

Manual de Construção do Jogo "Batalha Naval" em Assembly

Introdução

Este manual apresenta, de forma detalhada, o processo de construção de um jogo de "Batalha Naval" implementado em linguagem Assembly para o ambiente DOS. O objetivo é fornecer uma compreensão aprofundada de toda a lógica e estrutura do programa, visando atender aos requisitos de um projeto final de curso em nível superior.

Visão Geral do Jogo

"Batalha Naval" é um jogo clássico de estratégia em que dois jogadores posicionam navios em um tabuleiro e tentam adivinhar as posições dos navios do oponente. Neste projeto, o jogo foi adaptado para um único jogador, que tenta afundar todos os navios posicionados em um tabuleiro de 10×10 posições.

Requisitos

- **Ambiente de Desenvolvimento:** O programa foi desenvolvido para ser executado em sistemas que suportem o DOS, utilizando o MASM (Microsoft Macro Assembler) como compilador.
- **Linguagem:** Assembly x86, utilizando interrupções do DOS para manipulação de entrada/saída.
- **Funcionalidades:**
 - Exibir uma interface textual do tabuleiro.
 - Permitir que o jogador insira coordenadas para atacar.
 - Informar ao jogador se o ataque foi um acerto ou erro.
 - Atualizar o tabuleiro com os resultados dos ataques.
 - Finalizar o jogo quando todos os navios forem afundados ou quando as tentativas acabarem.

Estrutura do Programa

O programa está organizado em duas principais seções:

1. **Segmento de Dados (.DATA):** Contém a definição das matrizes de jogo, mensagens exibidas ao usuário e variáveis de controle.
2. **Segmento de Código (.CODE):** Inclui o ponto de entrada do programa e todos os procedimentos necessários para a execução do jogo.

Segmento de Dados (.DATA)

Matrizes do Tabuleiro

- **MATRIZ_1:** Representa o tabuleiro com a posição dos navios. É uma matriz de 10×10 posições, onde:
 - '1' indica a presença de uma parte de navio.
 - '0' indica água (sem navio).
- **MATRIZ_INICIAL:** Matriz de 10×10 posições inicializada com 'X', representando as posições não atacadas pelo jogador. Ao longo do jogo, essa matriz é atualizada com:
 - '1' para acertos (navio atingido).
 - '0' para erros (água).

Mensagens

Diversas mensagens são definidas para interação com o jogador, tais como:

- **MSG_BEM_VINDO:** Mensagem de boas-vindas ao iniciar o jogo.
- **MSG_LINHA e MSG_COLUNA:** Solicitações para que o jogador insira as coordenadas de linha e coluna.
- **ACERTO e ERROU:** Feedback sobre o resultado do ataque.
- **MSG_FIM:** Informada quando todos os navios forem afundados.

Variáveis de Controle

- **CONTADOR_O:** Armazena o número total de partes de navios presentes em **MATRIZ_1**. É decrementado a cada acerto, servindo para verificar quando o jogo deve ser encerrado.

Segmento de Código (.CODE)

Procedimento Principal (MAIN)

Inicialização

1. Configuração do Segmento de Dados:

- Carrega o endereço do segmento de dados em `DS` para acesso às variáveis definidas em `.DATA`.

2. Limpeza da Tela e Apresentação:

- Chama o procedimento `LIMPAR` para limpar a tela.
- Chama o procedimento `APRESENTACAO` para exibir a mensagem de boas-vindas e informações dos autores.

Loop Principal do Jogo

1. Inicialização do Contador de Tentativas:

- O registrador `CX` é configurado com o valor `30`, representando o número máximo de tentativas que o jogador tem para afundar todos os navios.

2. Execução do Loop:

- **COMPARA_MATRIZ:** Chama o procedimento que solicita as coordenadas ao jogador, verifica se são válidas e atualiza o estado do jogo.
- **IMPRIME1:** Atualiza a exibição do tabuleiro com base nos ataques realizados.
- **Verificação do Fim do Jogo:**
 - Se `CONTADOR_0` chegar a zero, significa que todos os navios foram afundados, e o jogo é finalizado.
 - Caso contrário, o loop continua até que o jogador esgote as tentativas.

3. Finalização:

- Ao encerrar o jogo, exibe a mensagem de fim (`MSG_FIM`) e termina a execução do programa.

Procedimentos Auxiliares

1. Procedimento APRESENTACAO

- **Objetivo:** Exibir uma tela inicial com mensagens de boas-vindas e informações sobre os autores do jogo.
- **Funcionamento:**
 - Utiliza a interrupção `INT 10h` para posicionar o cursor em locais específicos da tela.
 - Exibe as mensagens definidas em `.DATA` utilizando a interrupção `INT 21h` com a função `09h`.
 - Aguarda que o usuário pressione qualquer tecla para continuar, utilizando `INT 21h` com a função `01h`.
 - Chama o procedimento `LIMPAR` para limpar a tela antes de iniciar o jogo.

