CS310 – AI Foundations

Andrew Abel

January 2024

# Week 4: Adversarial Search

## Welcome!

- Welcome to CS310 week 4!
- Adversarial search
  - Introducing dealing with another agent
  - Handling small games
  - Minimax Algorithm
- Pruning Game Trees
  - Alpha-Beta pruning
  - Making your algorithm more efficient

# Adversarial Search

## Learning Outcomes

- At the end of this topic, you will know:
  - How two-player zero-sum games are an extension of the single agent problems we've examined so far
  - What a game tree looks like
  - How we can solve a game tree by assuming that the opponent plays rationally
  - How the Minimax algorithm works

## Games & Multi-agent environments

- Which Problems can we Solve (see week 2)?
- Task environments suitable for our search algorithms so far:
  - Fully observable (the easy option)
  - Single agent (the easy option)
  - Deterministic (the easy option)
  - Sequential (the normal option)
  - Static (the easy option)
  - Discrete (the easy option)

# Games

- To play a game, we need to relax the assumption that only one agent can change the state of the world.
  - Need to move away from static and deterministic search trees/graphs!
- Game theory: any environment with multiple agents in it can be regarded as a game.
- In AI, simple games are often what game theorists would call deterministic, turn-taking, two-player, zero-sum games of perfect information.
  - Examples: chess, checkers, Connect 4, Shogi, Othello, go, tic-tac-toe, dots and boxes…
  - Deterministic: no element of chance
  - Turn-taking: players take turns
  - Two player – 2 agents
  - Zero sum – 1 person wins, other loses.  Can’t both win
  - Perfect information – Everybody has all information about state of the game

## Features of these Games

- Fully observable: game state is visible to both players
    - Perfect information
- Deterministic: no element of chance
- Sequential: action taken now affects future choices
- Static: the world doesn’t change during deliberation
- Discrete: in each game state, there are a finite number of moves
- Known: the outcomes of an action are fully defined
- **Multi agent**: in the search, we have to allow for the fact that our opponent can also affect the game state when it is their turn, and will be planning against us

# Min-Max Game

## Representing a Game

- We are going to deal with a game where we have 1 opponent, so a 2 player game that is perfect information, turn based and zero sum
  - This is a min-max game
- We need:
  - An initial state, including who plays first
  - A successor (state) function, like our Nextstates() python function
  - A terminal state/test which determines whether a given state is a end state of the game (i.e. the game is over)
  - A utility function - for terminal states only, this is the reward each player gets (e.g. +1 for win, -1 for lose)
  - In a zero-sum game, the wins and losses for the two players sum up to zero (e.g. one player wins £7, the other one loses £7)

## The Two Players: Max and Min

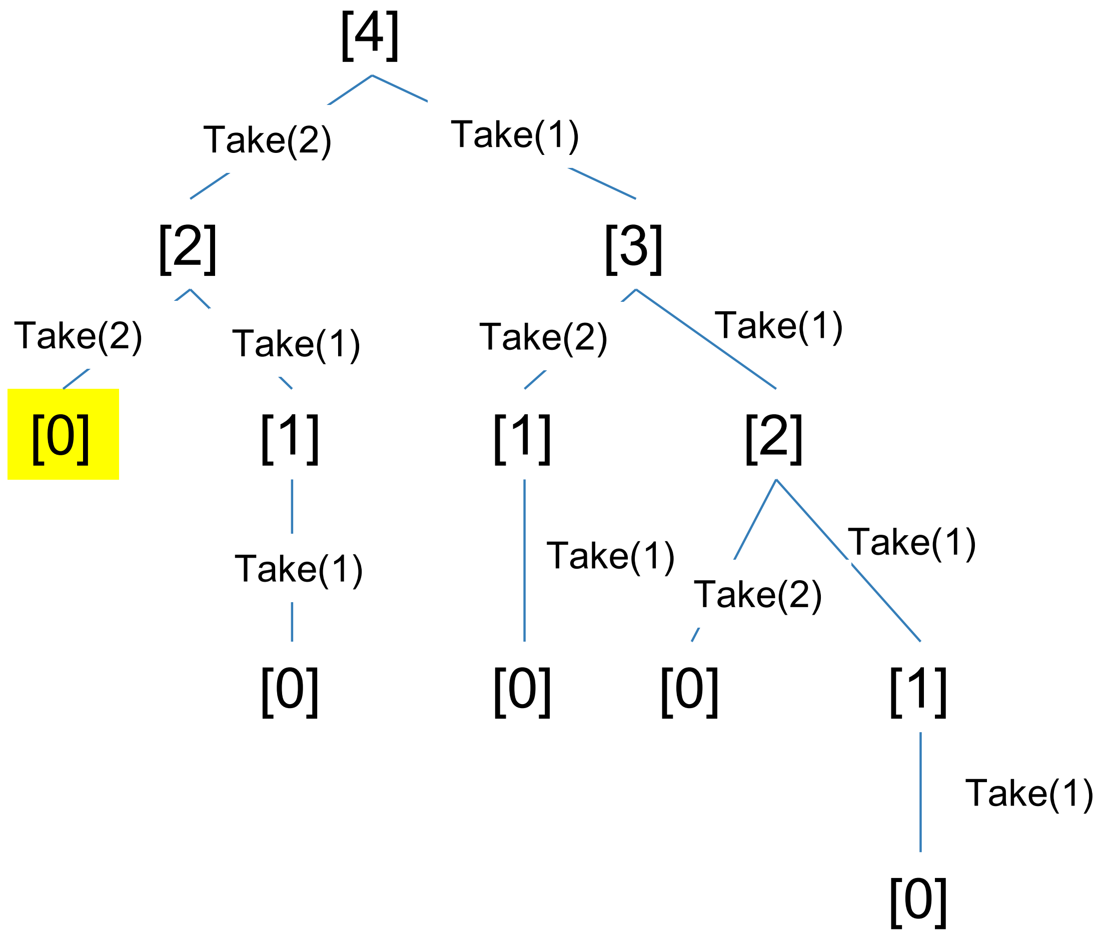
- Let's call the two players Max and Min: Max goes first
- Max's task is to maximise Max's reward (i.e. win!)
- Min's task is to minimise Max's reward (and therefore maximise Min's reward)
- Let's say Max can take actions *a, b* or *c* - which one will give Max the best reward when the game ends?
- To answer that question, we need to explore the game tree to a sufficient depth, and assume that Min plays optimally to minimise the reward that Max gets
  - The game tree is like a search tree, but different layers belong to different agents
  - Cannot just look for one path, because we need cooperation of an opponent

# An Example

- The world’s least fun game:
  - Four coins in a row, each player can pick up one coin or two coins.
  - The player who picks up the last coin wins.
  - Max plays first. What move should Max make?
- Initial State: Table with 4 coins
- Actions: take(1), take (2)

# Considering as a Search Tree

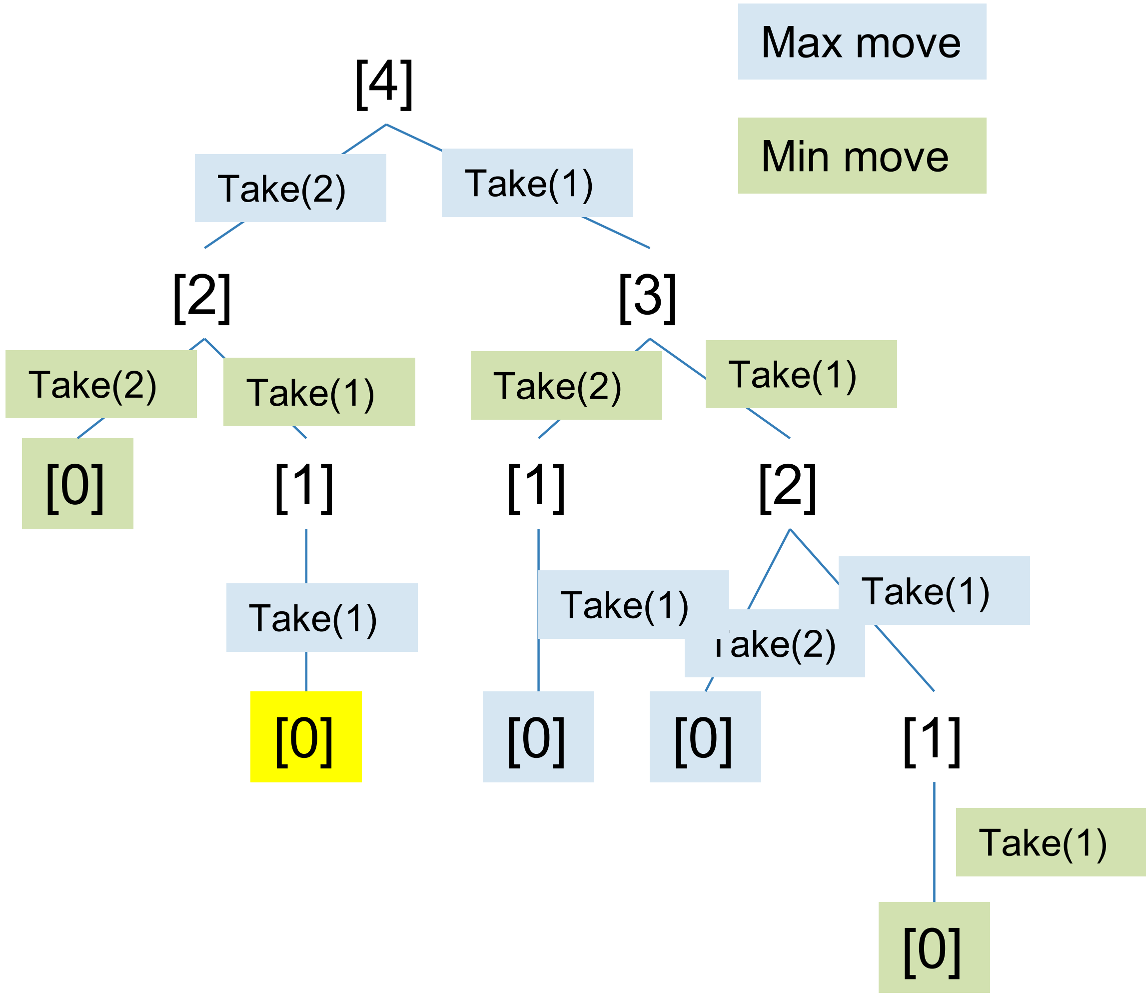
- Building a search tree of possible moves
- Immediate optimal solution on left of tree, would find first with breadth first or depth first!
- May even find first with heuristics
- But, this is a game tree, not a search tree, so we need to consider differently



# Considering as a Game Tree

- Different layers belong to different agents
- Each player makes an action
- Terminal nodes, [0]
- Utility function (reward)
  - +1 depending on player
- Possible optimal search path to victory through

  max(2), min(2), max(1)
- But playing optimally, min will never do that! So its not a sensible winnable path.



