Desenvolvimento de um Quiz Configurável

Marcos Vinicius Dias Oliveira[1]

[1]Departamento de Tecnologia – Universidade Estadual de Feira de Santana (UEFS)
Feira de Santana – BA – Brasil

marcosvdiaso@gmail.com

1. Introdução

O *AskMe* é um jogo de quis configurável, desenvolvido como parte do processo seletivo da *Intrusa Games*, uma desenvolvedora independente focada em jogos educativos. A proposta do jogo é oferecer uma experiência dinâmica e adaptável, permitindo que os jogadores escolham entre três modos de jogo: 1) número de questões fixas, 2) limite de tempo e 3) tente não errar. Cada modo possui regras específicas que desafiam os jogadores de formas diferentes, promovendo aprendizado e entretenimento.

Este relatório tem como objetivo descrever o processo de desenvolvimento do AskMe. O sistema foi projetado para incluir funcionalidades como configuração personalizada das questões, modos de jogo configuráveis, exibição de dicas limitadas e um *Hall* da Fama persistente, que registra os melhores resultados de cada modo de jogo. O sistema permite que projetistas configurem diversos aspectos, como a quantidade de dicas disponíveis e o número de questões exibidas. As decisões de design e implementação buscam equilibrar a jogabilidade, usabilidade e modularidade, garantindo que o jogo seja funcional e expansível.

2. Metodologia

Nesta seção, serão detalhados os requisitos e funcionalidades do jogo, conforme o que foi solicitado no problema original, novas solicitações, as discussões realizadas durante as sessões tutoriais e com base nos sistemas e ferramentas utilizados na elaboração do projeto. Serão abordadas as necessidades identificadas, as soluções propostas e como essas soluções foram implementadas para atender aos requisitos estabelecidos.

2.1. Ferramentas

O sistema foi desenvolvido utilizando a linguagem de programação *Python* 3.12.3, com o *Visual Studio Code* 1.95.3 como Ambiente Integrado de Desenvolvimento (*IDE*) e o sistema operacional *Ubuntu* 24.04.01 *LTS*.

2.2. Requisitos

O desenvolvimento do jogo *AskMe* exigiu o cumprimento de alguns requisitos:

- O Jogo deve permitir três modos de jogo configuráveis:
    - Número de questões fixas: o jogador responde a um número predefinido de questões e a pontuação é determinada pelas respostas corretas.

    - Limite de tempo: o jogador responde às questões dentro de um tempo limitado, a pontuação é baseado no tempo restante após acertar todas as questões.
    - Tente não errar: o jogador responde até errar ou esgotar as questões disponíveis, encerrando o jogo no primeiro erro.
- O banco de questões (arquivo) deve ser configurável, permitindo que projetistas definam categorias, textos, opções de resposta, respostas corretas, valores de pontuação e icas associadas.
- O *Hall* da Fama deve registrar os melhores resultados de cada modo de jogo e ser persistente, permanecendo disponível mesmo após o reinício do programa.
- Ao final de cada partida o programa deve verificar se o jogador atingiu uma pontuação que lhe permite entrar no *Hall* da Fama.
- O jogador deve ter acesso a dicas:
    - Consultar dica textual;
    - Pular questão;
    - Eliminar opções incorretas.
    - Quando o jogador for acertando questões, deve ganhar novas cargas de dica para usar.
- O jogo deve permitir a personalização da quantidade de questões e dicas iniciais em cada modo de jogo.

  Todos os requisitos foram totalmente cumpridos.

2.3. Modularização e Bibliotecas

Inicialmente, o programa não estava completamente modularizado. No entanto, conforme o desenvolvimento avançava, tornou-se evidente a necessidade de organizar o código de maneira modular. Essa abordagem foi adotada para torná-lo mais legível, facilitar sua manutenção e permitir futuras expansões. Para isso, foi criado um arquivo separado, chamado *funcs.py*, onde foram armazenadas todas as funções do programa. Ao todo, o programa possui 18 funções, classificadas da seguinte forma:

- 6 funções relacionadas aos modos de jogo, que implementam as lógicas específicas de cada modo e também mecânicas gerais compartilhadas entre os modos.
- 4 funções para o *CRUD* (*Create, Read, Update, Delete)* das questões, permitindo total gerenciamento do banco de questões.
- 2 funções para configuração dos modos de jogo, permitindo personalizações como número de questões e ajudas disponíveis.
- 3 funções para gerenciamento dos *rankings*, incluindo ordenação, exibição e registro dos melhores resultados.

