Mid-Term Report: MDPs and their Applications in AI

Tanay Jha
Student ID: 24B1040
Mentor: Harshitha Inampudi

June 15, 2025

Contents

1 Introduction

Reinforcement Learning (RL) is a dynamic field of machine learning focused on enabling agents to learn optimal sequences of decisions through interaction with an environment. This mid-term report details the foundational knowledge assimilated and the initial prac tical implementations undertaken during the Summer of Science program. The work spans from fundamental mathematical constructs, crucial for understanding the under pinnings of intelligent systems, to core RL concepts and initial solution implementations that demonstrate these principles in action.

The report is structured to reflect a progressive understanding, building from basic formalisms to complex decision-making frameworks:

1. Formal Foundations: Logic and Automata Theory: This section revisits formal systems for computation and state representation. These logical structures provide the essential tools for precisely defining states, transitions, and the rules governing system behavior, which are fundamental to any computational model of learning.

2. Stochastic Processes: Markov Chains (MCs): The report then explores sys tems with probabilistic state transitions that adhere to the Markov property. Un derstanding MCs is vital as they form the basis for modeling environments where outcomes are uncertain but dependably linked to the current state.

3. Introduction to Reinforcement Learning: K-Armed Bandits: Before tack ling full state-based RL, this section examines the K-Armed Bandit problem. It serves as a simplified context to explore the critical exploration-exploitation dilemma through the implementation and analysis of various heuristic algorithms.

4. The Core RL Framework: Markov Decision Processes (MDPs): Finally, the report delves into MDPs, detailing their components, the crucial roles of policies and value functions, the foundational Bellman equations, and the implementation of exact solution algorithms like Value Iteration and Policy Iteration. These methods are exemplified by their application to the classic Grid World problem and the Gambler's Problem.

This report aims to summarize the theoretical learning, document the practical imple mentations (with references to the project's code repository [1]), candidly discuss the challenges encountered, and articulate the key insights gained, all in alignment with the initial Plan of Action (PoA) and the subsequent progress achieved.

2 Formal Foundations: Logic and Automata

Understanding computation, state representation, and transitions is fundamental to grasp ing how an RL agent perceives its environment and makes decisions. This section revisits core concepts from propositional logic and finite automata, which provide the building blocks for more complex models of sequential decision-making.

2.1 Propositional Logic

Propositional logic is the simplest form of logic and provides the basic tools for formal reasoning about statements that can be definitively determined as either true or false. It is foundational for fields like AI, database theory, and circuit design.

- **Syntax & Semantics:** The syntax defines well-formed formulas (WFFs) using propositional variables (e.g., P, Q, R, representing atomic statements like "It is rain ing") and logical connectives such as ¬ (negation, "not"), (conjunction, "and"), (disjunction, "or"), → (implication, "if...then..."), and (biconditional, "if and only if"). The semantics are given by truth assignments, which map variables to truth values True, False. The truth value of a complex formula is determined composi- tionally using truth tables for these connectives.

- **Normal Forms:** Any propositional formula can be transformed into equivalent standard or normal forms. These are particularly useful for simplifying formulas, checking equivalences, and for algorithms in automated reasoning (e.g., SAT solvers often require CNF).

  - Conjunctive Normal Form (CNF): A conjunction of one or more clauses, where each clause is a disjunction of literals (a variable or its negation). E.g., (P ¬Q) (¬P R).
  - Disjunctive Normal Form (DNF): A disjunction of one or more terms, where each term is a conjunction of literals. E.g., (P ¬Q) (¬P R).

- **Key Concepts:**

  - Tautology: A formula true for all truth assignments (e.g., P¬P), representing a universally valid logical truth.
  - Contradiction: A formula false for all truth assignments (e.g., P ¬P), rep resenting a logical impossibility.
  - Satisfiability (SAT): A formula is satisfiable if there exists at least one truth assignment making it true. The SAT problem, determining satisfiability, is a central NP-complete problem with wide applications.

2.2 Finite Automata (FA)

Finite Automata are mathematical models of computation that accept or reject strings of symbols based on a predefined set of rules and a finite number of states. They are instrumental in pattern recognition, lexical analysis, and modeling systems with discrete states and transitions.

- **Deterministic Finite Automaton (DFA):** A DFA processes input strings de terministically: for each state and input symbol, the next state is unique. Formally, a DFA M = (Q, , , q0, F) comprises:
  - Q: A finite set of states, representing the machine's memory.
  - : A finite input alphabet.
  - : Q × → Q: The transition function, mapping a state and input symbol to a next state.
  - q0 Q: The designated initial state.
  - F Q: A set of accept or final states.

  A string is accepted if, starting from q0 and following transitions dictated by for each symbol in the string, the DFA ends in a state qf F. DFAs recognize the class of regular languages.

