TED University

Faculty of Computer Engineering

CMPE 491 – Analysis Report

by

Emir Irkılata, Caner Aras, Oğuzhan Özkaya

28.11.2025

1. Introduction

This analysis document defines the conceptual model, requirements, and constraints of MoonAI, a research-oriented computational framework developed to study and evaluate evolutionary algorithms and neural network evolution in controlled, synthetic environments. The primary purpose of the project is not to simulate ecological systems, but to create a structured, flexible, and data-generating experimental platform in which evolutionary computation methods can be observed, measured, and compared.

To achieve this, MoonAI incorporates a simplified predator–prey simulation as a testbed scenario used solely to expose evolutionary algorithms to dynamic and competitive conditions. This environment acts as an experimental benchmark, enabling the analysis of how genetic representations, mutation strategies, and topology-evolving neural networks behave under pressure, adapt over time, and influence learning efficiency.

The analysis presented here refines the initial specification and formalizes the requirements gathered during the proposal and requirement stages. The document establishes a clear foundation for subsequent architectural and implementation of work.

2. Current System

There is no existing system that fulfills MoonAI's research goals. While agent-based simulations and predator–prey models exist, their objectives differ fundamentally from MoonAI's purpose.

Limitations in Available Systems

- They focus on ecological accuracy, not the analysis of evolutionary computation techniques.
- They use predefined behaviors instead of neuroevolutionary driven decision-making.
- They lack comprehensive control over genetic encodings, mutation operators, and evolutionary parameters.
- They rarely support topology-evolving neural networks such as NEAT.
- They do not provide GPU-accelerated computation for large experimental populations.
- They are not designed to produce structured datasets for machine learning research.
- They do not meet academic requirements for transparency, ethical compliance, or reproducibility.

Conclusion

Because no existing software combines

- a controllable artificial environment
- an extensible evolutionary algorithm framework

- neural topology evolution
- research-grade data generation
- real-time visualization
- GPU-accelerated processing

The MoonAI system must be developed as a new, custom research platform.

3. Proposed System

3.1 Overview

MoonAI is designed as a modular and extensible experimental platform for analyzing evolutionary algorithms and neural network evolution. The predator–prey environment included in the system serves purely as a synthetic benchmark, providing a dynamic context in which algorithmic behaviors can be evaluated quantitatively and qualitatively. It is intentionally simplified and mechanically focused, as its purpose is not biological realism but controlled experimentation.

The system consists of several core components:

- Experimental Simulation Environment
- A simplified 2D environment that creates pressure, competition, and resource constraints so that evolutionary algorithms can be tested in a dynamic setting. The environment is minimal by design and only functions as a scenario generator for algorithm behavior.
- Evolutionary Algorithm Framework

MoonAI supports genetic and evolutionary process modeling through:

- genome representations configurable by the user,
- mutation and crossover operations,
- selection and reproduction strategies,
- evolving behavioral policies encoded in neural structures.

The framework enables systematic experimentation with different evolutionary configurations.

Neuroevolution of Augmenting Topologies Based Controller

agents in the test environment are controlled by neural networks whose

- weights
- Topologies
- architectural complexity

are altered across generations by the Neuroevolution of Augmenting Topologies algorithm. This allows direct study of structural innovation, complexity growth, evolution-driven feature extraction, and emergent behavioral strategies.

GPU-Accelerated Computation Layer

CUDA is employed to accelerate

- fitness evaluation
- evolutionary operations
- network inference
- large population updates

This ensures experiments with significantly larger evolutionary populations.

Real-Time Visualization

SFML-based visualization is used not to model ecological accuracy but to

- observe agent behavior
- visually track algorithm evolution
- verify system correctness
- support debugging and result interpretation

Data Generation and Analysis

The system logs

- evolutionary statistics
- genome histories
- neural architectures
- population metrics
- performance curves over generations

Logs are exported for further inspection using Python tools such as Pandas and Matplotlib.

Configurability and Extensibility

Researchers can adjust

- simulation parameters
- evolutionary parameters
- agent definitions
- fitness metrics
- environmental complexity

This enables MoonAI to be used as a flexible research platform for future algorithmic and computational studies.

3.2 Functional Requirements

3.2.1 Simulation Environment Requirements

FR-1 — Provide a Time-Stepped 2D Simulation Environment

Description:
The system shall create a configurable two-dimensional environment in which agents interact over discrete time steps.
Rationale:
A dynamic environment is necessary to impose selective pressure and evaluate evolutionary algorithms.
Acceptance Criteria:

- The simulation updates all agents every time step.
- Environmental size and physical constraints can be configured by the user.
- The system ensures deterministic updates when the same seed is used.

FR-2 — Support Environmental Entities and Interactions

Description:
The system shall manage all environment-related entities such as obstacles, resource nodes, and spawn areas.
Rationale:
Environmental complexity influences agent adaptation and must be controllable.
Acceptance Criteria:

- Entities can be created, modified, or disabled via configuration.
- Agents correctly detect and respond to environmental entities.

3.2.2 Agent Modeling Requirements

FR-3 — Implement Predator and Prey Agent Types

Description:
The system shall support at least two agent classes (predator and prey), each with configurable attributes (speed, vision, stamina, reproduction rate).
Rationale:

Diversity between roles enables meaningful evolutionary dynamics.
Acceptance Criteria:

- Agent attributes can be defined in configuration files.
- Predators and prey behave according to their roles (e.g., chase vs. evade).

