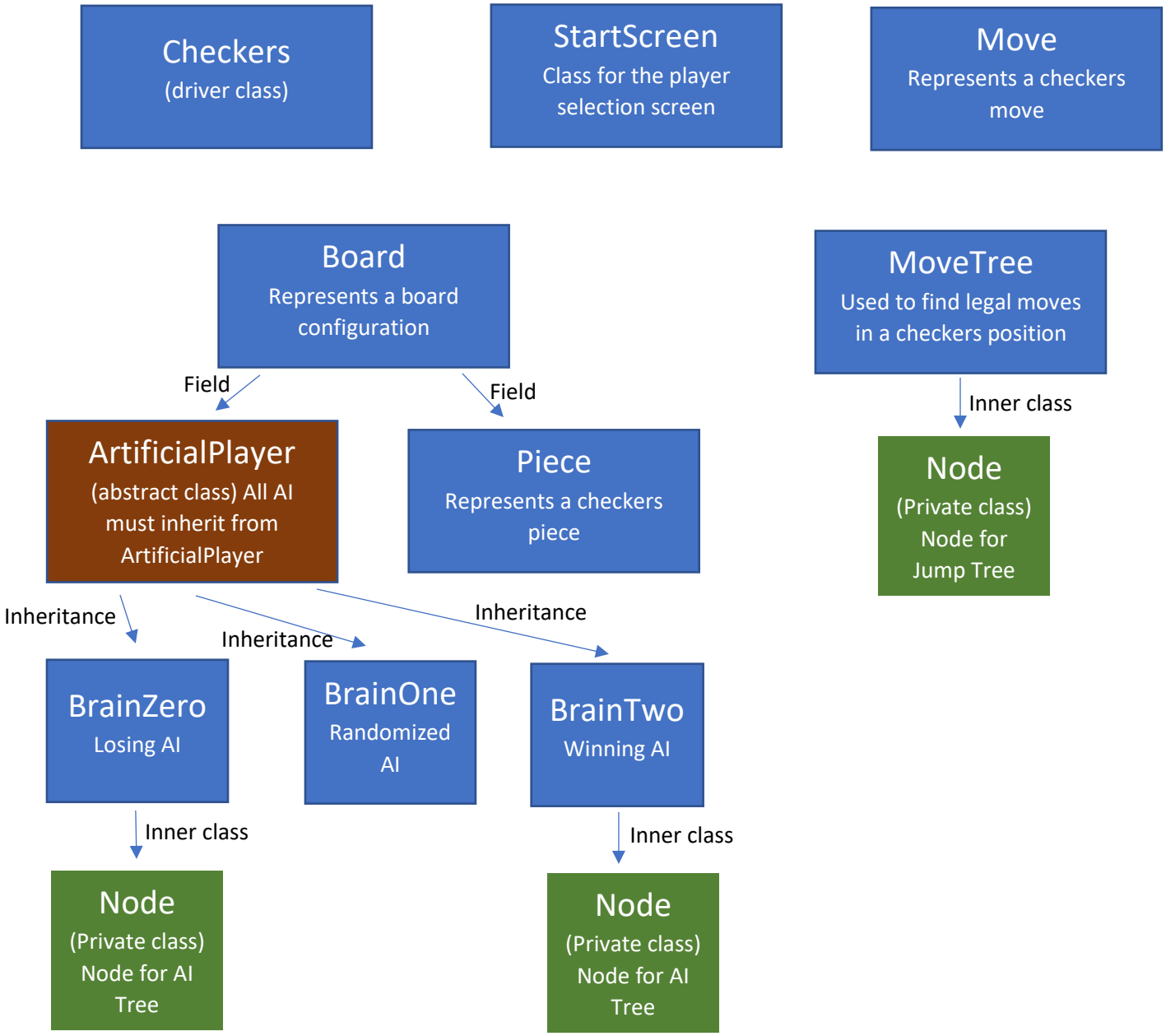
Checkers Explanation Document

This document focuses on explaining how the checkers game works, and what skills and techniques were used in its creation.

