Games AI

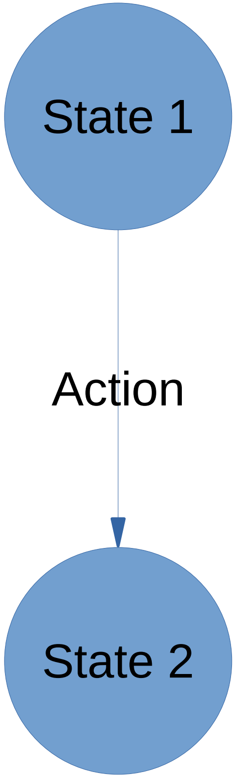
Lecture 7.1

Whole Game Search

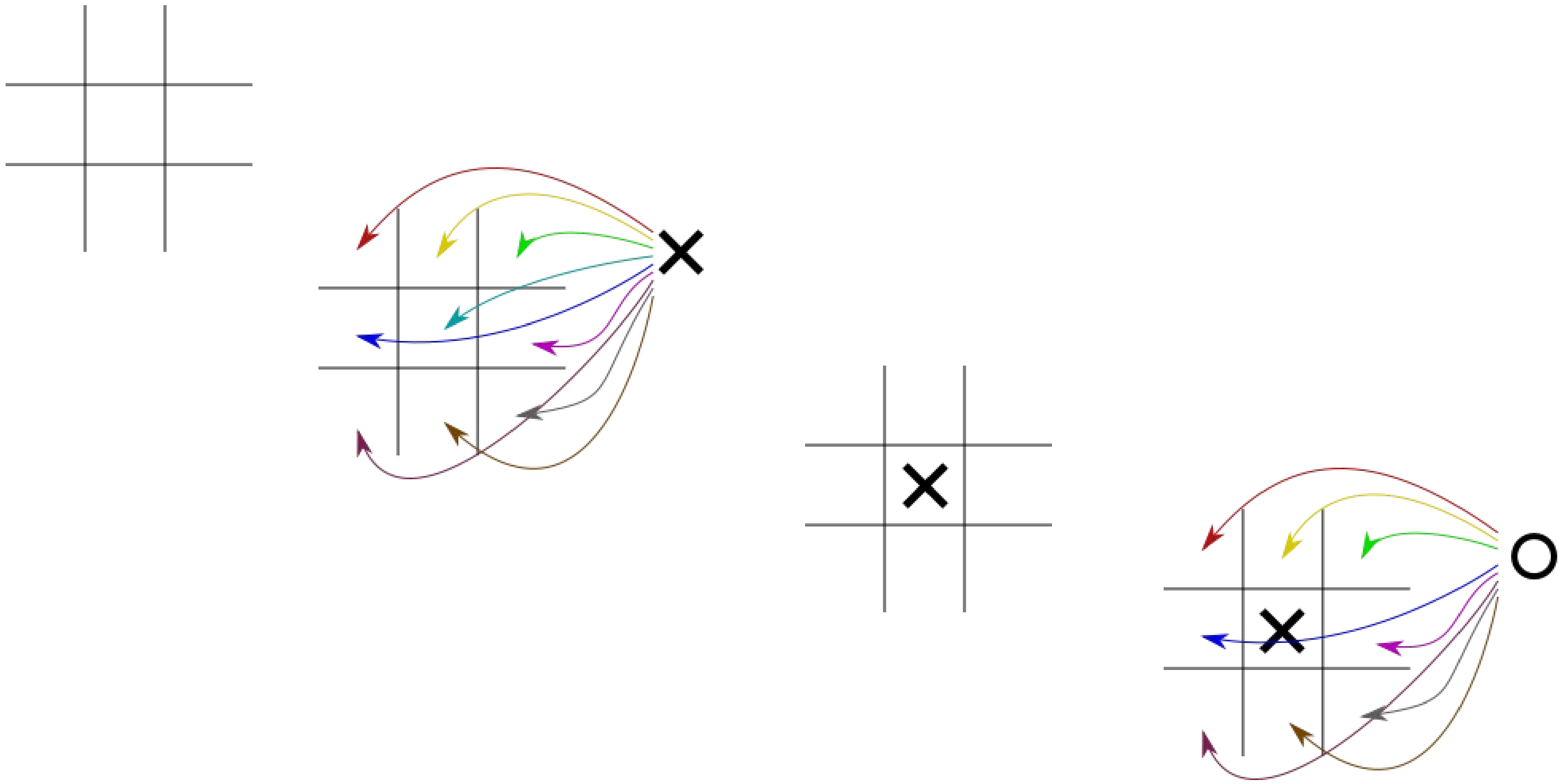
- Whole-Game Search
  - Solving Problems with Serach
  - Tree Search
  - Adversarial Search

