Artificial Neural Networks: Lecture 12
Use Cases of Deep Reinforcement Learning

Johanni Brea
EPFL, Lausanne, Switzerland

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

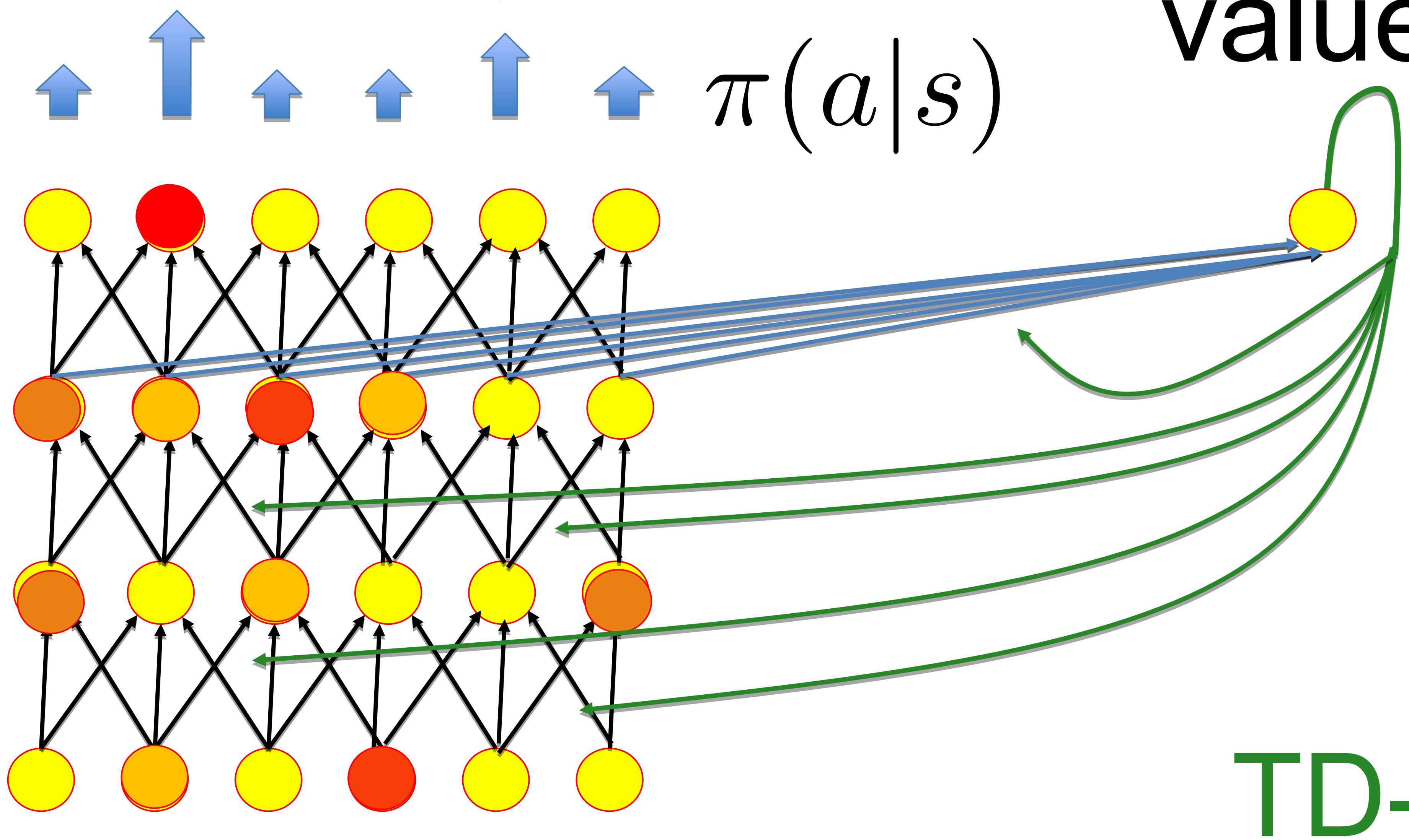
Review: Actor-Critic Policy Gradient

actions

advance *push left*

$\pi(a|s)$

value $V(s)$



- Estimate $V(s)$
- learn via TD error

$$\Delta w = \eta \delta_t \frac{\partial}{\partial w} \ln \pi(a_t|s_t)$$

TD-error

Estimate of total return

$$\delta_t = (r_t + \gamma V(s_{t+1})) - V(s_t))$$

baseline

TD-error (n-steps) e.g. n = 3

$$\delta = (r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \gamma^3 V(s_{t+3})) - V(s_t))$$

Asynchronous game play and entropy regularization

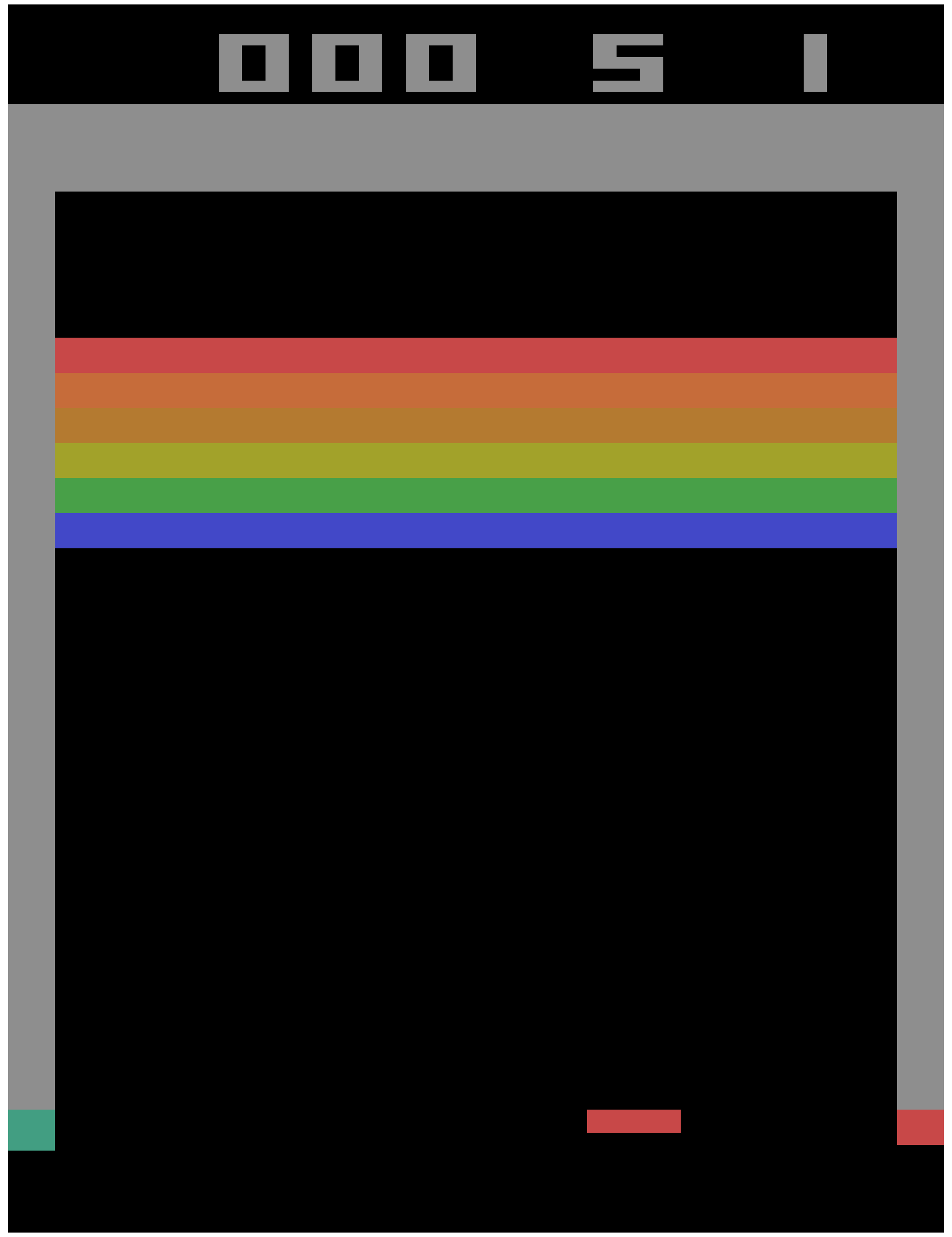
1) Minibatches allow to leverage parallelization e.g. GPU
   Problem: Minibatches for Actor-Critic Policy Gradient?
   Proposed solution: Interact with N environments in parallel.
2) Policy can become deterministic too quickly, e.g.

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

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

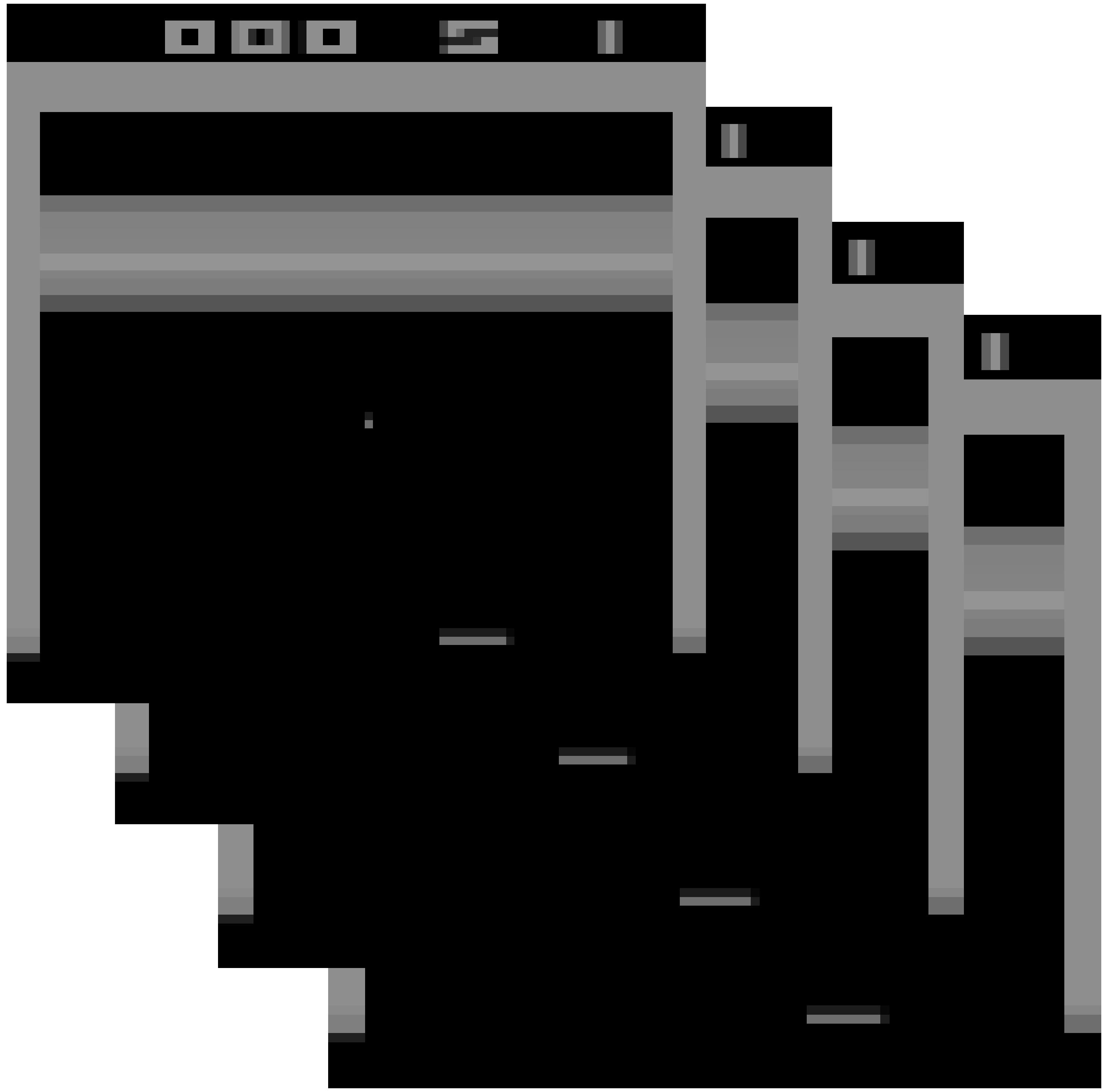
A3C = **A**synchronous (interaction with N environments)
Advantage (TD-error = advantage of chosen action)
Actor (policy network)
Critic (value network)

Atari Video Games (preprocessing)



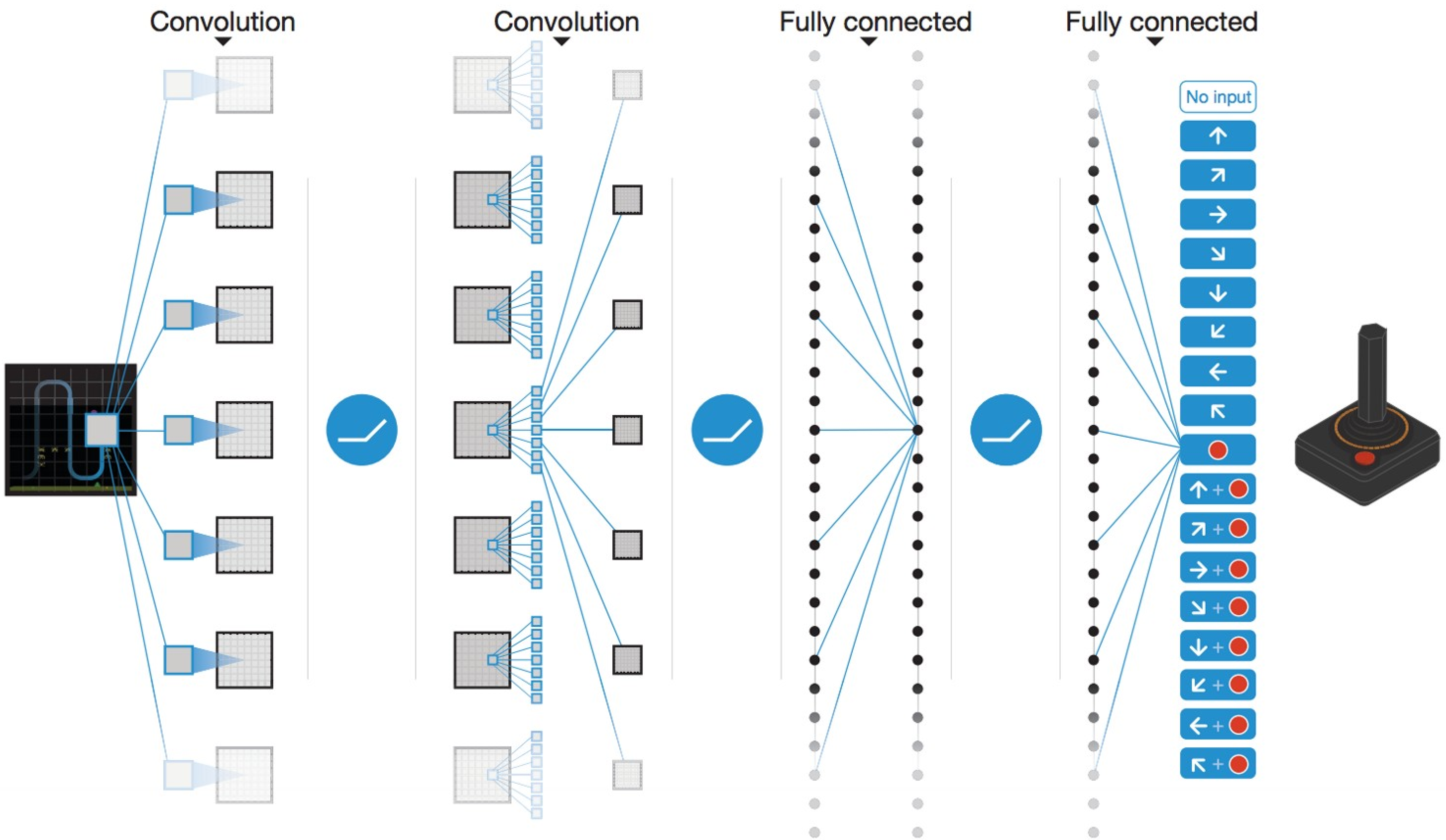
o(t) = original

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



s(t) = (g(t), …, g(t-3))
input to convnet

Learning Atari Games with A3C



- 8x8x32 stride 4 => 4x4x64 stride 2 => 3x3x64 => 512 => 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)

for all steps **do**

  select action a_t with ϵ-greedy policy using $Q_a(s_t)$ standard preprocessing

  observe reward r_t and image x_{t+1} and preprocess $s_{t+1} = \phi(s_t, x_{t+1})$

  Store transition (s_t, a_t, r_t, s_{t+1}) in replay memory 1M transitions

  Sample random minibatch (s_j, a_j, r_j, s_{j+1}) from replay memory

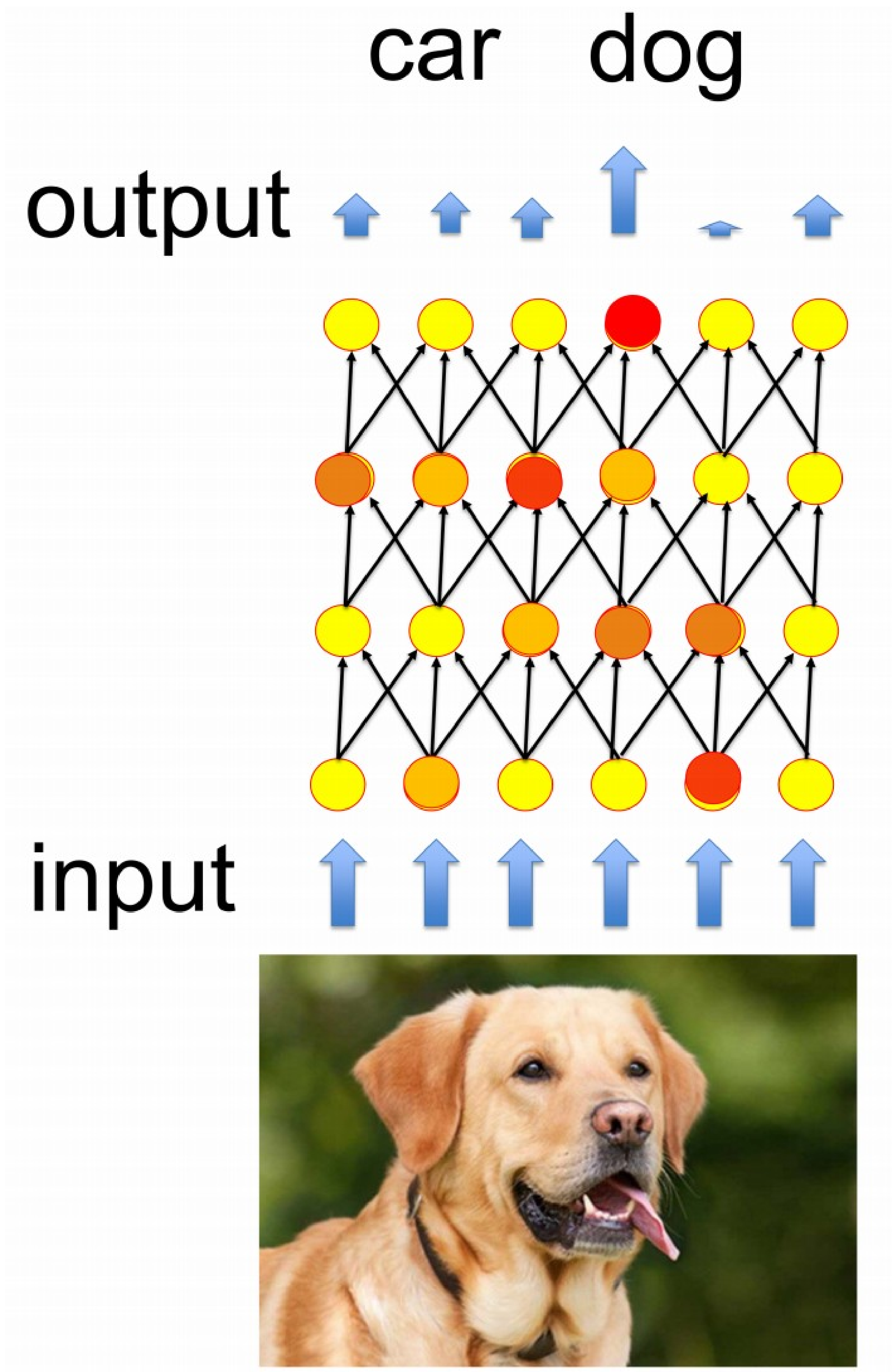
  Perform (semi-)gradient step on $\mathcal{L}\left(r_j + \gamma \max_a \hat{Q}_a(s_{j+1}) - Q_{a_j}(s_j)\right)$

  Every C steps reset $\hat{Q} = Q$.

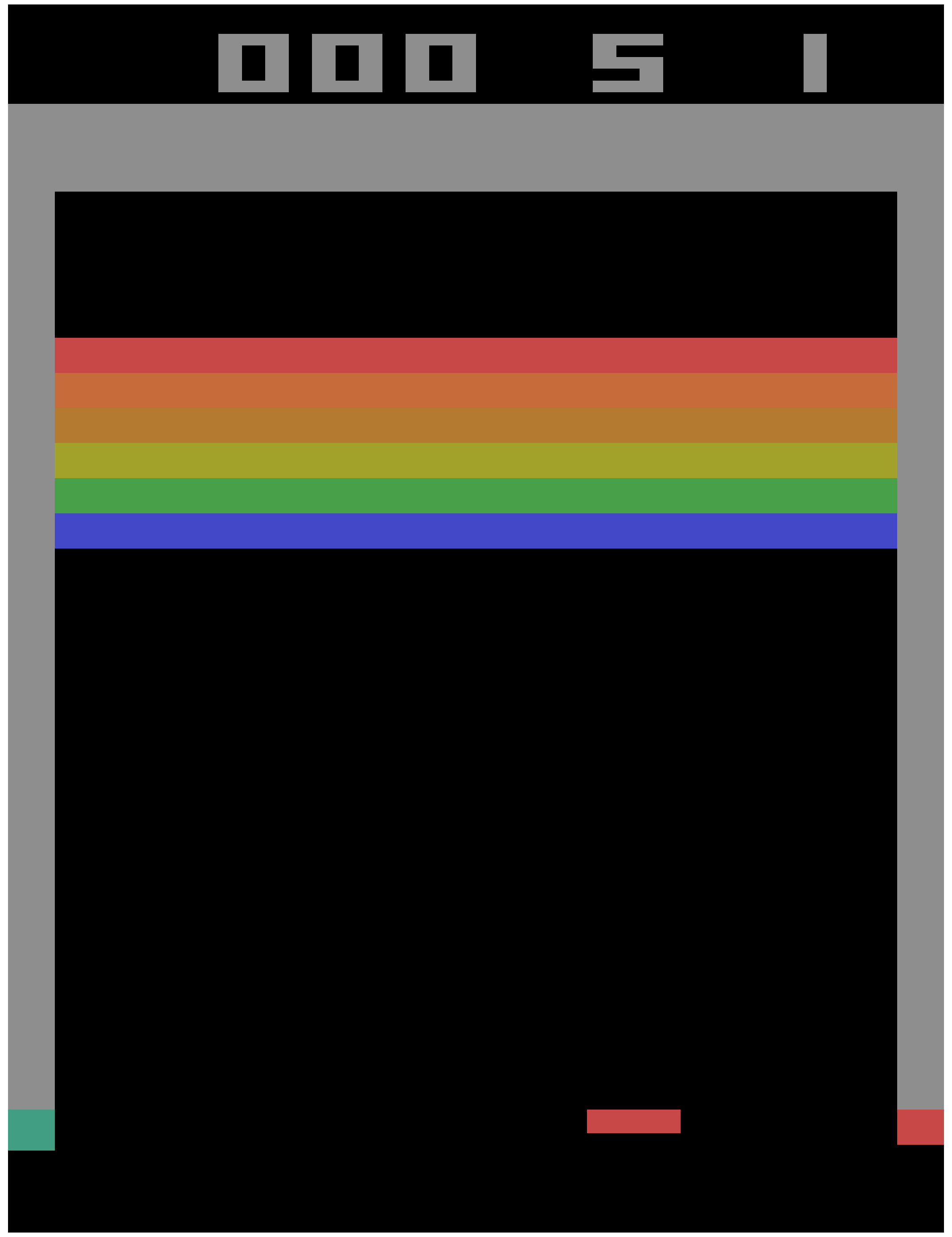
end for

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

Decorrelation for Deep RL



Classification:
uncorrelated sampling
of training data



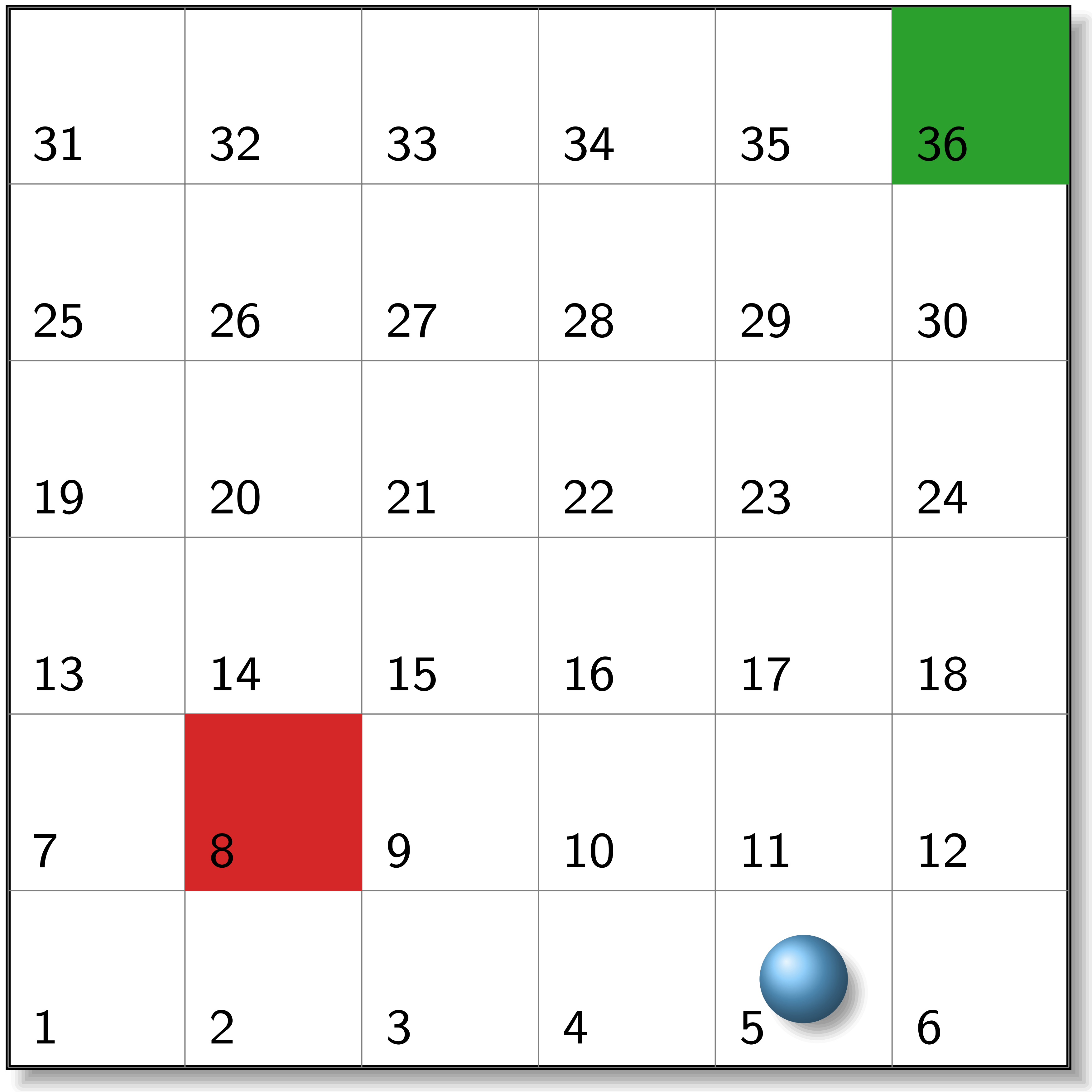
RL:
Subsequent inputs often
highly correlated

Possible solutions

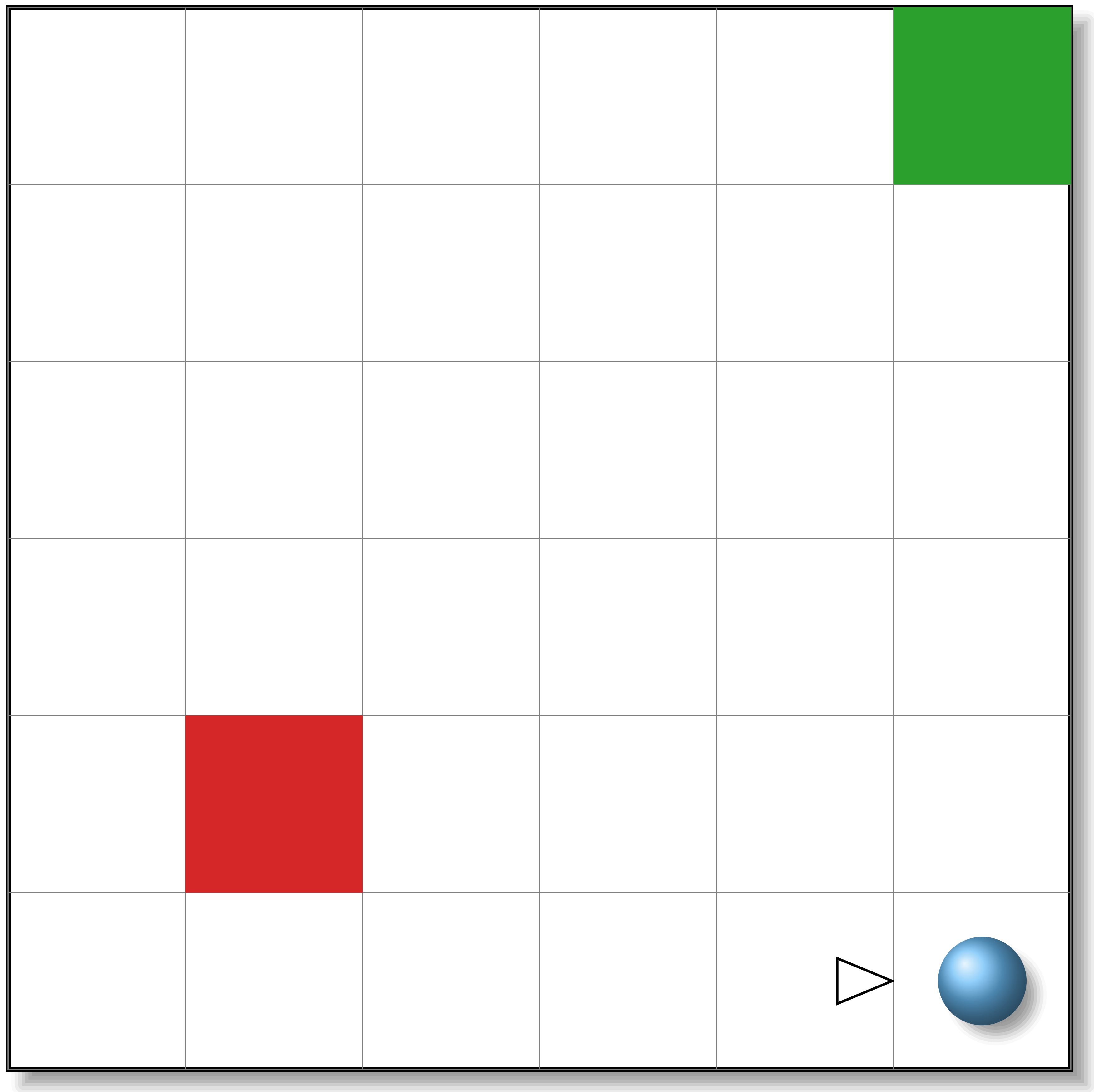
1) Parallel interaction with N independent environments (A3C)
2) Sampling from replay memory (DQN)

