Python 个人设计报告

选题:B1(时钟)

目录

一、引言

1.1 项目简介

本项目旨在创建一个简单而实用的桌面应用程序，用于显示当前时间。这个时钟应用程序将具备以下主要特点：

1. 用户友好的界面： 使用 Tkinter 库，我们将构建一个直观的用户界面，将当前北京时间以表盘的形式绘制在界面上，使用户能够轻松查看当前时间。
2. 24/12 小时显示制度切换： 用户将能够通过点击按钮的形式实现对时间显示格式中的 24 小时显示制度和 12 小时显示制度的自由切换
3. 自定义闹钟： 用户将能够通过输入自定义设计自己的闹钟，闹钟时间到时会弹出窗口提醒用户。
4. 多时区支持： 我们将实现多时区支持，允许用户同时查看多个地区的时间。
5. 定时器支持：我们将实现一个简单的定时器，运行用于通过输入时间来重置定时器，并在倒计时过程中自由的开始或停止倒计时。
6. 整点报时功能：我们的时钟还将提供整点报时功能，在整点时会弹出窗口提醒用户。

1.2 设计目标

本项目的设计目标是通过 python 编程语言和 python 中的 tkinter 等库，实现一个用户友好的时钟程序，其直接在设计时就进行封装，并将其开源到 GitHub 供用户自由下载使用。

二、技术栈

2.1 使用的编程语言

本项目主要使用的编程语言是 Python，其中 Python 版本为 3.9.18，主要用到的库是 python 内置的 tkinter 库，math 库，datetime 库

2.2 开发工具

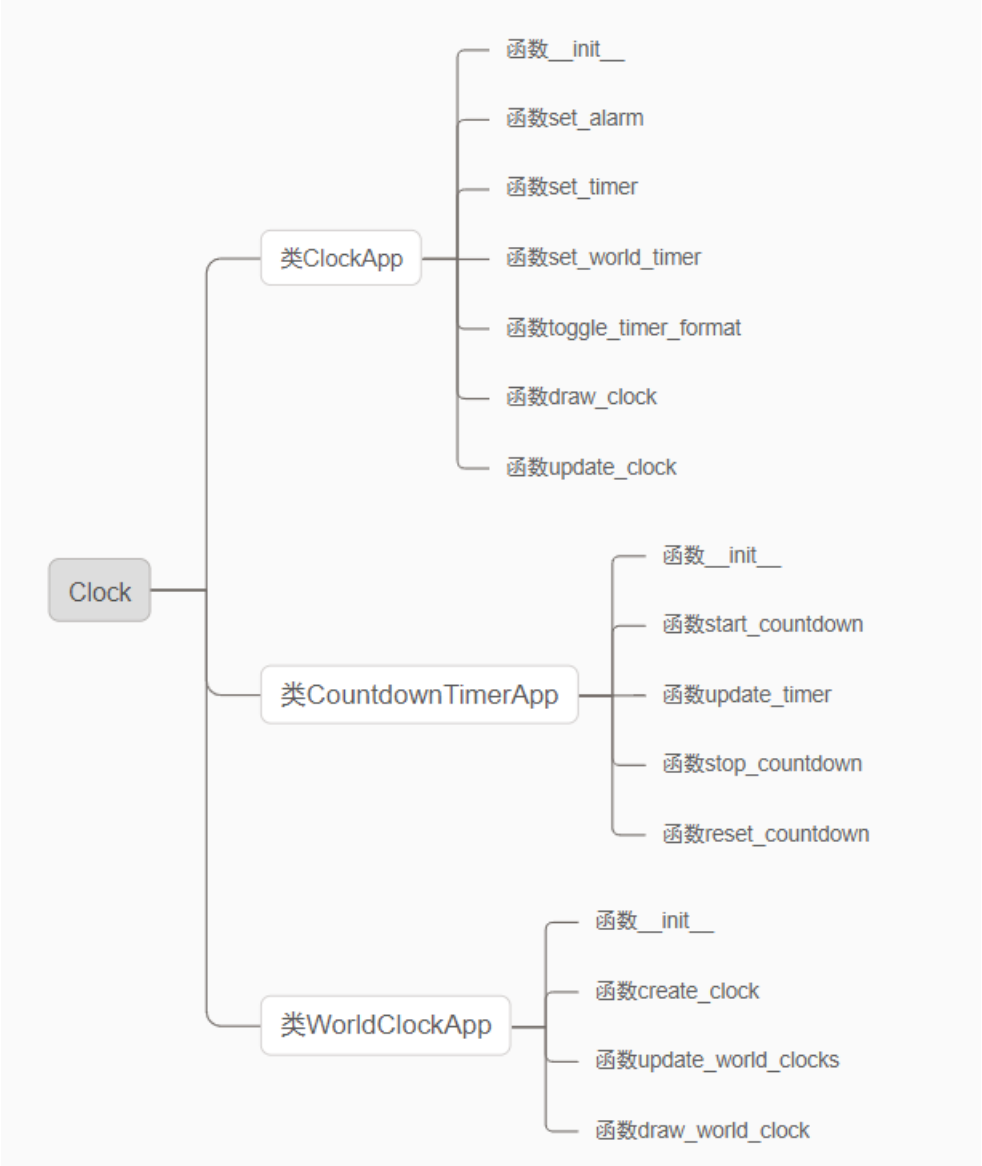
本项目主要基于 vscode 开发，其中安装了 Python 插件，并利用 Anaconda 进行包管理，环境中主要安装的包及其版本如下：

