

# Artificial Neural Networks: Lecture 11

Johanni Brea  
EPFL, Lausanne, Switzerland

## Use Cases of Deep Reinforcement Learning

### Outline of today:

- A3C, DQN and decorrelation for deep RL
- RL in the ATARI domain
- Replay Memory and Backward Planning in tabular environments
- Forward Planning in model-based RL (board games): Minimax vs. Monte Carlo Tree Search
- Alpha Zero
- Limitations of deep RL

## Reading for this lecture:

**Sutton and Barto 2018** *Reinforcement Learning*

- Ch 16, 8, (optional 17)

## Further (optional) reading for this lecture:

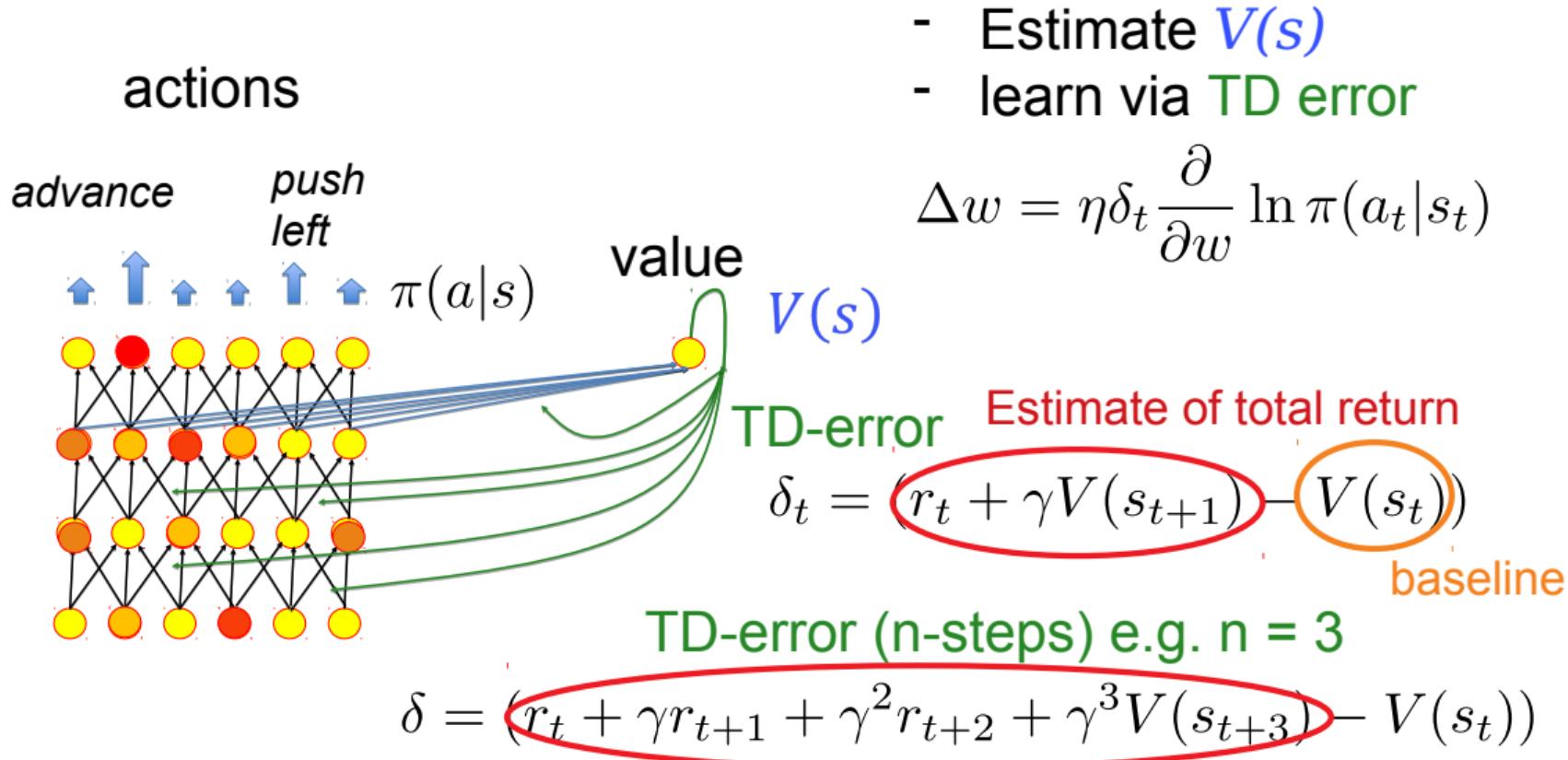
See references in the slides.

# **Asynchronous Advantage Actor Critic (A3C)**

# **Deep Q-Learning (DQN)**

# **And Decorrelation for Deep RL**

# Review: Actor-Critic Policy Gradient



# Asynchronous game play and entropy regularization

1) Minibatches allow to leverage parallelization e.g. GPU

Problem: Minibatches for Actor-Critic Policy Gradient?

Proposed solution: Interact with N environments in parallel.

2) Policy can become deterministic too quickly, e.g.

$$\pi(a = 1|s) = \frac{\exp(w_1 s)}{\sum_i \exp(w_i s)} \approx 1 \text{ if } w_1 \gg w_i$$

Proposed solution: add entropy  $H(\pi)$  to cost function  
(regularization to keep differences between w's small).

A3C = Asynchronous (interaction with N environments)

Advantage (TD-error = advantage of chosen action)

Actor (policy network)

Critic (value network)

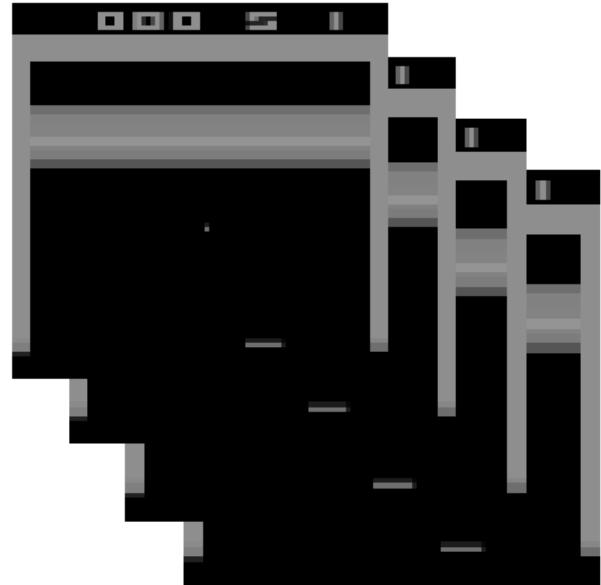
# Atari Video Games (preprocessing)



$o(t)$  = original

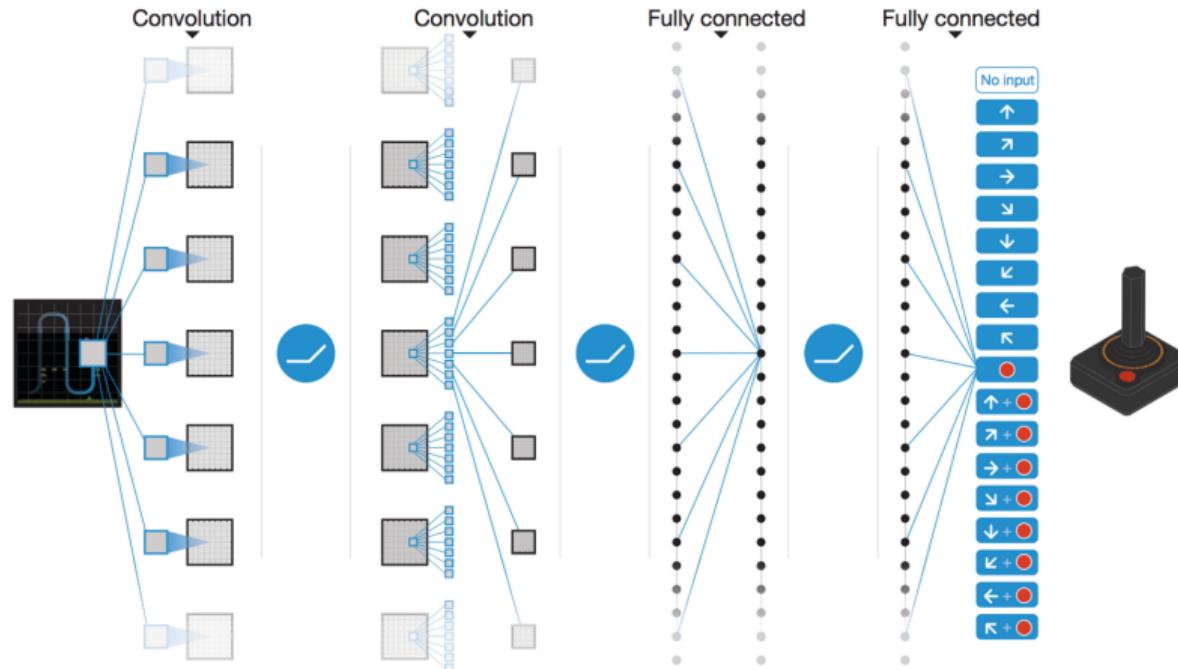


$g(t)$  = grayscaled(  
downsampled( $o(t)$ ))



$s(t) = (g(t), \dots, g(t-3))$   
**input to convnet**

# Learning Atari Games with A3C



- $8 \times 8 \times 32$  stride 4  $\Rightarrow 4 \times 4 \times 64$  stride 2  $\Rightarrow 3 \times 3 \times 64 \Rightarrow 512 \Rightarrow 4 - 18$
- 16 parallel threads on CPU per game for up to 4 days to reach superhuman performance in 57 games

# Deep Q-Network

$Q_a(s)$  = same network as on previous slide (different interpretation)

$\hat{Q}_a(s)$  = copy of  $Q_a(s)$  with old parameters (target network)

- 1: **for** all steps **do**
- 2:   select action  $a_t$  with  $\epsilon$ -greedy policy using  $Q_a(s_t)$  standard preprocessing
- 3:   observe reward  $r_t$  and image  $x_{t+1}$  and preprocess  $s_{t+1} = \phi(s_t, x_{t+1})$
- 4:   Store transition  $(s_t, a_t, r_t, s_{t+1})$  in **replay memory** 1M transitions
- 5:   Sample random minibatch  $(s_j, a_j, r_j, s_{j+1})$  from replay memory
- 6:   Perform (semi-)gradient step on  $\mathcal{L}\left(r_j + \gamma \max_a \hat{Q}_a(s_{j+1}) - Q_{a_j}(s_j)\right)$
- 7:   Every  $C$  steps reset  $\hat{Q} = Q$ .
- 8: **end for**

$$\mathcal{L}(x) = \begin{cases} |x| & x > 1 \\ x^2 & \text{otherwise} \end{cases}$$

# Why do target networks help?

Exercise 1  
now (8 min)

- 1) Stabilization in the case of input correlations
- 2) Reduction of maximization bias with double Q-Learning

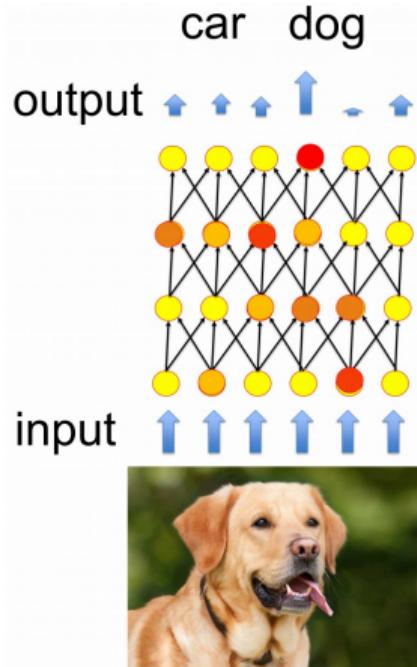
## Side remark

State of the Art Control of Atari Games Using Shallow RL

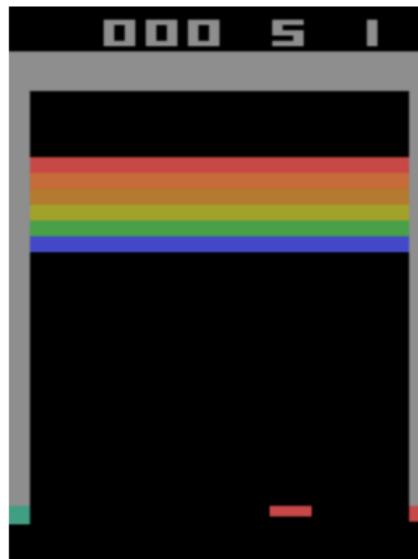
<https://arxiv.org/abs/1512.01563>

Using simple hand-crafted features (relative position of pixels with certain colors at different time-lags) leads to similar performance as DQN with SARSA and much less training time.

# Decorrelation for Deep RL



Classification:  
**uncorrelated sampling**  
of training data



RL:  
Subsequent inputs often  
**highly correlated**

## Possible solutions

- 1) Parallel interaction with N independent environments (A3C)
- 2) Sampling from replay memory (DQN)

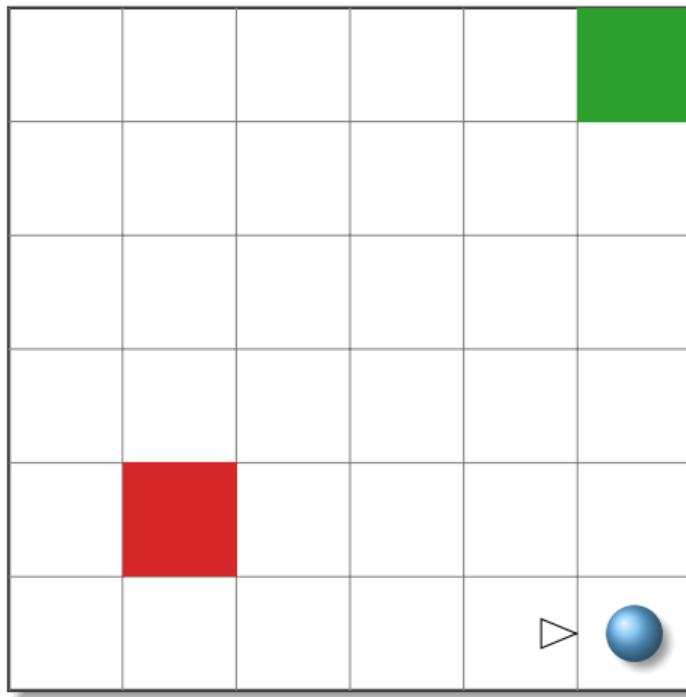
# **Replay Memory and Planning in Tabular Environments**

## standard tabular Q-Learning

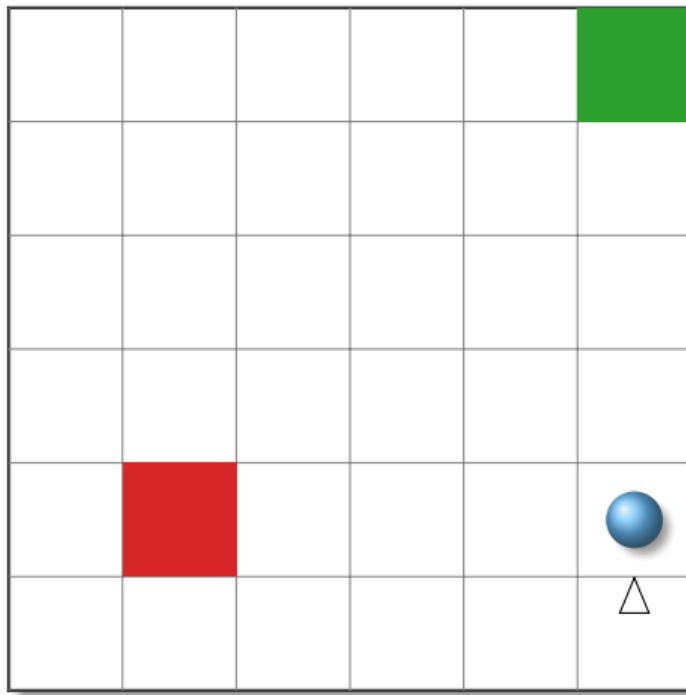
31	32	33	34	35	36
25	26	27	28	29	30
19	20	21	22	23	24
13	14	15	16	17	18
7	8	9	10	11	12
1	2	3	4	5	6



