Disciplina: Desenvolvimento Web I
AD1 1^o semestre de 2022.
Professor: Paulo Roma

AD1: Relógio GMT

1 Objetivo

Neste projeto será implementado um relógio GMT (Greenwich Mean Time) idêntico à face correspondente do Apple Watch [1]. O relógio GMT foi criado pela Rolex[2] em 1954 a pedido da Pan American Airlines, como uma solução para pilotos que viajam por diversas zonas de tempo, mas mantendo o tempo natal (da cidade original do viajante).

Basicamente, há um segundo ponteiro vermelho de horas para um segundo fuso horário[3]. Entretanto, ele completa uma rotação de 360° em 24 horas, ao invés de 12 horas[4]. Se o segundo fuso for igual ao fuso natal, o ponteiro aponta para cima à meia noite, e para baixo ao meio dia. Pode-se contar as horas usando-se as marcas da coroa (*bezel*), e há 24, incluindo a seta localizada no topo. Assim, o mostrador externo permite saber se a hora é AM ou PM, por exemplo, 8h ou 20h.

Se for escolhido um segundo fuso horário[5] (por ex., tocando no centro do relógio), será lido no mostrador externo (coroa), no ponto onde ponteiro vermelho aponta, a hora correspondente ao fuso horário alternativo (figura 1b). Isto será 1-12, se o relógio estiver no modo de 12 horas, ou 1-24 se o relógio estiver no modo de 24 horas (o modo de tempo é escolhido no iPhone pareado).

A cor da coroa muda para a hora do nascer / por do sol[6] [7] do fuso escolhido. Se não for escolhido um segundo fuso horário, ainda é possível tocar e escolher a hora natal (fuso corrente ou natal). Isto fará com que as cores da coroa mudem para a hora do nascer / por do sol[8] [9] e que seja exibida a hora completa (AM/PM) mais recente do fuso natal. Tocando no centro e escolhendo 'NONE', como na figura 1a, fará com que a coroa mude para aquilo que se vê, por ex., 06:00 até 18:00, se for lida a hora do mostrador externo de 24 horas.

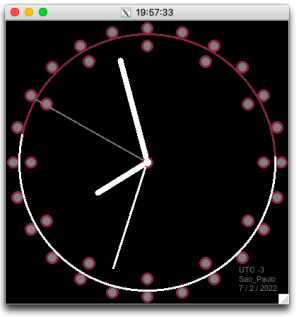
[1] `https://support.apple.com/guide/watch/faces-and-features-apde9218b440/watchos`
[2] `https://www.chrono24.com/rolex/gmt-master-ii--mod4.htm`
[3] `https://pt.wikipedia.org/wiki/Fuso_hor%c3%a1rio`
[4] `https://www.youtube.com/watch?v=3jTaFmy38xM`
[5] `https://www.youtube.com/watch?v=sEQPB9Pkrd8`
[6] `https://www.aa.quae.nl/en/reken/zonpositie.html`
[7] `https://www.inf.ufrgs.br/~cabral/Nascer_Por_Sol.html`
[8] `https://astral.readthedocs.io/en/latest/`
[9] `https://geopy.readthedocs.io/en/stable/`

(a) Relógio GMT: 10:09 AM.

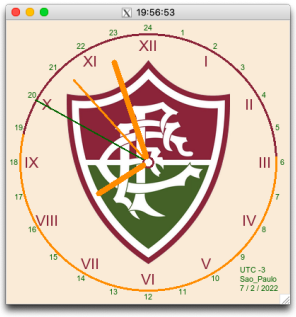
(b) Fuso: Londres (0:29h).

(c) Fusos horários.

Figura 1: Apple Watch.



(a) Relógio simples.



(b) Relógio tricolor.

(c) Relógio nascer/por do sol.

Figura 2: Relógio animado.

2 Implementação

A seção anterior descreveu a interface a ser implementada com o componente canvas do tkinter[10] na AD2. Para implementar uma animação básica que mova os ponteiros do relógio (figura 2), bem como ajustar as suas dimensões, são necessárias algumas funções[11].

