

TP3 - Projeto de bloco



Esse é o terceiro teste de performance da disciplina Projeto de Bloco.



Aluno: Bruno Fernandes

Email: bruno.fernandes@al.infnet.edu.br


Prof: Prof. Alcione Dolavale

Relatório:

Para acessar o repositório deste trabalho clique abaixo:

Bwendel26/PB_INFNET_Python_redes_SO

Um software cliente-servidor em Python que explore conceitos de arquitetura de redes, arquitetura de computadores e/ou de sistemas operacionais, acompanhado de relatório explicativo. Este

 https://github.com/Bwendel26/PB_INFNET_Python_redes_SO/tree/master/tp2



No TP3 o objetivo foi continuar a montagem de um pequeno sistema desktop de monitoramento dos componentes principais do computador. Foi codificado uma série de algoritmos, na linguagem Python, que utilizaram bibliotecas da linguagem para monitoramento do computador e para criar a interface que mostra os gráficos e informações a serem apresentadas pelo usuário.

Ferramentas utilizadas:

Linguagem de programação: Python (versão 3.8)

IDE: PyCharm Community Edition 64 bits

Bibliotecas utilizadas:

- Pygame

Pygame Front Page - pygame v2.0.0.dev25 documentation

Basic information about pygame: what it is, who is involved, and where to find it. Steps needed to compile pygame on several platforms. Also help on finding and installing prebuilt binaries for your


 <https://www.pygame.org/docs/>



- psutil

psutil documentation - psutil 5.7.4 documentation

psutil (python system and process utilities) is a cross-platform library for retrieving information on running processes and system utilization (CPU, memory, disks, network, sensors) in Python. It is useful mainly for system monitoring, profiling, limiting process resources and the management of running

 <https://psutil.readthedocs.io/en/latest/>

- platform

platform - Access to underlying platform's identifying data - Python 3.9.0 documentation

Source code: Lib/platform.py Note Specific platforms listed alphabetically, with Linux included in the Unix section. Queries the given executable (defaults to the Python interpreter binary) for various architecture information. Returns a tuple which contain information about the bit architecture and the

 <https://docs.python.org/3/library/platform.html>

- cpuinfo

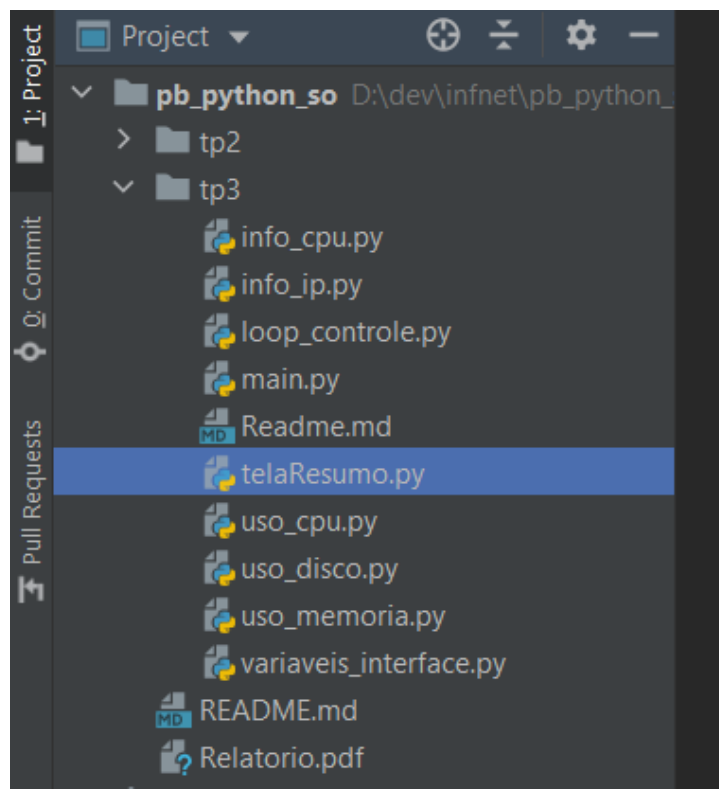
workhorsy/py-cpuinfo

Py-cpuinfo gets CPU info with pure Python. Py-cpuinfo should work without any extra programs or libraries, beyond what your OS provides. It does not require any compilation(C/C++,

