DECISÃO E APRENDIZADO NO CONTEXTO DE UM JOGO

WOLFGANG EIKMEIER

ORIENTADOR

PROF. DR. FERNANDO CURADO

Dissertação apresentada ao Instituto de Matemática, Estatística e Ciência da Computação como requisito parcial para obtenção do título de Mestre em Ciência da Computação.

Julho - 1982

O objetivo de Inteligência Artificial é a construção de programas que exibem um comportamento que chamamos de "comportamento inteligente" quando o observamos em um ser humano.

EDWARD A. FEIGENBAUM & JULIAN FELDMAN

A meus pais

Christel e Siegfried.

AGRADECIMENTOS

À *Fernando Curado* pelo trabalho de orientação.

À *Orlando Seigi Nakano* pela contribuição dada na otimização das rotinas de entradas/saídas.

À *minha esposa* pelo apoio fornecido.

À vários *amigos* com quem pude compartilhar as minhas idéias.

À *Alice Rodrigues de Souza* pelos trabalhos de edição e datilografia.

ÍNDICE

ANEXOS:

ANEXO-A - PROGRAMA QUE IMPLEMENTA O JOGO "TAC-TICKLE"

ANEXO-B - CONFIGURAÇÕES TERMINAIS PARA O JOGO "TAC-TICKLE"

ANEXO-C - PARTIDAS EXEMPLOS

ANEXO-D - NÚMERO DE JOGADAS SIMULADAS

ABSTRACT

This work shows an application of Artificial Intelligence techniques to the area of problem - solving. An implementation, in PASCAL, of an algorithm which plays TACK-TICKLE and learns with its mistakes is presented.

The algorithm diverges from the traditional published work in the sense that it tries to model plausible forms of playing by human beings. Its fundamental characteristic is to be structured around classes of "preocupations" typical of game players.

The learning module stores, in a limited and compact form, information needed so that past errors are not repeated. Consequently, the program improves its performance as more games are played.

RESUMO

Este trabalho mostra uma aplicação de técnicas de Inteligência Artificial à área de resolução de problemas. É apresentada a implementação, em Pascal, de um algoritmo que joga "Tac-Tickle" e aprende com os erros cometidos.

O algoritmo diverge das formas tradicionais da literatura, no sentido em que, procura modelar formas plausíveis de um ser humano realizar jogadas. Sua característica fundamental é ser estruturado em torno de classes de preocupações típicas de participantes de jogos.

O módulo de aprendizado armazena, de forma limitada e compacta, informações necessárias para que erros passados não sejam mais repetidos. Como consequência do módulo de aprendizagem, o programa melhora seu desempenho à medida em que mais partidas são realizadas.

1. INTRODUÇÃO

Este trabalho apresenta técnicas de Inteligência Artificial aplicada à área de resolução-de-problemas.

A aplicação consiste numa heurística associada a um processo simples de aprendizagem procurando modelar uma das classes do raciocínio humano. Foi escolhido o campo de jogos porque estes possuem as características básicas de uma atividade intelectual para a qual procedimentos heurísticos e processos de aprendizagem assumem papel importante.

Escolheu-se o jogo "Tac-Tickle" [1] e para este jogo implementou-se um programa que:

a). joga com tática própria - aplicação de técnicas heurísticas,

b). aprende com seus erros - implementação de um processo simples de aprendizagem.

Neste trabalho Inteligência Artificial está sendo vista segundo a definição de Feigenbaum & Feldman - [2] - Construção de programas que exibem um comportamento que seria chamado inteligente se observado em um ser humano. Segundo a classificação de Nilsson [3] este trabalho procura ampliar o núcleo de aplicações de primeiro nível.

Os jogos HOMEM VERSUS MÁQUINA, podem ser divididos em em duas categorias:

a). Jogos como o "Jogo da Velha", para os quais existe um algoritmo que garante senão a vitória, pelo menos o empate. Diz-se que existe estratégia de ganho.

b). Jogos como o "Xadrez", para os quais não se conhece uma estratégia de ganho. Nestes casos, processos heurísticos tornam-se importantes não só na implementação de programas, como também, na própria forma humana de jogar.

O jogo escolhido encontra-se evidentemente enquadrado na segunda categoria."Tac-Tickle" foi também escolhido porque a simplicidade de suas regras e sua complexidade - aproximadamente 9×10^6 configurações possíveis,facilitam a abordagem e implementação dos dois ítens centrais do estudo - heurística e aprendizagem.

1.2. REGRAS DO JOGO "TAC-TICKLE"

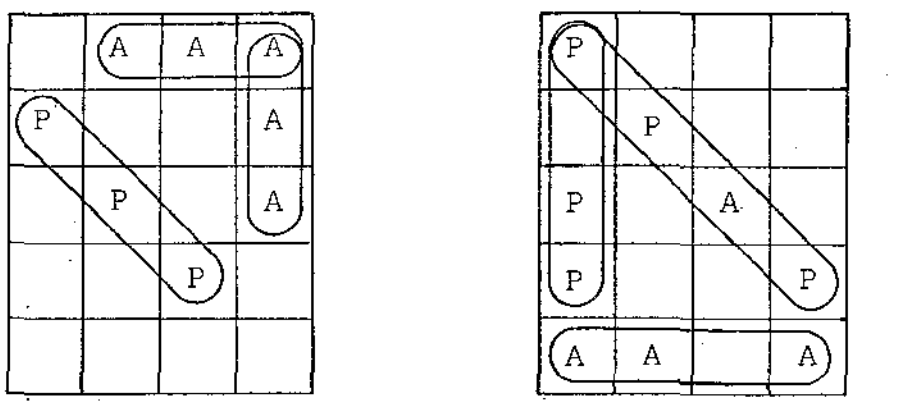
O jogo consiste em um tabuleiro retangular de 20 casas dispostas num arranjo de 5 x 4. Cada um dos dois jogadores dispõe de quatro (4) peças que são inicilamente dispostas como na figura 1.

P	A	P	A
A	P	A	P

FIGURA 1

Os jogadores alternam jogadas, isto é, movimentam as peças de sua cor segundo as seguintes normas:

1. As peças podem ser movimentadas para casas livres adjacentes situadas à direita, esquerda, acima ou abaixo de onde a peça está. Não são permitidos movimentos na diagonal, não existem saltos, nem capturas.
2. Vence o jogo aquele que primeiro conseguir agrupar três de suas peças em linha vertical, horizontal ou diagonalmente, sem que exista uma casa vaga entre elas (Figura 2).



Situações Vitoriosas Situações Não Vitoriosas

FIGURA 2

2. REVISÃO BIBLIOGRÁFICA

2.1. DESCRIÇÃO DO ALGORITMO MINIMAX

Os programas de jogos homem versus máquina baseados em técnicas de descrição de representação do espaço de estados ("State space representation") Nilsson [4] , utilizam o algoritmo Minimax proposto por Claude Shannon [2] em 1949. A idéia básica da técnica Minimax é a de supor que cada jogador efetua a jogada que maximiza o seu potencial de vitória e minimiza o do adversário.

O algoritmo divide-se em quatro fases distintas:

- Derivação/Seleção das jogadas
- Terminação
- Avaliação
- Escolha da jogada a ser realizada.

A fase de Derivação/Seleção consiste na geração, a partir da configuração de momento do jogo, das configurações (todas ou algumas se houver seleção) decorrentes da simulação das futuras jogadas do programa e do adversário.

Na fase de Terminação, determinam-se as configurações a partir das quais não mais será aplicado o processo de Derivação/Seleção, estas são chamadas de Configurações Folha. Normalmente efetua-se a Terminação uma vez atingido um número (nível) pré-determinado de derivações.

Na fase de Avaliação atribui-se um valor às configurações folha derivadas, qualificando-se quantitativamente a potencialidade da configuração em levar o programa à vitória. A avaliação é normalmente obtida através de valores numéricos representando um determinado aspecto da configuração (vantagem material, mobilidade, vulnerabilidade, etc). Atribui-se o nome de Função de Avaliação ao procedimento que especifica a regra de formação do valor numérico a ser associado a cada configuração folha. Terminada a fase de avaliação teremos numericamente qualificadas as configurações folha.

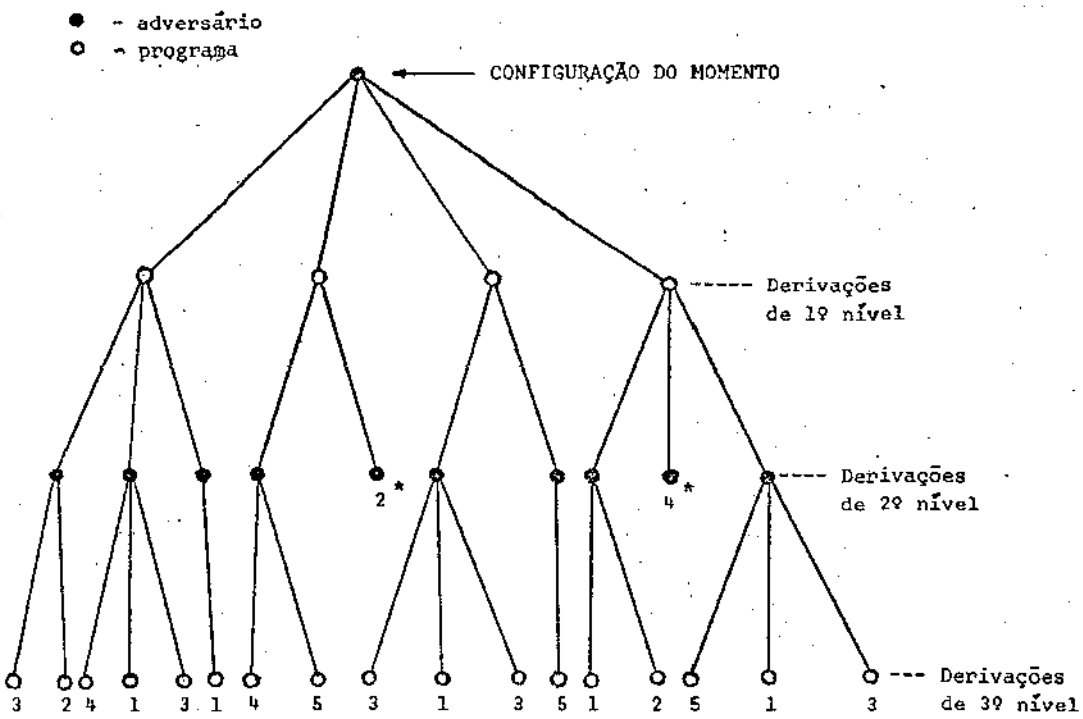


Figura 3

A figura 3 ilustra uma árvore de derivações obtida após as fases de Derivação/Seleção, Terminação e Avaliação.

Note-se que, neste exemplo, todas as configurações decorrentes das derivações de 3º nível bem como aquelas marcadas com asterisco são configurações folha, tendo sido numericamente avaliadas.

Partindo-se das configurações folha devidamente avaliadas, inicia-se o processo de determinação da melhor jogada, por meio de um critério minimax. Rotula-se cada nó pai com o maior (nó pai é adversário) ou menor (nó pai é programa) valor correspondente aos seus nós filhos. Repete-se o processo até que os nós correspondentes às derivações de 1º nível estejam rotulados. A atribuição do maior ou menor valor numérico a um dado nó pai traduz correspondentemente a maximização (programa joga) ou minimização (adversário joga) do potencial de vitória do programa. No procedimento Minimax supõe-se portanto, que o adversário, invariavelmente, efetuará a sua melhor jogada.

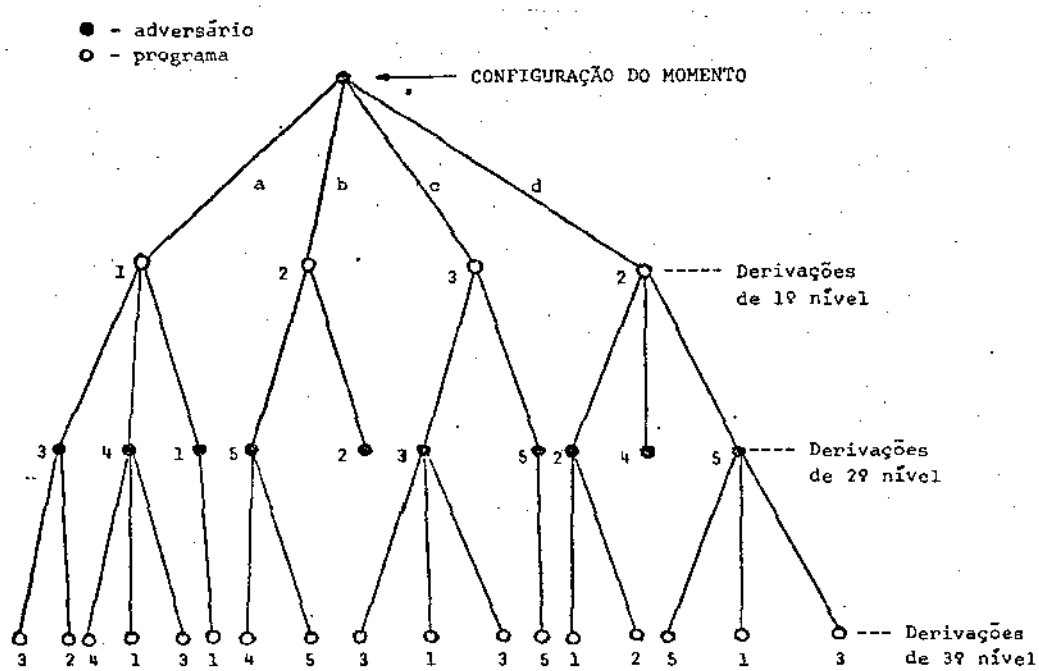


Figura 4

A figura 4 ilustra o processo de determinação da melhor jogada partindo-se das configurações folha da figura 3. A jogada C será a escolhida.

2.2. ANÁLISE CRÍTICA DO MINIMAX E ALGUMAS DE SUAS VARIANTES

A análise completa de todas as continuações a uma dada jogada, um dos aspectos mais críticos do procedimento Minimax, tem feito com que, desde 1950, variantes deste método, tal como o algoritmo Alfa-beta, tenham sido propostas e implementadas [5] . Diversas também foram as otimizações do procedimento Minimax, tanto no que se refere a sua concepção inicial Estratégia-A (análise completa de todas as continuações até um certo nível), quanto no que se refere à sua concepção Estratégia-B (somente jogadas plausíveis são consideradas em cada derivação) [5] .

Aliadas ao crescente avanço do "hardware", as alterações sugeridas têm tornado plausível (mas pouco elegante) a execução da estratégia-A de Shannon. CHESS 4.6 utiliza-se desta estratégia até a análise de nível-6, após o que utiliza conceitos de seleção e terminação.

A descoberta feita simultaneamente por SLATE e ATKIN, autores de CHESS 4.7, e pela equipe russa autora do programa KAISSA, de que, ironicamente, a abordagem de força bruta ("Brute Force Approach"), produz jogadas mais brilhantes do

que o da procura seletiva, leva-nos a questionar a validade das otimizações implementadas nas estratégias de Shannon. Se não vejamos: Os estudos têm-se concentrado em dois aspectos: o de seleção das jogadas e o de avaliação das configurações. A questão de descobrir heurísticas que façam escolhas adequadas de "jogadas plausíveis" esbarra na descoberta de Slate e Atkin e da equipe russa. O que pode parecer plausível numa visão local pode levar à eliminação de jogadas, que numa visão global, poderiam ser consideradas brilhantes. Uma vez que já existe poder computacional para implementar a estratégia-A de Shannon, restaria considerar apenas melhores formas de avaliar as configurações folha. Por outro lado, sabe-se que para tornar-se um mestre de xadrez, o ser humano necessita dedicar-se continuamente ao estudo deste jogo, por no mínimo cinco anos, à base de vinte horas semanais. Durante este tempo, grande quantidade de conhecimento específico adquirido possibilita o reconhecimento imediato de características "sui generis" de uma dada configuração, de forma a sugerir objetivos a curto e a longo prazo, bem como, jogadas específicas [5] . Decorre daí a dificuldade em aceitar-se que tamanho potencial de conhecimento (tático e estratégico) possa vir a ser condensado numa simples função de avaliação. Note-se que isto não exclui a possibilidade de que se possa ter programas que utilizando-se da estratégia-A de Shannon e capitalizando o poder computacional crescente das máquinas existentes, venham a vencer grandes mestres internacionais. Estar-se-á porém empregando a força

bruta para vencer a habilidade humana. Esta por sua vez baseia-se em dois fatores primordiais: reconhecimento de padrões e rápida recuperação de informação. De posse de um processador bio-químico, cujo tempo de ciclo é incrivelmente baixo, cerca de 100 operações por segundo, onde a unidade básica, o neurôneo, opera a taxas de milisegundos ao invés de nanosegundos, o ser humano consegue, com a análise de não mais que 100 e em média 35 configurações (considerando-se o jogo de xadrez), o que a máquina não consegue com a análise de milhares ou até milhões. Não seria a característica anti-natural do procedimento Minimax responsável por tão absurdas discrepâncias ? A implementação descrita nos capítulos 4 e 5 procura mostrar que é possível estabelecer algoritmos que se aproximam de uma metodologia humana de jogar e que estes algoritmos possuem uma certa generalidade que transcende às características específicas do jogo para o qual foi implementado.

2.3. APRENDIZAGEM NUM CONTEXTO DE MÁQUINA

Podemos diferenciar duas categorias básicas de aprendizado. Na primeira a máquina aprende tendo por base um mini mundo no qual existe um conjunto de regras bem definidas e conhecidas. Na segunda, muito mais complexa e aonde se enquadram a maioria dos problemas do nosso cotidiano, a determi

nação das regras é parte inerente ao processo de aprendizado. Aqui, com base nas informações obtidas no meio externo, induz-se o modelo a ser armazenado internamente. Neste contexto a formulação de hipóteses e o reconhecimento de padrões, ítens indispensáveis ao aprendizado, são aspectos ainda não solucionados em inteligência artificial.

As técnicas de aprendizado até então implementadas encontram-se evidentemente na primeira categoria. Estas se baseiam na otimização do procedimento Minimax, seja através do armazenamento e recuperação de configurações numericamente já avaliadas (processo de Decorar - "Rote-learning" [2] e [5]) ou da otimização da função de avaliação "learning involving generalization" [2].

Nem a economia de tempo decorrente do processo de Decorar, tão pouco a otimização da função de avaliação parecem poder vir a melhorar o nível estratégico dos programas (veja ítem 2.2.).

A utilização de um aprendizado diretamente em função dos erros cometidos, forma semelhante à descrita por Russel R.Yost Jr. em [6] , caracteriza uma das maneiras pela qual o ser humano aprende a jogar, tendo sido escolhida para a implementação do processo de aprendizado.

3. DEFINIÇÕES E NOTAÇÃO

As definições abaixo estabelecem a terminologia adotada no desenvolvimento e implementação deste trabalho:

JOGADA - movimentação de uma peça por um jogador. O jogador que executa a jogada tem o mando do jogo.

JT : JOGADA TERMINAL - é uma jogada que leva à vitória.

JTB : JOGADA TERMINAL BARRADA - é uma jogada terminal que poderia ser executada pelo adversário caso este tivesse o mando do jogo.

JD : JOGADA DECISIVA - é uma jogada que quando efetuada, leva à certeza de que o jogador que a executou poderá chegar a uma jogada terminal.

CONFIGURAÇÃO - é o estado do jogo (posição relativa das peças) num dado momento.

CT : CONFIGURAÇÃO TERMINAL - é a configuração que possibilita a execução de uma jogada terminal ou aponta a existência de uma jogada terminal barrada.

CD : CONFIGURAÇÃO DECISIVA - é a configuração que permite a execução de uma jogada decisiva.

CF : CONFIGURAÇÃO FINAL - é a configuração obtida após a execução de uma jogada terminal.

CP : CONFIGURAÇÃO DE PERDA - é a configuração a partir da qual existe, para o adversário, uma jogada decisiva.

CS : CONFIGURAÇÃO CONSEQUENTE - é qualquer configuração de perda derivada de um processo de perda previamente identificado.

O conceito de configuração consequente é mais facilmente entendido por um exemplo:

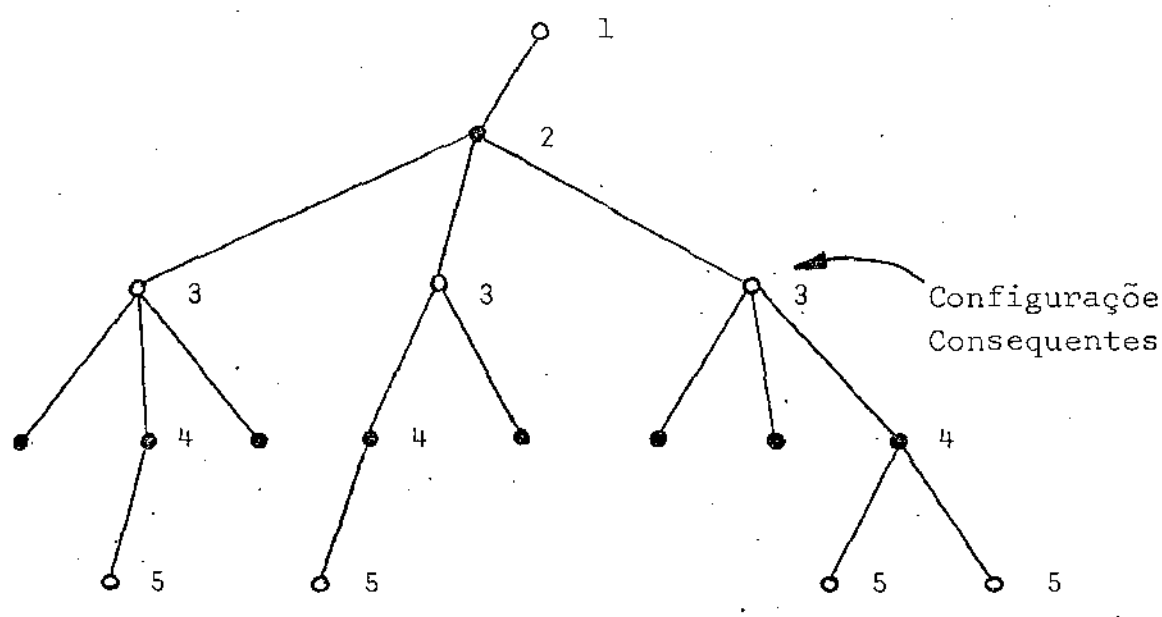


FIGURA 5 - EXEMPLO DE CONFIGURAÇÃO CONSEQUENTE

No ponto 2 o programa verifica que para qualquer uma de suas jogadas possíveis, representadas pelo conjunto 3, existe um possível caminho de vitória para o seu adversário, representado pelo conjunto 4. A configuração representada pelo

ponto 1 será armazenada como configuração de perda. Note-se que as jogadas do conjunto 5, momento em que o programa já se encontra no processo de perda, são efetuadas de forma a dar-se continuidade ao jogo, sabendo-se porém que as mesmas podem levar à derrota. Identificam-se as configurações do conjunto 3 como consequentes.

CN : CONFIGURAÇÃO NEUTRA - é qualquer configuração que não terminal, decisiva, final, de perda ou consequente.

VF : VERDADEIRO-FALSO - é a recuperação, durante o processo de aprendizagem, de uma configuração como sendo de perda sem que esta jamais tenha sido armazenada como tal.

Finalmente diz-se que determinado conceito se aplica ao programa ou ao adversário pela notação:

CONCEITO [x] , onde x = P - programa
x = A - adversário

Exemplo:

JD [P] - jogada decisiva para o programa
JTB [A] - jogada terminal barrada para o adversário

4. IMPLEMENTAÇÃO DO ALGORITMO

O algoritmo proposto neste trabalho para arcabouço geral de programas de jogos do tipo "homem versus máquina" [6], procura aproximar-se de uma das formas possíveis do raciocínio humano no tratamento de problemas desta natureza. Paralelamente procura contornar alguns aspectos desfavoráveis do algoritmo Minimax como:

- a dificuldade em limitar-se o número de jogadas consideradas pelo processo de seleção, sem que, inadvertidamente, elimine-se a melhor jogada.
- a não escolha da jogada que oferece as maiores chances de vitória quando a partida é contra um adversário fraco.
- o fato de que um grande número de continuações necessitam ser analisadas quando do incremento de um nível na análise.
- a necessidade de explorar-se cerca de 10^4 continuações a mais do que a média analisada por um ser humano (considerando-se o jogo de xadrez).
- a dificuldade de se associar uma equivalência precisa entre o nível de análise efetuado pelo programa e a habilidade decorrente do mesmo (característica normalmente não encontrada no ser humano).

Nem todos estes aspectos poderão ser eliminados única e exclusivamente pela ação do algoritmo. A técnica de aprendizado, por exemplo, será utilizada para eliminar a dificuldade em limitar-se o número de jogadas (no processo de seleção) sem que inadvertidamente se elimine a melhor jogada.

Portanto, o algoritmo e o fator aprendizado, descrito no capítulo 5, formarão um conjunto coeso levando aos objetivos propostos.

4.1. MACRO ESPECIFICAÇÃO DO ALGORITMO

O processo de jogar pode ser hierarquizado por seis classes de questões que representam preocupações humanas no momento da escolha da melhor jogada. As seis classes são:

CLASSE A : Existe a possibilidade de execução de uma jogada terminal ?

CLASSE B : Existe a possibilidade de execução, pelo adversário, de uma jogada terminal ? Uma vez que o programa neste momento tem mando, esta jogada terminal passa a ser chamada de barrada.

CLASSE C : Qual a melhor defesa para uma jogada terminal barrada já identificada ?

CLASSE D : Existe a possibilidade de execução de uma jogada decisiva ?

CLASSE E : Qual a melhor jogada fora do contexto das classes A - D ?

CLASSE F : Existe a possibilidade de uma jogada decisiva por parte do adversário ?

No algoritmo proposto estas classes de questões são hierarquizadas segundo uma heurística que parece ser natural ao ser humano. Primeiro é respondida a questão de classe A. A seguir são respondidas as questões B, C e D. Para qualquer uma destas questões, uma resposta afirmativa provoca uma jogada e a consequente troca de mando do jogo. A questão de classe E, por sua vez, exige o desenvolvimento de uma heurística própria decorrente da suposição de que a seguinte forma de jogar é comumente encontrada:

a). Selecionam-se as jogadas possíveis a serem analisadas. A seleção baseia-se no descarte das jogadas que comprovadamente levam a uma configuração de perda. Estas são identificadas por meio de um critério baseado em experiência (MÓDULO DE APRENDIZADO).

b). Ordenam-se as jogadas assim obtidas em ordem crescente de "favorabilidade" (jogadas mais favoráveis primeiro). A "favorabilidade" é obtida aplicando-se às configurações resultantes (após simulada a jogada) uma função de avaliação.

c). Escolhe-se a primeira jogada que, efetuada, garanta a inexistência de respostas afirmativas à questões de classes B ou F. Uma vez encontrada, a análise das demais é dispensada.

As classes são implementadas por cinco módulos que, partindo da configuração do momento, retornam a jogada mais indicada:

MÓDULO A : Procedimento que reconhece a existência de uma CONFIGURAÇÃO TERMINAL.
Retorna a jogada terminal (ou jogada terminal barrada) respectiva.

MÓDULO B : Procedimento que identifica, para o programa, a existência de uma CONFIGURAÇÃO DECISIVA, retornando a jogada decisiva a ser executada.

MÓDULO C : Procedimento que retorna a melhor defesa do programa para uma jogada terminal barrada do adversário. Melhor defesa é aquela que, nesta ordem, gera:

1). uma JD [P];

2). um laço de jogadas terminais barradas;

3). uma configuração neutra.

MÓDULO D : Procedimento que retorna a melhor jogada. Este módulo implementa a classe E.

MÓDULO E : Procedimento que identifica a existência de uma CONFIGURAÇÃO DECISIVA para o adversário.

A existência de uma CONFIGURAÇÃO DECISIVA necessariamente é decorrente da identificação de uma JTB. Existindo a JTB devem ser simuladas as possíveis defesas do adversário, caso se esteja procurando uma JD[P], ou deve ser encontrada a melhor defesa do programa (MODULO C), caso se esteja procurando uma JD[A]. No primeiro caso utiliza-se o MODULO B, no segundo o MODULO E.

A figura 6 apresenta, sob forma de português estruturado o algoritmo que implementa o processo descrito que será chamado de processo automático de decisão (Algoritmo PAD). A figura também mostra a relação entre os módulos e as classes. Note-se a inclusão do MODULO E no MODULO D, uma vez que a jogada [P] selecionada será validada somente quando não existir uma JD[A]. Neste caso o MODULO E retorna a condição de verdadeiro na variável JLEGAL. É utilizada a seguinte notação:

- Variáveis em MAIÚSCULA
- Procedimentos em MAIÚSCULA
- Demais termos em minúscula

```
Início (1)
   FRACASSO:= falso  SUCESSO:= falso  ACEITA-VEZ;
   Repita
      Caso VEZ de
      adversário :  Início (2.1)
                       ACEITA-JOGADA[A]  VALIDA-JOGADA[A]  VEZ:= programa
                    Fim (2.1)
      programa :    Inicio (2.2)
                       MODULO-A[P]                          ──► CLASSE A
                       Se existe CT[P] então
                    Inicio (3.1)
                       JLEGAL:= verdadeiro  SUCESSO:= verdadeiro
                    Fim (3.1)
                                                  senão
                    Inicio (3.2)
                       MODULO-A[A]                          ──► CLASSE B
                       Se existe CT[A] então
                    Inicio (4.1)
                       MODULO C                             ──► CLASSE C
                       Se existe DEFESA [P] então  JLEGAL:= verdadeiro
                                                   senão   JLEGAL:= falso
                    Fim (4.1)
                                                  senão
                    Início (4.2)
                       MODULO B                             ──► CLASSE D
                       Se existe JD[P] então  JLEGAL:= verdadeiro
                                              senão
                    Inicio (5.1)
                       MODULO D                             ──► CLASSE E
                       MODULO E                             ──► CLASSE F
                    Fim (5.1)
                    Fim (4.2)
                 Fim (3.2)
                    Se  JLEGAL:= verdadeiro então  EFETUA  jogada [P]
                        senão  FRACASSO:= verdadeiro
                 Fim (2.2)
                 VEZ:= adversário
   ate que FRACASSO = verdadeiro ou SUCESSO = verdadeiro
Fim (1)
```

FIGURA 6 - ALGORITMO PAD

4.2. DESCRIÇÃO DOS MÓDULOS

A característica fundamental do algoritmo PAD, visto como programa principal, é a grande interdependência entre seus módulos. Esta é decorrência natural da complexidade do processo de decisão, pois ataques podem gerar possibilidades de novos ataques, que por sua vez podem gerar a necessidade de possíveis defesas, que por sua vez podem gerar condições de jogo neutro. A interrelação é ainda mais complexa quando se introduz o conceito de nível de análise (variável QI). Se QI for diferente de zero o MODULO E simula o bloco 4.2 do programa principal (figura 6) tantas vezes quantas for o seu valor.

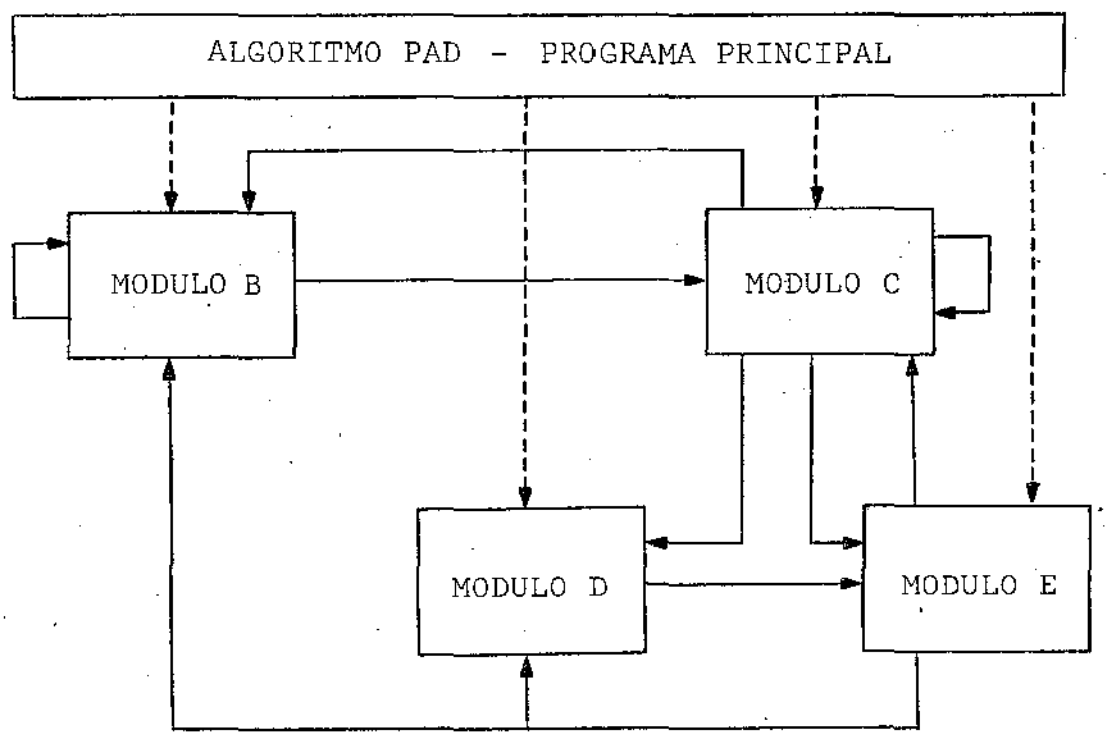


FIGURA 7 - CHAMADAS ENTRE MODULOS

A figura 7 ilustra esta situação e demonstra que a análise isolada de cada módulo não fornece uma visão geral do desempenho do algoritmo. Na figura foi omitido o MODULO A por ser sempre usado pelos demais.

