Relatório ED
Trabalho 2: Game Trees

João Pedro Batista Ribeiro Costa, 15/0080123
Lucas da Silva Moutinho, 15/0015747

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

{jpbrc1,lucas.silvamoutinho}@gmail.com

1. Introdução

Neste projeto fora realizado a elaboração de um jogo de mancala em ambiente computacional, utilizando-se a linguagem de programação *C*, onde o jogador enfrenta uma inteligência artificial, ou IA. A inteligência artificial do computador funciona por meio de uma árvore de altura 4, 5, 6 ou 8, de acordo com a dificuldade. Nesta, a partir das possíveis decisões futuras do oponente e da própria IA, a inteligência artificial escolhe qual jogada minimiza suas perdas máximas, ou seja, utiliza um algoritmo minimax para a decisão de sua jogada. A lógica envolvida é bem simples, a decisão é tomada de forma que o oponente não tenha possibilidade de ter uma vantagem significativa em relação ao computador. Para isso o algoritmo funciona analisando as possibilidades de posições no tabuleiro das folhas da árvore, selecionando assim o caminho que dificulta a vitória do oponente, minimizando a maior perda possível.

1.1. Objetivos

A partir da implementação da mancala, esta que utiliza de algoritmos de verificação, remoção, adição, busca, etc, em árvores. O trabalho proporciona um aprofundamento em estrutura de dados, principalmente no que diz respeito a implementação de árvores. Além disso a concepção do projeto como um todo, possibilita trabalhar com um código mais robusto, aumentando assim a habilidade e o conhecimento de programação.

2. Implementação

Para a sintetização do algoritmo que implementa o jogo Mancala e uma IA (inteligência artificial) básica para jogar contra o usuário, foram desenvolvidas 5 funções principais para tal, cada uma com seu conjunto de funções auxiliares, estas que são: a função *decisaoComputador()*, responsável por decidir a posição que a IA deve selecionar em sua rodada, ou seja, por retornar o número da posição de acordo com o algoritmo Minimax, a função *criaArvore()*, que, por sua vez, é responsável pela criação da *Gametree*, uma estrutura de árvore utilizada em teoria de jogos onde os nós armazenam os possíveis estados do jogo a partir do estado atual, a função *montaJogada()*, que é responsável por retornar o estado do tabuleiro após escolhida determinada posição para a jogada , a função *miniMax()* e sua auxiliar *avalia()* que ficam incubidas pela decisão de escolha da IA por meio da análise da árvore *Gametree* utilizando o algoritmo Minimax, este que minimiza a maior perda possível, a partir do resultado da avaliação dos tabuleiros nas folhas da árvore e, por fim, a função *organizaRodada()* que, a partir de todas as funções anteriores estabelece a organização de uma rodada do jogo Mancala.

Todas estas serão aprofundadas em suas respectivas seções quanto a estrutura de dados utilizada e possíveis procedimentos de relevância aplicados, porém, ressalta-se que antes da discussão as funções que ditam o funcionamento do jogo e de sua IA, será discutido sobre a estrutura de dados utilizada a partir de um parágrafo sobre a *Struct no*, discutindo sobre a como a árvore *Gametree* fora implementada.

2.1. A *Struct no*

O Struct no é a estrutura que define o formato que cada nó que nossa árvore apresenta. Como pode-se ver pelo código, primeiramente é criado uma matriz 2x7 do tipo inteiro, que chamamos de *tabuleiro*, esta que guarda quantas sementes tem em cada posição do tabuleiro e quantas estão na kahala. Logo em seguida tem-se o inteiro *jogador*, que serve para identificar se é a vez do jogador ou do computador, por convenção definimos que quando *jogador* recebe 1 é a vez do computador e quando recebe 2 é o usuário. Por fim, cada nó guarda 6 endereços do tipo nó, o número 6 representando o número máximo de jogadas possíveis para cada jogador em sua respectiva vez. Estes endereços são necessários para que assim possamos criar a estrutura básica de uma árvore.

