Deep Reinforcement Learning and its Applications in Games

Yuandong Tian

Facebook AI Research

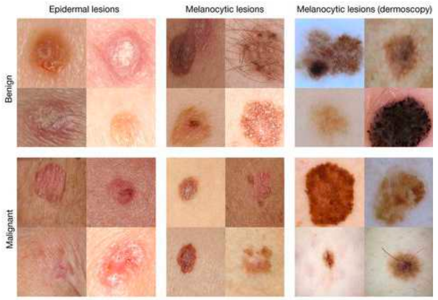
Overview

- Introduction: AI and Games
- Basic knowledge in Reinforcement Learning
  - Q-learning
  - Policy gradient
  - Actor-critic models
  - Game related approaches
- Case study
  - AlphaGo Fan and AlphaGo Zero
  - Our work
    - DarkForest
    - Doom AI bot
    - ELF platform

Part I: Introduction

AI works in a lot of situations

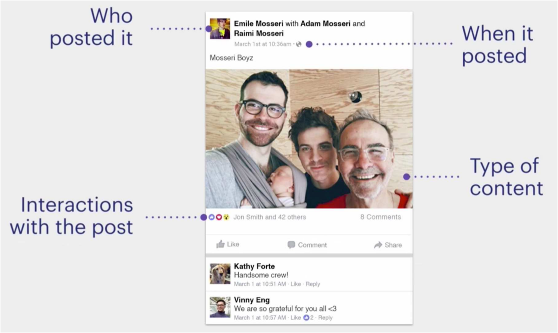
Object Recognition



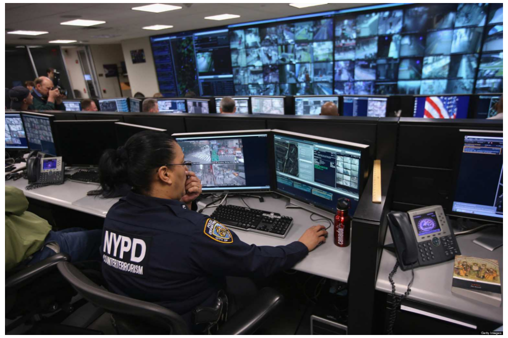
Medical

Translation

Speech Recognition

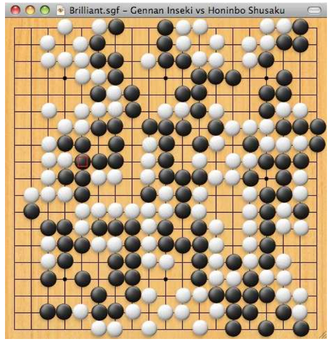


Personalization



Surveillance

Smart Design

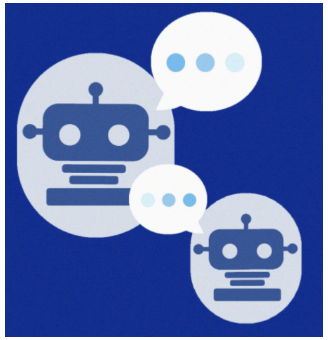


Board game

What AI still needs to improve

Home Robotics

Autonomous Driving



ChatBot

StarCraft

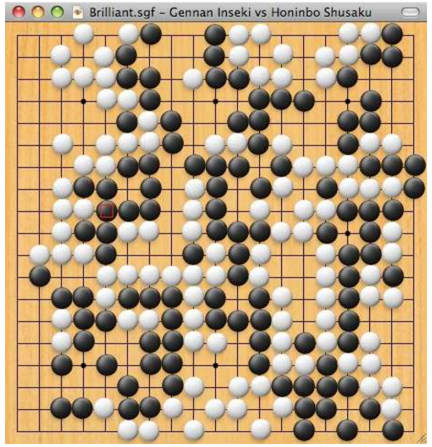
Question Answering

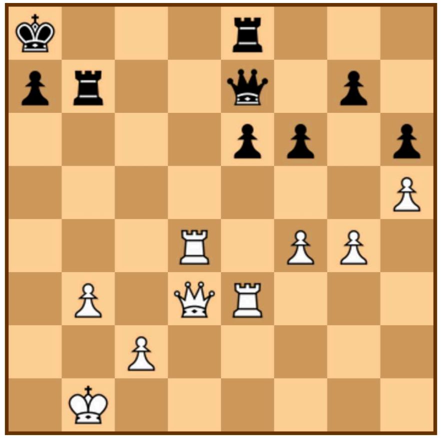
Less supervised data

Complicated/Unknown environments with lots of corner cases.

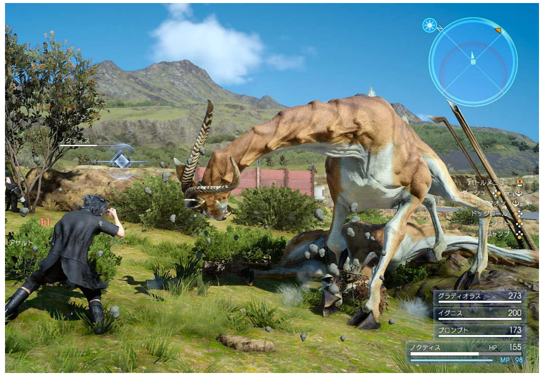
Common Sense

The Charm of Games



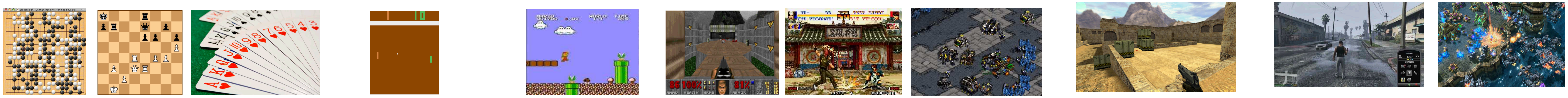


Complicated long-term strategies.



Realistic Worlds

Game as a Vehicle of AI



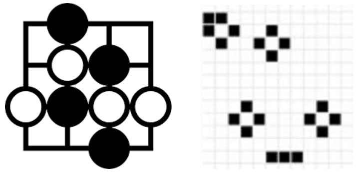
Infinite supply of *fully* labeled data

Controllable and replicable

Low cost per sample

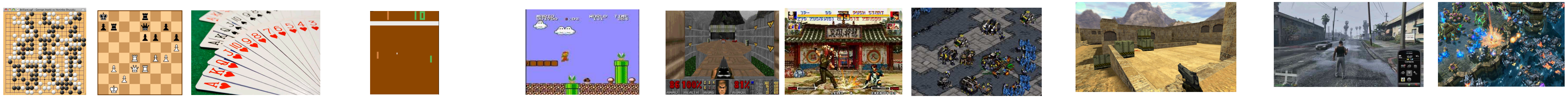
Faster than real-time

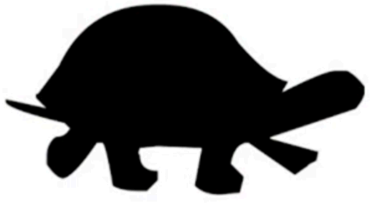
Less safety and ethical concerns



Complicated dynamics with simple rules.

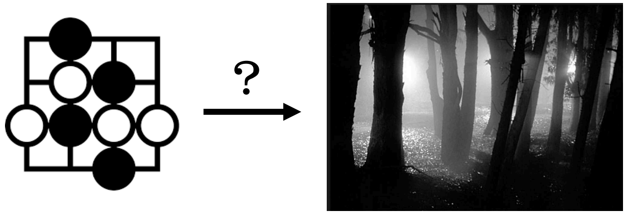
Game as a Vehicle of AI



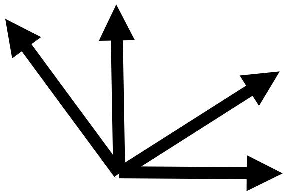


Algorithm is slow and data-inefficient

Require a lot of resources.

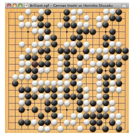
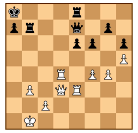


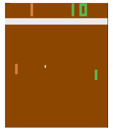
Abstract game to real-world



Hard to benchmark the progress

Game as a Vehicle of AI

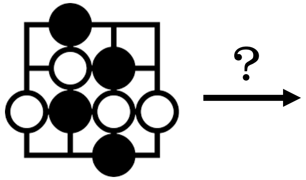







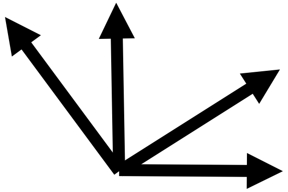



Algorithm is slow and data-inefficient



Abstract game to real-world

Require a lot of resources.



Hard to benchmark the progress

Better Algorithm/System

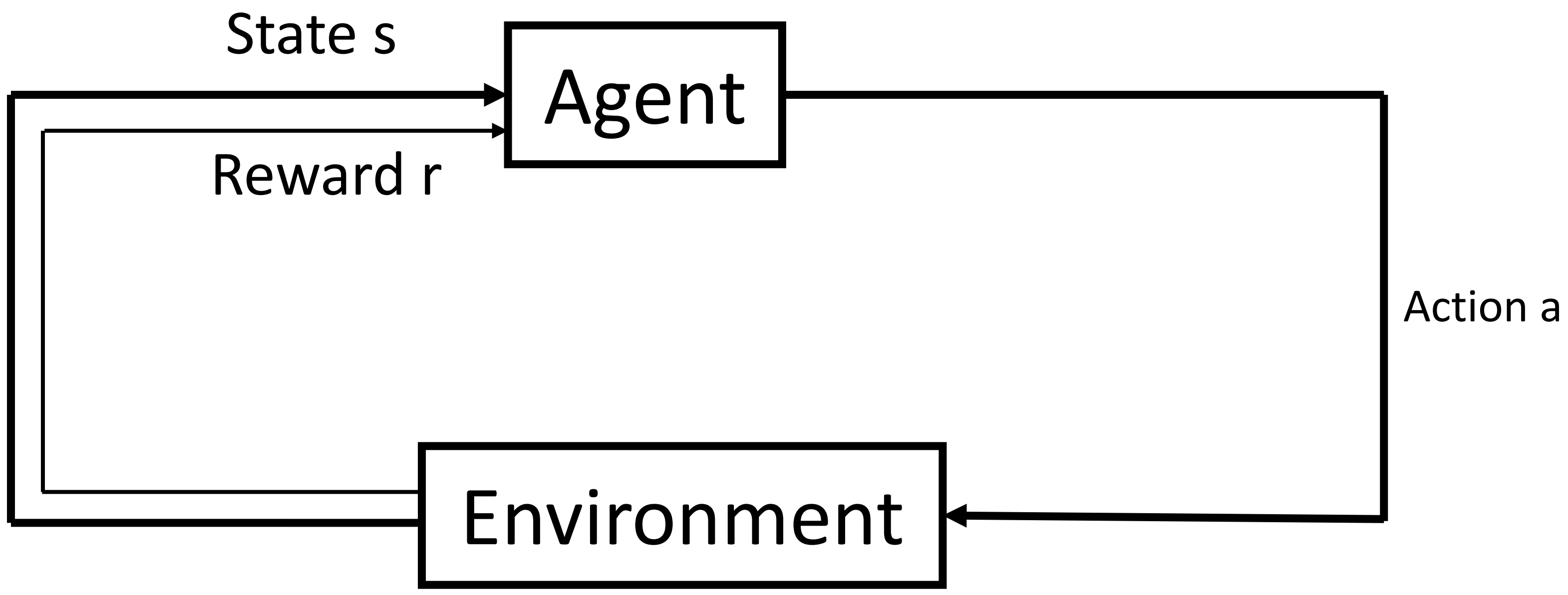
Better Environment

What’s new in Game environment?

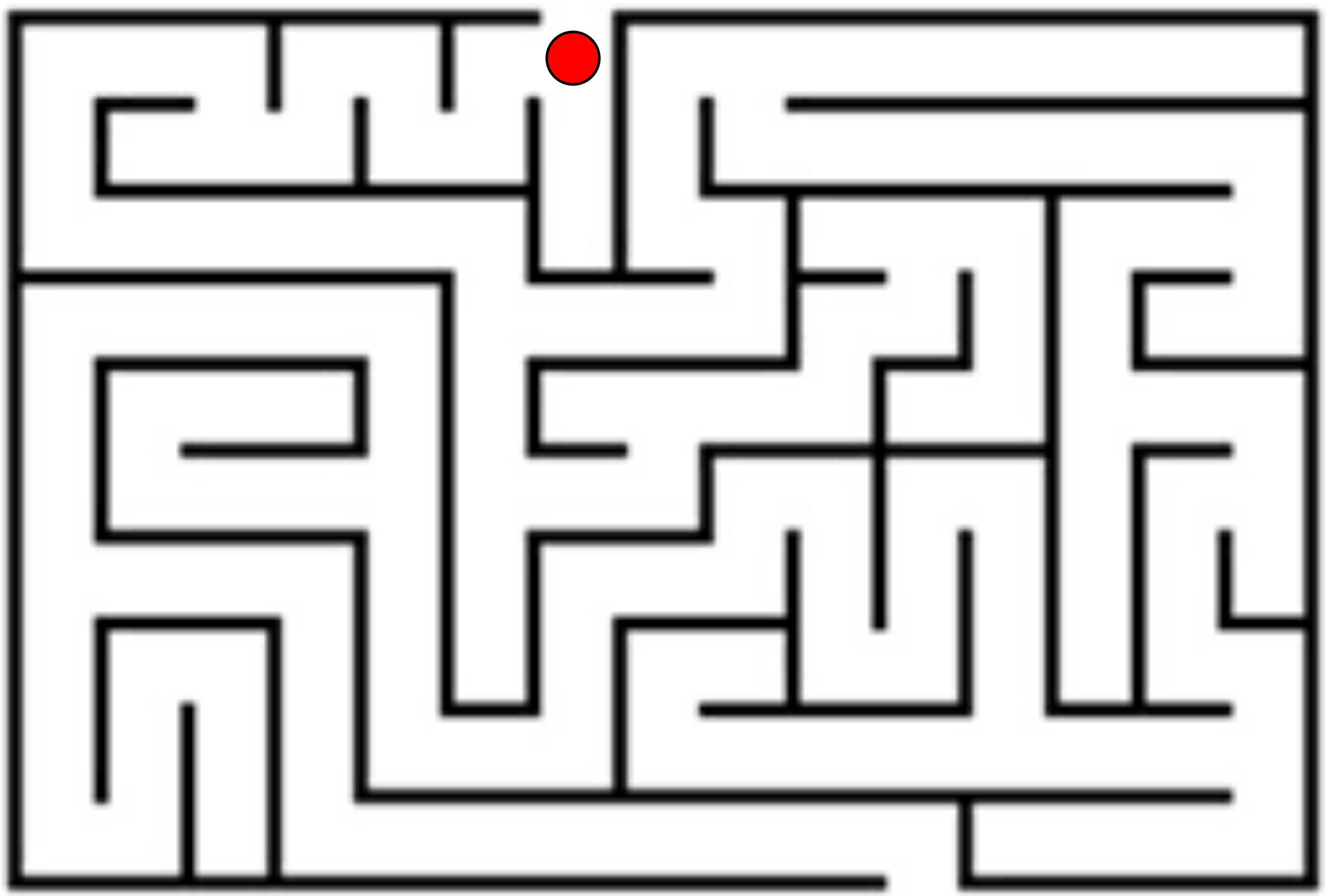
- Data are generated on the fly
- Agent not only learns from the data, but also choose which data to learn.

Part II: Reinforcement Learning

What is Reinforcement Learning?



What is Reinforcement Learning?

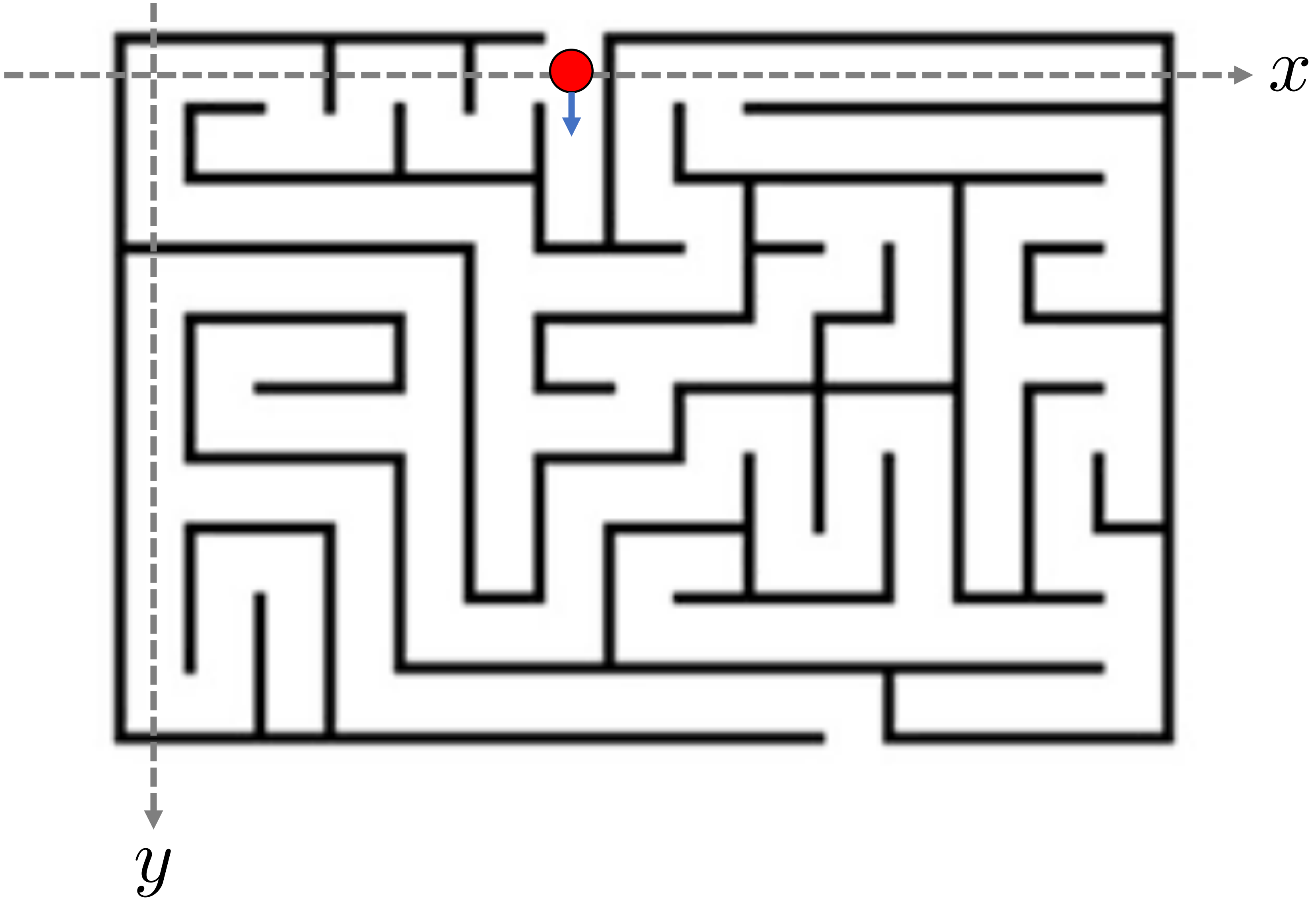


State:
where you are?

Action:
left/right/up/down

Next state:
where you are after the action?

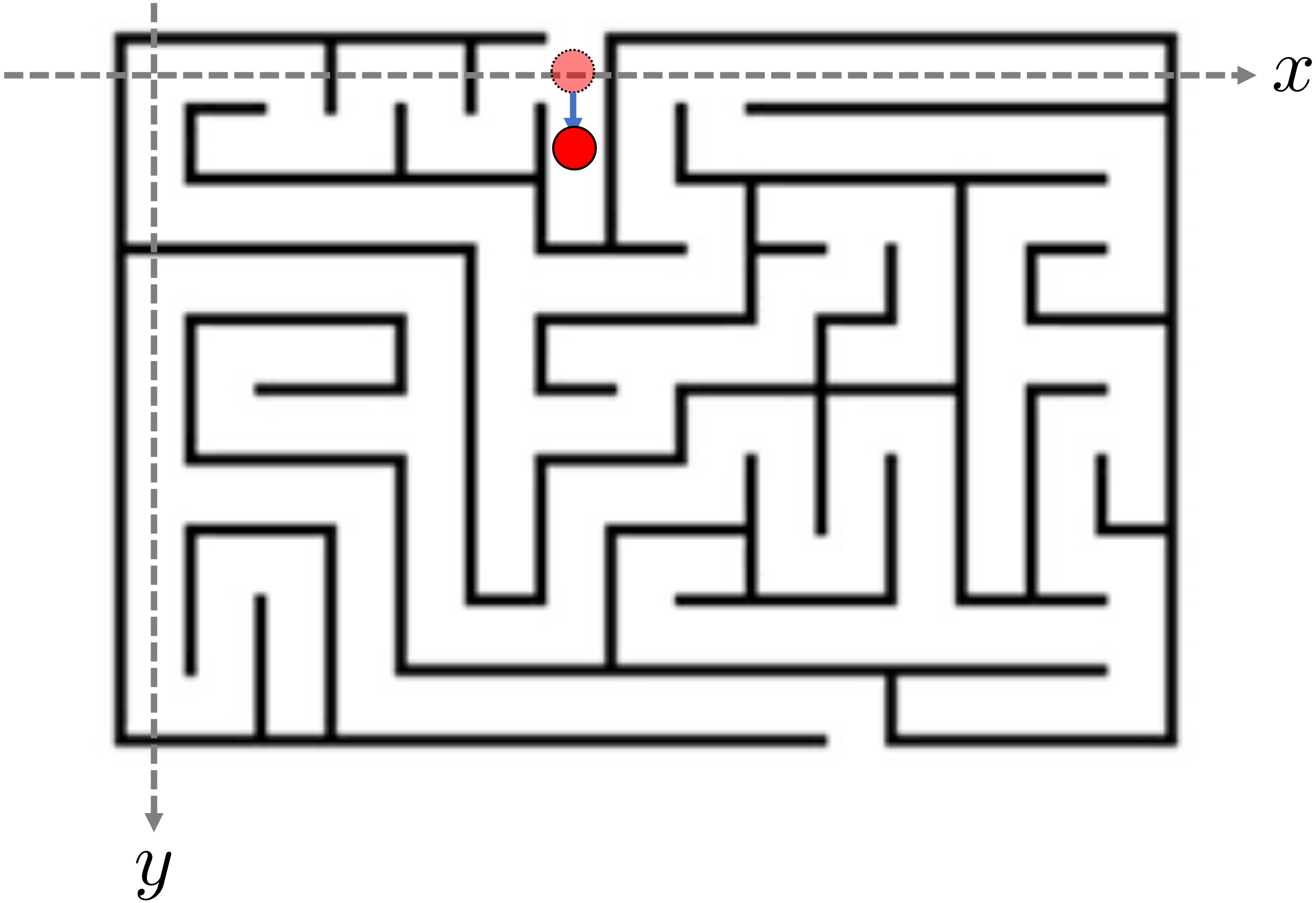
What is Reinforcement Learning?



State:
(x, y) = (6, 0)

Actions:
Left: x -=1
Right: x += 1
Up: y -= 1
Down: y += 1

What is Reinforcement Learning?