1. pyinstaller 6.1.0
2. pyinstaller 6.1.0
3. pyinstaller-hooks-contrib 2023.10
4. Pillow 10.1.0

三、设计和实现

3.1 总体设计

本项目的设计思路是创建一个 ClockApp 类用于负责完成本项目的所有功能实现，主函数只需要负责以 ClockApp 类为模板生成一个对象并调用即可。本项目的整体框架如下：

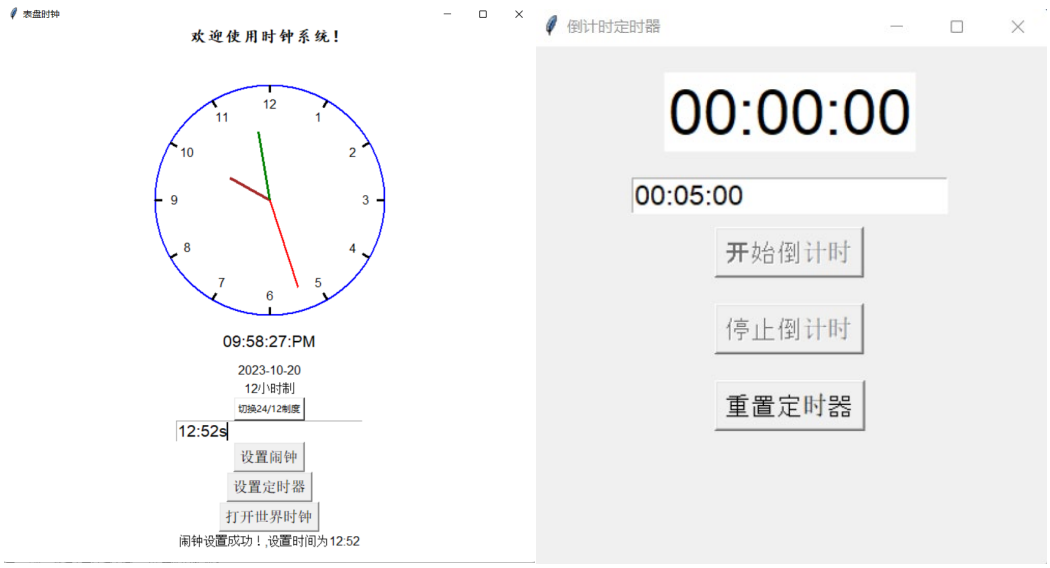


图一、项目整体框架

其中主函数通过调用并创建 ClockApp 对象开始运行程序，ClockApp 中的 set_timer（）函数会调用并创建 CountdownTimerApp 对象开始运行定时器功能，ClockApp 中的 set_world_timer（）函数会调用并创建 WorldClockApp 对象开始运行。其中主界面的时钟的绘制和整点报时以及闹钟的设置，24/12 小时显示切换这四个功能均在 ClockApp 类中完成，设置定时器功能单独在 CountdownTimerApp 类中完成，显示世界时钟功能在 WorldClockApp 类中完成。

3.2 用户界面设计

为了方便用户对时钟程序的使用，利用 tkinter 库绘制了一个主屏幕，用于显示当前北京时间和日期，为了更好的可视化时间，本项目还采用了绘制表盘的方法进一步可视化时钟。如果想要进一步实现 24/12 小时显示制度切换和设置定时器等功能，本项目均提供按钮的形式或文本输入的形式方便用户操作，其中对于闹钟和整点报时，当满足闹钟响铃条件和整点报时条件时，均会通过弹出子窗口的方法提醒用户。对于设置定时器和显示世界时钟需求，本项目提供子屏幕供用户操作，子屏幕中也有详细的文本引导和提示说明，帮助用户更便捷的操作。具体的用户界面如下图所示。