Replay Memory and Planning in Tabular Environments

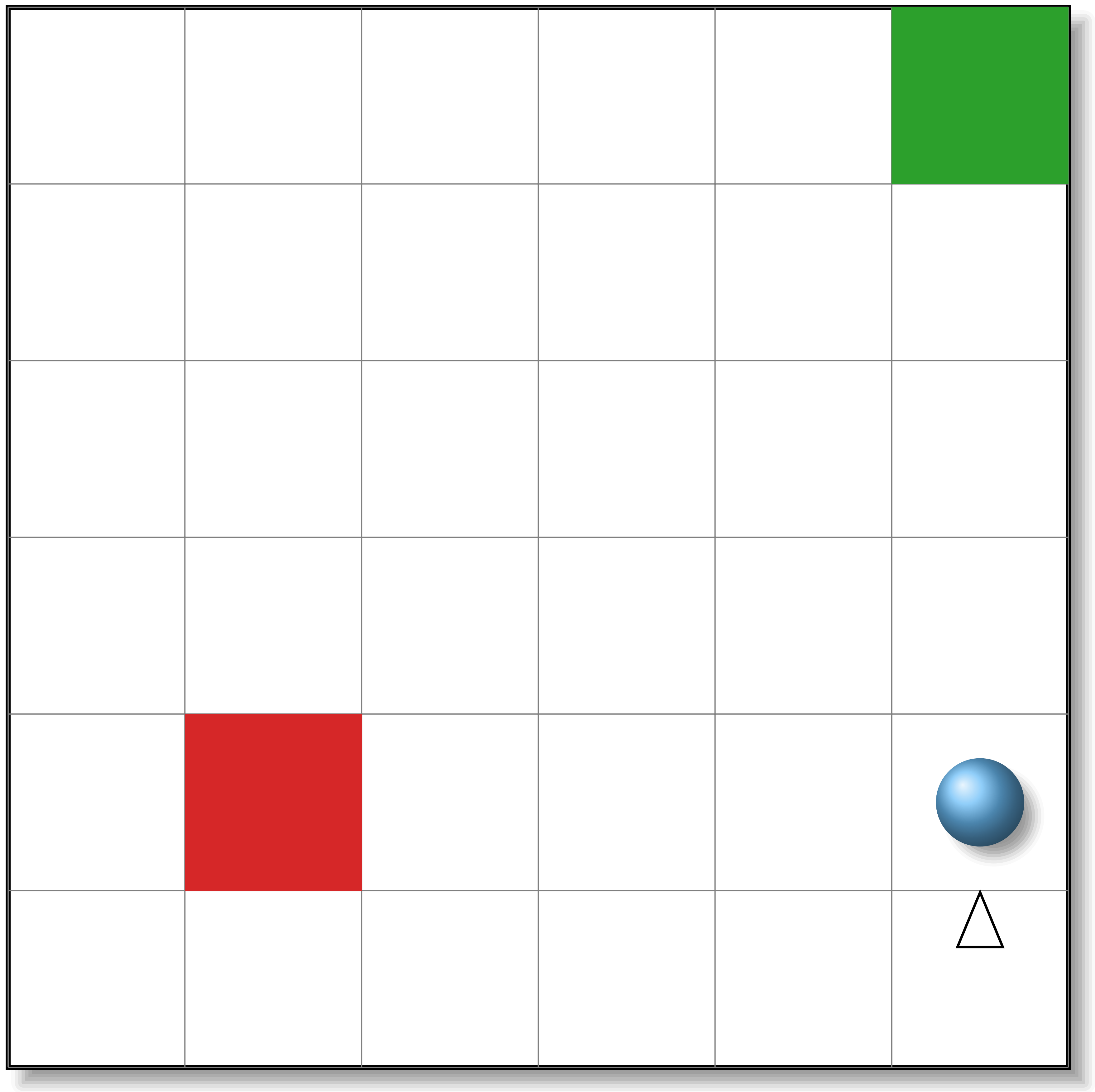
standard tabular Q-Learning



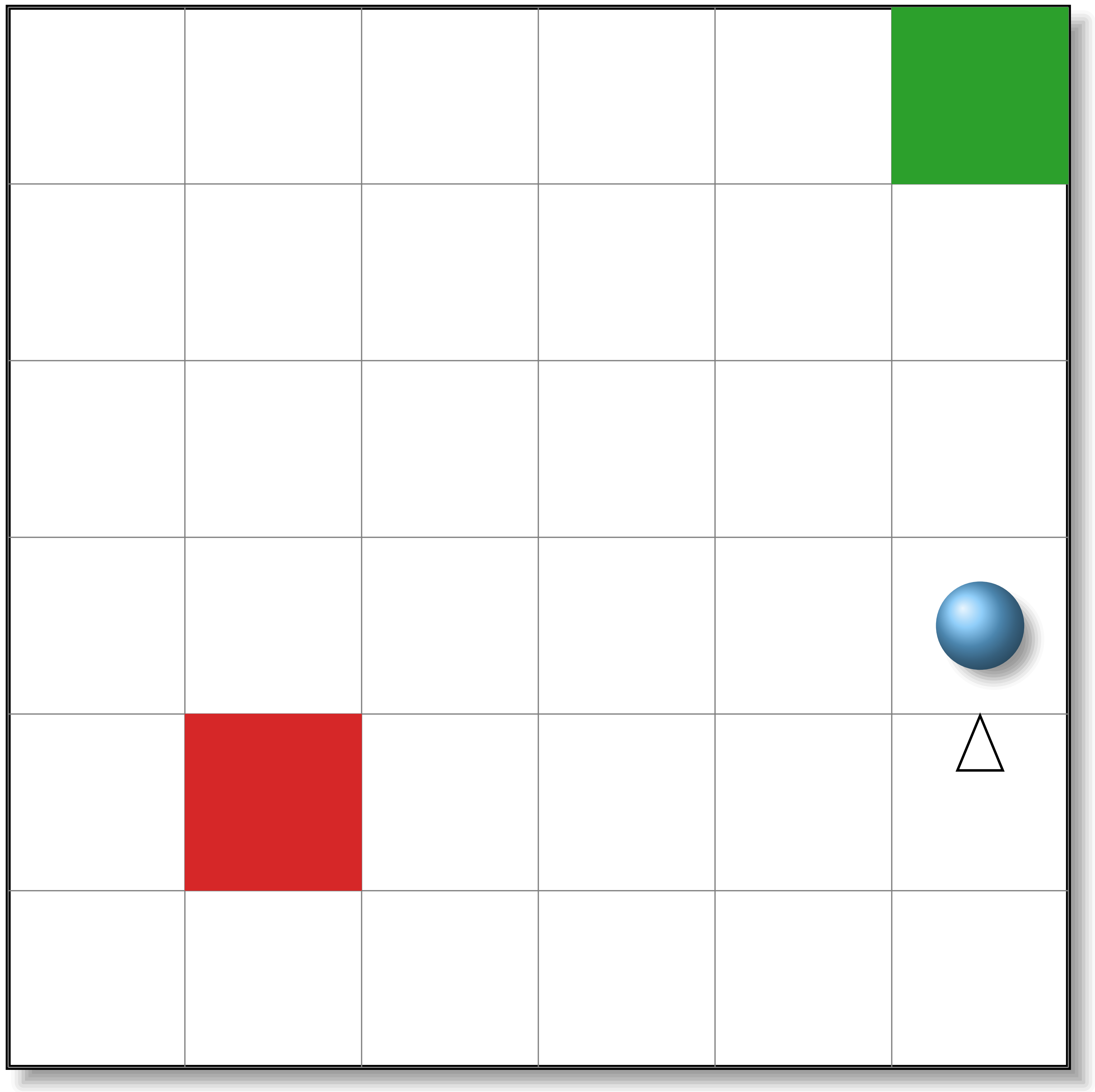
standard tabular Q-Learning



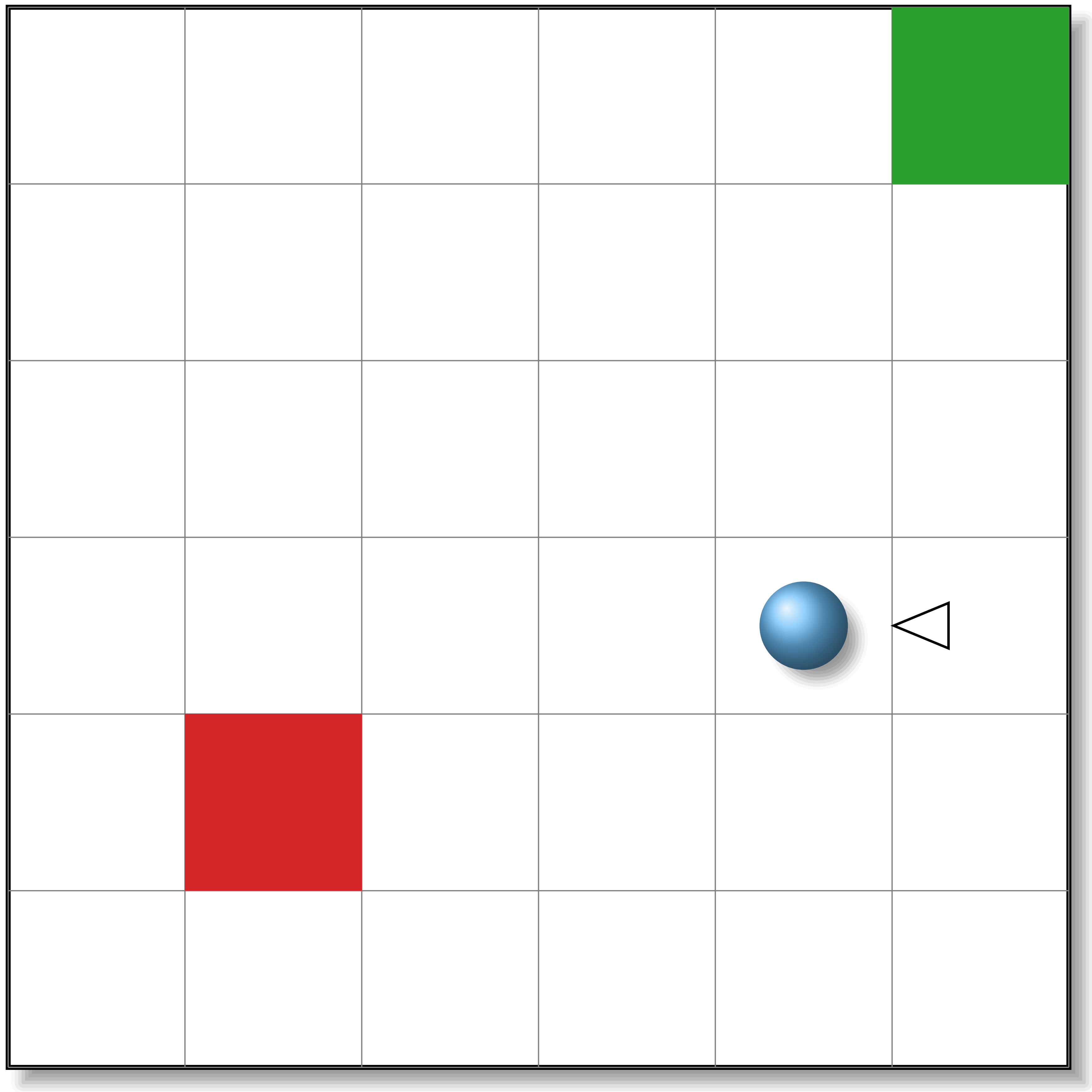
standard tabular Q-Learning



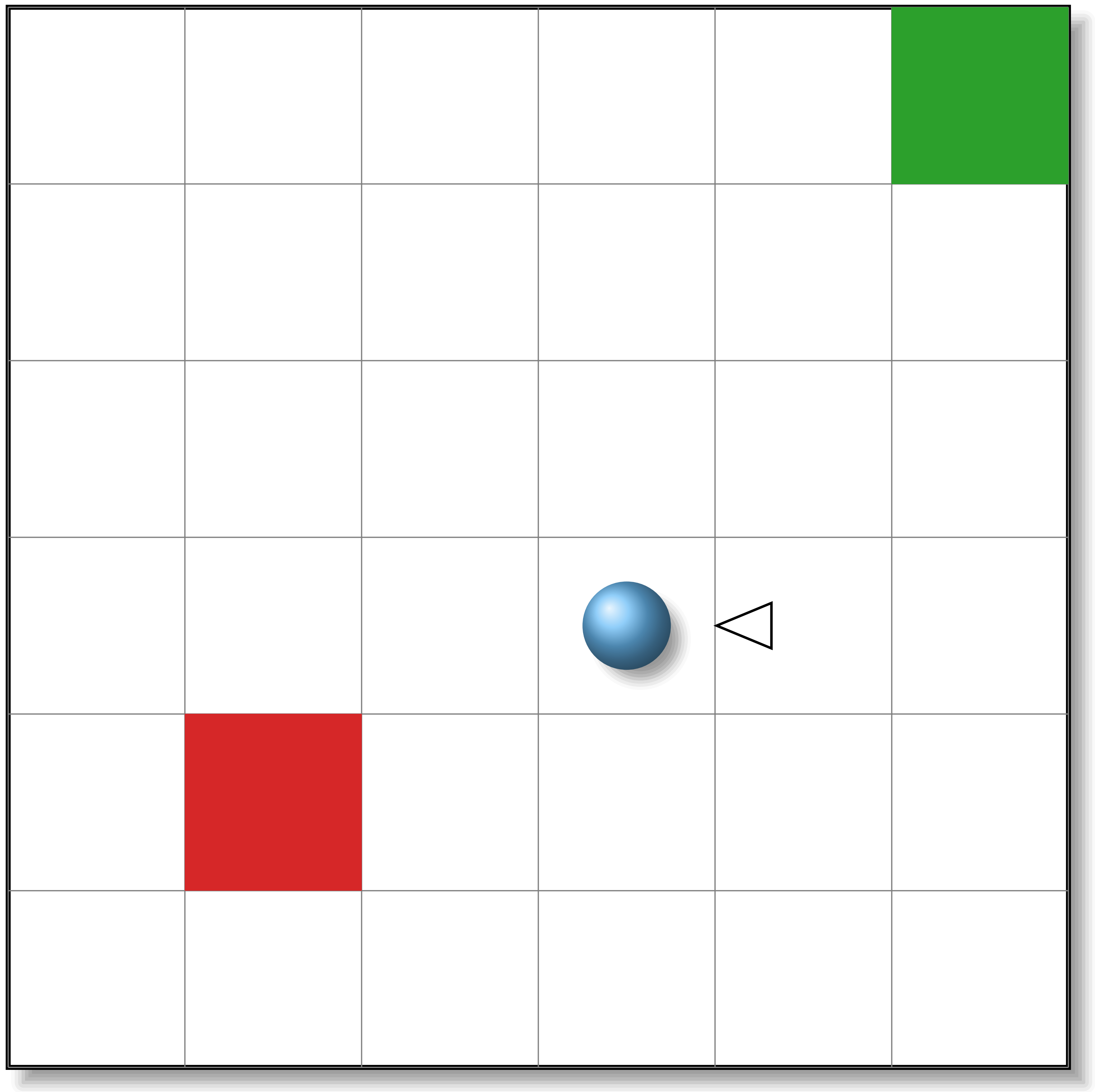
standard tabular Q-Learning



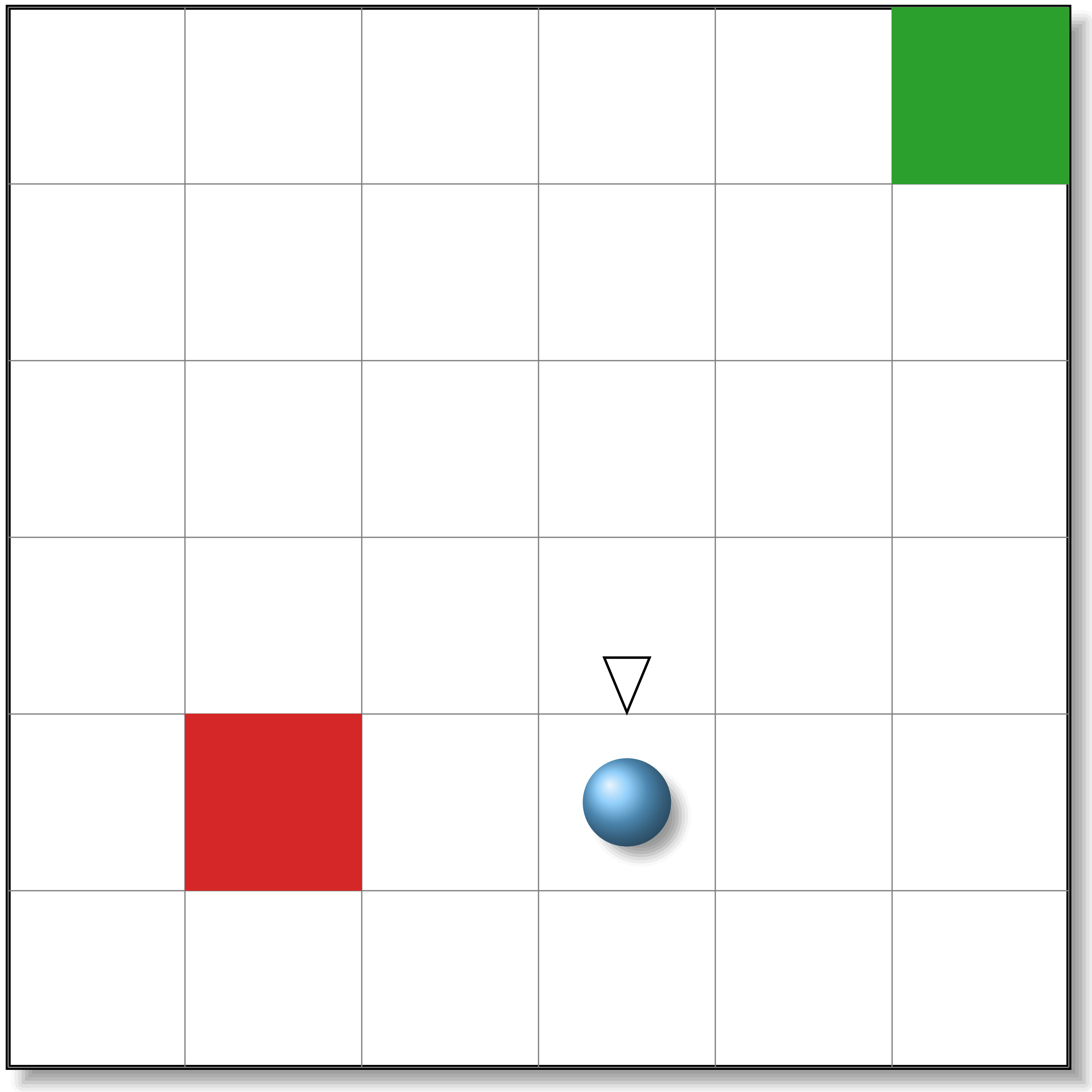
standard tabular Q-Learning



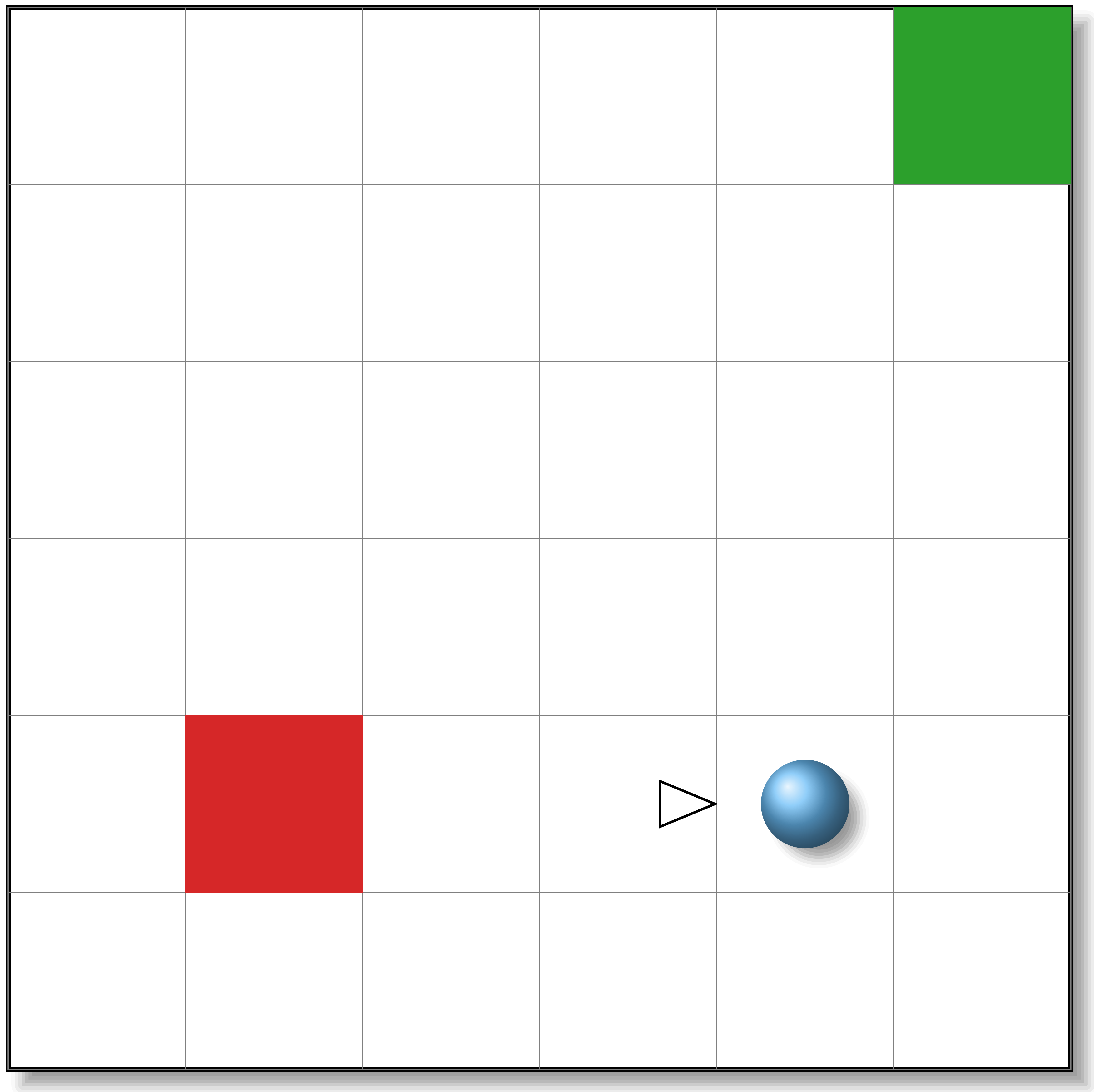
standard tabular Q-Learning



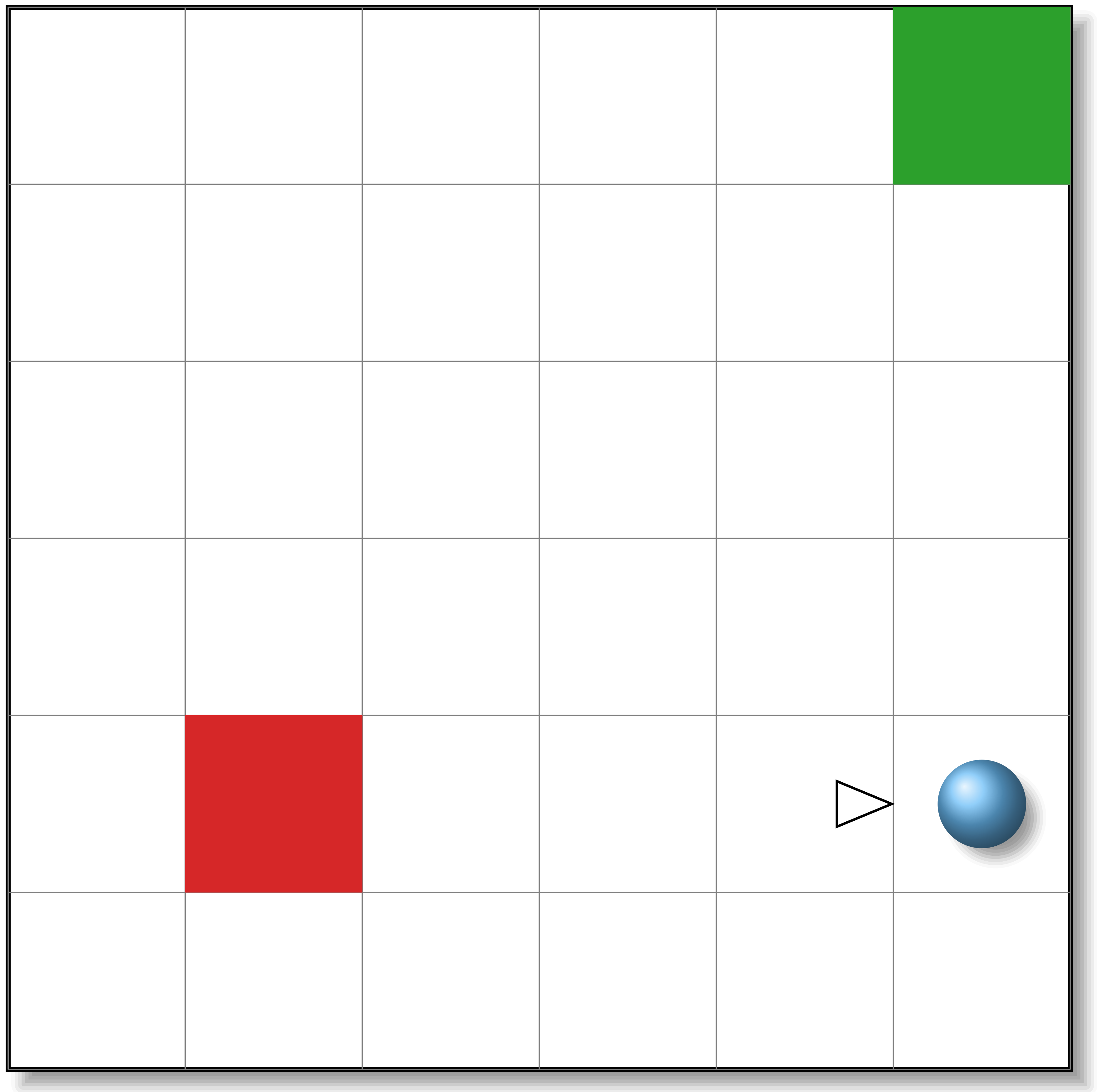
standard tabular Q-Learning



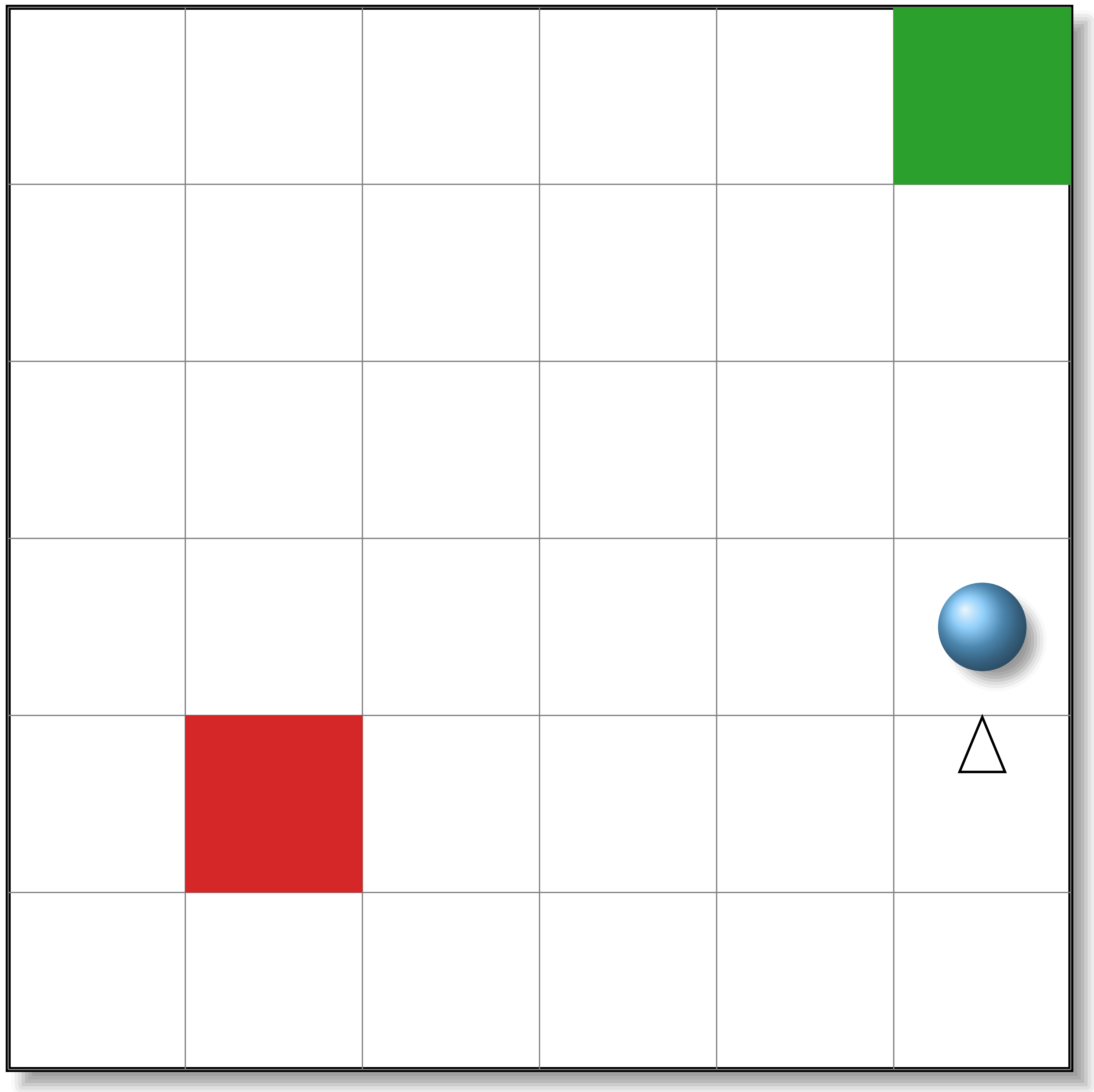
standard tabular Q-Learning



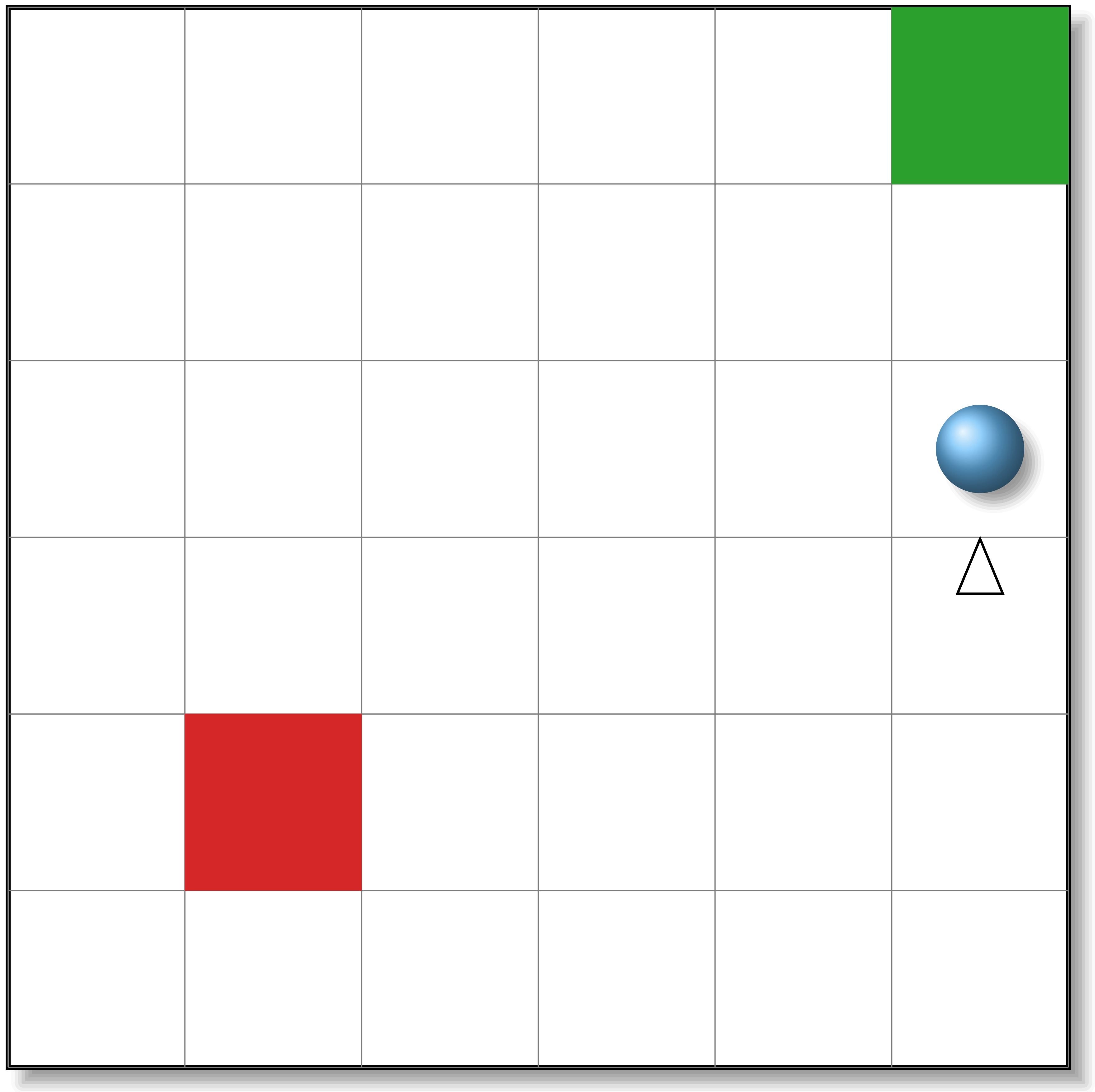
standard tabular Q-Learning



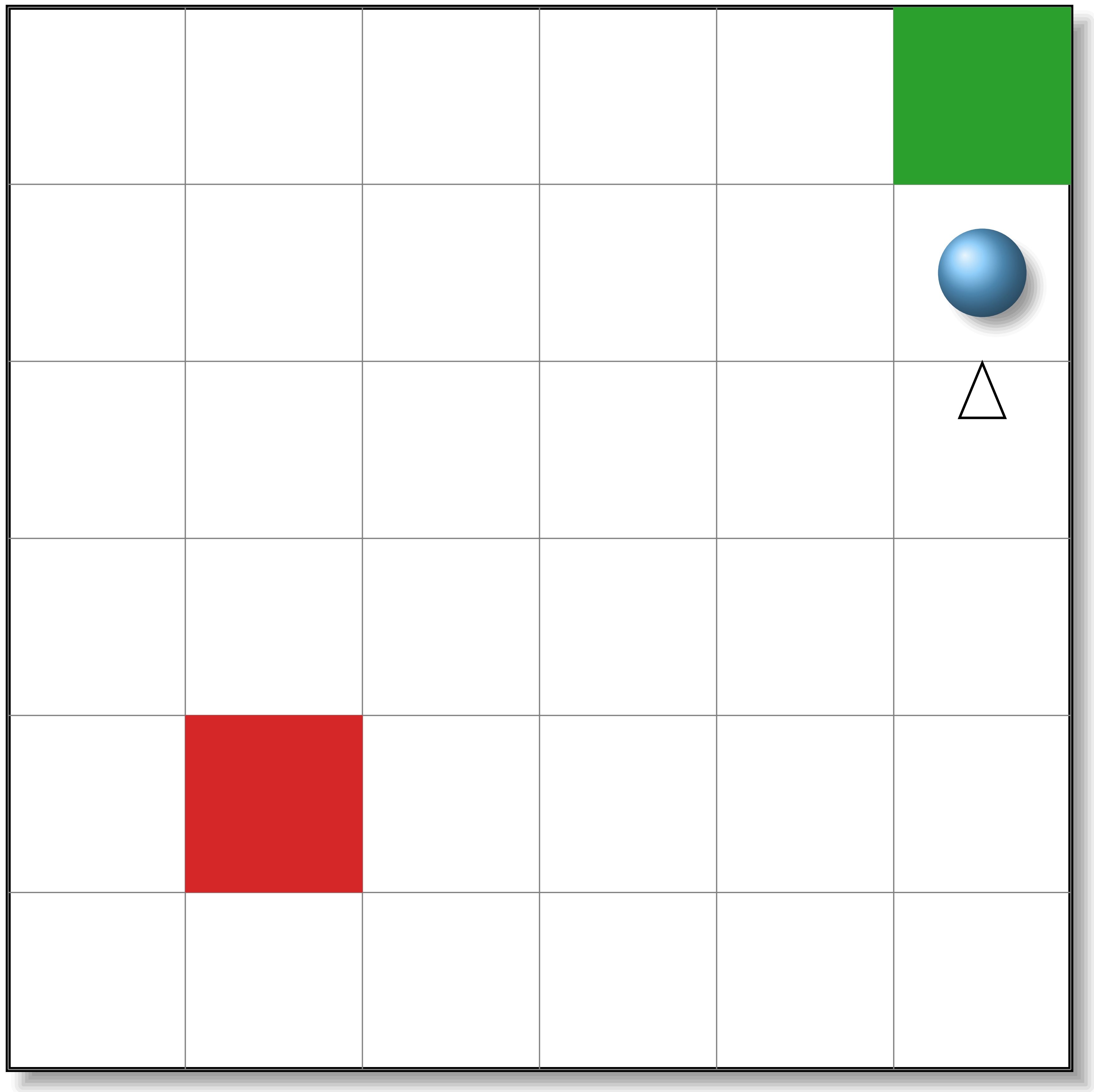
standard tabular Q-Learning



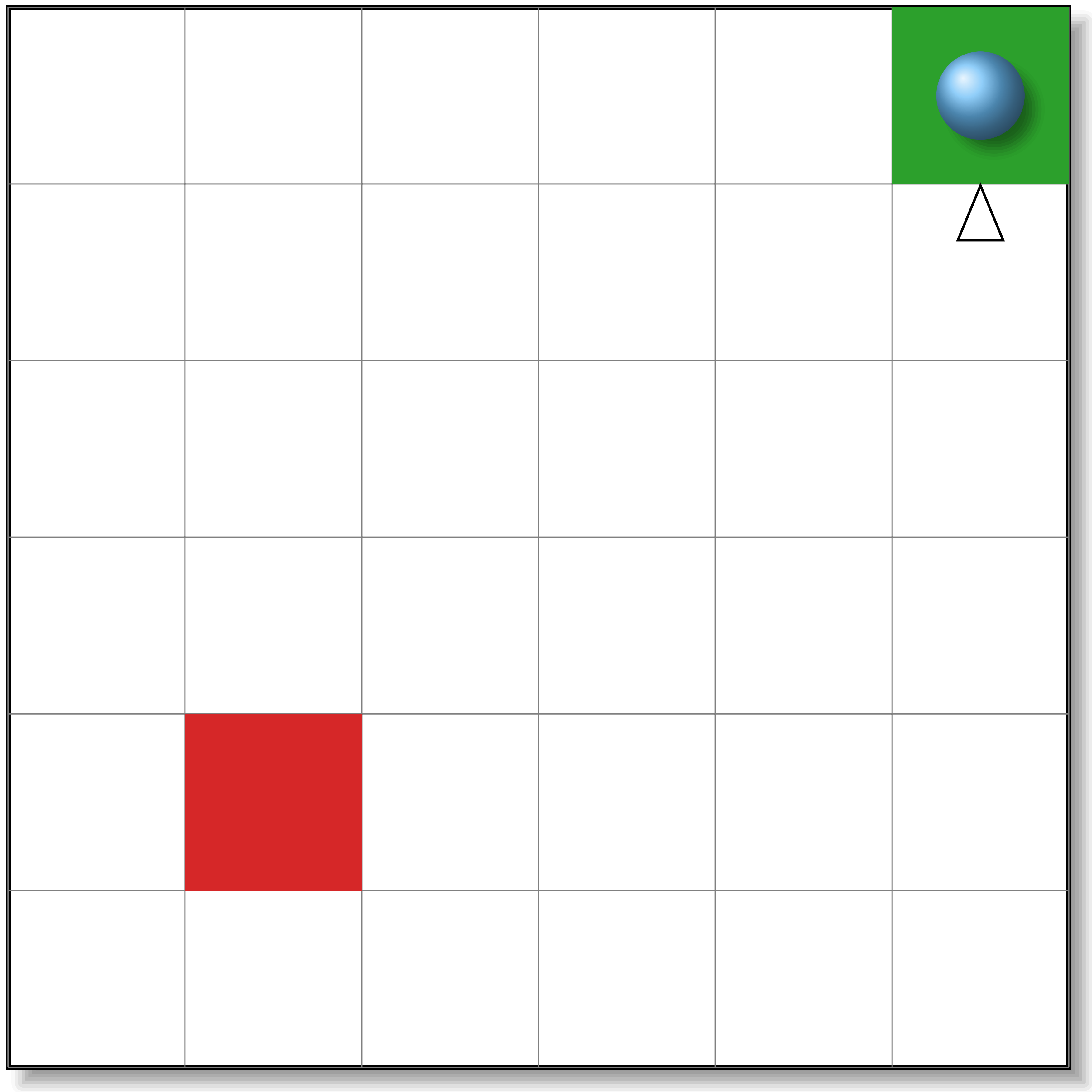
standard tabular Q-Learning



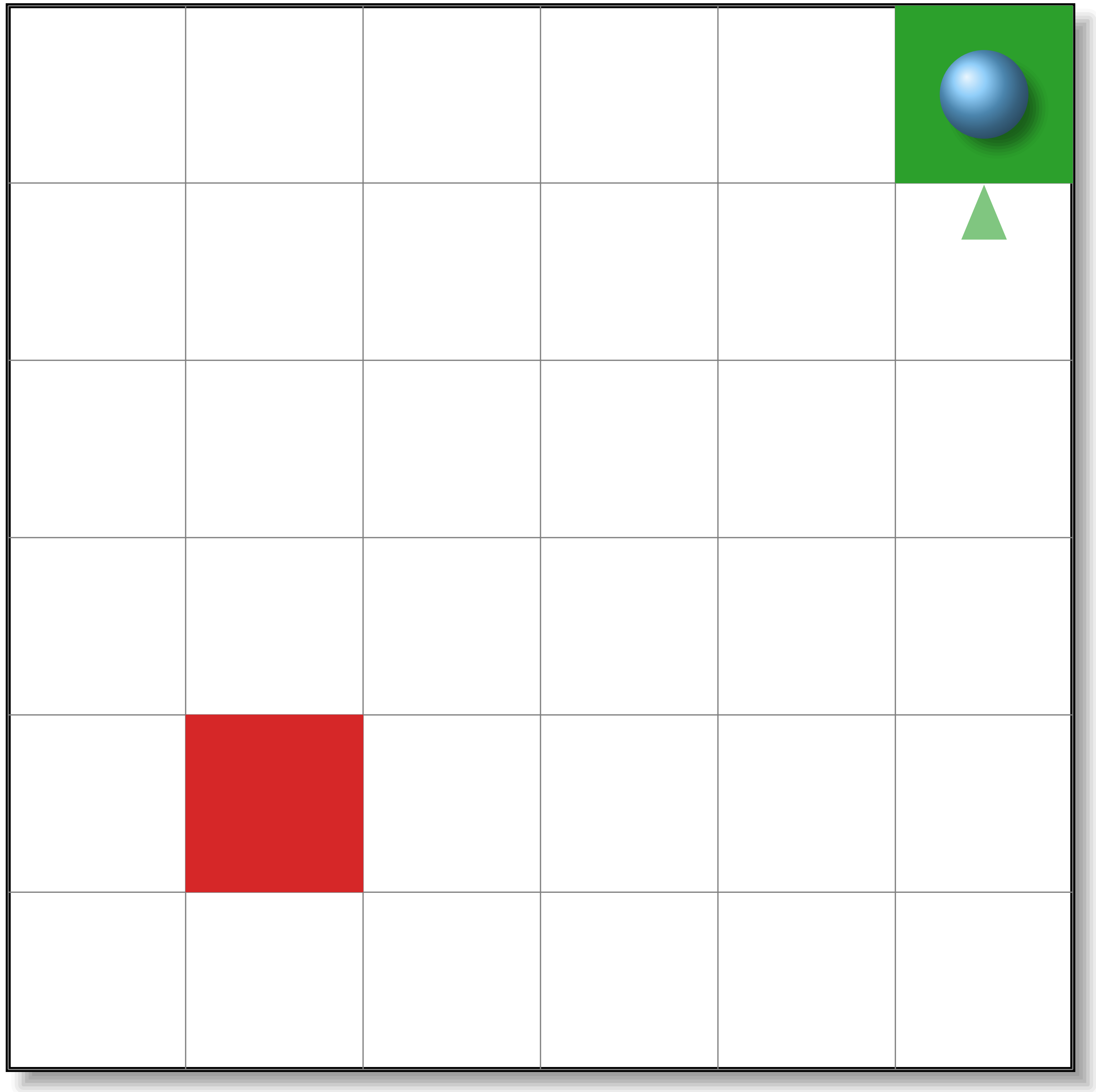
standard tabular Q-Learning



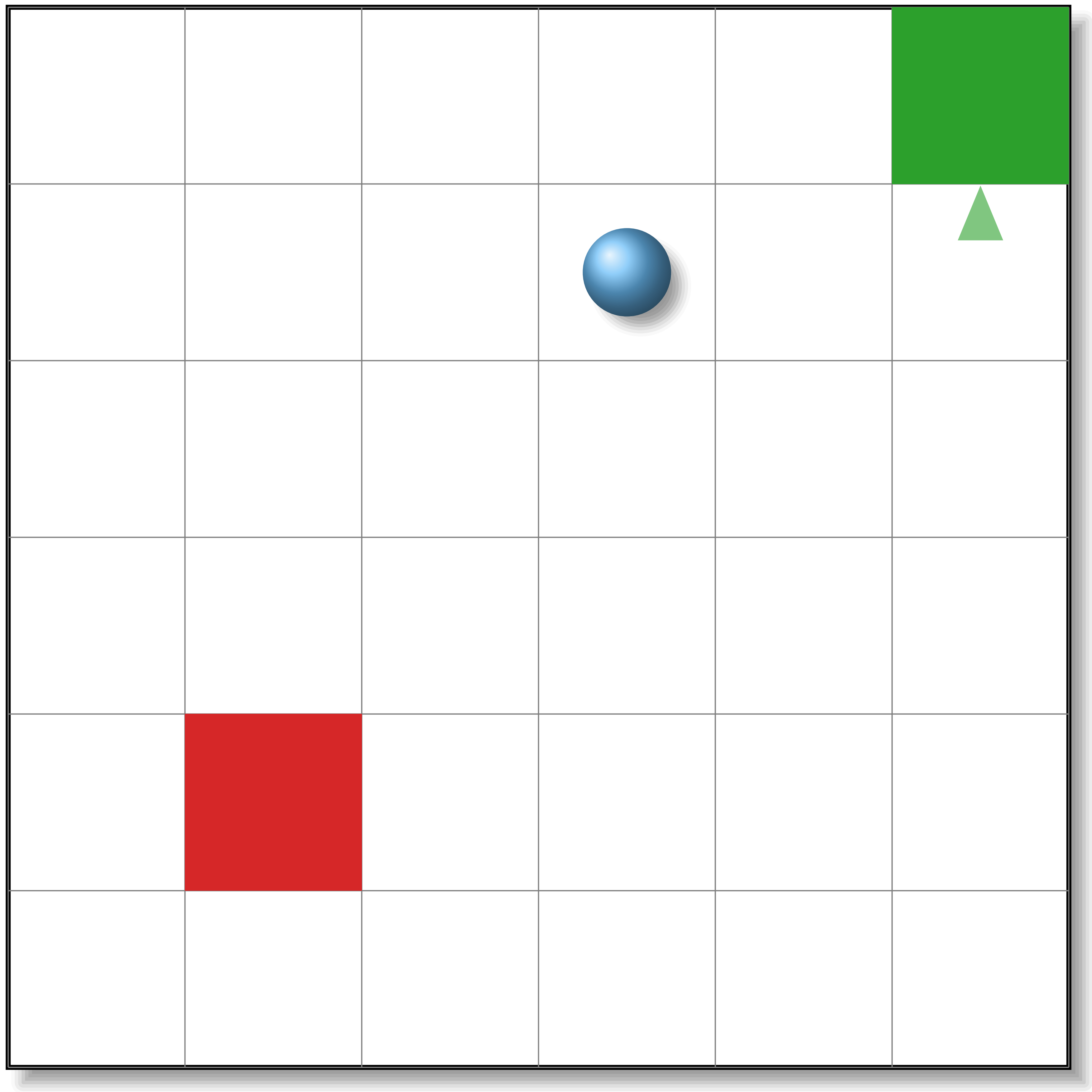
standard tabular Q-Learning



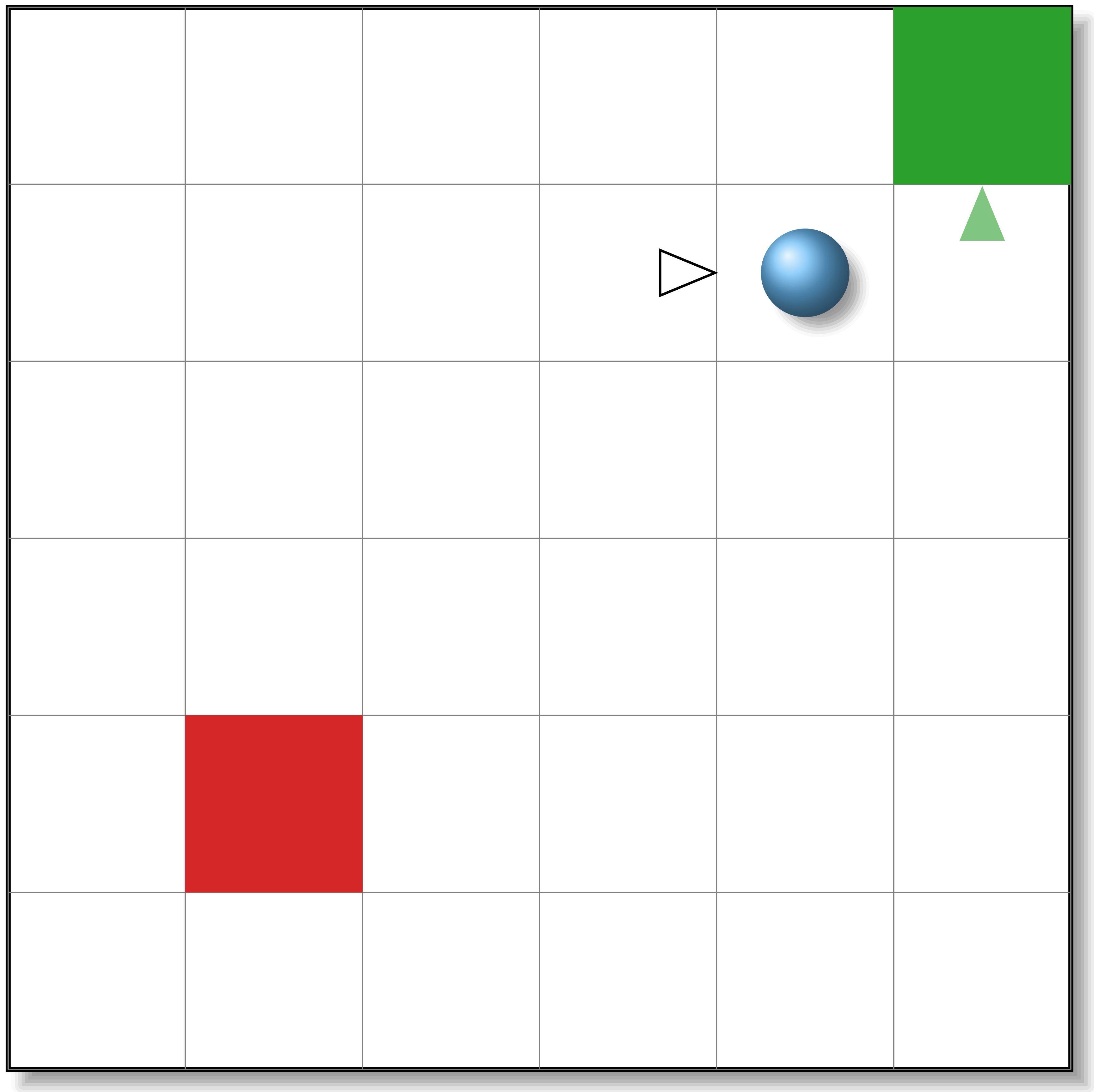
standard tabular Q-Learning



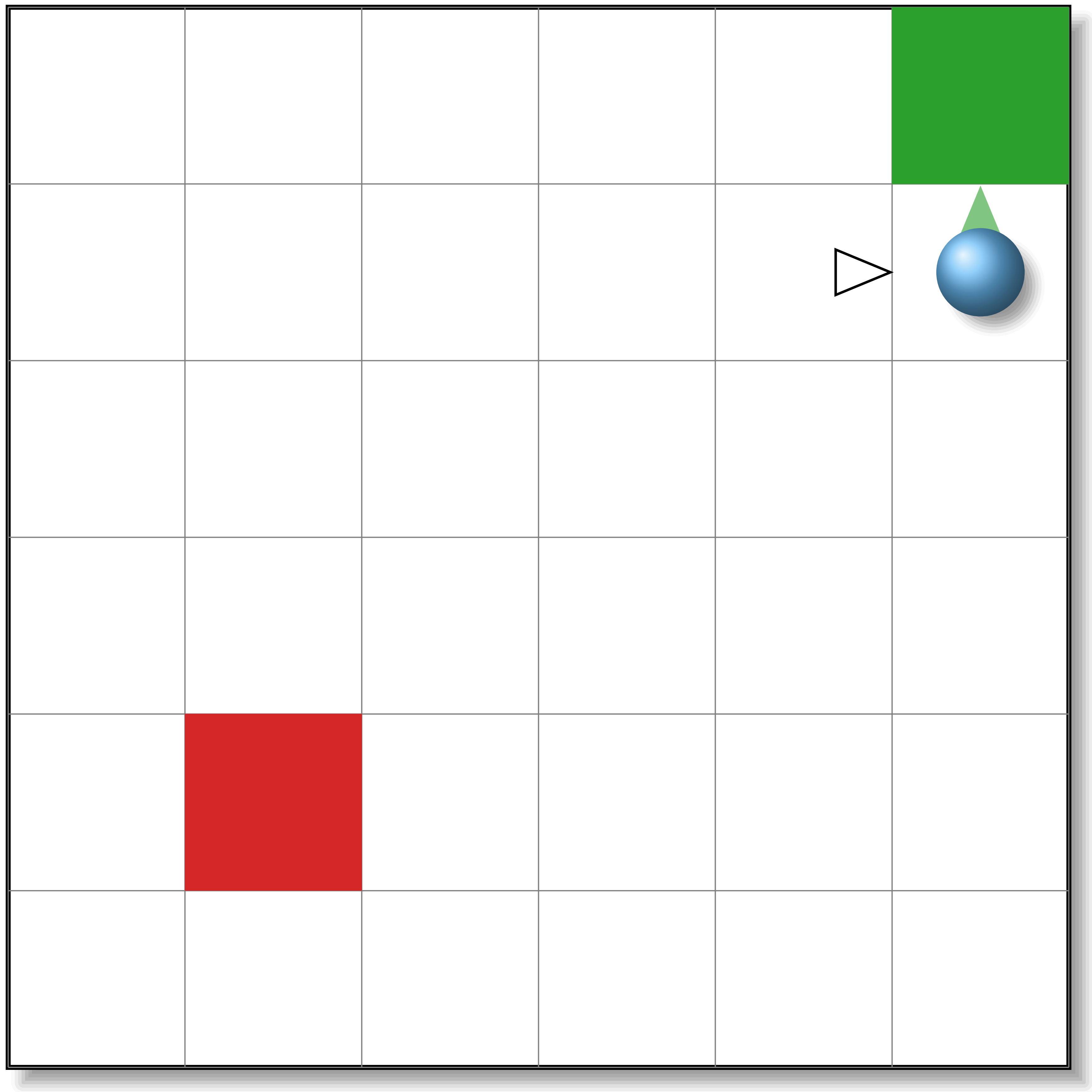
standard tabular Q-Learning



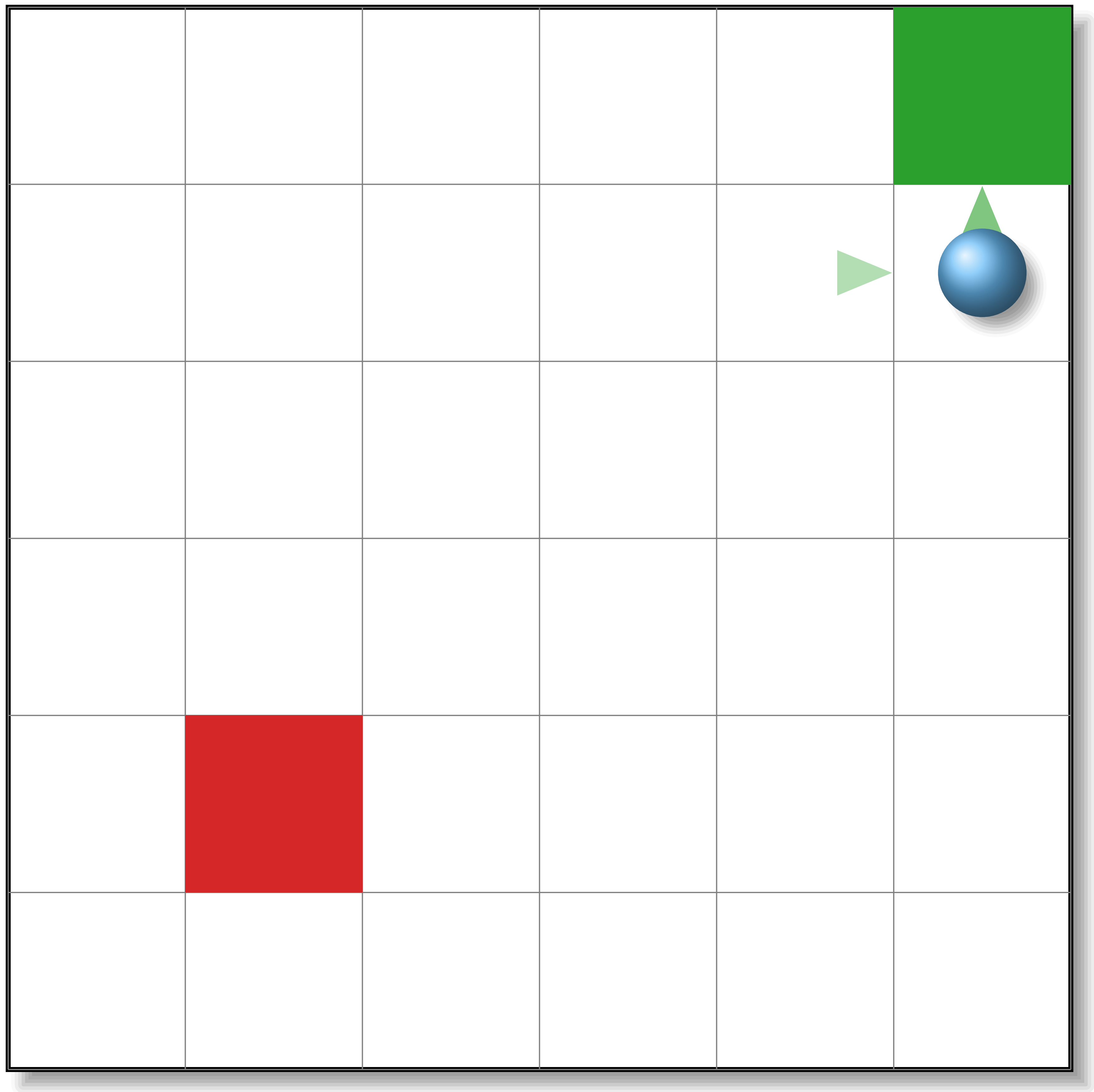
standard tabular Q-Learning



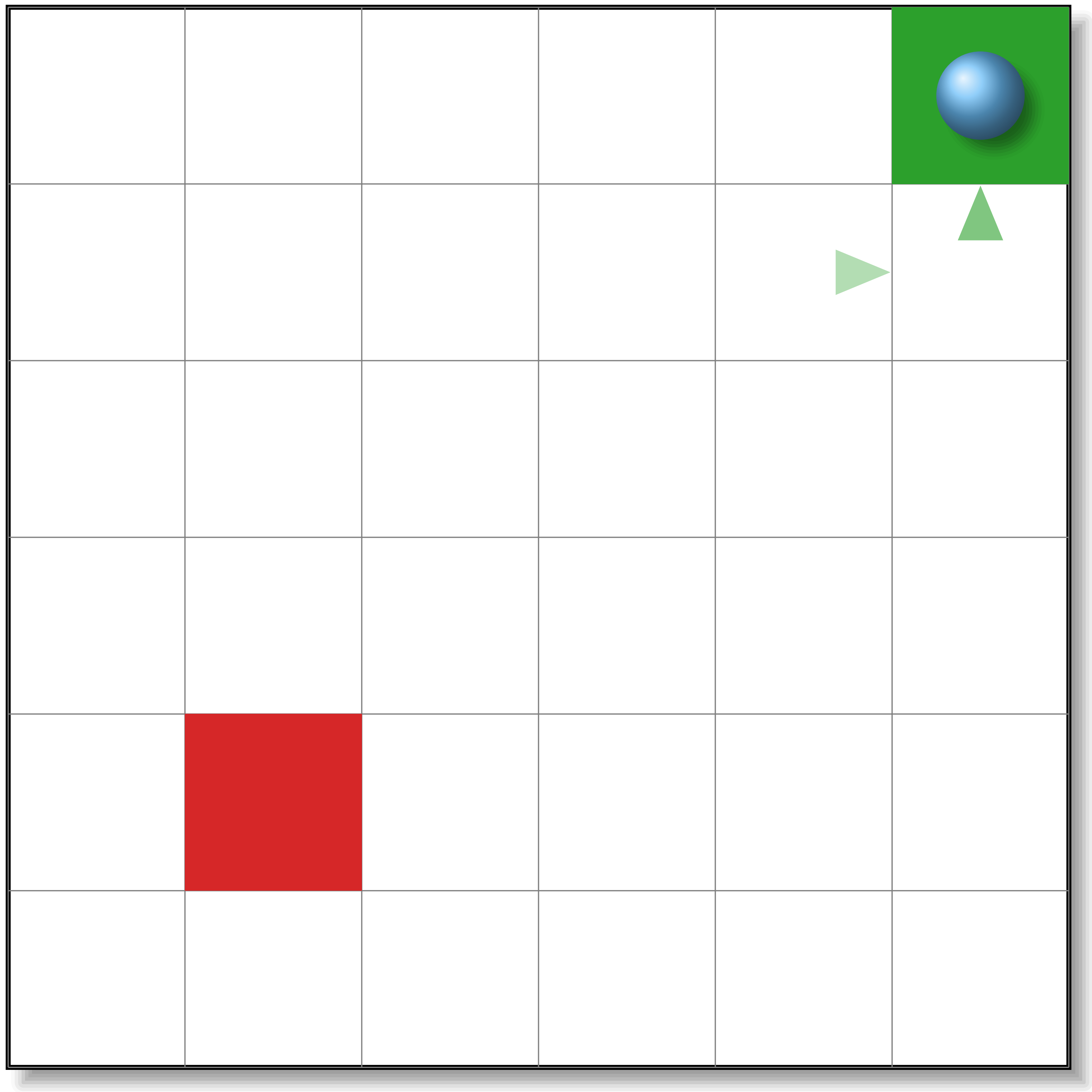
