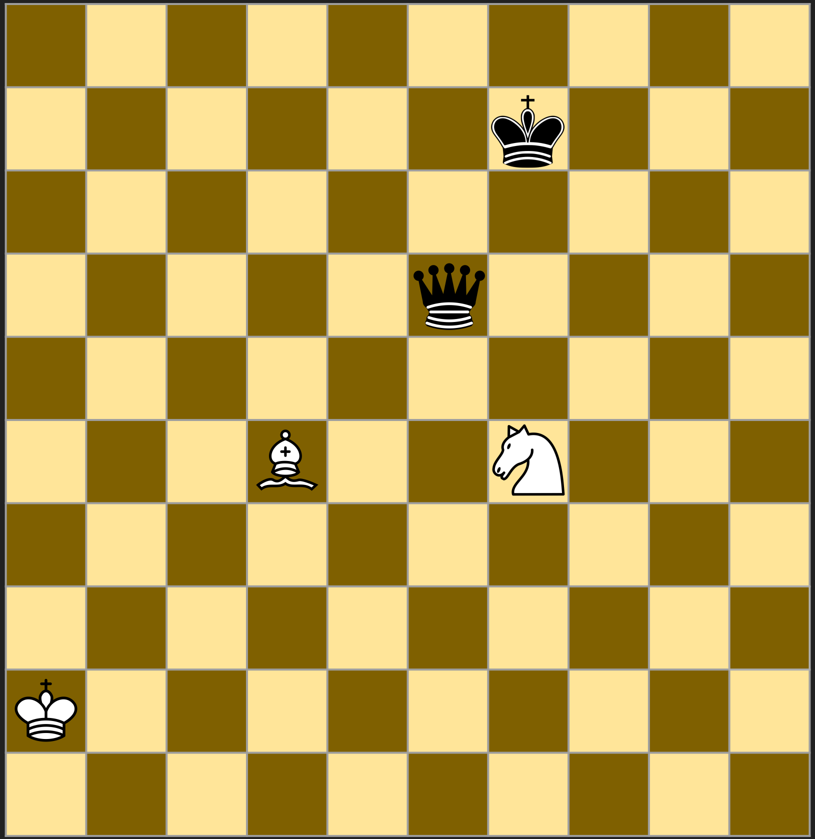
Values, Q-learning* and SARSA

**Not a conspiracy theory*

Values

- The value function measures the future expected reward *in the current state* s *(and if we take the action* a*).*
  - *V(s)* measures the value of a state.
  - *Q(s,a)* measures value of a state and action.
- **N.B.:** Two different actions can correspond to the same state with two different rewards.



Value Functions

Value function:

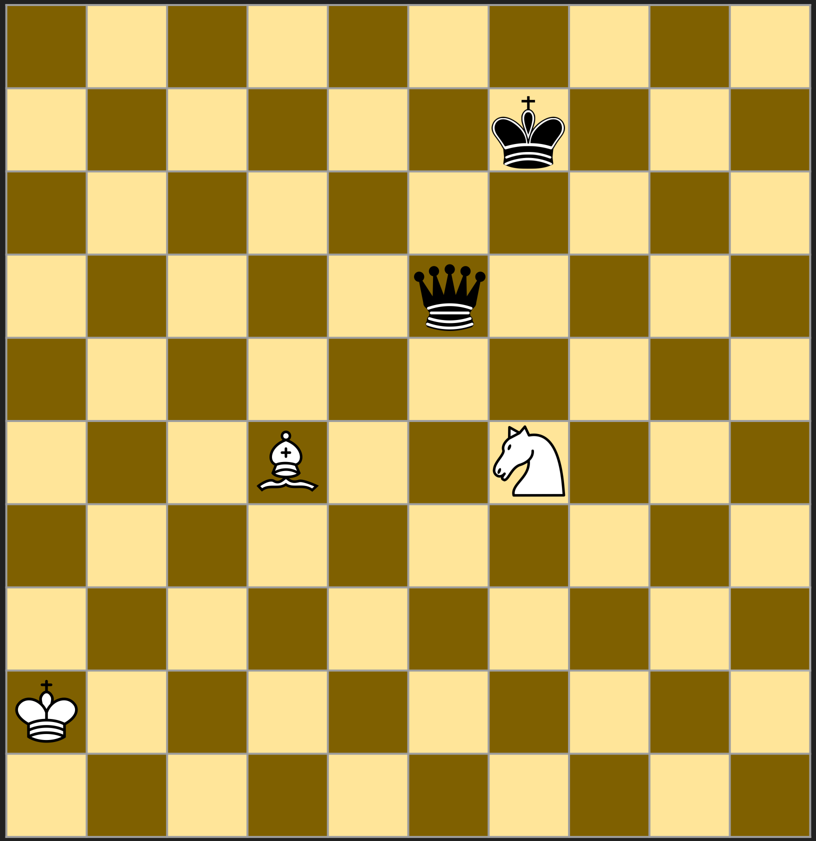
$$V(s) = E\left(r_t \middle| s_t = s\right) = E\left\{\sum_{i=0}^{\infty} \gamma^i r_{t+i+1} \middle| s_t = s\right\}.$$

