Eigenstate

Ricardo Jorge de Araújo Ferreira - up200305418

Grupo Eigenstate2

Projecto no github: `https://github.com/ricardojaferreira/Eigenstate`

November 18, 2018

1 HISTÓRIA

Eigenstate é um jogo de tabuleiro criado por Martin Grider [1]. É um jogo para dois jogadores com regras simples mas que cresce em complexidade à medida que o jogo avança. Este jogo ainda não está disponível no mercado, mas prêve-se a criação de um projecto no kickstarter para isso acontecer.
Eigenstate foi inspirado nos jogos The Duke [2] e Onitama [3]. Durante o seu desenvolvimento o autor escreveu vários cenários de jogabilidade e condições de victória, tendo pensado até num cenário semelhante ao checkmate do Xadréz. Mas no final, durante uma apresentação do jogo, decidiu manter apenas duas regras básicas. O jogador na sua vez só tem que mover uma peça e colocar dois novos "pegs" para possibilitar movimentos numa nova jogada. Termina com a eliminação das peças adversárias, no entanto, a condição de vitória ainda pode ser alterada na versão final.
O nome Eigenstate vem de um termo da física quântica que se refere ao possível movimento de uma partícula [4].

2 REGRAS

2.1 CONFIGURAÇÃO

O jogo joga-se num tabuleiro quadrado com 36 células (6x6). Cada jogador escolhe uma cor e coloca todas as peças dessa cor, 6 peças por jogador, no seu lado do tabuleiro.

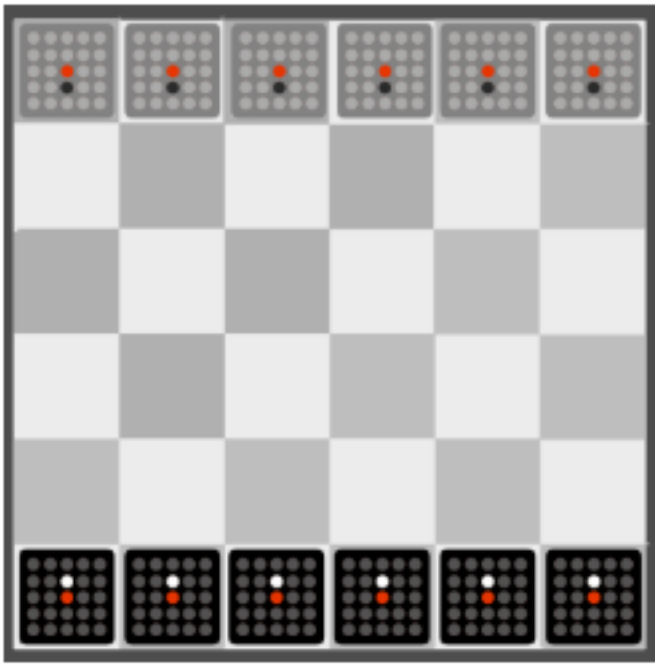

Figure 2.1: Eigenstate: Configuração inicial. [5]

Todas as peças começam com dois pegs, um no centro que representa a posição actual da peça e outro peg que permite a peça avançar uma casa.

2.2 JOGABILIDADE

Cada jogador no seu turno, se possível, deve:

1. Mover uma das suas peças em conformidade com os pegs (como explicado mais à frente);

2. Colocar dois pegs em qualquer uma das suas peças, na mesma, ou diferentes.

O movimento das peças é regido pelo seguinte. Todos os pegs colocados na peça representam possíveis jogadas relativamente à sua posição actual. A posição actual é marcada pelo peg colocado no centro, tipicamente com uma cor diferente dos outros.
Este movimento está ilustrado na figura 2.2. Na figura é possível ver que o jogador moveu a peça 1 e colocou dois novos pegs, um na peça que moveu e outro na peça 2. Na próxima jogada, este jogador poderá mover a peça 1 para as posições a ou b e a peça 2 pode ser colocada nas posições d ou c. Estes movimentos correspondem à posição dos pegs em cada peça.
O movimento das peças obdece ainda às seguintes regras:

- Os pegs nunca podem ser removidos das peças, garantindo que as peças podem sempre avançar uma casa;

