Fendoter

The `Fendoter` (Fendo + Roboter) class is a computer player for the fendo game.

`__init__`

Initializes a Fendoter object.

- player: int -> defines for which player Fendoter should play
- playing_method: str -> defines the search algorithm or other play-style Fendoter should deploy (For the different styles implemented go to evaluateMoves)
- search_depth: int -> the max. search depth to search the search tree of next moves

`makeMove`

Given a board returns the best move evaluated based on the given playing method.
In debug mode the search tree is printed out.

`evaluateMoves`

Executes the corresponding playing method search algorithm or play-style.
Currently implemented play-styles are:
`depth1` (old)
Chooses the best move in the next step (depth 1).
`random`
Chooses a random legal move.
`minimax`
Deploys the minimax-algorithm
`negamax`
Deploys the negamax-algorithm
`alpha-beta`
Deploys the alpha-beta pruning algorithm.

`calculateMoves`

Given an board calculates all possible next moves and returns them as a list of moves and a list of their resulting boards.
Move generation (pseudocode):

```
for pawn in currentTurnPawns:
    for field in allFields:
        # Place new Pawn
        if checkLegalMove(PlacePawn):
            duplicate Board
            apply Move to Board
            add Move to list
            add Board to list
        for direction in [North, South, East, West]:
            # Move Pawn and place Wall
            if checkLegalMove(MovePawnAndWall):
                duplicate Board
                apply Move to Board
                add Move to list
                add Board to list
            # Place Wall without moving Pawn
            if checkLegalMove(PlaceWall):
                duplicate Board
                apply Move to Board
                add Move to list
                add Board to list
```

In debug mode the transferred board is printed out.

`playRandom`

Calculates all possible next moves and chooses a random one.

`minimax`

The function is given a board, a search depth and a boolean indicating if the function is called in the maximizing step (`maximizing_player`). Minimax is a recursive function evaluating the board states from the bottom up.
At depth zero the boards get valuated and depending on which turn it is, the node above is choosing the minimal (the opponent) or the maximum (own player) grading.
The function returns the best move determined and their grading.
Next to that also a `TreeNode` is return for later examination of the search tree.
In debug mode each new board is printed with their grading.

`negamax`

Negamax is a way to implement minimax but get rid of duplicate code parts. The functionality and logic stays the same.
Instead of a boolean `maximizing_player` the integer `p` is used to switch between the opponents and own players choice of move. (1 == player, -1 == opponent). And makes it easier for calculation.

`alphabeta`

Alpha-beta pruning is a optimization of the minimax/negamax algorithm. It reduces the number of nodes evaluated in the search tree by implementing an alpha-beta window [α; β]. These values represent the minimum score that the maximizing player is assured of and the maximum score that the minimizing player is assured of. When the algorithm finds a move that leads to a value outside the alpha-beta window [α; β], it stops evaluating further moves from that node.
In this implementation, if there are multiple moves with the same grading, a random one is chosen.

`grade`

Given a board this function calculates the heuristic value for this state. The values perspective is always from the Fendoter player side.
At first the board has to evaluate its fields and thereby determine the number of conquered fields by each player. If one player wins the function returns `inf`/`-inf` depending on which player won.
The value is currently calculated like this:
$$ \mathrm{value} = a \cdot V_{Area} + f \cdot V_{Freedom} $$

The value V_{Area} is the difference between the conquered fields by Fendoter and its opponent. Scaled by a factor a.
The value $V_{Freedom}$ describes the difference between the freedom of the pawns of Fendoter and the pawns of its opponent. Scaled by a factor f.
The freedom of each pawn is estimated by looking at its neighboring fields (looking at the number of all possible moves for each pawn is too performance heavy). Depending on wether the pawn is next to a wall, other pawn or the boarder the freedom grade is reduced by 1.
In the case of bordering another pawn the freedom reduction is scaled by a factor `PAWN_BARRIER_COEF`(<1). This is because a pawn could move away again an is thereby not a hard barrier.
After calculating the freedom for the player and its opponent. The values are divided by the number of own pawns respectively and subtracted from each other.
Therefore the $V_{Freedom}$ is calculated like this: $$ V_{Freedom} = \frac{freedom_{player}}{pawns_{player}} - \frac{freedom_{opponent}}{pawns_{opponent}}$$