2. Procedimento LIMPAR

- **Objetivo:** Limpar a tela para iniciar o jogo com o tabuleiro vazio.
- **Funcionamento:**
 - Configura o modo de vídeo para `03h` (texto 80×25) utilizando `INT 10h` com a função `00h`.

3. Procedimento IMPRIME1

- **Objetivo:** Exibir o tabuleiro atualizado com as jogadas realizadas pelo jogador.
- **Funcionamento:**
 - Utiliza dois loops aninhados para percorrer as linhas e colunas da matriz.
 - Imprime os números das colunas no topo do tabuleiro para orientação.
 - Para cada posição, verifica o conteúdo de `MATRIZ_INICIAL`:
 - `'X'`: Posição não atacada.
 - `'1'`: Acerto (navio atingido).
 - `'0'`: Erro (água).

- Imprime o caractere correspondente na tela, mantendo a interface atualizada para o jogador.

4. Procedimento COMPARA_MATRIZ

- **Objetivo:** Processar a entrada do jogador, verificar se as coordenadas são válidas e atualizar o estado do jogo.

- **Funcionamento:**

1. Entrada de Coordenadas:

- Solicita ao jogador que insira o número da linha (`MSG_LINHA`) e captura o input.
- Verifica se a entrada é um valor numérico entre `0` e `9`.
- Repete o processo para a coluna (`MSG_COLUNA`).

2. Cálculo do Índice na Matriz:

- Converte as coordenadas de linha e coluna em um índice para acessar `MATRIZ_INICIAL` e `MATRIZ_1`.
- O cálculo é feito multiplicando a linha por `10` e adicionando a coluna.

3. Verificação de Coordenada Repetida:

- Verifica se a posição já foi atacada anteriormente, verificando se o valor em `MATRIZ_INICIAL` não é `'X'`.
- Se a posição já foi escolhida, exibe a mensagem `MSG_REPETIDO` e solicita novas coordenadas.

4. Comparação e Atualização:

- Compara o valor em `MATRIZ_1` na posição escolhida:
 - Se for `'1'`, significa que o jogador acertou um navio.
 - Atualiza `MATRIZ_INICIAL` com `'1'`.
 - Decrementa `CONTADOR_0`.
 - Exibe a mensagem de acerto (`ACERTO`).
 - Se for `'0'`, significa que o jogador errou.
 - Atualiza `MATRIZ_INICIAL` com `'0'`.
 - Exibe a mensagem de erro (`ERROU`).

Lógica do Jogo

Representação do Tabuleiro

- O tabuleiro é representado por matrizes de 10×10 posições.
- As matrizes utilizam índices de 0 a 99, correspondendo às coordenadas de linha e coluna após o cálculo do índice.

Cálculo do Índice na Matriz

- **Fórmula:**

```
mathematica
Copiar código
Índice = (Linha * Número de Colunas) + Coluna
```

- Para um tabuleiro de 10×10:

```
scss
Copiar código
Índice = (Linha * 10) + Coluna
```

- **Exemplo:**

- Linha 3, Coluna 5:

```
scss
Copiar código
Índice = (3 * 10) + 5 = 35
```

Fluxo de Jogo

1. Início:

- O jogador é apresentado ao jogo e inicia com um tabuleiro de posições não reveladas.

2. Jogadas:

- O jogador insere coordenadas de linha e coluna.
- O programa verifica a validade das coordenadas e se já foram escolhidas.
- Compara com `MATRIZ_1` para determinar acerto ou erro.
- Atualiza o tabuleiro e fornece feedback.

3. Terminação:

- O jogo termina quando:
 - O jogador afunda todos os navios (`CONTADOR_0` chega a zero).
 - O jogador esgota o número máximo de tentativas (`CX` chega a zero).

Interações com o Sistema

- **Entrada de Dados:**
 - Utiliza `INT 21h` com a função `01h` para capturar a entrada de um caractere do teclado.
- **Saída de Dados:**
 - Utiliza `INT 21h` com a função `09h` para exibir mensagens na tela.
- **Posicionamento do Cursor:**
 - Utiliza `INT 10h` com a função `02h` para posicionar o cursor em locais específicos durante a apresentação.

Conclusão

Este manual detalhou todo o processo de construção do jogo "Batalha Naval" em Assembly, explicando a lógica por trás de cada parte do código e como as diferentes seções interagem para criar a experiência de jogo. A implementação demonstra o uso eficaz de recursos da linguagem Assembly, como manipulação direta de memória e uso de interrupções para interação com o sistema.

O jogo oferece ao jogador uma interface simples, mas funcional, permitindo a interação e proporcionando uma experiência próxima ao clássico jogo de tabuleiro. O projeto serve como um exemplo prático de programação em baixo

nível, reforçando conceitos importantes em arquitetura de computadores e programação de sistemas.