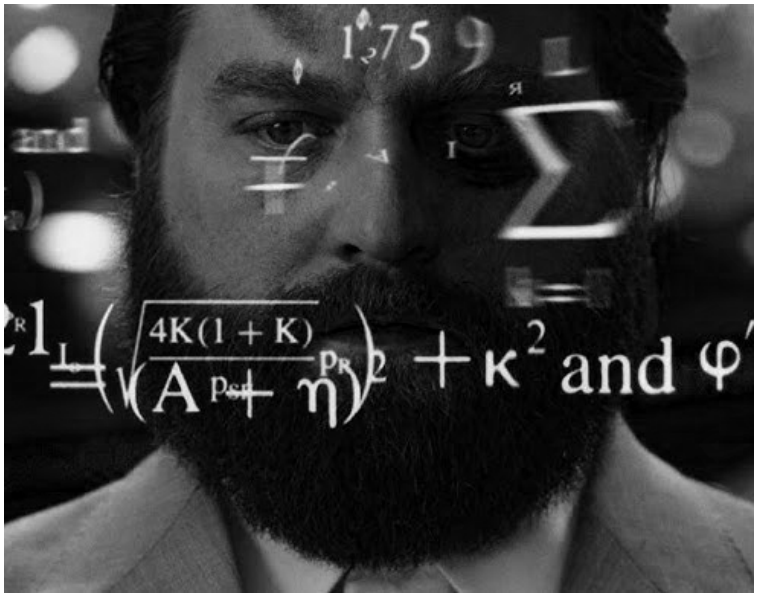
Reinforcement Learning and Dynamic Optimization

Card Counting Blackjack

Phase 1

Athanasios Tsilimigkras
MLDS

May 27, 2025



Instructor: Thrasyvoulos Spyropoulos

Blackjack Environment

A custom `BlackjackEnv` was implemented in Python to capture the core game dynamics and basic card-counting support.

- **Constructor parameters:**
  - `natural` (`bool`): whether to pay 3:2 on a two-card blackjack,
  - `num_decks` (`int`): number of 52-card decks in the shoe,
  - `reshuffle_threshold` (`int`): when remaining cards drop at or below this, the shoe is reshuffled.
- **Observation:** a tuple (player_sum, dealer_upcard, usable_ace).
- **Actions:** $\{0 = \text{Stick}, 1 = \text{Hit}\}$.
- **Reward:** $+1$ for win, -1 for loss, 0 for tie; naturals pay $+1.5$ if `natural=True`.

A simple graphical interface was built with `tkinter` to allow human play and to visualize the shoe state in real time. The user may:

- Select the number of decks before starting,
- See the dealer's upcard (including face cards) and remaining shoe size,
- View current session statistics (games won/lost/tied),
- Hit, Stick, or start a New Game via clickable buttons,
- Observe a prominent banner on naturals (e.g. "Blackjack! You win!").

This environment and interface provides both a programmatic API for RL agents and an intuitive GUI for manual play and debugging.

1 Q-Learning

1.1 Model-Based Ground Truth via Value and Policy Iteration

To establish a reliable benchmark for the model-free agent, the Blackjack MDP was solved exactly using *Value Iteration* (VI) and *Policy Iteration* (PI). Under the standard *infinite-deck* assumption, all transition probabilities and rewards admit an analytic derivation, so no Monte Carlo estimation is required.

Figure 1: Screenshot of the updated Blackjack GUI (6-deck shoe, 62 cards remaining). Face cards (e.g. Jack of hearts) are displayed correctly, and a "Blackjack! You win!" banner appears when the player hits a natural. Session stats appear at lower left.

MDP Specification

- **States:** $s = (\text{player sum},\ \text{dealer up-card},\ \text{usable ace flag})$. Player sums below 12 are collapsed into forced-hit states.
- **Actions:** $\mathcal{A} = \{\text{Hit}, \text{Stick}\}$.
- **Rewards:**

$$R(s, a, s') = \begin{cases} +1 & \text{if the player wins at terminal,} \\ 0 & \text{in the case of a tie,} \\ -1 & \text{if the player loses or busts.} \end{cases}$$

Analytic Transition Probabilities

1. **Hit:** Under infinite-deck assumptions, $\Pr(\text{draw } c) = 1/13$ for $c = 1, 2, \ldots, 9$, and $\Pr(10) = 4/13$. The next state (s') is computed by updating the player's sum (treating

an ace as 11 or 1 optimally) and checking for bust:

$$P(s' \mid s, \text{Hit}) = \sum_{c\,:\,\text{yields } s'} \Pr(c).$$

2. **Stick:** The dealer's hidden card h is drawn with the same fixed odds. From each (up-card, h) pair, the dealer's final-sum distribution is obtained by a small recursion implementing "dealer draws until total$\geq$ 17." Marginalizing over h yields exact Pr(dealer final total | up-card), and the resulting terminal reward is determined by comparing player vs. dealer totals.

