COTUCA - Colégio Técnico de Campinas
Disciplina: Estruturas de Dados
Professor: Francisco da Fonseca Rodrigues
Entrega: 14/06/2025

Projeto II - Jogo da Forca

Jogo de Forca usando lista duplamente ligada no lugar de vetor

Nome: Felipe Antônio de Oliveira Almeida RA: 22130

Nome: Miguel de Castro Chaga Silva RA: 22145

Relatório de Desenvolvimento - Jogo da Forca

Este relatório detalha o desenvolvimento e a implementação das classes Dicionário, Vetor Dicionário e do formulário principal Frm Alunos (Form1) para o projeto do Jogo da Forca, com foco na manipulação de palavras e dicas.

1. Classe Dicionario

A classe Dicionario foi projetada para representar uma única entrada no jogo da forca, composta por uma palavra a ser adivinhada e sua respectiva dica. Ela encapsula as informações essenciais para cada item do jogo.

Campos:

- palavra (string): Armazena a palavra secreta a ser adivinhada.
- dica (string): Armazena a dica associada à palavra.
- acertou (bool[]): Um array de booleanos que provavelmente seria usado para marcar as letras que já foram acertadas na palavra, embora não esteja explicitamente sendo usado para isso no código fornecido do Form1.cs para a forca em si (onde palavraOcultaDisplay é usado).

Propriedades:

- Palavra (string): Propriedade de leitura/escrita para o campo palavra. Definida com private set, indicando que o valor da Palavra só pode ser definido dentro da classe, provavelmente pelo construtor.
- Dica (string): Propriedade de leitura/escrita para o campo dica. Permite acesso e modificação externos.
- Acertou (bool[]): Propriedade de leitura/escrita para o campo acertou.

Construtores:

- Public Dicionario(string palavra, string dica): Construtor principal que inicializa uma nova instância de Dicionario com a palavra e a dica fornecidas.
- public Dicionario(): Construtor padrão que inicializa Palavra e Dica com espaços em branco.

Métodos:

- Ler Dados(StreamReader arq):
    - **Funcionalidade:** Lê uma linha de um StreamReader (arquivo de texto) e extrai a palavra e a dica.
    - **Implementação:** Assume que a palavra ocupa os primeiros tamanho Palavra (15) caracteres da linha e a dica o restante. Converte a palavra para maiúsculas e inicializa o array acertou com 15 elementos falsos.
    - **Observação:** O tamanho Palavra = 15 é uma constante importante. Se palavras maiores que 15 caracteres forem usadas no arquivo, elas serão truncadas. Isso pode ser uma fonte de erro ao processar "palavras grandes", conforme relatado.
- public int CompareTo(Palavra Dica outra) (Implementa IComparable<PalavraDica>):
    - **Funcionalidade:** Permite a comparação de objetos PalavraDica para fins de ordenação, baseando-se na propriedade Palavra.

    - **Implementação:** Utiliza `string.Compare(this.Palavra, outra?.Palavra, StringComparison.OrdinalIgnoreCase)` para uma comparação de strings insensível a maiúsculas/minúsculas.
- `public override string ToString()`:
    - **Funcionalidade:** Sobrescreve o método `ToString()` padrão para fornecer uma representação textual formatada de um objeto `Dicionario`.
    - **Implementação:** Retorna a `Palavra` (com `Trim()`) seguida por " - " e a `Dica`. Essencial para a exibição em `ListBox` ou para depuração.
- `public string FormatoDeArquivo()`:
    - **Funcionalidade:** Define um formato para representar o objeto `Dicionario` ao salvá-lo em um arquivo.
    - **Implementação:** Formata a `Palavra` para ter exatamente 30 caracteres (preenchendo com espaços à direita ou truncando) e concatena com a `Dica`. Isso é uma **inconsistência** com o método `LerDados`, que assume `tamanhoPalavra = 15`. Se a `Palavra` tiver 30 caracteres ao ser salva e 15 ao ser lida, haverá um descompasso. **Esta é uma provável causa de problemas com "palavras grandes".**

2. Classe VetorDicionario

A classe VetorDicionario atua como um gerenciador de uma coleção de objetos Dicionario, utilizando uma estrutura de dados de vetor (array) para armazená-los. Ela oferece funcionalidades básicas de navegação, inserção, pesquisa e exclusão.

