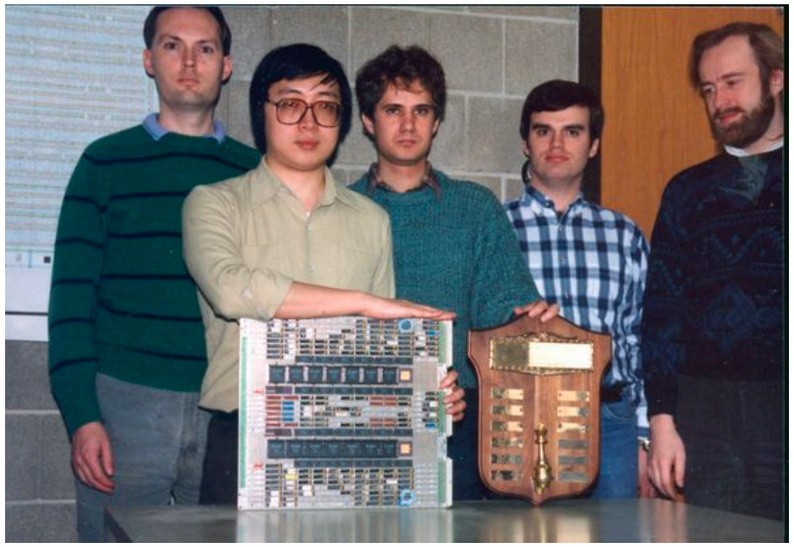
Deep Blue

Artificial Intelligence, 2002
Murray Campbell, A. Joseph Hoane Jr., Feng-hsiung Hsu

Presented by David Bellamy


“Play the opening like a book, the middle game like a magician, and the endgame like a machine.”
Rudolf Spielmann



CMU, 1980 (left to right): Murray Campbell, Feng-hsiung Hsu, Thomas Anantharaman, Mike Browne and Andreas Nowatzyk

Deep Blue’s predecessors: ChipTest and later called Deep Thought was made at CMU in the 80s, started as a PhD project of the computer science doctoral students Feng-hsiung Hsu and Thomas Anantharaman.

- Search speed ~500k positions/s (ChipTest), ~700k positions/s (Deep Thought)
- Deep Thought was the first engine to beat a GM in tournament play, 1988
- They made a bigger and better version 2, with the biggest success being a 3-1 win against the Danish national team, 1993
- In 89-90, the CMU team moved to IBM. Deep Thought was renamed to Deep Blue

Deep Blue I was developed at IBM Research in the mid 90s

- Overall search speed: 50-100 million chess positions/s
- Evaluation function: ~6400 features
- Played a tournament match of 6 games against against Garry Kasparov.

Kasparov won 4-2, but it was tied 2-2 after 4 games. Feb 1996. But they would play again the next year.

Deep Blue II:
Completely redesigned evaluation function: 8000 features (+1600 from Deep Blue I).
Had a “specialized move generator” like “generate all moves that attack the opponent’s pieces” – in reality this is ambiguous.
Overall search speed: 100-200 million chess positions/s (twice as fast as Deep Blue I)
Software for match preparation, tuning the evaluation function for a specific opponent.
Beat Kasparov 3.5-2.5 in 1997.

It was a spectator event!

Link here: https://www.youtube.com/watch?v=8RVa0THWUWw

Agenda

- Chess tutorial
- The size of the chess game tree
- How strong are chess engines, anyway?
- Some basics of search
- A look at the Deep Blue II system

Rules of the game: piece movement, check vs. checkmate, castling long vs. short, en passant, pawn promotion, 3-fold repetition, stalemate, and 50 move draw rule.
    Kasparov and Deep Blue agreed to a draw because of an incoming 3-fold repetition due to perpetual check

Game phases
    Openings, opening principles, and opening books.
    Middle games, tactics and plans.
    Endgames, technique, and theoretical draws.

King and pawn: winning vs. drawn

Fundamental skills: evaluation & calculation

Rating systems, titles, and how engines compare to humans

Chess evaluation language

- Chess piece value:
  - Pawns: 1
  - Minor pieces (bishops, knights): 3
  - Major/heavy pieces (queens and rooks): rooks = 5, queen = 9
  - King = undefined. Cannot be captured. Chess engines commonly assign 200.
    - In the endgame, the fighting value of a king is about 4 points (ex. Equal to a bishop and a pawn). It is better at attacking pieces than bishops (which can only see one color) and better at defending than knights (which can take several moves to re-route to reach the defense).

Chess evaluation language

- = Even position
- +/= Slight advantage for white
- =/+ Slight advantage for black
- +/- Clear advantage for white
- -/+ Clear advantage for black
- +- Decisive advantage for white
- -+ Decisive advantage for black
- Infinity: Unclear position

When will chess be solved?

And what does it mean to solve a game?

One definition of a solved game is a game that, when engaged in by players exhibiting “perfect play”, there is one and only one possible outcome. What is that outcome for chess? Is it white winning every time? Draw? Black winning?

But what is perfect play? What you’ll read is that perfect play is making the best move in every position.
But what is the best move?? I used to think that the best move should be defined as the move with the highest proportion of consequent branches in the game tree that result in wins.

When will chess be solved?

And what does it mean to solve a game?

- Wikipedia's page on Solved game: "a game whose outcome (win, lose or draw) can be correctly predicted from any position, assuming that both players **play perfectly.**"

When will chess be solved?

And what does it mean to solve a game?

- Wikipedia's page on Solved game: "a game whose outcome (win, lose or draw) can be correctly predicted from any position, assuming that both players **play perfectly.**"
- But what does perfect play mean?

When will chess be solved?

And what does it mean to solve a game?

- Wikipedia's page on Solved game: "a game whose outcome (win, lose or draw) can be correctly predicted from any position, assuming that both players **play perfectly.**"
- But what does perfect play mean?
  - "Playing the best move in any position."

When will chess be solved?

And what does it mean to solve a game?

- Wikipedia's page on Solved game: "a game whose outcome (win, lose or draw) can be correctly predicted from any position, assuming that both players **play perfectly.**"
- But what does perfect play mean?
  - "Playing the best move in any position."
    - But what is the "best move"?

When will chess be solved?

And what does it mean to solve a game?

- Wikipedia's page on Solved game: "a game whose outcome (win, lose or draw) can be correctly predicted from any position, assuming that both players **play perfectly.**"
- But what does perfect play mean?
  - "Playing the best move in any position."
    - But what is the "best move"?
      - Reasonable definition: the move that leads to the greatest probability of winning

When will chess be solved?

And what does it mean to solve a game?

- Wikipedia's page on Solved game: "a game whose outcome (win, lose or draw) can be correctly predicted from any position, assuming that both players **play perfectly.**"
- But what does perfect play mean?
  - "Playing the best move in any position."
    - But what is the "best move"?
      - Reasonable definition: the move that leads to the greatest probability of winning
      - How can we calculate the probability of winning from a given position?

When will chess be solved?

And what does it mean to solve a game?

- Wikipedia's page on Solved game: "a game whose outcome (win, lose or draw) can be correctly predicted from any position, assuming that both players **play perfectly.**"
- But what does perfect play mean?
  - "Playing the best move in any position."
    - But what is the "best move"?
      - Reasonable definition: the move that leads to the greatest probability of winning
      - How can we calculate the probability of winning from a given position?
        - The proportion of branches in the complete game tree (from that point onwards) that result in victory.

When will chess be solved?

And what does it mean to solve a game?

- Wikipedia's page on Solved game: "a game whose outcome (win, lose or draw) can be correctly predicted from any position, assuming that both players **play perfectly.**"
- But what does perfect play mean?
  - "Playing the best move in any position."
    - But what is the "best move"?
      - Reasonable definition: the move that leads to the greatest probability of winning
      - How can we calculate the probability of winning from a given position?
        - The proportion of branches in the complete game tree (from that point onwards) that result in victory.
    - **Wrong!**

When will chess be solved?

And what does it mean to solve a game?

- Wikipedia's page on Solved game: "a game whose outcome (win, lose or draw) can be correctly predicted from any position, assuming that both players **play perfectly.**"
- But what does perfect play mean?
  - "Playing the best move in any position."
    - But what is the "best move"?
      - Reasonable definition: the move that leads to the greatest probability of winning
      - How can we calculate the probability of winning from a given position?
        - The proportion of branches in the complete game tree (from that point onwards) that result in victory.
    - **Wrong!**
- Solved = the entire game tree has been traversed and the move that leads to "the one deterministic outcome" (ex. Draw for Tic-Tac-Toe) is known for every position, for both sides.

When will chess be solved?

And what does it mean to solve a game?

- Wikipedia's page on Solved game: "a game whose outcome (win, lose or draw) can be correctly predicted from any position, assuming that both players **play perfectly.**"
- But what does perfect play mean?
  - "Playing the best move in any position."
    - But what is the "best move"?
      - Reasonable definition: the move that leads to the greatest probability of winning
      - How can we calculate the probability of winning from a given position?
        - The proportion of branches in the complete game tree (from that point onwards) that result in victory.
    - **Wrong!**
- Solved = the entire game tree has been traversed and the move that leads to "the one deterministic outcome" (ex. Draw for Tic-Tac-Toe) is known for every position, for both sides.
- The Shannon number: a **conservative** lower-bound of the game-tree complexity of chess is 10^120 !!

When will chess be solved?

And what does it mean to solve a game?

- Wikipedia's page on Solved game: "a game whose outcome (win, lose or draw) can be correctly predicted from any position, assuming that both players **play perfectly.**"
- But what does perfect play mean?
  - "Playing the best move in any position."
    - But what is the "best move"?
      - Reasonable definition: the move that leads to the greatest probability of winning
      - How can we calculate the probability of winning from a given position?
        - The proportion of branches in the complete game tree (from that point onwards) that result in victory.
    - **Wrong!**
- Solved = the entire game tree has been traversed and the move that leads to "the one deterministic outcome" (ex. Draw for Tic-Tac-Toe) is known for every position, for both sides.
- The Shannon number: a **conservative** lower-bound of the game-tree complexity of chess is 10^120 !!
  - Back of the envelope: each pair of moves has ~1000 possibilities. An average game is 40 moves. (10^3)^40) = 10^120

When will chess be solved?

And what does it mean to solve a game?

- Wikipedia's page on Solved game: "a game whose outcome (win, lose or draw) can be correctly predicted from any position, assuming that both players **play perfectly.**"
- But what does perfect play mean?
  - "Playing the best move in any position."
    - But what is the "best move"?
      - Reasonable definition: the move that leads to the greatest probability of winning
      - How can we calculate the probability of winning from a given position?
        - The proportion of branches in the complete game tree (from that point onwards) that result in victory.
    - **Wrong!**
- Solved = the entire game tree has been traversed and the move that leads to "the one deterministic outcome" (ex. Draw for Tic-Tac-Toe) is known for every position, for both sides.
- The Shannon number: a **conservative** lower-bound of the game-tree complexity of chess is 10^120 !!
  - Back of the envelope: each pair of moves has ~1000 possibilities. An average game is 40 moves. (10^3)^40) = 10^120

- Chess computers can process ~100 x 10^6 positions per second (10^8).

When will chess be solved?

And what does it mean to solve a game?

- Wikipedia's page on Solved game: "a game whose outcome (win, lose or draw) can be correctly predicted from any position, assuming that both players **play perfectly.**"
- But what does perfect play mean?
  - "Playing the best move in any position."
    - But what is the "best move"?
      - Reasonable definition: the move that leads to the greatest probability of winning
      - How can we calculate the probability of winning from a given position?
        - The proportion of branches in the complete game tree (from that point onwards) that result in victory.
    - **Wrong!**
- Solved = the entire game tree has been traversed and the move that leads to "the one deterministic outcome" (ex. Draw for Tic-Tac-Toe) is known for every position, for both sides.
- The Shannon number: a **conservative** lower-bound of the game-tree complexity of chess is 10^120 !!
  - Back of the envelope: each pair of moves has ~1000 possibilities. An average game is 40 moves. (10^3)^40) = 10^120

- Chess computers can process ~100 x 10^6 positions per second (10^8).
- This would require 10^112 seconds to traverse the game tree. Over 3 x 10^101 millennia.

When will chess be solved?

And what does it mean to solve a game?