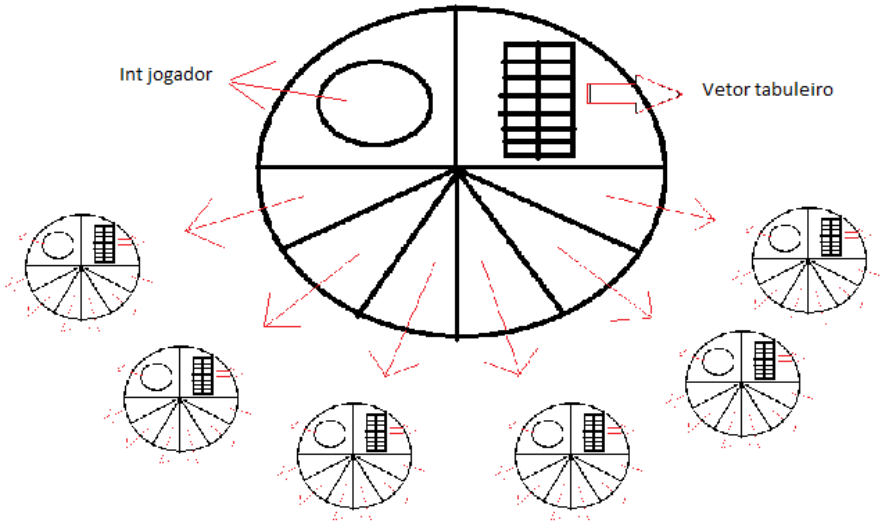

Figure 1. *Struct no*

2.2. A Função *decisaoComputador()*

A Função *decisaoComputador()* é encarregada de retornar a posição à qual a IA deve escolher em sua jogada por meio da análise da árvore *Gametree* utilizando do algortimo Minimax. Sendo assim, além de retornar a posição referente a jogada da IA, esta também é responsável por chamar a função para criar a árvore *Gametree* a partir do estado atual do tabuleiro no jogo.

A função *decisaoComputador()*, primeiramente, chama a função *criaArvore()* para gerar uma *Gametree* com os possíveis tabuleiros a partir do estado atual do tabuleiro no jogo. A altura desta árvore, ou quantos estados à frente serão gerados na *Gametree*,

é determinado a partir da escolha da dificuldade do jogo. Tendo obtido a árvore, esta é então analisada para todas as 6 arestas a partir da raiz, chamando a função *miniMax* para cada um dos 6 filhos da raiz. Cada um dos nós apresentará um retorno referente ao algoritmo Minimax, estes que serão discutidos em próximas seções. A partir destes retornos, a função *decisaoComputador()* irá selecionar qual dos filhos obteve o maior retorno por meio do algoritmo Minimax, pois deseja-se maximizar a escolha da IA em relação ao estado atual do jogo, ou seja, sendo cada um dos filhos um tabuleiro resultante de uma das posições escolhidas para a jogada, o que apresentar o maior retorno por meio do algoritmo Minimax representa a posição que maximiza a jogada da IA. Assim, a partir do que obtiver maior retorno, será retornado pela função *decisaoComputador()* a posição referente àquela jogada que maximiza os ganhos.

Ressalta-se que discussões sobre como o algortimo Minimax decide a melhor jogada e como se dá o processo de criação da árvore serão discutidos nas próximas seções nas respectivas funções responsáveis por tais tarefas, esta focando-se, exclusivamente, na explicação de como é decidido a posição escolhida na jogada da IA.

Ao final da expressão, a árvore é liberdada, pois tendo a IA já obtido a posição à qual deve escolher para sua jogada atual, a análise da árvore está encerrada.

2.3. A Função *criaArvore()*

A Função *criaArvore()* tem por incubência criar a árvore *Gametree*, esta que contém todos os possíveis tabuleiros a partir do tabuleiro atual e repete a lógica até obter todos os tabuleiros possíveis em uma certa quantidade de jogadas à frente do tabuleiro atual. Tal árvore é criada dinamicamente utilizando a função *malloc()*, e armazena os dados explicitados na *Struct no*. Os dados armazenados nesta árvore são utilizados tanto pela função *decisaoComputador()* quanto pela função *miniMax()*.

Para criar tal árvore, a função *criaArvore()*, primeiramente, recebe o tabuleiro atual e faz com que cada um dos ponteiros deste tabuleiro apontem para outros nós contendo o estado do tabuleiro após escolhida uma das seis posições possíveis para a jogada. Cada um dos ponteiros da árvore atual recebe um nó criado dinamicamente com a função *malloc()*, contendo o tabuleiro gerado a partir da função *montaJogada()*, esta que, por sua vez, recebe o tabueleiro atual e a posição a ser escolhida e retorna um tabuleiro com o estado resultante da jogada a partir da posição escolhida. O processo de geração da árvore e seus filhos contendo as possíveis jogadas repete-se até determinada altura, esta específica de cada dificuldade do jogo. Quanto maior a altura, mais jogadas à frente são armazenadas na árvore. Certos filhos da árvore, entretanto, podem conter ponteiros que apontam para NULL sem estarem no nível mais alto desta. Tais ponteiros em NULL são correspondentes as posições inválidas para a jogada dado o tabuleiro contido no nó pai. No caso do jogo Mancala, tais jogadas inválidas são aquelas em que não existem sementes em determinada posição do tabuleiro.

Após gerada a árvore com uma determinada quantidade de jogadas à frente, a função *criaArvore()* retorna a raiz, esta que contém o tabuleiro em seu estado atual de jogo, para a função *decisaoComputador()*, tendo, assim, a *Gametree* à ser analisada, mais tarde, pela função *miniMax()*.

Uma imagem para ilustrar o processo de criação da *Gametree* fora desenhada e pode ser visulizada abaixo.

