# Organic Learning Architecture (OLA) – Capability Summary and Snake-Game Validation

Author: Payton Miller

Date: 2025-11-14

## 1. Purpose of this Document

This document summarizes the demonstrated capabilities of the Organic Learning Architecture (OLA), with a specific focus on empirical evidence that:

1) OLA can be trained without gradient descent,
2) OLA learns non-trivial policies in a dynamic environment, and
3) the Snake game experiment provides concrete validation that OLA improves its behavior over time via evolutionary updates to its genomes.

The goal is not hype. The goal is to provide a technical, falsifiable record of what OLA has already done.

## 2. High-Level Overview of OLA

### 2.1. Core Idea

OLA is an evolutionary, genome-based controller. Instead of a single set of weights updated by backpropagation, OLA maintains a population of discrete “genomes.” Each genome encodes a small network (or logic-like mapping) from observations to actions. Genomes:

- Receive observations from the environment (e.g., Snake board state),
- Produce actions,
- Accumulate reward over an episode,
- Are reinforced, mutated, or discarded based on performance and “trust.”

In short, OLA is a real-time, non-gradient credit-assignment system that:

- Tracks performance over time,
- Promotes genomes that work,
- Mutates genomes that underperform,
- Maintains a small elite set that encodes the current “best-known” policy.

### 2.2. Key Components

At the stage demonstrated in Snake:

- Genome:
  - A set of weight and bias tensors mapping input to a small output vector.
  - Associated metadata: trust, age, usage stats.
  - Mutated via stochastic perturbations when selected for exploration.

- Population:
  - Fixed-size set of genomes.
  - Includes elites (high-trust, high-performance).
  - Underperforming genomes can be replaced or heavily mutated.

- Trust Mechanism:
  - Scalar per genome that increases when the genome performs well under current reward structure and decreases when it performs poorly.
  - Used to:
    - Select which genomes to try more frequently,
    - Decide when to mutate,
    - Decide when a genome should be retired or replaced.

- Evolution Loop:
  - For each episode:
    - Pick a genome (weighted by trust and exploration logic),
    - Run the full episode with that genome controlling the agent,
    - Compute total reward,
    - Update trust,
    - Potentially mutate or replace the genome based on how it did relative to prior performance.

### 2.3. Training Style

OLA is not trained in traditional “epochs” with gradient descent. Instead:

- It operates in episode-based loops (e.g., thousands of Snake games).
- Each episode is a real roll-out in the environment.
- Feedback (reward) is used to:
  - Adjust trust,
  - Apply mutations,
  - Maintain or replace elites.
- There is no backprop; credit assignment is implicit via evolutionary selection.

This is crucial: the Snake experiment shows that this style of training is sufficient to learn a competent policy in a non-trivial environment.

## 3. Snake Game Setup

### 3.1. Environment Summary

The Snake game used for validation has the following properties:

- Grid-based environment of fixed size (e.g., 10x10 or similar).
- Agent: a snake with:
  - A head position,
  - A body trail that grows when food is eaten,
  - Terminal conditions:
    - Collision with walls (boundaries),
    - Optional self-collision (enabled in later phases).
- Food:
  - Spawns at random grid locations not currently occupied by the snake.
  - Eating food:
    - Increases the snake length,
    - Triggers a strong positive reward,
    - Resets the food position.

The agent must learn to:

- Seek food,
- Avoid dying immediately to walls,
- Navigate the board without a global map,
- Exploit a 5x5 local observation window efficiently.

### 3.2. Observation Space (51 Features)

The observation vector given to OLA’s genomes is not raw pixels. Instead, it is a structured, low-dimensional state describing the local environment around the snake’s head and some global directional cues.

At the phase where learning clearly appeared, the obs vector had length 51, including:

- Local 5x5 grid around the head (flattened 25 cells):
  - Encoded as:
    - 0 = empty,
    - 1 = wall (outside grid bounds),
    - 2 = snake body (including head),
    - 3 = food.
  - This allows the controller to “see”:
    - Immediate nearby walls,
    - Its own body,
    - Nearby food.

- Direction encoding:
  - One-hot or similar encoding of current heading:
    - Up / Down / Left / Right.
  - This gives the model a notion of orientation.

- Food direction and distance:
  - Coarse information about where food is relative to the head.
  - For example:
    - dx = food_x - head_x,
    - dy = food_y - head_y,
    - Possibly normalized or clipped,
    - Additional “is food ahead” / “is food left/right” style flags.

- Other auxiliary features:
  - Game progress (e.g., steps taken),
  - Possibly “stuck” indicators or distance deltas.

