

# 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”

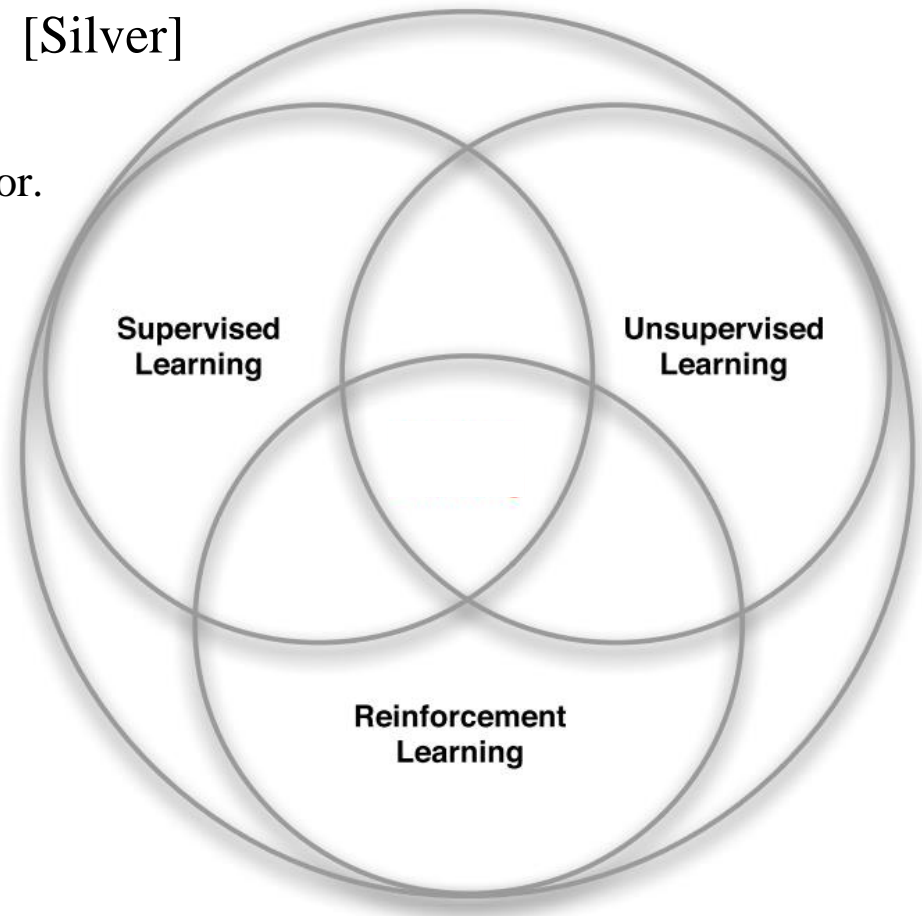
# Many Faces of Reinforcement Learning

- 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)**
  - 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

[Silver]



# 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.

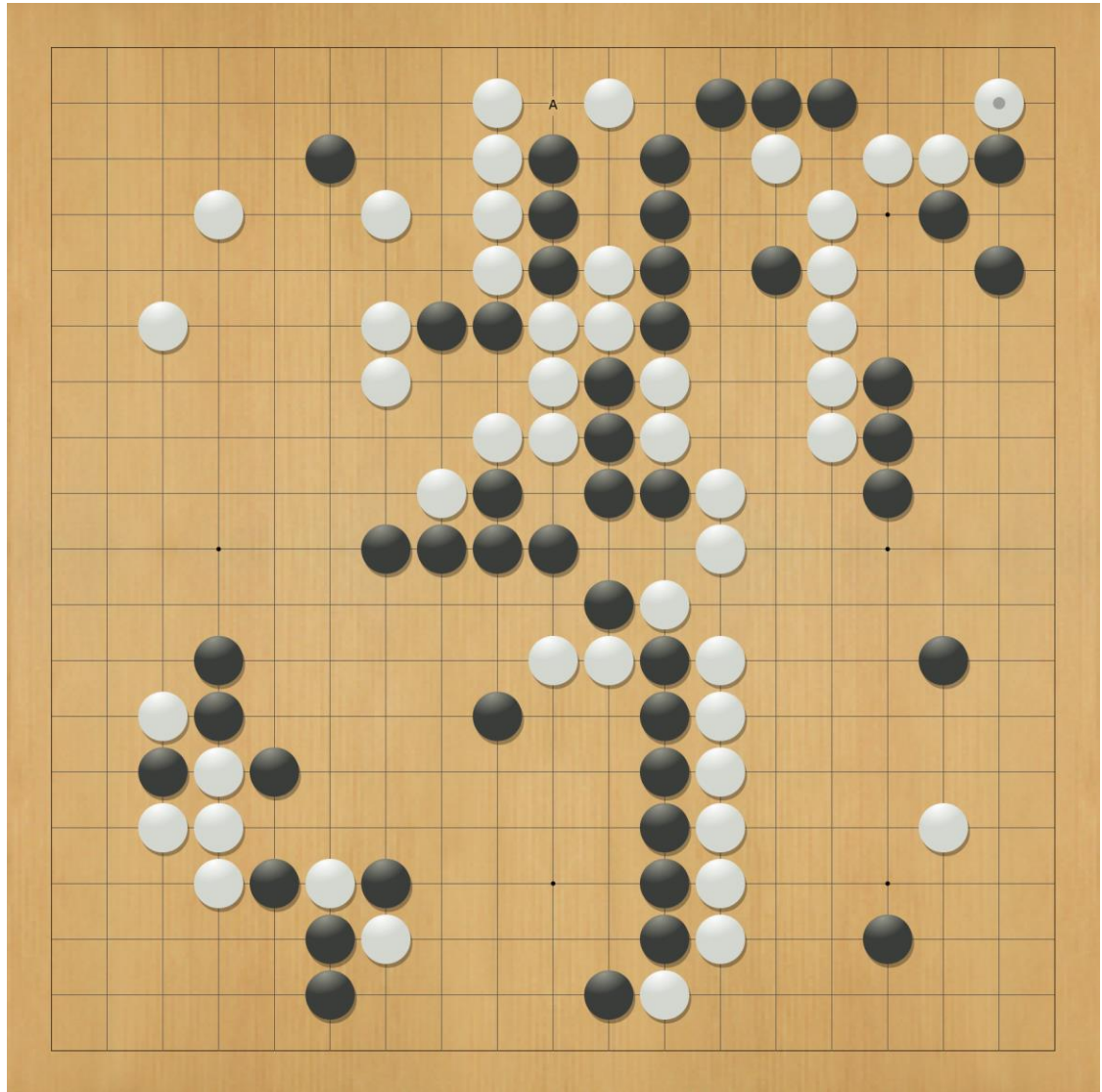


# Successful Examples

- 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.).
- (More recent successful examples for [deep reinforcement learning](#))

# Board Game: Go

- Game 1: AlphaGo  
vs. 李世石



# Stochastic Game: 2048 (lab)

2	32768	8192	4096
16384	1024	512	256
2048	32	64	128
16	16	2	4

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

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

2048

SCORE 1031392

BEST 1031392



I-Chen Wu

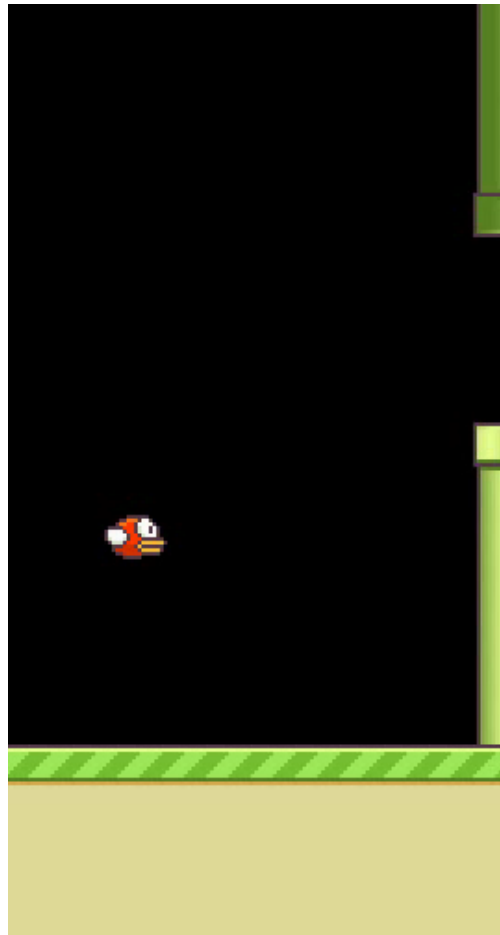
Game over!

Try again

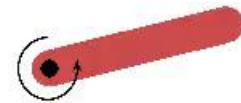
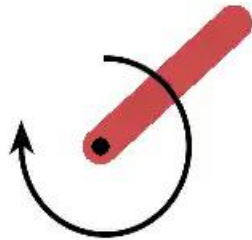
65536



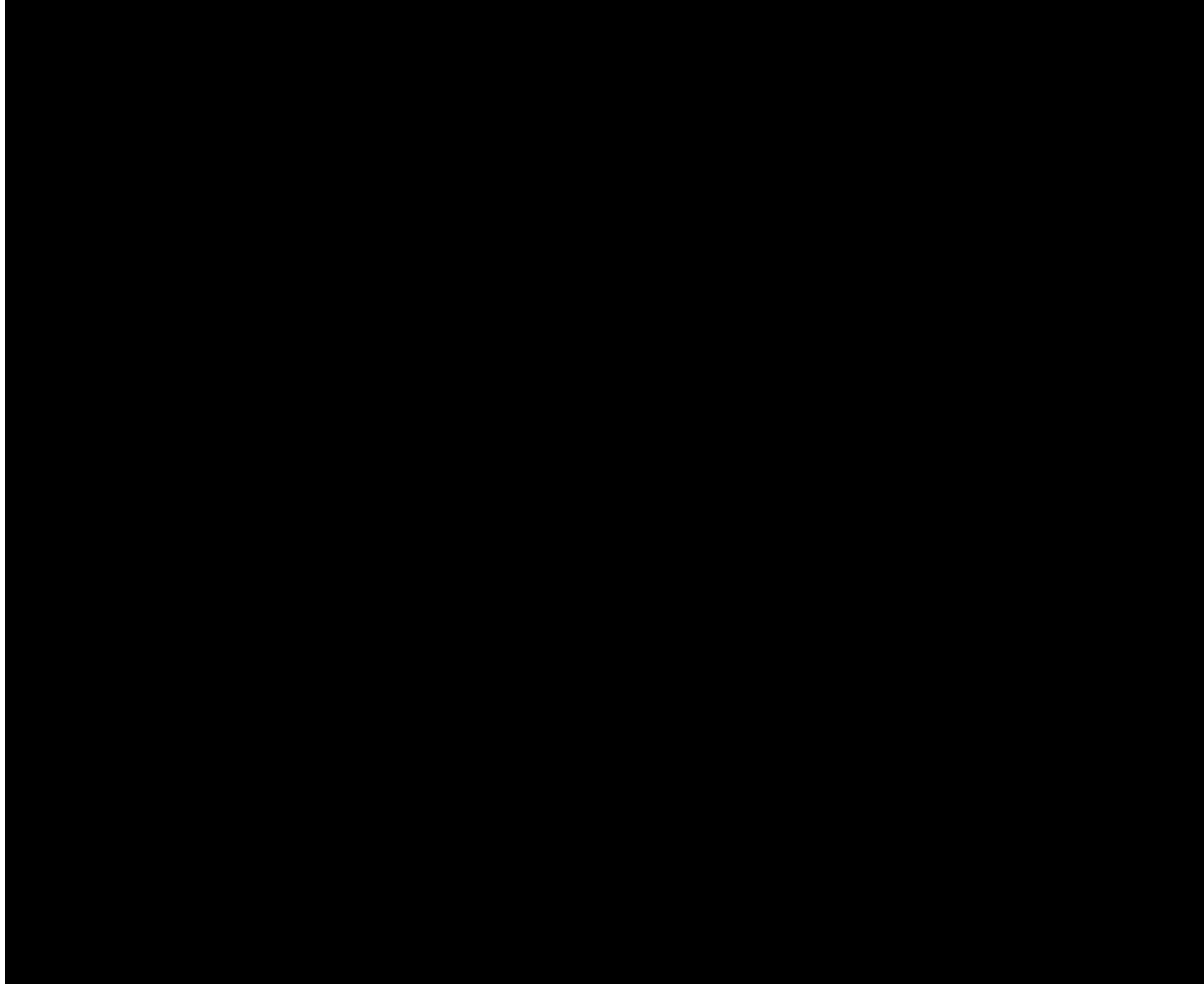
# Video Games: Flappy Bird (lab)



# 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

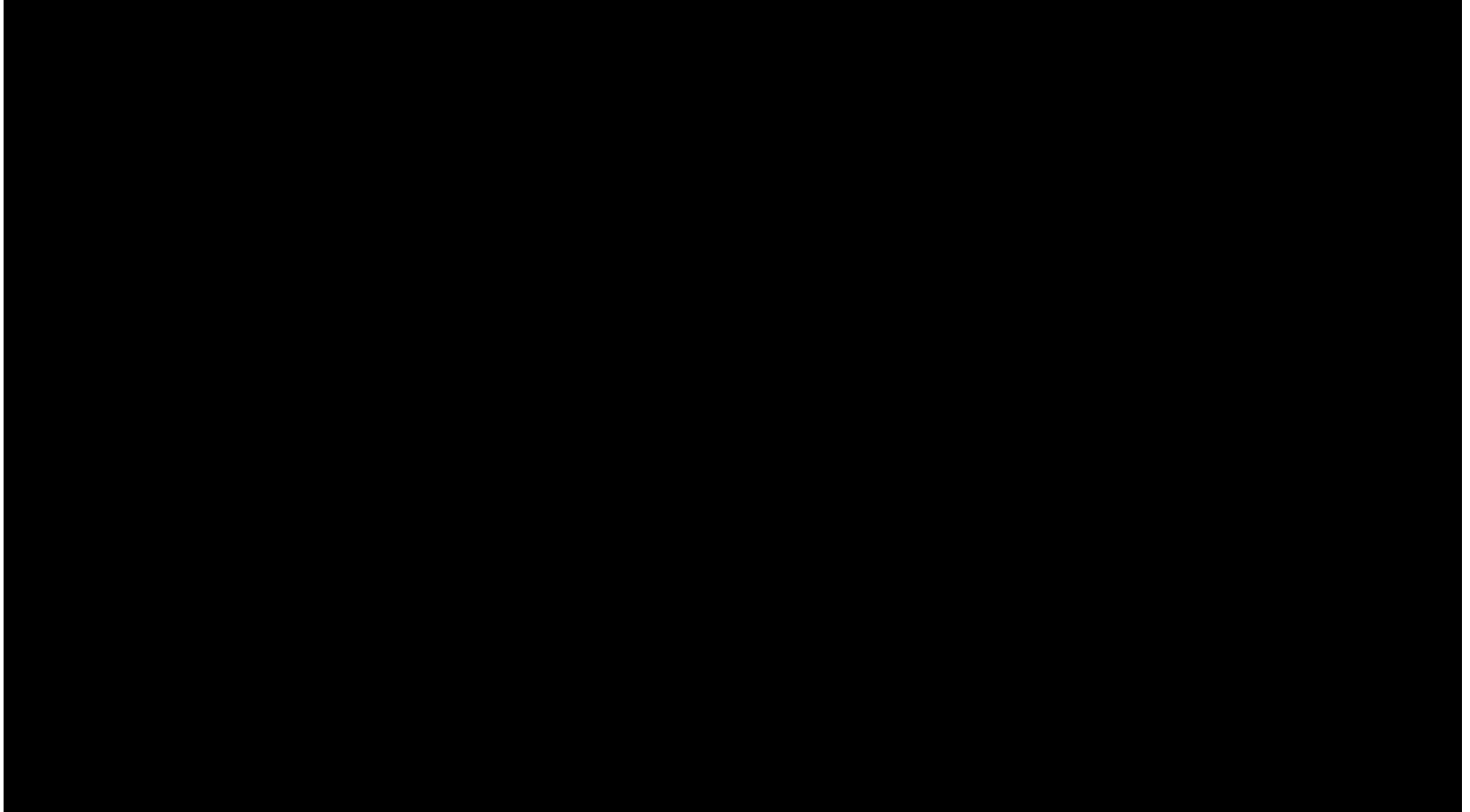
**Marc Peter Deisenroth, Carl Edward Rasmussen, Dieter Fox**

**Learning to Control a Low-Cost Robotic Manipulator  
using Data-Efficient Reinforcement Learning**

**R:SS 2011**



# 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

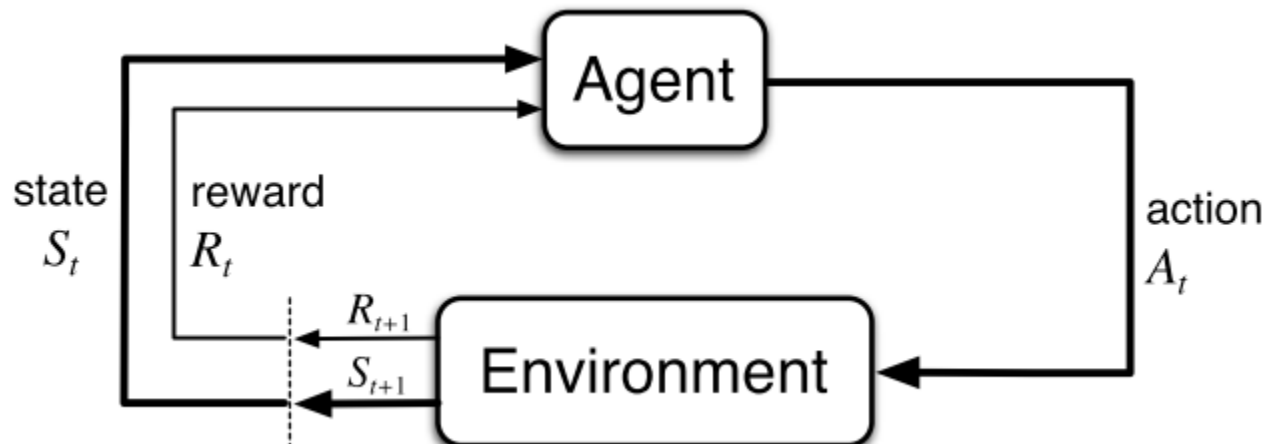
- A **computational approach** to learning from **interaction**
  - Explore designs for machines that are effective in
    - ▶ solving learning problems of scientific or economic interest,
    - ▶ evaluating the designs through mathematical analysis or computational experiments.
  - Focus on **goal-directed 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**



