Sistema Gamificado para Exercícios de Física

Heitor Barcellos

25 de abril de 2025

1.Introdução

1.1 Motivação

Durante meus estudos de Física, sempre tive dificuldade em transformar o conteúdo em uma rotina consistente. Acredito que muitos estudantes compartilham dessa sensação de estresse ao passar horas seguidas resolvendo exercícios. Com o tempo, comecei a explorar diferentes formas de estudar, utilizando prompts e ferramenta de chat com inteligência artificial para tornar o aprendizado mais interativo — com desafios, recompensas e um sistema de progressão. Isso me proporcionou uma nova perspectiva sobre o estudo, transformando-o em uma atividade mais dinâmica e engajadora. A experiência foi tão marcante que decidi transformar essa ideia no meu projeto, aplicando estruturas de dados e lógica de programação para desenvolver um ambiente gamificado focado no aprendizado de Física.

1.2 Sobre a Organização do Projeto

O projeto foi desenvolvido em Python. O código está estruturado de forma modular, com múltiplos arquivos responsáveis por diferentes funcionalidades do sistema, além de arquivos JSON utilizados para armazenar dados persistentes, como exercícios e progresso do jogador.

O arquivo *main.py* gerencia a execução do programa, incluindo o menu, o fluxo e chamadas aos demais módulos. O arquivo *dados.py* é responsável pelo controle de progresso do jogador, gerenciamento de vidas, dicas, pontuação, e oferece suporte a múltiplos jogos salvos. O módulo *verificador.py* contém a lógica de verificação das respostas fornecidas. O arquivo *loja.py* gerencia o sistema de compra de pulos e dicas utilizando os pontos acumulados. Por fim, os dados de exercício e resoluções estão organizados no arquivo *exercicios.json*, permitindo flexibilidade para expansão do conteúdo e categorização futura por nível de complexidade. O código completo do projeto está disponível em: https://github.com/pbheitor/minigame-fisica/

2. Verificador de Respostas – *verificador.py*

Começando pelo coração funcional do projeto, o módulo *verificador.py* é responsável por analisar a resposta fornecida pelo usuário e compará-la com a resposta correta do exercício atual. A comparação não é feita de forma direta, pois o usuário pode digitar a resposta da forma não esperada, como letras maiúsculas, minúsculas ou com espaços extras.

```python
from exercicios import obter_resposta_correta

def normalizar_texto(texto):
    return texto.replace(" ", "").lower()

def verificar_resposta(exercicio_id, resposta_usuario):
    resposta_correta = obter_resposta_correta(exercicio_id)

    if resposta_correta is None:
        return False

    resposta_normalizada = normalizar_texto(resposta_usuario)
    correta_normalizada = normalizar_texto(resposta_correta)


    return resposta_normalizada == correta_normalizada
```

Figura 1 – (Módulo de verificação)

O conteúdo do módulo *verificador.py* está na **Figura 1**. O módulo conta com uma função auxiliar chamada *normalizar_texto()*, que formata a resposta do usuário, garantindo uma comparação entre a resposta do usuário e a resposta esperada.

A função principal *verificar_resposta()*, recebe o ID do exercício e a resposta digitada. Em seguida, busca a resposta correta no banco de dados (*exercicios.json*), aplica a normalização nos dois textos e compara os resultados.

Se as duas respostas forem iguais após a normalização, a função retorna *True*, indicando que o usuário acertou. Caso contrário, retorna *False*.

Esse módulo também garante que pequenos erros de digitação, como “50n” ao invés de “50 N”, não prejudiquem o usuário, focando na lógica da resposta e não na forma exata como foi escrita.

Em relação à complexidade, a função *normalizar_texto()* possui complexidade O(n), onde *n* é um tamanho arbitrário da string, já que percorre o texto uma vez para remover espaços e converter para minúsculas. Já a função *verificar_resposta()* também possui complexidade O(n), considerando que ela chama *normalizar_texto()* duas vezes antes e compara as strings normalizadas.

