Functional Reactive Programming in K12 Education

Extended Abstract

John Peterson*
jpeterson@western.edu

Alan Cleary
alan.cleary@westernalum.org

1 Intro

The Western Computer Camp is a summer camp hosted by Western State Colorado University for students in grades 8 through 12 who are interested in mathematics, physics, design, and computer science. Each session lasts one week, during which time the students build 3D virtual worlds as a means of interdisciplinary learning. The worlds are created using the *Panda3D* game engine and a *Functional Reactive Programming* (FRP) Python library. adaptation of the original Fran Haskell library. Students are also able to create their own 3D models to load into their worlds using the *Maya* modeling system.

Panda3D, originally developed by Disney and later adopted by CMU, is an open-source game engine. It is scripted using Python, a language well suited for beginning programmers. Its key features are the ability to load user created models, interactive GUI objects, particle effects, sound effects, and dynamicly generated geometry [4].

Our camp is built around STEM (Science, Technology, Engineering, and Mathematics) topics. We present a wide variety of topics from mathematics (3-D geometry, coordinate systems, vectors, trigonometry, and calculus), physics (projectile motion, object collisions, colors, and control systems), computer science (functions, loops, decisions, and reactive programming), and design (3-D modeling).

Our software fills a gap between systems such as Alice or GameMaker, with fixed interaction vocabularies and only minimal exposure of the underlying mathematics, and game engines programmed through their often complex APIs. Functional Reactive Programming[3]. provides a high level programming environment without the complexity and detail of a full game engine API.

2 Examples

Our Python library is a full implemention of the original FRP language, Fran, in the context of the Panda3D game engine. Aside from the syntactic differences between Haskell and Python, the main difference lies in the treatment of switching, as shown below. We will demonstrate our system through examples taken from the camp. The following code controls the position of an object on the screen using arrow keys. This operates in a manner identical to the classic FRP. We do this by creating a signal for the current velocity that uses the hold function to convert key events into a velocity and then using the `integral` function to turn this into a position. We start with a model, in this case a panda, with a given size and orientation. Note that the python syntax makes this very readable. HPR constructs an orientation from a heading, pitch, and roll.

```
p = panda(hpr = HPR(pi*.5,0,0), size = .4)
```

Now we add a `hold` function that converts discrete events into a continuous signal. In classic FRP, the this function has the following signature:

```
hold :: a -> Event a -> Behavior a
```

Here, the argument to hold is the initial velocity, P3(0, 0, 0), and an event that is an event merge of key events which are mapped onto velocities. The overloaded python `+` is used in place of the FRP `.|.` operator and the event values (velocities) are given in the second argument to key rather than through the classic FRP `-=>` operator.

```
v = hold(P3(0,0,0), key("arrow_up", P3(0,0,1.5))
                  + key("arrow_down",P3(0,0,-1.5))
                  + key("arrow_left", P3(-1.5,0,0))
                  + key("arrow_right", P3(1.5,0,0))
```

Finally, the panda is attached to the controller by setting its position to be the integral of the velocity:

```
p.position = P3(0,-.8,0)+integral(v) # Initial position is 0, -.8, 0
```

Note that some of the trappings of object oriented programming are present: `p.position` refers to the position component of the panda.

The controller can be extended to limit the range of motion by adding synthetic events (the when function in classic FRP). For example, to make the model bounce when the x coordinate exceeds 3, you can add

```
happen(getX(p.position) > 3, P3(-1, 0, 0))
```

to the equation defining the velocity.

For a more complex example, we will present a non-interactive animation that was developed in response to a unit on random numbers. See Figure 1 for a still picture. Students were presented with a lecture on the use of randomness to make interesting animations and then developed scenes that made use of randomness in interesting ways. In the following example, the student decided to animate a volcano shooting variously colored spheres using random numbers to control the trajectory and color

of each sphere. Once a sphere hits the ground it changes to a panda and moves away from the volcano. The program starts with a static volcano model (created in Maya):

```
volcano(size = .8, position = P3(0,0,-1.3))
```

Instead of using the FRP switch combinator, our system uses reaction functions. A reaction function takes two parameters: the model which originates the reaction and the the value of the event that triggers the reaction. In this program, the `spew` function reacts to a timer by launching five spheres. The `launch` function (developed earlier in the camp) sets the position of an object using projectile motion equations given an initial position and velocity:

```
def spew(m, v):
   for i in range(5):
      hpr = HPR(randomRange(0,2*pi), randomRange(-pi/2,-pi/8),0)
      s = sphere(size = randomRange(.05,.2),
                 color = color(randomRange(0,1),randomRange(0,1),randomRange(0,1)))
      launch(s, P3(0,0,-.2), HPRtoP3(hpr)*5)
      p.heading = getH(hpr)
      p.when(getZ(p.position) < -2.5, changeToPanda)
```

The `spew` function uses the python for loop to generate the five spheres. The initial hpr (heading, pitch, roll), size, and color are computed randomly and use to add a new model to the scene. The management of scene objects is implicit in this FRP implementation - each type of object such as a panda or sphere has an associated constructor which adds it to the scene and sets it in motion. In this example, these equations are established by launch. Scene objects can be given user-defined attributes. Here, the `heading` attribute is used to remember the initial direction of travel. It is not a pre-defined attribute like color, size, hpr, or position. The final line creates a reaction function on the sphere: when the Z coordinate of its position drops below -2.5, the changeToPanda reaction is triggered.

