Projeto Final: *Pokemón*

Prof. Matheus Nohra Haddad
SIN 323 - Inteligência Artificial

Igor Lúcio Rocha Alves - Matrícula: 3902
Gabriel Alves dos Santos Ferreira - Matrícula: 4851

29 de Novembro de 2018

1 Explicação do Problema

O trabalho consiste em criar um conjunto de regras na linguagem de programação *Prolog* que delineie um ambiente do jogo *Pokémon*, de forma que o protagonista possa encontrar a primeira insígnia de Kanto. Supomos a existência de um agente capaz de executar ações que modificam o estado atual de seu ambiente.

Assim, dados um estado inicial representando a configuração corrente do mundo do agente, um conjunto de ações que o agente é capaz de executar e uma descrição do estado meta que se deseja atingir, a solução do problema consiste numa sequência de ações que, quando executada pelo agente, transforma o estado inicial num estado meta. [1] Portanto, se trata, por definição, de um problema de busca. O objetivo é desenvolver um modelo lógico consistente para auxiliar nosso protagonista (Ash) a chegar em seu objetivo.

Figura 1: Ash mostrando insígnia conquistada. [2]

1.1 Ambiente

O ambiente do jogo será uma tabela 5X5 com 25 posições. Ou seja, o jogador estará limitado a este mundo; não sendo possível movimentação do mesmo além das posições citadas. Não será permitida a movimentação para posições inexistentes da tabela, isto é, coordenadas em que seu X e Y sejam, respectivamente, maiores que 4 como também menores que 0.

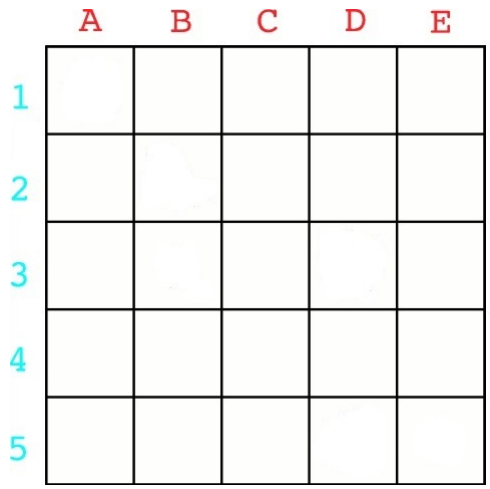

Figura 2: Simulação do ambiente - tabela 5x5 (25 posições)

1.2 Agente e restrições

Ash é um agente capaz se locomover no ambiente de exploração, ou seja, na tabela 5x5. Ele somente pode locomover para quadrantes adjacentes e não se movimenta na diagonal.

Entretanto, ele não pode se locomover para um quadrado em que haja algum *pokemon* sem ter pokebolas, além de que não se permite fazer ciclos, ou seja, voltar na posição anterior a que estava. Nosso protagonista será representado pelo ícone a seguir.

Figura 3: Representação de Ash no ambiente

Se o mesmo passar por um quadrante que contém pokebolas, pode-se capturar *pokemons* e consequentemente, movimentar-se por seus quadrantes.

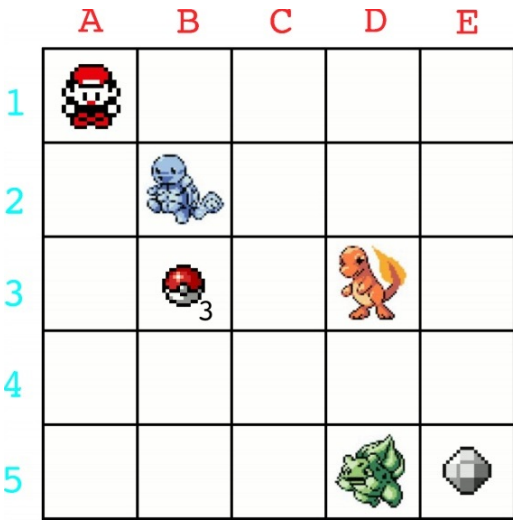

Figura 4: Ambiente inicializado com todas as 'variáveis'

2 Base de Conhecimento e Conjunto de Regras

Nossa base de conhecimento é composta pelas regras de sucessão de estados, isto é, movimentação. É contemplado também uma restrição do cenário (0 a 4), sendo assim, 5 linhas e 5 colunas. Fora utilizado quatro rotinas que trabalham com listas: verificar se dado elemento pertence a uma lista, inserir elemento no início, inverter lista e concatenar listas.

