Introduction to Reinforcement Learning

I-Chen Wu

- Sutton, R.S. and Barto, A.G., Reinforcement Learning: An Introduction, MIT Press, Cambridge, MA, 1998.
  - http://webdocs.cs.ualberta.ca/~sutton/book/ebook/the-book.html
  - Bible in this area.
- David Silver, Online Course for Deep Reinforcement Learning.
  - http://www.cs.ucl.ac.uk/staff/D.Silver/web/Teaching.html

David Silver:

(the leader of the AlphaGo team)

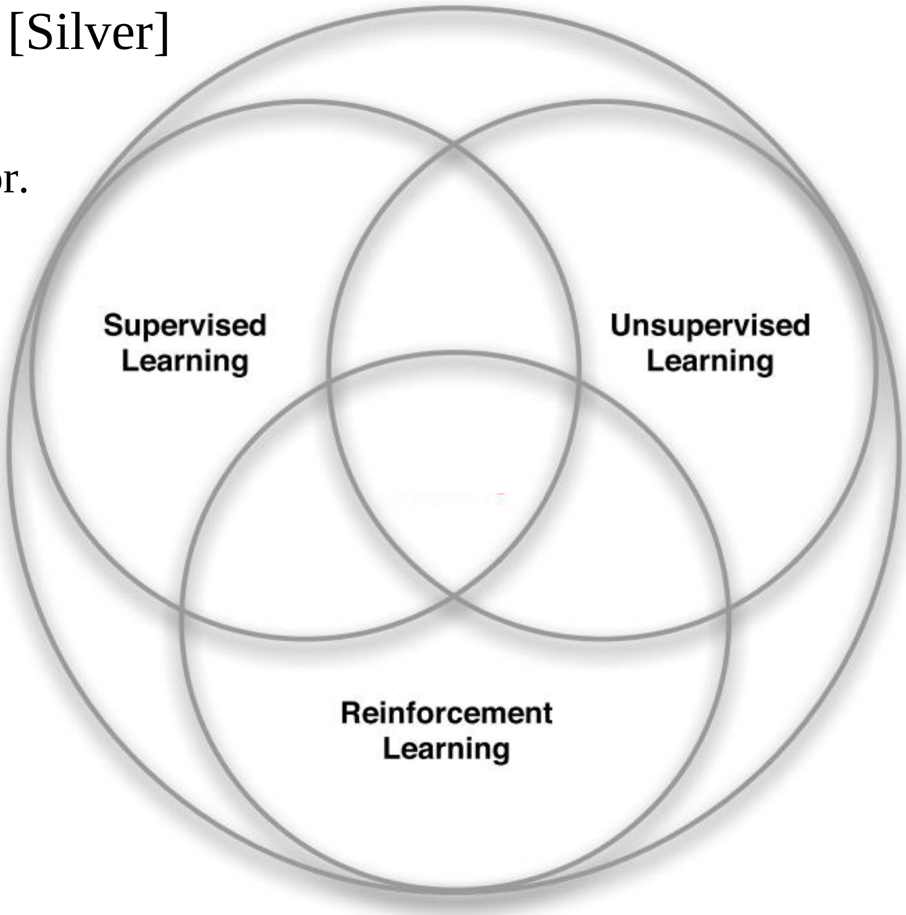
"DL+RL = AI"

Many Faces of Reinforcement Learning

- Computer Science
  - Machine Learning
- Engineering
  - Optimal Control
- Mathematics
  - Operations Research
- Economics
  - Bounded Rationality
- Psychology
  - Classical/Operant Conditioning
- Neuroscience
  - Reward System

Branches of Machine Learning

- Supervised Learning (SL) [Silver]
  - learning from a training set of labeled examples provided by a knowledgeable external supervisor.
- Unsupervised Learning (UL)
  - typically about finding structure hidden in collections of unlabeled data.
- Reinforcement Learning (RL)
  - learning from interaction



What are different from others?

- Characteristics:
  - No supervisor, only a reward signal
  - Feedback is delayed, not instantaneous
  - Time really matters
  - Agent's actions affect the subsequent data
- UL vs. RL:
  - RL is learning from interaction.
  - RL does not rely on examples of correct behavior.
  - RL is trying to maximize a reward signal,
    instead of trying to find hidden structure.

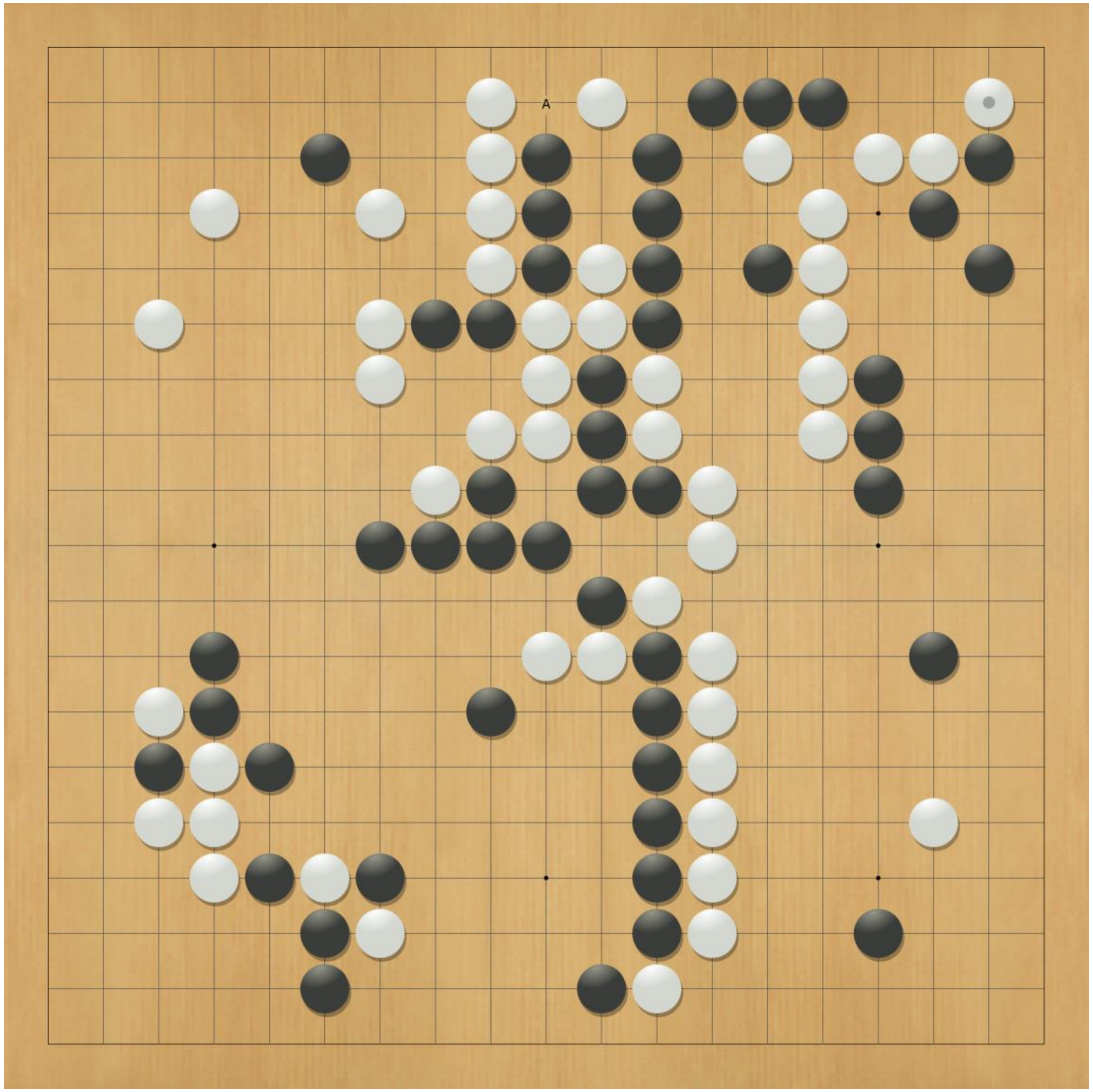
Successful Examples

- In AI, it has been used to defeat human champions at games of skill.
  - Backgammon (Tesauro, 1994).
  - Connect6/2048/Threes! (Wu et al., 2015). Reach the top levels.
  - Go programs, used in the past 10 years. (Monte-Carlo Tree Search)
  - AlphaGo, using deep reinforcement learning (2016)
- In robotics, fly stunt maneuvers in robot-controlled helicopters (Abbeel et al.) and make a humanoid robot walk.
- In economics, manage an investment portfolio (Choi et al.).
- In neuroscience, model the human brain (Schultz et al.);
- In psychology, predict animal behavior (Sutton and Barto).
- In systems, control a power station
- In engineering, it has been used to allocate bandwidth to mobile phones and to manage complex power systems (Ernst et al.).

(Not even include successful examples for deep reinforcement learning)

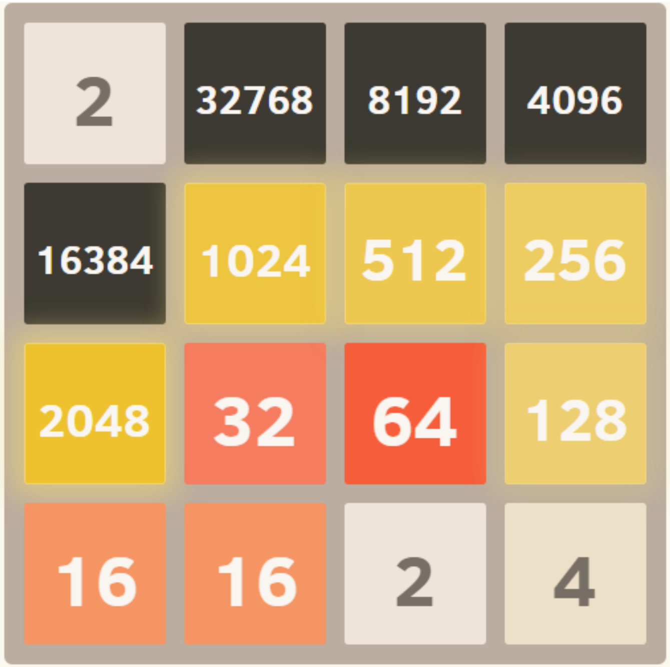
Board Game: Go

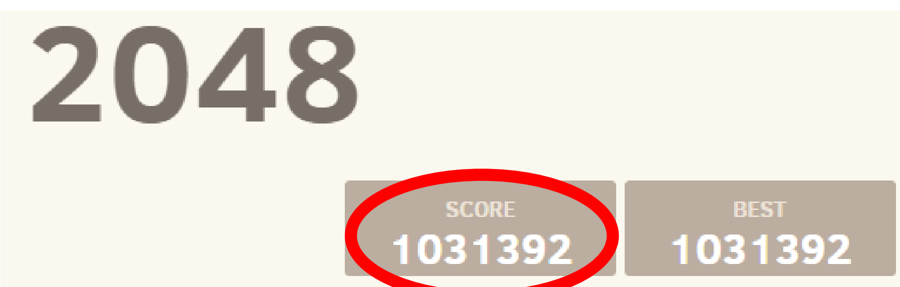
- Game 1: AlphaGo vs. 李世石



I-Chen Wu

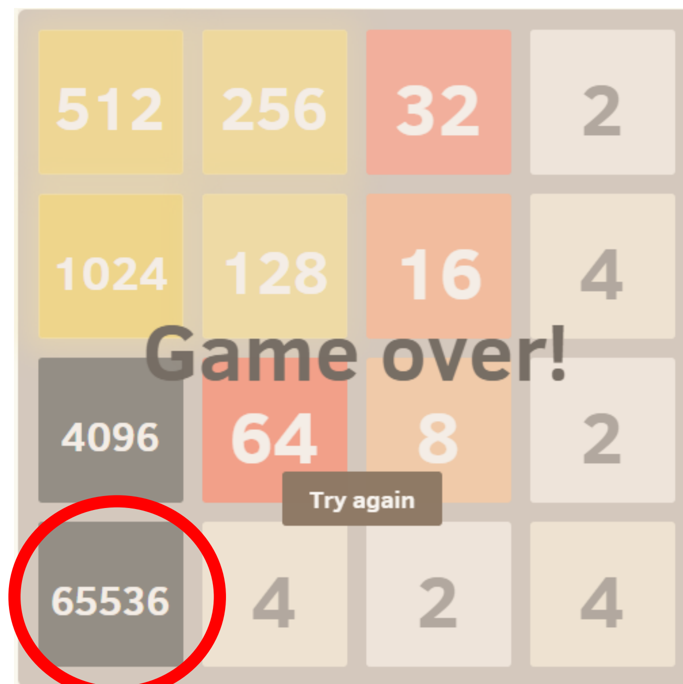
Stochastic Game: 2048 (lab)





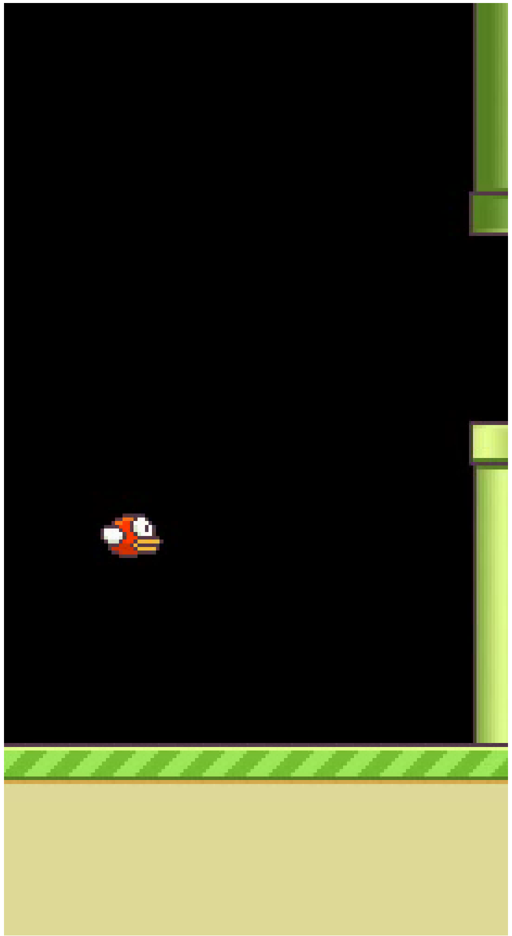
The First Game Reaching 65536 in the World (in 10,000 Trials)

http://2048.aigames.nctu.edu.tw/replay.php



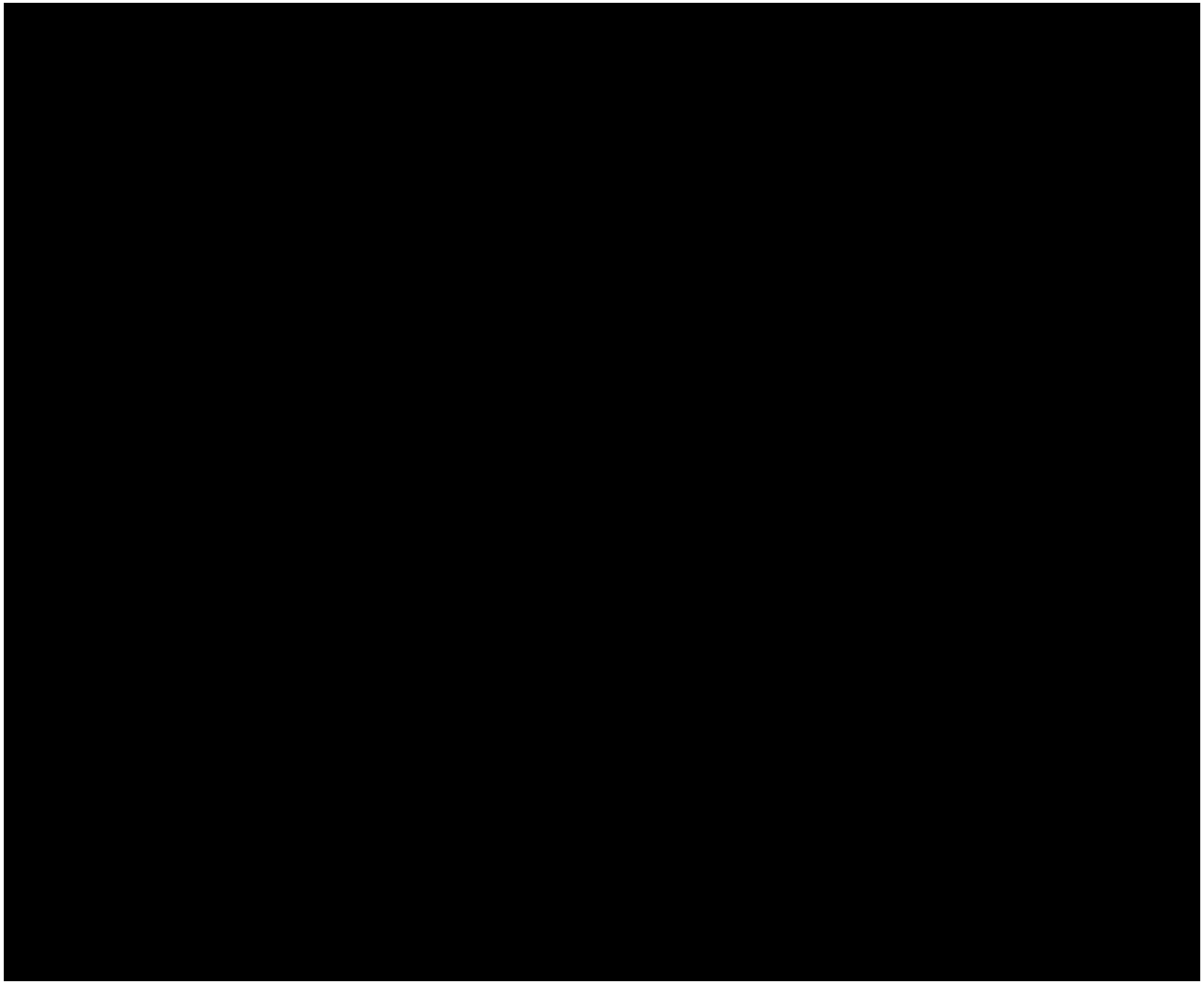
I-Chen Wu

Video Games: Flappy Bird (lab)



Open AI: Pole Balancing (lab)

