Artificial Intelligence 4 Games

MCTS with Action-based Heuristics

2020

Jakub Kowalski, University of Wrocław

Forward Model

Also known as a **reasoner** or (game) **engine**.

A program that is able to simulate the course of the game, i.e. it implements it rules.

In simulation-based AI the efficiency of reasoner has a crucial impact on the player's results.

There is a set of methods that a reasoner has to implement to by functional.

- ➔ ***init: S***
  Returning the initial state of the game
- ➔ ***legal:*** $S \times P \to \mathcal{P}(A)$
  Returning a collection of moves that are legal for the given player in the given state
- ➔ ***apply:*** $S \times A \to S$
  Returning the successor game state after application of the given action
- ➔ ***isTerminal:*** $S \to \{T,F\}$
  True for the terminal game states
- ➔ ***reward:*** $S \times P \to \mathbb{R}$
  Returning each players' score for a given (usually terminal) state

Domain-independent Action-based Heuristics

We will discuss a commonly used set of heuristics that does not require knowledge about the particular game and, in most cases, they improve the results when applied to MCTS.

They are based on the assumption that a move that seems to be "good" (i.e., it contributes to the player's high reward) in a particular context (from a given state) it is likely to be good also in other contexts (different game states).

We may apply this heuristic on both MCTS phases, during selection and simulation, and both approaches may complement each other.

Simulation Strategy

- MAST
- PPA

Selection Strategy

- AMAF, RAVE
- GRAVE / HRAVE

Move-Average Sampling Technique

(MAST)

Literature

★ Finnsson, H., Björnsson, Y.: Simulation-based approach to General Game Playing. AAAI. vol. 8, pp. 259–264 2008.

★ Finnsson, H.: Simulation-based General Game Playing. Ph.D. Thesis. Reykjavik University, 2012

★ Tak, M., Winands, M., Björnsson, Y.: N-grams and the Last-Good-Reply policy applied in General Game Playing. TCIAIG, vol. 4, no. 2, pp. 73–83, 2012

Move Average Sampling Technique (MAST)

MAST bias the random action selection during the simulation phase based on the average score this action previously contributed to.

The idea is loosely related to move-ordering mechanism for Chess called *history-heuristic*.

For each player ***p***, and action ***a***, we introduce two global lookup tables:

- $N_{MAST}(p,a)$, storing the number of times the action was performed by the player.
- $W_{MAST}(p,a)$, storing accumulated player reward associated with the action.

At the end of each playout, we iterate through all visited moves (both in selection and simulation phase), increase N_{MAST} if the move occured, and add the player's reward to W_{MAST} .

When performing a simulation, choice of each action is based on the ε-greedy strategy over the average value

$$Q_{MAST}(p,a) = W_{MAST}(p,a)/N_{MAST}(p,a)$$

of each legal action ***a*** (assuming ***p*** is the current player in the state).

Parameters

- ε∈[0, 1] greediness for ε-greedy
  Higher the value, more exploration during the simulation phase (1 is pure random)
  Values in literature: 0.4
- Keeping statistics from the previous turns. Recommended to be true.
- Decaying previous turns statistics (usually called γ), e.g. 0.2.

Various Modifications

- Originally, instead of ε-greedy, the move choice probability was based on Gibbs measure, which later was shown to perform worse
- Tree-Only MAST is a variant of MAST that updates only the data of moves chosen in the MCTS tree (in the selection phase)
- N-Gram Selection Technique (NST) is an extension of MAST that keeps track of move sequences instead of single moves
- There exist more variants generalizing action-averaging over any predicate/feature.

Performance

- Generally MAST improves the results of the MCTS UCT
- There may be still some games where it performs worse (some papers raport Skirmish), but it should be rare
- It requires additional space (not much) and each simulation is slightly slower
- (But as the simulations are more informed, they may lead to shorter games, thus more simulations can be performed in a given time)
- Generally always worth to try
- Works well in combination with selection-based heuristics such as RAVE

Playout Policy Adaptation

(PPA)

Literature

★ Cazenave, T.: Playout policy adaptation for games. Advances in Computer Games. pp. 20-28, 2015

★ Cazenave, T., Diemert, E.: Memorizing the playout policy. Workshop on Computer Games. pp. 96-107, 2017

Playout Policy Adaptation (PPA)

Similarly to MAST, PPA guides search during the simulation phase by altering the move distribution. It stores weights for each possible move of a player.