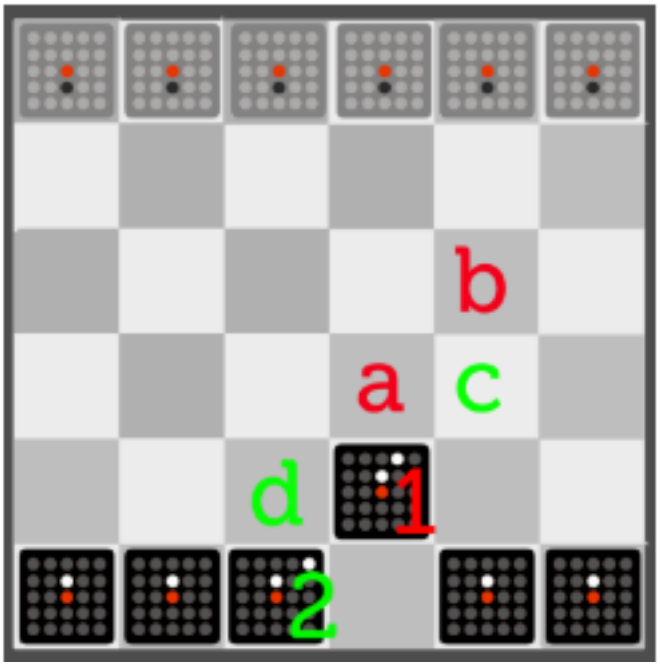

Figure 2.2: Eigenstate: Movimento de peças. [5]

- As peças podem saltar por cima de outras peças;
- As peças não podem sair do tabuleiro;
- As peças não rodam;
- As peças só se podem mover para trás se tiverem pegs atrás do marcador central;
- Uma peça que se mova para a mesma posição de outra peça, faz com que a outra seja removida do tabuleiro;
- Peças do mesmo jogador, podem mover-se para cima uma das outras e as peças têm que ser removidas conforme a regra anterior.

A colocação de pegs, obdece às seguintes regras:

- Só podem ser colocados em espaços livres, das peças do jogador não capturadas;
- Em cada turno podem ser colocados 2 pegs em peças diferentes ou na mesma;
- Não é necessário colocar nenhum peg na peça movida nessa ronda.

2.3 Condição de Victória

A forma principal de vitória consiste em reduzir o adversário a apenas uma peça. Num caso em que ambos os jogadores tenham duas peças cada um, a vitória pode ser conseguida se um jogador colocar pegs em todas as posições de uma das suas peças.

3 IMPLEMENTAÇÃO EM PROLOG

3.1 REPRESENTAÇÃO INTERNA DO ESTADO DO JOGO

Este jogo terá como representações principais o tabuleiro e a peça. Sendo que as peças são também elas tabuleiros para a colocação de pegs, a representação destes dois elementos será semelhante, embora a forma como se interpretam os resultados seja bastante diferente. Ambos terão uma representação baseada em listas de listas.
A representação inicial do tabuleiro será a seguinte:

```
1 [
2                     ['b1','b2','b3','b4','b5','b6'],
3                     ['e','e','e','e','e','e'],
4                     ['e','e','e','e','e','e'],
5                     ['e','e','e','e','e','e'],
6                     ['e','e','e','e','e','e'],
7                     ['a1','a2','a3','a4','a5','a6']
8 ]
```

O tabuleiro é uma lista com 6 sub-listas, uma para cada linha do tabuleiro. Com esta estrutura podemos pesquisar o tabuleiro de uma forma análoga a uma matriz, pesquisando por linha e coluna, tirando partido da divisão das listas em *Head* e *Tail*. Por exemplo, um predicado *boardLine(X,L)* em que X é o número da linha e L o tabuleiro, pode ser utilizado de forma recursiva para percorrer na vertical. Um excerto de código para esta finalidade será:

```
1 boardLine(_Z,[]).
2
3 boardLine(X,[H|T]):-
4         Line is 1+X,
5         boardColumn(C,H),
6         boradLine(Line,T).
```

O predicado *boardColumn(C,H)* terá uma função semelhante ao anterior mas irá percorrer a lista na horizontal, garantindo que são obtidos todos os elementos do tabuleiro. De notar que a recursividade apenas termina quando a lista fica vazia, significando que foram visitadas todas as células.
Tendo uma forma de pesquisar o tabuleiro, falta uma tradução do significado de cada átomo. Para isso podem ser utilizados predicados e estes adicionados à base de conhecimento do programa. Estes predicados serão de particular interesse para a impressão do tabuleiro, uma vez que os átomos já têm valores bastante expressivos, ou seja,

- 'e' - célula vazia;
- 'aX' - célula com peça X do jogador 1;
- 'bX' - célula com peça X do jogador 2.

