UNIVERSIDADE FEDERAL DO RIO GRANDE DO SUL
INSTITUTO DE INFORMÁTICA
INF01120 – TÉCNICAS DE CONSTRUÇÃO DE PROGRAMAS

Trabalho Final - Etapa 1

Bruno Castanho, Leandra Machado, Lucas Gomes, Maria Eduarda Casali Ricardo, Vitor Arguillar

UFRGS, 2025

Mudanças em Relação à Etapa 0

Não houve mudanças em relação aos detalhes especificados na etapa 0, mas sim a definição final dos pontos que estavam sendo discutidos. Neles, a fim de simplificar, ficou decidido que o jogo será implementado em sua forma clássica com duas dinâmicas adicionais: o jogador pode pedir para a categoria da palavra ser revelada, e para uma dica ser dita, sendo que cada palavra é atrelada a somente uma categoria e dica. Dessa forma, caso tenha dificuldades em acertar a palavra, pode recorrer a esses recursos para evitar perder o jogo.

Requisitos

Os requisitos funcionais abaixo foram deliberados de forma que a jogabilidade do aplicativo fosse priorizada. Dessa forma, observações relativas à lógica de funcionamento e entendimento do usuário em relação ao jogo da forca estão acima de observações que compreendem apenas seu escopo visual.
Já para os requisitos não funcionais, foram priorizadas decisões que proporcionam maior qualidade à aplicação, com foco em medidas referentes à experiência do usuário e à segurança e estabilidade do programa.

Requisitos Funcionais	Requisitos Não Funcionais
RF-1: O jogador deve conseguir acionar o botão regras, de forma que as informações de funcionamento do jogo sejam apresentadas na tela.	**RNF-1**: O jogo deve carregar a interface inicial em no máximo 2 segundos.
RF-2: O jogador deve ser capaz de acionar um botão (via interface ou via teclado, ainda a definir) correspondente à letra a ser adivinhada da rodada.	**RNF-2**: O sistema deve levar no máximo 1 segundo para verificar se alguma ação foi realizada, ou seja, se nenhum botão foi pressionado.
RF-3: O sistema implementado no jogo deve checar se a letra acionada corresponde a alguma presente na palavra secreta e, a partir disso, representar visualmente esse resultado.	**RNF-3**: O sistema deve permitir que o jogador inicie uma nova partida com 1 clique.

RF-4: O sistema deve ser capaz de verificar se o número de vidas do jogador chegou a zero ou se a palavra foi inteiramente adivinhada, encerrando, dessa forma, o jogo atual.	**RNF-4**: Os botões devem ter contraste suficiente para que sejam de fácil visualização.
RF-5: O jogador deve conseguir solicitar, acionando o botão referente a esta ferramenta, a categoria da palavra a ser adivinhada.	**RNF-5**: Toda nova funcionalidade do jogo deve ser validada por revisão de código antes de ser integrada ao programa principal.
RF-6: O jogador deve conseguir solicitar, acionando o botão referente a esta ferramenta, uma dica sobre a palavra a ser adivinhada.	**RNF-6**: As fontes usadas devem ter tamanho mínimo para garantir a legibilidade para o usuário.
RF-7: O jogador deve ser capaz de compreender visualmente quantos chutes ainda possui, através da representação gráfica de suas vidas ou partes do corpo já usadas.	
RF-8: O jogador deve ser capaz de compreender visualmente quais letras já foram selecionadas, através de representação visual que permita isso.	