State:

$$s = (x, y) = (6, 0)$$

Actions:

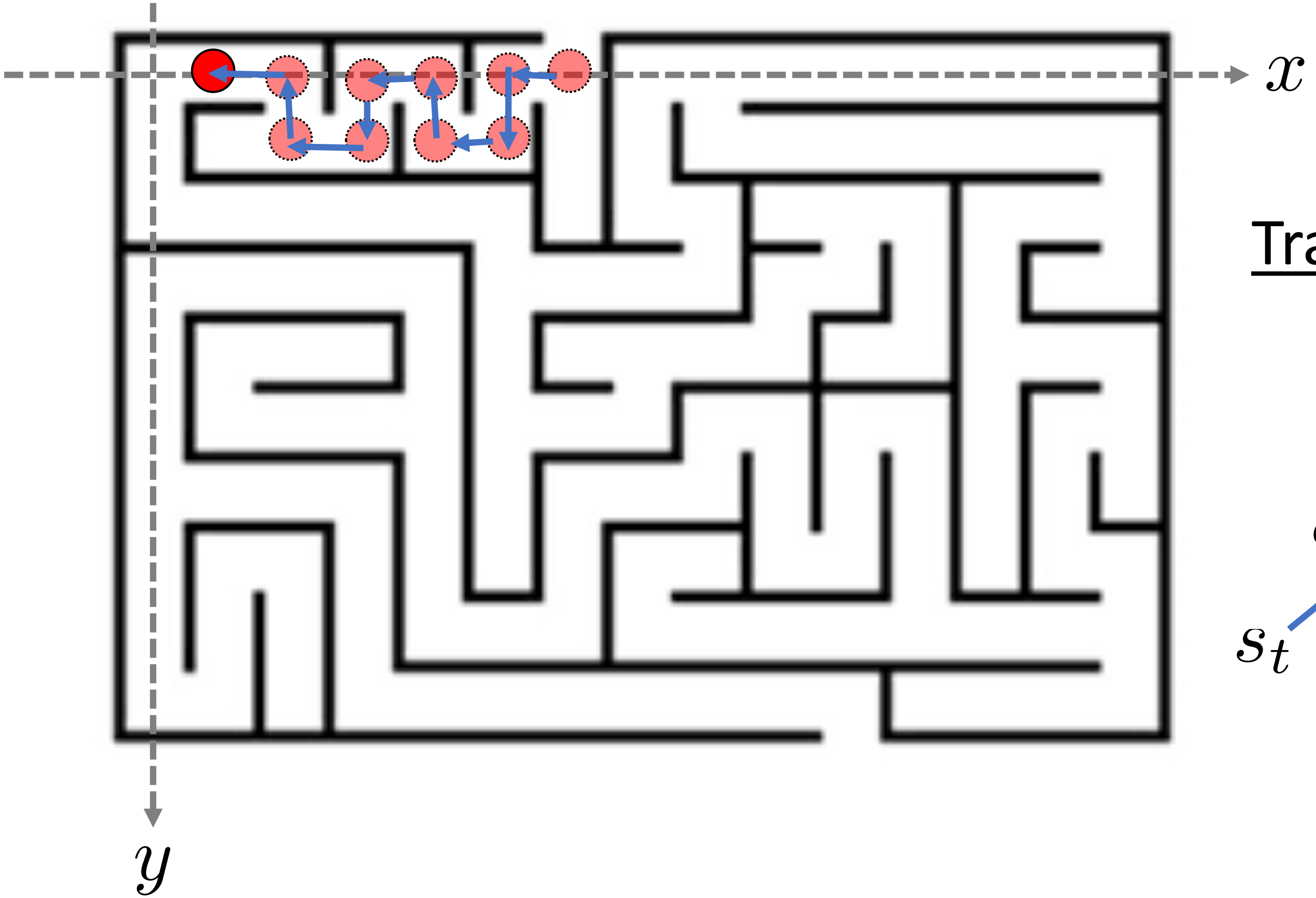
Left: $x \leftarrow x - 1$

Right: $x \leftarrow x + 1$

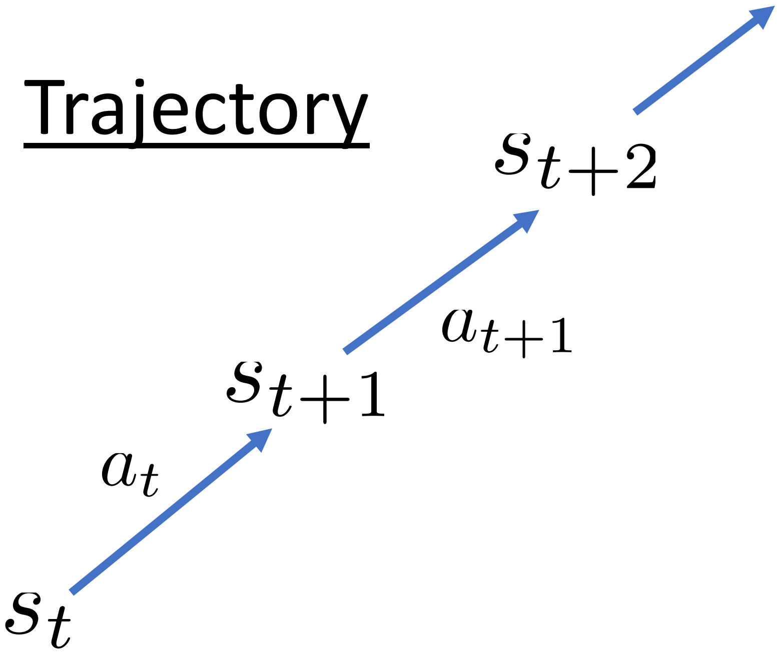
Up: $y \leftarrow y - 1$

Down: $y \leftarrow y + 1$

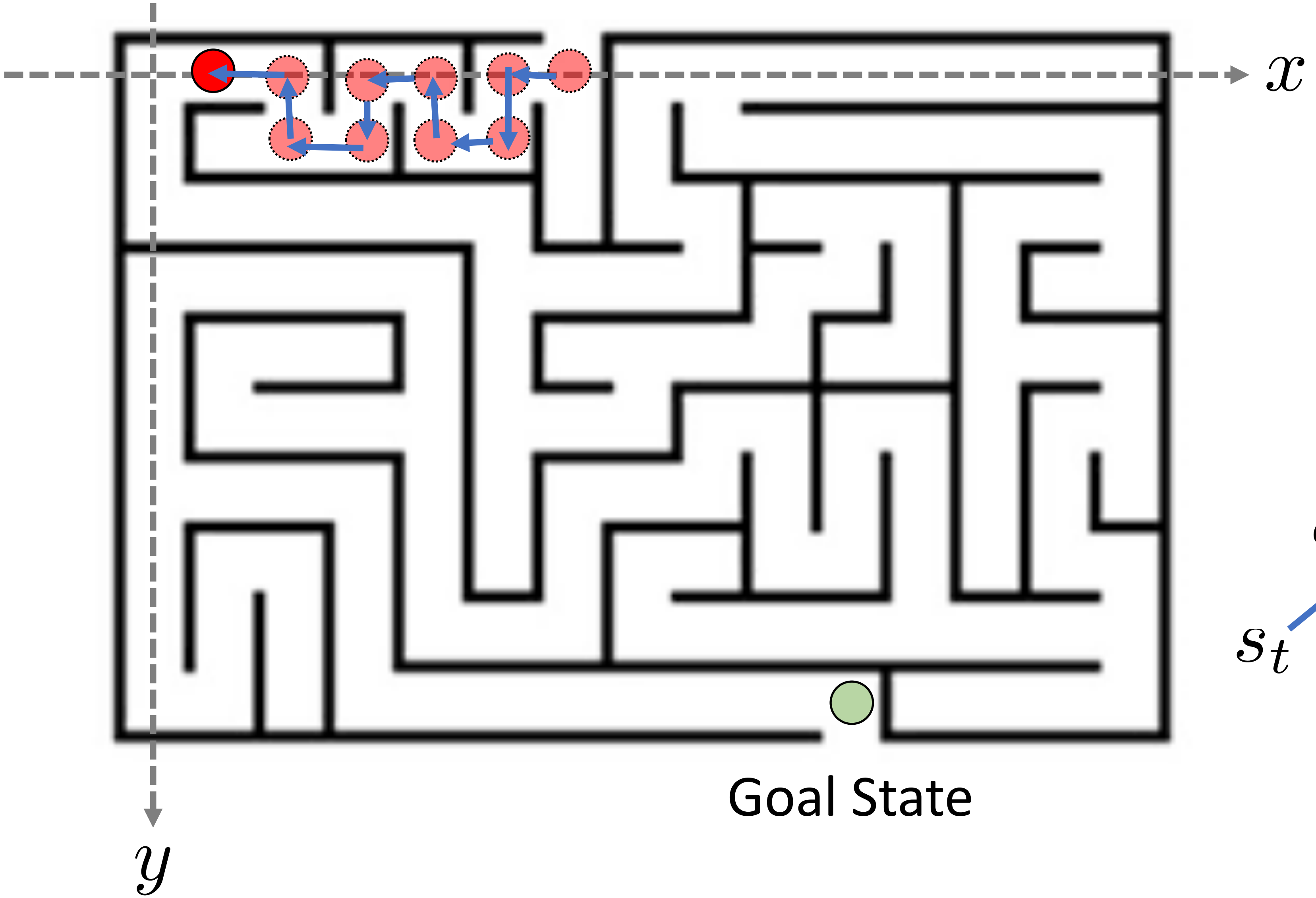
What is Reinforcement Learning?

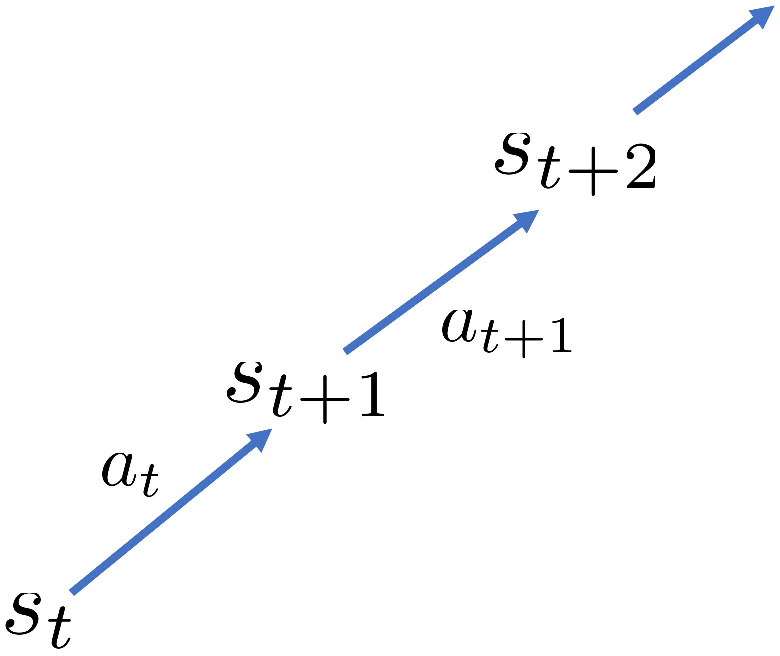


<u>Trajectory</u>

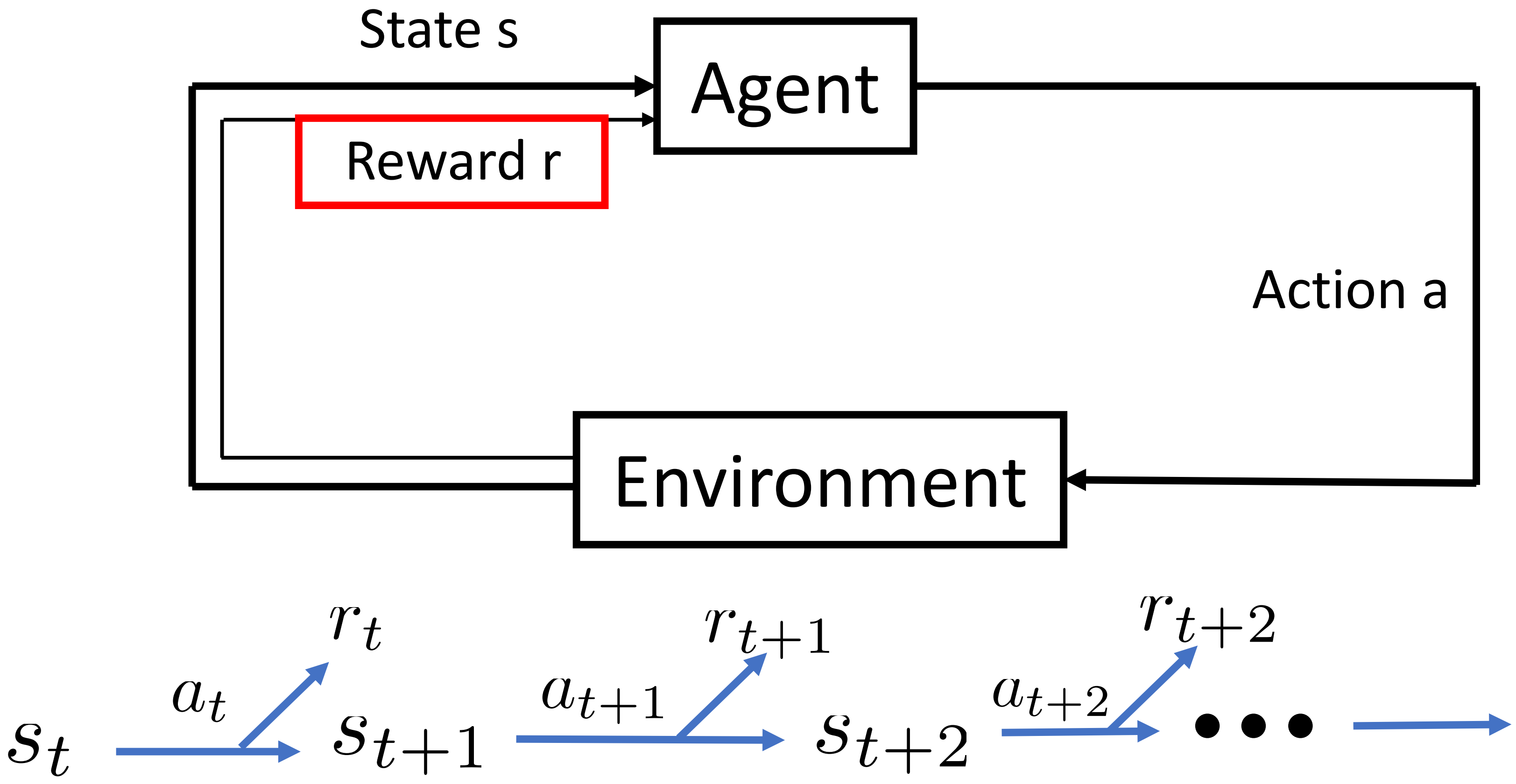


Goal of Reinforcement Learning



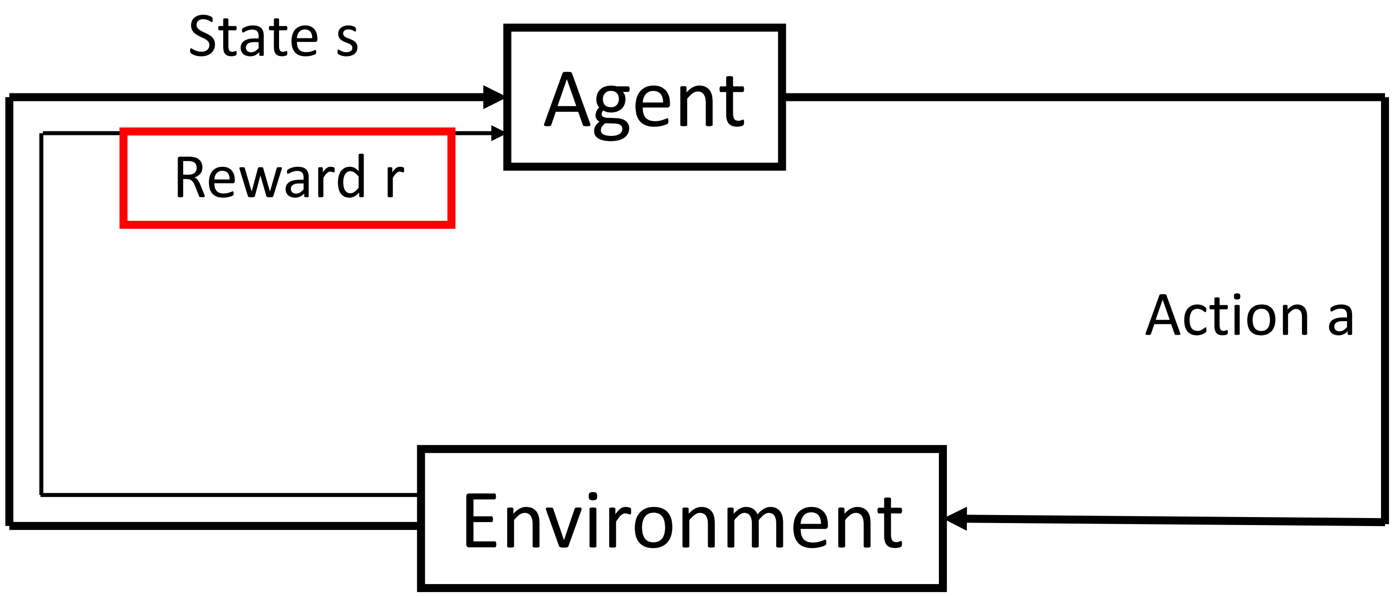


Goal of Reinforcement Learning



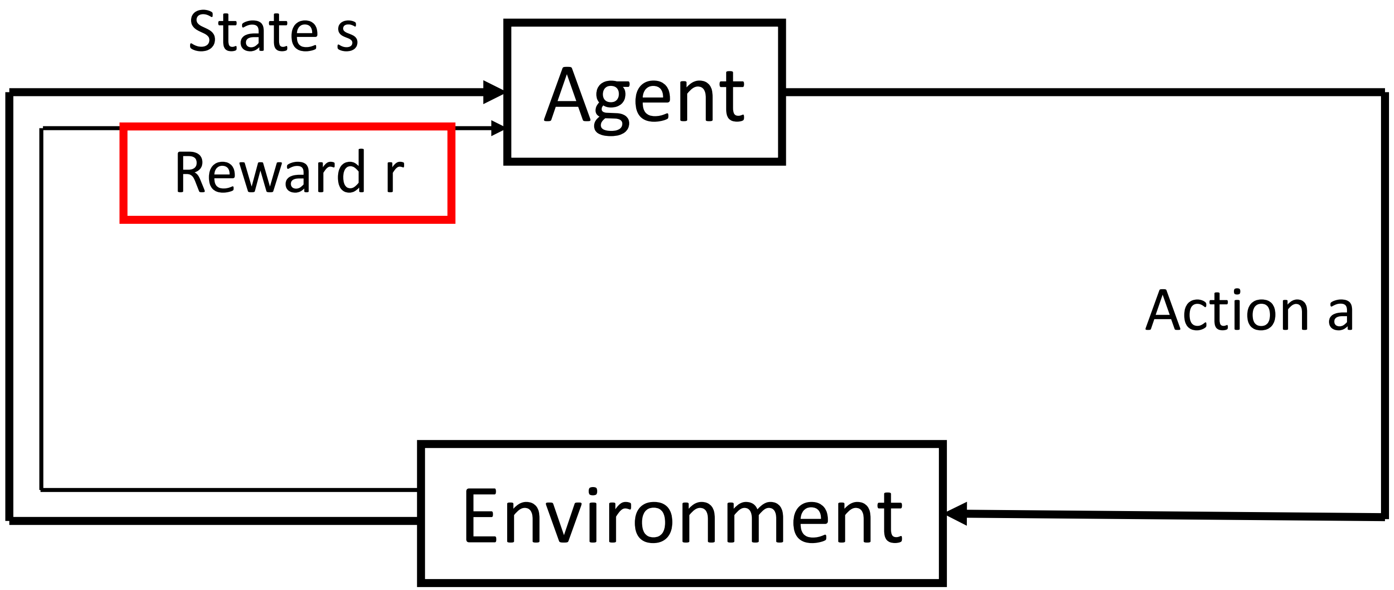
Maximize long-term reward: $\max \sum_{t'=t}^{+\infty} \gamma^{t'-t} r_t$

Key Quantities



$V(s)$ Maximal reward you can get starting from s
"Value" of state s

Key Quantities



$Q(s, a)$ Maximal reward you can get starting from s
"Q-function" of state s and action a

Bellman Equations

$$\boxed{Q^*(s,a)} = r(s,a) + \gamma \max_{a'} \boxed{Q^*}(s'(s,a),a')$$

$$\boxed{V^*(s)} = \max_{a} r(s,a) + \gamma \boxed{V^*}(s'(s,a))$$

Optimal solution

Algorithm

Iteratively table filling

Tabular Q-learning

$$Q^{(n)}(s,a) \leftarrow r(s,a) + \gamma \max_{a'} Q^{(n-1)}(s'(s,a),a')$$

Value Iteration

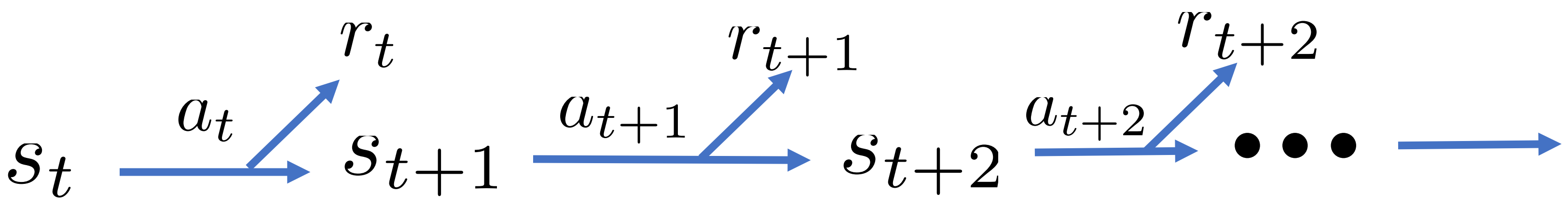
$$V^{(n)}(s) \leftarrow \max_{a} r(s,a) + \gamma V^{(n-1)}(s'(s,a))$$

As long as we can enumerate ***all possible*** states and actions

On trajectories

Q-learning

$$Q^{(n)}(s_t, a_t) \leftarrow r(s_t, a_t) + \gamma \max_{a'} Q^{(n-1)}(s_{t+1}, a')$$



On trajectories

Parametric function

Q-learning

$$Q_\theta^{(n)}(s_t, a_t) \leftarrow r(s_t, a_t) + \gamma \max_{a'} Q_\theta^{(n-1)}(s_{t+1}, a')$$

$Q_\theta(s, a)$ now have generalization capability

How could you take the gradient w.r.t θ ?

On trajectories

Q-learning

$$Q_{\theta}^{(n)}(s_t, a_t) \leftarrow r(s_t, a_t) + \gamma \max_{a'} \boxed{Q_{\boxed{\theta'}}^{(n-1)}(s_{t+1}, a')}$$

Old fixed parameters

Target network

On trajectories

Q-learning

$$Q_\theta(s_t, a_t) \leftarrow (1-\alpha) Q_\theta(s_t, a_t) + \alpha \left[r(s_t, a_t) + \gamma \max_{a'} Q_{\theta'}(s_{t+1}, a') \right]$$

[Mnih et al. Nature 2015]

Sample trajectories

Q-learning

$$Q_\theta^{(n)}(s_t, a_t) \leftarrow r(s_t, a_t) + \gamma \max_{a'} Q_\theta^{(n-1)}(s_{t+1}, a')$$

How could we sample a trajectory in the state space?

Behavior policy $\beta(\cdot|s)$

On-policy versus Off-policy approaches

Off-policy, sampled by some behavior policy $\beta(\cdot|s)$

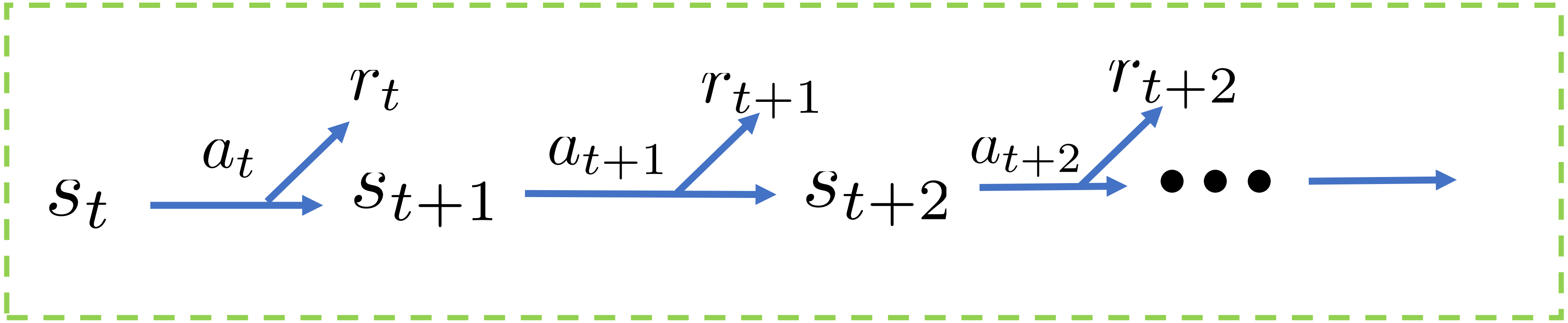
Expert behaviors (imitation learning)

Supervised learning

On-policy, sampled by the current models $Q^{(n)}(s,a)$ $\pi(\cdot|s)$

Agent not only learns from the data, but also chooses which data to learn.

