The Dice Game Pig as a Markov Decision Process: Greedy, Dynamic Programming, and Q-Learning Compared

Ni Made Sekar Jelita Parameswari - 18223101
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: sekarjelita12@gmail.com , 18223101@std.stei.itb.ac.id

***Abstract*— The dice game Pig is a simple stochastic game that requires players to repeatedly choose between taking additional risk by rolling a die or securing accumulated points by holding. This paper models Pig as a Markov Decision Process (MDP) and compares three approaches for decision-making: a greedy heuristic, Dynamic Programming through Value Iteration, and model-free Reinforcement Learning through tabular Q-Learning. The MDP state is defined by the active player's score, the opponent's score, and the current turn total, while the available actions are roll and hold. Value Iteration is used to compute the Bellman-optimal policy, while Q-Learning is trained through self-play without explicitly using the transition model. Experimental evaluation is conducted using a round-robin tournament with 10,000 matches per pair of strategies. The Value Iteration policy achieved the highest overall win-rate at 69.2%, followed by hold-at-20 at 65.3% and score-aware greedy at 64.4%. In contrast, the trained Q-Learning policy achieved only 1.1% overall win-rate, indicating that the learned policy had not yet converged to a competitive strategy within the recorded training range. These results show that simple greedy heuristics approximate the optimal policy in many common states, but Value Iteration remains superior because it accounts for the full state-dependent structure of the Pig MDP.**

Keywords—Markov Decision Process; Dynamic Programming; Value Iteration; Q-Learning; Reinforcement Learning, Dice Game Pig

I. Introduction

Many decision-making problems involve uncertainty and require balancing immediate rewards against potential future gains [2]. Markov Decision Processes (MDP) provide a mathematical framework for modeling such problems and serve as the foundation of Dynamic Programming (DP) and Reinforcement Learning (RL).

The Dice Game Pig is a stochastic turn-based game in which players repeatedly choose between rolling a die to accumulate additional points or holding to secure their current turn score. While continuing to roll may increase the player's score, rolling a value of one causes all points accumulated during the turn to be lost. This trade-off between risk and reward makes Pig a suitable environment for studying sequential decision-making under uncertainty.

This paper models the Dice Game Pig as a Markov Decision Process and compares three decision-making approaches: a Greedy heuristic policy, a Dynamic Programming approach using Value Iteration, and a Reinforcement Learning approach using tabular Q-Learning. By evaluating these approaches within the same MDP framework, the study highlights the relationship between heuristic, model-based, and model-free decision-making methods and analyzes their effectiveness in achieving optimal play.

II. Theoretical Background

A. Pig Dice Game

Pig is a two-player dice game. On each turn, a player repeatedly rolls a single six-sided die. If the outcome is 2-6, that value is added to the turn total. The player may choose either roll (roll again) or hold (bank the turn total into the permanent score). However, if the outcome is 1, the entire turn total is lost and the turn passes to the opponent. The first player to reach a permanent score of 100 wins the game.

The game has several important properties for modeling: perfect information (both players observe the full state), stochasticity (die outcomes are random), and episodic structure (the game eventually ends within a finite number of turns).

B. Markov Decision Process (MDP)

An MDP is a mathematical framework for sequential decision-making under uncertainty [2]. Formally, an MDP is defined as the tuple (S, A, T, R, γ):

- S: the set of states
- A: the set of available action
- $T(s' \mid s, a)$: the transition probability to state s' from state s after taking ation a
- $R(s.a)$: the reward function
- γ: the discount factor $(0 \leq \gamma \leq 1)$

The agent's objective is to find a policy $\pi: S \rightarrow A$ that maximizes the expected total reward. For episodic games such as Pig, $\gamma = 1$.

C. Greedy Strategy (Heuristics)

A greedy strategy selects an action according to a fixed local rule, without explicitly solving the full optimization problem. In this study, two greedy baselines are considered [6].

1) Hold-at-N.

