How to find the best move!

Search Strategies for two player games.

Seminar zu Vertiefungen und Anwendungen
Von Tilman Renneke

Outline

- Motivation
- Games
- Minimax algorithm
- Search vs lookup
- Monte Carlo tree search

Motivation

- Games are decision making models
- Games can require difficult decisions
- God human knowledge in many games
- Similarities to more “useful” problems

Outline

- Motivation
- Games
  - Types
  - Examples
  - Game tree
  - Algorithmic description
- Minimax algorithm
- Search vs lookup
- Monte Carlo tree search

Games

- Two Players
- Turn based
- Zero-sum
- Perfect/Imperfect Information

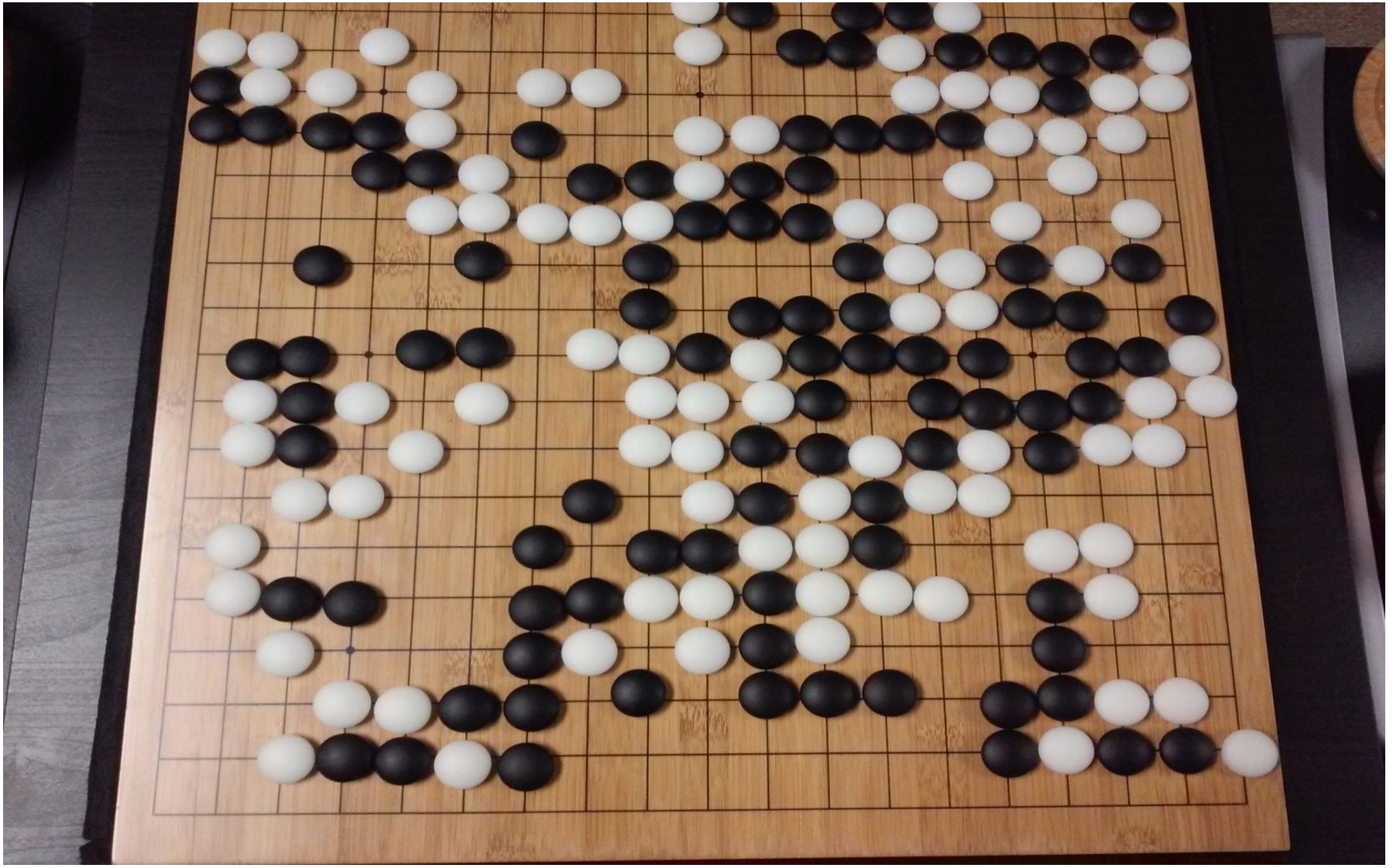
Games

Games – Examples

Chess

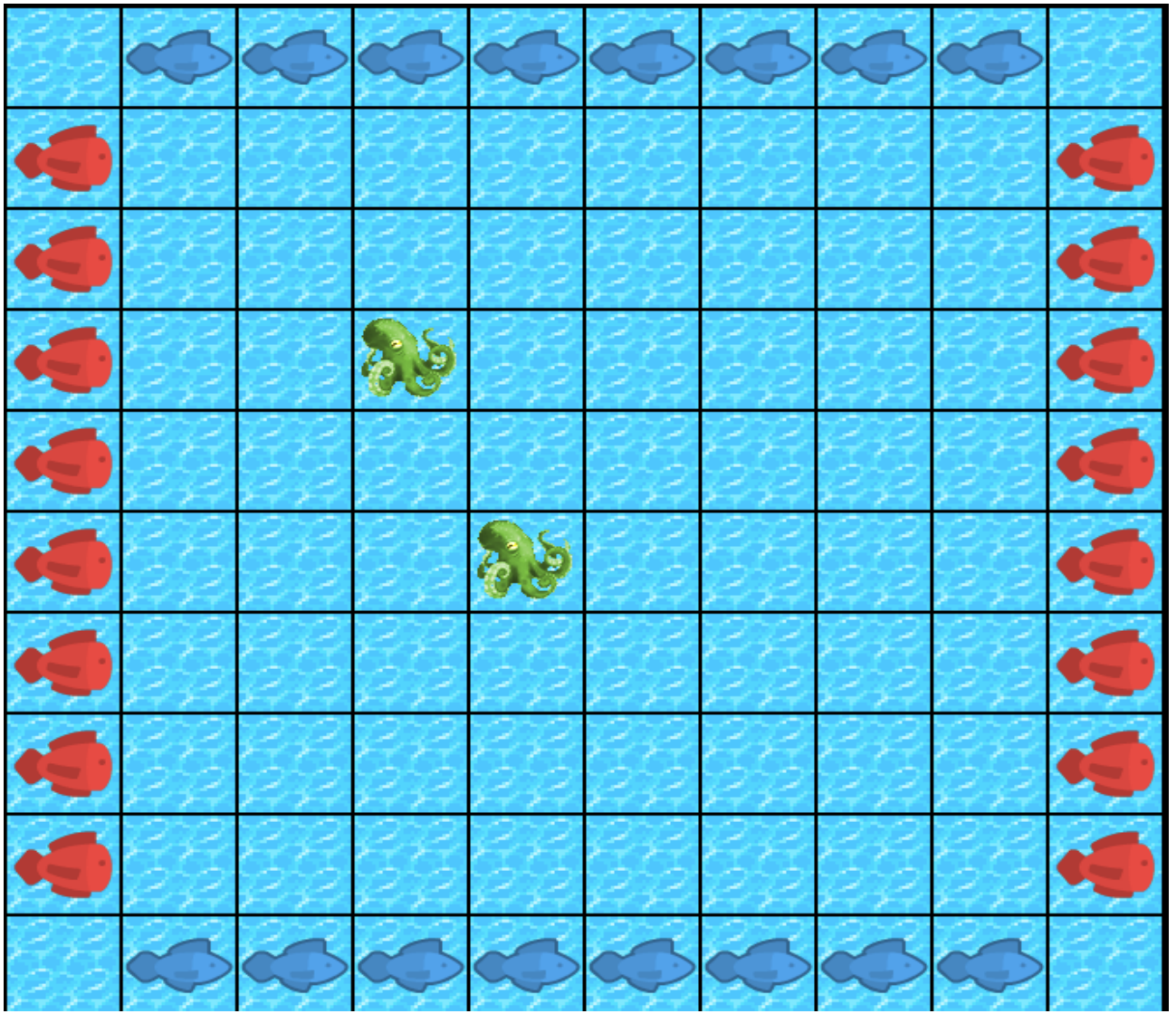
Games – Examples