- 3 funções utilitárias, como validação de entradas e carregamento de arquivos *JSON.*

Além disso, foram utilizadas cinco bibliotecas externas: *os, random, json, threading* e *time*:

- *Os*: utilizada exclusivamente para limpar o terminal, proporcionando uma interface mais limpa e organizada durante a execução do programa.
- *Time*: utilizada apenas pela função *time.sleep()*, essencial para implementar o sistema de timer no modo de jogo com limite de tempo.
- *Random:* responsável por garantir a aleatoriedade na seleção das questões e na eliminação de opções erradas, quando esse tipo de dica for escolhida.
- *Json*: escolhida para gerenciar os arquivos devido à sua simplicidade e eficiência. Com apenas dois comandos (*load* e *dump*), foi possível carregar os dados para variáveis e salvá-los de volta nos arquivos de forma prática e direta.
- *Threading*: utilizada no modo de jogo com limite de tempo para criar um *timer* que funciona em paralelo com o *input* de resposta do usuário. Esse uso será detalhado em uma seção posterior do relatório.

A modularização e o uso dessas bibliotecas foram fundamentais para o desenvolvimento do programa, contribuindo para um código mais estruturado, funcional e de fácil manutenção.

2.4. Navegação pelos menus

O programa foi desenvolvido com uma estrutura de menus que permite ao usuário navegar entre as diferentes funcionalidades de forma intuitiva. A navegação principal é dividida em três grandes seções: Jogar, Configurar e *Ranking*. Essas opções são apresentadas no menu inicial do programa, junto com a opção de Sair. Cada seção possui submenus específicas, descritos a seguir:

1. Menu principal
    1. Jogar: direciona para o menu de modos de jogo.
    2. Configurar: permite gerenciar questões e personalizar modos de jogo.
    3. *Ranking:* apresenta o *Hall* da Fama para cada modo de jogo.
    4. Sair: encerra o programa.
2. Menu de modos de jogo:
    1. Questões fixas: inicia o modo com um número fixo de questões.
    2. Limite de tempo: inicia o modo baseado em um *timer* decrescente.
    3. Tente não errar: inicia o modo onde o jogador joga até errar ou completar todas as questões.
    4. Retornar ao menu inicial: retorna ao menu principal.
3. Menu de configuração:

1. Criar nova questão: permite adicionar uma nova questão ao arquivo de questões.
2. Visualizar questões: permite que o projetista visualize todas as questões cadastradas ou apenas uma específica com base no *id.*
3. Atualizar questão: permite editar uma questão existente.
4. Excluir uma questão: remove uma questão do banco de dados.
5. Configurar modos de jogo: personaliza o número de questões e ajudas disponíveis para cada modo.
6. Retornar ao menu inicial: retorna ao menu principal.

4. Menu de rankings:
    1. Questões fixas: exibe o ranking desse modo.
    2. Limite de tempo: exibe o ranking desse modo.
    3. Tente não errar: exibe o ranking desse modo.
    4. Retornar ao menu inicial: retorna ao menu principal.

A navegação pelos menus é controlada por entradas numéricas, que são validadas pelo programa para evitar erros e garantir uma experiência fluida. A utilização de submenus assegura que as funcionalidades sejam acessadas de forma lógica e que fique tudo organizado, contribuindo para uma experiência mais intuitiva.

2.5. Modos de jogo

O jogo possui três modos distintos, cada um com uma forma de funcionamento diferente:

1. Questões fixas: neste modo, o jogador deve responder a um número predefinido de questões. A pontuação do usuário é determinado com base no valor das questões que ele acertar, caso o usuário decida pular a questão, ele recebe apenas metade da pontuação da questão e caso erre, não recebe nada. O modo de jogo termina quando o numero predefinido de questões for atingido.
2. Limite de tempo: nesse modo, o jogador tem um tempo limitado para responder a todas as questões. O *timer* é iniciado no começo da partida, utilizando da função *time.sleep()* junto com uma variável contador, foi possível simular a passagem dos segundos. A cada resposta errada do usuário, é descontado 3 segundos de seu tempo restante. A pontuação final é calculada com base no tempo restante do usuário após responder todas as questões predefinidas. O modo de jogo encerrar após o usuário acertar todas as questões ou quando o tempo chegar a zero. A funcionalidade do timer foi implementada com a biblioteca *threading*, fazendo com que o cronômetro funcione em paralelo com os *inputs* de resposta do jogador. Nesse modo de jogo foi utilizado uma variável compartilhada entre as funções de *timer()* e a função do modo de jogo, essa variável é uma lista com apenas um elemento, onde a cada iteração da função *timer()*, essa variável é deduzida em 1. Quando essa variável chega a 0, o modo