图二、时钟主界面（左）定时器主界面(右)展示

图三、世界时钟主界面展示

3.3 时钟逻辑设计

在本小节，主要叙述本项目的具体设计思路，叙述流程会参考实际代码的运行流程进行分析。

首先，在主函数中负责创建基于 ClockApp 为模板的对象，然后对该对象进行主循环。在 ClockApp 对象的创立中，__init__函数定义了一系列重要的参量和完成了对主屏幕的绘制。首先设定主界面的标题，

尺寸，背景。其次设置欢迎语与绘制画布，然后就开始创建时间日期标签，为了实现后续复杂功能，创建了一些必要的变量和控件用于提示和引导，比如小时显示制度的指示变量和切换 24/12 小时显示制度的按钮，以及设置定时器，世界时钟的按钮等等。完成对主界面的设置后，开始调用 draw_clock（）函数绘制时钟到表盘，并根据 update_clock（）实时更新表盘。在 update_clock()函数中还附有对当前时间是否到达闹钟时间的条件判断，如果达到闹钟时间，程序会弹出子窗口，提示闹钟时间到，并提供一个关闭按钮，同时为了实现我们整点报时的功能，当目前时间的分钟数为 0 时，同样的调用子窗口弹出显示提示语，提示用户现在北京时间的几时。

其次，在上一段介绍了表盘的绘制以及整点报时和闹钟报时的弹出功能后，我们来依次分析其他几个功能是如何实现的，在这里我们依次介绍，闹钟的设置和定时器的设置以及切换显示时制和显示世界时钟的设计思路。

在设置闹钟时，我们在主画布上设置了一个设置闹钟的按钮，该按钮点击时触发 set_alarm（）函数。Set_alarm（）函数设置前首先要清除上次设置的标签。然后利用 tkinter 中的输入函数读入用户输入，并对输入进行格式限制，如果不是以 datetime 里规定的%H:%M 格式，将会显示报错。如果设置成功，将闹钟时间保存到主类变量中，在 update_clock（）函数中进行判断调用。

在定时器的设置函数中，我们之前通过主界面的按钮链接到 set_timer（）函数，该函数主要用于根据 CountdownTimeApp 类创建

对象。在该类的__init__()函数中，定义了定时器界面的按钮和一些提示文本，以及一些重要的指示变量。在该类的 start_countdown（）函数中，将开始调用 update_timer()函数用于开始倒计时，uodate_timer（）函数主要负责在倒计时的过程中更新界面，如果设置的时间还没走完，update_timer 会每隔一秒更新自己一次，如果已经走完，则会显示提示语时间到，并关闭开始倒计时和停止倒计时按钮强制用户进行重置定时器。停止倒计时主要就是用于停止当前倒计时，其实现依赖于将 self.timer_running 置为 False。为了方便用户设置不同的倒计时时间，该类的 reset_countdown（）函数用于重置时间，并对设置的时间进行了一定的规范。

在 24/12 小时显示切换的实现上，我们在类 ClockApp 的主画布上创建了一个按钮，点击该按钮便调用 toggle_timer_format（）函数，通过切换 self.time_format 的形式显示制度的切换。