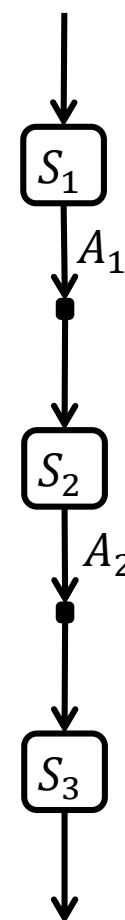
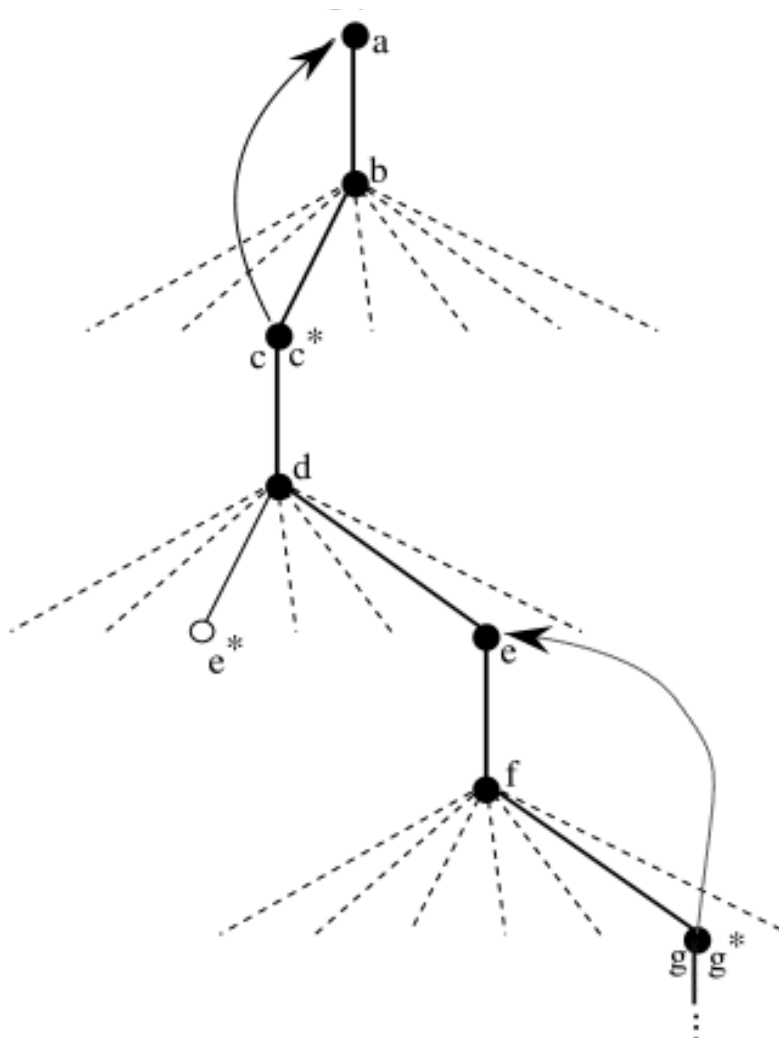
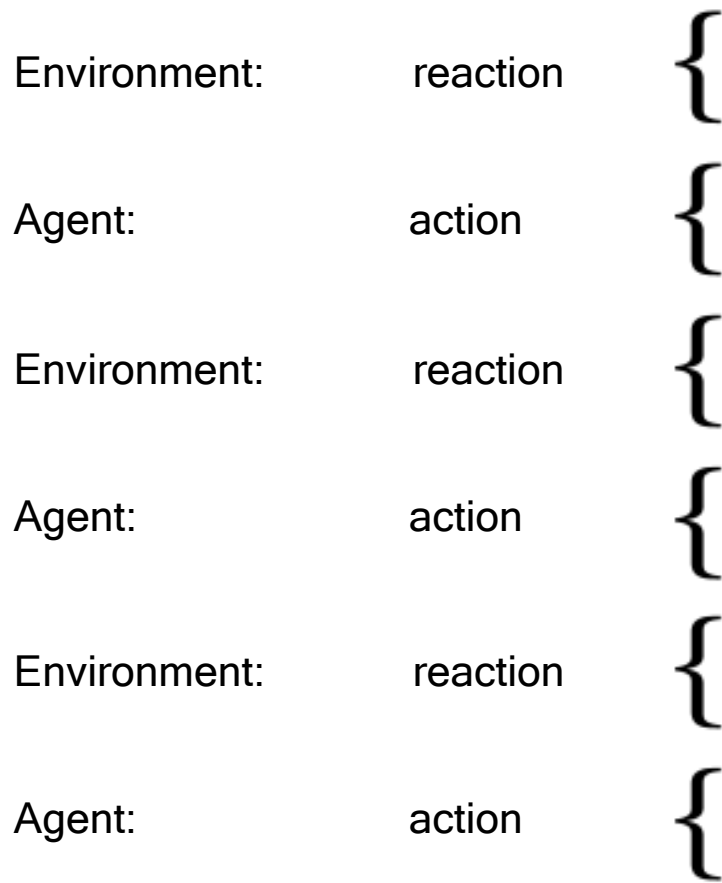


# Agent-Environment Interaction Framework

- **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



## Go

Environment: { opponent's move

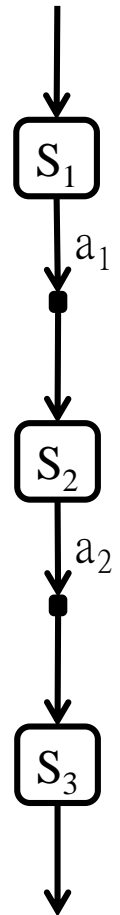
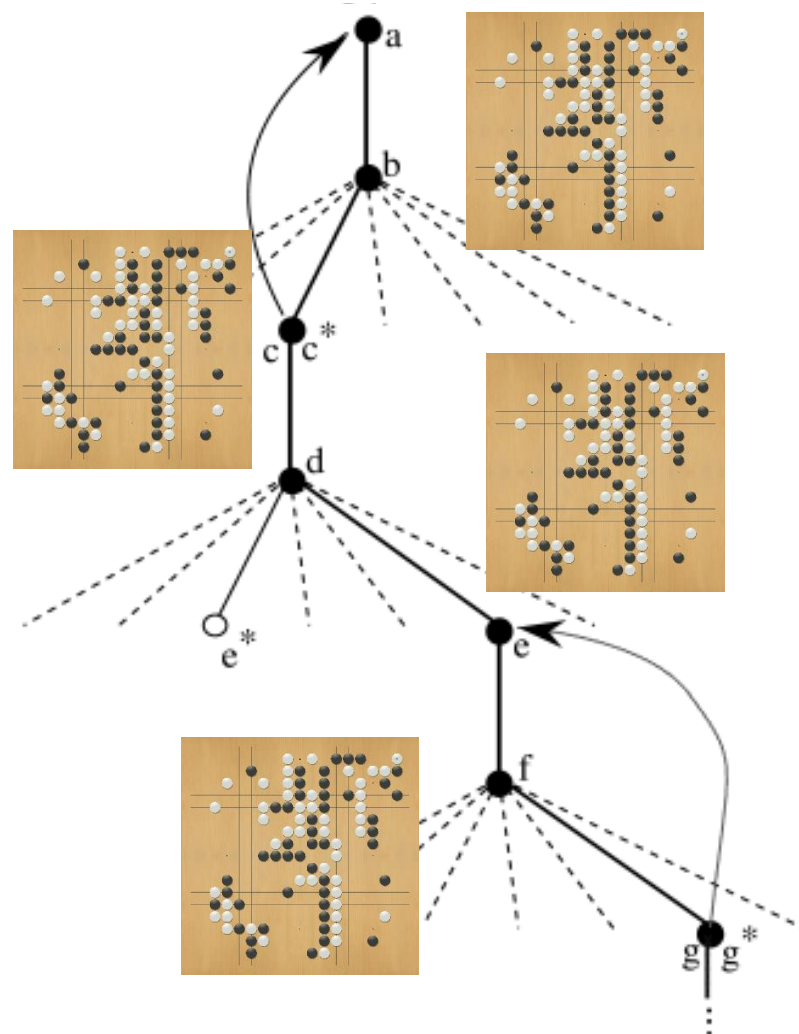
Agent: { our move

Environment: { opponent's move

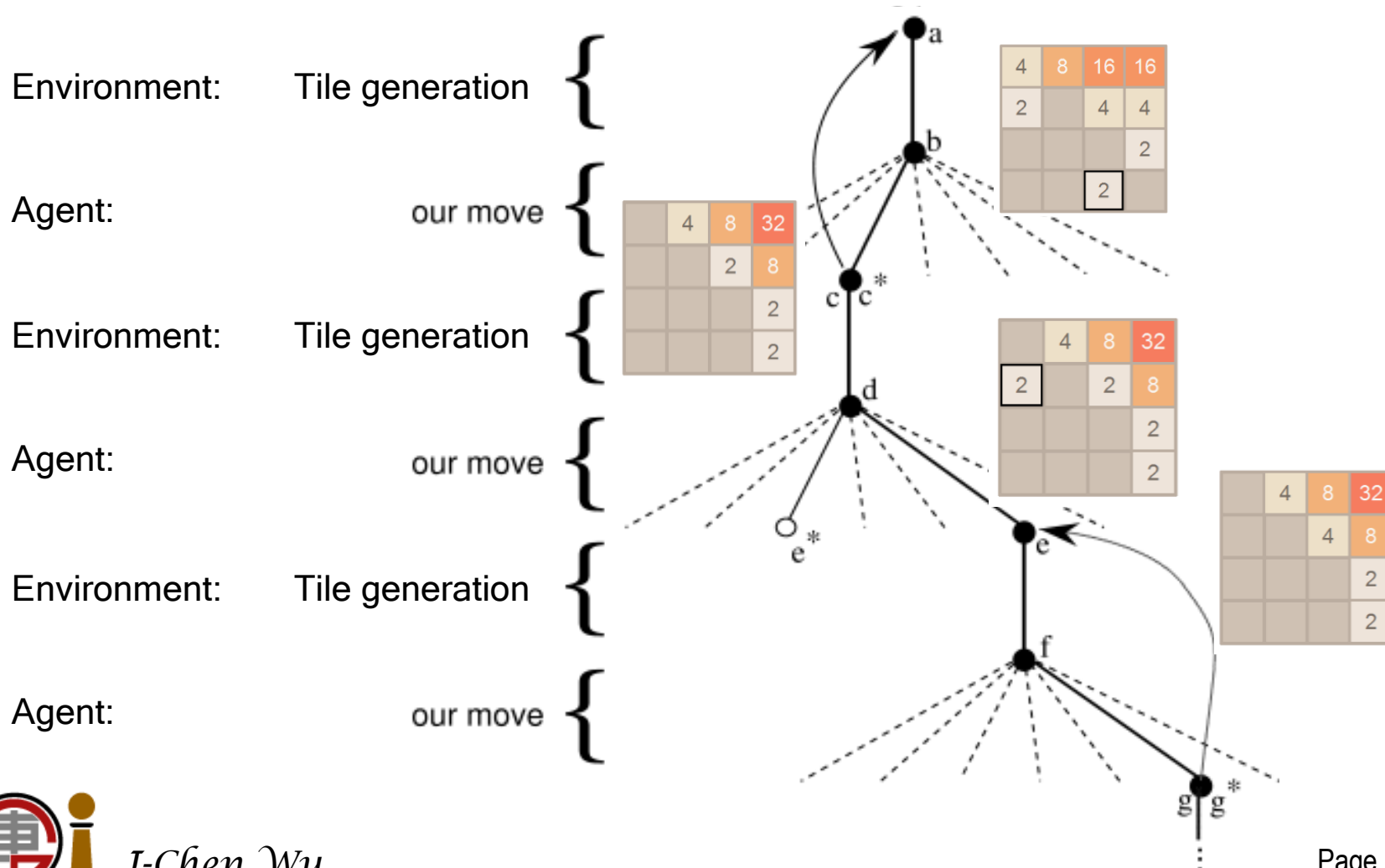
Agent: { our move

Environment: { opponent's move

Agent: { our move



## 2048



# Robot

Environment: Dynamics

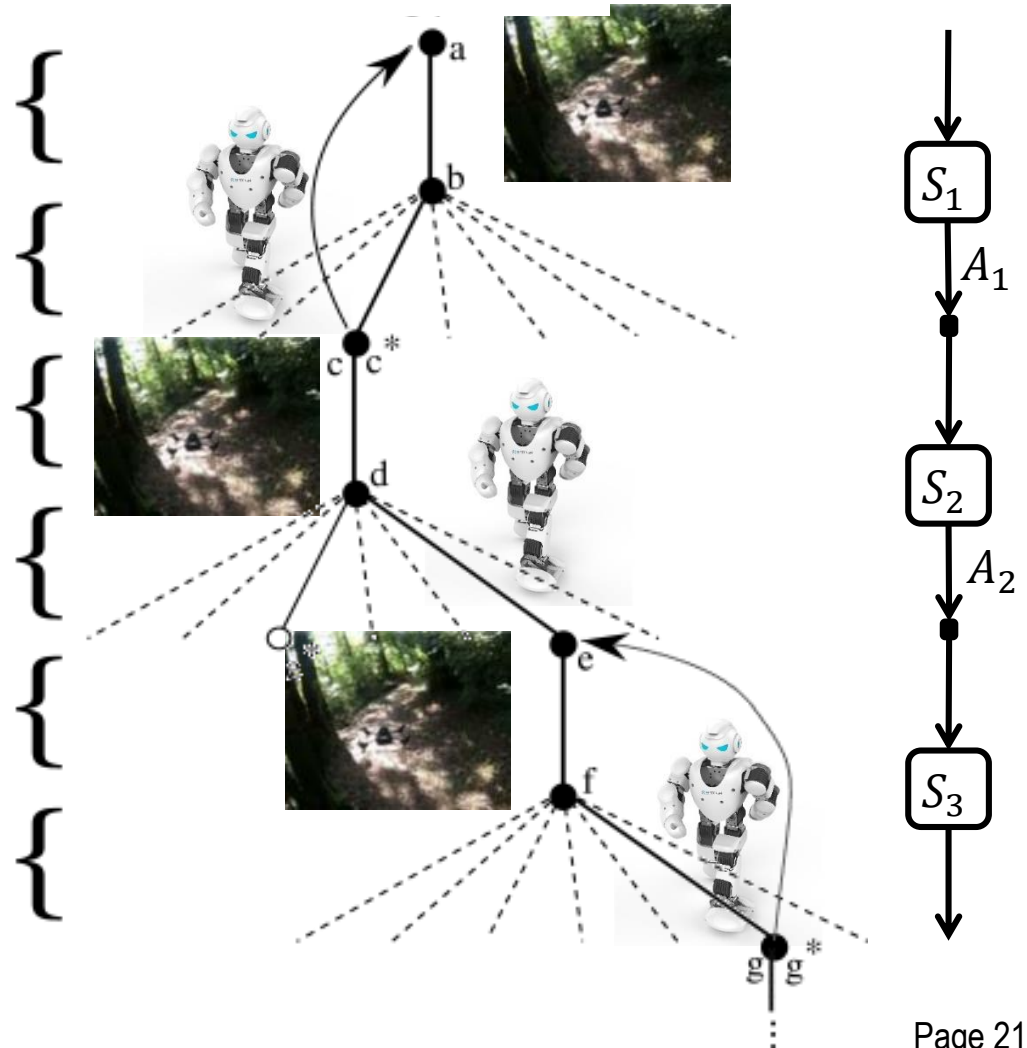
Agent: Navigate

Environment: Dynamics

Agent: Navigate

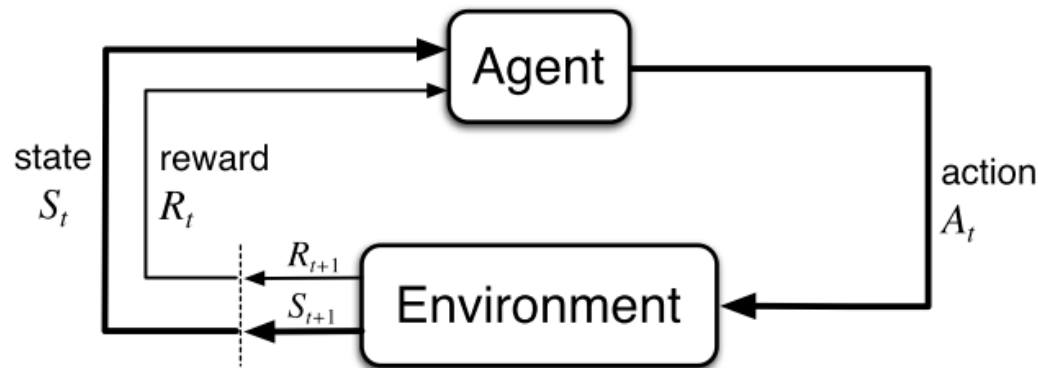
Environment: Dynamics

Agent: Navigate

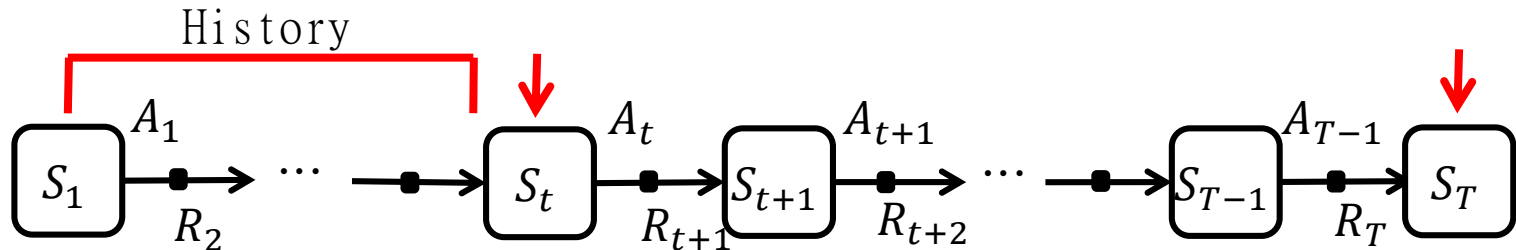


# Markov Decision Processes (MDP)

- A **Markov Decision Process** is a tuple  $\langle \mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma \rangle$ 
  - $\mathcal{S}$  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),  
 $\mathcal{P}_{ss'}^a = \mathbb{P}[S_{t+1} = s' \mid S_t = s, A_t = a]$
  - $\mathcal{R}$  is a reward function,  
 $\mathcal{R}_s^a = \mathbb{E}[R_{t+1} \mid S_t = s, A_t = a]$
  - $\gamma$  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**:  $H_t = (S_1, A_1, R_2, S_2, A_2, R_3, S_3, \dots, R_t)$

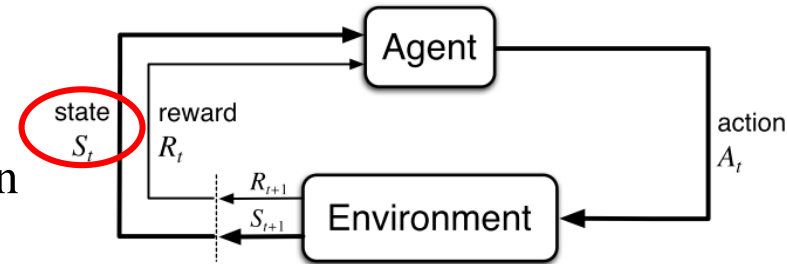
- Markov Property:

- “The future is independent of the past given the present”
- A state  $S_t$  is **Markov** if and only if
$$\mathbb{P}[S_{t+1} | S_t] = \mathbb{P}[S_{t+1} | S_1, \dots, S_t]$$



# 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  $S_t^a \neq S_t^e$



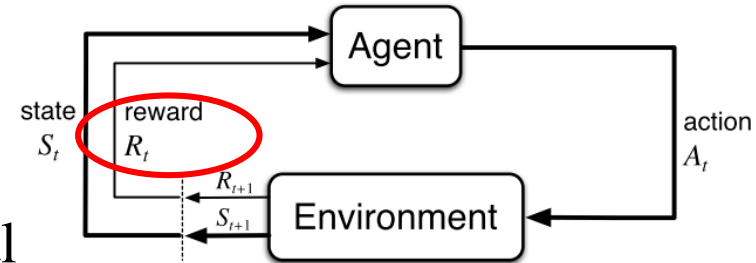


# Example: Mahjong

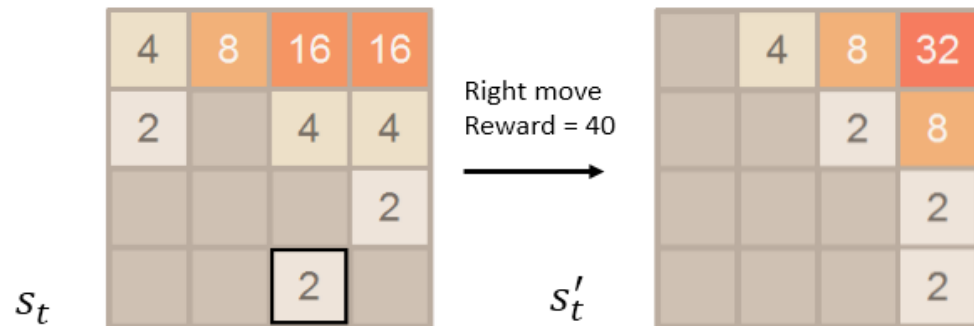
- Partially observable:



# 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

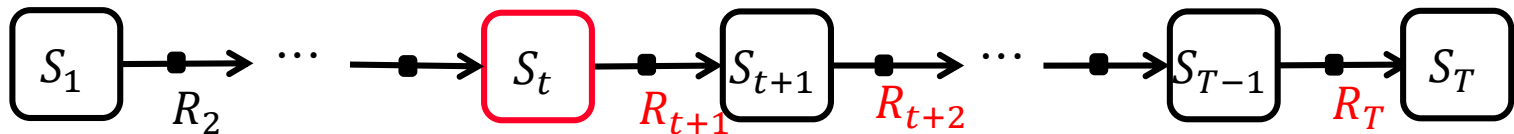
# Rewards for Previous Examples?

- 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  $R_{t+1} + R_{t+2} + \dots + R_T$ 
  - 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  $(2^t, 2^{t-1}, 2^{t-2}, \dots)$
- 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.

2	32768	8192	4096
16384	1024	512	256
2048	32	64	128
16	16	2	4

# Return

## Definition

- The return  $G_t$  is the total discounted reward from time-step  $t$ .

$$G_t = R_{t+1} + \gamma R_{t+2} + \cdots = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}$$

## Notes:

- The discount  $\gamma \in [0, 1]$  is the present value of future rewards
- The value of receiving reward  $R$  is diminishing
  - $\gamma^k R$ , after  $k + 1$  time-steps.
- This values immediate reward above delayed reward.
- Discount:
  - $\gamma$  close to 0 leads to "myopic" evaluation
  - $\gamma$  close to 1 leads to "far-sighted" evaluation
  - Important for infinite episodes.

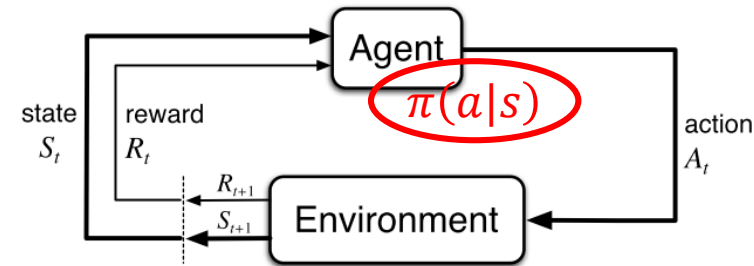


# Major Components of an RL Agent

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

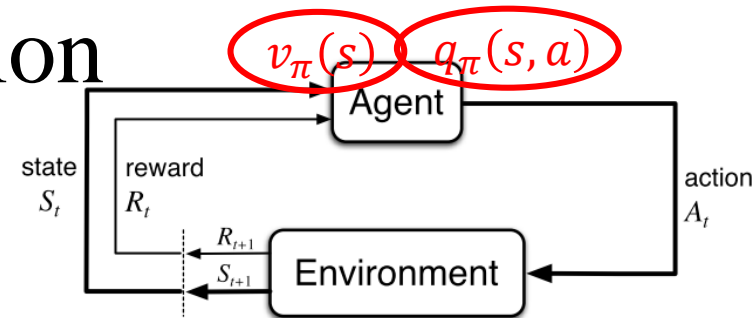
# Policy

- 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

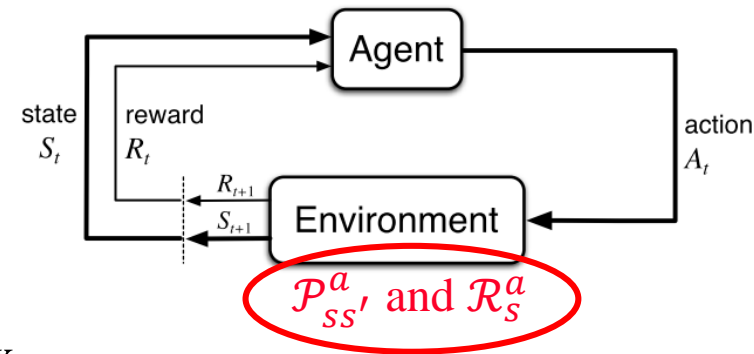


- A value function is **a prediction of future reward**
  - Used to evaluate the goodness/badness of states
    - ▶ therefore to select between actions.
  - Return  $G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots$
- Types of value functions under policy  $\pi$ :
  - **State value function**: the expected return from  $s$ .
 
$$v_\pi(s) = \mathbb{E}_\pi[R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \dots \mid S_t = s]$$

$$= \mathbb{E}_\pi[G_t \mid S_t = s]$$
  - **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,  
$$\mathcal{P}_{ss'}^a = \mathbb{P}[S_{t+1} = s' \mid S_t = s, A_t = a]$$
  - ▶ predicts the next state
- $\mathcal{R}$  is a reward function,  
$$\mathcal{R}_s^a = \mathbb{E}[R_{t+1} \mid S_t = s, A_t = a]$$
  - ▶ 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

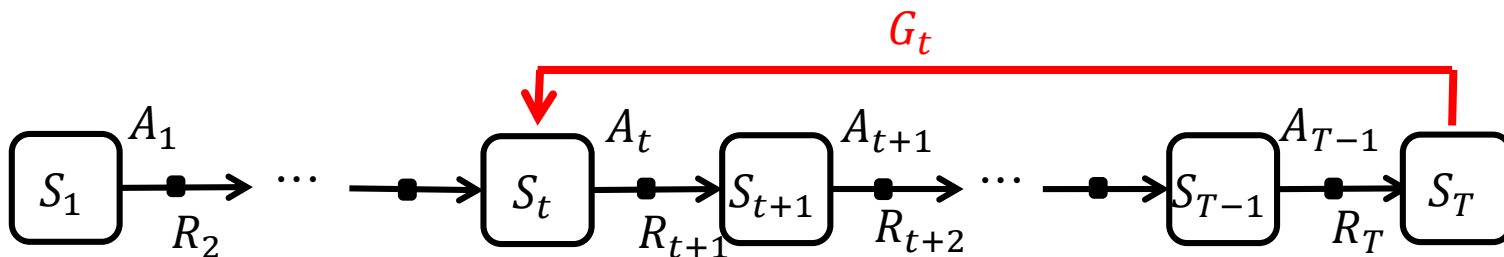
## ● Monte-Carlo (MC) Learning

- 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.



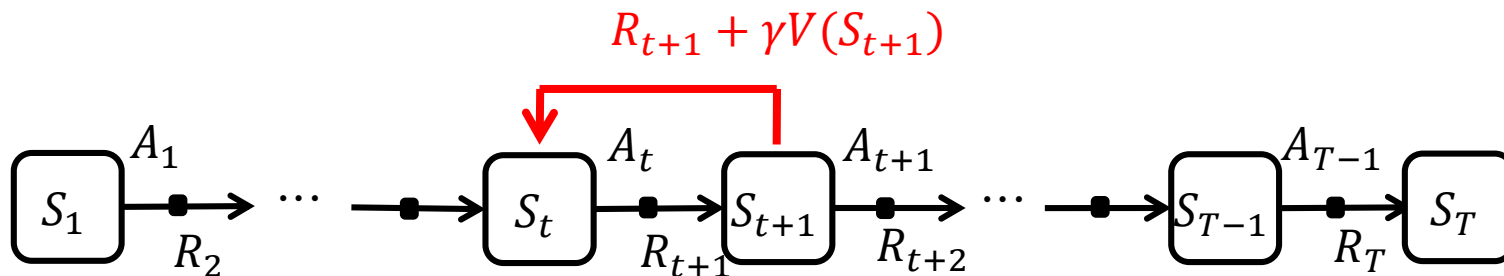
# Monte-Carlo Learning

- Incremental Monte-Carlo
  - Update value  $V(S_t)$  toward actual return  $G_t$   
$$V(S_t) \leftarrow V(S_t) + \alpha(G_t - V(S_t))$$
  - $\alpha$ : 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:  $R_{t+1} + \gamma V(S_{t+1}) - V(S_t)$
  - $\alpha$ : 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).





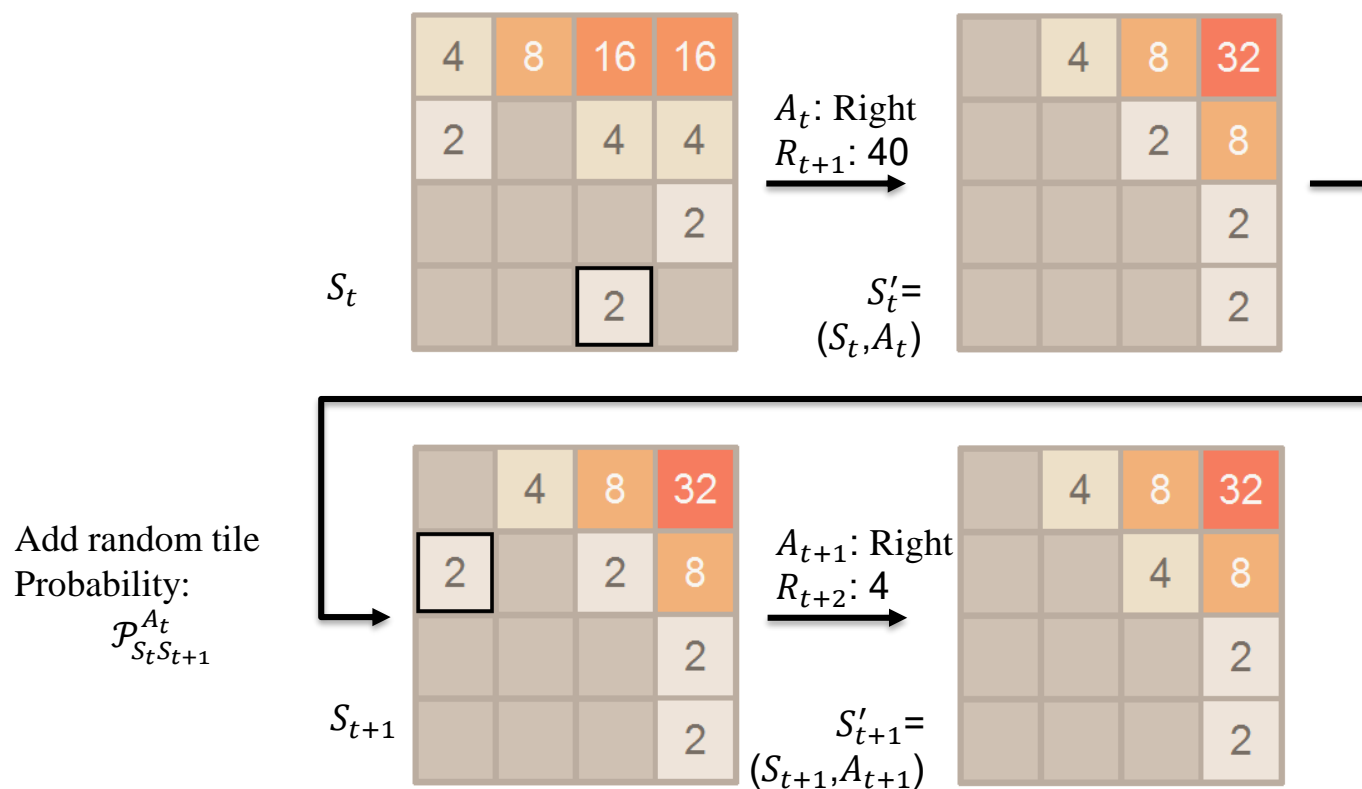
# Cases

- 2048
  - Temporal Difference (TD) Learning
  - N-tuple networks
- Go Programs (with Monte-Carlo Tree Search)
  - Monte-Carlo (MC) Learning
  - Multi-Armed Bandits
  - Planning
- Atari games
  - Temporal Difference (TD) Learning
  - Deep Q-networks (DQN), a kind of Deep NN
- Pole Balancing
  - 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.



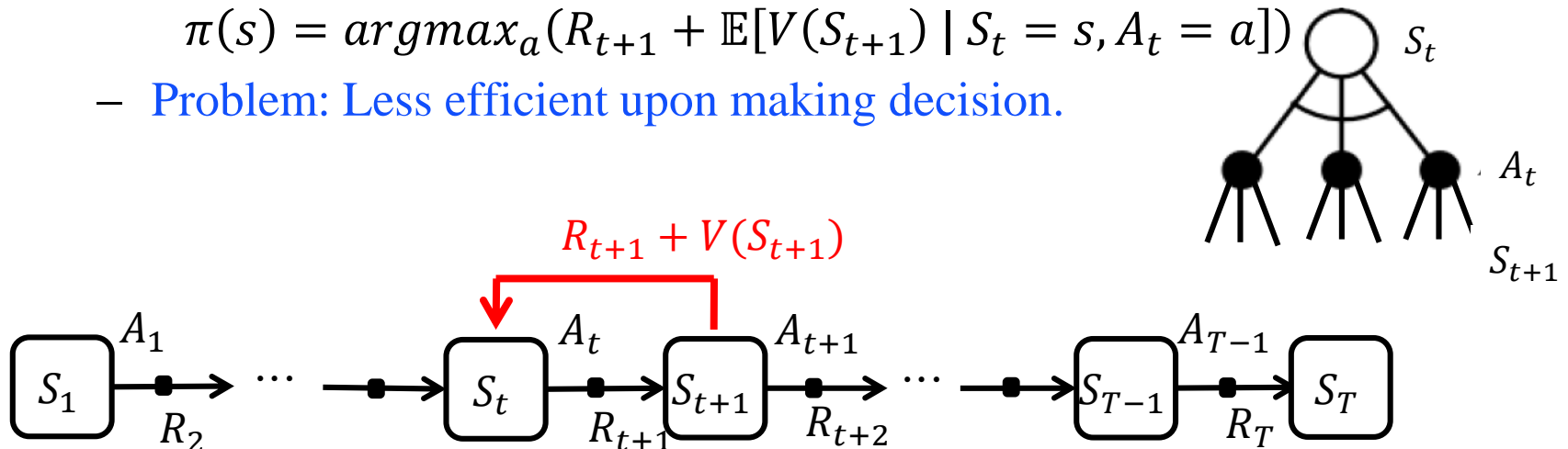
# TD Learning in 2048

- 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).

$$\pi(s) = \operatorname{argmax}_a (R_{t+1} + \mathbb{E}[V(S_{t+1}) \mid S_t = s, A_t = a])$$

- Problem: Less efficient upon making decision.