GO



Games – Examples

Piranhas



Games - Examples

- Checkers
- Mill
- Connect Four
- Tic-tac-toe

Games – game tree

- Nodes are game state's
- Edges are legal moves
- Comlexity: b^m
- b = number of average turns
- m = average game length
- Tic-tac-toe: 4^9
- Chess: 35^{70}
- GO: 250^{150}

Games

- GameState S_0 initial game state (Schach Bild)
- Player player(GameState s)
- Action[] actions(GameState s)
- GameState result(GameState s, Action a)
- Boolean terminalTest(GameState s)
- Float utility(GameState s, Player p)

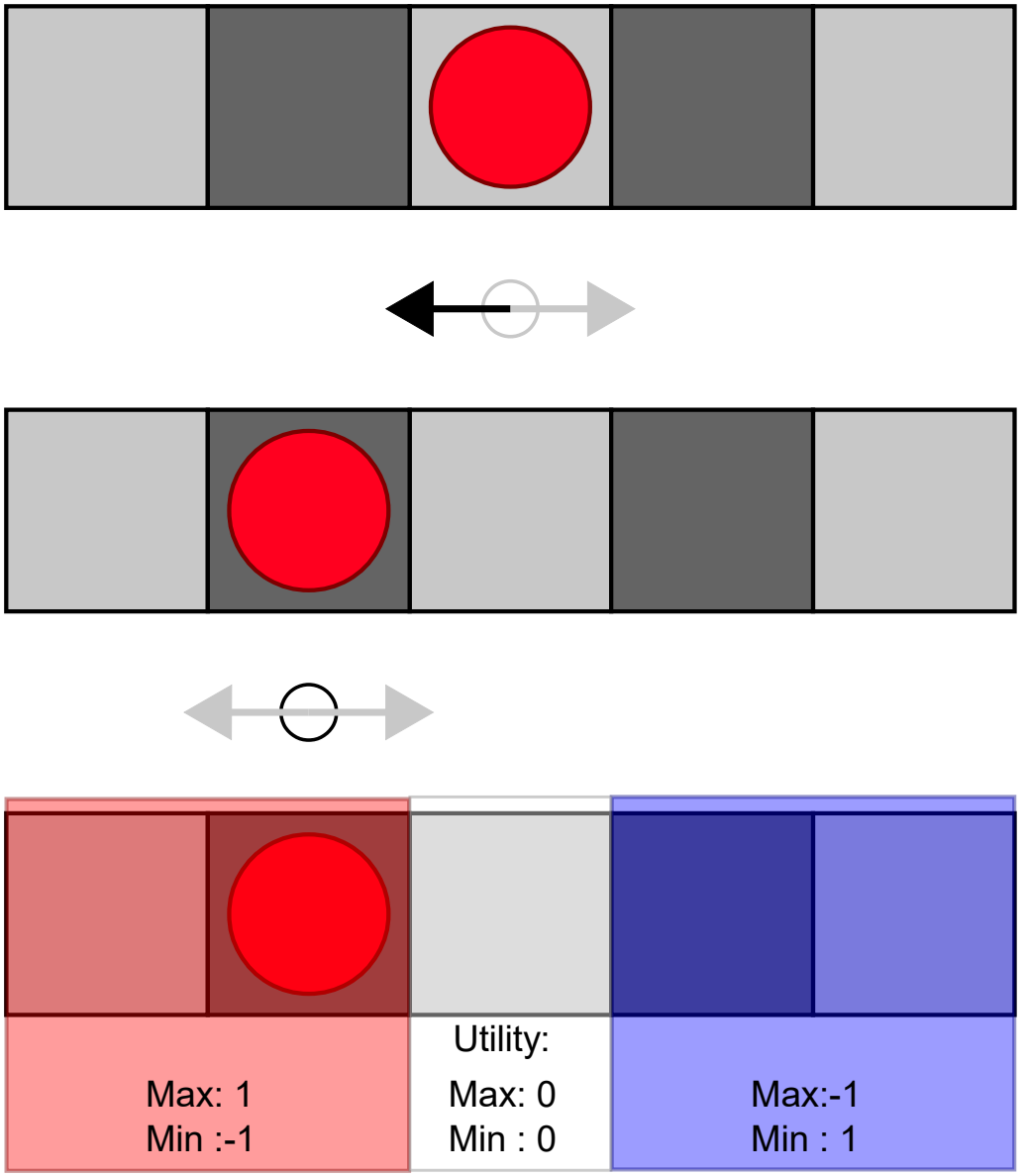
Outline

- Motivation
- Games
- Minimax algorithm
  - Alpha beta pruning
    - Move ordering
  - Imperfect real time decisions
- Search vs lookup
- Monte Carlo Tree search

