

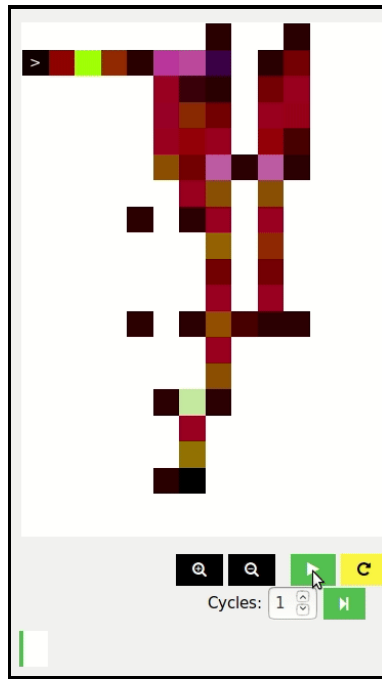
ENTRY // PixelLang Esoteric Language #2 // Showcase

[-] [X]

Date: 2019-09-07 Author: Ian Rash Index: #2

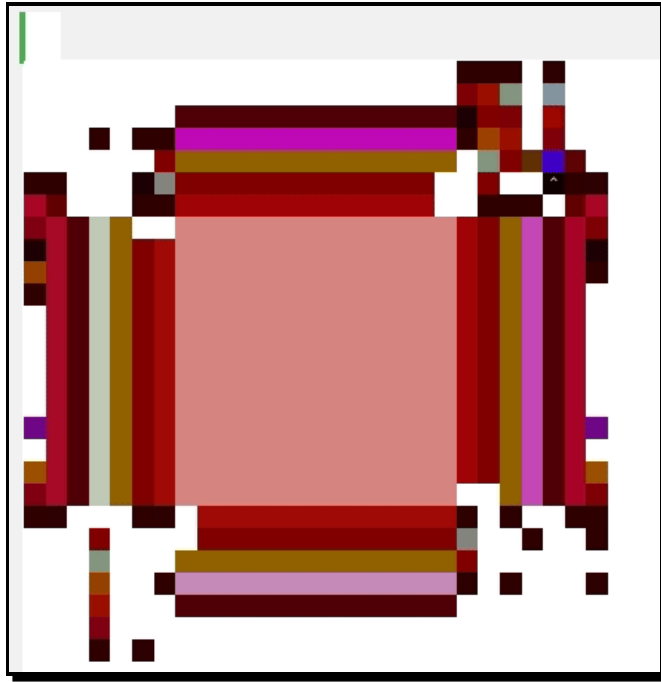
I'd like to take a little time today to showcase some of the programs I made with pixel_lang. I'm very proud of them, and I'm hoping someone will enjoy watching and dissecting them.

The first one I'd like to show off is my Ackermann function, which uses forking, and it's own in-memory call stack to orchestrate all the individual pistons to fire when necessary.



In the example above, there are three possible decisions made, if M is equal to 0, if N is equal to 0, and the recursive call, with the n-arg. That's why the fork splits the piston into three, one for the recursive call itself, and one for the n-arg, which is also a recursive call.

Here is the draft version of the function, that I made by hand with comments.



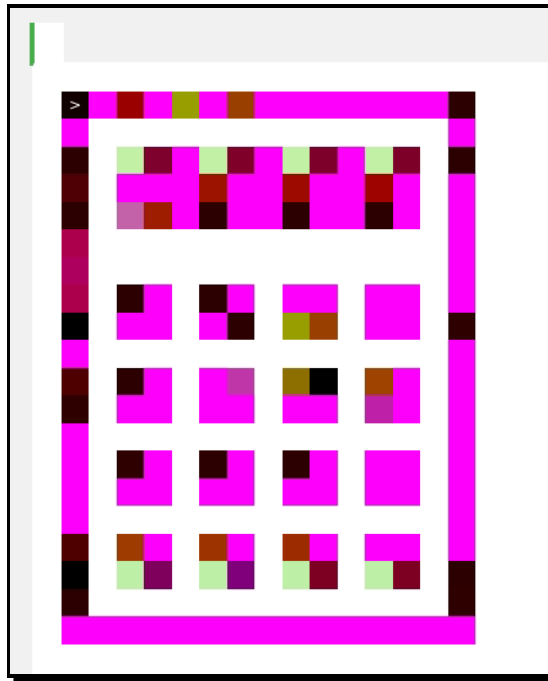
This program uses the meta-programming instruction `IMetaSet`, to color the middle square two different colors each rotation. The internals on it I'm pretty proud of.