# MiniMax Algorithm

## The Minimax Algorithm

- Should allow us to find an optimal solution for a 2 player zero sum game with perfect information
- A structured way of solving the problem introduced in previous slides
- Considers the tree as a game tree rather than a search tree
- Computes the payoff on the assumption that both players play the optimal strategy

## The Minimax Algorithm

- The Minimax Value
  - The minimax value of a node is the utility of the node if the node is a terminal  (i.e. the reward)

  - If the node is a non-terminal Max node, the minimax value of the node is the maximum of the minimax values of all of the node's successors

  - If the node is a non-terminal Min node, the minimax value of the node is the minimum of the minimax values of all of the node’s successors

  - Recursive definition: results in a depth-first traversal of the game tree
    - Start at the terminal, work your way backwards
    - Also known as Backwards Induction

# The Minimax Algorithm

- MaxValue(state) returns a utility value

  if Terminal-Test(state) then return Utility(state)

  v ← **MinimalGameValue ( = - ∞ )**

  for s in Successors(state) do

  v ← Max(v, MinValue(s))

  return v

- MinValue(state) returns a utility value

  if Terminal-Test(state) then return Utility(state)

  v ← **MaximalGameValue ( = + ∞ )**

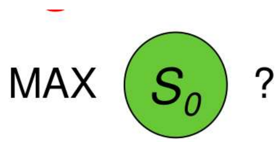
  for s in Successors(state) do

  v ← Min(v, MaxValue(s))

  return v

- If it's a terminal node, return the reward (utility)
- If its not, initially, not knowing the details, we can assume the worst for the player
- Can either calculate min value, or assume – infinity
- For successors, we compute the min values, and then take the max of the min value
- Vice versa for minValue

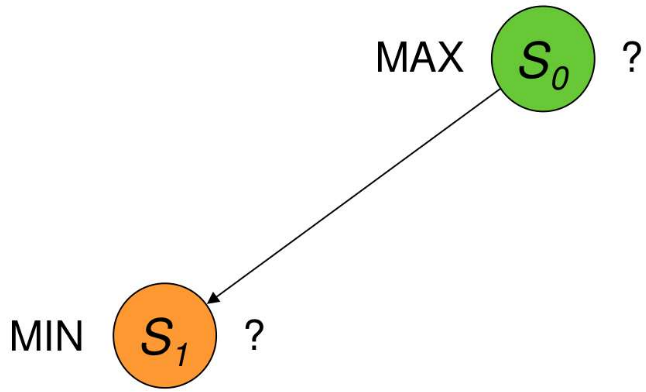
# The Minimax Algorithm: Step-By-Step




Carry out maxValue function
Want to known the minimum possible value

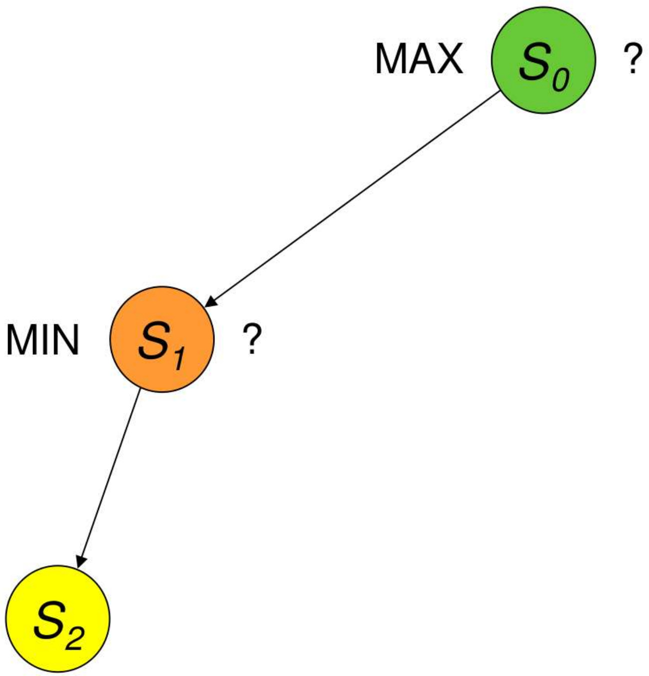
Still undecided as not a terminal known
Have to consider successors

# The Minimax Algorithm: Step-By-Step




Min value of first successor
Again, unknown, so max value,
or + infinity
Need to look at successors

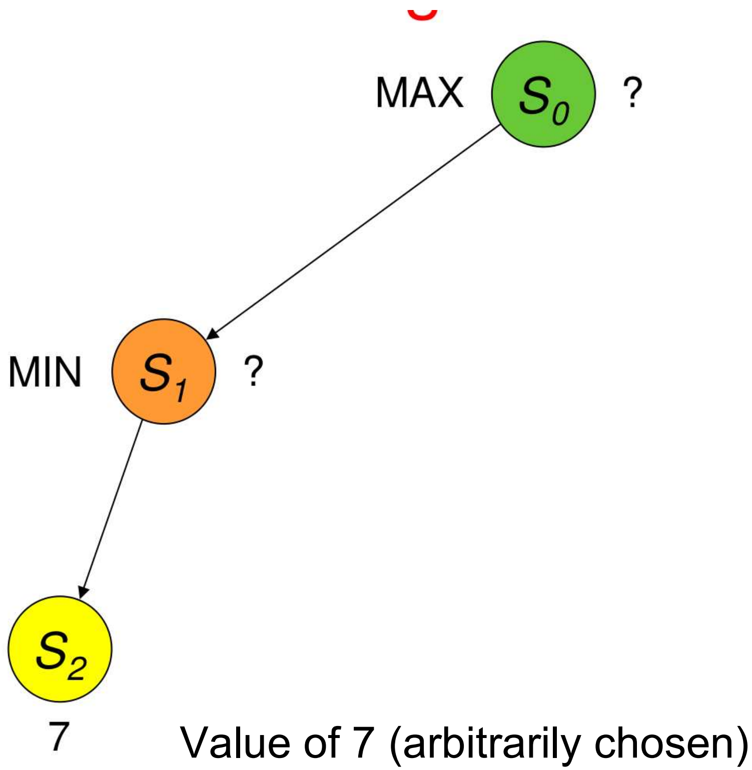
# The Minimax Algorithm: Step-By-Step



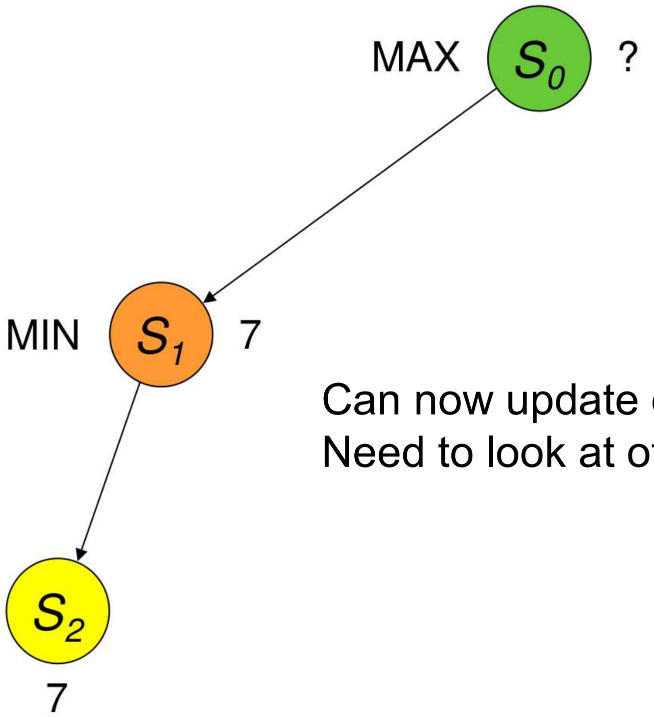

Terminal node located!