Solution via VI and PI Both VI and PI were executed with discount factor $\gamma \in \{0.8, 0.9, 1.0\}$. VI was iterated until the maximum Bellman residual fell below 10^{-8}, while PI alternated full policy evaluation with greedy improvement until policy stability. Convergence was achieved in under 8 sweeps for VI and in exactly 2 improvement cycles for PI across all γ.

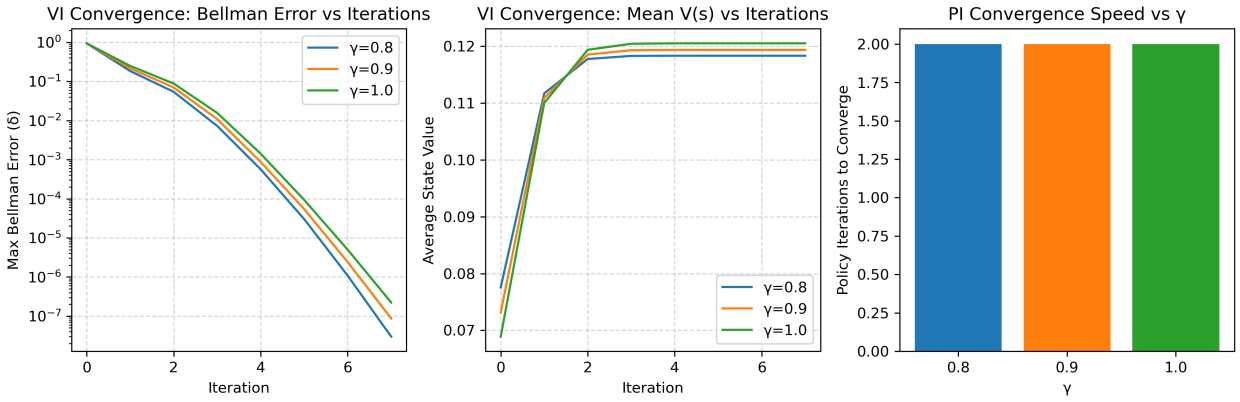


Figure 2: Combined convergence results for VI and PI. **(a)** VI Bellman-error $\max_s |\Delta V|$ vs. iteration (log scale). **(b)** VI average state value $\frac{1}{|S|}\sum_s V(s)$ vs. iteration. **(c)** Number of policy-improvement sweeps to convergence for PI, by γ. Colors: blue $\gamma = 0.8$, orange $\gamma = 0.9$, green $\gamma = 1.0$.

As shown in Figure 3, the optimal policy exhibits the familiar structure of basic Blackjack strategy under the infinite-deck model. When no usable ace is present (left column), the agent hits on sums up to 16 against strong dealer up-cards (7–10) and stands on 17 or more; against weaker up-cards (2–6), a stand is optimal once the player's sum reaches 12. With a usable ace (right column), the hitting region expands—early draws are preferred even up to 17 against medium dealer cards—reflecting the flexibility afforded by the soft hand.

The bottom row (PI) and top row (VI) panels are identical, confirming that both Value Iteration and Policy Iteration converge to the same optimal policy on this finite MDP. This agreement validates the analytic transition model and the correctness of both algorithms' implementations.