Com estas ressalvas, apresenta-se a seguir a descrição de cada módulo, enfatizando-se que, somente depois de se entender os detalhes será possível uma melhor compreensão do todo.

4.2.1. Módulo A

Argumento de entrada: jogador = (Programa ou adversário)

Argumento de saída: Pilha de JT's ou JTB's

É um módulo de suporte implicitamente utilizado pelos demais. Pode ser implementado através da simulação de todas as possíveis jogadas para o jogador indicado (argumento de entrada). Feita a simulação, verifica-se para cada uma das configurações obtidas, se a mesma representa ou não uma configuração final. Caso afirmativo empilha-se a jogada simulada na pilha de JT/JTB.

4.2.2. Módulo B

Argumento de entrada: Configuração do momento.

Argumento de saída: Existência ou não de uma CD [P] .
Indicação da respectiva JD [P] .

A figura 8 apresenta, sob a forma de português estruturado o MODULO B, que é acionado apenas se a jogada [P] simulada gerar uma CT [P], ou, equivalentemente, se existe uma JTB [P].

É continuado, simulando-se as possíveis defesas, sempre que existe uma JTB. Quando a JTB é uma JTB [A], o MODULO C é utilizado para determinar a melhor defesa [P].

É recursivo quando da não existência de uma JTB [A], simulando-se novas jogadas [P].

É interrompido quando:

a). existir uma JTB [P] que não admite defesa [A]. Neste caso pode existir uma JD [P];

b). existir um laço no processo de análise (configurações repetidas). Neste caso será atribuido o valor falso à variável global JOGADA-DECISIVA. Assume-se não ser vantajoso o laço, (não existência de JD [P]), pois o processo é de ataque. Este fato não está evidenciado no algoritmo da figura 8.

A existência de uma JD [P] será identificada retornando-se à condição de verdadeira na variável JOGADA-DECISIVA. Note-se que o MODULO C retornará a condição de verdadeira na variável JOGADA-DECISIVA, somente quando a defesa [P] sugerida levar a uma JD [P].

```
Inicio (1)
    JOGADA-DECISIVA:= falso
    Repita
        SIMULA JOGADA [P]   MODULO-A [P]
        Se existe JTB [P] então
        Inicio (2.1)
            JOGADA-DECISIVA:= verdadeiro
            Repita
                SIMULA DEFESA [A]   MODULO-A [A]
                Se não existe JTB[A]então MODULO-B
                Senão
                Inicio (3.1)
                    MODULO-C
                Fim (3.1)
            Até que JOGADA-DECISIVA = falso ou
                    não existe outra DEFESA [A]
        Fim (2.1)
    Até que JOGADA-DECISIVA = verdadeira ou
            não existe outra JOGADA [P]
Fim (1)
```

FIGURA 8 - MODULO B

4.2.3. Módulo C

Argumento de entrada: CT [A]

Argumento de saída: Existência ou não de uma defesa [P] e indicação da melhor defesa, caso exista.

A figura 9 apresenta sob a forma de português estruturado o algoritmo correspondente ao MODULO C. Note-se que:

a). Se depois de efetuada uma possível defesa [P], não existir qualquer JTB [P], verifica-se a existência ou não de uma JD [A] através da chamada do MODULO E (bloco 2.2 - figura 9). Caso existir uma JD [A] a defesa [P], será invalidada.

b). Quando não houver uma JTB [A] (bloco 3.2 - figura 9), determina-se a existência ou não de uma JD [P] por meio da chamada do MODULO B. A defesa [P] será uma JD [P], quando esta existir, já que o bloco REPITA (A) da figura 9 é terminado somente quando a variável JOGADA-DECISIVA for verdadeira ou não existir outra possível defesa [P].

c). Existindo uma JTB [A] (bloco 3.1 - figura 9), o processo é recursivo, sendo abortado quando ocorrer um laço. Neste caso atribui-se à variável DEFESA [P] qualquer uma das possíveis defesas [P]. Assume-se ser vantajoso o laço pois o processo é de defesa. Este fato não é evidenciado na figura 9.

```
Inicio (1)
   DEFESA [P] := inexistente
   Repita (A)
      SIMULA  POSSÍVEL-DEFESA [P]  MODULO-A [P]
      Se existe JTB [P] então
      Inicio (2.1)
         JOGADA-DECISIVA := verdadeiro  EXISTE-DEFESA := verdadeiro
         Repita (B)
            SIMULA POSSÍVEL-DEFESA [A]  MODULO-A [A]
            Se existe JTB [A] então
            Inicio (3.1)
               MODULO-C
               Se DEFESA [P] = inexistente então EXISTE-DEFESA := falso
            Fim (3.1)
                        senão
            Início (3.2)
               Se JOGADA-DECISIVA = verdadeiro então MODULO-B
             * Se JOGADA-DECISIVA = falso então MODULO-D
             * Se JLEGAL = falso então EXISTE-DEFESA := falso
            Fim (3.2)
         Até que EXISTE-DEFESA = falso ou  não existe outra POSSÍVEL-DEFESA [A]
      Fim (2.1)
                     senão
      Inicio (2.2)
         JOGADA-DECISIVA := falso MODULO-E
         Se JLEGAL = falso então EXISTE-DEFESA := falso
                           senão EXISTE-DEFESA := verdadeiro
      Fim (2.2)
      Se EXISTE-DEFESA = verdadeiro então DEFESA [P] = POSSÍVEL-DEFESA [P]
   Até que JOGADA-DECISIVA = verdadeiro ou
   não existe outra POSSÍVEL-DEFESA [P]
Fim (1)
```

FIGURA 9 - MODULO C

São variáveis globais:

- DEFESA [P] : indica a defesa [P] a ser efetuada.
- JOGADA-DECISIVA: indica se a defesa [P] leva a uma JD [P].
- JLEGAL : indica se a jogada [P] sugerida foi ou não aprovada pelo MODULO E.

As demais variáveis são locais. A variável EXISTE-DEFESA indica a existência ou não de uma DEFESA [P] independente do fato desta levar ou não a uma JD [P].

4.2.4. Módulo D

Argumento de entrada: Configuração do momento.

Argumento de saída: Existência ou não da MELHOR JOGADA e a identificação da mesma.

A figura 10 apresenta sob a forma de português estruturado o algoritmo correspondente ao MODULO D.

Note-se que:

a). Se para a jogada [P] proposta não existir alguma JD [A] (análise realizada pelo MODULO E), a mesma será validada, dispensando-se a análise das demais jogadas [P]. Portanto quanto melhor for a função de avaliação menos jogadas [P] necessitam ser analisadas.

b). Se para toda e qualquer jogada [P] houver uma JT [A], será indicada, pela condição verdadeira na variável global booleana FRACASSO, a derrota do programa.

c). Se a variável global booleana JLEGAL for falsa após a execução do bloco REPITA da figura 10, então para toda e qualquer jogada [P] poderá existir uma possível, mas não necessária, vitória [A]. Neste caso será identificada a condição de derrota [P], que poderá ou não ocorrer dependendo da habilidade do adversário. De forma a dar continuidade ao jogo a melhor jogada [P] será identificada simplesmente como a primeira jogada [P] simulada.

d). Se após a execução do bloco REPITA da figura 10, a variável global JLEGAL for verdadeira, então a jogada [P] sugerida será validada, o que significa que até o nível de análise pré-estabelecido pela variável global QI (veja MODULO E), não se registram preocupações relativas às classes B ou F .

As variáveis JLEGAL (que identifica a existência da melhor jogada [P]), MELHOR-JOGADA [P] e FRACASSO são variáveis globais.

4.2.5. Módulo E

Argumento de entrada: Configuração do momento.

Argumento de saída: Existência ou não de uma JD [A].

```
Inicio (1)
    SIMULAÇÃO de todas as possíveis jogadas [P]
    APLICAÇÃO de uma função de avaliação às configurações obtidas
    CLASSIFICAÇÃO das jogadas (melhor jogada primeiro)
    FRACASSO := verdadeiro    JLEGAL := falso
    Inicio (2.1)
        Repita
            SIMULA MELHOR-JOGADA [P] segundo classificação
            MODULO-A [A]
            Se não existe JT [A] então
                Inicio (3.1)
                    FRACASSO := falso    JLEGAL = verdadeiro
                    MODULO-E
                Fim (3.1)
        Até que JLEGAL = verdadeiro ou não existe outra MELHOR-JOGADA [P]
        Se JLEGAL = falso e FRACASSO = falso então
            Inicio (3.2)
                Identifica-se possível vitória [A]
                MELHOR-JOGADA [P]  :=  1a. JOGADA DA CLASSIFICAÇÃO
            Fim (3.2)
    Fim (2.1)
Fim (1)
```

FIGURA 10 - MODULO D

A figura 11 apresenta sob a forma de português estruturado o algoritmo correspondente ao MODULO E.

Note-se que:

a). A variável global QI é determinante do nível de análise a ser efetuada caso não exista JD [A] após a execução do bloco REPITA A - figura 11.

b). O bloco 3.1 da figura 11 simula uma chamada recursiva do próprio programa principal bloco 4.2 da figura 6.

c). O bloco 4.1 da figura 11 será executado somente quando não existir JT [P] ou JTB [A], uma vez que, a JT [P] leva à consequente vitória [P], e a condição de JTB [A] já foi anteriormente analisada por este mesmo módulo (bloco 2.1).

d). No bloco 5.1 da figura 11 a chamada do MODULO D forçará uma chamada indireta recursiva do próprio MODULO E. É neste ponto que, sob o controle da variável global QI, é efetuada a análise até o nível desejado.

e). O processo de análise será interrompido quando existir um laço. Neste caso atribui-se o valor verdadeiro à variável JLEGAL. Assume-se a não existência de uma JD [A] de forma a induzir-se um possível empate. Este fato não é evidenciado no algoritmo da figura 11.

As variáveis QI (que estabelece o nível de análise), JLEGAL (que indica o oposto da existência ou não de JD [A]) DEFESA [P] e JOGADA-DECISIVA são variáveis globais. A variável JD [A] é local.

```
Inicio (1)
  Repita (A)
    SIMULA JOGADA [A]   MODULO-A [A]   JD [A] := falso
    Se existe JTB [A] então
    Inicio (2.1)
      MODULO-C
      Se DEFESA [P] = inexistente então JD [A] := verdadeiro
    Fim (2.1)
  Até que JD [A] = verdadeiro ou não existe outra JOGADA [A]
  JLEGAL := oposto de JD [A]
  Se JD [A] = falso então
  Inicio (2.2)
    Se QI ≠ 0 então
    Inicio (3.1)
      QI := QI - 1
      Repita (B)
        SIMULA JOGADA [A]   JD [A] := falso
        MODULO-A [A]   MODULO-A [P]
        Se não existe JT [P] e não existe JTB [A] então
        Inicio (4.1)
          MODULO-B
          Se JOGADA-DECISIVA = falso então
          Inicio (5.1)
            MODULO-D
            Se JLEGAL = falso então JD [A] := verdadeiro
          Fim (5.1)
        Fim (4.1)
      Até que JD [A] = verdadeiro ou não existe outra JOGADA [A]
      QI := QI + 1   JLEGAL := oposto de JD [A]
    Fim (3.1)
  Fim (2.2)
Fim (1)
```

FIGURA 11 - MODULO E

4.3. IMPLEMENTAÇÃO DOS MÓDULOS

O anexo A apresenta a listagem do programa que implementa o jogo "Tac-Tickle" utilizando o algoritmo PAD. Pascal foi escolhida como linguagem de programação porque é recursiva, permite uma fácil definição de tipos de variáveis, programação estruturada e a consequente modularização do programa.

Definição de Tipos e Variáveis

São definidos os seguintes tipos de variáveis:

a). Tipo POSIÇÃO [Quem, Info, Proximo]:

Caracteriza uma célula do tabuleiro (uma posição) contendo as seguintes informações:

- QUEM - identifica a peça que a ocupa (programa ou adversário.
- INFO - armazena o número de possibilidades de se agrupar 3 peças consecutivas utilizando-se a célula em questão. Assim por exemplo na posição [1,1] do tabuleiro INFO assume o valor 3 e na posição [3, 2] o valor 9.
- PROXIMO - conjunto de 8 apontadores, cada um contendo o endereço de uma das células vizinhas. A utilização destes apontadores facilita a programação, uma vez que, a partir de qualquer célula acessamos natural

mente as suas 8 células vizinhas. A distribuição dos apontadores é tal que as células vizinhas acessadas pelos apontadores 1, 2, 3 e 4 traduzem exatamente os movimentos permitidos para a peça em questão.

b). Tipo JOGADA [De, Para] :

Caracteriza, pelos identificadores DE e PARA, as células envolvidas na realização de uma jogada:

c). Tipo DECISÃO [Lin, Col, Point, Info, Sseu, Saldo] :

Armazena o resultado da aplicação da função de avaliação. Contém as seguintes informações:

- LIN, COL, POINT - Identificadores que representam a jogada considerada. Lin, Col especificam a "posição de" e Point a "posição para".
- INFO - Número de possibilidades de se agrupar 3 peças consecutivas utilizando-se por base a célula correspondente à "posição para".
- SSEU - Representa numericamente a mobilidade do adversário (número de possíveis jogadas [A]), após efetuada a jogada considerada.

- SALDO - diferença numérica entre a mobilidade do adversário e a do programa.

d). Tipo ENDEREÇO [Lin, Col, Poss, Point] :

Caracteriza, através da variável inteira POSS, o número de JTB's existentes uma vez efetuada a jogada identificada pelas variáveis LIN, COL "posição de" e POINT "posição para". Dada às características do jogo podemos garantir a existência de uma jogada terminal sempre que para o jogador em questão existirem, a um dado momento, jogadas terminais barradas cujas "posição para" forem diferentes. Assim por exemplo, a jogada [2,1] para [1,1] na figura 12 não é vitoriosa ao passo que a jogada [3,2] para [3,3] leva à vitória.

Observe-se que para o jogo de xadrez o número de JTB's (número de formas diferentes pelas quais o rei pode ser capturado), não dita a existência de uma jogada terminal.

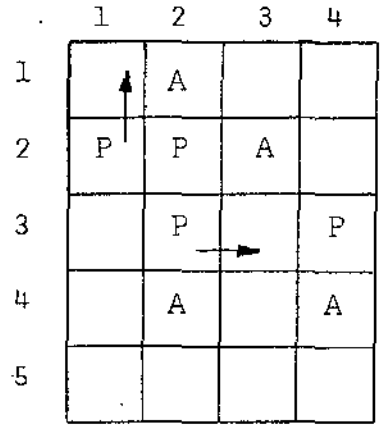


FIGURA 12

e). Tipo HISTORICO [Quem, Endi]:

Armazena uma configuração do jogo. O identificador QUEM (vetor de 8 posições) identifica o jogador (programa ou adversário) que ocupa a célula apontada pelo identificador ENDI (também vetor de 8 posições).

f). Tipo LINK

Variáveis deste tipo armazenam um endereço físico de memória correspondente à localização de uma variável do tipo POSIÇÃO. Estas variáveis são chamadas na linguagem Pascal de variáveis do tipo "Pointer", ou seja, apontadores.

As principais variáveis são:

a). TAB - variável do tipo Link que implementa o tabuleiro do jogo.

b). HISTORIA - variável do tipo Histórico. Possibilita o armazenamento de até 100 configurações de jogo distintas.

c). VETOR - variável do tipo Endereço. É utilizada na implementação dos módulos B e E. Possibilita o armazenamento de 16 jogadas distintas, número máximo de possíveis jogadas para um jogador a um dado momento.

d). LOCAL - variável do tipo Link. Armazena no máximo 4 células a partir da(s) qual(ais) existe uma possível defesa [P] uma vez caracterizada uma CT [A].

e). STACK - variável do tipo Jogada. É utilizada para armazenar as possíveis jogadas terminais, ou jogadas terminais barradas quando da implementação do MODULO A.

f). MOVE - variável do tipo Decisão. Armazena 16 jogadas distintas, número máximo de possíveis jogadas para um jogador a um dado momento. É utilizada quando da implementação do MODULO D.

g). IGUAL - variável booleana que acusa a existência de um laço (configurações repetidas).

h). REPETIÇÃO- variável booleana que armazena a ocorrência passada de um laço no processo de análise.

i). JLEGAL - variável booleana que,após análise efetuada pelo MODULO E, identifica a validade da jogada [P] proposta.

j). OK - variável booleana que,após a análise efetuada pelo MODULO B, identifica a existência de jogada decisiva [P].

k). QI - variável inteira que identifica o nível de análise a ser efetuado. Desconsiderando-se a aná

lise efetuada em função de JTB's, o nível, resultante após a identificação de uma configuração neutra, é igual à 2QI + 2.

Os tipos de variáveis Vinte, Bit e Arquivo, bem como, as variáveis Memória, Configuração, Entrada, Palavra, Bite, Máscara, Existe e Pilha serão discutidas juntamente com a implementação do processo de aprendizado no capítulo 5.

Lembramos que as variáveis mencionadas não representam a totalidade das variáveis utilizadas. Variáveis auxiliares e de trabalho tais como, Ppilha, Jogador, U, V, Sucesso, Fracasso serão naturalmente compreendidas uma vez analisado o programa.

4.4. DEFINIÇÃO DOS PROCEDIMENTOS

A implementação do algoritmo PAD (figura 6) e de seus módulos integrantes apresentará características próprias em função do jogo a ser programado. É natural que o jogo de xadrez, por exemplo, apresente um número bem maior de variáveis, um esquema de armazenamento e de determinação de JT's e JTB's diferente daquele usado na implementação do jogo "Tac-Tickle". Portanto teremos, independente do jogo a ser implementado, uma associação lógica quanto ao conteúdo, mas não necessariamente quanto à forma, entre o algoritmo PAD, tal como foi definido, e os procedimentos que implementam o jogo considerado.

Cinquenta e quatro procedimentos implementam o algoritmo PAD para o jogo "Tac-Tickle" (Anexo A).

MODULO-A - É implementado pelo procedimento DECIDE (#12) que utiliza os procedimentos auxiliares VERIFICA (#11), TRANSPECA (#10), TRANSFERE (#9), COMPLETA (#8), TRANSA (#7) e VER (#6). Em vez da forma descrita no ítem 4.2.1, o procedimento DECIDE identifica a existência de configurações terminais através de comparações com os 32 padrões que, para o jogo "Tac-Tickle", representam o conjunto possível de configurações terminais (ver Anexo B).

MODULO-B - É implementado pelo procedimento ANALISA (#19). Retorna, na variável global Ok a condição de existência de CD [P] e, nas variáveis U e V, do tipo Link, a respectiva JD [P].

Inicialmente armazenam-se na variável Vetor todas as jogadas que levam à existência de JTB's e o número de JTB's resultantes das jogadas consideradas. Neste contexto o procedimento FULMINA (#15) associa o valor verdadeiro à variável Ok se para qualquer uma das JTB's não existir possível defesa do adversário. O procedimento CONDENSA (#13) determina o número de JTB's diferentes decorrentes da jogada considerada. A variável global J indica o número de jogadas que levam à JTB's.

Terminado o primeiro bloco 2 do procedimento ANALISA (Anexo A), o procedimento ARRUMAVET (#16) classifica as jogadas de Vetor em ordem crescente do número de JTB's resultantes. Até este momento o procedimento ANALISA poderá ser

executado tendo por argumento de entrada o jogador programa ou adversário. Note-se que, no MODULO E a determinação do número de JTB's [A] é implementada através da chamada do procedimento ANALISA tendo por argumento de entrada o jogador adversário.

O segundo bloco 2 do procedimento ANALISA (Anexo A) implementa o bloco 2.1 do MODULO B (figura 8). Este será executado somente se a variável Ok for falsa, uma vez que, a condição verdadeira indica que já foi identificada uma JD [P]. A variável local T, indica o número de JTB's [A] decorrentes de uma possível defesa [A]. A chamada do MODULO C (bloco 3.1 - figura 8) é implementada pelo primeiro bloco 10 do procedimento ANALISA (Anexo A) e somente é efetuada se T = 1, uma vez que se T> 1, existe JD [A] e portanto Ok é falso. Se T = 0 não existe JTB [A] sendo o MODULO B chamado recursivamente (segundo bloco 10 do procedimento ANALISA - Anexo A).

A existência de um possível laço no processo de análise (configurações repetidas) é verificada trivialmente no início do processo. O procedimento COMPARA (#18) retorna a condição de existência do laço na variável global Igual. Se Igual assumir o valor verdadeiro o procedimento ANALISA somente indicará a condição de não existência de jogada decisiva, finalizando a seguir.

MODULO-C - Conforme foi visto no ítem 4.1, este módulo retorna a melhor defesa [P] para uma JTB [A]. É implementado pelo

procedimento SIMULA (#20). No jogo "Tac-Tickle" toda e qualquer defesa se resume em impedir que o adversário agrupe 3 de suas peças consecutivamente. Portanto defender-se implica no movimento de uma peça para uma determinada célula do tabuleiro de forma a impedir a JT por parte do adversário. A célula para a qual a peça [P] deverá ser movida, de forma a realizar-se a defesa [P], é determinada pelo MODULO A quando da identificação da JTB [A], antes mesmo da chamada do procedimento SIMULA.

O procedimento retornará através de seu argumento de entrada do tipo Link, o endereço correspondente à célula ocupada pela peça [P] que deverá ser movida de forma a efetuar-se a defesa [P]. Invariavelmente duas ações são realizadas antes da execução do procedimento SIMULA:

a). Determina-se, através do MODULO A, a célula para a qual a peça [P] deverá ser movida. Esta célula é identificada pela variável Stack [1]. para.

b). Com base em Stack [1]. para o procedimento PROCURA (#14) identifica as peças [P] passíveis de efetuar uma defesa [P]. O vetor Local armazena os endereços correspondentes à localização destas peças. Qualquer peça [P] poderá efetuar uma defesa [P] desde que ela seja vizinha da célula identificada pela variável Stack [1]. para e o seu movimento não possibilite a execução de uma JT [A].

Tão logo se inicia o procedimento SIMULA, verifica-se, por meio da variável global Plocal, a existência ou não de possíveis defesas [P]. Se Plocal for igual a zero, ou seja, não existe possível defesa [P], encerra-se o procedimento SIMULA identificando-se a não existência de defesa [P] (variável L:= Nil) e a consequente inexistência da defesa [P] que leve à JD [P] (variável Ok = falso). A existência de possível(eis) defesa(s) [P], caso a variável Plocal for diferente de zero, leva ao início propriamente dito do procedimento SIMULA.

Tal como no procedimento ANALISA (#19) verifica-se a condição de um laço no processo de análise através da chamada do procedimento COMPARA (#18). Existindo o laço identifica-se uma defesa P (L:= Local Plocal) e a inexistência de defesa [P] que leva à JD [P] (Ok:= falso). A variável global Repetição a rigor é dispensável (ver Análise, Conclusões e Recomendações - capítulo 7).

As variáveis Defesa [P], Jogada-decisiva, Existe-defesa, mencionadas no MODULO C - figura 9, são implementadas respectivamente pelas variáveis Qual, Ok e Da. As variáveis Decisiva e Posição foram localmente utilizadas em substituição às variáveis globais Ok e Qual visando única e exclusivamente facilitar eventuais processos de depuração do procedimento.

No segundo bloco 3 do procedimento SIMULA - (Anexo A), classificam-se as possíveis defesas [P] em ordem cres

cente do número de JTB's [P] resultantes. Armazena-se no vetor local Loc o endereço da peça [P] e no vetor, também local, Info, o correspondente número de JTB's [P] resultantes após o movimento da peça [P] da posição Loc para a posição Stack [1]. para.

O terceiro bloco 3 do procedimento SIMULA - (Anexo A) implementa o bloco Repita A do MODULO C (figura 9). As variáveis locais I e S representam correspondentemente a existência de Possível-defesa [P] e Possível-defesa [A]. Dentro deste bloco, o primeiro bloco 4 implementa o bloco 2.1 do MODULÓ C (figura 9) (existe JTB [P] se Info (i) = 1). No segundo bloco 4, verifica-se inicialmente se o número de JTB's [P] resultantes é maior do que 1 (Info (i) > 1). Se este for o caso identifica-se na variável Da a existência de defesa [P]. Neste momento a variável Decisiva assume o valor verdadeiro, atribuido ao iniciar-se o terceiro bloco 3 e consequentemente, encerra-se o processo repetitivo - terceiro bloco 3 - identificando-se a existência de defesa [P] que leva à JD [P]. Ainda no segundo bloco 4, se o número de JTB's [P] resultantes não for maior do que 1, então, não existirá JTB [P] e o bloco 5 implementa o bloco 2.2 do MODULO C (figura 9).

O bloco Repita B do MODULO C (figura 9) é implementado pelo primeiro bloco 5 dentro do terceiro bloco 3 do procedimento SIMULA (Anexo A). Na variável local T, armazena-se o número de JTB's [A] resultantes de uma possível defesa [A].

Note-se que se T > 1 o processo de simulação de possíveis defesas [A] é interrompido, uma vez que, identificou-se uma defesa [A] que possibilita uma JD[A]. Neste caso invalida-se a possível defesa [P] simulada (Da:= falso) retornando-se ao início do terceiro bloco 3 do procedimento SIMULA (Anexo A). Se T = 1, ou seja, se a defesa [A] simulada leva à somente uma JTB[A] resultante, armazena-se no vetor local Vet a defesa [A] considerada. Na variável local M é armazenado o número de defesas [A] que se encontram nestas condições. Posteriormente o quarto bloco 5 do procedimento SIMULA (Anexo A) implementa o bloco 3.1 do MODULO C (figura 9). Se T = 0, ou seja, a defesa [A] simulada não leva à JTB[A], é realizada a chamada ao MODULO B de forma a verificar-se a existência de JD[P]. Este processo é implementado pelo segundo bloco 9 do procedimento SIMULA (Anexo A). As instruções marcadas com asterisco "*" na figura 9 foram introduzidas após a implementação do jogo "Tac-Tickle" não se encontrando implementadas no procedimento SIMULA.

MODULO-D - É implementado pelos procedimentos SALDO (#25), ORDENA (#24) e ESCOLHEJOGADA (#26). O procedimento SALDO armazena no vetor global Move, do tipo Decisão, as possíveis jogadas [P], bem como o valor resultante da aplicação da função de avaliação à cada uma das configurações resultantes. O procedimento ORDENA, ordena as jogadas [P] avaliadas pelo procedimento SALDO em ordem crescente pelo campos Saldo, Sseu, Info.

O procedimento ESCOLHEJOGADA implementa o bloco 2.1 do MODULO D (figura 10). Nas variáveis globais U e V retorna-se a melhor jogada [P]. Note-se que a possível existência de JT [A] já foi anteriormente identificada pelo procedimento SALDO. Se o valor da variável Pmove for igual a zero, toda e qualquer jogada [P] possibilita uma JT [A]. Neste caso retorna-se na variável U o valor "NIL", identificando-se a derrota [P]. A variável global Fracasso é, na implementação, externa ao MODULO D, sendo-lhe atribuído o valor de verdadeiro ou falso de acordo com a existência ou não de um endereço na variável U.

O bloco Repita do MODULO D (figura 10) é implementado pelo segundo bloco 2 do procedimento ESCOLHEJOGADA (Anexo A). A variável local I identifica a existência da próxima melhor jogada [P].

O quarto bloco 2 do procedimento ESCOLHEJOGADA implementa o bloco 3.2 da figura 10.

MODULO-E - É implementado pelo procedimento LEGAL (#21). Tal como nos procedimentos ANALISA e SIMULA a existência de um laço no processo de análise é identificada ao iniciar-se o procedimento, através da chamada do procedimento COMPARA (#18). Havendo o laço, supõe-se a não existência de JD [A], retornando-se à condição verdadeira a variável global Jlegal.

A determinação das jogadas [A] que levam à JTB [A] é feita através da chamada do procedimento ANALISA; com argu-

mento de entrada o jogador adversário.

Os blocos 2.1, 2.2 e 3.1 do MODULO E da figura 11 são respectivamente implementados pelo segundo bloco 4, primeiro bloco 4 do procedimento LEGAL e pelo procedimento PENSA (#27) (ver Anexo A).

Os demais procedimentos utilizados para a implementação do jogo "Tac-Tickle", com exceção daqueles que implementam o processo de aprendizado, caracterizam-se como procedimentos auxiliares. Entre estes destacam-se:

a). FIRSTPOINTER (#3), FIRSTQUADRO (#4) que inicializam a estrutura de dados utilizada.

b). JOGA (#5) que efetua a jogada solicitada.

c). GUARDA (#17) - armazena no vetor História, a configuração do momento.

d). TESTAJOGADA (#28) - verifica se a jogada [A] é válida em conformidade com as regras do jogo.

e). GRAVA (#2), CONTROLE-TELA (#29), APAGA-CANTO (#30), NUMERO (#31), SIM-NÃO (#32), LEJOGADA (#33), ONDE-JOGUEI (#34), DA-UM-TEMPO (#35), APAGA (#36), MOSTRA-QUADRO-VT (#37), MOSTRA-PEÇA-VT (#38), MOSTRA (#39) e MOSTRA-JOGO (#40) são procedimentos que tratam as entradas/saídas.

f). FAZ (#41) - possibilita iniciar-se o jogo de qualquer configuração apresentada.

4.5. IMPLEMENTAÇÃO DO PROGRAMA PRINCIPAL

Com exceção do processo de inicialização (estrutura de dados, variáveis, etc) o programa principal listado no Anexo A, segue criteriosamente o algoritmo PAD retratado na figura 6. A figura 13 ilustra a associação entre os blocos da figura 6 e os do programa principal constante no Anexo A.

BLOCOS FIGURA 6	BLOCOS PROG-PRINCIPAL ANEXO A
BLOCO 2.1	SEGUNDO BLOCO 3
BLOCO 2.2	PRIMEIRO BLOCO 3
BLOCO 3.1	PRIMEIRO BLOCO 4
BLOCO 3.2	SEGUNDO BLOCO 4
BLOCO 4.1	PRIMEIRO BLOCO 5
BLOCO 4.2	SEGUNDO BLOCO 5
BLOCO 5.1	TERCEIRO BLOCO 6

FIGURA 13 - Relação entre blocos

Note-se que a condição de existência de CT é dada pela variável Pstack. Se Pstack for diferente de zero existirão, JT's ou JTB's e consequentemente existirá uma CT.

5. IMPLEMENTAÇÃO DO APRENDIZADO

O processo de aprendizado associado ao algoritmo PAD consiste no armazenamento das configurações que indicam uma derrota iminente para o programa, Para tanto é preciso definir COMO armazenar e QUANDO armazenar/consultar estas configurações, de forma a viabilizar-se um rápido acesso às mesmas e otimizar-se o espaço físico necessário para o seu armazenamento. O armazenamento/recuperação destas configurações, possibilitam quando da escolha da melhor jogada, a identificação das jogadas [P] que comprovadamente levam a uma possível derrota [P̄], tornando-se a análise destas desnecessária. O processo de seleção de jogadas será, portanto, função de experiência acumulada pelo programa.

5.1. FORMAS ALTERNATIVAS DE ARMAZENAMENTO

Por analogia aos conceitos de memória de curta e longa duração define-se que MCD (Mémoria de Curta Duração) será a forma de armazenamento que não introduz um valor probabilístico de recuperação e que MLD (Memória de Longa Duração) será a forma de armazenamento que introduz um valor probabilístico de recuperação.

No que se refere à MCD e considerando-se o jogo "Tac-Tickle", podemos associar a cada uma das 20 células do tabu

leiro (5 x 4) uma das unidades de um sistema de base 20. Neste contexto qualquer configuração de jogo pode ser armazenada como um par ordenado (x, y), onde X representa em ordem crescente a colocação das peças do programa e Y, também em ordem crescente, a colocação das peças do adversário. O armazenamento de uma configuração demanda então o espaço físico de 8 unidades (8 caracteres). Assim por exemplo, a configuração da figura 14 terá a ela associada o par (FHMP, BDIL).

