Minimax Revisited, Alpha-Beta pruning

Note Title

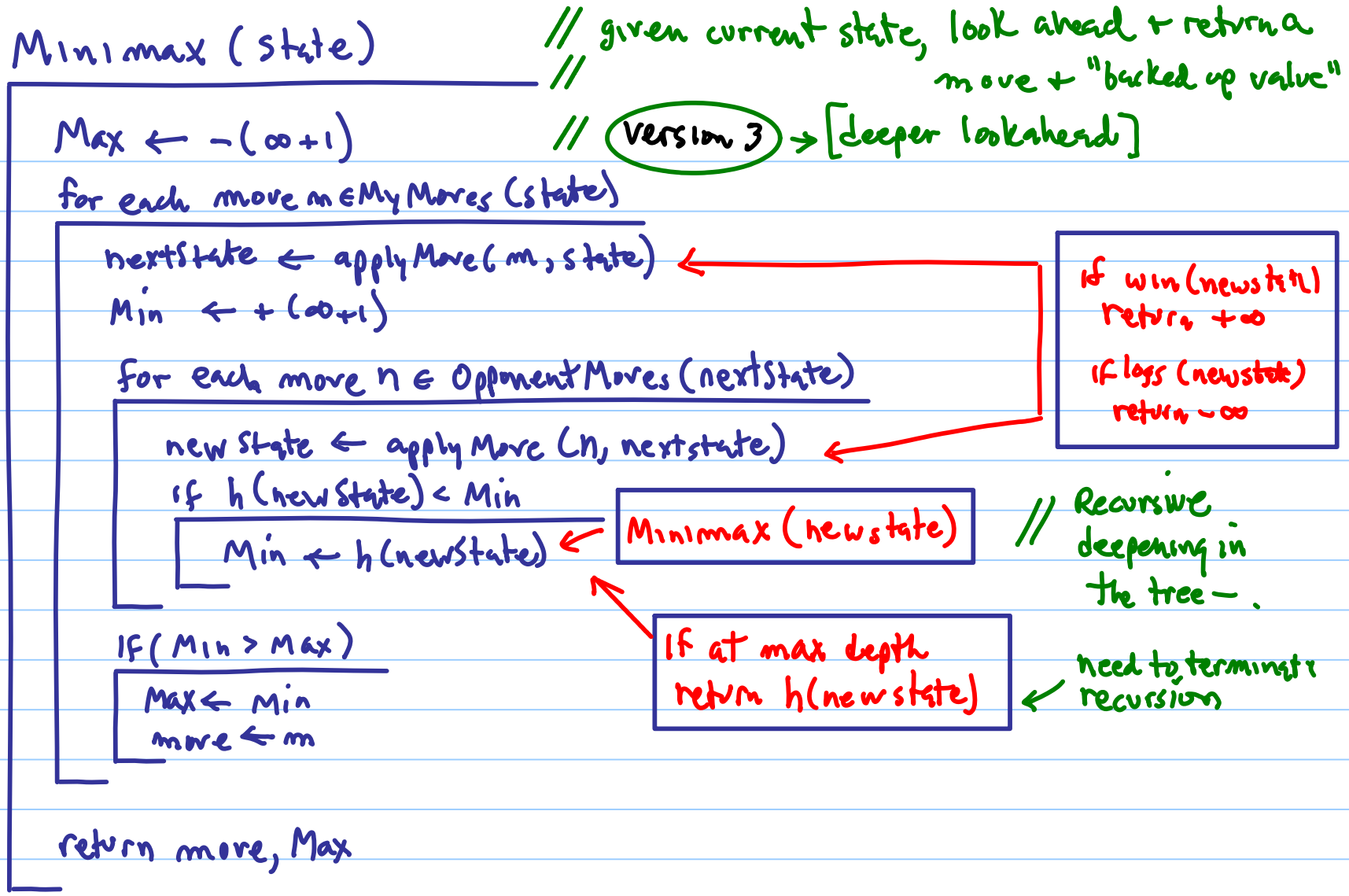
Version 2 of Minimax does a 2-move (1 ply) lookahead – one move each for you + your opponent

- returns your move + backed-up value
- does not matter what your opponent's moves are, just their backed-up values.
- employs heuristic at bottom level.

  opponent's move chosen by backed-up (minimum) heuristic values of possible moves

  your move chosen as maximum of opponent's backed-up values.

