Introduction to Two Model Free Reinforcement Learning

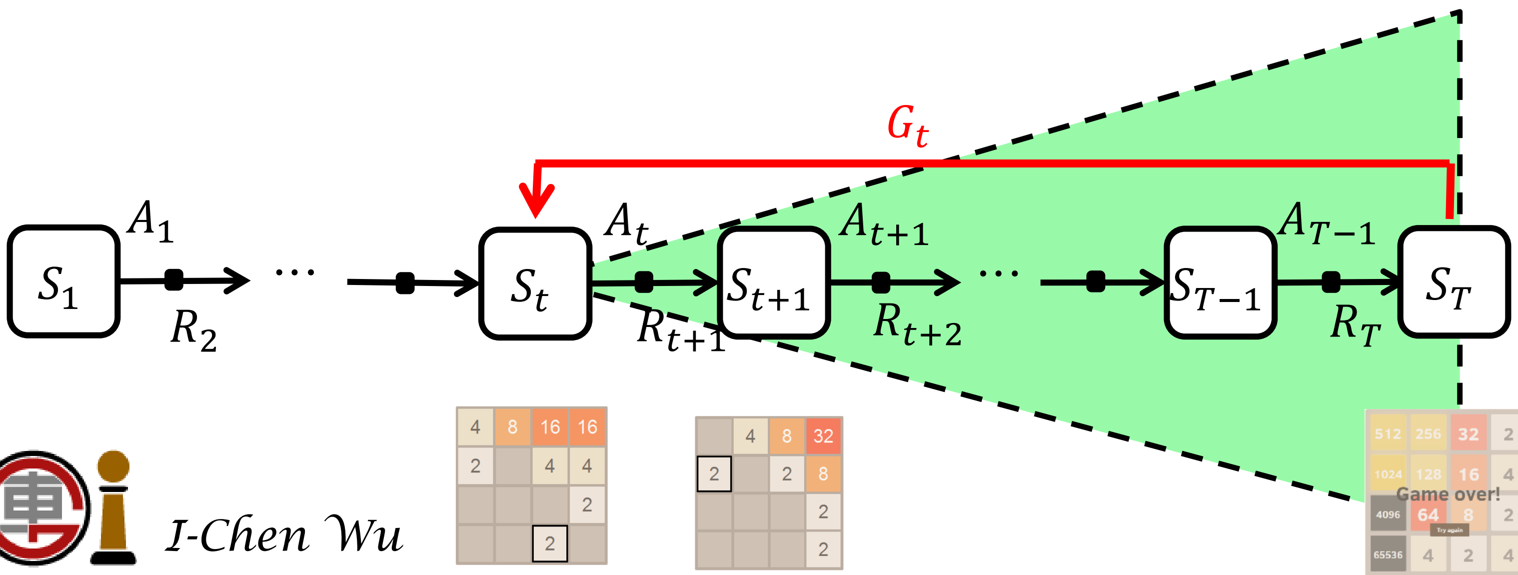
I-Chen Wu

Model-Free Reinforcement Learning

- Temporal Difference (TD) Learning
  - TD methods learn directly from episodes of experience
  - TD is model-free: no knowledge of MDP transitions / rewards
  - TD learns from incomplete episodes, by bootstrapping
  - TD updates a guess towards a guess
- Monte-Carlo (MC) Learning
  - MC methods learn directly from episodes of experience
  - MC is model-free: no knowledge of MDP transitions / rewards
  - MC learns from complete episodes: no bootstrapping
  - MC uses the simplest possible idea: value = mean return
  - Caveat: can only apply MC to episodic MDPs
    - All episodes must terminate
  - Monte-Carlo Tree Search (MCTS) is a successful one based on MC learning.

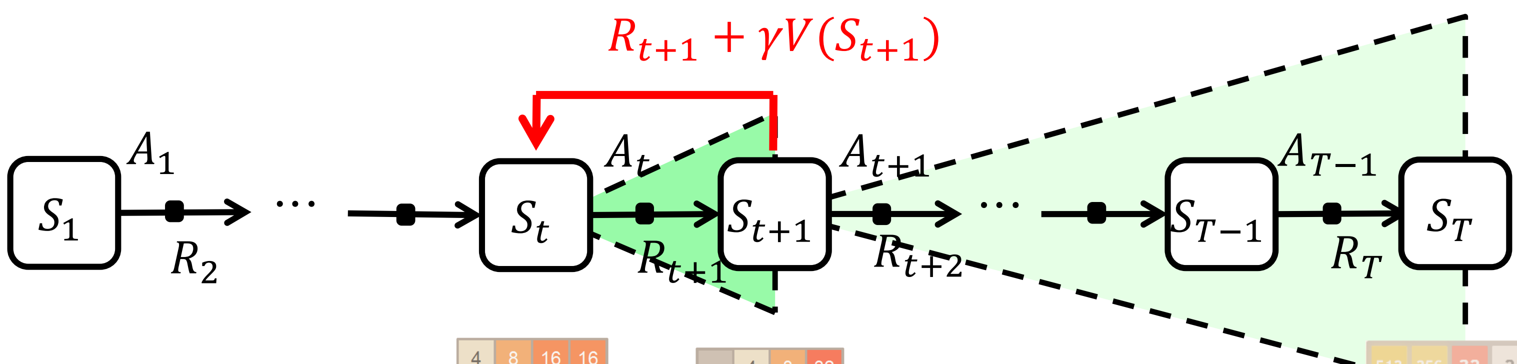
Monte-Carlo Learning

- Incremental Monte-Carlo
  - Update value $V(S_t)$ toward actual return G_t
    $V(S_t) \leftarrow V(S_t) + \alpha(G_t - V(S_t))$
  - α: learning rate, or called step size.
- Unbiased, but high variance.



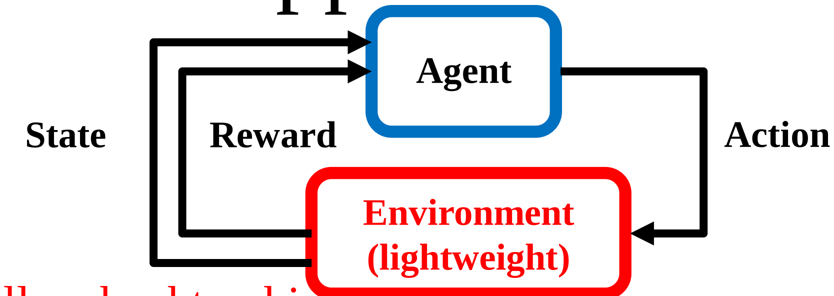
Temporal-Difference Learning

- Simplest temporal-difference learning algorithm: TD(0)
  - Update value $V(S_t)$ toward estimated return $R_{t+1} + \gamma V(S_{t+1})$
    $V(S_t) \leftarrow V(S_t) + \alpha(R_{t+1} + \gamma V(S_{t+1}) - V(S_t))$
  - TD target: $R_{t+1} + \gamma V(S_{t+1})$
  - TD error: $R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$
  - α: learning rate, or called step size.
- Biased, but lower variance

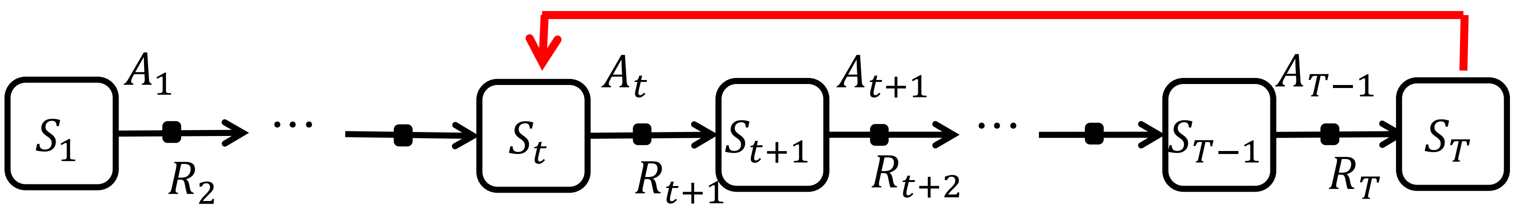


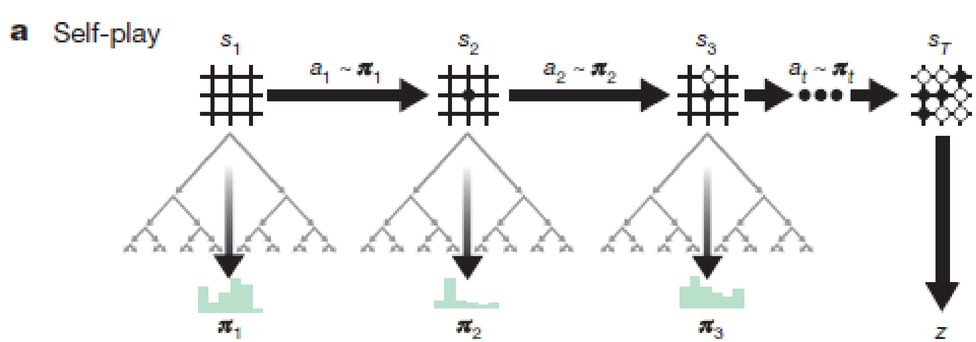
Application Classification of Deep Reinforcement Learning

Class 1: Lightweight-Model Applications

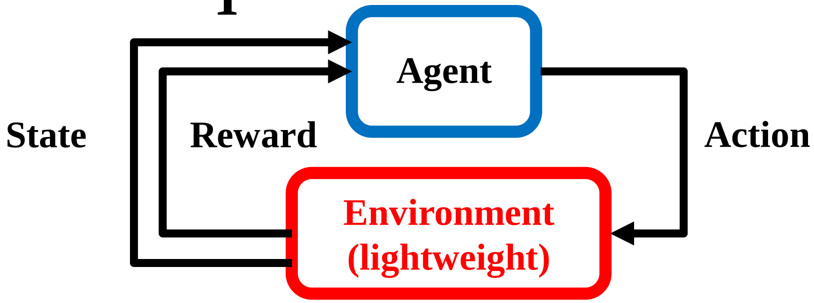


- Properties:
  - Model is well known or tractable
    - E.g., branching factor is limited.
  - Environments are simple to design, and allow backtracking
- Applications: Card/Board Games like Go, chess, etc.
- Possible Solutions: AlphaZero-like.





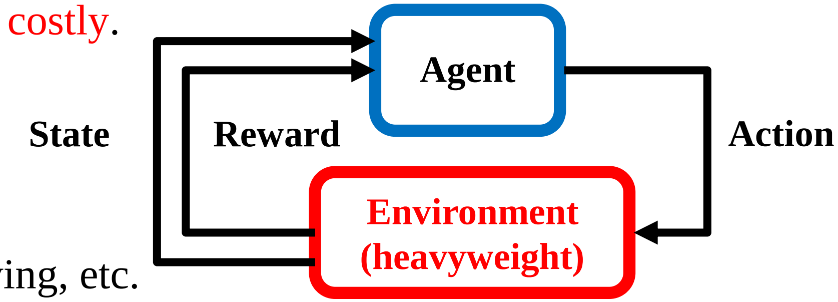
Related DRL Techniques

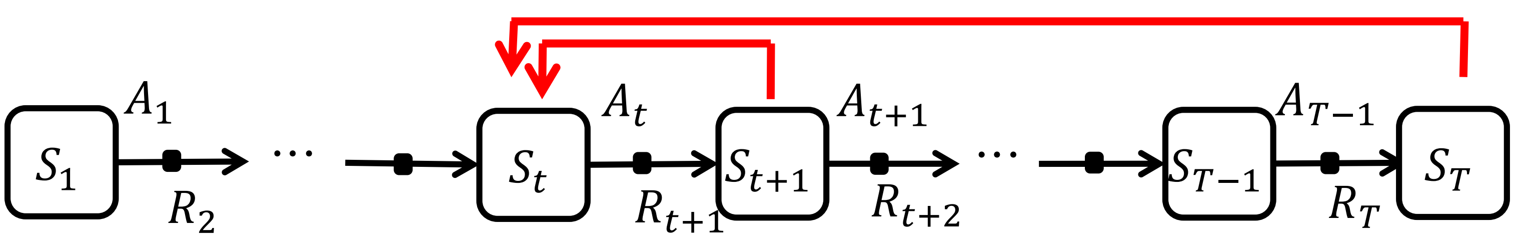


- Temporal Difference Learning
- Monte-Carlo Learning
- POMDP
- Monte-Carlo Tree Search (MCTS)
- AlphaGo/AlphaZero
- …

Class 2: Heavy-Weight-Model Applications

- Properties:
  - Model is well defined, but may be complex or intractable
    - E.g., environment dynamics are huge or continuous.
  - Simulators exist, but backtracking is hard and costly.
- Applications:
  - Video Games
  - ITM (intelligent traffic management)
  - Simulators for robots/drones/autonomous driving, etc.
  - Network resource allocation?
  - Mathematical optimization (like scheduling problems)?
- Related DRL Techniques (next pages)





Related DRL Techniques

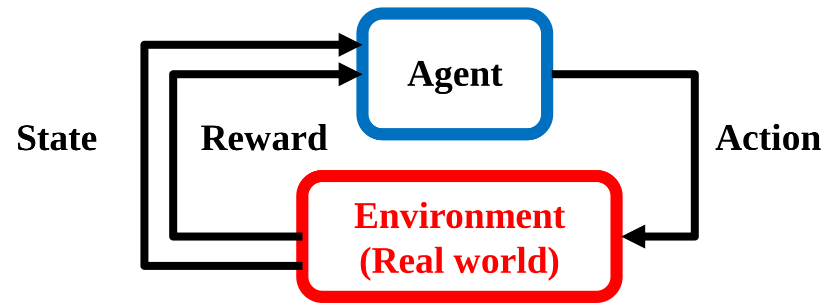
- Value-Based:
  - **DQN**
  - DDQN
  - Deuling Network
  - Bootstrapped DQN
  - Gorrila: Distributed DQN
  - MFEC: Model-free episodic control (like 2048)
  - NEC: Neural Episodic Control
  - D3QN: Double Deuling DQN
  - Rainbow: A mix with all kinds of value-based algorithms.
  - C51: a kind of distributional method
  - QR-DQN: a kind of distributional method
  - IQN: a kind of distributional method
  - FQF: a kind of distributional method
  - Ape-X DQN: a distributed method with n-step and Double Dueling
  - R2D2: Recurrent Replay Distributed DQN

