AskMe: O Grande Quiz da UEFS

João Marcus R. da Silva

[1]Bacharelado em Engenharia de Computação
Universidade Estadual de Feira de Santana (UEFS)
Av. Transnordestina, s/n, Novo Horizonte, Feira de Santana – BA, Brasil – 44036-900

joao.marcus045@gmail.com

Abstract. *This report describes the development process of a question and answer game in the latest version of the Python interpretive language, an idea leveraged by the search for talented young people for an internship, carried out by the educational game developer Intrusa Games. The project involves the creation of a Quiz, motivating students in the computer science areas. The offering provides conditional structures, loops, dictionaries, libraries, with a main focus on files, in addition to integrating functions throughout the software. The concepts applied were detailed in this document, informing their usefulness in the game's tasks. Finally, the game experience becomes real, going through the accumulation of points to occupy the Hall of Fame.*

Resumo. *Este relatório descreve o processo de desenvolvimento de um jogo de questões e respostas na última versão da linguagem interpretativa Python, ideia alavancada pela busca de jovens talentosos para um estágio, realizado pela desenvolvedora de jogos educativos Intrusa Games, o projeto envolve à criação de um Quiz, motivando estudantes nas áreas computacionais. A codificação utilizou estruturas condicionais, loops, dicionários, bibliotecas, com foco principal em arquivos, além de integrar funções por todo o software. Os conceitos aplicados foram detalhados neste documento, informando a sua utilidade nas tarefa do jogo. Por fim, a experiência do jogo se torna real, visando o acumulo de pontos para ocupar o Hall da Fama.*

1. Introdução

É inegável que os jogos estão na cultura do mundo inteiro por muito tempo, mas pouco se sabe sobre o uso de jogos para entretenimento em sociedades antigas, onde existem registros na Suméria, Mesopotâmia, Egito, além das culturas clássicas de gregos e romanos. Segundo sociólogo holandês Johan Huizinga, o jogo não é apenas uma forma de entreter mas é algo que transcende as necessidades biológicas e possuindo um objetivo e significado social e cultural[Juchem, H. 2018]. Partindo desse pressuposto, entende-se que os jogos representam conhecimento sobre diversos áreas do mundo, contribuindo não só para o desenvolvimento cognitivo e sociais aprimorando o pensamento lógico, a resolução de desafios, a capacidade de fazer escolhas, e o processo criativo.

Por essa perspectiva a desenvolvedora de jogos digitais educativos Intrusa Games abriu uma seleção de estágio para jovens, que consiste na produção de um jogo de perguntas e respostas, onde necessitou de algumas regras para uma melhor avaliação, como por exemplo, o jogo deve ser criado utilizando arquivos como base principal, além de possuir três diferentes modos:

- Modo 1: Número de questões fixas;
- Modo 2: Limite de tempo;
- Modo 3: Tente não errar.

O 1º modo, possui 15 questões escolhidas aleatoriamente pelo software, sua pontuação é atribuída a cada acerto, onde cada questão possui um valor de 10, 20 ou 30 pontos. A partida termina após responder todas as perguntas. O 2º modo parte da mesma base de perguntas aleatórias do primeiro, onde difere na adição de um cronômetro com uma duração de 5 minutos e na atribuição de pontos pelo tempo restante. O jogo se encerra após o temporizador zerar, se todas as questões não tiverem sido respondidas o jogador perde o jogo. No 3º modo, todas as questões do jogo são apresentadas, e o objetivo principal é responder o máximo possível sem errar. O jogo termina quando o jogador comete um erro ou responde corretamente todas as questões.