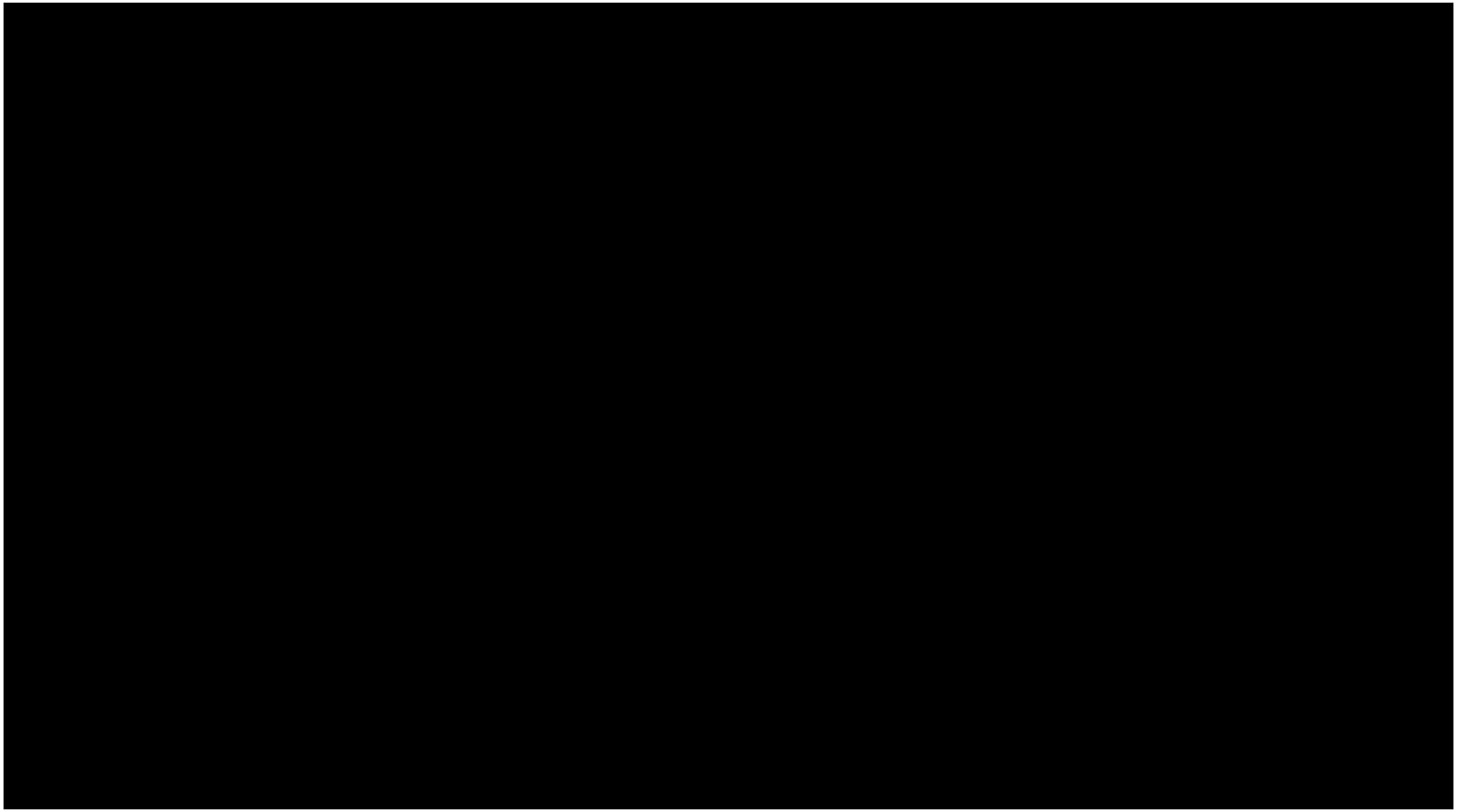
RL Demo



RL Demo

[Deisenroth et al, 2011] Learning to Control a Low-Cost Manipulator using Data-Efficient Reinforcement Learning

Learning Contact-Rich Manipulation Skills with Guided Policy Search [Levine et. al. 2015]



ChatBot

- Hi, may I help you?
- I'm looking for a Chinese restaurant
  - Which area in mind?
- Somewhere in the downtown.
  - Hu Nan Restaurant is recommended by many people.
- Noop! The food is too hot.
  - How about Dumpling House?
- Good. Give me the direction to it.
  - It is located in ….
- Thank you.

  - Are you satisfied with the service.
- Yes.

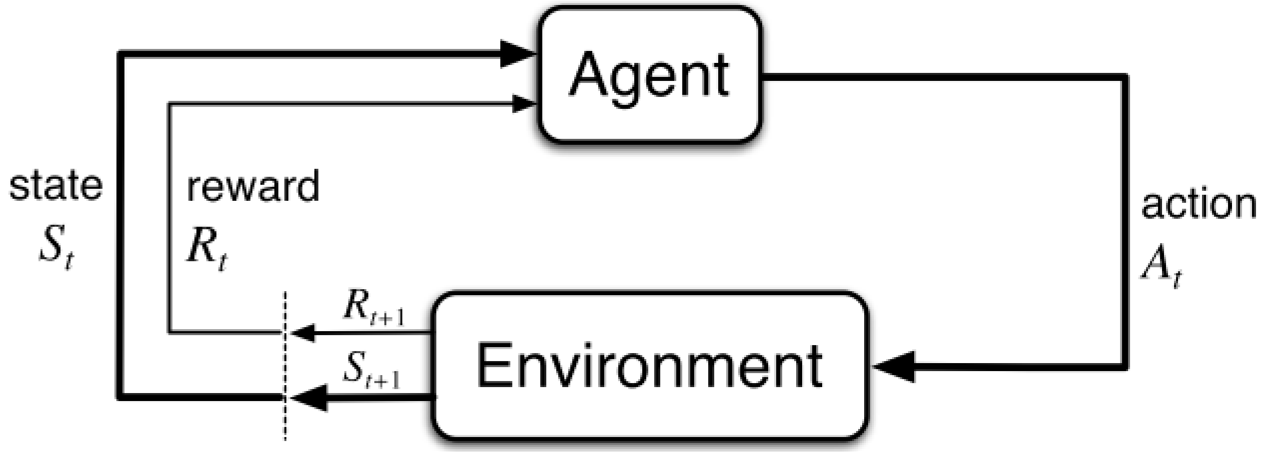
Stock Market

Reinforcement Learning

- A computational approach to learning from interaction
  - Explore designs for machines that are effective in
    - solving learning problems of scientific or economic interest,
    - evaluating the designs through mathematical analysis or computational experiments.
  - Focus on goal-directed learning from interaction, when compared with other approaches to machine learning.
  - The learner must discover which actions yield the most reward by trying them.
    - Two characteristics: most important distinguishing features of reinforcement learning.
      - trial-and-error search
      - delayed reward

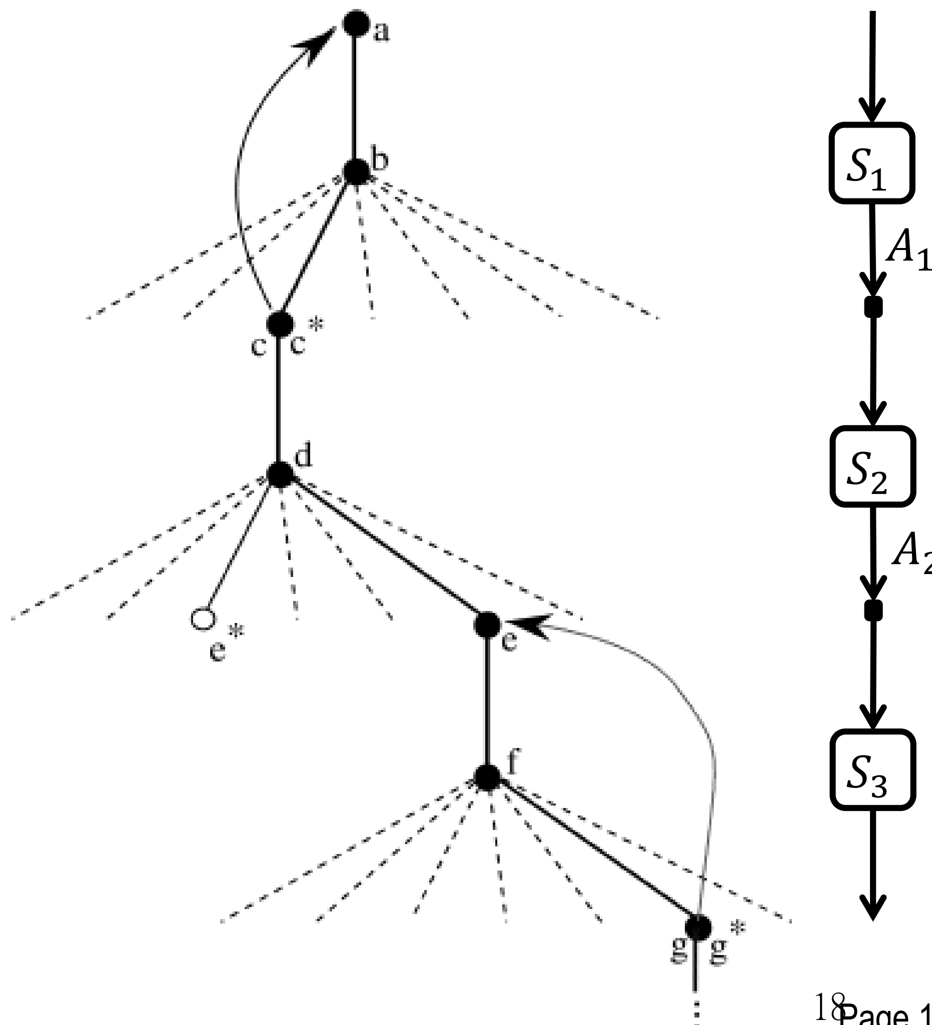
Agent-Environment Interaction Framework

- Agent: The learner and decision-maker.
- Environment: The thing it interacts with, comprising everything outside the agent.
- State: whatever information is available to the agent.
- Reward: single numbers.



States and Actions in the Framework

Environment:	reaction
Agent:	action
Environment:	reaction
Agent:	action
Environment:	reaction
Agent:	action



Go

Environment: opponent's move

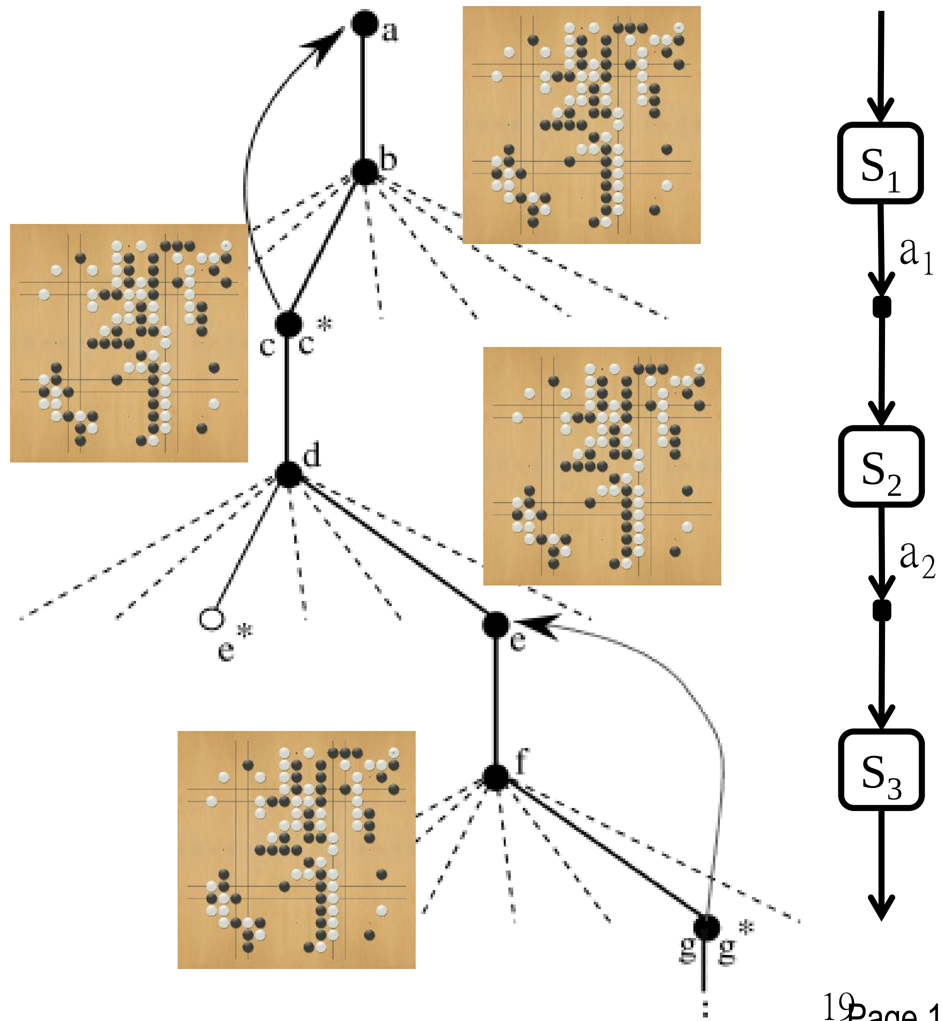
Agent: our move

Environment: opponent's move

Agent: our move

Environment: opponent's move

Agent: our move



2048

Environment: Tile generation

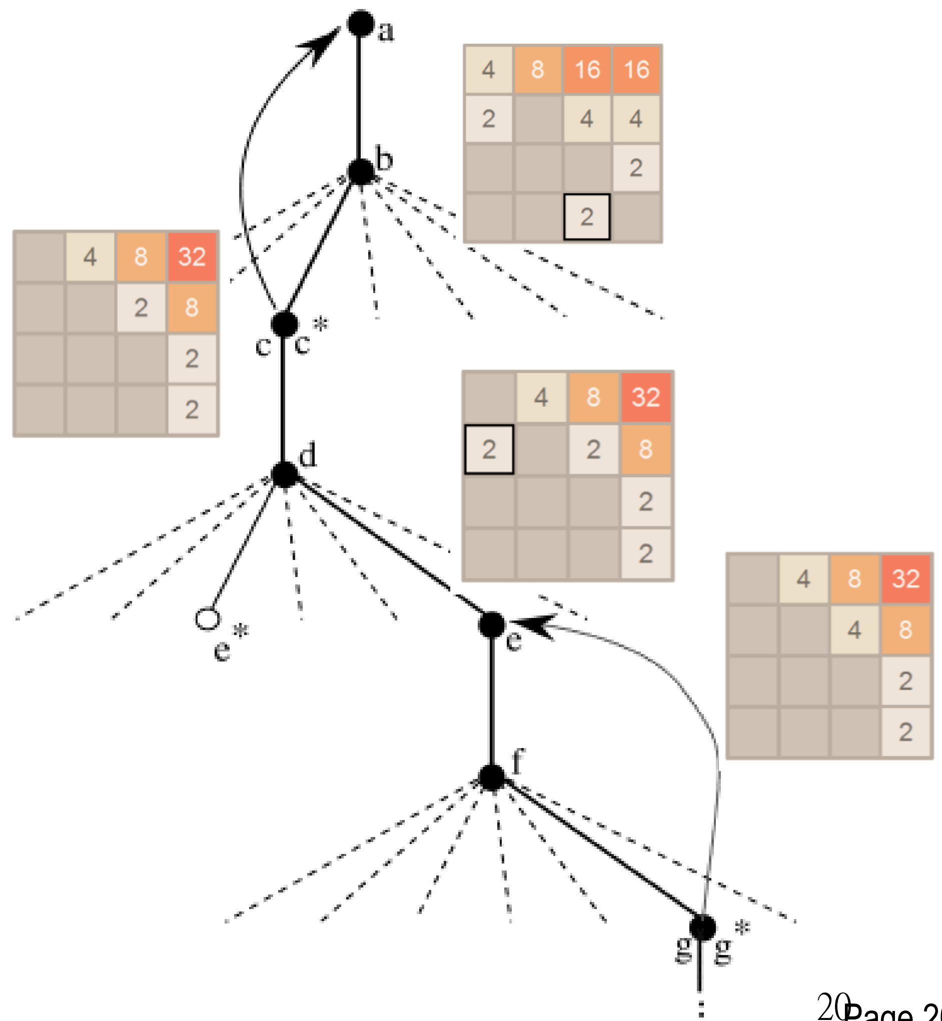
Agent: our move

Environment: Tile generation

Agent: our move

Environment: Tile generation

Agent: our move



Robot

Environment: Dynamics

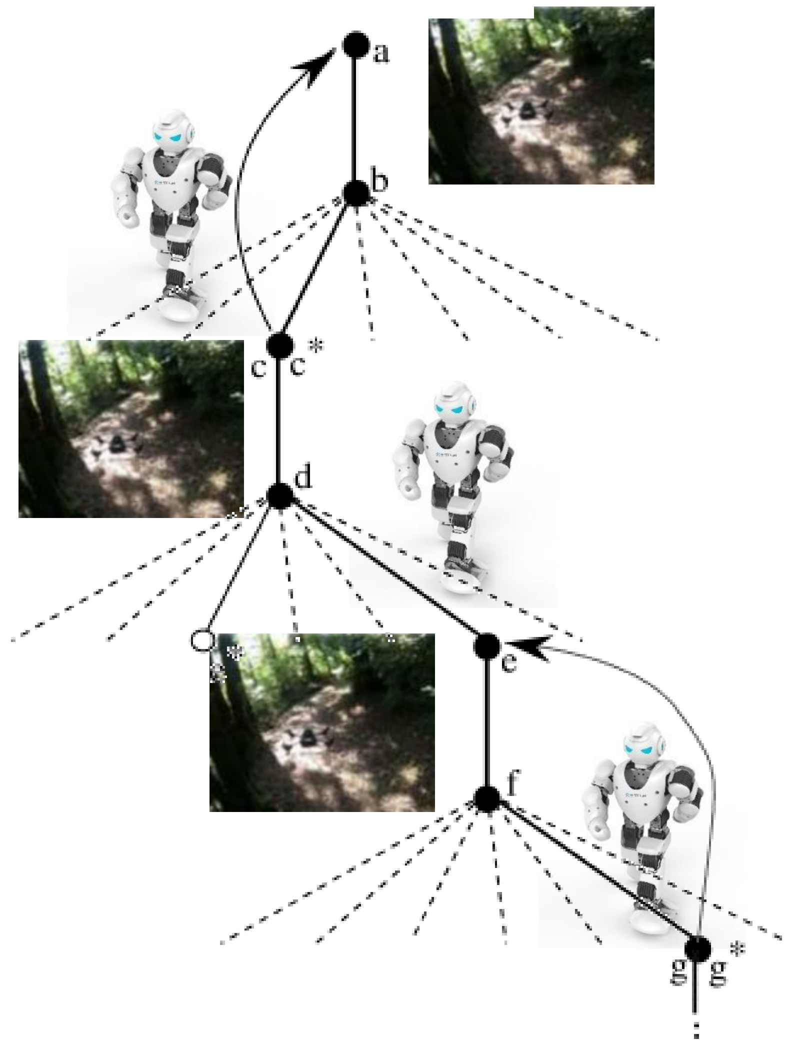
Agent: Navigate

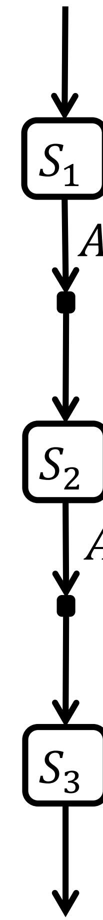
Environment: Dynamics

Agent: Navigate

Environment: Dynamics

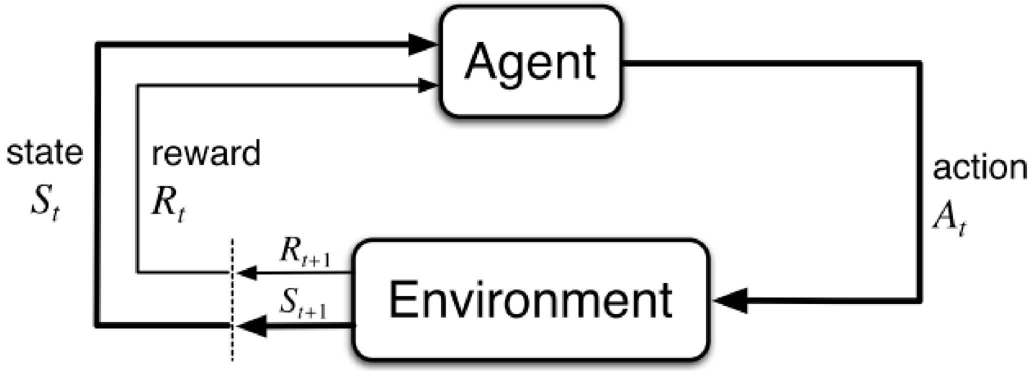
Agent: Navigate



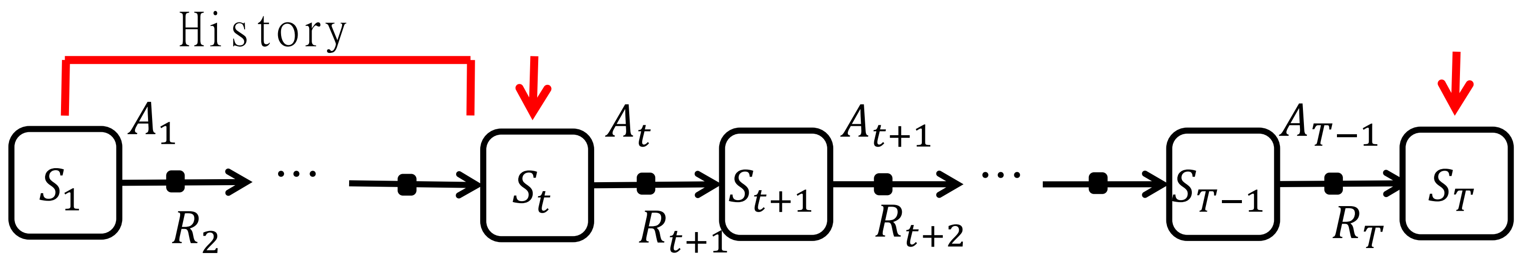


Markov Decision Processes (MDP)