standard tabular Q-Learning



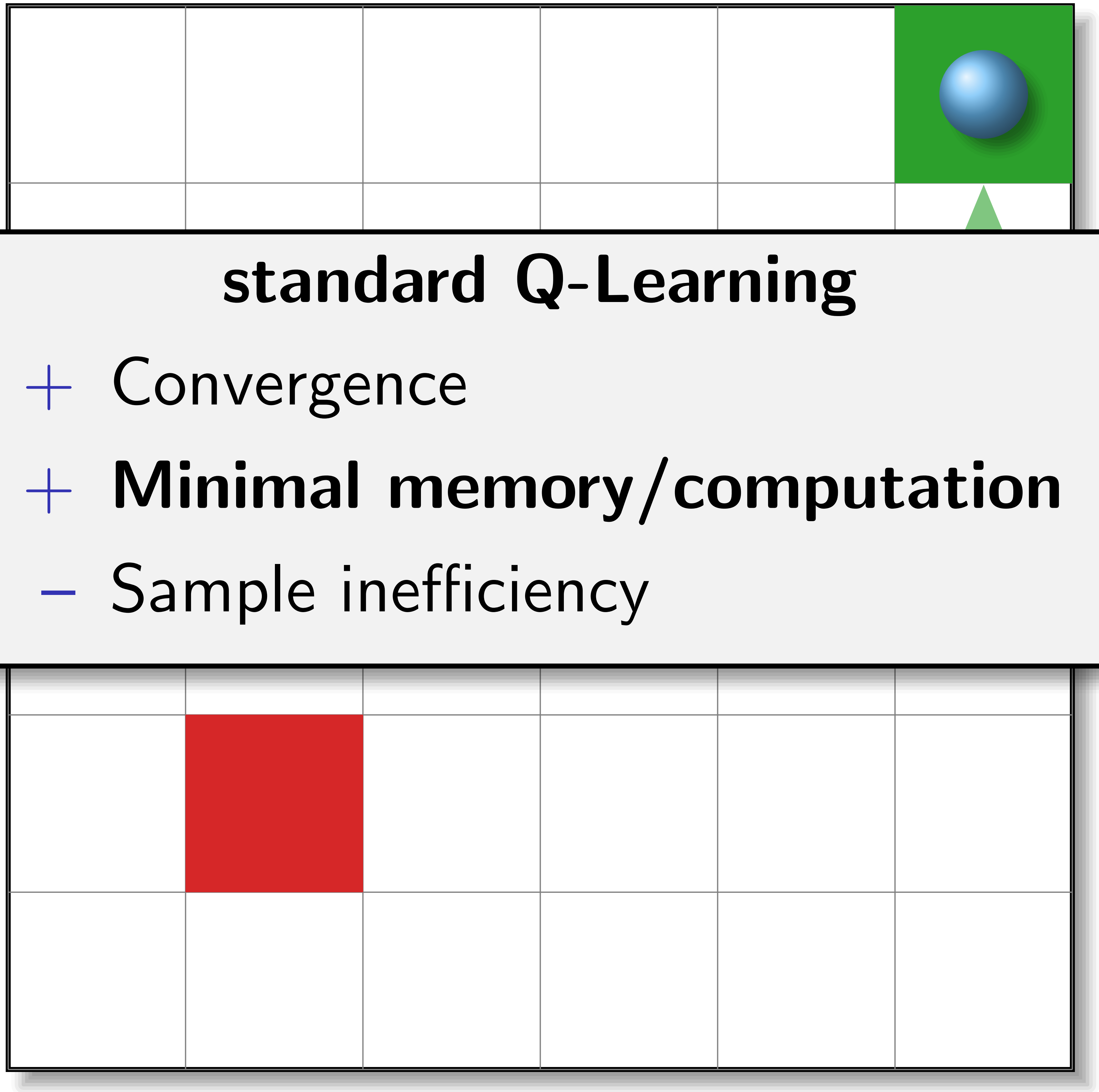
standard tabular Q-Learning



standard tabular Q-Learning

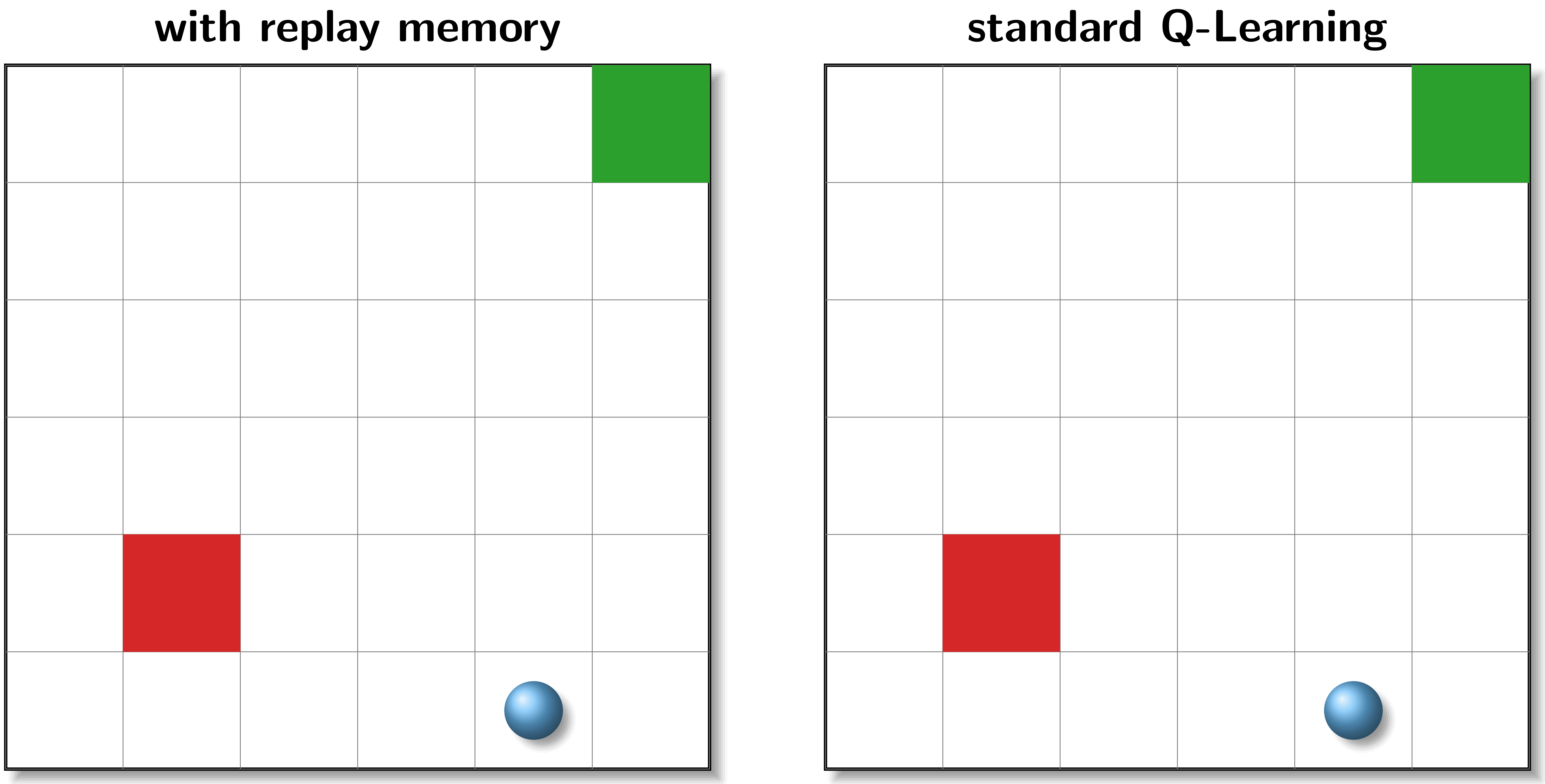


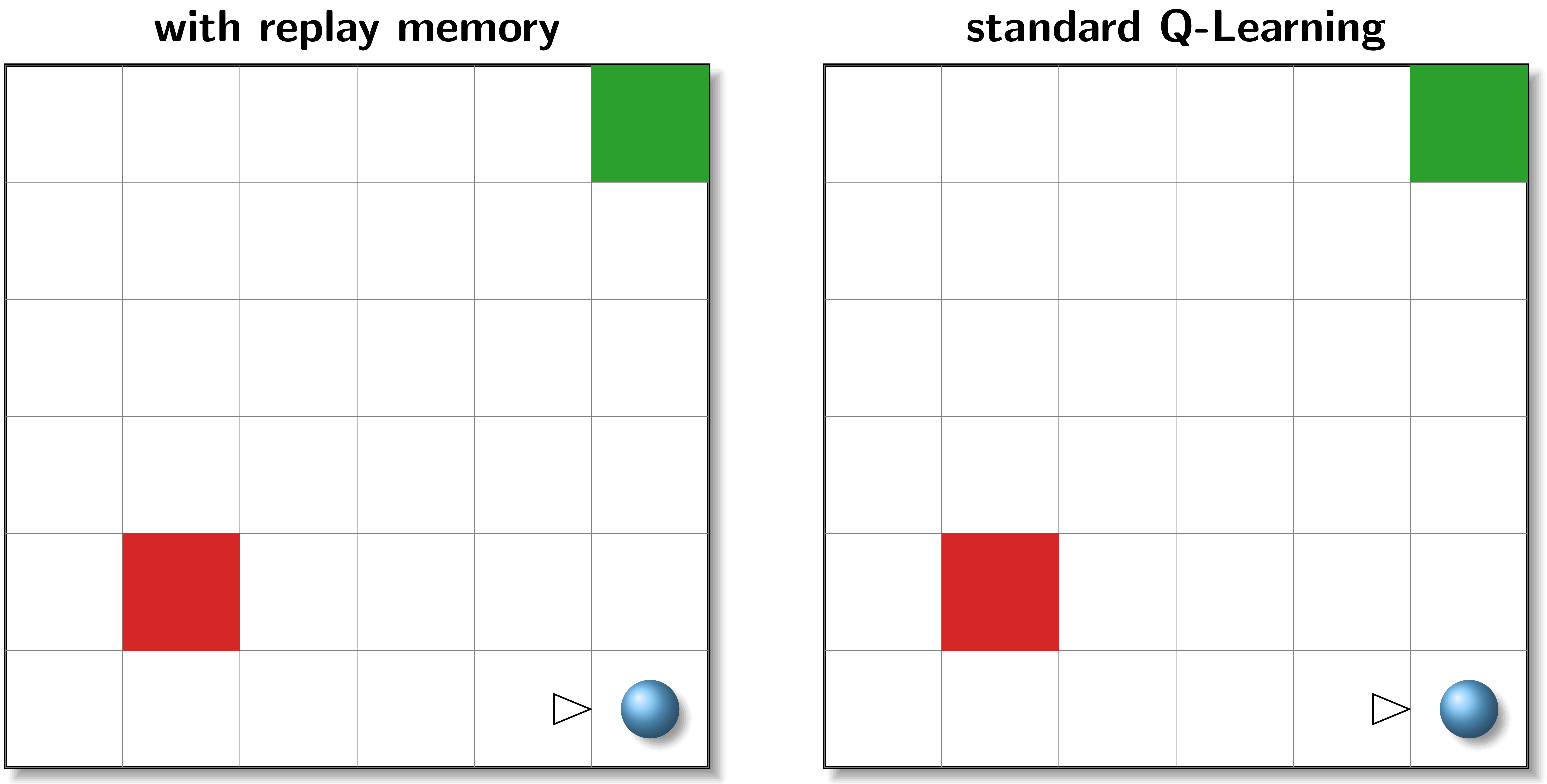
standard tabular Q-Learning

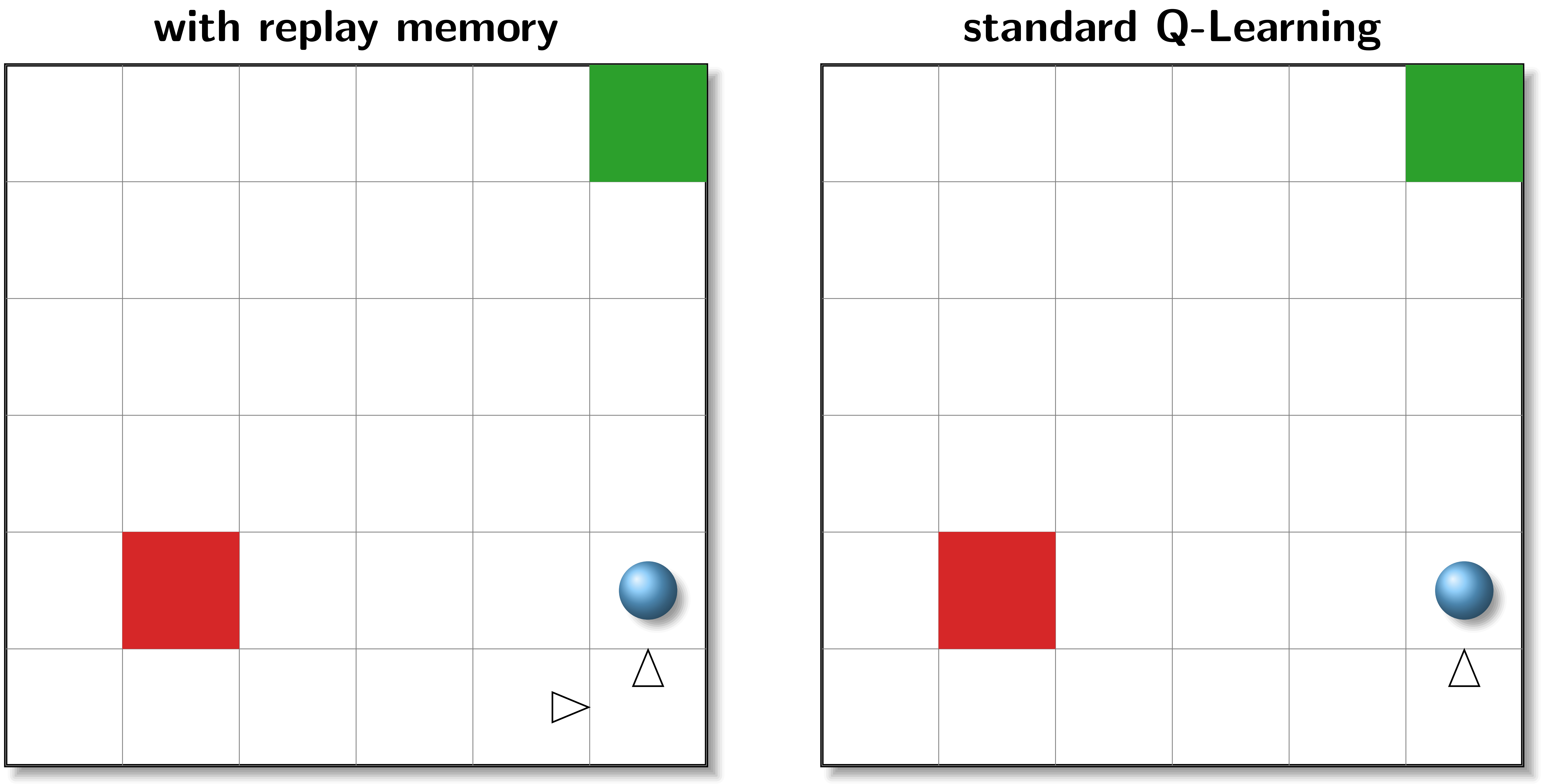


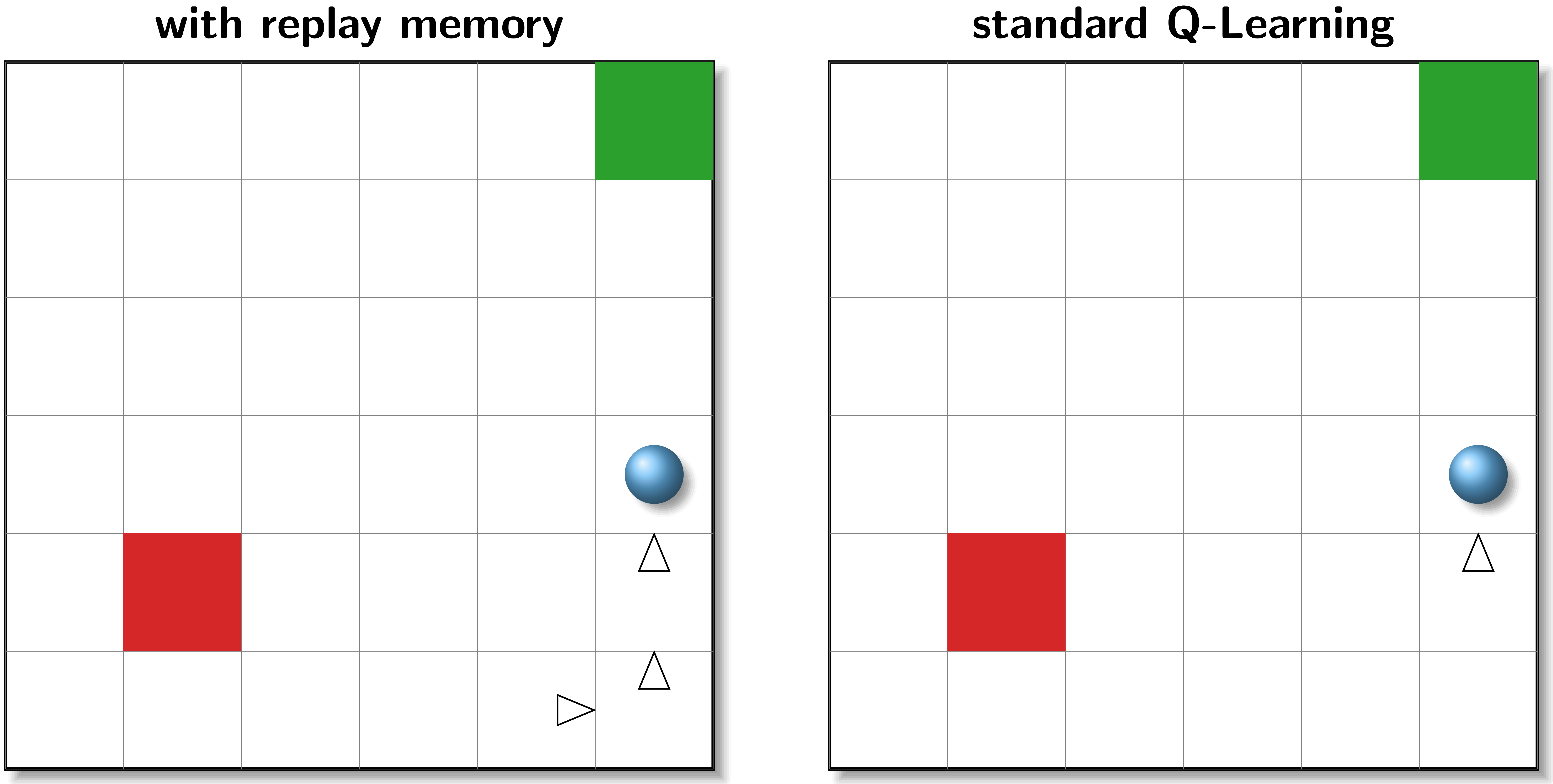
standard Q-Learning

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


with replay memory
standard Q-Learning

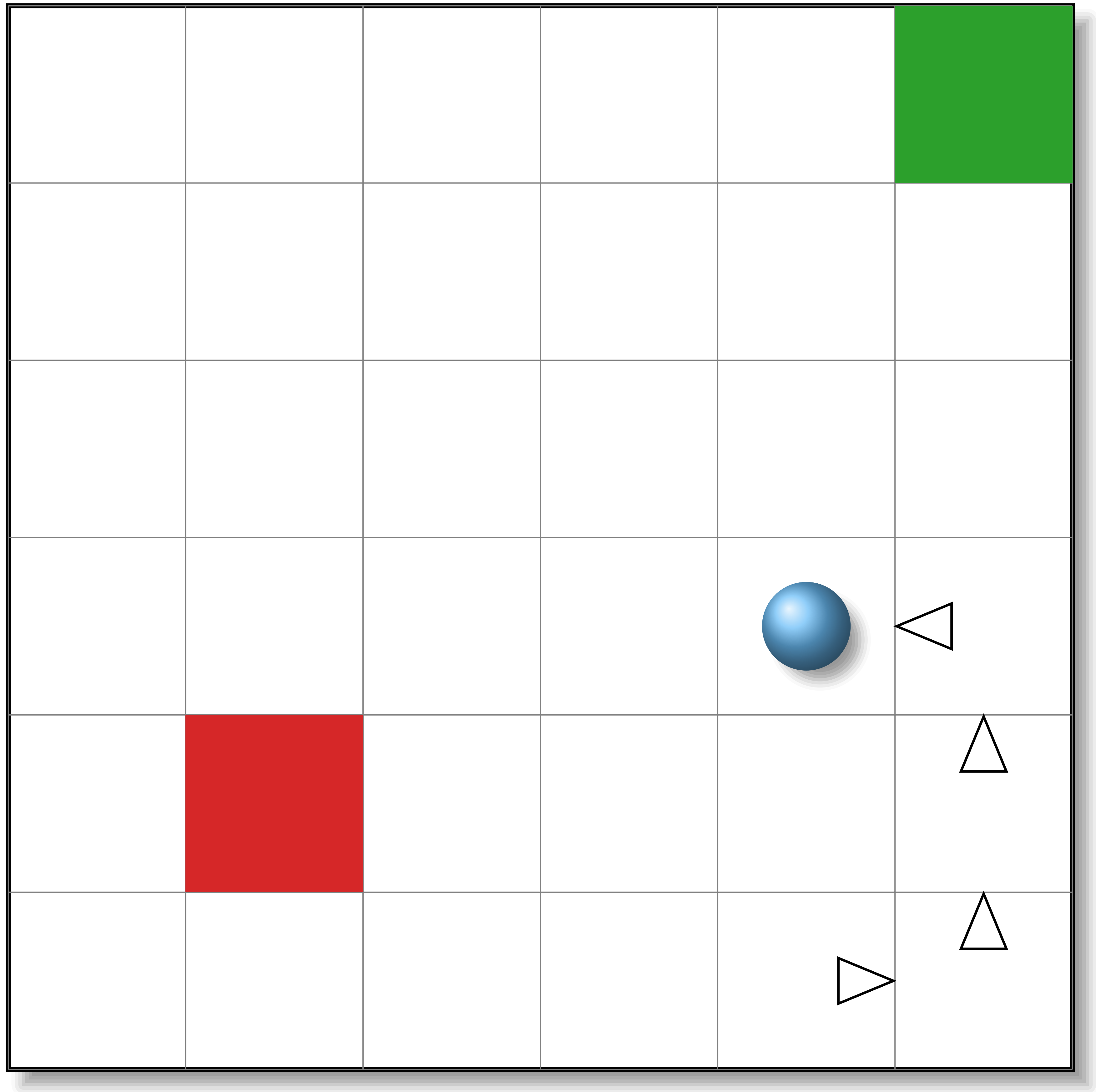

with replay memory
standard Q-Learning


with replay memory
standard Q-Learning

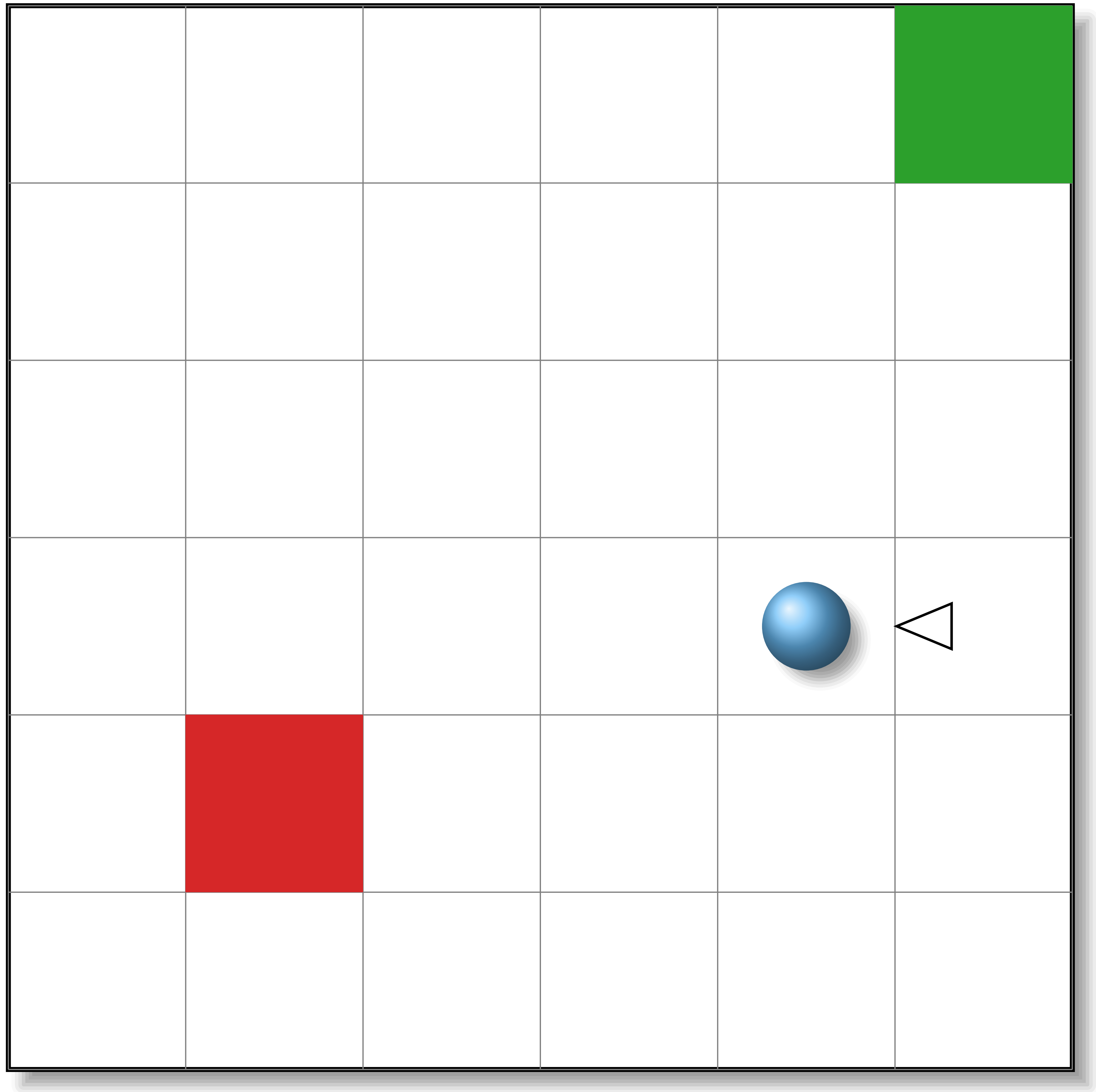


with replay memory
standard Q-Learning

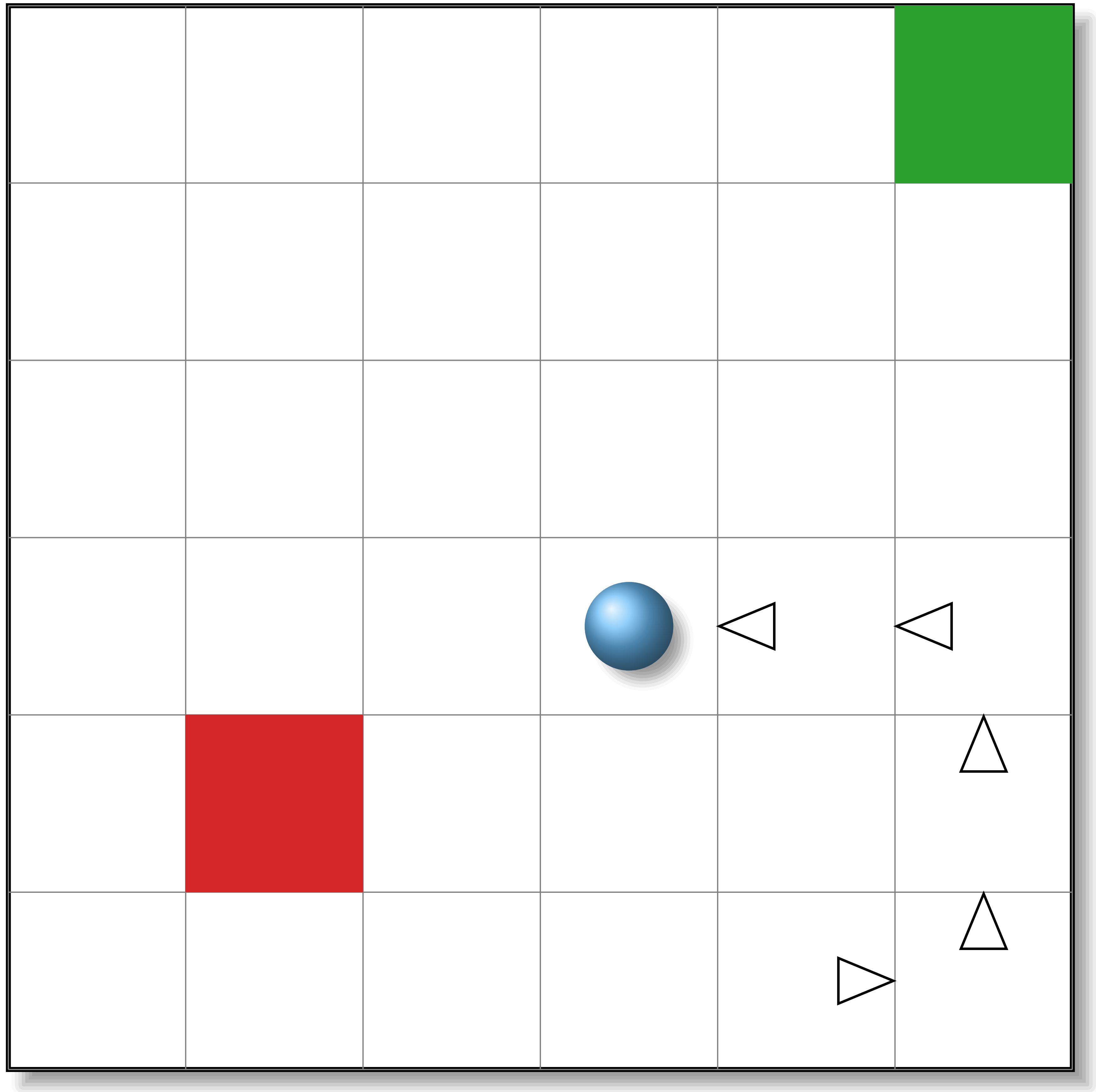
with replay memory



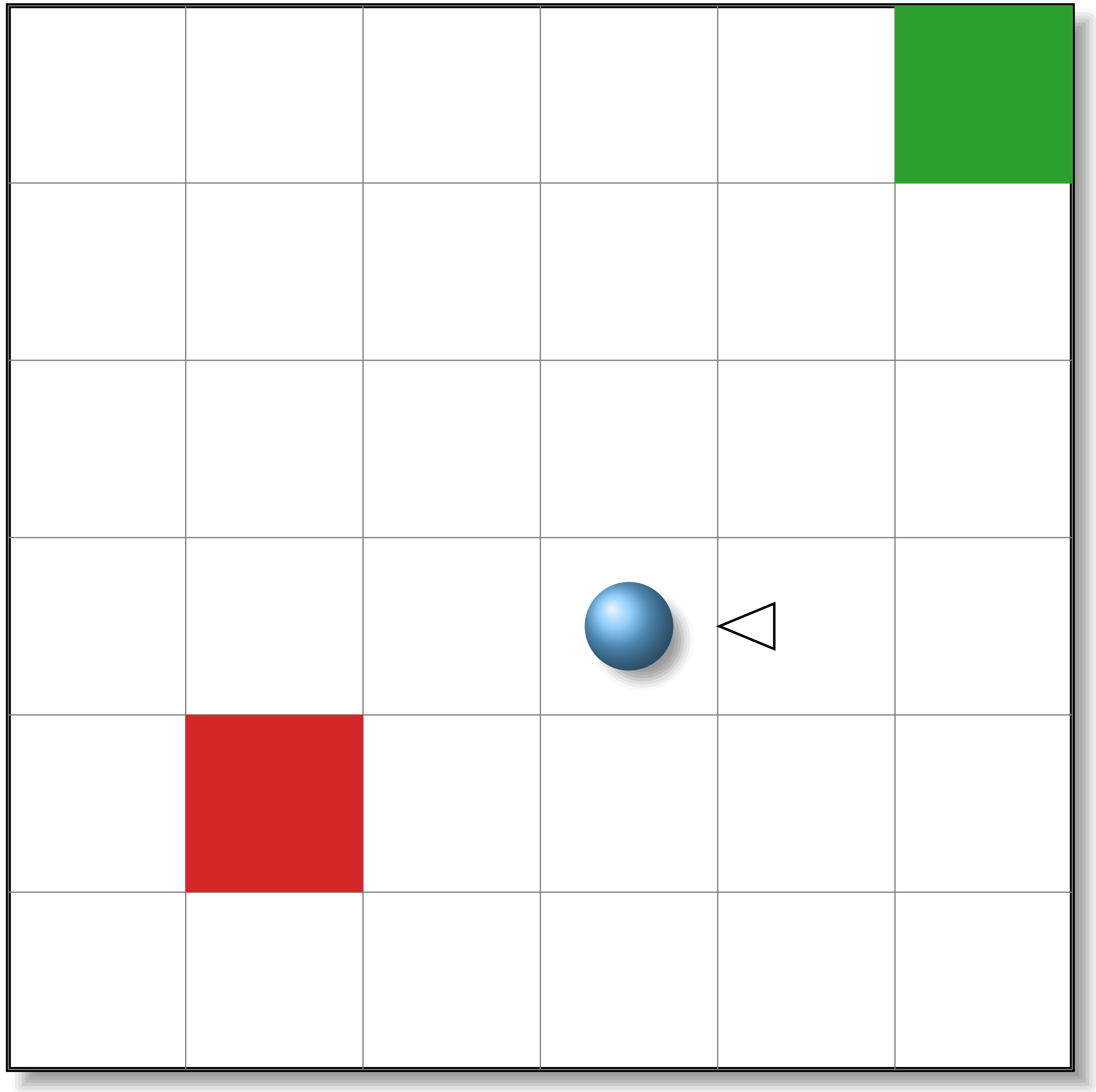
standard Q-Learning

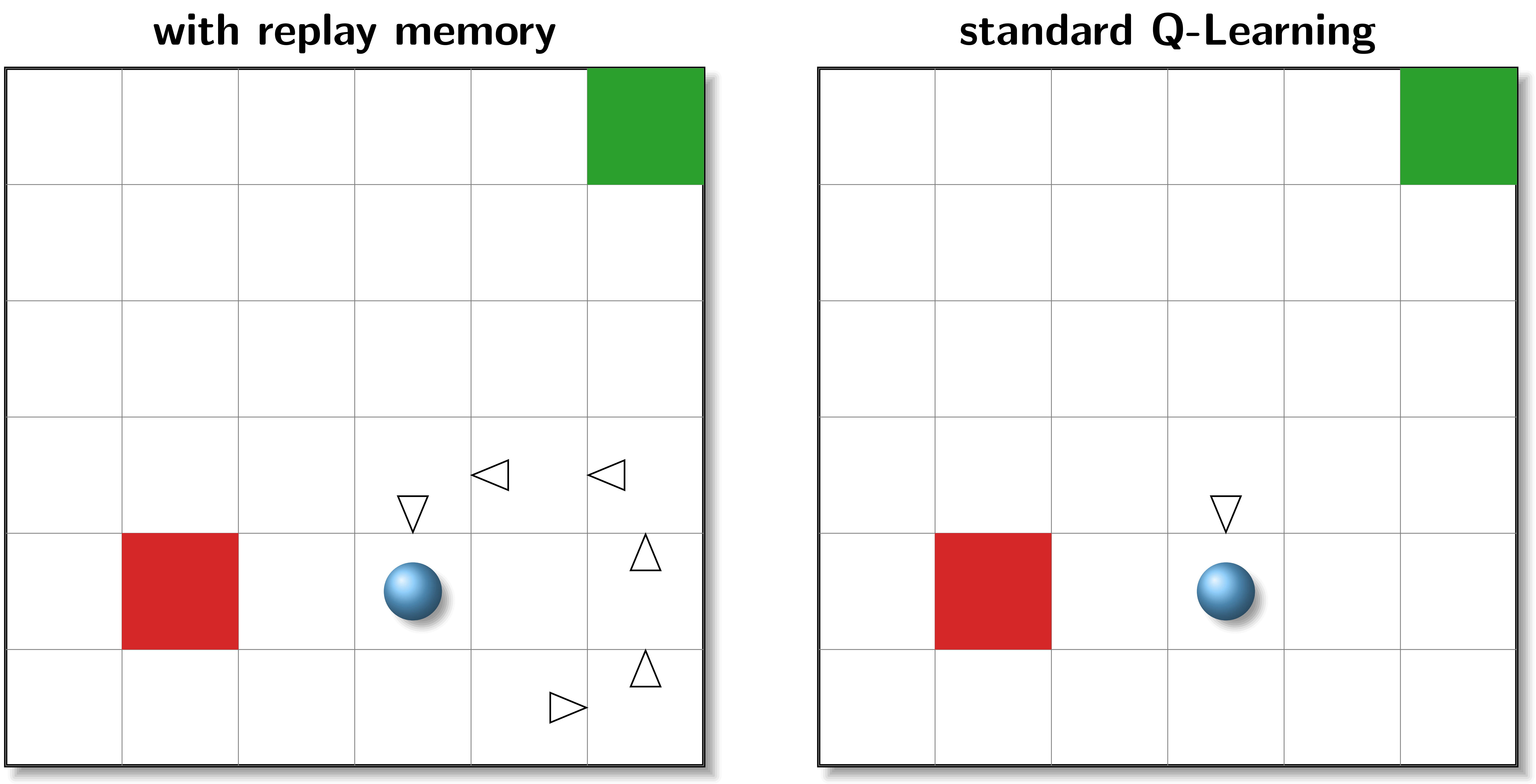


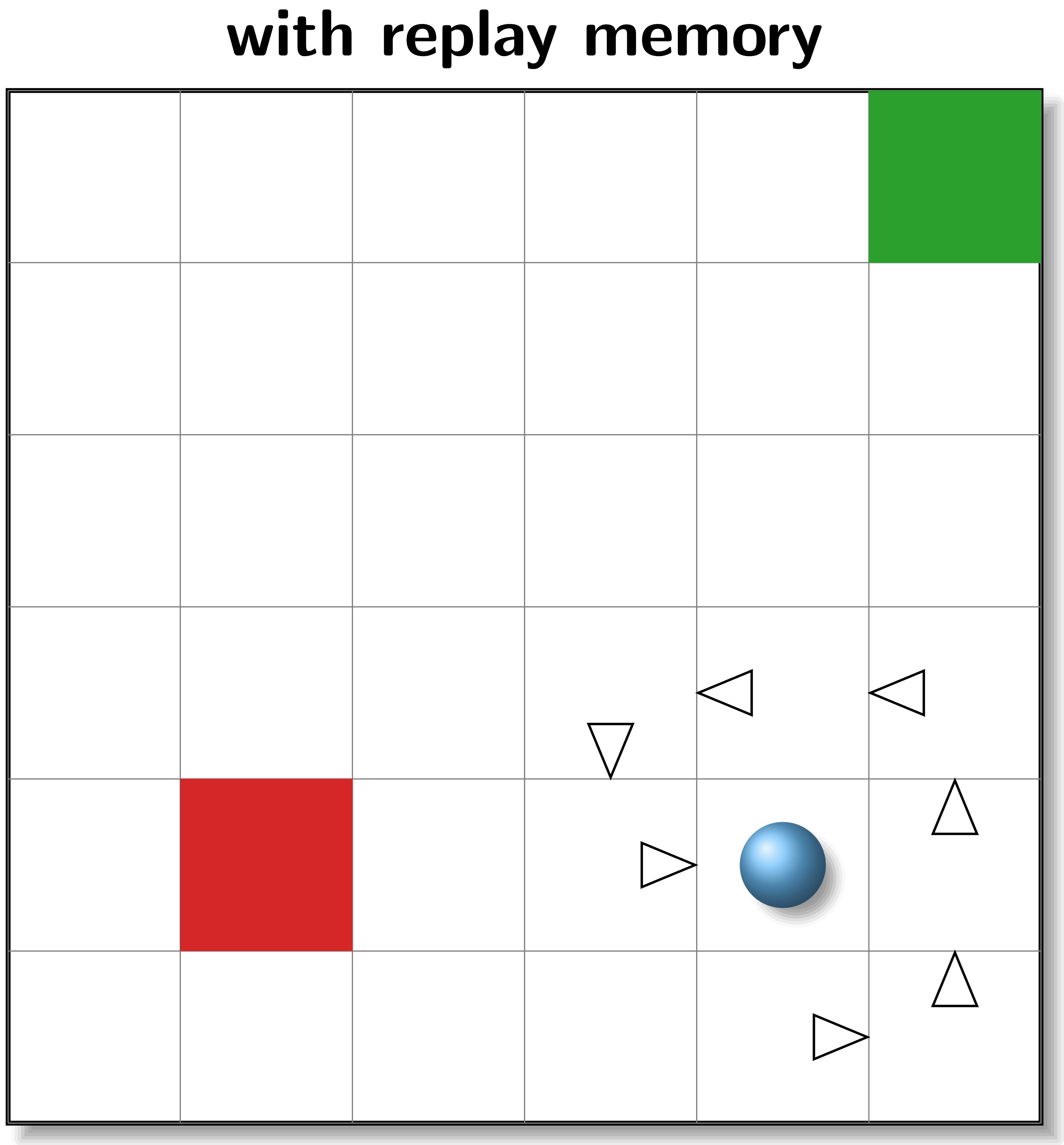
with replay memory

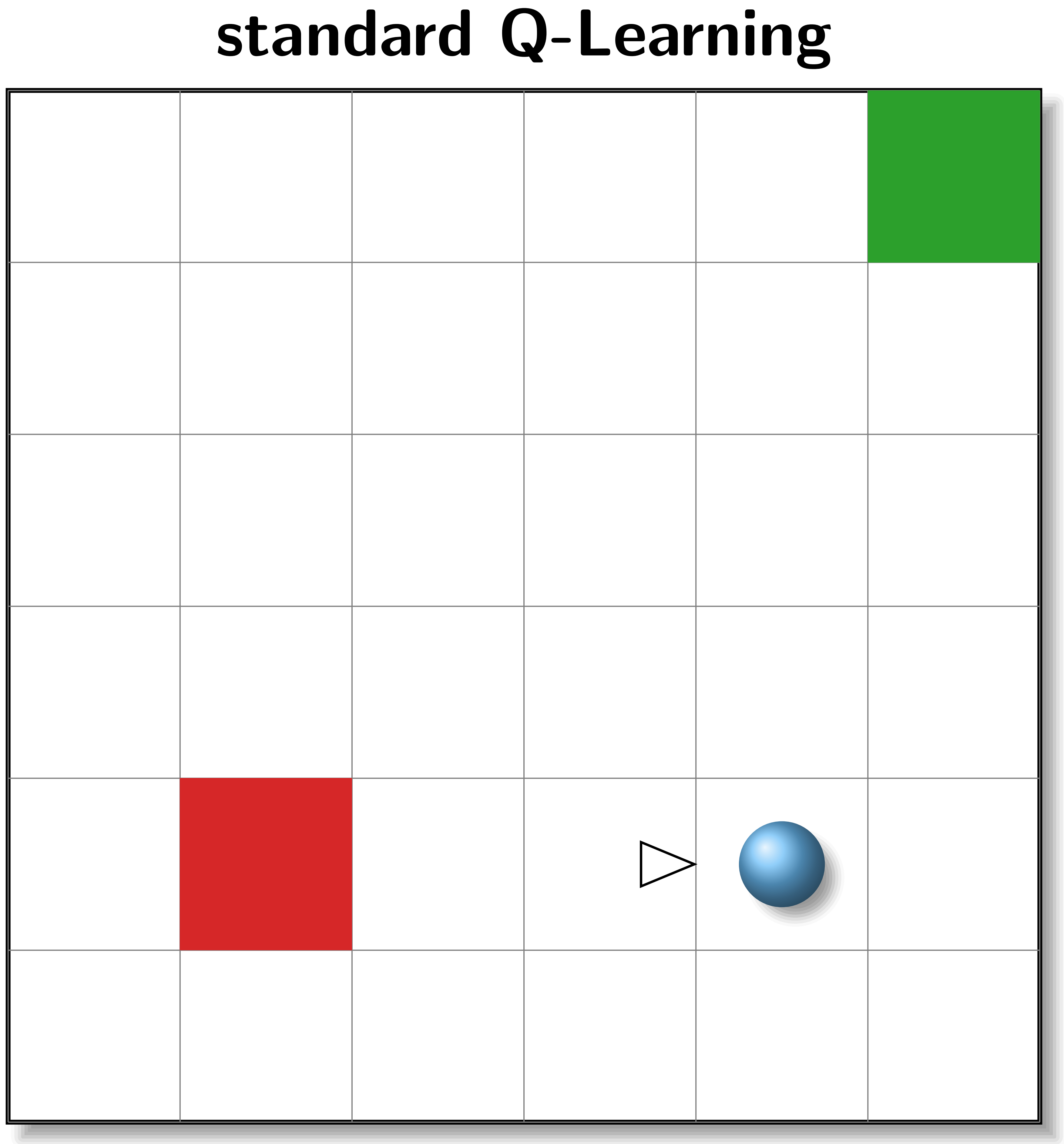


standard Q-Learning

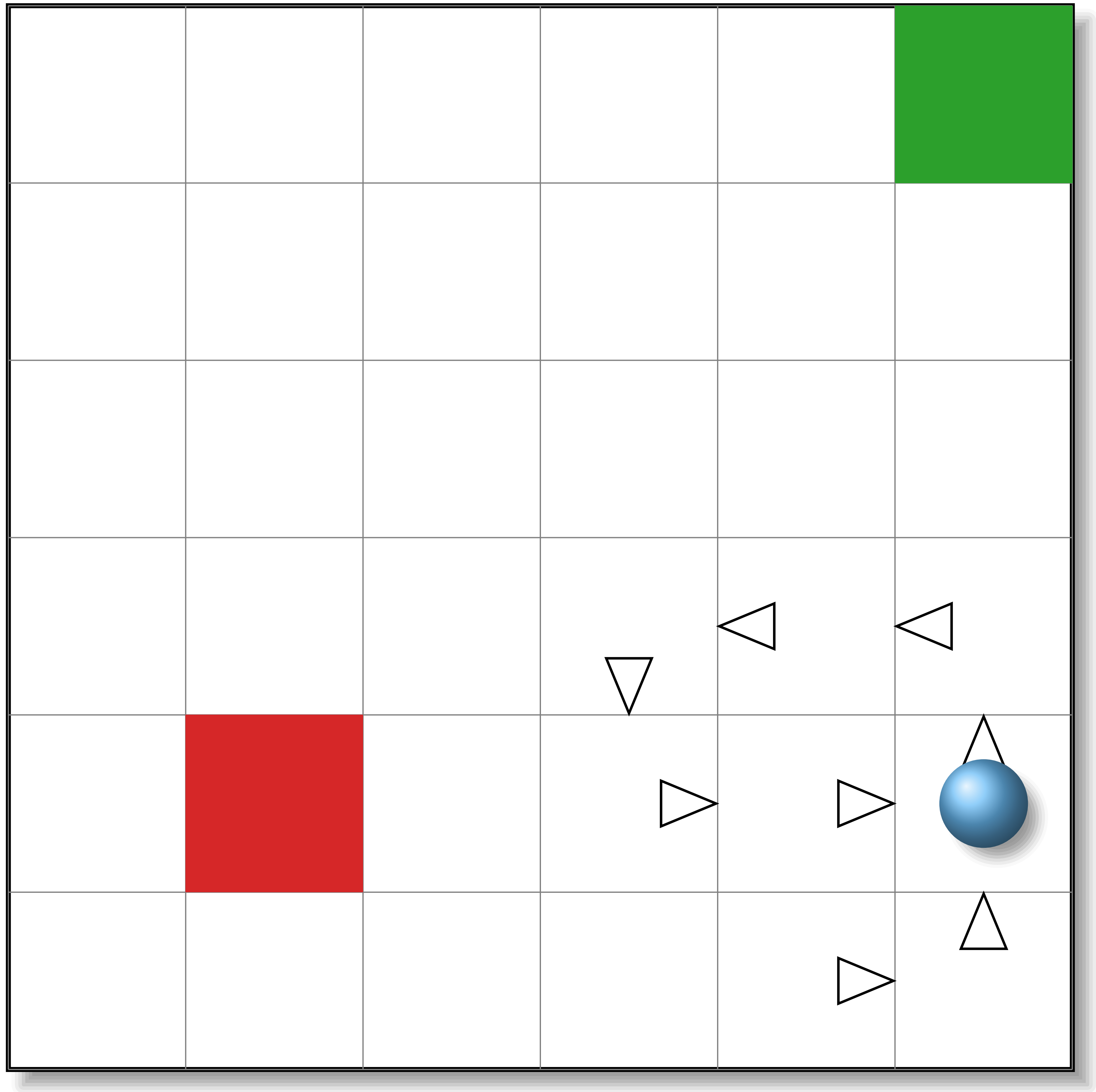



with replay memory
standard Q-Learning

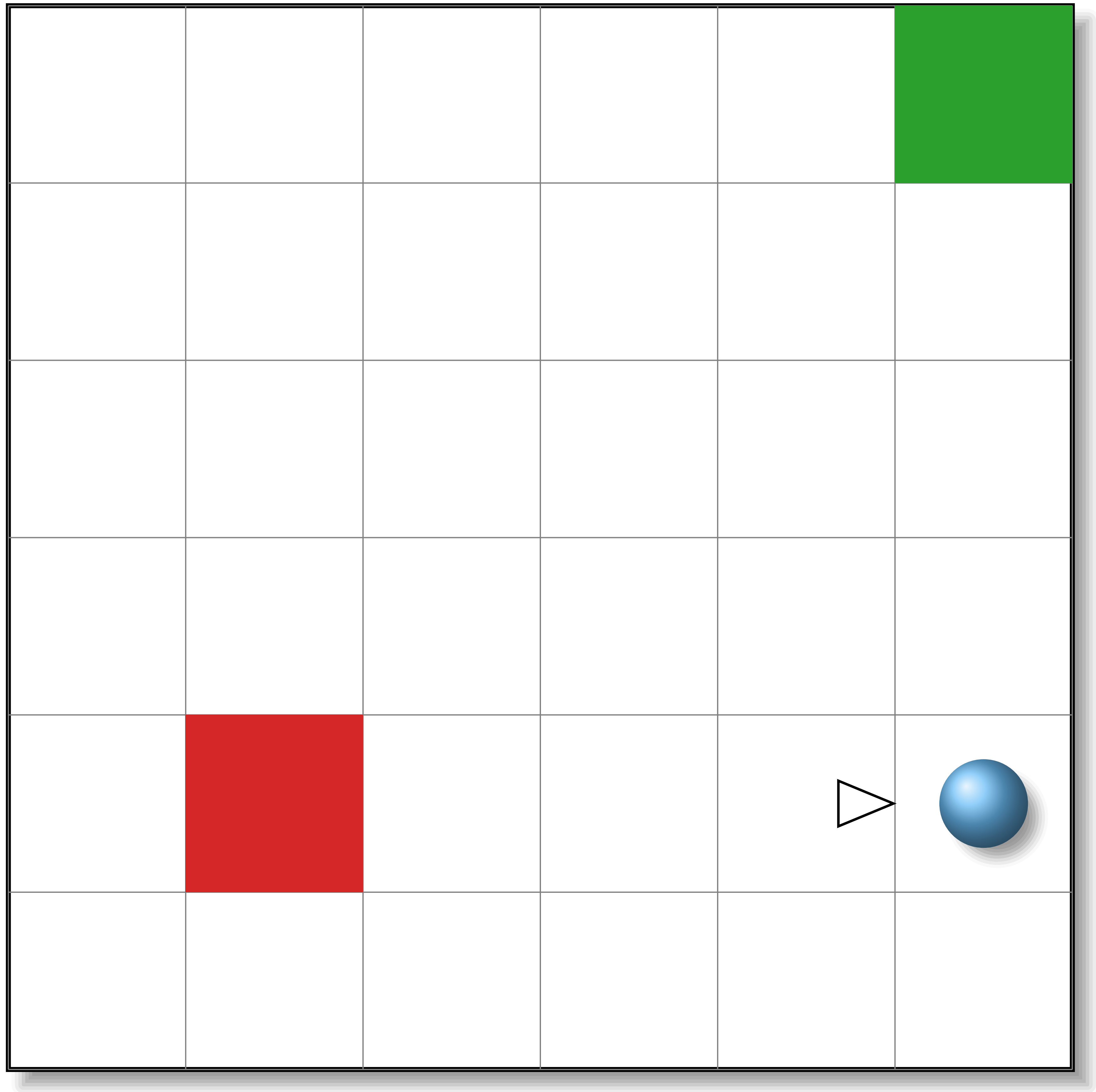

with replay memory


standard Q-Learning

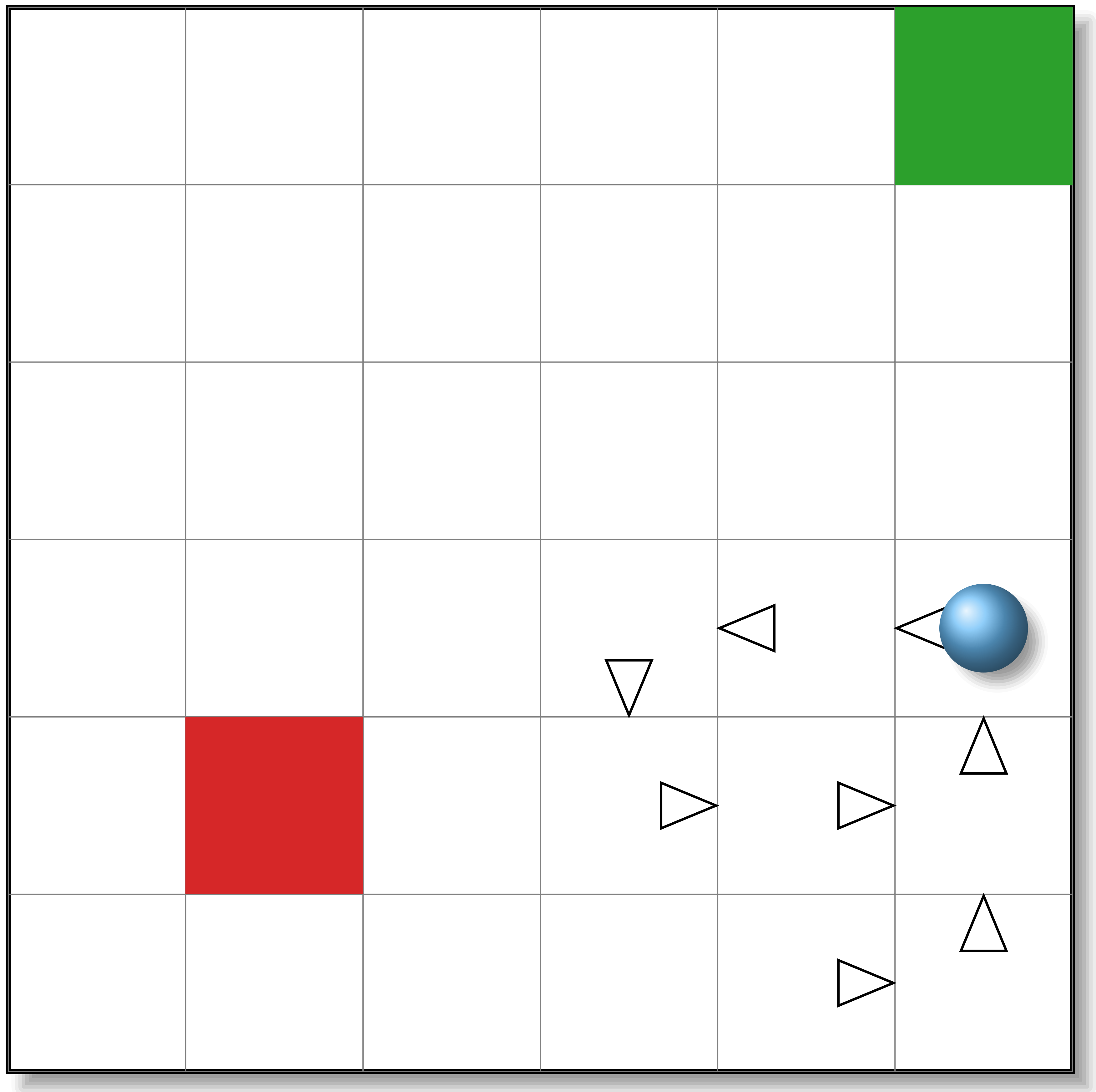
with replay memory



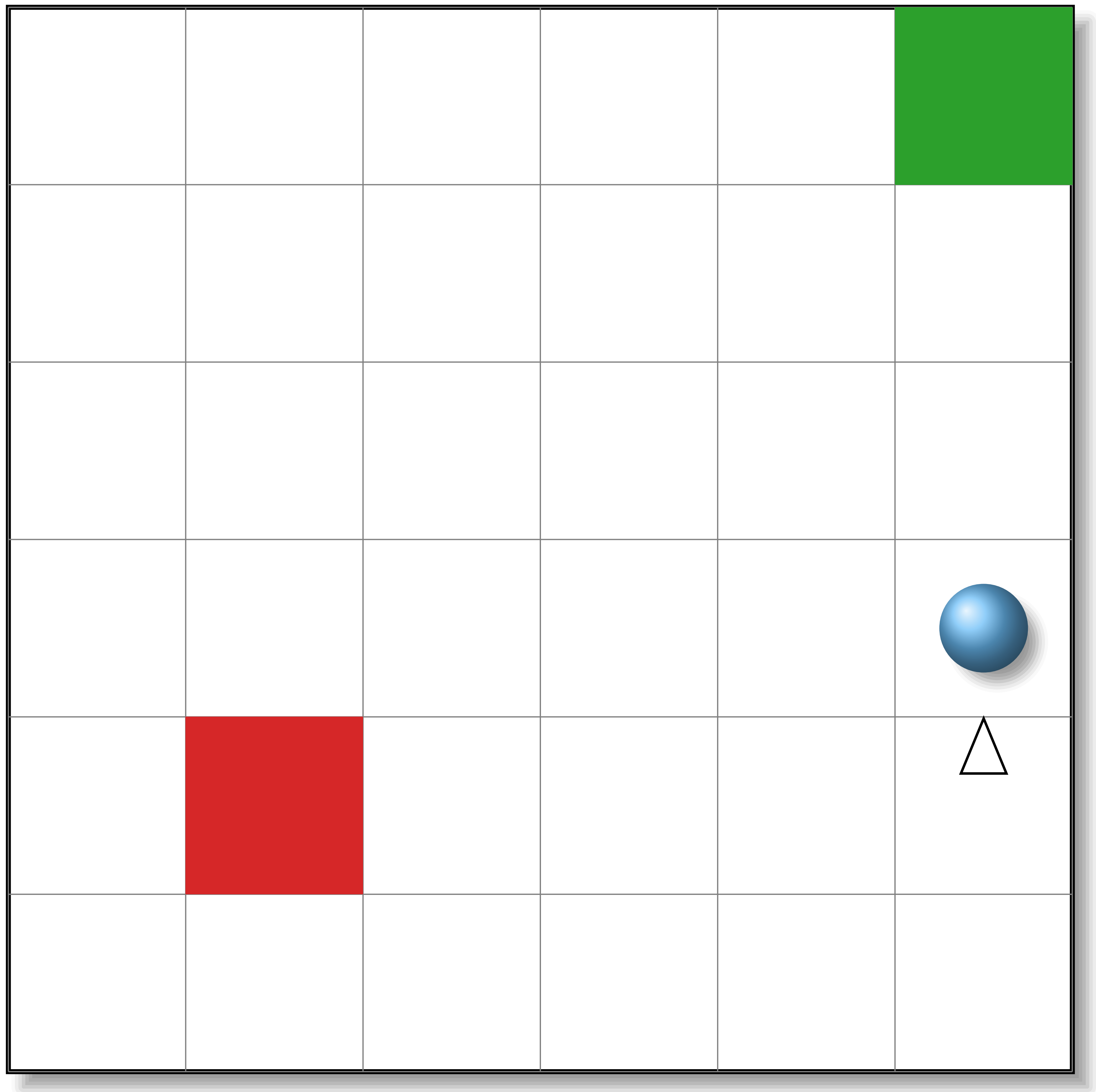
standard Q-Learning

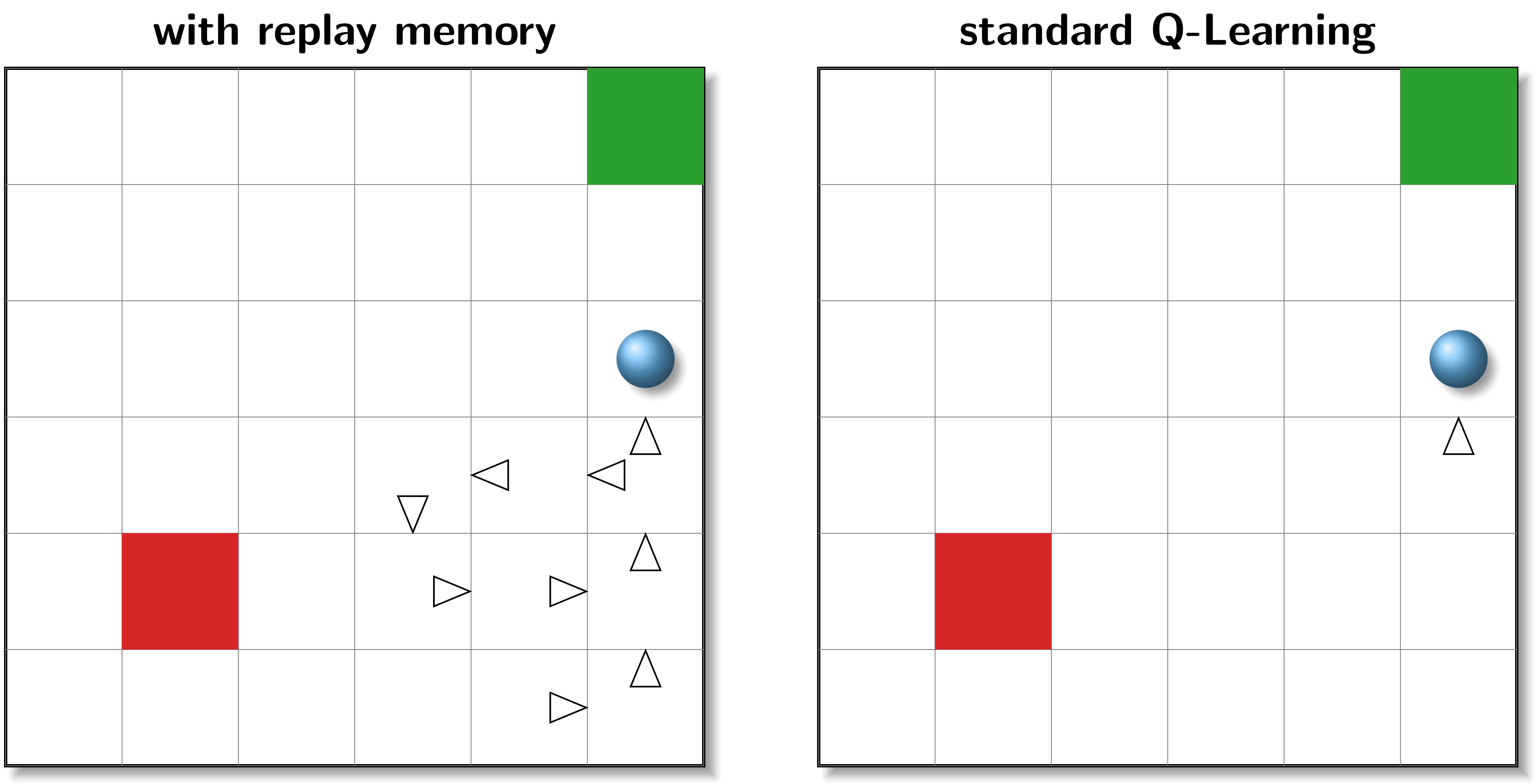