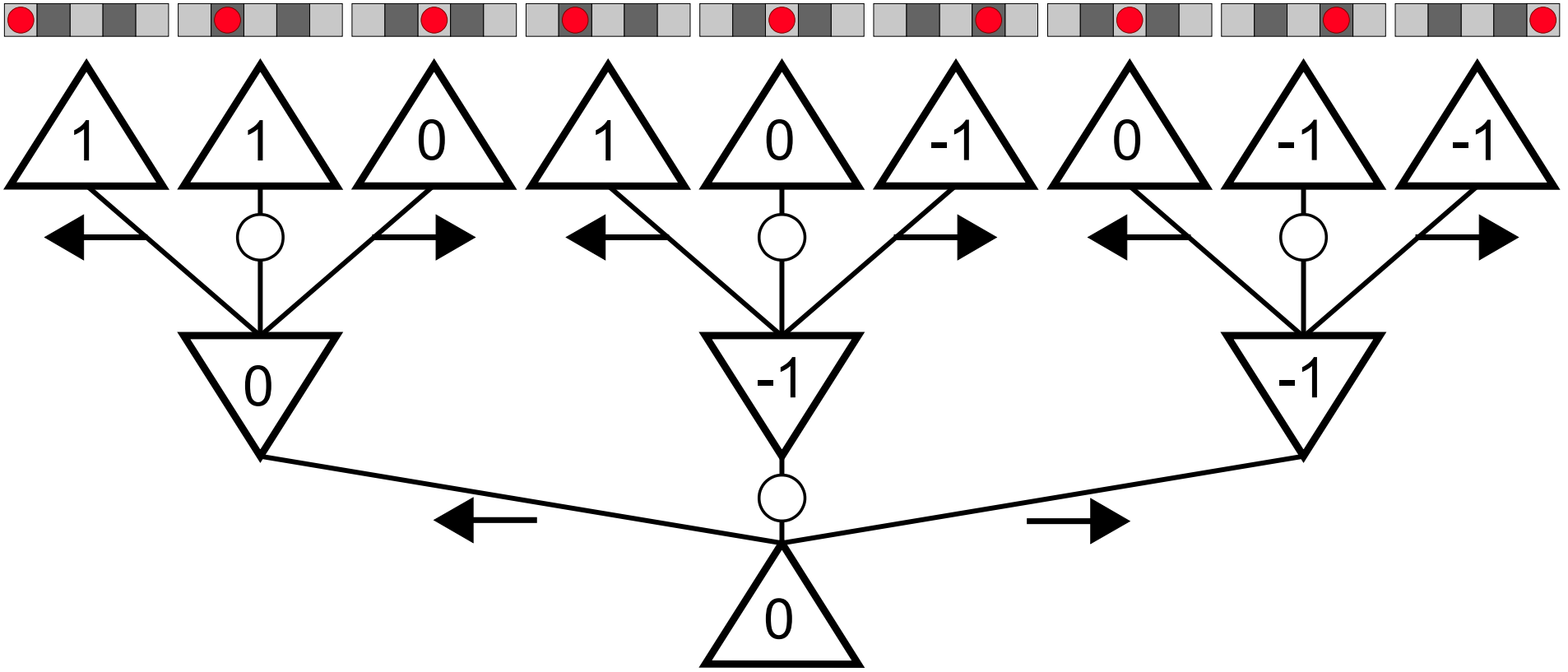
Minimax Algorithm

```
float maxValue (GameState s) {
    if (terminalTest(s)) {
        return utility(s);
    }
    float v = -float.MaxValue;
    foreach (Action a in actions(s)) {
        v = Max(v, minValue(result(s, a)));
    }
    return v;
}
float minValue (GameState s) {
    if (terminalTest(s)) {
        return utility(s);
    }
    float v = float.MaxValue;
    foreach (Action a in actions(s)) {
        v = Min(v, maxValue(result(s, a)));
    }
    return v;
}
```

