**FUSE**

Relatório Final



Mestrado Integrado em Engenharia Informática e Computação

Programação em Lógica

**Grupo 4**

Eduardo João Santana Macedo - 201703658

Francisco José Paiva Gonçalves - 201704790

Faculdade de Engenharia da Universidade do Porto

Rua Roberto Frias, sn, 4200-465 Porto, Portugal

15 de Novembro de 2019

**Índice**

Resumo 3

Introdução 4

O Jogo (Fuse) 4

Lógica do Jogo 4

Representação do estado jogo 4

Visualização do tabuleiro 5

Lista de jogadas válidas 5

Execução de jogadas 5

Final do jogo 5

Avaliação do tabuleiro 6

Jogada do computador 6

Conclusão 7

Anexos 8

**Resumo**

Foi-nos proposto desenvolver um jogo de tabuleiro numa linguagem de programação em lógica, Prolog. Escolhemos o Fuse, um jogo recentemente criado, para dois jogadores, que consiste em mover peças **brancas** e **pretas** das suas posições iniciais de maneira a no final do jogo criar o maior grupo consecutivo de peças dispostas ortogonalmente.

Este trabalho tinha como objetivo aprofundar e consolidar conceitos da linguagem Prolog, abordados nas aulas de Programação em Lógica. Foi benéfico ter um projeto como este, de maneira a ter um contacto aprofundado com a linguagem e a encontrar obstáculos que nos fazem evoluir.

Assim, concebemos um jogo de tabuleiro com 3 modos de jogo (Jogador v Jogador, CPU v CPU e Jogador v CPU).

As posições iniciais não podem conter peças nenhumas após a ficarem sem a peça inicial. Os movimentos das peças na vertical.

**1. Introdução**

O objetivo deste trabalho foi implementar um jogo de tabuleiro (Fuse), que tem interesse do ponto de vista da aprendizagem mais profunda e experimental da linguagem Prolog, já que há a necessidade de criar condições e regras para movimentar peças e também de determinar o resultado do jogo usando esta linguagem.

O relatório segue a seguinte estrutura:

* **O Jogo - Fuse**: Descrição do jogo e suas regras
* **Lógica do Jogo:** Descrição da lógica do jogo, mas desta vez da perspetiva da implementação. Esta secção subdivide-se em:
* **Representação do estado do jogo**: Exemplos visuais de estados de jogo (inicial, intermédio e final).
* **Visualização do Tabuleiro:** Descrição da lógica por trás dos predicados da disposição do tabuleiro
* **Lista de Jogadas Válidas:** Descrição da lógica por trás dos predicados que avaliam e geram jogadas válidas.
* **Execução e jogadas:** Descrição da evolução do jogo e de como é avaliada cada jogada.
* **Avaliação do tabuleiro:** Descrição dos predicados que analisam o estado do tabuleiro.
* **Final do Jogo:** Descrição dos predicados usados para verificar se o jogo já chegou ao fim.
* **Jogada do Computador:** Descrição dos predicados que geram jogadas automaticamente e jogando uma.
* **Interface disponível**: Descrição do interface criado que permite navegar nos menus do jogo.
* **Conclusões**: Ilações finais, comentários sobre o que foi implementado e reflexão.

**2. O Jogo - Fuse**

FUSE é um jogo de tabuleiro espanhol criado apenas em 2019 que iremos desenvolver usando a linguagem de Prolog. Trata-se de um jogo em que cada jogador joga alternadamente para no final obter a melhor pontuação.

**Preparação do Jogo:**

O jogo começa com as peças brancas e pretas distribuídas nas casas do perímetro aleatoriamente, porém nunca poderão estar mais que duas casas da mesma cor seguidas. Na imagem em baixo está representada uma possível configuração inicial do tabuleiro.

Exemplo de um tabuleiro no estado inicial

O material necessário para o jogo é um tabuleiro quadrado 7x7 (sendo que as casas dos cantos nunca serão usadas nem terão relevância no jogo), 12 peças brancas, 12 peças pretas e, para um modo de jogo de jogo extra, não implementado por nós, duas peças pequenas azuis.

