UNIVERSIDADE FEDERAL DE SÃO CARLOS
ENGENHARIA DE COMPUTAÇÃO - DC

INTELIGÊNCIA ARTIFICIAL

Trabalho 1

Busy Police

Professor Dr Murilo Naldi

Aluno: Juan Henrique dos Santos RA: 594946

SUMÁRIO

1. INTRODUÇÃO

Este projeto se baseia no game Busy Police, um jogo antigo da Atari no qual o personagem principal (policial) tem a missão de prender o antagonista (ladrão). Para cumprir tal missão, o policial possui diversos obstáculos como: pular objetos, subir escadas, subir em elevadores e fugir de eventos que poderiam tirar sua vida até que eventualmente o personagem morresse entre outros.

De forma bem mais simplificada e através do auxílio da inteligência artificial, o aplicativo irá retornar ao usuário os possíveis caminhos para que o policial chegue ao ladrão. Caso não exista um caminho o usuário também será informado a respeito disso. Através da modelagem dos fatos e regras, foi possível recriar o jogo na linguagem lógica Prolog.

2. DESENVOLVIMENTO

O jogo foi modelado de forma a tornar possível realizar buscas em profundidade numa estrutura de grafo. O mapa do aplicativo é baseado em 10 posições na horizontal e 5 posições na vertical. Utilizando o sistema cartesiano como forma de localização, cada elemento do grafo possui uma posição X e uma posição Y.

As ligações entre os elemento desta estrutura foram realizadas através da vizinhança. Para movimentos na horizontal, torna-se necessário que o par da ligação possua a mesma abcissa Y e que a ordenada X seja um valor inteiro e imediato ao seu valor, tanto para cima quanto para baixo. Para movimentos na vertical torna-se necessário que os elementos possuam a mesma ordenada X e que seus valores de abcissa Y sejam imediatas, assim como no caso do movimento horizontal com o valor de X. Para que as escadas fossem utilizadas para subir e descer no mapa, elas foram atribuídas a um par de posições, com valores em X constante e variando o Y conforme já explicado acima.

```
%MONTANDO AS FOLHAS DO GRAFO (HORIZONTAL)
caminho(posicao(1,1),posicao(2,1)).
caminho(posicao(2,1),posicao(3,1)).
caminho(posicao(3,1),posicao(4,1)).
caminho(posicao(4,1),posicao(5,1)).
caminho(posicao(5,1),posicao(6,1)).
caminho(posicao(6,1),posicao(7,1)).
caminho(posicao(7,1),posicao(8,1)).
caminho(posicao(8,1),posicao(9,1)).
caminho(posicao(9,1),posicao(10,1)).
caminho(posicao(1,2),posicao(2,2)).
caminho(posicao(2,2),posicao(3,2)).
caminho(posicao(3,2),posicao(4,2)).
caminho(posicao(4,2),posicao(5,2)).
caminho(posicao(5,2),posicao(6,2)).
caminho(posicao(6,2),posicao(7,2)).
caminho(posicao(7,2),posicao(8,2)).
caminho(posicao(8,2),posicao(9,2)).
caminho(posicao(9,2),posicao(10,2)).
caminho(posicao(1,3),posicao(2,3)).
caminho(posicao(2,3),posicao(3,3)).
caminho(posicao(3,3),posicao(4,3)).
caminho(posicao(4,3),posicao(5,3)).
caminho(posicao(5,3),posicao(6,3)).
caminho(posicao(6,3),posicao(7,3)).
caminho(posicao(7,3),posicao(8,3)).
caminho(posicao(8,3),posicao(9,3)).
```

(Figura 1: Exemplo de ligações de elementos do grafo na horizontal)

```
%MONTANDO AS FOLHAS DO GRAFO (VERTICAL)
escada(posicao(1,2),posicao(1,3)).
escada(posicao(4,1),posicao(4,2)).
escada(posicao(5,4),posicao(5,5)).
escada(posicao(10,3),posicao(10,4)).
```

(Figura 2: Exemplo de ligações de elementos do grafo na vertical)

Assim como escadas, outros objetos foram adicionados ao jogo. Para aumentar a dificuldade, foram inseridos carrinhos para que o policial pule e algemas para que prenda o ladrão. O carrinho impede a passagem do policial, forçando o movimento de pular, já a algema é um obstáculo que impede que o policial realize sua missão, caso não passe pela posição que contenha o item e o pegue.

```
70  carrinho(posicao(2,1)).
71  carrinho(posicao(2,5)).
72  carrinho(posicao(3,4)).
73  carrinho(posicao(6,1)).
74  carrinho(posicao(7,4)).
75  carrinho(posicao(8,2)).
76  carrinho(posicao(10,1)).
```

(Figura 3: Trecho de código que posiciona objeto carrinho ao mapa.)

