Jogo Minas

Fup - CC - prof: Anderson Lemos

Equipe: Henricky, Júlio

In [1]:

```
#IMPORTANDO BIBLIOTECAS
#
import pygame as pg # biblioteca de jogos
from pygame.locals import *
import pygame.font #trabalhar com fontes
from time import sleep #pausar o sistema
from random import randint #gerar numeros aleatórios
import numpy as np #trabalhar com vetores e matemática em geral
import sys
```

```
pygame 1.9.6
Hello from the pygame community. https://www.pygame.org/contribute.ht
ml (https://www.pygame.org/contribute.html)
```

Principais problamas:

Matemático:

- gerar uma tabela(matriz) com as bombas em posições aleaórias e os número numeros que representam as bombas a sua volta.

Gráfico:

- tela inical com botões que geram as céluas do jogo em dimenão (8x8, 12x12, 16x16, 20x20).
- telado jogo:
  - grid em banco onde pode-se clicar e mostrar os números da tabela.
    - obs: a tabela tem que ser criada após o click para que o jogador não comece perdendo.
  - mostrar o tempo na tela e o botão de reiniciar partida.

In [2]:

```
#TESTE PARA FONTES
#f=pygame.font.SysFont('Arial', 30)
#t=f.render('Your score was: '+str(score), True, (0, 0, 0))
#screen.blit(t, (10, 270))
```

