Relatório de Projeto: Flappy Pig

Turma: Engenharia da Computação 2025.1

Equipe 4: Arthur Moura, Davi José, Davi Pinheiro, Heberty Souza, Matheus Vasconcelos, Rodrigo Silveira

Divisão de Trabalho:

Rodrigo Silveira <rggs> - Responsável pelos assets e sprites do jogo.
Matheus Vasconcelos <msv5> - Relatório, preenchimento de checkpoints e slides
Arthur Moura <ambv>, Davi José <djsb>, Davi Pinheiro <dlp> e Heberty Souza <hsp2> - Códigos, organização e criação do repositório github.

Introdução:

Flappy Pig é um jogo do gênero "**side-scroller**" inspirado pelo icônico jogo de celular Flappy Bird, assim como diversos outros jogos, como Geometry Dash e T-Rex Game. O objetivo do jogador é sobreviver com o personagem - **um porco voador** - o máximo de tempo possível, desviando dos obstáculos no caminho enquanto coleta itens que mudam a aparência do animal conforme são adquiridos.

Arquitetura do Código:

Além das estruturas de loops e condicionais utilizadas ao longo do período, o código do jogo inclui também **Bibliotecas Externas** e **Classes**, partes indispensáveis do projeto.

Para as bibliotecas externas, além da biblioteca do **Pygame**, necessária para o funcionamento do projeto, também foi importada a biblioteca **Random**, utilizada para gerar, de maneira aleatória, o tamanho dos obstáculos do cenário e o tipo de item coletado.

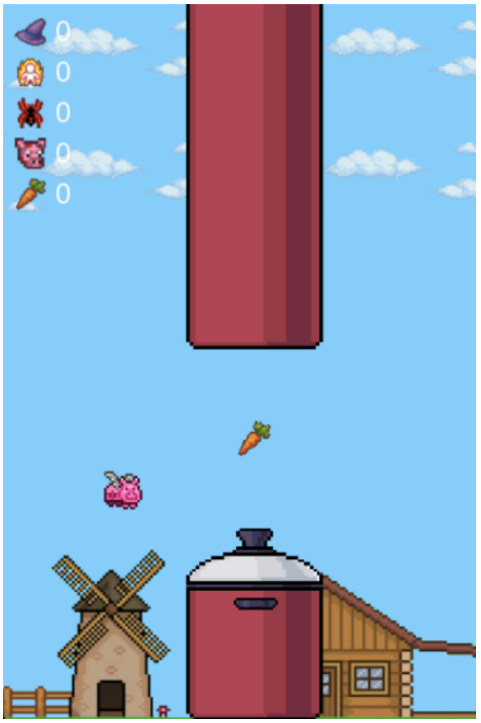
Já as **classes** foram cruciais na construção do programa. Cada uma - **Consts, GameObject, GameState,GameManager, Item, Pig, NewPipe, GameState, TelaMenu, TelaEnd,TelaBase** e **Assets-** contém classes genéricas que foram importadas no arquivo principal do jogo para torná-lo de mais fácil visualização e correção, e cada uma diz respeito a uma parte distinta do código base.

Classe Consts: Salva todas as combinações de cores no sistema RGB utilizadas no projeto em uma classe para tornar o código mais intuitivo.
Classe GameObject: Responsável por receber as entradas de coordenadas, dimensões e cores do objeto, "criando" ele propriamente.
Classe GameState: Checa constantemente o estado atual do jogo (se ele está rodando, se está na tela de menu, ou se chegou no Game Over).

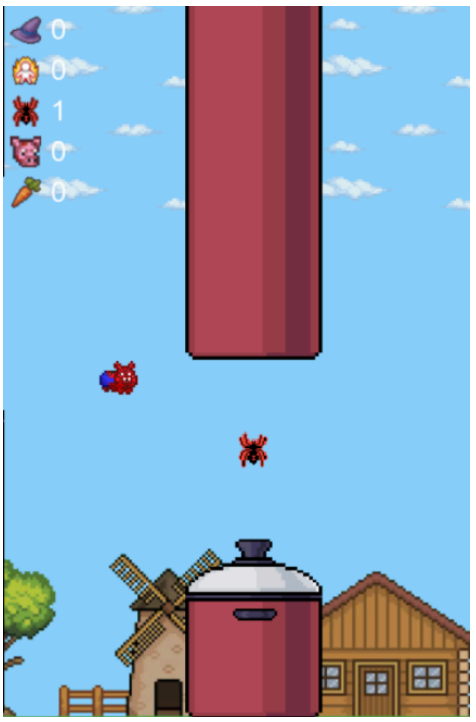
***Classe GameManager*:** Controla os estados dos itens e a pontuação deles, que serão exibidas continuamente no loop principal do jogo.
Classe Item: Atualiza a posição do item e avalia suas características.
Classe NewPipe: Encarregado da criação dos obstáculos (canos).
Classe Pig: Cria o sprite do personagem, com suas cores e dimensões.
Classe TelaBase: Define as dimensões da tela do jogo.
Classes TelaMenu e TelaEnd: Definem as configurações da tela de Menu e da tela de Game Over, respectivamente.
Arquivo Assets: Armazena em um dicionário o caminho para que o programa identifique quais serão os arquivos de assets atualizados constantemente no jogo.