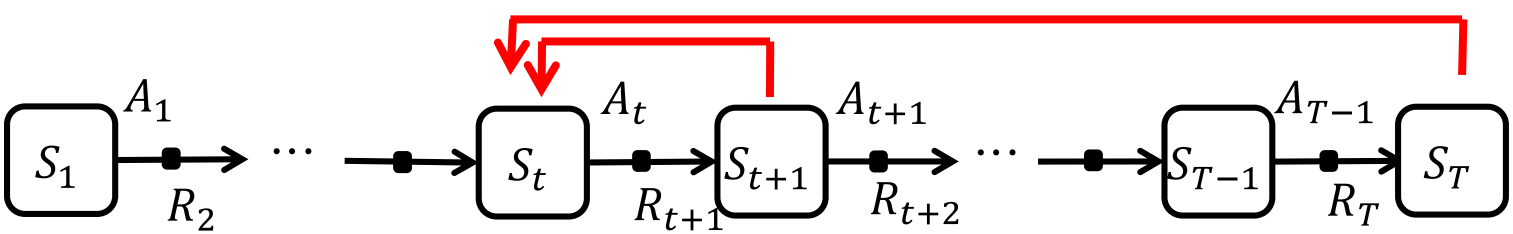
Related DRL Techniques

- Policy-based and Actor-Critic:
  - A3C: Asynchronous Advantage Actor-Critic
  - LASER: Off-Policy Actor-Critic with Shared Experience Replay (a kind of actor-critic that samples on-line sometimes)
  - ACER: Actor-Critic with Experience Replay
  - ACKTR: Actor Critic using Kronecker-Factored Trust Region (a kind of Natural Gradient)
  - TRPO: Trust-Region Policy Optimization
  - **PPO: Proximal Policy Optimization**
  - IMPALA: Importance Weighted Actor-Learner Architectures
- Miscellaneous:
  - NoisyNet and its variants
  - IDS: information directed sampling: Explore to the direction with information
  - RND: Random Network Distillation (for exploration)
  - NGU: Never Give up (for exploration; improving RND)
  - Agent57: Improve NGU
  - **muZero**

Class 3: Real-World-Model Applications

- Properties:
  - Model is unknown or too complex
  - Simulator does not exist or runs with expensive costs.
    - So, it is hard to produce a large data set.
- Applications:
  - Robots, Drones, Autonomous driving, etc.
- Related DRL Techniques:
  - Curriculum learning
  - Imitation Learning
  - Behavior Cloning
  - Transfer Learning (Sim2Real)
  - Meta Learning (one-shot/few-shot)
  - …



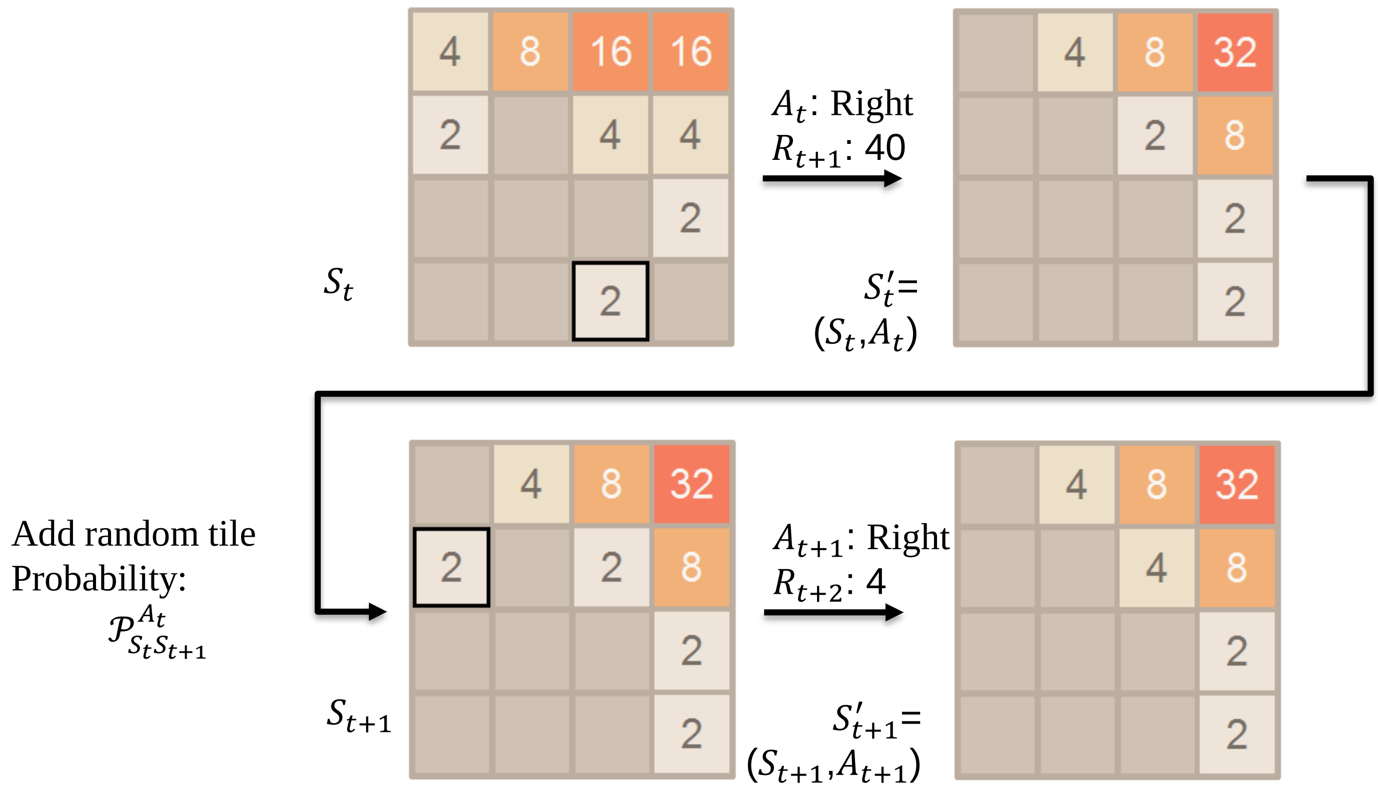


Reinforcement Learning for Lightweight Model

- Applications
  - 2048 (Temporal Difference Learning)
  - Go Programs (with Monte-Carlo Tree Search)

Case Study: 2048

- [Szubert et al., 2014; Yeh et al., 2016]



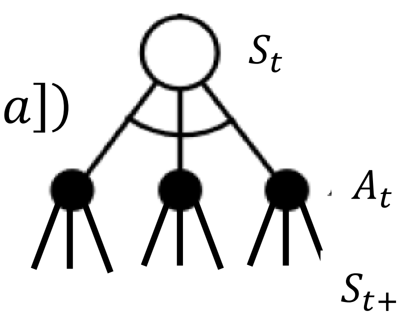
2048 RL Agent

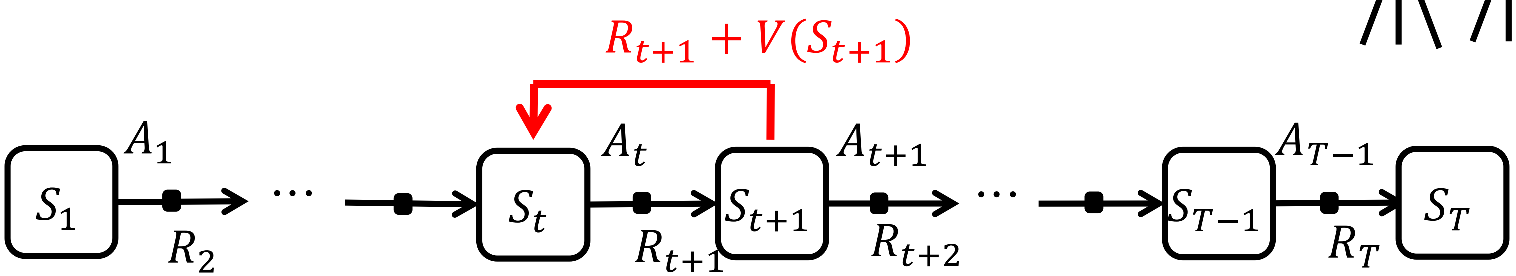
17 different numbers on each cell
And 4x4 (=16) cells in total.

- Value function:
  – The expected score/return G_t from a board S
  – But, #states is huge
    - About 17^{16} ($\cong 10^{20}$).
      – Empty (→ 0), 2 (=2^1 → 1), 4 (=2^2 → 2), 8 (=2^3 → 3), …, 65536 (=2^{16} → 16).
  – Need to use value function approximator.
- Policy:
  – Simply choose the action (move) with the maximal value based on the approximator.
- Model: agent's representation of the environment
  – After a move, randomly generate a tile:
    - 2-tile: with probability of 9/10
    - 4-tile: with probability of 1/10
  – Reward: simply follow the rule of 2048.

TD Learning in 2048

- Value function: (Normally $\gamma = 1$)
  - Update value $V(S_t)$ toward TD target $R_{t+1} + \gamma V(S_{t+1})$
    $V(S_t) \leftarrow V(S_t) + \alpha(R_{t+1} + \gamma V(S_{t+1}) - V(S_t))$
    - TD error: $R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$
- Making a decision (based on value).
  $\pi(s) = argmax_a(R_{t+1} + \mathbb{E}[V(S_{t+1}) \mid S_t = s, A_t = a])$
  - Problem: Less efficient upon making decision.





Q-Learning in 2048

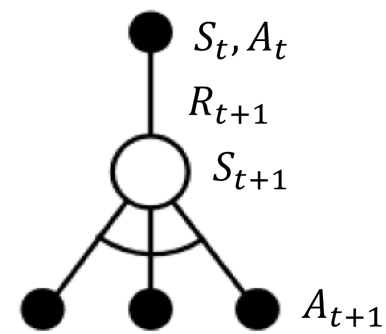
- Q-value function: (Normally $\gamma = 1$)
  - Update value $Q(S_t, A_t)$ toward TD target $R_{t+1} + \gamma \max_a Q(S_{t+1}, a)$

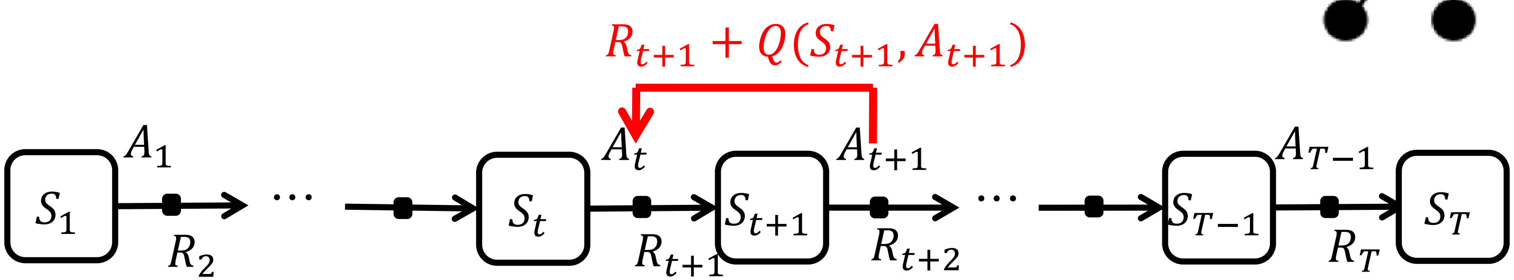
$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha\left(R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)\right)$$

- Making decision (based on value).

$$\pi(s) = argmax_a\big(Q(S_t, a)\big)$$

  - more efficient.
  - A minor problem: Four times more memory

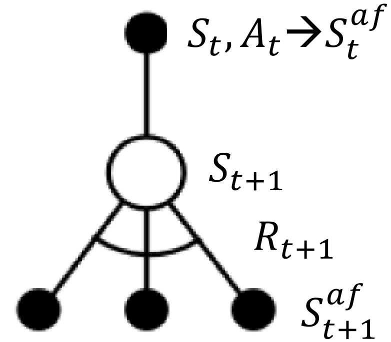




Afterstates in 2048

- Afterstate S_t^{af} is a state after action A_t at S_t.
  - Why not use S_t^{af} instead of (S_t, A_t)?
  - Note: in 2048, the reward R_{t+1} is not included in S_t^{af}.
- Afterstate value function: (Normally $\gamma = 1$)
  - Update value $V^{af}\left(S_t^{af}\right)$ toward TD target $R_{t+1} + \gamma \max_a(V^{af}\left(S_{t+1}^{af}\right))$

$$V^{af}\left(S_t^{af}\right) \leftarrow V^{af}\left(S_t^{af}\right) + \alpha(R_{t+1} + \gamma \max_a(V^{af}\left(S_{t+1}^{af}\right)) - V^{af}\left(S_t^{af}\right))$$

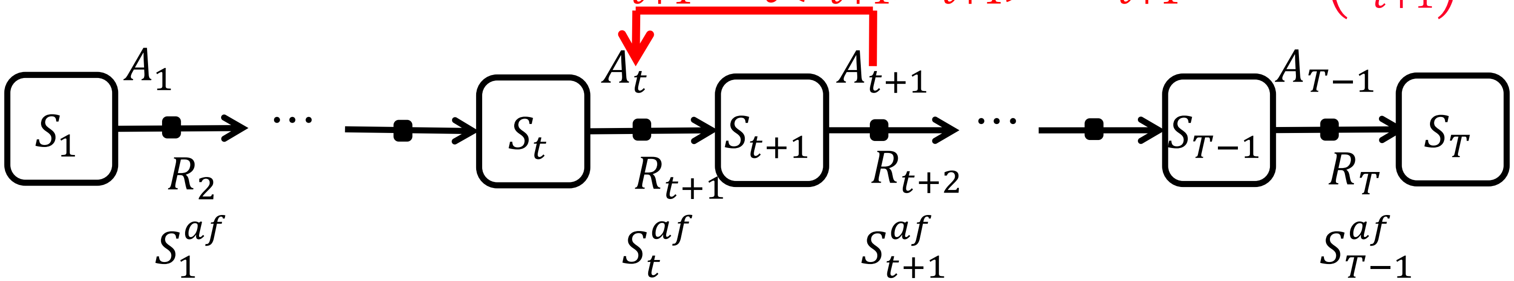


- Making decision (based on value).

$$\pi(s) = argmax_a\left(V^{af}\left(S_t^{af}\right)\right)$$

  - For simplicity, we use V, instead of V^{af}, if it can be applied to both.
  - More efficient.

$$R_{t+1} + Q(S_{t+1}, A_{t+1}) = R_{t+1} + V^{af}\left(S_{t+1}^{af}\right)$$



Value Function Approximation

- As mentioned above, #states is huge, so we need to use value function approximation.
  - Use a value function approximator, $\hat{v}(S,\theta) \approx V(S)$.
  - Simply use deterministic policy: $\pi(S) = argmax_a(\hat{v}(S,\theta))$
- But, what kind of value function approximator can we use?
  - What features can we choose?
    - Traditionally, # of empty cells, # of continuous cells, big tiles, etc.
  - Linear (like n-tuple network) vs. non-linear (like NN)
- n-tuple network is a powerful network for 2048.
  - Explore a large set of features.
  - Simplify operations by linear value function approximation.
  - Features in each network is one-hot vector.