Foi comunicado que os jogadores podem solicitar três tipos de auxílio, ”Dica sobre a questão”, ”Pular a questão”, ”Remover alternativas incorretas”, o usuário possui apenas um ponto por cada ajuda, durante a partida a cada 60 pontos, pode ser adicionado um ponto para os auxílios que já tiverem sido utilizados. Ao termino de cada partida, o código deve conferir se o jogador possui pontos ou tempo suficientes para entrar no Hall da Fama, caso possua, é introduzido na posição designada pela quantia de pontos ou restante de tempo de forma decrescente de acordo com o modo escolhido. Este Hall deve ser salvo em um arquivo para cada modo, além de poder ser consultado pelo jogador mantendo as informações salvas até mesmo quando o computador for reiniciado, cabe esclarecer que o arquivo utilizado foi em formato ”.txt”. O jogo deve exibir também a opção de jogar novamente, onde de acordo com a decisão a partida é reiniciada. Tomando aspectos técnicos como base, o código em linguagem interpretativa *Python* na versão mais recente, deve ser bem estruturado, apresentando estabilidade, resistência e segurança, prevenindo falhas e fácil modularização, permitindo alterações no futuro. Nessa perspectiva, fatores como desenvolvimento, metodologia, referências, manual, testes e erros estão descritas neste relatório.

Como forma de resolução deste problema apresentado, algumas perguntas necessitam de resposta: 1) Como inserir os três modos de jogo e criar as perguntas; 2) Como ler o arquivo de questões e apresenta-las aleatoriamente; 3)Como fazer as verificações para cada entrada; 4) Como implementar as ajudas; 5) Como cronometrar o tempo enquanto exibe as perguntas; 6)Como computar as respostas ; 7) Como ordenar o Hall da Fama por meio da pontuação ou restante de tempo; 8)Como criar e apresentar o Hall da Fama.

Soluções apresentadas para as questões acima:

1. Em uma função principal estão todas as funções iniciais do código;
2. Uma verificação de arquivo é feita antes de iniciar o jogo;
3. É criado um menu exibindo opções de modo de jogo e consulta;
4. É recrutada a função responsável pelo modo escolhido;
5. Mais funções são criadas para cada execução específica no código;
6. Os modos estão em *loops* , unidos com outras funções;
7. Utiliza-se manipulação de arquivo para a leitura das questões;
8. É cria-se uma lista de questões aleatórias pelo programa;
9. Em cada entrada do código é atribuída uma função de verificação;
10. As funções de ajuda são recrutadas com a escolha do jogador;

11. No modo 2, o cronômetro é atribuído por meio de bibliotecas;
12. A cada final de partida é verificado a pontuação do jogador;
13. É ordenado os valores do jogador por método de ordenação;
14. Os valores ordenados são escritos no Hall da Fama;
15. O *loop* principal acaba se o jogador opte reiniciar a partida;
16. Se reiniciada, todos componentes do jogo são redefinidos.

2. Desenvolvimento

Por meio deste tópico, as soluções e a elaboração serão apresentadas, levando como base as questões anteriores levantas durante a introdução.

Como inserir os três modos de jogo e criar as perguntas? Para tornar possível a produção de um Quiz digital com três diferentes modos de jogo foi utilizado uma função para o menu, o usuário teria a opção de escolher entre o 1º modo: ”Número de questões fixas”, o 2º: ”Limite de Tempo” e 3º :”Tente não errar”, além disso, a escolha de acessar o Hall da Fama inserindo o número ”4”. Após a inserção do modo de jogo, o código recruta a função responsável pelo modo escolhido. Ambos os modos utilizam as mesmas dinâmicas como *”loop while*” responsável por manter a execução das outras tarefas indentadas abaixo, enquanto a verificação presente no laço de repetição retornar o valor True, porém algumas verificações e funções, diferenciam-se em alguns aspectos. Por exemplo, entre o modo 1 e o modo 3, a diferença está na verificação de questão correta, onde o ”*loop*” termina se o ponto retornado pela questão for 0. Já o modo 2 se distingue do modo 1 por incluir um cronômetro de 5 minutos e atribuir pontuações com base no tempo restante.

