Alpha Go and Reinforcement Learning

Vincent Lepetit and Pascal Fua
ParisTech EPFL

Historical Perspective

1997: Chess program Deep Blue defeats Garry Kasparov.

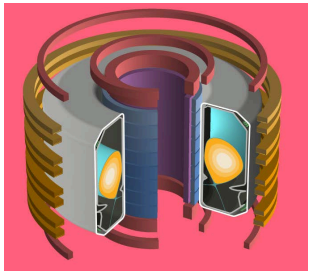
Until 2015: Go programs still play at amateur level.

October 2015: AlphaGo defeats Fan Hui, European Go champion, 5-0. First time a program has beaten a professional player.

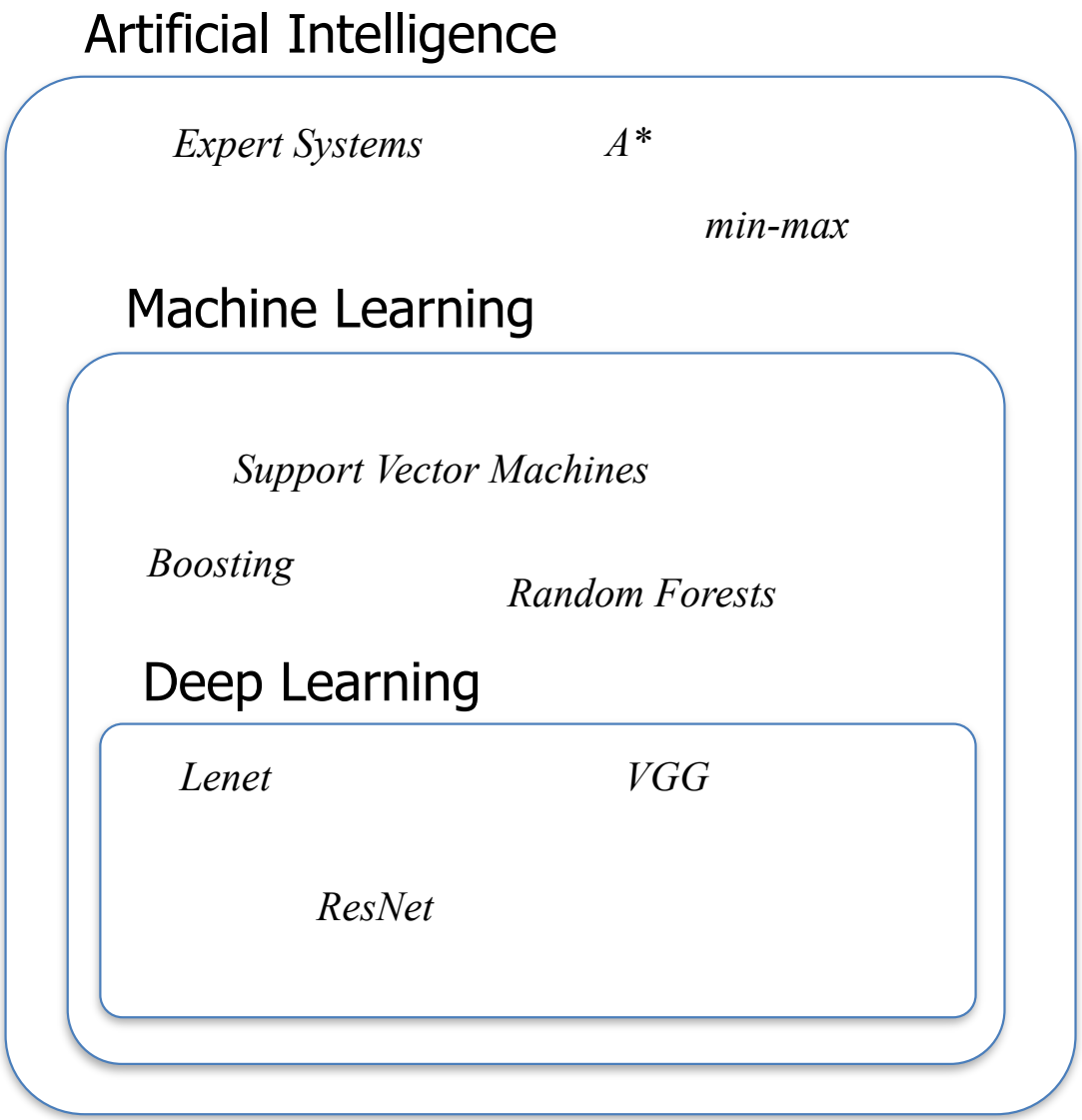
March 2016: AlphaGo Lee defeats Lee Sedol, one of the best world players, 4–1.

2017: Publication of AlphaGo Zero. 100-0 victory against AlphaGo after 3 days of training.

2018: Publication of AlphaZero. Same algorithm learns to play Go, Chess, and Shogi. Defeats all previous programs at Chess and Shogi.



2022: An algorithm of the same nature is used to control plasma in EPFL's Tokamak



Artificial Intelligence:

Algorithms that search the space of possible solutions.

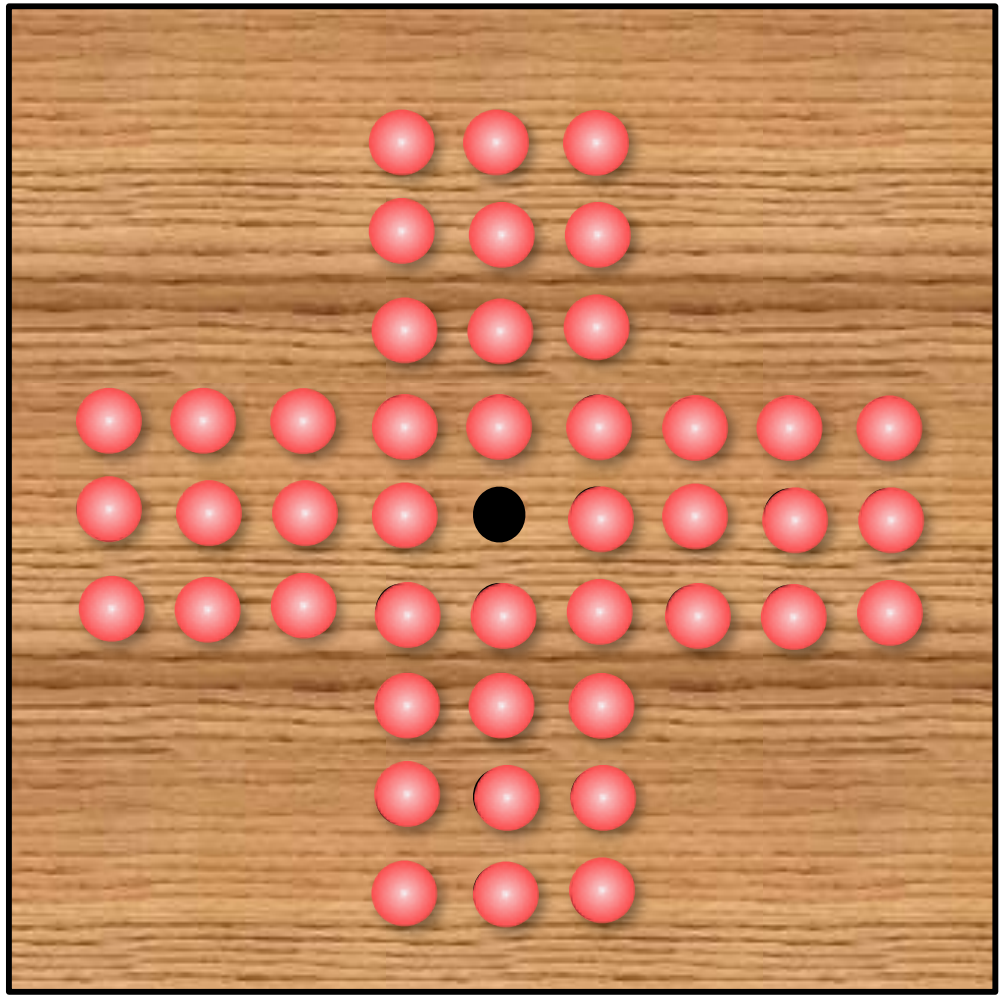
Machine Learning:

Algorithms whose performance can be improved using training data.

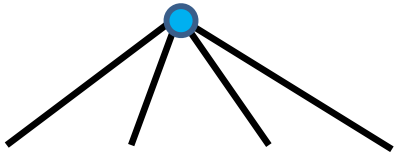
Deep Learning:

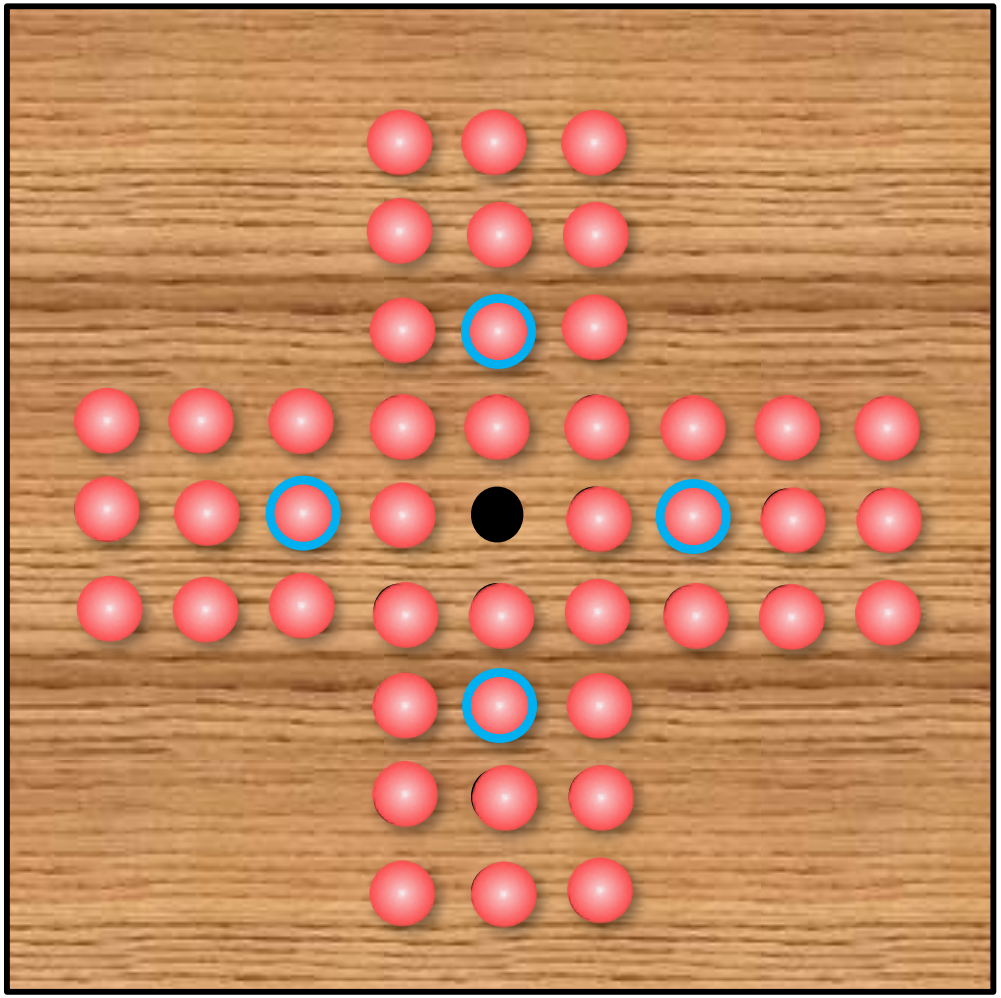
Algorithms whose performance can be improved using a lot of training data.

AI Formulation of Peg Solitaire

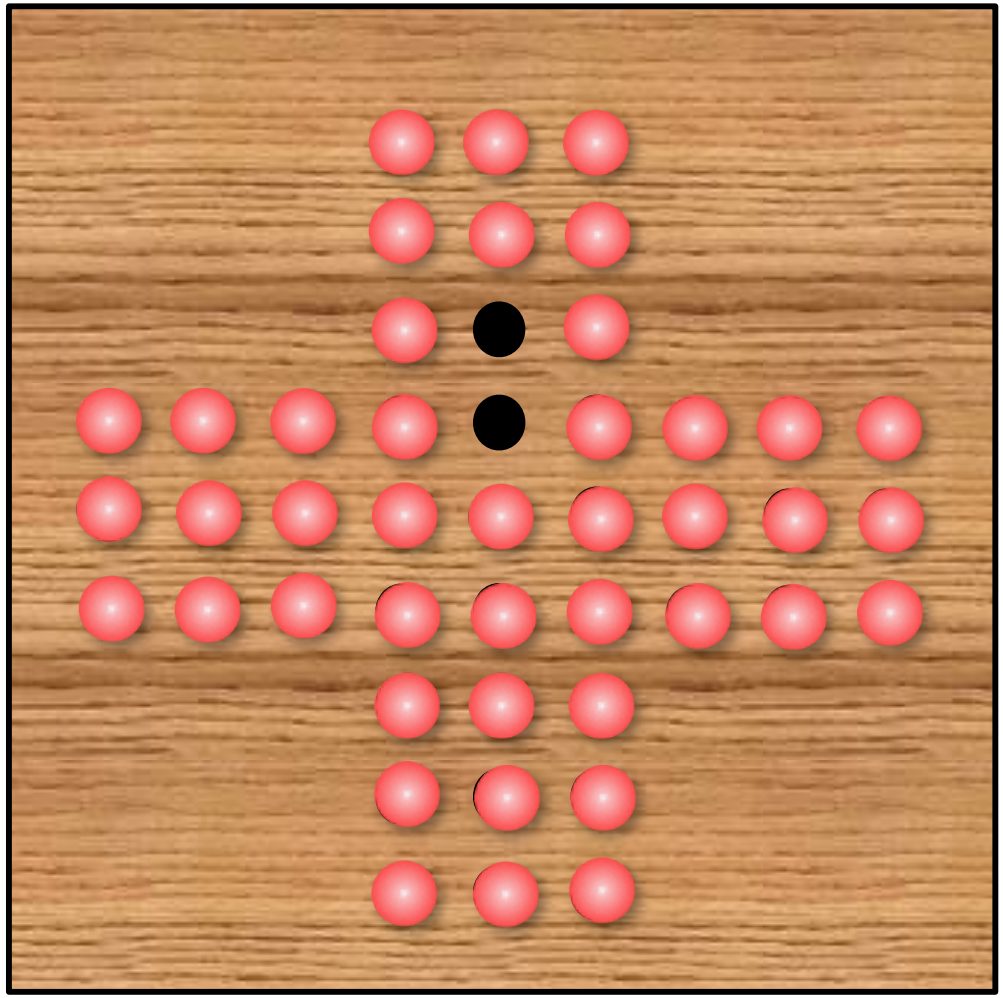


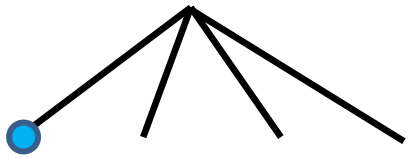
AI Formulation of Peg Solitaire



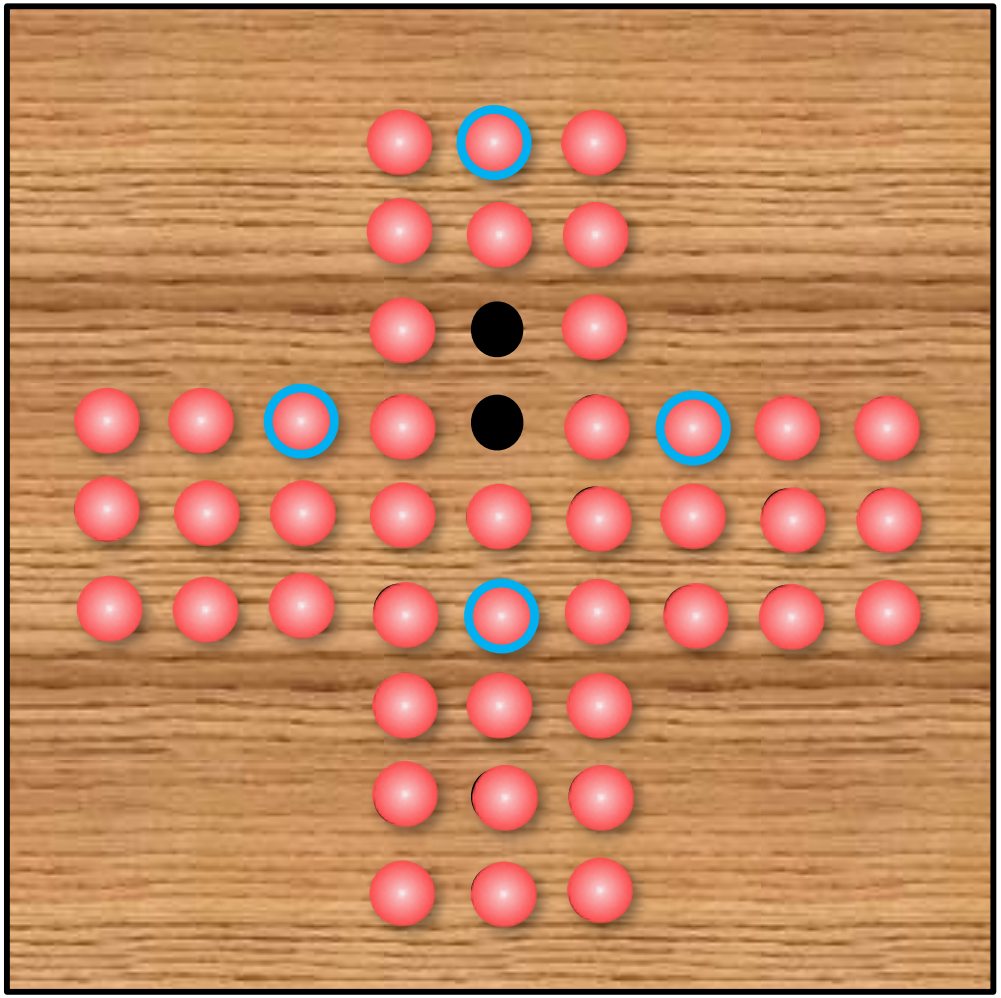


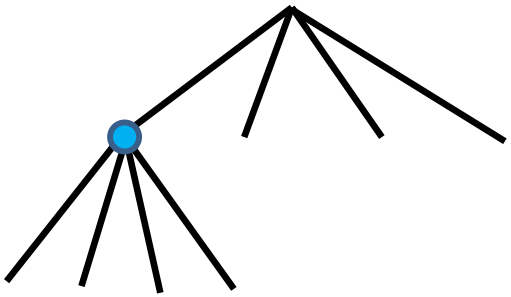
AI Formulation of Peg Solitaire



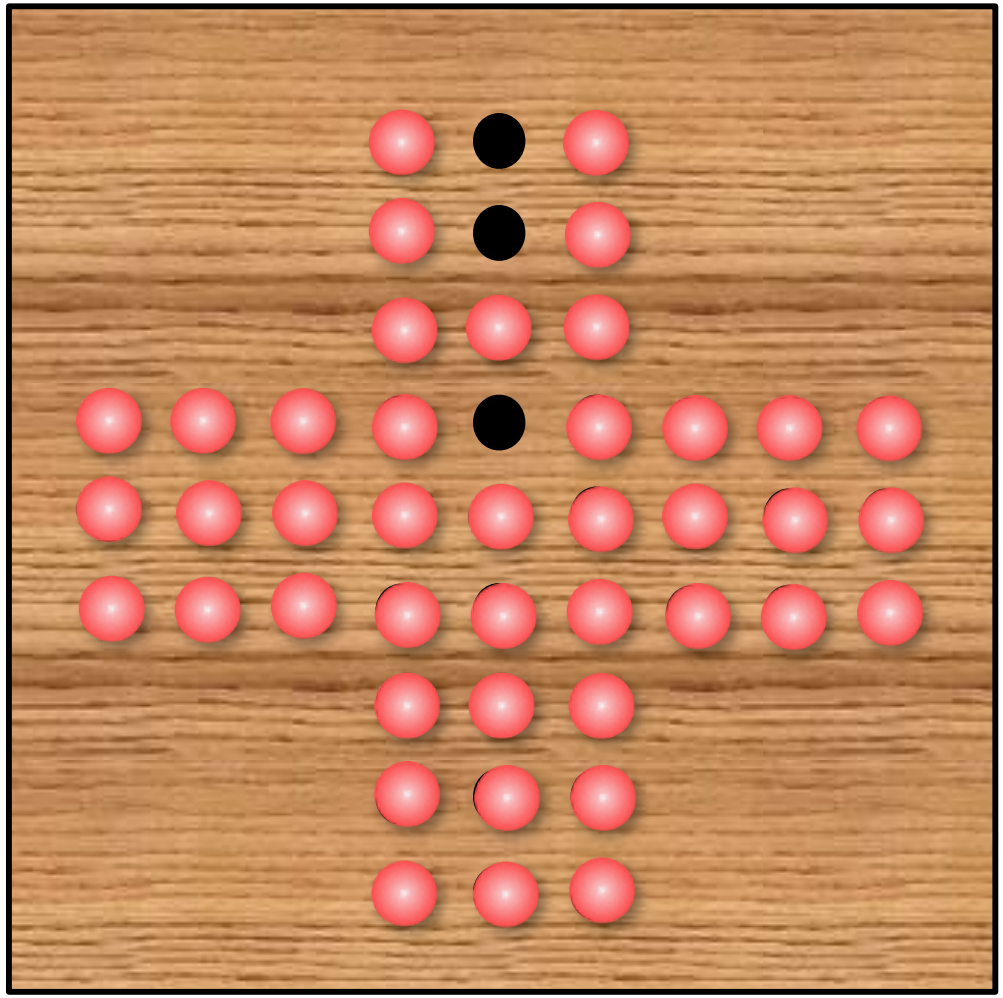


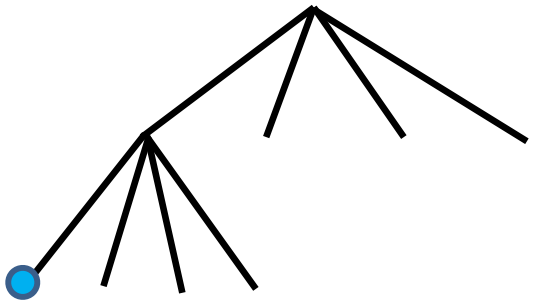
AI Formulation of Peg Solitaire



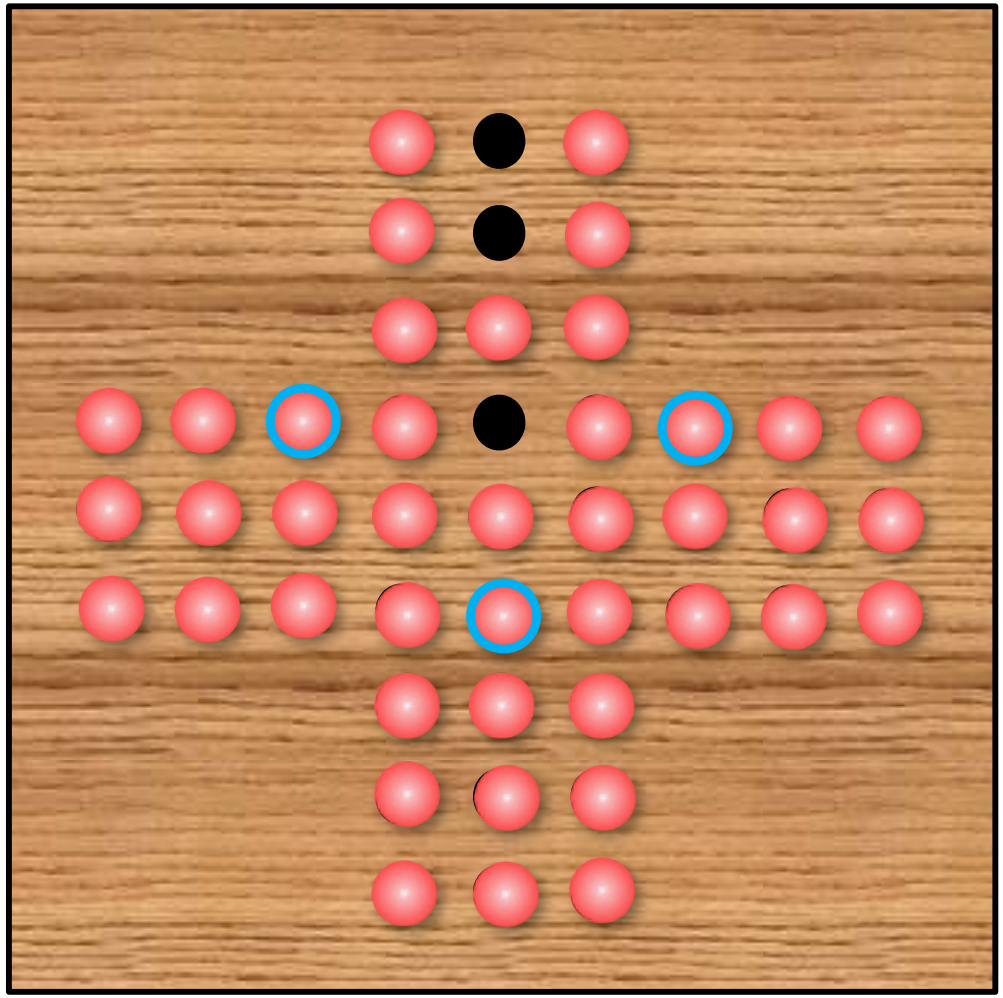


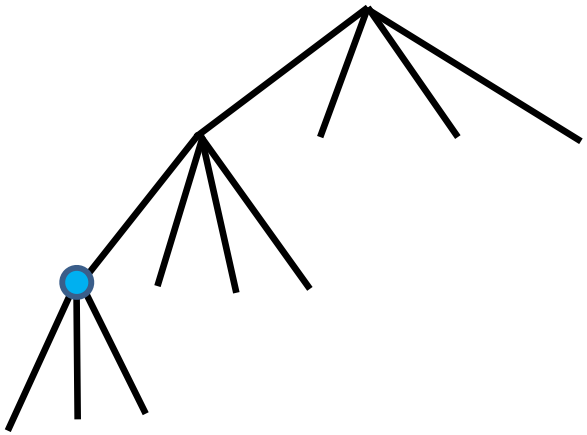
AI Formulation of Peg Solitaire



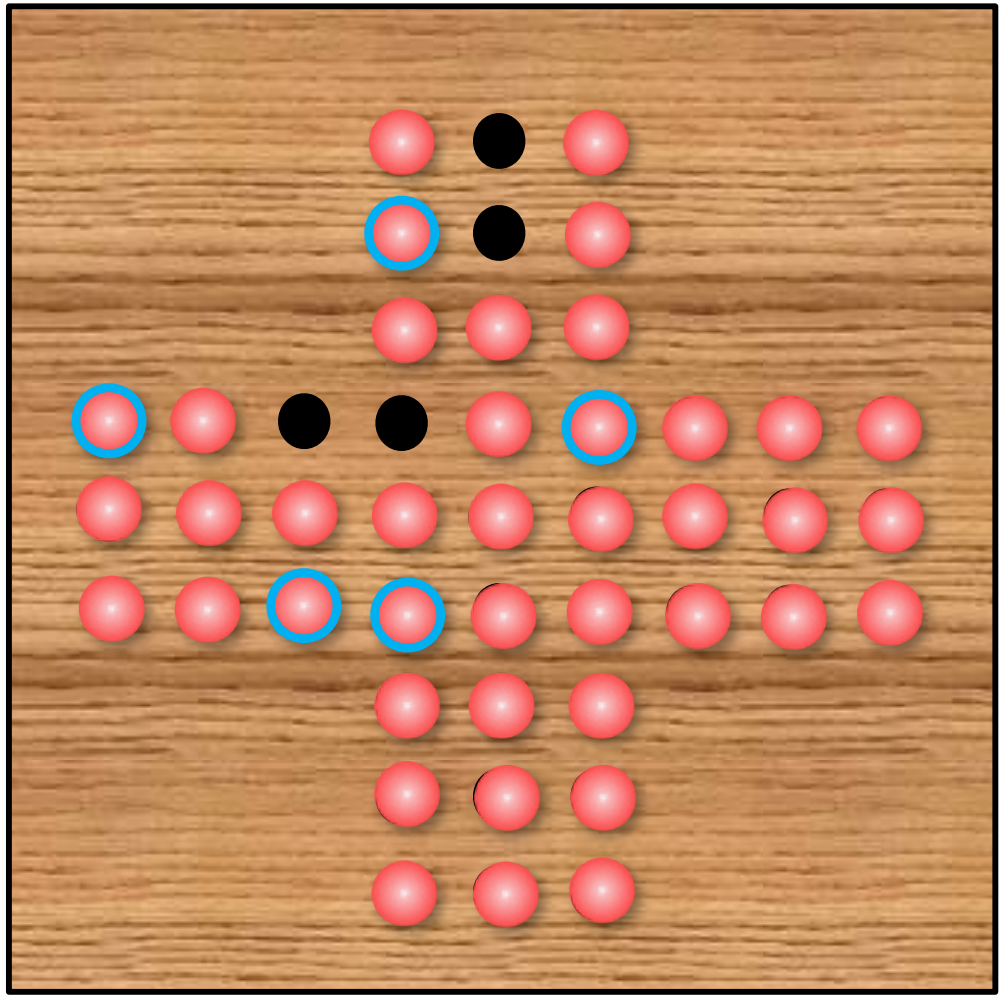


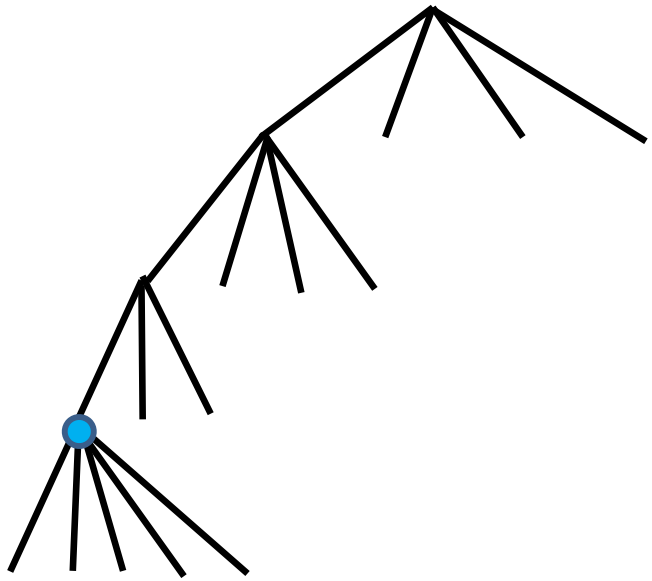
AI Formulation of Peg Solitaire



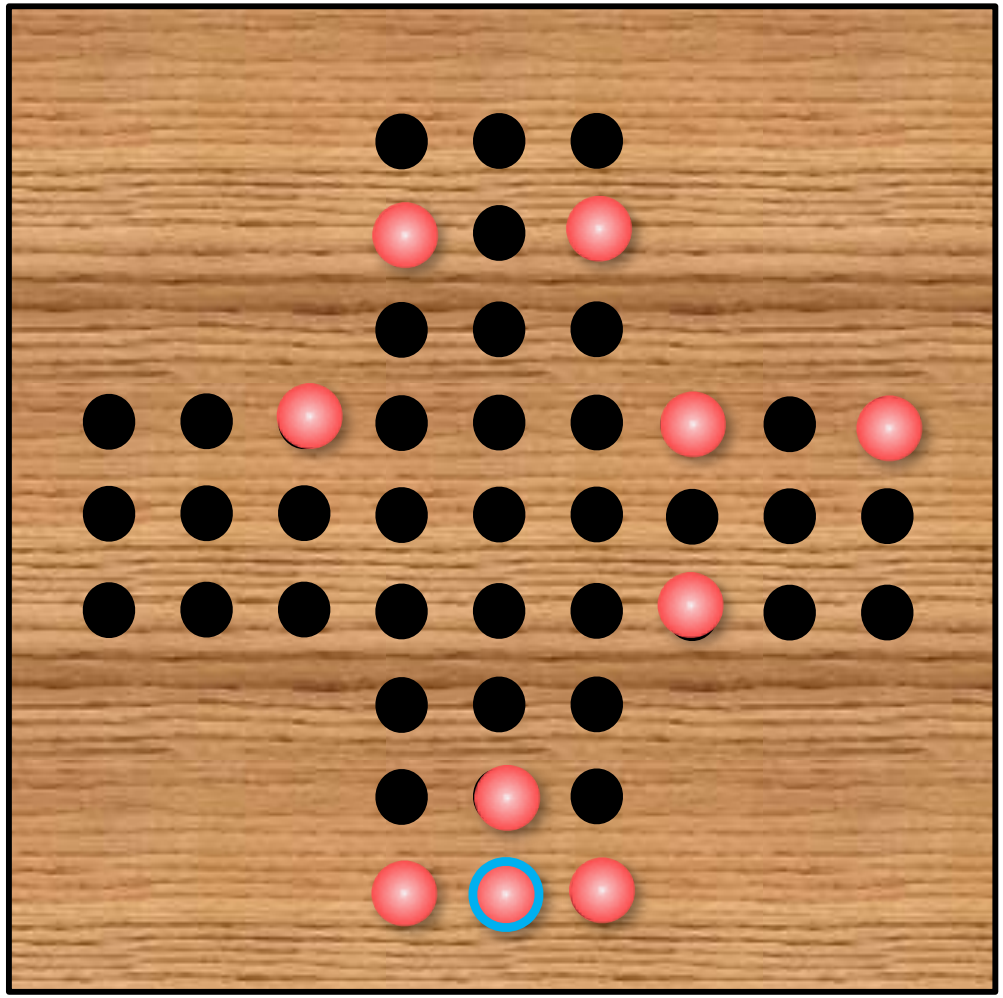


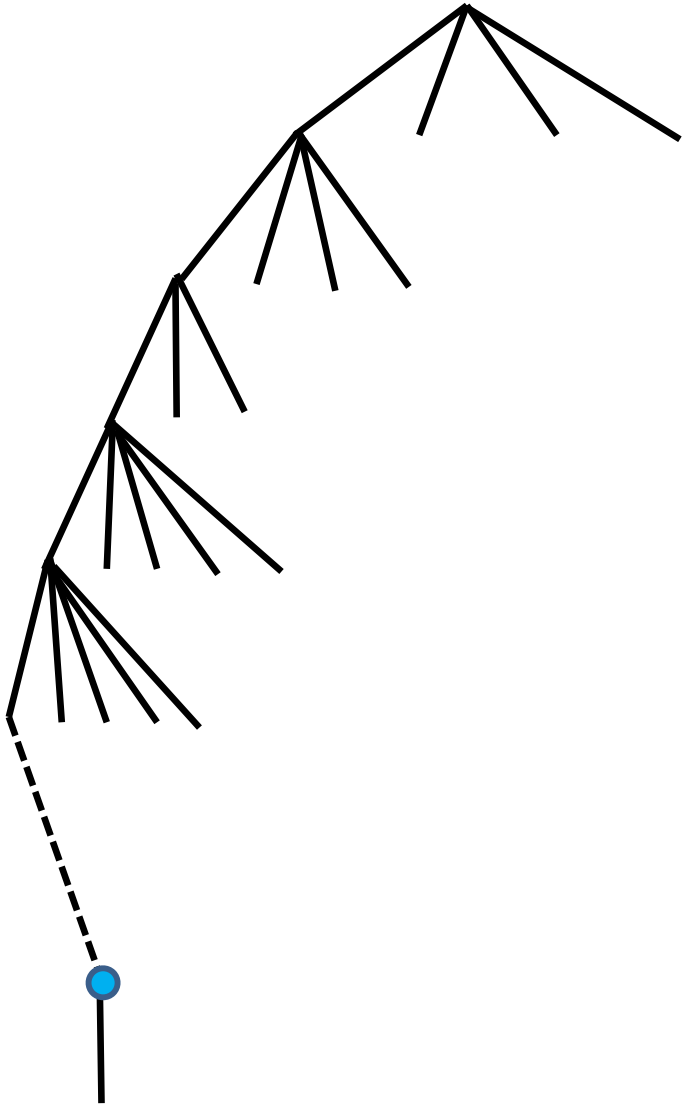
AI Formulation of Peg Solitaire



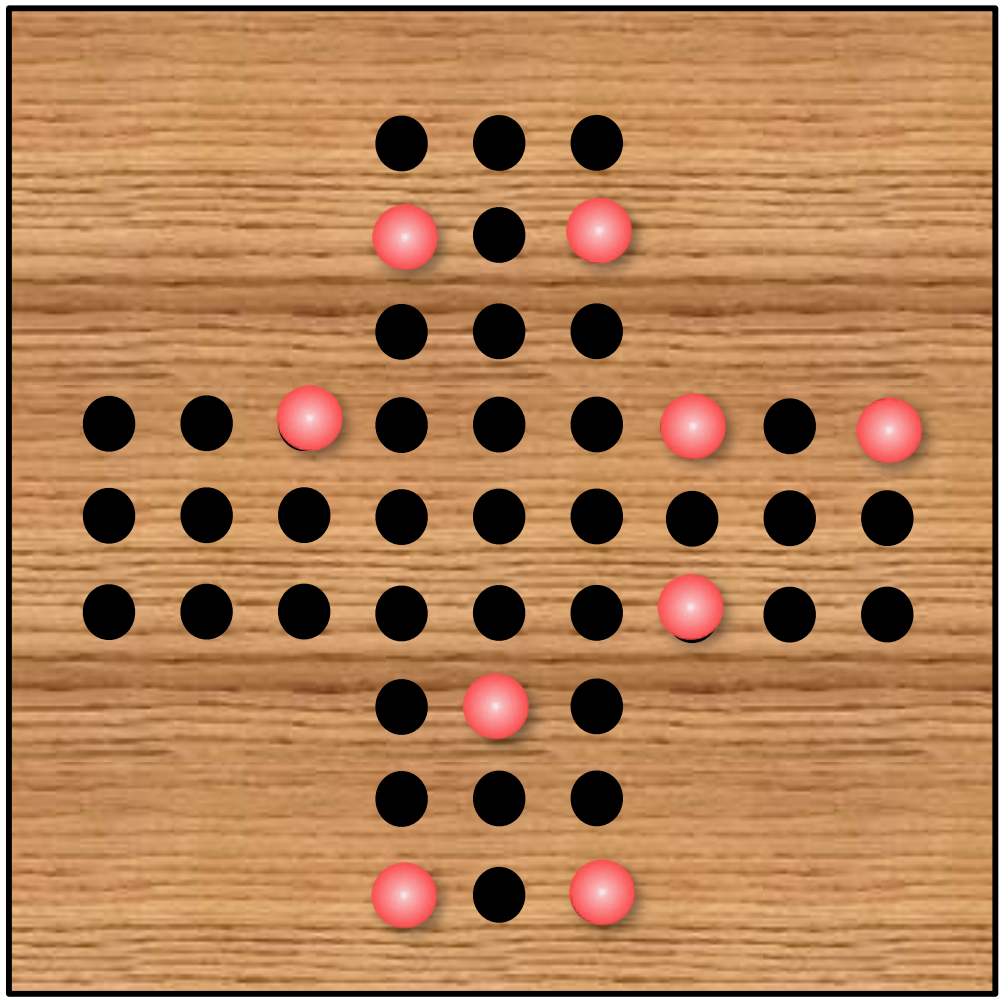


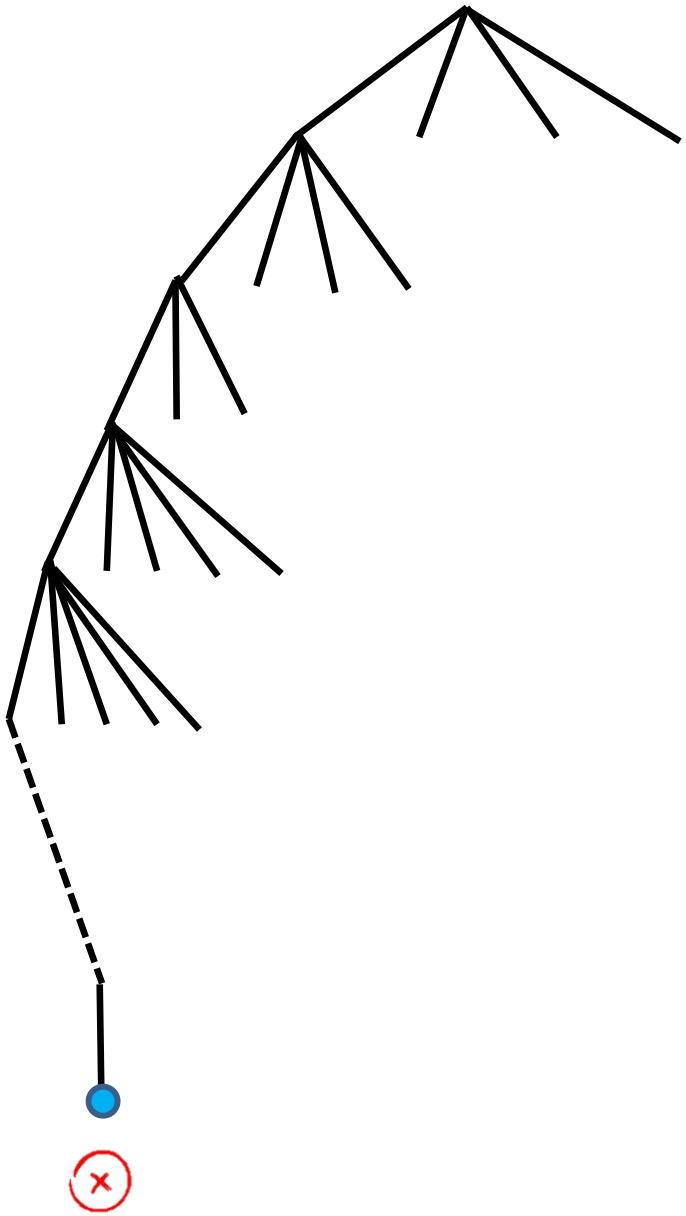
AI Formulation of Peg Solitaire



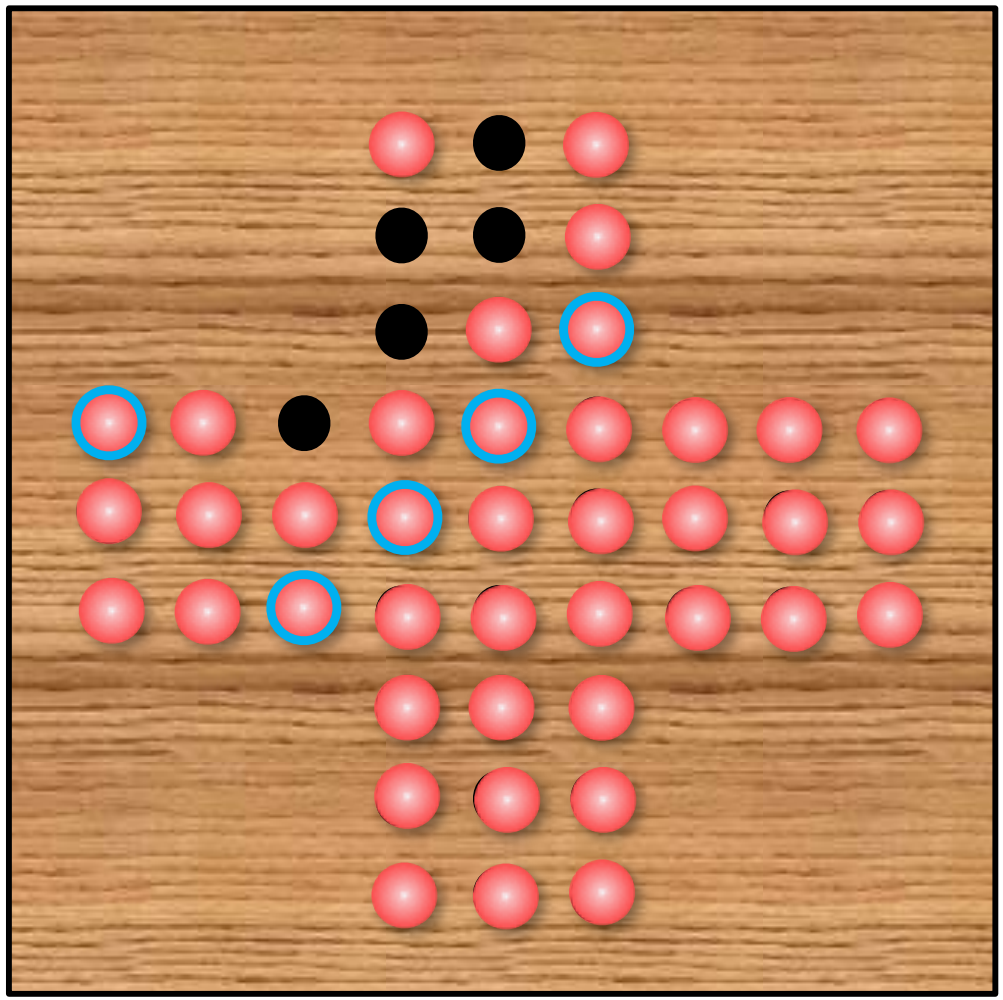


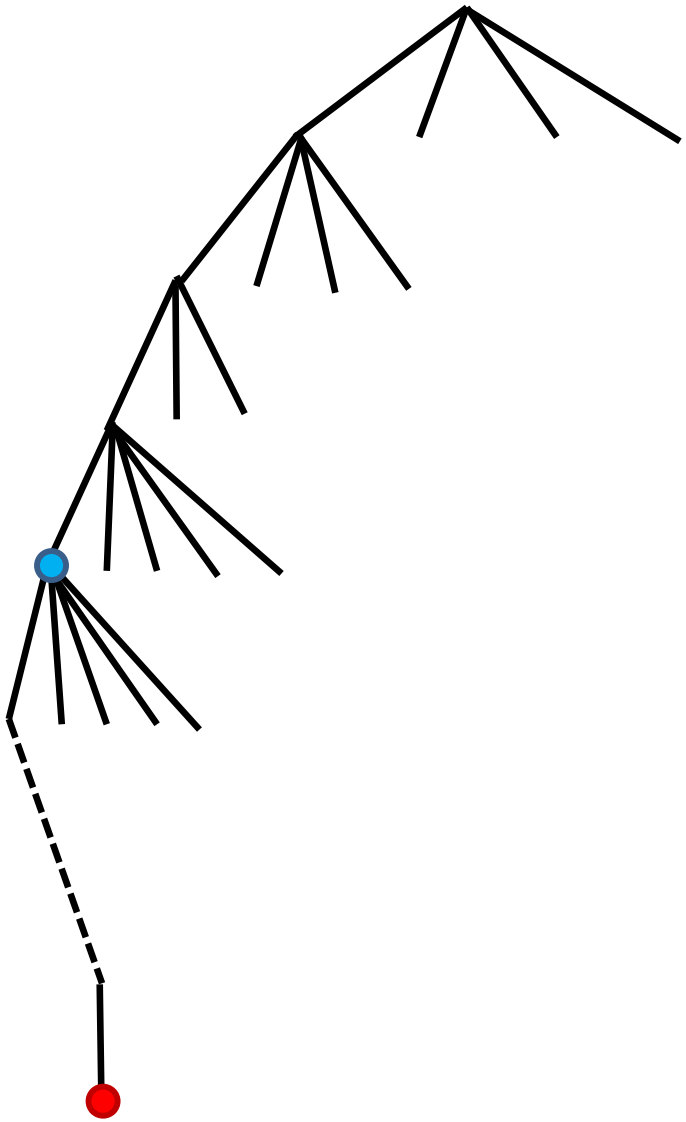
AI Formulation of Peg Solitaire



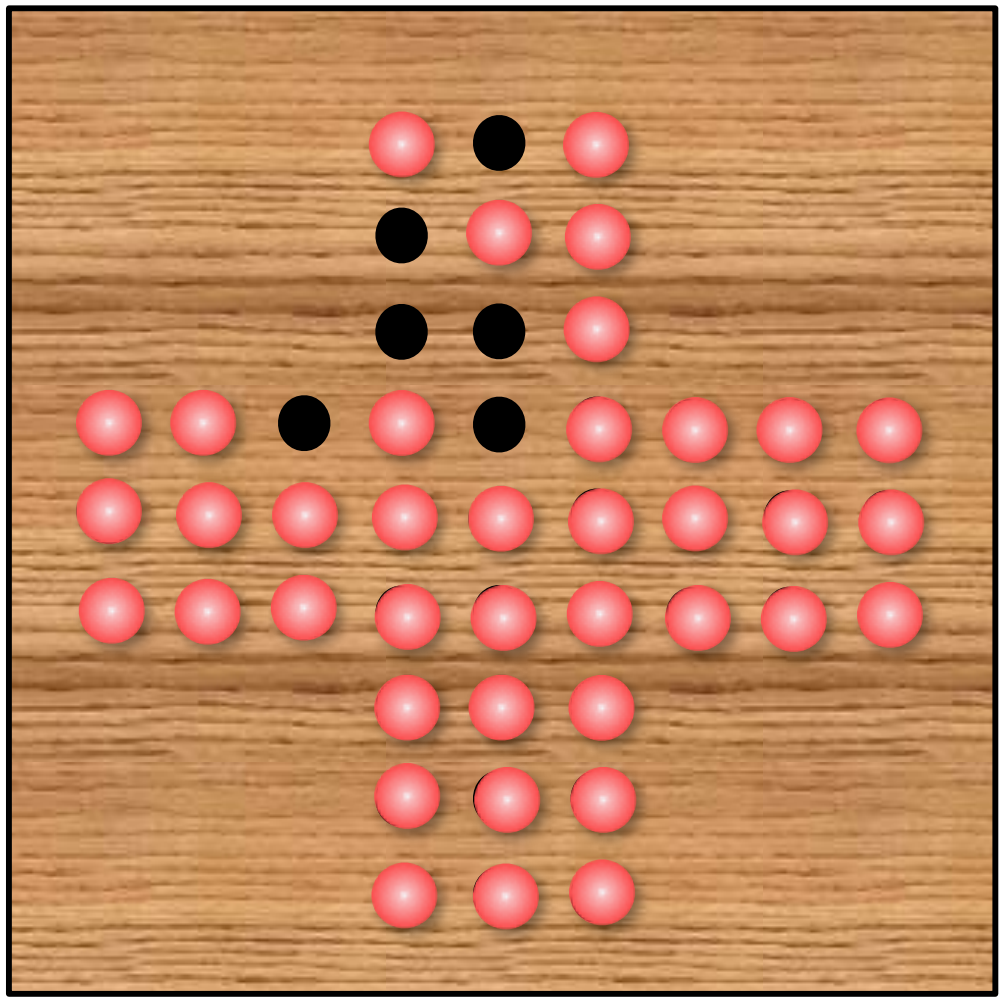


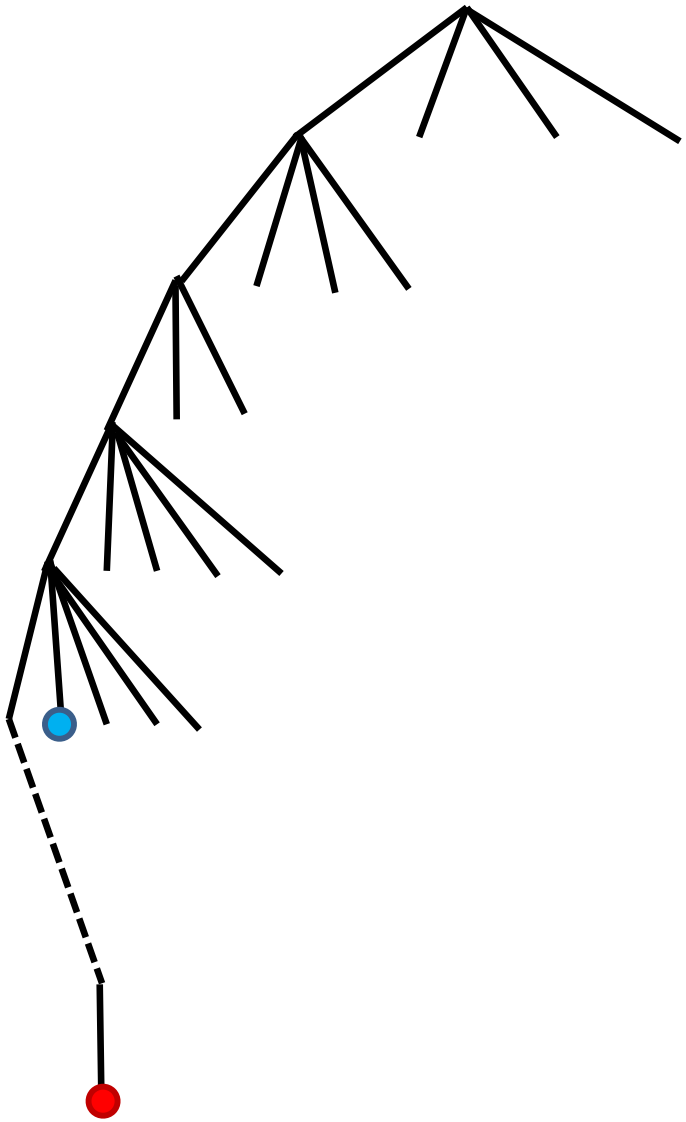
AI Formulation of Peg Solitaire



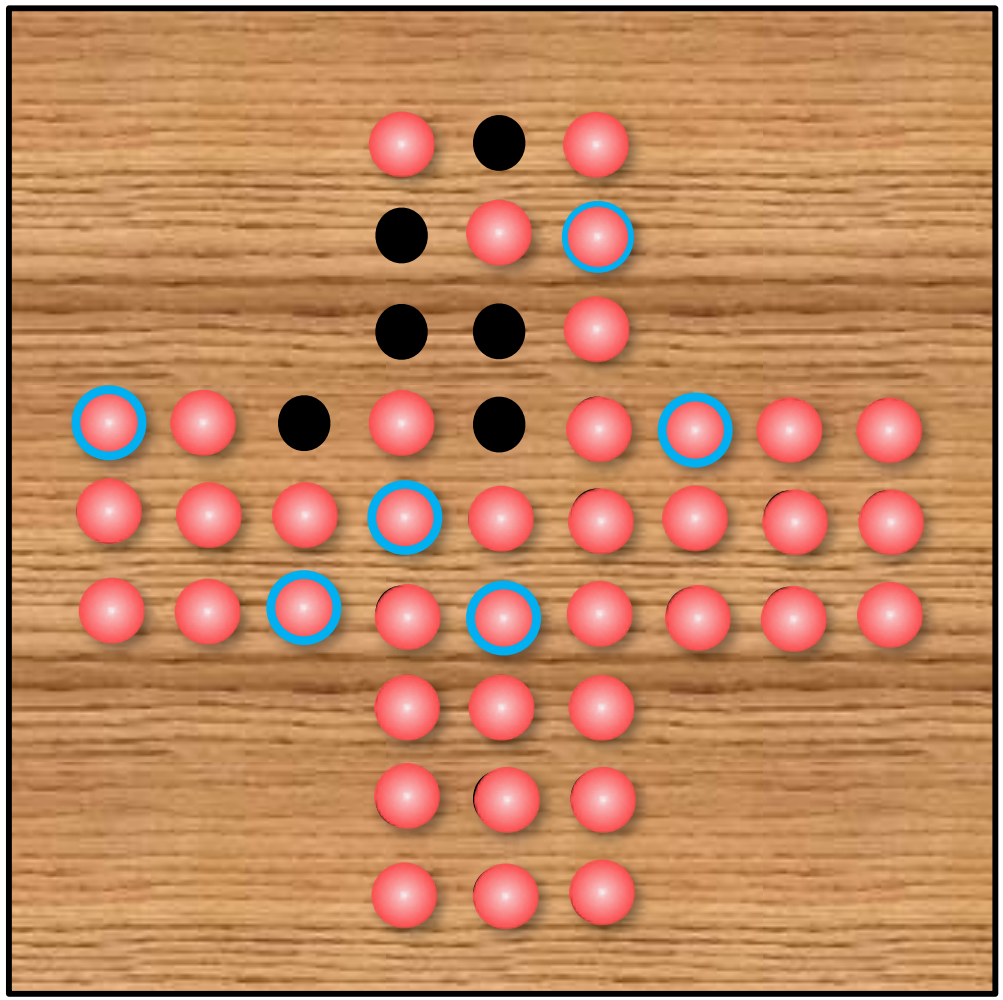


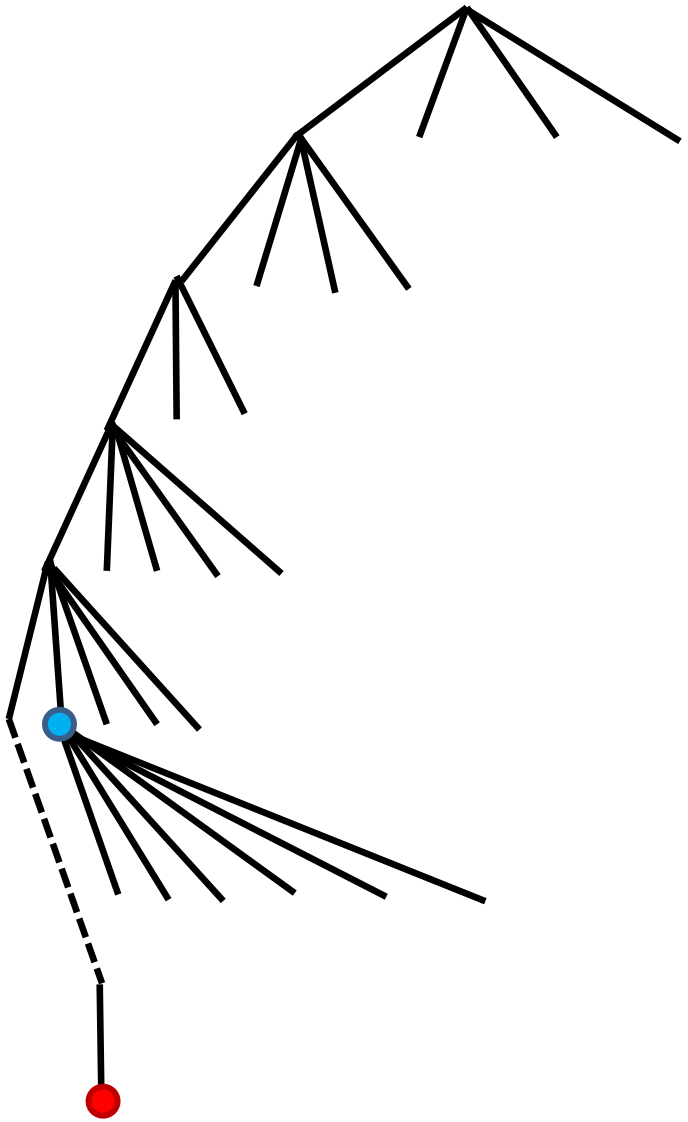
AI Formulation of Peg Solitaire



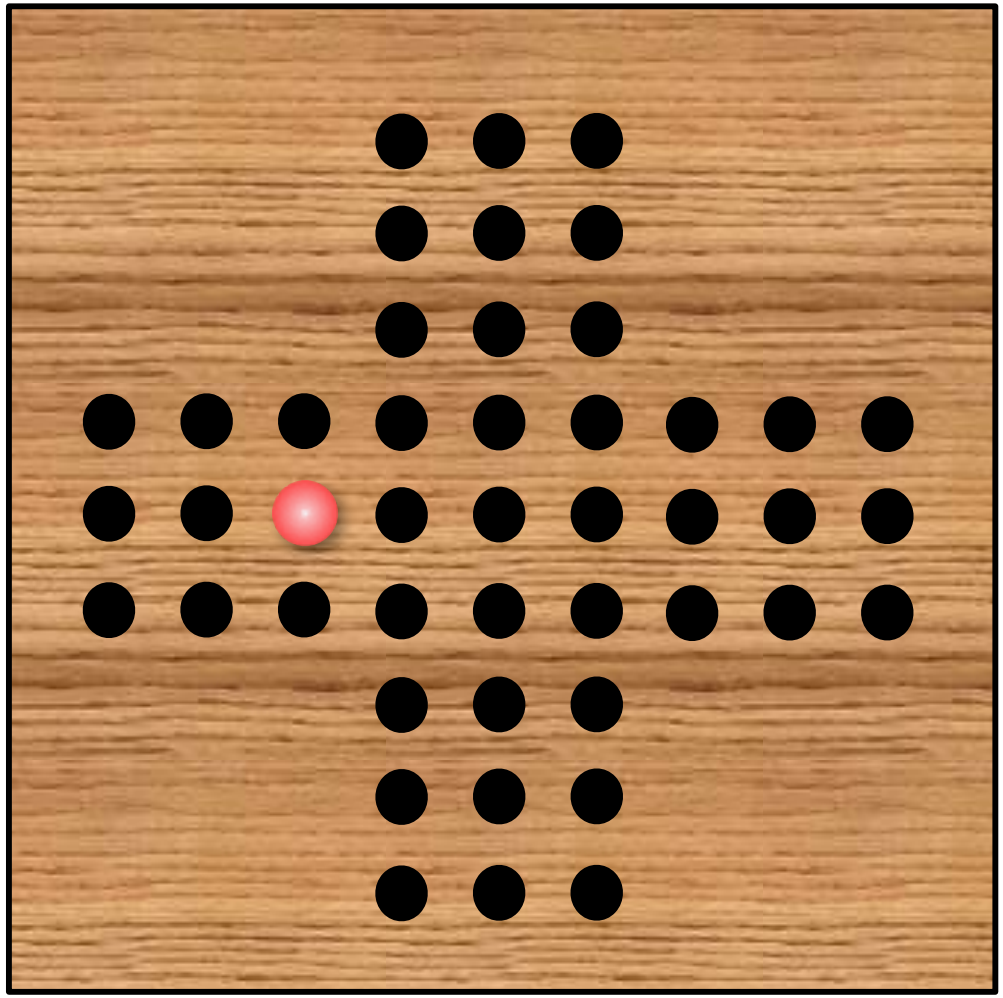


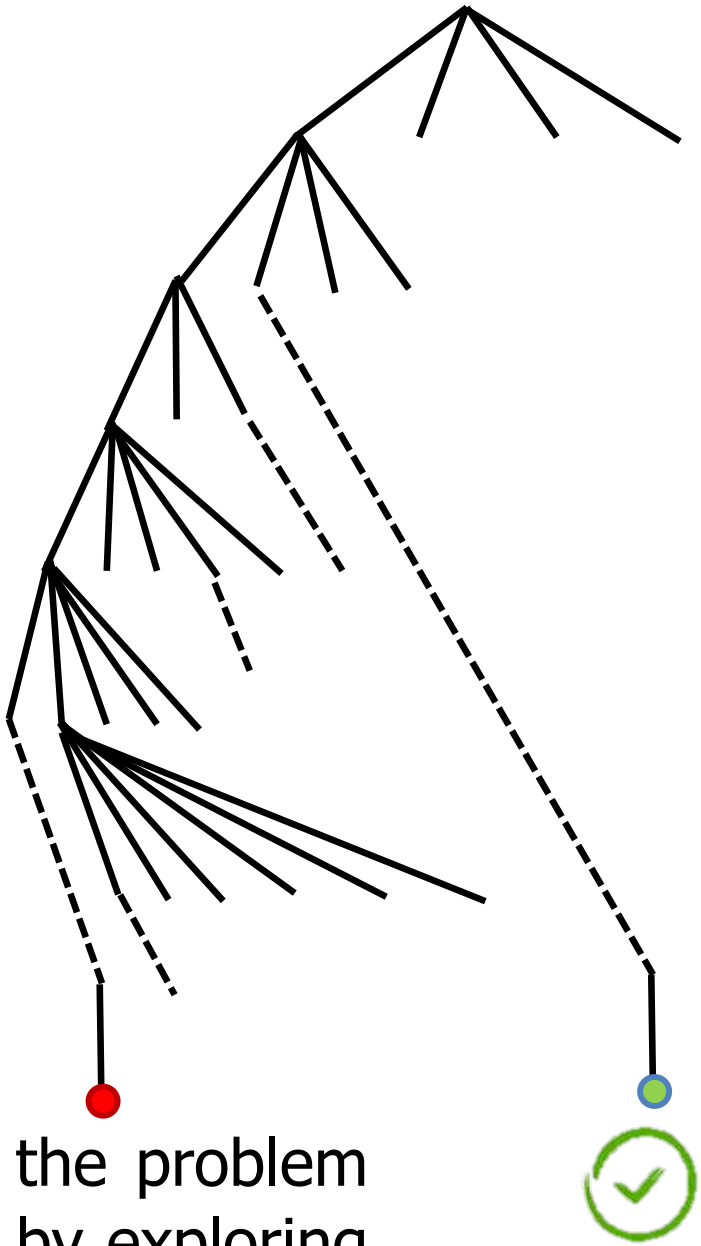
AI Formulation of Peg Solitaire





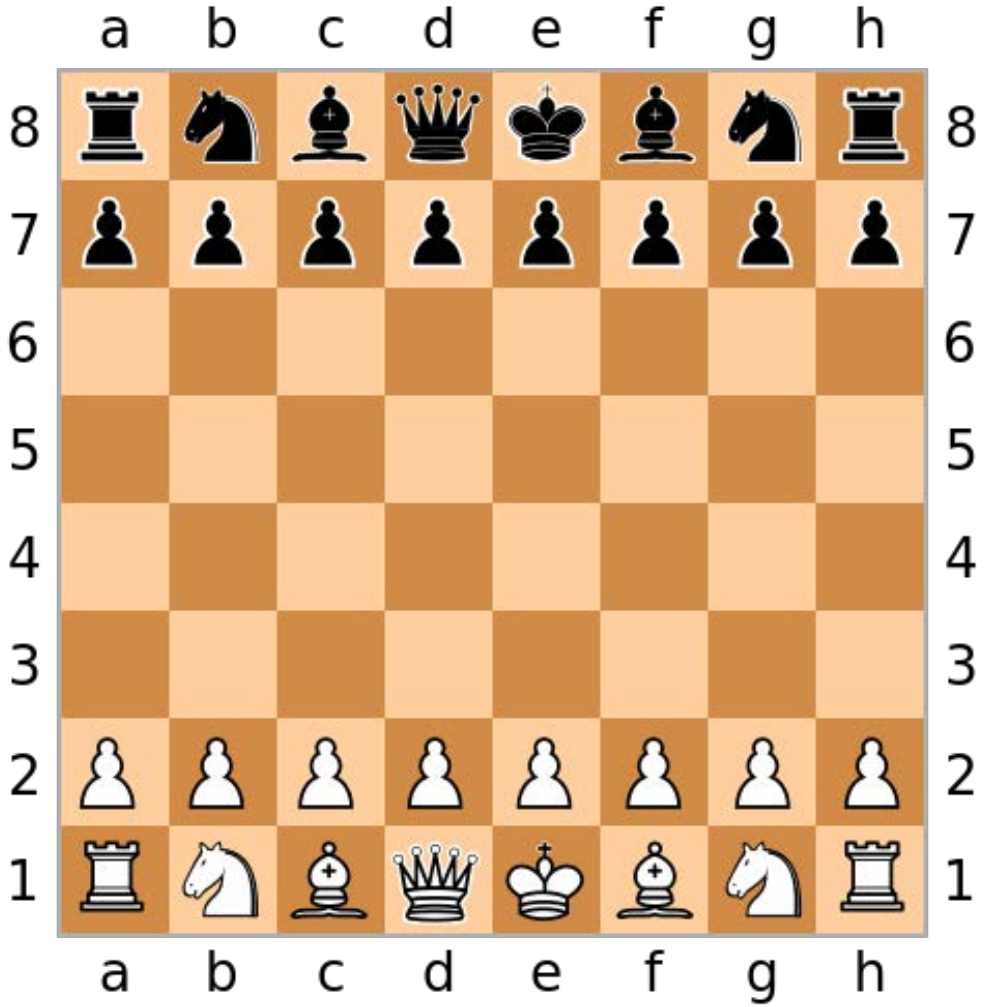
AI Formulation of Peg Solitaire



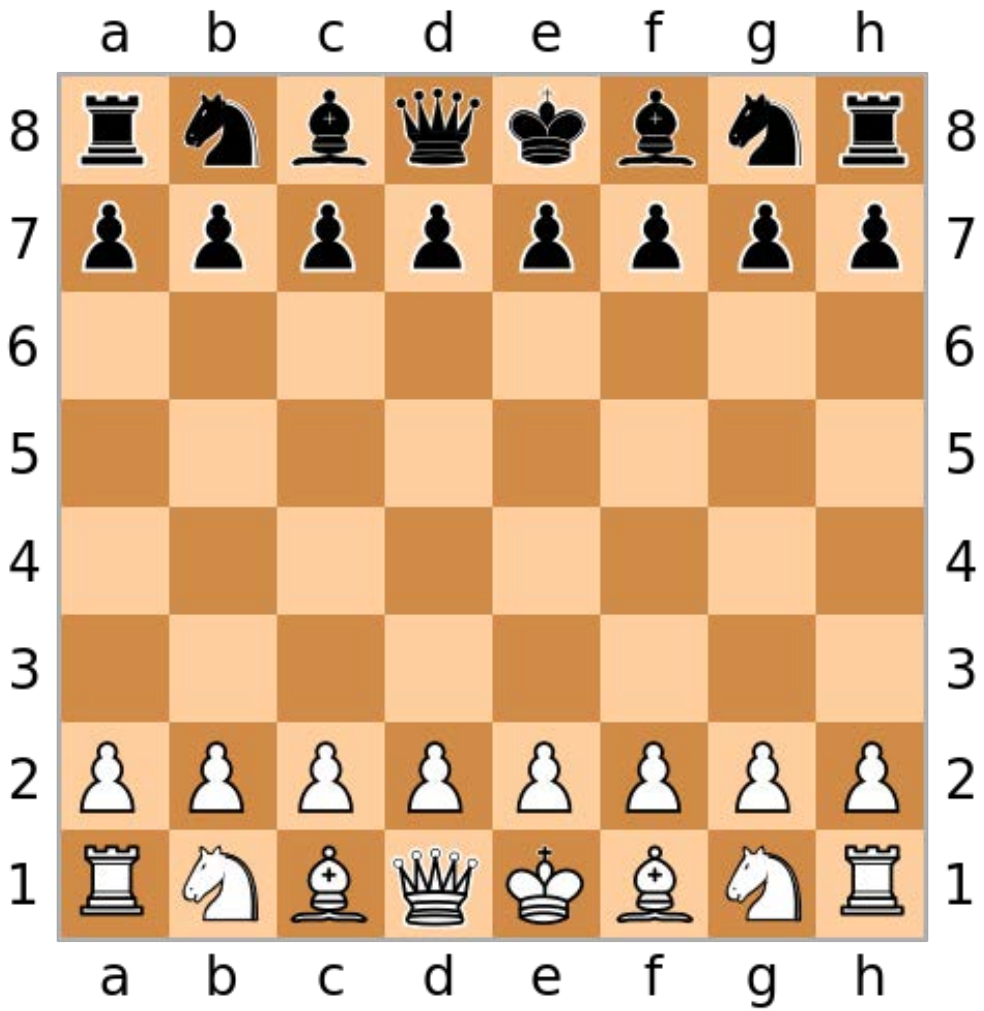


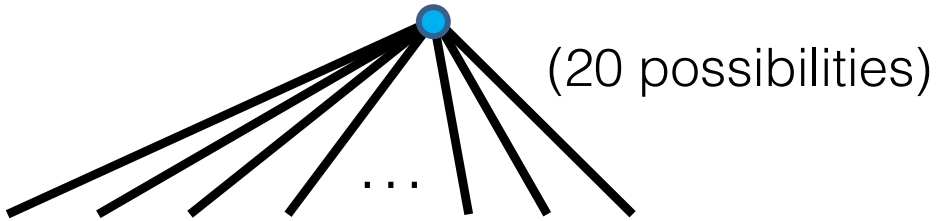
—>In theory, the problem can be solved by exploring a large tree of possibilities.