 <https://github.com/workhorsy/py-cpuinfo>



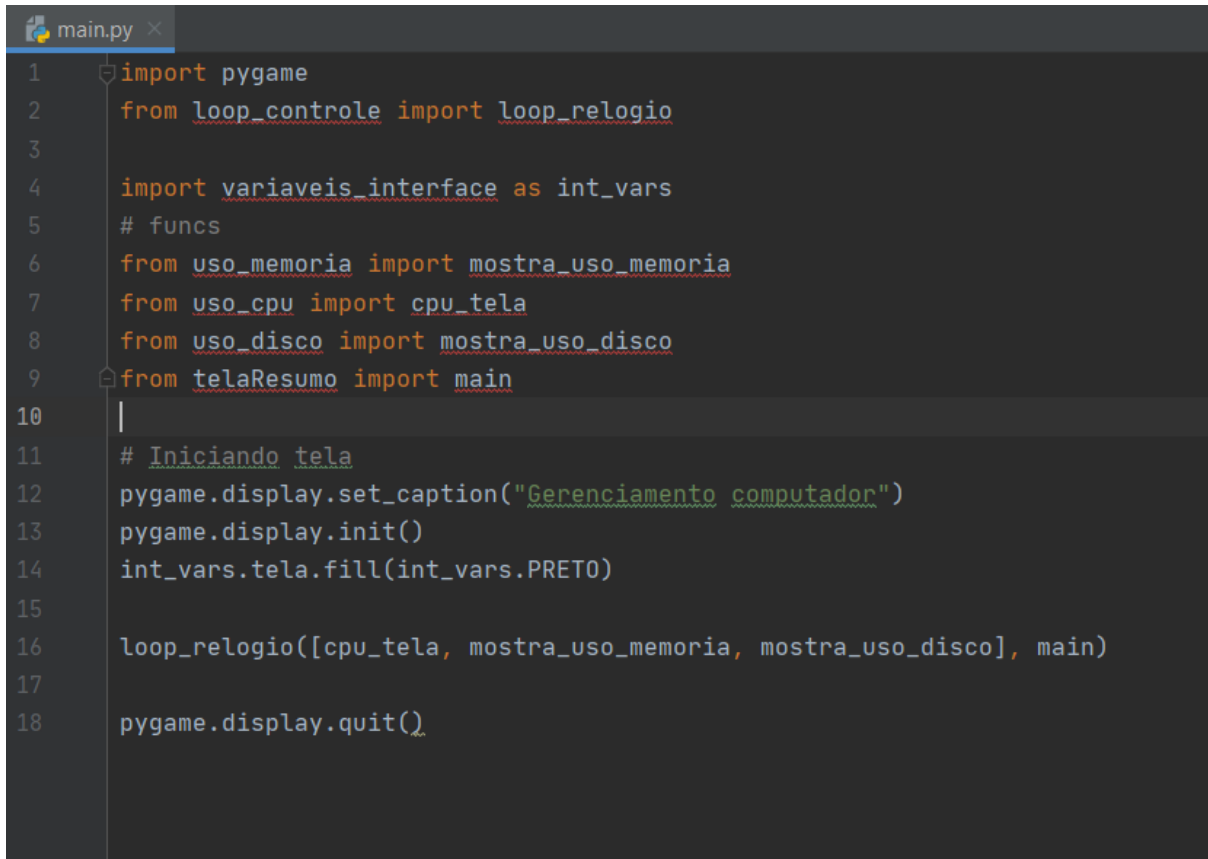
A estrutura do projeto do TP3 foi a seguinte:



Como podemos ver os arquivos foram divididos um para cada funcionalidade, e o arquivo main.py (principal) chama todas as funções que executam o programa por

um todo. Como podemos ver temos funções para cada chamada (uso_cpu.py, uso_memoria.py, uso_disco, info_cpu, etc...).

