Ben-Gurion University of the Negev

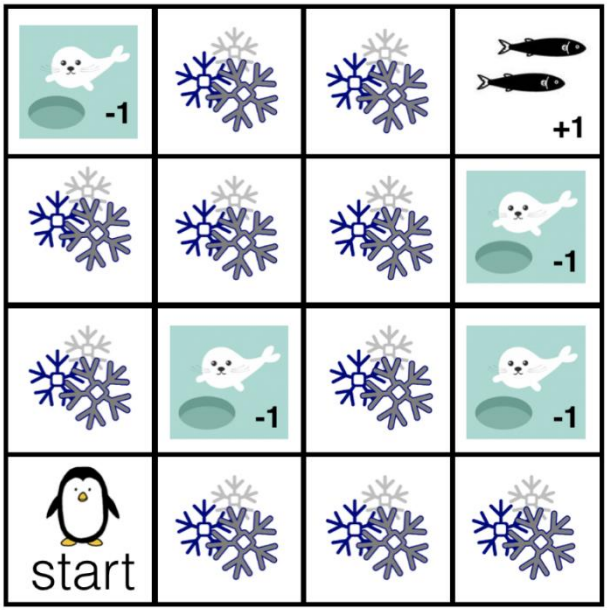
Deep Reinforcement Learning

Markov Decision Process - Frozen Lake

May 2020

Lecturer: Armin Biess, Ph.D.

MSc Student: Tom Landman



1. Introduction

This report composed of two main sections: Theory and Practice. In the first section, some key concepts regarding Reinforcement Learning in general and Markov Decision Process (MDP), in particular, is defined, explained, and referred from Sutton and Barto Book of Reinforcement Learning [1]. In the second section, the results of the given FrozenLake problem are displayed, compared, and discussed following the implementation of the given MDP's environment and the required algorithms presented in [1].

2. Theory

This section presents important key concepts relates to the theory of *Reinforcement Learning* (RL). Most of the examples and descriptions regarding these concepts are referenced from Sutton and Barto [1].

2.1 **Markov Decision Process (MDP)**

In general Markov Decision Process is a time control modeling process which can be used as a *reinforcement learning* (RL) task when the Markov property is satisfied. A Markov property refers to cases where the conditional probability distribution of future states of the process depends only upon the present state, not on the sequence of events that preceded it. As presented in Figure 1 below, in the case of RL, an agent needs to learn a certain task based on its interaction with its environment. At each time step $t \in \{0, 1, \dots\}$, the agent is located in a state $S_t \in S$ and then needs to decide which action $A_t \in A(s)$ to take. Based on the agent's current state S_t and a chosen action A_t it would be transformed into a new State S_{t+1} denoted also s' followed by a new Reward R_{t+1} that is used to transparent the feedback from the environment. Therefore, in some cases, an MDP can be used to model such a scenario.

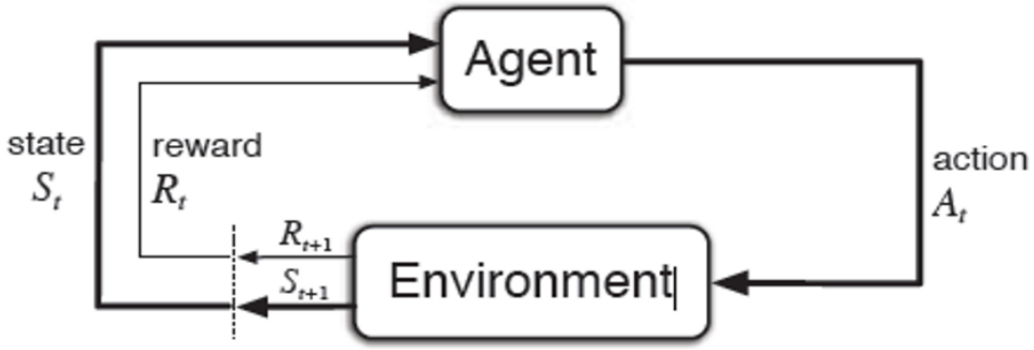


Figure 1 - The agent–environment interaction in a Markov decision process (Sutton and Barto, 2018)

An MDP can be both deterministic or stochastic and is denoted as Finite MDP, in the case of finite-state and action spaces and defined by its state and action sets and by a single step dynamics of the environment. Moreover, given a finite MDP dynamics of state s and action a the probability from of each possible pair of next state s' and reward r is denoted by equation (1):

$$p(s', r|s, a) = Pr\{S_{t+1} = s', R_{t+1} \doteq r|S_t = s, A_t = a\} \quad (1)$$

Given the abovementioned dynamics the expected reward for state-action pairs can be calculated by the following equation (2):

$$r(s, a) \doteq \mathbb{E}[R_{t+1}|S_t = s, A_t = a] = \sum_{r \in \mathcal{R}} r \sum_{s' \in \mathcal{S}} p(s', r|s, a) \quad (2)$$

Note that the probabilities of transitioning from a certain state s and a given action a to a new state s' are denoted as the *state-transition probabilities*, and can be calculated by the following equation (3):

$$p(s'|s, a) \doteq Pr\{S_{t+1} = s'|S_t = s, A_t = a\} = \sum_{r \in \mathcal{R}} p(s', r|s, a) \quad (3)$$

2.2 **State-value Function**

The value function of a state s under a given policy π, is defined as the expected return when starting in s and following π thenceforth, and denoted as $v_\pi(s)$. In the case of MDPs, $v_\pi(s)$ defined by the following equation (4):

$$v_\pi(s) \doteq \mathbb{E}_\pi[G_t|S_t = s] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \middle| S_t = s\right] \quad (4)$$

$\mathbb{E}_\pi[\cdot]$ denotes the expected value of a random variable given that the agent follows policy π, and t is any time step. Note that the value of the terminal state, if any, is always zero.