- Wikipedia's page on Solved game: "a game whose outcome (win, lose or draw) can be correctly predicted from any position, assuming that both players **play perfectly.**"
- But what does perfect play mean?
  - "Playing the best move in any position."
    - But what is the "best move"?
      - Reasonable definition: the move that leads to the greatest probability of winning
      - How can we calculate the probability of winning from a given position?
        - The proportion of branches in the complete game tree (from that point onwards) that result in victory.
    - **Wrong!**
- Solved = the entire game tree has been traversed and the move that leads to "the one deterministic outcome" (ex. Draw for Tic-Tac-Toe) is known for every position, for both sides.
- The Shannon number: a **conservative** lower-bound of the game-tree complexity of chess is 10^120 !!
  - Back of the envelope: each pair of moves has ~1000 possibilities. An average game is 40 moves. (10^3)^40) = 10^120

- Chess computers can process ~100 x 10^6 positions per second (10^8).
- This would require 10^112 seconds to traverse the game tree. Over 3 x 10^101 millennia.
- Even with speeds that are 1 trillion times faster, (10^20 positions/s) -> 3 x 10^89 millennia vs. 5 x 10^6 millennia until our sun dies.

When will chess be solved?

And what does it mean to solve a game?

- Wikipedia's page on Solved game: "a game whose outcome (win, lose or draw) can be correctly predicted from any position, assuming that both players **play perfectly.**"
- But what does perfect play mean?
  - "Playing the best move in any position."
    - But what is the "best move"?
      - Reasonable definition: the move that leads to the greatest probability of winning
      - How can we calculate the probability of winning from a given position?
        - The proportion of branches in the complete game tree (from that point onwards) that result in victory.
    - **Wrong!**
- Solved = the entire game tree has been traversed and the move that leads to "the one deterministic outcome" (ex. Draw for Tic-Tac-Toe) is known for every position, for both sides.
- The Shannon number: a **conservative** lower-bound of the game-tree complexity of chess is 10^120 !!
  - Back of the envelope: each pair of moves has ~1000 possibilities. An average game is 40 moves. (10^3)^40) = 10^120

- Chess computers can process ~100 x 10^6 positions per second (10^8).
- This would require 10^112 seconds to traverse the game tree. Over 3 x 10^101 millennia.
- Even with speeds that are 1 trillion times faster, (10^20 positions/s) -> 3 x 10^89 millennia vs. 5 x 10^6 millennia until our sun dies.

Chess engine ELOs

How strong are they, really?

Chess engine ELOs

How strong are they, really?

- FIDE ratings:

Chess engine ELOs

How strong are they, really?

- FIDE ratings:
  - “Super” grandmaster: 2700

Chess engine ELOs

How strong are they, really?

- FIDE ratings:
  - “Super” grandmaster: 2700
  - Grandmaster: 2500

Chess engine ELOs

How strong are they, really?

- FIDE ratings:
  - “Super” grandmaster: 2700
  - Grandmaster: 2500
  - International Master: 2400

Chess engine ELOs

How strong are they, really?

- FIDE ratings:
  - “Super” grandmaster: 2700
  - Grandmaster: 2500
  - International Master: 2400
  - Master: 2300

Chess engine ELOs

How strong are they, really?

- FIDE ratings:
  - “Super” grandmaster: 2700
  - Grandmaster: 2500
  - International Master: 2400
  - Master: 2300
  - Average tournament player: ~1500

Chess engine ELOs

How strong are they, really?

- FIDE ratings:
  - “Super” grandmaster: 2700
  - Grandmaster: 2500
  - International Master: 2400
  - Master: 2300
  - Average tournament player: ~1500
  - ~3500 rated chess engines today vs. 2800 Magnus Carlsen

Chess engine ELOs

How strong are they, really?

- FIDE ratings:
  - “Super” grandmaster: 2700
  - Grandmaster: 2500
  - International Master: 2400
  - Master: 2300
  - Average tournament player: ~1500
  - ~3500 rated chess engines today vs. 2800 Magnus Carlsen
- What does a 100 point differential between players mean?

Chess engine ELOs

How strong are they, really?

- FIDE ratings:
  - “Super” grandmaster: 2700
  - Grandmaster: 2500
  - International Master: 2400
  - Master: 2300
  - Average tournament player: ~1500
  - ~3500 rated chess engines today vs. 2800 Magnus Carlsen
- What does a 100 point differential between players mean?
  - Based on the function 1 / (1 + 10^(diff/400)), so a kind of logistic curve.

Chess engine ELOs

How strong are they, really?

- FIDE ratings:
  - “Super” grandmaster: 2700
  - Grandmaster: 2500
  - International Master: 2400
  - Master: 2300
  - Average tournament player: ~1500
  - ~3500 rated chess engines today vs. 2800 Magnus Carlsen
- What does a 100 point differential between players mean?
  - Based on the function 1 / (1 + 10^(diff/400)), so a kind of logistic curve.
  - +100 pts: 64% to 36%

Chess engine ELOs

How strong are they, really?

- FIDE ratings:
  - “Super” grandmaster: 2700
  - Grandmaster: 2500
  - International Master: 2400
  - Master: 2300
  - Average tournament player: ~1500
  - ~3500 rated chess engines today vs. 2800 Magnus Carlsen
- What does a 100 point differential between players mean?
  - Based on the function 1 / (1 + 10^(diff/400)), so a kind of logistic curve.
  - +100 pts: 64% to 36%
  - +500 pts: 96% to 4%

Chess engine ELOs

How strong are they, really?

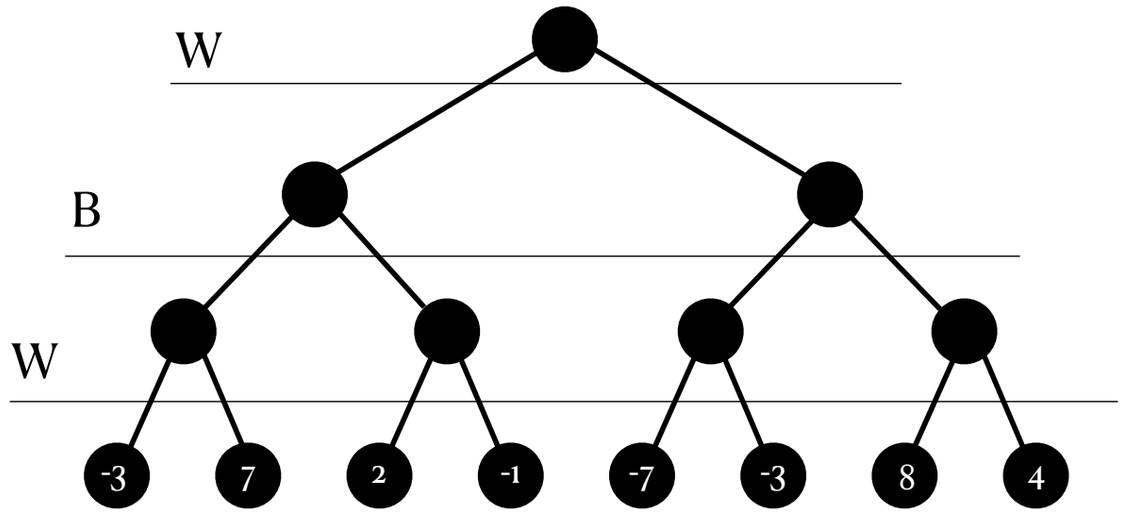
- FIDE ratings:
  - “Super” grandmaster: 2700
  - Grandmaster: 2500
  - International Master: 2400
  - Master: 2300
  - Average tournament player: ~1500
  - ~3500 rated chess engines today vs. 2800 Magnus Carlsen
- What does a 100 point differential between players mean?
  - Based on the function 1 / (1 + 10^(diff/400)), so a kind of logistic curve.
  - +100 pts: 64% to 36%
  - +500 pts: 96% to 4%
  - +700 pts: 99.3% to 0.7%

The basics of search: minimax

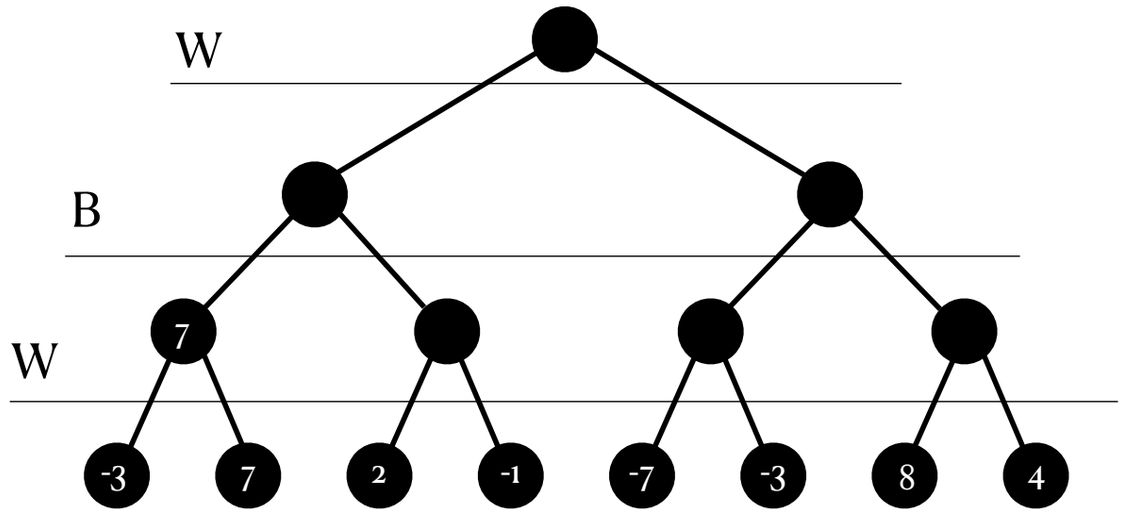
- Idea: “Always minimize the maximum pay-off for the opponent”
  - Equivalently: always minimize the worst-case scenario for yourself.
  - Equivalent by symmetry: maximize your own minimum payoff.
- The minimax theorem:
  - For every two-person, zero-sum game with finitely many strategies, there exists a value V and a mixed strategy for each player, such that
    - (a) Given player 2's strategy, the best payoff possible for player 1 is V, and
    - (b) Given player 1's strategy, the best payoff possible for player 2 is –V.

The history of AI has intricate connections to the paradigm referred to as “reasoning as search” – the idea that intelligence is the manifestation of a search process.

