
**Khulna University of Engineering & Technology**
Khulna-9203, Bangladesh
**Department of Computer Science and Engineering**

---

CSE 3210 – Artificial Intelligence Laboratory

# EcoForge

## AI Ecological Optimization Simulation

---

**Submitted By:**

**Shormi Ghosh**
Roll No.: 2107109
Department of Computer Science and Engineering
Khulna University of Engineering & Technology

**Adit Mugdha Das**
Roll No.: 2107118
Department of Computer Science and Engineering
Khulna University of Engineering & Technology

**Submitted To:**

**Md Mehrab Hossain Opi**
Lecturer
Department of Computer Science and Engineering
Khulna University of Engineering & Technology

**Waliul Islam Sumon**
Lecturer
Department of Computer Science and Engineering
Khulna University of Engineering & Technology

---

28 April 2026

# Abstract

EcoForge is an artificial intelligence project centered on a competitive environmental simulation in which two agents act on a shared planetary ecosystem. The system is designed as a turn-based strategy environment where agents must manage water, food, oxygen, and temperature while competing for higher performance over time. The project combines game design, ecological modeling, reinforcement learning, adversarial planning, and simulation-based decision making within a single experimental framework.The implemented project is organized into three major layers: a simulation engine that maintains the world state and ecological dynamics, an artificial intelligence layer that contains multiple decision-making agents, and a rendering layer that visualizes the evolving world through a Python Pygame interface. In this report, the comparison focuses on three selected agent architectures: Minimax, Monte Carlo, and Deep Q-Network (DQN). This makes the project suitable not only as a playable simulation but also as a laboratory platform for comparing different AI approaches under the same environmental constraints.This report analyzes the project from both a system and AI perspective. It explains the architecture of the simulation, the role of environment parameters and actions, the design of the competing agents, and the evaluation methods used to judge environmental and strategic performance. The overall aim is to show how the project integrates ecological balance with intelligent decision making in a structured and academically meaningful way.

# Contents

# 2 Introduction

## 2.1 Introduction

EcoForge addresses the problem of maintaining environmental balance in a shared ecosystem. In the project, water, food, oxygen, and temperature are tightly connected, so improving one part of the environment can disturb another part if decisions are not made carefully. This creates a meaningful problem for artificial intelligence because the system is not static; it changes after every action and forces agents to think about long-term consequences rather than short-term gain.The motivation behind the project is to build a simulation where environmental management can be studied in a simple but clear form. Instead of using a purely theoretical model, the project places the problem inside a playable grid-based world where agents interact with terrain, resources, and global ecological indicators. This makes the project more understandable, more visual, and more suitable for analysis in an AI laboratory setting.AI and simulation are combined in this project because simulation provides the evolving environment, while AI provides decision-making inside that environment. The simulation creates the ecological rules and feedback loops, and the AI agents attempt to operate efficiently within those rules. Together, they form a system that is useful for comparing planning methods, learning methods, and overall intelligent behavior.

## 2.2 Objectives

The main objectives of this project are:

- Simulate environmental balance through interacting variables such as water, food, oxygen, and temperature.
- Implement multiple AI agents that can act within the same environment.
- Compare agent performance under common simulation conditions.
- Analyze how the system behaves as actions and environmental conditions change over time.
- Study the relationship between local actions on the grid and global effects on planetary stability.
- Build a visual and interactive platform that makes AI behavior easier to observe and explain.
- Create a reusable framework for comparing classical search methods and deep learning methods in the same environment.

## 2.3 System Overview

At a high level, the system consists of four connected parts. The first part is the **environment**, which represents the world grid and the global ecological indicators. The second part is the **game engine**, which applies actions, updates the world state, and

controls turn progression. The third part is the **agents**, which choose actions based on their own decision logic. The fourth part is the **deep learning model**, which is used by the DQN agent to improve its decision making from experience.In simple terms, the system works as follows: the environment provides the current state, an agent selects an action, the game engine applies that action, and the world is updated through simulation rules. This updated state is then used by the next agent or the next learning step. Because all components are connected, the project works as a complete AI simulation framework rather than a standalone game or a standalone algorithm demo.The overall structure is designed to keep the system modular. The environment stores state, the engine controls execution, the agents contain decision logic, and the learning component improves behavior over repeated interaction. This separation is useful for report writing because each part can be explained independently while still contributing to the same final system. The high-level relationship among these parts is illustrated in Figure 2.1.