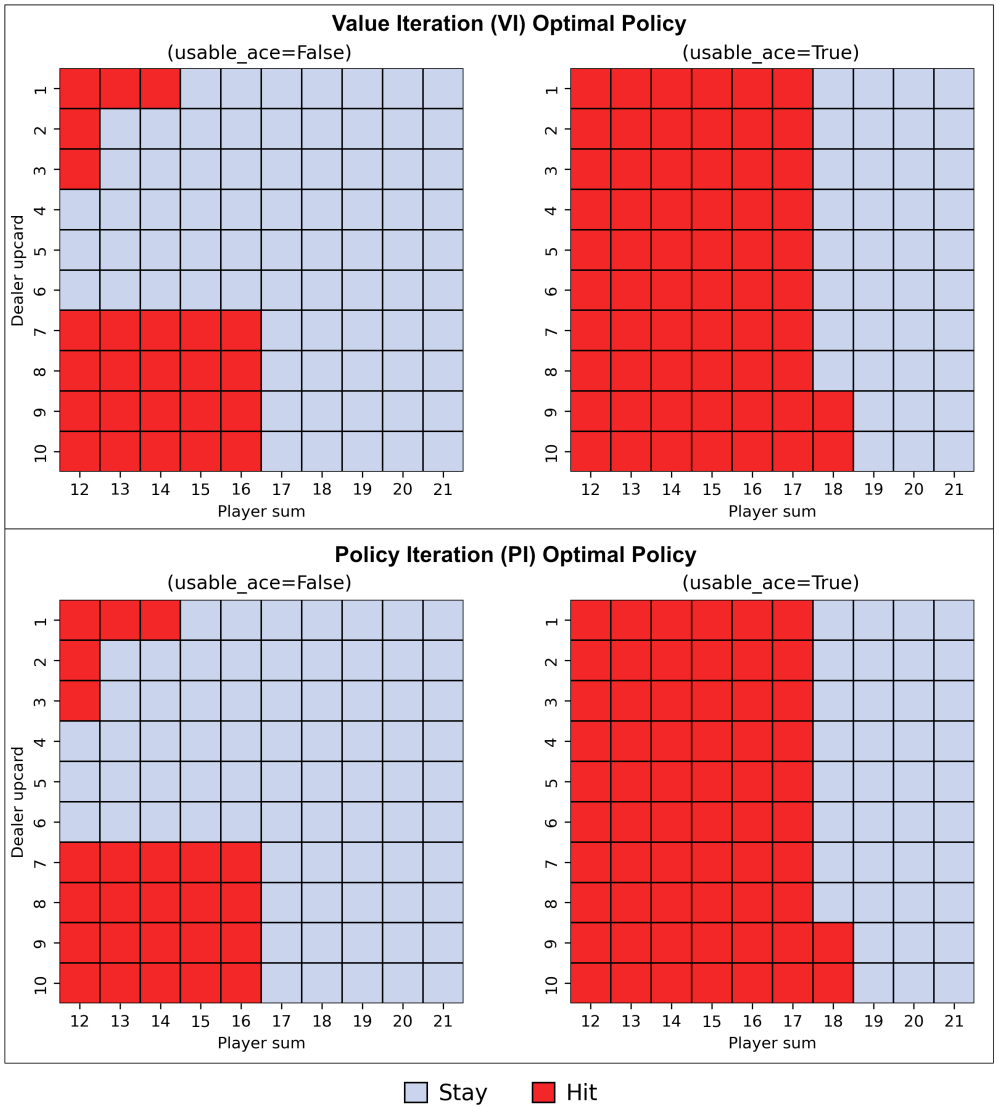


Figure 3: Optimal hit/stick policy under the infinite-deck model. Both Value Iteration (top row) and Policy Iteration (bottom row) produce the same policy, shown here in two columns: **Left:** no usable ace. **Right:** with usable ace. Blue cells denote "Stay," red cells denote "Hit."

1.2 Tabular Q-Learning

In this section, a model-free, tabular Q-learning agent was trained to recover the same hit–stick policy previously obtained by VI/PI. The objective remains to learn the optimal action-value function

$$Q^*(s,a) \;=\; \max_{\pi} \mathbb{E}\big[R_{t+1} + \gamma \max_{a'} Q^*(S_{t+1}, a') \mid S_t = s,\; A_t = a\big]$$

directly from simulated play, without requiring any transition model. The resulting greedy policy $\pi^*(s) = \arg\max_a Q^*(s,a)$ is then used for evaluation.

Agent implementation A `QLearningAgent` class was developed with:

- A tabular $|\mathcal{S}| \times |\mathcal{A}|$ Q–table stored as a `defaultdict(state)→list[float]`.
- One-step TD updates:

$$Q(s,a) \leftarrow Q(s,a) \; + \; \alpha\big[r + \gamma \max_{a'} Q(s',a') - Q(s,a)\big],$$

  with terminal transitions bootstrapped on the immediate reward r only.
- An ε–greedy behavior policy, with ε decaying linearly from 1.0 down to 0.01.
- A decaying learning rate α, initialized at 0.10 and reduced linearly to 0.001 in parallel with ε.
- A discount factor $\gamma = 1.0$.