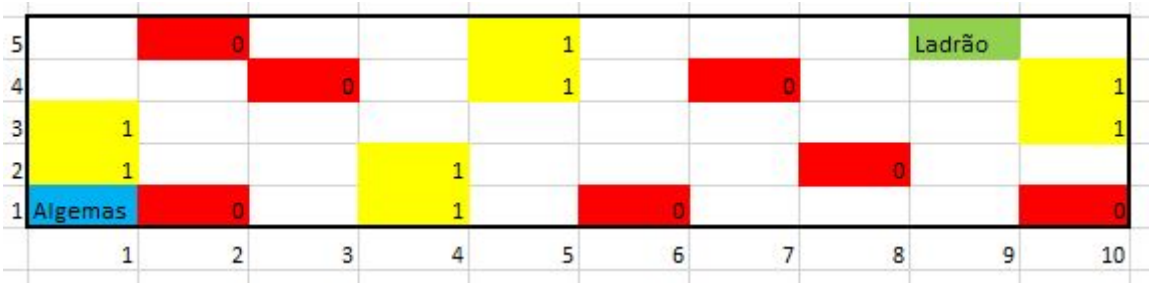
O jogo foi modelado de forma que o policial tome algumas ações de acordo com a situação. O personagem irá se mover em duas direções (esquerda, direita), podendo subir e descer escadas, pular carrinhos, obter algemas e trocar direção de movimento.

O estado do personagem foi modelado de forma que ela possua uma posição em X e uma posição em Y. Nesse estado também será informado se ele possui a algema para prender o ladrão e o sentido no qual ele está se movimentando.

Foram criadas algumas funções auxiliares para criar ligações bidirecionais entre os elementos. Ao mesmo tempo que cria esta ligação ela realiza verificações de existência.

```
79  %VERIFICO SE AS FOLHAS ESTÃO CONECTADAS (CAMINHO HORIZONTAL)
80  existe_caminho(X,Y,X2,Y2):- caminho(posicao(X,Y), posicao(X2,Y2)).
81  existe_caminho(X,Y,X2,Y2):- caminho(posicao(X2,Y2), posicao(X,Y)).
82  %VERIFICO SE AS FOLHAS ESTÃO CONECTADAS (CAMINHO VERTICAL)
83  existe_caminho(X,Y,X2,Y2):- escada(posicao(X,Y), posicao(X2,Y2)).
84  existe_caminho(X,Y,X2,Y2):- escada(posicao(X2,Y2), posicao(X,Y)).
85
```

(Figura 4: Trecho de código que tornam bidirecionais as relações do grafos.)



(Figura 5: Mapa criado para testar as movimentações do jogo).

3. RESULTADOS

A busca em profundidade permitiu ao jogo retornar diversos caminhos, não sendo necessariamente os mais otimizados. Os estados pelo qual o jogador passou são salvos numa lista e são exibidas ao final da execução, se houver sucesso. Os resultados se mostraram não ser otimizados, exibindo diversos combinações de caminhos.

```
%solucao por busca em profundidade (bp)
solucao_bp(X, Y, Sentido, Solucao) :- bp([], estado(posicao(X, Y), sem_algemas, Sentido), Solucao).

%Se o primeiro estado da lista é meta, retorna a meta
bp(Caminho, Estado, [Estado|Caminho]) :- meta(Estado).
%se falhar, coloca o no caminho e continua a busca
bp(Caminho, Estado, Solucao) :- acao(_, Estado, Prox_Estado), not(pertence(Prox_Estado,[Estado|Caminho])), bp([Estado|Caminho],Prox_Estado,Solucao).


%PROCURA OS CAMINHOS PARA PRENDER O LADRÃO
procura_ladrao(X,Y, Caminho):- solucao_bp(X, Y, direita, Solucao), formata_saida(Solucao, Caminho).