Temos o script main.py que faz a chamada das funções e variáveis dentro dos demais arquivos e consequentemente executa:



```
1 import pygame
2 from loop_controle import loop_relogio
3
4 import variaveis_interface as int_vars
5 # funcs
6 from uso_memoria import mostra_uso_memoria
7 from uso_cpu import cpu_tela
8 from uso_disco import mostra_uso_disco
9 from telaResumo import main
10
11 # Iniciando tela
12 pygame.display.set_caption("Gerenciamento computador")
13 pygame.display.init()
14 int_vars.tela.fill(int_vars.PRETO)
15
16 loop_relogio([cpu_tela, mostra_uso_memoria, mostra_uso_disco], main)
17
18 pygame.display.quit()
```

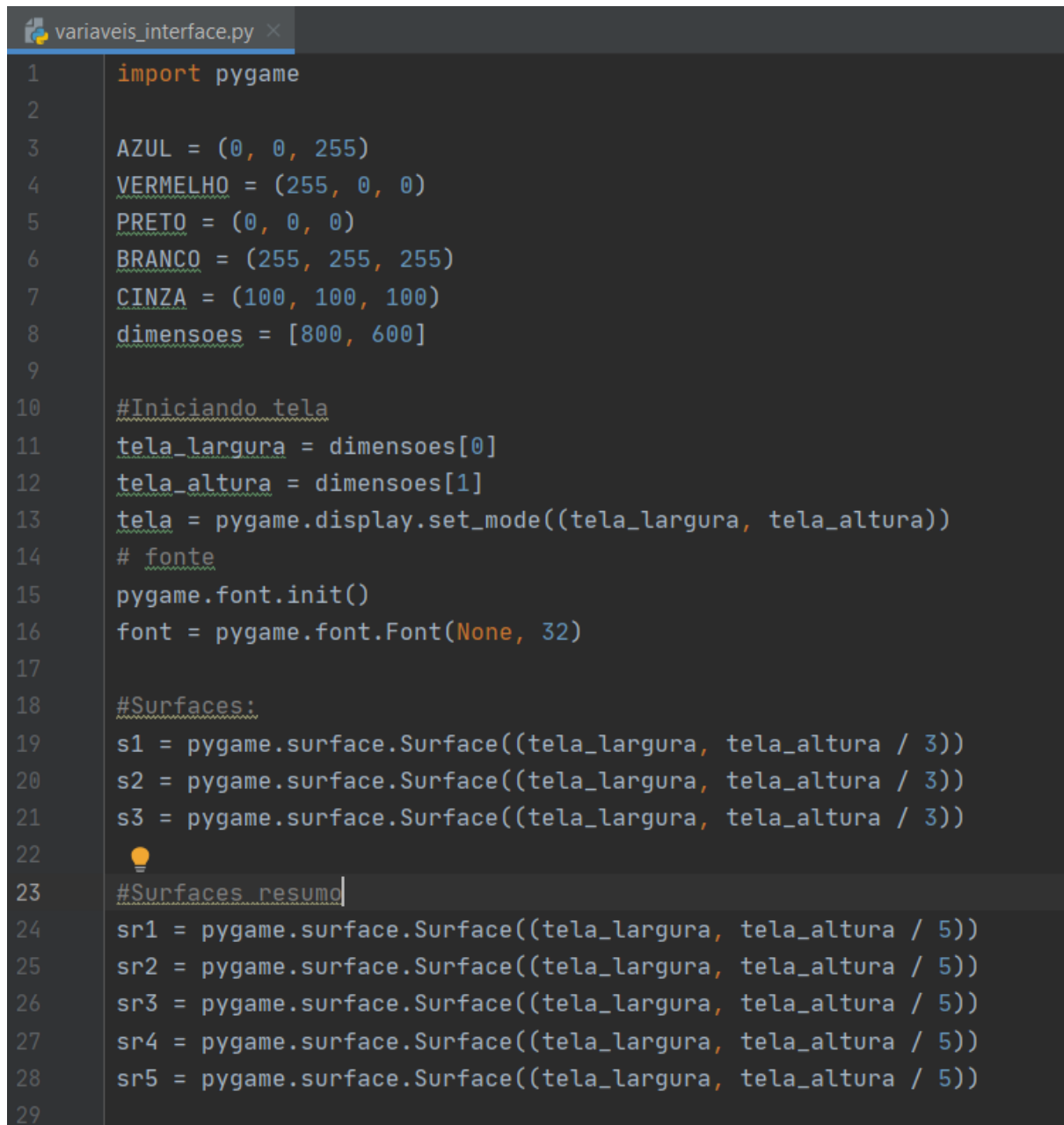
Logo no início fazemos o import da função loop_relogio, do arquivo loop_controle.py, que faz o controle das chamadas das funções dentro de um timer:

```
loop_controle.py x
2
3 def loop_relogio(funcoes, tela_resumo):
4     # cria relógio
5     clock = pygame.time.Clock()
6     cont = 60
7
8     terminou = False
9     func_atual = 0 #controle de funcao apresentada
10
11     while not terminou:
12         # Checar os eventos do mouse aqui:
13         for event in pygame.event.get():
14             if event.type == pygame.QUIT:
15                 terminou = True
16
17             if event.type == pygame.KEYDOWN:
18                 if event.key == pygame.K_RIGHT:
19                     func_atual += 1
20                     if func_atual >= len(funcoes):
21                         func_atual = 0
22                     funcoes[func_atual]()
23                 if event.key == pygame.K_LEFT:
24                     func_atual -= 1
25                     if func_atual < 0:
26                         func_atual = len(funcoes) - 1
27                     funcoes[func_atual]()
28
29                 # CHAMADA DA TELA DE RESUMO:
30                 if event.key == pygame.K_SPACE:
31                     func_atual = 10
32                     if func_atual == 10:
33                         tela_resumo()
34
35         # atualiza desenho
36         if cont == 60:
37             if func_atual == 10:
38                 tela_resumo()
39             else:
40                 funcoes[func_atual]()
41             # for i in range(len(funcoes)):
42             #     funcoes[i]()
43
44             cont = 0
45         # Atualiza o desenho na tela
46         pygame.display.update()
47         # 60 frames por segundo
48         clock.tick(60)
49         cont += 1
50
51     loop_relogio()
```

