Trabalho Prático de
Algoritmos e Técnicas de Programação

Pontifícia Universidade Católica de Minas Gerais

Instituto de Ciências Exatas e Informática

Curso de Sistemas de Informação - Noite

Algoritmos e Técnicas de Programação

Alunos: **Samuel Leite Diniz & Miguel Oliveira Bizzi**

Objetivo do programa

O nosso programa tem como objetivo desenvolver e implementar uma partida de Campo Minado, sendo possível ser jogada pelo usuário em c#. Utilizando de arquivos de texto para obter informações do tabuleiro.

Detalhes da implementação

1. Decisões de implementação:

Ao iniciar o projeto, decidimos por aplicar o conhecimento em classes para facilitar a indentação e produção do código. Decidimos por criar duas classes, uma chamada ArquivoTexto e outra chamada Tabuleiro, que auxiliariam nosso Main. Pensamos, a partir disso, receber as informações do tabuleiro (linhas, colunas e bombas) do usuário, após validá-las criaremos um arquivo de texto com as informações e leremos ele para criar o tabuleiro, distribuir as bombas e números.

Decidimos criar dois tabuleiros (matrizes de char) na nossa classe Tabuleiro, um que seria revelado para o usuário (para ele efetuar as jogadas) e um que estaria com todas as bombas e números já distribuídos, para nós consultarmos ele para validar as jogadas, abrir posições e checar se uma bomba foi atingida.

Usamos uma estrutura de repetição que sempre irá pedir para o usuário efetuar uma nova jogada até que o jogo termine, seja em derrota ou vitória.

2. Classes desenvolvidas:

=> Classe ArquivoTexto: essa classe é responsável por criar nosso arquivo texto chamado "info_tabuleiro.txt" e nele, escrever os dados que precisamos para dar início a criação do tabuleiro, sendo eles o número de linhas, o número de colunas e o número de bombas que o usuário deseja.

a. Métodos desenvolvidos na Classe ArquivoTexto:

Método CriarArquivoTexto(): responsável por receber as informações do usuário de quantas linhas, colunas e bombas ele gostaria, criar um arquivo texto de nome "info_tabuleiro.txt" no diretório do projeto e armazenar nesse arquivo essas mesmas informações anteriormente digitadas pelo usuário. Tem como parâmetros as variáveis int linhas, int colunas, int bombas, que são digitadas pelo usuário no Main. Não possui retorno.

Método LerArquivoTexto(): responsável por ler nosso arquivo texto anteriormente criado pelo método acima. É percorrido todo arquivo e lido linha por linha, identificado qual informação possui em cada linha e separando o valor que buscamos da frase escrita em cada linha. Exemplo:
Se a frase for "Número de linhas: 3", o método lê a frase e separa ela a partir do ":", criando assim dois índices. O índice [0] armazena a primeira parte da frase "Número de linhas" e o índice [1] armazena a segunda parte "3", que é a informação que procuramos (sabemos que a informação que precisamos encontra-se no final da frase, já que criamos o arquivo texto desse modo). Não possui parâmetros. Tem como retorno as variáveis num_linhas, num_colunas, num_bombas, que armazenam os valores lidos no arquivo texto.

=> Classe Tabuleiro: essa classe é responsável por criar nosso tabuleiro a partir do número de linhas, colunas e bombas lidas no arquivo texto. É distribuído pelo tabuleiro um número aleatório de bombas e, logo após, são distribuídos os números de dentro de cada casa, de acordo com quantas bombas existirem nas casas adjacentes. Também é responsável por exibir o tabuleiro ao jogador.
Essa classe também permite ao jogador fazer suas jogadas, liberando as posições no tabuleiro e verificando a cada jogada se o jogador venceu ou perdeu a partida.

a. Métodos desenvolvidos na Classe Tabuleiro:

Método CriaTabuleiro(): tem a função de criar o tabuleiro original do jogo (revelado) e o que mostraremos ao jogador, os dois de acordo com as linhas e colunas lidas no arquivo texto. Percorre a matriz e adiciona um x em todas as posições para dar vida ao tabuleiro. Possui os parâmetros int linhas, int colunas, que são o número de linhas e colunas digitadas pelo usuário e lida pelo arquivo texto. Não possui retorno.

Método DistribuirBombas(): tem a função de distribuir aleatoriamente o número de bombas, digitadas pelo usuário e lida pelo arquivo texto, pelo tabuleiro. Percorre a matriz e adiciona um b, nos lugares sorteados para ser as bombas. Possui o parâmetro int num_bombas, que são o número de bombas anteriormente digitados. Não possui retorno.