When selecting, it chooses move with the probability proportional to the exponential of its weight, against all other available legal moves, accordingly to the Gibbs measure:

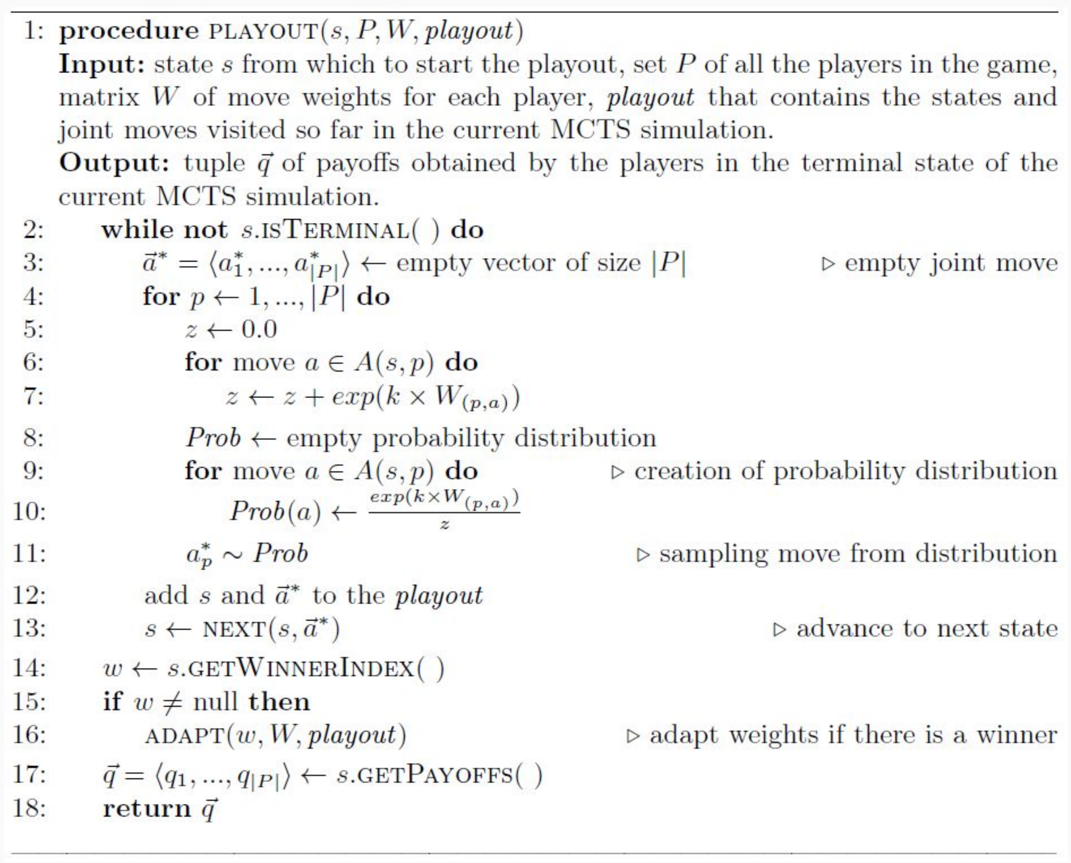
$$Prob(s,a) = \frac{e^{(Q(a)/\tau)}}{\sum_{a' \in A(s,p)} e^{(Q(a')/\tau)}}$$

At the end of each playout, weights of the moves visited by the winning player are increased by $\boldsymbol{\alpha}$.

At the same time, weights of all the moves of the winning player are proportionally decreased.