Projeto

Para a implementação do trabalho final, inicialmente serão seguidas as especificações definidas abaixo, podendo haver modificações conforme sua execução - problemas possivelmente encontrados ou lacunas descobertas no processo:

Classes

Public Class Jogo - administração lógica do jogo
Atributos:

- numvidas: int
- letra: char
- categoria: Categoria

Métodos:

+ main (args: String [ ]): void
+ getnumvidas( ): int
+ getletra( ): char
+ setletra(letra: char): void
+ checavalidadeletra(palavra: Palavra, letra: char): boolean
+ revelaletra(letra: char, palavra: Palavra): void
+ retiravida( ): void

Public Class GUI

Atributos:

- numvidas: int
- tela_atual: int

Métodos:

+ getnumvidas( ): int
+ get_tela( ): int
+ set_tela(tela: int): void
+ setnumvidas(numvidas: int): void
+ exibir_tela_principal( ): void
+ exibir_tela_perdeu(palavra_secreta: Palavra): void
+ exibir_tela_ganhou(palavra_secreta: Palavra): void
+ exibir_regras( ): void
+ desenhaparte_tela(numvidas: int): void

Public Class Palavra

Atributos:

- palavra: String
- numletras: int
- Categoria nome: Categoria

Métodos:

+ cria_lista_palavras( ): List<Palavra>
+ escolhe_palavra_secreta(lista: List<Palavra>): Palavra
+ getpalavra( ): String
+ getnumletras( ): int
+ getcategoria( ): Categoria

Construtor:

+ Palavra(String palavra, Int numletras, Categoria categoria), exemplo: Palavra p = new Palavra(“Voldemort”, 9, P) → Categoria P = new Vilao(“Personagem”) criada previamente

Public abstract Class Categoria → subclasses animais, países, personagens.

Atributos:
- + nomecategoria: String

Métodos:
- + abstract exibircategoria( ): void
- + abstract exibirdica( ): void

Construtor (classes abstratas não são instanciadas diretamente):
- + Categoria(String categoria), exemplo: Categoria c = new Europa(“Paises”)

Public abstract Class Animais extends Categoria

Atributos:
- + (super) nomecategoria: String

Métodos:
- + exibircategoria ( ): void, exemplo (“Categoria: animais.”)
- + abstract exibirdica( ): void

Construtor (classes abstratas não são instanciadas diretamente):
- + Animais(String categoria), Animais a = new Aves(“Animais”)

Public Class Mamifero extends Animais

Atributos:
- + (super) nomecategoria: String

Métodos:
- + exibirdica( ): void, exemplo (“Este animal é um mamífero.”)

Construtor:
- + Mamifero(String categoria), Mamifero m = new Mamifero(“Animais”)

Public Class Aves extends Animais

Atributos:
- + (super) nomecategoria: String

Métodos:
- + exibirdica( ): void, exemplo (“Este animal é uma ave.”)

Construtor:
- + Aves(String categoria), Aves a = new Aves(“Animais”)

Public Class Marinho extends Animais

Atributos:
- + (super) nomecategoria: String

Métodos:
- + exibirdica( ): void, exemplo (“Este é um animal marinho.”)

Construtor:
- + Peixes(String categoria), Marinho p = new Marinho(“Animais”)

Public Class abstract Paises extends Categoria
Atributos:
- + (super) nomecategoria: String

Métodos:
- + exibircategoria ( ): void, exemplo (“Categoria: países.”)
- + abstract exibirdica( ): void

Construtor (classes abstratas não são instanciadas diretamente):
- + Paises(String categoria), Paises p = new America(“Paises”)

Public Class America extends Paises
Atributos:
- + (super) nomecategoria: String

Métodos:
- + exibirdica( ): void, exemplo (“Este país é do continente americano.”)

Construtor:
- + America(String categoria), America a = new America(“Paises”)

Public Class Asia extends Paises
Atributos:
- + (super) nomecategoria: String

Métodos:
- + exibirdica( ): void, exemplo (“Este país é do continente asiático.”)

Construtor:
- + Asia(String categoria), Asia a = new Asia(“Paises”)

Public Class Europa extends Paises
Atributos:
- + (super) nomecategoria: String

Métodos:
- + exibirdica( ): void, exemplo (“Este país é do continente europeu.”)

Construtor:
- + Europa(String categoria), Europa e = new Europa(“Paises”)

Public abstract Class Personagem extends Categoria
Atributos:
- + (super) nomecategoria: String

Métodos:
- + exibircategoria ( ): void, exemplo (“Categoria: personagem.”)
- + abstract exibirdica( ): void

Construtor (classes abstratas não são instanciadas diretamente):
- + Personagem(String categoria), Personagem p = new SuperHeroi(“Personagem”)

Public Class SuperHeroi extends Personagem
Atributos:
- (super) nomecategoria: String

Métodos:
- exibirdica( ): void, exemplo (“Este personagem é um super-herói.”)

