



# Python Programming and Its Engineering Applications

## 大作业报告

学 号 2024280038

姓 名 Jason Rich Darmawan

日 期 2025-02-12

指导老师 李滔

西北工业大学

# 目 录

<b>1</b>	<b><i>The Red Fighter Jet with Bot Enemy</i></b> .....	<b>2</b>
1.1	Project Requirement.....	2
1.2	Diagram of the Code Structure .....	2
1.3	Results .....	3
1.4	Analysis .....	4
<b>2</b>	<b><i>The Media Player with Video Support</i></b> .....	<b>6</b>
2.1	Project Requirement.....	6
2.2	Diagram of the Code Structure .....	6
2.3	Results .....	7
2.4	Analysis .....	8
<b>3</b>	<b><i>The Diary App with Multiline Entries Support</i></b> .....	<b>9</b>
3.1	Project Requirement.....	9
3.2	Diagram of the Code Structure .....	9
3.3	Results .....	10
3.4	Analysis .....	10
<b>4</b>	<b><i>The Gauge Widget</i></b> .....	<b>12</b>
4.1	Project Requirement.....	12
4.2	Diagram of the Code Structure .....	12
4.3	Results .....	13
4.4	Analysis .....	13

# 1 The Red Fighter Jet with Bot Enemy

Code: [GitHub](#)

## 1.1 Project Requirement

The project was developed using Python version 3.10.14 and Pygame version 2.6.1. The project should work with the specified versions or any compatible releases.

