**Projeto Final - Avaliação A3**

# FLAPPY BIRD

Relatório técnico apresentado na UC Usabilidade, desenvolvimento web e jogos pelo Prof. MSc Flávio Henrique da Silva.

**Curitiba**

**2025**

## INTEGRANTES DO GRUPO

Ana Julia Bernardo Lazaro – 172211672

Carlos Daniel Steindorf Pereira - 172220112

João Gabriel Breve - 172317201

Luiz Guilherme Olibratoski Fernandes – 172220449

**SUMÁRIO**

## 1 INTRODUÇÃO

O desenvolvimento de jogos digitais é uma área em constante crescimento dentro do campo da programação, combinando lógica computacional, design gráfico e interatividade. Projetos de jogos, mesmo os mais simples que fizemos em sala, permitem a aplicação prática de conceitos fundamentais da computação, como estruturas de dados, orientação a objetos e manipulação de eventos.

Este trabalho foi realizado como avaliação da unidade curricular em que foram abordados temas como usabilidade e desenvolvimento de jogos digitais. O principal objetivo do projeto foi criar uma versão do jogo *“Flappy Bird”*, utilizando a linguagem de programação Python e a biblioteca Pygame. A proposta visou consolidar os conteúdos das aulas por meio da prática, oferecendo uma experiência de desenvolvimento e análise de jogabilidade.

A escolha do *“Flappy Bird”* como base do projeto foi feita por meio de um sorteio realizado em sala de aula, em que diferentes grupos foram designados a desenvolver jogos diferentes. O *“Flappy Bird”* foi atribuído ao nosso grupo e por sua mecânica simples, porém desafiadora, se mostrou uma escolha adequada para a utilização dos conceitos estudados.

Durante o processo de desenvolvimento, optou-se por manter todo o código em um único arquivo Python, mas com uma estrutura lógica bem definida, dividida em seções para configurações, classes e loop principal. O jogo foi desenvolvido com atenção à usabilidade, conceito visto em sala, para proporcionar uma experiência intuitiva e fluida ao jogador.

## 2 DESENVOLVIMENTO

O presente trabalho tem como objetivo apresentar o desenvolvimento de um jogo digital inspirado no clássico *"Flappy Bird"*, criado durante as aulas da disciplina de Usabilidade, Desenvolvimento Web e Jogos. O projeto foi idealizado como uma forma prática de aplicar conceitos de lógica de programação, estruturas de repetição, manipulação de gráficos e interação com o usuário.

Durante o desenvolvimento, os alunos foram incentivados a trabalhar em equipe, utilizando a linguagem Python e a biblioteca Pygame. O projeto envolveu etapas de planejamento, desenvolvimento da lógica de colisão e pontuação, além da implementação de interfaces e sons. O resultado foi uma versão funcional e jogável do Flappy Bird, servindo como base para a consolidação dos conceitos aprendidos em sala.

### 2.1 Divisão das Tarefas

O desenvolvimento do projeto foi dividido entre os membros do grupo, de modo a distribuir as tarefas conforme as habilidades e interesses de cada um, garantindo maior eficiência e aprendizado colaborativo.

Carlos Daniel Steindorf Pereira cuidou da coleta dos Assets na internet, realizou a montagem do esqueleto do jogo, fez a preparação do ambiente e animação de movimento.

João Gabriel Breve foi responsável pela programação da lógica principal do jogo, incluindo a detecção de colisões, controle da pontuação e integração dos elementos visuais ao código, garantindo a coerência estética do jogo. Também participando ativamente da fase de testes e correção de bugs.

Luiz Guilherme Olibratoski Fernandes ficou encarregado da implementação dos efeitos sonoros, bem como da configuração das telas inicial e final do jogo. Além de auxiliar na fase de testes e correções de bugs.

Ana Julia Bernardo Lazaro realizou o registro contínuo do andamento do projeto, e produziu a escrita do relatório. Também contribuiu com a codificação de partes complementares, como o menu inicial.

## 2.2 Estrutura do Projeto

Esta seção descreve a estrutura interna do projeto desenvolvido, detalhando os recursos utilizados, a organização dos arquivos, as classes implementadas e suas respectivas funcionalidades. O objetivo é apresentar de forma clara como o código foi estruturado para garantir o funcionamento do jogo, bem como as responsabilidades de cada componente dentro do sistema.