Ademais, com a ideia dos modos já apresentados, a criação do arquivo de questões foi realizado em grupo, os projetistas criaram perguntas com base em assuntos de matérias gerais, indo de português até mesmo esportes, após a separação de dicas, alternativas e enunciados era necessário transforma-lo em um arquivo funcional. É importante ressaltar que o modelo foi especificado pela desenvolvedora, nele existe uma lista com as questões que foram criadas em formato de dicionário, as chaves não contém aspas e são iguais, alterando somente os valores em cada pergunta:

- Questions = [category: ”Assunto da questão.”,
- value: ”Valor da pontuação.”,
- questionPath: ” Uma imagem (opcional).”,
- questionText: ”Enunciado.”,
- option1: ”Alternativa 1 da possível resposta.”,
- option2: ”Alternativa 2 da possível resposta.”,
- option3: ”Alternativa 3 da possível resposta.”,
- option4: ”Alternativa 4 da possível resposta.”,
- option5: ”Alternativa 5 da possível resposta.”,
- answer: ”Texto da Alternativa correta.”,
- explanation:”Explicação sobre a questão.”,
- hint:[”Textos com dicas para o jogador.”],
- category:”...,]

Utilizando esta formatação foi possível projetar um banco de questões para um limpo funcionamento do código.

Como ler o arquivo de questões e apresenta-las aleatoriamente?Durante a maioria do desenvolvimento do código, como parte principal, se fez necessário o conceito de manipulação de arquivo, foram utilizados por meio do comando "*open()*", que permite abrir arquivos em diferentes modos, como leitura e escrita . Para acessar o conteúdo deste programa foi importante o uso da biblioteca os, esta permite interagir com o sistema operacional, além de executar comandos diretamente no terminal, utilizada em conjunto com o "*try-except*" para tratamentos de erro, evitando quebrar o código por tentar interagir com arquivos inexistentes. A biblioteca mencionada já integrada ao *Python* necessitando somente da sua importação, ela também serviu para apagar o terminal utilizando a função *"os.system("cls")*," cabe ressaltar que a mesma só funciona para o sistema operacional Windows, para o SO Linux é necessário a modificação do "*cls*" para "*clear*", assim o código retoma sua ação normalmente evitando erros [Python Software Foundation 2024]. O conteúdo do arquivo possui formato de uma lista contendo dicionários, onde as chaves do mesmo estão sem aspas, para a resolução desta modificação foi designada a uma função chamada "Formatacao" utilizando o comando "*replace*", um método de manipulação em "*strings*" usado para substituir partes de uma "*string*" por outra [Python Software Foundation 2024]. Para tornar possível a interação da lista de dicionários recém modificada, foi usado o comando "*eval*" com o "*.join()*" e o "*split*", a junção separou o conteúdo lido como "*string*" do arquivo, tornando-o executável com o formato de uma lista.

Referindo-se sobre a escolha aleatória de questões, foi utilizada como solução mais apropriada o uso de uma biblioteca já integrada à linguagem Python chamada de *random, a mesma por já fazer parte da linguagem utilizada, exige somente sua importação no programa.* Essa biblioteca permite gerar números aleatórios com o comando "*random.randint()*", usando dois parâmetros: início e limite [Python Software Foundation, 2024]. Com base nisso, foi criada uma função para determinar a lista contendo o índice das questões, usando o comando citado e a quantidade de questões como o limite, que ao retornar a lista preenchida para o modo definido, tornou possível a escolha das questões.

Já se tratando da apresentação das questões, foi criada uma função chamada "ImprimeQuestao", esta recebe a lista do banco de perguntas "Quest", a lista de questões escolhidas e a lista "*P_mod*" em formato "*string*" contendo o nome das chaves do dicionário, através disso foi possível exibir somente os valores desejados de cada questão utilizando verificações e alterando valores, focando apenas em mostrar as perguntas e as opções.

