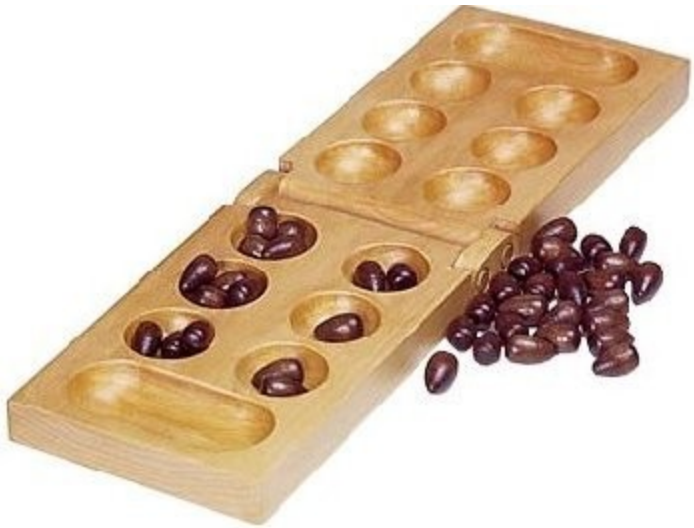
Universidade de Évora
Honesto Estudo com Longa Experiência Misturado
Departamento de Informática

PROGRAMAÇÃO DECLARATIVA



Ouri

TRABALHO ELABORADO POR:
Marlene Oliveira Nº 25999
Pedro Mateus Nº 26048
João Aiveca Nº 26175

ANO LECTIVO 2011 / 2012

Índice

Introdução

No âmbito da disciplina de Programação Declarativa foi-nos solicitada a elaboração, utilizando a linguagem Prolog, do jogo Ouri.

O Ouri é um jogo de tabuleiro, cujo objectivo é recolher mais sementes que o adversário. No Ouri todas as sementes possuem o mesmo valor.

No início do jogo cada jogador escolhe um lado do tabuleiro e é escolhido o primeiro jogador a efectuar uma jogada. Esta decisão é tomada após um dos jogadores esconder uma semente numa das mãos e o outro tentar adivinhar em qual das mãos do adversário está a semente escondida. Caso este adivinhe correctamente onde está escondida a semente, poderá ser ele a começar o jogo. Inicialmente, cada uma das doze casas do tabuleiro contém quatro sementes. O jogador irá então iniciar o jogo retirando todas as sementes de uma das suas casas e, avançando no sentido anti-horário, coloca-as uma a uma nas casas seguintes. Todas as jogadas que se seguem ocorrem do mesmo modo. Caso a casa de onde o jogador parte possua mais de doze sementes, este terá de distribuir as sementes de modo a dar uma volta ao tabuleiro, ignorando a casa de onde partiu. Enquanto existirem casas com mais do que uma semente, o jogador não pode mexer nas casas que possuam apenas uma semente. A última parte de uma jogada consiste na captura de sementes. Se na última casa onde foi depositada uma semente se encontrarem, contando já com a semente lá colocada, duas ou três sementes, o jogador pode e deve capturá-las. O processo repete-se tantas vezes quanto as casas anteriores à última onde foram colocadas sementes e que contenham duas ou três sementes que pertençam ao adversário até que o jogador encontre uma casa que não tenha nem duas nem três sementes e que não pertença ao adversário. O jogo passa para o adversário se na última casa onde é depositada uma semente se encontrem quatro ou mais sementes, que não poderão ser capturadas.

Se um dos jogadores ficar sem sementes quando efectua uma jogada, o adversário será obrigado a efectuar uma jogada em que sejam introduzidas sementes do lado do jogador que não tem sementes. Se for efectuada uma captura e o jogador adversário do que está a efectuar a jogada ficar sem sementes, o jogador que capturou sementes é obrigado a efectuar uma jogada de modo a que sejam introduzidas sementes nas casas do seu adversário. Se o jogador que efectuou a captura não tiver mais sementes o jogo termina.