### 2.2.1 Bibliotecas e Frameworks Utilizados

Para a implementação do jogo, foi utilizada a linguagem de programação Python, em conjunto com a biblioteca Pygame. O Pygame foi fundamental para a manipulação de imagens, controle de eventos e execução do loop principal do jogo. Além disso, a biblioteca “random” foi utilizada para gerar variações nos obstáculos (canos).

### 2.2.2 Estrutura dos Arquivos

Os arquivos do projeto foram organizados de forma organizada, com o objetivo de facilitar a manutenção, leitura e reutilização do código. Cada arquivo concentra responsabilidades específicas, como código fonte, dependências do projeto, e recursos multimídia. Permitindo uma melhor separação de funções e maior clareza na estrutura geral do sistema.

Descrição da estrutura de diretórios utilizada:

/assets: pasta que armazena todos os recursos visuais e sonoros do jogo.

- /fonts: contém as fontes tipográficas utilizadas para exibição de texto no jogo.
- /images: guarda as imagens do jogo, como o pássaro, os canos e o plano de fundo.

- /sounds: contém os efeitos sonoros usados durante a execução do jogo, como o som de pulo e colisão.

/scr: diretório onde está localizado o código-fonte do jogo, incluindo as classes e funções responsáveis pela lógica, interface e execução da aplicação.

.gitignore: arquivo utilizado para ignorar arquivos temporários e pastas.

README.md: documento que apresenta informações básicas sobre o projeto, como objetivos, funcionalidades, e instruções de instalação.

requirements.txt: lista de dependências necessárias para a execução do projeto, especificando a biblioteca externa utilizada, o Pygame.

### 2.2.3 Principais Objetos, Classes e Funções

O código do jogo *Flappy Bird* é estruturado com base em conceitos de orientação a objetos e divide suas funcionalidades em diferentes classes e funções para modularizar a lógica do jogo. Toda a execução é feita com o auxílio da bibliotecas Pygame e Random, utilizadas para desenvolver jogos em Python.

```
import pygame
import random
```

A classe “TelaManager” atua como um controlador de telas, responsável por alternar entre os diferentes estados do jogo, como menu principal, jogo em execução, tutorial e tela de morte. Ela mantém uma referência à tela atual por meio do atributo “current”, e o método “go_to()” permite a troca de telas conforme as interações do jogador. Todas as telas do jogo herdam da classe base “Tela”, que

define a estrutura padrão para os métodos “handle_events()”, “update()” e “draw()”, que são sobrescritos por suas subclasses.

```
class TelaManager:
    def __init__(self):
        self.pontuacao = 0
        self.current = MenuPrincipal(self)

    def go_to(self, tela):
        self.current = tela

class Tela:
    def __init__(self, manager):
        self.manager = manager

    def handle_events(self, events): pass
    def update(self): pass
    def draw(self, superficie): pass
```

A classe “MenuPrincipal” é uma dessas subclasses e representa a tela inicial do jogo. Ela exibe o fundo, o chão e o logo do jogo, além de instruções básicas para iniciar ou acessar o tutorial. Nela, ao usuário pressionar “P”, o jogo começa, e ao pressionar “T”, o jogador é levado à classe “TelaTutorial”, que mostra mensagens com dicas básicas sobre a jogabilidade, como pular com a barra de espaço e evitar os tubos. A transição entre essas telas é feita através das teclas pressionadas pelo jogador, capturadas no método “handle_events()”.

```
class MenuPrincipal(Tela):
    def __init__(self, manager):
        super().__init__(manager)
        self.fundo = FUNDODIA
        self.chao = CHAO
        self.logo = pygame.image.load(IMG + 'logo.png')
        self.logo = pygame.transform.scale(self.logo, (368, 94))
        self.logo_rect = self.logo.get_rect()
        self.logo_rect.centerx = WID // 2
        self.logo_rect.y = 100

    def handle_events(self, events):
        for e in events:
            if e.type == pygame.KEYDOWN:
```

```
                if e.key == pygame.K_p:
                    PLAY.play()
                    self.manager.go_to(TelaJogo(self.manager))
                if e.key == pygame.K_t:
                    self.manager.go_to(TelaTutorial(self.manager))

    def draw(self, superficie):
        superficie.blit(self.fundo, (0, 0))
        superficie.blit(self.chao, (0, 612))
        superficie.blit(self.logo, self.logo_rect)
        TextoCentralizado(TEL, MSG_MENU_PRINCIPAL[1], FONT, HEI // 2 - 30)
        TextoCentralizado(TEL, MSG_MENU_PRINCIPAL[2], FONT, HEI // 2 + 30)
```