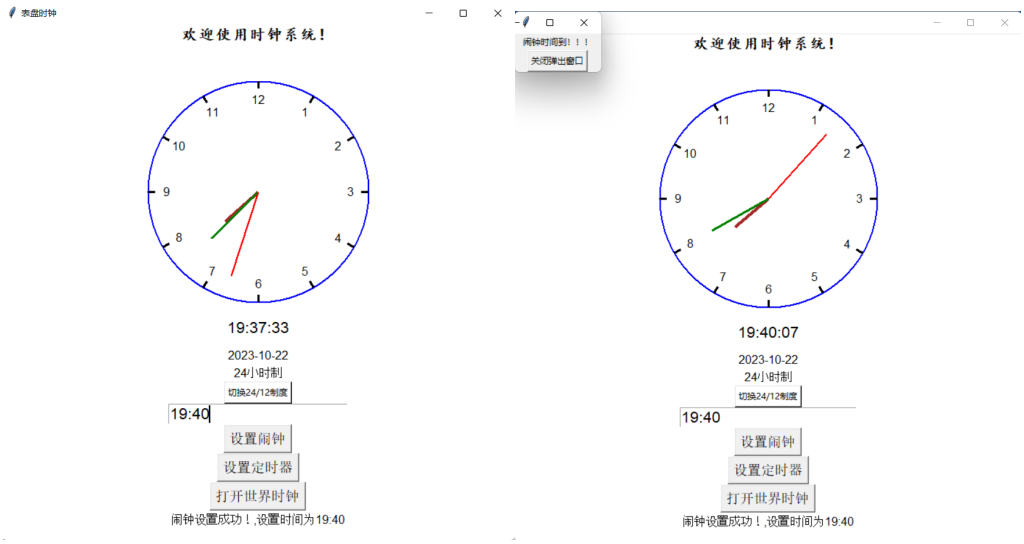
在显示世界时钟的设计上我们在类 ClockApp 的主画布上同样创建一个按钮用于调用 set_world_timer（）函数，该函数用于以类 WorldClockApp（）为模板创建一个对象显示世界时钟。类 WorldClockApp 的__init__（）函数定义了世界时钟画布上的各个控件，create_clock 函数用于创建世界时钟的对象，update_world_clocks 函数用于逐个显示世界各个时区的时间，这里的计算是根据各个时间与本地北京时间的时区差异计算得到，且只选取了几个代表性的时区。draw_world_clock()函数和类 ClockApp 的 draw_clock()函数类似，都是为了在画布上绘制时钟。

四、测试

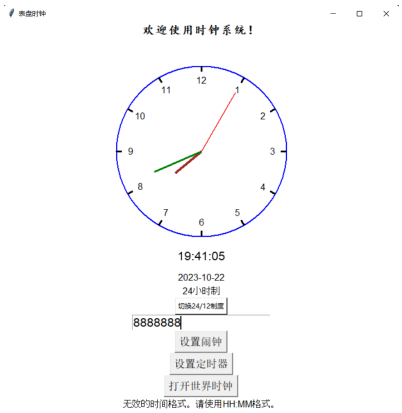
4.1 测试用例

在本程序的测试中，主要测试闹钟的正确输入与不正确输入，以及定时器的正确输入与不正确输入，这里我们对闹钟的测试用例选取为：正确输入：19:40，错误输入 8888888。对定时器的输入的测试用例选取如下：正确输入:00:00:05 ， 错误输入：5555555.

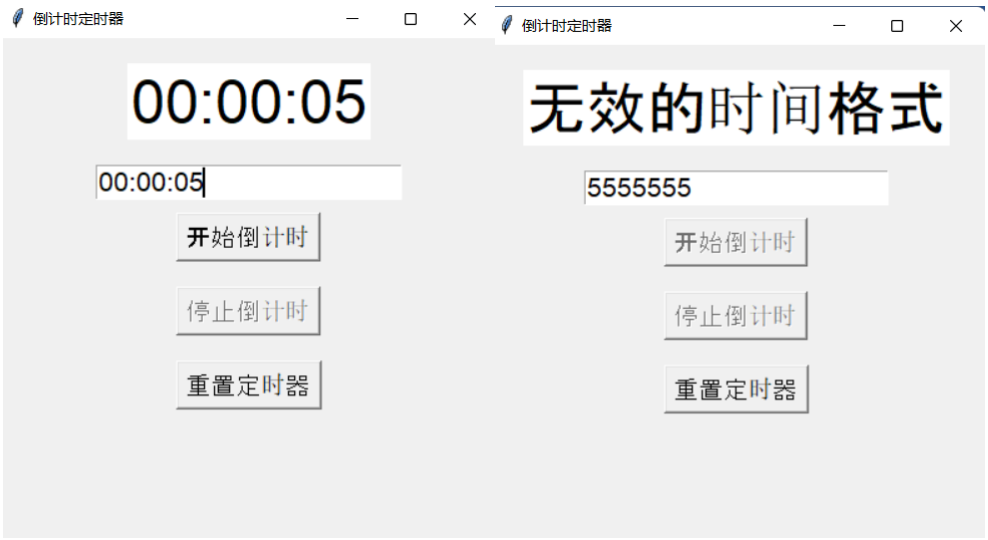
4.2 测试结果



图四、闹钟时间到测试结果



图五、闹钟错误输入测试结果



图六、定时器正确(左)错误(右)结果

五、部署和使用

5.1 部署说明

本项目的附录提供的源代码只能在本地运行，并不是一个可执行的 exe 文件，要封装为 exe 文件，首先需要在命令行窗口中安装 pyinstaller 包和 Pillow 包，版本建议和本文 2.2 节一致。安装好后，在命令行窗口切换到程序安装路径。输入命令 pyinstaller -F -w -i ./images/icon.ico clock.py。其中简单介绍用到的参数的含义：