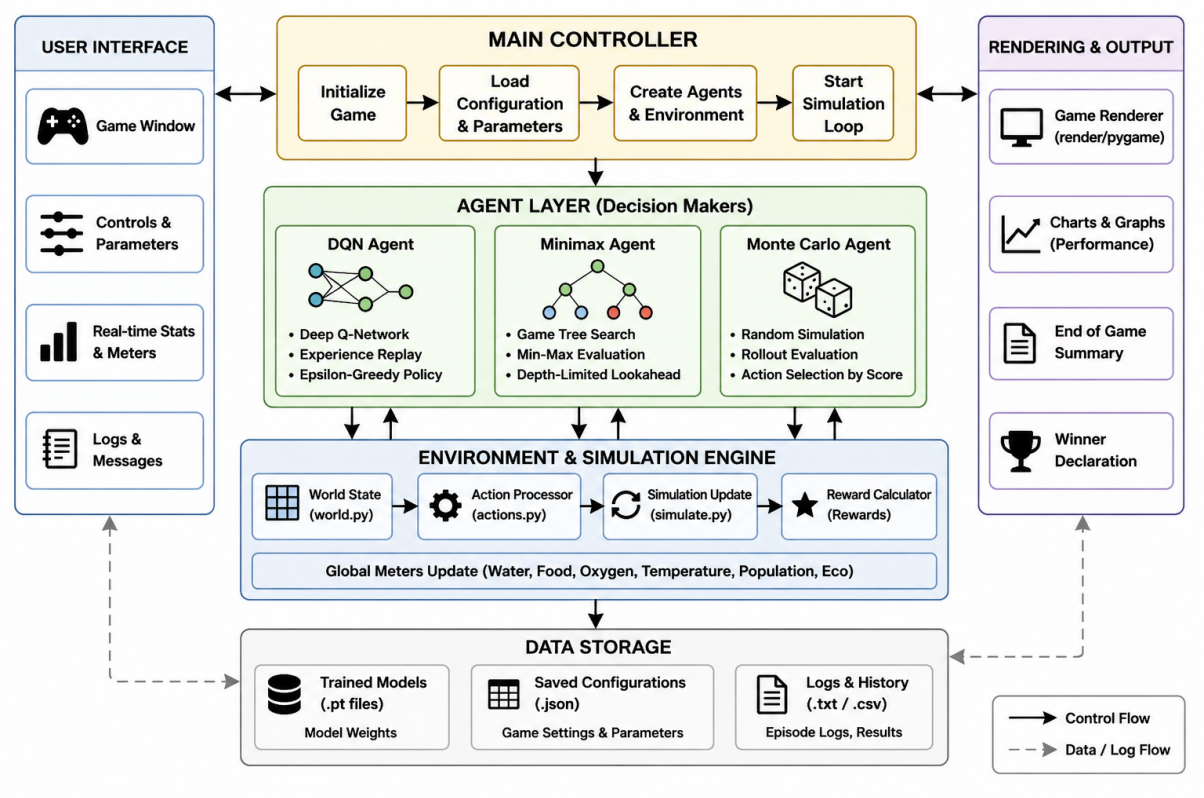


Figure 1: System Architecture of EcoForge

# 3 Methodology

## 3.1 Game Engine

The game engine is the core operational part of the project. It connects the environment, the agents, and the simulation logic into one continuous decision-making process. The environment is represented as a grid-based world in which each cell can hold a terrain type such as land, river, farm, forest, reservoir, or solar plant. Along with the grid, the engine also maintains global meters including water, food, oxygen, temperature, population, stability, score, and eco points. These variables together define the full state of the world at any moment.The simulation loop follows a turn-based structure. First, the active agent observes the current state of the environment and selects an action. That action is applied to the grid through the game engine. After the action is executed, the simulation updates the world by spreading water, growing crops, maturing forests, recalculating global meters, updating stability, and awarding scores. Then control passes to the next agent, and the same process repeats until the game reaches the maximum turn limit or the environment collapses. This loop is important because it allows every action to influence the next state in a measurable and repeatable way.The sequence of operations inside the engine is summarized in Figure 3.1. This figure is useful because it shows how the system moves from state observation to action execution and then to world update. A visual example of the implemented interface is also provided in Figure 3.2, which shows how the grid and the planetary indicators appear during execution.

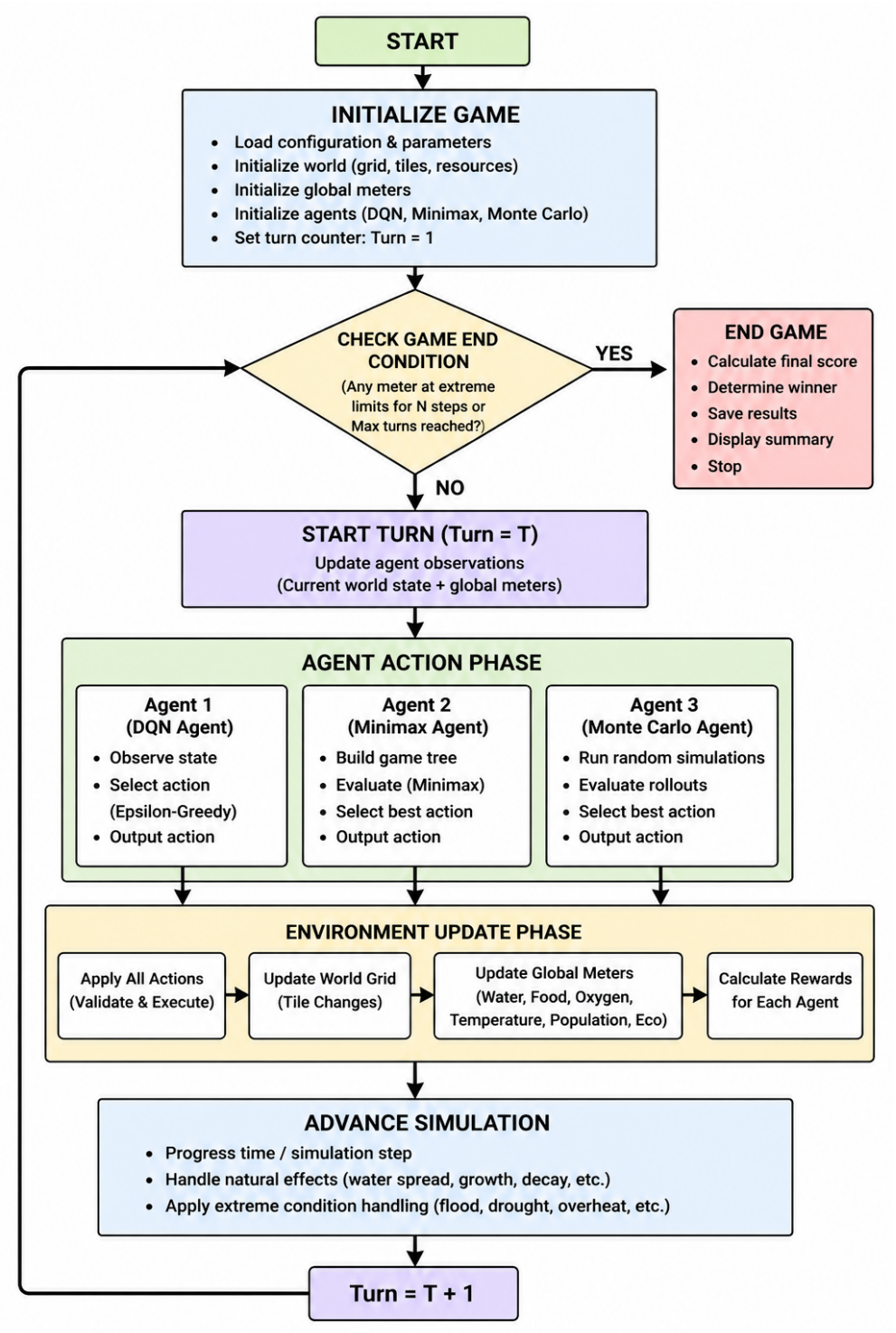


Figure 2: Game Flow of the Simulation Loop


Figure 3: Game World Interface

## 3.2 Action Design and Parameters

The action system defines how agents interact with the world grid and influence global ecological meters. EcoForge provides **eight distinct actions**, each with a specific eco-point cost, placement requirement, and environmental consequence. Table 3.1 summarizes the complete action space.

| Action | Eco Cost | Primary Effect |
|---|---|---|
| Build Canal | 3 | Converts Land → River; expands water network to adjacent cells |
| Build Reservoir | 10 | Converts Land → Reservoir; permanent water source with per-step asset score bonus |
| Plant Forest | 4 | Converts Land → Forest; produces oxygen and cools the planet |
| Clear Forest | 2 | Removes own Forest → Land; eliminates oxygen production and cooling from that tile |
| Plant Farm | 3 | Converts Land → Farm; requires adjacent water; grows crops through 4 stages |
| Harvest Crop | 1 | Instantly adds +15 food from a mature stage-3 farm; awards +2.0 score bonus |
| Build Solar Plant | 8 | Generates +1 eco/step (passive income); heats planet +0.4°/step |
| Adjust Allocation | 0 | Free action; sets agent priority to Farm, Forest, or Balanced |

Table 1: Complete action space with eco costs and primary effects

**Build Canal (Cost: 3 eco).** Converts a Land tile to a River tile, which must be placed adjacent to an existing River or Reservoir. It permanently sets the cell water

value to 10.0, expanding the water network so that surrounding tiles become eligible for farm placement. This is typically the first infrastructure action required before any agriculture is viable.