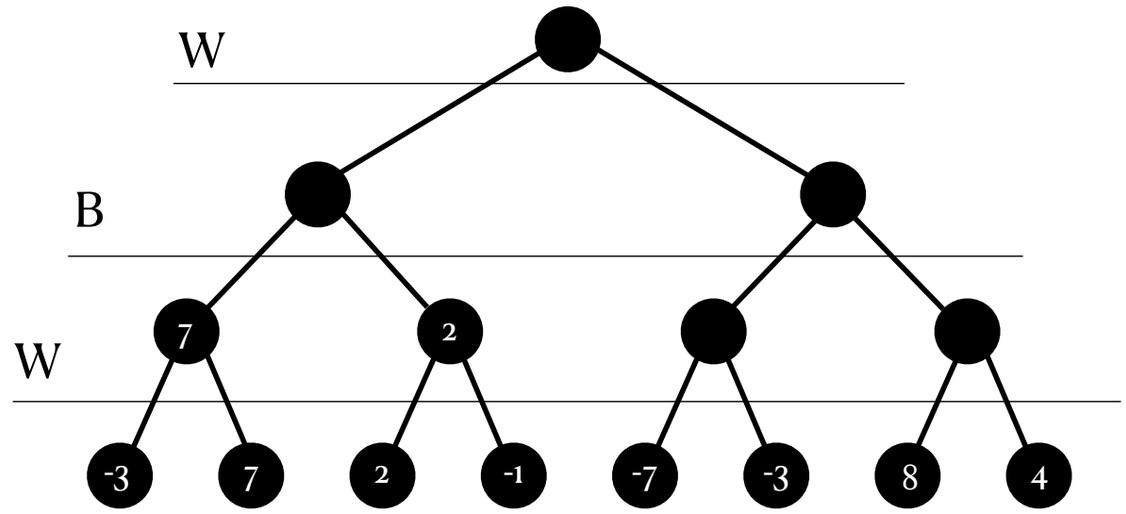
The basics of search: minimax



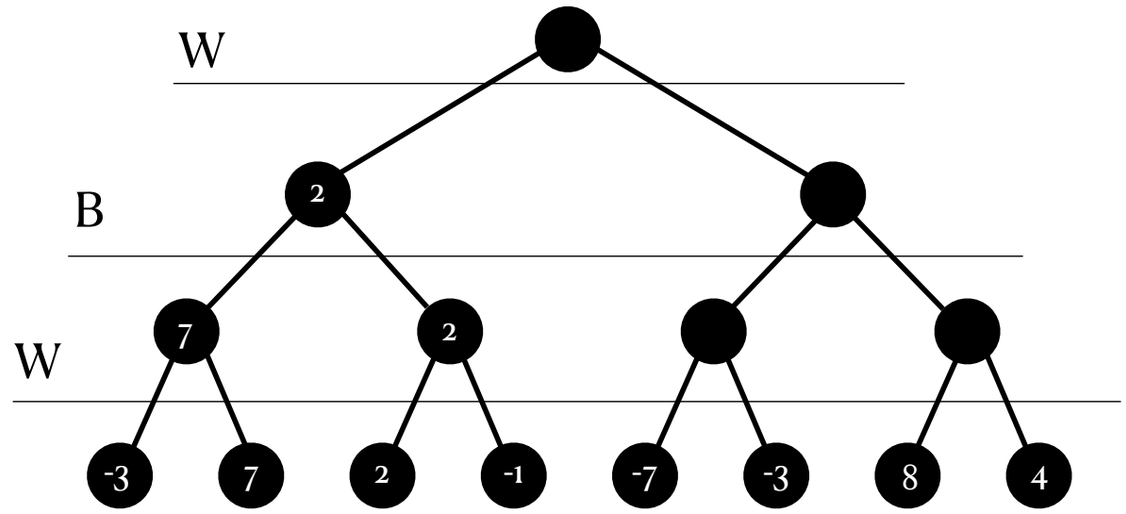
The basics of search: minimax



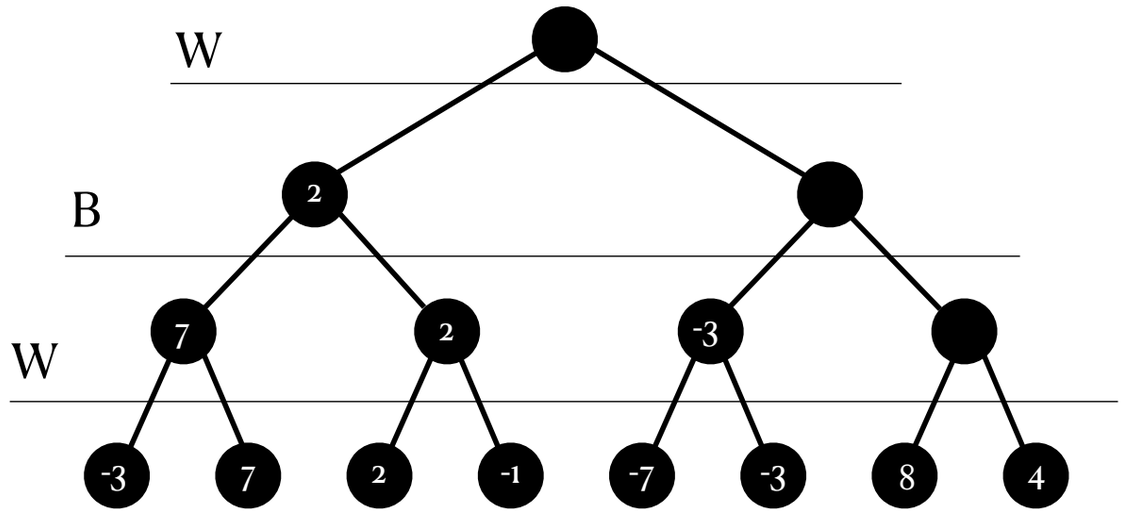
The basics of search: minimax



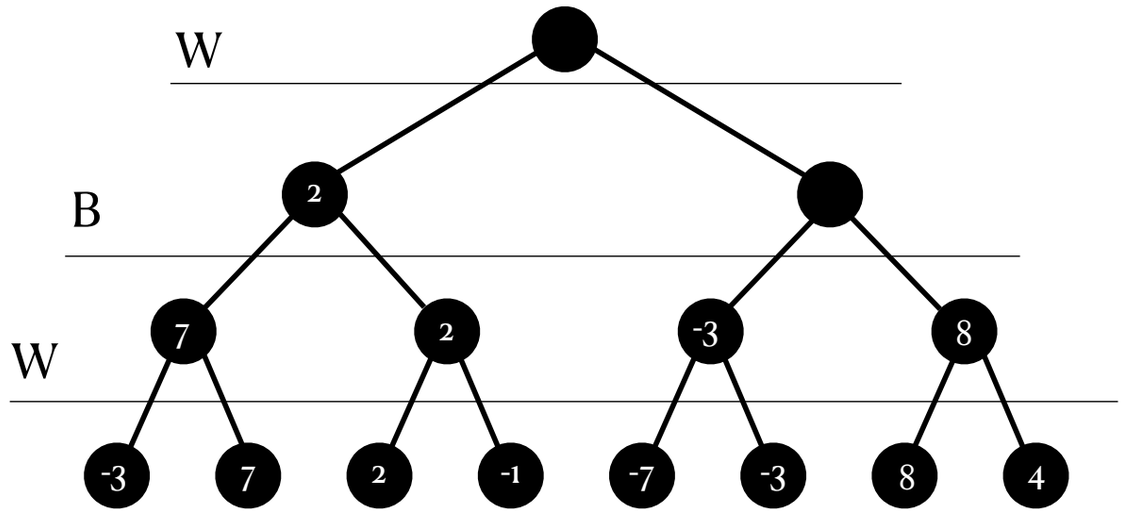
The basics of search: minimax



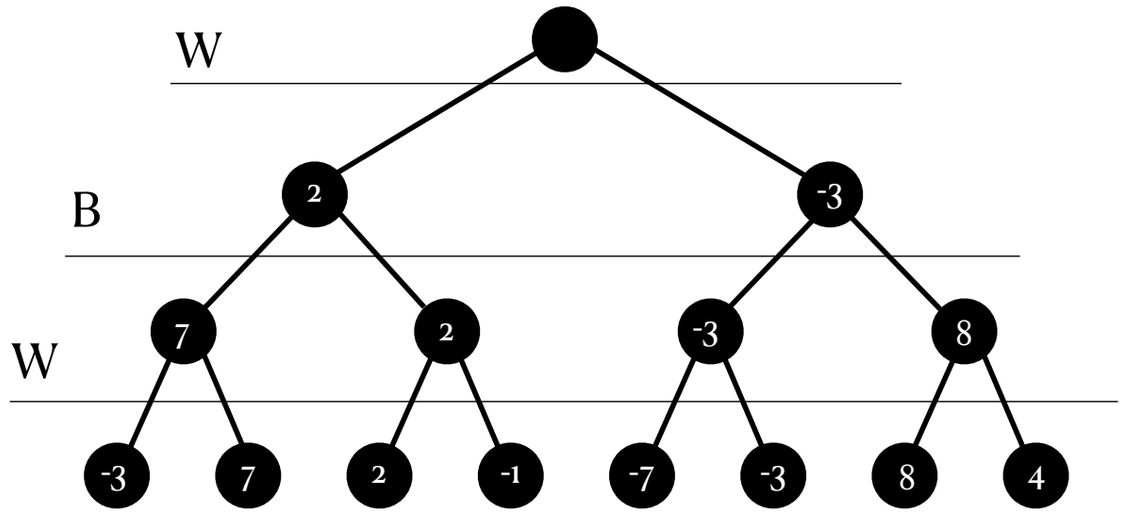
The basics of search: minimax



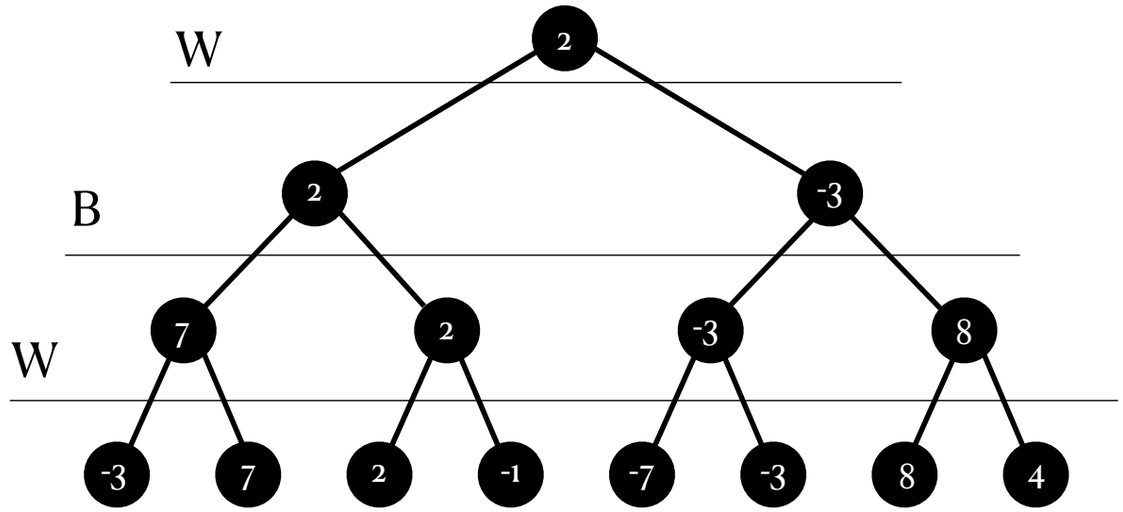
The basics of search: minimax



The basics of search: minimax



The basics of search: minimax



The basics of search: minimax

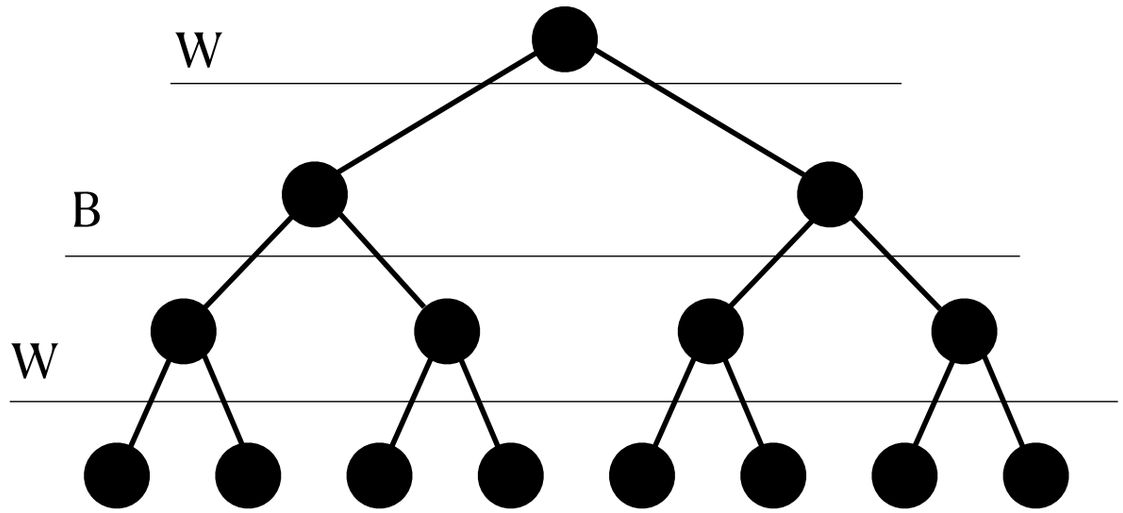
```
function minimax(position, depth, maximizingPlayer)
  if depth == 0 or game over in position
    return static evaluation of position

  if maximizingPlayer
    maxEval = -infinity
    for each child of position
      eval = minimax(child, depth - 1, false)
      maxEval = max(maxEval, eval)
    return maxEval

  else
    minEval = +infinity
    for each child of position
      eval = minimax(child, depth - 1, true)
      minEval = min(minEval, eval)
    return minEval
```

The basics of search: alpha-beta pruning

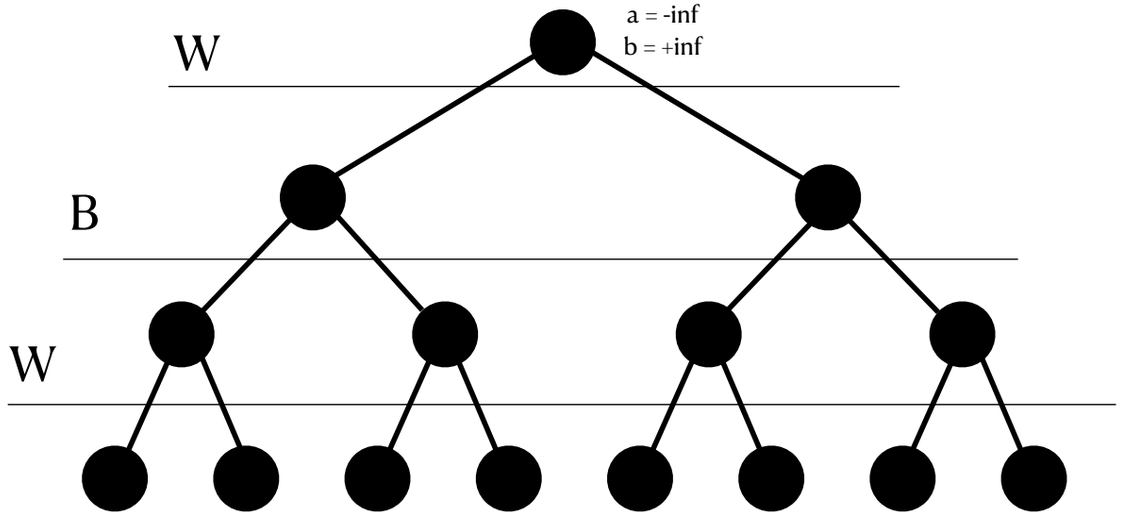
- Idea: “Don’t do work that you don’t have to do.”
  - Equivalently: only apply the minimax algorithm to nodes that have the possibility of affecting the outcome, V.
- Returns the same value as minimax, but with fewer steps (equal in the worst case).
- Alpha tracks the worst possible outcome for white, beta tracks the worst possible outcome for black.