- **Nondeterministic Finite Automaton (NFA):** NFAs generalize DFAs by al lowing multiple possible next states for a given state-input pair and by permitting -transitions (state changes without consuming an input symbol). The transition function is : Q × ( ) → P(Q) (the power set of Q). An NFA accepts a string if *any* sequence of valid transitions leads to an accept state. While NFAs can be more concise for describing certain languages, they do not possess more computational power than DFAs.

- **Equivalence of DFA and NFA:** For every NFA, an equivalent DFA (accepting the same language) can be constructed using the "subset construction" algorithm. This DFA's states correspond to sets of NFA states. This equivalence is a cornerstone of automata theory.

- **Regular Expressions (Regex):** A powerful, algebraic way to describe patterns in strings, defining regular languages. They are built from alphabet symbols using operations like concatenation, union, and Kleene star.

- **Kleene's Theorem:** This fundamental theorem states that a language is regular if and only if it can be described by a regular expression, which also means it is accepted by some FA (DFA or NFA). It unifies these three formalisms for describing regular languages.

2.3 Relevance to Sequential Decision-Making

Automata theory, particularly its concepts of states and transitions, serves as a conceptual precursor to the more complex framework of MDPs used in RL.

- **State Representation:** States in FAs model the "memory" of past inputs neces sary to decide on future transitions or acceptance. This is analogous to states in RL, which aim to capture all relevant information from the environment's history to make optimal decisions (the Markov property).
- **Transition Dynamics:** Transitions in FAs are driven by input symbols. In MDPs, transitions are driven by agent actions and are often probabilistic, reflecting the stochastic nature of many real-world environments.
- **Goal Achievement:** The notion of an FA "accepting" a string by reaching a final state parallels the RL agent's goal of reaching desirable terminal states or achieving high cumulative rewards through a sequence of actions.

Both formalisms model sequential processes, providing a structured way to think about how systems evolve over time based on events or interventions.

3 Stochastic Processes: Markov Chains

Markov Chains (MCs) are stochastic models describing a sequence of possible events (states) where the probability of transitioning to any future state depends only on the current state, not on the sequence of events that preceded it—this is the defining Markov property. They are essential for modeling systems with inherent randomness and time evolution.

3.1 Definition and The Markov Property

A discrete-time Markov Chain (DTMC) is a sequence of random variables Xt : t = 0, 1, 2, . . . taking values in a countable state space S. Its core characteristic is the Markov Property:

$$\mathbb{P}(X_{t+1} = s_{t+1}|X_t = s_t, Xt1 = s_{t1}, ..., X_0 = s_0) = \mathbb{P}(X_{t+1} = s_{t+1}|X_t = s_t)$$

This "memorylessness" means the future is conditionally independent of the past given the present state. For a time-homogeneous DTMC, the one-step transition probabilities $P_{ij} = \mathbb{P}(X_{t+1} = s_j|X_t = s_i)$ are constant over time and form the transition probability matrix P. Each row of P sums to 1.

3.2 Classification of States

The long-term behavior of an MC is determined by the properties of its states:

- **Accessibility and Communication:** State s_j is accessible from s_i if $P_{ij}^{(n)} > 0$ for some $n \geq 0$. States s_i and s_j communicate if they are accessible from each other. Communication forms equivalence classes, partitioning S. An MC is irreducible if all states form a single communicating class.

- **Recurrence and Transience:** A state s_i is recurrent if, starting from s_i, the probability of eventually returning to s_i is 1; otherwise, it is transient. Recurrence means the process will visit the state infinitely often if the class is re-entered. Transience implies the process eventually leaves the state permanently. This is a class property.

- **Periodicity:** The period $d(s_i)$ of a state s_i is the greatest common divisor (GCD) of all $n \geq 1$ such that $P_{ii}^{(n)} > 0$. If $d(s_i) = 1$, the state is aperiodic. A state being aperiodic is crucial for convergence to a limiting distribution. This is also a class property.

- **Ergodic MC:** An MC is ergodic if it is irreducible, positive recurrent (all states are positive recurrent, meaning expected return time is finite), and aperiodic. Ergodic MCs have strong, predictable long-term behavior.

3.3 Long-Term Behavior and Stationary Distributions

For ergodic MCs, the distribution over states converges to a unique stationary distribu tion π, regardless of the initial state. This distribution satisfies $\pi P = \pi$ and $\sum_j \pi_j = 1$. The component π_j represents the long-run proportion of time the chain spends in state s_j. This predictability is vital for analyzing systems that run for extended periods.

3.4 Absorbing Markov Chains

An absorbing state s_i is one where $P_{ii} = 1$ (once entered, never left). An MC is absorbing if it has at least one absorbing state, and every non-absorbing (transient) state can reach an absorbing state. Analyzing absorbing MCs often involves calculating:

- The probability of being absorbed in a specific absorbing state.

- The expected number of steps until absorption.