Como cada peça pode ter um estado diferente ao longo do jogo (dependendo do número de pegs que tem) é necessário uma representação individual de cada uma.
A aborgadem para a representação das peças será semelhante à tomada para o tabuleiro, embora com alguma complexidade acrescida. Cada peça será representada por uma lista com o seguinte formato (peça no estado inicial):

```
[
        ['o','o','o','o','o'],
        ['o','o','*','o','o'],
        ['o','o','@','o','o'],
        ['o','o','o','o','o'],
        ['o','o','o','o','o']
]
```

O método para percorrer as peças será o mesmo utilizado para o tabuleiro, pois cada peça, é ela própria um tabuleiro. Cada átomo nas peças terá o seguinte significado:

- 'o' - célula vazia, pode ser colocado um peg;
- '*' - célula com peg;
- '@' - célula central que marca a posição da peça no tabuleiro.

Uma peça num estado mais avançado do jogo poderá ter o seguinte estado:

```
[
        ['*','o','o','o','o'],
        ['o','*','*','o','o'],
        ['o','o','@','o','o'],
        ['o','o','o','*','o'],
        ['o','o','*','o','o']
]
```

O tabuleiro por sua vez poderá estar num estado semelhante ao seguinte:

```
[
        ['e','e','e','e','b5','b6'],
        ['e','a5','e','e','b3','e'],
        ['e','e','e','e','e','e'],
        ['e','b2','e','e','e','e'],
        ['e','e','e','e','e','e'],
        ['a1','e','e','e','e','a6']
]
```

Num caso de victória por eliminação das peças do adversário, o tabuleiro estará num estado semelhante ao seguinte:

```
[
        ['e','e','e','e','b5','b6'],
```

```
3           ['e','e','e','e','b3','e'],
4           ['e','e','e','e','a3','e'],
5           ['e','b2','e','e','e','e'],
6           ['e','e','e','e','e','e'],
7           ['e','e','e','e','e','e']
8   ]
```

Neste caso, o jogador 2 terá a victória pois o jogador 1 apenas tem a peça a3.
Num cenário em que cada jogador fica reduzido a apenas 2 peças cada, vence o jogador que conseguir colocar uma das suas peças no seguinte estado.

```
1   [
2           ['*','*','*','*','*'],
3           ['*','*','*','*','*'],
4           ['*','*','@','*','*'],
5           ['*','*','*','*','*'],
6           ['*','*','*','*','*']
7   ]
```

Na próxima secção serão apresentadas algumas imagens dos estados do tabuleiro durante o jogo.

3.2 VISUALIZAÇÃO DO TABULEIRO EM MODO DE TEXTO

O jogo está pensado para correr na consola e como tal terá uma representação com carácteres ASCII. Para a representação funcionar correctamente, recomenda-se uma consola com font do tipo monospace, para todos os carácteres ocuparem o mesmo espaço. As peças são representadas pelos mesmos simbolos utilizados na representação interna do jogo, as células do tabuleiro vazias utilizam um predicado de tradução que se baseia no facto da posição da célula ser par ou impar, para assim se conseguir uma distinçao entre células contíguas. Os predicados são:

```
1   boardCell(1,'e',':␣:␣:␣:␣:␣:␣:').
2   boardCell(0,'e','␣␣␣␣␣␣␣␣␣␣␣␣').
```

o valor 0 ou 1 é conseguido através da avaliação do resto da divisição inteira por 2 do número da célula.
Uma representação inicial do tabuleiro pode ser vista na Figura 3.1. O Jogador 1 tem as peças verdes e o jogador 2 as peças vermelhas.
O tabuleiro numa fase de jogo intermédia pode ser vista na Figura 3.2 e na fase final na Figura 3.3.

Board Game Geek. Eigenstate: Board game, 2018. URL: https://boardgamegeek.com/boardgame/250725/eigenstate.

Board Game Geek. The duke: Board game, 2013. URL: https://boardgamegeek.com/boardgame/36235/duke.