Uma partida de Ouri termina quando um dos jogadores captura vinte e cinco ou mais sementes. A partida de Ouri pode ainda terminar se um jogador ficar sem sementes e o adversário não puder jogar de modo a introduzir sementes nas casas do mesmo. Neste caso, o adversário recolherá todas as sementes que estão nas suas casas e junta-las-á às que capturou durante o jogo, sendo o vencedor o jogador que possuir mais sementes. Se a partida estiver perto do seu término e restarem poucas sementes no tabuleiro, originando uma situação que se repetirá sem que qualquer um dos jogadores possa ou pretenda fazer algo para que esta não se repita, cada um dos jogadores recolherá as sementes que se encontram nas suas casas e junta-las-ão às sementes que capturaram durante o jogo. O vencedor será o jogador que possuir um maior número de sementes.

1. O Ouri

Antes de ser efectuada qualquer jogada, o programa irá apresentar aos jogadores um pequeno menu onde será possível escolher um modo de jogo dos que se encontram à disposição. Os jogadores podem assim decidir jogar um contra o outro, jogar contra o computador ou jogar uma variante do Ouri. Os jogadores podem ainda sair do Ouri.

O predicado *sair* imprime uma mensagem. Se, no menu inicial, o jogador seleccionar a opção "0 – Sair" inserindo o valor 0, o programa será terminado e *sair* executado. Exemplo do output gerado quando o jogador escolhe a opção que lhe permite sair do programa:

```
1 ?- go.

Ouri UE

0 - Sair
1 - Humano vs Humano
2 - Humano vs Computador
3 - Ouri especial
|: 0.
**O programa fechou/terminou**
true.
```

1.1 Humano Contra Humano

Se for escolhida a opção "Humano Vs Humano", será possível disputar uma partida de Ouri jogando à vez.

O predicado *jogarHumano* controlam o fluxo do jogo, ou seja, recebem a jogada a ser efectuada e verificam qual é o jogador que irá jogar de seguida, de acordo com os resultados das validações da jogada. Mesmo que as validações falhem, estes predicados passam a vez ao jogador seguinte desde que a jogada suceda. Estes predicados recebem como argumentos os tabuleiros de ambos os jogadores e as pontuações de ambos os jogadores, retornando um valor que indica qual será o jogador a jogar de seguida.

inicializarTabuleiro, tal como o nome indica, é um predicado que permite inicializar dois tabuleiros de jogo. Um tabuleiro de jogo é constituído por uma lista que possui um tamanho dado pela variável *Size* que, por predefinição, é igual a 6 (este valor é fornecido no predicado *menu*), cujas posições têm todas o valor quatro (número de peças em cada casa do tabuleiro no início do jogo) . Existem dois predicados inicializar tabuleiro.

O predicado seguinte garante que não existe nenhum tabuleiro que seja inicializado com o *Size* igual a zero:

```
inicializarTabuleiro(_, _, 0) :- !.
```

O predicado seguinte garante que o tabuleiro de tamanho *Size* é inicializado com quatro elementos em cada casa:

```
inicializarTabuleiro([Y|Ys], [Y|Ys], Size) :-
        S is Size-1,
        Y=4,
        inicializarTabuleiro(Ys, Ys, S).
```

Os predicados *printTabuleiroAux* e *printTabuleiroAux2* são auxiliares do predicado printTabuleiro que recebem como argumento uma lista que representa o tabuleiro de jogo e imprimem-na no output. *printTabuleiroAux2* irá imprimir o tabuleiro do jogador dois no output e *printTabuleiroAux* irá imprimir o tabuleiro do jogador um no output.

O predicado *printTabuleiro* irá imprimir no output o tabuleiro de jogo completo, bem com as pontuações de ambos os jogadores. Este predicado recebe como argumentos dois tabuleiros (o do jogador 1 e o do jogador 2) e duas variáveis que correspondem às pontuações dos jogadores.

O output resultante do uso do predicado *printTabuleiro* deverá ser semelhante ao seguinte:

```
Pontuação
Jogador 2: 0
Jogador 1: 0

Tabuleiro:
_____________________________________________________________
                        Jogador 2                            |
_____________________________________________________________|
   F   |   E   |   D   |   C   |   B   |   A   |
_______|_______|_______|_______|_______|_______|
 4         4         4         4         4         4
_____________________________________________________________
 4         4         4         4         4         4
_____________________________________________________________
   A   |   B   |   C   |   D   |   E   |   F   |
_______|_______|_______|_______|_______|_______|
                        Jogador 1                            |
_____________________________________________________________|
```