2.3 **Action-value Function**

The value of taking action a in state s under a given policy π, is defined as the expected return starting from s, taking the action a and thenceforth following policy π, and denoted as $q_\pi(s, a)$. q_π is the action-value function for policy π, and it is calculated by the following equation (5):

$$q_\pi(s, a) \doteq \mathbb{E}_\pi[G_t|S_t = s, A_t = a] = \mathbb{E}_\pi\left[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \middle| S_t = s, A_t = a\right] \quad (5)$$

2.4 **Policy**

In psychology, a *policy* corresponds to a set of stimulus-response rules or associations. In Reinforcement Learning, however, a policy is a mapping from perceived states of the environment to actions to be taken when in those states in which defines the learning agent's way of behaving at a given time. The policy is the core of a reinforcement learning agent in the sense that it is sufficiently able to determine its behavior by itself. Note that in some cases the policy may be a simple function or lookup table, whereas in others it may involve extensive computation such as a search process. In general, policies may be stochastic.

2.5 **Dynamic Programming**

Dynamic Programming (DP) is a mathematical optimization method as well as a computer programming method. This method was developed by Richard Bellman during the 1950s [2] and has found applications in numerous fields, from aerospace engineering to economics. Based on the DP paradigm, a problem can be divided up into a series of overlapping subproblems followed by sub solutions that combined to form a solution to a large subproblem. In order to address the problem of designing a controller to minimize a measure of a dynamical system's behavior over time, denoted as "Optimal Control", Bellman and others suggested the DP approach which uses the concepts of a dynamical system's state and a value function, or "optimal return function," to define a functional equation, now often called the Bellman equation.

DP is widely considered the only feasible way of solving general stochastic optimal control problems. Note that its computational requirements grow exponentially with the number of state variables ("the curse of dimensionality"), but it is still far more efficient and more widely applicable than any other general method. In addition, DP can be used to solve MDPs and extensions to partially observable MDPs such that a given optimal policy can be found in a stepwise manner because of its recursive equation (6) appears below in which the function for each state in step n, can be calculated based on the next state and best action found in the next step :

$$f_n^* = \max_a F_n\,(s, a, f_{n+1}^*(a)) \quad (6)$$

The key idea of DP, and of reinforcement learning generally, is the use of value functions to organize and structure the search for good policies.

2.6 **Value Iteration**

Value Iteration algorithm is used to find the optimal value-function $v_*(s)$ and obtained by defining the Bellman optimality equation (7) as an update rule for all states except terminated states ($s \in \mathcal{S}$).

$$v_{k+1}(s) \doteq \max_a \mathbb{E}[R_{t+1} + \gamma v_k(S_{t+1}) \mid S_t = s\,,\ A_t = a] = \max_a \sum_{s',r} p(s', r|s, a)[r + \gamma v_k(s')] \quad (7)$$

Note that for arbitrary v_0, the sequence $\{v_k\}$ can converge to v_* under the same conditions that guarantee the existence of v_*. The value iteration update is similar to the policy evaluation update (equation (10)) except that it requires the maximum to be taken overall actions. Further, the value iteration algorithm requires an infinite number

of iterations to converge exactly to v_*, however, in practice, in order to terminate, the algorithm is stopped once the value function changes by only a small amount in a sweep. As can be seen in the algorithm presented in Figure 2, first, all states values are initialed arbitrarily, usually with zeros including terminal states ($s \in \mathcal{S}^+$). Then, the inner loop is iterating through all of the states except terminated states ($s \in \mathcal{S}$), and update the state's value function by taking the maximum value by a given action a and calculates it using the update rule presented above. Note, that for each state value function $v(s)$ (except terminal states), each transition probability by the given current state s and action a to a new state s' is being considered and multiplied by the reward r of the next state s' in addition to the next state's value function $v(s')$ which multiplied by a discount factor γ. Therefore, cases of transition probability of zeros represent states that are not achievable from a certain state and therefore will not take into consideration.

Value Iteration, for estimating $\pi \approx \pi_*$

Algorithm parameter: a small threshold $\theta > 0$ determining accuracy of estimation
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(terminal) = 0$

Loop:
| $\Delta \leftarrow 0$
| Loop for each $s \in \mathcal{S}$:
| $v \leftarrow V(s)$
| $V(s) \leftarrow \max_a \sum_{s',r} p(s', r \mid s, a)\left[r + \gamma V(s')\right]$
| $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$

Output a deterministic policy, $\pi \approx \pi_*$, such that
$\pi(s) = \operatorname{argmax}_a \sum_{s',r} p(s', r \mid s, a)\left[r + \gamma V(s')\right]$

Figure 2 - Value Iteration Algorithm (Sutton and Barto, 2018)

2.7 **Policy Iteration**

As part of the reinforcement learning task, Dynamic Programming is used to estimate the value functions to organize and structure the search for good policies. Therefore, the optimal policies can be obtained once we have found the optimal value functions, v_* or q_*, which satisfy the Bellman optimality equations (8)(9) for all states except terminated states ($s \in \mathcal{S}$) and actions ($a \in \mathcal{A}(s)$), and next states including terminal states ($s' \in \mathcal{S}^+$):

$$v_*(s) \doteq \max_a \mathbb{E}[R_{t+1} + \gamma v_*(S_{t+1}) \mid S_t = s\,,\ A_t = a] = \max_a \sum_{s',r} p(s', r|s, a)[r + \gamma v_*(s')] \quad (8)$$

$$q_*(s, a) \doteq \max_a \mathbb{E}\left[R_{t+1} + \gamma \max_{a'} q_*(S_{t+1}, a') \,\middle|\, S_t = s\,,\ A_t = a\right] = \sum_{s',r} p(s', r|s, a)\left[r + \gamma \max_{a'} q_*(s', a')\right] \quad (9)$$

