A Modern Array Language for Artificial Life Research

Noah Syrkis

October 6, 2025

1 | Motivation

2 | Glyphs

3 | The Stack

4 | Example: RHOS

5 | Example: Sortsol

1 | Motivation

For legacy reasons ALife research overwhelmingly largely use Python. This talk presents an alternative: Uiua, a stack based array language written in Rust inspired by APL. Working with non-coders (architects and artists), I've found that they find Uiua intuitive. Further, Uiua's array orientedness and visual compactness makes it a winner (for me), and it's already used in neuroevolutiuon [1]

```
lambda x, y, z: x + y + z
```

Python

```
+ +
```

Uiua

Listing 1: Two implementations of anonymous function that adds three numbers

2 | Glyphs

Uiua uses special symbols (glyphs). Two frequent initial reactions to these are intrigue (can you feel it?) and incredulity[1]. If you play with Uiua both quickly subside. A few common glyphs are:

duplicate flip power random range shape floor abs map

Listing 2: A few common glyphs

[1] I just got my drivers license and learning the signs of the road was harder than learning Uiua

2 | Glyphs

Some glyphs are more high-level. To visualize points, you could use ⍜ (under), which transforms, applies a function, and undoes transform, on an $n \times n$ array of zeros, by selecting the index of the particles in the array (transform), adding `1` (function), and putting the values back in the array.

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

Listing 3: Glyphs with descriptions and examples

3 | The Stack

```
1 2 3   # stack: 1 2 3
```

- Functions pop and push values on the stack
- Functions apply seamlessly across array dims

3 | The Stack

```
1 2 3   # stack: 1 2 3
+ +     # stack: 6
```

- Functions pop and push values on the stack
- Functions apply seamlessly across array dims

3 | The Stack

- Functions pop and push values on the stack
- Functions apply seamlessly across array dims

```
1 2 3   # stack: 1 2 3
+ +     # stack: 6
⇡       # stack: [0 1 2 3 4 5]
```

3 | The Stack

- Functions pop and push values on the stack
- Functions apply seamlessly across array dims

```
1 2 3   # stack: 1 2 3
+ +     # stack: 6
⇡       # stack: [0 1 2 3 4 5]
+ 1     # stack: [1 2 3 4 5 6]
```

3 | The Stack

- Functions pop and push values on the stack
- Functions apply seamlessly across array dims

```
1 2 3   # stack: 1 2 3
+ +     # stack: 6
⇡       # stack: [0 1 2 3 4 5]
+ 1     # stack: [1 2 3 4 5 6]
\+      # stack: [1 3 6 10 15 21]
```

3 | The Stack

- Functions pop and push values on the stack
- Functions apply seamlessly across array dims

```
1 2 3   # stack: 1 2 3
+ +     # stack: 6
↕       # stack: [0 1 2 3 4 5]
+ 1     # stack: [1 2 3 4 5 6]
\+      # stack: [1 3 6 10 15 21]
/+      # stack: 56
```

4 | Example: RHOS

```
R ← ⁿ:e ¯÷ ×2 ⁿ2 : √ /+ ⁿ2 ⌵
G ← ≡₁R ⌵ - ⌊ ÷ 2 ⊸ ⋕ ↟ ⊟ .
D ← ÷ ⊸Ö/↥ 2 √ ⌵ - ⊸ ↻ 0_1
```

Figure 1: Two works from RHOS

5 | Example: Sortsol

```
Init ← + ÷ 2 W × S - 0.5 gen N_2
Step ← + × S -0.5 gen N_2
Draw ←  ○(⊡|+1) ↥ 0 ↧ -1 W ⌊ : × 0 °△ W_W
```

Figure 2: 200 steps of a simple particle simulation with particles moving randomly

References

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

A | Saving

```
≡Draw ∧(.Step) ↟ T Init R    # run sim
&fwa "out.gif" gif 48 - : 1  # save gif
```

Listing 4: How Sortsol was run (top) and saved to gif (bottom)