Improvements on this: null-window, Judea Pearl

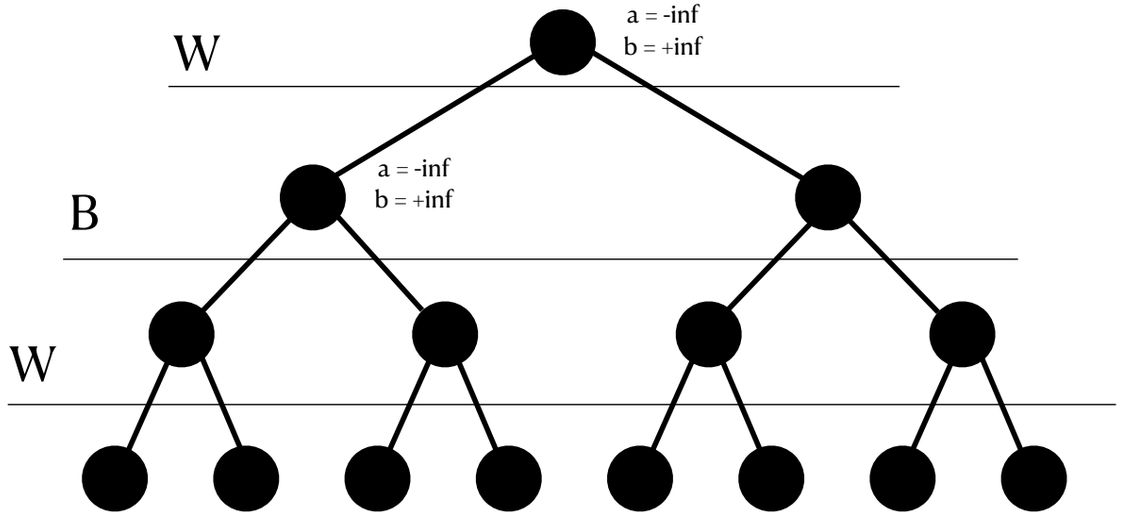
The basics of search: alpha-beta pruning



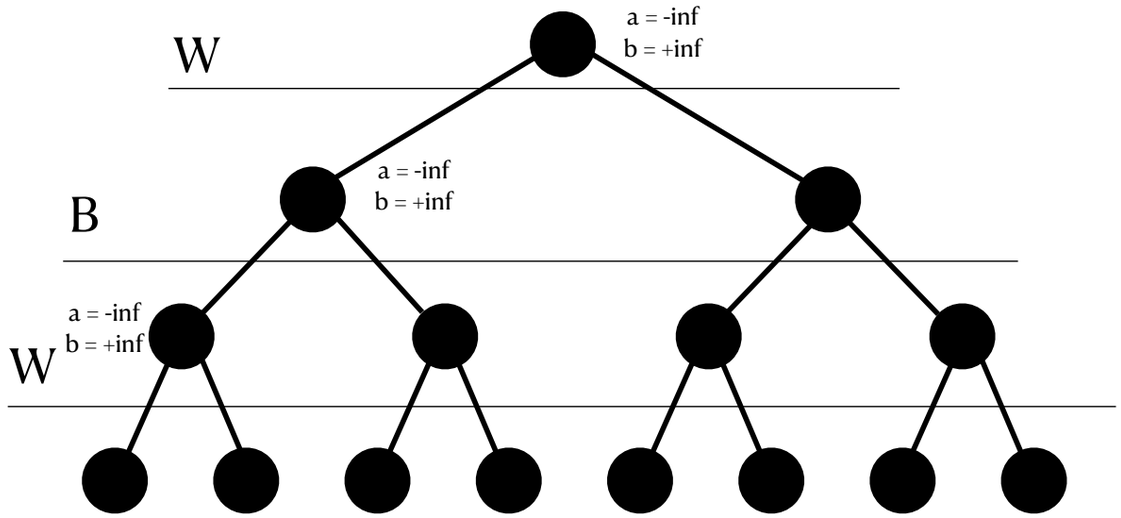
The basics of search: alpha-beta pruning



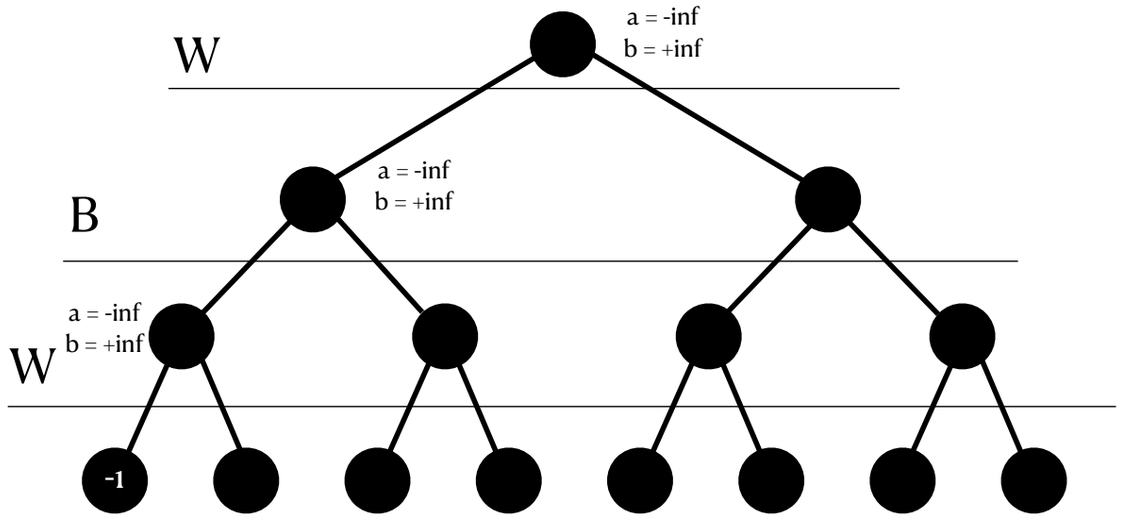
The basics of search: alpha-beta pruning



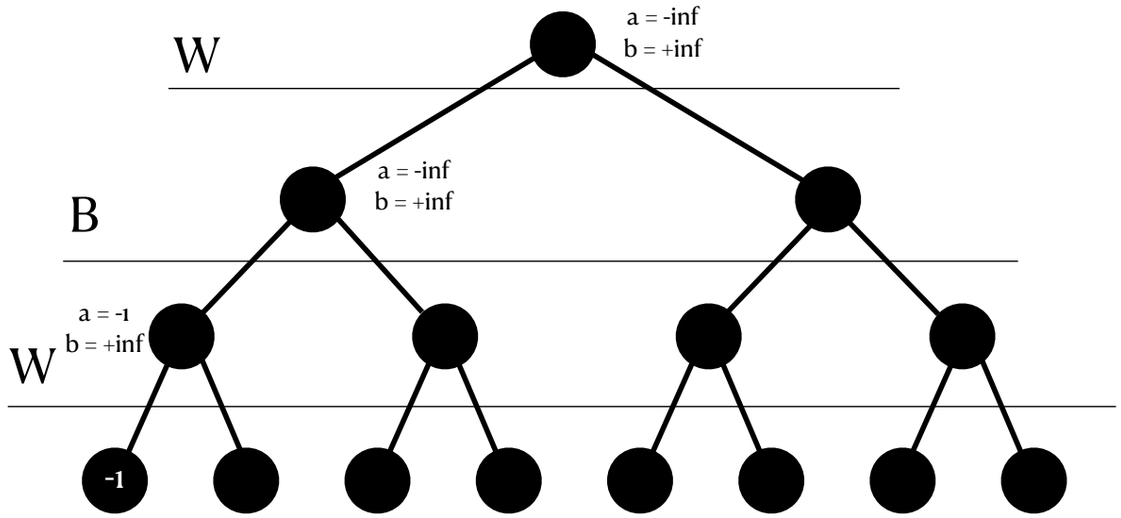
The basics of search: alpha-beta pruning



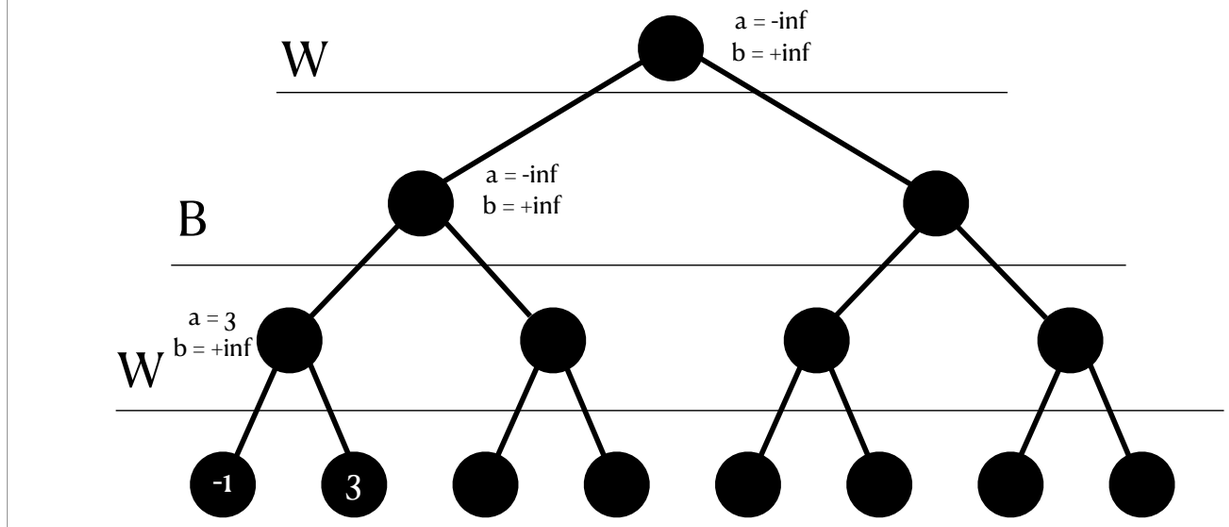
The basics of search: alpha-beta pruning