**Build Reservoir (Cost: 10 eco).** Converts Land to a Reservoir following the same adjacency rule as a Canal. Its key strategic value beyond water supply is a continuous asset scoring bonus of +0.3 points per simulation step, making it a long-term investment.

**Plant Forest (Cost: 4 eco).** Creates a Forest tile at maturity level 0. Forests grow from maturity $0 \to 1 \to 2 \to 3$ over nine simulation steps (3 steps per level). Each forest produces +0.2 oxygen per step as a base, plus $+0.08 \times$ maturity as a bonus. Temperature cooling is context-sensitive: no cooling below 40°; cooling equal to maturity_sum $\times 0.05$ in the optimal 40–60° range; cooling at maturity_sum $\times 0.10$ above 60°; and emergency cooling at maturity_sum $\times 0.20$ above 80°.

**Clear Forest (Cost: 2 eco).** Removes an owned Forest tile and immediately eliminates all oxygen production and temperature cooling from that cell. This action is used strategically when oxygen is already oversaturated above 85, because having many forests at high oxygen levels triggers wildfire drain proportional to forest count.

**Plant Farm (Cost: 3 eco).** Creates a Farm tile at crop stage 0, requiring water access in the target cell or any neighbor (water $\geq 3.0$). Crops advance from stage 0 through 3 every two simulation steps when water is available. Only stage-3 mature farms produce food passively at +0.5/step multiplied by farm efficiency. Each farm also drains −0.2 water per step through irrigation and warms the planet by +0.1° per step.

**Harvest Crop (Cost: 1 eco).** Adds +15 food instantly from a mature stage-3 farm, hard-capped at 85 to prevent oversaturation. The crop resets to stage 0 to regrow, and the harvesting agent receives a +2.0 score bonus. Passive farm production of +0.5/step is slow; harvesting provides the rapid food injection needed to prevent starvation crises.

**Build Solar Plant (Cost: 8 eco).** Generates +1 eco income per step, making it the primary passive eco source in the late game. However, each solar plant adds +0.4° of heat per step and drains −0.1 oxygen per step. Three solar plants alone produce +1.2°/step, which is sufficient to push temperature toward dangerous levels without compensating forests.

**Adjust Resource Allocation (Cost: 0 eco).** A free action that sets the agent's internal priority mode to Farm, Forest, or Balanced. It does not modify any tile and is valid at any point, allowing agents to signal their current strategic stance.

**Action Chain Reactions.** The actions are not independent — they trigger cascading effects through the simulation engine. The most important chains are:

- **Water–Agriculture chain:** Building a Canal or Reservoir spreads water to adjacent cells, enabling Farm placement. Farms then drain water (−0.2/step), so overbuilding farms without sufficient water infrastructure causes crops to stall and food production to collapse.

- **Forest–Solar conflict:** Forests cool the planet while Solar Plants heat it. An agent must balance both to maintain temperature in the optimal 40–60° range.
- **High water flooding:** Water exceeding 80 triggers a flood drain of $(water - 80) \times 0.15$/step. At water $> 85$, farm efficiency drops to $\times 0.6$; at water $> 92$, it collapses to $\times 0.35$, causing near-total crop failure even with many farms present.
- **Heat cascade:** If temperature rises above 65°, farm efficiency drops to $\times 0.7$; above 75°, it falls to $\times 0.4$ and population loses $-2.0$ health per step, creating an urgent recovery requirement.

**Action Effects on Global Parameters.** The effect of actions is not limited to the modified tile. Every action changes the counts of rivers, reservoirs, farms, forests, or solar plants, and those counts are then used by the simulation engine to update the global meters. In simplified form, the project updates the planet through four main delta equations:

$$\Delta W = a_1 N_{\text{river}} + a_2 N_{\text{reservoir}} + a_3 N_{\text{forest}} - b_1 N_{\text{farm}} - b_2 P - b_3 T_{\text{heat}}, \tag{1}$$

$$\Delta F = c_1 N_{\text{mature farm}} - c_2 P - c_3 F_{\text{spoilage}}, \tag{2}$$

$$\Delta O = d_1 N_{\text{forest}} + d_2 M_{\text{forest}} - d_3 N_{\text{farm}} - d_4 N_{\text{solar}} - d_5 P, \tag{3}$$

$$\Delta T = e_1 N_{\text{farm}} + e_2 N_{\text{solar}} + e_3 P - e_4 M_{\text{forest}} - e_5 W_{\text{coverage}}. \tag{4}$$

These equations show the central design idea of the project: local actions create global consequences. Building a canal or reservoir improves water support, planting forests improves oxygen and cooling, building farms increases food but also raises environmental pressure, and building solar plants improves eco income but increases temperature. Because the effects are connected, actions must be judged in relation to the full system rather than in isolation.

## 3.3 System Dynamics and Interactions

The project uses a dynamic ecological model in which all important variables interact with one another. Water influences crop growth, crops influence food production, forests influence oxygen and cooling, and solar infrastructure influences temperature and eco income. This creates chain reactions throughout the simulation. A single action such as building a canal may improve water access, which can later support farms, which can later increase food, which can then affect population growth and overall stability.The global meters are central to the system. Water, food, oxygen, and temperature are continuously updated after each turn. These meters are not independent; instead, they form feedback loops. High temperature can reduce water through evaporation and weaken agriculture. Too much water can flood the system and reduce efficiency. High oxygen can support ecological health, but extreme oversaturation can also create imbalance. In this way, the project avoids one-dimensional optimization and instead requires balanced management.Extreme conditions play a major role in the simulation. Very low water, food, or oxygen levels can push the world toward collapse, while excessive values can also create instability. The system therefore contains self-regulation behavior. Examples include food spoilage at high food levels, oxygen bleed at excessive oxygen levels, heat

radiation at very high temperature, and population-based consumption that naturally pulls the system back from unrealistic extremes. These mechanisms make the simulation more realistic and help create a stable but challenging decision environment.

**Extreme Case Handling.** The project explicitly handles extreme cases instead of allowing uncontrolled growth or collapse. When water, food, or oxygen fall below critical thresholds, the stability score is hard-capped, which makes collapse possible after consecutive critical turns. On the opposite side, the simulation also penalizes oversaturation: high water creates flood drain, high food produces spoilage, high oxygen produces oxygen bleed and wildfire-related drain, and very high temperature activates heat radiation. Population is also bounded and affected by drought, heat stress, and oxygen shortage. These rules are important because they prevent unrealistic runaway behavior and make the world self-regulating even under aggressive agent strategies.

## 3.4 Global Resource Meters

The simulation maintains four global resource meters (0–100) with defined optimal ranges. Planetary stability is the average of how close all four meters are to their optimal bounds simultaneously.