## 1.2 Diagram of the Code Structure

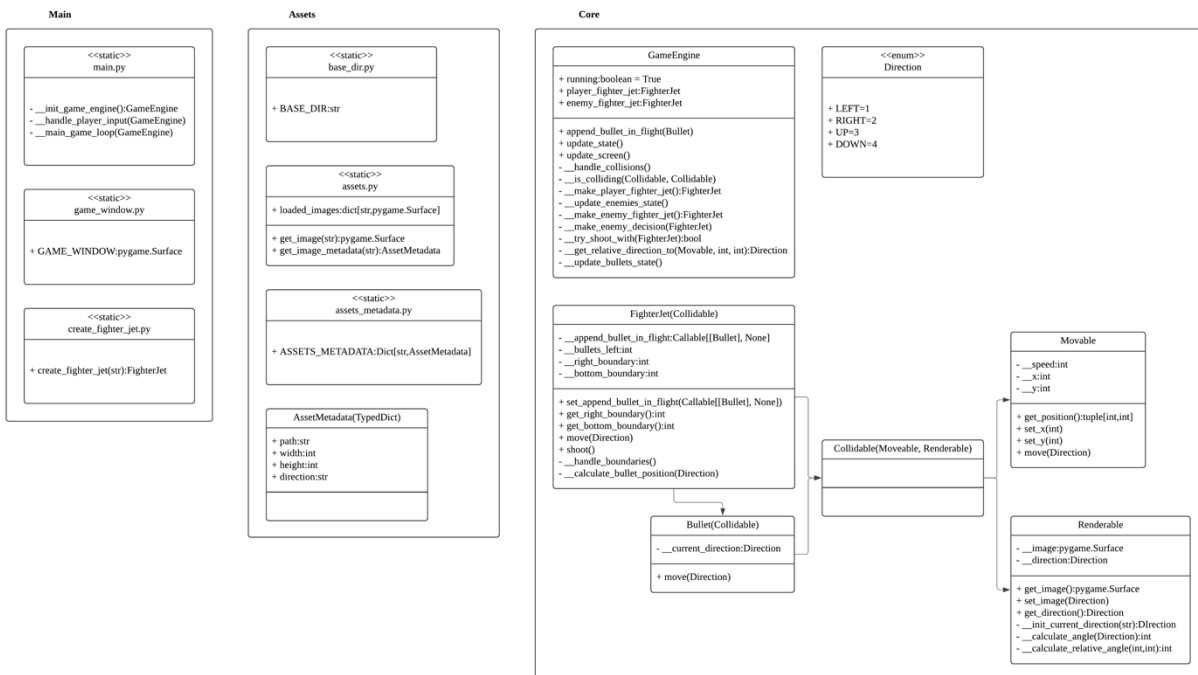


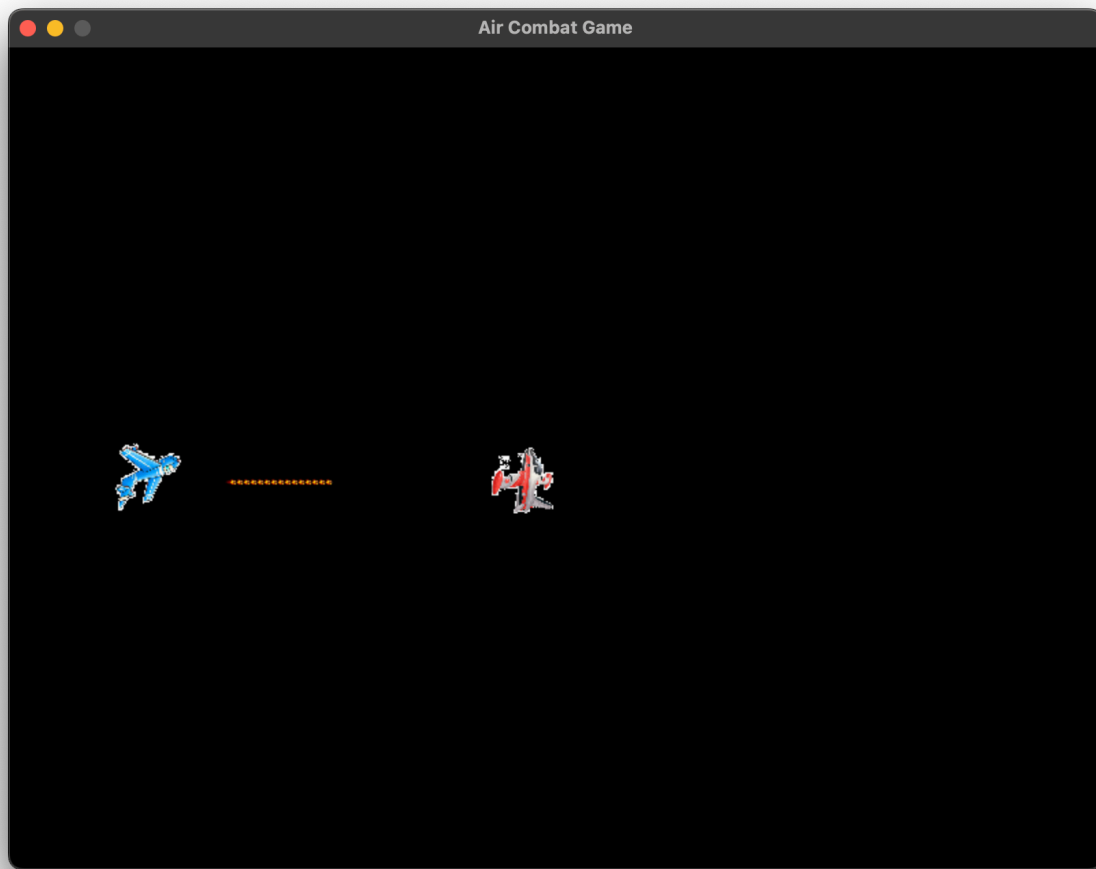
Figure 1 The Red Fighter Jet Code Structure

Figure 1 shows that the project consists of 3 key components: 1) The main program to run the Pygame and handle the keyboard input. 2) The assets program to load and cache images. 3) The core program to run the GameEngine.

The GameEngine class has two responsibilities: 1) To update the game's state and window. The game's state includes fighter jets, bullets positions, and collisions handling between fighter jets or a bullet. 2) To control the blue fighter jet. The controls include moving the blue fighter jet to target and shoot the player's fighter jet.

The Collidable class is used by the GameEngine object to detect collisions between any objects that inherit the Collidable class. Both the FighterJet class and the Bullet class inherit the Collidable class. Therefore, the following cases are considered as collisions: 1) a FighterJet object collides with a Bullet object. 2) a FighterJet collides with a FighterJet object.