FR-4 — Neural-Network-Based Behavior Control

Description:
Each agent shall be controlled by a neural network whose weights and topology can evolve over generations.
Rationale:
Allows observing the emergence of adaptive decision-making.
Acceptance Criteria:

- Networks receive perception inputs and produce motor/action outputs.
- Network structure is updated according to NEAT evolutionary rules.

3.2.3 Evolutionary Algorithm Requirements

FR-5 — Implement NEAT for Evolutionary Optimization

Description:
The system shall implement the NeuroEvolution of Augmenting Topologies (NEAT) algorithm to evolve neural network structures and weights.
Rationale:
NEAT enables the study of structural innovation and complexity growth.
Acceptance Criteria:

- Genome encoding for nodes, connections, and innovations is supported.
- Mutation, crossover, and speciation are correctly performed.

FR-6 — Evaluate Fitness for All Agents

Description:
The system shall compute a fitness score for every agent at the end of each generation.
Rationale:
Fitness evaluation drives the selection process in evolution.
Acceptance Criteria:

- Fitness metrics can be configured (survival time, energy gained, kills, etc.).
- Fitness values are logged and used in selection.

FR-7 — Perform Selection, Reproduction, and Population Update

Description:
The system shall generate the next generation by selecting parents, applying reproduction operators, and replacing or updating the population.
Rationale:
Ensures proper evolutionary dynamics.
Acceptance Criteria:

- Supports elitism, tournament selection, and mutation strategies.
- Population size remains consistent unless user-configured otherwise.

3.2.4 GPU and Performance Requirements

FR-8 — GPU-Accelerated Fitness and Simulation Computation

Description:
The system shall use CUDA for accelerating agent evaluations, neural inference, and evolutionary operations.
Rationale:
Allows scaling experiments to large populations.
Acceptance Criteria:

- GPU execution is used for at least fitness evaluation and forward passes.
- The system falls back to CPU mode when no GPU is available.

3.2.5 Visualization and Interaction Requirements

FR-9 — Real-Time Simulation Visualization

Description:
The system shall provide an SFML-based visualization to observe agent behavior and verify correctness.
Rationale:
Visualization aids debugging, validation, and interpretation.
Acceptance Criteria:

- Predators, prey, and environment elements are rendered in real time.
- Users can pause, speed up, or step through the simulation.

FR-10 — User Configuration Interface

Description:
The system shall allow users to adjust evolutionary parameters, environmental settings, and simulation properties.
Rationale:
Flexibility is central to MoonAI's research purpose.
Acceptance Criteria:

- Configurable via JSON/TOML/YAML files or command-line arguments.
- The system loads configurations at runtime without modifications to the code.

3.2.6 Data Logging and Output Requirements

FR-11 — Log Evolutionary and Simulation Data

Description:
The system shall record population statistics, network topologies, fitness distributions, and genome histories.
Rationale:
Data is required for research analysis and reproducibility.
Acceptance Criteria:

- Logs include timestamps, generation numbers, and all relevant metrics.
- Logs are exportable to formats readable by Python (CSV/JSON).

FR-12 — Export Data for External Analysis

Description:
The system shall provide structured output suitable for Python-based tools (Pandas, Matplotlib).
Rationale:
Enables external visualization and statistical evaluation.
Acceptance Criteria:

- Data is stored in standardized formats.
- Exported logs contain all necessary fields without additional preprocessing.

3.3 Nonfunctional Requirements

3.3.1 Performance Requirements

NFR-1 — Real-Time Simulation Capability

Description:
The system shall support real-time execution of the simulation with a minimum stable framerate under typical experimental loads.
Acceptance Criteria:

- The system maintains at least 30 FPS with several hundred agents on a GPU-enabled machine.
- Performance degradation is gradual rather than catastrophic when population size increases.

NFR-2 — Efficient GPU Utilization

Description:
The CUDA backend shall accelerate at least fitness evaluation and neural inference.
Acceptance Criteria:

- GPU computation must be measurably faster than CPU fallback mode.
- Long-running experiments must complete without memory leaks or GPU resource exhaustion.

3.3.2 Scalability Requirements

NFR-3 — Expandable Agent and Environment Models

Description:
The system shall support the addition of new agent types, environmental rules, and evolutionary operators without architectural changes.
Acceptance Criteria:

- New agent classes can be added by extending base interfaces.
- No major refactoring is required when increasing simulation complexity.

NFR-4 — Large-Population Support

Description:
The architecture shall allow experiments involving large evolutionary populations.
Acceptance Criteria:

- System supports scaling up to thousands of agents with proportional performance decline.
- Data logging and evolutionary operations remain stable at large scales.

3.3.3 Maintainability Requirements

NFR-5 — Modular, Object-Oriented Architecture

Description:
The system shall be implemented using modular, object-oriented design principles.
Acceptance Criteria:

- Components such as simulation, evolution, logging, and visualization are separate modules.
- Internal classes follow single-responsibility principles.

NFR-6 — Readable and Documented Codebase

Description:
The codebase shall include sufficient documentation for future researchers.
Acceptance Criteria:

- All public APIs, data structures, and configuration parameters are documented.
- Inline comments exist for complex algorithms (NEAT, CUDA kernels, etc.).

3.3.4 Usability Requirements

NFR-7 — Clear and Interpretable Visualization

Description:
The simulation visualization must clearly reflect agent behavior and environment structure.
Acceptance Criteria:

- Predators, prey, and their states must be visually distinguishable.
- Users can interpret agent paths, collisions, and interactions without ambiguity.

NFR-8 — Accessible Configuration Management

Description:
Experiment parameters must be easily adjustable by researchers.
Acceptance Criteria:

- Parameters can be edited without recompiling the software.
- Invalid configuration values generate clear error messages.