Código 1: Mapeamentos

```
from datetime import datetime, timedelta
from math import sin, cos, pi

class clock:
    ....
    ## Converte um vetor de coordenadas polares para cartesianas.
    #  - Note que três horas está em 0°.
    #  - Para o relógio, no entanto, 0° está em doze horas.
    #
    #  @param angle ângulo do vetor.
    #  @param radius comprimento do vetor.
    #  @return um ponto 2D.
    #
    def polar2Cartesian(self, angle, radius=1):
        angle = pi / 2 - angle
        return (radius * cos(angle), radius * sin(angle))

    ## Desenha um ponteiro.
    #  Atribuindo-se o tag 'handles' aos ponteiros do relógio, a animação dos ponteiros
    #  pode ser feita sem ter de redesenhar o canvas completamente.
    #
    # @param angle ângulo do ponteiro.
    # @param len comprimento do ponteiro.
    # @param wid largura do ponteiro.
    #
    def draw_handle(self, angle, len, wid=None):
        x, y = self.polar2Cartesian(angle, len)
        cx, cy = self.polar2Cartesian(angle, 0.05)
        self.canvas.create_line(cx, cy, x, y,
                                fill=self.timecolor, tag='handles', width=wid, capstyle=ROUND)

    ## Desenha os três ponteiros do relógio.
    def paint_hms(self):
        # remove apenas os ponteiros
        self.canvas.delete('handles')

        # Para o fuso horário do Rio de Janeiro, self.delta vale -3
        # (três horas para trás ou duas, no horário de verão).

        # hora, minutos e segundos: tempo UTC + delta horas
        h,m,s = datetime.timetuple(datetime.utcnow()+timedelta(hours = self.delta))[3:6]

        oneMin = pi / 30  # um minuto vale 6 graus
```

[10] https://python-course.eu/tkinter/canvas-widgets-in-tkinter.php
[11] https://lcg.ufrj.br/cwdc/10-html5css3/11.5.php?timeZone=America/Sao_Paulo

```
        fiveMin = pi / 6  # cinco minutos ou uma hora vale 30 graus

        hora = fiveMin * (h + m / 60.0)
        minutos = oneMin * (m + s / 60.0)
        segundos = oneMin * s

        self.draw_handle(hora, 0.5, 10)       # ponteiro das horas
        self.draw_handle(minutos, 0.9, 10)    # ponteiro dos minutos
        self.draw_handle(segundos, 0.95, 3.6) # ponteiro dos segundos
```

A forma mais simples de calcular as coordenadas dos ponteiros, é mapeando o tempo em ângulos e depois mapeando os ângulos para coordenadas cartesianas, conforme pode ser visto no código1.

Portanto, para desenhar cada ponteiro do relógio, basta traçar uma linha com origem no centro da janela (canvas) e extremidade nas coordenadas (x,y). Embora seja possível desenhar um relógio no terminal[12], isso envolveria um esforço considerável e foge ao escopo desse trabalho.

Para criar a animação é necessário redesenhar os ponteiros pelo menos a cada segundo. Como todo componente do tkinter possui um método after, um código como este é suficiente:

Código 2: Animação

```
    ## Movimenta o relógio, redesenhando os ponteiros
    #  após um certo intervalo de tempo.
    #
    def poll(self):
        self.paint_hms()  # só é necessário redesenhar os ponteiros a cada 200 ms
        self.root.after(200, self.poll)
```

3 Tarefas complementares

1. Os códigos 1 e 2 fazem parte de uma classe *clock* que deve ser implementada. No seu construtor devem estar definidas todas as variáveis e constantes necessárias ao funcionamento do relógio.

2. Como o relógio pode funcionar em diversos fusos horários[13], a variável *delta* do método *paint_hms*, que representa a compensação do tempo UTC, é uma variável de objeto, e na interface gráfica, seu valor deve ser fornecido de alguma forma. No Apple Watch, quando se toca na tela do relógio, abre-se um menu com todos os fusos horários possíveis (figura 1c). Supondo que o UTC agora seja 13:00h, então, em São Paulo será 10:00h (UTC -3). Abaixo, listamos alguns códigos e seus deslocamentos UTC[14], para algumas cidades da América do Norte.

    - SP 10:00 (UTC -3) São Paulo

[12] https://github.com/tenox7/aclock
[13] https://time.is/pt_br/UTC
[14] https://en.wikipedia.org/wiki/List_of_tz_database_time_zones

- ....
- EDM 6:00 (UTC -7) Edmonton, Alberta
- MC 7:00 (UTC -6) McKinney, Texas
- OMA 7:00 (UTC -6) Omaha, Nebraska
- CHI 7:00 (UTC -6) Chicago, Illinois