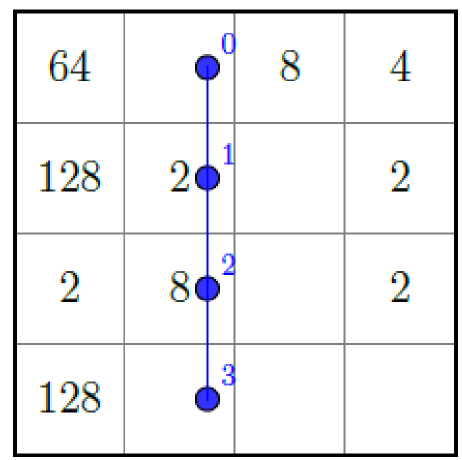
Gradient Descent

Now, how to do the update: $V(S_t) \leftarrow V(S_t) + \alpha \Delta V$

- Update value $V(S_t)$ towards TD target $y_t = R_{t+1} + V(S_{t+1})$
  $$\Delta V = (R_{t+1} + V(S_{t+1}) - V(S_t)) = (y_t - V(S_t))$$
  α: learning rate, or called step size.
  - Note: $\gamma = 1$ here.
- Objective function is to minimize the following loss in parameter θ. (note: $\hat{v}(S, \theta) = x(S)^{\mathrm{T}}\theta$)
  $$\mathcal{L}(\theta) = \mathbb{E}\left[\left(y_t - \hat{v}(S, \theta)\right)^2\right]$$
  $$\nabla_\theta \mathcal{L}(\theta) = \left(y_t - \hat{v}(S, \theta)\right) \cdot \nabla_\theta \hat{v}(S, \theta) = \Delta V \cdot x(S)$$
- Update features w: step-size * prediction error * feature value
  $$\theta \leftarrow \theta + \alpha \Delta V \cdot \frac{x(S)}{\|x(S)\|} \Rightarrow$$
  $$V(S_t) \leftarrow V(S_t) + \alpha \Delta V$$

N-Tuple Network

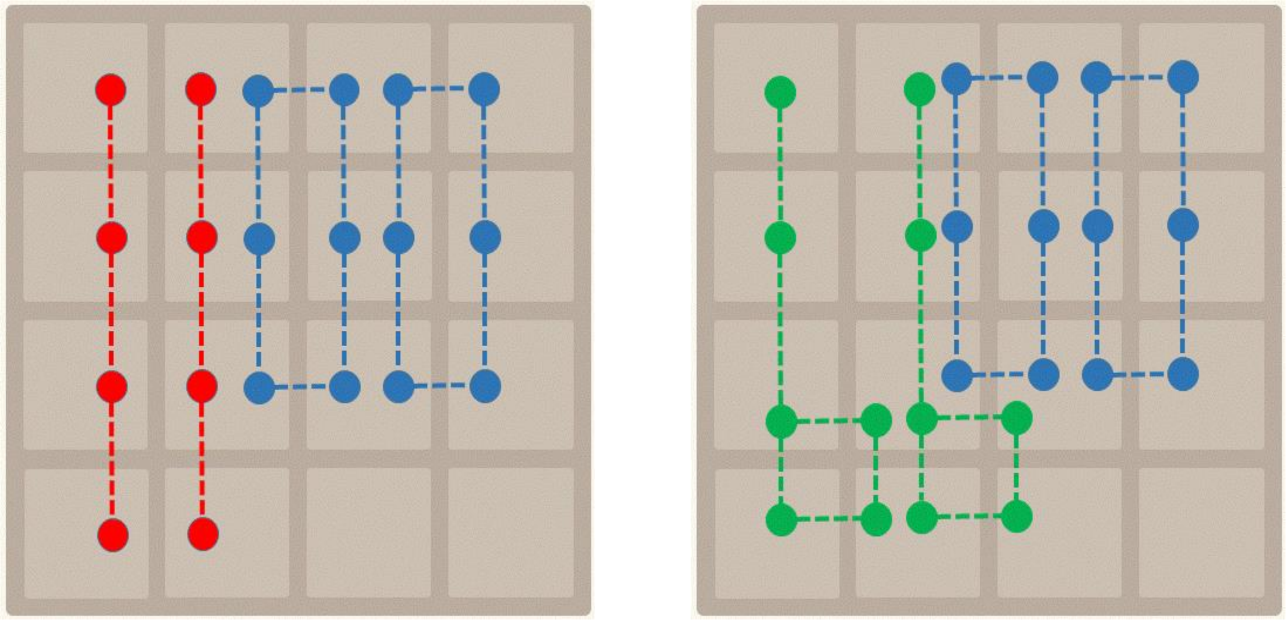
- Characteristics:
  - Provide with a large number of features.
  - Easily update.
- Example: 4-tuple networks as shown.
  - Each cell has 16 different tiles
  - 16^4 features for this network.
    - But only one is on, others are 0.
      - [..., 0, 0, 1, 0, 0, ...]
      - So-called one-hot vector.
    - So, we can use table lookup to find the feature weight.
  - Note: tabular RL is just like 16-tuple network in the case of 2048.



0123	weight
0000	3.04
0001	−3.90
0002	−2.14
⋮	⋮
0010	5.89
⋮	⋮
0130	-2.01
⋮	⋮

Other N-Tuple Networks

- Left: [Szubert et al., 2014]; Right: [Yeh et al., 2016]
- Some researchers even used 7-tuple network.



Update Features in N-Tuple Networks

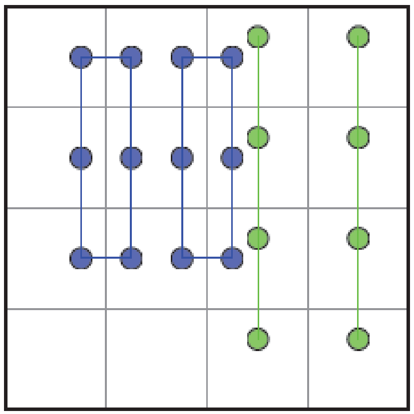
- For each n-tuple networks, simply update one weights.
- Features:
  - 8 x 16^4 features, $x(S) = [\underline{0, 1, 0, \ldots}, \underline{0, 0, 1, \ldots}, \ldots, \underline{1, 0, 0, \ldots}]$
    - All 0s, except for 8 ones.
      - One 1 every 16^4 features.
      - Let their indices be $\boldsymbol{s_1, s_2, s_3, s_4, s_5, s_6, s_7, s_8}$.
  - Only need to update $\alpha\Delta V$ at the features indexed by these indices.
  - Very efficient and fast.
- For k n-tuple networks,

$$\hat{v}(S,\theta) = x(S)^{\mathrm{T}}\theta = \sum_{j=1}^{n} x_j(S)\theta_j = \sum_{i=1}^{k} LUT_i[index(s_i)]$$

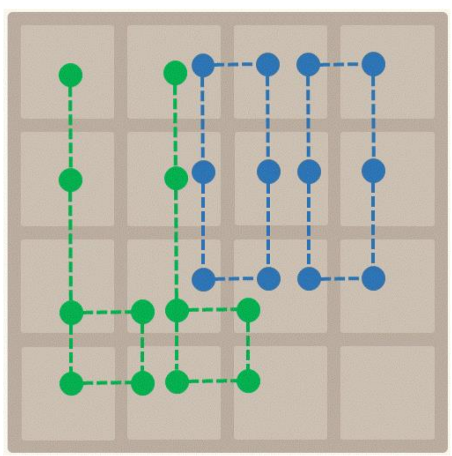
  - LUT_i: the i-th n-tuple network lookup table.
  - $index(s_i)$: The index in the i-th n-tuple network of state S.
- Update features w: step-size * prediction error * feature value
  - $\theta \leftarrow \theta + \alpha\Delta V \cdot x(S)$
  - Only need to update values θ_j with $\alpha\Delta V$ at $LUT_i[index(s_i)]$.

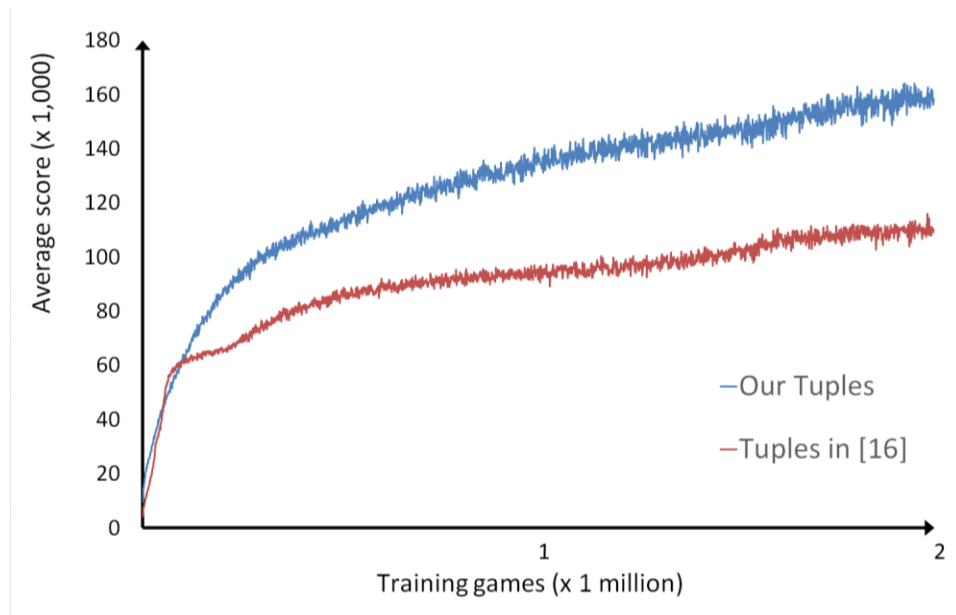
The N-Tuple Networks Used

- Use the following [Szubert and Jaskowaski 2014]



- Ours:

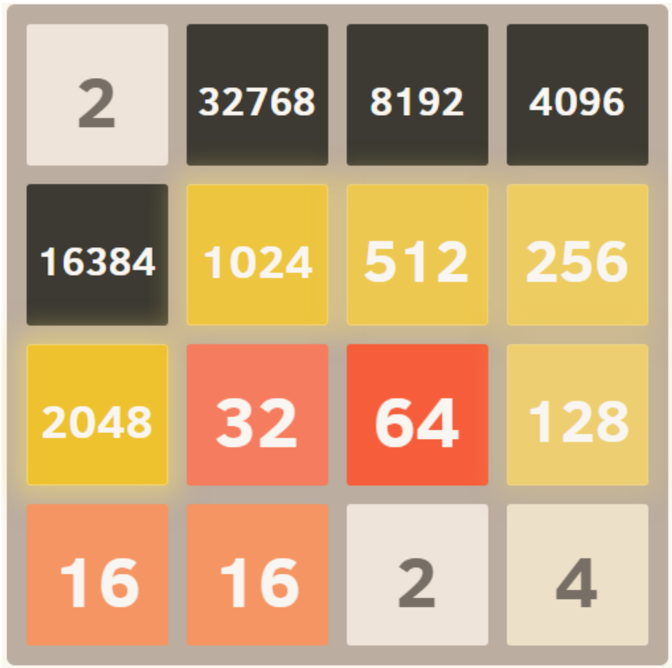


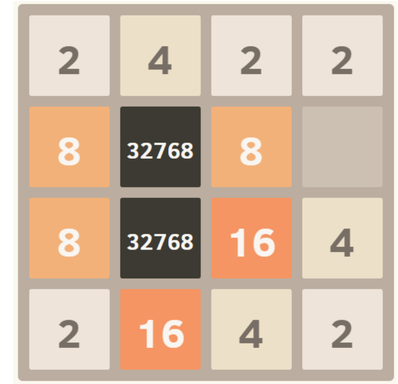


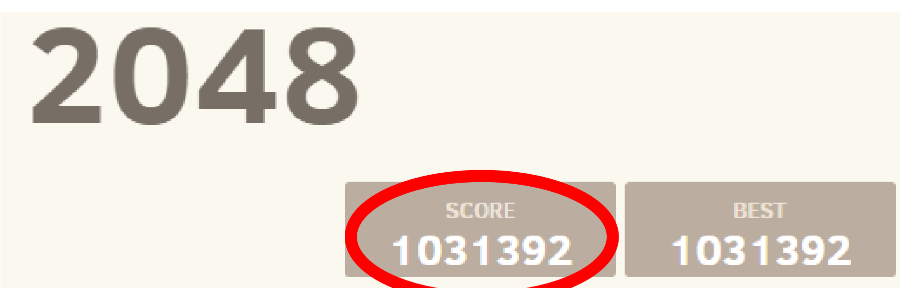
Our Results (2021)

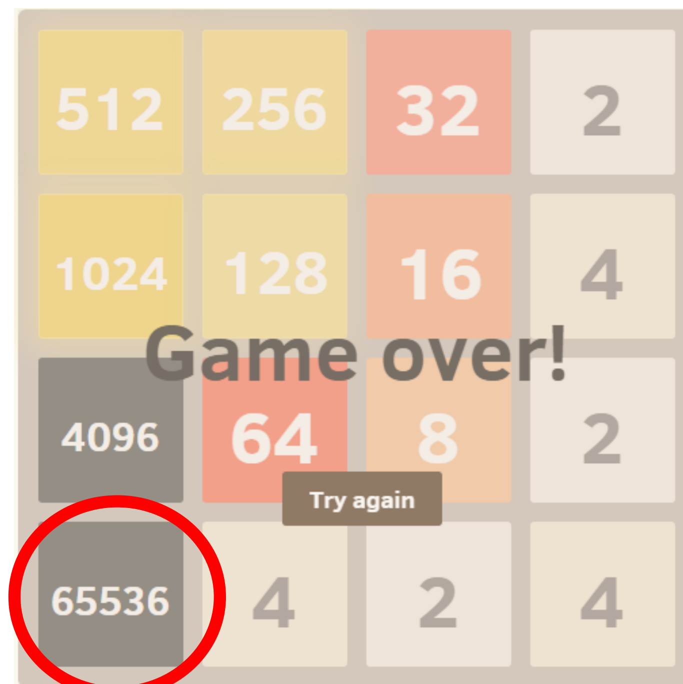
100 tested games	**CGI-2048 (2nd in contest, 2014)**	**Kcwu (1st in contest, 2014)**	**Jaśkowski (2018, Previous SOTA)**	**Current CGI-2048 (2021, Current SOTA)**
2048	100%	100%	**100%**	**100.0%**
4096	100%	100%	**100%**	**100.0%**
8192	94%	96%	**98%**	**99.8%**
16384	59%	67%	**97%**	**98.8%**
32768	0%	2%	**70%**	**72.0%**
Max score	367956	625260	**N/A**	**840384**
Avg score	251794	277965	**609104**	**625377**
Speed	500 moves/sec	>100 moves/sec	**1 move/sec**	**2.5 moves/sec**