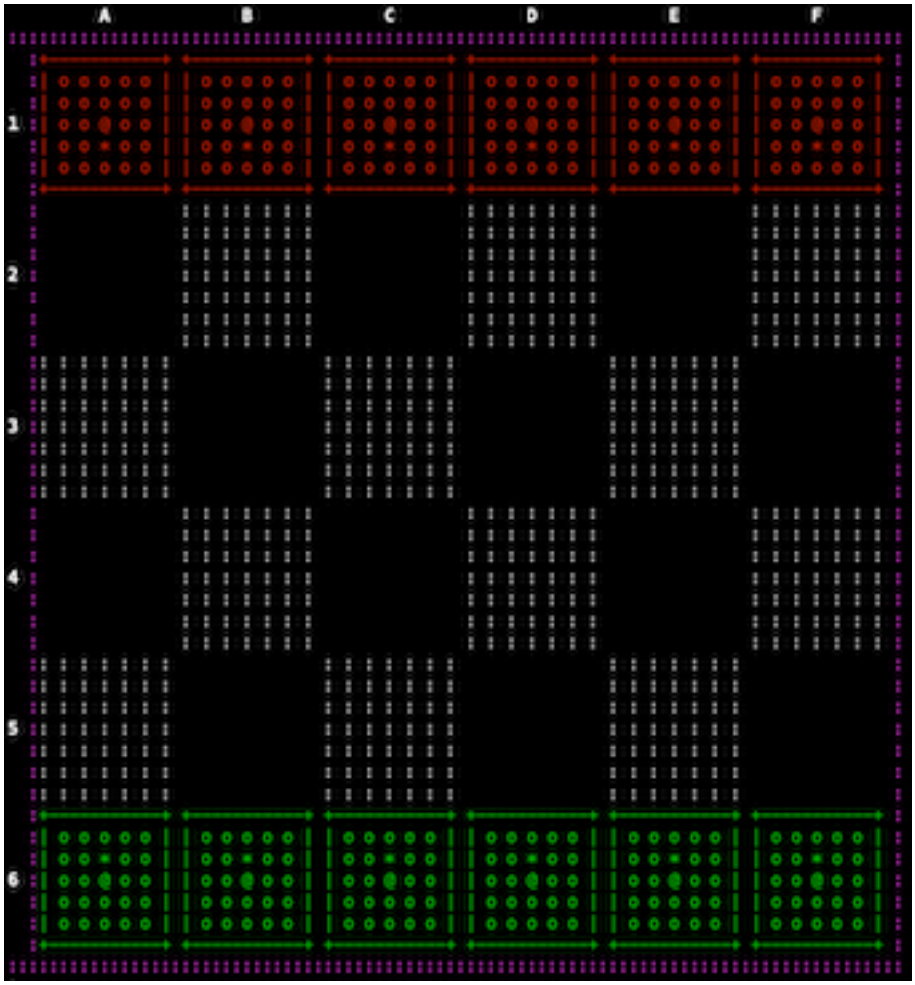

Figure 3.1: Tabuleiro estado inicial

Board Game Geek. Onitama: Board game, 2014. URL: https://boardgamegeek.com/boardgame/160477/onitama.

Martin Grider. Eigenstate on bgg, 2018. URL: http://chesstris.com/2018/04/15/eigenstate-on-bgg.

Martin Grider. Eigenstate pnp rules, 2018. URL: https://tinyurl.com/eigenstate.