The hold-at-N strategy chooses hold whenever the current turn total k reaches a predefined threshold N, or whenever holding would immediately end the game. Otherwise, the strategy chooses roll. Previous work on Pig has shown that N = 20 is a strong heuristic and is close to optimal in many settings [1]. The decision rule can be written as follows:

```
ALGORITHM: Hold-at-N
Input: state (i, j, k), threshold N
  IF i + k >= 100 THEN RETURN hold
  IF k >= N       THEN RETURN hold
  RETURN roll
```

2) Score-Aware Greedy

The score-aware greedy strategy extends hold-at-N by adapting the threshold according to the current score difference. When the player is behind by a large margin, the threshold is increased to encourage more aggressive rolling. Conversely, when the player is ahead, the threshold is reduced to favor safer play. In endgame states, particularly when i >= 80, the threshold is also reduced so that the player can secure a win with lower risk.

D. Dynamic Programming – Value Iteration

Value Iteration is a DP algorithm that computes the optimal value function iteratively [3]. It starts by initializing $V(s) = 0$ for all non-terminal states, then repeatedly applies the Bellman equation:

$$V(s) \leftarrow \max_{a \in A} \sum_{s'} T(s'|s,a) \; [R(s,a) + \gamma \, V(s')] \qquad (1)$$

The iteration stops when the maximum change $\max_s |V_{\text{new}}(s) - V_{\text{old}}(s)| < \varepsilon$ for a small convergence threshold ε (for example, 10^{-9}). The result is the optimal value function and the optimal policy extracted greedily from it.

```
ALGORITHM: Value Iteration
Input:  state space S, actions A, transition T, epsilon
Output: V*, pi*

  V(s) <- 0 for all non-terminal states s
  REPEAT
    delta <- 0
    FOR each state s:
      v_old <- V(s)
      V(s) <- max_a SUM_s' T(s'|s,a) * [R + gamma * V(s')]
      delta <- max(delta, |V(s) - v_old|)
  UNTIL delta < epsilon

  pi*(s) <- argmax_a SUM_s' T(s'|s,a) * [R + gamma * V(s')]
```

This algorithm produces an exact solution up to numerical precision. Unfortunately, it requires the complete transition model to be known.

E. Q-Learning (Reinforcement Learning)

Q-Learning is a model-free RL algorithm that learns the action-value function $Q(s,a)$ directly from experience without knowing the transition model [4] [5]. The update is performed whenever the agent acts a in state s, observes reward r, and reaches the next state s':

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[r + \gamma \max_{a'} Q(s',a') - Q(s,a) \right] \qquad (2)$$

- α: learning rate

During training, the agent uses an epsilon-greedy policy. With probability ε, it chooses a random action (exploration), otherwise it chooses the action with the highest Q value (exploitation). The values of α and V are decreased linearly during training to support convergence.

```
ALGORITHM: Q-Learning (Self-Play for Pig)
Input:  episodes, alpha_start, alpha_end, eps_start, eps_end
Output: Q-table

  Q(s,a) <- 0 for all (s,a)
  FOR episode = 1 TO episodes:
    s0, s1 <- 0;  turn <- 0
    WHILE game is not finished:
      i, j <- scores of the current player
      k <- 0   // beginning of turn, roll is mandatory
      LOOP:
        IF k = 0: action <- roll              // mandatory roll
        ELSE:     action <- eps-greedy(Q, s)
        IF action = hold:
          reward <- 1 if i+k >= 100, else 0
          Q(s,hold) <- Q(s,hold) + alpha * [target - Q(s,hold)]
          // target = reward if the player wins,
          //        = 1 - max_a' Q(opponent_state, a') otherwise
          BREAK
        ELSE: // roll
          die <- random(1..6)
          IF die = 1:
            Q(s,roll) <- ... update with opponent state
            BREAK
          k <- k + die
          IF i+k >= 100: update Q, BREAK
          Q(s,roll) <- ... update with next state
      Switch turns
```

Although it does not require a transition model and can learn purely from experience, the convergence is slow for large state

spaces, and the result depends on the number of episodes and the chosen hyperparameters.

III. Modelling and Implementation

A. MDP Formulation for Pig

The Pig game is modeled as an MDP from the perspective of the player whose turn is currently active. Because the game is symmetric, one value function is sufficient for both players since the roles are distinguished by the order of the arguments.

State. Each state is a triple $s = (i, j, k)$ where:

- i: the permanent score of the current player ($0 \leq i \leq 99$)
- j: the permanent score of the opponent ($0 \leq j \leq 99$)
- k: the current turn total ($k \geq 0$)

Actions. $\mathcal{A} = \{\text{roll,hold}\}$

Transitions. The die is fair and six-sided, with outcome $r \in \{1, \dots, 6\}$ occurring with probability $\frac{1}{6}$.