3.3.5 Portability Requirements

NFR-9 — Cross-Platform Execution

Description:
The system shall run on both Windows and Linux with minimal dependency issues.
Acceptance Criteria:

- Only portable libraries (C++, SFML, CUDA, Python) are used.
- Build scripts or instructions exist for both platforms.

3.3.6 Reliability Requirements

NFR-10 — Deterministic Experimental Reproducibility

Description:
The system shall support deterministic behavior when using a fixed random seed.
Acceptance Criteria:

- Given identical inputs and seed values, evolutionary outcomes and agent trajectories remain the same.
- Logs match across repeated runs.

NFR-11 — Fault Tolerance During Long Simulations

Description:
Long-running evolutionary experiments shall not crash due to memory leaks or unstable processes.
Acceptance Criteria:

- System successfully completes multi-hour or multi-day runs.
- Automatic memory management prevents resource exhaustion.

3.3.7 Security Requirements

NFR-12 — Safe Execution Environment

Description:
The system shall follow standard software safety practices to avoid unintended harmful operations.
Acceptance Criteria:

- No unbounded memory operations in CUDA kernels.
- System handles invalid inputs without undefined behavior

3.4 Pseudo Requirements

3.4.1 Technology Constraints

PR-1 — Programming Language Constraint

The system shall be implemented primarily in **C++** due to performance requirements and compatibility with SFML and CUDA.

PR-2 — GPU Acceleration Through CUDA

The project requires **NVIDIA CUDA** for GPU-accelerated computation. Systems without CUDA-capable GPUs may experience degraded performance.

PR-3 — Visualization Using SFML

All real-time visualizations must be implemented using the **Simple and Fast Multimedia Library (SFML)** for consistency and cross-platform rendering.

3.4.2 Platform and Compatibility Constraints

PR-4 — Operating System Compatibility

The system shall be runnable on **Windows** and **Linux**, the two platforms supported by the development team and course requirements.

PR-5 — Compiler Requirements

The project must compile on **GCC / Clang (Linux)** and **MSVC (Windows)** with C++17 or later.

PR-6 — Python-Based Data Analysis Requirement

Exported logs must be compatible with analysis tools such as **Python, Pandas, and Matplotlib**, as mandated by the project's research workflow.

3.4.3 Academic and Documentation Constraints

PR-7 — Open-Source Code Requirement

The project must publish non-sensitive components as open-source according to the course guidelines and supervisor expectations.

PR-8 — Report and Web Page Publication

All formal deliverables — proposal, specification, analysis report, design report, and final documentation — must be published on the project's **GitHub Pages website**.

PR-9 — Version Control

The team must use **Git and GitHub** to maintain version history, ensure transparency, and support reproducibility.

3.4.4 Performance and Resource Constraints

PR-10 — GPU Resource Limitation

Due to limited GPU hardware availability, population size and simulation complexity must be scaled to avoid exceeding hardware limits.

PR-11 — Real-Time Rendering Budget

The SFML visualization must maintain an interactive framerate (ideally ≥ 30 FPS), limiting the maximum number of agents rendered at once.

3.4.5 Ethical and Professional Constraints

PR-12 — Ethical Compliance

The development process must adhere to the **ACM and IEEE ethics codes**, ensuring fairness, transparency, and academic honesty.

PR-13 — Reproducibility and Data Integrity

All experiments must be reproducible; random seeds must be logged, and experiment metadata must be preserved.

3.5 System Models

This section presents the analysis-level models that describe how the MoonAI system behaves, how users interact with it, and how major components collaborate during execution. These models help formalize the system structure before architectural design.

3.5.1 Scenarios

Scenarios describe typical interactions or workflows within the system. They illustrate how the system responds under normal operating conditions.

Scenario 1 — Running an Evolutionary Simulation

Goal:
A researcher wants to run a full predator–prey evolutionary experiment.

Primary Actor:
Researcher (User)

Preconditions:

- Configuration file is provided.
- System initializes successfully.

Main Flow:

1. User launches the MoonAI system.
2. The system loads simulation and evolutionary parameters (environment size, population size, NEAT parameters, fitness function).
3. The system constructs the initial population of predators and prey.
4. Simulation begins and agents interact in the 2D environment.
5. The system evaluates fitness for each agent.

6. NEAT performs mutation, crossover, and speciation to produce a new population.
7. Logs and performance metrics are saved.
8. Simulation continues until the termination condition is met (max generations or fitness threshold).

Postconditions:

- Results are exported for analysis.
- Population history and genome data are logged.

Scenario 2 — Observing Real-Time Behavior

Goal:
A researcher monitors agent behavior during evolution.

Flow:

1. User enables visualization.
2. SFML renders the environment and agents.
3. Agent neural decisions are executed and displayed.
4. The user pauses, speeds up, or steps through frames.
5. User observes emergent strategies such as chasing, escaping, and exploring.

Scenario 3 — Data Extraction and Offline Analysis

Flow:

1. User loads exported CSV/JSON logs into Python.
2. Pandas reads generation statistics.
3. User visualizes fitness curves and topology changes with Matplotlib.
4. Results are used to compare different evolutionary settings.

3.5.2 Use Case Model

Below are the core use cases in analysis form.

Use Case UC-1: Start Simulation

Primary Actor: Researcher
Goal: Start a complete evolutionary run
Main Success Flow:

1. User provides config file.
2. System loads parameters.
3. System initializes environment and population.
4. Simulation begins.