### Water Level (0–100, optimal: 50–80)

| Source | Effect |
|---|---|
| Each River tile | +0.3/step |
| Each Reservoir | +0.3/step |
| Each Forest | +0.05/step (transpiration) |
| Each Farm | −0.2/step (irrigation drain) |
| Population | −(pop × 0.01) minimum/step |
| Passive evaporation | −0.6/step always |
| If water > 80 | extra drain: $(water - 80) \times 0.15$ |
| If temp > 70° | extra drain: $(temp - 70) \times 0.06$ |

### Food (0–100, optimal: 50–80)

| Source | Effect |
|---|---|
| Mature farm (stage 3) | +0.5/step × farm_efficiency |
| Harvest Crop action | +15 instantly (max up to 85) |
| Population eating | −(pop × 0.04) minimum/step |
| Overpopulation (> 100) | eats even more per capita |
| Food > 72 | spoilage: $(food - 72) \times 0.30$/step |

**Farm efficiency modifiers:**

- Temp > 75°: ×0.4

- Temp $> 65°$: $\times 0.7$
- Water $> 92$: $\times 0.35$ (also applied)
- Water $> 85$: $\times 0.6$ (also applied)

### Oxygen (0–100, optimal: 50–80)

| Source | Effect |
|---|---|
| Each Forest | +0.2/step |
| Each Forest (maturity bonus) | +0.08× maturity/step |
| Each Farm | −0.05/step |
| Each Solar Plant | −0.1/step |
| Population breathing | $-(\text{pop} \times 0.04)$/step (reduced when $O_2 < 40$) |
| Passive atmospheric drain | −0.3/step always |
| Temp $> 65°$ | heat decomposition: $(temp - 65) \times 0.06$ drain |
| Oxygen $> 80$ | bleeds off: $(oxy - 80) \times 0.65$/step |
| Oxygen $> 85$ + forests | wildfire drain: $(oxy - 85) \times \text{forests} \times 0.04$ |

### Temperature (0–100, optimal: 40–60)

| Source | Effect |
|---|---|
| Passive greenhouse | $+0.5°$/step always |
| Each Farm | $+0.1°$/step |
| Each Solar Plant | $+0.4°$/step |
| Oxygen $> 80$ | oxidation heat: $(oxy - 80) \times 0.04$ |
| Population $> 100$ | human activity heat: $(pop - 100) \times 0.005$/step |
| Forests (temp $> 60°$) | maturity_sum $\times -0.10$/step |
| Forests (temp $> 80°$) | maturity_sum $\times -0.20$/step (emergency) |
| Water cells on grid | −0.01 per water-covered cell |
| Temp $> 80°$ | radiates heat: $(temp - 80) \times 0.175$ (self-regulation) |

## 3.5 Population Dynamics

Population lives in the range **5 to 200**.

**It grows when:**

- Food $> 50$ (base growth $= (food - 50) \times 0.08$)
- All 4 meters in optimal range → extra +0.4/step
- Decent conditions → extra +0.2/step
- Logistic brake: growth slows as you approach 200 — $\times(1 - pop/200)$

**It declines when:**

- Oxygen $< 25$: $-2.5$/step (suffocation)
- Oxygen $< 40$: $-0.8$/step
- Temp $> 75°$: $-2.0$/step
- Temp $> 65°$: $-0.8$/step
- Water $< 15$: $-1.5$/step (drought)
- Water $< 30$: $-0.6$/step

**Population in turn feeds back into the meters:**

- Drinks water, eats food, breathes oxygen every step
- Above population 100: `overload_factor` increases per-capita consumption by up to 50% at pop = 200
- Above population 100: adds cumulative warming $+0.005°$/step per person above 100

**The cycle:**

```
Good food -> pop grows -> pop eats more food -> food drops
          -> pop breathes more oxygen -> oxygen drops
          -> pop drinks more water -> water drops
          -> high pop generates heat -> temp rises
              -> crops fail -> food collapses -> pop crashes
```

## 3.6 Scoring System

Every simulation step (every turn):

1. **Both agents** get stability $\times 1.0$ points — shared reward for keeping the planet healthy
2. **Individual asset scoring:**
    - Forest: $+0.2\times$ maturity/step
    - Farm: $+0.15\times$ crop_stage/step
    - Reservoir: $+0.3$/step
    - River (owned): $+0.1$/step
3. **Harvest bonus**: $+2.0$ immediately when you harvest

Planet stability is the average of how close all 4 meters are to their optimal ranges — so both agents are incentivized to keep the planet balanced, even while competing for territory.

## 3.7 Extreme Value Handling

The simulation explicitly handles extreme meter values to prevent unrealistic runaway behavior and create meaningful crisis scenarios for agents to navigate.

### Water at 0

- No farms can grow (no water access)
- Population takes drought penalty: $-1.5$/step if $< 15$
- Planet becomes a desert — game spirals fast

### Water at 100 (flooding)

- `flood_drain` removes water fast: $(100 - 80) \times 0.15 = -3$/step (self-correcting)
- Farm efficiency $\times 0.35$ (near total crop failure)
- Food collapses even if you have many farms

### Food at 0

- Population shrinks rapidly (no food = starvation)
- Pop drop reduces resource consumption which gives a partial recovery window

### Food at 100

- Spoilage: $(100 - 72) \times 0.30 = -8.4$/step (rapid self-correction)
- Hard cap at 85 enforced on harvests

### Oxygen at 0

- Population: $-2.5$/step (suffocation crisis)
- Essentially unrecoverable without urgent forest planting

### Oxygen above 85

- Wildfire drain: proportional to forest count — **ironically, having many forests when oxygen is too high makes it drain faster**
- This discourages pure oxygen stacking

### Temperature above 80

- Heat radiation kicks in: $(temp - 80) \times 0.175$ (passive self-correction, but slow)
- Forest emergency cooling doubles: maturity_sum $\times 0.20$
- Pop: $-2.0$/step, farm efficiency $\times 0.4$, heat evaporation worsens water drain
- Game becomes a crisis requiring heavy forest planting

### Temperature below 40 (cold)

- Forest cooling is disabled (forests don't cool when it's already cold)
- Passive greenhouse $+0.5°$/step will slowly warm it back up
- Rarely a real danger since solar and farms constantly generate heat

## 3.8 Agent Architectures

The report focuses on three agents: **Minimax**, **Monte Carlo**, and **DQN**. These three agents were selected because they represent three different styles of intelligence: search-based planning, simulation-based estimation, and deep reinforcement learning.

### 3.8.1 Minimax Agent

The **Minimax** agent is based on adversarial search. It explores possible future moves by building a decision tree and estimating which move gives the best outcome while assuming the opponent will also act intelligently. This makes Minimax suitable for competitive situations where the agent must think ahead and consider both its own actions and the likely responses of the opponent.Within this project, Minimax acts as a structured planning model. It is useful because it evaluates future possibilities before making a move, which allows it to respond strategically to competitive pressure. This makes it a strong representative of classical search-based artificial intelligence.