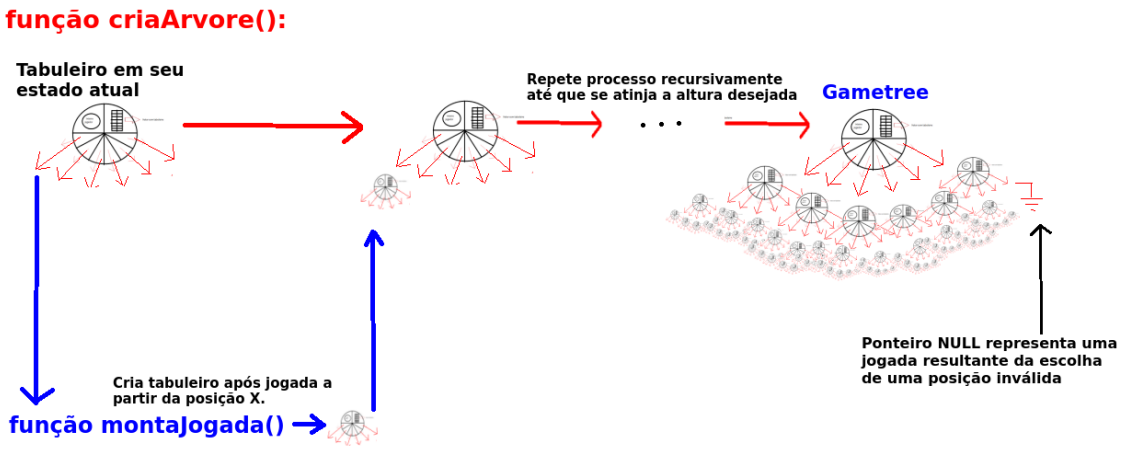

Figure 2. ***Processo de criação da Gametree***

2.4. A Função *montaJogada()*

A função *montaJogada()* é responsável por movimentar as sementes todas as rodadas e decidir para onde elas deverão ir, alem de determinar de quem é a vez, recebendo o tabuleiro em seu estado atual e retornando outro tabuleiro, com o resultado da movimentação das sementes a partir de uma determinada posição. A função *montaJogada()* primeiramente, a partir da posição selecionada, tanto pela IA ou pelo usuário, irá verificar quantas sementes tem na posição e, de acordo com o tanto de sementesuma destas colocará de um em um, cada uma destas na posição seguinte, até que estas se acabem. Quando as sementes acabarem, caso a última semente seja colocada na kahala, o jogador da vez tem o direito de jogar novamente, caso contrário a vez é passada para o adversário. Outro caso que ocorre é quando a última semente para em uma posição que não tem mais nenhuma semente, mas a posição simetricamente oposta do oponente tem alguma semente, neste caso todas as sementes do oponente e a a última semente do jogador, são mandadas para a kahala do jogador que acabou de semear, vale ressaltar que nesse caso o jogador não repete a vez.

Outro caso que a função também executa é quando o jogo está para finalizar, em que todas as sementes de um jogador acabam, ou seja todas as posições estão com zero sementes em um dos lados do tabuleiro, neste caso, todas as sementes que ainda estiverem nas posições do adversário devem ir para sua respectiva kahala. Por fim, a função *montaJogada()* também ao longo de sua execução, manda todos as novas posições que vão sendo feitas para um novo tabuleiro, este que por sua vez é criado logo no começo da função e que irá conter o jogo ”atualizado”, ao final da função. A função *montaJogada()* irá retornar exatamente este novo tabuleiro quando for chamada. O tabuleiro atualizado será utilizado tanto pela função *criaArvore()* para preencher os ponteiros dos filhos de cada nó quanto para alterar o tabuleiro atual do jogo em si. A seguir seguem algumas imagens do tabuleiro em seu estado atual juntamente com o estado seguinte:

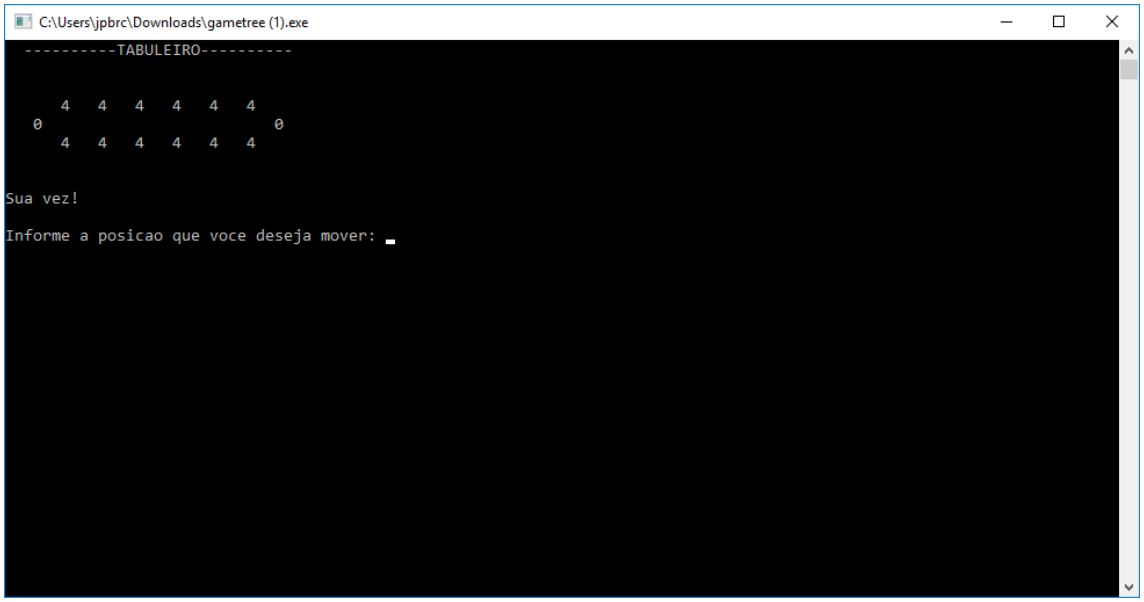

Figure 3. Posição inicial do tabuleiro

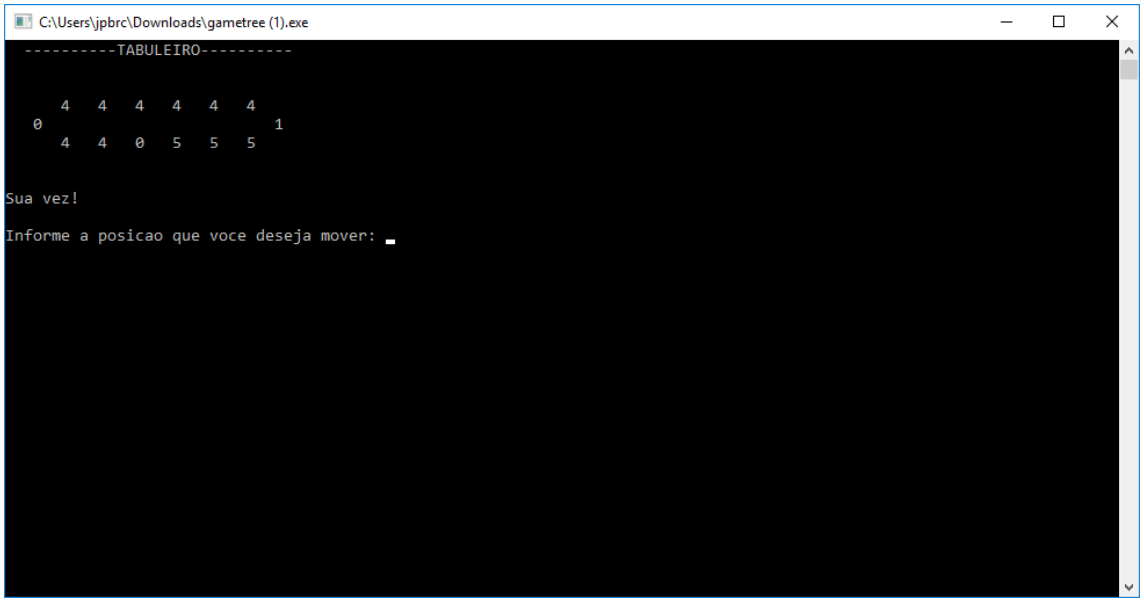

Figure 4. Posição do tabuleiro apos o usuario jogar na posição 3, em que repetirá a jogada

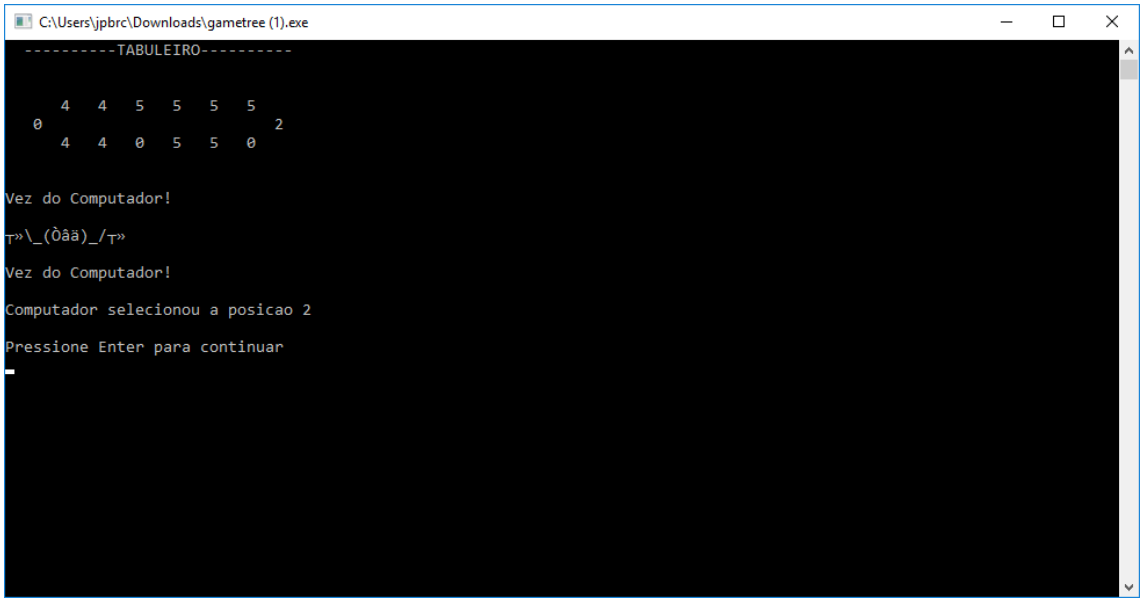

Figure 5. Posição após o usuário selecionar a posição 6 que somente passa a vez para o computador

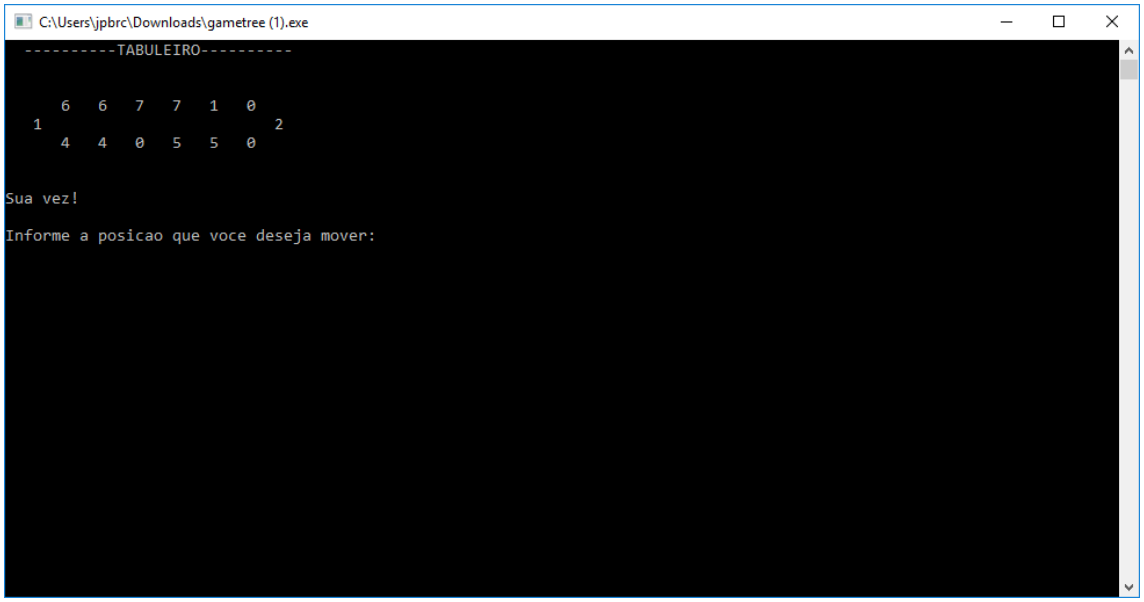

Figure 6. Posição após computador jogar na posição 2, tendo assim o direito de repetir a jogada