-F 表示打包(F 大写)

-w 取消控制台显示(w 小写)

-i 有错误也继续执行(i 小写)

ico 图片路径（绝对路径）

最后是代码名称。

安装好后，exe 在当前文件目录下的 dist 文件夹中，点击运行即可。

六、总结

6.1 成果和亮点

1. **多功能性：** 该程序不仅仅是一个简单的时钟，而是一个功能丰富的应用程序。它包括了表盘绘制、时间格式切换、整点报时、定时器、闹钟和世界时钟等多种功能。这种多功能性使该程序对用户来说非常有吸引力，因为它可以满足不同人的不同需求。

2. **用户友好的 GUI：** 使用 Tkinter 创建用户界面是一个很好的选择，因为它在 Python 中是一个流行的 GUI 库，具有良好的可定制性。该程序不仅提供了有用的功能，还以图形方式呈现，让用户能够直观地使用。

3. **封装为可执行文件：** 使用 PyInstaller 将项目封装成一个可执行文件是很聪明的举措。这使得用户不需要安装 Python 或其他依赖，就能轻松运行程序。这增加了项目的可用性。

6.2 挑战和未来改进

1. **跨平台兼容性：** 如果你的应用程序只在特定操作系统上运行良好，可能需要考虑提高跨平台兼容性。这可以通过确保你的程序在 Windows、macOS 和 Linux 等操作系统上都能够正常运行来实现。

2. **用户界面改进：** 考虑进一步改进用户界面的外观和交互性。使用更多的图标、主题或自定义小部件，以提高用户体验。确保用户界面易于导航，同时提供足够的帮助和文档，以帮助用户了解所有功能。

3. **错误处理和稳定性：** 确保你的程序对各种情况进行了适当的错误处理，以提高稳定性。这包括对不正确的输入、硬件问题和其他异常情况的处理。还可以添加日志记录功能，以便在出现问题时能够更轻松地调试。

