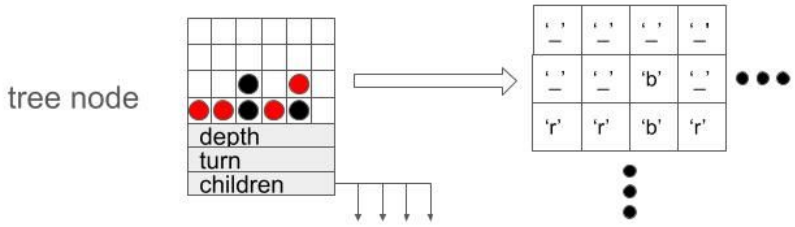
Project:
Our project was an AI that can play competitive 2-player board games, currently implemented for tic-tac-toe and connect four and extendable for others as well. The AI uses a minimax algorithm to recursively evaluate a directed graph containing the possible moves that both players can make. This evaluation is done using a heuristic specific to the game being played.

Implementation:

Data Structures:
The AI uses a directed graph in the form of a game tree to store all possible game states to a certain depth. Each node of the graph is a TreeNode class, storing:

- a unique game state object describing the game board,
- a vector of pointers to the node's children,
- the depth of the node,
- which player's turn it is for that node, and
- getter functions for these data members



The TreeNode data structure

The GameState is stored separately. The base GameState class contains:

- The game board, stored as a 2D dynamic array of characters ('x', 'o', or ' ' for empty)
- functions to deal with move validity and gamestates that could be valid "children" of the current one
- a heuristic function to evaluate the favorability of the gamestate for the player and AI

In addition, there are two classes derived from the base GameState class: TTTGamestate, which deals with the tic-tac-toe gamemode, and C4GameState, which deals with the connect four gamemode.

Program design:
The Driver file includes a master GameState object representing the current board, an AI object that takes turns against the player, and a main function executing the game loop. Both games are executed from the same loop - the AI and game flow is general enough to preclude separate functions for each game (this was a design element we aimed for).

The AI Class includes a private minimax function to recursively evaluate the game subtree beginning at the current gamestate, to help in choosing the move (assuming that the opponent plays optimally) and a public function that uses this minimax function to choose the most favorable move. This minimax function uses the heuristic included in the specific GameState class for the game being played.

Example outputs (beginning and end, for length):

```
==============================
Welcome! What game would you like to play?
1) Tic-tac-toe
2) Connect 4
==============================
2
Which player would you like to be?
1) x
2) o
1
Please enter the depth of the game tree to search (1 - 9)
6
AI is playing, may take a while…
==============================
|   |   | x |     |   |   |     |
| o |   | o | x   |   |   |     |
| x | o | o | o   |   | o |     |
| o | o | o | x   |   | x |     |
| x | x | x | o   | o | x | x   |
| o | x | o | x   | x | x | o   |
==============================
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
AI wins!!
```

```
==============================
Welcome! What game would you like to play?
1) Tic-tac-toe
2) Connect 4
==============================
2
Which player would you like to be?
1) x
2) o
1
Please enter the depth of the game tree to search (1 - 9)
6
AI is playing, may take a while...
==============================
| o |   |   | o | o | o| o |
| o | x |   | x | x | x | x |
| x | o |   | x | o | x | x |
| o | o |   | x | o | o | o |
| o | x |   | o | x | x | x |
| o | o | x | x | x | o | x |
==============================
| 1 | 2 | 3 | 4 | 5 | 6 | 7 |
Player wins!!
```