

# INE5404

# Interface Gráfica com o Usuário (GUI)

Prof. Jônata Tyska  
Prof. Mateus Grellert



UNIVERSIDADE FEDERAL  
DE SANTA CATARINA

# Parte 1:

# Fundamentos

# Elementos de um Sistema de Software

---

- **Modelo:** a forma como representamos nosso problema e a lógica para resolvê-lo
- **Interface:** lida com a interação com o mundo externo. Quando o mundo externo é um usuário, estamos falando de uma **interface com o usuário**

# Tipos de Interface

---

**Baseada em controle:** a aplicação dita que informação é necessária e quando ela deve ser recolhida. **Exemplo:** interfaces baseadas em texto (command line interfaces - CLI)

**Baseada em eventos:** a aplicação espera algum evento acontecer no ambiente. Quando o evento ocorre (dispara), a aplicação responde e espera o próximo

# Tipos de Interface

---

**Baseada em controle:** a aplicação dita que informação é necessária e quando ela deve ser recolhida. **Exemplo:** interfaces baseadas em texto (command line interfaces - CLI)

**Baseada em eventos:** a aplicação espera algum evento acontecer no ambiente. Quando o evento ocorre (dispara), a aplicação responde e espera o próximo

Aplicações com uma interface gráfica baseada em janelas - **Graphical User Interfaces ou GUIs** - são comumente **orientada a eventos**

