cats & dogs

Universidade do Porto

Faculdade de Engenharia

FEUP

PLOG

TP1 - Relatório Final

Novembro 2008

João Cristóvão Xavier – 060509116

João Pedro Ribeiro – 060509019

Resumo

Este trabalho foi desenvolvido como primeiro trabalho prático da cadeira PLOG (Programação em Lógica), do 3º ano do Mestrado Integrado em Engenharia Informática e Computação.

O seu objectivo foi recriar o jogo "cats & dogs" utilizando a linguagem *Prolog*. O jogo é para dois jogadores, que podem ser dois utilizadores, dois jogadores controlados pelo computador, ou um de cada; há ainda a possibilidade de escolher entre três níveis de dificuldade dos jogadores do computador.

Índice

1 - Introdução

A escolha do jogo "cats & dogs" deveu-se principalmente à sua simplicidade e facilidade de começar a jogar, sem que o jogador precise de aprender regras complexas para isso. Ainda assim é interessante jogar algumas vezes para se compreender melhor a mecânica do jogo e tentar optimizar as jogadas efectuadas; um jogador mais experiente tem bastante mais probabilidades de ganhar do que um jogador principiante.

Por outro lado, para diferenciar ligeiramente esta versão da original e de modo a aumentar a complexidade do problema, foram acrescentadas algumas variações. As dimensões do tabuleiro são configuráveis e, ao iniciar o jogo, algumas casas (escolhidas aleatoriamente) possuem obstáculos, não podendo nenhum jogador colocar peças nelas. Foram também criadas jogadas especiais, diferentes para *cats* e *dogs*, que podem mudar o rumo do jogo se forem bem utilizadas.

2 - Descrição do Problema

O jogo “cats & dogs” é bastante antigo, tanto que é bastante difícil de dizer quando foi criado ou quem o fez. No entanto, é provável que tenha tido bastante influência de jogos como o “Jogo do Galo” e semelhantes.

Objectivo do Jogo

Colocar mais peças no tabuleiro que o oponente e acabar com as possibilidades de jogada deste.

Apresentação do Jogo

Tabuleiro quadrado, lado de dimensão variável e configurável no início do jogo, com valor entre 4 e 12 (5 por 5 nos exemplos); o tabuleiro começa com um número aleatório de obstáculos (20% de probabilidade de uma dada casa ter um obstáculo).

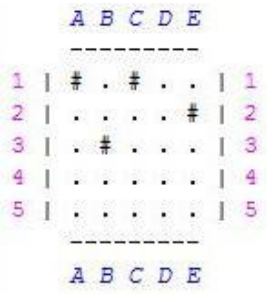


Fig. 1 - tabuleiro inicial

Desenvolvimento de uma Partida / Regras

Cada jogador com um tipo de peça: “cão” (X) ou “gato” (O).

Jogadas alternadas entre os adversários.

Peças de espécies diferentes nunca podem ser dispostas ortogonalmente adjacentes (i.e., um jogador não pode colocar uma peça numa casa se por cima, por baixo, ao lado esquerdo e/ou ao lado direito desta já se encontrar uma peça do adversário), nem nas casas com obstáculos.

Por cada *lado-do-tabuleiro* jogadas, os jogadores têm direito à sua jogada especial: o jogador com peças *Dogs* pode colocar uma peça adicional, o jogador com peças *Cats* pode remover uma peça do adversário.

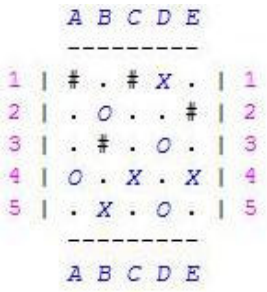
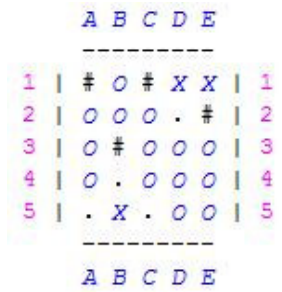


Fig. 2 – tabuleiro intermédio

Fim da Partida

A partida termina assim que ambos os jogadores estejam impossibilitados de colocar mais peças no tabuleiro.

Ganha o jogador que tiver colocado mais peças.



Duração de uma Partida

Cerca de 10 minutos.

3 - Arquitectura do Sistema

O projecto encontra-se dividido em diversos módulos, os quais agrupam predicados relacionados entre si:

- *Gameplay* – início do jogo e função principal da execução do programa;
- *Start-Up* – configuração do jogo em função das escolhas do utilizador;
- *Move Parser* – conversão de *inputs* para coordenadas do tabuleiro;
- *Board Generation* – criação do tabuleiro de jogo;
- *Visualization* – visualização do tabuleiro na consola, com coordenadas;
- *Move Computation* – cálculo de jogadas do computador, lista de possíveis jogadas;
- *Assignment of a Valid Move* – execução de jogadas;
- *Special Moves* – funções relativas às jogadas especiais;
- *Piece Count* – contagem das peças dos jogadores presentes no tabuleiro;
- *List Utilities* – funções de manipulação de listas;
- *Sockets* – comunicação com o módulo de visualização.