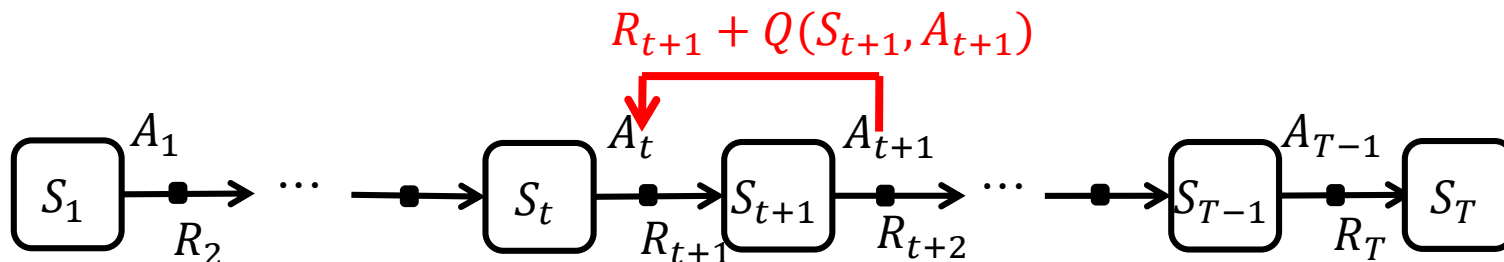
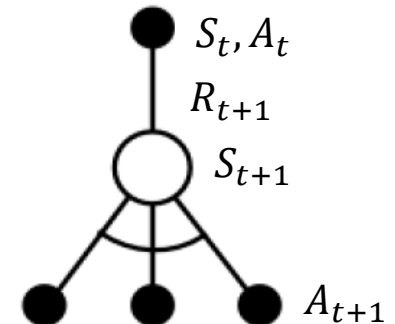
# Q-Learning in 2048

- 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).

$$\pi(s) = \operatorname{argmax}_a (Q(S_t, a))$$

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



# Afterstates in 2048

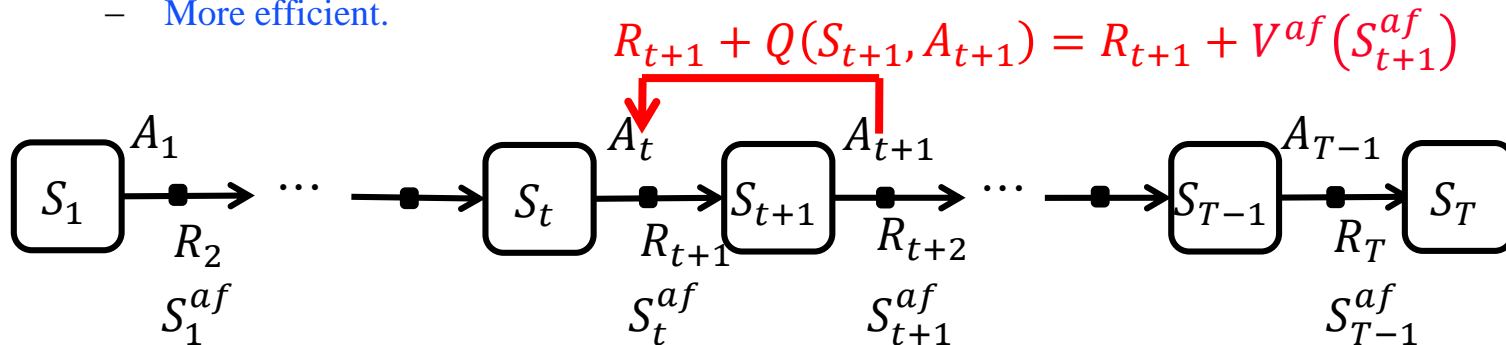
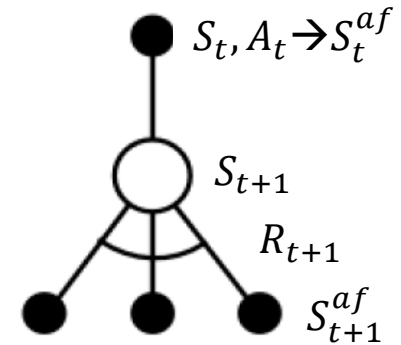
- Afterstate  $S_t^{af}$  is a state after action  $A_t$  at  $S_t$ .
  - Why not use  $S_t^{af}$  instead of  $(S_t, A_t)$ ?
  - Note: in 2048, the reward  $R_{t+1}$  is not included in  $S_t^{af}$ .
- Afterstate value function: (Normally  $\gamma = 1$ )
  - Update value  $V^{af}(S_t^{af})$  toward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

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

- Making decision (based on value).

$$\pi(s) = \operatorname{argmax}_a (V^{af}(S_t^{af}))$$

- 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**:  $\pi(S) = \operatorname{argmax}_a(\hat{v}(S, \theta))$
- 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

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

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

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









# Gradient Descent

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



# N-Tuple Network

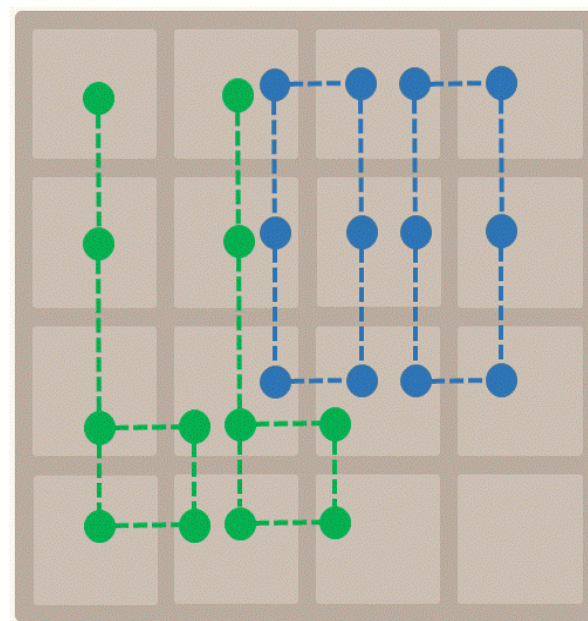
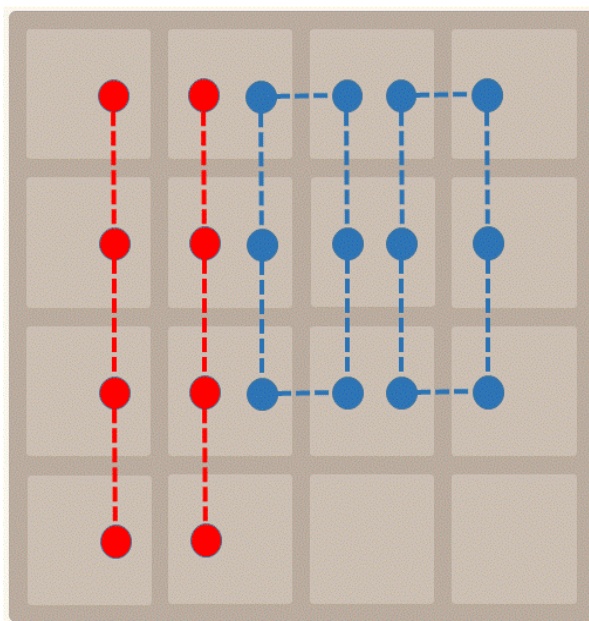
- Characteristics:
  - Provide with a large number of features.
  - Easily update.
- Example: 4-tuple networks as shown.
  - Each cell has 16 different tiles
  - $16^4$  features for this network.
    - ▶ But only one is on, others are 0.
    - ▶ So, we can use table lookup to find the feature weight.

64		8	4
128			2
2			2
128			

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.



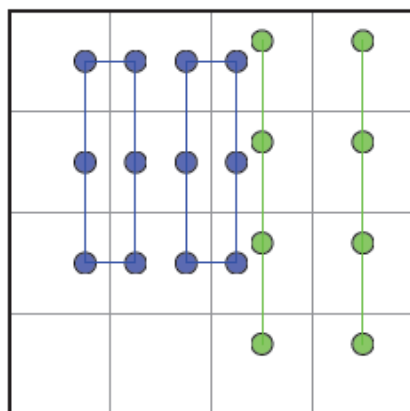
# Update Features in N-Tuple Networks

- For each n-tuple networks, simply update one weights.
- Features:
  - 8 x  $16^4$  features,  $x(S) = [0, 1, 0, \dots, 0, 0, 1, \dots, \dots, 1, 0, 0, \dots]$ 
    - ▶ 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,
 
$$\hat{v}(S, \theta) = x(S)^T \theta = \sum_{j=1}^n x_j(S) \theta_j = \sum_{i=1}^k LUT_i[index(s_i)]$$
  - $LUT_i$ : 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  $\theta_j$  with  $\alpha\Delta V$  at  $LUT_i[index(s_i)]$ .

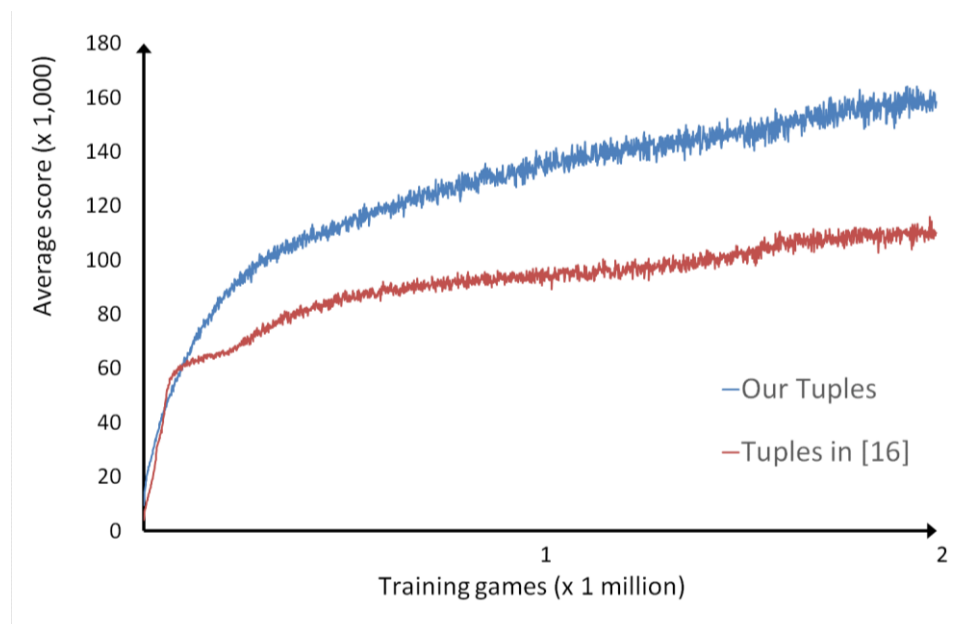
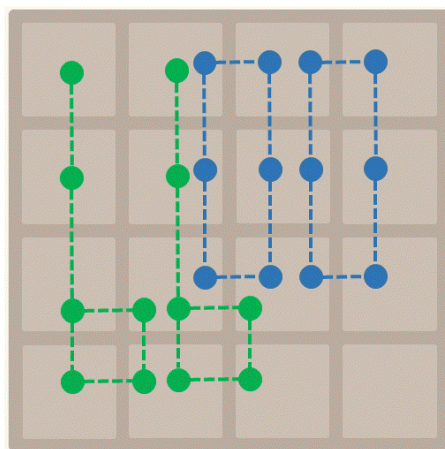


# The N-Tuple Networks Used

- Use the following [Szubert and Jaskowski 2014]



- Ours:



## Our Results (2015)

	<b>CGI-2048 (2<sup>nd</sup> in contest) (100 games)</b>	<b>Kcwu (1<sup>st</sup> in contest) (100 games)</b>	<b>Xificurk's Program (246 games)</b>	<b>Current CGI-2048 (1000 games)</b>
2048	100.0%	100.0%	<b>100.0%</b>	<b>100.0%</b>
4096	100.0%	100.0%	<b>100.0%</b>	<b>100.0%</b>
8192	94%	96%	<b>99.1%</b>	<b>99.5%</b>
16384	59%	67%	<b>92.7%</b>	<b>93.6%</b>
32768	0%	2%	<b>31.7%</b>	<b>33.5%</b>
Max score	367956	625260	<b>829300</b>	<b>833300</b>
Avg score	251794	277965	<b>442419</b>	<b>446116</b>
Speed	500 moves/sec	>100 moves/sec	<b>2-3 moves/sec</b>	<b>500 moves/sec</b>



# The First 65536

2	32768	8192	4096
16384	1024	512	256
2048	32	64	128
16	16	2	4

2		8192	
	32768	4096	4096
	8	16384	8
4	8	4	2

2	4	2	2
8	32768	8	
8	32768	16	4
2	16	4	2

## 2048

SCORE  
1031392

BEST  
1031392

512	256	32	2
1024	128	16	4
4096	64	8	2
65536	4	2	4

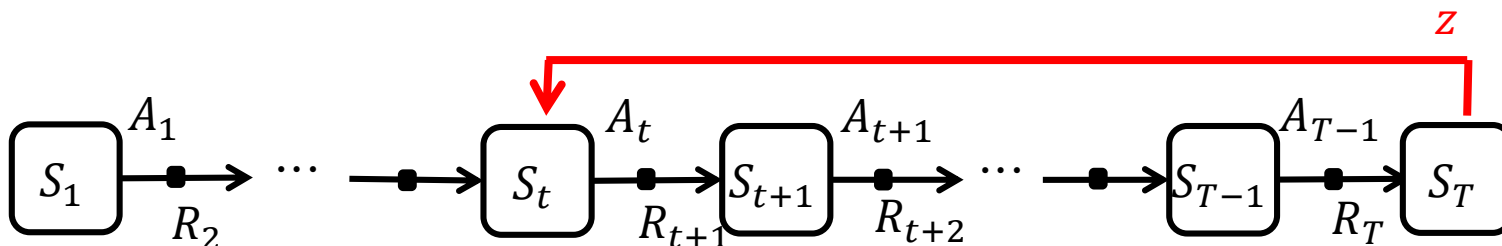
Game over!

Try again



# Case Study: Go (MCTS)

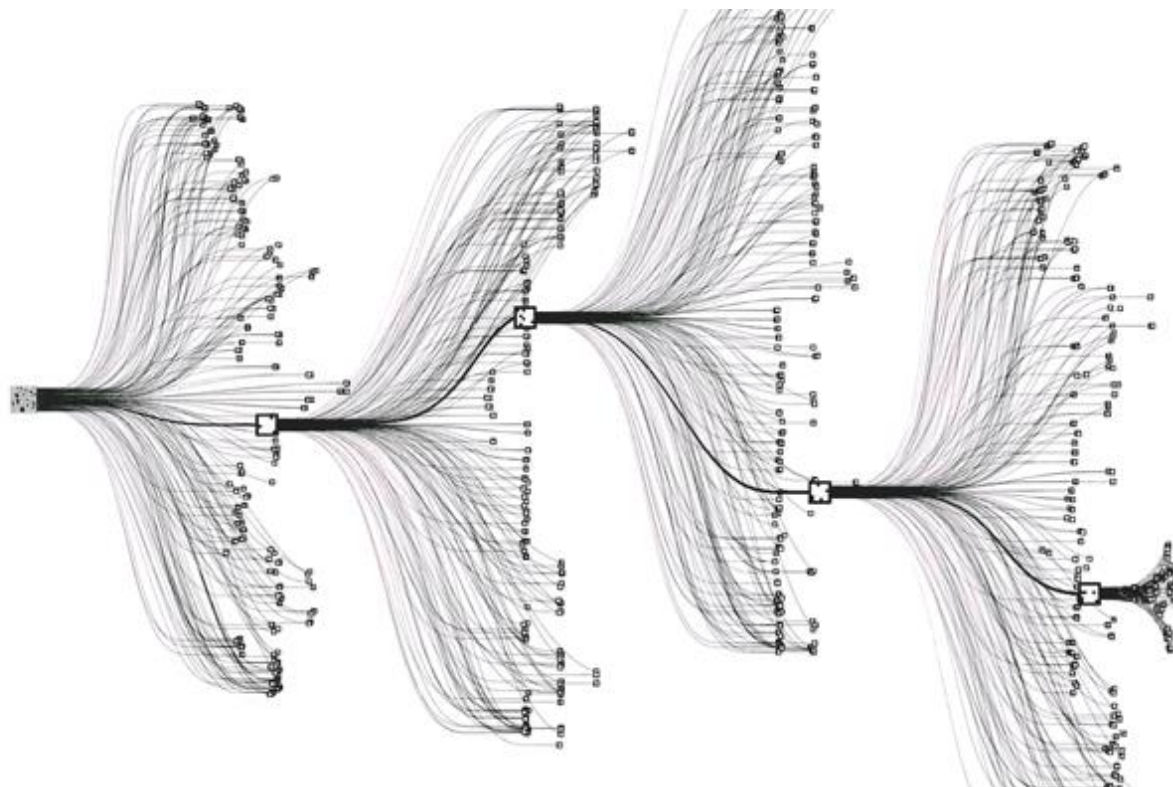
- Monte-Carlo Tree Search:
  - Monte-Carlo (MC) Learning (**z: 1 for win, 0 for loss**)
  - Multi-Armed Bandits
  - **Planning**
- Very successful for Go in the past decade.
- And also applied to others successfully.
  - Other games like Havannah, Hex, GGP
  - Other applications, like mathematical optimization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





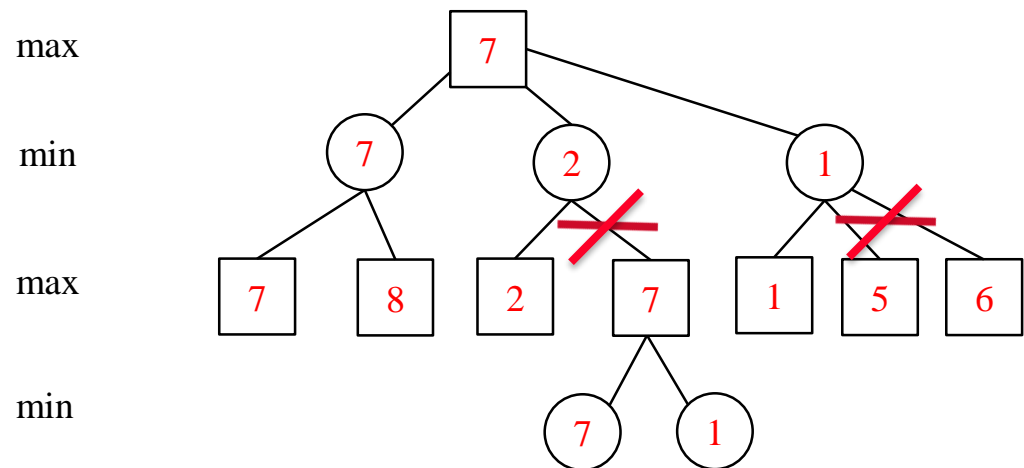
# Go – One of the Most Popular Games

- 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.**

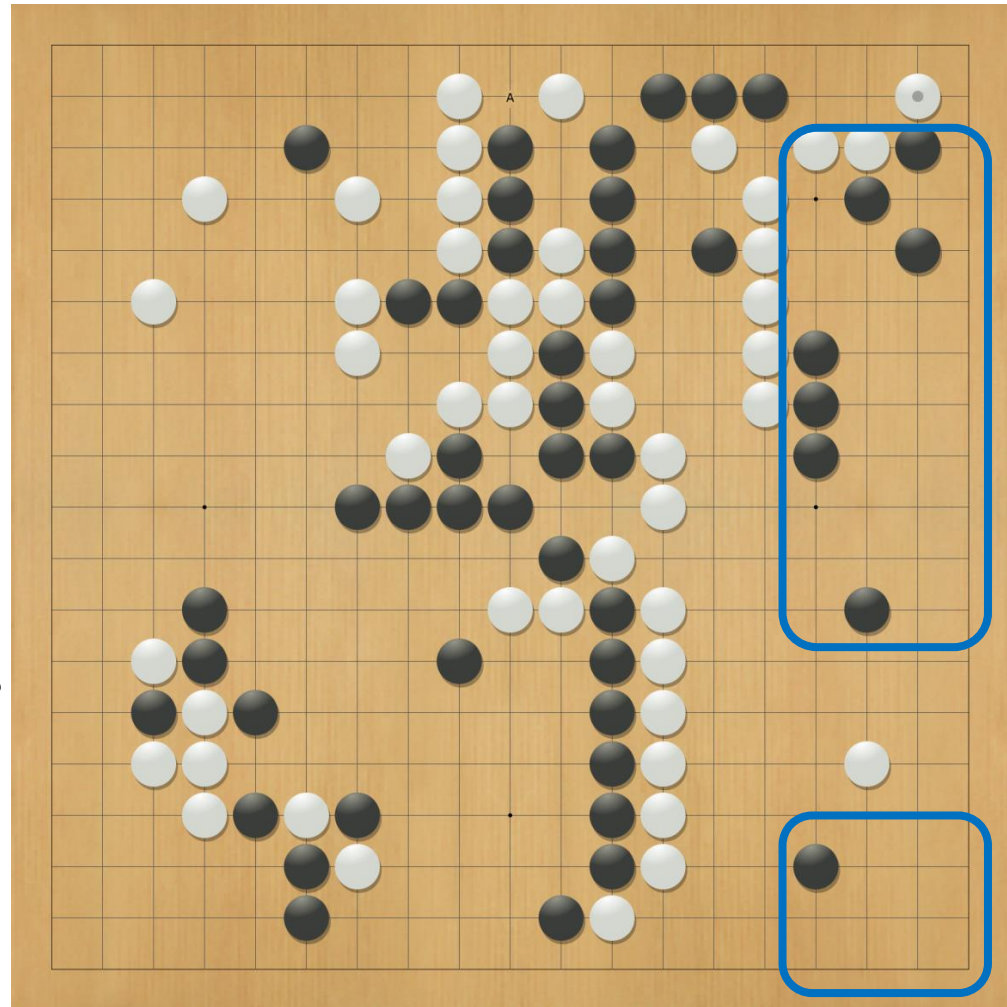


# Why not alpha-beta search for Go?

- 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.