3. Gerenciamento de Progresso – *dados.py*

Aqui já começamos a falar sobre os módulos auxiliares do projeto. Esse módulo cuida de tudo que está relacionado ao progresso do usuário. É aqui que salvamos os pontos, as vidas, dicas, pulos e o nível que o jogador está. Em outras palavras, é o que garante que o jogador possa continuar de onde parou, mesmo que feche o programa.

```python
def criar_novo_jogo(nome_arquivo="progresso.json"):
    progresso = {
        "vidas": 3,
        "pontos": 0,
        "nivel": "Novato",
        "pulos": 1,
        "dicas": 0
    }

    salvar_progresso(progresso, nome_arquivo, mostrar_mensagem=False)
    return progresso


def carregar_progresso(nome_arquivo="progresso.json"):
    if os.path.exists(nome_arquivo):
        with open(nome_arquivo, "r") as arquivo:
            progresso = json.load(arquivo)

            if "dicas" not in progresso:
                progresso["dicas"] = 0
            if "pulos" not in progresso:
                progresso["pulos"] = 1
            if "vidas" not in progresso:
                progresso["vidas"] = 3
            if "pontos" not in progresso:
                progresso["pontos"] = 0
            if "nivel" not in progresso:
                progresso["nivel"] = "Novato"

            return progresso
    return {
    "vidas": 3,
    "pontos": 0,
    "nivel": "Novato",
    "pulos": 1,
    "dicas": 0
    }
```

Figura 2 – (Módulo de dados)

Na **Figura 2**, podemos ver de início duas funções muito importantes. São elas: *criar_novo_jogo()*, que como o nome diz, inicia um progresso do zero. A função monta um dicionário com os valores iniciais e já salva essas informações num arquivo JSON com o nome que o usuário escolher.

Na função *carregar_progresso()*, vemos o oposto: ela tenta abrir um arquivo de save (e.g. “progresso_teste.json”) e ler os dados salvos. Se por acaso algum dado estiver faltando, ele garante que os valores padrão sejam usados para evitar erro.

```python
def salvar_progresso(progresso, nome_arquivo="progresso.json", mostrar_mensagem=True):
    with open(nome_arquivo, "w", encoding="utf-8") as arquivo:
        json.dump(progresso, arquivo)
    if mostrar_mensagem:
        print("Progresso salvo!")

def atualizar_nivel(progresso):
    if progresso["pontos"] ⩾ 350:
        progresso["nivel"] = "Mestre de Física"
    elif progresso["pontos"] ⩾ 150:
        progresso["nivel"] = "Intermediário"
    elif progresso["pontos"] ⩾ 50:
        progresso["nivel"] = "Iniciante"
    else:
        progresso["nivel"] = "Novato"
    return progresso
```

Figura 3 – (Módulo de dados, pt.2)

A função *salvar_progresso()* escreve as informações no arquivo de save, como representado acima na **Figura 3**, transformando o dicionário de progresso em um texto do tipo JSON. Ela também exibe uma mensagem de “Progresso salvo!” quando necessário exibir essa informação para o usuário.

E por fim, a função *atualizar_nivel()* ajusta o nível do jogador com base na pontuação. Ela define se o jogador é “Novato”, “Iniciante”, “Intermediário” ou “Mestre de Física” conforme a pontuação acumulada.

No que diz respeito à complexidade, a função *criar_novo_jogo()* tem complexidade O(1), pois apenas cria e salva um dicionário fixo. A função *salvar_progresso()* também é O(1), já que apenas grava o conteúdo no arquivo. A função *carregar_progresso()* tem complexidade O(n), onde n representa o número de chaves a verificar ou completar após carregar o JSON. Já a função *atualizar_nivel()* também é O(1), pois faz comparações diretas baseadas na pontuação.

3. Organização dos Exercícios – *exercicios.json*

O arquivo armazena todos os desafios de Física que serão apresentados ao usuário durante o desafio. Ele funciona como um pequeno banco de dados em formato JSON, que é lido e interpretado pelo programa a cada rodada.

Cada exercício dentro desse arquivo é representado por um bloco com algumas informações, estruturado como um dicionário. As chaves principais dos exercícios são:

- **id** – identificador único de cada exercício
- **pergunta** – enunciado do desafio que será mostrado ao jogador.
- **resposta_final** – resposta considerada correta para o verificador.
- **dica** – uma orientação que pode ser comprada na loja e usada durante as questões.

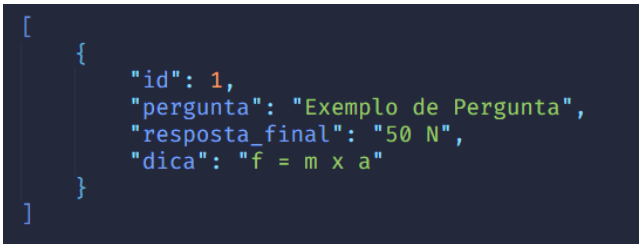

Figura 4 – Exemplo da organização dos exercícios em .json

Organizar os exercícios em um arquivo separado, torna o sistema flexível para implementação de categorias de dificuldade, que ainda pretendo fazer.