- A Markov Decision Process is a tuple $<\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma>$
  - $\mathcal{S}$ is a finite set of states
  - $\mathcal{A}$ is a finite set of actions
  - $\mathcal{P}$ is a state transition probability matrix (part of the environment), $\mathcal{P}_{ss'}^{a} = \mathbb{P}[S_{t+1} = s' | \ S_t = s, A_t = a]$
  - $\mathcal{R}$ is a reward function, $\mathcal{R}_s^a = \mathbb{E}[R_{t+1} | S_t = s, A_t = a]$
  - γ is a discount factor $\gamma \in [0, 1]$.



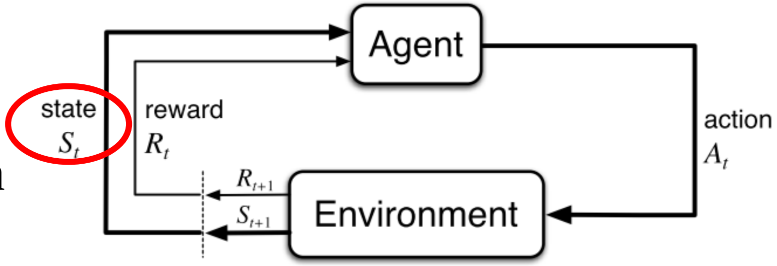
Markov Property



- An episode: (assuming finite and MDP here for simplicity)
  - States: S_i
    - Initial state: S_1
    - Current state: S_t
    - End state: S_T (not necessarily required)
  - Actions: A_i
  - History: $H_t = (S_1, A_1, R_2, \ S_2, A_2, R_3, S_3, \ldots, R_t)$
- Markov Property:
  - "The future is independent of the past given the present"
  - A state S_t is Markov if and only if

$$\mathbb{P}[S_{t+1} | S_t] = \mathbb{P}[S_{t+1} | S_1, \ldots, S_t]$$

Environment State vs. Agent State



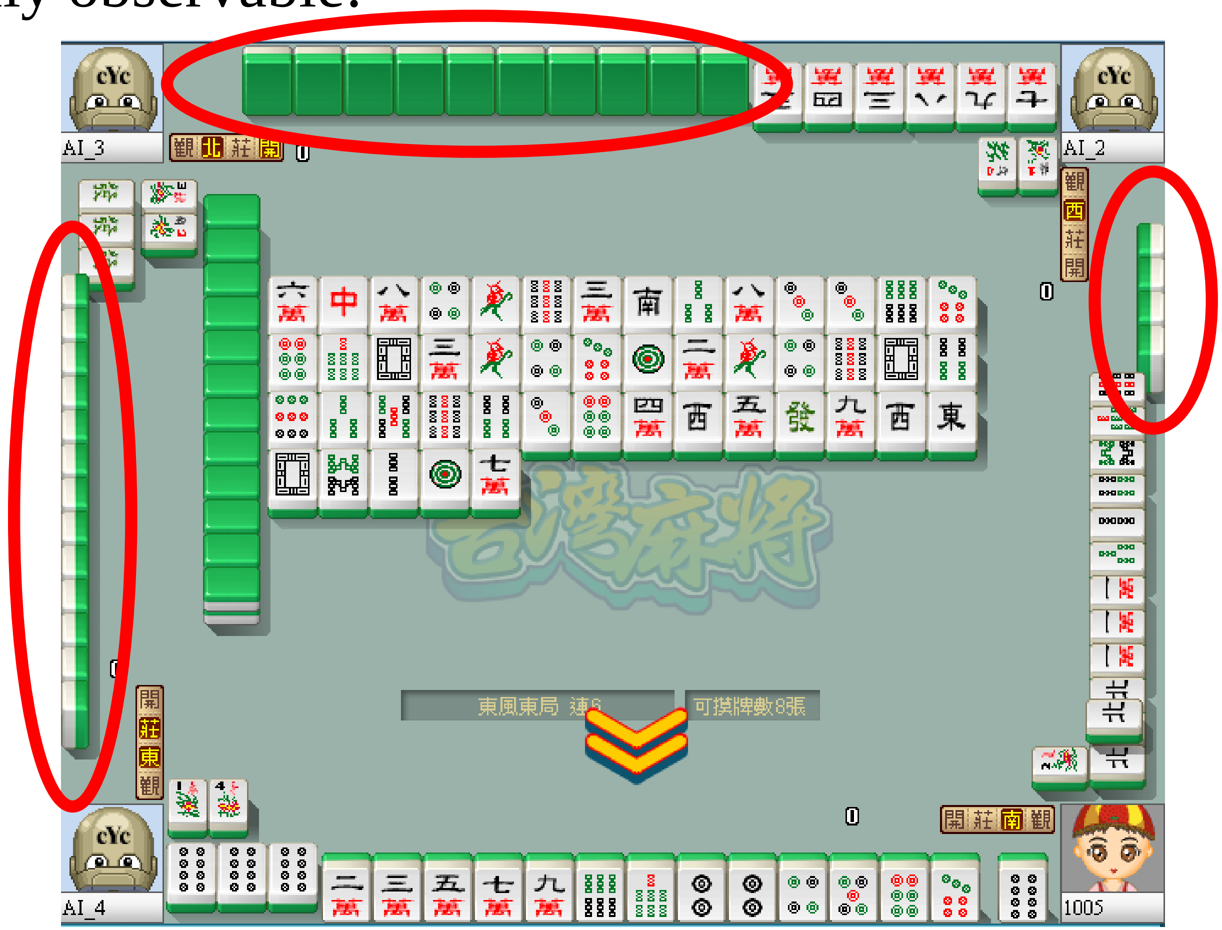
- The environment state S_t^e:
  - the environment's private representation
    - i.e. whatever data the environment uses to pick the next observation/reward
  - The environment state is not necessarily visible to the agent
    - Even if S_t^e is visible, it may contain irrelevant information
- The agent state S_t^a:
  - The agent's internal representation
    - i.e. whatever information the agent uses to pick the next action
    - i.e. it is the information used by reinforcement learning algorithms
  - It can be any function of history:

$$S_t^a = f(H_t)$$

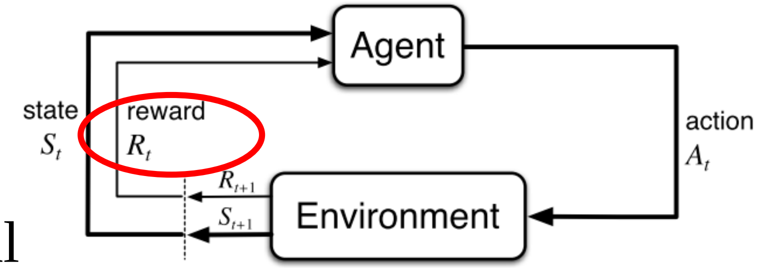
- Partially Observable: (not discussed here)
  - When $S_t^a \neq S_t^e$

Example: Mahjong

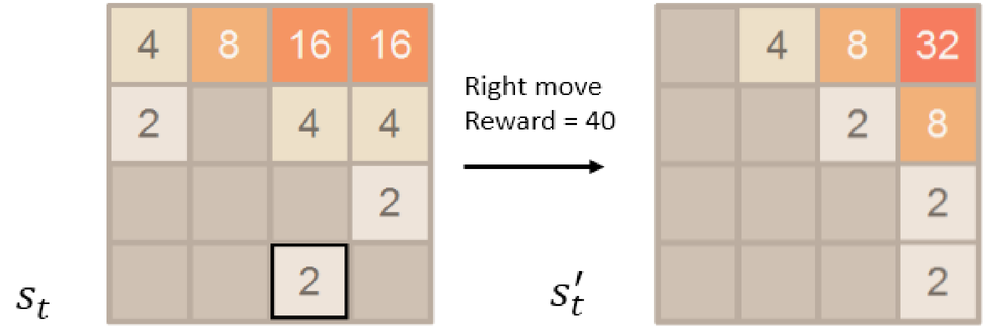
- Partially observable:



Rewards



- A reward R_t is a scalar feedback signal
  - Indicates how well agent is doing at step t
  - The agent's job is to maximize cumulative reward
  - Reinforcement learning is based on the reward hypothesis
  - Example: (2048)



Definition (Reward Hypothesis)

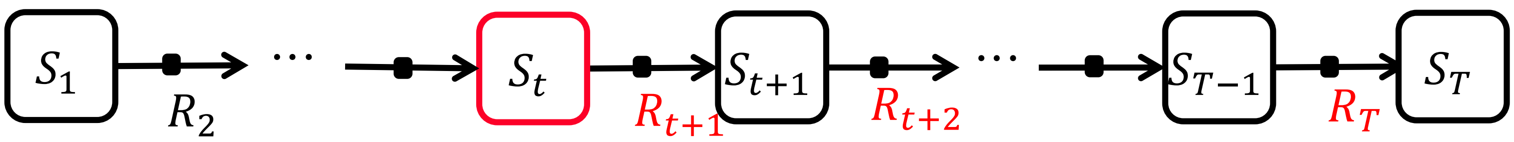
- All goals can be described by the maximization of expected cumulative reward

Rewards for Previous Examples?

- In AI, it has been used to defeat human champions at games of skill.
  - Backgammon (Tesauro, 1994).
  - Connect6/2048/Threes! (Wu et al., 2015). Reach the top levels.
  - Go programs, used in the past 10 years. (Monte-Carlo Tree Search)
  - AlphaGo, using deep reinforcement learning (2016)
- In robotics, fly stunt maneuvers in robot-controlled helicopters (Abbeel et al.) and make a humanoid robot walk.
- In economics, manage an investment portfolio (Choi et al.).
- In neuroscience, model the human brain (Schultz et al.);
- In psychology, predict animal behavior (Sutton and Barto).
- In systems, control a power station
- In engineering, it has been used to allocate bandwidth to mobile phones and to manage complex power systems (Ernst et al.).

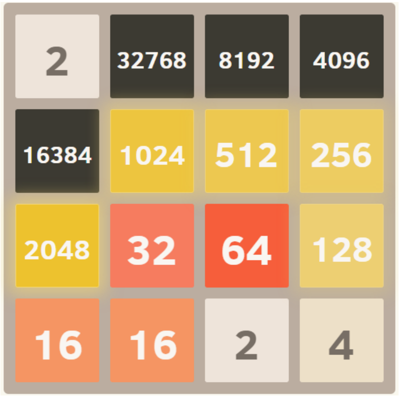
Sequential Decision Making

- Goal:
  - Select actions to maximize total future reward
- Maximize $R_{t+1} + R_{t+2} + \cdots + R_T$
  - assuming time = t.



- Notes:
  - Actions may have long term consequences
  - Reward may be delayed
  - It may be better to sacrifice immediate reward to gain more long-term reward

Sequential Decision Making – Examples

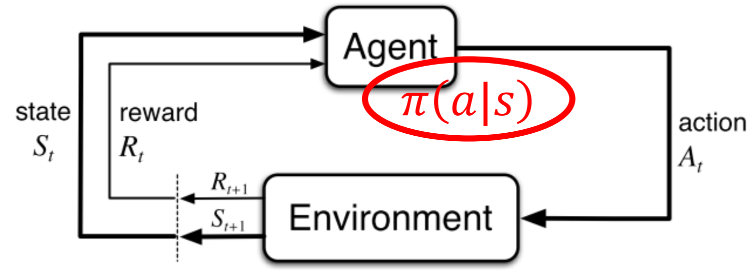


- Examples:
  - In 2048, establish a sequence of $(2^t, 2^{t-1}, 2^{t-2}, \ldots)$
  - In chess, block opponent moves to help winning chances many moves from now.
  - In a financial investment, may take months to mature
  - In robotics, refuel a helicopter to prevent a crash.

Major Components of an RL Agent

- Value function: how good is each state and/or action
- Policy: agent's behavior function
- Model: agent's representation of the environment

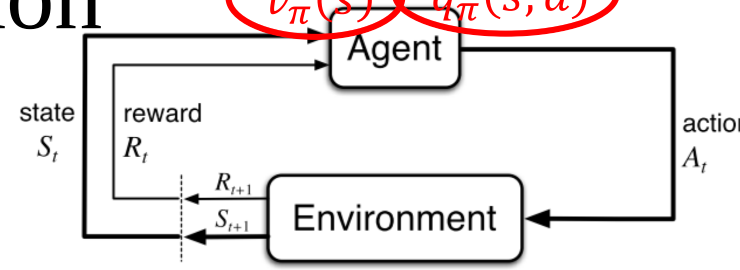
Policy



- A policy is the agent's behavior
  - It is a map from state to action,
- Policy types:
  - Deterministic policy: $a = \pi(s_i)$
  - Stochastic policy: $\pi(a|s) = \mathbb{P}[A_t = a|\ S_t = s]$
    - Sometimes, written in $\pi(s, a)$.
- Examples:
  - In 2048: Up/down/left/right
  - In robotics: angle/force/…

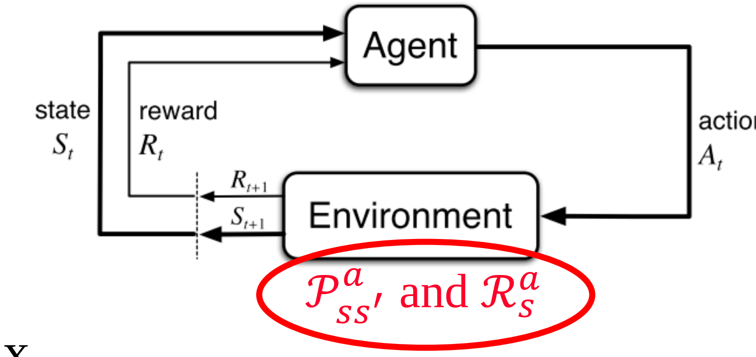
Value Function

$v_\pi(s)$ $q_\pi(s,a)$



- A value function is
  a prediction of future reward
  - Used to evaluate the goodness/badness of states
    - therefore to select between actions.
  - Return $G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots$
- Types of value functions under policy π:
  - State value function: the expected return from s.
    $$v_\pi(s) = \mathbb{E}_\pi[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots \mid S_t = s] = \mathbb{E}_\pi[G_t \mid S_t = s]$$
  - Q-Value function: the expected return from s taking action a.
    $$q_\pi(s,a) = \mathbb{E}_\pi[G_t \mid S_t = s, A_t = a]$$
- Examples:
  - In 2048, the expected score from a board S_t.

Model



- A model predicts what the environment will do next
  - $\mathcal{P}$ is a state transition probability matrix,
    $\mathcal{P}^a_{ss'} = \mathbb{P}[S_{t+1} = s' \mid S_t = s, A_t = a]$
    - predicts the next state
  - $\mathcal{R}$ is a reward function,
    $\mathcal{R}^a_s = \mathbb{E}[R_{t+1} \mid S_t = s, A_t = a]$
    - predicts the next (immediate) reward
- Examples:
  - In 2048:
    - After a move, $\mathcal{P}$ is to generate a tile randomly as follows:
      - 2-tile: with probability of 9/10
      - 4-tile: with probability of 1/10

Categorizing RL Agents (Policy & Value)

- Value Based
  - No Policy (Implicit)
  - Value Function
- Policy Based
  - Policy
  - No Value Function (Implicit)
- Actor Critic
  - Policy
  - Value Function

Categorizing RL Agents (Model)

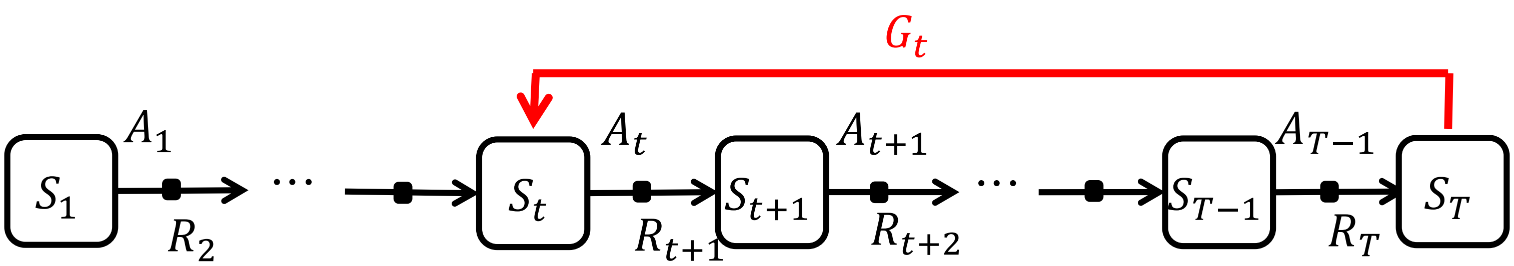
- Model Free
  - Policy and/or Value Function
  - No Model
- Model Based
  - Policy and/or Value Function
  - Model

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.

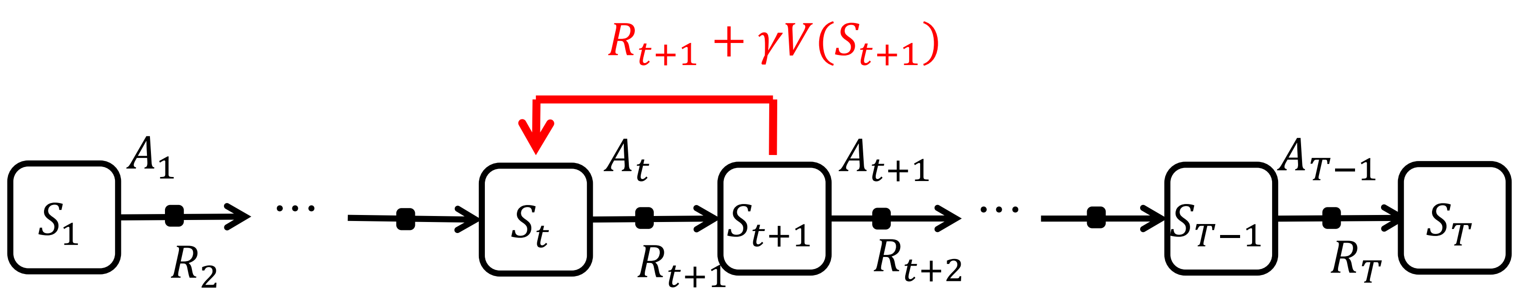
Monte-Carlo Learning

- 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



Case Studies

I-Chen Wu

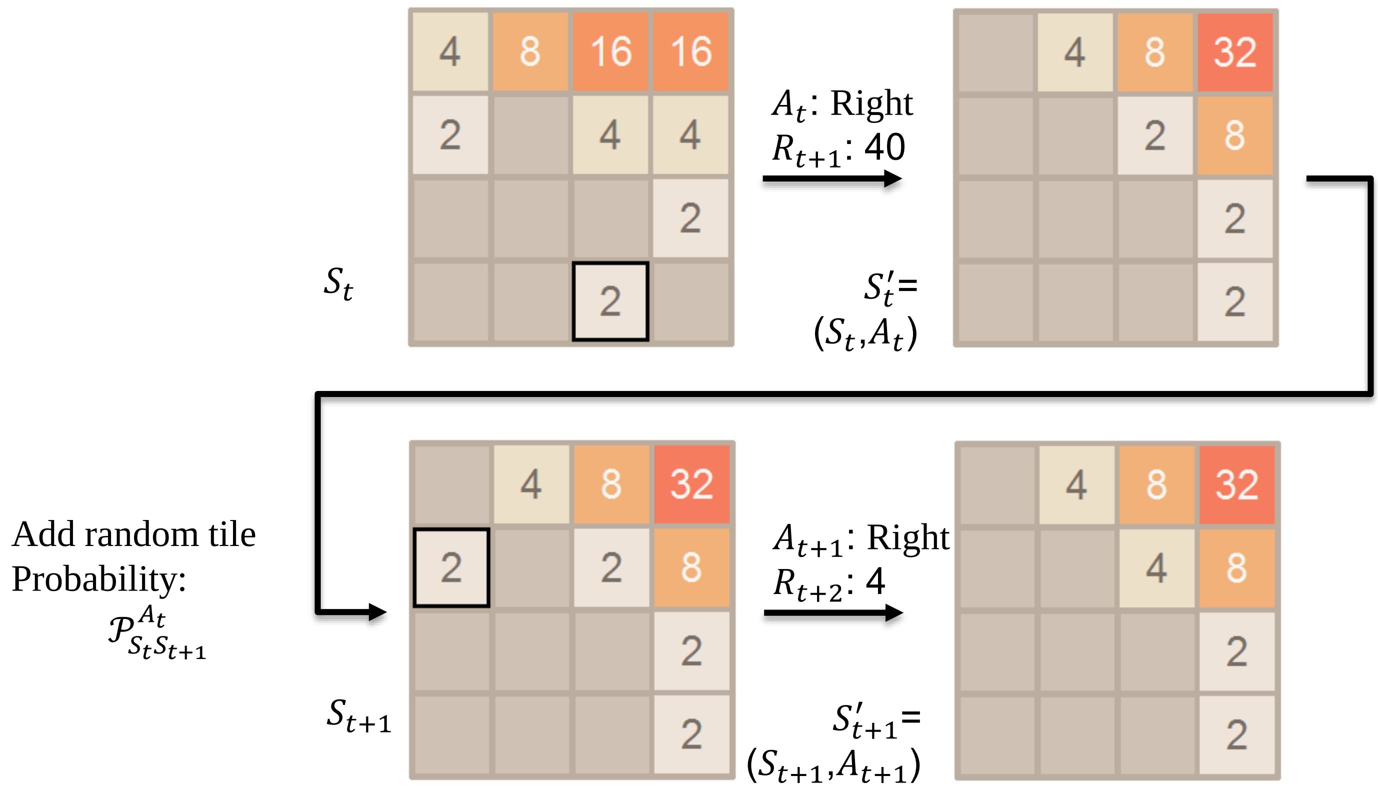
- David Silver, Online Course for Deep Reinforcement Learning.
  - http://www.cs.ucl.ac.uk/staff/D.Silver/web/Teaching.html
- M. Szubert and W. Jaśkowski, "Temporal difference learning of n-tuple networks for the game 2048," *2014 IEEE Conference on Computational Intelligence and Games (CIG)*, Aug. 2014, pp. 1–8.
- Kun-Hao Yeh, et al., Multi-Stage Temporal Difference Learning for 2048-like Games, accepted by IEEE Transactions on Computational Intelligence and AI in Games (SCI), doi: 10.1109/TCIAIG.2016.2593710, 2016.
- Mnih, V. et al. Human-level control through deep reinforcement learning. Nature 518, 529–533 (2015).

Cases

- 2048
  - Temporal Difference (TD) Learning
  - N-tuple networks
- Atari games
  - Temporal Difference (TD) Learning
  - Deep Q-networks (DQN), a kind of Deep NN
- Go Programs (with Monte-Carlo Tree Search)
  - Monte-Carlo (MC) Learning
  - Multi-Armed Bandits
  - Planning
- AlphaGo (with Reinforcement Learning) – to be added.
  - Monte-Carlo (MC) Learning
  - Policy Gradient
- Pole Balancing – to be added.
  - Policy Gradient
  - Actor-Critic

Case Study: 2048

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



2048 RL Agent

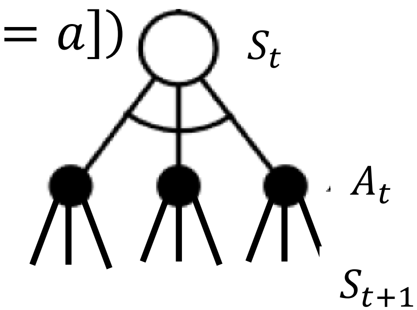
- Value function:
  - The expected score/return G_t from a board S
  - But, #states is huge
    - About 17^{16} (=10^{20}).
      - Empty, 2 (=2^1), 4 (=2^2), 8 (=2^3), …, 65536 (=2^{16}).
  - Need to use value function approximator.
- Policy:
  - Simply choose the action (move) with the maximal value 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.

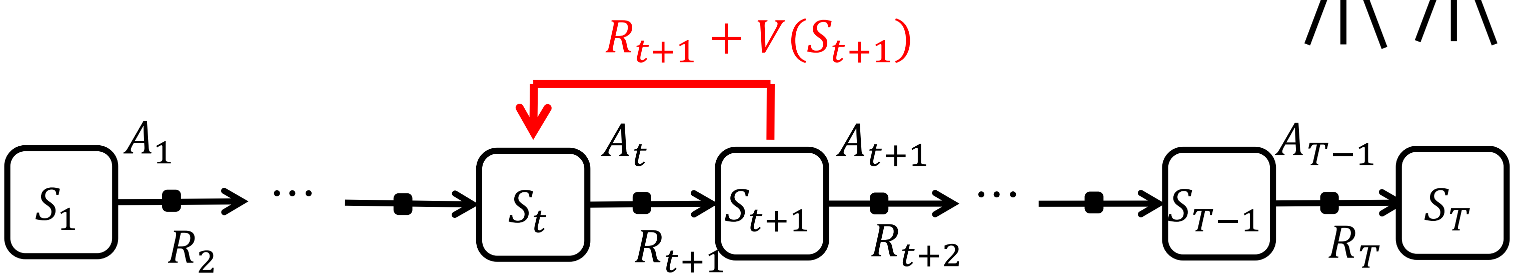
TD Learning in 2048

- State value function: (Normally $\gamma = 1$)
  - Update value $V(S_t)$ toward TD target $R_{t+1} + \gamma V(S_{t+1})$

    $$V(S_t) \leftarrow V(S_t) + \alpha(R_{t+1} + \gamma V(S_{t+1}) - V(S_t))$$
- 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.

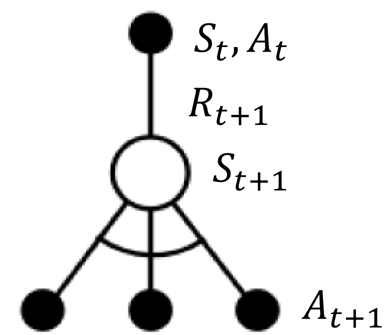




Q-Learning in 2048

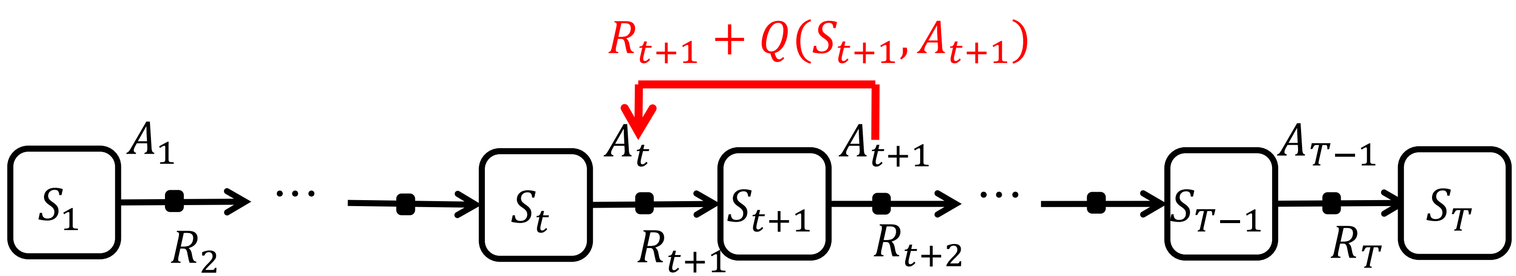
- Q-value function: (Normally $\gamma = 1$)
  - Update value $Q(S_t, A_t)$ toward TD target $R_{t+1} + \gamma \max_a Q(S_{t+1}, a)$

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



- Making decision (based on value).

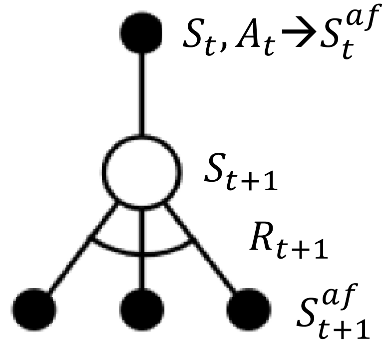
  $\pi(s) = argmax_a(Q(S_t, a))$
  - more efficient.
  - A minor problem: Four times more memory



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 $V^{af}(S_t^{af})$ toward TD target $\gamma \max_a (R_{t+1} + V^{af}(S_{t+1}^{af}))$

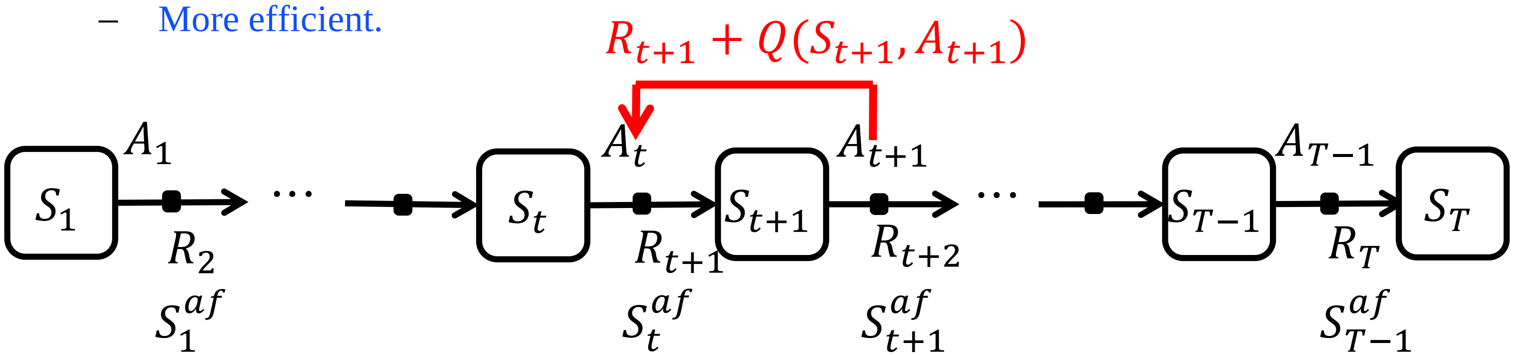
$$V^{af}(S_t^{af}) \leftarrow V^{af}(S_t^{af}) + \alpha(\gamma \max_a (R_{t+1} + V^{af}(S_{t+1}^{af})) - V^{af}(S_t^{af}))$$



- Making decision (based on value).

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

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



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.

Linear Value Function Approximation

- Represent value function by a linear combination of features

  $$\hat{v}(S;\theta) = x(S)^{\mathrm{T}}\theta = \sum_{j=1}^{n} x_j(S)\theta_j$$

- Gradient of $\hat{v}(S,\theta)$:

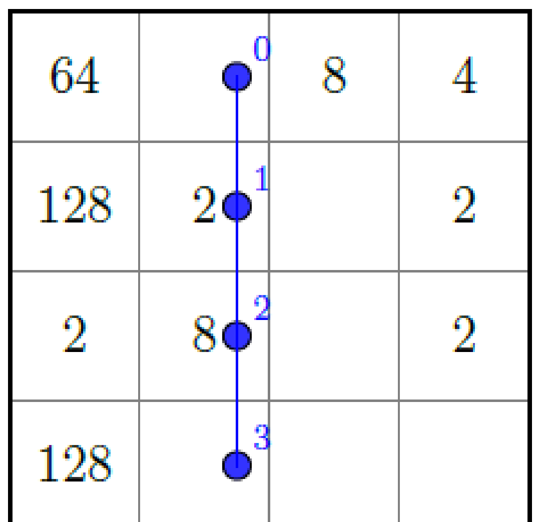
  $$\nabla_\theta \hat{v}(S,\theta) = x(S)$$

Gradient Descent

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

N-Tuple Network

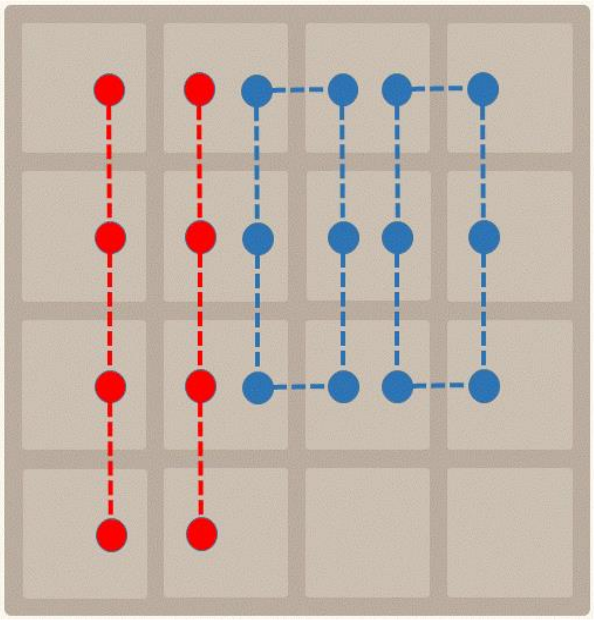
- Example: 4-tuple networks as shown.
  - Each cell has 16 different tiles
  - 16^4 features for this network.
    - But only one is on, others are 0.
    - So, we can use table lookup to find the feature weight.

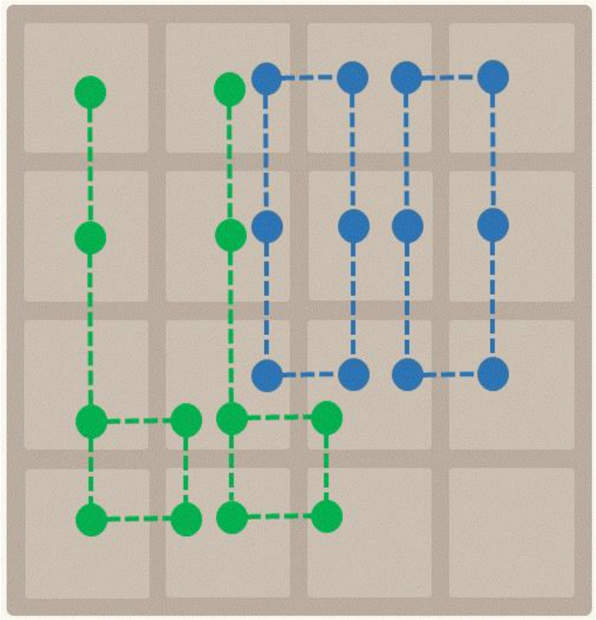


0123	weight
0000	3.04
0001	−3.90
0002	−2.14
⋮	⋮
0010	5.89
⋮	⋮
0130	**-2.01**
⋮	⋮

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 n-tuple networks, simply update values with $\alpha\Delta V$ at $LUT_i[index(s_i)]$
- 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 $s_1, s_2, s_3, s_4, s_5, s_6, s_7, s_8$.
  - 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 $LUT_i[index(s_i)]$.