In [6]:

```
#CLASSES

class colors: #Criada para não precisar escrever as tuplas RGB.
    def __init__(self):
        self.black=(0,0,0)
        self.white=(255,255,255)
        self.gray=(100, 100, 100)
        self.red=(255,0,0)
        self.green=(0,255,0)
        self.blue=(0,0,255)
        self.red_e=(120, 0, 0)
        self.green_e=(0, 120, 0)
cor= colors()
cor.white
class Default: #Criada para deixar valores padrões.
    window=[800,600]
    n=8 #grid_game_divisões
    aux=window[0]-window[1] #delta x: área de score |x x x|s|
    Game_area=[window[0]-aux,window[1]]
    title="Minas"
    Img_bomb = pg.image.load("bomb.png")
    #
############################

class Grid: # crindo linhas verticais e horizontais de tamanho seguindo padrões.
    G_cor=cor.white
    G_y=pg.Surface([1,Default.Game_area[1]])
    G_x=pg.Surface([Default.Game_area[0],1])
    G_x.fill(G_cor)
    G_y.fill(G_cor)
    G_pos=[Default.window[0]/2,Default.window[1]/2]
#############################################################################
# Classe principal onde há as funções principais do jogo
class APP:#################################################################
    cor=colors()
    window=Default.window
    title=Default.title
    n=Default.n

    def __init__(self):
        self.BG = None
        self.run = True
        self.G = Grid()
        self.Qb=round(self.n**2*0.15)#quantidade de bombas
        self.M_b=np.zeros([self.n+2,self.n+2])
        self.M_n=np.zeros([self.n+2,self.n+2])
        self.print_matriz=1
    #Matriz########
    def bomba(self):
        i=0
        V=[]
        while(i<self.Qb):
            x=randint(1,8)
            y=randint(1,8)
            if [x,y] not in V:
                V.append([x,y])
                i+=1
        for j in V:
            self.M_b[j[0]][j[1]]=1
```

```
    def Ver_redor(self,Mb,k,l):
        soma=0
        for i in range(-1,2):
            for j in range(-1,2):
                soma+=Mb[k+i][l+j]
        return soma

    def Num_redor(self):#M_bomb matriz de comparação

        aux=len(self.M_b[0])
        aux2=len(self.M_b[1])

        for i in range(1,aux-1):
            for j in range(1,aux2-1):
                if(self.M_b[i][j]==0):
                    self.M_n[i][j] = self.Ver_redor(self.M_b,i,j)
                else:
                    self.M_n[i][j] =-1
    ################
    def Button(self,text='Button',pos=2): #com base no que vai printar na tela inic
        if(pos==2):#x: um terço menos metade do tamanho da string +/- (para central
            x=Default.window[0]/3-(Default.window[0]/3)/8
            y=Default.window[1]/3
        elif(pos==3):
            x=Default.window[0]/3-(Default.window[0]/3)/8
            y=Default.window[1]*2/3

        elif(pos==4):
            x=Default.window[0]*2/3-(Default.window[0]/3)/8
            y=Default.window[1]/3
        else:
            x=Default.window[0]*2/3-(Default.window[0]/3)/8
            y=Default.window[1]*2/3
        botao=pg.Surface([150,150])
        botao.fill(cor.gray)
        self.BG.blit(botao,[x-75*2/3+10,y-70])

        f=pygame.font.SysFont(None,30)
        t=f.render(text, True, cor.green)
        B_pos=[x,y]
        self.BG.blit(t, B_pos)
    def Quit_game(self):
        self.BG.fill(cor.red_e)
        pg.display.update()
        sleep(0.5)
        pg.quit()
    def dr_grid(self):
        k=Default.Game_area[0]/self.n
        for x in range(0,int(Default.Game_area[0]/k +1)):
            for y in range(0,int(Default.Game_area[0]/k+1)):
                self.BG.blit(self.G.G_y,[x*k,0])
                self.BG.blit(self.G.G_x,[0,y*k])
    def draw_num(self):
        n=self.n #numero de linhas=colunas
        dist=(Default.Game_area[0]/n) # larg de cada celula
        f=pg.font.SysFont('Arial',30) #objeto fonte
        x,y=dist/3,dist/3 #posição inicial, centraliada na celula
        for i in range(1,len(self.M_b[0])-1):
            if(i!=1):x+=dist
            for j in range(1,len(self.M_b[0])-1):
                if(j!=1):y+=dist
```

```
                else: y=dist/3
                if(self.M_n[i][j]==-1):
                    self.BG.blit(Default.Img_bomb,[x-5,y-5])
                    t=f.render(str(int(self.M_n[i][j])), True, self.cor.red)
                else:
                    if(self.M_n[i][j]==0): text_cell =''
                    else: text_cell=str(int(self.M_n[i][j]))

                    t=f.render(text_cell, True, self.cor.white)
                self.BG.blit(t,[x,y])



    def draw_tela_inicial(self):
        self.BG.fill(cor.black)
        for i in range(2,6):
            if(i*4<10):
                label="%2d  x %2d"%(4*i,4*i)
            else:
                label="%2d x %2d"%(4*i,4*i)
            self.Button(label,i)
        pg.display.update()
        #sleep(5)
    def draw(self,):
        self.BG.fill(cor.black)
        self.dr_grid()
        #if(self.print_matriz):
        self.draw_num()
            #self.print_matriz=0
        #self.Button()
        #self.BG.blit(self.T.B,self.T.b_pos)
        pg.display.update()
    def main(self):
        pg.init()
        pg.font.init()
        self.BG=pg.display.set_mode(Default.window)
        pg.display.set_caption(Default.title)
        self.run=True
        #matriz
        self.bomba()
        self.Num_redor()
        print(self.M_b,'\n\n',self.M_n)
        #self.draw()
        self.settings()
        self.Quit_game()
    def settings(self):
        while(self.run):
            for event in pg.event.get():
                if event.type == QUIT:
                    self.run=False
                else:
                    #self.draw_tela_inicial()
                    #sleep(3)
                    self.draw()

theapp=APP()
theapp.main()
```

```
[[0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0. 0. 1. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
```

```
[0. 0. 0. 0. 0. 0. 0. 1. 1. 0.]
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
[0. 0. 0. 0. 0. 0. 0. 0. 1. 0.]
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
[0. 1. 0. 0. 0. 0. 0. 0. 1. 0.]
[0. 0. 0. 0. 1. 0. 1. 0. 1. 0.]
[0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]]

[[ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0. -1.  1.  0.  0.  1. -1.  1.  0.  0.]
 [ 0.  1.  1.  0.  0.  1.  2.  3.  2.  0.]
 [ 0.  0.  0.  0.  0.  0.  1. -1. -1.  0.]
 [ 0.  0.  0.  0.  0.  0.  1.  3.  3.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  1. -1.  0.]
 [ 0.  1.  1.  0.  0.  0.  0.  2.  2.  0.]
 [ 0. -1.  1.  1.  1.  2.  1.  3. -1.  0.]
 [ 0.  1.  1.  1. -1.  2. -1.  3. -1.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]]
```