with replay memory

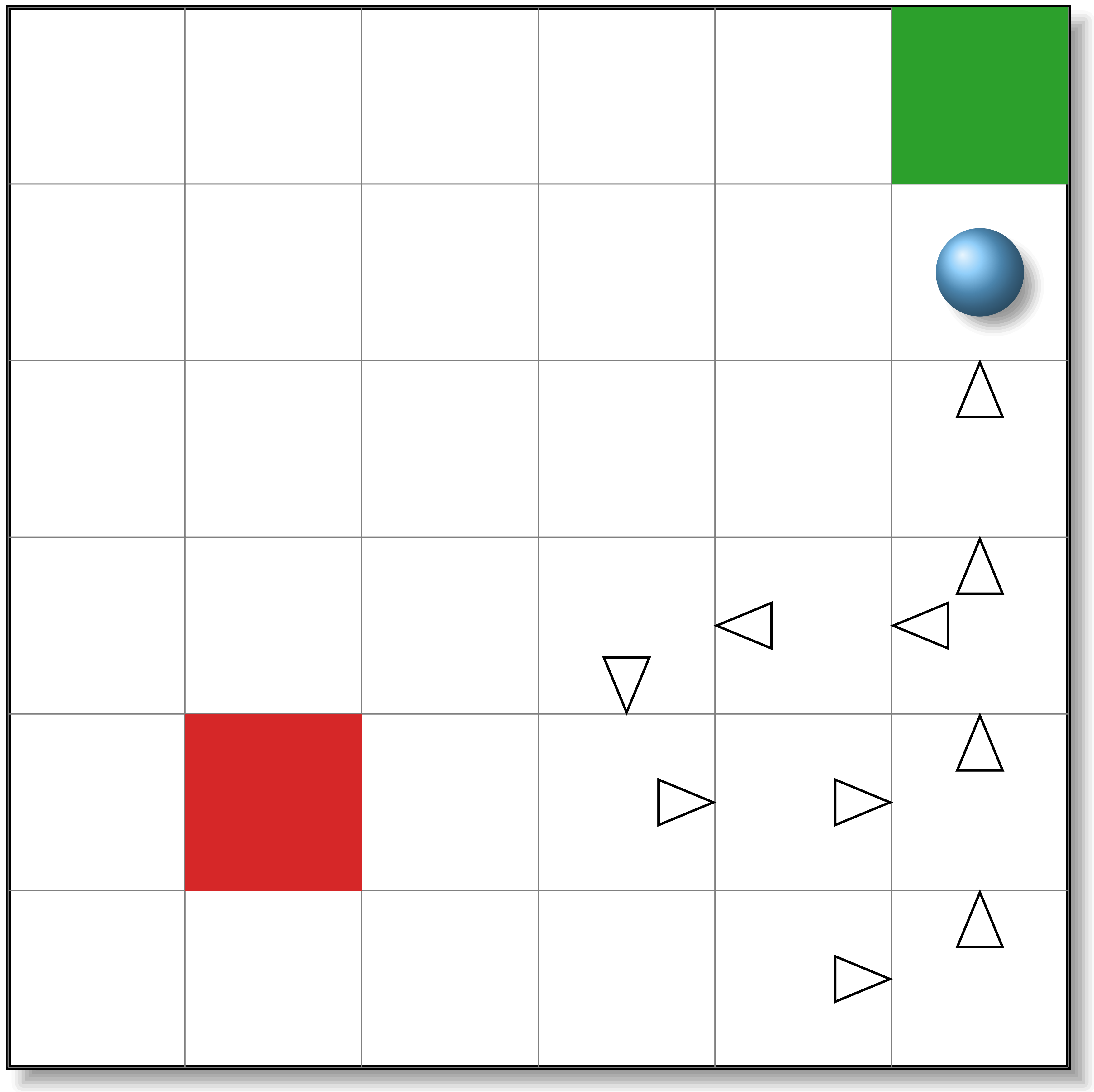


standard Q-Learning

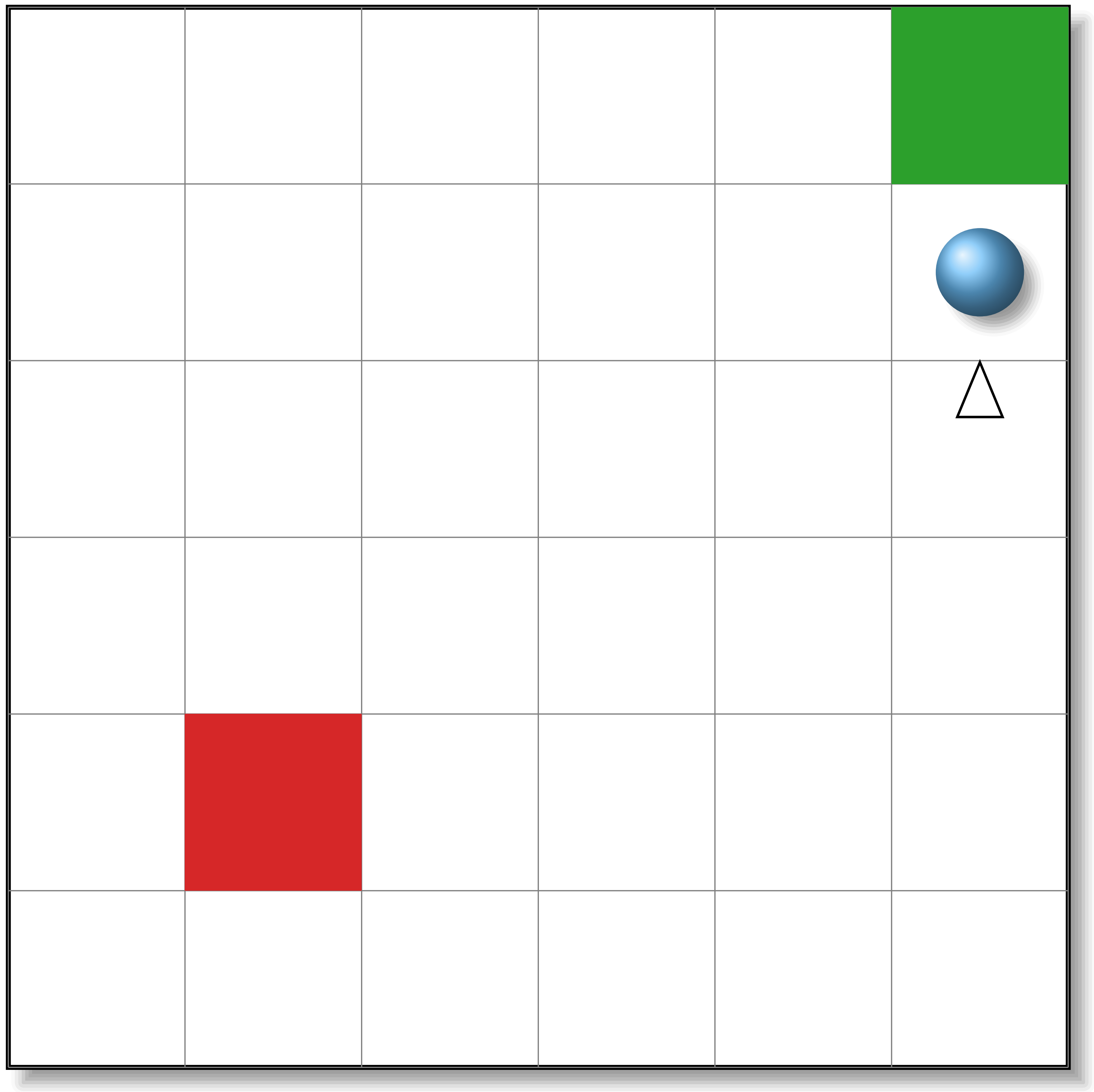



with replay memory
standard Q-Learning

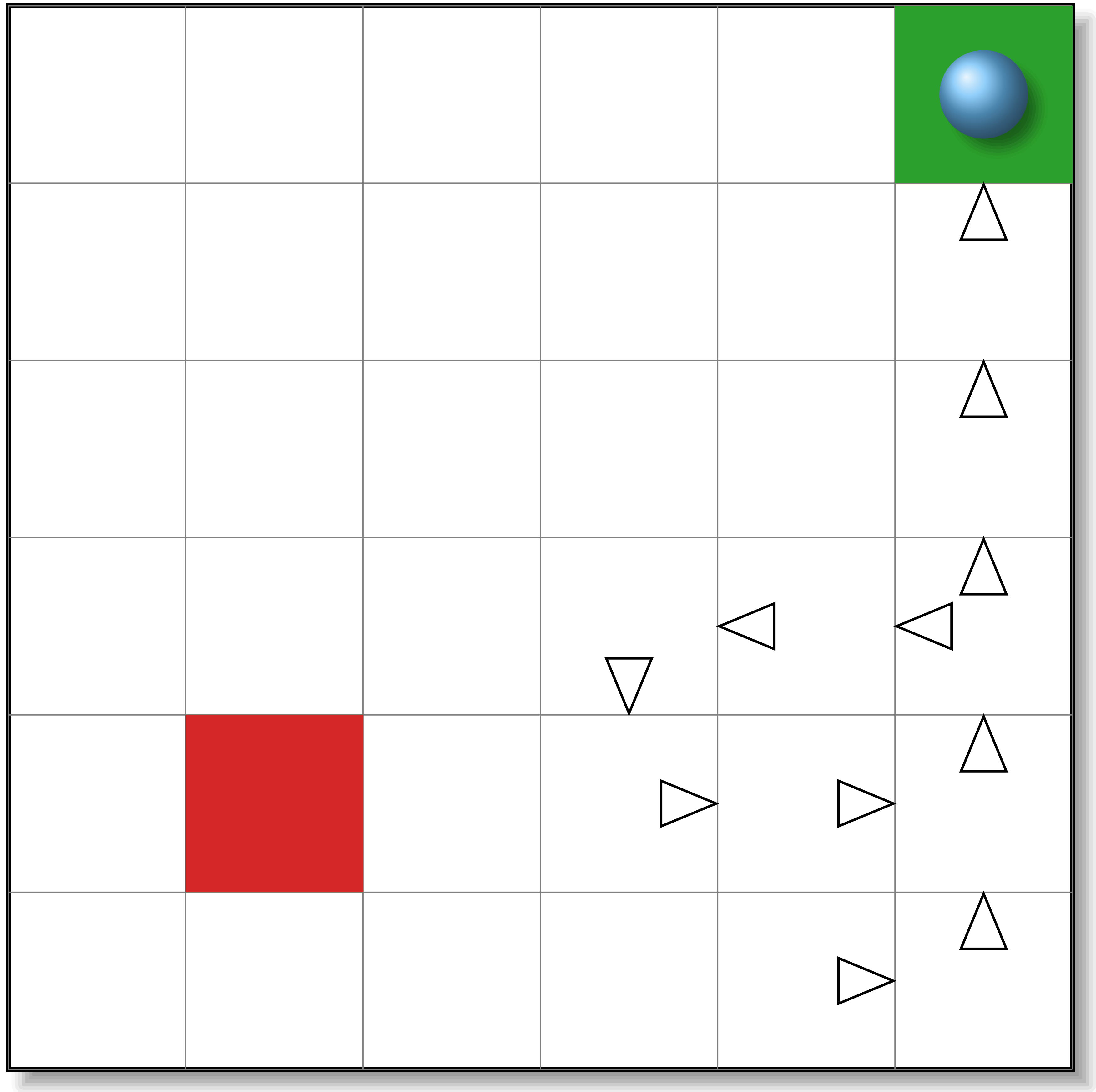
with replay memory



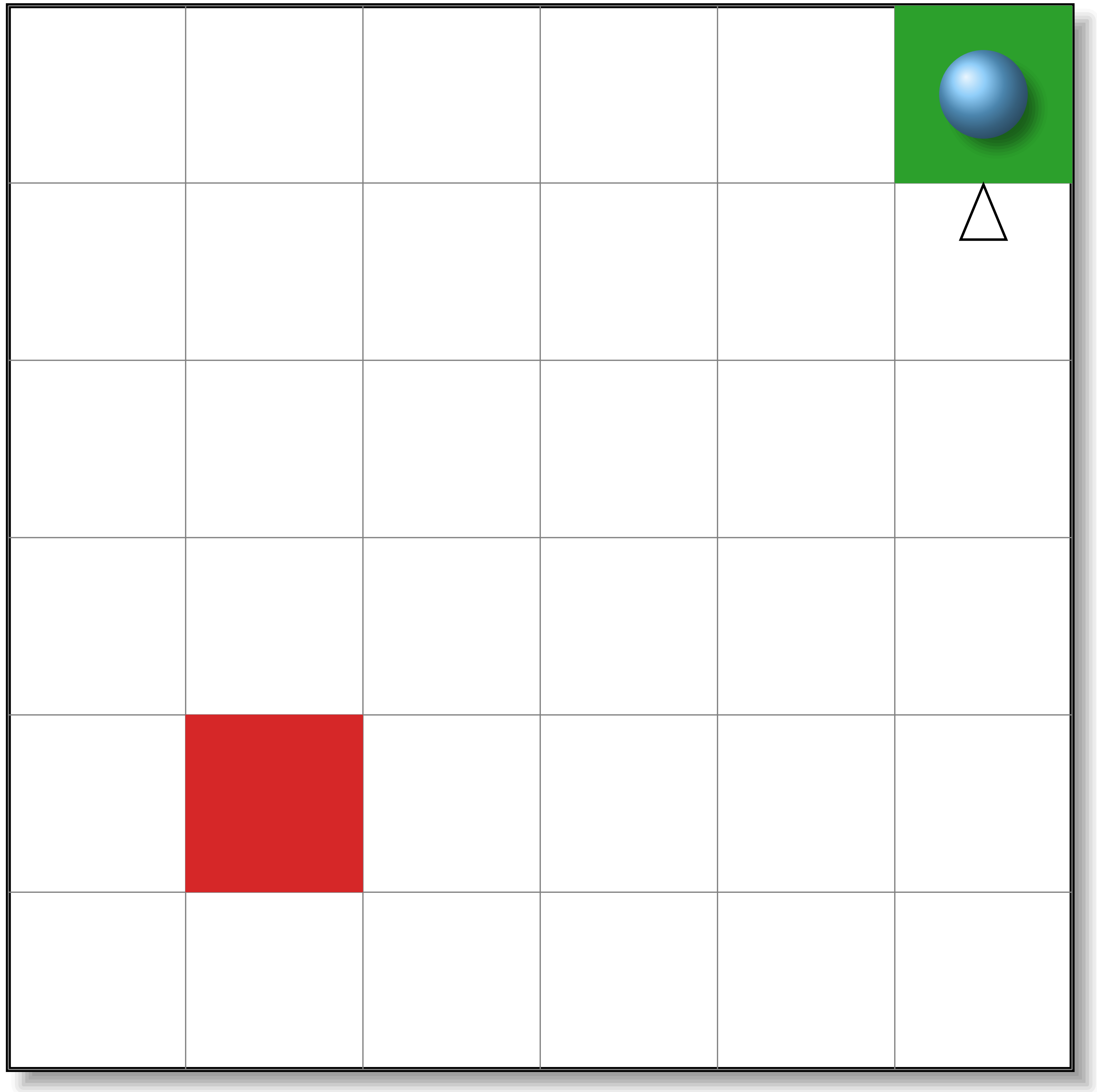
standard Q-Learning



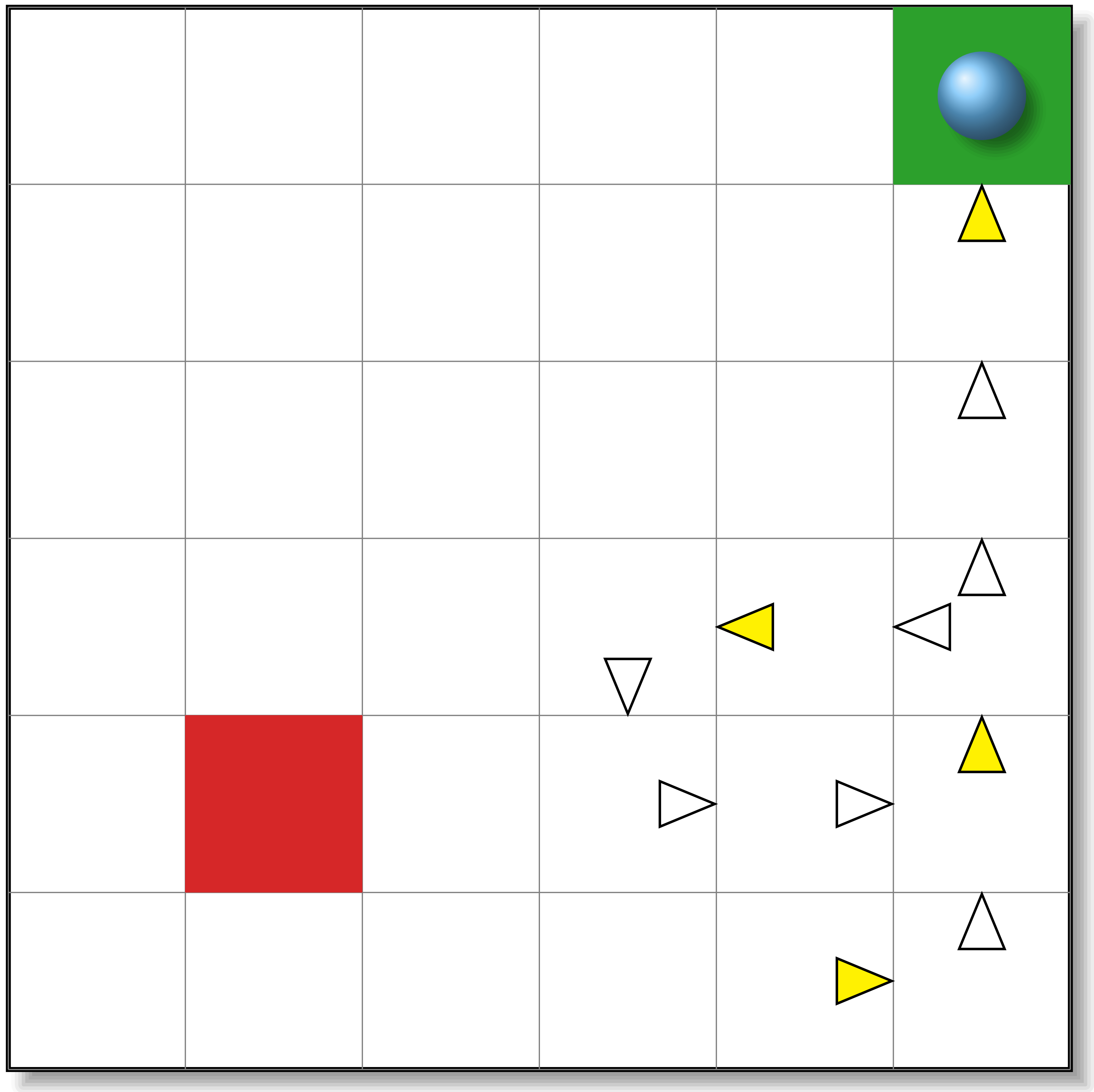
with replay memory



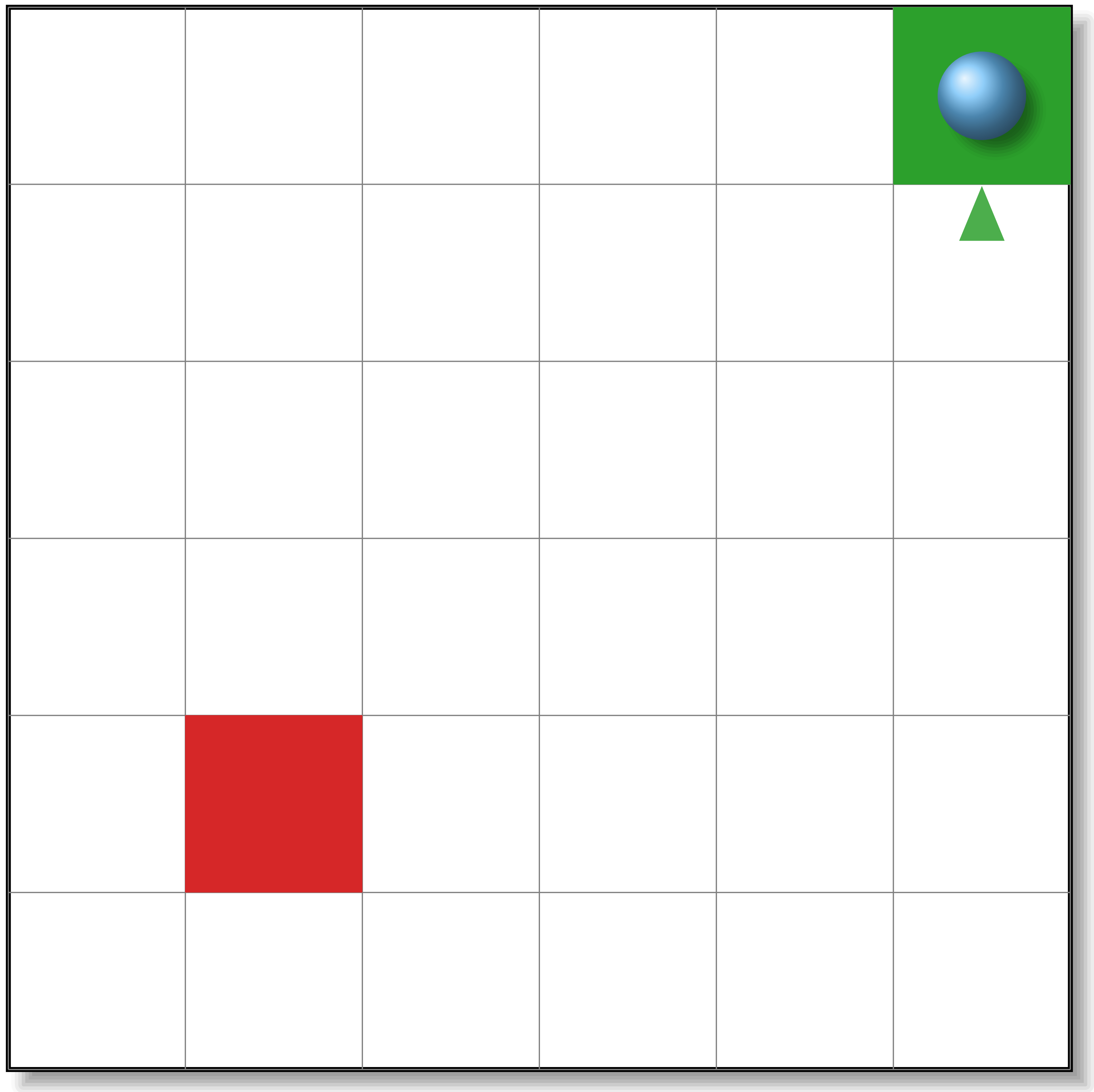
standard Q-Learning

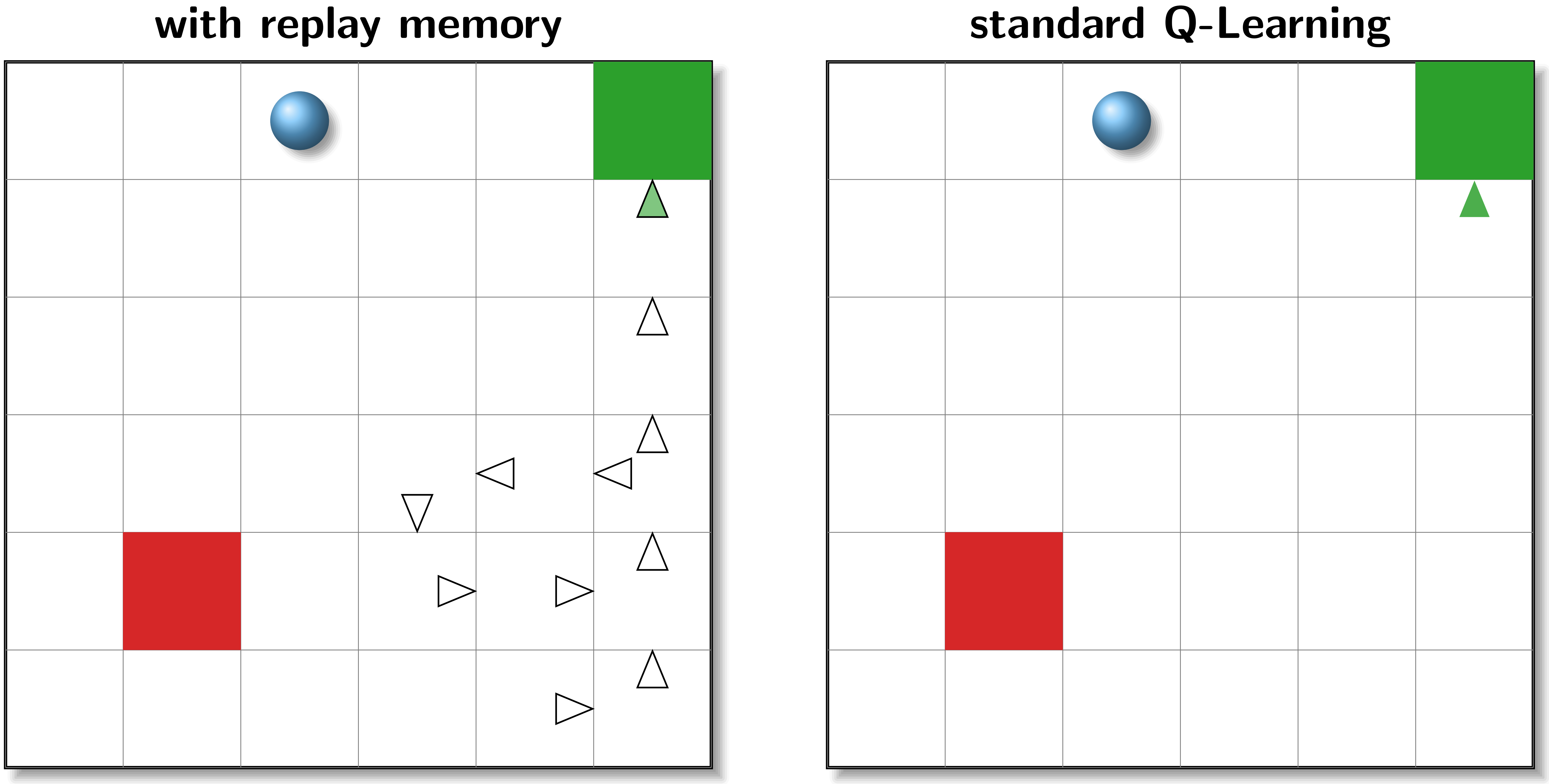


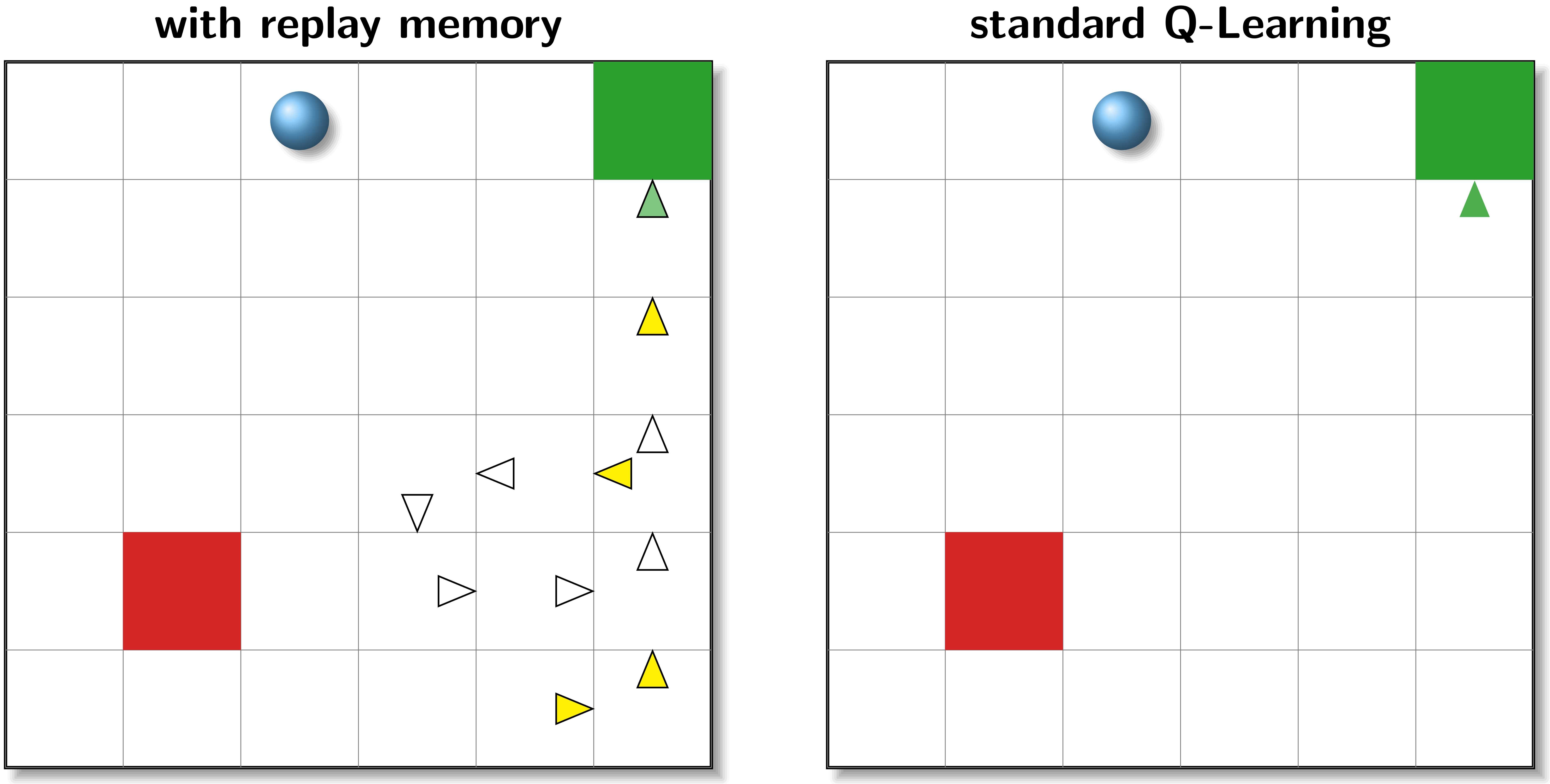
with replay memory



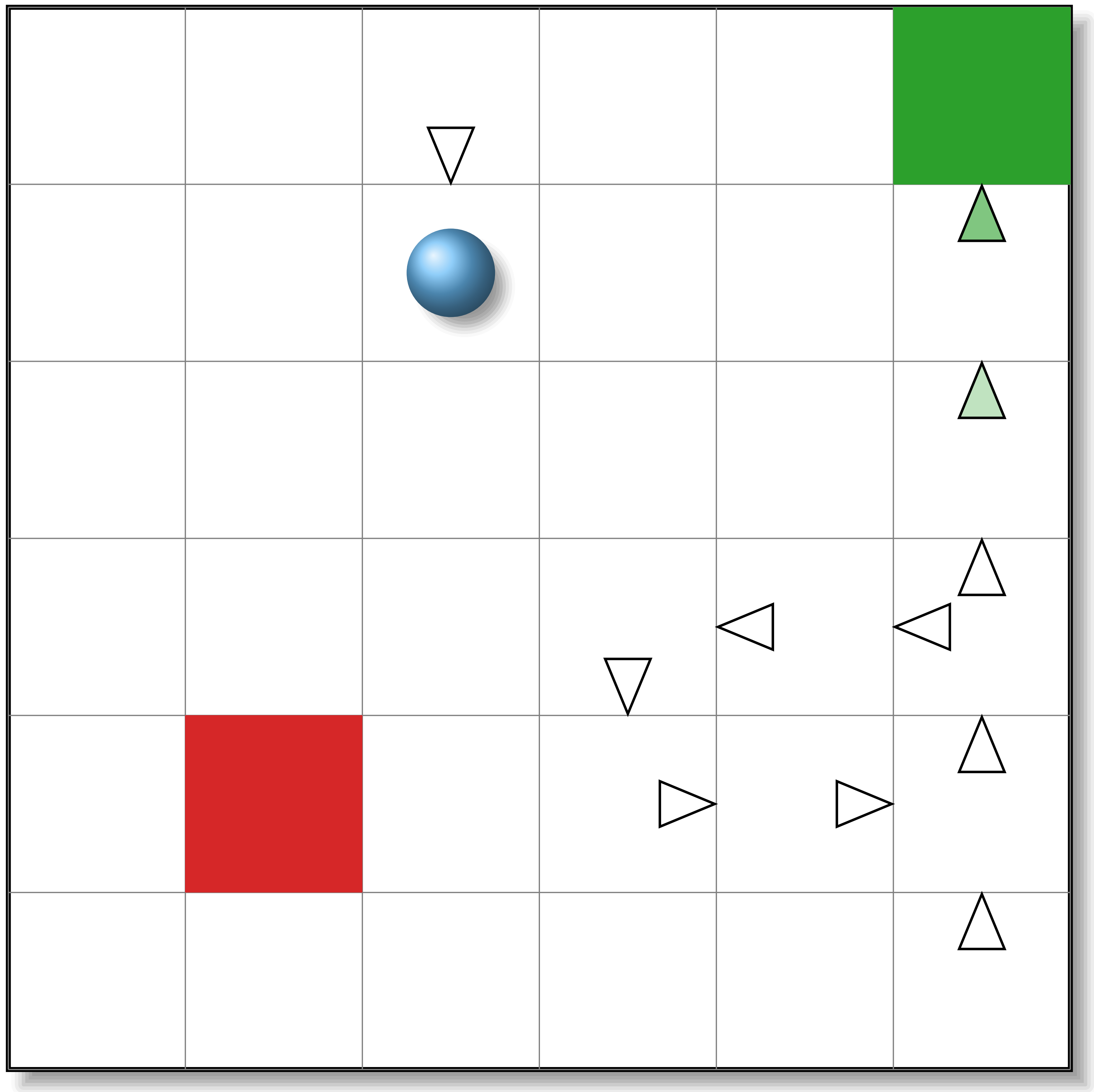
standard Q-Learning



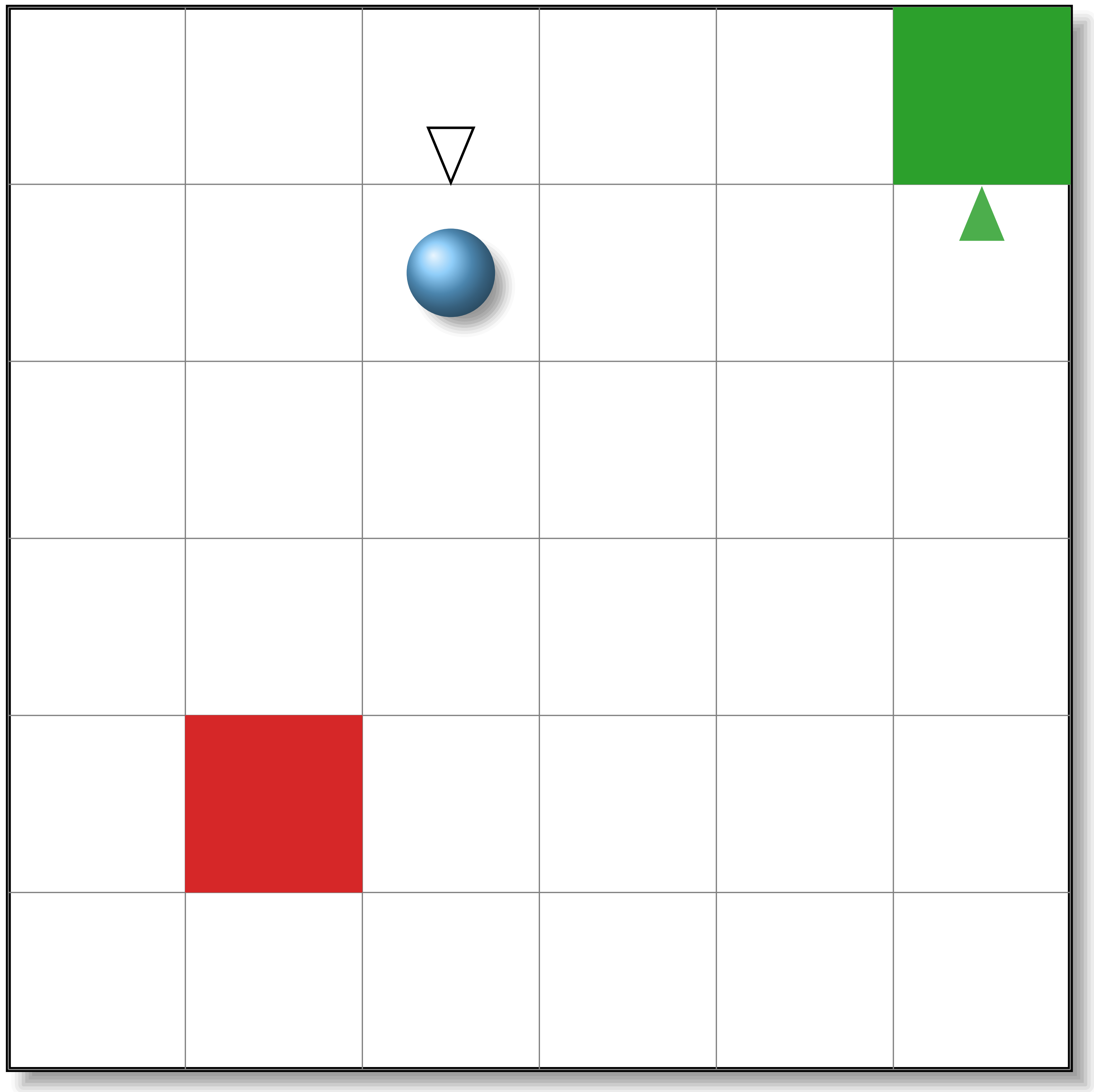

with replay memory
standard Q-Learning

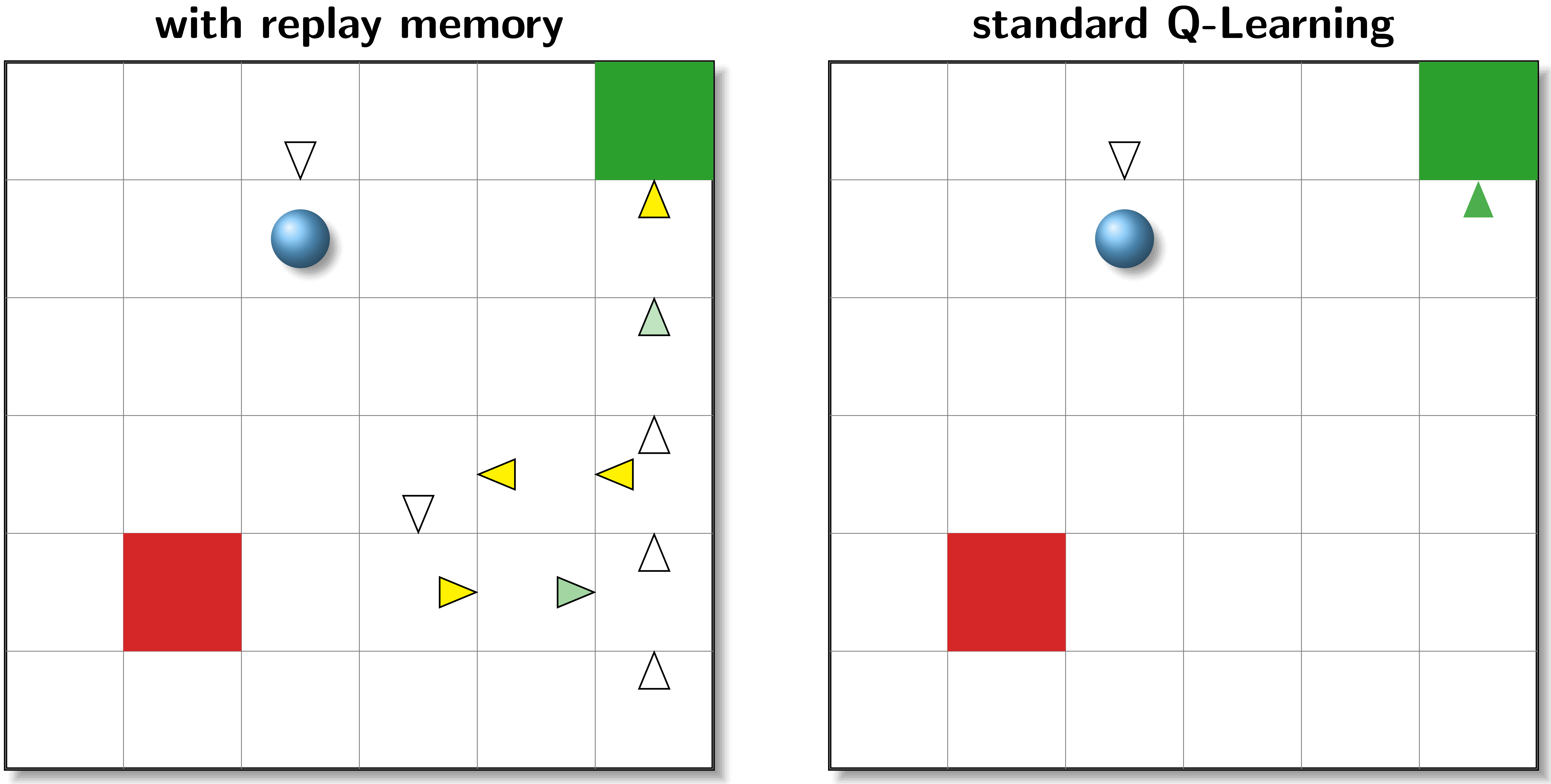

with replay memory
standard Q-Learning

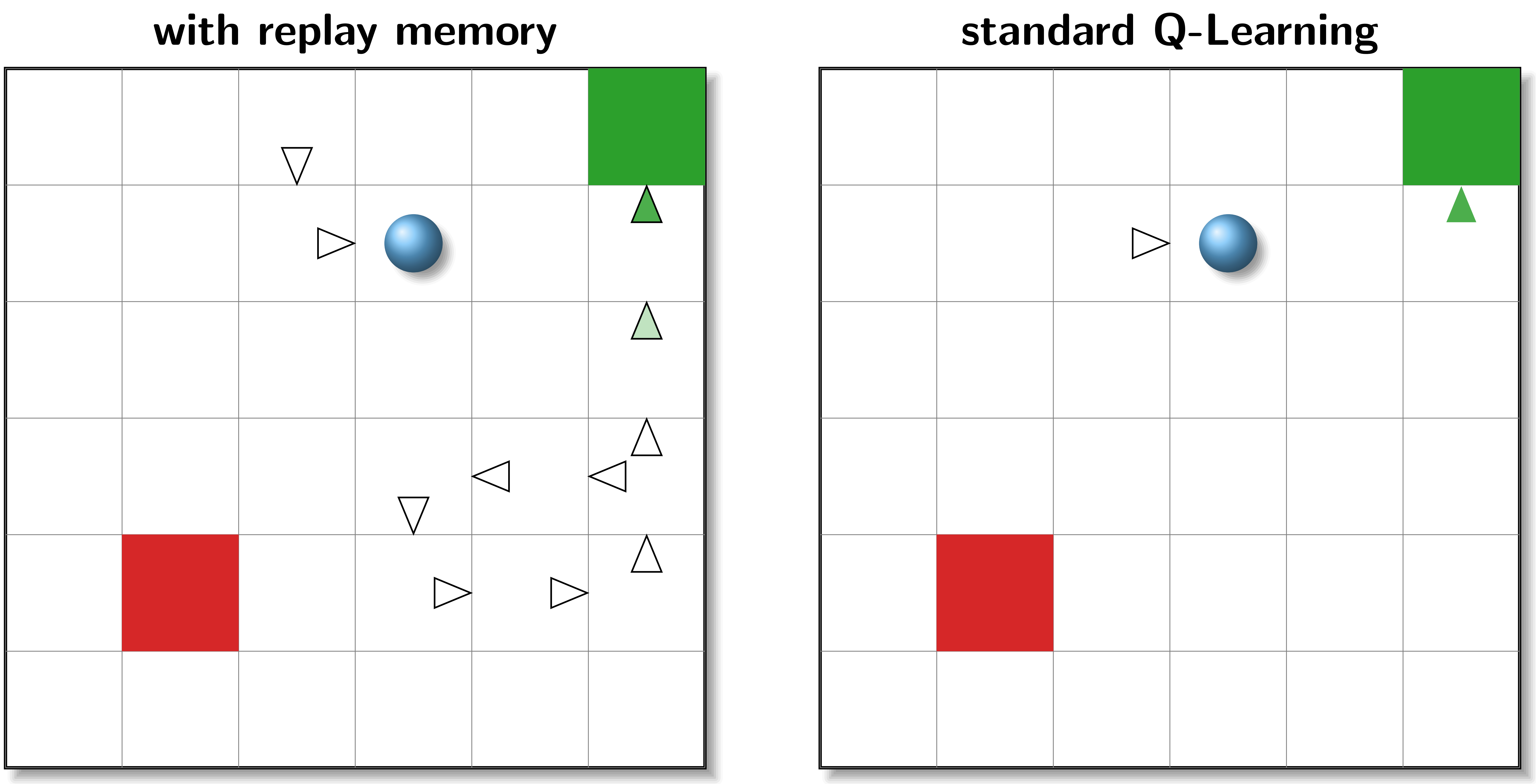
with replay memory

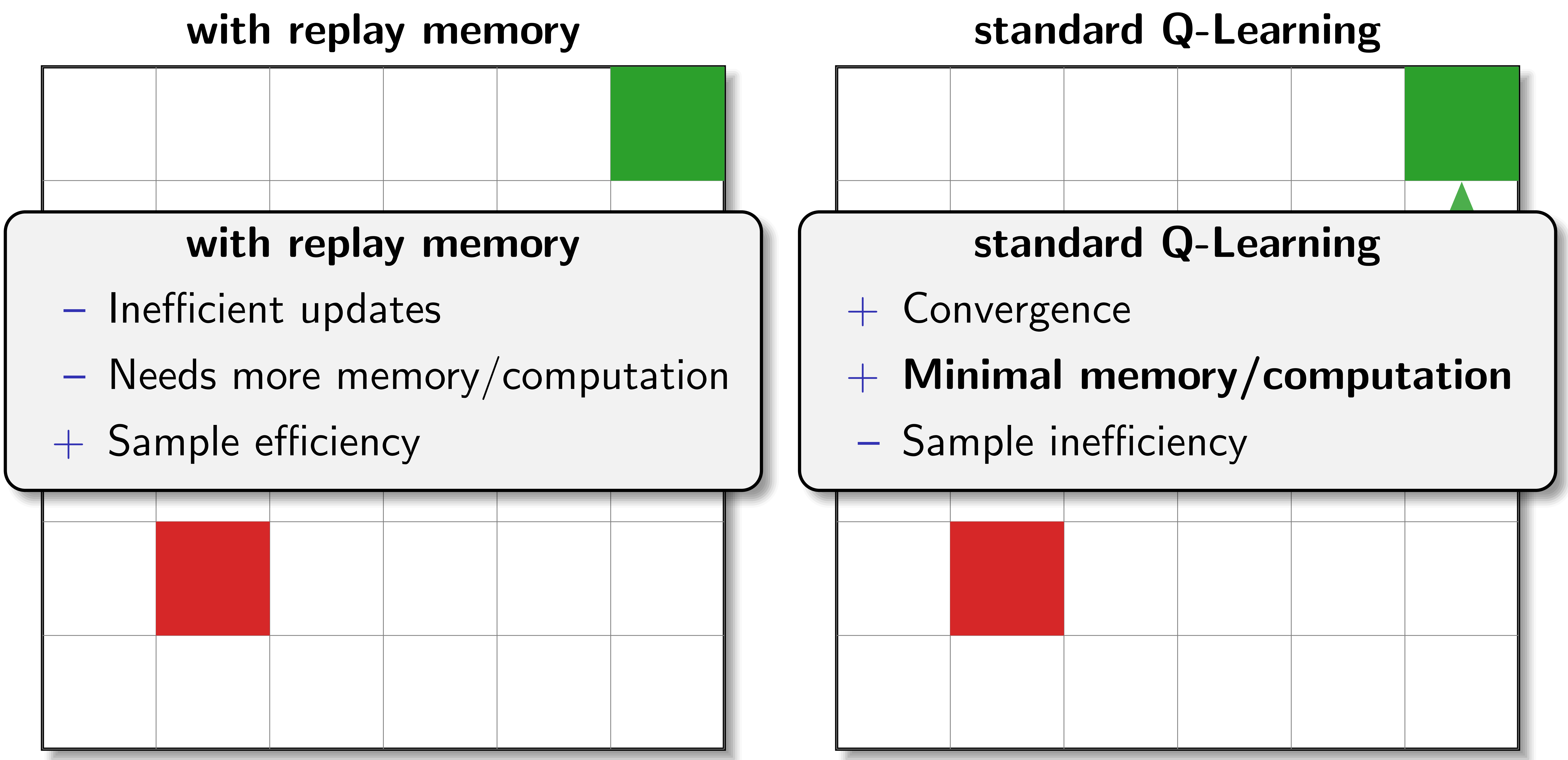


standard Q-Learning




with replay memory
standard Q-Learning


with replay memory
standard Q-Learning

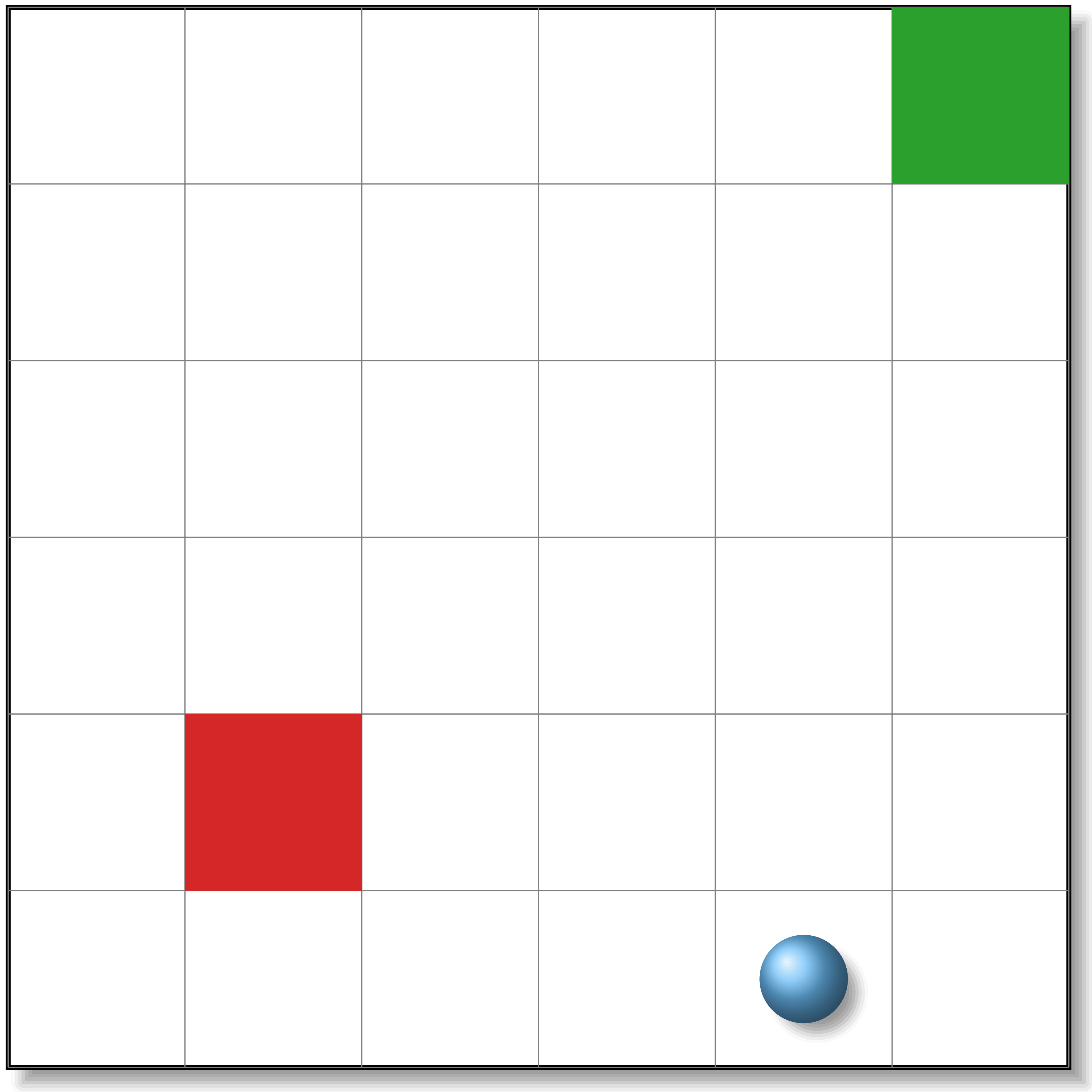

with replay memory
standard Q-Learning
with replay memory
– Inefficient updates
– Needs more memory/computation
+ Sample efficiency
standard Q-Learning
+ Convergence
+ **Minimal memory/computation**
– Sample inefficiency

Can we do better?

