Project 1

alja antni asmt

March 24, 2023

aiwdjawkudhawkduhakwudhawkudhawkudhwakudh

1 MiniMax

We use began by using a line by line translation of the alpha beta pruning Minimax seudo code from the slides. Later we changed it a bit in the name of optimisation. Here is our final result:

```
    public Position decideMove(GameState s){
        if (s.legalMoves().isEmpty())
^^I^^I^^Ireturn new Position(-1,-1);
        return ABSearch(clone(s));

    }
    private int cme;
    private Position ABSearch(GameState s){
        int size = s.getBoard().length;
        int me = s.getPlayerInTurn();
        if (pv == null || cme != me || pv[0].length != size ){
            cme = me;
            int[][] pv_plus = generatePosValue(size);
            int[][] pv_minus = new int[size][size];
            int[][] zeroes = new int[size][size];
            for (int i = 0; i<size; i++){
                for (int j = 0; j<size; j++){
                    pv_minus[i][j] = -pv_plus[i][j];
                }
            }
            pv = new int[3][][];
            pv[0] = zeroes;
            pv[me] = pv_plus;
            pv[3-me] = pv_minus;
        }
        return firstMaxValue(s, Integer.MIN_VALUE, Integer.MAX_VALUE, 6, me);
    }
```

```
private Position firstMaxValue(GameState s, int alpha, int beta, int count, int me){
    boolean fin = s.isFinished();
    if (fin || count <= 0)
        return null;
    int v = Integer.MIN_VALUE;
    Position move = null;
    var moves = s.legalMoves();
    if (moves.isEmpty())
        moves.add(new Position(-1, -1));
    int[] vs = new int[moves.size()];
    IntStream.range(0, moves.size()).parallel().forEach(i->{
        Position a = moves.get(i);
        GameState sPrime = clone(s);
        sPrime.insertToken(a);
        vs[i] = minValue(sPrime,alpha,beta, count-1, me);
    });
    for(int i = 0; i < vs.length; i++){
        if (vs[i]>v){
            v = vs[i];
            move = moves.get(i);
        }
    }
    return move;
}
private int maxValue(GameState s, int alpha, int beta, int count, int me){
    boolean fin = s.isFinished();
    if (fin || count <= 0)
        return utility(s,me, fin);
    int v = Integer.MIN_VALUE;
    var moves = s.legalMoves();
    if (moves.isEmpty())
        moves.add(new Position(-1, -1));
    for(Position a : moves){
        GameState sPrime = clone(s);
        sPrime.insertToken(a);
        int v2 = minValue(sPrime,alpha,beta, count-1, me);
        if (v2>v){
           v = v2;
           alpha = Math.max(alpha, v);
        }
        if (v >= beta)
            return v;
    }
    return v;
}
```

```
private int minValue(GameState s, int alpha, int beta, int count, int me){
    boolean fin = s.isFinished();
    if (fin || count <= 0)
        return utility(s,me, fin);
    int v = Integer.MAX_VALUE;
    var moves = s.legalMoves();
    if (moves.isEmpty())
        moves.add(new Position(-1, -1));
    for(Position a : moves){
        GameState sPrime = clone(s);
        sPrime.insertToken(a);
        int v2 = maxValue(sPrime,alpha,beta, count-1, me);
        if (v2<v){
           v = v2;
           alpha = Math.min(alpha, v);
        }
        if (v <= alpha)
            return v;
    }
    return v;
}
```

1.1 Evaluation Function

for our current utility/evaluation function we have this:

```
private int utility(GameState s, int me, boolean fin){
    int[] counts = s.countTokens();
    int diff = counts[me - 1] - counts[2-me];
    int placedTileCount = counts[0]+counts[1];
    if (fin) {
        if (diff > 0) return 1000 - placedTileCount;
        if (diff < 0) return -1000 + placedTileCount;
        return 0;
    }
    return -diff + getPostitionValues(s,me);
}
```

it consists of a couple of steps:

1. calculate the difference in the current players tokens and the other players tokens.
2. if we have reached an end state we do not care how much we win or lose by and therefor return a large fixed value moderated by the number of tiles placed to incentivise quick wins over later wins.

3. return the inverted difference in tokens plus the weighted sum of tokens on the board. each position on the board has a weigth associated with it to incentivise taking corners etc. the weights are inverted for the opponent. this incentivises having few tokens on the board to restrict the opponents choises and to get strategically important squares.

this gives us a very high win rate while having an average time to move of under a second spiking far into the game (average time is the worst on move 29 at around .8 seconds) to at most 4-5 seconds (8x8 board, search debth of 6, Random AI as opponent).

1.2 Cut-off function