4. **功能增强：** 考虑增加更多功能，如提供自定义闹钟铃声、定时器警报选项、不同风格的表盘等等，以进一步提高应用程序的吸引力。

七、附录

7.1 源代码

```
import tkinter as tk
from math import cos, sin, pi
from datetime import datetime,timedelta

# 时钟 APP 类
class ClockApp(tk.Tk):
    def __init__(self):
        super().__init__()
        self.title("表盘时钟")
        self.geometry("700x700")
        self.configure(bg='white') # 创建 gui 的 title,大小，背景

        # # 创建 PhotoImage 对象并加载图标图片
        # icon_image = tk.PhotoImage(file="./images/icon.png")
        # self.call('wm', 'iconphoto', self._w, icon_image)
```

```
        self.motto_label = tk.Label(self, text="欢迎使用时钟系统!
",font=('楷体', 16,'bold'), bg='white') # 创建欢迎语
        self.motto_label.pack() # .pack() 是在 tkinter 中用于将部件
（widget）显示在窗口上的方法

        self.canvas = tk.Canvas(self, width=400, height=360, bg='white',
highlightthickness=0)
        self.canvas.pack() #创建画布并显示

        self.time_label = tk.Label(self, font=('Helvetica', 16),
bg='white')
        self.time_label.pack(pady=10) # 创建时间标签并显示

        self.date_label = tk.Label(self, font=('Helvetica', 12),
bg='white')
        self.date_label.pack() # 创建日期标签并显示

        self.hour_hand = None  # 保存时针图像对象
        self.minute_hand = None  # 保存分针图像对象
        self.second_hand = None  # 保存秒针图像对象

        # 创建一个变量储存当前显示的时制，初始为 24 小时制
        self.time_format=tk.StringVar(value="24 小时制")

        self.format_label=tk.Label(self,textvariable=self.time_format,fo
nt=('Helvetica',12),bg='white')
        self.format_label.pack()

        # 创建一个按钮，用于切换制度
        self.toggle_format_button=tk.Button(self,text="切换 24/12 制度
",command=self.toggle_time_format,bg='white')
        self.toggle_format_button.pack()

        # 创建一个输入，让用户通过输入设定闹钟时间
        self.alarm_entry = tk.Entry(self, font=('Helvetica', 16))
        self.alarm_entry.pack()

        # 创建一个按钮，点击后开始设置闹钟
        self.set_alarm_button = tk.Button(self, text="设置闹钟",
command=self.set_alarm, font=('Helvetica', 14))
        self.set_alarm_button.pack()

        # 定义一个变量保存闹钟时间
```

```
        self.alarm_time=None

        # 用于指示闹钟是否响过
        self.alarm_flag=False

        # 用于指示是否进行过整点报时
        self.hour_flag=False

        # 创建一个按钮，点击后开始设置倒计时
        self.set_timer_button = tk.Button(self, text="设置定时器",
command=self.set_timer, font=('Helvetica', 14))
        self.set_timer_button.pack()

        # 创建一个按钮，点击后开始跳出世界时钟界面
        self.set_timer_button = tk.Button(self, text="打开世界时钟",
command=self.set_world_timer, font=('Helvetica', 14))
        self.set_timer_button.pack()

        self.draw_clock() # 调用函数绘制时钟
        self.update_clock() # 根据当前时间更新时钟

    # 该函数用于实现闹钟功能
    def set_alarm(self):
        # 删除之前的标签
        if hasattr(self, 'output_ok'):
            self.output_ok.destroy()
        if hasattr(self, 'output_alarm'):
            self.output_alarm.destroy()
        if hasattr(self, 'output_error'):
            self.output_error.destroy()

        # 根据输入设置闹钟
        alarm_time_str = self.alarm_entry.get()
        try:
            alarm_time = datetime.strptime(alarm_time_str, '%H:%M')
            self.alarm_time = alarm_time
            self.alarm_flag = False
            self.output_ok=tk.Label(self,text="闹钟设置成功！,设置时间为
"+str(alarm_time.hour)+":"+str(alarm_time.minute),font=('Helvetica',
12), bg='white')
            self.output_ok.pack()

        except ValueError:
```

```
            self.output_error=tk.Label(self,text="无效的时间格式。请使用
HH:MM 格式。",font=('Helvetica', 12), bg='white')
            self.output_error.pack()

    # 该函数用于设置定时器
    def set_timer(self):
        count_down_timer_app = CountdownTimerApp()

    # 该函数用于设置世界时钟
    def set_world_timer(self):
        world_clock_app = WorldClockApp()

    # 该函数用于切换 24/12 小时显示制度
    def toggle_time_format(self):
        current_format=self.time_format.get()
        if current_format == "24 小时制":
            self.time_format.set("12 小时制")
        else:
            self.time_format.set("24 小时制")

    # 该函数用于绘制表盘与时分秒针
    def draw_clock(self):
        center_x = 200
        center_y = 200
        radius = 150

        # 绘制表盘
        self.canvas.create_oval(center_x - radius, center_y - radius,
center_x + radius, center_y + radius,
                                width=2, outline='blue')

        # 绘制刻度线和小时标识
        for i in range(12):
            angle = (i / 12) * 2 * pi -1/3 * pi
            x1 = center_x + cos(angle) * (radius - 10)
            y1 = center_y + sin(angle) * (radius - 10)
            x2 = center_x + cos(angle) * radius
            y2 = center_y + sin(angle) * radius
            self.canvas.create_line(x1, y1, x2, y2, width=3)
            hour_x = center_x + cos(angle) * (radius - 25)
            hour_y = center_y + sin(angle) * (radius - 25)
            self.canvas.create_text(hour_x, hour_y, text=str(i+1),
