

Resolvendo tabuleiros de Sudoku com um agente inteligente desenvolvido em Python

Fabício da Silva Selotto; Geísa Laiane Cardim.

Introdução:

Sudoku é um antigo quebra-cabeça de origem japonesa. A ideia do jogo é bastante simples: preencher uma grade de 9 quadrados na horizontal por 9 quadrados na vertical de modo que, em cada linha, em cada coluna e em cada quadrado maior, haja números de 1 a 9 sem repetir nenhum deles.

O algoritmo funciona por busca bruta, buscando resultados possíveis e abandonando os que não foram satisfatórios, caracterizando assim um algoritmo backtracking, que é utilizado também em problemas como o Caixeiro Viajante, N-Rainhas, Passeio do Cavalo, problemas de labirintos, entre outros.

Objetivo:

O objetivo foi resolver jogos de Sudoku, retornando o tabuleiro resolvido.

Descrição do Agente:

Foi criada uma função que gera aleatoriamente um tabuleiro de Sudoku incompleto, checa se é um tabuleiro possível, e o retorna em formato de matriz de 9x9.

O algoritmo foi feito com base em backtracking, busca por força bruta, como uma árvore de possibilidades, buscando um caminho e caso não seja válido, exclui aquela possibilidade e parte para o próximo galho.

Como ambiente ele usa tabuleiros de sudoku, em formato de matriz de 9x9, sendo 0 as posições vazias. Foi usada a função que gera o gráfico aleatoriamente e também foram definidos alguns tabuleiros manualmente.

Como sensor uma função que localiza as posições vazias do tabuleiro.

O agente possui 2 funções para achar a solução, sendo estas os atuadores, que inserem no tabuleiro os números faltantes, conferindo primeiramente pelo sensor quais são vazias, tenta inserir números de 1 à 9, fazendo uma validação, se este número já existe na linha, coluna ou bloco, é setado o valor 0, fazendo uma nova tentativa com o próximo número de 1 à 9.

As verificações são percorridas por várias vezes, até preencher o tabuleiro por completo, retornando a solução, novamente em formato de matriz, sendo esta totalmente preenchida.

Como adendo, foi feita uma função que não interfere no funcionamento do agente, que tem como utilidade representar de melhor maneira a matriz, colocando-a em uma espécie de 'tabuleiro' de mesmo formato que no jogo.

Descrição do ambiente de desenvolvimento:

Para desenvolvimento foi utilizado a linguagem de programação Python versão 3.8.3. Usamos a biblioteca *random* para a função de geração aleatória do tabuleiro e *numpy* para melhor visualização da matriz. Utilizamos Visual Studio Code IDE e Jupyter Notebook.

Resultados:

O algoritmo conseguiu solucionar os tabuleiros gerados aleatoriamente (Figura 1) e também os tabuleiros que foram escritos manualmente (Figuras 2 e 3), o agente resolve qualquer tabuleiro válido de Sudoku. Fizemos o código pelo Jupyter Notebook, e disponibilizamos também um código em Python, ambos estão disponíveis pelo repositório GitHub, o link está anexo neste arquivo.

Figura 1 – Tabuleiro gerado aleatoriamente e a resolução

```
In [11]: Printgrid(aleatorio)
```

0	3	0	8	0	0	0	0	0
0	0	0	4	0	0	0	5	0
0	0	0	0	0	0	0	0	3
0	0	0	0	0	0	4	0	0
0	4	0	0	2	0	0	8	0
0	0	0	7	0	0	0	0	0
4	0	0	0	0	0	2	0	0
0	0	0	0	0	0	8	0	1
0	8	0	0	0	0	0	0	0

```
In [12]: resultadoaleatorio = resolucao_sudoku(aleatorio)
Printgrid(resultadoaleatorio[1])
```

1	3	2	8	5	6	7	4	9
6	7	9	4	3	2	1	5	8
8	5	4	1	7	9	6	2	3
9	6	3	5	8	1	4	7	2
7	4	1	6	2	3	9	8	5
5	2	8	7	9	4	3	1	6
4	1	5	9	6	8	2	3	7
3	9	7	2	4	5	8	6	1
2	8	6	3	1	7	5	9	4

Figura 2 – Tabuleiro de dificuldade Normal inserido manualmente e a resolução

```
In [13]: # ===== Dificuldade: normal =====
Printgrid(Tab_def)
```

5	1	7	6	0	0	0	3	4
2	8	9	0	0	4	0	0	0
3	4	6	2	0	5	0	9	0
6	0	2	0	0	0	0	1	0
0	3	8	0	0	6	0	4	7
0	0	0	0	0	0	0	0	0
0	9	0	0	0	0	0	7	8
7	0	3	4	0	0	5	6	0
0	0	0	0	0	0	0	0	0

```
In [14]: normal = resolucao_sudoku(Tab_def)
Printgrid(normal[1])
```

5	1	7	6	9	8	2	3	4
2	8	9	1	3	4	7	5	6
3	4	6	2	7	5	8	9	1
6	7	2	8	4	9	3	1	5
1	3	8	5	2	6	9	4	7
9	5	4	7	1	3	6	8	2
4	9	5	3	6	2	1	7	8
7	2	3	4	8	1	5	6	9
8	6	1	9	5	7	4	2	3

Figura 3 – Tabuleiro de dificuldade Expert inserido manualmente e a resolução

```
In [17]: # ===== Dificuldade: expert =====
Printgrid(Tab_def_expert)
```

0	0	0	1	0	0	0	7	0
0	3	0	0	0	0	0	6	0
0	0	2	0	6	4	5	0	0
0	0	0	0	0	0	0	4	0
0	5	0	0	0	0	0	0	0
0	0	0	2	0	6	8	9	1
0	0	0	9	0	0	0	0	7
8	6	5	0	2	0	0	0	0
0	0	1	0	0	3	0	0	0

```
In [18]: expert = resolucao_sudoku(Tab_def_expert)
Printgrid(expert[1])
```

5	8	6	1	3	2	4	7	9
4	3	7	5	9	8	1	6	2
9	1	2	7	6	4	5	3	8
6	2	9	8	7	1	3	4	5
1	5	8	3	4	9	7	2	6
3	7	4	2	5	6	8	9	1
2	4	3	9	1	5	6	8	7
8	6	5	4	2	7	9	1	3
7	9	1	6	8	3	2	5	4

Links:

Github do projeto: <https://github.com/geisalaiane/Sudoku>