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

Testando função geradora das matrizes

Solução:

- usar duas matrizes zeradas de mesma dimensão, em uma mudar os valores de 0 para 1, em posições aleatórias. O valor 1 equivale a ter bomba no local.
- Usando a primeira matriz como referencia, ver posição a posição onde tem bomba e gerar números que são resultado da soma dos valores ao redor da posição, ou deja contar quantas bombas existem ao redor da posição.
- Outra estratégia foi gerar uma matriz maior e trabalhar com a sub-matriz de dentro para não haver problema na verificação ao redor das bordas.

Soma ao redor:

$$Soma = \sum_{i=-1}^{1} \sum_{j=-1}^{1} A_{m+i,n+j}$$

representação:

$$\begin{matrix} A_{(m-1)(n-1)} & A_{(m-1)(n-0)} & A_{(m-1)(n+1)} \\ A_{(m-0)(n-1)} & A_{(m)(n)} & A_{(m-0)(n+1)} \\ A_{(m+1)(n-1)} & A_{(m+1)(n+0)} & A_{(m+1)(n+1)} \end{matrix}$$

Exemplo de matriz zerada criada, mas vamos trabalhar apenas com os valores de $A_{1,1}$ até $A_{(n-1)(n-1)}$.

$$A = \begin{matrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{matrix}$$

In [20]:

```
#Testando funções geradores das matrizes

def bomba(M_b,Qb):
    i=0
    V=[]
    while(i<Qb):
        x=randint(1,8)
        y=randint(1,8)
        if [x,y] not in V:
            V.append([x,y])
            i+=1
    for j in V:
        M_b[j[0]][j[1]]=1
    return M_b
def Ver_redor(Mb,k,l):# ver em Mb
    soma=0
    for i in range(-1,2):
        for j in range(-1,2):
            #if(Mb[k+i][l+j]==1):#Mn[k+i][l+j]==1
            soma+=Mb[k+i][l+j]
    return soma

def Num_redor(M_n,M_b):#M_bomb matriz de comparação

    aux=len(M_n[0])
    aux2=len(M_n[1])

    for i in range(1,aux-1):
        for j in range(1,aux2-1):
            if(M_b[i][j]==0):
                M_n[i][j] = Ver_redor(M_b,i,j)
            else:
                M_n[i][j] =-1
    return M_n
n=8
Mn=np.zeros([n+2,n+2])
Mb=np.zeros([n+2,n+2])
Mb=bomba(Mb,10)
Mn=Num_redor(Mn,Mb)
print(Mb,'\n',Mn)
```

```
[[0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 1. 0. 1. 0. 0. 0. 0.]
 [0. 0. 1. 0. 0. 1. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 1. 0. 0. 1. 0. 0. 0. 0.]
 [0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 1. 1. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0. 0. 0. 0.]]
 [[ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  1.  2. -1.  3. -1.  2.  0.  0.  0.]
 [ 0.  1. -1.  2.  3. -1.  2.  0.  0.  0.]
 [ 0.  2.  2.  1.  1.  1.  1.  0.  0.  0.]
 [ 0. -1.  1.  0.  0.  0.  0.  0.  0.  0.]
 [ 0.  2.  2.  1.  1.  1.  1.  0.  0.  0.]
 [ 0.  2. -1.  1.  1. -1.  1.  0.  0.  0.]
 [ 0. -1.  4.  3.  2.  1.  1.  0.  0.  0.]
```

```
 [ 0.  2. -1. -1.  1.  0.  0.  0.  0.  0.]
 [ 0.  0.  0.  0.  0.  0.  0.  0.  0.  0.]]
```

In [2]:

```
for i in range(-1,1): #tem que usar de -1 a 2
    print(i)
```

```
-1
0
```

In [ ]:

In [ ]: