HOMEWORK 2 - Report

Sunday, April 7, 2019 6:07 PM

Student Name	Mehmet Fatih Kavum
Student Number	529171010

NODE STRUCTURE (gameStateMinimax Class)

Before implementing the minimax algorithm to the game. First, I created a class named "gameStateMinimax". This class will represent the nodes in our tree. Each node (gameStateMinimax) class has the fallowing information in it.

- stateArray: It's a 8*8 2d array that represents the gameboard for the nodeState. It has the same logic with the GameStateClass stateArray
- previousState: This variable holds the previous node in the tree for the node.
- prevPoses: This is a PiecePosition that indicates the moved Piece before played.
- movedPos: This is a PiecePosition that indicates the moved Piece afterplayed.
- stateLayer: This is indicates the node depth.
- statePlayer: This variable holds information about which player is moving the Piece in the nodeState.
- stateOpponent : This variable holds information about statePlayer's Opponent.
- minMaxPlayer: This variable holds information about the miniMaxPlayer. So we can see which player will play after miniMaxAlgorithm ended. This variable is useful while calculating the score.
- score : This variable will hold the heuristic value after calculated.
- stateId: Unique ID for each node. It's just for debugging purpose.
- destList: this is a destination list for each piece. Heuristic1 will use this destinations. It's a static variable.

MINIMAX ALGORITHM IMPLEMENTATION

Here is the Pseudocode that I took from Wikipedia for MiniMaxAlgorithm:

The pseudocode for the depth limited minimax algorithm is given below.

```
function minimax(node, depth, maximizingPlayer) is
    if depth = 0 or node is a terminal node then
        return the heuristic value of node
    if maximizingPlayer then
        value := −∞
        for each child of node do
            value := max(value, minimax(child, depth − 1, FALSE))
        return value
    else (* minimizing player *)
        value := +∞
        for each child of node do
            value := min(value, minimax(child, depth − 1, TRUE))
        return value
```

```
(* Initial call *)
minimax(origin, depth, TRUE)
```

```
(* Initial call *)
minimax(origin, depth, TRUE)
```

- I implement these pseudocode in the "ComputerPlayerMinimax" as a "selectNode" function. It only takes two parameter; GameStateMinimax gamestate, int givenScore. gameState input is our node and you can see the detail of this node in the previous title. I' m using givenScore input to delete unnecessary nodes in the tree.
- I didn't use depth and maximizinPlayer as an input since we can obtain these informations in the node.
- ***Search depth limit can be set in the "ComputerPlayerMinimax" class. Variable: (private int breaklevel)***

Explanation of SelectNode function line by line:

- `List<GameStateMinimax> nodes = generateNextNodes(gamestate);`
  - I get the avaibleMoves as a node by the given node input

```
if(nodes.get(0).stateLayer == this.breakLevel) {
    for (GameStateMinimax node:nodes) {
        calculateStateHeuristicValue_2(node);
    }
```

- If nodes in the depth limit we directly calculating their heuristic scores.

```
}else {
    int tempBestScore = 999999;
    boolean skip = false;
    int deleteIndex = 0;

for (GameStateMinimax node: nodes) {
    if (skip) {
        while (nodes.size() > deleteIndex +1 ) {
            //System.out.println("Deleted at level:" + node.stateLayer);
            nodes.remove(nodes.size() -1);
        }
        break;}

    if (isGameFinished(node.stateArray) != 0) {
        calculateStateHeuristicValue_2(node);
        continue;
    }

    node.score = selectNode(node,tempBestScore).score;

    if (this.whichPlayer != node.statePlayer) {
        tempBestScore = node.score;
    }

    if (this.whichPlayer == node.statePlayer) {
        if (node.score > givenScore) {
            deleteIndex = nodes.indexOf(node);
            skip = true;
        }
    }
}}
```

- If nodes are not in the depth limit, we are fist checking if is game in the terminal state or not. If its in the terminal state we directly calculating their heuristic scores.
- If nodes are not in the terminal state we are calling selectNode function again to take nodes score.
- In the maximization layer, if we have a higher score than parents neighbor, we are deleting rest of the childs since they will be unnecessary and there is no need for calculations.

```
GameStateMinimax theBest = null;

if (gamestate.stateOpponent == this.whichPlayer) {
    int bestScore = -999999;
    for (GameStateMinimax nextNode: nodes) {
        if (nextNode.score > bestScore) {
            bestScore = nextNode.score;
            theBest = nextNode;
        }
    }
}else {
    int bestScore = 999999;
    for (GameStateMinimax nextNode: nodes) {
        if (nextNode.score < bestScore) {
            bestScore = nextNode.score;
            theBest = nextNode;
        }
    }
}
return theBest;
```

- Finally, if we are in maximization layer(depth), we returning the node that has the maximum score. If we are not, we are returning the node that has the minimum score.

Calculation times by depth:

- **Note:** *Parent node is Level: 0*
- **BreakLevel**: 4 **--> MoveTime:** 30-90 ms
- **BreakLevel**: 5 **--> MoveTime:** 500-800 ms
- **BreakLevel**: 6 **--> MoveTime:** 3000-6000 ms

1st HEURISTIC FUNCTION

- My first heuristic function is in the "ComputerPlayerMinimax" class and named as a "calculateStateHeuristicValue_1". It takes nodes as an input and calculate the heuristic value by looking the minMaxPlayer.
- Firstly, it checks if the state is in the terminal state or not. If it is in the terminal state, it returns a score as "+-100"
- If its not in the terminal state, it starts to loop through the board and checks the manhattan distance of each piece to its destination. Destination point for each piece decided by looking the "destList" in the node class.
- destList includes the all the destination points of the game area and maps it to the closest piece. By doing this mapping, there is no way for pieces to get stuck if we have a available move. So we don't need any extra calculations for not getting stuck like we do in the 2nd Heuristic function.
- However, there is one more disadvantage of these mapping. While we are getting close to our destination, sometimes we are not selecting the optimal move. Since our heuristic don't wanna pick the far pieces to the corner destinations.
- This heuristic function is very optimized and calculates the scores nearly 0 ms.

2nd HEURISTIC FUNCTION

- My second heuristic function is in the "ComputerPlayerMinimax" class and named as a "calculateStateHeuristicValue_2". It takes nodes as an input and calculate the heuristic value by looking the minMaxPlayer.
- Firstly, it checks if the state is in the terminal state or not. If it is in the terminal state, it returns a score as "+-100"
- If its not in the terminal state, it starts to loop through the board and checks the manhattan distance of each piece to the corner destination. With these calculation we aren't having the disadvantage of what first heuristic function have. However, now we will have a stuck problem.
- In order to resolve this problem, I hardcoded some rules that gives 100 point penalty if it leads a stuck state. "there can't be any 4 piece in a last line, there can't be any 6 pieces in the last second line, if there is one piece in the last line, there can't be any 5 pieces in the last second line etc…"
- This heuristic function is not optimised as first heuristic function, since it has many if cases. But it stills handles the calculations near 0-1ms.

GETMOVE FUNCTION

- Getmove function in minmaxPlayer is doing the followings:
    - It clones the gameState and makes the parent node with that.
    - selects child node to play with minimax algorithm.
    - with the selected node, we find the moved piece in the original gamestate.
    - Then we move.

FUN PLAYER / ABUSER PLAYER

- I made an extra player which is out of our homework topic, but I bet it's an unbeatable AI. It's created under "ComputerPlayerAbuser" class. It's simply abusing the game rules and force opponent to stuck himself, then win the game. You can give a try :)