- AI Player
  - Turn based games
  - Playing to win

- Turn-based Games
  - Game states
  - Actions



Introduction

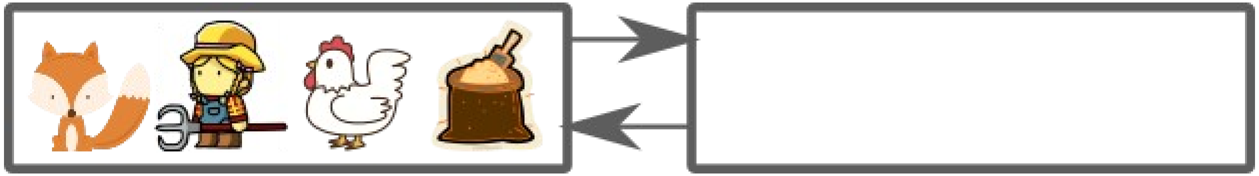


Solving Problems With Search

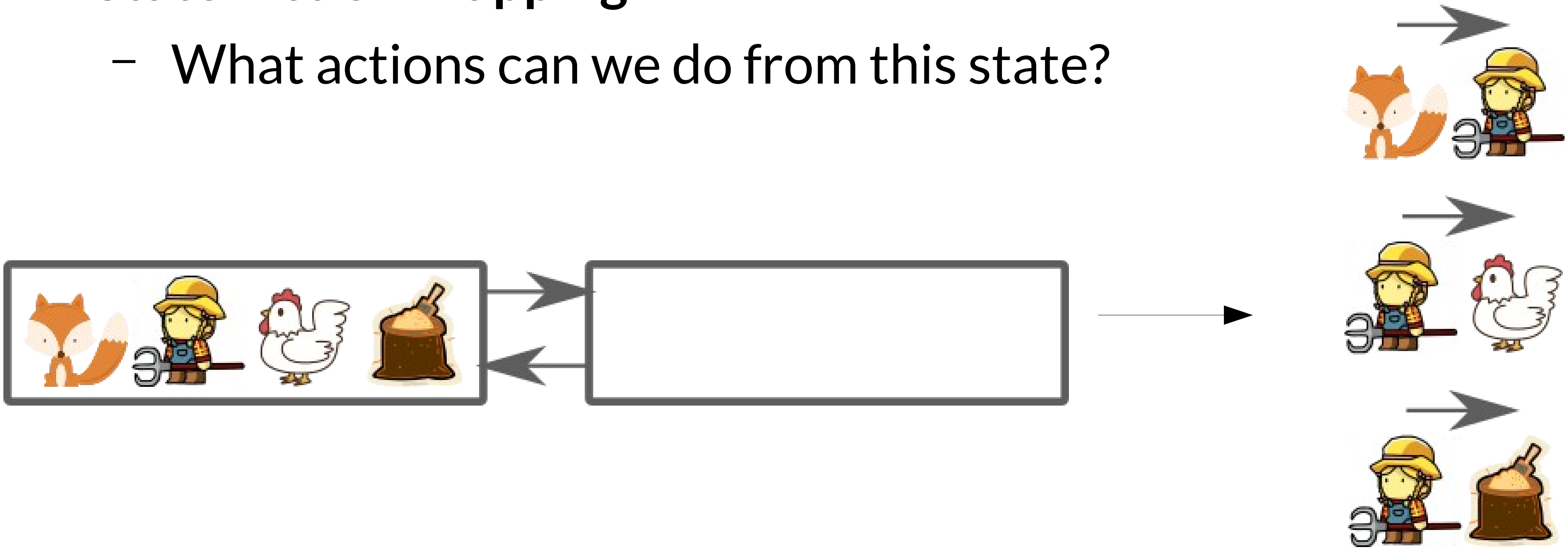
- Well-defined problems have five components
  - Initial state
  - State-action mapping
  - Transition model
  - Goal test
  - Path cost function

- 1 player game
  - A farmer has a **fox**, a **chicken**, and a **sack of corn**
  - He needs to get them **all across a river**
  - If he leaves the sack of corn with the chicken, **it eats it**
  - If he leaves the fox with the chicken, **it eats it**
  - Only **one thing can fit in his boat**

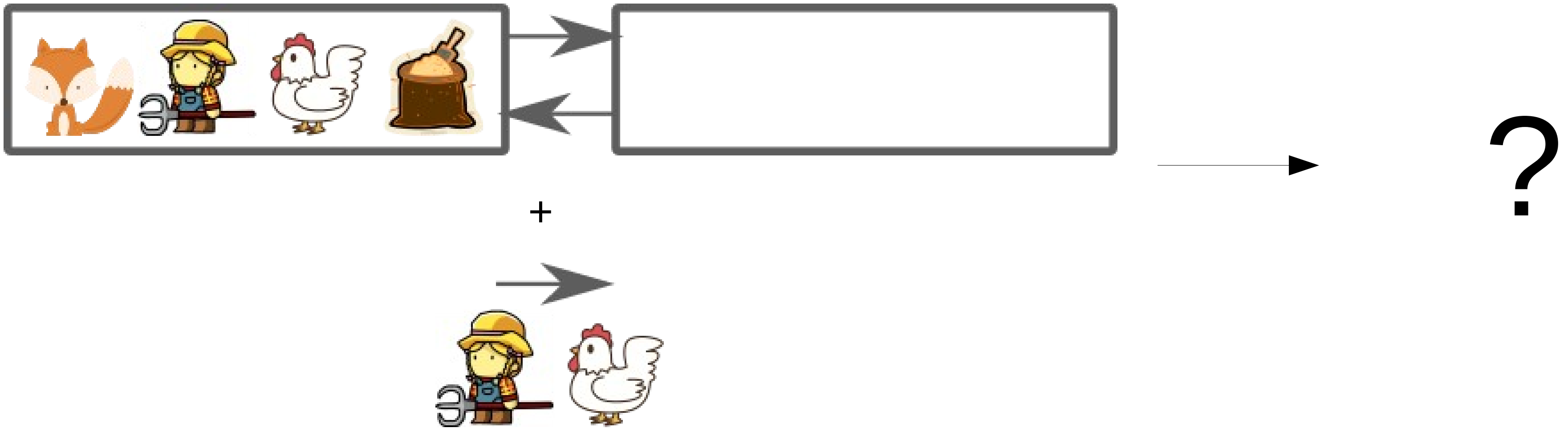
- **Initial State**
  - LeftBank: { Farmer, Chicken, Corn, Fox }
  - RightBank: { }



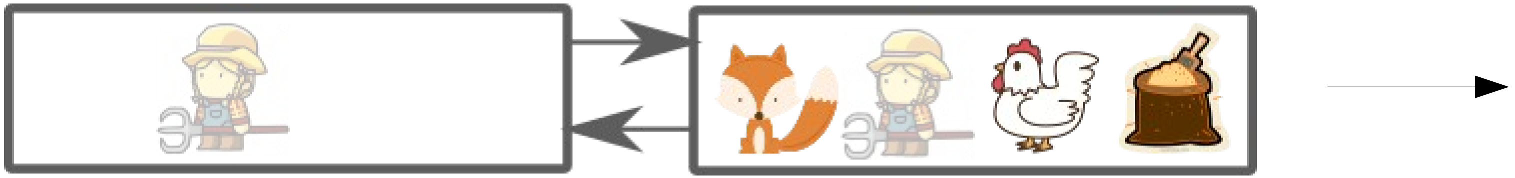
- **State-Action Mapping**
  - What actions can we do from this state?



- **Transition Model**
  - What state do we get if we apply action X in state Y?

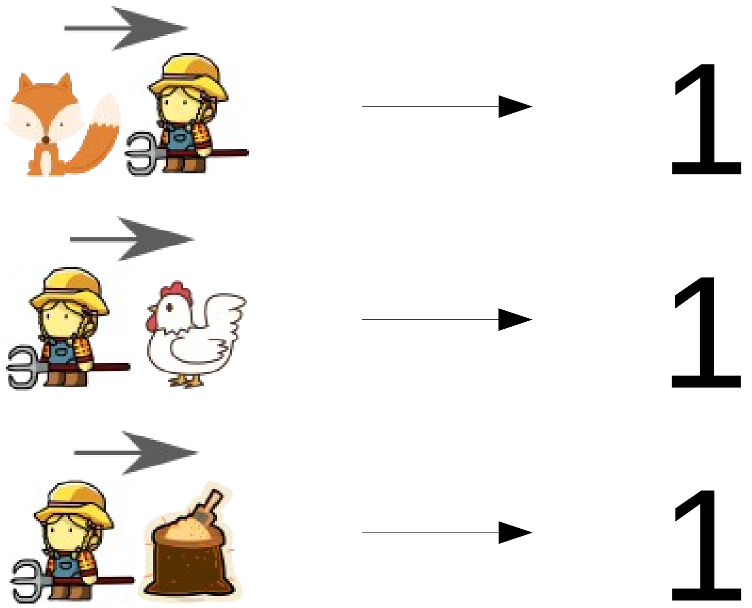


- **Goal Test**
  - Have we won?

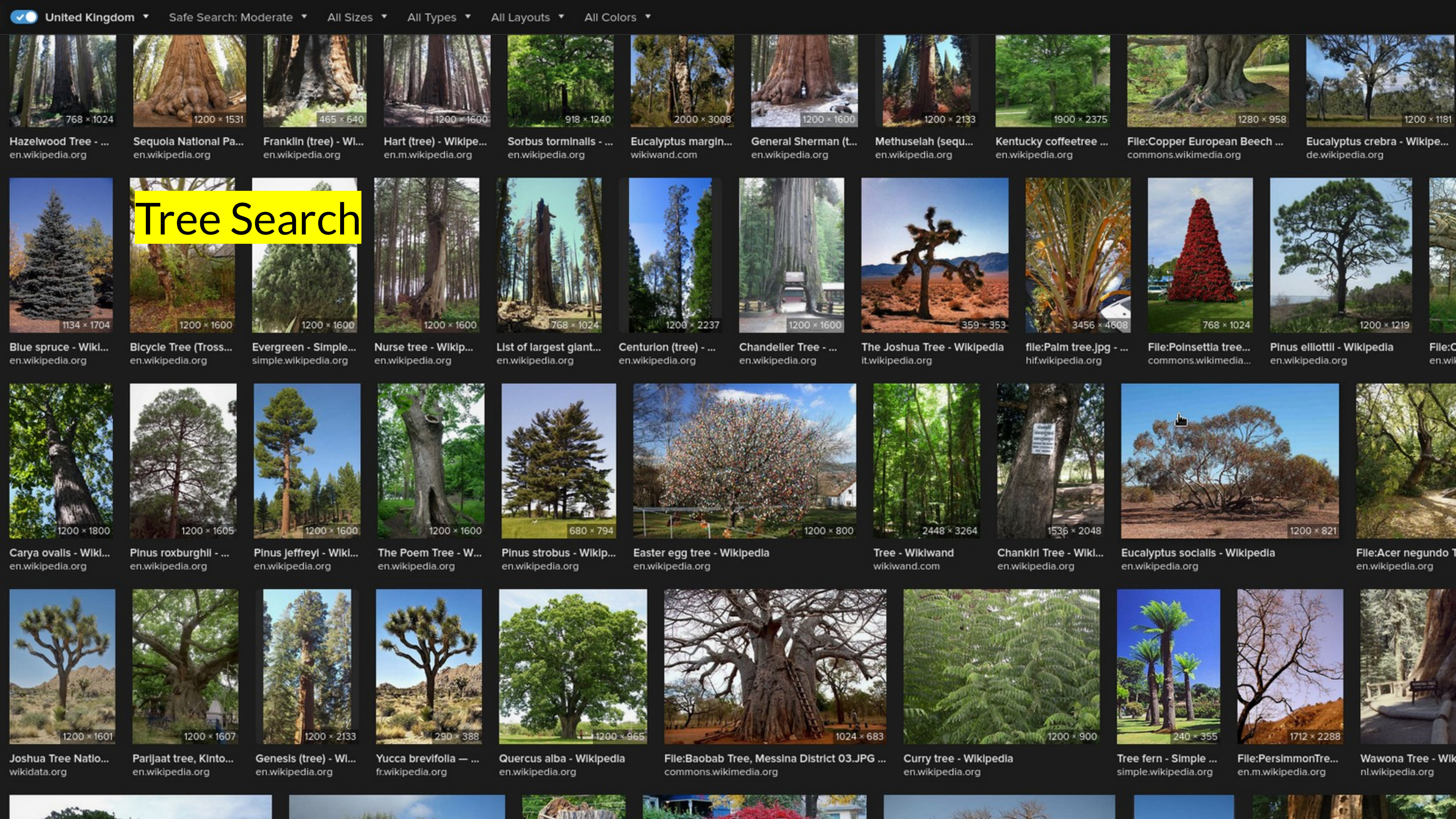


✓

- **Step Cost Function**
  - How expensive is a particular move?



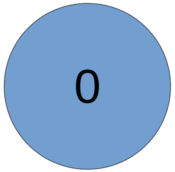
- For many **games**, we can define these five components
  - Initial state
  - State-action mapping
  - Transition model
  - Goal test
  - Path cost function
- Combined, these let us build a **tree of game states**
  - Then we **search** this tree



United Kingdom
Safe Search: Moderate
All Sizes
All Types
All Layouts
All Colors
Tree Search
768 × 1024
Hazelwood Tree - ...
en.wikipedia.org
1200 × 1531
Sequoia National Pa...
en.wikipedia.org
465 × 640
Franklin (tree) - Wi...
en.wikipedia.org
1200 × 1600
Hart (tree) - Wikipe...
en.m.wikipedia.org
918 × 1240
Sorbus torminalis - ...
en.wikipedia.org
2000 × 3008
Eucalyptus margin...
wikiwand.com
1200 × 1600
General Sherman (t...
en.wikipedia.org
1200 × 2133
Methuselah (sequ...
en.wikipedia.org
1900 × 2375
Kentucky coffeetree ...
en.wikipedia.org
1280 × 958
File:Copper European Beech ...
commons.wikimedia.org
1200 × 1181
Eucalyptus crebra - Wikipe...
de.wikipedia.org
1134 × 1704
Blue spruce - Wiki...
en.wikipedia.org
1200 × 1600
Bicycle Tree (Tross...
en.wikipedia.org
1200 × 1600
Evergreen - Simple...
simple.wikipedia.org
1200 × 1600
Nurse tree - Wikip...
en.wikipedia.org
768 × 1024
List of largest giant...
en.wikipedia.org
1200 × 2237
Centurion (tree) - ...
en.wikipedia.org
1200 × 1600
Chandelier Tree - ...
en.wikipedia.org
359 × 353
The Joshua Tree - Wikipedia
it.wikipedia.org
3456 × 4608
file:Palm tree.jpg - ...
hif.wikipedia.org
768 × 1024
File:Poinsettia tree...
commons.wikimedia...
1200 × 1219
Pinus elliottii - Wikipedia
en.wikipedia.org
1200 × 1800
Carya ovalis - Wiki...
en.wikipedia.org
1200 × 1605
Pinus roxburghii - ...
en.wikipedia.org
1200 × 1600
Pinus jeffreyi - Wiki...
en.wikipedia.org
1200 × 1600
The Poem Tree - W...
en.wikipedia.org
680 × 794
Pinus strobus - Wikip...
en.wikipedia.org
1200 × 800
Easter egg tree - Wikipedia
en.wikipedia.org
2448 × 3264
Tree - Wikiwand
wikiwand.com
1536 × 2048
Chankiri Tree - Wiki...
en.wikipedia.org
1200 × 821
Eucalyptus socialis - Wikipedia
en.wikipedia.org
File:Acer negundo
en.wikipedia.org
1200 × 1601
Joshua Tree Natio...
wikidata.org
1200 × 1607
Parijaat tree, Kinto...
en.wikipedia.org
1200 × 2133
Genesis (tree) - Wi...
en.wikipedia.org
290 × 388
Yucca brevifolia — ...
fr.wikipedia.org
1200 × 965
Quercus alba - Wikipedia
en.wikipedia.org
1024 × 683
File:Baobab Tree, Messina District 03.JPG ...
commons.wikimedia.org
1200 × 900
Curry tree - Wikipedia
en.wikipedia.org
240 × 355
Tree fern - Simple ...
simple.wikipedia.org
1712 × 2288
File:PersimmonTre...
en.m.wikipedia.org
Wawona Tree - Wik
nl.wikipedia.org

- We can construct a tree of Game States
  - Root of tree is **Initial State**
  - Each child is the result of **applying one action** (transition model)
  - **Goal states** are leaf nodes
- Let's remind ourselves of a basic tree search algorithm

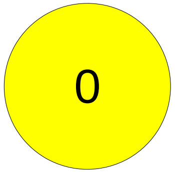
Tree Search



Queue
0

- Breadth-first search
- Set up:
  1. Create queue
  2. Add start to queue
  3. Mark as discovered

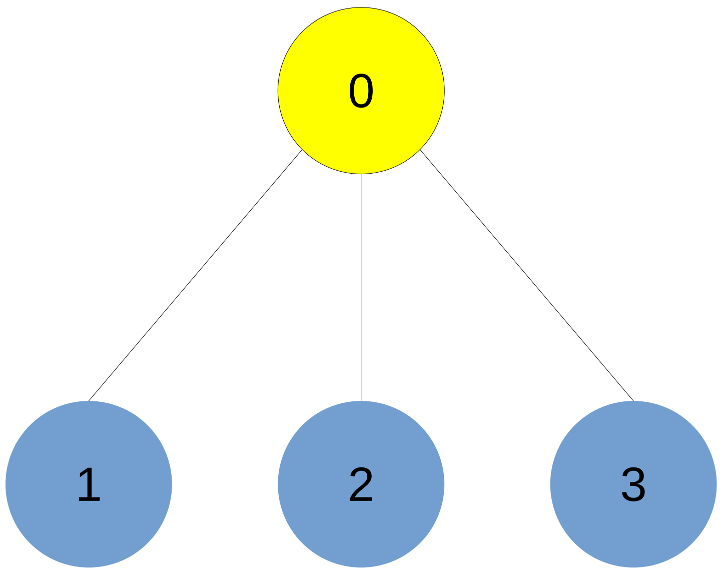
BFS



Queue

- While queue is not empty
  1. Select top of queue
  2. Check if goal

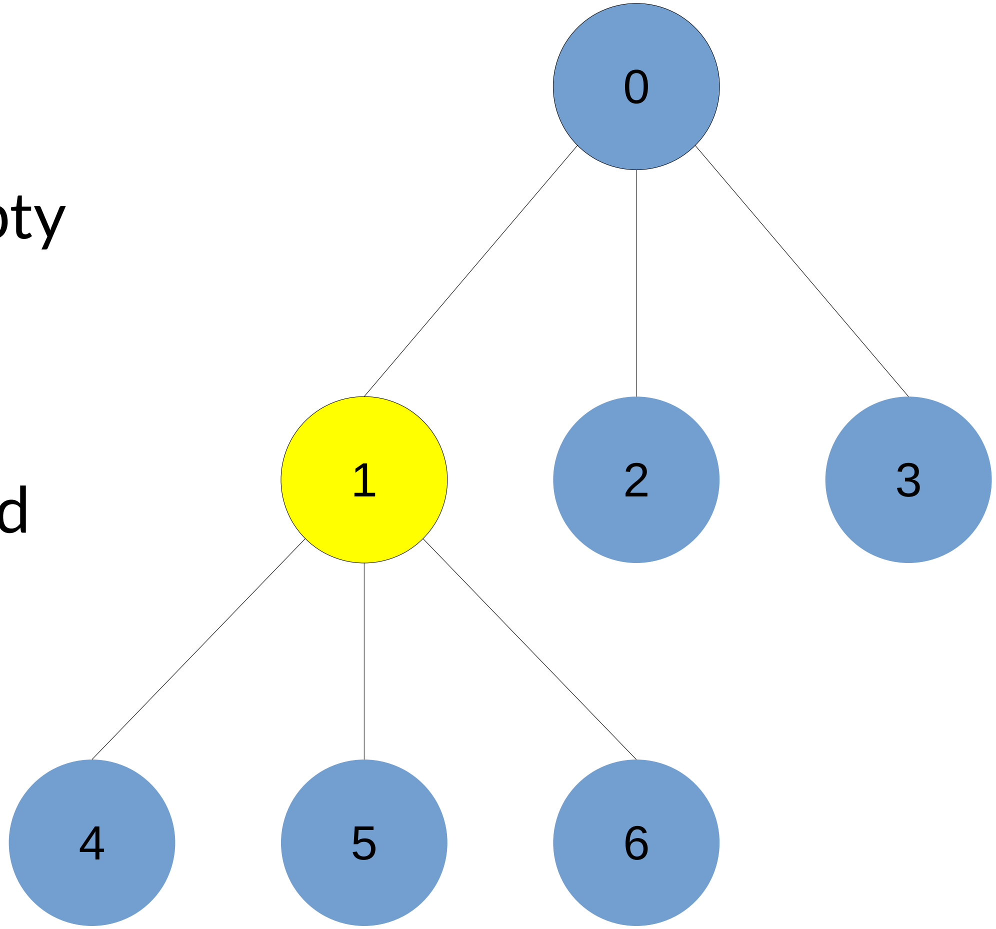
- While queue is not empty
  1. Select top of queue
  2. Check if goal
  3. Add all undiscovered children to queue
  4. Mark each child as discovered