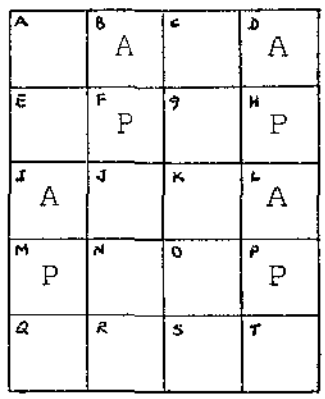


FIGURA 14 - CONFIGURAÇÃO (FHMP, BDIL)

Qualquer forma de armazenamento que demande, por configuração armazenada, menos de 8 unidades, forçosamente introduzirá um valor probabilístico quando da sua recuperação (verificar existência ou não da configuração). Foram consideradas as seguintes formas alternativas de armazenamento MLD:

a). Defini-se uma Matriz (20 x 20 x 20 x 20) de bit's. Podemos armazenar uma configuração de perda, por exemplo (FHMP, BDIL), assinalando-se os bit's referentes às posi

ções da Matriz (FHMP) e Matriz (LIDB). Note-se que ao invés de utilizarmos 2 matrizes (20 x 20 x 20 x 20), podemos, aproveitando-se o fato de cada uma delas ser triangular superior (as coordenadas X e Y são apresentadas em ordem crescente de suas unidades), utilizar somente uma, invertendo-se a ordem de apresentação da coordenada Y. Note-se ainda que, do ponto de vista probabilístico, se tivermos armazenadas as configurações (x_1, y_1) e (x_2, y_2), então identificaremos também como configurações de perda, as configurações (x_1, y_2) e (x_2, y_1) desde que sejam vazios os conjuntos de intersecção X_1, Y_2 e X_2, Y_1. Não se sabe qual será a probabilidade de serem realmente de perda as configurações (x_1, y_2) e (x_2, y_1) levando-se em consideração que X_1 e X_2 desfavorecem o programa, e, Y_1 e Y_2 favorecem o adversário.

b). Definem-se 7 matrizes de bit's:

P_1 (20 x 7 x 20) = 2.800 bit's

P_2 (20 x 6 x 20) = 2.400 bit's

P_3 (20 x 5 x 20) = 2.000 bit's

P_4 (20 x 4 x 20) = 1.600 bit's

P_5 (20 x 3 x 20) = 1.200 bit's

P_6 (20 x 2 x 20) = 800 bit's

P_7 (20 x 20) = 400 bit's

Quando da existência de uma configuração de perda, por exemplo (FHMP, BDIL), assinalamos os bit's referentes às combinações da Tabela I.

P_1 (F,1,H), P_1 (F,2,M), P_1 (F,3,P), P_1 (F,4,B), P_1 (F,5,D) P_1 (F,6,I), P_1 (F,7,L).

P_2 (H,1,M), P_2 (H,2,P), P_2 (H,3,B), P_2 (H,4,D), P_2 (H,5,I) P_2 (H,6,L).

P_3 (M,1,P), P_3 (M,2,B), P_3 (M,3,D), P_3 (M,4,I), P_3 (M,5,L).

P_4 (P,1,B), P_4 (P,2,D), P_4 (P,3,I), P_4 (P,4,L).

P_5 (B,1,D), P_5 (B,2,I), P_5 (B,3,L).

P_6 (D,1,L), P_6 (D,2,L).

P_7 (I,L).

TABELA I - RELAÇÕES REPRESENTATIVAS DA CONFIGURAÇÃO (FHMP, BDIL)

É claro que este tipo de armazenamento é MLD, ou seja, traz consigo os efeitos de uma recuperação probabilística. A probabilidade de ocorrência de um verdadeiro-falso (ver capítulo 3), cresce à medida em que novas configurações de perda forem sendo armazenadas. Quando todos os 11.200 bit's estiverem assinalados qualquer configuração será identificada como configuração de perda, dizemos que a memória atingiu o seu ponto de saturação. A probabilidade de ocorrência de um verdadeiro-falso é portanto função de distribuição de probabilidade do número de configurações armazenadas.

c). Define-se uma Matriz (20 x 8) de inteiros. Cada das posições desta matriz indicará o número de vezes que a célula k (k = 1, ... 20) ocorreu na posição z (z = 1,... 8) quando do armazenamento das configurações de perda. Assim por exemplo a identificação da configuração de perda (FHMP, BDIL) levará à soma de uma unidade às posições:

(F,1), (H,2), (M,3), (P,4), (B,5), (D,6), (I,7), (L,8).

Ao contrário das alternativas a e b de armazenamento MLD, que dispensam a utilização de uma MCD e de critérios específicos de interpretação, deveremos para a alternatica C:

1). Utilizar uma MCD na qual seriam armazenadas sempre as mais recentes configurações de perda (o número de configurações à armazenar depende única e exclusivamente do tamanho da MCD).

2). Definir um procedimento que mediante consulta à MLD, atribua à configuração do momento, um valor correspondente à probabilidade da mesma ser de perda.

3). Definir um limiar probabilístico a partir do qual uma dada configuração deverá ser considerada de perda.

A figura 15 ilustra o algoritmo proposto quando da recuperação de uma configuração de perda utilizando-se a alternativa c.

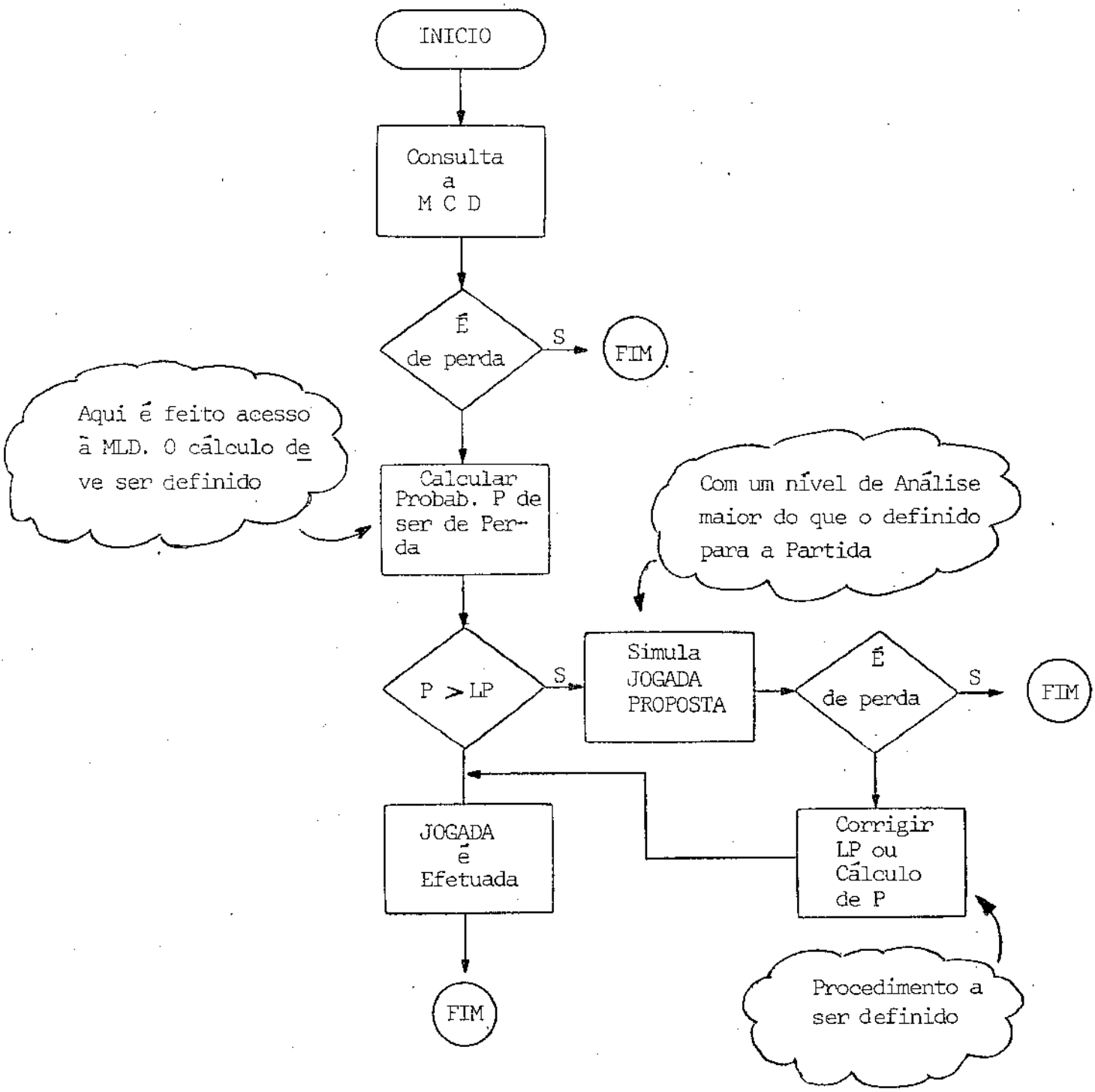

INICIO
Consulta a M C D
É de perda
S
FIM
Aqui é feito acesso à MLD. O cálculo deve ser definido
Calcular Probab. P de ser de Perda
Com um nível de Análise maior do que o definido para a Partida
P > LP
S
Simula JOGADA PROPOSTA
É de perda
S
FIM
JOGADA é Efetuada
Corrigir LP ou Cálculo de P
FIM
Procedimento a ser definido

FIGURA 15

5.2. ESQUEMA DE ARMAZENAMENTO ESCOLHIDO

Escolheu-se a alternativa b para o armazenamento MLD das configurações de perda, dada a facilidade de implementação, aos diminutos requisitos de memória necessária, bem como, à relativa baixa probabilidade de ocorrência de um verdadeiro-falso (VF - ver capítulo 3).

No que se refere a ocorrência de um verdadeiro-falso note-se que:

a). Pelo menos 3 configurações de perda distintas devem ser armazenadas para que a probabilidade de ocorrência de um verdadeiro-falso se torne diferente de zero. Assim, por exemplo, para recuperarmos a configuração (FHMP, BDIL) como de perda, sem que esta jamais tenha sido armazenada como tal, teremos que armazenar, por exemplo, as configurações (FHMP, BDIR), (FHMP, BDEL) e (ACFG, BDIL). Dos 28 bit's que traduzem a existência da configuração (FHMP, BDIL) somente 7 não foram assinalados quando do armazenamento da configuração (FHMP, BDIR):

P_1 (F,7,L), P_2 (H,6,L), P_3 (M,5,L), P_4 (P,4,L), P_5 (B,3,L) P_6 (D,2,L), P_7 (I,L).

O armazenamento da configuração (FHMP, BDEL) assinala, com exceção do bit P_7 (I,L), os restantes 6 bit's ainda não assinalados. Finalmente a configuração (ACFG, BDIL) assinala o bit faltante P_7 (I,L).

b). Do ponto de vista probabilístico o esquema de armazenamento representa um processo estocástico (existe uma população de bit's assinalados cujo crescimento é função do número de jogadas armazenadas). Os parâmetros deste processo são determináveis somente através de simulação.

c). Podemos definir a probabilidade de ocorrência de um verdadeiro-falso em função da cardinalidade da população de bit's assinalados imediatamente após o armazenamento da n-ésima configuração de perda como:

$$P_{VF}(n) = \frac{n(n-1)(n-2)\ldots(n-27)}{T(T-1)(T-2)\ldots(T-27)}.$$

Onde:

T = 11.200

n = cardinalidade do conjunto de bit's assinalados

d). Admitindo-se que cada configuração armazenada assinala um novo conjunto de 28 bit's, alcançaremos, no pior caso, o ponto de saturação após o armazenamento de 400 configurações de perda.

A figura 16 ilustra o gráfico representativo da função de distribuição probabilística do processo apresentado.

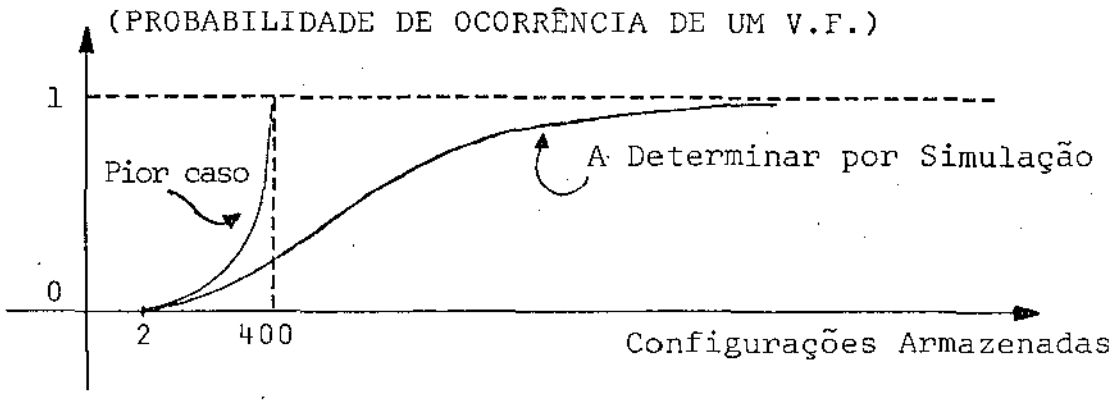


FIGURA 16 - FUNÇÃO DE DISTRIBUIÇÃO PROBABILÍSTICA

5.3. IMPLEMENTAÇÃO DO ESQUEMA PROPOSTO

No ítem anterior foi escolhida a forma de COMO armazenar. A questão de QUANDO adquirir ou utilizar o conhecimento decorrente do processo de armazenagem está ligada à concepção do algoritmo PAD. Os pontos de obtenção e utilização deste conhecimento são discutidos nos ítens 5.3.1. e 5.3.2. Para o melhor entendimento destes ítens descrevemos inicialmente a estrutura de dados e os procedimentos que implementam o processo de aprendizado.

1. Estrutura de Dados

A implementação lógica das sete matrizes pode ser feita através de uma única matriz M(20,28,20). Onze mil e duzentos bit's podem implementá-la fisicamente, uma vez que, cada uma das suas 11.200 posições representam situações biná

rias. Considerando-se que um número inteiro ocupa uma palavra de 36 bit's (Decsystem-10 Digital), podemos implementar a memória necessária para o processo de aprendizado através do vetor Memória [312] de Inteiros.

2. Procedimentos

Quatro procedimentos básicos viabilizam o tratamento a nível de bit, possibilitando a implementação do processo:

a). Procedimento CONFIGURA (# 46)

Entrada: Configuração do momento

Saída: Vetor Configuração contendo o par ordenado (X,Y), onde X representa em ordem crescente a colocação das quatro peças do programa, e Y, também em ordem crescente a colocação das peças do adversário (veja figura 17).

A	B A	C	D A
E	F P	G	H
I A	J	K P	L A
M P	N	O	P P
Q	R	S	T

Configuração [1] = 6 ⇒ posição F
Configuração [2] = 11
Configuração [3] = 13
Configuração [4] = 16

Configuração [5] = 2 ⇒ posição A
Configuração [6] = 4
Configuração [7] = 9
Configuração [8] = 12 ⇒ posição L

CONFIGURAÇÃO (FKMP,BDIL)

FIGURA 17 - VETOR CONFIGURAÇÃO APÓS PROCEDIMENTO CONFIGURA

b). Procedimento PALABITE (# 47)

Entrada: Qualquer uma das 28 relações representativas da configuração do momento (análogas à TABELA I). Para tanto são fornecidos os 3 índices da matriz M que traduzem a relação (veja figura 18).

Saída: Indicação do bit físico correspondente à relação fornecida. O bit é indicado através das variáveis Palavra (índice no vetor Memória) e Bite (bit a ser considerado na posição Memória [Palavra]). Assim por exemplo para a relação M (6,5,4) temos Palavra = 81 e Bite = 4.

M (6,1,11) M (6,2,13) M (6,3,16) M (6,4,2) M (6,5,4) M (6,6,9) M (6,7,12)

M (11,8,13) M (11,9,16) M (11,10,2) M (11,11,4) M (11,12,9) M (11,13,12)

M (13,14,16) M (13,15,2) M (13,16,4) M (13,17,9) M (13,18,12)

M (16,19,2) M (16,20,4) M (16,21,9) M (16,22,12)

M (2,23,4) M (2,24,9) M (2,25,12)

M (4,26,9) M (4,27,12)

M (9,28,12)

FIGURA 18 - RELAÇÕES REPRESENTATIVAS DA CONFIGURAÇÃO (FKMP, BDIL)

c). Procedimento DECODIFICA (# 48)

Entrada: Variável Palavra

Saída: Valor binário (complemento de 2) equivalente ao conteúdo da posição inteira Memória [Palavra]. Este valor é armazenado no vetor Máscara.

d). Procedimento CODIFICA (# 49)

Entrada: Qualquer uma das 28 relações representativas da configuração do momento. São fornecidas as variáveis Palavra e Bite.

Saída: É assinalado o bit físico correspondente à relação fornecida. Inicialmente obtem-se o valor binário correspondente ao conteúdo da localização de Memória [Palavra]. (utiliza-se o procedimento DECODIFICA). A seguir assinala-se no vetor Máscara o bit solicitado. Caso o mesmo tenha sido alterado (note-se que o mesmo já poderia estar assinalado), codifica-se o valor binário, então obtido, armazenando-se o correspondente número inteiro em Memória [Palavra].

Os demais procedimentos que implementam o processo de aprendizado, com excessão do procedimento SUCESSIVA (# 51) que implementa o processo de "RETORNO" (veja ítem 5.4) são os seguintes:

- LOAD (# 44) - Carrega no vetor Memória o conteúdo do arquivo externo (residente em disco) MEMORI.DAT. O arquivo MEMORI.DAT contém as configurações de perda até então armazenadas.

- STORE (# 45) - Restaura o arquivo externo MEMORI.DAT com o conteúdo do vetor Memória.

- GRAVCONF (# 52) - Inclui uma nova configuração de perda no vetor Memória.

- VERCONF (# 53) - Tem por entrada a configuração do momento e verifica se a mesma está armazenada no vetor Memória. Retorna na variável boolea na global Existe a condição de existência de uma configuração de perda. Existe con figuração de perda caso a configuração do momento esteja armazenada no vetor Memoria.

- MENSAGEM (# 50) - Este procedimento indica a existência de uma estratégia de ganho.

Identificados os procedimentos que viabilizam a implementação do processo de aprendizado, ou seja, viabilizam o armazenamento e a identificação de uma configuração de perda devemo-nos questionar:

a). O que exatamente e quando armazenar (no vetor Memória) uma Configuração de Perda.

b). Quando identificar (mediante consulta ao vetor Memória) a existência de uma Configuração de perda.

5.3.1. O Que e Quando Armazenar

Sempre que mediante o algoritmo PAD for identificada a existência de uma JD [A] a correspondente configuração de perda deve ser armazenada. Entende-se por correspondente CP, àquela obtida desfazendo-se a última jogada realizada pelo adversário. Note-se que neste momento o programa sempre tem o mando do jogo.

Como a determinação da existência de JD [A] é feita única e exclusivamente pelo MODULO-E torna-se fácil a determinação do local ou do momento exato em que a CP dve ser armazenada. Assim o procedimento GRAVCONF deve ser chamado imediatamente após a chamada do procedimento ESCOLHEJOGADA sempre que a variável JLEGAL assumir o valor falso. (programa principal terceiro - Bloco 6 - Anexo A). Uma vez que a identificação de perda realizada pelo procedimento ESCOLHEJOGADA, é sempre anterior ao processo de defesa [P] implementado pelo procedimento SIMULA, as CP's identificadas pelo procedimento SIMULA, são chamadas de Configurações Consequentes. Dada a sua determinação algorítmica estas configurações não necessitam ser armazenadas (veja também figura 5).

5.3.2. Quando Identificar

Antes da validação de qualquer jogada [P], o vetor Memória deve ser consultado de forma a verificar-se, se a configu

ração resultante (após efetuada a jogada [P]) é ou não uma CP. Caso afirmativo a jogada [P] deve ser invalidada poupando se o esforço de análise necessário para a validação algorítmica da mesma.

No algoritmo PAD o procedimento LEGAL (# 21) implementa a validação de qualquer jogada [P], verificando, se a mesma possibilita a execução de uma JD [A]. Portanto a identificação de uma CP deve ser feita antes desta validação dentro do procedimento LEGAL. A chamada do procedimento VERCONF no início do procedimento LEGAL (Anexo-A) implementa este processo.

5.4. PROCESSO DE RETORNO

Admitindo-se, quando da escolha da melhor jogada, a eficácia da função de avaliação e verificando-se posteriormente que esta "melhor jogada" leva o programa a uma CP, deveríamos também supor que as demais possíveis jogadas P também levam à CP's. Se testarmos a validade desta assertiva podemos, em caso afirmativo, efetuar um "retorno" na árvore de derivações das jogadas. Desfazendo a penúltima jogada efetuada pelo adversário, podemos armazenar a configuração assim obtida como CP. O processo de retorno é implementado pelos procedimentos SUCESSIVA (# 51) e GRAVCONF (# 52). Utiliza-se o vetor PILHA, onde são armazenadas as jogadas efetuadas no decorrer da partida, para efetuar o retorno das jogadas realizadas.

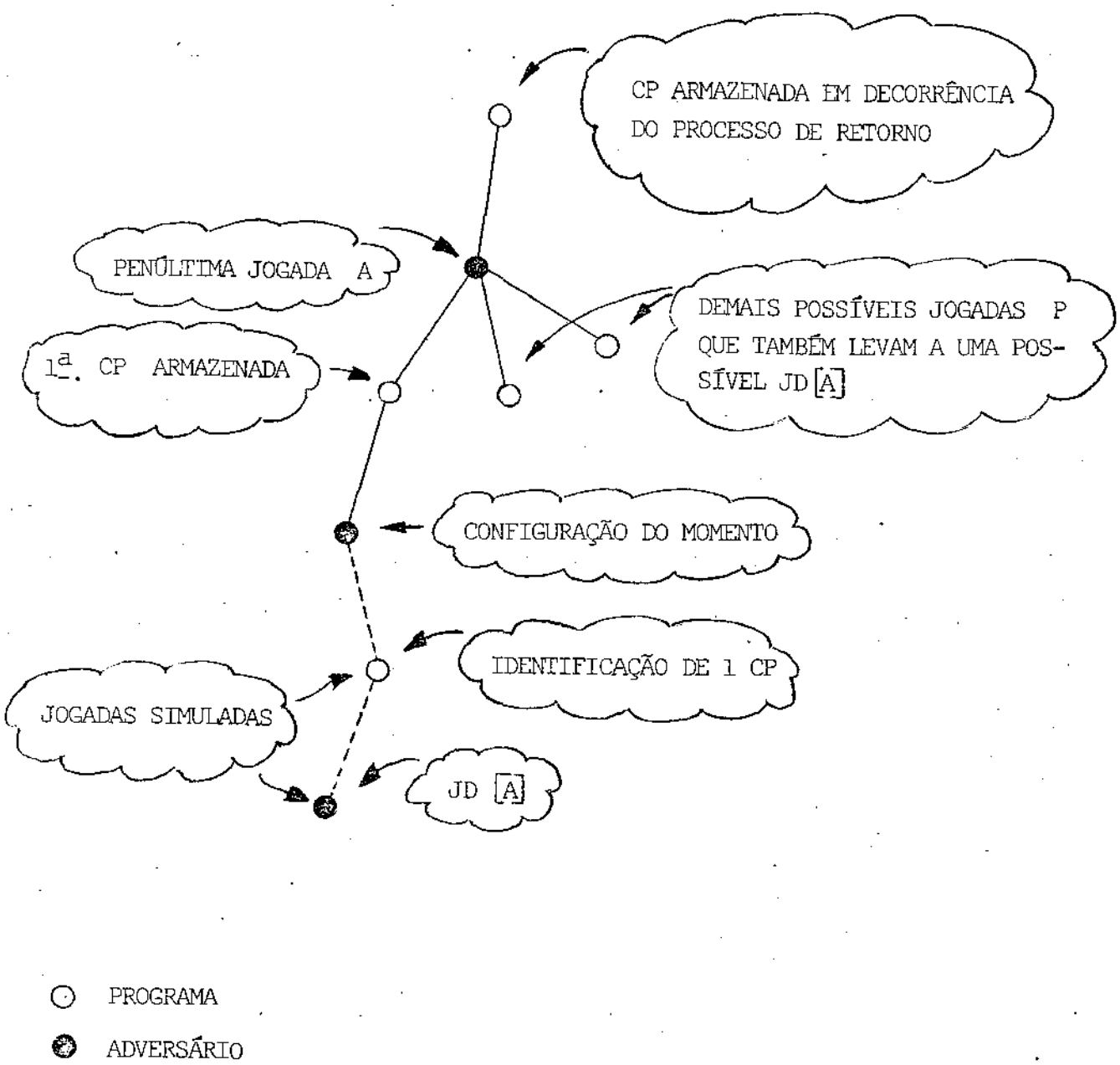


FIGURA 19 - PROCESSO DE RETORNO

6. EXEMPLOS E SUAS DISCUSSÕES

Para exemplificar a atuação do algoritmo PAD e do programa de aprendizado são discutidas, neste capítulo, seis partidas realizadas contra o programa que implementa, para o jogo "Tac-Tickle", o referido algoritmo e o processo de aprendizado (ver listagem no Anexo-A).

As quatro primeiras partidas ilustram o processo de aprendizado e as duas restantes comparam a utilização de níveis de análise diferentes (valores diferentes atribuídos à variável Qi). No Anexo C encontram-se as árvores de derivações correspondentes às jogadas das 3 primeiras partidas; as listagens produzidas (entradas/saídas) quando da realização das partidas 2 e 4; e a sequência de jogadas relativas às partidas 5 e 6.

Segue-se a análise das partidas apresentas no Anexo C. O acompanhamento das jogadas em um tabuleiro com peças identificando o programa e o adversário facilitará a sua compreensão.

6.1. ANÁLISE DA PRIMEIRA PARTIDA

Quando da realização desta partida o arquivo utilizado para o armazenamento das configurações de perda (MEMORI.DAT) estava vazio, ou seja, nenhuma experiência havia sido, até

então, acumulada pelo programa. A partida foi realizada atribuindo-se à variável Qi o valor zero, sendo portanto, efetuada uma análise de nível 2 (2 Qi + 2).

6.1.1. Escolha da Jogada [P] (20, 16)

O programa iniciou o jogo realizando a jogada [P] (20-16) e identificou a sua derrota no décimo sexto lance (veja Anexo C). A figura 20 retrata a chamada, pelo programa principal, dos principais procedimentos, bem como o valor atribuído às principais variáveis, quando da escolha da jogada [P] (20-16) (os parênteses indicam o início/término de cada procedimento).

(DECIDE [P] → Pstack:= 0) (DECIDE [A] → Pstack:= 0)
(ANALISA [P] → Ok:= falso) (ESCOLHEJOGADA (LEGAL (20, 16)
(VERCONF (20, 16) → Existe:= falso) (ANALISA [A] → Ok:= Falso)
JLEGAL:= verdadeiro))

FIGURA 20 - CHAMADA DOS PRINCIPAIS PROCEDIMENTOS PARA ESCOLHA DA JOGADA [P] (20, 16)

Conforme ilustra a figura 20, a escolha da jogada [P] (20, 16) foi possibilitada pela seguinte análise:

a). Inicialmente o procedimento DECIDE (# 12), retornando o valor zero na variável Pstack, indicou a inexistência, respectivamente de CT [P] e CT [A] (preocupações de CLASSES A e B veja ítem 4.1).

b). A seguir o procedimento ANALISA (# 19) identificou a inexistência de JD [P], fato este retratado pelo valor falso atribuído à variável Ok (preocupação de CLASSE D - segundo bloco 5 do programa principal - Anexo A).

c). Não existindo JD [P] é realizada a escolha da melhor jogada (preocupação de CLASSE E). Inicialmente são acionados os procedimentos SALDO (# 25) e ORDENA (# 24) que constroem o vetor Move contendo a avaliação realizada pela função de avaliação (figura 21).

POSSÍVEIS JOGADAS P	SALDO	SSEU	INFO
20,16	1	5	4
1,5	1	5	4
18,14	1	6	7
3,7	1	6	7

FIGURA 21 - Vetor (Pilha) Move após Aplicação da Função de Avaliação

A seguir é chamado o procedimento ESCOLHEJOGADA (# 26) que submete a melhor jogada [P], segundo avaliação realizada pela função avaliação, ao procedimento LEGAL (# 21). Este verifica se a jogada [P], uma vez efetuada, possibilita a execução de uma JD [A] (preocupação de CLASSE F). Ao iniciar-se, o procedimento LEGAL aciona o procedimento VERCONF (# 53) que, consultando o arquivo MEMORI.DAT, verifi

ca se a configuração obtida, com a execução da jogada [P] (20, 16), caracteriza ou não uma configuração de perda. Note-se que a identificação de uma configuração de perda elimina a necessidade da análise de possível JD[A]. Como a configuração resultante da jogada [P] (20, 16) não é uma configuração de perda, o procedimento LEGAL verifica a possibilidade de uma JD[A] através da chamada do procedimento ANALISA (# 19) tendo por argumento de entrada o jogador "adversário". A não existência de JD[A], retratada pelo valor falso atribuído à variável Ok, leva o procedimento LEGAL à atribuir o valor verdadeiro à variável Jlegal validando assim a jogada [P] lhe submetida.

6.1.2. Com Respeito ao Aprendizado

Uma vez realizada a jogada [A] (11, 15), o programa verifica que nenhuma de suas possíveis jogadas submetidas pelo procedimento ESCOLHEJOGADA ao procedimento LEGAL, é validada. Neste momento o programa identifica a sua possível derrota, uma vez que, independente da jogada [P] que possa ser realizada, existe sempre uma possível JD[A]. A correspondente configuração de perda, obtida retornando-se a última jogada [A] (11, 15), é armazenada no arquivo MEMORI.DAT. Note-se que quando da realização da jogada [P] anterior (12, 16) a inexistência de CT[A], após a simulação da jogada [A] (11, 15), em resposta à jogada [P] (12, 16), ditou a inexistência de

JD [A] dentro do escopo de uma análise de nível 2 (Qi = 0). Neste momento a jogada [P] (12, 16) era portanto validada e executada. Uma vez armazenada a configuração de perda (configuração número 15) inicia-se o processo de retorno, retornando-se a jogada [P] (12, 16) - procedimento SUCESSIVA (# 51). Verifica-se então que, a partir da configuração assim obtida (configuração número 14), as duas únicas possíveis jogadas [P] ,quais sejam, (12, 8) e (12, 6) possibilitam a execução de JD [A]. A segunda configuração de perda é identificada e armazenada retornando-se a jogada [A] (2, 6) (configuração número 13). O processo de retorno é repetido por mais duas vezes e como resultado armazenam-se as configurações de número 11 e 9 como configuração de perda. Note-se que a derrota [P] verificada na configurações de número 16 foi portanto, ocasionada (segundo análise realizada pelo processo de retorno) pela jogada [P] (5, 6).

6.1.3. Com Respeito a Função de Avaliação

Note-se que nem sempre as jogadas, tidas como melhores pela função de avaliação, são validadas pelo procedimento LEGAL. Este é, por exemplo, o caso das jogadas [P] (3, 7) e (1, 5) após a configuração de número 4. Este fato caracteriza uma deficiência na função de avaliação.

6.2. ANÁLISE DA SEGUNDA PARTIDA

Esta partida foi realizada tendo por entrada o arquivo MEMORI.DAT contendo as 4 configurações de perda armazenadas quando da realização da primeira partida, como nesta foi realizada uma análise de nível 2 (variável Qi = 0). Para melhor ilustrar o processo de aprendizado foram repetidas as jogadas [A] realizadas na partida anterior. Após a realização da jogada [A] (15, 11), dando origem à configuração de número 8 (veja Anexo C), o programa escolheu a jogada [P] (14, 10), uma vez que a configuração resultante da jogada [P] (5, 6), jogada [P] escolhida na primeira partida, foi identificada como configuração de perda. A jogada [P] (14, 10) foi validada pelo procedimento LEGAL, por não levar a uma CP. Note-se que esta jogada na realidade leva a uma CP não identificável em uma análise de nível 2, por não haver possível jogada [A], em resposta à jogada [P] (14, 10), que crie a condição de CT [A]. Se o valor da variável Qi fosse igual a 1, a jogada [P] (14, 10) seria invalidada pelo procedimento LEGAL, uma vez que, na análise de nível 4 (2Qi + 2), as duas únicas possíveis jogadas [P] (5, 1) e (5, 6), em resposta à jogada [A] (17, 13), possibilitam a execução de JD [A] (para a jogada [P] (5, 1) existe a JD [A] (13, 14) e a jogada [P] (5, 6) leva a uma CP já identificada - 3ª CP armazenada na primeira partida - veja figura 22).