Chess

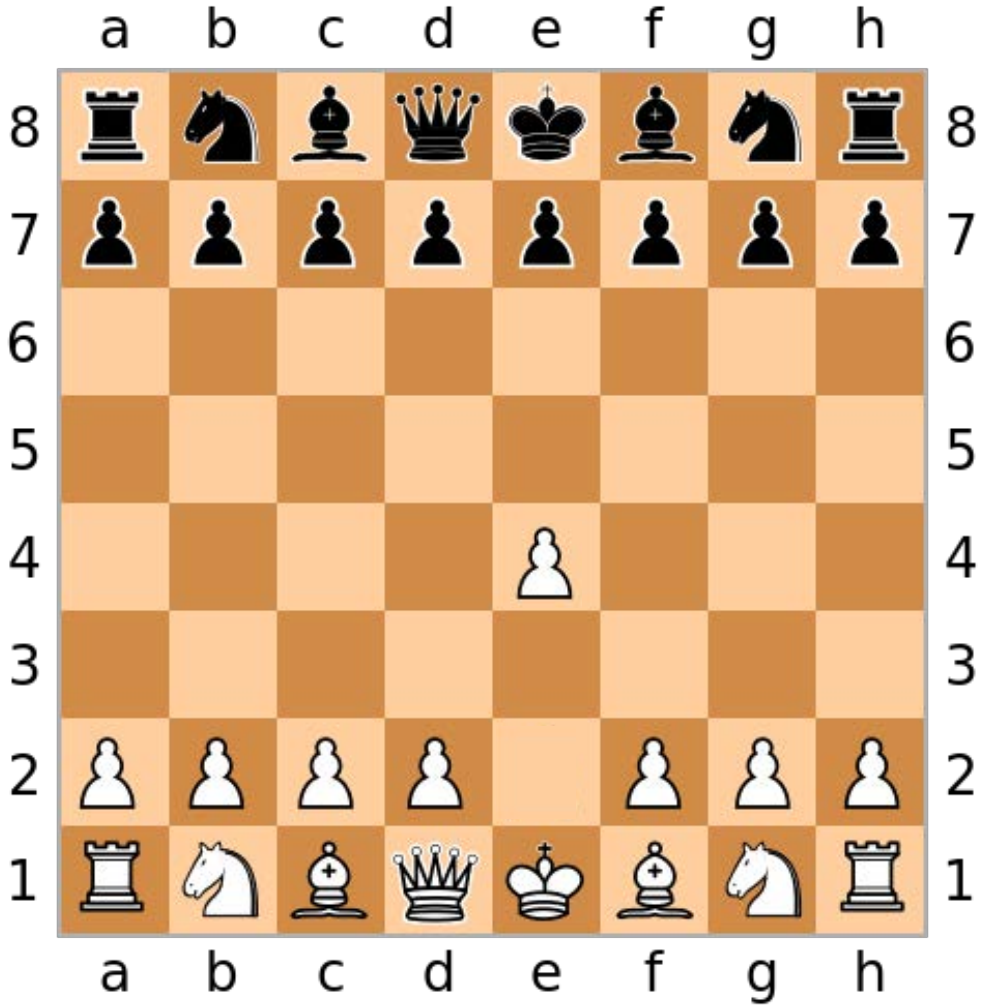


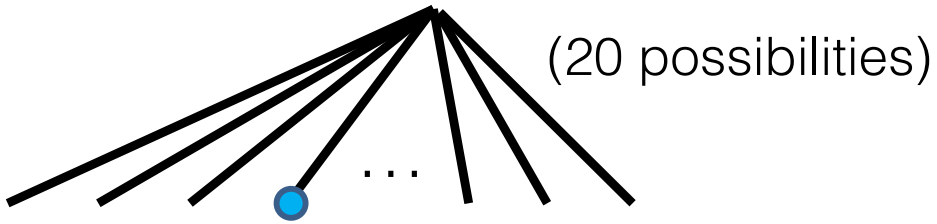
Chess



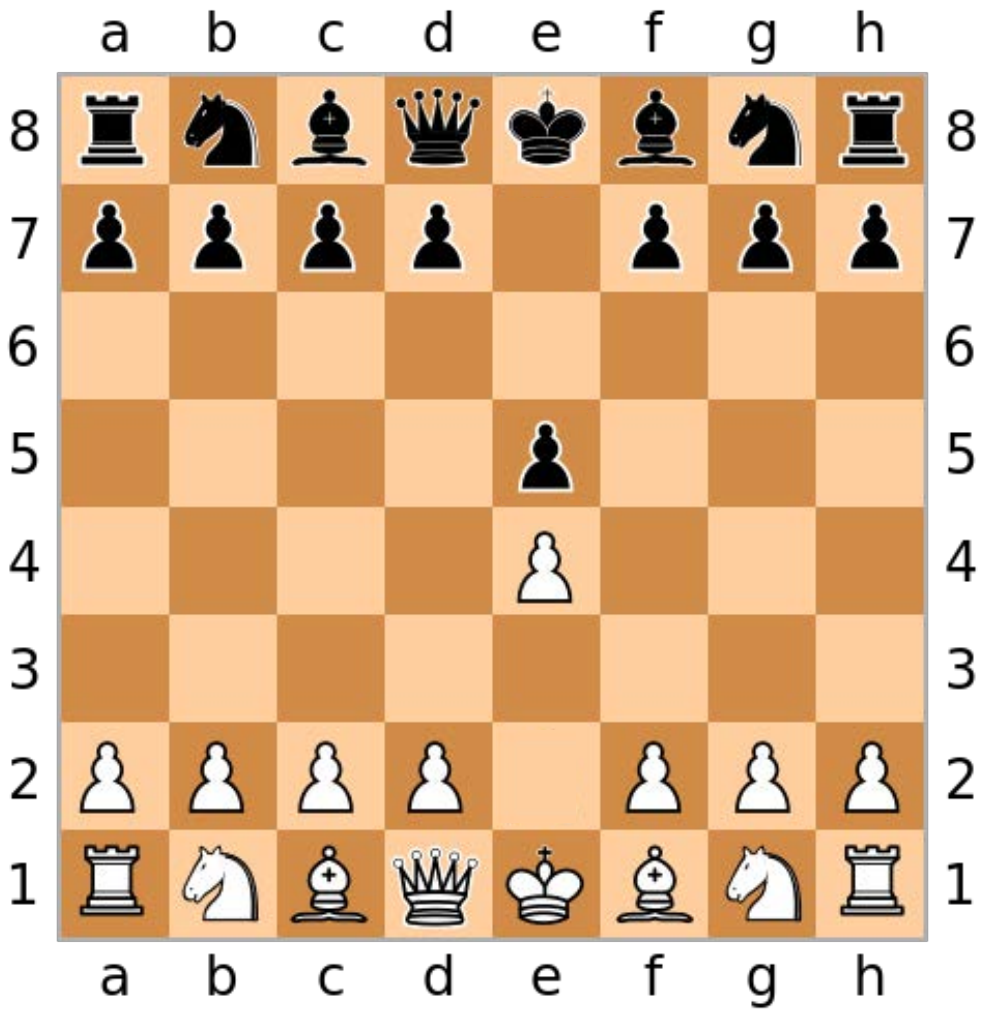


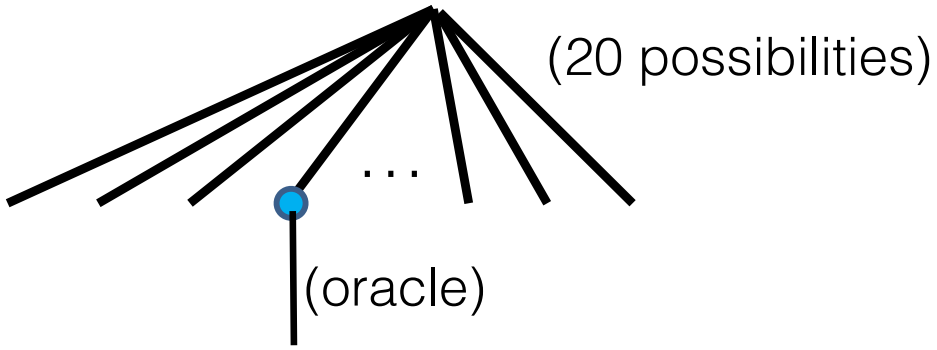
Chess



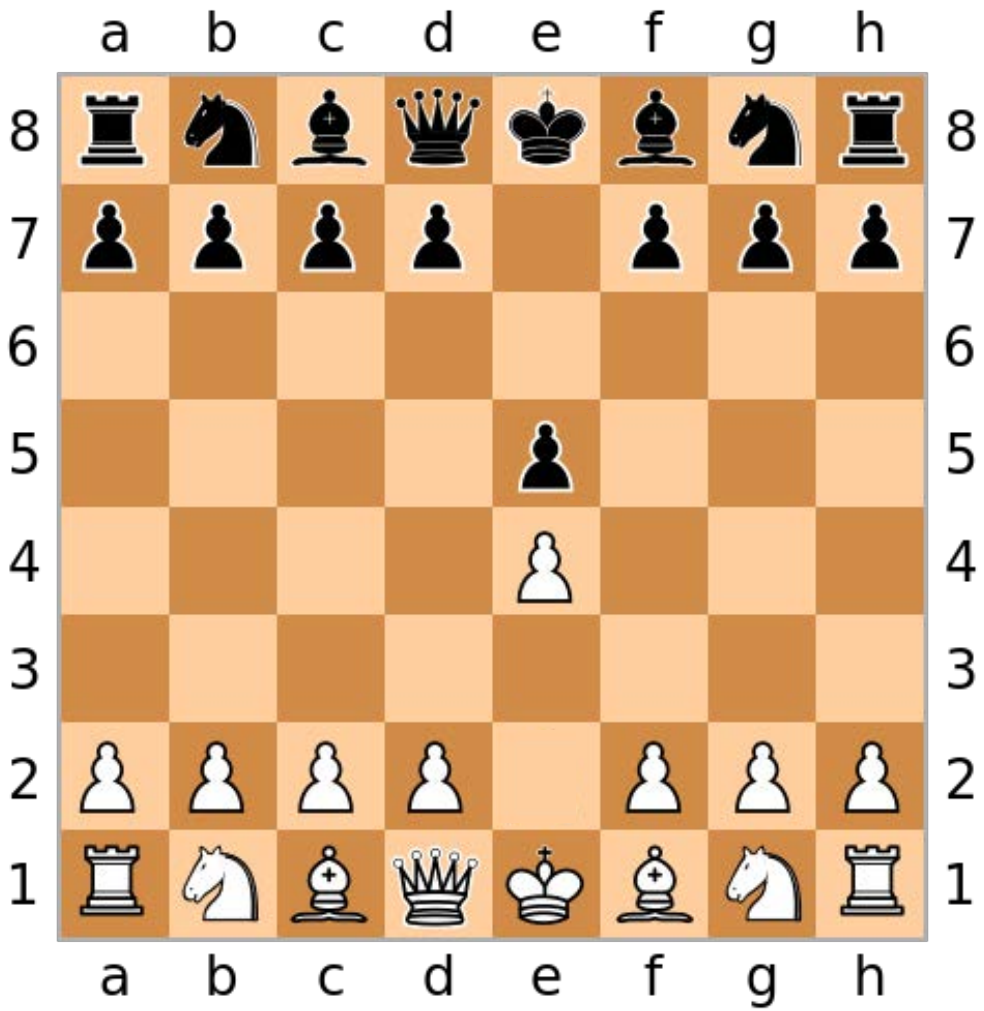


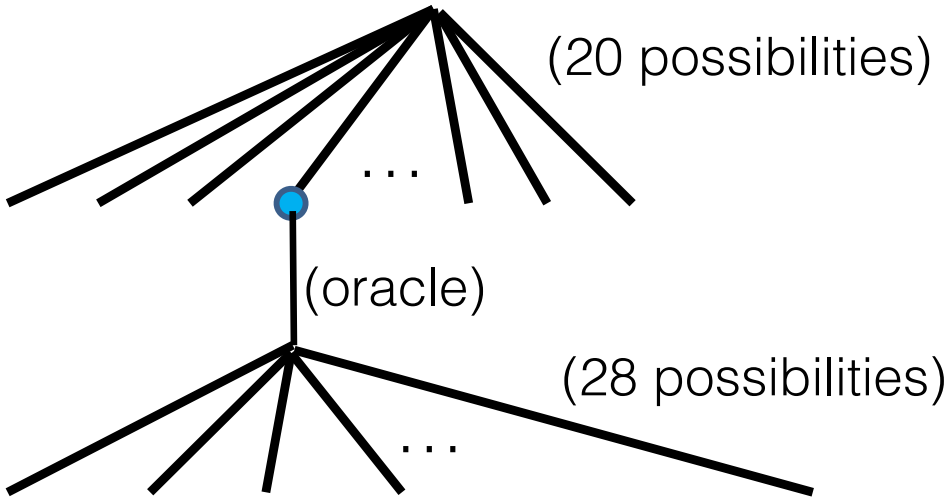
Chess



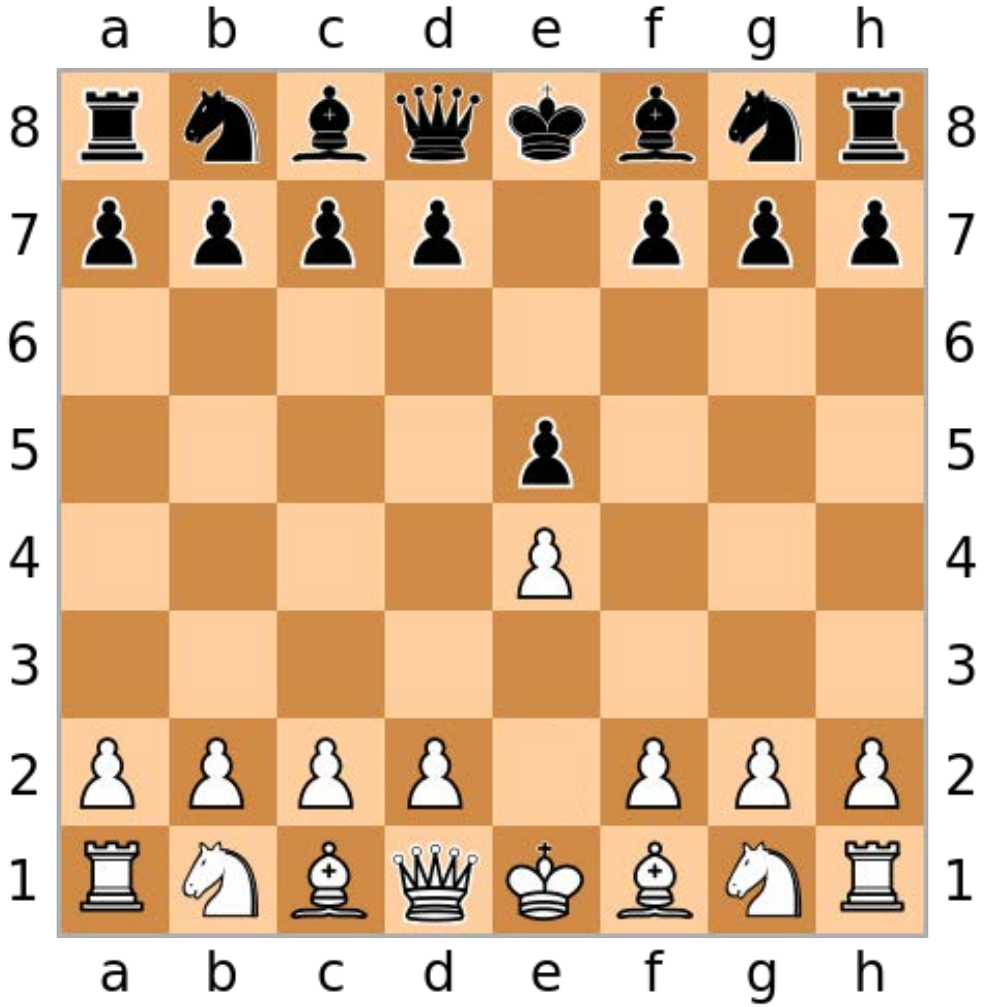


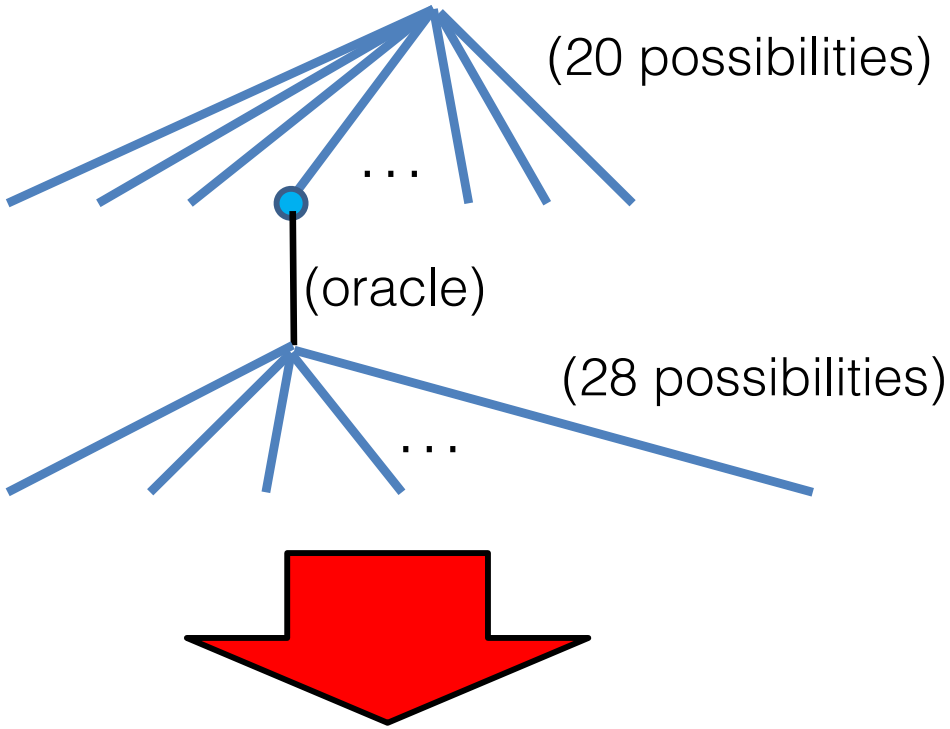
Chess


a b c d e f g h
8 7 6 5 4 3 2 1


(20 possibilities)
...
(oracle)
(28 possibilities)
...