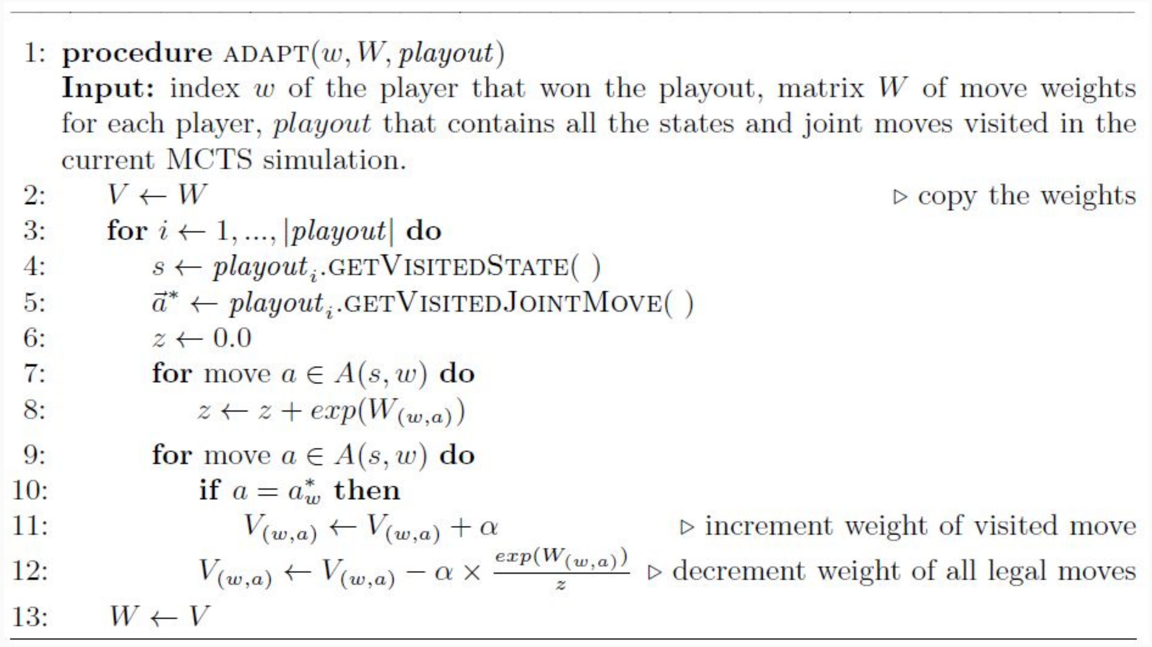
This is the feature distinguishing PPA from MAST the most, as it relies much more on the local context of the available moves.

PPA Pseudocode



```
procedure PLAYOUT($s, P, W, playout$)
    **Input:** state $s$ from which to start the playout, set $P$ of all the players in the game, matrix $W$ of move weights for each player, $playout$ that contains the states and joint moves visited so far in the current MCTS simulation.
    **Output:** tuple $\vec{q}$ of payoffs obtained by the players in the terminal state of the current MCTS simulation.
    while not $s$.ISTERMINAL( ) do
        $\vec{a}^* = \langle a_1^*, ..., a_{|P|}^* \rangle \leftarrow$ empty vector of size $|P|$        ▷ empty joint move
        for $p \leftarrow 1, ..., |P|$ do
            $z \leftarrow 0.0$
            for move $a \in A(s, p)$ do
                $z \leftarrow z + exp(k \times W_{(p,a)})$
            $Prob \leftarrow$ empty probability distribution
            for move $a \in A(s, p)$ do        ▷ creation of probability distribution
                $Prob(a) \leftarrow \frac{exp(k \times W_{(p,a)})}{z}$
            $a_p^* \sim Prob$        ▷ sampling move from distribution
        add $s$ and $\vec{a}^*$ to the $playout$
        $s \leftarrow$ NEXT($s, \vec{a}^*$)        ▷ advance to next state
    $w \leftarrow s$.GETWINNERINDEX( )
    if $w \neq$ null then
        ADAPT($w, W, playout$)        ▷ adapt weights if there is a winner
    $\vec{q} = \langle q_1, ..., q_{|P|} \rangle \leftarrow s$.GETPAYOFFS( )
    return $\vec{q}$
```

PPA Pseudocode



```
procedure ADAPT($w, W, playout$)
Input: index $w$ of the player that won the playout, matrix $W$ of move weights for each player, $playout$ that contains all the states and joint moves visited in the current MCTS simulation.
    $V \leftarrow W$                                                    ▷ copy the weights
    for $i \leftarrow 1, ..., |playout|$ do
        $s \leftarrow playout_i$.GETVISITEDSTATE( )
        $\vec{a}^* \leftarrow playout_i$.GETVISITEDJOINTMOVE( )
        $z \leftarrow 0.0$
        for move $a \in A(s, w)$ do
            $z \leftarrow z + exp(W_{(w,a)})$
        for move $a \in A(s, w)$ do
            if $a = a^*_w$ then
                $V_{(w,a)} \leftarrow V_{(w,a)} + \alpha$                  ▷ increment weight of visited move
            $V_{(w,a)} \leftarrow V_{(w,a)} - \alpha \times \frac{exp(W_{(w,a)})}{z}$    ▷ decrement weight of all legal moves
    $W \leftarrow V$
```

