PhysEx

A scripting language for fast prototyping and minimal effort physics engine development.

Joshua Nuez (jn2548)
David Pu (sp3396)
Steven Ulahannan (su2206)
Justin Pugliese (jp3571)

September 2016

Introduction:

A physics engine is a useful simulation tool to observe behaviors of physical systems. They allow observation of patterns which can assist in making future predictions. Unfortunately, traditional languages are not optimized for these types of applications. PhysEx is designed for fast prototyping and reduced complexity of applications which utilize a physics based environment.

PhysEx natively supports environment and object simulation, and can incorporate libraries that support different branches of physics such as kinematics, thermodynamics, and electromagnetism. Native objects are able to calculate and update their state and position at continuous time intervals. Objects are also independent of most variables, but can be manipulated through functions, such as Predictor and Simulator.

Language Design:

The two main components of a physics engine are the environment and the objects, called blobs, which exist within the environment. We call them blobs because we would like to distinguish it from objects, since we try to stray away from object-oriented programming and we don't intend to have hierarchical structures of objects. In our language, we translate the two components into two layers of abstractions. Environment describes the interactions (physical forces) among the blobs and each blob contains all the parameters sufficient to describe its state, such as mass, energy, position, velocity, temperature, etc.

In general, most applications will begin with the creation of an environment and its attributes. Once an environment is defined, objects can be added to it and their behavior can be observed and further manipulated. For example, if an environment similar to Earth was being created a gravity attribute could be defined. Then objects, such as an apple, could be created with its own attributes like mass or vertical position. As the simulation progresses the environment will exert forces on the contained objects at specific time intervals, which will update the state of the objects. The simulator will display the current state of the objects and environment on the terminal.

Data Types:

The native data types which create the foundation of PhysEx are:

char string int bool float array	Similar to C/C++
environment	Defines the conditions that containing blobs will be influenced by
blob	Set of attributes which define a physical body that lives within an environment

Variables do not have to be defined during declaration, instead the compiler will detect type during runtime.

Arithmetic Operators:

Integer Operators	= + - * / ** %
Float Point Operators	+. -. *. /. **.

Similar to comparators, arithmetic operations can only be done on objects of the same data types.

Logic Operators:

PhysEx supports most of the same comparators as many other languages and it ensures that data types are consistent between the objects being compared.

Comparators	== != < > <= >=
Boolean Operators	|| && not

Control Flow:

	Syntax

if, else statement	if *condition* {
	  ...
	} else if *condition* {
	  ...
	} else {
	  ...
	}

while loop	while *condition* {
	  ...
	}

for loop	for (*counter; condition; increment*) {
	  ...
	}

Native Functions:

Simulator	Monitors and manipulates the environment and objects which make up an experiment
Predictor	Takes in (object, length of time, helper function) as parameters and returns a new object with a different state. The helper function would come from libraries that are appropriate for the calculation you want. For example, if you want make predictions about a free fall object, then import the kinematics library and pass in the free fall equation function as a helper function; for thermo, pass in thermo equations as helper functions.

Roles:

Role	**Name**
Manager	Joshua Nuez
Language Guru	Justin Pugliese
System Architect	David Pu

Tester	Steven Ulahannan

Sample Program:

Empty sample program that creates a vacuum which is void of objects.

```
let env = Env.default; // Default: Vacuum
let blobs = []; // Default: Empty array of blobs
let time = 10; // length of simulation in seconds
let delta = 1; // Incremental change of time in the Simulation

Simulator(env, blobs);
Simulator.start(time);
```

Gravity (Object in Free Fall on Planet Earth)

```
import printl from print;

function earth(blobs) {
      let g = -9.81;
      let deltaT = Env.deltaT;
      for(let i = 0; i < blobs.length; i++) {
            let newVelocity = blobs[i].v +. g *. deltaT;
            let velocity = (newVelocity +. blobs[i].v) /. 2;
            blobs[i].v = newVelocity;
            blobs[i].y = blobs[i].y +. velocity *. deltaT +. 0.5 *. g *.
deltaT **. 2;
      }
}

let blob apple = {
      mass: 15,
      y: 0,
      v: 0
};

let env = Env(earth);
let fruits = [apple];
let time = 10;
let delta = 0.1;

Simulator(env, fruits, delta);
Simulator.start(time);
```

```
println(blobs);
```