O predicado *valid* irá verificar se a jogada inserida pelo jogador corresponde ou não a uma jogada válida, ou seja, verifica se o jogador inseriu uma letra válida. Entendem-se por letras válidas as letras: *a, b, c, d, e, f*. Este predicado recebe como argumentos uma variável correspondente à jogada, dois tabuleiros correspondentes aos tabuleiros de cada um dos jogadores, duas variáveis correspondentes às pontuações de cada um dos jogadores e uma variável que indica qual é o jogador a efectuar a jogada actual. Se a jogada for válida, o programa continua a sua execução normalmente. Se a jogada for inválida, o programa escreve no output uma mensagem que indica que a jogada não é válida e volta a solicitar ao jogador que efectue uma jogada. Este processo repetir-se-á tantas vezes quantas as necessárias até que o jogador insira uma jogada que seja válida.

Exemplo de uma jogada inválida:

```
Pontuação
Jogador 2: 0
Jogador 1: 0

Tabuleiro:
_____________________________________________________________
                        Jogador 2                            |
_____________________________________________________________|
   F   |   E   |   D   |   C   |   B   |   A   |
_______|_______|_______|_______|_______|_______|
 4         4         5         5         5         5
_____________________________________________________________
 4         4         4         4         4         0
_____________________________________________________________
   A   |   B   |   C   |   D   |   E   |   F   |
_______|_______|_______|_______|_______|_______|
                        Jogador 1                            |
_____________________________________________________________|


Jogador 2 (a,b,c,d,e,f)?
|: n.

Jogada inválida!
```

Os factos *verPos* têm como principal função indicar a posição da lista que compõe o tabuleiro de um jogador a que corresponde a jogada efectuada.

Exemplo de um facto *verPos* utilizado:

```
verPos(a, 0).
```

Os predicados *verEsq* e *verDir* têm como principal função calcular o número de peças a incrementar nas posições que se encontram à esquerda e à direita da posição onde foi efectuada a jogada, respectivamente. Estes predicados recebem como argumentos uma variável que indica o número de peças que irão ser inseridas no tabuleiro, uma variável que indica a posição correspondente à jogada e retornam uma variável que corresponde ao número de peças a incrementar nas posições que se encontram à direita e à esquerda da posição onde foi efectuada a jogada.

O predicado *verSobras2* é um predicado auxiliar de *efectuarJogadaAux*, que permite verificar se existem peças suficientes no tabuleiro do oponente para que possa ser efectuada uma volta completa ao tabuleiro de jogo, ou seja, permite verificar se quando for efectuada uma volta completa ao tabuleiro do jogo é possível passar do tabuleiro do jogador actual para o tabuleiro do seu oponente.

Os predicados *jogCommit* permitem efectuar uma jogada. Estes predicados recebem como argumentos o número de peças à esquerda da casa onde se pretende jogar que serão alteradas, o número de peças à direita da casa onde se pretende jogar que serão alteradas, a posição onde foi efectuada a jogada, uma variável que corresponde ao contador do número de voltas completas ao tabuleiro que foram efectuadas, o tabuleiro em que está a ser efectuada a jogada e uma variável que indica qual o jogador que se encontra a jogar. O número de peças à esquerda e à direita da posição onde se pretende jogar é calculado antes do predicado ser invocado.

Se a lista que constitui o tabuleiro for completamente percorrida e não existirem mais peças a inserir, é retornada uma lista vazia:

```
jogCommit(0,0, _, _, [], [],1).
```

Se a lista que compõe o tabuleiro ainda não foi totalmente percorrida e o contador de voltas completas dadas ao tabuleiro for igual a zero, o predicado continuará a percorrer a lista, mesmo que não existam mais peças a inserir, e retornará o tabuleiro resultante de inserir as peças da jogada. Note-se que só existem alterações do primeiro elemento da lista que compõe o tabuleiro resultante se o contador de voltas completas dadas ao tabuleiro for diferente de zero, ou seja, se for dada uma volta completa ao tabuleiro:

```
jogCommit(0,0, -1, Multiplier, [X|Xs], [Y|Ys],1) :-
        Y is X+Multiplier,
        jogCommit(0,0,-1,Multiplier,Xs,Ys,1).
```

Quando todas as peças da esquerda já foram lidas e a posição actual corresponde à indicada pelo jogador é colocado o valor zero nessa posição e continuam a ser inseridas peças nas casas à direita da posição seleccionada pelo jogador:

```
jogCommit(0,NPecasDir, 0,  Multiplier, [_|Xs], [Y|Ys],1) :-
        Y is 0,
        Temp is NPecasDir,
        jogCommit(0,Temp, -1, Multiplier, Xs, Ys,1).
```