**Evaluation Function.** Minimax uses the shared state-evaluation function that is also used by Monte Carlo at the end of rollouts. In simplified form, the evaluation score can be written as:

$$E(s) = 10S + 8R + 12T + 5D_{\text{score}} + P_{\text{collapse}} + 1.5A + 2D_{\text{territory}} + 6P_{\text{health}} + 0.3D_{\text{eco}}, \quad (5)$$

where $S$ is stability, $R$ is the combined resource score, $T$ is temperature optimality, $D_{\text{score}}$ is score difference, $P_{\text{collapse}}$ is the collapse penalty, $A$ is asset value, $D_{\text{territory}}$ is cell-control difference, $P_{\text{health}}$ is population health, and $D_{\text{eco}}$ is eco-point advantage. This evaluation function gives the agent a multi-objective view of the game instead of rewarding only one resource.

Table 3.9 details each component with its weight and meaning. Temperature receives the highest weight of $\times 12.0$ because it is the most critical systemic variable: out-of-range temperature simultaneously reduces farm efficiency, damages population health,

and destabilizes oxygen. The collapse penalty can produce values as severe as −100 per resource meter in crisis, meaning a near-collapse state dominates all positive components and ensures both Minimax and Monte Carlo strongly avoid dangerous world states.

| Component | Weight | Description |
|---|---|---|
| Planet Stability | ×10.0 | World stability score mapped from 0.0–1.0 |
| Resource Level Score | ×8.0 | Average of water, food, and oxygen meter scores (each scored −0.5 to 1.0) |
| Temperature Optimality | ×12.0 | Closeness to optimal 40–60°; highest weight in the function |
| Score Differential | ×5.0 | Agent score minus opponent score |
| Collapse Penalty | ×1.0 | Up to −100 per resource in crisis; −50 for extreme temperature |
| Strategic Asset Value | ×1.5 | Context-sensitive value of owned forests, farms, and reservoirs |
| Territory Control | ×2.0 | Owned cell count minus opponent cell count |
| Population Health | ×1.0 | Ranges from 0 to 9.0; peaks at population 150+ |
| Eco Advantage | ×0.3 | Own eco points minus opponent eco points |

Table 2: Component weights in `evaluate_state()` — shared by Minimax and Monte Carlo

**Algorithm 1:** Minimax Decision Procedure

**Input:** Current state $s$, search depth $d$, maximizing flag
**Output:** Evaluation value

**if** $d = 0$ *or* *Terminal*($s$) **then**
  **return** Evaluate($s$);
**end**
**if** *maximizing* **then**
  $value \leftarrow -\infty$;
  **foreach** $a \in$ *ValidActions*($s$) **do**
    $value \leftarrow \max(value, \text{Minimax}(\text{Result}(s, a), d - 1, \text{false}))$;
  **end**
  **return** *value*;
**end**
**else**
  $value \leftarrow +\infty$;
  **foreach** $a \in$ *ValidActions*($s$) **do**
    $value \leftarrow \min(value, \text{Minimax}(\text{Result}(s, a), d - 1, \text{true}))$;
  **end**
  **return** *value*;
**end**

Figure 3.3 presents the basic decision-tree idea behind the Minimax agent. The figure supports the explanation by showing how the agent expands candidate actions and evaluates possible future states before choosing a move.

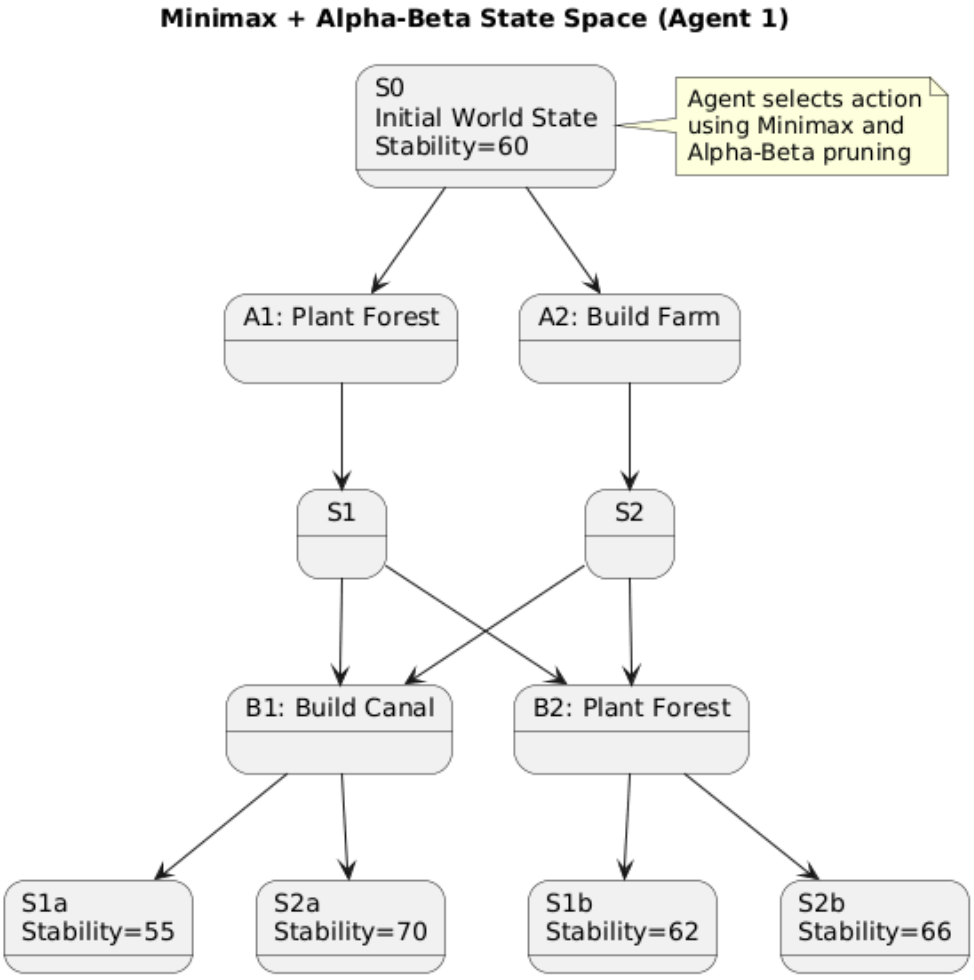


Figure 4: Minimax Decision Tree

### 3.8.2 Monte Carlo Agent

The **Monte Carlo** agent evaluates actions by simulation. Instead of constructing a full adversarial tree, it tries many rollout-based futures from a candidate action and then estimates which action produces the best average result. This method is useful when the environment is complex and exact search is expensive, because it provides a practical way to estimate action quality through repeated sampling.In the context of this project, Monte Carlo is effective because it estimates action quality from repeated experimental rollouts rather than strict tree expansion. This gives it flexibility in a dynamic ecological environment where many interacting variables can change the future outcome of an action.

**Evaluation Function.** Monte Carlo uses the same evaluation function as Minimax, but it applies it after several simulated futures instead of directly at the leaves of a search tree. If an action produces $N$ rollout results, then the action score is the mean final evaluation:

$$MC(a) = \frac{1}{N} \sum_{i=1}^{N} E(s_i^{\text{rollout}}). \tag{6}$$

This makes Monte Carlo less dependent on exact tree expansion and more dependent on the quality of sampled futures. It is therefore well suited for a dynamic simulation where approximate future estimation is often more practical than exhaustive search.