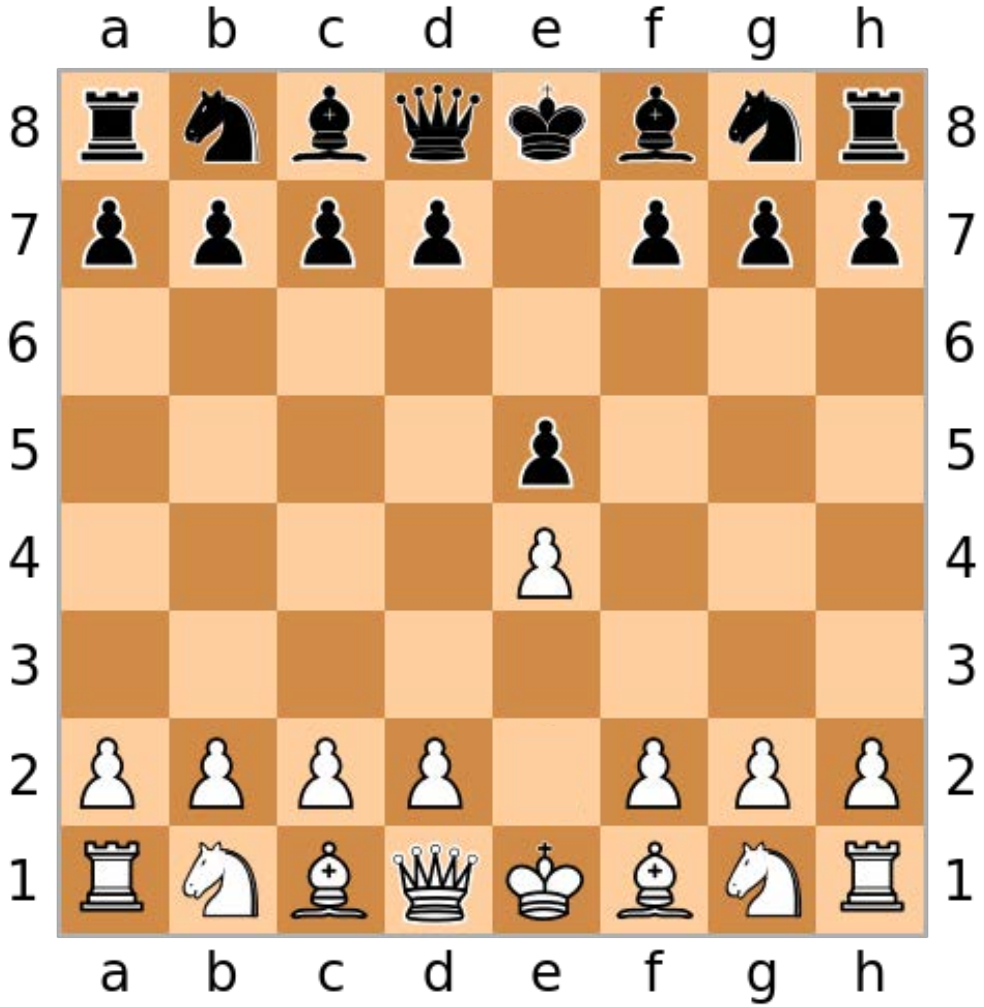
Chess

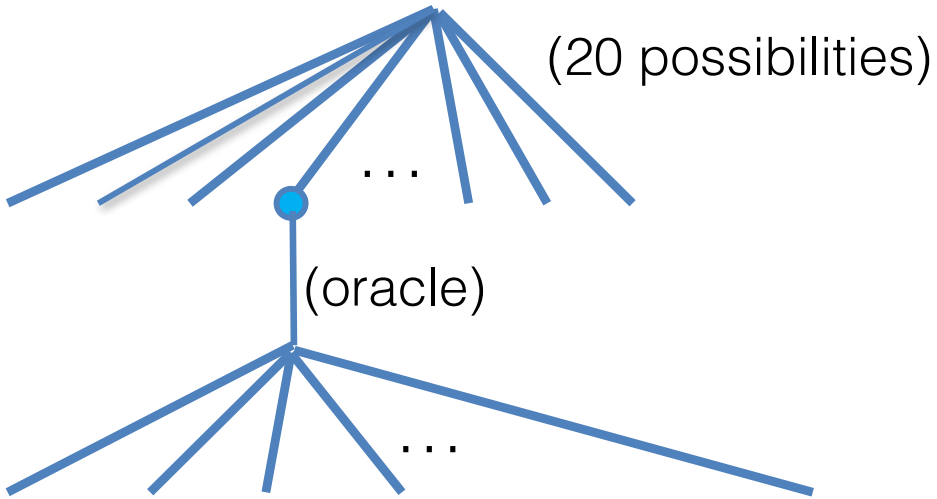




Exploring the whole tree game until we find a winning path is not doable: There are more than 10^{120} possible games!

Chess





Classic solution:

1) Explore the tree of possible games down to some depth.

2) Use a 'proxy' function P(s) to predict who will win, given the state s of the board at that depth.

Proxy Function *P*(*s*)

Given the state s of the board, the proxy function P(s)

- should be positive if the Whites are likely to win;
- should be negative if the Blacks are likely to win;
- Its absolute value should increase with the confidence.

How to build such a function?

Building the Proxy Function (1)

First, introduce 'Features' $f_i(s)$ based on expert knowledge such as:

- $f_h(s)$ = 1 if the White Queen is still alive, 0 otherwise;
- $f_i(s)$ = 1 if the Black Queen is still alive, 0 otherwise;
- $f_j(s)$ = 1 if the White Bishop #1 is still alive, 0 otherwise;
- $f_k(s)$ = number of possible moves for Black;
- $f_l(s)$ = $+\infty$ if Black is mate, $-\infty$ if White is mate, 0 otherwise.
- etc…

Building the Proxy Function (2)

Second, write P as a linear combination of the features:

$$P(s) = \sum_i \alpha_i f_i(s)$$

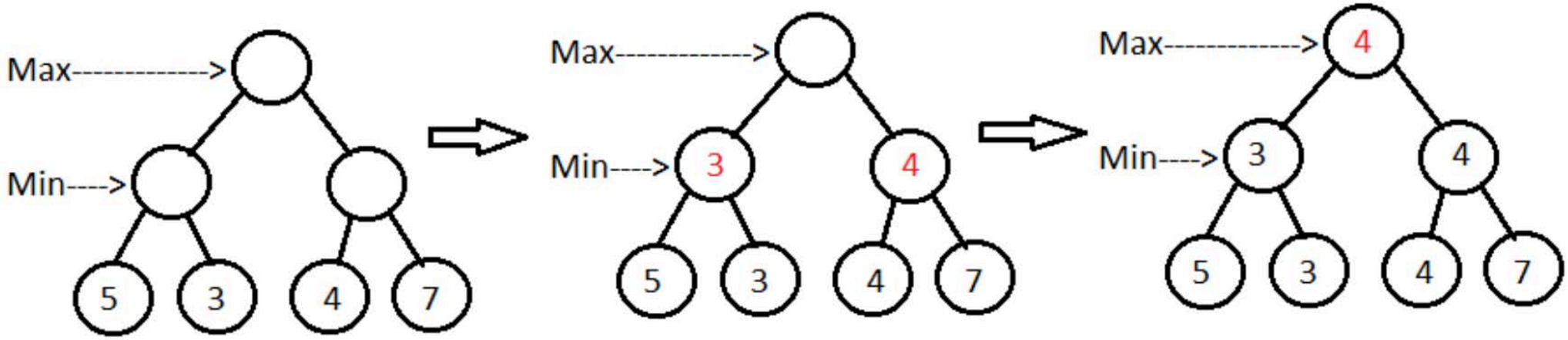
How can we learn the α_i weights?

→ Use many games from the history of chess to optimize.

This approach thus requires human expertise:

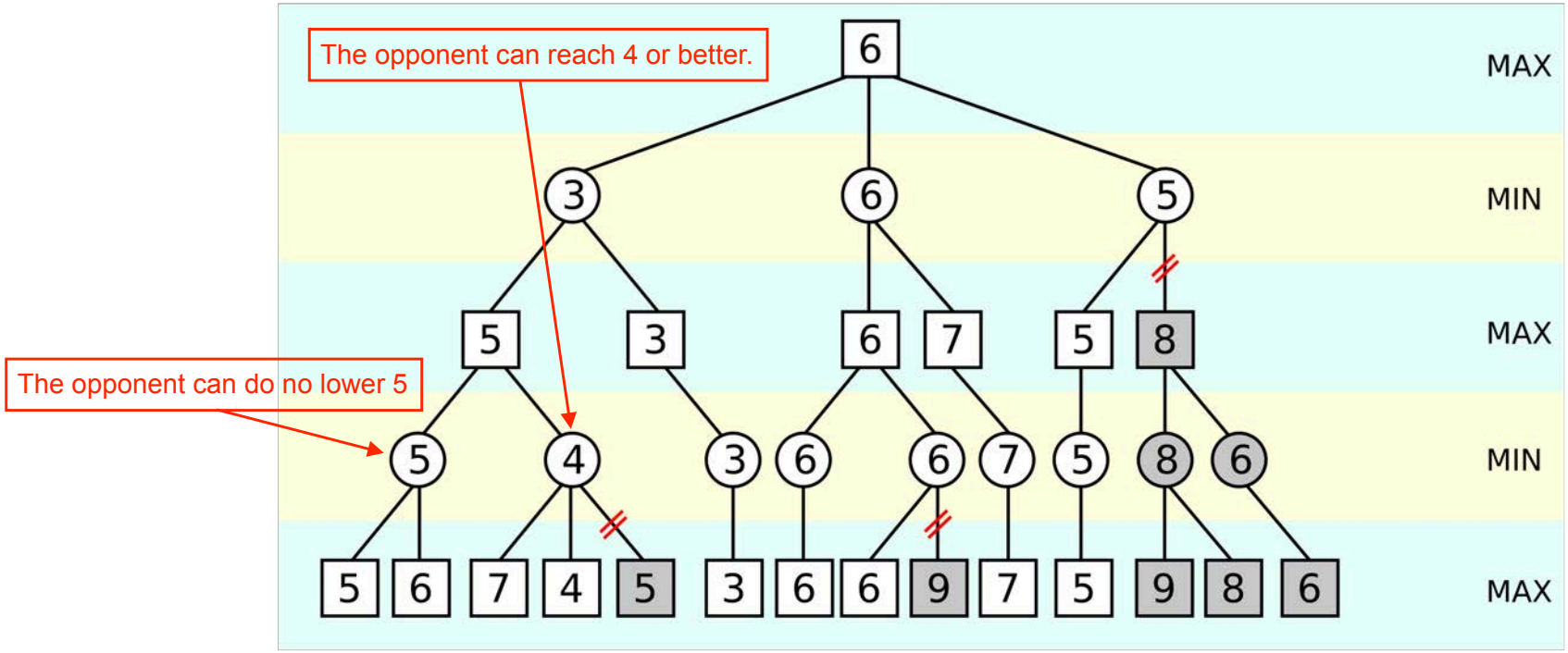
1. To design the features;
2. To estimate good weights for the features.

MinMax Algorithm



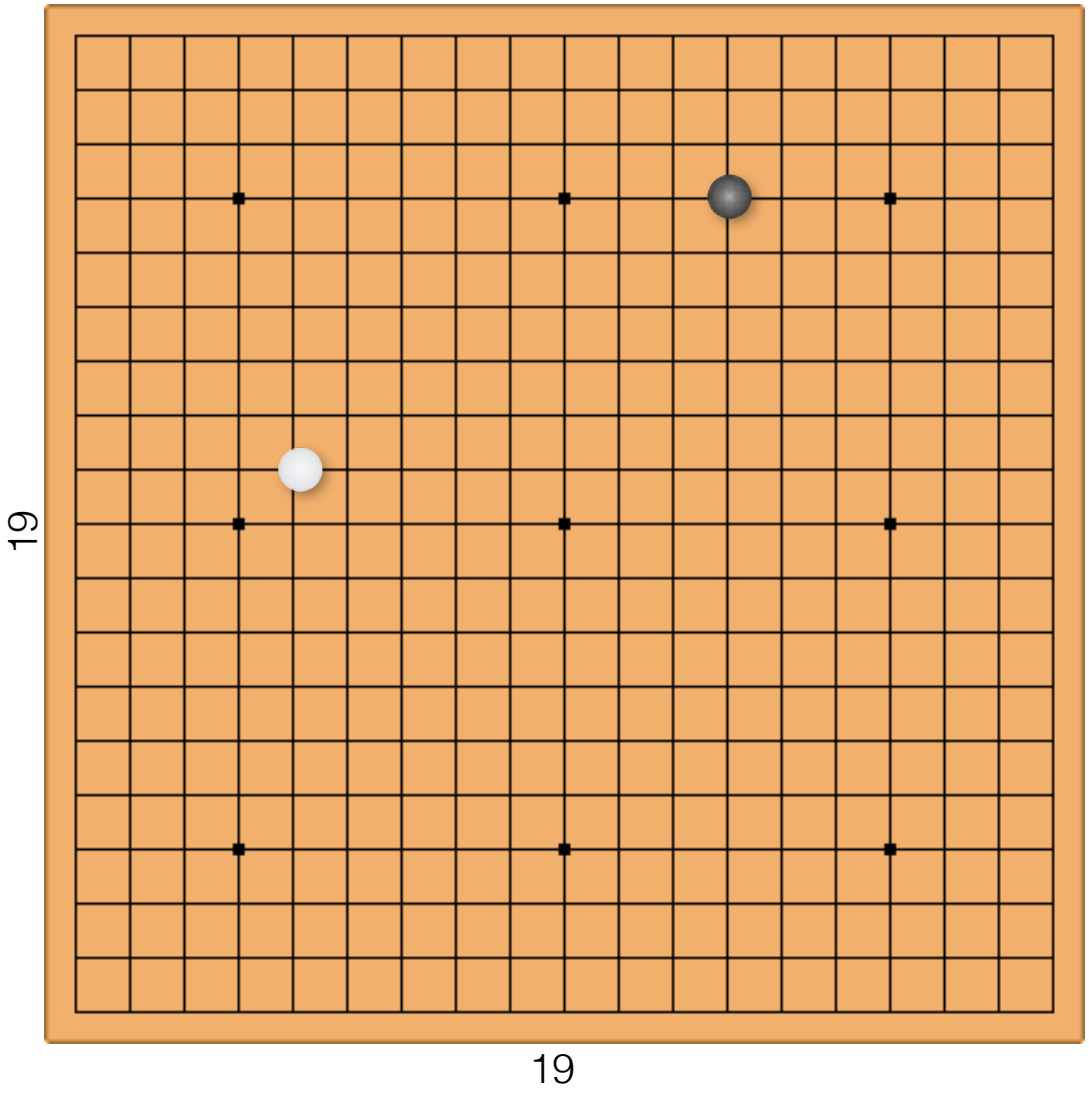
- The proxy function is evaluated after a set number of moves.
- To assign values and moves in the other nodes, one picks the min score for the adversary and the largest for oneself.
- This still requires a lot of computation even when building the tree only down to a certain level.
- Can be sped up using heuristics such as alpha-beta pruning.

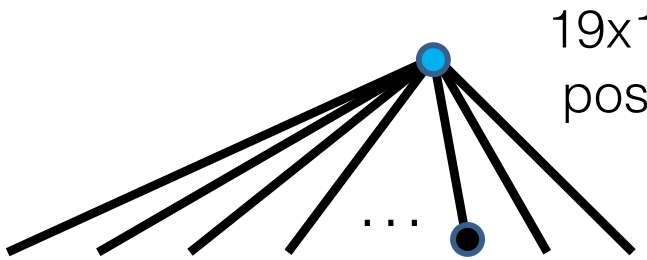
Alpha-Beta Pruning



- The max and min levels represent the turn of the player and the adversary, respectively.
- When traversing the tree from left to right, the grayed-out subtrees need not be evaluated, as they correspond to states no better than those that have been explored.
- This helps and makes it possible to play chess on a cellphone.