Parameters

- ➔ $\alpha \in [0, 1]$ learning rate
  Higher the value, more influential is the fact the move was visited during a play
  Values recommended in literature are 1.0 and 0.32
- ➔ ***k***=1/τ controls the shape of distribution in Gibbs measure
  High values of τ make it more uniform (higher exploration)
  Values used in literature: 1.0
- ➔ Keeping statistics from the previous turns
  Usually true
- ➔ Decaying previous turns statistics
  Usually false

Various Modifications

- ➔ We can update statistics for all the players, not only the winner, accordingly to their payoffs (rescaled to be from [0,1]).

$$W_{(p,a_p^*)} = W_{(p,a_p^*)} + \alpha \times q_p$$

$$W_{(p,a)} = W_{(p,a)} - \alpha \times q_p \times \frac{e^{W_{(p,a)}}}{\sum_{a' \in A(s,p)} e^{W_{(p,a')}}}, \forall a \in A(s,p)$$

- ➔ We can use ε-greedy instead of Gibbs. (literature shows no improvements)
- ➔ Similarly no improvement was observed with statistics decaying.
- ➔ PPA can be also mixed with N-grams of moves (more complicated).

Performance

- ➔ Wins against the random simulation policy given the same simulation budget (assuming UCB1 tree strategy for both)
- ➔ Rarely clearly wins against MAST, given the same simulation budget (α=0.32 performs better than 1.0)
- ➔ Because it performs more updates for each simulation, it is much slower than both random and MAST policies
- ➔ Still, there are some games where it was shown to perform generally better (Breakthrough, Sheep and Wolf), so it may be worth to give it a try.

Rapid Action Value Estimation

(RAVE)

Literature

★ Brügmann, B.: Monte Carlo Go. Tech. Rep., 1993

★ Gelly, S., Silver, D. Monte-Carlo tree search and rapid action value estimation in computer Go. Artificial Intelligence, vol. 175, no. 11, pp. 1856–1875, 2011

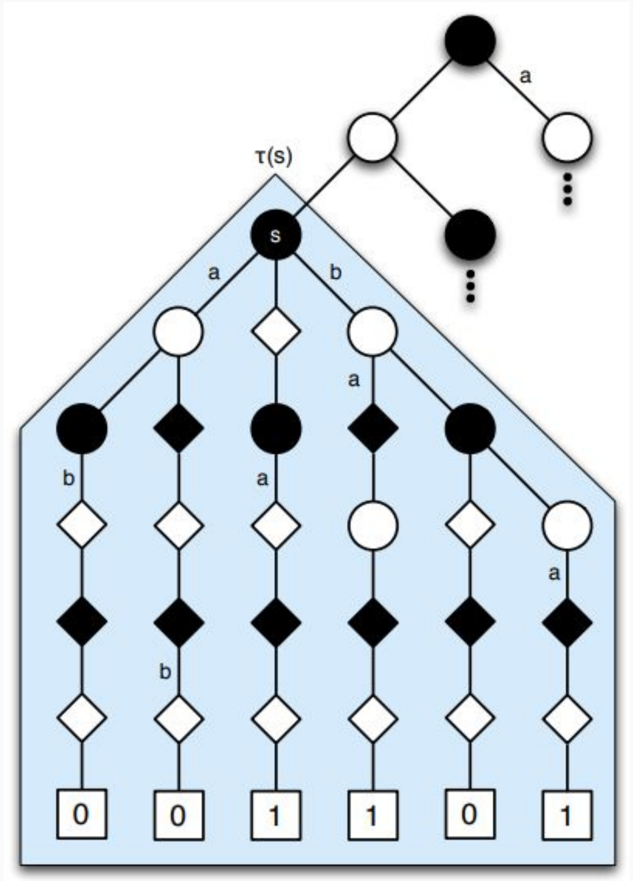
★ Sironi, C., Winands, M.: Comparison of rapid action value estimation variants for general game playing. CIG. 2016

All Moves As First (AMAF)

Originating from Go and the observation that the value of a move is often unaffected by the moves played elsewhere, AMAF collects statistics for all actions selected during a simulations as they were the first action applied.

For each state ***s*** in the MCTS tree and action ***a***, we need to compute a value $Q_{AMAF}(s,a)$ that is an average reward obtained from all simulations where ***a*** is played further down the path that passes ***s***.

Originally, AMAF heuristic was used to simply substitute the ***Q*** value with Q_{AMAF} value in the selection policy.



