Deep Reinforcement Learning

Exercise 1

Student: Arseni Pertzovskiy, ID: 317377372.

1. Define and explain the following terms [10 pts]

- Markov Decision Process (MDP). Which problems can be solved with an MDP? Provide an example.

Explanation: Markov Decision Processes are a tool for modeling sequential decision-making problems where a decision maker interacts with the environment in a sequential fashion. One of the basic and most important principals of MDP and Markov chains in general is the Markov Property that sais: <u>the future is independent of the past given the present</u>.
Definition: A Markov Decision Process is a tuple $<S, A, P, R, \gamma>$.
1 finite set of states (S)
2 finite set of actions (A)
3 state transition probability matrix (P)
4 a reward function (R)
5 a discount factor $\gamma \in [0, 1]$

The kind of problems that can be solved with MDP: optimization problems solved via dynamic programming and reinforcement learning. For example: robotics, automatic, control, economics, games and manufacturing. Example from our cource: finding the right policy for a sertain environment.

- State-value function

Definition: The state-value function $v_\pi(s)$ for policy π is the expected return (total discounted future reward) starting from state *s*, and then following policy π.
Explanation: The state-value function describes how good it is to be in a particular state *s*.

$v_\pi(s) = E[Gt|St = s, \pi] =: E\pi[Gt|St = s] = E[Rt+1 + \gamma Rt+2 + \gamma\ 2Rt+3 \ldots |St = s, \pi]$

- Action-value function

Definition: The action-value function $q_\pi(s, a)$ for policy π is the expected return (total discounted future reward) starting from state *s*, taking an (arbitrary) action *a* and then following policy π.
Explanation: The action-value function describes how good it is to take a particular action a (not necessarily from policy π) from a given state *s* and then following policy.

$q_\pi(s, a) = E[Gt|St = s, At = a, \pi] =: E\pi[Gt|St = s, At = a] = E[Rt+1 + \gamma Rt+2 + \gamma\ 2Rt+3 \ldots |St = s, At = a, \pi]$

- Policy

Definition: A policy is a mapping from states to actions. There are two types of policies:
deterministic policy: $a_t = \pi(s_t)$
stochastic policy: $\pi(a|s) = p(At = a|St = s, \pi)$

Explanation: a policy defines the agent's behavior. For the most of the times we are aiming to find an optimal policy that is mapping from states to actions and tries to maximize expected total future rewards. Optimal policy is solution to MDP.

- Dynamic programming

Definition: DP is a method for solving a complex problem by breaking it down into a collection of simpler subproblems, solving each of those subproblems just once, and storing their solutions using a memory-based data structure (array, map,etc).

Explanation: If sub-problems can be nested recursively inside larger problems, so that dynamic programming methods are applicable, then there is a relation between the value of the larger problem and the values of the sub-problems. In the optimization literature (as in our case also – RL) this relationship is called the Bellman equation.

- Value iteration

Explanation: Value iteration is a method of computing an optimal MDP policy and its value. Value iteration starts at the "end" and then works backward, refining an estimate of either Q* or V*.

Definition: At first, the method solves Bellman optimality equation to find the optimal value-function. Then, it derives optimal policy $\pi*$ from the optimal value-function. And this policy is the greedy policy because it greedily selects the best action using the value function v. As it was shown in cource's lecture in short:
Value Iteration: $v*(s) \Rightarrow \pi*(s)$.

- Policy iteration

Explanation: In Policy Iteration you randomly select a policy and find value function corresponding to it , then find a new (improved) policy based on the previous value function, and so on this will lead to optimal policy .

