

ATENÇÃO:

**ESTE PROJETO É PARTE OBRIGATÓRIA DA AVALIAÇÃO
DA DISCIPLINA DE APC E TEM PESO 3 NA NOTA FINAL.**

**ALÉM DISSO, CASO O(A) ALUNO(A) NÃO OBTENHA
NOTA MÍNIMA 5,0 NO PROJETO, O(A) ALUNO(A)
RECEBERÁ MENÇÃO MI CONFORME INFORMADO NO
PLANO DE ENSINO NO INÍCIO DO SEMESTRE.**

IMPORTANTE:

- INDIVIDUAL (utilizaremos detector de plágio)**
- NÃO PERMITIDO USO DE QUALQUER IA**
- LEMBRE-SE: você pode ajudar seu colega com o
código dele(a), mas jamais compartilhe seu código!**
- ENTREGA: 15/12/2025 (2a) ATÉ 23:55hs NO MOODLE**

Projeto Final

O objetivo deste projeto é que o aluno aprenda a usar **estruturas de dados do tipo vetor, matriz, strings, subalgoritmos (funções), macros do pré-processador, bibliotecas diferentes das utilizadas até então, registros, arquivos, e variáveis globais de forma correta** em algoritmos, bem como implementá-los corretamente em programas.

Problema: Implementar um jogo (2D) **baseado** no jogos *Word Spells* (*Malpa Games*) e *Zen Word* (*Oakever Games*) cujos ícones são mostrados respectivamente nas Figura 1 e 2 e estão disponíveis para dispositivos móveis Android e IOS.

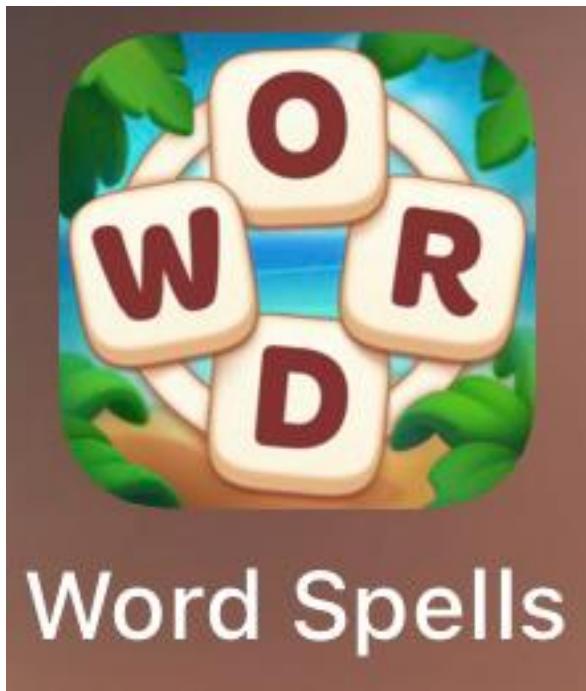


Figura 1: Jogo *Word Spells* (*Malpa Games*)



Figura 2: Jogo *Zen Word* (*Oakever Games*)

O jogo consiste na apresentação de letras isoladas as quais devem ser conectadas para que o jogador forme palavras utilizando as mesmas. As palavras são previamente definidas pelo jogo, porém o jogador as desconhece, sabendo-se apenas o número de palavras e o tamanho de cada uma. Nos jogos referência que serviram de inspiração para este projeto, o jogador utiliza a tela multitoque para deslizar o dedo de modo a conectar as letras e formar a palavra. Porém, neste projeto, utilizaremos o teclado como entrada de dados, onde o jogador digitará a palavra que julga estar correta, utilizando as letras

fornecidas pelo jogo. Lembrado que cada letra fornecida, pode ser usada apenas uma vez na palavra. Uma vez que a palavra foi encontrada, ela deve ser mostrada na tela e não poderá mais ser contabilizada como uma nova palavra, ou seja, ela só pode ser encontrada uma única vez. O jogador ganha pontos ao encontrar uma palavra. Cada vez que o jogador encontra uma palavra que não consta na lista de palavras a serem encontradas, ele é informado do “ERRO” e perde uma pequena pontuação por isso. Ao encontrar todas as palavras da fase, o jogador é parabenizado pela conclusão da fase e acumula também uma pontuação extra pela conclusão da fase. Após o fim da fase, é perguntado ao jogador se ele deseja prosseguir para a próxima fase. O jogo se encerra quando não houver mais fases para serem carregadas do arquivo texto de entrada, ou seja, o jogador “zerou” o jogo.

Detalhamento:

As unidades de entrada deste projeto deverão ser um arquivo tipo texto em disco e o teclado. As unidades de saída serão o monitor de vídeo e um arquivo binário em disco para armazenamento do ranking do jogo.

Para facilitar o manuseio do programa, deverão ser dadas mensagens explicativas em alguns casos conforme indicado posteriormente nesta especificação.

Inicialmente deve ser mostrada uma tela de boas vindas com o nome do jogo (escolhido pelo desenvolvedor) onde é solicitado o nickname do jogador (Figura 3). Em seguida deve ser apresentada ao usuário a tela do menu principal, até que o usuário digite o número da opção que deseja e tecle <enter>. Conforme mostrado na Figura 4.