All combined, this ~51-D vector encodes enough spatial structure that a non-trivial policy is possible but not trivial. A random controller will almost always die quickly and rarely eat food.

3.3. Action Space
The action space is discrete:

- 0 = turn left
- 1 = go straight
- 2 = turn right

The snake’s current direction plus this relative action determines the new direction (e.g., from Up, left means Left, straight means Up, right means Right). This relative control is standard in many Snake RL setups.

3.4. Reward Structure
The final successful configuration used a reward structure of roughly:

- +R_food for eating food (large, strongly positive),
- -R_death for dying (negative, but smaller in magnitude than the total potential food reward over an episode),
- No per-step living bonus in the final successful run (earlier versions included a small living bonus, but it led to “stalling” behaviors).

The key tuning decisions were:

1) Food reward must be high enough that evolving toward food-seeking behavior is favored.
2) Death penalty must be strong enough to discourage immediate suicide or reckless wall-charging, but not so huge that it dominates the signal.
3) Removing the living bonus eliminated incentives to “stall” without pursuing food.

3.5. Training Protocol
Typical runs:

- Training in batches of 5,000 episodes at a time.
- Occasionally saving checkpoints (e.g., JSON files containing genomes, their trust, and weights).
- Restarting training from checkpoints to continue improvement.
- Logging:
  - Per-episode:
    - success (1 if ate at least one food, 0 otherwise),
    - eaten (number of food items eaten),
    - death_type (wall vs. starvation),
    - steps,
    - total_reward,
    - active genome ID,

    - trust for that genome.
  - Summaries every N=200 episodes:
    - avg_success_200,
    - avg_steps_200,
    - avg_total_reward_200,
    - best_genome_id,
    - best_genome_trust.

This level of logging is sufficient to see clear trends in performance.

## 4. Empirical Evidence: Learning in Snake

### 4.1. Baseline Behavior (Early Phases)
In early configurations (before tuning rewards, observation space, and trust dynamics), typical characteristics were:

- avg_success_200 near 0.0–0.1 (often 0.0–0.05),
- avg_total_reward_200 strongly negative,
- Per-episode logs dominated by:
  - eaten = 0,
  - early deaths (wall collisions),
  - low or decaying trust scores.

Behaviorally:
- The snake would frequently crash into walls,
- Food pickups were rare and almost always accidental,
- No clear directional bias toward food was visible.

### 4.2. After Reward and Signal Fixes
Once the following were in place:

- 5x5 local grid observation,
- Food distance / direction features,
- Properly balanced:
  - High food reward,
  - Moderate death penalty,
  - No per-step living bonus,
- Warm-up period to accumulate useful trust statistics,
- Evolutionary updates that:
  - Mutated underperformers,
  - Preserved and sometimes retired elites,

the metrics changed dramatically.

Example: In one of the late-phase runs (post-fix), summary logs over 5,000 episodes showed:

- avg_success_200 stabilizing around 0.5–0.7;
- avg_total_reward_200 consistently positive;
- best_genome_trust regularly hitting 2.0 (capped), indicating:
  - Repeated episodes with strong performance,
  - Stable reinforcement of high-performing genomes.

Per-episode logs from those runs show many episodes with:

- eaten = 1, 2, 3, and higher,
- Multiple episodes with:

  - eaten ≈ 6–10+,
  - long survival times,
  - high positive reward.

Critically, this was achieved with:
- No gradient descent,
- No backprop,
- No replay buffer,
- No explicit value function,
- No policy gradient.

All credit assignment came from:
- Evolutionary selection on total episode reward,
- Trust-based reinforcement,
- Mutation of genome parameters.

## 4.3. High-Performance Snapshots
From the user’s logs (representative example):

- In one 5K-episode session after tuning:
  - avg_success_200 peaked around ≈ 0.7–0.79 (i.e., ~70–79% of episodes in the last 200 had at least one food eaten).
  - The best genomes achieved runs with:
    - up to ≈ 10–12 food eaten in a single episode,
    - very high total_reward,
    - trust saturated at 2.0 (max).

This is not random chance. The distribution of eaten and reward over thousands of episodes shows:

- A shift from mostly 0-food games to many 1–3-food games,
- Regular high-food games (5+),
- Clearly non-random trajectories with purposeful navigation.

## 4.4. Non-Trivial Policy Indicators
Key behavioral indicators that the learned policy is non-trivial:

1) Systematic Food Seeking:
   - The agent repeatedly navigates toward food when it appears in various positions, not just in a fixed location.
   - High-performing genomes continue to do this across many episodes with different food spawns.