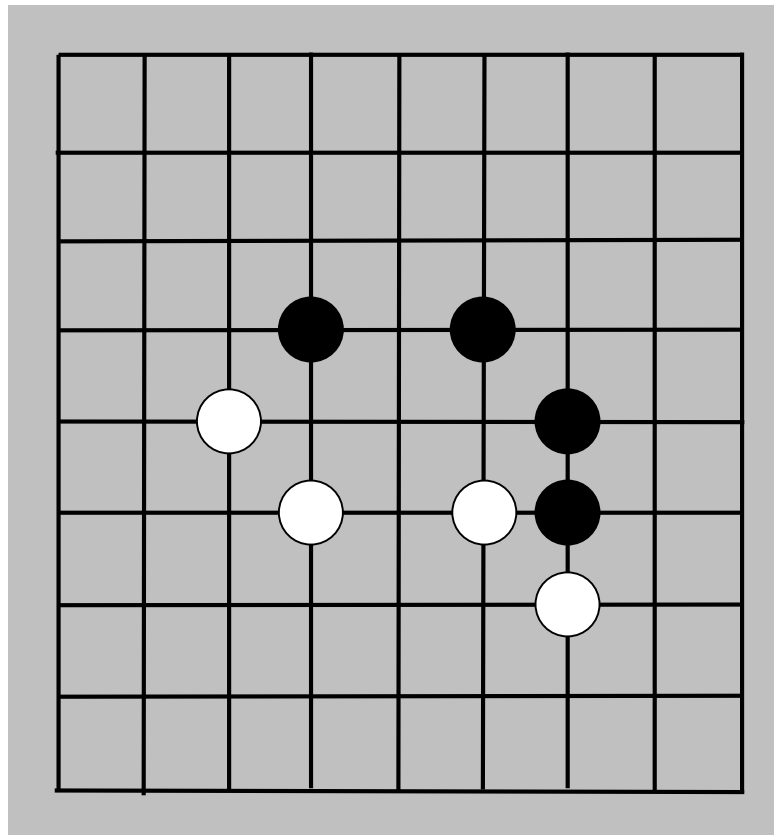
Since no simply heuristics to evaluate,

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



# Rules Overview Through a Game (opening 1)

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



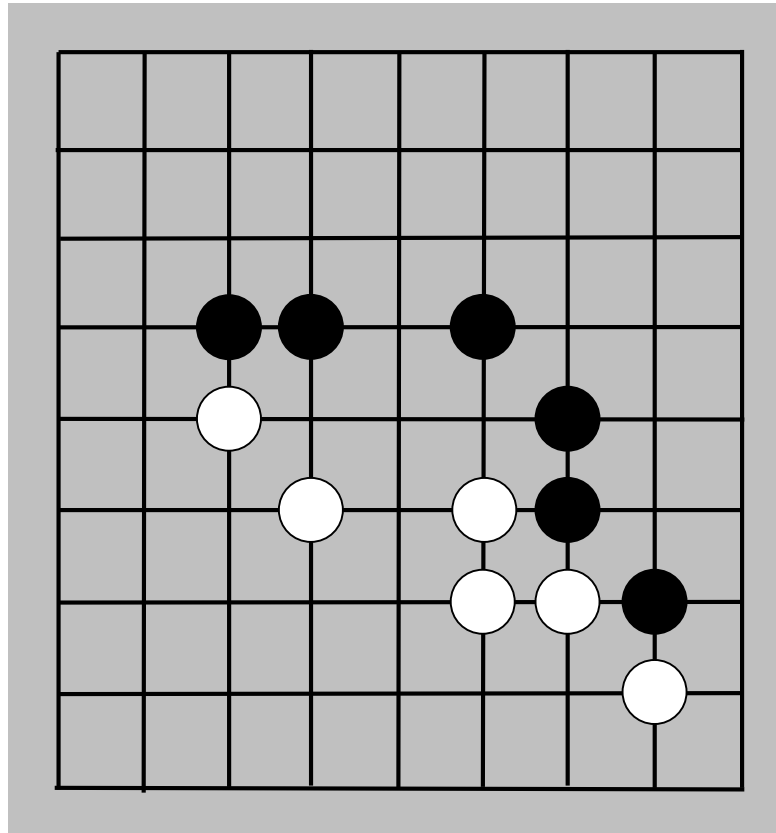
The example was given by B. Bouzy at CIG'07.



# Rules Overview Through a Game

## (opening 2)

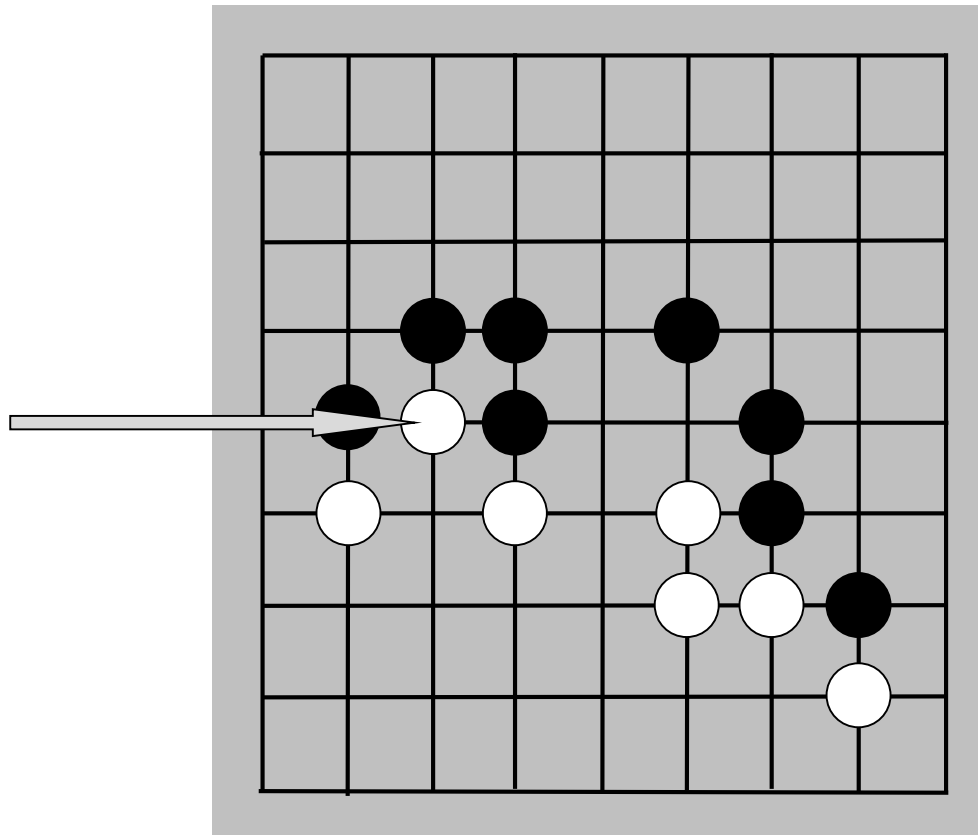
- Black and White aims at surrounding large « zones »



# Rules Overview Through a Game

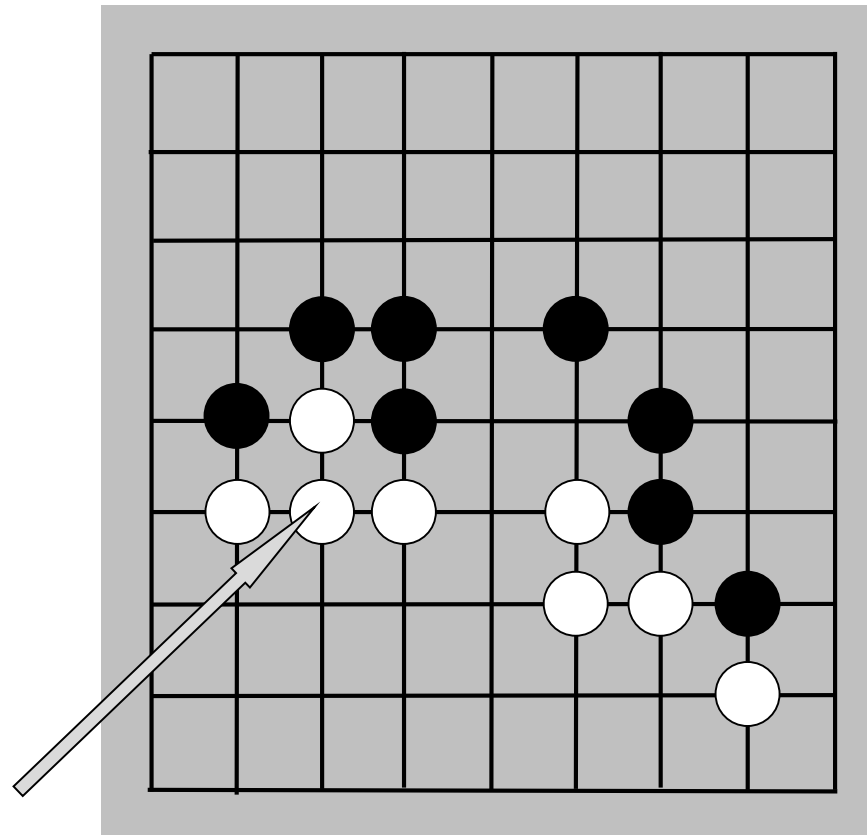
(atari 1)

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



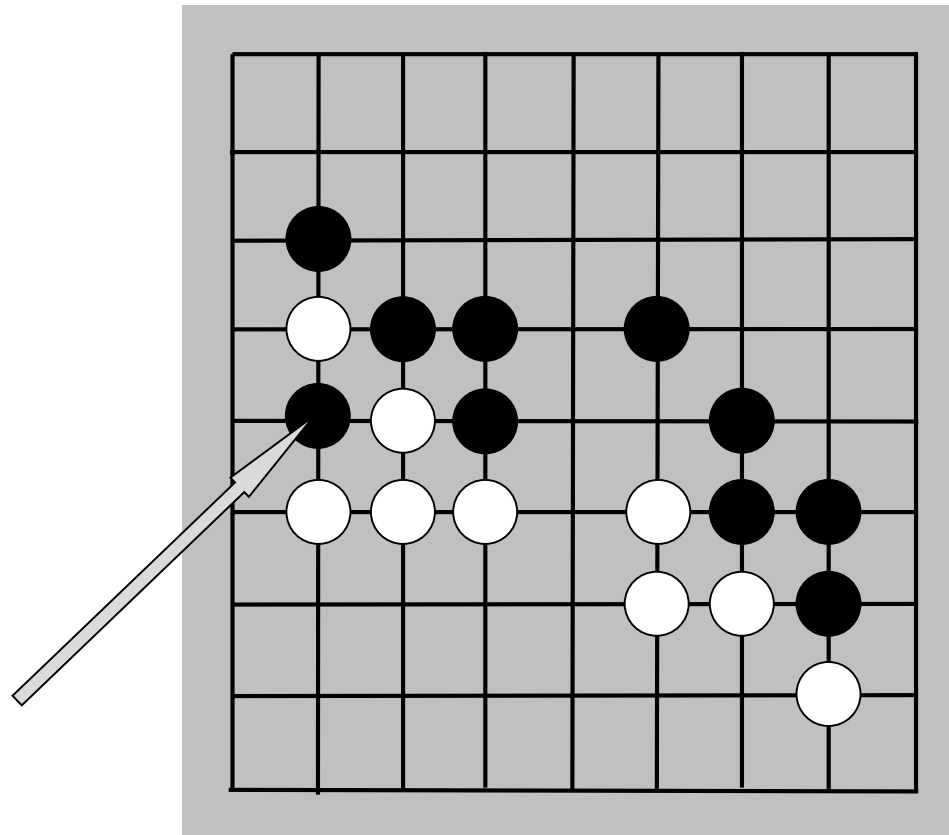
# Rules Overview Through a Game (defense)

- 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.

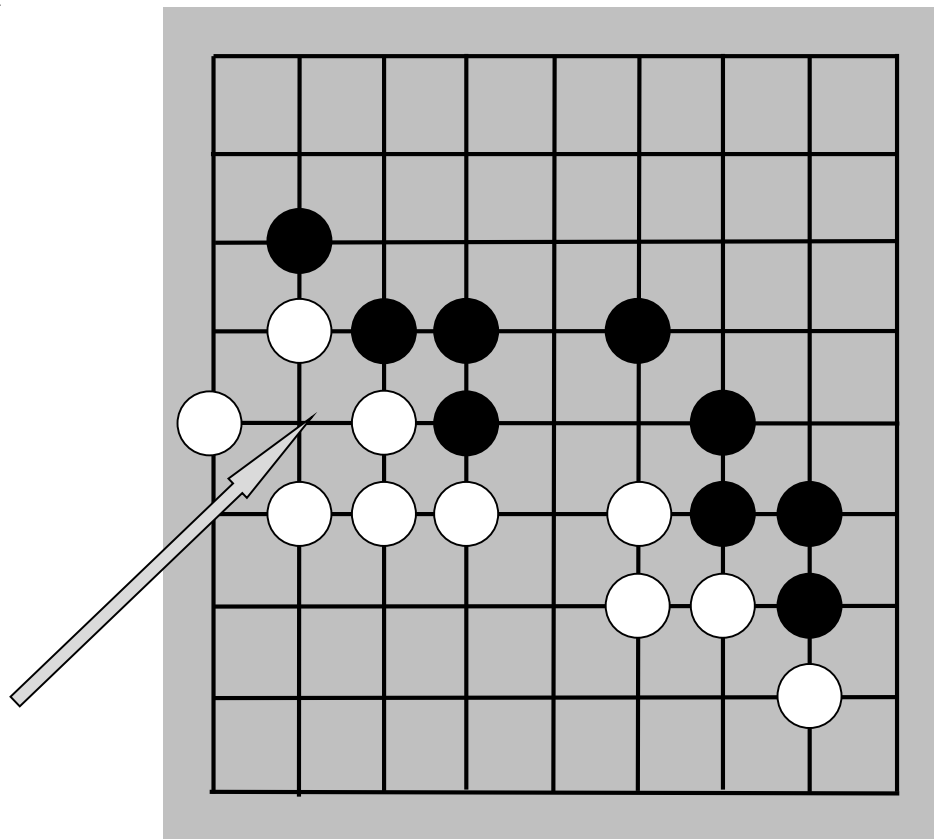




# Rules Overview Through a Game

## (capture 1)

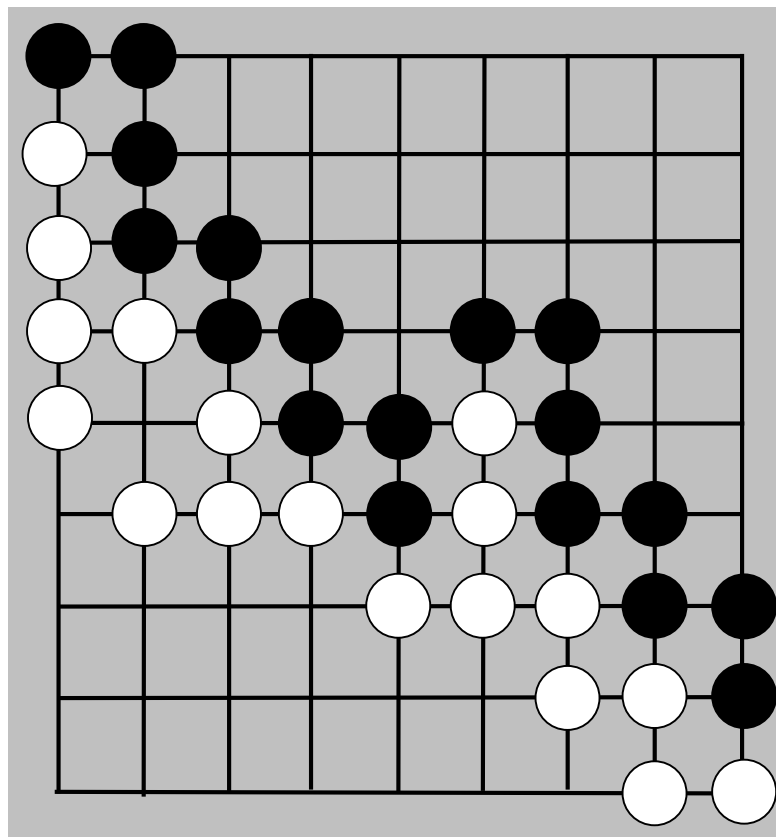
- White plays on the last liberty of the black stone which is removed



# Rules Overview Through a Game

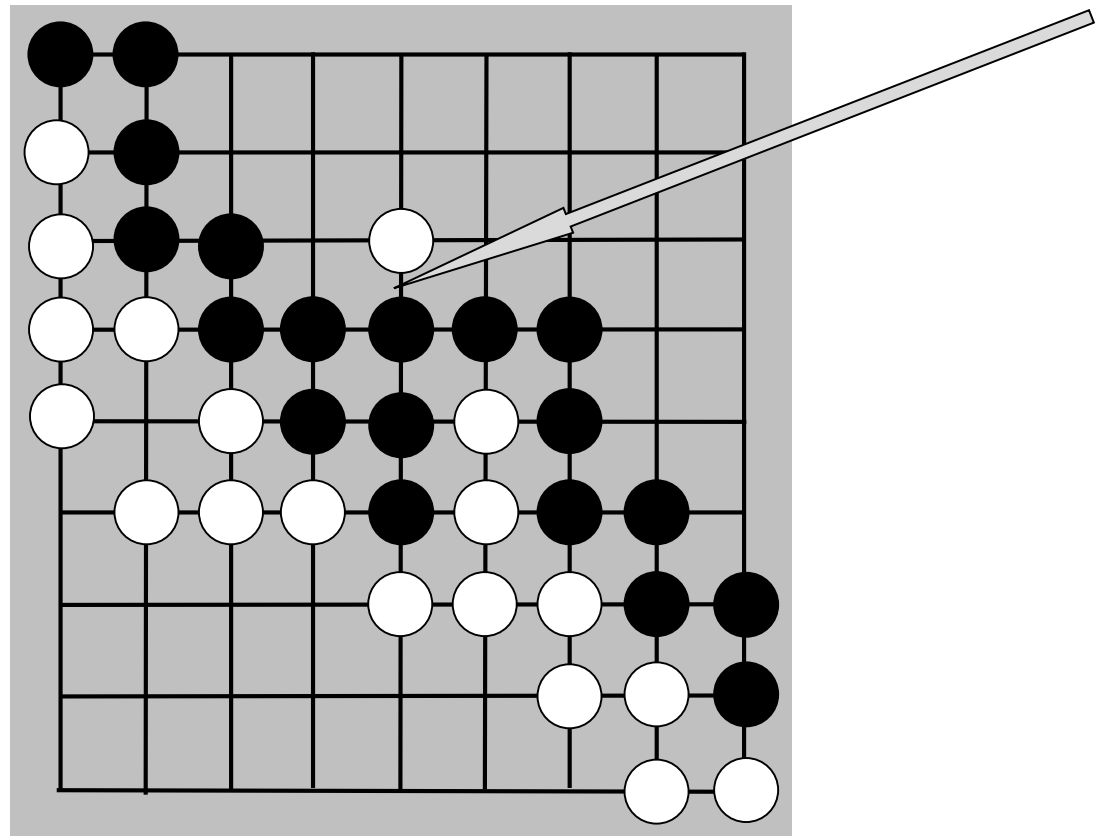
(human end of game)