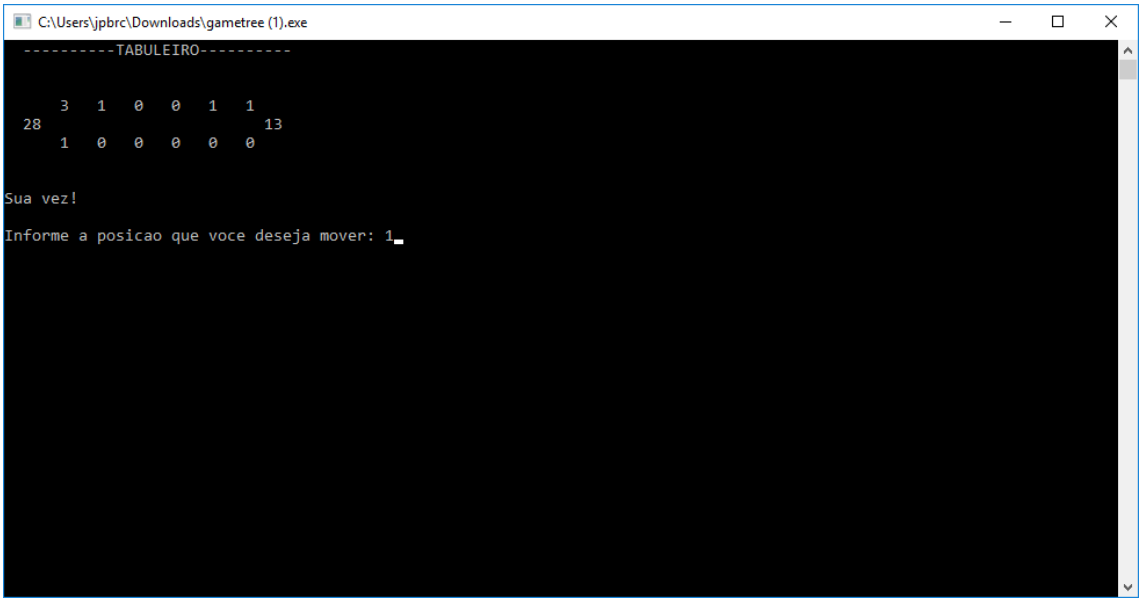

Figure 7. Posição antes do fim da partida, que será finalizada quando o usuário selecionar a primeira posição

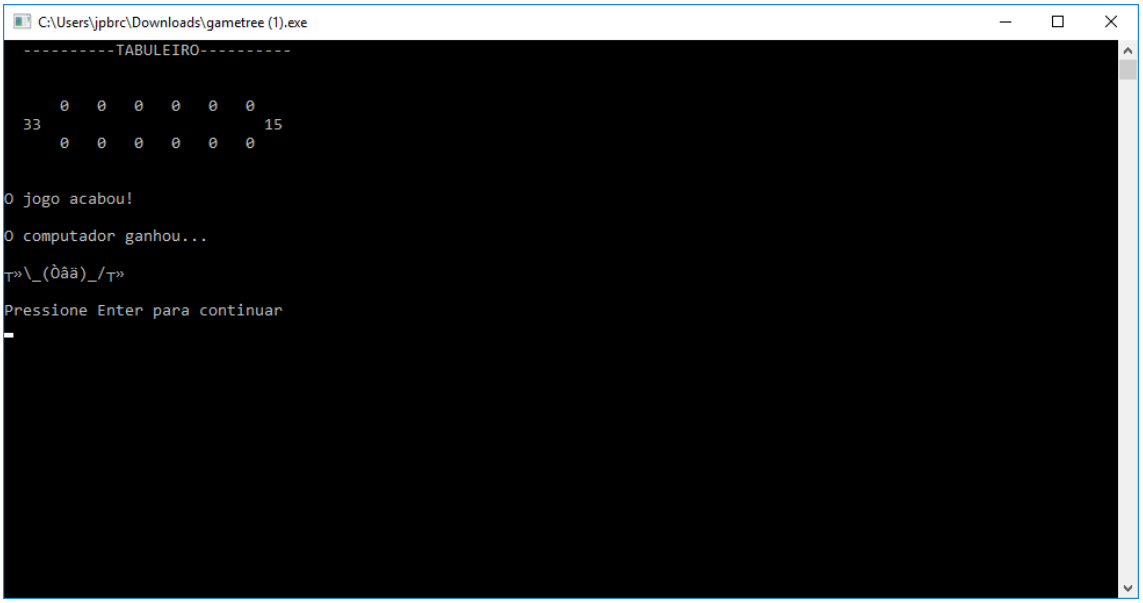

Figure 8. Tabuleiro ao fim da partida, em que após o usuário selecionar a última peça na única posição ainda possível, todas as peças do adversário vão para kahala do adversário

2.5. As Funções *miniMax()* e *avalia()*

A função *miniMax()* é a que é responsável por implementar o processo de decisão de escolha da IA para determinada jogada. Para entender tal função é necessária, previamente, discutir sobre o algoritmo Minimax.

Em teoria de decisão e em teoria de jogos, o algoritmo Minimax caracteriza-se de um método para minimizar a maior perda possível. Em jogos tradicionais, onde há

alternância entre os jogadores à cada jogada, o Minimax analisa todos os possíveis estados de jogo há uma determinada quantidade à frente do estado atual e retorna valores de acordo com o jogador da vez, retornando o menor valor representativo do estado do jogo caso a escolha seja do adversário, ou seja, minimizando suas escolhas, e retornando o maior valor representativo do estado do jogo caso a escolha seja a do jogador, maximizando suas escolhas.

Translocando tal algoritmo para o ambiente computacional, utiliza-se de uma *Gametree* para armazenar os estados possíveis do jogo à frente do estado atual. O conjunto das folhas da árvore são todas as possibilidades em uma quantidade específica à frente do estado atual. Deve-se então avaliar o estado do jogo para todas as folhas, retornando um valor que representa o quão vantajoso aquela situação é para o jogador ou para o adversário. A função responsável por avaliar as folhas da *Gametree* é a auxiliar *avalia()*.

