For all the other values, it either hits way above the target, or goes straight downhill after a small period of time. Sometimes, it cannot even reach the end of the plot. In figure 29, we can see that the projectile hits the target within a specific range only. If the angle is too high, it either goes above the target or goes way up and then suddenly goes downhill. On the other hand, if its too low, it goes downhill straight.

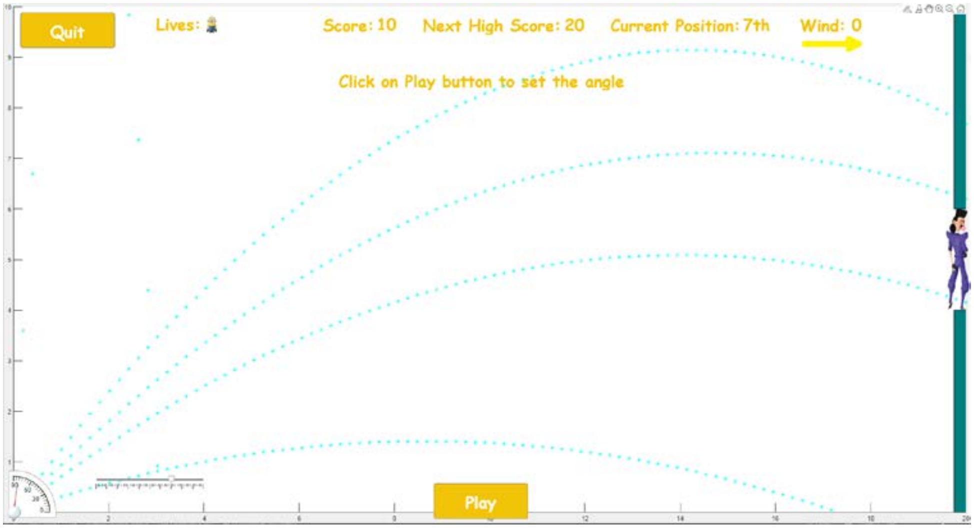


Figure 29: Results for different values of angle of projection

Our final parameter to be analyzed is the wind velocity, which is only present in hard mode. For the sake of analysis, we will consider the wind in easy mode so we can understand what is happening. Because in hard mode, it would be hard to analyze since the target is changing position in every game. When we are analyzing one parameter, we have to set all the other parameters to constants. We have chosen the projection angle to be 45° and initial speed to be 16. We will analyze how the wind velocity affects the results. In the game, wind velocity is either zero, or a random value in between 0.1 and 0.25.

Table 8: Results for different values of wind velocity

$\boldsymbol{\theta}$	$\boldsymbol{u}$	$\boldsymbol{\theta}$	$\boldsymbol{g}$	$\boldsymbol{x}$	$\boldsymbol{y}$	**Outcome**
0.00	16	45°	9.81	20	4.69	Miss
0.10					4.78	Miss
0.12					4.80	Miss
0.14					4.82	Hit
0.16					4.83	Hit
0.18					4.85	Hit
0.20					4.87	Miss
0.22					4.88	Miss
0.24					4.90	Miss
0.25					4.91	Miss

As we can see in table 8, the y coordinates of the projectile increases little by little when the wind velocity increases. This happens because the wind is flowing from left to right, which increases the horizontal velocity. So it hits the target faster, hitting it at an early stage. But the numbers would have gone down if it hit the target while coming down from the highest point in its trajectory.

7. DISCUSSION

Creating a game using MATLAB was a challenging assignment. Along the way I had to face some issues. For example, when I was creating the plot for the game, I needed a fixed width and height. I had to go through a long trial and error process to come with that width and height for the plot. Another issue was putting image of the snowball on the window, for which I had to convert from graph coordinates to window coordinates. It was also a very long process of trial and error. But I could have avoided this whole image on graph situation, if I could have put the background image on the plot itself, rather than putting it in front of the plot. Putting the image on the from does make the game look nicer but other way around would have been a very simple process.

When working on the high scores window, I encountered two challenging aspects. One of them was the length of the top scorers' names. The space for the name is limited, so if the user provides a lengthy name, part of their name will be covered up, which is unattractive. Initially, I considered providing input instructions to restrict the user's name to a certain number of characters. Later, I resolved this issue by limiting the maximum length of a user's name to three characters in the input box. Also, if they do not provide any name, default name will be taken. The other issue was to read from file. The method I used here to read from file reads the whole text into a string, which I had to convert into multiple 2D matrices.