These are found using the canonical form of $P = \begin{pmatrix} Q & R \\ 0 & I \end{pmatrix}$ and the fundamental matrix $N = (I - Q)^{-1}$. The matrix $B = NR$ gives absorption probabilities.

4 Introduction to Reinforcement Learning: K-Armed Bandits

The K-Armed Bandit problem serves as a simplified, yet insightful, introduction to rein forcement learning, isolating the critical challenge of balancing exploration of new options against exploitation of known good options.

4.1 Problem Definition

The agent is presented with K choices or "arms" (like slot machines). Each arm a, when selected (or "pulled"), yields a reward drawn from an unknown probability distribution specific to that arm. The agent's objective is to make a sequence of arm selections over time to maximize its total accumulated reward. The true expected reward of arm a is $q_*(a) = \mathbb{E}$[Reward from arm a]. Since $q_*(a)$ is unknown, the agent must estimate it, typically as $Q_t(a)$, the sample average of rewards received from arm a up to time t 1.

4.2 The Exploration-Exploitation Dilemma

At each step, the agent faces a dilemma:

- **Exploitation:** Choose the arm with the highest current estimated value $Q_t(a)$. This maximizes expected immediate reward based on current knowledge but risks missing out on a truly better arm whose value is currently underestimated.
- **Exploration:** Choose an arm that is not currently estimated to be the best. This may yield a lower immediate reward but provides more information, potentially improving future estimates and leading to higher overall reward.

Effective bandit algorithms must intelligently manage this trade-off to minimize long term "regret" (the difference between rewards obtained and rewards that could have been obtained by always picking the best arm).

4.3 Algorithms Implemented

Several algorithms were implemented to address this dilemma (code at [1]):

- **Greedy Algorithm:** Always selects $A_t = \arg\max_a Q_t(a)$. This purely exploits and can easily get stuck with suboptimal arms.
- **ϵ-Greedy Algorithm:** Exploits with probability $1 - \epsilon$ and explores (chooses a random arm) with probability ϵ. This ensures continued exploration. ϵ can be fixed or decay over time.

- **Upper-Confidence-Bound (UCB):** Selects $A_t = \arg\max_a \left[Q_t(a) + c\sqrt{\frac{\ln t}{N_t(a)}} \right]$. The second term is an "uncertainty bonus" that encourages exploration of less tried arms or arms whose estimates are less certain. $N_t(a)$ is the count of pulls for arm a.

- **Gradient Bandit Algorithm:** Learns preferences $H_t(a)$ for each arm. Actions are selected via a softmax distribution $\pi_t(a) \propto e^{H_t(a)}$. Preferences are updated using stochastic gradient ascent, $H_{t+1}(A_t) = H_t(A_t) + \alpha(R_t - \bar{R}_t)(1 - \pi_t(A_t))$, where $\bar{R}_t$ is a reward baseline.

- **Optimistic Initial Values:** Initializes all $Q_0(a)$ to a value higher than any possible true reward. This encourages systematic exploration of all arms initially, as pulling an arm tends to reduce its overestimated value.