The basics of search: alpha-beta pruning

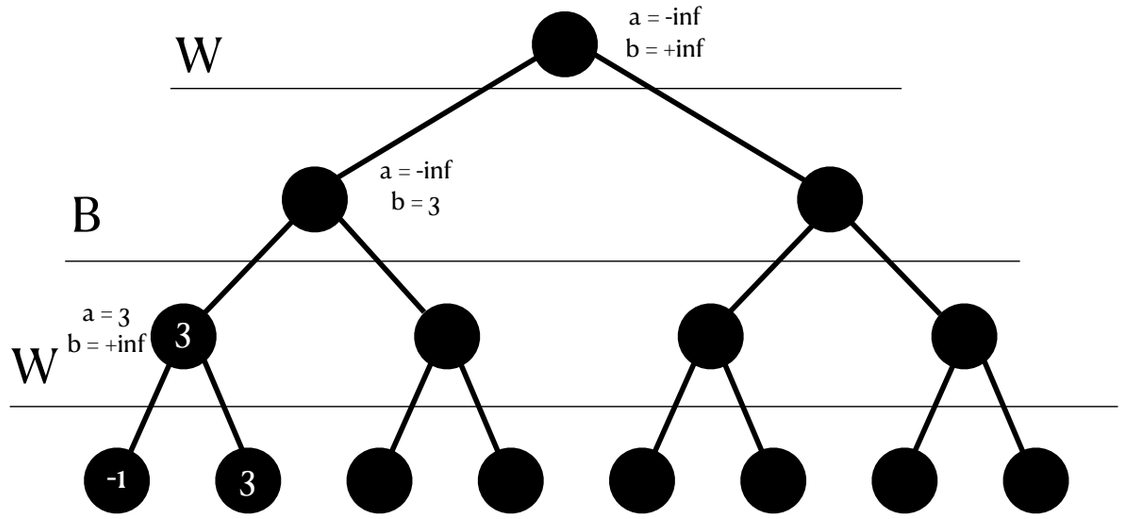


The basics of search: alpha-beta pruning

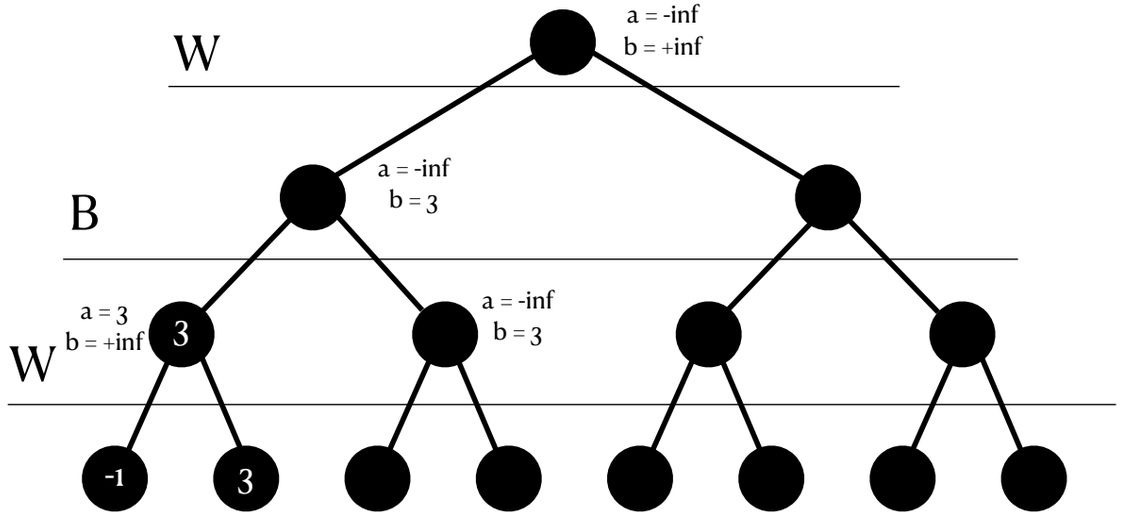


Beta is not less than alpha so no pruning occurs.

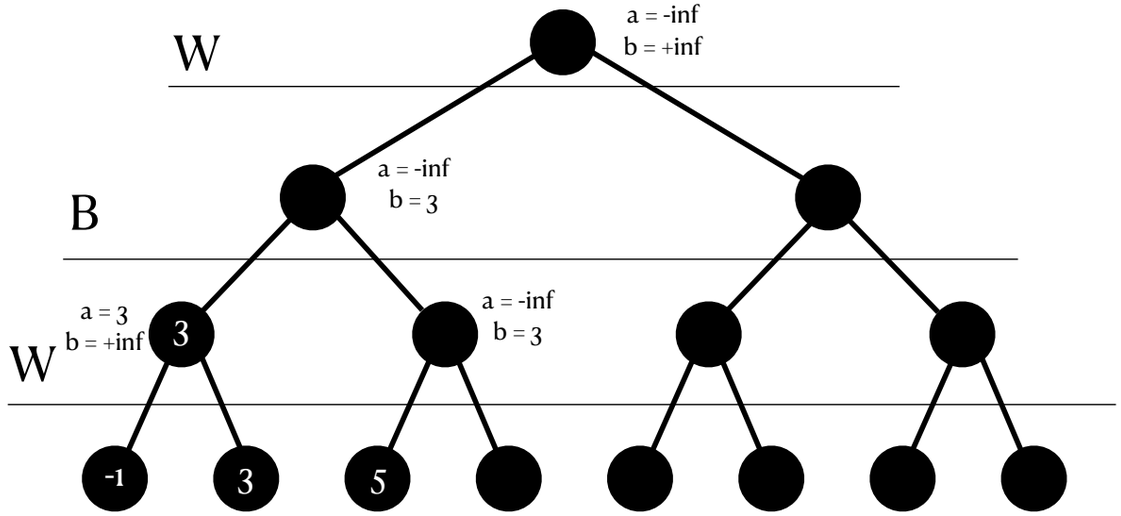
The basics of search: alpha-beta pruning

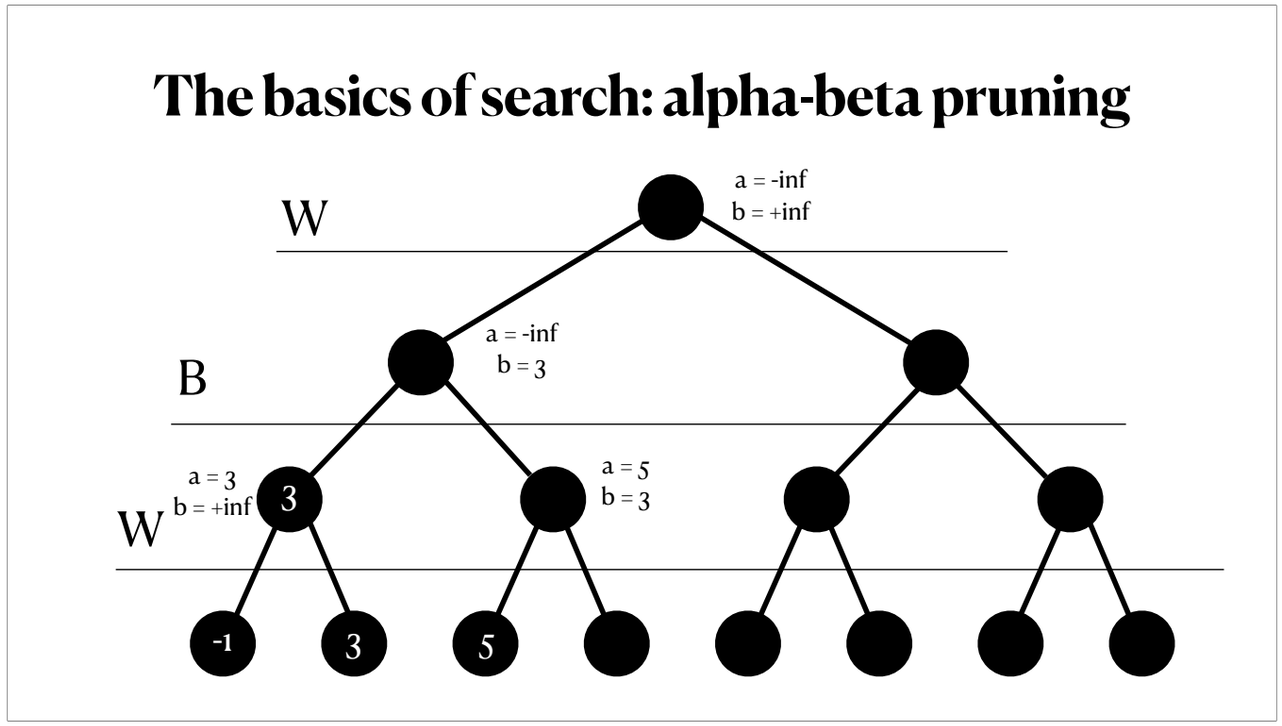


The basics of search: alpha-beta pruning

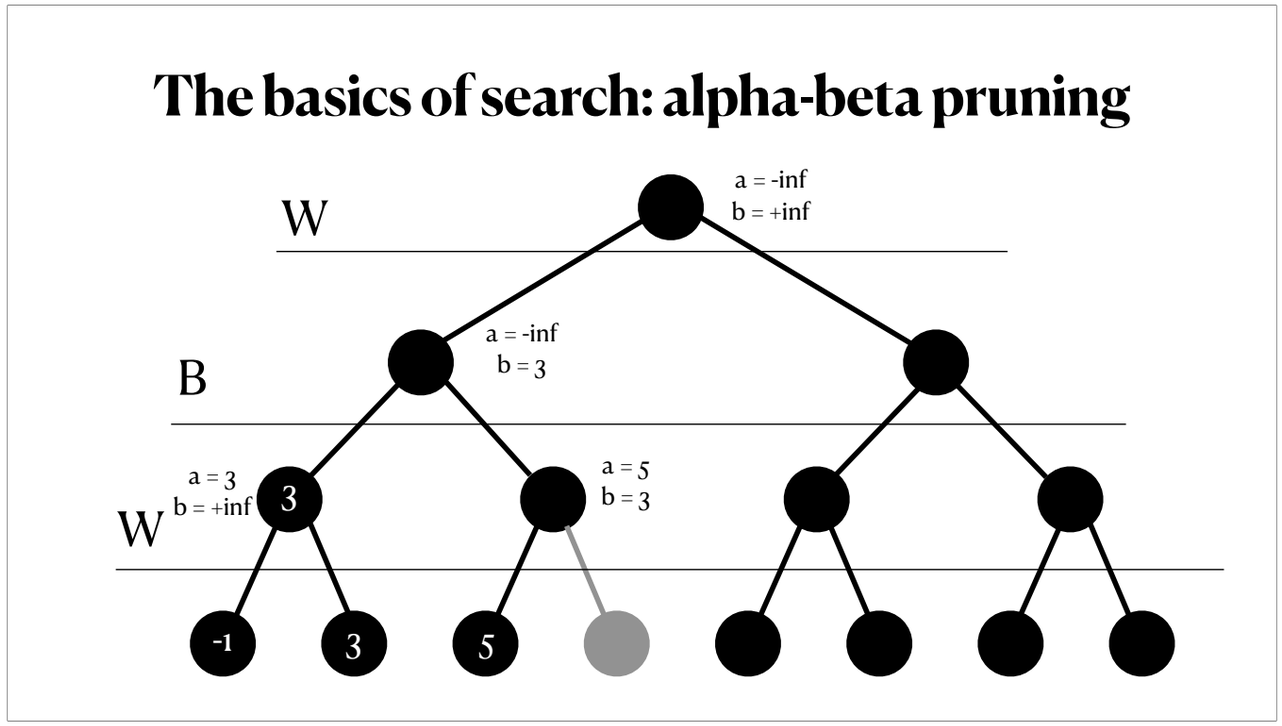


The basics of search: alpha-beta pruning



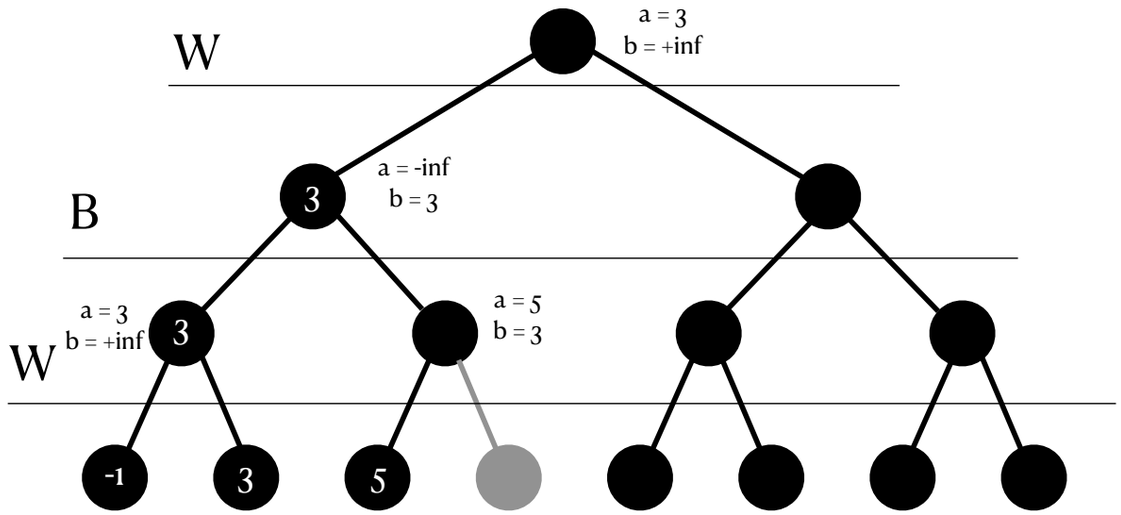


Now beta is less than alpha so we can prune the other child node.