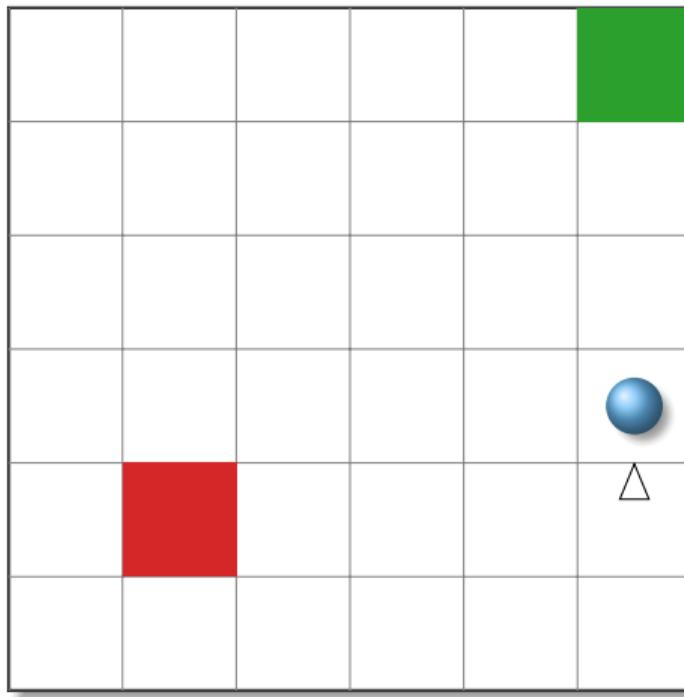
## standard tabular Q-Learning



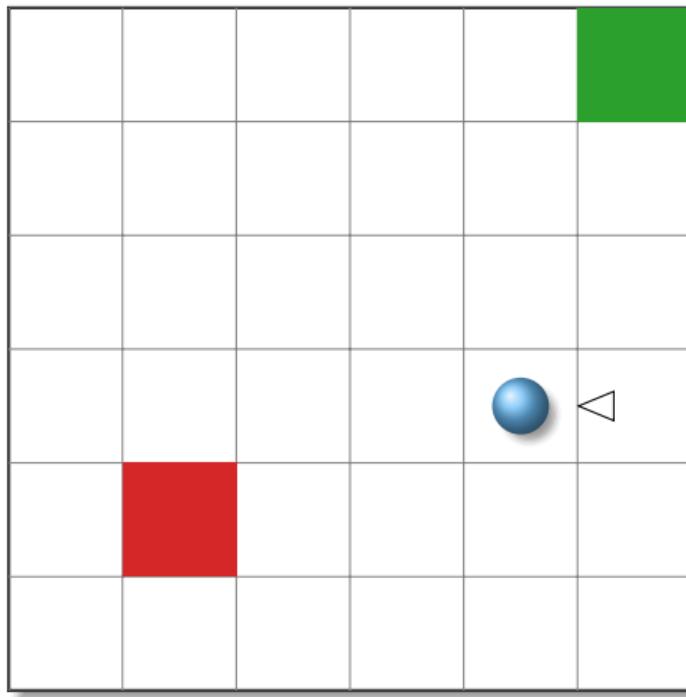
## standard tabular Q-Learning



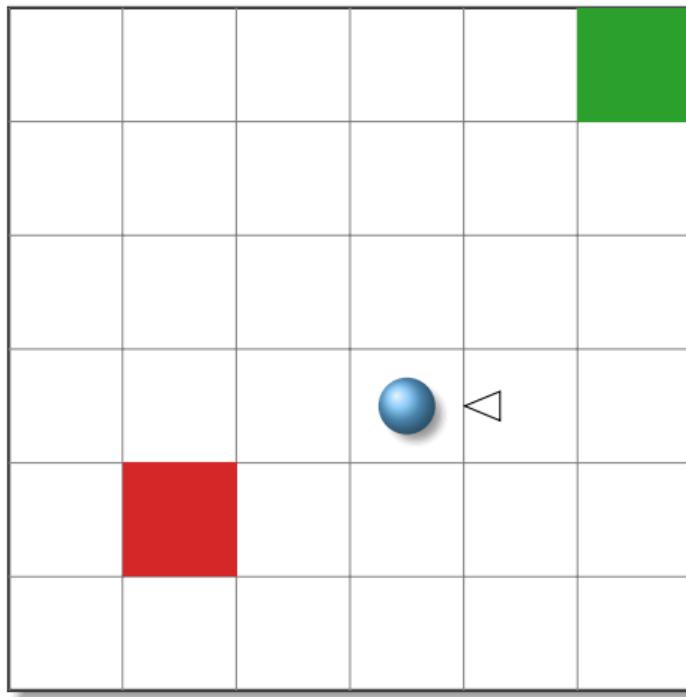
## standard tabular Q-Learning



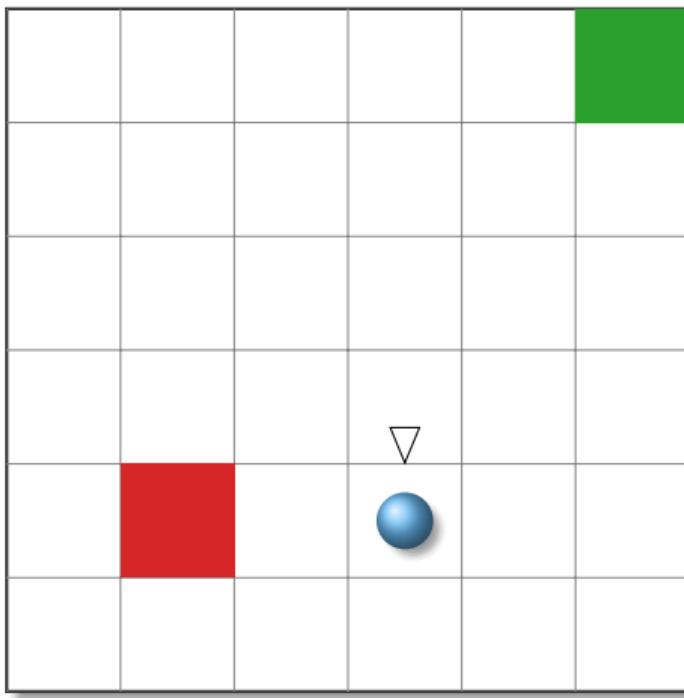
## standard tabular Q-Learning



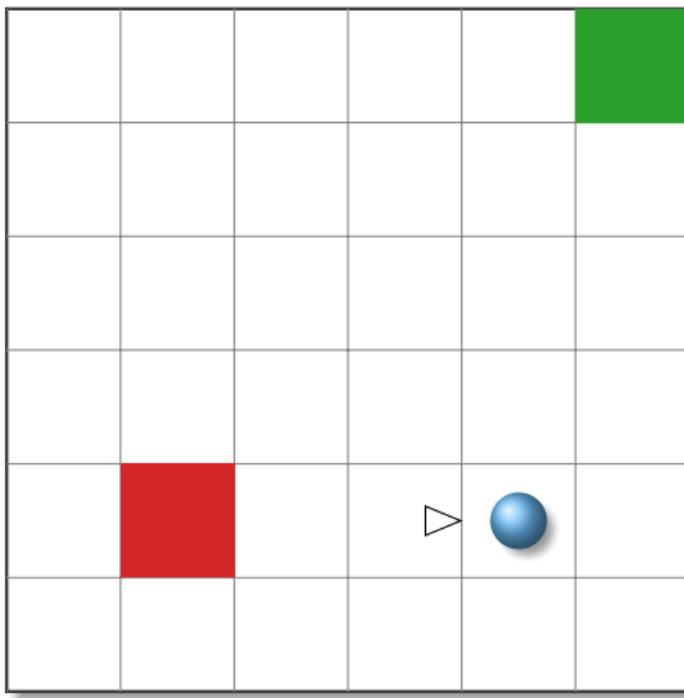
## standard tabular Q-Learning



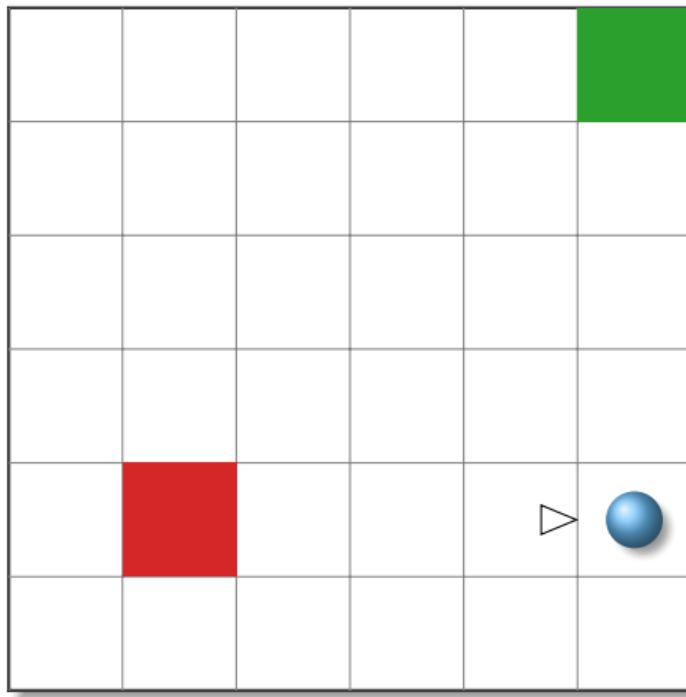
## standard tabular Q-Learning



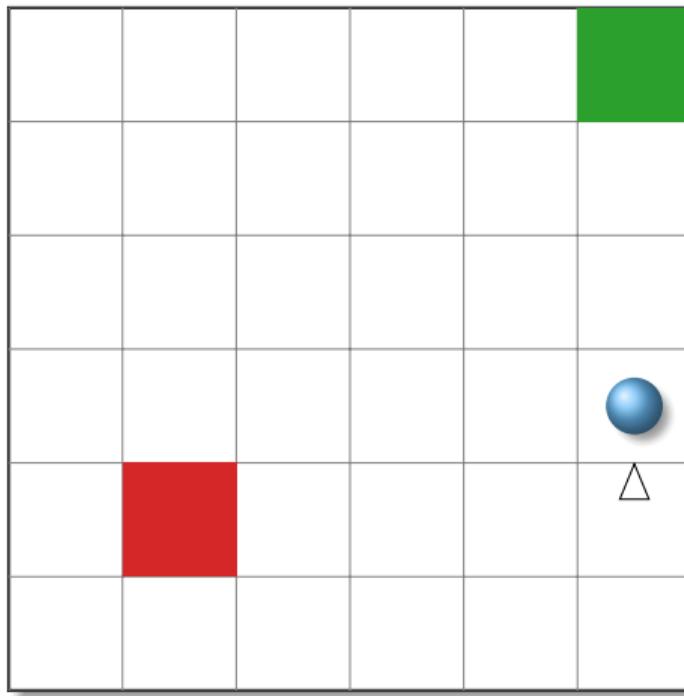
## standard tabular Q-Learning



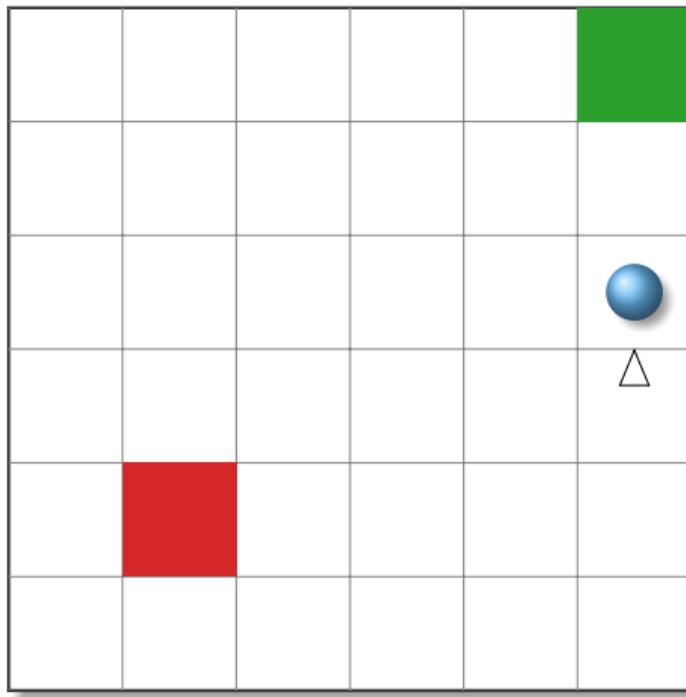
## standard tabular Q-Learning



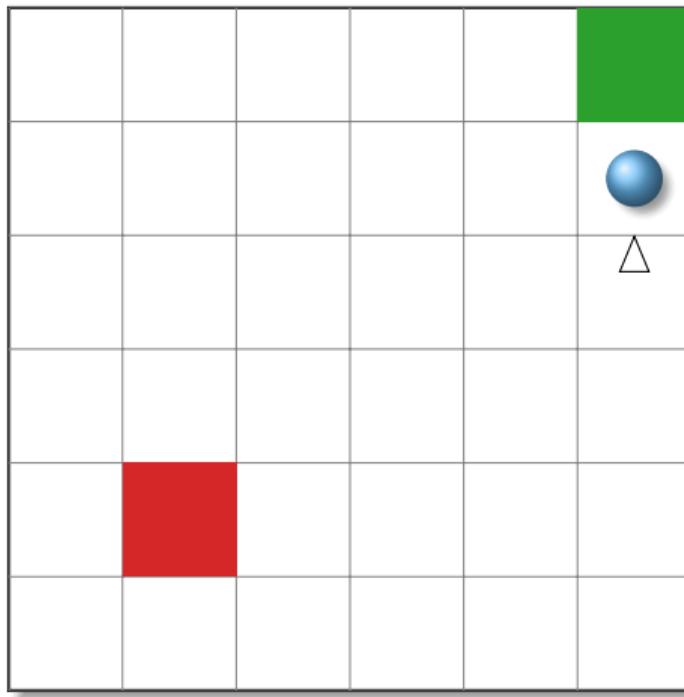
## standard tabular Q-Learning



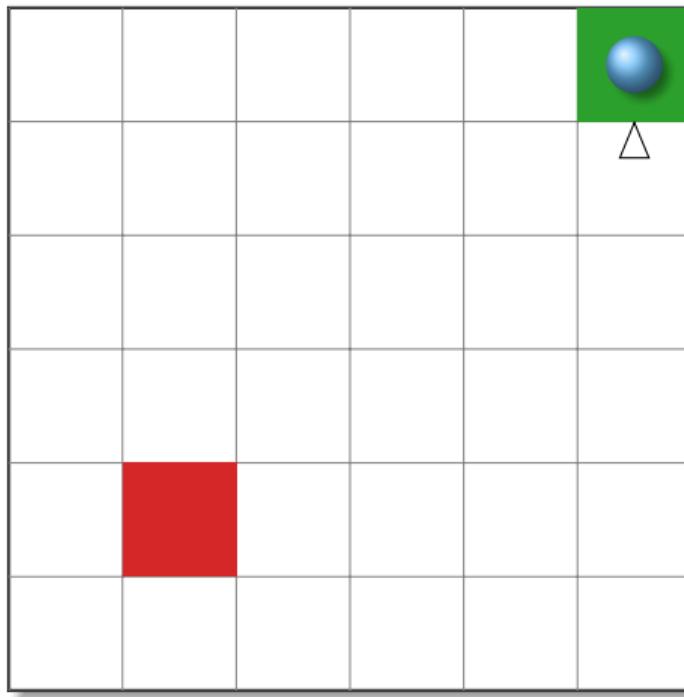
## standard tabular Q-Learning



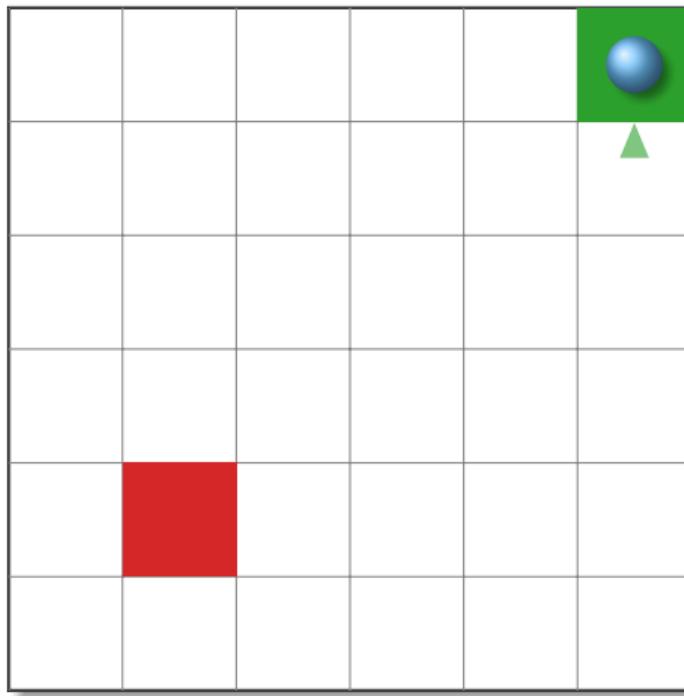
## standard tabular Q-Learning



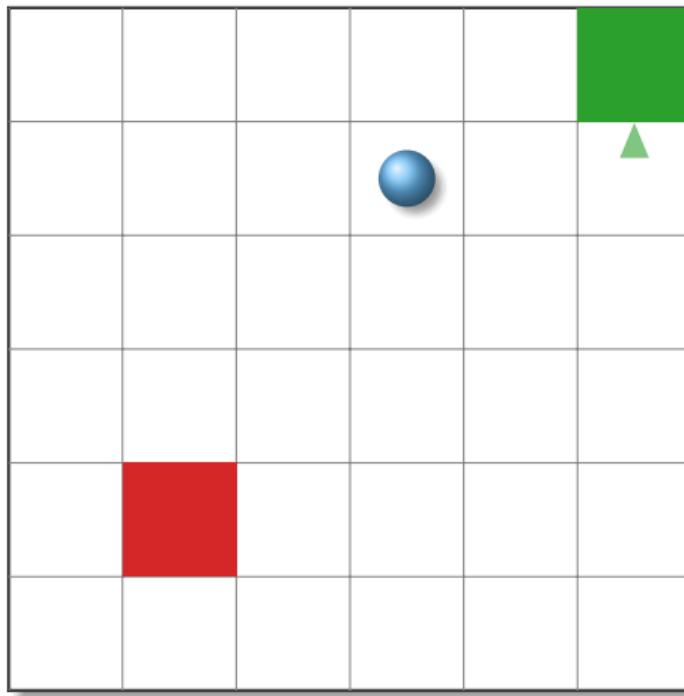
## standard tabular Q-Learning



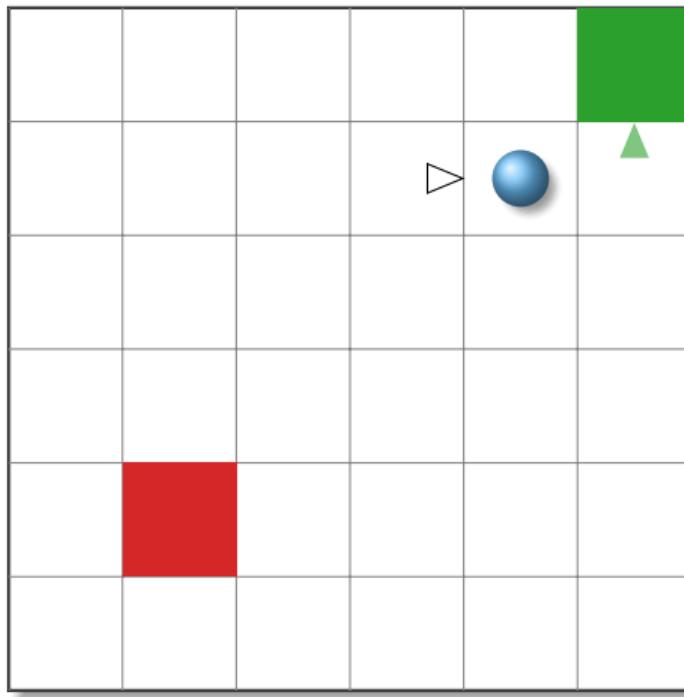
## standard tabular Q-Learning



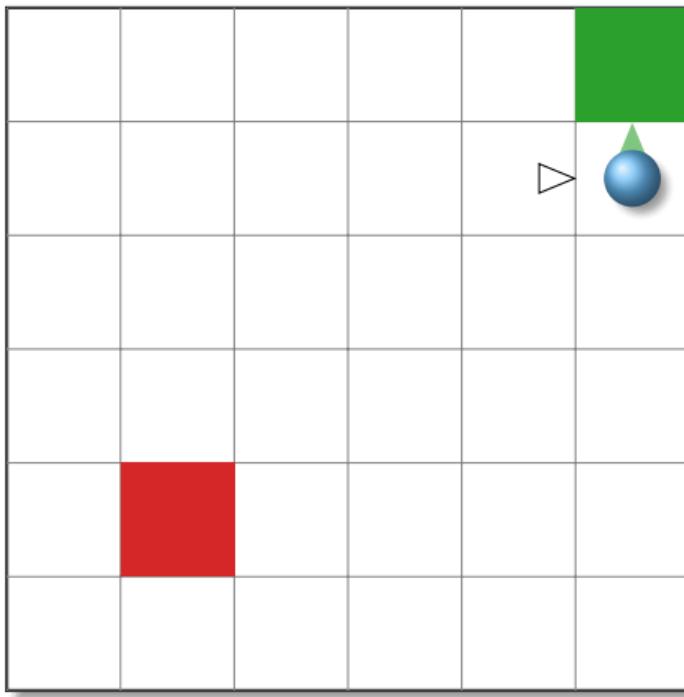
## standard tabular Q-Learning



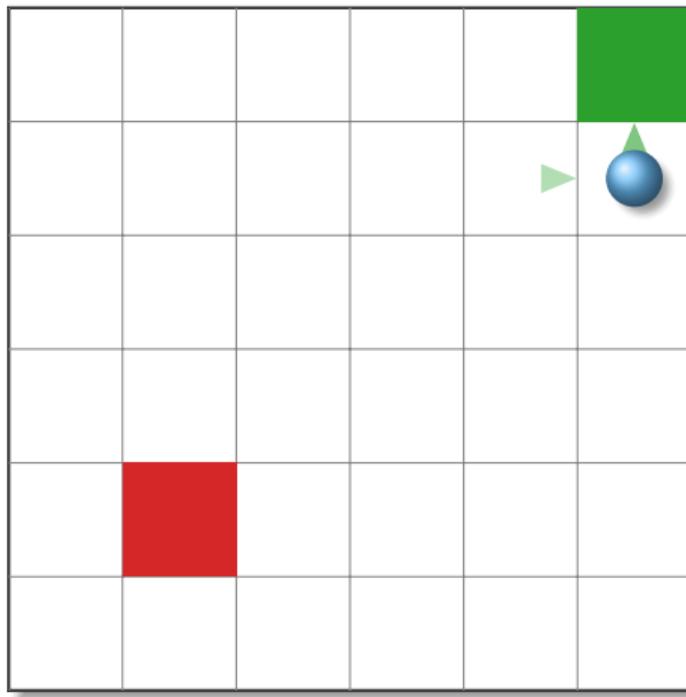
## standard tabular Q-Learning



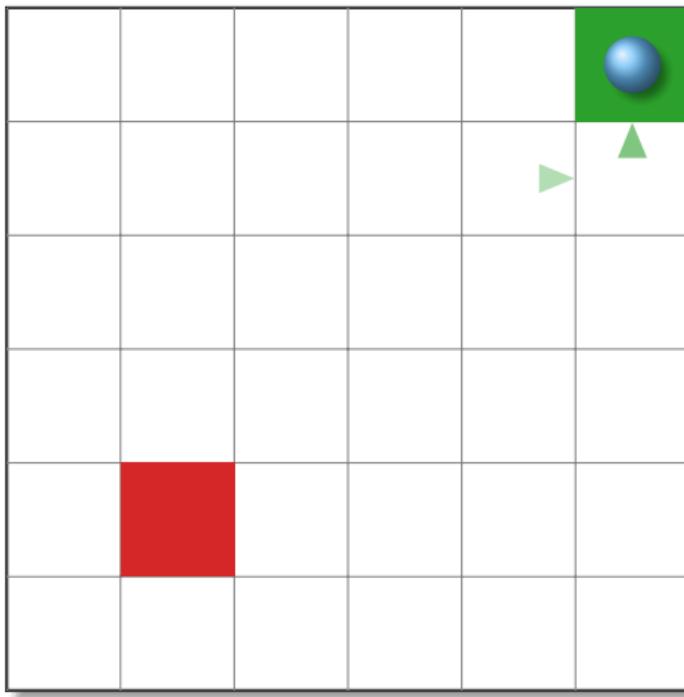
## standard tabular Q-Learning



## standard tabular Q-Learning



## standard tabular Q-Learning

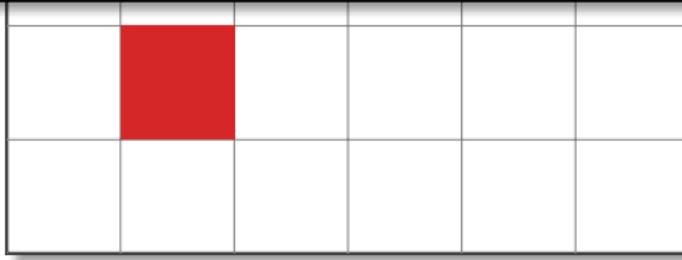


## standard tabular Q-Learning

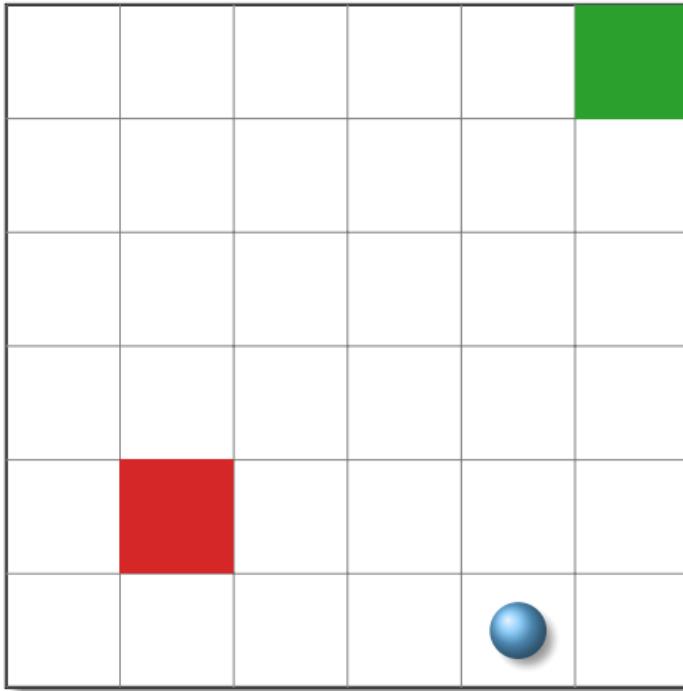


### standard Q-Learning

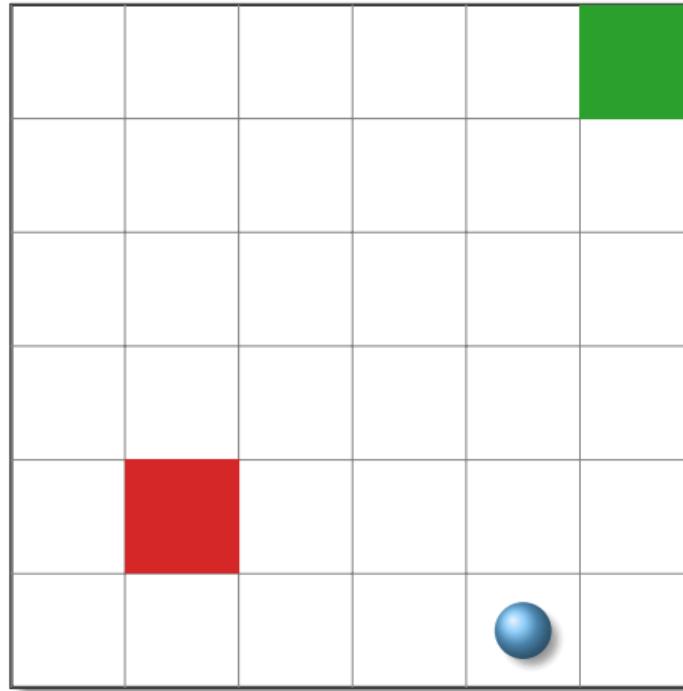
- + Convergence
- + **Minimal memory/computation**
- Sample inefficiency



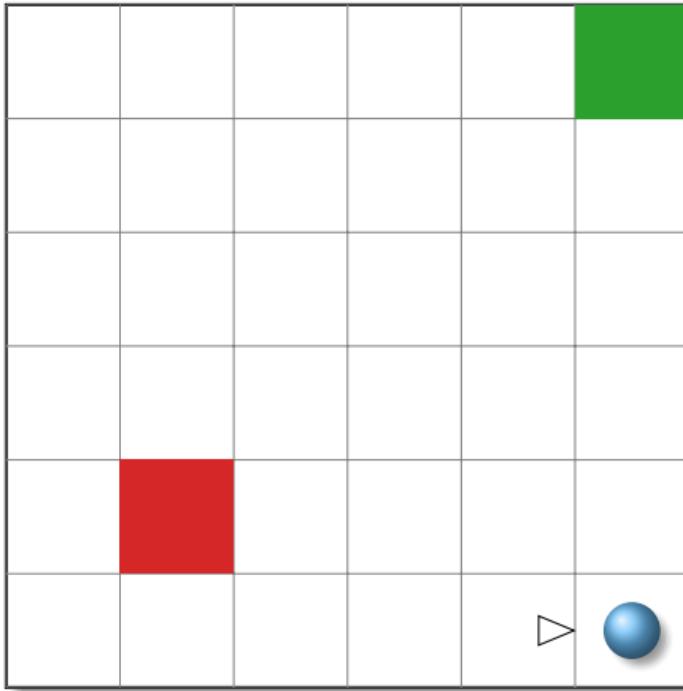
**with replay memory**



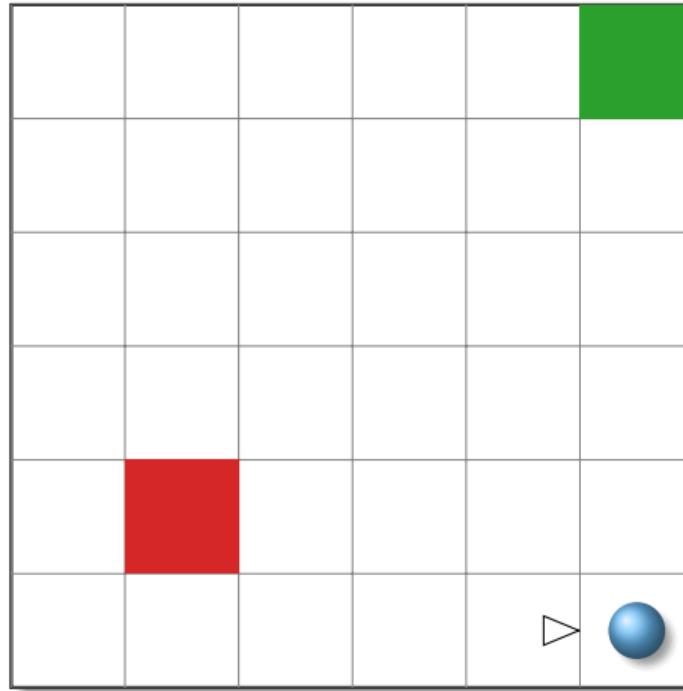
**standard Q-Learning**



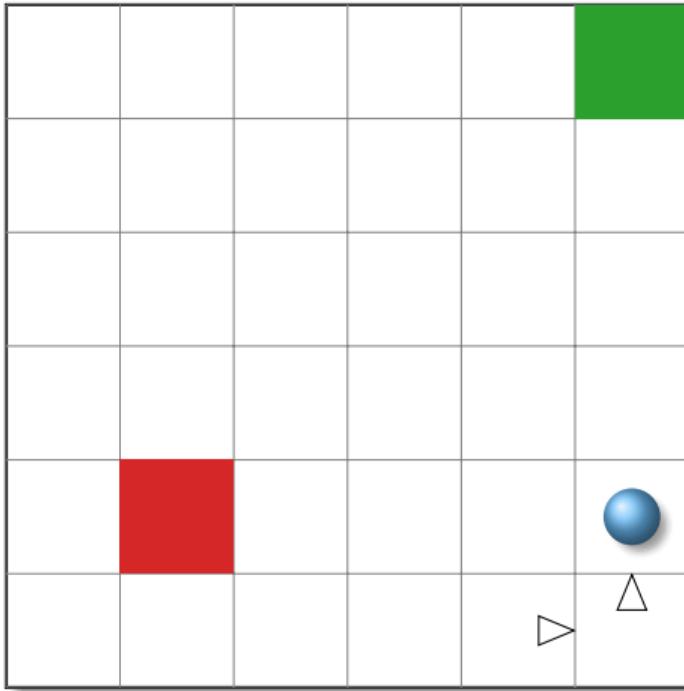
**with replay memory**



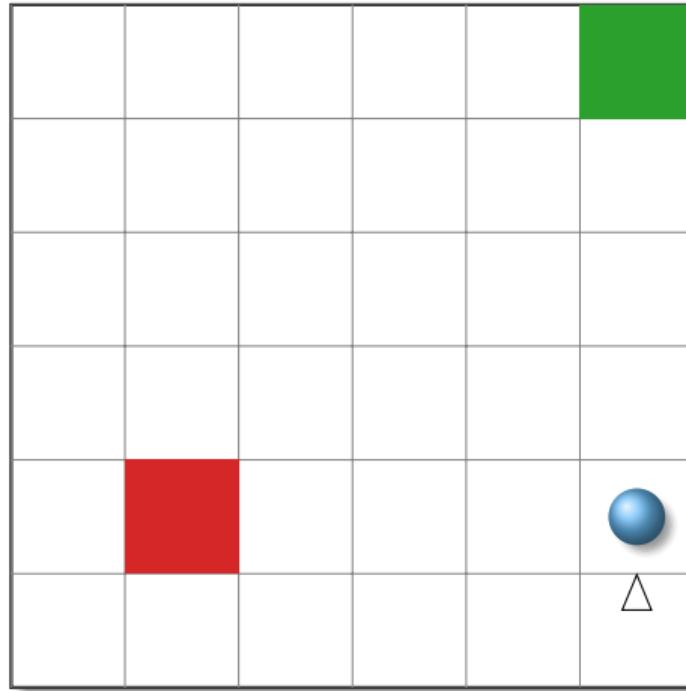
**standard Q-Learning**



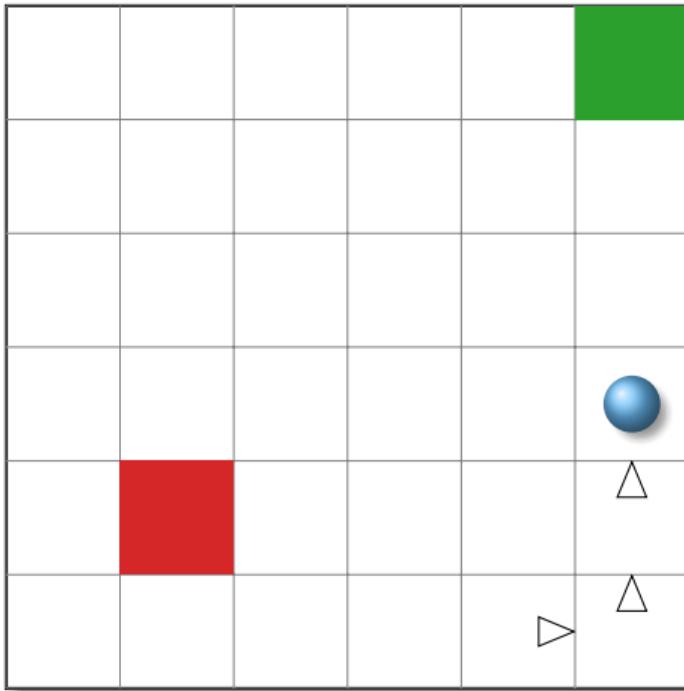
**with replay memory**



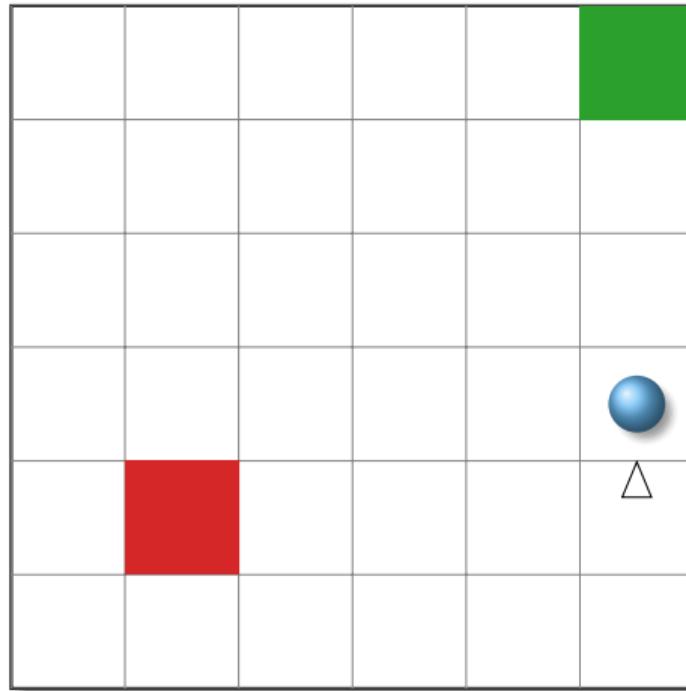
**standard Q-Learning**



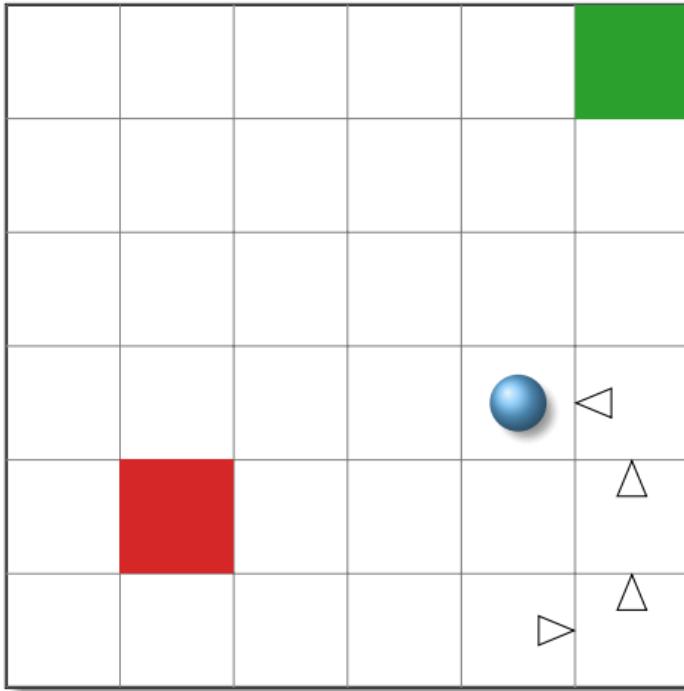
**with replay memory**



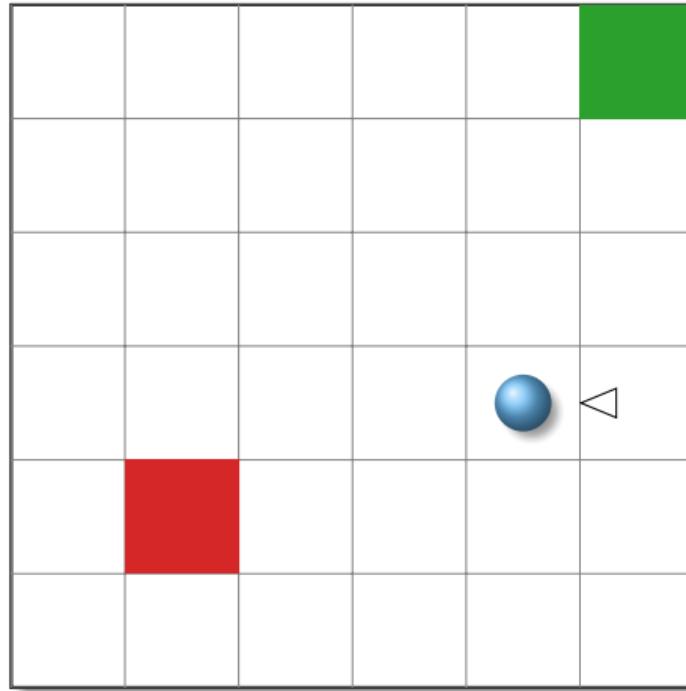
**standard Q-Learning**



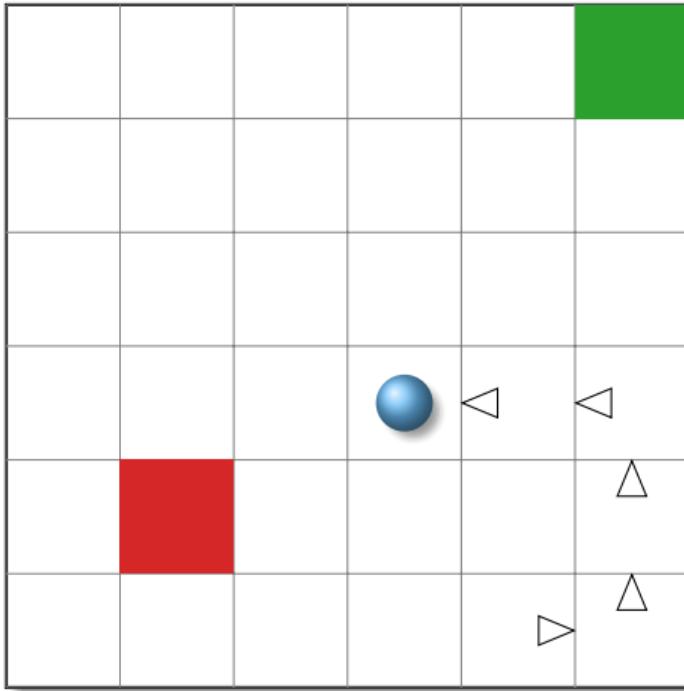
**with replay memory**



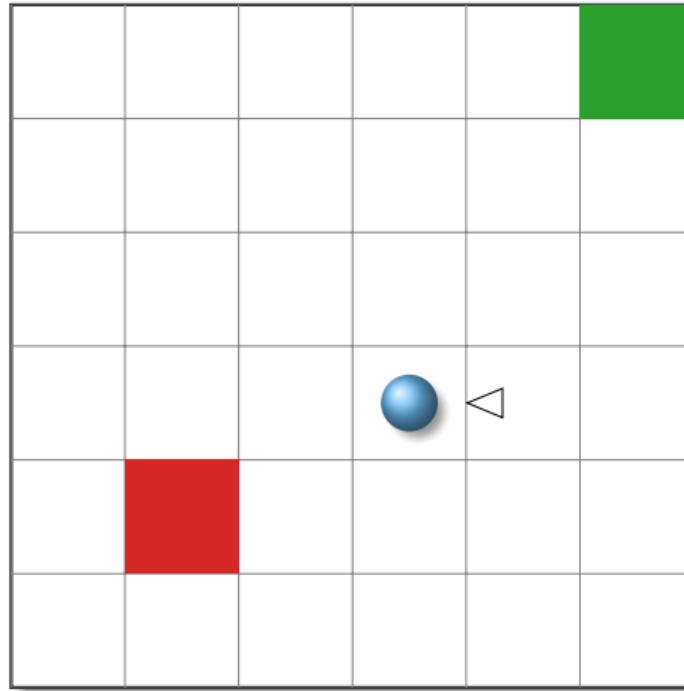
**standard Q-Learning**



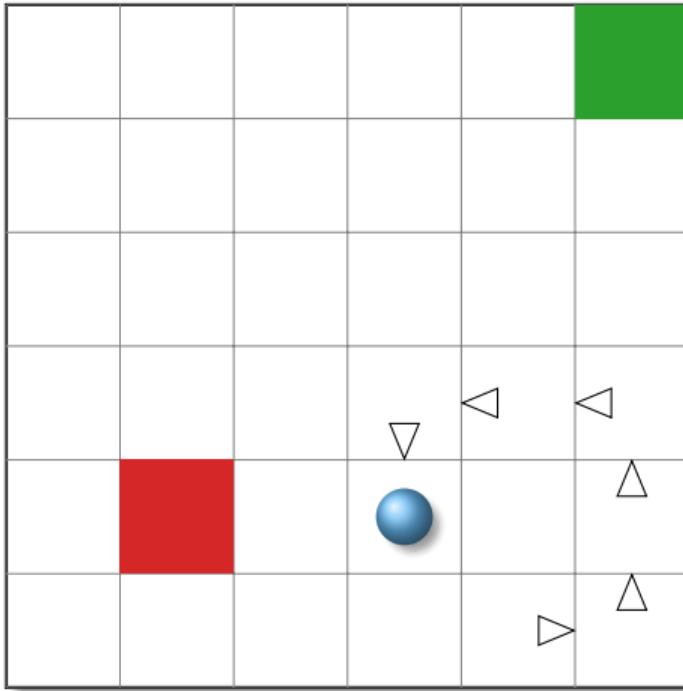
**with replay memory**



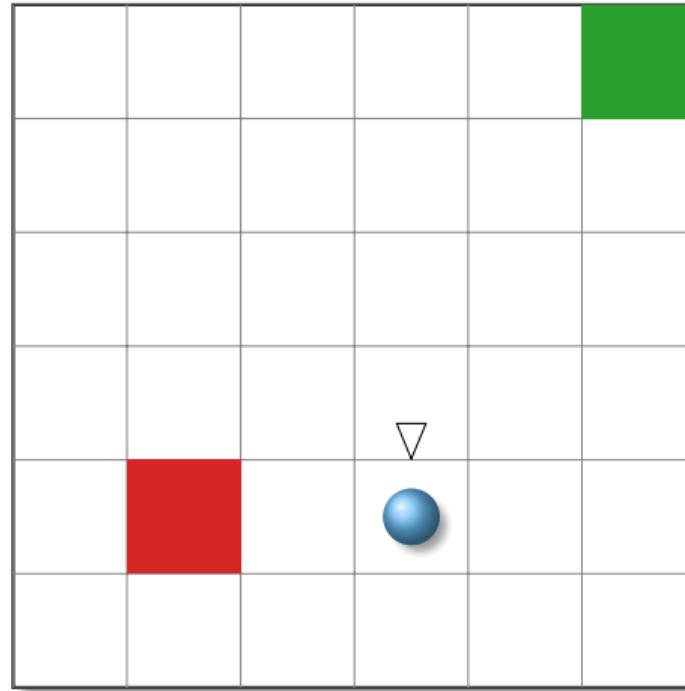
**standard Q-Learning**



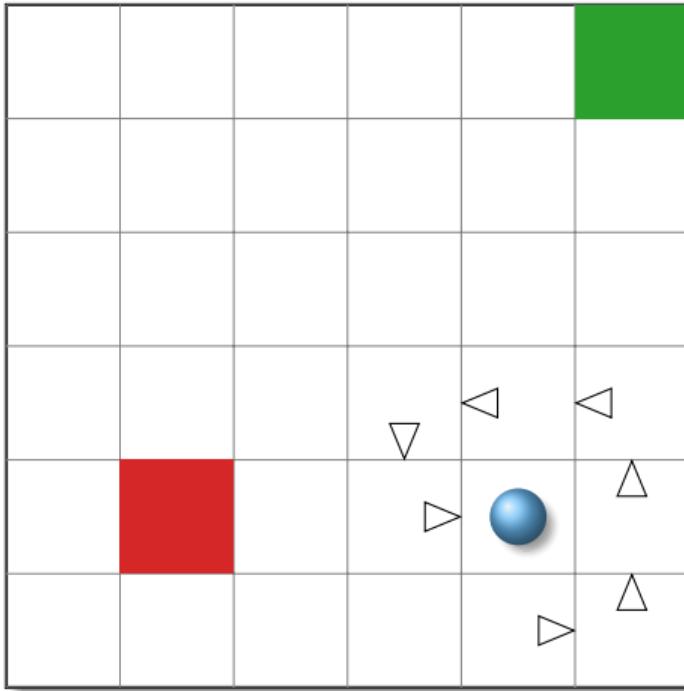
**with replay memory**



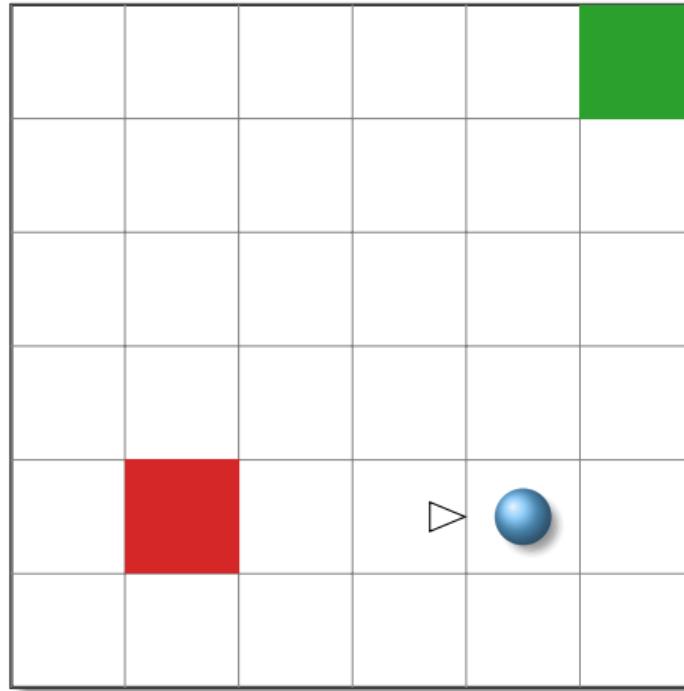
**standard Q-Learning**



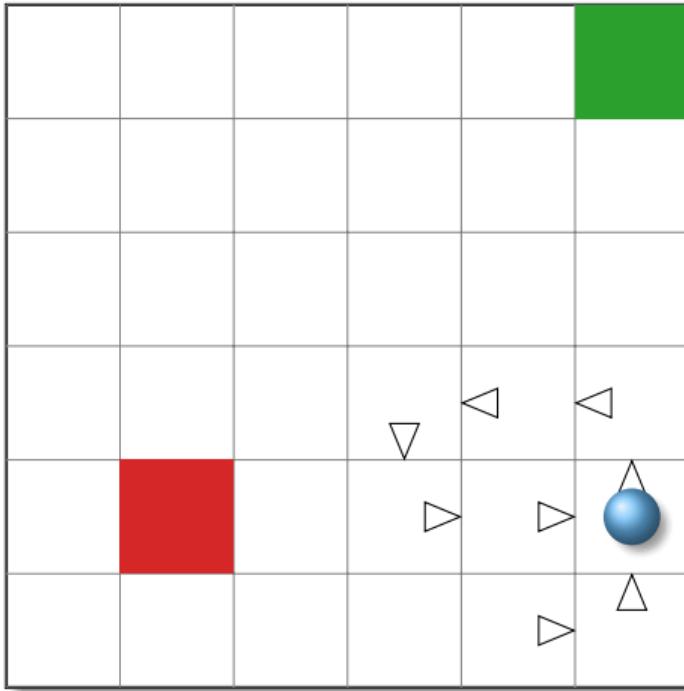
**with replay memory**



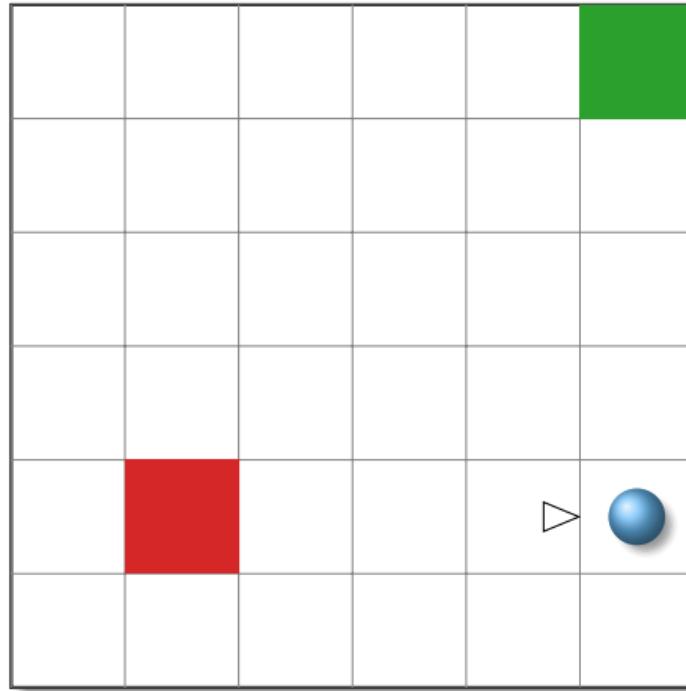
**standard Q-Learning**



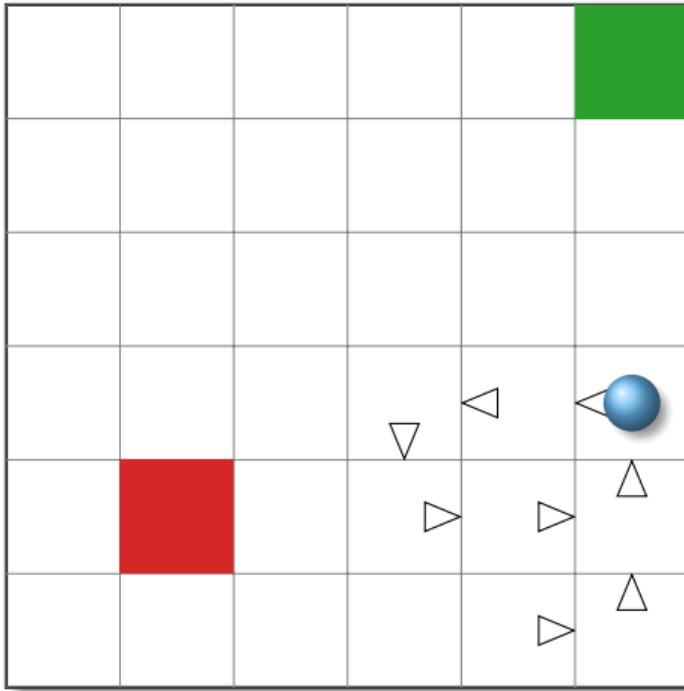
**with replay memory**



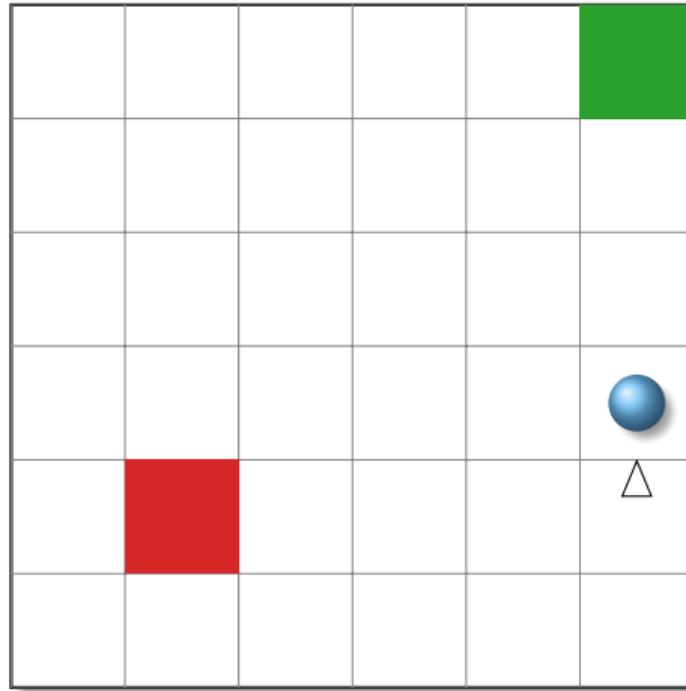
**standard Q-Learning**



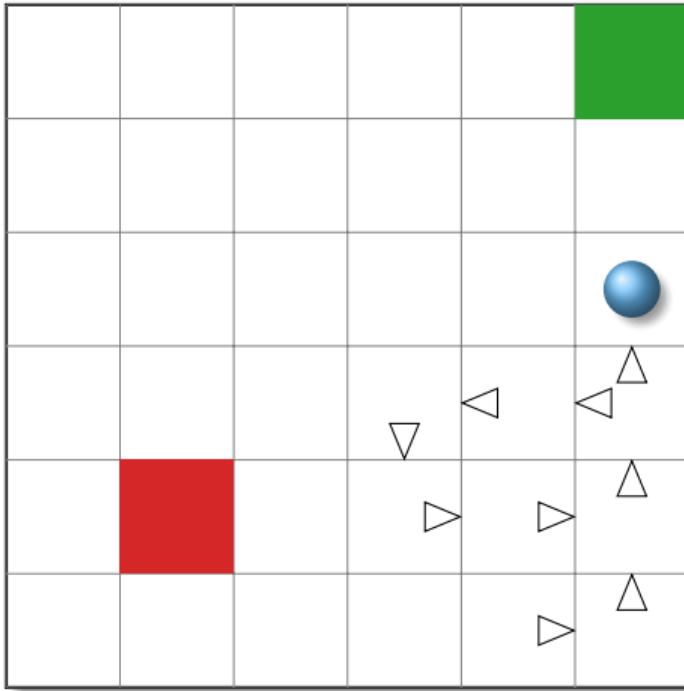
**with replay memory**



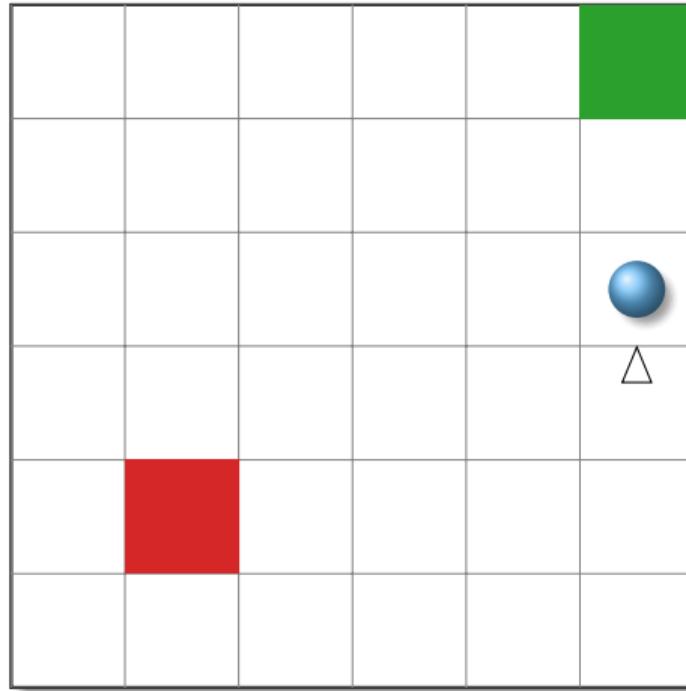