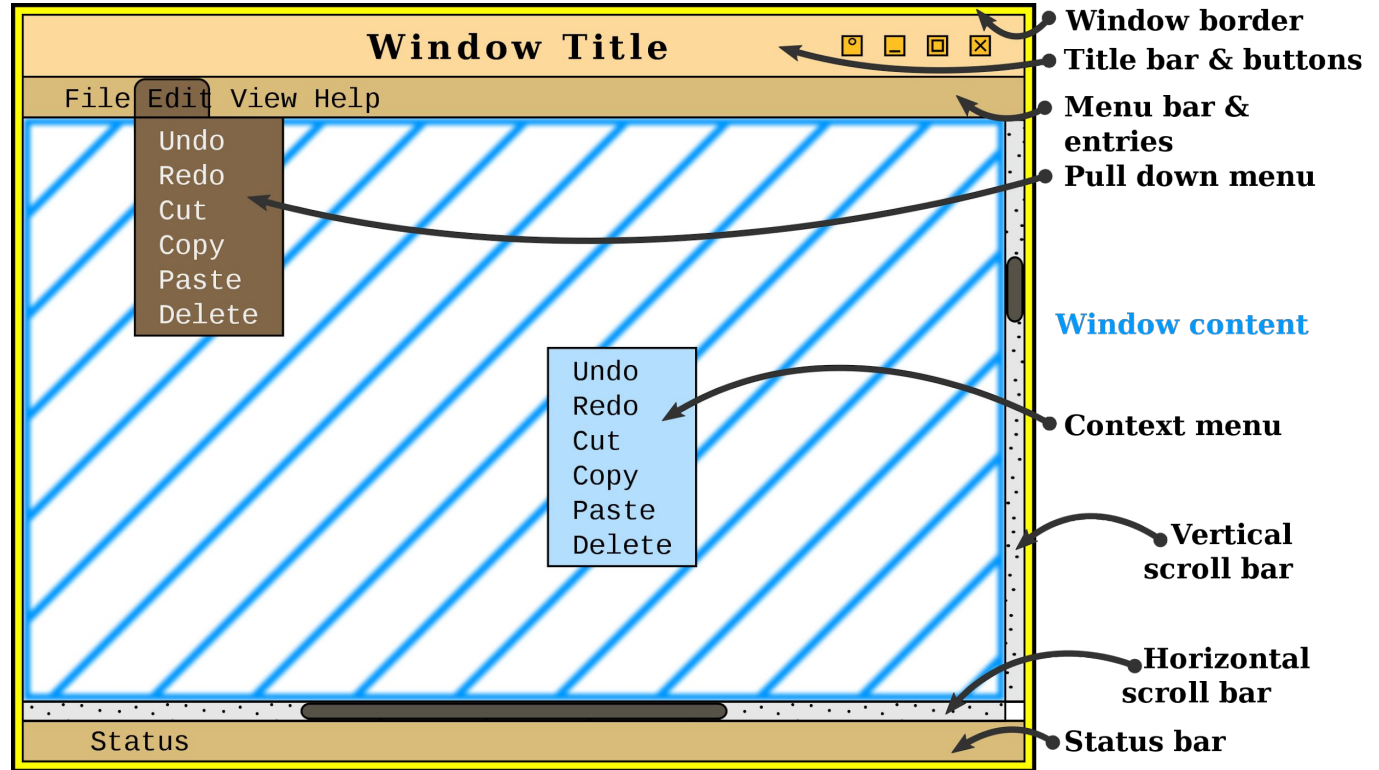
# WIMP

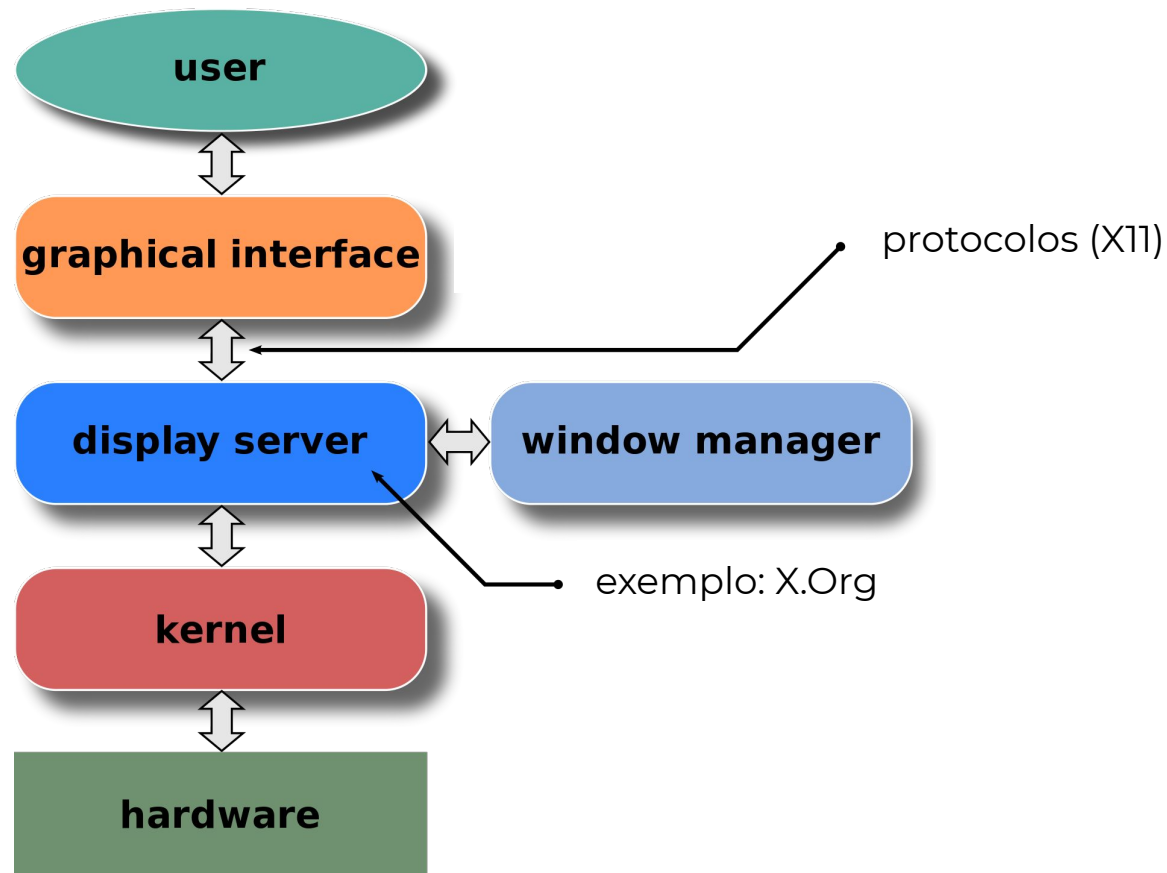
Windows

Icons

Menus

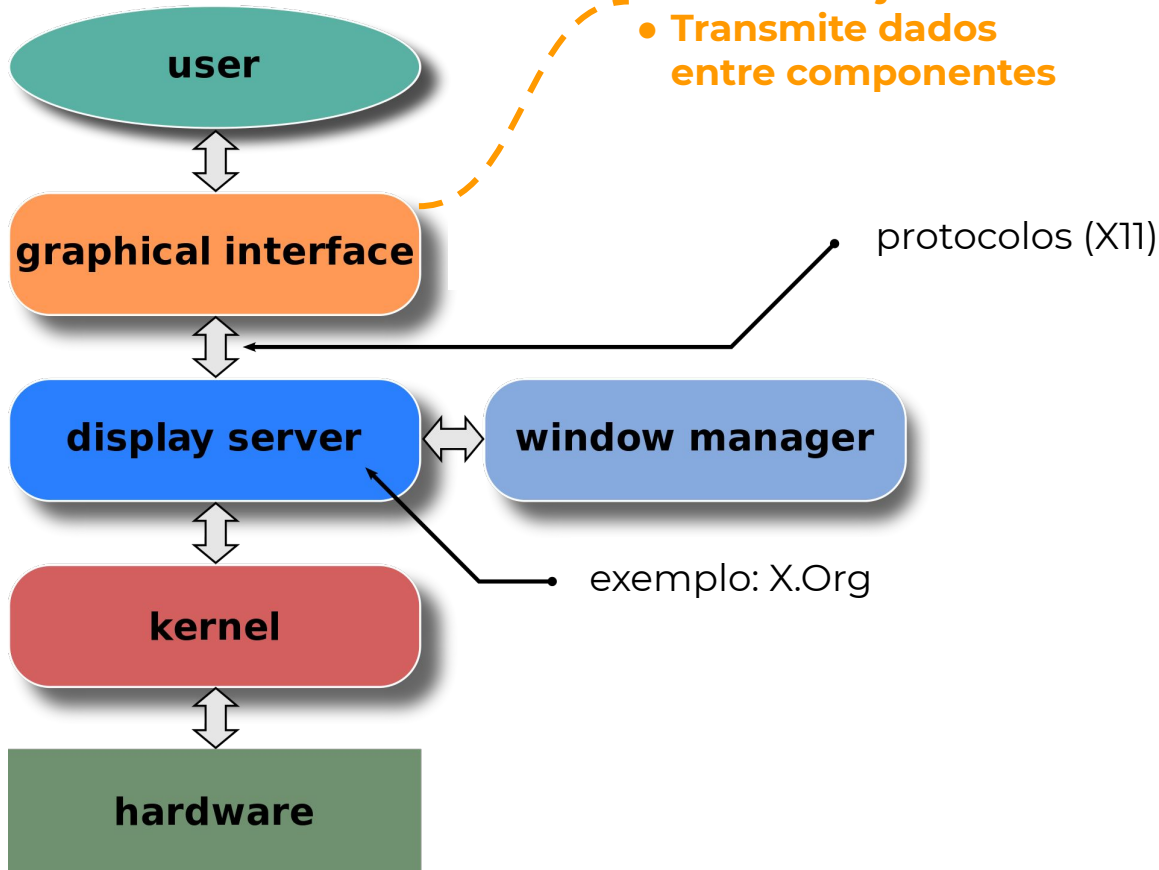
Pointers





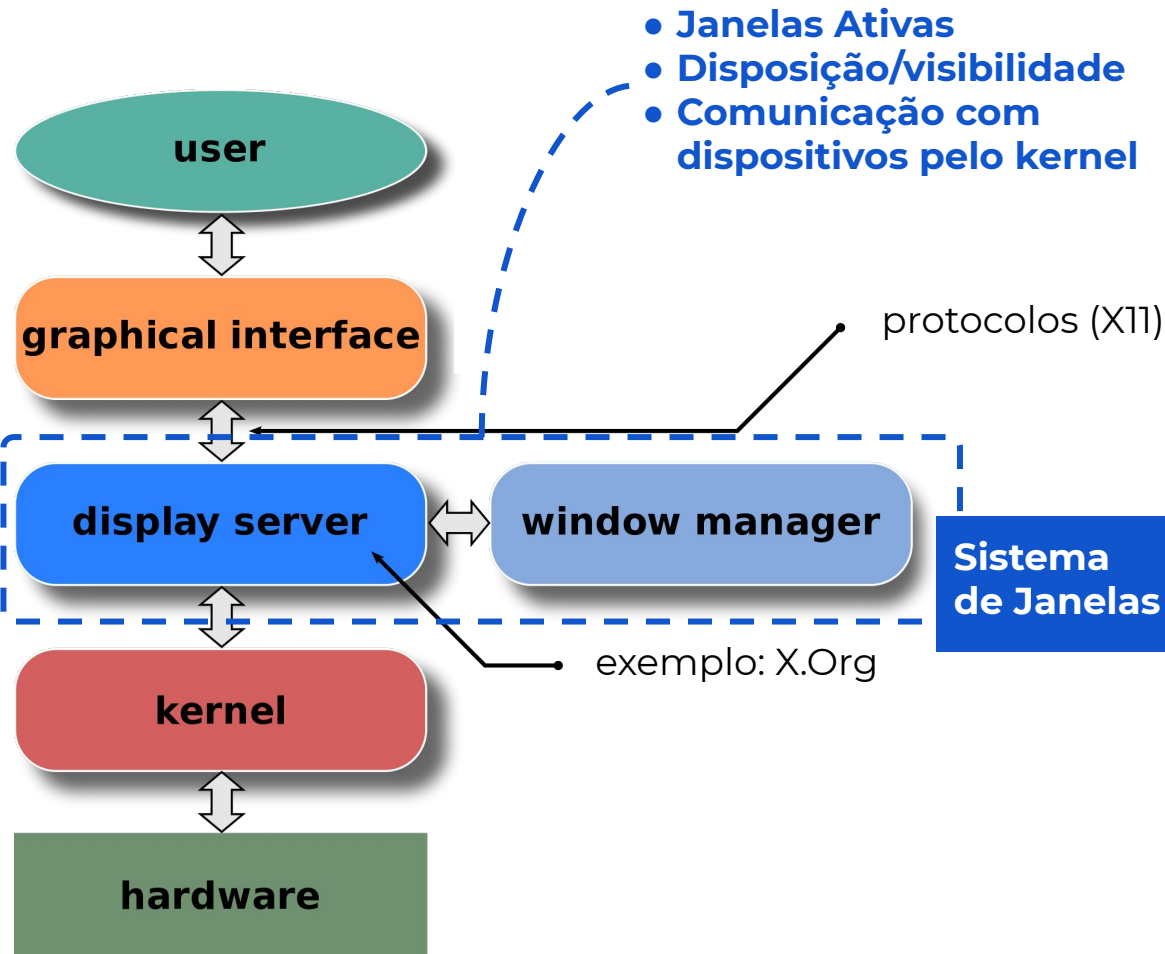
# Camadas de um Sistema com GUIs

- Captura eventos
- Modela as janelas
- Transmite dados entre componentes



# Camadas de um Sistema com GUIs

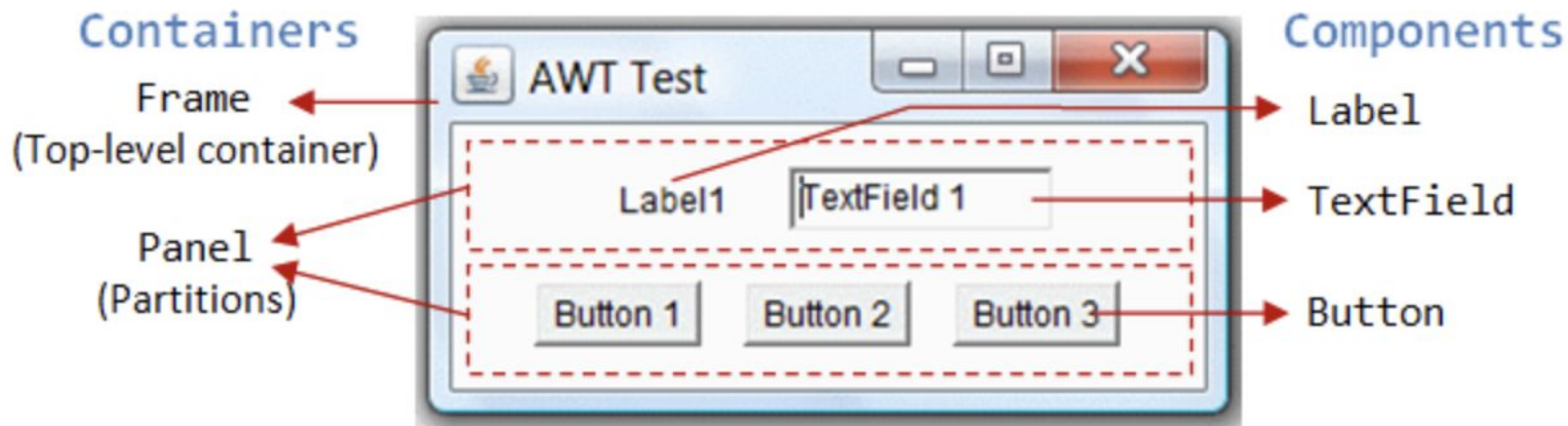




# Camadas de um Sistema com GUIs

# Elementos de UI

---

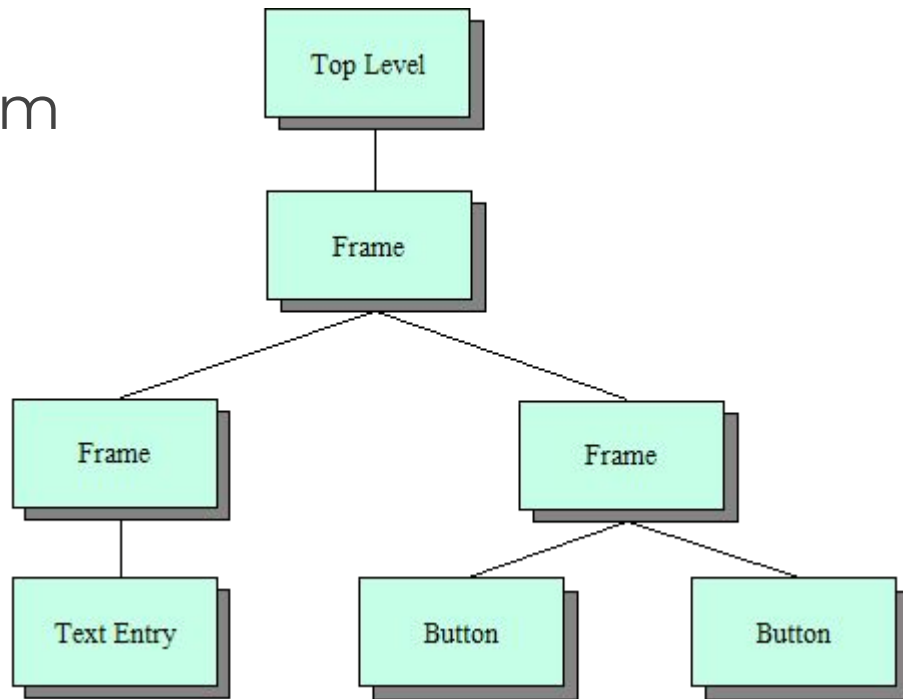


O estilo dos widgets vai depender de que **Widget Toolkit** estamos utilizando. O exemplo da figura é da biblioteca **AWT** do Java. Outros exemplos são **Tk** e **Qt**.