Apply max value (its max’s turn!)
Terminal node, so we get a value

# The Minimax Algorithm: Step-By-Step

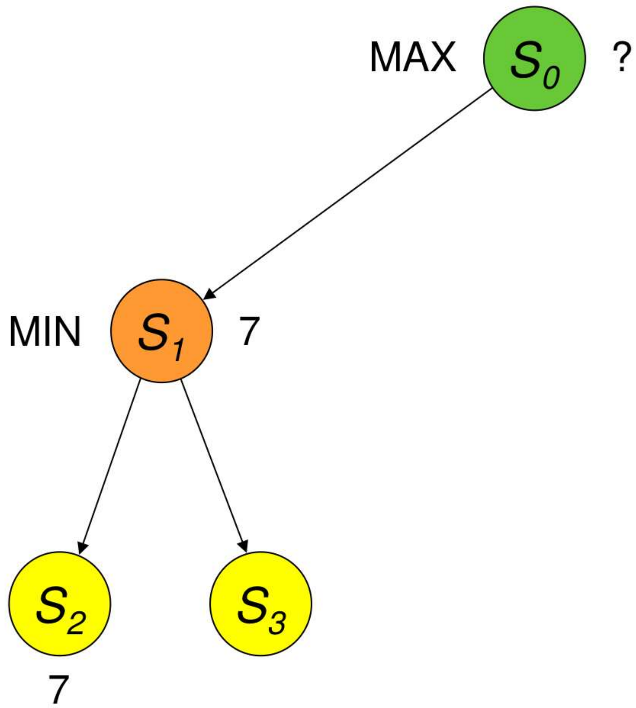


# The Minimax Algorithm: Step-By-Step

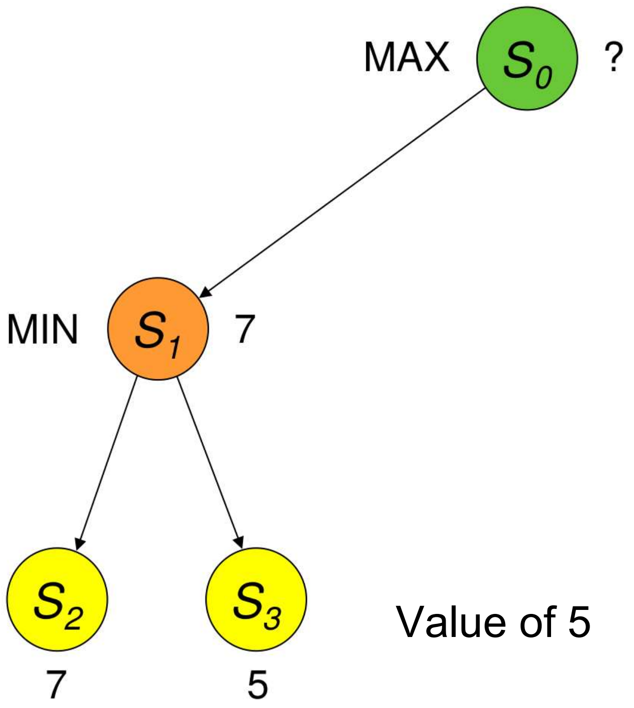



Can now update our min value to 7
Need to look at other successors

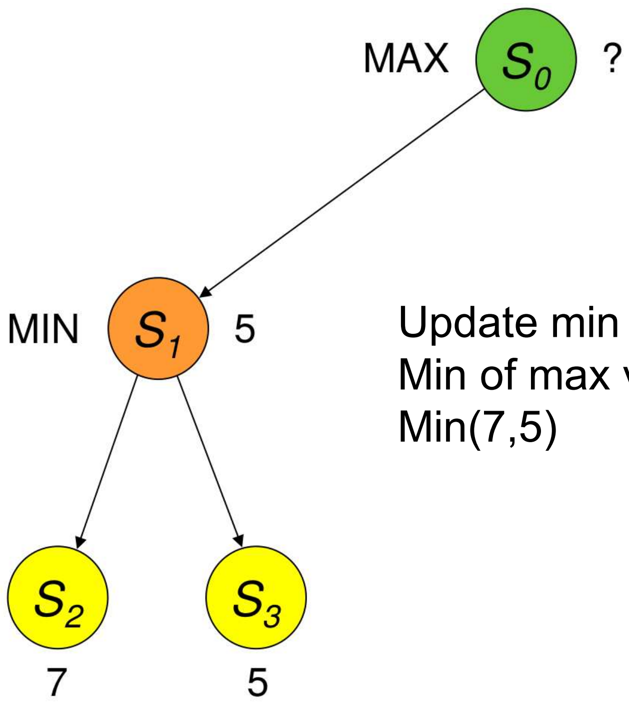
# The Minimax Algorithm: Step-By-Step



# The Minimax Algorithm: Step-By-Step

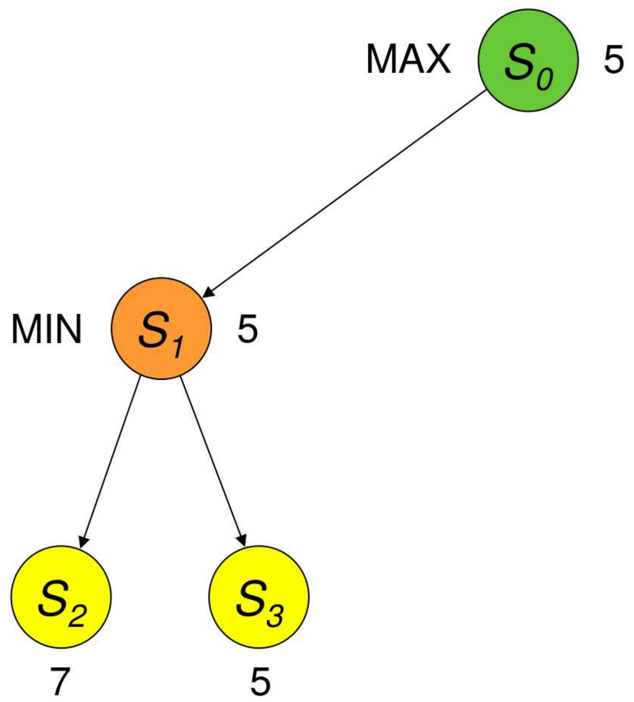


# The Minimax Algorithm: Step-By-Step



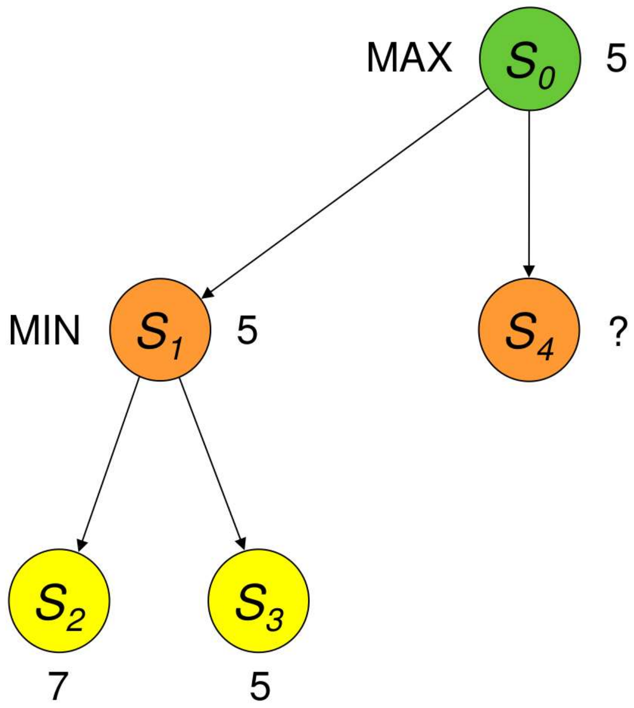

Update min value
Min of max values
Min(7,5)

