Programming soft alife with SPLAT and ulam

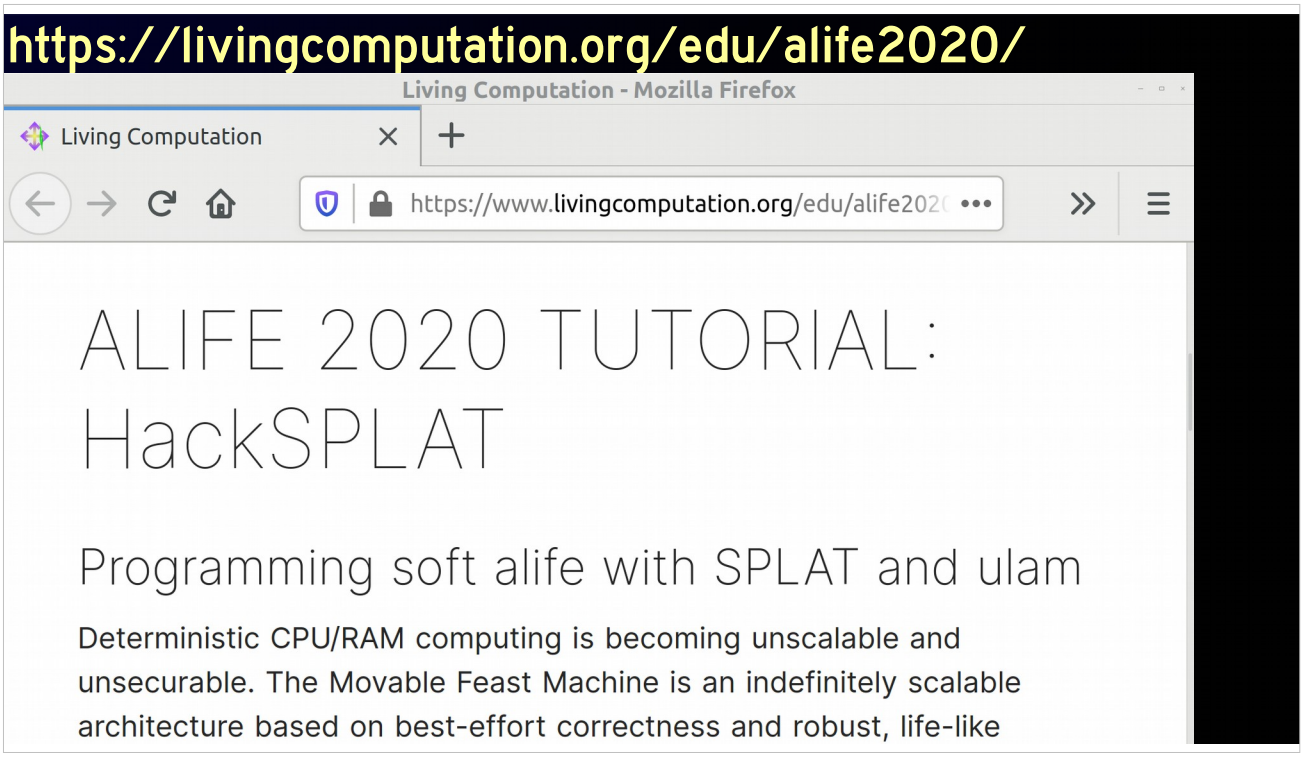
#tutorial-HackSPLAT
16:00-17:30 EDT
Monday, July 13, 2020
Virtual ALIFE2020
https://livingcomputation.org/edu/alife2020/

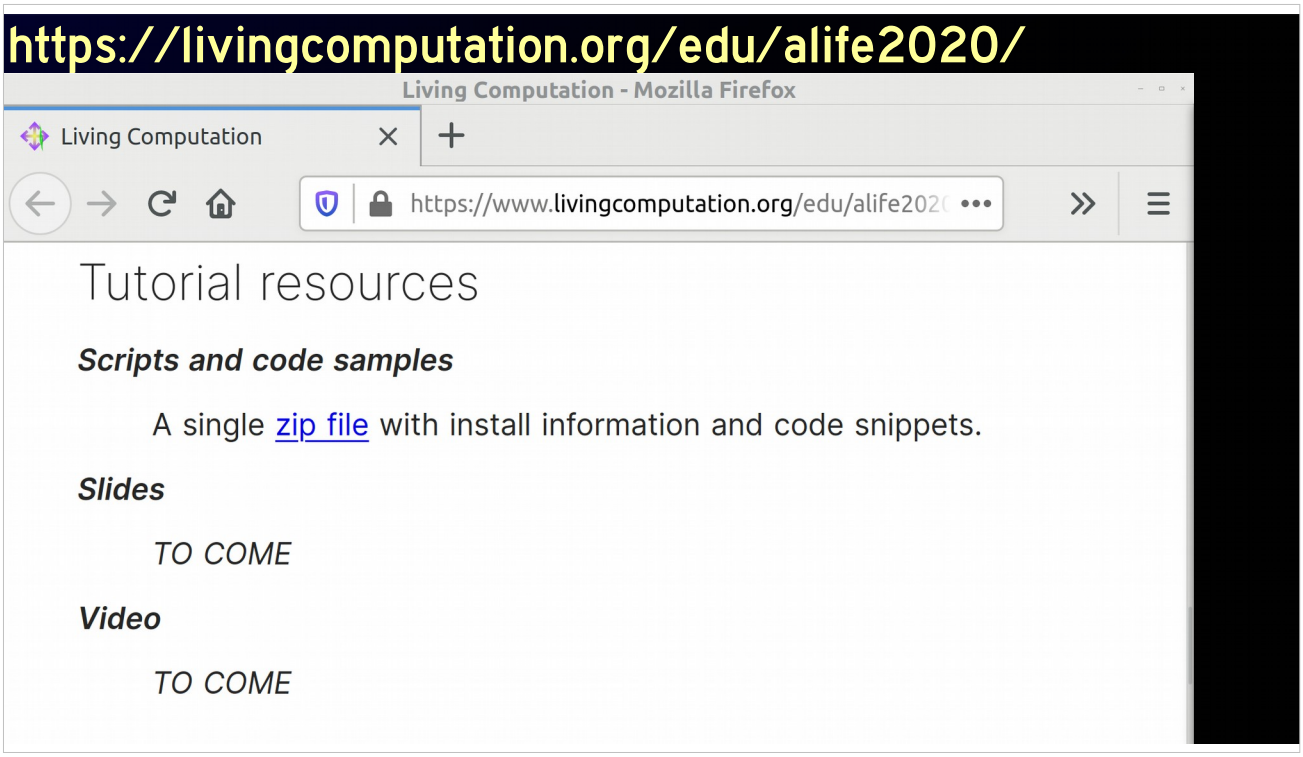
Dave Ackley
Living Computation Foundation

THANKS FOR COMING

Let's get started

Note the URL


https://livingcomputation.org/edu/alife2020/
Living Computation - Mozilla Firefox
Living Computation
https://www.livingcomputation.org/edu/alife202
ALIFE 2020 TUTORIAL:
HackSPLAT
Programming soft alife with SPLAT and ulam
Deterministic CPU/RAM computing is becoming unscalable and unsecurable. The Movable Feast Machine is an indefinitely scalable architecture based on best-effort correctness and robust, life-like



– code samples are available there now, and slides and video will be soon

OUTLINE

0) Organization: What are we doing here?
1) Best-effort computing: **MFM** & **ulam**
2) Space is the place: SPLAT
3) Biology and Beyond

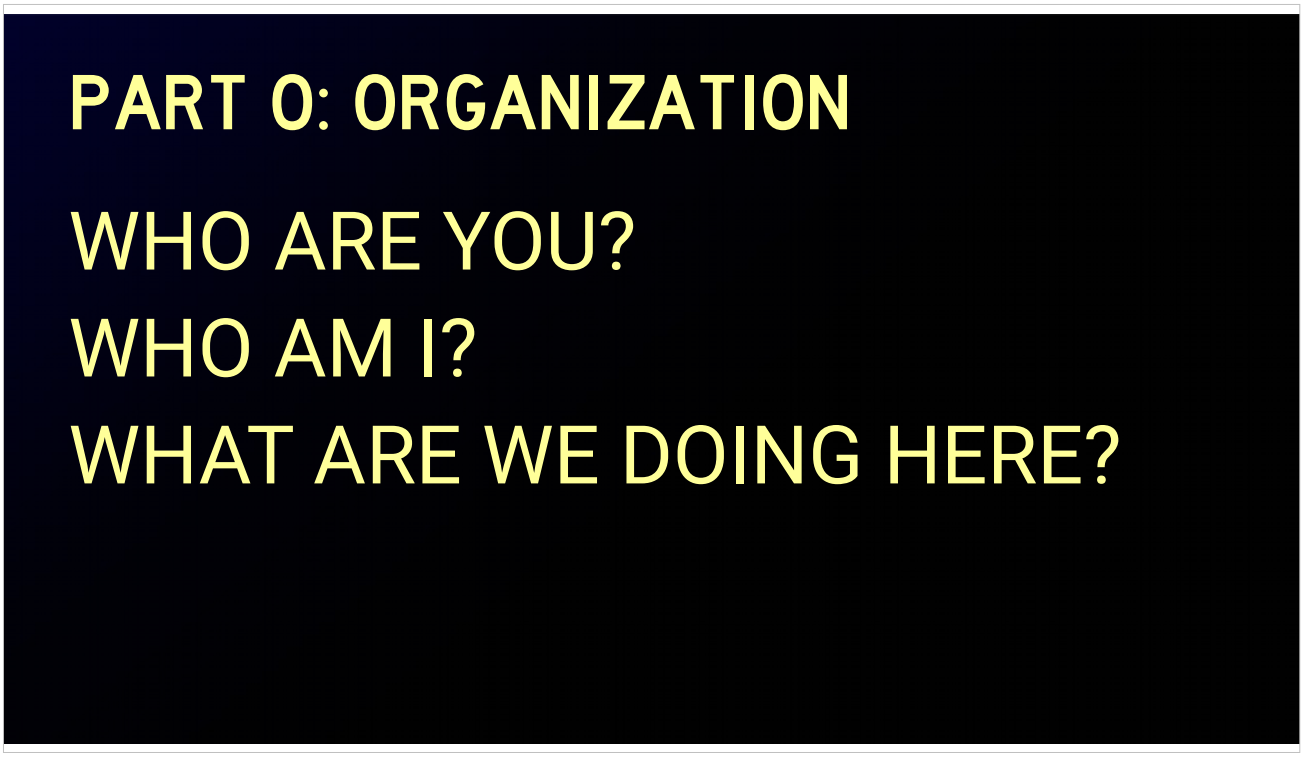
OUTLINE

0) Organization: What are we doing here?
1) Best-effort computing: **MFM** & **ulam**
2) Space is the place: SPLAT
3) Biology and Beyond

BIG PICTURE MOTIVATIONS + CONCRETE CODE EXAMPLES

OUTLINE

0) Organization: What are we doing here?
1) Best-effort computing: **MFM** & **ulam**
2) Space is the place: SPLAT
3) Biology and Beyond



No seriously, who ARE you?

You: Programmer of Deterministic Machines, Restless (at an Alife conference! Or you found this online in the future – what made you click?)

Me: Recovering Deterministic Machine Programmer, restless

This: (1) A conceptual reboot with (2) Lots of little code examples

PART 0: ORGANIZATION

WHO ARE YOU?
WHO AM I?
WHAT ARE WE DOING HERE?

CELLULAR AUTOMATA

WE'RE TALKING CELLULAR AUTOMATA OF A VERY DIFFERENT KIND – ASYNCHRONOUS, NON DETERMINISTIC, FAILURE-PRONE

PART 0: ORGANIZATION

WHO ARE YOU?

WHO AM I?

WHAT ARE WE DOING HERE?

CHAOTIC GOOD

CELLULAR AUTOMATA

PART 0: THE GOAL FOR SOCIETY

Develop and deploy **useful devices** based on indefinitely scalable, best-effort computing

VS RECREATIONAL MATHEMATICS

VS SCIENCE / PHILOSOPHICAL

Focus on robustness in deployable SYSTEMS, rather than elegance in asymptopia

WHERE IS THE ALIFE HERE?

INDEFINITE SCALABILITY IMPLIES ALIFE

OUTLINE

0) Organization: What are we doing here?
1) **Best-effort computing: MFM & ulam**
2) Space is the place: SPLAT
3) Biology and Beyond

PART 1: MFM & ULAM

MFM: Movable Feast Machine computer architecture:

- Indefinitely-scalable (no internal size limits at all)
- Best-effort (no guaranteed hardware determinism)

Ulam: An object-oriented procedural programming language for the MFM

- Very limited persistent state (just the CA grid itself)
- Stack and best-effort determinism during one event
- No pointers or general purpose RAM

PART 1: MFM & ULAM

HOW ABOUT AN EXAMPLE

Let’s try to do Hello World

EG10 Hello World

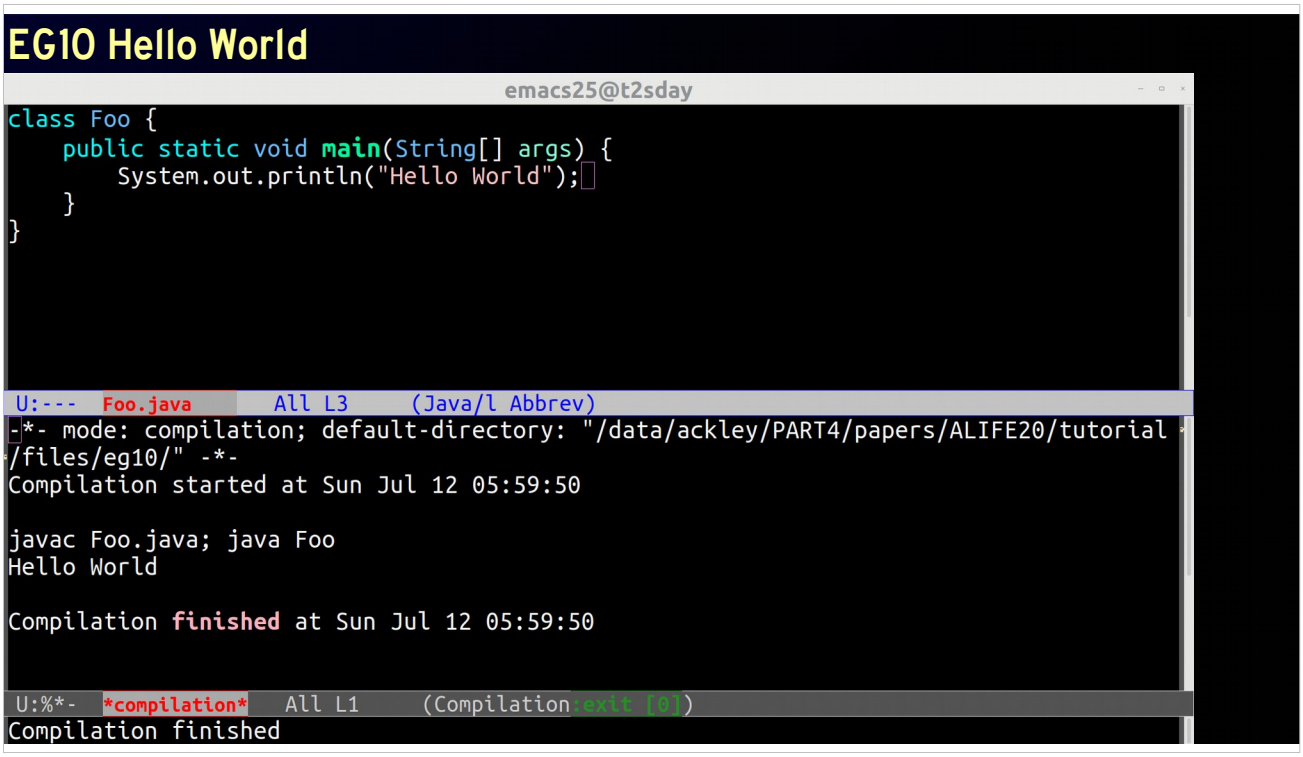
Here’s a java version

EG10 Hello World

Compile the program..

Run the program..

EG10 Hello World



What could be simpler?

But from our point of view, almost EVERYTHING about that example is wrong or even evil.

Let's try to write Hello World in ulam, the first of our two languages for today.

So here's an ulam take on Hello World

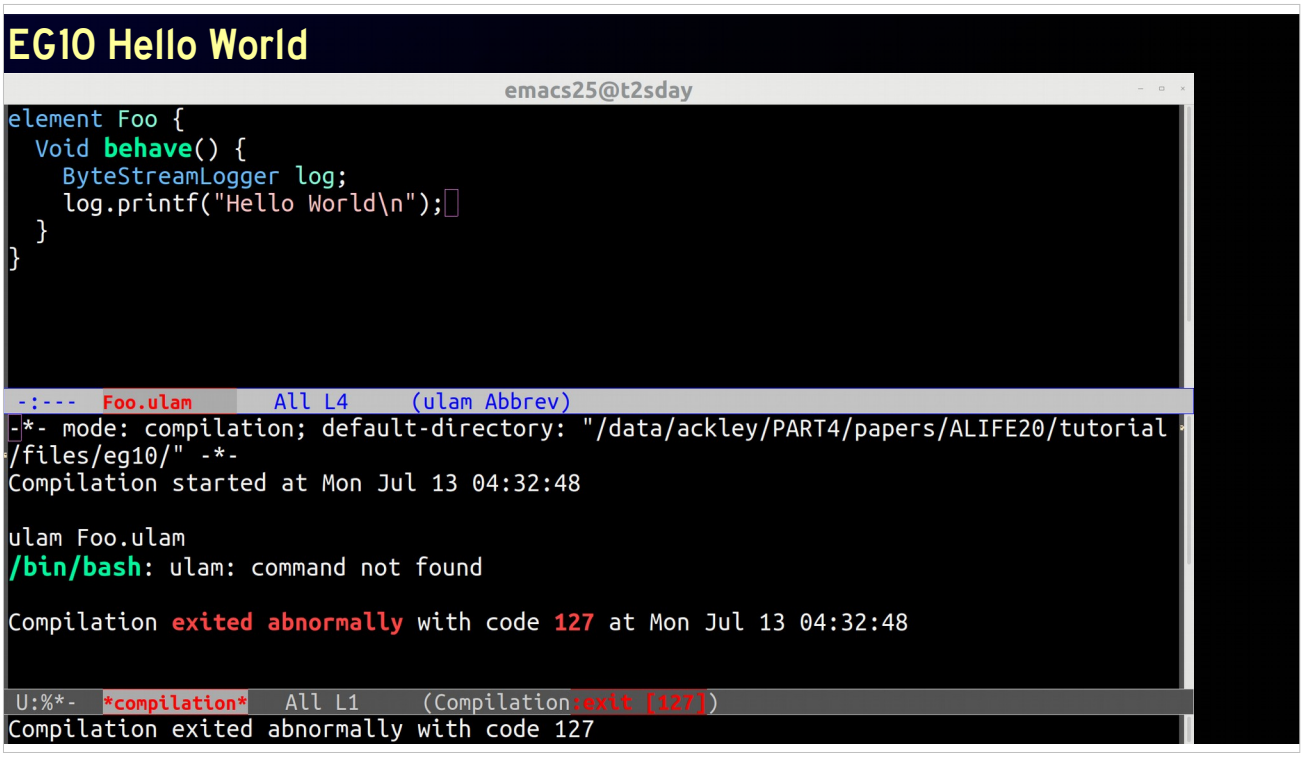
Keywords are different – 'element' vs 'class' – and 'Void' is capitalized for some reason, but hey, we're restless deterministic programmers – this is not that terrifying.

We notice there isn't any 'main'.. But it looks like 'behave' is playing that role.

EG10 Hello World

The ulam compiler is called ‘ulam’..

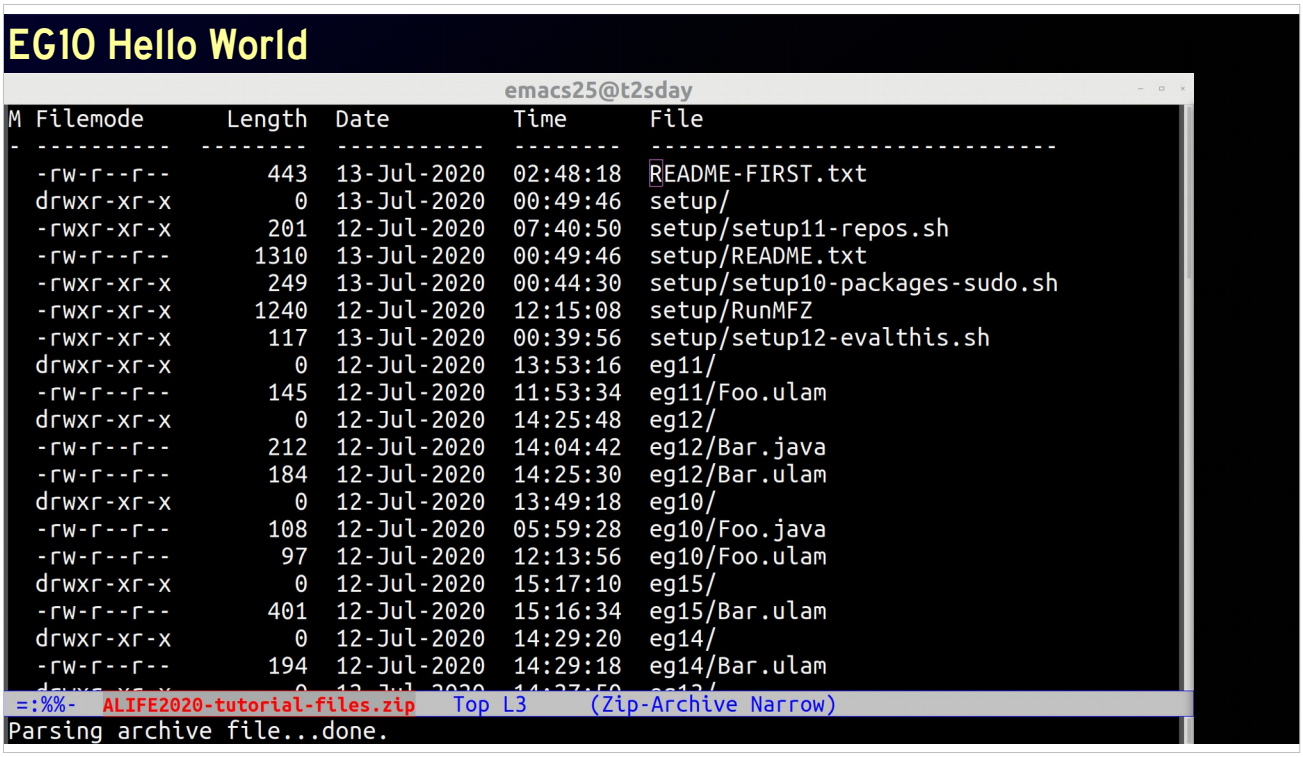
EG10 Hello World



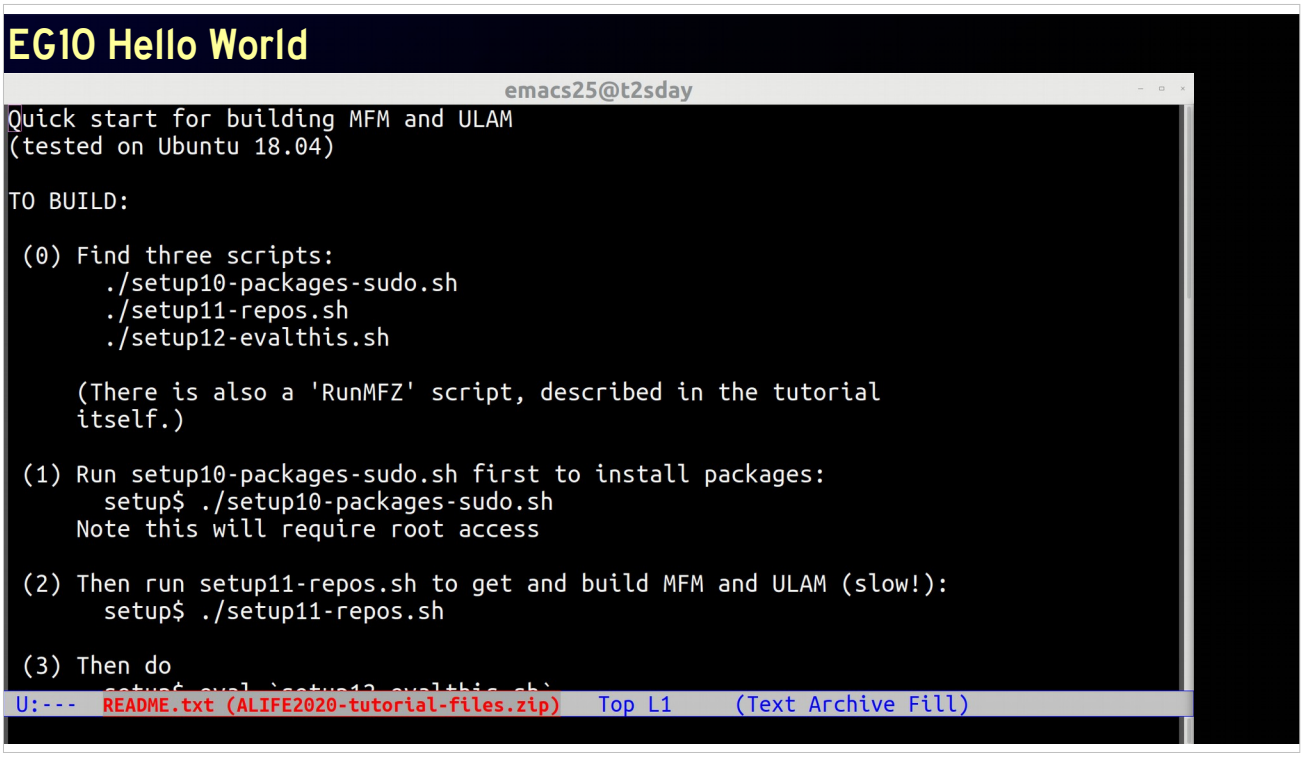
But our first problem is it’s way not standard.

But there's that zip file on the tutorial website ..