A função *avalia()*, para o jogo Mancala, retorna, em geral, a diferença entre as Kahalas da IA e do usuário. A diferença será positiva caso a IA esteja ganhando e negativa caso o usuário esteja ganhando. Assim, os valores positivos mais altos serão escolhidos para maximizar as escolhas e os negativos mais baixos para minimizar as escolhas. Caso o estado do jogo em determinada folha represente o final do jogo, analisa-se qual dos dois ganhou. Caso a IA tenha ganhado, retorna-se o valor 100 mais a diferença das kahalas e, caso o usuário tenha ganho, retorna-se o valor -100 mais a diferença das kahalas. Assim, estados que representam conclusões do jogo serão preferencialmente analisados pela função *miniMax()*. Caso o função *avalia()* receba um tabuleiro NULL, isto significa que a jogada atual é inválida. Assim, se a jogada anterior for uma à qual se deseja maximizar as escolhas, é retornado -1000 e, caso na jogada anterior se deseje minimizar as escolhas, retorna-se 1000, para que, assim, o algortimo Minimax nunca decida pela jogada inválida. Os valores, em módulo, 1000 e 100 foram escolhidos convencionalmente para as duas situações e, ambos, são valores que representam resultados acima da maior diferença possível entre as Kahalas, fazendo com que, assim, o algoritmo analise corretamente tais casos.

A partir da avaliação das folhas, o algoritmo Minimax então, volta pela árvore retornando valores destas folhas a partir de um critério de seleção. Para cada nó, é retornado o maior valor de avaliação dos filhos caso o jogador da vez para o estado do tabuleiro no nó presente seja o Jogador ao qual se deseja maximizar o ganho e, consequentemente, o menor valor de avaliação dos filhos caso o jogador da vez para o estado do tabuleiro no nó presente seja o Adversário para o qual se deseja minimizar o ganho. Os valores das folhas são retornados repetidamente até que se encontre a raiz da árvore. Como no jogo Mancala a alternância entre os jogadores não é precisa, já que o mesmo jogador pode repetir a vez diversas vezes, é necessário guardar em cada nó qual é o jogador da vez para que, assim, a função *miniMax()* possa decidir em quais casos deve maximizar os valores e quais deve minimizar. A função que implementa o método Minimax de escolha a partir da árvore *Gametree* é a função *miniMax()*. A partir deste método, então, retorna-se a jogada que irá minimizar a maior perda possível, tal jogada que será correspondente à posição que deve ser escolhida pela IA em sua jogada atual. Para que a IA determine qual a posição referente à esta jogada, a função *miniMax()* retorna o valor obtido para a função *decisaoComputador()*, esta que retorna para a função *organizaRodada()*, por sua vez, a posição que a IA escolheu por meio do método Minimax.

Uma imagem ilustrando o funcionamento destas duas funções pode ser visualizado abaixo.

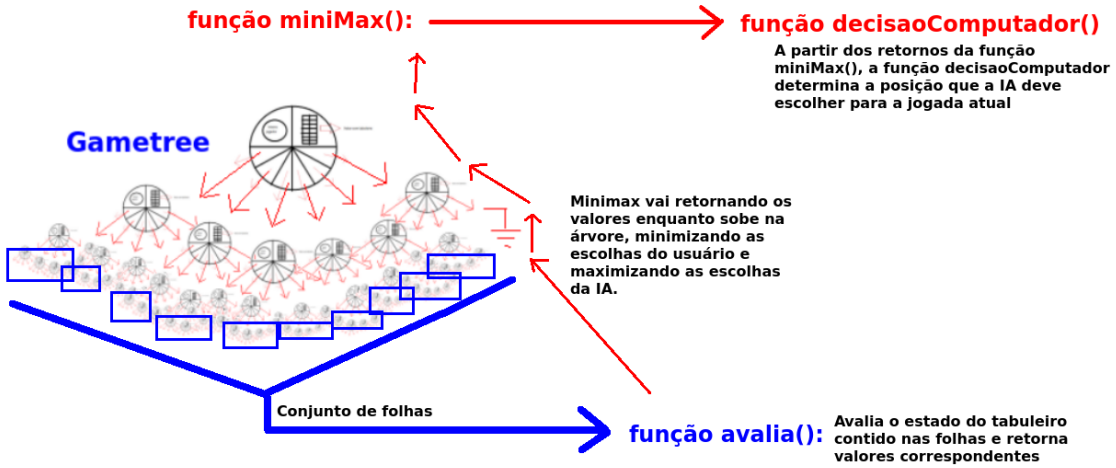

Figure 9. Ilustração do funcionamento das funções miniMax() e avalia()

2.6. A Função *organizaRodada()*

A função *organizaRodada()* é responsável por organizar o jogo toda rodada. Primeiramente ela exibe o tabuleiro do jogo, e logo em seguida, verifica se é a vez do computador ou do usuário. Sendo a vez do computador, é chamado a função *decisaoComputador()* (que já foi explicada anteriormente), somente para decidir qual será a jogada da IA. Após a jogada ter sido decidida, a função *organizaRodada()* então mostra na tela qual foi a escolha feita pela IA.

