We will look at the problem of navigating through a dynamically changing map. It can be represented as a sequence of optimization problems for every time step and in the end it will reduce to a specific case of the Bellmann equation.

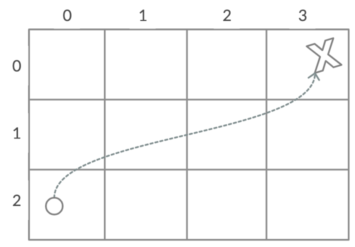

Figure 1: Sample world map

To navigate successfully through a map like the one in figure 1 for a time period $[0, T]$ we have to find a set of policies $\{\pi_t | t = 0, \dots T-1\}$ that will tell us how to move given the time and our position in the map. A possible journey can be described by the sequence of points s_T, s_{T-1}, ..., s_0 and the corresponding probability to use this sequence is given by:

$$\begin{aligned} P(s_T, s_{T-1}, \dots, s_0) = & P(s_T | s_{T-1}, \dots, s_0) \cdot \\ & P(s_{T-1} | s_{T-2}, \dots, s_0) \cdot \\ & \dots \\ & P(s_1 | s_0) \cdot \\ & P(s_0). \end{aligned} \tag{1}$$

The conditional probability density function $P(s_{t+1} | s_t, \dots, s_0)$ describes the probability to move to s_{t+1} given the history of previously visited points $s_t, \dots s_0$. This probability depends on the map (if a point s_{t+1} is an obstacle or not) and on the policy π_t. We will consider the case that the decision to which point s_t to move depends only on the current position s_t, i.e:

$$P(s_{t+1} | s_t, \dots, s_0) = P_{\pi_t(\lambda_t)}(s_{t+1} | s_t) \quad t = 0, 1, \dots T-1, \tag{2}$$

which is also known as the Markov assumption. To emphasize the dependency of P on the policy, in the last equation we have added $\pi_t(\lambda_t)$ as a subscript where λ_t represents all parameters that have to be learned.
Let give an example how $P_{\pi_t(\lambda_t)}(s_{t+1} | s_t)$ can look like. It makes sense to allow only movements towards the four closest bins (up, down, right,left) and to allow the possibility to stay idle. We represent the position s_t as a tuple (i, j) where i, j refer to the i-th row and j-th column of figure 1. In this case the conditional probability density function is given by:

$$P_{\pi_t(\lambda_t)}(\underbrace{s_{t+1}}_{(i'j')} | \underbrace{s_t}_{(i,j)}) = \begin{array}{l} \delta_{i',i} \quad \delta_{j',j} \quad \rho_{0,\lambda_t}(t, s_t) + \\ \delta_{i',i-1} \delta_{j',j} \quad \rho_{1,\lambda_t}(t, s_t) + \\ \delta_{i',i+1} \delta_{j',j} \quad \rho_{2,\lambda_t}(t, s_t) + \\ \delta_{i',i} \quad \delta_{j',j+1} \rho_{3,\lambda_t}(t, s_t) + \\ \delta_{i',i} \quad \delta_{j',j-1} \rho_{4,\lambda_t}(t, s_t), \end{array} \tag{3}$$

where $\delta_{a,b}$ is the Kronecker delta (it is equal to 1 if $a = b$ and 0 otherwise) and $\rho_{k,\lambda_t}(t, s_t)$ $(k = 0, 1, 2, 3, 4)$ are the probabilities to stay or move up, down, right, left from the point s_t at time t. These probabilities have to sum up to 1 which naturally leads to the following parametrisation:

$$\rho_{k,\lambda_t}(t, s_t) = \frac{e^{\lambda_k(t, s_t)}}{\sum\limits_{\tilde{k}=0}^{4} e^{\lambda_{\tilde{k}}(t, s_t)}} \quad k = 0, \dots 4. \tag{4}$$

The task of finding the most efficient set of policies is equivalent to finding $\lambda_k(t, s_t)$ for all k, t, s_t.

To find the best path we will assign a score to every path: we introduce rewards at the desired destination and penalties at the forbidden map points which are added to the score of a path if we cross or stay at one of these points. The task is to maximize the collected score which can happen if we reach the destination as fast as possible and stay there in order to collect the reward as many times as possible. For example, the reward and penalties can be parametrized as follows:

$$R(t, s_t) = \delta_{i,0}\delta_{j,7} - 6\delta_{i,1}\delta_{j,7}, \quad s_t \equiv (i,j). \tag{5}$$

The first term corresponds to the reward that we get at every time step when we are in the upper right corner of the map and the second term corresponds to the penalty that we get when we stay one bin below.