4. Módulo de Exercícios – *exercicios.py*

O módulo de apoio exercicios.py é o responsável por carregar os dados armazenados no arquivo JSON e torná-los utilizáveis dentro do jogo. Ele garante que os exercícios, respostas e dicas estejam disponíveis sempre que o usuário iniciar uma nova rodada.

```python
import json
import random

exercicios_arquivo = "exercicios.json"

def carregar_exercicios():
    try:
        with open(exercicios_arquivo, "r", encoding="utf-8") as arquivo:
            return json.load(arquivo)
    except Exception as e:
        print(f"Erro ao carregar exercícios: {e}")
        return []

exercicios = carregar_exercicios()

def escolher_exercicio():
    if not exercicios:
        return None
    return random.choice(exercicios)

def obter_resposta_correta(exercicio_id):
    for exercicio in exercicios:
        if exercicio["id"] == exercicio_id:
            return exercicio["resposta_final"]
    return None
```

Figura 5 – Módulo exercicios.py

O módulo contém apenas três funções, todas de suma importância no projeto: *carregar_exercicios()* lê o conteúdo do arquivo JSON e transforma em uma lista de dicionários, onde cada dicionário representa uma questão com seus respectivos campos (id, pergunta, resposta_final e dica). Já a função *escolher_exercicio()* escolhe aleatoriamente um dos exercícios carregados, permitindo que o jogo apresente uma questão

diferente a cada rodada. Essa aleatoriedade é feita com auxílio da biblioteca random.

O módulo também contém a função *obter_resposta_correta()*, que busca a resposta final de um exercício com base no seu ID único. É usada principalmente pelo módulo verificador para comparar a resposta do usuário com a correta.

No que se refere ao desempenho das funções deste módulo, a leitura e organização dos dados apresenta variações simples de complexidade. A função *carregar_exercícios()* possui complexidade O(n), onde *n* é um número arbitrário de exercícios presentes no arquivo JSON, já que ela precisa ler e converter todos os dados para uma lista de dicionários. A função *escolher_exercicio()* tem complexidade O(1), pois seleciona aleatoriamente um item da lista com *random.choice()*. As funções *obter_resposta_correta()* ou *obter_resolucao()* percorrem a lista procurando um exercício com ID correspondente, o que gera complexidade O(n) no pior caso.

4. Loja e Comandos Especiais – *loja.py + comandos*

Eu diria que, de todo o projeto, a loja é um dos elementos que deixam a experiência mais dinâmica. Nela, o jogador pode gastar seus pontos acumulados para comprar pulos ou dicas, que podem ser usados durante a resolução dos desafios. O funcionamento da loja está separado

no arquivo *loja.py*, como ilustra a **Figura 6**, e ela é acessada através do comando /shop no menu principal.

Cada item tem um custo: o pulo custa mais caro, pois permite evitar uma questão sem ser penalizado. Já a dica é mais acessível, e fornece uma ajuda com base no conteúdo do exercício.

```python
def loja(progresso):
    while True:
        print("\nLoja de Melhorias:")
        print(f"Pontos disponíveis: {progresso['pontos']}")
        print("1. Comprar Pulo (40 pontos)")
        print("2. Comprar Dica (20 pontos)")
        print("3. Sair da Loja")

        escolha = input("\nEscolha uma opção: ").strip()

        if escolha == "1" and progresso["pontos"] >= 40:
            progresso["pontos"] -= 40
            progresso["pulos"] += 1
            print("Você comprou um Pulo!")
        elif escolha == "2" and progresso["pontos"] >= 20:
            progresso["pontos"] -= 20
            progresso["dicas"] += 1
            print("Você comprou uma dica! Ela poderá ser usada durante uma pergunta di
        elif escolha == "3":
            break
        else:
            print("Opção inválida ou pontos insuficientes!")
```

Figura 6 – loja.py

Além da loja, o jogador também pode usar comandos durante a resolução de uma questão, como:

- **/pular** – para passar a pergunta (caso tenha pulos disponíveis)
- **/dica** – para receber uma sugestão (caso tenha comprado anteriormente)

Também é possível digitar */sair* no menu para salvar o progresso e encerrar o jogo com segurança.