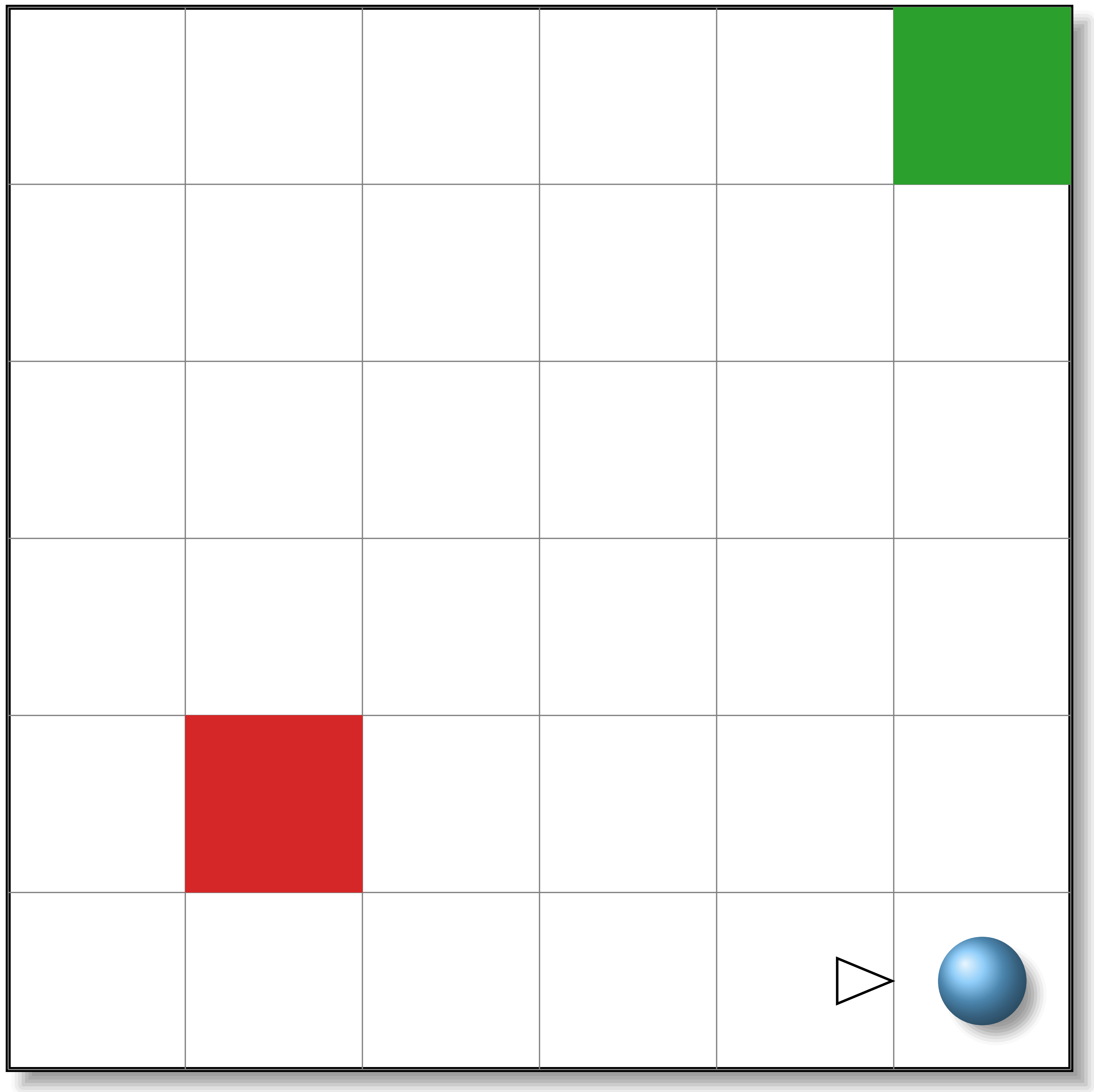
Yes.

1. Organize memory in table, i.e. estimate $P^a_{s \to s'} = N^a_{s \to s'} / N^a_s$ with counts $N^a_{s \to s'}$ and N^a_s
2. Prioritize backups cleverly.

