Implementação em Prolog do Jogo Oware

Relatório Final

Universidade do Porto

Faculdade de Engenharia

FEUP

Mestrado Integrado em Engenharia Informática e Computação

Programação em Lógica

Grupo 35:
André Freitas - ei10036
Rui Gonçalves - ei10100

Faculdade de Engenharia da Universidade do Porto
Rua Roberto Frias, sn, 4200-465 Porto, Portugal

4 de Novembro de 2012

Resumo

Este relatório tem o objetivo de descrever a implementação do Jogo Oware numa linguagem de programação lógica em matemática que é o Prolog. O jogo em questão é de tabuleiro, jogando-se com sementes, sendo muito popular na República do Gana. Apesar da simplicidade das regras tem um forte componente estratégico.

O jogo Oware foi implementado com recurso a listas para representar o tabuleiro e para representar as propriedades dos jogadores, estando organizando em módulos de manipulação do tabuleiro, rotinas de jogo, inteligência artificial, interfaces e predicados de teste. Existe ainda a possibilidade de jogar contra o computador com 2 níveis diferentes. Com isto, conseguiu-se uma interface de texto, não muito rica, mas adequada tendo em conta o contexto deste trabalho.

Índice

1 Introdução

Pretende-se explorar as capacidades do Prolog para representar um jogo e as suas regras através deste trabalho. Um jogo é uma excelente maneira de enriquecer o conhecimento na representação e estruturação dos dados bem como a aplicação de regras do jogo traduzidas nesta linguagem.

A implementação em questão é em modo de texto, não sendo muito refinada no sentido da apresentação mas sim na componente das funcionalidades.

Neste documento pretende-se apresentar a descrição do problema, a representação dos estados do jogo em estruturas conhecidas em Prolog, a representação das jogadas, a visualização do tabuleiro, a inteligência artificial e os aspetos de desenvolvimento do projeto.

2 Descrição do Problema

O Oware é dos jogos de tabuleiro mais antigos do mundo, tendo sido inventado há mais de 7 mil anos. É jogado por todo o Globo e não existem certezas relativamente à sua origem, porém, atribui-se a sua autoria tradicionalmente ao continente Africano. Atualmente é o jogo mais popular na República do Gana sendo um fenómeno nacional.

Figura 1: Pessoas jogando tradicionalmente o Oware

Como se pode constatar pela Figura 1, existe um tabuleiro com 2x6 cavidades onde se colocam sementes ou feijões. Existem muitas interpretações das regras deste jogo, pelo que iremos adotar apenas a que é mais conhecida. Assim, o jogo começa com 4 sementes em cada buraco. Os jogadores jogam alternadamente, e em cada jogada tira-se as sementes de um buraco da nossa linha de jogo e vai-se distribuindo as sementes no sentido anti-horário.

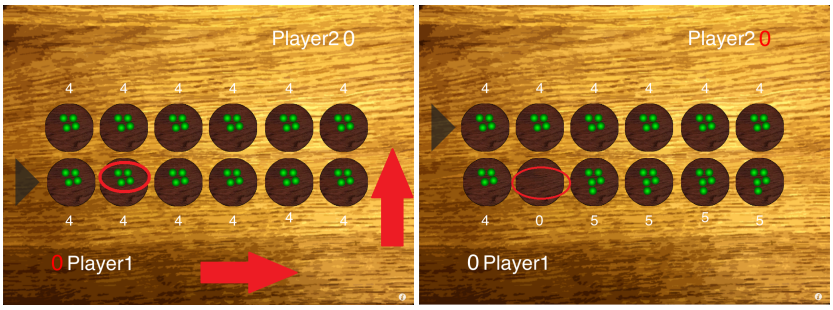
Figura 2: Distribuição das sementes

Quando distribuirmos a última semente e se colocarmos num buraco do adversário e esse sítio ficar com 2 ou 3 sementes, capturamos essas sementes. O jogo termina quando um jogador capturar 25 sementes ou ambos os dois jogadores capturarem 24 sementes (empate).