4 - Módulo de Lógica do Jogo

(nota: código integral no Anexo A)

4.1 - Geração e Visualização do Estado do Jogo

```
generateBoard(+Size, -Board)
drawBoard(+Board, +Size)
```

A partir do valor para *Size* inserido pelo jogador ao iniciar o jogo, o tabuleiro é gerado como uma lista de listas. A criação de novas células chama uma função de criação aleatória de obstáculos (tendo cada casa uma probabilidade de 20% de possuir um), os quais ficam representados com o carácter '3'; casas vazias ficam com o carácter '0'. Havendo peças dos jogadores colocadas, são guardadas nas posições respectivas dentro das listas como caracteres '1' e '2', para cada um deles.

Ao ser visualizado, e de modo a facilitar a leitura das coordenadas das células, são também mostrados números dos lados do tabuleiro e letras por cima e por baixo. Os caracteres representativos do estado das células são substituidos por: " ." - espaço vazio; " O" e " X" - peça dos jogadores; " #" – obstáculo.

4.2 - Validação de Jogadas

```
validMove(+Board, +Player, +Move, +CurrentPiece)
```

A função de validação de jogadas faz uma simples verificação de possibilidade de jogada, de acordo com as regras do jogo: a casa escolhida por um jogador para colocar a peça seguinte deve pertencer ao tabuleiro, não pode estar ocupada, e não pode ser ortogonalmente adjacente a uma peça do outro jogador.

4.3 - Execução de Jogadas

```
assignMove(+OldPiece, +NewPiece, +Move, +Board, -NewBoard)
specialMove(+Board,  +Player,  +Size,  +NextTurn,  +MoveList,
            +MoveCount)
```

Depois de verificada a correcção da jogada, esta função percorre as posições do tabuleiro, linha a linha e depois coluna a coluna, até encontrar a casa escolhida, inserindo aí o carácter da peça correspondente.

As jogadas especiais ocorrem a cada *Size* (medida do lado do tabuleiro) jogadas de cada jogador, e são obrigatoriamente executadas de imediato. São tratadas de modo semelhante às jogadas regulares, especialmente a dos *Dogs*, que corresponde a duas jogadas das normais; no

caso dos *Cats*, dado implicar a remoção de uma peça adversária, implica determinar a lista de células que correspondem à posição de peças do outro jogador.

4.4 - Lista de Jogadas Válidas e Avaliação do Tabuleiro

```
moveList(+Board, +Player, -MoveList, +CurrentPiece)
```

O predicado moveList tem dupla funcionalidade: não só coloca numa lista todas as jogadas possíveis de dado jogador, como também atribui uma chave (avaliação) a cada jogada.

Essa avaliação é correspondente ao número de jogadas com que o oponente vai ficar se se fizer esse movimento. É pela ordenação desta lista que posteriormente se irá determinar a jogada do computador.

4.5 - Final do Jogo

```
moveCount(+Board, +Player, -Count, +CurrentPiece)
countPieces(+Board, +Player, -Count)
endGame(+Board, +Size)
```

O jogo termina quando já não há mais casas vazias ou com possibilidade de ser utilizadas por nenhum dos jogadores, o que é determinado pela função *moveCount*. Nesse caso é chamada a função *endGame*, que finaliza a execução mostrando o resultado da partida e a pontuação de cada jogador (calculadas pela função *countPieces*).

4.6 - Cálculo da Jogada do Computador

```
computeMove(+Level, +MoveList, -Move)
```

A partir da lista gerada pela função *moveList*, e portanto já ordenada da melhor jogada possível para a pior, a acção do computador é determinada pelo seu grau de dificuldade: no nível mais alto, o movimento efectuado é o primeiro da lista, o melhor possível; no nível mais baixo, a lista é invertida, e realizada a pior das jogadas; no nível intermédio é realizada, aleatoriamente e com igual probabilidade, uma dessas duas jogadas.

4.7 - Recepção de mensagem do visualizador

```
readNumber(-Num)
readCharCode(-CharCode)
```

Os dados introduzidos pelo utilizador do programa são recebidos pelas duas funções aqui indicadas, para o caso de serem números ou letras. O seu funcionamento é semelhante, é lida toda a informação inserida, agrupada em listas de palavras, e feita a verificação de correcção, relativamente aos dados que eram esperados pelo programa (como tipo e dimensão do *input*).

5 - Interface com o Utilizador

A comunicação com o jogador não está ainda terminada, estando previsto que esta seja complementada com o terceiro trabalho prático da cadeira Laboratórios de Software e, assim, criada uma interface gráfica 3D, bastante mais interessante.

Na presente fase de desenvolvimento, toda a comunicação com o utilizador é feita através da consola. São geradas mensagens simples e de rápida leitura, pelo que o facto de se utilizar a consola não compromete a usabilidade do programa.