Definition: The policy iteration algorithm is composed out of a policy evaluation and policy improvement step.
Policy evaluation: for a given policy π the value function v_π is estimated.
Policy improvement: from the value function v_π an improved policy is derived following a greedy strategy.
This cycle is repeated until convergence to the optimal policy $\pi*$ and optimal value function $v*$ is obtained. As it was shown in cource's lecture in short:
Policy Iteration: $\pi(s) \rightarrow v_\pi(s) \rightarrow \pi_0 (s) \rightarrow v_{\pi 0} (s) \rightarrow \cdot\cdot\cdot \rightarrow \pi*(s) \rightarrow v*(s)$

- Reinforcement learning (RL). Which problems can be solved with RL? What is the diference to an MDP? Provide an example.

Explanation: RL is an area of machine learning concerned with how software agents ought to take actions in an environment in order to maximize the notion of cumulative reward.

Basic RL is modeled as a MDP that was discribed above. But there are other methods like **Monte-Carlo RL** that are based on averaging sample returns.
Their (Monte-Carlo RL methods) main advantage is that they are model-free. In means, they can have no knowledge of MDP transitions/rewards at all. In many real-life environments that is exactly the case (wether, stock markets, robot movements, games with unknown dinamics and so on). And it allows to approach those kind of problems that were very problematic to classical MDP formulations.

2. Markov decision process (MDP) – FrozenLake

a.) [30 pts] Construct the transition model p (s'|s; a) and reward function r (s) for this MDP. Draw a graphical model (by hand, powerpoint etc) of the MDP for a = N showing all states, transition probabilities and rewards.

Answer: The transition model function and the reward function are inside the code.
Reward function:

-1	-0.04	-0.04	+1
-0.04	-0.04	-0.04	-1
-0.04	-1	-0.04	-1
-0.04	-0.04	-0.04	-0.04

Transition model:
For a = N:

S\S'	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0.8	0.1	0	0	0	0.1	0	0	0	0	0	0	0	0	0	0
3	0	0.8	0.1	0	0	0	0.1	0	0	0	0	0	0	0	0	0
4	0	0	0.8	0.1	0	0	0	0.1	0	0	0	0	0	0	0	0
5	0.1	0	0	0	0.8	0	0	0	0.1	0	0	0	0	0	0	0
6	0	0.1	0	0	0.8	0	0	0	0	0.1	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0.1	0	0	0.8	0	0	0	0	0.1	0	0	0	0
9	0	0	0	0	0.1	0	0	0	0.8	0	0	0	0.1	0	0	0
10	0	0	0	0	0	0.1	0	0	0.8	0	0	0	0	0.1	0	0
11	0	0	0	0	0	0	0.1	0	0	0.8	0	0	0	0	0.1	0
12	0	0	0	0	0	0	0	0.1	0	0	0.8	0	0	0	0	0.1
13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0.1	0	0	0.8	0.1

For a = S:

S\S'	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0	0.1	0.8	0	0	0.1	0	0	0	0	0	0	0	0	0	0
3	0	0	0.1	0.8	0	0	0.1	0	0	0	0	0	0	0	0	0
4	0	0	0	0.9	0	0	0	0.1	0	0	0	0	0	0	0	0
5	0.1	0	0	0	0	0.8	0	0	0.1	0	0	0	0	0	0	0
6	0	0.1	0	0	0	0	0.8	0	0	0.1	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0.1	0	0	0	0.8	0	0	0	0.1	0	0	0	0
9	0	0	0	0	0.1	0	0	0	0	0.8	0	0	0.1	0	0	0
10	0	0	0	0	0	0.1	0	0	0	0	0.8	0	0	0.1	0	0
11	0	0	0	0	0	0	0.1	0	0	0	0	0.8	0	0	0.1	0
12	0	0	0	0	0	0	0	0.1	0	0	0	0.8	0	0	0	0.1
13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0.1	0	0	0	0.9