font=('Helvetica', 12), fill='black')
```

```
    # 该函数用于更新主表盘的时钟
    def update_clock(self):
        current_time = datetime.now().time()
        current_date = datetime.now().date()

        # 判断是否调用闹钟
        if self.alarm_time is not None and self.alarm_flag is False:
            if current_time.hour==self.alarm_time.hour and
current_time.minute==self.alarm_time.minute:
                alarm_popup = tk.Toplevel(self)
                alarm_popup.title("闹钟时间到")
                alarm_label = tk.Label(alarm_popup, text="闹钟时间到！！！
")
                alarm_label.pack()
                close_button = tk.Button(alarm_popup, text="关闭弹出窗口
", command=alarm_popup.destroy)
                close_button.pack()
                self.alarm_flag=True

        # 设置整点报时
        if current_time.minute == 0 and self.hour_flag is False:
            hour_popup = tk.Toplevel(self)
            hour_popup.title("整点报时")
            alarm_label = tk.Label(hour_popup, text="现在是北京时间
"+str(current_time.hour)+"点整")
            alarm_label.pack()
            close_button = tk.Button(hour_popup, text="关闭弹出窗口",
command=hour_popup.destroy)
            close_button.pack()
            self.hour_flag=True
        if current_time.minute!=0 and self.hour_flag is True:
            self.hour_flag=False

        # 根据制度标签更新时间标签
        current_format=self.time_format.get()
        if current_format == "24 小时制":
            time_str = current_time.strftime('%H:%M:%S')
        else:
            time_str = current_time.strftime('%I:%M:%S:%p')
        self.time_label.config(text=time_str)

        # 更新日期标签
        date_str = current_date.strftime('%Y-%m-%d')
        self.date_label.config(text=date_str)
```

```
        # 删除上一次的时针和分针图像
        if self.hour_hand:
            self.canvas.delete(self.hour_hand)
        if self.minute_hand:
            self.canvas.delete(self.minute_hand)

        # 更新时针
        hour = current_time.hour
        minute = current_time.minute
        second = current_time.second

        hour_angle = ((hour % 12) + minute / 60 + second / 3600) * (2 *
pi / 12) - pi / 2
        hour_length = hour_x = 200 + cos(hour_angle) * 60
        hour_y = 200 + sin(hour_angle) * 60
        self.hour_hand = self.canvas.create_line(200, 200, hour_x,
hour_y, width=4, fill='brown')

        # 更新分针
        minute_angle = ((minute + second / 60) / 60) * (2 * pi) - pi / 2
        minute_length = 90
        minute_x = 200 + cos(minute_angle) * minute_length
        minute_y = 200 + sin(minute_angle) * minute_length
        self.minute_hand = self.canvas.create_line(200, 200, minute_x,
minute_y, width=3, fill='green')

        # 更新秒针
        second_angle = (second / 60.0) * (2 * pi) - pi / 2
        second_length = 120
        second_x = 200 + cos(second_angle) * second_length
        second_y = 200 + sin(second_angle) * second_length
        if self.second_hand:
            self.canvas.delete(self.second_hand)
        self.second_hand = self.canvas.create_line(200, 200, second_x,
second_y, width=2, fill='red')

        self.after(1000, self.update_clock)

# 定时器类
class CountdownTimerApp(tk.Tk):
    def __init__(self):
        super().__init__()
        self.title("倒计时定时器")
```

```
        self.geometry("400x400")

        self.time_label = tk.Label(self, text="00:00:00",
font=('Helvetica', 36), bg='white')
        self.time_label.pack(pady=20)

        self.time_entry = tk.Entry(self, font=('Helvetica', 16))
        self.time_entry.insert(0, "00:05:00")  # 默认设置为 5 分钟
        self.time_entry.pack()

        self.start_button = tk.Button(self, text="开始倒计时",
command=self.start_countdown, font=('Helvetica', 14),state=tk.DISABLED)
        self.start_button.pack(pady=10)

        self.stop_button = tk.Button(self, text="停止倒计时",
command=self.stop_countdown, font=('Helvetica', 14), state=tk.DISABLED)
        self.stop_button.pack(pady=10)

        self.reset_button = tk.Button(self, text="重置定时器",
command=self.reset_countdown, font=('Helvetica', 14))
        self.reset_button.pack(pady=10)

        self.remaining_time = timedelta()
        self.timer_running = False

    # 开始倒计时
    def start_countdown(self):
        self.timer_running = True
        self.start_button.config(state=tk.DISABLED)
        self.stop_button.config(state=tk.NORMAL)
        self.update_timer()


    # 在倒计时的过程中更新界面
    def update_timer(self):
        if self.remaining_time.total_seconds() > 0 and
self.timer_running is True:
            self.time_label.config(text=str(self.remaining_time).rjust(8
, '0'))
            self.remaining_time -= timedelta(seconds=1)
            self.after(1000, self.update_timer)
        elif self.remaining_time.total_seconds() <= 0:
            self.time_label.config(text="时间到！")