EG10 Hello World



with READMEs and scripts to help set things up on a linux box –



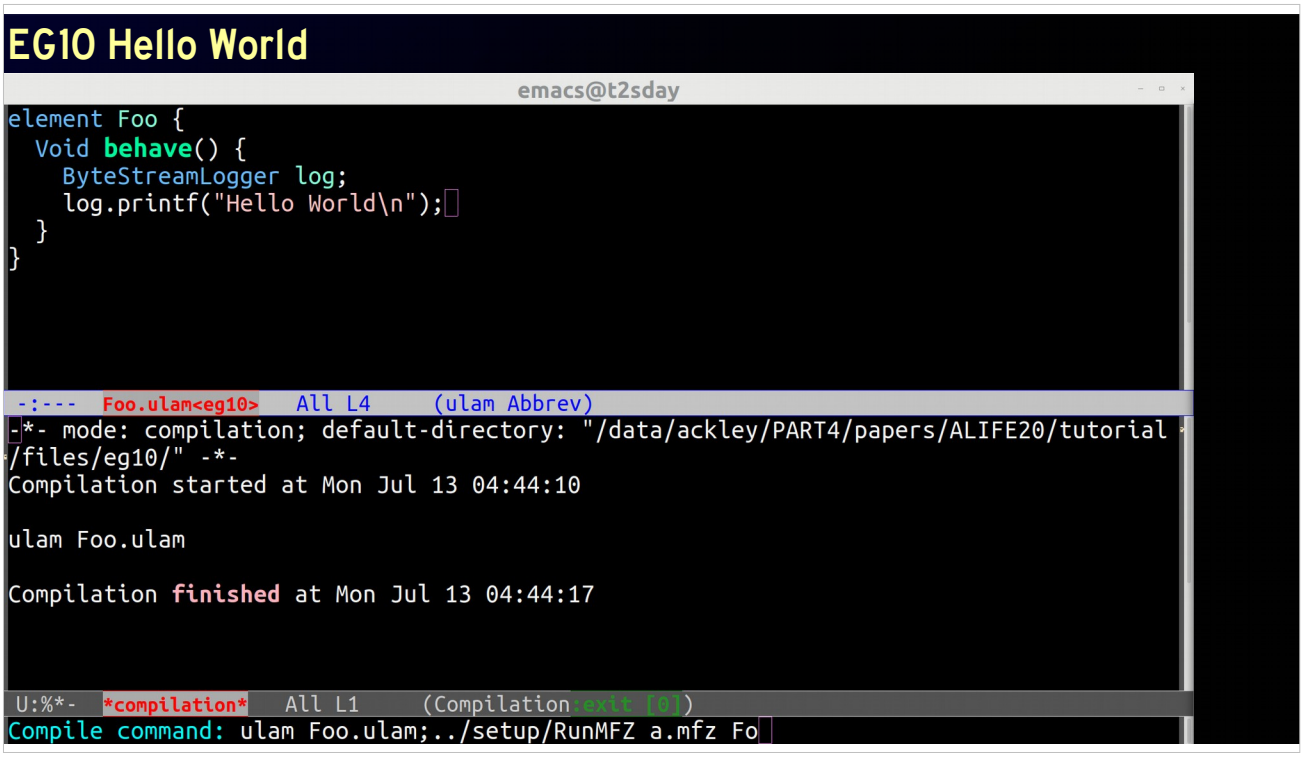
At least on Ubuntu 18.04. But it hasn't got anything that weird in it. It's just packages and code

We can handle this

And it compiles.

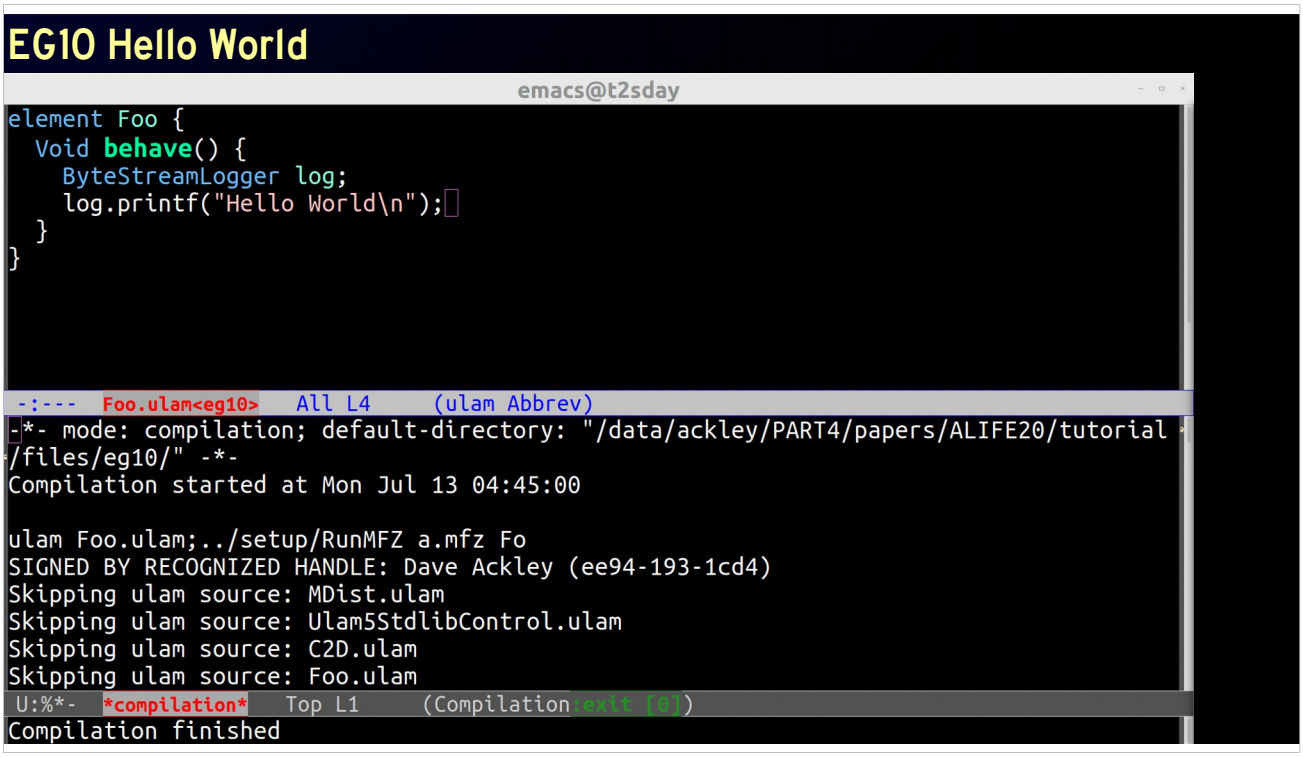
So now what?

By default, ulam makes an ‘a.mfz’ file,



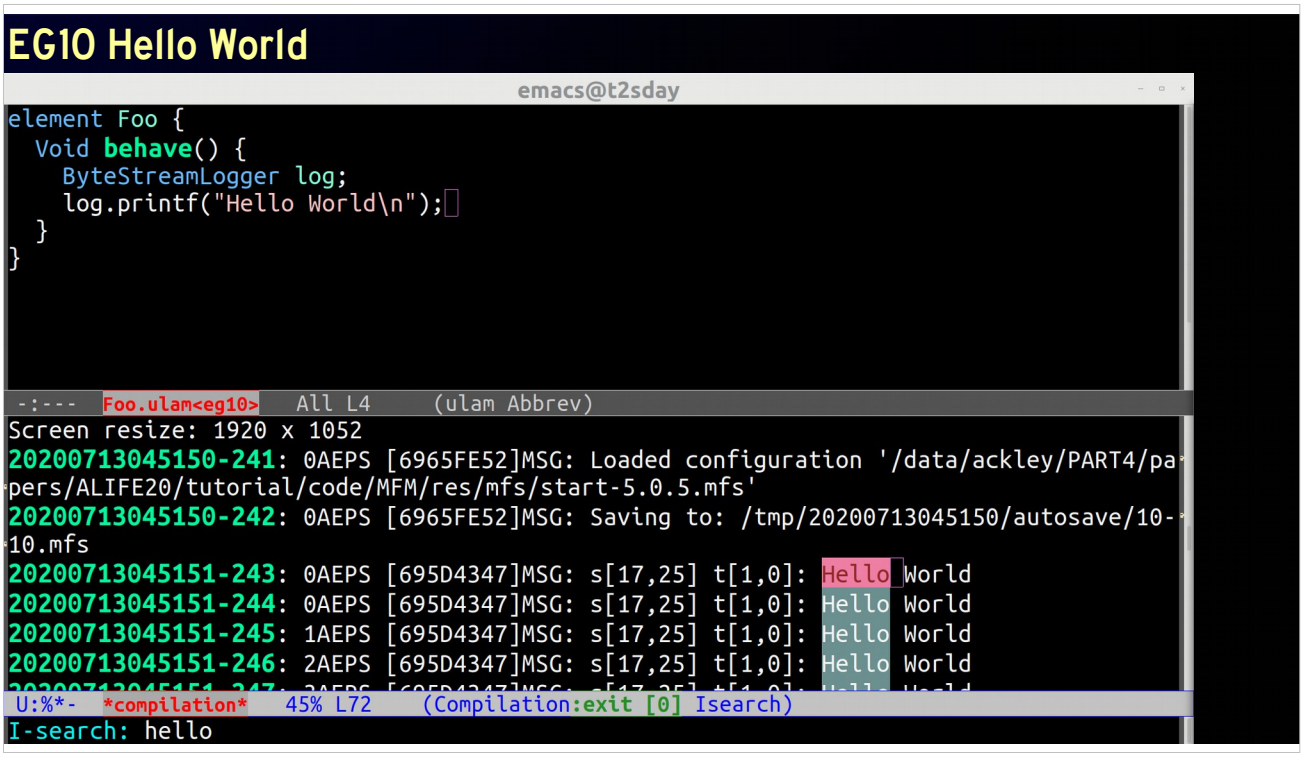
and the zip file comes with this weird RunMFZ script we can try:

EG10 Hello World



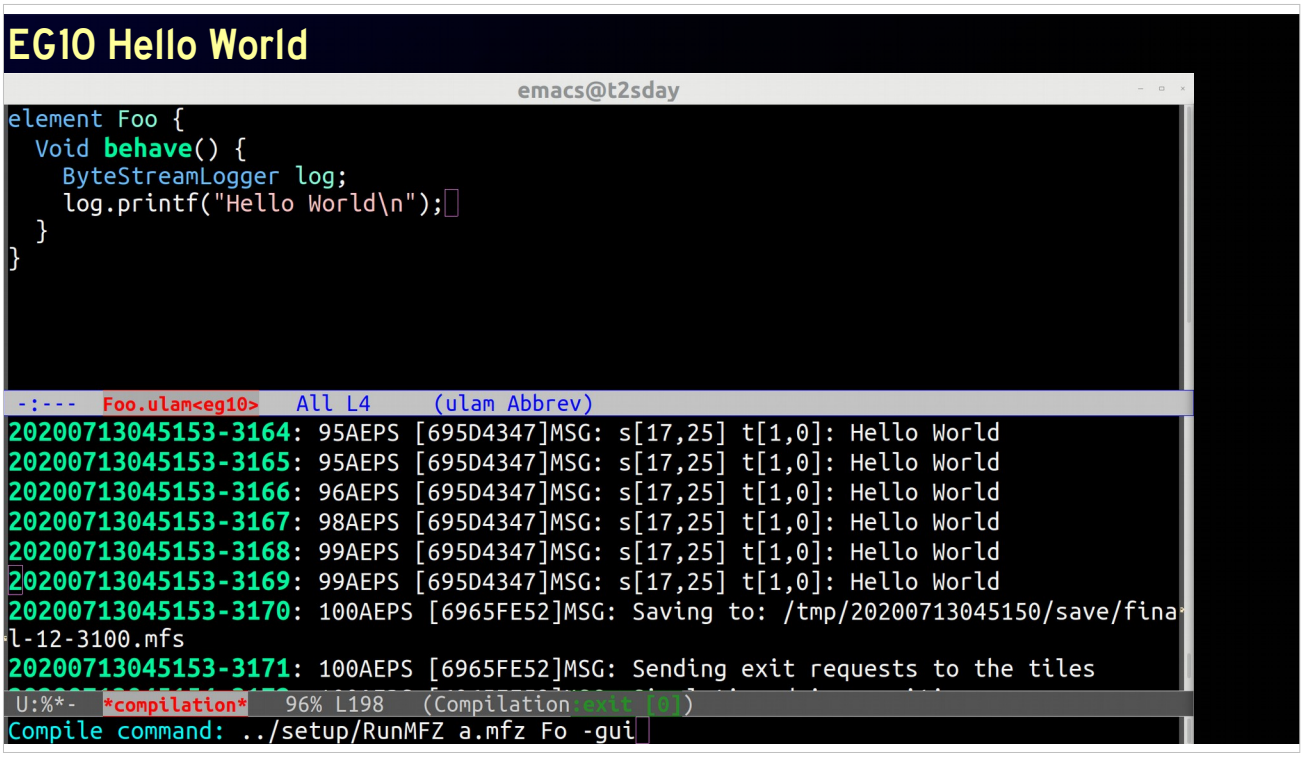
Which generates a TON of output.

Where the heck is our ‘Hello World’?



There it is.

Actually there it is over and over..



If we stick ‘-gui’ at the end of the RunMFZ command it’ll bring up the mfms simulator interface (which is really the main way mfms is used.)

This whole RunMFZ thing is kind of a hack to try to make ulam code look more the a ‘normal’ program.

But it’s really not. Ulam code is about writing spatio-temporally local state transitions, like ‘any’ cellular automata

And here's the mfms GUI

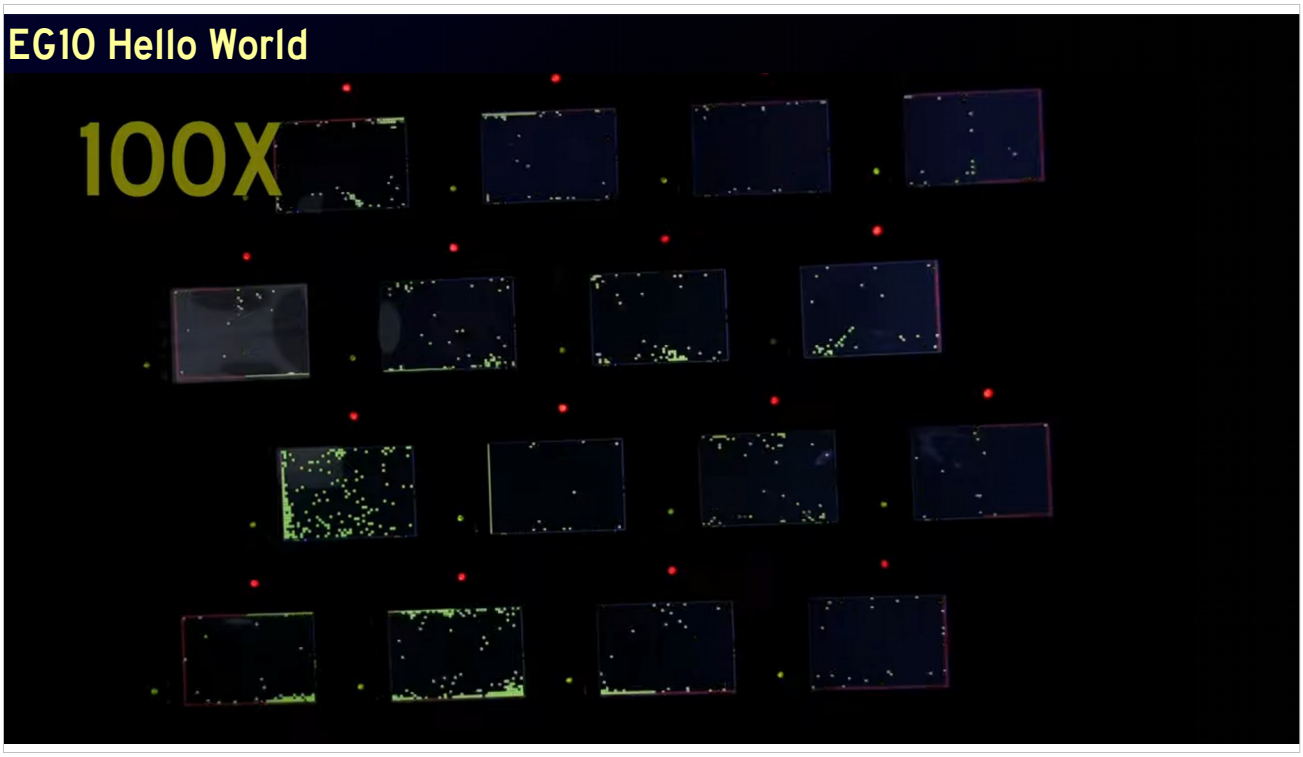
It's… quirky.

The grey rectangles represent TILES, units of computation that can be connected together to form bigger and bigger machines.

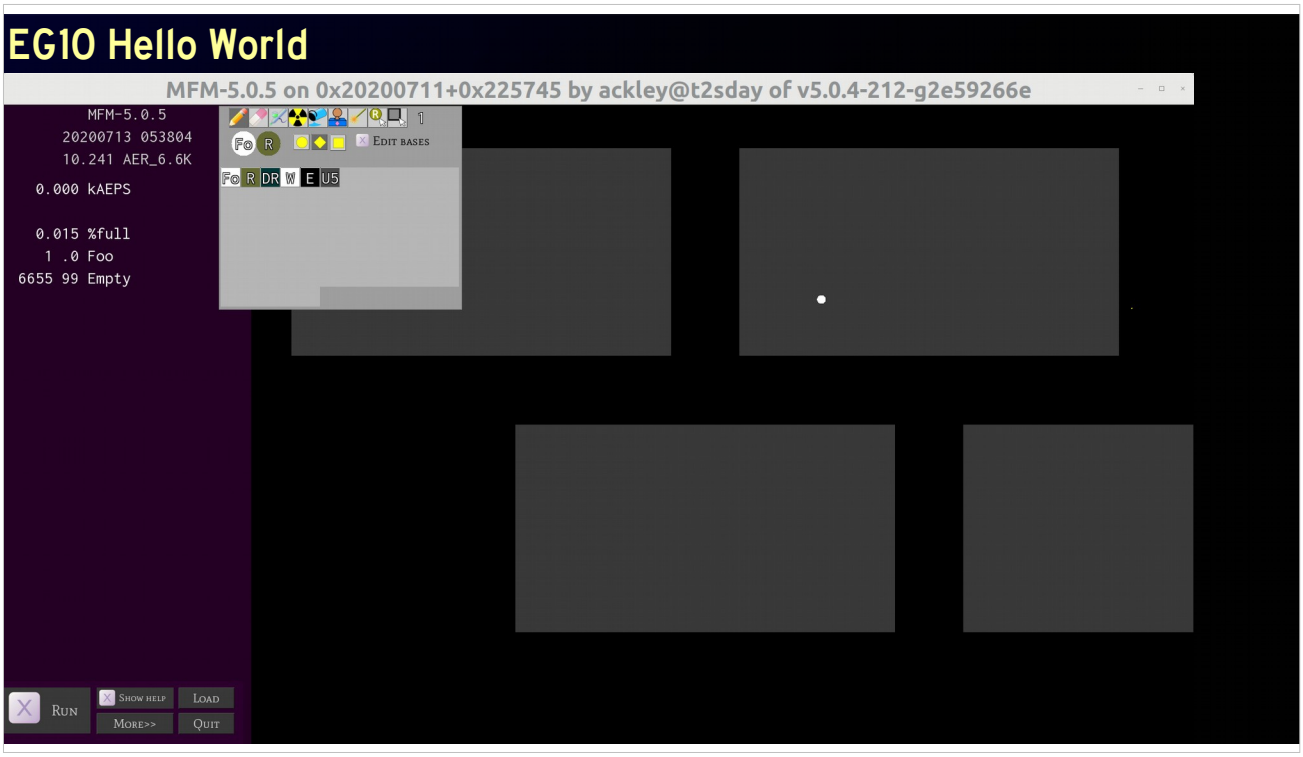
Just for comparison, here's sixteen of the 'T2 Tiles' in real life.

(Officially speaking that red one there isn't special – a grid of T2 tiles has no 'head node' or 'master unit' or anything like that.

But the red one is the one I've been hacking on, which is useful to know.)

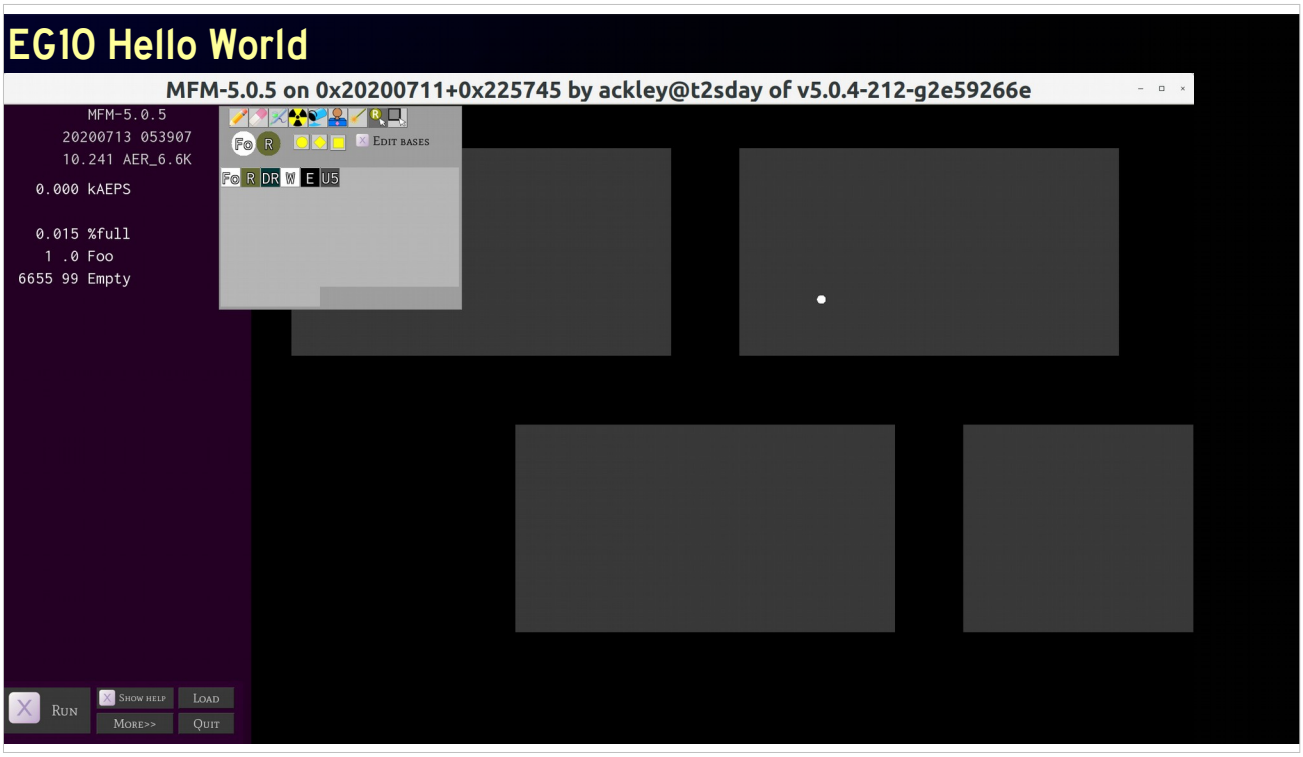


And here's that same grid running a simple program, from a recent time-lapse video segment.