Afterstate Evaluation Function

function EVALUATE(s, a)
    $s', r \leftarrow$ COMPUTE AFTERSTATE(s, a)
    return $r + V(s')$

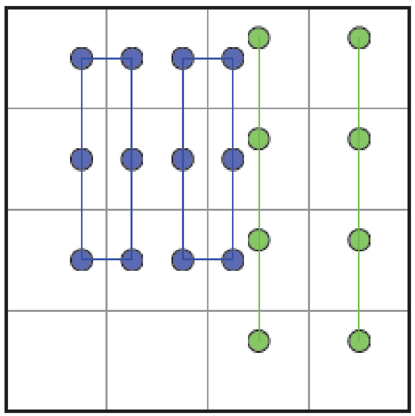
function LEARN EVALUATION(s, a, r, s', s'')
    $a_{next} \leftarrow \arg\max_{a' \in A(s'')}$ EVALUATE(s'', a')
    $s'_{next}, r_{next} \leftarrow$ COMPUTE AFTERSTATE(s'', a_{next})
    $V(s') \leftarrow V(s') + \alpha(r_{next} + V(s'_{next}) - V(s'))$

function PLAY GAME
    $score \leftarrow 0$
    $s \leftarrow$ INITIALIZE GAME STATE
    while $\neg$IS TERMINAL STATE(s) **do**
        $a \leftarrow \arg\max_{a' \in A(s)}$ EVALUATE(s, a')
        $r, s', s'' \leftarrow$ MAKE MOVE(s, a)
        if LEARNING ENABLED **then**
            LEARN EVALUATION(s, a, r, s', s'')
        $score \leftarrow score + r$
        $s \leftarrow s''$
    return $score$

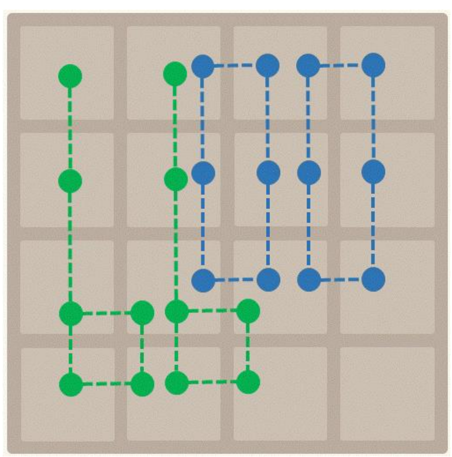
function MAKE MOVE(s, a)
    $s', r \leftarrow$ COMPUTE AFTERSTATE(s, a)
    $s'' \leftarrow$ ADD RANDOM TILE(s')
    return (r, s', s'')

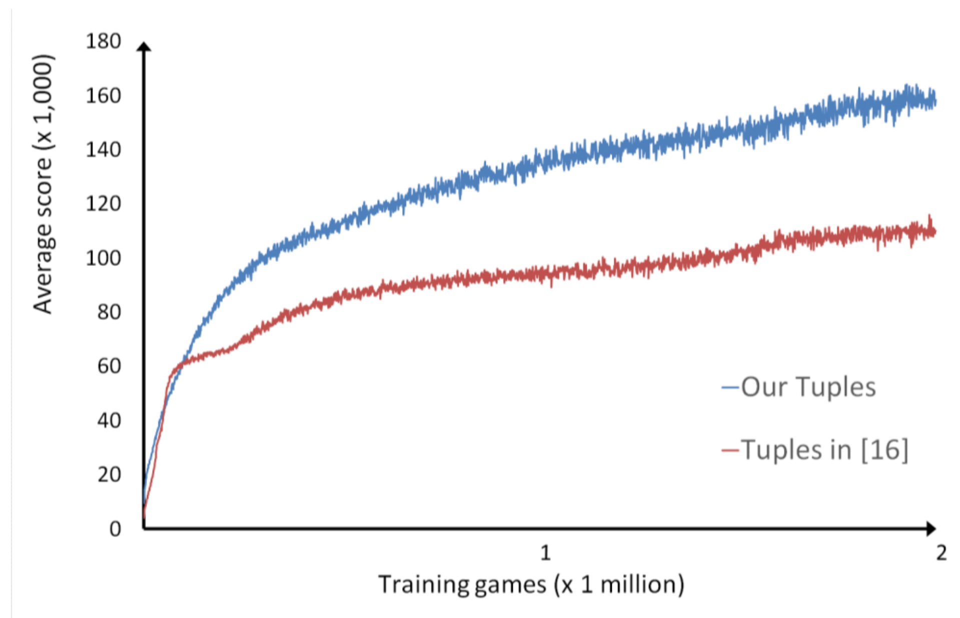
The N-Tuple Networks Used

- Use the following [Szubert and Jaskowaski 2014]



- Ours:





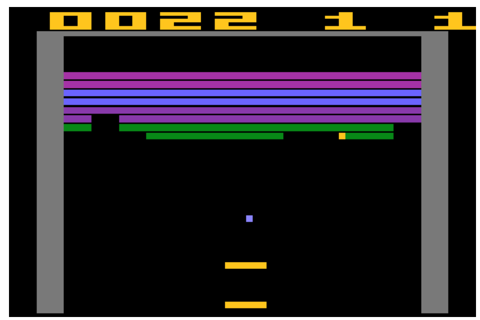
Performance Results (without search)

2048 rate	**100%**
4096 rate	100%
8192 rate	99.20%
16384 rate	83.30%
32768 rate	8.10%
Maximum score	607488
Average score	331820

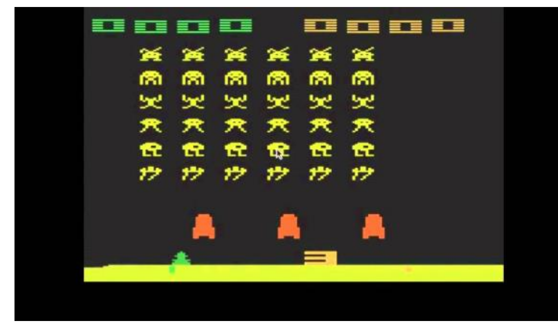
Case Study: Atari 2600 Games

- Learn to play Atari games from video only (without knowing the game a priori)

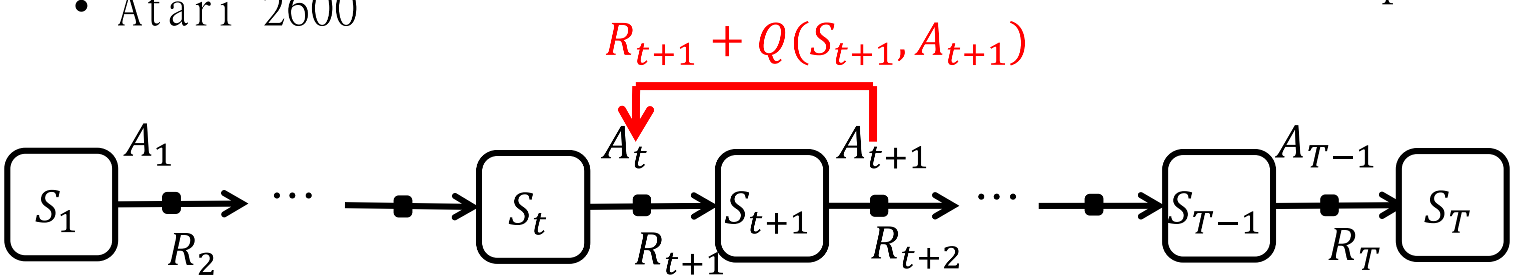
- Atari 2600



- Breakout



- Space Invaders



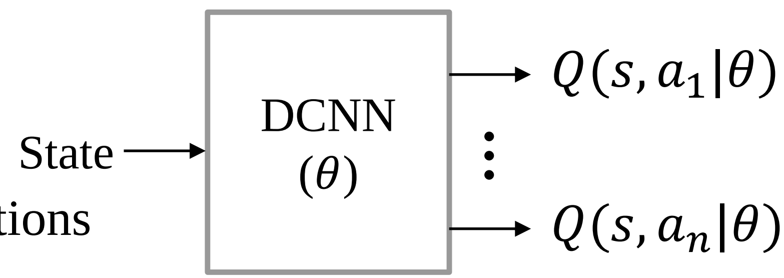
Deep Q-Networks (DQN)

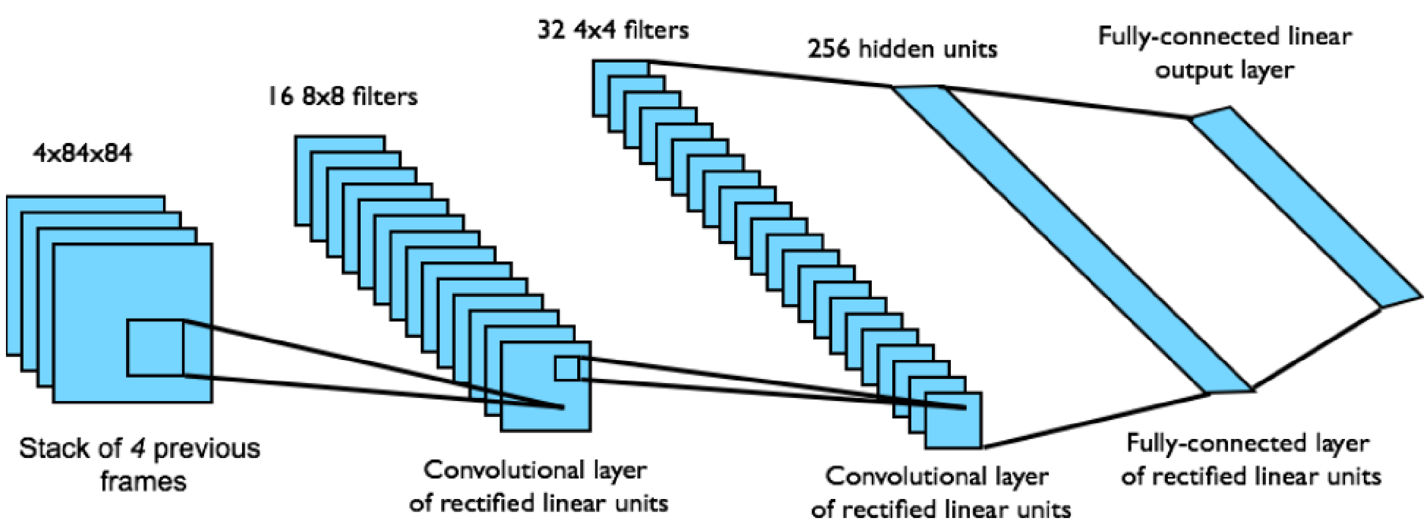
DQN uses experience replay and fixed Q-targets

- Take action according to ϵ-greedy policy
- Store transition $(s_t, a_t, r_{t+1}, s_{t+1})$ in replay memory $\mathcal{D}$
- Sample random mini-batch of transitions (s, a, r, s') from $\mathcal{D}$
- Compute Q-learning targets w.r.t. old, fixed parameters θ^-
- Optimize MSE between Q-network and Q-learning targets
  - Minimize a sequence of loss functions $\mathcal{L}(\theta_i)$ that changes at each iteration i.
  - $\mathcal{L}_i(\theta_i) = \mathbb{E}_{s,a,r,s' \sim \mathcal{D}_i}\left[\left(r + \gamma \max_{a'} Q(s', a'; \theta_i^-) - Q(s, a; \theta_i)\right)^2\right]$
- Using variant of stochastic gradient descent
  - Differentiating the loss function with respect to the weights we arrive at the following gradient
  - $\nabla_{\theta_i}\mathcal{L}_i(\theta_i) = \mathbb{E}_{s,a,r,s' \sim \mathcal{D}_i}\left[\left(r + \gamma \max_{a'} Q(s', a'; \theta_i^-) - Q(s, a; \theta_i)\right) \cdot \nabla_{\theta_i} Q(s, a; \theta_i)\right]$

DQN in Atari

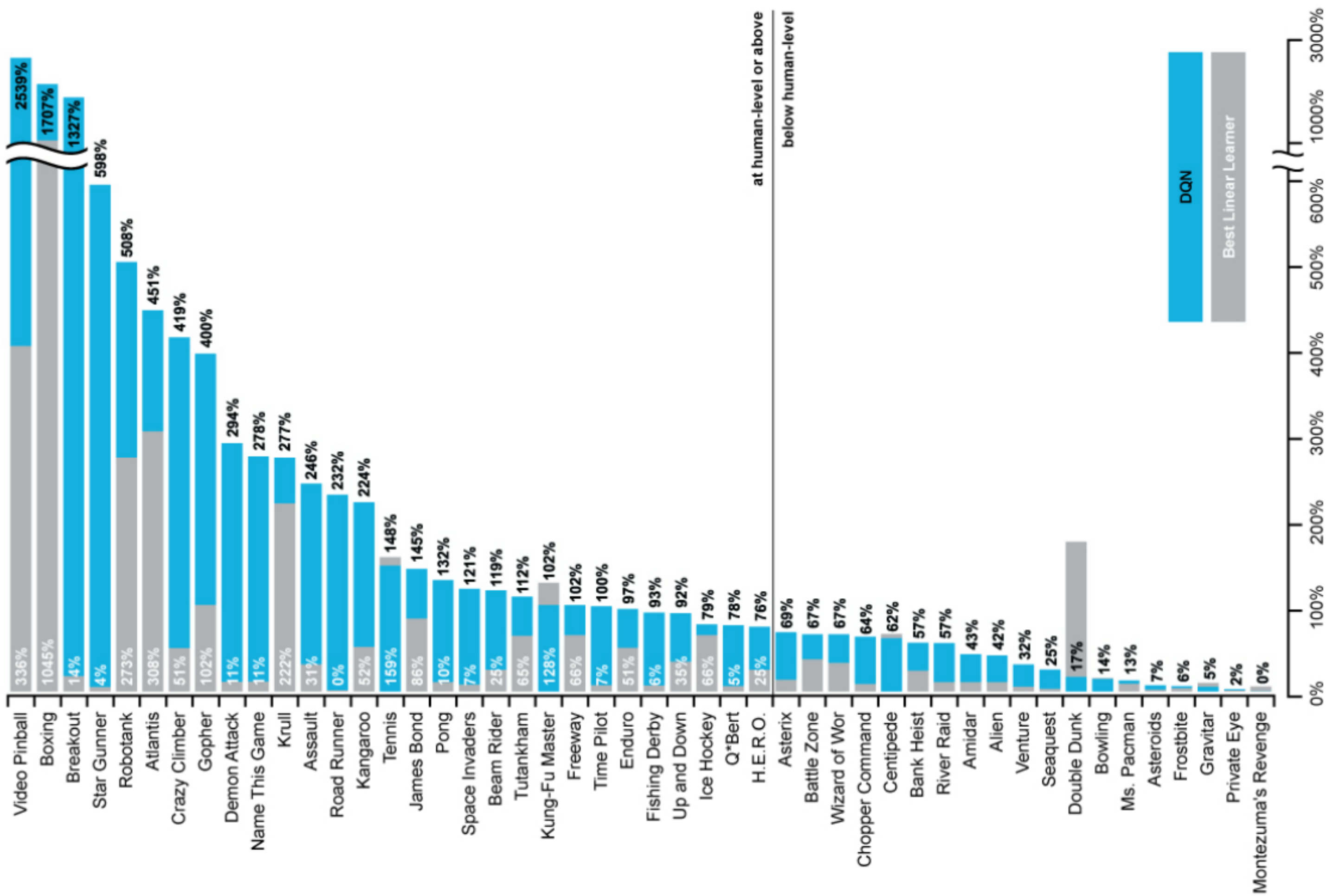
- End-to-end learning of values $Q(s,a)$ from pixels s
- Input state s
  - stack of raw pixels from last 4 frames
- Output
  - $Q(s,a_i|\theta)$ for 18 joystick/button positions
- Reward
  - change in score for that step





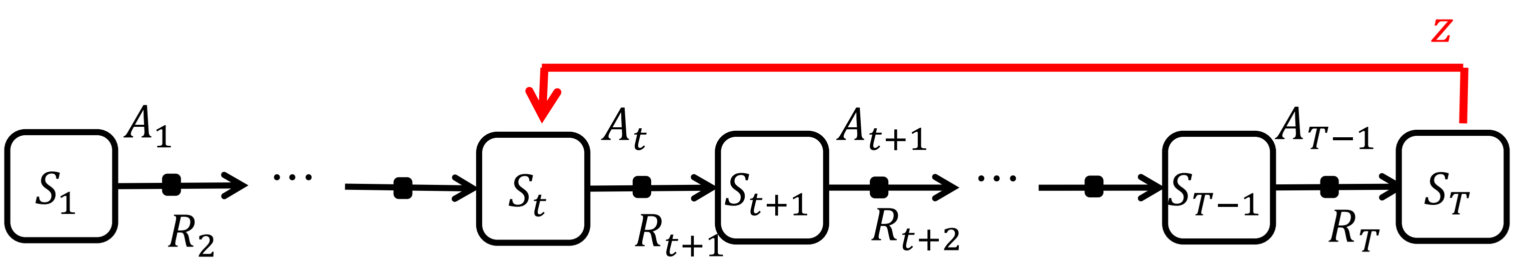
Performance of Deep Q-Learning

- Left (stronger than human)



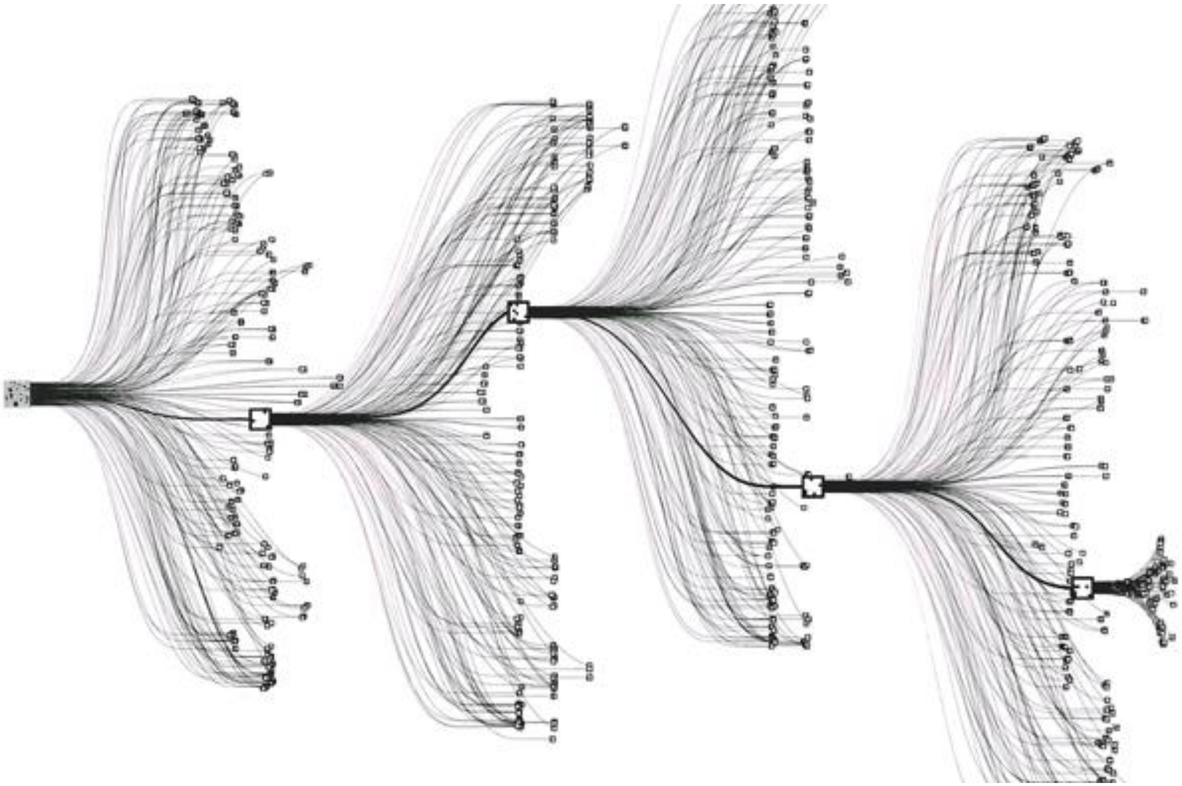
Case Study: Go (MCTS)

- 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 decade.
- And also applied to others successfully.
  - Other games like Havannah, Hex, GGP
  - Other applications, like mathematical optimization problems (scheduling, UCP, camera coverage).