Idea: instead of using heuristic at level 2, why not look ahead further? (This is more reliable than direct use of heuristic)


Minimax (state)
// given current state, look ahead & return a move & "backed up value"
//
Max ← -(∞+1)
// Version 3 → [deeper lookahead]
for each move m ∈ MyMoves (state)
nextState ← applyMove(m, state)
Min ← +(∞+1)
for each move n ∈ OpponentMoves (nextState)
newState ← applyMove (n, nextstate)
if h(newState) < Min
Min ← h(newState)
IF (Min > Max)
Max ← Min
move ← m
return move, Max
if win (newstate) return +∞
if loss (newstate) return -∞
Minimax (newstate)
// Recursive deepening in the tree.
If at max depth return h(newstate)
need to terminate recursion

Yet another version : Note that Version 1 → 2 move lookahead
2 → 2k move lookahead

Can also do an odd number of moves =

Another version : **Negamax** : Instead of your move : max f(x)
opponent : min f(x)

do your move : max f(x)
opponent : −(max −f(x))

Can collapse "two-move lookahead" into
"one-move lookahead, plus factor
of −1 * backed-up value"
to handle even or odd # moves

PRUNING

Search trees grow exponentially.

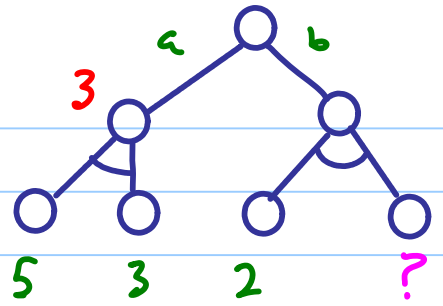
If it is possible to avoid searching every node in a tree, the savings in effort may be worthwhile.

Pruning refers to removing branches from a tree. As in gardening, the closer a cut is made to the root, the more branches are removed.

Taking advantage of symmetry is one type of pruning we have seen.

Alpha-Beta pruning is accomplished with very little extra code and can improve search results dramatically by removing branches that have no impact on the final result in a Minimax search.

Example

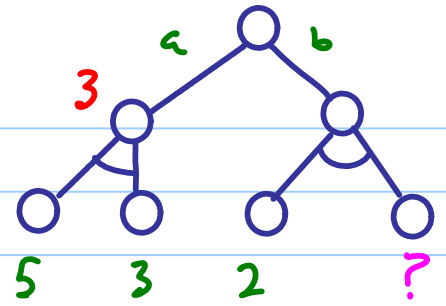


our move

opponent move

What move do we make?

Example



our move

opponent move

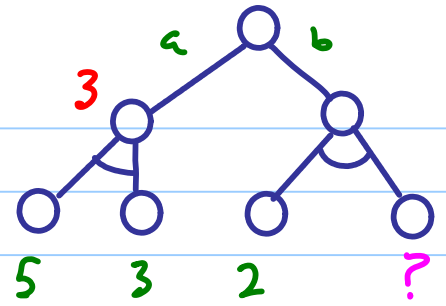
What move do we make?

Idea: Take a: ? could be lousy $(-\infty)$

Suppose ? is $+\infty$

? is $-\infty$

Example



our move

opponent move

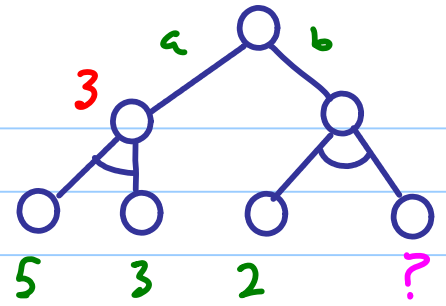
What move do we make?

Idea: Take a: ? could be lousy ($-\infty$)

Suppose ? is $+\infty$ opponent chooses "2" move → we choose "3" move

?

Example



our move

opponent move

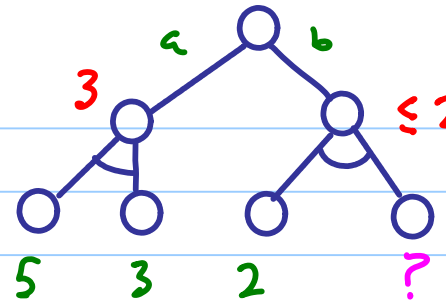
What move do we make?

Idea: Take a: ? could be lousy $(-\infty)$

Suppose ? is $+\infty$ opponent chooses "2" move → we choose "3" move

? is $-\infty$ opponent chooses "$-\infty$" move → we choose "3" move

Example



our move

$\leq 2 = \min(2, ?)$

opponent move

What move do we make?