**standard Q-Learning**



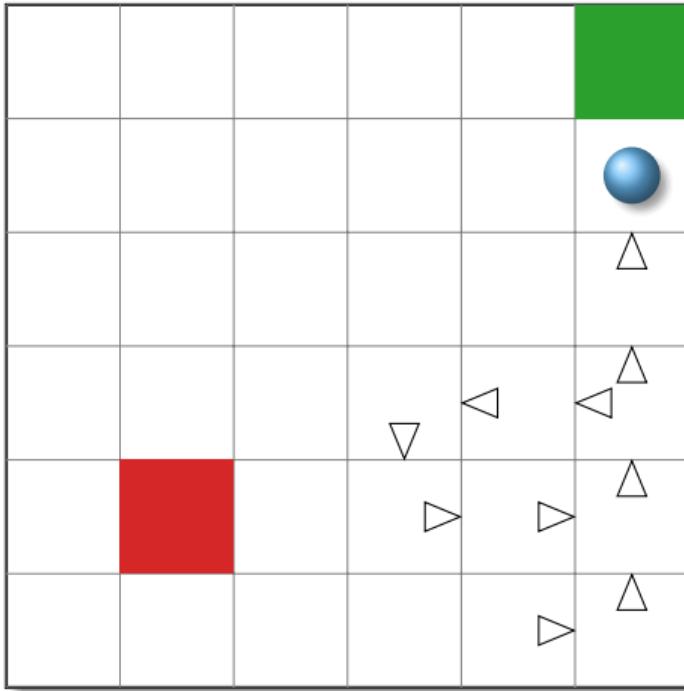
**with replay memory**



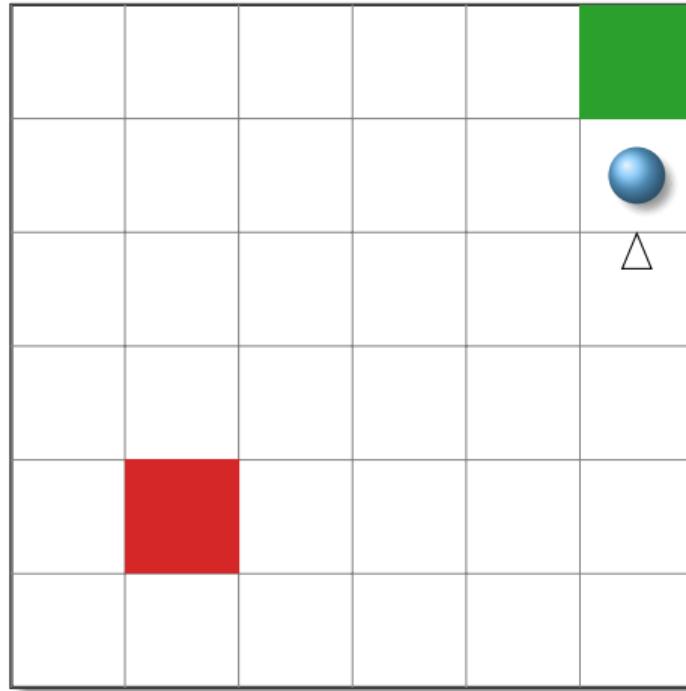
**standard Q-Learning**



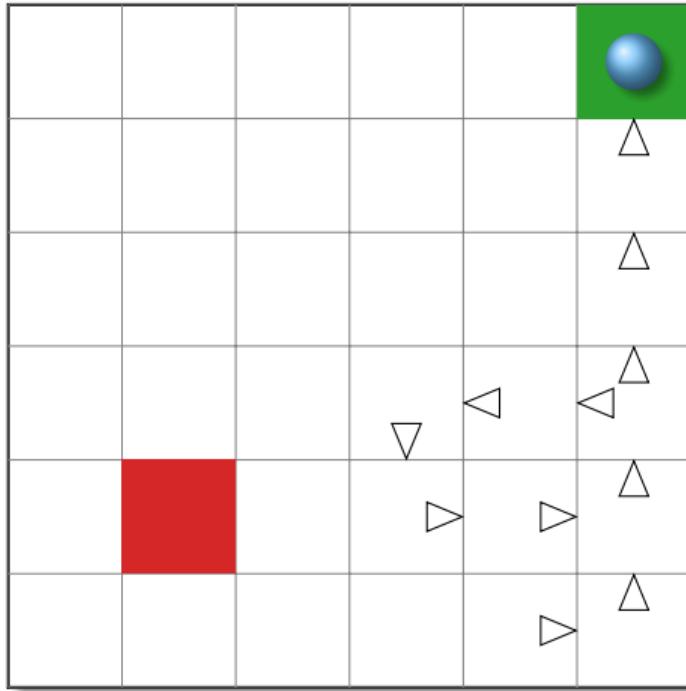
**with replay memory**



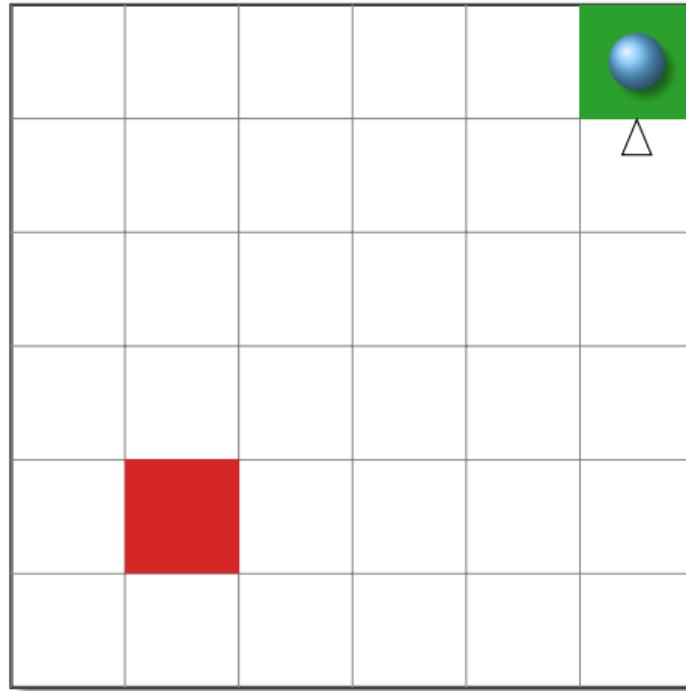
**standard Q-Learning**



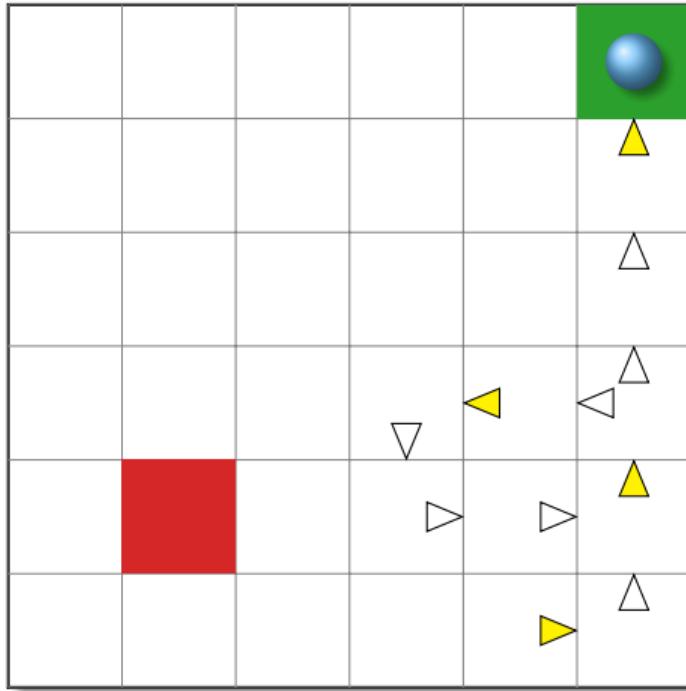
**with replay memory**



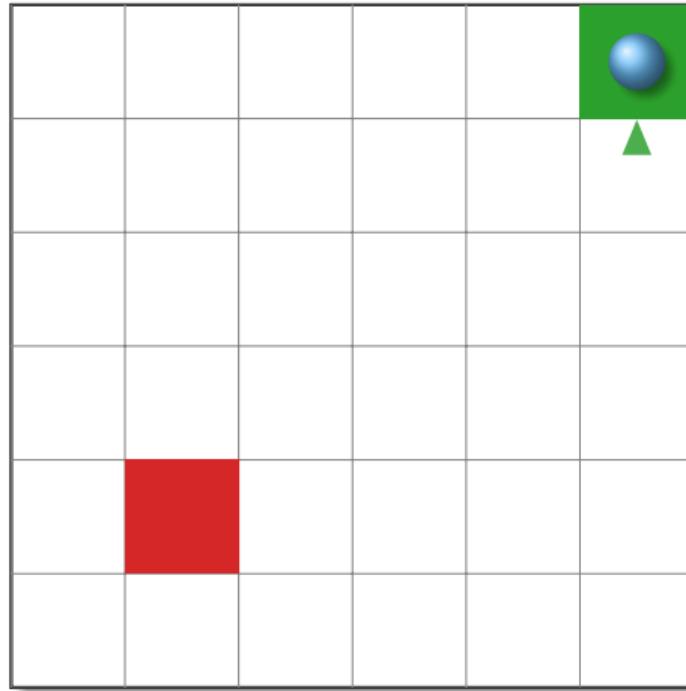
**standard Q-Learning**



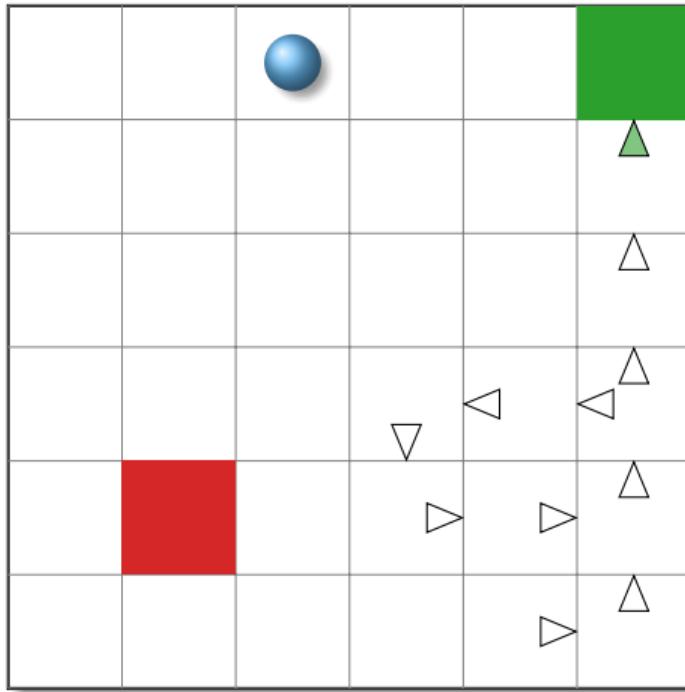
**with replay memory**



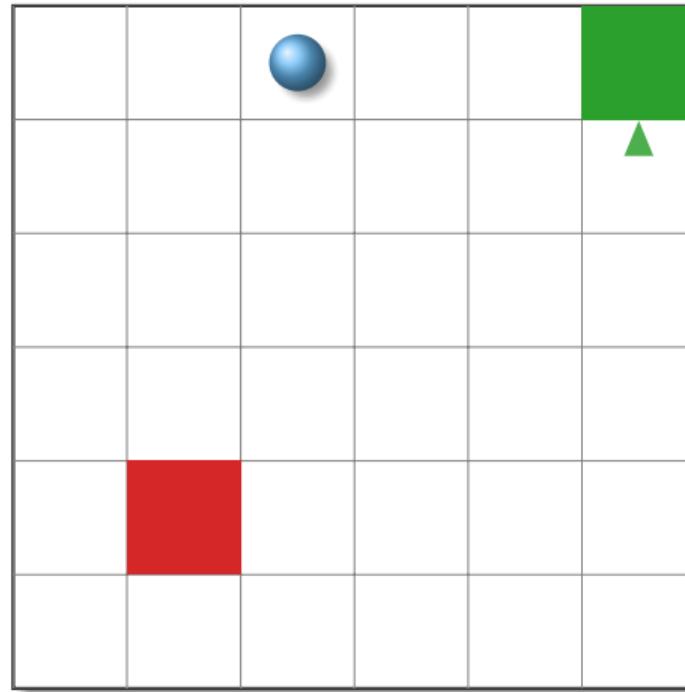
**standard Q-Learning**



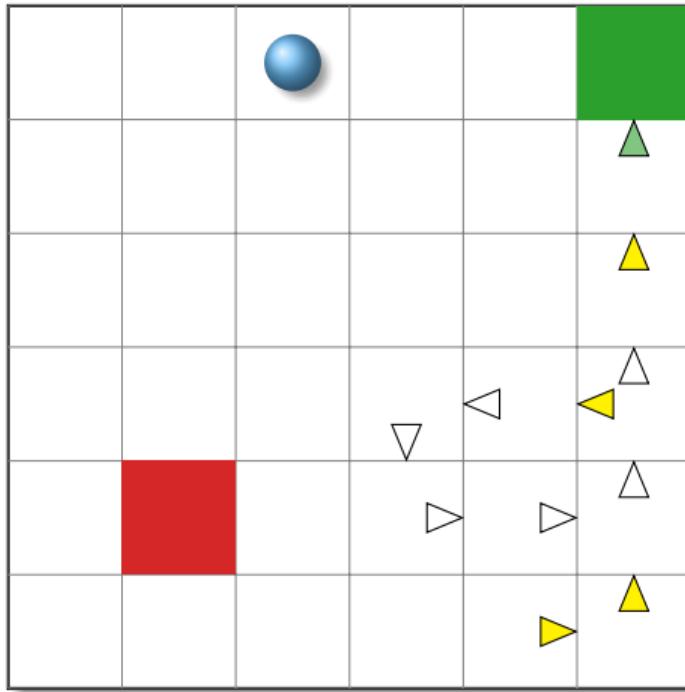
**with replay memory**



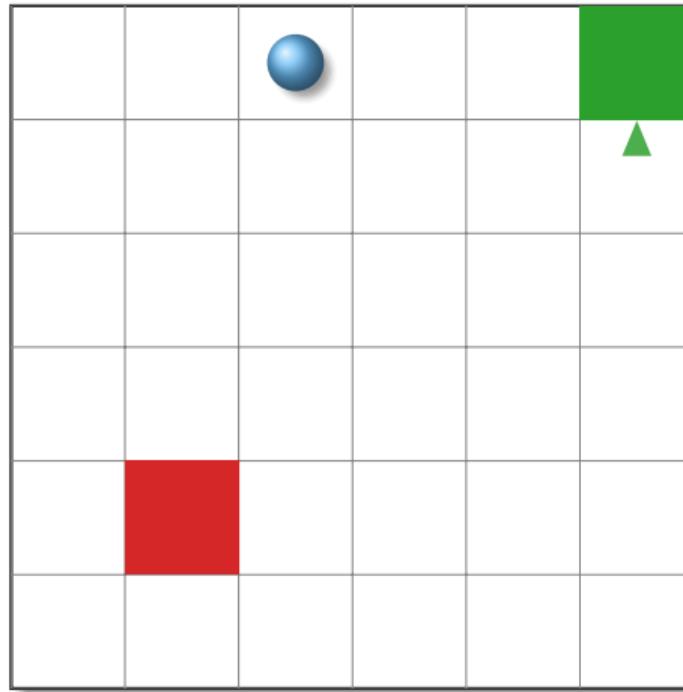
**standard Q-Learning**



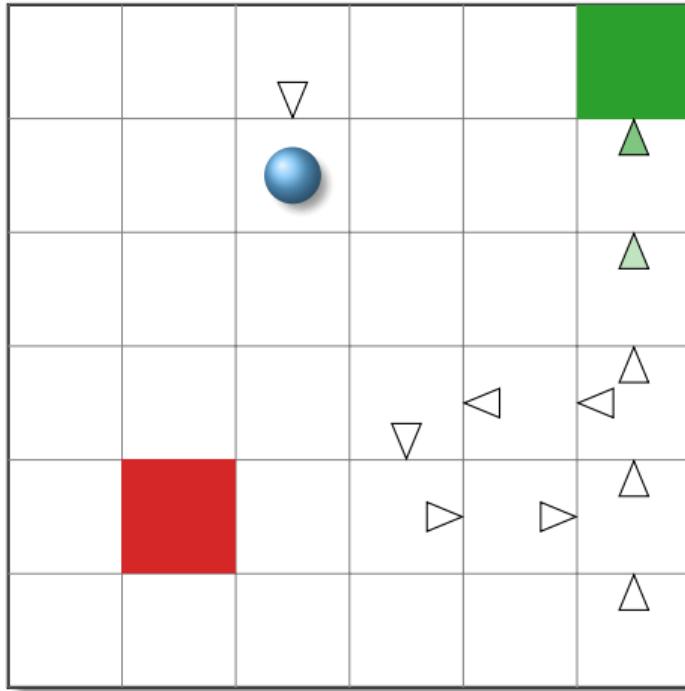
**with replay memory**



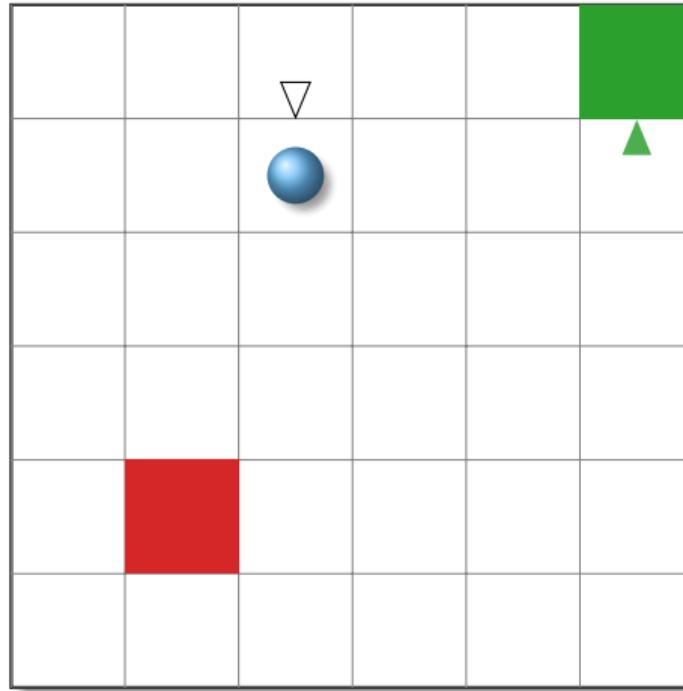
**standard Q-Learning**



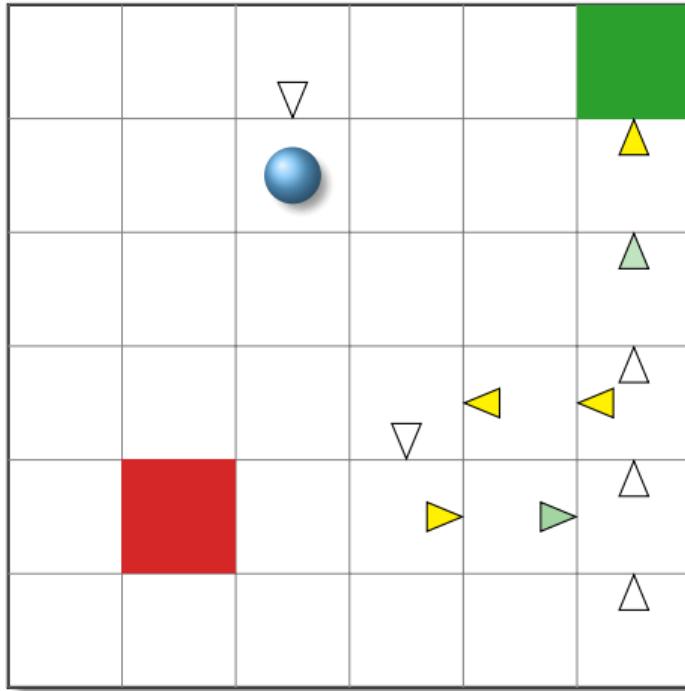
**with replay memory**



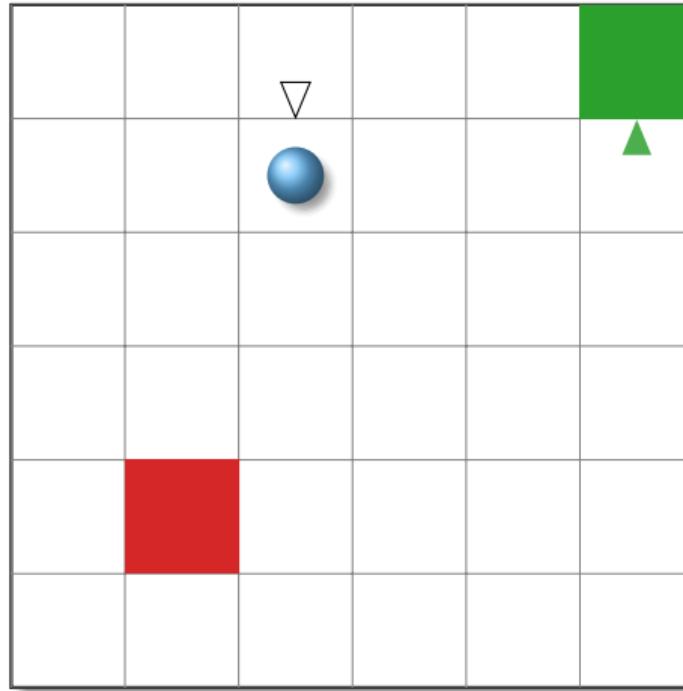
**standard Q-Learning**



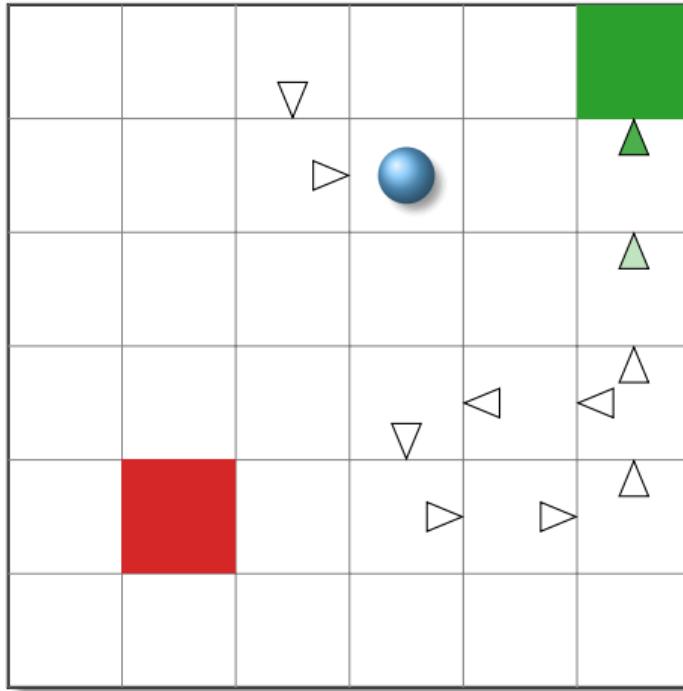
**with replay memory**



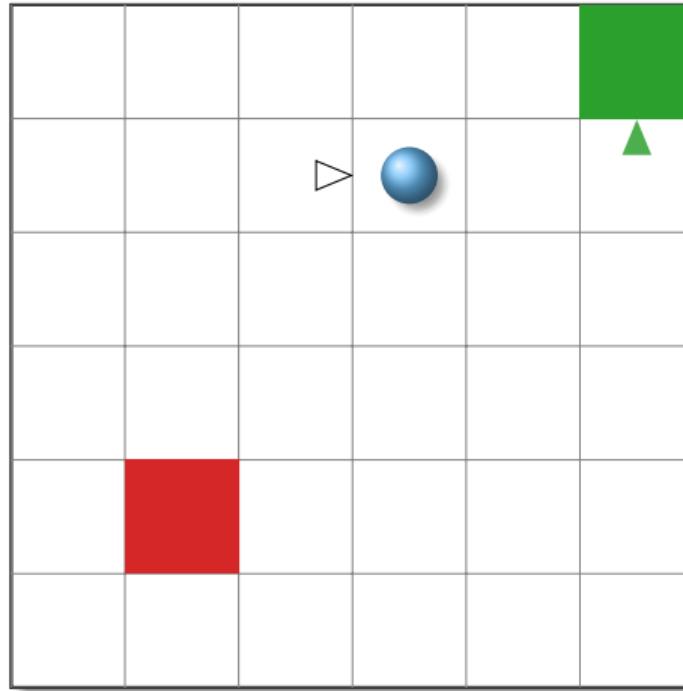
**standard Q-Learning**



**with replay memory**



**standard Q-Learning**

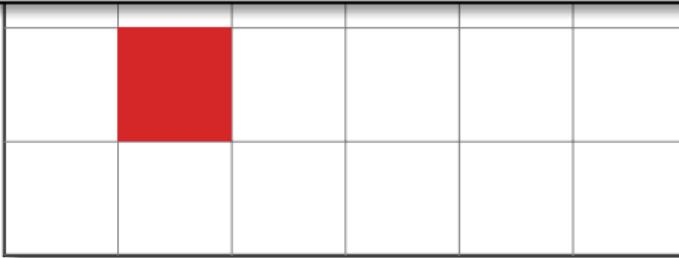


## with replay memory



### with replay memory

- Inefficient updates
- Needs more memory/computation
- + Sample efficiency