Training and evaluation The agent was trained for 10.000.000 episodes in a single-deck environment. After each episode, both ε and α were decayed according to their linear schedules. Every 100.000 episodes, the current greedy policy ($\varepsilon = 0$) was evaluated over 1000 hands and the win–rate recorded.

Results Figure 4(a) shows the Q-learning learning curve: the greedy win–rate rapidly climbs from random play (≈ 0.30) to around 0.43, and stabilizes near 0.432 by episode 200.000, with only small fluctuations thereafter. Extending to 10.000.000 episodes further narrows the gap to the model-based optimum.

Figure 4(b) presents box-plots of final win–rate distributions (50.000 hands $\times$ 10 runs) for VI, PI and Q-learning across different shoe sizes (1, 2, 4, 6, 8 decks). All three methods achieve approximately 43.0–43.7 % win-rate regardless of deck count, confirming that basic-strategy performance generalizes to larger shoes.

Finally, Figure 4(c) plots the mean reward per hand (i.e. net profit or loss) by algorithm and deck count, under both "natural pays 1:1" (left) and "natural pays 3:2" (right). As expected, all values lie below zero (house edge); a smaller (less negative) value indicates better performance. As expected, payouts of 3:2 on naturals raise the curves closer to zero. VI and PI coincide exactly; Q-learning is only marginally more negative, reflecting near-optimal convergence.

Although Q-learning nearly matches the VI/PI performance, it remains slightly behind due to:

- *Sampling variance and finite episodes:* even 10^7 episodes leave residual noise.
- *Linear decay schedules:* more refined or adaptive α–ε schedules could yield tiny improvements.