Since DP algorithms are obtained by turning Bellman equations such as these to update rules for improving approximations of the desired value functions, once a policy, π, has been improved using v_π to yield a better policy, π', we can then compute $v_{\pi'}$ and improve it again to yield an even better π''. As a result, a sequence of monotonically improving policies and value functions can be obtained:

$$\pi_0 \xrightarrow{E} v_{\pi_0} \xrightarrow{I} \pi_1 \xrightarrow{E} v_{\pi_1} \xrightarrow{I} \pi_2 \dots \xrightarrow{I} \pi_* \xrightarrow{E} v_*$$

Where $\xrightarrow{E}$ denotes a *policy evaluation* step and $\xrightarrow{I}$ denotes a *policy improvement* step.

The *Policy Iteration* algorithm seeks to find the optimal policy using continuous iterations of both policy evaluation and policy improvement steps, where each policy is guaranteed to be a strict improvement over the previous one until reaching the optimal policy. Therefore, in the case of a finite MDP, which has a finite number of policies, the policy iteration process must converge to an optimal policy and optimal value function after a finite number of iterations. This way of finding an optimal policy is called policy iteration.

As can be seen in the algorithm presented in Figure 3, each policy evaluation, itself an iterative computation, is started with the value function of the previous policy, therefore, results in a great increase in the policy evaluation step speed of convergence.

Policy Iteration (using iterative policy evaluation) for estimating $\pi \approx \pi_*$

1. Initialization
   $V(s) \in \mathbb{R}$ and $\pi(s) \in \mathcal{A}(s)$ arbitrarily for all $s \in \mathcal{S}$

2. Policy Evaluation
   Loop:
      $\Delta \leftarrow 0$
      Loop for each $s \in \mathcal{S}$:
         $v \leftarrow V(s)$
         $V(s) \leftarrow \sum_{s',r} p(s', r \mid s, \pi(s))\left[r + \gamma V(s')\right]$
         $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
   until $\Delta < \theta$ (a small positive number determining the accuracy of estimation)

3. Policy Improvement
   policy-stable $\leftarrow$ *true*
   For each $s \in \mathcal{S}$:
      old-action $\leftarrow \pi(s)$
      $\pi(s) \leftarrow \arg\max_a \sum_{s',r} p(s', r \mid s, a)\left[r + \gamma V(s')\right]$
      If *old-action* $\neq \pi(s)$, then *policy-stable* $\leftarrow$ *false*
   If *policy-stable*, then stop and return $V \approx v_*$ and $\pi \approx \pi_*$; else go to 2

Figure 3 - Policy Iteration Algorithm (Sutton and Barto, 2018)

2.7.1 Policy Evaluation

This step which is usually referred to as a *prediction problem* is called *policy evaluation* since we need to consider how to compute the state-value function v_π for an arbitrary policy π. If the environment's dynamics are completely known, then an iterative solution is suitable to address the task of evaluating a certain policy based on a sequence of mapped states ($s \in \mathcal{S}^+$) using approximated value functions $v_0, v_1, v_2, \dots \in \mathbb{R}$. Also, note that the initial approximation, v_0, is chosen arbitrarily beside terminal states in which needs to be initialized with zeros. Further, note that each successive approximation is obtained by using the Bellman equation for v_π (10) as an update rule for all states beside terminal states ($s \in \mathcal{S}$).:

$$v_{k+1}(s) \doteq \mathbb{E}_\pi[R_{t+1} + \gamma v_k(S_{t+1}) \mid S_t = s\,] = \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)\,[r + \gamma v_k(s')] \quad (10)$$

Further, note that $v_k = v_\pi$ is a fixed point for this update rule because the Bellman equation for v_π assures equality in this case. Furthermore, as can be seen in the algorithm described in Figure 4, in order to produce a successive approximation, the expected update to v_{k+1} from v_k, iterative policy evaluation applies the same operation to each state s and replaces the previous value of s with a new value. The new value is obtained from the previous values of the successor states of s, and the expected immediate rewards, including all single-step transitions possible under the given policy which is evaluated. Note that for each new value function approximation, the values of every state are being updated once during each iteration of the policy evaluation.

Iterative Policy Evaluation, for estimating $V \approx v_\pi$

Input π, the policy to be evaluated
Algorithm parameter: a small threshold $\theta > 0$ determining accuracy of estimation
Initialize $V(s)$, for all $s \in \mathcal{S}^+$, arbitrarily except that $V(terminal) = 0$

Loop:
  $\Delta \leftarrow 0$
  Loop for each $s \in \mathcal{S}$:
    $v \leftarrow V(s)$
    $V(s) \leftarrow \sum_a \pi(a|s) \sum_{s',r} p(s',r|s,a)\big[r + \gamma V(s')\big]$
    $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
until $\Delta < \theta$

Figure 4 - Policy Evaluation Algorithm (Sutton and Barto, 2018)