$Q(s,a)=0/2$

$Q(s,b)=2/3$

$Q_{AMAF}(s,a)=3/5$

$Q_{AMAF}(s,b)=2/5$

Gelly, S., Silver, D. Monte-Carlo tree search and rapid action value estimation in computer Go. Artificial Intelligence, vol. 175, no. 11, pp. 1856–1875, 2011.

Rapid Action Value Estimation (RAVE)

RAVE speeds up the learning process inside the MCTS tree. It tries to overcome the problem of undersampled nodes, which have high variance and thus low quality of estimated scores.

The idea is to use AMAF value (which is less accurate as it concerns less related statistics) for nodes with low number of visits and gradually switching to (more reliable) standard average payoff, as the number of visits increases.

The common usage formula is to use

$$Q_{RAVE} = (1 - \beta(s)) \times Q(s,a) + \beta(s) \times Q_{AMAF}(s,a)$$

where

$$\beta(s) = \sqrt{\frac{K}{3 \times N(s) + K}}$$

instead of ***Q(s,a)*** in the UCT formula.

- ***K***, is the *equivalence parameter* indicating for how many simulations the two scores have equal weight.

RAVE Pseudocode

```
procedure MC–RAVE($s_0$)
    while time available do
        SIMULATE($board, s_0$)
    end while
    $board.SetPosition(s_0)$
    return SELECTMOVE($board, s_0, 0$)
end procedure

procedure SIMULATE($board, s_0$)
    $board.SetPosition(s_0)$
    $[s_0, a_0, ..., s_T, a_T]$ = SIMTREE($board$)
    $[a_{T+1}, ..., a_D], z$ = SIMDEFAULT($board, T$)
    BACKUP($[s_0, ..., s_T], [a_0, ..., a_D], z$)
end procedure

procedure SIMDEFAULT($board, T$)
    $t = T + 1$
    while not $board.GameOver()$ do
        $a_t$ = DEFAULTPOLICY($board$)
        $board.Play(a_t)$
        $t = t + 1$
    end while
    $z = board.BlackWins()$
    return $[a_{T+1}, ..., a_{t-1}], z$
end procedure
```

```
procedure SIMTREE($board$)
    $t = 0$
    while not $board.GameOver()$ do
        $s_t = board.GetPosition()$
        if $s_t \notin tree$ then
            NEWNODE($s_t$)
            $a_t$ = DEFAULTPOLICY($board$)
            return $[s_0, a_0, ..., s_t, a_t]$
        end if
        $a_t$ = SELECTMOVE($board, s_t$)
        $board.Play(a_t)$
        $t = t + 1$
    end while
    return $[s_0, a_0, ..., s_{t-1}, a_{t-1}]$
end procedure
```

```
procedure EVAL($s, a$)
    $b = pretuned\ constant\ bias\ value$
    $\beta = \frac{\tilde{N}(s,a)}{N(s,a)+\tilde{N}(s,a)+4N(s,a)\tilde{N}(s,a)b^2}$
    return $(1 - \beta)Q(s, a) + \beta\tilde{Q}(s, a)$
end procedure
```

```
procedure SELECTMOVE($board, s$)
    $legal = board.Legal()$
    if $board.BlackToPlay()$ then
        return $\operatorname{argmax}_{a \in legal}$ EVAL($s, a$)
    else
        return $\operatorname{argmin}_{a \in legal}$ EVAL($s, a$)
    end if
end procedure

procedure
BACKUP($[s_0, ..., s_T], [a_0, ..., a_D], z$)
    for $t = 0$ to $T$ do
        $N(s_t, a_t)$ ++
        $Q(s_t, a_t)$ += $\frac{z - Q(s_t, a_t)}{N(s_t, a_t)}$
        for $u = t$ to $D$ step 2 do
            if $a_u \notin [a_t, a_{t+2}, ..., a_{u-2}]$ then
                $\tilde{N}(s_t, a_u)$ ++
                $\tilde{Q}(s_t, a_u)$ += $\frac{z - \tilde{Q}(s_t, a_t)}{\tilde{N}(s_t, a_t)}$
            end if
        end for
    end for
end procedure
```

Performance

- ➔ The simulation speed of RAVE is usually a dozen percents slower than that of pure UCT.
- ➔ But the single MCTS iteration is of higher quality.
- ➔ Memory usage is increased, as in every tree node additional data for each legal move in this state has to be stored.
- ➔ RAVE also benefits from MAST.

