B-Tree & Min-Max

Quinn Smith, David Motta, Felipe Pareja

History of Chess and Search Algorithms

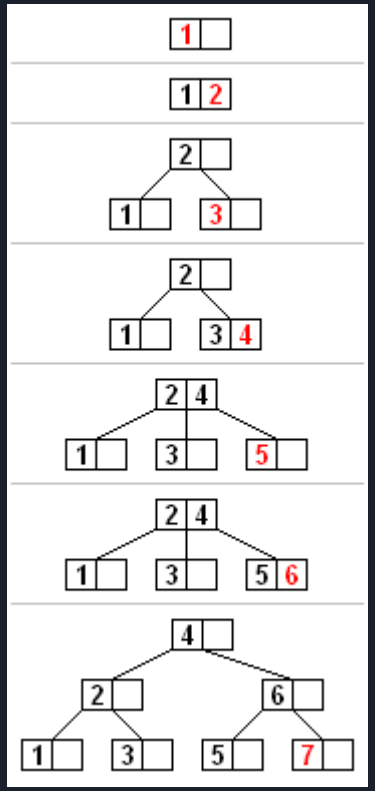
- 1950s John van Neumann classified chess in his game theory as a two-person zero-sum game with complete information that could be solved with the Min-max algorithm.
- 1965-1967 Mac Hack becomes the first computer program to defeat a human opponent
- 1979 a chess machine called Belle was developed that could reach a depth of 9
- 1985 Development on Supercomputer Deep Blue begins
- 1997 Deep Blue defeats Chess World Champion Gary Kasparov

B-Tree

Self-balancing tree data structure that allows searches, accesses, insertions, and deletions in logarithmic time complexity O(log n)

Can have more than two nodes

Well suited for storage systems such as databases and file systems



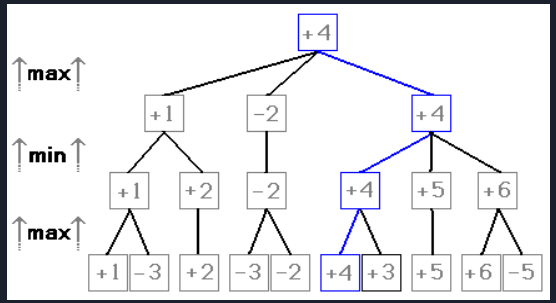
Min-Max

Utilized in two player games

Recursive and uses the time complexity O(b^d)

The values at each node represent how good a game move is.

The algorithm calculates moves that would generate the highest value for them, while also predicting the countermoves of the opponent that would produce the minimum value for them



Determination of Value in Chess

Each unique group of pieces has a different point value in terms of their importance in gaining a tactical advantage.

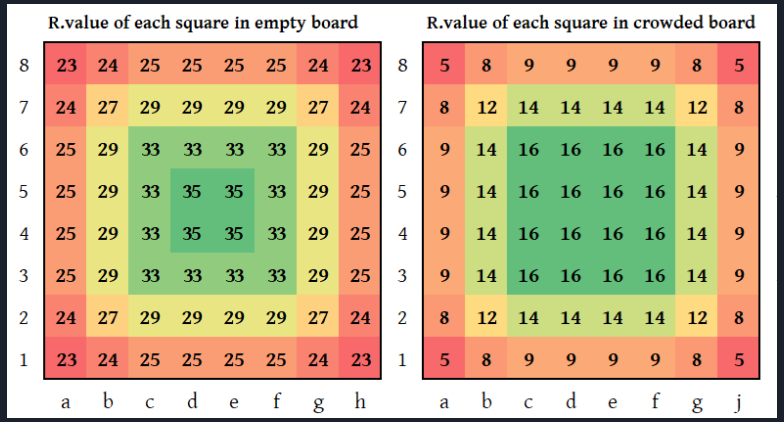
The relative strength of pieces is as follows: pawn, knight, bishop (knight & bishop are interchangeable), rook, queen, and king.

```
int pos_eval(char piece) {
    switch (piece) {
    case 'r': return -5;
    case 'n': return -3;
    case 'b': return -3;
    case 'q': return -9;
    case 'k': return -50;
    case 'p': return -1;
    case 'P': return 1;
    case 'R': return 5;
    case 'N': return 3;
    case 'B': return 3;
    case 'Q': return 9;
    case 'K': return 50;
    }
    return 0;
}
```

Determination of value in Chess 2

Estimation functions utilize heuristics. Heuristics represent the knowledge that we have about the chess game.

