Uiua: A Modern Array Language for Artificial Life Research

Noah Syrkis
IT University of Copenhagen
nobr@itu.dk

Abstract: Uiua[1], an APL-like stack-based array programming language, offers a powerful alternative for Artificial Life (ALife) research. Its concise syntax and efficient array operations enable rapid prototyping and complex simulations of bio-like systems. This abstract showcases Uiua's capabilities through a basic particle swarm simulation example and discusses its potential for ALife research.

Introduction

Artificial Life (ALife) research often involves complex simulations of bio-like systems, which require efficient manipulation of large datasets and intricate computational models. Traditional programming languages, while versatile, may not always provide the optimal balance of expressiveness and performance for such tasks. Uiua, an APL-like stack-based array language written in Rust, offers a compelling alternative for ALife researchers.

All operations operate on a global stack. For examples `+ + 1 2 3` pushes the values `3`, `2`, and `1` onto the stack while the left most `+` adds `3` to the sum of `1` and `2`. Uiua's syntax and semantics are designed to be intuitive and concise, allowing researchers to express complex algorithms with minimal code. Plugging these expressions into high level functional combinators, a subset of which is described in Table 1, can be used to create powerful and efficient ALife simulations.

`∧ fold`	`⍜ under`	`≡ rows`
Apply a function to aggregate arrays	Operate on a transformed array, then untransform it	Apply a function to each row of an array or arrays
`∧+ [1 2 3] 10` → `16`	`⍜+(×2) 1 5` → `11`	`≡∧+ [1_2 4_5] 0` → `3_9`

Table 1: Combinators (top) with descriptions (mid) and examples (bottom)

Note how each function in Table 1 takes in another function as an argument. In the context of ALife, this other function could be a step function. Uiua's ability to perform vectorized operations on entire arrays at once can significantly reduce computational overhead, while allowing seamless abstraction over low-level details. For example, the `Step` for Conway's game of life can be written as `Step ← ↥∩'=3⊙-,:/+⍚ 1_∞ ↻/⊂-1⇡ 3_3.` (by the user Garmelon on the very welcoming Uiua discord server).

[1] https://www.uiua.org/

Method

The following example demonstrates a simple particle swarm simulation in Uiua. Frist we define `Init` that, awaiting a seed, creates a 10 by 2 array (representing position) of numbers uniformly distributed between `0` and the map size `W`. Next we define `Step` to take in an array and add random noise to each element, clipping all values to be within the bounds of the map. Finally `Draw` turns an array of positions into a location matrix (which can be used with Uiua's inbuilt `&gifs` or `&ims` to generate gifs or images).

```
W    ← 100                             # size constant
Init ← |1 + ÷ 2 W gen N_2              # inits
Step ← |2 ↥ 0 ↧ - 1 W + -0.5 ⊙gen ⟜△ : # step
Draw ← |1 ⍜(⊡|-1) ⌊ : + 1 × 0 °△ W_W   # shows
```

Table 2: `Init`, `Step` and `Draw` functions for a basic particle swarm simulation

The vertical bar and number (`|1`) after the function assignment operators describes the number of arguments each function takes. Increasing the complexity of our simulations then amounts to merely elaborating the `Init` and `Step` functions.

Results

Combining the code in Table 4 we can run a full simulation (or arbitrarily many parallel simulations).

```
≡Draw ∧(.Step) ⇡ 1000 Init 0                             # Run and then draw sim
Save ← &fwa : img "png"                                  # save function
≡Save : {"1" "2" "3" "4" "5"} ⊏ ⊚ = 0 ◿ 200 ⇡ 1000   # actually save
```

Table 4: Running and saving (see Figure 1) a 1000 step simulation (with seed 0)

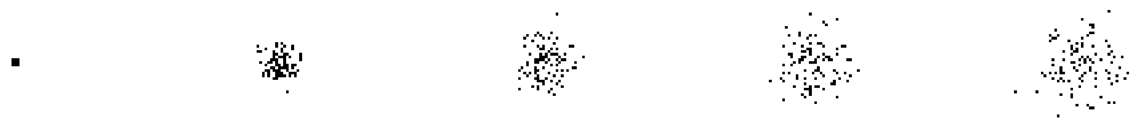

Figure 1: Every 200th step throughout the simulation

For more complex example, consider evolutionary neural network implementation [1] made by the creator of Uiua, which demonstrates Uiua's potential for implementing advanced ALife techniques.

Conclusion

The particle swarm simulation demonstrates Uiua's capabilities for ALife research. By leveraging Uiua's array operations and functional programming paradigms, researchers can create efficient and expressive simulations of bio-like systems. The concise syntax and stack-based approach enable rapid prototyping and iteration, facilitating hypothesis testing and model refinement. By adopting Uiua, ALife researchers can streamline their workflows, reduce boilerplate code, and focus more on the scientific challenges at hand.

References

[1] Kai Schmidt, “Evonet: Basic Evolutionary Neural Network in Uiua.” 2025.