Policy gradient



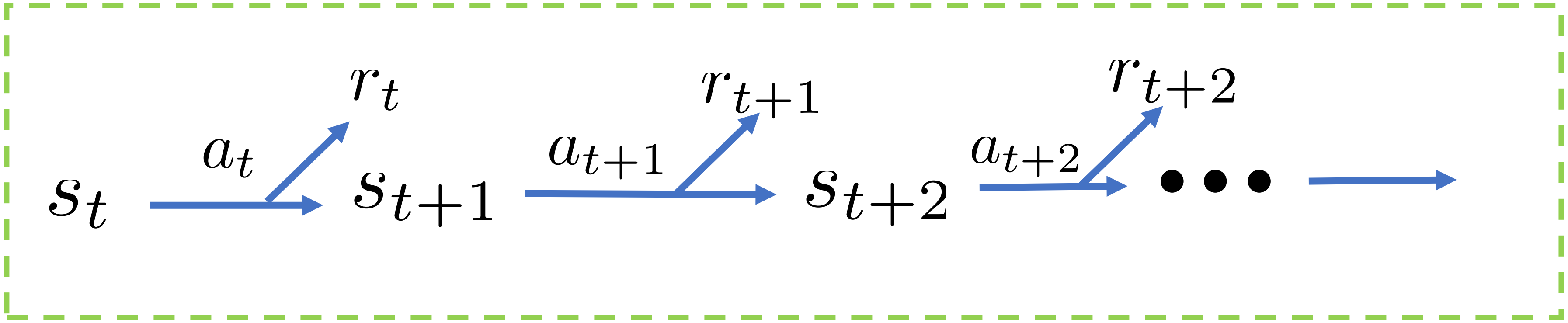
Trajectory τ

$$J(\theta) = \mathbb{E}_{\tau \sim p_\theta(\tau)}\left[r(\tau)\right]$$

$\pi_\theta(a|s)$ Probability of taking action a given state s

$r(\tau)$ Cumulative reward along a trajectory τ

Policy gradient

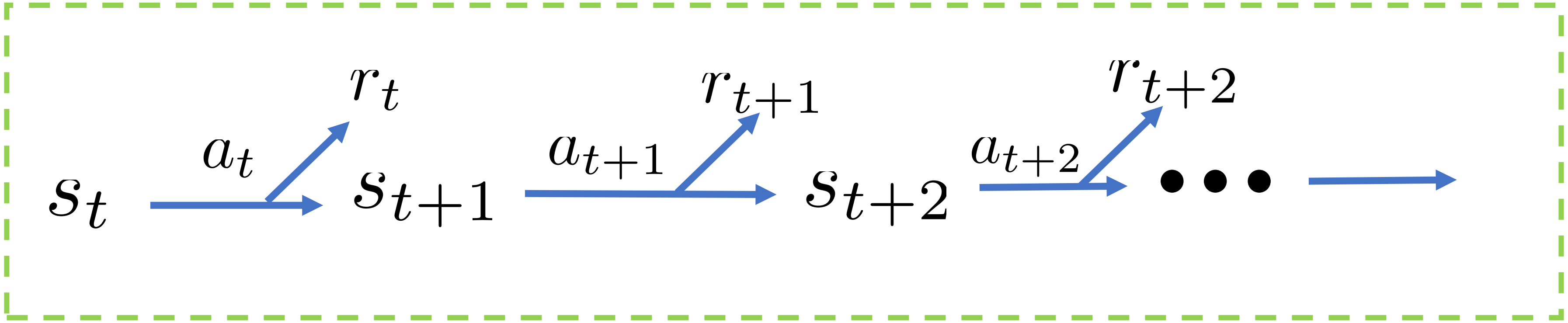


Trajectory τ

$$J(\theta) = \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[r(\tau) \right]$$

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[r(\tau) \nabla_\theta \log p_\theta(\tau) \right]$$

Policy gradient



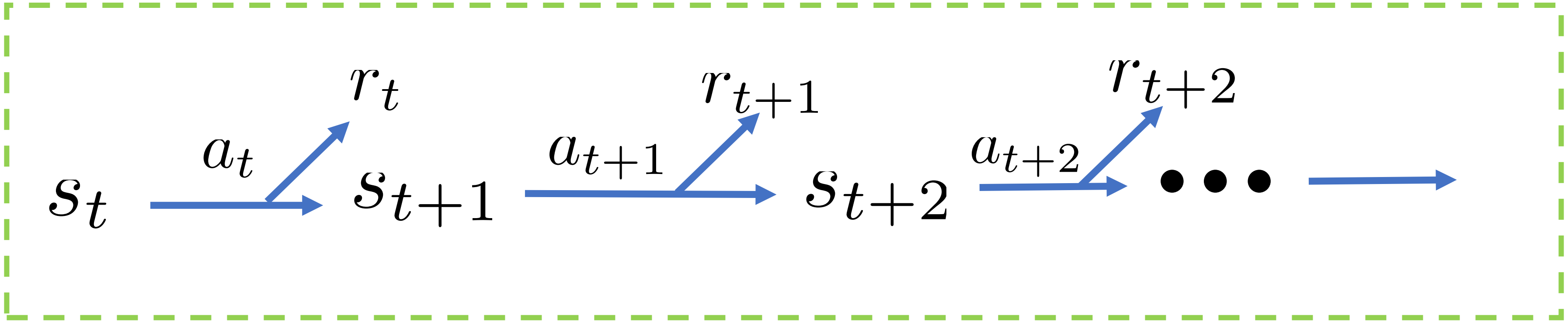
Trajectory τ

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[r(\tau) \nabla_\theta \log p_\theta(\tau) \right]$$

$$\log p_\theta(\tau) = \log p(s_1) + \sum_{t=1}^{T} \log \pi_\theta(a_t|s_t) + \sum_{t=1}^{T} \log p(s_{t+1}|s_t, a_t)$$

Independent of θ

Policy gradient



Trajectory τ

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[r(\tau) \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) \right]$$

Estimated by sampling $\pi_\theta(a|s)$

Baseline

$$\mathbb{E}_{\tau \sim p_\theta(\tau)} \left[\nabla_\theta \log p_\theta(\tau) \right] \equiv 0$$

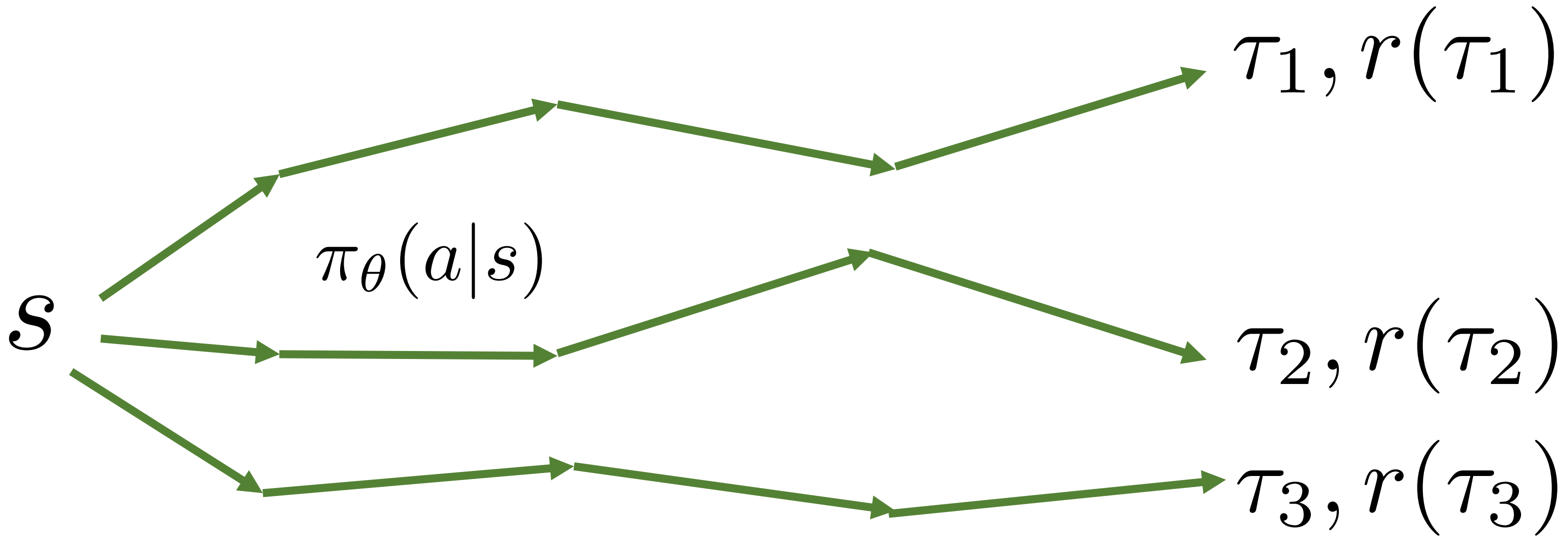
$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[r(\tau) \nabla_\theta \log p_\theta(\tau) \right]$$

$$\nabla_\theta J(\theta) = \mathbb{E}_{\tau \sim p_\theta(\tau)} \left[(r(\tau) - b) \nabla_\theta \log p_\theta(\tau) \right]$$

Can be any function that only depends on state.

REINFORCE

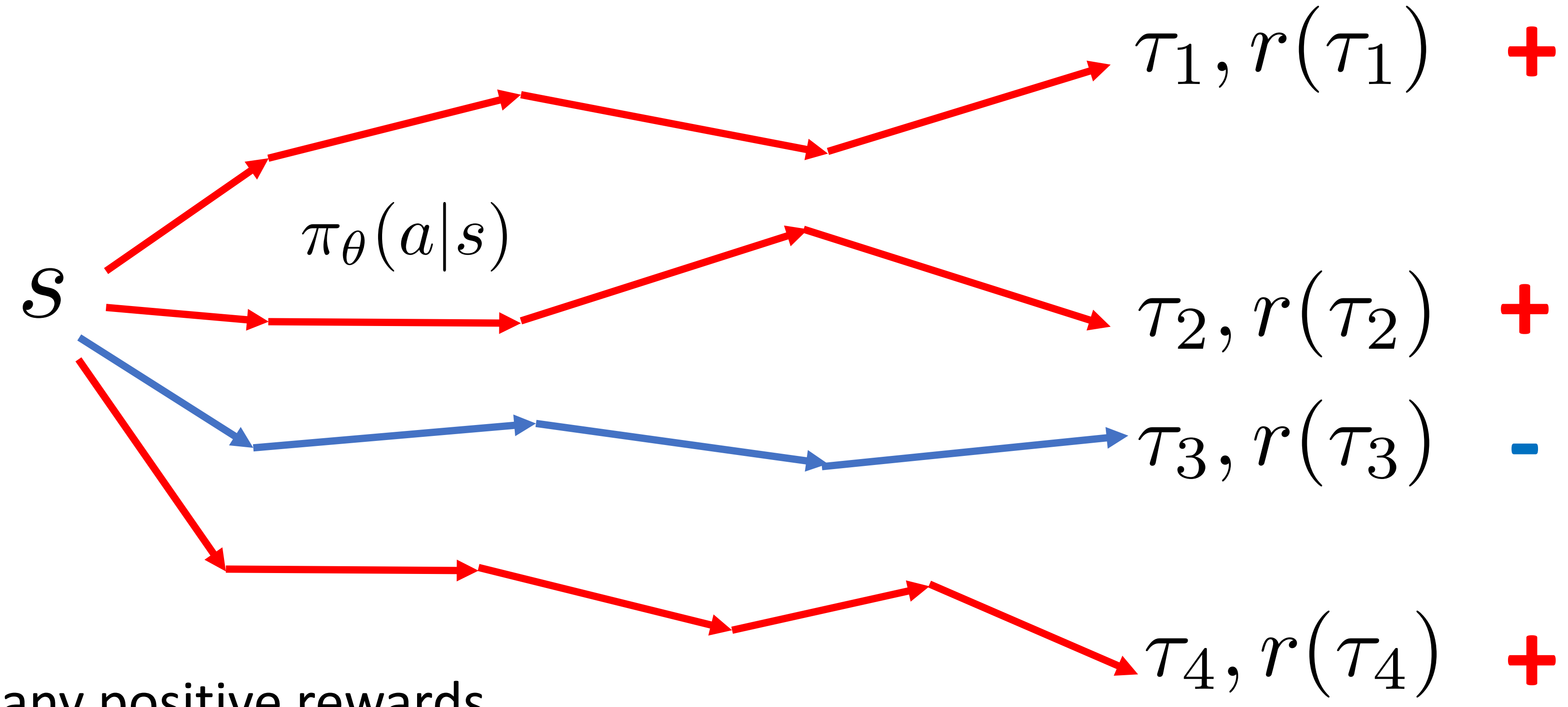
$r(\tau)$ Actual reward obtained by rolling out from $\boldsymbol{s}$



$$\nabla_\theta J(\theta) \approx \frac{1}{N} \sum_{i=1}^{N} \left[r_i(\tau) \sum_{t=1}^{T} \nabla_\theta \log \pi_\theta(a_t^i | s_t^i) \right]$$

REINFORCE

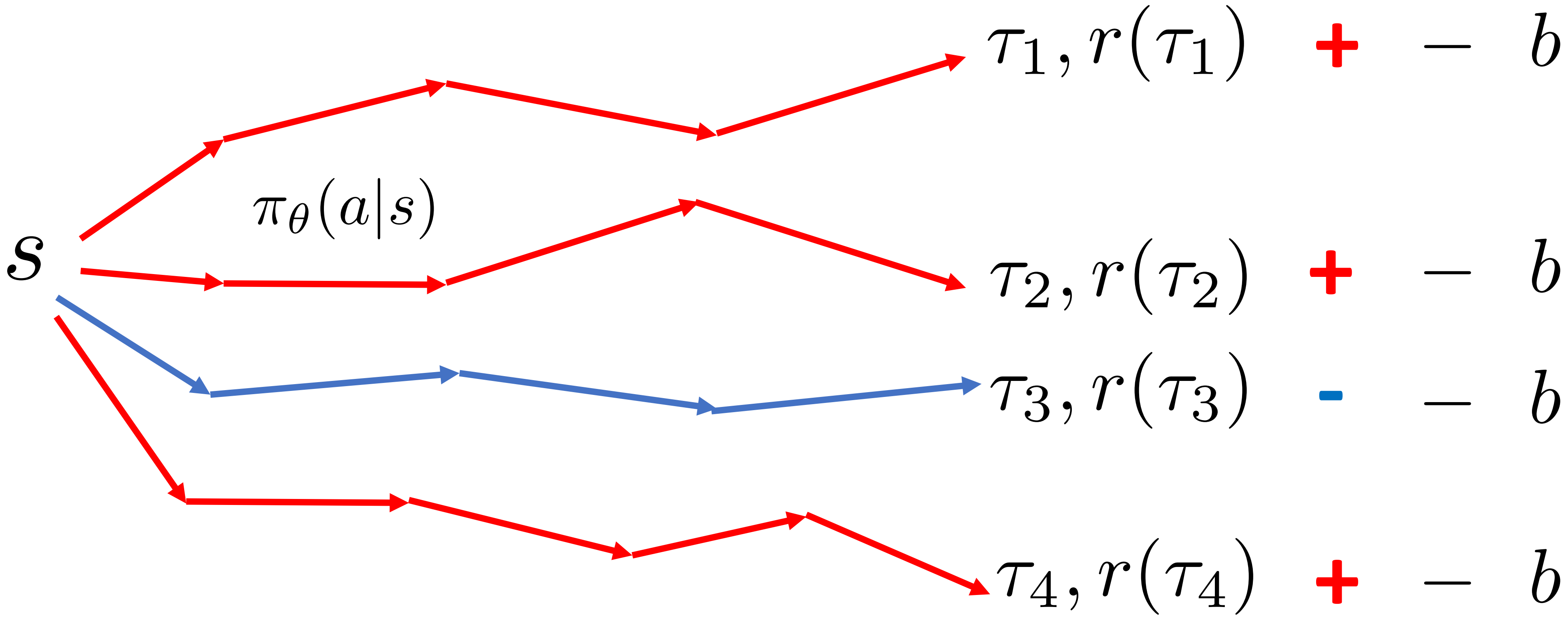
$r(\tau)$ Actual reward obtained by rolling out from s



Too many positive rewards.
We only want to pick the best of the best.

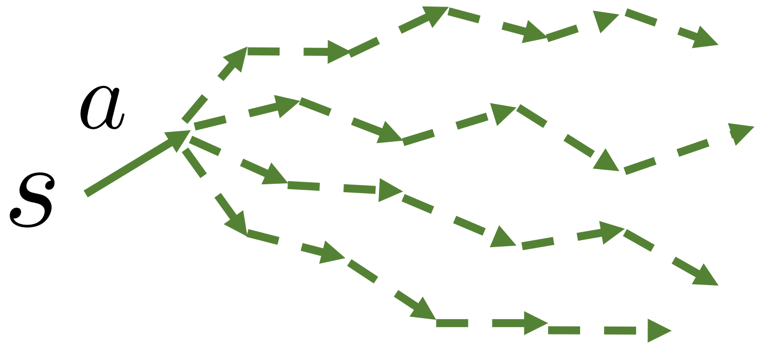
REINFORCE

$r(\tau)$ Actual reward obtained by rolling out from s



b mean of all the rewards

Actor-Critic Models



$$r(\tau) \approx Q^{\pi}_{\theta}(s, a)$$ Rollout return as a parametric function (critic)

Estimated from TD difference in the data

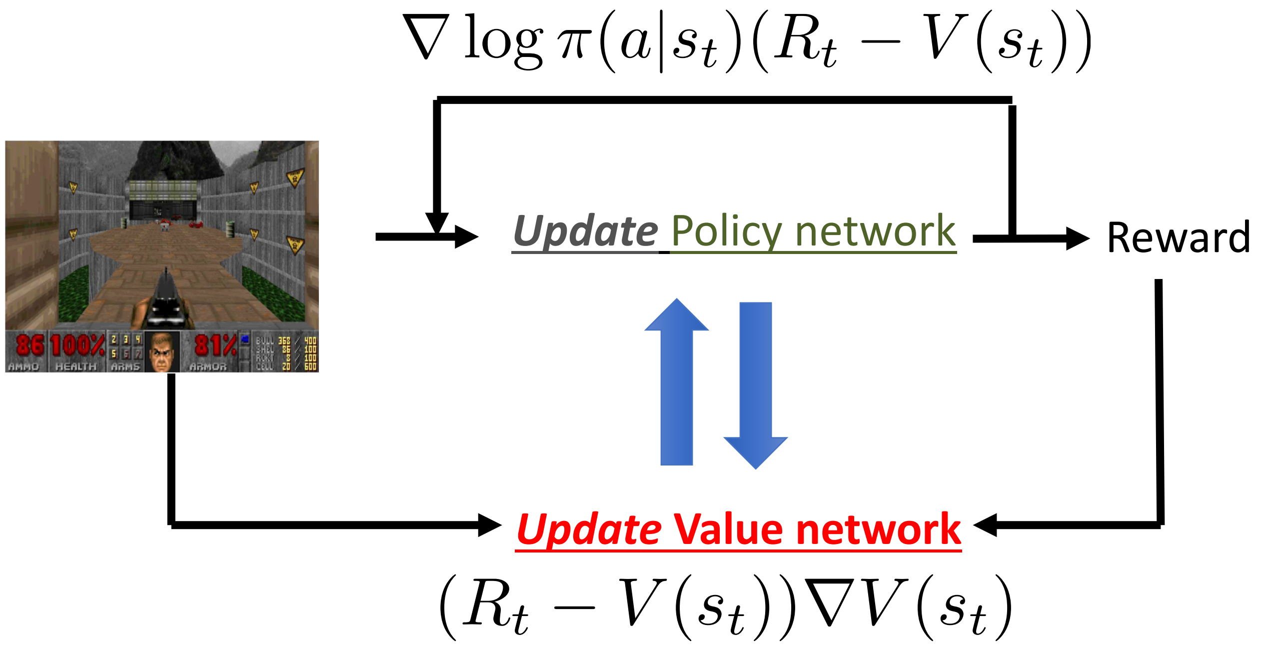
$$b(s) = V_{\theta}(s)$$ Use the value function as the baseline

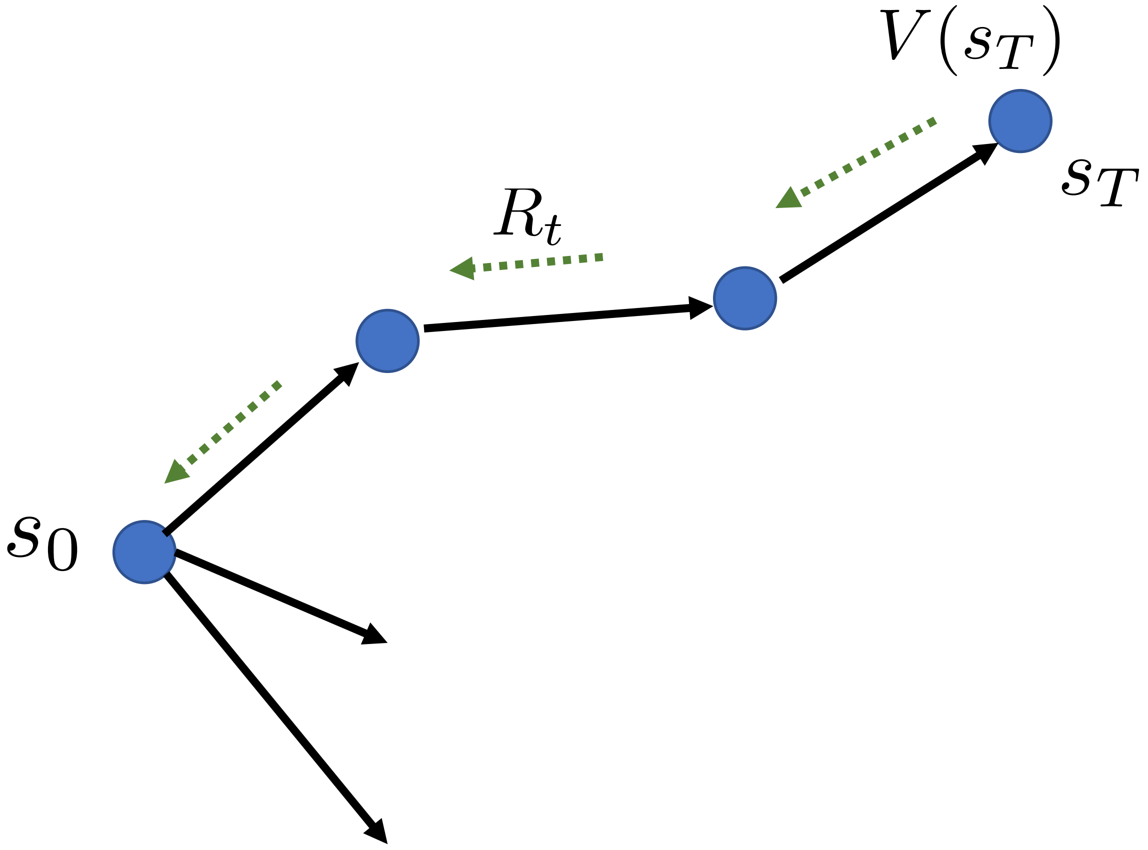
$$r(\tau) - b(s) \approx Q^{\pi}_{\theta}(s, a) - V_{\theta}(s) = \boxed{A^{\pi}_{\theta}(s, a)}$$

Advantage function

“Advantageous Actor-Critic”

A2C / A3C





Encourage actions leading to states with high-than-expected value.
Encourage value function to converge to the true cumulative rewards.
Keep the diversity of actions

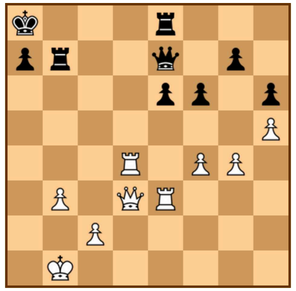
Part III: Algorithm used in Games

How Game AI works

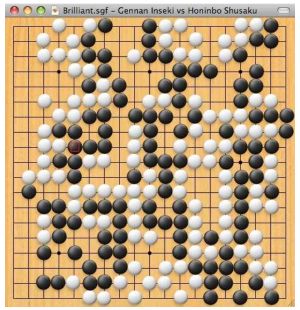
Even with a super-super computer,
it is not possible to search the entire space.

How Game AI works

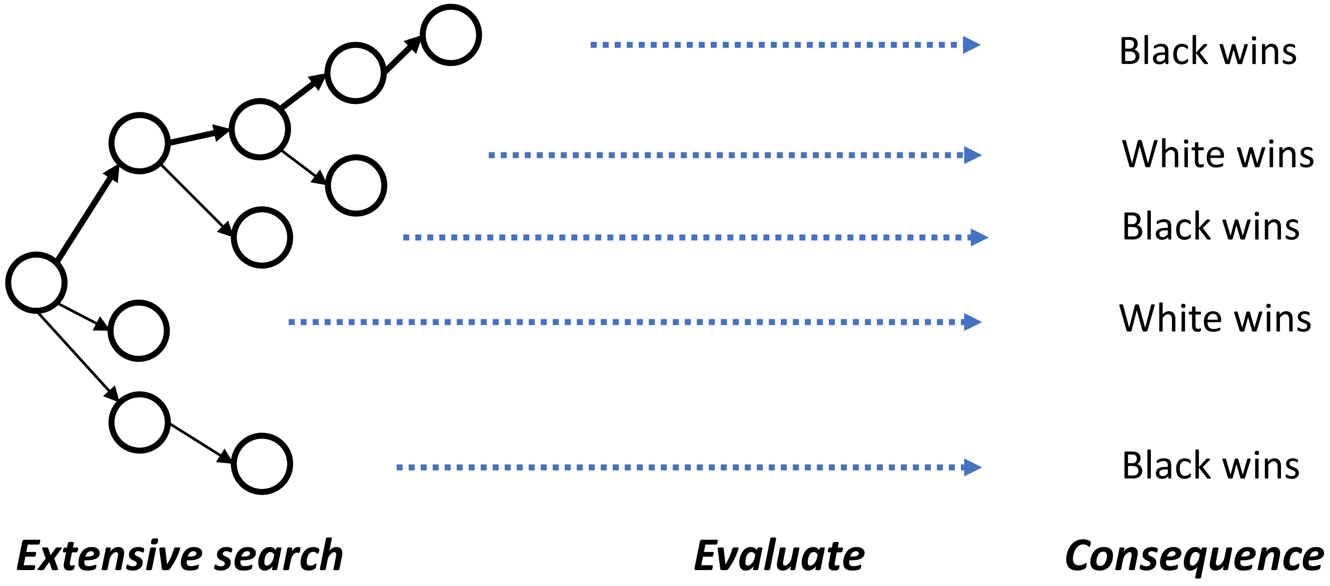
Even with a super-super computer,
it is not possible to search the entire space.



Lufei Ruan vs. Yifan Hou (2010)



Current game situation

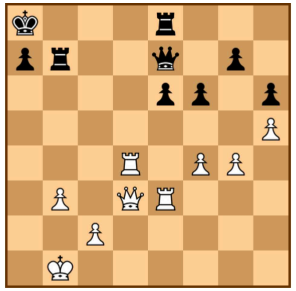


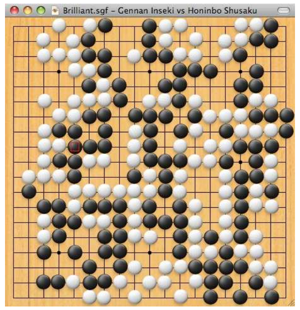
How Game AI works

How many action do you have per step?

