Documentação do Código

Eduardo Hubner Adriano

08/08/2024

inimigo.cpp

Descrição

Este arquivo contém a implementação da classe inimigo, que inclui métodos para imprimir, apagar e mover navios no tabuleiro do inimigo.

Código

```
#include "../include/inimigo.hpp"
#include "../include/matrizes.hpp"
#include "../include/config.hpp"
#include <cstdlib>
#include <ctime>
#include <conio.h>
#include <iostream>

using namespace std;

/**
 * @file inimigo.cpp
 * @brief Implementação da classe inimigo.
 */

/**
 * @brief Imprime o estado atual do tabuleiro de batalha do inimigo
(MA) na tela.
 *
 * Itera através de uma matriz de 22x22 e imprime cada valor.
 */
void inimigo::printarTabuleiro() {
    for (int i = 0; i < 22; i++) {
        cout << endl;
        for (int j = 0; j < 22; j++) {
            cout << MA[i][j];
        }
    }
}

/**
 * @brief Imprime o estado atual do tabuleiro interno do inimigo (en)
```

```
na tela.
 *
 * Semelhante à função anterior, mas para a matriz en.
 */
void inimigo::printarTabuleiroIN() {
    for (int i = 0; i < 22; i++) {
        cout << endl;
        for (int j = 0; j < 22; j++) {
            cout << en[i][j];
        }
    }
}

/**
 * @brief Apaga o caractere na posição (x, y) do cursor no tabuleiro.
 *
 * Move o cursor para essa posição e imprime um espaço.
 *
 * @param x Coordenada x do cursor.
 * @param y Coordenada y do cursor.
 */
void inimigo::apagar(int &x, int &y) {
    gotoxy(x, y); cout << " ";
}

/**
 * @brief Imprime o caractere '>' na posição (x, y) do cursor no
tabuleiro.
 *
 * Representa a posição atual do cursor.
 *
 * @param x Coordenada x do cursor.
 * @param y Coordenada y do cursor.
 */
void inimigo::printar(int &x, int &y) {
    gotoxy(x, y); cout << ">";
}

/**
 * @brief Move o cursor pelo tabuleiro baseado nas entradas do teclado
(setas de direção).
 *
 * Se uma tecla for pressionada (kbhit()), a função apaga a posição
anterior do cursor
 * e atualiza a nova posição baseada na tecla pressionada.
 *
 * @param x Coordenada x do cursor.
 * @param y Coordenada y do cursor.
 */
void inimigo::mover(int &x, int &y) {
```

```
    if (kbhit()) {
        int d = y - 1;
        if (en[d][x] != 'X' && en[d][x] != 'O') apagar(x, y);
        char tecla = getch();
        if (tecla == ESQUERDA && x > 2) x = x - 2;
        if (tecla == DIREITA && x < 20) x = x + 2;
        if (tecla == CIMA && y > 3) y = y - 2;
        if (tecla == BAIXO && y < 20) y = y + 2;
        d = y - 1;
        if (en[d][x] != 'X' && en[d][x] != 'O') printar(x, y);
    }
}

/**
 * @brief Gera e retorna uma coordenada X aleatória par entre 2 e 20.
 *
 * @param c Parâmetro não utilizado.
 * @param v Parâmetro não utilizado.
 * @return Coordenada X aleatória par entre 2 e 20.
 */
int inimigo::obterX(int c, int v) {
    (void)c;
    (void)v;
    int x;
    do {
        x = rand() % 19 + 2;
    } while (x % 2 != 0);
    return x;
}

/**
 * @brief Gera e retorna uma coordenada Y aleatória par entre 2 e 20.
 *
 * @param c Parâmetro não utilizado.
 * @param v Parâmetro não utilizado.
 * @return Coordenada Y aleatória par entre 2 e 20.
 */
int inimigo::obterY(int c, int v) {
    (void)c;
    (void)v;
    int y;
    do {
        y = rand() % 19 + 2;
    } while (y % 2 != 0);
    return y;
}

/**
 * @brief Retorna uma direção aleatória (0 ou 1).
 *
```

```
 * 0 representa vertical e 1 representa horizontal.
 * @return Direção aleatória.
 */
int inimigo::obterC() {
    return rand() % 2;
}

/**
 * @brief Imprime o porta-aviões (Carrier) no tabuleiro na posição (x,
y) e direção c.
 *
 * @param x Coordenada x do navio.
 * @param y Coordenada y do navio.
 * @param c Direção do navio (0 para vertical, 1 para horizontal).
 */
void inimigo::printarCARRIER(int x, int y, int c) {
    for (int i = 0; i < 4; ++i) {
        if (c == 0) {
            en[y + 2*i][x] = 'C';
        } else {
            en[y][x + 2*i] = 'C';
        }
    }
}

/**
 * @brief Imprime o navio tanque (Tanker) no tabuleiro na posição (x,
