Universidade Paulista UNIP

Instituto de Ciências Exatas e Tecnologia – ICET

Graduação em Ciências da Computação

Atividades Práticas Supervisionadas – APS

Campo Minado

Professor Responsável: Fernando Mariani

Heldeer Gomes de Oliveira – RA: B193JC-2

Henrique Ribeiro dos Santos Rocha – RA: B27CCI-5

Leandro do Nascimento Piroupo Francisco – RA: B44709-5

Ricardo Heronides Nogueira Filho – RA: B2504E2

Vitor Henrique Silva e Silva – RA: B42HH-7

São Paulo

2013

Sumário

[Objetivo 2](#_Toc371255580)

[Introdução 3](#_Toc371255581)

[Regras e Funcionamento do Jogo 4](#_Toc371255582)

[Regras 4](#_Toc371255583)

[Análise do jogo 4](#_Toc371255584)

[Padrões de problemas e solução 4](#_Toc371255585)

[Análise de um quadrado 4](#_Toc371255586)

[Análise de dois quadrados 4](#_Toc371255587)

[Análise de minas compartilhadas 5](#_Toc371255588)

[Análise final 5](#_Toc371255589)

[Adivinhação às vezes necessária 5](#_Toc371255590)

[NP-completude 5](#_Toc371255591)

[P Versus NP 5](#_Toc371255592)

[Definição do problema 5](#_Toc371255593)

[Formulando o problema do caixeiro viajante 6](#_Toc371255594)

[Exemplificando o caso n= 4 6](#_Toc371255595)

[Complexidade computacional do problema 6](#_Toc371255596)

[Plano de desenvolvimento do Jogo 7](#_Toc371255597)

[Projeto do programa 7](#_Toc371255598)

[Código Fonte 7](#_Toc371255599)

[Bibliografia 8](#_Toc371255600)

# Objetivo

O objetivo deste trabalho é explicitar o processo de construção de um jogo com interface gráfica, utilizando as tecnologias existentes na linguagem de programação JAVA.

Desta forma o grupo escolheu o jogo “Campo Minado” para demonstrar o que foi aprendemos em sala de aula e através de pesquisas, utilizando a linguagem JAVA.

# Introdução

O campo minado é um dos jogos mais populares dos computadores. Todas as pessoas, ao sentar na frente de um PC, com certeza, mesmo que só por curiosidade, já abriu o jogo do campo minado e jogou. A grande maioria desiste do jogo, pois é um jogo que envolve cálculos e dedução, onde somente se você prestar atenção, você conseguirá ganhar no jogo. Mas como joga? Como são feitos os cálculos? Existe empate? Ao longo deste documento serão explicadas as regras do jogo, métodos de dedução, cálculos feitos e claro, as tecnologias usadas.

# Regras e Funcionamento do Jogo

## Regras

A área de jogo consiste num campo de quadrados retangular. Cada quadrado pode ser revelado clicando sobre ele, e se o quadrado clicado contiver uma mina, então o jogo acaba. Se, por outro lado, o quadrado não contiver uma mina, uma de duas coisas poderá acontecer:

Um número aparece, indicando a quantidade de quadrados adjacentes que contêm minas;

Nenhum número aparece. Neste caso, o jogo revela automaticamente os quadrados que se encontram adjacentes ao quadrado vazio, já que não podem conter minas;

O jogo é ganho quando todos os quadrados que não têm minas são revelados.

Opcionalmente, o jogador pode marcar qualquer quadrado que acredita que contém uma mina com uma bandeira, bastando para isso clicar sobre ele com o botão direito do mouse. Em alguns casos, carregar com ambos os botões do mouse num número que contenha tantas bandeiras imediatamente à sua volta quanto o valor desse número revela todos os quadrados sem bombas que se encontrem adjacentes a ele. Em contrapartida, o jogo acaba se efetuar essa ação quando os quadrados errados estiverem marcados com as bandeiras.

Algumas versões do campo minado ajudam o jogador, na medida em que nunca colocam uma mina no primeiro quadrado clicado, como a maioria dos programas, ou sempre deixam uma abertura no primeiro clique, como ocorre na versão do Windows Vista.

## Análise do jogo

### Padrões de problemas e solução

Durante o andamento do jogo, existem diversos padrões de quadrados numerados que permitem somente uma determinada configuração de minas. A fim de terminar o jogo tão cedo quanto possível, É preferível processar tais padrões primeiramente, e continuar analisando padrões mais complexos posteriormente. Existem diversos métodos para resolver problemas do jogo sem contar com o uso de adivinhação.

