PyForca

DOCUMENTAÇÃO DO PROJETO PYFORCA

Moisés Paz Melo dos Santos

v. 2.0.0

SUMÁRIO

1. VISÃO GERAL DO PROJETO

1.1. Nome do programa

O nome escolhido para esse projeto faz referência a linguagem principal usada para o desenvolvimento e o jogo base implementado: “Python” + “Jogo da Forca” = PyForca.

1.2. Descrição geral

O PyForca é uma versão do clássico jogo da forca desenvolvida em Python. O objetivo do projeto é recriar a experiência tradicional desse desafio, utilizando uma interface totalmente baseada no terminal.

2. ESCOPO

2.1. Objetivo

Criar uma versão do jogo da forca tradicional, com interface via terminal, modularização de código, uso de classes, leitura e persistência de dados.

2.2. Funcionalidades incluídas

- Navegação por terminal;
- Jogabilidade;
- Sistema de jogador e Pontuação;
- Gerenciamento de palavras;
- Mecânica e Estruturas.

2.3. Resultados Esperados

- Aplicação funcional, modular e bem organizada;
- Interface intuitiva e agradável em terminal;
- Demonstração clara do uso de arquivos, classes, métodos mágicos, laços e compreensão de listas;
- Estrutura compatível com os critérios de avaliação.

3. REQUISITOS

3.1. Requisitos Funcionais

Código	Requisito	Descrição
RF01	Cadastrar jogador	O jogador informa um nome, que é salvo com sua pontuação no arquivo " jogador.json "
RF02	Selecionar categoria	O jogador escolhe a categoria da palavra (ex: animal, frutas, etc)
RF03	Escolher palavra aleatória	O sistema deve selecionar uma palavra aleatória da categoria escolhida
RF04	Jogar a forca	O jogador tenta adivinhar a palavra letra por letra
RF05	Exibir boneco da forca	A cada erro o sistema deve mostrar uma nova parte do boneco
RF06	Registrar pontuação	Ao vencer, o jogador ganha pontos (xp) que são somados ao seu registro
RF07	Salvar e ler pontuação	O sistema deve lê e atualizar o arquivo " jogador.json "
RF08	Exibir ranking de pontuação	Exibir um ranking ordenado por pontuação
RF09	Exibir tela de menu	O programa deve exibir uma tela de menu (com opções de ver ranking, iniciar jogo, fazer cadastro de jogador, etc)
RF10	Usar Sons	O programa deve conter sons durante o jogo e menus

3.2. Requisitos Não Funcionais

Código	Requisito	Descrição
RNF01	Interface amigável	O jogo deve ser visualmente claro e intuitivo, feito.
RNF02	Armazenamento simples e claro	Os dados devem ser salvos em arquivos .json
RNF03	Organização modular	O código deve ser dividido em múltiplos módulos e classes.
RNF04	Uso de boas práticas	Nomes de variáveis, funções e classes deve seguir a convenção Python (PEP8)
RNF05	Execução independente	O programa deve ser executado a partir de main.py
RNF06	Tratamento de exceções	O sistema deve implementar tratamento adequado de exceções para garantir estabilidade.

4. ARQUITETURA E TECNOLOGIAS

4.1. Linguagens

A linguagem **Python** foi escolhida para o desenvolvimento integral deste projeto por se tratar principalmente de um dos requisitos definidos no trabalho que deu origem à sua proposta.

4.2. Módulos python

- os

  Módulo interno do Python que permite interagir com o sistema operacional, incluindo manipulação de arquivos, diretórios, variáveis de ambiente e comandos como limpar o terminal.

- time

  Módulo interno que oferece funções para medir intervalos de tempo, controlar pausas no programa e acessar o horário do sistema

- json

  Módulo interno usado para codificar e decodificar dados em formato JSON (JavaScript Object Notation), facilitando a leitura dos dados salvos por jogador.json e palavras.json.

- random

  Módulo interno que fornece funções para geração de números aleatórios, escolha de elementos em listas e embaralhamento de dados

- pygame

  Biblioteca externa para desenvolvimento de jogos e aplicações multimídia em Python, usada nesse projeto para a manipulação sons.

4.3. Banco de dados

Neste projeto, o armazenamento das informações é realizado por meio de arquivos **.json**, em vez de um sistema gerenciador de banco de dados tradicional. Essa abordagem foi escolhida por ser simples, legível, leve e adequada à escala do sistema proposto.