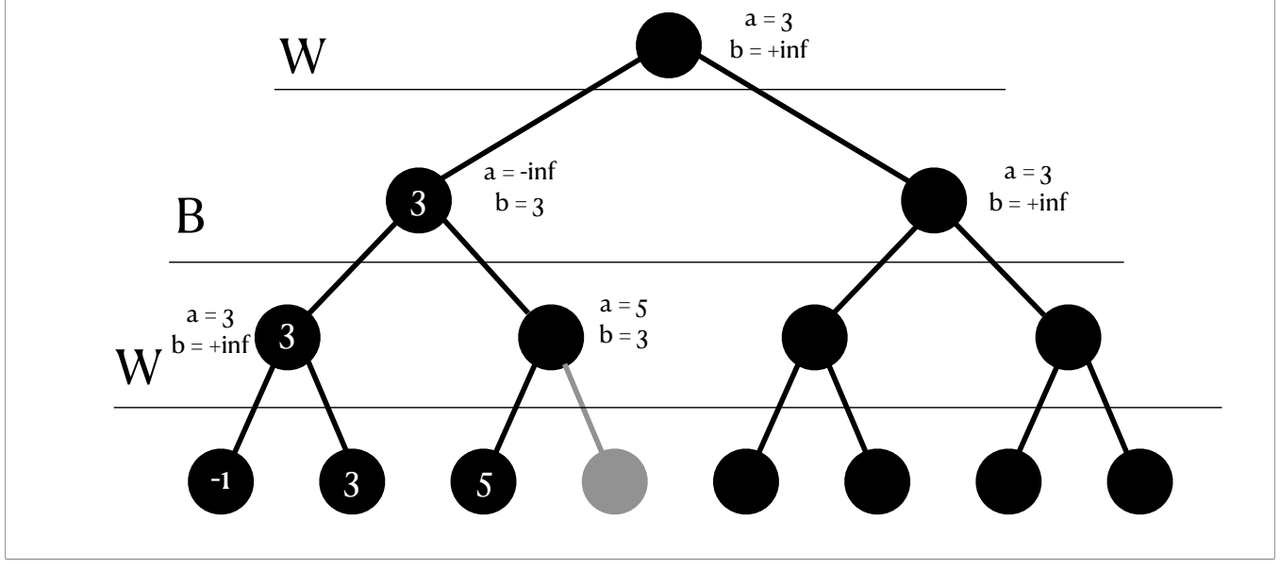
Now beta is less than alpha so we can prune the other child node.

The basics of search: alpha-beta pruning

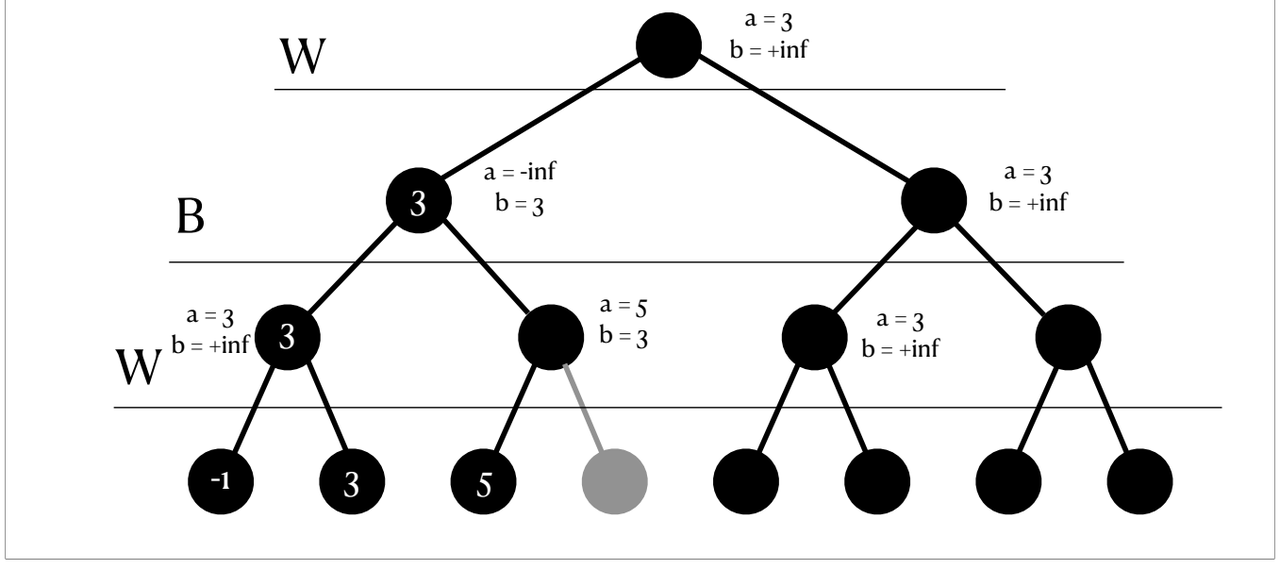


We propagate back up

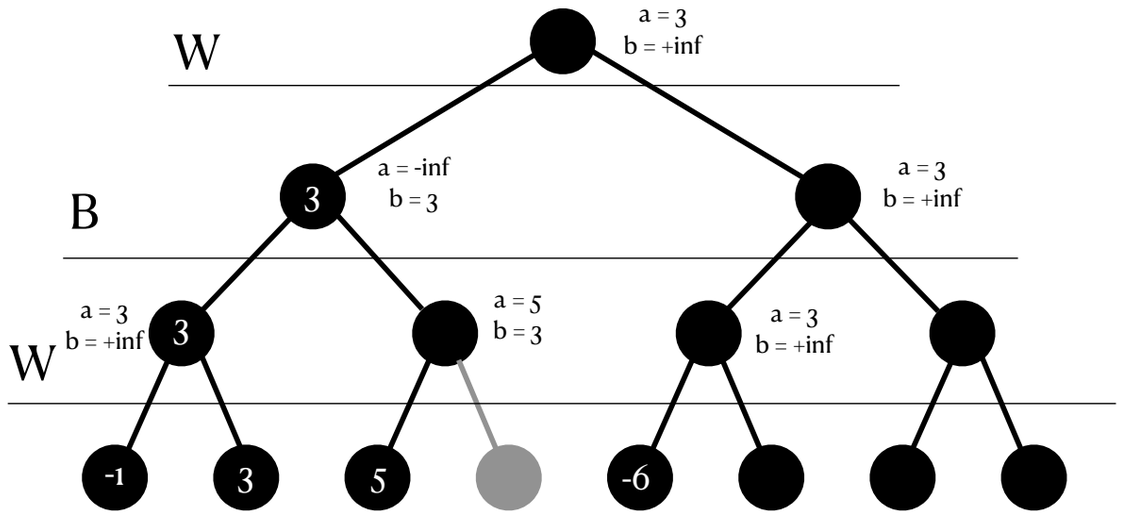
The basics of search: alpha-beta pruning



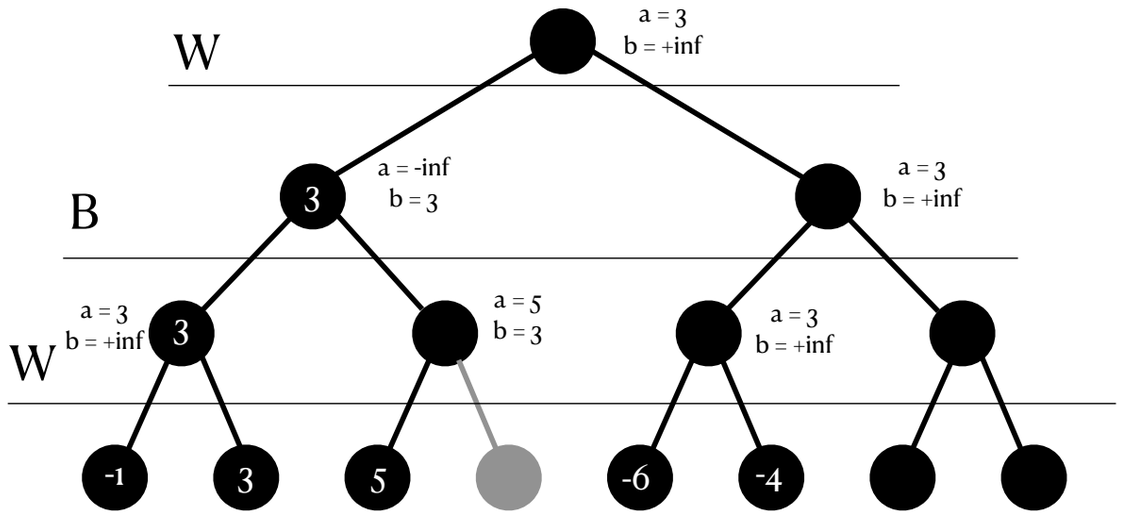
The basics of search: alpha-beta pruning



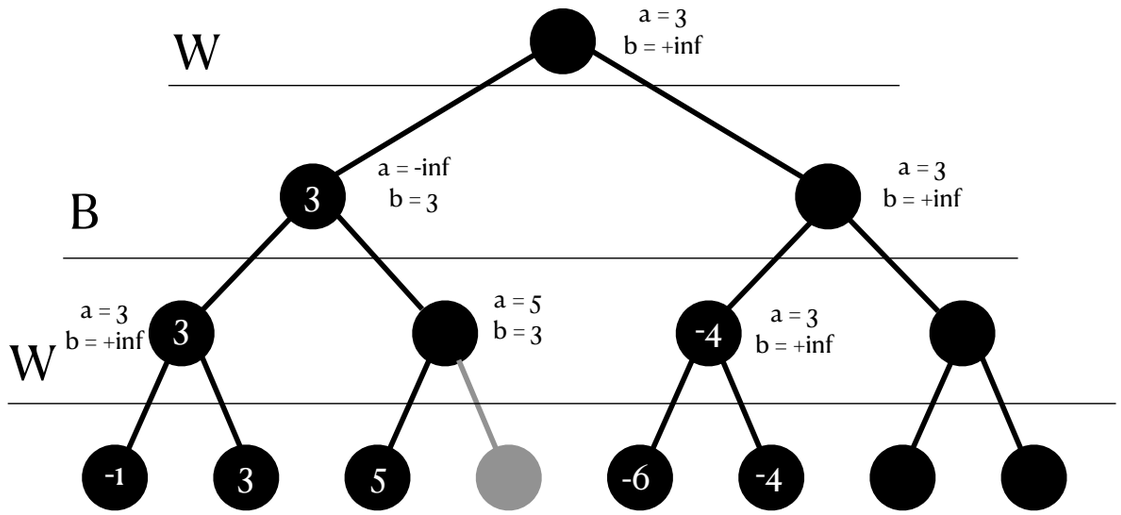
The basics of search: alpha-beta pruning