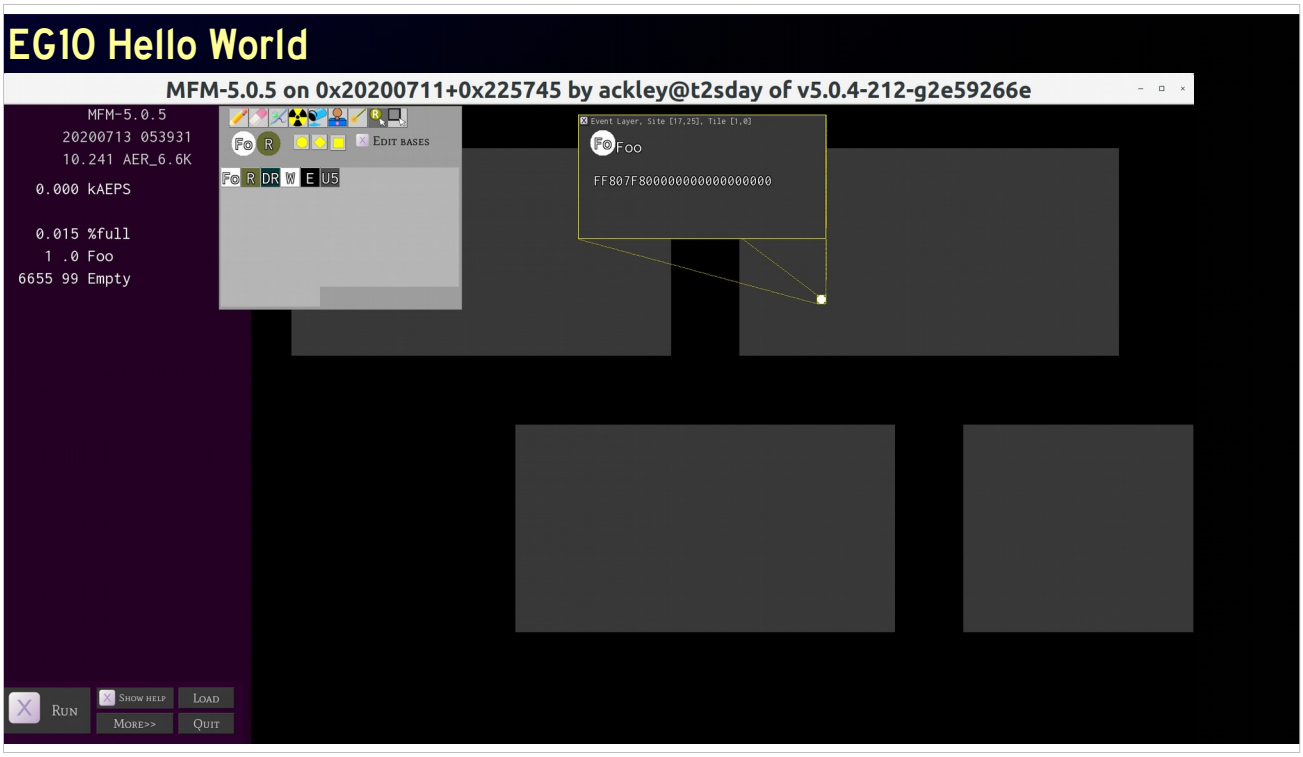


So the four tiles in this simulation are all empty – except for that white dot

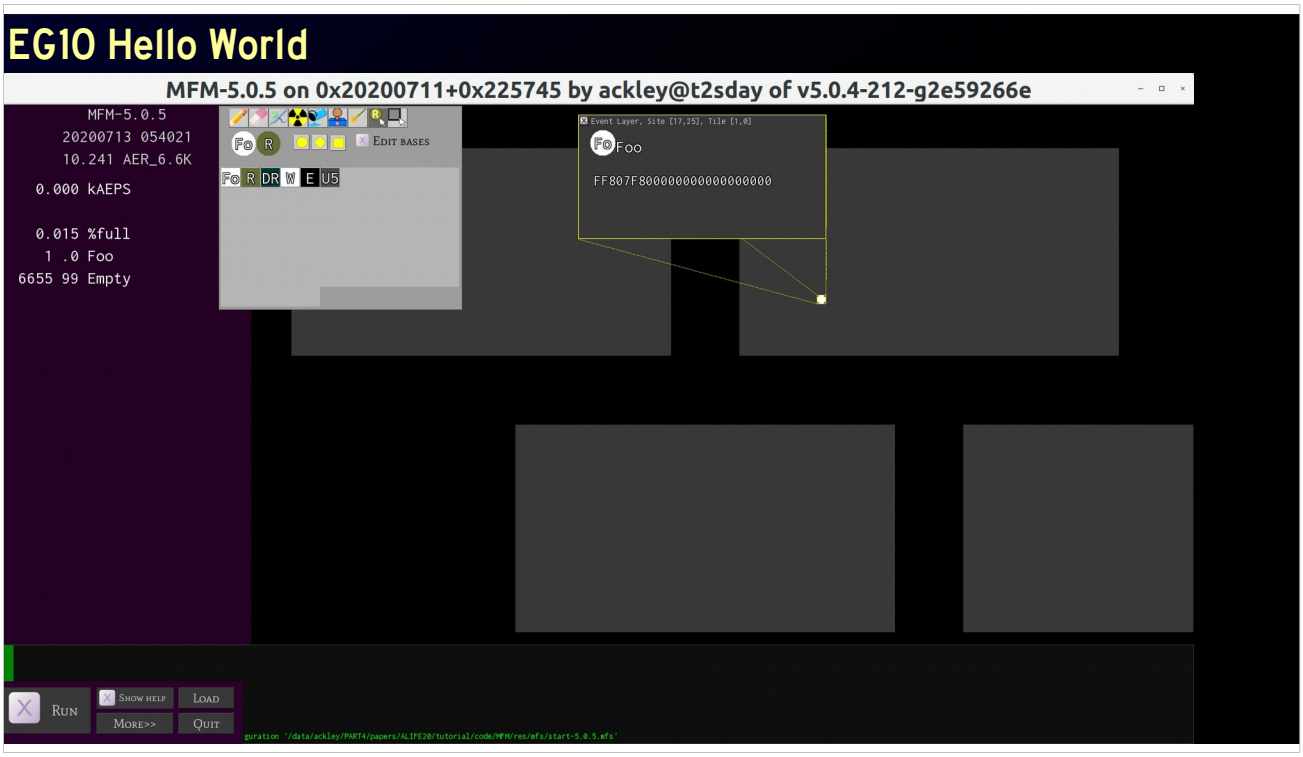
Here we've typed 't' to bring up the Tool palette.



If we click left on the ‘Atom View tool’ (second from the right in the top row), we can then click left on the white dot..)



And see that it is indeed an atom of Foo, the ‘element’ that we wrote in our Foo.ulam class.



It’s hard to see, but here we’ve typed the ‘l’ command, which bring up the logging window – that will show the same output we saw in the compilation window when we did RunMFZ without the ‘-gui’ argument.

And if we click the ‘Run’ button, now we see our Hello World output.

Over and over and over, in the log window.

Let’s fix that.

EG11 HELLO? WORLD

emacs@t2sday

```
//EG11
element Foo {
  Void behave() {
    Logger logger;
    logger.log("Hello World");
    EventWindow ew;
    ew[0] = Empty.instanceof;
  }
}
```

```
-:---  Foo.ulam<eg11>    All L7     (ulam Abbrev)
Quit
```

When a behave() method is called, the atom that it is called upon – called ‘self’ – is located at index 0 of a class called ‘EventWindow’.

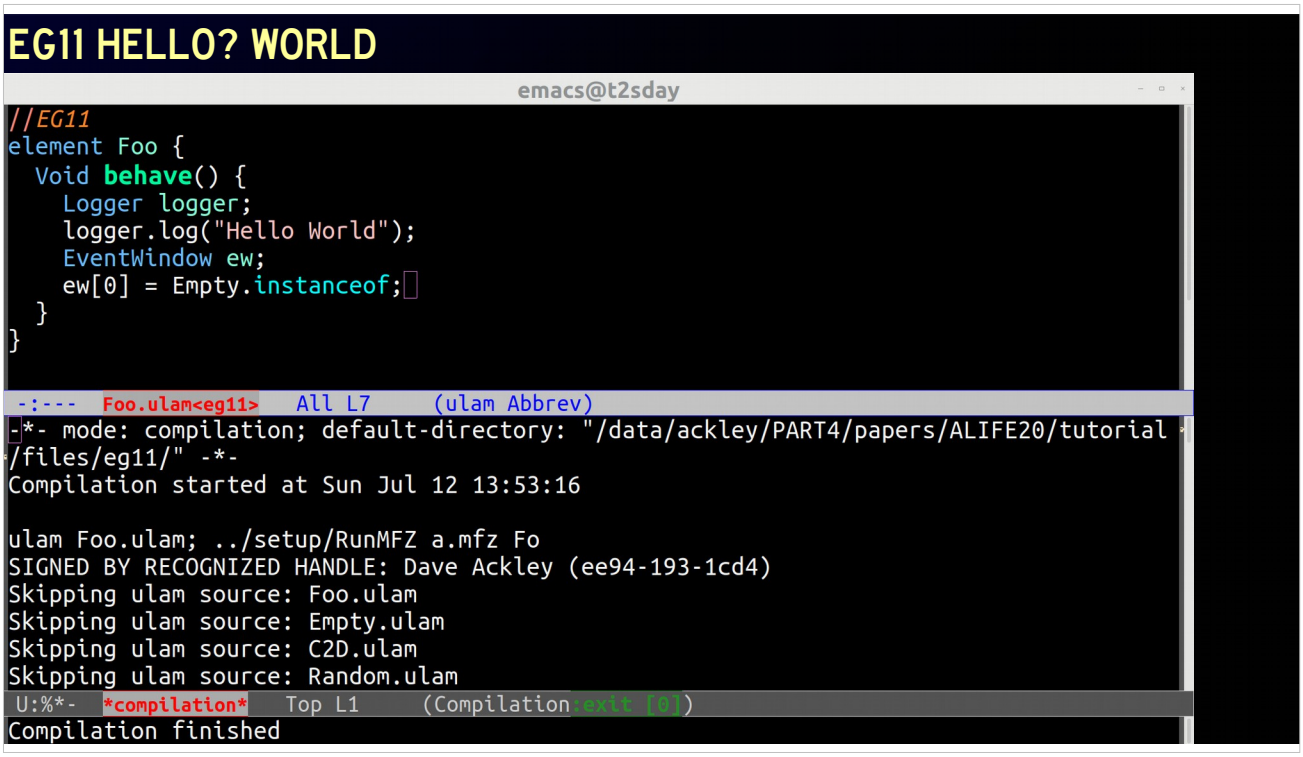
If our behave() method overwrites ew[0], self is modified or completely changed.

Doctor it hurts when I access self after rewriting ew[0].

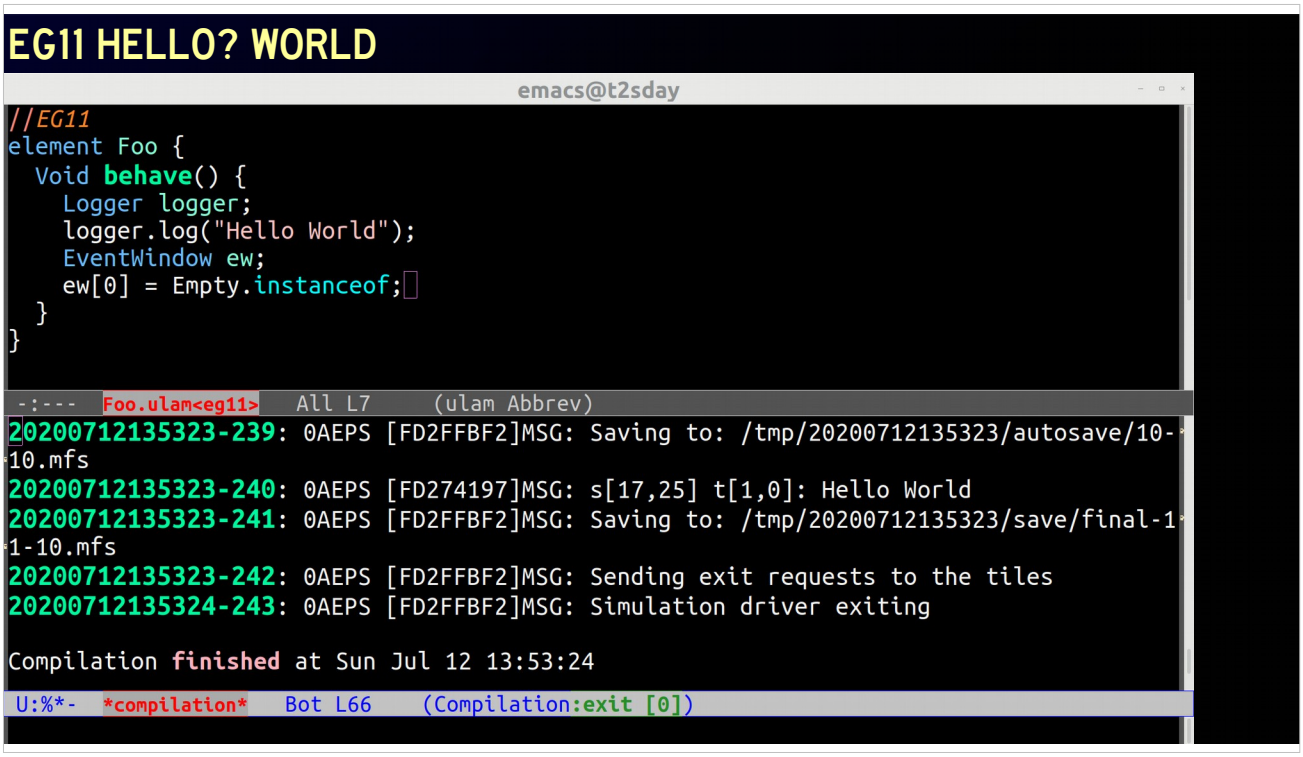
So note we modify ew[0] at the very end of behave()!

Let’s try it.

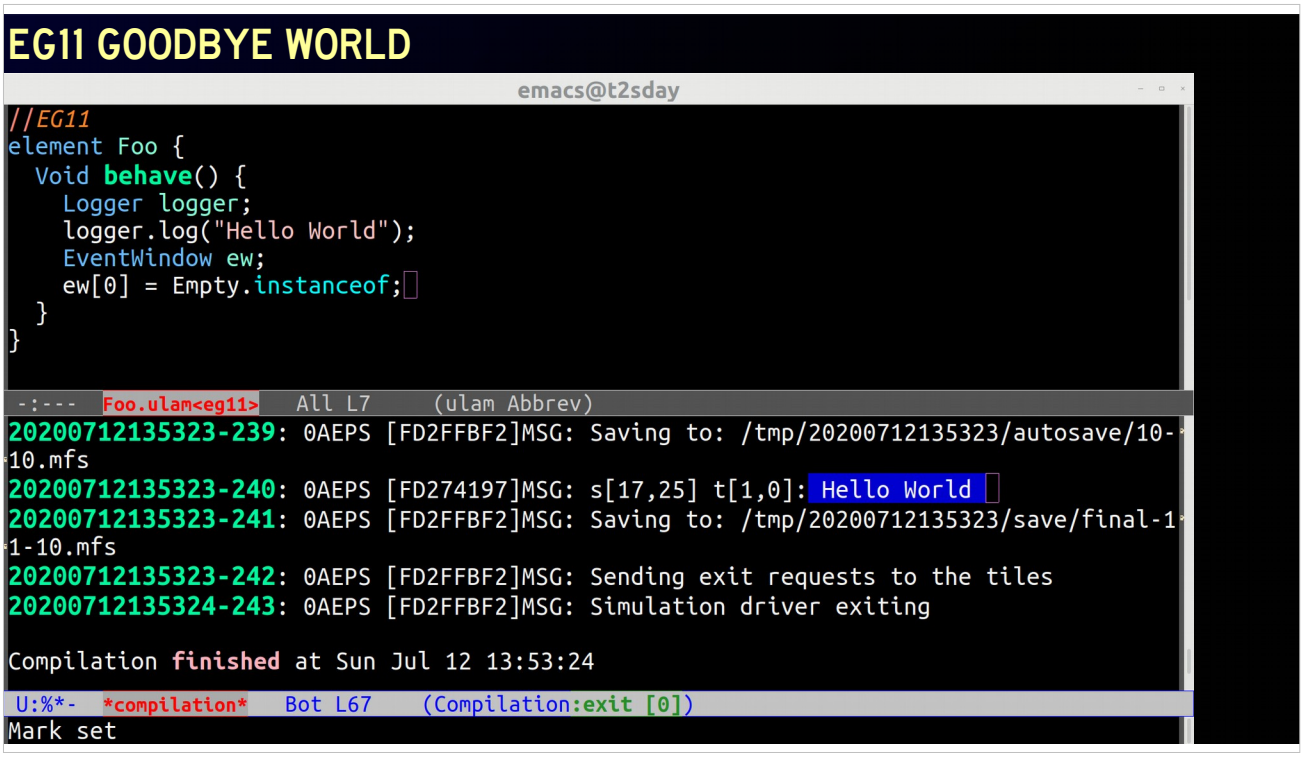
EG11 HELLO? WORLD



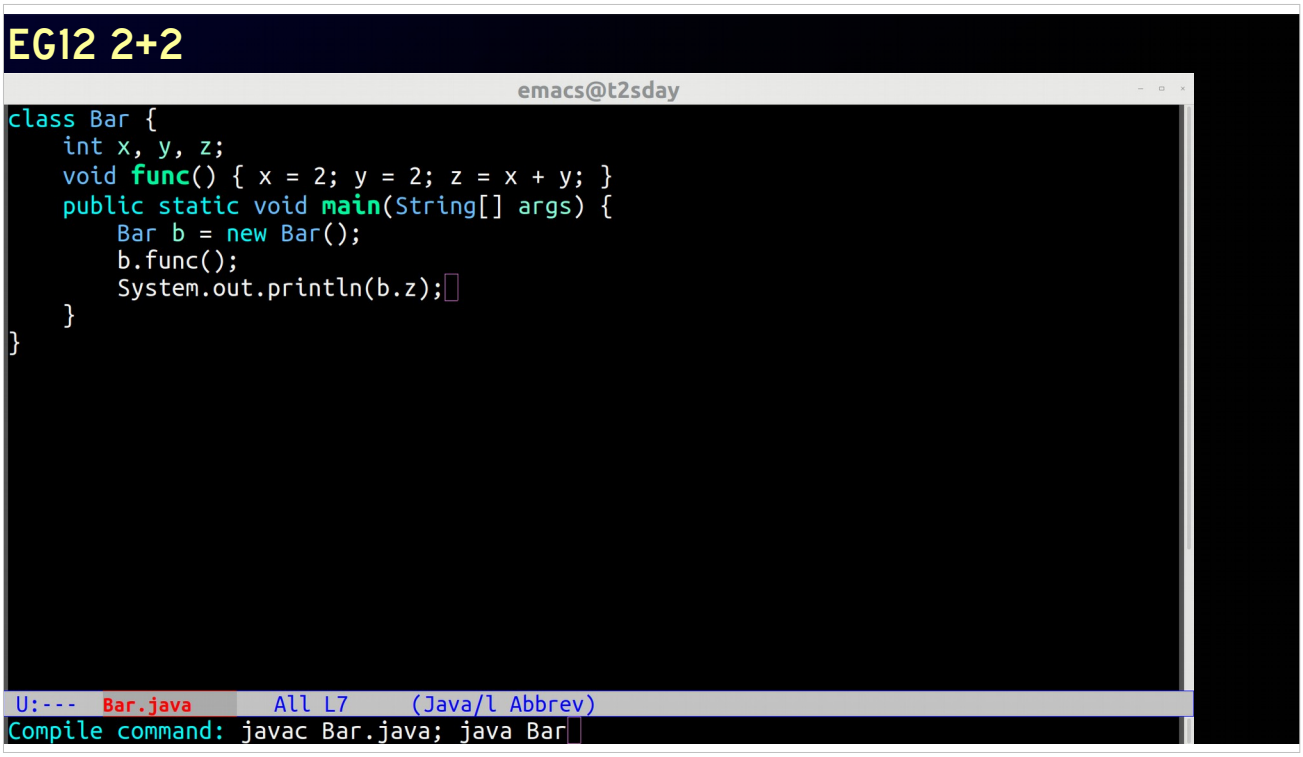
It builds okay, and runs.



And there’s our single instance of ‘Hello World’

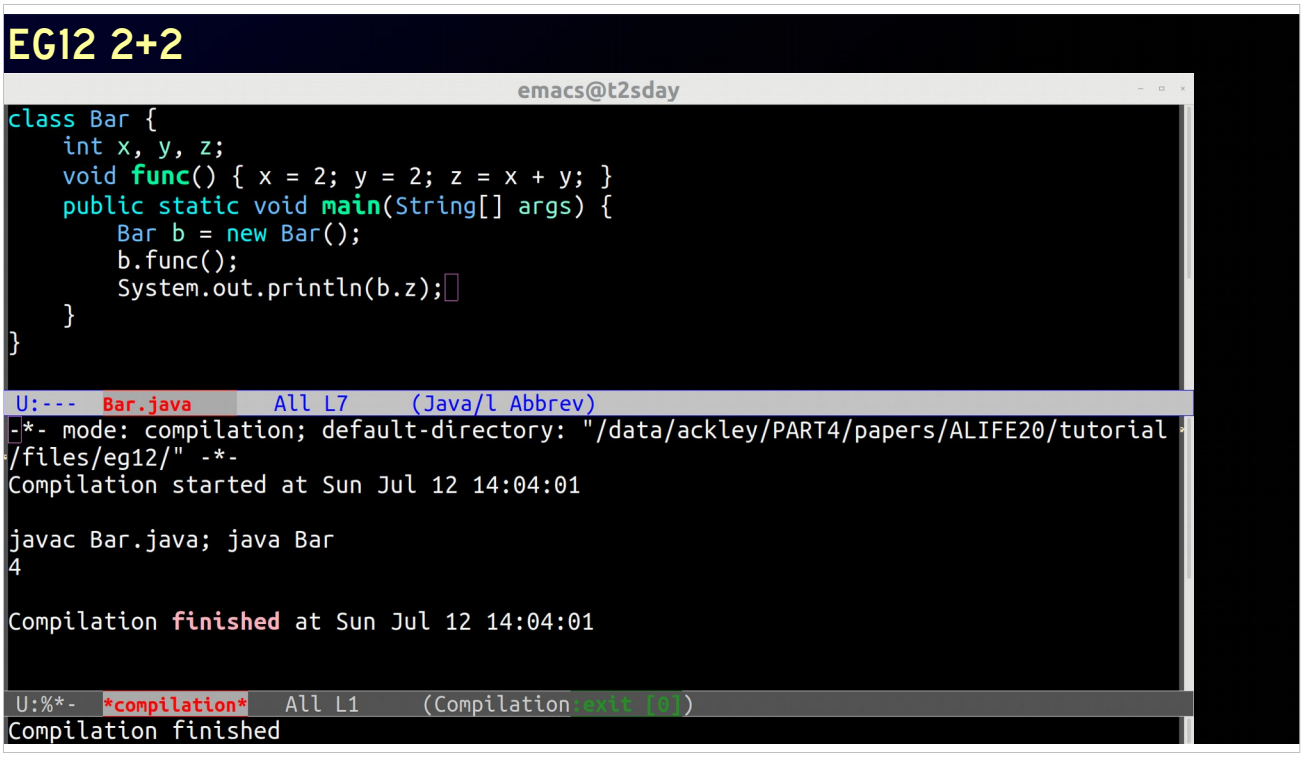


Half-buried in log detritus, but there.

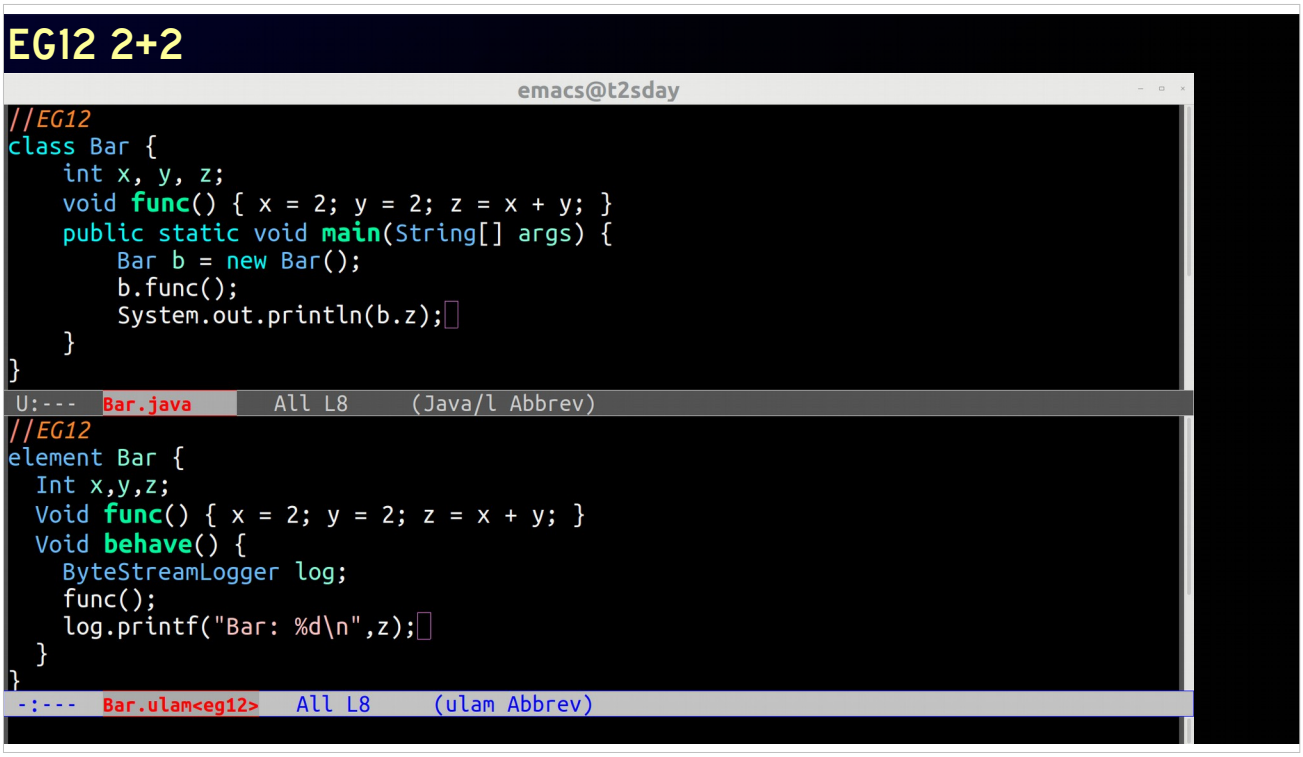


Let's try one more stereotypical first program. Let's add two and two.

Here's a java version. We use three int data members because, you'll see.