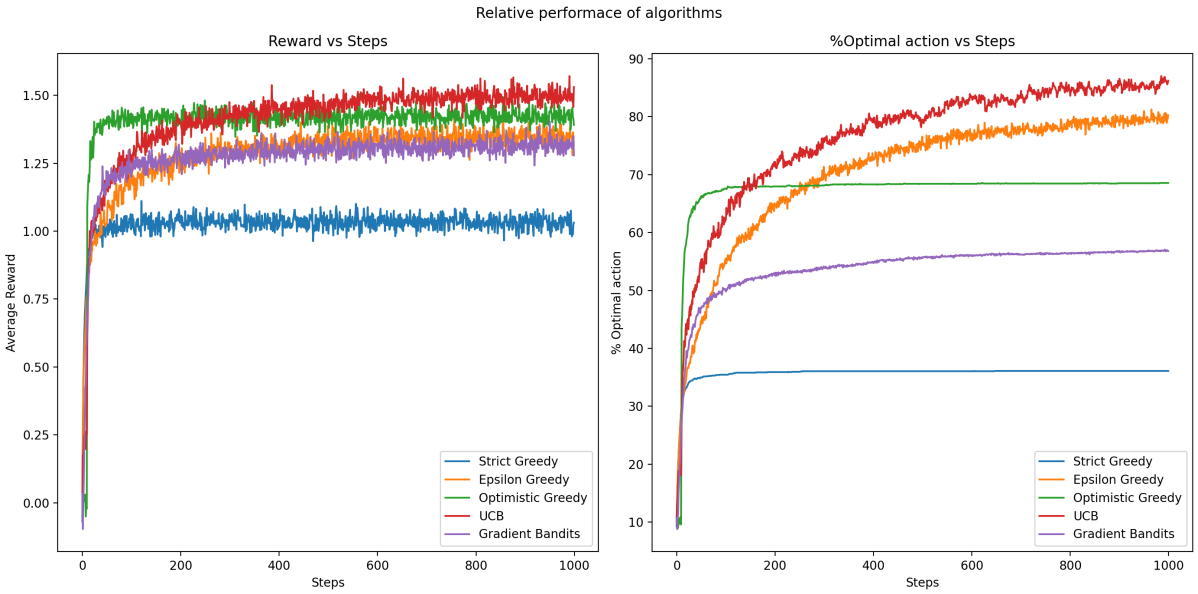

Figure 1: Various Bandit algorithms on the 10-armed testbed. Performance comparison showing average reward and optimal action percentage over steps.

4.4 Key Learnings from Bandit Problems

These implementations were crucial for understanding:

- The fundamental nature of the exploration-exploitation trade-off.

- How different algorithmic strategies (randomness, optimism, uncertainty quantifi cation, preference learning) address this trade-off.

- The sensitivity of algorithm performance to hyperparameters (e.g., ϵ, c, α).

- The bandit problem as a foundational stepping stone to more complex RL problems involving states, where similar value estimation and action selection strategies are employed.

5 The Core RL Framework: Markov Decision Processes

Markov Decision Processes (MDPs) provide a formal mathematical framework for mod eling sequential decision-making in environments where outcomes are partly random and partly under the control of an agent. They are the standard model for most problems in reinforcement learning.

5.1 MDP Components

An MDP is defined by a tuple $(\mathcal{S}, \mathcal{A}, \mathcal{P}, \mathcal{R}, \gamma)$:

- $\mathcal{S}$: A finite set of states.
- $\mathcal{A}$: A finite set of actions.
- $\mathcal{P}(s'|s,a)$: The state transition probability function, $\mathbb{P}(S_{t+1} = s'|S_t = s, A_t = a)$.
- $\mathcal{R}(s,a,s')$: The reward function, giving the immediate reward for the transition.
- $\gamma \in [0,1]$: The discount factor, valuing immediate rewards more than distant future rewards. It ensures bounded returns in infinite horizons and models preference for quicker gratification.

5.2 Policies and Value Functions

- **Policy $\pi(a|s)$:** The agent's strategy, $\mathbb{P}(A_t = a|S_t = s)$. A deterministic policy is a special case.
- **State-Value Function $V^\pi(s)$:** The expected discounted sum of future rewards (return) starting from state s and following policy π : $V^\pi(s) = \mathbb{E}_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_t = s]$. It quantifies the long-term desirability of state s under π.
- **Action-Value Function $Q^\pi(s,a)$:** The expected return starting from state s, taking action a, and then following policy π : $Q^\pi(s,a) = \mathbb{E}_\pi[\sum_{k=0}^{\infty} \gamma^k R_{t+k+1}|S_t = s, A_t = a]$. It quantifies the desirability of taking action a in state s when following π. The relationship is $V^\pi(s) = \sum_a \pi(a|s)Q^\pi(s,a)$.

5.3 Bellman Equations

Bellman equations provide recursive decompositions of the value functions, forming the basis for most RL algorithms. They relate the value of a state (or state-action pair) to the values of its potential successor states.

5.3.1 Bellman Expectation Equation

For a given policy π, the value functions satisfy:

$$V^{\pi}(s) = \sum_{a \in \mathcal{A}} \pi(a|s) \sum_{s' \in \mathcal{S}} \mathcal{P}(s'|s,a)[\mathcal{R}(s,a,s') + \gamma V^{\pi}(s')] \tag{1}$$

$$Q^{\pi}(s,a) = \sum_{s' \in \mathcal{S}} \mathcal{P}(s'|s,a)[\mathcal{R}(s,a,s') + \gamma V^{\pi}(s')] \tag{2}$$

Equation 1 states that the value of s is the expected immediate reward plus the dis counted expected value of the next state, averaged over actions taken by π and resulting transitions. Equation 2 has a similar interpretation for taking a specific action a.

5.3.2 Bellman Optimality Equation

For the optimal policy π^*, which maximizes value from all states, the optimal value functions V^* and Q^* satisfy:

$$V^*(s) = \max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} \mathcal{P}(s'|s,a)[\mathcal{R}(s,a,s') + \gamma V^*(s')] \tag{3}$$

$$Q^*(s,a) = \sum_{s' \in \mathcal{S}} \mathcal{P}(s'|s,a)[\mathcal{R}(s,a,s') + \gamma \max_{a' \in \mathcal{A}} Q^*(s',a')] \tag{4}$$

Equation 3 implies $V^*(s)$ is achieved by picking the best action a in state s and then con tinuing optimally. The non-linearity due to the ‘max‘ operator means these are generally solved iteratively.

