Reactive agents :

- Purely reactive agents : Stateless behavior, has a reward function and have the advantage to have instantaneous actions.

- Markov decision process : The state transitions are not deterministic. Use the

$$V(s_i) = R(s_i, a(s_i)) + \gamma \times V(T(s_i, a(s_i)))$$

  Bellman equation : to calculate the value function where gamma between 0 and 1 is the discount factor. Two different algorithms to use :
  - Value iteration : Take the action that maximize the left member of the Bellman equation and replace it by the old value function. Iterate until the value function don't change more than epsilon. We can then prove :

$$max_{s \in S} |V(s) - V^*(s)| \leq \frac{2\epsilon\gamma}{1-\gamma}$$

    where v* is the true value function.
  - Policy iteration : Choose the policy that maximize the reward and then resolve the Bellmann linear equations. After compute the policy improvement with the new value function. The algorithm stops when the policy remain unchanged.

- Partially observable Markov decision process : The state is not known with certainty. The difficulty is that we need to know the whole history of the iterations. Two algorithms :
  - Value iteration : Graphical resolution : the value function at horizon = 0 are

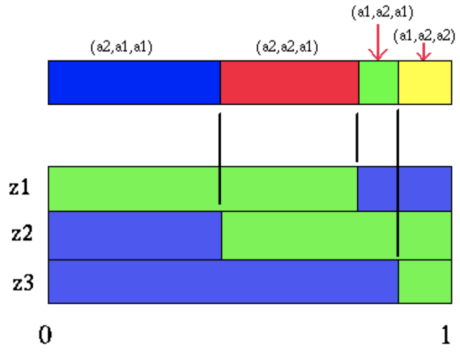
$$\rho(b, a) = \sum_{s \in \mathcal{S}} b(s) R(s, a)$$

    linear . At each action the agent make, he will get an observation that will change his belief space.

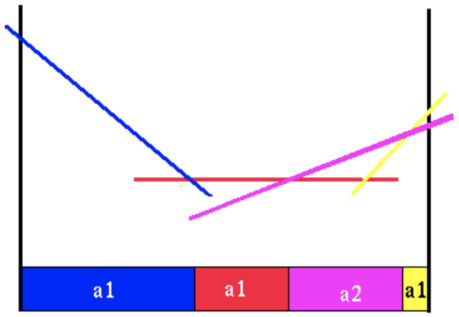
$$\tau(b, a, b') = \sum_{o | SE(b,a,o) = b'} Pr(o|a, b)$$

    . Thus, for each pair of action and observations, we can change the linear function to h = 2 in the belief space of h = 1. For each actio, we can then partition the belief space h = 1

depending on the possible observations.



The linear functions are aggregated for each possible actions giving the best policy to do at h= 1.



- Policy iteration : We can represent the memory policy as finite state controller or policy graph where each nodes represent an action and an arc a particular observation in a particular belief space depending on previous history. The tree search tend to be infinite and a stochastic method is needed to resolve the policy graph.

Machine learning for agents :

- Q-learning : Q-table consisting of number of states rows and number of actions columns. Each time action taken, reevaluate the corresponding case using the Bellman equation with learning rate alpha between 0 and 1 :

$$Q^{new}(s_t, a_t) \leftarrow (1-\alpha) \cdot \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \overbrace{\left(\underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}} \right)}^{\text{learned value}}$$

  Or for a stateless agent :

$$Q_{t+1}(a) = \begin{cases} \alpha r(a) + (1-\alpha) Q_t(a) & \text{for } a = a_t \\ Q_t(a) & \text{otherwise} \end{cases}$$

  Larger alpha gives more importance to recent observations. We can then decrease alpha. alpha = 1/N where N represent the number of times the action was taken. Initializing the Q-table to high value allow more exploration.

  $\Delta_{t+1} \leq (1 - \alpha + \alpha\gamma)\Delta_t$ where deltat is the difference between the Q-table at time t and the true value of the Q-table.

- Regret : Difference in reward between choosing the best action possible and the action that is really taken by the agent. Cumulative regret of the best action a*, LT(a*)

$$L_T = \sum_{t=1}^{T} l_t = max_a \sum_{t=1}^{T} r_t(a) - \sum_{t=1}^{T} r_t(\pi)$$

  :

- The multi-armed bandit problem : Q-learning applied to slot machines. Which slot machine should I use ? This problem underline the exploration-exploitation tradeoff since the agent want to lose the less money possible.

- Epsilon-greedy algorithm : Resolve the exploration-exploitation problem by exploring with probability epsilon time and exploiting with probability (1-epsilon). The cumulative regret grows linearly with time : $(O(c\epsilon t))$ where c is one regret. We can decrease epsilon $L_t = \int_t 1/t = O(\log t)$ but we need to be sure that the Q-table receive enough samples to converge.

- Upper confidence bound algorithm : Resolve the exploitation-exploration tradeoff by “being optimist under uncertainty”. First eliminates the actions such that their upper confidence bound is lower than the lower confidence bounds of other actions. Take the actions having the highest upper confidence bound (actions that are not visited often have an higher upper confidence bounds due to small sample size).

  $Q(a) + \sqrt{\frac{2 \log t}{N_t(a)}}$ where Nt(a) represent the number of time the action a was taken. Derived from the Hoeffding bound.

  Theorem: UCB1 algorithm achieves expected cumulative regret:

  $$lim_{t \to \infty} L_t \leq \frac{8 \log t}{\sum_a \Delta_a}$$

  where *gap* Δ_a is the difference $max_{a'} r(a') - r(a)$.

  Better if the reward distribution is stationary. Difficult to generalize with multiagent system alone.

- Adversarial bandit : Suppose that an adversary change the rewards of the slot machines with the aim to making your regret as big as possible. If we play randomized, the bandit will give a good reward increasing the regret.

- Regret matching : Can beat the adversarial bandit by taking taking an action proportional to the cumulative regret. When playing an action, the cumulative regret of that action cannot increase anymore. The cumulative regret grows as (O(sqrt(t)). It is widely used in game with chance like poker. However, we have to know the reward of every possible actions.

- Exponential weight update : Can beat the adversarial bandit. Imagine that each actions has a weight and the probability of choosing an action is

  $P(a) = \frac{w(a)}{\sum_a w(a)}$. With multiplicative weight update

  $w(a)_{t+1} = w(a)_t(1 + \epsilon r_t(a)/B)$ where B is the upper bound on possible rewards. In this case, $regret = O(\log |A| \sqrt{T})$ with $\epsilon = 1/\sqrt{T}$.

With exp-3 algorithm, $w(a)_{t+1} = w(a)_t e^{\epsilon r_t(a)/p_t(a)}$ where p(a) represents the probability of choosing action a. It gives a more balanced distribution of samples where the weight change is stronger when the action is unlikely to be taken (little previous weight). Regret bound is $O(\sqrt{T|A|\log|A|})$ for $\epsilon \propto 1/\sqrt{t}$.

- Contextual bandits and the exp4 algorithm : Assume that you receive a context at the beginning like the player you will play against, the day you play,... This factors can have an impact on the reward that the exp3 and the other previous algorithms don't take into account. We can use one agent using exp3 for each possible context but it is wasteful. So we can regroup each context into small sets that share similarities and that is the basis for the exp4 algorithm.

- Deep Q-learning : For huge amount of states, use a convolutional neural network that output the Q-value for every actions possible. Rearranging observations to avoid dependencies.