# Desenvolvendo GUIs com Classes de UI

---

Uma forma clássica de desenvolver GUIs é pensar em uma árvore de **containers** e **elementos de UI**

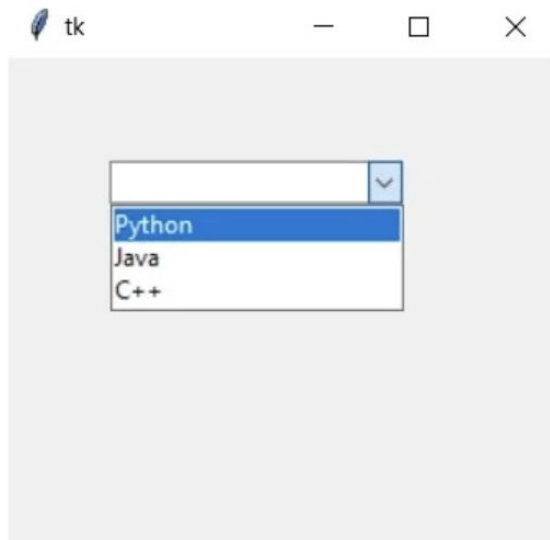


# Exemplo de dois Widget Toolkits - Tkinter e PyQt

---



**PyQt**

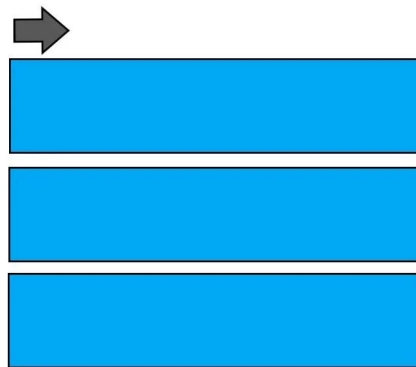


**Tkinter**

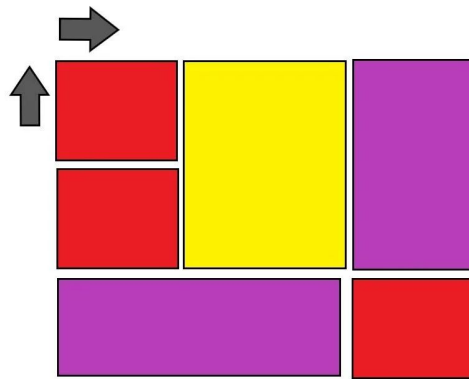
# Layouts

Além dos Widgets, as bibliotecas de GUI fornecem algum tipo de Layout Manager, que define como os elementos ficarão dispostos na tela

