Functional Reactive Python

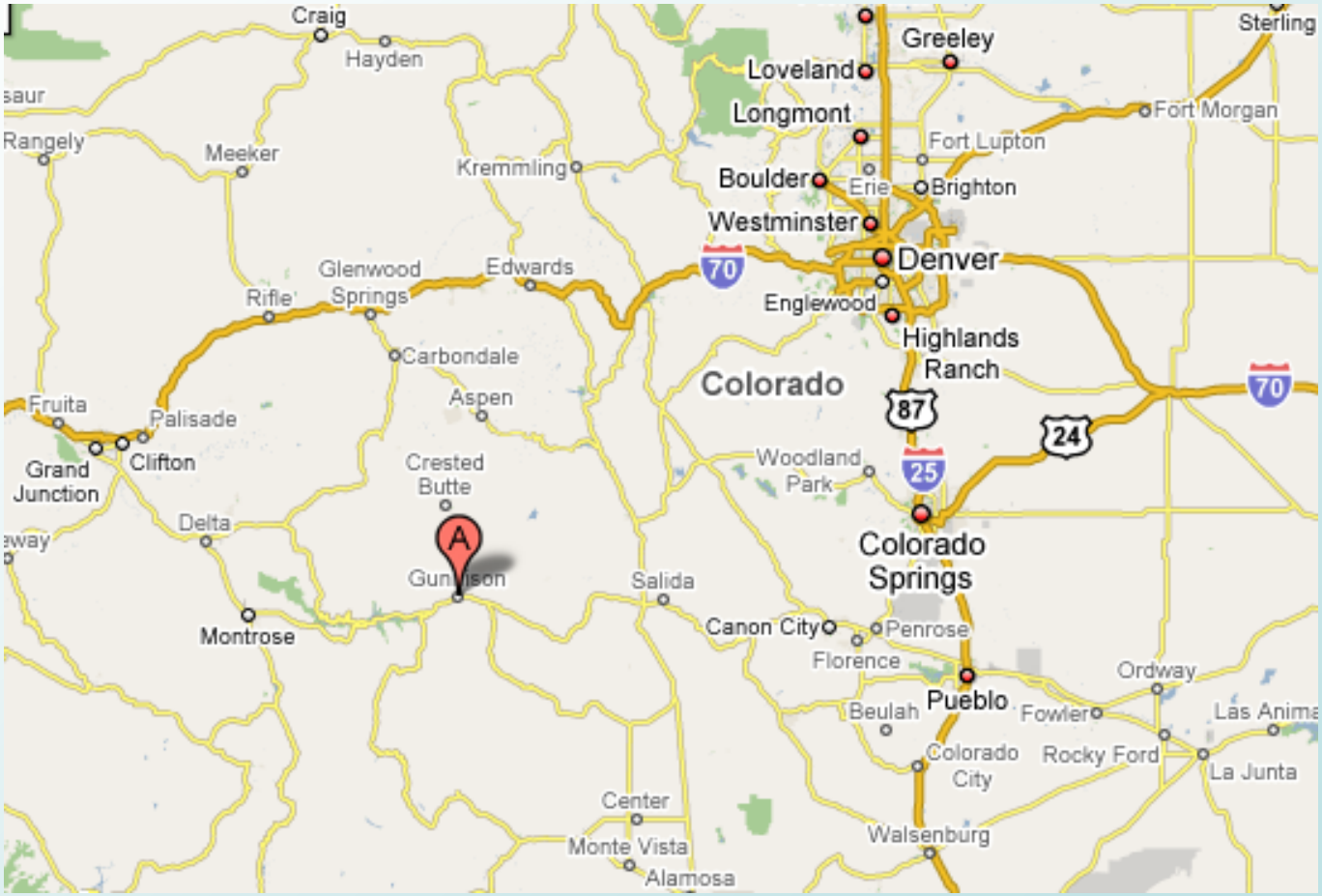
On Introducing CS to High School Students

