Specification and Design

Title: **Visualization and AI for 3D tic-tac-toe**

Student: **Nan Zhao**

Tutor: **Prof. Boris Konev**

1. Introduction of tic-tac-toe

The game played on a three-row chessboard dates back to ancient Egypt [1] and was found on roof tiles around 1300BC [2]. Early variations of tic-tac-toe were broadcast in the Roman Empire around the first century BC. Because of the simple structure of the tic-tac-toe game, it became a common example in artificial intelligence in the early years. From the existing gameplay, the winning method of the game can be summarized, and a program can be deduced from the strategy, allowing the computer to play against the user.

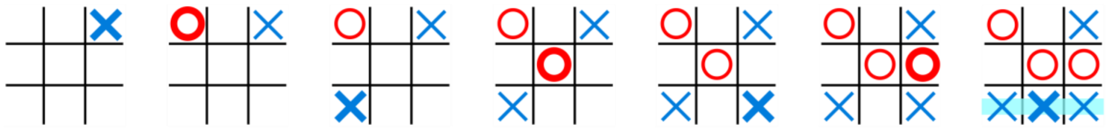

Figure1. A winning example of the player with a cross (X)

Two players, one with a circle (◯) and one with a cross (X), take turns playing their own symbols on the 3x3 grid, and the first to line up the horizontal, vertical, and diagonal lines is the winner (as shown in figure 1). If both players play correctly, the board would be filled and the game will be tied (as shown in figure 2).

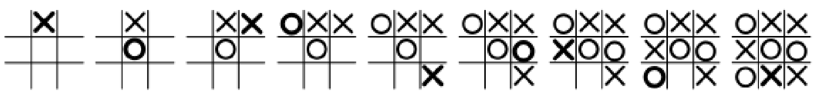

Figure 2. The draw of tic-tac-toe

The game is actually controlled by the first player, the first player is the offense, and the second is the defender. If the first player is in the corner with the first piece, the chances of winning are the largest (as shown in figure 3). If the second player is on the side and the corner with a piece, the first player can contain the second player with two pieces of connecting lines, so the second player has to go down the center. If the first player plays in the center, the second player must play in the corner so that the second player does not lose the game. If the first player plays on the side, the second player can play in the center or corner, or on the side opposite to where the first player played.

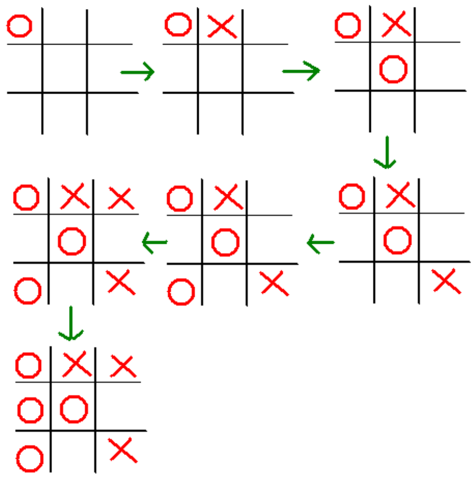

Figure 3. The first player is in the corner with the first stone

2. Specification of 3D tic-tac-toe

From the original plane to pass the three levels, change it to a three-dimensional 3x3x3 to pass the three levels. When the three symbols of the player side are connected in a three-dimensional, vertical, horizontal, and diagonal direction, the player would win (there are 49 winning lines in total). However, it is not very interesting, because as long as the first player is in the center of the cube, it is guaranteed to win (the center of the cube can control up to 13 connections) and there is no possibility of a tie in this game, no matter how many squares each side occupies. If the first player is prohibited from playing in the center of the cube, the second player would win. If the grid in the center of the cube could be deleted, it becomes the first player would win.

In this game, in order to increase the fun of the game, using 4x4x4 extending games of chess to pass four levels, there are totals of 76 winning lines. Although the first player is still sure to win, the difficulty is increased. At the same time, both sides hold two kinds of pieces and use them in sequence. When the pieces of the same kind connect in a line, the victory wins. For example, player A holds ○ and ◎ pieces of chess, player B holds X and ※ pieces of chess, and the order of the next piece of chess is player A (○ or ◎), player B(X or ※), player A (○ or ◎), player B(X or ※). If the line of ○◎○ ○ does not count as a win for a player of A, because they are both the chess of player of A, but they are different types of chess, a player of A need to make a line of ○ ○○ ○ or ◎◎◎◎ to win. Completing the two modes of player-to-player (PVP) and player-to-AI (PVAI) and choose to play first casually. The component of the grid is the current state of each grid, whether it is placed by player A (○ and ◎), placed by player B (X and ※), or empty. The component of the turn is whether it is Player A (○ and ◎) or Player B (X and ※).

