🛠️ Aula Prática: Construção de Projeto Python com CI Local (Poetry + Análise de Qualidade)

1. Preparação do ambiente

pip install poetry

2. Inicialização do projeto com Poetry

poetry init

Siga os prompts para configurar o projeto. Depois:

poetry add pygame

3. Criação do arquivo principal

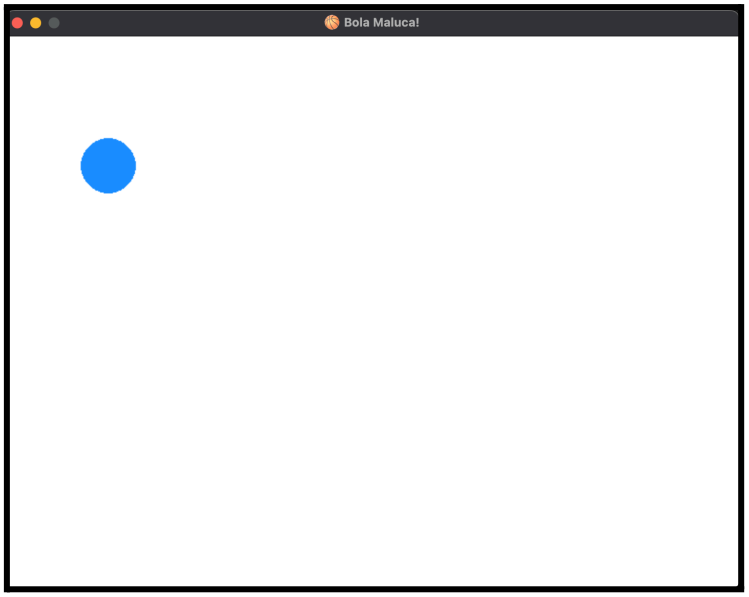
Crie o arquivo jogo.py com o código fornecido (jogo da bola maluca).

```
import pygame

import random



# Inicializa o pygame

pygame.init()



# Tamanho da tela

largura = 800

altura = 600

tela = pygame.display.set_mode((largura, altura))

pygame.display.set_caption("🏀 Bola Maluca!")



# Cores

BRANCO = (255, 255, 255)

AZUL = (30, 144, 255)
```

```
# Propriedades da bola

raio = 30

x = random.randint(raio, largura - raio)

y = random.randint(raio, altura - raio)

vel_x = random.choice([-4, 4])

vel_y = random.choice([-4, 4])



# Loop do jogo

relogio = pygame.time.Clock()

rodando = True



while rodando:

   for evento in pygame.event.get():

       if evento.type == pygame.QUIT:

           rodando = False



   # Atualiza a posição da bola

   x += vel_x

   y += vel_y



   # Rebater nas bordas

   if x - raio <= 0 or x + raio >= largura:

       vel_x *= -1

   if y - raio <= 0 or y + raio >= altura:

       vel_y *= -1
```

```
    # Preenche o fundo
    tela.fill(BRANCO)

    # Desenha a bola
    pygame.draw.circle(tela, AZUL, (x, y), raio)

    # Atualiza a tela
    pygame.display.flip()

    # Controla os FPS
    relogio.tick(60)

pygame.quit()
```

Realize um teste e valide que a imagem abaixo é exibida:



4. Análise de Qualidade com Ferramentas Dev

Instale os pacotes de desenvolvimento:

poetry add --group dev flake8 pylint mypy radon

5. Execute cada uma das ferramentas e observe os resultados:

poetry run flake8 jogo.py

poetry run pylint jogo.py

poetry run mypy jogo.py

poetry run radon cc jogo.py -a

6. Implementação de Testes com Pytest

Adicione o pytest:

poetry add --group dev pytest

Crie a pasta **tests** e adicione **test_jogo.py** com os testes fornecidos.

```
import pytest
from jogo import Bola


def test_bola_move_direcao_certa():
    bola = Bola(x=100, y=100, vel_x=5, vel_y=3)
    bola.mover()
    assert bola.x == 105
    assert bola.y == 103


def test_bola_quica_nas_bordas_horizontais():
    bola = Bola(x=30, y=100, vel_x=-5, vel_y=0)  # bem na borda esquerda
    bola.mover()
    assert bola.vel_x == 5  # deve inverter direção


def test_bola_quica_nas_bordas_verticais():
    bola = Bola(x=100, y=30, vel_x=0, vel_y=-5)  # bem no topo
    bola.mover()
    assert bola.vel_y == 5  # deve inverter direção
```

Para que os testes funcionem, transforme a lógica da bola em uma classe Bola. Sugestão de início:

```python
class Bola:
    def __init__(self, x: int, y: int, vel_x: int, vel_y: int) -> None:
        self.x = x
        self.y = y
        self.vel_x = vel_x
        self.vel_y = vel_y

    def mover(self) -> None:
        self.x += self.vel_x
        self.y += self.vel_y

        # Rebater nas bordas
        if self.x - RAIO_BOLA <= 0 or self.x + RAIO_BOLA >= LARGURA_TELA:
            self.vel_x *= -1
        if self.y - RAIO_BOLA <= 0 or self.y + RAIO_BOLA >= ALTURA_TELA:
            self.vel_y *= -1

    def desenhar(self, tela: pygame.Surface) -> None:
        pygame.draw.circle(tela, AZUL, (self.x, self.y), RAIO_BOLA)
```

7. Execução dos Testes

poetry run pytest

🔧 Workflow: CI/CD Pipeline

🧩 Parte 1: Definição do Gatilho (on)

```yaml
on:

 push:

   branches:

     - main

 pull_request:

   branches:
```

```
    - main
```

📌**Dispara a pipeline quando houver um push ou pull request na branch main.**

🧩 Parte 2: Definição do Job

```
jobs:

 build:

   runs-on: ubuntu-latest
```

📌 **Define o job build que será executado em um runner Ubuntu.**

🧩 Parte 3: Etapas do Job (steps)

Etapa 3.1 – Clonar o Código

```
   steps:

     - name: Checkout do Código

       uses: actions/checkout@v3
```

📌 **Baixa o código do repositório para o runner.**

Etapa 3.2 – Instalar o Poetry

```
     - name: Instalar o Poetry

       run: |

         curl -sSL https://install.python-poetry.org | python3 -

        echo "$HOME/.local/bin" >> $GITHUB_PATH
```

📌 **Função: Baixa e instala o gerenciador de dependências poetry.**

Etapa 3.3 – Configurar Cache

```
     - name: Configurar cache do Poetry
```

```
      uses: actions/cache@v3
      with:
        path: |
          ~/.cache/pypoetry
          ~/.cache/pip
        key: poetry-${{ runner.os }}-${{ hashFiles('**/poetry.lock') }}
        restore-keys: |
          poetry-${{ runner.os }}-
```

📌**Salva e reutiliza o cache das dependências para acelerar builds futuros.**

Etapa 3.4 – Instalar Dependências

```
    - name: Instalar dependências
      run: poetry install --no-root
```

📌 **Instala todas as dependências do projeto com o poetry.**

🧪 **Parte 4: Execução da Qualidade e Testes**

Etapa 4.1 – Executar Testes

Etapa 4.2 – Verificar Estilo com Flake8

Etapa 4.3 – Verificar Estilo com Pylint

Etapa 4.4 – Verificar Tipagem com Mypy

Etapa 4.5 – Analisar Complexidade com Radon