- Hold: if $i + k \geq 100$, the player wins (terminal state). Otherwise, the turn passes to state $(j, i + k, 0)$.
- Roll, $r = 1$: the turn total is lost and the turn passes to $(j, i, 0)$.
- Roll, $r \in \{2, \dots, g\}$: the player continues at $(i, j, k + r)$.

Rewards. A reward of \$+1\$ is given only at the winning terminal state. All other steps receive reward 0. With $\gamma = 1$ (episodic), the optimal value function $P(i, j, k)$ is equal to the probability of winning. This property unifies the DP and RL analyses.

State-space size. Since k is effectively bounded by $0 \leq k < 100 - i$ the number of non-terminal states is approximately 505,000, which is lightweight for a typical CPU.

B. Bellman Equation

Let $P(i, j, k)$ denote the probability that the active player eventually wins the game from state (i, j, k), assuming that both players follow optimal policies. Under this formulation, the value function directly represents the probability of victory.

For the hold action, the player banks the accumulated turn score k. If the resulting total score reaches or exceeds the target score of 100, the game terminates and the player wins immediately. Otherwise, the turn passes to the opponent, yielding

$$H(i, j, k) = \begin{cases} 1 & \text{if } i + k \geq 100 \\ 1 - P(j,\ i + k,\ 0) & \text{otherwise} \end{cases} \quad (3)$$

The second case follows from the zero-sum nature of the game: once the turn is passed, the opponent becomes the active player, and the current player's probability of winning is therefore the complement of the opponent's winning probability.

For the roll action, the outcome depends on the value obtained from the die. Rolling at 1 causes the player to lose all accumulated turn points and immediately pass the turn to the opponent. Rolling any value from 2 to 6 increases the turn score while allowing the player to continue the current turn. Consequently, the expected value of the roll action is

$$R(i, j, k) = \frac{1}{6}\,[\,1 - P(j,\ i,\ 0)\,] + \frac{1}{6}\sum_{r=2}^{6} P\,(i,\ j,\ k + r) \quad (4)$$

The first term represents the bust event $r = 1$, whereas the summation accounts for all non-busting outcomes. The optimal value function is obtained by selecting the action that maximizes the probability of eventual victory:

$$P(i, j, k) = \max\{\,H(i, j, k),\ R(i, j, k)\,\} \quad (5)$$

An important characteristic of this formulation is the presence of cyclic dependencies. Specifically, the value $P(i, j, k)$ depends on $P(i, j, 0)$ through the bust transition, while the latter may in turn depend on the original state. As a result, the state space cannot be arranged in a topological order suitable for direct dynamic programming updates. Therefore, an iterative solution method such as Value Iteration is required to compute the fixed point of the Bellman optimality equation.

C. Value Iteration Implementation

The Bellman optimality equation described in the previous section cannot be solved through a single backward pass because the Pig MDP contains cyclic dependencies between states. Therefore, an iterative Dynamic Programming approach, namely Value Iteration, is employed to compute the fixed point of the Bellman equation.

The value function is represented as a three-dimensional array $P(i, j, k)$, where i denotes the active player's score, j denotes the opponent's score, and k denotes the accumulated turn score. All non-terminal states are initialized to zero. States satisfying $i + k \geq G$ are terminal winning states and are initialized with value 1.

At each iteration, the hold value $H(i, j, k)$and roll value $R(i, j, k)$ are computed for every non-terminal state according to the Bellman equations. The state value is then updated using

$$P(i, j, k) = \max\{H(i, j, k), R(i, j, k)\} \quad (6)$$

The update process is repeated until the maximum absolute difference between consecutive iterations falls below a convergence threshold ($\varepsilon = 10^{-9}$). After convergence, the optimal policy is derived directly from the final value function by comparing the two action values at each state. The policy selects hold when $H(i, j, k) \geq R(i, j, k)$ and selects roll otherwise. In this way, Value Iteration produces both the estimated winning probability for each state and the corresponding optimal action.