Queue
1
2
3

BFS

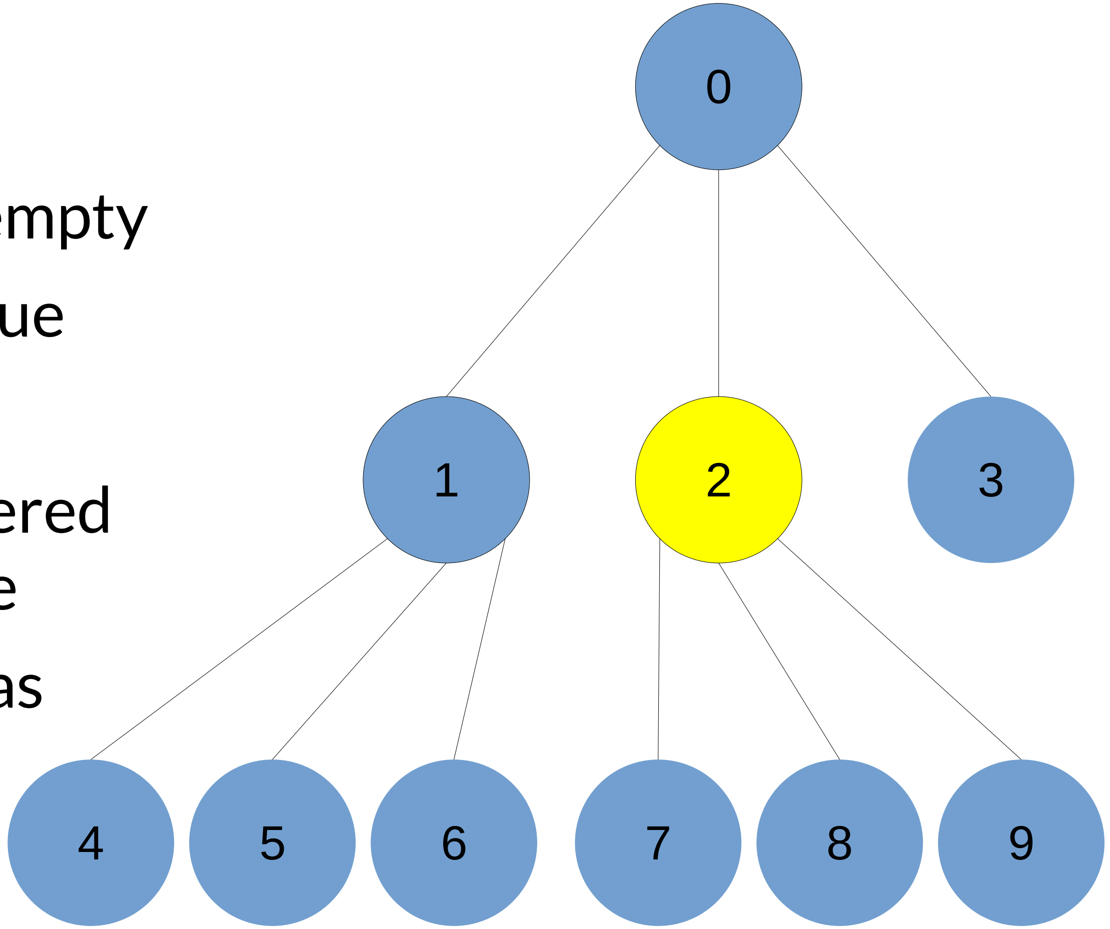
- While queue is not empty
  1. Select top of queue
  2. Check if goal
  3. Add all undiscovered children to queue
  4. Mark each child as discovered



Queue
2
3
4
5
6

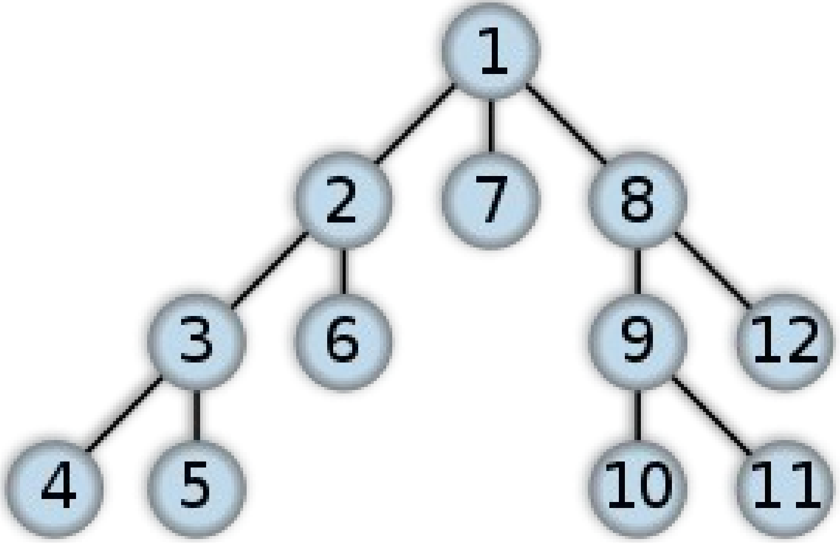
BFS

- While queue is not empty
  1. Select top of queue
  2. Check if goal
  3. Add all undiscovered children to queue
  4. Mark each child as discovered



Queue
3
4
5
6
7
8
9

- Depth-first Search
  - Start at the root node
  - Pick the first child
  - Keep exploring down the tree until terminal node is reached.
  - Then backtrack a step and take next branch...



- Depth-first Search Pseudocode

```
procedure DFS_iterative(G, v) is
    let S be a stack
    S.push(v)
    while S is not empty do
        v = S.pop()
        if v is not labeled as discovered then
            label v as discovered
            for all edges from v to w in G.adjacentEdges(v) do
                S.push(w)
```

- Depth-first Search
  - Path found may not be shortest
  - Might find a path quickly (depending on tree)
  - Might not terminate (if infinite tree)

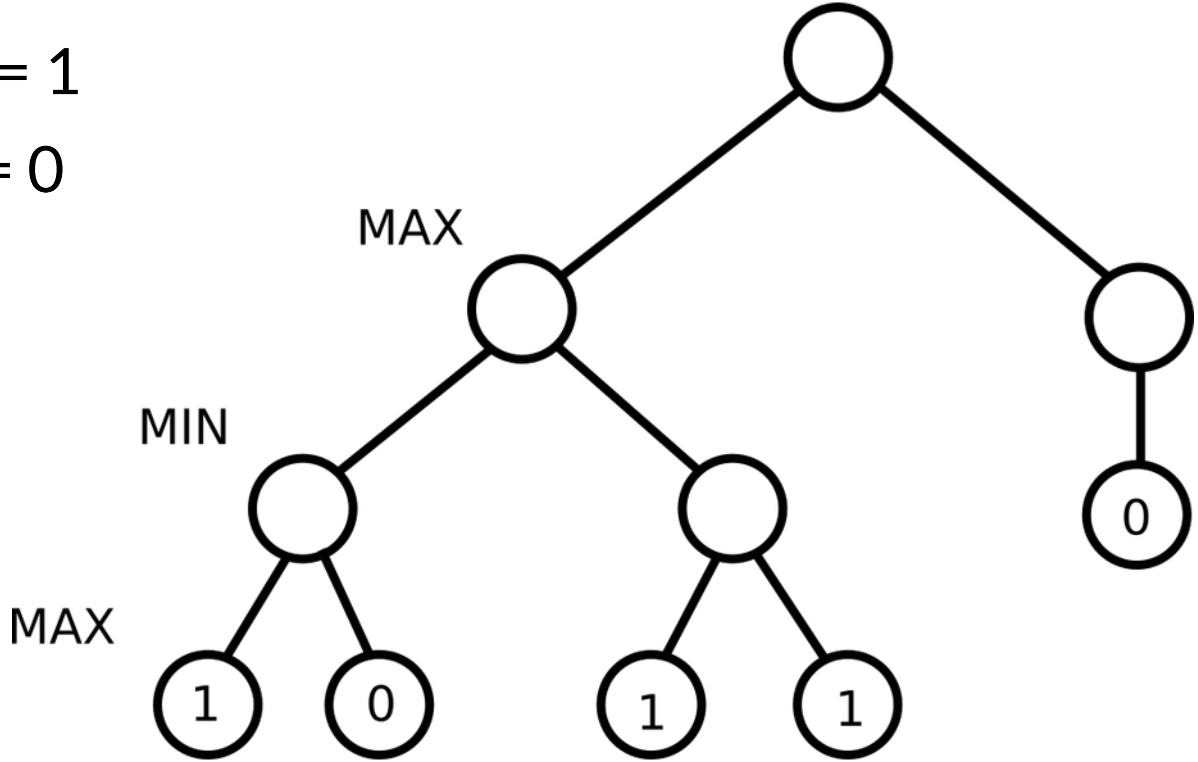
- Find a **path to goal state**
  - Path is a sequence of actions = **a plan**
- **Deterministic?**
  - Execute plan step by step
- **Stochastic**/Multiplayer?
  - Execute 1st action in plan
  - Replan

