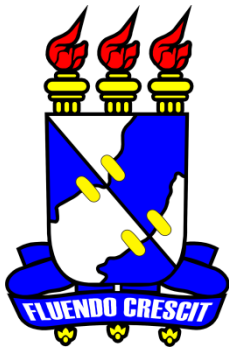

UFS - UNIVERSIDADE FEDERAL DE SERGIPE

CCET - CENTRO DE CIÊNCIAS EXATAS E TECNOLOGIA

DCOMP - DEPARTAMENTO DE COMPUTAÇÃO

CIÊNCIA DA COMPUTAÇÃO

PROGRAMAÇÃO DECLARATIVA

PROJETO FINAL: EXTENSÃO DO JOGO DAMAS

PROF. HENDRIK TEXEIRA MACEDO

JAINE DA CONCEICAO SANTOS – 201600017409

São Cristóvão - SE

29 de Setembro de 2017

SUMÁRIO

1- Introdução

Prolog é uma linguagem de programação que se enquadra no paradigma de Programação em Lógica Matemática. É uma linguagem de uso geral que é especialmente associada com a inteligência artificial e linguística computacional [1]. Assim, na matéria de Programação Declarativa foi ensinado Prolog na útlima unidade, e como tentativa de fazer os alunos ter um maior contanto com essa linguagem, o projeto final do curso consistiu em estender implementações de jogos utilizando Prolog. O jogo o qual esse relatório trata é o jogo de Damas.

Damas é um jogo de tabuleiro, praticado entre dois jogadores, num tabuleiro quadrado, de 64 casas alternadamente claras e escuras, dispondo de 12 peças brancas e 12 pretas. O objetivo é capturar ou imobilizar as peças do adversário. O jogador que conseguir comer todas as peças do inimigo ganha a partida. O tabuleiro deve ser colocado de modo que a casa angular à esquerda de cada parceiro seja escura. Além disso, no início da partida, as peças devem ser colocadas no tabuleiro sobre as casas escuras da seguinte forma: nas três primeiras filas horizontais, as peças brancas; e, nas três últimas, as peças pretas. A peça movimenta-se em diagonal, sobre as casas escuras, para a frente, e uma casa de cada vez [2]. Um detalhe a ser considerado é que as regras de damas podem variar em cada país. As regras seguidas na implementação da extensão do jogo desse relatório tentam seguir o máximo as regras oficiais e são elas:

- A peça pode capturar a peça do adversário movendo-se para frente e permitindo também capturar a peça do adversário movendo-se para trás.
- A peça que atingir a oitava casa adversária, parando ali, será promovida a "dama", peça de movimentos mais amplos que a simples peça.
- A dama pode mover-se para trás e para frente em diagonal uma casa de cada vez, diferente das outras peças, que movimentam-se apenas para frente em diagonal.
- Também, a tomada (comer a peça adversária) é obrigatória. A peça e a dama têm o mesmo valor para tomar ou ser tomada.
- As brancas (na interface implementada são representadas de amerelo) têm sempre a saída, isto é, o primeiro lance da partida.

A regra deixada de fora foi: Se, no mesmo lance, se apresentar mais de um modo de tomar, é obrigatório executar o lance que tome o maior número de peças (lei da maioria). Pois, do jeito que foi implementado a extensão do jogo só é permitido capturar uma peça adversária por vez.

2-Problemas encontrados

Diante da leitura e análise do relatório, do código fonte .pl e das classes java (para interface) usadas na implementação sem extensão desse jogo, notou-se que da forma que o jogo havia sido desenvolvido ele possuia alguns problemas. Assim, foram identificados quatro principais problemas:

1. A implementação havia sido feita usando JPL e dessa forma não era mais possível executar o jogo;
2. O predicado para comer a peça adversária dava erro e o jogo parava de funcionar;
3. O jogo só permitia jogar humano x humano;
4. O jogador não era obrigado a comer sempre que possível.

Segue abaixo figuras que demonstram esses problemas.

```
jpl_new('damas.Imagem',[],I),novoTabuleiro([LB,LP,LD]),
lerEntrada(E),
jpl_call('javax.swing.JOptionPane',showMessageDialog,[@null,'Jogador Amarelo Comeca'],_),
```

Figura 1. Exemplo do uso de predicados JPL no código, como *jpl_new* e *jpl_call.*

```
ERROR: Arguments are not sufficiently instantiated
ERROR: In:
ERROR:   [17] atomic_list_concat(_37684,',',_37688)
ERROR:   [16] interface([_37716,_37722|...]) at /home/jaine/Dropbox/Un
iversidade/3periodo/Prog Declarativa/Projeto Final - Damas/Damas/damas
/Projeto DAMAS/damasq.pl:262
ERROR:   [15] aux(y,[_37760,_37766|...],'Branco',_37754,44) at /home/j
aine/Dropbox/Universidade/3periodo/Prog Declarativa/Projeto Final - Da
mas/Damas/damas/Projeto DAMAS/damasq.pl:324
ERROR:   [13] aux(y,[[44|...],...|...],'Branco',_37798,44) at /home/ja
ine/Dropbox/Universidade/3periodo/Prog Declarativa/Projeto Final - Dam
as/Damas/damas/Projeto DAMAS/damasq.pl:329
ERROR:   [11] processa([[26|...],...|...],'Branco',_37846) at /home/ja
ine/Dropbox/Universidade/3periodo/Prog Declarativa/Projeto Final - Dam
as/Damas/damas/Projeto DAMAS/damasq.pl:286
ERROR:   [10] processa([[26|...],...|...],'Preto',_37892) at /home/jai
ne/Dropbox/Universidade/3periodo/Prog Declarativa/Projeto Final - Dama
s/Damas/damas/Projeto DAMAS/damasq.pl:287
ERROR:    [9] processa([[1|...],...|...],'Branco',_37938) at /home/jai
ne/Dropbox/Universidade/3periodo/Prog Declarativa/Projeto Final - Dama
s/Damas/damas/Projeto DAMAS/damasq.pl:287
ERROR:    [7] <user>
ERROR:
ERROR: Note: some frames are missing due to last-call optimization.
ERROR: Re-run your program in debug mode (:- debug.) to get more detai
l.
   Exception: (15) aux(y, _38006, 'Branco', _38010, 44) ?
```

Figura 2. Exemplo do erro que acontecia ao tentar comer a peça adversária.

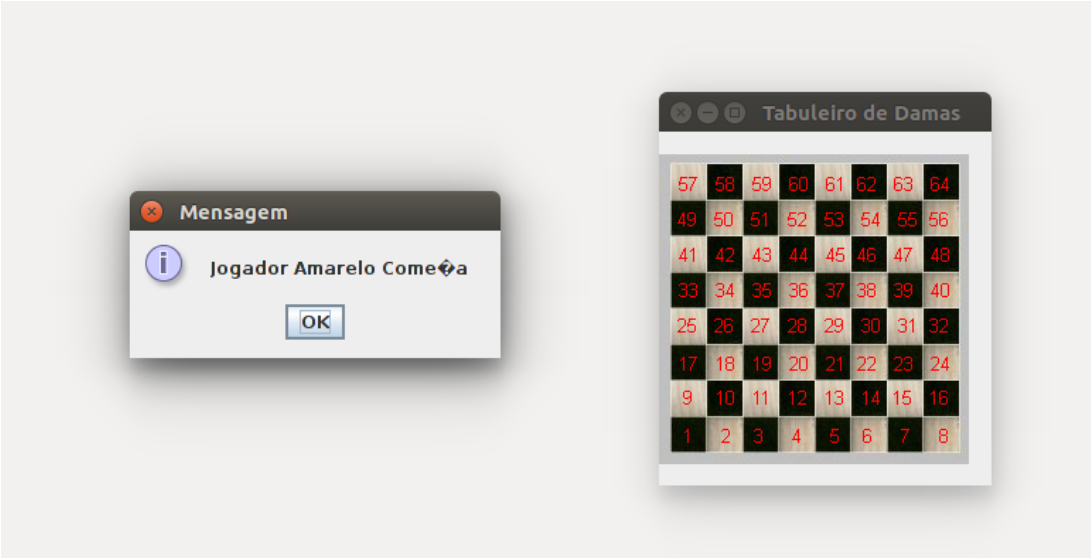

Figura 3. Exemplo do problema 3 (o jogo só permite jogar humano x humano).

A Figura 3 demonstra que ao iniciar o jogo não era dada uma opção de escolha ao usuário sobre jogar contra a máquina ou contra outro humano, apenas iniciava o jogo dizendo que o jogador amarelo começa.

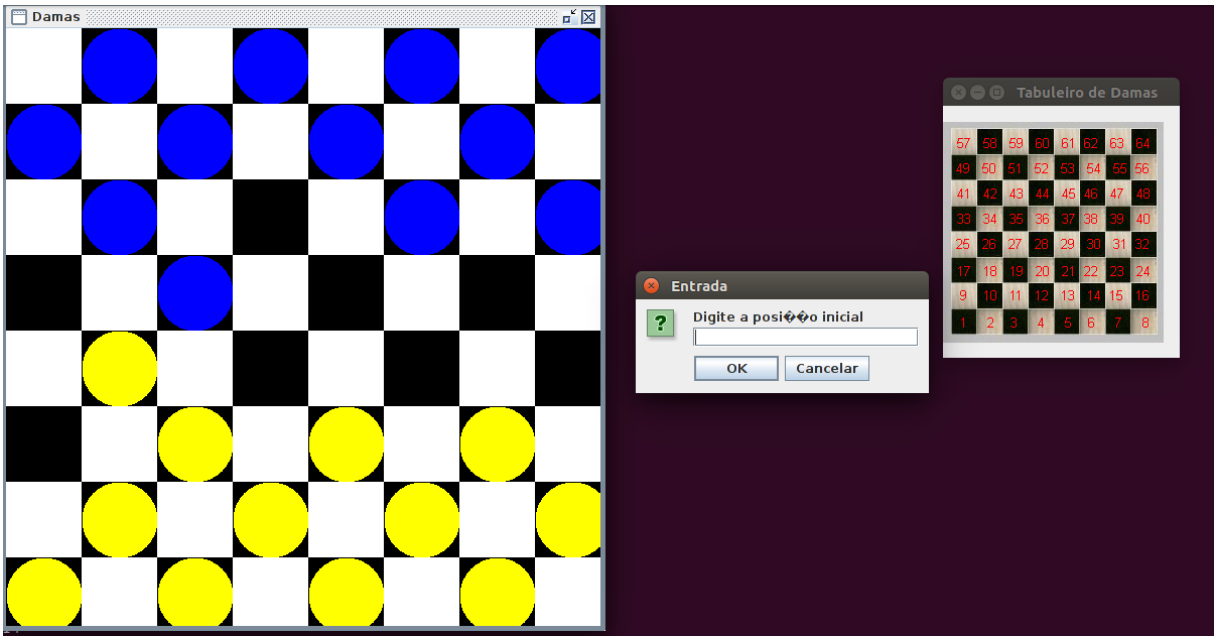

Figura 4. Exemplo do problema 4 (o jogador não é obrigado a comer sempre que possível).

A Figura 4 demonstra a vez de jogar do jogador humano representado pelas peças amarelas e que mesmo com a possibilidade de comer a peça do seu adversário, não precisa o fazer obrigatoriamente, tendo a opção de inserir em qual posição deseja jogar e com qual peça.

3- Proposta de extensão

Diante dos problemas encontrados e apresentados na seção 2, ficou combinado como proposta estender e contribir com o projeto do jogo de Damas, implementando o que fosse necessário para corrigir esses quatro problemas, utilizando o código fonte já existente, a linguagem Prolog para as regras de negócio do projeto e a linguagem Java para a interface gráfica.

4 - Implementação

A implementação para extensão do projeto se deu em quatro partes:

4.1 Primeira Parte: Execução do projeto com o JPL 7.4

Anter de implementar qualquer lógica nova, era necessário fazer o código executar para assim poder ver o seu funcionamento e testar implementações novas. Nesse contexto, temos o uso do JPL (Java interface to Prolog) que é um conjunto de classes Java que provê uma interface entre Java e Prolog. O JPL usa a Java Native Interface (JNI) para se conectar a um motor Prolog que está mais ou menos no processo de padronização em várias implementações do Prolog. JPL não é uma implementação Java pura do Prolog; Faz uso extensivo de implementações nativas do Prolog em plataformas suportadas [3].

Assim, o problema 1 encontrado e citado na seção 2 foi o de que a implementação havia sido feita usando o JPL e dessa forma não estava sendo possível executar o jogo. Dessa forma, a primeira coisa feita para a contribuição no projeto foi fazê-lo executar. Para isso, foi utilizada a versão mais recente encontrada para download do JPL, o JPL 7.4, e através de pesquisas e tentativas, descobriu-se que para usar Java de um programa Prolog, que é a situação do jogo damas desse relatório, era necessário realizar os seguintes passos:

- Adicionar ao projeto java já existente, que representa a interface, a biblioteca jpl-7.4.0.jar.
- Modificar o LD_LIBRARY_PATH para conter os diretórios que contêm libjava.so, libjni.so e libjsig.so.
- Buscar esses diretórios usando o seguinte comando: dpkg --search libjsig.so libjava.so
- E assim obter como comando final: LD_LIBRARY_PATH=/usr/lib/jvm/java-8-openjdk-amd64/jre/lib/amd64/server:/usr/lib/jvm/java-8-openjdk-amd64/jre/lib/amd64/ swipl

Além disso, o codigo.pl teve que estar na mesma pasta que o pacote com as classes Java da interface juntamente com a imagem tabuleiro.jpg. Também, não foi necessário mudar nada na sintaxe do código prolog e nem alterar nada das classes java.

Por fim, vale ressaltar que na máquina que executou o projeto roda o sistema operacional Ubuntu 16.10, a versão do swipl instalada é a SWI-Prolog version 7.4.2 for amd64 e a versão do java é a 8.

4.2 Segunda Parte: Corrigir o erro que acontece ao tentar comer peça adversária

A segunda contribuição se dá à correção do problema 2 (o predicado para comer a peça adversária apresenta erro e o jogo para de funcionar). Para corrigir esse erro, o predicado "estado" que implementava essa estratégia de comer foi modificado.

A implementação abaixo é a que apresenta o erro. Ela utiliza um predicado *lerC* que pergunta se o usário quer comer ou não (y/n) e armazena a resposta em C, passando C para o predicado *aux* que pede outra posição na tentativa de capturas consecutivas, porém apresenta erro como mostrado na seção 2.

```
estado(1,Y,Y1,Turno,P):-lerC(C),aux(C,Y,Turno,Y1,P).

aux('y',[LB,LP,LD],Turno,Y,PI):-
        interface([LB,LP,LD]),
        lerPF(PF1),
        str_int(PF1,PF),
        ehDama(LD,PI,D),
        ehJogadaValida2(D,1,[LB,LP,LD],Turno,X,PI,PF),
        estado(1,X,Y,Turno,PF).
aux(_,Y,Turno,Y,P).
```

Para correção então, esses predicados foram modificados de forma que a regra seguida foi a de apenas uma captura por vez. Ficando a implementação como mostrado abaixo, onde o predicado *lerC* foi renomeado para *comerC* e agora o usuário não insere mais (y/n) para comer, pois se ele move a peça de maneira que uma peça adversária fica no meio de seus movimentos, significa que a única possibilidade é capturar. Assim, o usuário apenas seleciona "ok" em um JOptionPane e o valor de C passado em *aux* é automaticamente 'y'. Por fim, *aux* agora apenas atualiza a interface do tabuleiro removendo a peça recém capturada.

```
estado(1,Y,Y1,Turno,P):-comerC(C),aux(C,Y,Turno,Y1,P).

aux('y',[LB,LP,LD],Turno,Y,PI):-interface([LB,LP,LD]).

comerC(C):-
        jpl_new( 'javax.swing.JFrame', ['frame with dialog'], F),
        jpl_call( F, setLocation, [400,300], _),
        jpl_call( F, setSize, [400,300], _),
        jpl_call( F, setVisible, [@(false)], _),
        jpl_call( F, toFront, [], _),
        jpl_call( 'javax.swing.JOptionPane', showMessageDialog, [F,'Aperte ok para comer'], C),
        jpl_call( F, dispose, [], _).
```

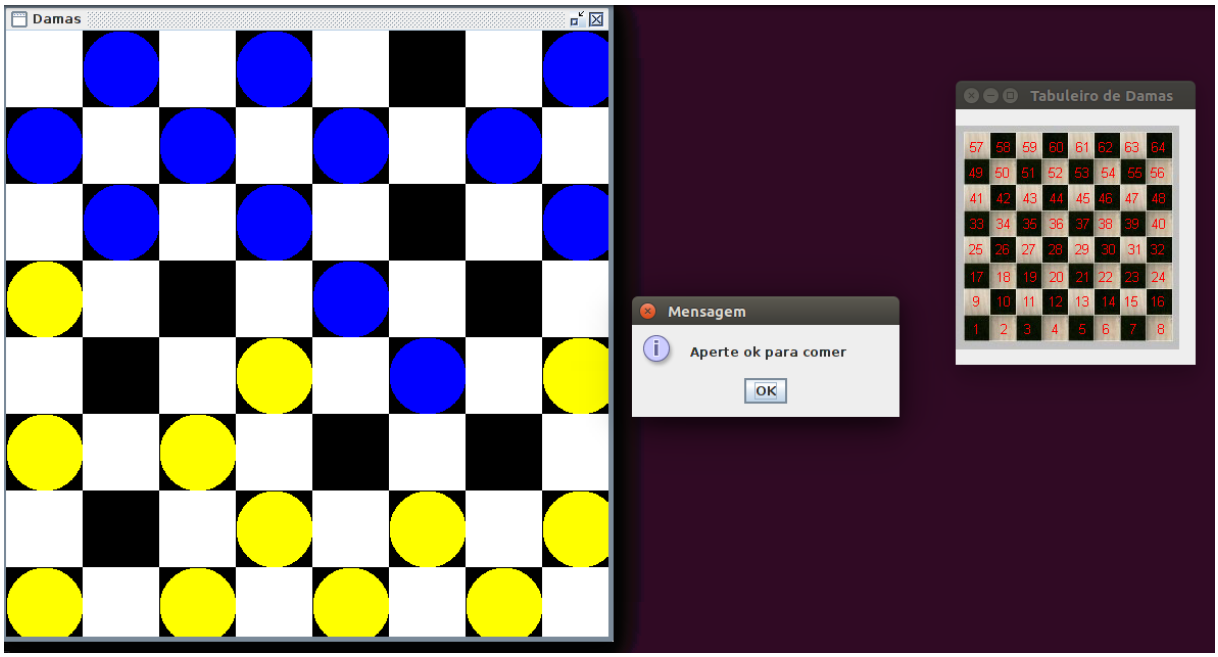

Figura 5. Execucação do predicado *estado* modificado com o novo *comerC*.

A Figura 5 demonstra uma jogada do jogador amarelo que vai da posição 28 para a 46 de modo que uma peça adversária fica entre seus movimentos e assim aparece na tela a mensagem “Aperte ok para comer”, como implementado no novo predicado *estado*.