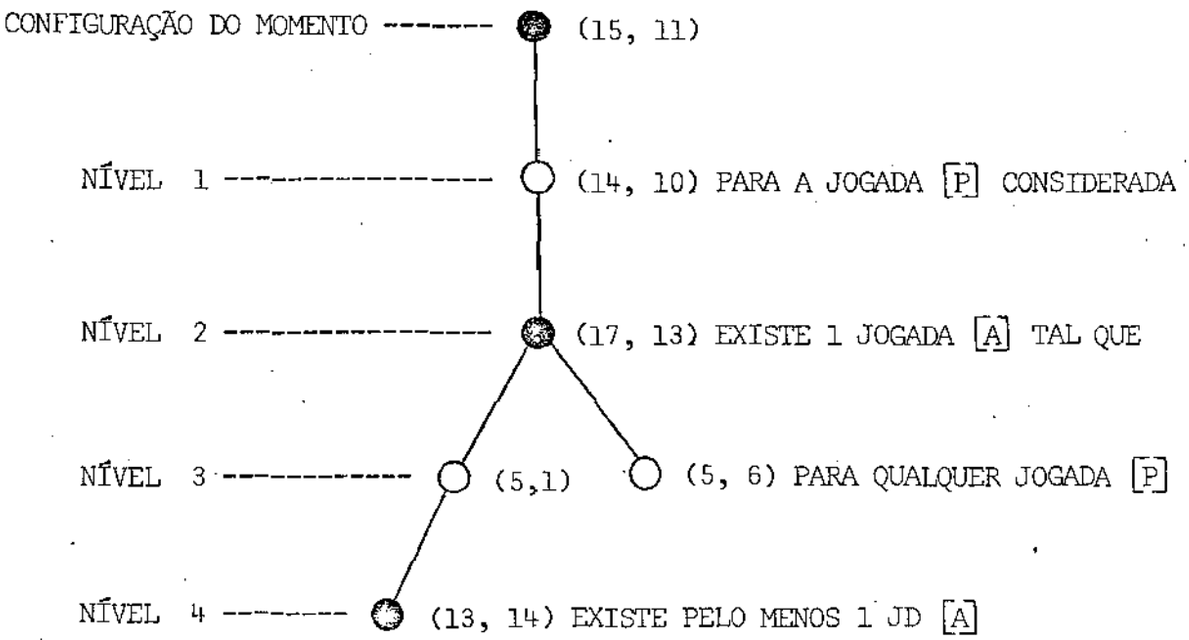


FIGURA 22 - ANÁLISE DE NÍVEL 4 (Qi = 1)

Realizada a jogada [P] (14, 10) tendo por resposta a jogada [A] (17, 13), o programa identifica a sua possível derrota e armazena a configuração de número 9 como CP. Em função do processo de retorno são ainda armazenadas como CP's as configurações de números 7 e 5. De forma a dar continuidade ao jogo o programa efetua a jogada [P] (5, 6).

As configurações de números 13 e 16 são configurações consequentes, uma vez que, possibilitam a exécução de uma JD [A] que pode ser identificada algoritmicamente, ou seja, pela aplicação do algoritmo PAD sem a necessidade de recorrer-se

ao processo de aprendizado. Por analogia as configurações obtidas com as jogadas [P] (3,7) e (1,5), a partir da configuração de número 4, também podem ser chamadas de consequentes pois possibilitam, da mesma forma, a execução de uma JD[A] identificada algoritmicamente. Note-se que este conceito de configuração consequente complementa aquele definido no capítulo 3 não tendo sido anteriormente definido como tal para não comprometer o seu entendimento.

6.3. ANÁLISE DA TERCEIRA PARTIDA

Esta partida tal como a segunda foi realizada tendo por entrada o arquivo MEMORI.DAT, contendo as configurações de perda armazenadas até então. O jogo foi realizado com a variável Qi assumindo o valor zero, sendo portanto efetuada uma análise de nível 2. Realizada a jogada [A] (4, 8) o programa respondeu com a jogada [P] (14, 18), uma vez que as jogadas [P] (3, 7), (1, 5) e (14, 13) possibilitam a execução de JD[A] e que a configuração resultante da jogada [P] (16, 12) é de perda (armazenada na segunda partida). A jogada [P] (14, 18) não possibilita, do ponto de vista de uma análise de nível 2 (Qi = 0), a execução de uma JD[A] e foi portanto a jogada [P] escolhida. Após a jogada [A] (8, 7) o programa identifica que, para qualquer uma de suas possíveis jogadas, existe pelo menos uma JD[A], armazenando a configuração de número

5 como de perda. Em decorrência ao processo de retorno, a configuração de número 3 também é armazenada como uma CP.

6.4. ANÁLISE DA QUARTA PARTIDA

Tendo por entrada o aprendizado decorrente da realização das 3 primeiras partidas (refletido pelo arquivo MEMORI.DAT), o programa alterou a sua segunda jogada de (18, 14) para (1, 5) conseguindo vencer a partida. No Anexo 'C encontra-se a listagem produzida (entradas/saídas) quando da realização desta partida.

Note-se que o processo de aprendizado caracterizou, em apenas 3 partidas, o fato de que a JD[A], inicialmente identificada na décima sexta jogada, foi na realidade, possibilitada por uma falha realizada pelo programa na terceira jogada.

6.5. ANÁLISE DA QUINTA E SEXTA PARTIDA

Estas partidas ilustram a utilização de níveis de análise diferentes. A partida de número 5 foi realizada com a variável Qi assumindo o valor zero, sendo portanto efetuada uma análise de nível 2. A partida de número 6 foi realizada com a variável Qi assumindo o valor 4, sendo portanto, efetuada uma análise de nível 10 (2Qi + 2). Note-se que na sexta partida,

já a primeira jogada [P] (20, 16), difere daquela realizada na partida anterior (1, 5). O vetor (pilha) Move, retratado na figura 23, contém o resultado da aplicação da função de avaliação às configurações resultantes das possíveis jogadas [P] após a realização da jogada [A] (2, 6).

POSSÍVEIS JOGADAS P	SALDO	SSEU	INFO	
1,5	0	6	4	← Topo da Pilha
3,7	0	7	7	
20,16	0	8	4	
18,14	0	9	7	
1,2	- 2	6	4	
3,2	- 3	7	4	← Base da Pilha

FIGURA 23 - VETOR (PILHA) MOVE APÓS APLICAÇÃO DA FUNÇÃO DE AVALIAÇÃO

Sabendo-se que o procedimento ESCOLHEJOGADA trata o vetor Move como uma pilha, jogadas são submetidas ao procedimento LEGAL do topo para a base, deduz-se que as jogadas [P] (1, 5) e (3, 7), por não terem sido validadas pelo procedimento LEGAL, quando da realização da sexta partida, leva(m) à possível(is) JD'(s) [A].

7. ANÁLISE, CONCLUSÕES E RECOMENDAÇÕES

Neste capítulo é realizada uma análise crítica do algoritmo PAD e do processo de aprendizado. São apresentadas conclusões, destacando-se a da avaliação do desempenho do algoritmo PAD nas situações onde a aplicação do algoritmo Minimax seria crítica (veja ítem 4). Finalmente são colocadas algumas recomendações quanto à possíveis otimizações e sugestões para estudos futuros.

7.1. ANÁLISE CRÍTICA DO ALGORITMO

a). Sejam:

K - o número médio de possíveis jogadas [P] ou jogadas [A] em cada nível ("Fanout" ou "Branching Factor");

N - o número médio de jogadas [P] analisadas em cada nível (quanto melhor for a função de avaliação menor será o valor de N);

Qi- variável que identifica o nível de análise efetuado (nível 2Qi + 2);

Desprezando-se a análise efetuada em função da existência de configurações terminais deduz-se (veja Anexo-D) que o número total de jogadas simuladas, para possibilitar a escolha da melhor jogada [P] (MODULO-D), é dado pela seguinte fórmula:

$$\text{JOGADAS}\ [K,N,Qi] = (2K + NK)\,(NK^{Qi+1} - 1) \,/\, NK - 1$$

Note-se que na dedução da fórmula apresentada (Anexo-D) não foram consideradas as jogadas cujas configurações resultantes já foram previamente analisadas (jogadas simuladas apenas para dar-se continuidade ao processo de análise). A fórmula traduz portanto, em termos do número de configurações analisadas, o esforço de análise necessário para a escolha da melhor jogada [P]

Sendo o "Fanout" do jogo de xadrez ≃ 38 [5] e assumindo-se a optimalidade da função de avaliação (N = 1), verificamos que são analisadas pelo algoritmo PAD aproximadamente 169.000 configurações, quando da escolha da melhor jogada [P] (considerando-se o jogo de xadrez e uma análise de nível 6, Qi = 2). Sabendo-se que no algoritmo Minimax são avaliadas aproximadamente 110.000 configurações folha [5 pag. 187] para a escolha da melhor jogada [P] (considerando-se o jogo de xadrez, uma análise de nível 6 e um aproveitamento integral do algoritmo Alfa-Beta), concluímos que os algoritmos Minimax e PAD são, desconsiderando-se neste último a análise efetuada em função da geração de configurações terminais, aparentemente equivalentes em termos de esforço computacional.

b). A rigor a variável booleana e global Repetição, que indica a ocorrência de um laço no processo de análise - utilizada para descartar a existência de JD[P] no procedimento ANALISA (# 19) - é dispensável, uma vez que, se a mesma assumir a condição de verdadeira, então a variável global Ok (que indica a existência de JD[P]), obrigatoriamente assumirá a condição de falsa. A eliminação desta variável descarta a necessidade do último bloco 5 no procedimento ANALISA (# 19); senão vejamos:

A atribuição de verdadeira à variável Repetição pode ser realizada pelos procedimentos SIMULA (# 20) ou LEGAL (#21). SIMULA somente atribui a condição verdadeira se houve um laço no processo de análise (configurações repetidas), caso em que, também será atribuida a condição de falsa à variável Ok, uma vez que, a defesa [P] não leva à JD [P].

LEGAL é chamado a partir de SIMULA e neste caso, será atribuida a condição de falsa à variável Ok em SIMULA, independente da condição atribuida à variável Repetição em LEGAL. Conclui-se portanto,que se a condição da variável global Repetição for verdadeira então a condição da variável global Ok necessariamente será falsa.

Como não poderia deixar de ser, o algoritmo PAD, sofreu diversas alterações/otimizações para finalmente chegar-se ao algoritmo aqui descrito. Deve-se retornar a um estágio anterior para poder justificar-se a então necessidade da variável global Repetição. Neste,não se verificava a pos-

sibilidade da existência de JD[P] quando da escolha da melhor defesa [P] (não existia o bloco 3.2 da figura 9). Neste caso a ocorrência de um laço no processo de análise da melhor defesa [P] (o que evidencia a não existência de JD[P]) não alterava a condição da variável global Ok; tornava-se portanto necessária a identificação da ocorrência do laço, para que o procedimento ANALISA (# 19) pudesse então descartar a existência de JD[P]. Utilizou-se a variável global Repetição para esta finalidade. Sem dúvida a existência do laço também poderia ter sido indicada atribuindo-se a condição falsa à variável global Ok com o que evidentemente seria descartada a existência de JD [P] no procedimento ANALISA (# 19). Neste caso porém, a variável Ok estaria sendo utilizada para duas finalidades diferentes - existência de JD[P] e ocorrência de um laço no processo de análise da melhor defesa [P]. A utilização de uma mesma variável para finalidades diferentes dificulta o processo de depuração do algoritmo, razão pela qual foi utilizada a variável Repetição indicando a ocorrência do laço.

c). A identificação de uma JD[P] (MODULO-B) é acionada pelo procedimento ANALISA (# 19) apenas se a jogada [P] inicialmente simulada gerar uma CT[P], ou equivalentemente, se existe uma JTB[P] (veja ítem 4.2.2). Note-se que, mesmo não levando a uma CT[P], a jogada [P] poderá levar a

uma JD [P], se com a sua realização qualquer jogada [A] possibilitar a execução de pelo menos uma JT [P]. Esta condição não é verificada pelo algoritmo PAD, podendo porém ser facilmente implementada pelo procedimento AUXILIAR (veja figura 24).

```
INICIO (1)

   JOGADA-DECISIVA: = verdadeiro
   SIMULA  JOGADA [P]  proposta
   MODULO-A  [P]
   Se não existe JTB [P]  então
   REPITA
      SIMULA possível JOGADA [A]
      MODULO-A  [P]
      Se não existe JT [P]  então JOGADA-DECISIVA: = falso
   Até que JOGADA-DECISIVA = falso ou não
   existe outra possível JOGADA [A]

FIM (1)
```

FIGURA 24 - PROCEDIMENTO AUXILIAR

Para viabilizar a identificação da condição mencionada o procedimento AUXILIAR deverá ser chamado quando da execução do procedimento ANALISA (# 19) sempre que a jogada [P] simulada não levar a uma CT [P] (imediatamente após o primeiro bloco 9) e quando da execução do procedimento SIMULA (# 20) sempre que a possível defesa [P] não levar a

uma JTB [P] (ao iniciar-se o quinto bloco 5); neste caso, se a defesa [P] levar a uma JD [P] (identificada pelo procedimento AUXILIAR) deve ser atribuido o valor verdadeiro à variável local Decisiva, dispensando-se a chamada do procedimento LEGAL. A variável global Jogada-Decisiva da figura 24 será implementada pela variável booleana e global Ok.

d). A escolha da melhor jogada [P] (MODULO-D), implementada pelos procedimentos SALDO (# 25), ORDENA (# 24) e ESCOLHEJOGADA (# 26) poderá eventualmente levar a uma JTB [P]. Sabe-se que a mesma não leva a uma JD [P], pois se este fosse o caso, a jogada [P] em questão teria sido anteriormente selecionada pelo procedimento ANALISA (# 19) (MODULO-B). Note-se que as possíveis defesas [A] (em decorrência da JTB [P]) são implicitamente simuladas (MODULO-E) pelos procedimentos ANALISA (# 19) e PENSA (# 27) chamados a partir do procedimento LEGAL (# 21).

7.2. ANÁLISE CRÍTICA DO APRENDIZADO

A técnica apresentada propõe o armazenamento da configuração de perda através da sua decomposição em um conjunto de 28 relações (não haveria um paralelo com a análise de Fourier ?). Um número qualquer de relações, até mesmo uma, pode na realidade ser utilizado. É claro que alterações no número de relações, acarretam alterações no tamanho da memória

necessária para o armazenamento das configurações de perda e na curva distribuição de probabilidade associada à ocorrência de um verdadeiro-falso. Das 28 relações propostas as 21 relações definidas pelas matrizes P_2, P_3, P_4, P_5, P_6 e P_7, são na realidade consequentes das sete relações definidas pela matriz P_1. Dada a configuração $(x_1, x_2, x_3, x_4; y_1, y_2, y_3, y_4)$ por que armazenamos a relação P_2 $(x_2, 1, x_3)$ se a mesma é consequência das relações P_1 $(x_1, 1, x_2)$ e P_1 $(x_1, 2, x_3)$? Na realidade só podemos afirmar que:

$$P_1 (x, 1, y) \wedge P_1 (x, 2, z) \Box P_2 (y, 1, z')$$

se os bit's referentes as relações P_1 $(x, 1, y)$ e P_1 $(x, 2, z)$ foram assinalados em função da mesma configuração de perda. Como não recuperamos em que configurações de perda os bit's foram assinalados não podemos proceder a tal inferência.

Sejam, por exemplo, duas configurações de perda distintas C_1 e C_2, dadas por $C_1 = (x_1, x_2, x_3, x_4; y_1, y_2, y_3, y_4)$ e $C_2 = (x_1, k_1, k_2, k_3; l_1, l_2, l_3, l_4)$. Se o esquema de armazenamento considerar somente as sete relações da matriz P_1, então a configuração $C_3 = (x_1, k_1, x_3, x_4; y_1, y_2, y_3, y_4)$ será recuperada como uma configuração de perda sem que esta jamais tenha sido armazenada como tal. Neste caso, a hipótese de $P_1 (x_1, 1, k_1) \wedge P_1 (x_1, 2, x_3) \Box P_2 (k_1, 1, x_3)$, o que leva a identificação da configuração C_3 como de perda, é na realidade falsa.

Podemos portanto, supor que quanto maior for o número de relações armazenadas, menor será, em função do nú-

mero de configurações de perda armazenadas, a probabilidade de ocorrência de um verdadeiro-falso.

7.3. CONCLUSÕES

No capítulo 4 foi feita uma análise crítica do algoritmo Minimax e, como consequência, propos-se o algoritmo PAD para contornar as restrições apresentadas à estratégia Minimax. Não é objetivo deste trabalho, apresentar uma prova (no sentido estrito do termo) da superioridade do algoritmo PAD sobre Minimax. No entanto acreditamos que os resultados obtidos permitem concluir que as restrições maiores foram contornadas e que PAD se aproxima de uma forma possível de um ser humano jogar. Especificamente pode-se dizer que:

- A naturalidade do PAD é decorrente das cinco classes de preocupações definidas, que, parecem representar as preocupações típicas de um jogador humano, quando da escolha da melhor jogada.
- No PAD a seleção das jogadas [P] , a serem analisadas para a escolha da melhor jogada, é realizada em função da experiência acumulada pelo programa. Assim, somente se descartam as jogadas [P] que comprovadamente levam a uma configução de perda. Assegura-se portanto, que a melhor jogada [P] não será inadvertidamente eliminada no processo de seleção. Isto obviamente não garante a escolha da efetivamente melhor jogada [P] .

- A não escolha da jogada [P] que oferece as melhores chances de vitória contra um adversário fraco, pode ser contornada, considerando-se como válida a jogada [P] considerada inválida pelo procedimento LEGAL (# 21). Neste caso, a jogada [P] possibilita a execução de uma JD [A], que, considerando-se um adversário fraco, poderá não ser executada. A experiência do adversário, fornecida como parâmetro de entrada, poderá ser utilizada quando da validação das jogadas [P] que possibilitam uma JD [A].

- Partindo-se do número total de configurações analisadas, para possibilitar a escolha da melhor jogada [P] (veja ítem 7.1) deduzimos que são analisadas NK^2 (N + 2) configurações a mais, quando do incremento do nível de análise de 2 para 4 (de Qi = 0 para Qi = 1). Portanto, assumindo-se a optimalidade da função de avaliação (N = 1), a alteração de 2 níveis na análise (de 2 para 4) implica, no algoritmo PAD, no incremento de $2K^2$ (K = "Fanout") configurações analisadas. No entanto, não podemos concluir precisamente quanto ao número de continuações (configurações) analisadas uma vez que:

  a). não podemos assumir a optimalidade da função de avaliação.

  b). a dedução do número de configurações analisadas (jogadas-simuladas - Anexo-D), embora retrate o pior caso, não considera as configurações analisadas decorrentes da existência de configurações terminais.

c). a utilização (não implementada) de um esquema de armazenamento/recuperação das configurações de perda para o adversário reduzirá o fator K (de quanto ?), reduzindo-se consequentemente o número de configurações analisadas.

- Como basicamente o objetivo do processo de análise é o de encontrar jogadas decisivas, pode-se concluir que a habilidade resultante do programa é decorrente do nível de análise efetuado (é também decorrente do processo de aprendizado), pois, quanto maior o nível, maior será a abrangência do processo e consequentemente a possibilidade de identificação de jogadas decisivas.

7.4. RECOMENDAÇÕES

Seguem-se algumas sugestões para futuros estudos que poderão vir à otimizar o algoritmo PAD e o processo de aprendizado apresentados:

a). A utilização de um esquema de armazenamento/recuperação das configurações de perda para o adversário reduz (as jogadas [A] que possibilitam uma JD[P] não necessitam ser analisadas), o esforço de análise necessário quando da determinação da melhor jogada [P] .

b). As formas alternativas de armazenamento/recuperação das CP's (alterando-se as relações em número e/ou forma) e o estudo detalhado das curvas de distribuição de probabilidade associadas, propiciam um amplo campo para futuras pesquisas. Pode-se, por exemplo, representar uma determinada configuração de xadrez pela sequência:

$\underbrace{(64 \times 7,\ 64 \times 7,\ 64 \times 7 \ldots 64 \times 7)}_{32 \text{ vezes}}$

64 - Posições no tabuleiro.
7 - Tipos de peças incluindo a não existência.
32 - peças no total.

e interpretá-la como:

$(64 \times 7\ \underbrace{(64 \times 7,\ 64 \times 7 \ldots 64 \times 7)}_{31 \text{ vezes}})$; ou seja da forma (l_1, l_2).

Associando-se à l_2 um dígito verificador, por exemplo, Módulo-20, pode-se armazenar a referida configuração de xadrez através de uma única relação (64 x 7 x DÍGITO-VERIFICADOR). Oito mil novecentos e sessenta bit's seriam necessários para o armazenamento das configurações de perda, sendo associado um único bit para cada configuração de perda armazenada. Portanto, uma vez armazenada a primeira configuração, a probabilidade de ocorrência de um verdadeiro-falso seria de $1/8960$, ou seja, $\simeq 10^{-4}$.

c). No processo de aprendizado obtém-se a configuração de perda a ser armazenada desfazendo-se a última jogada [A] sempre que for identificada uma situação de perda. Pode-se otimizar o processo de aprendizado, se, quando da identificação de uma situação de perda, for armazenada a configuração existente. Neste caso, além de aumentar-se a abrangência do aprendizado (jogadas que até então eram permitidas, não são mais validadas - veja Figuras 25 e 26), elimina-se também a duplicidade de análise existente no processo implementado (veja Figuras 27 e 28). Note-se que, para ser eliminada a duplicidade no processo de análise, algumas configurações de perda devem ser armazenadas pelo procedimento PENSA (# 27), tornando-se desnecessário o processo de retorno quando a configuração de perda a ser armazenada já se encontrar no arquivo MEMORI.DAT.

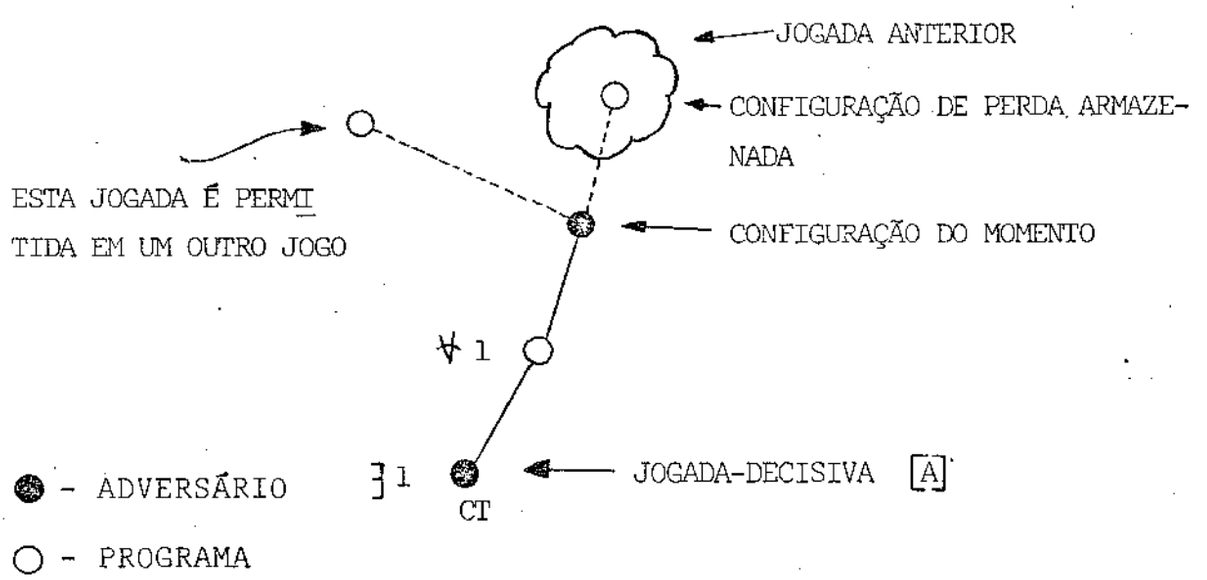


FIGURA 25 - PROCESSO IMPLEMENTADO COM Qi = 0

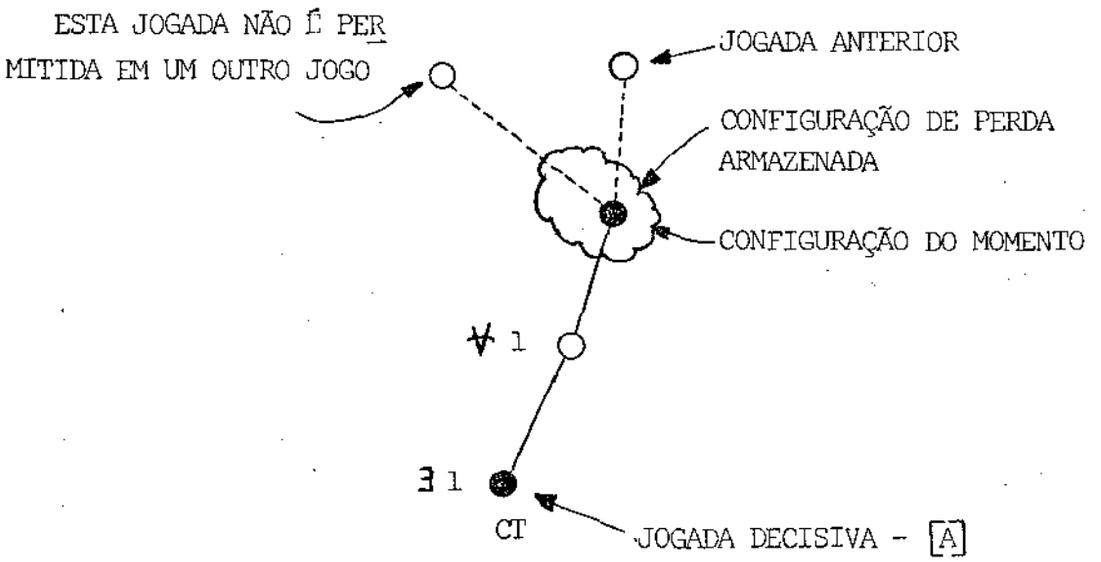


FIGURA 26 - NOVO PROCESSO COM Qi = 0

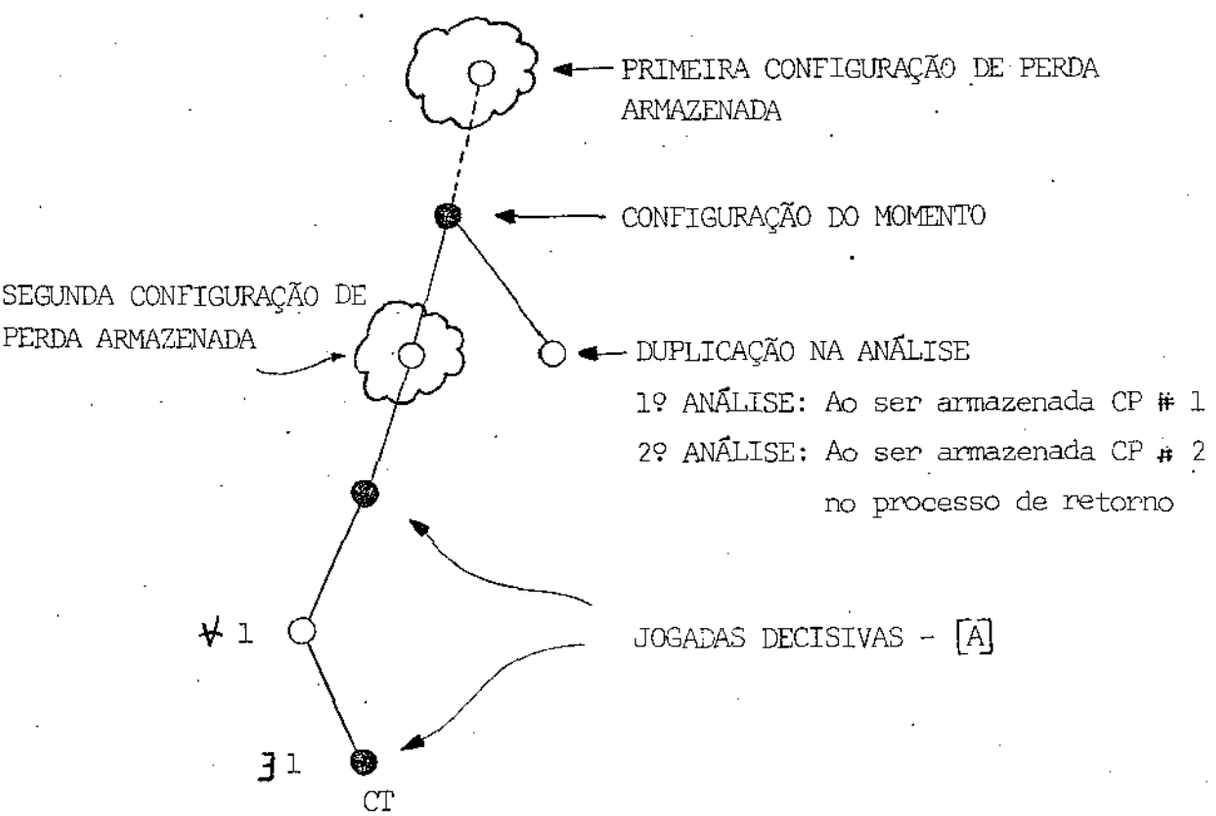


FIGURA 27 - PROCESSO IMPLEMENTADO COM Qi = 1

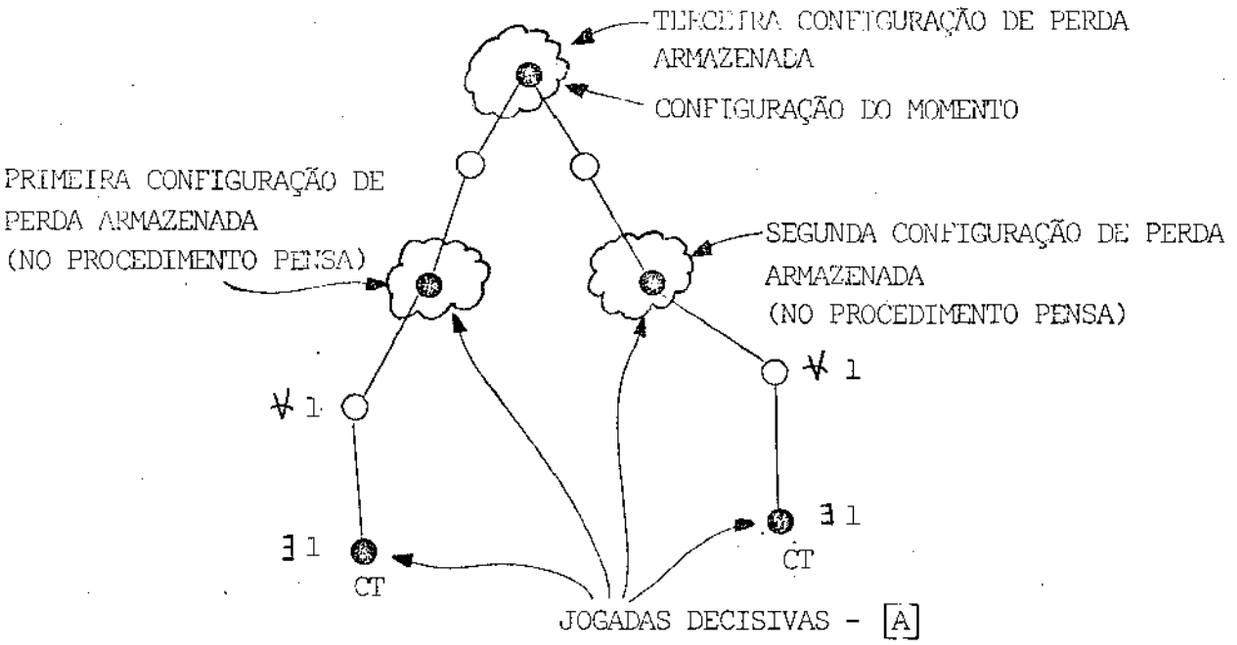


FIGURA 28 - NOVO PROCESSO COM Qi = 1

d). Mantendo-se o processo de aprendizado implementado, pode-se, da mesma forma que a descrita no ítem anterior, eliminar parte da duplicidade ocorrida no processo de análise, simplesmente inibindo-se o processo de retorno quando a configuração de perda a ser armazenada já se encontrar no arquivo MEMORI.DAT.

e). Finalmente podemos associar à configuração de perda um, digamos, Padrão de Perda, supondo-se que nem todas as peças de uma determinada configuração são decisivas para a determinação da CP. Identificadas as peças chave (como?) quando da existência de uma CP podemos armazená-las (atra

vés de suas inter-relação) como o Padrão de Perda. Posteriormente uma configuração somente deve ser dada como de perda, se de suas inter-relações, um número maior ou igual àquele utilizado para o armazenamento dos Padrões de Perda, estiver armazenado.