Figura 3: Tela de Boas Vindas

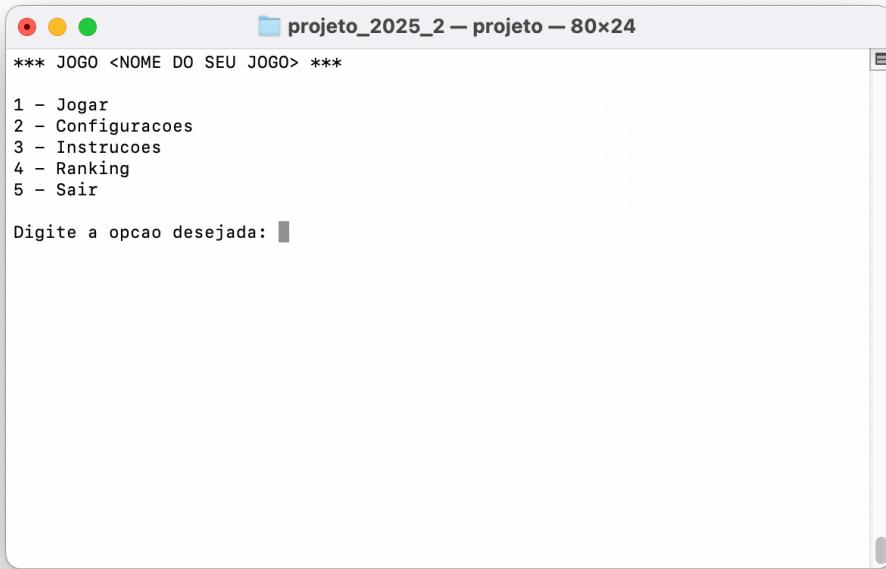


Figura 4: Menu Principal

A seguir será detalhado o quê deve ser feito em cada opção do Menu do Jogo.

1. JOGAR

Inicialmente o jogo deve funcionar na sua configuração básica, ou seja, utilizando o arquivo texto fornecido pela professora (`entrada.txt`). Posteriormente, nesta especificação, em EXTRAS (opcionais!) serão indicadas algumas sugestões de melhorias e características adicionais que poderão ser feitas para ganho de pontos extras, para quem desejar fazê-la(s) (**não obrigatório!**).

Tela inicial da Opção 1. JOGAR:

O jogo deve mostrar inicialmente a primeira fase que foi lida do arquivo texto. A Figura 5 ilustra o texto que será explicado a seguir. Primeiramente as letras da fase devem ser mostradas em uma matriz distribuídas aleatoriamente de forma espaçada. Logo abaixo deve aparecer uma lista das palavras a serem encontradas (e a quantidade delas), contudo inicialmente essa lista está vazia e aparecem somente os traços sublinhados para cada letra de cada palavra. Isso auxilia o jogador a identificar o tamanho das palavras que ele precisa encontrar para concluir a fase.

Objetivo do jogo:

O objetivo do jogo é passar por todas as fases que contém o arquivo de entrada, ou seja, “zerar o jogo”. Em cada fase, o objetivo do jogador é descobrir todas as palavras da fase. Não há limites de tentativas, ou seja, para o número de palavras que o jogador pode informar. Contudo, ele perde 10 pontos para cada palavra errada informada ou para cada palavra encontrada informada novamente. Isto significa que ele pode inclusive ficar com pontuação negativa durante o jogo.