In chess the positions at the center of the board are generally the most valuable



R.value of each square in empty board

	a	b	c	d	e	f	g	h
8	23	24	25	25	25	25	24	23
7	24	27	29	29	29	29	27	24
6	25	29	33	33	33	33	29	25
5	25	29	33	35	35	33	29	25
4	25	29	33	35	35	33	29	25
3	25	29	33	33	33	33	29	25
2	24	27	29	29	29	29	27	24
1	23	24	25	25	25	25	24	23

R.value of each square in crowded board

	a	b	c	d	e	f	g	j
8	5	8	9	9	9	9	8	5
7	8	12	14	14	14	14	12	8
6	9	14	16	16	16	16	14	9
5	9	14	16	16	16	16	14	9
4	9	14	16	16	16	16	14	9
3	9	14	16	16	16	16	14	9
2	8	12	14	14	14	14	12	8
1	5	8	9	9	9	9	8	5

Insertion method

```
void Tree::insert(Node* node){
    if(this->root == nullptr){
        this->root = node;
        return;
    }
    this->root->insert(node);

    return;
    //this->root->nexts.push_back(node);
}
```

Checking positions if available

```
bool is_free(int row, int col){
    if(row < 0 || row > 7 || col < 0 || col > 7){
        return false;
    }
    if(board[row][col] == 0){
        return true;
    }
    return false;
}

bool is_takeable(int row, int col){
    if(row < 0 || row > 7 || col < 0 || col > 7){
        return false;
    }
    char space = board[row][col];
    if(space == 0 || space == 'p' || space == 'r' || space == 'n' || space == 'b' || space == 'q' || space == 'k'){
        return true;
    }
    return false;
}
```

Creating vector with move and score

```
std::vector<int> add_position(int x, int y, int xx, int yy){
    std::vector<int> temp;

    char car = board[xx][yy];
    board[xx][yy] = board[x][y];
    board[x][y] = 0;
    int score = 0;
    for(int i = 0; i < 8; i++){
        for(int j = 0; j < 8; j++){
            score += pos_eval(board[i][j]);
        }
    }
    temp.push_back(x);
    temp.push_back(y);
    temp.push_back(xx);
    temp.push_back(yy);

    temp.push_back(score);

    board[x][y] = board[xx][yy];
    board[xx][yy] = car;

    return temp;
}
```

Implementation

```
//the logic behind black's moves.
void Board::black_moves(){
    //a vector or list that will hold all possible moves IN NODE FORM.

    //Position pos;
    std::vector<std::vector<int>> moves = find_all_moves();


    Tree moveTree;

    for(int i = 0; i < moves.size(); i++){
        Node * node = new Node(moves[i]);
        moveTree.insert(node);
    }

    int maxScore = -1000;
    int maxIndex = rand() % moves.size();

    moveTree.print();

    std::vector<int> bestMove = moveTree.get_best_move();
    board[bestMove[2]][bestMove[3]] = board[bestMove[0]][bestMove[1]];
    board[bestMove[0]][bestMove[1]] = 0;

}
```

Implementation 2

```
std::vector<std::vector<int>> Board::find_all_moves(){
    //positional score:
    //std::cout << "Score: " << score << "\n"; //Generate nodes for each possible move, store them in vector
    std::vector<std::vector<int>> moves;
     for(int i = 0; i < 8; i++){
        for(int j = 0; j < 8; j++){
            if(board[i][j] != 0){
                switch(board[i][j]){
                    case 'P':
                        if(is_free(i+1,j)){
                            //std::vector<int> temp = ;
                            moves.push_back(add_position(i,j,i+1,j));
                        }
                        if(!is_free(i+1,j+1) && is_takeable(i+1,j+1)){
                            moves.push_back(add_position(i,j,i+1,j+1));
                        }
                        if(!is_free(i+1,j-1) && is_takeable(i+1,j-1)){
                            moves.push_back(add_position(i,j,i+1,j-1));
                        }
                        break;
                    case 'R': //ROOK MOVES
                        for(int dis = 1; dis < 8; dis++){
                            if(is_takeable(i+dis,j)){
                                moves.push_back(add_position(i,j,i+dis,j));
                                if(!is_free(i+dis,j)){
                                    break;
                                }
                            }
                            else{
                                break;
                            }
                        }
                        for(int dis = 1; dis < 8; dis++){
                            if(is_takeable(i-dis,j)){
                                moves.push_back(add_position(i,j,i-dis,j));
                                if(!is_free(i-dis,j)){
                                    break;
                                }
                            }
                            else{
                                break;
                            }
                        }
                        for(int dis = 1; dis < 8; dis++){
                            if(is_takeable(i,i-dis)){
```

Questions?

Quinn Smith, David Motta, Felipe Pareja