### Análise de um quadrado

Quando o número de quadrados não descobertos ao redor de um quadrado numerado é igual ao número sendo mostrado, todos os quadrados adjacentes são minas. Em contrapartida, quando o número de quadrados com minas descobertas ao redor de um quadrado numerado é igual ao número sendo mostrado, quaisquer outro quadrado adjacente não possui mina, está seguro.

### Análise de dois quadrados

Com dois números de um campo minado, denominados x e y, existem três áreas distintas no campo: a) minas ao redor de tanto x e y, b) minas ao redor de x somente e c) minas ao redor de y somente. Esse método de análise funciona melhor quando os quadrados relativos a x e y são adjacentes, mas também pode ser usado em outras situações. Sabe-se que o número de minas exclusivas a x menos o número de minas exclusivas a y é igual a x-y, o que pode ser usado para marcar bandeiras em minas ou descobrir quadrados seguros.

### Análise de minas compartilhadas

Suponha que o campo possui um número "1" descoberto. De alguma forma descobre-se que dois quadrados adjacentes ao "1" compartilham uma mina. Isso significa que todos os outros quadrados adjacentes ao "1" são seguros, exceto pelos dois que compartilham a mina.

### Análise final

Usada no final do jogo, ela pode ser usada para descobrir um quadrado seguro quando todos os outros do campo são ou seguros ou marcados como minas. Geralmente, tais quadrados finais estão localizados nas paredes do campo. Em algumas versões do jogo, a quantidade de minas presentes no campo é conhecida. Perto do final do jogo quase todos os quadrados já foram descobertos, e saber a quantidade de minas restantes (ainda não descobertas) pode ser útil para resolver o padrão final do jogo.

### Adivinhação às vezes necessária

O jogador deve adivinhar em qual dos quadrados marcados com o ponto de interrogação está localizada a mina. Outro caso que demonstra a necessidade de adivinhação é aquele onde um quadrado não descoberto é completamente rodeado por minas com número "1", ou quando há uma combinação de minas e o perímetro do campo de jogo. Nesse caso, já que nenhum número rodeia o quadrado não descoberto, o jogador não possui informação sobre o conteúdo do quadrado. Entretanto, há uma estratégia para remediar a situação que evita a adivinhação: continuar jogando e ignorar o quadrado que requer adivinhação.

### NP-completude

Em 2000, Kaye publicou a prova de que o jogo é NP-completo para determinar se uma posição no campo minado é consistente com a marcação de minas. Atualmente, o campo minado é mencionado pelo Clay Mathematics Institute na descrição não oficial do problema P versus NP.

## Cálculo P Versus NP

O problema "P versus NP" é o principal problema aberto da Ciência da Computação. Possui também enorme relevância em campos que vão desde a Engenharia até a criptografia aplicada aos serviços militares e às transações comerciais e financeiras via Internet.

Definição do problema

De modo simplificado, o problema pergunta se existem problemas matemáticos cuja resposta pode ser verificada em tempo polinomial, que não possam ser resolvidos (diretamente, sem se ter um candidato à solução) em tempo polinomial. Ilustrando: se alguém lhe disser que o número 13.717.421 pode ser escrito como o produto de dois outros inteiros, você provavelmente demorará para provar isso; contudo, se lhe assoprarem que ele é o produto de 3.607 por 3.803, você seria capaz de muito rapidamente verificar tal fato.

O problema "P versus NP" parte da constatação que são muito frequentes as situações em que parece ser muito mais rápido verificar solução do que achar um processo de resolução, e então pergunta: isso sempre ocorre, ou simplesmente ainda não descobrimos um modo de resolvê-los rapidamente?

Para uma discussão mais desenvolvida, mas ainda evitando tecnicalidades, sobre o Problema "P versus NP", veja outro problema clássico: o problema do caixeiro viajante.

Não se iluda com a aparência de brincadeira deste problema. Ele não é uma curiosidade inconsequente. Em verdade, ele é um problema que deve fazer parte da bagagem de todo profissional competente da área matemática. Sua importância resume-se em duas capacidades:

de modo simples e concreto, exemplifica a enorme velocidade de crescimento da fatorial;

prova-se que muitos problemas combinatórios envolvem tantas alternativas de solução quanto este problema, de modo que ele é uma espécie de "metro" com o qual medimos a complexidade computacional dos problemas combinatórios ocorrendo em engenharia e no trabalho científico.