Houve a necessidade da criação de condições em relação aos *pokemons* e pokebola. O conjunto de regras a seguir nos possibilita, respectivamente, saber quantos *pokemons* foram capturados, condições para movimentação no cenário.

```
% contabilizar quantos pokemons peguei
find(_, [], 0).
find(El, [El|_], 1).
find(El, [_|T], X):-
        find(El, T, R),
        X = R.

%argumentos: caminho percorrido, lista pokemons, resultado
howMany(_,  [], 0).
howMany(List, [Pkm|T], Res):-
        find(Pkm, List, R2),    % se um pokemon pertence a esta lista, entao R2 = 1
        howMany(List, T, R3),
        Res is R2 + R3.
```

Figura 5: Regra para contabilização de pokemons capturados

Esta regra *howMany* é utilizada dentro da função principal (*main*), logo após fazermos a varredura. A ordem é bastante importante, pois através do resultado da função responsável pela busca, iremos contabilizar quantos *pokemons* foram capturados passando pelo seu parâmetro: caminho percorrido, lista dos *pokemons*, resultado.

```
conditions(Estado, Sucessor,Pkm, Pkb, Caminho):-
        s(Estado,Sucessor),  %Qualifica se o sucessor e valido, com os limites
        on(Pkb, [Estado|Caminho]),
        not(on(Sucessor,[Estado|Caminho])),
        %Para nao gerar ciclos
         on(Sucessor, Pkm).   %Estou pisando em um  pokemon


conditions(Estado, Sucessor, Pkm,_, Caminho):-
        s(Estado,Sucessor), %Qualifica se o sucessor e valido.
        not(on(Sucessor,[Estado|Caminho])), %Para nao gerar ciclos
        not(on(Sucessor,Pkm)). %Se nao estou pisando em pokemon, entao de boa
```

Figura 6: Condições para qualificação de estado sucessor

Em seu primeiro caso, a mesma é invocada dentro de um *bagof* da função *estende*, que, é um método que faz a extensão do caminho até os nós filhos do estado passado. Em outras palavras, a mesma irá retornar uma lista resultante que satisfaz as condições impostas na linha de condições; retorna uma lista de sucessores válidos.

Se o caso anterior não atender, fazemos a mesma lógica porém com o cuidado de não gerar ciclos em nosso caminho e verificar se não estamos pisando em um *pokemon* sem a pokebola. Pois do contrário, isso quebraria uma das definições do problema.

Logo após, chame a função principal passando os seguintes parâmetros: Posição Inicial, Lista De *Pokemons*, Pokebola, Insignia e Resultado. Um exemplo funcional é demonstrado na seção aseguir.

3 Como utilizar

Para executar o programa, é necessário um interpretador e compilador dos blocos de códigos, adaptados para *Prolog*, onde existem uma gama de possibilidades para tal função.

Para que o *Prolog*, possa mostrar em seu *prompt* a lista de caminhos completo, é necessário de utilizar a extensão da função write/1 [3], que é definida de forma própria pelo *Prolog*, sendo a mesma ativada após o aperto da tecla W, quando uma lista, com tamanho maior de 8 tenta ser exibida. Para auxiliar o usuário, durante a tarefa, foi criada a função err/1 Figura 7, que exibe uma lista longa o suficiente, para que o usuário em seguida possa apertar W e habilitar a função de escrita completa no *Prompt*.

Após chamar a função *err* e apertar W, está habilitado o *prompt* para exibição. Após isso, deve-se chamar a função main/5 Figura 9, instanciando os valores necessários. Os parâmetros seguem a seguinte ordem: Coordenada indicando estado inicial, lista de coordenadas indicando onde os *pokemons* se encontram no ambiente proposto, lista de coordenadas indicando locais com pokebolas, coordenada onde se encontra a insígnia.

```
?- err(X).
X = [[4, 4], [3, 4], [2, 4], [1, 4], [0, 4], [0, 3], [0, 2], [0|...], [...|...]] [write]
X = [[4, 4], [3, 4], [2, 4], [1, 4], [0, 4], [0, 3], [0, 2], [0, 1], [0, 0]] .

?- main([0,0],[[0,1],[2,1],[3,0]],[2,0],[4,4],S).
Pokemons, nos quadrantes: [[0,1],[2,1],[3,0]]
Pokebola no quadrante: [2,0]
Pokemons capturados: 0
Insignia no quadrante: [4,4]
Caminho do Ash:
S = [[0, 0], [1, 0], [1, 1], [1, 2], [1, 3], [1, 4], [2, 4], [3, 4], [4, 4]].
```

Figura 7: Saída para instância com 3 pokemons

4 Habilitando visualização gráfica do caminho

Afim de demonstrar de forma mais dinâmica e atrativa, fora desenvolvida uma ferramenta capaz de interpretar o caminho percorrido pelo agente na resolução do programa proposto. A ferramenta, a partir de um *script* em *Python*, juntamente com uma aplicação *web*, interpreta a saída, gerando uma animação (Figura 9) que pode ser exibida via navegador. Tal aplicação, está disponível em [4].

