Water Resource Management System

Submitted by,
Ridhin Issac Abraham (2348546)
Deyon Rose Babu Thomas (2348513)

Water Resource Management Simulation

This project implements a **Water Resource Management Simulation** using reinforcement learning principles, where the primary objective is to efficiently manage a water reservoir by balancing supply and demand. The simulation is designed to model the decision-making process for water allocation, considering various factors such as **water demand**, **reservoir capacity**, and **rainfall probability**.

The system operates in a **dynamic environment** where the user is tasked with allocating water to meet the demand for each time step. The simulation environment models the reservoir, rainfall events, and the effects of water allocation, while also providing feedback in the form of **rewards** and **penalties** based on the decisions made. These rewards are calculated based on several factors:

- **Over-allocation** of water (penalty for allocating more water than available).
- **Under-allocation** (penalty for allocating less water than required).
- **Matching the demand** (reward for meeting the demand closely).
- **Stabilizing the water level** (reward for maintaining the water level within optimal limits, especially when close to full capacity).

The project utilizes **Streamlit** for an interactive user interface, allowing users to input key simulation parameters, such as:

- **Reservoir capacity**
- **Water demand per time step**
- **Rainfall probability** and **rainfall amount**

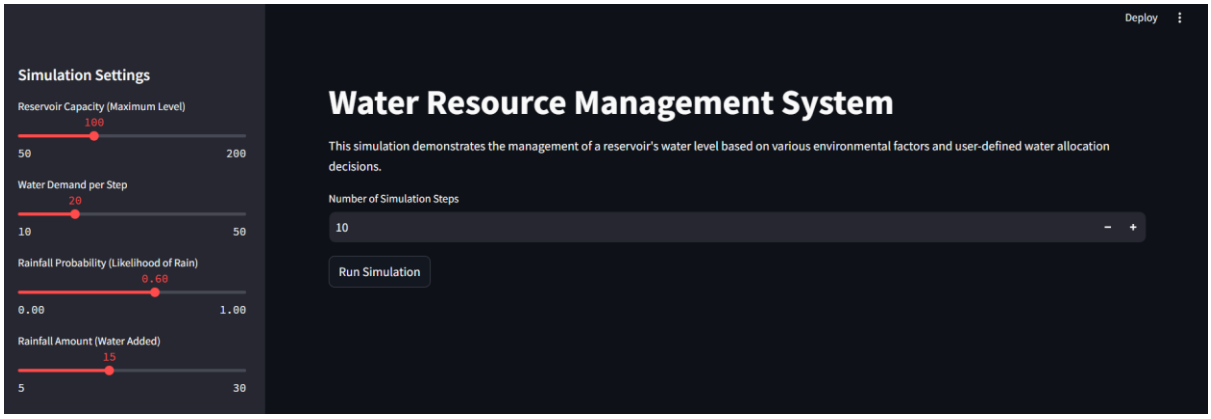
Users can run the simulation for a specified number of steps and observe how the reservoir's water level changes over time, in response to both allocated water and rainfall events. The simulation also produces visual outputs, such as:

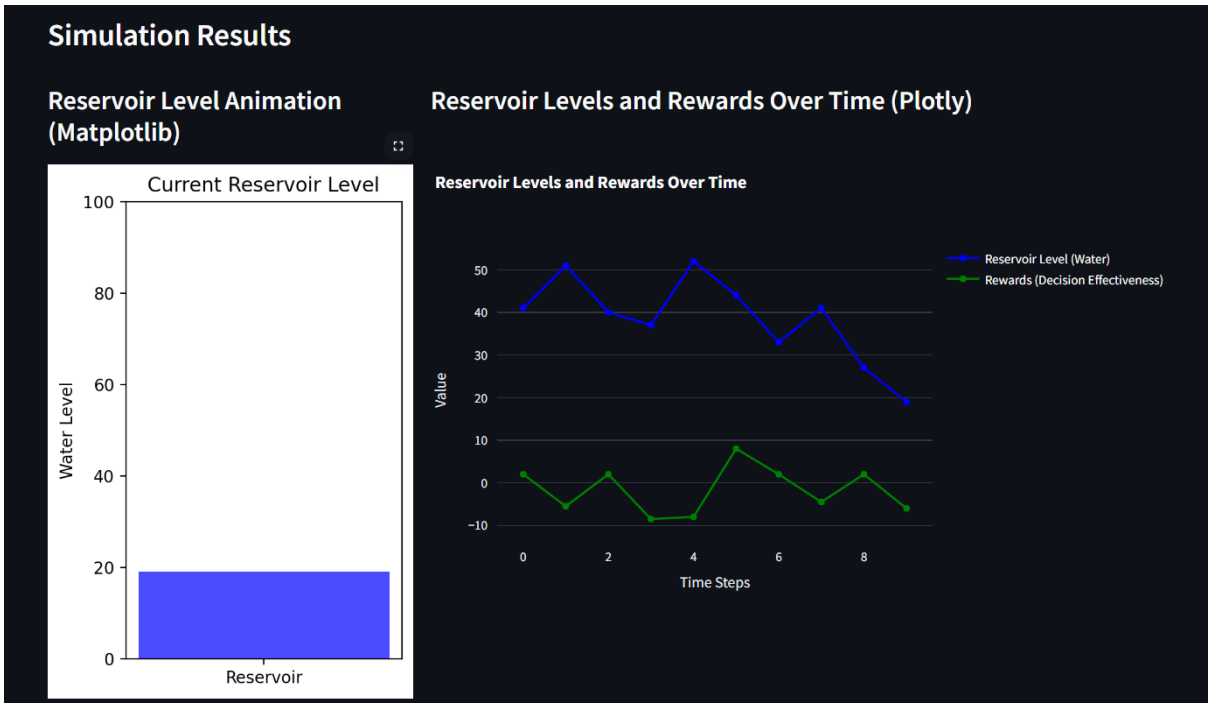
1. **Animated Bar Chart**: Shows real-time updates of the water level in the reservoir after each step of the simulation.
2. **Line Graph**: Displays the relationship between water allocation, demand, and the rewards over time using **Plotly**.
3. **Detailed Results**: Provides users with a summary of the final water level, average reward, and an interpretation of the results, including insights on successful decision-making.