São utilizados dois arquivos principais:

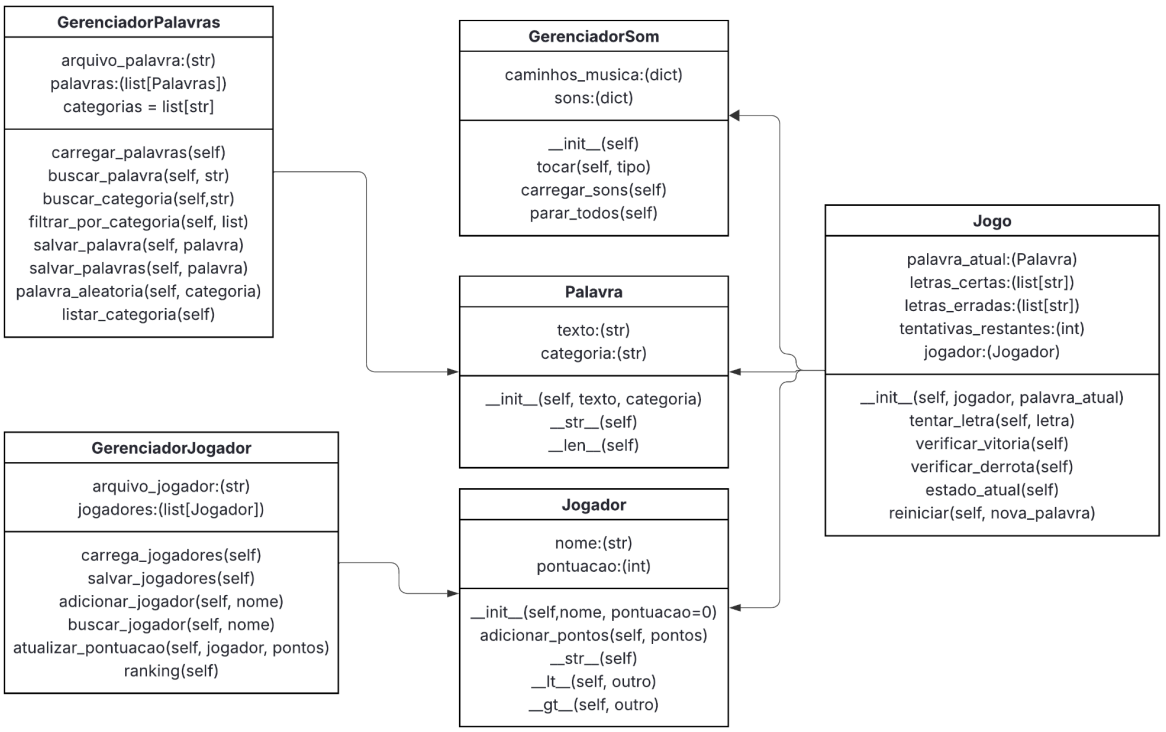
- **`jogador.json`** – responsável por armazenar os dados dos jogadores, incluindo nome e pontuação obtida nas partidas.

- **`palavra.json`** – utilizado para guardar as palavras do jogo da forca, que serão carregadas pelo programa durante a execução.

4.4. Estrutura de arquivos

```
PyForca/                                    #diretório principal
├── assets/                                 #diretório dos assets
│   └── som/
│       ├── acerto.wav.wav                  #som do acerto
│       ├── c_pixelado_menu.mp3             #música do menu
│       ├── derrota.wav                     #música da derrota
│       ├── error.wav                       #som do erro
│       └── musica_jogo.wav                 #música do jogo
├── dados/                                  #diretório dos dados
│   ├── jogador.json                        #banco de dados jogador
│   └── palavra.json                        #banco de dados palavra
├── jogador.py                              #módulo do jogador
├── jogo.py                                 #módulo do jogo
├── main.py                                 #módulo principal
├── palavra.py                              #módulo da palavra
└── som.py                                  #módulo do som
```

5. CLASSES E MODELAGENS



Classe Jogador

Responsabilidade: Representar o jogador, armazenar e atualizar suas informações.

Atributos:

- nome (str): nome do jogador.

- pontuacao (int): pontuação atual.

Métodos:

- init(self, nome, pontuacao=0): inicializa o jogador com nome e pontuação opcional.
- adicionar_pontos(self, pontos): adiciona pontos ao jogador.
- str(self): retorna uma string formatada com as informações do jogador.
- lt(self, outro): define a comparação por pontuação (menor que), usado para ranking.
- gt(self, outro): define a comparação por pontuação (maior que), também utilizado para ranking.

Classe GerenciadorJogador

Responsabilidade: Gerenciar o cadastro, carregamento, busca, atualização e persistência dos jogadores.