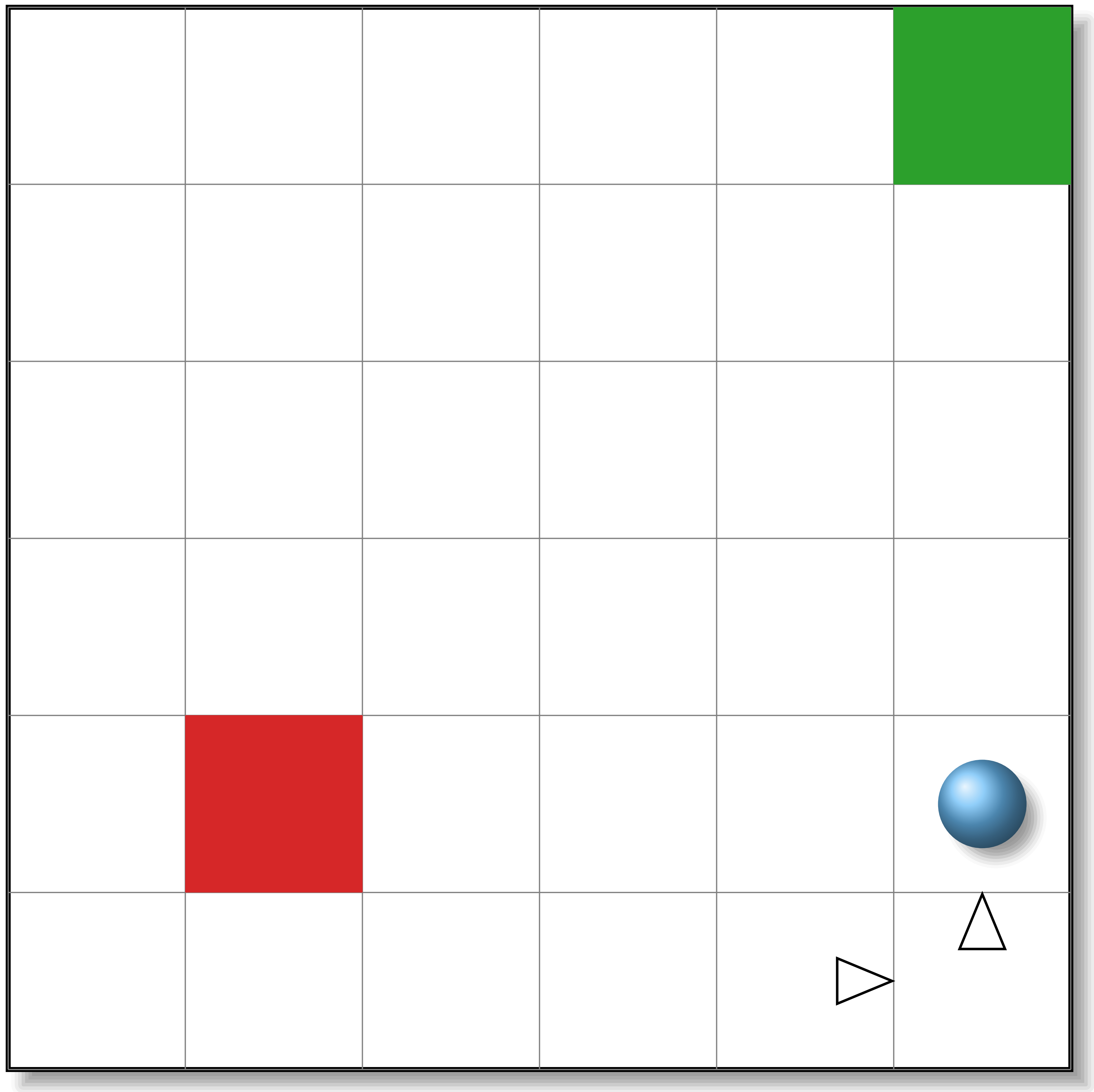
prioritized sweeping



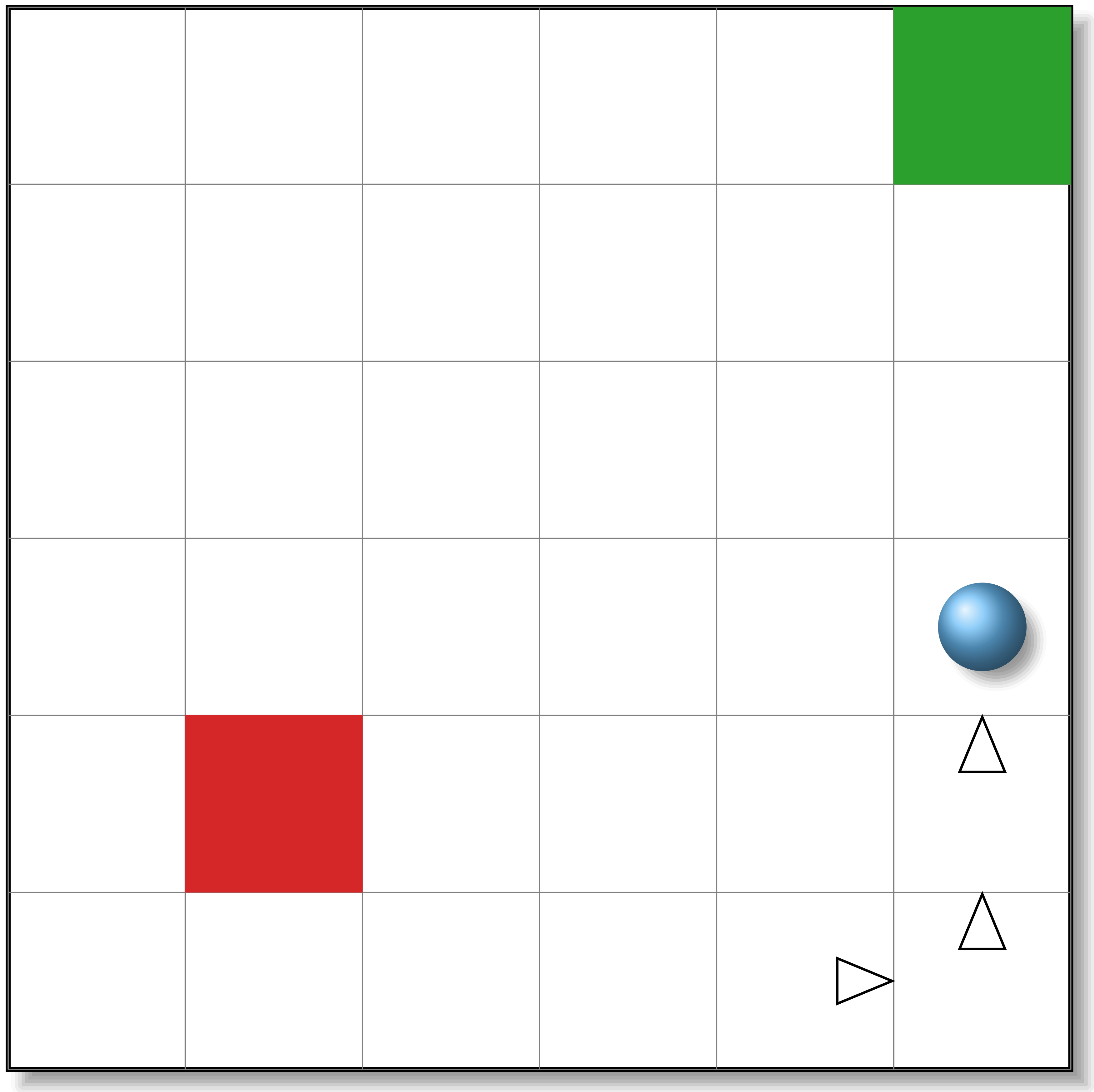
prioritized sweeping



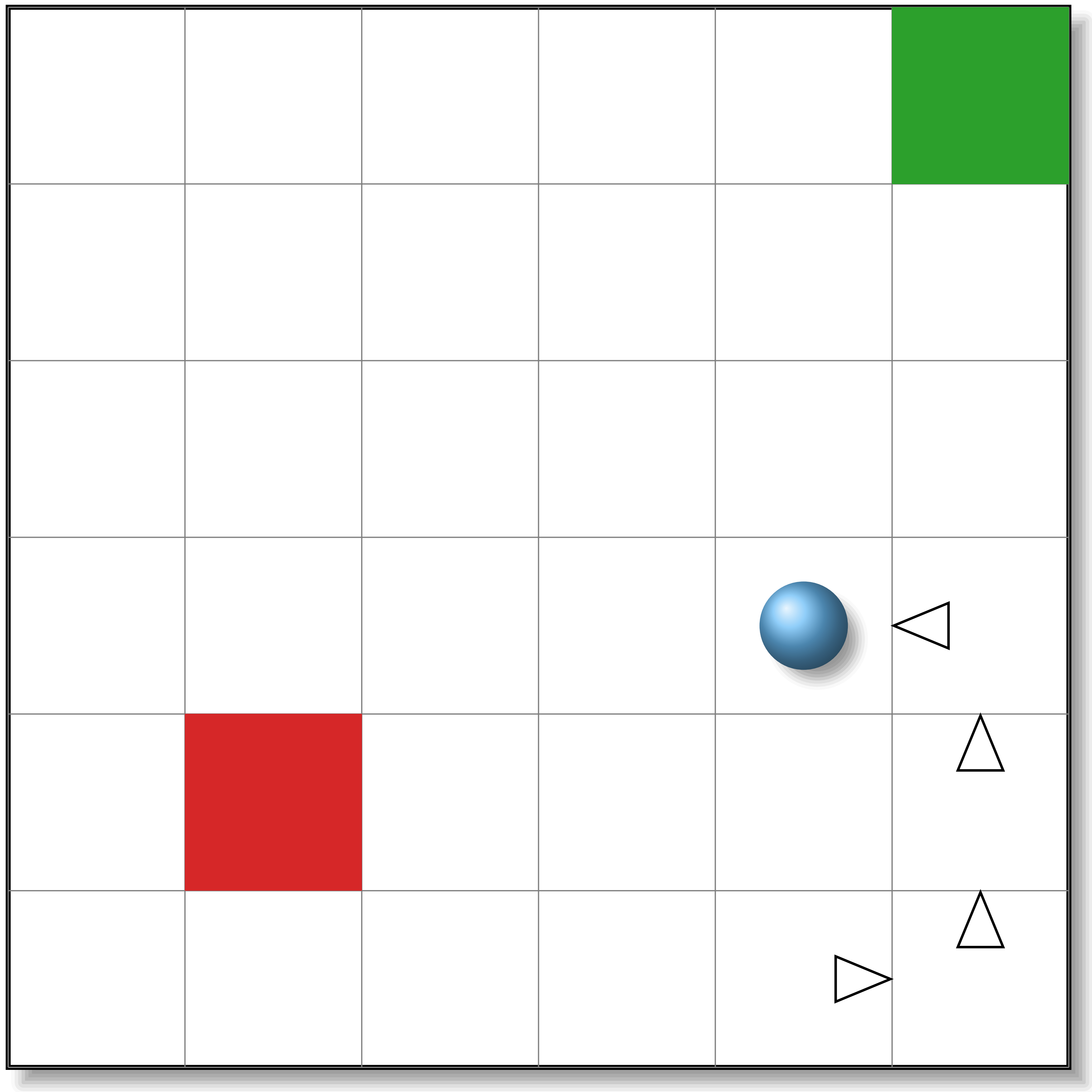
prioritized sweeping



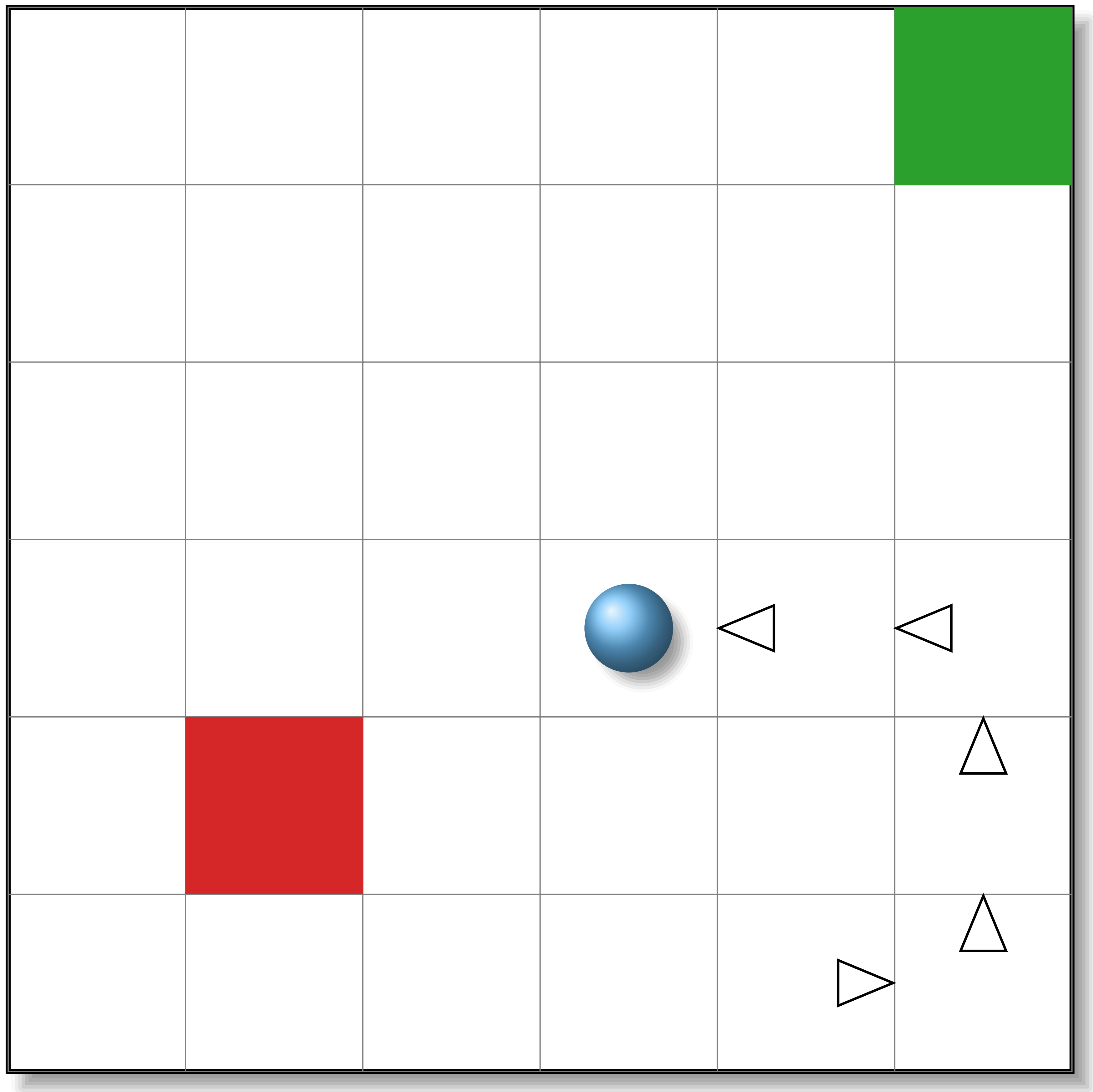
prioritized sweeping



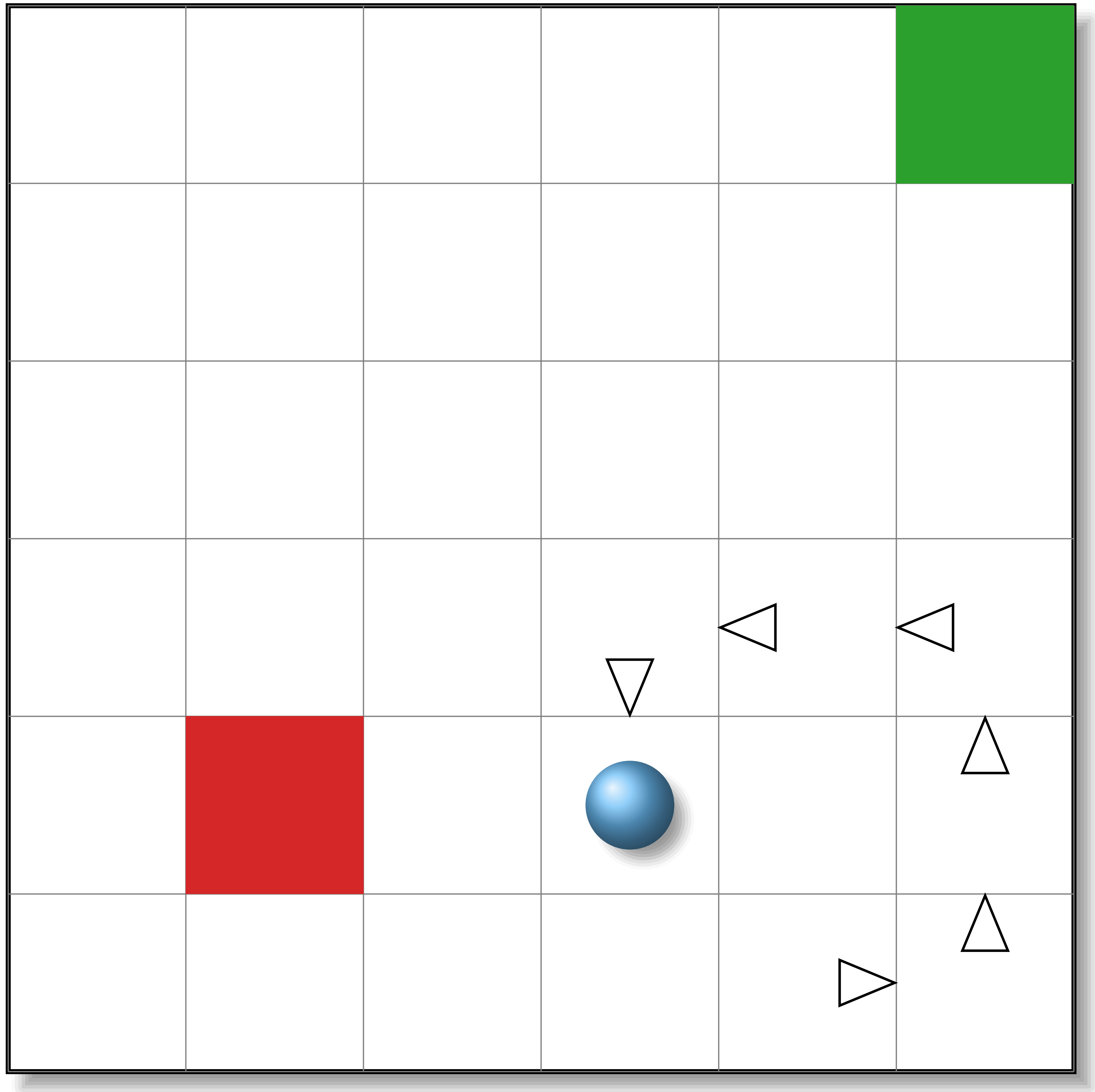
prioritized sweeping



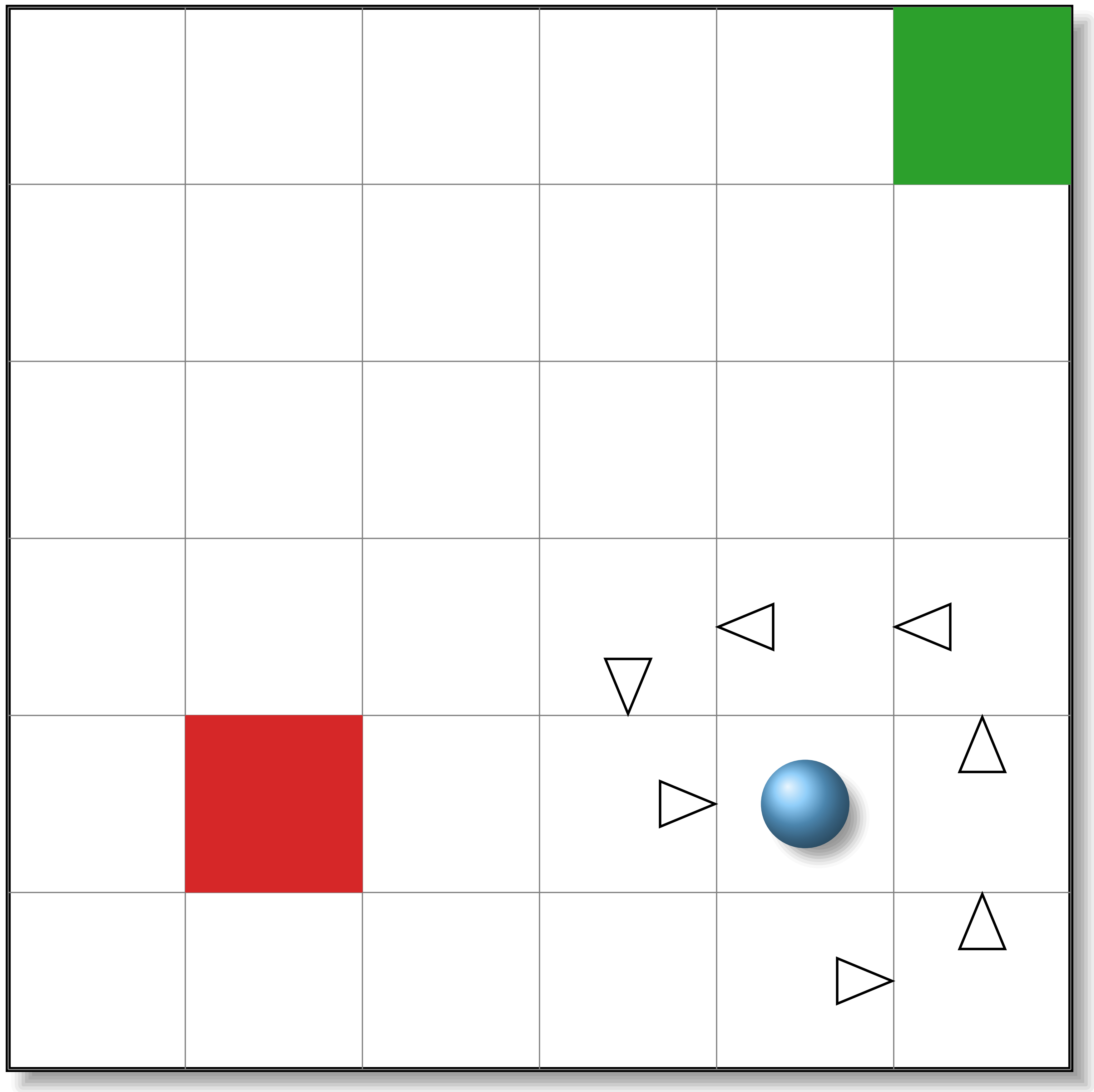
prioritized sweeping



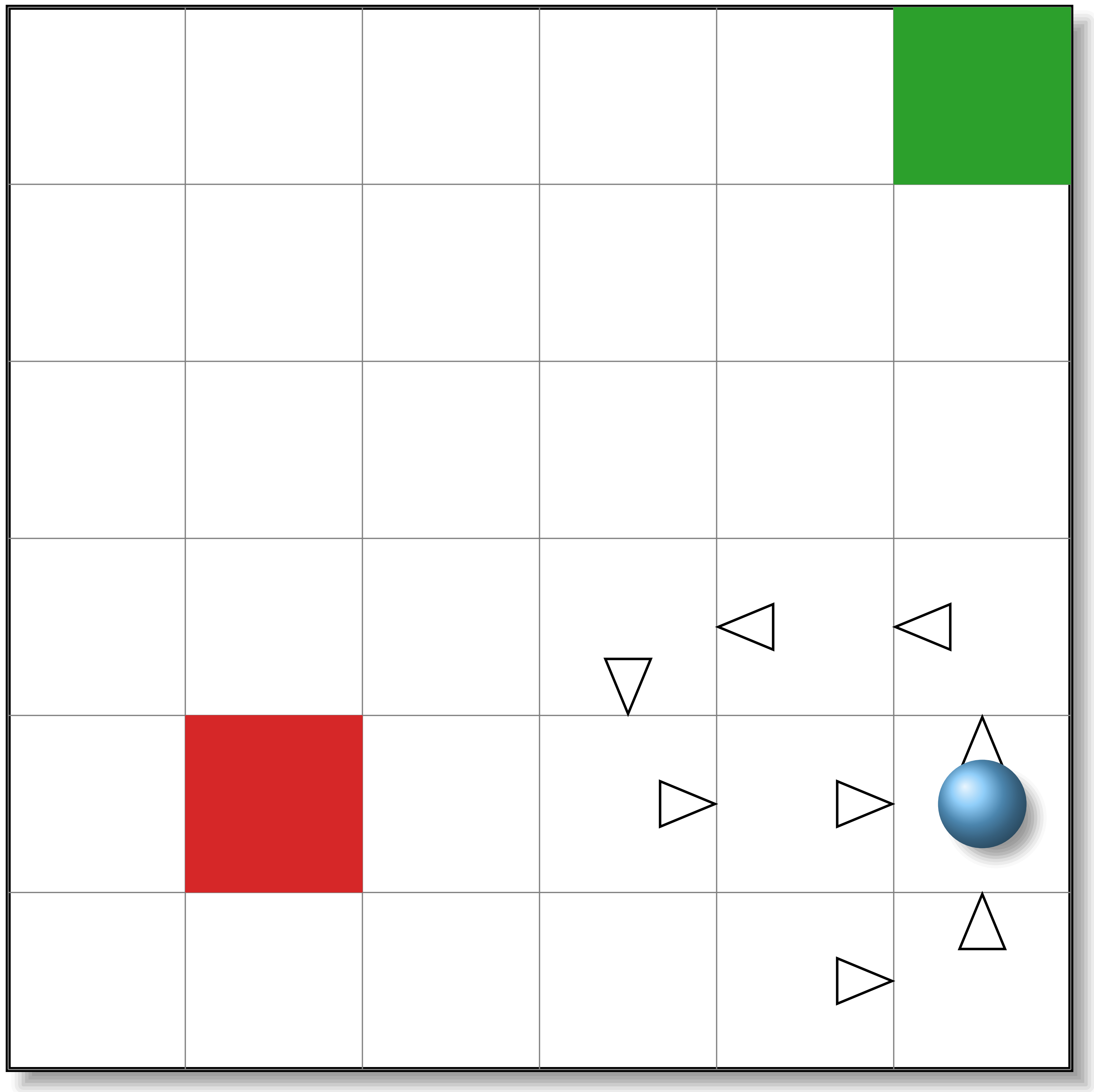
prioritized sweeping



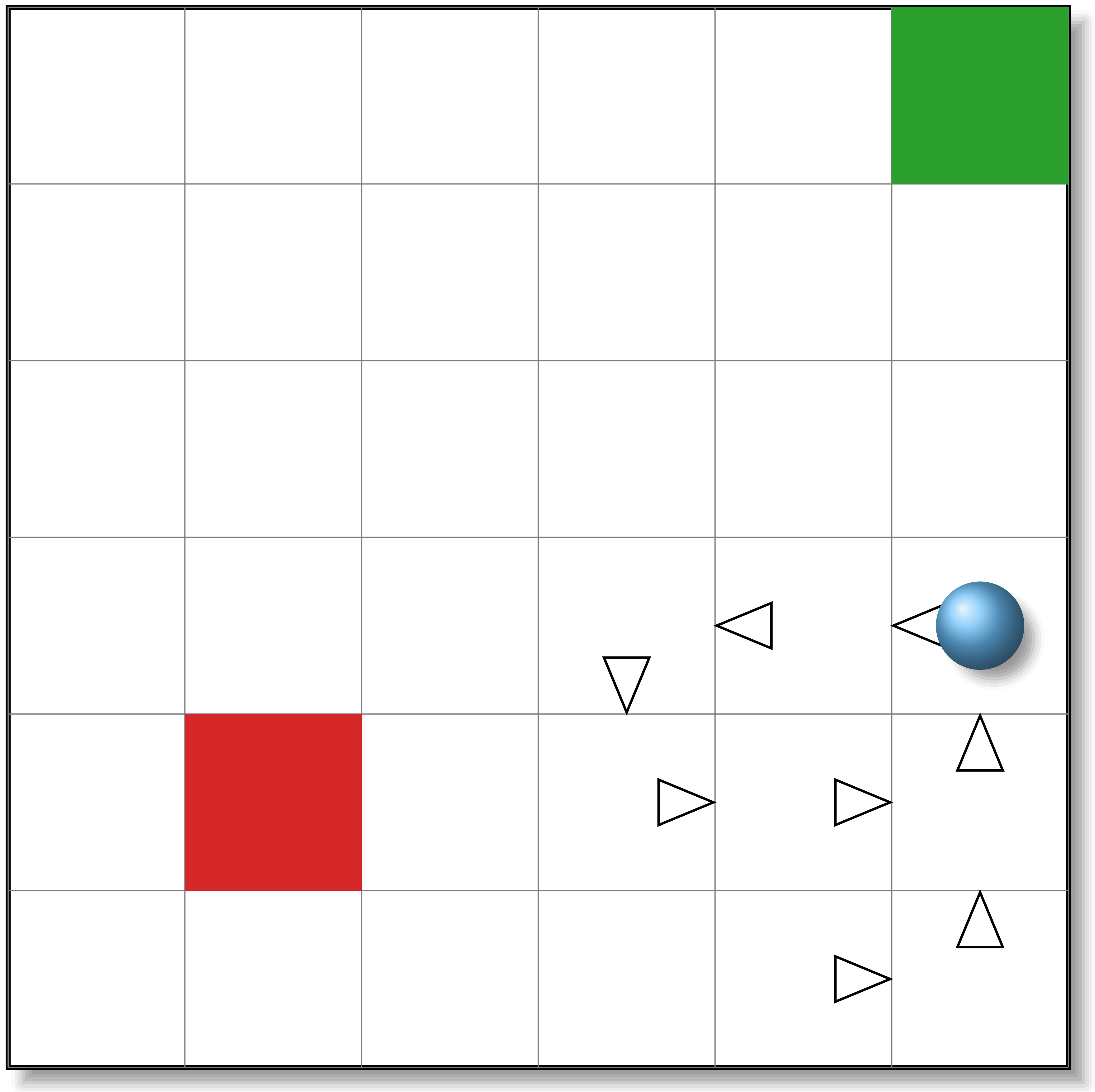
prioritized sweeping



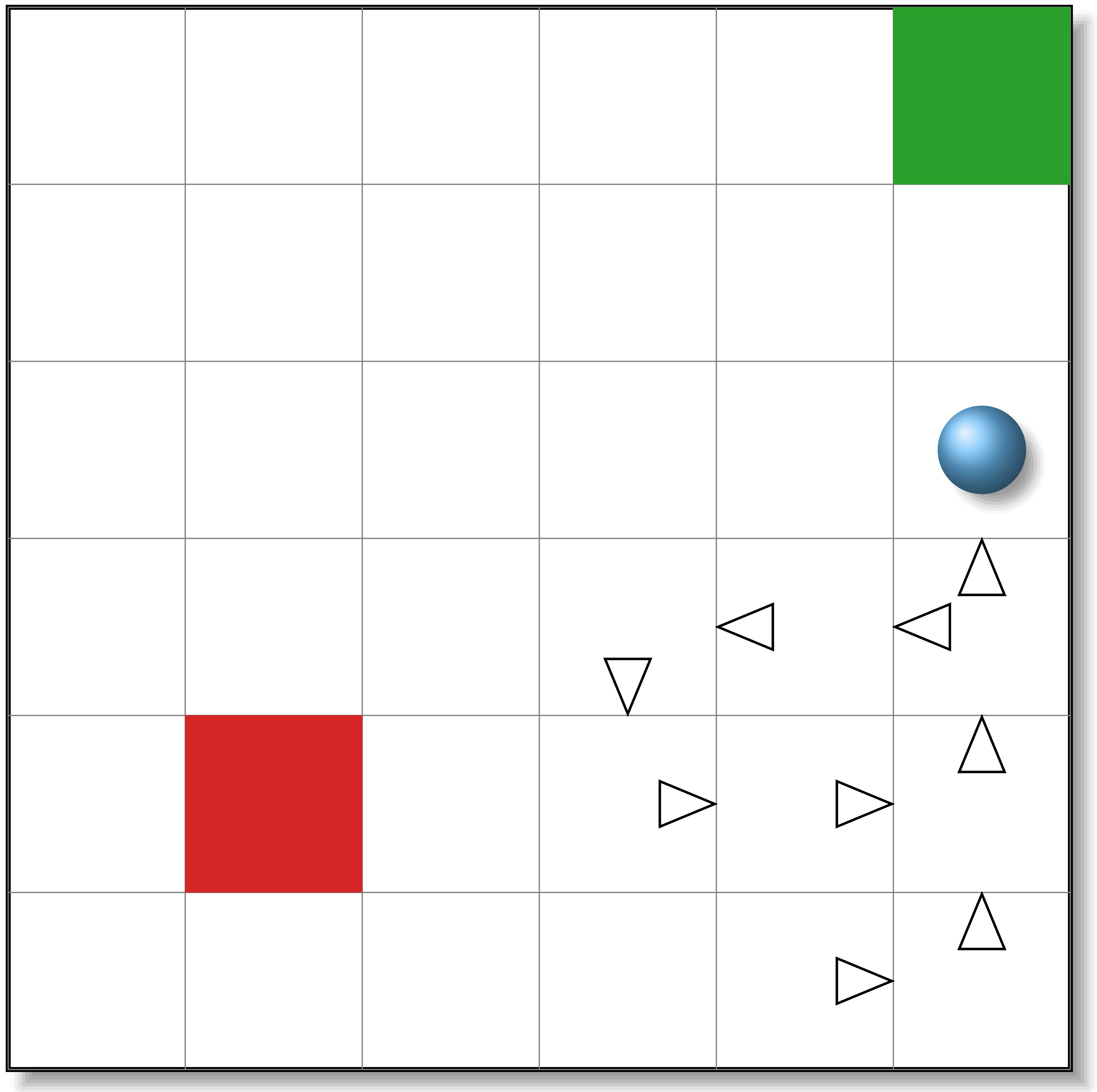
prioritized sweeping



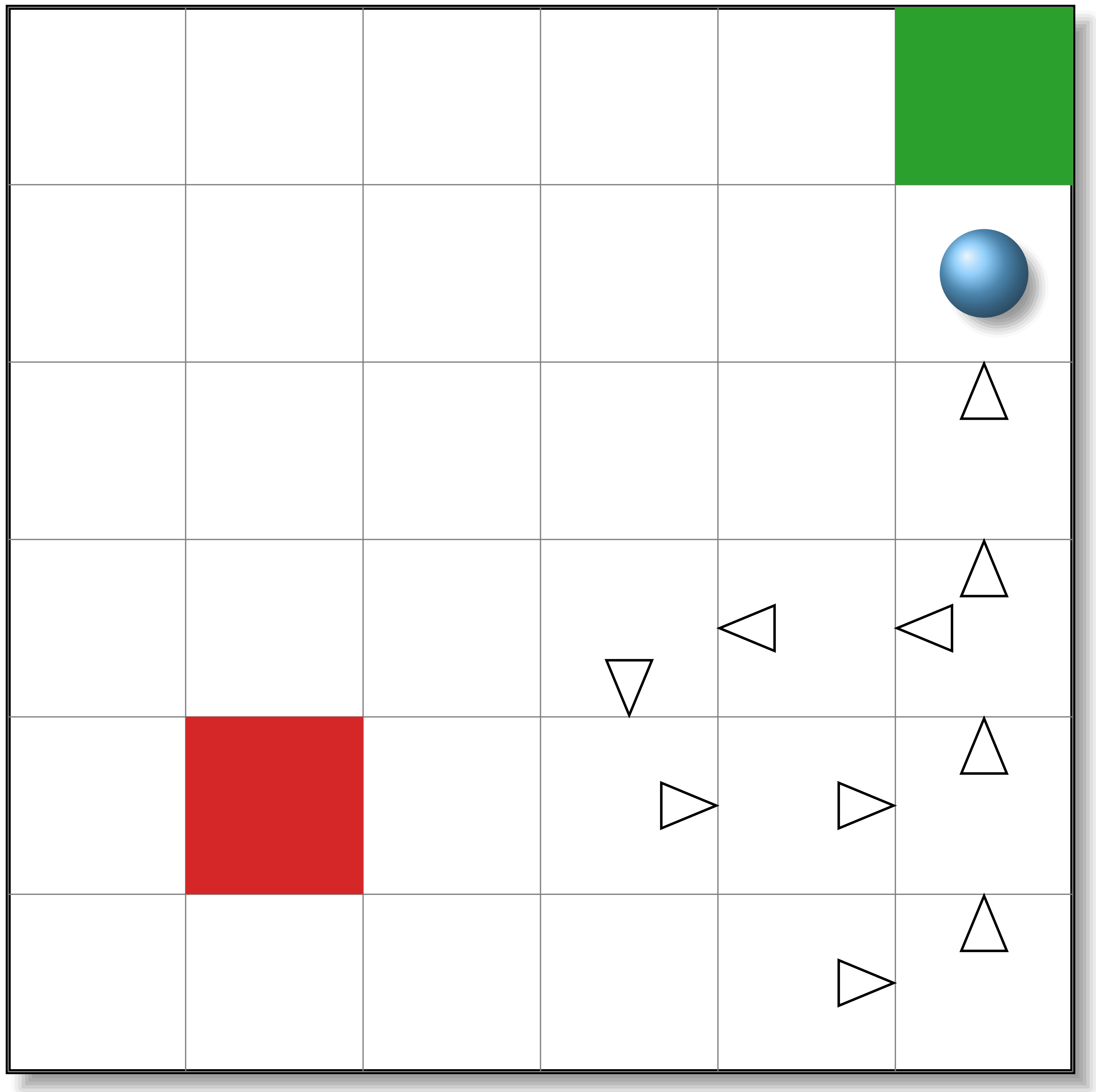
prioritized sweeping



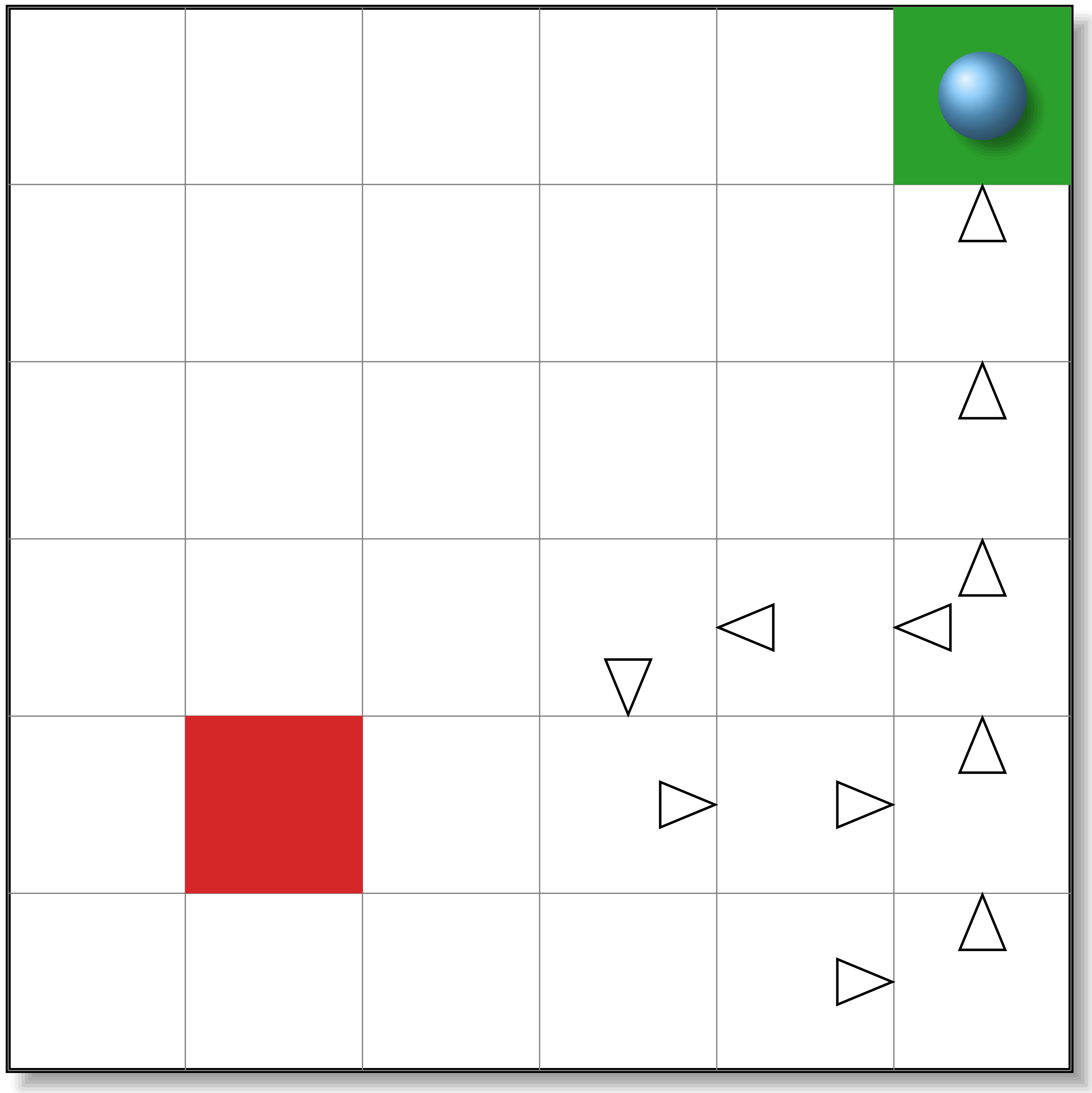
prioritized sweeping



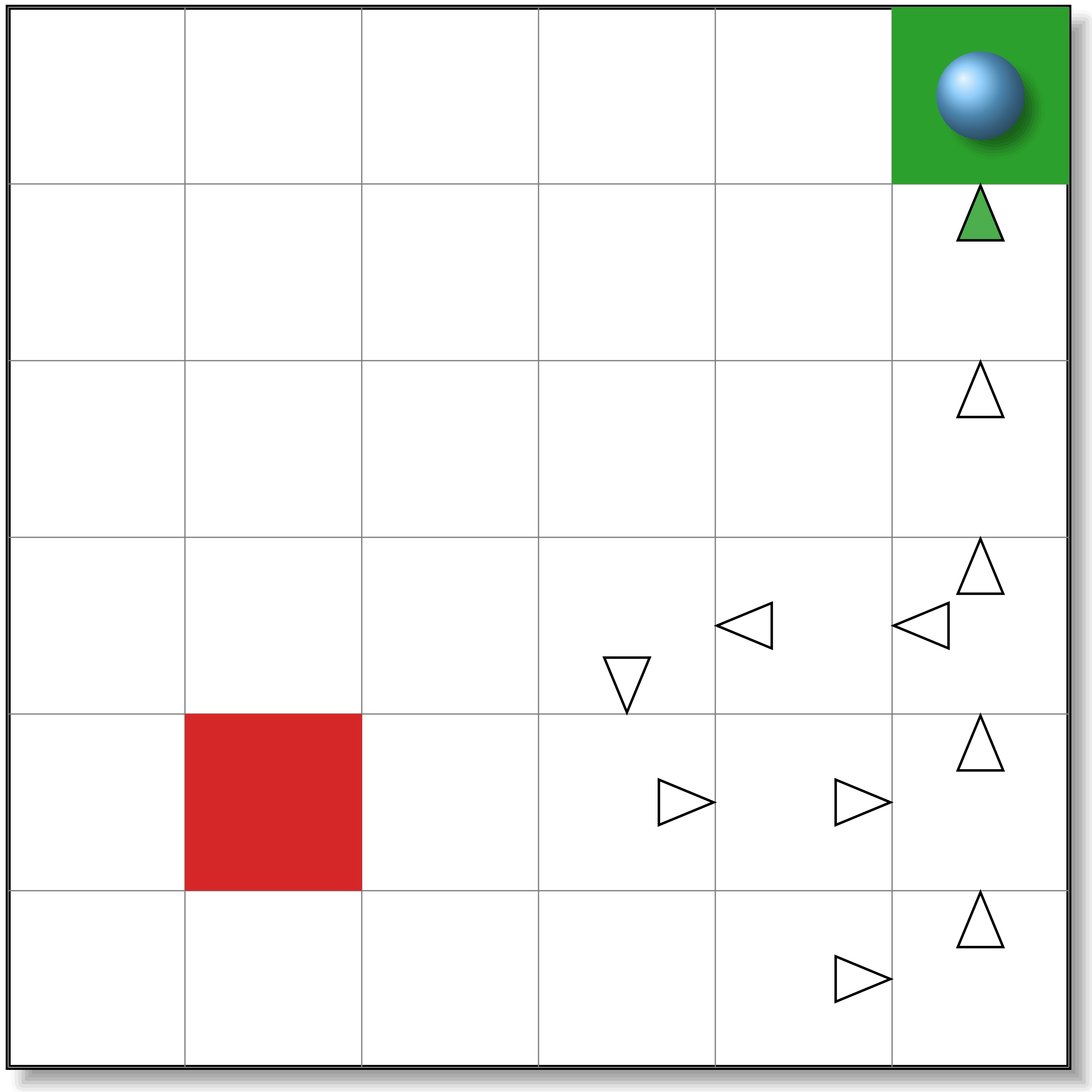
prioritized sweeping



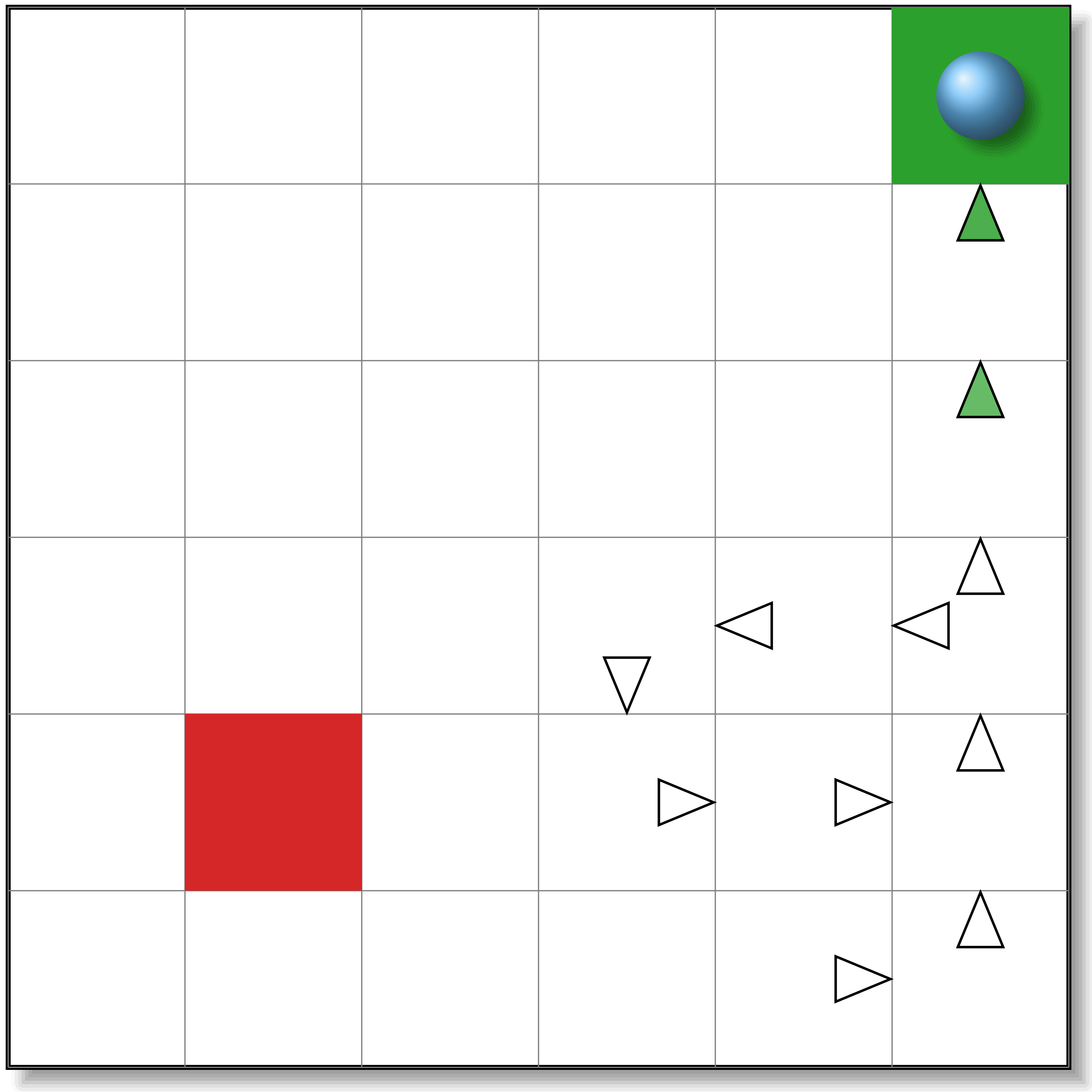
prioritized sweeping



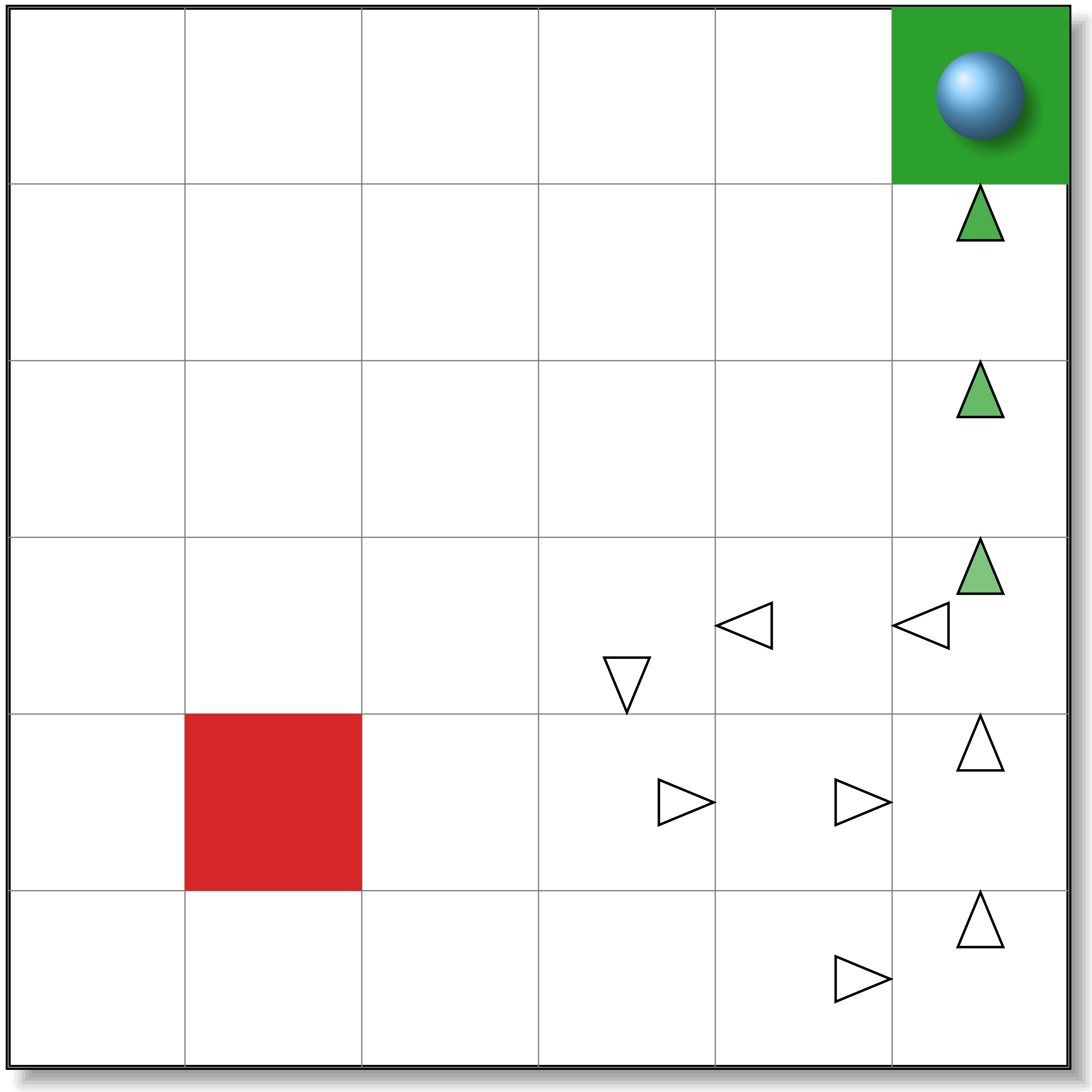
prioritized sweeping



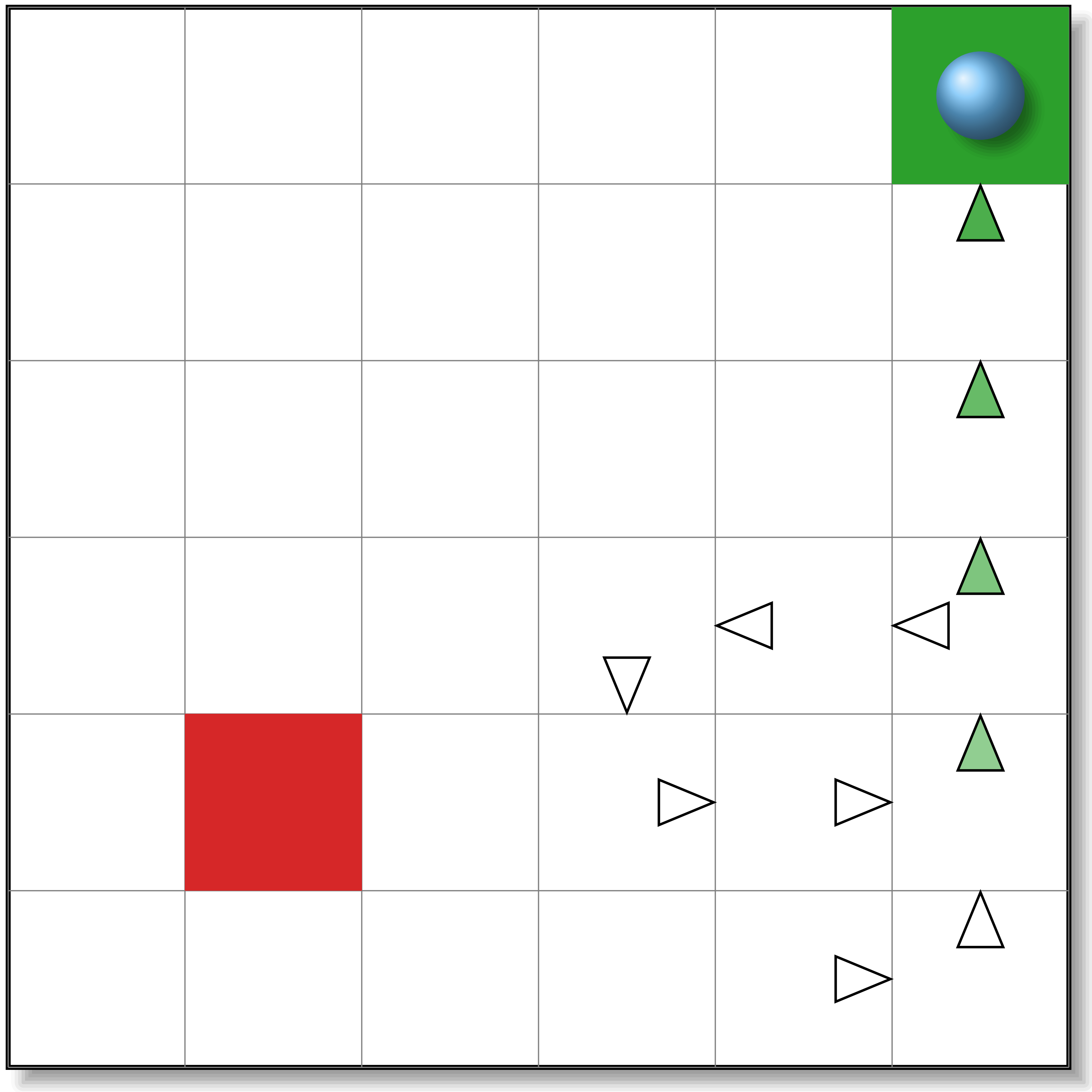
prioritized sweeping



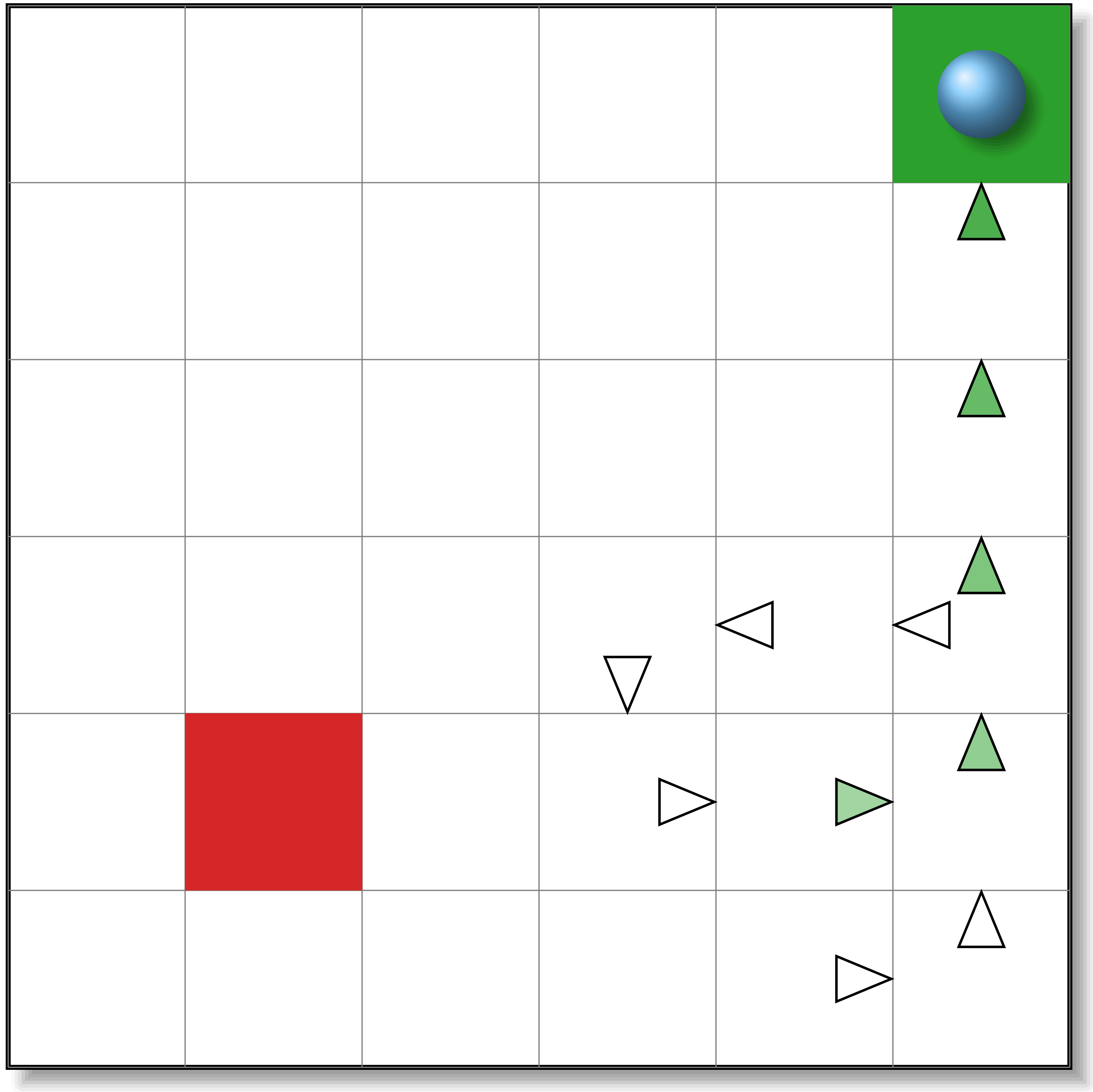
prioritized sweeping



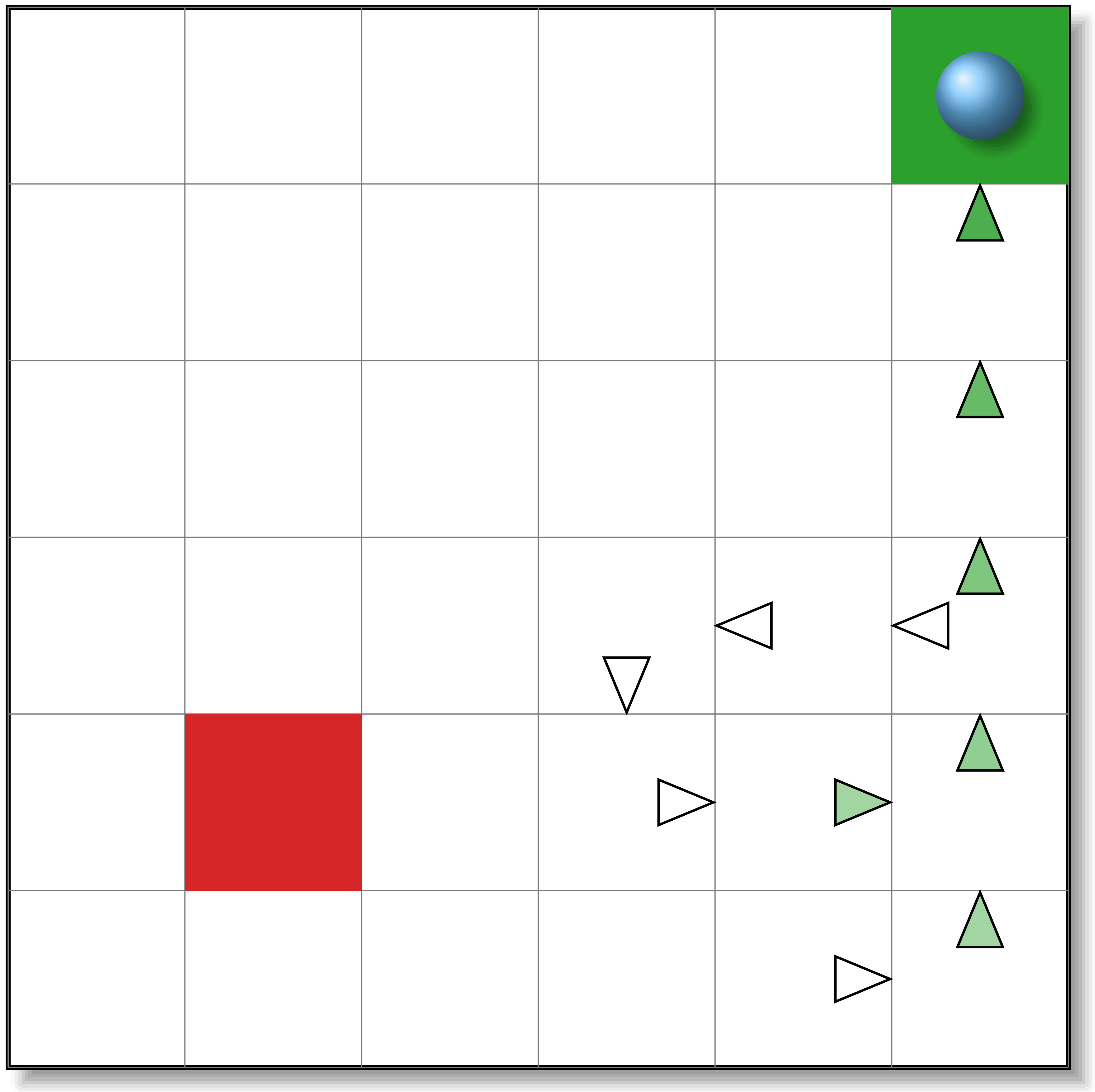
prioritized sweeping



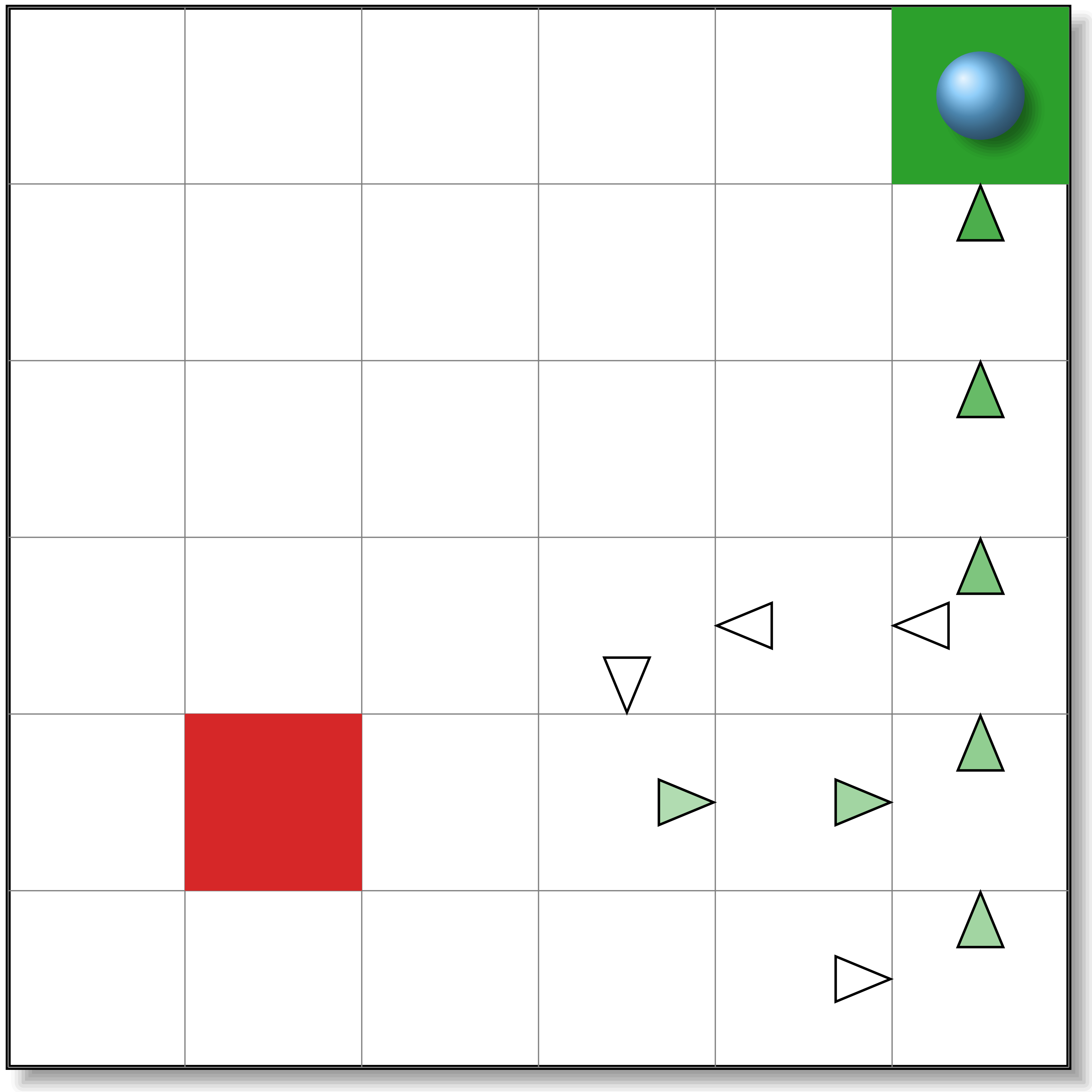
prioritized sweeping



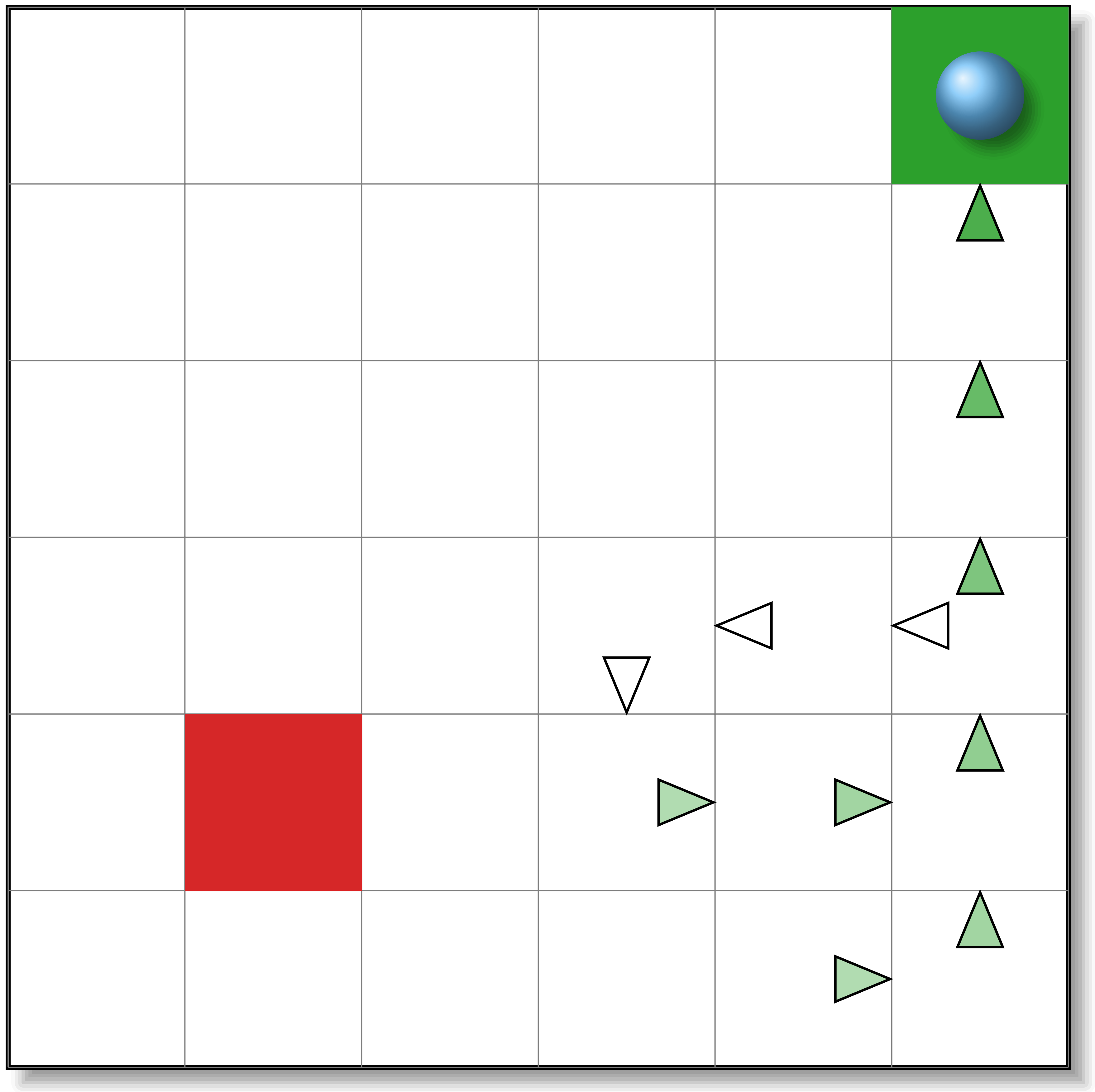
prioritized sweeping



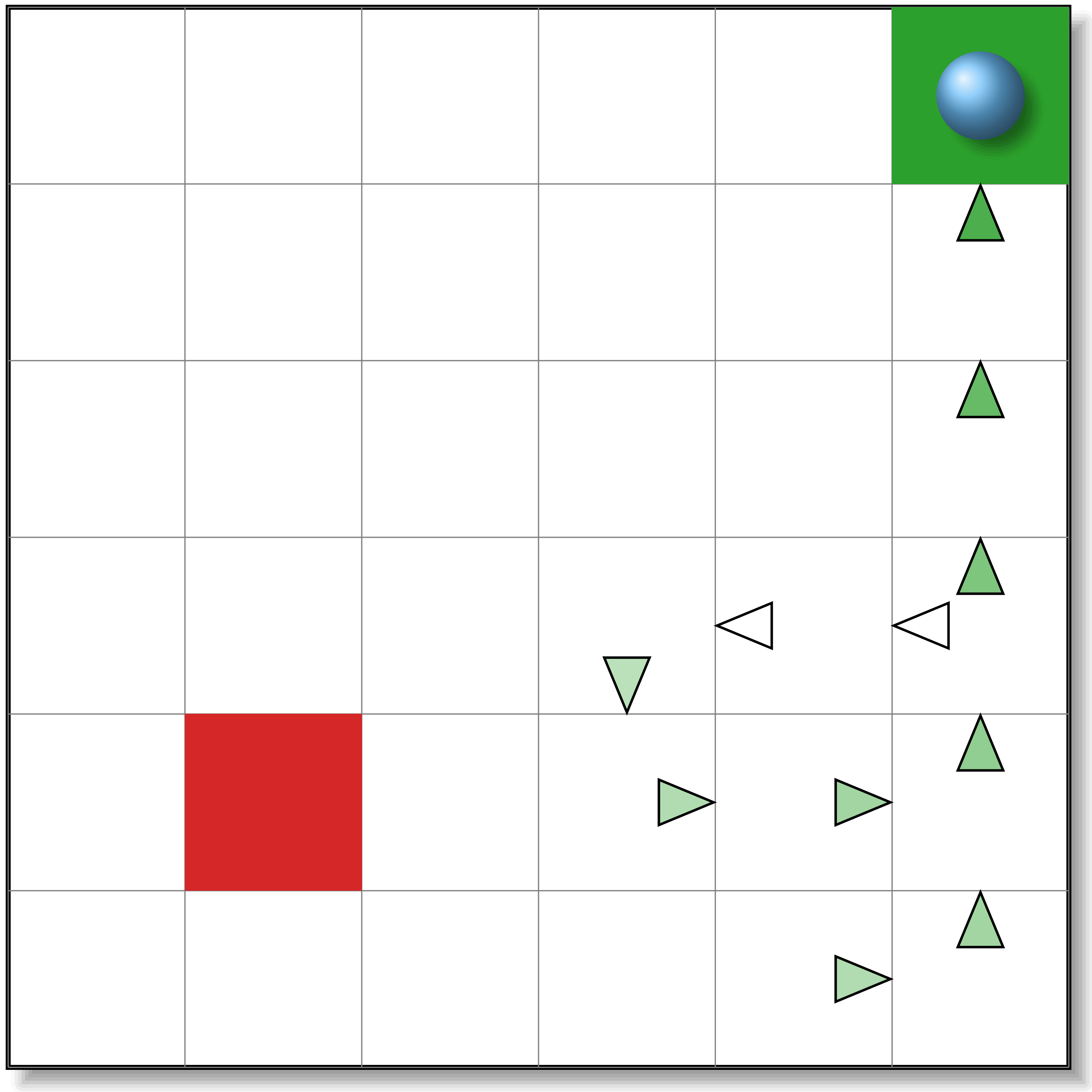
prioritized sweeping



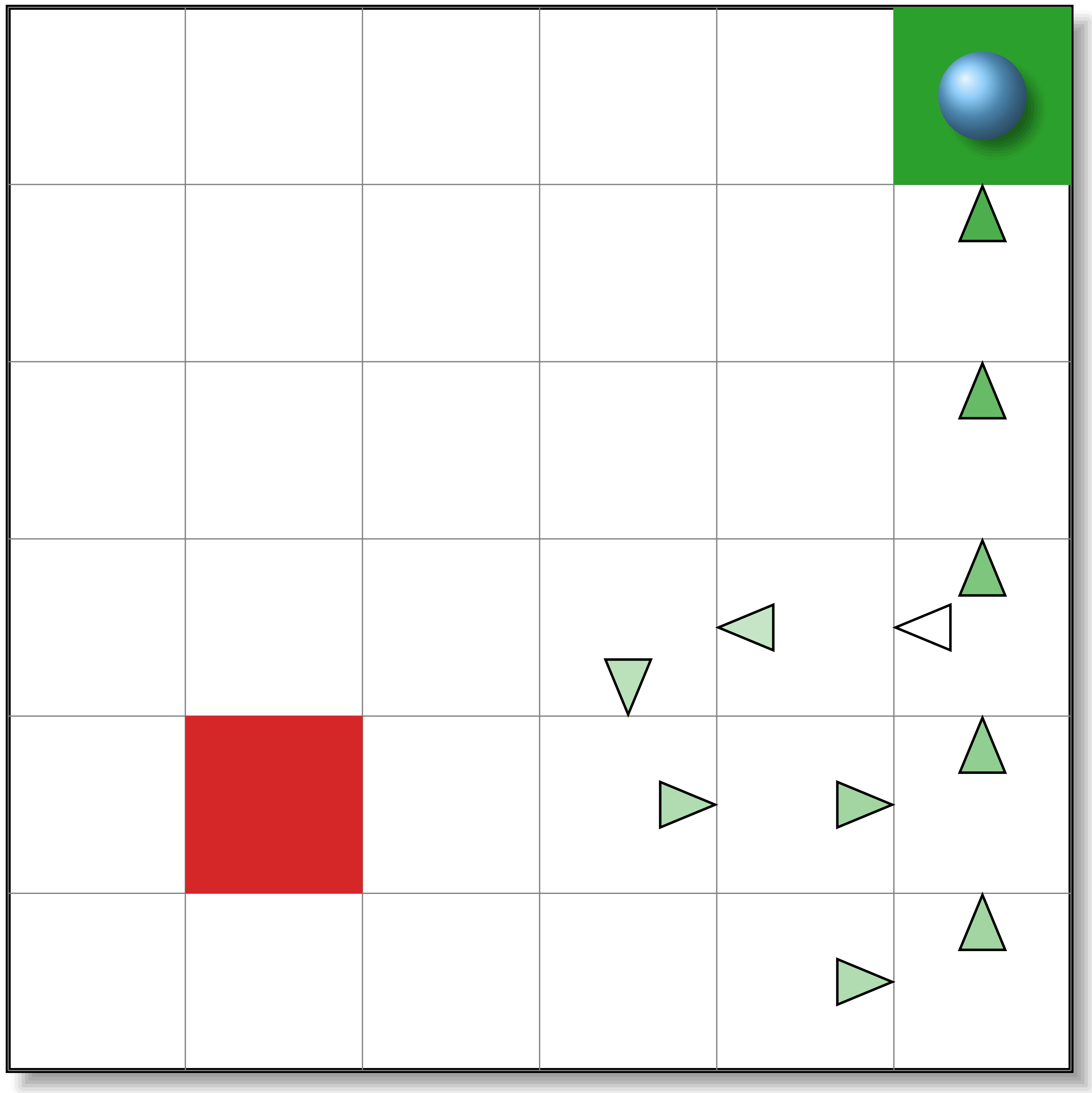
prioritized sweeping



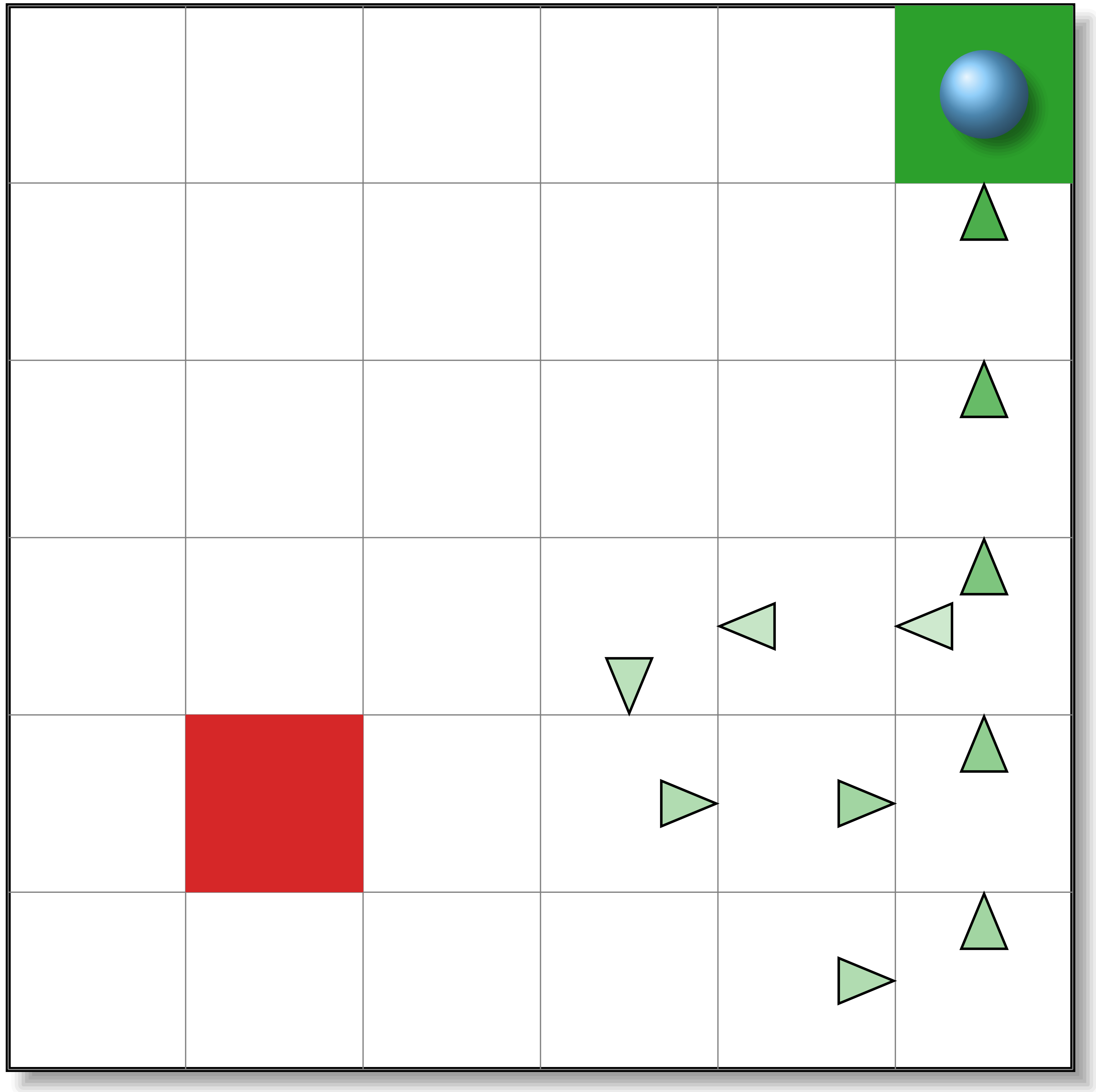
prioritized sweeping



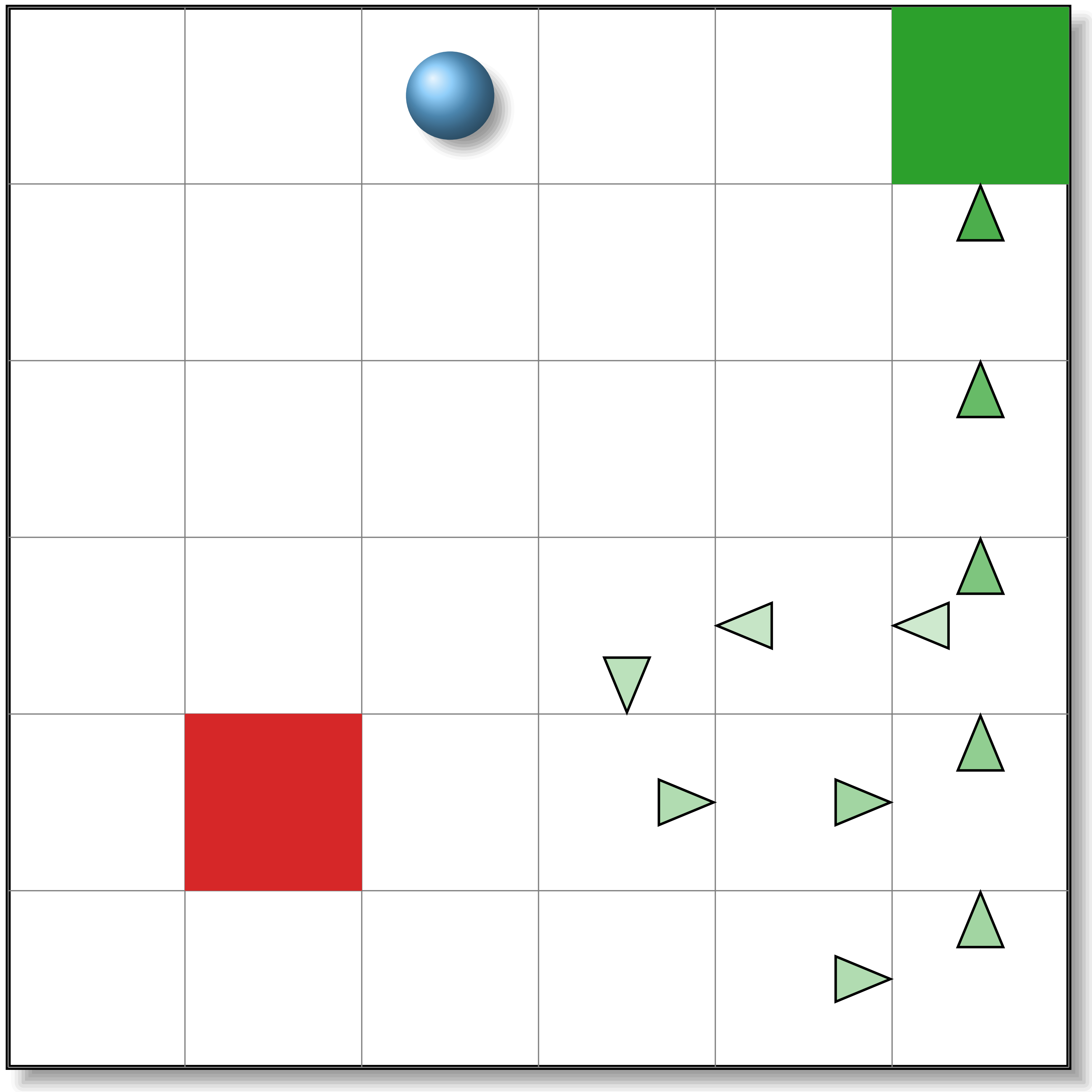
prioritized sweeping



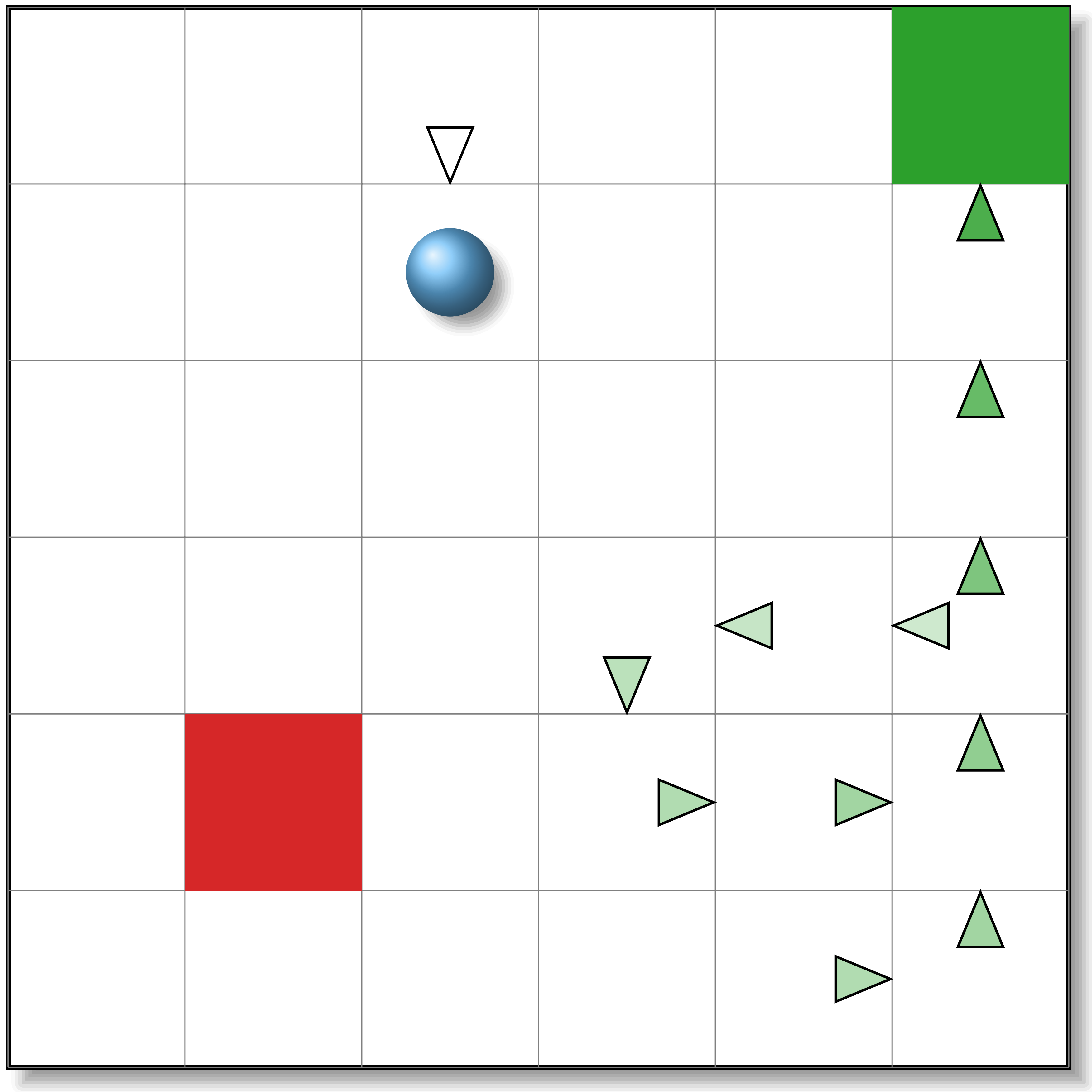
prioritized sweeping



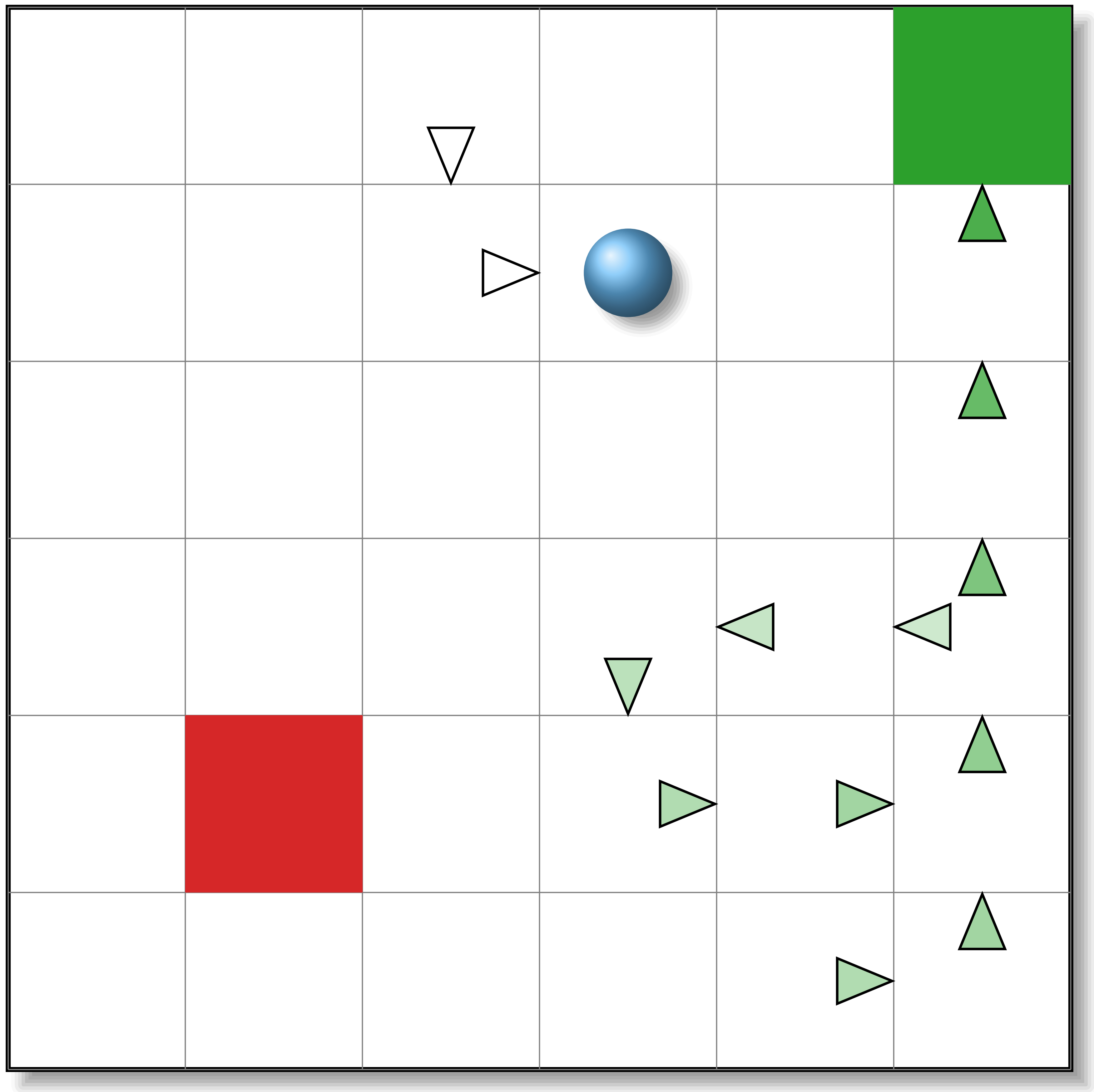
prioritized sweeping



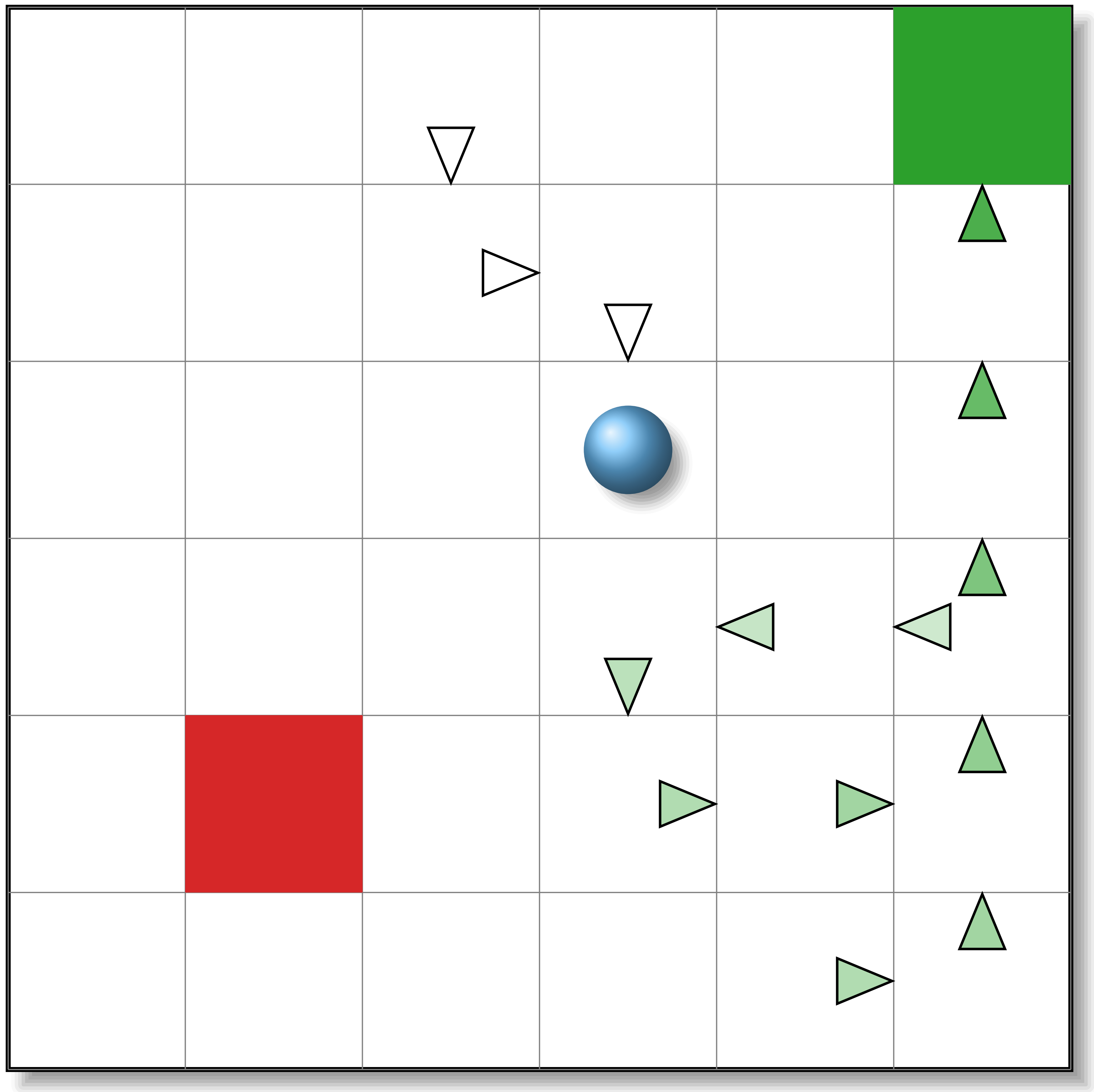
prioritized sweeping



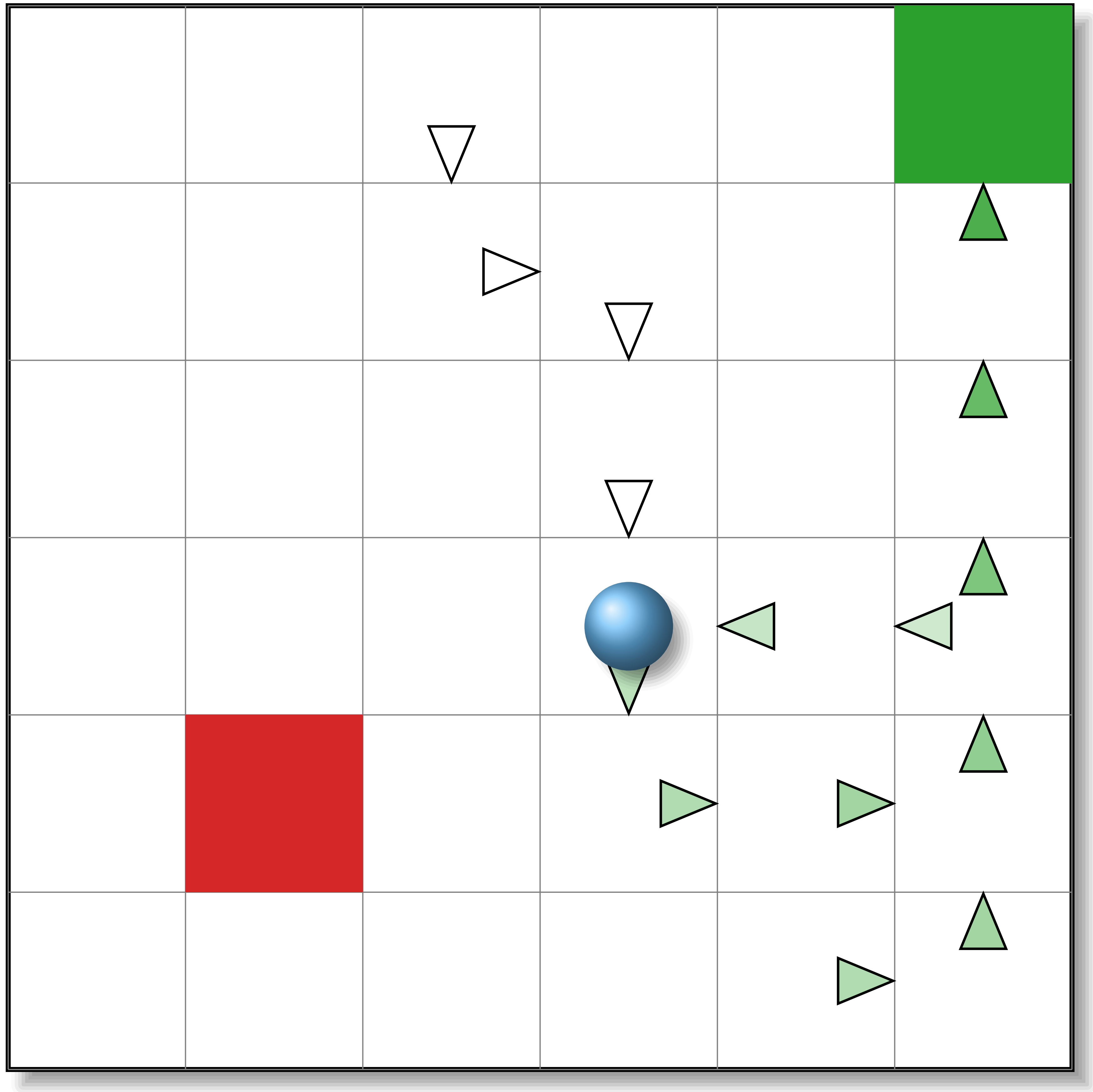
prioritized sweeping



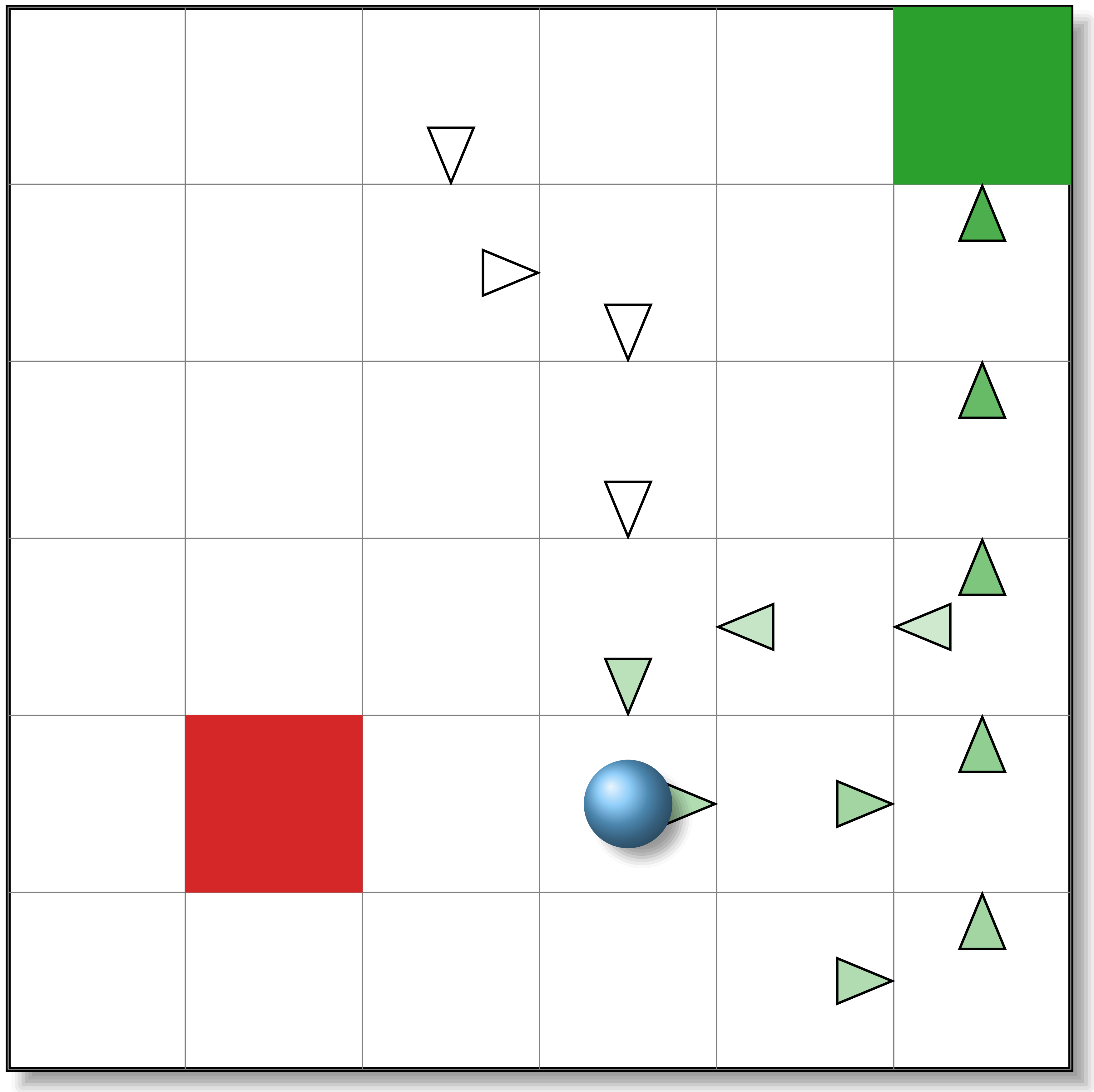
prioritized sweeping



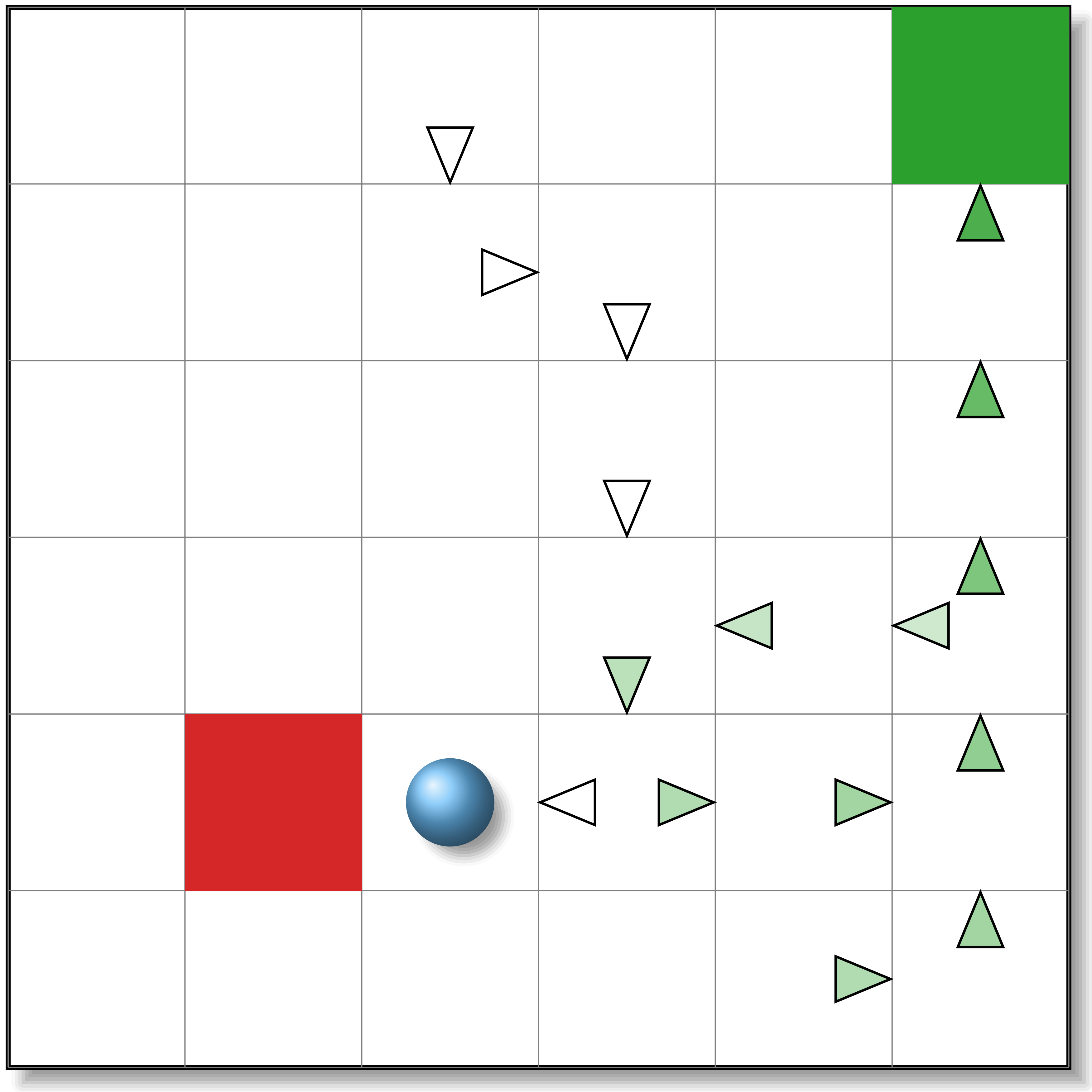
prioritized sweeping



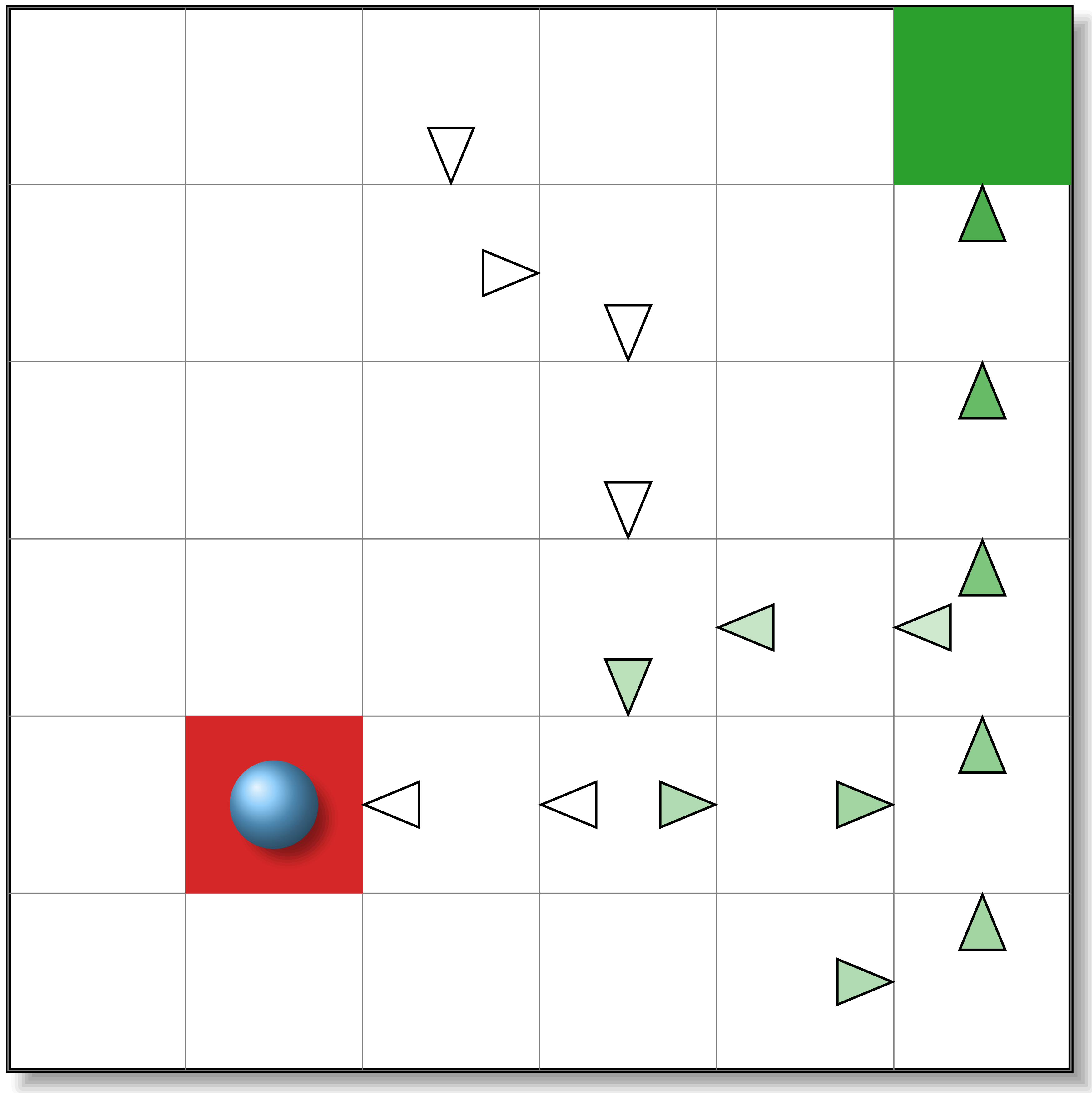
prioritized sweeping



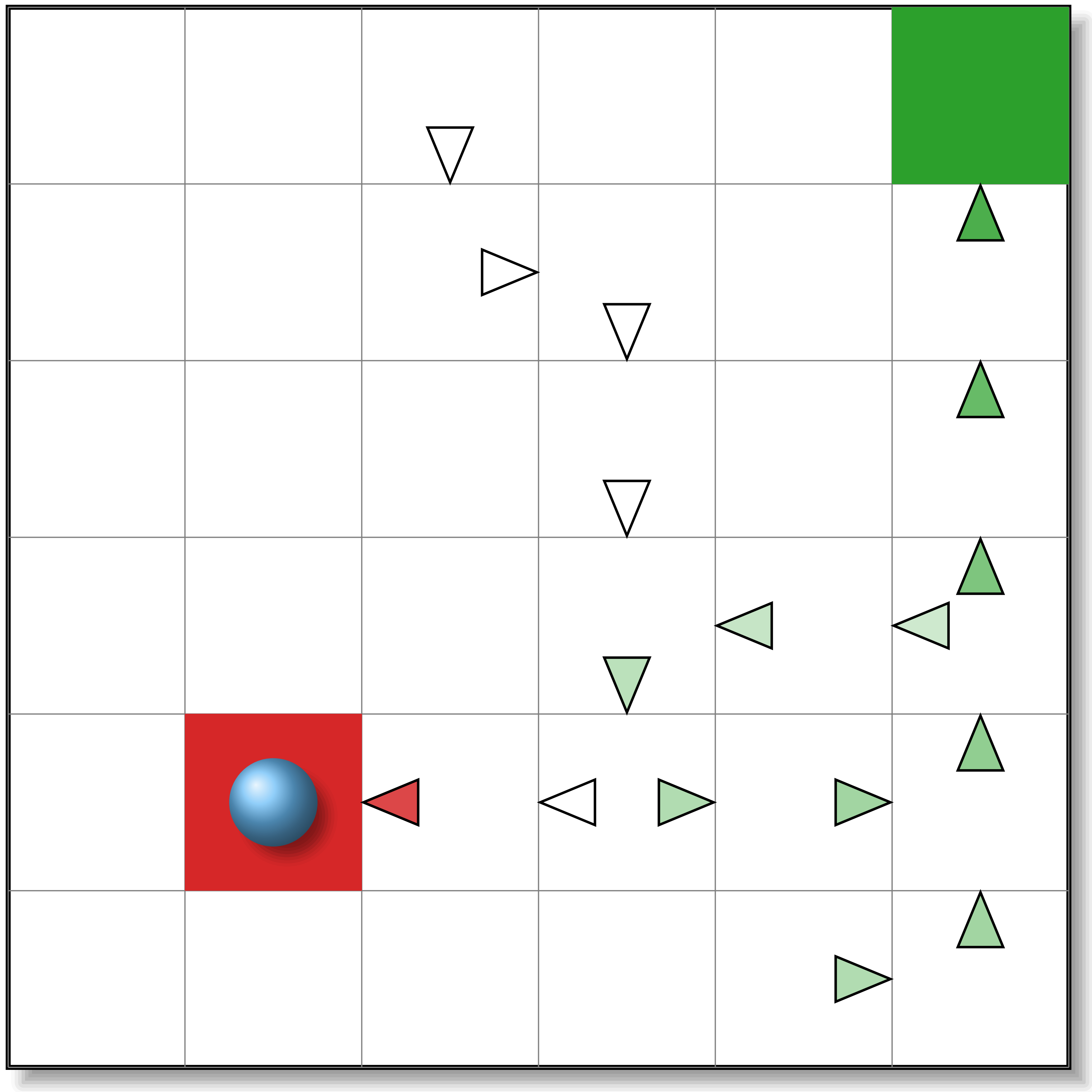
prioritized sweeping



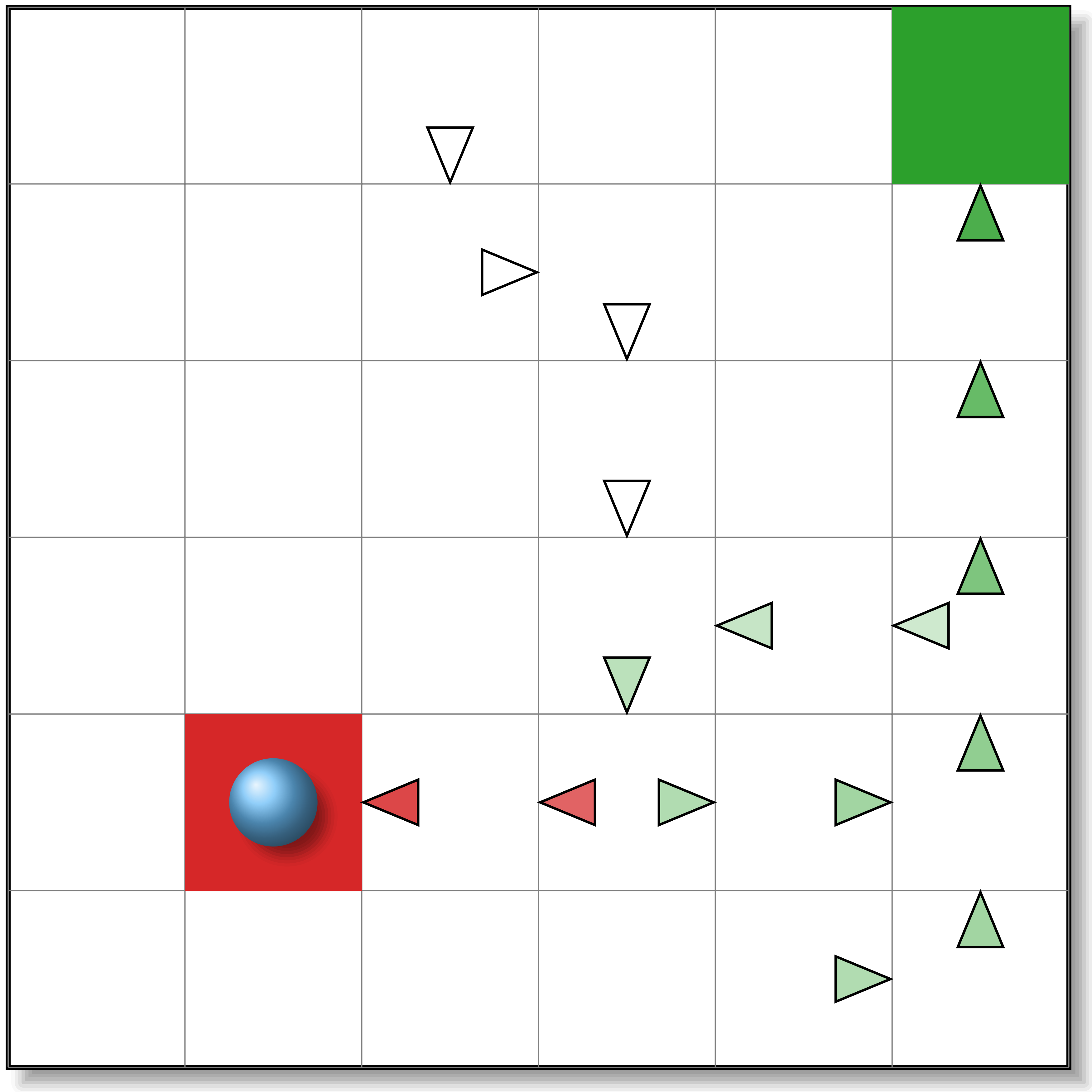
prioritized sweeping



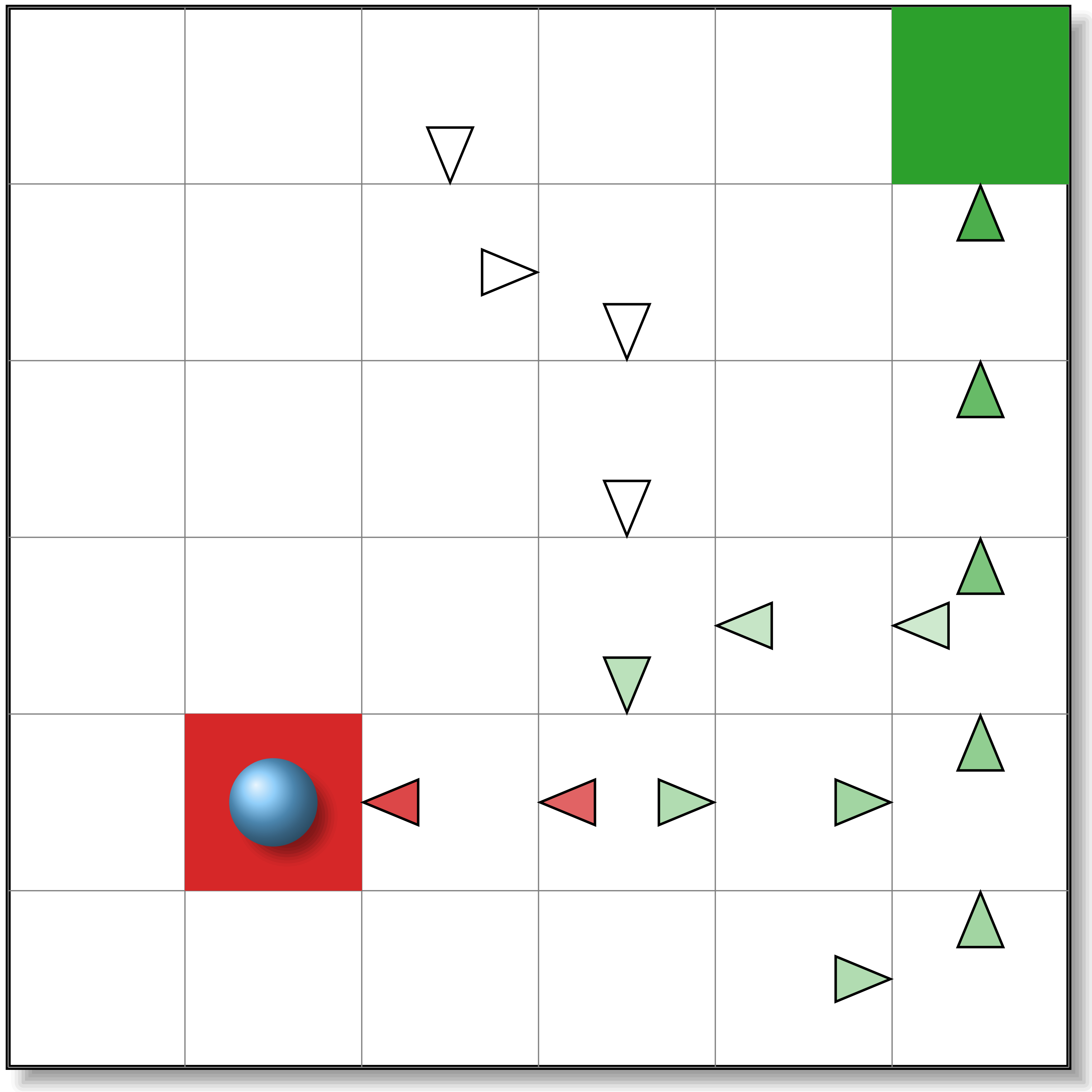
prioritized sweeping



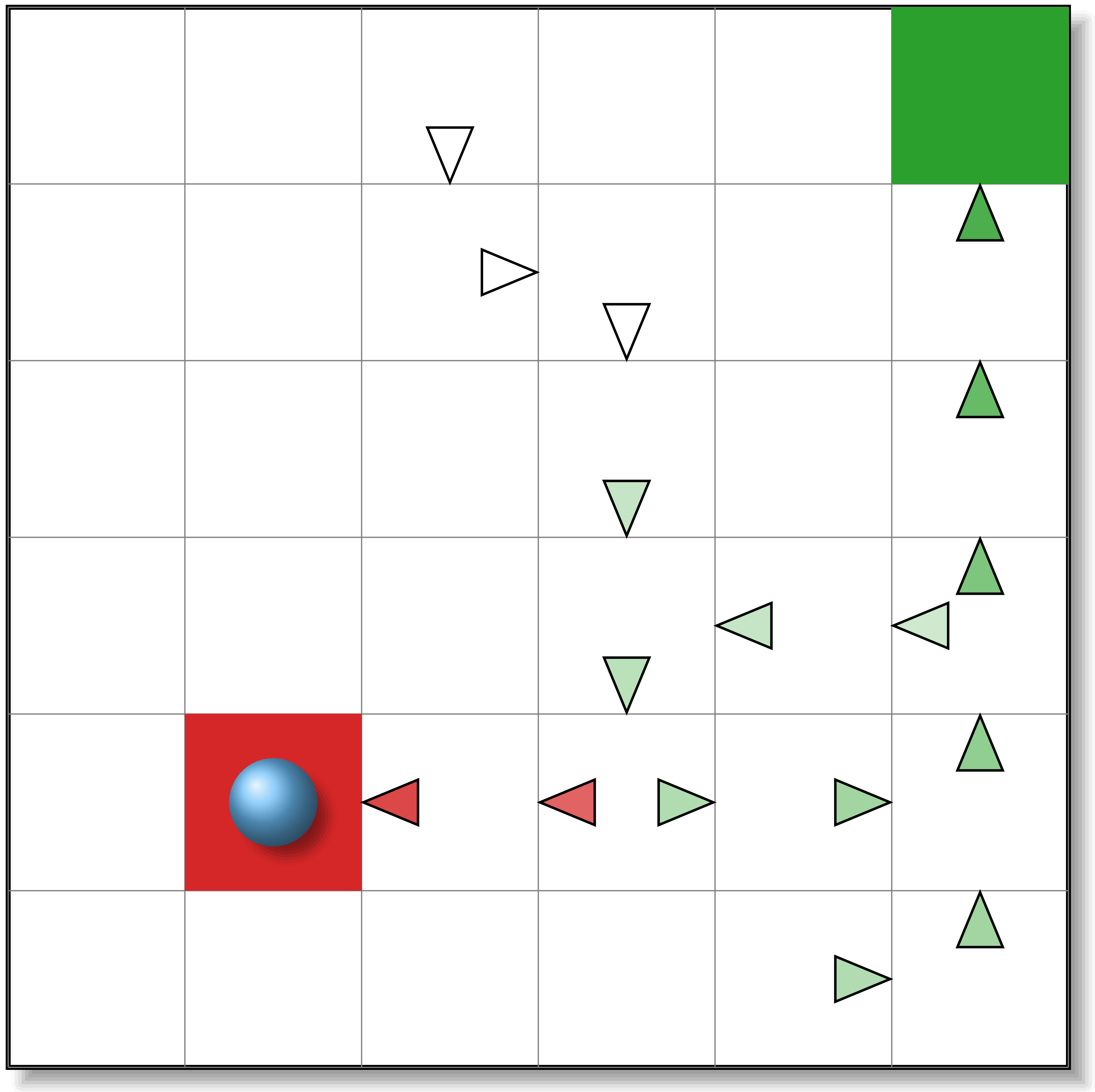
prioritized sweeping



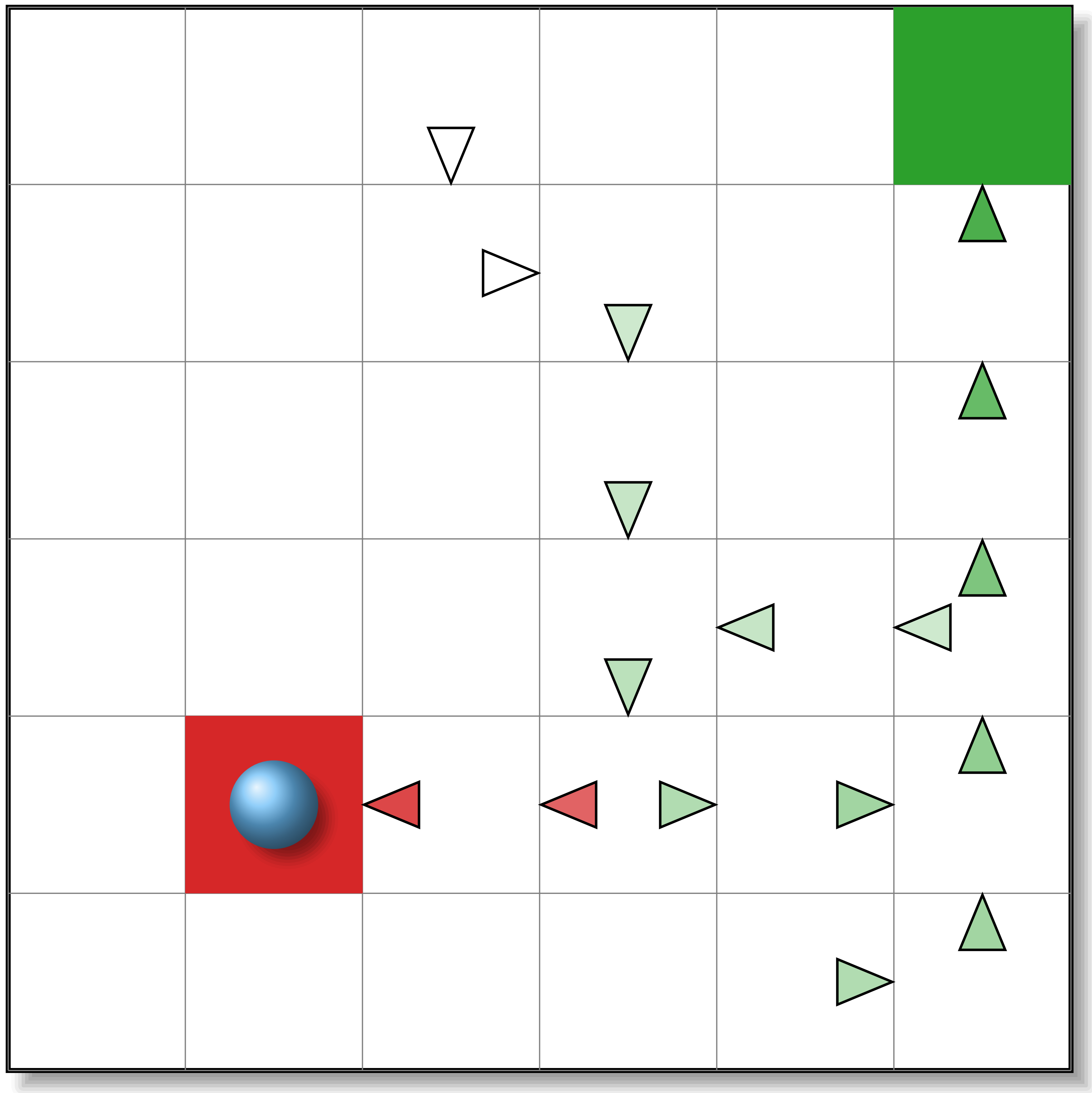
prioritized sweeping



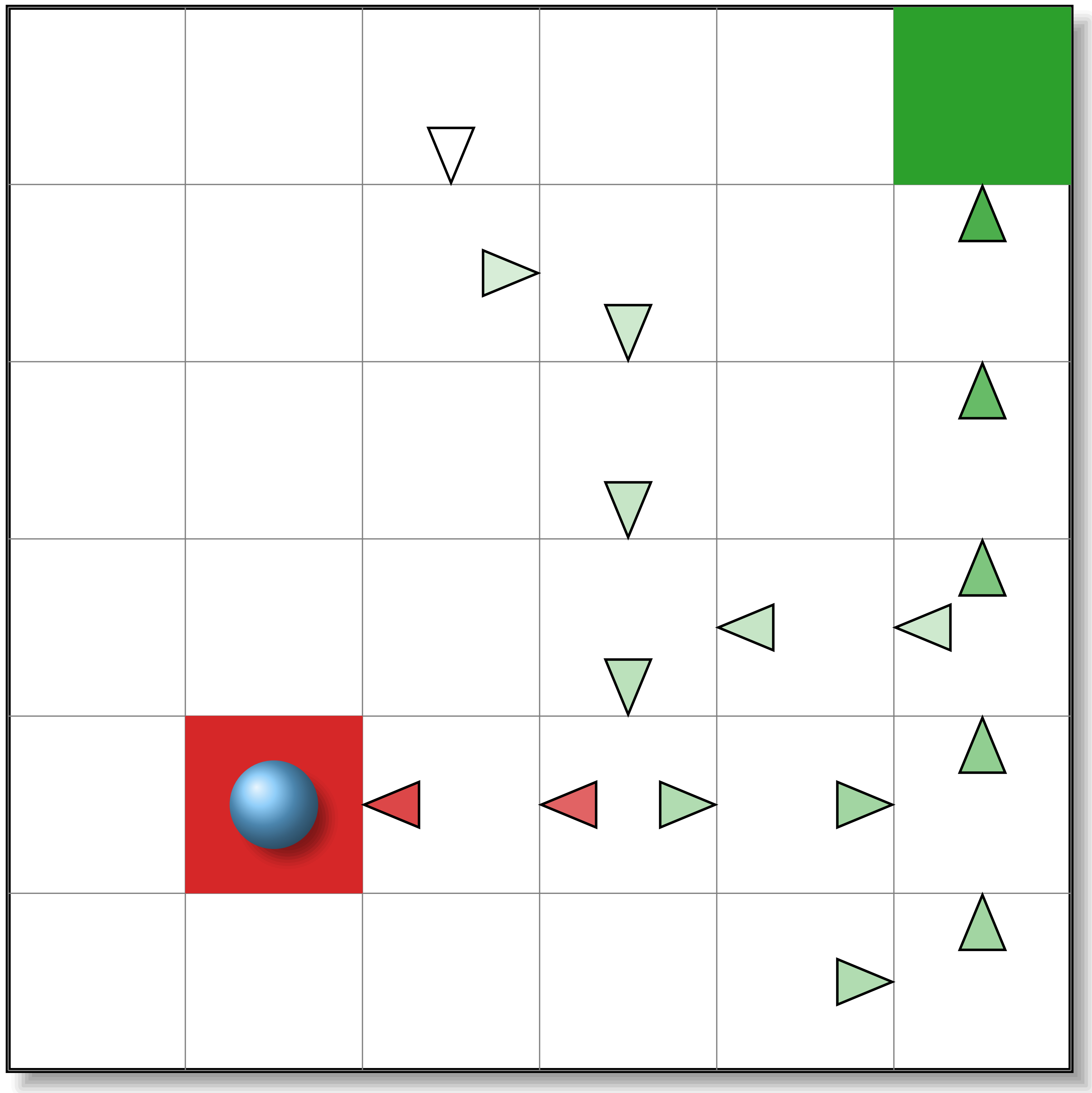
prioritized sweeping



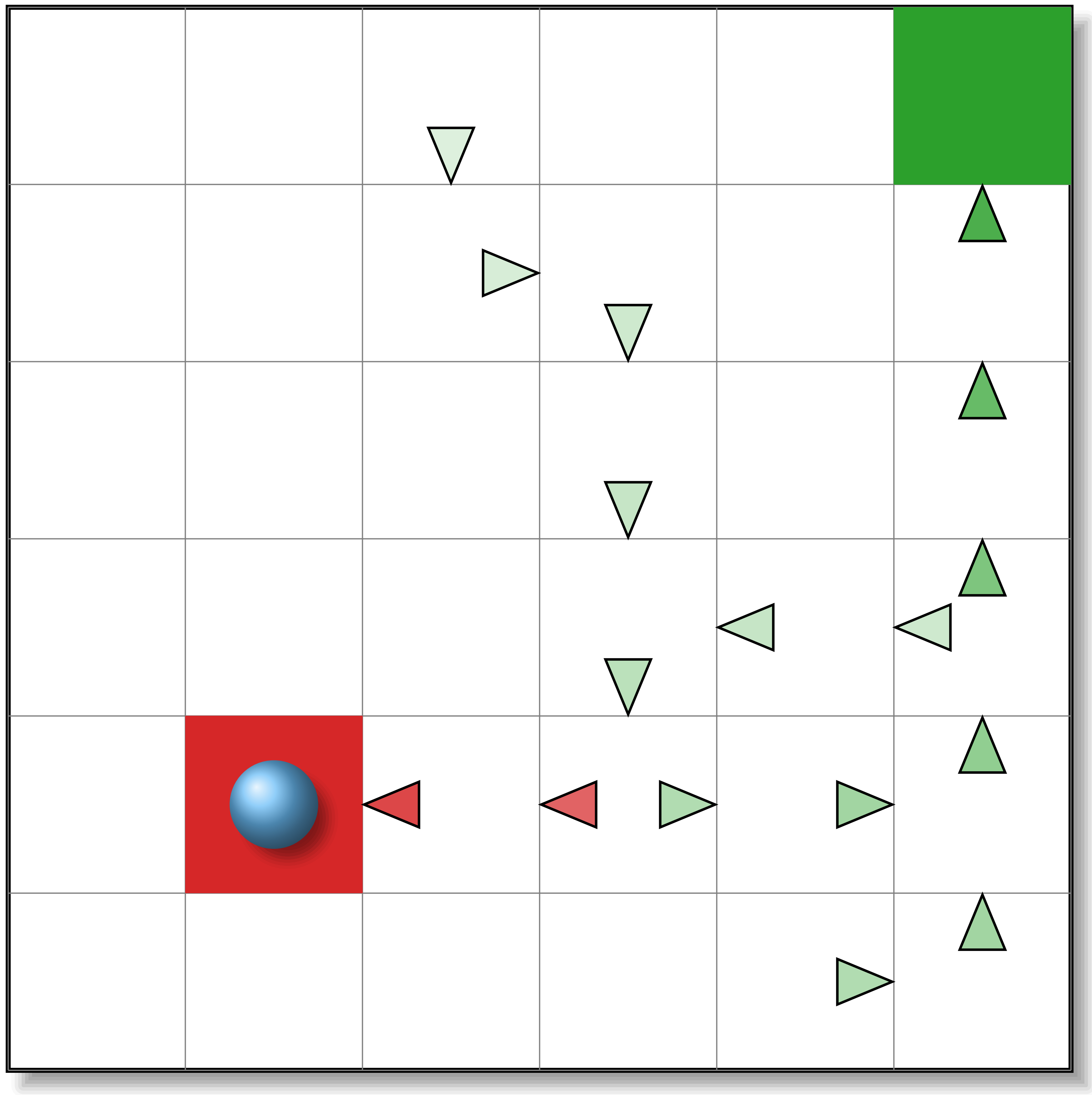
prioritized sweeping