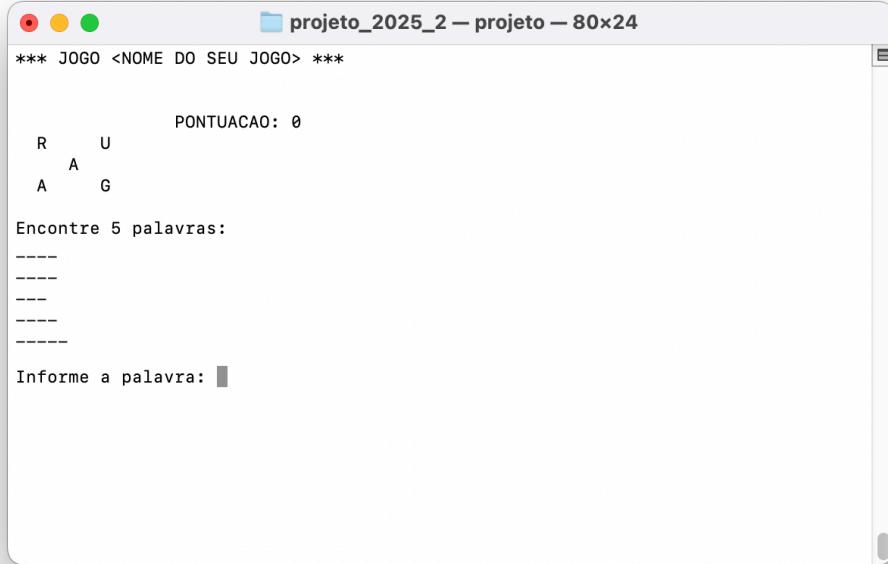


Figura 5: Tela inicial da Opção JOGAR

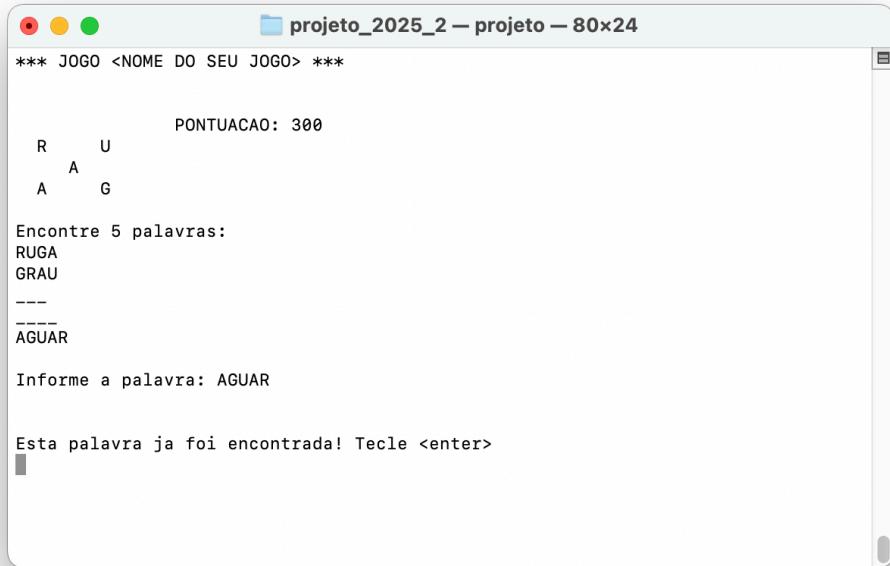
Para jogar, deve ser pedido que o jogador informe a palavra. Após, o programa deve verificar se a palavra informada consta na lista de palavras daquela fase. Caso conste, verificar se ela já não foi encontrada, pois caso tenha sido encontrada, uma mensagem deve ser dada ao usuário e ele perderá 10 pontos por isso. (Figura 6) Caso a palavra não tenha sido encontrada ainda, mas seja uma palavra da lista, o usuário deve ser parabenizado, receber 100 pontos, a lista de palavras encontradas do programa deve ser atualizada e esta palavra deverá ser mostrada na tela junto com as demais palavras encontradas (Figuras 7 e 8). Caso a palavra informada não esteja na lista das palavras da fase, uma mensagem de erro deve ser dada ao jogador e 10 pontos devem ser descontados (Figura 9).

Pontuação:

A pontuação do jogador é determinada pelo acerto de palavras e pelo fechamento da fase. Ele ganha 100 pontos a cada palavra acertada e mais 25 pontos extras quanto conclui a fase. Porém ele perde 10 pontos para cada palavra que ele erra ou palavra já encontrada que ele informa novamente. A pontuação do jogador deve sempre ser mostrada na tela ao lado direito superior das letras.

Fim de Jogo:

Conforme explicado acima, a fase encerra quando todas as palavras da fase tiverem sido encontradas pelo jogador (Figura 10). O fim do jogo se dá quando o jogador passar por todas as fases que estiverem no arquivo texto de entrada do jogo. Ao final de cada fase concluída deve ser perguntado se ele deseja continuar para a fase seguinte. Ou seja, o jogador tem a possibilidade de sair do jogo, antes de concluir todas as fases. Em ambos os casos: (1) interromper antes de concluir todas as fases ou (2) concluir o jogo por completo, o nível do jogador deve ser gravado no ranking conforme será explicado abaixo.



```
projeto_2025_2 — projeto — 80x24
*** JOGO <NOME DO SEU JOGO> ***

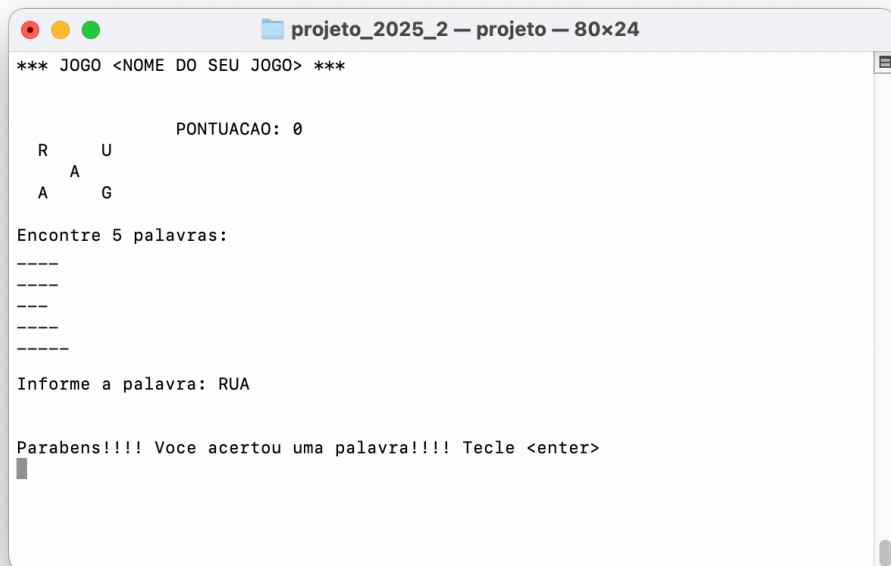
PONTUACAO: 300
R      U
      A
A      G

Encontre 5 palavras:
RUGA
GRAU
---
-----
AGUAR

Informe a palavra: AGUAR

Esta palavra ja foi encontrada! Tecle <enter>
```

