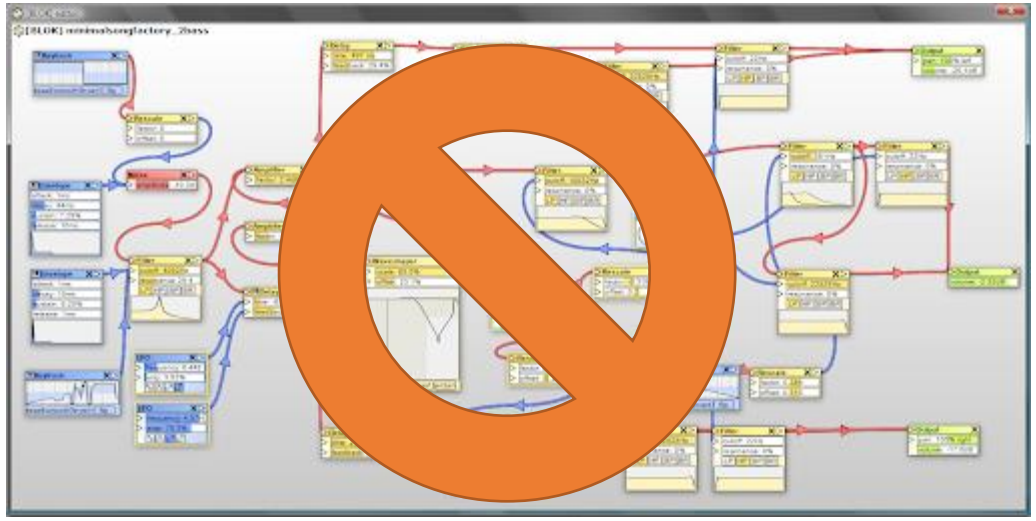
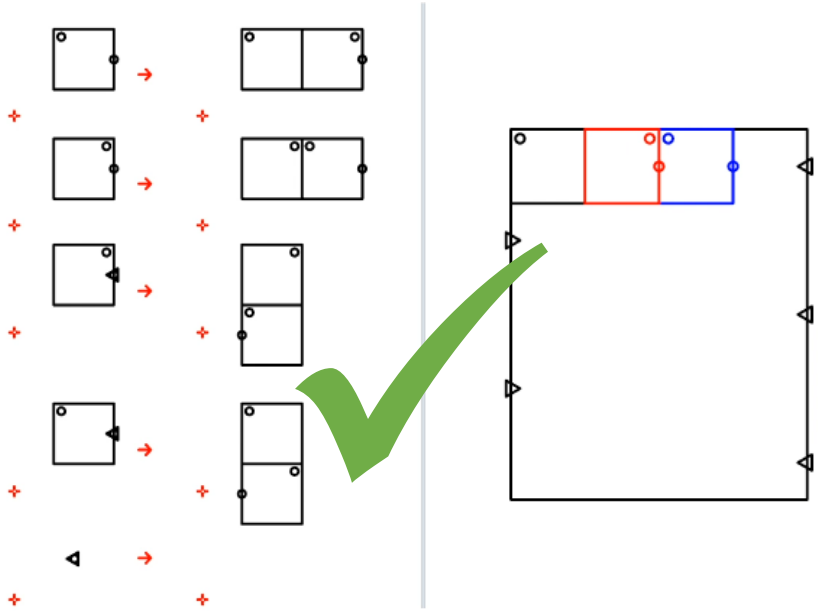
ShapeScript: Toward shape-based visual programming

Kelsey Kurzeja

Goal = Visual programming

- Create a visual programming language that does not use a graph-based dataflow paradigm.





Elements of programming languages

Element	Standard	ShapeScript
Data	Numbers	Shapes
Arithmetic	Add, multiply, etc.	Find & replace
Instruction specification	Function	Construction
Conditionals	If Boolean is true	If shape is found
Repetition	Loops, recursive functions	Recursive construction

What is a ShapeScript program?