Podemos observar que essa função faz todo o controle do tempo, chamada das funções e controle de cada evento do pygame, seja no momento de sair do programa ou ao utilizar setas ou barra de espaço para fazer a navegação dentro as telas que fazem o monitoramento do computador dentro do software.

A função `loop_relogio` recebe como parâmetros uma lista de funções para serem chamadas e uma função separada para chamada da tela de resumo.

Dentro do código temos um arquivo Python específico para as variáveis gerais utilizadas no sistema, segue a imagem:



```
1 import pygame
2
3 AZUL = (0, 0, 255)
4 VERMELHO = (255, 0, 0)
5 PRETO = (0, 0, 0)
6 BRANCO = (255, 255, 255)
7 CINZA = (100, 100, 100)
8 dimensoes = [800, 600]
9
10 #Iniciando tela
11 tela_largura = dimensoes[0]
12 tela_altura = dimensoes[1]
13 tela = pygame.display.set_mode((tela_largura, tela_altura))
14 # fonte
15 pygame.font.init()
16 font = pygame.font.Font(None, 32)
17
18 #Surfaces:
19 s1 = pygame.surface.Surface((tela_largura, tela_altura / 3))
20 s2 = pygame.surface.Surface((tela_largura, tela_altura / 3))
21 s3 = pygame.surface.Surface((tela_largura, tela_altura / 3))
22
23 #Surfaces_resumo
24 sr1 = pygame.surface.Surface((tela_largura, tela_altura / 5))
25 sr2 = pygame.surface.Surface((tela_largura, tela_altura / 5))
26 sr3 = pygame.surface.Surface((tela_largura, tela_altura / 5))
27 sr4 = pygame.surface.Surface((tela_largura, tela_altura / 5))
28 sr5 = pygame.surface.Surface((tela_largura, tela_altura / 5))
29
```

Dentro do sistemas temos um arquivo para mostrar de cada funcionalidade;