To make the computation efficient, the turn-score dimension k is bound by the winning condition and padded slightly to allow roll outcomes to be evaluated without additional boundary checks. The computation is vectorized over the opponent score and turn-score dimensions, so the main iteration only needs to sweep over the active player's score. In the experiment, the method converged after approximately 343 iterations. The

resulting value for the initial state was $P(0,0,0) \approx 0.5306$, indicating that the first player has a slight advantage under optimal play.

The optimal policy obtained from Value Iteration is used as the reference strategy in the subsequent evaluation. It also serves as a benchmark for comparing the greedy heuristics and the policy learned through Q-Learning.

D. Q-Learning Implementation

Q-Learning is used as a model-free reinforcement learning approach to approximate the optimal policy without explicitly using the transition probabilities of the Pig MDP. Unlike Value Iteration, which requires the full model of dice outcomes and state transitions, Q-Learning estimates the action-value function Q(i,j,k,a) through repeated interaction with the game environment.

The Q-table is represented as a four-dimensional array indexed by the active player's score i, the opponent's score j, the current turn total k, and the selected action a. All Q-values are initially set to zero. During training, the agent plays repeated self-play episodes. At each decision state, the agent selects an action using an epsilon-greedy policy: with probability epsilon, it explores by choosing a random valid action, and otherwise it exploits the action with the highest current Q-value.

The update rule follows the standard Q-Learning formulation. After taking an action, observing the resulting state, and receiving a reward, the corresponding Q-value is adjusted toward a target estimate. For a hold action that does not immediately end the game, the turn passes to the opponent. Therefore, the target value is expressed as one minus the opponent's best estimated winning probability,

$$1 - max_a{}'Q(j, i + k, 0, a') \quad (7)$$

Similarly, when a roll produces a value of 1, the turn total is lost and the opponent receives the next turn at state (j,i,0). If a roll or hold action reaches the goal score G, the target value is set to 1 because the active player wins immediately.

A special convention is applied when k = 0. In this state, the agent is required to roll, since holding zero points would not change the game state and could produce a non-progressing loop. This convention is also consistent with the game simulation, where each turn begins with an initial roll before a policy decision is requested.

Training uses symmetric self-play. Because Pig is a symmetric two-player game, a single Q-table is shared by both players; the state representation is rotated depending on which player is active. The learning rate alpha and exploration rate epsilon are both decreased linearly during training, with alpha reduced from 0.10 to 0.02 and epsilon reduced from 0.40 to 0.02. After training, the learned policy is obtained by selecting the action with the highest Q-value in each state.

E. Tournament and Evaluation

The trained Q-Learning policy is evaluated together with the Value Iteration policy and the greedy baselines through a round-robin tournament. To ensure a fair comparison, each pair of strategies is evaluated using a swap-first setting: one strategy starts the first half of the matches, while the other strategy starts the second half. This procedure reduces the bias caused by the first-player advantage, which is approximately 53% under optimal play.

Each pair of strategies plays N matches, with N set to 10,000 by default. Since four strategies are compared, the tournament consists of six pairwise matchups, resulting in 60,000 total games. The evaluation metrics include overall win-rate, head-to-head win-rate matrix, average number of turns per game, and the agreement between the Q-Learning policy and the Value Iteration policy.

IV. Result and Analysis

This chapter evaluates the quality of the policies produced by the three approaches: greedy heuristics, Dynamic Programming through Value Iteration, and model-free reinforcement learning through Q-Learning. The evaluation uses the experimental outputs generated by the repository, namely the round-robin tournament results, convergence records, learning-curve measurements, and policy heatmaps. All tournament results are comparation of four policies using 10,000 matches per pair, seed 42, and a swap-first design to reduce first-player bias.

A. Round-Robin Performance

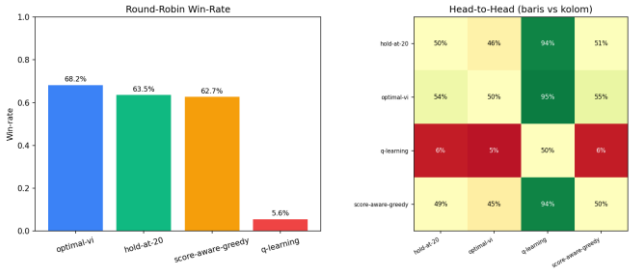

Fig. 1. Win-rate bar chart and head-to-head heatmap

The round-robin results show that the Value Iteration policy achieved the strongest overall performance among the evaluated strategies. As shown in Table I, the optimal VI policy won 20,464 of 30,000 matches, corresponding to an overall win-rate of 69.18%. This result is consistent with the role of Value Iteration in the Pig MDP: because it directly solves the Bellman optimality equation using the known transition model, it produces a policy that explicitly accounts for the future consequences of both holding and rolling.