Como fazer as verificações para cada entrada? As verificações foram importantes por todo o código, tornando-o mais seguro e com menos falhas. Uma das funções foi a "V_menu" onde o usuário deve inserir a opção desejada, entre 1 e 4, a depender da sua escolha o programa irá recrutar a função específica. Para a continuidade do código a "V_tipo" foi criada, esta aceita somente números de 1 á 3, sendo utilizada principalmente para exigir o modo para o jogador e o tipo de ajuda, outra bastante usada foi a "Vresp" a mesma permite a entrada de números de 1 á 6, ela foi responsável por pedir a resposta das questões ao usuário. Para a entrada do nome do jogador no Hall da Fama, foi criada a função "Vnome", esta evita falhas de entradas incorretas. Ambas as verificações acima fizeram necessário a utilização do "*try-except*" comentado anteriormente. Conforme a execução do software, algumas outras verificações são utilizadas para atribuição de val-

ores, como a função "V_ajuda" que valida a quantidade de pontos de auxílio que o usuário possui, permitindo ou não utilização da assistência escolhida. A função "V_Valor_Antigo" verifica a menor pontuação do Hall da Fama, comparando com a nova pontuação do jogador, permitindo ou não o jogador no Hall. As verificações são de extrema importância para um bom funcionamento do código, seja para monitorar a entrada, modificar valores ou permitir a continuação do "*loop*" de cada modo.

Por meio de condições foi possível dar estrutura ao AskMe, a maioria das funções trazem a lógica, onde se as verificações forem favoráveis retornam um valor *booleano True*, tornando possível a interação das perguntas e funções presentes no jogo, se não for favorável retornam um valor *booleano False*, interrompendo a próxima ação da questão. É importante falar que as funções de verificação estão presentes em boa parte das outras tarefas no código, como na ajuda, apresentação de questões, entrada de dados e continuidade da partida, reduzindo a taxa de falhas durante a execução do código.

Como computar as respostas? É de conhecimento geral que as perguntas são acompanhadas de respostas, partindo disso, durante a execução do programa, após a exibição das questões, a função "Respostas" é recrutada, esta foi criada utilizando um "*loop while*" e condições que chamam outras funções, como por exemplo as funções de entrada com verificação para a resposta e o tipo de ajuda. Além disso, a principal função de ajuda também se faz presente, ela agrupa outras funções de verificação e tem a capacidade de recrutar outras tarefas permitindo que o usuário utilize os recursos de assistência no jogo e um melhor funcionamento do código. Outra função inserida no *loop* comentado é a "V_Questao" que recebe as alternativas e a opção escolhida pelo usuário, essa tarefa executável compara a resposta correta com a escolha do jogador e retorna pontos por cada acerto, dependendo do valor definido de cada pergunta, variando de 10, 20 ou 30 pontos. Esta função permite que as próximas interações com o jogo se tornem possíveis, seja em estabelecer as pontuações ou o término de uma partida. Se tratando do modo 1, os acertos são computados normalmente, no caso do modo 2 os pontos são utilizados para gerar pontos de auxílio, além de repetir questões que o usuário errou, já o modo 3, se o valor de "p" for retornado como "0" o *"loop* while" é encerrado por meio de valores booleanos, computando a pontuação do jogador.

Como implementar as ajudas? Foi solicitado pela desenvolvedora Intrusa Games 1 ponto para cada 3 tipos de ajuda para o jogador em qualquer modo durante as partidas, poderia usar quantas ajudas estivessem disponíveis em uma única pergunta. Além disso, o jogador recebe 1 ponto de assistência adicional a cada 60 pontos acumulados durante a partida, mas somente se seus pontos de auxílio não estiverem no limite máximo. Tomando essas regras como base, foi possível implementar as assistências através de *condições*, *listas* e *loops* inseridos em funções. Se o jogador responder "6", o software exibe os tipos de assistência disponíveis, que a depender da escolha do jogador, o código verifica se o tipo de assistência selecionado possui algum ponto restante na lista "*aux*", esta guarda os pontos de cada tipo de ajuda. Caso haja pontos, à assistência escolhida é executada. Se não houver pontos disponíveis, o software informa que o auxílio não é possível e solicita novamente a resposta da questão ao jogador. De acordo com a decisão do usuário e a disponibilidade da assistência escolhida, se o tipo for "1", é exibido a dica da questão, se for "2" o software informa que pulou a pergunta, apaga o terminal e exibe outra, caso o tipo seja "3", o terminal é apagado e a função "*ImprimeSemErro*" é

recrutada, esta verifica qual é a questão correta, compara com as alternativas disponíveis e retira duas questões incorretas. Ambos os tipos de auxílios exibem na tela após o seu uso a lista atualizada de ajudas disponíveis "*aux*".