Use Case UC-2: Configure Parameters

Primary Actor: Researcher
Goal: Modify simulation/evolution parameters
Main Flow:

1. User edits configuration file (JSON/TOML/YAML).
2. System validates parameters.
3. Parameters apply at the start of the run.

Use Case UC-3: Visualize Simulation

Primary Actor: Researcher
Goal: Observe in real time
Flow:

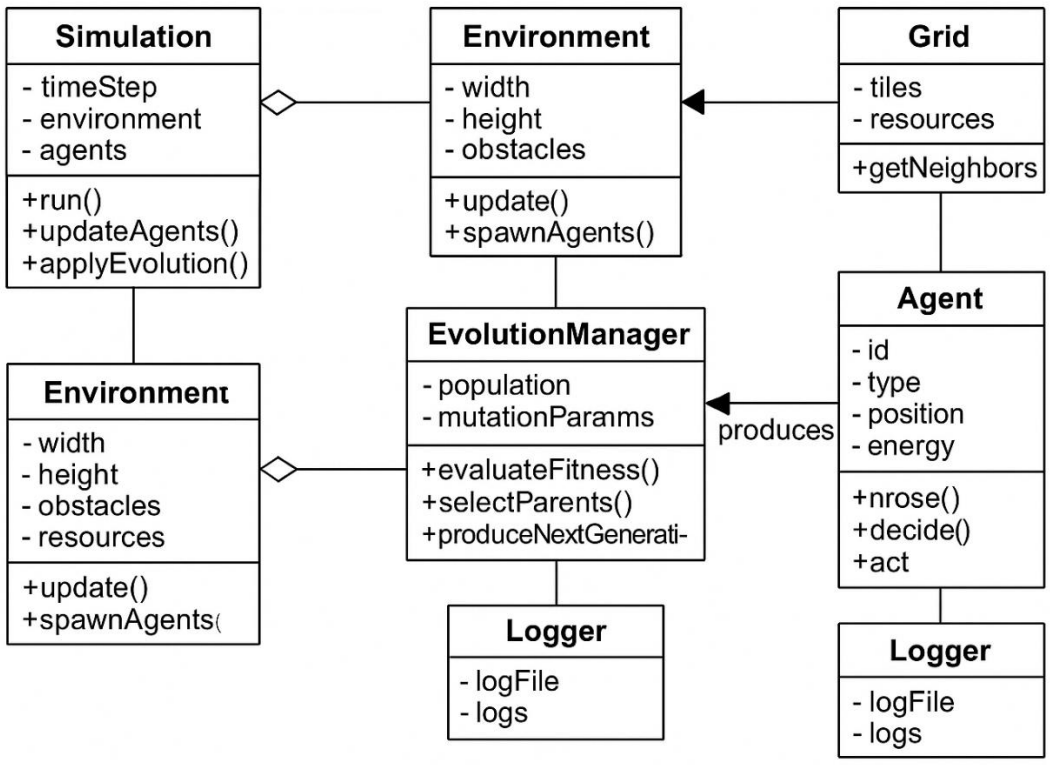
1. User enables visualization mode.
2. System renders agents and environment.
3. User controls time (pause, step, speed).

Use Case UC-4: Export Logs

Primary Actor: Researcher
Goal: Save experiment results
Flow:

1. System collects metrics.
2. Logs are written to CSV/JSON.
3. User uses Python for analysis.

3.5.3 Object and Class Model



Primary Classes and Responsibilities

1. Simulation

Responsibilities:

- Manage global simulation loop.
- Update all agents every time step.

- Handle environment interactions and termination conditions.

Key Attributes:

- environment : Environment
- agents : List<Agent>
- timeStep : int

Relationships:

- Aggregates Environment and multiple Agent objects.

---

2. Environment

Responsibilities:

- Provide spatial context for agents.
- Manage obstacles, resources, and spawn points.

Key Attributes:

- width : float
- height : float
- entities : List<EnvironmentObject>

Relationships:

- Contains many EnvironmentObject.

---

3. Agent (abstract)

Responsibilities:

- Represent common logic for predator and prey agents.
- Provide perception, decision, and movement methods.

Key Attributes:

- position : Vector2D
- velocity : Vector2D

- energy : float
- visionRange : float
- brain : NeuralNetwork

Relationships:

- Inherited by Predator and Prey.
- Composed with NeuralNetwork.

---

4. Predator (extends Agent)

Responsibilities:

- Implement chase and attack behavior.

Key Attributes:

- attackRange : float

Relationships:

- Interacts with Prey via environment.

---

5. Prey (extends Agent)

Responsibilities:

- Implement evasion, reproduction, and survival behavior.

Key Attributes:

- reproductionRate : float

---

6. NeuralNetwork

Responsibilities:

- Perform inference on agent inputs.
- Feed-forward network with an evolving topology.

Key Attributes:

- nodes : List<NodeGene>
- connections : List<ConnectionGene>

---

7. Genome

Responsibilities:

- Encode NEAT-compatible genetic representation.

Key Attributes:

- nodeGenes : List<NodeGene>
- connectionGenes : List<ConnectionGene>
- fitness : float

Relationships:

- Associated with one Agent.
- Managed by EvolutionManager.

---

8. EvolutionManager

Responsibilities:

- Apply NEAT operations (mutation, crossover, speciation).
- Generate new populations.