As bibliotecas para desenvolvimento de GUI podem oferecer mais de um **Layout Manager**. Em CSS, por exemplo, temos layouts unidimensionais (flex) e bidimensionais (grid)



Flexbox is one-dimensional



CSS Grid is two-dimensional

# **Programação Orientada a Eventos (POE)**

# Eventos

---

As GUIs são baseadas em **eventos**: o código não executa ao ser carregado, mas quando um evento é disparado



## Exemplo:

Elemento de UI com o qual o usuário pode interagir

# Eventos

---

As GUIs são baseadas em **eventos**: o código não executa ao ser carregado, mas quando um evento é disparado



## Exemplo:

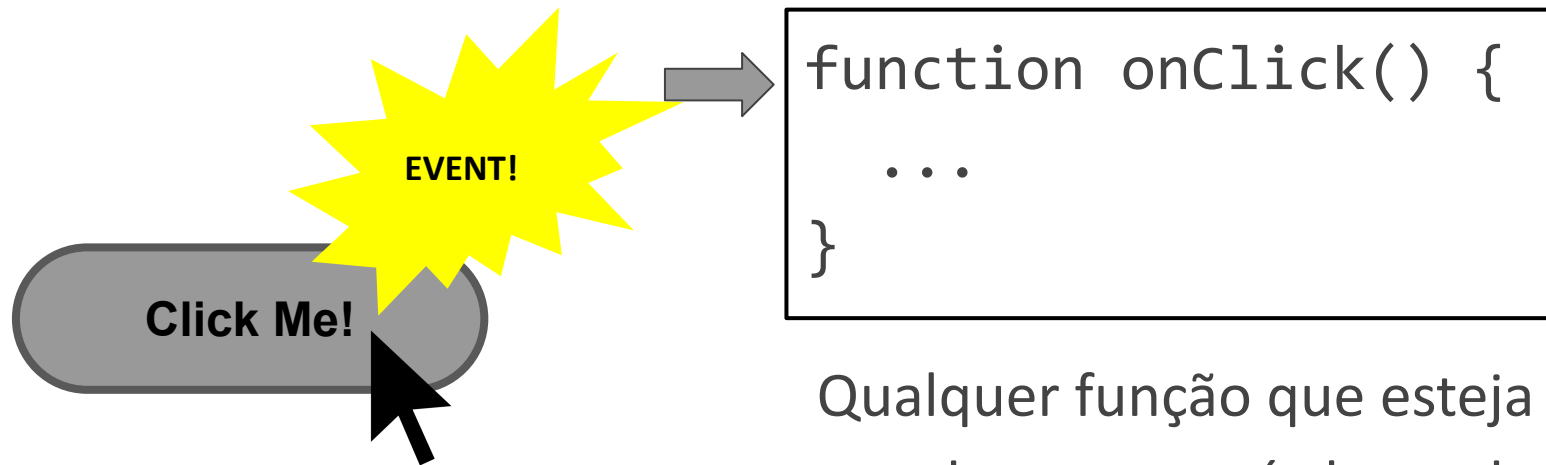
Botão **emite** um evento ao ser clicado



