Control-V

Relatório Final

Mestrado Integrado em Engenharia Informática e Computação

Grupo: Control-V_4:
João Carlos Carreira Martins - up201605373@fe.up.pt
João Francisco de Pinho Brandão - up201705573@fe.up.pt

17 de novembro 2019

Resumo

Este trabalho consiste na conceção de um jogo de tabuleiro utilizando uma linguagem de programação em lógica denominada Prolog. Control-V foi o jogo escolhido e é destinado a dois jogadores. Note-se que foram implementados três modos de utilização perfeitamente funcionais: Humano/Humano, Humano/Computador e Computador/Computador. Nestes três modos todas as regras do jogo foram implementadas com sucesso. Este trabalho permitiu a consolidação dos conhecimentos adquiridos nas aulas tanto teóricas como práticas da cadeira de Programação Lógica e confirmar o quão eficiente é usar a linguagem de Prolog para resolver problemas de decisão. Inicialmente, o principal obstáculo a ultrapassar foi o facto de nunca se ter tido contacto com esta linguagem, a adaptação à mesma demorou um pouco, mas à medida que o projeto foi sendo desenvolvido, as dúvidas foram esclarecidas. Assim, foi concebido com sucesso um jogo simples, intuitivo e de fácil interação com o utilizador, com três modos à escolha.

1 Introdução

Este projeto foi desenvolvido no Sistema de Desenvolvimento SICStus Prolog no âmbito da unidade curricular de Programação Lógica de 3º ano do curso Mestrado Integrado em Engenharia Informática e de Computação e tem como tema o jogo de tabuleiro Control-V. O objetivo deste trabalho foi implementar, em linguagem Prolog, um jogo de tabuleiro e de peças, pelas regras de movimentação das peças (jogadas possíveis) e pelas condições de terminação do jogo com derrota, vitória ou empate. Este relatório tem a seguinte estrutura:

- **O Jogo Control-V**: Descrição do jogo e das suas regras.
- **Lógica do Jogo**: Descrição da implementação da lógica do jogo, tendo a seguinte estrutura:
    - **Representação do Estado do Jogo:** Exemplificação de estados iniciais, intermédios e finais do jogo.
    - **Visualização do Tabuleiro:** Descrever a interface com o utilizador, descrevendo a visualização do estado do jogo e robustez da interface/validações de entrada.
    - **Lista de Jogadas Válidas:** Descrição dos predicados usadas para a validação das jogadas.
    - **Execução de Jogadas:** Explicitação da validação e execução de uma jogada num tabuleiro, obtendo o novo estado do jogo.
    - **Final do Jogo:** Descrição dos predicados que verificam o fim de jogo.
    - **Avaliação do Tabuleiro:** Descrição da avaliação do estado do jogo, que permite comparar a aplicação das diversas jogadas disponíveis.
    - **Jogada do Computador:** Explicação de como é efetuada a escolha da jogada a efetuar pelo computador, dependendo do nível de dificuldade.
- **Conclusões:** um resumo acerca deste projeto.

2 O Jogo Control-V

História

Control-V é um jogo de tabuleiro educacional de estratégia abstrata e matemática baseando-se na ocupação de território através da simetria. Este jogo foi autopublicado em 2019 pela JoyEnjoy Inc. e criado por Taewon Moon.

Regras

Pode-se jogar este jogo entre 2 a 4 pessoas, sendo a única diferença nas regras a dimensão do tabuleiro, 10x10 para 2 pessoas e 12x12 para 3 ou 4 pessoas. Neste projeto só nos interessa jogos entre 2 jogadores (humano/humano, humano/computador e computador/computador), logo utilizaremos um tabuleiro de dimensão 10 x 10. As imagens

utilizadas serão de um jogo de 4 pessoas, ou seja, de um tabuleiro de dimensão 12 x 12, por serem as únicas disponíveis online.

Inicialmente, todos os jogadores dispõem de 2 blocos de bloqueio que irão ser colocados numa posição no tabuleiro à sua escolha. A ordem de jogada é definida aleatoriamente.