John Peterson

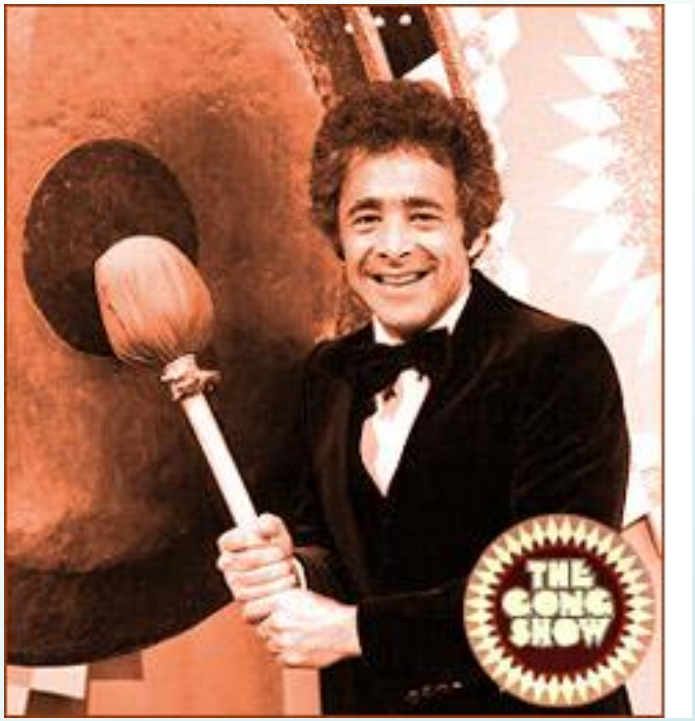
Western State College: Gunnison, CO

jpeterson@western.edu

Western State College?



Danger!



There will be no:

- Type rules
- Proofs
- Significant results
- Greek letters

All programs here are meant to be understandable to arbitrary high school students!

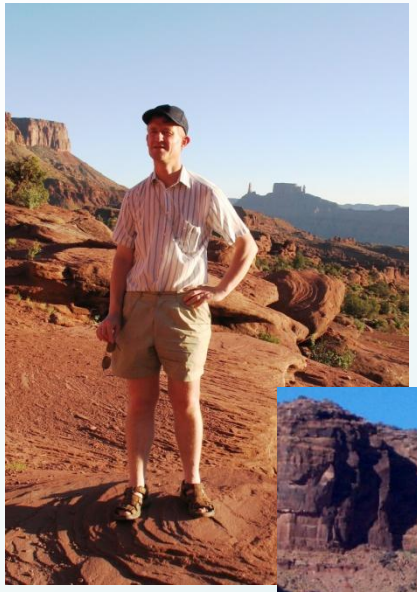
I'll be glad to go faster or get gonged.

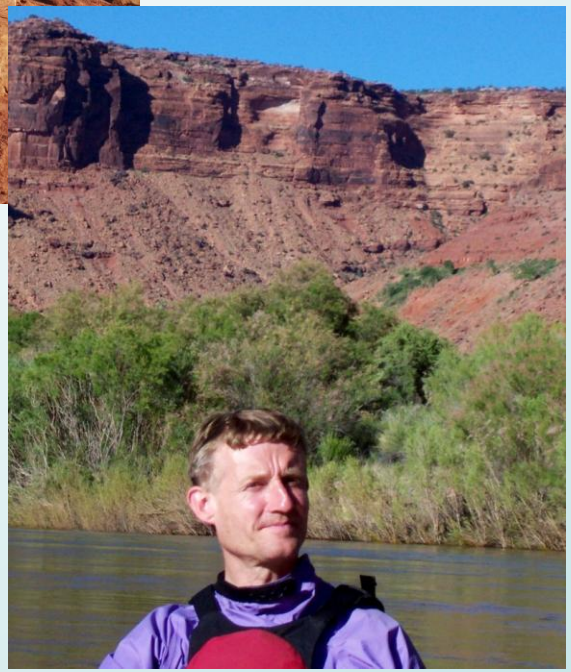
Goals

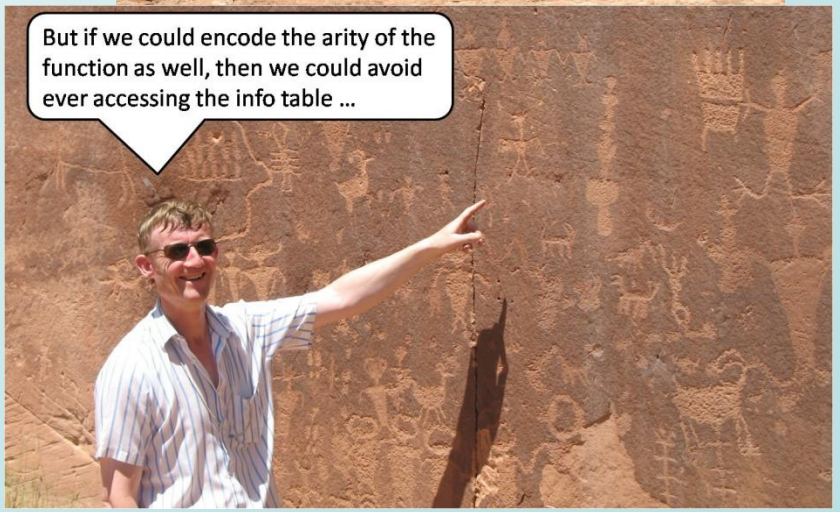
- Get high school students excited about CS with a 1 week summer camp
- Bring them to Gunnison in the summer *(last winter got to -42F)*
- Use virtual worlds as the bait
- Lots of outdoor activity (as in WG2.8!)
- Integrate CS with STEM education