5.4 Solving MDPs: Exact Methods (Dynamic Programming)

When the MDP model ($\mathcal{P}$ and $\mathcal{R}$) is known, dynamic programming (DP) finds optimal policies.

5.4.1 Value Iteration (VI)

VI iteratively applies the Bellman optimality update (Equation 3) to an estimate $V_k(s)$, starting with an arbitrary V_0. Each iteration involves a "backup" operation for every state:

$$V_{k+1}(s) \leftarrow \max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} \mathcal{P}(s'|s,a)[\mathcal{R}(s,a,s') + \gamma V_k(s')]$$

The sequence V_k converges to V^* because the Bellman optimality operator is a γ-contraction mapping. Once V^* is approximated, the optimal policy is extracted greedily.The pseudocode is shown in Algorithm 1.

Algorithm 1 Value Iteration

Initialize $V_0(s)$ arbitrarily for all $s \in \mathcal{S}$ (e.g., $V_0(s) = 0$), $k \leftarrow 0$
repeat
    $\Delta \leftarrow 0$
    for each state $s \in \mathcal{S}$ **do do**
        $v \leftarrow V_k(s)$
        $V_{k+1}(s) \leftarrow \max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} \mathcal{P}(s'|s,a)[\mathcal{R}(s,a,s') + \gamma V_k(s')]$
        $\Delta \leftarrow \max(\Delta, |v - V_{k+1}(s)|)$
    $V_k \leftarrow V_{k+1}$
    $k \leftarrow k + 1$
until $\Delta < \theta$ (convergence threshold)
return V_k (approximation of V^*)
Output policy $\pi^*(s) = \arg\max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} \mathcal{P}(s'|s,a)[\mathcal{R}(s,a,s') + \gamma V^*(s')]$

5.4.2 Policy Iteration (PI)

PI alternates between two steps until the policy stabilizes:

1. **Policy Evaluation:** For the current policy π_k, compute V^{π_k} by solving the linear system of Bellman expectation equations (Equation 1). This is often done itera tively: $V_{j+1}^{\pi_k}(s) \leftarrow \sum_{s'} \mathcal{P}(s'|s,\pi_k(s))[\mathcal{R}(s,\pi_k(s),s') + \gamma V_j^{\pi_k}(s')]$ until convergence.

2. **Policy Improvement:** Improve π_k by acting greedily with respect to V^{π_k} : $\pi_{k+1}(s) = \arg\max_a Q^{\pi_k}(s,a) = \arg\max_a \sum_{s'} \mathcal{P}(s'|s,a)[\mathcal{R}(s,a,s') + \gamma V^{\pi_k}(s')]$.

If $\pi_{k+1} \neq \pi_k$, then $V^{\pi_{k+1}}(s) \geq V^{\pi_k}(s)$ for all s (Policy Improvement Theorem). PI converges to π^* and V^*. The pseudocode is in Algorithm 2. While often converging in fewer iterations than VI, each PI iteration (policy evaluation) can be more costly.

Algorithm 2 Policy Iteration

Initialize policy $\pi(s)$ arbitrarily for all $s \in \mathcal{S}$
Initialize $V(s)$ arbitrarily for all $s \in \mathcal{S}$
repeat
    // 1. Policy Evaluation
    repeat
        $\Delta \leftarrow 0$
        for each state $s \in \mathcal{S}$ **do do**
            $v \leftarrow V(s)$
            $V(s) \leftarrow \sum_{s' \in \mathcal{S}} \mathcal{P}(s'|s, \pi(s))[\mathcal{R}(s, \pi(s), s') + \gamma V(s')]$
            $\Delta \leftarrow \max(\Delta, |v - V(s)|)$
    until $\Delta < \theta_{eval}$
    // 2. Policy Improvement
    policy_stable $\leftarrow$ true
    for each state $s \in \mathcal{S}$ **do do**
        old_action $\leftarrow \pi(s)$
        $\pi(s) \leftarrow \arg\max_{a \in \mathcal{A}} \sum_{s' \in \mathcal{S}} \mathcal{P}(s'|s, a)[\mathcal{R}(s, a, s') + \gamma V(s')]$
        if *old_action* $\neq \pi(s)$ **then**
            policy_stable $\leftarrow$ false
until *policy_stable*
return π and V

5.5 Applications of Dynamic Programming in MDPs

The theoretical DP methods find practical application in solving well-defined MDPs. Two classic examples were explored:

5.5.1 The Grid World Problem (Solved with Policy Iteration)

The Grid World is a standard testbed for illustrating MDP concepts and algorithms. It's a 2D grid where each cell is a state.

- **Setup:** Defined by grid dimensions, cell types (normal, obstacle, terminal goal, terminal penalty), available actions (typically Up, Down, Left, Right), transition probabilities (deterministic or stochastic, e.g., 80% chance of intended move, 10% chance to either perpendicular side), and rewards (e.g., small negative per step, large positive for goal, large negative for penalty).