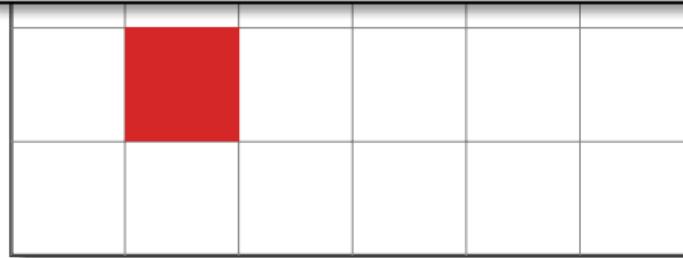


## standard Q-Learning



### standard Q-Learning

- + Convergence
- + **Minimal memory/computation**
- Sample inefficiency



## Can we do better?

Yes.

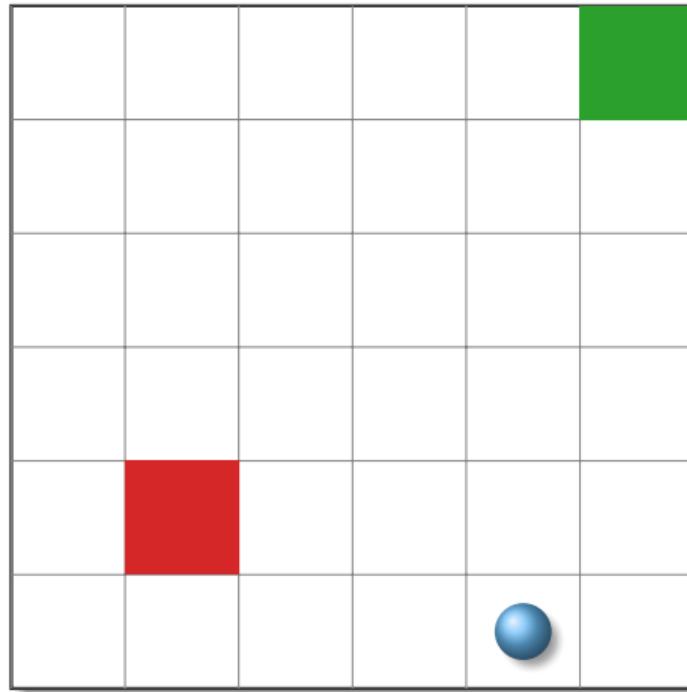
1. Organize memory in table, i.e.

$$\text{estimate } P_{s \rightarrow s'}^a = N_{s \rightarrow s'}^a / N_s^a$$

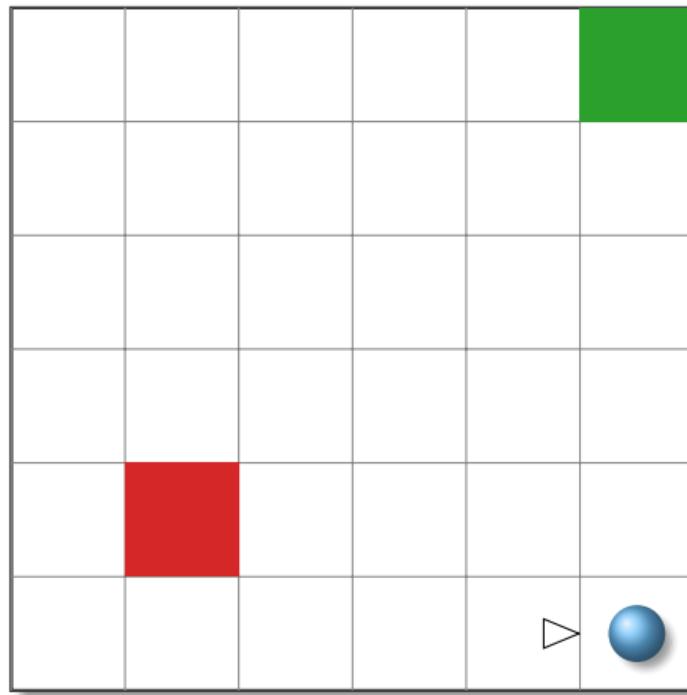
with counts  $N_{s \rightarrow s'}^a$  and  $N_s^a$

2. Prioritize backups cleverly.

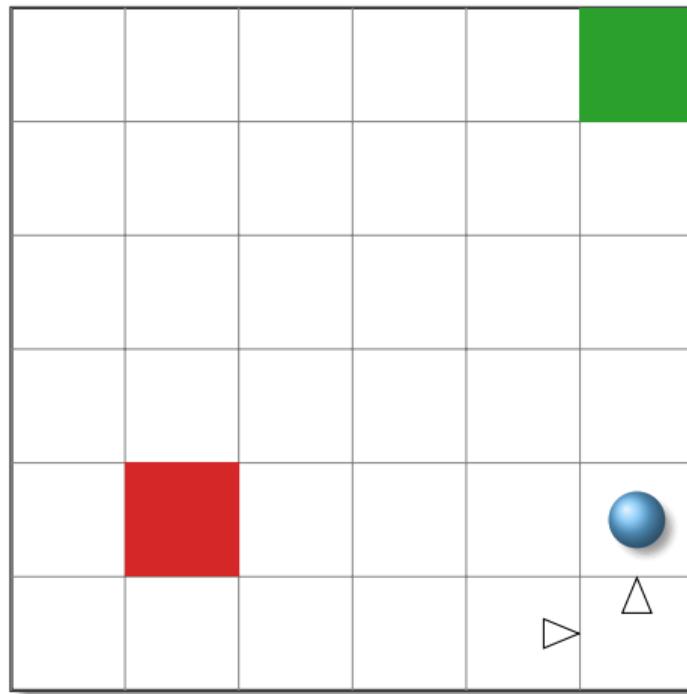
## prioritized sweeping with small backups



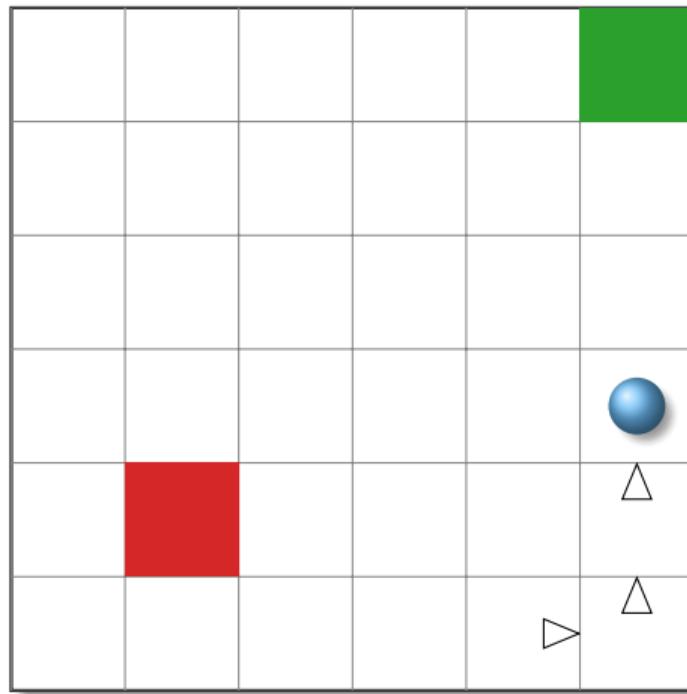
## prioritized sweeping with small backups



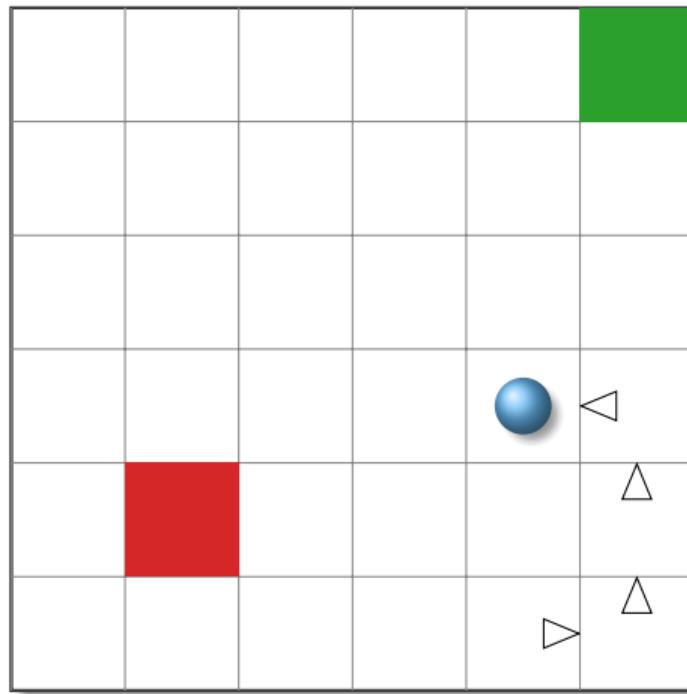
## prioritized sweeping with small backups



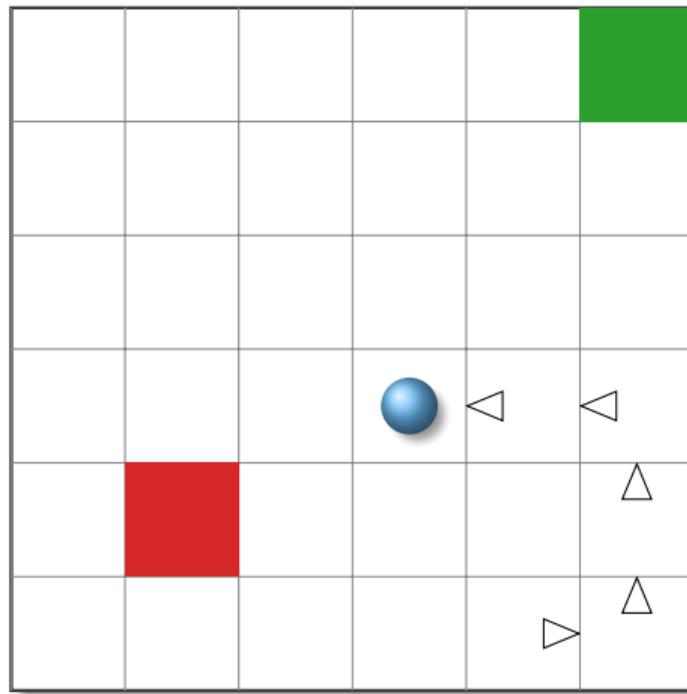
## prioritized sweeping with small backups



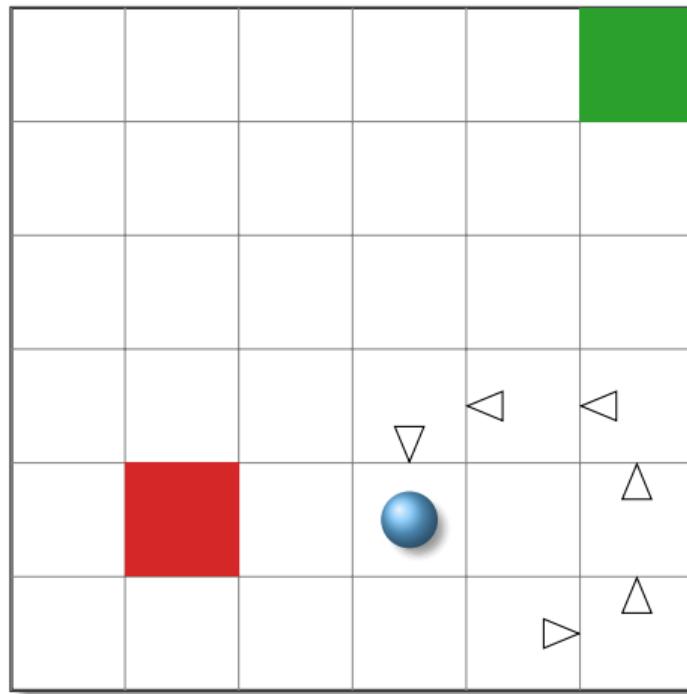
## prioritized sweeping with small backups



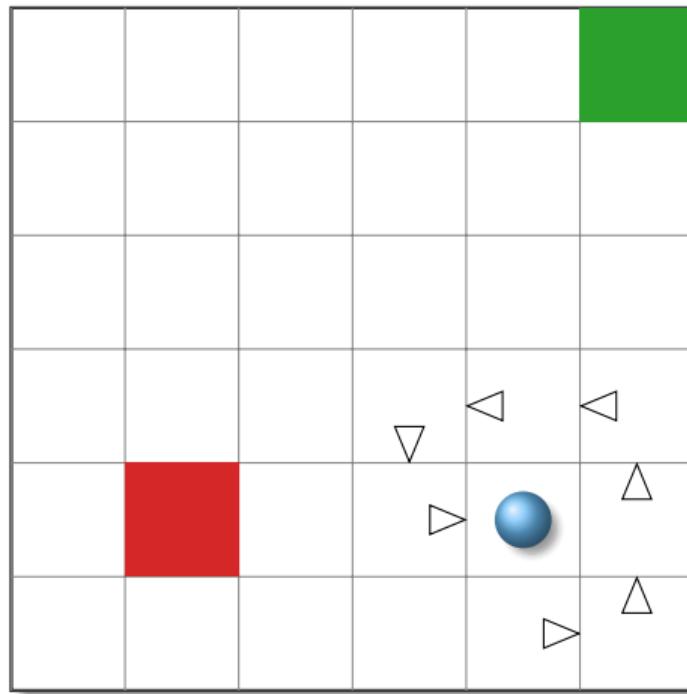
## prioritized sweeping with small backups



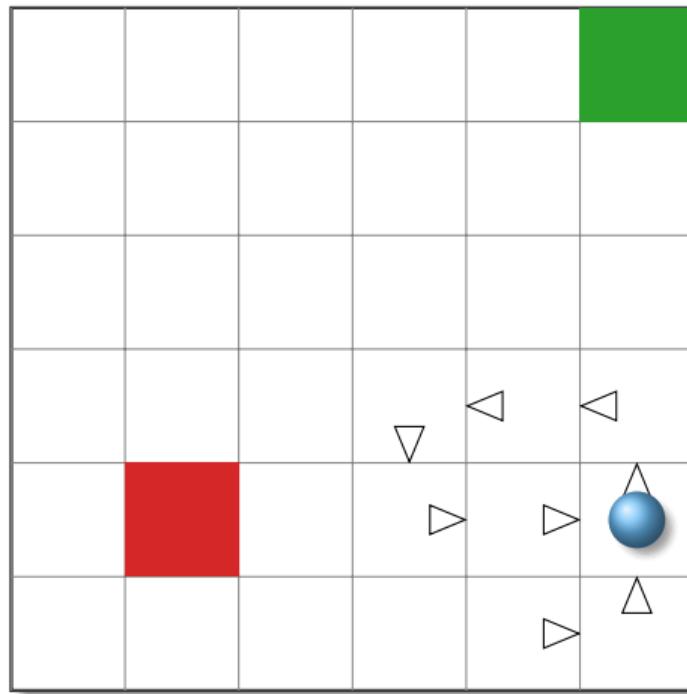
## prioritized sweeping with small backups



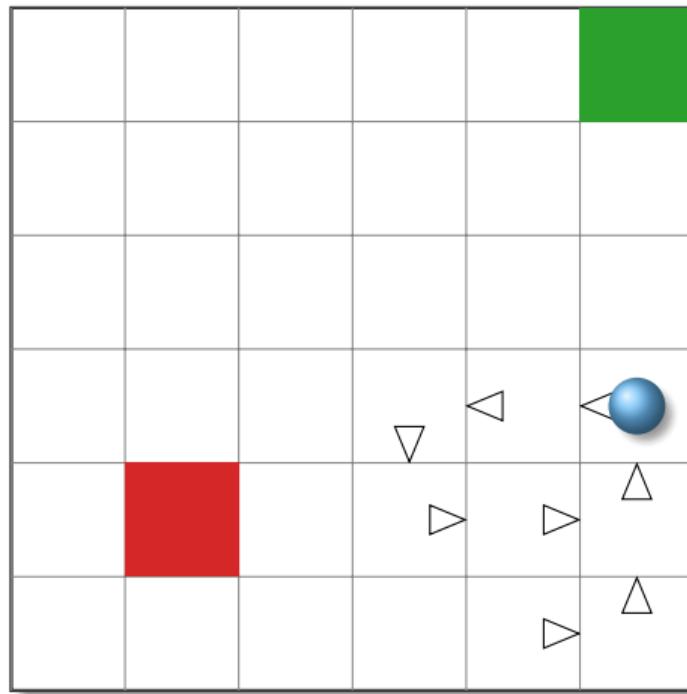
## prioritized sweeping with small backups



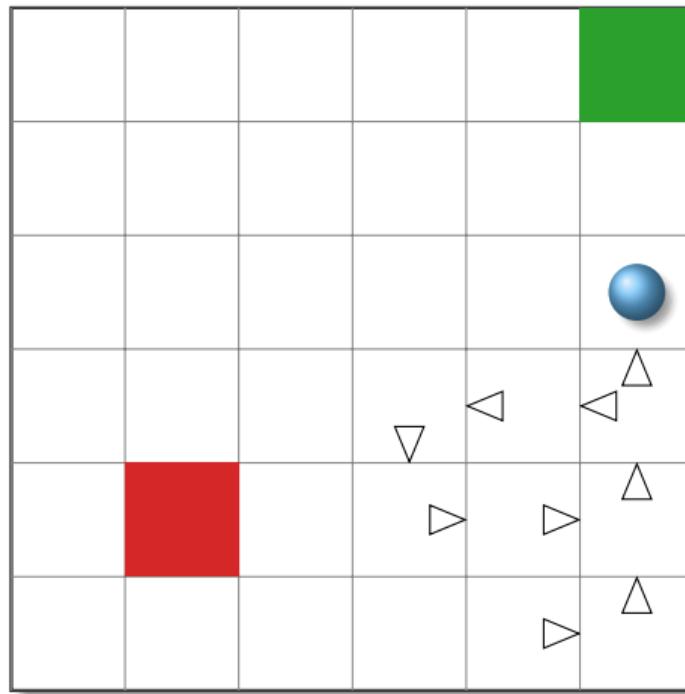
## prioritized sweeping with small backups



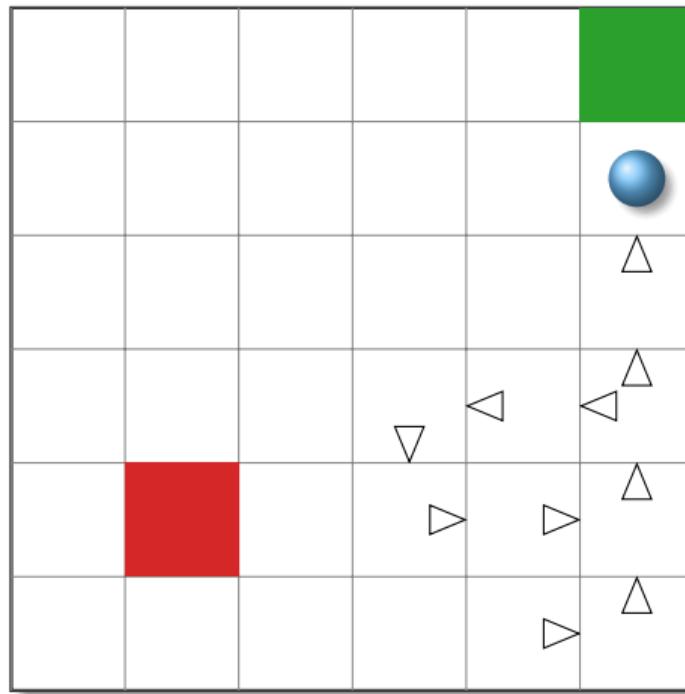
## prioritized sweeping with small backups