3. Sugiro ler, de um arquivo *localtime.txt* (ou *.json*[15]), um conjunto de coordenadas de latitude e longitude, e, a partir dele, descobrir os fusos horários (*timezones*) das cidades correspondentes[16]:

```
-22.908333 -43.17375348463498       # São Paulo -3
-3.117034 -60.025780                # Manaus -4
40.730610 -73.935242                # New York -5
41.881832 -87.623177                # Chicago -6
51.049999 -114.066666               # Calgary -7
32.698437 -114.650398               # Yuma -7
71.290556 -156.788611               # Barrow -9
62.00973 -6.77164                   # Tórshavn 0
48.864716 2.349014                  # Paris +1
41.902782 12.496366                 # Rome +1
38.246639 21.734573                 # Patras +2
55.751244 37.618423                 # Moscow +3
```

Código 3: localtime.txt

   Com o fuso horário pode-se obter facilmente o deslocamento UTC e os dados solares para determinar as horas do nascer e por do sol, com o pacote Astral, como pode ser visto no código 4.

4. A janela do relógio deve poder ser redimensionada fazendo com que o relógio aumente ou diminua de tamanho, mas sem distorção.

5. É comum que empresas de desenvolvimento de software obriguem que testes sejam escritos, antes da implementação de cada método. Nesta tarefa, usaremos o unittest[17], para escrever testes unitários, que basicamente comparam os resultados dos cálculos, após a execução de um método.

6. Cada classe, método, e variável de instância, quer pública ou privada, deve ter um comentário ao estilo Doxygen[18]. Classes devem incluir a etiqueta @author, e métodos devem incluir as etiquetas @param e @return quando apropriado[19].

[15] https://lcg.ufrj.br/cwdc/10-html5css3/localtime.json
[16] https://pypi.org/project/timezonefinder/
[17] https://www.youtube.com/watch?v=6tNS--WetLI
[18] https://www.doxygen.nl/index.html
[19] https://lcg.ufrj.br/python/ADs/AD1_2020-1.pdf#page=5

7. Crie um arquivo clockTest.py com uma classe que utiliza o unittest[20] para testar cada método/função da sua classe *clock*.

8. A maior parte do trabalho é a criação da interface gráfica, que só precisa ser feita completamente na AD2. Porém, boa parte da classe *clock* precisa estar pronta e testada. Você pode se inspirar no código javascript do meu curso de Desenvolvimento Web[21]. Idealmente, você deveria implementar uma interface simples, similar à figura 2a, na AD1 e a interface GMT na AD2. Contudo, isto vai lhe obrigar a interagir com o tkinter antes das aulas ocorrerem, de acordo com o cronograma. Todavia, se você não fizer desta forma, a AD2 ficará muito pesada, mas esta decisão é sua.

```
from astral import LocationInfo
from astral.sun import sun
from timezonefinder import TimezoneFinder
import pytz
import json
     ....

     self.places = []
     try:
         f = open('./localtime.json', 'r')
         # returns JSON object as a dictionary
         data = json.load(f)
         for c in data['cities']:
             self.places.append(
                        (c['coordinates']['latitude'], c['coordinates']['longitude']))
         f.close()
     except Exception as e:
         print(e)
         print("No localtime file available")

     ...

     self.region, self.local = self.timezone.split("/")

     self.deltahours = pytz.timezone(self.timezone).utcoffset(
         datetime.now()).total_seconds() / 3600

     self.delta = timedelta(hours=self.deltahours)

     city = LocationInfo(self.local, self.region, self.timezone,
                         latitude=self.latitude, longitude=self.longitude)

     today = datetime.date(datetime.now())

     # Sun rise x sun set
     sun_data = sun(city.observer, today,
                    tzinfo=pytz.timezone(self.timezone))
```

[20] https://docs.python.org/3/library/unittest.html
[21] https://www.lcg.ufrj.br/cwdc/10-html5css3/docjs/index.html

```
    hr, mr, _ = datetime.timetuple(sun_data['sunrise'])[3:6]
    hs, ms, _ = datetime.timetuple(sun_data['sunset'])[3:6]
```

Código 4: Dados solares

9. Se você optar por produzir a saída da AD1 no terminal, instancie um objeto tk e utilize o método after de qualquer componente para imprimir o tempo num terminal a cada 200ms, conforme pode ser visto no código 5.

```
import sys
import os
from datetime import timedelta, datetime
from tkinter import *

def poll():
    global root
    delta = -3
    t = datetime.timetuple(datetime.utcnow()+timedelta(hours = delta))[3:6]
    print("{:02d}:{:02d}:{:02d}".format(*t,'02'),end='\r')
    root.after(200, poll)

def main():
    global root
    root = Tk()
    root.geometry('+0+0')
    poll()
    root.mainloop()

if __name__ == '__main__':
    sys.exit(main())
```

Código 5: Relógio no terminal