2.7.2 Policy Improvement

In the policy evaluation step, we described how given a policy and its value function, we can evaluate the change in the policy at a single state to a particular action. Therefore, it can be used to evaluate certain changes at all states and to all possible actions, in order to select at each state the best action greedily according to $q_*(s,a)$. It can be calculated using the following equation (11):

$$\pi'(s) \doteq \underset{a}{argmax}\, q_\pi(s,a) = \underset{a}{argmax}\, \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1}) \mid S_t = s\,,\, A_t = a]$$
$$= \underset{a}{argmax} \sum_{s',r} p(s',r|s,a)[r + \gamma\, v_\pi(s')] \quad (11)$$

Note that the $argmax_a$ denotes the value of a at which the expression that follows is maximized, meaning that the greedy policy chooses the best action in the short term, after one step of lookahead to v_π. Therefore, this step is called *policy improvement*, and as can be seen in Figure 5 we want to improve a given policy π as part of the process of converging to the optimal policy π^*. Given a new greedy policy, π', which is good as the old policy π, then $v_\pi = v_{\pi'}$, and from (12) it follows that for all $s \in \mathcal{S}$:

$$v_{\pi'}(s) \doteq \max_a \mathbb{E}[R_{t+1} + \gamma v_{\pi'}(S_{t+1}) \mid S_t = s\,,\, A_t = a] = \max_a \sum_{s',r} p(s',r|s,a)[r + \gamma\, v_{\pi'}(s')] \quad (12)$$

Note that this equation is similar to the Bellman optimality equation, and therefore, $v_{\pi'}$ must be v_* and both π and π' must be optimal policies. Therefore, policy improvement will result in a better policy except when the original policy optimality is satisfied.

Algorithm: Policy Improvement

Input: MDP, V
Output: policy π'
  for $s \in \mathcal{S}$ **do**
    for $a \in \mathcal{A}(s)$ **do**
      $Q(s,a) \leftarrow \sum_{s',r \in \mathcal{R}} p(s',r|s,a)[r + \gamma V(s')]$
    end
    $\pi'(s) \leftarrow \max_{a \in \mathcal{A}(s)} Q(s,a)$
  end

Figure 5 - Policy Improvement Algorithm (DRL Course Lectures, 2020)

2.8 **Reinforcement Learning (RL)**

Reinforcement Learning (RL) is used to learn a mapping between situations to actions where the goal is to maximize a numerical reward signal for the task of learning how to act. The learner is not told which actions to be taken, however, it must discover which actions yield the most reward by trial and error. Also, note that in challenging cases, actions may affect not only the immediate reward but also the next situation and, through that, all subsequent rewards. Therefore, trial and error and delayed reward, considered to be the two most important distinguishing features of reinforcement learning.

RL's main idea is to capture the important aspects of the real problem facing a learning agent while interacting with its environment over time to achieve a certain goal. The learning agent must be able to sense the environment's states and must be able to take actions that affect the state. Any method that is well suited to solve such problems we consider to be a reinforcement learning method. Unlike supervised learning in which rely on labeled examples of the correct behavior or unsupervised learning which does not rely on labeled examples, reinforcement learning is trying to maximize a reward signal instead of trying to find a hidden structure and therefore, considered as a third machine learning paradigm.

The trade-off between exploration and exploitation considered a challenge that arises only in the case of RL. The agent has to exploit what it has already experienced to obtain a reward, but it also has to explore to make better action selections in the future. The agent must try a variety of actions and progressively favor those that appear to be best. On a stochastic task, each action must be tried many times to gain a reliable estimate of its expected reward. In contrast to many approaches in which subproblems are considered without addressing how they might fit and form a larger view, RL considers the whole problem of a goal-directed agent interacting with an uncertain environment. Note that all RL-based agents have explicit goals, and can sense aspects of their environments, and choose actions to influence their environments

The problem of RL is usually formalized using dynamical systems theory, specifically, as the optimal control of incompletely, referred to as the Markov Decision Process (MDP). MDP is intended to include the aspects of sensation, action, and goal in their simplest possible forms without trivializing any of them. Note that in order to solve an RL task using MDP, transition probabilities and rewards are required, unlike in model-free-based approaches in which the underlying model and the reward function are unknown or partially known, thus, no a-prior knowledge can be used but, can be learned using the agent's experience.

There are many examples of problems that can be solved using an RL approach, e.g.:

- A mobile robot decides whether it should enter a new room in search of more trash to collect or start trying to find its way back to its battery recharging station. It makes its decision based on the current charge level of its battery and how quickly and easily it has been able to find the recharger in the past.

- A master chess player makes a move. The choice is informed both by planning anticipating possible replies and counterreplies and by immediate, intuitive judgments of the desirability of particular positions and moves.

3. Markov Decision Process - *FrozenLake*

This section describes the implementation of the famous FrozenLake problem where given a penguin on a frozen lake, which is described by a 4x4 grid world with holes and a goal state (fish), both defining terminal states. For transitions, to terminal states, the penguin gets a reward of +1 for the goal state and a reward of -1 for the holes, whereas for all other transitions the penguin gets a reward of $r = 0.04$. The penguin can take four possible actions = $\{N, E, S, W\}$, but the surface is slippery and only with probability 0.8 the penguin will go into the intended direction and with probability 0.1 to the left and 0.1 to the right of it. It is assumed that the boundaries are reflective, i.e., if the penguin hits the edge of the lake it remains in the same grid location. Therefore, the task is to find the optimal policy for the penguin to get to the fish by solving the corresponding MDP. Further, note that in the first subsection, a model of the transition model for the action North is presented, and afterward the results of value iteration and policy iteration algorithm for different configurations are presented and discussed.

a. MDP Transition Model (a=N)