Quando o jogador inicia o jogo, a tela muda para a classe “TelaJogo”, que representa o núcleo da experiência de jogo. Ela controla todos os elementos visuais e lógicos que ocorrem durante a partida, incluindo o movimento do fundo e do chão, a criação e movimentação dos obstáculos (os canos), a atualização da pontuação e a verificação de colisões. Caso o jogador colida com um cano ou com o chão, o método “morrer()” é chamado, tocando um som e mudando para a classe “TelaMorte”, que exibe a pontuação obtida e oferece opções para reiniciar (R) ou voltar ao menu principal (M).

```
class TelaJogo(Tela):
    def __init__(self, manager):
        super().__init__(manager)
        self.fundo = FUNDODIA
        self.chao = CHAO
        self.LARGURA_FUNDO = self.fundo.get_width()
        self.LARGURA_CHAO = self.chao.get_width()

        self.mov_chao = 0
        self.mov_fundo = 0
        self.vel_chao = 4
        self.vel_fundo = 2

        self.jogador_grupo = pygame.sprite.Group()
        self.flappy = Jogador(100, HEI // 2)
        self.jogador_grupo.add(self.flappy)

        self.canos = pygame.sprite.Group()
        self.tempo_ultimo_cano = pygame.time.get_ticks()
```

```
        self.delay_entre_pipes = 1500

        self.pontuacao = 0


    def handle_events(self, events):
        for e in events:
            if e.type == pygame.KEYDOWN and e.key == pygame.K_SPACE:
                self.flappy.pular()
                PULO.play()
            elif e.type == pygame.KEYDOWN and e.key == pygame.K_m:
                self.manager.go_to(MenuPrincipal(self.manager))

    def morrer(self):
        self.manager.pontuacao = self.pontuacao
        MORTE.play()
        self.manager.go_to(TelaMorte(self.manager))
```

O personagem principal (pássaro), é representado pela classe “Jogador”, que herda de “pygame.sprite.Sprite”. Essa classe lida com a física do personagem, como gravidade, velocidade vertical e o efeito do pulo. O método “pular()” altera a velocidade vertical negativamente, simulando um impulso para cima, e troca a imagem do pássaro para dar a sensação de movimento. Já o método “update()” aplica a gravidade e ajusta a imagem conforme a velocidade, além de restringir o movimento do pássaro ao chão da tela.

```
class Jogador(pygame.sprite.Sprite):
    def __init__(self, x, y):
        super().__init__()
        self.image_down = pygame.image.load(IMG + 'redbird-downflap.png')
        self.image_mid = pygame.image.load(IMG + 'redbird-midflap.png')
        self.image_up = pygame.image.load(IMG + 'redbird-upflap.png')
        self.image = self.image_mid
        self.rect = self.image.get_rect()
        self.rect.center = [x, y]
        self.velocidade = 0
        self.gravidade = 0.5
        self.forca_pulo = -10

    def update(self):
        self.velocidade += self.gravidade
        self.rect.y += self.velocidade
```

```
        if self.rect.bottom > 612:
            self.rect.bottom = 612
            self.velocidade = 0

        if self.velocidade > 0:
            self.image = self.image_up
        elif self.velocidade == 0:
            self.image = self.image_mid

    def pular(self):
        self.velocidade = self.forca_pulo
        self.image = self.image_down
```

Os obstáculos do jogo são gerados por meio da classe “Canos”, também derivada de “pygame.sprite.Sprite”. Ela é responsável por criar pares de canos com uma abertura no meio (gap), com posições verticais aleatórias. Os canos se movem para a esquerda, e quando ultrapassam a tela, são removidos do grupo de sprites. O método “draw()” garante que os canos sejam desenhados de forma contínua, repetindo as imagens ao longo da altura da tela.

```
class Canos(pygame.sprite.Sprite):
    def __init__(self, x):
        super().__init__()
        self.image_cano = PIPE_IMAGE
        self.image_cano_top = PIPE_IMAGE_TOP

        self.gap = 180  # Espaço entre os canos.
        self.velocidade = 4

        self.altura = random.randint(150, 450)
         self.top_rect = self.image_cano_top.get_rect(midbottom=(x, self.altura
- self.gap // 2))
          self.bottom_rect = self.image_cano.get_rect(midtop=(x, self.altura +
self.gap // 2))

        self.ponto_contado = False

    def update(self):
        self.top_rect.x -= self.velocidade
        self.bottom_rect.x -= self.velocidade
```

```
        # reset logic (optional, or remove pipes when offscreen)
        if self.top_rect.right < 0:
            self.kill()

    def draw(self, surface):
        top_x = self.top_rect.x
        y = self.top_rect.bottom
        while y > 0:
                        rect  =  self.image_cano_top.get_rect(midbottom=(top_x  +
self.top_rect.width // 2, y))
            surface.blit(self.image_cano_top, rect)
            y -= self.image_cano_top.get_height()

        bottom_x = self.bottom_rect.x
        y = self.bottom_rect.top
        while y < 612:
                           rect  =  self.image_cano.get_rect(midtop=(bottom_x  +
self.bottom_rect.width // 2, y))
            surface.blit(self.image_cano, rect)
            y += self.image_cano.get_height()
```