- **Construction**: a ShapeScript procedure
  - A sequence of instructions
- **Instruction:**
  - CreateShape, CopyShape, Construct, *Evaluate*, Find, Replace, PickRandom, PickConstruction
  - Creates a new object (usually)
- **Object**:
  - Shape, Subshape, Construction, or List

Construction1(parameters)
object1 = Instruction1(…)
object2 = Instruction2(…)
object3 = Instruction3(…)
object4 = Instruction4(…)
…

Construction2(parameters)
object1 = Instruction1(…)
object2 = Instruction2(…)
object3 = Instruction3(…)
object4 = Instruction4(…)
…

Instruction specifications

- CreateShape() -> Shape
- CopyShape(reference) -> Shape
- Construct(blueprint, parameters…) -> Construction
- Evaluate(construction) -> None
- Find(supershape, search shape, transform type) -> List<Subshape>
- Replace(subshape list, replacement shape) -> Shape
- PickRandom(shape list) -> Shape
- PickConstruction(supershape, search shape, transform type, construct1, construct2) -> Construct

Create Shape

Inputs: None

Returns: Shape

Description: Creates a shape provided by the programmer. In a future version, we want that if no shape is provided, then the user of the program may provide a shape at runtime.

Copy Shape

Inputs: 1) a shape

Returns: Shape

Description: Creates a copy of the input shape. This may be used to “move” a shape from one construction to another.

Create Construction

Inputs: 1) a blueprint, 2) a mapping from input names to objects

Returns: Construction

Description: Creates an instance of a construction based on the input blueprint. The blueprint has a set of named input parameters and the parameter objects of the created construction are initialized using the input mapping from names to objects. Note, the instructions of a construction are not executed when the construction is created. The evaluate instruction must be used to trigger the execution of a construction. This allows the execution of constructions to be chosen conditionally.

Evaluate

Inputs: 1) a construction

Returns: None

Description: Executes the instructions of the input construction. The execution of the current construction halts until the execution of the input construction is complete.

Find

Inputs: 1) a supershape, 2) a template shape, 3) a transform type

Returns: List of subshapes

Description: Finds and returns a list of subshapes, of the input supershape, that are equivalent to the template shape under the specified transformation type. Currently supported transformations include identity, translation, rigid, isometry, and similarity transformations.

Replace

Inputs: 1) a subshape, 2) a replacement shape

Returns: Shape

Description: Creates a shape that is the subshape's supershape, minus the subshape, unioned with the replacement shape transformed by the subshape's transformation.

Pick Random

Inputs: 1) a shape list

Returns: Shape

Description: Picks and returns a random shape from the input shape list. If the output of a find is to be used in a replace operation, then a single subshape must be picked from the list of found subshapes. Currently, we only support random picking, but if the programmer can be guarantee that only single subshapes are ever found, then this instruction will not cause a program to have non-deterministic results.

Pick Construction

Inputs: 1) a supershape, 2) a template shape, 3) a transform type, 4) a “true” construction, 5) a “false” construction

Returns: Construction

Description: Returns the “true” construction if the input supershape contains the template shape under the specified transformation type. Otherwise, the “false” construction is returned. This instruction is comparable to the conditional ternary operator in many other programming languages. Recursive constructions can be implemented by picking and evaluating a construction of the same type as the construction containing this instruction.

Completed features

- Simple set of instructions that could be implemented visually
- ShapeScript interpreter
- Example programs

- Future features:
  - Graphical user interface for visual scripting
  - Many other possible instructions (replace all, manual find, etc.)
  - More transformations

Example: Looping program

- Points simulate numbers. Find & replace simulates arithmetic.

Main()
i = CreateShape()
c = Construct(Loop, i)
Evaluate(c)

Return(input)
result = input

Loop(i)
zero = CreateShape()
negative_one = CreateShape()
numbers = Find(i, zero, Translation)
number = PickRandom(numbers)
i_minus_one = Replace(number, negative_one)
noop = Construct(Return, i_minus_one)
recurse = Construct(Loop, i_minus_one)
picked = PickConstruction(zero, i_minus_one, Identity, noop, recurse)
Evaluate(picked)
result = CopyShape(picked.result)

Example: Peg solitaire board