The average score for a set of policies $\pi_{T-1}, \ldots, \pi_0$ under the condition that at $t = 0$ we are at position s_0 is given by:

$$\sum_{s_T,\ldots,s_1} \sum_{t=0}^{T-1} R(t+1, s_{t+1}) \underbrace{\prod_{\tau=0}^{T-1} P_{\pi_\tau(\lambda_\tau)}(s_{\tau+1}|s_\tau)}_{P(s_T,\ldots s_1|s_0)}\,. \tag{6}$$

The first summation over s is responsible for the exploration of all possible trajectoies, and the last term (underbraces) represents the probability for a realization of a trajectory starting from s_0. The reader can compare this term with equation (1) and see that

$$P(s_T, \ldots s_1|s_0) = P(s_T, \ldots s_0)/P(s_0), \tag{7}$$

as expected. Our task is to define the policies such that the expected score is maximized:

$$V^*_{[T-1,\ldots,0]}(s_0) =^{\max}_{\pi_{T-1},\ldots,\pi_0} \sum_{s_T,\ldots,s_1} \sum_{t=0}^{T-1} R(t+1, s_{t+1}) \prod_{\tau=0}^{T-1} P_{\pi_\tau(\lambda)}(s_{\tau+1}|s_\tau). \tag{8}$$

The indices in the square brackets in the left hand side refer to the policies $\pi_{T-1} \ldots, \pi_0$ that have to be optimized. The equation above suggests that we have to optimize all policies simultaneously which seems to be hard. Fortunately, we can represent the equation in a form that will allow us to optimize every policy separately:

$$V^*_{[T-1,\ldots,t]}(s_t) = {}^{\max}_{\pi_t} V_{[T-1,\ldots,t]}(s_t) \tag{9a}$$

$$V_{[T-1,\ldots,t]}(s_t) = \begin{cases} \sum\limits_{s_{t+1}} \left[V^*_{[T-1,\ldots,t+1]}(s_{t+1}) + R(t+1, s_{t+1})\right] P_{\pi_t(\lambda)}(s_{t+1}|s_t) & t < T-1 \\ \sum\limits_{s_{T-1}} R(T, s_T) P_{\pi_{T-1}(\lambda)}(s_T|s_{T-1}) & t = T-1 \end{cases} \tag{9b}$$

More details about the derivation of these equations from eq. (8) can be found **here**. We can solve eq. (9a) for $t = T-1$, for $t = T-2$ and so on. In each one of these equations the unknown policy $\pi_t(\lambda_t)$ depends only on $\lambda_k(t, s_t)$ $(k = 0, \ldots 4)$. The best λ can be found via the gradient ascent method: we move λ in the parameter space in a direction defined by the derivative of $V_{[T-1,\ldots,t]}(s_t)$ with respect to the λ vector and do this until the gradient becomes zero, i.e. until we reach a local maximum of $V_{[T-1,\ldots,t]}(s_t)$. For our problem the λ vector is given by:

$$\vec{\lambda}(t, s_t) =[\lambda_0(t, s_t), \ldots, \lambda_4(t, s_t)]^T, \tag{10}$$

where T stays for 'transposed', i.e. this is a column vector. A possible iterative algorithm to obtain the optimal λ for every (t, s_t) is defined below:

$$\vec{\lambda}^{\{0\}}(t, s_t) =0 \in \mathbb{R}^5, \tag{11a}$$

$$\vec{\lambda}^{\{n\}}(t, s_t) =\vec{\lambda}^{\{n-1\}}(t, s_t) + \alpha \vec{\nabla}_{\vec{\lambda}(t,s_t)} V_{[T-1,\ldots,t]}(s_t)\Big|_{\vec{\lambda}(t,s_t)=\vec{\lambda}^{\{n-1\}}(t,s_t)} \tag{11b}$$

where $\{n\}$ denotes the iteration step. Another option could be the use of the L-BFGS-B algorithm which is much more computationally efficient than the previous algorithm. Also in this case we will have to calculate the derivative of V with respect to the λ vector. We have to take into account that the gradient $\vec{\nabla}_{\vec{\lambda}(t,s_t)}$ is applied only to the P_π term since $V^*_{[T-1,\dots,t+1]}(s_{t+1})$ depends only on $\lambda(\tau, s_\tau)$ with $\tau > t$ and $R(t+1, s_{t+1})$ does not depend on λ at all. The gradient of $P_{\pi_t(\lambda)}$ with respect to λ is given by:

$$\vec{\nabla}_{\vec{\lambda}} P_{\pi_t(\lambda)}(\underbrace{s_{t+1}}_{(i',j')} | \underbrace{s_t}_{(i,j)}) = -P_{\pi_t(\lambda)}(s_{t+1}|s_t) \cdot \begin{bmatrix} \rho_{0,\lambda_t}(t,s_t) \\ \rho_{1,\lambda_t}(t,s_t) \\ \rho_{2,\lambda_t}(t,s_t) \\ \rho_{3,\lambda_t}(t,s_t) \\ \rho_{4,\lambda_t}(t,s_t) \end{bmatrix} + \begin{bmatrix} \delta_{i',i} & \delta_{j',j} & \rho_{0,\lambda_t}(t,s_t) \\ \delta_{i',i-1} & \delta_{j',j} & \rho_{1,\lambda_t}(t,s_t) \\ \delta_{i',i+1} & \delta_{j',j} & \rho_{2,\lambda_t}(t,s_t) \\ \delta_{i',i} & \delta_{j',j+1} & \rho_{3,\lambda_t}(t,s_t) \\ \delta_{i',i} & \delta_{j',j-1} & \rho_{4,\lambda_t}(t,s_t) \end{bmatrix}. \tag{12}$$

We have everything to solve the problem: starting from $t = T-1$ we find $\lambda(t, s_t)$ via 11a, 11b and then use it calculate $V^*_{[T-1,\dots,t]}(s)$ for all positions s_t. This procedure is repeated for $t = T-2, T-3, \dots 0$.

We also have to mention how the impenetrable points are taken into account. For every time step t and position s_t we calculate a mask that takes into account the position of the impenetrable points at $t+1$ and gives us information which actions are possible. Each action corresponds to one of the five indices in eq. (4). If a given action is forbidden, then the corresponding k-index is removed from eq. (4) and the probability $\rho_{k,\lambda_t}(t, s_t)$ is set to 0. If, for example, the actions for moving left and down are not allowed then $\rho_k(t, s_t)$ has the following parametrization:

$$\rho_{2,\lambda_t}(t, s_t) = 0, \qquad k = 2, 4 \tag{13a}$$

$$\rho_{k,\lambda_t}(t, s_t) = \frac{e^{\lambda_k(t,s_t)}}{\sum\limits_{\tilde{k}=0,1,3} e^{\lambda_{\tilde{k}}(t,s_t)}} \quad k = 0, 1, 3. \tag{13b}$$

In this case the equations (11a), (11b) reduce to equations for a three dimensional $\vec{\lambda}$: $\vec{\lambda} = [\lambda_0, \lambda_1, \lambda_4]^T$. In our example the impenetrable points are moving and have 3 distinct states that repeat in the same order.

The python code that is used to solve this problem can be found in this Github repository. The optimal path for the case of starting from $s_0 = (2, 0)$ can be found in the image below. The optimal decisions for each one of the three map states (the three states of the moving obstacle in the 1st, 3rd column of the map) are given in Fig. XY, where under the best possible action k ($k = 0, 1, 2, 3, 4$) you can also see the corresponding probability of ρ_k.

There are two interesting conclusions that can be drawn from Fig. XY:

- The places where the highest probability among all possible actions is 0.5 or 0.33 correspond to the case of having 2 or 3 equally good choices. For example, if you stay at $(0, 2)$ in the top figure you can either wait for 2 turns or to move one bin to the left/down and come back. We have to mention that due to the fact of working with finite precision numbers it is always ossible that some actions get higher probability than other actions even if they are equally good.

- The high penalty for passing through $s = (1, 7)$ forces a person at position $s = (2, 7)$ to pick a longer c-shaped trajectory around the monster and the static obstacle.

The presented map navigation problem was solved by assuming that the decision to which state to move depends only on the current state which allowed us to derive a sequence of simple optimization subproblems. These problems were a specific case of the Bellman equation and were solved by using a gradient ascent method. The proposed solution is not computationally efficient and quickly becomes infeasible if it is applied to more complex and larger maps. Fortunately, there are other approaches to tackle this optimization problem which could be discussed in detail in anther article.