Preparação

Inicialmente, o tabuleiro está vazio. O jogador 1 começa por colocar um “bloco de bloqueio” (Blocking Block) em qualquer posição. Esta peça não tem valor no resultado final do jogador, apenas ocupa uma posição do tabuleiro. De seguida, o jogador 2 coloca o seu “bloco de bloqueio” numa posição vazia do tabuleiro. Cada jogador coloca 2 “blocos de bloqueio”. Posteriormente, cada jogador coloca uma peça sua no tabuleiro também numa posição livre à sua escolha. Esta fase do jogo termina com um tabuleiro com 4 “blocos de bloqueio”, uma peça do jogador 1 e uma peça do jogador 2.

Desenvolvimento

Nesta fase os jogadores copiam e colam as suas peças no tabuleiro usando simetria, daí o nome Control-V (atalho no teclado para copiar texto ou outro conteúdo). Se 2 ou mais peças estiverem em posições adjacentes, verticalmente ou horizontalmente, estas funcionam como um bloco que deve ser copiado e colado em conjunto. Os eixos de simetria podem ser verticais ou horizontais e tem que estar posicionados entre as várias posições do tabuleiro, tal como é demonstrado na Figura 1.

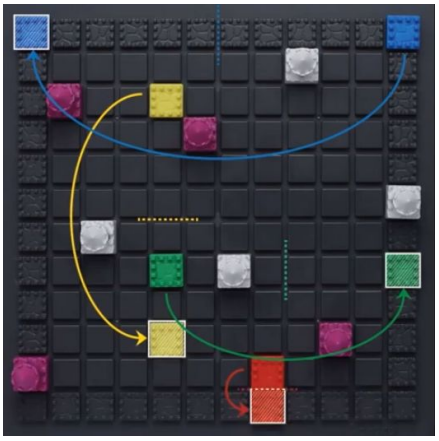


Figura 1: Demonstração de possíveis jogadas copiando e colando blocos de peças, usando eixos de simetria.

Fim

O jogo acaba assim que nenhum jogador tenha jogadas possíveis, ou seja, assim que nenhuma peça ou bloco de peças possa ser colado no tabuleiro.

O vencedor é aquele que tiver mais peças no tabuleiro.

3 Lógica do Jogo

3.1 Representação do Estado do Jogo

Estado Inicial

```
initialBoard([
[empty,empty,empty,empty,empty,empty,empty,empty,empty,empty],
[empty,empty,empty,empty,empty,empty,empty,empty,empty,empty],
[empty,empty,empty,empty,empty,empty,empty,empty,empty,empty],
[empty,empty,empty,empty,empty,empty,empty,empty,empty,empty],
[empty,empty,empty,empty,empty,empty,empty,empty,empty,empty],
[empty,empty,empty,empty,empty,empty,empty,empty,empty,empty],
[empty,empty,empty,empty,empty,empty,empty,empty,empty,empty],
[empty,empty,empty,empty,empty,empty,empty,empty,empty,empty],
[empty,empty,empty,empty,empty,empty,empty,empty,empty,empty],
[empty,empty,empty,empty,empty,empty,empty,empty,empty,empty]
]).
```

Estado Intermédio

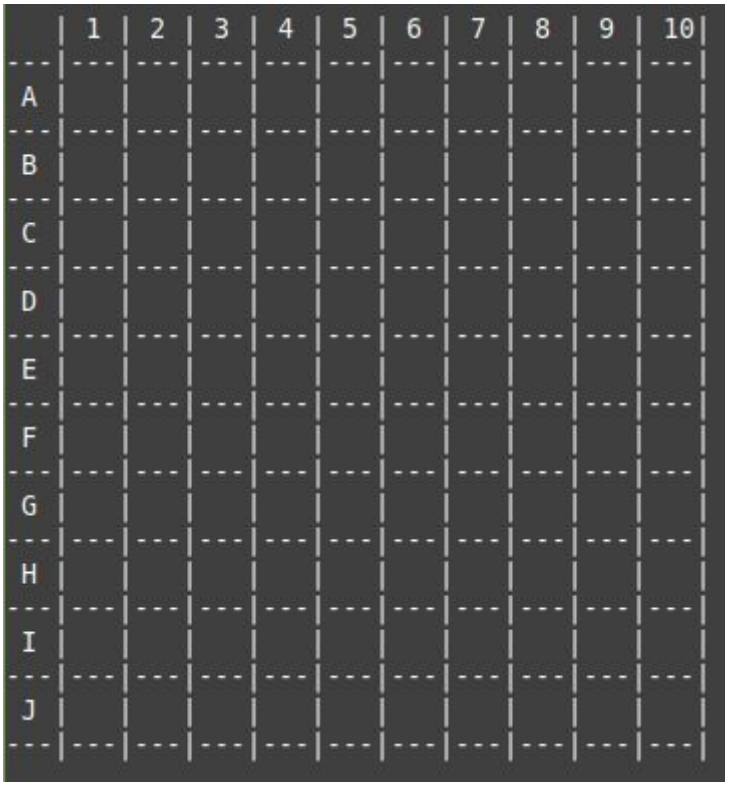
```
intermediateBoard([
[empty,empty,empty,block,empty,empty,empty,empty,empty,empty],
[empty,empty,empty,empty,empty,empty,empty,block,empty,empty],
[empty,empty,empty,empty,empty,empty,empty,empty,empty,empty],
[playerA,playerA,empty,playerA,empty,empty,empty,empty,empty,empty],
[empty,empty,empty,empty,empty,empty,empty,empty,empty,empty],
[empty,empty,empty,empty,empty,playerB,empty,empty,playerB,playerB],
[playerA,playerA,empty,block,empty,empty,empty,empty,playerB,playerB],
[empty,empty,empty,empty,empty,empty,empty,empty,empty,empty],
[empty,empty,empty,empty,empty,empty,block,empty,empty,empty],
[empty,empty,empty,empty,empty,empty,empty,empty,empty,empty]
]).
```