**Algorithm 2:** Monte Carlo Action Selection

**Input:** Current state $s$, rollout count $N$, rollout depth $d$
**Output:** Best selected action

```
best_action ← None;
best_score ← −∞;
foreach a ∈ CandidateActions(s) do
    total ← 0;
    for i ← 1 to N do
        next_state ← Result(s, a);
        rollout_state ← RandomRollout(next_state, d);
        total ← total+ Evaluate(rollout_state);
    end
    avg ← total/N;
    if avg > best_score then
        best_score ← avg;
        best_action ← a;
    end
end
return best_action;
```

The logic of this rollout-based process is illustrated in Figure 3.4. This figure helps explain how the Monte Carlo agent samples possible futures and then uses average outcome quality to guide the final action choice.

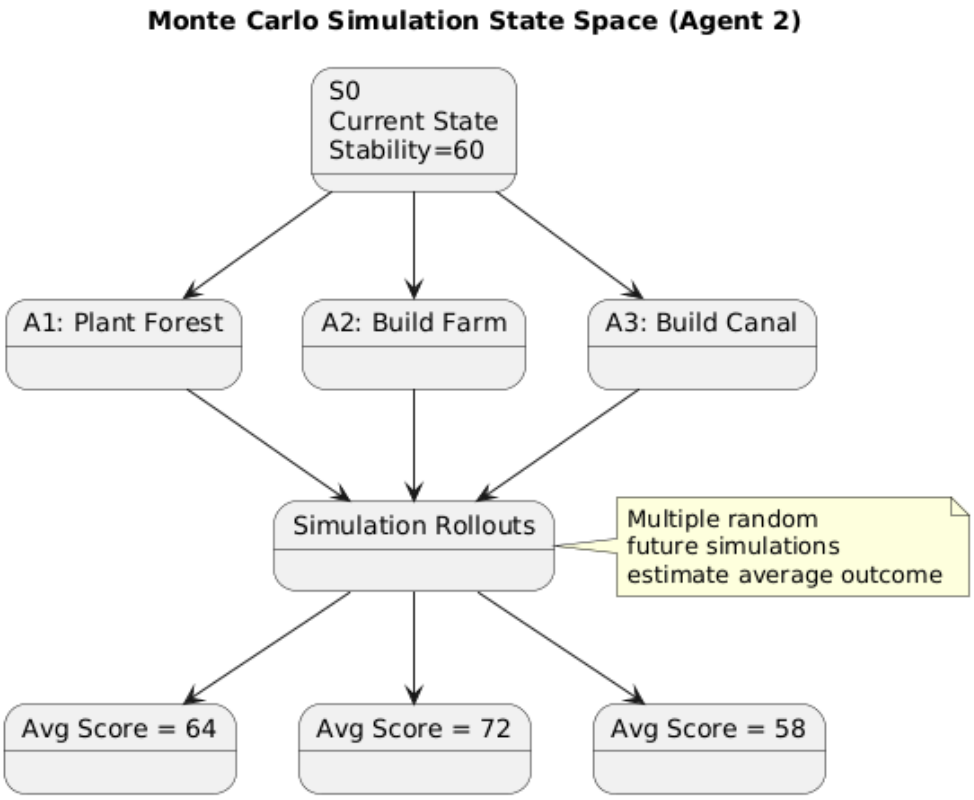


Figure 5: Monte Carlo Decision Flow

## 3.9 DQN Model

### 3.9.1 DQN Architecture

The Deep Q-Network (DQN) is the learning-based agent in the project. Unlike Minimax and Monte Carlo, which evaluate actions through search or simulation at decision time, DQN learns a mapping from world states to action values through repeated training. Its input is a numerical representation of the current environment, including major planet conditions and competitive features, and its output is a set of predicted values for the available actions.The DQN model uses a neural network to approximate the quality of actions. Instead of storing a direct value for every possible situation, it learns patterns from numerical state features and predicts which actions are likely to be beneficial. In this project, the state representation includes important information such as water, food, oxygen, temperature, population, stability, owned cells, opponent cells, eco points, and score difference. These features allow the network to reason about both environmental balance and competitive advantage at the same time.During training, the agent stores previous experiences and learns from them over time. It uses a replay buffer to sample past experiences, which helps reduce correlation between consecutive updates, and a target network to stabilize training. This allows the model to gradually improve its policy instead of relying only on immediate handcrafted reasoning. The architecture of the model is shown in Figure 3.5, where the relationship between input features, hidden processing layers, and action-value outputs can be presented clearly.

**Learning Equation.** The core update rule of the DQN agent is based on the Bellman target:

$$y_t = \begin{cases} r_t, & \text{if the state is terminal,} \\ r_t + \gamma \max_{a'} Q_{\text{target}}(s_{t+1}, a'), & \text{otherwise.} \end{cases} \tag{7}$$

The online network predicts $Q(s_t, a_t)$ and is trained to move closer to $y_t$. In the implementation, this difference is optimized with Huber loss, which is more stable than plain mean squared error for noisy reinforcement-learning updates.

