Universidade Estadual do Oeste do Paraná
Centro de Ciências Exatas e Tecnológicas
Campus de Cascavel
Curso de Bacharelado em Ciência da Computação
Disciplina: Inteligência Artificial

1ª Avaliação
1º Trabalho – Jogo da Velha

1. Descrição do Problema

O Jogo da Velha (*Tic-Tac-Toe*) é um passatempo bem popular, com regras extremamente simples, facilmente aprendido e que não traz grandes dificuldades para seus jogadores. Entretanto, analisando o número de possibilidades de forma simplista, existem 362.880 (ou 9!) maneiras de se dispor a cruz e o círculo no tabuleiro, sem considerar jogadas vencedoras (considerando só jogadas vencedoras são 255.168 jogos possíveis).

Um programa simples, que resolve o jogo, pode ser visto no Programa 1. Perceba que um dos requisitos é preencher, corretamente e de forma completa, a tabela de movimentos.

Programa 1: Jogo da Velha (força bruta)

- Estruturas de Dados:

Tabuleiro:

- Um vetor de 9 elementos que representa o tabuleiro, onde os elementos do vetor correspondem às seguintes posições no tabuleiro:

1	2	3
4	5	6
7	8	9

- Um elemento contém o valor 0 (zero) se o quadrado correspondente estiver em branco, 1 (um) se estiver preenchido com X ou 2 (dois) se estiver preenchido com O.

Tabela de Movimentos: um vetor de 19.683 elementos (3^9) onde cada elemento é um vetor de 9 elementos. O seu conteúdo deve ser preenchido adequadamente para que o algoritmo funcione.

- Algoritmo:

```
Para obter um movimento faça:
  1. visualize o vetor 'tabuleiro' como um número ternário
     e converta-o em decimal;
  2. use o número calculado na etapa 1 como índice da
     'tabela de movimentos' e acesse o vetor lá armazenado;
```

o vetor selecionado na etapa 2 representa o modo como o tabuleiro ficará após o movimento. Iguale o tabuleiro a este vetor.

Algumas observações a serem feitas sobre esse programa:

1. É muito eficiente em termos de tempo. E, em tese, consegue um nível ideal para o jogo;
2. É preciso muito espaço para armazenar a tabela que especifica o movimento correto a ser feito a partir de cada posição do tabuleiro;
3. Alguém terá de trabalhar muito para especificar todas as inserções da tabela de movimentações;
4. Não é provável que todas as inserções da tabela de movimentações sejam determinadas e inseridas sem erros;
5. Se quisermos ampliar o jogo, digamos, para 3 dimensões, teremos de partir do "zero" novamente. Na verdade, este programa jamais funcionaria para 3 dimensões, já que haveria a necessidade de armazenar 3^{27} posições no quadro (o que estouraria a capacidade de memória).

Poderíamos melhorar esse programa? É possível melhorar o uso de espaço: utilizando algumas estratégias, não é necessário armazenar todas as possíveis jogadas. Essa melhora pode ser vista no Programa 2.

Programa 2: Jogo da Velha (com estratégia)
- Estruturas de Dados: *Tabuleiro:* vetor com 9 elementos representando o tabuleiro conforme escrito no programa 1, mas em vez de se usar os números 0, 1 e 2 em cada elemento, armazenamos 2 (indicando espaço em branco), 3 (indicando X) ou 5 (indicando O). *Jogada:* um inteiro indicando que movimento do jogo está para ser feito: 1 indica o primeiro movimento, 9 o último. ***- Algoritmo:*** O algoritmo principal utiliza 3 procedimentos:

Faz2:
Retorna 5 se o quadrado central está em branco, isto é, se Tabuleiro[5] = 2. Caso contrário, retorna qualquer quadrado que esteja em branco e que não seja de canto (2, 4, 6, 8).

Ganha(p):
Retorna 0 se o jogador p não tiver condições de ganhar na sua próxima movimentação. Caso contrário retorna o número do quadro que constitui movimento **vitorioso**. Esta função permite que o programa vença e bloqueie a vitória do oponente. Ganha(p)

opera verificando as linhas, colunas e diagonais, uma de cada vez. Devido ao modo como os valores são numerados, ele pode testar uma linha inteira (ou coluna ou diagonal), multiplicando os valores de seus quadrados. Se o produto é 18 (3 x 3 x 2), então X pode ganhar. Se o produto é 50 (5 x 5 x 2) então O pode ganhar. Se encontramos uma linha (ou coluna ou diagonal) vencedora, determinamos que elemento está em branco e retornamos seu índice.