The First 65536









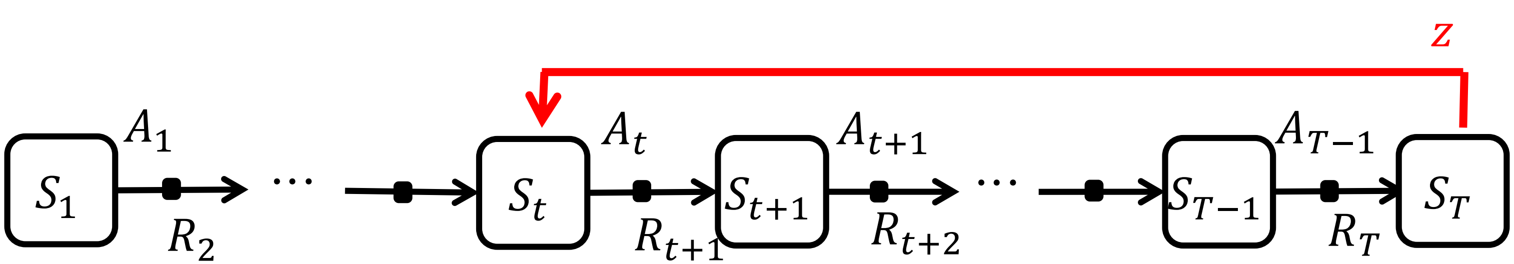
I-Chen Wu

Reinforcement Learning for Lightweight Model

- Applications
  - 2048 (Temporal Difference Learning)
  - Go Programs (with Monte-Carlo Tree Search)

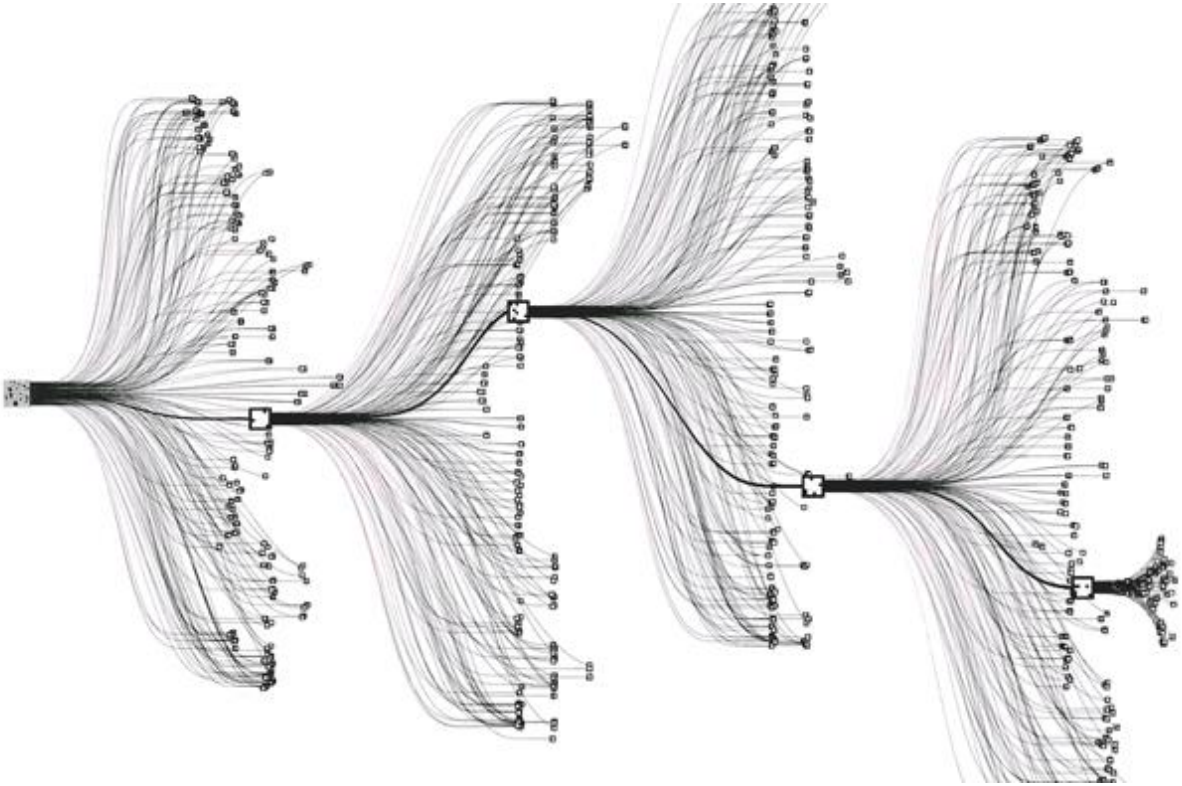
Case Study: Go

- Monte-Carlo Tree Search:
  - Monte-Carlo (MC) Learning (z: 1 for win, 0 for loss)
  - Multi-Armed Bandits
  - **Planning**
- Very successful for Go in the past two decades.
- And also applied to others successfully.
  - Other games like Havannah, Hex, GGP
  - Other applications, like mathematical optimization problems (scheduling, UCP, camera coverage).



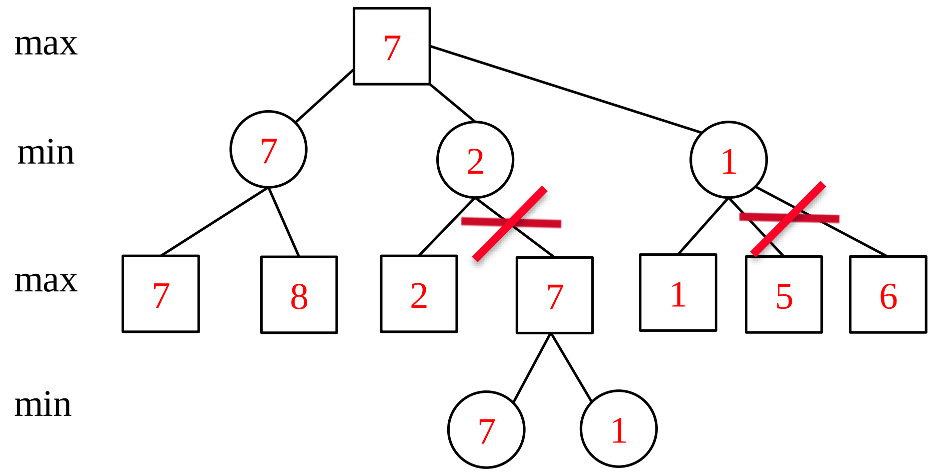
Go – One of the Most Popular Games

- Game tree complexity: about 10^{360}
  - It is just impossible to try all moves. (from DeepMind)



Can Alpha-Beta Search Work for Go?

- Alpha-Beta Search
  - Very successful for many games such as chess.
    - Almost dominate all computer games before 2006.
    - This is what Deep Blue used.
- The key for chess: evaluate position accurately and efficiently. E.g., features:
  - King: 1000
  - Queen: 200
  - Rook: 100
  - Knight: 80
  - Bishop: 70
  - Pawn: 30
  - Guarded Pawns: 30
  - Guarded Knights: 40
  - …



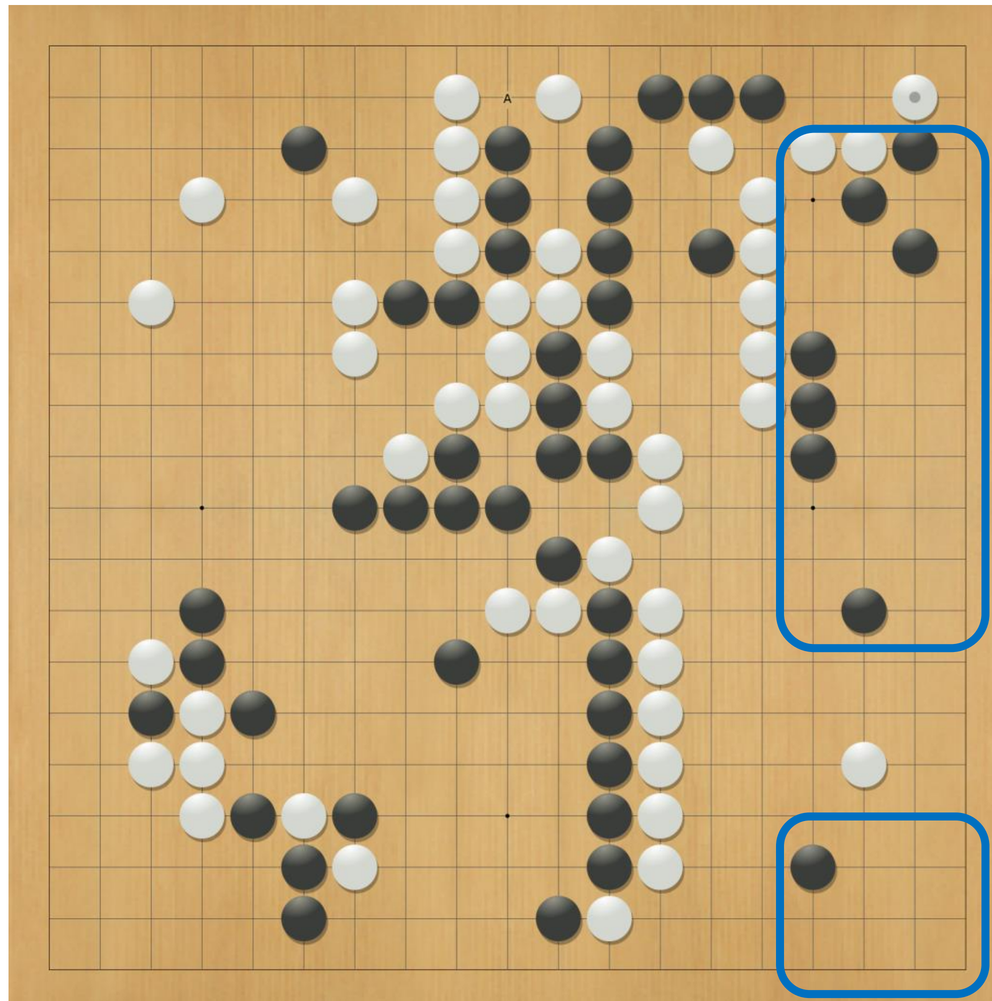
- Problem for chess:
  - need to consult with experts for feature values.

Why not alpha-beta search for Go?

- No simple heuristics like chess.
  - Only black/white pieces (no difference)
- Must know life-and-death
  - But, all are correlated.
    - Like the lower-right one.
  - But, this is very complex.

Since no simply heuristics to evaluate,

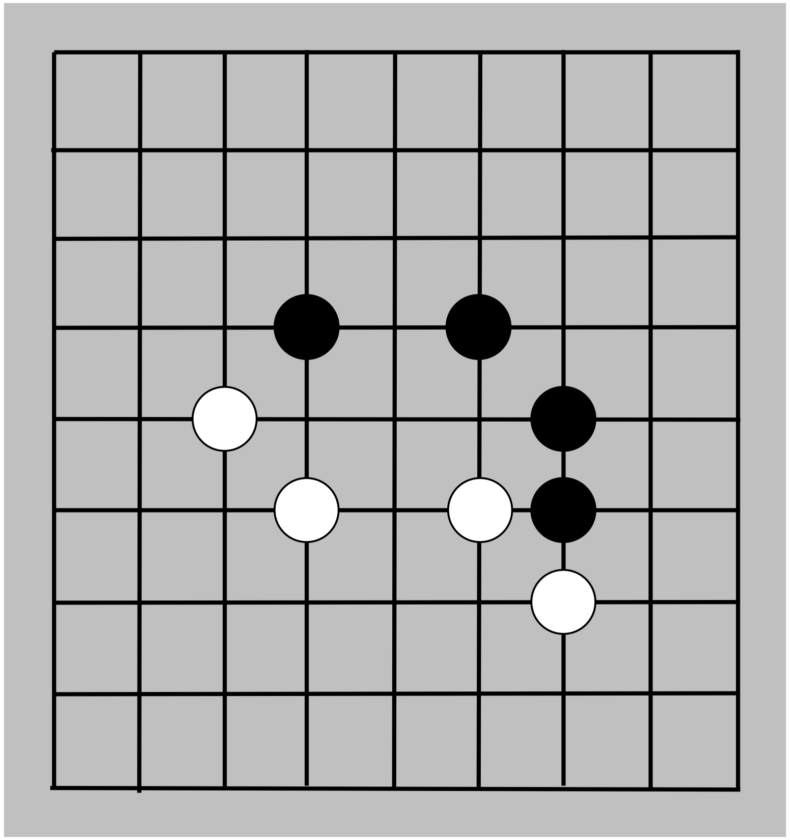
- Why not use Monte-Carlo?
- Calculate it based on stochastics.



Game 1: AlphaGo vs. 李世石

Rules Overview Through a Game (opening 1)

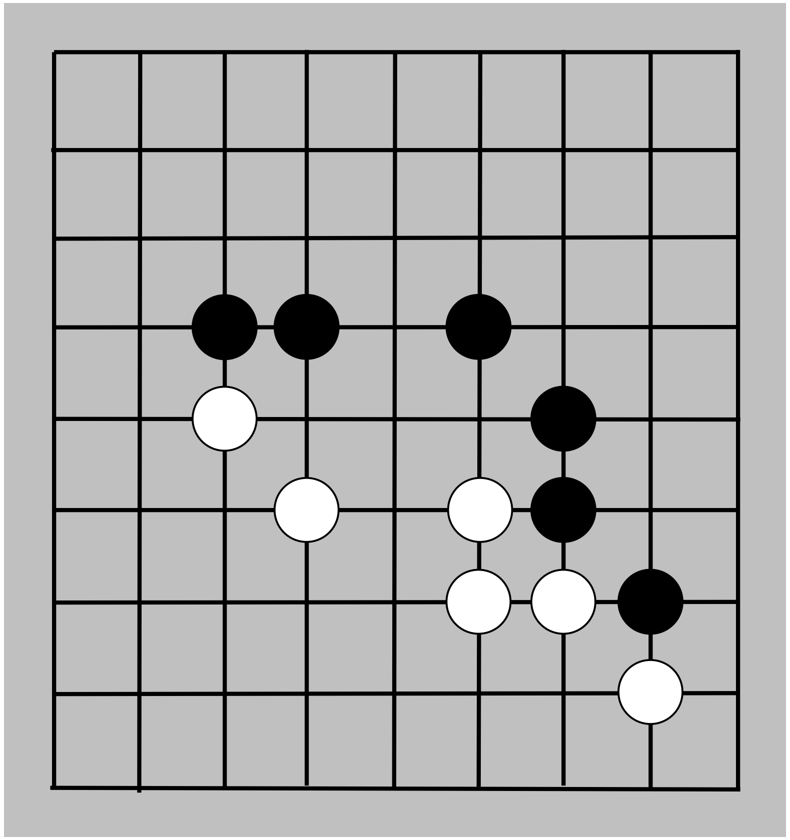
- Black/White move alternately by putting one stone on an intersection of the board.



The example was given by B. Bouzy at CIG'07.

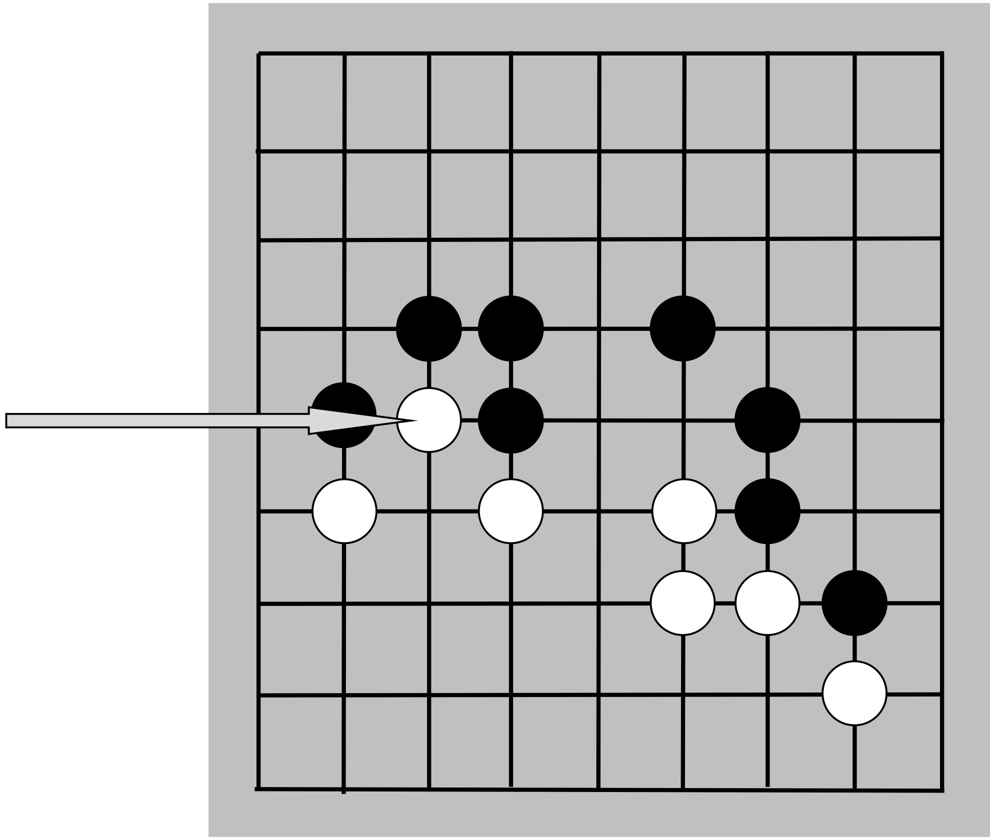
Rules Overview Through a Game (opening 2)

- Black and White aims at surrounding large « zones »



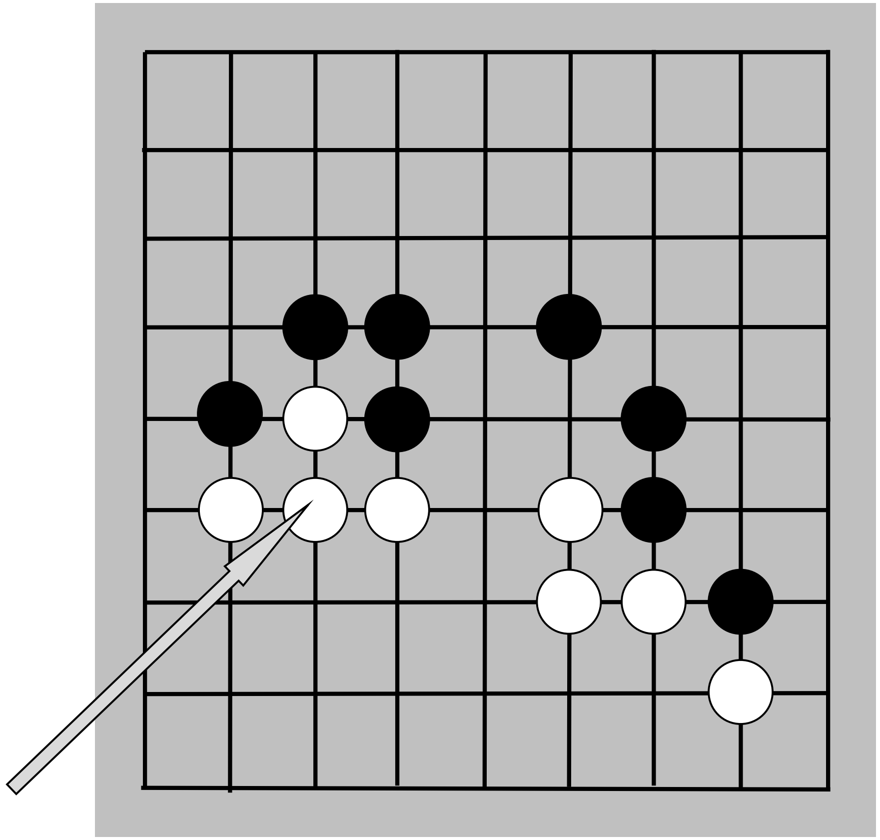
Rules Overview Through a Game (atari 1)

- A white stone is put into « atari » : it has only one liberty left.



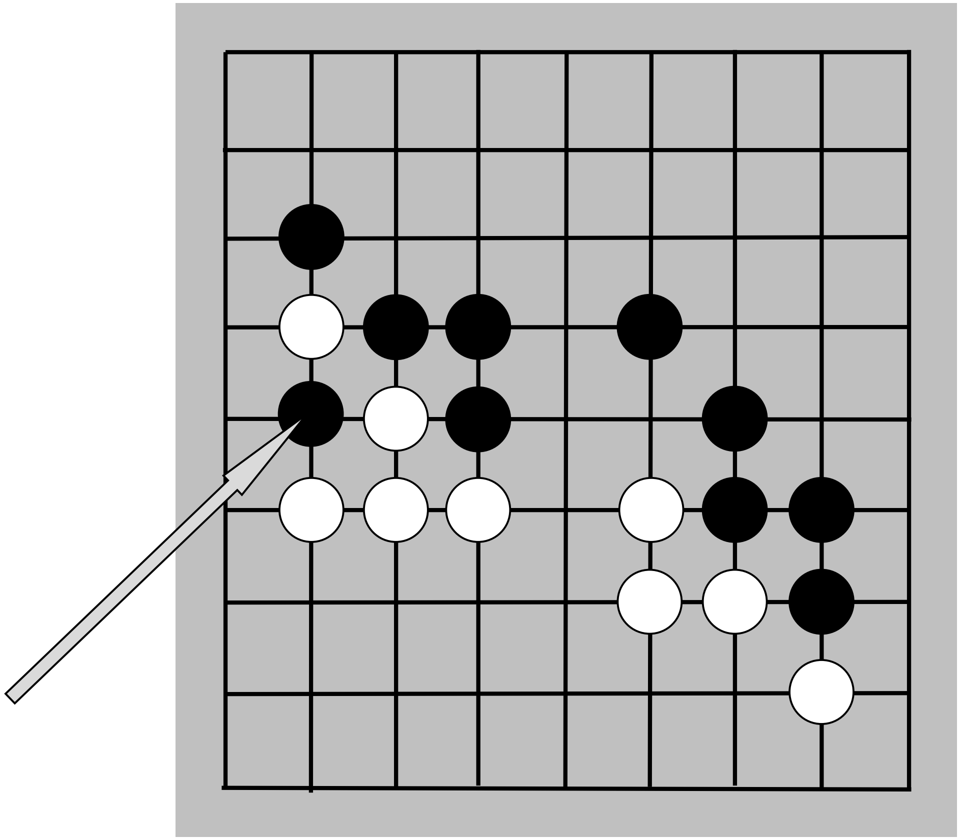
Rules Overview Through a Game (defense)

- White plays to connect the one-liberty stone yielding a four-stone white string with 5 liberties.



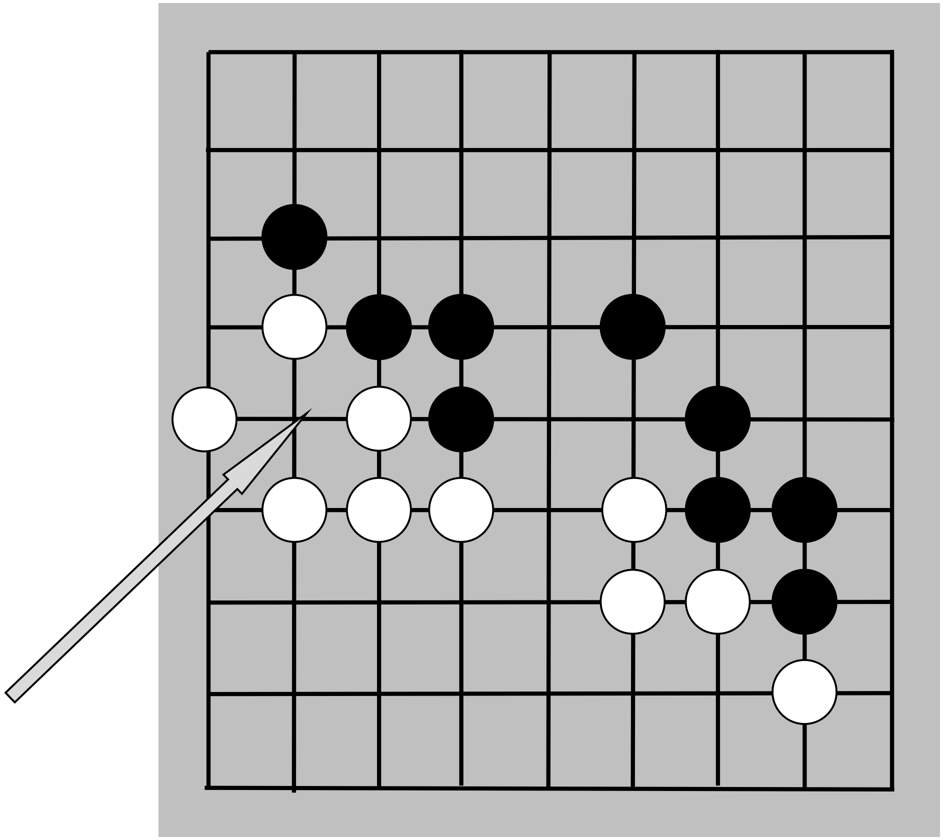
Rules Overview Through a Game (atari 2)

- It is White's turn. One black stone is atari.



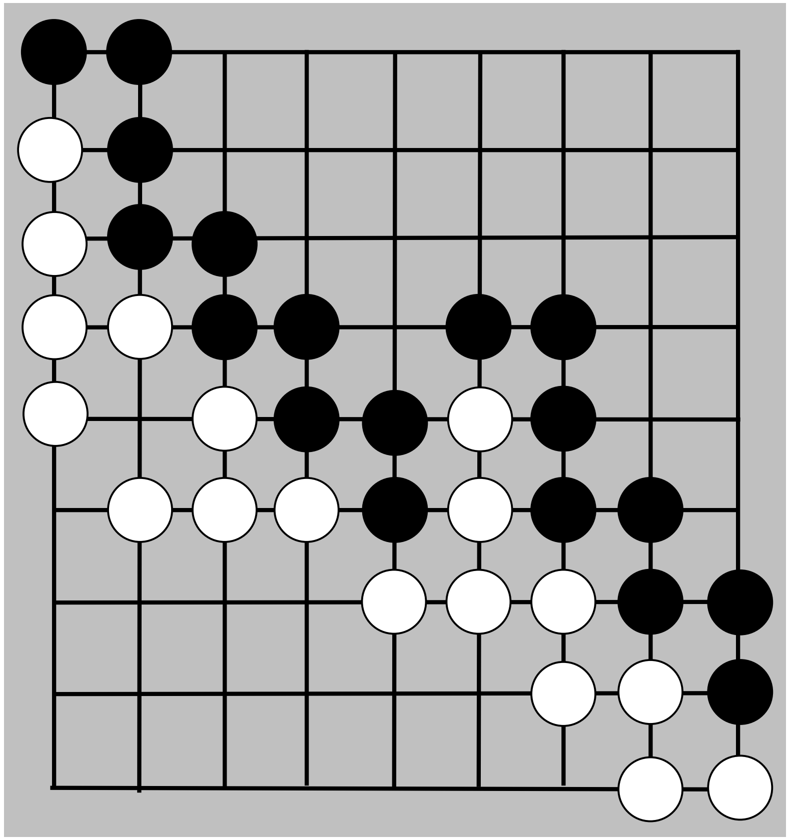
Rules Overview Through a Game (capture 1)

- White plays on the last liberty of the black stone which is removed



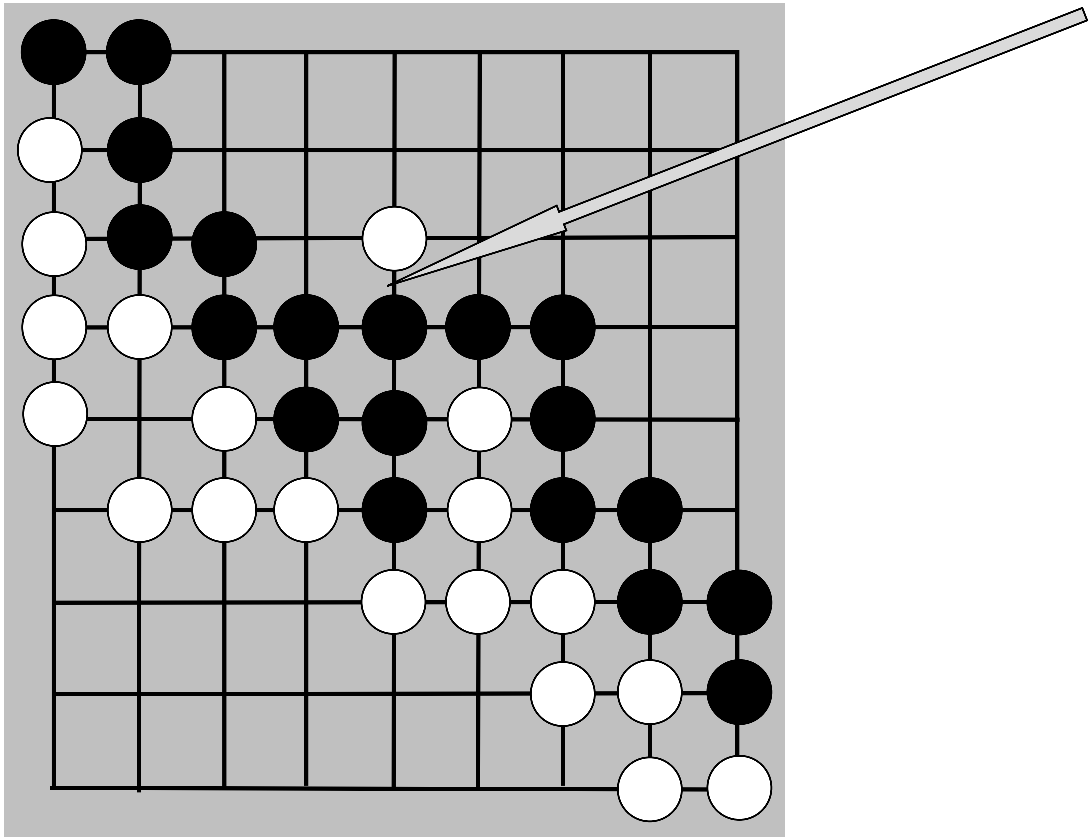
Rules Overview Through a Game (human end of game)

