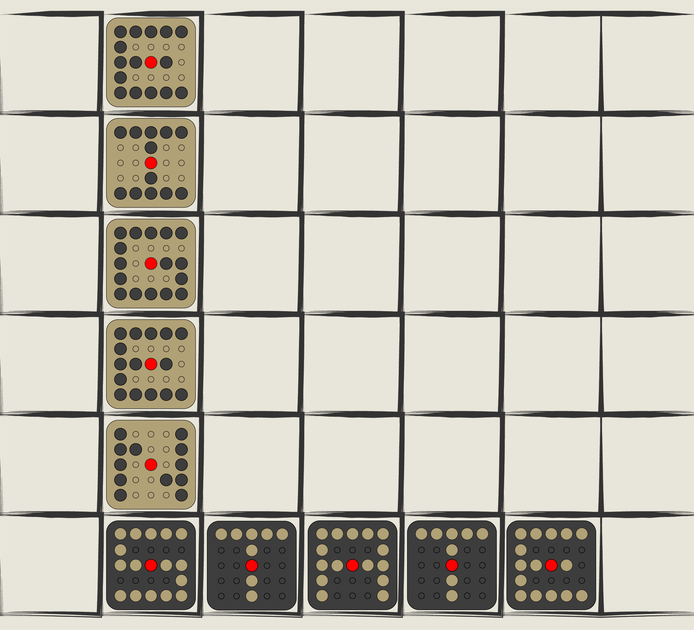


**Relatório Intercalar**

Programação em Lógica

(21 de outubro de 2018)



André Filipe Pinto Esteves [up201606673@fe.up.pt](mailto:up201606673@fe.up.pt)

Luís Diogo dos Santos Teixeira da Silva [up201503730@fe.up.pt](mailto:up201503730@fe.up.pt)

## História

Eigenstate foi criado em 2016 por Martin Grider, um desenvolvedor de software e criador de jogos amador americano. O jogo resulta de uma mistura entre as mecânicas de movimento de outros dois jogos abstratos mais aclamados: The Duke (criado por Jeremy Holcomb e Stephen McLaughlin) e Onitama (criado por Shimpei Sato). Um protótipo do jogo foi criado em 2017 e a sua primeira publicação está agendada para o final de 2018.

O termo Eigenstate refere-se ao possível movimento de uma partícula na física quântica. O autor do jogo refere que, apesar de tangencial, poder-se-ia atribuir ao jogo este tema, sendo as peças partículas num sistema e o objetivo de cada jogador é observar o sistema adversário primeiro.

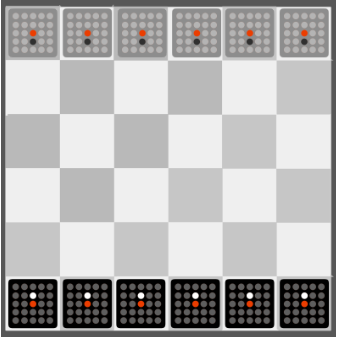
## Regras

Eigenstate é um jogo de estratégia abstrata para duas pessoas com regras simples de complexidade crescente com o decorrer do jogo.

*Peça*

Inicialmente cada jogador seleciona a cor com que pretende jogar.

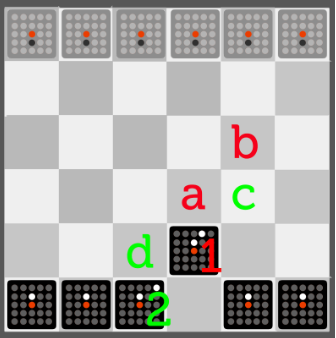
*Pino central*

Cada peça começa com dois pinos, o pino no centro representa a posição no tabuleiro e um pino adicional permitindo a peça ser movida um espaço para a frente. (figura 1)

*Pino dos movimentos possíveis*

Em cada jogada o jogador move uma das suas peças e acrescenta duas eigenstates (pinos) em qualquer uma das suas peças aumentando assim as possibilidades de movimento das peças nas seguintes jogadas. Todos os pinos numa peça exceto o centro representam os movimentos possíveis relativos à posição da peça no tabuleiro.

