Universidade Federal de São João Del-Rei

Ciência da Computação – 1º período

AEDS I – TP2

Jogo da velha
Batalha Naval
Imobiliária Xulambs

Professor – Leonardo Rocha

Isabella Vieira Ferreira

(Matrícula: 112050060)

Mônica Neli de Resende

(Matrícula: 112050045)

01 de julho de 2011

Jogo da Velha

Índice

Introdução..... 5
1. Regras do jogo..... 6
2. Problema proposto..... 6
3. Descrições da solução.....6
4. Listagem de rotinas.....7
5. Testes.....8
Conclusão.....17
Referências Bibliográficas.....18
Definição da pontuação.....18

Batalha Naval

Índice

Introdução..... 5
1. Regras do jogo..... 9
2. Problema proposto..... 9
3. Descrições da solução.....9
4. Listagem de rotinas.....11
5. Testes.....11
Conclusão.....17
Referências Bibliográficas.....18
Definição da pontuação.....18

Imobiliária Xulambs

Índice

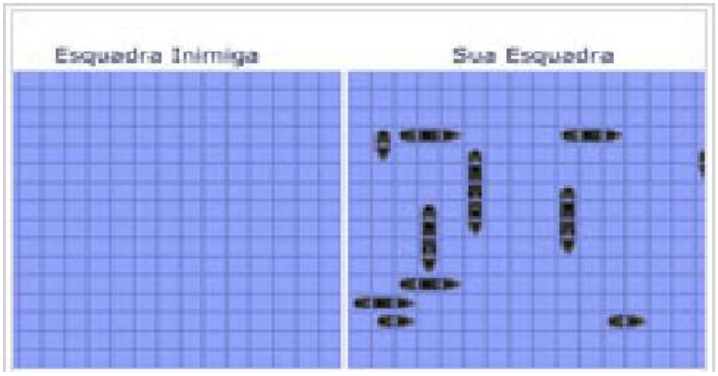
Introdução..... 5
1. Problema proposto.....12
2. Descrições da solução.....12
3. Listagem de rotinas.....13
4. Testes.....16
Conclusão.....17
Referências Bibliográficas.....18
Definição da pontuação.....18

Introdução

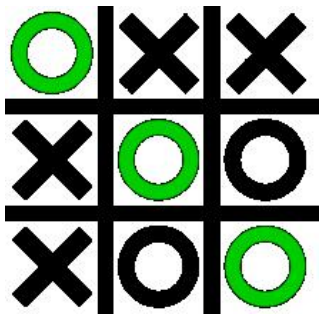
O nome jogo da velha teria se originado na Inglaterra, quando nos finais da tarde, mulheres se reuniram para conversar e bordar. As mulheres idosas, por não terem mais condições de bordar em razão da fraqueza de suas vistas, jogavam este jogo simples, que passou a ser conhecido como o da "velha". Muito popular por sua disponibilidade, pode ser jogado sobre um tabuleiro, no computador ou mesmo sendo riscado sobre um pedaço de papel ou mesa.

Batalha naval é um jogo de tabuleiro de dois jogadores, no qual os jogadores têm de adivinhar em que quadrados estão os navios do oponente. Embora o primeiro jogo em tabuleiro comercializado e publicado pela Milton Bradley Company em 1931, o jogo foi originalmente jogado com lápis e papel.

A Imobiliária Xulambs está entre as maiores no ramo de administração de imóveis na cidade de Xulambs, com aluguel de apartamentos, casas, salas comerciais, kitnets, entre outros. A Imobiliária Xulambs se destaca pelo seu método de trabalho e sua política fazendo com que a satisfação de nossos clientes esteja sempre em primeiro lugar.



Demonstração de um jogo Batalha Naval



Exemplo de um jogo da velha em que o círculo (O) ganha o jogo.

Jogo da Velha

1. Regras do jogo

O tabuleiro é uma matriz de três linhas por três colunas.

Os jogadores jogam alternadamente, uma marcação por vez, numa lacuna que esteja vazia.