Começando a execução, é mostrado o título do trabalho e os nomes dos autores (consultar Anexo B – imagens 1 e 2). Logo de seguida deve-se escolher, para cada um dos jogadores, se será do tipo humano ou computador e, neste caso, qual a sua dificuldade. É então pedido ao utilizador que indique o tamanho do tabuleiro, sendo este visualizado logo de seguida. A partir daqui, e havendo algum jogador humano, a cada um dos seus turnos é-lhe pedido que indique a célula na qual deseja jogar; o computador joga instantaneamente. O mesmo acontece com as jogadas especiais (Anexo B – imagem 3; 2 jogadores do computador no exemplo).

Os dados fornecidos pelo utilizador são sempre verificados, e caso não estejam correctos, o programa não avança até que seja introduzida informação correcta (Anexo B - imagens 4, 5 e 6).

Quando um dos jogadores fica sem possibilidades de jogar é mostrada uma mensagem, passando para o turno do outro jogador em seguida (Anexo B – imagem 7) e, quando ambos os jogadores já não tem mais jogadas possíveis, o jogo é terminado com uma indicação do vencedor, e a pontuação de ambos (Anexo B – imagem 8).

6 - Conclusões e Perspectivas de Desenvolvimento

Os objectivos propostos para o trabalho foram atingidos plenamente. Durante o seu desenvolvimento foram adquiridos os conhecimentos necessários para a recriação do jogo *Cats&Dogs*.

Foram criados mecanismos de construção de tabuleiro com dimensões configuráveis e obstáculos aleatórios, implementadas funções para que dois utilizadores pudessem fazer uma partida, e também desenvolvidos jogadores controlados pelo computador, com 3 níveis de dificuldade distintos.

Apesar de estar terminado, espera-se que venha a ser possível a criação de uma interface 3D, complementando-se assim este trabalho com outro da cadeira de LASO.

7 - Bibliografia

Para a realização do trabalho:

[1] Eugénio Oliveira e Luís Paulo Reis, Materiais da Disciplina de Programação em Lógica, disponível *online* a partir de http://www.fe.up.pt/~eol/LP/0809

[2] Amzi! inc. , Adventure in Prolg, disponível online a partir de
http://www.amzi.com/AdventureInProlog/advfrtop.htm

[3] BAP (Brain Aid Prolog) Online Manuals, disponível online a partir de
http://www.fraberde/bap/bap72.html

[4] SWI-Prolog Mailing List, "Re: [SWIPL] About keysort/2 and msort/2...", disponível on-line a partir de http://gollem.science.uva.nl/SWI-Prolog/mailinglist/archive/old/2966.html

Para a realização do relatório:

[5] Games of Soldiers – CATS & DOGS -
http://homepages.di.fc.ul.pt/~jpn/gv/catdogs.htm

8 - Anexos

4.8 - Anexo A - Código

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%% CATS & DOGS  %%%%%%
%%%%%%    PROLOG    %%%%%%
%%%%%%  2008.11.02  %%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%
% João Xavier  - ei06116 %
% João Ribeiro - ei06019 %
%%%%%%%%%%%%%%%%%%%%%%%%%%


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%      BRIEF      DESCRIPTION      AND      RULES
%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%
%    The    concept    of    Cats    &    Dogs    is    quite    simple.
%
% On each turn, each player drops one of his pets into an empty cell.
%
% However, the cell where the new animal is dropped cannot be orthogonally
%
% adjacent to a different animal (i.e, no dogs and cats close to each oth-
er). %
% Wins the player that after there is no more valid moves, has more ani-
mals   %
%                               on                               board.
%
%
%
%                  abridged and adapted from "Games of Soldiers: CATS &
DOGS" %
%
http://homepages.di.fc.ul.pt/~jpn/gv/catdogs.htm %
%
%
%                                                              Features:
%
% - Special moves (once in a few turns, cats can take a dog off the board,
%
%    while    dogs    can    put    two    pieces    in    one    turn);
%
% - Randomized, dynamic obstacle board: instead of the typical clean 8x8
%
% board, this version features obstacles on the board and asks the user to
%
%             input              the              board              size;
%
%   -   3   different   levels   of   AI:   can   you   beat   them   all?
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%
```

```
% :-use_module(library(sockets)).



%%%%%%%%%%%%%%%%%%
%%%% GAMEPLAY %%%%
%%%%%%%%%%%%%%%%%%

%begin()
begin:-
	abolish(player/3),
	introduction,
	chooseGameType,
	chooseBoardSize(Size),
	generateBoard(Size, Board),
	Player is random(2)+1,
	play(Board, Player, Size, 1).

%play(+Board, +Player, +Size, +Turn)
play(Board, Player, Size, Turn):-
	moveList(Board, Player, MoveList, 0),
	moveCount(Board, Player, Count, 0),
	NextTurn is Turn+1,
	SpecialMove is Turn mod (Size*2),
	drawBoard(Board, Size),
	nl, writePlayer(Player), write('´ turn:'), nl,
	((SpecialMove =:= (Size*2-1) ; SpecialMove =:= 0) ->
		specialMove(Board, Player, Size, NextTurn, MoveList, Count),
!;
		changePlayer(Player, NextPlayer),
		(Count =:= 0 ->
			moveCount(Board, NextPlayer, Count2, 0),
			write('No moves available!'), nl,
			(Count2 =:= 0 ->
				endGame(Board, Size), !;
				play(Board, NextPlayer, Size, NextTurn));
			player(Player, PlayerType, AI),
			getMove(PlayerType, MoveList, Move, Size, AI),
			validMove(Board, Player, Move, 0),
			assignMove(0, Player, Move, Board, NewBoard),
			play(NewBoard, NextPlayer, Size, NextTurn))).