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



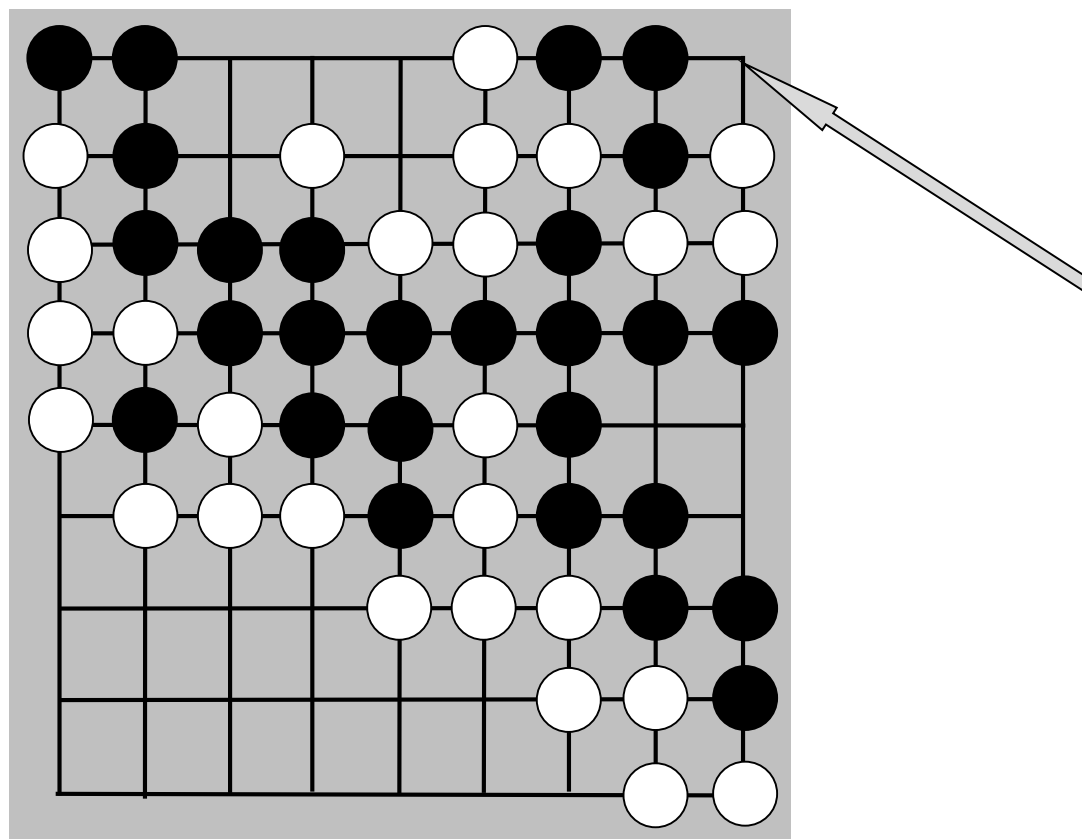
# Rules Overview Through a Game (contestation 1)

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



# Rules Overview Through a Game (contestation 2)

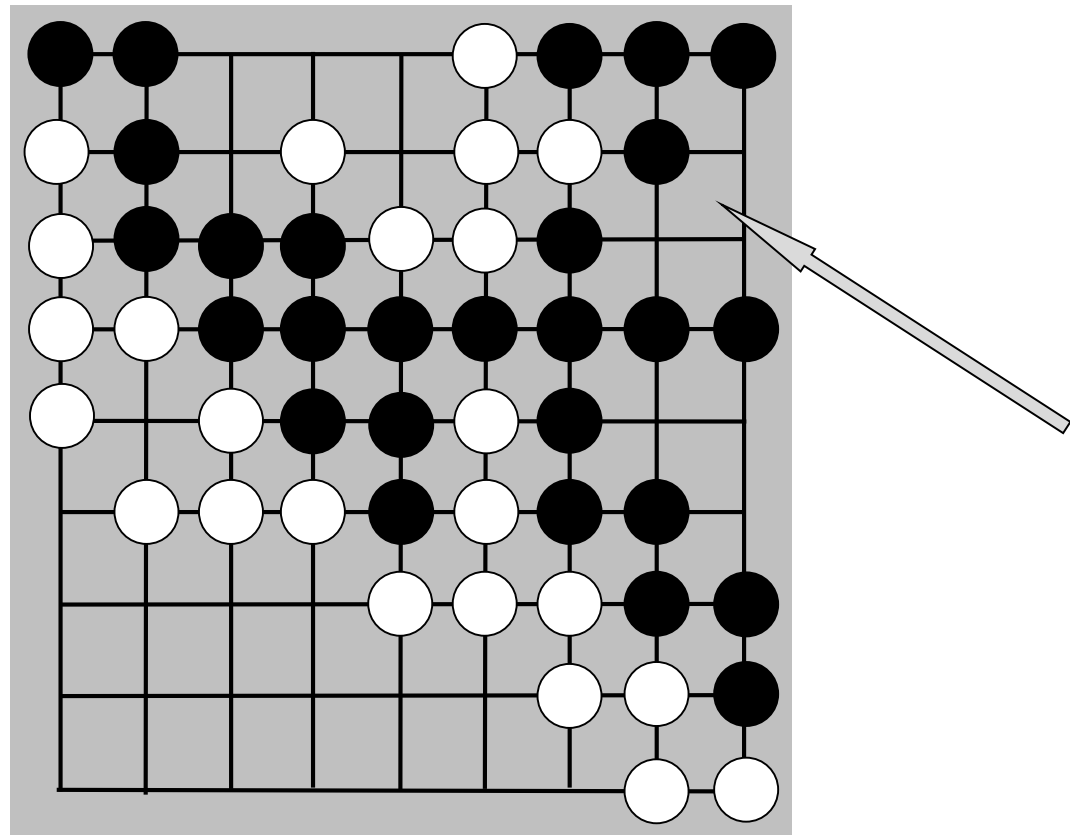
- White contests black territory, but the 3-stone white string has one liberty left



# Rules Overview Through a Game

## (follow up 1)

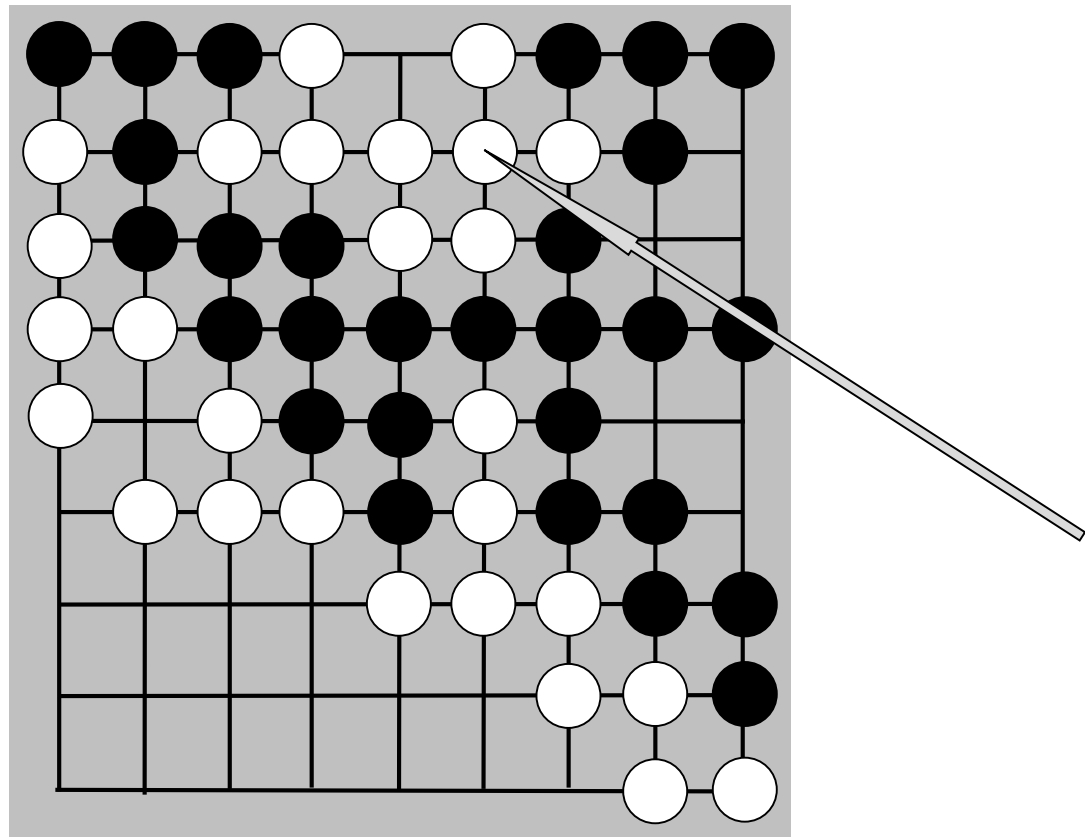
- Black has captured the 3-stone white string



# Rules Overview Through a Game

## (follow up 2)

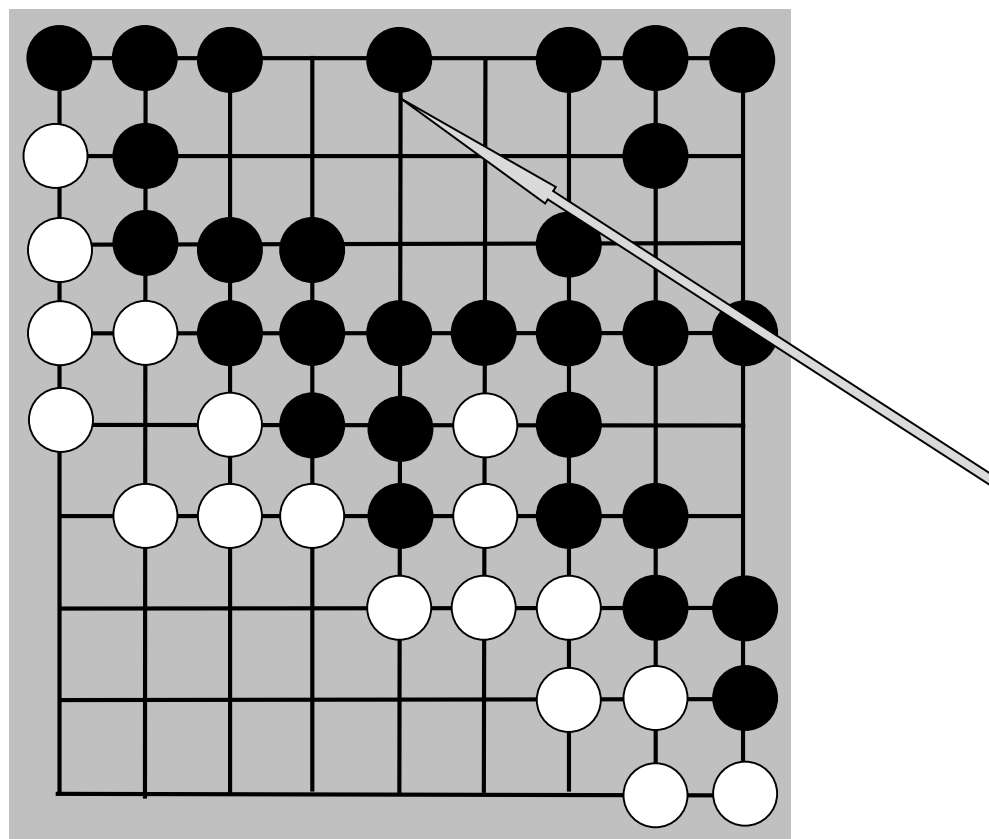
- White lacks liberties...



# Rules Overview Through a Game

## (follow up 3)

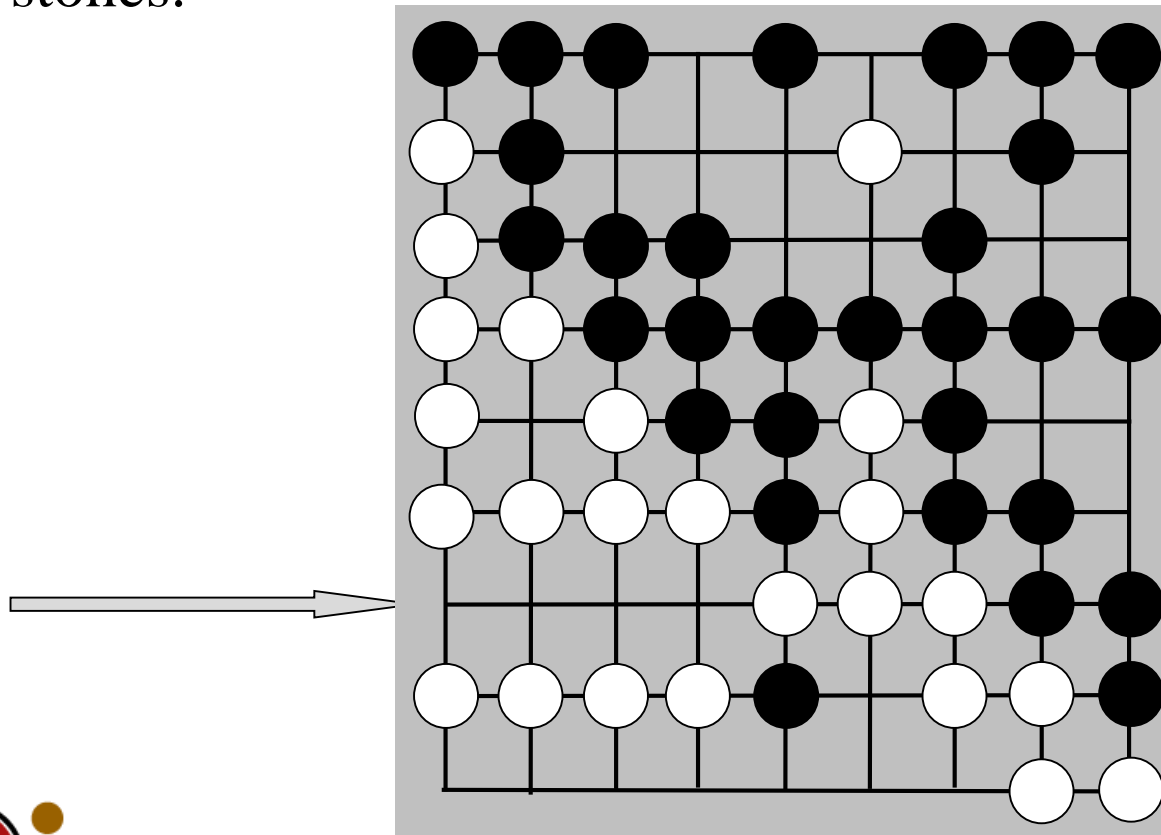
- Black suppresses the last liberty of the 9-stone string
- Consequently, the white string is removed



# Rules Overview Through a Game

## (follow up 4)

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

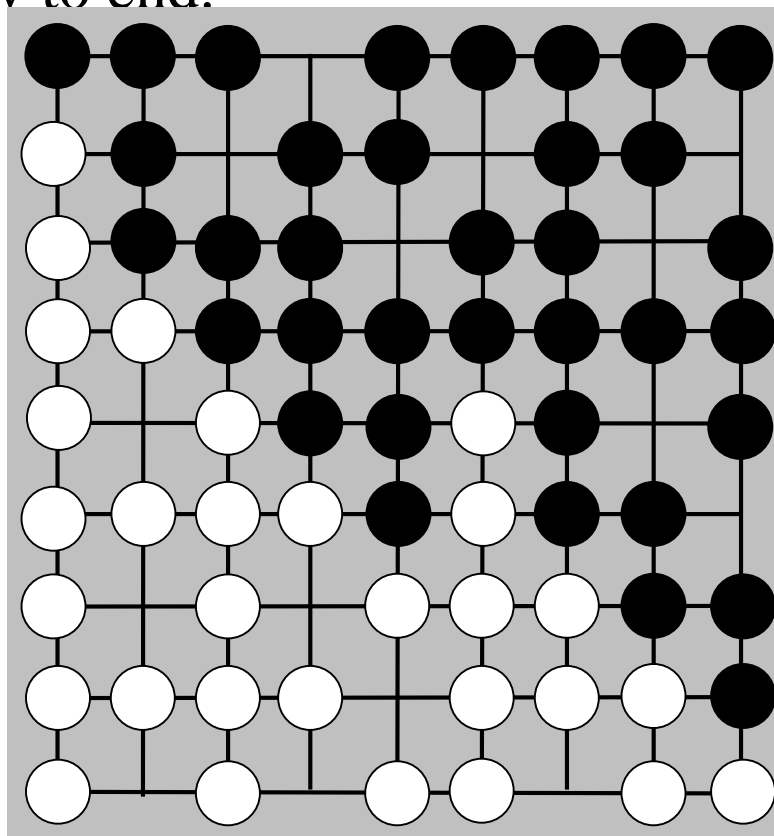




# Rules Overview Through a Game

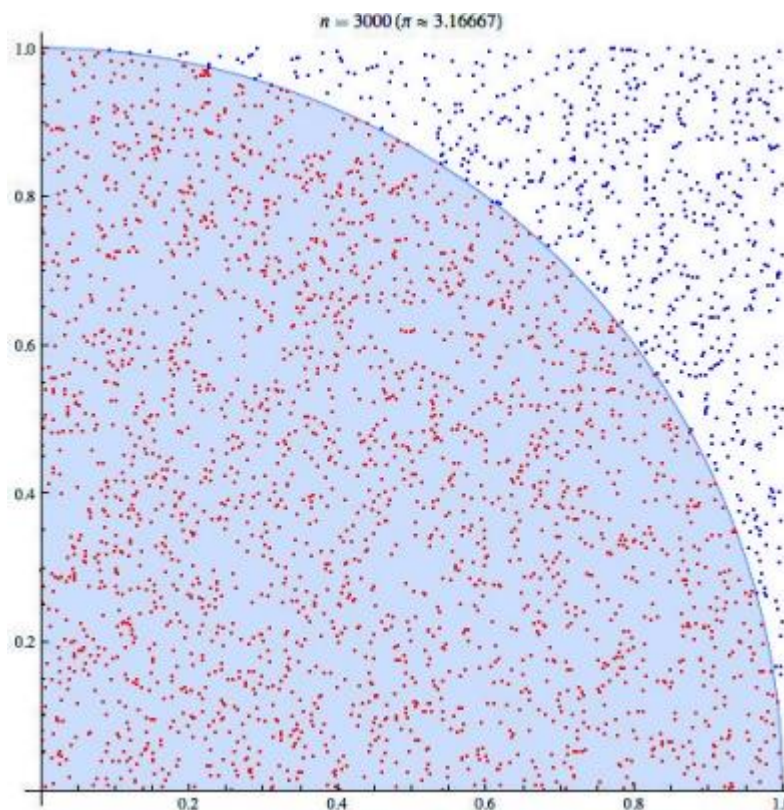
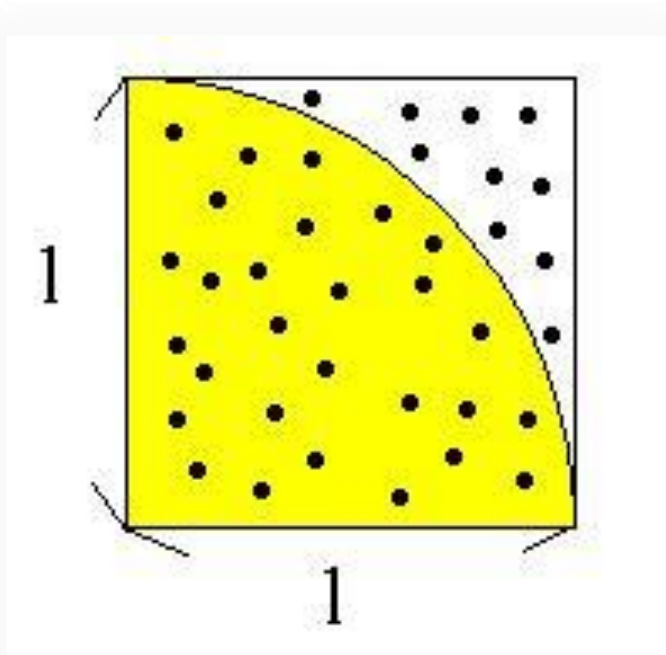
## (concrete end of game)

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



# Stochastics

- Calculate values based on stochastics.
  - Good example: calculate  $\pi$ .