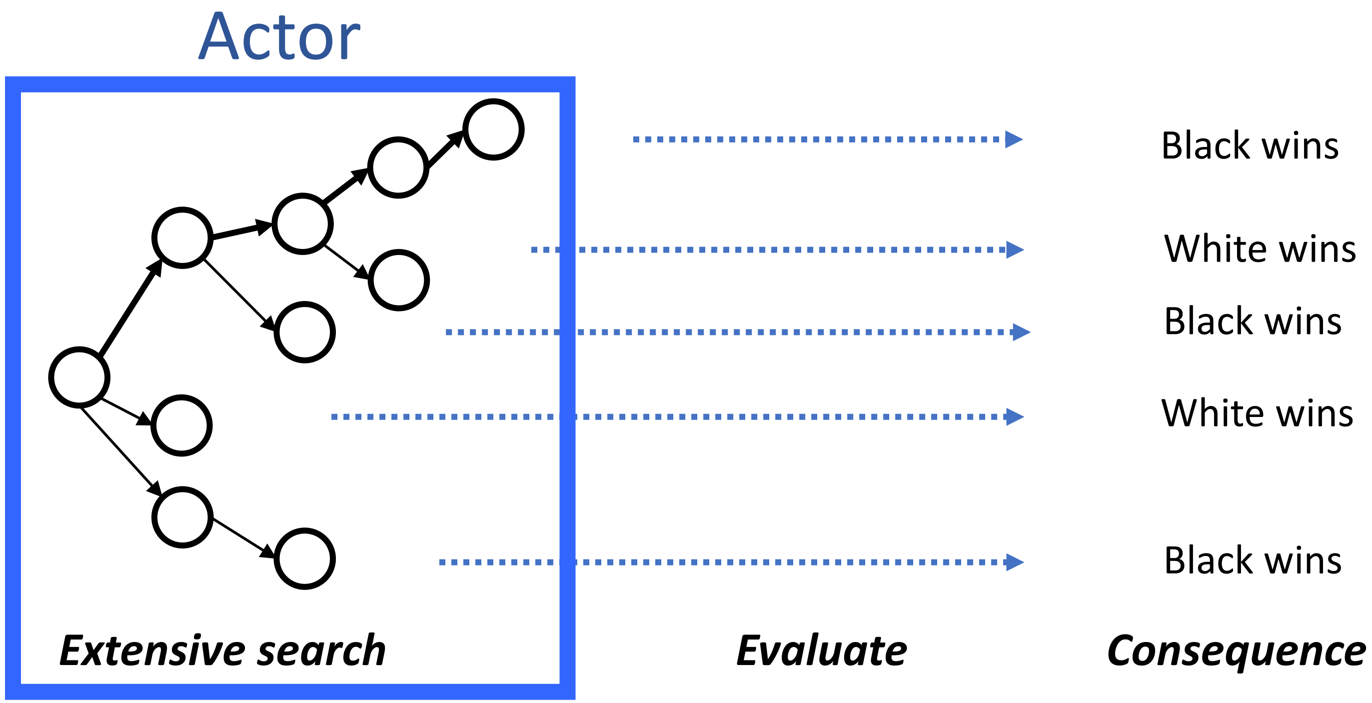
Checker: a few possible moves
Poker: a few possible moves
Chess: 30-40 possible moves
Go: 100-200 possible moves
StarCraft: 50^100 possible moves

Alpha-beta pruning + Iterative deepening [Major Chess engine]
Counterfactual Regret Minimization [Libratus, DeepStack]
Monte-Carlo Tree Search + UCB exploration [Major Go engine]
???





Current game situation

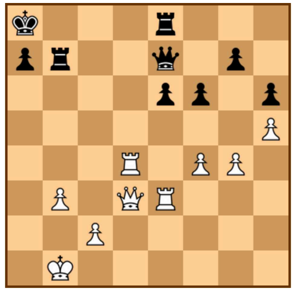


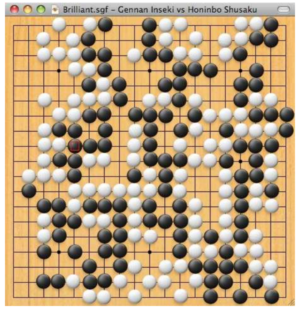
How Game AI works

How complicated is the game situation? How deep is the game?

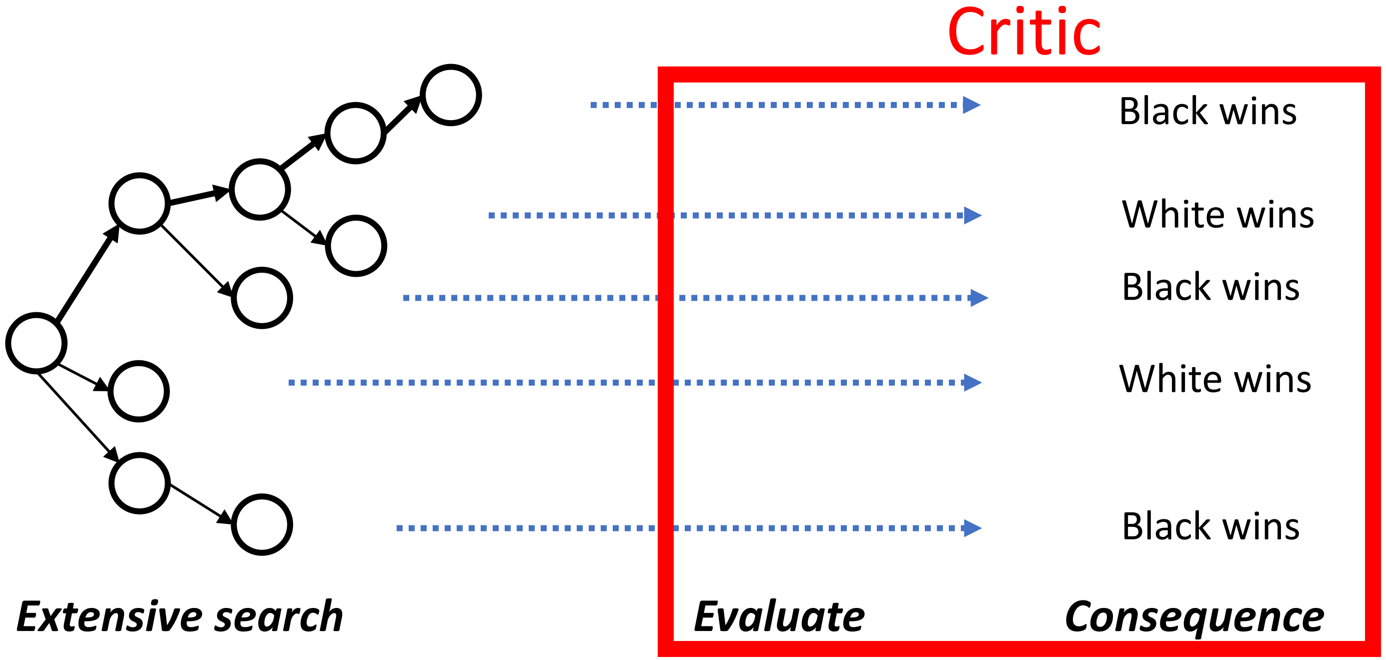
Chess
Go
Poker
StarCraft

Rule-based
Linear function for situation evaluation [Stockfish]
End game database
Random game play with simple rules [Zen, CrazyStone, DarkForest]
Deep Value network [AlphaGo, DeepStack]





Current game situation



How to model Policy/Value function?

Non-smooth + high-dimensional
Sensitive to situations. One stone changes in Go leads to different game.

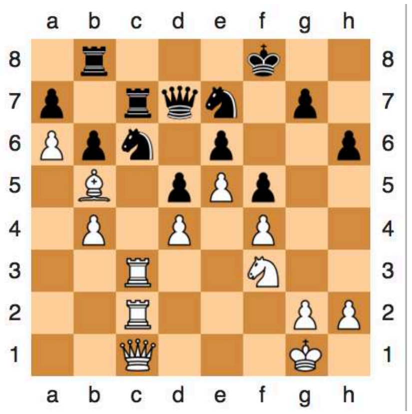
Traditional approach

- Many manual steps
- Conflicting parameters, not scalable.
- Need strong domain knowledge.

Deep Learning

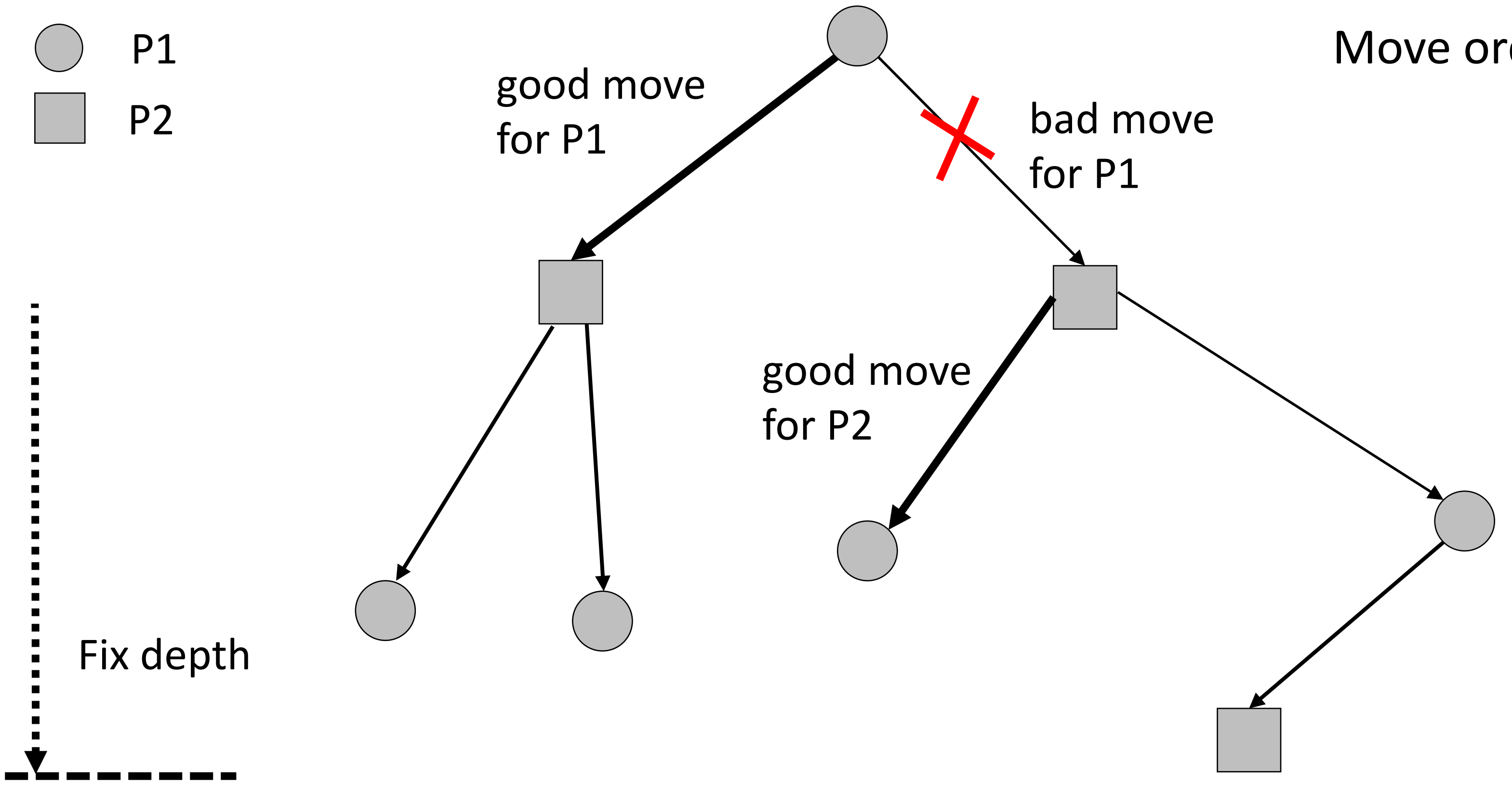
- End-to-End training
    - Lots of data, less tuning.
- Minimal domain knowledge.
- Amazing performance

Alpha-beta Pruning

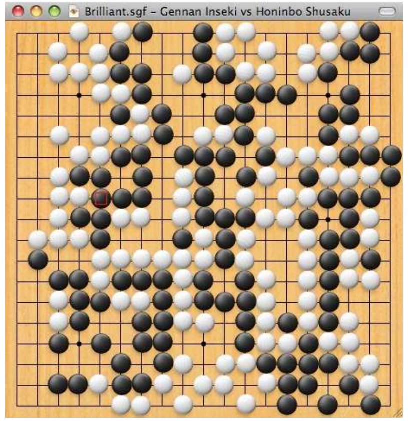


A good counter move eliminates other choices.

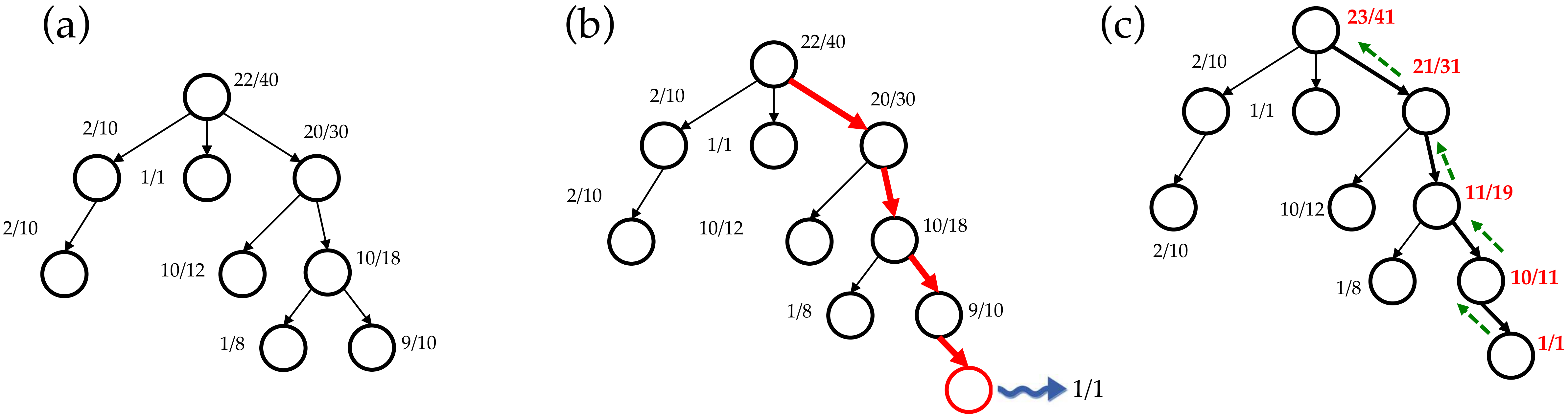
Move order is important!



Monte Carlo Tree Search



Aggregate win rates, and search towards the good nodes.

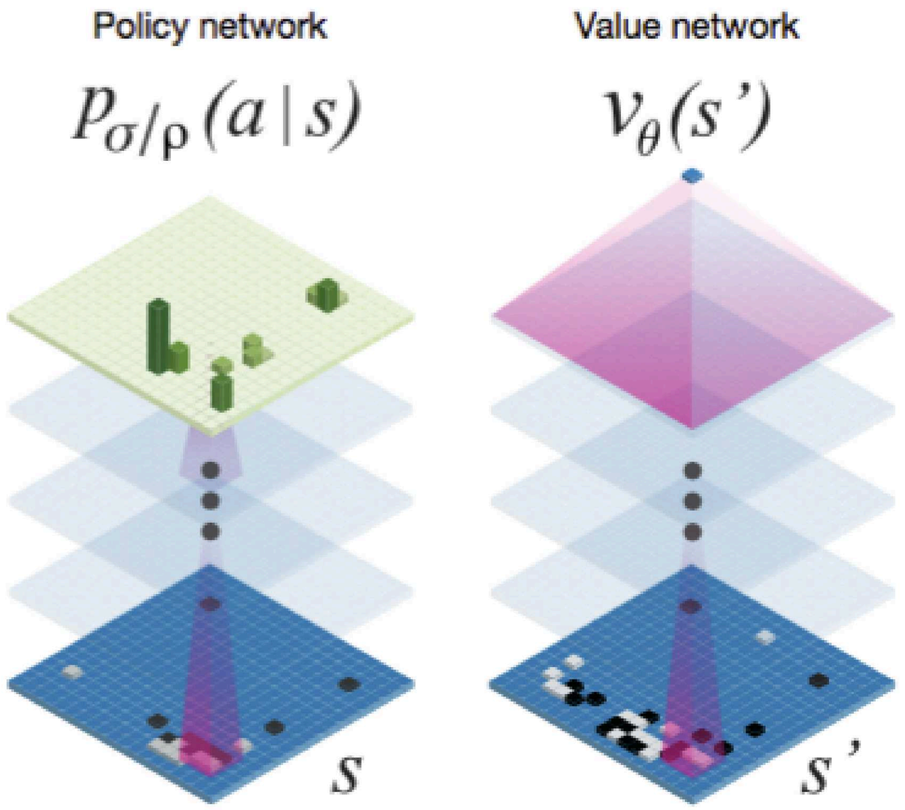


Tree policy
Default policy

$$a_t = \underset{a}{\operatorname{argmax}}\left(Q(s_t, a) + u(s_t, a)\right) \qquad u(s,a) \propto \frac{P(s,a)}{1 + N(s,a)}$$ PUCT

Part IV: Case Study

AlphaGo Fan



- Computations
  - Train with many GPUs and inference with TPU.
- Policy network
  - Trained supervised from human replays.
  - Self-play network with RL.
- High quality playout/rollout policy
  - 2 microsecond per move, ~~24.2% accuracy~~. ~30%
  - Thousands of times faster than DCNN prediction.
- Value network
  - Predicts game consequence for current situation.
  - Trained on 30M self-play games.

"Mastering the game of Go with deep neural networks and tree search", Silver et al, Nature 2016

AlphaGo Fan

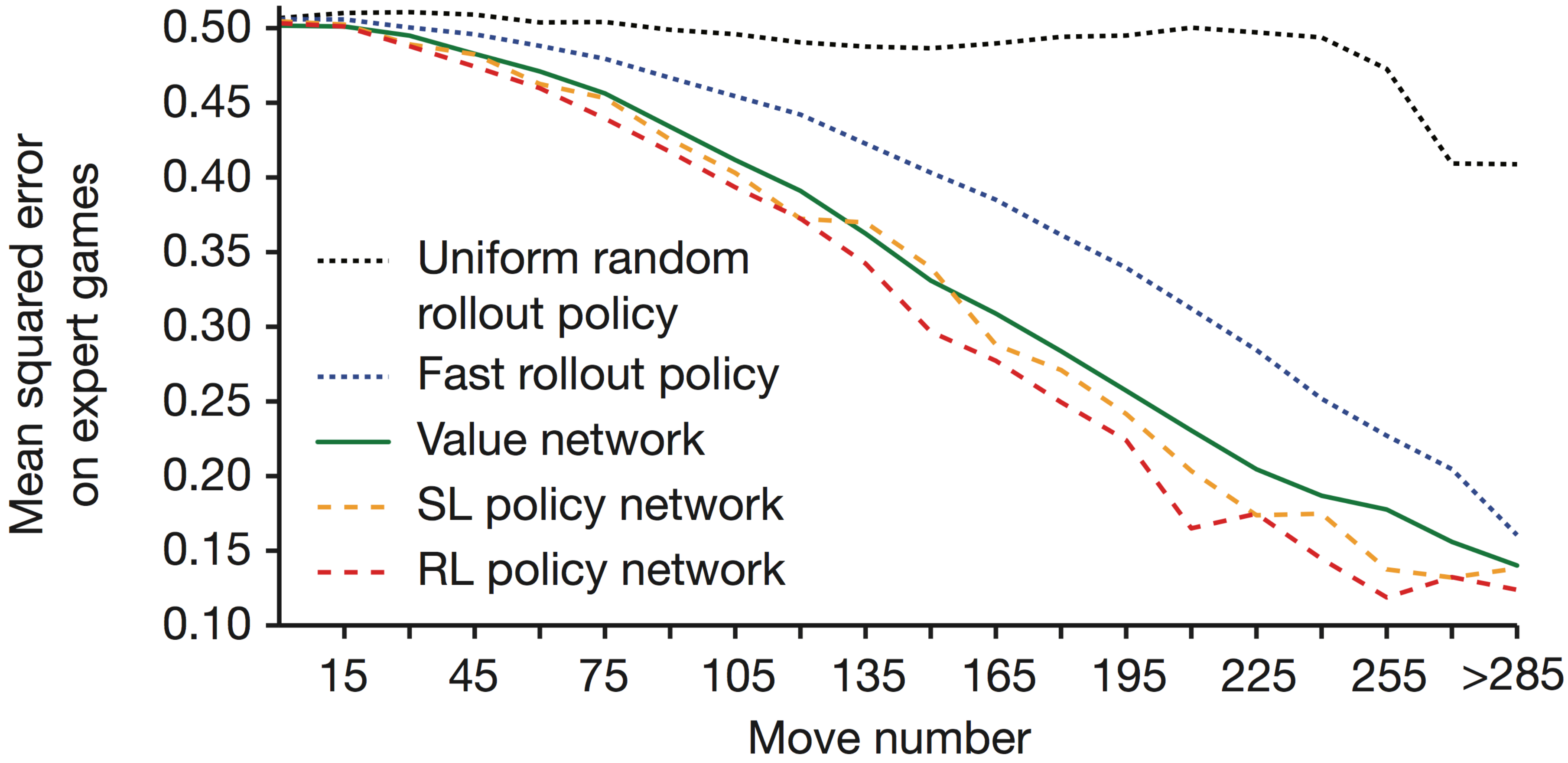
- Policy network SL (trained with human games)

Architecture			Evaluation				
Filters	Symmetries	Features	Test accuracy %	Train accuracy %	Raw net wins %	*AlphaGo* wins %	Forward time (ms)
128	1	48	54.6	57.0	36	53	2.8
192	1	48	55.4	58.0	50	50	4.8
256	1	48	55.9	59.1	67	55	7.1
256	2	48	56.5	59.8	67	38	13.9
256	4	48	56.9	60.2	69	14	27.6
256	8	48	57.0	60.4	69	5	55.3
192	1	4	47.6	51.4	25	15	4.8
192	1	12	54.7	57.1	30	34	4.8
192	1	20	54.7	57.2	38	40	4.8
192	8	4	49.2	53.2	24	2	36.8
192	8	12	55.7	58.3	32	3	36.8
192	8	20	55.8	58.4	42	3	36.8

"Mastering the game of Go with deep neural networks and tree search", Silver et al, Nature 2016

AlphaGo Fan

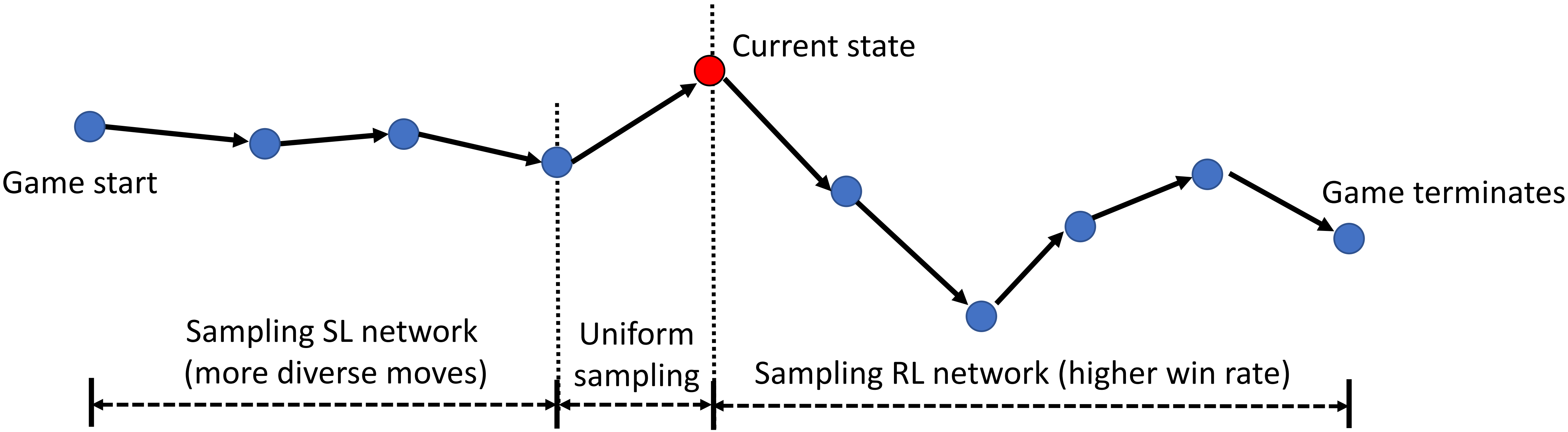
- Fast Rollout (2 microsecond), ~30% accuracy



"Mastering the game of Go with deep neural networks and tree search", Silver et al, Nature 2016

AlphaGo Fan

- Value Network (trained via 30M self-played games)
- How data are collected?