Construtor:
- Personagem(String categoria), SuperHeroi p = new SuperHeroi(“Personagem”)

Public Class Animacao extends Personagem
Atributos:
- (super) nomecategoria: String

Métodos:
- exibirdica( ): void, exemplo (“Este personagem é de uma animação.”)

Construtor:
- Animacao(String categoria), Animacao a = new Animacao(“Personagem”)

Public Class Vilao extends Personagem
Atributos:
- (super) nomecategoria: String

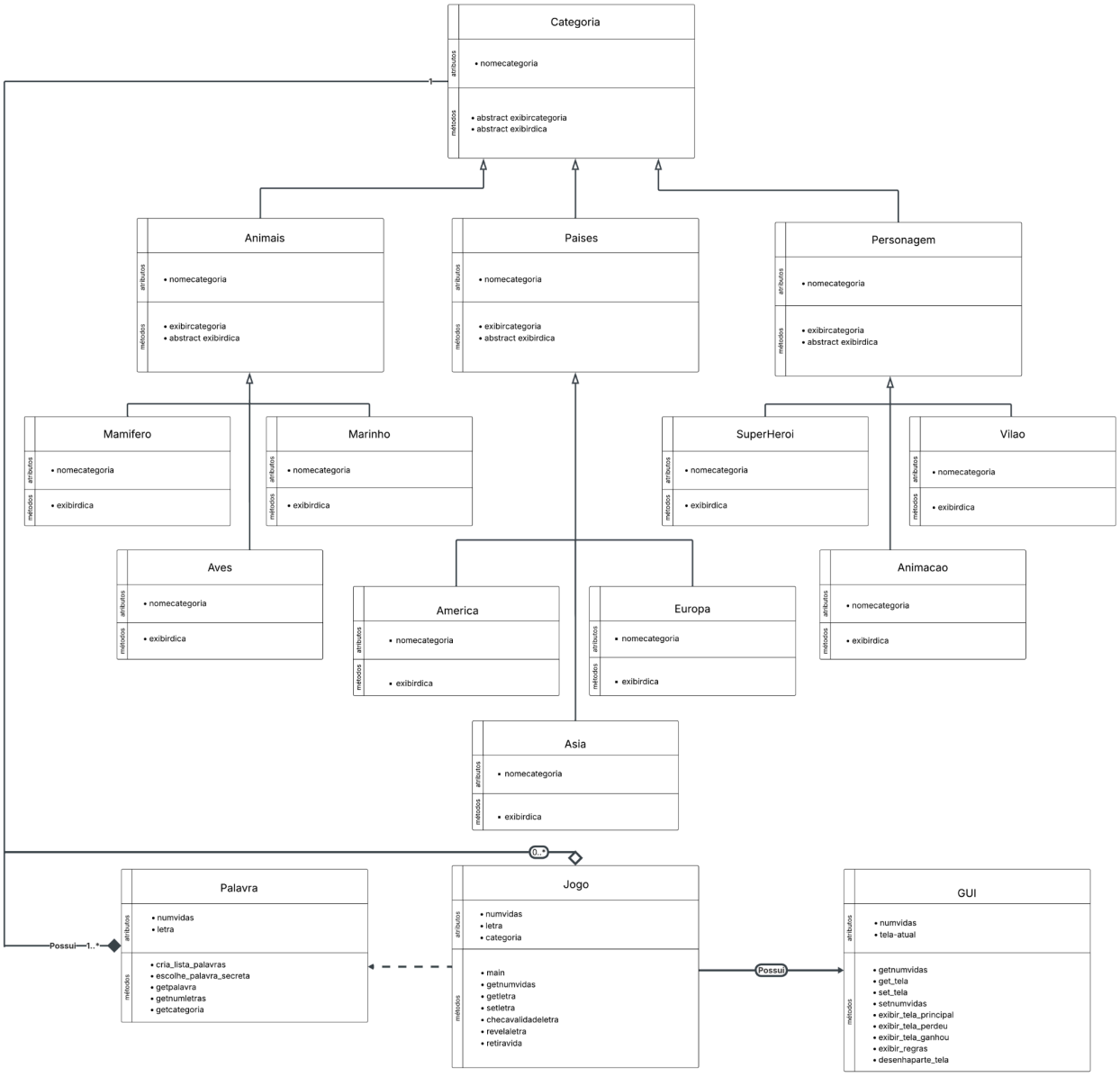
Métodos:
- exibirdica( ): void, exemplo (“Este personagem é um vilão.”)