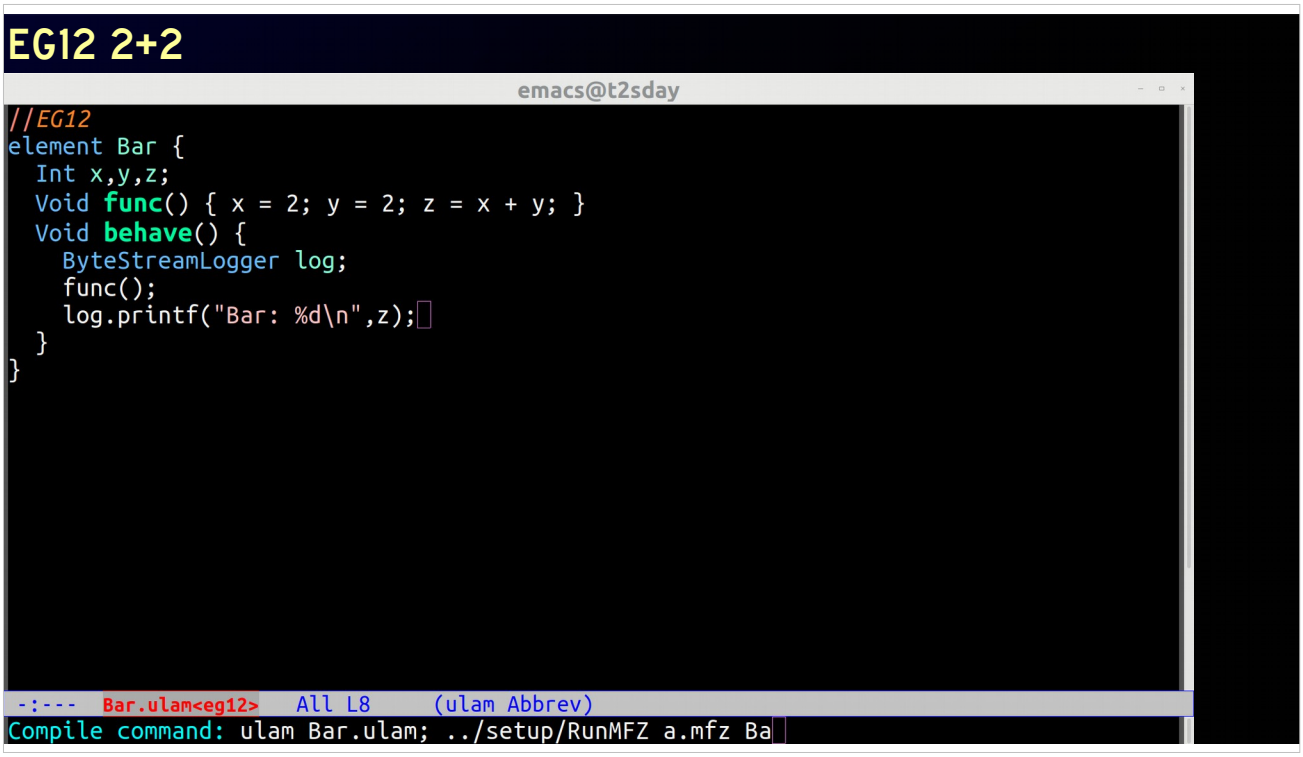


And it computes 4, the right answer

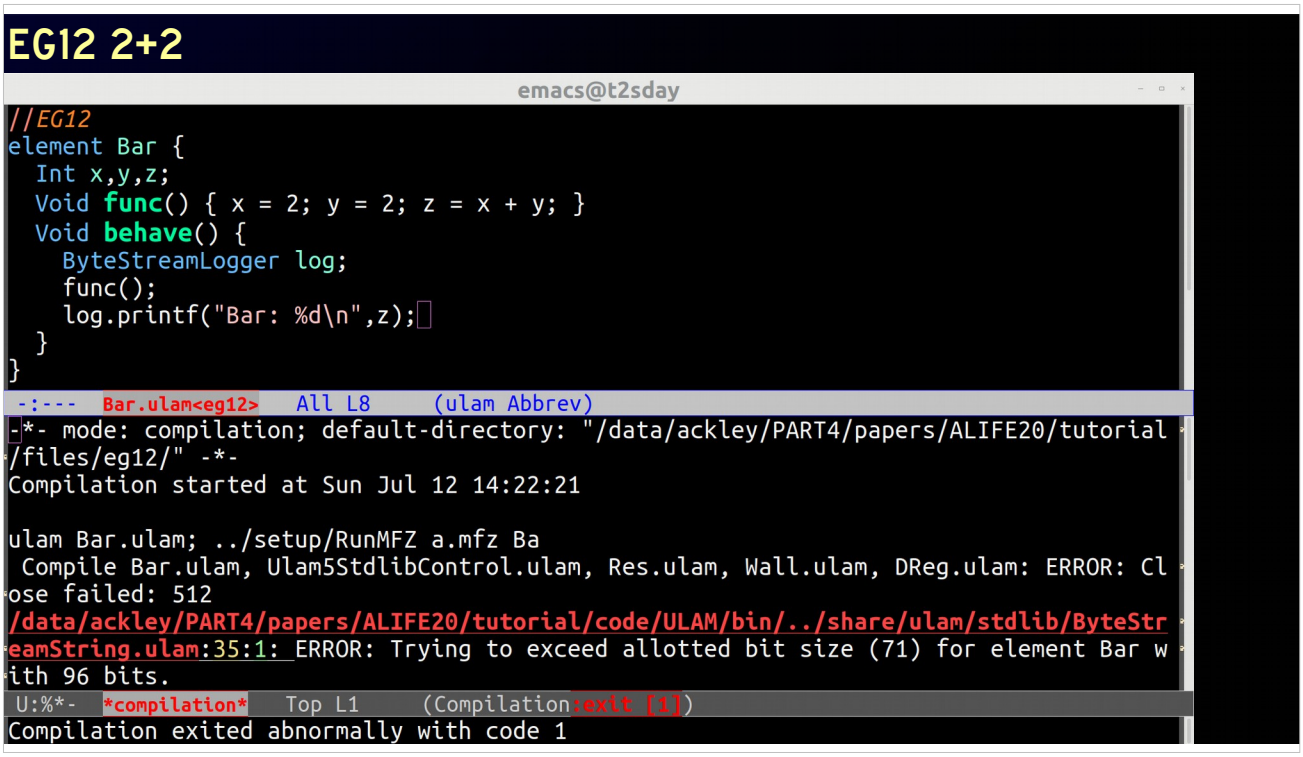


Compare the java code on top to the ulam code on the bottom

Similar, but different. Nothing too shocking.

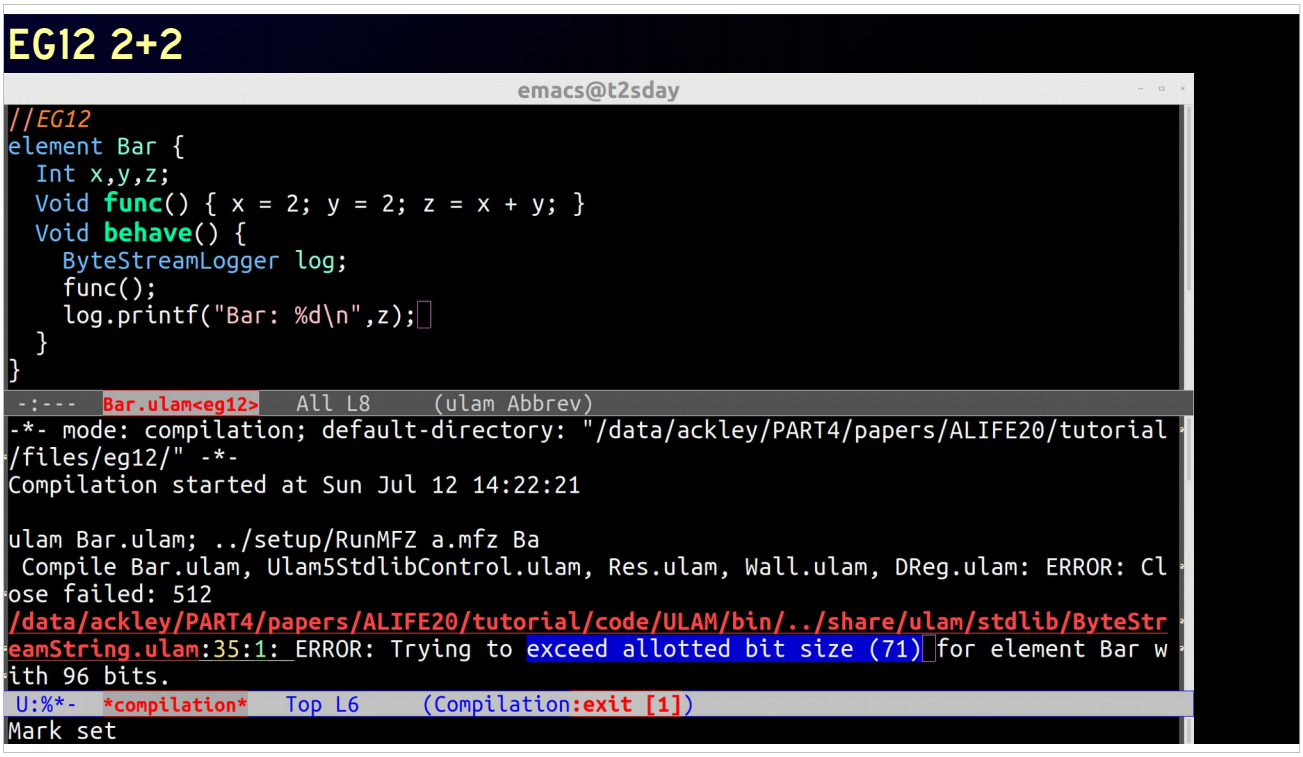


Let’s compile Bar and use RunMFZ a ‘Ba’ element

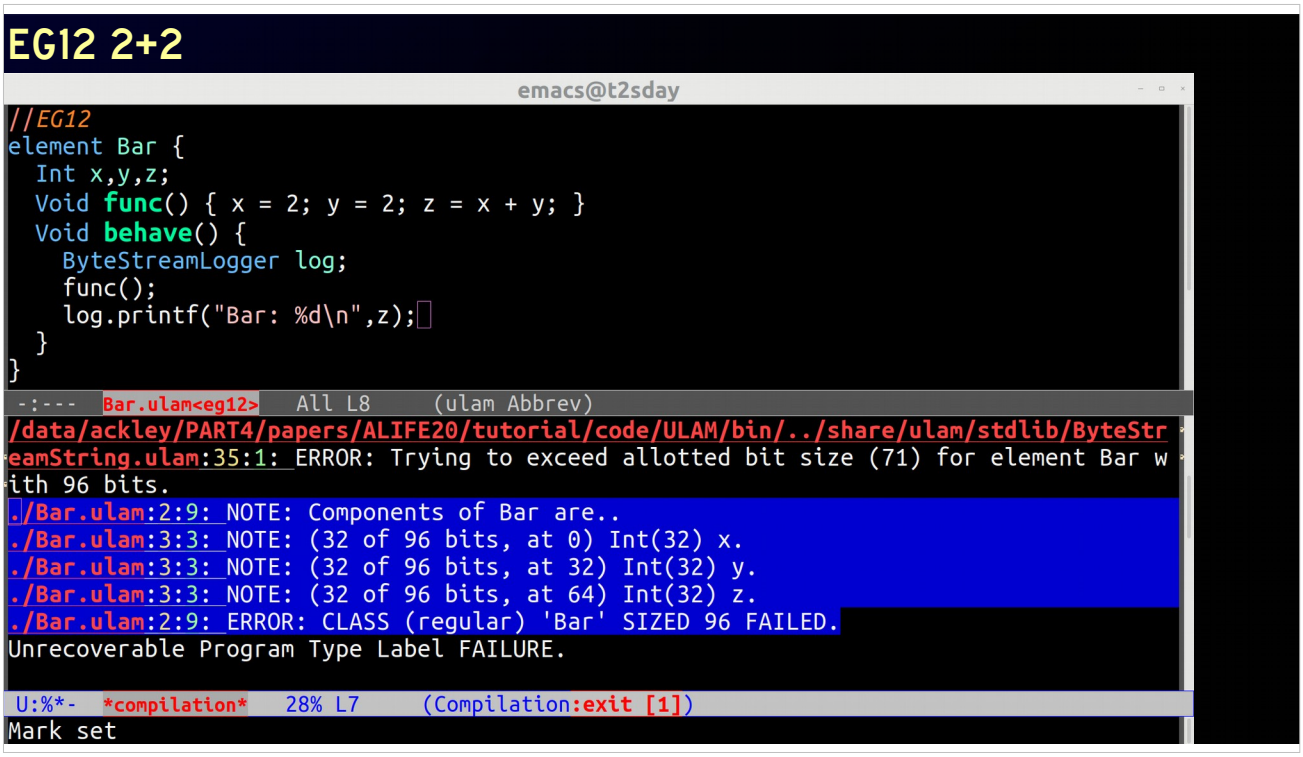


Uh oh

What did we do wrong?

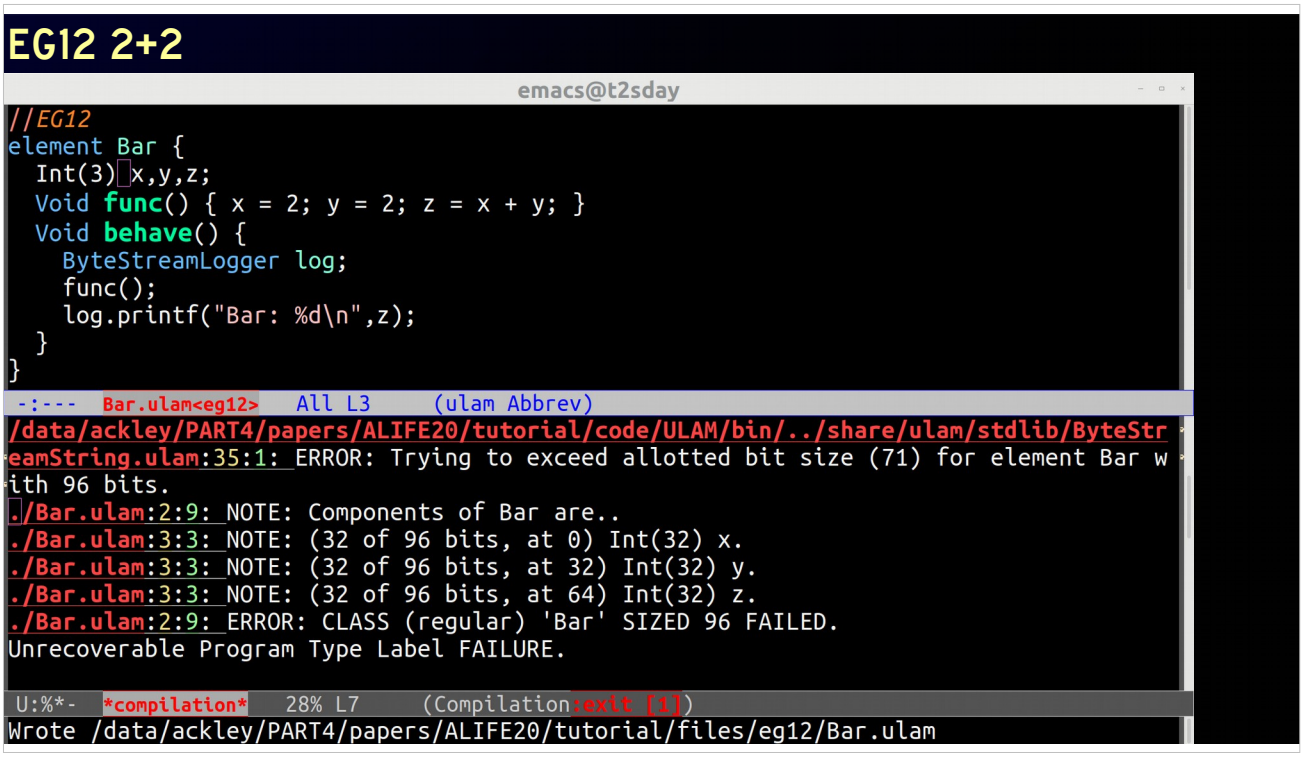


Exceed allotted bit size 71??



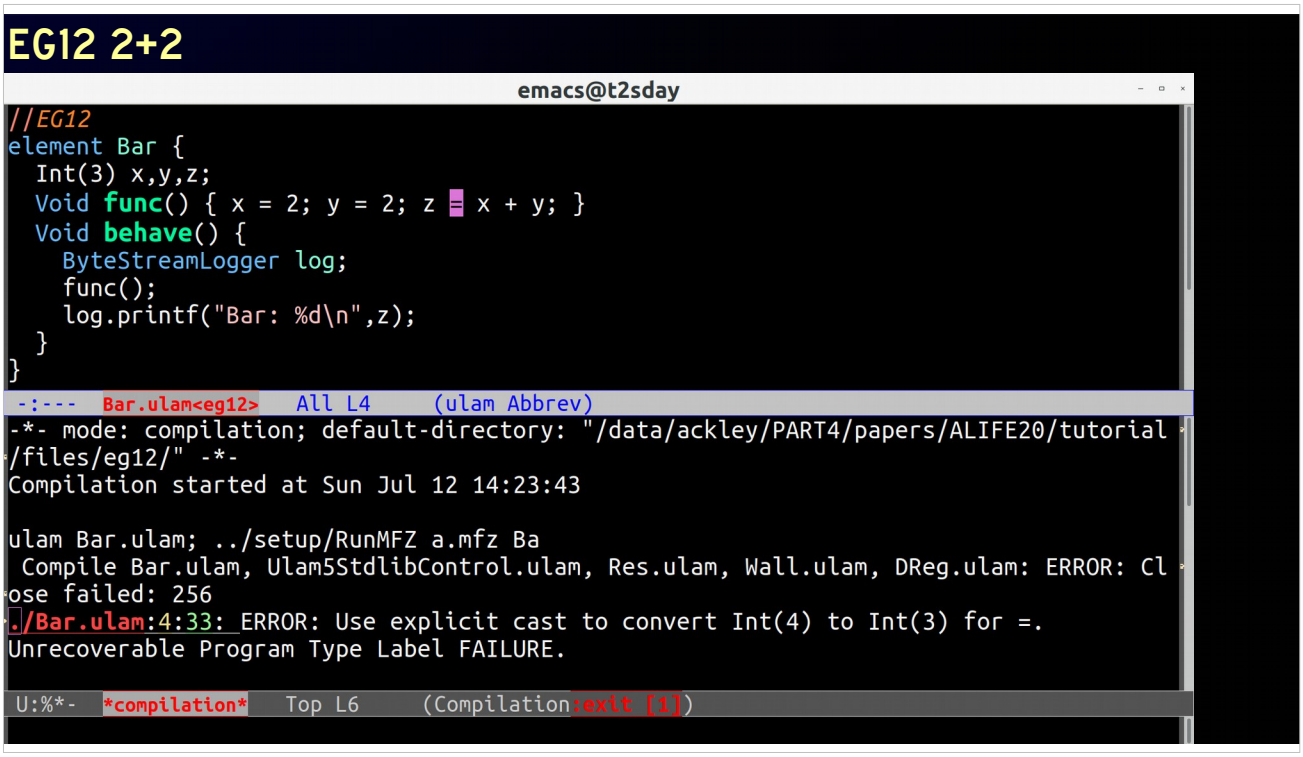
Yeah. 71 bits.

Three 32 bit Ints takes 96 bits. An ulam object – an Atom – isn't that big.



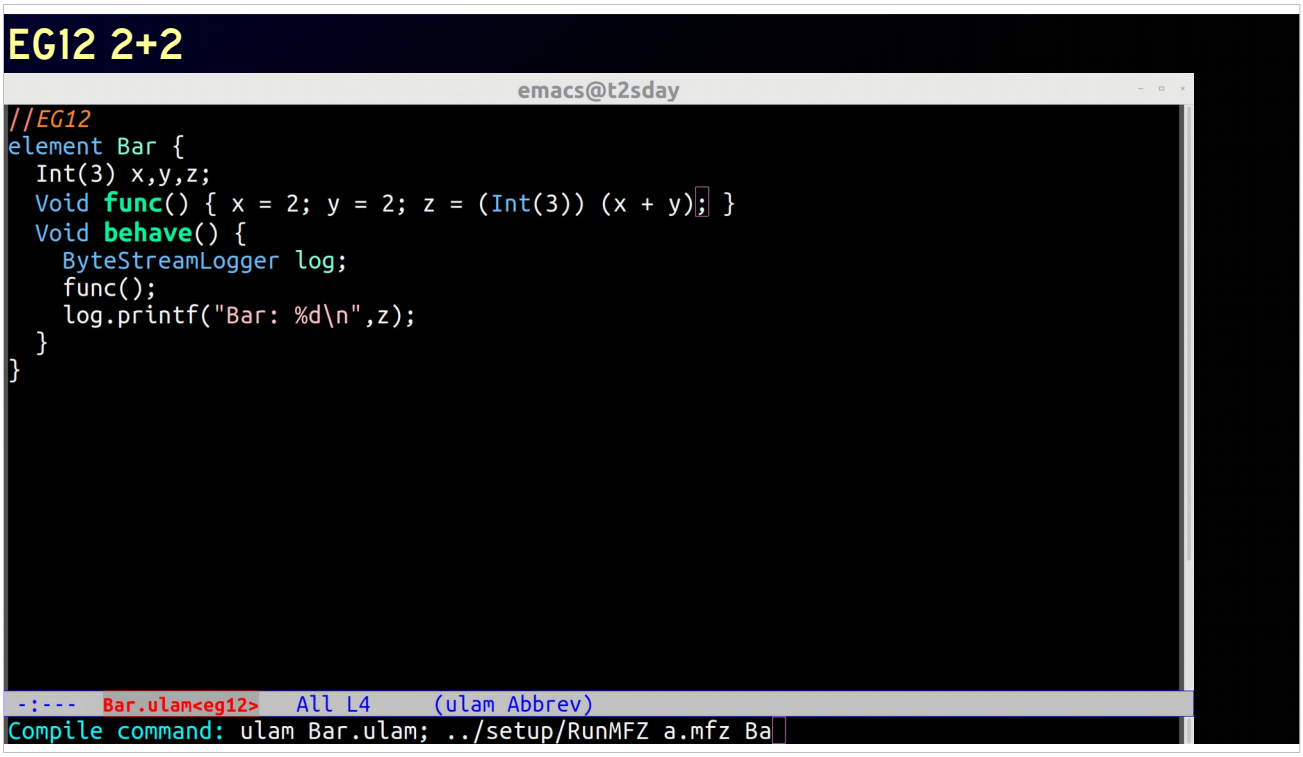
But ulam supports variable sized primitive types like ints.

Let's make them 3 bit ints. Three Int(3)'s will only burn 9 bits.



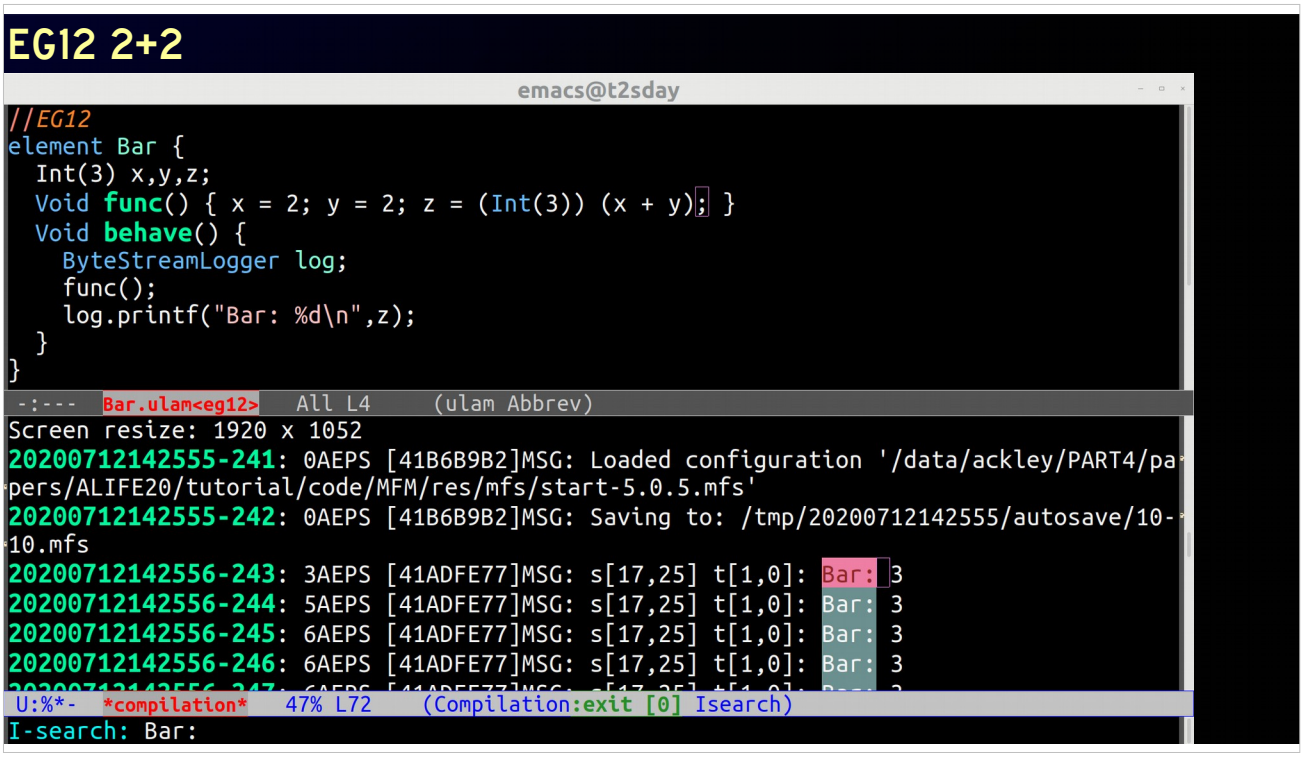
Now what is the compiler mad about??

Well, if you add two Int(3)s, the result could be an Int(4). But z is declared as an Int(3); it might not fit.



So we need to cast the result

Getting careful about primitive sizes and the results of primitive operations is part of the ulam way of life. We embrace it.



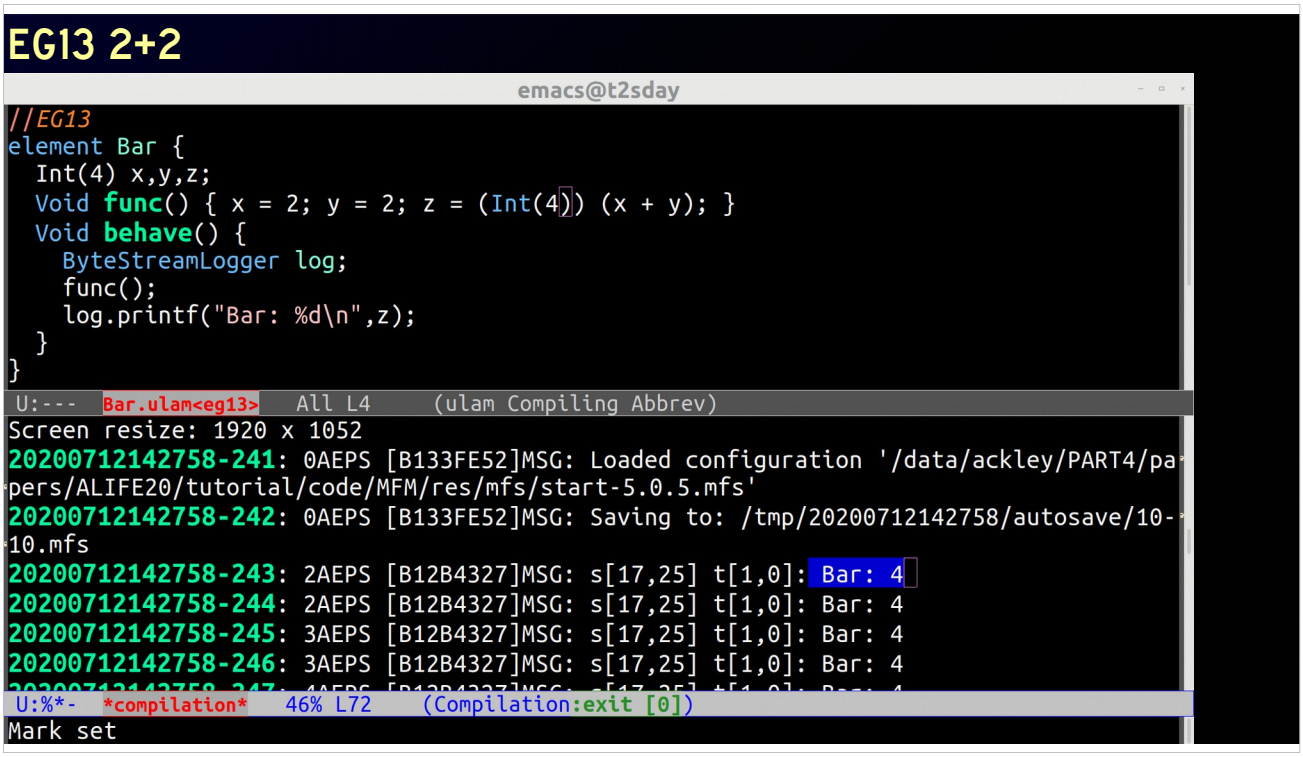
And then it compiles and runs.