- ➔ RAVE is generally considered as a heuristic that always improves the results.
- ➔ It heavily depends on the proper choice of the ***K*** value .
- ➔ (Which is closely correlated with the agent’s simulation speed and allowed time limit.)

Generalized Rapid Action Value Estimation

(GRAVE)

HRAVE

Literature

★ Cazenave, T.: Generalized rapid action value estimation. IJCAI. pp. 754–760, 2015

★ Sironi, C., Winands, M.: Comparison of rapid action value estimation variants for general game playing. CIG. 2016

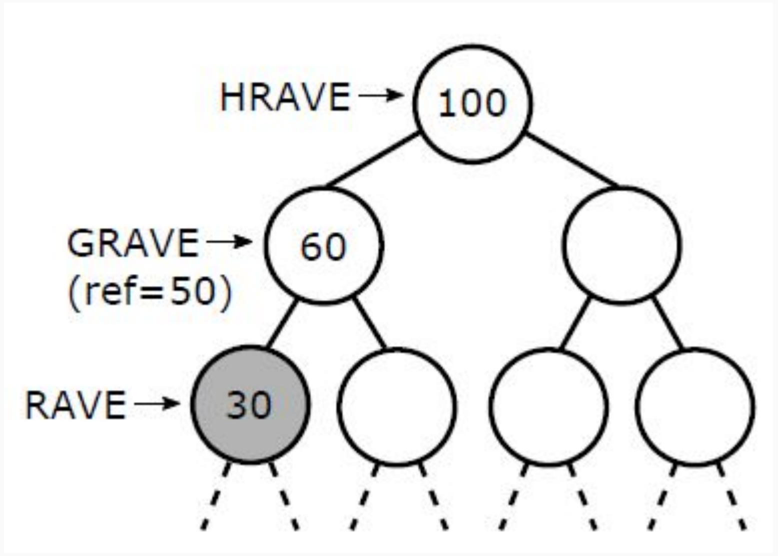
Generalized Rapid Action Value Estimation

One of the RAVE drawbacks is that near the leaves of the MCTS tree the estimation is still of low quality, as both ***Q*** and ***Q****AMAF* are computed based on a low number of samples.

The idea is to use AMAF values of some ancestor node if the number of visits is too low.

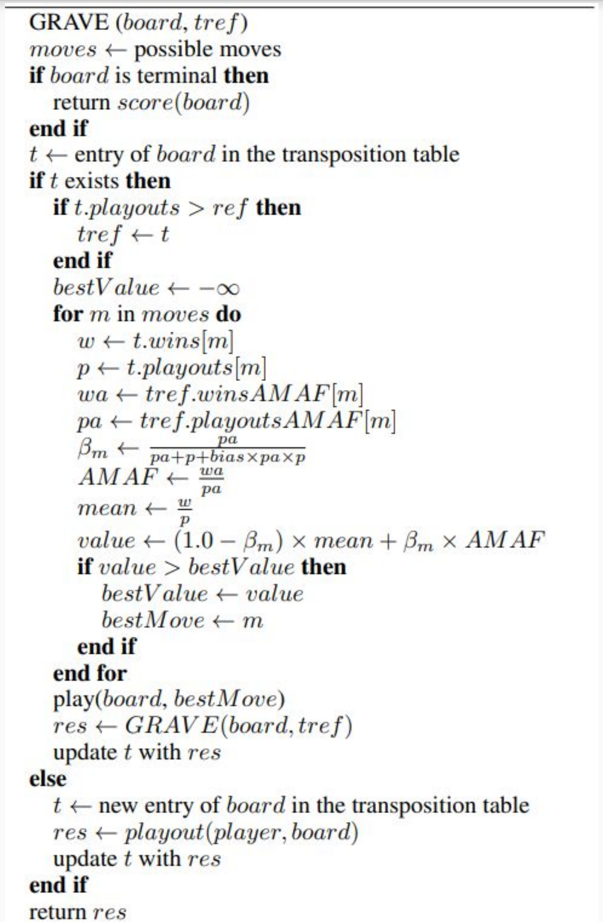
Thus, we are backtracking until we find a node that have more visits than a parameter ***ref***.