BIBLIOGRAFIA

[1] RUDERMAN, HARRY D. - Tac-Tickle a challeging game of strategy - Tuifle Creek, Pa, Wff'n Proof Publishers, 1965, 1967.

[2] FEIGENBAUM, EDWARD A. & FELDMAN, JULIAN - University of California, Berkeley - Computers and Thought - MacGraw-Hill, 1963.

[3] NILSSON, NILS J. - Artificial Intelligence - Information processing,74; proceedings of IFIP Congress, Stockholm, 1974. London, North-Holland, 1974.

[4] NILSSON, NILS J. - Problem Solving Methods in Artificial Intellingence - MacGraw - Hill.

[5] WHOLAND, NORMAN D. - A Computer Chess Tutorial - Byte Publications Inc., October 1978.

FREY, PETER W. & ATKIN, LARRY A. - Creating a Chess Player - An Essay on Human and Computer Chess Skill - Byte Publications Inc., October, 1978.

[6] JR., RUSSEL R. YOST - Eighteen with a Die - A Learning Game Player - Byte Publications Inc., January 1980.

[7] MATSURA, EDUARDO KOITI & BACCHO, SERGIO - Aplicação de processos heurísticos de decisão em um programa para jogar xadrez - Orientador Prof. Alberto Elfes, Divisão de Processamento de Dados, Centro Técnico Aeroespacial - ITA, 1979.

[8] HEWITT, CARL - Viewing Control Structures as Pattern of Passing Messages - North-Holland Publishing, 1977.

[9] STUCK, H. L. - Approaching Game Program Design - Byte Publications, February 1978.

[10] FILHO, HEITOR PINTO - Nos Bastidores do Computador que joga Xadrez - Dados e Idéias - Agosto/Setembro 1975.