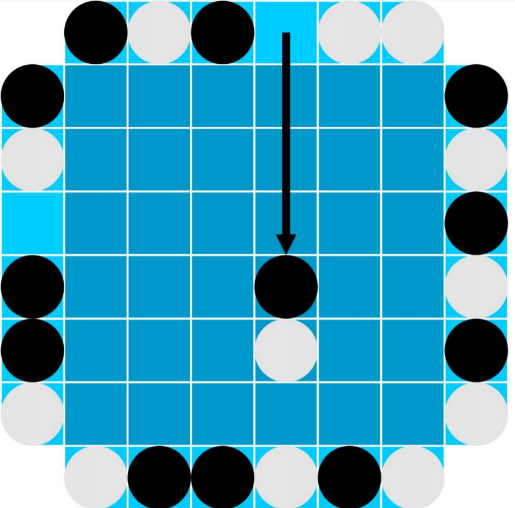
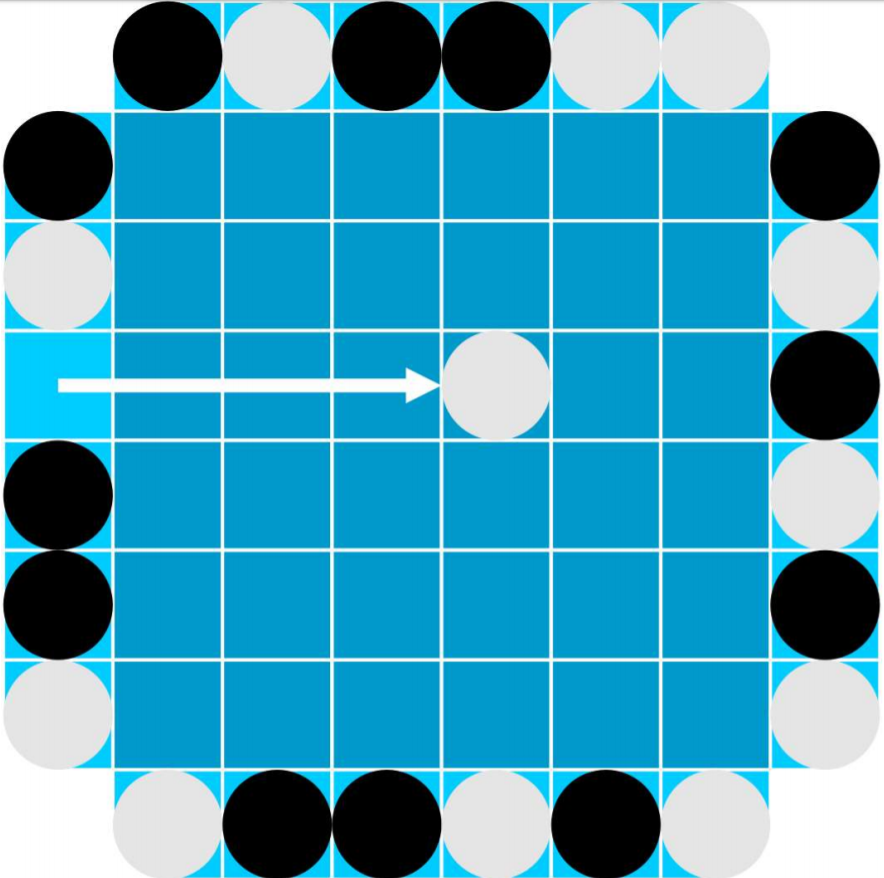
**Como jogar:**

Começando com as peças brancas, os jogadores vão alternando jogadas, arrastando os discos do perímetro para a área do tabuleiro 6x6 (azul escura na imagem). O jogador é obrigado a usar um disco, exceto quando não tem alternativa sendo que, nesse caso, é obrigado a passar a vez. Os discos apenas se podem mover na coluna ou linha em que começaram e podem se mover tantas casas sendo que, dependendo de estarem inicialmente em baixo ou cima (movimento vertical) ou à esquerda ou direita (movimento horizontal):

**Tópicos e notas importantes do jogo:**

- Um ou mais discos que estejam no caminho podem ser empurrados como resultado da jogada.

- Todos os discos no fim da jogada têm de estar dentro da área jogável (6x6), ou seja, não é possível fazer uma jogada em que sejam empurrados discos para fora da área azul escura. Isto implica que, uma vez que a casa azul clara deixe de ter uma peça, nunca mais pode voltar a ter uma peça.