Figura 6: Palavra da fase já foi encontrada antes



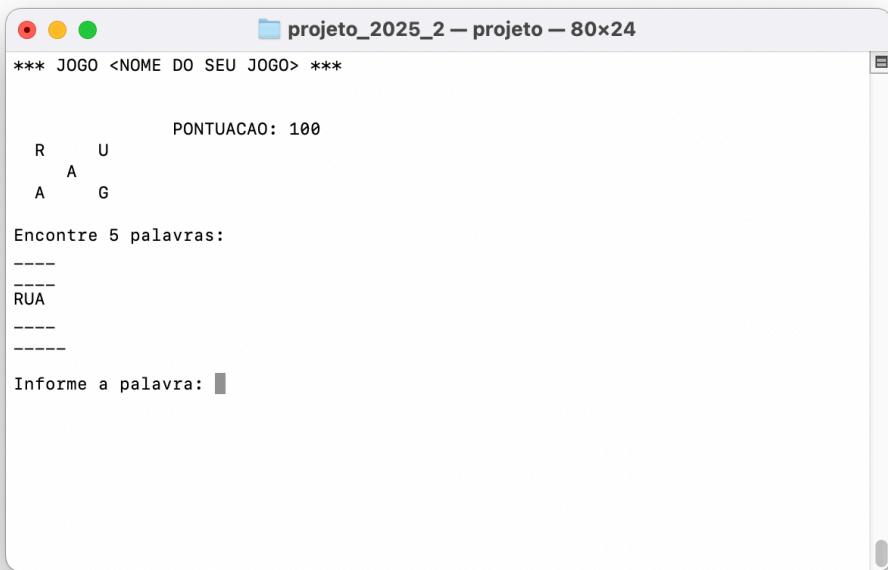
```
projeto_2025_2 — projeto — 80x24
*** JOGO <NOME DO SEU JOGO> ***

PONTUACAO: 0
R      U
      A
A      G

Encontre 5 palavras:
---
-----
---
-----
-----
Informe a palavra: RUA

Parabens!!!! Voce acertou uma palavra!!!! Tecle <enter>
```

Figura 7: Quando uma palavra da fase é encontrada

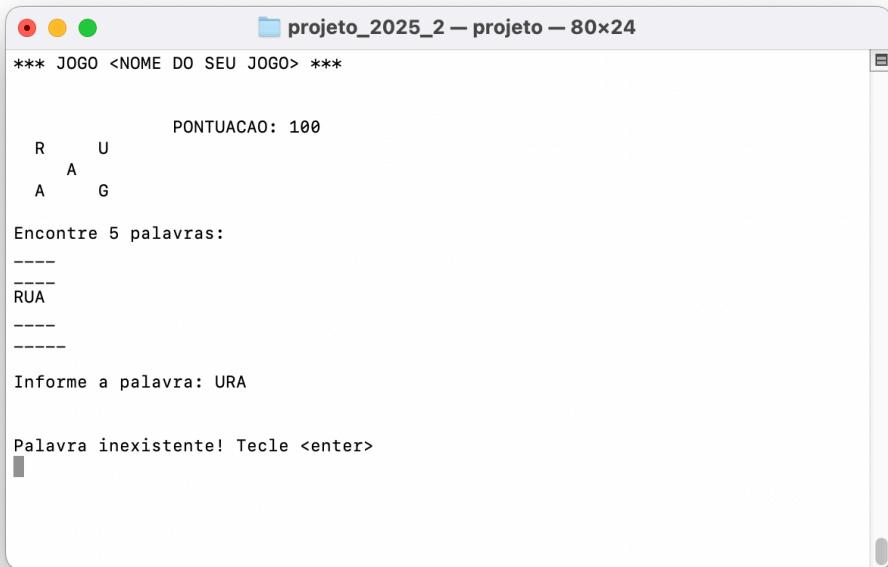


```
projeto_2025_2 — projeto — 80x24
*** JOGO <NOME DO SEU JOGO> ***

PONTUACAO: 100
R     U
A
A     G

Encontre 5 palavras:
-----
RUA
-----
-----
Informe a palavra: 
```

Figura 8: Atualização da lista e pontuação após palavra da fase ser encontrada



```
projeto_2025_2 — projeto — 80x24
*** JOGO <NOME DO SEU JOGO> ***

PONTUACAO: 100
R     U
A
A     G

Encontre 5 palavras:
-----
RUA
-----
-----
Informe a palavra: URA

Palavra inexistente! Tecle <enter>
```

Figura 9: Palavra informada não está na lista de palavras da fase

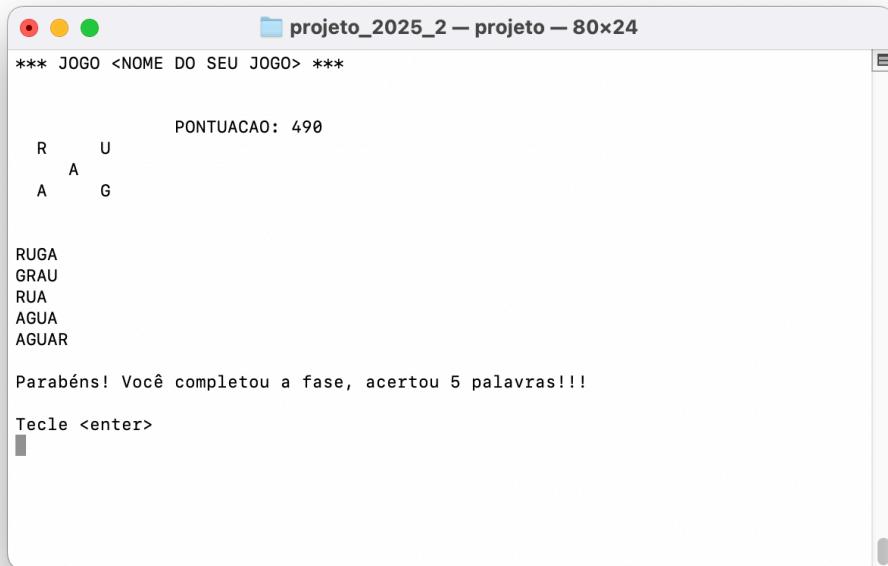


Figura 10: Fase concluída

Gravação no Ranking:

Caso o arquivo `ranking.bin` não exista na primeira vez da gravação, ele deve ser criado. O nickname e o respectivo nível atingido devem ser inseridos no ranking, mesmo que este jogador já esteja lá, pois ele(a) pode obter diferentes níveis em diferentes partidas. Lembrando que caso o arquivo `ranking.bin` já exista ele deve ser aberto no modo **append** para a inserção de um registro ao seu final.