Key Attributes:

- population : List<Genome>
- innovationHistory : InnovationTracker

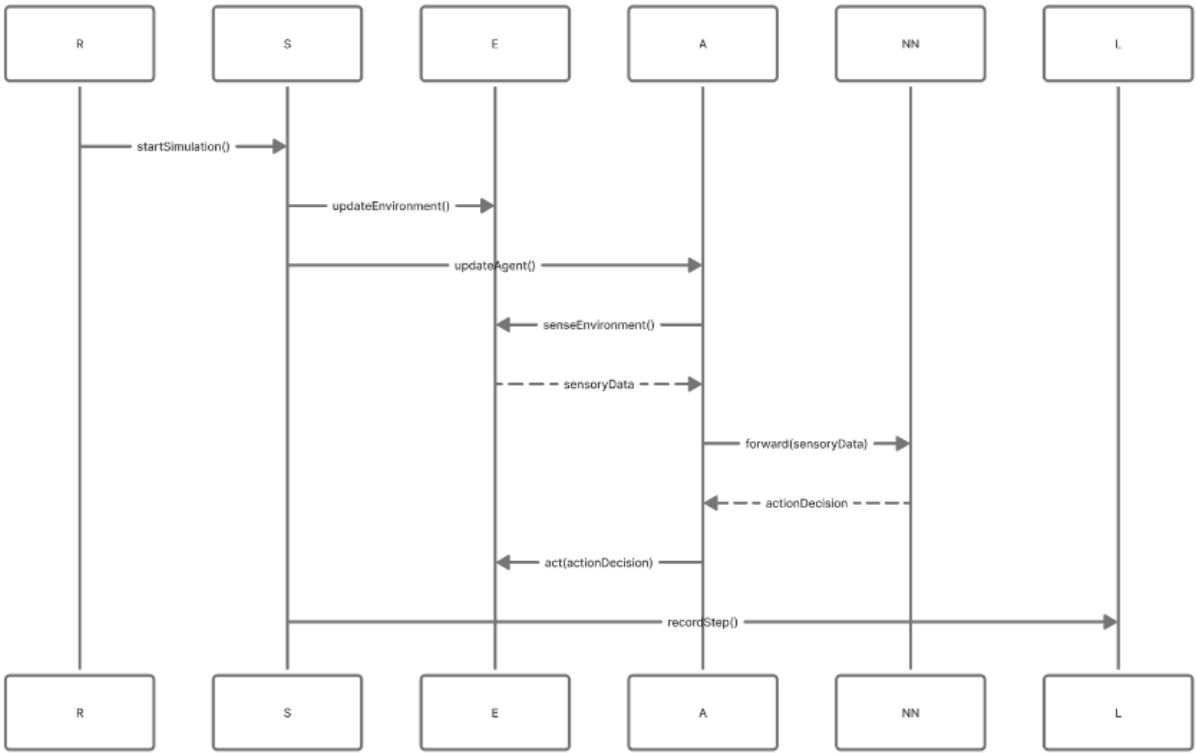
---

9. Logger

Responsibilities:

- Store simulation statistics, genomes, fitness, and neural structures.
- Export data to CSV/JSON for Python analysis.

3.5.4 Dynamic Models



The sequence diagram illustrates the **Agent Decision Loop** that occurs during a single simulation step in MoonAI. This loop represents how each agent perceives its surroundings, processes this information through its evolving neural controller, and performs an action in response. The interaction between the main system components is shown through ordered message exchanges.

During each simulation step, the **Simulation** initiates the update cycle by requesting the **Agent** to perform its decision process. The agent first gathers perceptual information from the **Environment** (e.g., nearby agents, obstacles, resources). This sensory data is then forwarded to the agent's **NeuralNetwork**, which computes an output representing the agent's behavioral decision. Based on this output, the agent executes an action in the environment (movement, chase, evade, reproduce, etc.). After all agents are updated, the **Logger** records the step results for later analysis.

This sequence demonstrates the core decision-making pipeline that drives the evolutionary dynamics of MoonAI, as the behavior encoded in neural networks evolves over generations and directly influences simulation outcomes.

Actor Definitions (For Diagram Letters)

Symbol	Meaning	Description
S	*Simulation*	Manages the global simulation loop and updates all agents every time step.
E	*Environment*	Provides sensory data to agents and receives their actions.
A	*Agent*	Represents a predator or prey agent making decisions each step.
NN	*NeuralNetwork*	The agent's NEAT-evolving controller that computes decisions.
L	*Logger*	Records simulation statistics, fitness values, and network data.