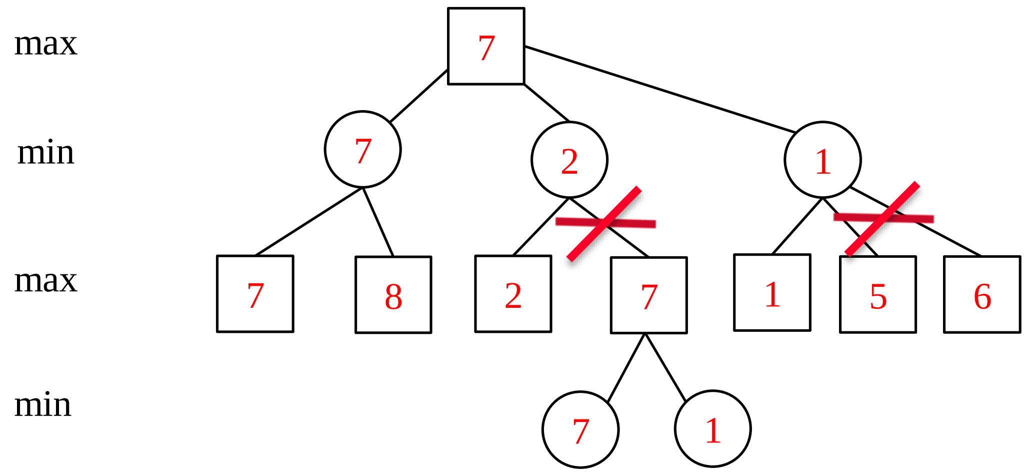
Go – One of the Most Popular Games

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



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
  - …



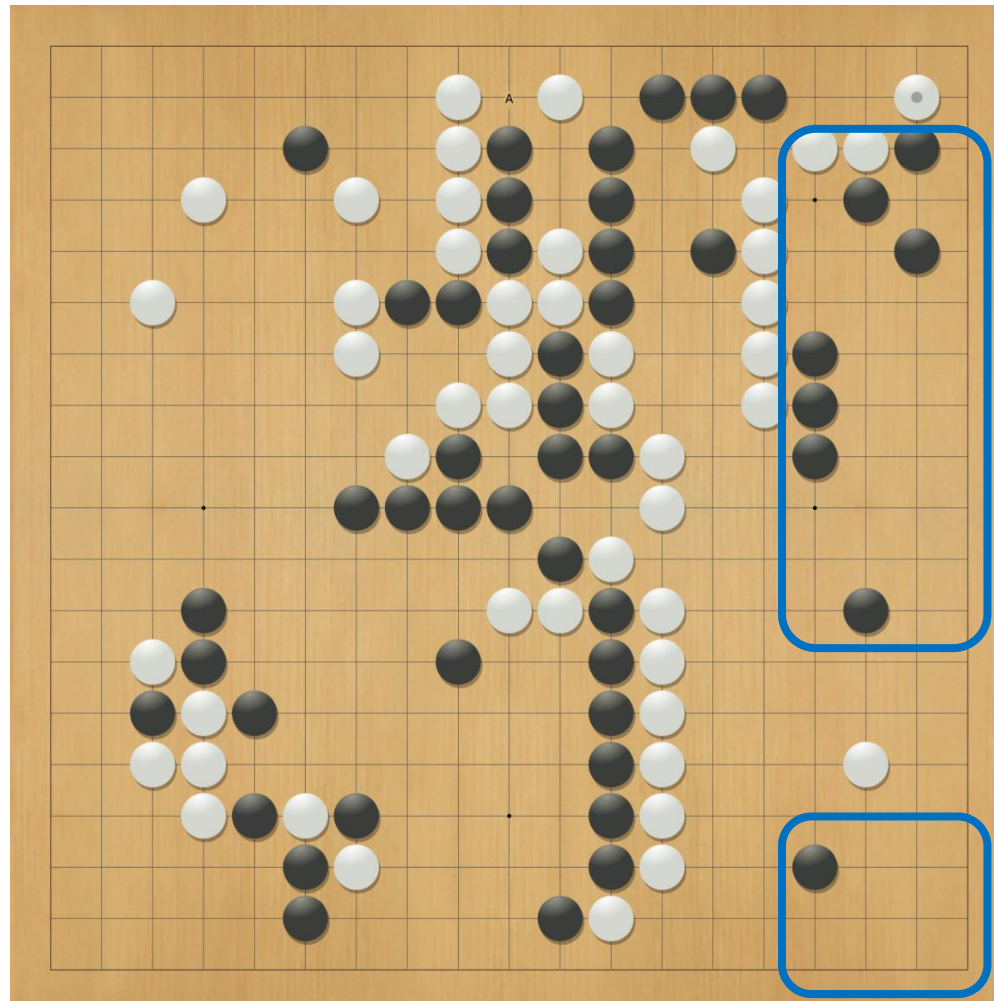
- Problem for chess:
  - need to consult with experts for feature values.

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,

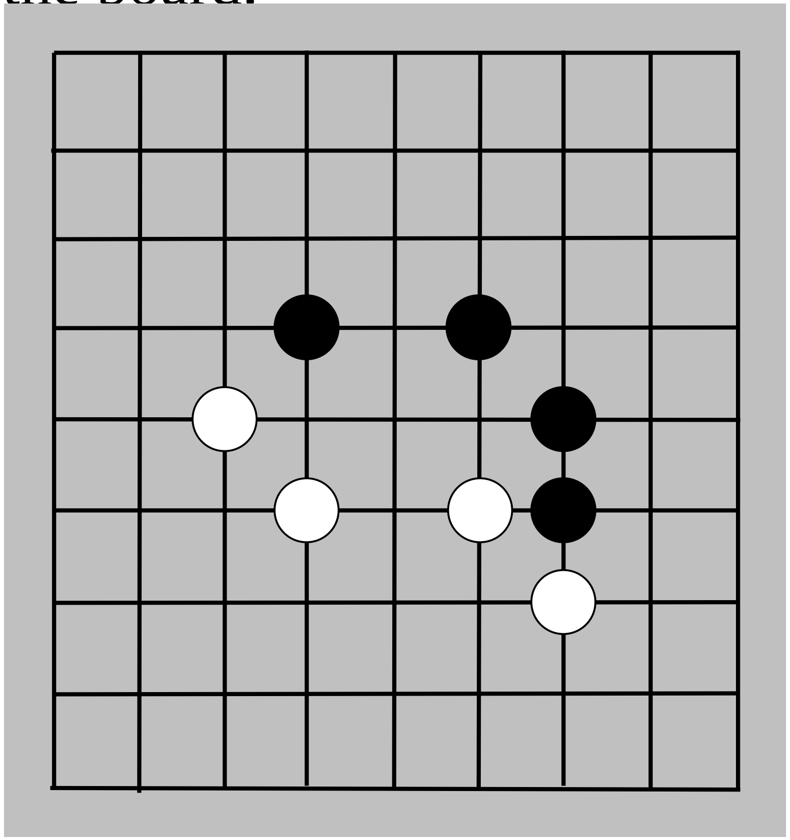
- Why not use Monte-Carlo?
- Calculate it based on stochastics.



Game 1: AlphaGo vs. 李世石

Rules Overview Through a Game (opening 1)

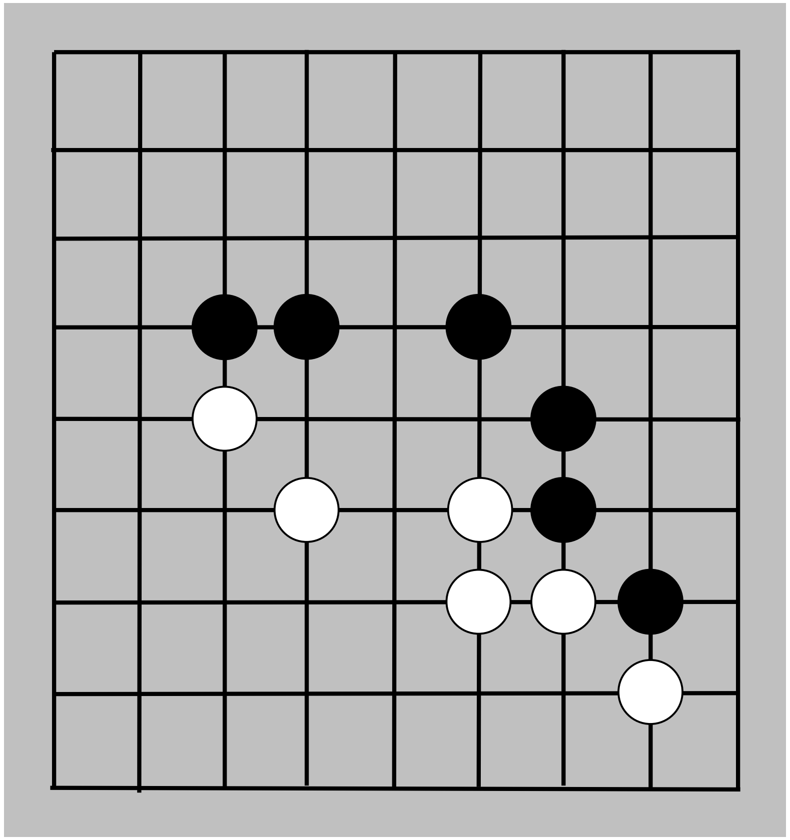
- Black/White move alternately by putting one stone on an intersection of the board.



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

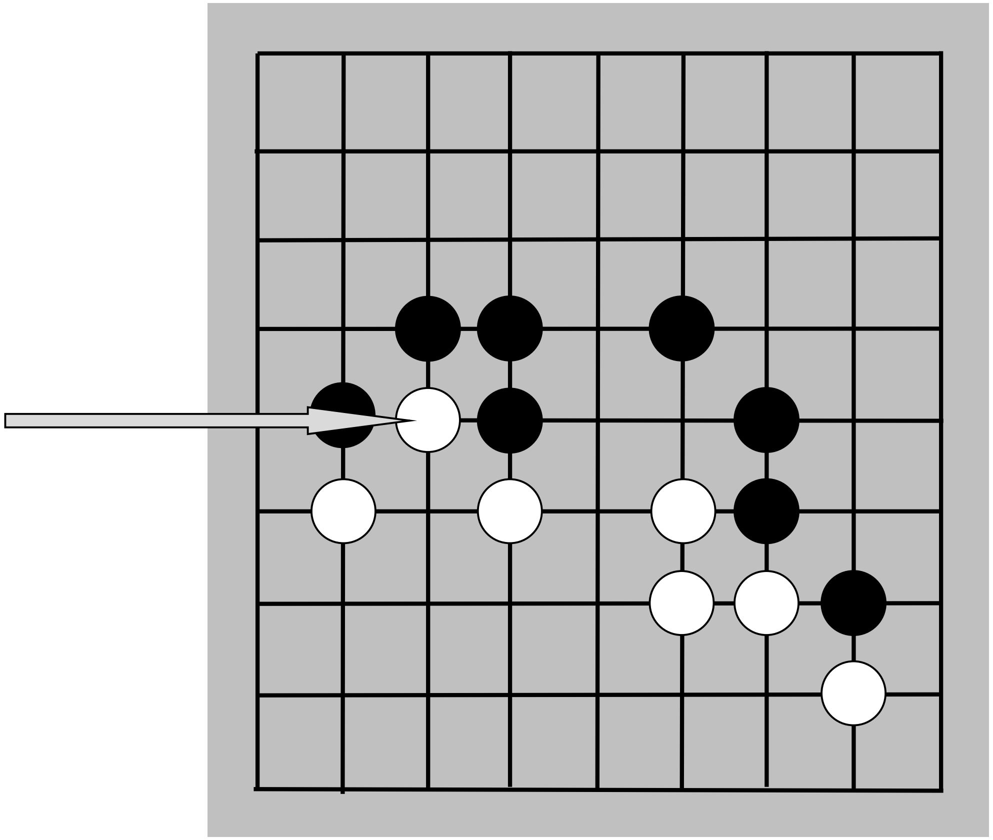
Rules Overview Through a Game (opening 2)

- Black and White aims at surrounding large « zones »



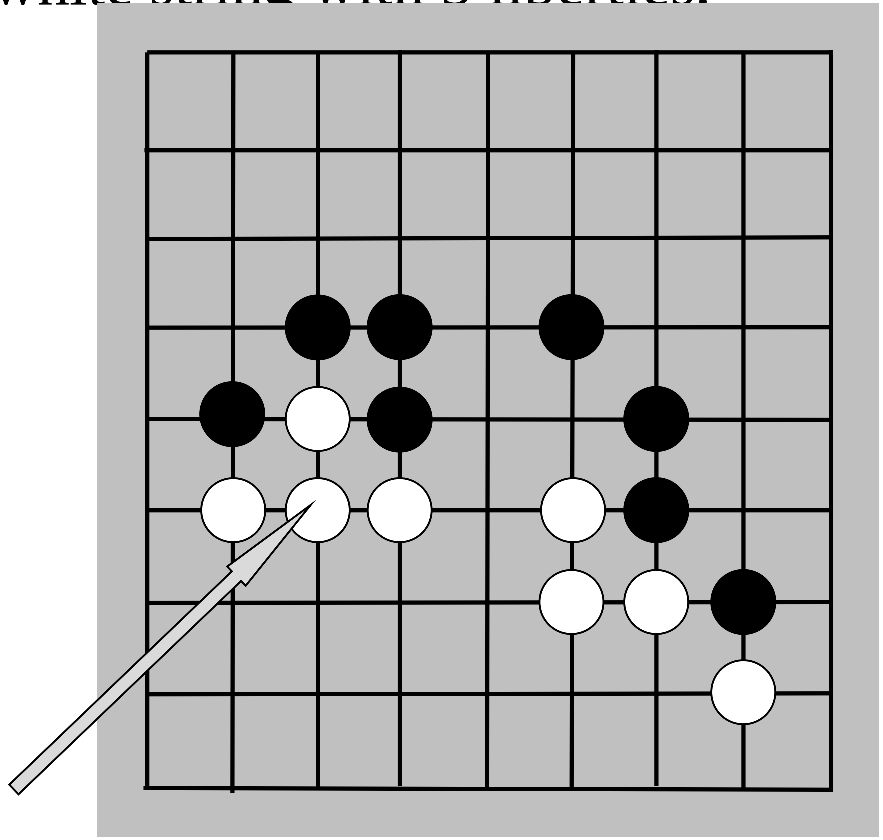
Rules Overview Through a Game (atari 1)

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



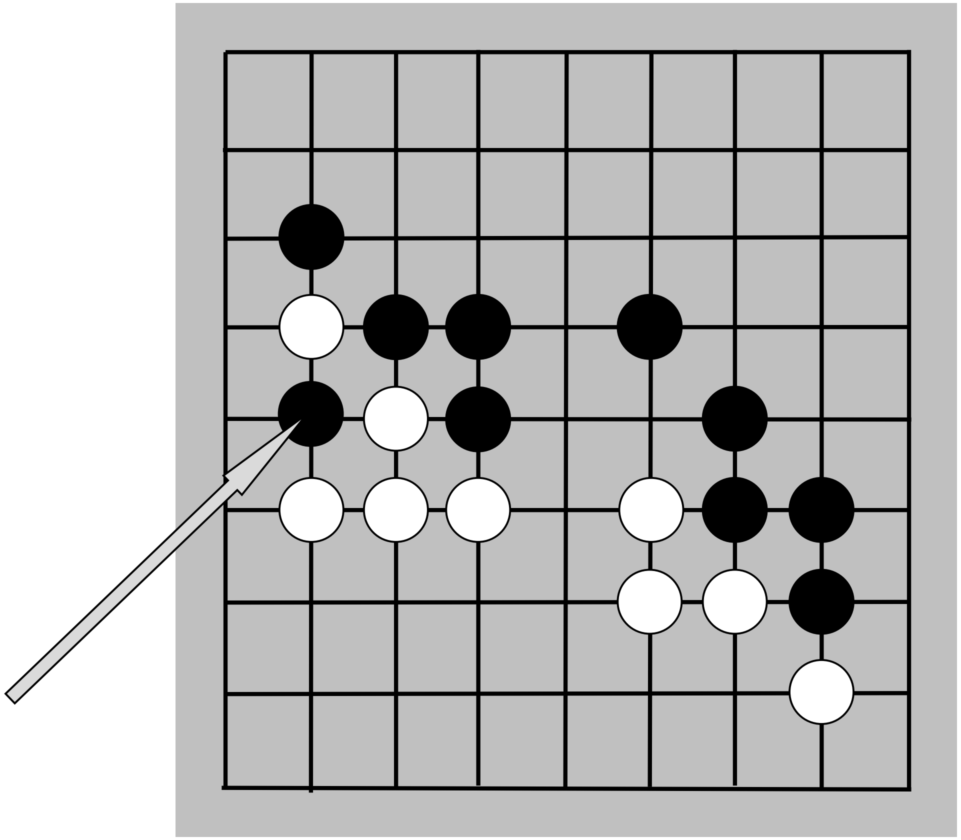
Rules Overview Through a Game (defense)

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



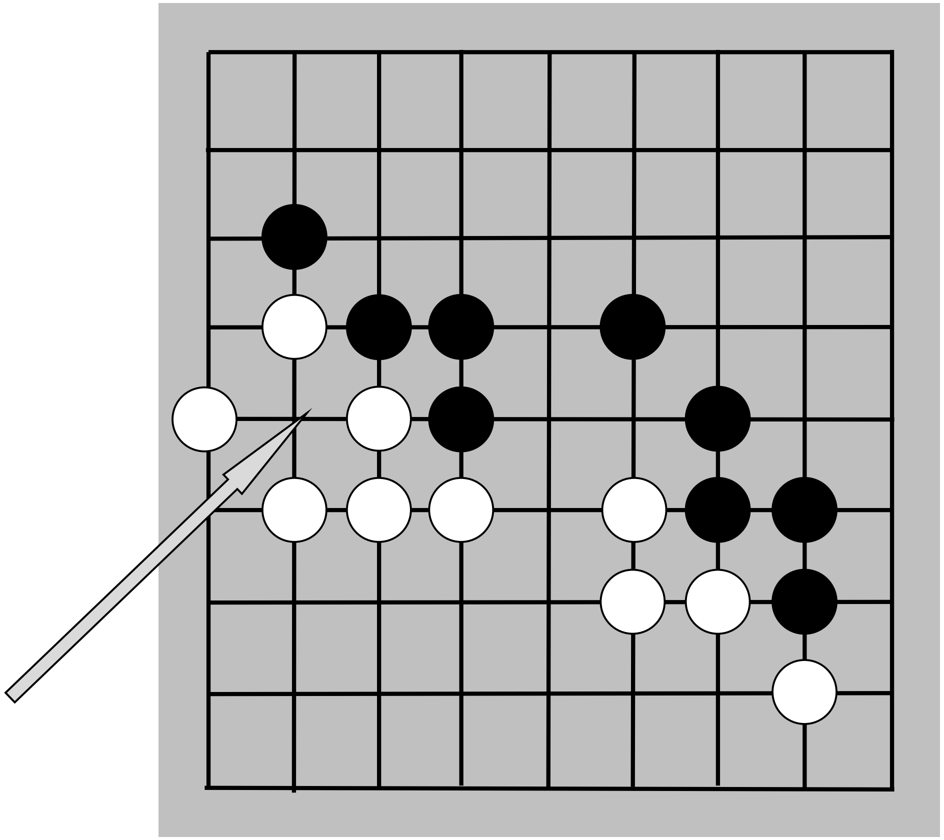
Rules Overview Through a Game (atari 2)