Figura 1

 Por exemplo (figura 2), o jogador com as peças pretas, na sua jogada, moveu a peça 1 para a frente, e, de seguida, adicionou um pino na peça 1 e na peça 2. Quando o jogador com as peças pretas voltar a jogar pode mudar a peça 1 para as posições a e b (pinos a branco representando os movimentos relativos possíveis) e a peça 2 para as posições c e d.

- Os pinos colocados nas peças nunca podem ser removidos. Assim, cada peça poderá sempre pelo menos ser movida para a frente.

Figura 2

- Peças podem saltar por cima de outras peças (movimento de 2 para c).

- Peças não podem mover-se para fora do tabuleiro.

- Peças não podem ser rodadas.

- Uma peça não pode mover-se para trás a não ser que tenha um pino atrás do pino central.

- Quando uma peça é movida para uma posição onde existe outra peça (do próprio jogador ou do adversário), essa peça é removida do jogo (não sendo, obviamente, uma boa ideia remover as próprias peças).

**Posicionamento dos pinos**

Os pinos têm de ser colocados nos espaços disponíveis das próprias peças que ainda estão em jogo. Em cada jogada os dois pinos podem ser colocados em diferentes peças.

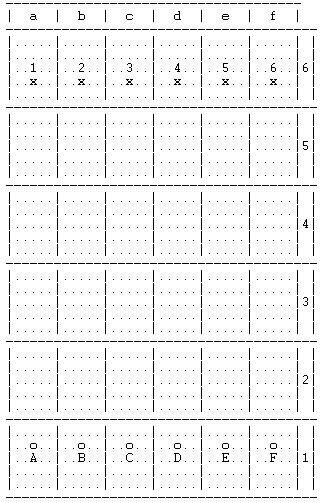
**Objetivo**

O objetivo principal é reduzir o oponente a apenas uma peça restante.

Como objetivo secundário, usado quando os dois jogadores têm exatamente duas peças restantes, o jogador deve, em vez de seguir o objetivo principal, (o qual seria impossível de alcançar, a não ser que seja propositado), preencher todos os pinos de uma das peças restantes. Assim, o jogo nunca acaba em empate.

# 

# Representação interna do estado do jogo

**Estado Inicial do Jogo**

piece(p1, [ [., ., ., ., .],

[., ., ., ., .],

[., ., 1, ., .],

[., ., x, ., .],

[., ., ., ., .] ]).

piece(p2, [ [., ., ., ., .],

[., ., ., ., .],

[., ., 2, ., .],

[., ., x, ., .],

[., ., ., ., .] ]).

piece(p3, [ [., ., ., ., .],

[., ., ., ., .],

[., ., 3, ., .],

[., ., x, ., .],

[., ., ., ., .] ]).

piece(p4, [ [., ., ., ., .],

[., ., ., ., .],

[., ., 4, ., .],

[., ., x, ., .],

[., ., ., ., .] ]).

piece(p5, [ [., ., ., ., .],

[., ., ., ., .],

[., ., 5, ., .],

[., ., x, ., .],

[., ., ., ., .] ]).

piece(p6, [ [., ., ., ., .],

[., ., ., ., .],

[., ., 6, ., .],

[., ., x, ., .],

[., ., ., ., .] ]).

piece(pA, [ [., ., ., ., .],

[., ., o, ., .],

[., ., 'A', ., .],

[., ., ., ., .],

[., ., ., ., .] ]).

piece(pB, [ [., ., ., ., .],

[., ., o, ., .],

[., ., 'B', ., .],

[., ., ., ., .],

[., ., ., ., .] ]).

piece(pC, [ [., ., ., ., .],

[., ., o, ., .],

[., ., 'C', ., .],

[., ., ., ., .],

[., ., ., ., .] ]).

piece(pD, [ [., ., ., ., .],

[., ., o, ., .],

[., ., 'D', ., .],

[., ., ., ., .],

[., ., ., ., .] ]).

piece(pE, [ [., ., ., ., .],

[., ., o, ., .],

[., ., 'E', ., .],

[., ., ., ., .],

[., ., ., ., .] ]).

piece(pF, [ [., ., ., ., .],

[., ., o, ., .],

[., ., 'F', ., .],

[., ., ., ., .],

[., ., ., ., .] ]).

piece(e, [ [., ., ., ., .],

[., ., ., ., .],

[., ., ., ., .],

[., ., ., ., .],

[., ., ., ., .] ]).

board( [[p1, p2, p3, p4, p5, p6],

[e, e, e, e, e, e],

[e, e, e, e, e, e],

[e, e, e, e, e, e],

[e, e, e, e, e, e],

[pA, pB, pC, pD, pE, pF]]).