- The game ends when the two players pass. (Experts would stop here)



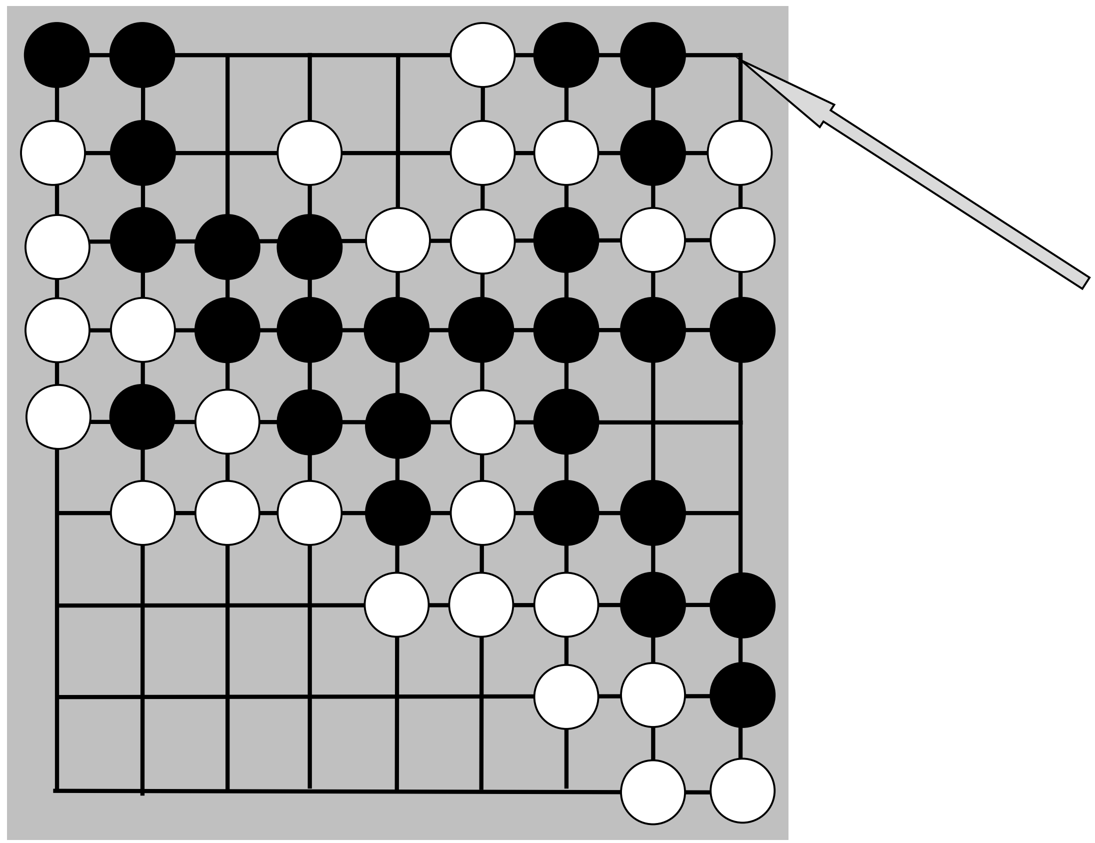
Rules Overview Through a Game (contestation 1)

- White contests the black « territory » by playing inside.



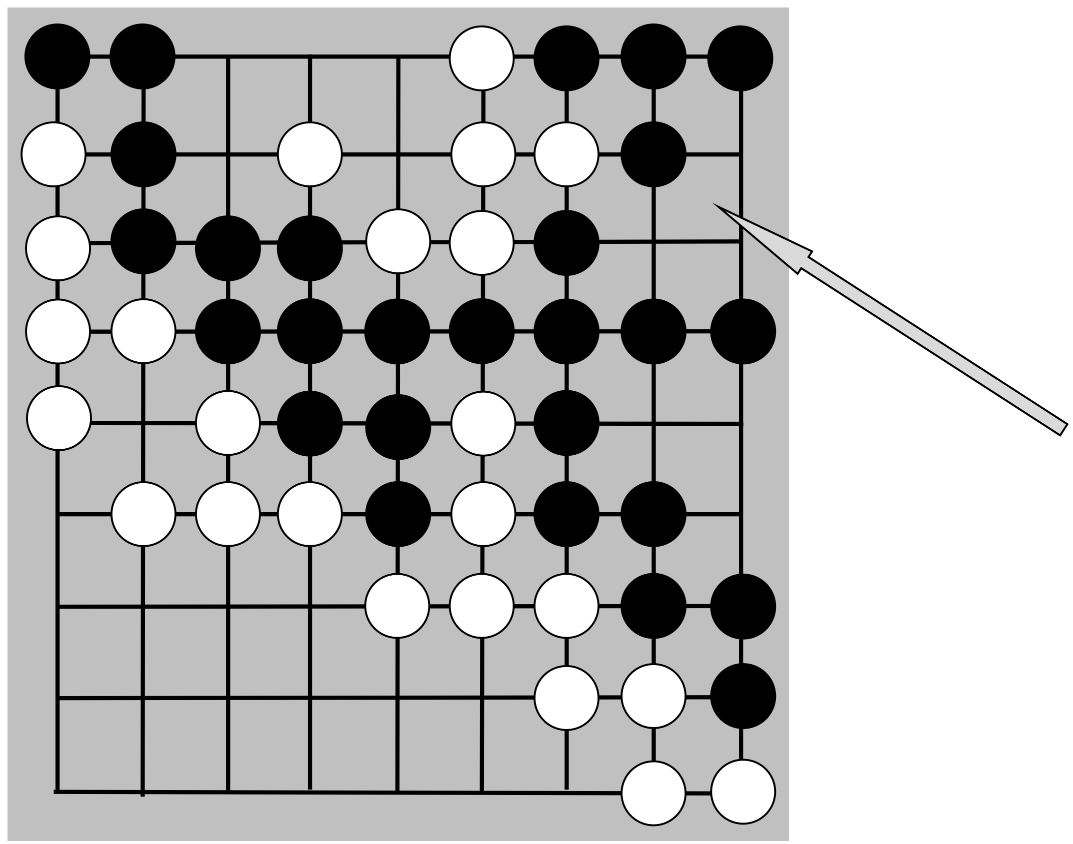
Rules Overview Through a Game (contestation 2)

- White contests black territory, but the 3-stone white string has one liberty left



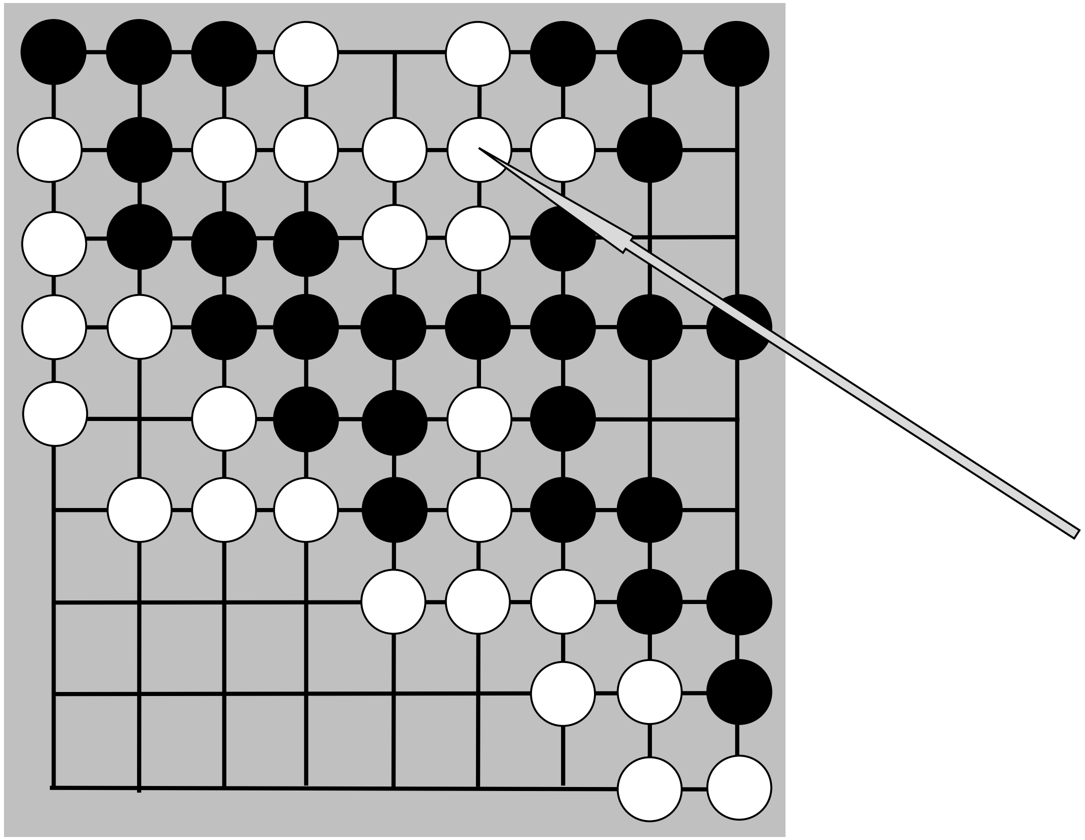
Rules Overview Through a Game (follow up 1)

- Black has captured the 3-stone white string



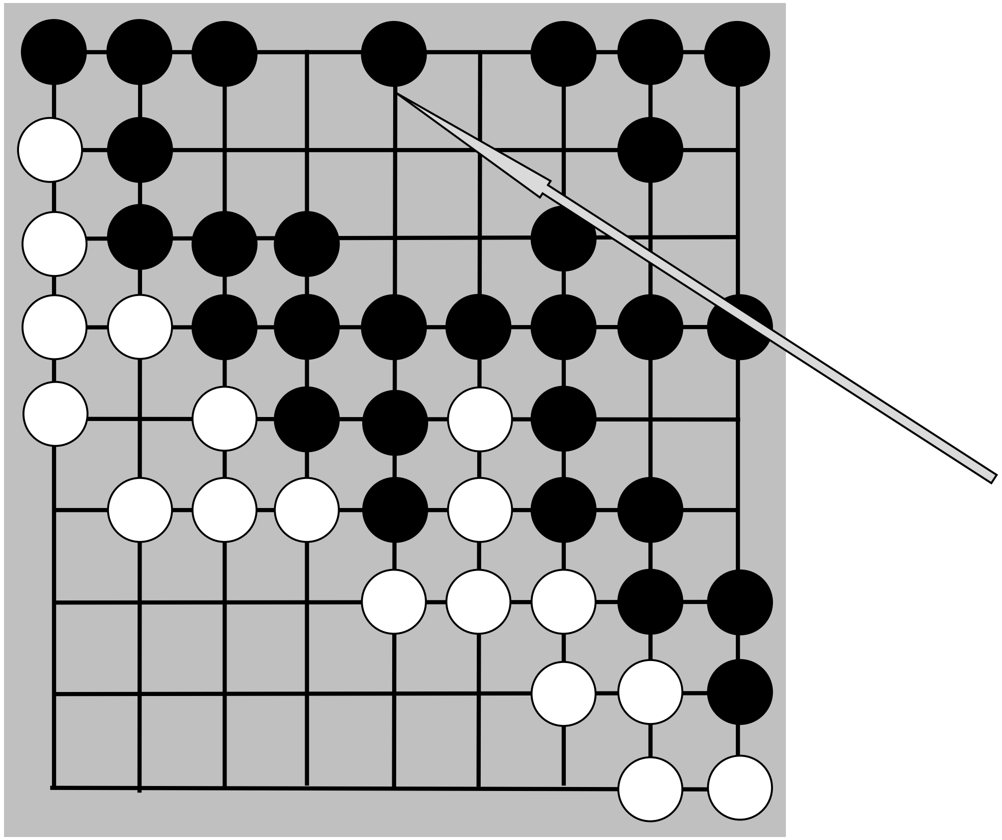
Rules Overview Through a Game (follow up 2)

- White lacks liberties…



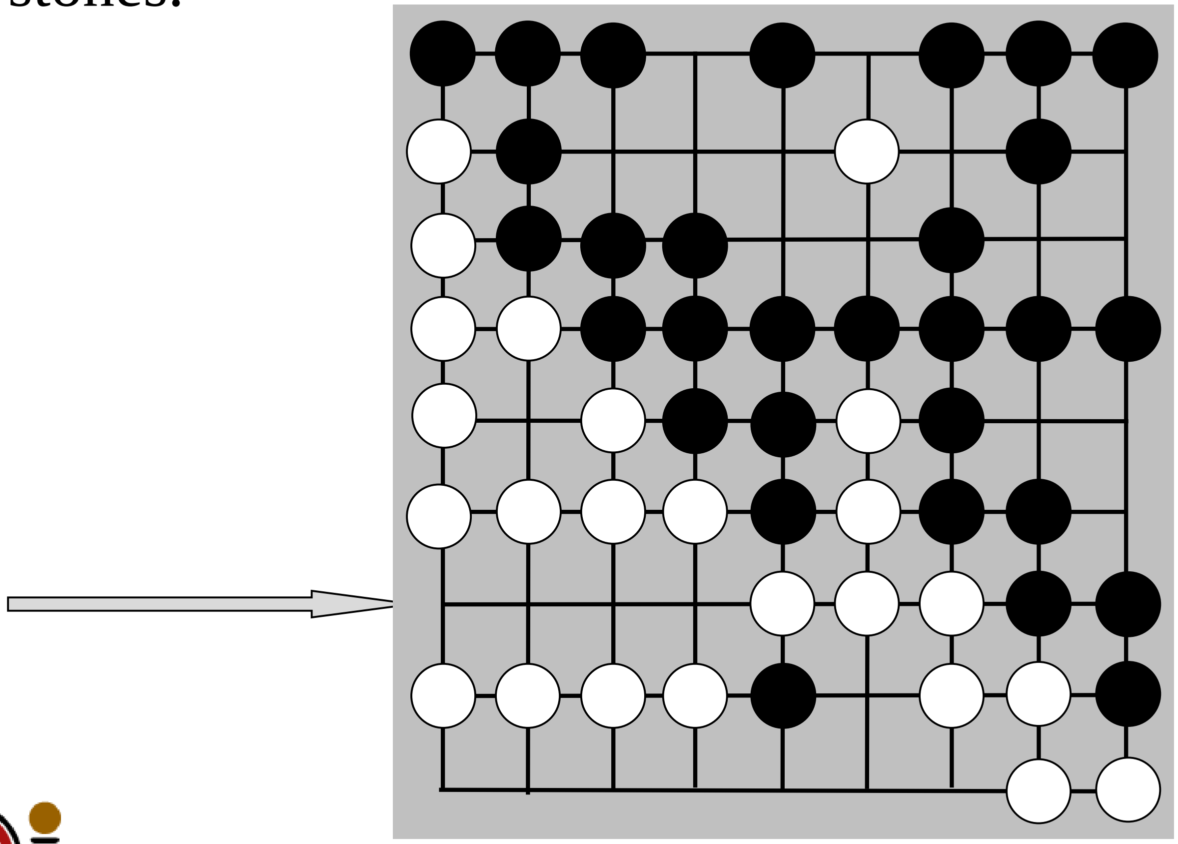
Rules Overview Through a Game (follow up 3)