Sobre a atribuição de uma assistência para cada vez em que o jogador atingir 60 pontos, foi utilizada uma função em ambos os modos chamada "*C_pontos*", ela acumula os pontos de cada questão por meio de um contador, que ao verificar se a meta foi atingida, exibe no terminal, solicita em qual auxílio deseja inserir ponto e acaba sendo reiniciado logo após as interações. A situação comentada anteriormente só é apresentada na forma de entrada para o usuário se houver o número "0" na lista "*aux*", caso o contrário, apenas exibe que o jogador deve gastar suas ajudas para receber mais, perdendo o ponto adicional.

Como cronometrar o tempo enquanto exibe as perguntas? Conforme já discutido anteriormente, sobre o 2º modo do jogo, se faz necessário o uso de um cronômetro, este é responsável não só pelo funcionamento da partida, mas pela pontuação ou termino do jogo. A função atribuída a este *timer* chama-se "*Cronometro*", mas para que o código pudesse exibir as questões, computar o tempo e as respostas de forma simultânea, foi utilizada a biblioteca "*time*" utilizando a função "*time.time()*" que é usado para obter a marca de tempo atual, ela pode retornar o número em segundos, com fração decimal, mas foi implementada no software para medir a duração do cronômetro. A outra biblioteca utilizada foi a "*threading*" com as funções "*event*", onde permite que uma *thread* espere até que outra sinalize que uma ação deve ocorrer, "*start()*" que inicia a *thread* do cronômetro, "*set()*" para ativar evento criado, "is set" para verificar se o evento está ativo ou inativo, "*join()*" que aguarda a *thread* terminar antes de continuar o programa principal [Python Software Foundation 2024]. Essa biblioteca permite que múltiplas partes de um programa sejam executadas de maneira aparentemente simultânea, aproveitando melhor os recursos do sistema. A função "*time.sleep( )*" da biblioteca time também foi usado permitindo criar *delays* durante a execução do código. Ambas as bibliotecas já pertencem ao Python exigindo somente a sua importação no programa, além de terem sido utilizadas em conjunto como alternativa mais adequada[Python Software Foundation 2024]. A tarefa executável chamada "*InicioCronometro*" fez com que a variável "*tmp sobra*" se tornasse global, onde após a *threading* poderia acessar seu valor para computar a pontuação.

Como ordenar o Hall da Fama por meio da pontuação ou restante de tempo? Posteriormente, com as pontuações já estabelecidas e com a vaga no Hall da Fama já disponível, o jogador é inserido na lista dos jogadores que é recebida pela função que utiliza um método de ordenação chamado "InsertionSort", por meio desta foi possível ordenar os jogadores da lista de forma decrescente utilizando seus pontos ou o tempo restante de acordo com o modo selecionado. Por se tratar de uma lista pequena e pré ordenada foi a mais viável e mais simples implementação, realizando menos comparações e otimizando o trabalho. Como pode ser observado acima, a figura demonstra parte da função "*InsertionSort*" no que diz respeito a ordenação por pontos, trata-se dos modos 1 e 3, onde o Hall da Fama é caracterizado pela pontuação de cada jogador. Já relacionado ao modo de "Limite de Tempo", a única alteração na função acima é a verificação do modo e modificação da palavra "*pontos*" para "*tempo*", ordenando normalmente a lista de jogadores e retornando-a.