2) Wall Avoidance:
   - The snake survives long enough to chain multiple food pickups.
   - This implies avoiding immediate collision and using the 5x5 local observation effectively.

3) Policy Drift and Over-Training:
   - If training continues too long with fixed reward settings, performance can drift downward:
     - avg_success_200 can drop,
     - trust can redistribute to less optimal genomes.
   - This is a hallmark of a live evolutionary system:
     - The policy is not statically “locked.”
     - Continuous mutation + selection can both refine or degrade behavior depending on reward and mutation schedule.

## 4.5. Checkpointing and Stability
The introduction of automatic checkpointing demonstrated that:

- Training from scratch can require anywhere from ~5K to 25K+ episodes to hit a high-performing policy, depending on luck and mutation path.

- Once a strong policy is found and checkpointed:
  - Reloading the checkpoint and continuing training allows:
    - Further refinement,
    - Or controlled experimentation (e.g., changing reward structure or enabling self-collision) without losing the learned base policy.

This establishes that:
- OLA-based controllers are reproducibly improvable,
- Learned policies are persistent across runs via JSON checkpoints.

## 5. Why This Validates OLA Trainability

### 5.1. Core Claim
The Snake results validate the following claim:

“OLA, using only evolutionary updates over a population of genomes and a scalar trust mechanism—without any gradient descent—can learn a non-trivial control policy in a dynamic environment and improve systematically with experience.”

This is not a toy bandit or static mapping. Snake has:

- Delayed reward (reward only when food eaten or episode ends),
- Long-horizon credit assignment,
- Partial observability (5x5 local window),
- Stochasticity (random food positions, different episode start conditions),
- Temporal coupling (snake body length changes difficulty over time).

### 5.2. Comparison to Classic RL
Traditional RL on Snake usually uses:
- Gradient-based updates (Q-learning, DQN, actor-critic),
- Replay buffers,
- Neural networks trained with backprop,
- Carefully tuned learning rates and exploration schedules.

In the OLA experiment:

- There is no backpropagation.
- There is no Q-table or value function.
- The “policy” is encoded implicitly in the evolved genome weights.
- Credit assignment is done by:
  - Evaluating full-episode reward,
  - Increasing trust for high-performing genomes,
  - Mutating and replacing low-performing ones.

Yet, the system:
- Moves from random behavior to purposeful behavior,
- Achieves high success rates and multi-food episodes,
- Maintains this performance across thousands of episodes, subject to mutation dynamics.

### 5.3. Implications
The Snake results imply that:

1) OLA is capable of learning from sparse reward signals in a dynamic environment.
2) OLA’s evolutionary-trust framework can stabilize into high-performing controllers.
3) OLA is fast and lightweight enough to train thousands of episodes in minutes on consumer hardware, which makes it a viable alternative for continuous and on-device learning schemes.

## 6. Limitations and Open Issues

These results are promising but not magical. The Snake experiment also exposed:

- Sensitivity to reward shaping:
  - Too much death penalty or wrong living bonus leads to degenerate strategies.
- Sensitivity to mutation rate:
  - Excessive mutation can destroy good genomes,
  - Too little mutation can freeze sub-optimal policies.
- Possibility of performance drift:
  - Continuous blind mutation can reduce performance if not balanced by:
    - Elite preservation,
    - Retirement logic,
    - Careful trust dynamics.

These are engineering challenges, not fundamental barriers. They are the same class of issues faced by all evolutionary algorithms and online RL systems.

## 7. Conclusion

The Snake-game experiment provides concrete, empirical evidence that:

- OLA is a trainable architecture in practice, not just in theory.
- It can learn meaningful policies in a dynamic, partially observed environment using:
  - No gradients,
  - No backpropagation,
  - Only evolutionary genome selection and trust-based reinforcement.
- Performance metrics (success rate, total reward, food eaten) moved from “near-random” to “highly structured and competent” over the course of training.
- Behaviorally, the agent learned to:
  - Seek food,
  - Avoid walls,
  - Sustain long survival chains with multiple food pickups.

This validates the central premise that OLA can function as a real learning engine and not just a static heuristic system. Snake is the first fully closed-loop demonstration, but the same mechanisms can be extended to other domains (vision, audio, language, etc.) by:
- Redefining the observation space,
- Redefining the action space,
- Adapting the reward structure,
- Keeping the same evolutionary + trust core.

The key takeaway: OLA is not hypothetical. It has already learned.