# The Minimax Algorithm: Step-By-Step

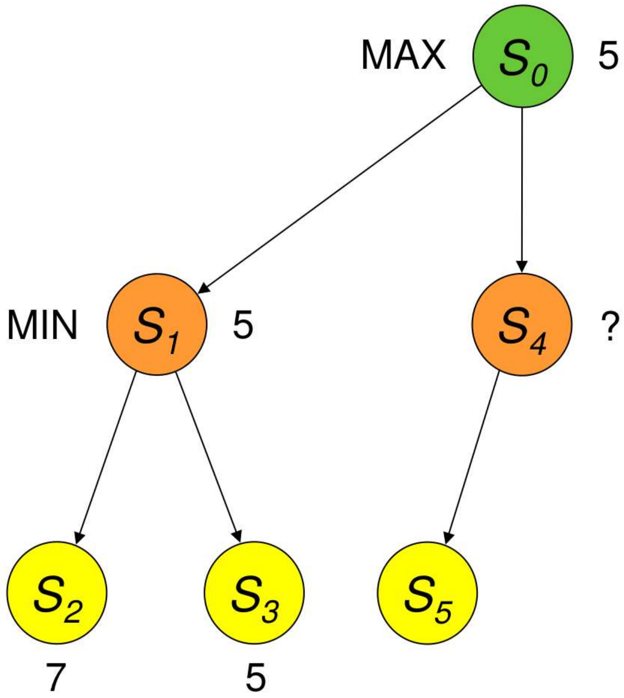



Max Value 5

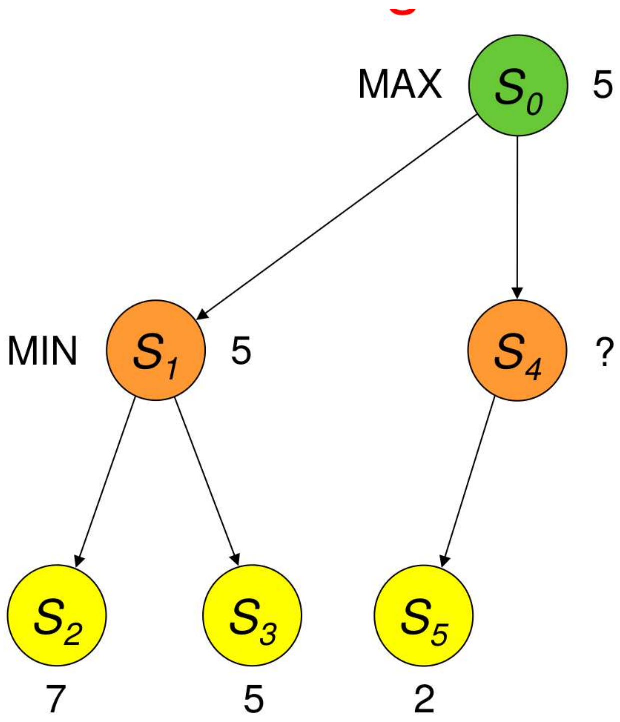
# The Minimax Algorithm: Step-By-Step



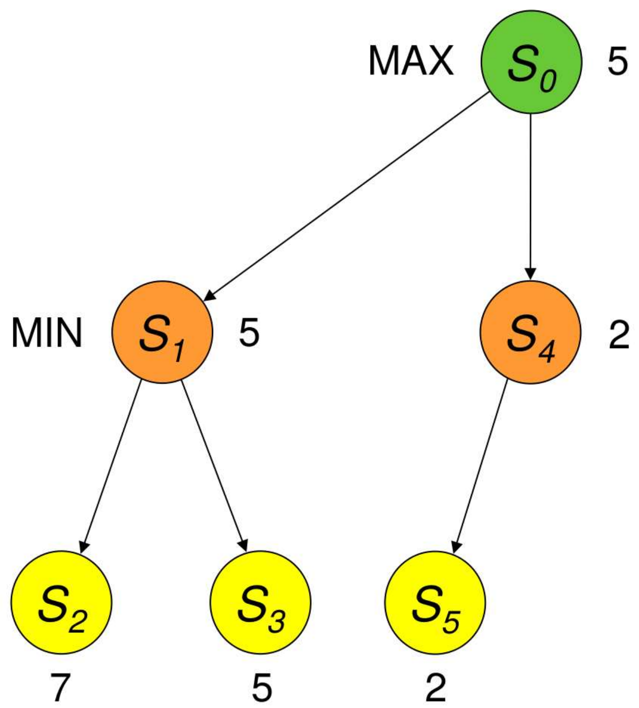
# The Minimax Algorithm: Step-By-Step



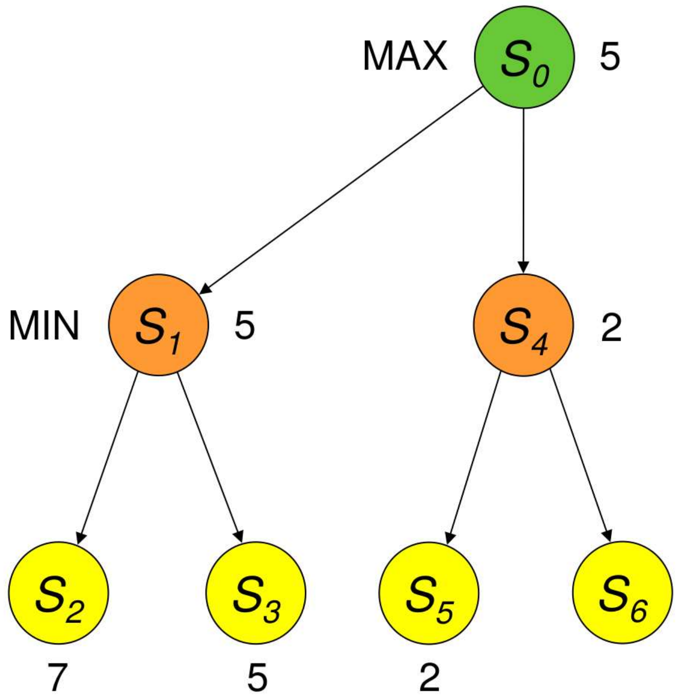
# The Minimax Algorithm: Step-By-Step



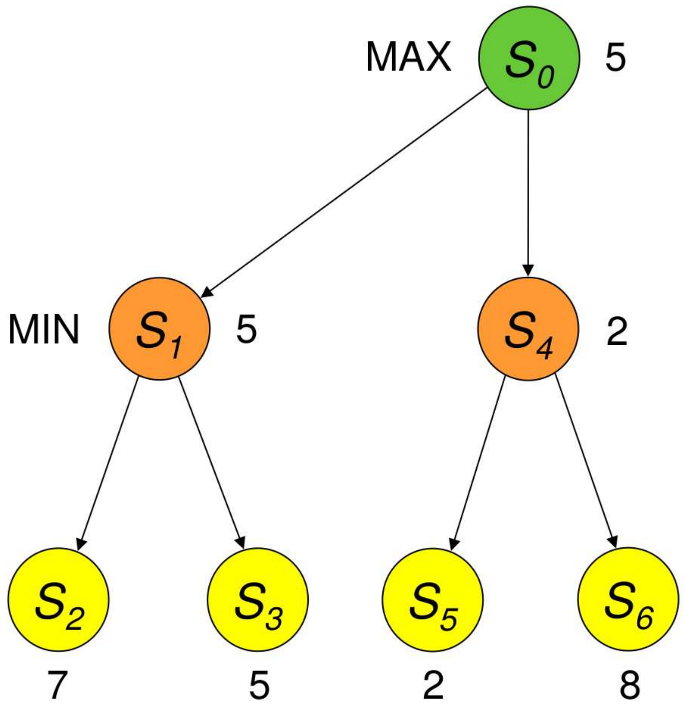
# The Minimax Algorithm: Step-By-Step



# The Minimax Algorithm: Step-By-Step



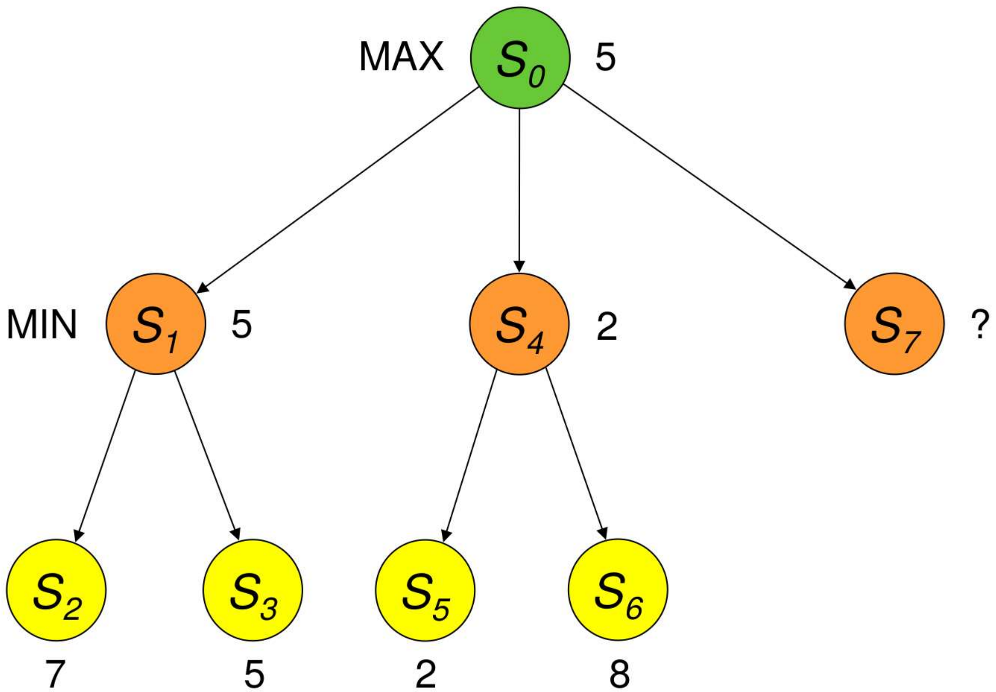
