Jogo da forca em Python

resolução de problemas com logica matematica

Lucas Dal Pra Brascher[[1]](#footnote-1)\*

RESUMO

Esse projeto foi desenvolvido para a matéria de Resolução de Problemas com Lógica Matemática, no segundo período do curso de Engenharia de Software. O projeto consiste em um jogo da forca em Python, onde foi utilizado a biblioteca CustomTkinter para fazer as telas e a logica do jogo. O jogo contem 4 telas, a primeira sendo a tela inicial, a tela do jogo, a tela de como jogar, e a de configurações.

Palavras-chave: python; matematica; jogo; engenharia; software.

ABSTRACT

This project was developed for the "Problem Solving with Mathematical Logic" course in the second semester of the Software Engineering program. The project consists of a Hangman game in Python, where the CustomTkinter library was used to create the screens and the game logic. The game includes 4 screens: the main screen, the game screen, the how-to-play screen, and the settings screen.

Keywords: python; mathematics; game; engineering; software.

# INTRODUÇÃO

Este artigo descreve o desenvolvimento de um jogo da forca em Python, projetado como uma ferramenta educativa para o ensino de lógica matemática, com ênfase nos conceitos de conjuntos, como interseção e união. O jogo foi desenvolvido utilizando a biblioteca CustomTkinter para criação da interface gráfica e tem como objetivo proporcionar uma forma interativa e divertida de aprender matemática.

No jogo, o jogador deve adivinhar respostas relacionadas a questões sobre conjuntos, com a evolução de um boneco na forca a cada erro cometido. Além de ensinar conceitos matemáticos, o jogo visa aprimorar habilidades de resolução de problemas de forma lúdica, com uma interface amigável e de fácil navegação.

Este projeto foi desenvolvido como parte do curso de "Resolução de Problemas com Lógica Matemática", no segundo período do curso de Engenharia de Software. O objetivo deste artigo é apresentar a estrutura do jogo, discutir os aspectos técnicos da implementação, os desafios encontrados durante o desenvolvimento e a eficácia do jogo como uma ferramenta educacional.

# TÍTULO DO CAPÍTULO

## Objetivo do jogo

O objetivo principal do jogo da forca desenvolvido neste projeto é proporcionar uma experiência interativa e educativa, utilizando o contexto de lógica matemática para ensinar conceitos relacionados a conjuntos, como interseção e união. O jogo desafia o jogador a responder perguntas sobre operações de conjuntos de forma divertida, enquanto resolve as questões por meio de tentativas e erros, onde cada erro adiciona uma parte ao boneco da forca.

Além de incentivar o raciocínio lógico, o jogo busca familiarizar os jogadores com a aplicação prática dos conceitos de matemática discreta, promovendo o aprendizado através de um ambiente dinâmico e visual. O jogador deve adivinhar as respostas corretas para evitar que o boneco da forca seja completamente desenhado, ao mesmo tempo em que aprende mais sobre os temas de conjunto, como os exemplos de interseção e união de conjuntos fornecidos no jogo.

O jogo termina quando o jogador acerta todas as respostas de uma sequência de perguntas ou quando o boneco da forca é completado, sinalizando que o jogador errou um número excessivo de vezes.