- **Solution with Policy Iteration: Policy Iteration was implemented to find optimal policies.** This involved defining the MDP components for the Grid World and then applying the PI algorithm. The practical exercise of coding these, debugging transitions (es pecially at boundaries or near obstacles), and observing the propagation of values and convergence of policies was highly instructive. The resulting optimal policy can be visualised as a vector field guiding the agent. The

light blue squares denote the paths, the dark blue squares denote walls and the white square denotes the goal (Code at [1]).

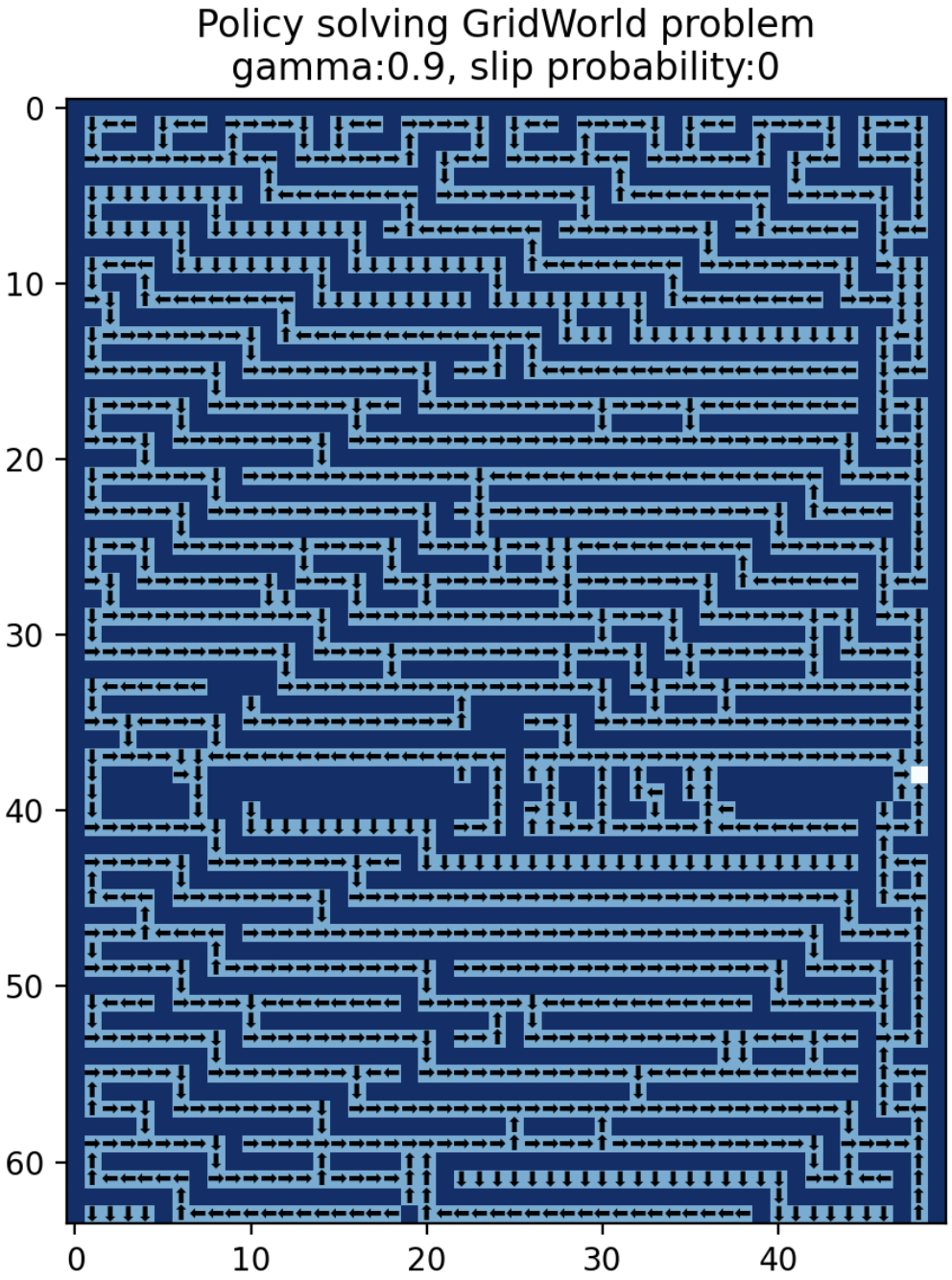

Figure 2: An optimal Grid World policy showing states, actions, and a goal, derived using Policy Iteration

5.5.2 The Gambler's Problem (Solved with Value Iteration)

Another classic MDP, described in Sutton and Barto [6], is the Gambler's Problem.