- Guided Search
  - We can use **heuristics** to guide our search (remember **A***)
  - Heuristics often involve **domain-specific knowledge**
    - e.g. count value of peices in chess
    - What would be a good heuristic for the farmer puzzle?
  - **MCTS** uses **rollouts** (playing the game randomly until win/lose) to decide how interesting a state is
    - It’s an example of a **general game-playing** algorithm

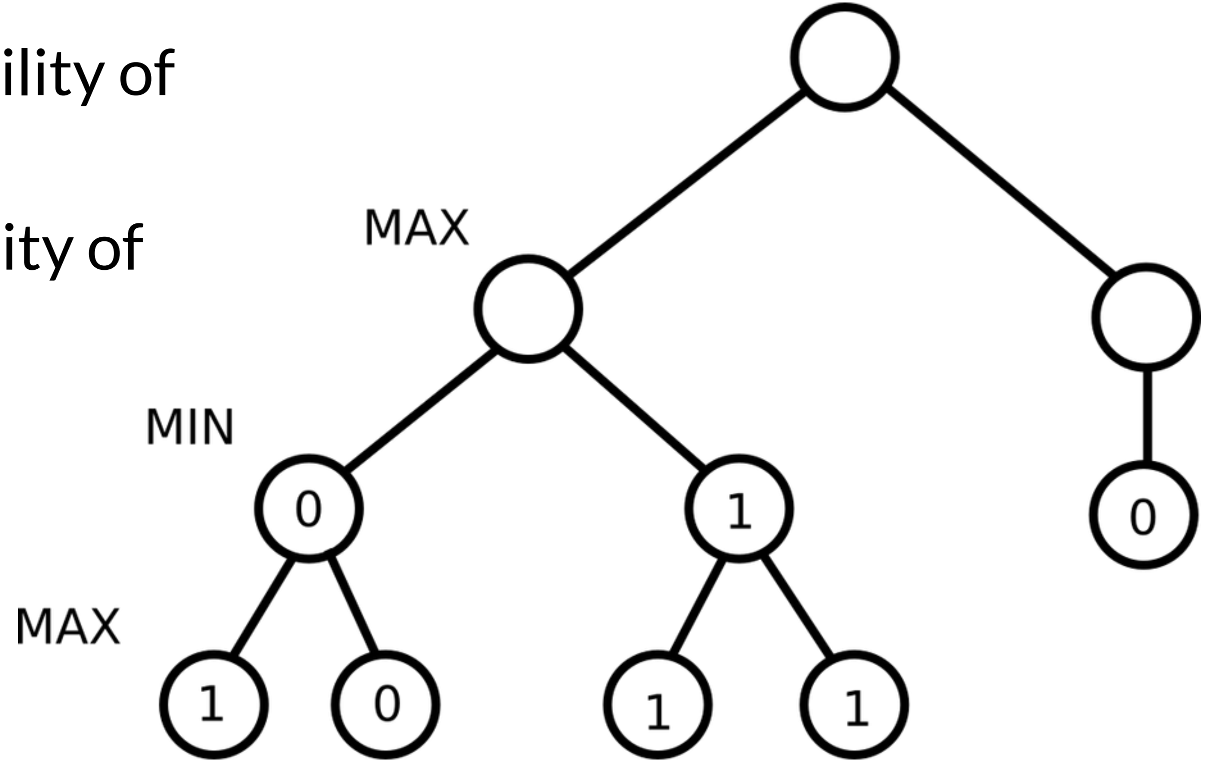
Adversarial Search

- Two-player **zero-sum games** (e.g. chess/tic-tac-toe)
  - Two players called **MIN** and **MAX**
- Imagine we're playing MAX
  - On our turn we (MAX) want to pick a **good move**
  - On their turn, MIN wants to pick a **bad move** (for us)
    - Assume MIN is **playing optimally**

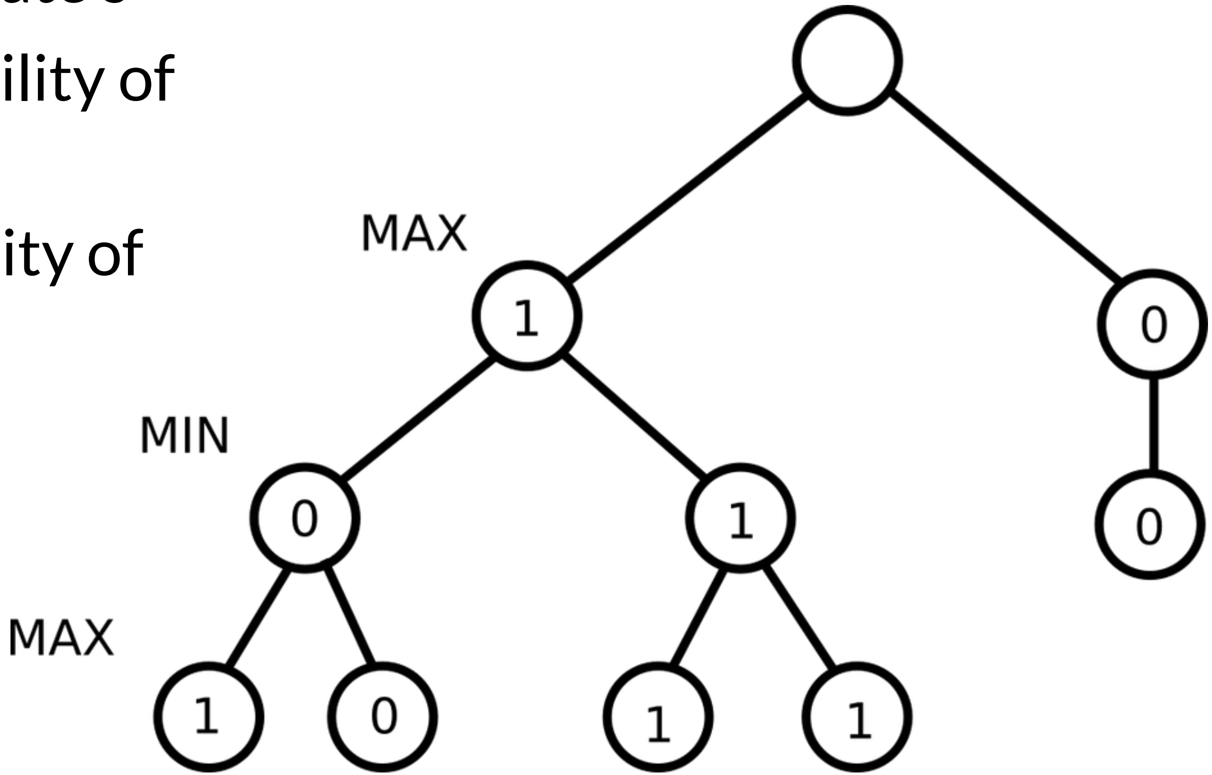
- Terminal States
  - If MAX wins, utility = 1
  - If MIN wins, utility = 0