General Class Structure



Flow of the game

(Note: ClassNameInstance refers to an instance of ClassName for tracing purposes, and does not reflect the actual variable name)

- **Checkers:** General setup occurs. The scene is set to "start", so the menu screen is displayed in draw(). The program listens for mouse clicks, and runs the click() method of the StartScreen if a click is detected.
    - **StartScreenInstance.click():** If the click is within the bounds of the three buttons on screen, the method will either change the value of the current players or begin the game.
    - **Checkers.mouseClicked():** If the begin button is selected, the scene is changed and the main Board's players are set to their correct values.
- **Checkers.draw():** BoardInstance.displayBoard() is called, which displays the board and all the pieces.
- **Multiple files:** This segment will loop until the game status has changed (the game has ended)
    - **Multiple files:** The game awaits a move and submits it if found.
        - **Checkers.draw():** BoardInstance.pollForMove() is called. If the current player is a computer opponent, this method calls the ArtificialPlayerInstance.getMove() method of the current player, then sends the move to BoardInstance.submitMove().
            - **ArtificialPlayerInstance.getMove(Board b)**: An abstract method returning a move string, given a board position.
            - **BoardInstance.submitMove(String moveStr):** Will call BoardInstance.checkValidity() to make sure the given move is legal, then actually moves the pieces on the board according to the move string.
        - **Checkers.mouseClicked():** Call the BoardInstance.click(float x, float y) method. This method will add the square to a list of currently selected squares.
        - **Checkers.keyPressed():** If the enter key is pressed, call BoardInstance.submitClicks(). This will combine the selected squares into a move string, then attempt to submit it through BoardInstance.submitMove(), which is detailed above.
- **Checkers.draw():** Once the game state is no longer zero, the game is stopped with noLoop() and the end screen is shown.

MoveTree class logic

This class is responsible for finding all legal checkers moves for any given board position.

getMoves method logic:

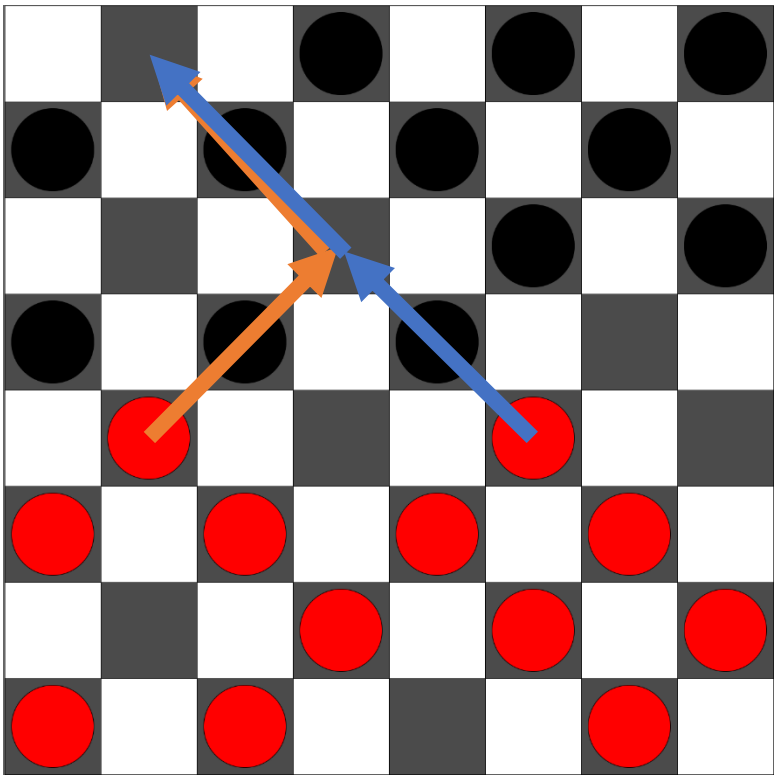
- This is the method that finds all the different possible moves.
- The overall sequence of logic is as follows:
    - Clear all fields of the MoveTree object
    - Build the jump tree (find all single and multi-jumps)
    - If the jump tree exists, return the leaves of the tree
    - Otherwise, find and return non-jump moves

Jump tree:

(Moves here are represented by the moveStr it would use in this program. If a piece jumps from (3,3) to (5,1), its moveStr is 3351. Pipes connect moves to make a single move.)