HRAVE is a specific variant of GRAVE that always uses the AMAF values of the current root of the tree (i.e., ***ref***=∞)



Sironi, C., Winands, M.: Comparison of rapid action value estimation variants for general game playing. CIG. 2016

GRAVE Pseudocode



```
GRAVE ($board, tref$)
$moves \leftarrow$ possible moves
**if** $board$ is terminal **then**
    return $score(board)$
**end if**
$t \leftarrow$ entry of $board$ in the transposition table
**if** $t$ exists **then**
    **if** $t.playouts > ref$ **then**
        $tref \leftarrow t$
    **end if**
    $bestValue \leftarrow -\infty$
    **for** $m$ in $moves$ **do**
        $w \leftarrow t.wins[m]$
        $p \leftarrow t.playouts[m]$
        $wa \leftarrow tref.winsAMAF[m]$
        $pa \leftarrow tref.playoutsAMAF[m]$
        $\beta_m \leftarrow \frac{pa}{pa+p+bias \times pa \times p}$
        $AMAF \leftarrow \frac{wa}{pa}$
        $mean \leftarrow \frac{w}{p}$
        $value \leftarrow (1.0 - \beta_m) \times mean + \beta_m \times AMAF$
        **if** $value > bestValue$ **then**
            $bestValue \leftarrow value$
            $bestMove \leftarrow m$
        **end if**
    **end for**
    play($board$, $bestMove$)
    $res \leftarrow GRAVE(board, tref)$
    update $t$ with $res$
**else**
    $t \leftarrow$ new entry of $board$ in the transposition table
    $res \leftarrow playout(player, board)$
    update $t$ with $res$
**end if**
return $res$
```

