Disciplina: Programação com Interfaces Gráficas
AD2 2^o semestre de 2020.

AD2: Simulador de Doenças

1 Objetivo

O objetivo da AD2 é complementar as tarefas que ficaram faltando na AD1:

- A interface gráfica para a simulação é relativamente simples, porque basicamente deveria exibir a Figura 2 da AD1. Para tornar as coisas um pouco mais interessantes, vamos tomar a liberdade de incluir alguns requisitos novos.
  - As forças das doenças ficam constantes em 1 ou crescem indefinidamente. Portanto, se quisermos, por exemplo, que os raios dos círculos sejam proporcionais a elas, estes acabariam por tomar toda a janela, eventualmente.
  - O segundo ponto é que não devemos permitir que círculos se intersectem entre si, ou que cruzem as fronteiras dos quadrantes que os contém.
- A saída do programa também deve ser efetuada usando componentes de interface apropriados.
  - Normalmente, usuários não estão acostumados a usar terminais que executam um shell qualquer, como bash ou tcsh.
  - Portanto, exiba a saída da simulação usando componentes de interface adequados. Não há um formato fixo. Use a sua criatividade e bom senso. Utilize um canvas do Tkinter para exibir graficamente as doenças no mundo.
- A classe SimulationPanel a ser implementada deve ser a mais adequada possível aos requisitos da aplicação.

2 Diretivas Gerais

O principal problema de implementação é que o tamanho da janela, que exibe a simulação, não tem nada a ver com o número de células do mundo. Por exemplo, a janela pode ser 512×512 pixels, e o mundo pode ser 8×8, ou seja, com 64 células.

Matematicamente, é um problema de determinar a transformação linear que mapeia um retângulo no outro. Ou, em outras palavras, encontrar a escala e translação, em cada dimensão, apropriadas.