**Estado Intermédio do Jogo**

piece(pA, [ [., ., ., ., .],

[., ., o, ., .],

[., ., 'A', ., .],

[., ., ., ., .],

[., ., ., ., .] ]).

piece(pB, [ [., ., ., ., .],

[., ., o, ., .],

[., ., 'B', o, .],

[., ., o, ., .],

[., ., ., ., .] ]).

piece(pC, [ [., ., ., ., .],

[., ., o, ., .],

[., ., 'C', o, o],

[., ., ., ., .],

[., ., ., ., .] ]).

piece(pD, [ [., ., o, ., .],

[., ., o, ., .],

[., o, 'D', ., .],

[., ., ., ., .],

[., ., ., ., .] ]).

piece(pE, [ [., ., ., ., .],

[., ., o, ., .],

[., ., 'E', ., .],

[., ., ., ., .],

[., ., ., ., .] ]).

piece(pF, [ [., ., ., ., .],

[., ., o, ., .],

[., ., 'F', ., .],

[., ., ., ., .],

[., ., ., ., .] ]).

piece(p1, [ [., ., ., ., .],

[., ., ., ., .],

[., ., 1, ., .],

[., ., x, ., .],

[., ., ., ., .] ]).

piece(p2, [ [., ., ., ., .],

[., ., ., ., .],

[., ., 2, ., .],

[., ., x, ., .],

[x, ., ., ., .] ]).

piece(p3, [ [., ., ., ., .],

[., ., ., ., .],

[x, ., 3, ., .],

[., ., x, ., .],

[., ., x, ., .] ]).

piece(p4, [ [., ., ., ., .],

[., ., ., ., .],

[., ., 4, ., .],

[., ., x, ., .],

[., ., ., ., .] ]).

piece(p5, [ [., ., ., ., .],

[., ., ., ., .],

[., ., 5, ., .],

[., ., x, ., .],

[., ., ., ., .] ]).

piece(p6, [ [., ., ., ., .],

[., ., ., ., .],

[., ., 6, ., .],

[., ., x, ., .],

[., ., ., ., .] ]).

piece(e, [ [., ., ., ., .],

[., ., ., ., .],

[., ., ., ., .],

[., ., ., ., .],

[., ., ., ., .] ]).

board( [[e, e, e, e, e, e],

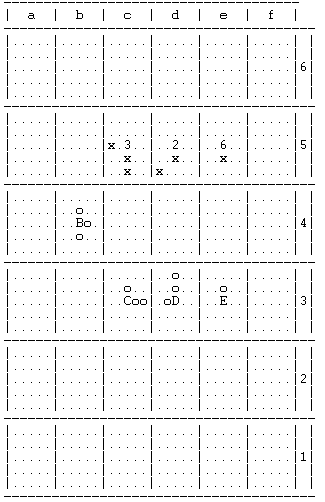
[e, e, p3, p2, p6, e],

[e, pB, e, e, e, e],

[e, e, pC, pD, pE, e],

[e, e, e, e, e, e],

[e, e, e, e, e, e]]).



**Estado Final do Jogo**

piece(p1, [ [., ., x, ., .],

[., ., x, ., .],

[., ., 1, x, x],

[x, x, x, ., .],

[x, x, x, ., .] ]).

piece(p2, [ [., ., ., ., .],

[., ., ., ., .],

[., ., 2, ., .],

[., ., x, ., .],

[x, ., ., ., .] ]).

piece(p3, [ [., ., ., ., .],

[., ., ., ., .],

[x, ., 3, ., .],

[., ., x, ., .],

[., ., x, ., .] ]).