And we see (once again over and over), that 2+2 = 3. Wait what?

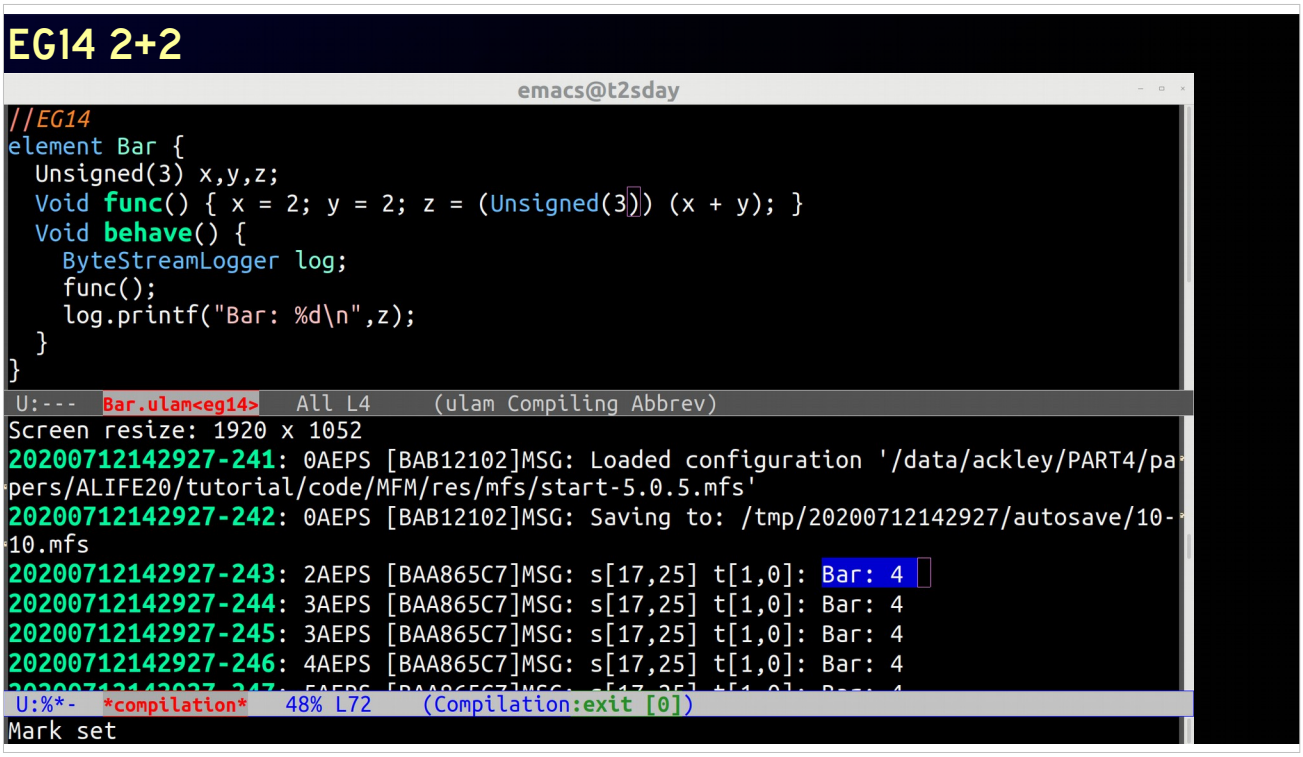
The maximum value of an Int(3) is 3. Four doesn't fit.

But wouldn't we expect the answer to be like negative something, then, due to overflow?

Except ulam arithmetic doesn't overflow, it saturates. Because that's almost always less wrong.

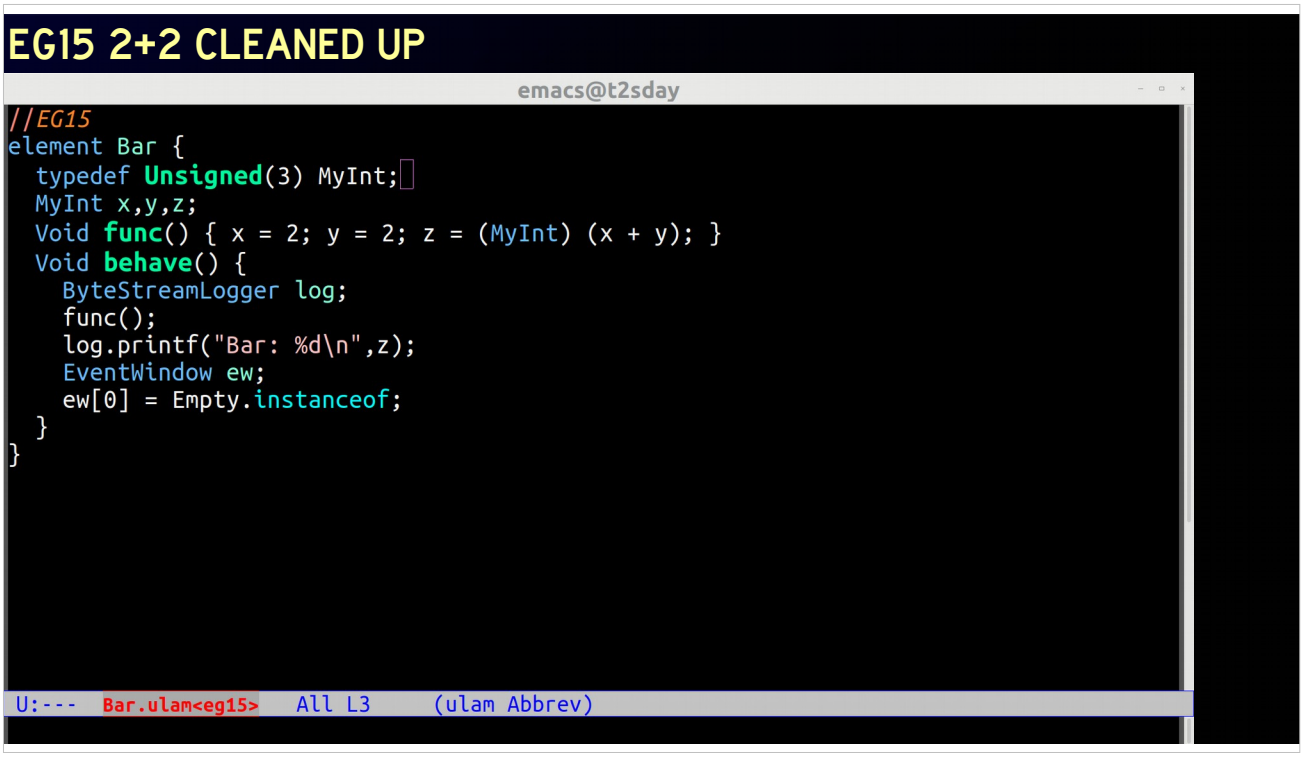


We could use Int(4), which goes from -8 to +7



And if we like we could use Unsigned(3), which covers 0..7

EG15 2+2 CLEANED UP

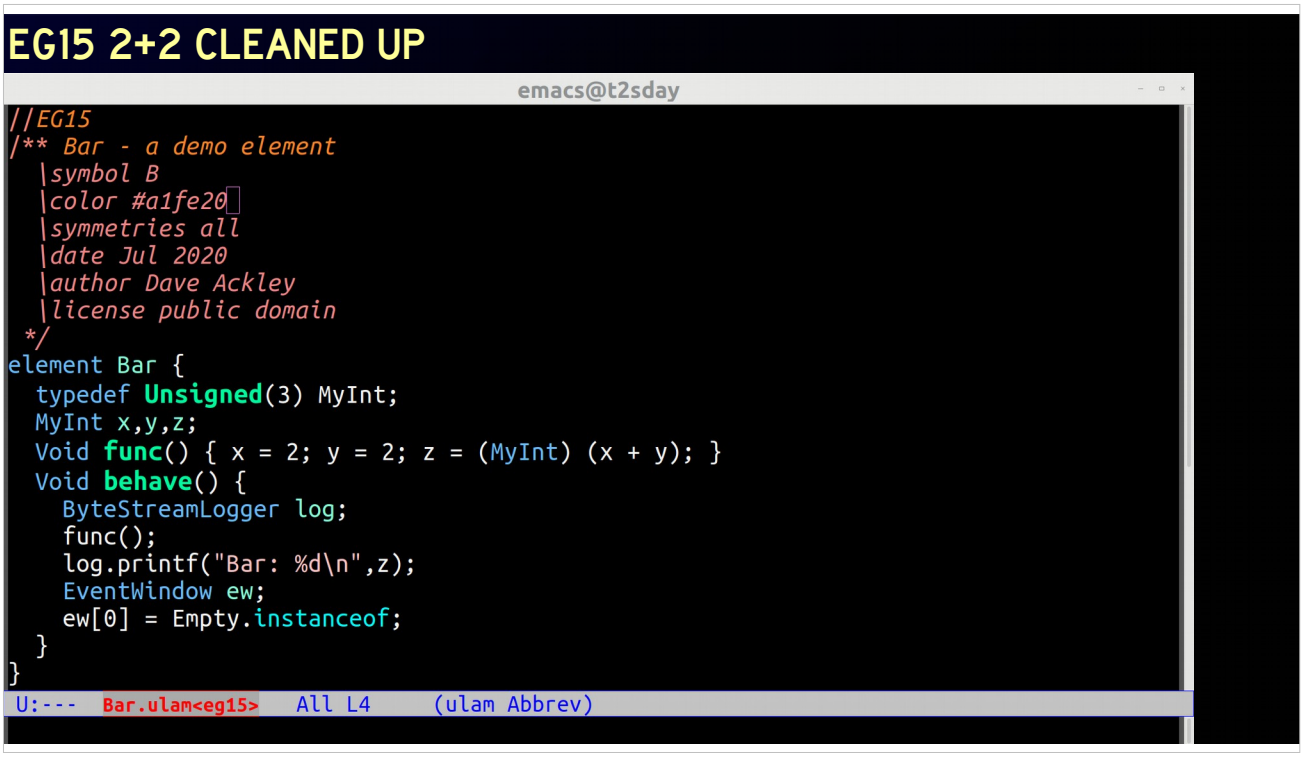


We could clean it up with a typedef

Typedefs are good.

And, here, we politely erase ourselves to simulate an old-fashioned terminating program

EG15 2+2 CLEANED UP



We can also add various kinds of metadata about our element

Those are backslashes in the structured comment, even though the italicizing almost stands them upright.

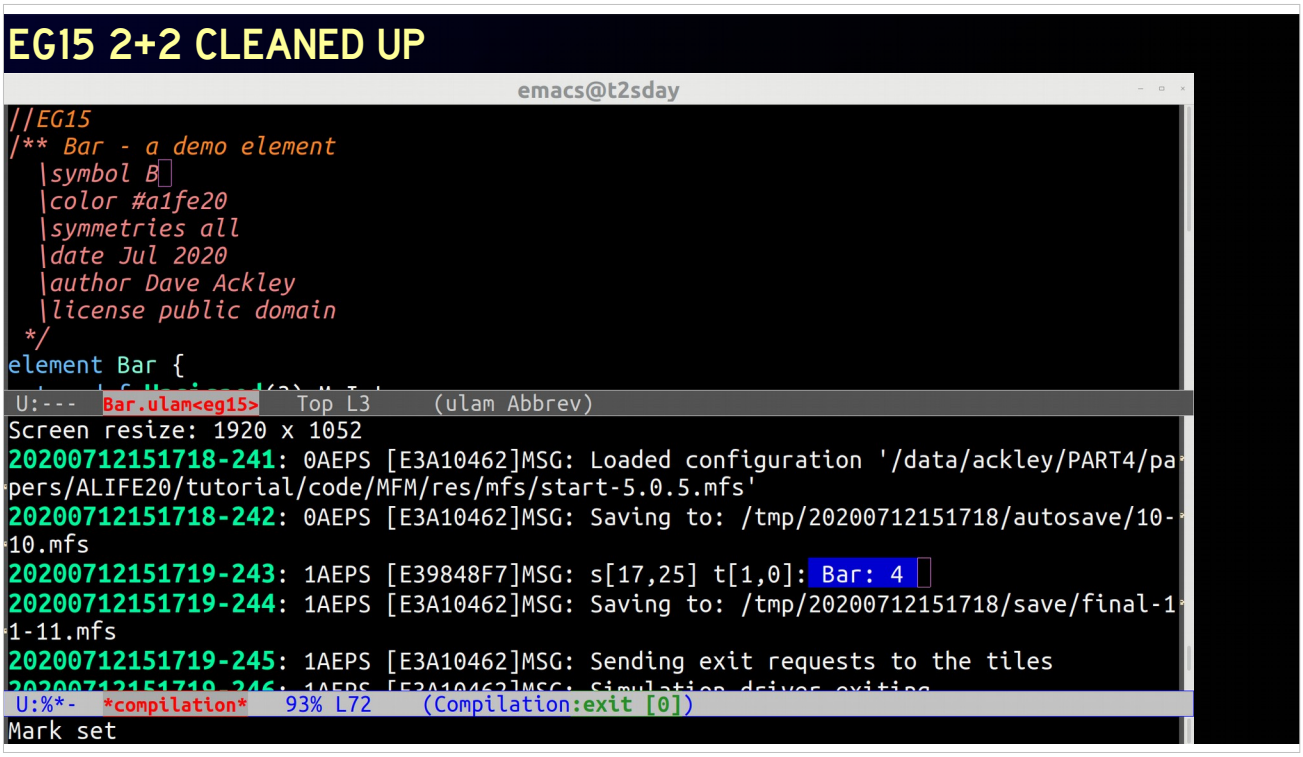
EG15 2+2 CLEANED UP

emacs@t2sday

```
//EG15
/** Bar - a demo element
  \symbol B
  \color #a1fe20
  \symmetries all
  \date Jul 2020
  \author Dave Ackley
  \license public domain
 */
element Bar {
  typedef Unsigned(3) MyInt;
  MyInt x,y,z;
  Void func() { x = 2; y = 2; z = (MyInt) (x + y); }
  Void behave() {
    ByteStreamLogger log;
    func();
    log.printf("Bar: %d\n",z);
    EventWindow ew;
    ew[0] = Empty.instanceof;
  }
}
```

```
U:---  Bar.ulam<eg15>   All L4     (ulam Abbrev)
Compile command: ulam Bar.ulam; ../setup/RunMFZ a.mfz B
```

Now that we declared Bar's atomic symbol is 'B' – not 'Ba' – that's what goes in the RunMFZ command

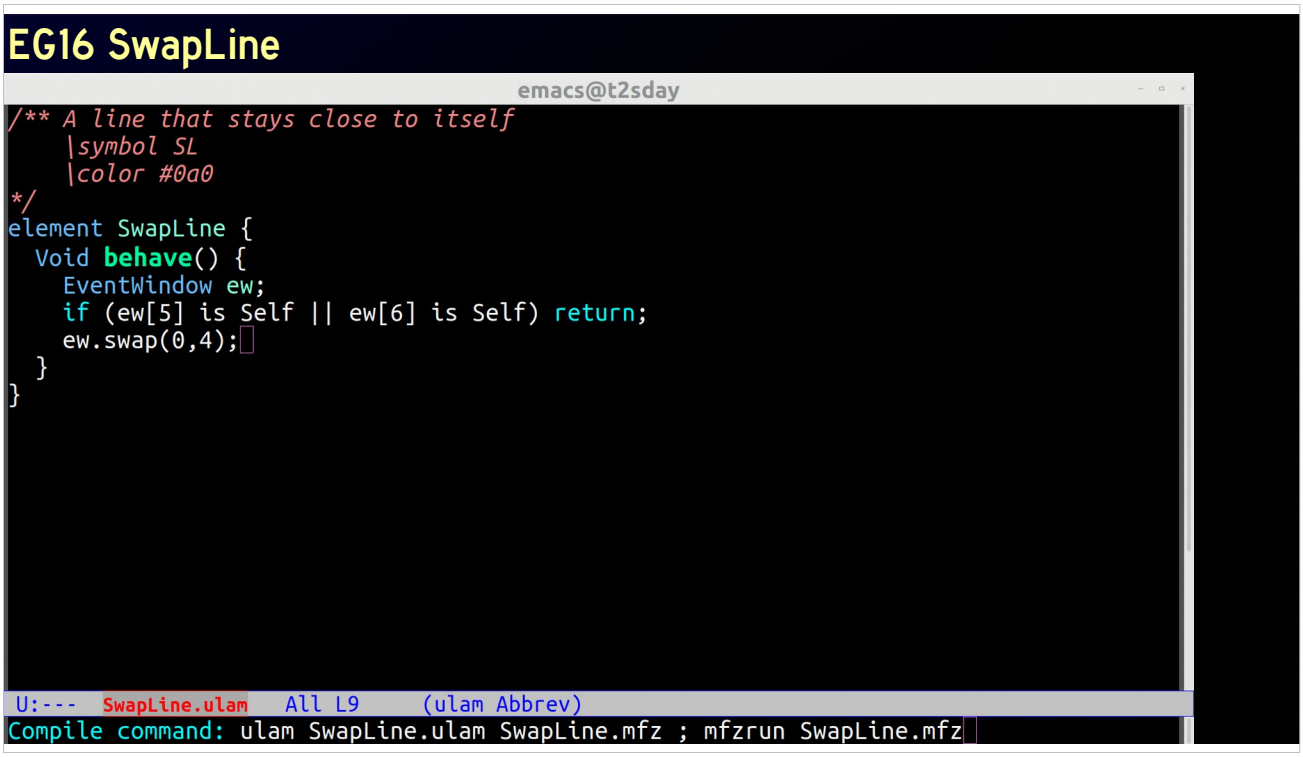


And it all still works.

ULAM TAKE-AWAYS PART I

- Atoms ('objects') are instances of Elements ('classes')
  - An atom has only 71 bits of state
  - All atoms are the same size
- State transition code is per-element, not per atom
  - Void behave() is the entry point for a state transition
  - The behave()-ing atom is stored in the 'event window', at ew[0]
- The initial configuration is (generally) determined externally
  - Do not try and call the main. That's impossible. Instead, ...

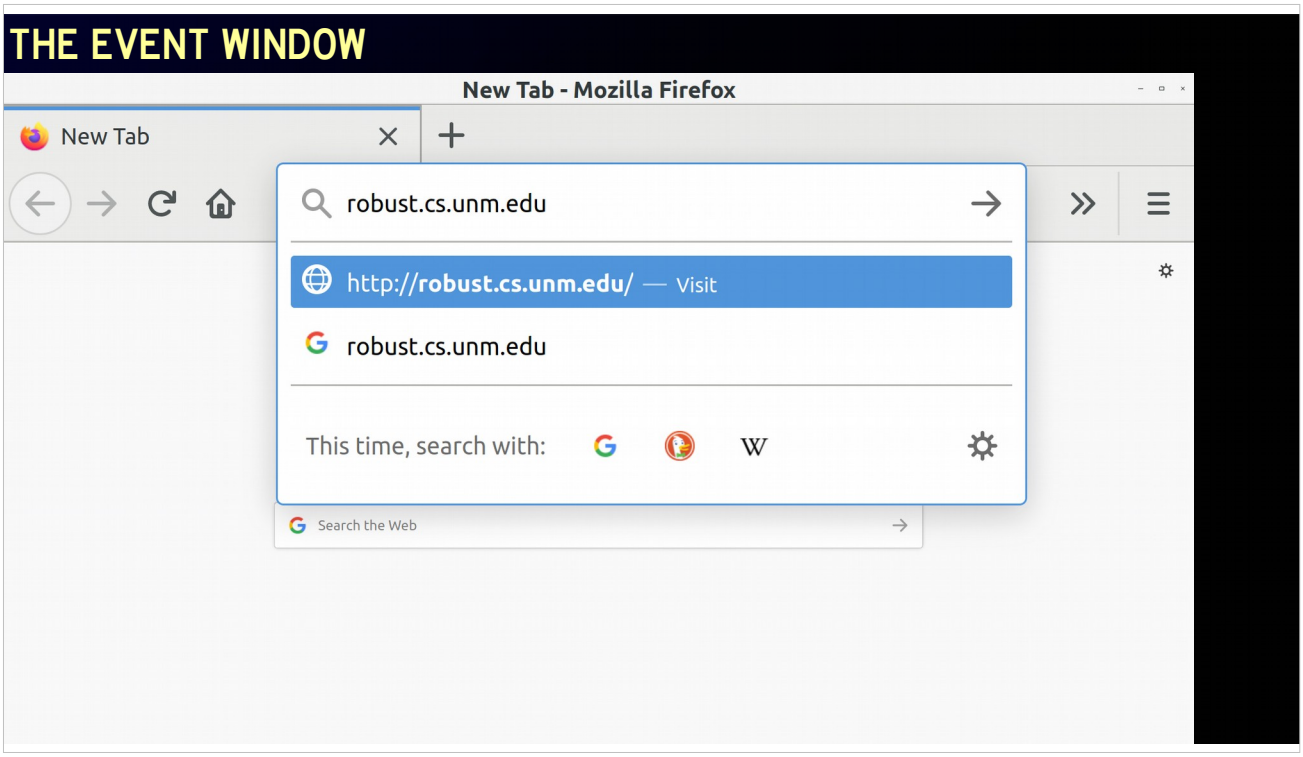
EG16 SwapLine



This is the primordial SwapLine

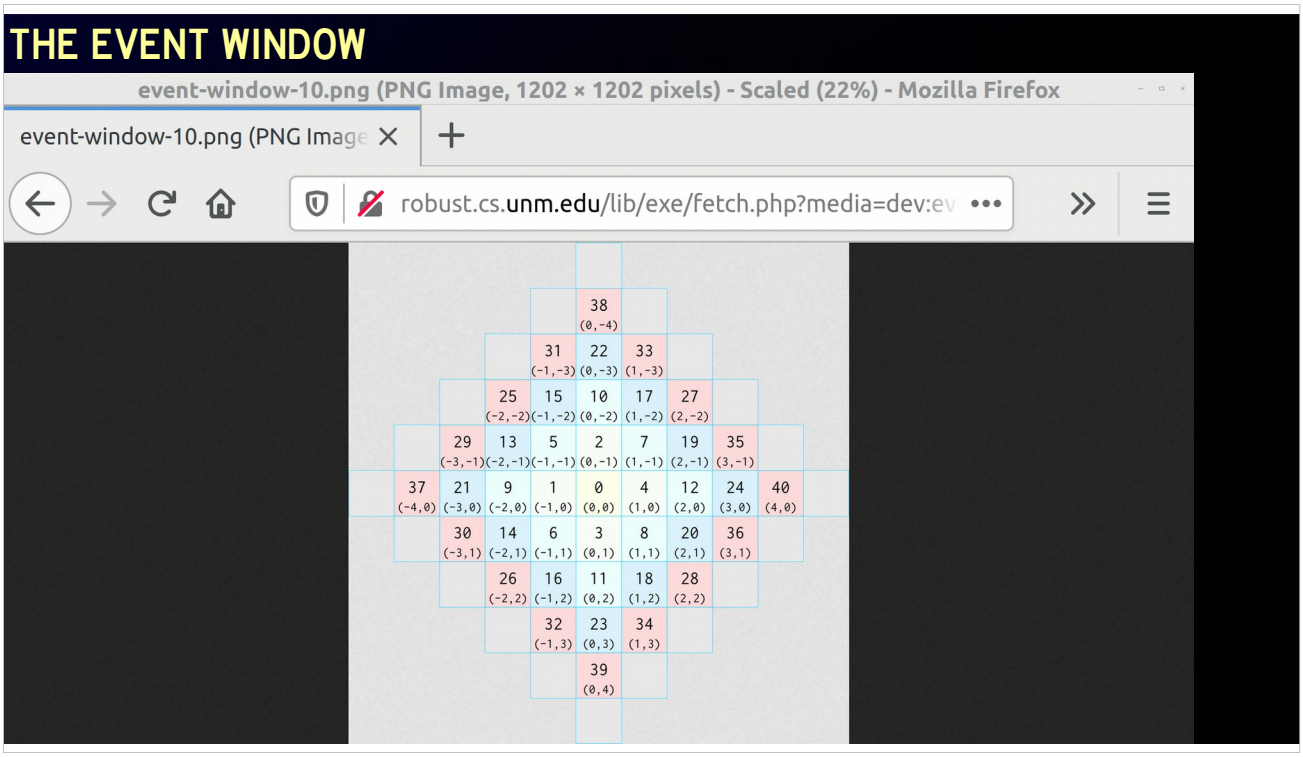
I'll be giving a lightning talk about some great grandchildren of SwapLine on Friday.

But for now it's just about the EventWindow. What's ew[5]?