TABLE I. Overall Win-Rate

Strategy	Win	Match total	Win-Rate
Optimal (VI)	20,464	30,000	69.2%
Hold-at-20	19,063	30,000	65.3%
Score-Aware Greedy	18,800	30,000	64.4%
Q-Learning	1,673	30,000	1.1%

The two greedy baselines performed below the VI policy but remained competitive with each other. The hold-at-20 strategy achieved a 65.3% overall win-rate, while the score-aware greedy strategy achieved 64.4%. Their direct matchup was nearly balanced: hold-at-20 won 50.75% against score-aware-greedy. This suggests that the additional score-aware adjustment did not provide a measurable advantage in the current tournament output. Although score awareness is intended to adapt risk-taking to the score difference, the fixed hold-at-20 threshold remained slightly stronger in the generated results.

TABLE II. HEAD-TO-HEAD MATRIX

	Hold-at-20	Score-Aware	Optimal (VI)	Q-Learning
Hold-at-20	-	50.8%	45.9%	93.9%
Score-Aware	49.3%	-	44.8%	93.8%
Optimal (VI)	54.1%	55.1%	-	95.5%
Q-Learning	6.1%	6.2%	4.5%	-

The head-to-head matrix provides a clearer comparison between the heuristic and optimal strategies. The VI policy won 54.06% against hold-at-20 and 55.17% against score-aware greedy. These margins are not extremely large, which indicates that simple greedy thresholds capture a substantial portion of the useful structure of the Pig decision problem. However, the consistent advantage of VI shows that local threshold rules do not fully reproduce the optimal policy implied by the Bellman equation. In particular, the MDP state includes both players' scores as well as the current turn total, while a simple greedy rule only approximates this dependence through a limited threshold rule.

The current Q-Learning policy performed poorly in the tournament. It won only 322 of 30,000 matches, or 1.07% overall, and lost more than 98% of its head-to-head games against each of the other strategies. This result should be interpreted as evidence about the current saved experimental output, not as a general limitation of Q-Learning. The placement guide and table notes state that the current tournament data uses a sandbox-trained Q-Learning policy, and the available learning-curve CSV shows only limited convergence up to 500,000 episodes. Therefore, the tournament result indicates that the current cached Q-table has not learned a competitive policy.

B. Value Iteration Convergence

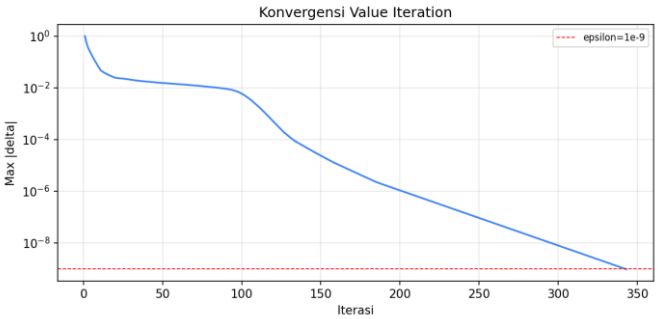

Fig. 2. VI Convergence Curve (log delta vs iteration)

The convergence behavior of Value Iteration supports its suitability for the formulated Pig MDP. According to the experiment, the maximum absolute update began at 1.0 in the first iteration and decreased below the convergence threshold of 343 iterations. The final recorded maximum update was approximately $9.54e^{-10}$. The corresponding convergence plot in Fig. 2 shows a steady reduction in the Bellman residual on a logarithmic scale.

This convergence pattern is important because the Pig MDP contains cyclic dependencies. A bust transition after rolling a 1 maps (i, j, k) to the opponent's state $(i, j, 0)$, while holding also changes the active player when the game does not end. As a result, the state values cannot be computed by a single backward pass. The observed convergence confirms that repeated Bellman updates successfully compute the fixed point of the optimality equation for the implemented state space.