There are some other challenges that I faced. Such as, animating the projectile, which I fixed using a for loop and pause functionality; holding some of the plots while remove the other, assigning plots to a variable fixed this issue easily. At the beginning I wanted to create a multi-windowed app. But that seemed a very complicated process. Easiest option was to use multi-tab feature and put the top part of the tab which lets user switch between tabs outside of the window. Therefore, the tabs can be switched only from the code, not from UI.

8. SPECIAL FEATURES

One of the requirements of the assignment is to implement at least one innovative idea into the program. For this program, I managed to implement a few. For example: Score system and difficulty. Instead of just seeing the plot of the projectile, it sets a target for the user to hit and gives reward or punishment based on whether the user hit the target or missed it. The user gets 10 points rewards for hitting and loses one life for missing it. It also keeps track of the high scores in the game while showing the next high score and current position to the user during gameplay. For different difficulties, the target position changes and there is random wind for hard mode.

For the calculation of the trajectory of the projectile, the program needs two inputs from the user, the initial speed and the angle it will be thrown at. Instead of taking these inputs as numeric values from the user, the program makes it a bit more interactive and challenging. It constantly moves the angle gauge and the speed slider up and down for the user to choose. User clicks on the specified button to set the angle and the speed of the projectile.

Another one is saving the high scores into a file. When the user closes the program, it will save all the high scores into the file. Next time when the user opens the file, it will read all of it again from the file and show it to the user. Therefore, progress is never lost unless the user deletes the high scores file.

To make the program more appealing, background images were added to every window. But the *Game* window shown in figure 7 has a plot which gets hidden behind the background image. Because of that, the actual game that is happening on the plot cannot be seen. Actual game here uses the coordinates on the plot but the snowball and the target image use coordinates on the window. The coordinates needed to be converted from plot to window for the snowball and target to follow the projectile and goal bars on the plot respectively.

9. LIMITATIONS

It does not matter, how good a program is, there will always be some limitations. There are some limitations I could not overcome due to lack of time and planning. For example: The program keeps the high scores text file saved after each game. That file is not encrypted, meaning that not only everyone can see the file, but they can also manipulate it and change the high scores. If someone manipulates the file, and they do a bad job at manipulating, then there is big chance of program crashing when it tries to read the data from the file.

MATLAB is a single threaded program, meaning it can execute only one command at a time. If there are multiple commands to execute, it will execute the first command. Once it successfully executes the command, it moves to the next once and so on. Due to this reason, wind velocity was shown as a text in the game (figure 7) instead of animating. If I was to animate the wind, the program would need to run certain commands in a loop while doing the other task. As I have mentioned before, running two tasks at once is not possible to accomplish, even if it is possible, it is beyond my area of expertise.

10. CONCLUSION

Objective of this assignment was to create a MATLAB GUI app using App Designer. I managed to create a game with projectile motion, which should meet all the requirements. The game has a lot of special features but there are infinite number of improvements that can be done. One of the improvements I would love to see in it would be the game sound. There is a huge room for improvements in the wind velocity. Right now, it does not affect the results that much, where it should be. Animation and sound could be played when the projectile hits the target. Most importantly, code of the game could be more optimized. As of now, every time the game is running, it is piling one process on top of another. This takes a huge hardware resource. The code could be optimized so that when it goes into a different window, it cancels all the process running on the old window. Nevertheless, it is a moderate program and fulfills the requirements. It was fun working on it.

11. REFERENCE

Computer Hope. (2021). *GUI.* Retrieved from: https://www.computerhope.com/jargon/g/gui.htm [Accessed date: 28 August 2022]

Lithmee. (2018). *Difference Between GUI and CLI.* Retrieved from: https://pediaa.com/difference-between-gui-and-cli/ [Accessed date: 28 August 2022]

MathWorks. (n.d.). *Callbacks – Programmed Response to User Action.* Retrieved from: https://www.mathworks.com/help/matlab/creating_plots/callbacks-programmed-response-to-user-action.html [Accessed date: 28 August 2022]

MathWorks. (n.d.). *MATLAB App Designer*. Retrieved from: https://www.mathworks.com/products/matlab/app-designer.html [Accessed date: 28 August 2022]

MathWorks. (n.d.). *Share Data Within App Designer Apps.* Retrieved from: https://www.mathworks.com/help/matlab/creating_guis/share-data-across-callbacks-in-app-designer.html [Accessed date: 28 August 2022]

MathWorks. (n.d.). *What is MATLAB?* Retrieved from: https://www.mathworks.com/discovery/what-is-matlab.html [Accessed date: 28 August 2022]

Turito Team. (2022). *Projectile Motion: Definition, Equations, & FAQs.* Retrieved from: https://www.turito.com/blog/physics/projectile-motion [Accessed date: 28 August 2022]

10. APPENDIX

Despicable Me & Minion movies: https://www.imdb.com/list/ls068935616/