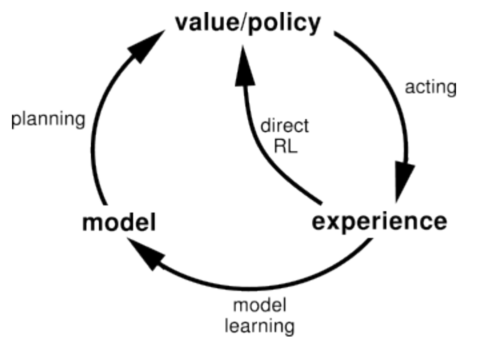
Reinforcement Learning Overview

	Planning: = Dynamic Programming (DP)	**Learning:** = Model-free Reinforcement Learning (RL) [1]	**Hybrid:** **(Planning & Learning)** = Model-based RL
Requirements + Short overall explanation Then, for each method (columns) we iterate between Policy Evaluation (second row) and Policy Improvement (third row)	**Need full knowledge** of environmental dynamics, i.e: $P^a_{ss'}$: (transition probabilities) and $\Re^a_{ss'}$ (expected rewards) Since we have full knowledge, we can solve the problem in full **sweeps** through state-space (i.e. we calculate the update for every state s). Find solution by iterating between policy evaluation and improvement (see below).	**No model** (no $P^a_{ss'}$ or $\Re^a_{ss'}$). **Need learning parameter (α) and exploration method** Relies on **sampling** to explore. At each sample step t observe $\{s_t, a_t, r_t, s'_t\}$ (i.e. state-action-reward-new state). Use this observation to update values (see below). **Action selection (exploration method)**: Vital extra step, only needed for sampling/learning approaches. Need to specify how to select the next sampling action, to ensure we see (explore) the whole space. Over time, we want to start exploiting more and more (to converge to the optimal policy). Best known exploration methods (where $\mathcal{A}$ denotes the (discrete) action space): • ϵ-greedy: $\pi(s,a) = \begin{cases} 1-\epsilon, & if \quad a = \max_b Q(s,b) \\ \frac{\epsilon}{\lvert\mathcal{A}\rvert}, & else \quad (= random) \end{cases}$ • Softmax: $\pi(s,a) = \frac{e^{Q(s,a)}}{\sum_{b \in \mathcal{A}} e^{Q_t(s,b)}}$	Intermediate between planning and learning. Start without knowledge, i.e. based on sampling. Use observed data (transitions and rewards) to gradually built a model (learn $P^a_{ss'}$ and $\Re^a_{ss'}$). Over time, include planning-like updates as well.
Policy Evaluation Update: $V(S)$ (value function) or $Q(S,A)$ (action-value function) [2]	For the current policy π, each state has a unique value that solves the Bellman equation (makes it consistent). We can find this solution by just iteratively applying the Bellman equation from any starting point until convergence. So, in one sweep we do: For each s or s,a: $V^\pi(s) = \sum_a \pi(s,a) \sum_{s'} P^a_{ss'}[\Re^a_{ss'} + \gamma V^\pi(s')]$ $Q^\pi(s,a) = \sum_{s'} P^a_{ss'}[\Re^a_{ss'} + \gamma \sum_{a'} \pi(s',a') Q^\pi(s',a')]$ Note: This equation is **recursive**, i.e. the value of state s depends on the values of its successors. We iterate this equation until convergence and then improve the policy (called *policy iteration*), or we can stop earlier. If we iterate the Bellman equation only once here and then improve the policy (called *value iteration*), we can write both steps in 1 equation (try it out): $V^\pi_{k+1}(s) = \max_a \sum_{s'} P^a_{ss'}[\Re^a_{ss'} + \gamma V^\pi_k(s')]$	As we are sampling, we **only update the value of the state we are currently in**. We base the update on the temporal difference, which is the difference between what we experience this sample step (reward r_t plus stored value of next state $V(s')$) and what we previously thought the value to be ($V(S)$): (so TD = $[r_t + \gamma V_t(s') - V_t(s)]$): **TD-learning** $V_{t+1}(s) = V_t(s) + \alpha[r_t + \gamma V_t(s') - V_t(s)]$ There are two Q-value variants (omitting t now): **Q-learning** (off-policy, sample $\{s,a,r,s'\}$ then max over second action) [3] $Q(s,a) = Q(s,a) + \alpha[r_{sas'} + \gamma \max_b Q(s',b) - Q(s,a)]$ **SARSA** (on-policy, sample $\{s,a,r,s',a'\}$) $Q(s,a) = Q(s,a) + \alpha[r_{sas'} + \gamma Q(s',a') - Q(s,a)]$	Equations not given, but focus on the principle (TD is *acting - direct RL* loop, after some time we start including the model loop as well):  value/policy acting experience model learning model planning direct RL
Policy Improvement Update policy π	**Greedy**: In the previous step we obtained the value function consistent with our current policy. Now we want to improve our policy, based on the value function we found. As we have full knowledge, we can greedily optimize the policy: $\pi(s) = \arg\max_a \sum_{s'} P^a_{ss'}[\Re^a_{ss'} + \gamma V^\pi(s')]$ $\pi(s) = \arg\max_a Q(s,a)$	As we are sampling-based, policy improvement is closely coupled to action selection (see first step). We cannot optimize greedily, as we would loose exploration. The policy improvement still implicitly happens through the change in value estimates. When the values change, we also start to sample different action (e.g. for ϵ-greedy or softmax, see above). Over time, we may decrease exploration (e.g. taking a smaller ϵ to converge to the optimal policy). Interestingly, we may use a different policy to act then to update the value function. This is called *off-policy* learning, of which Q-learning is the prime example. See footnote 3.	Similar to learning.

[1] All sampling methods considered here only make 1-step backups. We might actually backup from multiple steps forward, but we skip that here, see Monte Carlo methods, eligibility traces, and for example TD(λ), SARSA(λ) and Q(λ)

[2] $V(s)$ and $Q(s,a)$ can be rewritten into eachother: $V^\pi(s) = \sum_a \pi(s,a) Q^\pi(s,a)$ and $Q^\pi(s,a) = \sum_{s'} P^a_{ss'}[\Re^a_{ss'} + \gamma V^\pi(s')]$. Try to derive the Bellman equations for V and Q by substituting these equations into eachother. The main point is that each specification (V or Q) is solving the same problem.

[3] Note the difference between Q-learning and SARSA. SARSA (state-action-reward-state-action) samples an action, observes a new state, samples a next action according to the current policy and then uses all this to perform the update. However, Q-learning does need the second sampling step to perform the update, but it 'acts' as if it would continue greedy from there on (see the 'max' in the Q-learning equation). However, the agent will of course not do so, but in fact explore with (small) probability. Thereby, Q-learning learn the values associated with the greedy policy, while it is actually behaving with exploration. SARSA uses the same method for action selection and value estimation. Both methods have pro's and con's: Q-learning finds the best policy, but if the agent keeps exploring in practice, it might continuously incur huge penalties it did not take into account. SARSA is more robust for continuous exploration, but will not find the optimal policy.