Idea: Take a : ? could be lousy $(-\infty)$

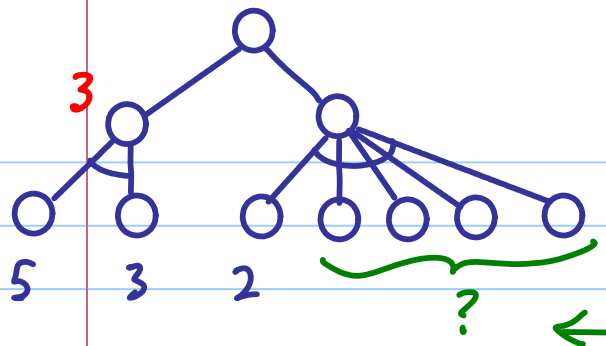
Suppose ? is $+\infty$ opponent chooses "2" move → we choose "3" move

? is $-\infty$ opponent chooses "$-\infty$" move → we choose "3" move

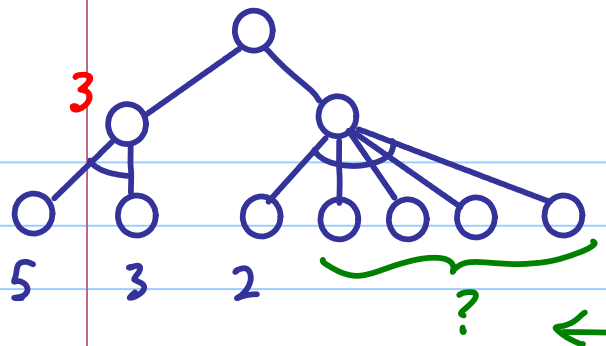
NOTE: we have 3 in hand (from move a)

move b ⇒ ≤ 2

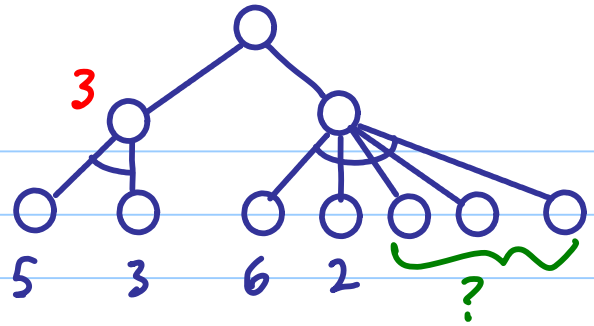
So, we don't care about value of ?