Construtor:
- Vilao(String categoria), Vilao v = new Vilao(“Personagem”)

Relacionamentos

Dependência: funções da classe Jogo que usam como parâmetro a classe Palavra.
Generalização/Especialização - subclasses em Categoria.
Associação-Agregação - Jogo tem uma Categoria, mas podem existir sozinhos.
Associação-Composição - Palavra com Categoria, sem a existência de Palavra não faria sentido a Categoria existir.

Diagrama



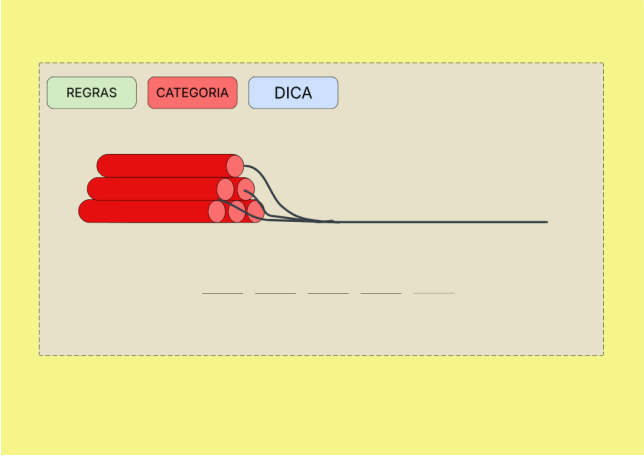
Link para melhor visualização: https://lucid.app/lucidchart/07c0e30e-35e2-4f06-b8b8-b9b15c0c2c50/edit?viewport_loc=-11345%2C-2768%2C16365%2C7686%2C0_0&invitationId=inv_2f991e56-98cf-4a07-8e59-d370e22e5a48

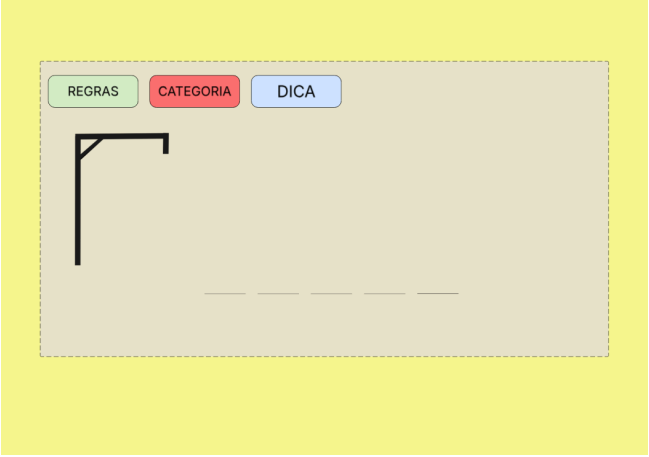
Interface com o Usuário

Para nos guiar no processo de criação do trabalho foram criados protótipos de tela que ilustram a lógica a ser seguida durante o jogo, abaixo seguem as opções no momento, que diferem apenas em questões visuais - a serem deliberadas durante o processo de criação:

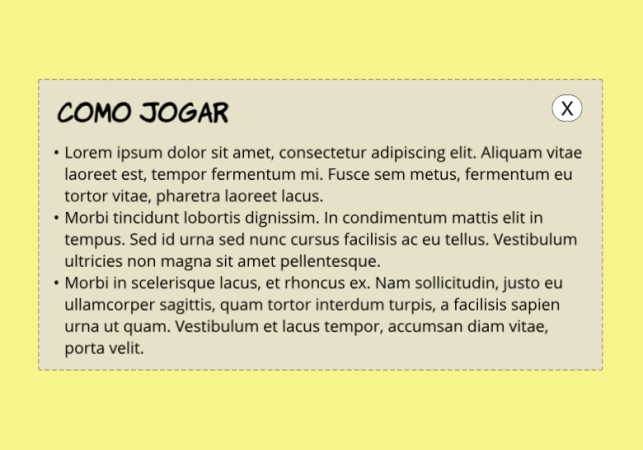
Telas de jogo

1. **Tela ilustrando o menu principal.**

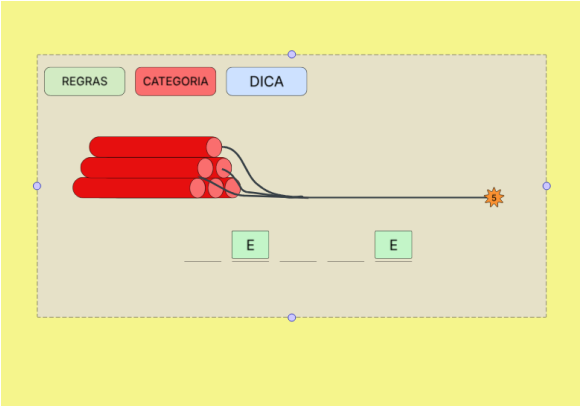


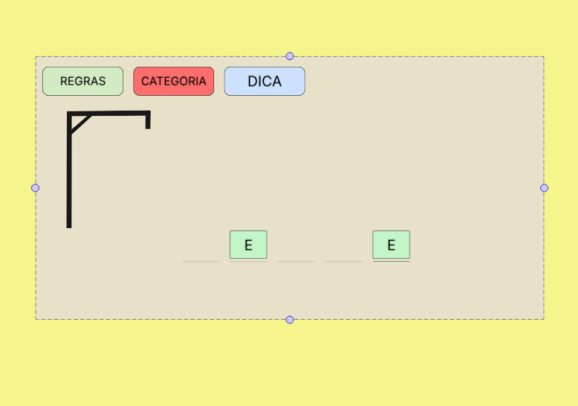


2. **Tela explicando como jogar, disponível quando o usuário aciona o botão “regras”.**

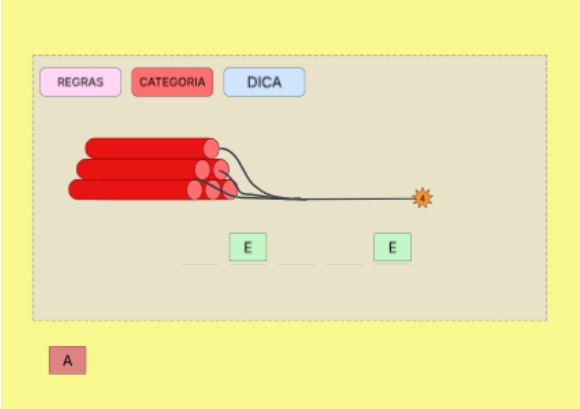


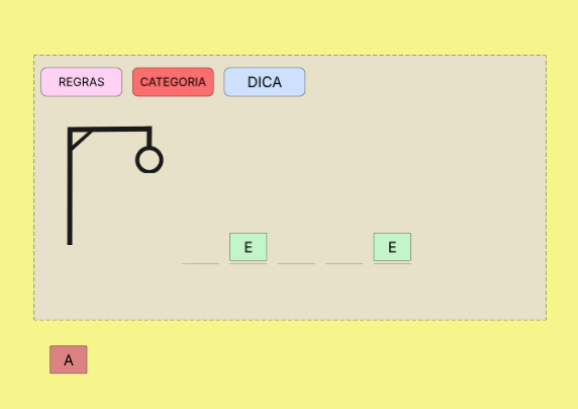
3. **Tela de jogo ilustrando o palpite correto da letra ‘E’, sem erros cometidos pelo usuário.**