Este predicado permite também inserir sementes nas casas que se encontram à direita da posição indicada pelo jogador:

```
jogCommit(0,NPecasDir, -1,Multiplier, [X|Xs],[Y|Ys],1) :-
        Temp is NPecasDir-1,
        Y is X+Multiplier+1,
        jogCommit(0,Temp, -1,Multiplier,Xs,Ys,1).
```

Este predicado permite ainda procurar a casa onde foi efectuada a jogada. Só são alterados os valores correspondentes à posição actual e à cabeça da lista que constitui o resultado se for dada uma volta completa ao tabuleiro, ou seja, se o contador de voltas completas dadas ao tabuleiro for diferente de zero:

```
jogCommit(0, NPecasDir, Posicao, Multiplier, [X|Xs], [Y|Ys],1) :-
        T1 is Posicao-1,
        Y is X+Multiplier,
        jogCommit(0, NpecasDir,T1, Multiplier,Xs,Ys, 1).
```

Caso se verifique a existência de peças suficientes no campo do adversário para que possa ser efectuada uma volta completa ao tabuleiro depois da inserção de peças à direita, são adicionadas peças nas posições iniciais da lista que compõe o tabuleiro. O valor correspondentes ao número de peças no campo do adversário é calculado antes da chamada recursiva do predicado.

```
jogCommit(NPecasEsq, NPecasDir, Posicao, Multiplier, [X|Xs], [Y|Ys],1) :-
        T1 is Posicao-1,
        Temp is NPecasEsq-1,
        Y is X+Multiplier+1,
        jogCommit(Temp, NPecasDir,T1, Multiplier,Xs,Ys, 1).
```

Note-se que para o adversário só interessam as peças que sobram depois de serem inseridas sementes nas posições à direita da posição onde foi efectuada a jogada.

Não existirão então mais peças a serem incrementadas no tabuleiro do jogador actual.

Como o tabuleiro do adversário é escrito no sentido oposto ao do jogador, o predicado irá chamar recursivamente cada posição antes de ser incrementado qualquer valor correspondente ao número de sementes numa dada casa.

```
jogCommit(0, Pos, Multiplier, [X|Xs], [Y|Ys], 2) :-
        !,
        T is Pos-1,
        jogCommit(0, T, Multiplier, Xs, Ys,2),
        Y is X+Multiplier.

jogCommit(_, _, _, [], [],_).
```

Se existirem peças a serem jogadas, o predicado efectua a jogada.

```
jogCommit(NPecas, Posicao, Multiplier, [X|Xs], [Y|Ys],2) :-
        T1 is Posicao-1,
        T2 is NPecas -1,
        Y is X+Multiplier+1,
        jogCommit(T2,T1,Multiplier,Xs,Ys, 2).
```

Os predicados *validRules* garantem que o fluxo de jogo está de acordo com as regras básicas do Ouri. Os predicados *validRules* de aridade quatro recebem como argumentos um tabuleiro, o número de peças numa determinada posição, um valor que activa ou desactiva a regra a ser verificada e um valor que determina se é impresso no output a mensagem que indica que existem ainda casa com mais do que uma semente e retornam um valor correspondente à posição do tabuleiro que possui maior número de sementes.

Os predicados seguintes garantem que o jogador não pode jogar em casas que possuam uma semente enquanto existirem no seu tabuleiro casas com mais do que uma semente:

```
validRules([],1,_,Y,FlagWrites):-
        Y>1,
        !,
        (
        (   FlagWrites==0,!,
        write('Há casas com mais de 1 semente!')
        )
        ),
        fail.
```

Se o predicado enviado não unificar com o predicado anterior, garantidamente irá unificar com o predicado seguinte.

```
validRules([],_,_,_,_).
```

A única diferença entre as duos predicados anteriores e os predicados seguintes prende-se com o facto destas percorrerem a lista que compõe o tabuleiro.

```
validRules([X|Xs],NPecas,Flag,Y,FlagWrites):-
        X=<Y,
        validRules(Xs,NPecas,Flag,Y,FlagWrites).

validRules([X|Xs],NPecas,Flag,Y,FlagWrites):-
        X>Y,
        Temp is X,
        validRules(Xs,NPecas,Flag,Temp,FlagWrites).

validRules(Tab,NPecas, Flag,FlagWrites):-
        Flag==0,
        validRules(Tab, NPecas, Flag, 0,FlagWrites).

validRules(_,_, Flag,_):-
        Flag==1.
```

Os predicados *validRules0* verificam se o jogador está a tentar efectuar uma jogada numa posição onde já não se encontram sementes. Estes predicados recebem como argumentos uma lista que representa um tabuleiro, uma variável que indica a posição onde se pretende jogar e um valor que activa ou desactiva o predicado a ser verificada. Caso não existam sementes na posição indicada, o programa escreve uma mensagem no output.