- **Setup:** A gambler has an initial capital and wants to reach a goal capital (e.g., \$100) by making a series of bets. On each bet, the gambler can stake an integer amount of money. With probability p_h the gambler wins and receives an amount equal to the stake; with probability $1 - p_h$, the gambler loses the stake. The game ends when the gambler reaches the goal capital or runs out of money (capital becomes \$0).
- **MDP Formulation:**
  - States $\mathcal{S}$: The gambler's capital, $\{0, 1, \ldots, \text{GoalCapital}\}$. States 0 and GoalCapital are terminal.
  - Actions $\mathcal{A}_s$: The possible stakes the gambler can bet in state s (current capital), e.g., $\{1, 2, \ldots, \min(s, \text{GoalCapital} - s)\}$.
  - Transition Probabilities $\mathcal{P}$: Determined by p_h. If stake a is bet in state s:
    - Transition to $s + a$ with probability p_h.
    - Transition to $s - a$ with probability $1 - p_h$.
  - Rewards $\mathcal{R}$: Typically, $+1$ upon reaching GoalCapital, and 0 for all other transitions (or small negative cost per bet).
- **Solution with Value Iteration: Value Iteration was implemented to determine the optimal betting policy.** This involves finding the optimal value function $V^*(s)$ (maximum probability of reaching the goal from capital s) and the corresponding optimal stake $\pi^*(s)$ for each capital level. The results show interesting patterns in betting strategy depending on p_h and proximity to the goal or ruin. (Code at [1]).

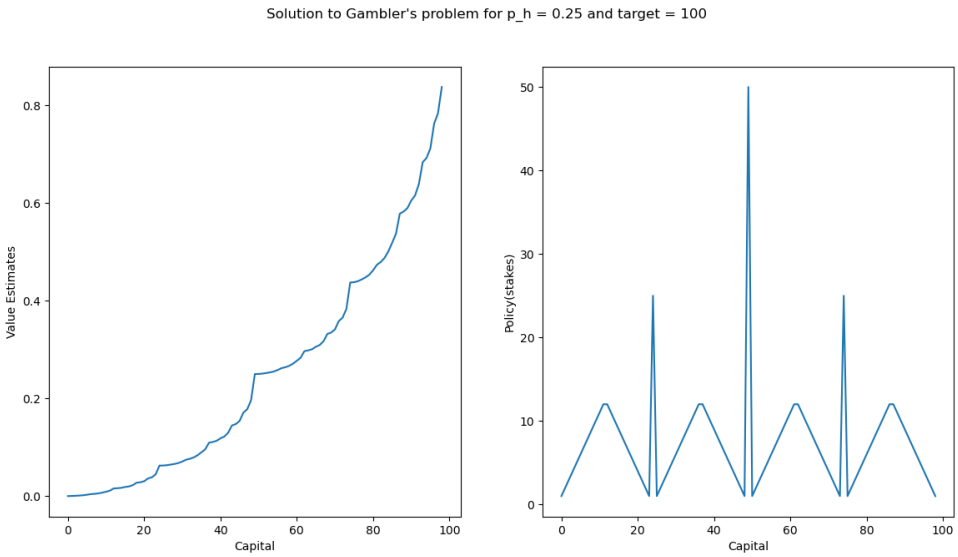

Figure 3: Optimal policy and value function for the Gambler's Problem (e.g., target=\$100, p_h=0.25).

6 Progress, Challenges, and Learnings

6.1 Progress Summary

The project has systematically progressed through foundational topics and initial imple mentations, building a strong base for advanced RL studies:

- **Weeks 1-2 (Formal Systems & Stochastic Processes):** Developed a solid un derstanding of Propositional Logic, Finite Automata (DFA, NFA, Regex, Kleene's Theorem), and Markov Chains (Markov Property, state classification, long-term be havior, absorbing chains). The relevance of these formalisms to modeling stateful, transitional systems was clearly established.
- **Week 3 (Intro to RL & MDPs):**
  - Explored K-Armed Bandit problems, implementing and comparing various al gorithms (Greedy, ϵ-Greedy, UCB, Gradient Bandit, Optimistic Initialization). This provided critical hands-on experience with the exploration-exploitation trade-off.
  - Mastered core MDP components (states, actions, transitions, rewards, dis count factor), the roles of policies (deterministic/stochastic), and value func tions (V^π, Q^π). A deep understanding of the Bellman expectation and opti mality equations was achieved.
- **Weeks 4-5 (MDP Solutions and Midterm Prep):**
  - Successfully implemented the **Value Iteration algorithm for solving the Gambler's Problem**.
  - Successfully implemented the **Policy Iteration algorithm for solving known MDPs, specifically applied to various Grid World problem configura tions**. This involved translating the theoretical Bellman updates into working Python code.
  - Consolidated all learning from the preceding weeks and prepared this Midterm Report to document progress, challenges, and key insights.

Implementations for Bandits, Gambler's Problem (using VI), and Grid World (using PI) solutions are version-controlled and available at [1].