# Eventos

---

As GUIs são baseadas em **eventos**: o código não executa ao ser carregado, mas quando um evento é disparado



Qualquer função que esteja  
escutando o evento é chamada  
(**event handlers**)

# Eventos Típicos em GUIs

---

- **On load:** quando a página/aplicação é carregada
- **On click:** quando o elemento é clicado
- **On mouse up:** quando o botão do mouse é solto
- **On key pressed:** quando alguma tecla é pressionada
- **On mouse over:** quando o cursor do mouse passa pelo elemento

**Até a  
próxima!**

# Parte 2:

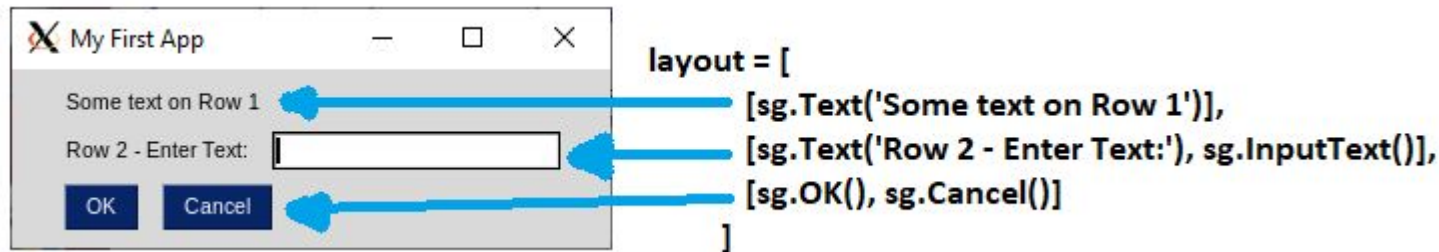
# Um Primeiro

# Exemplo

# PySimpleGUI

---

- Vamos usar PySimpleGUI por ser uma das bibliotecas mais amigáveis para interfaces gráficas simples
- É uma biblioteca que implementa diferentes **Widget Toolkits** (padrão é Tk)
- Possui os componentes mais básicos e um layout **baseado em grade** (grid)



# Vamos implementar essa interface

— — —

Calcule seu IMC

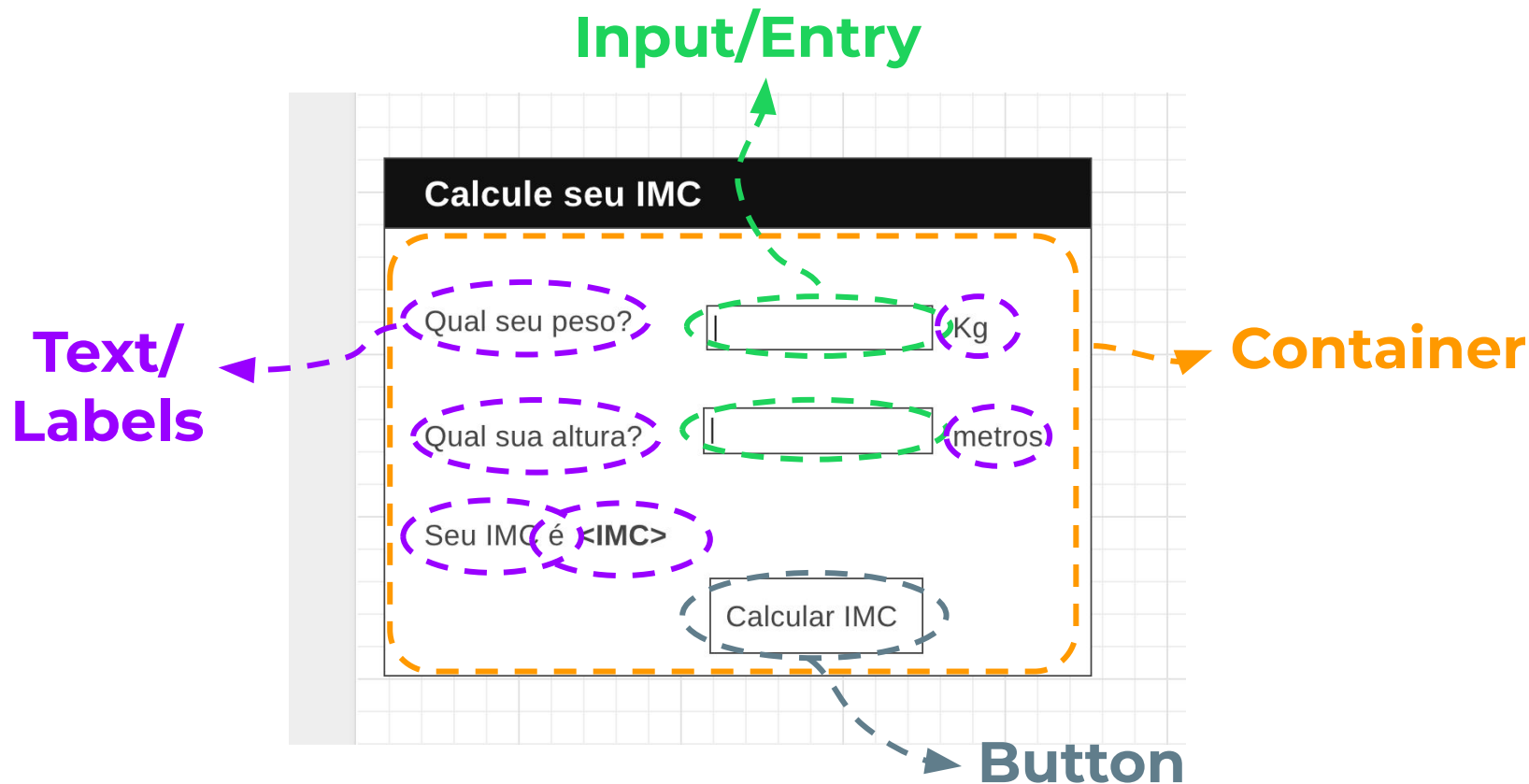
Qual seu peso?  Kg

Qual sua altura?  metros

Seu IMC é <IMC>

Calcular IMC

# Identificando os Elementos



# Identificando os Elementos

3 colunas  
4 linhas

Calcule seu IMC		
Qual seu peso?	<input type="text"/>	Kg
Qual sua altura?	<input type="text"/>	metros
Seu IMC é <IMC>		
	<input type="button" value="Calcular IMC"/>	