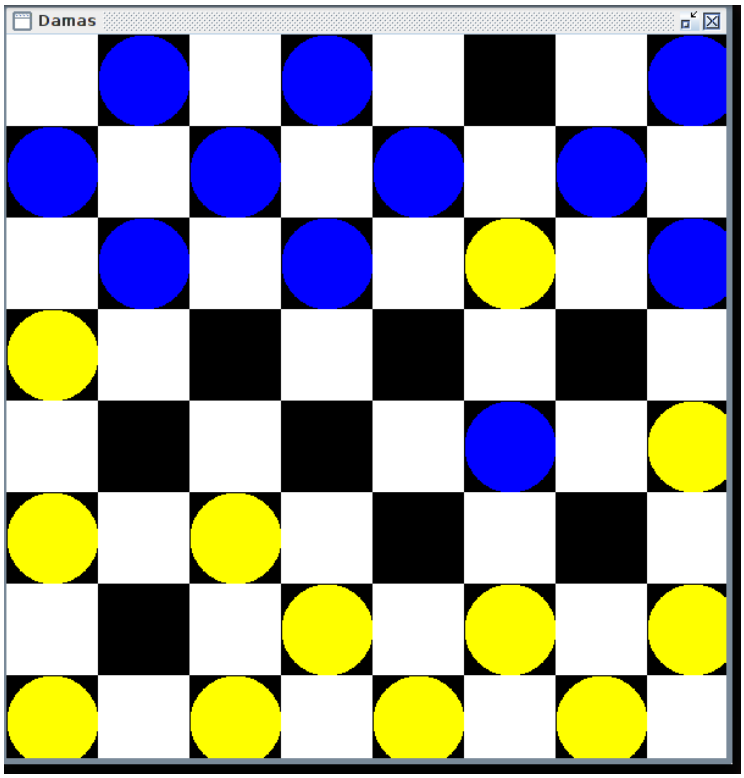

Figura 6. Execução do predicado *aux*.

Já a Figura 6, mostra a execução de *aux*, que como explicado anteriormente, agora apenas atualiza o tabuleiro removendo a peça comida.

4.3 Terceira Parte: Implementar a opção de jogar humano x computador

A terceira contribuição consiste em permitir o usuário jogar contra o computador se assim ele quiser. Para implementar tal lógica foi usado um novo predicado *lerEntrada* que utiliza um JOptionPane para perguntar ao usuário se ele deseja jogar contra a máquina (opção 1) ou contra outro humano (opção 2). Assim, a opção dada pelo usuário é massa para o predicado principal para execução do jogo, *processa*, nele também é passado um conjuntos de listas com as peças amarelas, azuis e as damas, assim como o turno do jogador, e uma variável auxiliar utilizada na implementação da Quarta Parte. Vale reassaltar que a máquina é sempre representada pelas peças azuis no tabuleiro da interface Java. Segue abaixo o predicado *lerEntrada* e *processa* da máquina.

```
%ler a opção do usuario para jogar contra a máquina ou contra outro humano.
lerEntrada(E):-
        jpl_new( 'javax.swing.JFrame', ['frame with dialog'], F),
        jpl_call( F, setLocation, [400,300], _),
        jpl_call( F, setSize, [400,300], _),
        jpl_call( F, setVisible, [@(false)], _),
        jpl_call( F, toFront, [], _),
        jpl_call( 'javax.swing.JOptionPane', showInputDialog, [F,'Pressione 1 para jogar
        contra a maquina e 2 para jogar contra outra pessoa'], E),
        jpl_call( F, dispose, [], _).

processa([LB,LP,LD],'Preto',_,'1',0):-          %jogada da maquina
        interface([LB,LP,LD]),nl,
        pick_nums(PI1,LP,AUX,PF1),              %lógica da maquina(aleatoria)
        ehDama(LD,PI1,D),
        ehJogadaValidaMaquina(D,E,[LB,LP,LD],Turno,X,PI1,PF1,C),
        estadoMaquina(E,X,Y,Turno,PF1),
        jpl_call('javax.swing.JoptionPane',showMessageDialog,[@null,'AmaquinaJogou'],_),
        trocarTurno(Turno,Turno2),
        processa(Y,Turno2,_,'1',C).
```

Na implementação da jogada da máquina foi tentado utilizar o aspecto de agência que é um mecanismo simplificado para que o usuário alimente com sua lógica. Assim, toda a lógica da máquina está no predicdo *pick_nums*, que na implementação feita para essa extensão foi utilizada uma lógica aleatória. Logo, se algupem quiser entrar com uma nova lógica, basta remover pick_nums e colocar seu novo predicado. Segue abaixo a figura da lógica de *pick_nums*.