Why Your Kid Should Come!

Recreation + CS: Endorsed by SPJ







Objectives

- Teach basic principles in many subjects – try to go to "first principles" when possible.
- Avoid big chunks of code – no time to teach software engineering
- Lots of small projects rather than one big one
- Don't hide from basic math / physics (STEM)
- Use freely available software

Declarative Language Research

High school students are a great way to evaluate your declarative language ideas!

- No pre-conceived notions about how to program
- Eager to learn
- Will tell you what they think
- STEM integration makes a lot of sense here – an excuse to expose HS students to good CS practices

The Game Engine

This implementation of FRP is built around the Panda 3-D game engine (could also be used with a standard GUI library)

- Good support from CMU
- Runs everywhere (not just Windows)
- Python is an OK first language
- All game engine features we needed were available in Panda
- Python is "functional enough" – with much fiddling I could recreate FRP on top of it.

History of FRP

FRP has been around for a long time now – why hasn't it caught on?

- Incomplete / obsolete / poorly documented implementations
- Not integrated to useful domains
- Lingering implementation issues
- Tends to swallow entire programs – an all or nothing proposition
- Doesn't always simplify things.

The Goal

The big question is whether the examples are easily understood – I've "cut corners" on some semantic issues but I've tried to remain mostly faithful to FRP / Haskell.

Think of assignment as definition and an implicit monad around the non-reactive code.

Models

The objects in our system are "models" used by the game engine

- Models are built on an interesting declarative language (not described here!)
- Separated into geometry and texture (skin)
- Geometry is parameterized – we can adjust joints (bones) within the model
- The model authoring system is Blender
- Models (geometric objects) can be created dynamically
- Issues of default size, position, and orientation
- Nice connection to O-O programming

FRP Implementation Basics

- Two "modes" of code: reactive (signals) and non-reactive (ordinary Python here) (lifted vs nonlifted)
- Signal expressions are "factories" – these are bound to initial times to produce running signals
- Time uniqueness: each signal has just one value at a time step (share single values)
- Signal uniqueness: if you initiate equal signals at the same time you get equal running signals (share entire signals)
- Time leak avoidance: make sure you pull on every active signal at each time step to avoid leaking computation or remember old signal values (see the RT-FRP paper)

Example: Signal Factory

```
# x is a "factory", red code is reactive
x = localTime*localTime
panda = pandaBear()
panda.position = P3(x, 0, 0)
def reset(m, v):
    m.position = P3(x, 0, 0)
panda.react( key('a'), reset)
```

Example: Signal Factory

```
# move right and up, reset up on 'a'
panda = pandaBear()
panda.temp = localTime
panda.position = P3(localTime, 0,
   panda.temp)
def reset(m, v):
   m.temp = localTime
panda.react( key('a'), reset)
```

Example: Shared Signal / Value

```
p = pandaBear()
x = sin(time)
p.position = P3(x, x, x)
```

O-O FRP

We started trying to recreate "classic" FRP in Python – this didn't work well:

- Need to describe the "world" (set of models) – to do this in FRP we would need some very nasty combinators (see "The Yampa Arcade")
- Signal level switching constructs are very awkward
- No notion of object identity inside FRP
- Hard to connect reactive and non-reactive code (code that executes continuously vs code that services an event)

Objects and FRP

- FRP is the basis for purely function GUIs (Antony Courtney & Conal)
- Didn't do a good job of making GUIs easier to express: complex objects with many attributes and a dynamic collection of objects.
- Father Time did a better job but I'm not sure exactly what the semantics are
- Objects are a good abstraction in this domain

Objects and FRP

Basic ideas:

- All signals belong to objects (top level signals belong to "world")
- Each signal is named: Use "object.name" to access signals
- Some signals control the object (position, HPR, …), others are just local variables to the object
- Solve the "running in" problem by allowing different start times in different objects

Objects and FRP

Basic ideas:

- Access signals in non-reactive code using .now() to get current value
- Place reaction at the object level rather than the signal level
- Allow redefinition of signals
- Manage an implicit "World" which encompasses all objects on the screen (hide the collection combinators, use only one collection, unlike Yampa Arcade)

GUI Example

Note the use of object contructors (hangglider, Text, Slider) and named signals.

```
h = hangglider()
Text("Use slider to adjust heading")
s = Slider(max = 2*pi)
Text(s.value)
h.position = P3(sin(time), 3, cos(time))
h.HPR = P3(s.value, 0, 0)
world.cameraPos = P3(0,-5,0)
```

Non-reactive example

```
lim = 3
m = world.mouse()
for x in range(-lim,lim+1):
    for y in range(-lim,lim+1):
        for z in range(-lim,lim+1):
            if (x + y + z) % 2 == 0:
                soccerBall(position=P3(x,y,z), color=red,
                           scale=.1, HPR = P3(time, 0, 0))
            else:
                soccerBall(position=P3(x,y,z), color=blue,
                           scale=.2)
world.cameraPos = P3(4*getX(m), -4, 4*getY(m))
```

FRP Is Still There

```
pos = accum(0,
        key("a", add(1)) +
        key("s",sub(1)) +
        key('d', times(2)) +
        key('f', const(1)))
```

Note that + is event merging.

Sadly we have no sections in Python!

Adding Reactions

```
c = alarm(start = 0, step = 2)
def launch(x):
    pandaBear(
        position = P3(2*localTime-4, 0, 0),
        HPR = P3(time.now()/3, 0, 0))

react(c, launch)
```

Bouncing Balls

```
def addReflect(b):
    b.when(getZ(b.position) < floor, reflectFloor)
    b.when(getX(b.position) > wallRight, reflectRight)
    b.when(getX(b.position) < wallLeft, reflectLeft)

def launch(b, p0, v0):
    setType(b.velocity, P3Type) # For forward reference
    b.velocity = v0 + integral(a)
    b.position = p0 + integral(b.velocity)
```

Bouncing Balls

```
ball = soccerBall()
launch(ball, ballp0, ballv0)
def reflectLeft(b):
    p1 = b.position.now()
    v1 = b.velocity.now()
    p0 = P3(2*wallLeft-p1.x, p1.y, p1.z)
    v0 = P3(-v1.x, v1.y, v1.z)
    print "Reflecting left"
    launch(b, p0, v0)
```

Other Gizmos

We won't talk about these:

- Pose: a record that maps joints to angles
- Script: a mapping from time to events, signals, or poses. Interpolate for intermediate times.
- Interpolator: a way of building scripts on the fly

We use simple file formats to allow students to understand what's under the hood

Signal bundling is very important!

Programming Language Issues

Making time flow implicit is a big win.

Laziness is crucial. We spent way to much time having to re-invent lazy evaluation. Forward reference was a big problem

Python's O-O system kept getting in the way

Python lacks the syntactic flexibility that we really needed

Implicit lifting is a big win – avoids much clutter

Good type systems make things a LOT easier. We did some "load time" type checking but didn't get full H-M. Students didn't understand the debugger.

Multi-Disciplinary Education

- Integrating the math and physics into our curriculum broadens the appeal and sends a message to future CS students
- The creative side of CS is often hidden in CS1 – the game engine gives us a chance to work in a more creative environment
- We were able to cover a wide range of math – not just one specific topic. The visual nature of the system helps a lot.

The "Killer App"

What is it that will get HS students interested in computing?

- Video games?
- Board games?
- Good software that addresses STEM issues?
- ???

The "Killer App"

My observation: music videos!

Creating things that move to music seems to be endlessly fascinating.

Not as good as gaming in terms of teaching programming logic though.

Conclusions

Embedded languages are a pain in the ass – lots of minor frustrations with Python

Static typing would eliminate much frustration

O-O programming is sometimes the answer!

We need to bring good languages to students as early as possible

STEM education is a perfect place to experiment with languages and can motivate language features

The high level language stuff that we do is highly relevant to this task (witness all the bad educational languages!)

The Last Conclusion

A good recreational program will make students forget the frustration of software development.