→ Primeiro vamos definir nossa grade e nosso container

```
import PySimpleGUI as sg
```

```
linha0 = [sg.Text("Qual seu peso?"), sg.InputText("", key="peso"), sg.Text("Kg")]
linha1 = [sg.Text("Qual sua altura?"), sg.InputText("", key="altura"), sg.Text("cm")]
linha2 = [sg.Text("Seu IMC é"), sg.Text('', key="imc", size=(6,1))]
linha3 = [sg.Text('', size=(14,1)), sg.Button("Calcular IMC")]
container = [linha0, linha1, linha2, linha3]
```

## 1. Cada linha vira uma lista de colunas de elementos

```
import PySimpleGUI as sg
```

```
linha0 = [sg.Text("Qual seu peso?"), sg.InputText("", key="peso"), sg.Text("Kg")]
linha1 = [sg.Text("Qual sua altura?"), sg.InputText("", key="altura"), sg.Text("cm")]
linha2 = [sg.Text("Seu IMC é"), sg.Text('', key="imc", size=(6,1))]
linha3 = [sg.Text('', size=(14,1)), sg.Button("Calcular IMC")]
container = [linha0, linha1, linha2, linha3]
```

## 2. Nosso container também passa a ser uma lista de linhas

```
import PySimpleGUI as sg

linha0 = [sg.Text("Qual seu peso?"), sg.InputText("", key="peso"), sg.Text("Kg")]
linha1 = [sg.Text("Qual sua altura?"), sg.InputText("", key="altura"), sg.Text("cm")]
linha2 = [sg.Text("Seu IMC é"), sg.Text('', key="imc", size=(6,1))]
linha3 = [sg.Text('', size=(14,1)), sg.Button("Calcular IMC")]
container = [linha0, linha1, linha2, linha3]
```

### 3. Elementos que envolvem valores a serem lidos/escritos podem ter uma chave para futura referência

```
import PySimpleGUI as sg

linha0 = [sg.Text("Qual seu peso?"), sg.InputText("", key="peso"), sg.Text("Kg")]
linha1 = [sg.Text("Qual sua altura?"), sg.InputText("", key="altura"), sg.Text("cm")]
linha2 = [sg.Text("Seu IMC é"), sg.Text('', key="imc", size=(6,1))]
linha3 = [sg.Text('', size=(14,1)), sg.Button("Calcular IMC")]
container = [linha0, linha1, linha2, linha3]
```

➔ Agora vamos implementar nossa lógica de execução

```
19  # Janela principal
20  window = sg.Window("Calculadora de IMC", container, font=("Helvetica", 14))
21
22  # Loop de eventos
23  rodando = True
24  ✓ while rodando:
25      |     event, values = window.read()
26      |     print(values)
27  ✓   |     if event == sg.WIN_CLOSED:
28      |         rodando = False
29  ✓   |     elif event == 'Calcular IMC':
30      |         imc = calcularIMC(values)
31      |         window.Element('imc').Update(imc)
32
33  window.close()
```

#### 4. Nossa janela principal recebe o container no seu construtor

```
19  # Janela principal
20  window = sg.Window("Calculadora de IMC", container, font=("Helvetica", 14))
21
22  # Loop de eventos
23  rodando = True
24  √ while rodando:
25      |     event, values = window.read()
26      |     print(values)
27  √   |     if event == sg.WIN_CLOSED:
28      |         rodando = False
29  √   |     elif event == 'Calcular IMC':
30      |         imc = calcularIMC(values)
31      |         window.Element('imc').Update(imc)
32
33  window.close()
```

## 5. Loop de eventos que repete até que a janela seja fechada (PoE)

```
19  # Janela principal
20  window = sg.Window("Calculadora de IMC", container, font=("Helvetica", 14))
21
22  # Loop de eventos
23  rodando = True
24  while rodando:
25      event, values = window.read()
26      print(values)
27      if event == sg.WIN_CLOSED:
28          rodando = False
29      elif event == 'Calcular IMC':
30          imc = calcularIMC(values)
31          window.Element('imc').Update(imc)
32
33  window.close()
```



## 6. Os eventos e valores são lidos a cada loop e tratados dependendo do caso

```
19  # Janela principal
20  window = sg.Window("Calculadora de IMC", container, font=("Helvetica", 14))
21
22  # Loop de eventos
23  rodando = True
24  ✓ while rodando:
25      |   event, values = window.read() ←
26      |   print(values)
27      ✓ |   if event == sg.WIN_CLOSED: ←
28      |       |   rodando = False
29      ✓ |   elif event == 'Calcular IMC': ←
30      |       |       imc = calcularIMC(values)
31      |       |       window.Element('imc').Update(imc)
32
33  window.close()
```



# Resultado

---

[Github](#)

Calculadora de IMC

Qual seu peso?  Kg

Qual sua altura?  cm

Seu IMC é

**Calcular IMC**

## Exercício recomendado

---

- Torne nossa aplicação Orientada a Objetos
- Implemente a validação do formulário para evitar erros (caso ocorra, dê feedback ao usuário)
- Ao calcular IMC, avise ao usuário se ele está com sobrepeso, peso normal etc