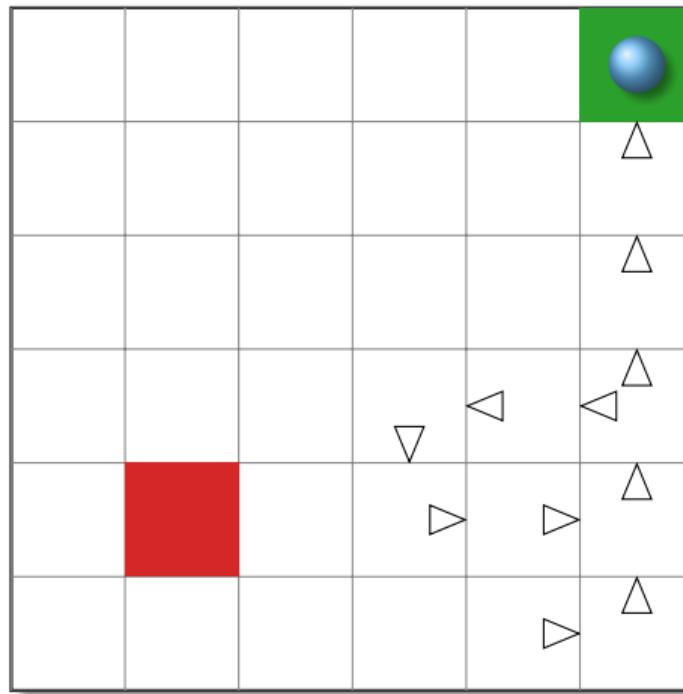
## prioritized sweeping with small backups



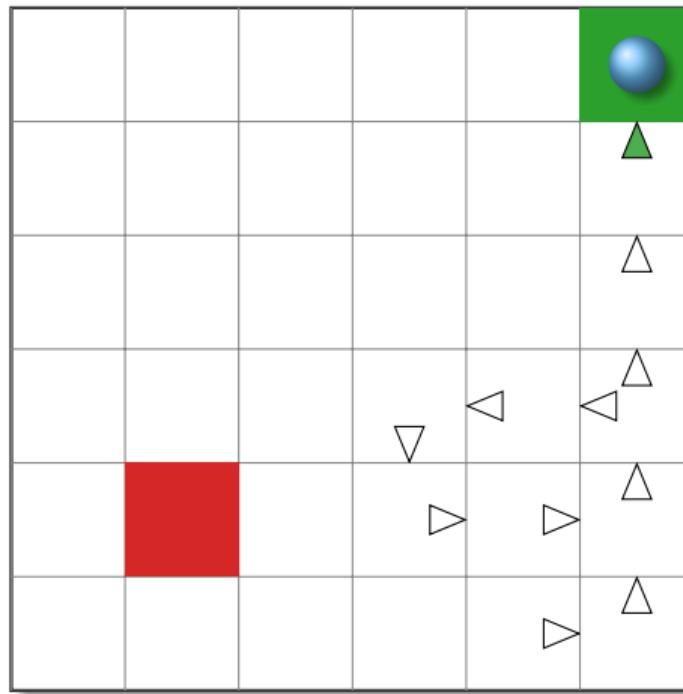
## prioritized sweeping with small backups



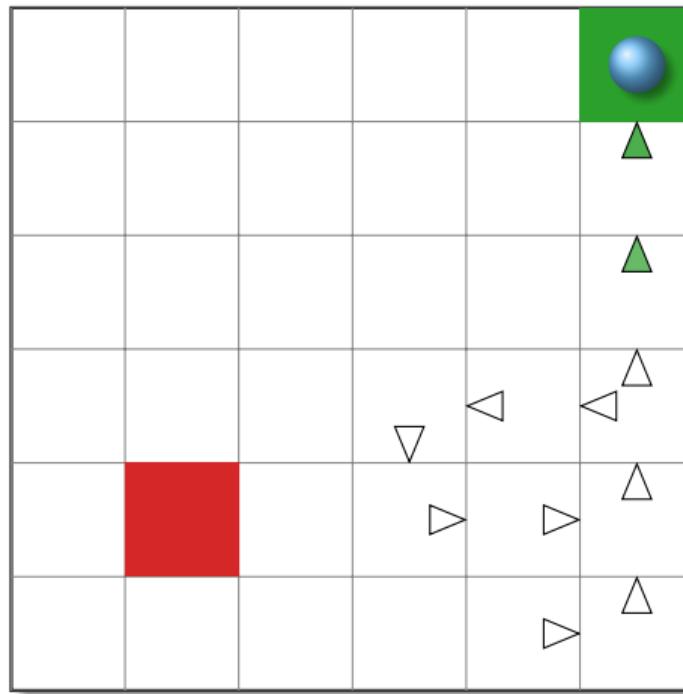
## prioritized sweeping with small backups



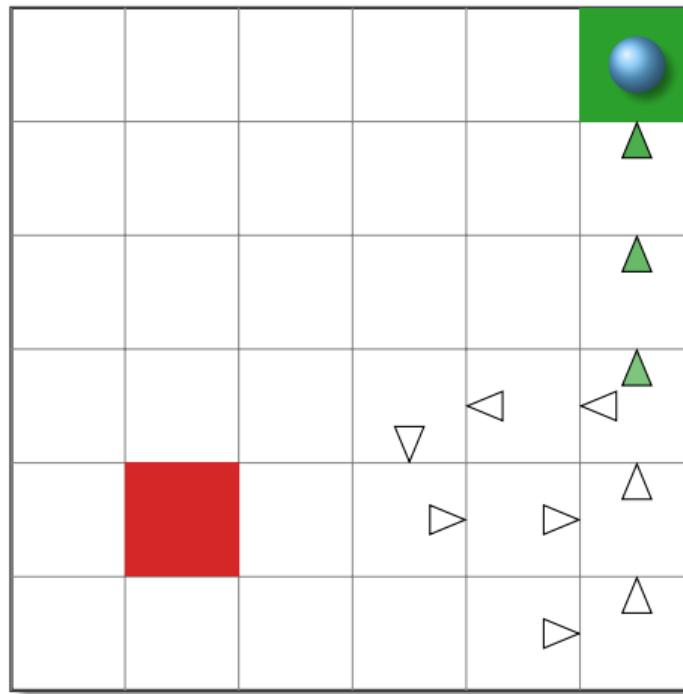
## prioritized sweeping with small backups



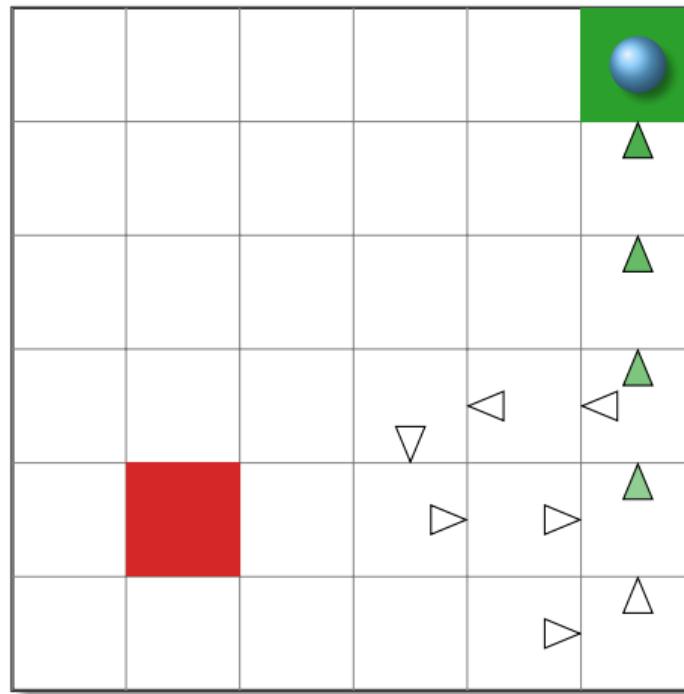
## prioritized sweeping with small backups



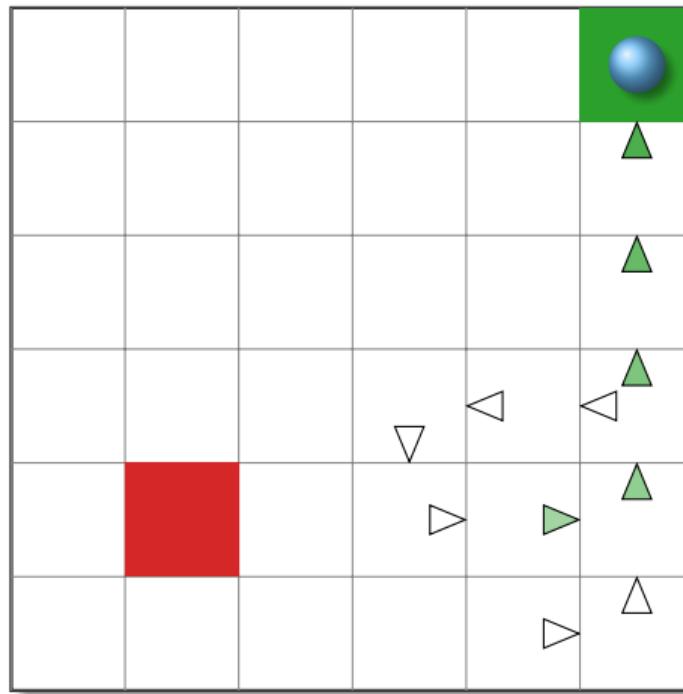
## prioritized sweeping with small backups



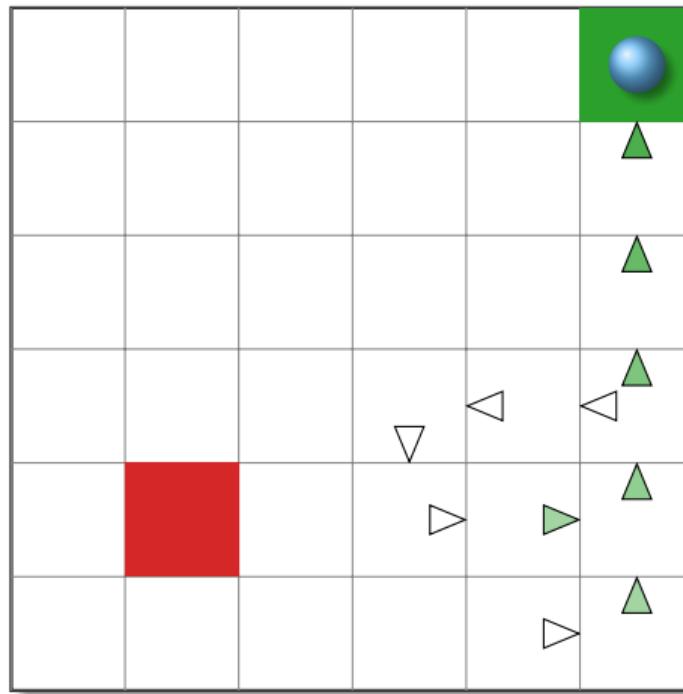
## prioritized sweeping with small backups



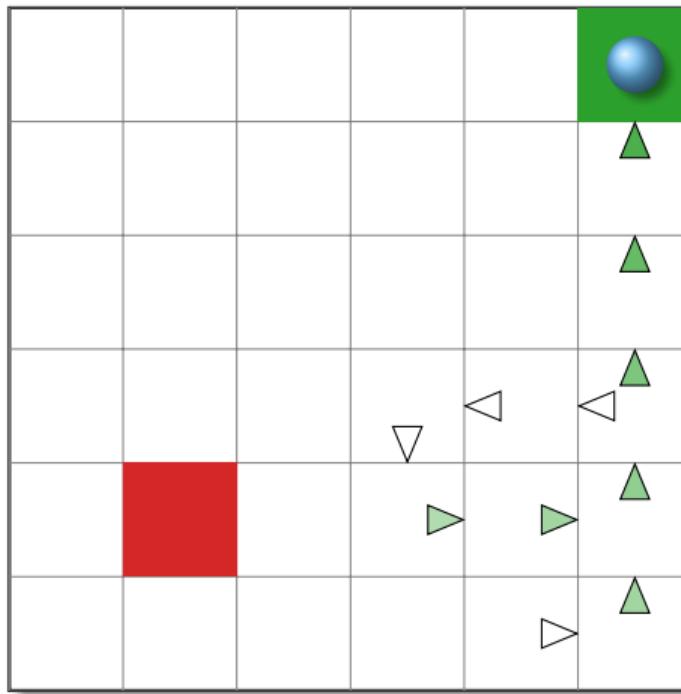
## prioritized sweeping with small backups



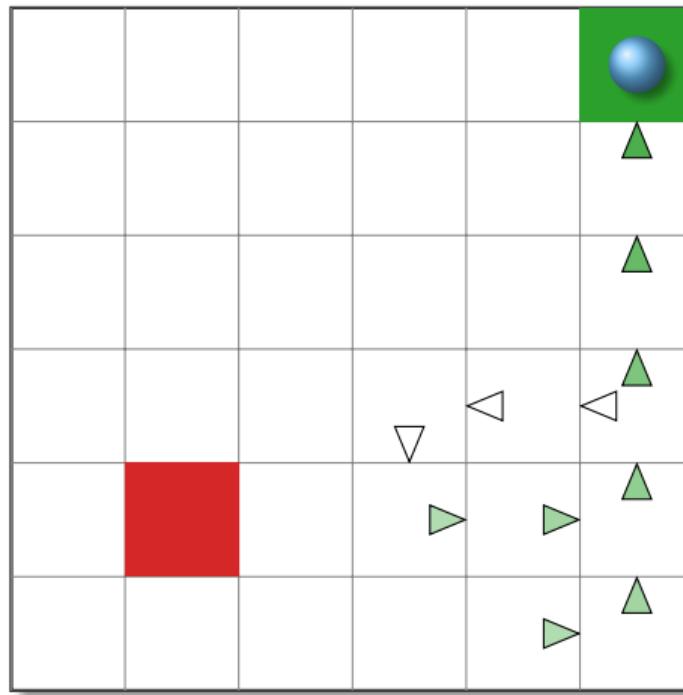
## prioritized sweeping with small backups



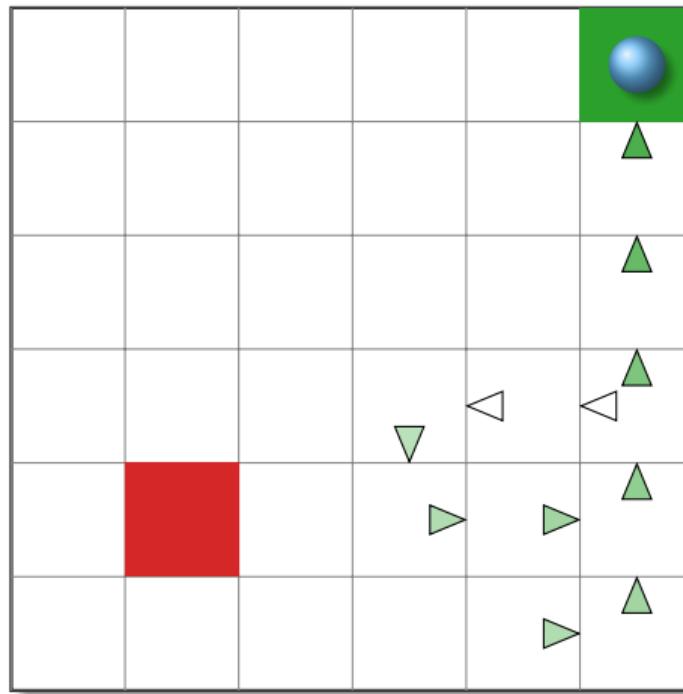
## prioritized sweeping with small backups



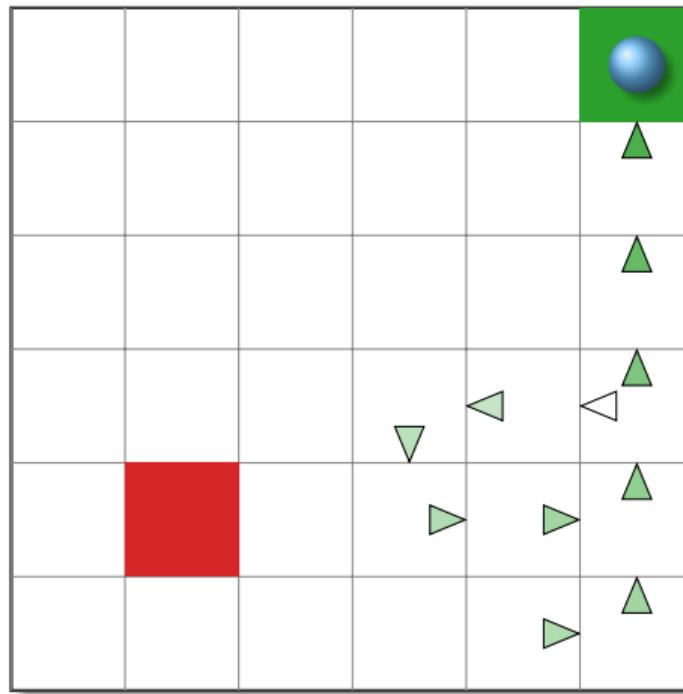
## prioritized sweeping with small backups



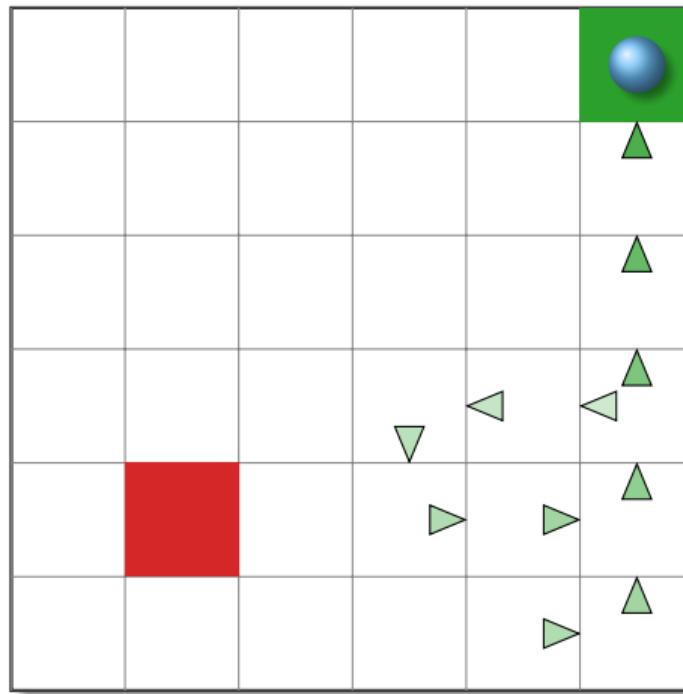
## prioritized sweeping with small backups



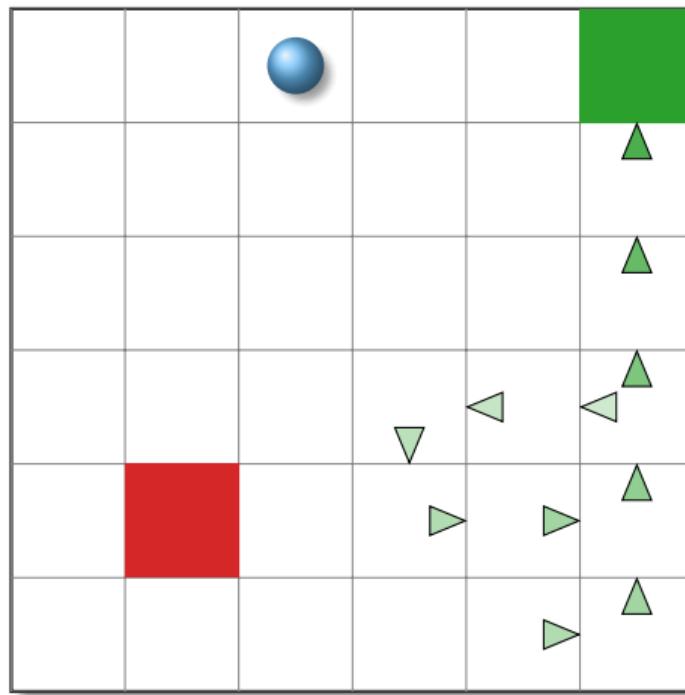
## prioritized sweeping with small backups



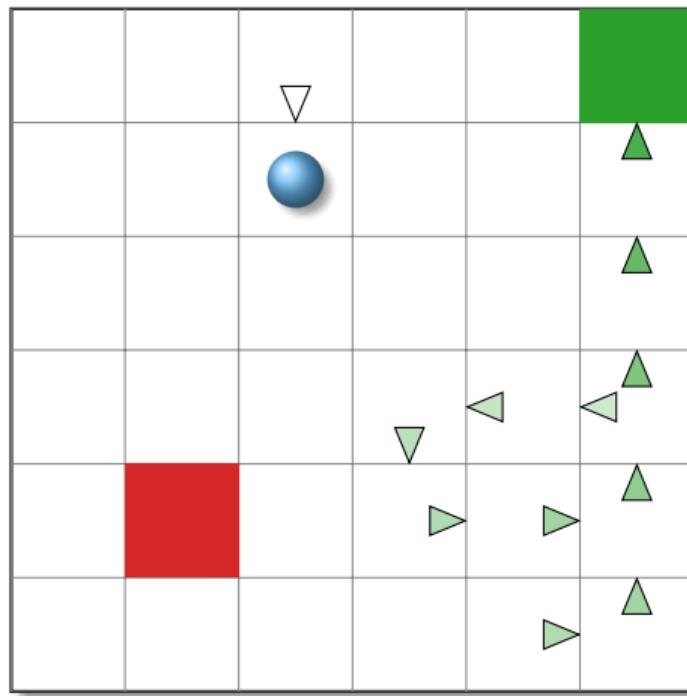
## prioritized sweeping with small backups



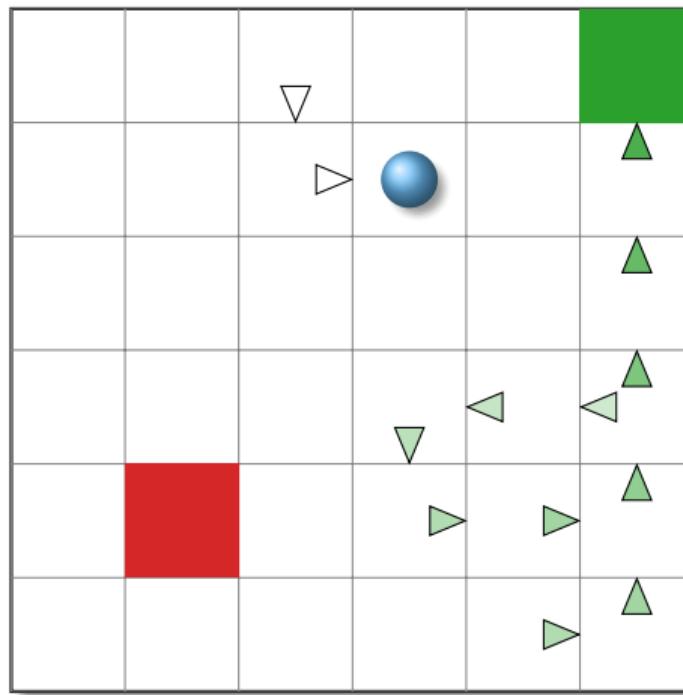
## prioritized sweeping with small backups



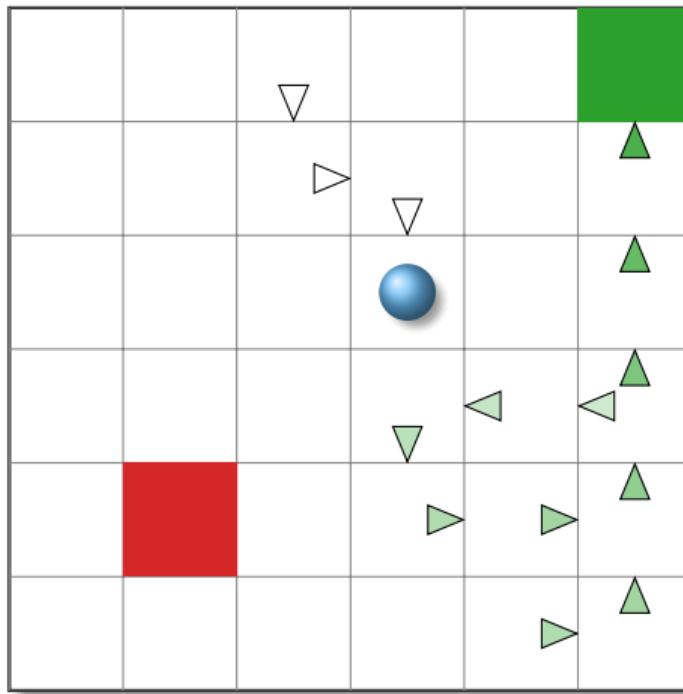
## prioritized sweeping with small backups