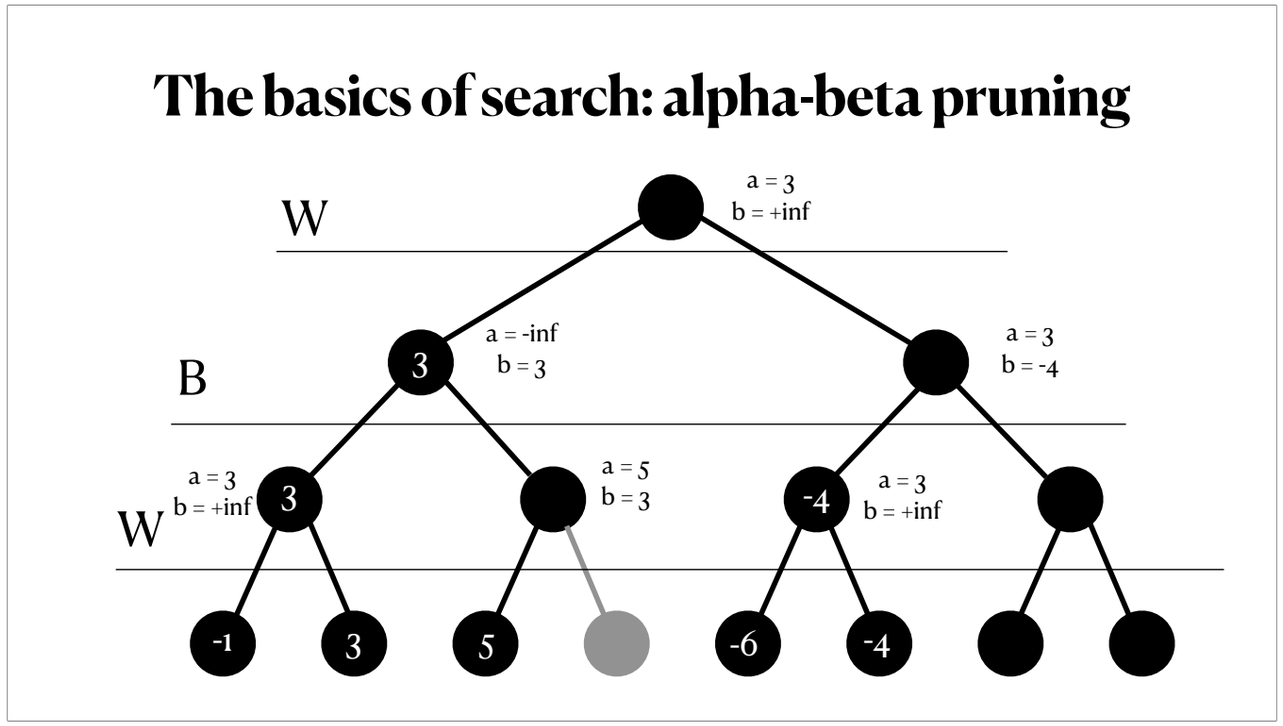


The basics of search: alpha-beta pruning

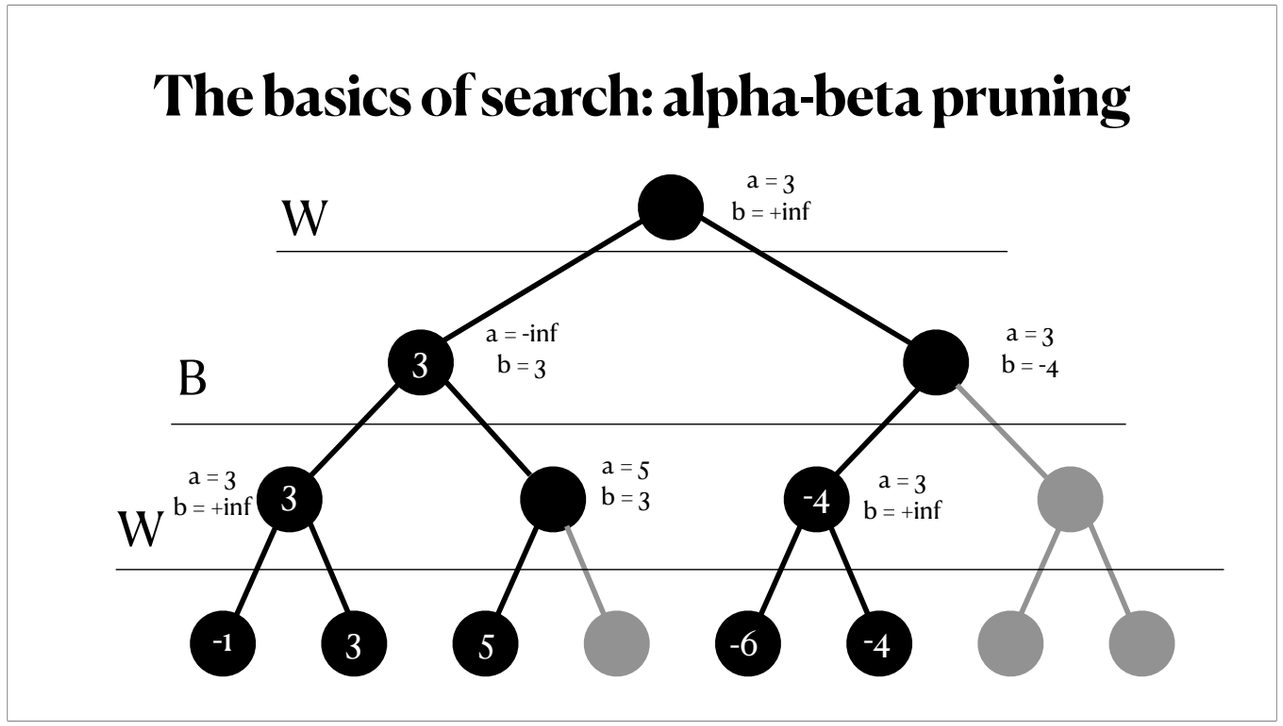


The basics of search: alpha-beta pruning

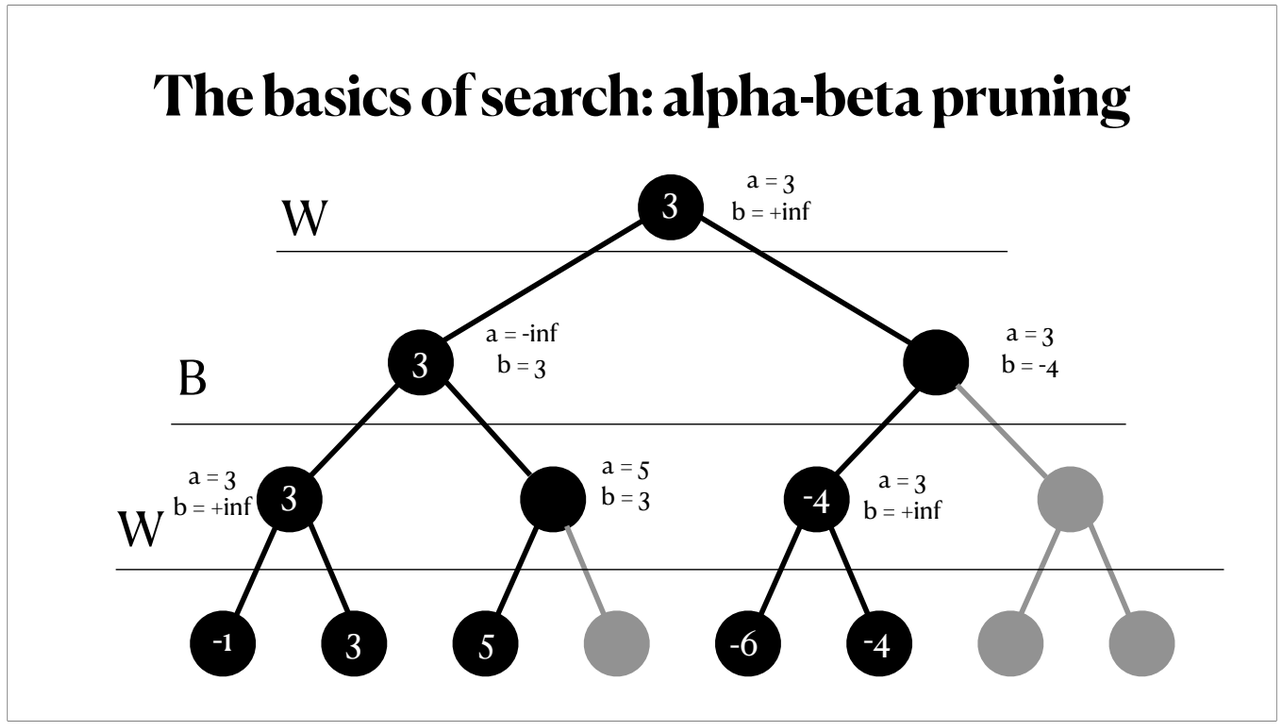




Now beta is less than alpha so we can prune the other child node.



Now beta is less than alpha so we can prune the other child node.



Now beta is less than alpha so we can prune the other child node.

The basics of search: alpha-beta pruning

```
function minimax(position, depth, alpha, beta, maximizingPlayer)
  if depth == 0 or game over in position
    return static evaluation of position

  if maximizingPlayer
    maxEval = -infinity
    for each child of position
      eval = minimax(child, depth - 1, alpha, beta false)
      maxEval = max(maxEval, eval)
      alpha = max(alpha, eval)
      if beta <= alpha
        break
    return maxEval

  else
    minEval = +infinity
    for each child of position
      eval = minimax(child, depth - 1, alpha, beta true)
      minEval = min(minEval, eval)
      beta = min(beta, eval)
      if beta <= alpha
        break
    return minEval
```

The basics of search: Quiescence

- Quiescent means “dormant” or “resting”.
- The idea is to extend searches until the position is quiescent, or quiet.
- Why is this important?
  - Imagine you calculate a sequence 5 turns deep, which ends with you capturing a pawn with your Queen. If you stop the evaluation there, you might use a heuristic to assign +1 to your position and deem that a favorable variation.
  - But what if, on the next move (which you didn’t calculate), the opponent has a knight move that captures your queen? Then in fact, this position is likely completely losing for you.
- Example: Vienna game, copycat variation.

The basics of evaluation

- In order to guide these search algorithms, some property of the tree's nodes must be known.
- In chess, that property is the "value" of the position represented by that node.
- How can we measure this value?
- There are 2 fundamental ways to approach position evaluation.
  - Model-free: Considering explicit features in a chess position; hand-crafted.
  - Model-based: Training a model to learn to do so.

Deep Blue II overview / fun facts

- "Deep Blue is a massively parallel system designed for carrying out chess game tree searches."
  - More specifically, to carry out the alpha-beta search algorithm in parallel.
- 30-processor IBM RS/6000 SP computer, 480 single-chip chess search engines, 16 chess chips per processor.
  - Each processor has 1 Gb RAM, 4 Gb disk space.
  - Chess playing program was written in C
  - In 97, Deep Blue was the 259th most powerful supercomputer (11.38 GFLOPS on LINPACK benchmark)
  - Each chess chip can search 2-2.5M positions/s. They each communicate with their host processor. Together, they can search 100-200 million positions/s.
  - Challenge: integrating all parallel search processes.

Deep Blue II overview / fun facts

- “It was clear at the time that the search should be highly non-uniform. A strong human player can search much deeper than any uniform searcher with reasonable time constraints.”
  - 3 minute search would reach a full-width depth of 12.2 on average.
  - They wanted a minimum depth (ex. 3 turns) on all possible moves, but otherwise only wanted depth on critical or otherwise principal variations: majoritively forcing moves.

Deep Blue II overview / fun facts

- Hardware search; conducted on chess chips
- Software search
- The evaluation function; conducted on the chess chips
- Integrating parallel searches; won't touch on this here
- Miscellaneous

The Deep Blue chess chip

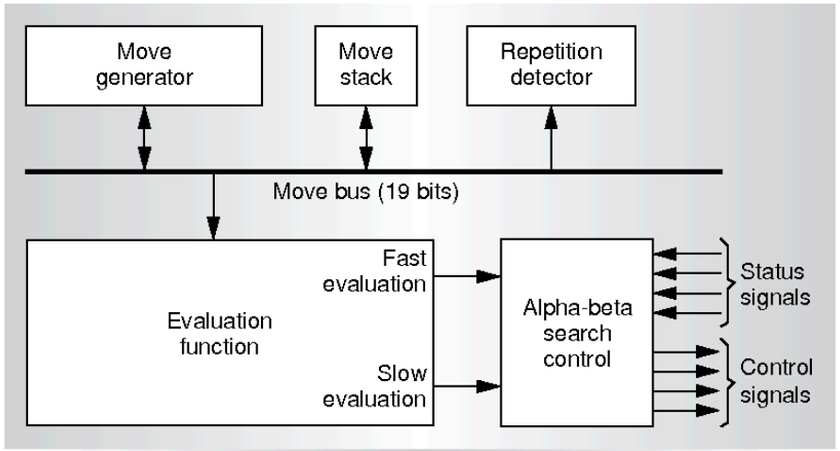


Figure 1. Block diagram of the chess chip.