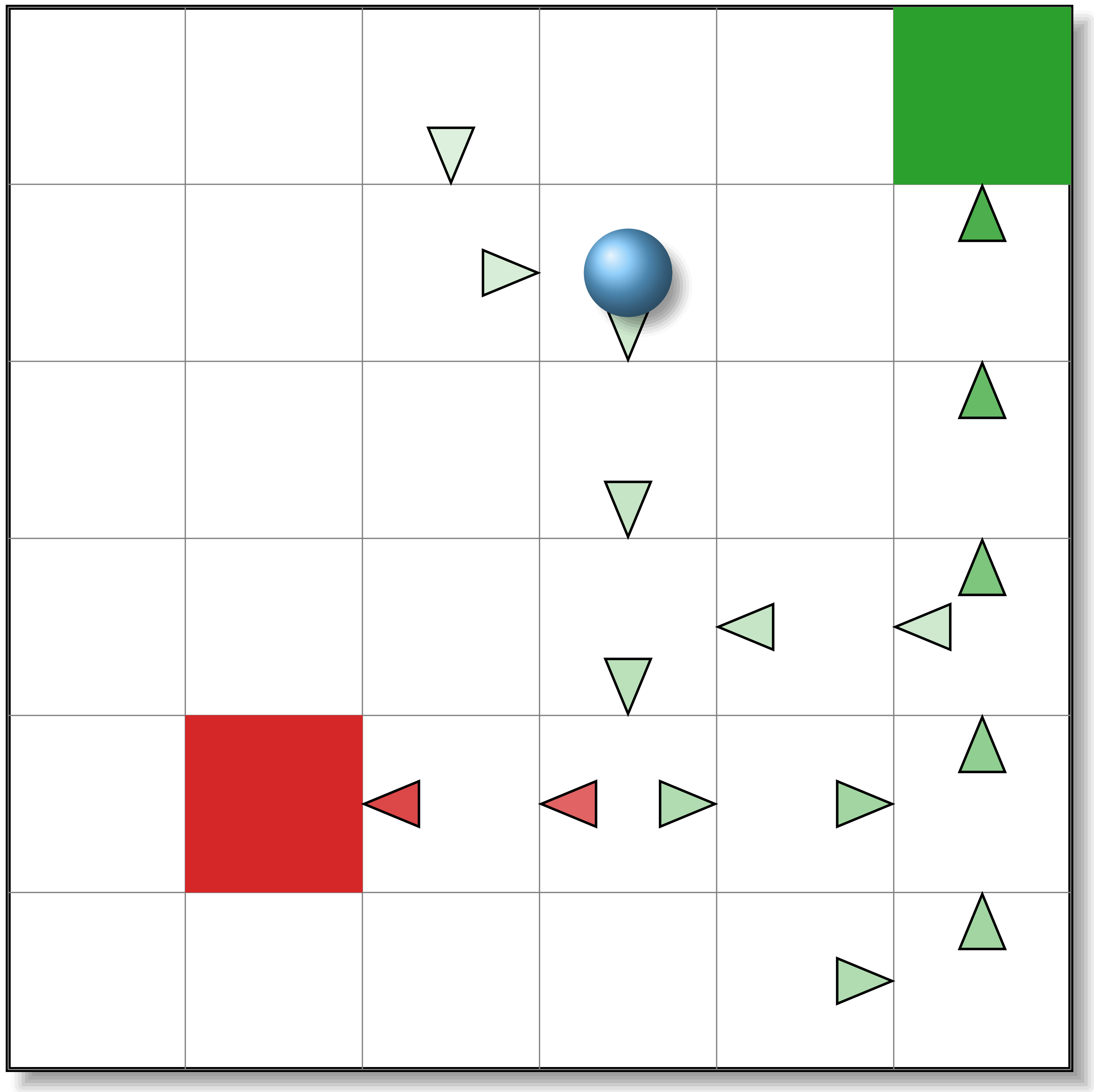
prioritized sweeping



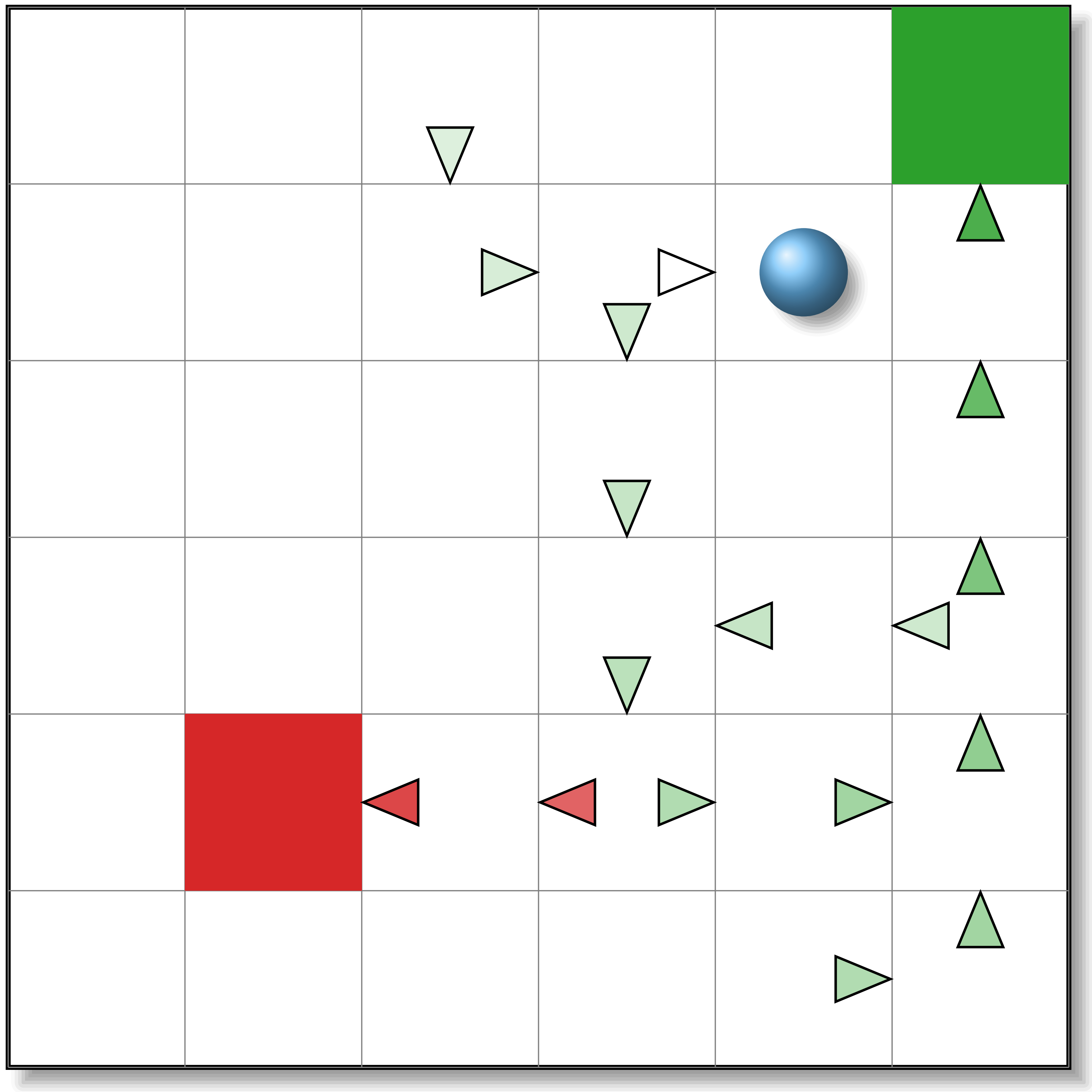
prioritized sweeping



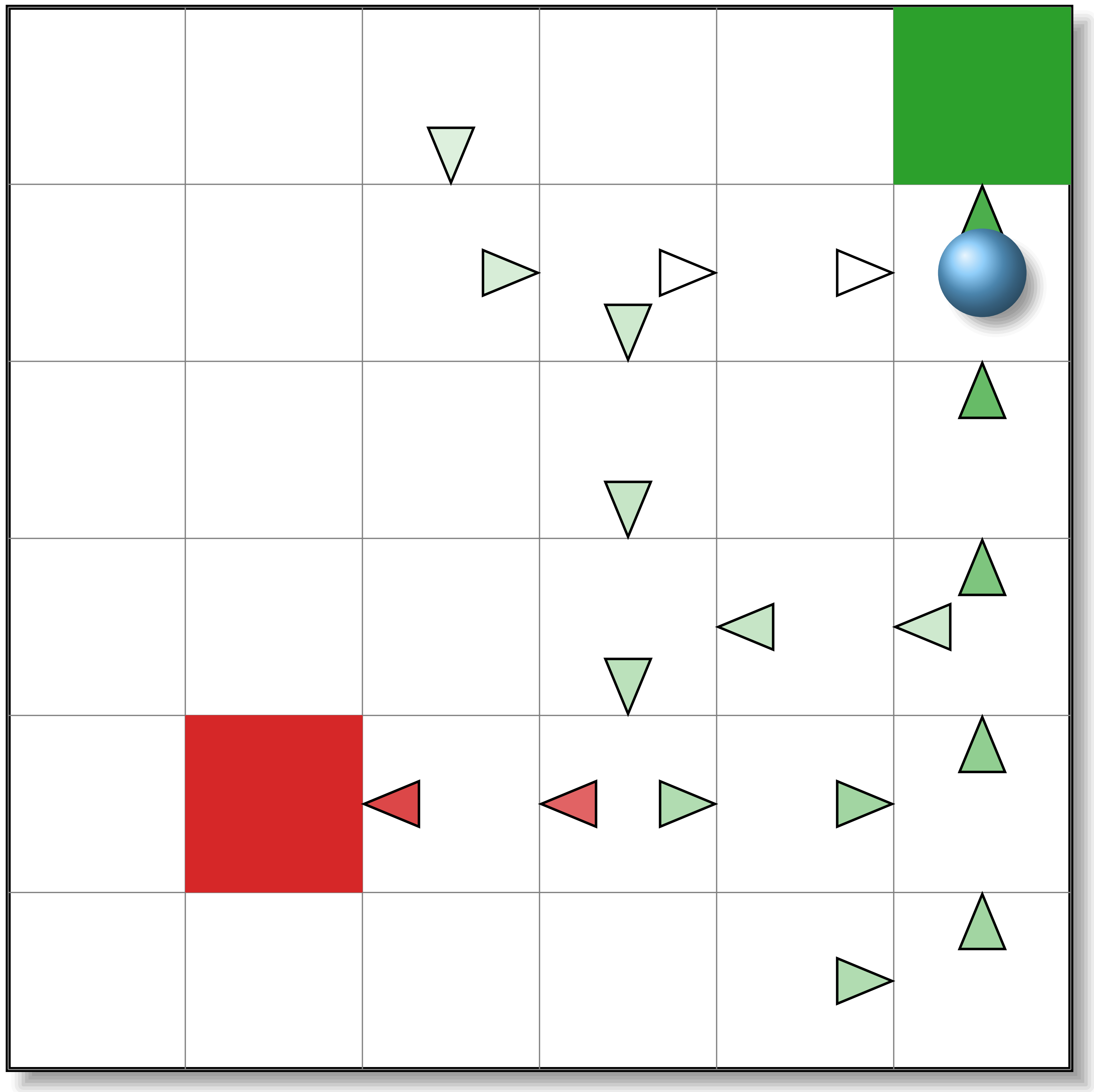
prioritized sweeping



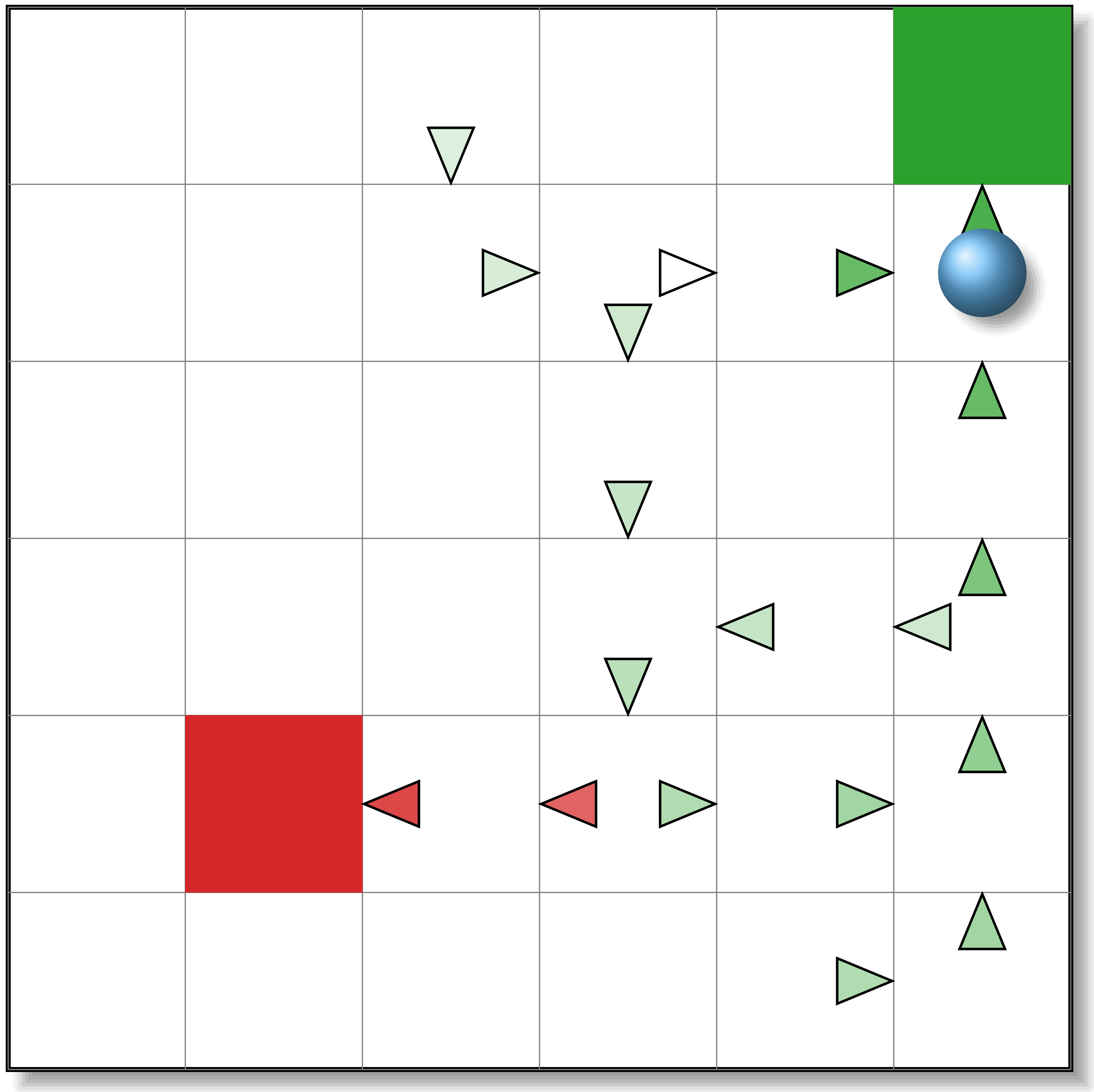
prioritized sweeping



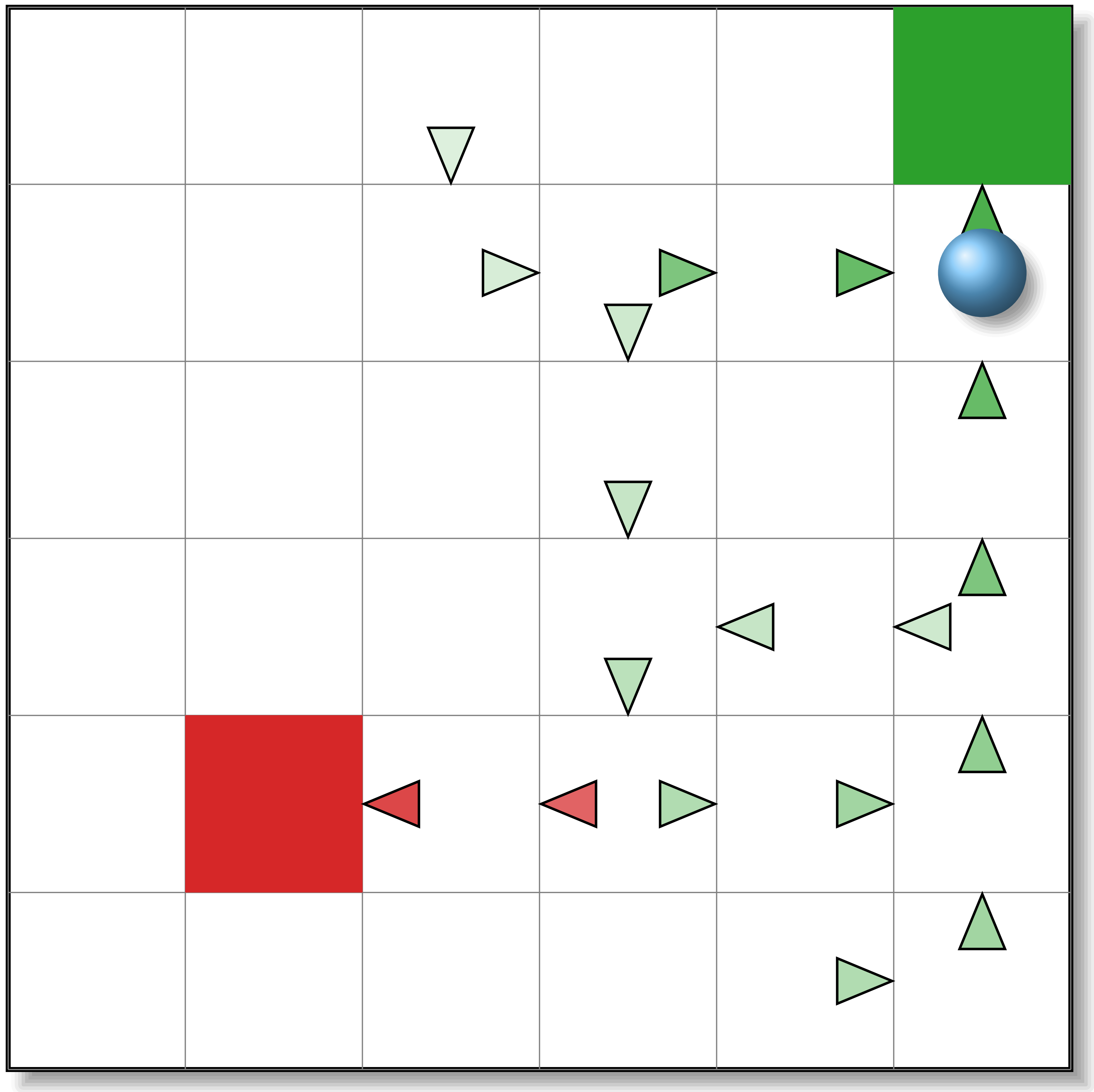
prioritized sweeping



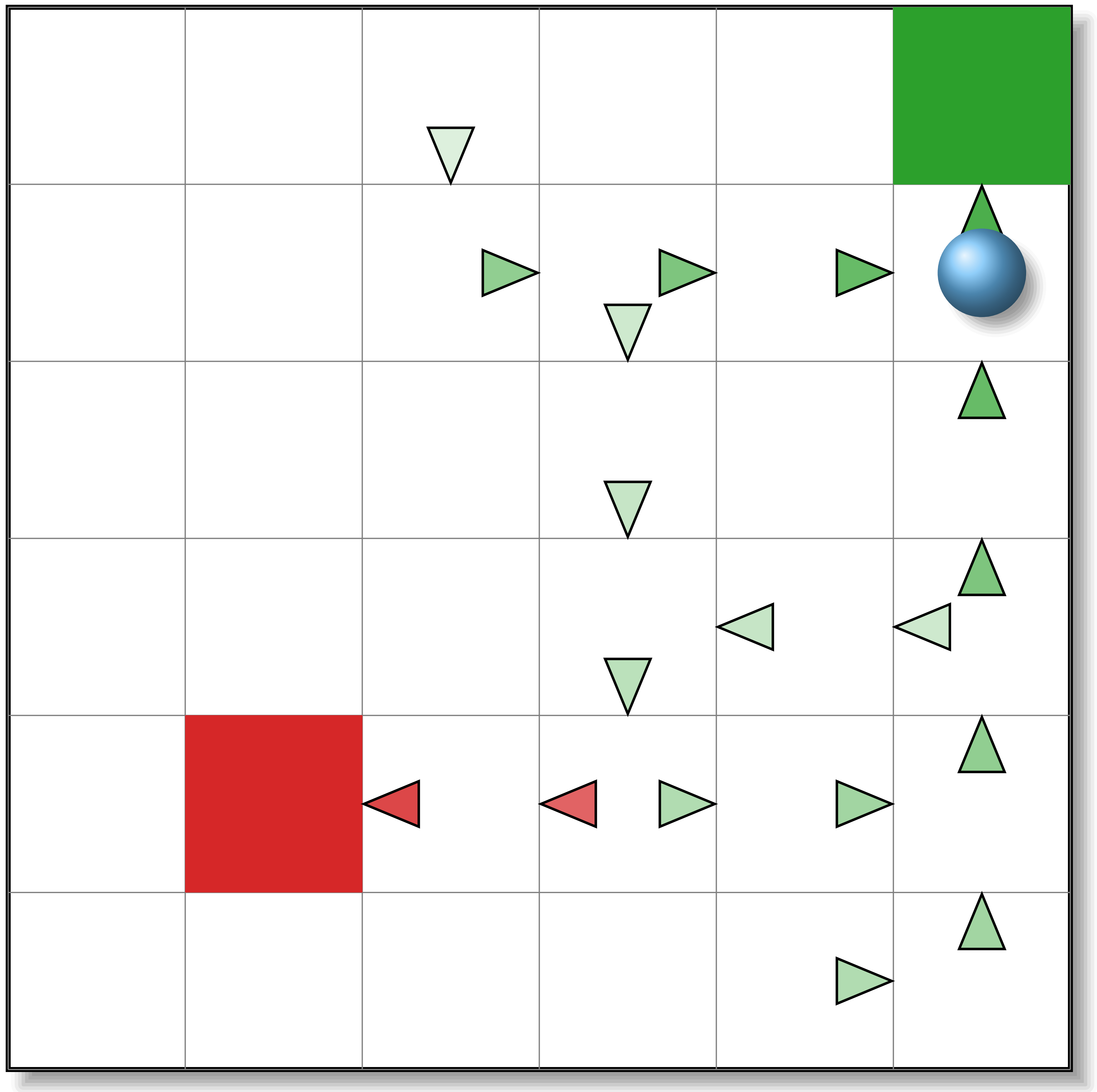
prioritized sweeping



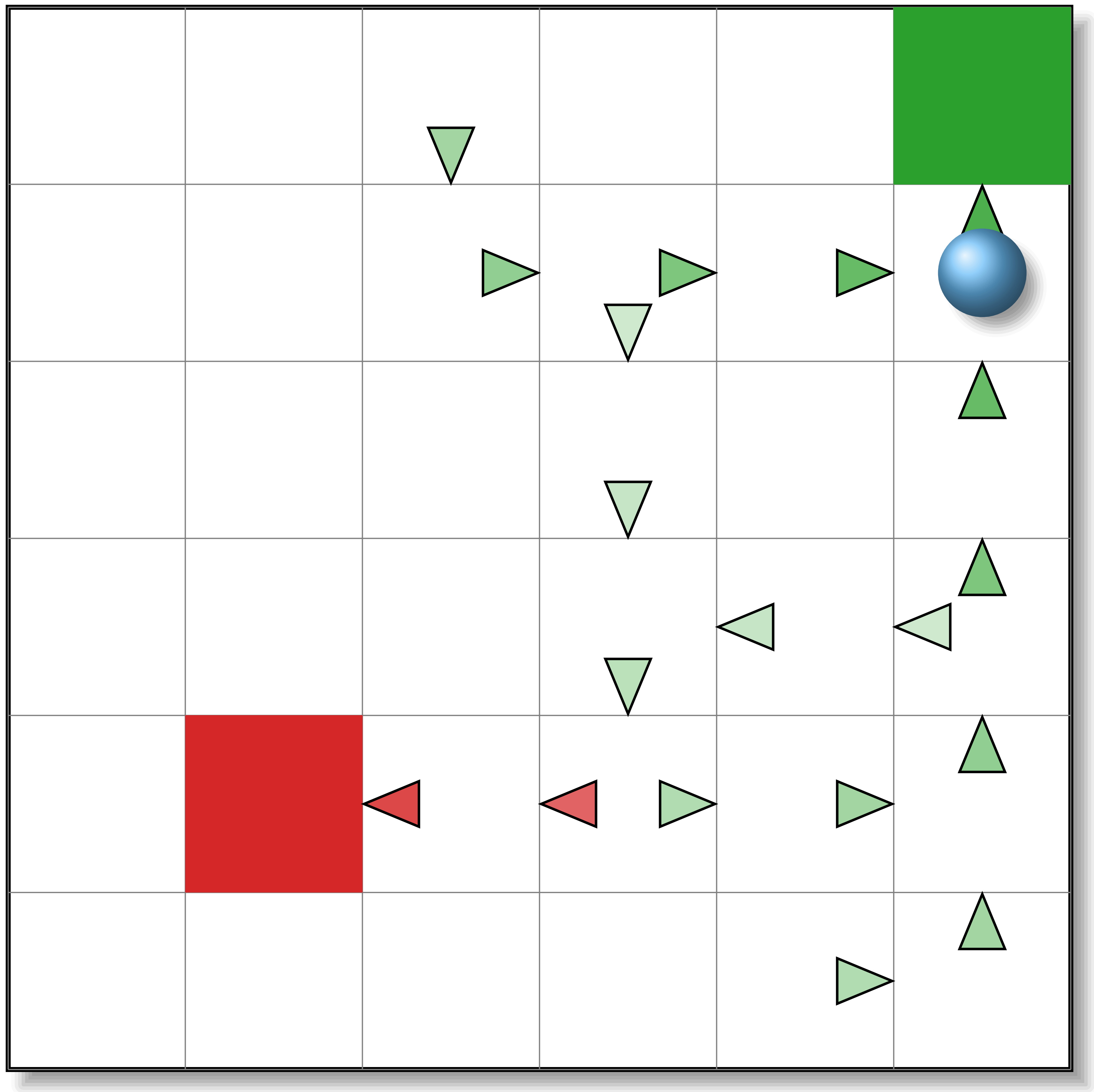
prioritized sweeping



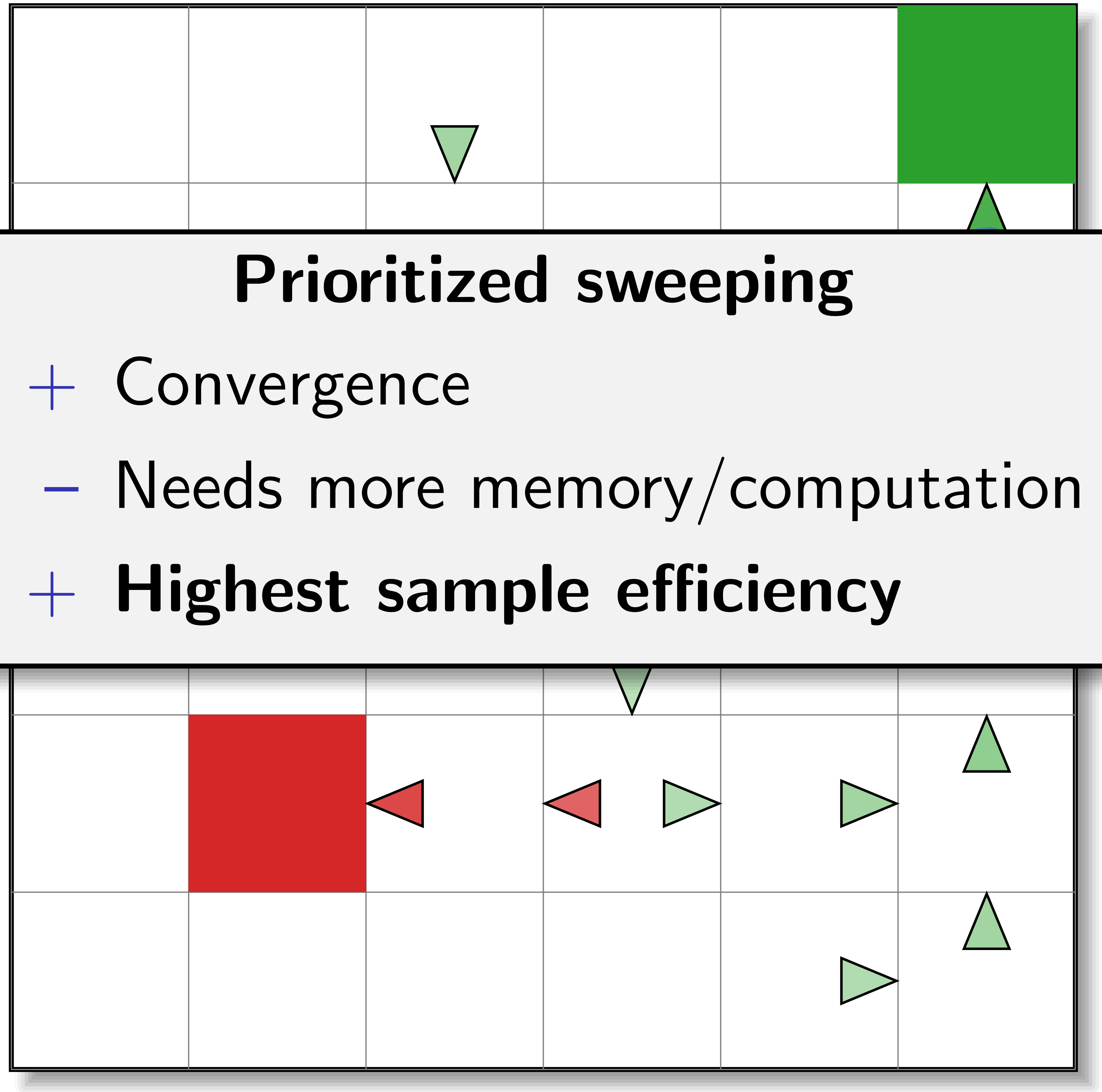
prioritized sweeping



prioritized sweeping



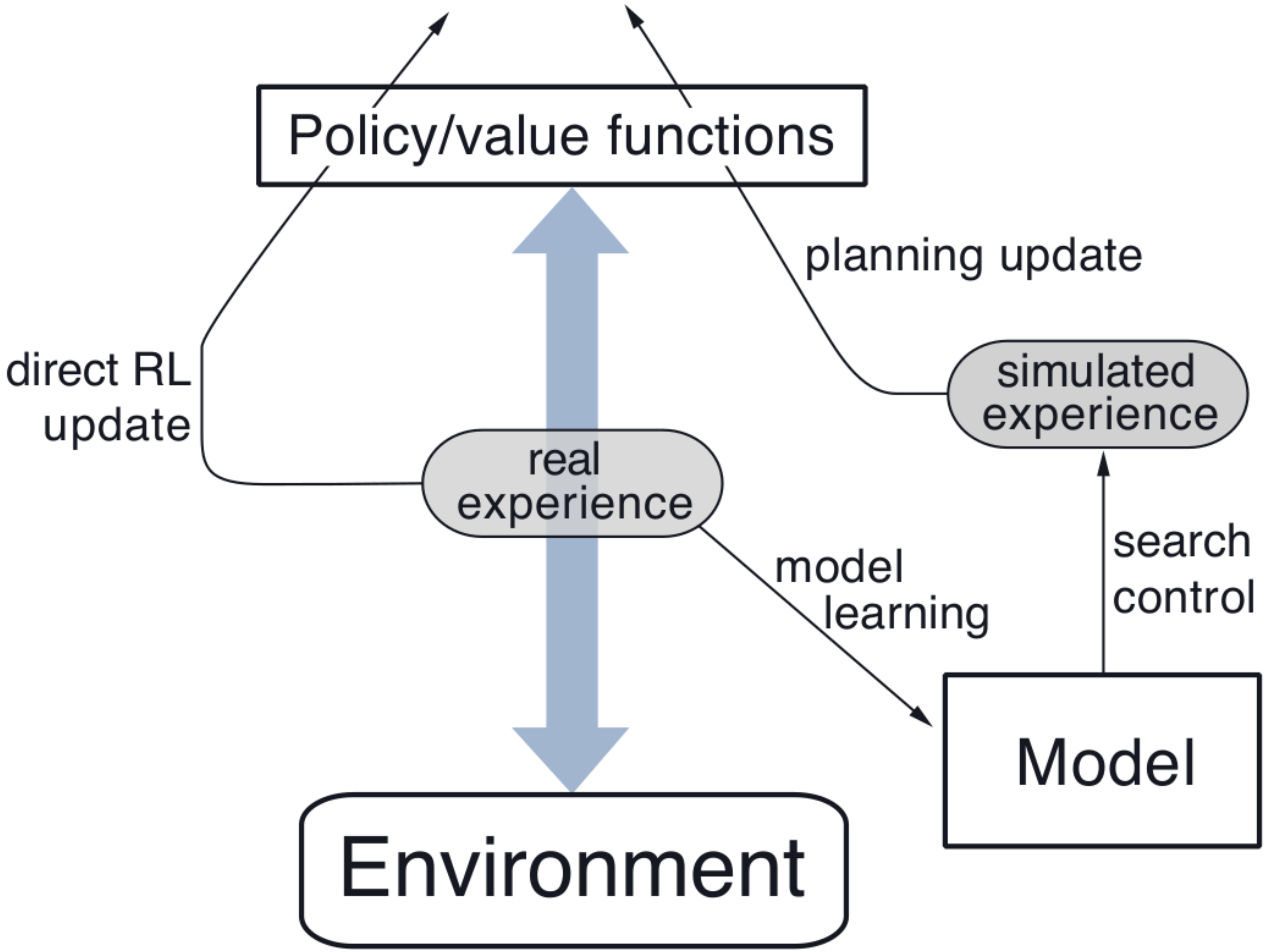
prioritized sweeping



Prioritized sweeping

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

Dyna Architecture



Summary and Conclusions

1. Methods that estimate and use $P^a_{s \to s'}$ for planning are called **model-based**.
2. The **Dyna Architecture** visualizes, how model-based methods could work.
3. Prioritized sweeping 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.
[ ] Prioritized sweeping is a model-based algorithm because it uses $N^a_{s \to s'}/N^a_s$ to backup the Q-values.
[ ] 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.
[ ] Prioritized DQN learns faster than DQN, because the samples lead to better propagation of changes in Q-values.
[ ] 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?

- $P^a_{s \to s'}$ is perfectly known => planning methods can be applied
- State space is large
  Chess ~ 10^{40} – 10^{50} positions
  Go 19x19 ~ 10^{170} positions
  (number of atoms on earth) ~ 10^{50}
- Action space is not small
  Chess ~ 10 – 30 actions per position
- Go ~ 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 $\textsc{Max-Value}(s, \alpha, \beta)$
  if terminal(s) **return** V(s)
  $v = -\infty$
  for all c in next-states(s) **do**
    $v' = \textsc{Min-Value}(c, \alpha, \beta)$
    if $v' > v, v = v'$
    if $v' \geq \beta$ **return** v
    if $v' > \alpha, \alpha = v'$
  end for
  return v
end function

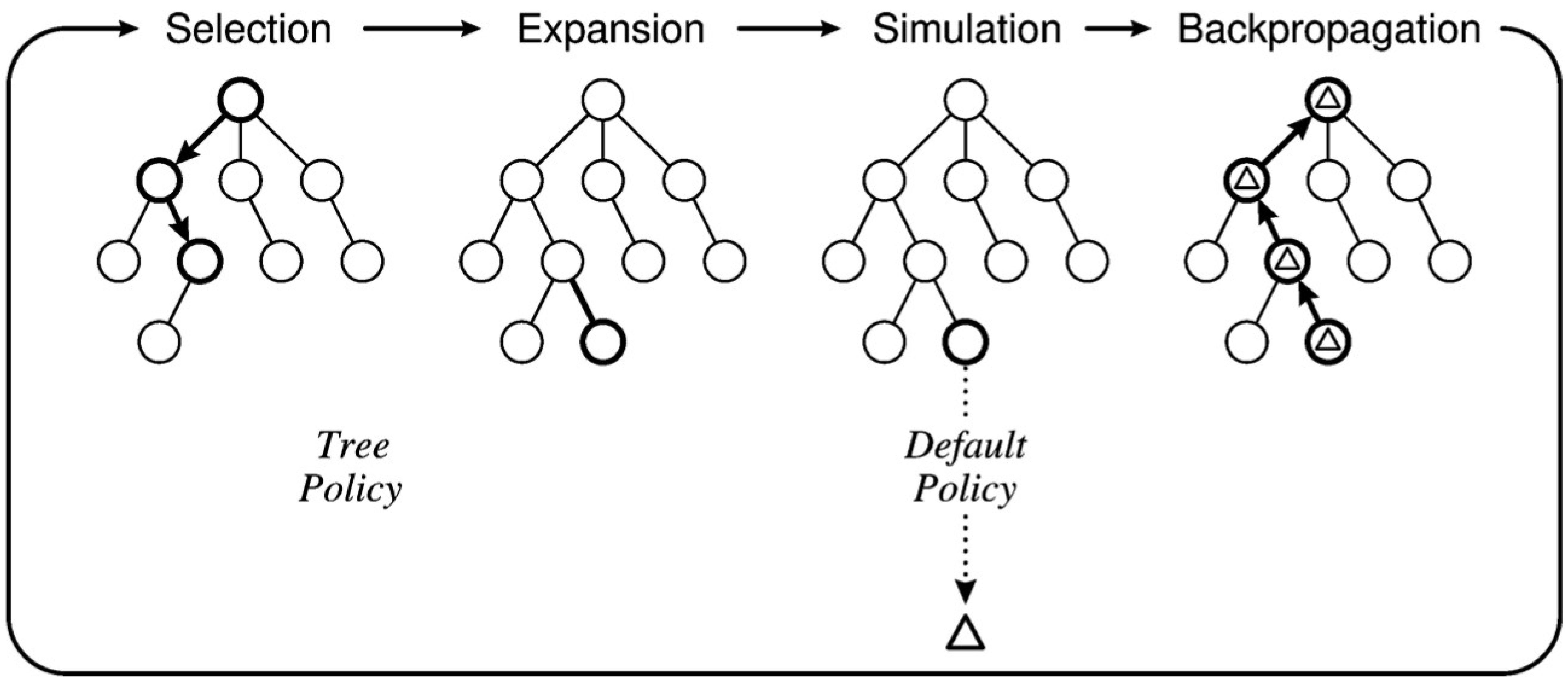
function $\textsc{Min-Value}(s, \alpha, \beta)$
  if terminal(s) **return** V(s)
  $v = \infty$
  for all c in next-states(s) **do**
    $v' = \textsc{Max-Value}(c, \alpha, \beta)$
    if $v' < v, v = v'$
    if $v' \leq \alpha$ **return** v
    if $v' < \beta, \beta = v'$
  end for
  return v
end function

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

http://inst.eecs.berkeley.edu/~cs61b/fa14/ta-materials/apps/ab_tree_practice/