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%  CALCULO DA JOGADA DO COMPUTADOR  %%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

chute(N,S):-repeat, S is random(N).            %jogada atual é aleatória
selecionaPF(1,PI1,PF1):- PF1 is PI1-7.
selecionaPF(2,PI1,PF1):- PF1 is PI1-9.
selecionaPF(3,PI1,PF1):- PF1 is PI1-18.
selecionaPF(4,PI1,PF1):- PF1 is PI1-14.
selecionaPF(5,PI1,PF1):- PF1 is PI1+7.
selecionaPF(6,PI1,PF1):- PF1 is PI1+9.
selecionaPF(7,PI1,PF1):- PF1 is PI1+18.
selecionaPF(8,PI1,PF1):- PF1 is PI1+14.

pick_nums(PI1,LP,AUX,PF1):-
        repeat, random_member(PI1,LP),chute(9,AUX),selecionaPF(AUX,PI1,PF1).
```

Figura 7. Predicado pick_nums que representa a jogada da máquina.

Na implementação mostrada na Figura 7, a máquina a princípio utiliza o predicado *random_member* na tentativa de ser mais eficiente, pois com esse predicado a máquina só seleciona aleatoriamente uma posição inicial que faça parte das posições possíveis da lista de peças da máquina, assim não precisamos tentar as 64 peças possíveis. Depois de escolher a posição inicial (PI1), utiliza-se o predicado *chute* que pode cair aleatoriamente em 8 opções, que são as posiçoes finais possíveis da peça (PF1), seleciona uma dessas posições através dos fatos *selecionaPF* e assim verifica se é uma jogada válida. Caso não seja, repete a lógica.

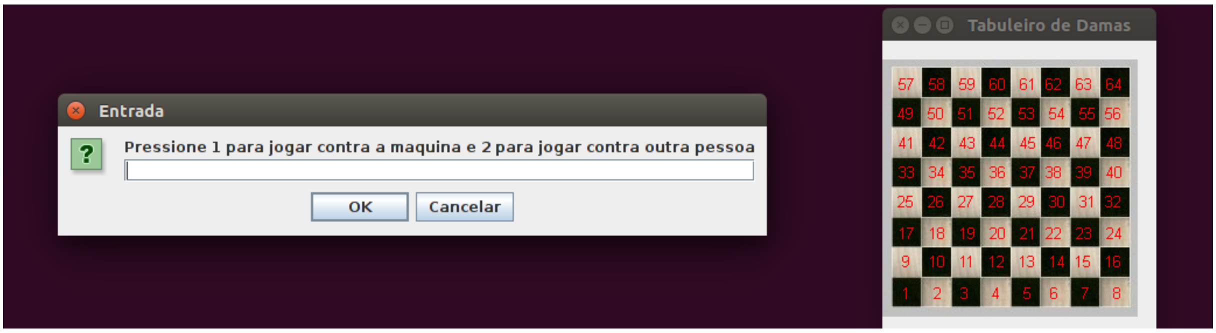

Figura 8. Execução do predicado *lerEntrada*.

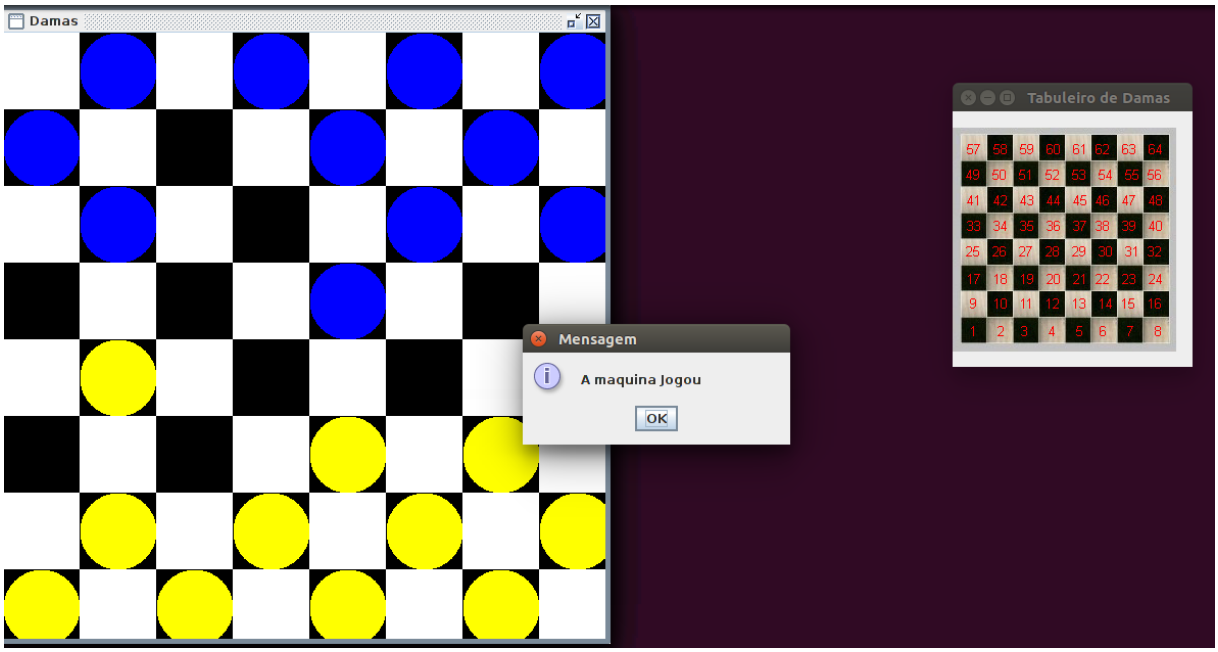

Figura 9. Execução do predicado *processa* da máquina.

4.4 Quarta Parte: Implementar tomada/captura obrigatória

A última contribuição na extensão desse jogo foi implementar a captura obrigatória. Para isso, foram implementados novos predicados *processa* para verificar se há a possibilidade de comer e assim comer. Para implementação dessa estratégia foi implementado também um predicado *checaPossibilidadeDeComer* que verifica as diagonais da última peça jogada pelo adversário e checa se é possível comer, considerando inclusive comer pra trás.