3.5.5 User Interface – Navigational Paths and Screen Mock-ups.

A. UI Navigation Flow

Main Navigation Structure

Main Menu

├— Simulation Configuration

├— Run Simulation

├— Live Visualization

└— Data & Logs

  ├— Fitness Trends

  ├— Population Statistics

  └— Export Logs

Description of Navigation Flow

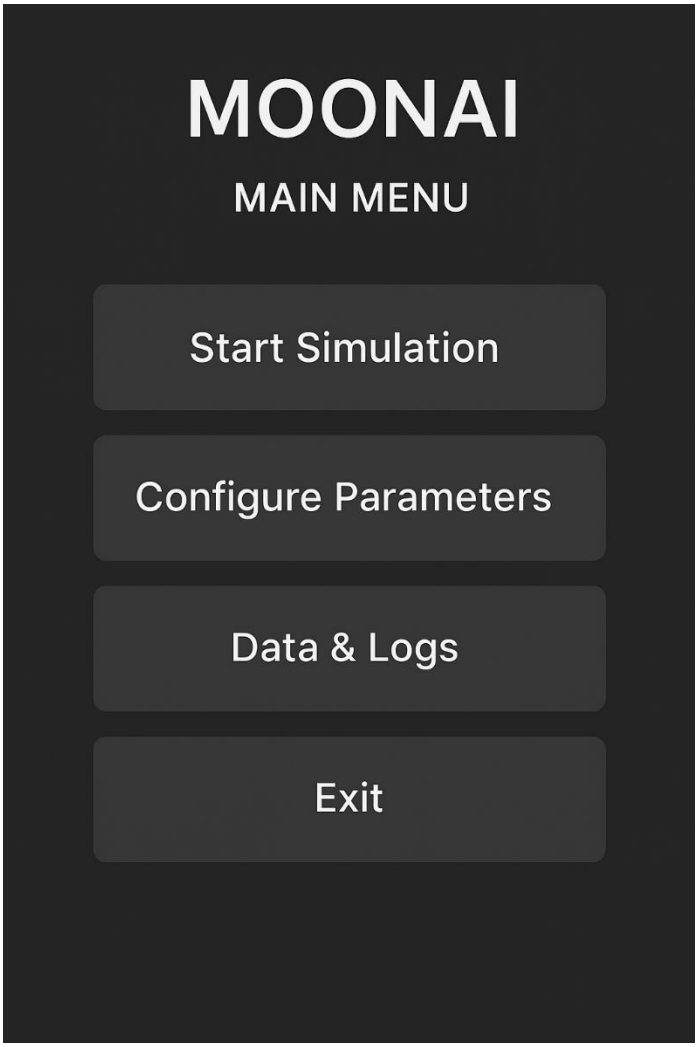
1. **Main Menu → Simulation Configuration**
   The researcher sets evolutionary parameters, environment size, NEAT mutation rates, agent counts, and initial conditions.
2. **Main Menu → Run Simulation**
   Starts a simulation instance with the selected configuration. The system validates inputs before launching the run.
3. **Run Simulation → Live Visualization**
   Real-time SFML rendering displays agents, positions, energy levels, and state transitions.
4. **Main Menu → Data & Logs**
   Displays exported statistics such as:
   - fitness per generation
   - species clusters
   - network topology growth
   - population curves
   - timing and performance metrics
5. **Data & Logs → Export Logs**
   Researchers export CSV/JSON logs for Python analysis.

B. Screen Mock-up Descriptions

1. Main Menu Screen

Purpose: Entry point for all system functions.
Elements:

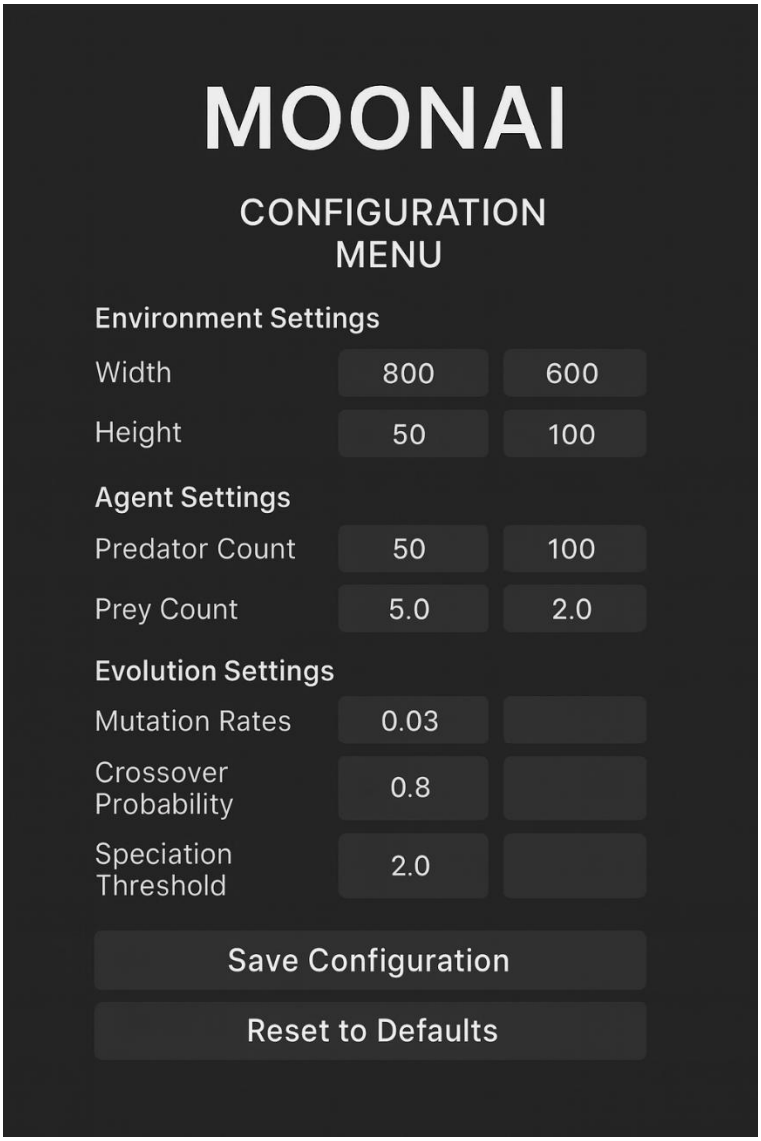
- *Start Simulation* button
- *Configure Parameters* button
- *Data & Logs* button
- *Exit*



2. Configuration Screen

Purpose: Define all experimental parameters.
Sections:

- **Environment Settings**: width, height, resource density
- **Agent Settings**: predator count, prey count, initial energy
- **Evolution Settings**: mutation rates, crossover probability, speciation threshold
- **Simulation Settings**: time step, duration, random seed
  Actions: Save configuration / Reset to defaults