Porém, lembre-se que na hora de consultar o ranking (Opção 4 do Menu Principal) o mesmo deve ser mostrado de **forma ordenada decrescente**. Para isso, o arquivo deverá ser carregado em memória em um vetor de structs para ser ordenado pela pontuação, antes de ser mostrado na tela. Obs: Use o algoritmo *Bubblesort* visto em aula para ordenar o vetor de structs contendo o ranking.

As gravações no arquivo `ranking.bin` devem ser feitas sempre em struct ou vetor de struct:

```
struct dadosjogador {
    char nickname[20];
    int pontuacao;
};
```

Organização do arquivo `entrada.txt`:

O dados da partida estão armazenados em um arquivo do tipo texto chamado `entrada.txt`. A organização do arquivo se dá da seguinte forma: Cada fase está separada por um caracter hífen (-), ou seja, ao ler este caracter o programa deve parar de carregar a fase e começar a executar o jogo. Na primeira linha da fase consta uma string composta pelas letras que o jogador poderá usar para formar as palavras. Em seguida há um caracter numérico que indica a quantidade de palavras da fase, as quais

aparecem nas linhas subsequentes, sendo uma palavra por linha. Após a última palavra aparece o caracter hífen (-) de final de fase. O arquivo deve ser lido a partir do seu início, de modo que a primeira fase seja carregada na memória do computador para as devidas estruturas e/ou variáveis; enquanto isso o arquivo permanecerá aberto de modo que a leitura da próxima fase só irá ocorrer após a fase atual terminar. Lembre-se que a priori, o programador não sabe o tamanho do arquivo .txt, então você vai abrindo-o apenas para leitura, e vai ler (com `fscanf`) linha apóis linha, e isso será feito enquanto o retorno do `fscanf` for diferente da marca de final de fase (-, hífen) e diferente de EOF (End of File). O arquivo `entrada.txt` sempre será aberto apenas para leitura!

2. CONFIGURACOES

Ao escolher a opção 2. CONFIGURACOES, deve ser mostrado ao jogador um novo menu (Figura 11). **A seguir serão detalhadas as opções do Menu CONFIGURACOES:**

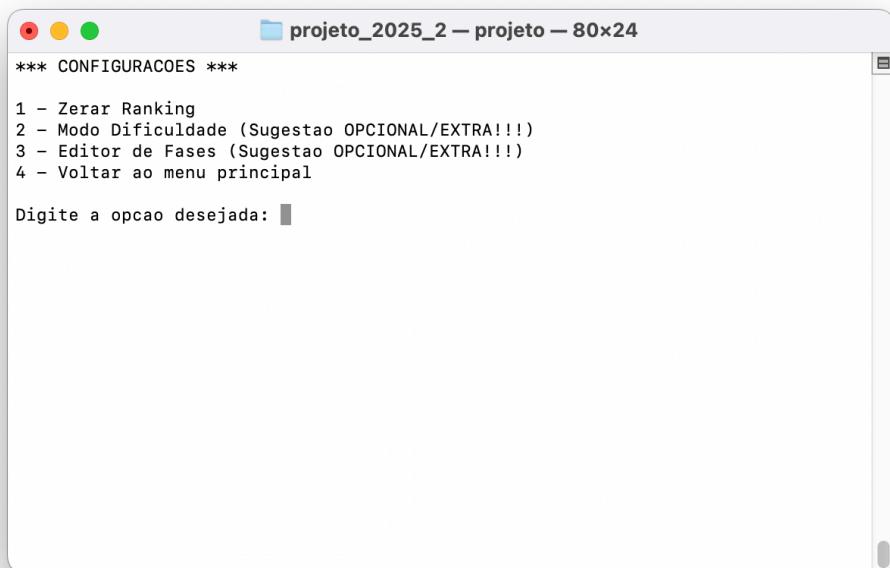


Figura 11: Menu Configurações

(MENU CONFIGURACOES) **1. ZERAR RANKING:** nesta opção o programa deve zerar completamente o arquivo `ranking.bin`, ou seja, deve recriá-lo vazio. É necessário perguntar ao usuário se ele confirma a operação, e dar uma breve mensagem mostrado nas Figuras 12 e 13. Caso o usuário digitar “N” ou “n” nenhuma ação deve ser executada no arquivo `ranking.bin`.

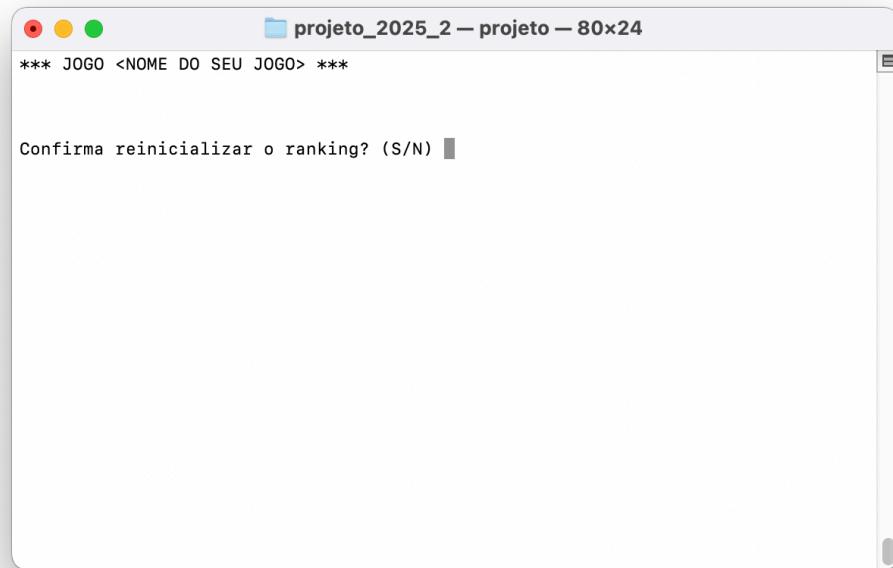


Figura 12: Confirmando reinicialização do ranking.