Além das classes, o código conta com funções utilitárias, como “ConvPontos()”, que converte a pontuação do jogador de número para imagens dos dígitos, formando o placar visual. A função “TextoCentralizado()” é usada para desenhar mensagens no centro da tela, com bordas estilizadas geradas pela função “MargemParaTexto()”. Essas funções ajudam na exibição visual do jogo, tornando-o mais interativo e claro para o jogador.

```
def ConvPontos(score, surface, x, y):
    for digito in str(score):
        imagem = NUM_IMAGES[digito]
        surface.blit(imagem, (x, y))
        x += imagem.get_width()

def MargemParaTexto(texto, fonte, cor_texto, cor_borda, tamanho_borda=2):
    base = FONT.render(texto, True, cor_texto)
    tamanho = base.get_size()
     img  =  pygame.Surface((tamanho[0]  +  2  *  tamanho_borda,  tamanho[1]  +  2  *
tamanho_borda), pygame.SRCALPHA)

    for dx in range(-tamanho_borda, tamanho_borda + 1):
        for dy in range (-tamanho_borda, tamanho_borda + 1):
            if dx == 0 and dy == 0:
```

```
                continue
            img.blit(FONT.render(texto, True, cor_borda), (dx + tamanho_borda,
dy + tamanho_borda))

    img.blit(base, (tamanho_borda, tamanho_borda))
    return img

def TextoCentralizado(superficie, texto, fonte, y, cor_texto=(255,255,255),
cor_borda=(0,0,0), espessura=3):
    texto_img = MargemParaTexto(texto, fonte, cor_texto, cor_borda, espessura)
    texto_rect = texto_img.get_rect(center=(WID // 2, y))
    superficie.blit(texto_img, texto_rect)
```

Para o aspecto sonoro, o jogo utiliza arquivos de som para reforçar eventos importantes como pulo, ponto, morte e início de jogo. Esses sons são carregados e configurados com volume específico no início da execução, por meio do módulo “pygame.mixer”.

```
pygame.mixer.pre_init(frequency=44100, size=-16, channels=2, buffer=256)
```

Por fim, a execução principal do jogo está no loop “while RUNNING”, onde a tela atual é atualizada a cada frame. Nele, eventos do teclado são processados, a lógica de jogo é atualizada e os elementos visuais são desenhados na tela. Quando o jogador decide sair, o loop é encerrado e o “pygame.quit()” é chamado para fechar o jogo adequadamente.

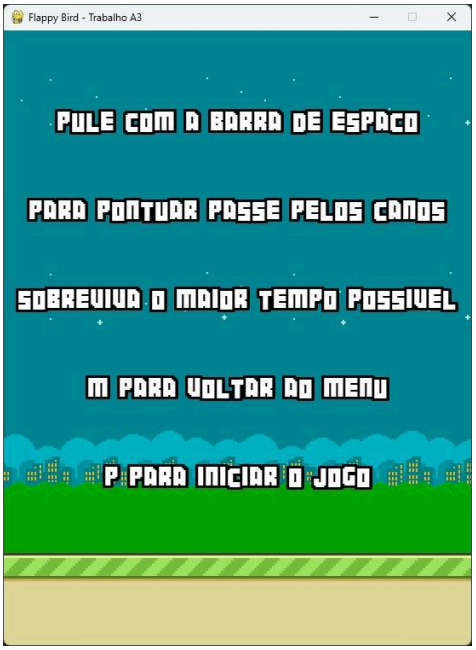
```
while RUNNING:
    CLOCK.tick(FPS)
    events = pygame.event.get()

    for event in events:
        if event.type == pygame.QUIT:
            RUNNING = False

    manager.current.handle_events(events)
    manager.current.update()
    manager.current.draw(TEL)

    pygame.display.update()

pygame.quit()
```