2.1 Ilustração de jogadas

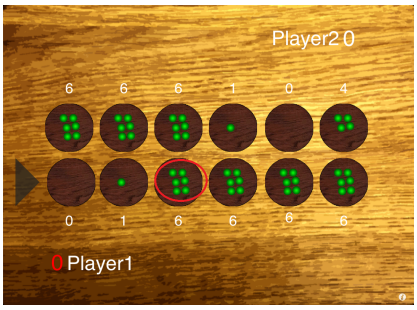
Nas seguintes ilustrações serão descritas duas situações de jogos fundamentais. Pede-se especial atenção para as ilustrações, dado que é necessário estar atento às situações que descrevem para perceber o jogo.

Nesta primeira situação o jogo está a começar e o jogador vai distribuir a segunda cavidade de sementes. Assim, as sementes ficaram distribuídas e não houve captura. De notar novamente que o sentido da distribuição das sementes é anti-horário e que o jogador só pode retirar as sementes da sua linha.

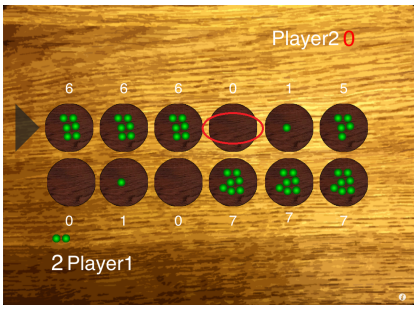


(a) Estado Inicial do Jogo (b) Jogada sem Captura

A outra jogada importante é a da captura de sementes, ou seja, quando um jogador ao distribuir a última semente calha numa linha do seu adversário, ficam 2 ou 3 sementes nesse sítio, capturando-as. Atenção que se a meio da distribuição das sementes conseguir este número nas cavidades, não as pode capturar, por isso é só na semente final. Destaca-se esta situação dado que existem várias interpretações relativas a esta regra.



(c) Antes da Captura



(d) Após a Captura de Sementes

Pode-se constatar assim a simplicidade do jogo, mas apesar disso requer um componente estratégico para que se possa ganhar, tentando prever-se uma panóplia de jogadas possíveis para vencer o adversário e maximizar as nossas capturas.

3 Arquitectura do Sistema

Ora, como referido anteriormente, o sistema é decomposto em módulos o que permite um melhor isolamento no desenvolvimento e manutenção futura do código. Assim sendo, os módulos são os seguintes:

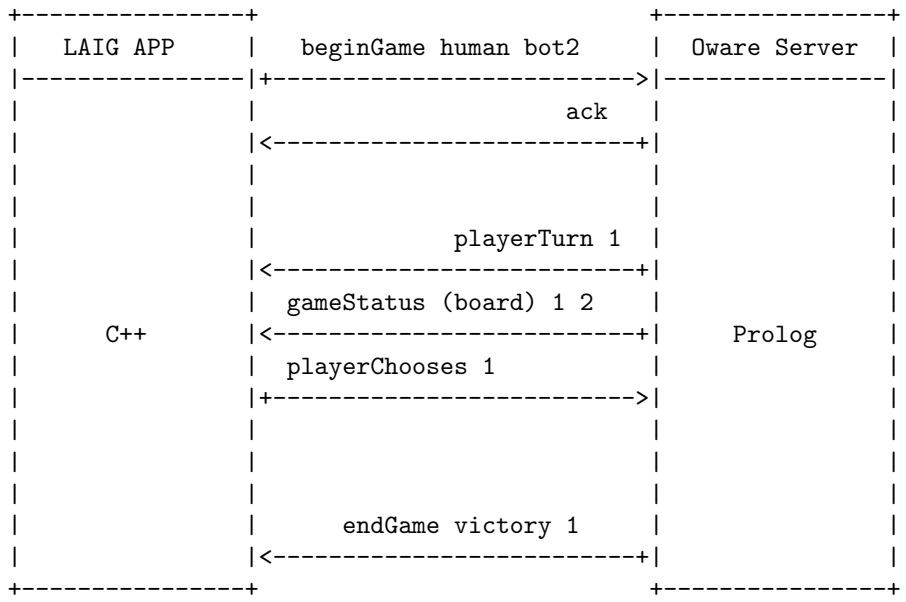
1. oware.pl - possui todos os predicados inerentes à rotina de jogo;
2. owareBoard.pl - contém os predicados para manipular e aceder à estrutura do tabuleiro;
3. owareCLI.pl - alberga os predicados que interagem na consola de texto com o utilizador;
4. owareAI.pl - módulo que contém os predicados para os bots;
5. owareTest.pl - módulo que possui testes considerados pertinentes;

Destaca-se a atenção na análise do código dos predicados gameRoutine, aiPlay e playSeeds, que são o ponto de partida para a compreensão da implementação do jogo.