%changePlayer(+P1, -P2)
changePlayer(0, 1).
changePlayer(1, 2).
changePlayer(2, 1).

%writePlayer(+Player)
writePlayer(1):- write('Cats').
writePlayer(2):- write('Dogs').

%endGame(+Board, +Size)
endGame(Board, Size):-
	drawBoard(Board, Size),
	countPieces(Board, 1, Count1), countPieces(Board, 2, Count2),
	write(Count1), write(' - '), write(Count2), nl,
	(Count1 > Count2, write('Cats win!');
	Count1 < Count2,  write('Dogs win !');
	write('Draw!')).
```

```
%%%%%%%%%%%%%%%%%%
%%%% START-UP %%%%
%%%%%%%%%%%%%%%%%%

%introduction()
introduction:-
       nl, write('Cats & Dogs - Prolog - 2008.11.02'),
       nl, write('Joao Xavier'),
       nl, write('Joao Ribeiro'), nl, nl.

%chooseGameType()
chooseGameType:-
       write('Input game mode:'), nl,
       write('Cats: '), nl, write('1 - Human, 2 - CPU'), nl,
       repeat, readNumber(Cats), Cats>=1, Cats=<2, nl,
       write('Dogs: '), nl, write('1 - Human, 2 - CPU'), nl,
       repeat, readNumber(Dogs), Dogs>=1, Dogs=<2, nl,
       gameType(Cats, Dogs, P1, P2),
       selectAI(Cats, Dogs, AI1, AI2),
       assert(player(1, P1, AI1)), assert(player(2, P2, AI2)).

%gameType(+Cats, +Dogs, -P1, -P2)
gameType(1, 1, human, human).
gameType(1, 2, human, comp ).
gameType(2, 1, comp,  human).
gameType(2, 2, comp,  comp ).

%selectAI(+Cats, +Dogs, -AI1, -AI2)
selectAI(1, 1, 0, 0).
selectAI(1, 2, 0, AI):-
       write('Input CPU (Dogs) Level:'), nl, write('1 - Easy, 2 - Medium,
3 - Hard'), nl,
       repeat, readNumber(AI), AI>=1, AI=<3, nl.
selectAI(2, 1, AI, 0):-
       write('Input CPU (Cats) Level:'), nl, write('1 - Easy, 2 - Medium,
3 - Hard'), nl,
       repeat, readNumber(AI), AI>=1, AI=<3, nl.
selectAI(2, 2, AI1, AI2):-
       write('Input CPU (Cats) Level:'), nl,
       write('1 - Easy, 2 - Medium, 3 - Hard'), nl,
       repeat, readNumber(AI1), AI1>=1, AI1=<3, nl,
       write('Input CPU (Dogs) Level:'), nl,
       write('1 - Easy, 2 - Medium, 3 - Hard'), nl,
       repeat, readNumber(AI2), AI2>=1, AI2=<3, nl.

%chooseBoardSize(-Size)
chooseBoardSize(Size):-
       write('Choose a board size between 4 and 12:'), nl,
       repeat, readNumber(Size), Size>=4, Size=<12, nl.



%%%%%%%%%%%%%%%%%%%%%%
%%%% MOVE PARSER %%%%
%%%%%%%%%%%%%%%%%%%%%%

%getMove(+PlayerType, +MoveList, -Move, +Size, +AI)
getMove(human, _, Move, Size, _):- inputMove(Move, Size).
getMove(comp, MoveList, Move, _, AI):- computeMove(AI, MoveList, Move).

%readNumber(-Num)
readNumber(Num):-
```

```
        readln(Line),
        lengthList(Line, ListLen),
        ListLen =:= 1,
        Line = [Num|_],
        integer(Num).

%readCharCode(-CharCode)
readCharCode(CharCode):-
        readln(Line),
        lengthList(Line, ListLen),
        ListLen =:= 1,
        Line = [String|_],
        atom(String),
        atom_codes(String, CharCodeList),
        lengthList(CharCodeList, StrLen),
        StrLen =:= 1,
        CharCodeList = [CharCode|_].

%inputMove(-Move, +Size)
inputMove(Move, Size):-
        repeat,
        write('X: '), readCharCode(SX), convertAlpha(SX,X,Size),
        write('Y: '), readNumber(Y), Y>=1, Y=<Size,
        cons(X, Y, Move).

%convertAlpha(+Let, -Val, +Size)
convertAlpha(Let, Val, Size):- isUpper(Let, Size), Val is Let-64.
convertAlpha(Let, Val, Size):- isLower(Let, Size), Val is Let-96.

%isUpper(+Let, +Size)
isUpper(Let, Size):- Let>=65, UpperLimit is 65+Size, Let<UpperLimit.
%isLower(+Let, +Size)
isLower(Let, Size):- Let>=97, UpperLimit is 97+Size, Let<UpperLimit.