Atributos:

- arquivo_jogador (str): caminho do arquivo JSON que armazena os jogadores.
- jogadores (list): lista de objetos Jogador carregados do arquivo.
- dados_sobrescreveu (int): flag de controle para escrita de dados.

Métodos:

- init(self): inicializa o gerenciador e carrega os jogadores a partir do arquivo.
- carregar_jogadores(self): lê os dados do arquivo JSON e inicializa os objetos Jogador.
- adicionar_jogador(self, nome): adiciona novo jogador à lista e ao arquivo, se não existir.
- buscar_jogador(self, nome): retorna o objeto Jogador com o nome fornecido, se existir.
- atualizar_pontuacao(self, jogador, pontos): atualiza a pontuação do jogador.
- ranking(self): retorna os jogadores em ordem de pontuação decrescente.
- salvar_jogadores(self): salva os dados dos jogadores no arquivo JSON.

Classe Jogo

Responsabilidade: Gerenciar a lógica principal da partida, controlando o progresso, tentativas, checagem de vitória ou derrota e vínculo com jogador e palavra.

Atributos:

- palavra_atual (str): palavra que o jogador deve adivinhar, normalizada para minúsculo.
- letras_certas (list[str]): lista das letras que foram corretamente adivinhadas.
- letras_erradas (list[str]): lista das letras incorretas já tentadas.
- tentativas_restantes (int): número de erros restantes antes da derrota.
- jogador (str): nome do jogador participando da rodada.

Métodos:

- init(self, palavra: Palavra, jogador: Jogador): inicializa o jogo conectando a palavra e o jogador; ajusta valores iniciais do estado da partida.
- tentar_letra(self, letra): processa uma tentativa de letra, valida entrada, atualiza listas e verifica vitória ou derrota.
- verificar_vitoria(self): confere se todas as letras da palavra foram adivinhadas corretamente.
- verificar_derrota(self): verifica se o número de tentativas acabou; determina fim do jogo por derrota.
- estado_atual(self): retorna uma string mostrando o estado atual da palavra, com letras descobertas e espaços em branco.
- reiniciar(self, nova_palavra): reinicia o estado de partida com uma nova palavra, resetando tentativas e listas de letras.

Classe Palavra

Responsabilidade: Representar uma palavra do jogo e associá-la à sua categoria.

Atributos:

- texto (str): texto da palavra a ser adivinhada.
- categoria (str): categoria temática da palavra.

Métodos:

- init(self, texto, categoria): inicializa a palavra e sua categoria.
- str(self): retorna uma string formatada com a palavra e categoria.
- len(self): retorna o tamanho da palavra (número de caracteres).

Classe GerenciadorPalavras

Responsabilidade: Gerenciar o cadastro, busca, filtragem, sorteio e persistência das palavras do jogo e suas categorias.

Atributos:

- arquivo_palavra (str): caminho do arquivo JSON que armazena as palavras.
- palavras (list): lista de objetos Palavra carregados do arquivo.
- categorias (list[str]): lista de categorias cadastradas.

Métodos:

- init(self): inicializa o gerenciador e carrega as palavras do arquivo.
- carregar_palavras(self): lê e processa o arquivo JSON para popular a lista de palavras e categorias.
- buscar_palavra(self, palavra): retorna o objeto Palavra correspondente ao texto informado, se existir.

- buscar_categoria(self, categoria): retorna uma palavra de determinada categoria, se existir.
- salvar_palavra(self, palavra): adiciona uma nova palavra à lista caso não exista e salva no arquivo.
- filtrar_por_categoria(self, lista, categoria): retorna lista de palavras filtradas por categoria.
- palavra_aleatoria(self, categoria): retorna uma palavra aleatória da categoria informada.
- salvar_palavras(self): salva a lista de palavras atualizada no arquivo JSON.

Classe GerenciadorSom

Responsabilidade: Gerenciar o carregamento, execução e parada dos sons do jogo utilizando o módulo pygame.

Atributos:

- caminhos_musica (dict): dicionário com nomes e caminhos dos arquivos de som utilizados no jogo.
- sons (dict): dicionário de objetos pygame.mixer.Sound, contendo os sons carregados para uso durante o jogo.