- Black suppresses the last liberty of the 9-stone string
- Consequently, the white string is removed



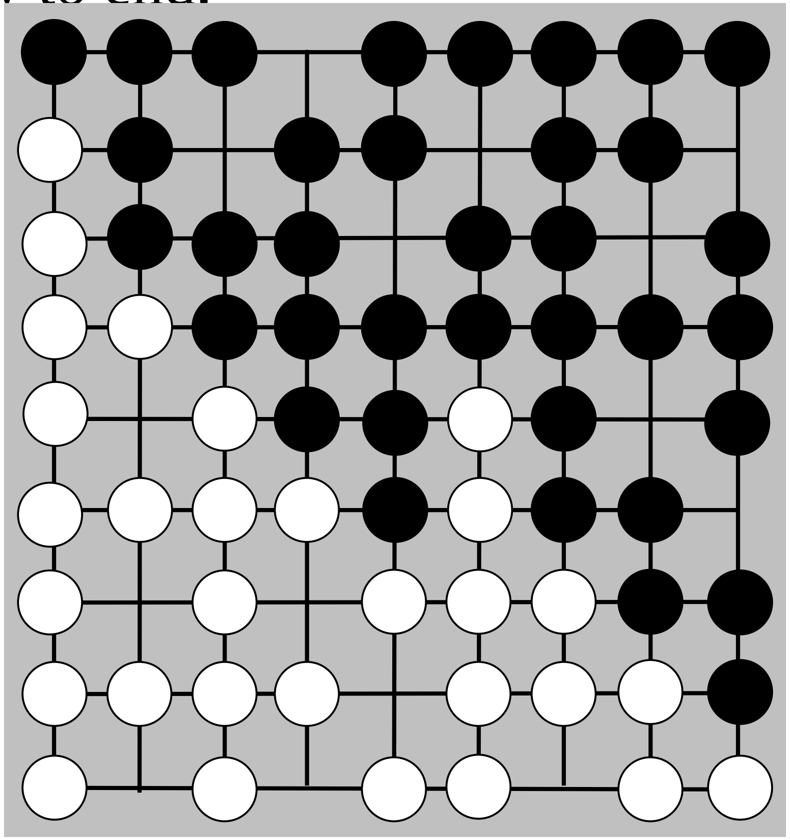
Rules Overview Through a Game (follow up 4)

- Contestation is going on. White has captured four black stones.



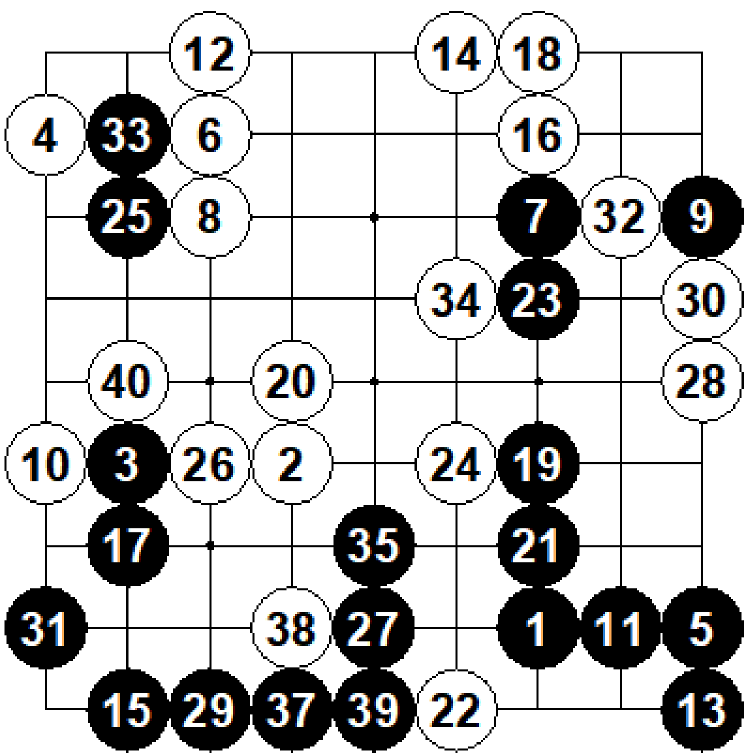
Rules Overview Through a Game (concrete end of game)

- The board is covered with either stones or « eyes ». Programs know to end.



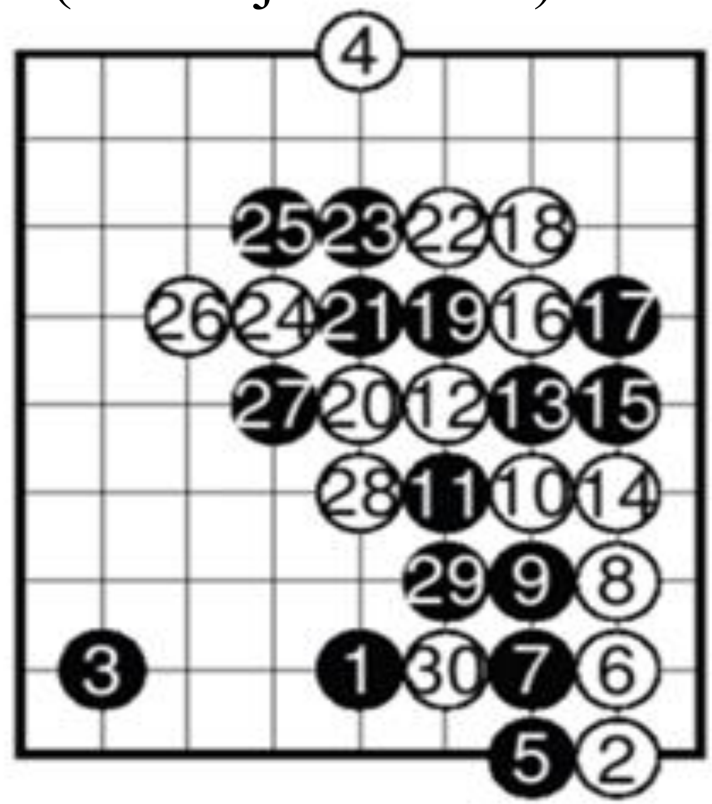
Performed OK Even for Moves (Nearly) at Random

Purely at random



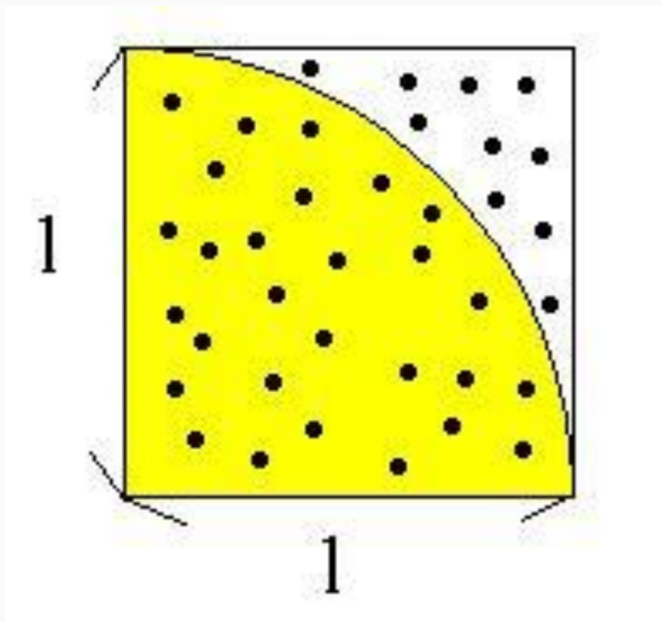
Have some heuristic

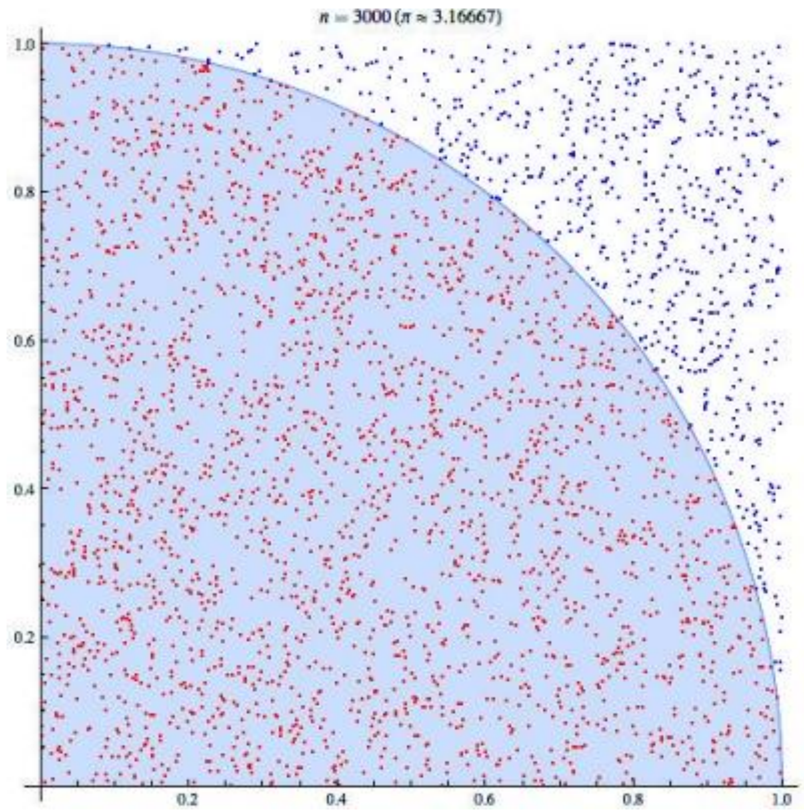
(from Aja's Thesis)



Stochastics

- Calculate values based on stochastics.
  - Good example: calculate π.





Multi-Armed Bandit Problem (吃角子老虎問題)

- Assume that you have infinite plays
  - How to choose the one with the maximal average return?

Exploration vs. Exploitation

- Example for the exploration vs exploitation dilemma
  - **Exploration**: is a long-term process, with a risky, uncertain outcome.
  - **Exploitation**: by contrast is short-term, with immediate, relatively certain benefits

Deterministic Policy: UCB1

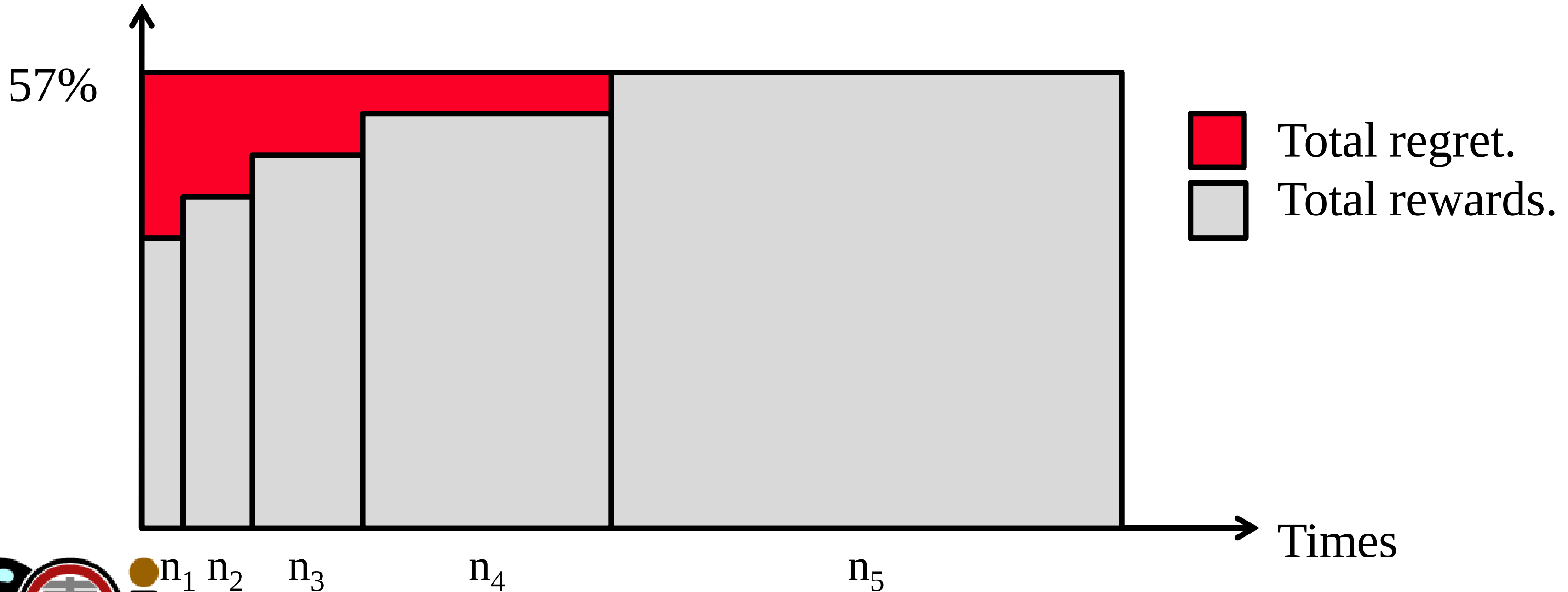
- UCB: Upper Confidence Bounds. [Auer *et al.*, 2002]
- Initialization: Play each machine once.
- Loop:
  - Play machine i that maximizes,

$$X_i + \sqrt{\frac{2\log n}{n_i}}$$

  - where
    - $n = \sum_{i=1}^{k} n_i$ is the total number of playing trials.
    - n_i is the number of playing trials on machine i.
    - X_i is the (average) win rate on machine i.
- Key:
  - Ensure optimal machine is played exponentially more often than any other machine.

