Eigenstate

Programação em Lógica

Relatório Intercalar

Grupo Eigenstate\_4

* Miguel Jorge Maia de Magalhães Barros Lançós up201506342
* Tiago Pinho Cardoso up201605762

## Introdução

O jogo Eigenstate é um jogo de tabuleiro para duas pessoas inventado por Martin Grider em 2016.  
Baseia-se em dois tipos de peças colocadas num tabuleiro com um conjunto simples de regras que aumentam a complexidade do jogo à medida que este avança.

A ideia do jogo foi obtida num auto-desafio do autor que consistia em pensar numa ideia para um jogo novo todos os dias. A ideia para eigenstate veio do desejo de criar um jogo com uma mecânica de movimentos idêntica à do jogo “The Duke and Onitama”. Um mês depois o autor já tinha um tabuleiro protótipo e através de tentativa erro foi aperfeiçoando as regras até chegar ao jogo atual, que iremos implementar em prolog.

## Constituição

O jogo é constituído por um tabuleiro, tipo xadrez, com 36 divisões (6x6), por 12 peças, distribuídas pelos 2 jogadores, que contêm 25 buracos (5x5) onde se inserem os “eigenstates” à medida que o jogo vai avançando.

|  |  |
| --- | --- |
| Uma peça com os 25 buracos. | Tabuleiro de Eigenstate |

## 

## Posição inicial

No início do jogo cada jogador coloca as suas 6 peças nas 6 posições do tabuleiro mais perto de si.

Cada peça terá dois “eigenstates”, um na posição do centro (vermelho) e outro (branco) na posição imediatamente a seguir na direção do adversário. Este último eigenstate permitirá, na jogada inicial, o movimento da peça para a posição em frente.

|  |  |
| --- | --- |
| **game board after setup** |  |

## Jogadas

Os dois jogadores alternam entre jogadas e só passa para a vez do jogador seguinte após a jogada estar completa.

Cada jogada consiste em 3 partes:

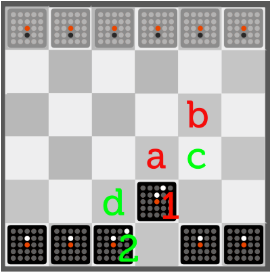
1. Movimento da peça;
2. Inserção de um “eigenvalue”;
3. Inserção do segundo “eigenvalue”.

Estas 3 tarefas devem ser executadas por esta exata ordem podendo, no entanto, para algum passo caso seja impossível a sua execução.

## 

## Movimentos legais

Cada peça tem 25 buracos que, fora o central, indicam a habilidade de movimento relativo da mesma.



A peça 1 pode mover-se para a posição “a” ou “b” na próxima jogada. Estes movimentos correspondem aos eigenstates colocados na peça. Analogamente a peça 2 tem “d” e “c” como possíveis movimentos, notando-se que as peças podem saltar por cima de outras peças.

As peças não podem sair do tabuleiro, serem rodadas nem os seus eigenvalues podem ser removidos ou trocados de lugar dentro da peça.

Quando um jogador move uma peça para cima de outra captura-a, eliminando-a do jogo. É possível capturar as próprias peças mas é desaconselhado de um ponto de vista tático.

## Objetivo do jogo

O jogador progride no jogo à medida que vai comendo peças do seu adversário.

O jogo acaba quando um jogador ficar só com uma peça perdendo assim a partida. Alternativamente o jogo acaba se ambos os jogadores tiverem duas peças e um deles tiver uma peça completamente preenchida, ganhando assim esse jogo.

## 

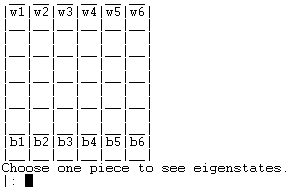
## Representação interna e visualização correspondente

### Representação inicial

A representação geral do tabuleiro na posição inicial, em prolog, pode ser escrita da seguinte forma:

|  |
| --- |
| **initial\_board([[w1,w2,w3,w4,w5,w6],**  **[x,x,x,x,x,x],**  **[x,x,x,x,x,x],**  **[x,x,x,x,x,x],**  **[x,x,x,x,x,x],**  **[b1,b2,b3,b4,b5,b6]]).** |

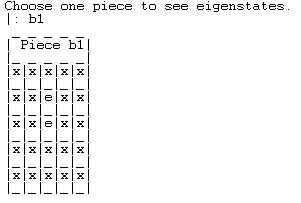
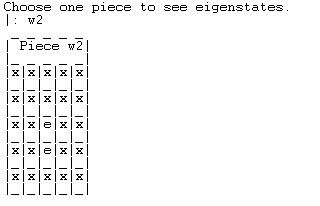
A visualização do estado inicial corresponderá a um tabuleiro em que os átomos “w(x)” corresponde ao jogador 1 e os átomos “b(x)” representam as peças do jogador 2.