Exemplo de primeira jogada do jogo

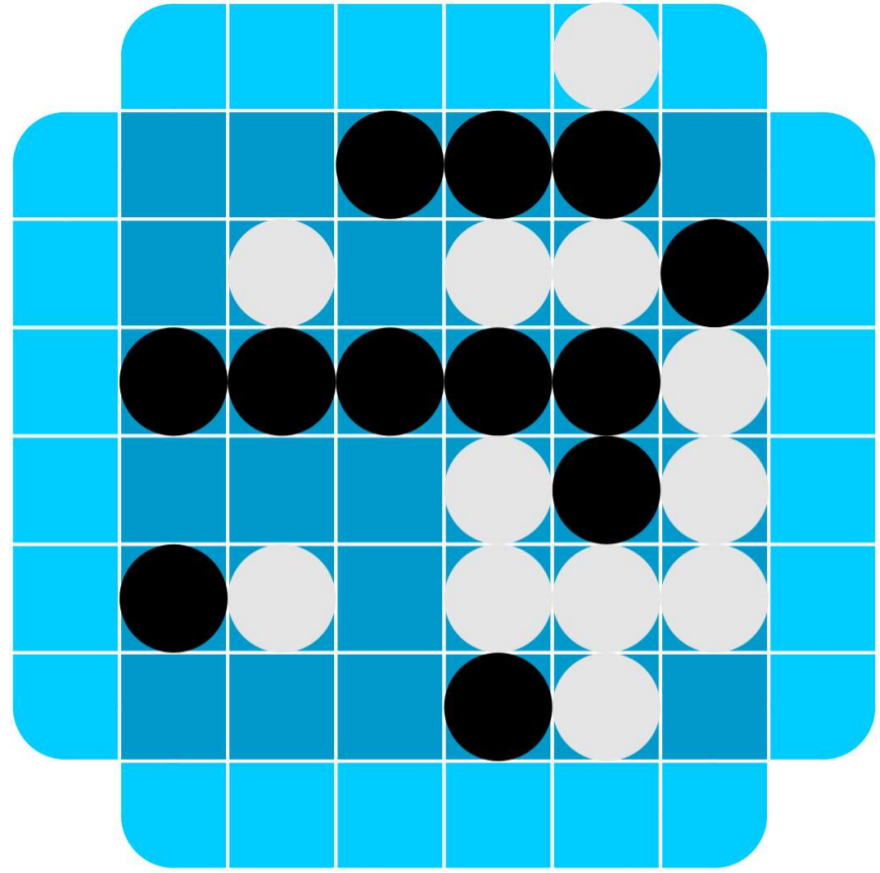
Exemplo de segunda jogada do jogo

Note-se que a peça branca jogada na primeira jogada foi empurrada pela peça preta, já que o segundo jogador decidiu empurrar a sua peça 4 casas.

**Fim do jogo:**

O jogo termina quando mais nenhum jogador já não tiver mais jogadas possíveis a fazer. O jogador que tiver mais peças conectadas ortogonalmente da sua cor ganha sendo que empates são possíveis.

Devido à rapidez do jogo, é recomendado este ser jogado a melhor de 3 ou de 5, funcionalidade que seria feita por conta própria, já que não existe nenhuma implementação nossa com um contador desse género ou modo de jogo nesses moldes.



