Yeahlowflicker Production

PRJ001 - Productivity Clock

Table of Contents

Project ID: PRJ001
Project Name: Productivity Clock

0 | Definitions and References

Qt:	A widget toolkit for creating graphical user interfaces
QML:	A user-interface markup language developed by Qt
PySide2:	A Python library that implements Qt

HH:MM:SS: A display format of time in 24-hour format (e.g. 23:15:30)

Variables and functions will be set to italic style for better understanding.

1 | Introduction

Time-blocking is an important strategy to boost productivity. The idea is to divide time into multiple blocks for one to complete multiple tasks in a day.

The project is to build a simple countdown clock that loops continuously to notify people of the time remaining in each time block.

There are many other existing countdown clocks available online. However, they usually provide very basic countdown functions, which I find insufficient for myself. Therefore, the project is launched to build my very own productivity clock.

2 | Tools

The project is completed using the following tools and libraries:

- Python 3.8
- PyCharm Community
- Qt for Python (Pyside2)
- PyInstaller

PyCharm is the Python IDE used to develop the application. The Pyside2 library provides Qt GUI implementation, while PyInstaller handles the application export.

3 | Code Structure

The application contains the following scripts:

- bootstrap.py
- main.py
- main.qml

The bootstrap.py script is a bootstrap program to launch the application. It defines the Qt application and application engine, as well as loading the main.qml file into the application engine before the window shows up.

The main.py script is a management script that holds functions necessary for the system. It does not process the interface, but executes the timer and emit signals of the calculated time.

The main.qml script is the main reference file of the user-interface. Being a QML script, it defines the layout and the GUI objects visible in the main window. As QML supports JavaScript functions, this script is used for signal handling that takes signal values and applies to GUI elements as well.

4 | Logic

The application is full of signal connections between Python and QML.

When the Start button is clicked, a signal with a HH:MM:SS string will be emitted by the QML script to tell the Python script to calculate the *period*. The python script will then start the QTimer. The QTimer has an interval of 1000msec (1 second). Whenever *timeout()* is emitted, the *elapsed* value will increment by 1.

When the *elapsed* value is smaller than the *period*, it means the countdown is done. This will automatically reset the *elapsed* value to 0 and the system will countdown from the *period* again. This is the automatic looping of the system.

Once the Stop button is clicked, the elapsed value will be set to 0 again, and the Python script will emit a signal with the period to tell the QML script to set the display label to the original period (in HH:MM:SS format).

The conversion between HH:MM:SS and seconds are done using the following algorithms:

HH:MM:SS to Seconds:

Value = Hours * 3600 + Minutes * 60 + Seconds

Seconds to HH:MM:SS:

Value
=
(Input / 3600) + “:” + (Input - (Hours * 3600)) / 60 + “:”
+ (Input - (Hours * 3600) - (Minutes * 60))

5 | User Interface

The user interface is a simple dark theme interface. It contains two elements only - the main time display/edit area and the Start/Stop button.

The time label is a QML TextField in which users can modify the text. When the countdown is running, the field will be set to read-only and its text will be updated every second to display the remaining time. Once the countdown is stopped, the field becomes non-read-only and the user can modify the countdown period.

The Start/Stop button works as a toggle. When clicked, the *counting* boolean variable is changed to the opposite state. If *counting* is true, the button's text is "Stop", and otherwise "Start".

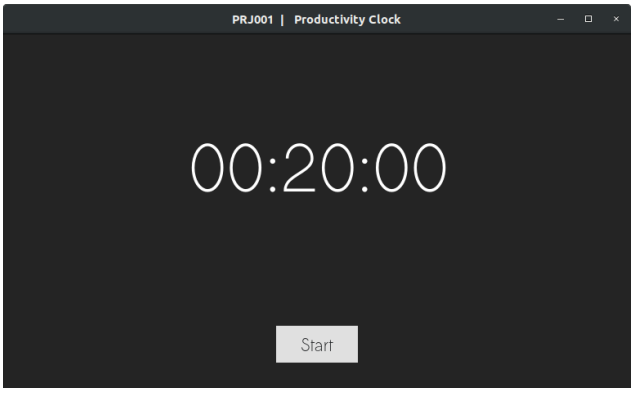

Figure 5a: Screenshot of user interface when not counting

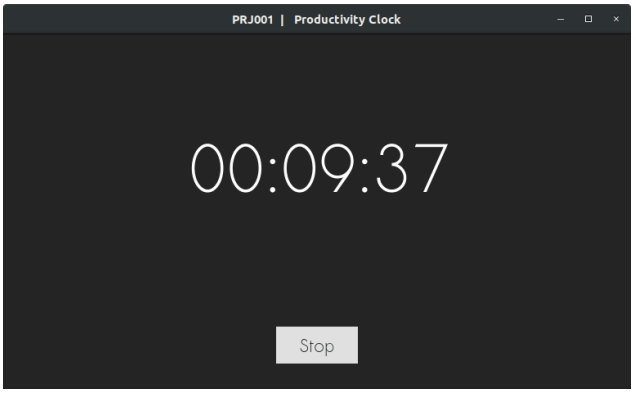

Figure 5b: Screenshot of user interface when counting

6 | Conclusion

This project is very useful to me. I have been looking for a countdown timer clock for a long time and I still find the existing ones not meeting my demands. By working on a custom timer clock, I can have the user-interface extremely simple and work well.

I appreciate the automatic looping feature the most, as I do not have to manually reset and restart the timer.

Working on this project is not difficult, as the structure and user-interface are small and simple. However, the signal connection is quite tricky for me to get data correctly transferred. What I learned from this project is that it is useful to reuse GUI elements. By implementing reuse algorithms, I do not have to manipulate a large number of GUI objects, so the signal connection is much easier, as I only have to define a few variables to check the current states.

Possible features for improvements of the project

- Queue system for task display
- Pause/Resume feature
- Analysis system