O objetivo é conseguir três círculos ou três xis em linha, quer horizontal, vertical ou diagonal, e ao mesmo tempo, quando possível, impedir o adversário de ganhar na próxima jogada.

2. Problema proposto

O desafio proposto é desenvolver um programa que sempre deverá verificar:

- se a posição fornecida pelos jogadores é válida e/ou já está marcada.
- a cada jogada se o jogo terminou, obedecendo as regras do jogo da velha.

3. Descrições da Solução

Primeiramente é exposto aos jogadores uma matriz 3 x 3, onde a cada jogada,o jogador deverá informar o número da opção desejada de acordo com as especificações na tela. É atribuído automaticamente, ‘X’ ao primeiro jogador, e ‘O’ ao segundo jogador.

3.1 Listagens de rotinas

- **procedure valor_valido (jog_atual, jog1, jog2: string);**

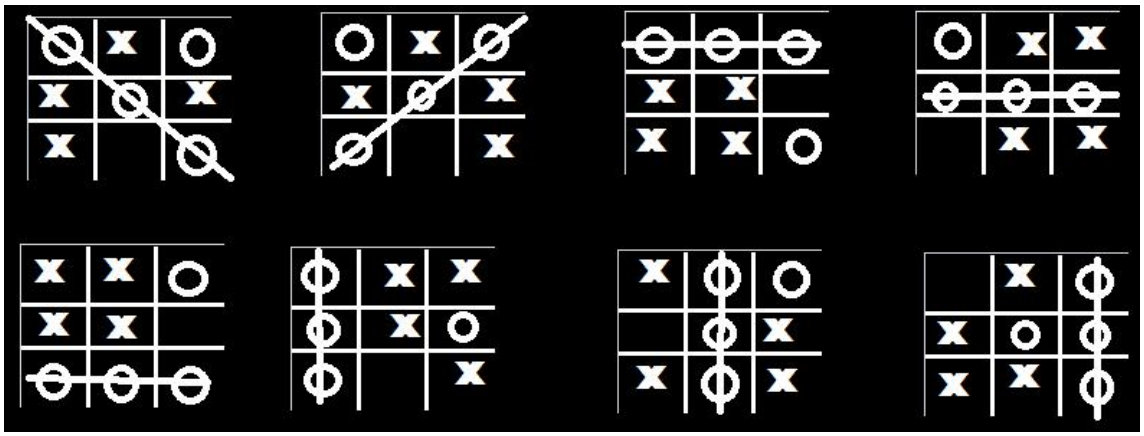
É um procedimento que solicita ao usuário que digite uma nova posição na matriz, caso essa posição que já tenha sido marcada em outra jogada.

Este procedimento possui dois laços aninhados e uma *flag* para verificar se entrou nos laços ou não. Após o usuário digitar um valor válido é atribuído à matriz o símbolo correspondente ao jogador 1 ou ao jogador 2.

- **procedure verifica_ganhou;**

É um procedimento que verifica se a 'combinação' de 'X' ou 'O' é válida ou não.

O jogador ganhará quando for satisfeita uma das combinações ilustradas abaixo:



Se nenhuma das condições acima for satisfeitas, ninguém ganhou.

Este procedimento possui estruturas de comparação e uma *flag* para determinar se a condição foi satisfeita ou não.

- **Programa principal**

No programa principal, atribuímos na matriz valores de 1 a 9 para que o usuário possa escolher sua opção na hora de jogar. A alternância de jogadores é feita através de uma variável contadora, onde, quando a contadora for igual a 1, se refere a vez do primeiro jogador e quando a contadora for igual a 2, se refere a vez do segundo jogador.

Se houver probabilidade de ninguém ganhar o jogo, ou seja, se nenhuma das condições ilustradas acima forem satisfeitas, é necessário que se jogue 9 rodadas até completar a matriz, caso contrário, o jogo emitirá uma mensagem informando quem é o ganhador.

4. Testes

O código foi gerado pelo compilador Geany 0.20.

