The model would consist of an overall Checkers Game object. Within the Checkers Game object would be:

two Players,
a Board,
the Turn Limit,
and the Current Turn.

The Checkers Game object can get, and set a player's name if they are not a computer. Each time a player finishes their turn, the Current Turn is incremented in half-steps starting with the black player's first turn. This will indicate whose turn it is currently since if the turn is an integer value, that indicates it is the black player's turn, if it has a decimal value, it is the red players turn. The game can continue as long as the Current Turn is less than the Turn Limit. If the Current Turn is greater than the Turn Limit, and both players still have at least one piece on the board, the game is a draw.

Each Player would have:

a Name,
Color (red/black),
a list of Checker Pieces,
a list of the opponent's Checker Pieces that have been captured,
and a flag if they are a computer player (which would change their name to 'Computer - <Color>').

The color of a player indicates the turn order that they play in (black, then red). It also indicates the starting side of the players pieces on the Board, and the direction the pieces may move, unless the piece is marked as a king (it may move backwards as well as forwards). If a player no longer has any pieces remaining on the Board, the game ends with the player with no remaining pieces being the loser, and the other player being the winner.

Each Checker Piece would contain information on:

what color it is,
its position on the board,
whether it is a king or not,
and if it has been captured.
    When still on the board this can be set to some default value indicating such, otherwise when captured it will indicate the turn it was captured, it's position unchanged, showing where it was when it was captured.

Checker Pieces can determine their possible, legal moves by checking what diagonally adjacent spaces around them are empty, or contain a piece of another color. If there are empty spaces, it must then check the direction required to move that way, which may be invalidated if it is the backward direction for the Checker Piece's given color, unless it is a king. Similarly, if there is a Checker Piece in one of the diagonally adjacent spots, the direction is valid depending on whether the Checker Piece is a king or not. It then must check if there is another Board Space diagonally adjacent in the same direction as the one under consideration, and that that Board Space is also empty. If these conditions are met, then the move is legal, and may be made.

After a move is made, a Checker Piece can check if it is on the opponent's far side of the board, based on the Checker Piece's Color, and if so, it is marked as a king.

The Board itself would be a two-dimensional array of Board Spaces. In theory the board could be of an arbitrary size assuming the number of rows and columns are the same. Each Board Space would contain:

- a flag of if it is a Board Space meant for play (dark colored spaces)
- its position within the Board Spaces array (index of the row and column),
- a flag of if it is empty,
- a Checker Piece if there is one currently on it,
- and a list of the one, two, or four possible Board Spaces that are diagonally adjacent to it.

Each Board Space can determine its diagonally adjacent spaces based on its own row column indices, as well as how those relate to the number of rows and columns that make up the whole board. If a Board Space's indices mark it as being on an edge, or corner of the Board, that indicates it has only one or two possible diagonally adjacent Board Spaces. All other Board Spaces that are flagged as being usable for play will have four diagonally adjacent Board Spaces. Unusable Board Spaces for play are flagged as such, and are otherwise ignored when determining the diagonally adjacent Board Spaces and otherwise within the game itself. If a move by a player somehow ends up on a Board Space flagged as not for play, it is immediately considered illegal.