Relatório - Trabalho 2
Mancala

Thiago Veras Machado, 16/0146682
Vitor Fernandes Dullens, 16/0148260

[1]Dep. Ciência da Computação – Universidade de Brasília (UnB)
CiC 116319 - Estrutura de Dados - Turma A

{thiago.veras.machado,vitordullens2.0}@gmail.com

1. Introdução

1.1. Descrição

O trabalho consiste em um famoso jogo, mais popular na região da África e Ásia, onde pode ser considerado o xadrez do Ocidente, seu nome é Mancala. O jogo se baseia em turnos, onde cada jogador faz uma jogada, com um principal obetivo, o de capturar as peças adversárias e coloca-las em suas Kalah, quando um jogador não possui mais peças o jogo acaba, e o jogador que possuir mais peças em suas Kalah é o vencedor. Existem outras regras adcionais que serão explicadas ao longo do relatório.

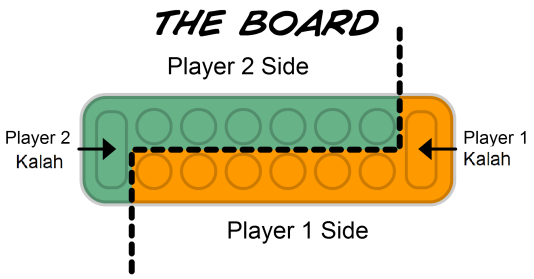

Figura 1. Tabuleiro do jogo Mancala

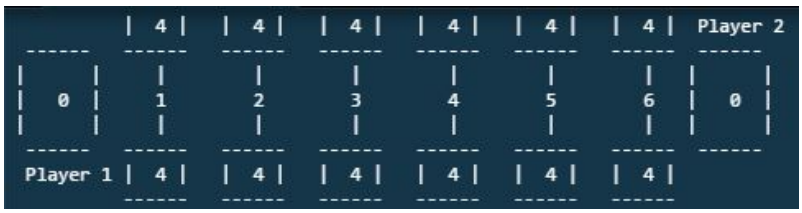

Figura 2. Nosso tabuleiro do jogo Mancala, feito com caractéres

1.2. Visão Geral

O programa, primeiramente, lê qual a opção de jogo o usuário quer jogar, sendo elas, multijogador, ou contra uma inteligencia artificial, que possui 4 dificuldades. Após feito isso, é mostrado pra o usuário o tabuleiro do jogo, feito com apenas carácteres. Com

isso, o(s) usuário(s) deve colocar qual casa deseja fazer a jogada e as peças vão andando no sentido anti-horário, conforme os padrões do tabuleiro, mostrado na Figura 1, e as regras do jogo.

A inteligência artificial do jogo avalia a maior pontuação possível e a jogada que o usuário faz a menor pontuação, feito isso avalia-se qual é a melhor jogada a se fazer. Quanto maior a dificuldade, mais jogadas ele vai ”prever”tornando ele mais preciso em sua decisões de qual casa escolher.

2. Implementação

2.1. Estrutura de Dados

Foi utilizado vários tipo de estruturas de dados, como vetores, principalmente o que armaze os dados do tabuleiro. Porém os mais complexo e detalhado foi o implementado na parte da inteligencia artificial do jogo, mais precisamente na função `int ai(int qnt, int turno)`, onde foi implementado uma espécie de Game tree.

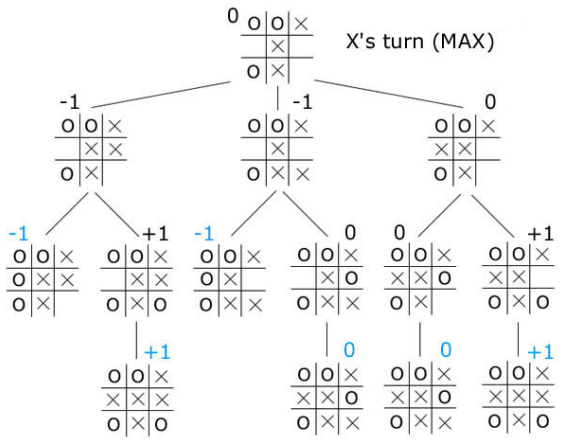

Figura 3. Game tree

Como a Figura 2 mostra, a Game Tree é uma árvore que armazena o maximo de pontos que a inteligencia pode fazer, baseando-se também, no minimo de pontos que o oponente consegue fazer, prevendo as proximas jogadadas e sabendo qual ela deve realizar. Com isso, a dificuldade dp jogo é basicamente baseada na altura dessa árvore, quanto mais previsões o computador fizer, mais ele vai selecionar a melhor posição para jogar.