Os testes foram realizados em uma máquina com a seguinte descrição:

Hardware/Software	Modelo
Sistema operacional:	Windows 7 Ultimate 32 bits e Ubuntu 10.04
Processador:	Intel Core 2 Duo CPU T6400 2.00 GHz
RAM:	3,00 GB, DDR2, 667MHz

Batalha Naval

1. Regras do Jogo

Cada jogador posiciona as suas embarcações no tabuleiro, na horizontal ou vertical, de modo que uma não toque a outra. Sem que o adversário saiba onde foi colocado.

Os jogadores “atiram” no tabuleiro do adversário, escolhendo a linha e a coluna, visando atingir as embarcações ocultas no tabuleiro. Segue-se alternando os jogadores até que algum jogador atinja todas as embarcações do adversário, ou seja, vence quem terminar primeiro.

2. Problema Proposto

O problema consiste em implementar um jogo de batalha naval de acordo com as regras acima contendo:

- Posicionamento das embarcações;
- Ataque no tabuleiro adversário;
- Informar qual embarcação foi atingida ou se nenhuma embarcação foi atingida;

3. Descrição da solução

Ao iniciar o programa é apresentada aos jogadores uma tela de bem vindo informando como jogar. Em seguida é feito o cadastro dos jogadores e o posicionamento dos navios na matriz (tabuleiro), atribuindo ‘1’ nas posições onde se encontra as embarcações. Após esses passos

começa-se o jogo, o primeiro jogador digita a posição onde deseja atacar, se acertar alguma embarcação é atribuído ‘-1’ à posição digitada e o jogador continua jogando até que erre, caso contrário passa-se para o segundo jogador. A posição é composta pelo número da linha e número da coluna.

4. Listagem das rotinas

- **Function verifica_esquerda (x, y: integer; m: matriz; n: integer): boolean;**
- **Function verifica_direita (x, y: integer; m: matriz; n: integer): boolean;**

No posicionamento das embarcações o usuário tem a opção de escolher entre horizontal e vertical, as funções acima referem-se à opção de horizontal. O usuário digita a posição de onde quer posicionar a embarcação então verfica-se à direita e à esquerda da posição retornando *true* se puder ser colocado e *false* caso contrário. Se for possível colocar tanto à direita quanto à esquerda então é apresentado ao usuário a possibilidade de escolha.

- **Function verifica_acima (x, y: integer; m: matriz; n: integer): boolean;**
- **Function verifica_abaixo (x, y: integer; m: matriz; n: integer): boolean;**

Caso o usuário escolher a opção vertical, as funções acima verificam a possibilidade de colocar para cima ou para baixo, retornando *true* se puder ser colocado e *false* caso contrário. Se for possível colocar tanto para cima quanto para baixo então o usuário faz a sua escolha.

- **Function embarcacoes (x, y : integer; m: matriz) : string ;**

É uma função responsável por retornar qual embarcação foi atingida. Ao atingir uma embarcação a função verifica se à direita ou à

esquerda há alguma parte da embarcação, atingida ou não. Em caso afirmativo contam-se as partes pela direita e pela esquerda e soma-se com a posição acertada, a partir desse número retorna o nome da embarcação. De modo análogo se tiver partes para cima ou para baixo.

- **Procedure jogadas (var m1, m2: matriz; var jog1, jog2: string);**

Procedimento responsável pelos ataques, alternância entre os jogadores e as matrizes, verificando se acertou alguma embarcação, se atirou na água ou se já atacou na posição digitada, até que alguém vença.

- **Function verifica_ganhador (m1: matriz): boolean;**

Esta função verifica se ainda resta alguma embarcação na matriz, retornando *true* se não tiver mais nenhuma e *false* caso contrário.

5.Testes

O código foi gerado pelo compilador Geany 0.20.