Os eigenstate de cada peça são representados através de predicados dinâmicos:

|  |
| --- |
| **:-dymanic pieces/2.**  **pieces(w2, [[x,x,x,x,x],**  **[x,x,x,x,x],**  **[x,x,e,x,x],**  **[x,x,e,x,x],**  **[x,x,x,x,x]]).**  **pieces(b1, [[x,x,x,x,x],**  **[x,x,e,x,x],**  **[x,x,e,x,x],**  **[x,x,x,x,x],**  **[x,x,x,x,x]]).** |

Onde “e” representa a presença de um eigenstate nessa posição.



### Representação intermédia

O jogo irá continuar à medida que os jogadores vão tentando comer as peças do adversário.

|  |
| --- |
| **intermediate\_board([[x,x,x,x,x,x],**  **[w1,x,x,x,x,x],**  **[x,x,b4,b6,x,w6],**  **[x,b1,x,x,b3,x],**  **[x,x,w3,x,x,x],**  **[x,x,x,x,x,x]]).** |

### 

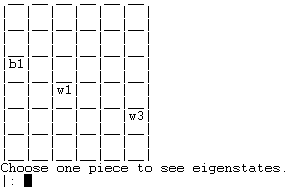
### 

### Representação Final

No final do jogo um dos jogadores terá apenas uma peça (caso 1) ou ambos têm duas peças mas uma delas está completamente preenchida com eigenstates (caso 2).

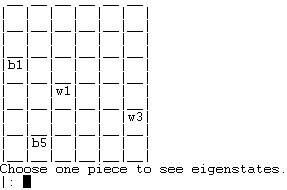
Exemplo caso 1:

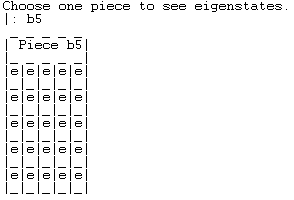
|  |
| --- |
| **final\_board([[x,x,x,x,x,x],**  **[x,x,x,x,x,x],**  **[b1,x,x,x,x,x],**  **[x,x,w1,x,x,x],**  **[x,x,x,x,x,w3],**  **[x,x,x,x,x,x]]).** |



Exemplo caso 2:

|  |
| --- |
| **final\_board2([[x,x,x,x,x,x],**  **[x,x,x,x,x,x],**  **[b1,x,x,x,x,x],**  **[x,x,w1,x,x,x],**  **[x,x,x,x,x,w3],**  **[x,b5,x,x,x,x]]).** |





## Visualização do tabuleiro

A visualização é feita através de um conjunto de funções que recursivamente fazem a mostra do tabuleiro e das peças.

|  |
| --- |
| **display\_game(Board, Player) :-**  **display\_limite\_sup,**  **display\_line(Board).** |

|  |
| --- |
| **display\_limite\_sup :-**  **write(' \_\_ \_\_ \_\_ \_\_ \_\_ \_\_ '),nl.**  **display\_limite :-**  **write('|\_\_|\_\_|\_\_|\_\_|\_\_|\_\_|'),nl.** |

|  |
| --- |
| **display\_line([]).**  **display\_line([X|Sobras]) :-**  **write('|'),**  **display\_ele(X),**  **nl,**  **display\_limite,**  **display\_line(Sobras).** |

|  |
| --- |
| **display\_ele([]).**  **display\_ele([Ele|Resto]) :-**  **write\_piece(Ele),**  **write('|'),**  **display\_ele(Resto).** |

|  |
| --- |
| **write\_piece(x) :- write(' ').**  **write\_piece(X) :- write(X).** |

|  |
| --- |
| **piece\_display(P) :-**  **pieces(P, Board),**  **piece\_display\_header(P),**  **piece\_display\_line(Board).** |

|  |
| --- |
| **piece\_display\_header(X) :-**  **write(' \_ \_ \_ \_ \_ '),nl,**  **write('| Piece '), write(X), write('|'),nl,**  **write('|\_ \_ \_ \_ \_|'),nl.** |

|  |
| --- |
| **piece\_display\_line([]).**  **piece\_display\_line([X|Xs]) :-**  **piece\_display\_eigenstate(X),**  **write('|'),nl,**  **write('|\_|\_|\_|\_|\_|'),nl,**  **piece\_display\_line(Xs).** |

|  |
| --- |
| **piece\_display\_eigenstate([]).**  **piece\_display\_eigenstate([X|Xs]) :-**  **write('|'),**  **write(X),**  **piece\_display\_eigenstate(Xs).** |

## 

## Bibliografia

<http://chesstris.com/2018/04/15/eigenstate-on-bgg/>

https://docs.google.com/document/d/1V9pzs\_HOobASODl9hpyOfJR9YOPF5HLwYxNjPkoatm4/edit#heading=h.xn7mtc37hp5b