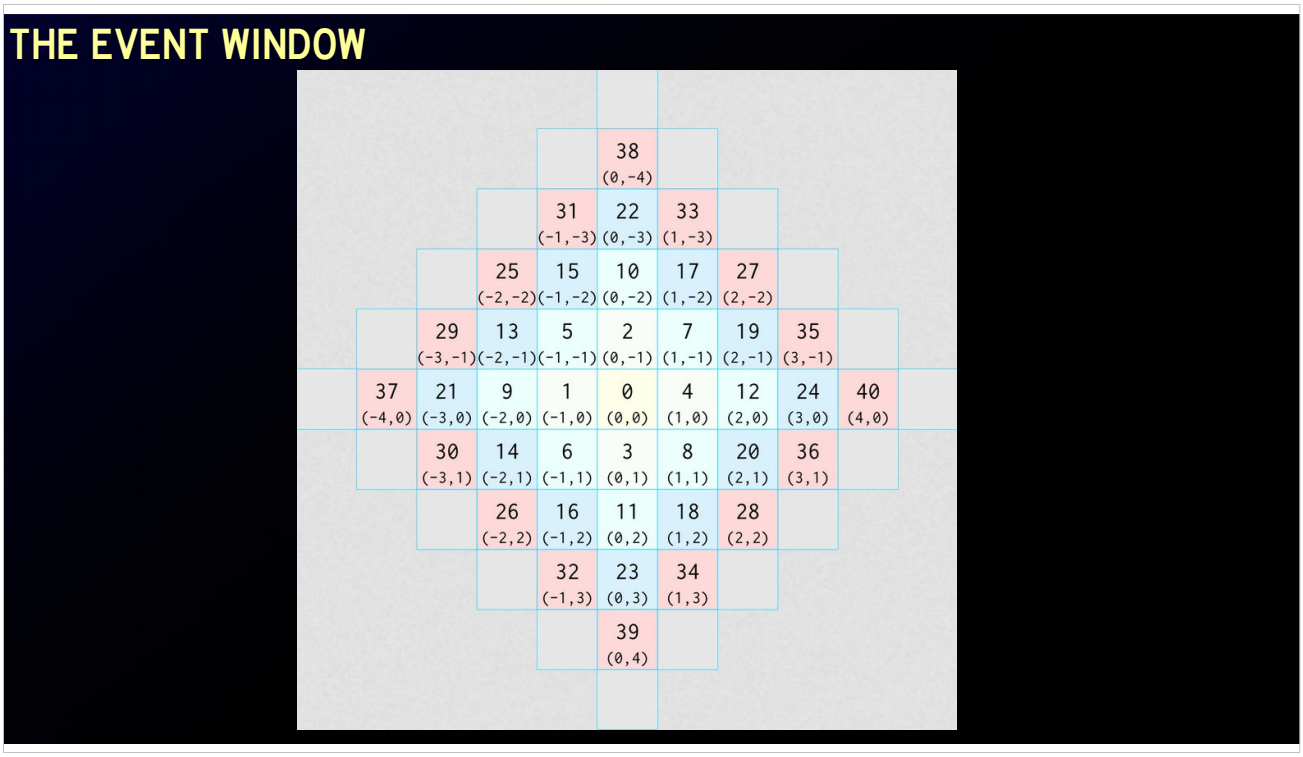


I always go here

To find the event window image



Showing the various indexing schemes that ulam code can use to examine its environment

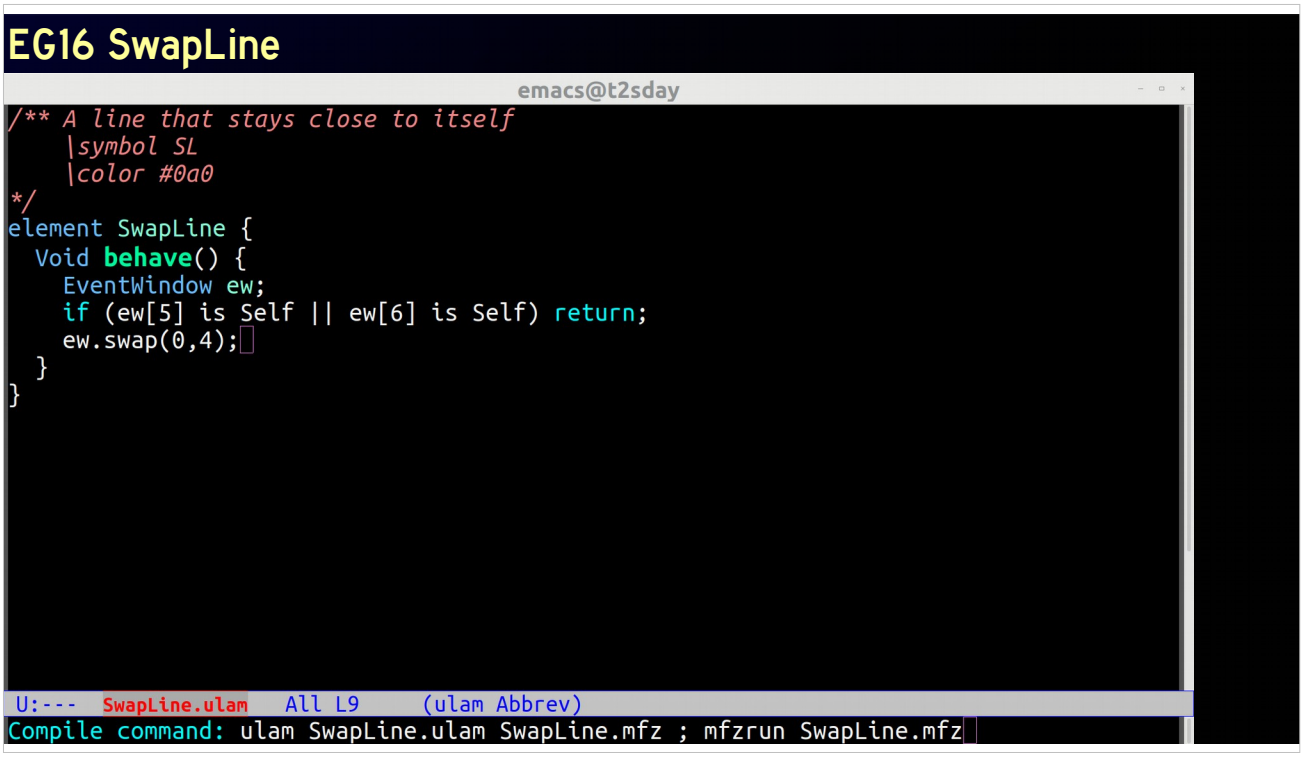


ew[0] is always ‘me’.

So ew[5] is like ‘Northwest of me’

Similarly for ew[6]

EG16 SwapLine

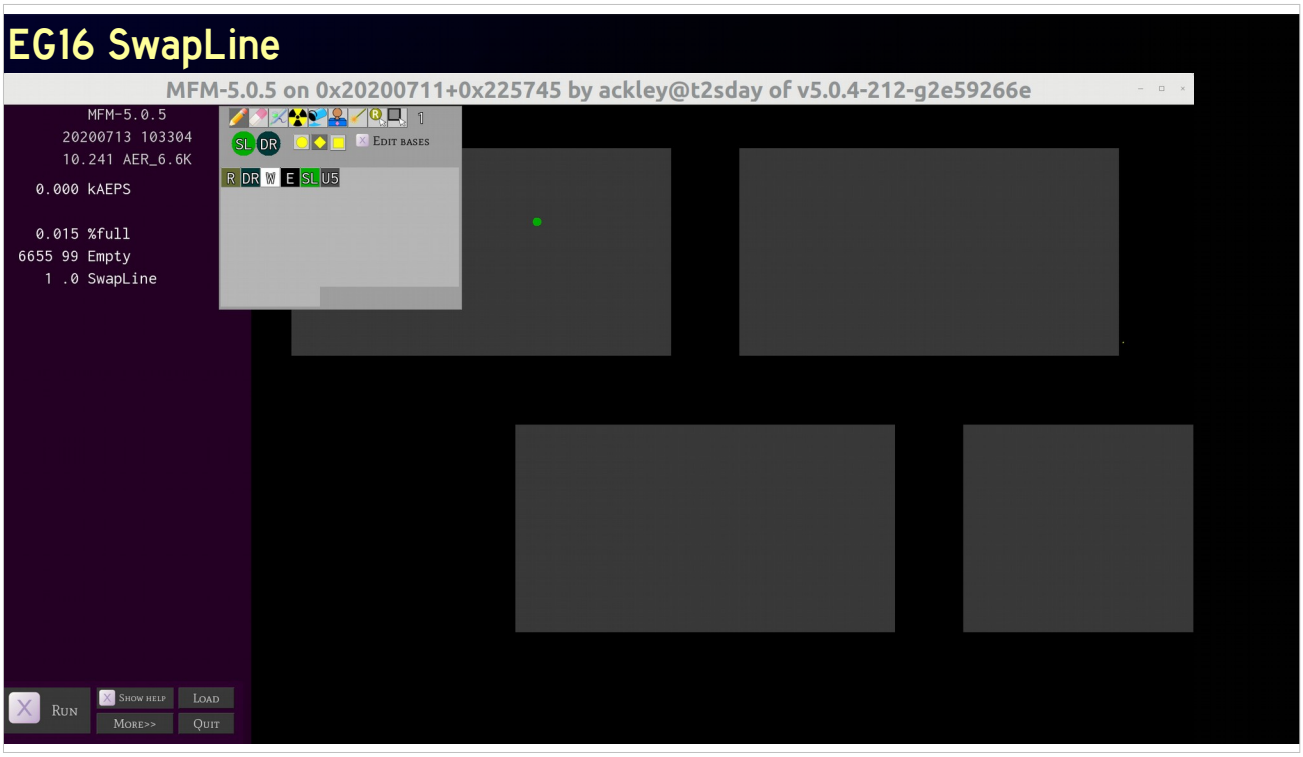


This code also uses capital Self, a predefined type meaning ‘my own type’ – so ‘if (ew[5] is Self)’ means if nw is also a SwapLine like I am

Let’s stop using RunMFZ and be ‘official’ by using mfzrun. We can specify an .mfz output file on the ulam command if we don’t want to end up with ‘a.mfz’. So let’s use SwapLine.mfz, and run that..

Where's the SwapLine?

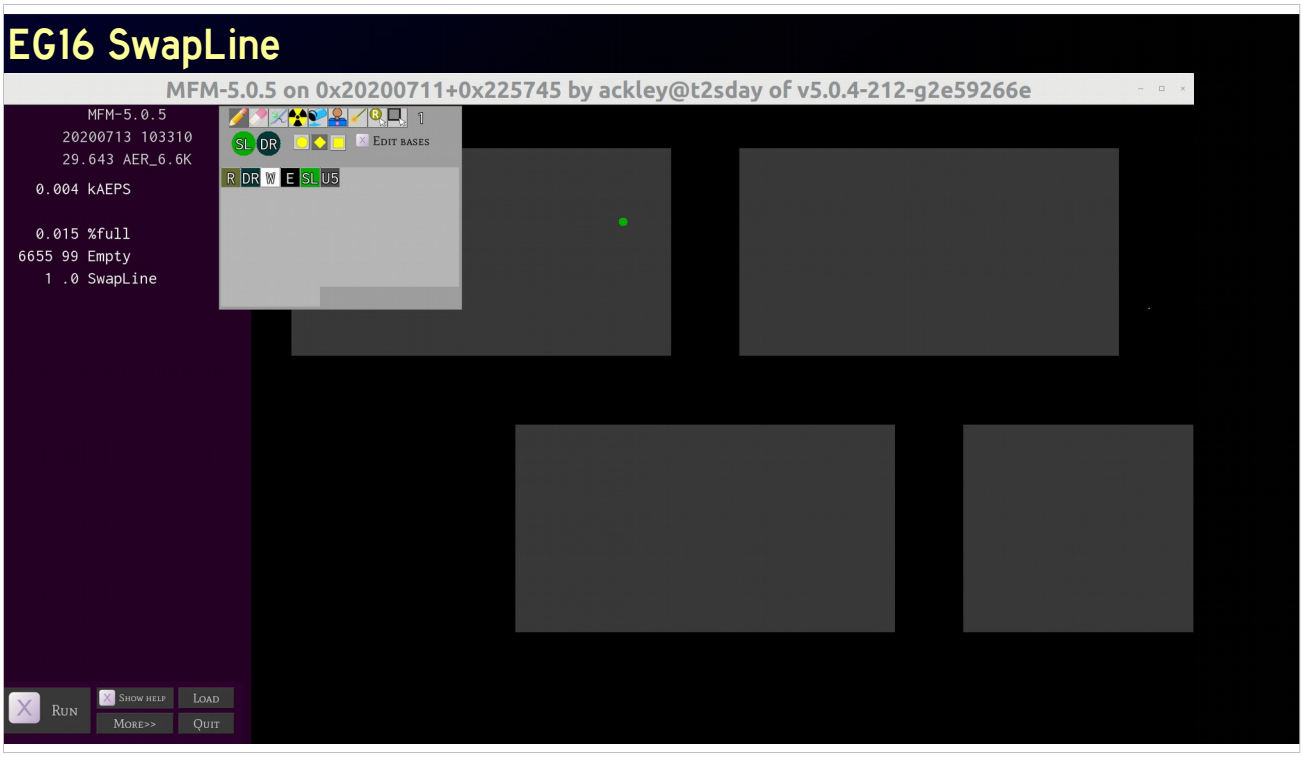
The single atom we were seeing before came from RunMFZ. Normally the world starts out empty, as here.



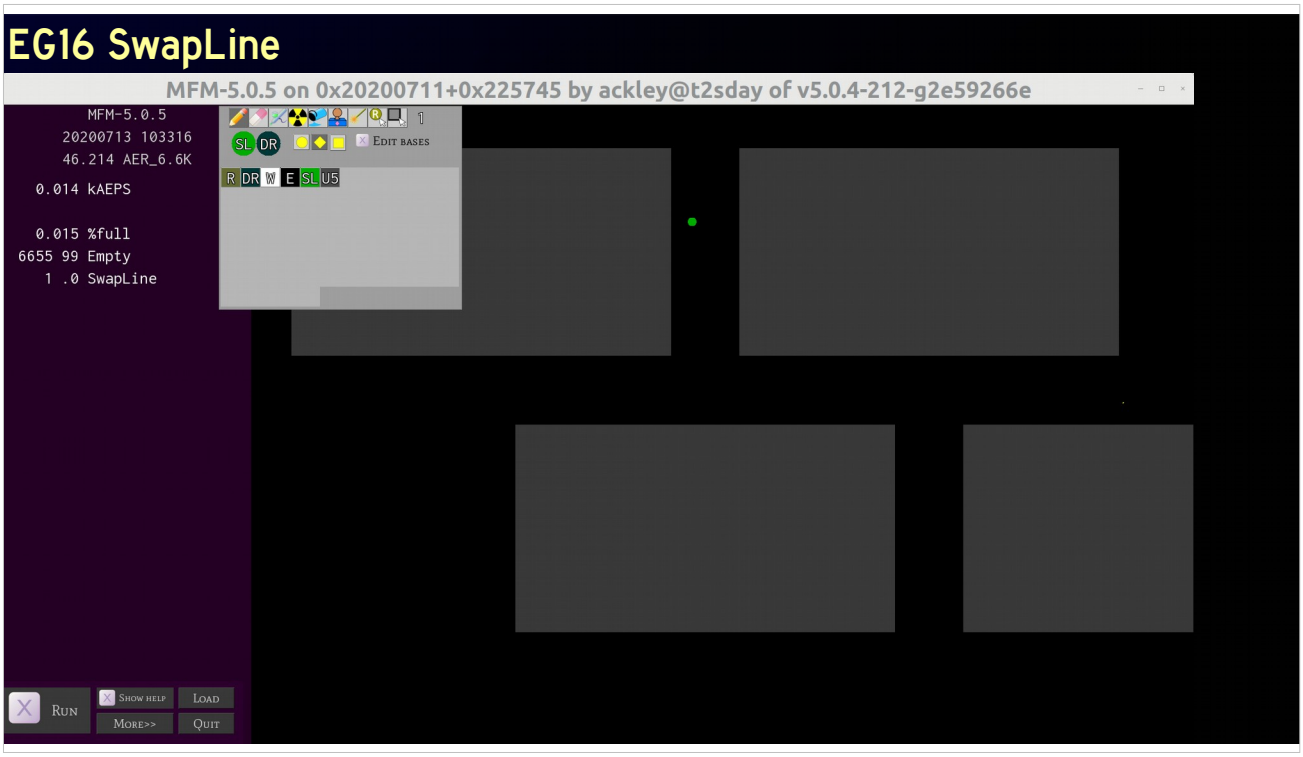
But we bring up the tool palette with ‘t’

And click left on ‘SL’ to select it

And use the pencil tool to create an SL in the grid.

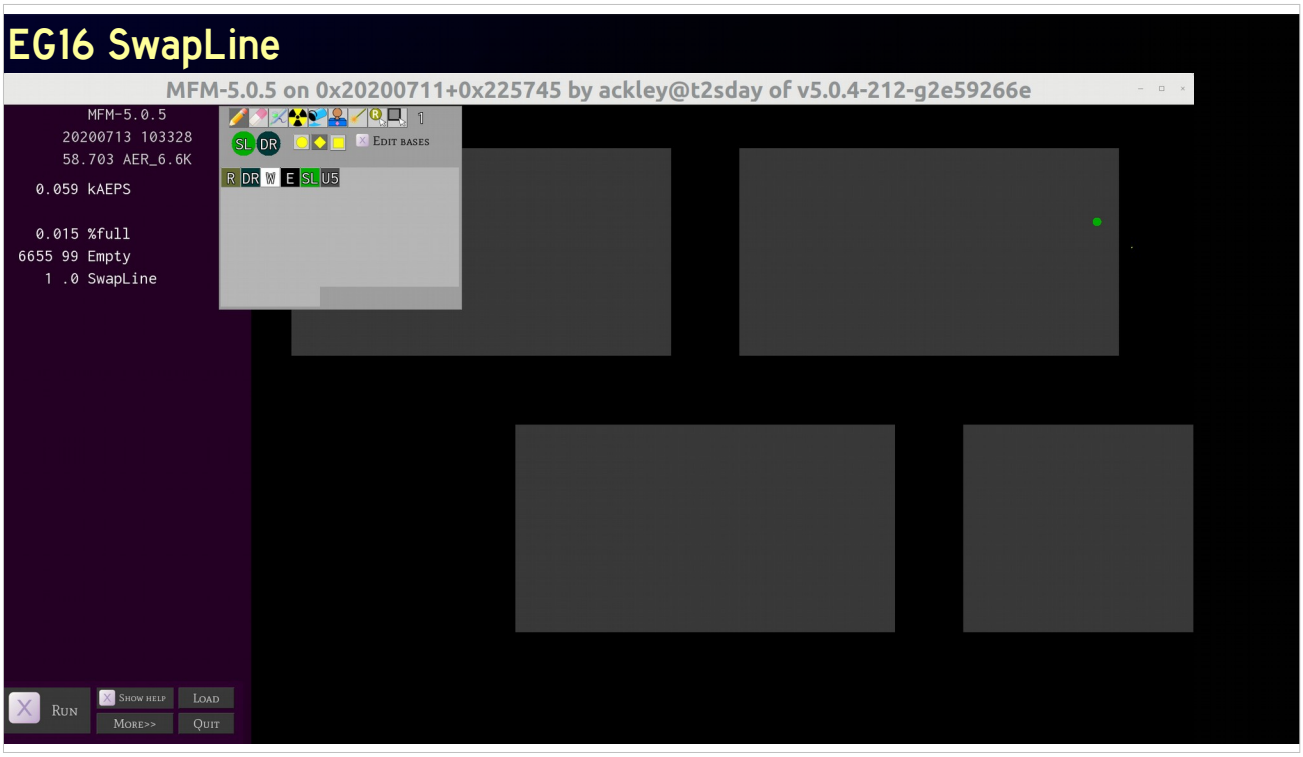


And if we click Run on and off..

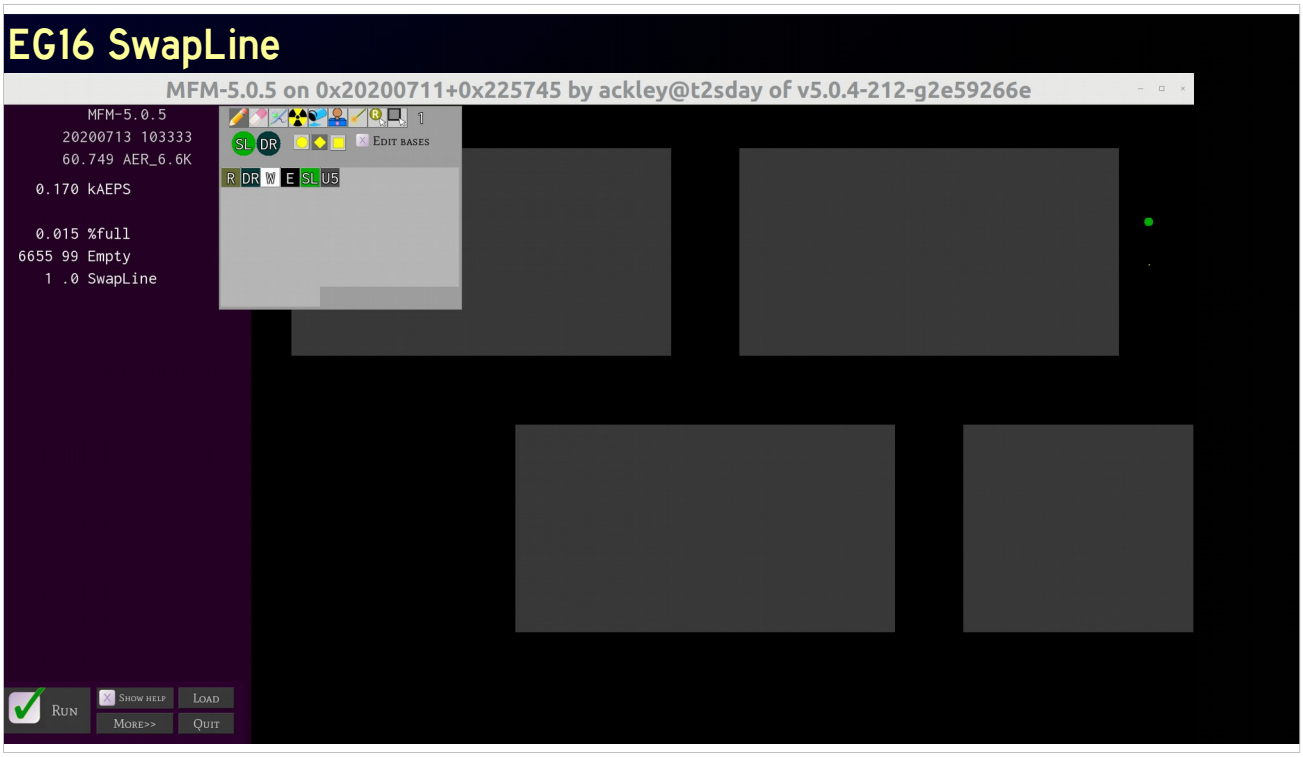


We see it heading east.

Why? Because its NW and SW are always empty, so it does the swap with east.

