Proposed Chess tutor Implementation

Authors

shr27, gwh18

Contents

Implementation Problems and Solutions 03/02/22

1 - Representing the game

1. *How do we represent pieces?*

A parent abstract class storing all the common attributes and methods, each type of piece has its own class which inherits from the piece class.

2. *How do we represent the board?*

A class which stores the board itself in an eight by eight, two dimensional array of objects of type piece, indexes with null objects will be treated as vacant squares.

2 - How do we find possible moves?

1. *FindMoves()*

The methods that define how pieces move should be inside the respective piece classes, they should return all of the possible board positions that the piece can be moved to in a map of coordinates.

2. *Every move not only possible moves*

These methods should not only return the legal moves, but every move the piece can perform regardless of the board state since the piece will not know the board state only its own position.

3 - Filtering for legal moves, checking for check, checkmate, and stalemate.

1. *Finding every attacked square.*

We loop over each of the opponent's pieces at the start of the turn and find their possible moves *(see 2.1)*, then filter for the legal moves, but ignoring whether making the move will put them in check. And we add each of the moves into a map of “attacked squares”.

2. *Determining check.*

To check if the player is in check we check if the king is on any of the coordinates in the “attacked square map” *(see 3.1).*

3. *Finding possible moves.*

Loop over each of the current players pieces and create a map for each of the pieces possible moves ignoring if they are legal or not *(see 2.1)*.

4. Filtering for legal moves.

For each piece we then check each of the moves in its map and filter for the legal moves in the map, relative to the state of the board, we take whether or not the player is in check when we filter for legal moves, and if the player will be in check after the move is performed by creating a projection of the state of the board after moving the piece, and then finding every attacked square in that state *(see 3.1)*, then determining check for that state *(see 3.2*) if they are still in check then we remove this coordinate from the pieces map.

5. *If we are in check.*

We call the function described in *(see 3.1)*

We should check if it’s a checkmate, we can check the map of legal moves for each piece and if they are all empty we know that there are no possible moves, therefore it's a checkmate.

6. *If we are not in check.*

We call the function described in *(see 3.1)*

If we are not in check, using the same method that checks for a checkmate we can check for stalemate by checking the map of legal moves for each piece and if they are all empty we know that there are no possible moves, therefore it's a stalemate.

7. *Displaying legal moves.*

At the start of each turn all of the above steps should be taken, so at any point the legal moves are all figured out, then we wait for the UI to ask for a piece to be selected, if a position on the board is selected it will check if there is a piece on that coordinate, if there is none then it will return back up to the UI and the process continues, if a coordinate is passed down which does have a piece on it, it can then return the map of possible moves up to the UI to display its possible moves.

Then it will wait for a square to be selected if the square shares its coordinate with one of the coordinates in the map then we will pass that coordinate down to update the position of the piece on the board and then it becomes the next player's turn, and the process repeats itself if a piece is selected which is of the same colour as the current players pieces then it will select that piece instead.