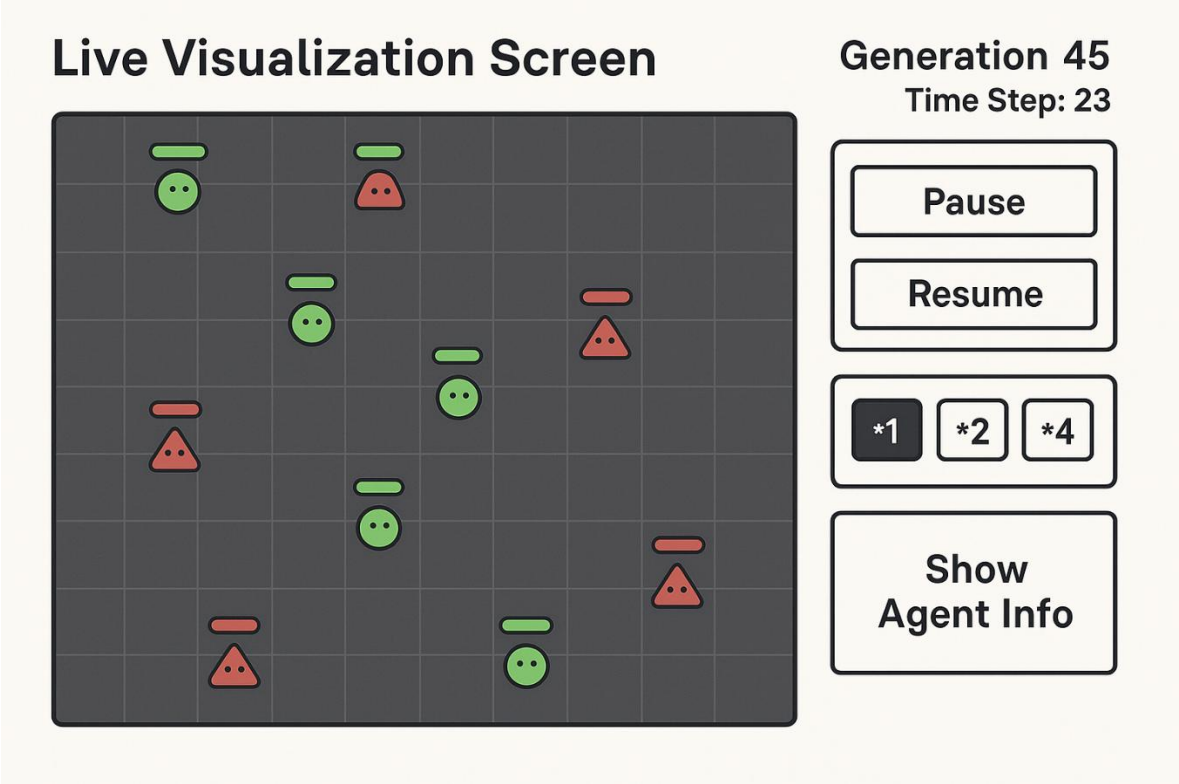


3. Live Visualization Screen

Purpose: Observe simulation behavior visually.
Displayed:

- 2D grid
- Predator and prey icons
- Energy bars
- Generation number & time step
  Controls:
- Pause / Resume

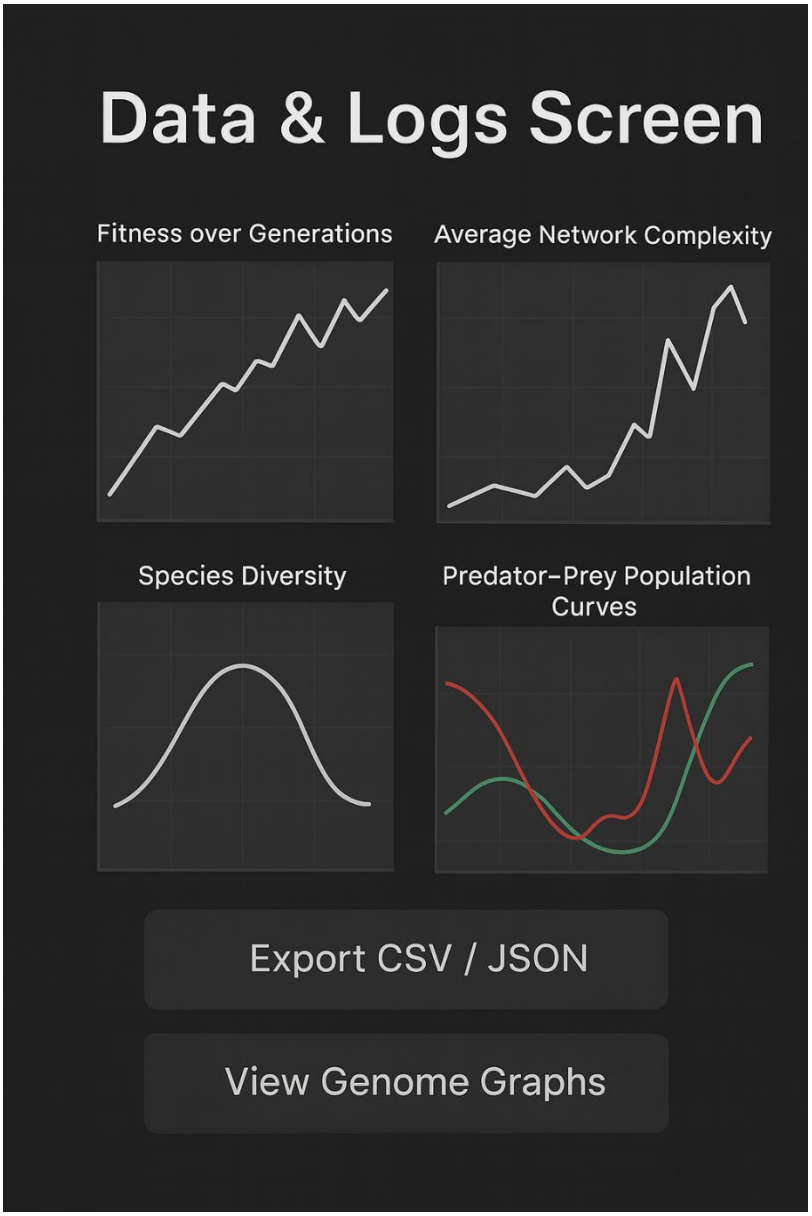
- Speed ×1 / ×2 / ×4
- Show/Hide agent info