In order to construct the given MDP transition model, the *buildMDPTransitionMatrix* function was developed using the problem settings and for a given probability. Although, the transition model has 3 dimensions (s, s', a), in this subsection, a transition model for action North is being displayed in Figure 7. Note that nodes represent the possible states whereas the arrows represent the transition from a state to another. Also, note that the transition probabilities are located on the arrows as well as the reward derived from the reward of the new state to be reached. Further, the full transition matrix can be seen in the appendix. Moreover, for the given FrozenLake settings each reward for a given state s, action a, and next state s', denoted as $r(s, a, s')$ is equal to the reward of the state to be transitioned to. Therefore, in the case of transitioning to the state goal (13) the reward will be $+1$ and -1 for the holes (1, 7, 14, 15). The rest of the states' rewards depends on each experiment configuration wherein some it is equal to -0.04 and -0.02 in others. As a result, before calculating the multiplication between the transition probabilities matrix and the reward vector, rewards are deterministic and initialized by the following vector:
$[-1, -0.04, -0.04, -0.04, -0.04, -0.04, -1, -0.04, -0.04, -0.04, -0.04, -0.04, +1, -1, -1, -0.04]$.
However, by multiplying the transition probabilities for each action with the initial reward vector the function $r(s, a, s')$ can be calculated using the following equation (13):

$$r(s, a, s') = \mathbb{E}[R_t = r | S_{t-1} = s, A_{t-1} = a, S_t = s'] = \sum_{r \in \mathcal{R}} rp(r|s, a, s') \quad (13)$$

Therefore, by using equation (13) the following reward function presented in Figure 6 below.

	N	E	S	W
1	0.000	0.000	0.000	0.000
2	-0.808	-0.136	-0.040	-0.136
3	-0.136	-0.808	-0.136	-0.040
4	-0.040	-0.040	-0.040	-0.040
5	-0.136	-0.040	-0.136	-0.808
6	-0.040	-0.136	-0.808	-0.136
7	0.000	0.000	0.000	0.000
8	-0.808	-0.136	-0.040	-0.136
9	0.064	0.792	0.064	-0.040
10	-0.136	-0.808	-0.136	-0.040
11	-0.232	-0.808	-0.232	-0.808
12	-0.040	-0.040	-0.040	-0.040
13	0.000	0.000	0.000	0.000
14	0.000	0.000	0.000	0.000
15	0.000	0.000	0.000	0.000
16	-0.808	-0.136	-0.040	-0.136

Figure 6 - Reward function
$r(s, a, s')$ calculation for $r = -0.04$

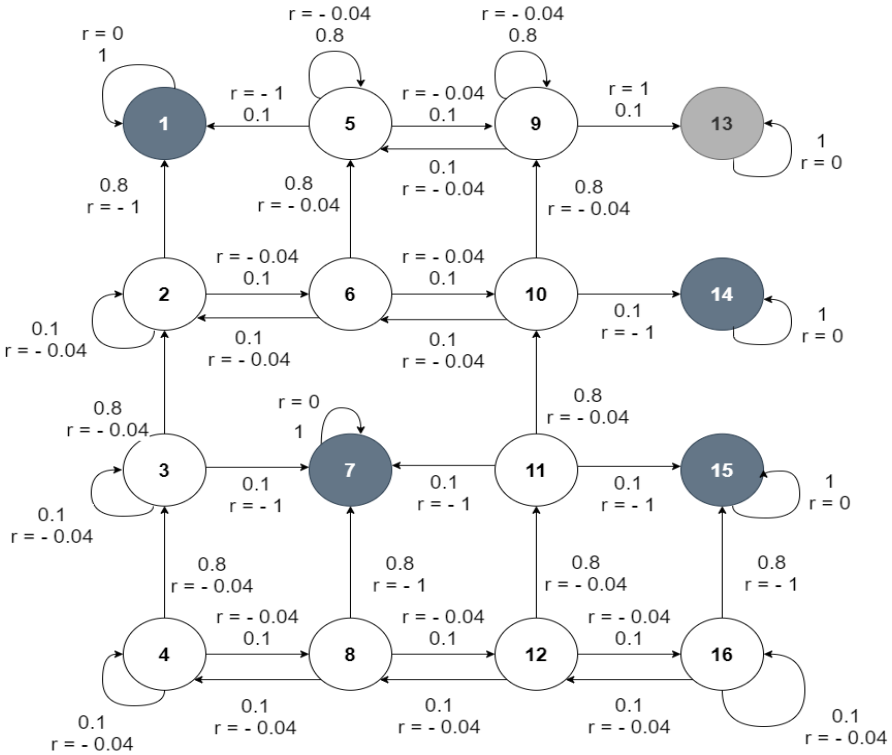


Figure 7 - MDP Transition Model (a = N)

b. *Value iteration:* $\gamma = 1, \theta = 10^{-4}, r = -0.04$:

As can be seen in Table 1, in this experiment, it seems that the penguin found a reasonable policy to reach its goal. Also, by viewing the achieved values sequences, we can notice that the probabilities within the sequence of the north path are higher than the path to the east, therefore, it can be interpreted as less risky than the east path which has more obstacles around. However, note that by taking the east path, when the penguin arrives to state 10, the penguin prefers to turn west instead of north which is longer than the optimum.