y) e direção c.
 *
 * @param x Coordenada x do navio.
 * @param y Coordenada y do navio.
 * @param c Direção do navio (0 para vertical, 1 para horizontal).
 */
void inimigo::printarTANKER(int x, int y, int c) {
    for (int i = 0; i < 3; ++i) {
        if (c == 0) {
            en[y + 2*i][x] = 'T';
        } else {
            en[y][x + 2*i] = 'T';
        }
    }
}

/**
 * @brief Imprime o Destroyer no tabuleiro na posição (x, y) e direção
c.
 *
 * @param x Coordenada x do navio.
 * @param y Coordenada y do navio.
 * @param c Direção do navio (0 para vertical, 1 para horizontal).
```

```
 */
void inimigo::printarDESTROYER(int x, int y, int c) {
    for (int i = 0; i < 3; ++i) {
        if (c == 0) {
            en[y + 2*i][x] = 'D';
        } else {
            en[y][x + 2*i] = 'D';
        }
    }
}

/**
 * @brief Imprime o Submarino no tabuleiro na posição (x, y) e direção
c.
 *
 * O submarino ocupa 2 posições, com a segunda posição dependendo da
direção c.
 * @param x Coordenada x do navio.
 * @param y Coordenada y do navio.
 * @param c Direção do navio.
 */
void inimigo::printarSUBMARINE(int x, int y, int c) {
    en[y][x] = 'S';
}

/**
 * @brief Verifica se a posição é válida para colocar um navio.
 *
 * @param x Coordenada x inicial.
 * @param y Coordenada y inicial.
 * @param c Direção do navio.
 * @param tamanho Comprimento do navio.
 * @param tipo Tipo do navio.
 * @return true se a posição for válida, false caso contrário.
 */
bool inimigo::posicaoValida(int x, int y, int c, int tamanho, char
tipo) {
    if (c == 0) { // Vertical
        if (y + (tamanho - 1) * 2 >= 22) return false; // Fora dos
limites
        for (int i = 0; i < tamanho; ++i) {
            if (en[y + 2 * i][x] != ' ') return false; // Verifica
sobreposição
        }
    } else { // Horizontal
        if (x + (tamanho - 1) * 2 >= 22) return false; // Fora dos
limites
        for (int i = 0; i < tamanho; ++i) {
            if (en[y][x + 2 * i] != ' ') return false; // Verifica
sobreposição
```

```
        }
    }
    return true;
}

/**
 * @brief Imprime todos os navios do inimigo no tabuleiro em posições
válidas, evitando sobreposição.
 *
 * Coloca cada tipo de navio (Carrier, Tanker, Destroyer, Submarine)
em posições aleatórias usando as funções obterX, obterY e obterC.
 */
void inimigo::printarNavios() {
    // Coloca o porta-aviões (Carrier)
    int c1, x1, y1;
    do {
        c1 = obterC();
        x1 = obterX(c1, 4);
        y1 = obterY(c1, 4);
    } while (!posicaoValida(x1, y1, c1, 4, 'C'));
    printarCARRIER(x1, y1, c1);

    // Coloca o navio tanque (Tanker)
    int c2, x2, y2;
    do {
        c2 = obterC();
        x2 = obterX(c2, 3);
        y2 = obterY(c2, 3);
    } while (!posicaoValida(x2, y2, c2, 3, 'T'));
    printarTANKER(x2, y2, c2);

    // Coloca o destruidor (Destroyer)
    int c3, x3, y3;
    do {
        c3 = obterC();
        x3 = obterX(c3, 3);
        y3 = obterY(c3, 3);
    } while (!posicaoValida(x3, y3, c3, 3, 'D'));
    printarDESTROYER(x3, y3, c3);

    // Coloca o submarino (Submarine)
    int c4, x4, y4;
    do {
        c4 = obterC();
        x4 = obterX(c4, 2);
        y4 = obterY(c4, 2);
    } while (!posicaoValida(x4, y4, c4, 2, 'S'));
    printarSUBMARINE(x4, y4, c4);
}
```

```
/**
 * @brief Registra um disparo no tabuleiro do inimigo na posição (x,
y).
 *
 * Se um navio for atingido, a posição é marcada como 'O' e a vida do
inimigo (vidaIN) é incrementada.
 * Se o disparo não atingir um navio, a posição é marcada como 'X'.
 *
 * @param x Coordenada x do disparo.
 * @param y Coordenada y do disparo.
 */
void inimigo::disparo(int x, int y) {
    y = y - 1;
    if (en[y][x] == 'C' || en[y][x] == 'T' || en[y][x] == 'D' || en[y]
[x] == 'S' || en[y][x] == 'O') {
        en[y][x] = 'O';
        y = y + 1;
        gotoxy(x, y); std::cout << "O";
        vidaIN = vidaIN + 1;
    } else {
        en[y][x] = 'X';
        y = y + 1;
        gotoxy(x, y); std::cout << "X";
    }
}
```