"Mastering the game of Go with deep neural networks and tree search", Silver et al, Nature 2016

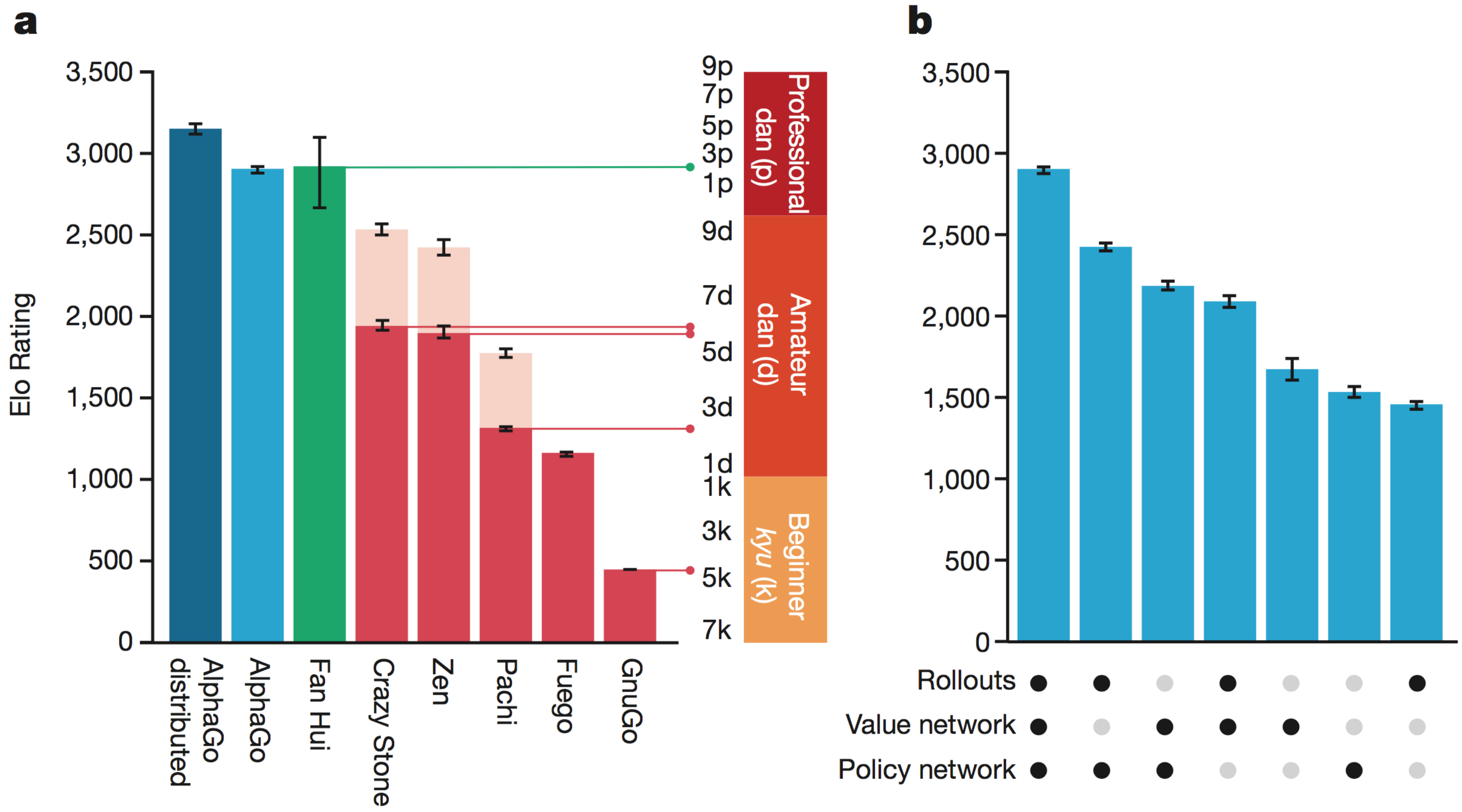
AlphaGo Fan

- Value Network (trained via 30M self-played games)

Short name	Policy network	Value network	Rollouts	Mixing constant	Policy GPUs	Value GPUs	Elo rating
α_{rvp}	p_σ	v_θ	p_π	$\lambda = 0.5$	2	6	2890
α_{vp}	p_σ	v_θ	–	$\lambda = 0$	2	6	2177
α_{rp}	p_σ	–	p_π	$\lambda = 1$	8	0	2416
α_{rv}	$[p_\tau]$	v_θ	p_π	$\lambda = 0.5$	0	8	2077
α_v	$[p_\tau]$	v_θ	–	$\lambda = 0$	0	8	1655
α_r	$[p_\tau]$	–	p_π	$\lambda = 1$	0	0	1457
α_p	p_σ	–	–	–	0	0	1517

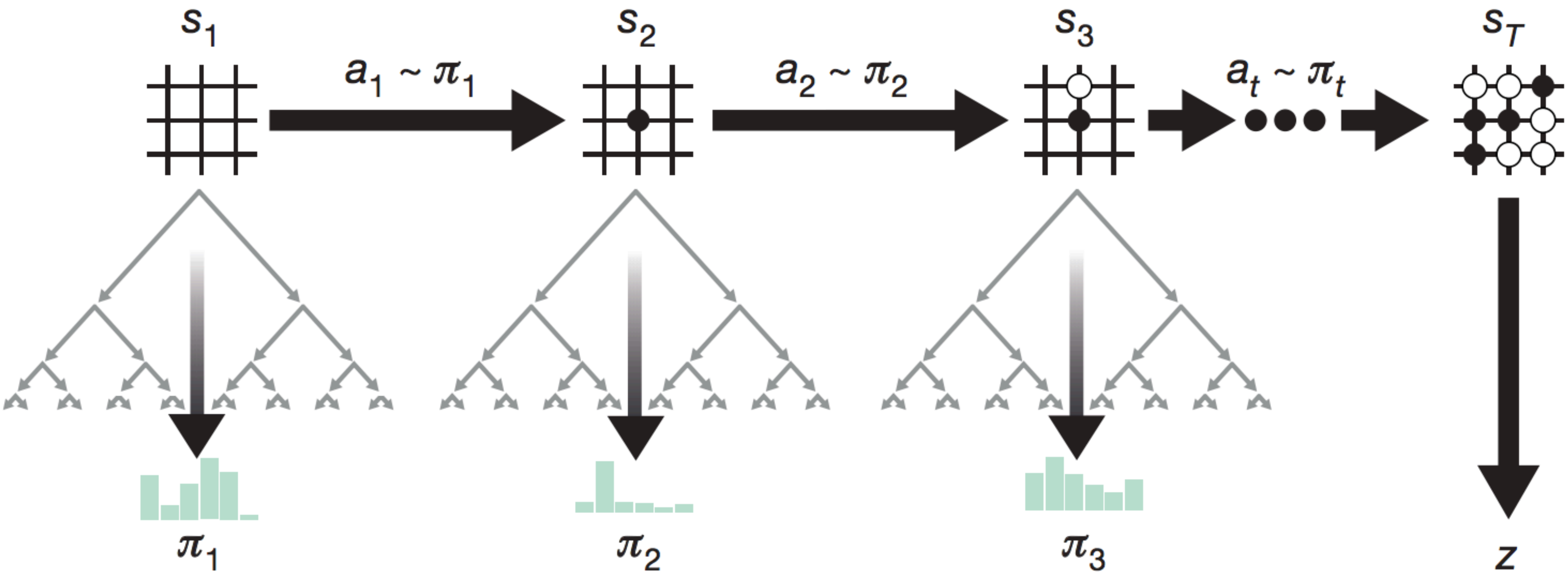
"Mastering the game of Go with deep neural networks and tree search", Silver et al, Nature 2016

AlphaGo Fan

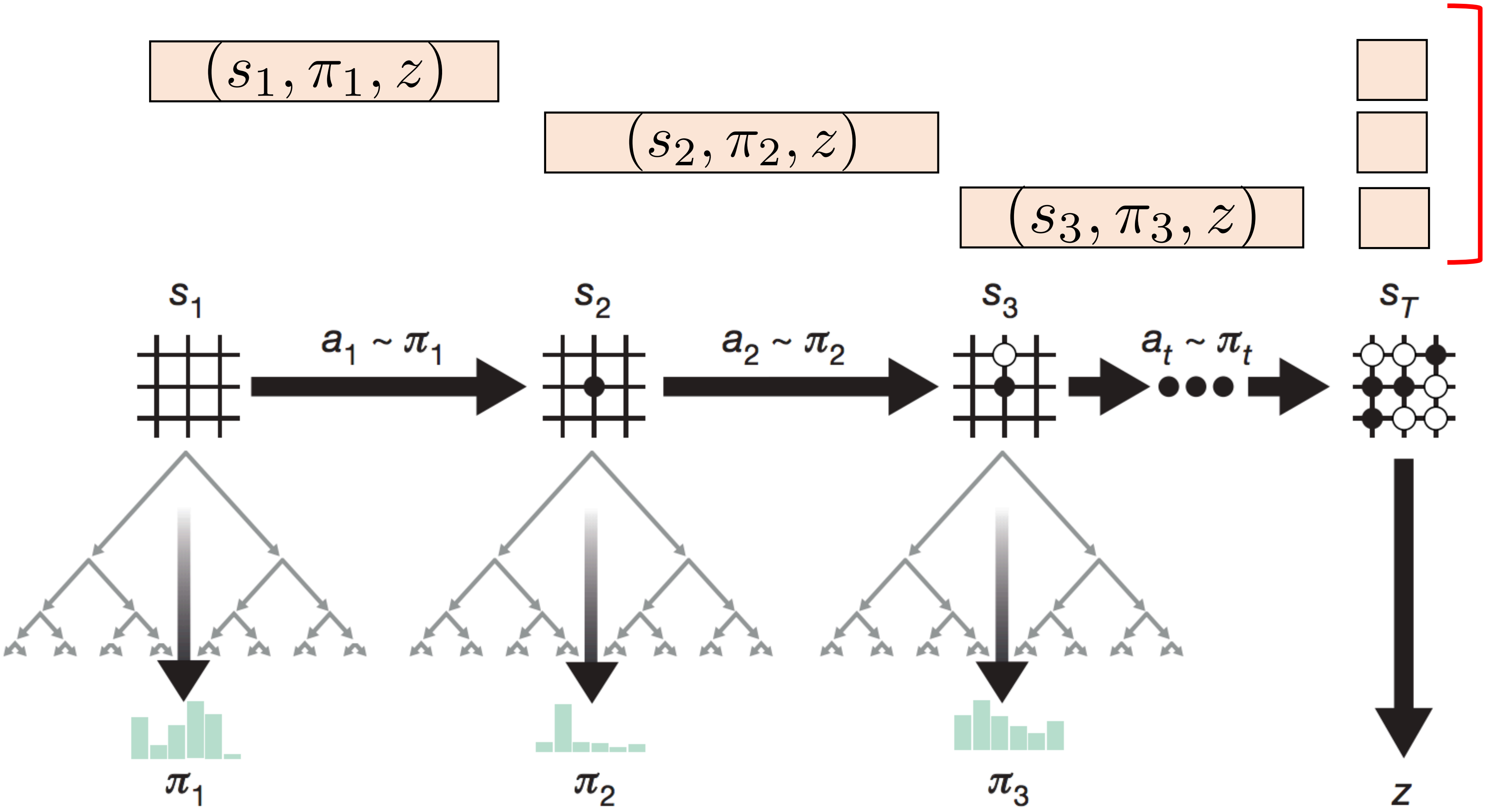


"Mastering the game of Go with deep neural networks and tree search", Silver et al, Nature 2016

AlphaGo Zero



AlphaGo Zero



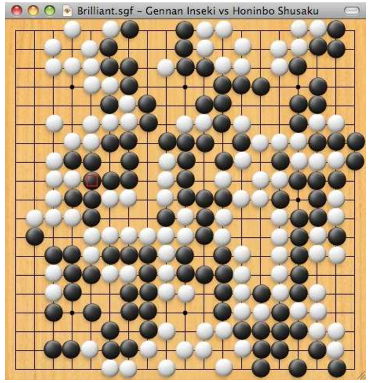
Training samples for θ_{i+1}

MCTS θ_i

AlphaGo Zero

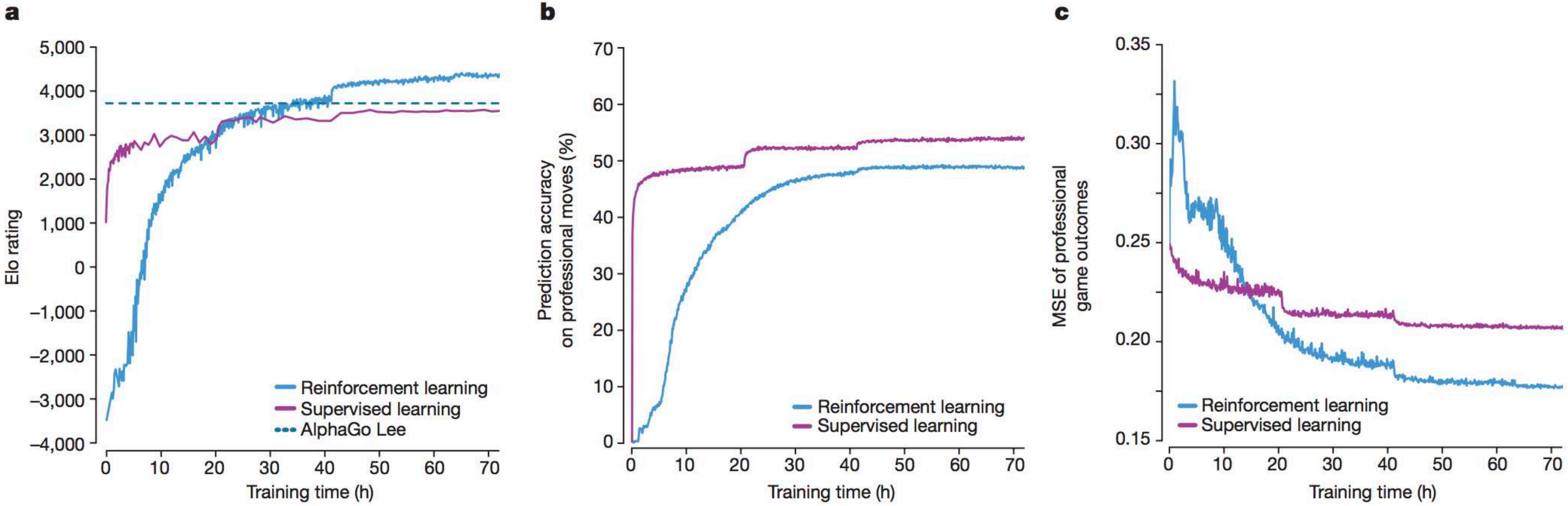
$$(s, \boldsymbol{\pi}, z)$$

$$J(\theta) = (z - V_\theta)^2 - \boldsymbol{\pi}^T \log \mathbf{p}_\theta + c\|\theta\|^2$$

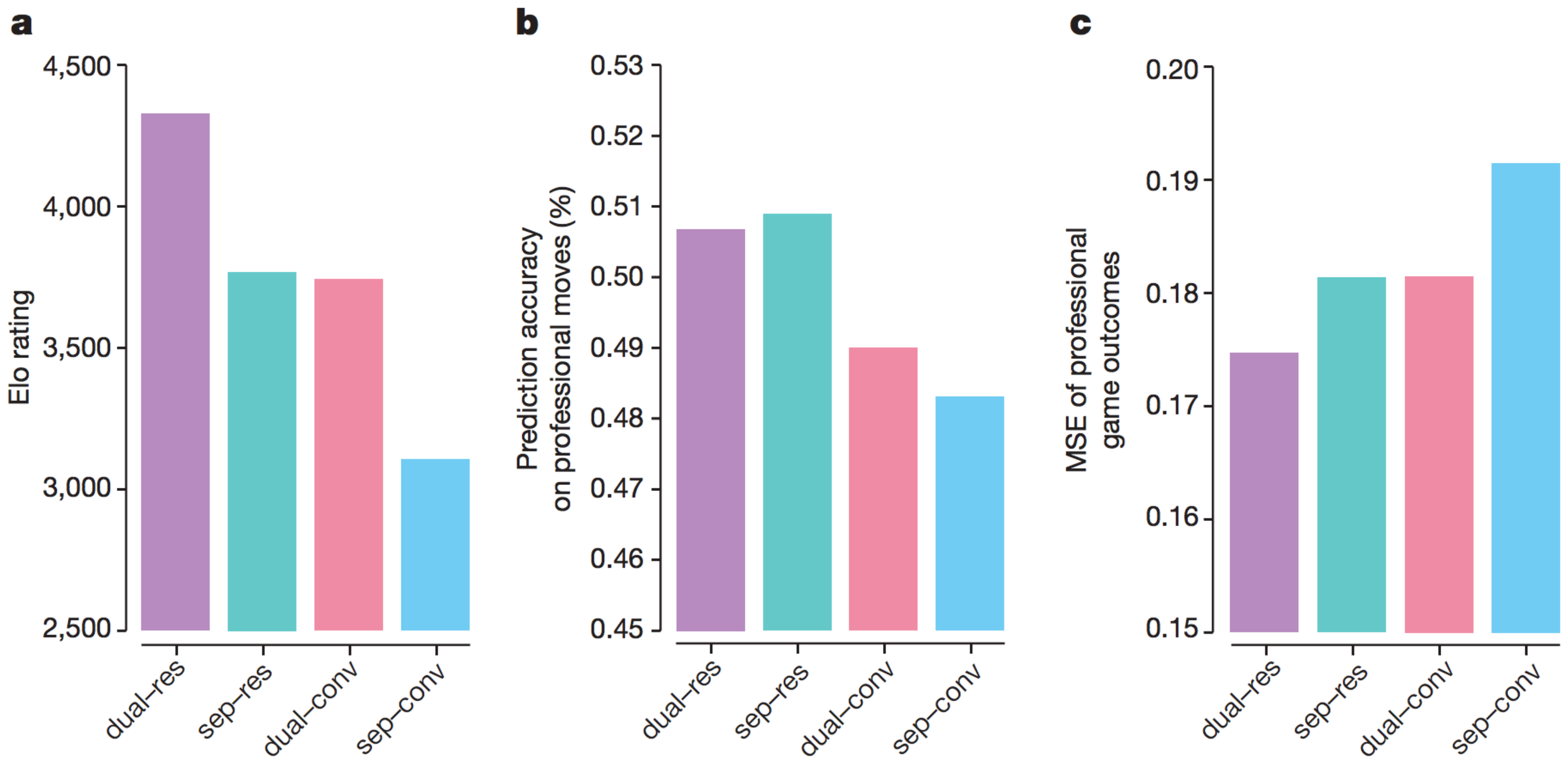


$\boldsymbol{s}$ → ResNet → $\mathbf{p}_\theta(s)$, $V_\theta(s)$

AlphaGo Zero



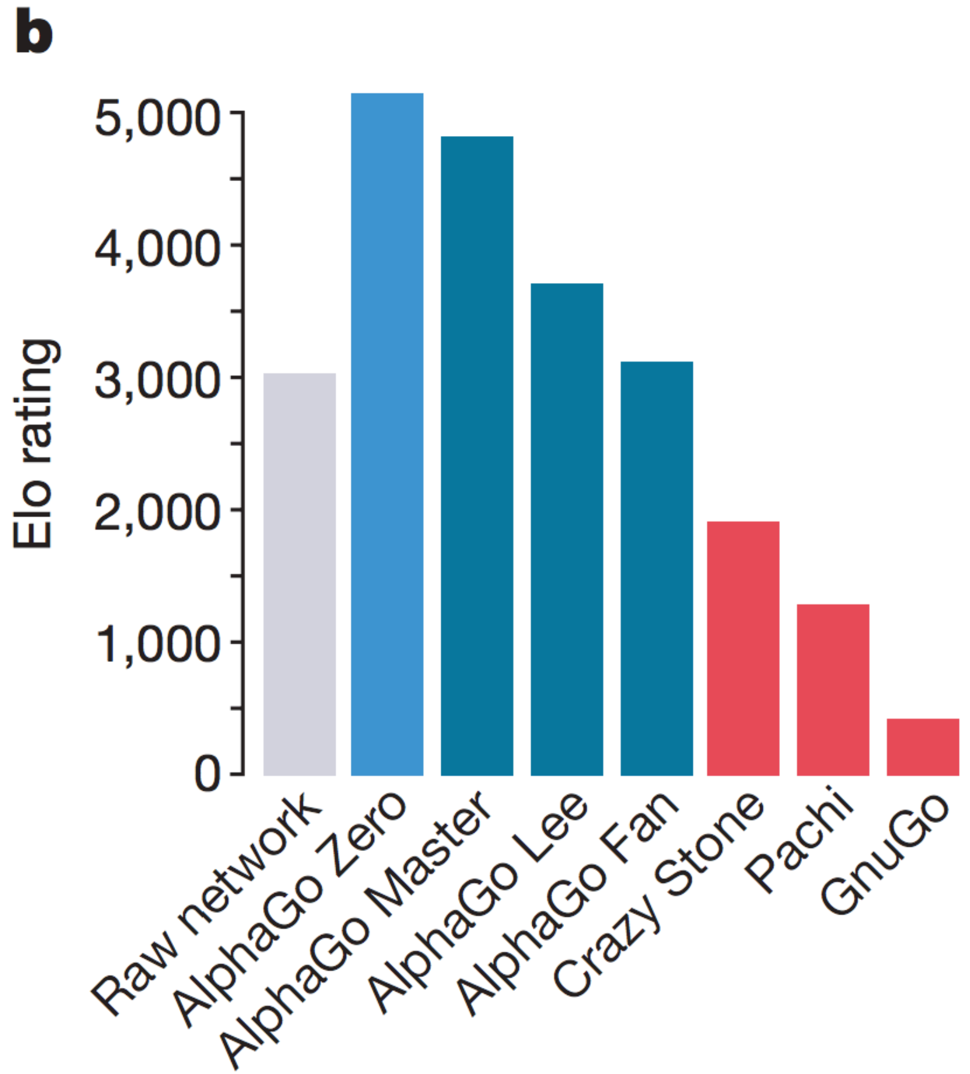
Using ResNet and shared network



AlphaGo Zero Strength

- 3 days version
  - 4.9M Games, 1600 rollouts/move
  - 20 block ResNet
  - Defeat AlphaGo Lee by 100:0.

- 40 days version
  - 29M Games, 1600 rollouts/move
  - 40 blocks ResNet.
  - Defeat AlphaGo Master by 89:11



Computation Time

Game Playing: 4 TPUs

Supervised network training

64 GPU (32 batchsize/GPU)

0.7 million mini-batch of size 2048 (370ms per batch)

Training data generation

Using TPU, single rollout 0.25ms

4.9 M Games * 1600 rollouts/move (0.4s) * (~250 move/game)

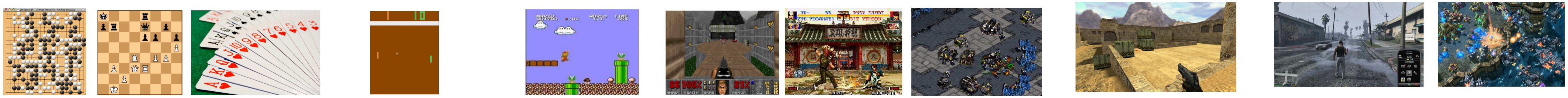
= 15.5 years

15.5 years / 3 days = 1890 machines

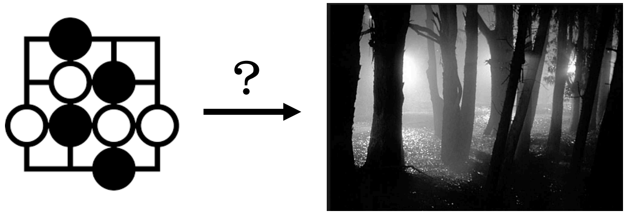
4.7k game situations/sec

18.9 games / sec

Game as a Vehicle of AI

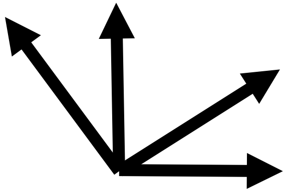


Algorithm is slow and data-inefficient



Abstract game to real-world

Require a lot of resources.