Action–value function:

$$Q(s,a) = E\left(r_t \middle| s_t = s,\ a_t = a\right) = E\left\{\sum_{i=0}^{\infty} \gamma^i r_{t+i+1} \middle| s_t = s,\ a_t = a\right\}.$$

Policy and Value

- The policy π determines which actions is to be taken in a state.
- The optimal policy π^* maximises the value function over all states (and actions).



Learning the Value Function: the Naïve Approach

Suppose the final state is F, and $V(F) = 100$. Let $0 \leq \gamma \leq 1$.

1. Initialise an empty look-up table of state transitions.
2. Randomly explore the state-space until the absorbing state is reached.
3. Update the value of the last-visited state k-steps back: $V(s_{t-k}) = \gamma^k r$.
4. Repeat from step 2.

Learning the Value Function: the Naïve Approach

Strengthen and constrain the algorithm:

- Include a discounted estimate of future rewards, a time-difference (TD) algorithm:

$$V(s_t) \leftarrow V(s_t) + \mu\left(r_{t+1} + \gamma V(s_{t+1}) - V(s_t)\right).$$

- Include an eligibility trace with a parameter λ, named TD(λ):

$$e_t(s', a') = \begin{cases} 1 & \text{if } s' = s,\ a' = a, \\ \gamma\lambda e_{t-1}(s', a') & \text{otherwise.} \end{cases}$$

Learning the Value Function: the Q-learning Algorithm

Uses the *Q(s,a)* value function. No eligibility trace, therefore written *TD(0)*.

1. Initialise value look-up table *Q(s,a)* with small values.
2. Select a random initial state *s*.
3. Repeat for each step of the episode:
    a. Select and take an action *a* into state *s′*.
    b. Receive reward.
    c. Update value of *s*: $Q(s,a) \leftarrow Q(s,a) + \mu\left(r + \gamma \max_{a'} Q(s', a') - Q(s,a)\right).$
    d. Set *s* to *s′*.

We use the action with the highest reward from the next state to update the value.

This is called an *off-policy decision*.

Learning the Value Function: the Sarsa Algorithm

We may modify the Q-learning algorithm to be *on-policy*, which yields the Sarsa algorithm.

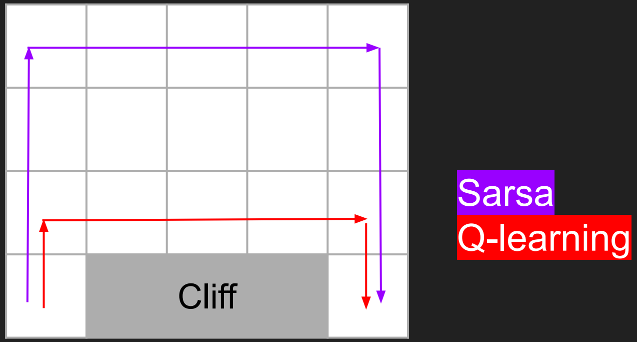
1. Initialise value look-up table *Q(s,a)* with small values.
2. Select a random initial state *s*.
3. Repeat for each step of the episode:
    a. Select and take an action *a* into state *s′*.
    b. Receive reward.
    c. **Select an action *a′* using the current policy.**
    d. **Update value of *s*:** $Q(s,a) \leftarrow Q(s,a) + \mu(r + \gamma Q(s', a') - Q(s,a))$.
    e. Set *s* to *s′*, *a* to *a′*.

sarsa

Sarsa and Q-learning: Similarities and Differences

Similarities:

- Both start out by exploring the search-space randomly.



Differences:

- Q-learning always attempts to follow the shortest path. Merely assumes that the policy will always take the optimal action. The ε-greedy action selection chooses something else.
  - Since Sarsa includes the policy's action selection in the learning, it will avoid obstacles like the plague.