```

(Figura 6: Busca em profundidade no grafo de posições. Os estados são salvos numa pilha de estados que serão retornados caso haja sucesso na busca pelo caminho).

```
?- procura_ladrao(5,1,X).
 <- andar <- andar <- andar <- andar <- subir <- andar <- pular <- virar_esq <- andar <- virar_esq <- a
ndar <- virar_esq <- subir <- andar <- andar <- andar <- andar <- andar <- andar <- andar <- andar <- a
ndar <- subir <- andar <- virar_esq <- andar <- virar_esq <- andar <- virar_esq <- subir <- andar <- pu
lar <- virar_dir <- pegar_algema <- pular <- virar_esq <- andar <- descer <- virar_esq <- subir <- anda
r <- pular <- virar_esq <- pular
X = [estado(posicao(9, 5), com_algemas, direita), estado(posicao(8, 5), com_algemas, direita), estado(p
osicao(7, 5), com_algemas, direita), estado(posicao(6, 5), com_algemas, direita), estado(posicao(5, 5),
 com_algemas, direita), estado(posicao(5, 4), com_algemas, direita), estado(posicao(6, 4), com_algemas,
 esquerda), estado(posicao(8, 4), com_algemas, esquerda), estado(posicao(8, 4), com_algemas, direita),
estado(posicao(9, 4), com_algemas, esquerda), estado(posicao(9, 4), com_algemas, direita), estado(posic
ao(10, 4), com_algemas, esquerda), estado(posicao(10, 4), com_algemas, direita), estado(posicao(10, 3),
 com_algemas, direita), estado(posicao(9, 3), com_algemas, direita), estado(posicao(8, 3), com_algemas,
 direita), estado(posicao(7, 3), com_algemas, direita), estado(posicao(6, 3), com_algemas, direita), es
tado(posicao(5, 3), com_algemas, direita), estado(posicao(4, 3), com_algemas, direita), estado(posicao(
3, 3), com_algemas, direita), estado(posicao(2, 3), com_algemas, direita), estado(posicao(1, 3), com_al
gemas, direita), estado(posicao(1, 2), com_algemas, direita), estado(posicao(2, 2), com_algemas, esquer
da), estado(posicao(2, 2), com_algemas, direita), estado(posicao(3, 2), com_algemas, esquerda), estado(
posicao(3, 2), com_algemas, direita), estado(posicao(4, 2), com_algemas, esquerda), estado(posicao(4, 2
), com_algemas, direita), estado(posicao(4, 1), com_algemas, direita), estado(posicao(3, 1), com_algema
s, direita), estado(posicao(1, 1), com_algemas, direita), estado(posicao(1, 1), com_algemas, esquerda),
 estado(posicao(1, 1), sem_algemas, esquerda), estado(posicao(3, 1), sem_algemas, esquerda), estado(pos
icao(3, 1), sem_algemas, direita), estado(posicao(4, 1), sem_algemas, esquerda), estado(posicao(4, 2),
sem_algemas, esquerda), estado(posicao(4, 2), sem_algemas, direita), estado(posicao(4, 1), sem_algemas,
 direita), estado(posicao(5, 1), sem_algemas, esquerda), estado(posicao(7, 1), sem_algemas, esquerda),
estado(posicao(7, 1), sem_algemas, direita), estado(posicao(5, 1), sem_algemas, direita)]
```

(Figura 7: Exemplo de chamada da busca pelo ladrão. As ações realizadas foram escritas apenas no intuito de facilitar a compreensão do caminho).

Inicialmente o trabalho havia sido implementado de forma semelhante ao do código do "macaco", porém se não houvesse caminho o programa entrava em loop e era encerrado.

4. CONCLUSÃO

Através deste trabalho foi possível observar o poder e a capacidade que a programação lógica tem em relação a outros paradigmas de programação. Para recriar um jogo como esse em outra paradigma, provavelmente o esforço seria bem maior.

A busca em largura mostrou possuir um grande poder em procurar às cegas, retornando um resultado sempre que possível, entretanto, a busca não retorna o caminho mais otimizado.

5. BIBLIOGRAFIA

Slides disponibilizados no AVA. Matéria: Inteligência Artificial, ministrada pelo Professor Murilo Naldi.