- Temos os arquivos de controle da cpu:

```

1 import psutil
2 import cpuinfo
3 import pygame
4 from info_cpu import mostra_info_cpu
5 import variaveis_interface as int_vars
6
7 #vars
8 surface1 = int_vars.s1
9 surface2 = int_vars.s2
10 lista_percentual_cpus = psutil.cpu_percent(1, percpu=True)
11
12 #funçs
13 def percentual_cpu():
14     """
15     Função que retorna o percentual de uso da CPU
16     do computador.
17     :param tempo: em segundos para retornar o percentual.
18     :return: float percentual_usado
19     """
20     percentual_usado = psutil.cpu_percent(interval=0)
21
22     return percentual_usado
23
24
25 def mostra_uso_cpu():
26     capacidade = percentual_cpu()
27     larg = int_vars.tela_largura - 2*20
28     int_vars.tela.fill(int_vars.PRETO)
29     pygame.draw.rect(surface1, int_vars.AZUL, (20, 50, larg, 70))
30     larg = larg*capacidade/100
31     pygame.draw.rect(surface1, int_vars.VERMELHO, (20, 50, larg, 70))
32     texto_barra = "Uso de CPU (Total: " + str(capacidade) + "%):"
33     text = int_vars.font.render(texto_barra, 1, int_vars.BRANCO)
34     int_vars.tela.blit(surface1, (0, 0)) # setando divisao tela
35     int_vars.tela.blit(text, (20, 10))
36
37
38 def mostra_uso_de_cada_cpu(s, l_cpu_percent):
39     s.fill(int_vars.CINZA)
40     int_vars.tela.blit(s, (0, int_vars.tela_altura / 3))
41
42     num_cpu = len(l_cpu_percent)
43
44     percentual_cpu()

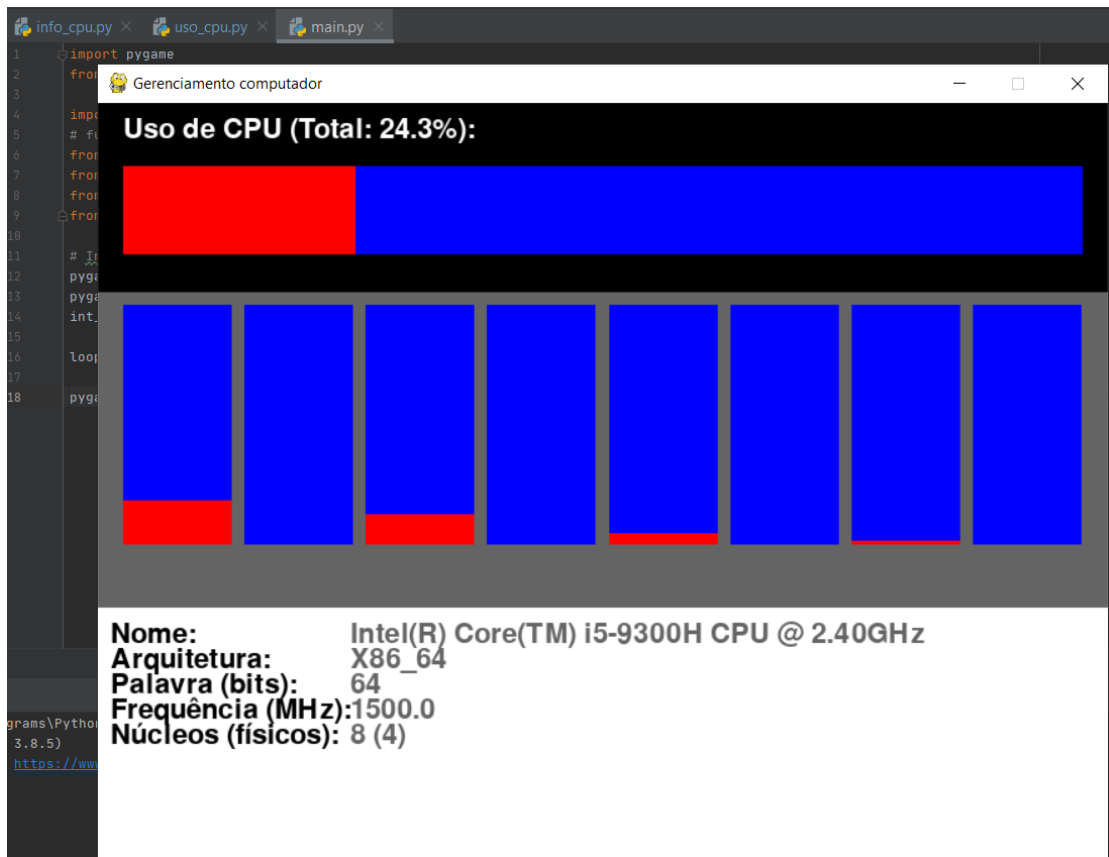
```

Nesse arquivo temos as funções que mostram o uso da cpu e uma para cada um de seus núcleos, fazendo a plotagem com pygame, em barras que mostram o total de uso.