Campos:

- dados (Dicionario[]): O array que armazena os objetos Dicionario.
- qtosDados (int): Contador do número de itens atualmente armazenados no array.
- posicaoAtual (int): Índice do item atualmente "selecionado" ou em foco.
- situacaoAtual (Situacao): Enum que descreve o estado atual da operação (navegando, incluindo, etc.).

Propriedades:

- SituacaoAtual (enum): Getter/setter para o estado operacional.
- EstaVazio (bool): Retorna true se qtosDados for 0.
- PosicaoAtual (int): Getter/setter com validação para garantir que a posição está dentro dos limites do array.
- Tamanho (int): Retorna qtosDados.
- RegistroAtual (Dicionario): Retorna o Dicionario na posicaoAtual, se o array não estiver vazio.

Construtores:

- public VetorDicionario(int tamanho): Construtor que inicializa o array dados com o tamanho especificado, qtosDados e posicaoAtual para 0. Define situacaoAtual como navegando.
    - **Observação:** O tamanho fixo do vetor (tamanhoPalavra = 15 em Dicionario.cs e o tamanho do VetorDicionario em Form1.cs como 100 ou 5000) é uma limitação. Se o número de palavras no arquivo exceder a capacidade inicial, ocorrerá um erro de "array out of bounds" ao tentar adicionar novos elementos, contribuindo para o erro em "processar" palavras.

Métodos (Principais destacados):

- Adicionar(Dicionario elemento):
    - **Funcionalidade:** Insere um novo objeto Dicionario no vetor, mantendo a ordem alfabética.
    - **Implementação:** Percorre o vetor para encontrar a posição correta, desloca os elementos existentes para abrir espaço e insere o novo elemento. Incrementa qtosDados.
    - **Potencial Problema:** Se qtosDados atingir o dados.Length (a capacidade máxima do vetor), este método causará um erro de "índice fora do limite" ao tentar adicionar mais elementos.
- Buscar(Dicionario chave):

    - **Funcionalidade:** Pesquisa um Dicionario específico no vetor usando busca binária.
    - **Implementação:** Requer que o vetor esteja ordenado. Retorna true se encontrado, false caso contrário.
- Excluir(Dicionario chave):
    - **Funcionalidade:** Remove um Dicionario do vetor.
    - **Implementação:** Busca o elemento e, se encontrado, desloca os elementos subsequentes para preencher o espaço. Decrementa qtosDados.
- PosicionarNoPrimeiro(): Define posicaoAtual como 0, se houver dados.
- PosicionarNoUltimo(): Define posicaoAtual como qtosDados - 1, se houver dados.
- Proximo(): Avança posicaoAtual.
- Anterior(): Retrocede posicaoAtual.
- LerDados(string nomeArquivo):
    - **Funcionalidade:** Carrega dados de um arquivo de texto para o VetorDicionario.
    - **Implementação:** Lê cada linha, cria um objeto Dicionario e o adiciona usando Adicionar().
    - **Importante:** Este método é crucial para popular o dicionário. Se o formato do arquivo não corresponder ao que Dicionario.LerDados espera (tamanhoPalavra = 15), ou se houver mais palavras do que a capacidade do vetor, ocorrerão erros.
- GravarDados(string nomeArquivo): Salva os dados do vetor em um arquivo de texto.
- ExibirDados(DataGridView grade):
    - **Funcionalidade:** Exibe o conteúdo do VetorDicionario em um DataGridView.
    - **Implementação:** Limpa as linhas existentes, define RowCount e preenche as células da grade com a Palavra e Dica de cada Dicionario.

3. Formulário FrmAlunos (Form1)

O formulário FrmAlunos é a interface principal do usuário, integrando a lógica do jogo da forca com a funcionalidade de cadastro/visualização de palavras e dicas.