A N E X O A

PROGRAMA QUE IMPLEMENTA O JOGO "TAC-TICKLE"

```
        CONST   SEPAR = '+------------------------------+';
                LIN = '|      |      |      |      |';

{**********  T I P O S   D E   V A R I A V E I S  **********}

        TYPE    LINK = ^POSICAO;
                POSICAO = RECORD
                            QUEM : CHAR;
                            PROXIMO : ARRAY[1..8] OF LINK;
                            INFO : INTEGER
                          END;
                JOGADA = RECORD
                           DE : LINK;
                           PARA : LINK
                         END;
                DECISAO  = RECORD
                             LIN : INTEGER;
                             COL : INTEGER;
                             POINT : LINK;
                             INFO : INTEGER;
                             SSEU : INTEGER;
                             SALDO : INTEGER
                           END;
                ENDERECO = RECORD
                             LIN : INTEGER;
                             COL : INTEGER;
                             POSS : INTEGER;
                             POINT : INTEGER
                           END;
                HISTORICO = RECORD
                              QUEM : ARRAY[1..8] OF CHAR;
                              ENDI : ARRAY[1..8] OF LINK
                            END;

                 VINTE = 1..20;
                 BIT = 0..1;
                 ARQUIVO = FILE OF INTEGER;

{**********     V A R I A V E I S   G L O B A I S     **********}

        VAR TAB : ARRAY[0..6,0..5] OF LINK;
            HISTORIA : ARRAY[1..100] OF HISTORICO;
            VETOR : ARRAY[1..16] OF ENDERECO;
            LOCAL : ARRAY[1..4] OF LINK;
            STACK : ARRAY[1..16] OF JOGADA;
            MOVE  : ARRAY[1..16] OF DECISAO;
            REPETICAO,IGUAL,ILEGAL,ESCOLHE,SUCESSO,FRACASSO,OK,VISOR : BOOLEAN;
            HIST,MEU,SEU,LD,CD,LP,CP,I,J,K,T : INTEGER;
            QI,REC,POS,LINHA,COLUNA,VEZ,PLOCAL,PSTACK,PMOVE : INTEGER;
            NUMLANC : INTEGER;
            LIBERA,JOGADOR : CHAR;
            QUAL,U,V,P,Q,NU,NV : LINK;
            JOGO_VELHO : PACKED ARRAY [1..20] OF CHAR;
            BEL,LF,CR,CAN,SUB,GS,RS,US : CHAR;
            CONTINUA,ESPECIAL,GRAVA,OPCAO,EXPLICO,DEVICE,VER_QUADRO   : CHAR;

            MEMORIA : ARRAY[1..312] OF INTEGER;
            CONFIGURACAO : ARRAY[1..8] OF VINTE;
            ENTRADA : ARQUIVO;
            PALAVRA,BITE : INTEGER;
```

```
    MASCARA : ARRAY[1..36] OF BIT;
    EXISTE : BOOLEAN;
    PILHA : ARRAY[1..50] OF JOGADA;
    PPILHA : INTEGER;

%********** AQUI ESTAO AS PROCEDURES **********\

%01\  PROCEDURE TRANSFORMA;
      BEGIN
       IF JLEGAL = TRUE THEN
       BEGIN
        WRITELN(TTY,' JLEGAL E TRUE');BREAK(TTY)
       END
             ELSE
       BEGIN
        WRITELN(TTY,'  JLEGAL E FALSE');BREAK(TTY)
       END
      END;

%02\  PROCEDURE GRAVO;
      VAR LINHA,COLUNA : INTEGER;
      BEGIN %1\
       FOR LINHA := 1 TO 5 DO
       BEGIN %2\
        WRITELN(' ':20,LIN);
        WRITE(' ':20);
        FOR COLUNA := 1 TO 4 DO
        BEGIN %3\
         WRITE('|   ',TAB[LINHA,COLUNA]^.QUEM,'   ')
        END; %3\
        WRITELN('|');
        WRITELN(' ':20,LIN);
        IF LINHA # 5 THEN
        BEGIN %3\
         WRITELN(' ':20,SEPAR)
        END %3\
       END; %2\
       WRITELN;WRITELN
      END; %1\

%03\  PROCEDURE FIRSTPOINTER;
      BEGIN %1\
       FOR LINHA := 0 TO 6 DO
       BEGIN %2\
        FOR COLUNA := 0 TO 5 DO
        BEGIN %3\
         TAB[LINHA,COLUNA] := NIL
        END %3\
       END; %2\
       FOR LINHA := 5 DOWNTO 1 DO
       BEGIN %2\
        FOR COLUNA := 4 DOWNTO 1 DO
        BEGIN %3\
         NEW(P);
         TAB[LINHA,COLUNA]:= P;
         P^.PROXIMO[1] := TAB[LINHA,COLUNA+1];
         P^.PROXIMO[2] := TAB[LINHA+1,COLUNA];
         P^.PROXIMO[5] := TAB[LINHA+1,COLUNA+1]
        END %3\
       END %2\;
```

```
      FOR LINHA := 1 TO 5 DO
      BEGIN %2\
       FOR COLUNA := 1 TO 4 DO
       BEGIN %3\
        P := TAB[LINHA,COLUNA];
        P^.PROXIMO[3] := TAB[LINHA,COLUNA-1];
        P^.PROXIMO[4] := TAB[LINHA-1,COLUNA];
        P^.PROXIMO[6] := TAB[LINHA+1,COLUNA-1];
        P^.PROXIMO[7] := TAB[LINHA-1,COLUNA-1];
        P^.PROXIMO[8] := TAB[LINHA-1,COLUNA+1]
       END %3\
      END; %2\
      TAB[1,1]^.INFO := 3 ; TAB[1,2]^.INFO := 4;
      TAB[1,3]^.INFO := 4 ; TAB[1,4]^.INFO := 3;
      TAB[2,1]^.INFO := 4 ; TAB[2,2]^.INFO := 7;
      TAB[2,3]^.INFO := 7 ; TAB[2,4]^.INFO := 4;
      TAB[3,1]^.INFO := 6 ; TAB[3,2]^.INFO := 9;
      TAB[3,3]^.INFO := 9 ; TAB[3,4]^.INFO := 6;
      TAB[4,1]^.INFO := 4 ; TAB[4,2]^.INFO := 7 ;
      TAB[4,3]^.INFO := 7 ; TAB[4,4]^.INFO := 4;
      TAB[5,1]^.INFO := 3 ; TAB[5,2]^.INFO := 4;
      TAB[5,3]^.INFO := 4 ; TAB[5,4]^.INFO := 3
     END %1\;

%04\  PROCEDURE FIRSTQUADRO;
     BEGIN %1\
      TAB[1,1]^.QUEM := 'X'; TAB[1,2]^.QUEM := 'O';
      TAB[1,3]^.QUEM := 'X'; TAB[1,4]^.QUEM := 'O';
      TAB[5,1]^.QUEM := 'O'; TAB[5,2]^.QUEM := 'X';
      TAB[5,3]^.QUEM := 'O'; TAB[5,4]^.QUEM := 'X';
      FOR LINHA := 2 TO 4 DO
      BEGIN %2\
       FOR COLUNA := 1 TO 4 DO
       BEGIN %3\
        TAB[LINHA,COLUNA]^.QUEM := ' ';
       END %3\;
      END %2\;
      FOR LINHA := 1 TO 5 DO
       BEGIN%2\
        FOR COLUNA := 1 TO 4 DO
         BEGIN%3\
          JOGO_VELHO[(4*(LINHA-1)+(COLUNA-1))+1] := TAB[LINHA,COLUNA]^.QUEM;
         END%3\;
       END%2\;
     END; %1\

%05\  PROCEDURE JOGA(L1,L2 : LINK);
     BEGIN %1\
      L1^.QUEM := ' ';
      L2^.QUEM := JOGADOR
     END %1\;

%06\  PROCEDURE VER(P,L : LINK);
     BEGIN %1\
      IF L^.QUEM = JOGADOR THEN
      BEGIN %2\
       PSTACK:=PSTACK+1;
       STACK[PSTACK].DE := L;
       STACK[PSTACK].PARA := P
      END %2\
```

```
        END; %1\

%07\    PROCEDURE TRANSA(L1,P : LINK);
        BEGIN %1\
         PSTACK:=PSTACK+1;
         STACK[PSTACK].DE := L1;
         STACK[PSTACK].PARA := P
        END; %1\

%08\    PROCEDURE COMPLETA(I:INTEGER;P:LINK;LINHA,COLUNA:INTEGER);
        VAR L,L1,L2 : LINK;
        BEGIN %1\
          CASE I OF
            1,2,3,4 : ;
        5,6 : BEGIN %2\
               L := TAB[LINHA,COLUNA]^.PROXIMO[I+2];
               IF L<>NIL THEN
               BEGIN %3\
                IF L^.QUEM = JOGADOR THEN
                BEGIN %4\
                        L1:=P^.PROXIMO[I-4];
                 L2:=P^.PROXIMO[I-3];
                 IF L1 <> NIL THEN
                 BEGIN %5\
                VER(P,L1)
                 END; %5\
                 IF L2 <> NIL THEN
                 BEGIN %5\
                  VER(P,L2)
                 END %5\
                END %4\
               END %3\
              END; %2\
        7:    BEGIN %2\
               L:=TAB[LINHA,COLUNA]^.PROXIMO[5];
                IF L <> NIL THEN
               BEGIN%3\
                IF L^.QUEM  = JOGADOR THEN
                BEGIN %4\
                 L1:=P^.PROXIMO[4];
                 L2:=P^.PROXIMO[3];
                 IF L1 <> NIL THEN
                 BEGIN %5\
                  VER(P,L1)
                 END; %5\
                 IF L2 <> NIL THEN
                 BEGIN %5\
                  VER(P,L2)
                END %5\
               END %4\
              END %3\
             END; %2\
        8:    BEGIN %2\
               L:=TAB[LINHA,COLUNA]^.PROXIMO[6];
                IF L<> NIL THEN
          BEGIN
           IF L^.QUEM= JOGADOR THEN
           BEGIN
            L1:=P^.PROXIMO[1];
            L2:=P^.PROXIMO[4];
```

```
            IF L1 # NIL THEN
            BEGIN%5\
             VER(P,L1)
            END; %5\
            IF L2 # NIL THEN
            BEGIN %5\
             VER(P,L2)
            END %5\
           END %4\
          END %3\
         END; %2\
OTHERS : P:=P
     END %CASE\
END; %1\

%09\  PROCEDURE TRANSFERE(P:LINK;L1,L2:INTEGER);
      VAR Q : LINK;
      BEGIN %1\
       Q:=P^.PROXIMO[L2];
       IF Q # NIL THEN
       BEGIN %2\
        IF Q^.QUEM = ' ' THEN
        BEGIN
         PSTACK := PSTACK+1;
         STACK[PSTACK].PARA := Q;
         Q := P^.PROXIMO[L1];
         STACK[PSTACK].DE := Q
        END %3\
       END %2\
      END ; %1\

%10\  PROCEDURE TRANSPECA(P:LINK;L:INTEGER);
      VAR Q : LINK;
      BEGIN %1\
       Q := P^.PROXIMO[L];
       IF Q # NIL THEN
       BEGIN %2\
        IF Q^.QUEM = ' ' THEN
        BEGIN %3\
         PSTACK := PSTACK+1;
         STACK[PSTACK].PARA := Q;
         STACK[PSTACK].DE := P
        END %3\
       END %2\
      END; %1\

%11\  PROCEDURE VERIFICA(LINHA,COLUNA,X,Y:INTEGER);
      VAR P : LINK;
      BEGIN %1\
      P := TAB[LINHA,COLUNA];
         CASE X OF
       1  : BEGIN %2\
              CASE Y OF
            6 : BEGIN %3\
                 TRANSFERE(P,6,3);
                 TRANSFERE(P,1,8)
                END ; %3\
            7 : BEGIN %3\
                 TRANSFERE(P,7,3);
                 TRANSFERE(P,1,5)
```

```
        END; %3\
      OTHERS : P:=P
       END %CASE\
    END; %2\
2   :  BEGIN %2\
         CASE Y OF
       7 : BEGIN %3\
            TRANSFERE(P,7,4);
            TRANSFERE(P,2,5)
           END; %3\
       8 : BEGIN %3\
            TRANSFERE(P,8,4);
            TRANSFERE(P,2,6)
           END %3\;
       OTHERS :P:=P
         END %CASE\
       END ; %2\
3   :  BEGIN %2\
            CASE Y OF
       5 : BEGIN %3\
            TRANSFERE(P,5,1);
            TRANSFERE(P,3,7)
           END; %3\
       8 : BEGIN %2\
            TRANSFERE(P,8,1);
            TRANSFERE(P,3,6)
           END; %3\
       OTHERS :P:=P
         END %CASE\
       END ; %2\
4   :  BEGIN %2\
            CASE Y OF
       5 : BEGIN %3\
            TRANSFERE(P,5,2);
            TRANSFERE(P,4,7)
           END; %3\
       6 : BEGIN %3\
            TRANSFERE(P,6,2);
            TRANSFERE(P,4,8)
           END; %3\
       OTHERS :P:=P
         END %CASE\
       END; %2\
5   :  BEGIN %2\
            CASE Y OF
       6 : TRANSPECA(P,2);
       8 : TRANSPECA(P,1);
       OTHERS :P:=P
         END %CASE\
       END; %2\
6   :  BEGIN%2\
            CASE Y OF
       7 : TRANSPECA(P,3);
       OTHERS :P:=P
         END %CASE\
       END ; %2\
7   :  BEGIN %2\
            CASE Y OF
       8 : TRANSPECA(P,4);
       OTHERS : P:=P
```

```
            END %CASE\
           END; %2\
OTHERS : P:=P
    END %CASE\
END ; %1\

%12\   PROCEDURE DECIDE;
VAR PILHA : ARRAY [1..3] OF INTEGER;
    POINTER,I,K,KK,LINHA,COLUNA,M,MM : INTEGER; L1,P,Q,L2,L : LINK;
BEGIN %1\
 LINHA := 1 ; PSTACK:=0 ;
 WHILE LINHA <= 5 DO
 BEGIN %2\
  COLUNA:= 1;
  WHILE COLUNA <= 4 DO
  BEGIN %3\
   IF TAB[LINHA,COLUNA]^.QUEM = JOGADOR THEN
   BEGIN %4\
    POINTER := 0;
    FOR I:=1 TO 8 DO
    BEGIN %5\
     IF TAB[LINHA,COLUNA]^.PROXIMO[I]<>NIL THEN
     BEGIN %6\
      P := TAB[LINHA,COLUNA]^.PROXIMO[I];
      IF P^.QUEM = JOGADOR THEN
      BEGIN %7\
       POINTER := POINTER +1;
       PILHA[POINTER] := I
      END %7\
                                          ELSE
      BEGIN %7\
       IF P^.QUEM = ' ' THEN
       BEGIN %8\
        COMPLETA(I,P,LINHA,COLUNA);
        IF P^.PROXIMO[I] <> NIL THEN
        BEGIN %9\
         Q := P^.PROXIMO[I];
         IF Q^.QUEM = JOGADOR THEN
         BEGIN %10\
            CASE I OF
          1,2 : BEGIN %11\
                L := TAB[LINHA,COLUNA]^.PROXIMO[I+2];
                IF L <> NIL THEN
                BEGIN %12\
                 IF L^.QUEM = JOGADOR THEN
                 BEGIN %13\
                  TRANSA(Q,P)
                 END %13\
                END %12\
               END; %11\
          3,4 : BEGIN %11\
                L := TAB[LINHA,COLUNA]^.PROXIMO[I-2];
                IF L <> NIL THEN
                BEGIN %12\
                 IF L^.QUEM = JOGADOR THEN
                 BEGIN %13\
                  TRANSA(Q,P)
                 END %13\
                END %12\
               END; %11\
```

```
          5,6,7 : BEGIN %11\
                   L1 := TAB[LINHA,COLUNA]^.PROXIMO[I-4];
                   L2 := TAB[LINHA,COLUNA]^.PROXIMO[I-3];
                   IF L1 <> NIL THEN
                   BEGIN %12\
                    VER(P,L1)
                   END; %12\
                   IF L2 <> NIL THEN
                   BEGIN %12\
                    VER(P,L2)
                   END %12\
                  END; %11\
           8    :  BEGIN %11\
                   L1 := TAB[LINHA,COLUNA]^.PROXIMO[1];
                   L2 :=TAB[LINHA,COLUNA]^.PROXIMO[4];
                   IF L1 <> NIL THEN
                   BEGIN %12\
                    VER(P,L1)
                   END; %12\
                   IF L2 <> NIL THEN
                   BEGIN %12\
                    VER(P,L2)
                   END %12\
                   END %11\
              END %CASE\
            END %10\
           END %9\
          END %8\
         END %7\
        END %6\
       END; %5\
       IF POINTER > 1 THEN
       BEGIN %5\
        FOR K := POINTER DOWNTO  2 DO
        BEGIN %6\
         FOR M := K-1 DOWNTO 1 DO
         BEGIN %7\
          KK:=PILHA[K];
          MM:=PILHA[M];
          VERIFICA(LINHA,COLUNA,MM,KK)
         END %7\
        END %6\
       END %5\
      END; %4\
      COLUNA := COLUNA +1
     END; %3\
     LINHA := LINHA +1
    END %2\
   END; %1\

%13\  PROCEDURE CONDENSA(VAR TOTAL : INTEGER);
      VAR EXISTE : BOOLEAN; K,L : INTEGER;
      BEGIN %1\
       TOTAL := PSTACK;
       IF PSTACK > 1 THEN
       BEGIN %2\
        FOR K:= 1 TO PSTACK-1 DO
        BEGIN %3\
         EXISTE := FALSE; L:=K;
         REPEAT
```

```
          L:= L+1;
          IF STACK[K].PARA=STACK[L].PARA THEN
          BEGIN %5\
           EXISTE := TRUE;
           TOTAL :=TOTAL-1
          END %5\
         UNTIL (L=PSTACK) OR (EXISTE=TRUE)
        END %3\
       END %2\
                      ELSE
       BEGIN %2\
        IF PSTACK = 1 THEN TOTAL := 1 ELSE TOTAL := 0
       END %2\
      END; %1\

%14\  PROCEDURE PROCURA(L:LINK);
      VAR T:LINK;I:INTEGER;
      BEGIN %1\
       PLOCAL:=0;
       FOR I:=1 TO 4 DO
       BEGIN %2\
        IF L^.PROXIMO[I] # NIL THEN
        BEGIN %3\
         T:= L^.PROXIMO[I];
         IF T^.QUEM = 'X' THEN
         BEGIN %4\
          JOGADOR:='X';JOGA(T,L);
          JOGADOR:='O' ; DECIDE;
          IF PSTACK = 0 THEN
          BEGIN %5\
           PLOCAL:=PLOCAL+1;
           LOCAL[PLOCAL]:=T
          END; %5\
          JOGADOR:='X';JOGA(L,T)
         END %4\
        END %3\
       END %2\
      END; %1\

%15\  PROCEDURE FULMINA(PROX:INTEGER;JOGAD:CHAR;R:LINK);
      VAR L,M : INTEGER; ADV : CHAR; Q,S : LINK;
      BEGIN %1\
       M:=0;
       IF JOGAD = 'X' THEN ADV := 'O' ELSE ADV := 'X';
       REPEAT
        BEGIN %2\
         OK:=TRUE;
         M:=M+1;
         S:=STACK[M].PARA;
         FOR L := 1 TO 4 DO
         BEGIN %3\
          Q:=S^.PROXIMO[L];
          IF Q # NIL THEN
          BEGIN %4\
           IF Q^.QUEM = ADV THEN OK := FALSE
          END %4\
         END %3\
        END %2\
       UNTIL (M = PSTACK) OR (OK = TRUE);
       IF OK = TRUE THEN
```

```
 BEGIN %2\
  U:= P;
  V:=R^.PROXIMO[PROX]
 END %2\
END; %1\

%16\  PROCEDURE ARRUMAVET;
VAR I,K,AUX1,AUX2,AUX3,AUX4 : INTEGER;
BEGIN %1\
 IF J > 1 THEN
 BEGIN %2\
  FOR I := 1 TO J-1 DO
  BEGIN %3\
   FOR K := I+1 TO J DO
   BEGIN %4\
    IF VETOR[I].POSS > VETOR[K].POSS THEN
    BEGIN %5\
     AUX1 := VETOR[I].LIN ; AUX2 := VETOR[I].COL;
     AUX3 := VETOR[I].POSS ; AUX4 := VETOR[I].POINT;
     VETOR[I].LIN := VETOR[K].LIN;
     VETOR[I].COL := VETOR[K].COL;
     VETOR[I].POSS := VETOR[K].POSS;
     VETOR[I].POINT := VETOR[K].POINT;
     VETOR[K].LIN := AUX1 ; VETOR[K].COL := AUX2;
     VETOR[K].POSS := AUX3 ; VETOR[K].POINT := AUX4
    END %5\
   END %4\
  END %3\
 END %2\
END ; %1\

%17\  PROCEDURE GUARDA;
VAR  K,LINHA,COLUNA : INTEGER; P,Q : LINK;
BEGIN %1\
 HIST := HIST +1; K:=0;
 FOR LINHA := 1 TO 5 DO
 BEGIN %2\
  FOR COLUNA := 1 TO 4 DO
  BEGIN %3\
   P := TAB[LINHA,COLUNA];
   IF P^.QUEM # ' ' THEN
   BEGIN %4\
    K:= K+1;
    HISTORIA[HIST].QUEM[K]:=P^.QUEM;
    HISTORIA[HIST].ENDI[K] := P
   END %4\
  END %3\
 END %2\
END; %1\

%18\  PROCEDURE COMPARA;
VAR I,K : INTEGER;
BEGIN %1\
 IF HIST > 1 THEN
 BEGIN %2\
  I:=0;
 REPEAT
  BEGIN %3\
   I:=I+1;
   IGUAL := TRUE ; K:= 0;
```

```
      REPEAT
       K:= K+1;
       IF NOT(HISTORIA[HIST].ENDI[K]=HISTORIA[I].ENDI[K])
       OR NOT(HISTORIA[HIST].QUEM[K]=HISTORIA[I].QUEM[K] ) THEN
        BEGIN %5\
         IGUAL := FALSE
        END %5\
      UNTIL (K=9) OR (IGUAL = FALSE)
     END %3\
    UNTIL (I=HIST-1) OR (IGUAL=TRUE)
    END %2\
                  ELSE
    BEGIN %2\
     IGUAL := FALSE
    END %2\
   END; %1\

%19\  PROCEDURE SIMULA(VAR L : LINK); FORWARD;

      PROCEDURE ANALISA(JOGAD : CHAR);

      VAR ADV : CHAR;
          L,P,Q,R,S,UU,VV : LINK;
          LINHA,COLUNA,T,I,Z,MAX : INTEGER;
          AUX : ARRAY[1..4] OF INTEGER;
          VET : ARRAY[1..16] OF ENDERECO;
      BEGIN %0\
      WRITELN(TTY,' [ANALISA]');BREAK(TTY);TRANSFORMA;\
      COMPARA;
      IF IGUAL = FALSE THEN
      BEGIN %1\
       I:=0;  OK:=FALSE;
       IF JOGAD = 'X' THEN ADV := 'O' ELSE ADV := 'X';
       FOR LINHA := 1 TO 5 DO
       BEGIN %2\
        FOR COLUNA := 1 TO 4 DO
        BEGIN %3\
         P:=TAB[LINHA,COLUNA];
         IF P^.QUEM = JOGAD THEN
         BEGIN %4\
          FOR I := 1 TO 4 DO
          BEGIN %5\
           Q:=P^.PROXIMO[I];AUX[I]:=0;
           IF Q <> NIL THEN
           BEGIN %6\
            IF Q^.QUEM = ' ' THEN
            BEGIN %7\
             Q^.QUEM := JOGAD; P^.QUEM := ' ';
             JOGADOR := ADV ; DECIDE;
             IF PSTACK = 0 THEN
             BEGIN %8\
              JOGADOR := JOGAD ; DECIDE;
              IF PSTACK <> 0 THEN
              BEGIN %9\
               CONDENSA(T);AUX[I]:=T;
               FULMINA(I,JOGAD,P)
              END %9\
             END ; %8\
             Q^.QUEM := ' ' ; P^.QUEM := JOGAD;
             IF OK = TRUE THEN
```

```
      BEGIN %8\
       I:=4;
       LINHA:=5;
       COLUNA:=4
      END %8\
     END %7\
    END %6\
   END; %5\
   FOR I:= 1 TO 4 DO
   BEGIN %5\
    IF AUX[I] # 0 THEN
    BEGIN %6\
     J:= J+1;
     VETOR[J].LIN := LINHA;
     VETOR[J].COL := COLUNA;
     VETOR[J].POSS := AUX[I];
     VETOR[J].POINT := I
    END %6\
   END %5\
  END %4\
 END %3\
END ; %2\
IF OK  = FALSE THEN
BEGIN %2\
 IF J # 0 THEN
 BEGIN %3\
  ARRUMAVET;
  IF JOGAD = 'X' THEN
  BEGIN %4\
   Z:=J;
   FOR I := 1 TO J DO
   BEGIN %5\
    VET[I].LIN := VETOR[I].LIN;
    VET[I].COL := VETOR[I].COL;
    VET[I].POSS := VETOR[I].POSS;
    VET[I].POINT := VETOR[I].POINT
   END ; %5\
   Z:= Z+1;REPETICAO :=FALSE;
   REPEAT
    Z:= Z-1;
    LINHA := VET[Z].LIN;
    COLUNA := VET[Z].COL;
    I := VET[Z].POINT;
    MAX := VET[Z].POSS;
    UU := TAB[LINHA,COLUNA];
    VV := UU^.PROXIMO[I];
    OK := TRUE;
    IF MAX = 1 THEN
    BEGIN %5\
     JOGADOR := 'X';JOGA(UU,VV);DECIDE;GUARDA;
     R:=STACK[1].PARA; I:= 0;
     REPEAT
      I := I+1;
      S:=R^.PROXIMO[I];
      IF S # NIL THEN
      BEGIN %6\
       IF S^.QUEM = 'O' THEN
       BEGIN %7\
        JOGADOR := 'O' ;JOGA(S,R);GUARDA;
        JOGADOR := 'X';DECIDE;
```

```
            IF PSTACK = 0 THEN
            BEGIN %8\
             JOGADOR := 'O';DECIDE;CONDENSA(T);
             IF T > 1 THEN OK := FALSE
                      ELSE
             BEGIN %9\
              IF T = 1 THEN
              BEGIN %10\
               L:=STACK[1].PARA; PROCURA(L);
               SIMULA(QUAL);
              END %10\
                       ELSE
              BEGIN %10\
               JOGADOR := 'X';
               ANALISA(JOGADOR)
              END %10\
             END %9\
            END; %8\
            JOGADOR := 'O'; JOGA(R,S);HIST:=HIST-1
           END %7\
          END %6\
         UNTIL (OK = FALSE) OR (I = 4);
         JOGADOR := 'X';JOGA(VV,UU);HIST := HIST-1
        END; %5\
        IF OK = TRUE THEN
        BEGIN %5\
         IF REPETICAO =TRUE THEN
         BEGIN %6\
         OK := FALSE;
         REPETICAO := FALSE
         END %6\
        END %5\
       UNTIL ( MAX > 1 ) OR (OK = TRUE) OR (Z = 1);
       U:=UU ; V := VV
      END %4\
     END %3\
    END %2\
    END %1\
             ELSE
    BEGIN %1\
    OK := FALSE; J:=0
    END %1\
%   WRITELN(TTY,' FIM DA ANALISA');BREAK(TTY);TRANSFORMA\
   END ; %0\

%20\ PROCEDURE LEGAL (L1,L2 : LINK);FORWARD;
   PROCEDURE VERCONF(L1,L2 :LINK);FORWARD;

   PROCEDURE SIMULA;
   VAR T,I,K,S,M,AUX : INTEGER ; INFO : ARRAY[1..4] OF INTEGER;
       VET : ARRAY[1..4] OF JOGADA; LOC : ARRAY[1..4] OF LINK;
       DA,DECISIVA : BOOLEAN ;
       L1,L3,L2,Q,P,POSICAO : LINK;

   BEGIN %0\
%  WRITELN(TTY,' [SIMULA]');BREAK(TTY);TRANSFORMA;\
    IF PLOCAL = 0 THEN
    BEGIN %1\
     L := NIL;
     OK := FALSE
```

```
END %1\
                ELSE
BEGIN %1\
COMPARA;
IF IGUAL = FALSE THEN
BEGIN %2\
 POSICAO := NIL;
 JOGADOR := 'O' ; DECIDE;
 FOR K := 1 TO PLOCAL DO LOC[K] := LOCAL[K];
 P := STACK[1].PARA;
 FOR K := 1 TO PLOCAL DO
 BEGIN %3\
  Q:= LOC[K];
  JOGADOR := 'X'; JOGA(Q,P) ; DECIDE;
  CONDENSA(T); INFO[K] := T;
  JOGA(P,Q)
 END; %3\
 IF PLOCAL > 1 THEN
 BEGIN %3\
  FOR K := 1 TO PLOCAL-1 DO
  BEGIN %4\
   FOR M :=K+1 TO PLOCAL DO
   BEGIN %5\
    IF INFO[K] > INFO[M] THEN
    BEGIN %6\
     AUX := INFO[K]; Q:= LOC[K];
     INFO[K] := INFO[M]; LOC[K] := LOC[M];
     INFO[M]:= AUX ; LOC[M] := Q
    END %6\
   END %5\
  END %4\
 END; %3\
 I := PLOCAL +1;
 REPEAT
  BEGIN %3\
   OK := TRUE;
   I := I - 1; M:= 0;
   DECISIVA := TRUE;
   IF INFO[I] = 1 THEN
   BEGIN %4\
    Q:= LOC[I];
    VERCONF(Q,P);
    IF EXISTE = TRUE THEN
    BEGIN %4.1\
     DA := FALSE;
     DECISIVA := FALSE
    END %4.1\
                        ELSE
    BEGIN %4.1\
     DA := TRUE ;
     JOGADOR := 'X'; JOGA(Q,P);GUARDA;
     DECIDE;
     L1 := STACK[1].PARA; S:=0;
     REPEAT
      BEGIN %5\
       S:= S+1;L2:=L1^.PROXIMO[S];
       IF L2 # NIL THEN
       BEGIN %6\
        IF L2^.QUEM = 'O' THEN
        BEGIN %7\
```

```
      JOGADOR := 'O';JOGA(L2,L1);
      JOGADOR := 'X'; DECIDE;
      IF PSTACK = 0 THEN
      BEGIN %7.5\
       JOGADOR := 'O'; DECIDE;
       CONDENSA(T);
       IF T > 1 THEN
       BEGIN %8\
        DA := FALSE;
        DECISIVA := FALSE
       END %8\
              ELSE
       BEGIN %8\
        IF T = 1 THEN
        BEGIN %9\
         M:= M+1;
         VET[M].DE := L2;
         VET[M].PARA := L1
        END %9\
                ELSE
        BEGIN %9\
         IF OK = TRUE THEN
         BEGIN %10\
          GUARDA; JOGADOR := 'X'; ANALISA(JOGADOR);
          IF OK = FALSE THEN DECISIVA := FALSE;
          HIST := HIST - 1
         END %10\
        END %9\
       END %8\
      END; %7.5\
      JOGADOR := 'O';JOGA(L1,L2)
     END %7\
    END %6\
   END %5\
  UNTIL (S=4) OR (DA = FALSE);
IF DA = FALSE THEN
BEGIN %5\
 M:=0;JOGADOR := 'X';JOGA(P,Q);HIST:=HIST-1
END %5\
          ELSE
BEGIN %5\
 IF M # 0 THEN
 BEGIN %6\
  S:= 0;
  REPEAT
   BEGIN %7\
    S:=S+1;
    L1:=VET[S].DE;L2:=VET[S].PARA;
    JOGADOR := 'O'; JOGA(L1,L2);GUARDA;
    DECIDE;
    L3 := STACK[1].PARA;PROCURA(L3);
    SIMULA(QUAL);
    IF QUAL = NIL THEN
    BEGIN %8\
        DA := FALSE;
        DECISIVA := FALSE
     END %8\
                ELSE
     BEGIN %8\
      DA := TRUE;
```

```
            IF OK = FALSE THEN DECISIVA := FALSE
           END; %8\
          JOGADOR := 'O';JOGA(L2,L1);HIST:=HIST-1
         END %7\
        UNTIL (DA = FALSE) OR (S=M)
       END;%6\ ;JOGADOR :='X'; JOGA(P,Q);HIST:=HIST-1
      END %5\
     END %4.1\
    END %4\
                    ELSE
    BEGIN %4\
     IF INFO[I] > 1 THEN DA := TRUE
                    ELSE
     BEGIN %5\
      DECISIVA := FALSE;
      Q := LOC[I]; LEGAL(Q,P);
      DA := JLEGAL
     END %5\
    END %4\
   END; %3\
   IF DA = TRUE THEN POSICAO := LOC[I]
  UNTIL (I=1) OR (DECISIVA = TRUE);
  IF DECISIVA = FALSE THEN OK := FALSE ELSE OK := TRUE;
  IF POSICAO # NIL THEN L := POSICAO ELSE  L := NIL
  END %2\
        ELSE
  BEGIN %2\
   L := LOC[PLOCAL]; REPETICAO :=TRUE; OK := FALSE
  END %2\
 END %1\
%       WRITELN(TTY,' FIM DA SIMULA');BREAK(TTY);TRANSFORMA\
 END; %0\

%21\  PROCEDURE PENSA ; FORWARD ;

 PROCEDURE LEGAL  ;
 VAR P,Q,R,S :LINK ; I,X,Y,Z,LINHA,COLUNA : INTEGER;
     VET : ARRAY[1..16] OF ENDERECO;
 BEGIN %0\
%       WRITELN(TTY,' [LEGAL] ');BREAK(TTY);TRANSFORMA;\
 VERCONF(L1,L2);
 IF EXISTE = FALSE THEN
 BEGIN %0.5\
  COMPARA;
  IF IGUAL = FALSE THEN
  BEGIN %1\
%\    JLEGAL := FALSE; JOGADOR := 'X';JOGA(L1,L2);GUARDA;
  JOGADOR := 'O';DECIDE;
  IF PSTACK = 0 THEN
  BEGIN %2\
   ANALISA(JOGADOR);
   IF OK = FALSE THEN
   BEGIN %3\
    IF J=0 THEN
    BEGIN %4\
     IF QI = 0 THEN
     BEGIN %5\
      JLEGAL := TRUE
     END %5\
                ELSE
```

```
          BEGIN %5\
           PENSA
          END %5\
         END %4\
               ELSE
         BEGIN %4\
          IF VETOR[J].POSS=1 THEN
          BEGIN %5\
           X:=0;Z:=J;
           FOR I := 1 TO Z DO
           BEGIN %6\
            VET[I].LIN := VETOR[I].LIN;
            VET[I].COL := VETOR[I].COL;
            VET[I].POSS := VETOR[I].POSS;
            VET[I].POINT := VETOR[I].POINT
           END; %6\
           REPEAT
            BEGIN %6\
             X:=X+1;
             LINHA := VET[X].LIN;
             COLUNA := VET[X].COL;
             S:=TAB[LINHA,COLUNA];
             Y := VET[X].POINT;JOGADOR :='O';
             R:= S^.PROXIMO[Y];
             JOGADOR := 'O';JOGA(S,R);DECIDE;GUARDA;
             P :=STACK[1].PARA;
             PROCURA(P);
             SIMULA(QUAL);
             IF QUAL # NIL THEN JLEGAL :=TRUE ELSE JLEGAL := FALSE;
             JOGADOR := 'O';JOGA(R,S);HIST :=HIST-1
            END %6\
           UNTIL (X=Z) OR (JLEGAL=FALSE);
           IF JLEGAL = TRUE THEN
           BEGIN %6\
            PENSA
           END %6\
          END %5\
         END %4\
        END %3\
%\     END;  %2\ JOGADOR := 'X'; JOGA(L2,L1);HIST := HIST-1
       END %1\
             ELSE
       BEGIN %1\
        JLEGAL := TRUE; REPETICAO :=TRUE
       END %1\
      END %0.5\
                    ELSE
      BEGIN %0.5\
       JLEGAL := FALSE
      END %0.5\
%      WRITELN(TTY,' FIM DA LEGAL');BREAK(TTY);TRANSFORMA\
       END; %0\

%22\   PROCEDURE CONTA;
       VAR I,LINHA,COLUNA : INTEGER; R,S : LINK;
       BEGIN %1\
        MEU := 0; SEU := 0;
        FOR LINHA := 1 TO 5 DO
        BEGIN %2\
         FOR COLUNA := 1 TO 4 DO
```

```
      BEGIN %3\
       R := TAB[LINHA,COLUNA];
       IF R^.QUEM  =  'X' THEN
       BEGIN %4\
        FOR I := 1 TO 4 DO
        BEGIN %5\
         S := R^.PROXIMO[I];
         IF S # NIL THEN
         BEGIN %6\
          IF S^.QUEM = ' ' THEN MEU := MEU + 1
         END %6\
        END %5\
       END %4\
                          ELSE
       BEGIN %4\
        IF R^.QUEM = 'O' THEN
        BEGIN %5\
         FOR I:= 1 TO 4 DO
         BEGIN %6\
          S := R^.PROXIMO[I];
          IF S # NIL THEN
          BEGIN %7\
           IF S^.QUEM = ' ' THEN SEU := SEU + 1
          END %7\
         END %6\
        END %5\
       END %4\
      END %3\
     END %2\
    END; %1\

%23\  PROCEDURE TROCA(I,K : INTEGER);
    VAR AUX1,AUX2,AUX3,AUX4,AUX6 : INTEGER;
        AUX5 :LINK;
    BEGIN %1\
     AUX1 := MOVE[I].LIN; AUX2 := MOVE[I].COL;
     AUX3 := MOVE[I].SSEU; AUX4 := MOVE[I].SALDO;
     AUX5 := MOVE[I].POINT; AUX6 := MOVE[I].INFO;
     MOVE[I].LIN := MOVE[K].LIN;MOVE[I].COL := MOVE[K].COL;
     MOVE[I].SSEU := MOVE[K].SSEU; MOVE[I].SALDO := MOVE[K].SALDO;
     MOVE[I].POINT := MOVE[K].POINT; MOVE[I].INFO := MOVE[K].INFO;
     MOVE[K].LIN := AUX1 ; MOVE[K].COL := AUX2;
     MOVE[K].SSEU := AUX3 ; MOVE[K].SALDO := AUX4;
     MOVE[K].POINT := AUX5 ; MOVE[K].INFO := AUX6
    END; %1\

%24\  PROCEDURE ORDENA;
    VAR I,K : INTEGER;
    BEGIN %1\
     IF PMOVE > 1 THEN
     BEGIN %1.5\
     FOR I := 1 TO PMOVE -1 DO
     BEGIN %2\
      FOR K := I+1 TO PMOVE DO
      BEGIN %3\
       IF MOVE[I].SALDO > MOVE[K].SALDO THEN TROCA(I,K)
      END %3\
     END; %2\
     FOR I := 1 TO PMOVE -1 DO
     BEGIN%2\
```

```
        FOR K := I+1 TO PMOVE DO
        BEGIN %3\
         IF MOVE[I].SALDO = MOVE[K].SALDO THEN
         BEGIN %4\
          IF MOVE[I].SSEU < MOVE[K].SSEU THEN TROCA(I,K)
         END %4\
        END %3\
       END; %2\
       FOR I := 1 TO PMOVE -1 DO
       BEGIN %2\
        FOR K:= I+1 TO PMOVE DO
        BEGIN %3\
         IF MOVE[I].SALDO= MOVE[K].SALDO THEN
         BEGIN %4\
          IF MOVE[I].SSEU = MOVE [K].SSEU THEN
           BEGIN %5\
           IF MOVE[I].INFO > MOVE[K].INFO THEN TROCA(I,K)
          END %5\
         END %4\
        END %3\
       END %2\
      END %1.5\
      END; %1\

%25\  PROCEDURE SALDO;
      VAR LINHA,COLUNA,I : INTEGER; P,Q : LINK;
      BEGIN %1\
       PMOVE := 0;
       FOR LINHA := 1 TO 5 DO
       BEGIN %2\
        FOR COLUNA := 1 TO 4 DO
        BEGIN %3\
         P := TAB[LINHA,COLUNA];
         IF P^.QUEM = 'X' THEN
         BEGIN %4\
          FOR I := 1 TO 4 DO
          BEGIN %5\
           Q := P^.PROXIMO[I];
           IF Q # NIL THEN
           BEGIN %6\
            IF Q^.QUEM = ' ' THEN
            BEGIN %7\
             JOGADOR := 'X';
             JOGA(P,Q);
             JOGADOR := 'O' ; DECIDE;
             IF PSTACK =0 THEN
             BEGIN %8\
              CONTA;
              MEU := MEU- SEU;
              PMOVE := PMOVE +1;
              MOVE[PMOVE].LIN := LINHA;
              MOVE[PMOVE].COL := COLUNA;
              MOVE[PMOVE].SSEU := SEU ;
              MOVE[PMOVE].SALDO := MEU;
              MOVE[PMOVE].POINT := Q;
              MOVE[PMOVE].INFO := Q^.INFO
```

```
          END %4\
        END %3\
      END %2\
    END; %1\

%26\  PROCEDURE ESCOLHEJOGADA;
      VAR Z,LINHA,COLUNA,I : INTEGER; F1,F2 :LINK;
          MOV : ARRAY [1..16] OF DECISAO;
      BEGIN %1\
%     WRITELN(TTY,' [ESCOLHEJOGADA]');BREAK(TTY);TRANSFORMA;\
       JLEGAL := FALSE; Z:= PMOVE;
       IF Z <> 0 THEN
       BEGIN %1.5\
       FOR I:= 1 TO PMOVE DO
       BEGIN %2\
        MOV[I].LIN := MOVE[I].LIN;
        MOV[I].COL := MOVE[I].COL;
        MOV[I].SSEU := MOVE[I].SSEU;
        MOV[I].SALDO := MOVE[I].SALDO;
        MOV[I].POINT := MOVE[I].POINT;
        MOV[I].INFO := MOVE[I].INFO
       END; %2\
       I := Z+1;
       REPEAT
        BEGIN %2\
        I := I-1;
        LINHA := MOV[I].LIN ; COLUNA := MOV[I].COL;
        F1 := TAB[LINHA,COLUNA] ; F2 := MOV[I].POINT;
        LEGAL(F1,F2)
%     WRITELN(TTY,' VOLTA DA LEGAL NA ESCOLHE');BREAK(TTY);TRANSFORMA\
        END %2\
       UNTIL (JLEGAL = TRUE) OR (I=1);
       IF JLEGAL = TRUE THEN
       BEGIN %2\
         U := F1; V :=F2
       END %2\
                          ELSE
%**** SE ENTRAR AQUI O ADVERSARIO PODERA GANHAR O JOGO ***\
       BEGIN %2\
        LINHA := MOV[Z].LIN;
        COLUNA := MOV[Z].COL;
        U := TAB[LINHA,COLUNA];
        V := MOV[Z].POINT
       END %2\
      END %1.5\
                      ELSE
      BEGIN %1.5\
       U := NIL
      END %1.5\
%     WRITELN(TTY,' FIM DA ESCOLHEJOGADA');BREAK(TTY);TRANSFORMA\
      END; %1\

%27\  PROCEDURE PENSA;
      VAR S,LINHA,COLUNA : INTEGER; U,V,P,Q : LINK;
          MOV : ARRAY[1..16] OF DECISAO;
      BEGIN %1\
%     WRITELN(TTY,' [PENSA]');BREAK(TTY);TRANSFORMA;\
       REC := REC + 1;
       IF REC <= QI THEN
```

```
        BEGIN %2\
         LINHA := 0 ; JLEGAL := TRUE;
         REPEAT
         BEGIN %3\
          LINHA := LINHA + 1;
          COLUNA := 0;
          REPEAT
          BEGIN %4\
           COLUNA := COLUNA + 1;
           P := TAB[LINHA,COLUNA];
           IF P^.QUEM = 'O' THEN
           BEGIN %5\
            S := 0;
            REPEAT
            BEGIN %6\
             S := S+ 1;
             Q := P^.PROXIMO[S];
             IF Q <> NIL THEN
             BEGIN %7\
              IF Q^.QUEM = ' ' THEN
              BEGIN %8\
%\             JOGADOR := 'O'; JOGA(P,Q);
               DECIDE;
               IF PSTACK = 0 THEN
               BEGIN %8.1\
                JOGADOR := 'X' ; DECIDE;
                IF PSTACK <>0 THEN
                BEGIN %9\
                 JOGADOR := 'X';
                 ANALISA(JOGADOR);
                 IF OK = FALSE THEN
                 BEGIN %10\
                  SALDO ; ORDENA;
                  ESCOLHEJOGADA
                 END %10\
                END %9\
               END; %8.1\
%\             JOGADOR := 'O'; JOGA(Q,P)
              END %8\
             END %7\
            END %6\
           UNTIL (S=4) OR (JLEGAL = FALSE)
           END %5\
          END %4\
         UNTIL (COLUNA = 4) OR (JLEGAL = FALSE)
        END %3\
        UNTIL (LINHA=5) OR (JLEGAL = FALSE);
        REC := REC -1
       END  %2\
                        ELSE
        BEGIN %2\
         JLEGAL := TRUE;
         REC := REC -1
        END %2\
%       WRITELN(TTY,' FIM DA PENSA');BREAK(TTY);TRANSFORMA\
       END; %1\

%28\   PROCEDURE EMPILHA(L1,L2 :LINK) ;FORWARD;
       PROCEDURE TESTAJOGADA;
       VAR P,Q,L : LINK ; I : INTEGER;
```

```
        BEGIN %1\
         OK := FALSE;
         IF (LD <= 5) AND (CD <= 4) THEN
         BEGIN %2\
          IF (LP <= 5) AND (CP <= 4) THEN
          BEGIN %3\
           P := TAB[LD,CD] ; Q:= TAB[LP,CP];
           IF P^.QUEM  = 'O' THEN
           BEGIN %4\
            FOR I := 1 TO 4 DO
            BEGIN %5\
             L := P^.PROXIMO[I];
             IF L # NIL THEN
             BEGIN %6\
              IF L^.QUEM = ' ' THEN
              BEGIN %7\
              IF Q = L THEN
               BEGIN %8\
                OK := TRUE;
                JOGADOR := 'O'; JOGA(P,Q);EMPILHA(P,Q)
               END %8\
              END %7\
             END %6\
            END %5\
           END %4\
          END %3\
         END %2\
        END; %1\

%29\    PROCEDURE CONTROLE_TELA;
        BEGIN
         BEL := CHR(007B);
         LF := CHR(012B);
         CR := CHR(015B);
         CAN := CHR(030B);
         SUB := CHR(032B);
         GS := CHR(035B);
         RS := CHR(036B);
         US := CHR(037B);
        END;

%30\    PROCEDURE APAGA_CANTO;
        VAR     I : INTEGER;
        BEGIN
         IF VISOR = TRUE THEN
         BEGIN
          WRITE(TTY,GS);BREAK(TTY); WRITE(TTY,GS);BREAK(TTY);
          WRITE(TTY,RS);BREAK(TTY);
          FOR I := 1 TO 10 DO
          BEGIN
           WRITELN(TTY,' ':20);BREAK(TTY);
          END;
          WRITE(TTY,GS);BREAK(TTY); WRITE(TTY,GS);BREAK(TTY);
         END
         ELSE
         BEGIN
          WRITELN(TTY);BREAK(TTY);
          WRITELN(TTY);BREAK(TTY);
         END;
```

```
        END;

%31\    PROCEDURE NUMERO(VAR NUM : INTEGER);
        VAR CAR : CHAR; I : INTEGER;
        BEGIN
        LOOP
                I := 1;
                NUM := 0;
                READLN(TTY);BREAK(TTY);
                WHILE NOT EOLN(TTY) DO
                BEGIN
                        READ(TTY,CAR);BREAK(TTY);
                        IF CAR IN ['0'..'9'] THEN
                        BEGIN
                                NUM := (NUM * 10) + (ORD(CAR) - 60B);
                                I := I + 1;
                        END;
                END;
        EXIT IF I # 1;
                BEGIN
                        WRITE(TTY,'BATA UM NUMERO :');BREAK(TTY);
                END;
        END%LOOP\;
        END;

%32\    PROCEDURE SIM_NAO(VAR S_N : CHAR);
        VAR ERRO : BOOLEAN; CAR : CHAR; I : INTEGER;
        BEGIN
                REPEAT
                        ERRO := FALSE;
                        WRITE(TTY,'BATA ''S'' OU ''N'':');BREAK(TTY);
                        READLN(TTY);BREAK(TTY);
                        I := 1;
                        WHILE NOT EOLN(TTY) DO
                        BEGIN
                                READ(TTY,CAR);BREAK(TTY);
                                CASE CAR OF
                                        'S','N': I := I + 1;
                                        ' ' : ;
                                        OTHERS : ERRO := TRUE
                                END;
                        END;
                UNTIL ( I = 2 ) AND ( ERRO = FALSE );
                S_N := CAR;
        END;

%33\    PROCEDURE LEJOGADA;
        VAR     I,LINHA,COLUNA,ND,NP : INTEGER;
        BEGIN %1\
         APAGA_CANTO;
         WRITELN(TTY, ' ';10,'     ENTRE COM A SUA JOGADA');BREAK(TTY);
         LOOP
          BEGIN %2\
          WRITE(TTY,'DE   => ');BREAK(TTY);
          NUMERO(ND);
          CD := ( ( ND - 1 ) MOD 4 ) + 1;
          LD := ( ( ND - 1 ) DIV 4 ) + 1;
          WRITE(TTY,'PARA => ');BREAK(TTY);
```

```
        NUMERO(NP);
        CP := ( ( NP - 1 ) MOD 4 ) + 1;
        LP := ( ( NP - 1 ) DIV 4 ) + 1;
        TESTAJOGADA;
        END%2\;
        EXIT IF ( OK = TRUE );
        BEGIN %2\
         APAGA_CANTO;
         WRITELN(TTY,' '110,'*** JOGADA INVALIDA ***');BREAK(TTY);
        END %2\
       END%LOOP\;
       FOR LINHA := 1 TO 5 DO
       BEGIN
        FOR COLUNA := 1 TO 4 DO
        BEGIN
         JOGO_VELHO[(4*(LINHA-1)+(COLUNA-1))+1] := TAB[LINHA,COLUNA]",QUEM;
        END;
       END;
       IF VISOR = TRUE THEN
       BEGIN
        WRITE(TTY,GS);BREAK(TTY); WRITE(TTY,GS);BREAK(TTY);
       END;
      END; %1\

%34\  PROCEDURE ONDE_JOGUEI;
      VAR     LINHA,COLUNA,DE,PARA,I : INTEGER;
      BEGIN
       APAGA-CANTO;
       WRITELN(TTY,'     MINHA JOGADA');BREAK(TTY);
       FOR LINHA := 1 TO 5  DO
       BEGIN
        FOR COLUNA := 1 TO 4 DO
        BEGIN
         I :=((4*(LINHA-1))+(COLUNA-1))+1;
         IF (JOGO_VELHO[I] = 'X') AND (TAB[LINHA,COLUNA]",QUEM = ' ') THEN DE := I;
         IF (JOGO_VELHO[I] = ' ') AND (TAB[LINHA,COLUNA]",QUEM = 'X') THEN PARA := I;
        END;
       END;
       WRITELN(TTY,'DE   => ',DE:2);BREAK(TTY);
       WRITELN(TTY,'PARA => ',PARA:2);BREAK(TTY);
       IF VISOR = TRUE THEN
       BEGIN
        WRITE(TTY,GS);BREAK(TTY); WRITE(TTY,GS);BREAK(TTY);
       END;
      END;

%35\  PROCEDURE DA_UM_TEMPO;
      BEGIN
       IF VISOR = TRUE THEN
        BEGIN
        WRITE(TTY,'<-');BREAK(TTY);
        READLN(TTY);BREAK(TTY);
        WHILE NOT(EOLN(TTY)) DO
        BEGIN
         READ(TTY,LIBERA);BREAK(TTY);
        END;
        END;
      END;

%36\  PROCEDURE  APAGA;
```

```
BEGIN
 IF VISOR = TRUE THEN
 BEGIN
  WRITE(TTY,GS);BREAK(TTY);WRITE(TTY,GS);BREAK(TTY);
  WRITE(TTY,US);BREAK(TTY);WRITE(TTY,US); BREAK(TTY);
  WRITE(TTY,GS);BREAK(TTY);WRITE(TTY,GS);BREAK(TTY);
 END
 ELSE
 BEGIN
  WRITELN(TTY);BREAK(TTY);WRITELN(TTY);BREAK(TTY);
 END;
END;

%37\  PROCEDURE MOSTRA_QUADRO_VT;
      VAR     CASA,I,LINHA,COLUNA : INTEGER;
      BEGIN %1\
       WRITE(TTY,GS);BREAK(TTY);WRITE(TTY,GS);BREAK(TTY);
       CASA := 1;
       WRITELN(TTY);BREAK(TTY);
       FOR LINHA := 1 TO 5 DO
       BEGIN %2\
        FOR I := 1 TO 2 DO
        BEGIN%3\
         WRITELN(TTY,' ':20,LIN);BREAK(TTY);
        END%3\;
        WRITE(TTY,' ':20);BREAK(TTY);
        FOR COLUNA := 1 TO  4 DO
        BEGIN%3\
         WRITE(TTY,'|',' ':5,CASA:2);BREAK(TTY);
         CASA := CASA + 1;
        END%3\;
        WRITE(TTY,'|');BREAK(TTY);
        IF LINHA # 5 THEN
        BEGIN%3\
          WRITELN(TTY);BREAK(TTY);
          WRITELN(TTY,' ':20,SEPAR);BREAK(TTY);
        END%3\;
       END%2\;
       WRITE(TTY,SUB);BREAK(TTY);WRITE(TTY,CR);BREAK(TTY);
       WRITELN(TTY,'EU JOGO COM = X =');BREAK(TTY);
       WRITE(TTY,'VOCE JOGA COM = O =');BREAK(TTY);
       WRITE(TTY,GS);BREAK(TTY);WRITE(TTY,GS);BREAK(TTY);
      END%1\;

%38\  PROCEDURE MOSTRA_PECA_VT;
      VAR I,LINHA,COLUNA : INTEGER;
      BEGIN%1\
       WRITE(TTY,GS);BREAK(TTY);WRITE(TTY,GS);BREAK(TTY);
       FOR I := 1 TO 2 DO
       BEGIN
        WRITE(TTY,LF);BREAK(TTY);
       END;
       FOR LINHA := 1 TO 5 DO
       BEGIN
        FOR I := 1 TO 24 DO
        BEGIN
         WRITE(TTY,CAN);BREAK(TTY);
        END;
        FOR COLUNA := 1 TO 4 DO
        BEGIN
```

```
      WRITE(TTY,TAB[LINHA,COLUNA]^.QUEM);BREAK(TTY);
      IF COLUNA # 4 THEN
      BEGIN
       FOR I := 1 TO 7 DO
       BEGIN
        WRITE(TTY,CAN);BREAK(TTY);
       END;
      END;
     END;
     IF LINHA # 5 THEN
     BEGIN
      FOR I := 1 TO 4 DO
      BEGIN
       WRITE(TTY,LF);BREAK(TTY);
      END;
     END;
     WRITE(TTY,CR);BREAK(TTY);
     END;
   WRITE(TTY,GS);BREAK(TTY);WRITE(TTY,GS);BREAK(TTY);
  END;

%39\ PROCEDURE MOSTRA;
  VAR CASA,LINHA,COLUNA : INTEGER;
  BEGIN %1\
  CASA := 1;
   FOR LINHA := 1 TO 5 DO
   BEGIN %2\
    WRITELN(TTY,' ':20,LIN);BREAK(TTY);
    WRITE(TTY,' ':20);BREAK(TTY);
    FOR COLUNA := 1 TO 4 DO
    BEGIN %3\
     WRITE(TTY,'|   ',TAB[LINHA,COLUNA]^.QUEM,'   ');BREAK(TTY)
    END %3\;
    WRITELN(TTY,'|');BREAK(TTY);
    WRITE(TTY,' ':20);BREAK(TTY);
    FOR COLUNA := 1 TO 4 DO
    BEGIN
     WRITE(TTY,'|',' ':5,CASA:2);BREAK(TTY);
     CASA := CASA + 1;
    END;
    WRITE(TTY,'|');BREAK(TTY);
    IF LINHA # 5 THEN
    BEGIN %3\
     WRITELN(TTY);BREAK(TTY);
     WRITELN(TTY,' ':20,SEPAR);BREAK(TTY)
    END %3\;
   END%2\;
  END %1\;

%40\ PROCEDURE MOSTRA_JOGO;
  BEGIN
   WRITE(TTY,BEL,BEL);BREAK(TTY);
   IF VISOR = TRUE THEN BEGIN
    MOSTRA_PECA_VT;
    END
    ELSE
    BEGIN
     WRITE(TTY,'QUER VER O TABULEIRO ?? ');BREAK(TTY);
       SIM_NAO(VER_QUADRO);
       CASE VER_QUADRO OF
```

```
        'S' : BEGIN
         WRITELN(TTY);BREAK(TTY);WRITELN(TTY);BREAK(TTY);
         MOSTRA;
         WRITELN(TTY);BREAK(TTY);WRITELN(TTY);BREAK(TTY);
         END;
        'N' : BEGIN
        WRITELN(TTY);BREAK(TTY);
       END
       END%CASE\;
     END;
   END;

%41\  PROCEDURE FAZ;
   VAR I,LINHA,COLUNA,CASA,NUM_B,NUM_X,NUM_O : INTEGER;
       NUMERO : CHAR;
       ERRO : BOOLEAN;
   BEGIN
    WRITELN(TTY,' ':20,'FACA A SEGUINTE ASSOCIACAO');BREAK(TTY);
    WRITELN(TTY,' ':25,'B ==> POSICAO COM BRANCO');BREAK(TTY);
    WRITELN(TTY,' ':25,'X ==> POSICAO COM O SIMBOLO X');BREAK(TTY);
    WRITELN(TTY,' ':25,'O ==> POSICAO COM O SIMBOLO O');BREAK(TTY);
    LOOP
     CASA := 1;
     NUM_B := 0;
     NUM_X := 0;
     NUM_O := 0;
     FOR LINHA := 1 TO 5 DO
     BEGIN
      FOR COLUNA := 1 TO 4 DO
      BEGIN
        WRITE(TTY,'ENTRE COM A POSICAO ',CASA:2,' : ');BREAK(TTY);
        LOOP
         ERRO := FALSE;
         READLN(TTY);BREAK(TTY);
         I := 1;
         WHILE NOT EOLN(TTY) DO
         BEGIN
          READ(TTY,NUMERO);BREAK(TTY);
          CASE NUMERO OF
           'X','B','O' : I := I + 1;
           ' ' : ;
           OTHERS : ERRO := TRUE
           END%CASE\;
         END;
        EXIT IF ( I = 2 ) AND ( ERRO = FALSE );
         WRITE(TTY,'DATA ''X'',''O'' OU ''B'' : ');BREAK(TTY);
        END%LOOP\;
        CASE NUMERO OF
           'B' : BEGIN
              TAB[LINHA,COLUNA]^.QUEM := ' ';
              NUM_B := NUM_B + 1;
           END;
           'X' : BEGIN
              TAB[LINHA,COLUNA]^.QUEM := 'X';
              NUM_X := NUM_X + 1;
           END;
           'O' : BEGIN
              TAB[LINHA,COLUNA]^.QUEM := 'O';
              NUM_O := NUM_O + 1;
           END
```

```
        END;
       CASA := CASA + 1;
      END;
      IF VISOR = TRUE THEN
       BEGIN
        WRITE(TTY,GS);BREAK(TTY);WRITE(TTY,GS);BREAK(TTY);
        FOR I := 1 TO 4 DO
        BEGIN
         WRITE(TTY,LF);BREAK(TTY);
        END;
        WRITE(TTY,US);BREAK(TTY);WRITE(TTY,US);BREAK(TTY);
       END;
      END;
     EXIT IF ( NUM_X = NUM_O ) AND ( NUM_X = 4 );
      BEGIN
       WRITELN(TTY);BREAK(TTY);WRITELN(TTY);BREAK(TTY);
       WRITELN(TTY,'REENTRE NOVAMENTE');BREAK(TTY);
       WRITELN(TTY,'HA''',NUM_X:2,' SIMBOLOS = X =');BREAK(TTY);
       WRITELN(TTY,'HA''',NUM_O:2,' SIMBOLOS = O =');BREAK(TTY);
       WRITELN(TTY,'HA''',NUM_B:2,' CASAS EM BRANCO');BREAK(TTY);
       WRITELN(TTY);BREAK(TTY); WRITELN(TTY);BREAK(TTY);
       DA_UM_TEMPO;
       IF VISOR = TRUE THEN
        BEGIN
         WRITE(TTY,GS);BREAK(TTY);WRITE(TTY,GS);BREAK(TTY);
         FOR I := 1 TO 4 DO
          BEGIN
           WRITE(TTY,LF);BREAK(TTY);
          END;
          WRITE(TTY,CR);BREAK(TTY);
          WRITE(TTY,US);BREAK(TTY);WRITE(TTY,US);BREAK(TTY);
        END;
       END;
       END%LOOP\;
      END;

%42\  PROCEDURE EXPLIQUE;
      BEGIN %1\
       WRITELN(TTY,' ':17,'*****************************************');BREAK(TTY);
       WRITELN(TTY,' ':17,'*                                       *');BREAK(TTY);
       WRITELN(TTY,' ':17,'* --- R E G R A S  D O  J O G O  --- *');BREAK(TTY);
       WRITELN(TTY,' ':17,'*                                       *');BREAK(TTY);
       WRITELN(TTY,' ':17,'*****************************************');BREAK(TTY);
       WRITELN(TTY,' ':14,'0.  JOGADORES');BREAK(TTY);
       WRITELN(TTY,' ':14,'    ----------');BREAK(TTY);
       WRITELN(TTY,' ':17,'==>  EU JOGO COM O SIMBOLO = X = ');BREAK(TTY);
       WRITELN(TTY,' ':17,'==> VOCE JOGA COM O SIMBOLO = O =');BREAK(TTY);
       WRITELN(TTY,' ':14,'1.  MOVIMENTOS PERMITIDOS');BREAK(TTY);
       WRITELN(TTY,' ':14,'    --------------------');BREAK(TTY);
       WRITELN(TTY,' ':17,'==>  MOVER SUA PECA DE UMA POSICAO NA HORIZONTAL  OU ');BREAK(TTY);
       WRITELN(TTY,' ':17,'==>  MOVER SUA PECA DE UMA POSICAO NA VERTICAL  ');BREAK(TTY);
       WRITELN(TTY,' ':17,'OBSERVACAO :  NAO E PERMITIDO ANDAR NA DIAGONAL');BREAK(TTY);
       WRITELN(TTY,' ':17,'              JOGA-SE ALTERNADAMENTE');BREAK(TTY);
       WRITELN(TTY,' ':17,'              NAO E PERMITIDO PULAR PECAS');BREAK(TTY);
       WRITELN(TTY,' ':14,'2.  VENCE AQUELE QUE : ');BREAK(TTY);
       WRITELN(TTY,' ':14,'    ---------------');BREAK(TTY);
       WRITELN(TTY,' ':17,'==>  PRIMEIRO AGRUPAR 3(TRES) DE SUAS PECAS NA ');BREAK(TTY);
       WRITE(TTY,' ':17,'==> HORIZONTAL VERTICAL OU DIAGONAL');BREAK(TTY);
       DA-UM-TEMPO;APAGA;
       WRITELN(TTY,'  ':10,'*** ESTA E A CONFIGURACAO INICIAL DO TABULEIRO ***');BREAK(TTY);
```

```
 IF VISOR = FALSE THEN
 BEGIN
  WRITELN(TTY);BREAK(TTY);WRITELN(TTY);BREAK(TTY);
 END;
 MOSTRA;
 BA_UM_TEMPO;APAGA;
 WRITE(TTY,'VOCE QUER VER AS REGRAS NOVAMENTE ?    ');BREAK(TTY);
 SIM_NAO(EXPLICO);
END; %1\

%43\  PROCEDURE TICTAC;

BEGIN %1\
 APAGA;
 WRITELN(TTY);BREAK(TTY);WRITELN(TTY);BREAK(TTY);
 WRITELN(TTY,' ':17,'TTTTT     AAA      CCCC   TTTTT    III     CCCC');BREAK(TTY);
 WRITELN(TTY,' ':17,'  T     A   A    C         T       I     C');BREAK(TTY);
 WRITELN(TTY,' ':17,'  T     A   A    C         T       I     C');BREAK(TTY);
 WRITELN(TTY,' ':17,'  T     A   A    C         T       I     C');BREAK(TTY);
 WRITELN(TTY,' ':17,'  T     AAAAA    C         T       I     C');BREAK(TTY);
 WRITELN(TTY,' ':17,'  T     A   A    C         T       I     C');BREAK(TTY);
 WRITELN(TTY,' ':17,'  T     A   A     CCCC     T      III     CCCC');BREAK(TTY);
 WRITELN(TTY);BREAK(TTY);WRITELN(TTY);BREAK(TTY);
 WRITE(TTY,' ':25,'Q I ==> ');BREAK(TTY);
 NUMERO(QI);
 WRITELN(TTY);BREAK(TTY);WRITELN(TTY);BREAK(TTY);
 WRITE(TTY,' ':30,'B O A  S O R T E !!!!!!! ');BREAK(TTY);
 BA_UM_TEMPO;APAGA;
 WRITE(TTY,'DESEJA EXPLICACOES ?     ');BREAK(TTY);
 SIM_NAO(EXPLICO);
 REPEAT
 BEGIN %2\
  CASE EXPLICO OF
  'S' : BEGIN
   APAGA;
   EXPLIQUE;
  END;
  'N' : BEGIN
   WRITELN(TTY);BREAK(TTY);
   END
  END%CASE\
  END%2\;
  UNTIL ( EXPLICO = 'N' );
   APAGA;
   WRITE(TTY,'DESEJA HISTORICO EM ARQUIVO OUTPUT ?     ');BREAK(TTY);
   SIM_NAO(GRAVA);
  APAGA;
  WRITE(TTY,'DESEJA COMECAR DE ALGUMA CONFIGURACAO INICIAL ?     ');BREAK(TTY);
  SIM_NAO(ESPECIAL);
  CASE ESPECIAL OF
   'S' : BEGIN
    APAGA;
    FAZ;
   END;
   'N' : BEGIN
    WRITELN(TTY);BREAK(TTY);
    END
 END%CASE\;
 APAGA;
END; %1\
```

```
%44\    PROCEDURE LOAD;

        VAR I,J : INTEGER;

        BEGIN %1\
 %       WRITELN(TTY,'   [INICIO-LOAD]'); BREAK(TTY); \
         FOR I := 1 TO 312 DO
         BEGIN %2\
          MEMORIA[I] := 0
         END %2\;
         RESET(ENTRADA,'MEMORIDAT');
         J := 0;
         WHILE NOT EOF(ENTRADA) DO
         BEGIN %2\
          J := J + 1;
          MEMORIA[J] := ENTRADA";
          GET(ENTRADA)
         END %2\
 %       WRITELN(TTY,'   [FIM-LOAD]');BREAK(TTY); \
        END %1\;

%45\    PROCEDURE STORE;

        VAR I : INTEGER;

        BEGIN %1\
 %       WRITELN(TTY,'   [INICIO-STORE]');BREAK(TTY); \
         REWRITE(ENTRADA,'MEMORIDAT');
         FOR I := 1 TO 312 DO
         BEGIN %2\
          ENTRADA" := MEMORIA[I];
          PUT(ENTRADA)
         END %2\
 %       WRITELN(TTY,'   [FIM-STORE]');BREAK(TTY); \
        END %1\;

%46\    PROCEDURE CONFIGURA;
        VAR LINHA,COLUNA,K,X,Y : INTEGER;
            R : LINK;

        BEGIN %1\
 %       WRITELN(TTY,'   [INICIO-CONFIGURA]');BREAK(TTY); \
         X := 0 ; Y := 4 ;
         FOR LINHA := 1 TO 5 DO
         BEGIN %2\
          FOR COLUNA := 1 TO 4 DO
          BEGIN %3\
           R := TAB[LINHA,COLUNA];
           IF R".QUEM <> ' ' THEN
           BEGIN %4\
            IF R".QUEM = 'X' THEN
            BEGIN %5\
             X := X + 1;
             K := (LINHA-1)*4 + COLUNA;
             CONFIGURACAO[X] := K
            END %5\
                                ELSE
            BEGIN %5\
             Y := Y + 1 ;
```

```
          K := (LINHA - 1)*4 + COLUNA ;
          CONFIGURACAO[Y] := K
         END %5\
        END %4\
       END %3\
      END %2\
%     WRITELN(TTY,'  FIM-[CONFIGURA]');BREAK(TTY); \
     END %1\;

%47\ PROCEDURE PALABITE(X,Y,Z : INTEGER);
     VAR K,QUOCIENTE,RESTO : INTEGER;

     BEGIN %1\
%     WRITELN(TTY,'   INICIO-[PALABITE]');BREAK(TTY); \
      K := 560*(X-1) + 20*(Y-1) + Z ;
      QUOCIENTE := K DIV 36;
      RESTO := K MOD 36;
      IF RESTO = 0 THEN
      BEGIN %2\
       PALAVRA := QUOCIENTE;
       BITE := 36
      END %2\
                    ELSE
      BEGIN %2\
       PALAVRA := QUOCIENTE + 1;
       BITE := RESTO
      END %2\
%     WRITELN(TTY,'   FIM-[PALABITE]');BREAK(TTY); \
     END %1\;

%48\ PROCEDURE DECODIFICA;
     VAR NUMERO,I : INTEGER;

     BEGIN %1\
%     WRITELN(TTY,'   INICIO-[DECODIFICA]');BREAK(TTY); \
      NUMERO := MEMORIA[PALAVRA];
      IF NUMERO < 0 THEN MASCARA[1] := 1
                    ELSE MASCARA[1] := 0;
      IF (NUMERO = -34359738368) OR (NUMERO = 0) THEN
      BEGIN %2\
       FOR I := 2 TO 36 DO
       BEGIN %3\
        MASCARA[I] := 0
       END %3\
      END %2\
                                                 ELSE
      BEGIN %2\
       I := 37 ;
       IF NUMERO < 0 THEN NUMERO := NUMERO*(-1);
       REPEAT
        BEGIN %3\
         I := I-1;
         MASCARA[I] := NUMERO MOD 2;
         NUMERO := NUMERO DIV 2
        END %3\
       UNTIL (NUMERO = 0)
      END %2\
%     WRITELN(TTY,'   FIM-[DECODIFICA]');BREAK(TTY); \
     END %1\;
```

```
%49\    PROCEDURE CODIFICA;
        VAR NUM,EXPOENTE,I : INTEGER;

        BEGIN %1\
%        WRITELN(TTY,'   INICIO-[CODIFICA]');BREAK(TTY); \
         IF (MEMORIA[PALAVRA] = 0) AND (BITE = 1) THEN
         BEGIN %2\
          MEMORIA[PALAVRA] := -3435973868
         END %2\
                                                ELSE
         BEGIN %2\
          DECODIFICA;
          IF MASCARA[BITE] = 0 THEN
          BEGIN %3\
           MASCARA[BITE] := 1;
           I := 37 ; NUM := 0 ; EXPOENTE := 1 ;
           REPEAT
            BEGIN %4\
             I := I-1;
             NUM := NUM + MASCARA[I]*EXPOENTE;
             EXPOENTE := EXPOENTE*2
            END %4\
           UNTIL (I = 2);
           IF MASCARA[1] = 1 THEN MEMORIA[PALAVRA] := NUM*(-1)
                             ELSE MEMORIA[PALAVRA] := NUM;
          END %3\
         END %2\
%        WRITELN(TTY,'   FIM-[CODIFICA]');BREAK(TTY); \
        END %1\;


%50\    PROCEDURE MENSAGEM;

        BEGIN %1\
         WRITELN(TTY);BREAK(TTY);
         IF PPILHA = 0 THEN
         BEGIN %2\
          WRITELN(TTY,' ':5,'EXISTE E.G. : QUEM COMECA PERDE');
          BREAK(TTY)
         END %2\
                       ELSE
         BEGIN %2\
          WRITELN(TTY,' ':5,'EXISTE E.G : INICIE JOGANDO ');BREAK(TTY);
          WRITELN(TTY,' ':5,'COMO VOCE INICIOU');BREAK(TTY)
         END %2\
        END %1\;


%51\    PROCEDURE SUCESSIVA;

        BEGIN %1\
%        WRITELN(TTY,'   INICIO-[SUCESSIVA]');BREAK(TTY); \
         P := PILHA[PPILHA].DE ; Q := PILHA[PPILHA].PARA;
         JOGADOR := 'X';
         JOGA(Q,P);
         PPILHA := PPILHA - 1;
         JOGADOR := 'O' ; DECIDE;
         IF PSTACK # 0 THEN
         BEGIN %2\
```

```
        P := STACK[1].PARA ; PROCURA(P);
        SIMULA(QUAL);
        IF QUAL # NIL THEN JLEGAL := TRUE ELSE JLEGAL := FALSE
       END %2\
                    ELSE
       BEGIN %2\
        SALDO; ORDENA; ESCOLHEJOGADA
       END %2\
%      WRITELN(TTY,'   FIM-(SUCESSIVA)');BREAK(TTY) \
      END %1\;


%52\  PROCEDURE GRAVCONF;
      VAR L,K,N,J,I : INTEGER;

      BEGIN %1\
%      WRITELN(TTY,'   INICIO-(GRAVCONF)');BREAK(TTY); \
       IF (PPILHA = 0 ) OR (PPILHA = 1 ) THEN MENSAGEM ELSE
       BEGIN %2\
        P := PILHA[PPILHA].DE ; Q := PILHA[PPILHA].PARA ;
        JOGADOR := 'O';
        JOGA(Q,P);
        PPILHA := PPILHA - 1;
        CONFIGURA;
        N := 0 ; I := 0 ;
        REPEAT
         BEGIN %3\
          I := I + 1 ; J := I ; K := CONFIGURACAO[I];
          REPEAT
           BEGIN %4\
            J := J + 1 ; N := N + 1 ; L := CONFIGURACAO[J];
            PALABITE(K,N,L);
            CODIFICA
           END %4\
          UNTIL (J =8)
         END %3\
        UNTIL (I = 7);
        SUCESSIVA;
        IF JLEGAL = FALSE THEN GRAVCONF;
        PPILHA := PPILHA + 1; JOGADOR := 'X';
        P := PILHA[PPILHA].DE ; Q := PILHA[PPILHA].PARA;
        JOGA(P,Q);
        PPILHA := PPILHA + 1 ; JOGADOR := 'O';
        P := PILHA[PPILHA].DE ; Q := PILHA[PPILHA].PARA;
        JOGA(P,Q);
       END %2\
%      WRITELN(TTY,'   FIM-(GRAVCONF)');BREAK(TTY) \
      END %1\;


%53\  PROCEDURE VERCONF;
      VAR  K,L,N,I,J : INTEGER;

      BEGIN %1\
%      WRITELN(TTY,'   INICIO-(VERCONF)');BREAK(TTY); \
       JOGADOR := 'X' ; JOGA(L1,L2);
       N :=0; I:= 0; EXISTE := TRUE;
       CONFIGURA;
       REPEAT
        BEGIN %2\
```

```
          I := I + 1;  J := I;
          K := CONFIGURACAO[I];
          REPEAT
           BEGIN %3\
            J := J + 1;
            L := CONFIGURACAO[J];
            N := N + 1;
            PALABITE(K,N,L);
            DECODIFICA;
            IF MASCARA[BITE] = 0 THEN EXISTE := FALSE
           END %3\
          UNTIL (J =8) OR ( EXISTE = FALSE)
         END %2\
        UNTIL (I = 7) OR ( EXISTE = FALSE);
        JOGA(L2,L1)
%       WRITELN(TTY,'   FIM-(VERCONF)');BREAK(TTY); \
       END %1\;

%54\   PROCEDURE EMPILHA;

       BEGIN %1\
        PPILHA := PPILHA + 1 ;
        PILHA[PPILHA].DE := L1;
        PILHA[PPILHA].PARA := L2
       END %1\;

%*************************************************************************
*                                                                        *
*     P R O G R A M A  P R I N C I P A L              **                  *
*                                                                        *
*************************************************************************
\

       BEGIN %1\
        LOAD;
        QI := 0; PPILHA := 0;
        FIRSTPOINTER;FIRSTQUADRO;
        CONTROLE_TELA;
        BEGIN
         WRITE(TTY,' ':10,'VOCE ESTA NO VIDEO (VT05/GT40)) ?? ');
         BREAK(TTY);
         SIM_NAO(DEVICE);
        END;
        IF DEVICE = 'S' THEN VISOR := TRUE
        ELSE VISOR := FALSE;
        %TICTAC;
        BEGIN %2\
         WRITE(TTY,'VOCE QUER COMECAR A JOGAR ?   ');BREAK(TTY);
         SIM_NAO(OPCAO);
        END %2\;
        APAGA;
        IF OPCAO = 'N' THEN
        BEGIN
          VEZ := 1;
          WRITELN(TTY,'     EU COMECO');BREAK(TTY);
        END
        ELSE
        BEGIN
```

```
 VEZ := -1;
 WRITELN(TTY,'     VOCE COMECA');BREAK(TTY);
END;
IF VISOR = TRUE THEN
BEGIN
 MOSTRA_QUADRO_VT;
 MOSTRA_PECA_VT;
END
ELSE
BEGIN
 MOSTRA;
END;
NUMLANC  := 0;
REC := 0;
HIST := 0;
 REPEAT
 BEGIN %2\
   CASE VEZ OF
  1 : BEGIN %3\
       SUCESSO := FALSE; JOGADOR := 'X' ; DECIDE;
       IF PSTACK # 0 THEN
       BEGIN %4\
        P := STACK[1].DE ; Q := STACK[1].PARA;
        SUCESSO := TRUE ; JOGADOR := 'X' ; JOGA(P,Q)
       END %4\
                    ELSE
       BEGIN %4\
        JOGADOR := 'O' ; FRACASSO := FALSE; DECIDE ;
        IF PSTACK # 0 THEN
        BEGIN %5\
         CONDENSA(T);
         IF T > 1 THEN FRACASSO := TRUE
                  ELSE
         BEGIN %6\
          P := STACK[1].PARA ; PROCURA(P);
          SIMULA(QUAL);
          IF QUAL # NIL THEN
          BEGIN %7\
           JOGA(QUAL,P);
           EMPILHA(QUAL,P)
          END %7\
                       ELSE
          BEGIN %7\
           WRITELN;BREAK;WRITELN;BREAK;
           WRITE(' ',10,'VOCE PODERA GANHAR O JOGO SE FOR INTELIGENTE !! [FINAL SIMULA]');BREAK;
           WRITELN;BREAK;WRITELN;BREAK;
           JOGADOR := 'O' ; DECIDE;
           Q := STACK[1].PARA;
           PROCURA(Q);
           IF PLOCAL  # 0 THEN
           BEGIN %8\
            P := LOCAL[PLOCAL];
            JOGADOR := 'X';
            JOGA(P,Q);
            EMPILHA(P,Q)
           END %8\
                ELSE
           BEGIN %8\
            FRACASSO := TRUE
           END %8\
```