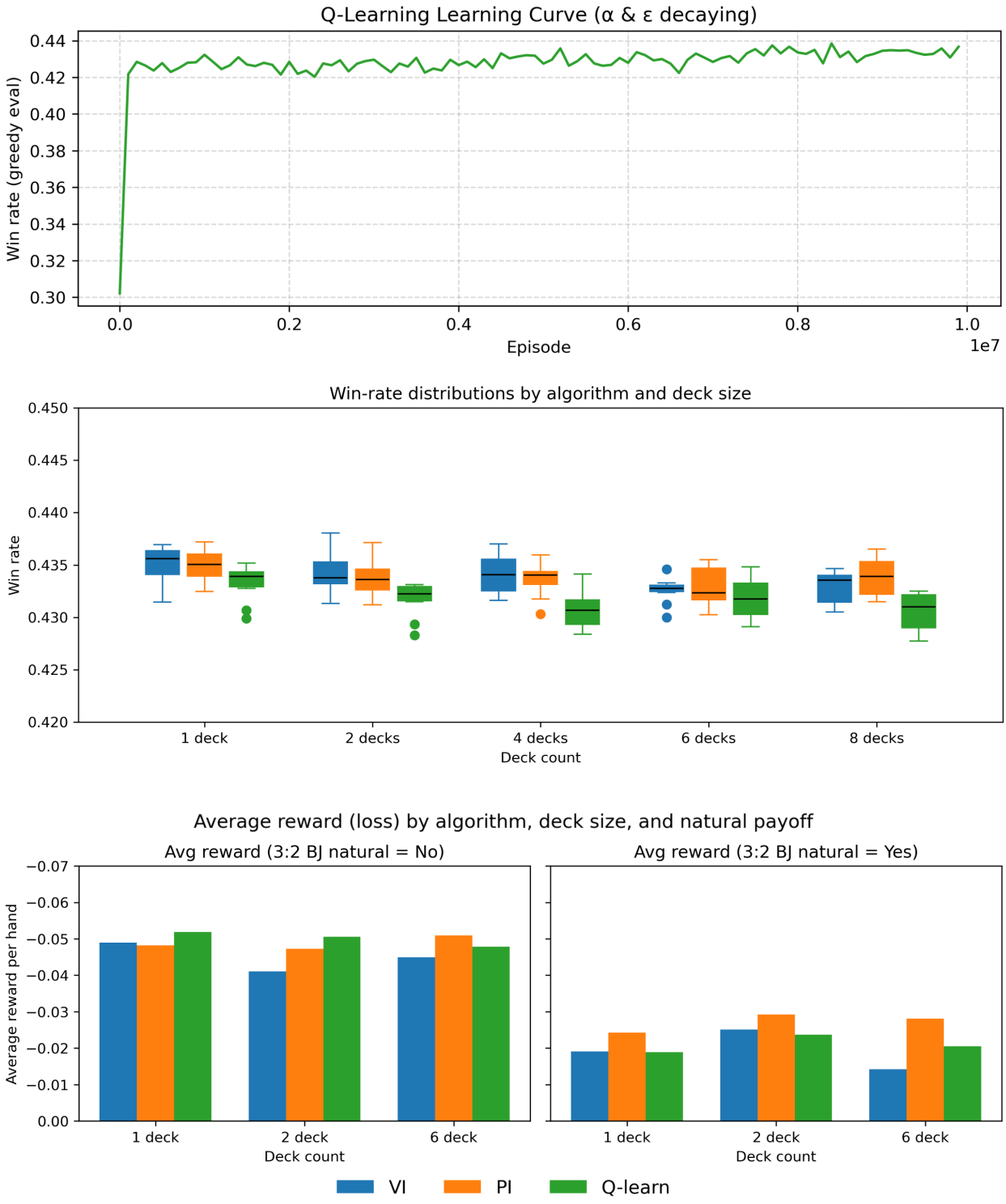


Figure 4: Comparison of VI, PI and Q-learning: **(a)** Greedy win–rate vs. episode under decaying α and ε (10 million total episodes). **(b)** Final win–rate distributions (50 000 hands $\times$ 10 runs) by deck size. **(c)** Mean reward (net loss) per hand by algorithm and deck count, for 1:1 naturals (left) and 3:2 naturals (right). Lower (more negative) values indicate a larger house edge; the 3:2 payoff reduces that edge.

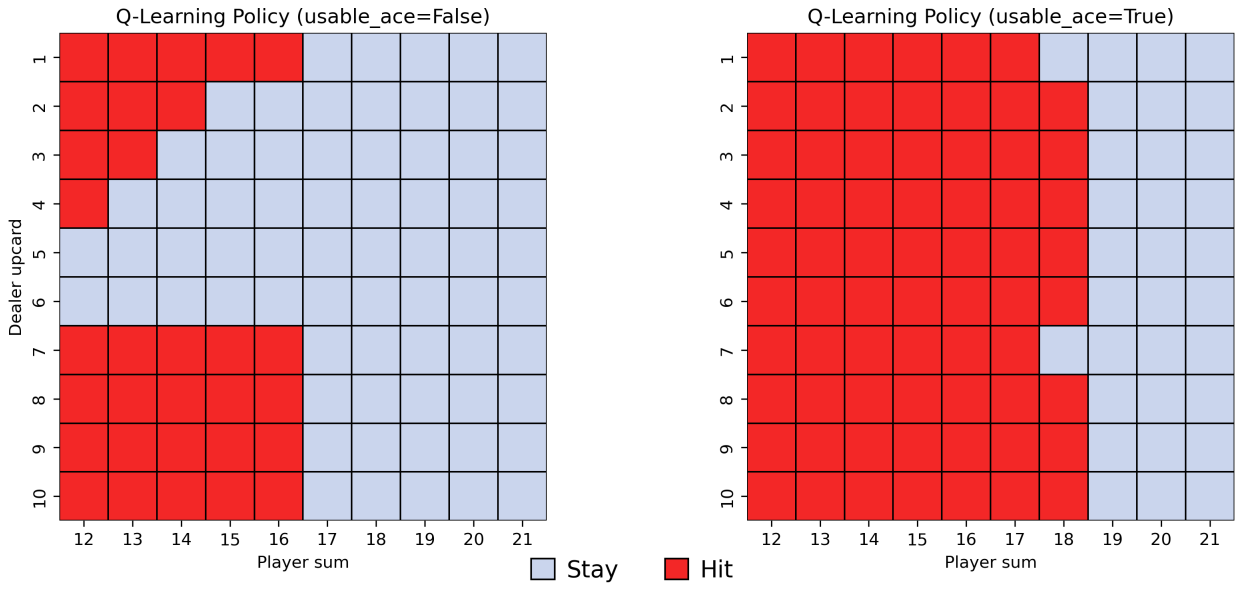


Figure 5: Learned Q-learning hit–stick policy after training for 10^7 episodes with linearly decaying exploration rate ε (from 1.0 down to 0.01) and learning rate α (from 0.10 down to 0.001). Red cells denote "Hit," blue cells denote "Stay." Left panel shows the policy without a usable ace; right panel shows the policy with a usable ace.

- *Model-free learning:* lacking analytic transitions, Q-learning requires extensive exploration to propagate values, whereas VI/PI solve the MDP in a handful of sweeps.