TABLE III. CONVERGENCE SUMMARY

Algorithm	Iterations/Episodes	Time	Notes
Value Iteration	343 iterations		ε = 10^{-9}, P(0,0,0) = 0.5306
Q-Learning	500.000 episodes	123.573	Agreement = 49.51%, win rate = 4.4% vs optimal

The saved value table gives $P(0,0,0) = 0.5305927250180329$, which means that the first player has a slight advantage under optimal play. This value is consistent with the tournament design choice to use swap-first evaluation: if the starting player has an advantage, then alternating the starting policy across each pairwise series is necessary to make win-rate comparisons reflect policy quality rather than turn order.

C. Q-Learning Behavior

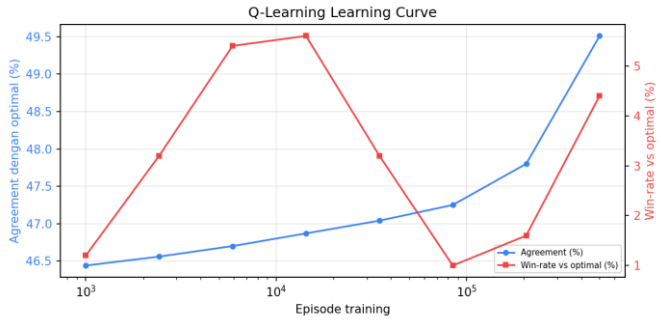

Fig. 3. Q-Learning Learning Curve (agreement + win rate vs episode)

The Q-Learning results show limited learning progress in the current generated outputs. In Fig. 1, policy agreement with the VI policy increases from 46.44% at 1.000 episodes to 49.51% at 500,000 episodes. This upward trend indicates that the learned action preferences move slightly closer to the VI policy as training increases. However, the magnitude of the improvement remains small, and the win-rate against the optimal VI policy remains low and unstable across checkpoints.

At the final recorded checkpoint of 500,000 episodes, Q-Learning reaches 49.51% agreement with the VI policy and a 4.4% win-rate against the VI policy. Earlier checkpoints show win-rates between 1.0% and 5.6%. Because these evaluations use finite match samples, some variation is expected; nevertheless, the overall pattern does not show a competitive policy emerging within the recorded training range. This is consistent with the tournament output, where the cached Q-Learning policy achieved only a 1.07% overall win-rate.

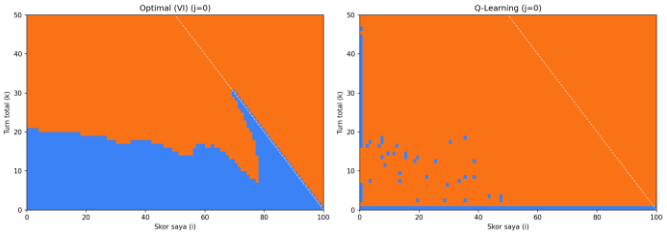

Fig. 4. VI vs. Q-Learning policy heatmap j=0

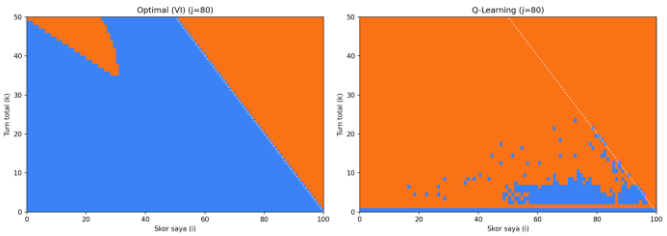

Fig. 5. VI vs. Q-Learning policy heatmap j=80

The policy heatmaps provide a qualitative explanation for this weak performance. In Fig. 4 and Fig. 5, the VI policy exhibits structured roll and hold regions that vary with the active score, opponent score, and turn total. By contrast, the Q-Learning heatmaps show a policy that holds across most non-terminal states, with only sparse roll regions. A direct comparison of the cached Q-table with the VI policy confirms this observation: over non-terminal states, the cached Q policy agrees with VI on 47.08% of states and selects hold in 97.22% of non-terminal states. Such a policy is overly conservative and frequently banks very small turn totals, which explains why it performs poorly against both the optimal and greedy baselines.

