Adaptive Min-Max

Basic Implementation

The overall results of the min-max implementation have been quite good. With a depth of just three moves, the algorithm is capable of playing a player of roughly 900 points of rating.

Implementation of the heuristics

Three heuristics has been used to evaluate the position of the pieces. Information on which heuristic to implement, and a general idea on how to implement them has been gathered from this online resource that describes the Stockfish's implementation.

Negative values are assigned to the advantage of the black, and positive values are assigned to the advantage of the white, thus the difference between the two describe the total advantage from the point of view of the white player.

The implementation can be found in the file `/engine/euristics.py` of the project.

Material

This heuristic asses the material advantage of the player. The pieces values has been assigned following the AlphaZero implementation (Additional informations can be found at this link (source).

Pseudocode:

```
material_advantage = 0

for piece in white_pieces:
    material_advantage += piece_value[piece]

for piece in black_pieces:
    material_advantage -= piece_value[piece]

return material_advantage
```

Piece square table

Apart from its intrinsic value, how valuable each piece is heavily dependent on its position. For example, the knight is stronger when at the center of the board, same for the queen, while the rook is stronger when guarding the ranks opposite to its starting position. Another example is the pawn, that dramatically increases its value the closer it gets to the promotion ranks.
This heuristic tries to address the positional advantage of the pieces. The constant values used to calculate the advantage are taken from this online resource ↗.

Pseudocode:

```
positional_advantage = 0

for piece in white_pieces:
    positional_advantage += position_points[get_position(piece)]

for piece in black_pieces:
    positional_advantage -= position_points[get_position(piece)]

return positional_advantage
```

Attack

The position of a piece, other than having a value without considering other pieces, also has a value measured in the number of square controlled by each pieces. The more square a piece controls, that is, can attack, the stronger that piece is. At the same time, the number of square controlled by a piece is also an hint of the mobility of that piece. To make an example, a bishop that attacks only 4 square or less, its a trapped bishop.
This heuristic tries to assess this advantage simply counting the number of square attacked by each pieces.

Pseudocode:

```
attack_advantage = 0

for piece in white_pieces:
    attack_advantage += get_number_of_controlled_square(piece)
```

```
for piece in white_pieces:
    attack_advantage -= get_number_of_controlled_square(piece)
```

Min-Max implementation

The implementation of the min-max algorithm can be found in the file `/engine/minimax.py`.

Two techniques has been used to try to make the minimax computation quicker. The first technique is alpha-beta pruning, and the second one is saving the heuristic evaluation of each board in dictionary. However, the second technique has been later abandoned since it was generating roughly 50 Mb of data for each match.

In the exit conditions of the minimax it can be read that a negative value of 10000 (the value of the king) get assigned to the position only in the case of a gameover with a checkmate, while a positive value get assigned in case of a draw. This means that min-max algorithm values a draw equally to a loss. This change was made because in multiple situation the heuristic ended the match with a stalemate or a draw.

Advanced implementation

First part

In the first part, a model has been used to predict the heuristic evaluation of the score at depth L.

The dataset use by the model contains three features (h_1, h_2, h_3), that is, the evaluation given by the three heuristics, and the total evaluation (H) that was calculate as
$H = h_1 + 0.5 * h_2 + 0.3 * h_3$.
The three heuristic evaluation don't have the same weight since especially in the first phase of a game, the algorithm tended to sacrifice some pieces just for the positional advantage of some other pieces. It is important to note that the features in the dataset are already weight adjusted, and that means that a single record of the dataset is composed as follows:
$h_1, 0.5 * h_2, 0.3 * h_3, H$

To generate the dataset, the model and the heuristic have played as white and black for roughly 200 matches, saving the evaluation of each move. At the end of the process, the dataset reached roughly 81000 records.

The model was implemented as a Keras Regressor, and can be found in the file `engine/predictor.py`. To train the model, after each match the dataset was update with the new moves evaluated from the heuristic, and the model was trained with the updated dataset. To increase the precision of the model the dataset was scaled.
Since the dataset was generate playing with the heuristic at depth 2, theoretically the model should have been capable of predicting the value of a board at $L = 3$.

Second Part

In the second part, the min-max algorithm has been slightly modified to use the predictor instead of the heuristic as its evaluation function. This means that in theory, instead of evaluating the board at depth 3, the min-max algorithm should have been capable to predict the value of a board at depth 6.
Unfortunately, the computation time of each move was to high (around 5 minutes per move) to properly test this implementation. This probably happened because the prediction of the model was too inaccurate for the alpha-beta pruning to be effective.

Results

Out of the 200 matches played as white, the model won 0. With previous versions of the heuristics (the one that didn't distinguished draws from losses), out of 177 matches played as white, the model won 0 and drew 25 times.
I consider the training as failed, since the mean absolute error was around 700, and the mean square error was around 3 millions.
The model played apparently randomly, continuously hanging pieces, and moving the king too early in the game, resulting in a very easy win for the heuristic. To give an example, in the last match played by the model, white lost in in 15 moves. If the model played with black, results were even worst, with losses in 9 moves. To give some context, a standard chess match in 9 moves barely enters the middle game.