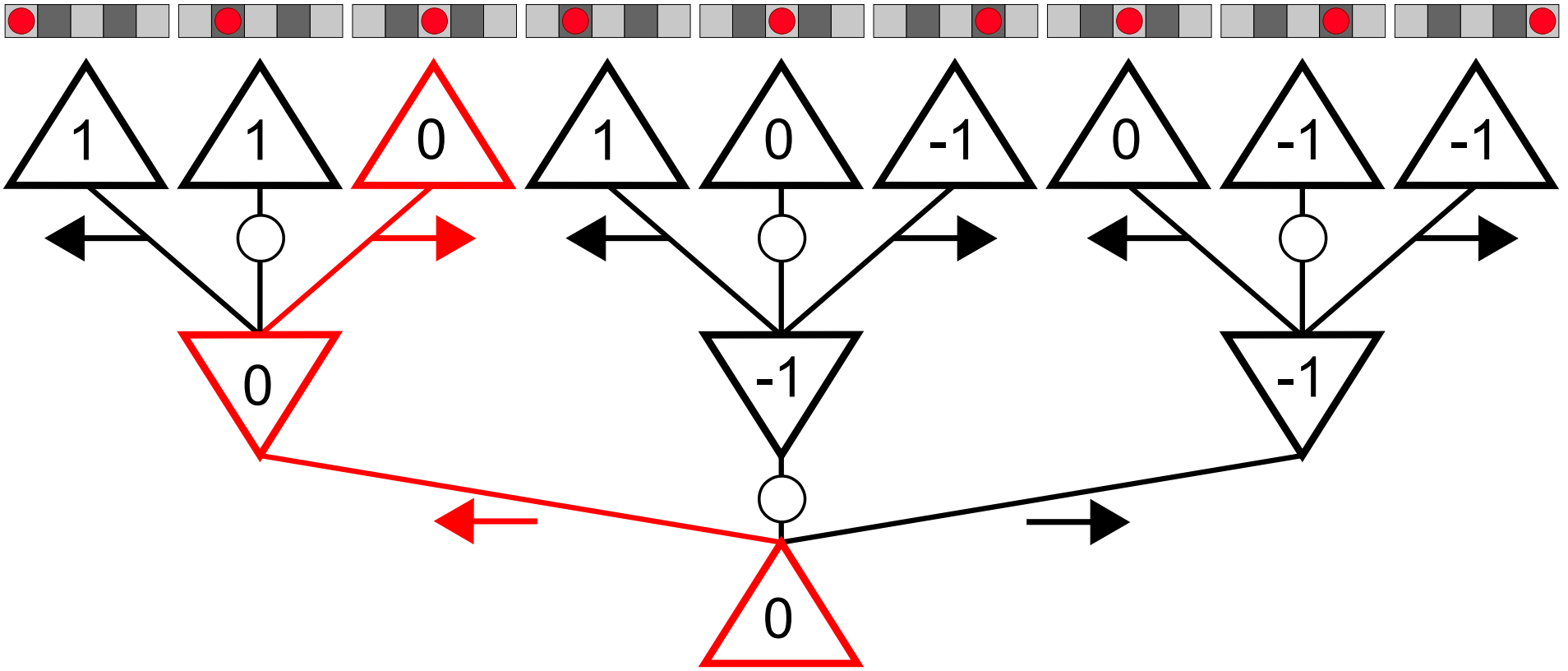
Minimax Algorithm



Minimax Algorithm



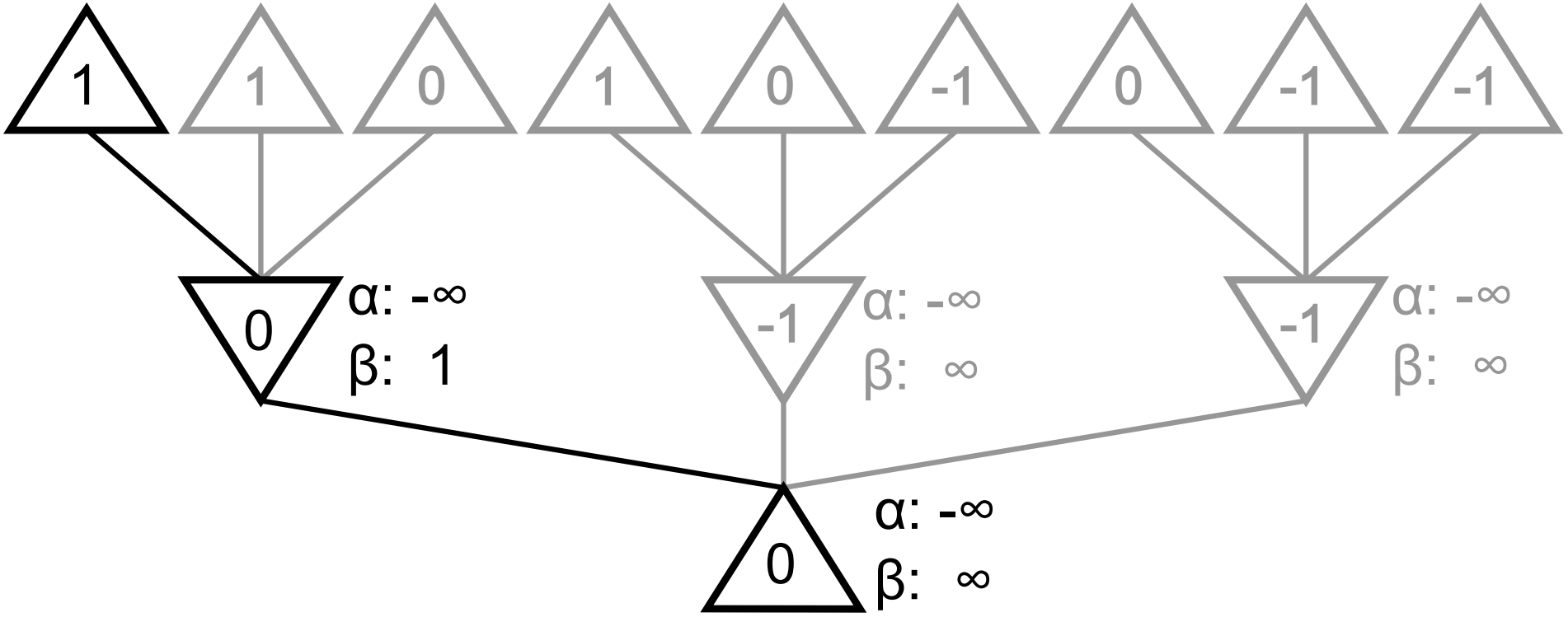
Minimax Algorithm



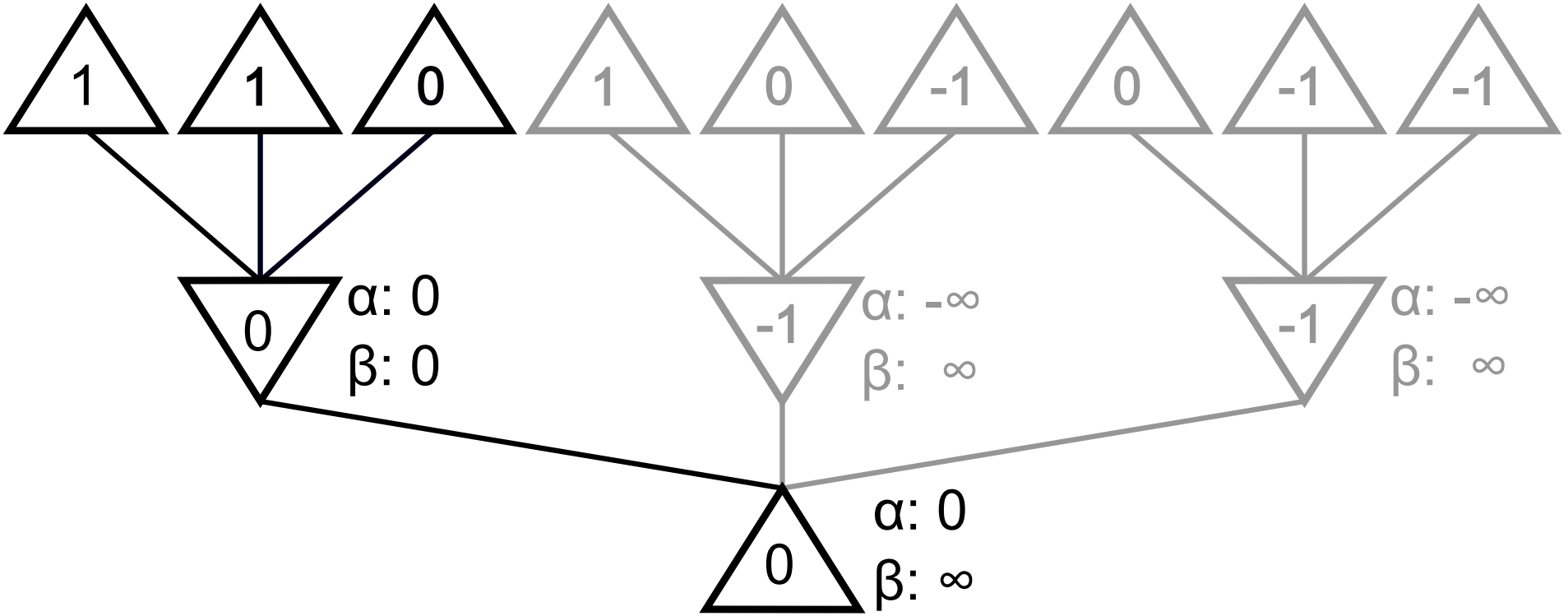
Minimax Algorithm
Alpha Beta Pruning

```
float maxValue (GameState s, float alpha, float beta) {
    if (terminalTest(s))
        return utility(s);
    float v = -float.MaxValue;
    foreach(Action a in actions(s)){
        v = Max(v, minValue(result(s, a), alpha, beta));
        if (v >= beta)
            return v + 1;
        alpha = Max(alpha, v);
    }
    return v;
}
float minValue (GameState s, float alpha, float beta) {
    if (terminalTest(s))
        return utility(s);
    float v = float.MaxValue;
    foreach (Action a in actions(s)) {
        v = Min(v, maxValue(result(s, a), alpha, beta));
        if (v <= alpha)
            return v - 1;
        beta = Min(beta, v);
    }
    return v;
}
```