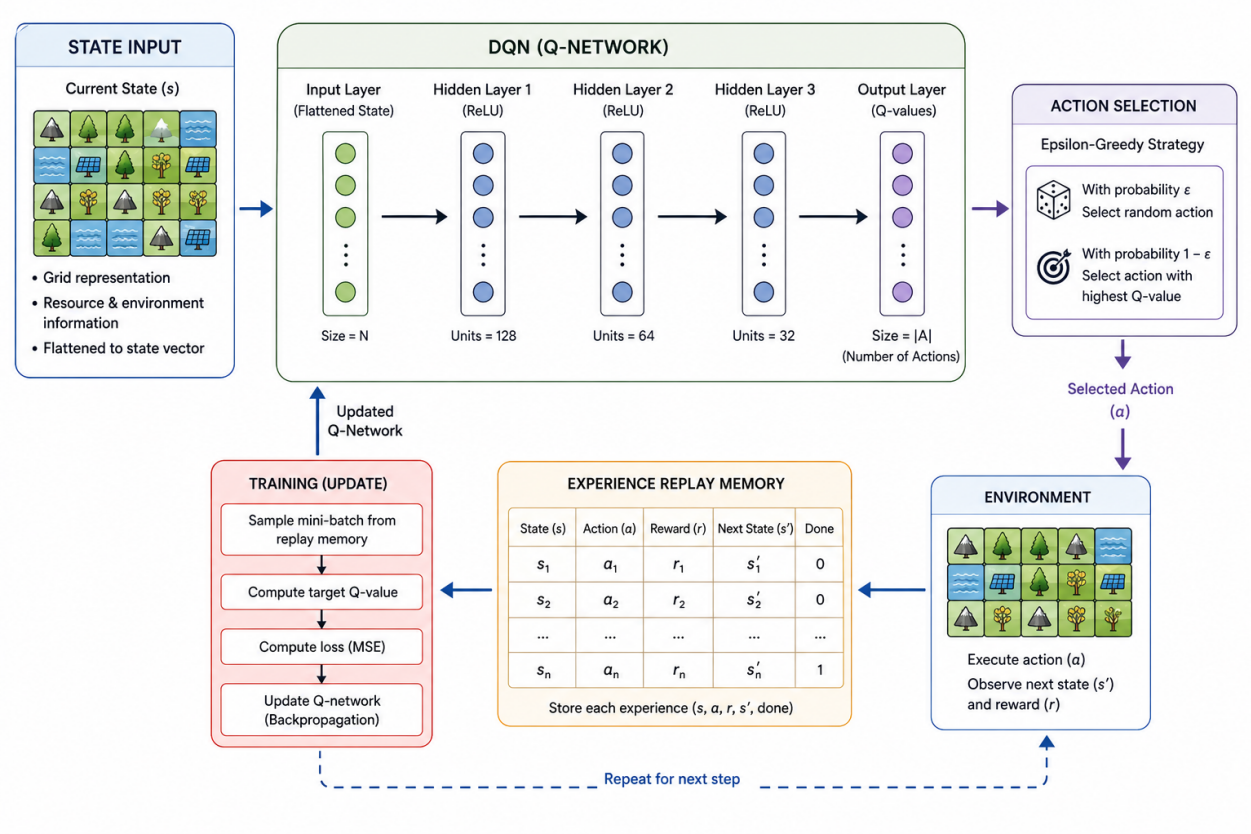


Figure 6: DQN Model Architecture

### 3.9.2 DQN Decision Flow

The training process is performed over many episodes. In each episode, the DQN agent interacts with the environment, takes actions, receives rewards, and updates its parameters based on the results. Over time, the model learns which actions are more useful in different ecological situations. This makes DQN especially important in the project because it represents the deep learning component and provides a clear contrast to the two classical AI methods.From a system perspective, the DQN decision flow follows a repeated sequence: the agent observes the current state, predicts action values, chooses an action, receives a reward after simulation, stores the transition, and updates the network gradually through training. This learning cycle is what distinguishes DQN from the other two agents and makes it the deep learning component of the report.

Another important point is that DQN does not become strong from a single game.

Its performance depends on repeated exposure to the environment, gradual parameter updates, and the ability to learn from both successful and unsuccessful decisions. Figure 3.6 should therefore be used to show the complete flow of observation, action, reward, storage, and learning update so that the reader can understand how the model improves over time.During each training step, the network minimizes the difference between the predicted Q-value and the Bellman target using **Huber loss**, which is more robust to large reward outliers than plain mean squared error and produces more stable gradients in environments with highly variable rewards. To prevent gradient explosion, all gradient norms are **clipped at 1.0** before each parameter update, which keeps the network weights from changing too drastically in a single step. The **target network** is synchronized with the online network only once every fixed number of steps rather than after every update, which prevents the training target from shifting too rapidly and causing the oscillation that typically destabilizes naive Q-learning implementations.

**Reward Function.** Unlike Minimax and Monte Carlo, DQN does not rely on a single final state evaluation only. It learns from a reward function after each action and simulation step. In simplified form, the reward used in the project can be expressed as:

$$r_t = 20(S_t - S_{t-1}) + B_{\text{stable}} - P_{\text{collapse}} + 1.5T_t + 1.5R_t + 0.15D_{\text{score}} + 0.12D_{\text{territory}} + 0.2A_t - P_{\text{eco}}, \tag{8}$$

where the reward includes stability improvement, temperature quality, resource quality, score difference, territory advantage, asset value, and eco-hoarding penalty. This design is important because it teaches the DQN agent not only to survive, but also to keep the ecosystem balanced while remaining competitive.

The key difference between the reward function and the state evaluation function is that `compute_reward` is **delta-based**: the core signal is the change in stability rather than its absolute value. This means the DQN is rewarded for *improving* the world, not merely for occupying a good state. Table 3.10 breaks down every component.

| Component | Scale | Description |
|---|---|---|
| Stability delta | $\times 20.0$ | Core signal: change in stability this step ($\Delta S \times 20$) |
| Stability zone bonus | $+3.0$ / $+1.0$ | $+3.0$ if stability is high; $+1.0$ if moderate |
| Stability zone penalty | $-5.0$ / $-20.0$ | $-5.0$ if stability is low; $-20.0$ near collapse |
| Temperature optimality | $\times 1.5$ | Score for temperature closeness to 40–60° range |
| Resource optimality | $\times 1.5$ | Average water, food, oxygen meter score |
| Population health | $+2.0$ to $-8.0$ | $+2.0$ at pop 50–100 (thriving); $-8.0$ if pop crashes below 20 |
| Score differential | $\times 0.15$ | Competitive signal: own score vs opponent score |
| Territory control | $\times 0.12$ | Cell count advantage over opponent |
| Asset value | $\times 0.2$ | Value of owned buildings and terrain |
| Eco hoarding penalty | $-0.05$/unit | Penalty for unspent eco above 60; discourages inaction |

Table 3: Component breakdown of `compute_reward()` used exclusively by the DQN agent

The eco hoarding penalty is unique to the DQN reward and does not appear in `evaluate_state()` — it specifically addresses a failure mode where the DQN accumulates eco points without spending them, which reduces its ability to build assets and weakens its long-term performance.