Hard to benchmark the progress

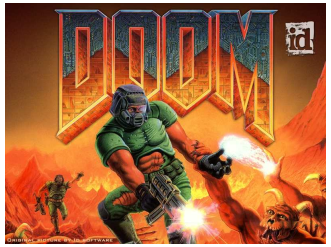
Better Algorithm/System

Better Environment

Our work

Better Algorithm/System

DarkForest Go Engine
(Yuandong Tian, Yan Zhu, ICLR16)



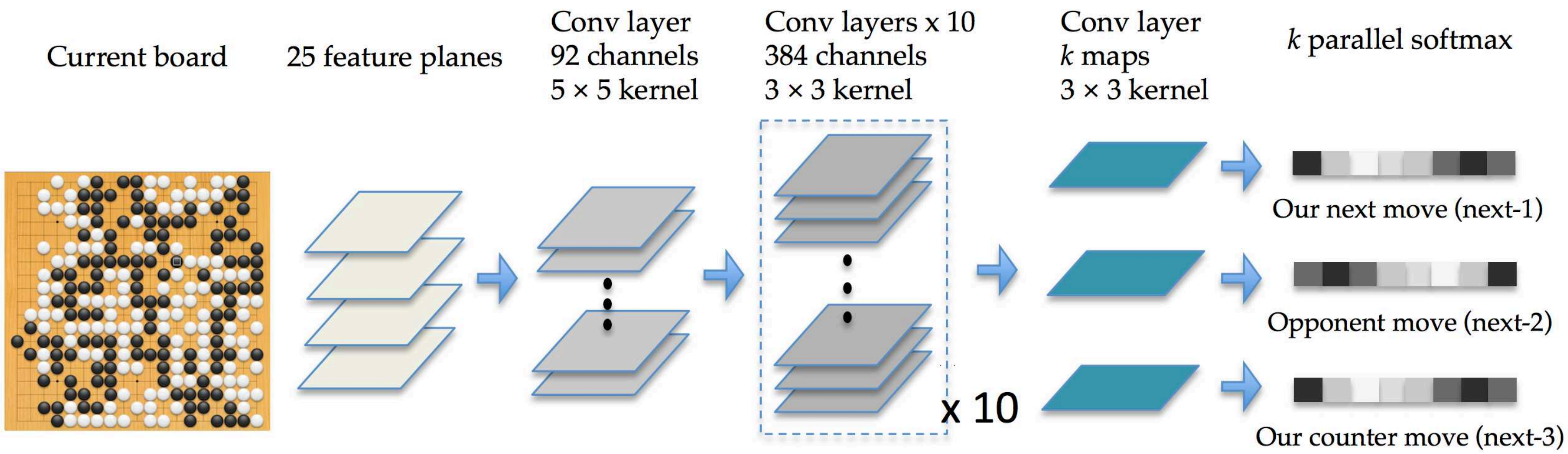
Doom AI
(Yuxin Wu, Yuandong Tian, ICLR17)

Better Environment

ELF: Extensive Lightweight and Flexible Framework
(Yuandong Tian et al, arXiv)

Our computer Go player： DarkForest

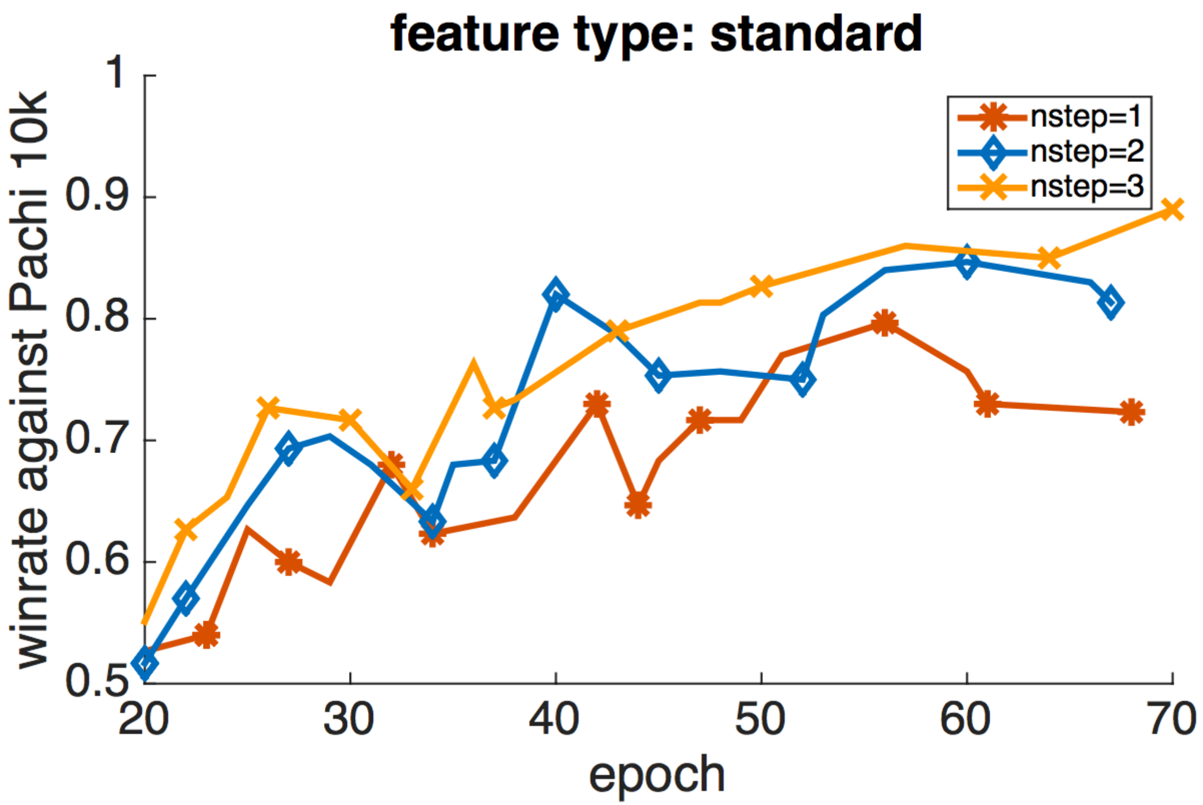
- DCNN as a tree policy
  - Predict next k moves (rather than next move)
  - Trained on 170k KGS dataset/80k GoGoD, ***57.1%*** accuracy.
  - KGS 3D without search (0.1s per move)
  - Release 3 month before AlphaGo, < 1% GPUs (from Aja Huang)



Our computer Go player： DarkForest

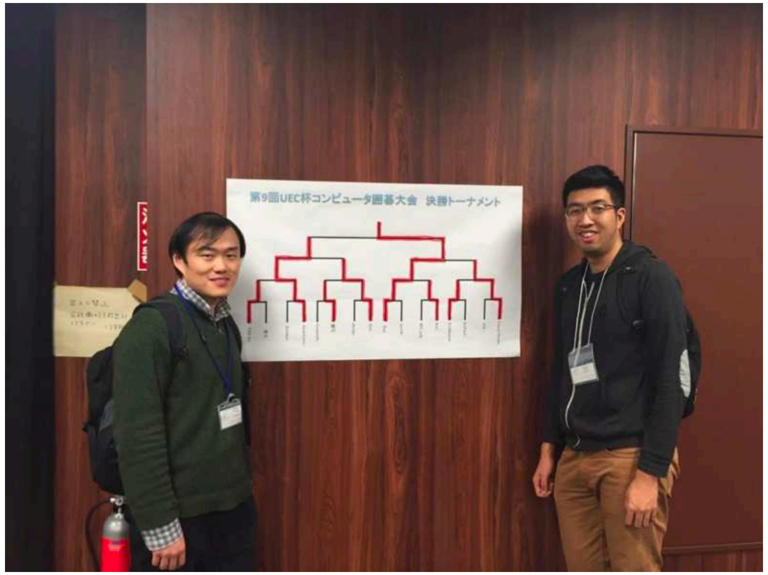
Name
Our/enemy liberties
Ko location
Our/enemy stones/empty place
Our/enemy stone history
Opponent rank

Feature used for DCNN



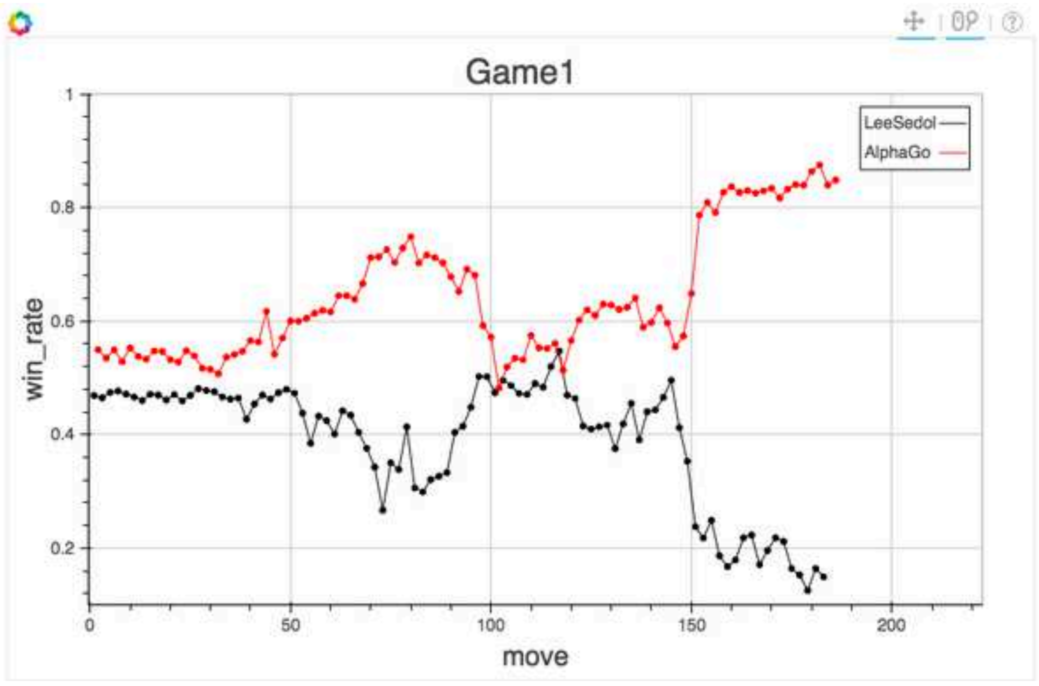
Our computer Go player ： DarkForest

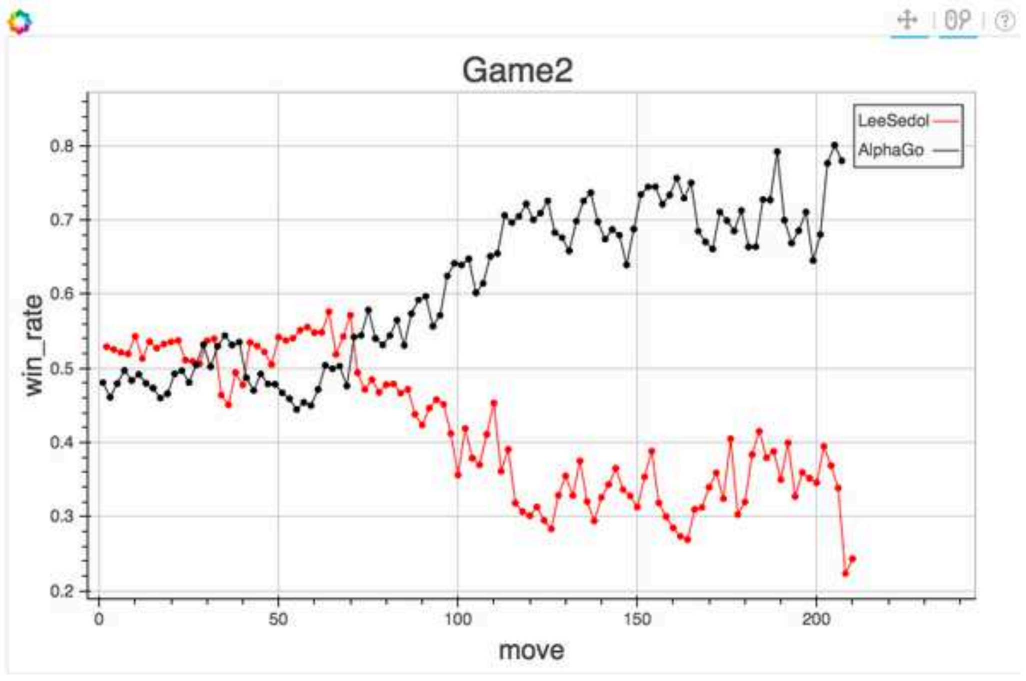
- DCNN+MCTS
  - Use top3/5 moves from DCNN, 75k rollouts.
  - Stable KGS 5d. Open source. https://github.com/facebookresearch/darkforestGo
  - 3rd place on KGS January Tournaments
  - 2nd place in 9th UEC Computer Go Competition (Not this time ☺)



DarkForest versus Koichi Kobayashi (9p)

Win Rate analysis (using DarkForest) (AlphaGo versus Lee Sedol)





Win Rate analysis (using DarkForest)

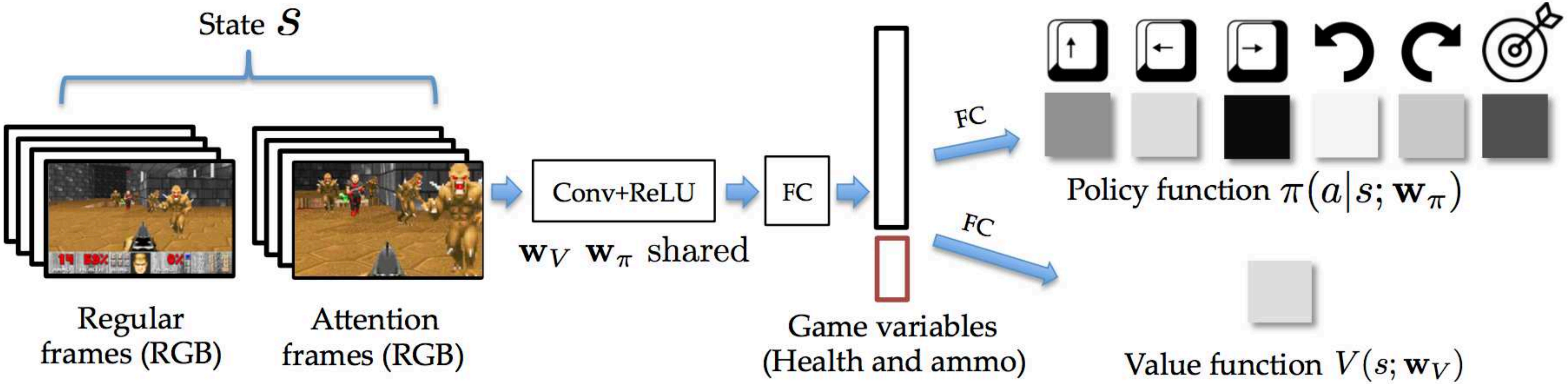
First Person Shooter (FPS) Game

Yuxin Wu, Yuandong Tian, ICLR 2017

Play the game from the raw image!

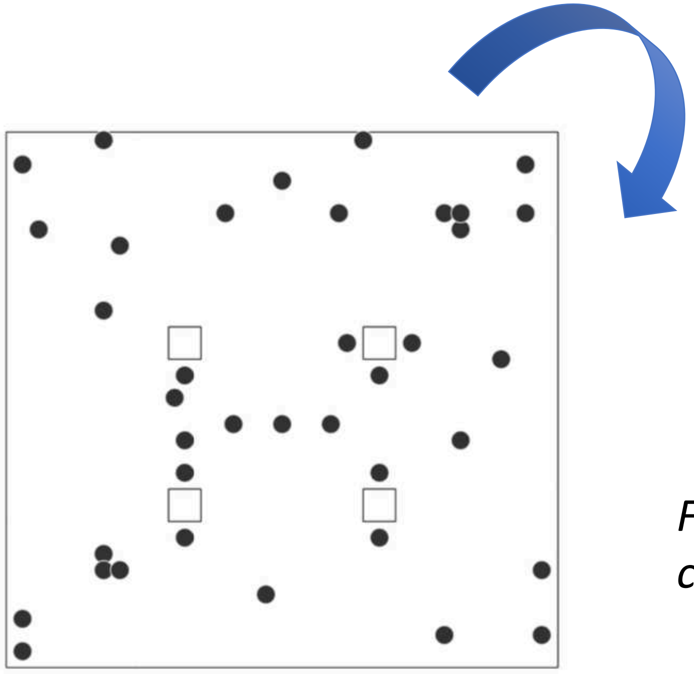
Yuxin Wu

Network Structure



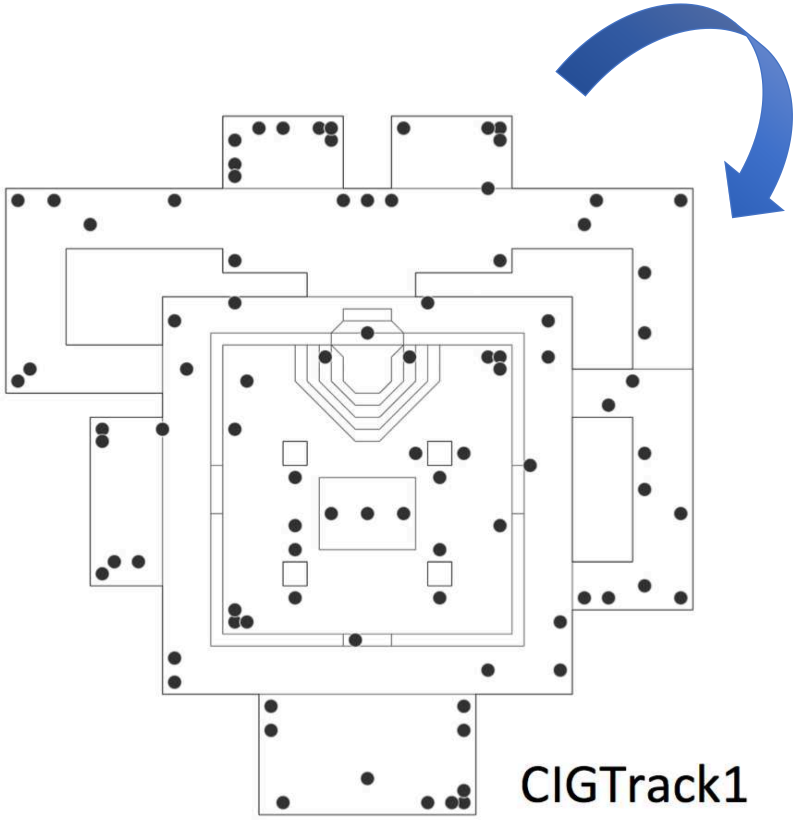
Simple Frame Stacking is very useful (rather than Using LSTM)

Curriculum Training



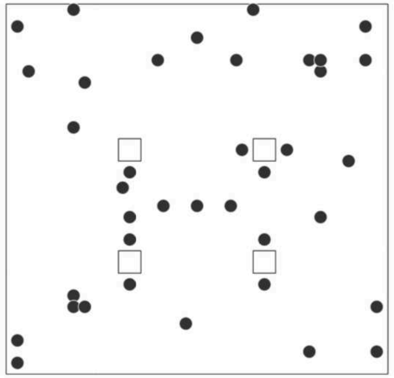
FlatMap

From simple to complicated



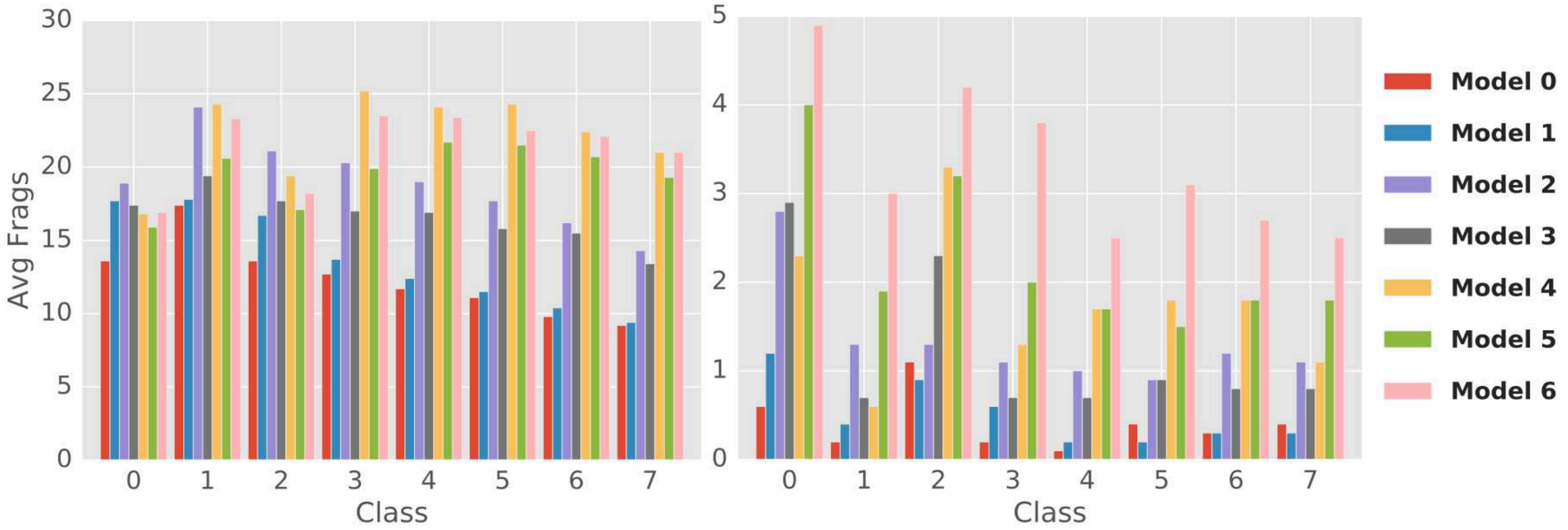
CIGTrack1

Curriculum Training



FlatMap

	Class 0	Class 1	Class 2	Class 3	Class 4	Class 5	Class 6	Class 7
Speed	0.2	0.2	0.4	0.4	0.6	0.8	0.8	1.0
Health	40	40	40	60	60	60	80	100



VizDoom AI Competition 2016 (Track1)

We won the first place!

Rank	Bot	1	2	3	4	5	6	7	8	9	10	11	Total frags
1	F1	**56**	**62**	n/a	**54**	**47**	43	**47**	**55**	**50**	**48**	**50**	**559**
2	Arnold	36	34	**42**	36	36	**45**	36	39	n/a	33	36	413
3	CLYDE	37	n/a	38	32	37	30	46	42	33	24	44	393

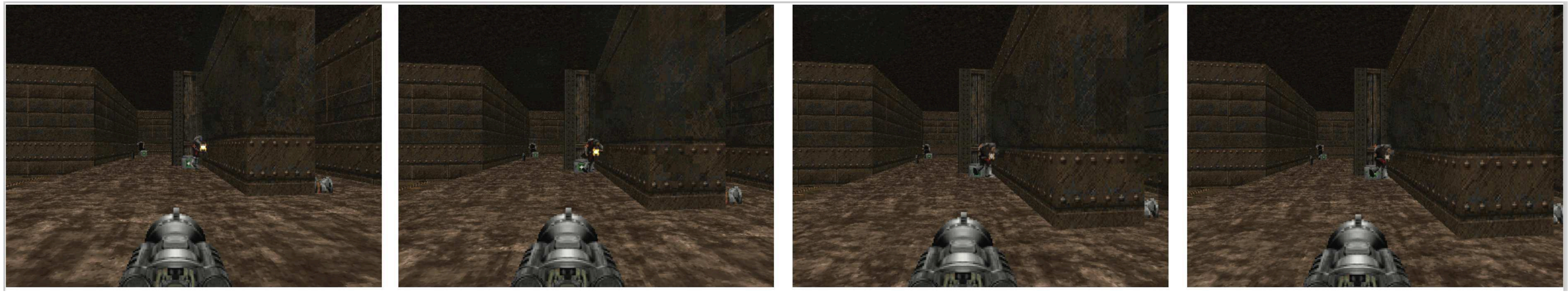
Videos:

https://www.youtube.com/watch?v=94EPSjQH38Y
https://www.youtube.com/watch?v=Qv4esGWOg7w&t=394s

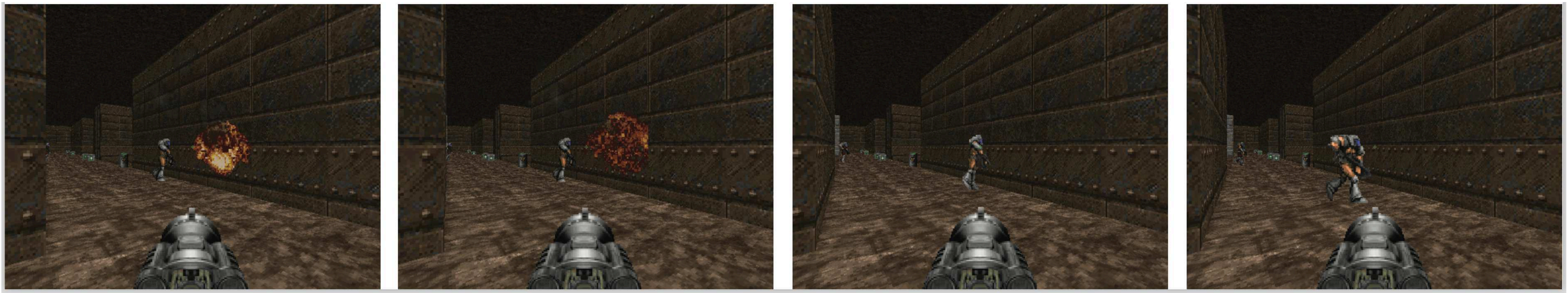

14
93%
1
0%
AMMO
HEALTH
FRAGS
ARMOR
0 / 200
0 / 50
14 / 50
0 / 300

Visualization of Value functions

Best 4 frames (agent is about to shoot the enemy)



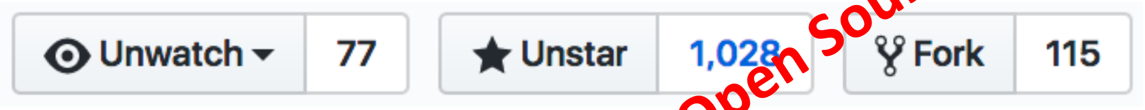
Worst 4 frames (agent missed the shoot and is out of ammo)



ELF: Extensive, Lightweight and Flexible Framework for Game Research

Yuandong Tian, Qucheng Gong, Wendy Shang, Yuxin Wu, Larry Zitnick (NIPS 2017 Oral)

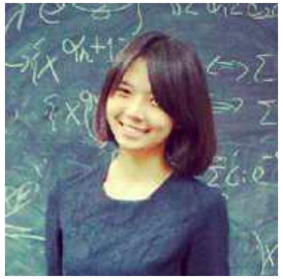
https://github.com/facebookresearch/ELF



Open Sourced!

- Extensive
  - Any games with C++ interfaces can be incorporated.
- Lightweight
  - Fast. Mini-RTS (40K FPS per core)
  - Minimal resource usage (1GPU+several CPUs)
  - Fast training (a couple of hours for a RTS game)
- Flexible
  - Environment-Actor topology
  - Parametrized game environments.
  - Choice of different RL methods.