A jump tree is created by listing single jumps as children of the root, then double jumps as children of the single jump they continue, etc. According to the rules of checkers, if the jump tree exists

Here is an example of a jump tree, with red to play:



The jump tree is as follows:

- root
    - 1335
        - 1335|3517
    - 5335
        - 5335|3517

This tree fits the possible moves, as shown by the arrows on the left.

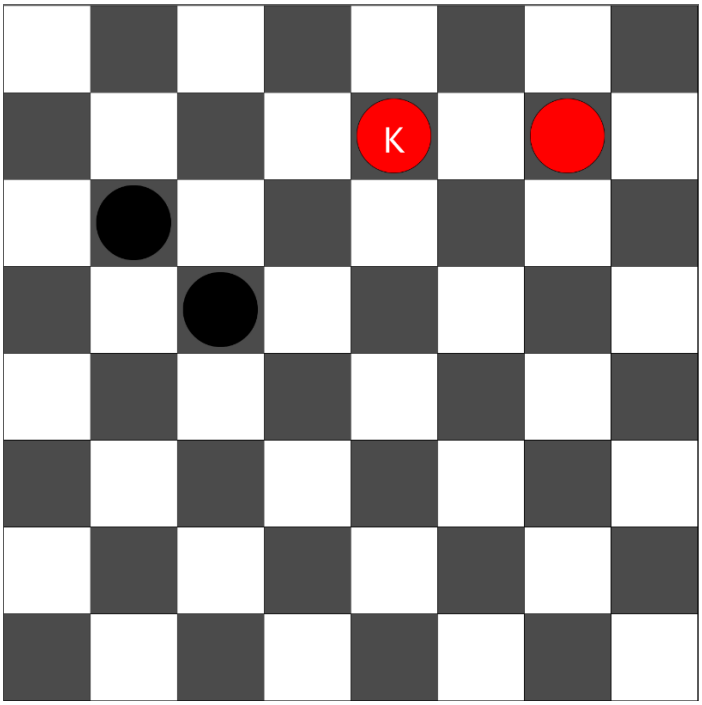
The legal moves in this situation are the leaves shown above, 1335|3517 and 5335|3517.

BrainZero and BrainTwo logic

These classes are two very similar AI, with one crucial difference: BrainZero plays what it considers its worst possible move, while BrainTwo plays what it considers it best possible move.

In order to do this, both AI create a move tree. Each leaf's evaluation is based on the difference in the number of checkers between red and black in its board position, where kings are worth 3 normal checkers. These leaves' parents' evaluations are based on the highest or lowest evaluation among the leaves, depending on the turn (if it is black's turn, it will look for the minimum evaluation, if it is white's turn, it will look for the maximum evaluation).

An example is below (with depth of one full move, while both AI have a depth of three full moves), with black to play:



The move tree is below, with their evaluations:

- Root:
  - 1506: 2 (lowest score)
    - 4635: 2
    - 4637: 2
    - 4655: 2
    - 4657: 2
  - 1526: 2 (lowest score)
    - 4635: 2
    - 4637: 2
    - 4655: 2
    - 4657: 2
  - 2435: 4 (lowest score)
    - 4624|2406: 4

A higher score is better for white, and a lower score is better for white. In the above position, a losing AI with depth 1 would play 2435 (forcing the opponent to take a double jump), while a winning AI with depth 1 would play either of the other two legal moves, since they are equally good in its eyes.