de jogo não terá mais continuidade. Essa variável também é a responsável pela pontuação do jogador. Outro ponto importante a ser mencionado, é a utilização de uma variável "interruptor", foi criada uma variável chamada "parar_*timer*", essa variável é do tipo "*threading.Event()",* onde basicamente é uma variável que permite a comunicação entre *threads,* ela emite um sinal que outras *threads* podem receber. Nesse caso, quando o loop do jogo se encerra devido as questões terem acabado, é dado o comando *.set()* nessa variável interruptor, com isso uma das condições da função *timer()* passa a ser *False,* assim encerrando precocemente o contador.

3. Tente não errar: Nesse modo, o jogo termina assim que o jogador errar uma questão ou quando todas as questões da lista tiverem sido feitas. A pontuação é baseada no número de questões acertadas pelo usuário, com cada questão valendo 1. Caso o usuário decida pular a questão, ele não recebe pontuação alguma.

Os modos de jogo 1 e 3 tiveram aplicação relativamente simples, porém o modo de jogo 2 deu algum desafio devido à necessidade de aplicação de *threads.* Cada modo de jogo possui sua própria função específica, além disso há também uma função para selecionar questões aleatórias, onde as questões já feitas são armazenadas em uma lista, para que sempre que houver um novo sorteio de questão, seja verificado se ela já não foi feita ou não, afim de evitar repetição de perguntas.

2.6. Dicas

No programa há 3 tipos de ajudas diferentes:

1. Dica textual: exibe uma dica textual, que pode ajudar o jogador a escolher a resposta correta.
2. Eliminar duas opções erradas: remove duas alternativas incorretas, aumentando as chances do jogador acertar. Nessa opção, é feito um sorteio entre as 5 opções, para determinar quais serão as opções removidas. Para evitar que a opção correta seja removida ou que a mesma opção seja sorteada duas vezes e com isso somente uma opção seja removida, foi feita uma validação, onde caso qualquer uma dessas opções ocorra, o sorteio se repete.
3. Pular questão: pula a questão atual.

Para acessar o painel de dicas, o jogador deve digitar 0 no campo de resposta durante a exibição da questão. O painel só é acessível se o jogador ainda possuir dicas disponíveis. Além disso, as dicas são limitadas, mas podem ser conseguidas ao longo da partida: a cada três respostas corretas, o jogador ganha uma dica adicional.

2.7. *CRUD* das questões

O programa implementa funcionalidades completas de *CRUD* para o gerenciamento das questões:

1. Criar questão:

- O programa solicita que o projetista insira os valores para cada atributo da questão, tendo validações para os atributos numéricos.
- Após o preenchimento a questão é armazenada no arquivo de dados “questoes.json”

2. Visualizar questões:
   - Oferece duas opções:
     1. Visualizar todas as questões cadastradas, exibindo seus detalhes.
     2. Visualizar uma única questão específica, identificada pelo seu índice no banco de dados.
3. Editar questão:
   - Permite editar qualquer atributo de uma questão existente.
   - O projetista escolhe o índice da questão a ser modificada e, em seguida, seleciona qual campo deseja editar.
   - Após a edição, é perguntado ao projetista se quer editar mais algo, caso a resposta seja negativa, a questão atualizada já é armazenada no arquivo de questões.
4. Deletar questão:
   - O projetista insere o índice da questão a ser excluída, e o programa então apaga a questão do arquivo.

Para armazenas as questões foi utilizada uma lista de dicionários, onde cada dicionário é uma questão. Essa estrutura é armazenada dentro um arquivo *json.*

2.8. Configuração de modos de jogo

É possível configurar os modos de jogo, sendo possível configurar a quantidade inicial de dicas dos 3 modos, e a quantidade de questões do modo limite de tempo e do modo questões fixas. Para armazenar essas configurações, foi utilizado um arquivo do tipo *json*, que possui dentro dele uma estrutura de dicionários dentro de um dicionário, onde cada dicionário armazena as configurações de cada modo.