Logo em seguida temos um arquivo para capturar e apresentar as informações sobre a cpu e sistema no geral.

```
info_cpu.py x uso_cpu.py x main.py x
4 import pygame
5 import cpuinfo
6
7 info = cpuinfo.get_cpu_info()
8 surface3 = int_vars.s3
9
10 def mostra_texto(s1, nome, chave, pos_y):
11     text = int_vars.font.render(nome, True, int_vars.PRETO)
12     s1.blit(text, (10, pos_y))
13     if chave == "freq":
14         s = str(round(psutil.cpu_freq().current, 2))
15     elif chave == "nucleos":
16         s = str(psutil.cpu_count())
17         s = s + " (" + str(psutil.cpu_count(logical=False)) + ")"
18     else:
19         s = str(info[chave])
20     text = int_vars.font.render(s, True, int_vars.CINZA)
21     s1.blit(text, (200, pos_y))
22
23
24 def mostra_info_cpu():
25     # Mostra as informações de CPU escolhidas:
26     surface3.fill(int_vars.BRANCO)
27     mostra_texto(surface3, "Nome:", "brand_raw", 10)
28     mostra_texto(surface3, "Arquitetura:", "arch", 30)
29     mostra_texto(surface3, "Palavra (bits):", "bits", 50)
30     mostra_texto(surface3, "Frequência (MHz):", "freq", 70)
31     mostra_texto(surface3, "Núcleos (físicos):", "nucleos", 90)
32     int_vars.tela.blit(surface3, (0, 2 * int_vars.tela_altura / 3))
```

E o resultado da chamada da função main quando chamamos a tela de cpu é o seguinte:



- Temos, também, o arquivo de controle da memória principal (RAM):

```

1 import psutil
2 import pygame
3 import variaveis_interface as int_vars
4 #variaveis
5 memoria = psutil.virtual_memory()
6
7 #funcs
8 def percentual_memoria():
9     """
10     Função que retorna o percentual de memória RAM
11     utilizado por um computador.
12     :return: float percentual-usado
13     """
14
15     mem = psutil.virtual_memory().percent
16     return mem
17
18 surface1 = int_vars.s1
19
20 def mostra_uso_memoria():
21     mem = percentual_memoria()
22     larg = int_vars.tela_largura - 2 * 20
23     int_vars.tela.fill(int_vars.PRETO)
24     pygame.draw.rect(surface1, int_vars.AZUL, (20, 50, larg, 70))
25     larg = larg * mem / 100
26     pygame.draw.rect(surface1, int_vars.VERMELHO, (20, 50, larg, 70))
27     texto_barra = "Uso de Memória (Total: " + str(mem) + "%):"
28     text = int_vars.font.render(texto_barra, 1, int_vars.BRANCO)
29     int_vars.tela.blit(surface1, (0, 0)) # setando divisao tela
30     int_vars.tela.blit(text, (20, 10))

```

temos uma função que extrai a informação sobre a porcentagem de uso da memória e a segunda função usa a biblioteca pygame para mostrar a barra de uso da memória.

