Requirements Document: MMM 1.0

Baimukan, Nurpeiis & Ghosheh, Ghadeer

New York University Abu Dhabi
nb2577@nyu.edu; gog211@nyu.edu

I. Objective & Result

The objective of this project is to utilize the concepts learned in the object-oriented programming class and create a checkers game. Unlike most of the board games, where two players play against each other, our game utilizes MiniMax and Alpha Beta pruning algorithms that constitute our artificial intelligence portion of the project.

We called the project MMM 1.0, which stands for MiniMax Machine. The virtual player is able to go up to the 9th future move and try to predict the most suitable move. The more detailed description is provided in the following sections. Each section provides detailed description of concepts used in every class and it's relation with other classes, data structures and the way algorithms are being optimized. If the average time to decide next move for the human being is around 40 seconds, for the MMM it takes in average 10 seconds, and 1.5 minutes in worst case scenario

II. Rules & Constraints of the MMM 1.0

Most of the rules are similar to the introduced in our introductory documentation, however during the first development stage of this project we decided to constraint some of the rules and end game conditions.
1. Checkers cannot perform multiple combat in one move
2. The game is not using kings, so when the checker reaches the other side of the board, the only way it can move is if there is an opposite player's checker on a diagonal behind.
3. When checking whether the game is ended, the program also checks if there are any possible moves left, if not the one with more checkers left on the board wins.
4. It is not necessary to combat the checker if you have an opportunity

III. Implementation

For the implementation there are 5 main classes and in each of them we tried to use the OOP concepts and optimize memorey and time complexities.

I. Cell

Aim:

The Cell class is a class primarily used to get and set the state of each checkers board's cell. This Cell class of information is integral for multiple reasons such as drawing the board and the

methods declared in this class is used in bigger classes to check and set new moves. Implementation:
The cell class has 2 main functions which are the getter and setter of the cell state.
The OOP concepts:
1. The cell states can be either Black, White, White King, Black King or None, this information is stored in enum, which stands for an enumeration- a distinct type whose value is restricted to a range of values which may include several explicitly named constants ("enumerators"). As mentioned before White and Black Kings are included in the enumerator for future modification. 2. One of the flaws of C++ programming is that the values of private data members can be changed through the non-constant getter function. Therefore, the getter functions in all classes are made constant.

II. Board

Aim:
The Board Class is where the checkerboard gets set up. Furthermore, the Board class classifies the moves made by the players and moves the checker in the board if it is not prohibited move.
Implementation:
In the board header file the Player and Movetype is declared as a enumeration to be used in the classes functions. Player has four members = White, Black, Tie, and None. It is not limited with two as in other classes this enumeration is used to depict who is the winner of the game. MoveType as was mentioned before classifies the type of move made by the players: Usual (non-combat valid move), Combat and Prohibited.
OOP concepts:
1. Unorderedmap namespace was created in this class. One might as: what is the difference between the usual unorderedmap class in STL? The answer is that C++ does not support pair hashing, there it base class has to be modified by implementing pair hashing in the header file. Pair hashing is required as the key of hash table is the (x,y) coordinate of the cell and the value this table stores is the state of that particular cell (out of 64 cells).
Hashtable is used because the data set is small, and the key values are unique, thus the access and modification time complexity is O(1) and the space complexity is O(n). In total Boards stores 64 members of Cell Class
2. One more type was declared as a namespace: position, which is a pair value storing the x and y coordinates of the cell
3. Checking the move using standard for loops and if statements would have taken about 200 lines of code for checkers game. However, it is essential to have flexible code that can be easily understood without extra lines of codes. Therefore, we have used array indexing technique, so that every player does not need to have separate code: state of the cell and player is put into two separate arrays.
State sign [3] = State::BLACK, State::WHITE, State::NONE;
int index = player == Player::BLACKK ? 0:1;

During the checking time if the player is black the index is 0, which corresponds to black state of the cell in sign array. Thus when checking you have to check if the checker taken has state[index] and if the next move directed towards state[2] (None) or state[1-index], which is the checker of opposite player.

III. IOstream

Aim:
The IOstream class is a class that takes the input from the user and draws the board in terminal.
Implementation:
As the IOstream is used to get the input and interaction from the user. It loops through the board and draws the board.
OOP Concepts:
1. When declaring the individual input/output functions we made them static. There is no need in instantiating an object of IOstream, if we want to get the input and print the outputs.This is the reason behind the static member functions of IOstream. The function getMove takes the inputs from the user and then stores it in a pair. Drawing the Board is performed by DrawBoard function where the unicode of the black and white circles are used for the graphical interface.