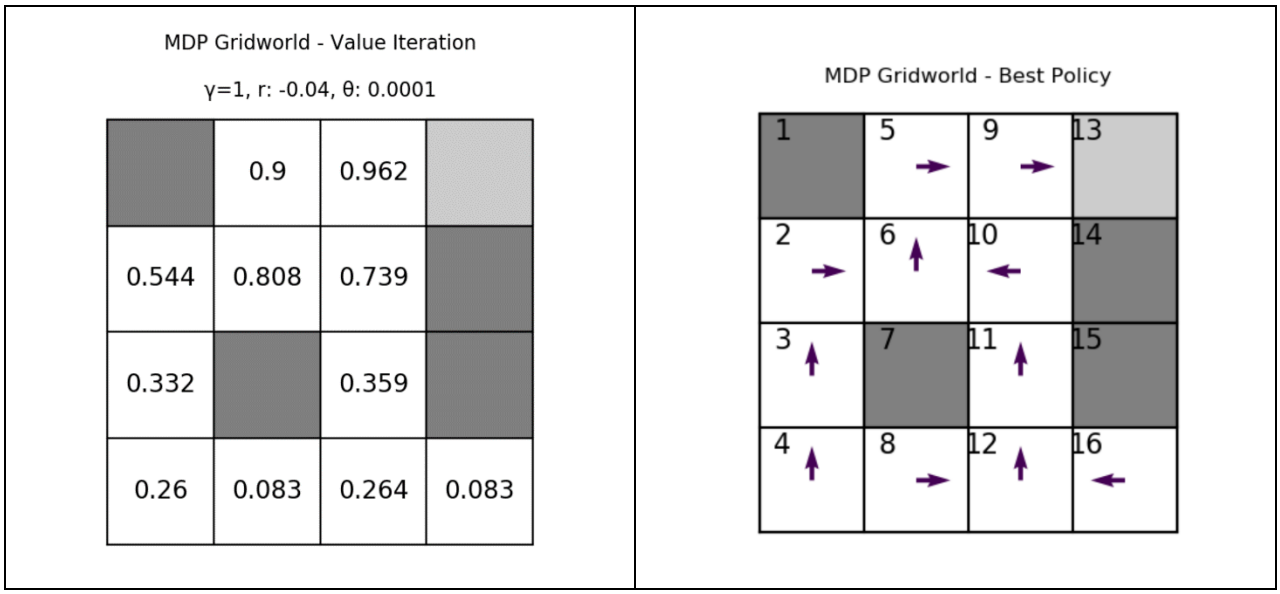


Table 1 - Value Iteration results (b)

c. *Value iteration:* $\gamma = 0.9\ , \theta = 10^{-4}\ ,\ r = -0.04$:

As can be seen in Table 2, the results of this experiment retrieve a better policy comparing to the above configuration which is also the optimal policy. In this scenario, besides reaching its goal, the penguin found a shorter way to the goal when he decides to take the path to the east and when it arrives to state 10, he continues north unlike in the previous experiment when he turned to the west. Further, by viewing the achieved values sequences, we can notice that the probabilities within both sequences are lower but the risk ratio between the two possible paths is also distributed and favors the north path above the east. Note that this experiment's configuration resulted in the best policy and outperformed the rest of the experiments, that is, by lowering the discount factor to 0.9 and keeping the reward as -0.04.

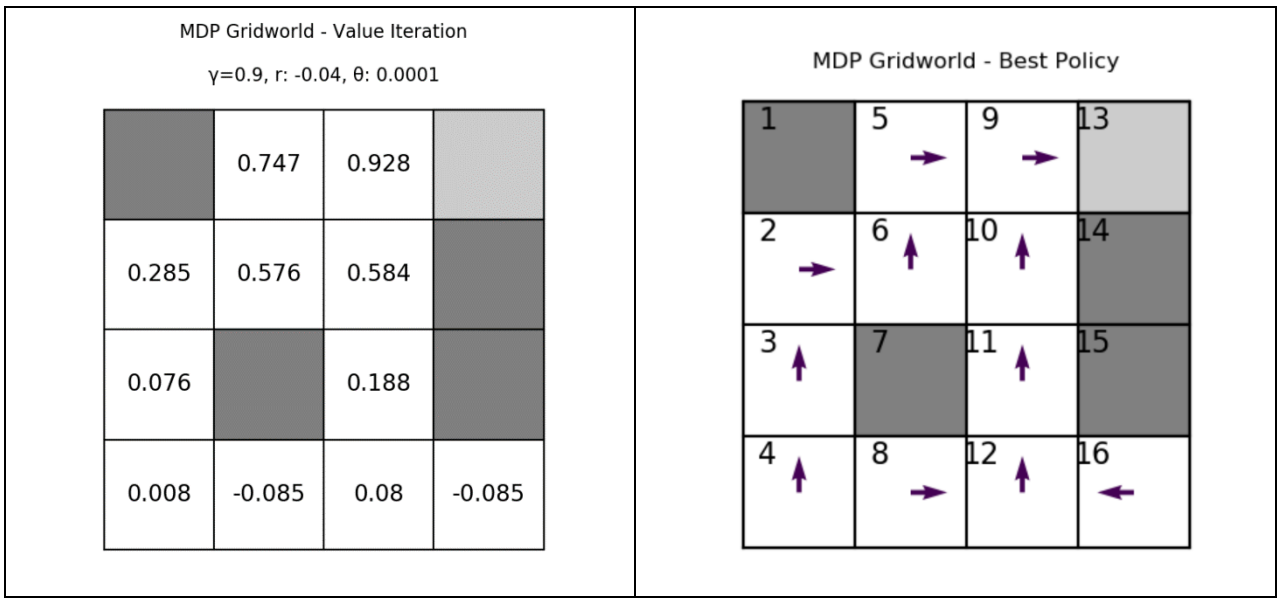


Table 2 - Value Iteration results (c)

d. Value iteration: $\gamma = 1$, $\theta = 10^{-4}$, $r = -0.02$:

As can be seen in Table 3, the results of this experiment retrieved the worst policy among the rest of the experiments. Since the discount factor is similar to that of the first experiments, the only change in the configuration was increasing the reward to -0.02. As a result, the policy of the first experiments is identical except for the chosen action in state 8 which in this case, directs the penguin to go south, an action that causes it to stay in its place.