First of all, this program has an infinite loop with no possibility of memory leakage. In `pixel_lang`, a piston can potentially crash a program after millions of loops because of things like left-over values in `I`. Since `I` is a collection with no bounds, if it gets to full the program either crawls to a halt, or straight crashes.

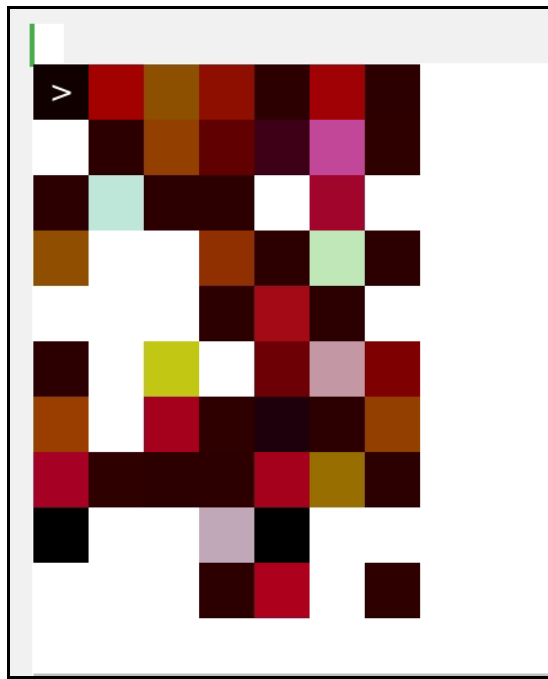
If you look on the left and right sides, you can see two pistons, each going in a loop. The right side is an incrementer, and the left side is a decrementer. They count either up to 19, or down to 7, which is the exact coordinates of our center square, on two separate `SV` values. The inner piston changes it's `S` to read from either `SV` depending on what stage of the program it's at.

The painting bot itself uses a neat effect/tactic. The two colors it's changing itself between are `IMetaSet(:sv, 0, :mav, 0, 2, :ma, 0)`, `IMetaSet(:mav, 0, :sv, 0, 2, :ma, 0)`. `SV` and `MAV` are either `X` or `Y`, depending on the direction. `MA` is the control code register, which is always `0xD` (for `IMeta`). On `I` we stack the color value, which in this case is either, `0x68480` or `0x49480`. Whatever color is currently on the board, it paints the opposite of it.

The timing was really difficult on this program, because fork priority had to be respected at every turn. Since each piston has `Read`, `Execute`, and `Move` phases, the two pistons on the sides only increment/decrement after the painting piston has moved. This gives the effect of painting whats behind it, even though it actually changes the instruction it was on during execute phase, and then moves forward one.



Next is my calculator program, this one takes a non-parenthesized math expression, the runs the operations from left to right (does not respect OoO). This was just a fun artsy project I wanted to do to show art was possible (If you can call it art).



Another program I wrote is the Sieve of Eratosthenes, a fun way to filter prime numbers. This program uses Call/Return to move the pistons into an "incremental tar pit", the higher the number of the current index, the higher the wait for a piston to be released from the tar pit. Basically, a generator forks pistons with the current index (which it increments every cycle), then checks to see if it went over the limit. The forked pistons go through the program. When a prime number is found, it is added to a list of prime numbers which is checked against all other future potential primes. This list is output at the end of the program, once the last piston is done. This example of the program runs up to 30.

TAGGED SKILLS:

 Crystal

 Esoteric Programming

 Generative Art

 Raylib