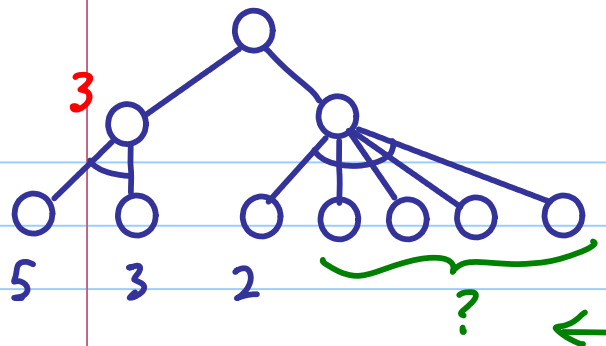
we don't care about any of these



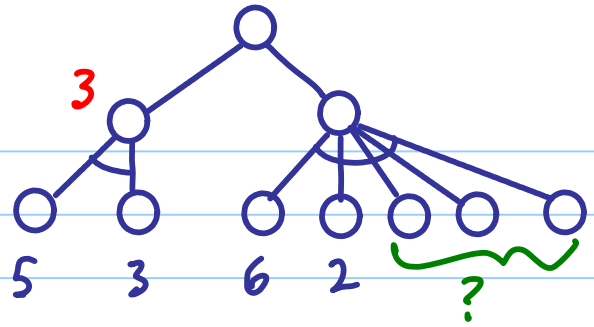
we don't care about any of these



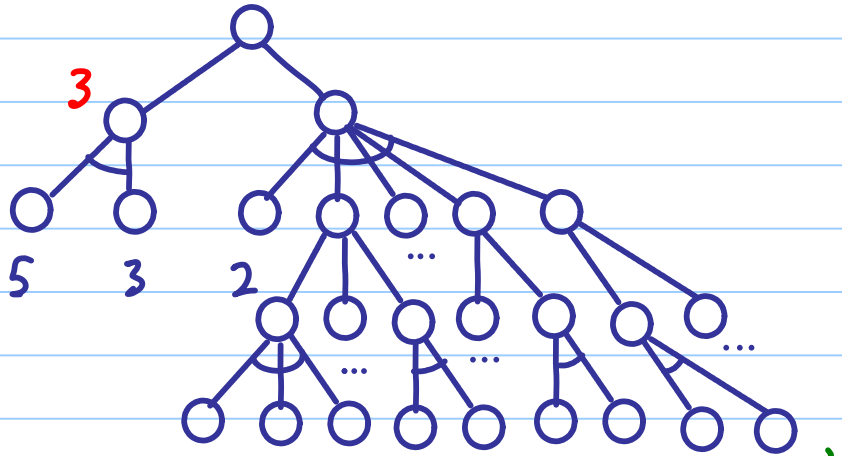
or these