These results illustrate the difference between solving the known MDP model and learning from sampled experience. Value Iteration uses the exact transition probabilities of the die and propagates values through the entire state space. Q-Learning, in contrast, must estimate action values from sampled self-play trajectories. The current experimental evidence shows that, under the saved training regime, the Q-table does not yet approximate the Bellman-optimal policy closely enough to compete with either VI or the simple greedy heuristics.

D. Policy Structure and Greedy Failure Modes

The VI heatmaps show that the optimal policy cannot be fully described by a single fixed turn-total threshold. When the opponent score is low, as in Fig. 4, the optimal roll/hold boundary resembles a threshold near the common hold-at-20 heuristic for low active scores. For example, the VI policy first holds at $k = 21$ for state slice $i = 0, j = 0$, and at $k = 19$ for $i = 20, \ j = 0$. This explains why hold-at-20 performs reasonably well: in many early-game states, it approximates the optimal boundary.

However, the optimal threshold changes substantially as the score state changes. When the opponent score is higher, the VI policy becomes more aggressive. For example, at $i = 0$, the first hold threshold rises from $k = 21$ when $j = 0$ to $k = 49$ when $j = 80$. This pattern is visible in the heatmap slices and follows directly from the MDP formulation: when the active player is far behind, ending the turn with a small gain may leave the opponent close to winning, so rolling longer can have greater expected value despite the bust risk.

The heatmaps also show special endgame behavior. In several high-score states, the VI policy continues rolling in non-terminal states rather than holding a small turn total. For example, for $i = 80$and $j = 0$, no non-terminal hold threshold appears before reaching the terminal boundary in the saved VI table. This does not mean that the policy refuses to win; terminal states satisfying $i + k \geq 100$ are automatically winning states. Rather, it means that before the turn total is large enough to end the game, the optimal action may still be to roll. This illustrates that a fixed threshold can fail because it does not distinguish between banking a modest gain and continuing toward an immediate terminal win.

Overall, the experimental evidence supports three conclusions. First, Value Iteration provides the best policy among the tested strategies because it directly solves the Bellman optimality equation for the formulated MDP. Second, the greedy policies are strong baselines because their thresholds approximate the optimal boundary in many common states, but they lose to VI because they cannot fully adapt to opponent score and endgame structure. Third, the current Q-Learning output is not yet competitive; its learned policy is dominated by

excessive holding and has not converged sufficiently toward the VI policy in the available experiments.

ACKNOWLEDGMENT

The author would like to express sincere gratitude to Mr. Rinaldi as the lecturer of IF2211 Algorithm Strategy and teaching assistants for their guidance, feedback, and support throughout the journey of the course which led the author to finish this project. Their instruction and insights were invaluable in developing the concepts and implementation presented in this paper. The author would also like to mention the fellow Information System and Technology'23 students whom also take this course and has supported author throughout the semester.

REFERENCES

For papers published in translation journals, please give the English citation first, followed by the original foreign-language citation [6].

[1] T. W. Neller and C. G. M. Presser, "Optimal Play of the Dice Game Pig," *The UMAP Journal*, vol. 25, no. 1, pp. 25-47, 2004.

[2] M. L. Puterman, Markov Decision Processes: Discrete Stochastic Dynamic Programming. New York, NY, USA: Wiley, 1994.

[3] R. Bellman, Dynamic Programming. Princeton, NJ, USA: Princeton University Press, 1957.

[4] R. S. Sutton and A. G. Barto, Reinforcement Learning: An Introduction, 2nd ed. Cambridge, MA, USA: MIT Press, 2018.

[5] C. J. C. H. Watkins and P. Dayan, "Q-learning," Machine Learning, vol. 8, no. 3–4, pp. 279–292, 1992.

[6] R. Munir, "Algoritma Greedy (Bagian 1)," Bahan Kuliah IF2211 Strategi Algoritma, Program Studi Teknik Informatika, Sekolah Teknik Elektro dan Informatika, Institut Teknologi Bandung, Bandung, Indonesia, 2026.

APPENDIX

- Github repository: https://github.com/ielita/dice-pig-mdp.git

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 19 Juni 2026

Ni Made Sekar Jelita Parameswari (18223101)