Os testes foram realizados em uma máquina com a seguinte descrição:

Hardware/Software	Modelo
Sistema operacional:	Windows 7 Ultimate 32 bits e Ubuntu 10.04
Processador:	Intel Core 2 Duo CPU T6400 2.00 GHz
RAM:	3,00 GB, DDR2, 667MHz

Imobiliária Xulambs

1. Problema proposto

O desafio é escrever um programa que utilize três arquivos do tipo binário para guardar as informações dos clientes, dos imóveis e das locações. Além disso, o programa deverá informar qual será o valor arrecadado pela imobiliária, quais imóveis estão alugados (onde a busca deverá ser feita pela quantidade de quartos), quais imóveis não estão alugados, nome e telefone de todos os locatários (onde a busca será feita pelo valor do aluguel). Será criado pelo menos mais três consultas extras (utilizando mais de um arquivo).

2. Descrições da solução

Foram criados vários procedimentos para efetuar as exigências do desafio. Além do que foi solicitado, criamos procedimentos para inserção de dados no arquivo e impressão na tela de todas as informações cadastradas.

Será criado na pasta que estará o arquivo (*.pas*), arquivos com os nomes: *Cadastro_clientes.bin* (que conterá todas as informações dos clientes cadastrados), *Cadastro_imovel.bin* (que conterá todas as informações dos imóveis que a imobiliária possui), *Cadastro_locacao.bin* (que conterá todas as informações das locações feitas).

A tela inicial do programa será um menu, que quando acessada uma das opções,”chamará” a rotina associada a mesma.

2.1 Listagens de rotinas

- **procedure cadastro_clientes (var arquivo: arquivo_cliente);**

Procedimento responsável pelo cadastro de dados dos clientes. O usuário deverá informar o CPF, nome, telefone e endereço do cliente. À medida que for cadastrado, será escrito no arquivo binário.

- **procedure cadastro_imovel (var arquivo: arquivo_imovel);**

Procedimento responsável pelo cadastro de dados dos imóveis que a imobiliária possui. O usuário deverá informar o tipo do imóvel, endereço, quantidade de quartos, quantidade de vagas na garagem, informar se tem piscina ou não, o código do imóvel e as observações que for cabível aquele imóvel. À medida que for cadastrado, será escrito no arquivo binário.

- **procedure cadastro_locacao (var arquivo: arquivo_locacao);**

Procedimento responsável pelo cadastro das locações efetuadas. O usuário deverá informar o código do imóvel, o CPF do locatário, o CPF do proprietário, o valor do aluguel e o prazo de locação. À medida que for cadastrado, será escrito no arquivo binário.

- **procedure informacoes_clientes (var arquivo: arquivo_cliente);**
- **procedure informacoes_imoveis (var arquivo: arquivo_imovel);**
- **procedure informacoes_locacoes (var arquivo:arquivo_locacao);**

Os três procedimentos citados acima são responsáveis pela impressão na tela de todos os dados cadastrados. O primeiro procedimento se refere às informações dos clientes, o segundo as informações dos imóveis e o terceiro as informações das locações.

- **procedure arrecadacao (var arquivo: arquivo_locacao);**

Procedimento responsável por calcular o valor que a imobiliária arrecadará. Ficou estipulado que serão 10% do valor de cada aluguel. Será calculada a porcentagem dos valores, somado e impresso na tela, à medida que for solicitado no meu de opções.

- **procedure busca_quartos (var arq: arquivo_locacao; var arquivo: arquivo_imovel);**

Este procedimento listará todos os imóveis que estão alugados e possuem 'x' quartos. O usuário deverá informar a quantidade de quartos para efetuar a busca.

Foram atribuídos a um vetor todos os códigos dos imóveis que estão alugados. Dessa forma, serão comparados os códigos do vetor com os códigos de todos os imóveis cadastrados. Quando o código for igual, será comparada a quantidade de quartos que o usuário digitou com a quantidade de quartos daquele imóvel (de código igual ao do vetor). Se for igual, imprimirá na tela, as informações daquele imóvel. Se não for encontrado, será impresso uma mensagem do tipo: "Não há imóveis alugados com essa quantidade de quartos".