4. Data & Logs Screen

Purpose: Present simulation statistics & allow data export.
Displayed graphs:

- Fitness over generations
- Species diversity
- Average network complexity
- Predator–prey population curves
  Buttons:
- Export CSV / JSON
- View Genome Graphs
- Open Folder



C. Key UI Design Principles

- **Minimalism:** Avoid unnecessary visual complexity since MoonAI is research-focused.
- **Transparency:** All simulation and evolutionary parameters must be visible and modifiable.
- **Reproducibility:** UI actions generate versioned config files.

- **Real-time interpretability:** Visualization supports debugging and sanity-checking evolving behaviors.

4. Glossary

A

Agent

- An autonomous entity in the simulation (predator or prey) that senses the environment, evaluates inputs using a neural network, and performs actions such as moving, chasing, escaping, or consuming resources.

Action Decision

- The output produced by the agent's neural controller, representing movement direction, speed, or other behavior parameters.

C

Crossover

- A genetic operation in evolutionary algorithms where two parent genomes combine to form an offspring genome, inheriting characteristics from both.

CUDA (Compute Unified Device Architecture)

- NVIDIA's GPU programming framework used in MoonAI to accelerate fitness evaluation, neural inference, and evolutionary operations.

E

- **Environment**
  A simplified 2D simulated world containing obstacles, resources, and spatial constraints where agents interact and evolve.
- **Evolutionary Algorithm (EA)**
  A computational method inspired by biological evolution, used to optimize agent behavior through mutation, crossover, and selection.

F

Fitness

- A quantitative measure of an agent's performance (e.g., survival duration, prey hunted, energy maintained). Used by the evolution engine to select parents for reproduction.

G

Genome

- A structured representation of an individual's neural architecture in the NEAT algorithm, including node genes, connection genes, and mutation history.

Generation

- A complete cycle of evaluating a population, selecting parents, and producing a new set of genomes.

L

Logger

- Component responsible for recording simulation data, fitness values, neural architectures, and exporting structured logs for post-analysis.

M

Mutation

- A stochastic alteration to a genome—such as adding a node, altering a weight, or adding a connection—used to introduce diversity in evolution.

Mini-map

- A simplified visualization showing global agent distribution, heatmaps, or resource density.

N

NEAT (NeuroEvolution of Augmenting Topologies)

- An evolutionary algorithm that evolves both neural network weights and topologies, allowing structural growth over time.

Neural Network (NN)

- The agent's controller, responsible for processing sensory inputs and producing behavioral outputs.

P

Population

- The total set of genomes or agents in one evolutionary generation.

Predator / Prey

- Two predefined agent categories with distinct behavioral roles:
  • Predators chase and attack prey
  • Prey attempt to evade and survive

S

Simulation Step

- A single iteration of the simulation loop where all agents sense, decide, and act.

Speciation

- A NEAT mechanism that groups genetically similar individuals into species to protect innovation.

SFML (Simple and Fast Multimedia Library)

- The visualization library used to render the simulation in real time.
-

T

Topology (Neural Topology)

- The structure of a neural network, including nodes and connections. Evolves dynamically in NEAT.

Time Step

- A discrete unit of simulated time.

V

Visualization Manager

- The component that manages real-time rendering of agents, the environment, overlays, and UI elements.

5. References

Bruegge, B., & Dutoit, A. H. (2004). *Object-Oriented Software Engineering: Using UML, Patterns, and Java* (2nd ed.). Prentice Hall.

Stanley, K. O., & Miikkulainen, R. (2002). **Evolving Neural Networks Through Augmenting Topologies**. *Evolutionary Computation*, 10(2), 99–127.

Stanley, K. O., Bryant, B. D., & Miikkulainen, R. (2003). **Evolving Adaptive Neural Networks with and without Adaptive Synapses**. *Proceedings of the 2003 IEEE Congress on Evolutionary Computation*, 2557–2564.

Floreano, D., & Mattiussi, C. (2008). *Bio-Inspired Artificial Intelligence: Theories, Methods, and Technologies*. MIT Press.

Wilensky, U. (1999). *NetLogo*. Center for Connected Learning and Computer-Based Modeling, Northwestern University.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. MIT Press.

SFML Development Team. (2023). *Simple and Fast Multimedia Library – Official Documentation*.

NVIDIA Corporation. (2023). *CUDA Toolkit Documentation*.

Holland, J. H. (1975). *Adaptation in Natural and Artificial Systems*. University of Michigan Press.