Como criar e apresentar o Hall da Fama? Por se tratar de um jogo, a sensação de competitividade e a apresentação de pontos é essencial, então para isso foi solici-

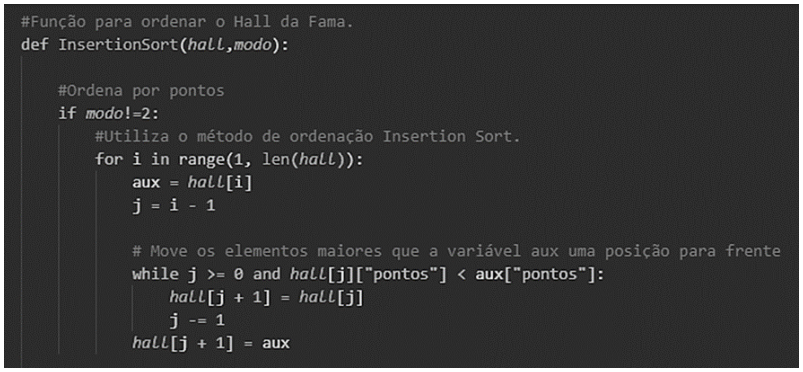

Figure 1. Parte da função de ordenação

tado pela desenvolvedora que o AskMe possuísse para cada modo, um Hall da Fama, onde seriam salvos em arquivos ”*.txt*”, cabe ressaltar que para cada modo, o Hall assume nomes diferentes no software, evitando modificações incorretas. Sobre o encerramento de uma partida o *loop* principal do modo é interrompido, e logo após chama-se a função ”*GameOver*”, esta imprime a pontuação final do jogador e recruta uma função para verificar a última pontuação do Hall da Fama permitindo ou não que o jogador ocupe uma posição.

Através disso, foi implementado ao código uma verificação responsável para identificar se já existe algum arquivo criado, se não houver, o software preenche uma lista com dicionários contendo as chaves ”nome”, ”pontos” ou ”tempo” a depender do modo de jogo, além dos valores zerados. Envia essa lista para uma função que faz a escrita somente dos valores, salvando as informações do Hall em um arquivo específico do modo escolhido. Caso já exista ou tenha sido criado o arquivo recente, o código recruta uma função que carrega os valores do arquivo retirando virgulas e espaços atribuindo-os como um dicionário, solicita o nome do usuário inserindo na lista com a sua pontuação específica por meio de uma função que atualiza o Hall da Fama, logo após, envia a lista para outra tarefa ordenando os valores utilizando a função ”InsertionSort”, retornando a lista com fatiamento ”[:10]” para que fique apenas com 10 jogadores, assim como foi pedido. As duas imagens acima retratam o Hall da Fama do modo ”Limite de Tempo”

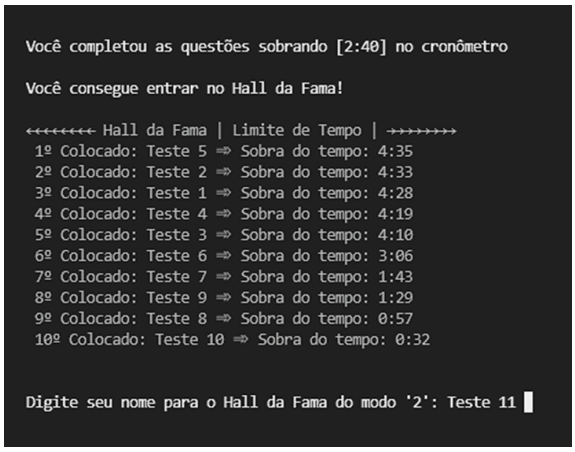

Figure 2. Hall da Fama modo 2

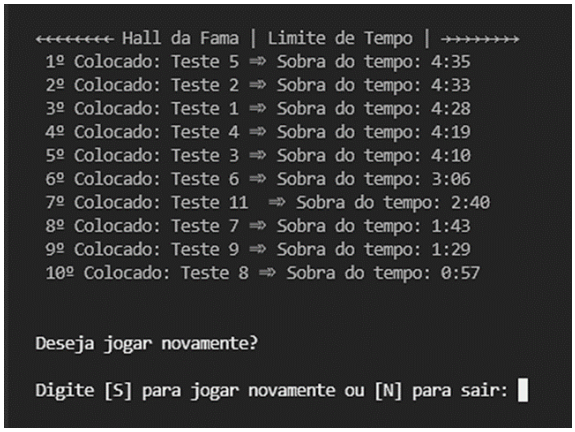

Figure 3. Hall da Fama modo 2 atualizado

tomando o nome ”Teste 11” como exemplo, a primeira figura demonstra a entrada de mais um jogador no Hall com as 10 posições já ocupadas, já a segunda imagem mostra o mesmo Hall após a atualização das posições, modificando a posição que pertencia ao ”Teste 10”.