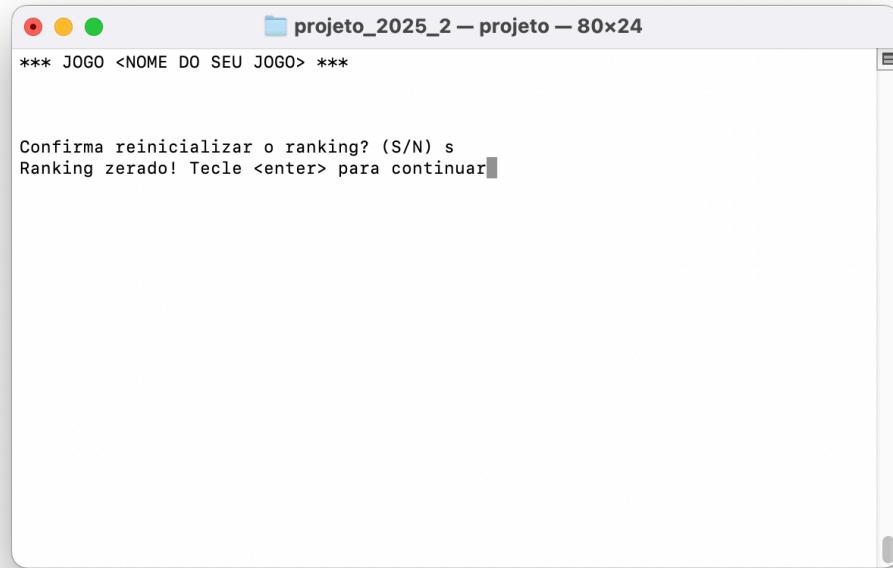


Figura 13: Mensagem de Ranking Zerado

(MENU CONFIGURACOES) 2. MODO DIFICULDADE: esta opção deve constar no Menu mas sua implementação não é obrigatória. Caso o aluno não implemente ela, quando a Opção 2 for selecionada simplesmente nada acontece, ou pode ser data uma mensagem do tipo: “Em desenvolvimento...”. Caso o aluno deseje implementar algum modo de dificuldade, nesta opção jogador poderá definir o modo de dificuldade atual, lembrando que o jogo sempre começa no modo FÁCIL. Algumas sugestões são: ter fases de dificuldades média ou difícil em outros arquivos .txt (que não o ENTRADA.TXT), e ao alterar o modo de dificuldade o programa abre outro arquivo para jogar. Ou também o programador pode inventar algum tipo de modo de dificuldade próprio. A criatividade é bem vinda! O importante é lembrar que neste opção o usuário poderá alterar o modo de dificuldade atual.

(MENU CONFIGURACOES) 3. EDITOR DE FASES: esta opção deve constar no Menu mas sua implementação não é obrigatória. Caso o aluno não implemente ela, quando a Opção 3 for selecionada simplesmente nada acontece, ou pode ser data uma mensagem do tipo: “Em desenvolvimento...”. Nesta opção, o seu programa pode dar ao jogador a possibilidade de ele criar novas fases (novos arquivos tipo texto para entrada do jogo) por meio do próprio programa, ou seja, o jogador pode digitar os dados para as fases no próprio jogo e o programa se encarrega de montar as fases adequadamente dentro do arquivo texto, inserindo os separadores de fase e validando os requisitos obrigatórios para montar uma fase, como: string com as letras, quantidade de palavras, e as palavras propriamente ditas.

IMPORTANTE: Lembrando, que caso este EXTRA seja implementado, o jogador pode dar nome para cada arquivo de fase que ele criar e, do mesmo modo, ao selecionar a opção jogar, ele pode informar o nome do arquivo que deseja carregar para jogar.

(MENU CONFIGURACOES) 4. VOLTAR AO MENU PRINCIPAL: nesta opção o programa deve retornar para o menu principal/inicial do jogo.

Aqui encerramos a descrição do sub-menu CONFIGURACOES e prosseguimos com a explicação do menu principal

3. INSTRUÇOES

A opção INSTRUÇOES deve mostrar um texto explicativo sobre o funcionamento do jogo, as ações que o jogador pode fazer, o objetivo do jogo, pontuação, evolução, fases, regras, formas de encerramento do jogo, etc. A Figura 14 mostra um exemplo ainda sem as instruções escritas. Ao final uma mensagem para prosseguir deve ser mostrada até que o usuário pressione <enter>. Após isso, deve-se retornar ao menu principal.

4. RANKING

Ao escolher a opção 4. RANKING, deve ser mostrado, em uma nova tela, o ranking atual (que pode estar vazio ou não), ordenado de forma decrescente (ou seja, do maior para o menor) por ordem de pontuação (Figura 15).

5. SAIR

Ao escolher a opção 5. SAIR, o programa deve ser encerrado.