Métodos:

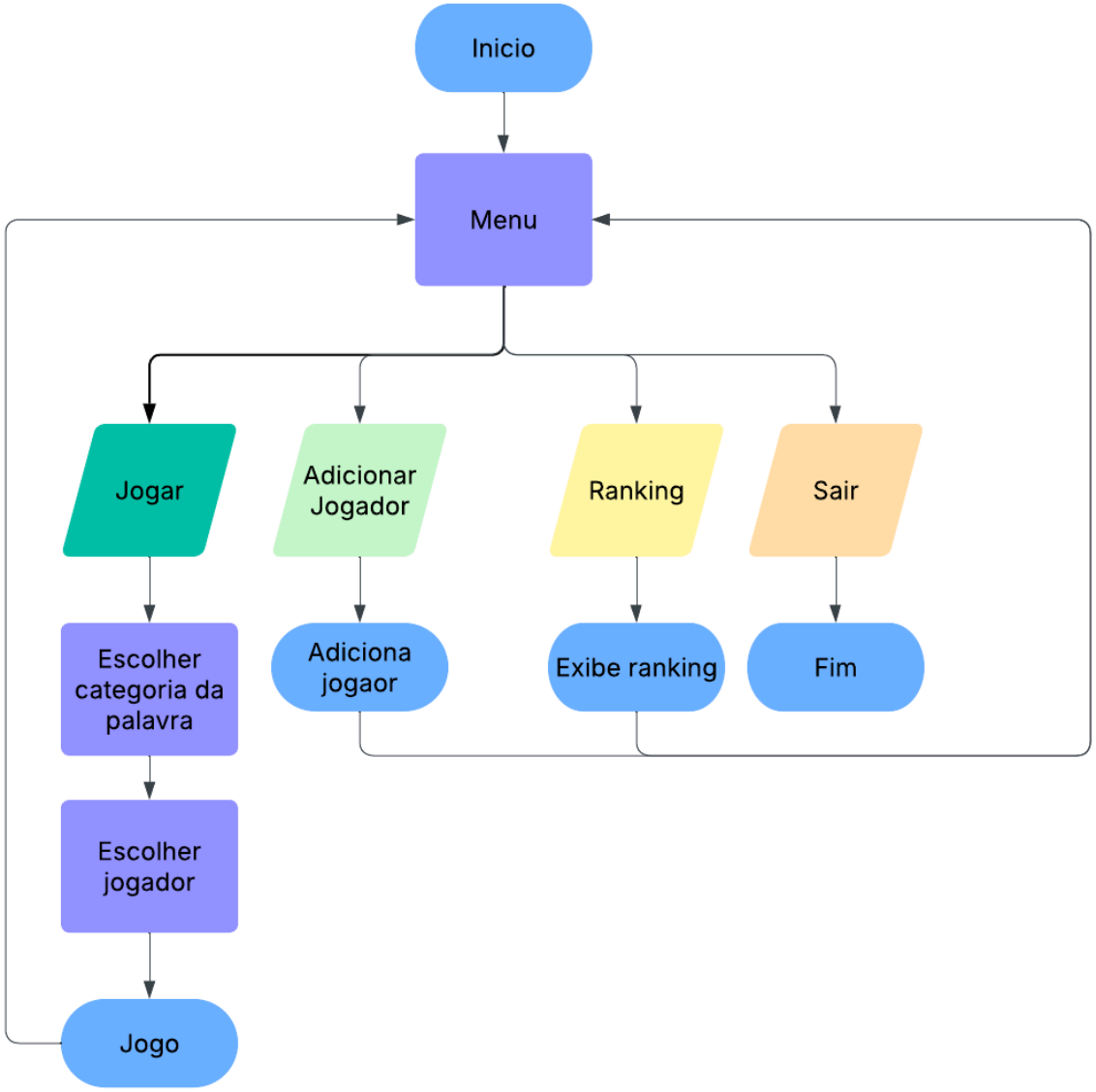
- init(self): inicializa o sistema de áudio, define caminhos dos arquivos de som e carrega os sons para uso.
- carregar_sons(self): percorre os caminhos de música, carrega cada som e armazena no dicionário de sons.
- tocar(self, nome): toca o som correspondente ao nome informado, caso esteja entre os sons carregados.
- parar_todos(self): para todos os sons atualmente em reprodução.

Módulo “Main.py”

O módulo main.py é o ponto de entrada do sistema PyForca, responsável por orquestrar toda a execução do jogo. Ele gerencia o fluxo principal, apresentando o menu inicial, controlando a escolha do jogador, categoria e palavra, bem como executando a lógica de partida, exibição de letras, tentativas, ranking, inclusão de novos jogadores e interação com sons e arquivos de dados.

Além disso, main.py coordena a comunicação entre os módulos de lógica (jogo, jogador, palavra, som), realizando operações como inicialização, chamada de métodos principais e persistência de dados ao encerrar a aplicação.

Fluxo de navegação simplificado



6. COMO EXECUTAR

OBS: Para executar você deve ter o pygame instalado via pip

1 - esteja no diretório raiz do projeto
2 - execute o módulo principal → python main.py