Inicialização (FrmAlunos() e FrmAlunos_Load):

- No construtor, InitializeComponent() é chamado primeiro.
- listaPalavras = new ListaDupla<PalavraDica>() é inicializada. (Esta é a lista usada na aba de cadastro).
- dicio = new VetorDicionario(100) (ou 5000) é inicializado para o jogo da forca.
- ConfigurarDataGridView() é chamado.
- botoesAlfabeto = new List<Button>() é inicializada, e os botões do alfabeto são adicionados a essa lista **após InitializeComponent()**, o que é a prática correta.
- No FrmAlunos_Load (conforme Form1.txt):
    - dicio é inicializado (potencialmente duplicado se já no construtor).
    - CarregarDadosDoArquivoJogoDaForca() é chamado, que deve popular o dicio.
    - dicio.PosicionarNoPrimeiro() e AtualizarTela() são chamados.
    - Os botões do alfabeto são **desabilitados** no Load (correto para iniciar com o teclado inativo).
    - btnIniciar, txtNome, chkComDica são habilitados para permitir o início do jogo.
    - A coluna Placeholder é adicionada ao dgv Palavra se ele estiver vazio.

Funcionalidades Principais:

- **Jogo da Forca (tpForca):**
    - btnIniciar_Click: Inicia um novo jogo.
        - Seleciona uma palavra aleatória de dicio.
        - Inicializa palavraOcultaDisplay com StringBuilder.
        - Configura dgvPalavra dinamicamente:
            - Limpa colunas e linhas.
            - Adiciona uma linha (dgv Palavra.Rows.Add();).
            - **Adiciona uma coluna para CADA caractere da palavra secreta**, definindo a largura. (Esta é a solução para "palavras grandes" no dgvPalavra para o jogo).
            - Preenche palavraOcultaDisplay e dgvPalavra com underlines ou caracteres não-letras.
        - Atualiza labels (lbPontos, lbErros, lbDica, lbTempo).
        - Chama ResetarImagens() para a forca.
        - **ReabilitarBotoesAlfabeto() é chamado, habilitando o teclado para o jogo.**
        - Configura e habilita tmrForca se chkComDica estiver marcado.
        - Ajusta estados de btnIniciar, txtNome, chkComDica, btnJogarNovamente.
    - processarLetraClicada(char letra):

        - Verifica se a letra clicada está na palavraSecreta.
        - Atualiza pontos e palavraOcultaDisplay/dgvPalavra.
        - Se a letra não foi encontrada, incrementa erros e chama AtualizarImagemForca().
        - Se erros atinge LIMITE_ERROS_PARA_PERDER (9), deshabilita o teclado (DesabilitarTeclado()) e chama PerdeuOJogo().
        - Se a palavra é adivinhada, desabilita o teclado e chama ImagensGanhou() e MessageBox.Show().
    - tmrForca_Tick: Gerencia o tempo restante.
        - Decrementa tempoForca a cada tick (1 segundo, se Interval for 1000).
        - Se tempoForca atinge 0, desabilita o timer, mostra mensagem de "Tempo esgotado" e chama FinalizarJogo(false).
    - tmrEspiritoBoneco_Tick: Gerencia a animação do espírito.
- **Barra de Status (tmrBarraDeStatus_Tick):**
    - Atualiza slRegistro.Text a cada tick.
    - **Diferencia o conteúdo com base na TabPage selecionada:**
        - Se tpForca está selecionada: exibe "Data: {data} Hora: {hora} {inspiracao}".
        - Se tpCadastro está selecionado, exibe "Registro: {posicaoAtual}/{totalPalavras}".
    - **Dependência:** posicaoAtual e totalPalavras precisam ser corretamente atualizadas pelos métodos de manipulação de dados do cadastro para exibir a informação correta.

Considerações Finais:

O desenvolvimento do projeto evidencia uma estrutura bem definida, utilizando classes específicas para a manipulação e organização dos dados, como Dicionario, VetorDicionario, ListaDupla e NoDuplo, além de uma interface gráfica funcional que facilita a interação com o usuário.

Durante a implementação, observou-se que os principais desafios estão relacionados à robustez no tratamento dos dados, especialmente na gestão de strings com tamanhos fixos e na limitação da capacidade das coleções. Tais aspectos demandam atenção criteriosa para evitar inconsistências e falhas no funcionamento da aplicação.

Outro ponto crítico identificado é a necessidade de garantir que todas as coleções de dados — como listaPalavras e dicio — sejam corretamente populadas, manipuladas e esvaziadas de acordo com o ciclo de vida da aplicação, assegurando a integridade dos dados e o bom desempenho do sistema.

De modo geral, o projeto cumpre seus objetivos, apresentando uma arquitetura sólida, com organização modular e uma interface intuitiva.