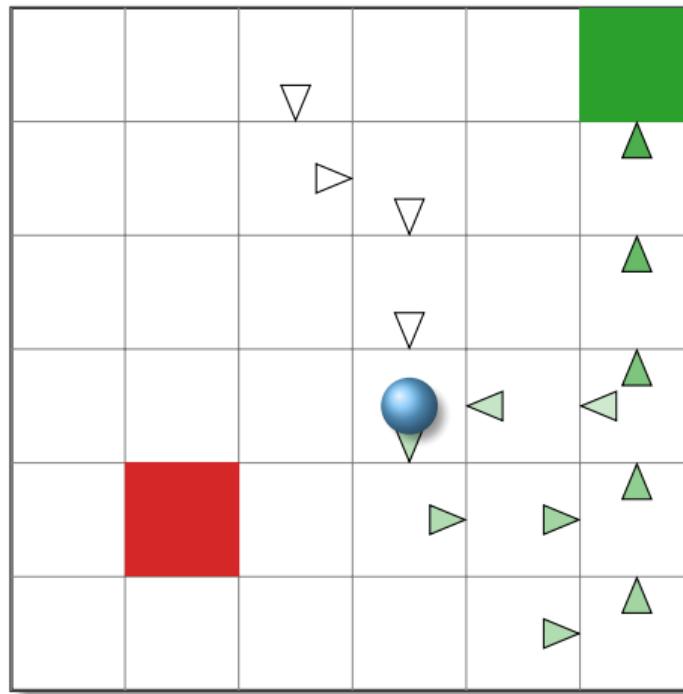
## prioritized sweeping with small backups



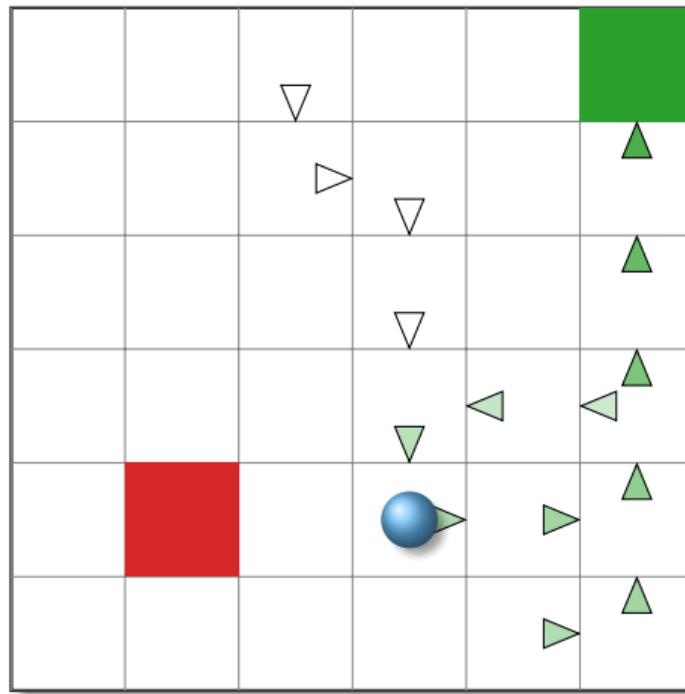
## prioritized sweeping with small backups



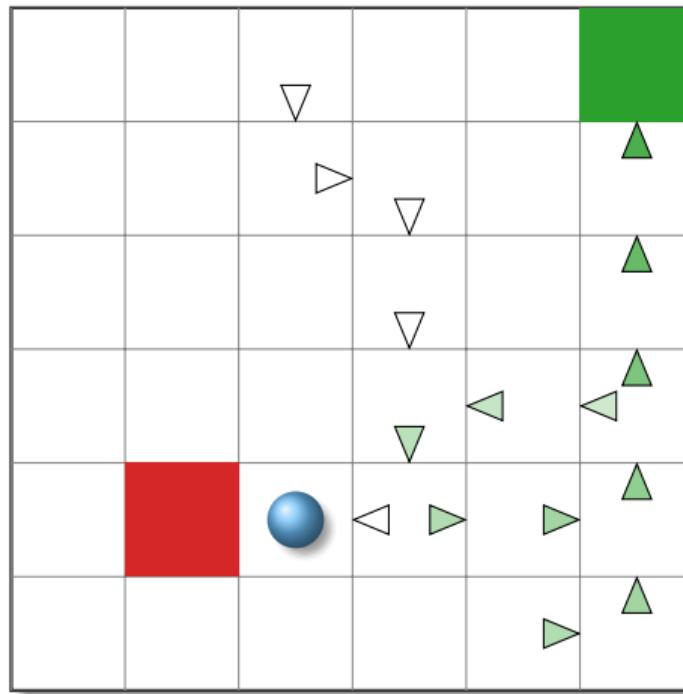
## prioritized sweeping with small backups



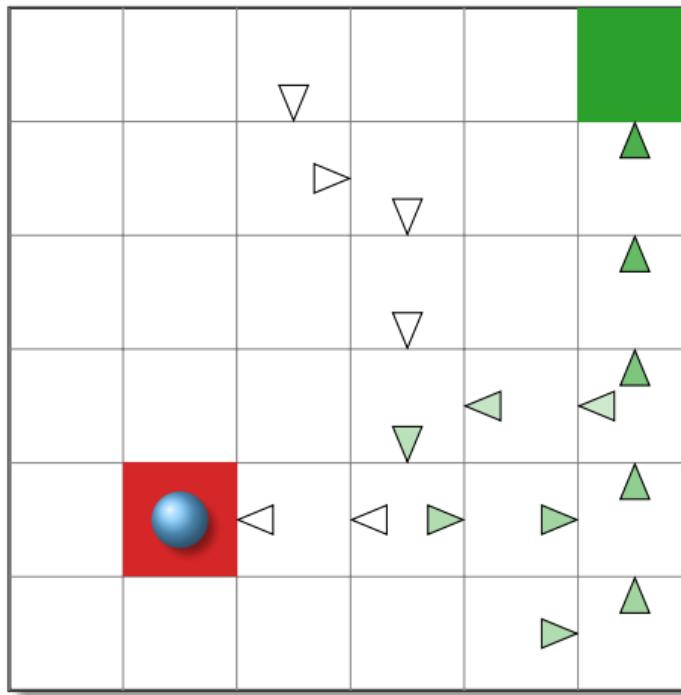
## prioritized sweeping with small backups



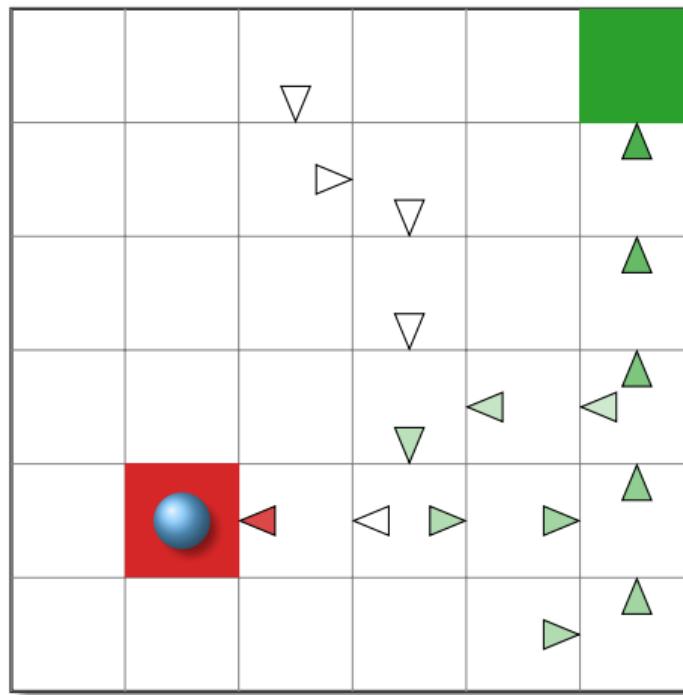
## prioritized sweeping with small backups



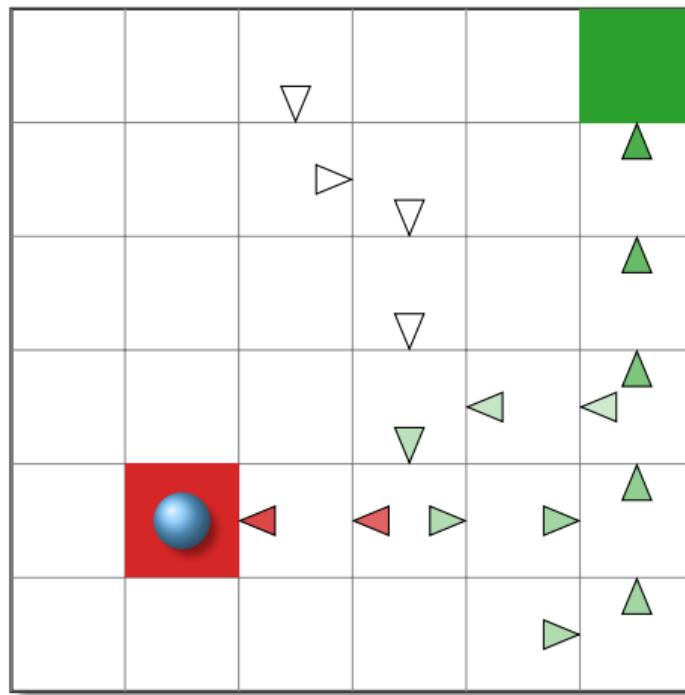
## prioritized sweeping with small backups



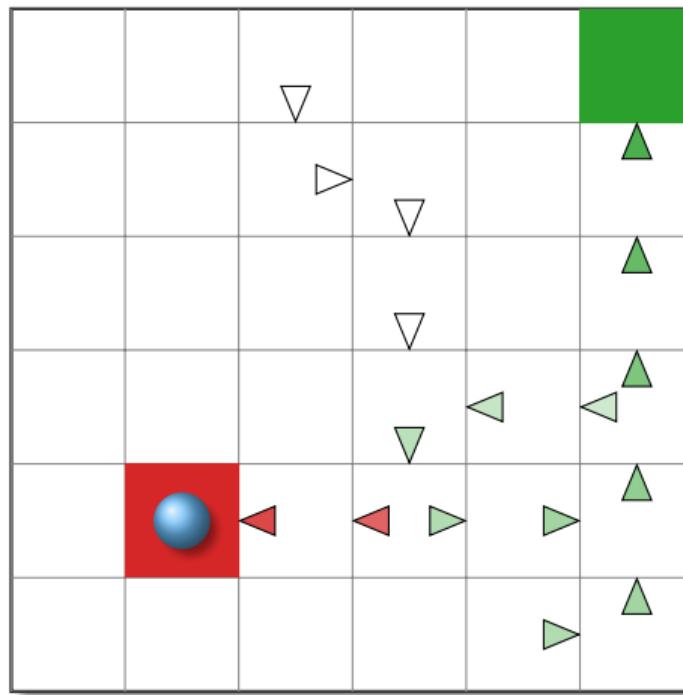
## prioritized sweeping with small backups



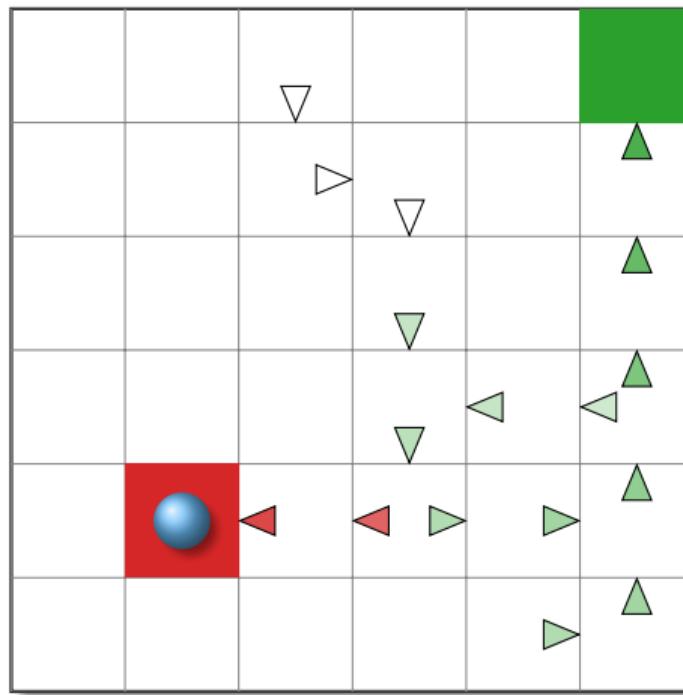
## prioritized sweeping with small backups



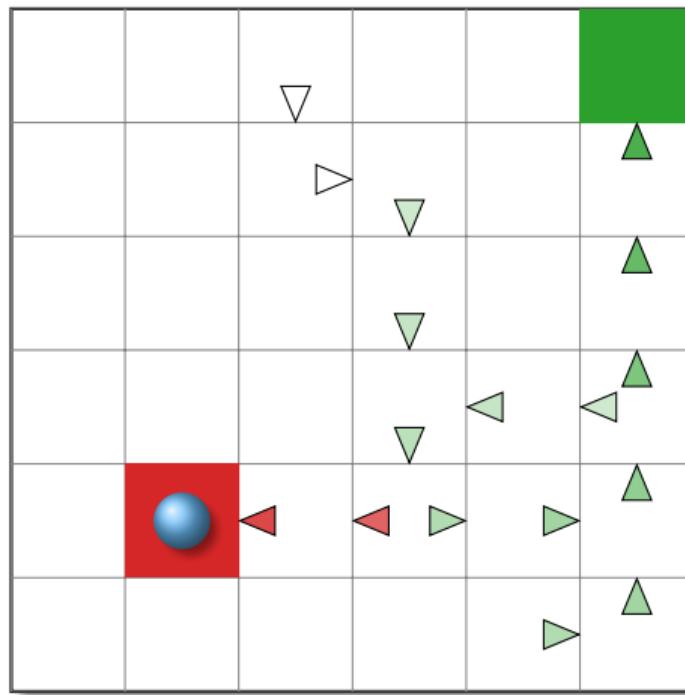
## prioritized sweeping with small backups



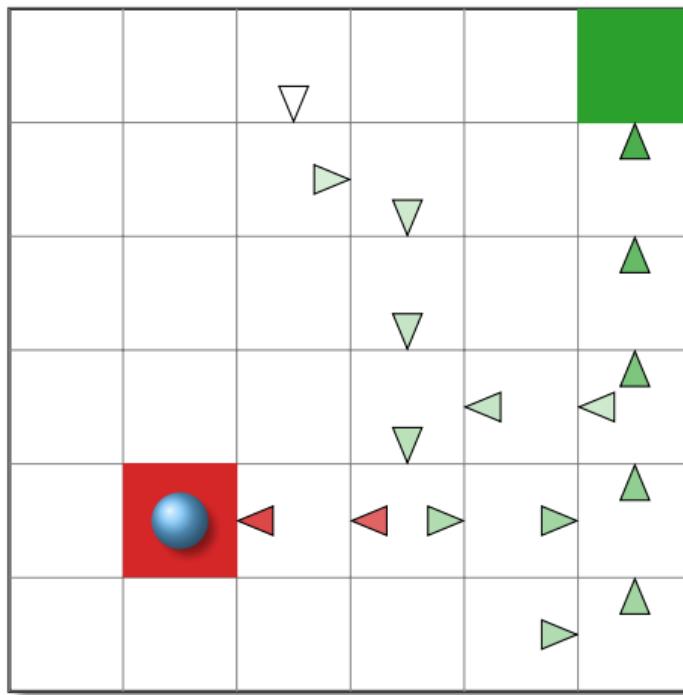
## prioritized sweeping with small backups



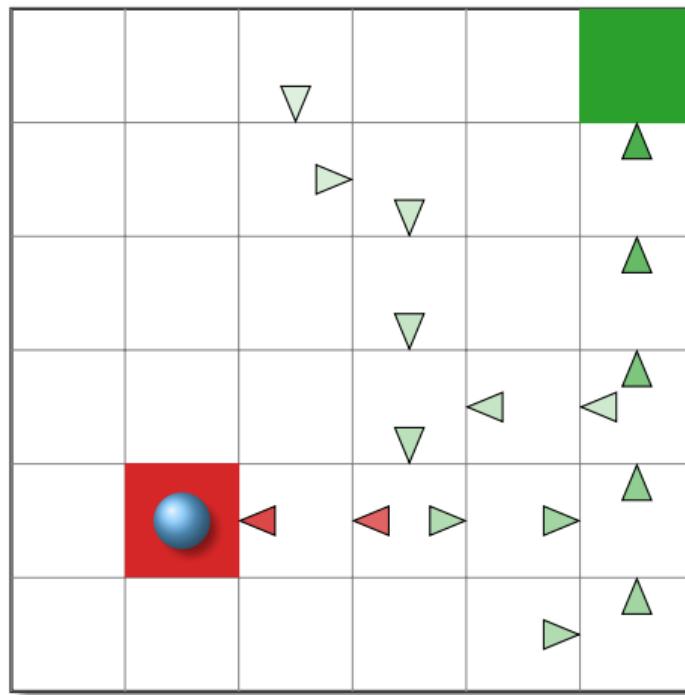
## prioritized sweeping with small backups



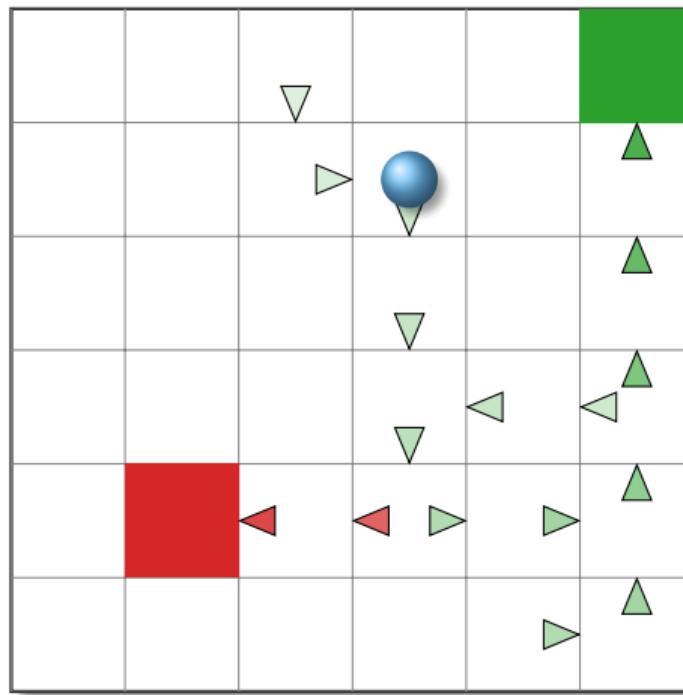
## prioritized sweeping with small backups



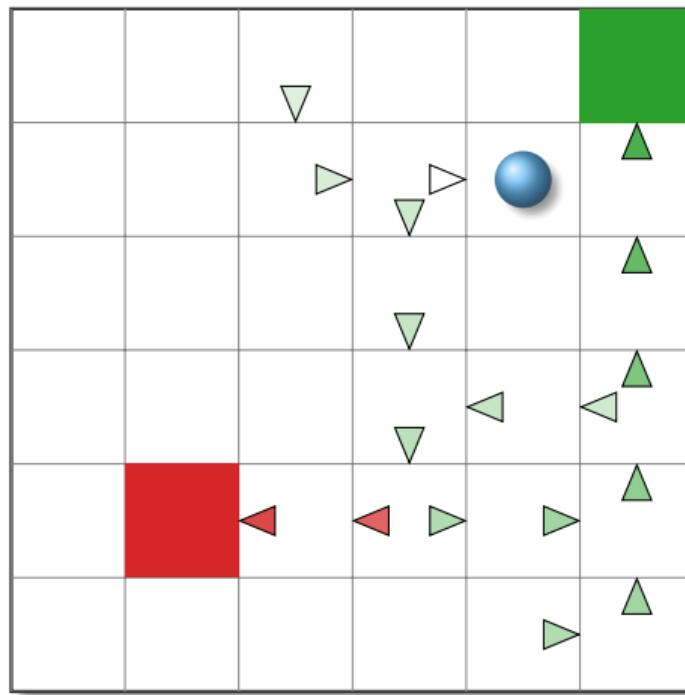
## prioritized sweeping with small backups



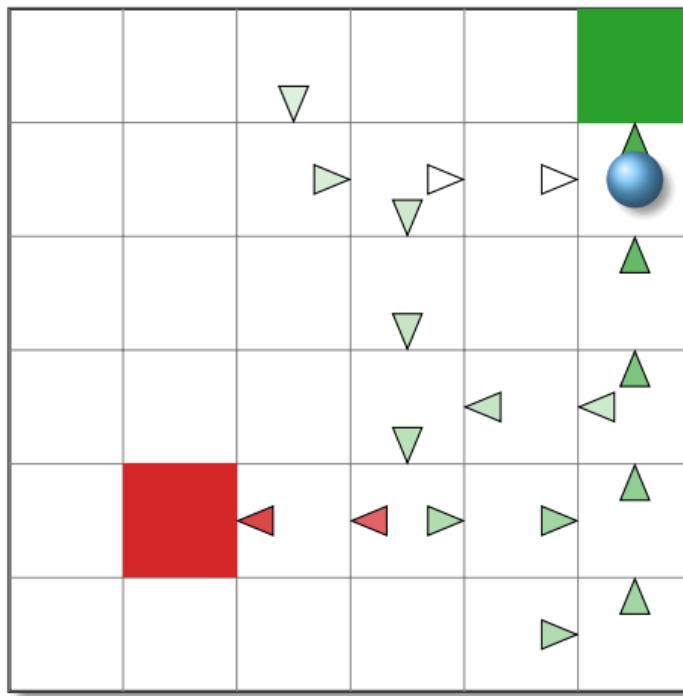
## prioritized sweeping with small backups



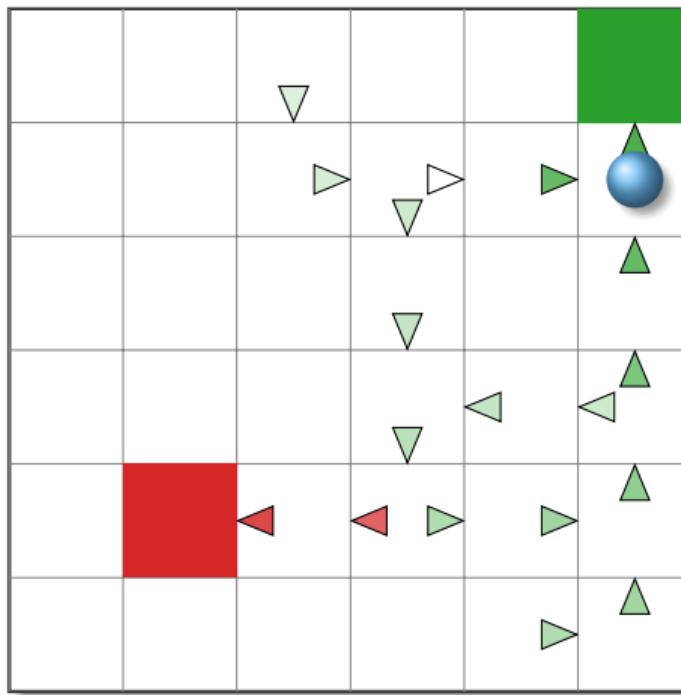
## prioritized sweeping with small backups