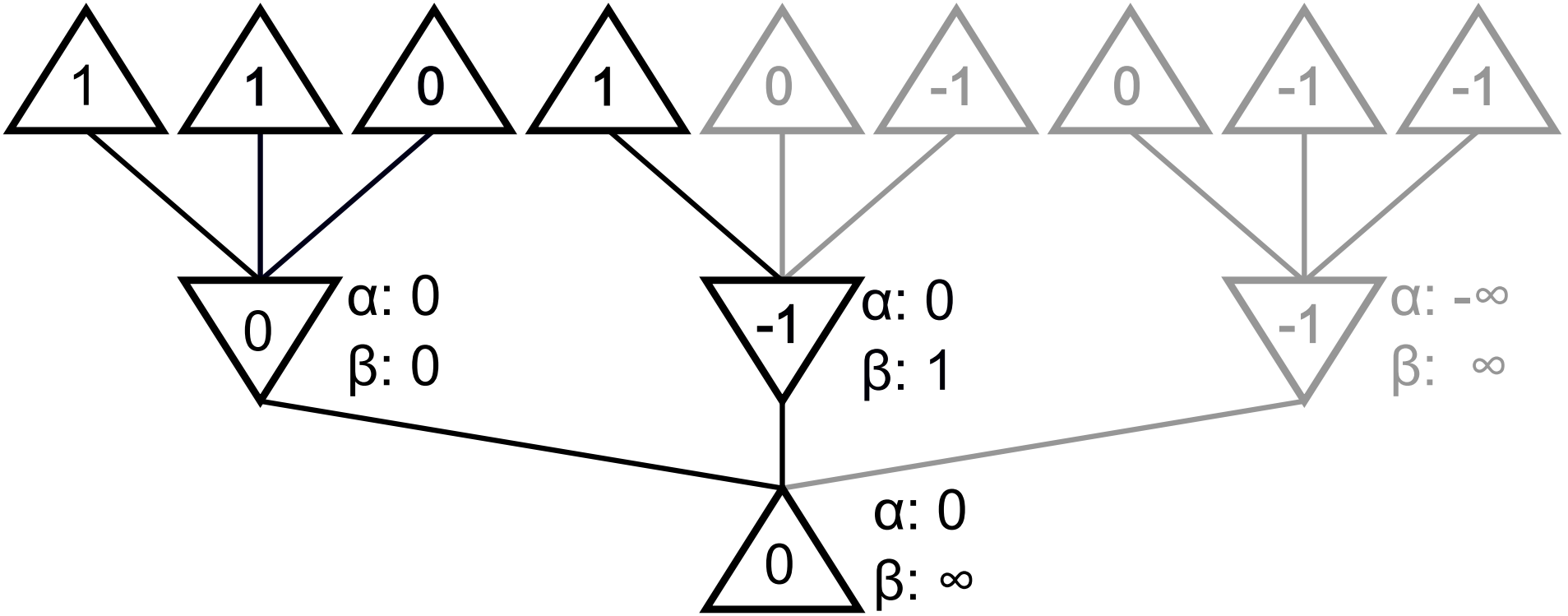
Minimax Algorithm
Alpha Beta Pruning



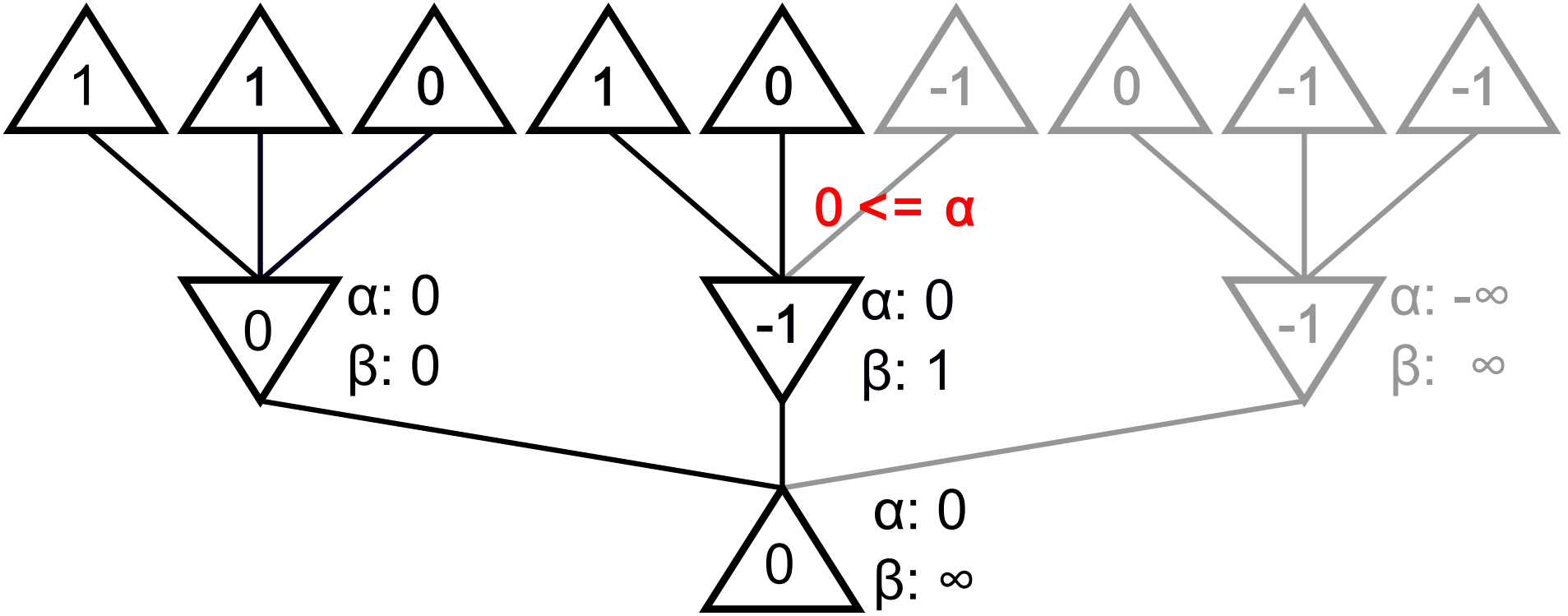
Minimax Algorithm
Alpha Beta Pruning



Minimax Algorithm
Alpha Beta Pruning

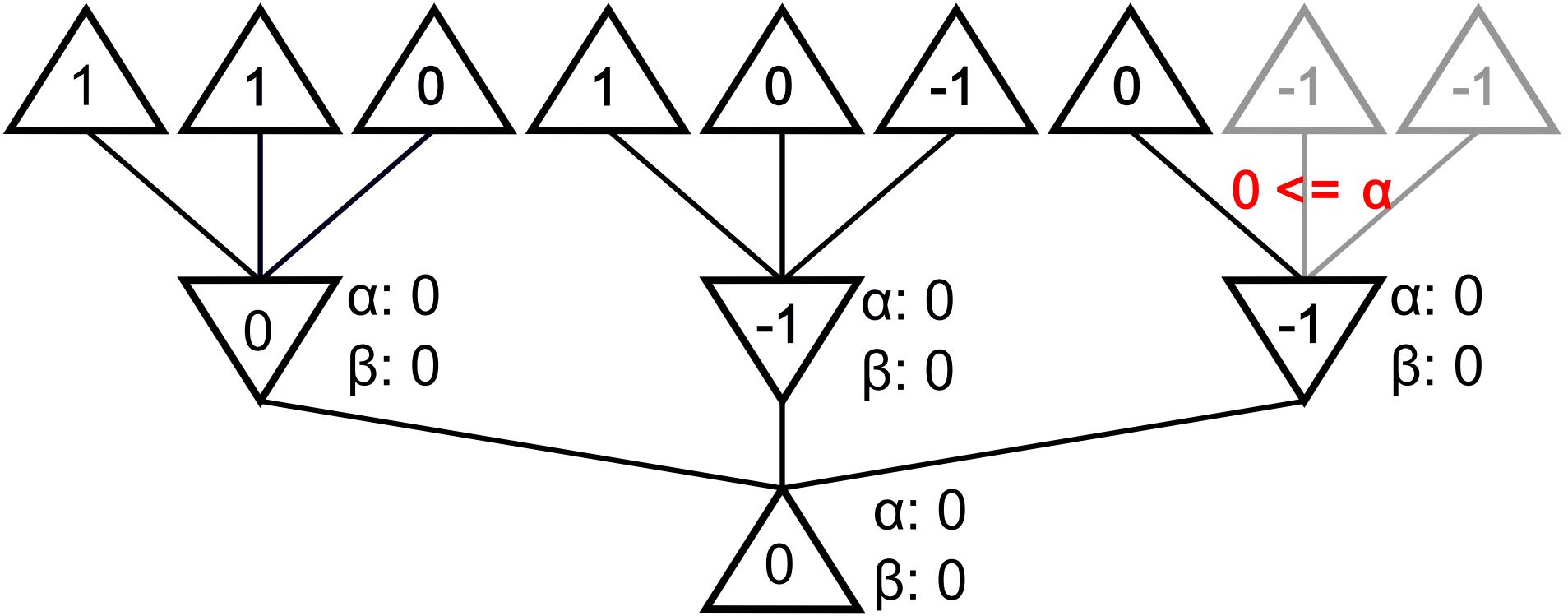


Minimax Algorithm
Alpha Beta Pruning



Minimax Algorithm

Alpha Beta Pruning



Minimax Algorithm
Alpha Beta Pruning

- Optimal complexity: $O(b^{1/2\ m})$
- Depends on the turn evaluation order
- Random Order Complexity: $O(b^{3/4\ m})$
- Sort with a specific ordering function
- Sort using an evaluation function

Outline

- Motivation
- Games
- Minimax algorithm
  - Alpha beta pruning
  - Imperfect real time decisions
    - Evaluation-function
    - Cutoff-function
- Search vs lookup
- Monte Carlo Tree search

Imperfect real time decisions

- Minimax can only solve “simple” Games
- Tree's of complex Games are to Big.
- Minimax → H-Minimax
- Termination-Test → Cutoff-Test
- Utility → Evaluation

Evaluation Functions

- Heuristic to estimate the utility of the game for a given state.
- Example chess: Adding up the values of each piece on the field.
- Has to order Terminal notes the same as Utility
- Has to be fast
- Should be strongly correlated with a chance of winning.