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



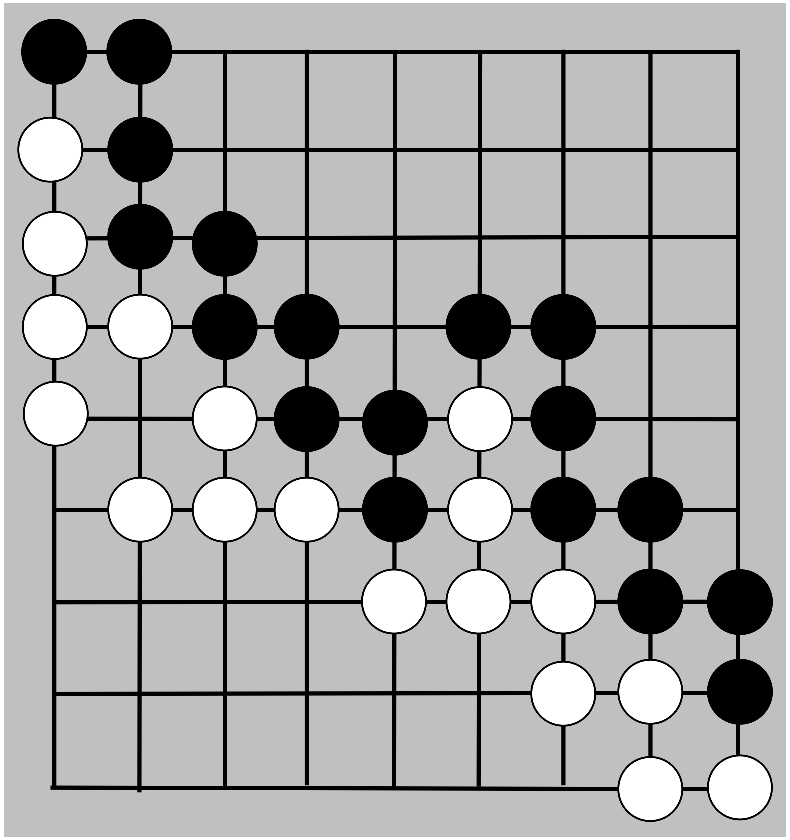
Rules Overview Through a Game (capture 1)

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



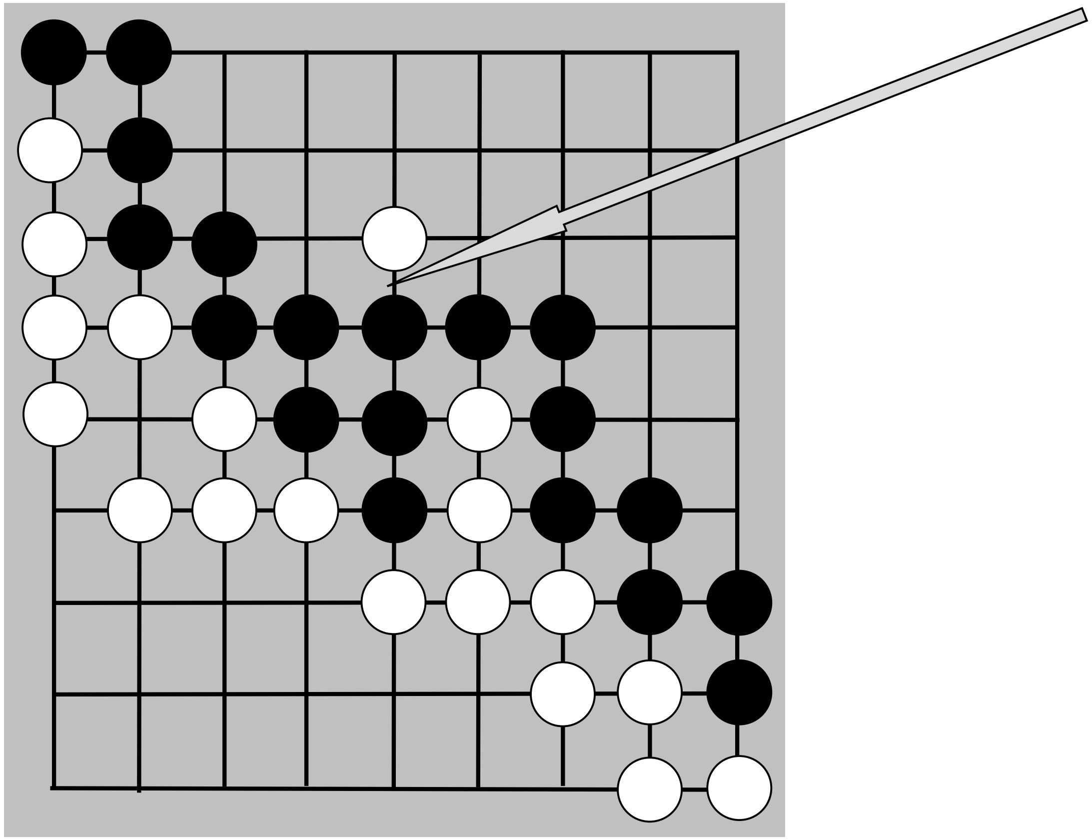
Rules Overview Through a Game (human end of game)

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



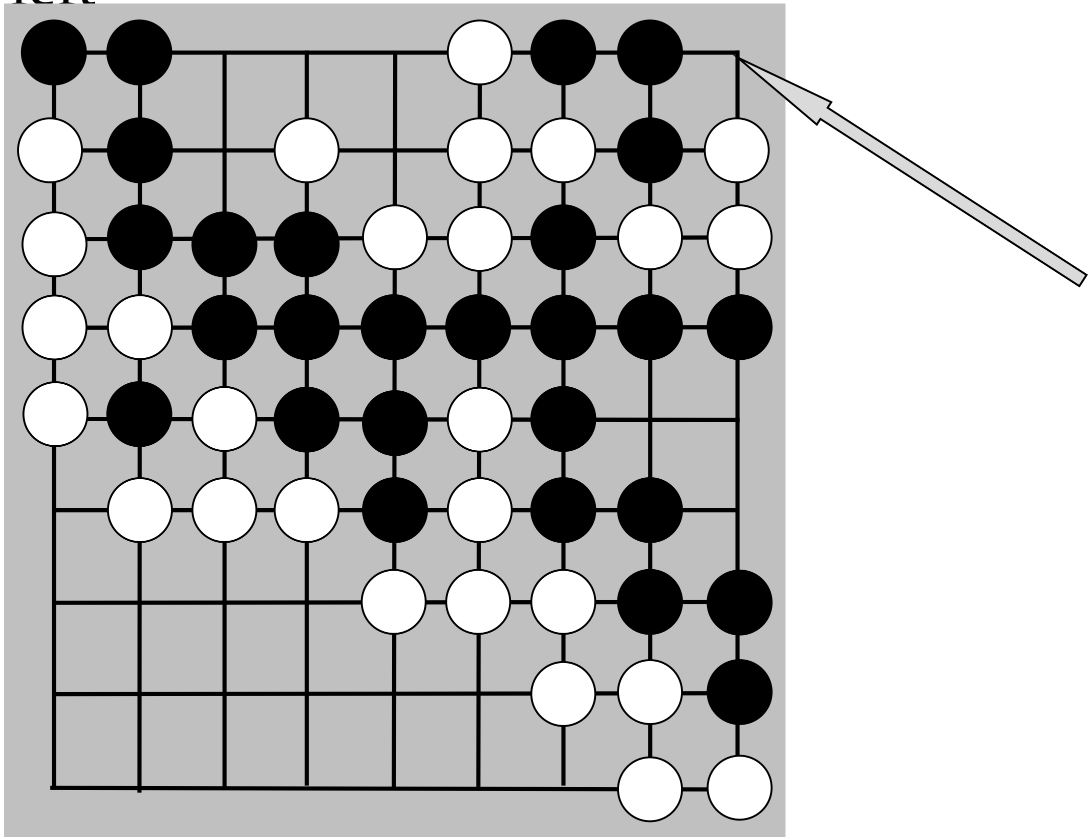
Rules Overview Through a Game (contestation 1)

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



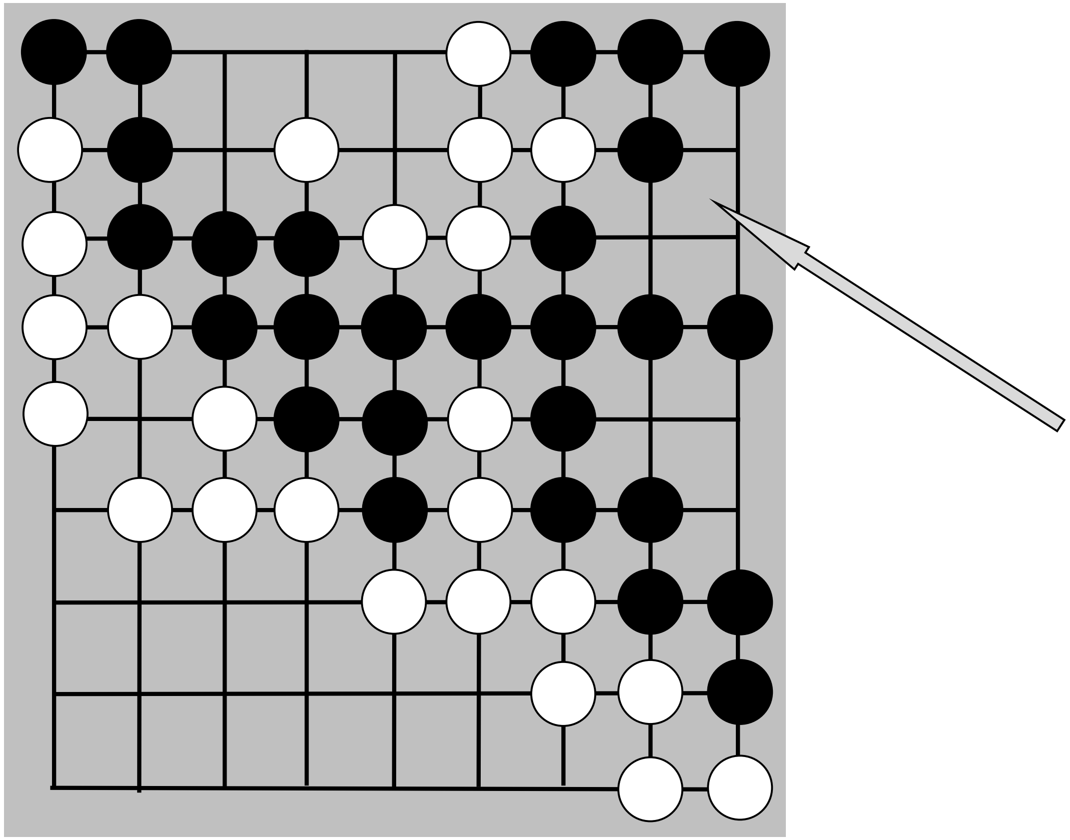
Rules Overview Through a Game (contestation 2)

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



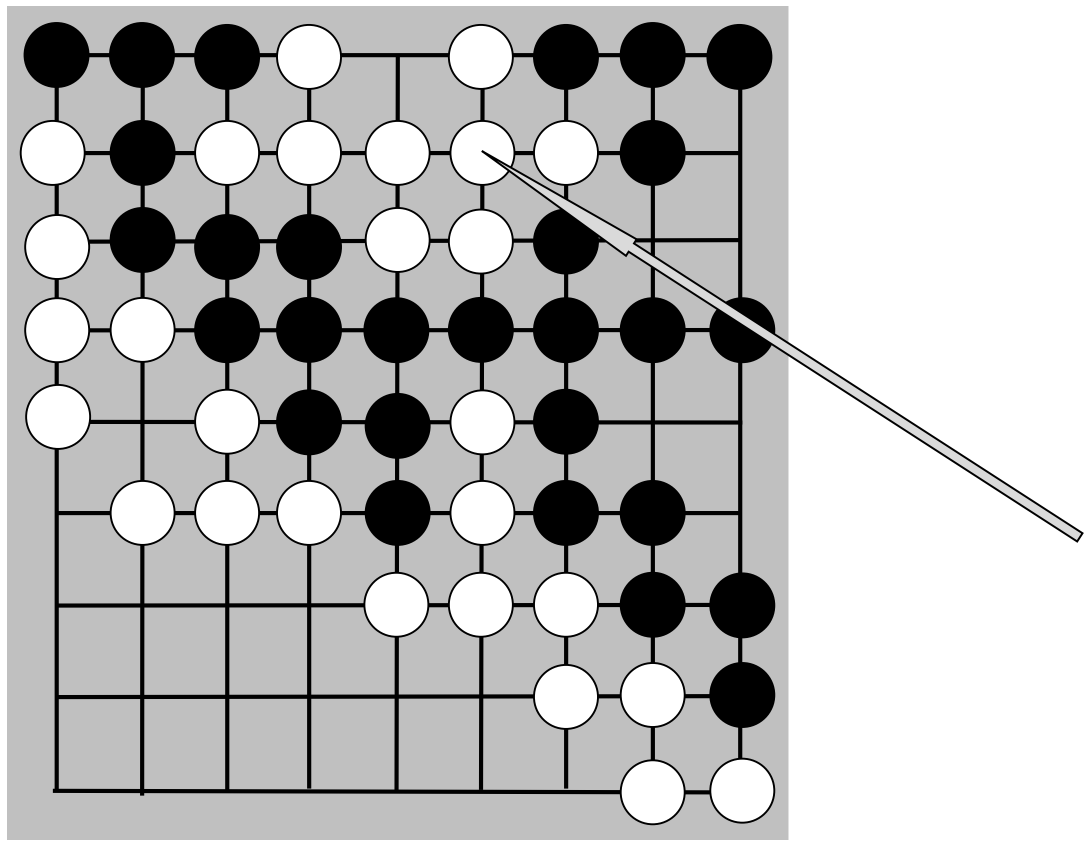
Rules Overview Through a Game (follow up 1)

- Black has captured the 3-stone white string



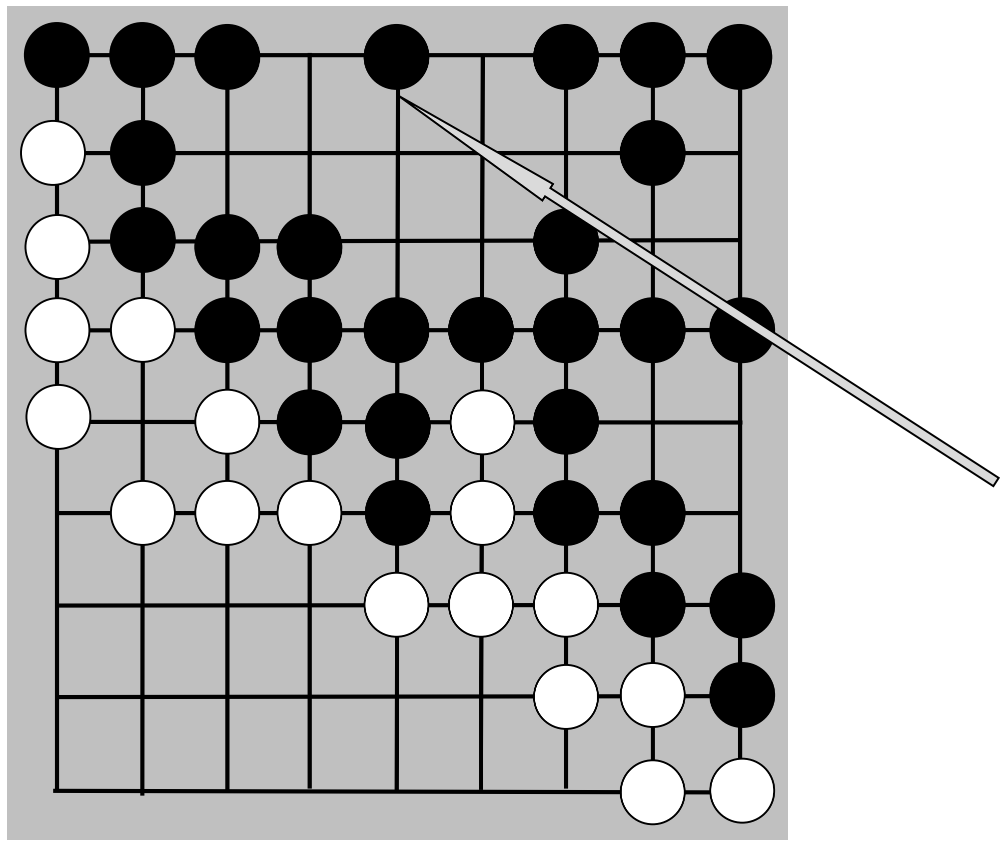
Rules Overview Through a Game (follow up 2)

- White lacks liberties…



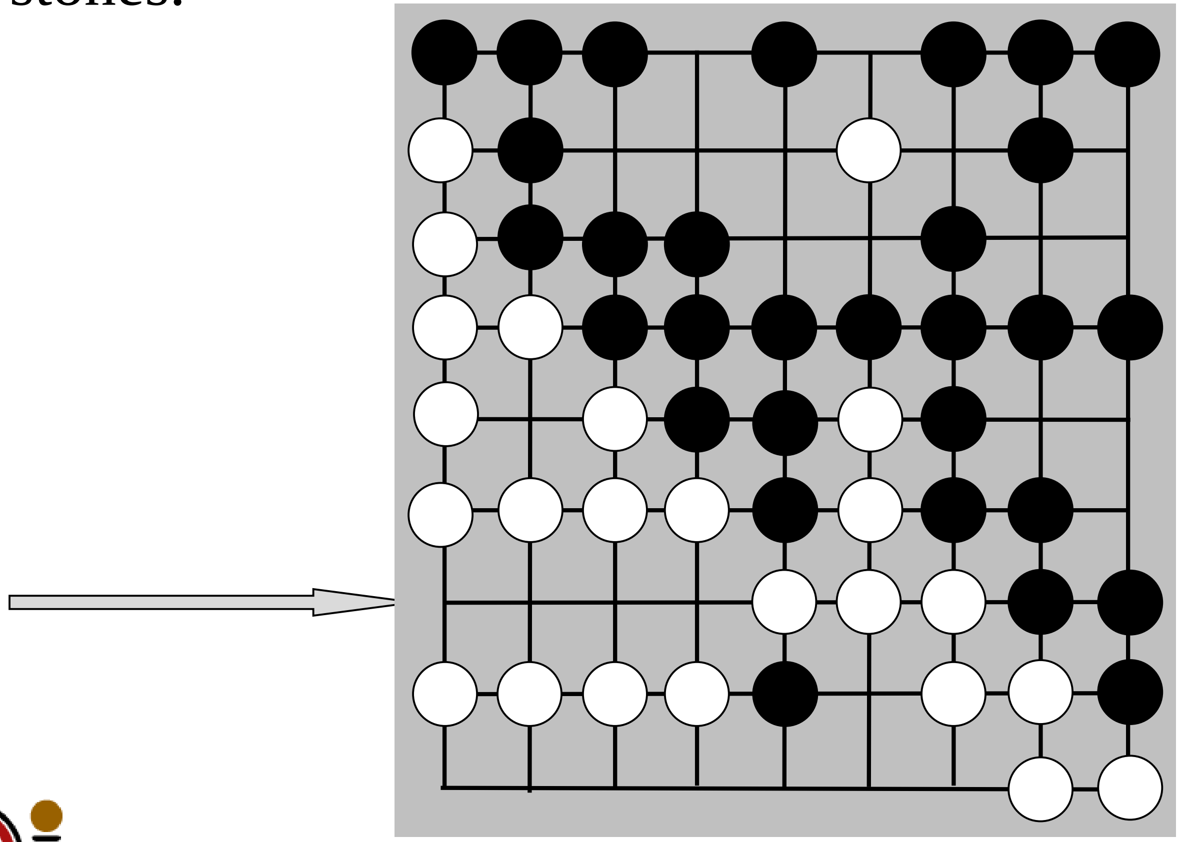
Rules Overview Through a Game (follow up 3)