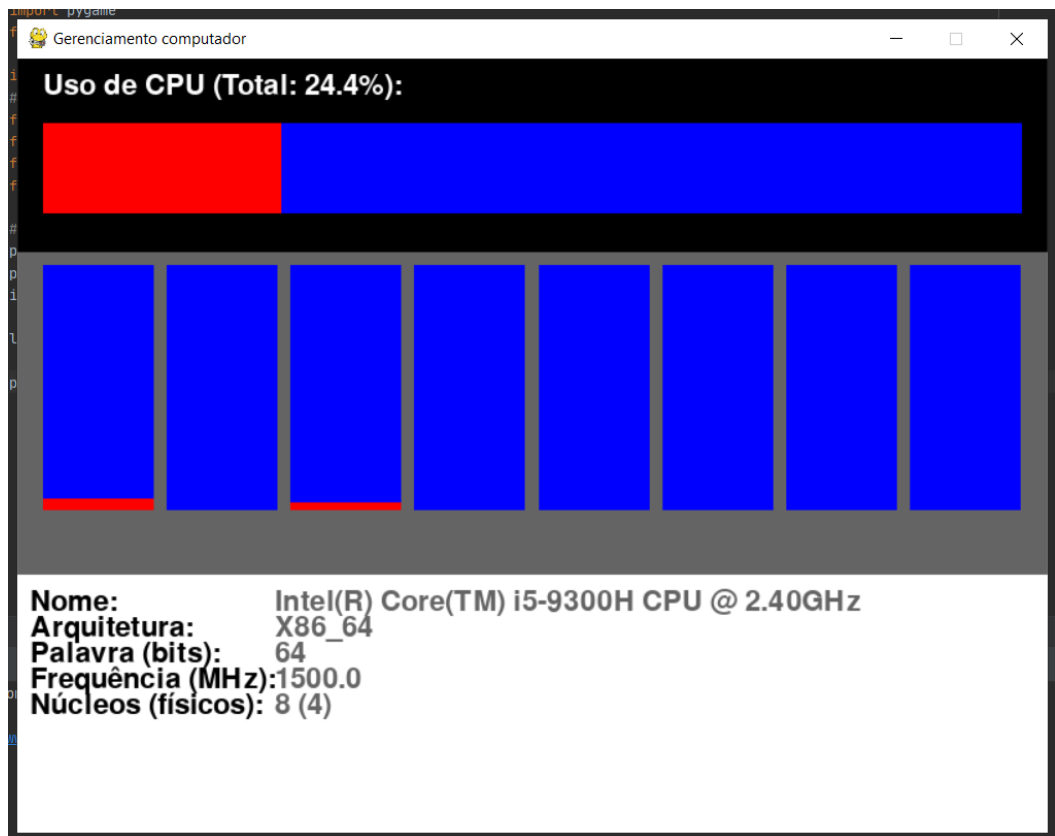
- Função que mostra os discos e uso do disco em percentual:

```
main.py x uso_disco.py x
1 import psutil
2 import pygame
3 import variaveis_interface as int_vars
4
5 surface1 = int_vars.s1
6
7 def mostra_uso_disco():
8     # disk = str(disk)
9     disco = psutil.disk_usage('/')
10    larg = int_vars.tela_largura - 2 * 20
11    int_vars.tela.fill(int_vars.PRETO)
12    pygame.draw.rect(surface1, int_vars.AZUL, (20, 50, larg, 70))
13    larg = larg * disco.percent / 100
14    pygame.draw.rect(surface1, int_vars.VERMELHO, (20, 50, larg, 70))
15    total = round(disco.total / (1024 * 1024 * 1024), 2)
16    texto_barra = "Uso de Disco: (Total: " + str(total) + "GB):"
17    text = int_vars.font.render(texto_barra, 1, int_vars.BRANCO)
18    int_vars.tela.blit(surface1, (0, 0)) #setando divisao tela
19    int_vars.tela.blit(text, (20, 10)) # setando posicao do text
20
21 # print(psutil.disk_partitions())
22 # Modificar para iterar sobre cada disco do pc
23 # e consequentemente criar uma surface para cada um
```

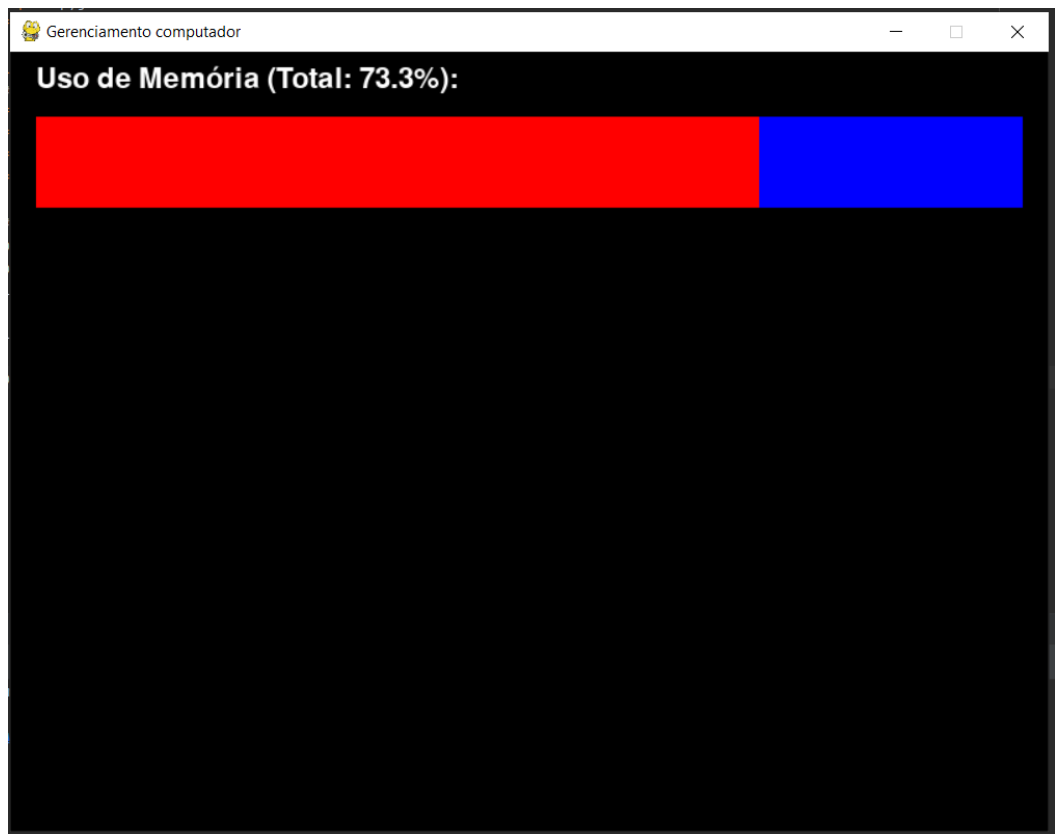
Temos uma função que pega, diretamente da biblioteca psutil, o percentual da memória utilizada e após mostra na interface do pygame.