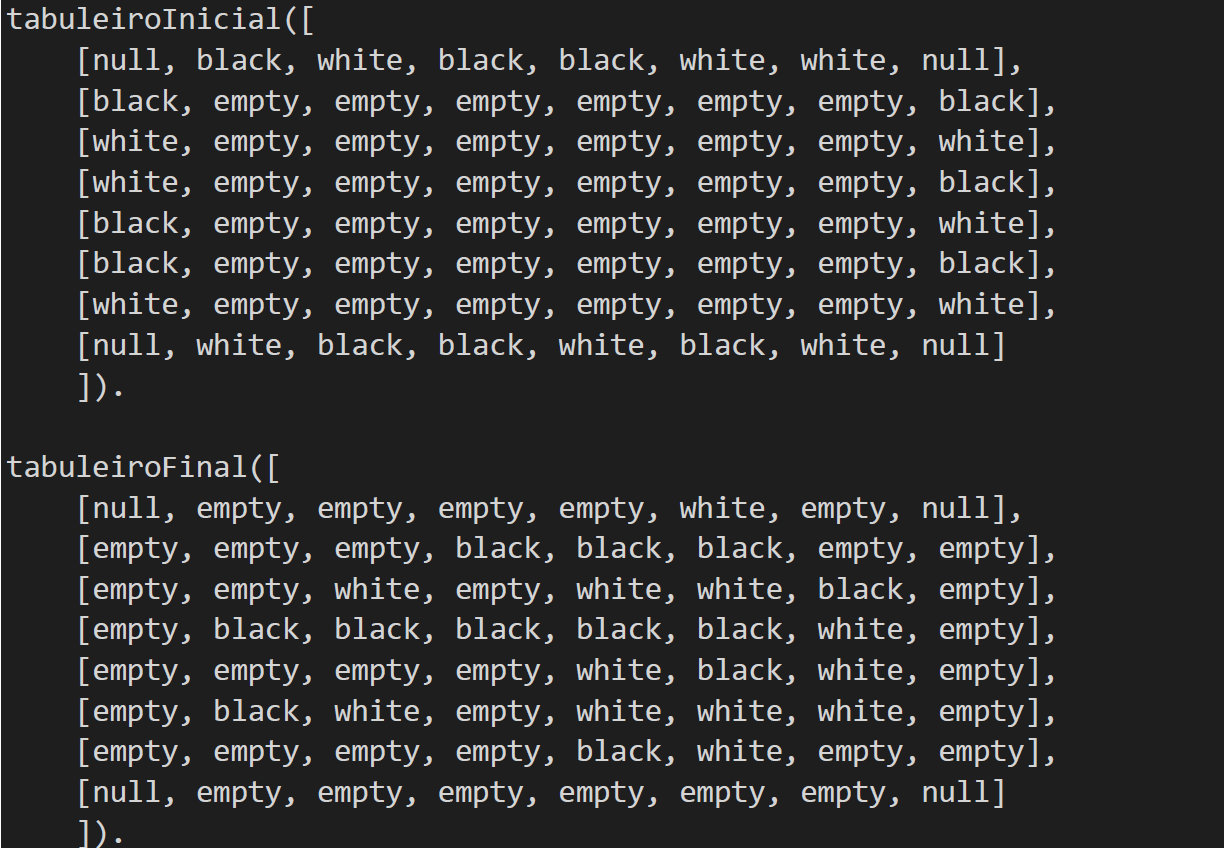
Exemplo de vitória do jogador de peças brancas (7 – 6)

Note-se que o jogo terminou mesmo com a peça branca no perímetro pois já não são possíveis mais nenhumas jogadas (jogar a peça branca implicaria empurrar a outra peça branca da extremidade para fora da área jogável)

**3. Lógica do Jogo**

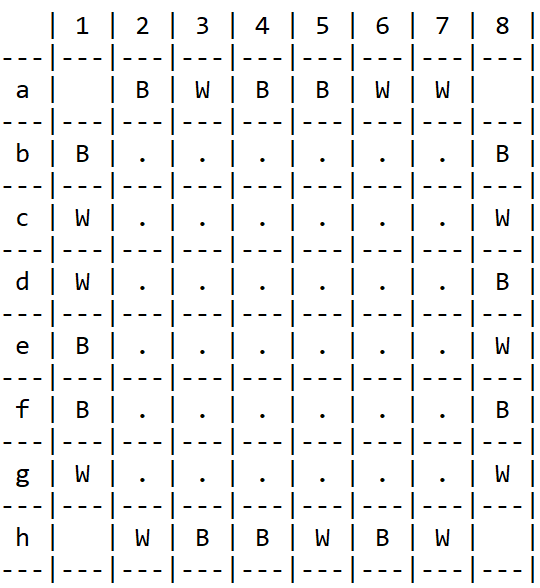
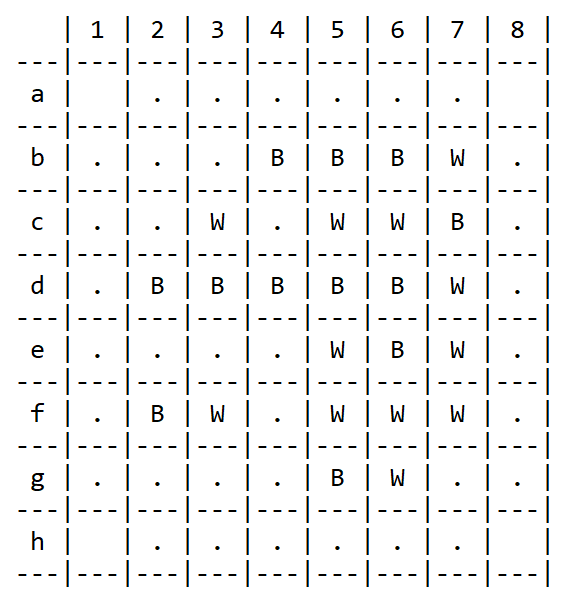
Descrever os objetivos e motivação do trabalho. Descrever num parágrafo breve a estrutura do relatório.