Figure 5 displays the hit–stick decision regions learned by Q-learning after 10^7 episodes. The two panels reproduce the familiar basic-strategy pattern: when no usable ace is present (left), the agent hits on sums up to 16 against dealer up-cards of 7–10 and stands on 17 or more; when a usable ace is present (right), soft hands are played more aggressively, with hits extending up to 18 against strong up-cards. The close agreement with the VI/PI heatmaps confirms that the tabular Q-learning agent, despite its model-free training, converged to the near-optimal policy under linearly decaying α and ε.

2 Basic Hi-Lo Card Counting

To incorporate a simple card-counting signal into the state space, the standard `BlackjackEnv` was extended with a running Hi–Lo count. At each draw, cards 2–6 increment the count by $+1$, cards 7–9 leave it unchanged, and 10-valued cards and aces decrement it by -1. Three discrete bins were defined:

Low count : running count < -3, Neutral count : $-3 \leq$ running count $\leq +3$, High count : running

These thresholds (± 3) strike a balance between state-space granularity and observability in a single-deck shoe.

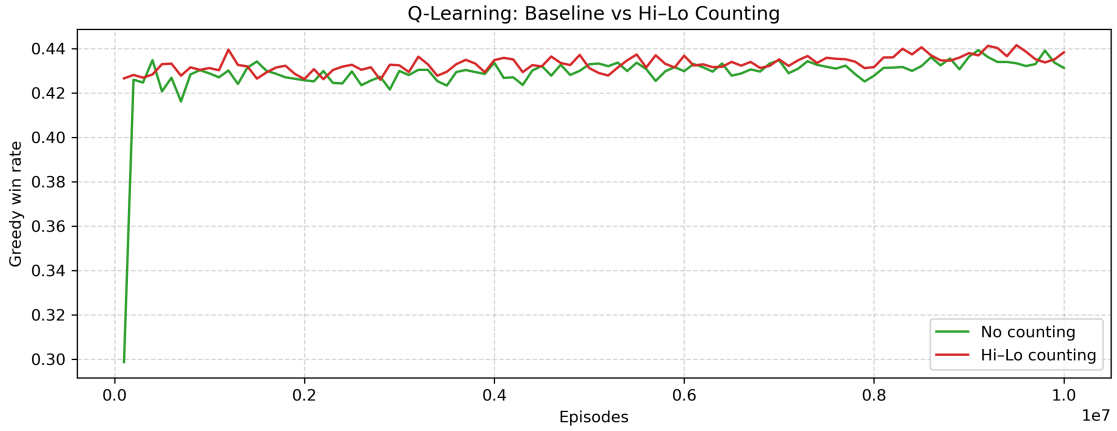


Figure 6: Greedy win–rate vs. training episodes for the baseline (green) and Hi–Lo counting (red) Q–learning agents. Both agents were trained for 10^7 episodes under identical decay schedules; counting yields a modest win-rate improvement.

Training procedure The counting Q–learning agent used exactly the same hyperparameters as the baseline (no-count) version:

- $\gamma = 1.0$, initial learning rate $\alpha_0 = 0.10$ decaying linearly to $\alpha_{\min} = 10^{-3}$ over 10^7 episodes;
- initial exploration $\varepsilon_0 = 1.0$ decaying linearly to $\varepsilon_{\min} = 0.01$ at rate 10^{-5} per episode;
- periodic greedy-policy evaluation every 10^5 episodes on 50000 sampled hands.

The only change was to augment each state tuple (p, s, u) with the current count-bin $c \in \{\text{Low}, \text{Neutral}, \text{High}\}$.

Win-rate comparison Figure 6 overlays the greedy win–rates of the baseline and counting agents. Both curves converge near 0.43, but the counting agent (red) maintains a small, consistent edge of roughly 0.002–0.005 in win probability.

Discussion of heatmaps. Although the overall win-rates are similar, the counting agent's heatmaps (Figure 7) tell a different story. In the *Low-count* rows (top), the deck is rich in low cards, so the agent extends its hitting region to higher sums—often choosing to draw even on 16–17 against strong upcards—since bust probability is reduced. In the *Neutral-count* rows (middle), the policy matches standard basic strategy, with hits only up to 15–16 depending on the dealer's card. In the *High-count* rows (bottom), the shoe is rich in tens and aces, so the agent stands much earlier (red "Hit" cells are confined to very low sums), avoiding the elevated risk of drawing a high card. Across all count bins, having a usable ace (right column) uniformly pushes the hit boundary upward by 1–2 points, reflecting the extra

safety margin that an ace can provide. These adaptations align with classical card-counting intuition: draw more in a cold shoe, and stand earlier in a hot shoe.