Como resultado final temos a seguinte interface controlada por pelas setas laterais:

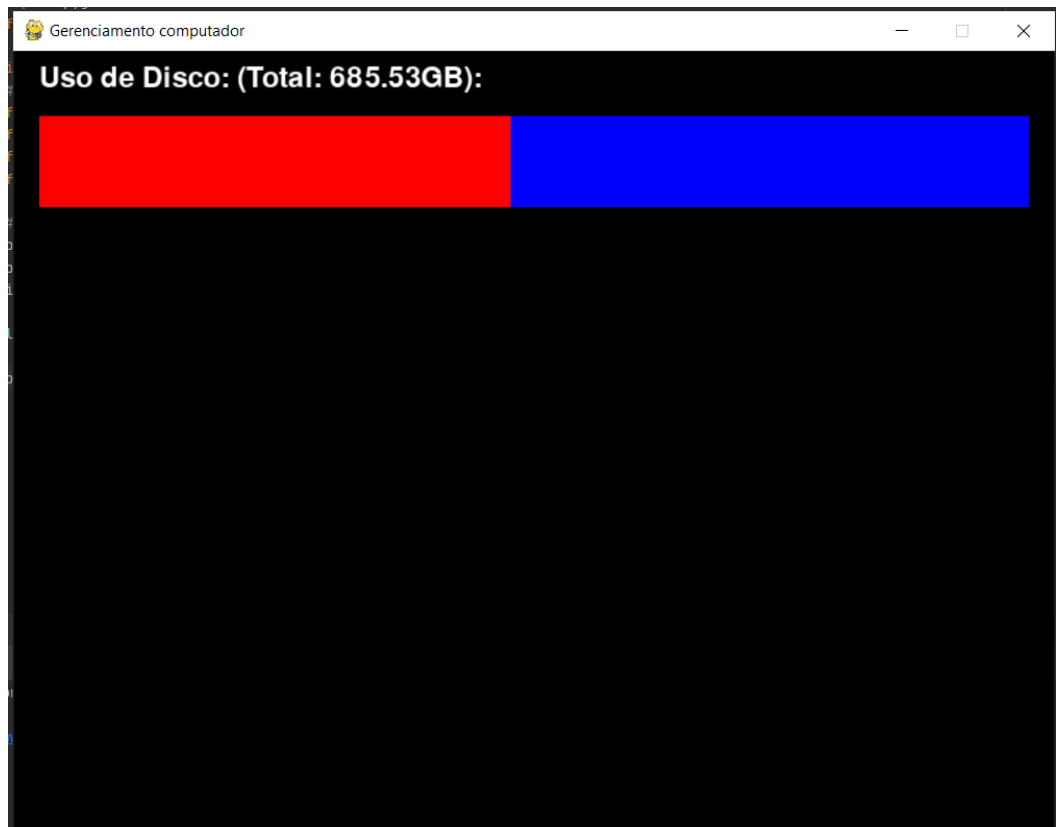
Tela 1



Tela 2



Tela 3



Tela 4 (Resumo)