Qucheng Gong



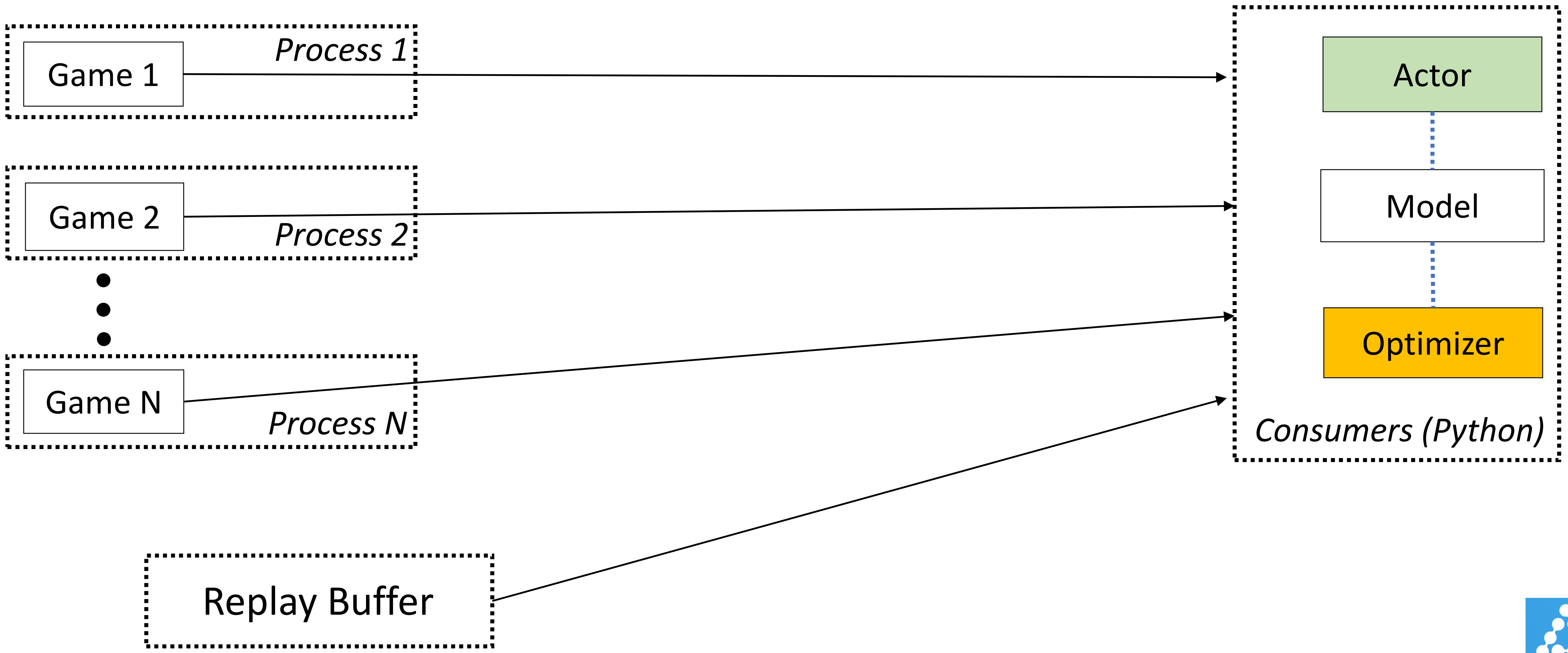
Wendy Shang

Yuxin Wu

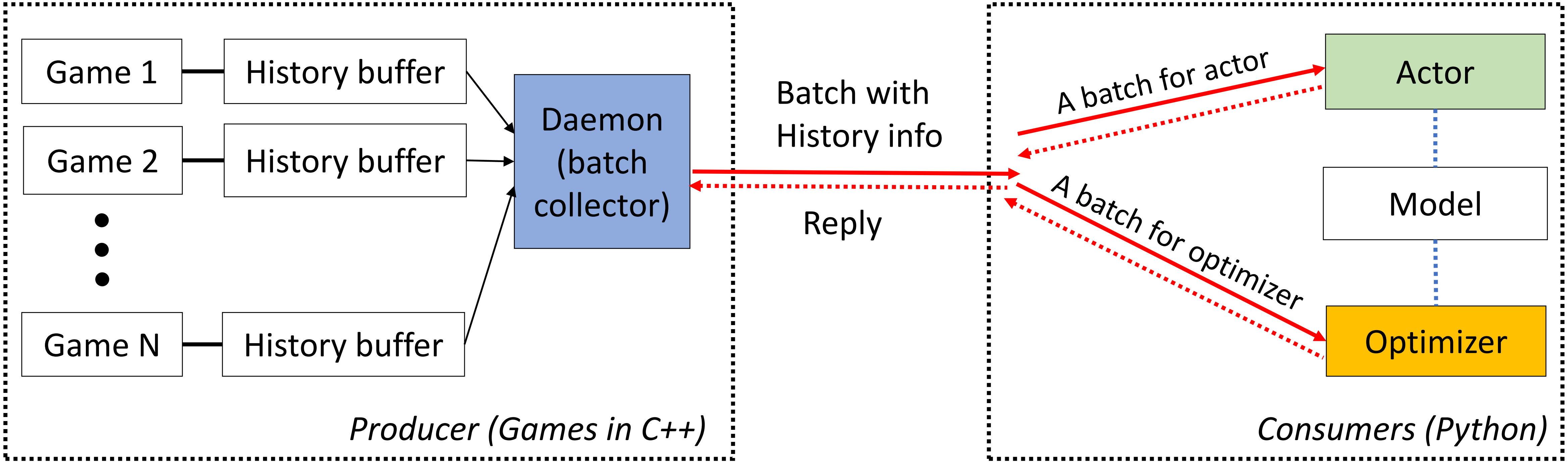
Larry Zitnick

Arxiv: https://arxiv.org/abs/1707.01067

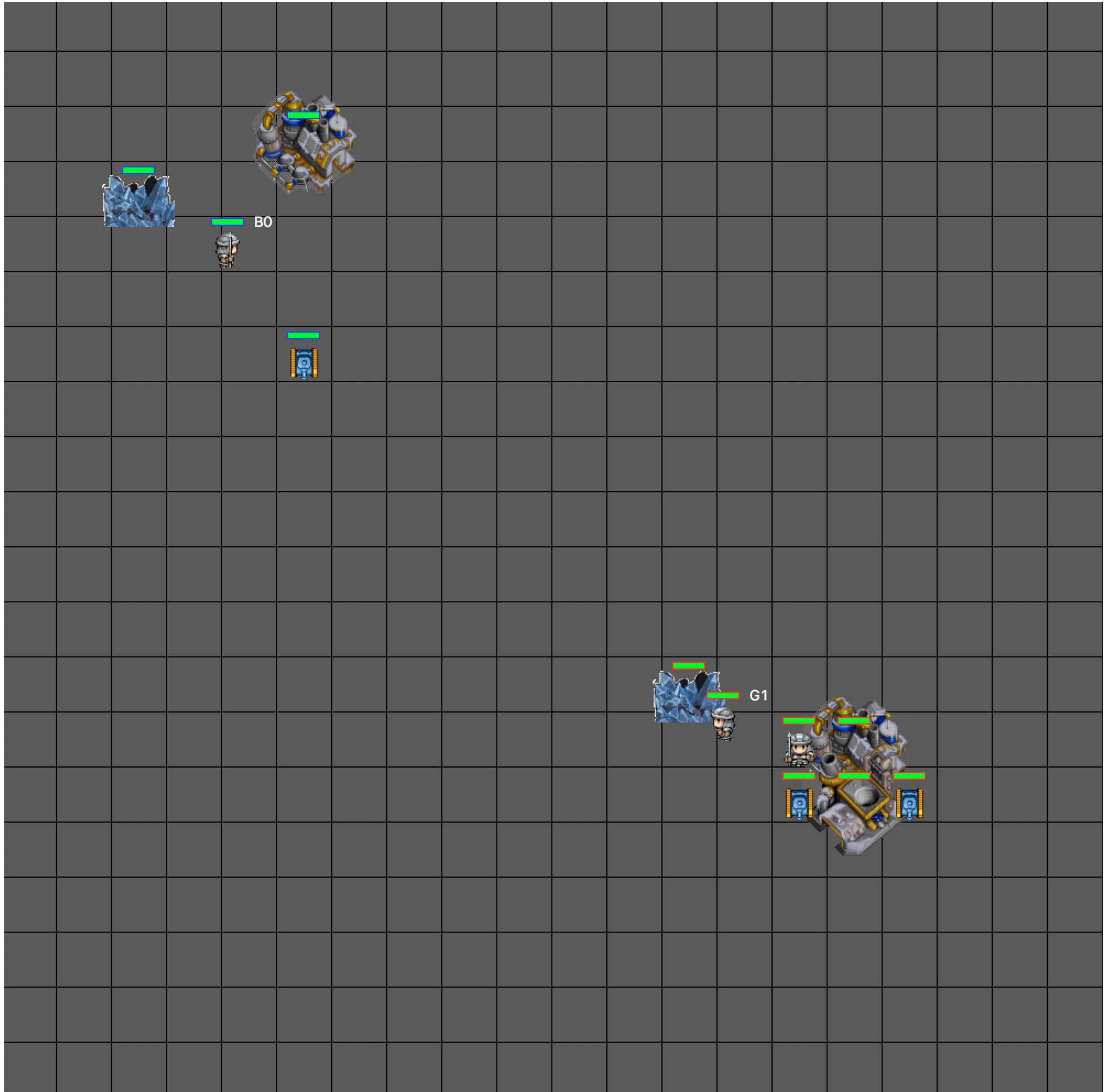
How RL system works



ELF design



Plug-and-play; no worry about the concurrency anymore.


B0
G1

Possible Usage

- Game Research
  - Board game (Chess, Go, etc)
  - Real-time Strategy Game
- Complicated RL algorithms.
- Discrete/Continuous control
  - Robotics
- Dialog and Q&A System

Initialization

```
# Sample Usage
# We run 1024 games concurrently.
num_games = 1024

# Every time we wait for an arbitrary set of 256 games and return.
batchsize = 256

# The return states contain key 's', 'r' and 'terminal'
# and the reply contains key 'a', 'V' and 'pi', which is to be filled from the Python side.
# Their definitions are defined in the C++ wrapper of the game.
desc = dict(
    actor = dict(
        batchsize=args.batchsize,
        input=dict(T=1, keys=set(["s", "last_r", "last_terminal"])),
        reply=dict(T=1, keys=set(["pi", "V", "a"]))
    )
)

GameContext = InitializeGame(num_games, batchsize, desc)
```

Main Loop

```
# Start all game threads
GameContext.Start()

while True:
    # Wait until a batch of game states are returned.
    # Note that these game instances will be blocked.
    batch = GameContext.Wait()
    if batch.desc == "actor":
        # Apply a model to the game state. you can do forward/backward propagation here.
        output = model(batch)

        # Sample from the output to get the actions of this batch.
        reply['pi'][:] = output['pi']
        reply['a'][:] = SampleFromDistribution(output)
        reply['V'][:] = output["V"]

    # Resume games.
    GameContext.Steps()

# Stop all game threads.
GameContext.Stop()
```

Training

```
desc = dict(
    actor = dict(
        batchsize=args.batchsize,
        input=dict(T=1, keys=set(["s", "last_r", "last_terminal"])),
        reply=dict(T=1, keys=set(["pi", "v", "a"]))
    ),
    train = dict(
        batchsize=args.batchsize,
        input=dict(T=6, keys=set(["s", "last_r", "last_terminal", "a", "pi"])),
        reply=None
    )
)
while True:
    ...
    if batch["desc"] == "actor":
        # Act given the current states to move the game environment forward.
        # It could be an act for a game, for its internal MCTS search, etc.
    elif batch["desc"] == "train":
        # Train your model. All the previous actions of the games and
        # their probabilities can be made available.
    ...
```

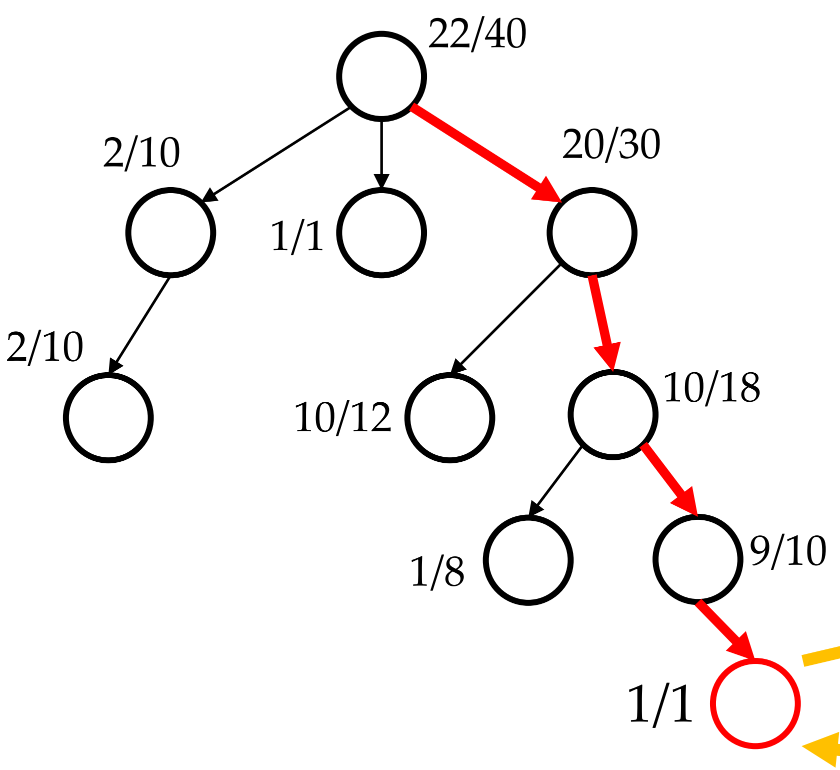
Self-Play

```
desc = dict(
    actor0 = dict(
        batchsize=args.batchsize,
        input=dict(T=1, keys=set(["s", "last_r", "last_terminal"])),
        reply=dict(T=1, keys=set(["pi", "v", "a"])),
        filter=dict(id=0)
    ),
    actor1 = dict(
        batchsize=args.batchsize,
        input=dict(T=1, keys=set(["s", "last_r", "last_terminal"])),
        reply=dict(T=1, keys=set(["pi", "v", "a"])),
        filter=dict(id=1)
    ),
    train = dict(
        batchsize=args.batchsize,
        input=dict(T=6, keys=set(["s", "last_r", "last_terminal", "a", "pi"])),
        reply=None,
        filter=dict(id=0)
    )
)
while True:
    ...
    if batch["desc"] == "actor0":
        # Act for player 0
    elif batch["desc"] == "actor1":
        # Act for player 1
    elif batch["desc"] == "train":
        # Train your model only for player 0.
    ...
```

Multi-Agent

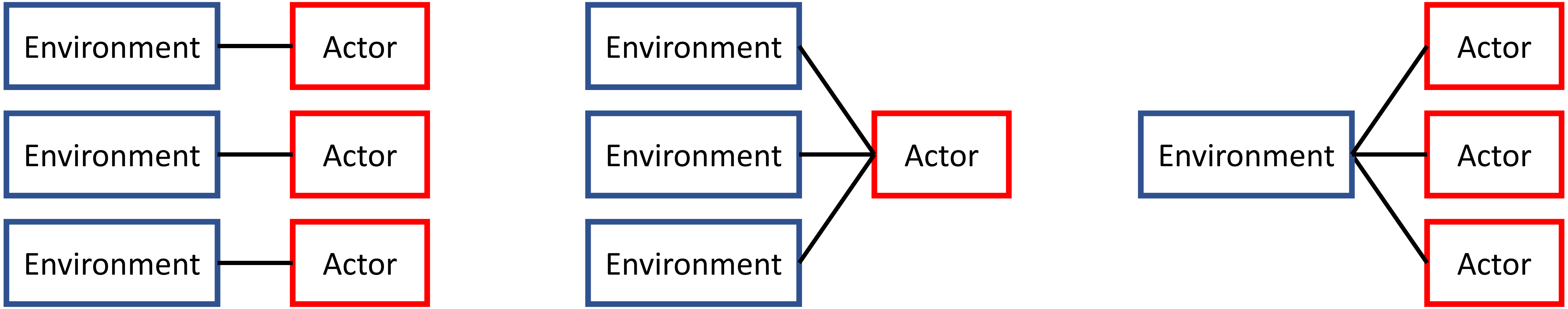
```
desc = { }
for i in range(num_agents):
    desc["actor%d" % i] = dict(
        batchsize=args.batchsize,
        input=dict(T=1, keys=set(["s", "last_r", "last_terminal"])),
        reply=dict(T=1, keys=set(["pi", "v", "a"])),
        filter=dict(id=i)
    )
while True:
    ...
    for i in range(num_agents):
        if batch["desc"] == "actor%d" % i:
            # Act for player i
    ...
```

Monte-Carlo Tree Search




```
desc = dict(
    actor = dict(
        batchsize=args.batchsize,
        input=
          dict(T=1,
               keys=set([
                  "s", "last_r", "last_terminal"])),
        reply=dict(T=1, keys=set(["pi", "v", "a"])),
    )
)
while True:
    batch = GameContext.Wait()
    if batch["desc"] == "actor":
        # Act for player. During MCTS search, one
        # game instance could send multiple requests
        # for python side to respond.
    GameContext.Step()
```

Flexible Environment-Actor topology



(a) One-to-One
Vanilla A3C

(b) Many-to-One
BatchA3C, GA3C

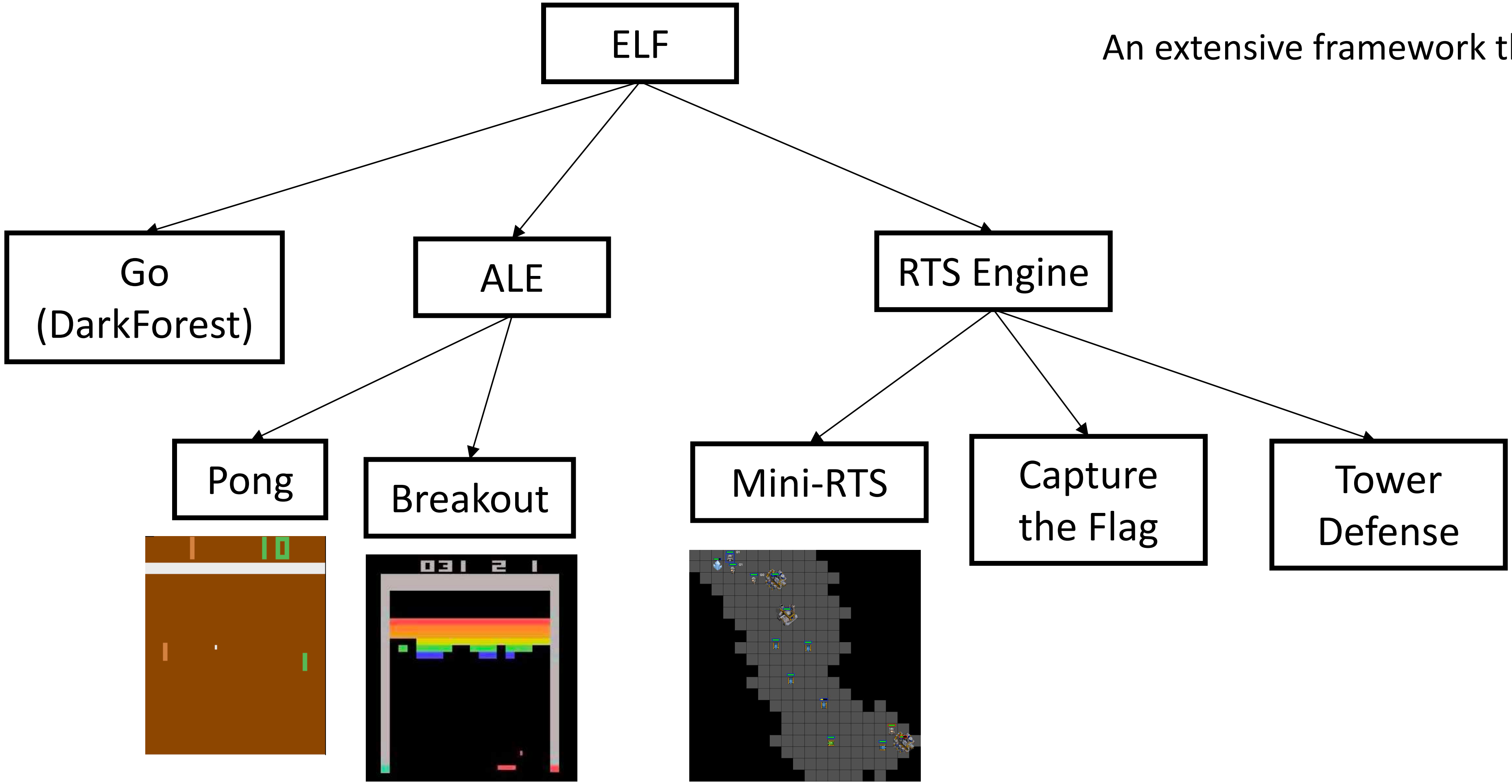
(c) One-to-Many
Self-Play,
Monte-Carlo Tree Search

RLPytorch

- A RL platform in PyTorch
- A3C in 30 lines.
- Interfacing with dict.

```
# A3C
def update(self, batch):
    ''' Actor critic model '''
    R = deepcopy(batch["V"][T - 1])
    batchsize = R.size(0)
    R.resize_(batchsize, 1)

    for t in range(T - 2, -1, -1):
        # Forward pass
        curr = self.model_interface.forward("model", batch.hist(t))

        # Compute the reward.
        R = R * self.args.discount + batch["r"][t]
        # If we see any terminal signal, do not backprop
        for i, terminal in enumerate(batch["terminal"][t]):
            if terminal: R[t][i] = curr["V"].data[i]

        # We need to set it beforehand.
        self.policy_gradient_weights = R - curr["V"].data

        # Compute policy gradient error:
        errs = self._compute_policy_entropy_err(curr["pi"], batch["a"][t])
        # Compute critic error
        value_err = self.value_loss(curr["V"], Variable(R))

        overall_err = value_err + errs["policy_err"]
        overall_err += errs["entropy_err"] * self.args.entropy_ratio
        overall_err.backward()
```

Architecture Hierarchy

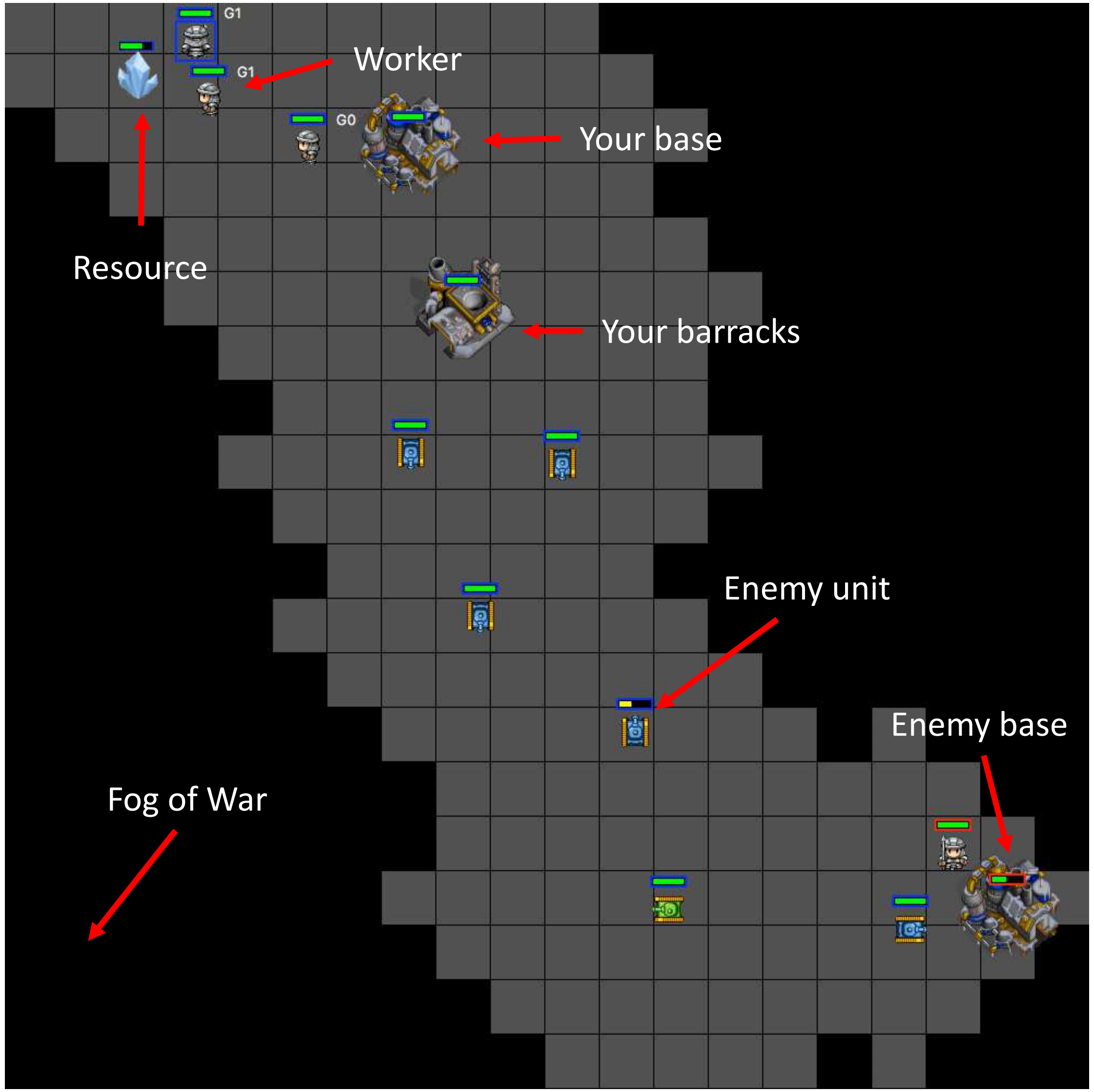


An extensive framework that can host many games.

Specific game engines.