Para começar o jogo, basta fazer uma consulta ao ficheiro oware.pl e invocar o predicado startGame que recebe 2 parâmetros, que são os tipos do jogador 1 e 2, que podem ser human, bot1 ou bot2. Assim sendo, se pretende um jogo entre um humano e um bot de nível 2, será startGame(human,bot2).

3.1 Protocolo de mensagens com o visualizador

Para a implementação com o visualizador 3D, será criado um módulo que terá o nome "owareServer.pl", usando a comunicação por socket do Sicstus com o seguinte protocolo de mensagens:



Assim sendo, a sintaxe das mensagens, é a seguinte:

1. beginGame (player1Type) (player2Type) - indica a informação para iniciar o jogo;
2. ack - confirma o início do jogo;
3. playerTurn (1 or 2) - indica qual é o jogador que vai jogar;
4. gameStatus (board) (player1 score) (player2 score) - indica o estado do jogo;
5. playerChoose (1-6) - indica qual a posição escolhida pelo jogador;
6. endGame (victory (1 or 2)) or (draw) - sinaliza o fim do jogo.

Dado que as mensagens que são passadas entre os sistema são relativamente simples, este formato é suficiente, porém, se se tratasse de um sistema que necessitasse de algo mais complexo, faria todo o sentido usar um formato para representar estruturas de dados como JSON, pois do lado da aplicação de LAIG em C++ existem bibliotecas para tal.

4 Módulo de Lógica do Jogo

Como descrito anteriormente, o projecto foi implementado em módulos para as diferentes camadas do jogo Oware.

4.1 Representação do Estado do Jogo

Ora, todos os jogos possuem o ciclo, sendo o jogo representado em cada iteração por estruturas conhecidas. Assim, existe a lista que representa o tabuleiro e cada jogador tem uma lista que possui o tipo de jogador(human, bot1,bot2) e a sua pontuação. Um exemplo possível de um estado de jogo é o seguinte:

```
gameRoutine([[1,0,7,0,0,0],[0,4,2,1,1,0]],[human,20],[bot1,12],1).
```

4.2 Visualização do Estado do Jogo

Ora o estado do jogo é visualizado pelo utilizador em modo de texto ou pelo visualizador CGI. O estado do jogo é apresentado pelo predicado:

```
printBoard(Board,Player1Score,Player2Score).
```

4.3 Validação de Jogadas

É necessário validar o input das posições quando se trata de um jogador Humano, pelo que simplesmente garante-se que é fornecida uma posição de 1 a 6. O predicado readUserInput pode ser encontrado no ficheiro owareCLI.pl.

```
readUserInput(Pos):-
(read(Pos),Pos >0,Pos <7);
write(':( Invalid Position!. Insert again:'),
readUserInput(Pos).
```

4.4 Execução de Jogadas

Ora o predicado mais importante é claramente o da execução de jogadas, sendo o playSeeds que recebe 5 parâmetros que são o tabuleiro, número do jogador, índice de 0 a 5, novo tabuleiro e pontuação da jogada. O índice começa em 0 e não em 1 por ser mais ortodoxo no acesso às listas. De notar que o jogador 1 joga na primeira lista e o 2 na segunda lista. Um exemplo de uma chamada poderia ser:

```
playSeeds([[1,0,7,0,0,0],[0,4,2,1,1,0]],1,2,[[2,1,0,0,0,0],[1,5,0,0,0,0]],7).
```

4.5 Jogadas Válidas

Ora, o jogador pode jogar todas as posições da sua linha do tabuleiro, exceptuando quando a posição escolhida não ter sementes. Se tal acontecer, dado que existe um predicado para obter as sementes de uma posição, o programa simplesmente pede de novo uma posição ao jogador. Assim sendo, não se torna útil neste tipo de jogo de tabuleiro, com uma restrição muito simples, criar um predicado que a partir de um tabuleiro devolva as jogadas válidas. Acresce ainda o facto de que, se na linha do tabuleiro do jogador em questão, todas as posições não tiverem sementes, o jogo cede a vez ao próximo jogador até que existe pelo menos 1 semente para jogar.