**Até a  
próxima!**

# Parte 3:

# Frameworks para

# GUIs e Games

# Python SimpleGUI - Fácil de usar

---

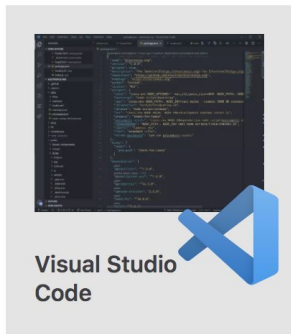
- <https://pypi.org/project/PySimpleGUI/>
- Fácil de aprender e usar
- Integra tkinter, Qt, WxPython e Remi
- Ampla documentação e tutoriais
- Controle do layout um pouco limitado



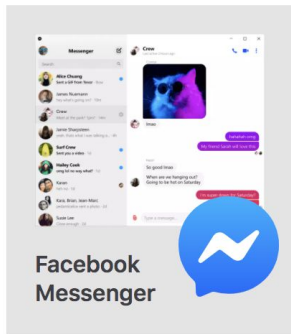
# ElectronJS - para quem gosta de Web



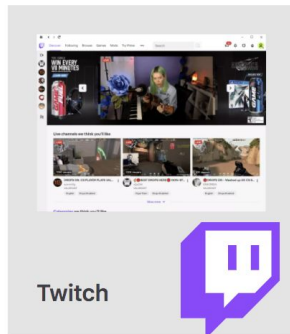
- <https://www.electronjs.org/>
- Bom para construir interfaces fluidas e responsivas com as ferramentas de web dev
- Permite o desenvolvimento de sistemas mais confiáveis com TypeScript
- Exige conhecimentos de programação web



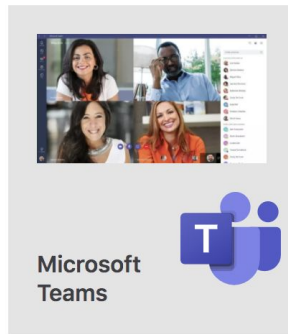
Visual Studio  
Code



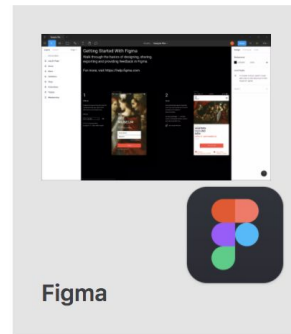
Facebook  
Messenger



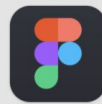
Twitch



Microsoft  
Teams



Figma



# ElectronJS - para quem gosta de Web

---

Com ElectronJS, a janela da aplicação passa a ser um **Chromium Web Browser**

app.js

```
// create a window
const myWindow = new BrowserWindow({
  width: 800,
  height: 600,
  webPreferences: {
    nodeIntegration: true
  }
});

// load a webpage
myWindow.loadFile('index.html');
})
```

index.html

```
<head></head>

<body>
  Hello Electric Universe ⚡!
</body>
```

# Elementos HTML relacionados a eventos

---

## Botões:


```
<button>Click me</button>
```



Click me

## Texto de uma linha:

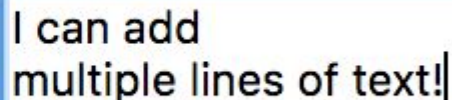
```
<input type="text" />
```



hello

## Texto de várias linhas:

```
<textarea></textarea>
```



I can add  
multiple lines of text!



# PyGame - voltado para Games

---

- <https://www.pygame.org>
- Baseado em Python (duh)
- Fácil de aprender e usar
- Ampla documentação, tutoriais e exemplos
- Bastante lento
- Não suporta 3D



## app.py

```
10  pygame.init()
11
12  game = Game()
13
14  # Game loop: laço que termina quando a janela é fechada
15  running = True
16  while running:
17      game.run()
18
19      # pygame.event.get() retorna uma lista com os eventos
20      for event in pygame.event.get():
```

## Player.py

```
import pygame
from pygame.locals import * # K_UP, K_DOWN, ...
from colors import COLORS

class Player(pygame.sprite.Sprite):
    def __init__(self):
        super(Player, self).__init__()
        self.surf = pygame.Surface((75, 25))
        self.surf.fill(COLORS["dodgerblue"])
        self.rect = self.surf.get_rect()
```

## Game.py

```
class Game:
    def __init__(self):
        # tela do jogo com 500x500 e centro da tela
        self.window = pygame.display.set_mode((500, 500))
        # nosso jogador (um retângulo mto intrépido)
        self.player = Player()
```

## app.py

```
10  pygame.init()
11
12  game = Game()
13
14  # Game loop: laço que termina quando a janela é fechada
15  running = True
16  while running:
17      game.run()
18
19      # pygame.event.get() retorna uma lista com os eventos
20      for event in pygame.event.get():
```

## Player.py

```
import pygame
from pygame.locals import * # K_UP, K_DOWN, ...
from colors import COLORS

class Player(pygame.sprite.Sprite):
    def __init__(self):
        super(Player, self).__init__()
        self.surf = pygame.Surface((75, 25))
        self.surf.fill(COLORS["dodgerblue"])
        self.rect = self.surf.get_rect()
```

## Game.py

```
class Game:
    def __init__(self):
        # tela do jogo com 500x500 e centro da tela
        self.window = pygame.display.set_mode((500, 500))
        # nosso jogador (um retângulo mto intrépido)
        self.player = Player()
```

[Github](#)

# Unity e Unreal - para quem gosta de desafios

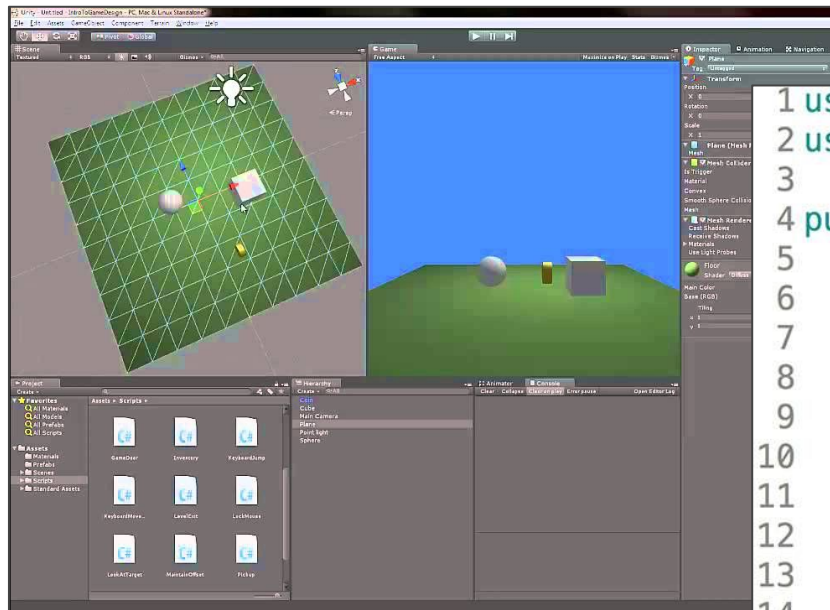
— — —

- Toolkits para geração intuitiva de cenários e objetos
- Ampla documentação
- DirectX API somente
- Lógica de jogo implementada em C#
- Licença proprietária
- Mais difícil de aprender