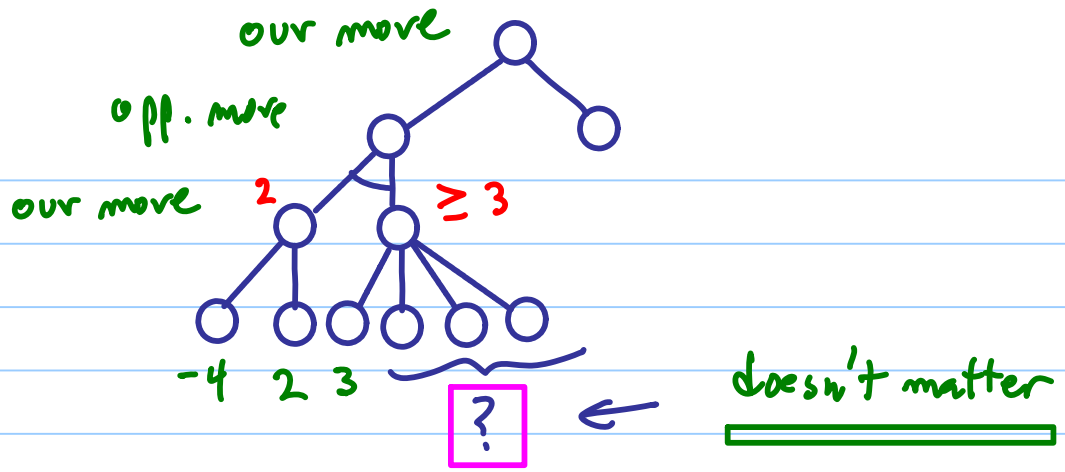
we don't care about any of these

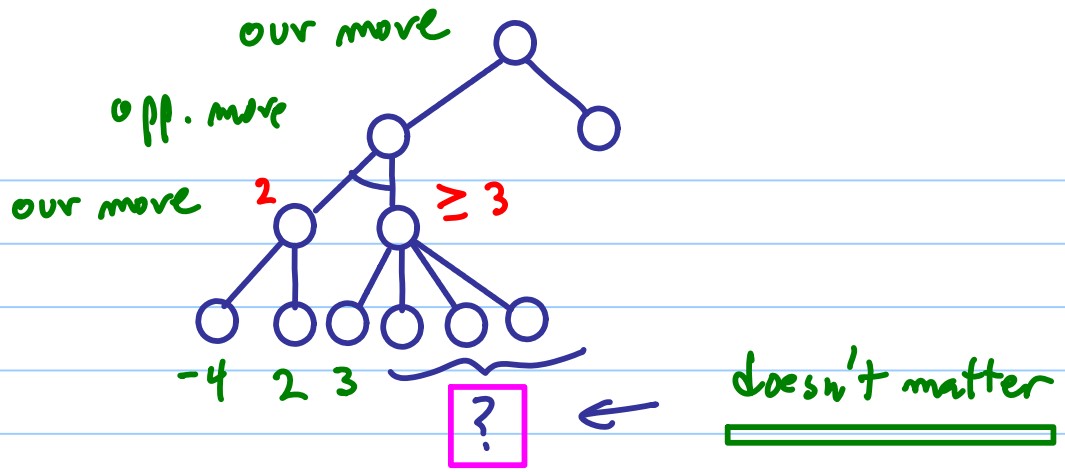


or these

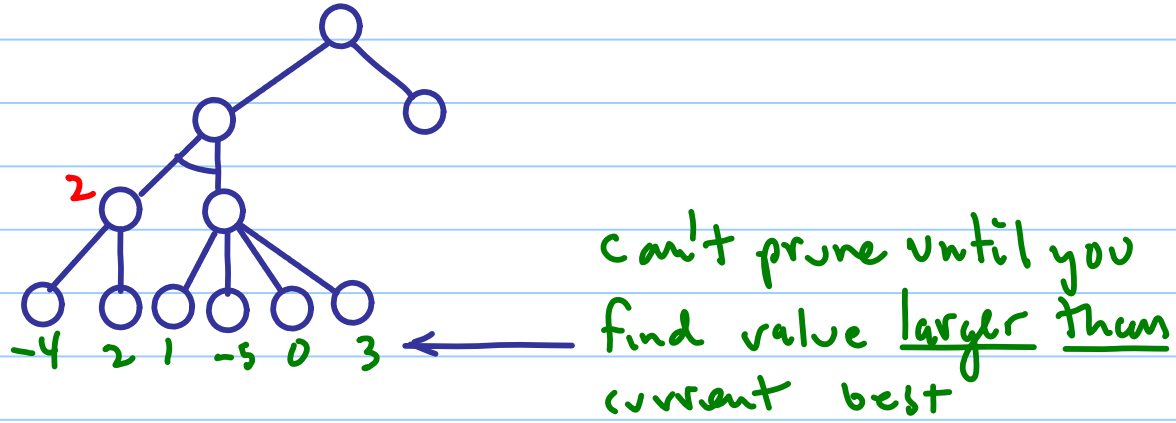


↑ we don't care about any of these, either!