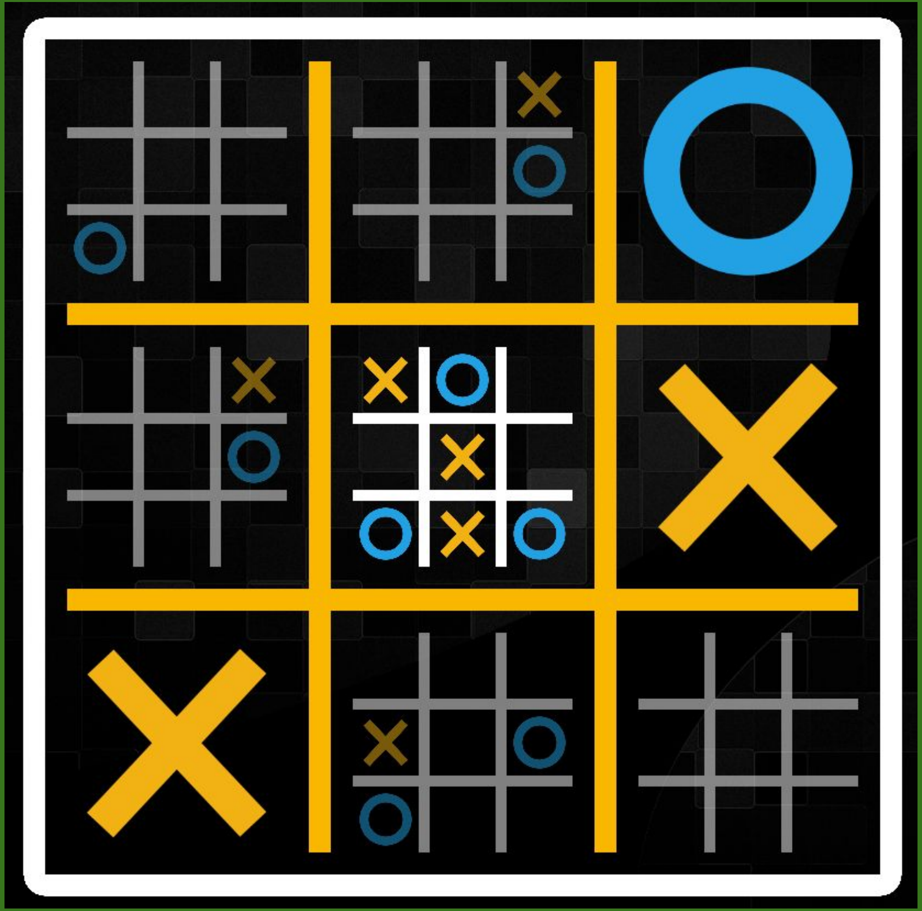
Performance

- ➔ The simulation speed of GRAVE/HRAVE is comparable or slower than the one of RAVE
- ➔ MCTS iterations are of better quality, so GRAVE clearly wins when the same simulation budget is given.
- ➔ GRAVE and HRAVE are much more memory consuming as they require to store AMAF values for every possible moves (actually legal in the lower part of the tree), not only the ones legal in the state.
- ➔ In practice the number of move statistics required to be stored in GRAVE may be much larger!
- ➔ Performances of RAVE, GRAVE and HRAVE are very close, with GRAVE having a larger potential to optimize given its additional ***ref*** parameter.
- ➔ Generally GRAVE is rarely worse than RAVE
- ➔ GRAVE still benefits from MAST but the strength improvement is smaller than in case of RAVE.
- ➔ RAVE/GRAVE/HRAVE performs significantly better with lower ***C*** constant value (e.g. 0.2, instead of usual UCT values 0.4, 0.7).

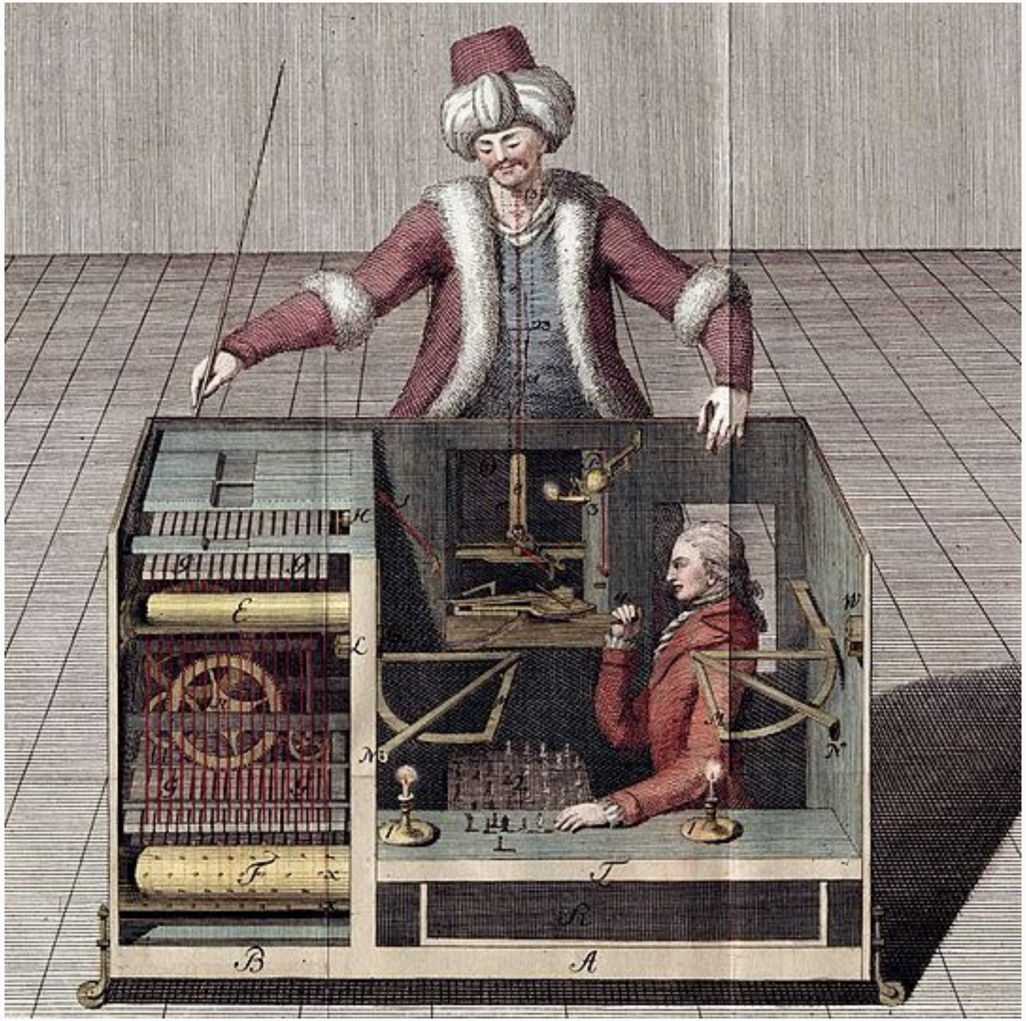
Summary

Summary

- ★ Use MCTS where it suits, it is a really nice algorithm
- ★ Tune it a lot to get it working
- ★ Apply action-based heuristics, as they will most likely improve the results



Thanks!



Bonus reference quiz