**3.1 Representação do Estado do Jogo**

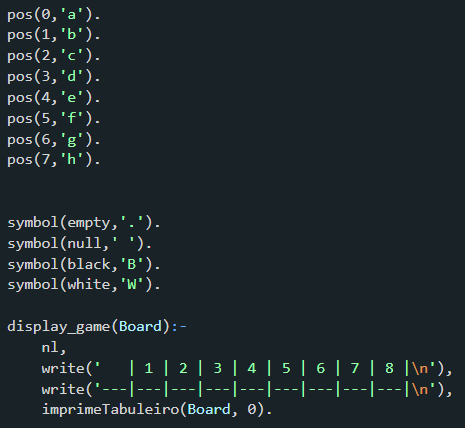
 Para armazenar a informação do jogo foi utilizada uma lista de listas com diferentes átomos (white, black, null e empty) para representarem as peças no tabuleiro ou ausência destas mesmas. Em baixo encontra-se a definição usada em código para dois exemplos de tabuleiros, inicial e final.

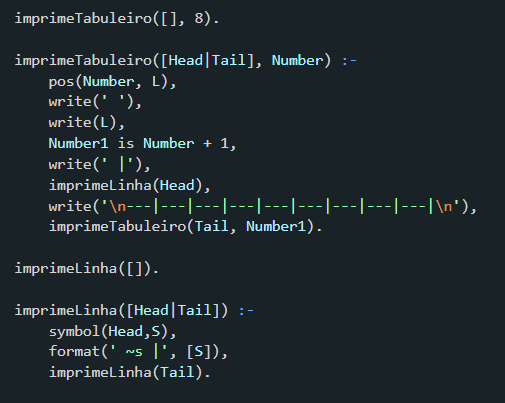
Tabuleiro Inicial

Tabuleiro Final

****

**3.2 Visualização do Tabuleiro**

Na entrega intermédia os predicados de display continham apenas tabuleiros constantes. Para esta entrega temos predicados que geram um tabuleiro aleatório no seu formato inicial, isto é, tendo 12 peças de cada cor e seguindo a regra de ter no máximo duas peças da mesma cor juntas.



As figuras acima representam o código de display de um tabuleiro (deve se lida primeiro a imagem da esquerda e depois a da direita).

No Anexo I estarão disponíveis os predicados responsáveis por gerar o tabuleiro aleatório (ficheiro display.pl), juntamente com todo o código, organizado.

**3.3 Lista de Jogadas Válidas**

Descrever os objetivos e motivação do trabalho. Descrever num parágrafo breve a estrutura do relatório.

**3.4 Execução de Jogadas**

Validação e execução de uma jogada num tabuleiro, obtendo o novo estado do jogo. Predicado move.