Para essas configurações foram criadas duas funções, uma para alterar o número de questões e outra para alterar o número de dicas, em ambas funções há validações, para caso o usuário tente digitar números negativos ou outros tipos de dados que não sejam *int.*

2.9. *Hall* da Fama

O *Hall* da Fama de cada modo possui 10 lugares, ao fim de toda partida é verificado se a pontuação conquistada pelo usuário é maior que a pontuação do 10º lugar do ranking, caso seja, então o usuário substitui o 10º lugar. Após a substituição é chamada uma função de ordenação, para garantir que o *ranking* esteja sempre ordenado pela pontuação de forma decrescente, para essa função de ordenação foi utilizado o método *Selection Sort*, onde é iterado por toda lista, selecionado o maior valor, e após

selecionar, ele troca de posição com o *"i"*, e então o loop se reinicia, a partir do *"i+1"*, se repetindo até todo o *ranking* estar ordenado. Após o *ranking* estar ordenado, o arquivo *ranking.json*, é atualizado com o novo *Hall* da Fama. Esse arquivo basicamente consiste em uma estrutura de uma lista principal que possui outras 3 listas dentro dela, cada uma para um modo de jogo, e cada lista possui 10 dicionários, cada um representando um lugar do *ranking*.

O *Hall* da Fama de cada modo pode ser visualizado através do programa. Onde é exibido os 10 lugares ordenados, com seus respectivos nomes e pontuações.

2.10. Arquivos

Como mencionado durante o relatório, foi utilizada a biblioteca *JSON* para o manuseio dos arquivos. Basicamente foram utilizados três arquivos no programa, um para armazenar as questões, outro para armazenar o *ranking* e um último para armazenar as configurações dos modos de jogo.

Como todas essas informações, são armazenadas em arquivos que são constantemente carregados e atualizados no programa, é possível assim obter a persistências dos dados, mesmo que o programa seja reiniciado. Assim mantendo a base de dados de questões, os dados do *Hall* da Fama e as configurações dos modos de jogo, mesmo fechando o jogo.

3. Resultados e Discussões

O programa funciona exclusivamente pelo terminal e precisa apenas do teclado. É necessário um interpretador *Python* instalado na máquina para execução do programa.

O desenvolvimento do jogo resultou em um sistema funcional e modularizado, atendendo aos requisitos definidos e proporcionando uma experiência personalizável. O programa foi estruturado de forma a dividir suas funcionalidades principais em três etapas: configuração, execução dos modos de jogo e exibição dos *rankings.*

O programa permite que os projetistas gerenciem o banco de questões de forma eficiente utilizando as funcionalidades de *CRUD,* com validações adequadas para evitar inconsistências nos dados. Os três modos de jogo operam conforme esperado, atribuindo suas pontuações de forma correta, e com suas condições de parada sendo atendidas. O ranking está sendo corretamente organizado de forma decrescente com base na pontuação, utilizando de um método de ordenação eficiente.

O uso da modularização no desenvolvimento foi essencial para a organização e manutenção do código, principalmente pela necessidade de gerenciar múltiplas funcionalidades. A utilização da biblioteca *JSON* para gerenciamento de dados e da biblioteca *threading* para a execução do timer foram decisões importantes, que garantiram simplicidade e um bom desempenho.

4. Conclusão

Em resumo, o programa foi bem-sucedido, todos os requisitos foram atendidos e não foram encontrados *bugs* ou erros que prejudiquem o funcionamento do programa. Todas as funções do jogo funcionam adequadamente.

Alguns desafios enfrentados durante o desenvolvimento incluíram a criação do modo de jogo Limite de Tempo, que inicialmente apresentou dificuldades na atribuição da pontuação e na sincronização da exibição do cronômetro com a entrada de respostas. Mas no fim, o modo de jogo ficou com uma boa apresentação, embora pudesse ter sido melhorada com a utilização de uma interface gráfica.

5. Referências Bibliográficas

Documentação da biblioteca *json*, disponível em: https://docs.python.org/3/library/json.html; Acesso em 02 de Novembro de 2024

HowToDoInJava, *Python – Append to JSON File,* disponível em: https://howtodoinjava.com/python-json/append-json-to-file/; Acesso em 02 de Novembro de 2024

Documentação da biblioteca *threading,* disponível em: https://docs.python.org/3/library/threading.html; Acesso em 16 de Novembro de 2024

PythonForTheLab, *Handling and Sharing Data Between Thread,* disponível em: https://pythonforthelab.com/blog/handling-and-sharing-data-between-threads/; Acesso em 03 de Dezembro de 2024

StackOverflow, "Python Threading with Event object", disponível em: https://stackoverflow.com/questions/18485098/python-threading-with-event-object; Acesso em 03 de Dezembro de 2024