Relatório do projeto de Programação em Lógica

**Trabalho 1**

** Quantum Leap\_2**

**Turma 4 Grupo:**

Joel Márcio Torres Carneiro 201100775

Miguel Geraldes Antunes Mendes

**Resumo:**

O problema abordado pelo nosso grupo trata-se da realização da lógica do jogo de tabuleiro Quantum Leap\_2. Foi feito um relatório intercalar onde foram especificadas algumas das funções finais do jogo, como por exemplo: as regras, representações de estados de jogo, visualização do tabuleiro em modo de texto e movimentos.

O objetivo do trabalho é fazer o código em linguagem de programação lógica capaz da realização da lógica do jogo em questão. Como tal foram criadas funções e listas de forma a resolver o problema em mãos.

O jogo de tabuleiro pode ser jogado em modo jogador vs computador ou jogador vs jogador.

Principais resultados…

Conclusões….

**Introdução**

O objetivo do trabalho é criar o código em linguagem de programação lógica que realize a lógica do jogo de tabuleiro Quantum Leap\_2. É um jogo com características peculiares que lhe trazem uma jogabilidade diferente do comum.

Motivação do trabalho….

Este relatório descreve as várias componentes do código que elaboramos. Começamos por descrever o jogo e as suas regras. De seguida mostramos a lógica utilizada no jogo, a representação do estado de jogo e a visualização em modo de texto do tabuleiro de jogo. Continuamos com a demonstração das jogadas válidas, com a execução de jogadas e com uma avaliação do tabuleiro. Na parte final do relatório falamos sobre o final do jogo, a jogada do computador e a interface do utilizador.

**Breve descrição do jogo:**

Quantum Leap é um jogo criado em 2013 pela Nestor Games.

O jogo toma lugar numa grelha hexagonal com cinco casas de cada lado. Um total de sessenta peças entram em jogo, sendo trinta peças pretas e trinta brancas. Inicialmente, todas as peças são colocadas de forma aleatória no tabuleiro, restando um único espaço livre. Jogando o jogador de peças brancas o primeiro turno, o outro jogador tem a oportunidade de trocar a posição de quaisquer duas peças que deseje.

Cada jogada é bastante peculiar, sendo que cada peça tem um potencial de salto correspondente ao número de peças aliadas que a rodeiam. Em cada jogada, esse potencial de salto é utilizado para eliminar uma peça adversária, tomando a posição desta última. Logo, numa jogada normal, hipotetizando que uma peça prestes a ser jogada se encontra rodeada de duas peças aliadas, pode mover-se através de 3 celas, sendo as duas primeiras as que efetivamente “saltou” com o seu potencial.

 Este tipo de jogadas repete-se, sendo o vencedor o último jogador a conseguir efetuar uma jogada válida. Consequentemente haverá uma altura em que um jogador não conseguirá utilizar uma das suas peças em conjunto com o seu potencial para eliminar uma peça adversária, apercebendo-se que perdeu.

Figura de possível estado inicial e do tabuleiro, com possível jogada de uma peça branca

**Lógica de Jogo:**

**1.Representação do estado do jogo:**

Tratando-se de uma grelha hexagonal, sugere-se um método alternativo para a representação do tabuleiro. Utilizando uma matriz normal com espaços tornados inválidos, podemos simular as posições e movimentos de peças de forma hexagonal, sem grandes esforços. Tome-se o exemplo da imgem fornecida anteriormente, tendo em conta que W = peças brancas, B = peças pretas e # denota um espaço inválido:  
ExampleBoard( [[W,W,B,W,W,#,#,#,#],

[B,B,W,W,B,B,#,#,#],

[B,W,W,B,B,B,W,#,#],

[W,B,B,W,W,B,W,B,#],

[B,B,W,W,B,B,W,B,\_],

[#,W,B,B,W,W,W,B,W],

[#,#,W,B,W,B,B,W,B],

[#,#,#,B,B,W,W,B,W],

[#,#,#,#,W,W,B,W,B]]).