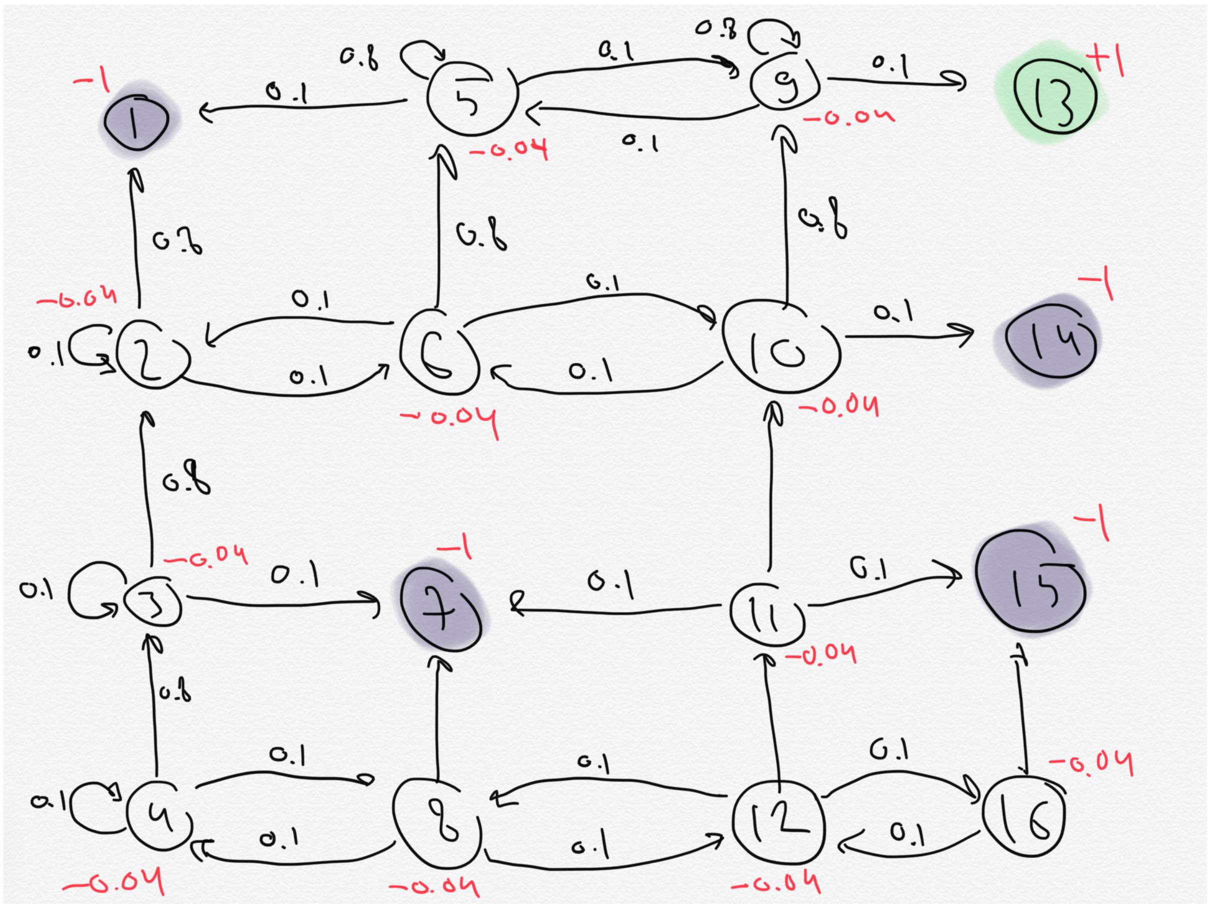
For a = E (->):

S\S’	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0.1	0	0.1	0	0	0.8	0	0	0	0	0	0	0	0	0	0
3	0	0.1	0	0.1	0	0	0.8	0	0	0	0	0	0	0	0	0
4	0	0	0.1	0.1	0	0	0	0.8	0	0	0	0	0	0	0	0
5	0	0	0	0	0.1	0.1	0	0	0.8	0	0	0	0	0	0	0
6	0	0	0	0	0.1	0	0.1	0	0	0.8	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0	0	0	0.1	0.1	0	0	0	0.8	0	0	0	0
9	0	0	0	0	0	0	0	0	0.1	0.1	0	0	0.8	0	0	0
10	0	0	0	0	0	0	0	0	0.1	0	0.1	0	0	0.8	0	0
11	0	0	0	0	0	0	0	0	0	0.1	0	0.1	0	0	0.8	0
12	0	0	0	0	0	0	0	0	0	0	0.1	0.1	0	0	0	0.8
13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.1	0.9