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

Most important modifications:

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

Prior probability (focus)
learned by neural net

2. Update of Q(s, a) with estimated win probability V(s) computed by a separate output of the neural net instead of just rollout values.

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)

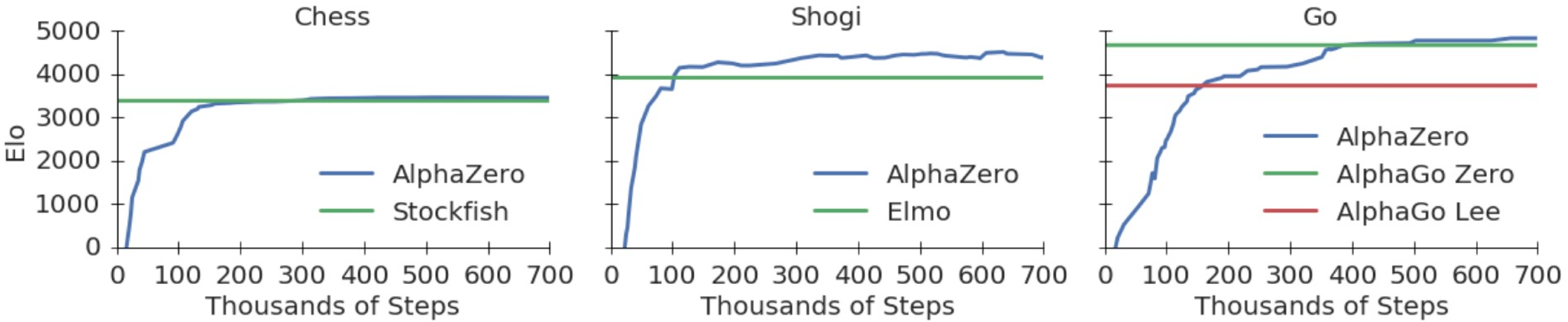
Training:
Uniform sampling of 2048 positions s from last 500 000 games to form a minibatch. Loss:

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

Result of game

Action probability from MCTS

AlphaZero: success story



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

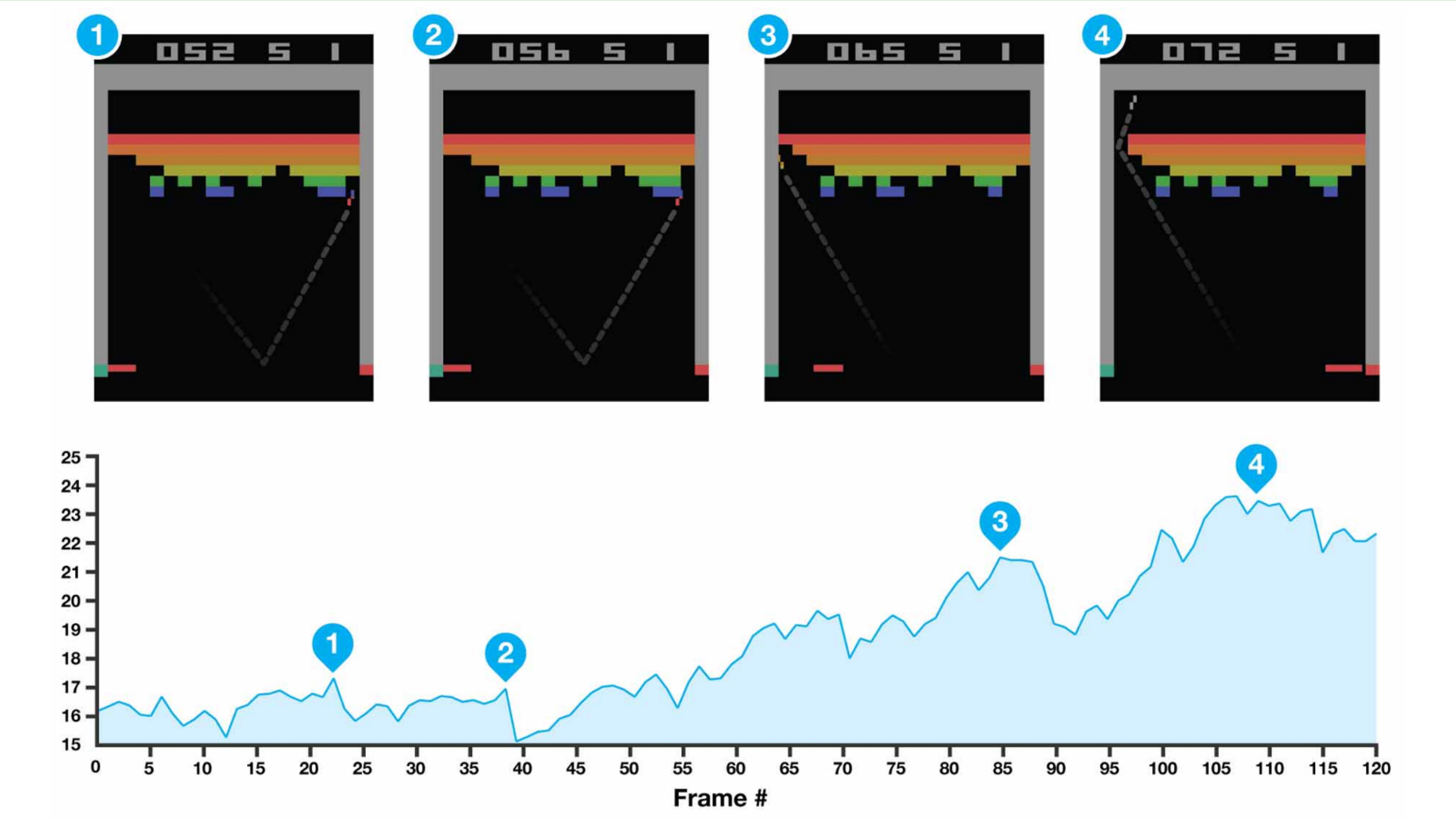
Quiz:

[ ] MiniMax and Monte Carlo Tree Search require $P^a_{s \to s'}$.
[ ] MCTS requires a value function to evaluate the leafs.
[ ] AlphaZero uses a learned value function to update the leaf values.
[ ] The probability of selecting a move in AlphaZero is given by the output of the policy neural network.
[ ] The probability of selecting a move in AlphaZero is determined by MCTS.
[ ] The output of the policy network is used in the selection phase of MCTS.

[ ] Instead of a hand-crafted value function used in chess engines with MiniMax, one could learn the value function through self-play like AlphaZero.

Success stories and limitations of deep reinforcement learning

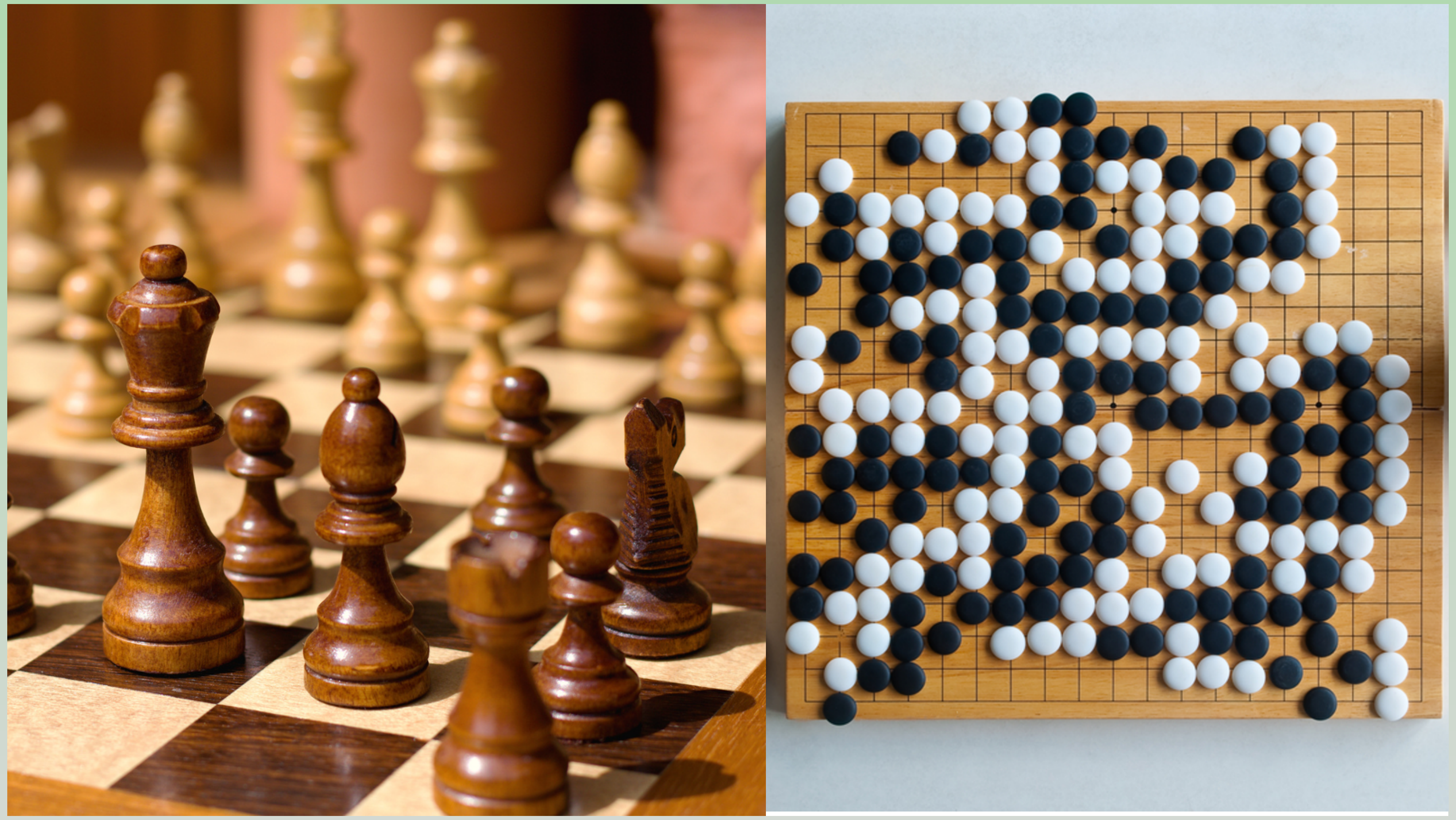
video games



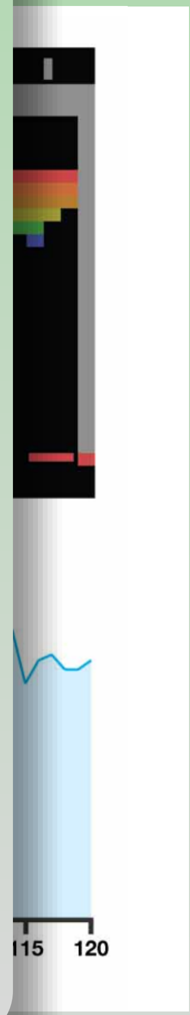
Mnih et al. (2015), Nature

Success stories and limitations of deep reinforcement learning

board games



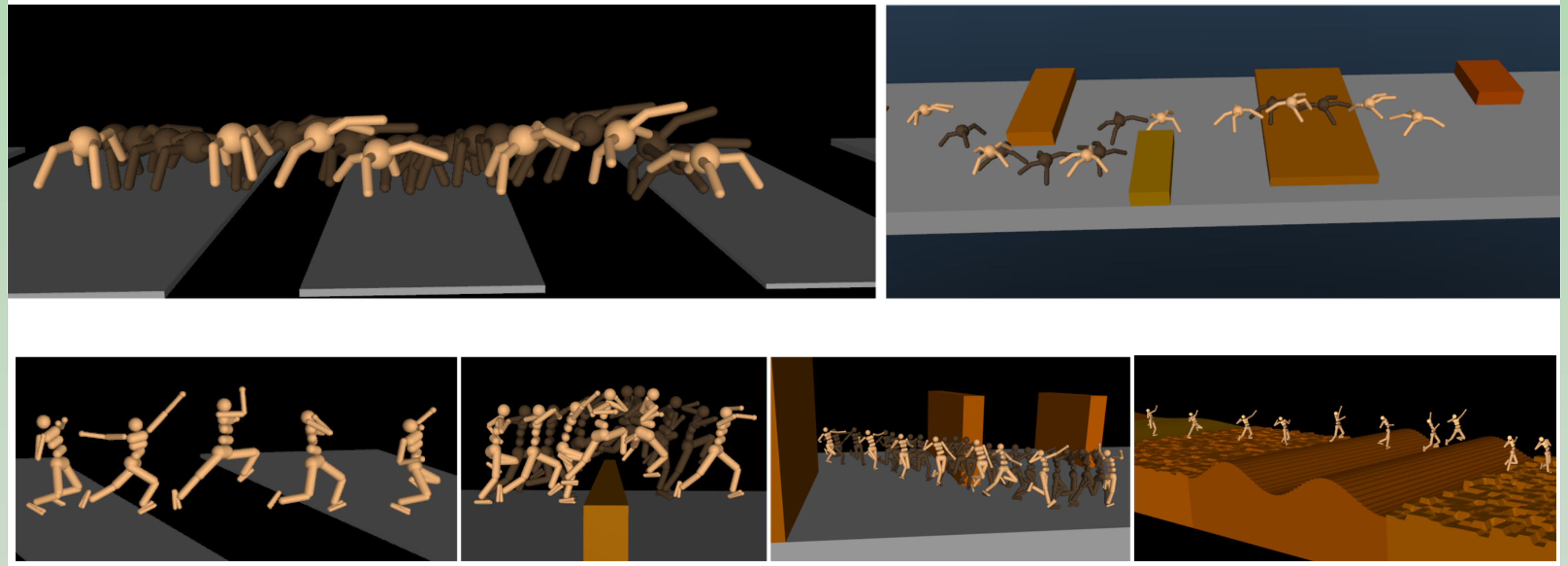
Silver et al. (2017), Arxiv:1712.01815



Mnih et al. (2015), Nature

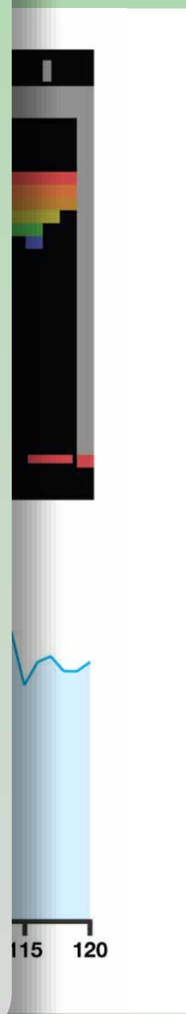
Success stories and limitations of deep reinforcement learning

simulated robotics



Heess et al. (2017), Arxiv:1707.02286

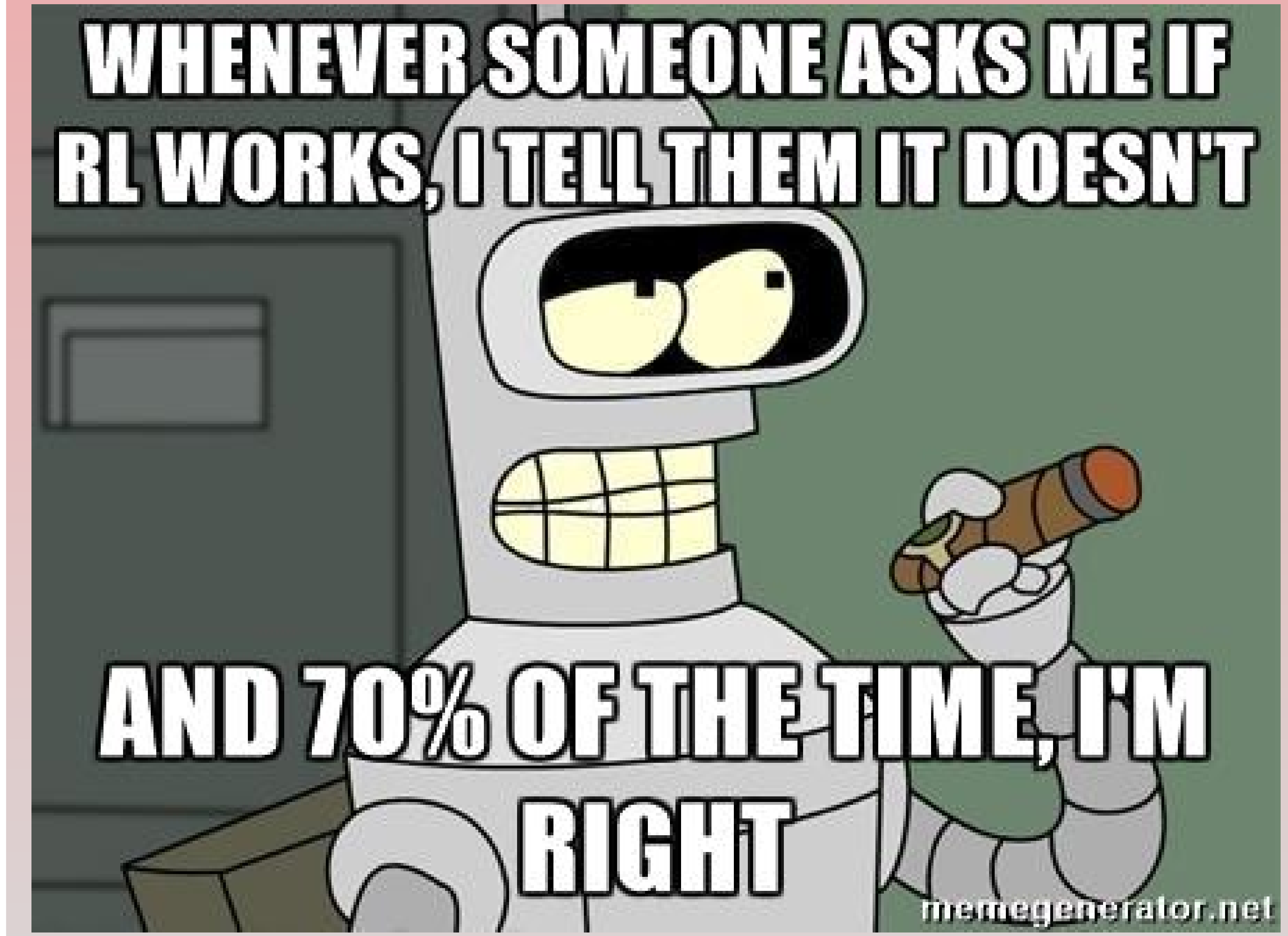
Silver et al. (2017), Arxiv:1712.01815



Mnih et al. (2015), Nature

Success stories and limitations of deep reinforcement learning

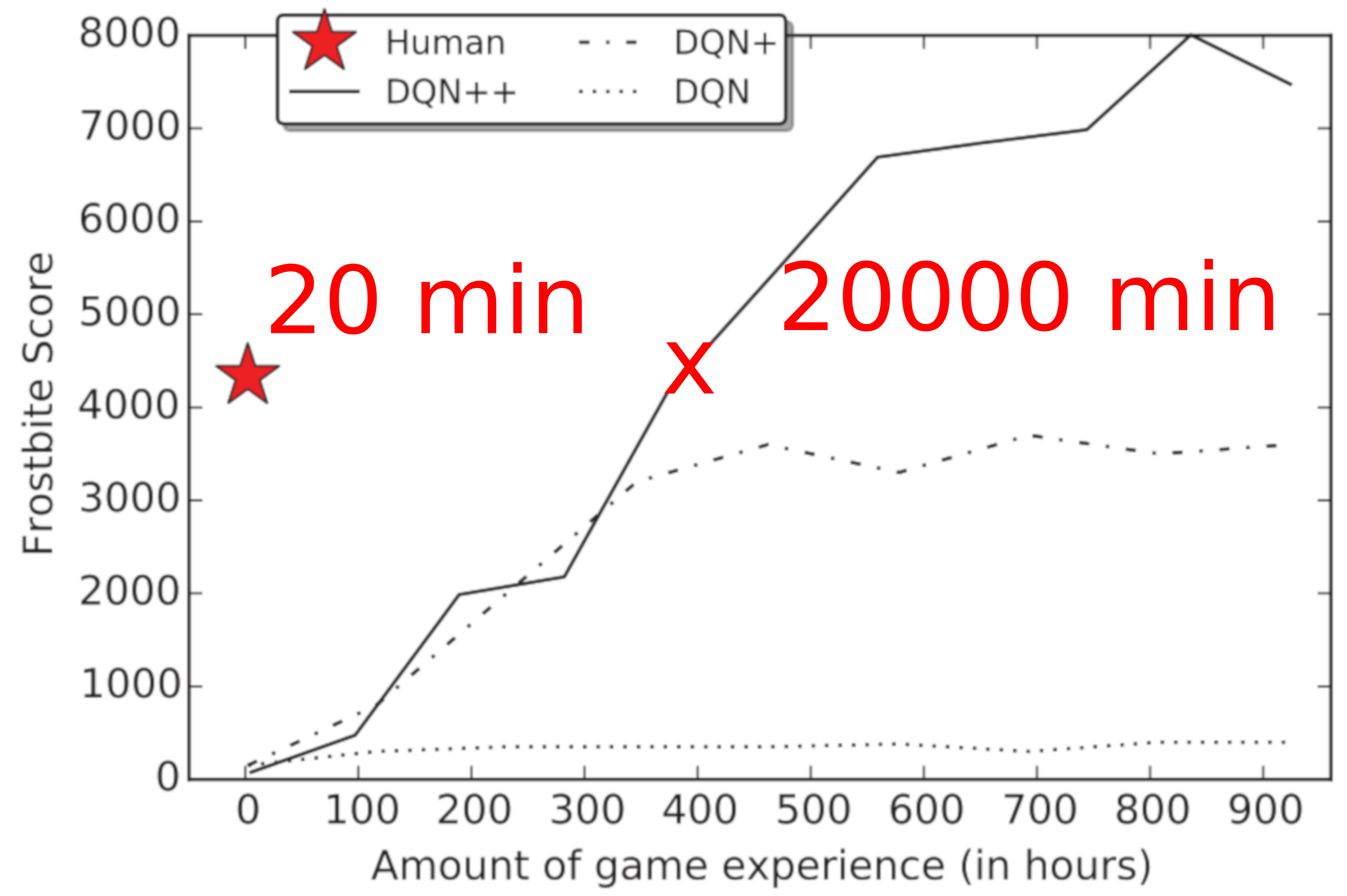
Deep Reinforcement Learning Doesn't Work Yet



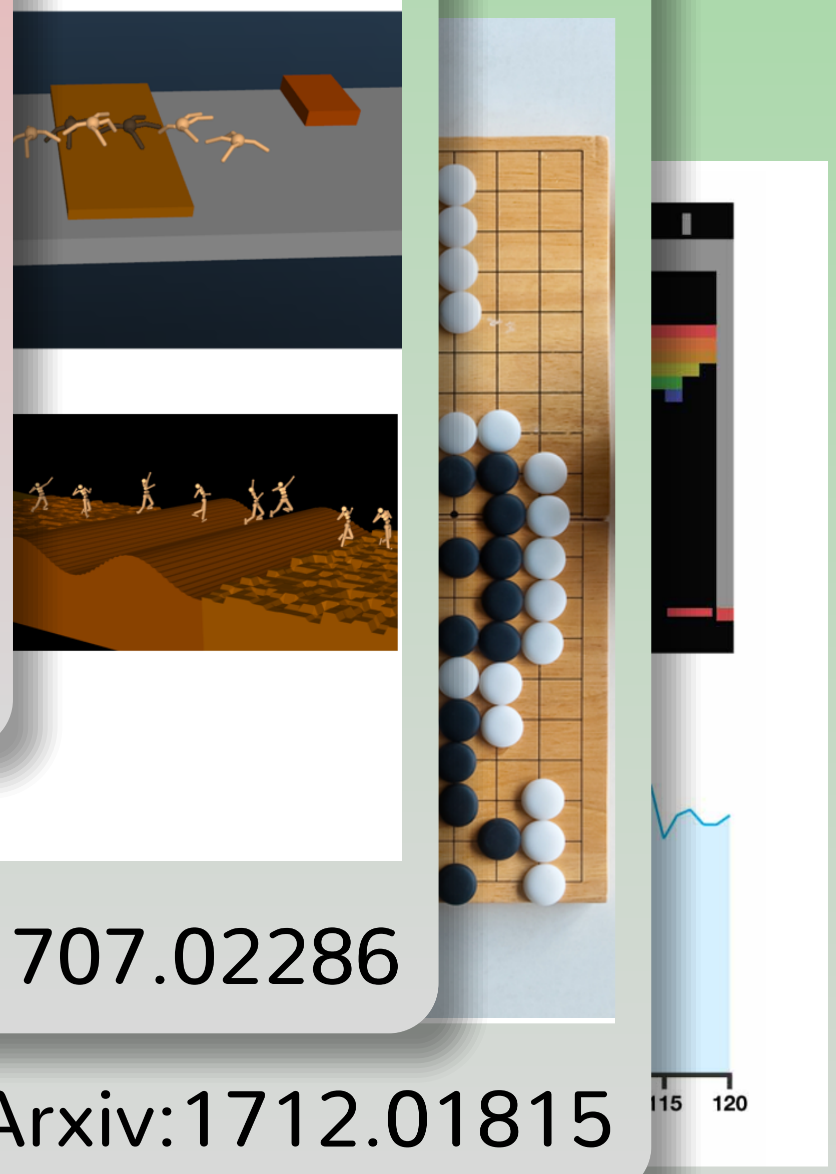
https://www.alexirpan.com/2018/02/14/rl-hard.html

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

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