A classe que implementa o mapeamento de pontos de um retângulo no outro, chamados de window no espaço do mundo, e viewport no espaço da tela (em pixels), é fornecida no código 1. Mesmo que você não entenda a matemática, porque não estamos num curso de CG, é importante conseguir usar a classe, a partir do exemplo fornecido no método `main()`.

```
#!/usr/bin/env python
# coding: UTF-8
#

import sys

class mapper:
    ## Constructor.
    #
    #  @param world window rectangle.
    #  @param viewport screen rectangle.
    #  @param ydown whether Y axis is upside down.
    #  @param noDistortion whether to use the same scale for both X and Y.
    #
    def __init__(self, world, viewport, ydown=True, noDistortion=True):
        self.world = world
        self.viewport = viewport
        x_min, y_min, x_max, y_max = self.world[:4]
        X_min, Y_min, X_max, Y_max = self.viewport[:4]
        self.fx = float(X_max-X_min) / float(x_max-x_min)
        self.fy = float(Y_max-Y_min) / float(y_max-y_min)
        self.ys = -1 if ydown else 1

        if noDistortion:
           self.f = min(self.fx,self.fy)
           self.fx = self.fy = self.f

        x_c = 0.5 * (x_min + x_max)
        y_c = 0.5 * (y_min + y_max)
        X_c = 0.5 * (X_min + X_max)
        Y_c = 0.5 * (Y_min + Y_max)
        self.c_1 = X_c - self.fx * x_c
        self.c_2 = Y_c - self.fy * y_c

    ## Maps a single point from world coordinates to viewport
    #  (screen) coordinates.
    #
    #  @param x, y given point.
    #  @return a new point in screen coordinates.
```

```
        #
        def __windowToViewport(self, x, y):
            X = round(self.fx *           x + self.c_1)
            # Y axis maybe upside down
            Y = round(self.fy * self.ys * y + self.c_2)
            return X , Y

        ## Maps a single vector from world coordinates to viewport
        #  (screen) coordinates.
        #
        #  @param x, y given vector.
        #  @return a new vector in screen coordinates.
        #
        def windowVecToViewport(self, x, y):
            X = round(self.fx *           x)
            # Y axis maybe upside down
            Y = round(self.fy * self.ys * y)
            return X , Y


        ## Maps a single point from screen coordinates to window
        #  (world) coordinates.
        #
        #  @param x, y given point.
        #  @return a new point in world coordinates.
        #
        def viewportToWindow(self, x, y):
            X = (x - self.c_1) / self.fx
            Y = (y - self.c_2) / self.fy
            return X , Y

        ## Maps points from world coordinates to viewport (screen) coordinates.
        #
        #  @param p a variable number of points.
        #  @return two new points in screen coordinates.
        #
        def windowToViewport(self,*p):
            return [self.__windowToViewport(x[0],x[1]) for x in p]

    def main():
        # maps the unit rectangle onto a viewport of 400x400 pixels.
        map = mapper([-1,-1,1,1], [0,0,400,400])
        p1, p2 = map.windowToViewport((0,0),(1,1))
        p = map.viewportToWindow(400,400)
        print ("%s - %s" % (p1,p2)) # (200, 200) - (400, 0)
        print ("(%d,%d)" % p)       # (1,1)

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

Código 1: mapper - transformação window para viewport.

O procedimento a ser usado então é mapear cada ponto em coordenadas do mundo para

coordenadas de tela, antes de chamar qualquer método da classe canvas, como `create_line` ou `create_oval`.

A figura 1 exibe um mundo com 8 doenças, numa grade 720×640, distribuídas pelos seus 4 quadrantes, e a Figura 2 uma grade 8×8 com 64 células.

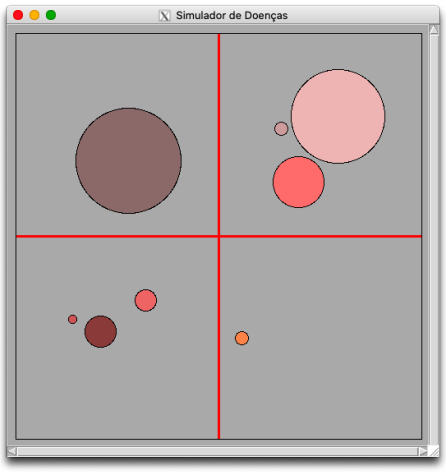

Figura 1: Mundo da simulação.

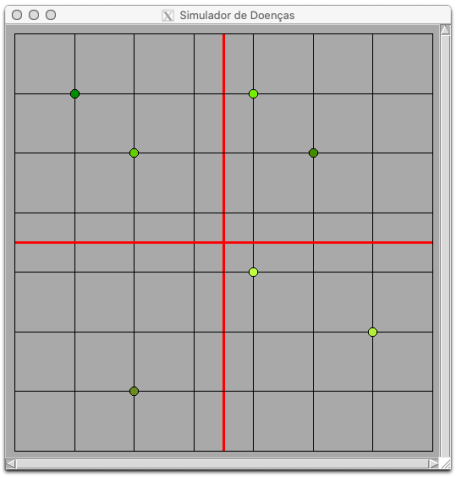

Figura 2: Grade de células.

3 Tarefas complementares

1. Para garantir que nenhum círculo intersecte algum outro, é necessário manter um dicionário, com todos os círculos no mundo, que associa centros - onde as chaves são tuplas (x_c, y_c) - aos seus raios. Cada vez que o raio de um círculo mudar, ou um novo círculo for criado, o dicionário deve ser atualizado de acordo.

2. Uma forma simples de garantir que um raio proporcional a força da doença, o, que representa, seja limitado, é simplesmente fazendo o módulo com o maior raio possível que não gere nenhuma interseção: $r = o.getStrength()\ \%\ r_{max}$. Isso fará com que os raios sejam menores que o maior raio possível, e que os círculos aumentem e diminuam ao longo do tempo, permitindo uma animação da simulação.

3. O código 3 implementa uma classe `Timer` para permanecer chamando a rotina de desenho `draw()` após um intervalo de tempo de 0.5 segundos, e o código 2 implementa a função `main` do simulador.

4. Use o botão 1 do mouse para indicar o centro de novos círculos a serem adicionados. Os parâmetros de crescimento podem ter uma razão fixa qualquer e as temperaturas mínima e máxima podem ser um a memos e um a mais, respectivamente, em relação a temperatura do quadrante onde o ponto foi inserido.

5. Finalmente, adicione uma opção para salvar todas as doenças do mundo corrente em um arquivo, no mesmo formato do simulation.config.

```
def main():
    wsizex = 512
    wsizey = 512
    margin = 10
    root = Tk()
    root.title("Simulador de Doenças")
    world = MyWorld()
    # maps the world rectangle onto a viewport of wsizex x wsizey pixels.
    canvas = Canvas(root,width=wsizex,height=wsizey,background='dark grey')
    sp = SimulationPanel(world,canvas)
    sp.wvmap = mapper([0,0,world.getWidth()-1,world.getHeight()-1], \
                      [margin,margin,wsizex-margin,wsizey-margin], False, False)
    poll = Timer(root,sp.draw,500)
    canvas.bind("<Configure>", sp.resize)
    root.bind("<Escape>", lambda _ : root.destroy())
    root.bind("s", lambda _ : poll.stop())
    root.bind("r", lambda _ : poll.restart())
    root.bind("p", sp.printData)
    root.bind("<Button-1>", lambda e: sp.mousePressed(e))
    poll.run()
    root.mainloop()
if __name__=='__main__':
   sys.exit(main())
```

Código 2: Função main do simulador.

O resultado da animação pode ser visto no vídeo [1].

```
class Timer:
    """Keep packing (drawing) circles, after a certain time interval."""

    def __init__(self,root,callback,delay):
        self.root = root
        self.callback = callback
        self.delay = delay
        self.task = None

    def run(self):
        """Run the callback function every delay ms."""
        self.callback()
        self.task = self.root.after(self.delay,self.run)

    def stop(self):
        """Stop the drawing process."""
        if self.task is not None:
           self.root.after_cancel(self.task)
           self.task = None

    def restart(self):
        """Restart the drawing process."""
        self.stop()
        self.run()
```

Código 3: Timer - para criar uma animação.

[1] http://orion.lcg.ufrj.br/python/ADs/world.mp4