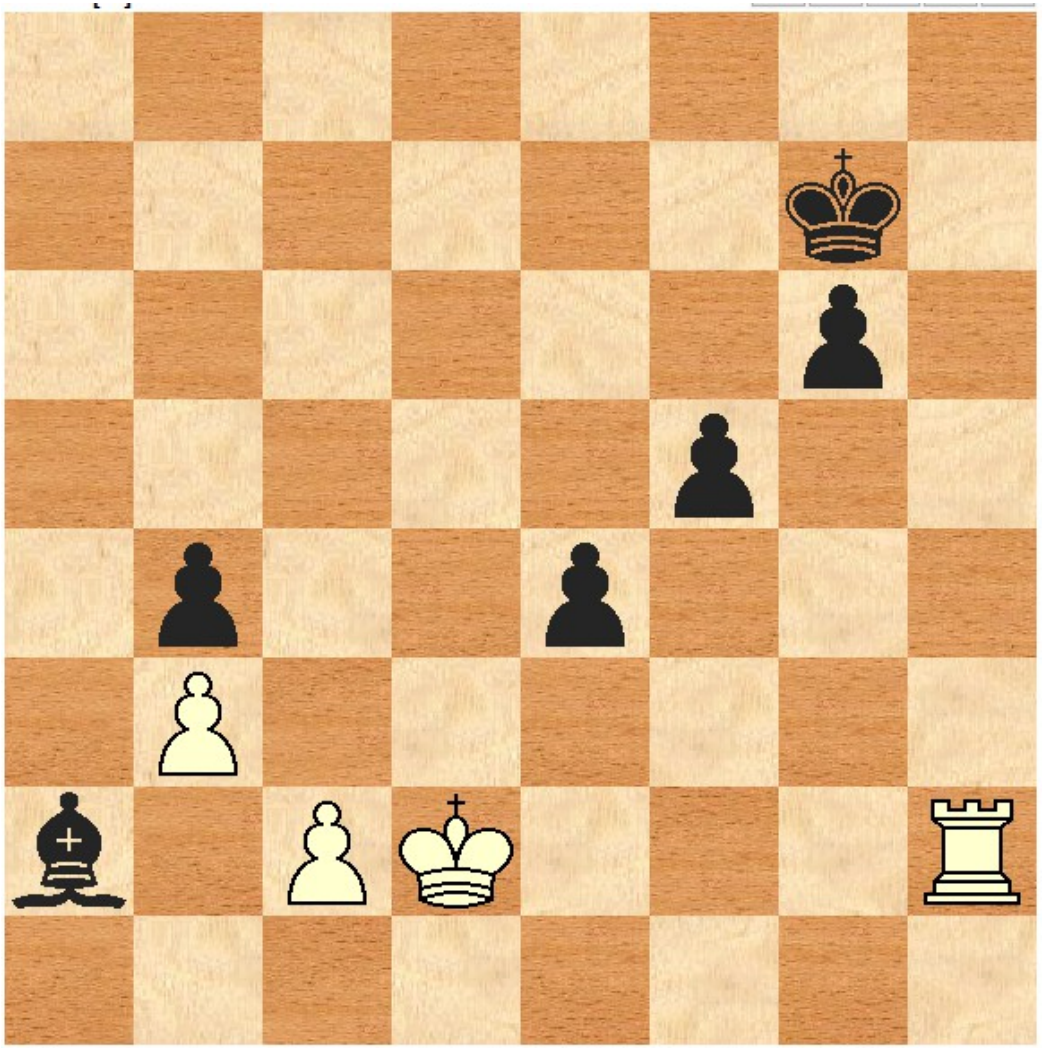
Evaluation function

- Idea 1: Equivalence classes
  - Calculate features of a state
  - For each state class compute the expected outcome
- Idea 2: Linear weighted functions
  - Multiply each feature value with a weight
  - Compute the Sum of all weighted features
- Idea 3: Nonlinear functions
  - Some feature can influence the weight of others

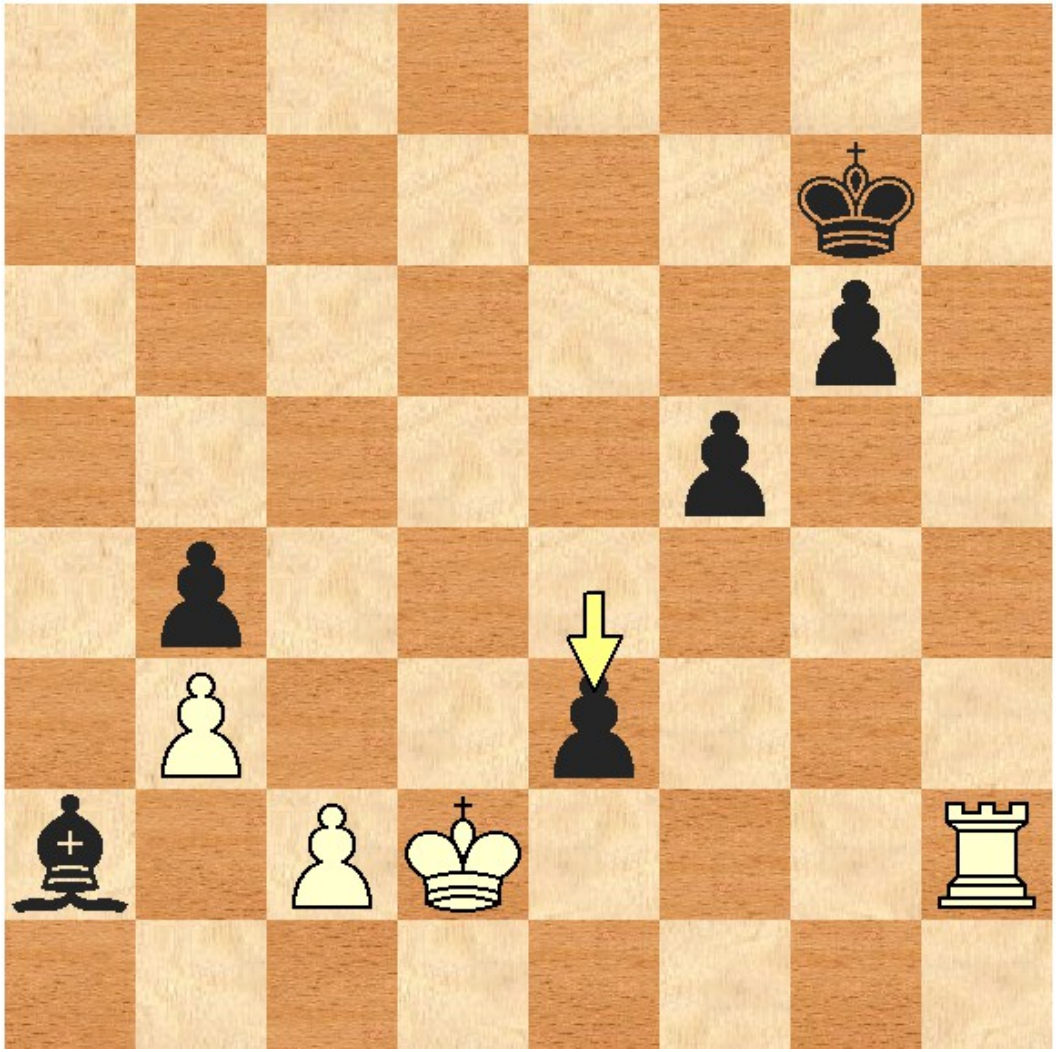
Cutoff-function

- Fixed depth cutoff
- Quiescence
  - A value for how much the heuristic can change
- Quiescence search
  - If a position isn't quiescent return false
- Horizon Effect
  - Pushing unavoidable losses over the horizon with delaying tactics.