3. Design of 3D tic-tac-toe

3.1 Game rule of 3D tic-tac-toe

Firstly, initialization of the game. Secondly, the player puts the pieces and finds the corresponding array element changes, and pays attention to judging the end of the game and changing the round. Thirdly, AI chess placement strategy, AI should do local optimal operations, for each square, if AI can win, AI will win first, then if the opponent can win, AI will block first, otherwise, playing randomly. Especially, AI should first traverse the squares to judge that AI won, and then traverse the squares to judge that the opponent won, otherwise AI will not get the optimal solution. Finally, judging the game over.

3.2 Introduction of Algorithm

When using game tree search, there is an assumption that both sides of the game are smart enough to choose their optimal strategy every time.

In the MinMax algorithm, there is an evaluation function for each board position, and the evaluation in favor of player A is positive, and the evaluation in favor of player B is negative. Therefore, when player A makes a move, player A will definitely choose the step that maximizes the position evaluation function, which corresponds to Max, on the contrary, when player B makes a move, player B will definitely choose the step that maximizes the position evaluation function, corresponding to Min. Using Alpha-Beta pruning optimization space. For each node, define two values, alpha and beta, which represent the minimum value guaranteed by the max player and the maximum value guaranteed by the min player from the root node to the current situation. Initially, the root node alpha = $-\infty$, beta = $+\infty$, indicating that the final value of the game is in the interval $[-\infty, +\infty]$. In the process of downward traversal, the child node first inherits the alpha-beta value of the parent node and then inherits the interval [alpha, beta]. When the child node is traversed downward, alpha or beta is updated synchronously, and once the interval [alpha, beta] is invalid, it will return up immediately.

In the nine-square checkerboard, we calculate the best next move for each chess game and do Alpha-Beta pruning in the process, that is, return directly when the current player's winning move has been found. This is better for single-position calculations, but the AI needs to repeat this process at every move, which is very time-consuming when the board size is greater than three.

In this game, using the sixteen-square checkerboard, the Minimax algorithm is used when the initial empty board is used to ensure that all states are traversed, and the best results of all the games are cached. For each chess position faced by the AI, it only needs to find all possible positions in this chess game and return the best decision, which greatly reduces the repeated minimax recursive calculation in each chess game.

The evaluation function is a very important part of the algorithm. The evaluation function is to fill all the empty squares with the current player's pawns and calculate how many of all the rows, columns, and diagonals are connected into four pawns, and use the total number of these pawns as the evaluation value. If one player has won in the current situation, then the valuation value is set to infinity (a very large value can be used in the program).

3.3 Visualization

Using Mayavi and TraitsUI completes the visualization of 3D tic-tac-toe. Building steps of Mayavi window include building class from the inheritance of HasTraits, defining different variations from window, definition layout of view, and then building MlabSceneModel scene of 3D tic-tac-toe, building View, and definition _init_ function for producing data. Secondly, building examples of Class and transfer configure_trains()

method. Building steps of TraitsUI visualization has definition class from the inheritance of HasTraits and showing windows. Finally, the game has a judgment of victory, defeat, and draw. Finally, the game has a judgment of victory, defeat, and draw.

Create a two-dimensional array matrix to record the chessboard, 0 means the grid is empty, 1 means player A plays O chess, 2 means player A plays ◎ chess, 3 means player B plays X chess, and 4 means player B plays ※ chess. The click function is responsible for modifying the state of the grid and the rounds. When refreshing each time, the check function checks whether the chessboard has the same vertical line, horizontal line, and diagonal line (1 or 2 or 3 or 4, not 0), and if so, a winner appears. Or checking whether the chessboard is full With sixteen pieces and no winner, it's a tie. The situation of the chessboard should be reflected on the UI, so the bottom displayed by OnGUI displays 'O', '◎' 'X' '※', blank, according to the state of two two-dimensional array matrices. The text UI of the battle situation also changes the display according to whether there is a winner (this depends on the output of the result), and which player's turn is displayed in each round.

Reference:

1. Zaslavsky, Claudia. *Tic Tac Toe: And Other Three-In-A Row Games from Ancient Egypt to the Modern Computer. Crowell. 1982. ISBN 0-690-04316-3.*

2. Parker, Marla. *She Does Math!: Real-life Problems from Women on the Job. Mathem -atical Association of America.1995:153[2020-06-08]. ISBN 978-0-88385-702-1.*

3. Kupiainen, H. Extending Unity Game Engine Through Editor Scripting. *2018*.

4. Meijers, A. (2020). Unity. In: Immersive Office 365. Apress, Berkeley, CA. https://doi.org/10.1007/978-1-4842-5845-3_4