Os predicados *validRulesSupp* garantem o cumprimento das regras suplementares do Ouri. Os predicados *validRulesSupp* de aridade três recebem como argumentos um tabuleiro e o valor que indica a posição onde será efectuada a jogada e retornam um valor que indica se a jogada é válida.

Estes predicados de aridade três verificam se existe uma jogada possível. Com aridade 7, com o primeiro argumento a 0, o tabuleiro oponente não está vazio, e a regra é desnecessária, aceitando automáticamente. Com outro valor, verifica-se se há uma jogada disponível (IsAvailable, chamando a função de aridade 3). Estando, valida a jogada do jogador com validate. Não estando, dá erro para o output, e falha o predicado. O output é ainda condicionado por FlagWrites, sendo apenas escrito com este a 0; com este a 1, o output é nulo.

Os predicados *verifyZeros* permitem verificar se o oponente já não tem nenhuma semente no seu tabuleiro (se o tabuleiro do oponente se encontra preenchido com zeros). Caso o oponente possua sementes no seu tabuleiro, o programa activa os predicados que implementam as regras suplementares do Ouri. Estes predicados recebem como argumento um tabuleiro e retornam um valor que indica se o tabuleiro se encontra ou não vazio (preenchido com zeros). O valor retornado tomará o valor 1 caso o tabuleiro do oponente esteja preenchido com zeros.

O predicado *sumPontos* calcula o número de sementes que ainda se encontram no tabuleiro. Este predicado recebe como argumentos uma lista com o número de sementes em cada casa do tabuleiro e retorna o número total de sementes que ainda se encontram no mesmo.

Os predicados *gameOver* permitem verificar qual dos dois jogadores é o vencedor ou se existe um empate. Quando for determinado o vencedor, o programa escreve para o output uma mensagem que indica qual dos jogadores é o vencedor e a pontuação obtida pelo mesmo. Em caso de empate, o programa escreve para o output uma mensagem que indica que o jogo terminou empatado.

O predicado *efectuarJogadaAux* de aridade oito é, tal como o nome indica, um predicado auxiliar do predicado *efectuarJogada*. Este predicado auxiliar permite efectuar uma jogada, calculando os valores correspondentes ao número de peças a incrementar à esquerda e à direita da posição onde foi efectuada a jogada, gerando também o tabuleiro resultante de tais alterações. Este predicado recebe como argumentos os tabuleiros de cada um dos jogadores, o número de peças numa dada posição, a posição da jogada a ser efectuada (que irá ser ignorada) e a pontuação actual e retorna o tabuleiro do jogador 1 resultante das alterações, o tabuleiro do jogador 2 resultante das alterações e retornará a pontuação resultante de efectuar a jogada pretendida.

Os predicados *efectuarJogadaAux* de aridade três têm como função encontrar o número de peças que constam no tabuleiro na posição indicada pelo jogador. Recebe como argumentos uma lista que representa um tabuleiro e uma variável que indica a posição em que irá ser efectuada a jogada e retornarão o número de peças que se encontram na posição em que irá ser efectuada a jogada.

Os predicados *efectuarJogada* de aridade onze têm como principal função efectuar e validar uma jogada efectuada, verificando o cumprimento dos predicados básicos e, caso se aplique, dos predicados suplementares do Ouri, bem como a validade da posição onde se pretende jogar. Os predicados *efectuarJogada* recebem como argumentos os tabuleiros de ambos os jogadores, uma jogada, um valor que indica qual dos jogadores está a jogar neste momento (0 corresponde ao jogador 1 e 1 corresponde ao jogador 2) e as pontuações actuais de ambos os jogadores. Estes predicados retornarão dois tabuleiros correspondentes, respectivamente, aos tabuleiros do jogador 1 e do jogador 2 depois de efectuada a jogada, as pontuações de ambos os jogadores e um valor que indica se a vez será ou não passada para o jogador adversário.