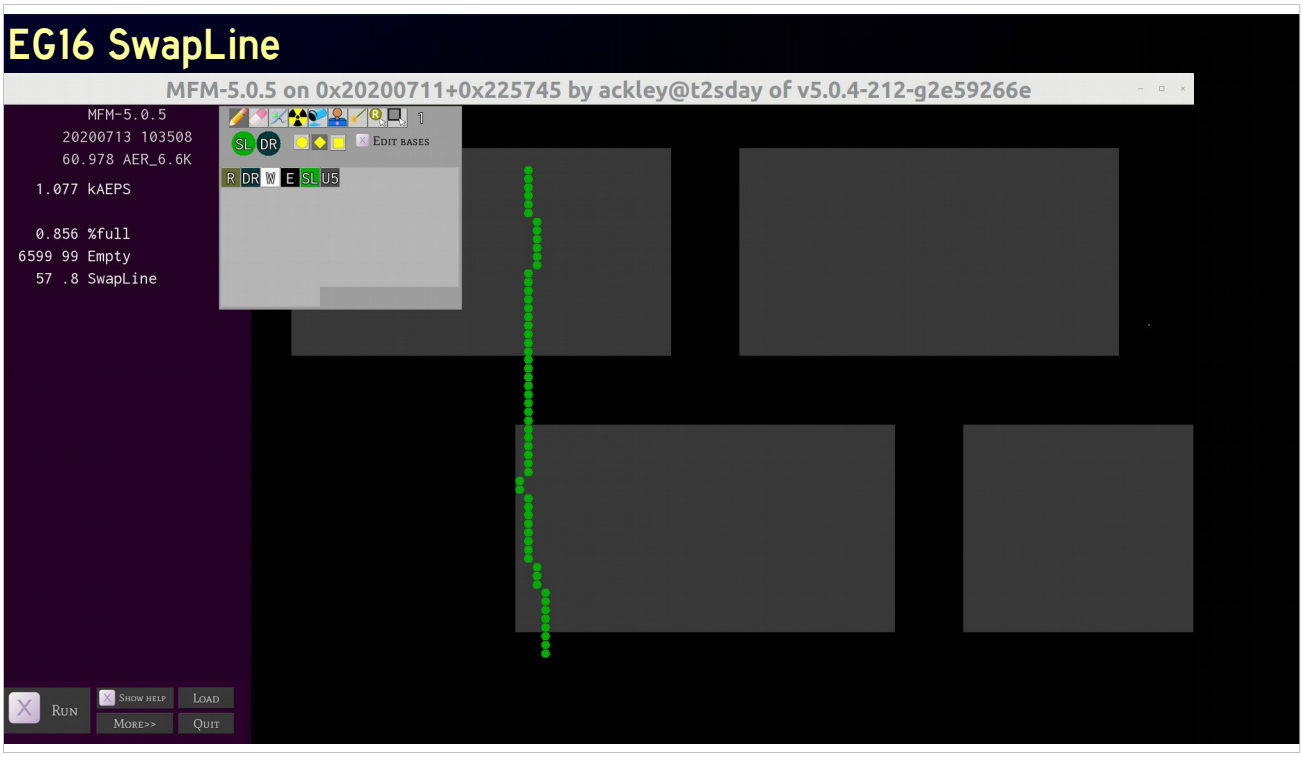


Until eventually

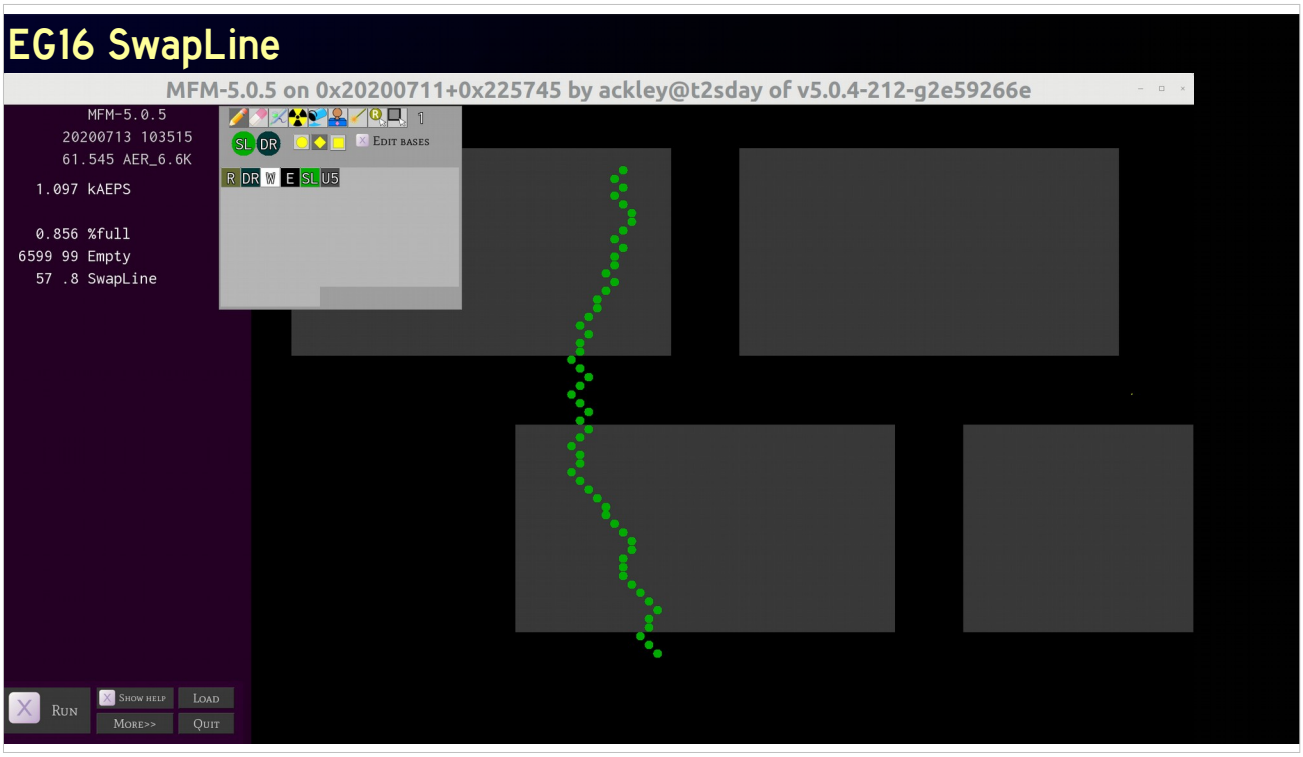


It runs out of universe

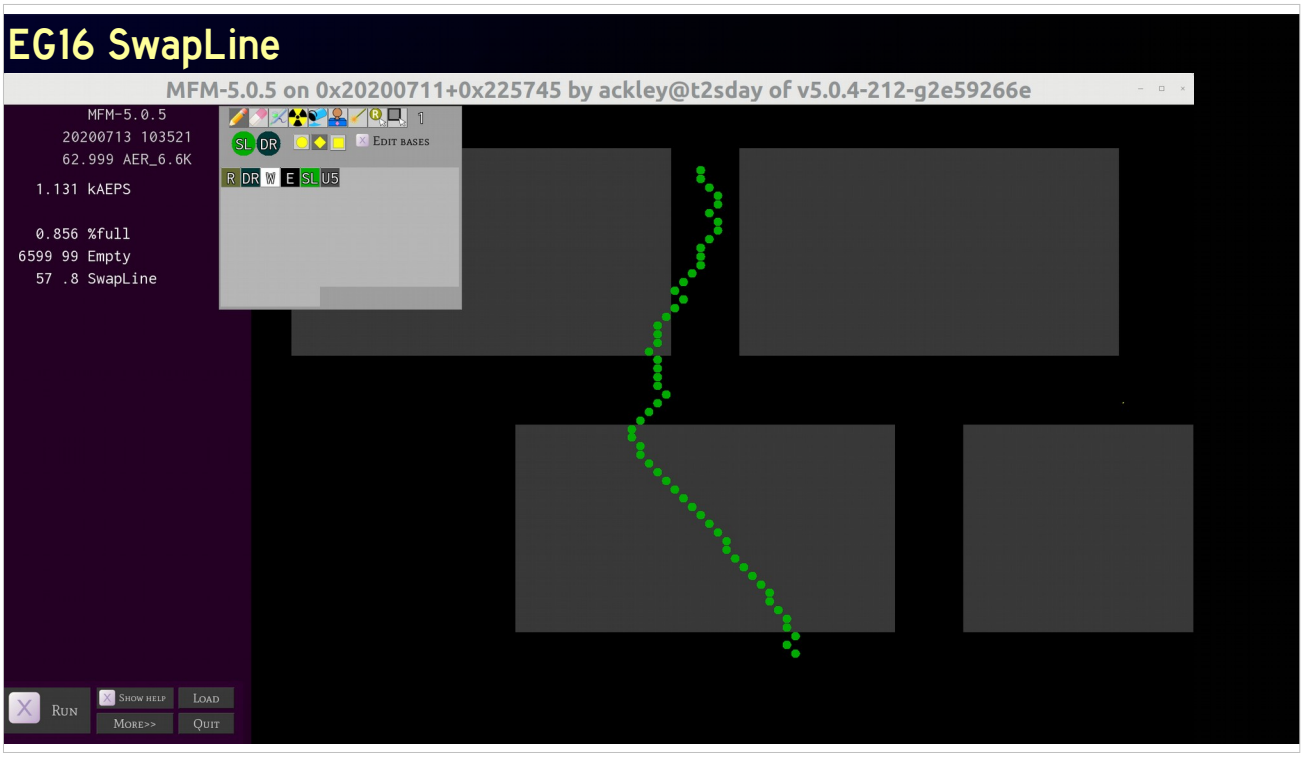
Note that ew.swap refuses to swap off the end of the universe! Sometimes that's what you want; sometimes not.



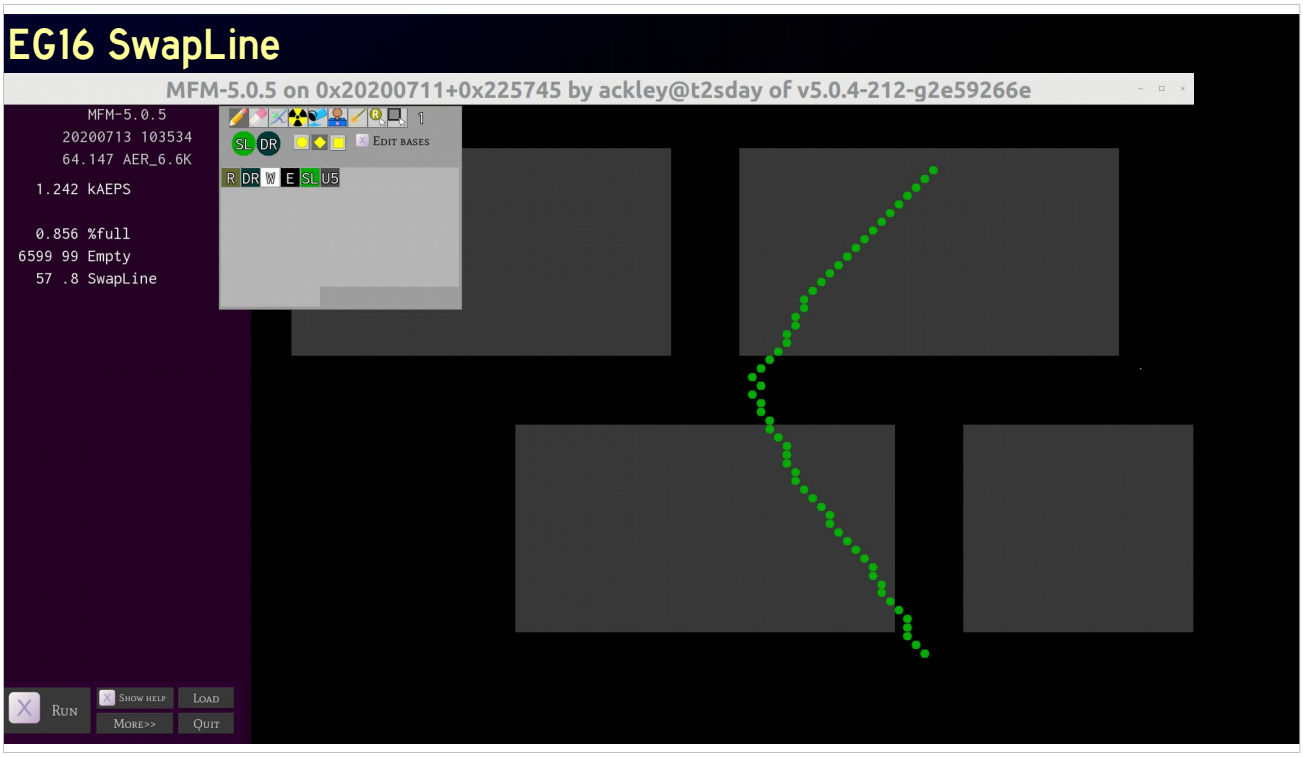
But suppose we clear the grid and then pencil in a whole line



Now the whole line heads east

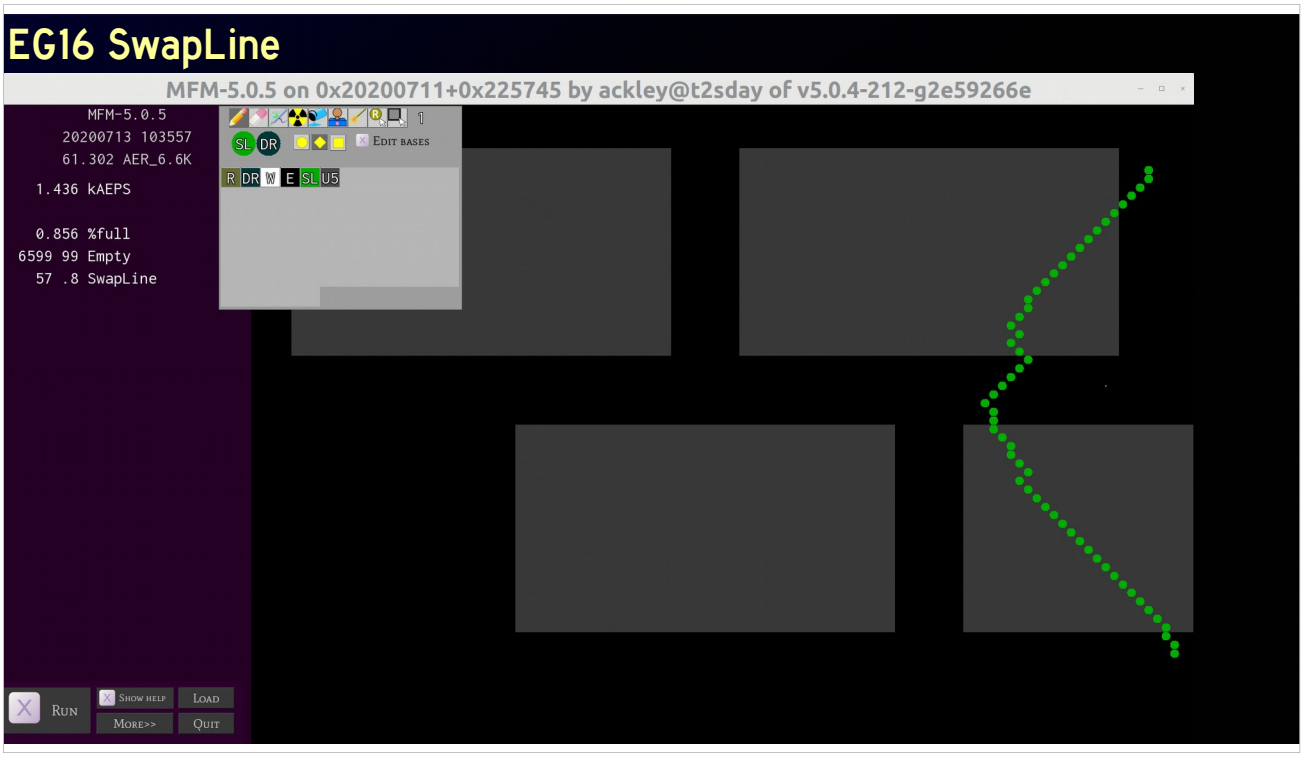


But it’s slope is never more than 45 degrees



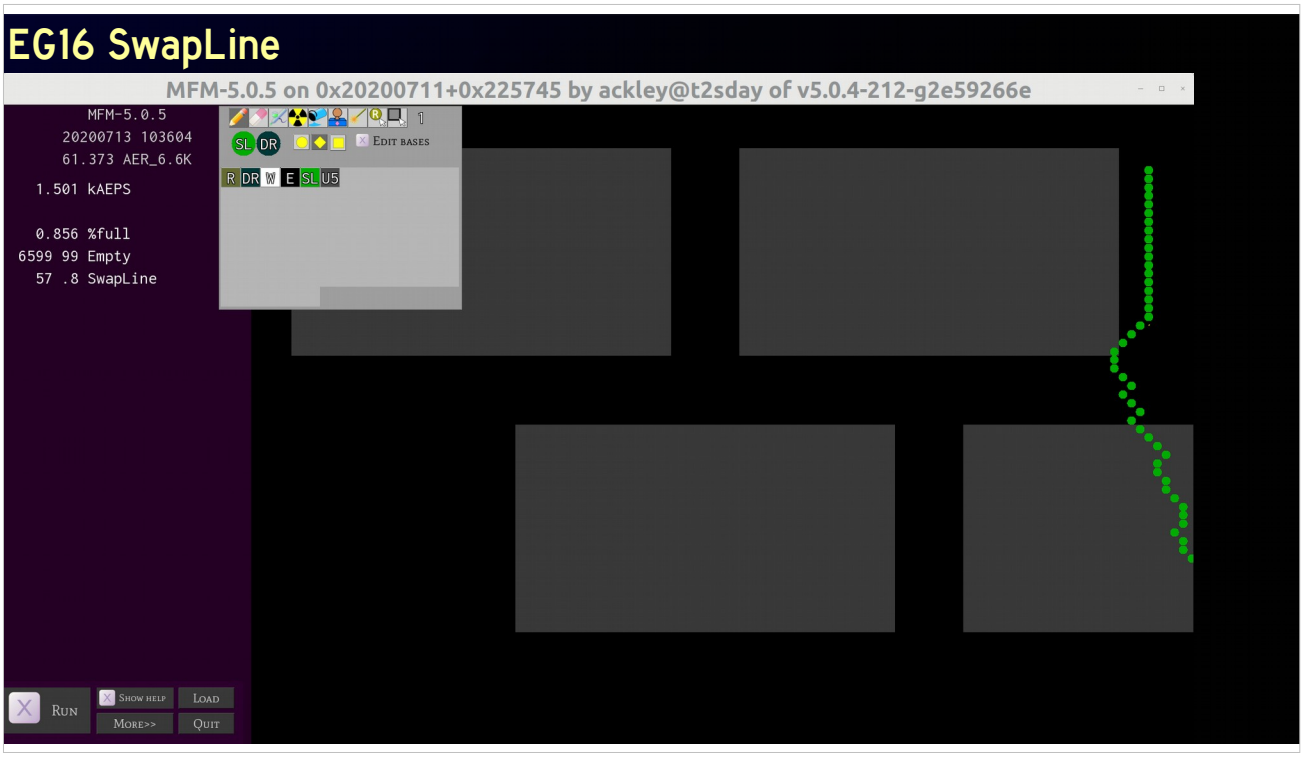
Note how it tends to fall behind in the black areas

Why? Because those are ‘intertile zones’, where performing events requires coordination between multiple tiles.

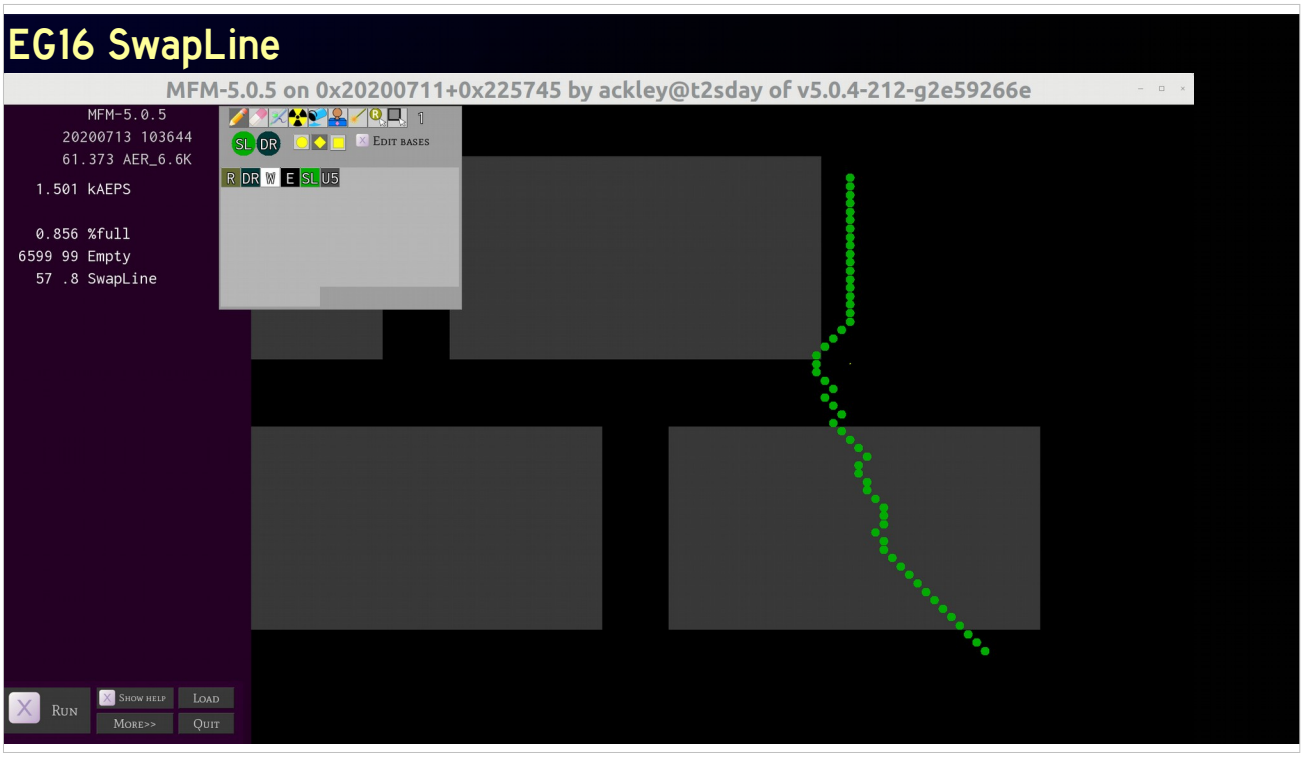


And that slows things down in those areas.

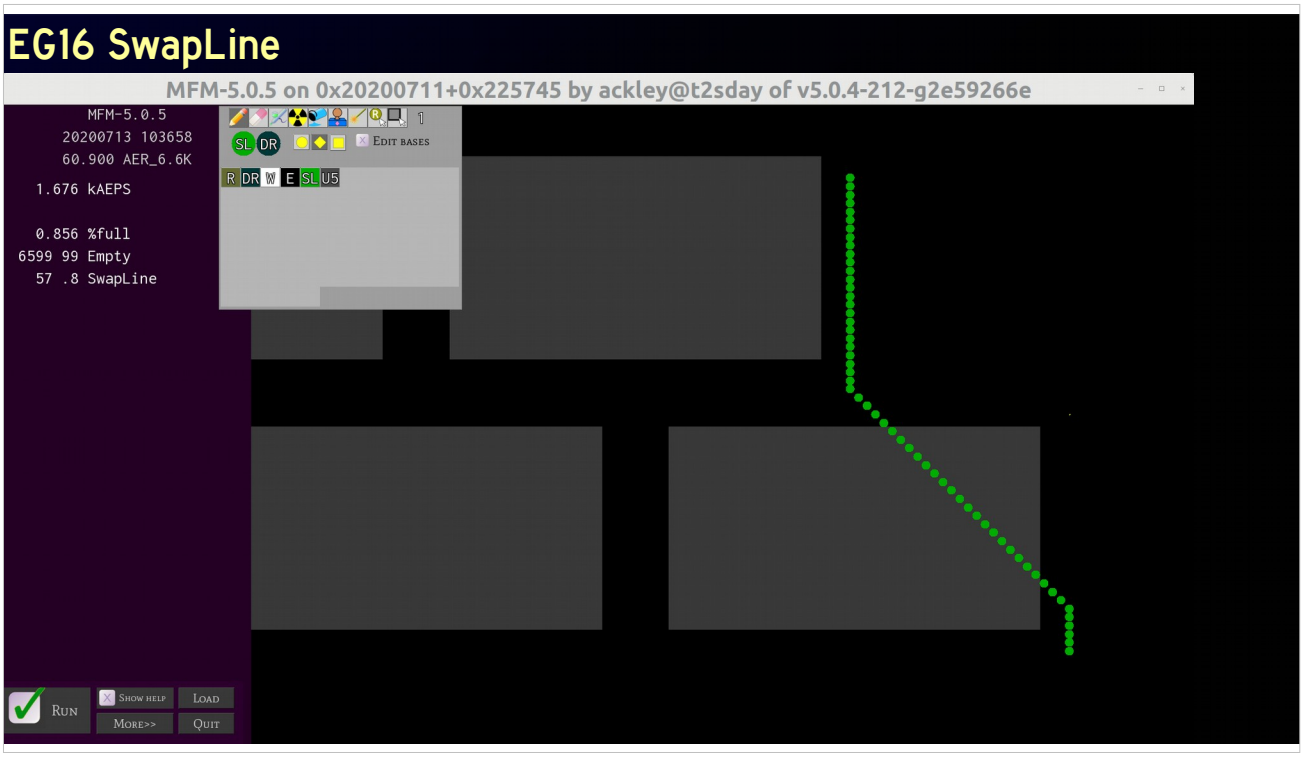
And see what ‘asynchronous cellular automata’ really means! It’s not just a random permutation, or uniform random sampling. There can be HIGH-ORDER spatial biases



Eventually the SwapLine starts to reach the end of the universe



And in this version, stops there.



But note all those behave() methods are still being called! There is no actual halt or exit here!

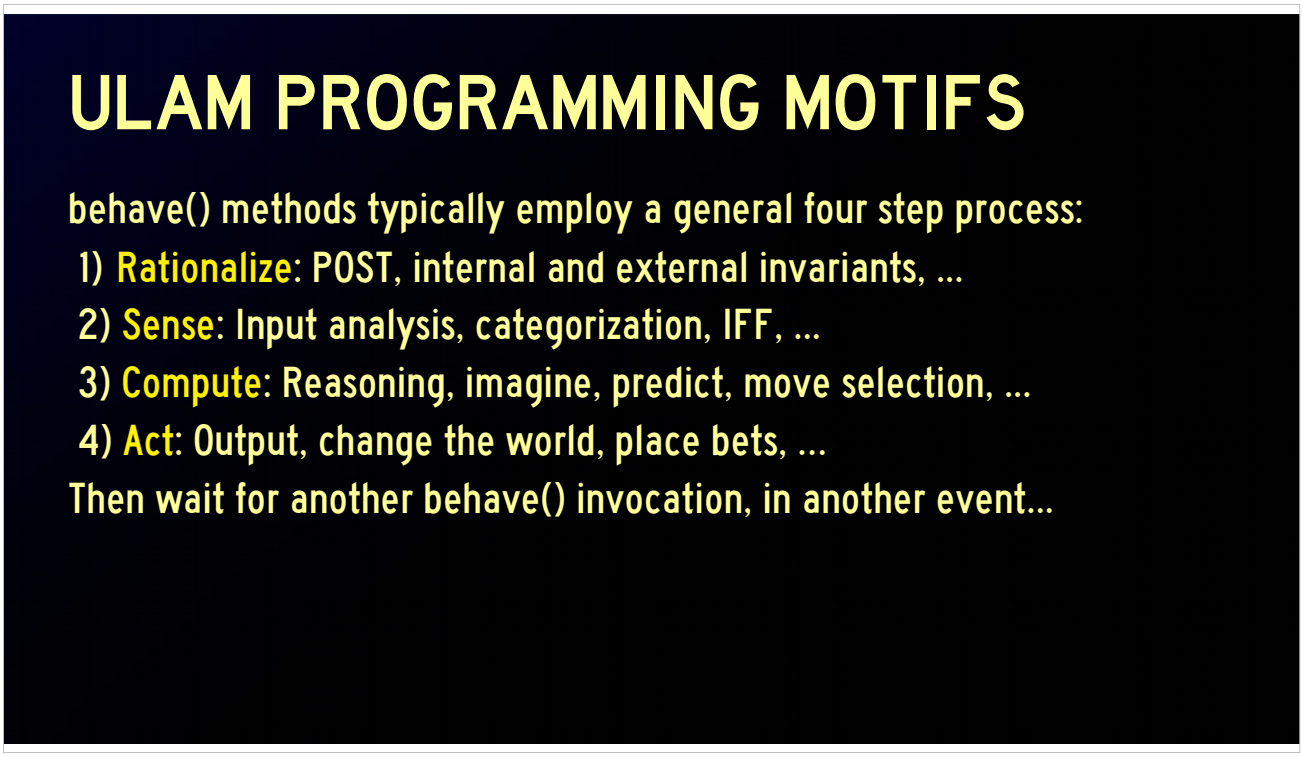
If another tile got connected in this black area here, for example...

ULAM/MFM DISCUSSION

- Isn't this fantastically inefficient? The 'program' keeps restarting!
  - That is a principal source of its robustness.
- How can you write anything serious with only 71 bits of state?
  - View an atom less as a program and more as an ALU or 'mobile FPGA'
  - Many atoms will coordinate to perform complex and stateful tasks
- But how can atoms coordinate without a synchronous architecture?
  - As demonstrated by the SwapLine, atoms can synchronize as needed.
- It's just too hard! Where's my IDE? Where's my autocomplete?
  - Indeed, living on the frontier is not for everybody.

OK, let's wrap up this section

Some Q&A-ish thoughts.



Big picture thought

Traditional CPU/RAM computing is all about complete flexibility about WHAT INSTRUCTION COMES NEXT.

Ulam has that too, but a lot less – just for one event. And even within an event, a single call on behave, there's a typical sequence of phases.