Jogue(n):

Faz um movimento no quadrado n. Este procedimento ajusta tabuleiro[n] para 3 se Jogada for ímpar ou em 5 se Jogada for par. Também incrementa Jogada em um.

A estratégia de cada movimento é:

Jogada = 1: Jogue(1) (canto esquerdo superior)

Jogada = 2: Se Tabuleiro[5] estiver em branco, Jogue(5), caso contrário, Jogue(1).

Jogada = 3: Se Tabuleiro[9] estiver em branco, Jogue(9), caso contrário, Jogue(3).

Jogada = 4: Se Ganha(X) ≠ 0, então Jogue(Ganha(X)) **(bloqueia vitória do oponente)**. Caso contrário, Jogue(Faz2).

Jogada = 5: Se Ganha(X) ≠ 0 então Jogue(Ganha(X)) **(vença)**. Senão, se Ganha(O) ≠ 0, Jogue(Ganha(O)) **(bloqueia a vitória do oponente)**. Senão, se Tabuleiro[7] estiver em branco, Jogue(7), senão, Jogue(3) **(tenta fazer um garfo)**.

Jogada = 6: Se Ganha(O) ≠ 0, Jogue(Ganha(O)). Senão, se Ganha(X) ≠ 0, Jogue(Ganha(X)), senão Jogue(Faz2).

Jogada = 7: Se Ganha(X) ≠ 0 então Jogue(Ganha(X)). Senão, se Ganha(O) ≠ 0, Jogue(Ganha(O)). Senão, jogue em qualquer espaço em branco.

Jogada = 8: Se Ganha(O) ≠ 0, Jogue(Ganha(O)). Senão, se Ganha(X) ≠ 0, Jogue(Ganha(X)). Senão, jogue em qualquer espaço em branco.

Jogada = 9: Idem Jogada = 7.

Observações sobre esse programa:

1. Não é tão eficiente em termos de tempo quanto o primeiro (ele precisa verificar várias condições antes de fazer cada jogada, no primeiro, o acesso ao movimento é direto, via índice do vetor);
2. É muito mais eficiente em termos de espaço;
3. É mais fácil entender a estratégia do programa ou alterar a estratégia, se desejado;
4. Problema: a estratégia global ainda precisa ser preparada com antecedência pelo programador (qualquer falha na capacidade do programador em jogar o jogo da velha aparecerá na execução do programa);
5. Também não é possível generalizar para um domínio diferente, como um jogo da velha em 3 dimensões.

Uma solução mais eficiente, utilizando uma técnica de IA mais aprimorada, utiliza o procedimento de busca minimax, que é um procedimento de busca em profundidade, de profundidade limitada. Esse tipo de busca é mais avançado, e não será visto neste momento.

2. O trabalho

O trabalho consiste na implementação do programa 2, descrito neste documento, e vale 40 pontos para a 1ª avaliação.

O programa implementado pode ou não ter interface gráfica. Se a equipe optar por não criar a interface gráfica, a equipe deverá criar um arquivo (pode ser `.txt`) com as instruções para jogar via linha de comando.

Demais informações importantes:

- **Equipes:** esse trabalho pode ser realizado em equipes de até 4 alunos. Não serão permitidas equipes maiores.
- **Linguagem:** cada equipe poderá realizar a implementação em uma das 3 linguagens: C, C++ ou Java.
- **Implementação:** todos os códigos devem estar bem comentados e com o nome dos integrantes da equipe. Devem ser entregues todos os códigos-fontes da implementação e todos os arquivos necessários à compilação (se não implementarem interface gráfica, incluir o arquivo com as instruções).
- **Ambiente de testes:** a execução dos testes para a correção do trabalho será realizada via console, em máquina com sistema Unix. Não será realizada execução via ambiente de edição (Netbeans, Eclipse, QT, etc). Não usar bibliotecas para Windows.
- **Entrega:** via Moodle, até as 08hs do dia 15/04/2019. **NÃO SERÃO ACEITOS** trabalhos atrasados **E/OU** entregues fora da plataforma Moodle. Todos os arquivos da implementação devem ser compactados em um arquivo `.zip`, nomeado com o nome dos integrantes da equipe. Basta que apenas um dos integrantes submeta o trabalho.