For a = W(<-):

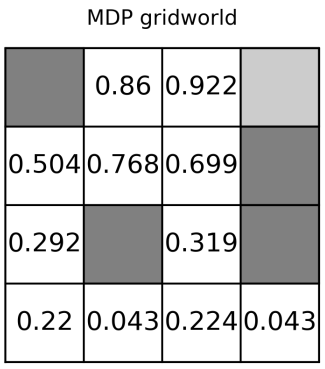
S\S’	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	0.1	0.8	0.1	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0.1	0.8	0.1	0	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0.1	0.9	0	0	0	0	0	0	0	0	0	0	0	0
5	0.8	0	0	0	0.1	0.1	0	0	0	0	0	0	0	0	0	0
6	0	0.8	0	0	0.1	0	0.1	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	0	0	0	0.8	0	0	0.1	0.1	0	0	0	0	0	0	0	0
9	0	0	0	0	0.8	0	0	0	0.1	0.1	0	0	0	0	0	0
10	0	0	0	0	0	0.8	0	0	0.1	0	0.1	0	0	0	0	0
11	0	0	0	0	0	0	0.8	0	0	0.1	0	0.1	0	0	0	0
12	0	0	0	0	0	0	0	0.8	0	0	0.1	0.1	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0.8	0	0	0.1	0.1

A graphical model of the MDP for a = N:

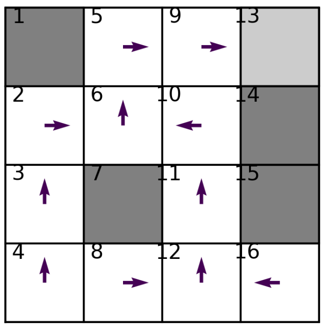


b.)[10 pts] Solve the MPD using **value iteration** with γ = 1 and a termination threshold of θ = 10^{-4}. Plot the optimal values into the gridworld by using the plot value of the World class. Plot the optimal policy in a diferent figure by using the plot policy function of the World class.

Answer:


MDP gridworld

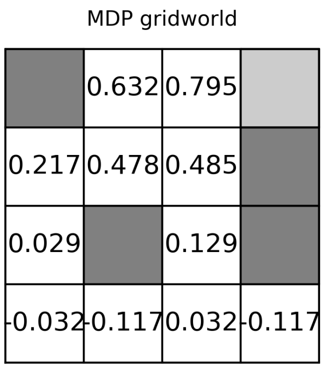
▨	0.86	0.922	▨
0.504	0.768	0.699	▨
0.292	▨	0.319	▨
0.22	0.043	0.224	0.043

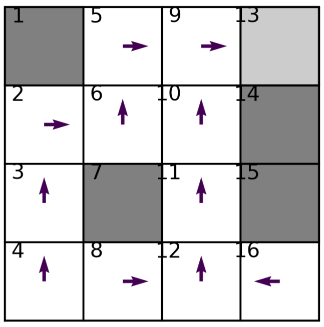


1 ▨	5 →	9 →	13 ▨
2 →	6 ↑	10 ←	14 ▨
3 ↑	7 ▨	11 ↑	15 ▨
4 ↑	8 →	12 ↑	16 ←

c.)[10 pts] Repeat your **value iteration** algorithm with a reduced discount factor γ = 0.9 and r = -0.04. Explain how these changes in parameters affect the optimal policy. Generate two plots showing the optimal value function and optimal policy.