The chess chip

- 1) Move generation
- 2) Evaluation function
- 3) Search control

The chess chip: move generation

- Done by a hardwired finite state machine
  - Generates all possible moves and selects one via an arbitration network for evaluation.
  - The order of moves generated matters for efficiency, and they'd like it to be as close to best-first as possible.
  - The order of generation is:
    - Captures; ordered from low-value pieces capturing high-value pieces to the opposite.
    - Non-capture moves; ordered by centrality (closeness to center of the board)

The chess chip: evaluation function

- Fast evaluation + slow evaluation
- Fast eval: can compute the score of a position in a single clock cycle. Contains all the easily computed major evaluation terms with high values.
  - The most significant part: piece values + square-based location adjustments.
  - Positional features that are quick to compute are also included; like “has a passed pawn”.
- Slow eval: scans the board one column at a time, computing scores for concepts like: square control, pins, X-rays (skewers), king safety, pawn structure, passed pawns, “ray control” (file control, but maybe also rank control), outposts, pawn majority, rook on 7th, blockades, color complex, trapped pieces, development, and so on.
- **All the slow and fast features have programmable weights, allowing their relative importance to be easily adjusted.**

The chess chip: search control

- Implements null-window alpha-beta search.
  - Null windows were first described by Judea Pearl in 1980! Link here.
  - Hardware has no transposition table. Software search elements do have access to one.
    - Transposition table: a cache of previously seen positions, and their associated evaluations, from a game tree.
    - The search requires a move stack to keep track of previous moves/positions. Deep Blue II keeps track of the last 32 moves (# of pieces displaced from the current board position). When the # displaced = 0, the position has been seen before.
- Extendability: the chess chips had support for external FPGAs, which could have been used to provide transposition tables, more complicated search control and additional terms for the evaluation function but this was never implemented due to time constraints.

Hardware search

- Carried out on each chess chip: a fixed-depth null-window search, including a quiescence search component at the end of any full-width searching.
- The search is fast but simple. The idea is to balance this with the efficiency and complexity of the software search.
  - Thus the hardware is limited to shallow searches; 4-5 plies with quiescence in middle-games and somewhat deeper searches in endgames.
- Hardware evaluation
  - Deep Blue evaluation function is implemented in hardware, so its execution time is a fixed constant. Software evaluation functions have to deal with slow-downs when incorporating new features. But the irony is that you can use so many features (8000) with a hardware eval function that tuning their relative value is difficult.

Software search

- Hybrid software/hardware search
  - Software search in compile C code on a general purpose CPU + hardware search on the chess chip. The software search is highly flexible and can be changed.
  - Contrast: The hardware search is parameterized but the form of the search is fixed – new search behaviors can't be introduced and the existing parameters require careful tuning.
  - It is difficult to decide the strategy of when to switch from the hardware to the software search.

Software search

The dual credit with delayed extensions algorithm

```
int DC(
  position p,
  int alpha, int beta,
  int depthToGo,
  float myCredit, float hisCredit)
{
  int numOfSuccessors;
  int bestScore;
  int i;
  int sc;
  float newCredit;
  int extensionAmount;

  if (hisCredit >= CREDIT_LIMIT) {
    extensionAmount = ceiling(hisCredit - CREDIT_LIMIT);
    hisCredit = hisCredit - extensionAmount;
    myCredit = max(myCredit - extensionAmount, 0);
    depthToGo = depthToGo + extensionAmount;
  }

  if (depthToGo == 0) { return Evaluate(p); }

  bestScore = alpha;
  numOfSuccessors = GenerateSuccessors(p);
  for (i = 1; i <= numOfSuccessors; i++) {
    sc = -DC(p.succ[i], -beta, -alpha, depthToGo - 1,
             hisCredit, myCredit);
    if (sc > bestScore) {
      newCredit = GenerateCredit();
      if (newCredit > 0)
        sc = -DC(p.succ[i], -beta, -alpha, depthToGo - 1,
                 hisCredit, myCredit + newCredit);
      if (sc > bestScore)
        bestScore = sc;
    }
    if (bestScore >= beta) { return bestScore; }
  }
  return bestScore;
}
```

Basically alpha-beta pruning with the addition of a “credit assignment mechanism”. The search is initially depth-fixed, but is extended if a move on either side generates sufficient credit to warrant the extension.

Software search: credit generation

4.1. Credit generation mechanisms

There is a large set of mechanisms to identify nodes that should receive credit.

1. *Singular, binary, trinary, etc.* [12]
   A singular move is one that is significantly better than all the alternatives [2]. One can generalize this to the case where there are two, three or more good moves. Of course the more reasonable moves that are available, the less credit that should be given. It is clear that a large part of what a strong human chess player would define as forcing is covered by singularity. It is in just these kinds of positions that Grandmasters are able to calculate very deeply.
2. *Absolute singular.*
   When there is only one legal move a large credit can be given with very little risk. The reason is that, if the absolute singular move ends up failing low, there are no alternatives to examine so the cost is contained. It is reasonable in many cases to give a full two ply extension. This type of move is usually a check evasion move.
3. *Threat, mate threat.*
   It is relatively simple using a null move search to detect if there is a threat in the current position [1]. A null move search is a search conducted after one side passes. The intuition here is that if one side passes, then loses quickly, that side is deemed to have a pending threat against it which the other side is close to exploiting. Positions where a threat exists tend to be constrained in the number of reasonable alternatives. If a large threat exists, such as a threat to checkmate, a higher level of credit can be given. The Deep Blue implementation required that recent ancestors of a given position have forcing characteristics before a threat credit is given.
4. *Influence.*
   This mechanism gives credit for moves which are enabled by previous moves. For example, credit may be given for an apparently good white response to a black move which was not available the previous time black moved. The idea here is that we assume black is developing a combination even if we don't quite see what the combination is.
5. *Domain dependent.*
   Traditional search extension schemes, such as check evasion and passed pawn pushes, can also be incorporated into this method. For example, a passed pawn push can be considered a forcing move, and the response, if it does not fail low, can generate credit.

Evaluation function

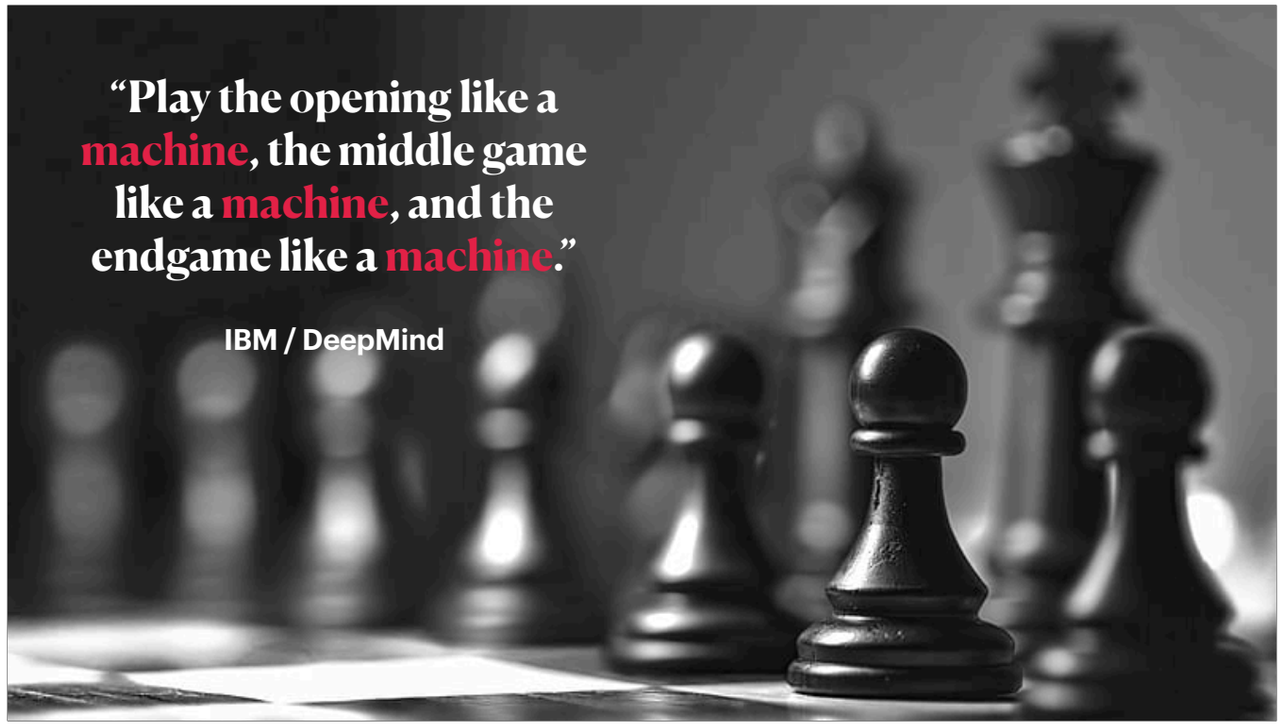
- Overview:
  - Eval = linear combination of ~8000 features
    - Some weights in the linear combination are static (assigned at the start of a node search), others are dynamic (scaled throughout a search; such as king safety's weight being scaled according to the amount of material on the board).
  - Initially written in generalized form, with many untuned parameters (ex. Importance of King safety, central space advantage, etc.).
  - Deep Blue estimated these by analyzing master games.

Miscellaneous

- Opening book: made by hand by 3 GMs, with about 4000 positions
- Extended book: 700,000 master games
  - Deep Blue assigned bonuses to moves that had been played in the master games.
  - Ex. In position p, say that d4 has a 10 point bonus. Then the alpha-beta search is offset by 10, so that d4 will be deemed the best move unless another move found exceeds d4's score by >10 points.
  - This "10 point score" is estimated based on: absolute # times it was played in the master games, the relative # times it has been played, strength of the players that made the move, recentness of the move (contemporary theory vs. old theory), game results of that move, annotation of the move (ex. !), and whether the move was a game move or a commentary move.
  - These factors are combined in a non-linear way to produce a scalar value as output = the bonus for that master-made move.
- Endgame databases: select 6-piece positions, and all positions with <=5 pieces.
  - Stored with one bit per position (1 if losing, 0 if not-losing).
  - Not important against Kasparov – only game 4 approached an endgame that required access to the databases.
- Time control
  - Kasparov match: 40 moves to be played in 2 hours.
  - Before each search, 2 time targets are set:
    - Normal time target (time till the next time control / # moves remaining to reach it) with a considerable time buffer
    - Panic time target: 1/3 of the normal time target.

Conclusion

- Success of Deep Blue wasn't due to just one factor.
- 3 were critical: Large search capability, Non-uniform search, Complex evaluation function
- Other factors: endgame databases, the extended book, and evaluation function tuning.
- Areas to improve: parallel search efficiency, hardware search and evaluation, addition of pruning mechanisms to search, and evaluation function tuning was crude.


“Play the opening like a machine, the middle game like a machine, and the endgame like a machine.”
IBM / DeepMind


Thank you!

How does Deep Blue's speed compare to today's top engines?

- #1 Stockfish:
  - ~3550 ELO, 5400 years of CPU time, 3.1B chess games
  - Capable of ~70M positions/s search
  - 512 CPU threads, 32 TB transposition table, alpha-beta search
    - Superior search depth, thanks to more aggressive pruning and late move reductions.
  - NN for its evaluation function (called NNUE). Cut search speeds in half, but still caused +100 ELO
    - Description of NNUE here: https://github.com/glinscott/nnue-pytorch/blob/master/docs/nnue.md
    - 5-layer feedforward NN: input layer shape 81,920 (binary digits), 3 hidden layers with shapes 512, 8, 8, and output is a single float, trained to be the centipawn evaluation of the position.
    - Capitalizes on minimal input changes per move, storing hidden state parameters and updating them based solely on what input features changed.
  - HalfKP feature set: just the right size, and requires very few updates per move on average.
    - Tuple of (king_square, piece_square, piece_type, piece_color): 64 x 64 x 5 x 2 = 40,960. So a binary array of length 40,960 where each index has value 1 or 0 depending on whether the corresponding tuple is manifested on the board.
      - Piece_type does not include the king.
      - king_square refers to one side's king.
- #2 Leela Chess Zero (adaptation of AlphaZero):
  - DRL-based chess engine, open-source, and free.
  - Made by the developer of Stockfish. Purpose was to verify methods from the AlphaZero paper applied to the game of chess.
  - Starts with no intrinsic chess-specific knowledge other than the basic rules of the game.