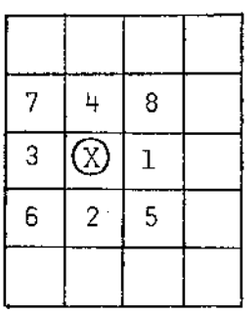
```
        END %7\
       END %6\
      END %5\
                  ELSE
      BEGIN %5\
       JOGADOR := 'X' ; ANALISA(JOGADOR);
       IF OK = TRUE THEN
       BEGIN %6\
        JOGADOR := 'X' ; JOGA(U,V);
        EMPILHA(U,V)
       END %6\
                  ELSE
       BEGIN %6\
        SALDO ; ORDENA;
        ESCOLHEJOGADA;
        IF (JLEGAL = TRUE) OR (U # NIL )  THEN
        BEGIN %7\
         IF JLEGAL = FALSE THEN
         BEGIN %8\
          NU := U ; NV := V ;
          GRAVCONF;
          U := NU ; V := NV ;
          WRITELN;BREAK;WRITELN;BREAK;
          WRITELN(' ';10,'VOCE PODERA GANHAR O JOGO SE FOR INTELIGENTE !! (ESCOLHEJOGADA)');BREAK;
          WRITELN;BREAK)WRITELN;BREAK
         END; %8\
         JOGADOR := 'X'; JOGA(U,V);
         EMPILHA(U,V)
        END %7\
                        ELSE
        BEGIN %7\
         GRAVCONF;
         FRACASSO := TRUE
        END %7\
       END %6\
      END %5\
     END; %4\
     IF FRACASSO = FALSE THEN
      BEGIN
       ONDE_JOGUEI; MOSTRA_JOGO;DA_UM_TEMPO;
      END;
    END; %3\
 "1 : BEGIN %3\
       LEJOGADA;MOSTRA_JOGO
      END; %3\
 OTHERS : P := P
   END ; %CASE\

 VEZ := VEZ*(-1) ;
 IF GRAVA = 'S' THEN GRAVO;
 NUMLANC := NUMLANC + 1
END %2\
UNTIL (SUCESSO = TRUE) OR (FRACASSO = TRUE) OR (NUMLANC = 50);
STORE;
APAGA;
IF SUCESSO = TRUE THEN
BEGIN %2\
 WRITELN(TTY,BEL);BREAK(TTY);WRITELN(TTY,BEL);BREAK(TTY);
 WRITELN(TTY,' ';10,'FIZEMOS ',NUMLANC:2,'  LANCES');
 BREAK(TTY);
```

```
 WRITELN(TTY);BREAK(TTY);WRITELN(TTY);BREAK(TTY);
 WRITELN(TTY,' ':10,'HE HE HE  * GANHEI *  HE HE HE ');BREAK(TTY);
WRITELN(TTY);BREAK(TTY);
WRITELN(TTY,' ':10,'VE SE PRATICA MAIS UM POUCO !!!! ');BREAK(TTY);WRITELN(TTY);BREAK(TTY);
WRITELN(TTY,' ':10,'NAO ME LEVE A MAL MAS VOCE ESTA BEM RUINZINHO');BREAK(TTY);
WRITELN(TTY);BREAK(TTY);
END %2\
                    ELSE
BEGIN %2\
 IF FRACASSO = TRUE THEN
 BEGIN %3\
  WRITELN(TTY);BREAK(TTY);WRITELN(TTY);BREAK(TTY);
 WRITELN(TTY,' ':10,'FIZEMOS ',NUMLANC,'  LANCES');
 BREAK(TTY);
 WRITELN(TTY);BREAK(TTY);WRITELN(TTY);BREAK(TTY);
 WRITELN(TTY,' ':10,'MEUS PARABENS VOCE CONSEGUIU GANHAR');BREAK(TTY);
 WRITELN(TTY);BREAK(TTY);
WRITELN(TTY,' ':10,'MAS EU TEREI A MINHA REVANCHE NAO E ?');BREAK(TTY);
 WRITELN(TTY);BREAK(TTY);
END %3\
              ELSE
 BEGIN %3\
  WRITELN(TTY);BREAK(TTY);WRITELN(TTY);BREAK(TTY);
  WRITELN(TTY,' ':10,' EMPATAMOS E UM BOM SINAL PARA VOCE !');BREAK(TTY);
  WRITELN(TTY);BREAK(TTY);
  WRITELN(TTY,' ':10,'FIZEMOS ',NUMLANC:2,' NUMERO DE LANCES');BREAK(TTY);
  WRITELN(TTY);BREAK(TTY);
  WRITELN(TTY,' ':10,'REALMENTE FOI UM JOGO DURO');BREAK(TTY);
 WRITELN(TTY);BREAK(TTY);
  WRITELN(TTY,' ':10,'ESPERO VE-LO NOVAMENTE AI'' ENTAO VEREMOS');BREAK(TTY);
  WRITELN(TTY);BREAK(TTY);
  WRITELN(TTY,' ':10,'NAO ME LEVE A MAL MAS EU ACHO QUE FOI SORTE SUA');BREAK(TTY);
  WRITELN(TTY);BREAK(TTY);
 END %3\;
END %2\;
WRITELN(TTY);BREAK(TTY);
WRITELN(TTY,' ':10,'ESPERO PODER JOGAR NOVAMENTE COM VOCE');BREAK(TTY);
WRITELN(TTY);BREAK(TTY);
WRITELN(TTY,' ':10,'ATE'' A PRO''XIMA TCHAU !!!!!!!');BREAK(TTY);
END %1\ .
```

A N E X O B

CONFIGURAÇÕES TERMINAIS PARA O JOGO "TAC-TICKLE"

Existem para o jogo "Tac-Tic1ke", 32 padrões de configu ções terminais identificando, cada um, a existência de pelo menos uma JT ou JTB. Os padrões são identificados pelo procedimento DECIDE (# 12) que se utiliza dos procedimentos COMPLETA (# 8) e VERIFICA (# 11). Note-se que, a estrutura de dados implementada pelo procedimento FIRSTPOINTER (# 3) (ilustrada na figura b1), é utilizada na identificação das configurações terminais que se seguem:



Onde:

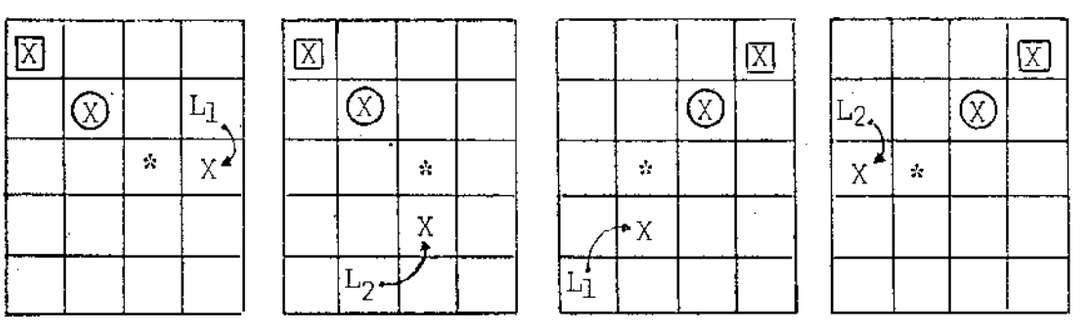
(X) - célula apontada pela variável X

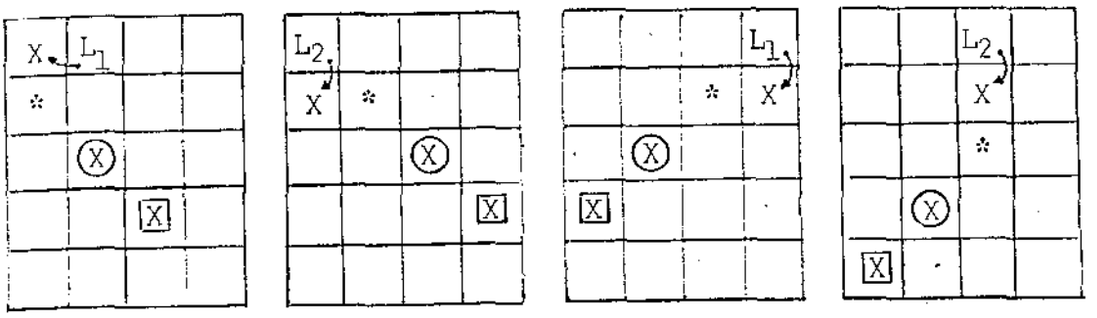
1 - célula apontada pela variável X^. Próximo [1]

8 - célula apontada pela variável X^. Próximo [8]

FIGURA b1 - ESTRUTURA DE DADOS

a). O procedimento COMPLETA, que se utiliza do procedimento VER (# 6), identifica as seguintes configurações terminais:





I = 7 I = 7 I = 8 I = 8

Onde:

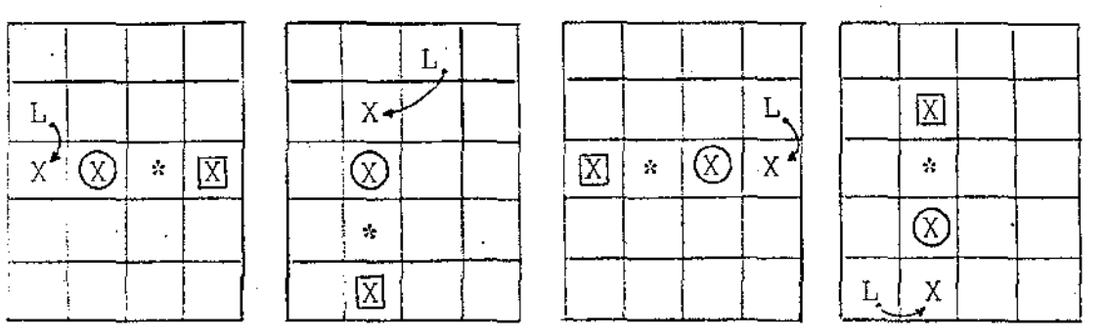
I - variável I utilizada no procedimento DECIDE e COMPLETA.

Ⓧ - posição Tab [Linha, Coluna] sendo analisada.

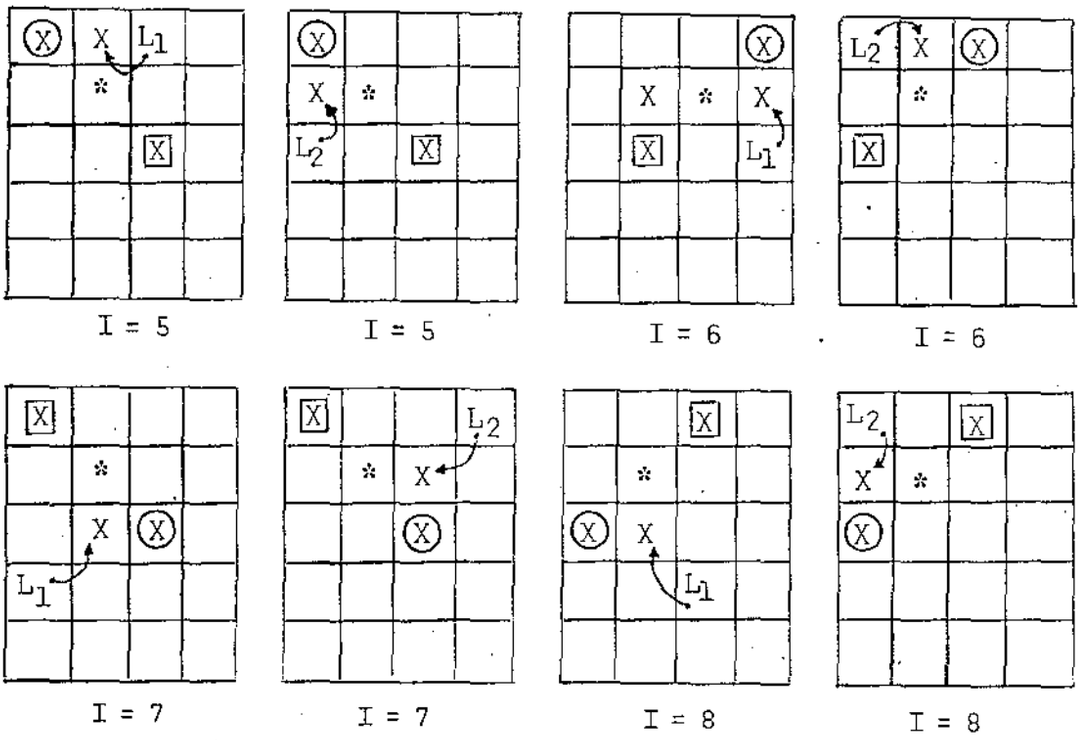
X - célula apontada pela variável L.

* - célula apontada pela variável P.

b). O procedimento DECIDE, através da chamada dos procedimentos TRANSA (# 7) e VER (# 6), identifica 12 configurações terminais:



I = 1 I = 2 I = 3 I = 4



Onde:

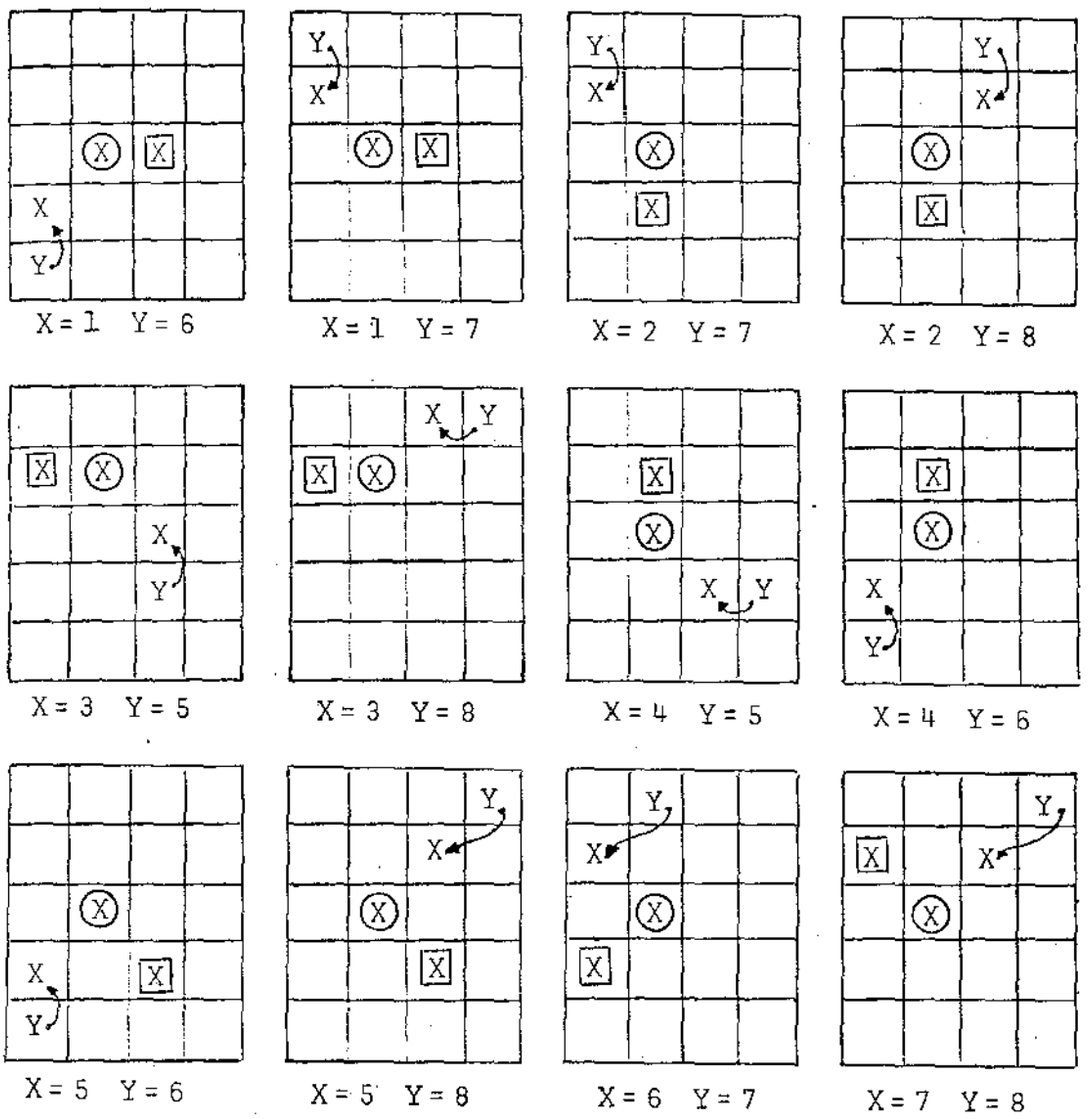
I - variável I utilizada pelo procedimento DECIDE.

Ⓧ - posição Tab [Linha, Coluna] sendo analisada.

[X] - célula apontada pela variável Q.

* - célula apontada pela variável P.

c). Finalmente o procedimento VERIFICA identifica, com auxílio dos procedimentos TRANSFERE (# 9) e TRANSPEÇA (# 10), as seguintes configurações terminais:



onde:

X - variável X utilizada pelo procedimento VERIFICA.

Y - variável Y utilizada pelo procedimento VERIFICA.

[X] - célula apontada pela variável X.

(X) - posição Tab [Linha, Coluna] sendo analisada;
também célula apontada pela variável P.

