Group Project #1 - 3-d connect-4

Team name: eat_or_dai, Group ID: 09

Members: 0716057 張家誠、0716209 戴靖婷、0716231 黃嘉渝

Code Overview

- Compile command:
  - `g++ -std=c++14 -o Team_9.exe Team_9.cpp -lws2_32`
- Goal:
  - Win the 3-d connect-4 game with higher points
  - the kth four-in-a-line gets $\lfloor 100/k \rfloor$ points
- Constructing Classes
  - **Node**
    - nodes in the Monte Carlo tree, representing the possible moves for each player
    - Attributes:
      - ParentNode: a pointer to the previous step
      - ChildNode: a set of expanded next moves
      - board: the board status after moving to this node (after moving this step)
      - unexpanded_move: the possible next moves which are not expanded yet
      - move: the move it takes to get to this node from its parent
      - pieces: how many pieces on this board / # of steps it takes to this node / the depth of this node
      - visit: the number of times this node has been traversed by simulation
      - win: the difference between the number of times this node wins (higher score than the opponent's score) and the number of times this node does not win after several rollout
      - score: the difference between the score of this node and the opponent's score
      - lines: the number of lines formed in the current status
      - color: this move is chosen by black (1) or white (2)
    - Functions:
      - UCB: to find the balance between exploitation (keep trying certain move) and exploration (try new unexpanded moves)
      - select: choose the next move (child) with the highest UCB
        - if exploration_ratio is set to 0: GetAction selects the next move

        - expand: randomly choose an unexpanded path, and construct a new expanded child node
        - backpropagate: backpropagate the number of wins and visits to all of its path (parents)
        - rollout: after selecting the next move, randomly choose a path from the available moves until the game is over and return whether it is win or not
        - deletnodes: due to bad_alloc, we sacrifice some time to free our memory
    - **MCTS** (Monte Carlo Tree Search)
        - Attributes:
            - root: the opposite color of our color
                - if we are black: root is the status without anything
                - if we are white: root is the status with the first piece by the opponent
            - curNode: the current status of the Node, including the board status
        - Functions:
            - traverse_to_leaf: include selection and expansion until the new leaf node is expanded
            - simulation: an iteration of the whole process of MCTS, including traverse_to_leaf, rollout for finding whether win or lose, and then backpropagate the result back to every nodes on the path
            - GetAction:
                - if it is the first time entering this function, initialize the MCTS status
                - then, find whether the opponent’s move is within our expanded range
                    - if in our expanded child, then move on to that status
                    - otherwise, find the opponent’s move from the unexpanded move and construct the move status
                - simulate 4 seconds (but not 5 seconds) because we have to reserve some time for deleting nodes information in order to deal with bad_alloc (no more free memory)
                - after simulation, select the node with the highest win ratio
                - after one game, delete every nodes and play another game
- Functions
    - calculate_move_score:
        - there are 13 directions of forming lines
        - score: the difference between my score and the opponent score (which may be negative)
        - for vertical and horizontal lines,
            - check the positions near the move in one direction first, and find the maximum number of the same color nodes which are connected; then, calculate the number of formed lines and the score
        - for oblique lines,

        - find the relation of equation between l, i, and j, and check the neighbors which are possible to form lines with the move directly
    - get_available_move: return a set of available moves whose board status is 0 (space)
    - get_random_idx: choose a random index for unexpanded nodes when expansion and rollout
    - GetStep: return the move chosen by us to the server in the use of GetAction in mcts

Experiments and Experiences

	Sample1	**Sample2**
Us (black)	**219**: 65	**170**: 74
Us (black)	61 :**183**	86 :**195**
Us (white)	**173**:108	**172**:119
Us (white)	**175**:116	**249**: 49

which we can see that we win all the games.