5 Conclusões

O problema proposto foi solucionado utilizando a busca em largura, onde o número de *pokemons* Figura 7, a posição inicial, a lista de pokebolas e a insígnia, são setadas e instanciadas pelo usuário,

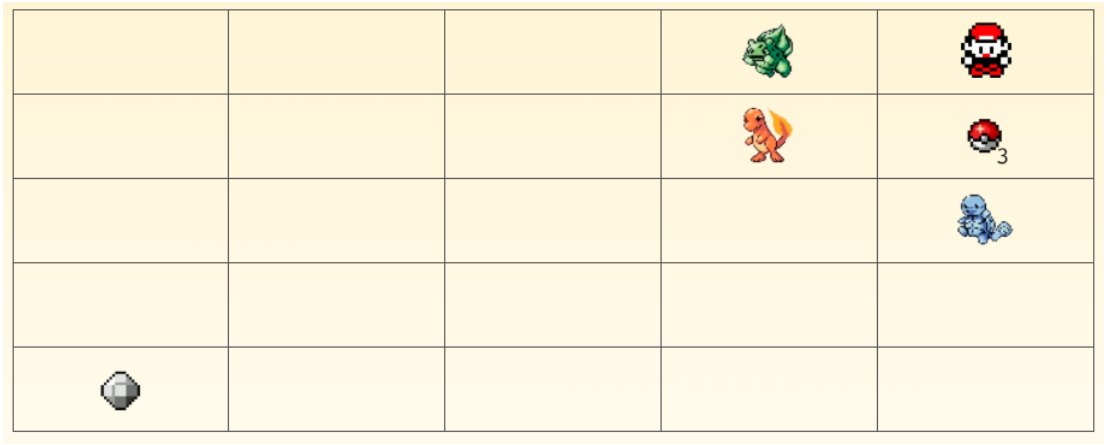

Figura 8: Exemplificação de uma instância do dado problema graficamente

sendo a insígnia o ponto final. Como saída, o algoritmo mostra em formato de listas: a lista de *pokemons*, coordenada da pokebola, número de *pokemons* capturados, coordenada da insígnia, ou objetivo e o caminho escolhido por *Ash*, como demonstra a Figura 9.

A busca por largura mostrou-se muito eficiente, mesmo com desafios propostos, podendo futuramente ser implementada uma melhoria, no sentido de indicar uma busca pelo menor caminho, como com a utilização do algoritmo de *Dijkstra* [5].

```
?- main([0,0],[[0,1],[2,1],[3,0],[0,2]],[2,0],[4,4],S).
Pokemons, nos quadrantes: [[0,1],[2,1],[3,0],[0,2]]
Pokebola no quadrante: [2,0]
Pokemons capturados: 0
Insignia no quadrante: [4,4]
Caminho do Ash:
S = [[0, 0], [1, 0], [1, 1], [1, 2], [1, 3], [1, 4], [2, 4], [3, 4], [4, 4]].
```

Figura 9: Saída para instância com 4 pokemons

Referências

[1] S. do Lago Pereira, "Busca no espaço de estados." https://www.ime.usp.br/~slago/IA-Busca.pdf, 2005.

[2] "*Pokémon - The Battle of the Badge.*" https://www.imdb.com/title/tt0761144/, 1998. TV Tokyo, The WB Television Network, Kids' WB.

[3] "*write/1 Function.*" http://www.swi-prolog.org/FAQ/AllOutput.html, 2004. Prolog, helping with output.

[4] "*GitHub - Visualização do problema pokemon.pl.*" https://github.com/iguit0/Inteligencia-Artificial, 2018. visualização e interpretação dos dados do trabalho pokemon, para disciplina IA - UFv.

[5] L. Wijngaards-de Meij, M. Stroebe, H. Schut, and W. Stroebe, "& dijkstra, i. patterns of attachment and parents' adjustment to the death of a child," *Personality and Social Psychology Bulletin*, vol. 33, no. 4, pp. 537–548, 2007.