4.6 Final do Jogo

O jogo acaba quando um dos jogadores chegar aos 25 pontos ou ambos os jogadores empatarem por 24 pontos. Esta restrição é garantida no predicado gameRoutine, sendo o código correspondente a estas situações o seguinte:

```
gameRoutine(_,Player1,Player2,_):-
Player1=[_,P1Score],
Player2=[_,P2Score],
P1Score=24,P1Score=P2Score,
write('You Both Win!').

gameRoutine(_,Player1,Player2,_):-
Player1=[_,P1Score],
Player2=[_,P2Score],
(
(P1Score>=25),
write('\nPlayer 1 Wins!');

(P2Score>=25),
write('\nPlayer 2 Wins!')
).
```

4.7 Cálculo da Jogada do Computador

Os bots têm 2 níveis de dificuldade, sendo o bot1 um jogador que escolhe posições aleatoriamente e o bot2 um jogador com capacidade de decidir qual a jogada que lhe irá dar mais pontos, aplicando uma estratégia gananciosa (greedy algorithms). Foi colocado um sleep de um segundo quando o bot joga para que o utilizador possa aperceber-se da sua presença facilmente.

De notar que o jogo em si, poderá levar um tempo considerável a acabar, dado que chega-se a um ponto em que ambos os jogadores impedem o adversário de pontuar, ou seja, esta nuance também se reflecte nos bots que, ao jogararem entre si, podem levar imenso tempo a finalizar o jogo.

A invocação do bot é feita pelo aiPlay que tem como parâmetros o número do jogador, o tabuleiro, a posição e o tipo de bot (bot1 ou bot2).

```
aiPlay(PlayerNum,Board,Pos,BotType):-
% If it's level 1 bot
BotType = bot1,
stupidBot(PlayerNum,Board,Pos);

% If it's level 2 bot and score
BotType = bot2,
(
aiTryAll(Board, PlayerNum, ScoreList,0),
scoreListGetPos(ScoreList,1,Pos,MaxScore),
MaxScore>0
);
%write('Going random...\n'),
aiPlay(PlayerNum,Board,Pos,bot1).
```

4.8 Mensagens com o visualizador

A integração da troca de mensagems com o utilizador será feita na secção onde se imprime o tabuleiro e onde se obtem a posição das sementes por parte do jogador. No nível mais alto da implementação será o predicado readCGIUserInput(Pos), que irá recorrer ao predicados de troca de mensagen: sendMessage(Message) e readMessage(Message). Estas funcionalidades serão implementadas em conjunto com o terceiro prático de LAIG, pelo será esse o momento oportuno para o seu desenvolvimento.

5 Interface com o Utilizador

A interface com o utilizador é relativamente simples, usando-se a linha de comandos do interpretador Prolog.

A cada iteração do jogo o tabuleiro é apresentado, pedindo-se ao utilizador que escreve a posição que quer jogar, ou se se tratar do bot, imprime-se a sua escolha. Para começar o jogo basta invocar o predicado startGame/2.