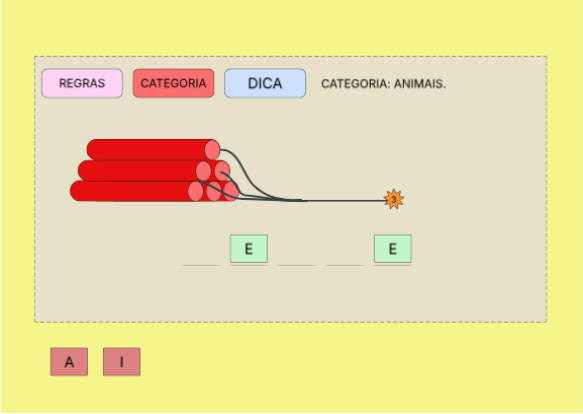


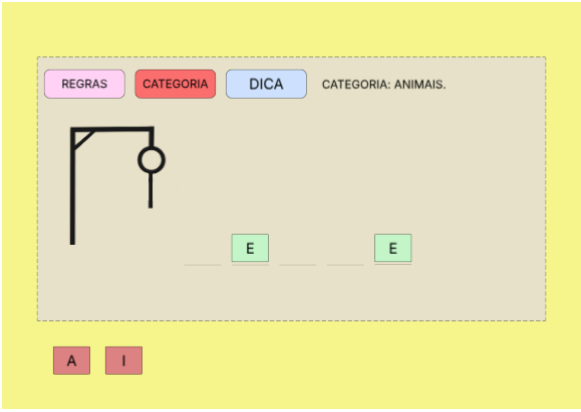
4. **Ilustração de palpite incorreto da letra ‘A’.**



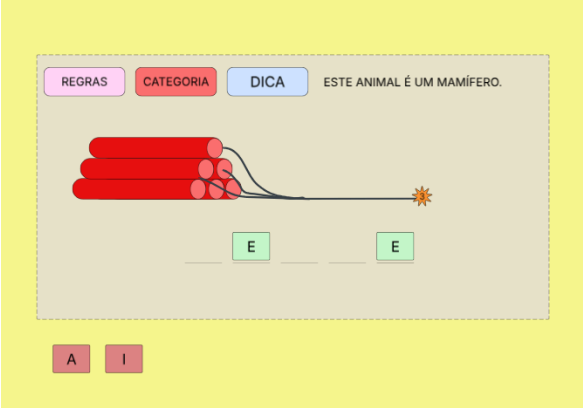


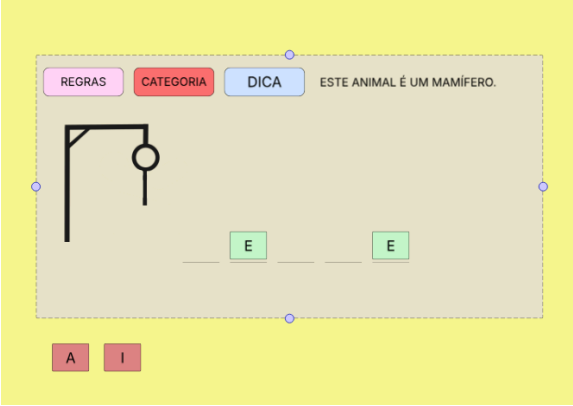
5. **Ilustração de solicitação de categoria. A categoria é exibida ao lado do botão “DICA” (apenas uma categoria por palavra).**





6. **Ilustração de solicitação de dica. A dica é exibida ao lado do botão “DICA” (apenas uma dica por palavra).**





7. **Situação de vitória. A mensagem “Você Venceu” é exibida na tela, a palavra é revelada e o jogador pode escolher entre encerrar o jogo ou reiniciá-lo.**

8. **Situação de derrota. A mensagem “Você Perdeu” é exibida na tela, a palavra é revelada e o jogador pode escolher entre encerrar o jogo ou reiniciá-lo.**