## 2.3 Explicação da Aplicação/Software

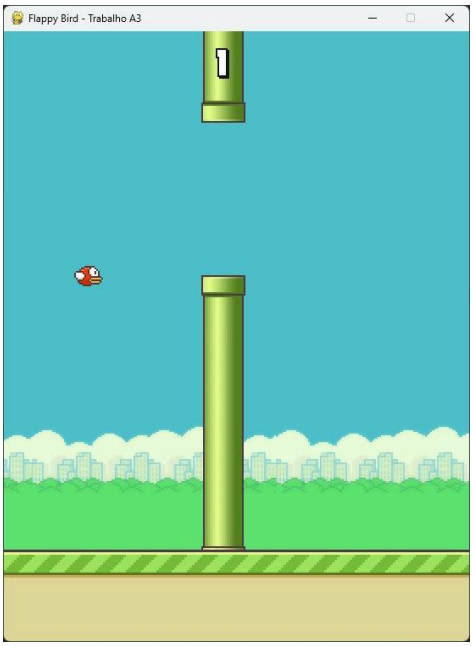
FlappyBird é uma releitura do clássico jogo “Flappy Bird”, desenvolvida com a biblioteca Pygame. O objetivo do jogo é controlar um pássaro que precisa desviar de obstáculos e atravessar aberturas entre tubos verticais, acumulando pontos a cada passagem bem-sucedida. Suas principais funcionalidades foram divididas de forma a apresentar ao usuário um menu inicial, manual, uma tela interativa de jogo, e uma tela apresentada no momento em que o jogador colide com um obstáculo.

No menu inicial há duas opções para o usuário, sendo a primeira para iniciar o jogo (apertando a tecla “P”), e a segunda para ter a visualização do manual (apertando a tecla T).


Quando o usuário abre o manual ele possui acesso às regras do jogo. Logo abaixo das regras são apresentadas duas opções: apertar “M” para voltar ao menu, ou “P” para iniciar o jogo.




Após a seleção da tecla “P”, o usuário poderá controlar o pássaro utilizando o mouse, ou a barra de espaço do teclado.




Assim que houver uma colisão do pássaro com um obstáculo, automaticamente o jogo é encerrado apresentando a tela de “Game Over”, onde poderá ser visualizada a pontuação final do jogo, e serão apresentadas duas opções: apertar o “M” para voltar ao menu, e “R” para iniciar.


## 2.4 Orientações de Execução da Aplicação/Software

Para executar a aplicação *Flappy Bird*, é necessário que o usuário tenha o Python 3.7 ou superior instalado, juntamente com o Pygame 2.6.1. Após serem baixados ou clonados os arquivos do projeto do repositório, devem ser instaladas as dependências usando o comando “pip install -r requirements.txt” no terminal. Em seguida, deve ser executado o jogo rodando “python game.py” na mesma pasta do arquivo. A estrutura do projeto inclui uma pasta “assets” com imagens, sons e fontes, o arquivo principal “game.py”, um executável opcional “FlappyBird.exe” e o arquivo “requirements.txt” com as dependências.

## 2.5 Repositório

https://github.com/joaogbreve/A3-Flappy-Bird

## 3 CONCLUSÃO

O desenvolvimento do projeto *Flappy Bird* ao longo do semestre permitiu aplicar de forma prática os conhecimentos adquiridos em programação e design de interfaces, promovendo uma compreensão mais profunda sobre lógica de jogos e experiência do usuário. Durante o processo, o objetivo não foi apenas replicar a mecânica original do jogo, mas também aprimorar aspectos de usabilidade, como a simplicidade na navegação, a resposta imediata aos comandos do jogador, e uma interface limpa e intuitiva que facilita a interação com o sistema.

Além disso, elementos como a consistência visual e o posicionamento estratégico dos botões foram planejados com base em princípios de usabilidade, garantindo uma experiência fluida mesmo para usuários iniciantes. A possibilidade de reiniciar o jogo com apenas um clique e a exibição clara da pontuação também contribuíram para tornar o jogo mais acessível e envolvente.

Concluímos, portanto, que o projeto cumpriu seu objetivo principal de desenvolver um jogo funcional e atrativo, ao mesmo tempo em que serviu como um exercício valioso para integrar teoria e prática em design centrado no usuário. A análise final demonstra que, ao incorporar princípios de usabilidade, é possível tornar um jogo simples como o *Flappy Bird* em uma experiência mais completa e agradável para o usuário.

## REFERÊNCIAS

https://github.com/samuelcust/flappy-bird-assets