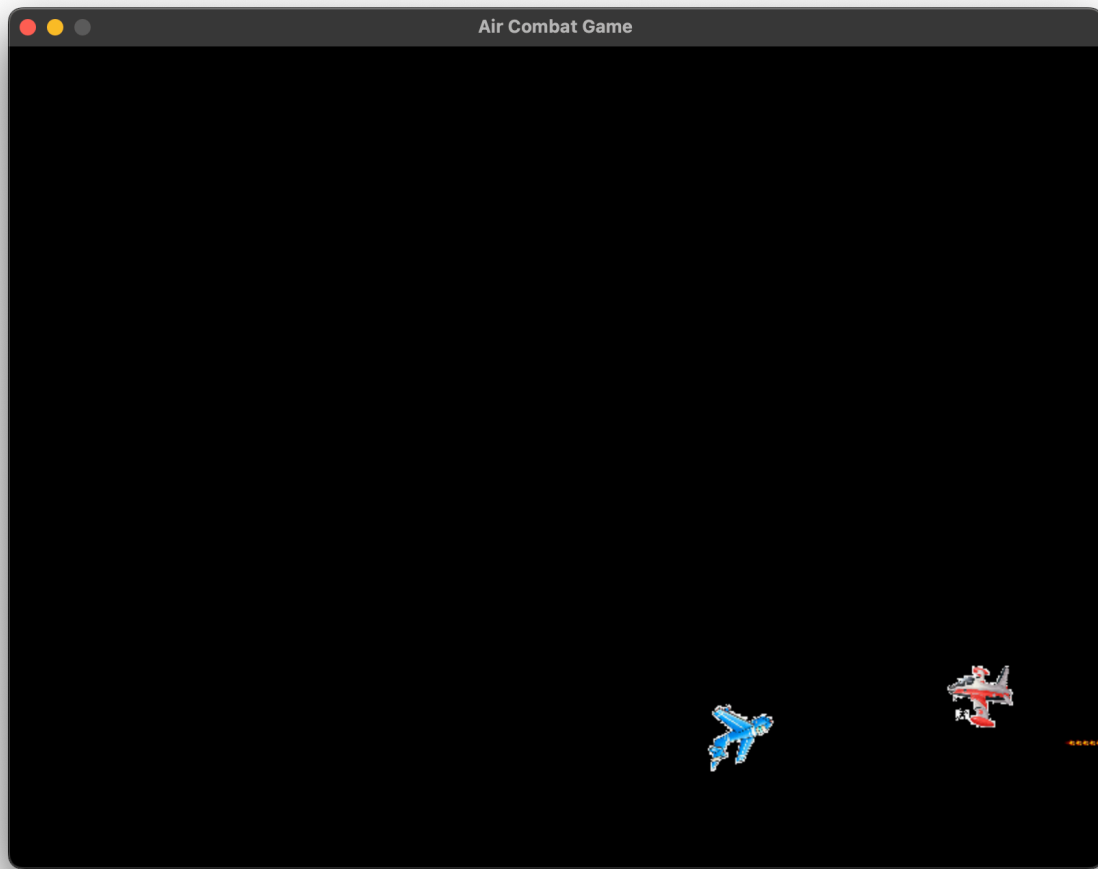
### 1.3 Results



*Figure 2 Blue Fighter Jet Shooting from Left Side*

The Movable class as shown in Figure 1 enables the fighter jet to move to the left, to go up, to the right, and to go down. Therefore, the following cases are possible: 1) The blue or the player's fighter jet shooting from left/up/right/down side. 2) The player and the blue fighter jets engage in a dog fight—shoot 1 bullet and run away hoping the bullet will hit the enemy fighter jet.

## 1.4 Analysis



*Figure 3 The Blue Fighter Jet Locks onto the Player's Fighter Jet*

Once the blue fighter jet locks onto the player's fighter jet, it will be hard to avoid a dog fight. This is because the `__try_shoot_with(FighterJet):bool` function of the `GameEngine` class as shown in Figure 1 will move the blue fighter jet towards the player's fighter jet and shoot when the player's fighter jet is in the shooting range. Therefore, the best strategy is to shoot 1 bullet and run away hoping the bullet will hit the enemy fighter jet.