Esses comandos tornam o desafio mais estratégico, dando ao usuário a liberdade de escolher como gerenciar seus recursos e continuar avançando.

A lógica por trás da loja e dos comandos especiais envolve verificações simples de condições e modificações no progresso do jogador. A função *loja()* apresenta complexidade O(1) para cada verificação de condição, pois as escolhas do jogador não dependem do tamanho de nenhuma estrutura de dados. Da mesma forma serve pros comandos */dica* e */pular*.

5. Módulo Principal – *main.py*

Nesse módulo, temos o ponto central do jogo. Ele é o responsável por organizar o fluxo principal e conectar diferentes módulos do sistema, como o verificador, o controle de progresso, loja e leitura dos exercícios.

```
progresso_existe = False
escolha = int(input("1 - Novo jogo\n2 - Carregar jogo\nEscolha: ").strip())
if escolha == 1:
    nome_save = input("Nome do save: ").strip()
    progresso = criar_novo_jogo("progresso_"+nome_save+".json")
else:
    while progresso_existe is False:
        nome_save = input("Nome do save: ").strip()
        caminho = "progresso_"+nome_save+".json"

        if os.path.exists(caminho):
            progresso_existe = True
            progresso = carregar_progresso("progresso_"+nome_save+".json")
        else:
            print("O progresso não existe.")
menu()
```

Figura 7 – (Escolha de progresso, main.py)

Logo no início da execução, o jogador é apresentado a um menu (**Figura 7**) com duas opções: iniciar um novo jogo ou carregar um progresso salvo. Essa escolha determina qual arquivo de progresso será utilizado e é feita através da entrada de um nome de save.

Depois disso, o menu principal do jogo oferece comandos como:

- **/start** – Para começar os desafios
- **/profile** – Para visualizar o progresso e itens
- **/shop** – Para acessar a loja
- **/sair** – Para encerrar o jogo

Os comandos são processados dentro de um loop (**Figura 8**), permitindo que o usuário interaja livremente com o sistema.

```
def menu():
    while True:
        opcao = input("\nJogo de Desafios de Física\n/start - Jogar\n/profile - Ver pr
        if opcao == "/start":
            iniciar_jogo()
        elif opcao == "/profile":
            print(f"\nNível: {progresso['nivel']}, Pontos: {progresso['pontos']}, Vida
        elif opcao == "/shop":
            abrir_loja(progresso)
        elif opcao == "/sair":
            print("Progresso salvo. Encerrando...")
            salvar_progresso(progresso, mostrar_mensagem=False)
            break
        else:
            print("Comando inválido!")

if __name__ == "__main__":
    try:
        menu()
    except Exception as e:
        print(f"\nOcorreu um erro inesperado: {e.__class__.__name__}: {e}")
```

Figura 8 – (Menu principal, main.py)

Ao iniciar os desafios, a função *iniciar_jogo()* sorteia um exercício e dá ao usuário a possibilidade de responder, usar uma dica, pular a pergunta ou encerrar o jogo. Essa função cuida do controle de tempo,

verificação da resposta e atualização do progresso, tornando o ciclo do jogo bem funcional e contínuo.

