Universidade de Brasília

Instituto de Ciências Exatas

Departamento de Ciência da Computação

Estrutura de Dados - Turma "A"

Relatório - Trabalho 2

Nome:
Renato Nobre
Khalil Carsten

Matrícula:
15/0146698
15/0132662

03 de Março de 2016

1 Introdução

Mancala é denominada à uma categoria de jogos com sua evidencia de criação no século 6 e 7 *Anno Domini*, havendo evidencias de sua criação em Eritreia e Etiópia. O Mancala apresenta claramente, similaridades com a agricultura, e a ausência de equipamentos que podem ser evidencia do inicio da civilização em si [4]. Uma versão moderna do jogo é o Kalah, usualmente jogado nos Estados Unidos e na Europa, onde há a confusão entre as nomenclaturas do jogo. Como outros jogos de tabuleiro, o mancala já serviu de diversos estudos, tanto psicológicos, como na ciência da computação [5].

O Kalah consiste de um tabuleiro de 14 cavidades, sendo 2 maiores denominadas Kahalas, e outras 12 cavidades menores, sendo 6 para cada jogador. A Kahala de cada jogador fica ao lado direito das cavidades menores, e o número de sementes define a pontuação do jogador.

O estado inicial do jogo consiste em 4 sementes em cada cavidade dos jogadores, uma jogada é realizada quando o jogador escolhe uma das seis cavidades, retira todas as suas sementes e as distribui uma para cada cavidade no sentido anti-horário. A Kahala do jogador deve ser semeada também, entanto a Kahala do adversário deve ser ignorada. O jogo termina quando não há mais sementes em algum dos lados do tabuleiro. O vencedor é aquele que no final do jogo tiver mais sementes em sua Kahala. Regras adicionais são descritas detalhadamente dentro do programa.

Em estrutura de dados, uma árvore é uma forma de organização hierárquica. Contendo, nós, raiz, ramos, folhas, e mais diversas terminologias para sua classificação. Tal estrutura é amplamente utilizada em classificações e tomada de decisões, como por exemplo no desenvolvimento de uma inteligência artificial básica para um jogo simples de tabuleiro.

Uma árvore usualmente utilizada para esse propósito é a árvore genética, *Game Tree*. Tal estrutura de árvore é montada de maneira a representar as possibilidades de jogadas de um jogador, a partir do estado do jogo. Os filhos de um nó representam todos os estados de jogada a partir da situação atual do jogo. Tal árvore pode ser criada com uma heurística para cada nó, e com uma função de avaliação *Minimax*, retornar a melhor jogada.

2 Implementação

Com o propósito de entender por total o funcionamento e regras do jogo, um tempo foi gasto realizando seu estudo. Para isso foi utilizado uma versão online do jogo, a mesma versão nos forneceu as regras, a quais foram utilizadas na seção *Regras*, do programa [3].

O código é feito de maneira a tentar maximizar a separação da lógica da interface, contendo diversas funções para mostrar as mensagens na tela, disponibilizadas no começo do código, *tabuleiro*, *menu*, *menu_ dific*. Tais funções servem respectivamente para mostrar o tabuleiro na tela, mostrar o menu principal e o menu de regras, e mostrar o menu após o modo *Player vs IA* ser escolhido. Há também uma função *popular*, que preenche o tabuleiro com os valores iniciais do jogo.

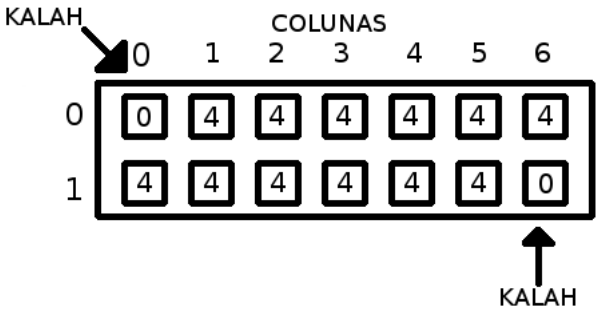

Figura 1

O jogo foi primeiramente implementado em uma versão jogador contra jogador, *Player vs Player*, para verificar se a lógica implementada para o funcionamento estaria correta e concisa, dirimindo erros desnecessários na hora de implementar a árvore. Ao implementar o tabuleiro do jogo, foi decidido utilizar uma matriz 2 por 7, o que na visão dos desenvolvedores, facilita o entendimento, devido a sua disposição espacial ser equivalente ao do tabuleiro.

Para realizar as manipulações do jogador foram criadas funções que controlam a lógica para ambos os jogadores, estas são, *turno_p1*, *referencia*, *turno_p2*, *referencia2*. Tais funções possuem uma lógica espelhada, sendo sua lógica geral praticamente imutável, isso decorre do fato do tabuleiro possuir dois lados de lógicas opostas, porém similares. *referencia*, são funções criadas para adaptar a jogada, passada pelo usuário como uma letra, à lógica matricial, um número. Foi decidido o uso de letras para escolher a cavidade a ser jogada, para prevenir confusões entre a escolha da cavidade e a quantidade de sementes na mesma.