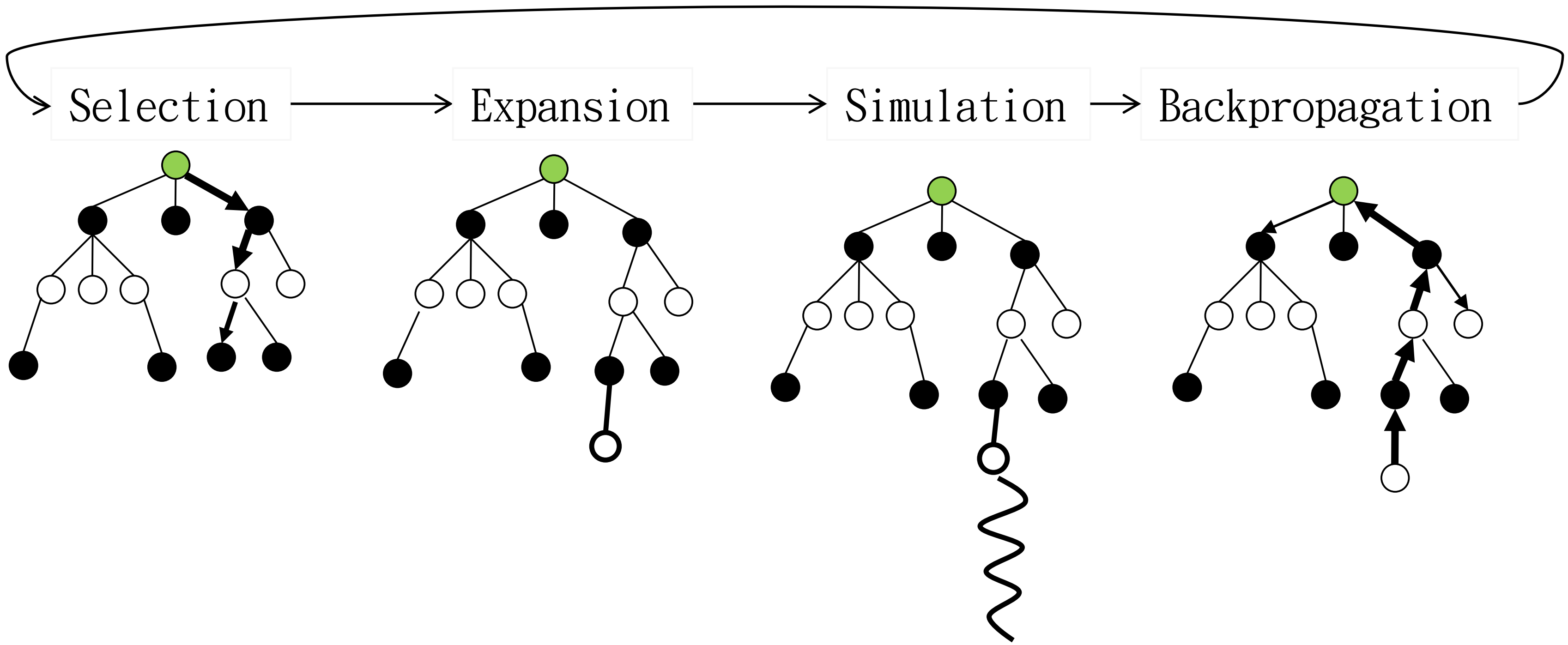
Cumulative Regret

- Assume Machines M_1, M_2, M_3, M_4, M_5
  - Win rates: 37%, 42%, 47%, 52%, 57%
  - Trial numbers: n_1, n_2, n_3, n_4, n_5.



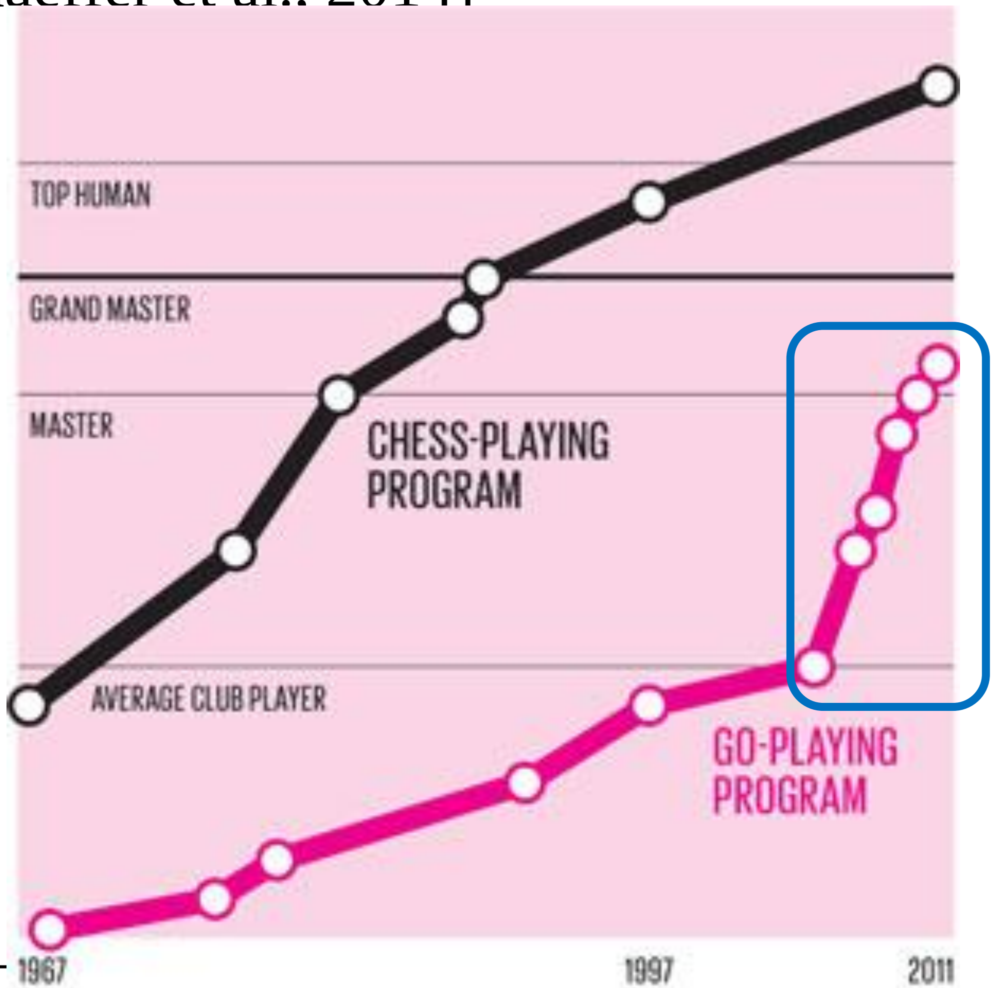
Monte-Carlo Tree Search

- A kind of planning
- A kind of Reinforcement learning



Strength of Go Program after MCTS

- [Schaeffer et al.. 2014]



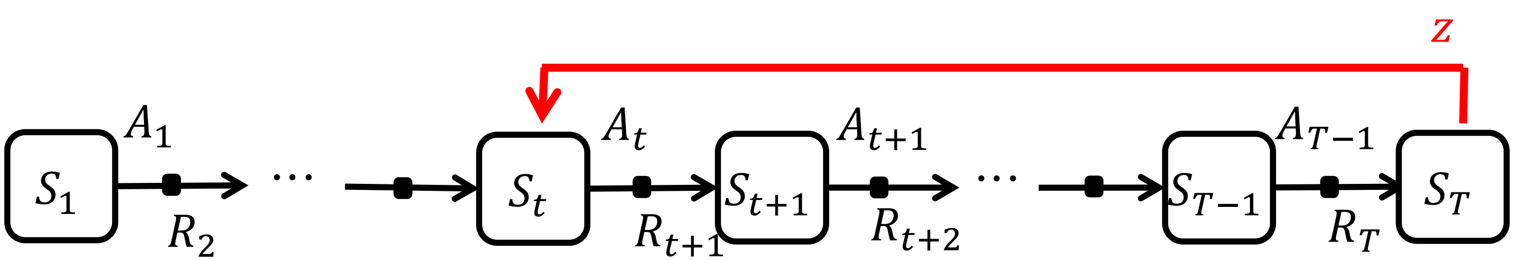
Strength grew fast, after MCTS.

Case Study: AlphaGo

- Use stochastic policy gradient ascent to maximize the likelihood of the human move a selected in state s

$$\Delta\theta = \alpha \nabla_\theta \log \pi_\theta(s_t, a_t) \cdot z$$

  - θ: network parameter.
  - α: learning rate
  - z: the value of the episode
    - win/loss (1/-1) of the game



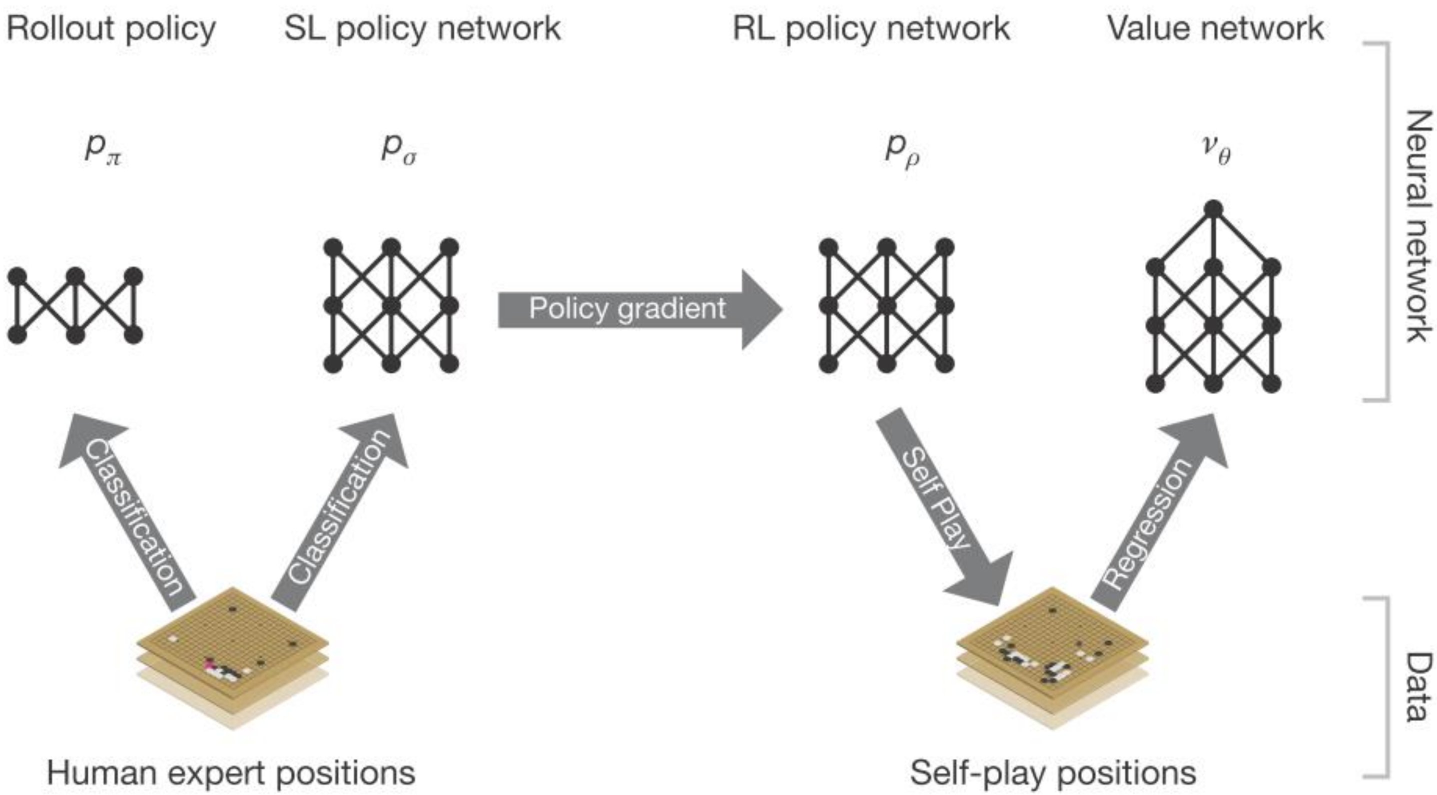
AlphaGo's Algorithm

- Use DCNN to learn experts' moves
  - (學習高手的著手策略)
- Use Monte-Carlo Tree Search (MCTS) for search to avoid pitfalls (避開陷阱)
  - MCTS is a kind of RL that do planning.
- Use DCNN to train "reinforcement learning (RL) network"
- Use DCNN to train "value network" (價值網路)
  - Learn the values of game positions (學習盤勢之優劣)

AlphaGo's Algorithm

- Use DCNN to learn experts' moves → DL
  - (學習高手的著手策略)
- Use Monte-Carlo Tree Search (MCTS) for search to avoid pitfalls (避開陷阱) → RL
  - MCTS is a kind of RL that do planning.
- Use DCNN to train "reinforcement learning (RL) network" → DRL (Policy Gradient)
- Use DCNN to train "value network" (價值網路)
  - Learn the values of game positions (學習盤勢之優劣) →DL

Policy Network and Value Network

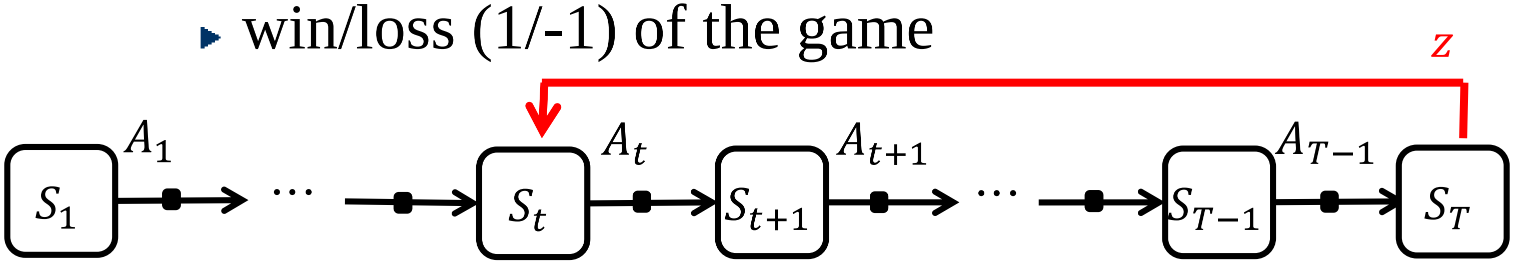


RL Policy Network: AlphaGo

- Use stochastic policy gradient ascent to maximize the likelihood of the human move a selected in state s

$$\Delta\theta = \alpha \nabla_\theta \log \pi_\theta(s_t, a_t) \cdot z$$

  - θ: network parameter.
  - α: learning rate
  - z: the value of the episode
    - win/loss (1/-1) of the game



AlphaGo Zero

- Use Monte-Carlo Tree Search (MCTS) → RL
  - Learn to find the best move (avoid pitfalls)
- Combine “value/policy network” → DRL

Like a tutor

Learn from Zero Knowledge!!!

Like a student