# The Minimax Algorithm: Step-By-Step



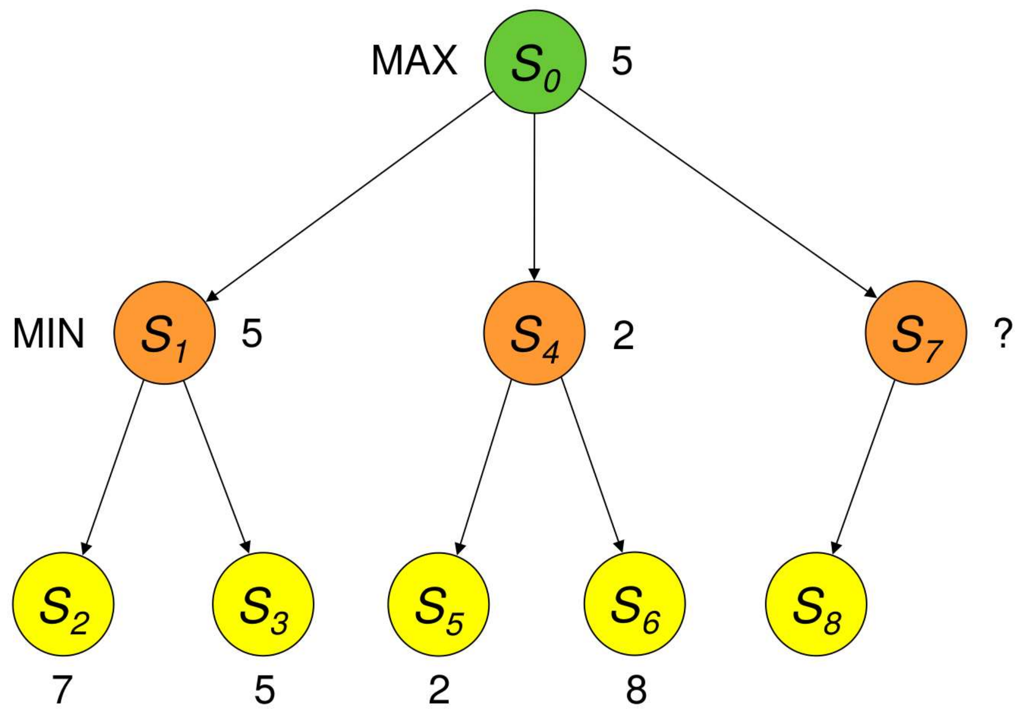

Max stays 5
Max of mins
Max(5,2) = 5

Min is 2 (min of max)
Min(2,8) = 2

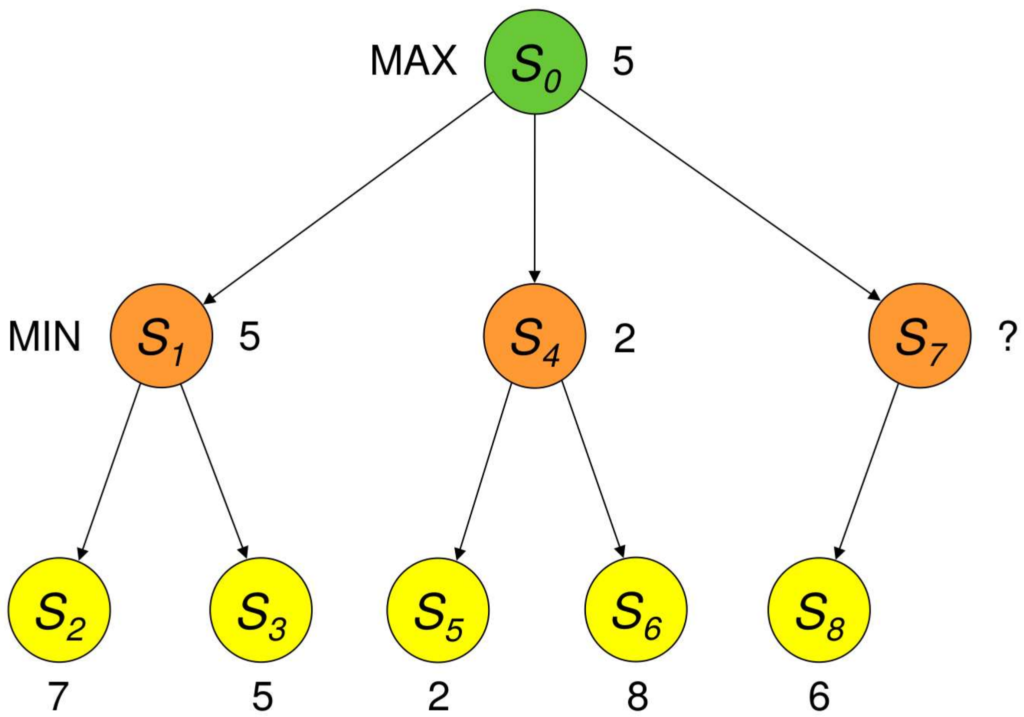
# The Minimax Algorithm: Step-By-Step



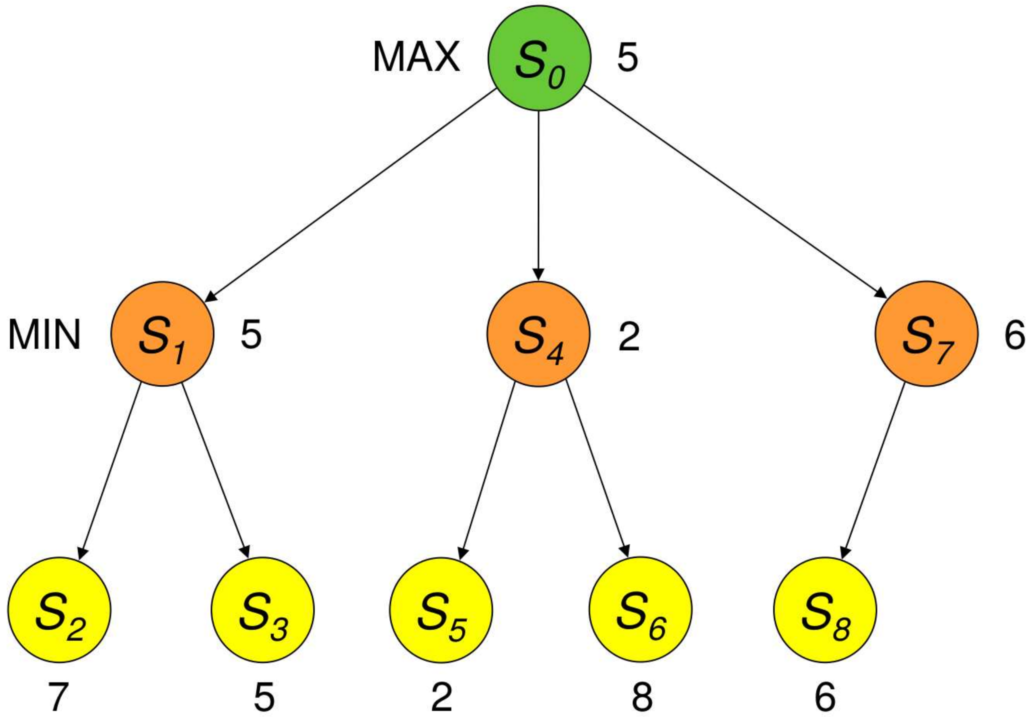
# The Minimax Algorithm: Step-By-Step



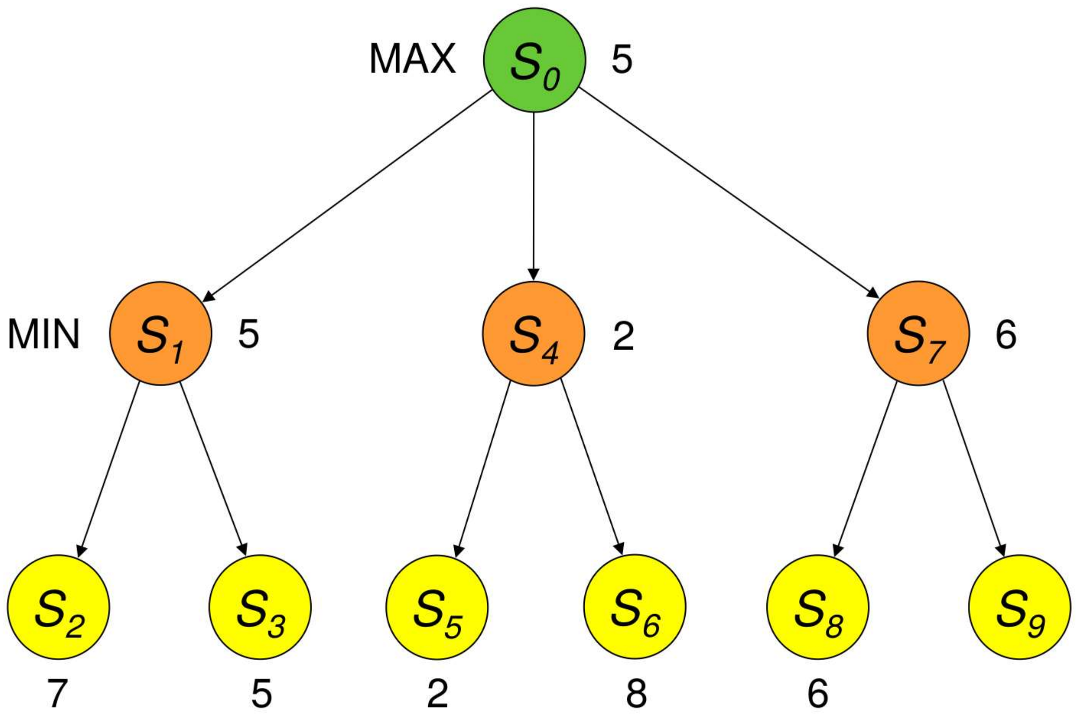
# The Minimax Algorithm: Step-By-Step



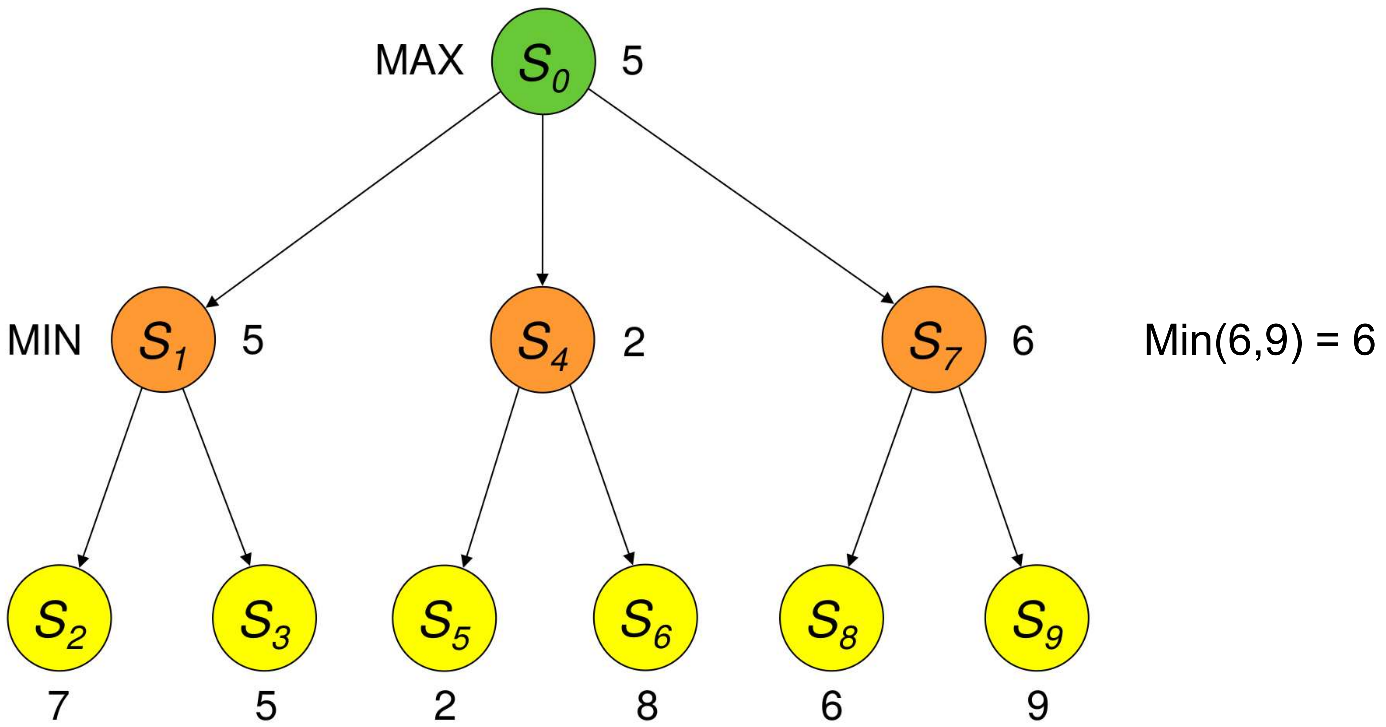
# The Minimax Algorithm: Step-By-Step



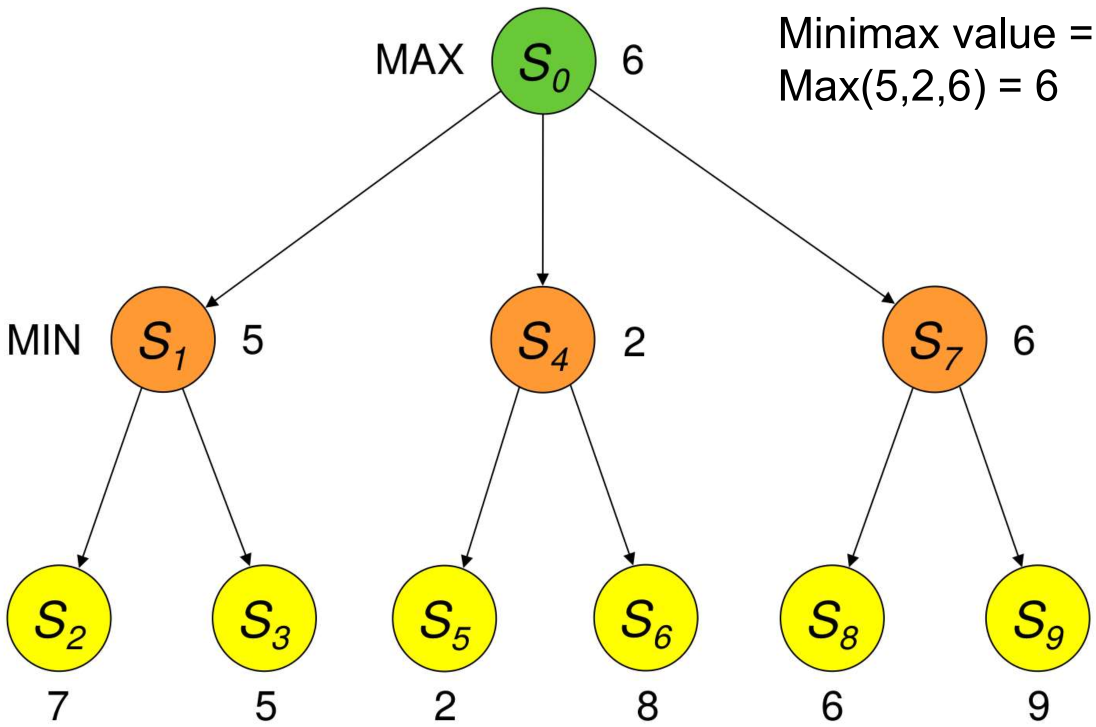