%%%%%%%%%%%%%%%%%%%%%%%%%
%%%% BOARD GENERATION %%%%
%%%%%%%%%%%%%%%%%%%%%%%%%

%generateBoard(+Size, -Board)
generateBoard(Size, Board):-
        genListLists(0, Size, [], Board),!.

%genListLists(+N, +Size, +OldBoard, -Board)
genListLists(Size, Size, Board, Board).
genListLists(N, Size, OldBoard, Board):-
        genList(0, Size, [], List),
        appendList(OldBoard, [List], NewBoard),
        N2 is N+1,
        genListLists(N2, Size, NewBoard, Board).

%genList(+N, +Size, +OldList, -List)
genList(Size, Size, List, List).
genList(N, Size, OldList, List):-
        Obs is random(5),
        genObstacle(Obs, Field),
        appendList(OldList, [Field], NewList),
        N2 is N+1,
        genList(N2, Size, NewList, List).

%genObstacle(+Obs, -Field)
genObstacle(0, 3).
```

```
genObstacle(_, 0).



%%%%%%%%%%%%%%%%%%%%%%%%
%%%% VISUALIZATION %%%%
%%%%%%%%%%%%%%%%%%%%%%%%

%drawBoard(+Board, +Size)
drawBoard(Board, Size):-
       DashNum is Size*2-1,
       nl, write('    '), drawLetters(65, Size),
       nl, write('     '), drawDashes(DashNum), nl,
       drawLines(1, Board),
       write('     '), drawDashes(DashNum), nl,
       write('    '), drawLetters(65, Size), nl, nl,!.

%drawDashes(+DashNum)
drawDashes(0).
drawDashes(DashNum):-
       write('-'),
       DashNewNum is DashNum-1,
       drawDashes(DashNewNum).

%drawLetters(+Char, +Size)
drawLetters(_, 0).
drawLetters(Char, Size):-
       write(' '), put_char(Char),
       NewSize is Size-1,
       NextChar is Char+1,
       drawLetters(NextChar, NewSize).

%drawLines(+N, +Board)
drawLines(_,[]).
drawLines(N, [Line|T]):-
       (N < 10 -> write(' '); write('')),
       write(N), write(' |'), drawLine(Line), write(' | '),
       write(N), nl,
       N2 is N+1,
       drawLines(N2, T).

%drawLine(+Line)
drawLine([]).
drawLine([Char|T]):-
       writeChar(Char),
       drawLine(T).

%writeChar(+Char)
writeChar(0):- write(' .').
writeChar(1):- write(' O').
writeChar(2):- write(' X').
writeChar(3):- write(' #').



%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%% MOVE COMPUTATION %%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%

%computeMove(+Level, +MoveList, -Move)
computeMove(1, MoveList, Move):-
       keysort(MoveList, MoveListSorted),
       reverse(MoveListSorted, MoveListReversed),
```

```
            MoveListReversed = [_-Move|_].

      computeMove(2, MoveList, Move):-
            Level is random(2)*2 + 1,
            computeMove(Level, MoveList, Move).

      computeMove(3, MoveList, Move):-
            keysort(MoveList, MoveListSorted),
            MoveListSorted = [_-Move|_].

      %reverseAI(+AI, -ReversedAI)
      reverseAI(0, 0).
      reverseAI(1, 3).
      reverseAI(2, 2).
      reverseAI(3, 1).

      %moveCount(+Board, +Player, -Count, +CurrentPiece)
      moveCount(Board, Player, Count, CurrentPiece):-
            moveCountAuxY(Board, Board, 1, Player, Count, 0, CurrentPiece).

      %moveCountAuxY(+Board,   +BoardAux,   +Y,   +Player,   -Count,   +CountAux,
+CurrentPiece)
      moveCountAuxY(_, [], _, _, CountAux, CountAux, _).
      moveCountAuxY(Board, [H|T], Y, Player, Count, CountAux, CurrentPiece):-
            moveCountAuxX(Board, H, 1, Y, Player, CountX, 0, CurrentPiece),
            Yn is Y+1,
            CountTemp is CountX+CountAux,
            moveCountAuxY(Board,  T,  Yn,  Player,  Count,  CountTemp,  Current-
Piece).

      %moveCountAuxX(+Board,  +BoardAux,  +X,  +Y,  +Player,  -CountX,  +CountAux,
+CurrentPiece)
      moveCountAuxX(_, [], _, _, _, CountAux, CountAux, _).
      moveCountAuxX(Board,  [_|T],  X,  Y,  Player,  CountX,  CountAux,  Current-
Piece):-
            cons(X, Y, Move),
            Xn is X+1,
            (validMove(Board, Player, Move, CurrentPiece) ->
                  CountTemp is CountAux+1,
                  moveCountAuxX(Board,  T,  Xn,  Y,  Player,  CountX,  CountTemp,
CurrentPiece);
                  moveCountAuxX(Board,  T,  Xn,  Y,  Player,  CountX,  CountAux,
CurrentPiece)).

      %moveList(+Board, +Player, -MoveList, +CurrentPiece)
      moveList(Board, Player, MoveList, CurrentPiece):-
            moveListAuxY(Board, Board, 1, Player, [], MoveList, CurrentPiece).