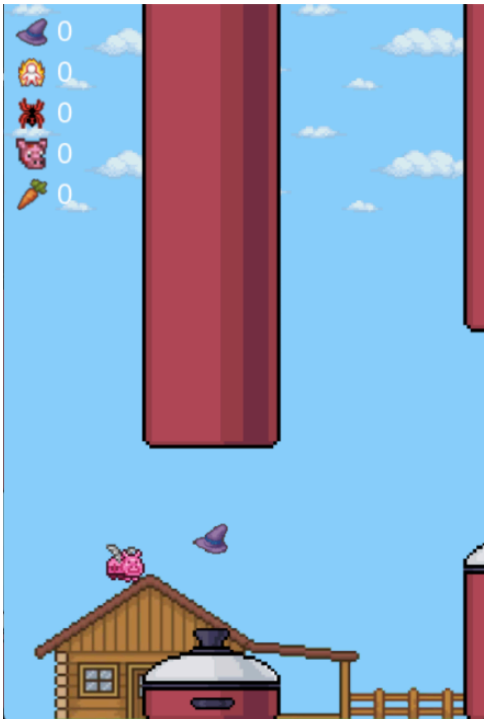
Assim, os arquivos acima foram importados no arquivo principal **FlappyPig** (a classe **"main"**), onde o jogo é executado. É nele que são inicializados os objetos fundamentais para o funcionamento do jogo, tal como a velocidade de movimentação do background, a velocidade de queda e de subida do personagem, a criação dos espaçamentos entre os obstáculos no cenário e o loop de carregamento do jogo. Além disso, o grupo também optou por formatar a chamada dos blocos de código de cada criação de objeto (Criação dos itens, canos e da pontuação de cada coletável) em funções presentes na Classe Main(FlappyPig,py) para atualizar os atributos de cada objeto e dar continuidade ao jogo de forma mais otimizada.

Capturas de Tela:




0
0
0
0
0


0
0
1
0
0


0
0
0
0
0


0
1
0
0
1

Influências da Disciplina:

O aprendizado de **estruturas condicionais**, **comandos de loop** e **definição de funções**, ensinados durante o período de Introdução à Programação, se mostraram extremamente necessários para a construção do código.

As estruturas condicionais if/else foram usadas em diversos momentos para checar se as condições de execução do jogo foram satisfeitas, bem como para movimentar o personagem para cima de acordo com as teclas pressionadas.

Os loops foram usados não somente para manter o programa constantemente rodando, mas também para receber as entradas de eventos do código, como possíveis colisões do personagem com os obstáculos e receber as entradas de botões pressionados.

Já as funções foram utilizadas para definir as principais ações que tangem o jogo, como a renderização dos coletáveis, dos obstáculos e principalmente da execução do jogo.

Principais Desafios e Aprendizados:

Dificuldades e Erros:

Uma das principais dificuldades do projeto foi o aprendizado da engine “Pygame”, já que apresentou uma perspectiva completamente nova e diferente de enxergar a programação em Python.
A programação orientada a objetos também apresentou desafios formidáveis, afinal a maioria dos membros do grupo nunca teve uma introdução devida ao tema, exigindo uma quantidade considerável de tempo para que pudesse ser dominada e compreendida pelos participantes.

Grande parte dos erros cometidos durante a idealização do projeto envolveu a movimentação do personagem e a sua colisão com o ambiente, já que, nas etapas iniciais do trabalho, a equipe ainda não havia aprendido a trabalhar nos repositórios do GitHub, tornando a criação do código um processo muito mais trabalhoso e demorado.

Aprendizados:

Apesar de todas as dificuldades, o projeto proporcionou uma enorme gama de aprendizados para a equipe como um todo, estimulando o trabalho em equipe, a criatividade e a organização de todos os membros.

Sem dúvidas, um dos maiores aprendizados do projeto foi o domínio das funcionalidades do GitHub, já que trata-se de uma plataforma extremamente útil para a carreira acadêmica e profissional de um programador.
Além disso, a introdução à Programação Orientada a Objetos também foi extremamente útil, visto que será uma constante em projetos futuros e extremamente requisitada no mercado de trabalho.