Na visão da impressão do Hall, foi utilizado em conjunto a função que carrega a lista em formato de dicionário e a tarefa que exibe o Hall da Fama de acordo com o modo de jogo, onde possui apenas 10 posições. Todo final de partida em que o jogador consegue alguma posição, é exibido o Hall antes e depois de inseri-lo. Porém se tratando do menu no início do jogo, foi utilizado as mesmas funções citadas anteriormente adicionando somente a verificação de existência do arquivo no modo escolhido, onde só seria exibido se sua existência fosse confirmada. Ao final de todas as situações, seja na consulta ao Hall pelo menu ou ao final de cada modo, a oportunidade de jogar novamente ou encerrar o programa é apresentada, caso o usuário escreva a letra ”*S*” o menu é exibido novamente e todos os componentes são reiniciadas com exceção do Hall, se o jogador pressionar ”*N*” o ”*loop*” principal é encerrado e o jogo é finalizado.

3. Manual de uso e Testes

3.1. Manual de uso

Para rodar o programa, você precisará ter um interpretador Python instalado em sua máquina. Siga estas etapas para executar o jogo:

1. Inicie o interpretador Python e acesse o arquivo..
2. Execute o código-fonte por meio do interpretador.

No início do jogo, há uma interface simples onde o jogador deve inserir o modo de jogo ou consultar o Hall da Fama. Caso seja selecionado algum modo o jogo é iniciado, o objetivo é responder questões e conseguir pontos e ocupar o Hall da Fama. Assistências são possíveis dentro do limite de pontos. Ao final de cada partida, o sistema verifica se o usuário pode ocupar o Hall solicitando seu nome, logo após, pergunta se o jogador deseja jogar novamente. Se a resposta for afirmativa, o jogo reinicia. Caso o jogador opte por não continuar, o jogo será encerrado. Se a consulta ao Hall da Fama for selecionado, o software pergunta o modo e o exibe, em seguida ou caso o arquivo do Hall ainda não exista, a opção de jogar novamente já mencionada é apresentada.

3.2. Testes

Levando em consideração a variação do nome ”Teste”, para cada teste do software. Apresentação da primeira interface do programa, onde o usuário irá informar o sua es-

Figure 4. Menu principal

colha, dando início ao jogo ou exibindo o Hall da Fama. Representada na imagem acima. Considerando o fim de partida do modo 3 causado por um erro de questão, observa-se o

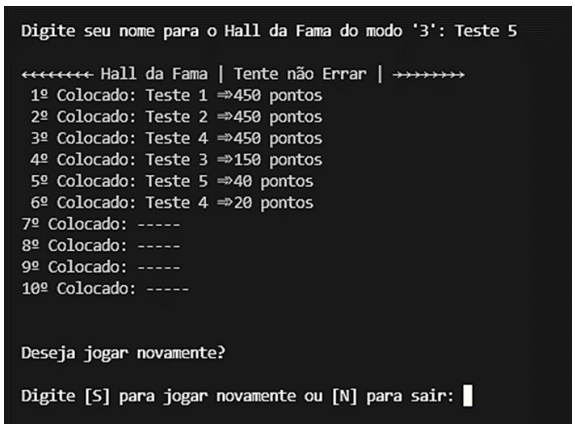

Figure 5. Entrada no Hall da Fama do modo 3

Hall da Fama na imagem acima que o usuário possui duas opções, essas determinam se o jogador irá reiniciar a partida ou sair do jogo. Além disso, esta figura demonstra a entrada de mais de um jogador com pontuações iguais em diferentes testes.