- **procedure imoveis_naoalugados (var arquivo: arquivo_imovel ; var arq: arquivo_locacao);**

Este procedimento foi feito de maneira similar ao citado anteriormente. São guardados no vetor os códigos de todos os imóveis alugados e logo após, comparado com os códigos do arquivo 'Imóveis', ou seja, todos os imóveis cadastrados. Se for igual, significa que aquele imóvel está alugado, se não, são impressos na tela todas as informações daquele imóvel. Se caso, todos os imóveis estiverem alugados, será impresso uma mensagem do tipo: "Todos os imóveis estão alugados".

- **procedure busca_valoraluguel (var arquivo: arquivo_cliente; var arq: arquivo_locacao);**

  Procedimento que busca o nome e telefone do locatário pelo valor do aluguel. O usuário deverá informar o valor de aluguel a ser buscado. Se o CPF do arquivo ‘Clientes’ for igual ao CPF do locatário (arquivo ‘Locacao’), é comparado se o valor de aluguel digitado pelo usuário é igual aquele, em que o CPF é igual. Se for imprime o nome e telefone do locatário, caso contrário, será impresso uma mensagem do tipo: 'Não ha imóveis alugados com esse valor de aluguel'.

Consultas extras

- **procedure consulta1 (var arquivo: arquivo_cliente; var arq: arquivo_locacao);**

  Este procedimento é responsável por fazer uma das consultas extras. Optamos, primeiramente, em fazer uma busca para verificar se o cliente é locatário ou proprietário. Se ele for locatário, será informado o valor que ele paga de aluguel e se for proprietário, será informado o valor que ele receberá pelo aluguel (descontando os 10% de arrecadação da imobiliária)

- **procedure consulta2 (var arquivo: arquivo_imovel; var arq: arquivo_locacao);**

  Esse procedimento é responsável por fazer a segunda consulta extra. O usuário poderá buscar pelo número de vagas na garagem e verificar se aquele imóvel está alugado ou não. Se estiver alugado, imprime na tela as informações da locação, se não, imprime as informações daquele imóvel. Caso não haja imóveis com a quantidade de garagens informada pelo usuário, será impresso uma mensagem do tipo: “Nao ha imoveis com essa quantidade de garagens”.

- **procedure consulta3 (var arquivo: arquivo_imovel; var arq: arquivo_locacao);**

  Esse procedimento é responsável pela terceira consulta extra. O usuário busca os imóveis pelo bairro. Se o imóvel estiver alugado, imprime as informações da locação, caso contrário, imprime as informações daquele imóvel. Se não houver imóveis com o bairro informado pelo usuário, será impresso mensagem do tipo: "'Não ha imóveis cadastrados com esse bairro".

3.Testes

O código foi gerado pelo compilador Geany 0.20.

Os testes foram realizados em uma máquina com a seguinte descrição:

Hardware/Software	Modelo
Sistema operacional:	Windows 7 Ultimate 32 bits e Ubuntu 10.04
Processador:	Intel Core 2 Duo CPU T6400 2.00 GHz
RAM:	3,00 GB, DDR2, 667MHz

Conclusão

Utilizando os conceitos de modularização, aproveitando as vantagens oferecidas pela prática, favorecendo inclusive o trabalho em equipe, empregamos nossos conhecimentos na implementação de jogos.

No algoritmo Imobiliária Xulambs, relacionamos elementos de mais de um arquivo, e percebemos que isso é possível desde que se tenha alguma semelhança entre eles (Exemplo: o código do imóvel no arquivo ‘Locações’ e no arquivo ‘Imóveis’, o é o item CPF).

Ao implementar problemas do cotidiano e jogos desenvolvemos e aplicamos nossos conhecimentos de modo a interagir com problemas reais, pois são interessantes meios para despertar o raciocínio e a curiosidade.

Referências Bibliográficas

[1]- Wikipédia, a enciclopédia livre

http://pt.wikipedia.org/wiki/Jogo_da_velha

Acessado em: 23/06/2011

[2]- Wikipédia, a enciclopédia livre

http://pt.wikipedia.org/wiki/Batalha_naval_%28jogo

Acessado em: 23/06/2011

Definição da pontuação

- Jogo da Velha – 3 pontos
- Batalha Naval – 3 pontos
- Imobiliária Xulambs – 4 pontos