However, the blue fighter jet will not change direction to go up and move to the right before shooting the player's fighter jet as shown in Figure 3. This is because the `__try_shoot_with(FighterJet):bool` function of the `GameEngine` class does not calculate the direction with the shortest path to the player's fighter jet. As a result, the blue fighter jet chosen paths sometimes can be inefficient.

This suggests an opportunity for further development to implement logic to determine which direction is the shortest path to the player's fighter jet. This is because the `try_shoot_with(FighterJet):bool` function of the GameEngine class only tasks are: 1) To change the blue fighter jet's direction towards the player's fighter jet. 2) To shoot when the player's fighter jet is in the shooting range.

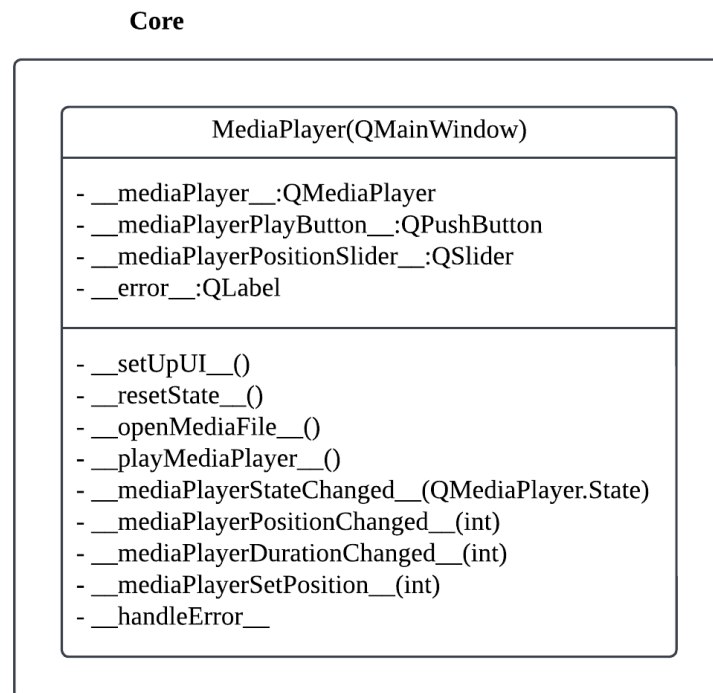
## 2 The Media Player with Video Support

Code: [GitHub](#)

### 2.1 Project Requirement

The project was developed using Python version 3.10.14 and PyQt5 version 5.15.11. The project should work with the specified versions or any compatible releases.

### 2.2 Diagram of the Code Structure



*Figure 4 The Media Player Code Structure*

Figure 4 illustrates four of six key components of the project: 1) The media player, which plays the audio or video. 2) The play/pause button on the media player to start or stop the audio or video. 3) The position slider on the media player, allowing users to quickly skip forward or backward through the audio or video. 4) The error label, which displays an error message if the media player fails to load an audio or video file.

Two key components are not shown in the figure due to a design choice—there is no need to programmatically adjust the component for this project. First, the volume control, which allows users to adjust the audio or video volume. Second, the video widget, which displays the video for the user.

## 2.3 Results



*Figure 5 The Media Player Playing a Video with Audio*

The Media Player playing a video with audio. The `QMediaPlayer` class from the PyQt5 package is responsible for loading, playing the audio and sending the video frame to the `QVideoWidget` class. The `QVideoWidget` class from the PyQt5 package is responsible for displaying the video frames from the `QMediaPlayer` class, as shown in Figure 5.



## 2.4 Analysis

The `mediaPlayerPositionSlider` variable, shown in Figure 4, allows users to quickly skip forward or backward through the audio or video. As the user moves the slider, the variable adjusts the playback position, and the media player immediately output the video frame to the `QVideoWidget` class, as seen in Figure 5, even if the media player is paused.