6.2 Challenges Faced

While progress has been steady, several intellectual and practical challenges were encoun tered:

- **Conceptual Depth and Mathematical Nuances:**
  - Understanding the meaning of Automata, its applications and relations to MCs and MDPs. The importance of rigorous definitions and propositional logic in defining Automata and MCs.
  - Fully internalizing the conditions for the existence and uniqueness of stationary distributions in Markov Chains, and the subtle distinctions between different types of state recurrence.
- **Implementation of Dynamic Programming Algorithms:**
  - Debugging Iterative Processes: Ensuring correct loop termination conditions and value updates in VI and PI . For PI, a key challenge was ensuring the policy evaluation step converged sufficiently (enough inner iterations) before proceeding to policy improvement, to guarantee overall convergence.
  - State and Action Space Management: Efficiently representing and iterating over states and actions in code for Grid Worlds and Gambler's Problem, especially when handling boundary conditions or variable action sets.
  - Translating Equations to Code: Accurately translating the summations and maximizations in Bellman equations into Python code required meticulous attention to indexing and variable scopes.
- **Learning rate, iterations, and computing time tradeoff:** Balancing the learn ing rate and computing resources to achieve acceptable results without too much training time.
- **Bandit Algorithm Parameter Sensitivity:** Experimentally tuning parameters like ϵ (for ϵ-greedy), c (for UCB), and α (step-size for gradient bandits) and ob serving their significant impact on learning curves and final performance was an insightful, albeit sometimes time-consuming, process.

6.3 Key Learnings and Insights

This initial phase has yielded several profound learnings that form a strong foundation for future work in RL:

- **Interconnectedness of Concepts:** A clear understanding emerged of how foun dational theories—Logic/Automata (state representation, discrete transitions), Markov Chains (stochastic transitions, Markov property), K-Armed Bandits (exploration/exploitation fundamentals)—synergistically build up to the comprehensive framework of MDPs for full decision-making.

- **Practicality and Power of Dynamic Programming:** Implementing VI and PI transformed abstract Bellman equations into tangible computational tools. Witnessing their ability to find optimal solutions in model-based settings underscored their power and foundational role in RL theory.
- **The Critical Role of Exploration Strategy:** The K-Armed Bandit experiments were particularly illuminating in demonstrating that naive exploitation (greedy ap proach) is often insufficient. Sophisticated and principled exploration is vital for robust learning and achieving optimal long-term rewards.
- **Iterative Refinement as a Core RL Theme:** Many RL algorithms, including those studied so far (VI, PI, iterative policy evaluation within PI, and many bandit updates), rely on iterative refinement of value estimates or policies. This theme of progressive improvement through repeated computation is central to the field.
- **Value of Precise Problem Formulation:** The rigor required in defining states, actions, transition probabilities, and especially rewards in the Grid World MDP and Gambler's Problem highlighted how critical a precise and well-thought-out problem formulation is for the success of any RL algorithm. Ambiguities or poorly designed rewards can easily lead to flawed or suboptimal solutions.

7 Updated Plan of Action (Summary)

The project will continue to follow the established Plan of Action, with the next phases focusing on Model-Free RL algorithms and the main coding project. The solid theoretical foundation and practical experience with model-based MDP solutions provide an excellent springboard for these upcoming topics.

- Further exploration of MDP solution nuances and related concepts (e.g., Hidden Markov Models, advanced Reward Engineering techniques).
- Delving into Model-Free Reinforcement Learning: This will include Monte Carlo Methods, and Temporal Difference Learning (SARSA, Q-Learning), with imple mentations.
- Main Coding Project: Development of an RL agent for a mini-game (e.g., sim plified Flappy Bird) using tabular Q-learning, potentially extending to Deep Q Networks (DQN) if time and progress permit.
- Conceptual introduction to advanced RL topics such as Function Approximation and Policy Gradients, to understand how RL scales to more complex problems.

A detailed updated Plan of Action, consistent with the original schedule and reflecting current progress, is submitted as a separate document.

References

[1] Markov-Decision-Processes-and-their-applications-in AI. GitHub Repository. https://github.com/Tanay2104/Markov-Decision-Processes-and-their-applications-in-AI

[2] Huth, M., & Ryan, M. (2004). *Logic in Computer Science: Modelling and Reasoning about Systems* (2nd ed.). Cambridge University Press.

[3] Baier, C., & Katoen, J.-P. (2008). *Principles of Model Checking*. MIT Press.

[4] Parker, D. Slides on Model Checking (PRISM). Available: https://www.prismmodelchecker.org/lectures/pmc/

[5] Silver, D. UCL Course on Reinforcement Learning. YouTube Playlist.

[6] Sutton, R. S., & Barto, A. G. (2018). *Reinforcement learning: An introduction* (2nd ed.). MIT Press.