# The Minimax Algorithm: Step-By-Step



# The Minimax Algorithm: Step-By-Step

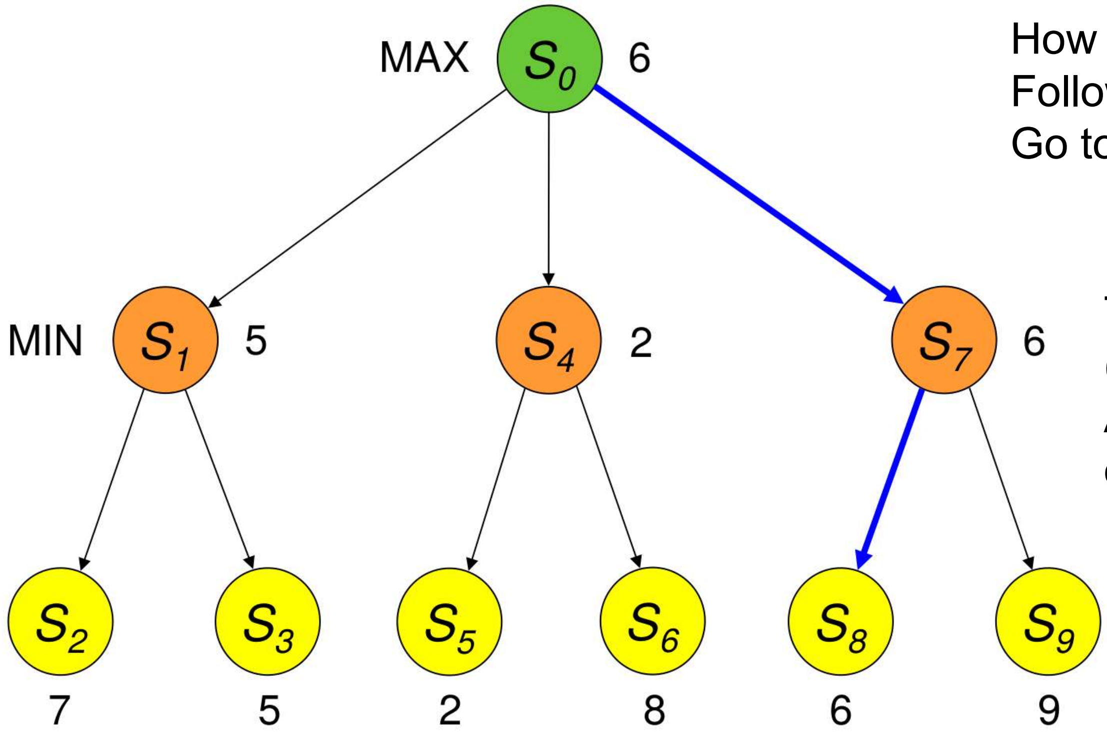


# The Minimax Algorithm: Step-By-Step




Minimax value = 6
Max(5,2,6) = 6

# The Minimax Algorithm: Step-By-Step




How to find optimum path?
Follow all the '6' nodes
Go to successor

This maximises max reward (6 or 9)
Assuming min also plays optimally

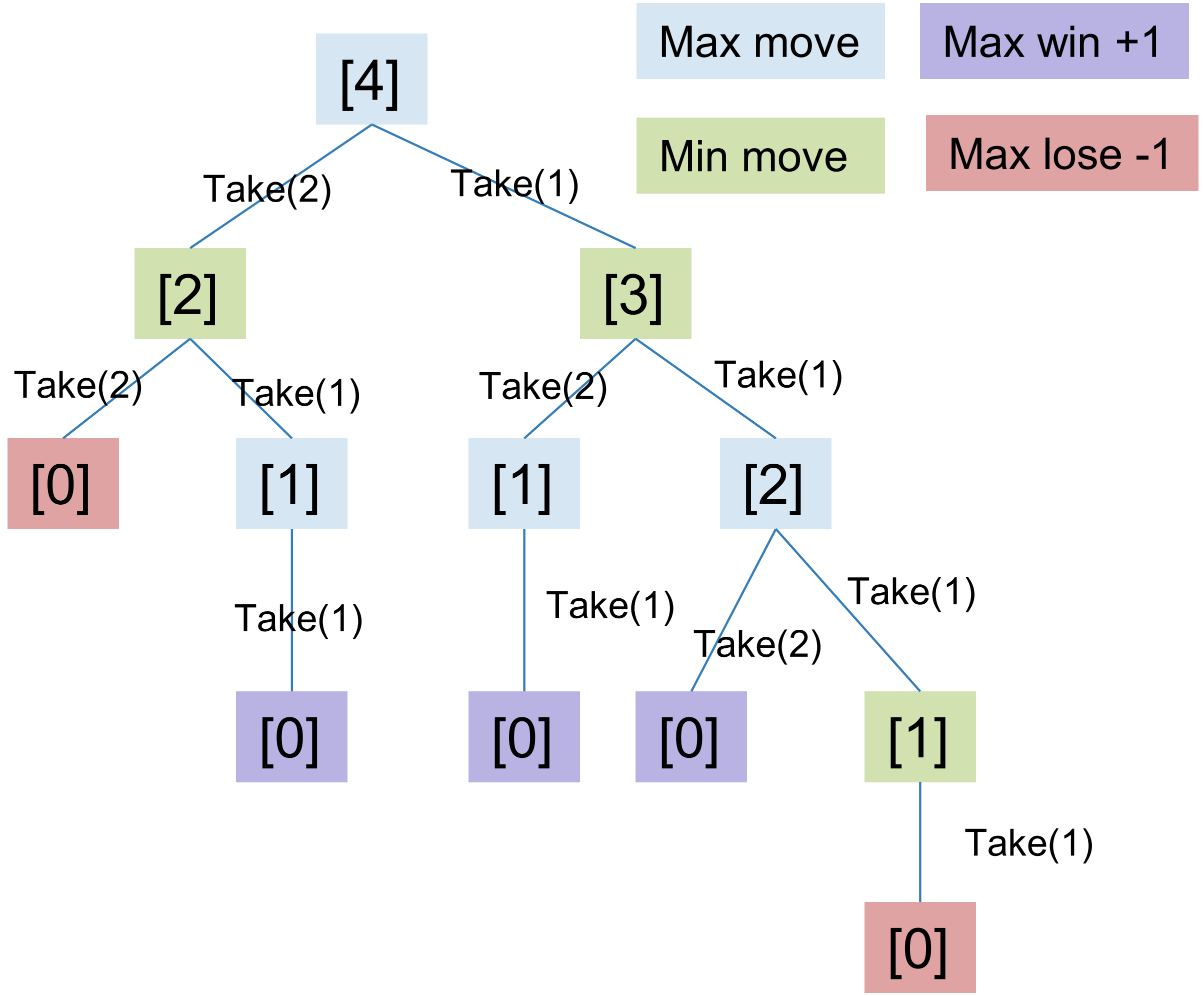
Min can only go to $S_8$ to minimise max's reward

Its not the highest reward, but it is the highest reward assuming that both players play optimally!