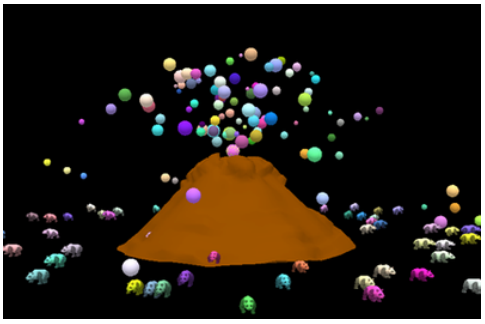

Figure 1: Student volcano

The `changeToPanda` function sends a panda off in the same direction as the sphere was traveling. Note the equation for linear motion. The `HPRtoP3` converts a heading in spherical coordinates into rectangular ones. The duration is used to remove the panda from the scene after 2 seconds. The use of `now` is important - this corresponds to the `snapshot` function in the original Fran system. It takes a static sample of a behavior. Without it, the panda would be tied to the continued motion of the sphere instead of its state at the time the event is triggered. The `exit` function removes the sphere from the scene.

```
def changeToPanda(m, v):
    h = m.heading panda(position = now(m.position) + integral(HPRtoP3(HPR(h, 0, 0))),
                        hpr = HPR(h, 0, 0),
                        duration = 2, color = now(m.color), size = .3)
    exit(m)
```

Finally, this clock causes the volcano to spew every .1 seconds. The `react` function is similar to `when`: it takes an event rather than a reactive boolean.

```
clock = alarm(step = .1) react(clock, spew)
```

Other capabilities of our software include

- Reactive GUI objects such as sliders and buttons
- Collision events generated by the game engine
- Sounds and particle effects
- Camera control
- Import of models designed in Maya
- Control over model texture
- Models with joints

FRP is able to integrate all of these aspects of the game engine in a simple, consistant framework that hides underlying detail and allows our students to use a declarative style of progamming.

3 Experiences

Our camp has been running since 2006, giving us a wide variety of experience using this software. Our software is used in a highly supervised environment, allowing us to use a much more complex system than would be possible in an unsupervised environment. Most students require help in program development and debugging - we can't really compare the usability of our software with commercial efforts like Alice or Gamemaker.

Our emphasis is not on games, but rather, gaining insight into a wide variety of subject matter. Virtual worlds expose students to design and subject elements independent of the programming content. These elements hold the students attention and provoke further creativity. The FRP library has allowed us to put much of the computing content in the backgraound as we explore a wide variety of STEP topics such as coordinate-systems, parametric equations, interpolation, integrals and derivatives, intersections, projectile motion, friction, springs, Newton's laws, and collisions.

This software has an advantage beyond the instruction of students in our camp: undergraduates in our program gain valuable experience developing software and lessons for the camp and assisting the campers. The expressiveness of our system allows them to share advanced material with the campers and help them develop systems that would be much more complex in conventional languages.

4 Related Work

FRP has been used in a number of similar contexts in the Haskell comminity. The original Fran syntax gave way to an arrowized notation[5] which provided semantic leverage but made the reactive code much more complex. Game programming started with the Yampa Arcade ([2]). Currently Reactive Banana[1] is widely used and provides a similar programming experience. More recently HGame3D has been released, integrating reactivity directly into a game engine. Our system has the advantage of ease of use and a more familiar base language. Python features such as named parameters, dot notation for object component access, and dynamic typing make the syntax and programming style more familiar. Our system lacks the static type guarantees of Haskell (which is unfortunate) but shields students from arrows, do notation, type signatures, and other high level features used in FRP libraries. We have designed our system with static typing in mind and it should be possible to add static typing at some point.

5 Conclusions and Future Work

We believe that FRP and Python combine to provide a novel and useful basis for programming in the service of general STEM education. Our students have built many small, creative projects which draw on basic STEM material. We have also used this software in an introductory computer science course and are currently adapting it for an interdisciplinary course in kinetic art, using FRP to control the mechanics of kinentic sculptures.

References

[1] Reactive banana. Website, 2013. http://www.haskell.org/haskellwiki/Reactive-banana.

[2] Henrik Nilsson Antony Courtney and John Peterson. The yampa arcade. *Proceedings of the 2003 ACM SIGPLAN Haskell Workshop*, 2003.

[3] Conal Elliott and Paul Hudak. Functional reactive animation. *International Conference on Functional Programming*, 1997.

[4] Mike Goslin and Mark R. Mine. The panda 3d graphics engine. *Computer*, 37:112–114, Oct 2004.

[5] Henrik Nilsson Paul Hudak, Antony Courtney and John Peterson. Arrows, robots, and functional reactive programming. *Summer School on Advanced Functional Programming*, 2003.