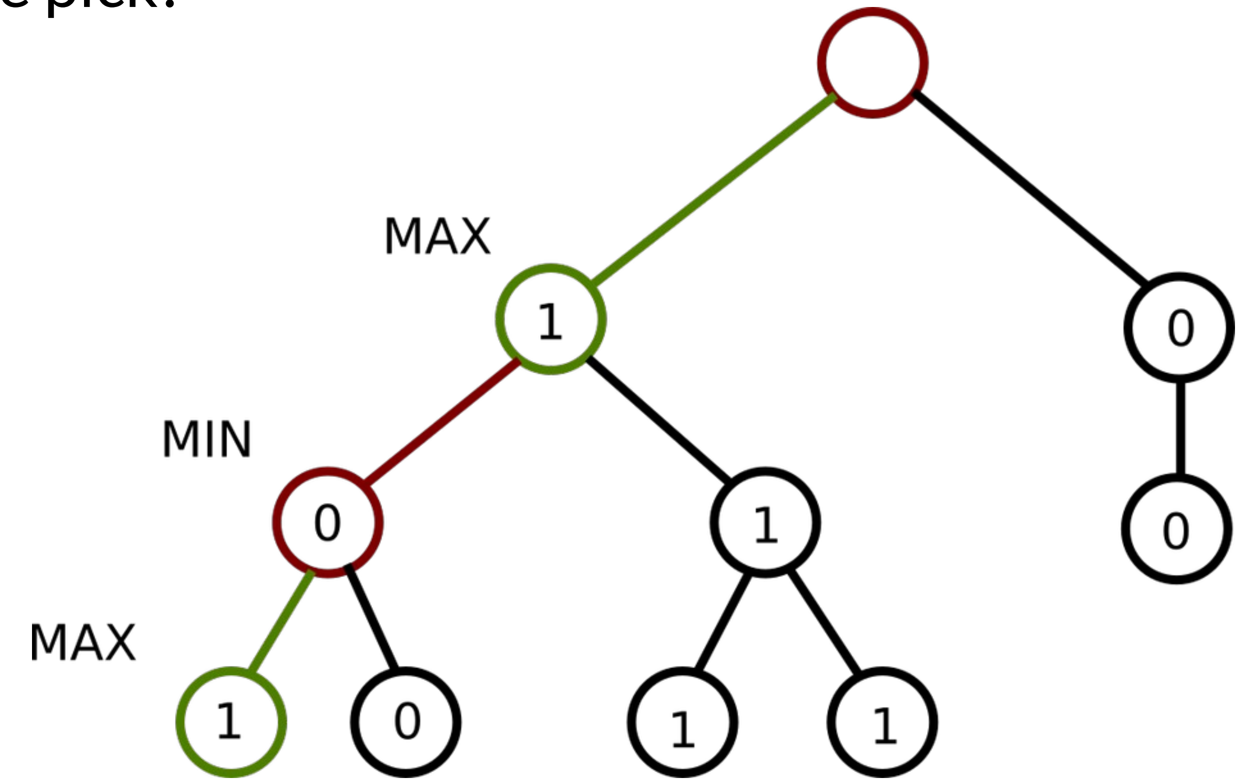
- MINIMAX value of a state s
  - MAX's turn: **max** utility of all children
  - MIN's turn: **min** utility of all children



- MINIMAX value of a state s
  - MAX’s turn: **max** utility of all children
  - MIN’s turn: **min** utility of all children

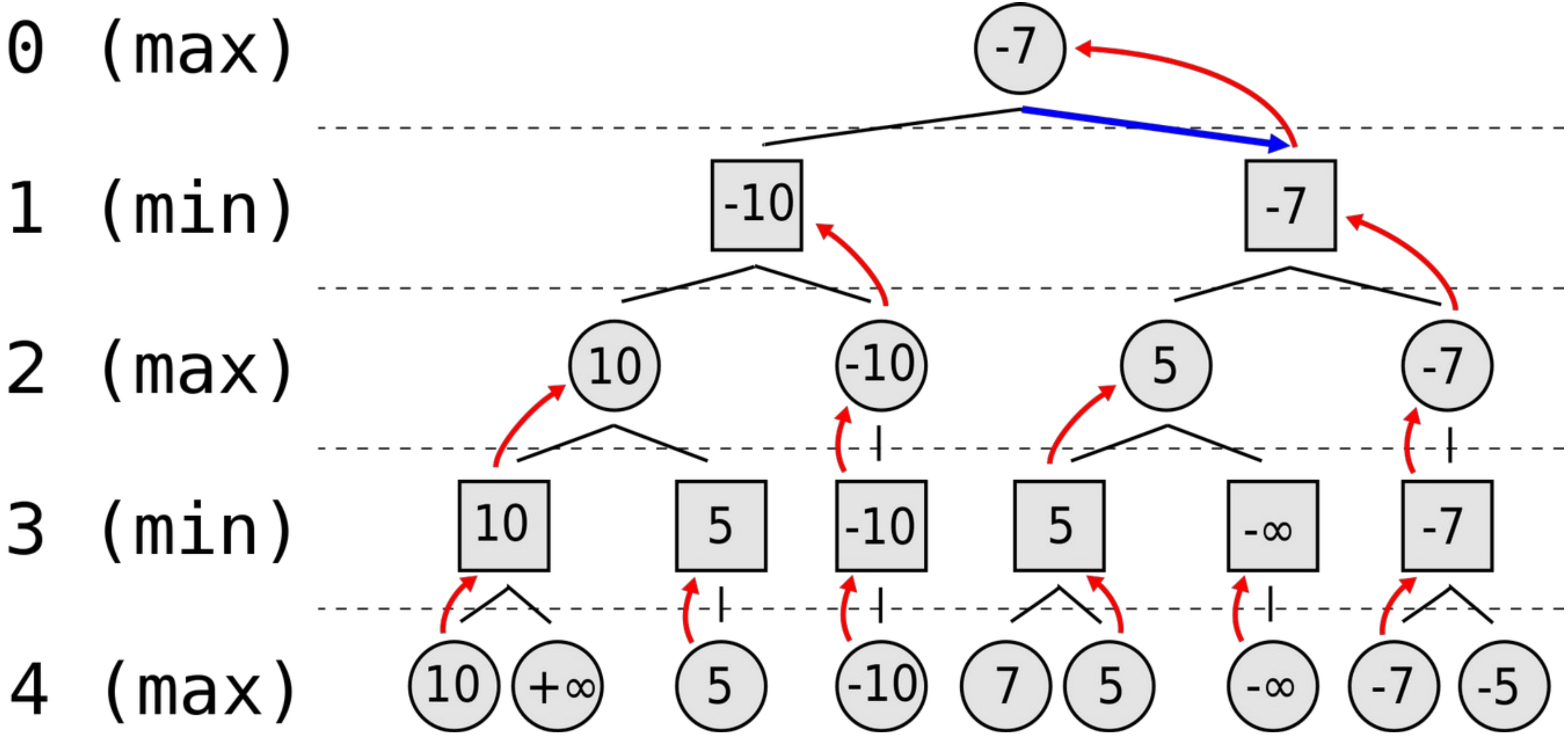


- Which move should we pick?