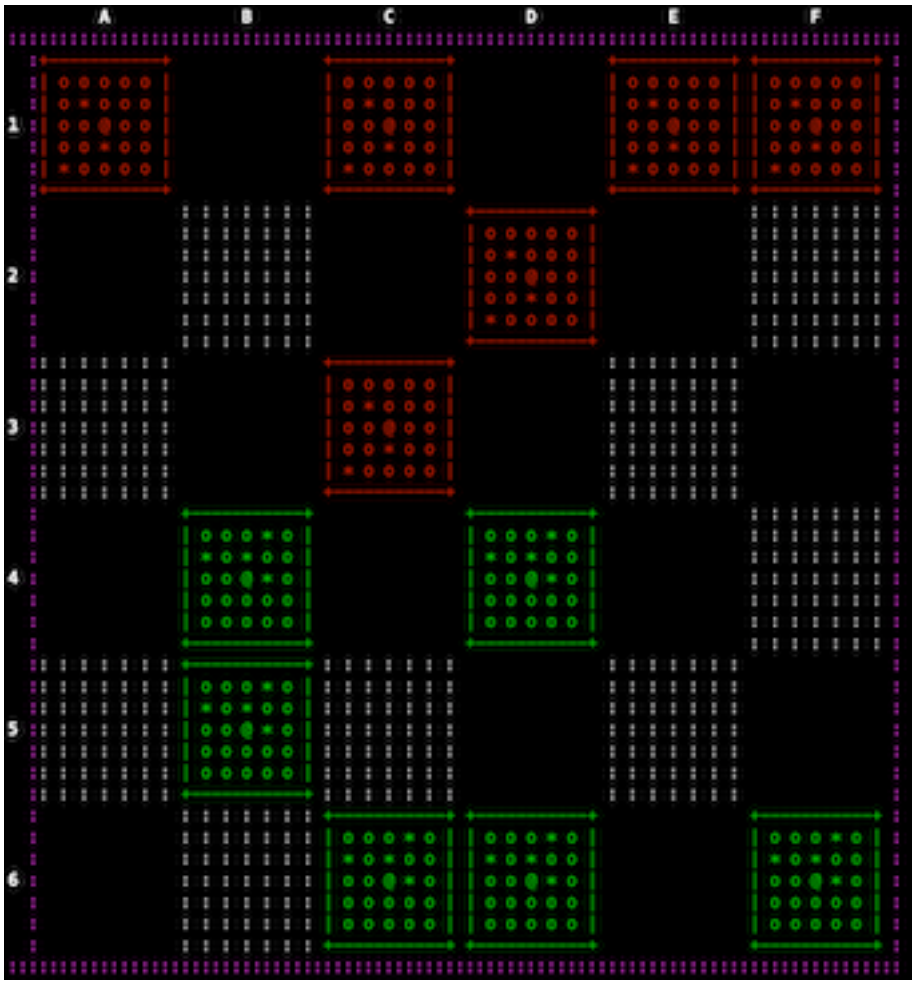

Figure 3.2: Tabuleiro estado intermédio

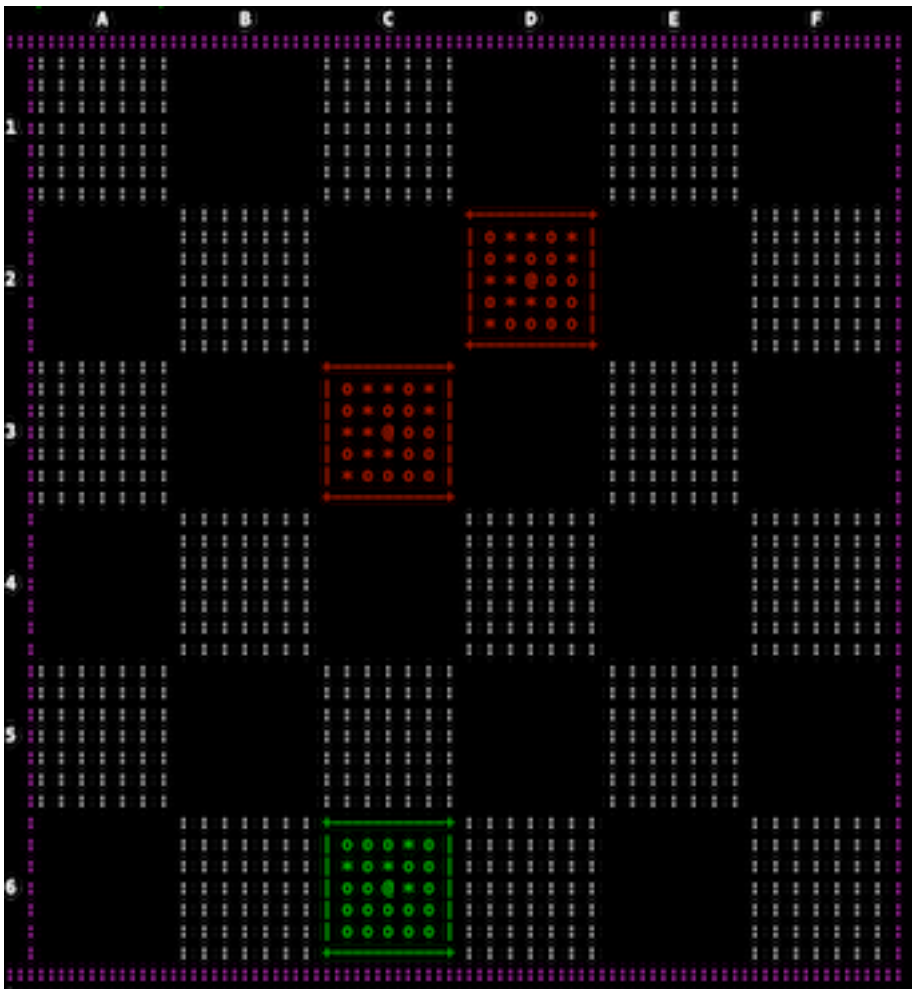

Figure 3.3: Tabuleiro estado final