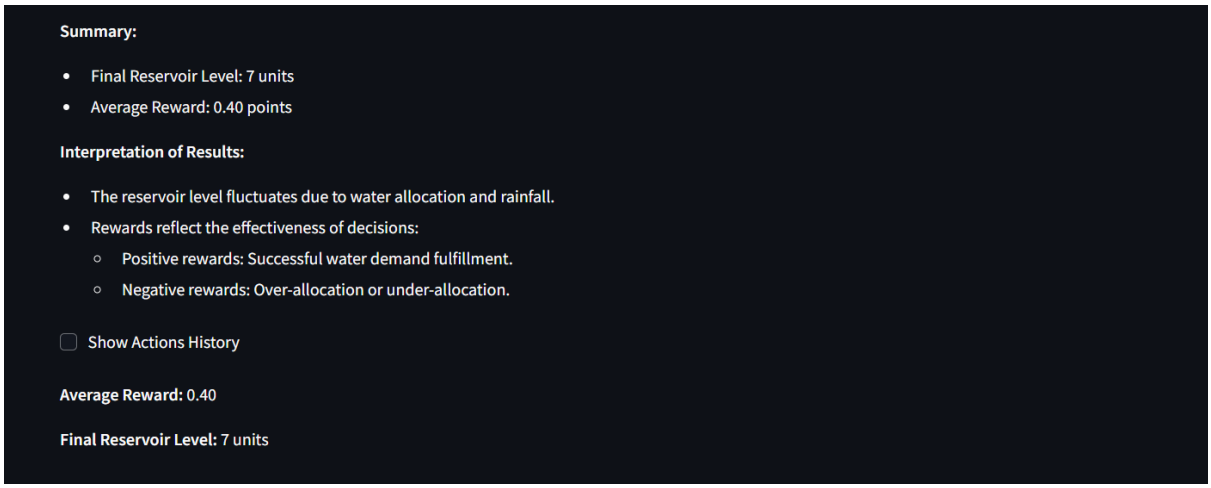
4. **Action History**: Users can view a detailed table of actions taken throughout the simulation, showcasing the choices made and their outcomes.

This project serves as an educational tool for understanding the complexities involved in water resource management, emphasizing the need for balancing water allocation to meet demand, maintaining reservoir stability, and responding to environmental changes like rainfall. The incorporation of **rewards** and **penalties** mirrors real-world constraints in water management systems, offering valuable insights into decision-making under uncertainty.

Screenshots:







Source Code:

```
import streamlit as st
import numpy as np
import random
import matplotlib.pyplot as plt
from matplotlib.ticker import MaxNLocator
import time
import plotly.graph_objects as go

# Water Resource Environment
class WaterResourceEnv:
    def __init__(self, max_capacity, demand, rainfall_prob, rainfall_amount):
        self.max_capacity = max_capacity
        self.current_level = max_capacity // 2
        self.demand = demand
        self.rainfall_prob = rainfall_prob
        self.rainfall_amount = rainfall_amount
        self.state = self.current_level

    def step(self, action):
        reward = 0

        # Check for over-allocation (action > current level)
        if action > self.current_level:
            reward -= 1  # Reduced penalty for over-allocation
            action = self.current_level  # Adjust to max available water

        # Check for under-allocation (action < demand)
        if action < self.demand:
            reward -= 0.5 * (self.demand - action)  # Smaller penalty for
under-allocation

        # Reward for being close to demand (within a range of ±3)
        if abs(action - self.demand) <= 3:
            reward += 6  # Reward for being close to the demand

        # Bonus for exact match
        if action == self.demand:
            reward += 5  # Bonus for exact match

        # Reward for maintaining a stable water level (e.g., within 80% to
100% of max capacity)
        if self.current_level >= self.max_capacity * 0.8:
            reward += 4  # Reward for keeping the reservoir between 80% and
100% capacity

        # Deduct the allocated water
        self.current_level -= action
```

```
        # Check for rainfall and add water
        if random.random() < self.rainfall_prob:
            self.current_level += self.rainfall_amount
            reward += 2  # Small reward for rainfall adding water

        # Ensure the water level does not exceed the maximum capacity
        self.current_level = min(self.current_level, self.max_capacity)

        # Update the state
        self.state = self.current_level
        return self.state, reward

    def reset(self):
        self.current_level = self.max_capacity // 2
        self.state = self.current_level
        return self.state


# Streamlit UI
st.set_page_config(layout="wide", page_title="Water Resource Management
System")
st.title("Water Resource Management System")
st.write("""
This simulation demonstrates the management of a reservoir's water level based
on
various environmental factors and user-defined water allocation decisions.
""")

# Sidebar Input Controls
st.sidebar.header("Simulation Settings")
max_capacity = st.sidebar.slider("Reservoir Capacity (Maximum Level)", 50,
200, 100)
demand = st.sidebar.slider("Water Demand per Step", 10, 50, 20)
rainfall_prob = st.sidebar.slider("Rainfall Probability (Likelihood of Rain)",
0.0, 1.0, 0.6)
rainfall_amount = st.sidebar.slider("Rainfall Amount (Water Added)", 5, 30,
15)

# Initialize Environment
env = WaterResourceEnv(max_capacity, demand, rainfall_prob, rainfall_amount)

# Simulation State
if "state" not in st.session_state:
    st.session_state.state = env.reset()
    st.session_state.history = []

# Simulation Controls
steps = st.number_input("Number of Simulation Steps", 1, 50, 10)
```

```
if st.button("Run Simulation"):
    for _ in range(steps):
        action = np.random.randint(0, demand + 10)  # Random decision for
water allocation
        state, reward = env.step(action)
        st.session_state.history.append((action, state, reward))
    st.success(f"Simulation completed for {steps} steps!")

# Display Results
if st.session_state.history:
    st.write("### Simulation Results")

    # Extract Data
    actions, states, rewards = zip(*st.session_state.history)

    # Dashboard Layout
    col1, col2 = st.columns([1, 2])

    # Reservoir Animation (Single Animated Bar Chart with Matplotlib)
    with col1:
        st.write("#### Reservoir Level Animation (Matplotlib)")
        chart_placeholder = st.empty()  # Placeholder to hold the chart

        for state in states:
            fig, ax = plt.subplots(figsize=(3, 5))
            ax.bar(["Reservoir"], [state], color="blue", alpha=0.7)
            ax.set_ylim(0, max_capacity)
            ax.set_ylabel("Water Level")
            ax.set_title("Current Reservoir Level")
            chart_placeholder.pyplot(fig)  # Update the plot
            time.sleep(0.2)  # Pause to create the animation effect

    # Line Chart using Plotly for Rewards and Reservoir Levels over Time
    with col2:
        st.write("#### Reservoir Levels and Rewards Over Time (Plotly)")
        fig = go.Figure()

        # Plot the Reservoir Levels
        fig.add_trace(go.Scatter(x=list(range(len(states))), y=states,
mode='lines+markers', name='Reservoir Level (Water)',
line=dict(color='blue')))

        # Plot the Rewards
        fig.add_trace(go.Scatter(x=list(range(len(rewards))), y=rewards,
mode='lines+markers', name='Rewards (Decision Effectiveness)',
line=dict(color='green')))

        # Add Layout Details
```

```
        fig.update_layout(
            title="Reservoir Levels and Rewards Over Time",
            xaxis_title="Time Steps",
            yaxis_title="Value",
            showlegend=True
        )
        st.plotly_chart(fig)

    # Summary
    st.write("""
    **Summary:**
    - Final Reservoir Level: {} units
    - Average Reward: {:.2f} points
    """.format(states[-1], np.mean(rewards)))

    # Interpret Results
    st.write("""
    **Interpretation of Results:**
    - The reservoir level fluctuates due to water allocation and rainfall.
    - Rewards reflect the effectiveness of decisions:
      - Positive rewards: Successful water demand fulfillment.
      - Negative rewards: Over-allocation or under-allocation.
    """)

    # Show Actions History (optional)
    if st.checkbox("Show Actions History"):
        st.write("### Actions History")
        action_df = {"Step": list(range(1, len(actions) + 1)), "Action":
actions, "Reward": rewards, "Reservoir Level": states}
        st.table(action_df)

    # Show Average Reward and Final Reservoir Level
    st.write(f"**Average Reward:** {np.mean(rewards):.2f}")
    st.write(f"**Final Reservoir Level:** {states[-1]} units")
```