Estado Final

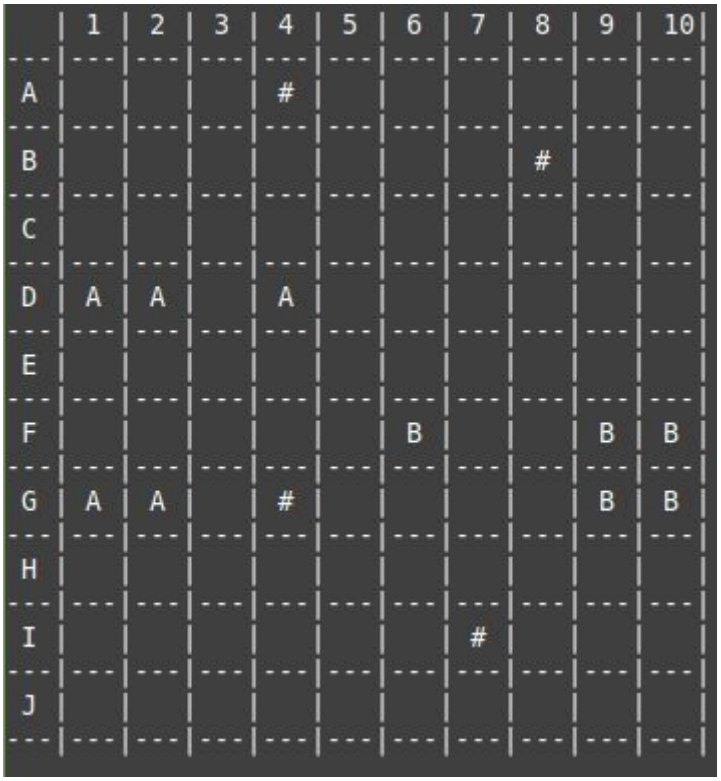
```
finalBoard([
[playerA,playerA,empty,block,empty,empty,empty,empty,empty,empty],
[playerA,playerA,empty,playerA,playerA,empty,empty,block,playerB,playerB],
[playerA,playerA,empty,playerA,playerA,empty,empty,empty,playerB,playerB],
[playerA,playerA,empty,playerA,playerA,empty,playerA,playerA,playerB,playerB],
[empty,empty,empty,empty,playerA,playerA,playerA,playerA,playerB,playerB],
[empty,empty,empty,empty,empty,playerB,playerB,empty,playerB,playerB],
[playerA,playerA,empty,block,empty,playerB,playerB,empty,playerB,playerB],
[playerA,playerA,empty,empty,empty,empty,empty,empty,playerB,playerB],
[playerA,playerA,playerB,playerB,playerB,playerB,block,empty,playerB,playerB],
[playerA,playerA,playerB,playerB,playerB,playerB,empty,empty,playerB,playerB]
]).
```