Método DistribuirNumeros(): tem a função de distribuir os números pelo tabuleiro. Percorre toda a matriz, passando por cada índice individualmente, e, a cada bomba que existir nas casas adjacentes, aumenta em 1 o valor da posição. É necessário checar se todas as casas adjacentes existem (estão no limite do tabuleiro). Não possui parâmetros nem retorno.

Método LiberarPosicao(): é o método booleano utilizado para realizar a jogada do usuário. Tem a função de realizar o processo de liberação da posição escolhida e é responsável por verificar se a casa já está aberta e se há bomba (caso houver, encerrar a aplicação com a derrota do usuário). Além disso, chama o método CalcularPosicoesAdjacentes para abrir as outras posições caso o usuário tenha revelado uma posição vazia. Possui os parâmetros int linha, int coluna, que são os valores escolhidos pelo usuário ao fazer a jogada. Primeiramente retorna false, retorna false se a posição já foi revelada e retorna true se revelar uma bomba. Isso será usado posteriormente no Main para verificar se o jogo acabou.

Método CalcularPosicoesAdjacentes(): Método responsável por abrir as outras posições de acordo com a posição selecionada do tabuleiro. Primeiro, é verificado se a posição está fora dos limites do tabuleiro ou se a célula já foi revelada. Caso a posição seja 0, verificar em todas as posições adjacentes se há mais algum 0 ou número que possa ser liberado. Caso sim, a posição é aberta e o cálculo realizado novamente. Possui os parâmetros int linha, int coluna, que são os valores digitados na jogada do usuário. Não possui retorno.

Método VerificarVitoria(): é responsável por verificar se o usuário ganhou. Verificar se todo o tabuleiro foi revelado sem ter explodido alguma bomba. Percorre o tabuleiro e verifica quantas posições não abertas ainda existem e armazena o valor no contador. Se o número de posições não abertas restantes (contador) for igual ao número de bombas do tabuleiro, o usuário conseguiu revelar todos os campos numerais sem ter revelado nenhuma das bombas, então é declarada a vitória ao usuário. Não possui parâmetros. Primeiramente retorna false e retorna true se o contador for igual ao número de bombas, ou seja, se o usuário vencer. Isso será usado posteriormente no Main para verificar se o jogo acabou.

Método MostrarTabuleiroRevelado(): Método criado para exibir o tabuleiro toda vez que o usuário for efetuar uma jogada, exibindo o tabuleiro de acordo com as posições liberadas pelo mesmo. Percorre a matriz e mostra o tabuleiro. É chamado toda vez que o usuário faz uma jogada, para exibir o novo tabuleiro após a liberação da posição escolhida. Não possui parâmetros. Não possui retorno.

Método MostrarTabuleiroGameOver(): Método criado exclusivamente para exibir o tabuleiro no momento de GameOver, exibindo uma mensagem de Fim de jogo e o Tabuleiro com todas as bombas, números e espaços revelados. É chamado caso o usuário vença ou perca a partida. Não possui parâmetros. Não possui retorno

3. Como executar o programa:

Primeiro, pedimos ao jogador digitar o número de linhas, colunas e bombas do tabuleiro desejado, respectivamente. Após validar as informações (caso não seja válida encerramos a aplicação), criamos um tabuleiro desejado a partir da criação e leitura de um arquivo de texto, distribuímos as bombas e os números pelo tabuleiro.

E então iniciará o jogo, o jogador poderá fazer sua jogada até que o jogo termine em vitória quando não sobrar nenhum espaço não revelado que não seja bombas ou que ele atinja uma bomba, resultando em derrota.

4. Testes realizado:
    a. Fizemos dois tabuleiros, um que já está todo revelado e um sem posições reveladas para o usuário. Utilizamos os dois tabuleiros exibidos simultaneamente a cada jogada para checarmos se o jogo estava funcionando corretamente. Checamos se a criação do tabuleiro foi feita corretamente, se abertura de células estava sendo executada sem erros. Conferimos também se o usuário perdia se encontrasse uma bomba e se ganhava caso todo o tabuleiro fosse revelado sem ter explodido alguma bomba.
    b. Realizamos testes a cada jogada do usuário, verificando se ele estava digitando valores que correspondem aos necessários para jogar o jogo. Valores iguais ou maiores que zero e valores menores que o tamanho das linhas e colunas, estando assim, dentro dos limites do tabuleiro.
    c. Nos certificamos, que o arquivo de texto estava sendo criado corretamente de acordo com as informações obtidas do usuário. Também garantimos que as informações lidas do arquivo de texto estejam corretas. Utilizamos try-catch (estruturas de tratamento) juntamente com estruturas de condição para alcançar tal objetivo.