Choosing a Paradigm

There are many options for choosing a paradigm for the execution of the program. We need an execution paradigm that allows the user to easily write code to manipulate visual elements. The user should find it easy to write independent agents for the 2D game grid stage, as well as manipulating the turtle graphics stage. For this reason we want the user to be able to define standard imperative code to be run from some sort of `main` function, as well as defining behaviours for agents that can be released to act independently, without manual looping through and updating state.

Independent agents using the JavaScript task queue.

For an Agent object define a `step` (`céim`) function that acts as a transition function taking the agent to its next state in the stage. I.e., moving along defined trajectory, pathfinding, falling due to "gravity", user input movement, etc. When an agent is added to the stage and activated, a task is added to run its `step` function, when the `step` function is complete, another task is added to the queue to run it again, etc. etc. The user can perform whatever set up is required before attaching the agent to the stage. Care must be taken to avoid an infinite loop in the `step` function.

Only one `step` function will be run at a time, blocking for input or sleep if desired.

Each agent is in some way an automaton, with transition functions of arbitrary complexity. The user can program new automata in an OOP manner, then the stages can accept and attach an automaton and execute its step function.

This raises some problems to be considered.

- How do you create new automata at run time from within a step function?
- Is everything an automaton? Including the stages?
- How is pre-step setup defined? Another magic function akin to loop for setup?
- How to decide when to halt execution? Require user input to halt?

Maybe only the stage needs a step function that's called on loop after the setup. The stage can call `step` on all of its attached children. No need for behaviour defined outside the language.

The user could define the following steps

1. Setup variables, functions, agents, custom step behaviours.
2. Attach all starting agents to the stage
3. **Execute** the stage, starting its step loop.

Asynchronous execution

Using the javascript task queue we can implement “simultaneous” execution of multiple agents. The execution will be actually single threaded, only one thing happening at any given moment, but the constructs we use for loops and statements will hide preemptability, meaning we can jump between concurrent tasks.

There needs to be a construct to “launch” the asynchronous tasks. A semantic possibility is an event driven approach. A syntactic approach might be something akin to go’s go keyword. The use of the go keyword would cause an asynchronous execution of the given function. This is effectively an inversion of the javascript `await` keyword, the await keyword is used to specify a synchronous execution, we require the opposite.

Update 31/10/2019

Experimenting with the asynchronous execution paradigm outlined above has been pretty positive. I’ve been successful in writing several demos in this style, including Game Of Life and Langtons Ant. It seems promising, More experimentation will tell whether it is the right approach to use.