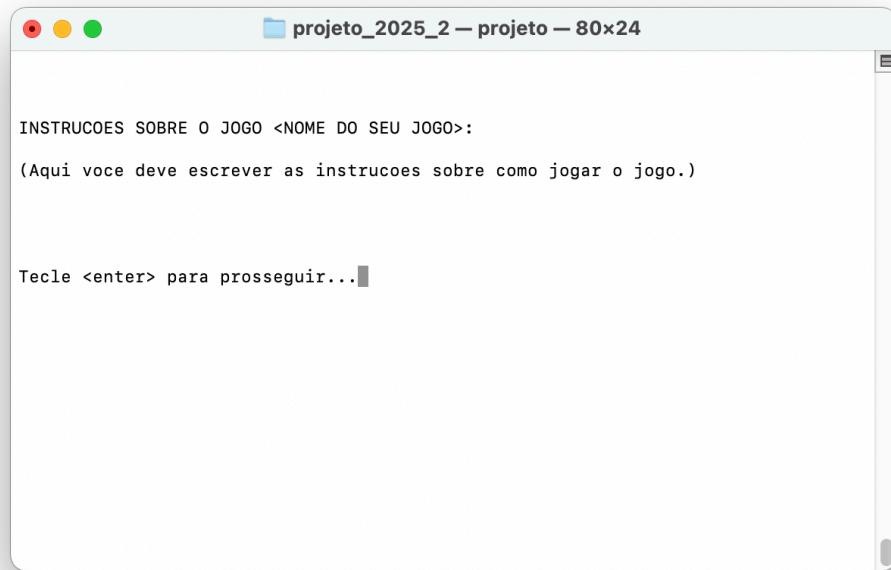


Figura 14: Exemplo da tela de instruções do jogo

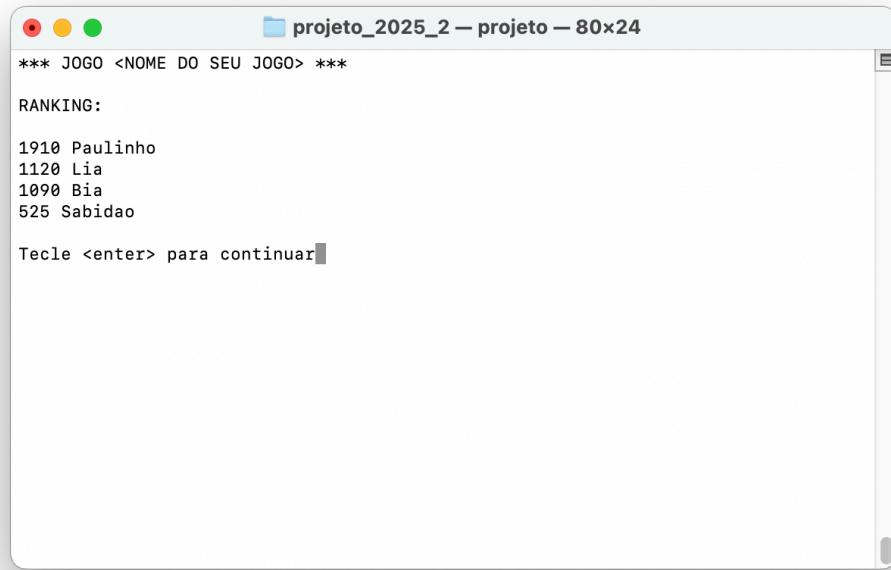


Figura 15: Mostrando ranking em ordem decrescente de pontuação

(Nas próximas páginas veja o APÊNDICE com detalhes de implementação que irão te auxiliar no desenvolvimento do projeto!

APÊNDICE:

Neste projeto você DEVE modularizar o seu programa criando funções. Conforme já dito em sala de aula, funções tornam o seu programa mais legível, mais fácil de ser entendido, e mais curto, já que evita a duplicação de código para uma mesma ação. Será de total responsabilidade do programador decidir quais funções serão criadas, definindo seus tipos, conteúdo e funcionalidade.

Seguem algumas sugestões de funções:

- Funções para mostrar cada menu na tela (são 2 menus!);
- Função para executar o jogo ou jogar quando a Opção 1 do Menu Principal for selecionada; esta função contém o que chamamos de “main game loop” (ou loop principal do jogo);
- Função que verifica se a palavra digitada está na lista de palavras da fase;
- Função para limpar a tela;
- Função para mostrar o painel das letras da fase na tela;
- Função para ler o ranking do arquivo, ordenar e mostrar o ranking na tela (lembrando que se desejar você pode guardar o ranking ordenado e fazer esta ordenação no momento em que adicionar um novo jogador nele);
- Função para gravar no ranking um novo jogador e sua pontuação;
- Função para zerar o ranking;
- Função para mostrar as instruções (Opção 3 do menu principal);
- etc...

Novo cabeçalho, Macros e variáveis globais: neste trabalho, será necessário o uso de macros do pré-processador, o cabeçalho *stdlib.h* e variáveis globais.

1) Das macros do pré-processador

As diretivas do pré-processador, isto é, os comandos iniciados com o caractere # que incluímos no início de nossos arquivos de código-fonte são meios de realizar modificações em um programa em tempo de compilação. Cada uma das diretivas instrui o pré-processador a realizar uma modificação no código-fonte, antes que o compilador vá, de fato, gerar o arquivo executável de um programa.

Até agora, conhecemos apenas a diretiva *#include*. Esta diretiva diz ao pré-processador para, literalmente, substituir sua declaração por todo o código que está dentro do arquivo especificado.