Em nosso programa, utilizamos como base a ideia da Game tree. Fazemos com que o programa faça as jogadas em um clone do tabuleiro, fazendo todas as recursões necessárias e calculando a mlehor joga a se fazer. Feito isso, ele armazena a posição de melhor pontuação a partir do algoritmo de `min-max` (armazena a melhor potuação do jogador e a pior pontuação do adversário). Ele faz isso diversas vezes dependendo da dificuldade escolhida, para, com isso, conseguir maior precisão na jogada a ser feita.

A precião aumenta a cada recursão feita, especificamente 5 recursões no modo mais fácil e 11 no mais dificil, sendo que, com 11 recursões o computador ja leva um tempo a cada turno, calculando a melhor jogada a se fazer, porém o jogo fica bem desafiador, examente como foi proposto pela dificuldade.

2.2. Principais Funções

`void start()`: A função menu imprime na tela todas as possibilidades que o jogo permite, como multijogador ou contra a inteligencia artificial. O jogador pode escolher qual a dificuldade que deseja executar o jogo, afim de uma melhor experiência no game. Como é mostrado na figura abaixo

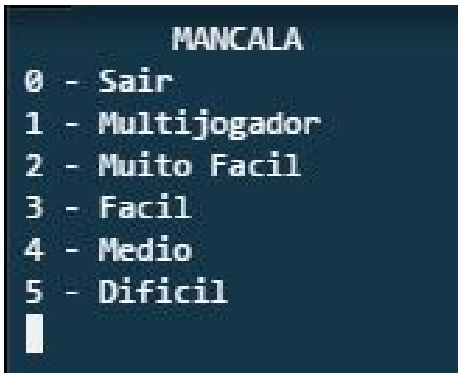

Figura 4. Menu do nosso jogo, com as opções que o usuário pode escolher

`void game(int modo)`: Esta função chama a função `initGame()` no qual zera todas as casa e kalahs para começar o jogo. O jogo funciona em volta do `while` principal, no qual verifica se há pelo menos uma pedra em uma casa do campo do jogador, se não houver o jogo é finalizado e é chamado a função `resultado()`.

`void initGame()`: Essa função é muito básica, mas essencial, ela é chamada ao iniciar o jogo na função `game(modo)`, ela é responsável por deixar as duas kalahs vazias e todas as casas com o valor 4, conforme as regras do jogo.

`void table()`: Função responsável por limpar a tela e mostrar o tabuleiro, ela é chamada ao inicio de cada novo turno, atualizando o tabuleiro para o usuário de um forma que não fique muito aparente, pois ele usa o comando `CLEAR` que limpa a tela.

`int pick(int turno)`: Essa função é responsável por ler qual a casa o usuario deseja jogar, ele verifica, também, se a casa escolhida é um número válido (entre 1 e 6), se não o usuário deve digitar novamente uma casa. A partir dessa função é chamada a função `play(casa, kalah, begin)`.

`int play(int casa, int kalah, int begin)`: Esta é a função mais importante de todo o programa, pois ela que gera as jogadas feitas pelo(s) usuário(s) ou pela inteligencia artificial. A função recebe a casa que ele deseja jogar, a kalah do usuario e onde começa a sua área do tabuleiro. Essa função também verifica se deve ser jogado novamente, conforme as regras do jogo. Seu retorno se baseia no jogador que deve jogar, sendo 1 o próprio jogador(repetir a jogada) e 0 o outro jogador/inteligencia artificial.

`void resultado()`: Função que é chamada ao finalizar o loop principal na função `game(modo)`, ela é reponsável por pegar as peças que sobraram no tabuleiro e coloca-las na kalah de quem as possui. Feito isso ele verifica qual player fez mais ponto. Ao final ele mostra qual foi o resultado da partida, sendo possível ela terminar empatada.

`int ai()`: A função mais complexa de todo o trabalho, possui um vetor temporário que recebe os dados do tabuleiro antes de realizar todas as possíveis jogadas (pelo

for de 7 a 13), uma variável ”pts”que significa o melhor, ou pior placar de acordo com a suposta jogada. A variável ”player”recebe a possível jogada, caso seja possível efetuá-la (caso tenha pelo menos 1 pedra dentro da casa), se após a jogada a última pedra não cair na sua kahala, a função retornará 1 e efetuará recursivamente a função ai() com uma profundidade maior, sem mudar o jogador, caso contrário ele apenas troca o jogador, caso a jogada resulte em uma pontuação maior que a variável pts, ele a atualiza (no primeiro caso sempre será verdade pois setamos a variável pts com um valor pequeno) Se na casa nao tiver pedra, a variável que contabiliza quantidade de zeros será incrementada, para ter controle se acabou a partida ou não. Qnt é a profundidae da árvore, no qual quando qtd = 0 significa que ele foi o mais profundo possivel e voltou tudo para retornar qual a melhor posição. Se a variável que contabiliza os zeros for = 6, significa que acabou o jogo e deve retornar o resultado daquela jogada. A segunda pare é uma simulação de todas as possíveis jogadas do player real, fazendo com que pegue a pior jogada possível. Pois o computador supõe que após ele fazer uma jogada o jogador real irá fazer a jogada que mais atrapalha o computador, então o computador simula as jogadas do jogador real pra que ele escolha a jogada em que a melhor resposta do jogador seja a pior possível.