### Formulando o problema do caixeiro viajante

Suponha que um caixeiro viajante tenha de visitar n cidades diferentes, iniciando e encerrando sua viagem na primeira cidade. Suponha, também, que não importa a ordem com que as cidades são visitadas e que de cada uma delas pode-se ir diretamente a qualquer outra. O problema do caixeiro viajante consiste em descobrir a rota que torna mínima a viagem total.

Exemplificando o caso n= 4

Se tivermos quatro cidades A, B, C e D, uma rota que o caixeiro deve considerar poderia ser: saia de A e daí vá para B, dessa vá para C, e daí vá para D e então volte a A. Quais são as outras possibilidades? É muito fácil ver que existem seis rotas possíveis:

ABCDA

ABDCA

ACBDA

ACDBA

ADBCA

ADCBA

### Complexidade computacional do problema

O problema do caixeiro é um clássico exemplo de problema de otimização combinatória. A primeira coisa que podemos pensar para resolver esse tipo de problema é reduzi-lo a um problema de enumeração: achamos todas as rotas possíveis e, usando um computador, calculamos o comprimento de cada uma delas e então vemos qual a menor. É claro que se acharmos todas as rotas estaremos contando-as, daí podermos dizer que estamos reduzindo o problema de otimização a um de enumeração.

Para acharmos o número R(n) de rotas para o caso de n cidades, basta fazer um raciocínio combinatório simples e clássico. Por exemplo, no caso de n =4 cidades, a primeira e última posição são fixas, de modo que elas não afetam o cálculo; na segunda posição podemos colocar qualquer uma das 3 cidades restantes B, C e D, e uma vez escolhida uma delas, podemos colocar qualquer uma das 2 restantes na terceira posição; na quarta posição não teríamos nenhuma escolha, pois sobrou apenas uma cidade; consequentemente, o número de rotas é 3 × 2 × 1= 6, resultado que tinhamos obtido antes contando diretamente a lista de rotas acima.

De modo semelhante, para o caso de n cidades, como a primeira é fixa, o leitor não terá nenhuma dificuldade em ver que o número total de escolhas que podemos fazer é (n-1) × (n-2) × ... × 2 × 1. De modo que, usando a notação de fatorial: R(n) = (n - 1)!.

Assim que nossa estratégia reducionista consiste em gerar cada uma dessas R(n) = (n - 1)! rotas, calcular o comprimento total das viagens de cada rota e ver qual delas tem o menor comprimento total (esse método é chamado no jargão matemático de força bruta e ignorância1 ). Trabalho fácil para o computador, diria alguém. Bem, talvez não. Vejamos o porquê.

Suponhamos que temos um computador muito veloz, capaz de fazer 1 bilhão de adições por segundo. Isso parece uma velocidade imensa, capaz de tudo. Com efeito, no caso de 20 cidades, o computador precisa apenas de 19 adições para dizer qual o comprimento de uma rota e então será capaz de calcular 109 / 19 =53 milhões de rotas por segundo. Contudo, essa imensa velocidade é um nada frente à imensidão do número 19! de rotas que precisará examinar. Com efeito, o valor de 19! é 121.645.100.408.832.000 (ou , aproximadamente, 1,2 × 1017 em notação científica). Consequentemente, ele precisará de 1,2 × 1017 / (53 × 106)= 2,3 × 109 segundos para completar sua tarefa, o que equivale a cerca de 73 anos. O problema é que a quantidade (n - 1)! cresce com uma velocidade alarmante, sendo que muito rapidamente o computador torna-se incapaz de executar o que lhe pedimos. Constate isso mais claramente na tabela a seguir:

Observe que o aumento no valor do n provoca uma grande diminuição na velocidade com que o computador calcula o tempo de cada rota (ela diminui apenas de um sexto ao n aumentar de 5 para 25), mas provoca um imensamente grande aumento no tempo total de cálculo. Em outras palavras: a inviabilidade computacional é devida à presença da fatorial na medida do esforço computacional do método da redução. Com efeito, se essa complexidade fosse expressa em termos de um polinómio em n o nosso computador seria perfeitamente capaz de suportar o aumento do n. Confira isso na seguinte tabela que corresponde a um esforço computacional polinomial R(n) = n5:

Então o método reducionista não é prático (a não ser para o caso de muito poucas cidades), mas será que não se pode inventar algum método prático (por exemplo, envolvendo esforço polinomial na variável número de) para resolver o problema do caixeiro viajante? Bem, apesar de inúmeros esforços, ainda não foi achado tal método e começa-se a achar que o mesmo não existe.