3.2 Visualização do tabuleiro

Estado Inicial

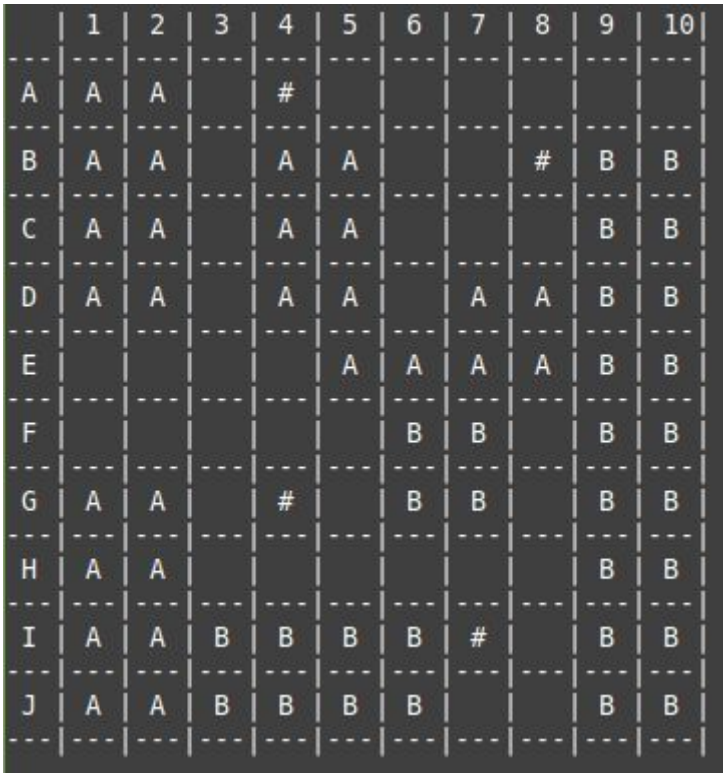


Estado Intermédio



	1	2	3	4	5	6	7	8	9	10
A				#						
B								#		
C										
D	A	A		A						
E										
F						B			B	B
G	A	A		#					B	B
H										
I							#			
J										

Estado Final



	1	2	3	4	5	6	7	8	9	10
A	A	A		#						
B	A	A		A	A			#	B	B
C	A	A		A	A				B	B
D	A	A		A	A		A	A	B	B
E					A	A	A	A	B	B
F						B	B		B	B
G	A	A		#		B	B		B	B
H	A	A							B	B
I	A	A	B	B	B	B	#		B	B
J	A	A	B	B	B	B			B	B

Elementos:

A – Peça do jogador 1
B – Peça do jogador 2
- Bloco de Bloqueio

Segue-se o código que será utilizado para mostrar o tabuleiro na consola:

```
initialBoard([
[empty,empty,empty,empty,empty,empty,empty,empty,empty,empty],
[empty,empty,empty,empty,empty,empty,empty,empty,empty,empty],
[empty,empty,empty,empty,empty,empty,empty,empty,empty,empty],
[empty,empty,empty,empty,empty,empty,empty,empty,empty,empty],
[empty,empty,empty,empty,empty,empty,empty,empty,empty,empty],
[empty,empty,empty,empty,empty,empty,empty,empty,empty,empty],
[empty,empty,empty,empty,empty,empty,empty,empty,empty,empty],
[empty,empty,empty,empty,empty,empty,empty,empty,empty,empty],
[empty,empty,empty,empty,empty,empty,empty,empty,empty,empty],
[empty,empty,empty,empty,empty,empty,empty,empty,empty,empty]
]).

symbol(empty,S) :- S=' '.
symbol(player1,S) :- S='A'.
symbol(player2,S) :- S='B'.
symbol(block,S) :- S='#'.

letter(1, L) :- L='A'.
letter(2, L) :- L='B'.
letter(3, L) :- L='C'.
letter(4, L) :- L='D'.
letter(5, L) :- L='E'.
letter(6, L) :- L='F'.
letter(7, L) :- L='G'.
letter(8, L) :- L='H'.
letter(9, L) :- L='I'.
letter(10, L) :- L='J'.

printBoard(Board) :-
   nl,
   write('   | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10|\n'),
   write('---|---|---|---|---|---|---|---|---|---|---|\n'),
   printMatrix(Board, 1).
```

```
printMatrix([], 11).

printMatrix([Head|Tail], N) :-
   letter(N, L),
   write(' '),
   write(L),
   N1 is N + 1,
   write(' | '),
   printLine(Head),
   write('\n---|---|---|---|---|---|---|---|---|---|---|\n'),
   printMatrix(Tail, N1).

printLine([]).

printLine([Head|Tail]) :-
   symbol(Head, S),
   write(S),
   write(' | '),
   printLine(Tail).
```

3.3 Lista de Jogadas Válidas

Uma jogada é considerada válida se a posição (ou posições) a ocupar estiverem vazias e forem simétricas, horizontalmente ou verticalmente, a uma peça ou bloco de peças.