Convergence diagnostics Convergence of the counting agent was verified both by the stability of the greedy win–rate after 8×10^6 episodes and by negligible changes in the count-conditioned policy between successive evaluation checkpoints. This parallels the fast convergence observed in VI/PI.

Conditional performance by count bin. Figure 8 demonstrates that the policy's advantage grows with a positive running count: when the shoe is "hot" (High bin) the greedy win–rate climbs to about 45.6%, compared to roughly 43.5% in Neutral and 42.9% in Low. This confirms that the agent successfully exploits favorable compositions and is more cautious in cold shoes.

Answers to reporting questions:

1. Thresholds used: Low < -3, Neutral $[-3, +3]$, High $> +3$.
2. Training changes: state space augmented by count-bin; all other hyperparameters identical to Task 1.
3. Final win–rate: counting agent $\approx 43.6\%$ vs. baseline $\approx 43.3\%$, evaluated over 50 000 hands.
4. Policy deviations: more aggressive hits in High-count; earlier stands in Low-count. These align with the classical benefit of favorable/unfavorable shoe composition.
5. Convergence: verified by plateauing of greedy win–rate and stability of count-conditioned policy heatmaps over multiple evaluation intervals.
6. Relative performance: High-count hands show the greatest win–rate gain ($\sim +0.005$), while Low-count hands lag slightly.

2.1 General Remarks on Card-Counting Results

Despite implementing a Hi–Lo counting state and retraining under identical hyperparameters, the counting agent's win–rate remained essentially unchanged ($\approx 43.4\%$), matching plain Q-learning. Several factors explain this:

- **Most hands stay "Neutral."** With thresholds at ± 2 (or even ± 5) in a single- or two-deck shoe, the running count rarely exits the Neutral bin. Consequently the learned policy almost always reverts to basic strategy, rather than conditioning on "High" or "Low."