Figura 1 ‒ Primeira tela imaginada para o jogo, desenvolvida no FIGMA

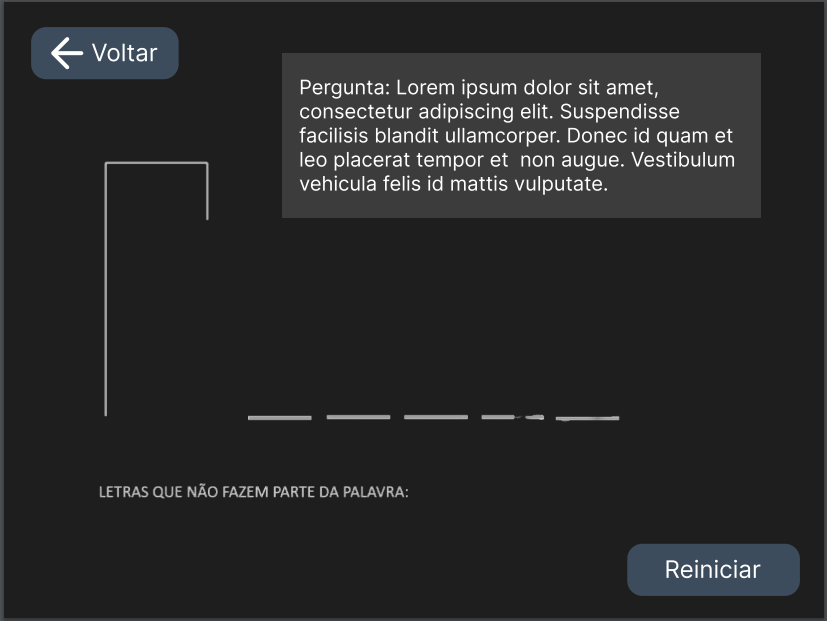
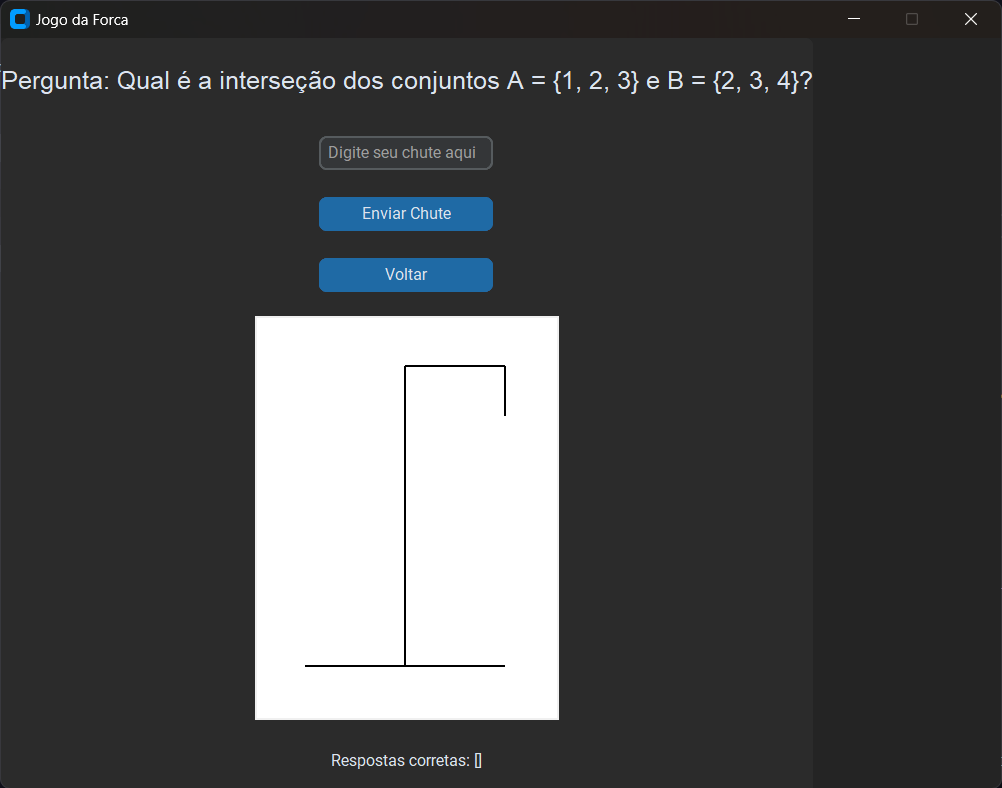
  
Fonte: Lucas Dal Pra Brascher (2024).

Figura 2 ‒ A tela desenvolvida em Python

  
Fonte: Lucas Dal Pra Brascher (2024).

## Tecnologias Utilizadas

* Python: A linguagem de programação principal utilizada para o desenvolvimento da lógica do jogo e a interação com o usuário. O Python foi escolhido pela sua simplicidade e versatilidade, além de ser amplamente utilizado em projetos de diferentes áreas, incluindo jogos educativos e interfaces gráficas.
* CustomTkinter: Biblioteca de interface gráfica utilizada para a criação das telas do jogo. CustomTkinter é uma extensão da biblioteca Tkinter, permitindo um design mais moderno e personalizável para interfaces gráficas em Python. Com ela, foi possível criar as telas de navegação (início, como jogar, jogo, configurações), assim como os componentes interativos, como botões, entradas de texto e rótulos.
* Tkinter: Biblioteca padrão para construção de interfaces gráficas em Python. Embora CustomTkinter tenha sido utilizada para a estilização, Tkinter foi a base para a construção dos frames e o gerenciamento da interface de usuário.
* Canvas (Tkinter): O Canvas foi utilizado para desenhar a estrutura da forca e os diferentes estágios do boneco da forca à medida que o jogador comete erros. Ele permite desenhar formas geométricas e é ideal para criar gráficos simples e dinâmicos, como as partes do corpo do boneco.
* Matemática Discreta (Conceitos de Conjuntos): O jogo integra conceitos de matemática discreta, especificamente sobre operações de conjuntos, como interseção e união. Estes conceitos foram incorporados nas perguntas do jogo, permitindo que o jogador aprenda de maneira prática enquanto se diverte.

## Estrutura do jogo

O jogo da forca foi estruturado de maneira modular, dividindo-se em diversas telas e componentes responsáveis pela gestão da lógica do jogo, interface com o usuário e controle do fluxo de execução. Abaixo, descreve-se a estrutura geral do jogo:

1. Tela Inicial (Inicio):
   * A tela inicial é a primeira interface exibida ao jogador. Nela, o usuário pode iniciar o jogo, acessar as instruções de como jogar, modificar as configurações ou sair do jogo.
   * Contém os seguintes elementos:
     + Botão Iniciar: Leva o jogador à tela de jogo.
     + Botão Como Jogar?: Exibe as instruções de como jogar.
     + Botão Configurações: Permite que o jogador ajuste preferências do jogo, como a ativação do modo de tela cheia.
     + Botão Sair: Encerra o jogo.
2. Tela de Como Jogar (ComoJogar):
   * Esta tela fornece ao jogador uma explicação sobre as regras do jogo. O jogador aprende como ele deve interagir com o sistema, qual é o objetivo e como os erros afetam o progresso no jogo.
   * Contém os seguintes elementos:
     + Texto explicativo: Descreve as regras e como o jogo funciona.
     + Botão Voltar: Retorna à tela inicial.
3. Tela de Jogo (Jogo):
   * A tela principal do jogo, onde o jogador interage com as perguntas e tenta adivinhar as respostas. O objetivo é acertar o maior número de respostas sem cometer erros suficientes para completar o boneco da forca.
   * Contém os seguintes elementos:
     + Rótulo de Pergunta: Exibe a pergunta atual que está sendo feita ao jogador.
     + Entrada de Chute: Permite que o jogador digite uma tentativa de resposta.
     + Botão Enviar Chute: Envia a resposta do jogador e avalia se ela está correta.
     + Rótulo de Respostas Corretas e Erradas: Exibe as respostas já acertadas e os erros cometidos.
     + Canvas da Forca: Representa a estrutura da forca, onde partes do corpo são desenhadas à medida que o jogador comete erros.
     + Botão Voltar: Permite que o jogador retorne à tela inicial.
4. Tela de Configurações (Configuracoes):
   * Tela que permite ao jogador alterar preferências do jogo, como o modo de tela cheia. Essa tela também oferece a opção de voltar à tela inicial.
   * Contém os seguintes elementos:
     + Botão Modo de Tela Cheia: Ativa ou desativa o modo de tela cheia.
     + Botão Voltar: Retorna à tela inicial.
5. Tela de Próxima Pergunta (ProximaPergunta):
   * Após o término de uma rodada, esta tela exibe uma mensagem de feedback sobre a conclusão da pergunta e oferece a opção de avançar para a próxima questão ou finalizar o jogo.
   * Contém os seguintes elementos:
     + Mensagem de Feedback: Informa ao jogador se a pergunta foi respondida corretamente ou não.
     + Botão Próxima Pergunta: Avança para a próxima pergunta ou termina o jogo se não houver mais questões.
6. Lógica de Jogo:
   * Perguntas: O jogo apresenta uma lista de perguntas relacionadas a conceitos de conjuntos matemáticos. Cada pergunta possui uma resposta correta que pode ser uma ou mais opções.
   * Contagem de Erros: O jogador possui um número limitado de tentativas antes de completar o boneco da forca. A cada erro, uma parte do corpo do boneco é desenhada.
   * Condicional de Vitória/Derrota: O jogo avança para a próxima pergunta se o jogador acertar todas as respostas ou termina se o jogador cometer erros suficientes para completar o boneco da forca.

A estrutura modular do jogo permite fácil manutenção e expansão, como a adição de novas perguntas ou a modificação de configurações do jogo. Cada tela e funcionalidade foi projetada para proporcionar uma experiência interativa e educacional, promovendo o aprendizado de matemática através de um jogo envolvente.

## Logica do jogo

A lógica do jogo da forca segue um fluxo interativo onde o jogador tenta adivinhar as respostas corretas para questões relacionadas a conjuntos matemáticos. O objetivo é acertar as respostas sem cometer erros suficientes para completar o desenho do boneco da forca. A seguir, detalha-se a lógica do jogo, com exemplos baseados no código fornecido:

1. **Início do Jogo**:
   * O jogo começa com a tela inicial (classe Inicio), onde o jogador pode optar por iniciar o jogo, acessar as instruções ou configurar o modo de tela cheia. Quando o jogador clica no botão "Iniciar", o método show\_frame("Jogo") é chamado, exibindo a tela do jogo.
   * O código:

ctk.CTkButton(self, text="Iniciar", command=lambda: parent.show\_frame("Jogo")).pack(pady=10)

1. **Estrutura das Perguntas**:
   * As perguntas do jogo estão armazenadas na lista self.questions e cada pergunta possui uma chave question para o enunciado e uma chave answer para as respostas corretas. Por exemplo:

self.questions = [

{"question": "Qual é a interseção dos conjuntos A = {1, 2, 3} e B = {2, 3, 4}?", "answer": ["2", "3"]},

{"question": "Qual é a união dos conjuntos A = {5, 6} e B = {6, 7}?", "answer": ["5", "6", "7"]}

]

1. **Interação do Jogador**:
   * O jogador é convidado a inserir uma resposta por meio de um campo de texto (ctk.CTkEntry). O código a seguir trata a resposta do jogador:

self.guess\_entry = ctk.CTkEntry(self, placeholder\_text="Digite seu chute aqui")

self.submit\_button = ctk.CTkButton(self, text="Enviar Chute", command=self.submit\_guess)

* + O método submit\_guess processa a entrada do jogador. Se a resposta for correta, o jogo atualiza a lista de respostas corretas; caso contrário, adiciona um erro ao boneco da forca:

if guess in question["answer"]:

# Se a resposta for correta

if guess not in self.parent.correct\_answers:

self.parent.correct\_answers.append(guess)

else:

# Se a resposta for errada

if guess not in self.parent.incorrect\_guesses:

self.parent.incorrect\_guesses.append(guess)

self.parent.hangman\_parts += 1

self.draw\_hangman()

1. **Feedback ao Jogador**:
   * Para dar feedback sobre as respostas do jogador, o jogo exibe as respostas corretas e erradas. Isso é feito por meio dos rótulos correct\_label e incorrect\_label:

self.correct\_label = ctk.CTkLabel(self, text="Respostas corretas: []", wraplength=700, justify="left")

self.incorrect\_label = ctk.CTkLabel(self, text="Respostas erradas: []", wraplength=700, justify="left")

* + A cada tentativa do jogador, o jogo atualiza essas listas, refletindo as respostas corretas ou erradas.

1. **Desenhando a Forca**:
   * Quando o jogador erra, o jogo desenha partes do boneco da forca usando o método draw\_hangman. Cada erro adiciona uma parte ao boneco:

parts = [

lambda: self.canvas.create\_oval(230, 100, 270, 140, width=2), # Cabeça

lambda: self.canvas.create\_line(250, 140, 250, 240, width=2), # Corpo

lambda: self.canvas.create\_line(250, 160, 230, 200, width=2), # Braço esquerdo

lambda: self.canvas.create\_line(250, 160, 270, 200, width=2), # Braço direito

lambda: self.canvas.create\_line(250, 240, 230, 280, width=2), # Perna esquerda

lambda: self.canvas.create\_line(250, 240, 270, 280, width=2), # Perna direita

]

if self.parent.hangman\_parts <= len(parts):

parts[self.parent.hangman\_parts - 1]()

* + O método draw\_hangman é chamado a cada erro, desenhando uma parte adicional do corpo do boneco.

1. **Condicional de Vitória**:
   * O jogador vence a pergunta quando todas as respostas corretas foram adivinhadas antes de cometer erros suficientes. A condição de vitória é verificada pela comparação entre as respostas corretas e o número total de respostas para a pergunta:

if len(self.parent.correct\_answers) == len(question["answer"]):

self.parent.show\_frame("ProximaPergunta")

* + Caso o jogador acerte todas as respostas, o jogo avança para a tela de "ProximaPergunta".

1. **Condicional de Derrota**:
   * O jogador perde quando o número de erros atinge o limite (6 erros, no caso), completando o boneco da forca. O código que verifica essa condição é:

elif self.parent.hangman\_parts >= 6:

self.parent.show\_frame("ProximaPergunta")

1. **Tela de Próxima Pergunta**:
   * Após o término de uma pergunta, o jogo exibe a tela "ProximaPergunta", permitindo ao jogador avançar para a próxima questão ou finalizar o jogo:

ctk.CTkButton(self, text="Próxima Pergunta", command=parent.next\_question\_or\_end).pack(pady=10)

1. **Controle de Fluxo**:
   * A navegação entre as telas é feita através do método show\_frame, que troca entre diferentes telas, como "Inicio", "Jogo" e "ProximaPergunta". O controle de fluxo permite que o jogo seja dinâmico e o jogador interaja de forma contínua:

def show\_frame(self, name):

frame = self.frames[name]

frame.tkraise()

1. **Reinício e Finalização do Jogo**:

* Ao término do jogo, o jogador pode reiniciar a partida, o que é feito por meio do método reset\_game:

def reset\_game(self):

self.correct\_answers = []

self.incorrect\_guesses = []

self.hangman\_parts = 0

self.current\_question\_index = 0

self.frames["Jogo"].update\_question()

self.frames["Jogo"].reset\_hangman()

Com essa lógica, o jogo cria uma experiência de aprendizado interativa, onde o jogador é desafiado a aplicar seus conhecimentos de conjuntos matemáticos de forma divertida, enquanto gerencia os erros e acertos que afetam o progresso no jogo.

## ALGORITMO DE FUNCIONAMENTO

O algoritmo de funcionamento do Jogo da Forca segue uma sequência de etapas que guiam o jogador desde o início do jogo até a vitória ou derrota. Abaixo, descreve-se o fluxo geral do jogo, utilizando exemplos práticos do código para ilustrar o funcionamento:

1. **Início do Jogo**:
   * Quando o jogo é iniciado, a interface apresenta a tela inicial onde o jogador pode optar por começar a jogar, ver as instruções ou configurar as preferências.
   * Ao clicar em "Iniciar", o sistema muda para a tela do jogo e começa a exibir as perguntas.
2. **Seleção e Exibição das Perguntas**:
   * O jogo seleciona uma pergunta aleatória da lista de questões pré-definidas. Cada pergunta possui um enunciado e um conjunto de respostas corretas associadas.
   * O jogador visualiza a pergunta e é convidado a inserir suas respostas. Por exemplo, a pergunta pode ser sobre a interseção de conjuntos:
     + Pergunta: "Qual é a interseção dos conjuntos A = {1, 2, 3} e B = {2, 3, 4}?"
     + Respostas corretas: [2, 3]
3. **Recebendo o Chute do Jogador**:
   * O jogador insere seu palpite por meio de um campo de texto na interface. Quando ele envia o chute, o sistema valida se a resposta está correta ou incorreta.
   * Se a resposta for correta, o sistema a registra como uma resposta válida e a marca como "acertada". Se a resposta for errada, o sistema adiciona um erro ao desenho da forca.
4. **Exibição do Feedback**:
   * O feedback sobre a resposta do jogador é mostrado na tela. As respostas corretas e incorretas são exibidas separadamente, permitindo ao jogador ver seu progresso. Além disso, o número de partes do corpo do boneco da forca é atualizado.
     + Se a resposta for certa, a lista de respostas corretas é atualizada.
     + Se a resposta for errada, uma parte do boneco é desenhada. Se o jogador cometer 6 erros, o jogo é encerrado.
5. **Avanço para a Próxima Pergunta ou Fim do Jogo**:
   * Quando o jogador acerta todas as respostas de uma pergunta antes de cometer 6 erros, o jogo avança para a próxima questão.
   * Caso o jogador erre o número máximo de vezes, o jogo é encerrado e o resultado final é mostrado.
   * A troca de telas é feita dinamicamente, permitindo uma transição fluida entre as perguntas e a tela de fim de jogo.
6. **Reinício do Jogo**:
   * O jogador tem a opção de reiniciar o jogo a qualquer momento. Isso faz com que o jogo recomece do início, com novas perguntas e um novo controle de erros.
7. **Fim do Jogo**:
   * Quando o jogador termina todas as perguntas ou chega ao número máximo de erros, o jogo exibe uma tela final, oferecendo a opção de reiniciar ou encerrar a partida.

## Desafios e soluções

Durante o desenvolvimento do Jogo da Forca, diversos desafios foram encontrados, desde questões relacionadas à lógica de jogo até a implementação da interface gráfica. Abaixo, são apresentados alguns dos principais desafios enfrentados e as soluções adotadas para superá-los:

1. **Desafio: Criação de uma Interface Intuitiva e Funcional**
   * A criação de uma interface gráfica que fosse ao mesmo tempo funcional e intuitiva para o jogador foi um dos maiores desafios. O uso da biblioteca **CustomTkinter** foi essencial para que fosse possível criar telas interativas e responsivas para o jogo.
   * **Solução**: Utilizamos a estrutura de Frames para dividir o jogo em diferentes seções, como a tela inicial, a tela do jogo, a tela de como jogar e a tela de configurações. Isso permitiu ao jogador navegar entre as diferentes telas de forma simples e organizada. A utilização de botões e campos de entrada também garantiu uma interação fluida. O código de navegação entre as telas foi implementado por meio do método show\_frame(), que permite exibir a tela correspondente conforme a escolha do jogador.
2. **Desafio: Gerenciamento das Perguntas e Respostas**
   * Um dos aspectos cruciais do jogo é a verificação das respostas do jogador. Cada pergunta possui um conjunto de respostas corretas, e o jogo deve ser capaz de validar as respostas fornecidas.
   * **Solução**: Para gerenciar isso, criamos uma estrutura de dados no formato de lista de dicionários, onde cada dicionário contém a pergunta e suas respectivas respostas. Quando o jogador fornece um chute, o código verifica se a resposta está correta comparando-a com as respostas armazenadas. A verificação é feita no método submit\_guess(), onde a entrada do jogador é comparada com as respostas corretas e a atualização do estado do jogo é realizada conforme o resultado.
3. **Desafio: Implementação do Desenho da Forca**
   * Um dos desafios mais visíveis foi a criação da dinâmica de desenho da forca. Cada erro cometido pelo jogador deve resultar em uma parte do boneco sendo desenhada na tela, até que o jogador perca o jogo.
   * **Solução**: Utilizamos o widget **Canvas** do **CustomTkinter** para desenhar a forca e as partes do corpo do boneco. O método draw\_gallows() é responsável por desenhar a estrutura inicial da forca, enquanto o método draw\_hangman() desenha as partes do corpo conforme o número de erros do jogador aumenta. A cada erro, o número de partes do corpo desenhadas é controlado pela variável hangman\_parts, e o código verifica se o número de erros atingiu o limite máximo para determinar o fim do jogo.
4. **Desafio: Navegação Entre as Perguntas e Controle do Jogo**
   * Outro desafio foi o controle de avanço entre as perguntas e a gestão do estado do jogo, garantindo que o jogador pudesse navegar entre as questões sem dificuldades, e que o jogo fosse reiniciado corretamente quando necessário.
   * **Solução**: Utilizamos um índice de perguntas (current\_question\_index) para controlar a posição da pergunta atual. Ao final de uma pergunta, o jogo verifica se há mais perguntas disponíveis para mostrar. Caso contrário, o jogo exibe a tela de fim de jogo. O método next\_question\_or\_end() é responsável por avançar para a próxima pergunta ou terminar o jogo quando necessário. Para reiniciar o jogo, o método reset\_game() redefine as variáveis de estado do jogo, como as respostas corretas e incorretas, além de resetar o desenho da forca.
5. **Desafio: Validação das Respostas Parciais**
   * Permitir respostas parcialmente corretas é uma funcionalidade interessante, mas foi um desafio implementá-la de forma eficiente. Isso porque o jogo deve verificar se o chute do jogador corresponde a parte da resposta correta.
   * **Solução**: Implementamos a validação de respostas parciais verificando se o chute do jogador está presente nas respostas corretas (mesmo que o jogador não tenha acertado todas as respostas de uma vez). O método submit\_guess() foi ajustado para permitir que os acertos parciais fossem registrados, o que mantém o jogador envolvido e permite um progresso mais dinâmico durante o jogo.
6. **Desafio: Gerenciamento de Erros e Exibição de Feedback**
   * O gerenciamento de erros, como entradas inválidas (respostas vazias ou caracteres não esperados), foi outro desafio importante para garantir que o jogo fosse robusto e oferecesse uma boa experiência ao jogador.
   * **Solução**: Implementamos uma verificação simples para garantir que o jogador inserisse uma resposta antes de processá-la. No método submit\_guess(), o código verifica se o campo de entrada está vazio antes de tentar registrar o chute. Além disso, fornecemos feedback claro sobre as respostas corretas e incorretas, garantindo que o jogador sempre soubesse o estado atual do jogo.