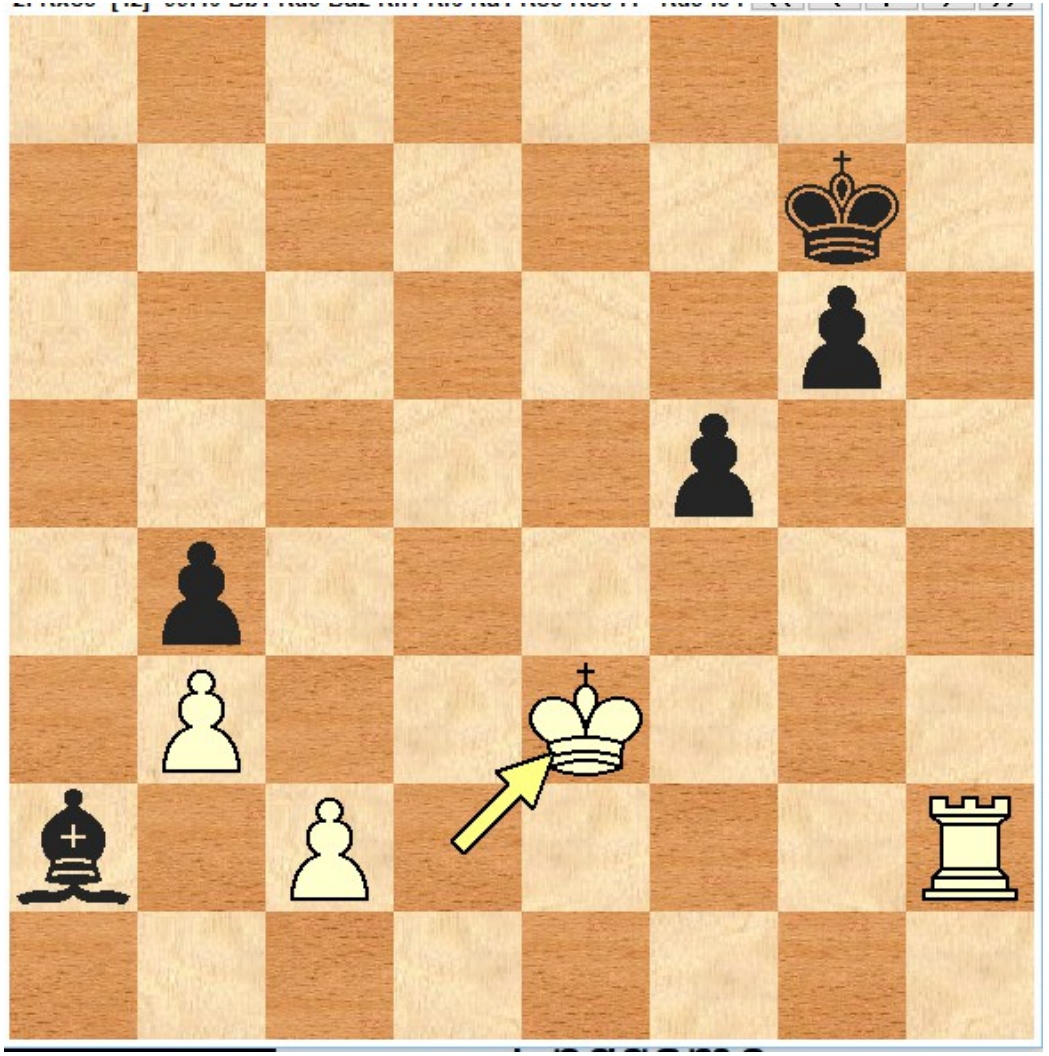
Cutoff-function



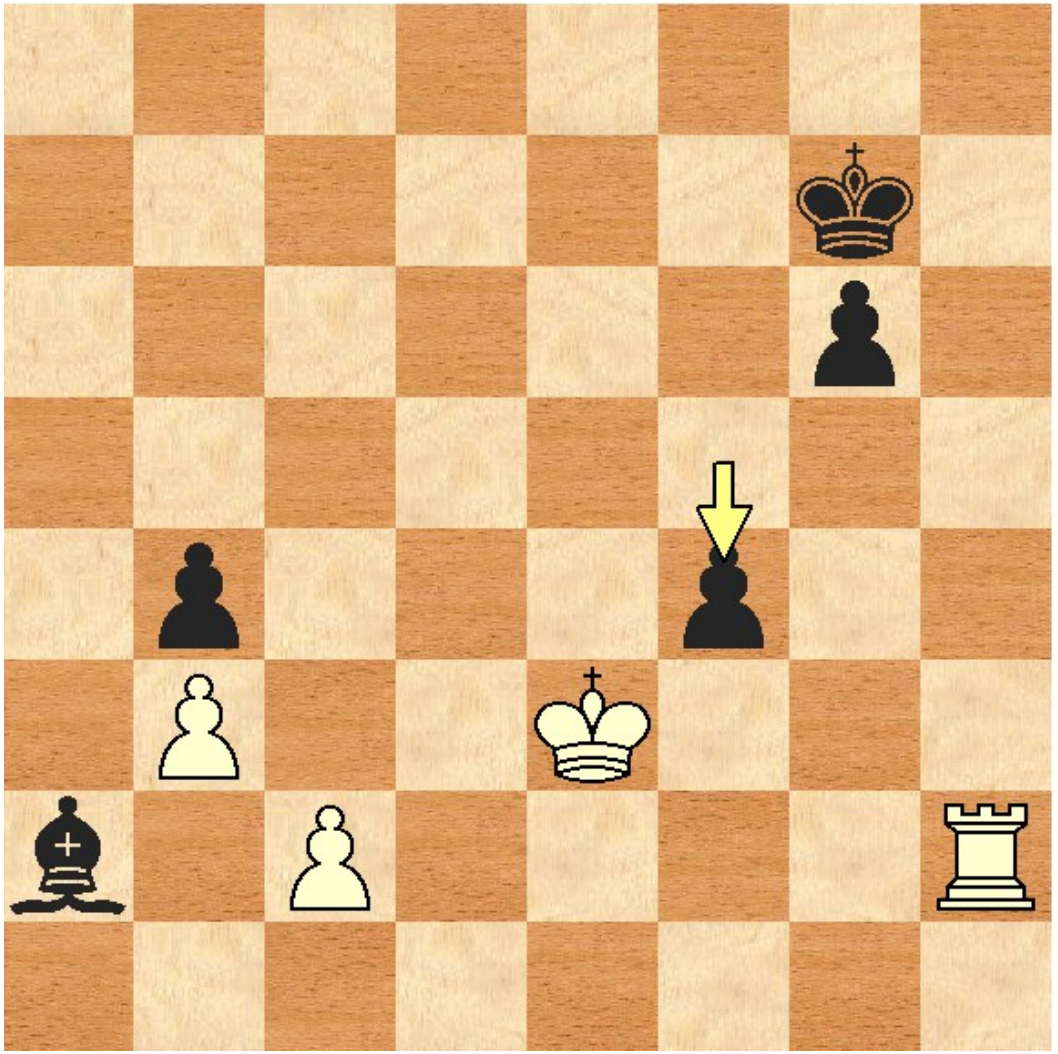
Cutoff-function



Cutoff-function



Cutoff-function



Search vs lookup

- Openings
  - For many games S_0 is always the same.
  - For the first moves, look them up.
- Endgame
  - The state complexity in some games is lower in the endgame
  - Compute all possible game states of a given situation
  - Compute their values using retrograde minimax search

Conclusion

- Can solve any game that is not to complex
- Complex games require some amount of game knowledge
- Chance games possible with chance nodes
- Foundation of many chess engines.
- Not as good in Games with a high branching factor and little Game knowledge like GO

Outline

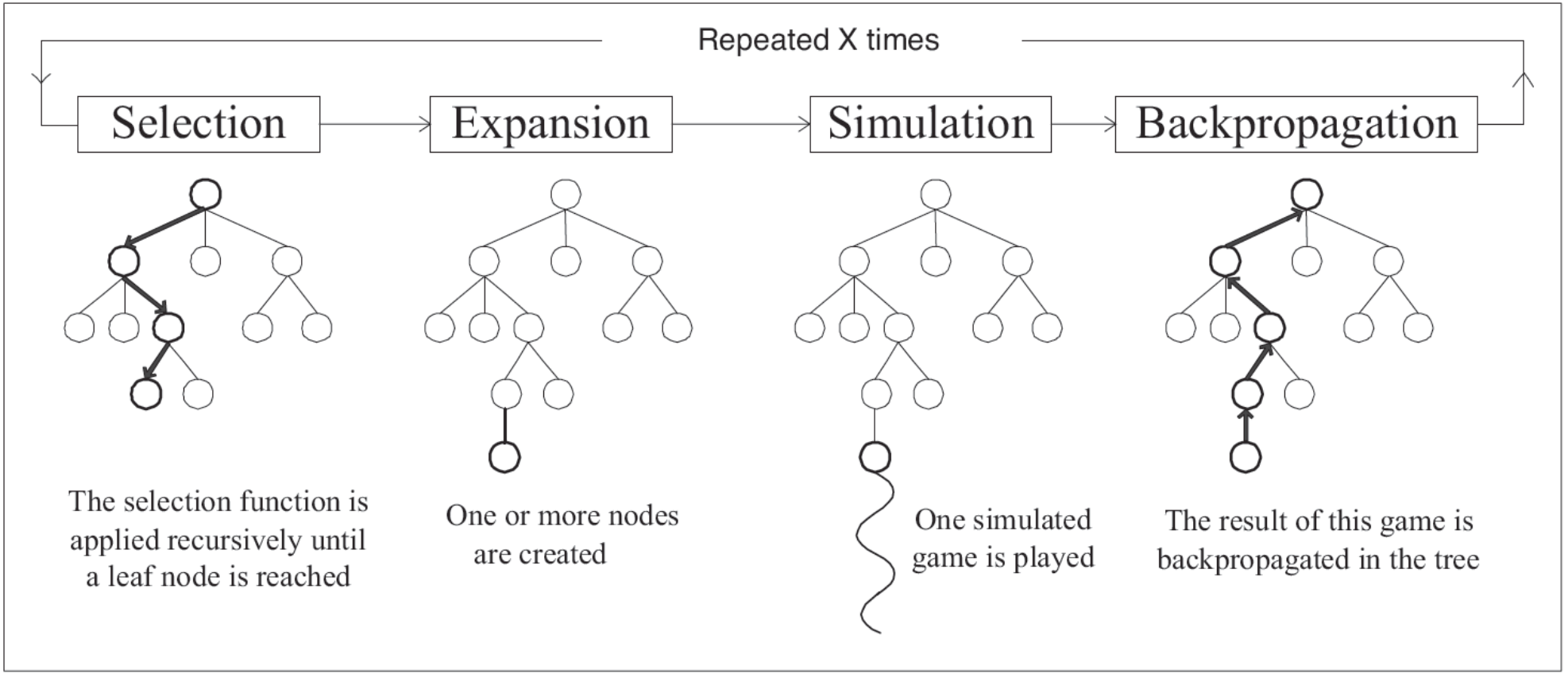
- Motivation
- Games
- Minimax algorithm
- Search vs lookup
- Monte Carlo tree search
  - Origin
  - Base algorithm
  - Exploration function
  - Domain knowledge application

Monte Carlo tree search
Origin

- No good evaluation function for GO
- Use random playouts to evaluate positions
- Promising results for game states that require strategic thinking
- Reaches it's optimum after a small number of playouts
- Focus on more promising nodes...

Monte Carlo tree search

Base algorithm



Monte Carlo tree search
exploration formula - multi arm bandit problem

- See the game tree as an multi arm Bandit problem