PHASES OF AN MFM EVENT

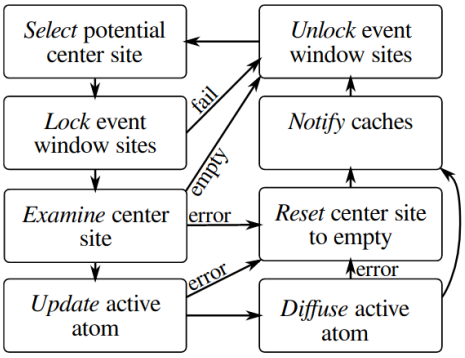


Figure 2: Event window generation. See text for details.

And underneath the hood, down in the MFM architecture, this is the event processing loop.

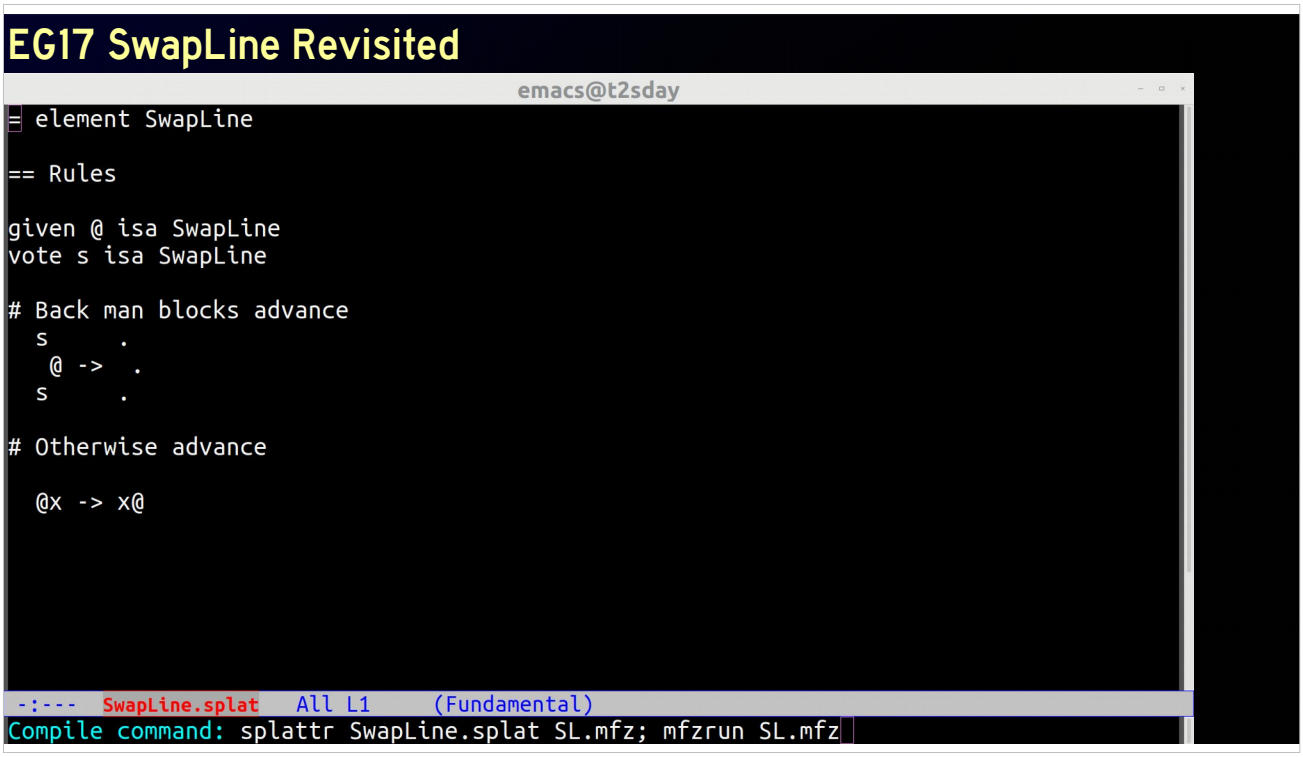
A sequence of steps.

Of course that's true of all sequential computer hardware – at some level it's got to be a looping sequence of steps, or else how is it implemented.

MFM and ulam carry that phased-computation-style up the stack.

OUTLINE

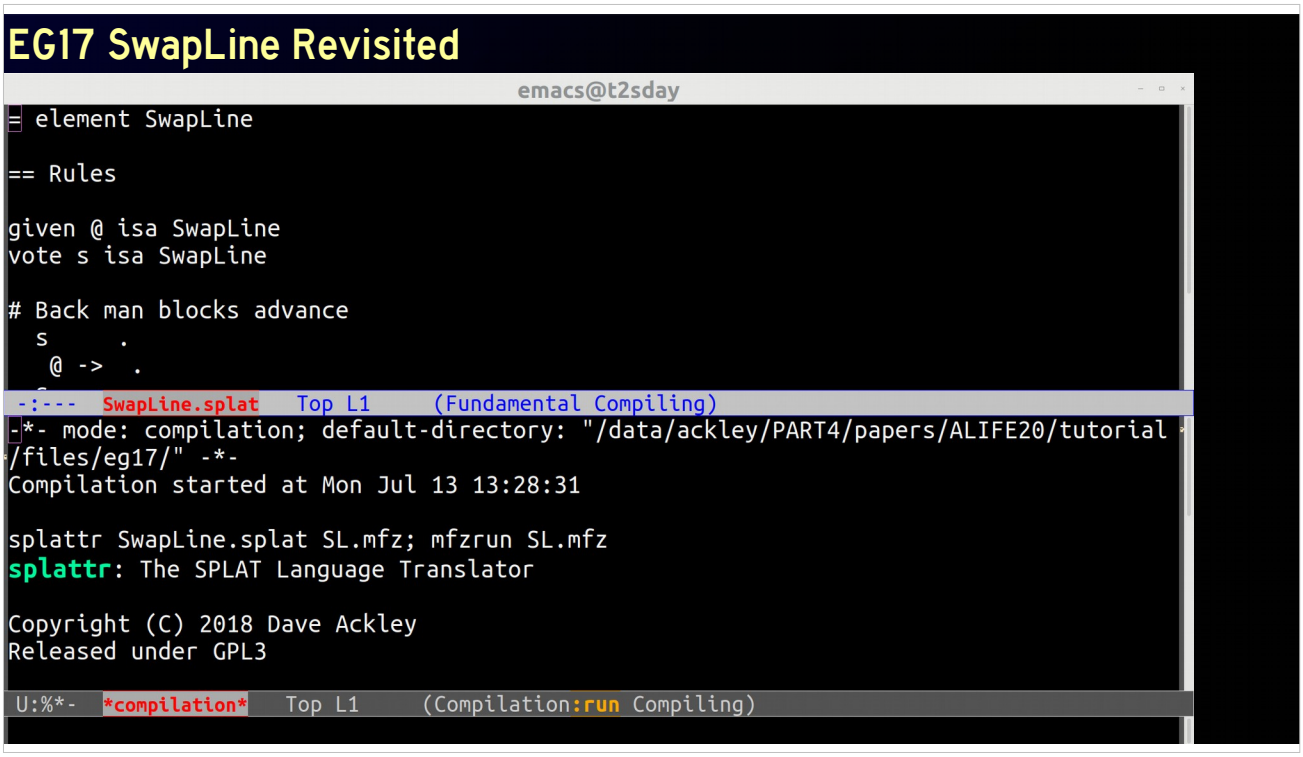
0) Organization: What are we doing here?
1) Best-effort computing: **MFM** & **ulam**
2) Space is the place: SPLAT
3) Biology and Beyond



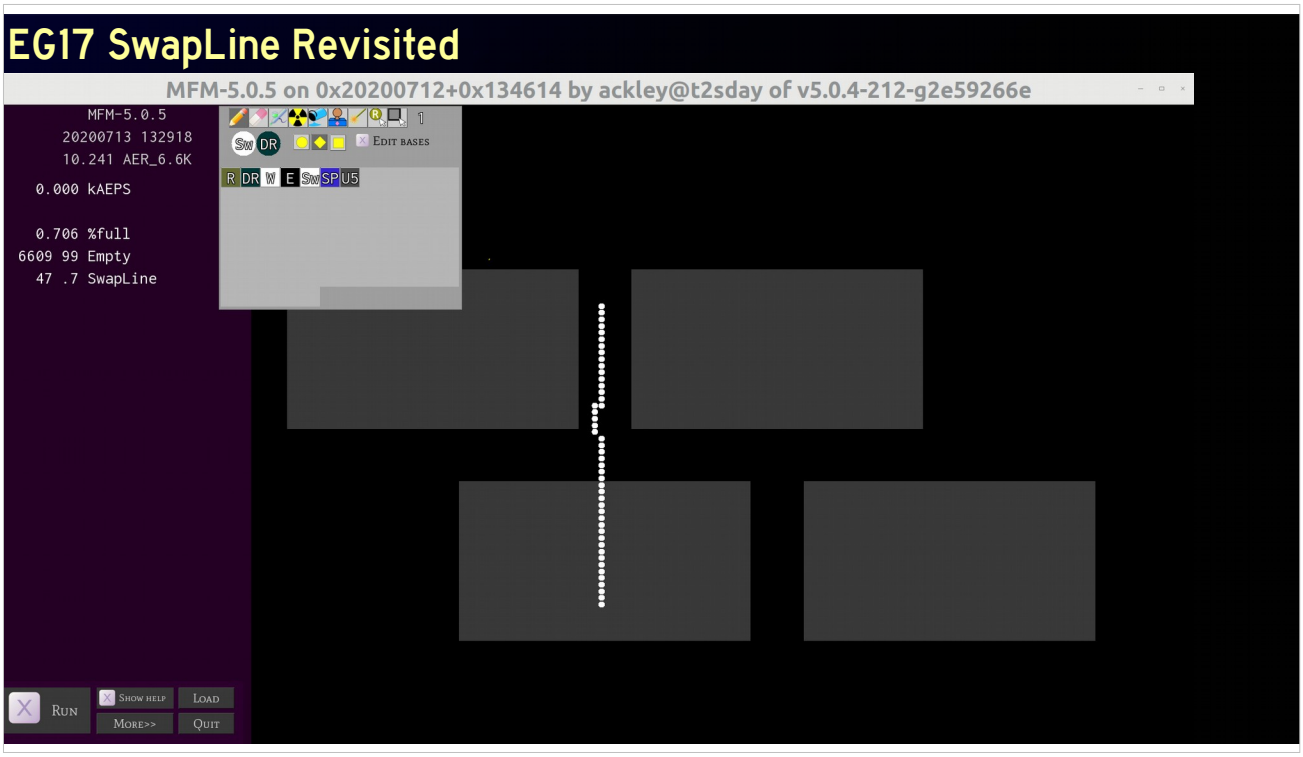
Spatial rules

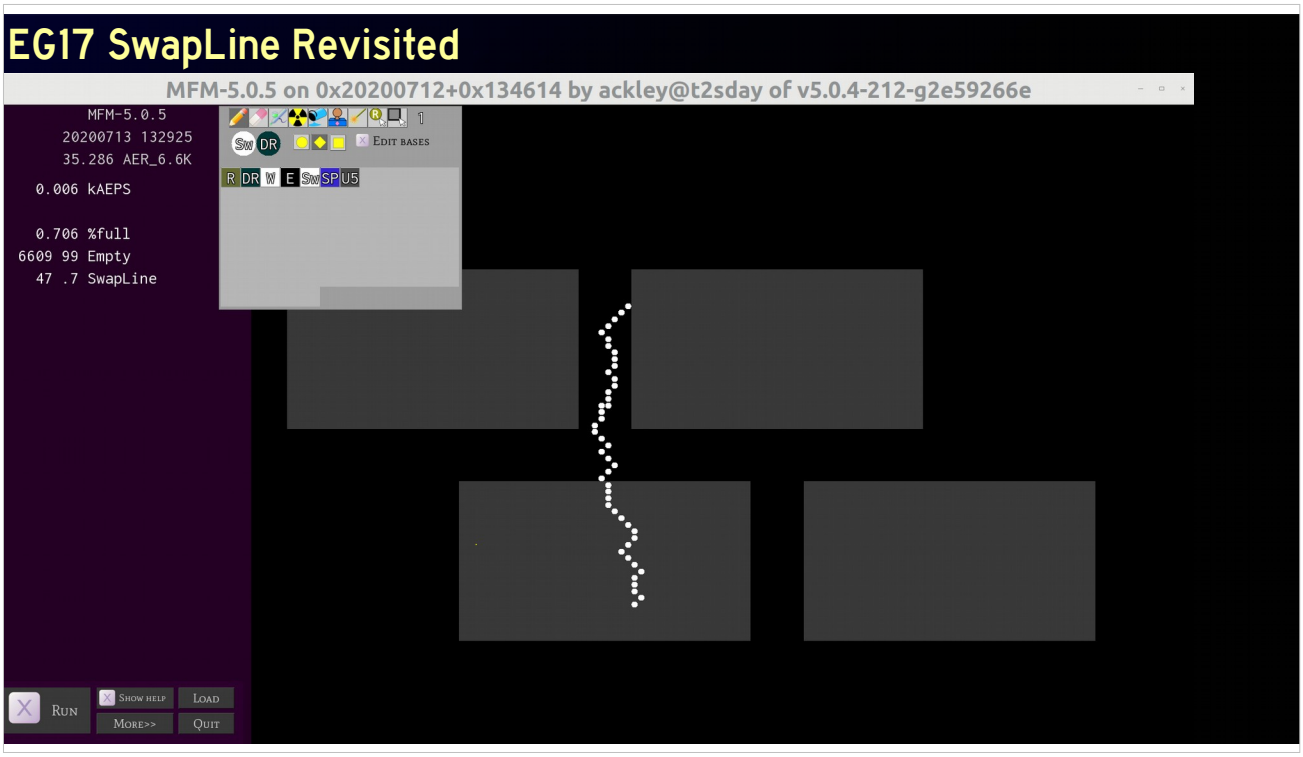
Structure of a rule.

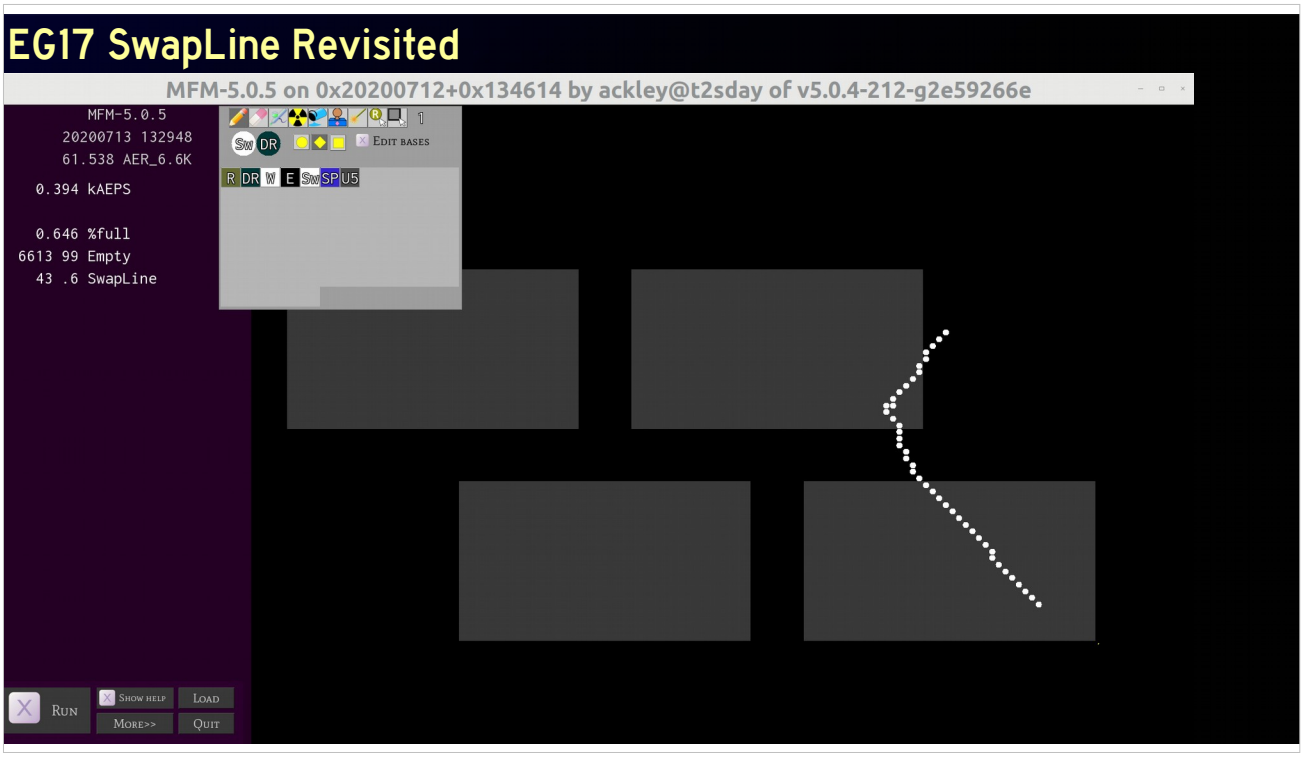
LHS.. check.. RHS..


EG17 SwapLine Revisited
emacs@t2sday
= element SwapLine
== Rules
given @ isa SwapLine
vote s isa SwapLine
Back man blocks advance
s .
@ -> .
-:--- SwapLine.splat Top L1 (Fundamental Compiling)
-*- mode: compilation; default-directory: "/data/ackley/PART4/papers/ALIFE20/tutorial/files/eg17/" -*-
Compilation started at Mon Jul 13 13:28:31
splattr SwapLine.splat SL.mfz; mfzrun SL.mfz
splattr: The SPLAT Language Translator
Copyright (C) 2018 Dave Ackley
Released under GPL3
U:%*- *compilation* Top L1 (Compilation:run Compiling)

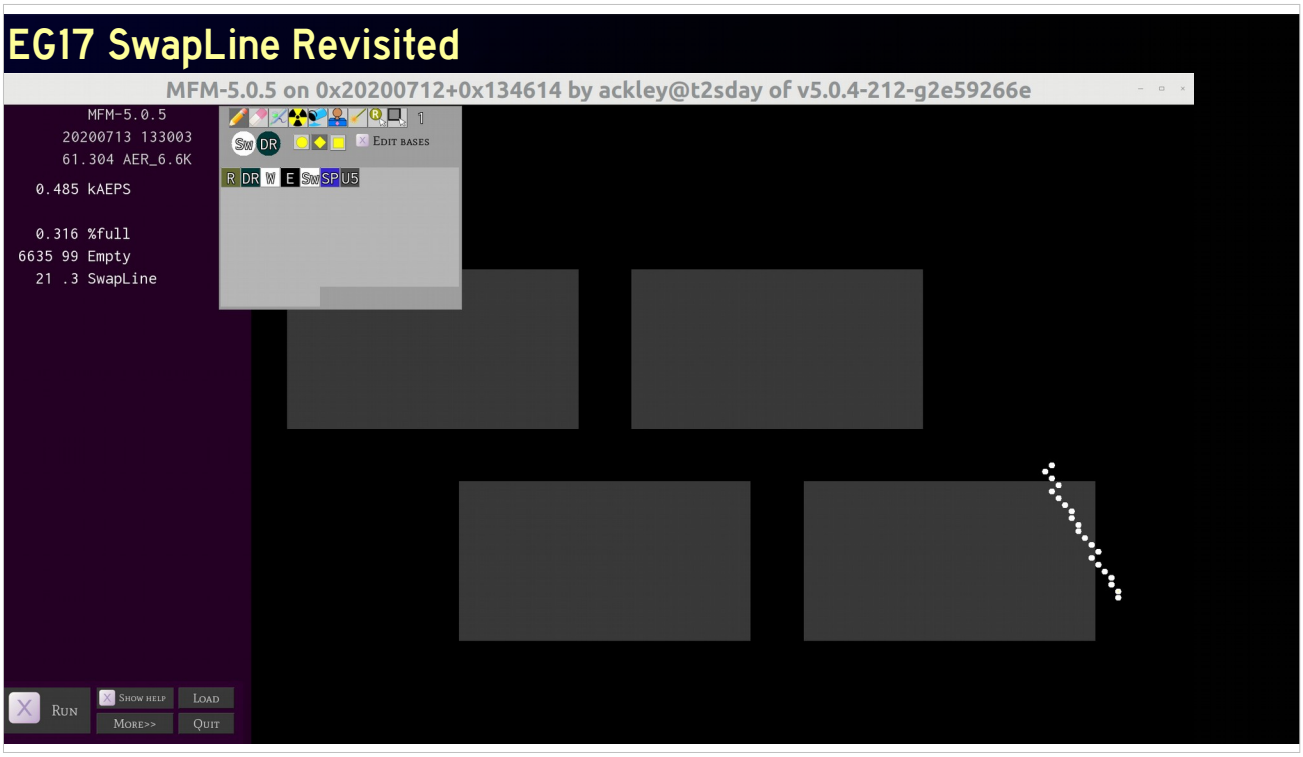
It builds

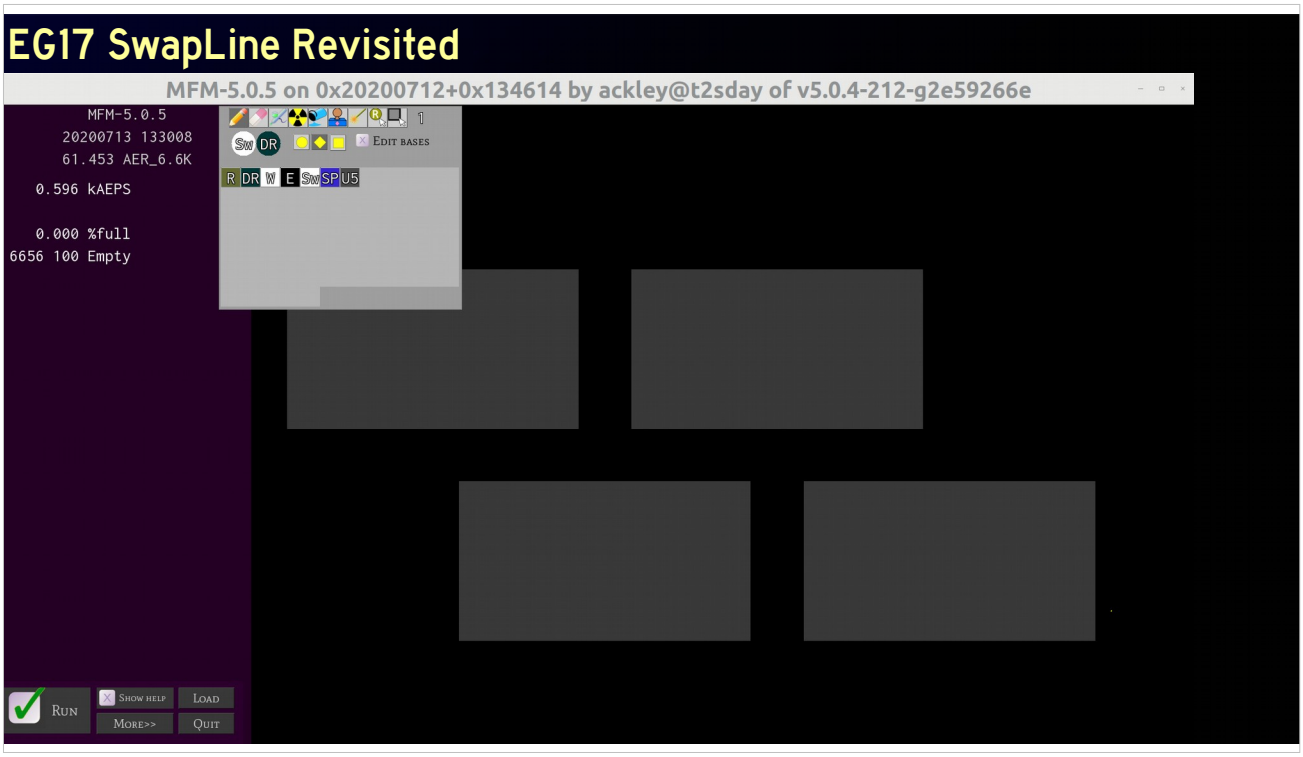

EG17 SwapLine Revisited
MFM-5.0.5 on 0x20200712+0x134614 by ackley@t2sday of v5.0.4-212-g2e59266e
MFM-5.0.5
20200713 132918
10.241 AER_6.6K
0.000 kAEPS
0.706 %full
6609 99 Empty
47 .7 SwapLine
Edit bases
R DR W E Sw SP U5
Run
Show help
Load
More>>
Quit


EG17 SwapLine Revisited
MFM-5.0.5 on 0x20200712+0x134614 by ackley@t2sday of v5.0.4-212-g2e59266e
MFM-5.0.5
20200713 132925
35.286 AER_6.6K
0.006 kAEPS
0.706 %full
6609 99 Empty
47 .7 SwapLine
Edit bases
R DR W E Sw SP U5
Run
Show help
Load
More>>
Quit



But this one runs right off the edge of the universe


EG17 SwapLine Revisited
MFM-5.0.5 on 0x20200712+0x134614 by ackley@t2sday of v5.0.4-212-g2e59266e
MFM-5.0.5
20200713 133003
61.304 AER_6.6K
0.485 kAEPS
0.316 %full
6635 99 Empty
21 .3 SwapLine
Edit bases
R DR W E Sw SP U5
Run
Show help
Load
More>>
Quit


EG17 SwapLine Revisited
MFM-5.0.5 on 0x20200712+0x134614 by ackley@t2sday of v5.0.4-212-g2e59266e
MFM-5.0.5
20200713 133008
61.453 AER_6.6K
0.596 kAEPS
0.000 %full
6656 100 Empty
Edit bases
R DR W E Sw SP U5
Run
Show help
Load
More>>
Quit

ANATOMY OF A SPLAT RULE

- Phases of rule processing
  - Given
  - Vote
  - Check
  - Change
- Escape to ulam

Easy to focus on the 2D rules, but...

OUTLINE

0) Organization: What are we doing here?
1) Best-effort computing: **MFM** & **ulam**
2) Space is the place: SPLAT
3) **Biology and Beyond**

Discuss

CONCLUSIONS

THE PAST

- Traditional computer architecture: Good for safe, small tasks

THE PRESENT

- Computer security failures are the symptom
- ***Hardware determinism*** **is the disease**

THE FUTURE

- Living computation on robust-first architecture

THE LIVING COMPUTATION FOUNDATION

...advances a **unified view** of life and computing.

By supporting science and engineering to explore that connection, and education and outreach to convey it, we seek to improve individual liberty and equal opportunity for the citizens of our emerging information technological societies.

https://livingcomputation.org/support.html

Brought to you by LCFN220