      %moveListAuxY(+Board,   +BoardAux,   +Y,   +Player,   +TempList,   -MoveList,
+CurrentPiece)
      moveListAuxY(_, [], _, _, TempList, TempList, _).
      moveListAuxY(Board, [H|T], Y, Player, TempList, MoveList, CurrentPiece):-
            moveListAuxX(Board,  H,  1,  Y,  Player,  TempList,  AuxList,  Current-
Piece),
            Yn is Y+1,
            moveListAuxY(Board,  T,  Yn,  Player,  AuxList,  MoveList,  Current-
Piece).

      %moveListAuxX(+Board,  +BoardAux,  +X,  +Y,  +Player,  +TempList,  -MoveList,
+CurrentPiece)
      moveListAuxX(_, [], _, _, _, TempList, TempList, _).
      moveListAuxX(Board,  [_|T],  X,  Y,  Player,  TempList,  MoveList,  Current-
Piece):-
```

```
            cons(X, Y, Move),
            Xn is X+1,
            (validMove(Board, Player, Move, CurrentPiece) ->
                  assignMove(CurrentPiece, Player, Move, Board, NewBoard),
                  changePlayer(Player, NextPlayer),
                  moveCount(NewBoard, NextPlayer, Count, 0),
                  appendList(TempList, [Count-Move], NewList),
                  moveListAuxX(Board, T, Xn, Y, Player, NewList , MoveList,
CurrentPiece);
                  moveListAuxX(Board, T, Xn, Y, Player, TempList, MoveList,
CurrentPiece)).



      %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
      %%%% ASSIGNMENT OF A VALID MOVEMENT %%%%
      %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

      %validMove(+Board, +Player, +Move, +CurrentPiece)
      validMove(Board, Player, Move, CurrentPiece):-
            changePlayer(Player, OtherPlayer),
            car(Move, X), cdr(Move, Y),
            boardMember(CurrentPiece, X, Y, Board),
            XnRight is X+1,          not(boardMember(OtherPlayer, XnRight, Y,
Board)),
            XnLeft is X-1,            not(boardMember(OtherPlayer, XnLeft, Y,
Board)),
            YnUp is Y+1,         not(boardMember(OtherPlayer, X, YnUp, Board)),
            YnDown is Y-1,            not(boardMember(OtherPlayer, X, YnDown,
Board)),!.

      %assignMove(+OldPiece, +NewPiece, +Move, +Board, -NewBoard)
      assignMove(OldPiece, NewPiece, Move, Board, NewBoard):-
            car(Move, X), cdr(Move, Y),
            assignMoveAux(1, OldPiece, NewPiece, X, Y, Board, NewBoard),!.

      %assignMoveAux(+Yn, +OldPiece, +NewPiece, +X, +Y, +Board, -NewBoard)
      assignMoveAux(_, _, _, _, _, [], []).
      assignMoveAux(Y, OldPiece, NewPiece, X, Y, [Line|T1], [NewLine|T2]):-
            changeLine(1, OldPiece, NewPiece, X, Line, NewLine),
            Yn is Y+1,
            assignMoveAux(Yn, OldPiece, NewPiece, X, Y, T1, T2).

      assignMoveAux(Yn, OldPiece, NewPiece, X, Y, [Line|T1], [Line|T2]):-
            Yn \= Y, Ytemp is Yn+1,
            assignMoveAux(Ytemp, OldPiece, NewPiece, X, Y, T1, T2).

      %changeLine(+Xn, +OldPiece, +NewPiece, +X, +Board, -NewBoard)
      changeLine(_, _, _, _, [], []).
      changeLine(X, OldPiece, NewPiece, X, [OldPiece|T1], [NewPiece|T2]):-
            Xn is X+1,
            changeLine(Xn, OldPiece, NewPiece, X, T1, T2).

      changeLine(Xn, OldPiece, NewPiece, X, [H|T1], [H|T2]):-
            Xn \= X, Xtemp is Xn+1,
            changeLine(Xtemp, OldPiece, NewPiece, X, T1, T2).



      %%%%%%%%%%%%%%%%%%%%%%%%
      %%%% SPECIAL MOVES %%%%
      %%%%%%%%%%%%%%%%%%%%%%%%
```

```
%specialMove(+Board, +Player, +Size, +NextTurn, +MoveList, +MoveCount)
specialMove(Board, 1, Size, NextTurn, _, _):-
	write('Cats special move! Remove a piece from the dogs.'), nl,
	moveList(Board, 0, MoveList, 2),
	player(1, PlayerType, AI),
	reverseAI(AI, ReversedAI),
	getMove(PlayerType, MoveList, Move, Size, ReversedAI),
	validMove(Board, 0, Move, 2),
	assignMove(2, 0, Move, Board, NewBoard),
	moveCount(NewBoard, 1, Count, 0),
	moveList(NewBoard, 1, NextMoveList, 0),
	(Count =:= 0 ->
		moveCount(NewBoard, 2, Count2, 0),
		write('No moves available!'), nl,
		(Count2 =:= 0 ->
			endGame(NewBoard, Size), !;
			play(NewBoard, 2, Size, NextTurn));
		drawBoard(NewBoard, Size),
		nl, writePlayer(1), write('´ turn:'), nl,
		getMove(PlayerType, NextMoveList, NextMove, Size, AI),
		validMove(NewBoard, 1, NextMove, 0),
		assignMove(0, 1, NextMove, NewBoard, FinalBoard),
		play(FinalBoard, 2, Size, NextTurn)).