IV. MiniMax

Aim:
This is the "brain" of the game that stores possible moves and points at certain state of the board in two separate hashtables and furthermore it stores the methods of AI player. This code is classified to be AI, since the conditions are not hard-coded when virtual player is making choice, but Implementation:
Algorithm
The algorithm used in this function is called MiniMax, and it is optimized by using alpha-beta pruning, which enables us to ignore some of the branches of the tree, thus decreasing the time complexity for 95%. There are two parts for the algorithm: getting unique hash key to store the points and moves at certain state of the board. The second part is using this hashtables implement minnimax and alpha-beta pruning algorithms.

Getting unique key for the hashcode:
1. Initialize the string state to empty string
2. Loop through the board and get the state of each cell of the board
3. If the state of the None, push back 0 into state
4. Else if the state is Black, push back 1 into state
5. Else if the state is White, push back 2 into state
6. As the last number push 1 if the current player is Black and 2 if the player is black
7. Convert state which is base 3 number into decimal (long long) number and save it into key
8. Return key, which is long long number
This hash function is used both in hashtables that stores the points of the current board and that stores the move

of the virtual player given this board state.
Using this hashTables, the program does not need to compute the point evaluation for all the children of the board state if it already computed it in the past.
The MiniMax algorithm uses depth-first tree search and Alpha-Beta pruning makes sure that program does not go to every single leaf of the future possibilities of the board state. The depth of the tree is equal to 9 in this game. And choice of this depth will be explained after explaining the algorithm. The MiniMax and Alpha-Beta Pruning Algorithm can be explained in simplified way, as follows:
1. Get The Board state and get the key value for this board state using the hash function described above.
2. If the moveTable (move hash table) has this key return the points at this board state.
3. While the depth of evaluation is less than 10 and the board state does not satisfy the end condition, perform following steps.
4. Get the next possible move for the current player. In order to get the possible move the program loops through the board and performs either Usual or Combat move, similar to the function described in Board
5. Pass the updated depth and change the player into second player and do further exploration.
6. If you reached the depth of 9 or the end game condition evaluate the points, by using the function shown below it uses basic heuristics where we assign more points if there are 5b and 4w rather than if there are 4b and 3w, although the net is 1 for both, as the probability of black winning with more checkers is higher than with less. The white checkers weigh -1 and black checkers weigh +1 points, thus black player tries to maximize points whilst white minimizes its points:

```
double MiniMax::evaluatePoints (string boardState){
    int noBlack = 0;
    int noWhite = 0;
    for (char & c: boardState.substr(0, boardState.length()-1) ){
        if (c == '1')
            noBlack++;
        else if (c == '2')
            noWhite++;
    }
    double points = 0;
    if (noBlack == noWhite){
        return points;
    }
    else if (noBlack > noWhite){
        points = noBlack - noWhite + (noBlack/12.0);
    }
    else {
        points = -(noWhite - noBlack) - (noWhite/12.0);
    }
    return points;
}
```

Figure 2: Evaluating points

7. If the current player is black it tries to maximize its points, whilst if it is white player it tries to minimize the points of board, thus if the parent node is max node it will choose the child node with maximum points for board state. The inverse is true with min parent node. The simplified binary tree model of the minimax tree is shown in the figure below:

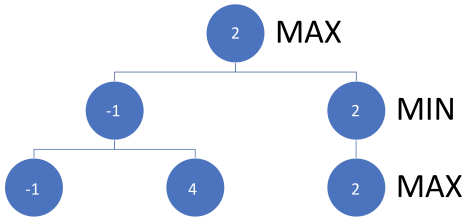

Figure 3: MiniMax tree simplified model for tree with depth = 3 and branching factor of 2

8. If the current node is maximum node and the nextboardstate maximum possible points is less than its current point do not evaluate that branch of the tree at all 9. Else if it is minimum node with current point less than the minimum possible points for the nextboardState do not evaluate this branch of the minimax tree. 10. Return the points The development of the minimax algorithm can be described using the following diagram:

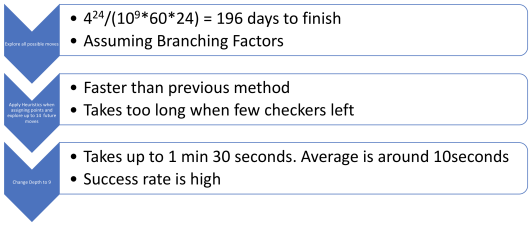

Figure 4: Development of the project

Initially we have tried to calculate all the possibilities, and save it into the hashtable of all possible move. Thus, it would have mean that it is impossible to win that virtual player, because it knows all the possible moves until the end condition. However, making some calculations we have found that it would take around 200 days for this code to finish computing, as the average branching factor is about 4.
Next, we decided to use heuristics to evaluate points although computer has not reached the end state, as shown in the figure 2. *OOP concepts*
1. Optimize the time and space complexity by reducing the steps
2. Store all the members of the class in a private portion, so that on Game, which is its friend class can have access into it
3. Only explore is public member, thus the vulnerabilities inn the "brain" of the program is minimized