## Código do Jogo

import customtkinter as ctk

class ForcaApp(*ctk*.CTk):

    def \_\_init\_\_(*self*):

        super().\_\_init\_\_()

*self*.title("Jogo da Forca")

*self*.geometry("800x600")

*self*.resizable(False, False)

*self*.text\_size = 20

*self*.fullscreen = False

*self*.questions = [

            {"question": "Qual é a interseção dos conjuntos A = {1, 2, 3} e B = {2, 3, 4}?", "answer": ["2", "3"]},

            {"question": "Qual é a união dos conjuntos A = {5, 6} e B = {6, 7}?", "answer": ["5", "6", "7"]},

            {"question": "Qual é a diferença entre os conjuntos A = {1, 2, 3} e B = {3, 4, 5}?", "answer": ["1", "2"]},

            {"question": "Qual é a interseção dos conjuntos A = {a, b, c} e B = {b, c, d}?", "answer": ["b", "c"]},

            {"question": "Qual é a união dos conjuntos A = {x, y} e B = {y, z}?", "answer": ["x", "y", "z"]},

            {"question": "Qual é a diferença simétrica entre os conjuntos A = {1, 2, 3} e B = {2, 3, 4}?", "answer": ["1", "4"]},

            {"question": "Qual é a união dos conjuntos A = {1, 2} e B = {2, 3}?", "answer": ["1", "2", "3"]},

            {"question": "Qual é a interseção dos conjuntos A = {p, q, r} e B = {q, r, s}?", "answer": ["q", "r"]},

            {"question": "Qual é a diferença entre os conjuntos A = {7, 8, 9} e B = {9, 10, 11}?", "answer": ["7", "8"]},

            {"question": "Qual é a união dos conjuntos A = {10, 11, 12} e B = {12, 13, 14}?", "answer": ["10", "11", "12", "13", "14"]},

            {"question": "Qual é a interseção dos conjuntos A = {a, b, c, d} e B = {c, d, e, f}?", "answer": ["c", "d"]},

            {"question": "Qual é a diferença simétrica entre os conjuntos A = {1, 2, 3, 4} e B = {3, 4, 5, 6}?", "answer": ["1", "2", "5", "6"]},

            {"question": "Qual é a união dos conjuntos A = {a, b, c} e B = {b, c, d, e}?", "answer": ["a", "b", "c", "d", "e"]},

            {"question": "Qual é a interseção dos conjuntos A = {m, n, o} e B = {o, p, q}?", "answer": ["o"]},

            {"question": "Qual é a diferença entre os conjuntos A = {1, 2, 3} e B = {1, 3, 5}?", "answer": ["2"]},

            {"question": "Qual é a diferença simétrica entre os conjuntos A = {1, 2, 3} e B = {2, 3, 4}?", "answer": ["1", "4"]}

        ]

*self*.current\_question\_index = 0

*self*.correct\_answers = []

*self*.incorrect\_guesses = []

*self*.hangman\_parts = 0

*self*.frames = {}

*self*.create\_frames()

*self*.show\_frame("Inicio")

    def create\_frames(*self*):

        for F in (Inicio, ComoJogar, Jogo, Configuracoes, ProximaPergunta):

            frame = F(*self*)

*self*.frames[F.\_\_name\_\_] = frame

            frame.grid(*row*=0, *column*=0, *sticky*="nsew")

    def show\_frame(*self*, *name*):

        frame = *self*.frames[*name*]

        frame.tkraise()

    def reset\_game(*self*):

*self*.correct\_answers = []

*self*.incorrect\_guesses = []

*self*.hangman\_parts = 0

*self*.current\_question\_index = 0

*self*.frames["Jogo"].update\_question()

*self*.frames["Jogo"].reset\_hangman()

    def next\_question\_or\_end(*self*):

        if *self*.current\_question\_index + 1 < len(*self*.questions):