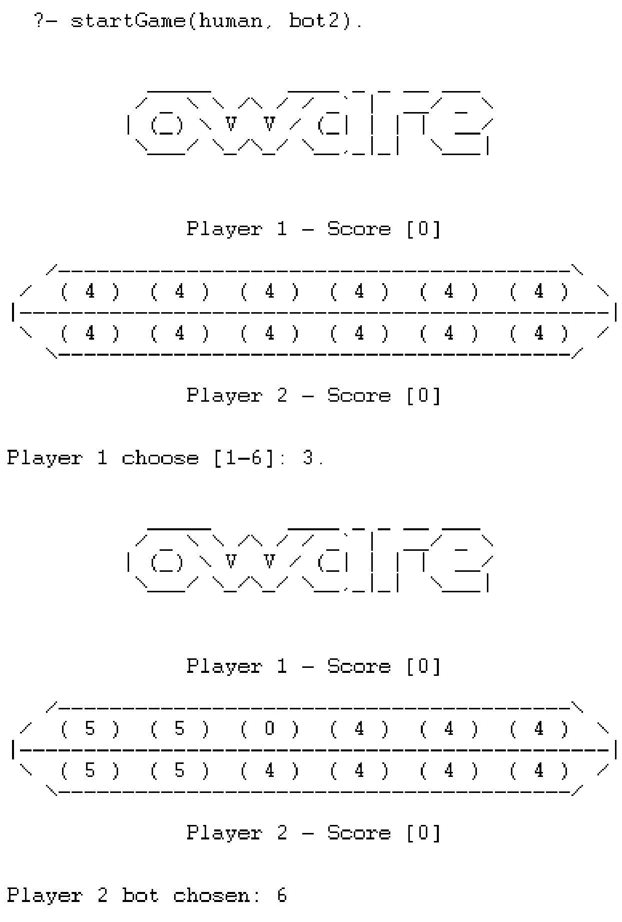


Figura 3: Exemplo de uma visualização do jogo

6 Conclusões e Perspetivas de Desenvolvimento

Para um programador comum que está habituado a linguagens funcionais, no primeiro contato com o Prolog é normal sentir-se um bocado limitado devido ao paradigma que está habituado a ter no desenvolvimento de soluções. Porém, após um período, consegue aperceber-se da potencialidade que esta linguagem tem, especialmente em computar soluções.

O desenvolvimento do Oware em Prolog teve um ritmo estável, apesar de que se teve obviamente momentos em que foram alocados períodos de tempos consideráveis para despitar bugs que numa linguagem funcional seriam triviais. Apesar disso, cumpriram-se os objetivos inicialmente propostos, o que se traduz num desafio superado com sucesso.

Ora reforça-se que esta implementação se enquadra no contexto da Unidade Curricular de programação em lógica e admite-se que com o Prolog consegue-se mais facilmente computar soluções, que envolvam processos heurísticos e recursivos. Porém, existe uma perda enorme na legibilidade do código visto que as instruções que habitualmente descrevem um corpo de uma função, são substituídas por uma série de predicados, sendo necessário interpretar todos os parâmetros que esses predicados contêm, tornando mais difícil a compreensão das decisões tomadas.

Os interpretadores SWI Prolog e Sicstus, foram usadas para correr o nosso código, porém, carecem de um bom editor de texto, pelo que a melhor solução encontrada foi o Sublime Text 2 que possui a sintaxe de Prolog, tornando o desenvolvimento muito mais produtivo. Outro problema encontrado é que estes em certos aspectos, não são uniformes entre si relativamente ao Standard do Prolog. Possivelmente estes problemas serão concerteza resolvidos no futuro com a motivação da comunidade que desenvolve em Prolog.

Consideramos que fizemos uma implementação eficiente do Oware e, dado que o nosso esquema de tabuleiro é simples, foi fácil representá-lo em modo de texto, pois para um jogo mais complexo seria concerteza uma barreira a ultrapassar. O desenvolvimento de testes nos aspectos críticos do Jogo garantem que entregamos um produto final. Pode-se sugerir no futuro uma abordagem de testes mais exaustiva mas para projectos que tenham um grau de complexidade superior.

7 Bibliografia

Referências

[1] The Oware Society. 2010. *Oware- Played all over the World*. Acedido a 4 de Outubro de 2012. http://www.oware.org.

[2] Awale.jpg. 2006. *A game of awale*. Acedido a 4 de Outubro de 2012. http://upload.wikimedia.org/wikipedia/commons/1/14/Awale.jpg.

[3] oware.jpg. *Playing Oware in Ghana*. Acedido a 4 de Outubro de 2012. http://exploringafrica.matrix.msu.edu/teachers/events/oware.jpg.

[4] Easy Oware 2012. *Play the classic strategy game from Africa*. Acedido a 2 de Outubro de 2012. http://itunes.apple.com/br/app/easy-oware/id408219960?mt=8.

[5] SICStus Prolog. *SICStus Prolog - Random Number Generator*. Acedido a 1 de Novembro de 2012. http://www.sics.se/sicstus/docs/3.7.1/html/sicstus_23.html.

A Código Implementado