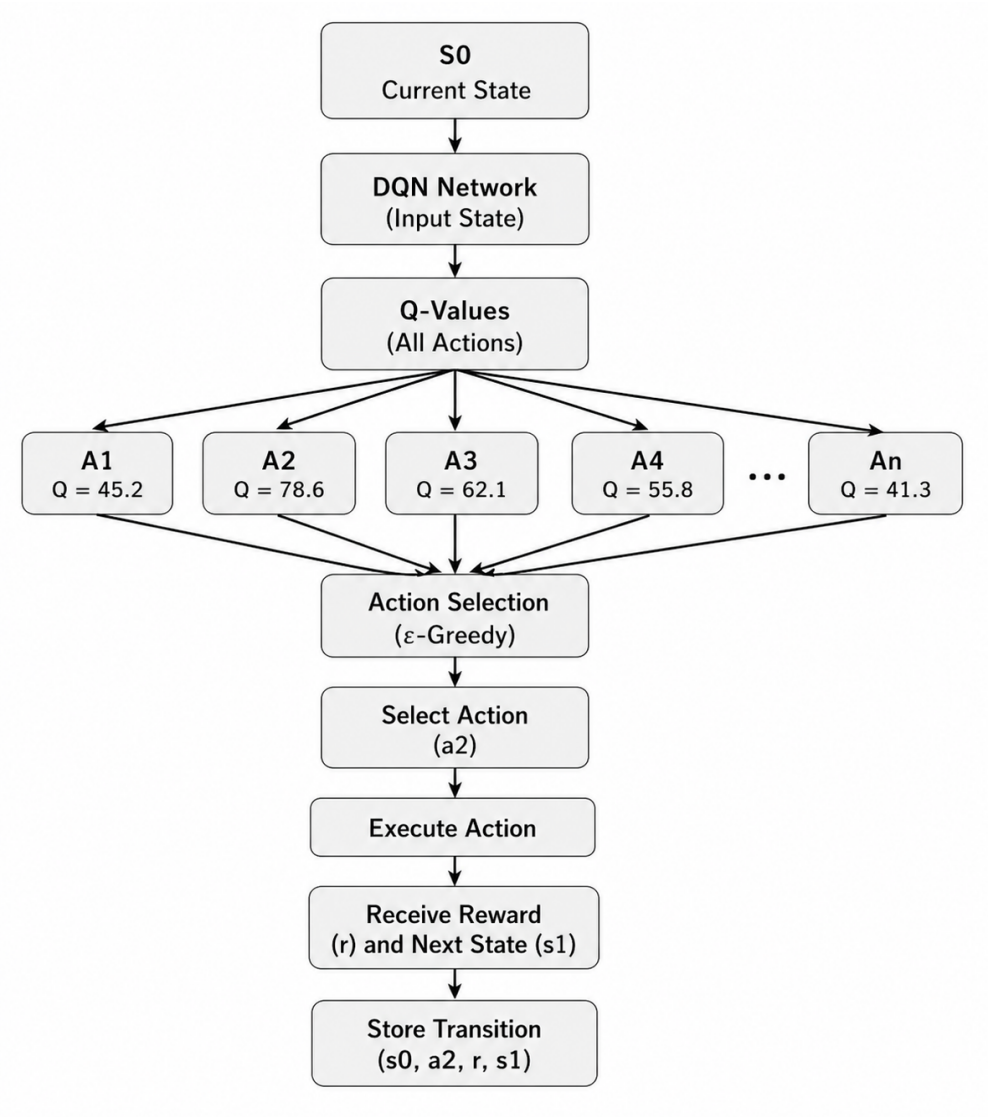


Figure 7: DQN Decision Flow

# 4 Evaluation

## 4.1 Metrics

The performance of the system can be evaluated using several metrics. The most important measures are planetary stability, resource balance, temperature control, food availability, oxygen level, score accumulation, survival duration, and eco-point efficiency. These metrics show not only whether an agent is winning, but also whether it is maintaining the environment in a sustainable way.

## 4.2 Comparison

The three selected agents were compared under the same environment settings and core simulation rules. In the observed runs, **Monte Carlo generally performed better than Minimax**, mainly because rollout-based sampling adapted more effectively to the changing ecological state than limited-depth adversarial search. The trained **DQN agent achieved an approximate win rate of 62%** against the other agents, showing that the learned policy was able to compete successfully against both planning-based approaches.

## 4.3 Results and Discussion

The results indicate that the project successfully supports meaningful comparison among different AI methods. Minimax performed as a stable baseline and showed good short-horizon planning, but its effectiveness was reduced when the search depth had to be kept small for practical execution. Monte Carlo performed better in most cases against Minimax because it estimated action quality from multiple sampled futures instead of relying on a strict depth-limited tree. This made it more adaptable in a simulation where the world changes continuously after every action.The best overall performance came from the trained DQN agent. After training, DQN achieved an approximate win rate of **62%** against the other two agents. This suggests that the network was able to learn useful patterns from repeated interaction with the environment and convert them into stronger decisions over time. In particular, DQN benefited from learning the long-term relationship between local actions and global ecological balance, which is difficult to encode perfectly in a manually designed search strategy.

| **Agent** | **Observed Outcome** | **Remarks** |
|---|---|---|
| Minimax | Usually below Monte Carlo | Sensitive to reduced search depth |
| Monte Carlo | Usually beats Minimax | Better adaptation through rollouts |
| DQN | About 62% win rate overall | Strong learned policy after training |

Table 4: Summary of observed comparative performance

From a discussion perspective, these results are consistent with the structure of the project. The environment contains many interacting variables, delayed consequences, and feedback loops. In such a setting, a purely depth-limited search method can miss important long-term effects, while rollout-based or learned methods can perform better.

# 5 Limitations

Although the project is functional and suitable for comparison, it has several practical limitations. First, the full game was not always simulated to its natural endpoint during experimentation. In some cases, execution was cut short intentionally in order to manually play the game or inspect the system earlier. This means that some observations were based on partially completed runs rather than a perfectly uniform batch of full-length matches.

Second, the search depth for the planning-based agents was deliberately reduced to keep execution time manageable. This especially affects Minimax, because its quality depends strongly on how far ahead it can search. A shallow search depth makes the agent faster and more playable, but it also reduces its ability to capture long-term environmental consequences.

Third, the project uses a simplified grid world and a simplified ecological model rather than a real-world environmental simulation. While this is appropriate for an AI laboratory project, it means that the results should be interpreted as algorithmic comparisons inside a designed simulation, not as predictions about real ecological systems.

Finally, the reported results are based on observed project behavior and training outcomes, not on a large-scale statistically controlled benchmark. Therefore, the conclusions are meaningful for project analysis and demonstration, but they should not be treated as final universal rankings of the algorithms.

# 6 Conclusion and Future Work

## 6.1 Conclusion

EcoForge successfully combines ecological simulation, strategic decision making, and artificial intelligence in a single interactive framework. The project demonstrates how a shared environment with water, food, oxygen, temperature, population, and stability can be used to evaluate different AI approaches under the same conditions. Through this structure, the project achieves both educational and technical value: it is visually understandable, experimentally useful, and rich enough to support meaningful agent comparison.The comparison of Minimax, Monte Carlo, and DQN shows that the project is capable of distinguishing different types of intelligence. Minimax provides a classical planning baseline, Monte Carlo offers stronger adaptability through simulation, and DQN shows the advantage of learning from repeated experience. The observed results, especially the approximate 62% win rate of DQN and the frequent advantage of Monte Carlo over Minimax, support the usefulness of the project as an AI laboratory study.

## 6.2 Future Work

There are several natural directions for improving the project in the future:

- Run larger and more consistent batches of full-game simulations for stronger result validation.
- Increase Minimax depth and Monte Carlo rollout budget when more computation time is available.
- Improve the DQN training process with longer training schedules and additional tuning.
- Add more detailed logging and result tracking for clearer quantitative comparison.
- Expand the visual presentation with richer assets, animations, and interface polish.
- Introduce more environment features or actions while preserving balance and interpretability.

# 7 References

# References

[1] Stuart Russell and Peter Norvig, *Artificial Intelligence: A Modern Approach*, Pearson, 4th edition, 2020.

[2] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, et al., "Human-level control through deep reinforcement learning," *Nature*, vol. 518, pp. 529–533, 2015.

[3] Pygame Community, *Pygame Documentation*, `https://www.pygame.org/docs/`

[4] EcoForge Project Source Code, Python implementation files including `main.py`, `engine/simulate.py`, `agents/minimax.py`, `agents/monte_carlo.py`, and `agents/dqn_agent.py`.