- The Minimax Algorithm

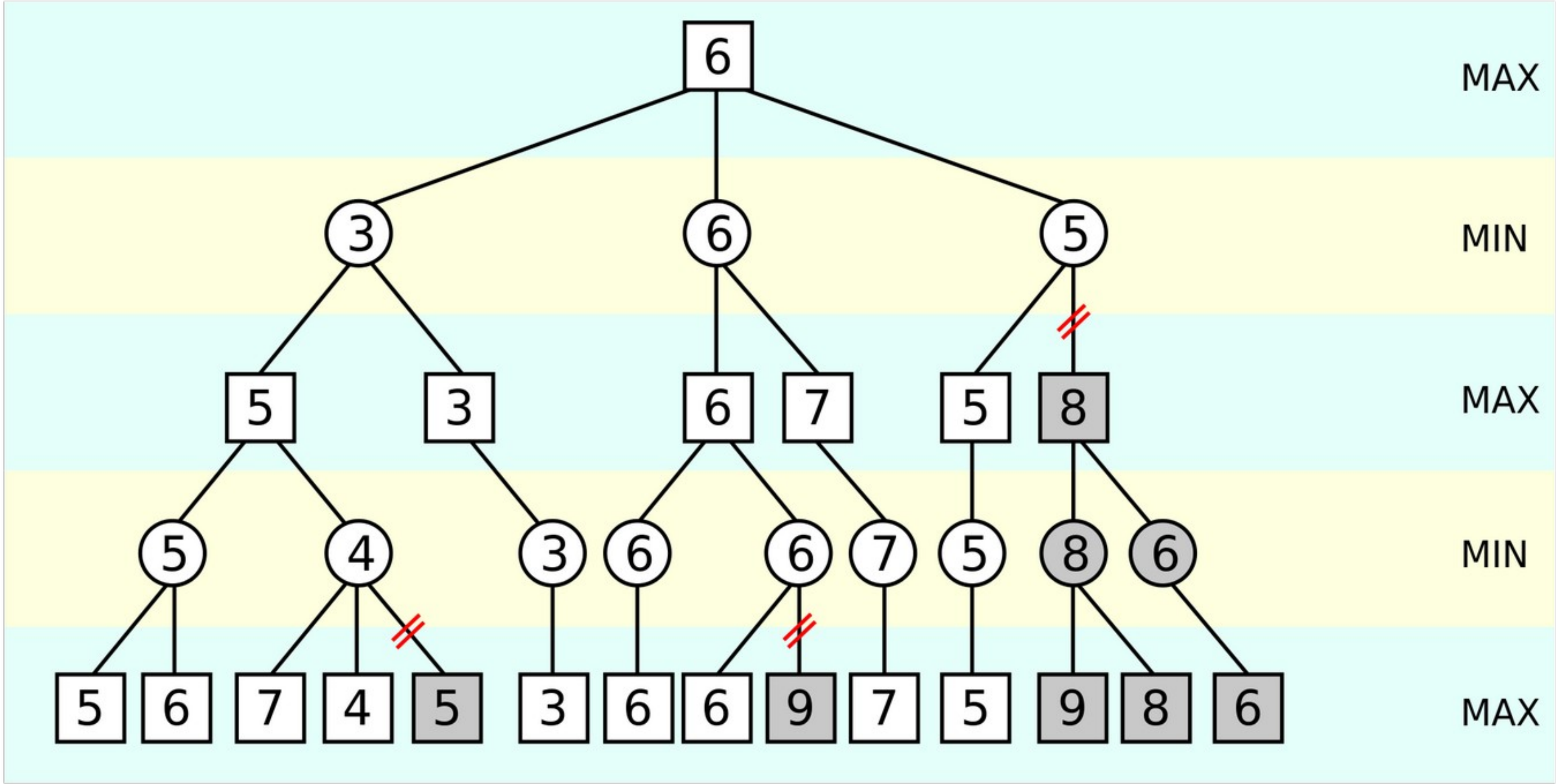
```
function minimax(node, maximizingPlayer)
    if node is a terminal node then
        return the heuristic value of node
    if maximizingPlayer then
        value := −∞
        for each child of node do
            value := max(value, minimax(child, FALSE))
        return value
    else /* minimizing player */
        value := +∞
        for each child of node do
            value := min(value, minimax(child, TRUE))
        return value
```



0 (max)
1 (min)
2 (max)
3 (min)
4 (max)
-7
-10
-7
10
-10
5
-7
10
5
-10
5
-∞
-7
10
+∞
5
-10
7
5
-∞
-7
-5

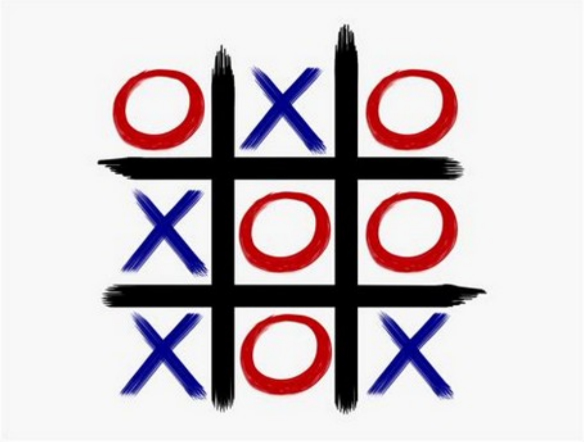
- Alpha-beta Pruning
  - Sometimes parts of a minimax tree will have no effect on the final outcome
    - We can “prune” those branches (not explore them)

- Say you’ve got a good move, Move A
  - You check other moves just in case one is better
  - Move B looks pretty good at first, but then you find your opponent can force a win
  - Even if every other outcome of Move B is amazing, you’ll never want to take Move B.


6
MAX
3
6
5
MIN
5
3
6
7
5
8
MAX
5
4
3
6
6
7
5
8
6
MIN
5
6
7
4
5
3
6
6
9
7
5
9
8
6
MAX

Abstracting for Search

- **Branching Factor**
  - The more moves available, the more search required



- **Depth**
  - The move move it takes to a goal state, the more search required
    - Use **depth-limited** search, and value non-terminal states using a heuristic

- **Continuous games** still have states - they're just very quick (leading to very deep trees)
  - MCTS can play simple continuous games
  - **Macro actions** = e.g. "Move Left for 10 turns"

- Too complex?
  - Devise strong heuristics
  - Search a **simplified version** of the game
    - Just like we can simplify our pathfinding graph
      - Non-optimal, but faster
  - Cheat

- Summary
  - Construct tree of game states
  - Search tree for path to goal
  - Can do adversarial search
  - Games are often too complex for simple search algorithms
    - Use heuritics
    - Alpha-beta pruning
    - MCTS

- Recommended Reading
  - Russel & Norvig, Ch. 5