specialMove(Board, 2, Size, NextTurn, MoveList, MoveCount):-
	write('Dogs special move! You got an extra round.'), nl,
	(MoveCount =:= 0 ->
		moveCount(Board, 1, MoveCount2, 0),
		write('No moves available!'), nl,
		(MoveCount2 =:= 0 ->
			endGame(Board, Size), !;
			play(Board, 1, Size, NextTurn));
		player(2, PlayerType, AI),
		getMove(PlayerType, MoveList, Move, Size, AI),
		validMove(Board, 2, Move, 0),
		assignMove(0, 2, Move, Board, NewBoard),
		moveCount(NewBoard, 2, Count, 0),
		moveList(NewBoard, 2, NextMoveList, 0),
		drawBoard(NewBoard, Size),
		nl, writePlayer(2), write('´ turn:'), nl,
		(Count =:= 0 ->
			moveCount(NewBoard, 1, Count2, 0),
			write('No moves available!'), nl,
			(Count2 =:= 0 ->
				endGame(NewBoard, Size), !;
				play(NewBoard, 1, Size, NextTurn));
			getMove(PlayerType,  NextMoveList,  NextMove,  Size,
AI),
			validMove(NewBoard, 2, NextMove, 0),
			assignMove(0, 2, NextMove, NewBoard, FinalBoard),
			play(FinalBoard, 1, Size, NextTurn))).



%%%%%%%%%%%%%%%%%%%%%
%%%% PIECE COUNT %%%%
%%%%%%%%%%%%%%%%%%%%%

%countPieces(+Board, +Player, -Count)
countPieces(Board, Player, Count):-
	countPiecesY(Board, Player, 1, 0, Count), !.

%countPiecesY(+Board, +Player, +Y, +CountAux, -Count)
```

```
countPiecesY([], _, _, CountAux, CountAux).
countPiecesY([H|T], Player, Y, CountAux, Count):-
       countPiecesX(H, Player, 1, 0, CountX),
       CountTemp is CountAux+CountX,
       Yn is Y+1,
       countPiecesY(T, Player, Yn, CountTemp, Count).

%countPiecesY(+Line, +Player, +X, +CountAux, -Count)
countPiecesX([], _, _, CountAux, CountAux).
countPiecesX([H|T], H, X, CountAux, Count):-
       CountTemp is CountAux+1,
       Xn is X+1,
       countPiecesX(T, H, Xn, CountTemp, Count).
countPiecesX([_|T], Player, X, CountAux, Count):-
       Xn is X+1,
       countPiecesX(T, Player, Xn, CountAux, Count).



%%%%%%%%%%%%%%%%%%%%%%%%
%%%% LIST UTILITIES %%%%
%%%%%%%%%%%%%%%%%%%%%%%%

%boardMember(?Piece, +X, +Y, +Board)
boardMember(Piece, X, Y, Board):-
       searchList(Line, Y, Board),
       searchList(Piece, X, Line).

%searchList(?Member, +N, +List)
searchList(Member, N, List):-
       searchListAux(Member, 1, N, List).

%searchListAux(?Member, +Idx, +N, +List)
searchListAux(Member, N, N, [Member|_]).
searchListAux(Member, Idx, N, [_|T]):-
       IdxTemp is Idx+1,
       searchListAux(Member, IdxTemp, N, T).

%appendList(+List1, +List2, -Result)
appendList([],X,X).
appendList([H|T1],X,[H|T2]):-
       appendList(T1,X,T2).

%lengthList(+List, -Length)
lengthList(List, Length):-
       lengthListAux(List, Length, 0).

%lengthListAux(+List, -Length, +LengthAux)
lengthListAux([], LengthAux, LengthAux).
lengthListAux([_|T], Length, LengthAux):-
       LengthTemp is LengthAux+1,
       lengthListAux(T, Length, LengthTemp).

%cons(+X, +Y, -Pair)
cons(X, Y, [X|[Y]]).

%car(+Pair, -Car)
car([Car|_], Car).

%cdr(+Pair, -Cdr)
cdr([_|[Cdr|_]], Cdr).
```

```
%%%%%%%%%%%%%%%%
%%%% SOCKETS %%%%
%%%%%%%%%%%%%%%%
/*
port(60001).