Caso contrário, ou seja, caso seja a vez do usuário a função *organizaRodada()*, mostra uma mensagem indicando que é a vez do usuário e lê qual será sua jogada. Após a leitura da jogada, é então verificado se a jogada é válida ou não, sendo válida, a jogada selecionada é mandada para outra função que gera um novo tabuleiro com o novo estado do jogo a partir dessa jogada. Por fim a função retorna o novo estado do tabuleiro após a jogada, para assim o jogo dar continuidade.

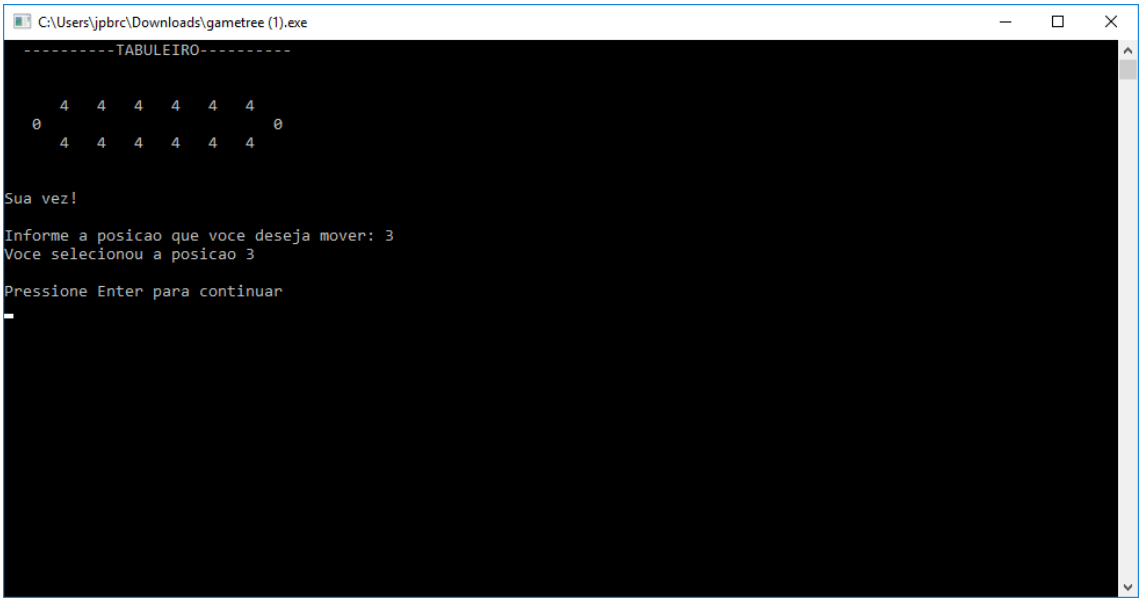

Figure 10. Interface do jogo gerada toda rodada a partir da função *organizaRodada()*

3. Conclusão

Através da implementação do programa, fora possível conhecer melhor o funcionamento das estruturas de dados utilizadas, em destaque o das árvores, como também o estudo da manipulação desta estrutura específica, para que fossem criados algoritmos de inserção, remoção e criação de nós. O trabalho também permitiu a visualização de uma aplicação recorrente desta estrutura de dados no mercado, no caso, para a estruturação de IA's em jogos. Além disso, por meio integração que o projeto exigiu, o trabalho proporcionou a utilização de ferramentas colaborativas como o git, aumentando o entendimento em realizar tarefas em equipe, principalmente voltadas para elaboração de um código. Também foi possível trabalhar mais técnicas de programação ao implementar diversos códigos ao longo do programa.

Dificuldades para a elaboração do jogo foram encontradas e superadas ao recorrer do trabalho. Entre estas pode-se citar a dificuldade para decidir quando retornar valores que maximizem as escolhas ou minimizem dado que um mesmo jogador pode repetir sua vez diversas vezes no Mancala, o que fora resolvido por meio de uma flag armazenada no próprio nó da árvore. Outra dificuldade marcante para a equipe fora para determinar como o algoritmo Minimax deveria lidar com situações de jogadas inválidas, sendo, por fim, decidido que estas deveriam ser representados por um ponteiro NULL na árvore *Gametree*.

References

[Alchieri 2017] Alchieri (2017). Estrutura de dados — alchieri-cic-unb. `http://cic.unb.br/~alchieri`. [Online; accessed 8-May-2017].

[Tenenbaum 1995] Tenenbaum, A. M. (1995). *Estrutura de Dados Usando C*. Makron Books, 1st edition.

[Wikipedia 2017a] Wikipedia (2017a). Game tree — wikipedia, the free encyclopedia. https://en.wikipedia.org/wiki/Game_tree. [Online; accessed 19-June-2017].

[Wikipedia 2017b] Wikipedia (2017b). Minimax — wikipedia, the free encyclopedia. https://en.wikipedia.org/wiki/Minimax. [Online; accessed 19-June-2017].