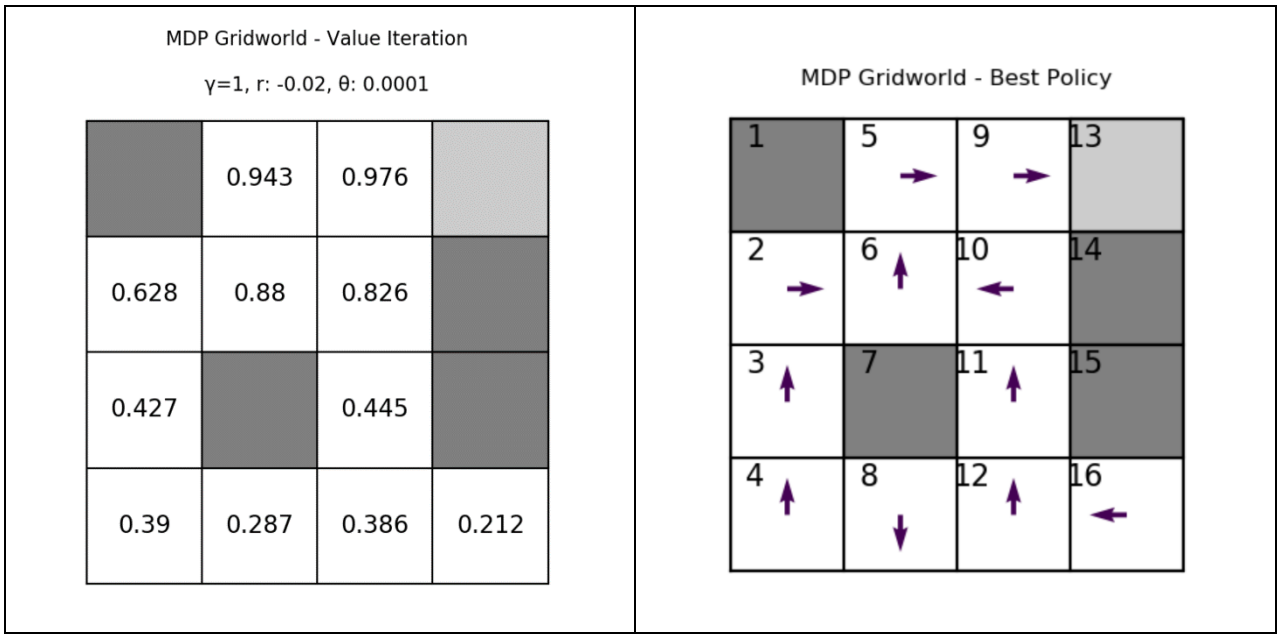
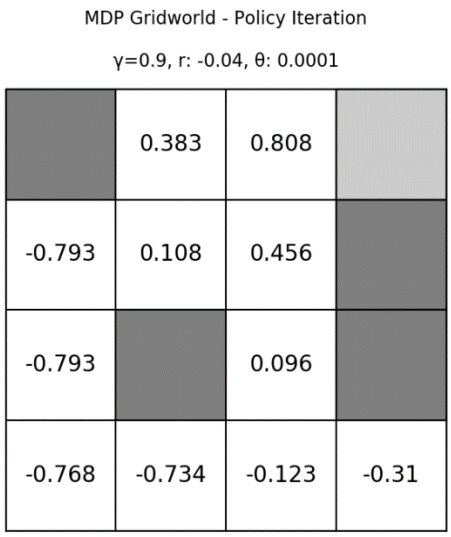


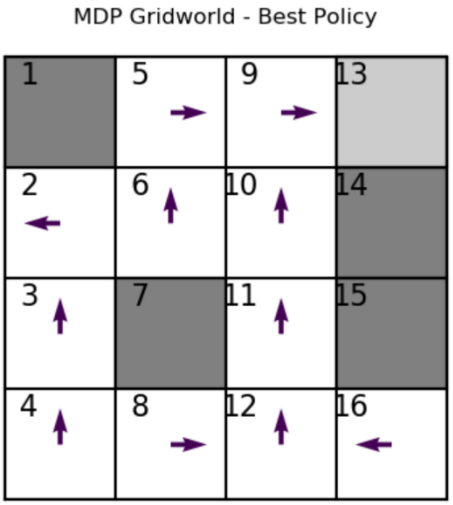
Table 3 - Value Iteration results (d)

e. *Policy iteration:* $\gamma = 0.9\ , \theta = 10^{-4}\ ,\ r = -0.04$:

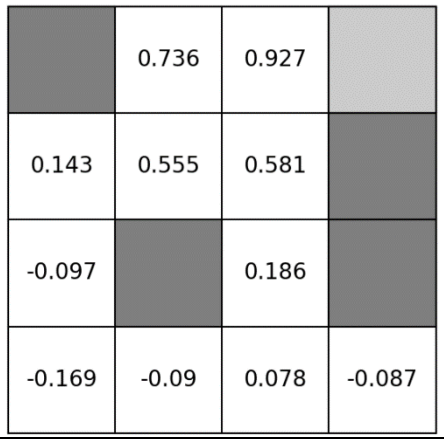
As can be seen in Table 4, in the case of the policy iteration, after initializing the policy vector with randomized actions chosen using a uniform distribution function with an equal chance for each action to be selected (e.g. $0.25\ per\ each\ action\ a$). Moreover, the algorithm converged after 4 iterations to the optimal policy which is similar to the value iteration with the same hyperparameters configuration setting in the 2nd experiment (c). Note that both the policy and the values are equal to the 2nd experiment and therefore, it proofs the theory that the optimal policy must achieve the highest value function as covered in the first part of this report.