our move
opp. move
our move
2
≥ 3
-4
2
3
?
doesn't matter

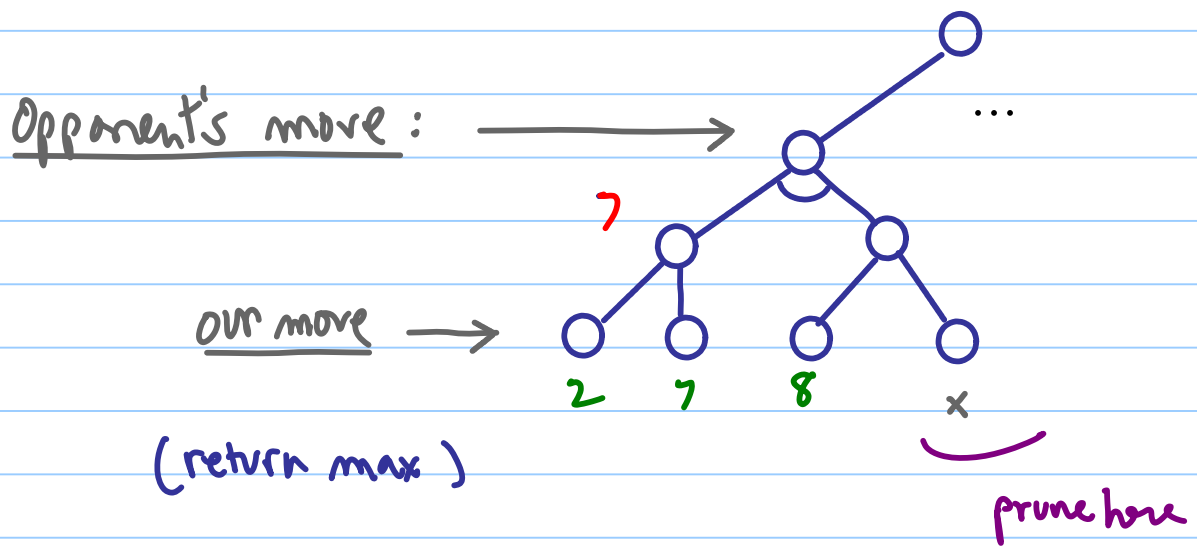


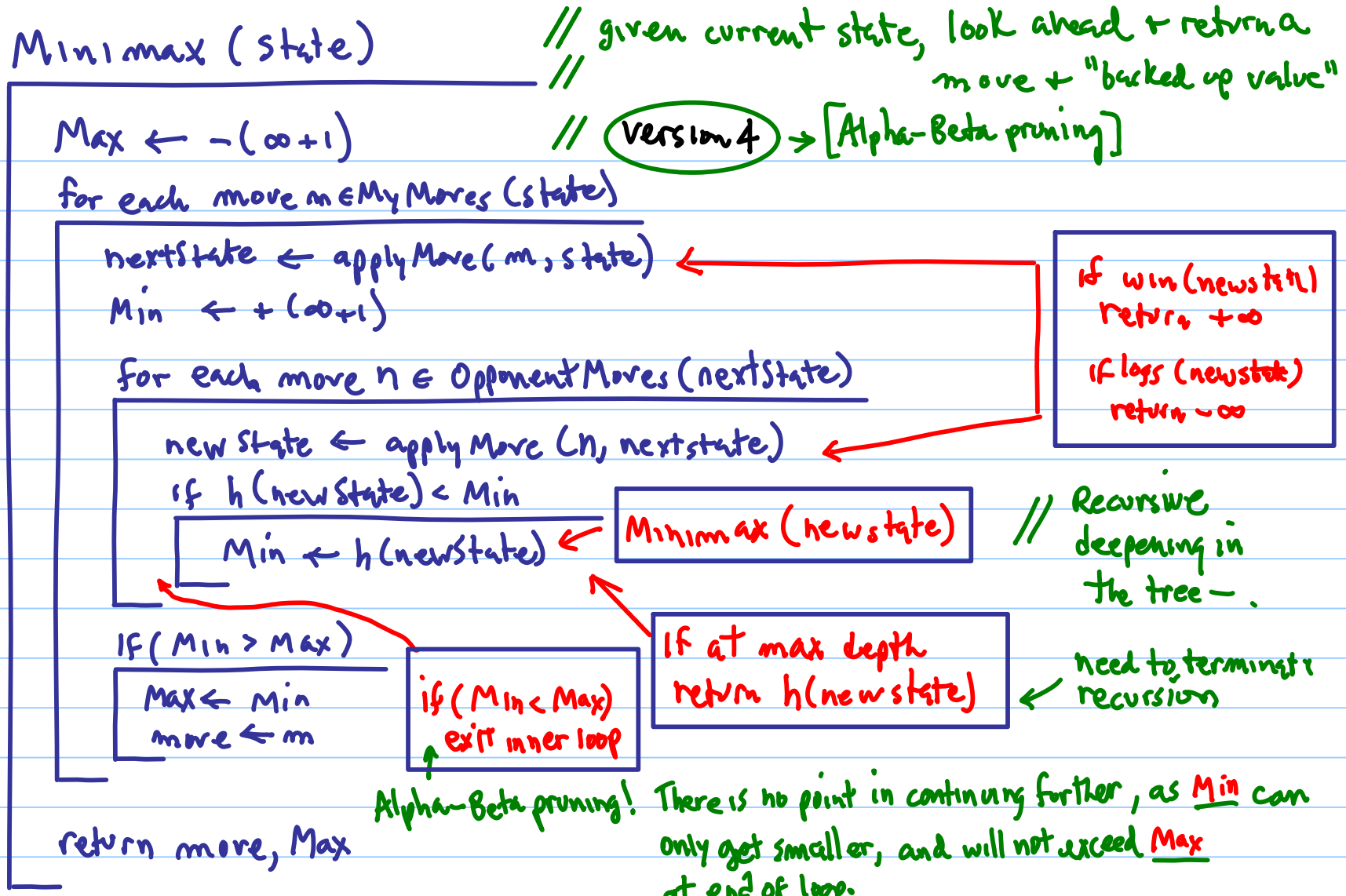
Moral of story : arrange moves so that ones likely to have best values get evaluated first



$\alpha - \beta$ pruning = maintain value of current best for you (max = α)
current best for opponent at next level (min = β)

prune when $\alpha > \beta$




Minimax (state)
// given current state, look ahead & return a move & "backed up value"
//
Max ← -(∞+1)
// Version 4 → [Alpha-Beta pruning]
for each move m ∈ MyMoves (state)
nextState ← applyMove(m, state)
Min ← +(∞+1)
for each move n ∈ OpponentMoves (nextState)
newState ← applyMove (n, nextstate)
if h(newState) < Min
Min ← h(newState)
Minimax (newstate)
// Recursive deepening in the tree.
if win(newstate) return +∞
if loss (newstate) return -∞
IF (Min > Max)
Max ← Min
move ← m
if (Min < Max) exit inner loop
Alpha-Beta pruning!
If at max depth return h(newstate)
need to terminate recursion
return move, Max
There is no point in continuing further, as Min can only get smaller, and will not exceed Max at end of loop.

More ideas : More intelligence

Order list of moves with a heuristic ("static evaluation function")
which rates some moves as probably better than others —
+ evaluates those first —
Can help with more pruning

(e.g. in chess, rate moves where you capture a piece
higher than other moves
- in end, backed-up value tells whether it
was really a good move -
helps find good moves faster