3. Estudo de Complexidade

1. `int play()`: Esta função possui a complexidade linear O(n), com n = quantidade de pedras na posição, pois a função apenas vai distribuindo as pedras para os demais buracos.
2. `int pick()`: Esta função possui a complexidade linear 0(n), pois ela somente le uma jogada e e chama a função play() que possui complexidade O(n)
3. `void resultado()`: Esta função possui a complexidade constante 0(6), pois possui 2 laços separados que executam 6 operações cada , depois imprime na tela a tabela e qual dos *players* venceu.
4. `void start()`: Esta função possui a complexidade constante O(1) pois apenas lê do teclado a opção e começa o game.
5. `void game()`: Esta função possui a complexidade O(n.m), pois a função funciona enquanto é possivel ter uma jogada e a cada vez ela chama a função pick que possui complexidade O(n), como dito acima.
6. `int ai()`: Esta função possui a complexidade O(n.m.6), com n = quantidade de recursões, de acordo com a dificuldade do game e m = quantidade de pedras em uma possível jogada na casa e 6 devido a 6 possiveis diferentes jogadas.
7. `int initGame()`: Esta função possui a complexidade constante O(1), pois apenas acessa as posições no tabuleiro para zerar as kalahs e colocar as pedras nas casas para começar o game.
8. `void table()`: esta função possui a complexidade constante O(1), pois apenas imprime na tela o tabuleiro com a quantidade de pedra em cada casa, ele é chamado a cada repetição do loop principal.

4. Testes Executados

- Se a ultima peça do jogador cair na própria Kalah, ele deverá jogar novamente;
- Conforme selecionado, o jogo pode ser jogado com dois jogadores humanos;
- Se a ultima peça do jogador cair em uma casa vazia (do mesmo), ele ”roubará”as peças do lado oposto do adversário;

- A dificuldade do jogo corresponde ao que foi selecionado pelo jogador (quantidade de recursões);
- Na parte do menu do jogo, função `start()`, cada item selecionado deve corresponder ao que o usuário deseja fazer de forma correta;
- O tabuleiro deve ser atualizado corretamente, sendo facilmente viualizado pelo usuário;
- Manter o programa rodando a uma velocidade aceitável, conforme o esperado para um jogo;

5. Conclusão

5.1. Comentários Gerais

Podemos concluir que, com esse trabalho, pudemos colocar em prática o desenvolvimento de um jogo mais complexo, com várias regras e exeções a serem consideradas na sua implementação. Além disso, essa foi nossa primeira experiência com a criação de um tipo de inteligência artificial, algo que é muito interessante, já que, praticamente tudo que envolve tecnologia, ultimamente, possui algum tipo de inteligencia.

Com isso, foi uma ótima experiencia, que nos trouxe uma ampliação no conhecimento no desenvolvimento de algoritmos capazes de obedecer várias regras(como a de um jogo compolexo como Mancala), além de nos mostrar como é possível fazer um computador ”pensar”e agir da melhor maneira possível.

5.2. Principais Dificuldades

Na visão da dupla, a parte mais complexa do trabalho foi, sem dúvida, a implementação da inteligencia artificial, função `int ai(int qnt, int turno)`. Pois, como nunca haviamos utilizado algo parecido, foi preciso um maior estudo sobre como a inteligência artificial é aplicada em um jogo, para depois conseguir implementar da melhor maneira possível uma inteligencia que correspondesse as expectativas de dificuldade que o jogo sugere.

Após estudos sobre como é a base para a inteligência artifical, notamos que as vezes, ultrapassava um tempo razoável de espera para que uma jogada aconteça, devido a um grande número de testes que ”a inteligência artificial executava”, pois o nosso programa consiste em uma simulação de todas as possíveis jogadas do computador, fazendo com que a melhor jogada do computador resulte em um jogada que influencie o oponente, tornando sua melhor jogada a pior possível, para isso, inúmeros cálculos e funções são chamadas. O que nos levou a uma otimização do código, diminuindo o número de testes executados.