Como módulo central do sistema, *main.py* atua principalmente como controlador de fluxo entre as funções e módulos auxiliares. A função *menu()* apresenta complexidade O(1). A função *iniciar_jogo()* contém estruturas de repetição e chamadas a outras funções, o que confere a ela uma complexidade O(n) no pior caso, considerando a quantidade de interações do jogador ou verificações encadeadas durante uma sessão.

```
print(f"\n{exercicio['pergunta']}")
print("Você tem até 10 minutos para resolver essa questão fora do programa
print("Durante a questão, você pode digitar:")
print("  /pular → para pular (se tiver pulos)")
print("  /dica → para ver uma dica (se tiver dicas)\n")

inicio = time.time()

while True:
    resposta_usuario = input("Digite a resposta final: ").strip().lower()

    if resposta_usuario == "/pular":
        if progresso["pulos"] > 0:
            progresso["pulos"] -= 1
            print("Questão pulada! Nenhuma penalidade aplicada.")
            salvar_progresso(progresso, mostrar_mensagem=False)
            break
        else:
            print("Você não tem pulos disponíveis.")
    elif resposta_usuario == "/dica":
        if progresso['dicas'] > 0:
            print(f"Dica: {exercicio['dica']}")
            progresso['dicas'] -= 1
            salvar_progresso(progresso, mostrar_mensagem=False)
        else:
            print("Você não tem dicas disponíveis.")
    else:
        fim = time.time()
        break

if resposta_usuario not in ["/pular", "/dica"]:
    tempo_gasto = fim - inicio

    if tempo_gasto > tempo_limite:
        print(f"\nTempo esgotado! Você levou {int(tempo_gasto)} segundos."
        print("Você perdeu 5 pontos por demorar demais.")
        progresso["pontos"] = max(progresso["pontos"] - 5, 0)

    if verificar_resposta(exercicio['id'], resposta_usuario):
        print("Resposta correta! +10 pontos")
        progresso["pontos"] += 10
    else:
        print("Resposta incorreta. -1 vida")
        progresso["vidas"] -= 1

if progresso["vidas"] <= 0:
    print("GAME OVER! Você perdeu todas as suas vidas!")
    break

progresso = atualizar_nivel(progresso)
salvar_progresso(progresso)
```

Figura 9 – (Função iniciar_jogo(), destacando os blocos de maior relevância, como a tomada de decisão do jogador e o tempo sendo contabilizado)

Conclusão

Resumo do projeto

O projeto teve como objetivo desenvolver um sistema de estudo mais interativo incorporando elementos de gamificação. A ideia surgiu antes mesmo de se tornar um trabalho acadêmico, com a intenção ded transformar a prática de resolução de exercícios em uma experiência mais leve, dinâmica e motivadora. Para isso, a linguagem Python foi utilizada como base para a construção do sistema.

Aprendizado com a construção

Desde que comecei a dar meus primeiros passos na programação, considero que este foi um dos trabalhos mais bem organizados e completos que já desenvolvi — principalmente por ter sido algo planejado com antecedência, que me incentivou a enxergar como um hobby, e não como uma obrigação. Além disso, foi essencial para que eu entendesse melhor o processo de construção de um sistema como um todo.

Durante o desenvolvimento, aprendi como estruturar melhor o código, separando as funções em módulos e facilitando futuras manutenções. Também tive meu primeiro contato com arquivos JSON e

passei a enxergar com mais clareza como as funções se comunicam e se complementam dentro de um programa.

Destaques Técnicos

Entre os principais destaques técnicos do sistema, estão a implementação da loja e o controle de progresso do jogador. A loja foi desenvolvida para tornar o sistema mais interativo, permitindo que o jogador use seus pontos para comprar pulos e dicas. Isso trouxe um elemento de estratégia que vai além de resolução de questões.

Outro ponto forte foi o sistema de salvamento de progresso, que permite que o usuário continue exatamente de onde parou, inclusive com suporte a múltiplos saves nomeados. Essa funcionalidade foi implementada com uso de arquivos JSON e ajudou a reforçar o conceito de persistência de dados.

Desafios Enfrentados

Durante o desenvolvimento, um dos principais desafios foi tentar criar um verificador de respostas mais robusto. A ideia inicial era permitir diferentes formas de resolução, analisando não apenas a resposta final, mas também os passos lógicos seguidos pelo jogador. No entanto essa abordagem exigiria um nível de complexidade maior na verificação, o que acabou sendo simplificado por ideia do professor. A versão final do verificador foca na comparação direta entre a resposta do

usuário e a resposta correta, com apenas flexibilidades como ignorar espaços e letras maiúsculas. Ainda assim, esse processo me ajudou a entender melhor os limites e as possibilidades de uma verificação automatizada.

Possíveis melhorias e próximos passos

Apesar do sistema já estar funcional, existem várias melhorias que já tenho em mente e que podem ser implementadas futuramente, de acordo com as necessidades que vierem. A primeira delas é a categorização dos exercícios por áreas da Física (I, II e III), o que tornaria o progresso mais estruturado e interessante para o usuário.

Outra melhoria seria o aperfeiçoamento do sistema de verificação, que acabou ficando mais simples do que o planejado inicialmente. A ideia seria permitir uma análise mais detalhada do raciocínio do jogador, considerando também os passos intermediários da resolução. Por fim, a inclusão de uma interface gráfica deixaria o sistema mais acessível e amigável para quem não está acostumado a interagir via linha de comando.