# Coin Game

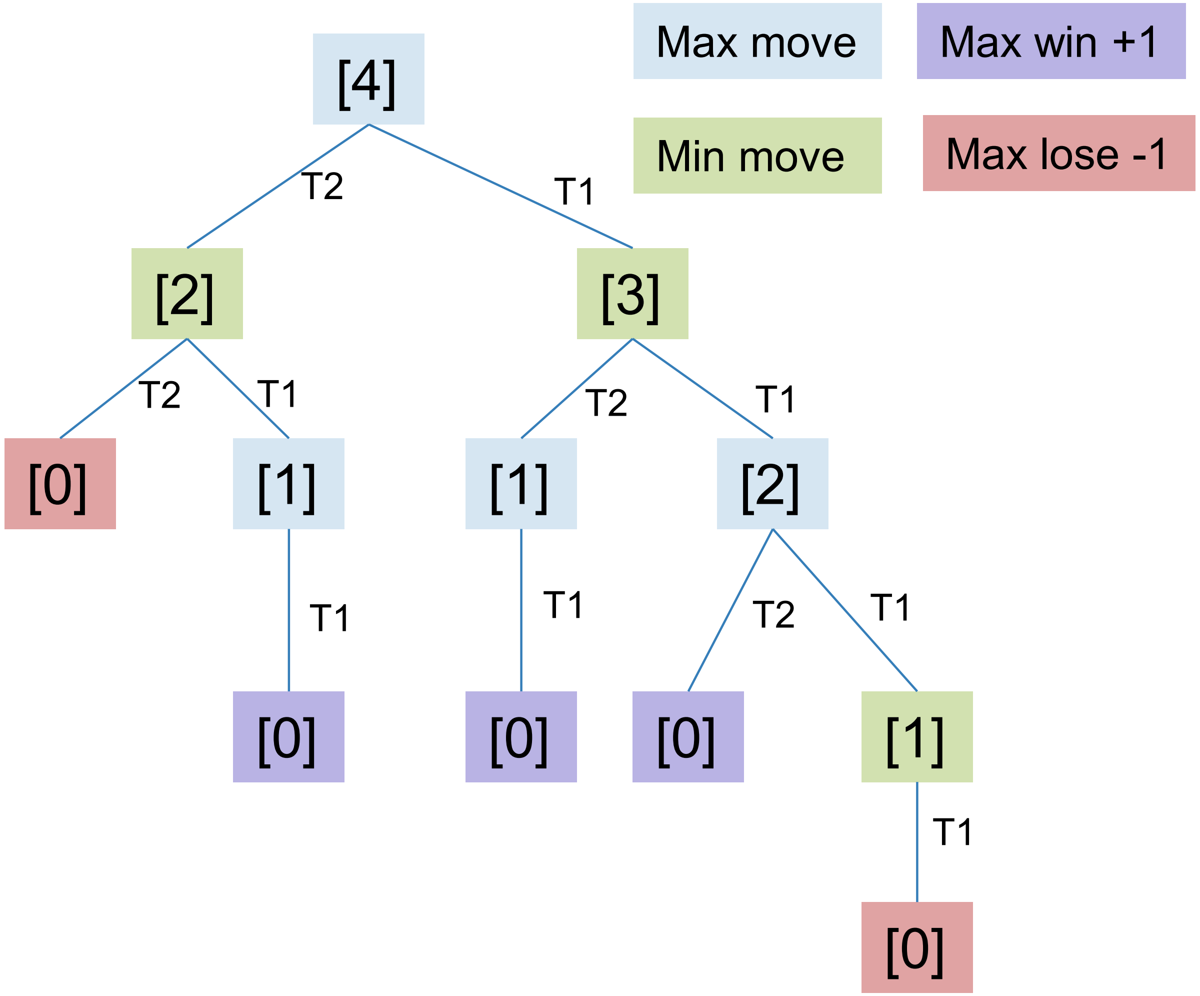
# Our Coin game

- Different layers belong to different agents
- Each player makes an action
- Terminal nodes, [0]
- Utility function (reward)
  - +1 for max win (purple)
  - -1 for min win (red)
- Start with max