# Plano de desenvolvimento do Jogo

Nesta seção do documento, serão apresentados os componentes utilizados para a criação da interface, eventos e imagens do jogo.

## Componentes

### JFrame

JFrame nada mais é que uma classe, como outra qualquer que estudamos e criamos ao longo de nosso curso.

Porém, essa classe é que será responsável por criar a tela em que iremos desenhar, colocar botões, menus, caixas de texto e tudo mais que existem nas janelas de aplicativos.

Para fazer uso do JFrame, temos que importar essa classe do pacote swing, que contém diversas funcionalidades para programação gráfica:

import javax.swing.JFrame;

Cada objeto que criamos é um frame, ou janela, diferente. Uma aplicação normal tem dezenas ou centenas de frames.

Nessa seção vamos criar alguns objetos da classe JFrame passando uma String argumento para o construtor padrão, essa string será o título de nossa aplicação.

Esses JFrames, porém, possuem dezenas de funcionalidades e opções, usaremos as funcionalidades: o que ocorre quando clicamos no 'x' de close, adicionar panels, definir o tamanho do frame e se ele será visível ou não.

### JPanel

A rigor, JPanel é uma classe que é um container. Podemos ver esse container, ou JPanel, como um painel.

Embora possa ocupar a janela (frame ou moldura) inteira, se usa JPanel mais como um painel com alguns elementos.

Por exemplo, a tela inicial do MSN Messenger seria o frame, os painéis seriam a lista de contatos, aquele bloco embaixo com propaganda, tem o panel com os menus em cima e outro panel na esquerda com aquelas opções de chat, redes sociais etc.

Ou seja, JPanel são blocos dentro da frame.

São muito bons na organização das janelas que você criar.

Crie um painel pros botões, pros menus, pras imagens, outro panel para as caixas de texto, etc etc.

Para usar o JPanel, importamos essa classe da package swing:

import javax.swing.JPanel;

### JButton

O JButton, também é uma classe que é container. Como o próprio nome nos indica é um botão.

Os botões possuem diversas funções, que são definidas de acordo com os eventos definidos pelo programador.

A classe JButton também faz parte da package swing:

import javax.swing.JButton

### JMenu

A classe JMenu é utilizada para a criação dos Menus de opções alocados na JMenuBar, são os famosos botões de Arquivo, Editar, Salvar, Sair. Que temos na maioria das janelas do programas em que utilizamos.

A classe JMenu faz parte da package swing:

import javax.swing.Jmenu

### JmenuBar

A classe JmenuBar, é a tão comum barra de menu presente na maioria dos programas e janelas que utilizamos diariamente, em alguns casos fica oculta e só aparece quando pressionamos a tecla “Alt”.

A classe JmenuBar faz parte da package swing:

import javax.swing.JmenuBar

### ComponenteBotao

Componente criado pelo grupo para atender as necessidades do jogo. O componente é uma extensão da classe JButton, com os demais atributos necessários, sendo eles:

Mina: Atributo booleano que identifica se o botão clicado pelo jogador possui uma bomba ou não.

Linha: Atributo do tipo inteiro que indica qual a linha que o componente está no quadro de jogo.

Coluna: Atributo do tipo inteiro que indica qual a coluna que o componente está no quadro de jogo.

Posicao: Atributo inteiro que guarda o par ordenado da posição completa do botão.

Clicado: Atributo booleano que identifica se o botão foi clicado pelo usuário ou não.

# Projeto do programa

Explicar partes do programa

# Código Fonte

Pacote br.unip.com

# Bibliografia

<http://nothings.org/games/minesweeper/>

<http://www1.folha.uol.com.br/ciencia/782833-solucao-para-maior-problema-da-computacao-esta-provavelmente-errada.shtml>

<http://www.win.tue.nl/~gwoegi/P-versus-NP.htm>

<http://docs.oracle.com/javase/6/docs/api/javax/swing/JFrame.html>

<http://docs.oracle.com/javase/1.4.2/docs/api/javax/swing/JPanel.html>