**3.5 Final do Jogo**

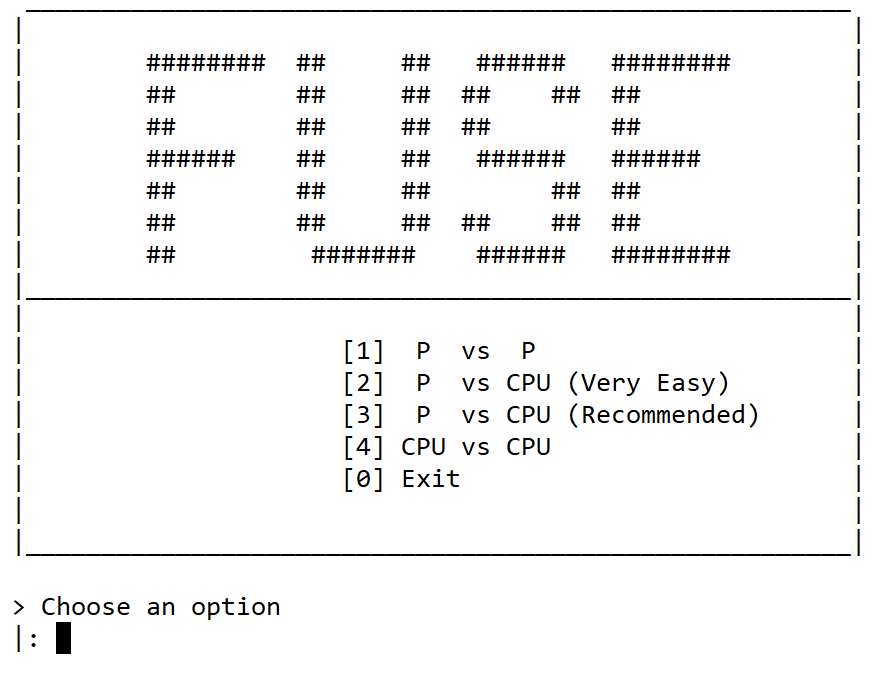
Os predicados de verificação de final do jogo (game\_over(Board)) estão separados dos predicados de determinar o vencedor. O raciocínio por trás do predicado game\_over(Board) foi dividi-lo, por assim dizer, em 4 predicados distintos: um para o topo (linha), outro para a direita (coluna), outro para baixo (linha) e outro para a esquerda (coluna). Como estes predicados são iguais entre si, explicar a lógica de um, explica a lógica dos quatro. O predicado checkGameOverTop(Board, Row, Col) recebe as 3 variáveis indicadas, sem devolver nenhuma. O objetivo dele é verificar peça a peça o conteúdo da linha (neste caso). Caso a linha esteja completamente vazia, passa para o checkGameOverRight(Board, Row, Col) que fará o mesmo, mas para a coluna da direita e assim sucessivamente. Se a linha não estiver vazia será feita a verificação da linha ou coluna (dependendo do predicado atual – top, right, bottom ou left) e em caso de não estar é porque o jogo ainda pode continuar. Se a linha estiver cheia, os predicados devem continuar a verificação até terminar o predicado checkGameOverLeft. Se o jogo estiver terminado é chamado duas vezes o predicado traverseBoard(Board, BoardV, Row, Col, Player, Score AuxScore): traverseBoard(Board, BoardV, 1, 1, black, 0, BScore) e traverseBoard(Board, BoardV, 1, 1, white, 0, WScore). Nas variáveis BScore e WScore serão retornados os pontos do jogador com peças pretas e peças brancas, respetivamente. Com ao predicado printWinner(B, W) é possível imprimir se houve um empate ou vitória das peças brancas ou vitórias das peças pretas. A ideia por trás destes predicados é usar dois tabuleiros como variáveis de entrada, um que representa o jogo (Board) e o outro com átomos diferentes (true e false), valores que representam se as casas já foram visitadas.

**3.6 Avaliação do Tabuleiro**

Verificação do fim do jogo, com identificação do vencedor. O predicado chama-se game\_over(+Board, -Winner).

**3.7 Jogada do Computador**

Verificação do fim do jogo, com identificação do vencedor. O predicado chama-se game\_over(+Board, -Winner)

****

**Interface**

A interface é curta, simples e auto explicativa. A opção 1 leva ao modo jogador contra jogador (humanos), a opção 2 ao modo jogador contra computador, a opção 3 ao modo anterior numa dificuldade mais elevada e a quarta opção computador contra computador. A opção 0 permite sair do programa. Quando o jogo termina também há uma opção de introduzir 0 para voltar ao menu principal, acompanhado de um *print* no ecrã a indicar para o fazer. É importante sublinhar que todas as *inputs* devem seguir a regra de terminar com um ponto final.

**4. Conclusões**

Conclusão.