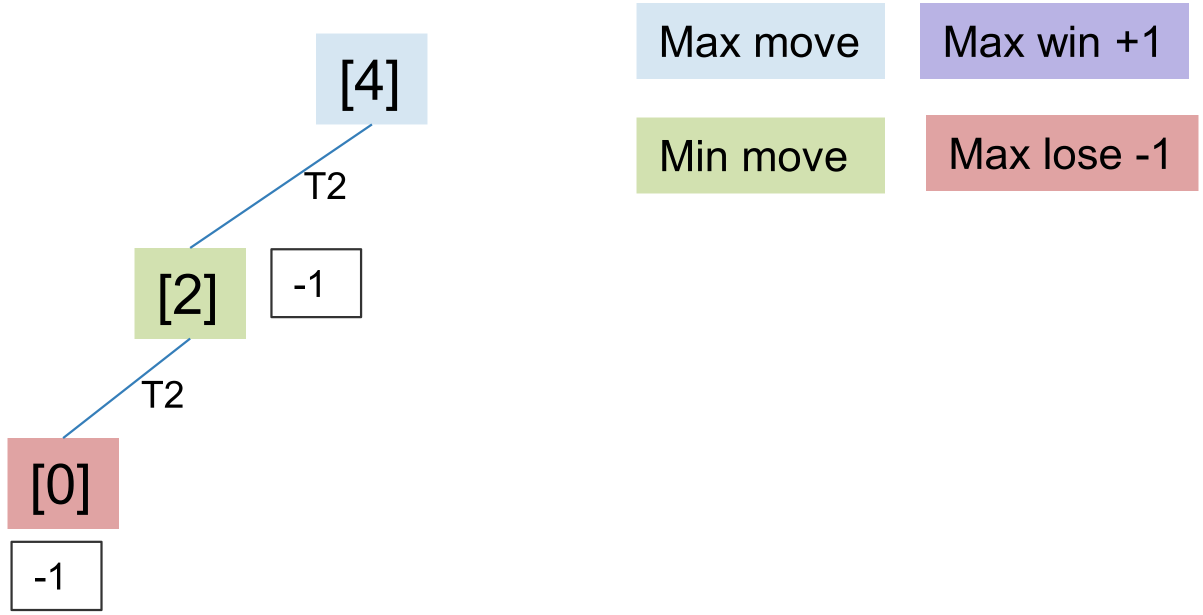
## Our Coin game

- Left path first



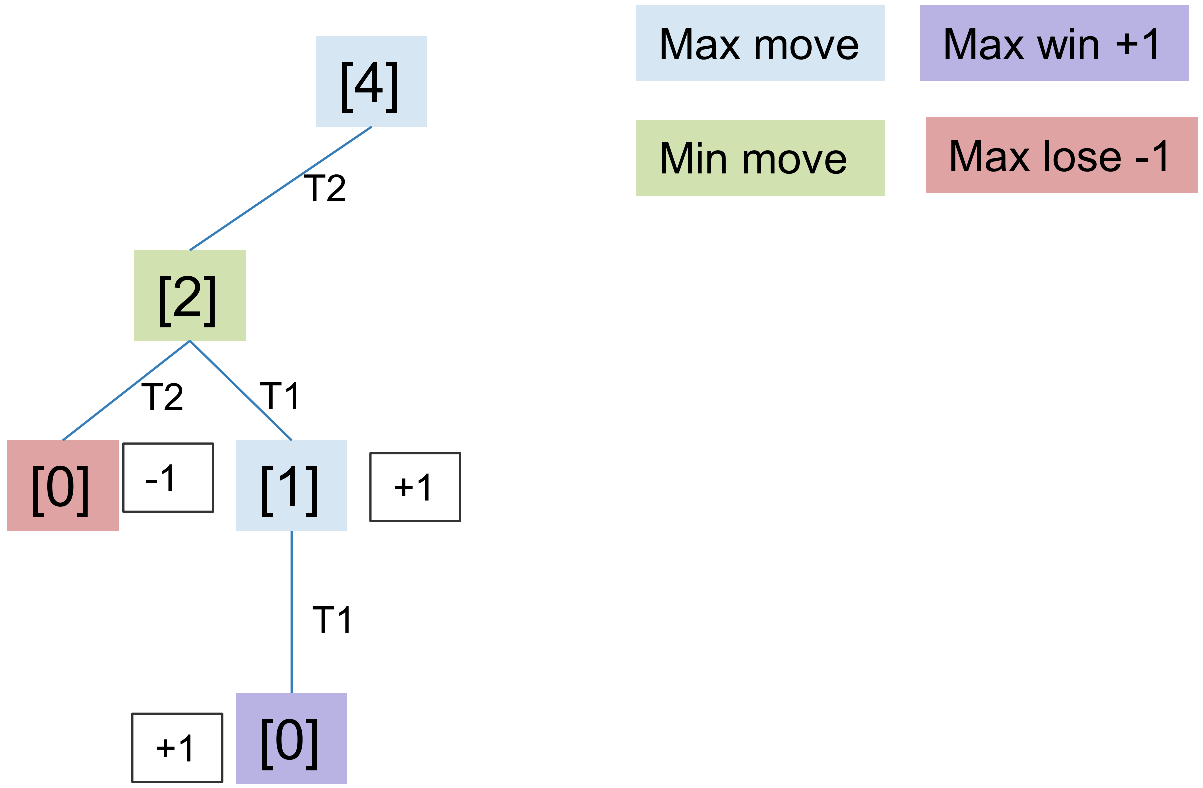
# Our Coin game

- Left path first
- Find terminal state
- First state, so set val to -1



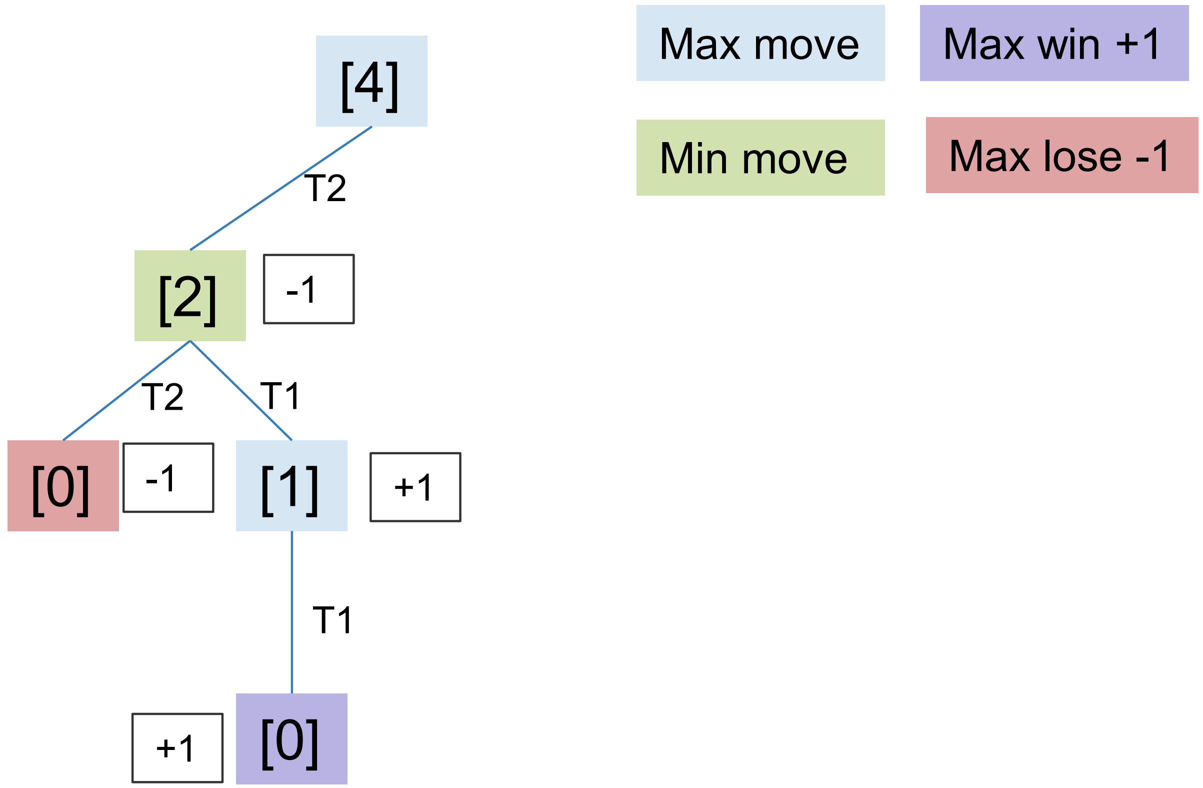
# Our Coin game

- Other successor
- Finds +1
- Sets val to +1



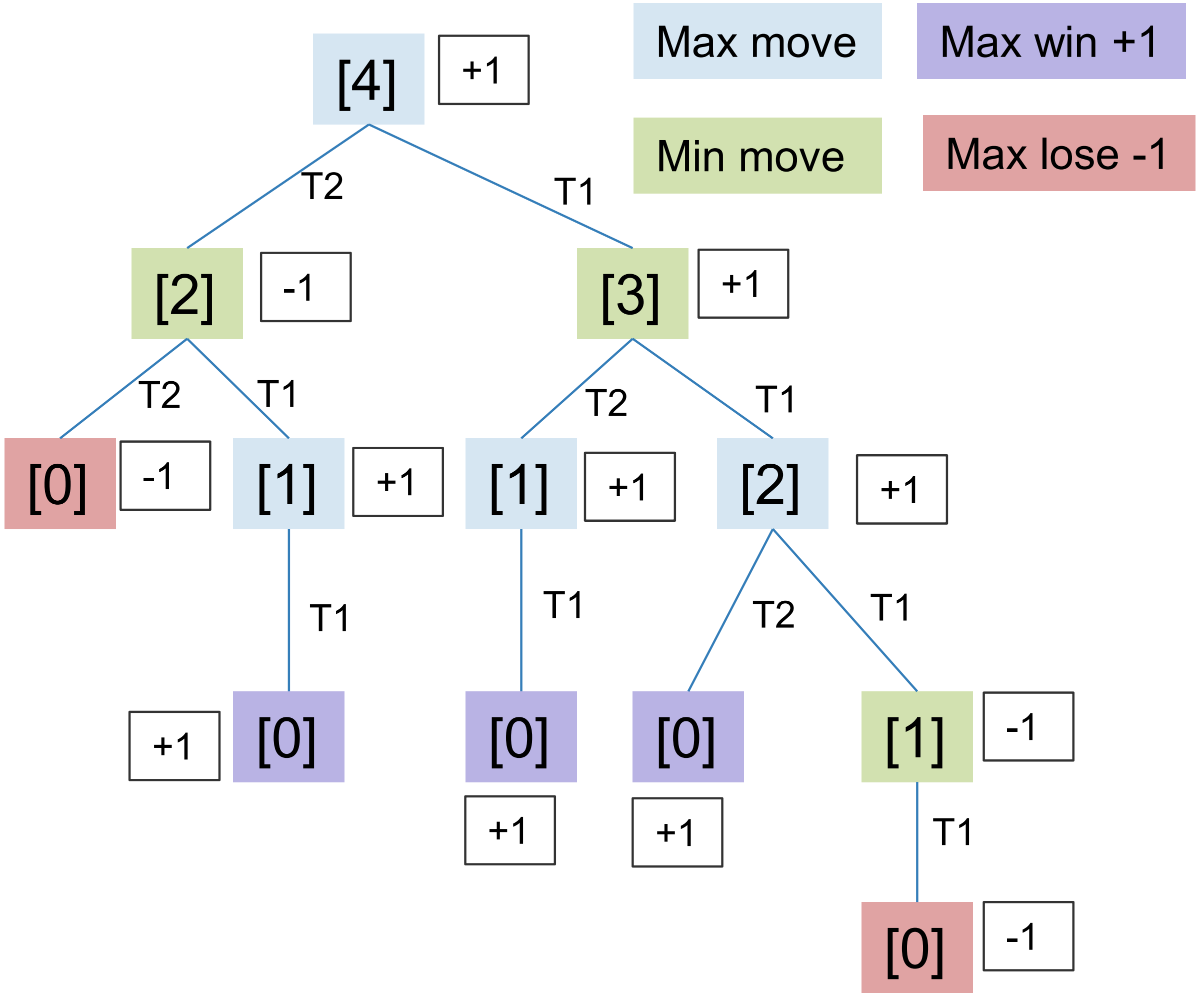
# Our Coin game

- As its mins move, we look to find the minimal value
- Min of -1 and +1



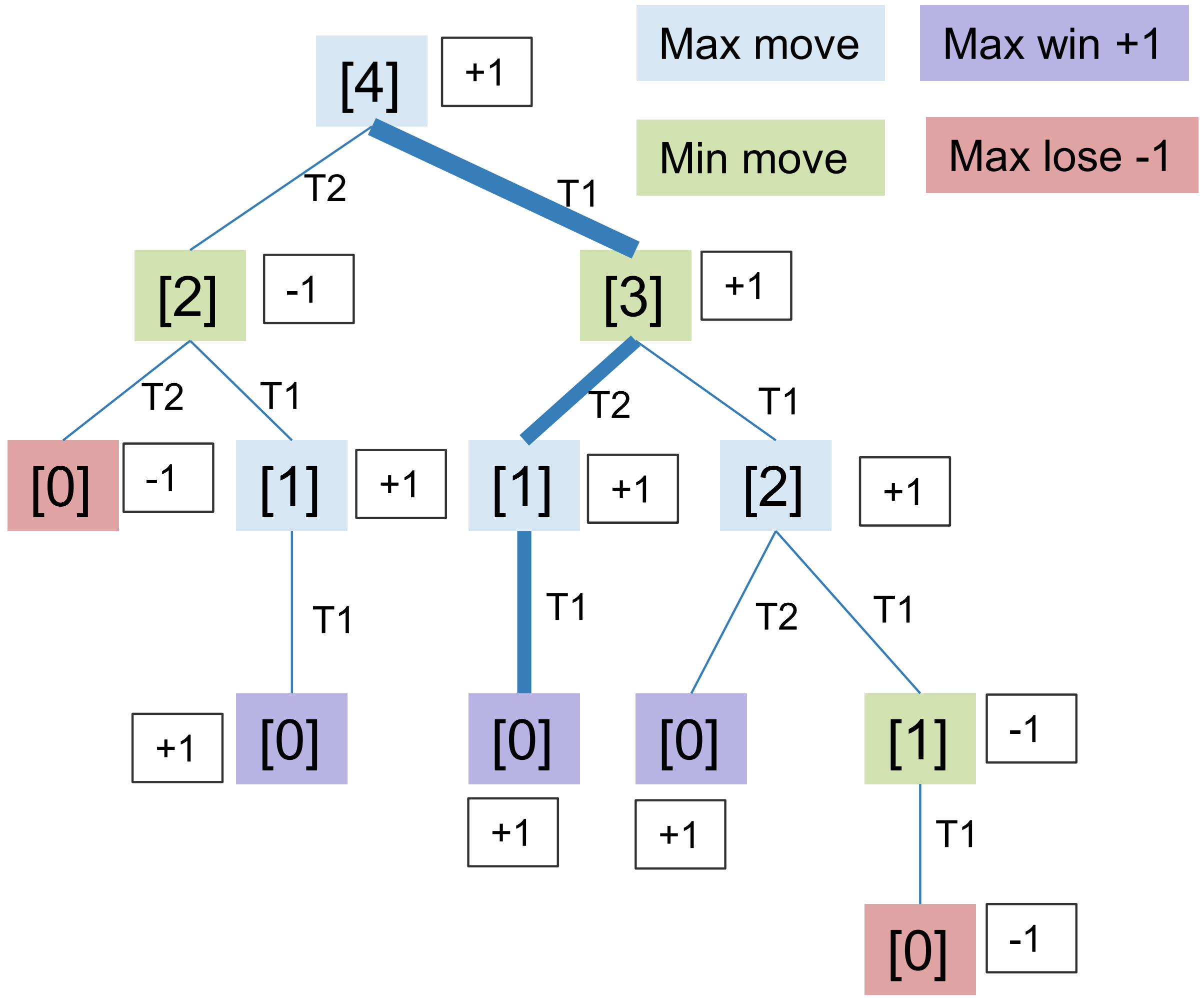
# Our Coin game

- Repeat same process
- Find terminal states and work upwards



# Our Coin game

- Advice:
- Always take 1 coin!
    - Path in blue
- May be more than one optimal path
- Can use metrics to decide (such as first one found etc.)



## Summary

- 2 player perfect information game
- Game trees rather than search trees
- Minimaxing
  - However, this is not efficient, as all states need to be explored
  - What if there are time constraints?
    - i.e. speed chess?