Para a validação das jogadas os principais predicados utilizados que se encontram no ficheiro logic.pl foram:

```
      valid_moves(Board, Player, ListOfMoves)
      checkSurroundings(Player, Line, Column, Board, FinalBoard,
ListOfBlocks)
      getPiecesBlocks(Board, Player, Line, Column, ListOfBlocks)
      getValidMovesUp(Board, Block, N, LineMin, ListOfMoves)
      getValidMovesDown(Board, Block, N, LineMax, ListOfMoves)
      getValidMovesLeft(Board, Block, N, ColumnMin, ListOfMoves)
      getValidMovesRight(Board, Block, N, ColumnMax, ListOfMoves)
      getValidMoves(Board, Player, ListOfBlocks, ListOfMoves)
```

valid_moves(Board, Player, ListOfMoves) - Obtém uma lista, **ListOfMoves**, de todas as jogadas válidas do jogador **Player**, recorrendo ao predicado **getPiecesBlocks(Board, Player, Line, Column, ListOfBlocks)** e **getValidMoves(Board, Player, ListOfBlocks, ListOfMoves)**.

checkSurroundings(Player, Line, Column, Board, FinalBoard, ListOfBlocks) - Obtém um bloco de peças, **ListOfBlocks**, do jogador **Player**. Para isso, percorre o tabuleiro **Board** e verifica se a peça é do jogador **Player**. Caso seja, cria um novo tabuleiro **FinalBoard** semelhante ao tabuleiro **Board** só que com a posição onde estava a peça que encontrou vazia. Este passo é feito para que a mesma peça não seja verificada mais do que uma vez. Posteriormente, este predicado é invocado mais 4 vezes, uma para cada sentido (Cima, Baixo, Esquerda, Direita), usando recursividade. A função termina quando o tabuleiro é totalmente analisado.

getPiecesBlocks(Board, Player, Line, Column, ListOfBlocks) - Obtém uma lista, **ListOfBlocks**, de todos os blocos de peças no tabuleiro **Board** do jogador **Player**. Os argumentos **Line** e **Column** passados são ambos 1, para percorrer o tabuleiro **Board** a partir da posição [1,1] (posição do topo esquerdo do tabuleiro). Para isto, usa o predicado **checkSurroundings(Player, Line, Column, Board, FinalBoard, ListOfBlocks)**.

getValidMovesUp(Board, Block, N, LineMin, ListOfMoves) - Obtém uma lista, **ListOfMoves**, de todas as jogadas válidas do bloco de peças **Block** acima deste.

getValidMovesDown(Board, Block, N, LineMax, ListOfMoves) - Obtém uma lista, **ListOfMoves**, de todas as jogadas válidas do bloco de peças **Block** abaixo deste.

getValidMovesLeft(Board, Block, N, ColumnMin, ListOfMoves) - Obtém uma lista, **ListOfMoves**, de todas as jogadas válidas do bloco de peças **Block** à esquerda deste.

getValidMovesRight(Board, Block, N, ColumnMax, ListOfMoves) - Obtém uma lista, **ListOfMoves**, de todas as jogadas válidas do bloco de peças **Block** à direita deste.

getValidMoves(Board, Player, ListOfBlocks, ListOfMoves) - Obtém uma lista de todas as jogadas válidas, **ListOfMoves**, do jogador **Player**. Esta lista tem o seguinte formato:

```
[ [ Bloco de peças , [Todas as jogadas válidas para o respetivo bloco
de peças] ],
   ... ]
Exemplo:
[
   [ [[1,5]] , [ [[2,5]],[[4,5]],[[6,5]],[[8,5]],[[10,5]],[[1,10]] ] ],
   [ [[1,7]] , [ [[2,7]],[[4,7]],[[6,7]],[[8,7]],[[10,7]],[[1,10]] ] ]
   ...
]
```

Esta lista é obtido invocando os predicados **getValidMovesUp(Board, Block, N, LineMin, ListOfMoves)**, **getValidMovesDown(Board, Block, N, LineMax, ListOfMoves)**, **getValidMovesLeft(Board, Block, N, ColumnMin, ListOfMoves)** e **getValidMovesRight(Board, Block, N, ColumnMax, ListOfMoves)** e usando recursividade para analisar todos os blocos da lista **ListOfBlocks**.