Para o processo de finalização do jogo foram criadas outras duas funções, *m_vazia*, *final*. Estas respectivamente, checa se algum dos lados dos jogadores esta vazio, e distribui as sementes restantes caso o jogo finalizado e mostra a mensagem de vitória ou derrota.

Após todo esse processo para o funcionamento trivial do jogo ser criado, começou-se o desenvolvimento da Inteligência Artificial, que para facilitar o desenvolvimento da idéia será explicado em três partes, o processo de criação do nó, o processo de gerar a árvore e o retorno da jogada a ser feita.

A estrutura básica do nó, foi recebendo parametros novos de acordo com a necessidade, no final do projeto obtivemos um nó com *mat_estado*, *no *filhos*, *heuristica*, *player*, *jogada*. *mat_estado* é a matriz do estado atual do jogo, *no *filhos*, é um vetor de ponteiros de nós, após bastante debate, foi definido que esse seria o jeito mais prático e eficiente de definir os filhos, pois trabalhariamos apenas com as quantidades necessárias. *heuristica*, é o valor da jogada, *player*, quem está jogando no momento, jogador 1 ou jogador 2, e *jogada* define qual a jogada foi feita para gerar o mat_estado. Para realizar o preenchimento dos nós, foi feita a função *criaNo*.

Para implementar a função geradora da árvore implementamos uma função recursiva *geraArvore* que recebe um nó previamente criado para servir de raiz de origem para a geração de outros nós. Incialmente definimos um critério de parada para o *loop* recursivo, para isso utilizamos a variável *dificuldade*. Ao entrar na função checa-se se a dificuldade é maior que zero, se sim, então prossegue-se com a criação de nós e para cada chamada da função *dificuldade-1* é passado. Após passar por esse teste o algoritmo procede para um loop do tipo *for* que testará através das possibilidades de jogadas, encontradas na matriz, se é possível criar um nó para aquele *filho*, ou seja, se a casa no tabuleiro for diferente de

zero, então cria-se um filho respectivo a posição daquela casa. Assim passamos a matriz do tabuleiro atual para uma *mat_aux* para que possamos simular para cada nó um jogada usando as funções *turno_p1* ou *turno_p2*. Para saber qual das funções anteriores utilizar para simular a jogada criamos um marcador global com o nome de *flag* que assim sendo 1 ou 2 saberíamos qual delas utilizar. O último detalhe é a atribuição da heurística para cada nó. Toda vez que se cria um nó analisa-se a jogada anterior e a atual e subtrai-se resultando assim, na heurística.

A função *minimax* foi a última a ser implementada e passou por diversas mudanças. Incialmente é importante esclarecer que ela retorna um ponteiro do tipo nó. Ao entrar na função cada nó passa por um *loop* do tipo *for* onde se checa se o nó é uma folha ou não. Isso faz parte do primeiro teste da função, onde se testa se a *dificuldade*, ou seja, altura da árvore, é maior que zero e se o nó é uma folha. Caso passe no teste o procedimento levará para um outro *loop* que irá passar por todos os nós e seus respectivos filhos aplicando a lógica da função *Min* e *Max*. Para isso foram criados duas funções auxiliares chamadas de *Max* que retorna o maior dos valores e o *Min* que retorna o menor dos valores. Ao final do processo o valor de retorno é um nó que possui a melhor heurística.

Encerrando a lógica do modo *Player vs IA* após a a escolha do nó com melhor heurística este representa um filho dentre os 6 criados para cada elemento, e como cada filho representa uma possibilidade de jogada automaticamente detectamos qual será a seleção de jogada e a introduzimos na função *turno_p2*. Por devir o *loop* de chamar a *geraArvore*, *minmax* e selecionar a jogada se repete até o encerramento do jogo.

3 Conclusão

Árvore de jogos é uma alternativa viável para implementação de uma inteligência artificial básica, utilizando a possibilidade de calcular jogadas futuras para tentar prever as jogadas do inimigo e planejar suas próprias.

Durante o desenvolvimento do trabalho foram encontrados vários problemas de implementação. Um dos motivos principais para tais problemas foi a possibilidade do jogador repetir sua jogada, gerando inconsistências na árvores e funções de avaliação.

Referências

[1] Árvores de Jogos, https://en.wikipedia.org/wiki/Game_tree

[2] Prof. Eduardo Alchieri, Estrutura de Dados, Slides, Árvores, http://cic.unb.br/∼alchieri/disciplinas/graduacao/ed/arvores.pdf

[3] Jogo Mancala Online, http://play-mancala.com

[4] Mancala, https://en.wikipedia.org/wiki/Mancala

[5] Gobet, F. (2009). “Using a cognitive architecture for addressing the question of cognitive universals in cross-cultural psychology: The example of awalé”. Journal of Cross-Cultural Psychology 40 (4): 627–648. doi:10.1177/0022022109335186