*self*.current\_question\_index += 1

*self*.frames["Jogo"].update\_question()

*self*.frames["Jogo"].reset\_hangman()

*self*.show\_frame("Jogo")

        else:

*self*.show\_frame("Historico")

    def toggle\_fullscreen(*self*):

*self*.fullscreen = not *self*.fullscreen

*self*.attributes("-fullscreen", *self*.fullscreen)

class Inicio(*ctk*.CTkFrame):

    def \_\_init\_\_(*self*, *parent*):

        super().\_\_init\_\_(*parent*)

        ctk.CTkLabel(*self*, *text*="Jogo da Forca", *font*=("Arial", 30)).pack(*pady*=20)

        ctk.CTkButton(*self*, *text*="Iniciar", *command*=lambda: *parent*.show\_frame("Jogo")).pack(*pady*=10)

        ctk.CTkButton(*self*, *text*="Como Jogar?", *command*=lambda: *parent*.show\_frame("ComoJogar")).pack(*pady*=10)

        ctk.CTkButton(*self*, *text*="Configurações", *command*=lambda: *parent*.show\_frame("Configuracoes")).pack(*pady*=10)

        ctk.CTkButton(*self*, *text*="Sair", *command*=*parent*.quit).pack(*pady*=10)

class ComoJogar(*ctk*.CTkFrame):

    def \_\_init\_\_(*self*, *parent*):

        super().\_\_init\_\_(*parent*)

        ctk.CTkLabel(*self*, *text*="Como Jogar", *font*=("Arial", 30)).pack(*pady*=20)

        ctk.CTkLabel(*self*, *text*=(

            "O jogo consiste em responder perguntas sobre conjuntos.\n"

            "Você pode tentar chutar as respostas, e erros adicionam partes ao boneco na forca.\n"

            "Respostas parcialmente corretas são aceitas."

        ), *wraplength*=700, *justify*="left").pack(*pady*=20)

        ctk.CTkButton(*self*, *text*="Voltar", *command*=lambda: *parent*.show\_frame("Inicio")).pack(*pady*=10)

class Jogo(*ctk*.CTkFrame):

    def \_\_init\_\_(*self*, *parent*):

        super().\_\_init\_\_(*parent*)

*self*.parent = *parent*

*self*.question\_label = ctk.CTkLabel(*self*, *text*="", *font*=("Arial", 20), *wraplength*=700, *justify*="center")

*self*.question\_label.pack(*pady*=20)

*self*.guess\_entry = ctk.CTkEntry(*self*, *placeholder\_text*="Digite seu chute aqui")

*self*.guess\_entry.pack(*pady*=10)

*self*.submit\_button = ctk.CTkButton(*self*, *text*="Enviar Chute", *command*=*self*.submit\_guess)

*self*.submit\_button.pack(*pady*=10)

*self*.back\_button = ctk.CTkButton(*self*, *text*="Voltar", *command*=lambda: *parent*.show\_frame("Inicio"))

*self*.back\_button.pack(*pady*=10)

*self*.canvas = ctk.CTkCanvas(*self*, *width*=300, *height*=400, *bg*="white")

*self*.canvas.pack(*pady*=10)

*self*.draw\_gallows()

*self*.correct\_label = ctk.CTkLabel(*self*, *text*="Respostas corretas: []", *wraplength*=700, *justify*="left")

*self*.correct\_label.pack(*pady*=10)

*self*.incorrect\_label = ctk.CTkLabel(*self*, *text*="Respostas erradas: []", *wraplength*=700, *justify*="left")

*self*.incorrect\_label.pack(*pady*=10)

*self*.update\_question()

    def update\_question(*self*):

        question = *self*.parent.questions[*self*.parent.current\_question\_index]["question"]

*self*.question\_label.configure(*text*=f"Pergunta: {question}")

    def submit\_guess(*self*):

        guess = *self*.guess\_entry.get().strip()

*self*.guess\_entry.delete(0, ctk.END)

        if not guess:

            return

        question = *self*.parent.questions[*self*.parent.current\_question\_index]

        if guess in question["answer"]:

            if guess not in *self*.parent.correct\_answers:

*self*.parent.correct\_answers.append(guess)

*self*.correct\_label.configure(*text*=f"Respostas corretas: {*self*.parent.correct\_answers}")

        else:

            if guess not in *self*.parent.incorrect\_guesses:

*self*.parent.incorrect\_guesses.append(guess)

*self*.parent.hangman\_parts += 1

*self*.draw\_hangman()

*self*.incorrect\_label.configure(*text*=f"Respostas erradas: {*self*.parent.incorrect\_guesses}")

        if len(*self*.parent.correct\_answers) == len(question["answer"]):

*self*.parent.show\_frame("ProximaPergunta")

        elif *self*.parent.hangman\_parts >= 6:

*self*.parent.show\_frame("ProximaPergunta")

    def draw\_gallows(*self*):

*self*.canvas.create\_line(50, 350, 250, 350, *width*=2)

*self*.canvas.create\_line(150, 350, 150, 50, *width*=2)

*self*.canvas.create\_line(150, 50, 250, 50, *width*=2)