A N E X O C

PARTIDAS - EXEMPLO

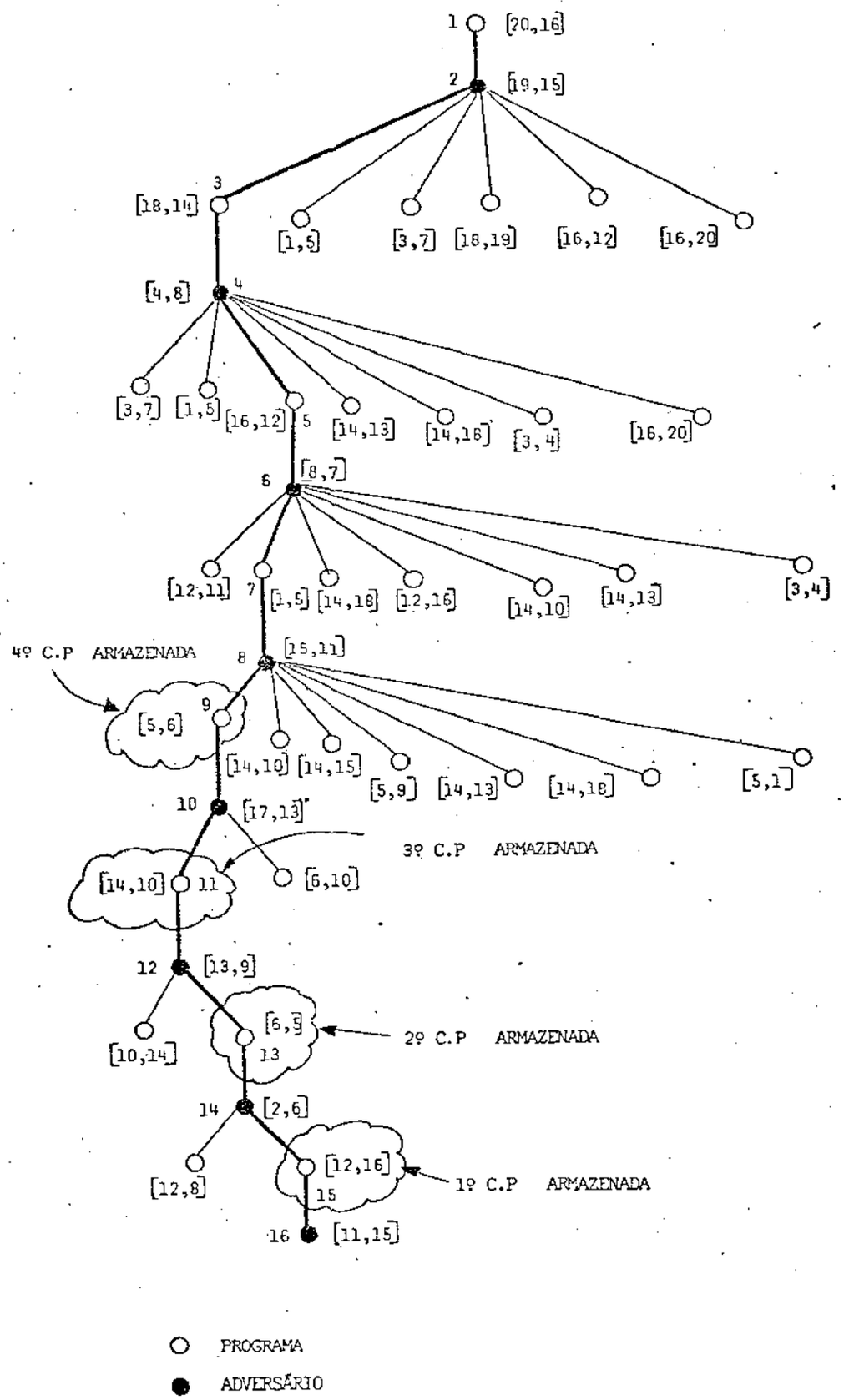
Com respeito às árvores de derivações, note-se que:

a). Junto à cada um dos nós da árvore encontra-se o número da configuração associada.

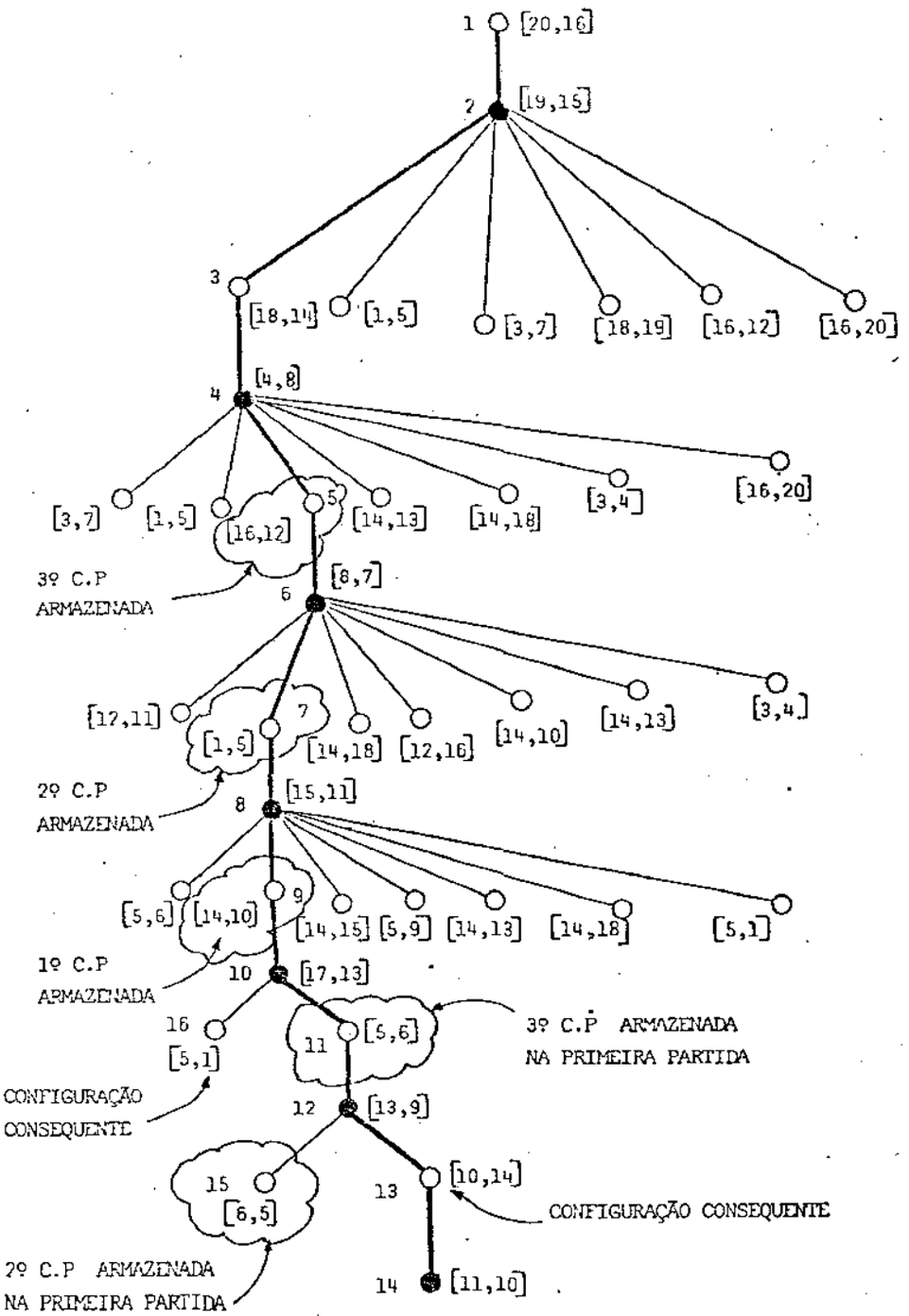
b). A ordem de apresentação das possíveis jogadas [P] representa o resultado da aplicação da função de avaliação (melhor jogada à esquerda).

c). Os ramos correspondentes as jogadas efetivamente realizadas encontram-se em destaque.

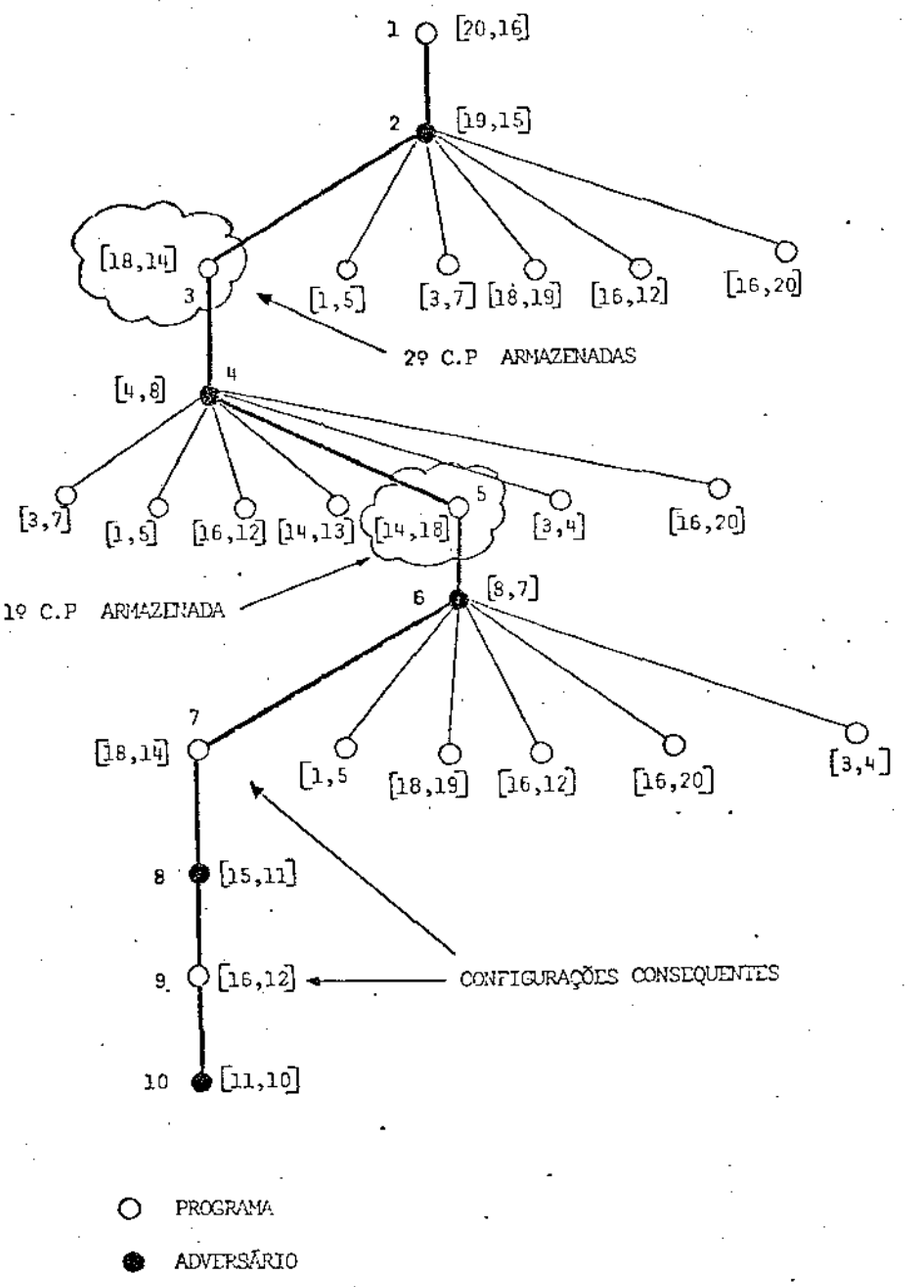
ÁRVORES DE DERIVAÇÕES DAS PARTIDAS 1, 2 e 3



ÁRVORE DE DERIVAÇÕES DA PRIMEIRA PARTIDA



ÁRVORE DE DERIVAÇÕES DA SEGUNDA PARTIDA

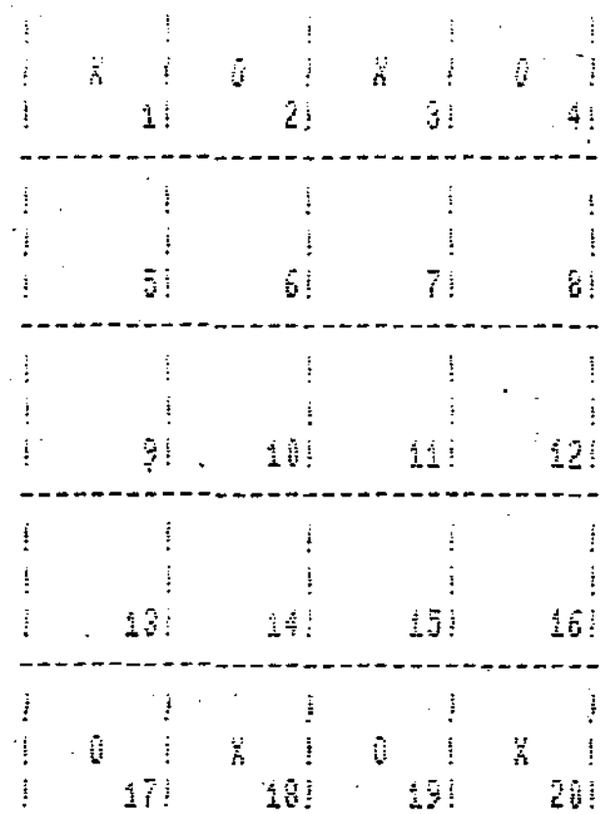


ÁRVORE DE DERIVAÇÕES DA TERCEIRA PARTIDA

LISTAGEM (ENTRADAS/SAÍDAS) PRODUZIDA PELA SEGUNDA PARTIDA

```
.RUN TACTIC 23

                                                                    -C8-

*
      VOCE ESTA NO VIDEO (VT05/GT40)) ?? BATA 'S' OU 'N':N


          TTTTT    AAA     CCCC   TTTTT    III     CCCC
            T     A   A   C         T       I     C
            T     A   A   C         T       I     C
            T     A   A   C         T       I     C
            T     AAAAA   C         T       I     C
            T     A   A   C         T       I     C
            T     A   A    CCCC     T      III     CCCC

                  @ I ===> 0

                     B O A  S O R T E !!!!!!!

DESEJA EXPLICACOES ?    BATA 'S' OU 'N':S

          ***************************************
          *                                     *
          * ---  A E G R A S  I D  J O G O  ---  *
          *                                     *
          ***************************************
     0.  JOGADORES
         ---------
         ==)  EL JOGO COM O SIMBOLO - X -
         ==) VOCE JOGA COM O SIMBOLO - 0 -
     1.  MOVIMENTOS PERMITIDOS
         ---------------------
         ==)  MOVER SUA PECA DE UMA POSICAO NA HORIZONTAL  OU
         ==)  MOVER SUA PECA DE UMA POSICAO NA VERTICAL
         OBSERVACAO :  NAO E PERMITIDO PULAR NA DIAGONAL
                       JOGA-SE ALTERNADAMENTE
                       NAO E PERMITIDO PULAR PECAS
     2.  VENCE AQUELE QUE :
         ------------------
         ==)  PRIMEIRO AGRUPAR 3(TRES) DE SUAS PECAS NA :
         ==) HORIZONTAL VERTICAL OU DIAGONAL
```

*** ESTA E A CONFIGURACAO INICIAL DO TABULEIRO ***



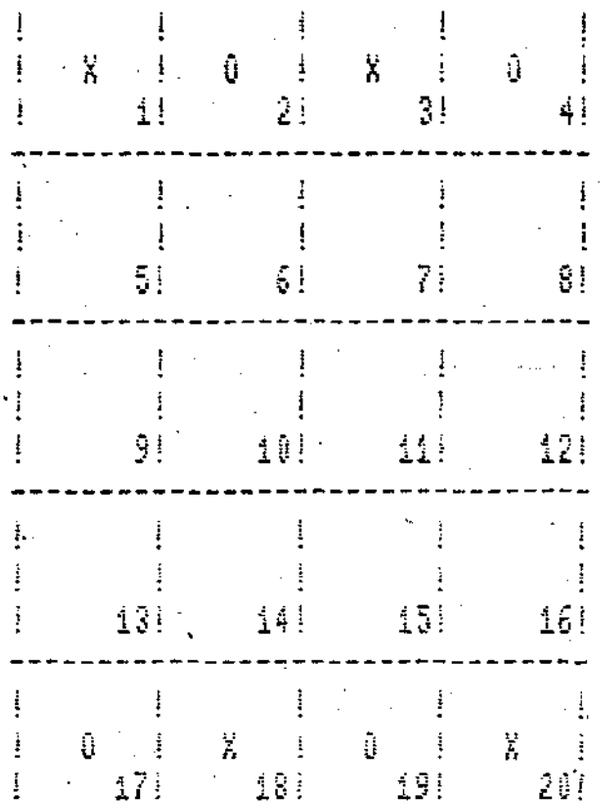
VOCE QUER VER AS REGRAS NOVAMENTE ? BATA 'S' OU 'N':N

DESEJA HISTORICO EM ARQUIVO OUTPUT ? BATA 'S' OU 'N':S

DESEJA COMECAR DE ALGUMA CONFIGURACAO INICIAL ? BATA 'S' OU 'N':N

VOCE QUER COMECAR A JOGAR ? BATA 'S' OU 'N':N

EU COMECO



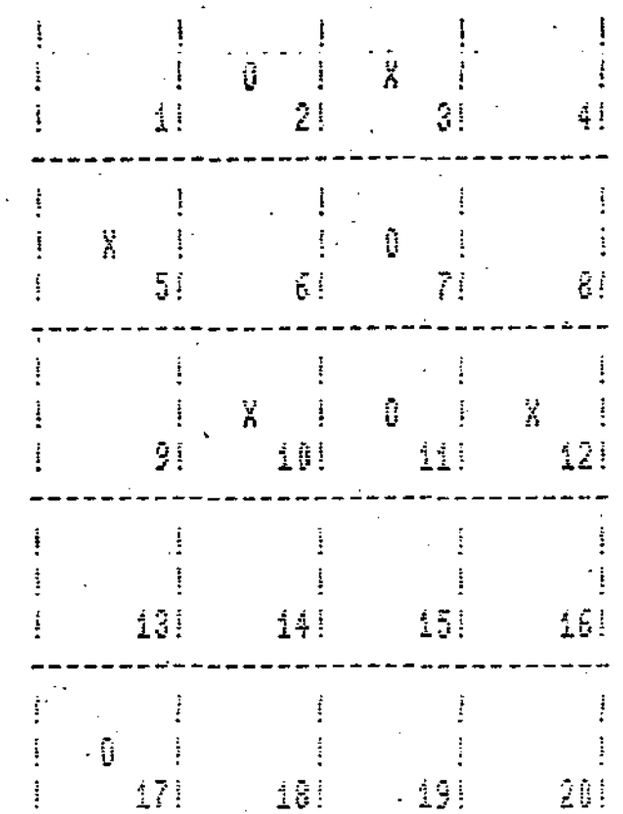
MINHA JOGADA
DE => 20
PARA => 16
QUER VER O TABULEIRO ?? BATA 'S' OU 'N':N

ENTRE COM A SUA JOGADA
DE => 19
PARA => 15
QUER VER O TABULEIRO ?? BATA 'S' OU 'N':N

MINHA JOGADA
DE => 18
PARA => 14
QUER VER O TABULEIRO ?? BATA 'S' OU 'N':N

```
              ENTRE COM A SUA JOGADA
DE   => 4
PARA => 9


         *** JOGADA INVALIDA ***
DE   => 4
PARA => 8
QUER VER O TABULEIRO ?? BATA 'S' OU 'N':N



     MINHA JOGADA
DE   => 16
PARA => 12
QUER VER O TABULEIRO ?? BATA 'S' OU 'N':B
?BATA 'S' OU 'N':N
BATA 'S' OU 'N':N


              ENTRE COM A SUA JOGADA
DE   => 8
PARA => 7
QUER VER O TABULEIRO ?? BATA 'S' OU 'N':N.



     MINHA JOGADA
DE   =>  1
PARA =>  5
QUER VER O TABULEIRO ?? BATA 'S' OU 'N':N



              ENTRE COM A SUA JOGADA
DE   => 15
PARA => 11
QUER VER O TABULEIRO ?? BATA 'S' OU 'N':N
```

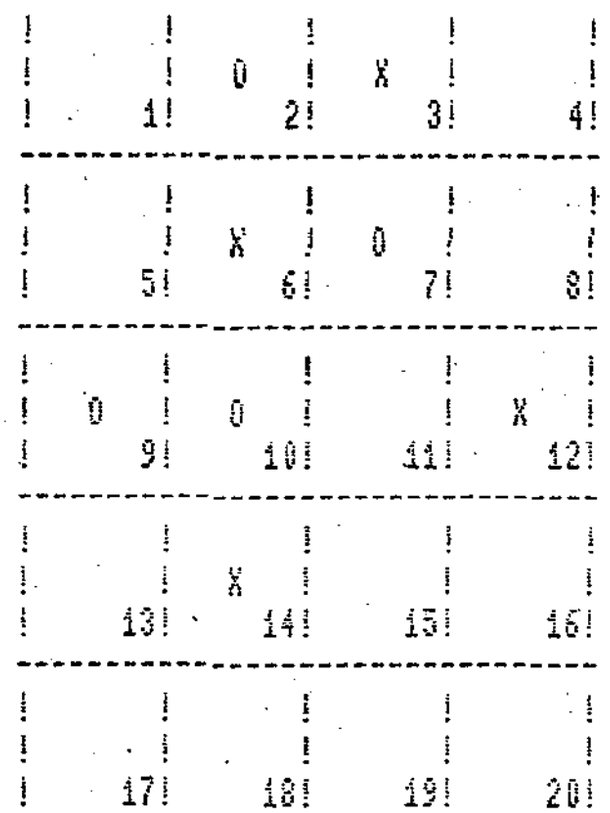
```
     MINHA JOGADA
DE   => 14
PARA => 10
QUER VER O TABULEIRO ?? BATA 'S' OU 'N':S
```




```
               ENTRE COM A SUA JOGADA
DE   => 17
PARA => 13
QUER VER O TABULEIRO ?? BATA 'S' OU 'N':N


     MINHA JOGADA
DE   =>  5
PARA =>  6
QUER VER O TABULEIRO ?? BATA 'S' OU 'N':S
```

```
!       !       !       !       !
!       !   0   !   X   !       !
!     1 !     2 !     3 !     4 !
-------------------------------------
!       !       !       !       !
!       !   X   !   0   !       !
!     5 !     6 !     7 !     8 !
-------------------------------------
!       !       !       !       !
!       !   X   !   0   !   X   !
!     9 !    10 !    11 !    12 !
-------------------------------------
!       !       !       !       !
!   0   !       !       !       !
!    13 !    14 !    15 !    16 !
-------------------------------------
!       !       !       !       !
!       !       !       !       !
!    17 !    18 !    19 !    20 !
```

```
            ENTRE COM A SUA JOGADA
DE   => 13
PARA => 9
QUER VER O TABULEIRO ?? BATA 'S' OU 'N':N


    MINHA JOGADA
DE   => 10
PARA => 14
QUER VER O TABULEIRO ?? BATA 'S' OU 'N':N


            ENTRE COM A SUA JOGADA
DE   => 11
PARA => 10
QUER VER O TABULEIRO ?? BATA 'S' OU 'N':S
```

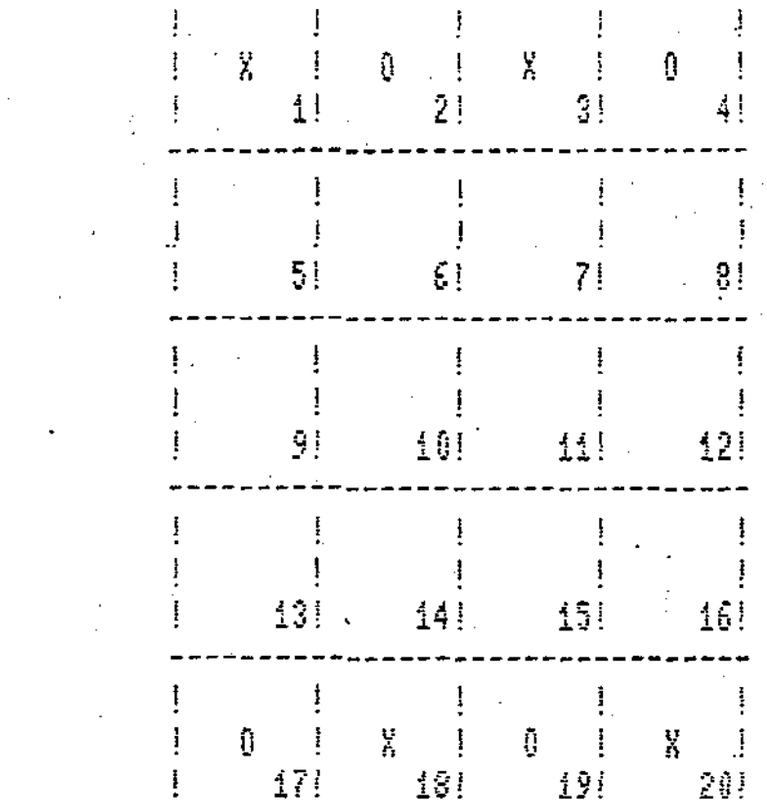



```
FIZEMOS          15  LANCES

MEUS PARABENS VOCE CONSEGUIU GANHAR

MAS EU TEREI A MINHA REVANCHE NAO E ?

ESPERO PODER JOGAR NOVAMENTE COM VOCE

ATE' A PRO'XIMA TCHAU !!!!!!!

EXIT
```

LISTAGEM (ENTRADAS/SAÍDAS) PRODUZIDA PELA QUARTA PARTIDA

```
.RUN TACTIC 23

*
        VOCE ESTA NO VIDEO (VT05/GT40)) ?? BATA 'S' OU 'N':N


              TTTTT    AAA      CCCC   TTTTT    III     CCCC
                T     A   A    C         T       I     C
                T     A   A    C         T       I     C
                T     A   A    C         T       I     C
                T     AAAAA    C         T       I     C
                T     A   A    C         T       I     C
                T     A   A     CCCC     T      III     CCCC

                      @ I ==> 0

                          B O A   S O R T E !!!!!!!

DESEJA EXPLICACOES ?    BATA 'S' OU 'N':N


DESEJA HISTORICO EM ARQUIVO OUTPUT ?     BATA 'S' OU 'N':S

DESEJA COMECAR DE ALGUMA CONFIGURACAO INICIAL ?     BATA 'S' OU 'N':N


VOCE QUER COMECAR A JOGAR ?   BATA 'S' OU 'N':N
```

EU COMECO



MINHA JOGADA
DE   => 20
PARA => 16
QUER VER O TABULEIRO ?? BATA 'S' OU 'N':N

ENTRE COM A SUA JOGADA
DE   => 19
PARA => 15
QUER VER O TABULEIRO ?? BATA 'S' OU 'N':N

MINHA JOGADA
DE   => 1
PARA => 5
QUER VER O TABULEIRO ?? BATA 'S' OU 'N':N

```
                ENTRE COM A SUA JOGADA
DE   => 4
PARA => 8
QUER VER O TABULEIRO ?? BATA 'S' OU 'N':N


     MINHA JOGADA
DE   =>  5
PARA =>  6
QUER VER O TABULEIRO ?? BATA 'S' OU 'N':N

[EXCEEDING QUOTA ON DSKB]


                ENTRE COM A SUA JOGADA
DE   => 17
PARA => 13
QUER VER O TABULEIRO ?? BATA 'S' OU 'N':N


[EXCEEDING QUOTA ON DSKB]


     MINHA JOGADA
DE   => 18
PARA => 14
QUER VER O TABULEIRO ?? BATA 'S' OU 'N':N


                ENTRE COM A SUA JOGADA
DE   => 6
PARA => 7


          *** JOGADA INVALIDA ***
DE   => 8
PARA => 7
QUER VER O TABULEIRO ?? BATA 'S' OU 'N':N
```

```
     MINHA JOGADA
DE   => 14
PARA => 10
QUER VER O TABULEIRO ?? BATA 'S' OU 'N':1
BATA 'S' OU 'N':N


               ENTRE COM A SUA JOGADA
DE   => 13
PARA => 9
QUER VER O TABULEIRO ?? BATA 'S' OU 'N':S
```

	0	X	
1	2	3	4
	X	0	
5	6	7	8
0	X		
9	10	11	12
		0	X
13	14	15	16
17	18	19	20

```
     MINHA JOGADA
DE   => 10
PARA => 11
QUER VER O TABULEIRO ?? BATA 'S' OU 'N':N



        FIZEMOS 11  LANCES


        HE HE HE  * GANHEI *  HE HE HE

        VE SE PRATICA MAIS UM POUCO !!!!

        NAO ME LEVE A MAL MAS VOCE ESTA BEM RUINZINHO


        ESPERO PODER JOGAR NOVAMENTE COM VOCE

        ATE' A PRO'XIMA TCHAU !!!!!!!
```

SEQUÊNCIA DE JOGADAS DAS PARTIDAS 5 E 6

SEQUÊNCIA DE JOGADAS DA QUINTA PARTIDA (Qi = 0)

○ - ADVERSÁRIO X - COMPUTADOR

	JOGADOR	DE	PARA
1	○	2	6
2	X	1	5
3	○	19	15
4	X	20	16
5	○	6	7
6	X	18	14
7	○	15	11
8	X	14	10
9	○	11	15

	JOGADOR	DE	PARA
10	X	5	6
11	○	17	18
12	X	10	11

SEQUÊNCIA DE JOGADAS DA SEXTA PARTIDA (Qi = 4)

○ - ADVERSÁRIO X - PROGRAMA

	JOGADOR	DE	PARA
1	○	2	6
2	X	20	16
3	○	19	15
4	X	3	7
5	○	4	8
6	X	16	12
7	○	6	2
8	X	12	11
9	○	2	6

	JOGADOR	DE	PARA
10	X	1	2
11	○	8	4
12	X	2	3

A N E X O D

NÚMERO DE JOGADAS SIMULADAS

O número total de jogadas simuladas para a escolha da melhor jogada [P] baseia-se no bloco 4.2 da figura 6 (algoritmo PAD), uma vez que, a geração de configurações terminais é imprevisível, não podendo, portanto, ser medida.

Sendo k o número total de possíveis jogadas a um dado instante, e desprezando-se as jogadas que levam à configurações já analisadas (simuladas apenas para dar-se continuidade ao processo de análise), deduzimos que:

1. Se Qi = 0:

MODULO-B	Simula k jogadas [P] para verificar existência de JD [P].
MODULO-C	Simula k jogadas [P] para a aplicação da função de avaliação.

Para cada uma das n jogadas consideradas, para o teste de validação, temos:

MODULO-E	Simula k jogadas [A] para verificar existência de JD [A];

São simuladas 2k + nk jogadas.

2. Se Qi = 1 :

MODULO-B	Simula k jogadas para verificar existência de JD [P].
MODULO-D	Simula k jogadas [P] para a aplicação da função de avaliação.

Para cada uma das n jogadas consideradas:

MODULO-E	Simula k jogadas [A] para verificar existência de JD [A].

Para cada uma das k jogadas [A] temos:

REPETE-SE O PROCESSO COM Qi = 0

São simuladas:

$$2k + n\,(k + k\,(2k + nk)) = 2k + nk + 2k^2n + n^2k^2 =$$

$$= 2k\,(1 + nk) + nk\,(1 + nk) \text{ jogadas.}$$

3. Se Qi = 2 deduz-se, de forma análoga (onde se lê "repete-se o processo com Qi = 0", leia-se "repete-se o processo com Qi = 1"), que são simuladas:

$$2k + n\,(k + k\,(2k + nk + 2k^2n + n^2k^2)) =$$

$$= 2k + nk + nk2k + k^2n^2 + 2kn^2k^2 + n^3k^3 =$$

$$= 2k\,(1 + nk + n^2k^2) + nk\,(1 + nk + n^2k^2) \text{ jogadas.}$$

De forma geral temos:

$$J\,(k,n,Qi) = 2k \underbrace{(1 + nk + n^2k^2 + \ldots n^{Qi}k^{Qi})}_{\text{PG de razão} = nk} + nk \underbrace{(1 + nk + n^2k^2 + \ldots n^{Qi}k^{Qi})}_{\text{PG de razão} = nk}$$

Onde: A1 = 1 e n = Qi + 1 (número de termos da PG),

Logo:

$$J\,(k,n,Qi) = \frac{2k\,(nk^{Qi+1} - 1)}{nk - 1} + \frac{nk\,(nk^{Qi+1} - 1)}{nk - 1} =$$

$$= \frac{(2k + nk)\,(nk^{Qi+1} - 1)}{nk - 1}$$

A fórmula traduz o pior caso, uma vez que, quando da existência de JD [P] no MODULO-B ou JD [A] MODULO-E, as restantes k jogadas [P] ou k jogadas [A] não são simuladas.