piece(p4, [ [., ., ., ., .],

[., ., ., ., .],

[., ., 4, ., .],

[., ., x, ., .],

[., ., ., ., .] ]).

piece(p5, [ [., ., ., ., .],

[., ., ., ., .],

[., ., 5, ., .],

[., ., x, ., .],

[., ., ., ., .] ]).

piece(p6, [ [., ., ., ., .],

[., ., ., ., .],

[., ., 6, ., .],

[., ., x, ., .],

[., ., ., ., .] ]).

piece(pA, [ [., ., ., ., .],

[., ., o, ., .],

[., ., 'A', ., .],

[., ., ., ., .],

[., ., ., ., .] ]).

piece(pB, [ [o, ., ., ., .],

[., o, o, ., .],

[., ., 'B', o, o],

[., ., o, ., .],

[., ., ., ., .] ]).

piece(pC, [ [., ., ., ., .],

[., ., o, ., .],

[., ., 'C', o, o],

[., ., ., ., .],

[., ., ., ., .] ]).

piece(pD, [ [., ., o, ., .],

[., ., o, ., .],

[., o, 'D', ., .],

[., ., ., ., .],

[., ., o, o, o] ]).

piece(pE, [ [., ., ., ., .],

[., ., o, ., .],

[., ., 'E', ., .],

[., ., ., ., .],

[., ., ., ., .] ]).

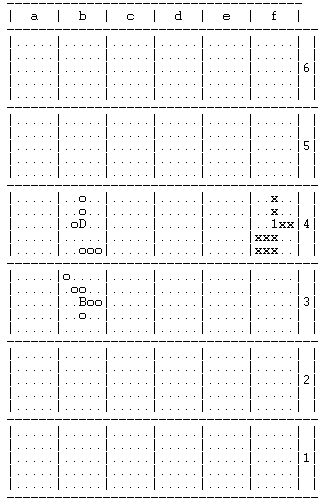
piece(pF, [ [., ., ., ., .],

[., ., o, ., .],

[., ., 'F', ., .],

[., ., ., ., .],

[., ., ., ., .] ]).



board( [[e, e, e, e, e, e],

[e, e, e, e, e, e],

[e, pD, e, e, e, p1],

[e, pB, e, e, e, e],

[e, e, e, e, e, e],

[e, e, e, e, e, e]]).

piece(e, [ [., ., ., ., .],

[., ., ., ., .],

[., ., ., ., .],

[., ., ., ., .],

[., ., ., ., .] ]).

# 

# Visualização do tabuleiro em modo texto

As imagens presentes na secção anterior foram geradas por este código:

display\_line([], Original, 3, X\_val) :-

write('|'),

write(X\_val),

write('|'),

nl,

display\_line(Original, Original, 2, X\_val).

display\_line([], Original, N\_line, X\_val) :-

Next is N\_line - 1,

N\_line > 0,

N\_line =\= 3,

write('| |'),

nl,

display\_line(Original, Original, Next, X\_val).

display\_piece\_line(N\_line, [\_|Tail], Current\_line) :-

Next is Current\_line - 1,

Current\_line > N\_line,

display\_piece\_line(N\_line, Tail, Next).

display\_piece\_line(N\_line, [Head|\_], N\_line) :-

display\_piece\_element(Head).

display\_piece\_element([]).

display\_piece\_element([Head|Tail]) :-

write(Head),

display\_piece\_element(Tail).

display\_horizontal:-

write('-------------------------------------'), nl,

write('| a | b | c | d | e | f |'), nl.

display\_game([Head|Tail],player1) :-

display\_horizontal,

display\_board([Head|Tail], player1, 6).

display\_board([], \_, \_) :-

write('---------------------------------------'), nl.

display\_board([Head|Tail], \_, X\_val) :-

write('---------------------------------------'), nl,

display\_line(Head, Head, 5, X\_val), nl,

X\_next is X\_val - 1,

display\_board(Tail, \_, X\_next).

display\_line([], \_, 1, \_) :- write('| |').

display\_line([Head|Tail], Original, N\_line, X\_val) :-

write('|'),

piece(Head, Piece\_Rep),

display\_piece\_line(N\_line, Piece\_Rep, 5),

display\_line(Tail, Original, N\_line, X\_val).