% Server %

server:-
       current_host(Host),
       server(Host).

server(Host):-
       port(Port),
       socket('AF_INET', Socket),
       socket_bind(Socket, 'AF_INET'(Host,Port)),
       socket_listen(Socket, 5),
       socket_accept(Socket, _Client, Stream),
       server_loop(Stream),
       socket_close(Socket),
       write('Server exit'), nl.

server_loop(Stream):-
       repeat,
       read(Stream, ClientRequest),
       format('Server received: ~q~n', [ClientRequest]),
       server_input(ClientRequest, ServerReply),
       server_reply(ServerReply, Stream),
       (ClientRequest == bye; ClientRequest == end_of_file), !.

server_input(initialize, ok):-
       begin, !.
server_input(execute(Move, Board), ok(NewBoard)):-
       validMove(Move, Board),
       assignMove(Move, Board, NewBoard), !.
server_input(calculate(AI, Player, Board), ok(Move, NewBoard)):-
       moveList(Board, Player, MoveList, 0),
       computeMove(AI, MoveList, Move)
       assignMove(Move, Board, NewBoard), !.
server_input(end(Board), ok):-
       endGame, !.
server_input(bye, ok):- !.
server_input(end_of_file, ok):- !.
server_input(_, invalid):- !.
*/
```

4.9 - Anexo B - Screenshots da Interface Visual

```
3 ?- begin.

Cats & Dogs - Prolog - 2008.11.02
Joao Xavier
Joao Ribeiro

Input game mode:
Cats:
1 - Human, 2 - CPU
|: 2

Dogs:
1 - Human, 2 - CPU
|: 2

Input CPU (Cats) Level:
1 - Easy, 2 - Medium, 3 - Hard
|: 2

Input CPU (Dogs) Level:
1 - Easy, 2 - Medium, 3 - Hard
|: 3

Choose a board size between 4 and 12:
|: 5


     A B C D E
     ---------
 1 | # . . . . | 1
 2 | . . # . . | 2
 3 | . . # . . | 3
 4 | . . . # # | 4
 5 | # . . # . | 5
     ---------
     A B C D E
```

Imagem 1

```
Input game mode:
Cats:
1 - Human, 2 - CPU
|: 1

Dogs:
1 - Human, 2 - CPU
|: 2

Input CPU (Dogs) Level:
1 - Easy, 2 - Medium, 3 - Hard
|: 3

Choose a board size between 4 and 12:
|: 4


     A B C D
     -------
 1 | . . . . | 1
 2 | . . . . | 2
 3 | . . . . | 3
 4 | . . . . | 4
     -------
     A B C D


Cats' turn:
X: b
Y: 2

     A B C D
     -------
 1 | . . . . | 1
 2 | . O . . | 2
 3 | . . . . | 3
 4 | . . . . | 4
     -------
     A B C D
```

Imagem 2

```
Dogs´ turn:
Dogs special move! You got an extra round.

     A B C D E
     ---------
 1 | O . X . O | 1
 2 | . X . O . | 2
 3 | # . X . X | 3
 4 | . O . # . | 4
 5 | X . # # # | 5
     ---------
     A B C D E


Dogs´ turn:

     A B C D E
     ---------
 1 | O . X . O | 1
 2 | . X . O . | 2
 3 | # . X . X | 3
 4 | . O . # X | 4
 5 | X . # # # | 5
     ---------
     A B C D E


Cats´ turn:
Cats special move! Remove a piece from the dogs.

     A B C D E
     ---------
 1 | O . X . O | 1
 2 | . X . O . | 2
 3 | # . X . X | 3
 4 | . O . # X | 4
 5 | . . # # # | 5
     ---------
     A B C D E
```

Imagem 3

```
Input game mode:
Cats:
1 - Human, 2 - CPU
|: hgdosapd9n 8
|: ae2 232s asda
|: 11
|: 2

Dogs:
1 - Human, 2 - CPU
|: 2

Input CPU (Cats) Level:
1 - Easy, 2 - Medium, 3 - Hard
|: 3
```

Imagem 4

```
Choose a board size between 4 and 12:
|: 567890'
|: 16
|: 3
|: a
|: 5


    A B C D E
    ---------
1 | . . . . . | 1
2 | . . . . . | 2
3 | # . . . . | 3
4 | . . . # . | 4
5 | . . # # # | 5
    ---------
    A B C D E
```

Imagem 5

```
Dogs´ turn:

    A B C D E
    ---------
1 | . . . . . | 1
2 | . O # . . | 2
3 | . . . X . | 3
4 | . # . . . | 4
5 | # . . . . | 5
    ---------
    A B C D E


Cats´ turn:
X: b
Y: 2
X: c
Y: 2
X: d
Y: 3
X:
```

Imagem 6

```
Dogs´ turn:
No moves available!

    A B C D E
    ---------
1 | O . X . O | 1
2 | . X . O . | 2
3 | # . X . X | 3
4 | O O . # X | 4
5 | O . # # # | 5
    ---------
    A B C D E


Cats´ turn:

    A B C D E
    ---------
1 | O . X . O | 1
2 | . X . O . | 2
3 | # . X . X | 3
4 | O O . # X | 4
5 | O O # # # | 5
    ---------
    A B C D E
```

Imagem 7

```
Dogs´ turn:
No moves available!

    A B C D E
    ---------
1 | O . X . O | 1
2 | . X . O . | 2
3 | # . X . X | 3
4 | O O . # X | 4
5 | O O # # # | 5
    ---------
    A B C D E

7 - 5
Cats win!
true
```

Imagem 8