# 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  $i$ :  $X_{i,1}, X_{i,2}, \dots$
- 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,

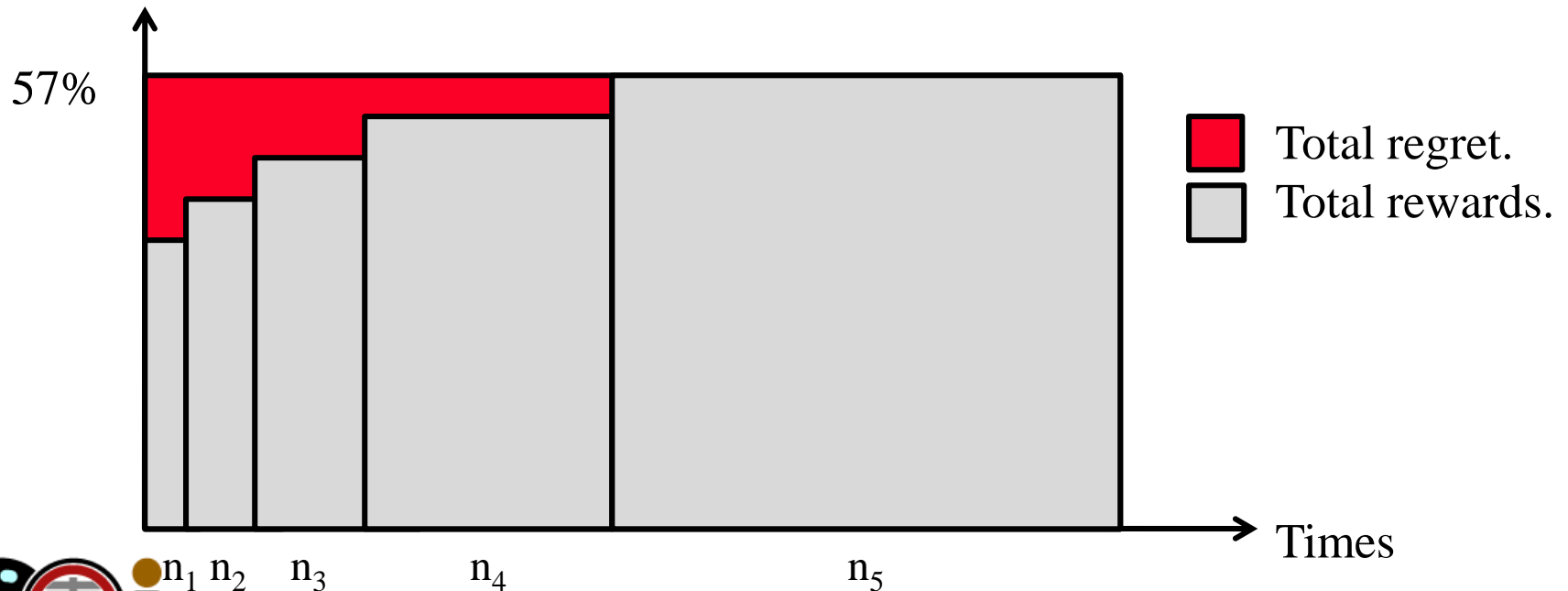
$$\bar{X}_{i,s} = \frac{1}{s} \sum_{j=1}^s X_{i,j} \quad , \quad \bar{X}_i = \bar{X}_{i,T_i(n)} \quad ,$$

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



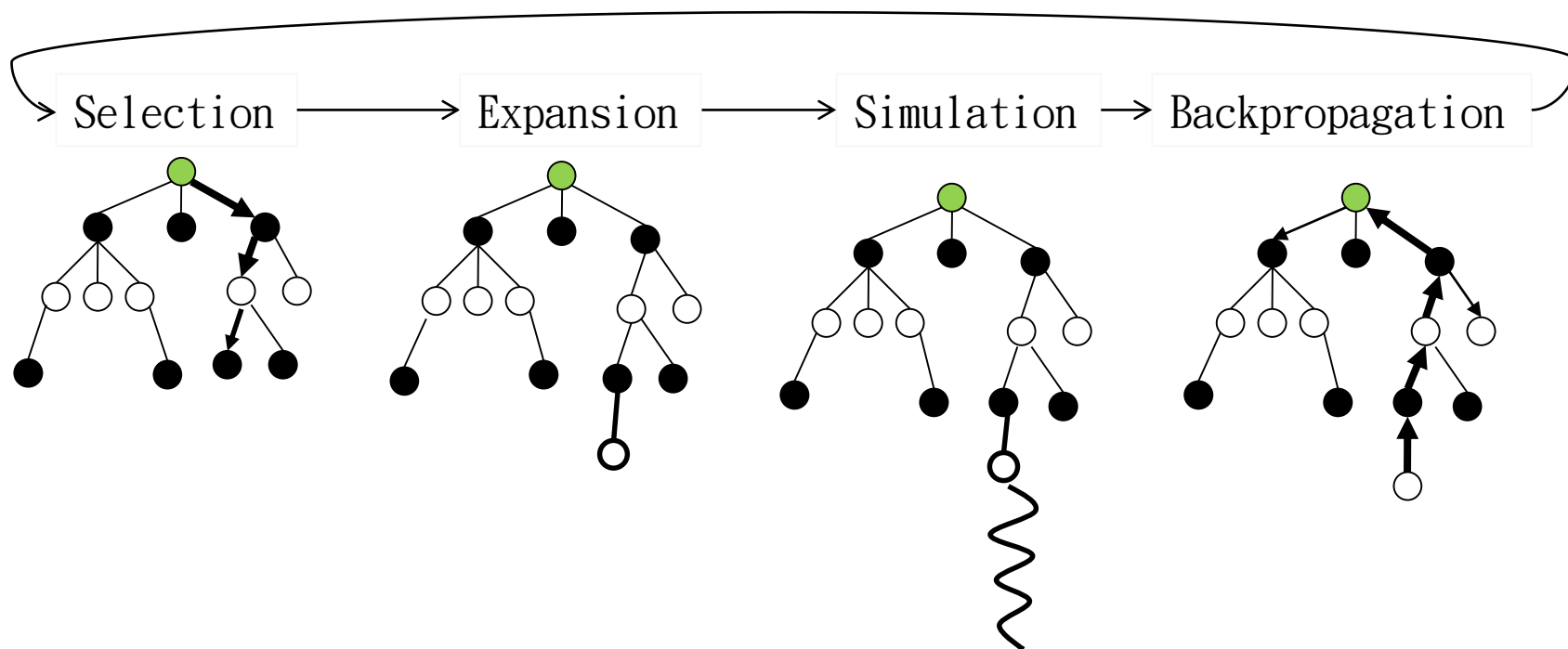
# Cumulative Regret

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



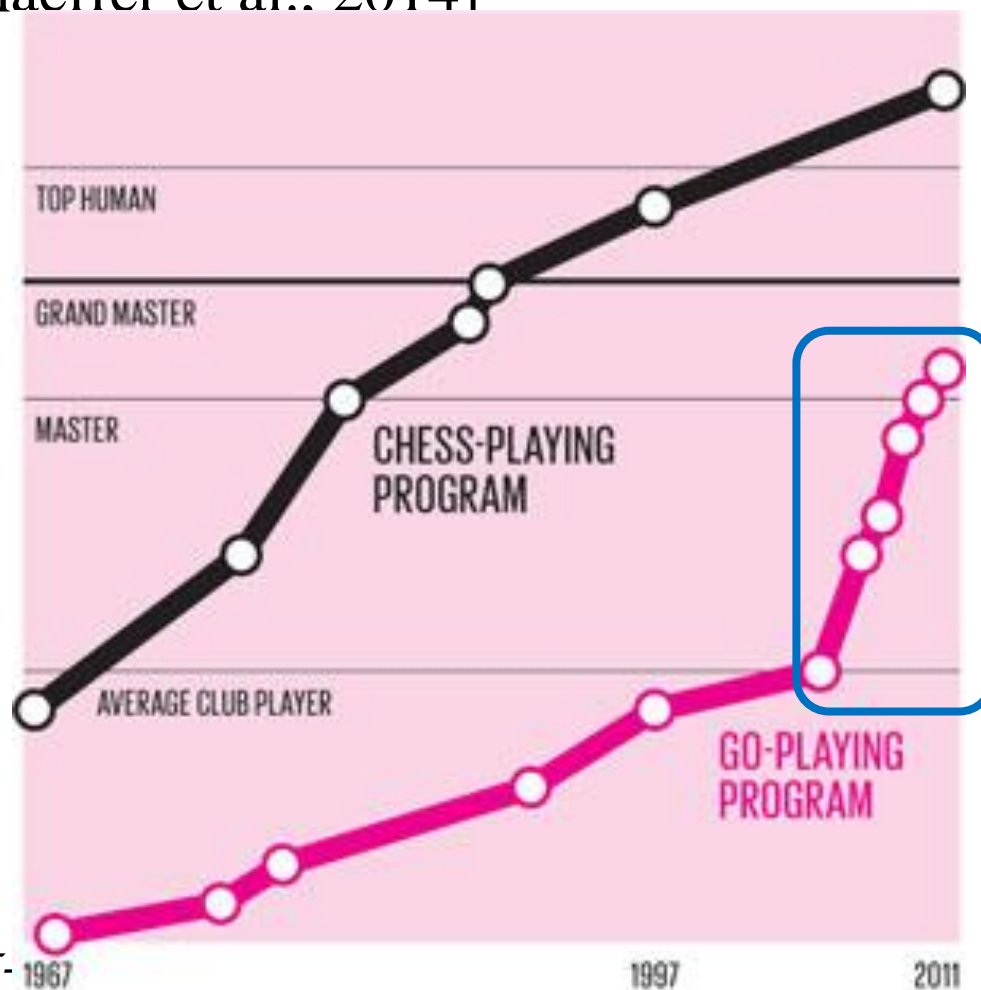
# Monte-Carlo Tree Search

- A kind of planning
- A kind of **Reinforcement learning**



# Strength of Go Program after MCTS

- [Schaeffer et al., 2014]



Strength grew fast, after MCTS.



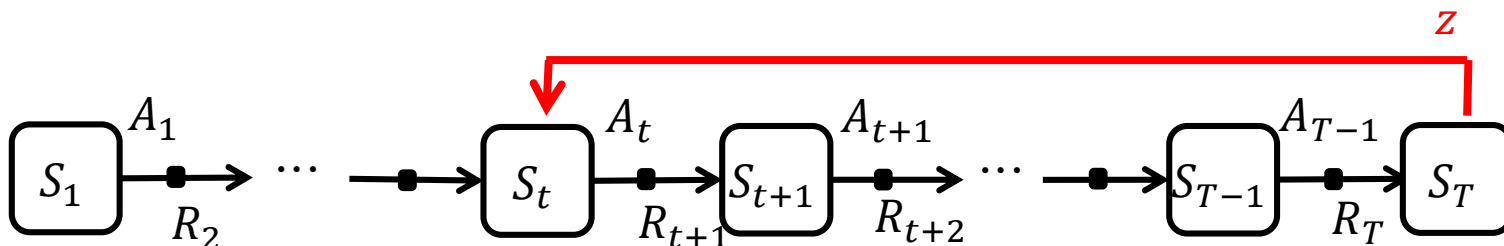


# Case Study: AlphaGo

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

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

- $\theta$ : network parameter.
- $\alpha$ : learning rate
- $z$ : the value of the episode
  - ▶ win/loss (1/-1) of the game

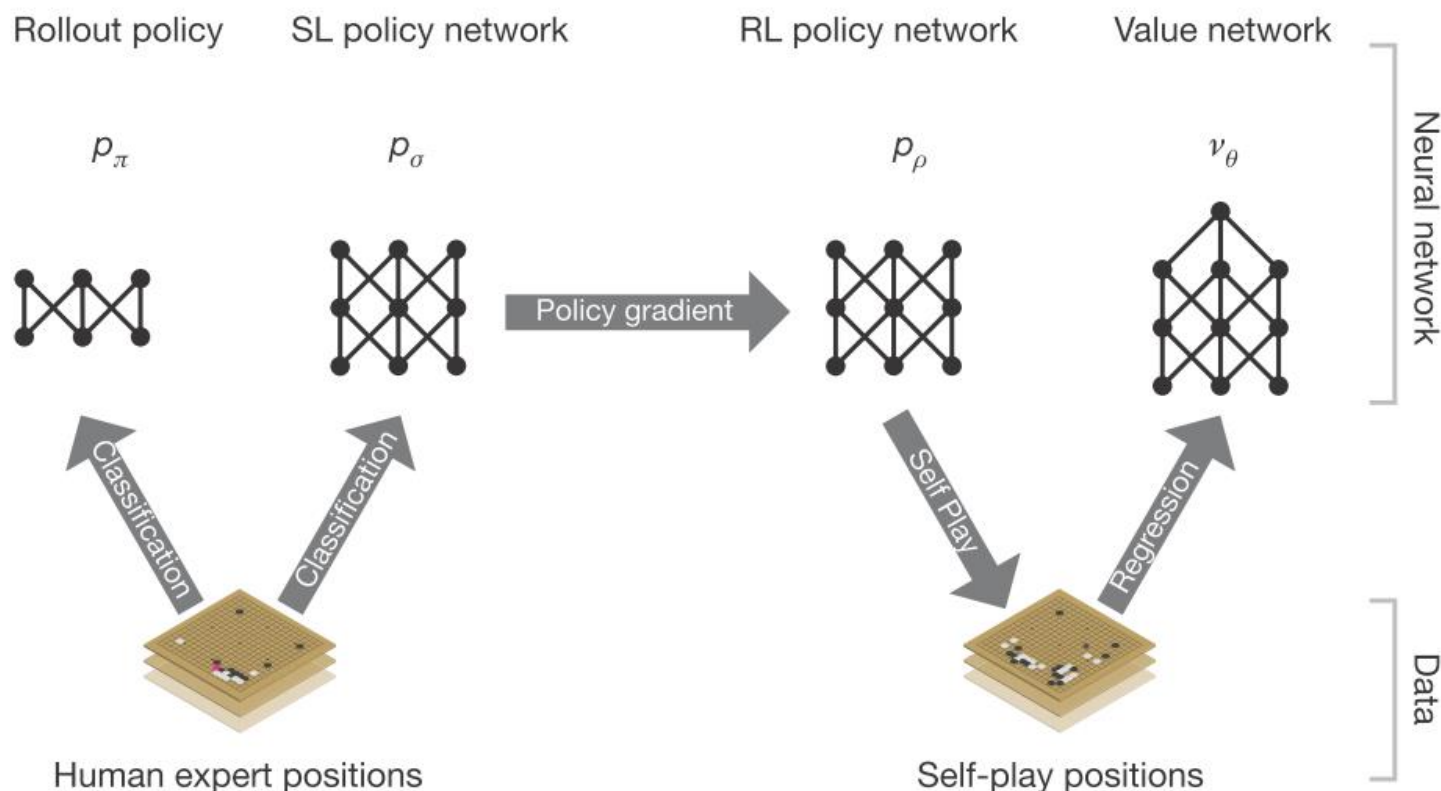


# AlphaGo的技術特點

- 採用 Monte-Carlo Tree Search (MCTS) → RL
  - 可以自主搜尋, 找尋最佳的下法, 避開陷阱.
- 利用 DCNN 辨識棋型, 學習高手的著手策略,
  - 找出最可能的下法, 並專注在這些下法. → DL
- 以 DCNN, 設計 “reinforcement learning (RL) network” (強化式學習) → DRL (Policy Gradient).
  - 自主學習: 利用自我對打, 學習更好更新的下法.
- 以 DCNN, 設計 “value network” (價值網路)
  - 學習盤面局勢之優劣 → DL