Os predicados *validate* garantem que, quando o oponente não possui peças no seu tabuleiro, o jogador é obrigado a efectuar uma jogada que coloque sementes no campo do adversário, falhando caso o jogador não cumpra esta regra. Os predicados *validate* recebem como argumentos o número de peças e a posição onde se pretende efectuar a jogada e um valor que determina se é impresso no output a mensagem.

Os predicados *chagelist* são auxiliares dos predicados captura e permitem percorrer a lista que constituí o tabuleiro e altera o valor que se encontra na posição dada. Tais predicados recebem como argumentos o tabuleiro actual, a posição onde foi colocada a última peça e o valor a colocar na posição onde foi inserida a última peça. Os predicados *changelist* retornam o tabuleiro alterado.

Os predicados *captura* permitem efectuar uma captura de sementes, ou seja, se na última

casa onde são depositadas sementes se encontrarem duas ou três sementes que pertençam ao adversário, estas são capturadas e adicionadas às que o jogador actual já capturou. Este processo repetir-se-á enquanto nas casas anteriores existirem duas ou três sementes do adversário. Os predicados *captura* recebem como argumentos o tabuleiro actual, a posição onde foi depositada a última semente e o número de pontos do jogador activo, sendo retornado o tabuleiro resultante depois de serem efectuadas todas as capturas possíveis e a pontuação resultante da captura. Caso não exista nenhuma captura válida, o tabuleiro é retornado igual.

captura_valida verifica se existem duas ou três peças na casa actual.

1.2 Humano Contra Computador

O AI que implementámos tem várias fases.

O objectivo é simular todas as casas do tabuleiro. A primeira jogada válida disponível, na ordem a>b>c>d>e>f, é guardada, tal como a primeira jogada válida que cause uma captura favorável ao computador. Das duas, é escolhida a captura se existir, e a jogada válida caso contrário.

Dada a natureza da implementação, distinguimos o primeiro turno saltando a simulação. Já que não pode ocorrer uma captura com o tabuleiro inicializado, é inútil simular.

Como as regras garantem que nunca se recebem tabuleiros inválidos, estas simulações cobrem todas as situações disponíveis.

Os predicados *jogarAI* têm como função controlar o fluxo de jogo contra o computador.

Os predicados *jogarAIFirstTurn* só são utilizadas no primeiro turno do segundo modo de jogo e têm como principal funcionalidade permitir ao computador iniciar na casa correspondente à letra B (por predefinição). Dado que com o tabuleiro inicial é impossível ocorrer uma captura, é desnecessária a simulação.

Os predicados *simularAI* de aridade oito permitem simular a jogada do computador. Estes predicados recebem como argumentos os tabuleiros de ambos os jogadores, as pontuações de ambos os jogadores, uma variável que indica em que posição do tabuleiro se pretende efectuar a jogada e retorna a pontuação nova após a jogada efectuada.

Os predicados *simPlay* permitem ao computador calcular qual é a melhor jogada a ser efectuada. Estes predicados recebem como argumentos a nova pontuação após a jogada do computador simulada, a pontuação anterior à jogada do computador, a casa simulada, e devolvem a jogada com captura se existir, ou a jogada válida sem captura.

A diferença na validação da jogada prende-se com o facto de ser invocada o predicado *jogarAI* ao invés do predicado *jogarHumano* que era invocada no predicado que efectua a validação para as partidas que utilizam os predicados originais.

1.3 Variante do Ouri

A variante escolhida para esta parte do trabalho é muito semelhante à versão original, porém nesta versão o vencedor é o jogador que possuir menos peças.

Esta variante do jogo utiliza as mesmos predicados que a versão original, à excepção dos predicados que determinam quem é o vencedor e que validam a jogada efectuada.

A diferença nos predicados que determinam o vencedor do jogo consiste apenas no facto da pontuação que identifica o vencedor ser a menor possível, ou seja, o jogador que capturar menos sementes vence o desafio.

A diferença na validação da jogada prende-se com o facto de ser invocado o predicado jogarVariante ao invés do predicado *jogarHumano* que era invocada no predicado que efectua a validação para as partidas que utilizam os predicados originais.

O predicado *jogarVariante* possui um comportamento semelhante ao predicado *jogarHumano*.