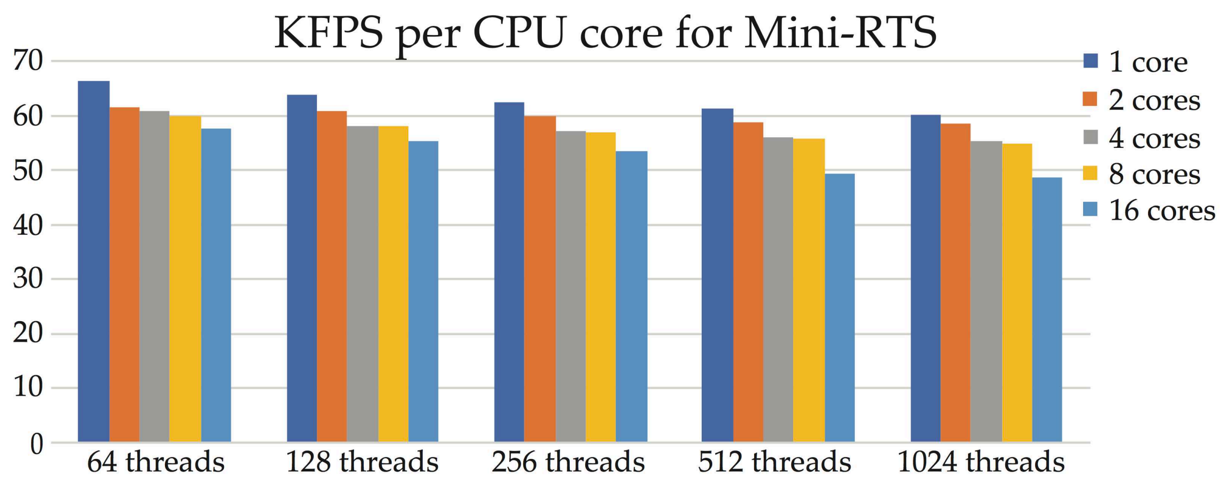
Environments

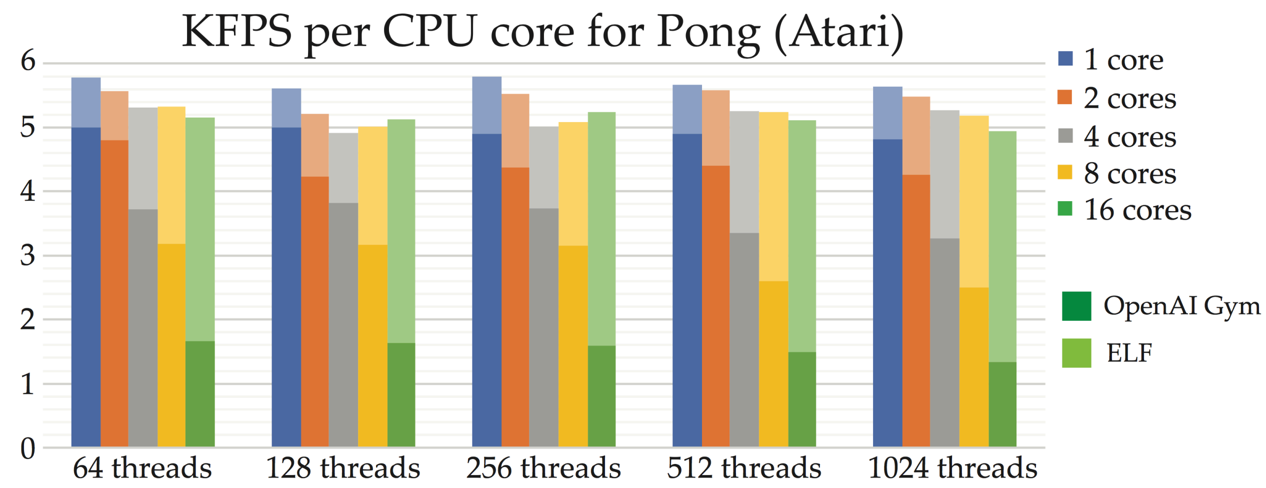
A miniature RTS engine



Game Name	Descriptions	Avg Game Length
Mini-RTS	Gather resource and build troops to destroy opponent's base.	1000-6000 ticks
Capture the Flag	Capture the flag and bring it to your own base	1000-4000 ticks
Tower Defense	Builds defensive towers to block enemy invasion.	1000-2000 ticks

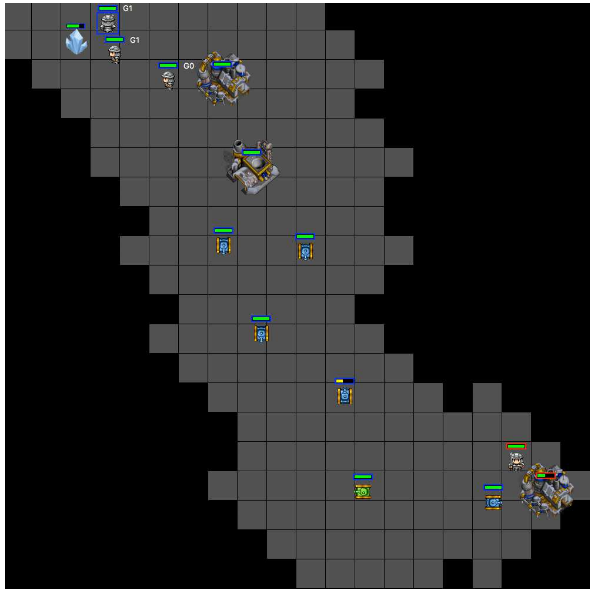
Simulation Speed



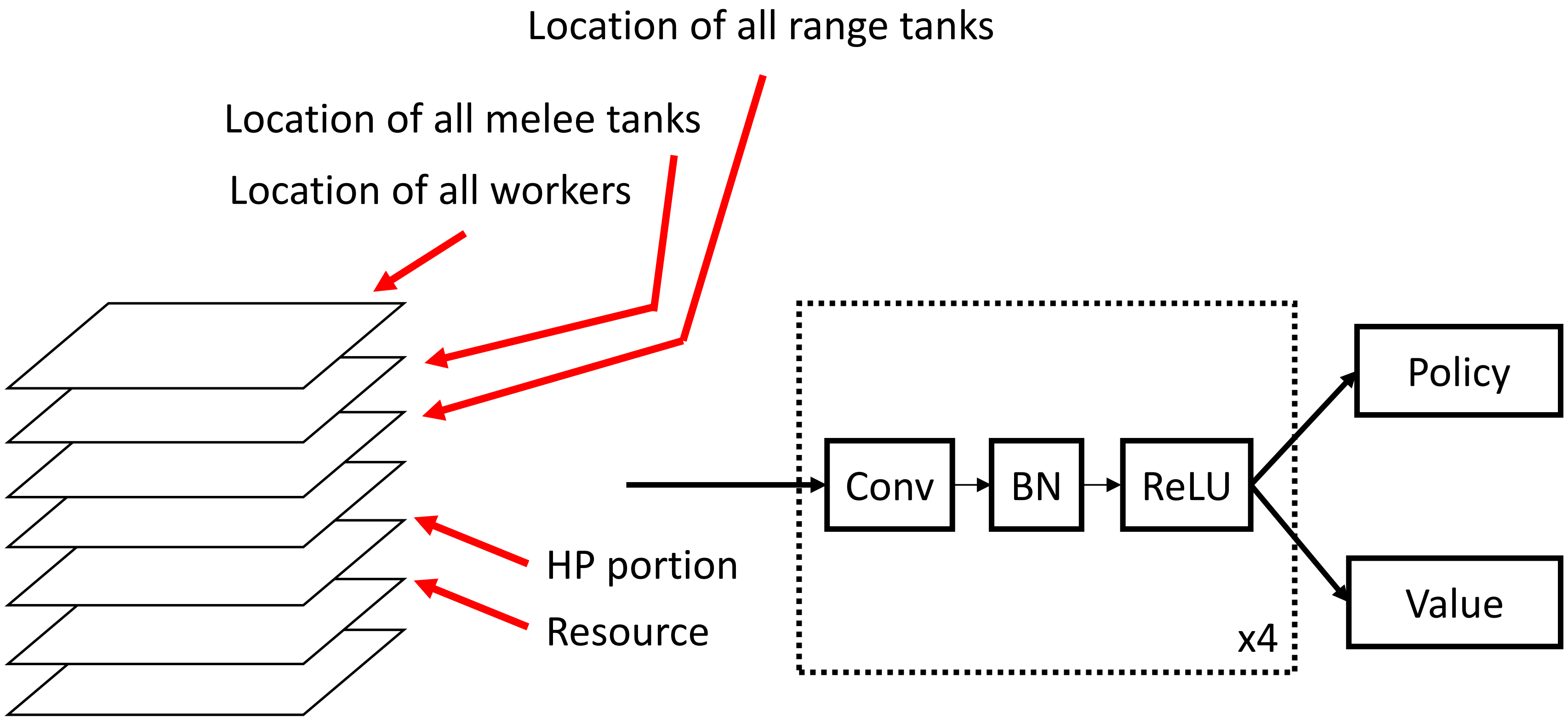


Platform	**ALE**	**RLE**	**Universe**	**Malmo**
FPS	6000	530	60	120
Platform	**DeepMind Lab**	**VizDoom**	**TorchCraft**	***Mini-RTS***
FPS	287(C) / 866(G) 6CPU + 1GPU	7,000	2,000 (Frameskip=50)	*40,000*

Training AI



Game visualization



Game internal data
(respecting fog of war)

Using Internal Game data and A3C.
Reward is only available once the game is over.

MiniRTS

Building that can build workers and collect resources.

Resource unit that contains 1000 minerals.

Building that can build melee attacker and range attacker.

Worker who can build barracks and gather resource.
Low speed in movement and low attack damage.

Tank with high HP, medium movement speed, short attack range, high attack damage.

Tank with low HP, high movement speed, long attack range and medium attack damage.

Training AI

9 discrete actions.

No.	Action name	Descriptions
1	IDLE	Do nothing
2	BUILD WORKER	If the base is idle, build a worker
3	BUILD BARRACK	Move a worker (gathering or idle) to an empty place and build a barrack.
4	BUILD MELEE ATTACKER	If we have an idle barrack, build an melee attacker.
5	BUILD RANGE ATTACKER	If we have an idle barrack, build a range attacker.
6	HIT AND RUN	If we have range attackers, move towards opponent base and attack. Take advantage of their long attack range and high movement speed to hit and run if enemy counter-attack.
7	ATTACK	All melee and range attackers attack the opponent's base.
8	ATTACK IN RANGE	All melee and range attackers attack enemies in sight.
9	ALL DEFEND	All troops attack enemy troops near the base and resource.

Win rate against rule-based AI

Frame skip (how often AI makes decisions)

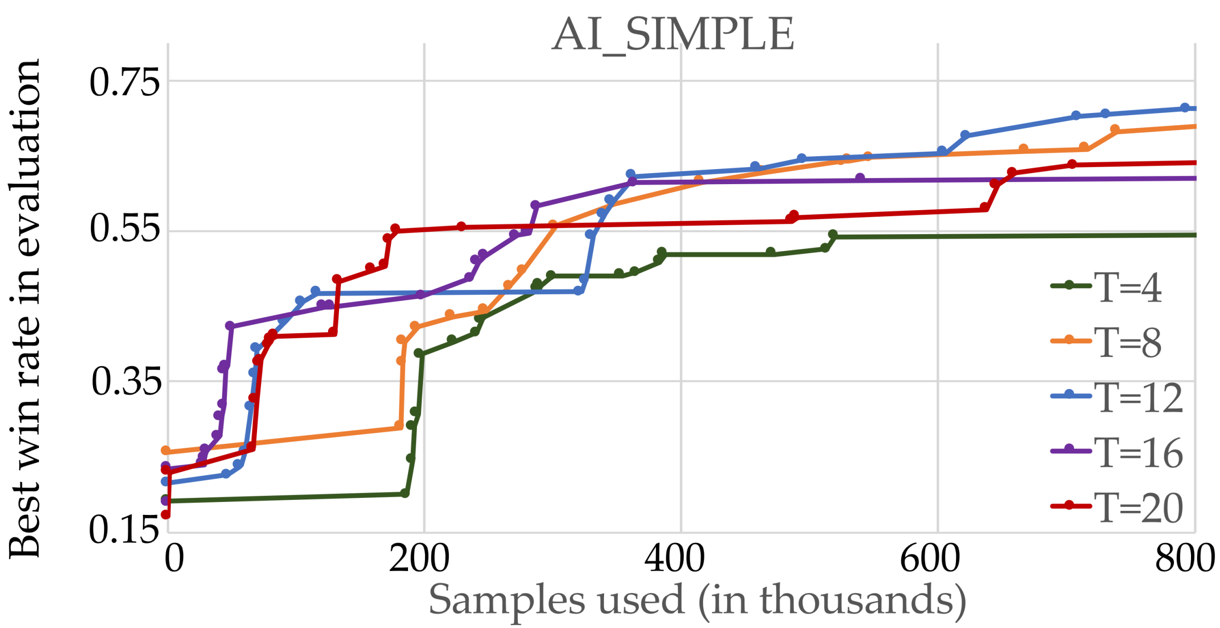
Frame skip	AI_SIMPLE	AI_HIT_AND_RUN
50	68.4(±4.3)	63.6(±7.9)
20	61.4(±5.8)	55.4(±4.7)
10	52.8(±2.4)	51.1(±5.0)

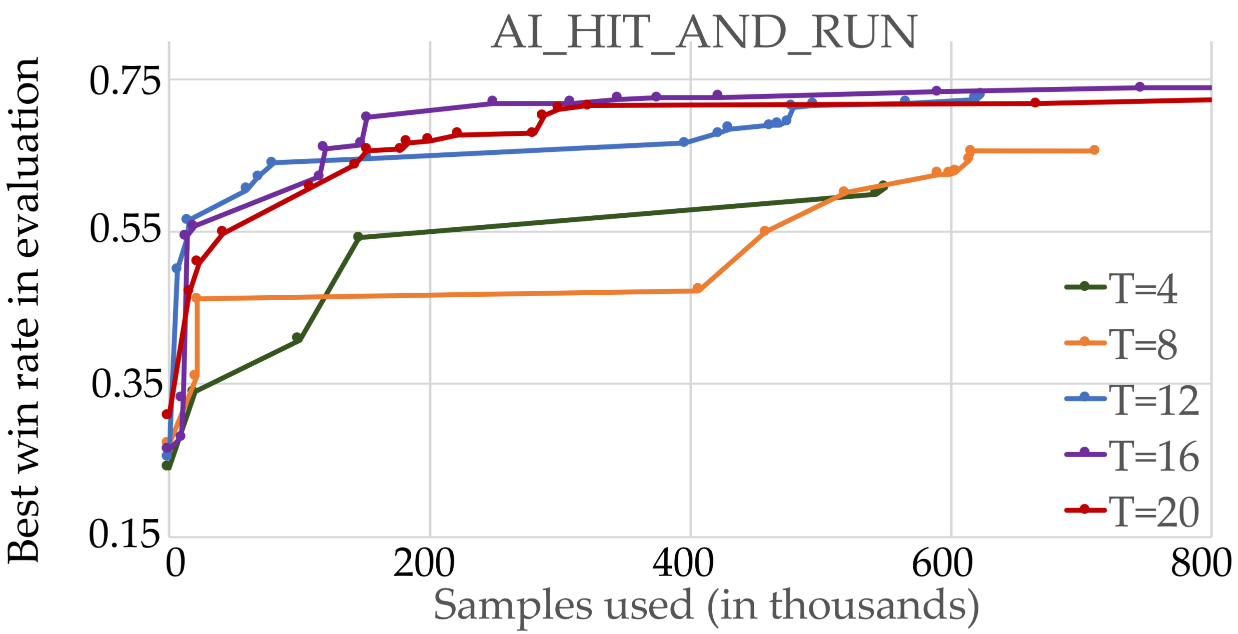
Network Architecture

Conv → BN → ReLU

	SIMPLE (median)	SIMPLE (mean/std)	HIT_AND_RUN (median)	HIT_AND_RUN (mean/std)
ReLU	52.8	54.7(±4.2)	60.4	57.0(±6.8)
Leaky ReLU	59.8	61.0(±2.6)	60.2	60.3(±3.3)
ReLU + BN	61.0	64.4(±7.4)	55.6	57.5(±6.8)
Leaky ReLU + BN	72.2	68.4(±4.3)	65.5	63.6(±7.9)

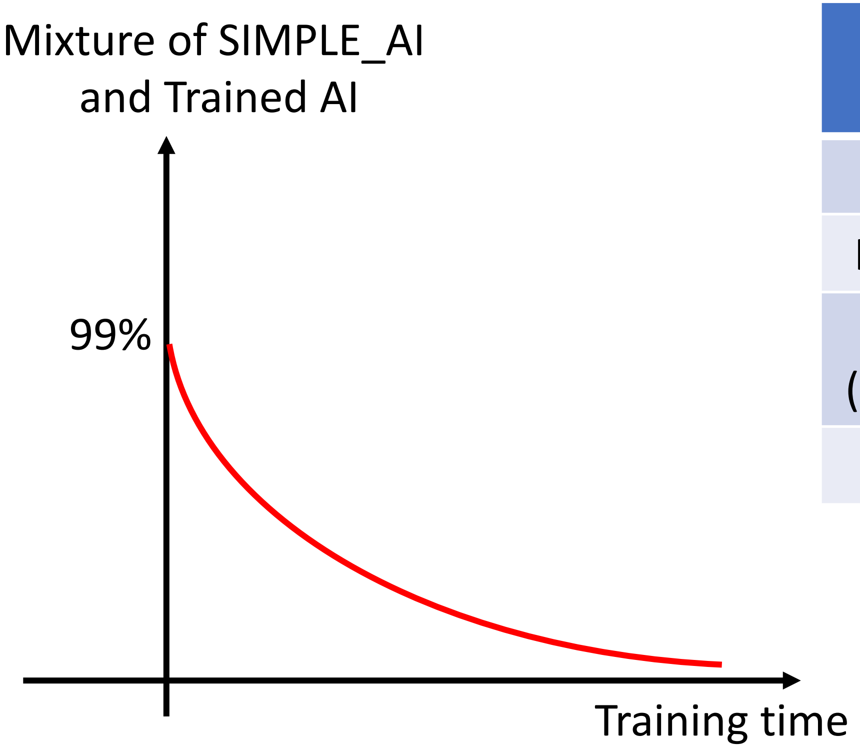
Effect of T-steps





Large T is better.

Transfer Learning and Curriculum Training



	AI_SIMPLE	AI_HIT_AND_RUN	Combined (50%SIMPLE+50% H&R)
SIMPLE	**68.4 (±4.3)**	26.6(±7.6)	47.5(±5.1)
HIT_AND_RUN	34.6(±13.1)	**63.6 (±7.9)**	49.1(±10.5)
Combined (No curriculum)	49.4(±10.0)	46.0(±15.3)	47.7(±11.0)
Combined	51.8(±10.6)	54.7(±11.2)	**53.2(±8.5)**

Highest win rate against AI_SIMPLE: 80%

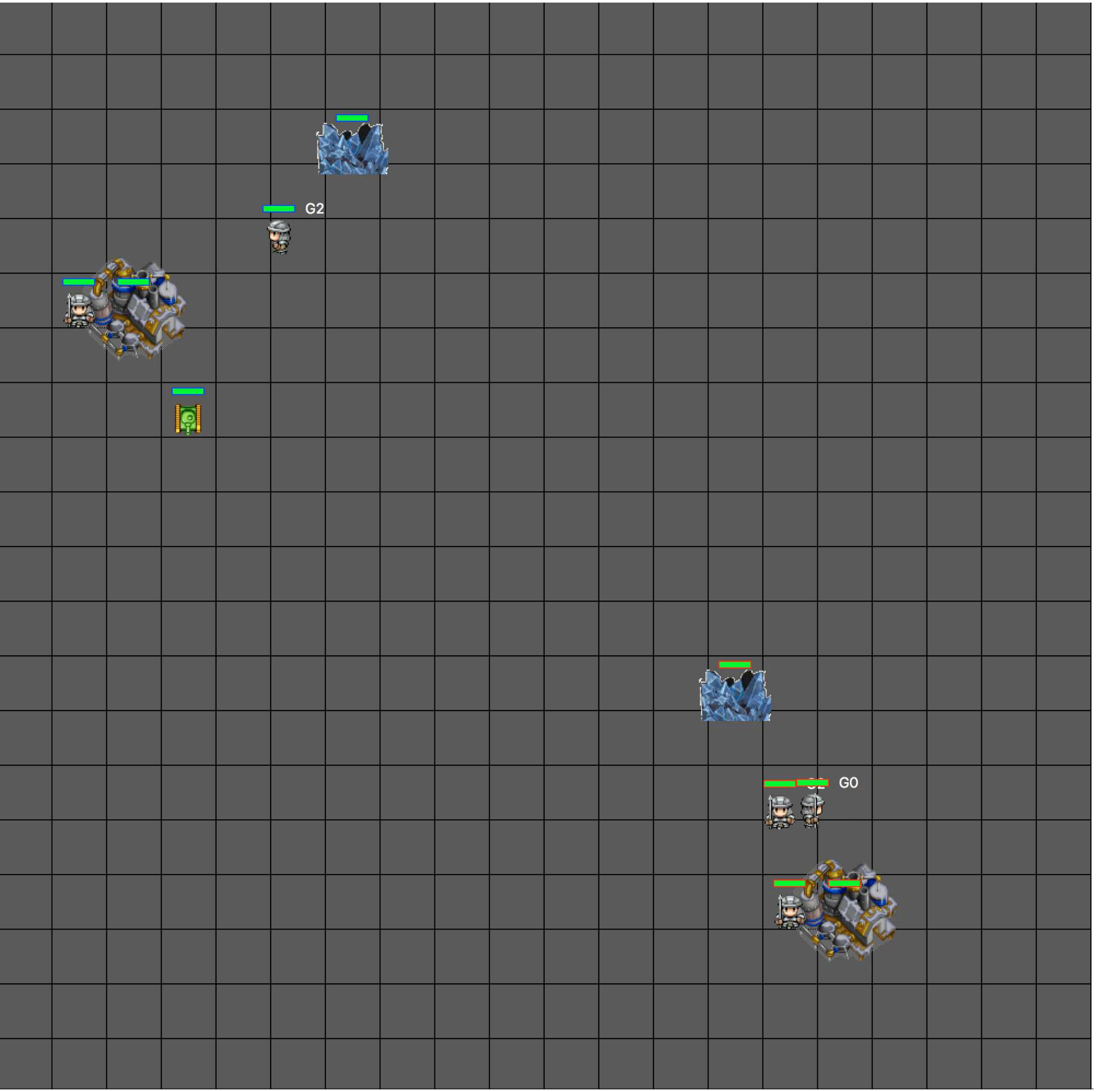
	AI_SIMPLE	AI_HIT_AND_RUN	CAPTURE_THE_FLAG
Without curriculum training	66.0 (±2.4)	54.4 (±15.9)	54.2 (±20.0)
With curriculum training	**68.4 (±4.3)**	**63.6 (±7.9)**	**59.9 (±7.4)**

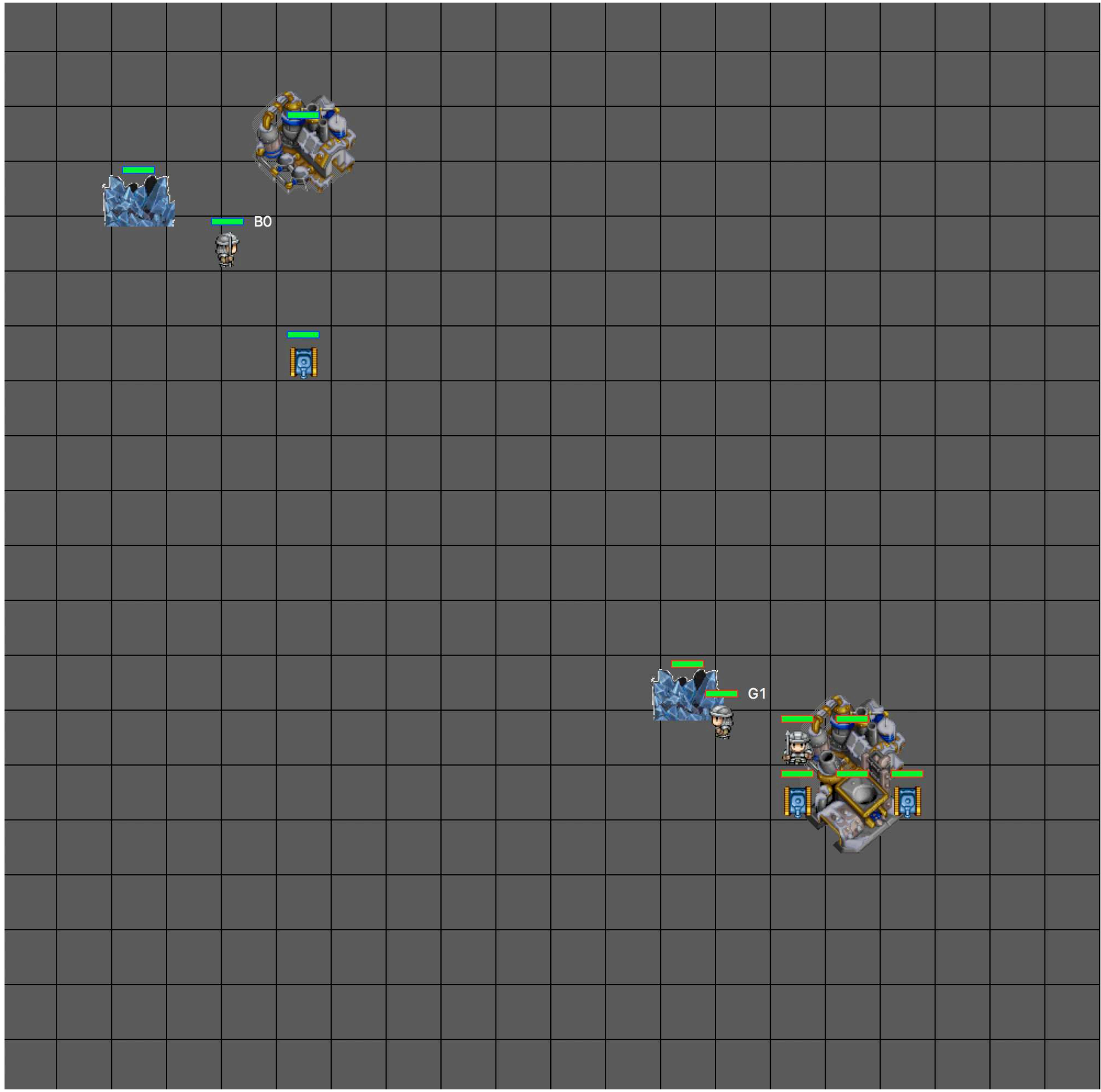
Monte Carlo Tree Search

	MiniRTS (AI_SIMPLE)	MiniRTS (Hit_and_Run)
Random	24.2 (±3.9)	25.9 (±0.6)
MCTS	73.2 (±0.6)	62.7 (±2.0)

MCTS evaluation is repeated on 1000 games, using 800 rollouts.

MCTS uses complete information and perfect dynamics


G2
G0


B0
G1

Thanks!