*self*.canvas.create\_line(250, 50, 250, 100, *width*=2)

    def draw\_hangman(*self*):

        parts = [

            lambda: *self*.canvas.create\_oval(230, 100, 270, 140, *width*=2),  # Cabeça

            lambda: *self*.canvas.create\_line(250, 140, 250, 240, *width*=2),  # Corpo

            lambda: *self*.canvas.create\_line(250, 160, 230, 200, *width*=2),  # Braço esquerdo

            lambda: *self*.canvas.create\_line(250, 160, 270, 200, *width*=2),  # Braço direito

            lambda: *self*.canvas.create\_line(250, 240, 230, 280, *width*=2),  # Perna esquerda

            lambda: *self*.canvas.create\_line(250, 240, 270, 280, *width*=2),  # Perna direita

        ]

        if *self*.parent.hangman\_parts <= len(parts):

            parts[*self*.parent.hangman\_parts - 1]()

    def reset\_hangman(*self*):

*self*.canvas.delete("all")

*self*.draw\_gallows()

class ProximaPergunta(*ctk*.CTkFrame):

    def \_\_init\_\_(*self*, *parent*):

        super().\_\_init\_\_(*parent*)

        ctk.CTkLabel(*self*, *text*="Pergunta Finalizada!", *font*=("Arial", 30)).pack(*pady*=20)

        ctk.CTkButton(*self*, *text*="Próxima Pergunta", *command*=*parent*.next\_question\_or\_end).pack(*pady*=10)

class Configuracoes(*ctk*.CTkFrame):

    def \_\_init\_\_(*self*, *parent*):

        super().\_\_init\_\_(*parent*)

        ctk.CTkLabel(*self*, *text*="Configurações", *font*=("Arial", 30)).pack(*pady*=20)

        ctk.CTkButton(*self*, *text*="Modo de Tela Cheia", *command*=*parent*.toggle\_fullscreen).pack(*pady*=10)

        ctk.CTkButton(*self*, *text*="Voltar", *command*=lambda: *parent*.show\_frame("Inicio")).pack(*pady*=10)

if \_\_name\_\_ == "\_\_main\_\_":

    app = ForcaApp()

    app.mainloop()

# consideraçÕES FINAIS

O desenvolvimento do "Jogo da Forca" foi uma oportunidade para aplicar e aprofundar conhecimentos em lógica matemática, especialmente no que tange ao entendimento e manipulação de conjuntos. O jogo não só proporciona uma maneira divertida de revisar conceitos como união, interseção e diferença entre conjuntos, mas também serve como uma ferramenta interativa para o aprendizado de forma lúdica e envolvente.

Ao longo do desenvolvimento, foram enfrentados desafios técnicos e conceituais, como a implementação de uma interface gráfica intuitiva e a criação de uma mecânica de jogo que equilibrasse a dificuldade das questões e a interatividade. A solução desses desafios foi possível graças à escolha da biblioteca **CustomTkinter**, que permitiu a construção de uma interface simples, mas eficaz, além do uso de lógica de controle que integra perfeitamente a jogabilidade com as questões matemáticas.

A implementação de perguntas relacionadas a conjuntos oferece uma excelente maneira de reforçar o aprendizado de tópicos importantes da matemática, como a operação entre conjuntos, de forma prática e interativa. Além disso, a inclusão de feedback visual, como o desenho da forca, agrega um elemento de motivação e engajamento, incentivando o jogador a tentar acertar as respostas corretamente.

Por fim, este projeto contribui não só para o aprendizado de conceitos matemáticos, mas também para o aprimoramento das habilidades em programação, design de interfaces e criação de jogos educativos. É importante destacar que, embora o jogo tenha sido projetado com foco no ensino de matemática, ele pode ser facilmente adaptado para ensinar outros conteúdos, bastando para isso alterar as perguntas e respostas, tornando-o uma ferramenta flexível para diferentes áreas do conhecimento.

REFERÊNCIAS

CUSTOMTKINTER. CustomTkinter: A modern and customizable Tkinter library. Disponível em: <https://github.com/TomSchimansky/CustomTkinter>. Acesso em: 15 nov. 2024.

PYTHON SOFTWARE FOUNDATION. Python Programming Language. Disponível em: <https://www.python.org/>. Acesso em: 15 nov. 2024.

TKINTER. Tkinter documentation: A standard Python interface to the Tk GUI toolkit. Disponível em: <https://docs.python.org/3/library/tkinter.html>. Acesso em: 15 nov. 2024.

W3SCHOOLS. HTML and CSS Reference. Disponível em: <https://www.w3schools.com/>. Acesso em: 15 nov. 2024.

AGRADECIMENTOS

Agradeço ao professor Holisses Bellon pelo período de aprendizagem, foi um professor muito bom, muitas aulas incríveis com muitas explicações perfeitas. Todo esse período e os trabalhos irão ajudar muito na minha vida, e digo que essas aulas foram muito boas para todo mundo.

1. Eu estudei na tecpuc, faço engenharia de software na pucpr, faço um estagio [↑](#footnote-ref-1)