Neste trabalho, utilizaremos uma nova diretiva: o *#define*. Esta diretiva diz ao pré-processador para, literalmente, substituir a **macro definida** em todos os lugares que ele aparecer no código-fonte por sua **expansão**. **Por exemplo:** no código abaixo, será compilada a instrução *printf("%d\n", 50)* e não a instrução *printf("%d\n", MAX_JOGOS)*. Além do mais, o compilador não saberia como compilar esta última instrução, já que não declaramos nenhuma variável com o identificador *MAX_JOGOS*. Vejam:

```
#define MAX_JOGOS 50
...
printf("%d\n", MAX_JOGOS);
...
```

Por ser, tradicionalmente, uma boa prática de programação, definimos macros utilizando caracteres maiúsculos, para que seja fácil identificá-los em um código-fonte. Isto é, ao visualizarmos identificadores em que estão presentes apenas caracteres maiúsculos, podemos nos assegurar de que este identificador é uma macro, um *#define*.

É possível combinar diretivas utilizando diretivas condicionais, exemplo:

```
#ifdef _WIN32
    #define CLEAR "cls"
#else
    #define CLEAR "clear"
#endif
```

As diretivas acima definem uma macro com um comando utilizado para limpar a tela, de acordo com o sistema operacional em que o código-fonte for compilado. Quando um compilador está sendo executado no Windows, a macro `_WIN32` fica definida no ambiente e pode ser utilizada em diretivas do pré-processador. O trecho acima exemplifica uma maneira de escrever código que compila e executa corretamente em diferentes sistemas operacionais.

Observação: No Windows, o comando de terminal “`cls`” de fato limpa todo o terminal. No Linux, o comando equivalente “`clear`” apenas coloca algumas quebras de linha e sobe a tela.

2) switch – Uma estrutura condicional extremamente recomendada para a implementação de **MENUS** é a estrutura `switch`. Esta estrutura é apenas uma maneira simplificada de escrever uma sequência `if-else if-else if-...-else if-else`, com a diferença de que o teste condicional realizado apenas verifica se um valor inteiro bate com uma das constantes selecionadas. Exemplo para o menu principal do seu jogo:

```
switch (menuMain()) {
    case MENUMAIN_JOGAR: jogar(); break;
    case MENUMAIN_INSTRUCOES: instrucoes(); break;
    case MENUMAIN_CONFIGURACOES: configuracoes(); break;
    case MENUMAIN_RANKING: ranking(); break;
    case MENUMAIN_SAIR: sair(); break;
    default:
        exit(EXIT_FAILURE);
}
```

Uma sequência equivalente seria:

```
MenuMainOpt opt = menuMain();
if (opt == MENUMAIN_JOGAR)
    jogar();
else if (opt == MENUMAIN_CONFIGURACOES)
    configuracoes();
else if (opt == MENUMAIN_INSTRUCOES)
    instrucoes();
else if (opt == MENUMAIN_RANKING)
    ranking();
else if (opt == MENUMAIN_SAIR)
    sair();
else
    exit(EXIT_FAILURE);
```

3) Das variáveis globais

No início do curso, vimos que variáveis globais podem confundir o programador ao manipular seus dados e decidimos não utilizá-las. No entanto, em alguns casos, a melhor opção pode ser utilizar

variáveis globais, mas o cuidado deve ser mantido. O objetivo desta seção é mostrar formas de se detectar em quais situações uma variável deve ser global. A seguir são indicadas algumas variáveis que podem ser globais neste trabalho.

Abaixo segue um **exemplo**, das variáveis globais que a professora usou no programa exemplo do jogo (mostrado em aula).

ATENÇÃO: você deve criar as suas próprias variáveis globais, conforme achar necessário. **Não é obrigatório** utilizar as variáveis abaixo.

```
FILE* fp; //ponteiro para o arquivo
char nickname[20], arquivo[20];
int pontuacao=0;
char painel_letras[3][5]; //matriz que guarda as letras da fase
char lista_palavras[10][15]; //matriz que guarda em cada linha uma
                             //palavra da lista da fase carregada
int palavras_corretas[10];//vetor que guarda 0 ou 1 indicando se cada
                           //palavra foi ou não encontrada. Cada
                           //posição do vetor representa uma palavra
```

É possível que outras variáveis também precisem ser globais neste programa. Fica a cargo do programador tomar as decisões sobre quais variáveis devem ser globais. **Na correção do trabalho, estas decisões serão avaliadas.**

4) Outras observações

Função system() - utilize a função `system()` do cabeçalho “`stdlib.h`” para limpar o terminal, a cada vez que o programa for exibir uma nova tela. Utilizando a dica do item 1 do Apêndice, basta realizar a chamada `system(CLEAR)` ;

Crie variáveis e nomes de funções mnemônicos!!!

Utilize variáveis globais **com cautela** conforme explicado acima.

Nas funções, utilize **passagem de parâmetros por valor e por referência**, conforme necessário. Lembrando que as funções podem não receber nenhum parâmetro, como por exemplo, quando estão trabalhando apenas com variáveis globais. Além disso, podem ser do tipo `void` caso não retornem nada.

Faça diversos testes no seu jogo, jogando é claro. Bom trabalho e boa diversão!

Observações Gerais:

1. Incluir cabeçalho como comentário, informando sua matrícula, seu nome completo, e qualquer outra informação que você julgar necessário informar sobre o seu código.
2. Se você implementar algum EXTRA, enviar junto com o seu código fonte um arquivo `readme.txt` explicando o que foi implementado.
3. A data de entrega do programa é: 15/12/2025 (2a) até às 23:55 hs pelo link disponível no Moodle.