Answer:

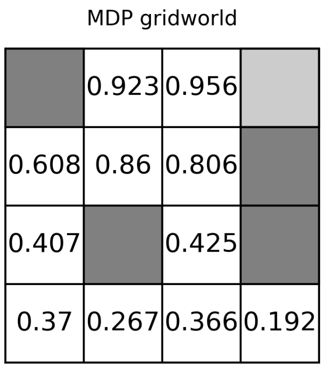


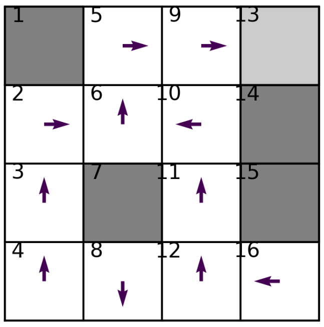


Explanation: The only diference in policies is inside cell number 10. In that cell we want to get away from "bad" cell (cell 4) and then reach the goal while γ = 1, but it changes with γ = 0.9. In the later case we less care about escaping the "bad" cells and we want to reach our goal as soon as possible. This happends because with γ = 0.9 long paths cost us more (imidiate rewars matter more) than in case of γ = 1.

d.)[10 pts] Repeat your **value iteration** algorithm with a reduced discount factor γ = 1 and reward r = -0.02. Explain how these changes in parameters affect the optimal policy. Generate two plots showing the optimal value function and optimal policy.

Answer:





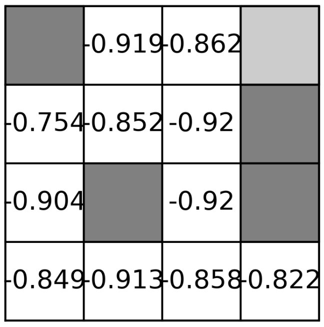
Explanation: While r = -0.02 we care less about length of our path to the goal we prefer to avoid "bad" cells (even more than in case of 2(b) section) in order to increase our chances to success. In cell 8 our action is S – avoiding totally the theat from "bad" 7-th cell.

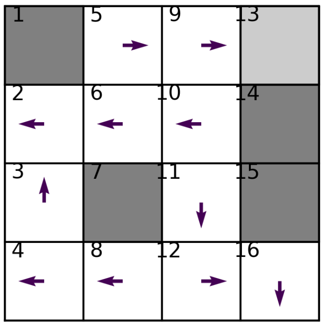
e.)[30 pts] Solve the MPD using **policy iteration** with γ = 0.9 and r = -0.04. Initialize your policy iteration algorithm with a **uniform random policy**. Plot the value function and policy **after each iteration step** into two different figures of the gridworld by using the plot value and plot policy function of the World class, respectively. Compare your results with the results obtained using value iteration.

Answer:
Explanation: Final result is identical in both cases (Value Iteration and Policy Iteration). At the beginning, the policy is trying to escape from threats more than to achive the goal, but at the next steps the policy learns how to get to the goal in an optimal way.

Iteration 1:

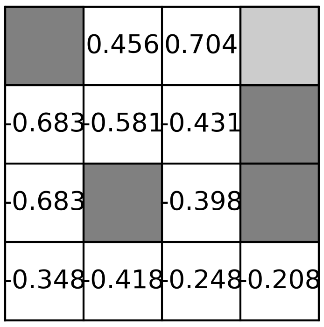
MDP gridworld

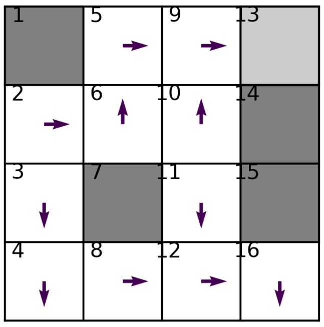




Iteration 2:

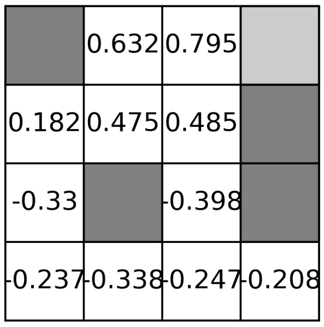
MDP gridworld

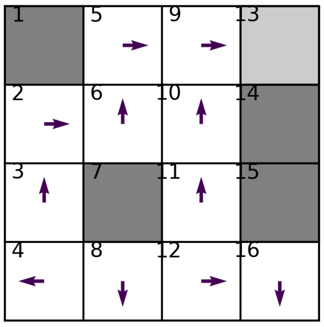




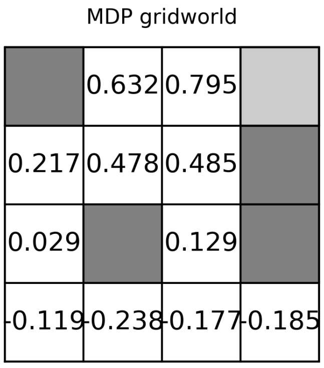
Iteration 3:

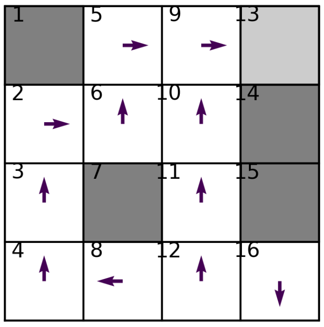
MDP gridworld



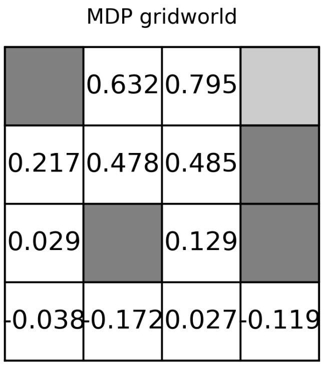


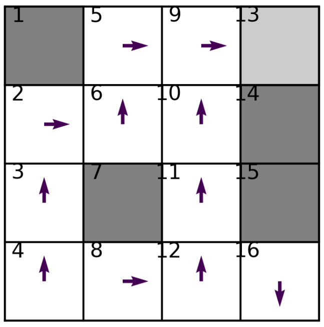
Iteration 4:



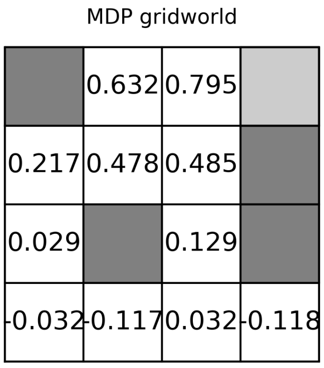


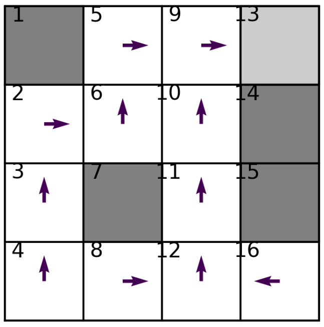
Iteration 5:



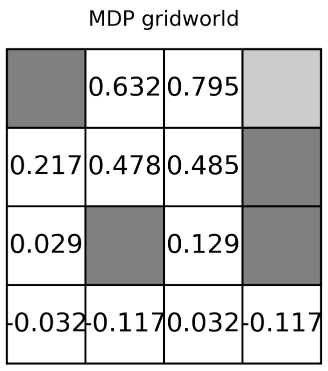


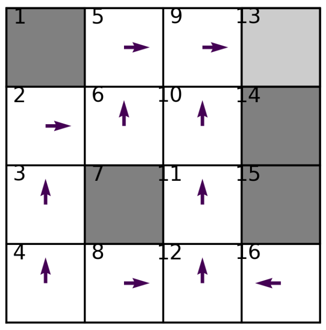
Iteration 6:





Iteration 7 (final):


MDP gridworld
0.632
0.795
0.217
0.478
0.485
0.029
0.129
-0.032
-0.117
0.032
-0.117


1
5
9
13
2
6
10
14
3
7
11
15
4
8
12
16