Java Chess

Michal Příhoda

Spse Ječná 2023-24

Tato práce se bude zabývat mým projektem vytvořit v programovacím jazyce Java hru šachy z vizuálním rozhraním a počítačovým protivníkem co za pomocí umělé inteligence vybere nejlepší tah.

Tento problém je řešen za pomocí algoritmu, který maně kolik částí:
část 1:

```
for (Move move : allMoves) {
    if (MoveIsCheckmate(board, move)) {
        return move;
    }
}
```

tato část algoritmu vráti tah pokud dostane protivníka do matu, protože takový tah je vždy nejlepší.
část 2:

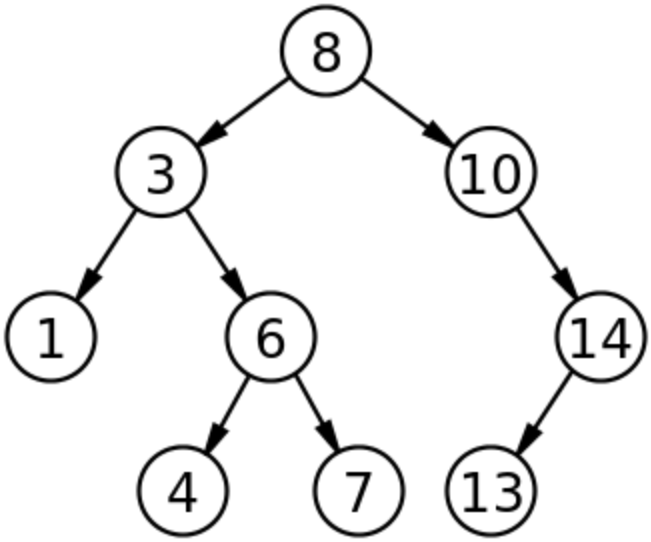
```
allMoves = allMoves.stream().sorted(Comparator.comparingInt(move -> 8 - getValueOfAtSquare(board, move.getTo()))).toList();

int bestEval = board.getSideToMove() == Side.WHITE ? Integer.MIN_VALUE : Integer.MAX_VALUE;
ArrayList<Move> bestMoves = new ArrayList<~>();
```

tato část kódu seřadí tahy podle figury, kterou potencionálně bere a nastavi evaluační proměnnou. Pro pomněnou platí že, je jí bílá strana co nejvíce zvýšit a černá co nejvíce snížit (mini max).
část 3:

```
for (Move allMove : allMoves) {
    int eval = 0;
    Move current = allMove;
    board.doMove(current);
    eval += AlphaBeta(board, this.searchDepth, Integer.MIN_VALUE, Integer.MAX_VALUE,  maximizingPlayer: board.getSideToMove() == Side.WHIT
    board.undoMove();
    if (bestMoves.isEmpty()) {
        bestMoves.add(current);
        bestEval = eval;
    } else if ((board.getSideToMove() == Side.WHITE && eval > bestEval) || (!(board.getSideToMove() == Side.WHITE) && eval < bestEval
        bestMoves.clear();
        bestMoves.add(current);
        bestEval = eval;
    } else if (eval == bestEval) {
        bestMoves.add(current);
    }
}
```

Tato část se stará o vygenerování seznamu více méně dobrých tahu za pomocí techniky tree search, kde každý tah vygeneruji všechny požné nasledující tahy a to se opakuje až se vytvroží jakýsi pomyslný strom.



*zjednodušený příklad
část 4:

```java
public int AlphaBeta(Board node, int depth, int alpha, int beta, boolean maximizingPlayer) {
    List<Move> childNodes = node.legalMoves();
    childNodes = childNodes.stream().sorted(Comparator.comparingInt(move -> 8 - getValueOfAtSquare(node, move.getTo()))).toList();
    if (depth == 0 || childNodes.isEmpty()) {
        return EvaluatePosition(node);
    }
    if (maximizingPlayer) {
        int value = Integer.MIN_VALUE;
        for (Move child : childNodes) {
            node.doMove(child);
            value = Math.max(value, AlphaBeta(node, depth: depth - 1, alpha, beta, maximizingPlayer: false));
            node.undoMove();
            alpha = Math.max(alpha, value);
            if (beta <= alpha) {
                break;
            }
        }
```

Algoritmus také využívá princip Alfa Beta pruning, kde pokud nas dana větev stromu oddálí od našeho požadovaného výsledku, tak ji ze stromu odebereme.

Pro vygenervoání možných tahu jsem využil externí knihovnu:
https://github.com/bhlangonijr/chesslib

Tento postup sice funguje, ale není optimální, protože evaluace dané pozice je určovaná jen z pomocí počtu fugur a ne jejich pozici na herním polí (pěšce do předu atd..),tedy tento, algoritmus funguje prakticky náhodně než se dostane k tehům, při kterých, může brát nějaké figury.Tento problém by se mohl vyřešit tabulko pozic prokaždou figuru, která by její hodnotu vynásobila.

Jako knihovnu pro user Interface jsem využil JavaFx, která nebyla pro tyto účely ideální, vyžaduje totiž spoustu externích knihoven a modulů, které jsou pro takto jednoduché řešení nepotřebné, také java kvůli její složitosti při operaci z polí a složitějšími datovými strukturami také není ideální.

Tento projekt je pravděpodobně poslední věc co v programovacím jazyce Java udělám, jsem rád že už to asi nebudu muset používat, ale zase jsem zklamán že tento projekt nedosáhl kvality, které jsem originálně chtěl.

Michal Příhoda
1/6/2024