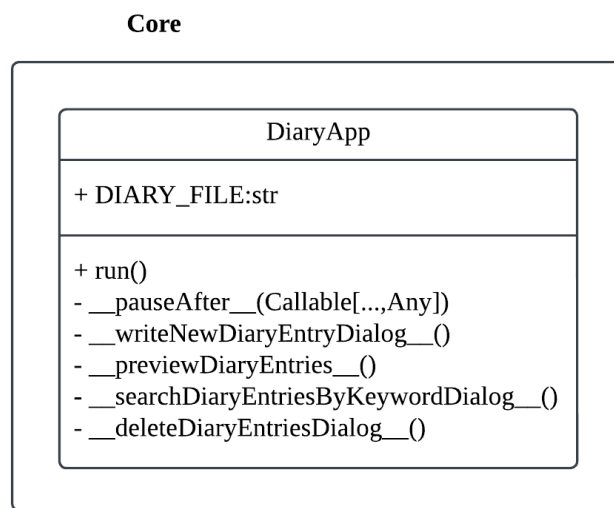
### 3 The Diary App with Multiline Entries Support

Code: [GitHub](#)

#### 3.1 Project Requirement

The project was developed using Python version 3.10.14. The project should work with the specified versions or any compatible releases.

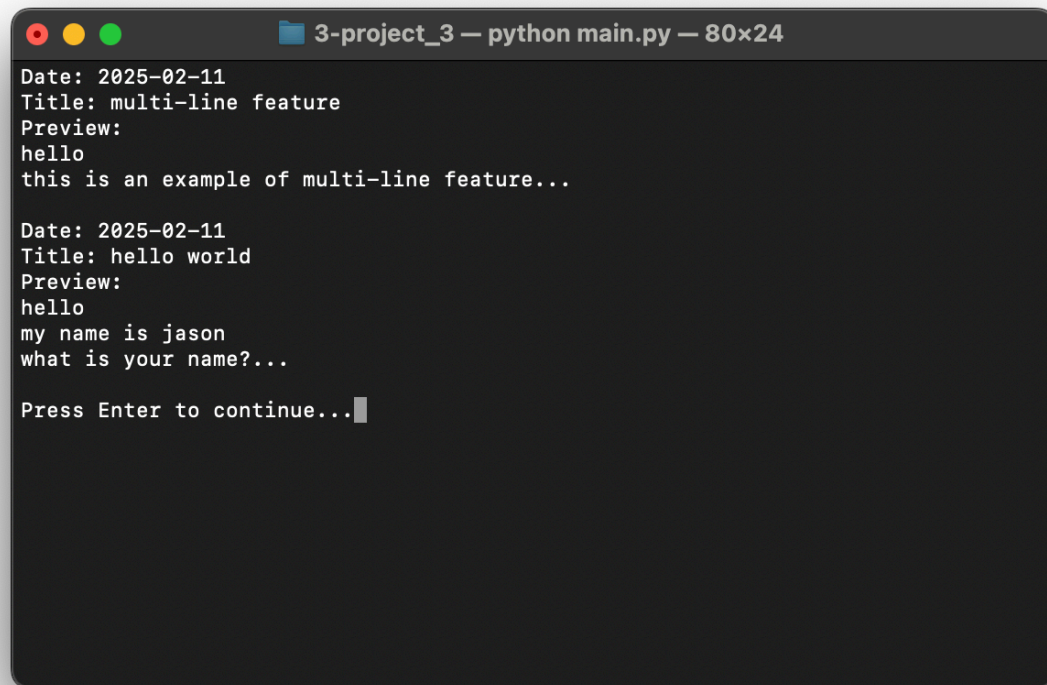
#### 3.2 Diagram of the Code Structure



*Figure 6 The Diary App Code Structure*

Figure 6 shows that the four main features of the project. These includes: 1) A dialog for writing a new diary entry. 2) A function to preview all diary entries. 3) A dialog for searching diary entries by keyword. 4) A dialog to delete diary entries by date and title. In addition to these main features, there are two other important functions: 1) The `run()` function is responsible for programmatically starting the app. 2) The `__pauseAfter__(Callable[...,Any])` function is a Python decoration function that is invoked before any of the five main features. This function clears the screen, executes the selected feature, and then pauses the app, prompting the user to press Enter to continue. This approach improves the user experience by ensuring that only the relevant information is visible, as the previous content is cleared before displaying the next menu.