V. Game

Aim:
The game class is an aggregator glass that connects previously discussed classes and allows for playing.
Implementation:
OOP concepts:
1. The class Game has a private member of type current player as well as board. Also it has member functions such as getPlayer and switch player to allow for turns and a proper game dynamics.While the function makeIO integrates the IOstream with board and its drawing and setting.The Game starts with the black player which happens to be the virtual player using Artificial Intelligence part of the project. 2. The major and only public function Game is the function start. This function uses the Minimax class and the logic from the board class along with the Game class functions to have a fully independent game with its full game dynamics. 3. As it is a friend of MiniMax class it has an access to its private functions such as checkEnd, which is used in game to decide the winner of the game.

VI. Menu

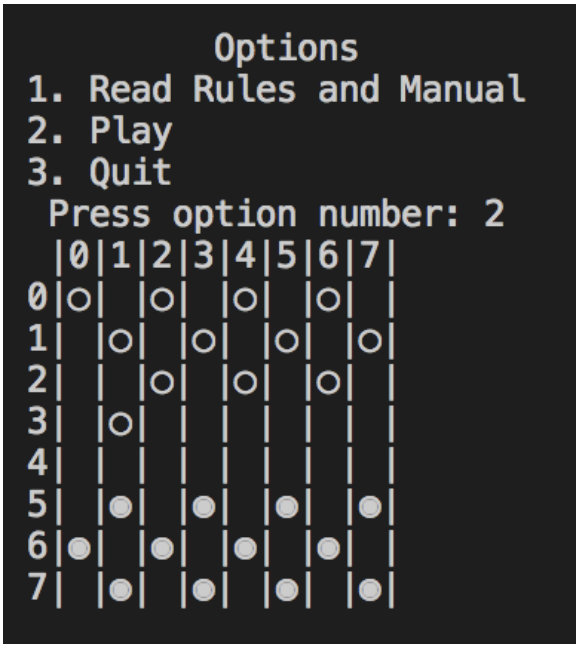

Figure 5: Menu

Aim:
This function allows the user more flexibility and options to use the game.
Implementation:
Once the program gets executed the user will have menu with 3 options to choose from. Th first allows the user to read the game manual. The second is the actual playing of the game. And the last one is quitting the game. The Menu class has a void member function called Options which is basically the function that takes the user choice or option and then either prints out the rules, calls the game object or even just quit or exit the program.

IV. Future Plans and challenges

1. Use hybrid of MiniMax and Neural Networks, where the evaluate points is not decided heuristics, but is decided based on the training data set.
2. Implement more rules, such as the kings and multiple combat
3. Fix Graphical User interface and implement mouse clickable checkerboard
Some challenges faced were:

1. Windows and SFML do not work on mac environment
2.Open GL and GLUT frameworks also had problems so we had to to add extra files and libraries to make them compatible with IDEs on Mac and Linux.
3.Still even after modifications faced problems in getting the visuals to work and respond to mouse clicks and drags.

References

[1] Saqib Shah, 19 October 2017, *Google's AlphaGo AI no longer requires human input to master Go https://www.engadget.com/2017/10/19/google-alphago-zero-ai/*

[2] Accessed 11 Feb 2019 , *Draughts History. âĂIJ Draughtshistory.nl, ww6.draughtshistory.nl/./*

[3] J. Schaeffer, Jan 1994, *Chinook Is World Checkers Champion! ICGA Journal, vol. 17, no. 3, 1 Jan. 1994, pp. 174âĂŞ174., doi:10.3233/icg-1994-17312./*

[4] Samuel, Arthur L. , Jan 1988, *Some studies in machine learning using the game of checkers IBM J. Res. Develop., vol. 3.3, pp. 210âĂŞ219, 1959./*

[5] Chellapilla, K., and D.b. Fogel. , Jan 1999, *Evolving Neural Net-*

works to Play Checkers without Relying on Expert Knowledge. IEEE Transactions on Neural Networks, vol. 10, no. 6, 1999, pp. 1382âĂŞ1391., doi:10.1109/72.809083./

[6] Akshay L. Aradhya, 19 October 2017, *Minimax Algorithm in Game Theory | Set 1 (Introduction) https://www.geeksforgeeks.org/minimax-algorithm-in-game-theory-set-1-introduction/*

[7] *Akshay L. Aradhya, 19 October 2017, Minimax Algorithm in Game Theory | Set 4 (Alpha-Beta Pruning)* https://www.geeksforgeeks.org/minimax-algorithm-in-game-theory-set-4-alpha-beta-pruning/