3.3. Erros

Em relação aos erros encontrados no código, o problema mais comum é uma falha visual, ela ocorre simultaneamente durante a finalização do cronômetro enquanto o intervalo da apresentação de outra questão está em andamento. A acima é um exemplo disso. A verificação falha e acaba permitindo partes da pergunta enquanto o cronômetro está se esgotando. Esse problema pode ser resolvido por meio de uma melhor verificação, que leve em conta as finalizações do tempo e intervalos da questão. Ademais, se tratando dos auxílios, após o jogador utilizar o tipo 3, retirando duas alternativas incorretas, ainda é possível responder utilizando as alternativas recém excluídas, assim o usuário pode errar a questão mesmo que não esteja visível a opção. É importante esclarecer que durante os testes, não ocorreu erros que poderiam interromper a execução do código, apenas algumas falhas de verificações e visuais que atrapalham a experiência do jogador.

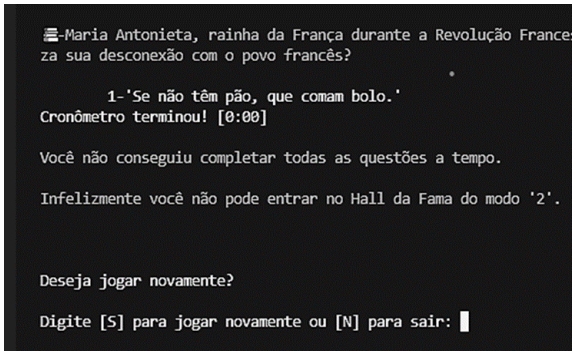

Figure 6. Exemplo de erro ao fim do cronômetro

4. Conclusão

Este relatório exibiu uma resolução computacional em Python, uma linguagem interpretativa, para criação do jogo AskMe. A abordagem desenvolvida resolve o problema proposto utilizando métodos e estruturas de dados, como listas, bibliotecas, métodos de ordenação, dicionários, e *loops*, além de arquivos para armazenar e verificar interações do jogador. O projeto inclui funções que dividem as tarefas em blocos, facilitando a compreensão do código e tornando a codificação mais organizada e modular. Por isso foi possível tornar o jogo mais interativo e atraente por meio de questões interessantes ,funcionalidades, assistências e menus. No entanto, o sistema apresenta erros que podem surgir por verificações incorretas, estas atrapalham parte da jogabilidade. Possíveis modificações no futuro poderão melhorar o tratamento desse tipo de falha, implementando um sistema de verificação que limite interações indesejadas. Isso resultará em um código com menos falhas visuais e de execução, proporcionando uma melhor experiência ao jogador.

5. Referências

PYTHON SOFTWARE FOUNDATION. Python documentation. Disponível em: https:/ /docs.python.org/. Acesso em: 13 nov. 2024.

CÓDIGO FONTE TV. Thread entenda como sua aplicação funciona Dicionário do Programador. Disponível em: https:/ /www.youtube.com/watch?v=xNBMNKjpJzM. Acesso em: 21 nov. 2024.

DEV APRENDER JHONATAN DE SOUZA. Python Arquivos TxT Em 7 Minutos FÁCIL DESAFIOS. Disponível em: https: / /www.youtube.com/watch?v=F8KB5_sEQH0. Acesso em: 07 nov. 2024.

HASHTAG PROGRAMAÇÃO. Trabalhando com TEMPO no Python - Biblioteca Datetime. Disponível em: https: / /www.youtube.com/watch?v=eBYSjP4vf_w. Acesso em: 21 nov. 2024.

JUCHEM, H. Sobre o uso de jogos no ensino de história. Revista Brasileira de Educação Básica, Ano 3, Número 7, 2018. Disponível em: https: / /rbeducacaobasica.com.br/2018/04/01/sobre-o-uso-de-jogos-no-ensino-de-historia/. Acesso em: 07 dez. 2024.

COELHO, Vânia Maria. O jogo como prática pedagógica na escola inclu-

siva. 2010. Trabalho de Conclusão de Curso (Especialização em Educação Especial - Déficit Cognitivo e Educação de Surdos, EaD) – Universidade Federal de Santa Maria, Centro de Educação, Santa Maria, RS, Brasil. Disponível em: https: / /repositorio.ufsm.br/handle/1/1485. Acesso em: 13 nov. 2024.