### 3.3 Results



```
3-project_3 — python main.py — 80x24
Date: 2025-02-11
Title: multi-line feature
Preview:
hello
this is an example of multi-line feature...

Date: 2025-02-11
Title: hello world
Preview:
hello
my name is jason
what is your name?...

Press Enter to continue...|
```

*Figure 7 The Diary App with Multiline Entries Support Previewing Entries*

Figure 7 demonstrates the multiline entries support of the Diary App. This is accomplished by saving literal `\n` characters in the `DIARY_FILE.txt` file. When the app processes the file, these newline characters are loaded as `\\n`. Consequently, before displaying the content to the user, the app must remove the extra backslash to ensure the text appears with the proper line breaks.

### 3.4 Analysis

The Diary App does not currently support hotkeys, such as using the `Control+s` hotkey to save a diary entry or the `Control+w` hotkey to close the window or discard a new diary entry. This limitation arises from the app's reliance on the built-in `input()` function in Python, which does not recognize key combinations or modifier keys like the `Control` key. As a result, it becomes difficult to trigger other functions that can listen for hotkeys. However, this issue could be addressed using Python's threading capabilities.

This presents an opportunity for further development by implementing logic to support hotkeys, similar to other note-taking apps like Microsoft Word, Notepad, and TextEdit, which use hotkeys to save the file. Adding this feature would enhance the user experience in the Diary App.

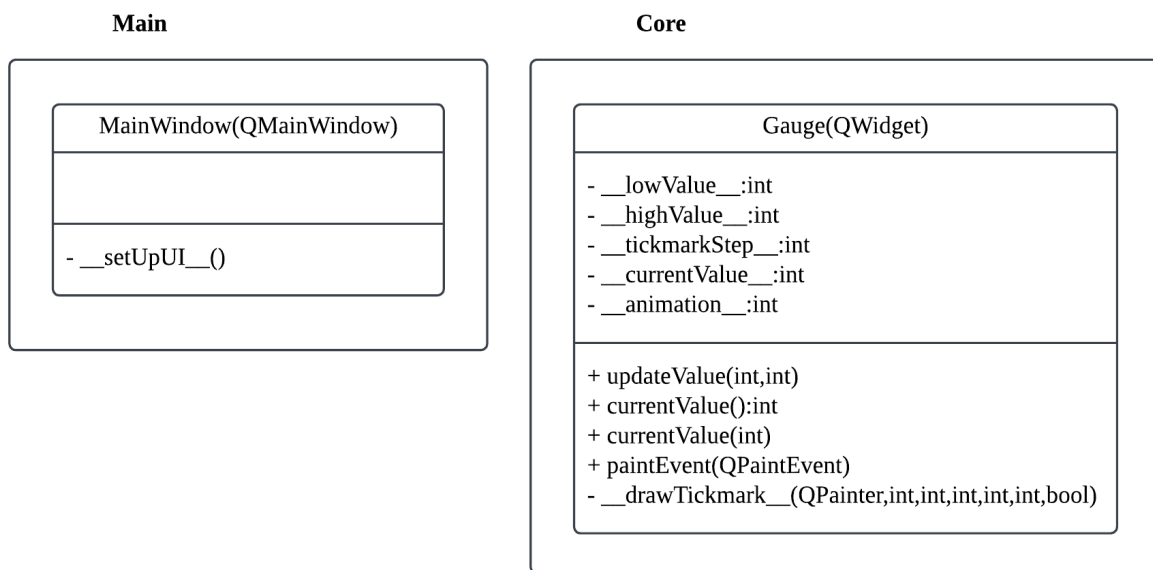
## 4 The Gauge Widget

Code: [GitHub](#)

### 4.1 Project Requirement

The project was developed using Python version 3.10.14 and PyQt5 version 5.15.11. The project should work with the specified versions or any compatible releases.