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



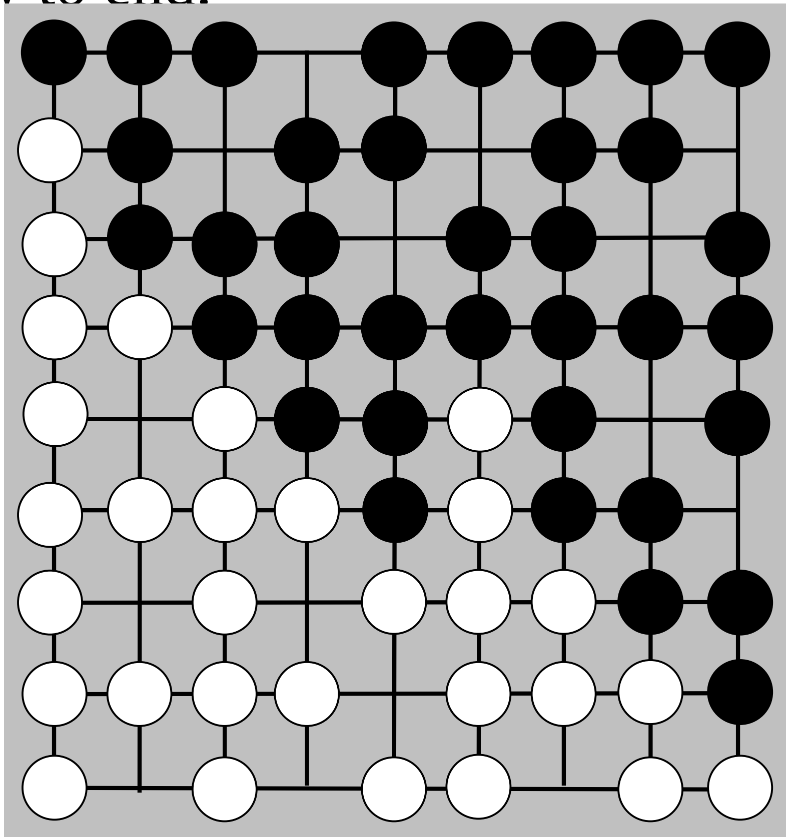
Rules Overview Through a Game (follow up 4)

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



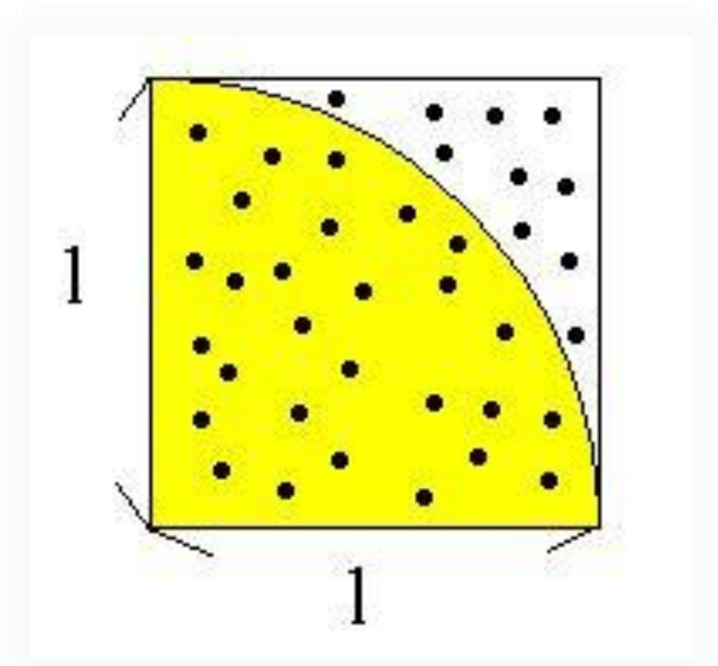
Rules Overview Through a Game (concrete end of game)

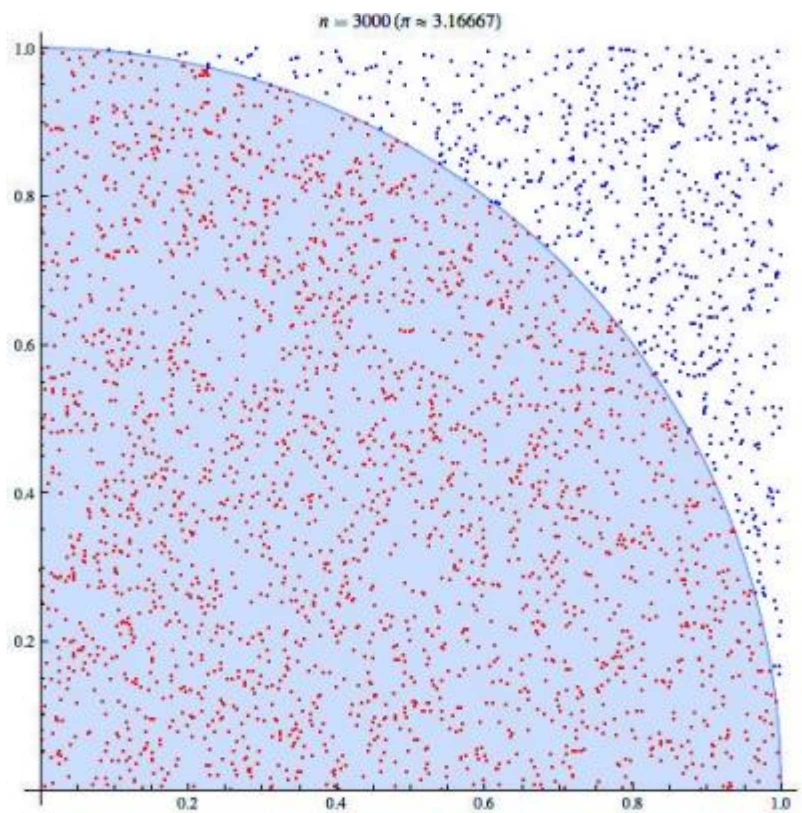
- The board is covered with either stones or « eyes ». Programs know to end.



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]
- Observed rewards when playing machine i: $X_{i,1}, X_{i,2}, \ldots$
- Initialization: Play each machine once.
- Loop:
  - Play machine j that maximizes,

$$\bar{X}_j + \sqrt{\frac{2\log n}{T_j(n)}}$$

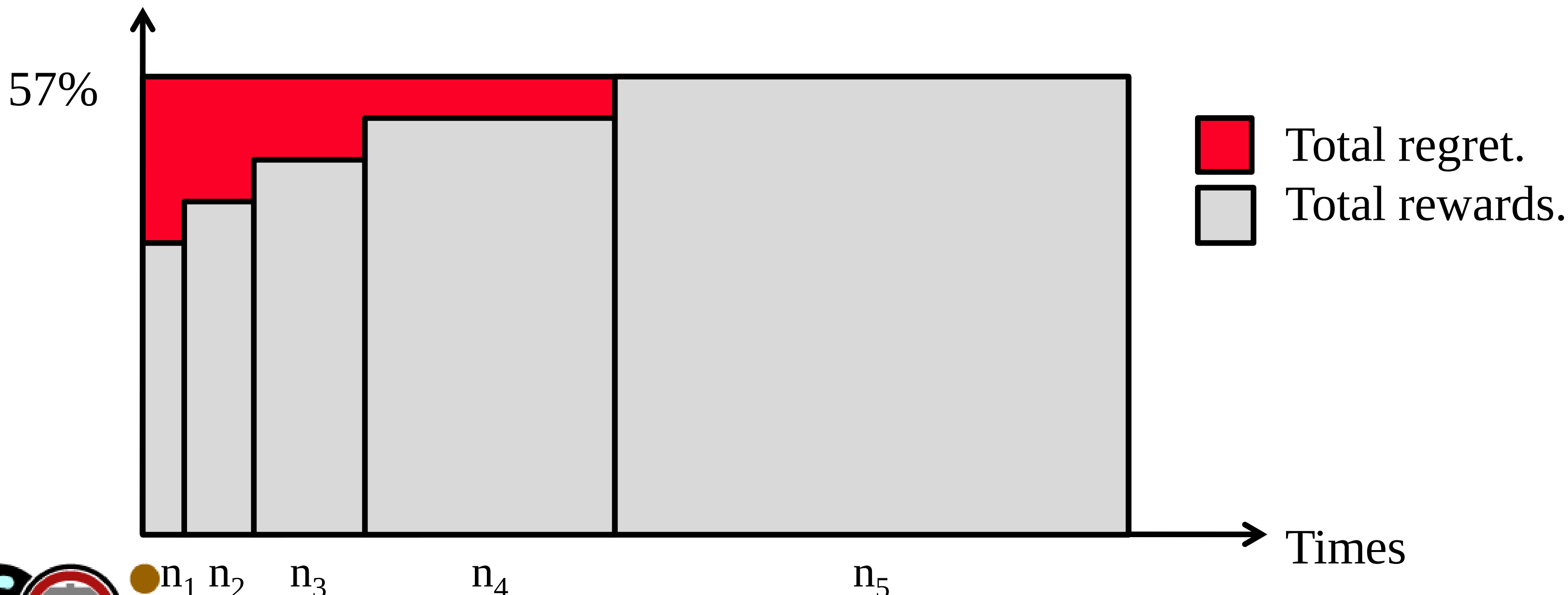
    where n is the overall number of plays done so far,

$$\bar{X}_{i,s} = \frac{1}{s}\sum_{j=1}^{s} X_{i,j} \quad , \quad \bar{X}_i = \bar{X}_{i,T_i(n)} \, ,$$

- Key:
  - Ensure optimal machine is played exponentially more often than any other machine.

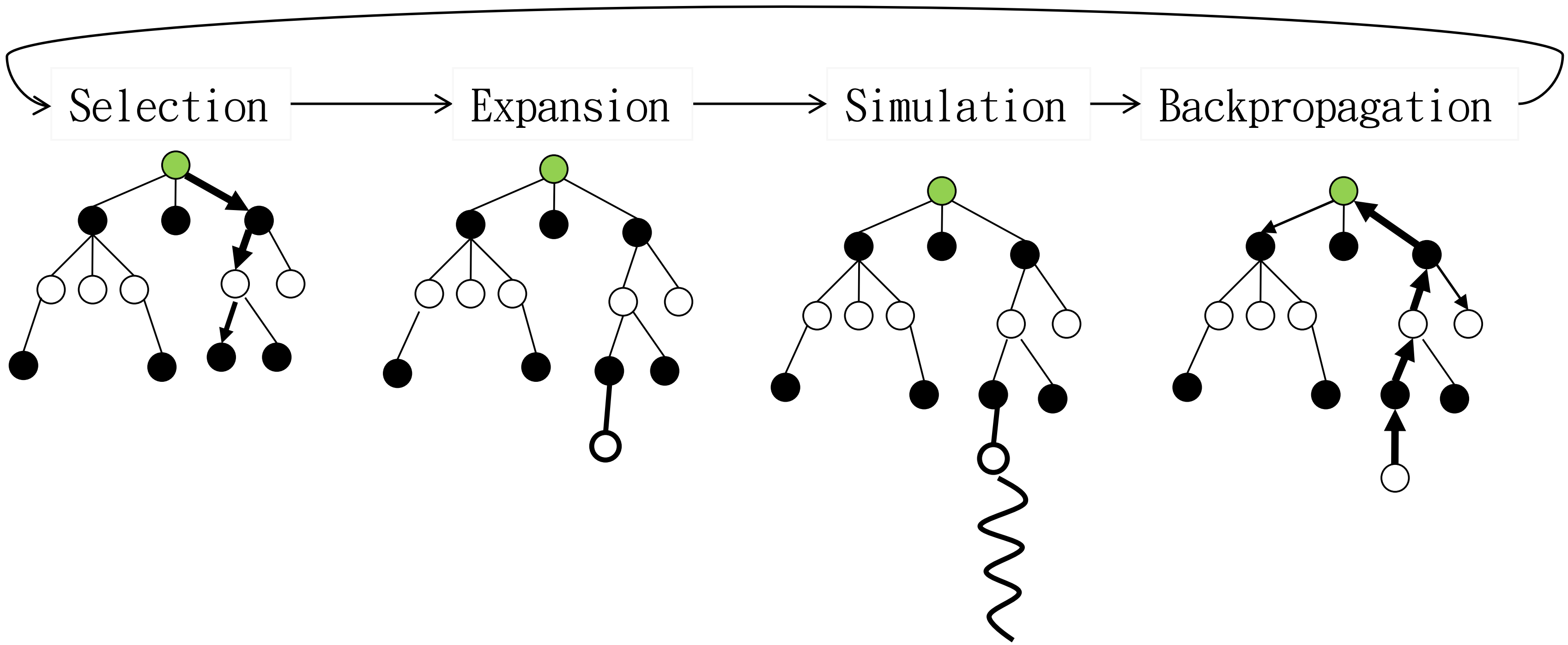
Cumulative Regret

- 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.



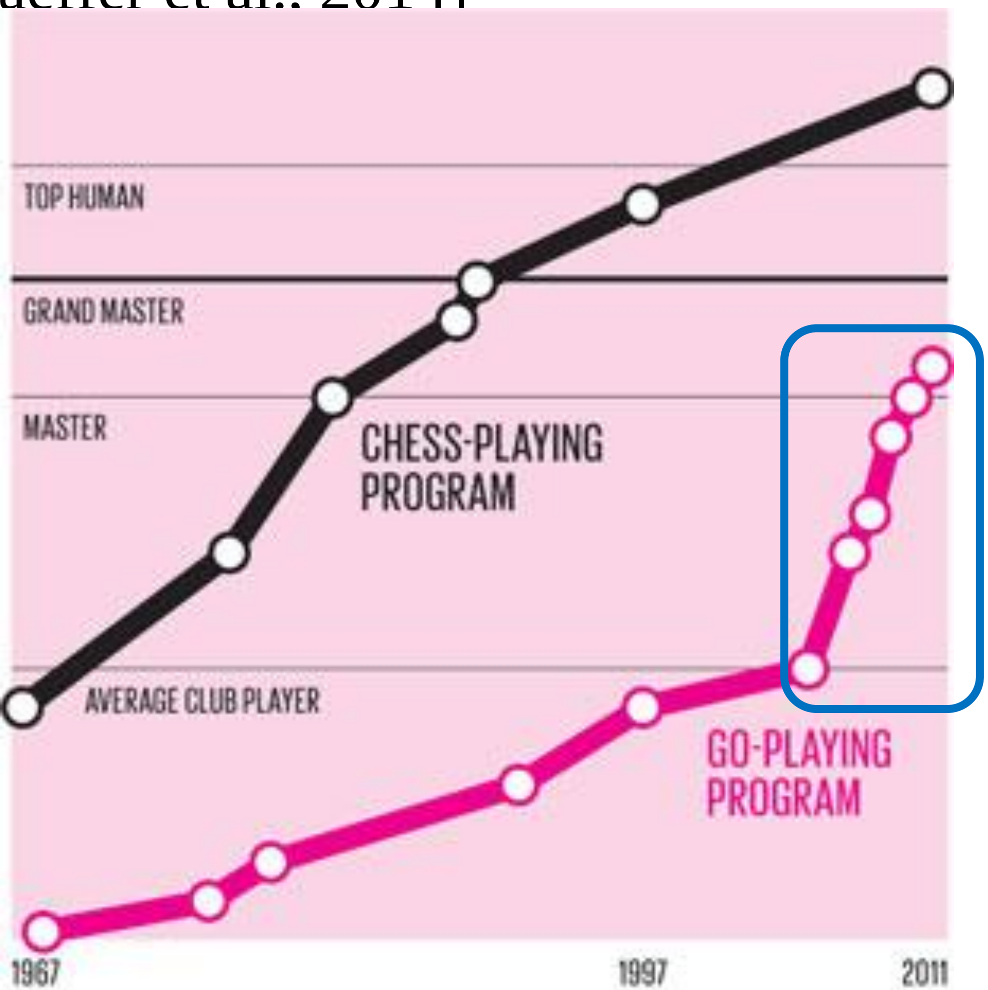
Monte-Carlo Tree Search

- A kind of planning
- A kind of Reinforcement learning



Strength of Go Program after MCTS

- [Schaeffer et al.. 2014]



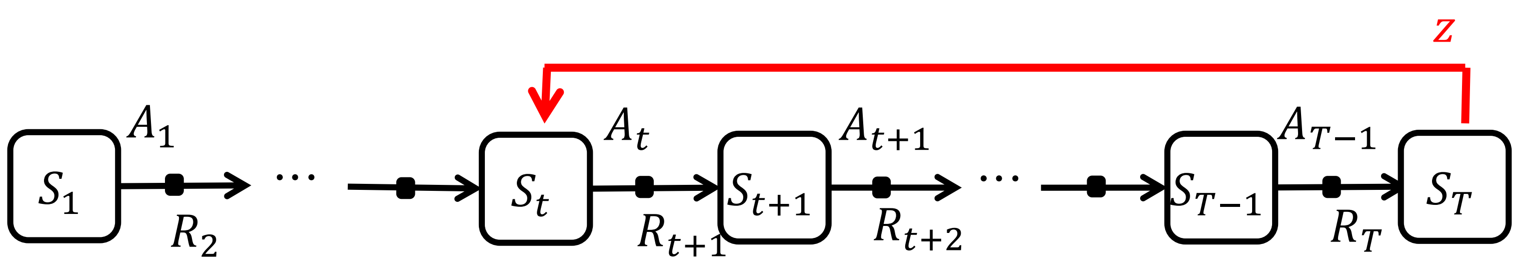
Strength grew fast, after MCTS.

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



Case Study: Pole Balancing

[Lillicrap 2016] Continuous control with deep reinforcement learning, ICLR, 2016 (DeepMind),

- Deep Policy Gradient
  - Actor-critic, model-free algorithm
  - underlying the success of Deep Q-Learning to the continuous action domain

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