- Multi arm Bandit Problem
  - which slot machine should we choose
  - Solution:
    - Play each machine once.
    - Play the arm that maximizes: $X_i + \sqrt{\frac{2\log(n)}{n_i}}$

Monte Carlo tree search

Domain knowledge application

- Heavy playouts
  - Weight the chance of every move based on Game knowledge
- Node initialization
  - Initialize each expanded node with an evaluation function

MCTS - Conclusion

- No game knowledge needed.
- Support for chance games.
- Can be enhanced with game knowledge
- Quite good at Go

Sources

- Tomáš Kozelek (2009). Methods of MCTS and the game Arimaa (PDF). Master's thesis, Charles University in Prague.
- Stuart J. Russell and Peter Norvig (2016). Artificial Intelligence A Modern Approach Third Edition. Pearson
- Web Links:
  - https://en.wikipedia.org/wiki/Game_complexity
- Images:
  - https://www.flickr.com/photos/160866001@N07/30606387067/in/photolist-NCzJyK-2cjqMzw-6MW8AR-6M8gPj-8t1Cgq-bLed9T-bweHFg-DBPVdT-7K7Lkj-bxjB7Y-qx4xVd-bwfg7P-bwfgwa-bxjBc9-bLefDi-bLei24-bxjANd-bxjBy7-cB6RGS-bLegop-bLeguD-bLeehZ-bLeiFB-bxjx8N-bxjAxC-5ep9Qv-bLed5R-dYczef-bxjADG-bweXGa-bxjAcj-bxjC37-bLedR2-bLefMe-bxjxnA-bLegWM-bLedyr-bxjBGo-bxjxYJ-bxjBqb-bxjzzN-bxjzwN-bxjyd7-bLegd6-bxjy6y-bLedqV-9vU5J9-bLeeBt-bLeddc-bLeiXg

Sources

- https://pixabay.com/photos/checkmate-chess-resignation-1511866/
- https://publicdomainpictures.net/en/view-image.php?image=163474&picture=