            self.timer_running = False
```

```
            self.start_button.config(state=tk.DISABLED)
            self.stop_button.config(state=tk.DISABLED)

    # 停止倒计时
    def stop_countdown(self):
        self.timer_running = False
        self.start_button.config(state=tk.NORMAL)
        self.stop_button.config(state=tk.DISABLED)

    # 重置定时器
    def reset_countdown(self):
        input_time = self.time_entry.get()
        self.timer_running = False
        try:
            hours, minutes, seconds = map(int, input_time.split(':'))
            self.remaining_time = timedelta(hours=hours,
minutes=minutes, seconds=seconds)
            self.time_label.config(text=str(self.remaining_time).rjust(8
, '0'))
            self.start_button.config(state=tk.NORMAL)
            self.stop_button.config(state=tk.DISABLED)

        except ValueError:
            self.time_label.config(text="无效的时间格式")


# 世界时钟类
class WorldClockApp(tk.Tk):
    def __init__(self):
        super().__init__()
        self.title("世界时钟")
        self.geometry("1200x600")
        self.configure(bg='white')

        self.motto_label = tk.Label(self, text="欢迎使用世界时钟系统！",
font=('楷体', 16, 'bold'), bg='white')
        self.motto_label.pack()

        self.clock_frames = []  # 存储时钟小部件的列表
        self.time_formats = []  # 存储时制标签的列表
        self.locations = ['UTC', 'New York', 'London', 'Paris',
'Tokyo']  # 添加不同城市的时钟

        for location in self.locations:
            frame = tk.Frame(self, bg='white')
```

```
            frame.pack(side=tk.LEFT, padx=20)
            label = tk.Label(frame, text=location, font=('Helvetica',
16), bg='white')
            label.pack()
            clock = self.create_clock(frame)
            self.clock_frames.append(clock)
            self.time_formats.append(tk.StringVar(value="24 小时制"))

        self.update_world_clocks()

    # 创建世界时钟
    def create_clock(self, frame):
        canvas = tk.Canvas(frame, width=200, height=180, bg='white',
highlightthickness=0)
        canvas.pack()

        time_label = tk.Label(frame, font=('Helvetica', 16), bg='white')
        time_label.pack(pady=10)

        self.hour_hand = None
        self.minute_hand = None
        self.second_hand = None

        return {
            'canvas': canvas,
            'time_label': time_label,
            'hour_hand': None,
            'minute_hand': None,
            'second_hand': None
        }

    # 更新世界时钟
    def update_world_clocks(self):
        for i, location in enumerate(self.locations):
            current_time = datetime.now()
            time_format = self.time_formats[i].get()
            timezone_diff = 0

            if location == 'UTC':
                timezone_diff = 0
            elif location == 'New York':
                timezone_diff = -5
            elif location == 'London':
                timezone_diff = 0
```

```
            elif location == 'Paris':
                timezone_diff = 1
            elif location == 'Tokyo':
                timezone_diff = 9

            current_time += timedelta(hours=timezone_diff)

            if time_format == "24 小时制":
                time_str = current_time.strftime('%H:%M:%S')
            else:
                time_str = current_time.strftime('%I:%M:%S:%p')

            self.clock_frames[i]['time_label'].config(text=f"{location}:
{time_str}")

            self.draw_world_clock(self.clock_frames[i], current_time)

        self.after(1000, self.update_world_clocks)

    # 更新世界时钟指针
    def draw_world_clock(self, clock_frame, current_time):
        hour = current_time.hour
        minute = current_time.minute
        second = current_time.second

        if clock_frame['hour_hand']:
            clock_frame['canvas'].delete(clock_frame['hour_hand'])
        if clock_frame['minute_hand']:
            clock_frame['canvas'].delete(clock_frame['minute_hand'])

        hour_angle = ((hour % 12) + minute / 60 + second / 3600) * (2 *
pi / 12) - pi / 2
        hour_length = 40
        hour_x = 100 + cos(hour_angle) * hour_length
        hour_y = 90 + sin(hour_angle) * hour_length
        clock_frame['hour_hand'] =
clock_frame['canvas'].create_line(100, 90, hour_x, hour_y, width=4,
fill='brown')

        minute_angle = ((minute + second / 60) / 60) * (2 * pi) - pi / 2
        minute_length = 70
        minute_x = 100 + cos(minute_angle) * minute_length
        minute_y = 90 + sin(minute_angle) * minute_length
```

```
        clock_frame['minute_hand'] =
clock_frame['canvas'].create_line(100, 90, minute_x, minute_y, width=3,
fill='green')


if __name__ == "__main__":
    app = ClockApp()
    app.mainloop()
```

7.2 参考资料

1. 如何将 python3 gui 界面（py 文件）打包成 exe 文件

2. 菜鸟教程(python GUI)

3. ChatGPT

4.《Python 编程 从入门到实践》 [美] Eric Matthes 著，袁国忠译