# Policy Network and Value Network

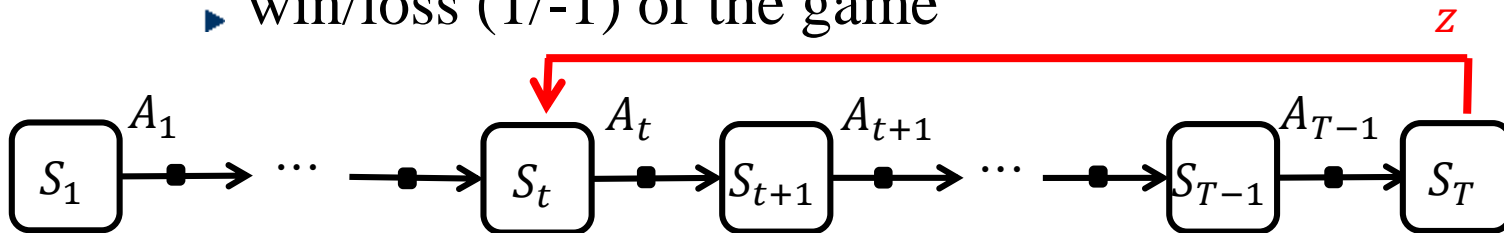


# RL Policy Network: AlphaGo

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

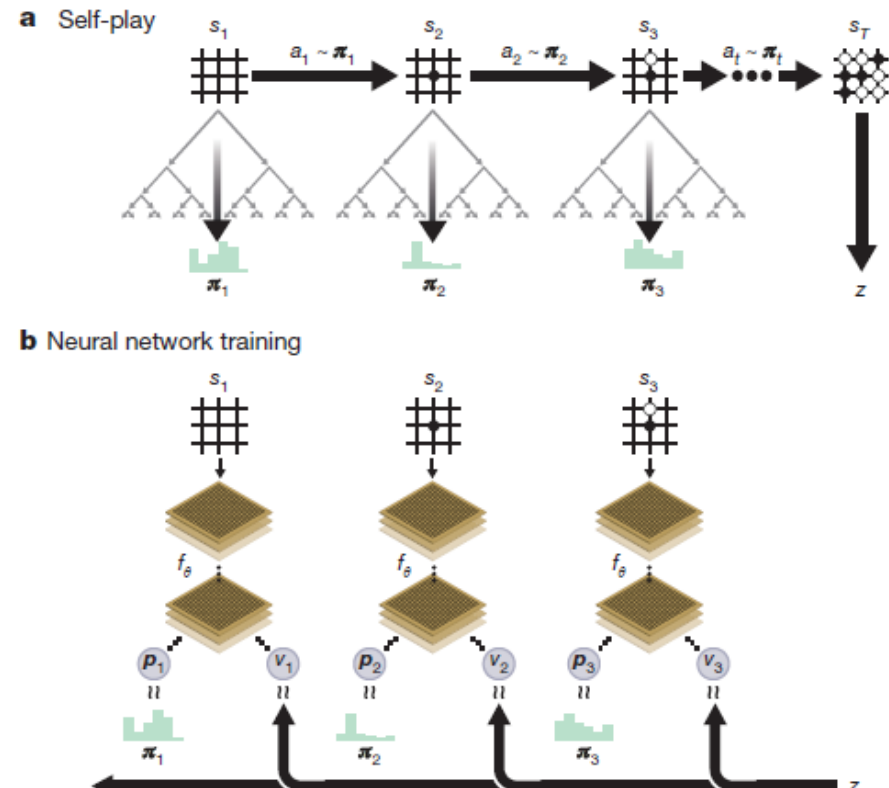
$$\Delta\theta = \alpha \nabla_{\theta} \log \pi_{\theta}(s_t, a_t) \cdot z$$

- $\theta$ : network parameter.
- $\alpha$ : learning rate
- $z$ : the value of the episode
  - ▶ win/loss (1/-1) of the game



# AlphaGo Zero

- 採用 Monte-Carlo Tree Search (MCTS) → RL
  - 可以自主搜尋, 找尋最佳的下法, 避開陷阱.
- Combine “value/policy network” → DRL



**Learn from Zero Knowledge!!!**

# Case Study: Atari 2600 Games

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



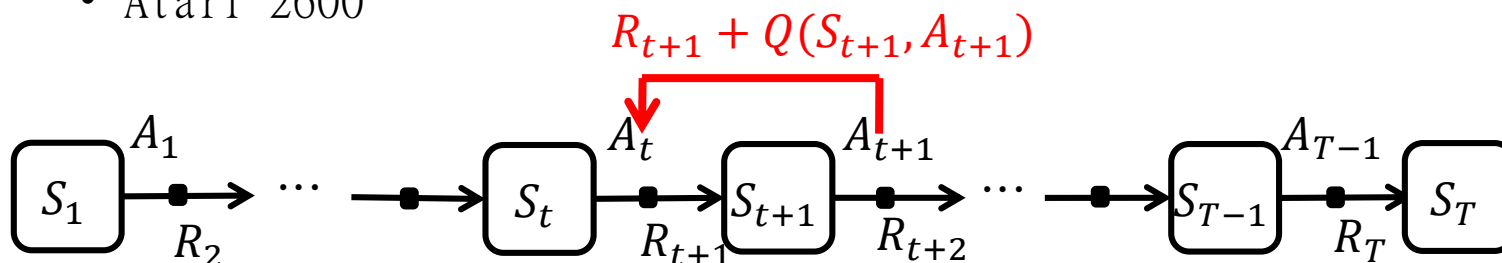
- Atari 2600



- Breakout



- Space Invaders



# Deep Q-Networks (DQN)

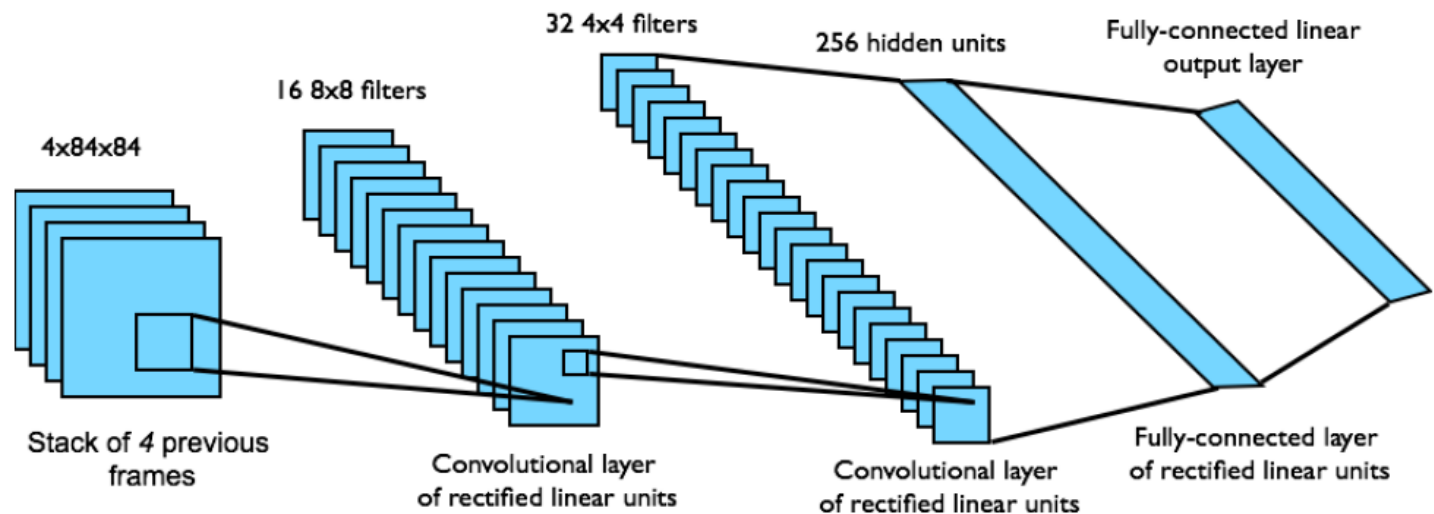
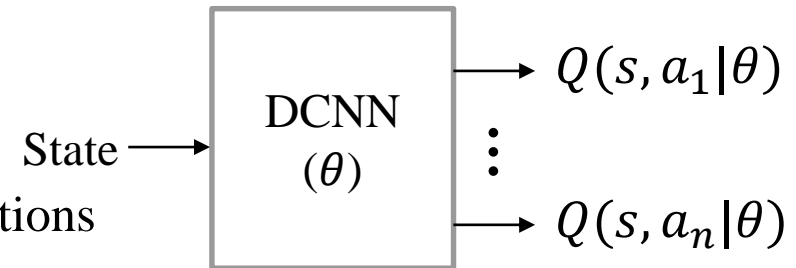
DQN uses experience replay and fixed Q-targets

- Take action according to  $\epsilon$ -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  $\theta^-$
- 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

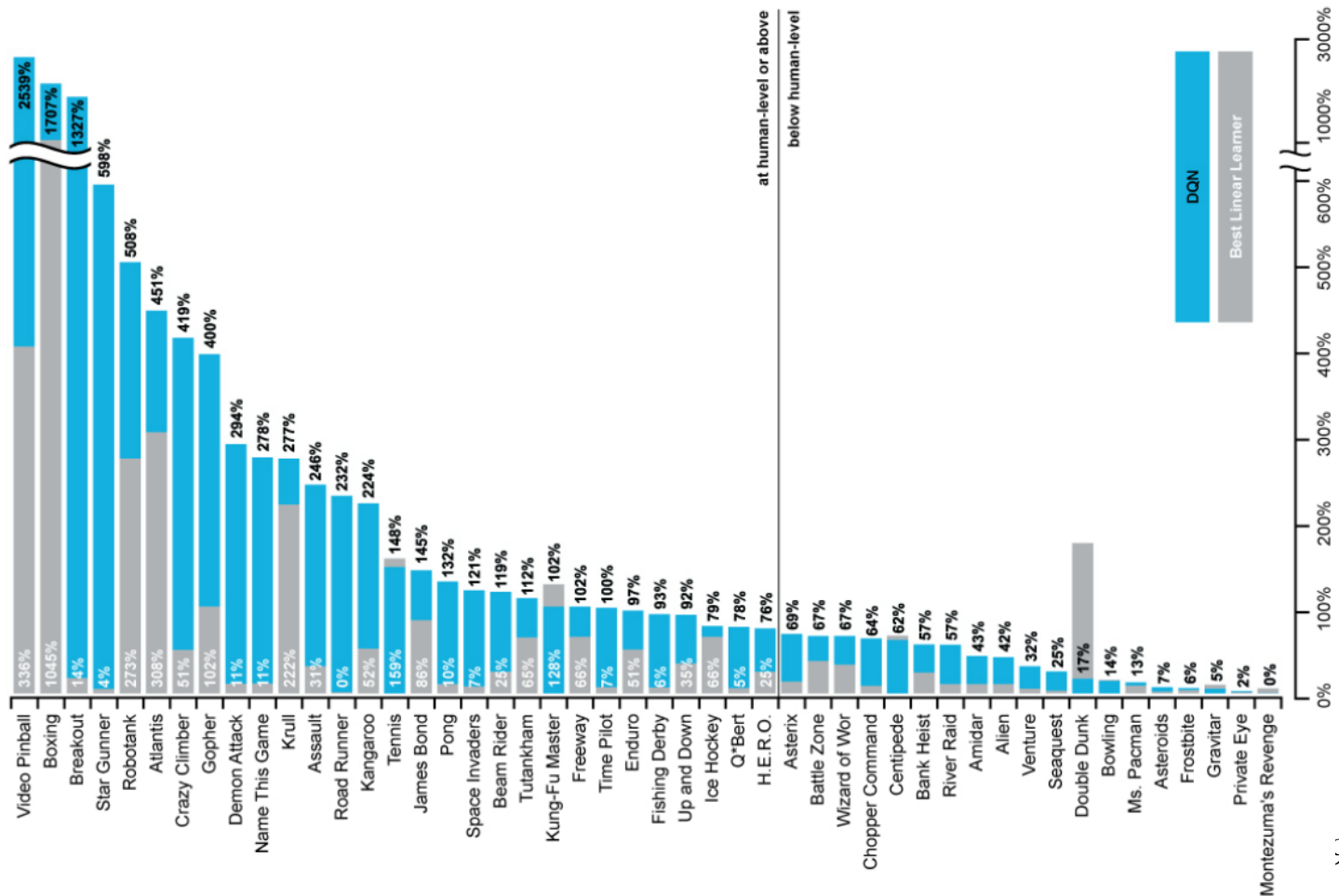
- End-to-end learning of values  $Q(s, a)$  from pixels  $s$
- Input state  $s$ 
  - stack of raw pixels from last 4 frames
- Output
  - $Q(s, a_i | \theta)$  for 18 joystick/button positions
- Reward
  - change in score for that step





# Performance of Deep Q-Learning

- Left (**stronger than human**)



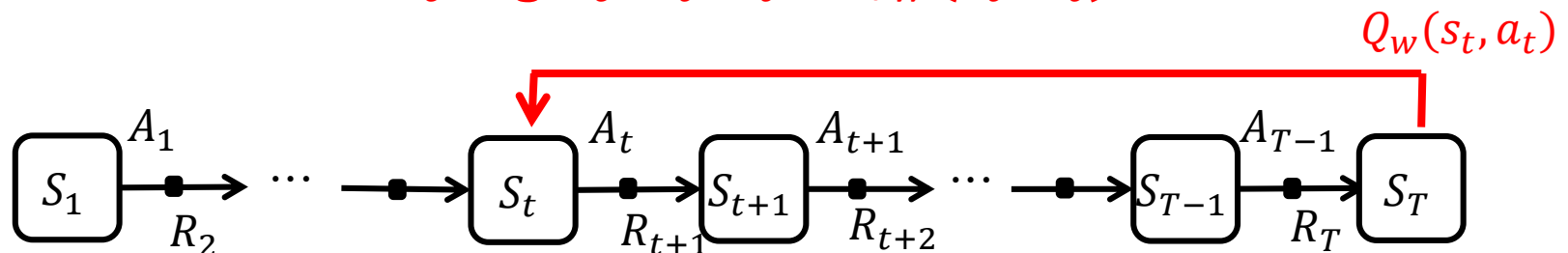
# Case Study: Robotics Grasping

[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

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



# Case Study: Robotics Grasping

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

**Marc Peter Deisenroth, Carl Edward Rasmussen, Dieter Fox**

**Learning to Control a Low-Cost Robotic Manipulator  
using Data-Efficient Reinforcement Learning**

**R:SS 2011**

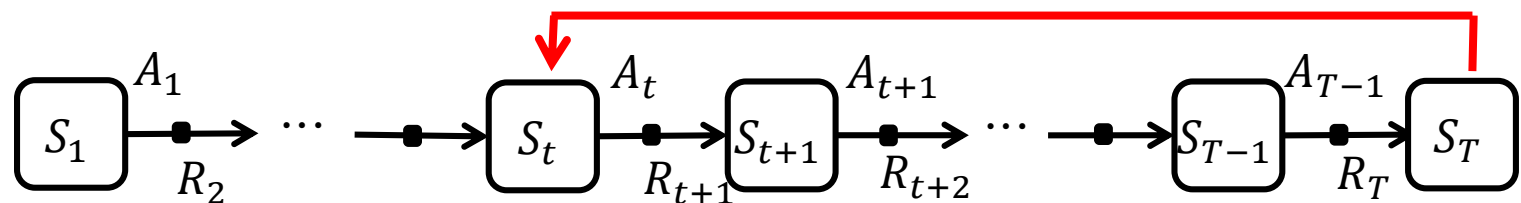


# 深度強化式學習應用類型

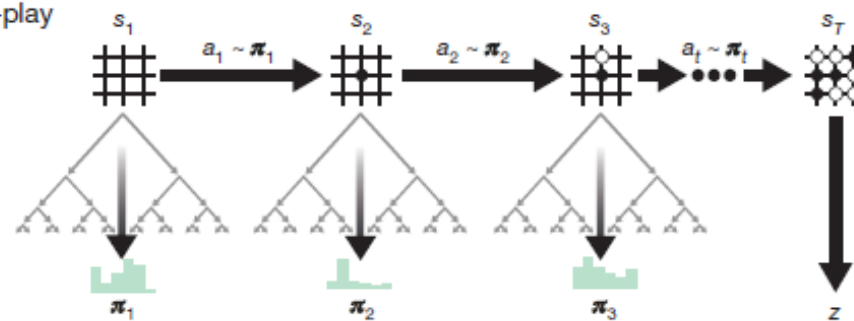
## Application Classification of Deep Reinforcement Learning

# Class 1

- Properties:
  - Model is well known or defined
  - Simulator exists.
- Applications: Games, Education, etc.
- Possible Solutions: AlphaZero-like.



a Self-play

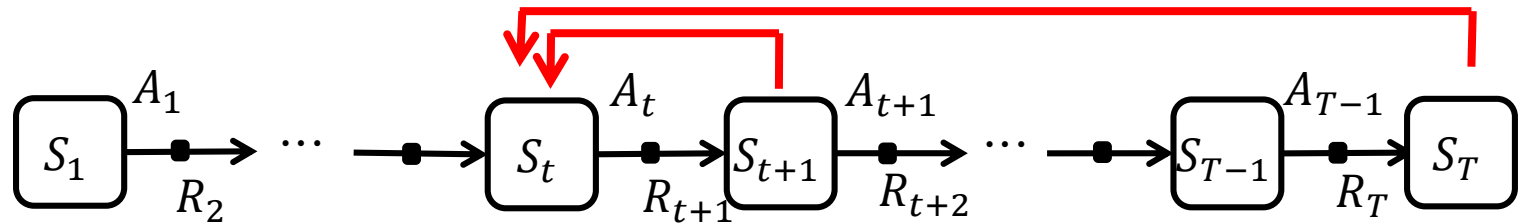


# Related Reinforcement Learning Techniques

- TD Learning
- Monte-Carlo Learning
- POMDP
- Monte-Carlo Tree Search (MCTS)
- AlphaZero

## Class 2

- Properties:
  - Model is unknown or too complex.
  - Simulator exists.
- Applications: Video Games, Robots with Simulator, etc.
- Possible Solutions: (next page)



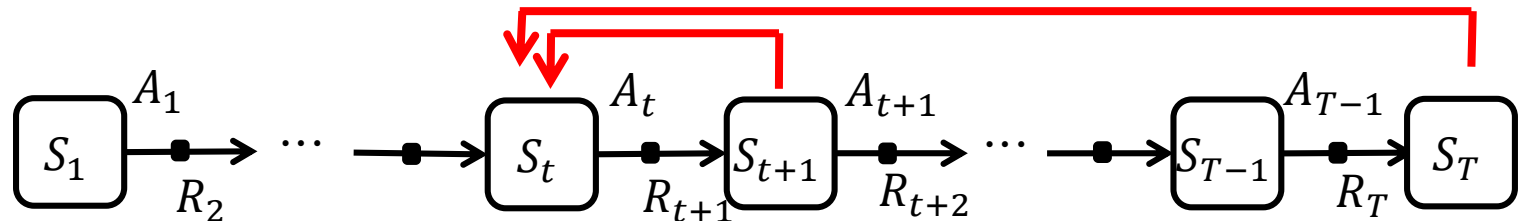
# More Related Deep Reinforcement Learning Techniques

- Deep Q Network (DQN)
- Double DQN (DDQN)
- Actor-Critic
- Dueling Network
- Deep Deterministic Policy Gradient (DDPG)
- Asynchronous Advantage Actor-Critic (A3C)
- Trust Region Policy Optimization (TRPO)
- Proximal Policy Optimization (PPO)



# Class 3

- 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, Drone, Auto-driving, etc.
- Solutions: (see next page)



# More Related Machine Learning Techniques

- Curriculum learning
- Transfer Learning
- Imitation Learning
- Behavior Cloning
- Dagger
- GAIL (like GAN)