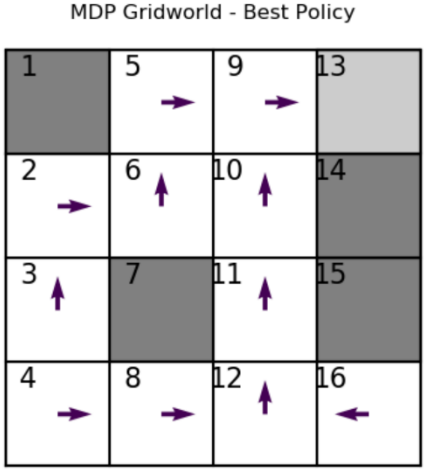
1st Iteration



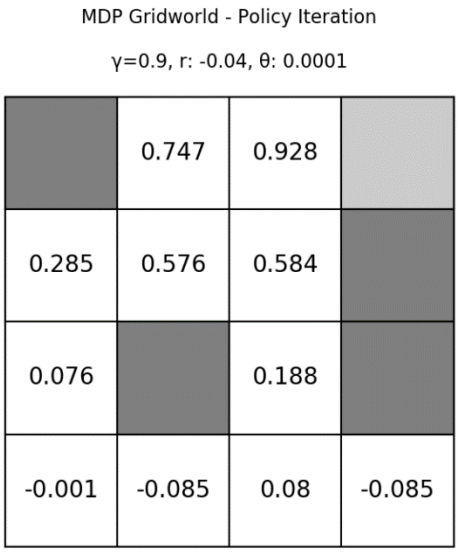


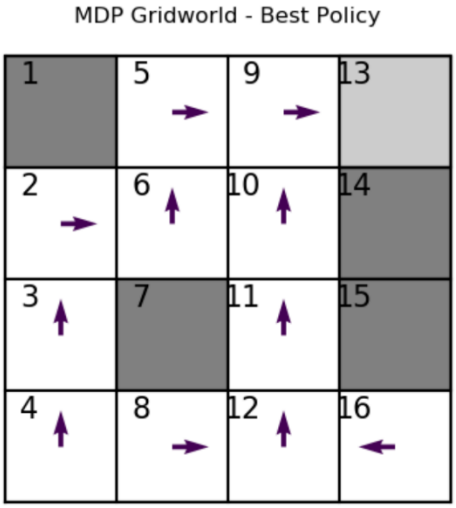
2nd Iteration



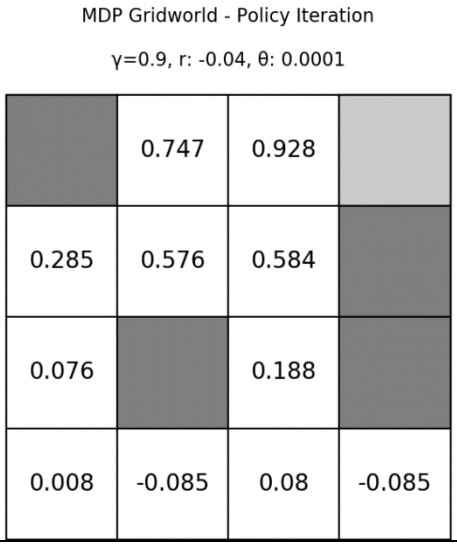


3rd Iteration





4th Iteration



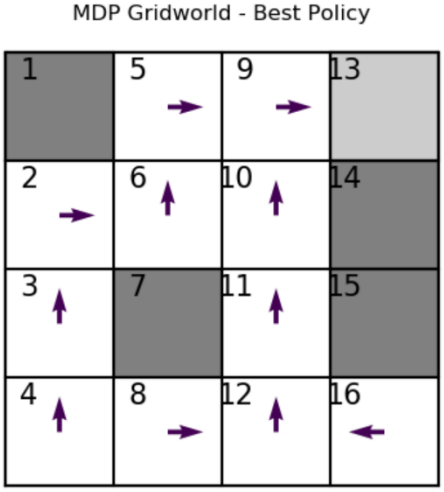


Table 4 - Policy Iteration results (e)

4. Reference

[1] R. S. Sutton and A. G. Barto, “Reinforcement Learning: An Introduction Second edition, in progress,” 2018.

[2] R. Bellman, “The Theory of Dynamic Programming,” *Bull. Am. Math. Soc.*, vol. 60, no. 6, pp. 503–515, 1954.

5. Appendix

5.1. Transition Matrix

From State\ To State (a=N)	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	1	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	1	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	1	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
16	0	0	0	0	0	0	0	0	0	0	0	0.1	0	0	0.8	0.1

From State\ To State (a=E)	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	1	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	1	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	1	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
16	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0.1	0.9

From State\ To State (a=S)	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	1	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	1	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	1	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
16	0	0	0	0	0	0	0	0	0	0	0	0.1	0	0	0	0.9

From State\ To State (a=W)	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
1	1	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	1	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	1	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	0
16	0	0	0	0	0	0	0	0	0	0	0	0.8	0	0	0.1	0.1