Go

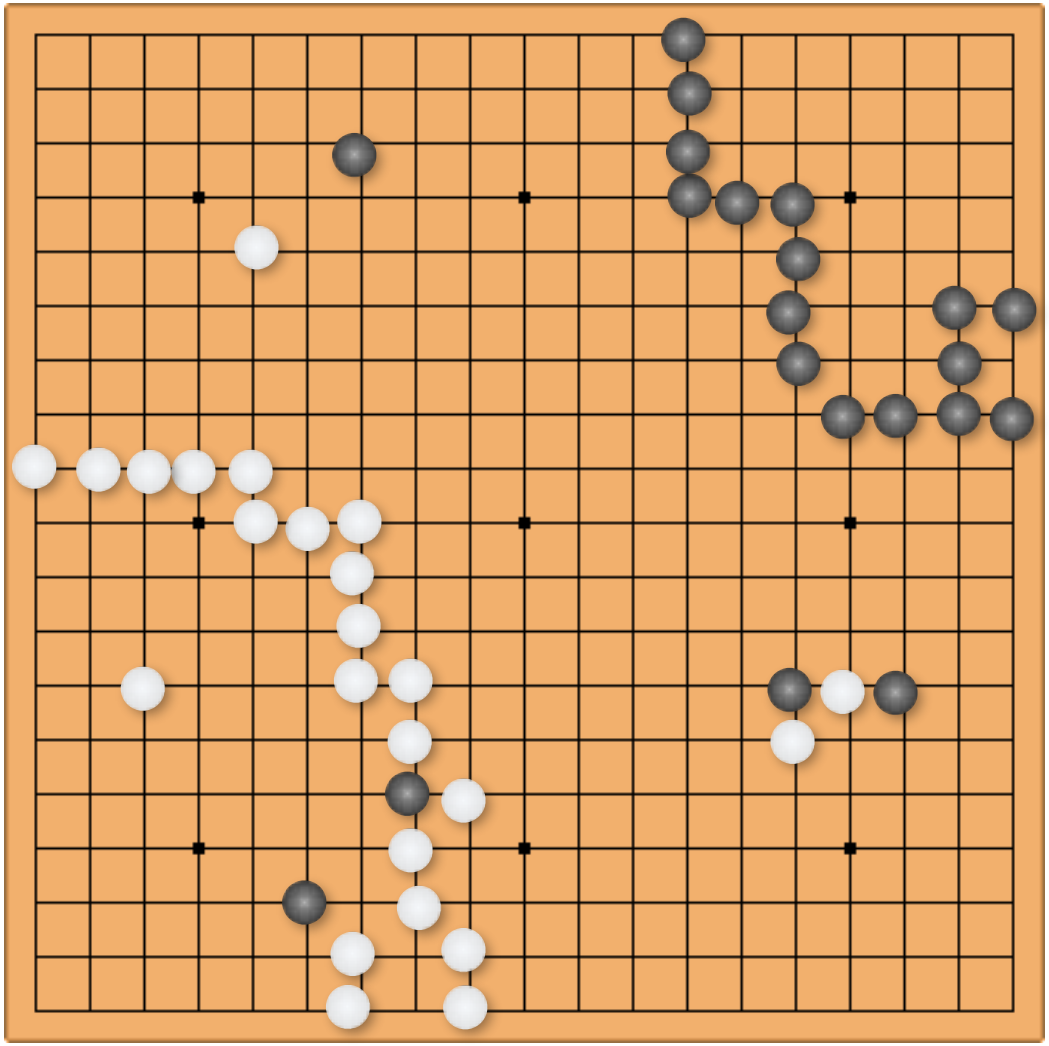




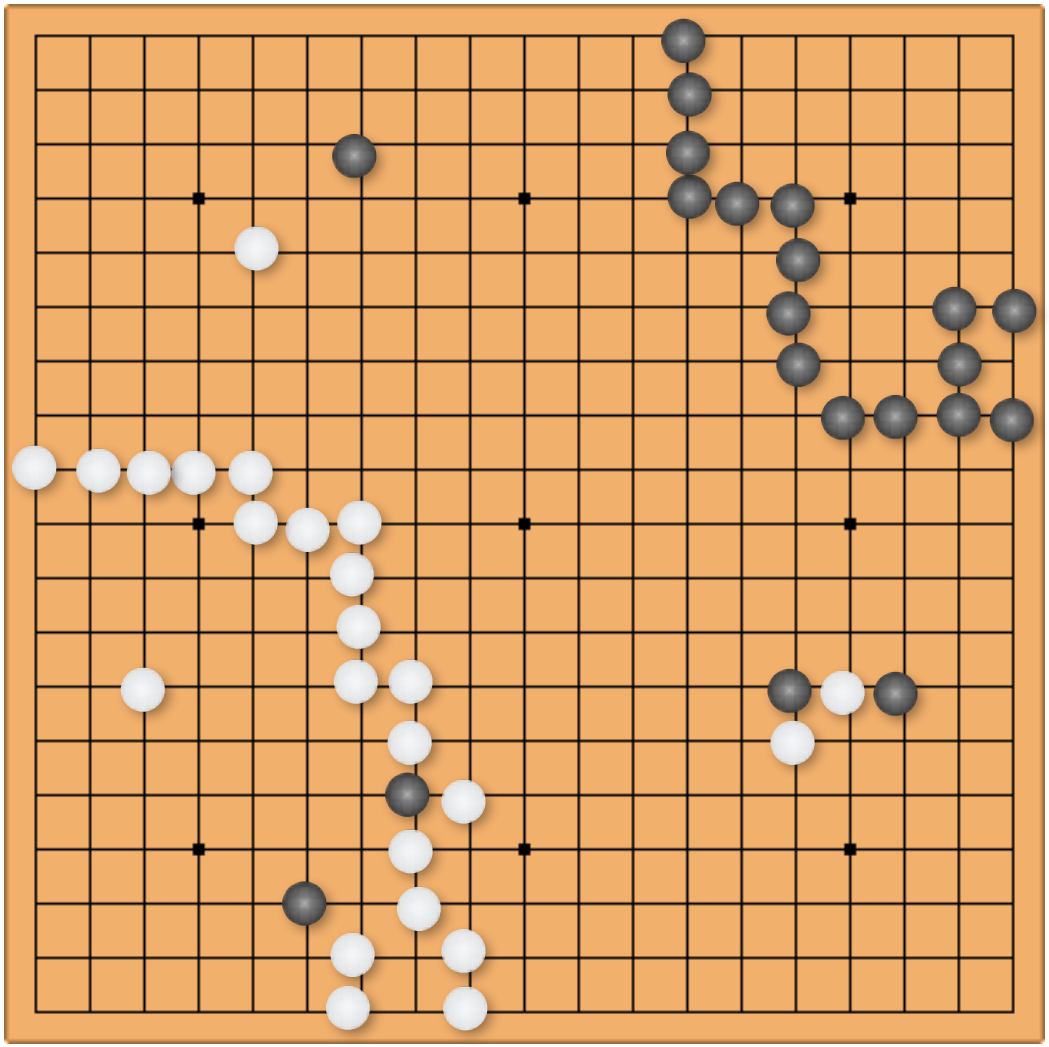
Estimated number of possible games: 10^{172} vs 10^{120} for chess.

—> Massively larger!

Goal: Building Territories

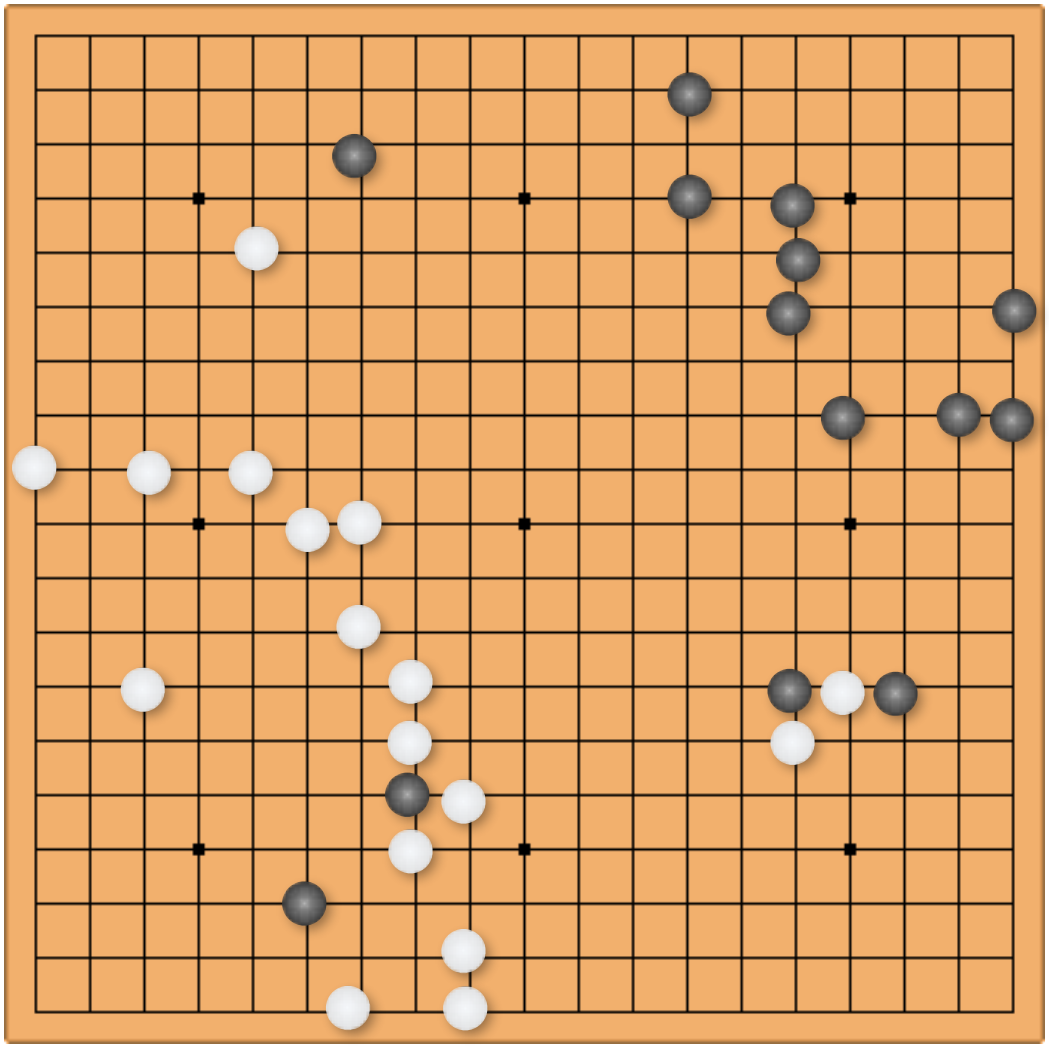


Proxy Function?



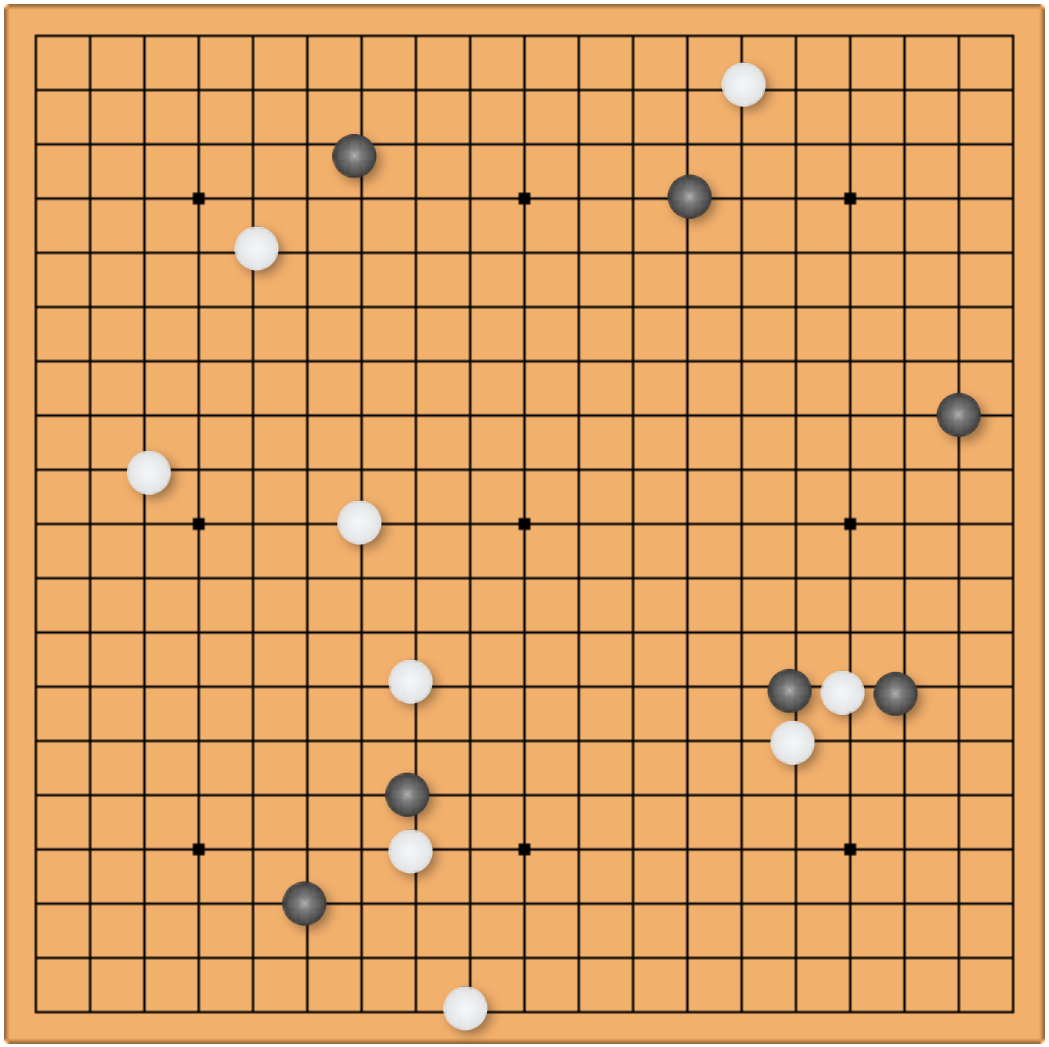
For this situation.

Proxy Function?



And this one.

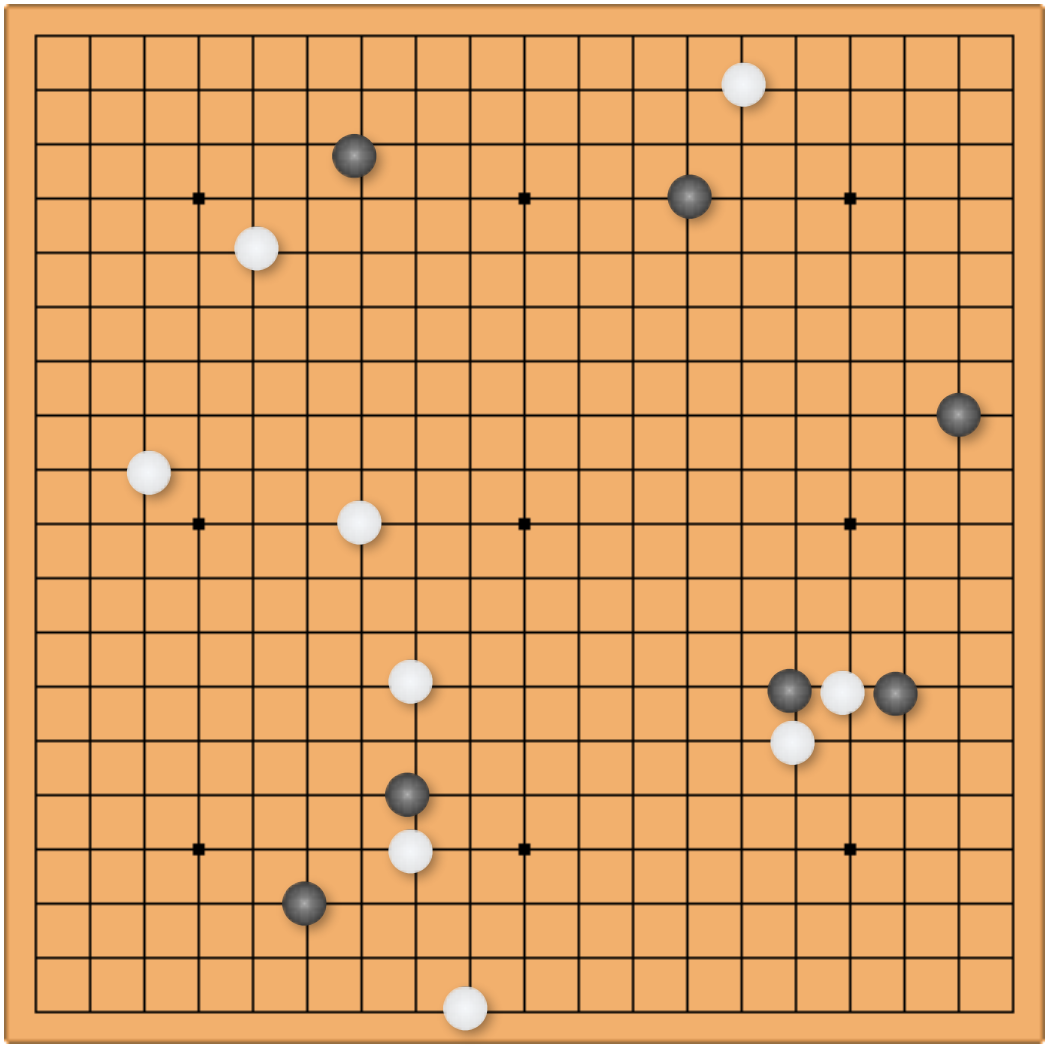
Proxy Function?



And that one.

- Very complex.
- Difficult to build a good proxy function.

The AlphaGo Zero Algorithm



Mastering the Game of Go without Human Knowledge
David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy Lillicrap, Fan Hui, Laurent Sifre, George van den Driessche, Thore Graepel, Demis Hassabis.
Nature, 2017.

Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm
David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, Demis Hassabis
Science, 2018.

Policy and Value

Let us assume we are given the function

$$f : \mathcal{S} \rightarrow \mathbf{p}$$

- $\mathcal{S}$ denotes the current state of the goban.
- $\mathbf{p}$ is the *policy,* that is, the vector of probabilities $\mathbf{p}_a$ that playing move **a** will lead to victory.

How should we use it?

- In theory, we could choose to play action $\hat{a} = \arg\max_a p(a)$.
- In practice, f is never perfect, and it is often better to explore many possible moves to decide on the next action.