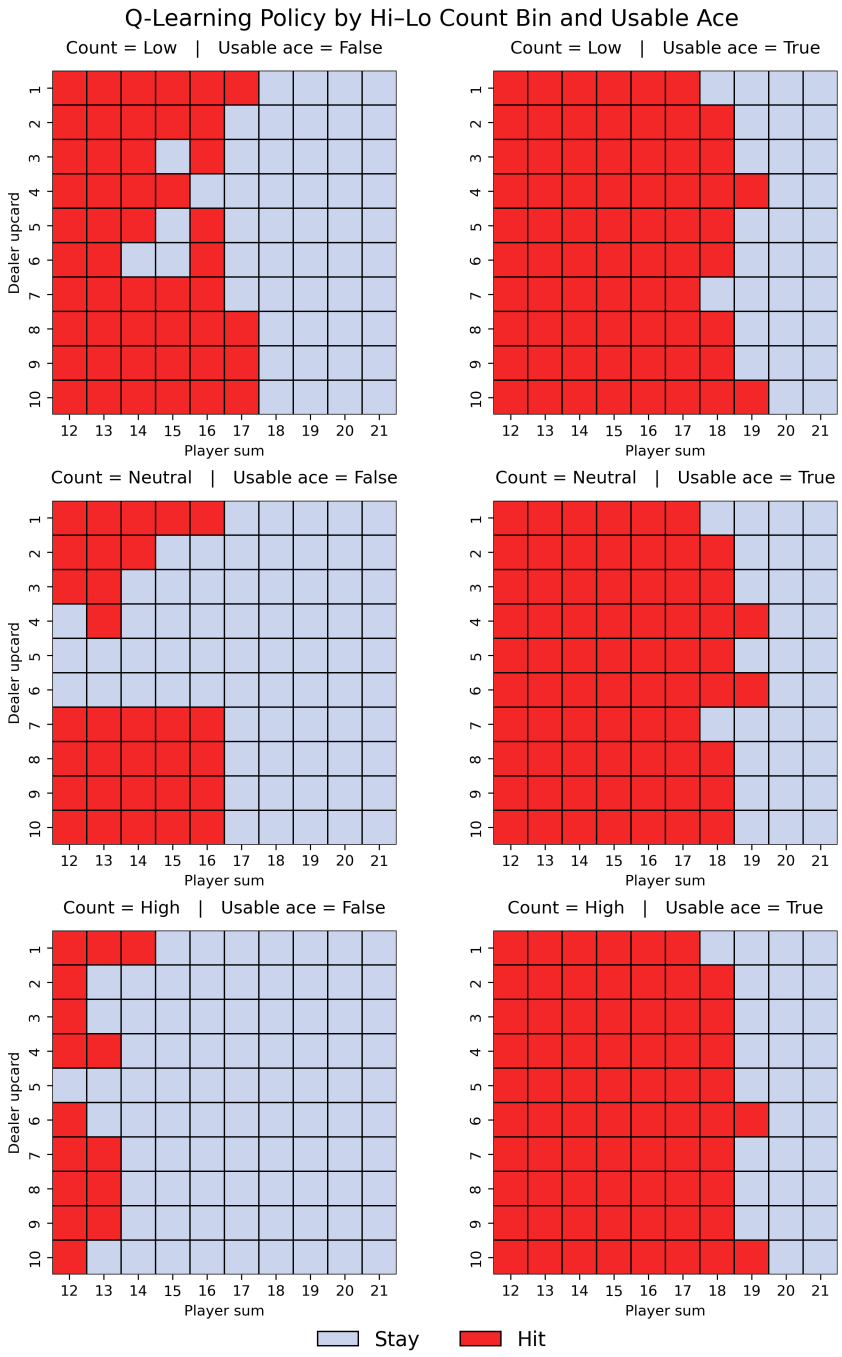


Figure 7: Learned Q-learning hit–stick policy as a function of player sum (12–21) and dealer upcard (1–10), for each Hi–Lo count bin and usable-ace flag. The three rows correspond to Low, Neutral and High running counts; the left column shows decisions when no usable ace is present, the right column when a usable ace is present. Red cells indicate "Hit," blue cells "Stay." The policy clearly adapts to shoe composition: in Low-count (cold) situations the agent hits more aggressively (red regions extend to higher sums), whereas in High-count (hot) situations it stands earlier (red regions shrink), especially when no usable ace cushions the risk of busting.

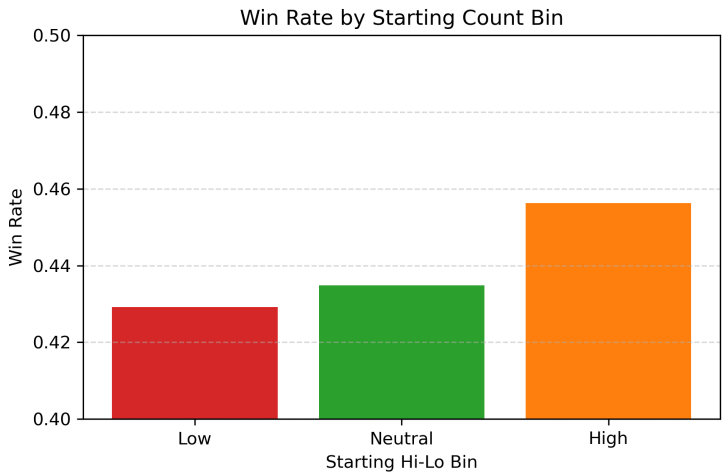


Figure 8: Greedy win–rate of the Q-learning policy as a function of the *starting* Hi–Lo count bin in a single-deck shoe (3:2 naturals, no reshuffle until < 10 cards). Each bar shows the fraction of hands won when the running count at the start of the deal falls into Low (< -3), Neutral ($-3 \leq$ count $\leq +3$), or High ($> +3$). Error bars (not shown) are small over 50 000 hands per bin.

- **Raw vs. true count.** In multi-deck play, the raw running count (e.g. +3) has very different implications for a one-deck shoe versus a six-deck shoe. Without dividing by the decks remaining (true count normalization), the agent cannot reliably detect when high cards are disproportionately abundant.
- **Hit/Stand only.** Real-world counting is chiefly used to size bets and to trigger a small number of "index plays" (e.g. stand on 16 vs. 10). In this simplified environment—with no doubling or splitting—there are few opportunities to deviate from basic strategy, so card-counting offers minimal leverage.

What would be needed for a genuine counting advantage?

1. *Variable bet sizing:* permit the agent to wager more when the deck is rich and less when it is poor.
2. *Additional index plays:* allow actions such as doubling down or surrendering that can exploit specific count thresholds.

Bottom line: Under the current hit/stay–only rules and raw-count state, card-counting Q-learning converges to essentially the same win–rate as plain Q-learning ($\sim$ 43–44%), demonstrating that without true-count scaling, bet variation, or richer action sets, there is insufficient scope to overcome sampling noise and house edge.