- 用差分 >0 或是用差分 >=0 來判斷是否為好的一步的比較：
- using **Us vs Sample1** & **Us vs Sample2**

Win (Our color)	**Us vs Sample1 (black-white)**	**Us vs Sample2 (black-white)**
>=0 (black)	**160**:121 (59)	**192**:130 (62)
>=0 (white)	123:**147** (-24)	132:**176** (-44)
>0 (black)	**219**: 96 (123)	**170**: 74 (96)
>0 (white)	61 :**183** (122)	86 :**195** (-109)

可以發現使用 >0 作為判斷基準的話，分數落差會更大，代表選擇到好的一步的機率變大，讓我方更有機會下對我方更有利的下一步。

- 比較不同的 simulation 時間差

	3s (black - white)	4s (black - white)
Us(black) vs Sample1	**203**: 97 (106)	**219**: 96 (123)
Us(white) vs Sample1	94 :**150** (-56)	61 :**183** (-122)
Us(black) vs Sample2	**202**: 89 (113)	**170**: 74 (96)
Us(white) vs Sample2	103:**178** (-75)	86 :**195** (-109)

可以發現simulation的時間比較長時，總分可以拿到比較高，且得分也有落差比較大的趨勢，因為simulation會盡可能的在時間限制內走到遊戲的最後，來判斷這步是否是對我們有利的一步，所以simulation的時間比較長的話，能較精準的判斷，進而讓獲取分數變高且分數差比較大。

Things We have Learned and Ideas of Future Investigation

- The function, get_random_idx, is not an actual random move. Although there is another way to implement actual random work, we are stricted to the time limit. Therefore, we chose a faster way to find a random number by srand and rand provided in std namespace.
- 如果拿到每一次下棋的位置，可以透過機器學習建一個model(CNN)train 一個 expert pool(可以試試看把每下一步的棋盤當作input data放進Convolution3D)，可以多做幾層後中間使用relu等activation function，最後再做flatten，得到預測下一步（unsupervised machine learning），做出比 random 更好的選擇方法——expert pool.
- 當 simulation 時間比較長時，我們贏的分數會比較多，因為 simulation 會盡可能的走到遊戲的最後（在時間限制內），如果時間開的比較長，得到的好壞（這一步下的好或是壞）的模擬就可以逼近真實，而 simulation 判斷好壞是透過計算彼此的得分差距，當差距 >0 時才會被認定是個好的一步。
- 中間在測試過程中，有遇到記憶體不夠用而導致程式停下來的狀況（沒有把確定不會用到 node 做刪除），我們透過在 select時，將我們要的點留下後，其他做刪除（因為下一步時也不會再用到這些被撿剩的 node），如此來減少記憶體的負擔。
- 可以利用多線程對simulation進行多次，或是利用多線程進行UCT tree的搜索，但可能要上鎖等避免同時更動值。
- minimax 與 MCTS 的比較
  - minimax
    - 對於當前的局面，盡自己所能（也許是四步或是更多）來模擬對手可能會下的步，並從中選擇對自己最有利的（可能是將獲得的分數最多或是分數落差最大）來當作下一步。
    - 而 minimax 就是用來 pruning 掉一些一看就知道不適合或是沒有比較適合的步，將自己 maximum，而對手 minimum，以減少計算量，根據一些 evaluation function 來計算每一個 node 的分數。
    - 因為需要同一層的點都要進行探索(pruning 了也不是一個點)，在記憶體跟時間有限的情況下，沒辦法看到更後面的狀況，比起 MCTS 的估計會略輸一截。

- MCTS
  - **Exploitation**：Follow the best known path
  - **Exploration**：Try unknown paths
  - 而這題就是應該要在現有的線上繼續發展，還是要去發展新的一條連線的權重上進行分配。
  - MCTS 在 simulation 時會盡可能試到遊戲的最後，（每層只會選擇一個點繼續做發展，可以走得深度比minimax還要深），對於遊戲的掌握度比 minimax 還要高。
- We have tried to implement minimax at the stage of selecting the next move; however, the result does not improve obviously. Therefore, we maintained the original MTCS algorithm.

Contributions of individual team members

- 0716057 張家誠：discuss MCTS algorithm and construct classes, build the architecture of program, implement and debug most of MCTS and MCTS node
- 0716209 戴靖婷：discuss MCTS algorithm and construct classes, write and debug calculate score function, tries of minimax, write report
- 0716231 黃嘉渝：discuss MCTS algorithm and construct classes, debug the calculating score function, write report