- Toolkits para geração intuitiva de cenários e objetos
- Lógica de jogo implementada em C++
- Suporte a OpenGL
- Ampla documentação
- Licença baseada em royalty
- Mais difícil de aprender





Unity e Unreal combinam uma GUI para montar os cenários e **C#/C++** para a lógica dos jogos

```

1 using System.Collections;
2 using UnityEngine;
3
4 public class DemoScript: MonoBehaviour {
5
6     //Variables
7     //Functions
8     //Classes
9
10    //Use this initialization
11
12    void Start () {
13
14    }
15
16
17    //Update is called once per frame
18
19    void Update () {
20
21    }
22
23 }

```

# **Algumas Dicas de Games e Game UI**

# Game Dev #1 - Narrativa e Diegese

---

**Narrativa:** história que o jogo conta. Todos os elementos do jogo devem trabalhar para reforçar a narrativa

**Elementos diegéticos:** fazem parte do universo do jogo. Os personagens estão cientes que os elementos existem

**Elementos não diegéticos:** estão fora do universo do jogo. São utilizados para dar feedback ao jogador

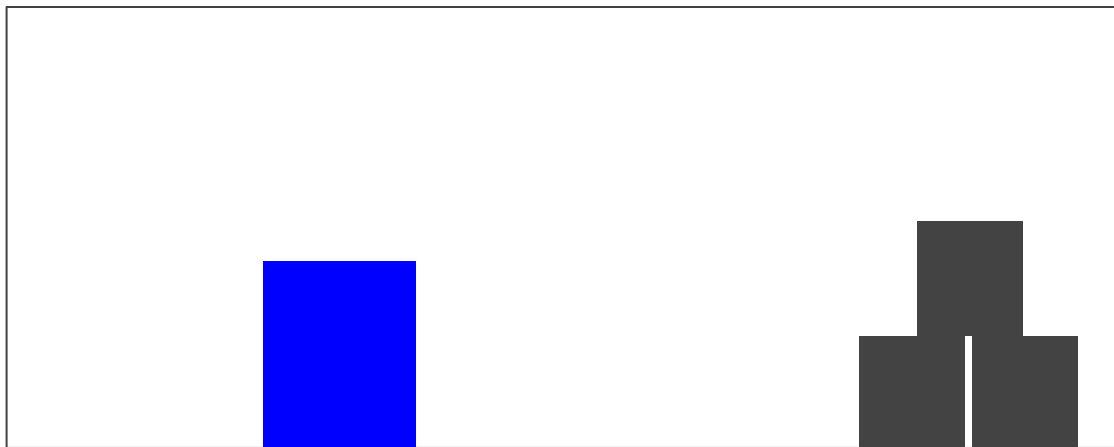
# Game Dev #2 - Por onde começo?

---

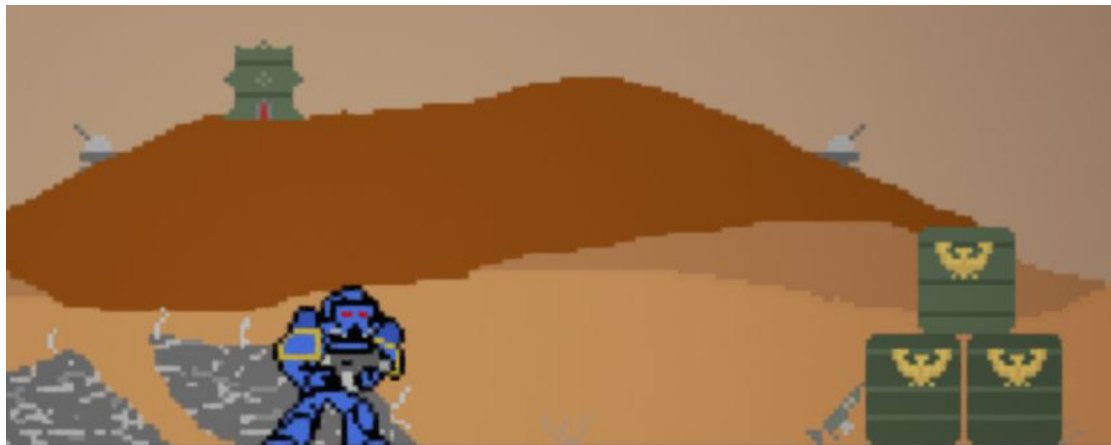
- 1) Parta da **descrição** do jogo
- 2) Tudo que **tem estado ou comportamento** é uma classe
- 3) Tudo que **ocupa o espaço do jogo** é uma classe
- 4) Barras de status/informações e elementos do tipo são classes
- 5) Detalhe essa descrição durante o desenvolvimento do jogo para gerar o **manual de desenvolvimento**
- 6) Busque por **assets** para não ter que desenhar tudo
- 7) Implemente um **protótipo** primeiro
- 8) Jogabilidade >>> Complexidade

\*Esse guia [aqui](#)  
é bem legal





Protótipo



Versão Final

# Game Dev #3 - Assets

Tiles

Background

Sprites

Minimap

User Stats

Meta  
elementos



# Game Dev #3 - Assets

---

→ Algumas fontes de assets:

- ◆ <https://opengameart.org/>
- ◆ <https://www.gameart2d.com/freebies.html>

→ Áudios:

- ◆ <https://kenney.nl/assets/rpg-audio>
- ◆ <https://99sounds.org/free-sound-effects/>

→ **PS:** Cuidado com **copyright!**

**Até a  
próxima!**