### 4.2 Diagram of the Code Structure

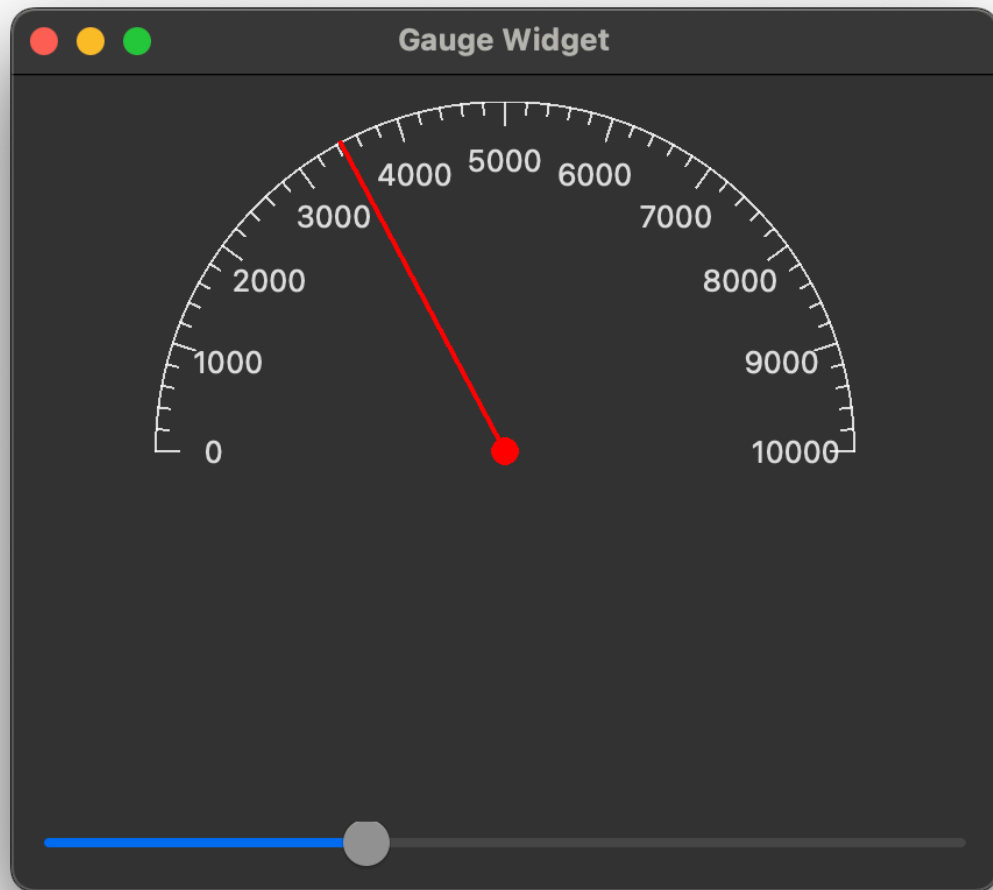


*Figure 8 The Gauge Widget Code Structure*

Figure 8 shows that the project is made up of two components: 1) The main program, which handles setting up the program window. 2) The core program, which is responsible for animating the gauge.

The **Gauge** class has two main responsibilities: 1) Drawing the tick marks. 2) Animating the needle. The needle's movement is controlled by the `updateValue(int,int)` function, where the second parameter determines the speed of the needle in milliseconds. This means the needle will move by the value of the second parameter for each millisecond, effectively controlling how fast the needle moves.

## 4.3 Results



*Figure 9 Gauge Widget with Control Slider*

Figure 9 showcases the `Gauge` widget. Below the `Gauge` widget, there is a `control` slider that allows the user to move the needle by sliding it. The core idea behind drawing the needle involves using the cosine and sine of an angle, which are then multiplied by the length of the needle. This calculation determines the  $(x_1, x_2)$  coordinates of the needle's tip, allowing it to be properly positioned on the gauge.

## 4.4 Analysis

The current version of the Gauge Widget has a fixed span angle of 180 degrees, meaning it does not currently support dynamically adjust the angle of the gauge.

However, certain gauges—such as those found on car dashboards—often use a span angle of 270 degrees or other custom values. The main focus of this project is to illustrate the core concept of how the gauge animation works. If the ability to support different span angle is needed, we can modify the `paintEvent(QPaintEvent)` function to make the span angle adjustable.