```
%se a ultima posicao do adversario -7 for uma peca do jogador tento comer com a peça PF1+7
checaPossibilidadeDeComer(1,PF1,LB,S,PI):- PI is PF1-7,write(PI),member(PI,LB),S is PF1+7.
%se a ultima posicao do adversario -9 for uma peca do jogador tento comer com a peça PF1+9
checaPossibilidadeDeComer(2,PF1,LB,S,PI):- PI is PF1-9,write(PI),member(PI,LB),S is PF1+9.
%se a ultima posicao do adversario +7 for uma peca do jogador tento comer com a peça PF1-7
checaPossibilidadeDeComer(3,PF1,LB,S,PI):- PI is PF1+7,write(PI),member(PI,LB),S is PF1-7.
%se a ultima posicao do adversario +9 for uma peca do jogador tento comer com a peça PF1-9
checaPossibilidadeDeComer(4,PF1,LB,S,PI):- PI is PF1+9,write(PI),member(PI,LB),S is PF1-9.
```

Vale ressaltar que a implementação da captura obrigatória foi implementada tanto para as jogadas dos humanos na opção humano x humano, como para as jogadas do humano e da máquina na opção humano x computador.

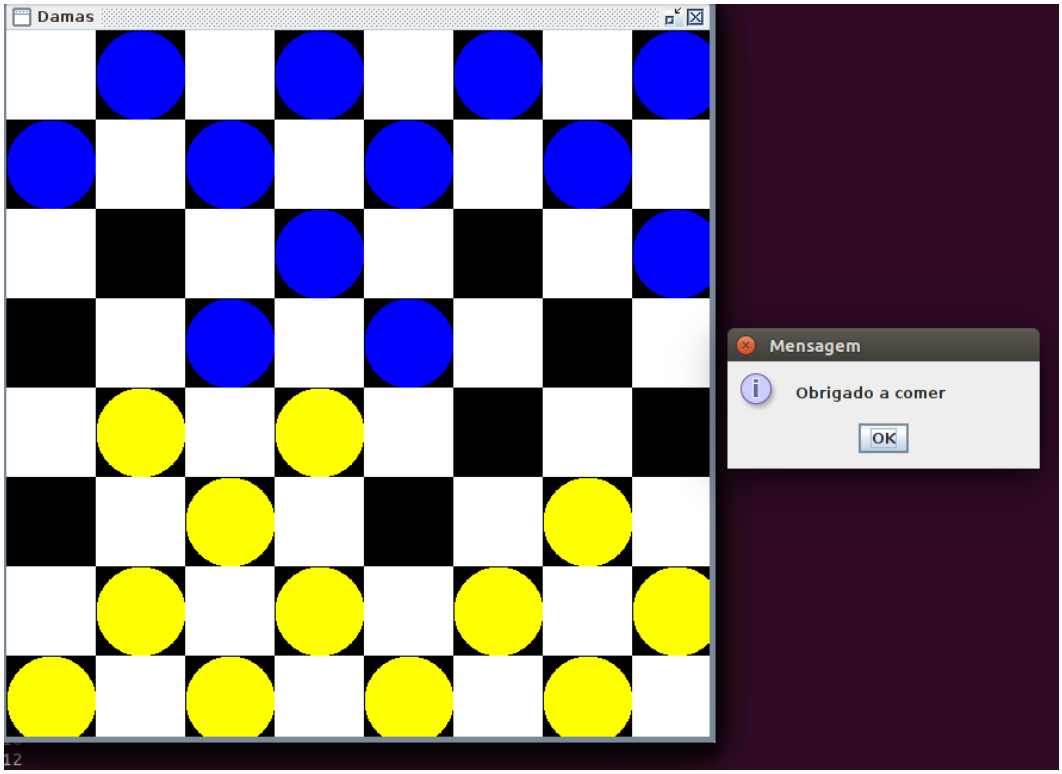

Figura 10. Execução da implementação de captura obrigatória pelo humano (peças amarelas).

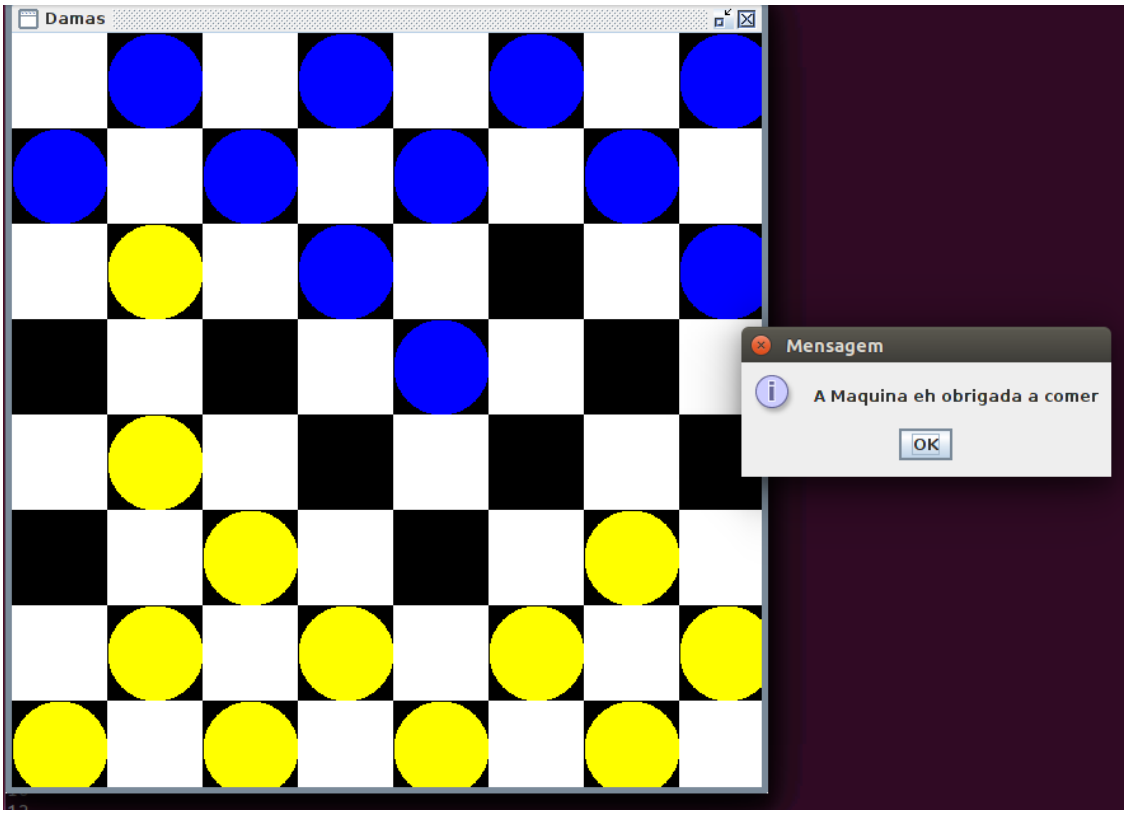

Figura 11. Execução da implementação de captura obrigatória pela máquina (peças azuis).

5-Conclusão