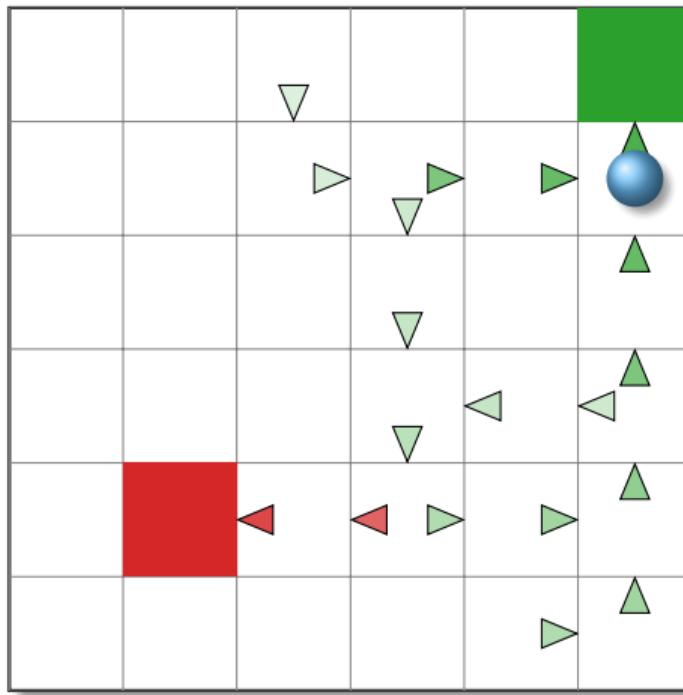
## prioritized sweeping with small backups



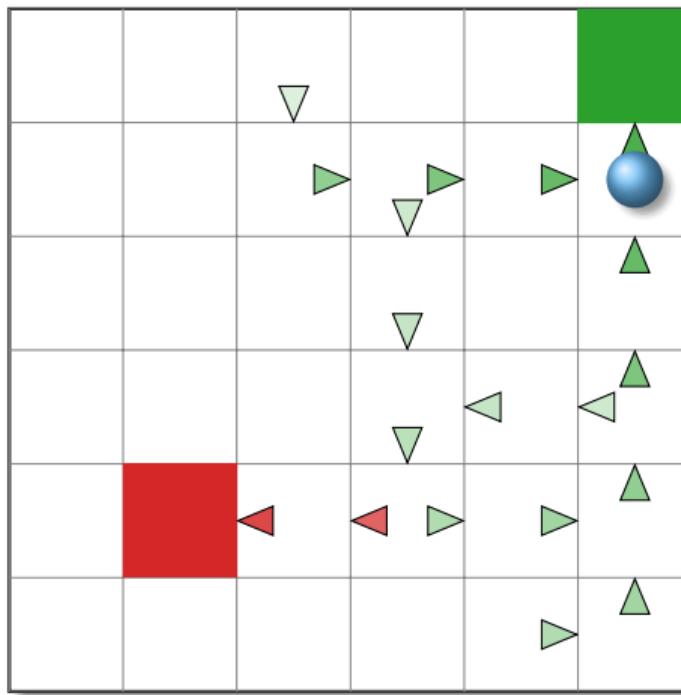
## prioritized sweeping with small backups



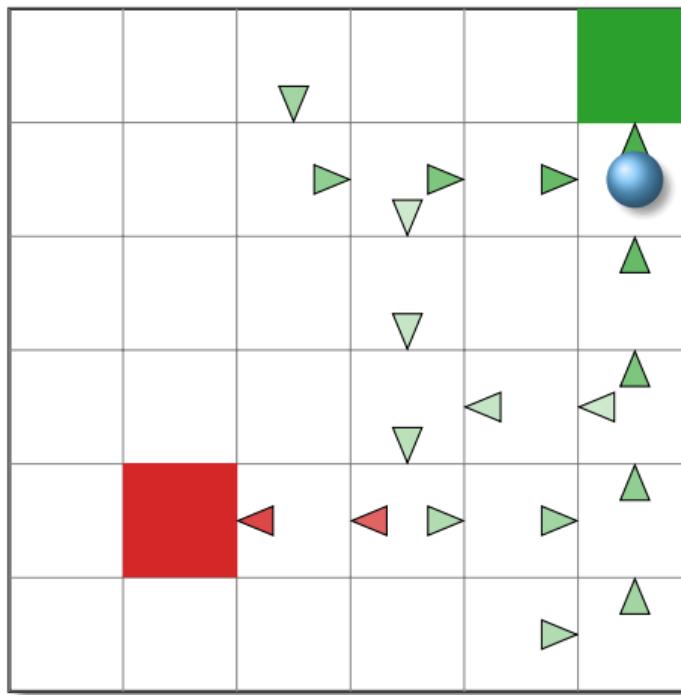
## prioritized sweeping with small backups



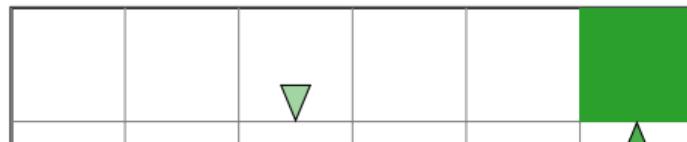
## prioritized sweeping with small backups



## prioritized sweeping with small backups

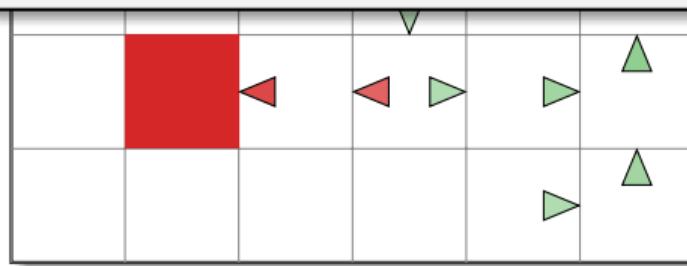


## prioritized sweeping with small backups

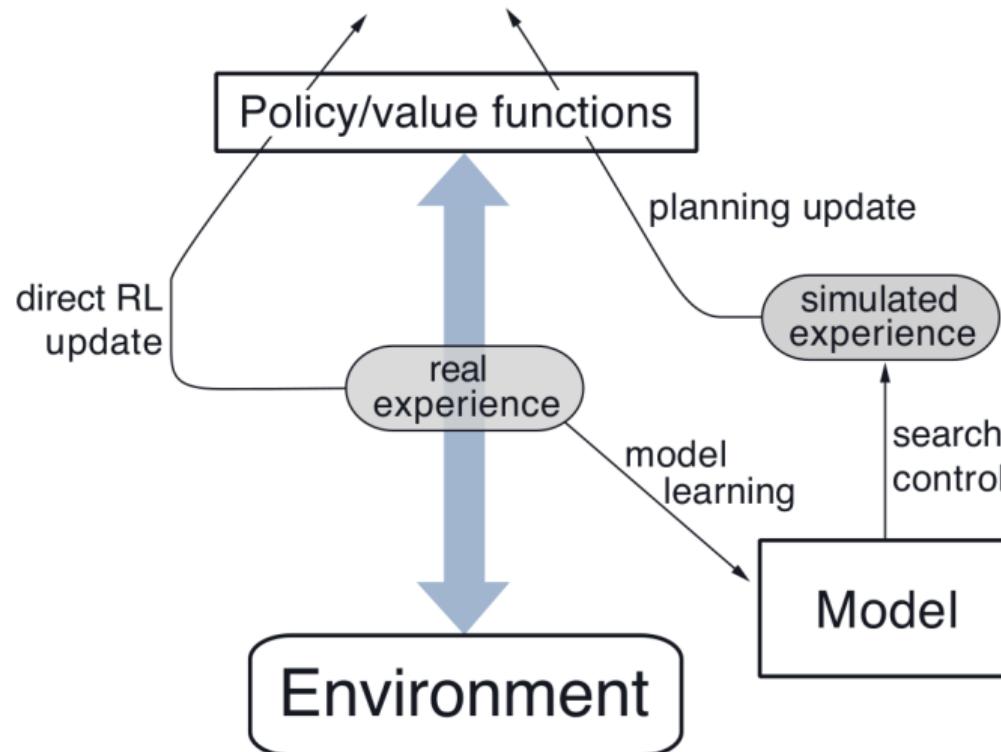


### Prioritized sweeping

- + Convergence
- Needs more memory/computation
- + **Highest sample efficiency**



# Dyna Architecture



# Summary and Conclusions

1. Methods that estimate and use  $P_{s \rightarrow s'}^a$  for planning are called **model-based**.
2. The **Dyna Architecture** visualizes, how model-based methods could work.
3. Prioritized sweeping with small backups is a **backward-focusing** planning method.
4. Application of insights to DQN:  
**Prioritized-DQN** samples replay memory better than uniform random (<https://arxiv.org/abs/1511.05952>).

## Quiz:

- [ ] SARSA is a model-based RL algorithm because Q(s, a) is learned.
- [ ] DQN is a model-based algorithm because of its use of a replay memory.
- [x] Prioritized sweeping is a model-based algorithm because it uses  $N_{s \rightarrow s'}^a / N_s^a$  to backup the Q-values.
- [x] Uniform sampling from replay memory or parallel interaction with independent environments reduces the variance of the gradient estimator.
- [ ] Prioritized DQN further reduces the variance of the gradient estimator.
- [x] Prioritized DQN learns faster than DQN, because the samples lead to better propagation of changes in Q-values.
- [x] Without the target network DQN is less stable.
- [x] To detect the motion of objects in ATARI games  
4 subsequent frames form the input of DQN and A3C.

**Two player board games (Go, Chess, Shogi)**

# What is special about board games?

- $P_{s \rightarrow s'}^a$  is perfectly known => planning methods can be applied
- State space is large
  - Chess  $\sim 10^{40} - 10^{50}$  positions
  - Go 19x19  $\sim 10^{170}$  positions  
(number of atoms on earth)  $\sim 10^{50}$
- Action space is not small
  - Chess  $\sim 10 - 30$  actions per position
  - Go  $\sim 100 - 361$  actions per position

**Should we use prioritized sweeping? No.**

Backups only along visited positions.  
No generalization to other positions.

# Classical approaches 1: MiniMax (with alpha-beta pruning)

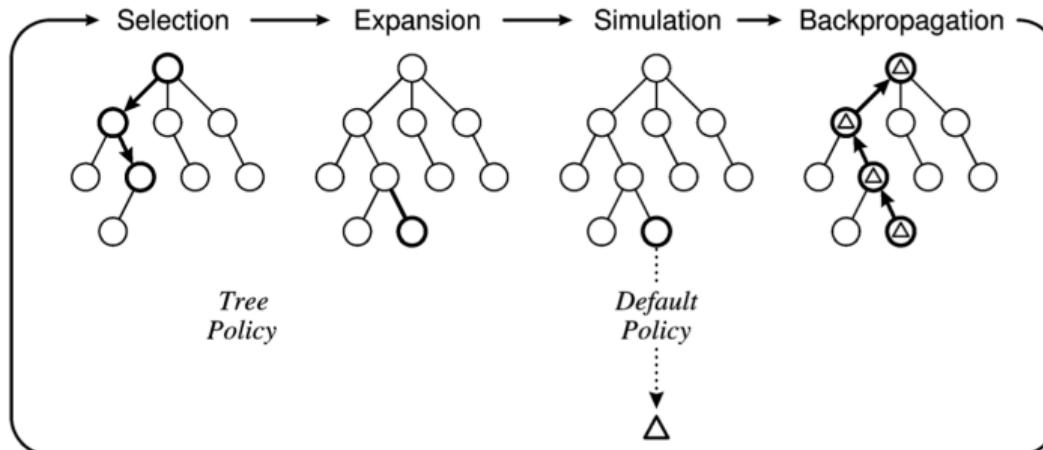
```
function MAX-VALUE( $s, \alpha, \beta$ )
    if terminal( $s$ ) return  $V(s)$ 
     $v = -\infty$ 
    for all  $c$  in next-states( $s$ ) do
         $v' = \text{MIN-VALUE}(c, \alpha, \beta)$ 
        if  $v' > v, v = v'$ 
        if  $v' \geq \beta$  return  $v$ 
        if  $v' > \alpha, \alpha = v'$ 
    end for
    return  $v$ 
end function
```

```
function MIN-VALUE( $s, \alpha, \beta$ )
    if terminal( $s$ ) return  $V(s)$ 
     $v = \infty$ 
    for all  $c$  in next-states( $s$ ) do
         $v' = \text{MAX-VALUE}(c, \alpha, \beta)$ 
        if  $v' < v, v = v'$ 
        if  $v' \leq \beta$  return  $v$ 
        if  $v' < \beta, \beta = v'$ 
    end for
    return  $v$ 
end function
```

Example on  
blackboard

- Typically used in chess engines (e.g. StockFish)
- $V(s)$  typically hand-crafted evaluation function of board position,  
e.g. a queen is more valuable than a pawn

# Classical approaches 2: Monte Carlo Tree Search (MCTS)



Example UCT  
on blackboard

- Typically used in go engines (e.g. MoGo, FueGo, Zen)
- No hand-crafted evaluation function of board positions needed
- (slow) convergence to minimax solution

# AlphaZero: the MCTS variant

$$1. \quad PUCB(s, a) = Q(s, a) + cP(s, a) \frac{\sqrt{N(s)}}{1 + N(s, a)}$$

Prior probability (focus)  
learned by neural net

2. Replace simulation part of MTCS by estimated win probability  $V(s)$  computed by a separate output of the neural net.
3. Select action at root node with probability

$$\pi(a|s_0) = N(s_0, a)^{1/\tau} / \sum_b N(s_0, b)^{1/\tau}$$

$\tau$  = 1 first 30 moves  
~ 0 afterwards

## **AlphaZero: the neural network**

- 1) Input: 17 planes ( $8 + 8 + 1$ )  
8 planes for own stones in last eight board positions  
8 planes for opponent stones in last eight board positions  
plane (all 0 or all 1) to indicate if white or black is to play
  - 2) Deep res-net with batch-normalization (79 layers)
  - 3) Output: policy head  $P(s, a)$     value head  $V(s)$

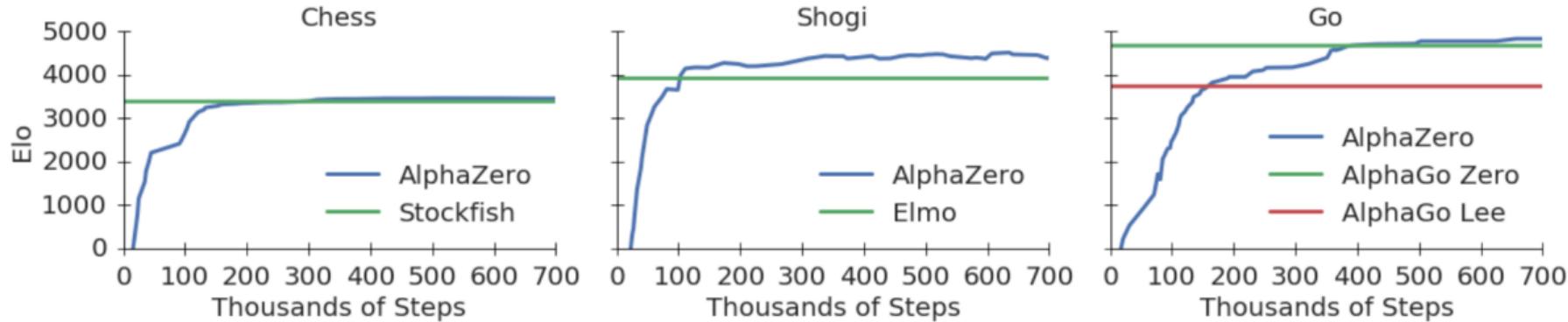
## Training:

Uniform sampling of 2048 positions  $s$  from last 500 000 games to form a minibatch. Loss:

$$(z - V(s))^2 - \sum_a \pi(a|s) \log P(s, a) + c\|\theta\|^2$$

Result of game
 $a$ 
Action probability from MCTS

# AlphaZero: success story



700 000 steps (minibatches of size 4096) using >5000 TPU

## Quiz:

- [x] MiniMax and Monte Carlo Tree Search require  $P_{s \rightarrow s'}^a$ .
- [x] MiniMax and Monte Carlo Tree Search can be part of a Dyna-Architecture.
- [ ] MCTS requires a value function to evaluate the leafs.
- [x] AlphaZero uses MCTS with a value function to evaluate the leafs.
- [ ] The probability of selecting a move in AlphaZero is given by the output of the policy neural network.
- [x] The probability of selecting a move in AlphaZero is determined by MCTS.
- [x] The output of the policy network is used in the selection phase of MCTS.
  
- [x] Instead of a hand-crafted value function used in chess engines with MiniMax, one could learn the value function through self-play like AlphaZero.

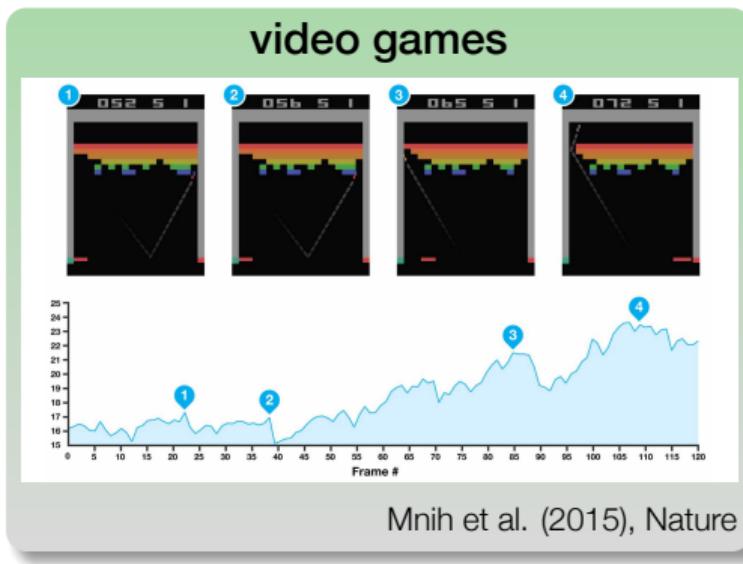
# From games to reality: what if the model is unknown?

Very active research:

- Oh et al. 2017 <https://arxiv.org/abs/1707.03497>  
Learn abstraction with neural network & MCTS-like planning
- Corneil, Gerstner, Brea 2018, <https://arxiv.org/abs/1802.04325>  
Learn abstraction with neural network & prioritized sweeping
- Nagabandi et al. 2017 <https://arxiv.org/abs/1708.02596>  
Learn continuous dynamics in simulated robotics domain
- Weber et al. 2017 <https://arxiv.org/abs/1707.06203>  
Learn abstraction and rollout strategy

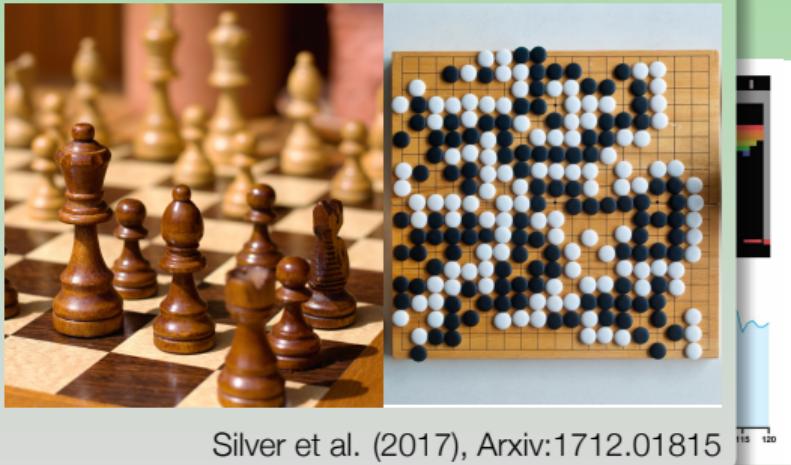
General problem: errors accumulate in planning with imperfect model

# Success stories and limitations of deep reinforcement learning



# Success stories and limitations of deep reinforcement learning

## board games

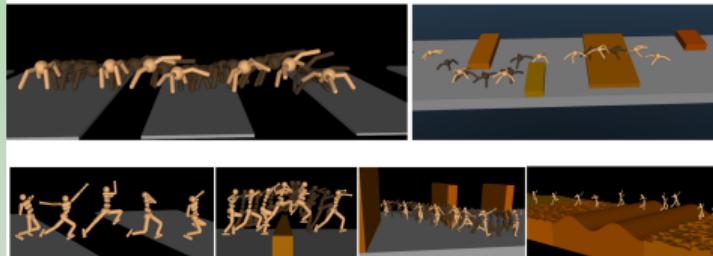


Silver et al. (2017), Arxiv:1712.01815

Mnih et al. (2015), Nature

# Success stories and limitations of deep reinforcement learning

## simulated robotics

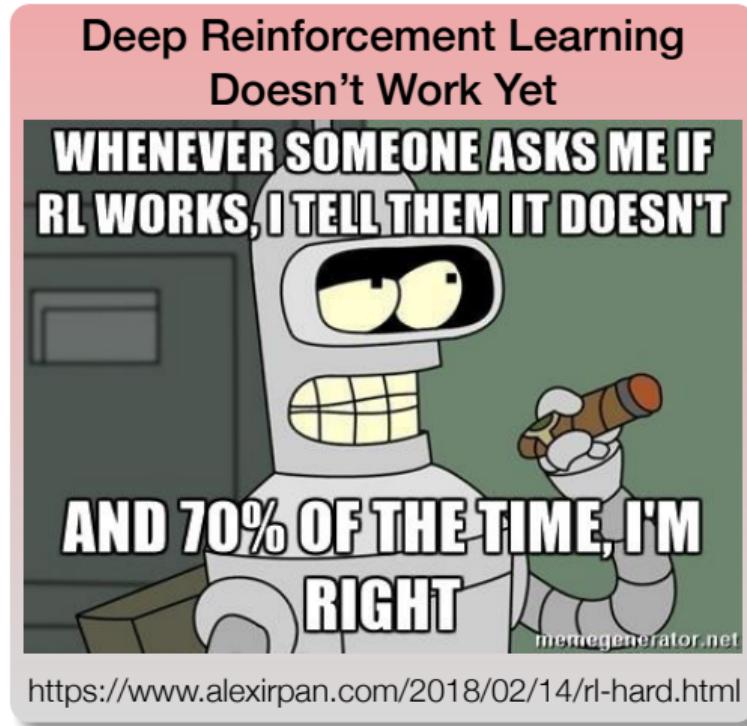


Heess et al. (2017), Arxiv:1707.02286

Silver et al. (2017), Arxiv:1712.01815

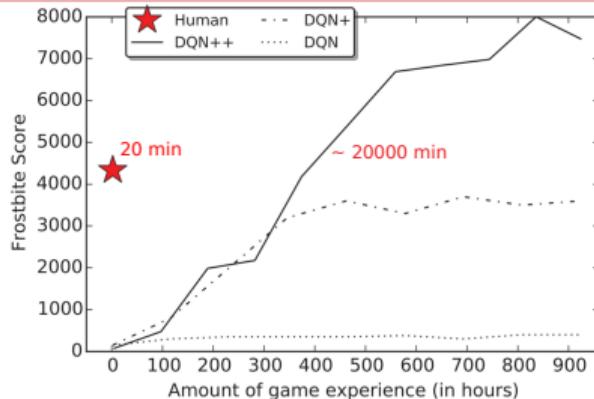
Mnih et al. (2015), Nature

# Success stories and limitations of deep reinforcement learning

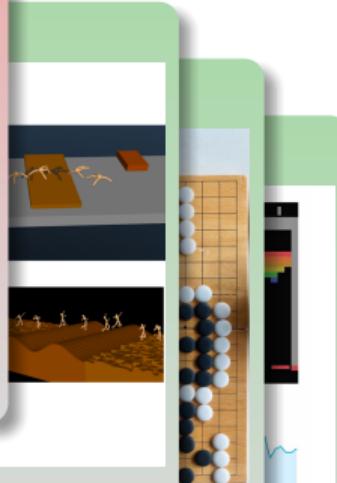


# Success stories and limitations of deep reinforcement learning

Humans learn much faster



Lake et al. (2016), Behav. and brain sc.



Heess et al. (2017), Arxiv:1707.02286

Silver et al. (2017), Arxiv:1712.01815

Mnih et al. (2015), Nature