Vejamos agora uma situação em que estejamos a meio do jogo:

MiddleBoard( [[W,\_,\_,\_,W,#,#,#,#],

[\_,W,\_,\_,\_,W,#,#,#],

[\_,\_,\_,W,B,B,\_,#,#],

[\_,W,\_,\_,B,\_,W,\_,#],

[\_,B,\_,\_,\_,\_,B,W,\_],

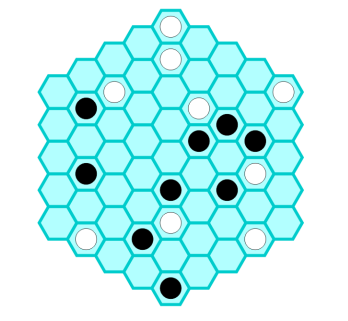
[#,\_,\_,\_,\_,B,\_,\_,W],

[#,#,\_,B,\_,\_,W,\_,\_],

[#,#,#,\_,\_,\_,B,\_,\_],

[#,#,#,#,\_,W,\_,\_,B]]).

Uma situação possível de fim de jogo:



EndBoard([[W,\_,\_,\_,W,#,#,#,#],

[\_,W,\_,\_,\_,\_,#,#,#],

[\_,\_,\_,W,B,B,\_,#,#],

[\_,W,\_,\_,B,\_,W,\_,#],

[\_,B,\_,\_,\_,\_,B,\_,\_],

[#,\_,\_,\_,\_,B,\_,\_,W],

[#,#,\_,B,\_,\_,W,\_,\_],

[#,#,#,\_,\_,\_,B,\_,\_],

[#,#,#,#,\_,W,\_,\_,B]]).

**2.Visualização do tabuleiro em modo de texto:**

Tomemos de novo este exemplo:

A representação sugerida para este exemplo de estado de jogo seria a seguinte:

 | W | W | B | W | W |

| B | B | W | W | B | B |

| B | W | W | B | B | B | W |

| W | B | B | W | W | B | W | B |

| B | B | W | W | B | B | W | B | \_ |

| W | B | B | W | W | W | B | W |

| W | B | W | B | B | W | B |

| B | B | W | W | B | W |

| W | W | B | W | B |  
  
  
 Os predicados necessários para representar o tabuleiro ao utilizador aquando do correr do jogo seriam:

board(current).

display(board(X)) :-

write(“ “),

/\* output de primeira linha \*/

write(“ “),

/\* output de segunda linha \*/

write(“ “),

/\* output de terceira linha \*/

write(“ “),

/\* output de quarta linha \*/

/\* output de quinta linha \*/

write(“ “),

/\* output de sexta linha \*/

write(“ “),

/\* output de sétima linha \*/

write(“ “),

/\* output de última oitava linha \*/

write(“ “),

/\* output de última linha \*/.

**3.Lista de jogadas válidas**

Obtenção de uma lista de jogadas possíveis. Exemplo: valid\_moves(+Board,-ListOfMoves)

**4.Execução de jogadas**

Validação e execução de uma jogada num tabuleiro, obtendo o novo estado do jogo. Exemplo: move(+Move,+Board,-NewBoard).

**5.Avaliação do Tabuleiro**

Avaliação do estado do jogo, que permitirá comparar a aplicação das diversas jogadas disponíveis. Exemplo: value(+Board,+Player, -Value).

**6.Final do Jogo**

Verificação do fim do jogo, com identificação do vencedor. Exemplo: game\_over(+Board, -Winner).

**7.Jogada do Computador**

Escolha da jogada a efetuar pelo computador, dependendo do nível de dificuldade. Por exemplo: choose\_move(+Level, +Board, -Move).

**Interface com o Utilizador**

Descrever o módulo de interface com o utilizador em modo de texto.

**Conclusões**

**Bibliografia**

**Movimentos:**

Só é possível movimentar peças em casos especiais. Se a pedra que quisermos mover não tiver nenhuma pedra da mesma cor (amiga) a sua volta, essa pedra não se poderá mexer. Neste caso o potencial da pedra é de zero. Se tiver uma pedra amiga à sua volta pode mover-se uma casa (potencial de um), se forem duas pedras amigas à sua volta pode movimentar-se duas casas (potencial de dois), e assim sucessivamente.

stone(x).

cellboard(c).

move(x,c).

O predicado *move* vai depender do potencial de cada pedra que se quer jogar.

**Anexos**