3.4 Execução de Jogadas

Uma vez escolhida uma jogada da lista de jogadas válidas, essa jogada é concretizada usando os seguintes predicados que se encontram no ficheiro logic.pl:

```
move(Move, Board, NewBoard)
makeMove(Moves, Player, Board, NewBoard)
```

move(Move, Board, NewBoard) - **Move** é uma lista com o formato [**Player**, **Moves**], em que **Player** é o jogador que realiza a jogada e **Moves** é uma lista das posições onde vai ser inserida a peça ou o bloco de peças do jogador **Player**. Este predicado chama **makeMove(Moves, Player, Board, NewBoard)**.

makeMove(Moves, Player, Board, NewBoard) - Cria um novo tabuleiro **NewBoard**, igual ao tabuleiro **Board** mas com as peças do jogador **Player** de posições **Moves** inseridas.

3.5 Final do Jogo

A cada ciclo de jogo é verificado as jogadas válidas de ambos os jogadores e assim que nenhum tenha jogadas válidas possíveis o ciclo das jogadas termina e é chamado o seguinte predicado que se encontra no ficheiro logic.pl:

```
game_over(Board, Winner)
```

Outros predicados usados:

```
value(Board, Player, Value)
```

game_over(Board, Winner) - Calcula a pontuação de cada jogador no tabuleiro **Board** usando o predicado **value(Board, Player, Value)** e compara-as para saber quem ganhou o jogo ou se houve empate. O valor de **Winner** depende desta comparação: “player1” caso a pontuação do jogador 1 seja superior à pontuação do jogador 2, “player2” caso contrário e “tie” em caso de igualdade.

value(Board, Player, Value) - Calcula a pontuação do jogador **Player** percorrendo o tabuleiro **Board** e contando o número de peças do jogador. **Value** é onde fica guardada essa informação.

3.6 Avaliação do Tabuleiro

A avaliação do estado do jogo é feita através da contagem do número de peças de cada jogador no tabuleiro. Assim o "score" de cada jogador será o seu número de peças. O predicado implementado é **value(Board, Player, Value)**, que está explicado na secção 3.5 acima.

3.7 Jogada do Computador

No início de um jogo, os predicados **startGame/4** recebe 2 átomos, **Player1Mode** e **Player2Mode**, que por sua vez passa-os ao predicado **gameLoop/6** e este passa-os ao predicado **play/4**. Estes 2 átomos contém informação acerca do modo do jogador: 'P' para humano, 'C0' para computador de nível 0 e 'C1' para computador de nível 1. O predicado **play/4**, responsável por realizar cada jogada, verifica este modo e caso seja computador ('C0' ou 'C1') chama o seguinte predicado:

```
choose_move(Board, PlayerLevel, PlayerMove)
```

choose_move(Board, PlayerLevel, PlayerMove) - Escolhe uma jogada para o computador. Esta jogada depende do seu nível: nível 0 escolhe uma jogada aleatória e nível 1 verifica qual é o maior bloco de peças no tabuleiro e escolhe uma jogada válida aleatória para esse bloco de peças.

4 Conclusões

Ao longo do desenvolvimento deste projeto foram encontradas algumas dificuldades, nomeadamente o facto de ter sido a primeira vez que realizamos um projeto em linguagem Prolog, o pensamento recursivo e a abordagem a problemas grandes e complexos. Todos estes problemas foram eventualmente superados.

Todos os objetivos foram cumpridos. No entanto, a eficiência dos métodos concebidos podia ser melhorada.

Em suma, o trabalho foi concluído com sucesso, e o seu desenvolvimento contribuiu positivamente para uma melhor compreensão da linguagem Prolog e um melhor pensamento recursivo.

Bibliografia

https://www.boardgamegeek.com/image/4832083/control-v
https://www.youtube.com/watch?v=t615ediK8EY
https://www.youtube.com/watch?v=KRzdyQAOkUE
https://www.swi-prolog.org/