Nesse trabalho foi explicado e mostrado a extensão de um jogo de Damas implementado usando Prolog para as regras de negócio e Java para interface. O trabalho foi baseado em um código fonte do jogo e estendido com opções como: jogar contra a máquina e fazer da captura obrigatória.

Assim, esse trabalho se mostrou bastante positivo quanto ao aprendizado e consolidação de conceitos de Prolog visto em sala, pois fez com que a aluna pesquisasse e praticasse bastante na tentativa de implementação bem sucedida das propostas de extensão. Portanto, o conhecimento adquirido nesse projeto foi muito proveitoso e servirá de base para situações que surgirão adiante.

Para trabalhos futuros, pode-se implementar a estratégia de que no mesmo lance, se apresentar mais de um modo de capturar/comer, é obrigatório executar o lance que tome o maior número de peças (lei da maioria). Pois, do jeito que foi implementado a extensão do jogo só é permitido capturar uma peça adversária por vez.

6-Referências Bibliográficas

[1] WIKIPEDIA, **Prolog**. Disponível em:<https://pt.wikipedia.org/wiki/Prolog>. Acesso em 29 de Setembro de 2017.

[2] WIKIPEDIA, **Damas**. Disponível em:<https://pt.wikipedia.org/wiki/Damas>. Acesso em 29 de Setembro de 2017.

[3] DUSHIN, Fred. **JPL.** Disponível em <http://www.swi-prolog.org/packages/jpl/java_api/index.html>. Acesso em 29 de Setembro de 2017.