Monte Carlo Tree Search

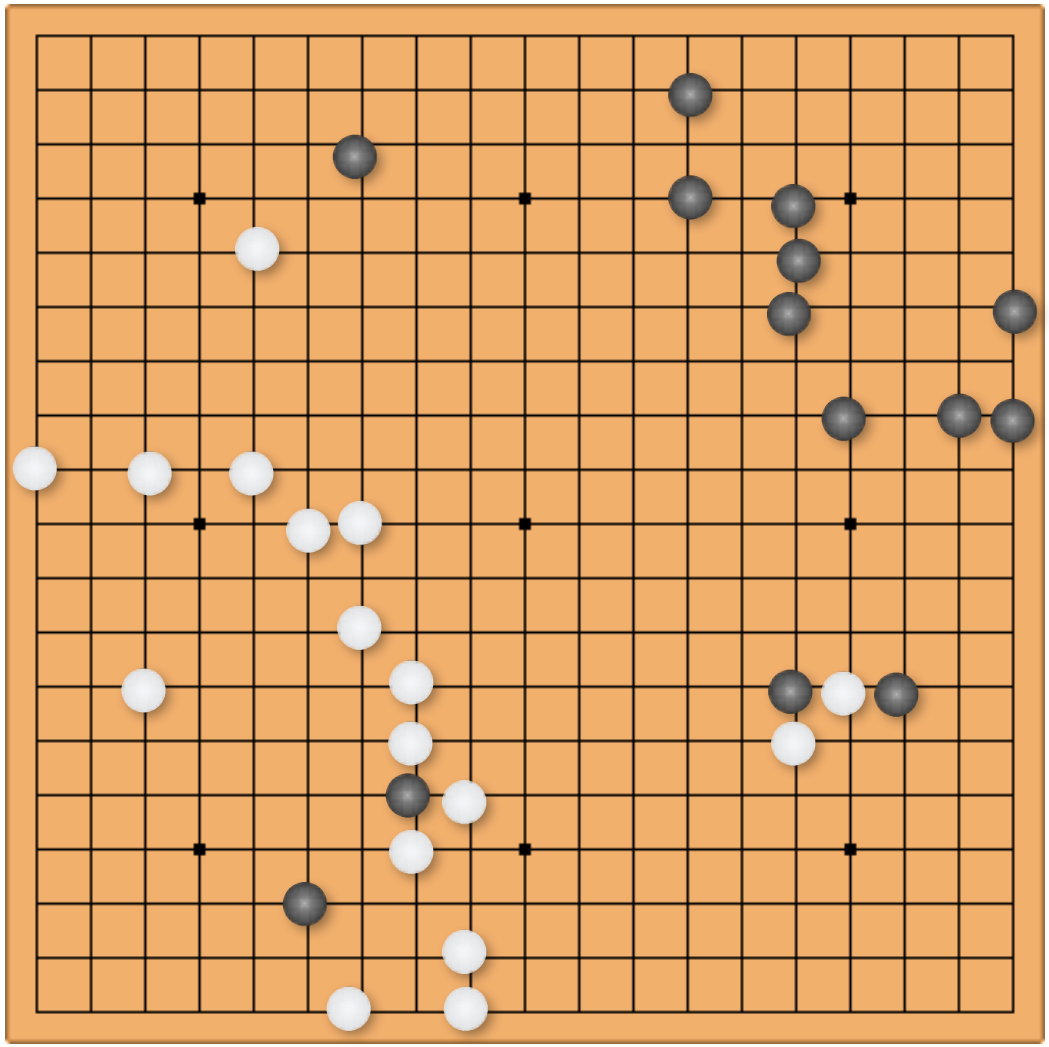
Allow the system to play against itself many times:

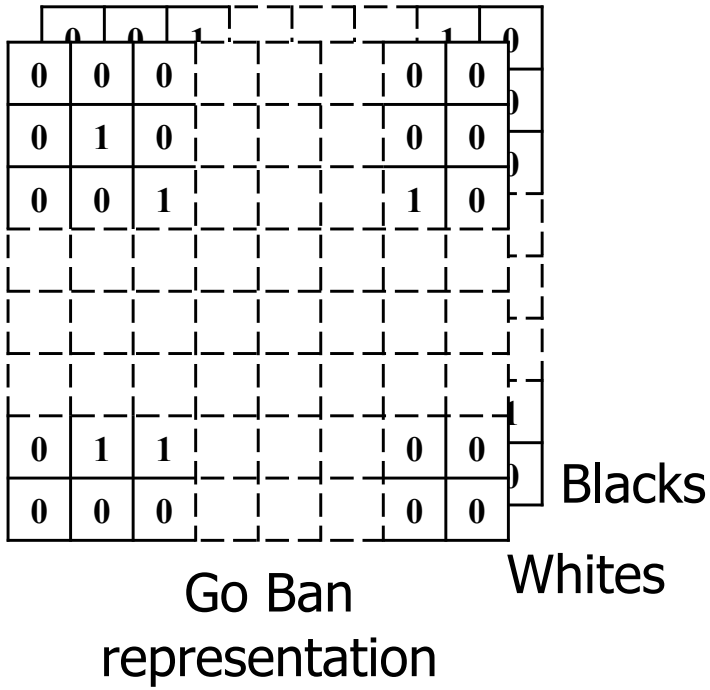
1. For each move, play according to the probabilities in $\mathbf{p} = f(\mathcal{S}, \Theta)$ and the number of times a node has been visited before.
2. Run each simulation *until the end of the game*. Keep track of how many times each board is visited along with wins and losses.
3. Update all $\mathbf{p}$ accordingly and record $(\mathcal{S}, \mathbf{p})$ in a set T.

➡ A large set $T = \{\ldots, (\mathcal{S}, \mathbf{p}), \ldots\}$ that can be used to retrain f.

➡An iterative algorithm that progressively improves the policies.

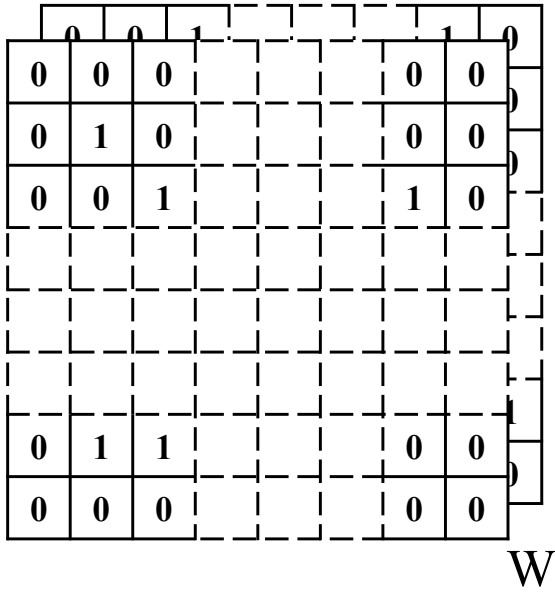
Implementing the Function F





- The state of the goban can be represented by two binary images, one for white and one for black.
- They can be used as input to a CNN.

CNNs to the Rescue

$$f\left(\right.$$  $$\left., \Theta\right) = (\mathbf{p}, \nu)$$

Blacks

Whites

- f is be implemented by a CNN with weights Θ.
- T is used to learn the weights.

Why Does this Work?

- If a move results in a win, the search is implemented in such as way that the corresponding board is more likely to be visited again.
- The policy updates are based on the number of times a node is visited and its probability will increase.

But

- If a board is initially given a low probability, it may never be explored even if it is a good one.
- Heuristics are required to prevent this kind of behavior.
- The real algorithm is more complex than what is shown on these slides.

Optional: Policy and Value Revisited

Let us assume we are given the function

$$f : \mathcal{S} \rightarrow \mathbf{p}, \nu$$

- $\mathcal{S}$ denotes the current state of the goban.
- $\mathbf{p}$ is the *policy,* that is, the vector of probabilities $\mathbf{p}_a$ that playing move **a** will lead to victory.
- ν is the probability that a policy p will lead to victory.

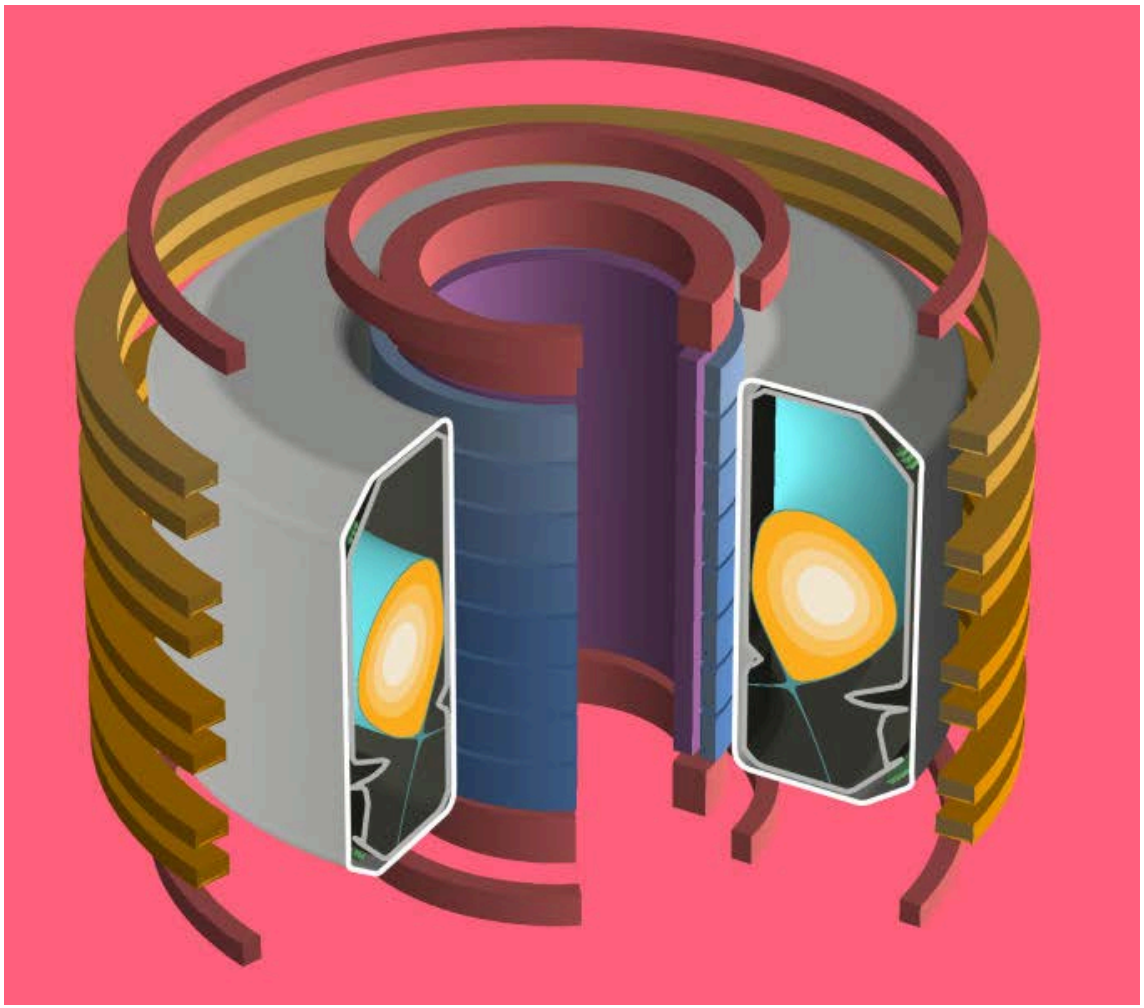
➡ It can be used to bias the MCTS algorithm and make it visit unexplored nodes.

Training Process

1. Initialize the parameters Θ of f randomly and play against itself.
2. Run many simulations and create a training set T.
3. Use T to retrain f.
4. Goto 2.

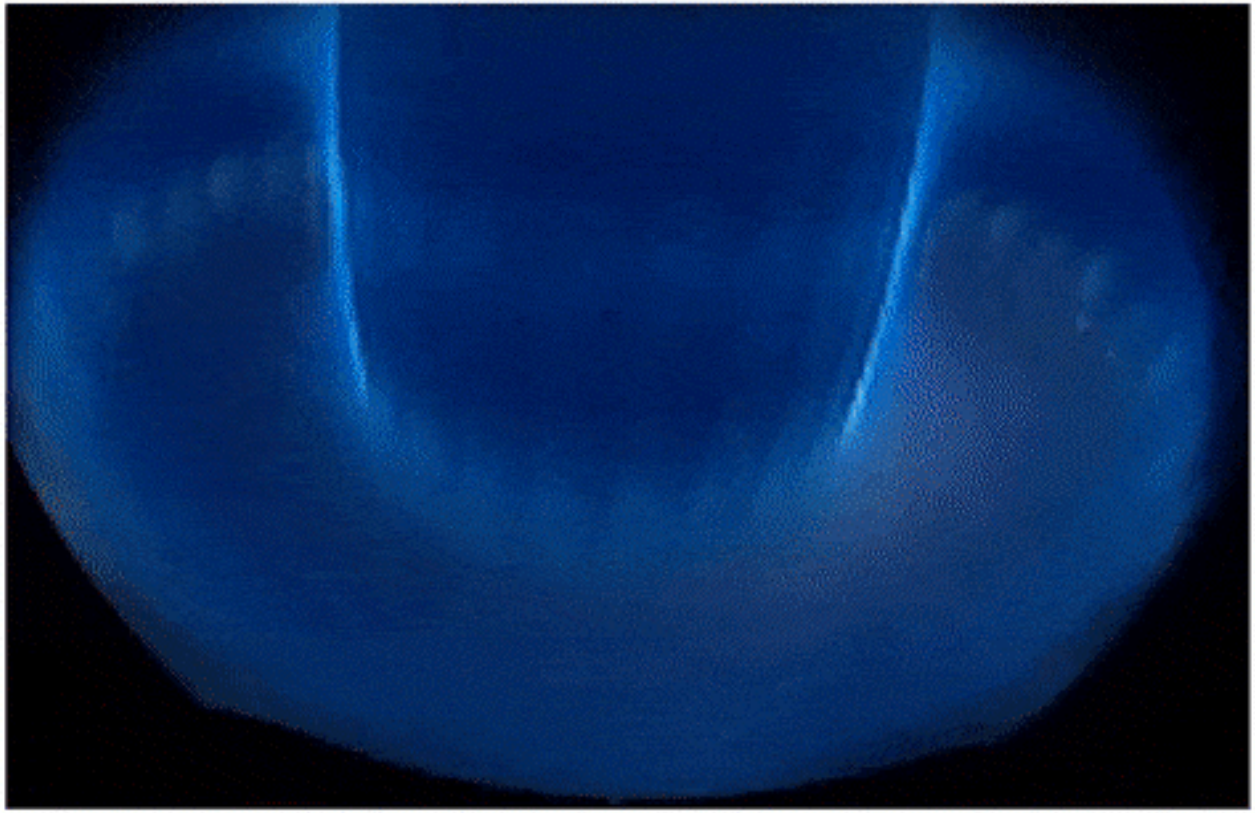
- Simple in principle but the devil is in the details.
- Requires immense amounts of computing power.

From Go to Nuclear Fusion

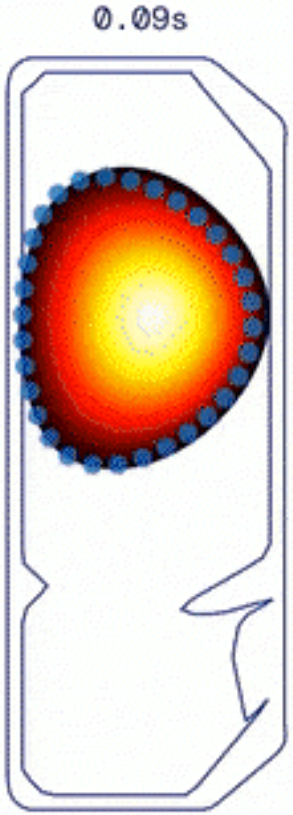


Variable Configuration Tokamak (TCV) at EPFL

Optional: Deep Control



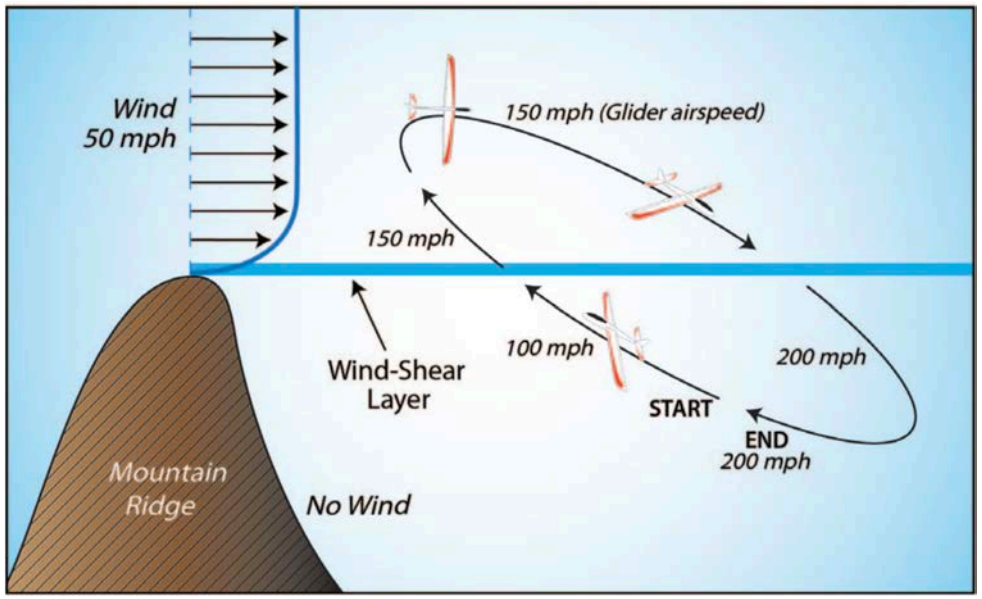
View from inside the tokamak

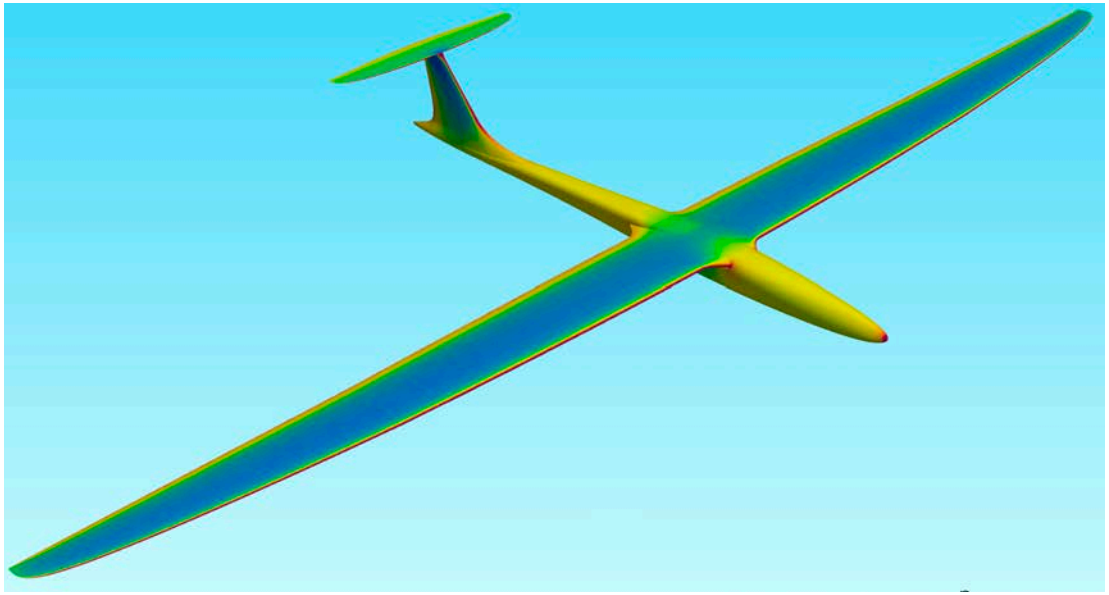


Plasma state reconstruction

- The policies involve driving the magnets in the Tokamak.
- They are learned from simulation data.

Optional: Dynamic Soaring





- We plan to design for ease of control.
- We will use dynamic soaring to prove the concept.

Reinforcement Learning

- The algorithm described on the previous slides tries many different policies and learns from its mistakes.
- This requires large numbers of simulations to cover the space of possibilities well.
- It is a generic paradigm that works well, but mostly on simulations and computer games so far.
- But the Tokamak experiment is very real …

See details in