The Joys of Writing Code in Only One Line

Hugo Robinson

Contents

Analysis 1

Background 1

Smaller Than Indivisible? 1

You Must Be Joking Mr Feynman 1

Description of project 2

Inside a Simulation 2

The Class System 2

Why Functional? 1

You Must be Joking Mr McCarthy 1

Challenges - Imagine Programing Without, Variables, Loops, Selection or Sequence 1

What is the Point? 1

Approach 2

Specific objectives 2

Problem analysis 3

Design 4

Overall design approach 4

Plan for final development structure 4

Like an Onion 4

Why C# and not Haskell 5

Specific problems and their solutions 5

Heads and tails 5

How to Randomise a Determined Value 5

Generic Parameters 5

One of These is Not Like the Others 5

Areas involving technical complexity 5

Technical implementation 6

Testing 7

Evaluation 8

# Analysis

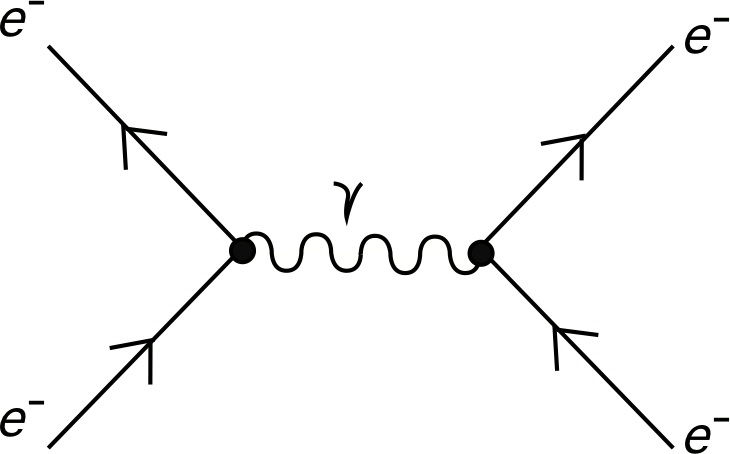
## Background

### Smaller Than Indivisible?

Particle physics, as we know it today was first discovered between 1879 and 1897 when the electron was discovered. This meant that the atom was not indivisible but was comprised of many smaller particles. The discovery led Joseph Thompson to release his Plum Pudding representation of an atom where a dense cloud of positive charge was interlaced with electrons. However as we know today, this model is not correct, the new model with a nucleus of positive charge was proposed Ernest Rutherford in 1919 after his experiment which involved firing alpha particles at a very thin sheet of gold foil. Most of the particles passed through the foil as expected however some diffracted and even fewer bounced back. This was so astonishing at the time that Ernest Rutherford said, “It was quite the most incredible event that has ever happened to me in my life. It was almost as incredible as if you fired a 15-inch shell at a piece of tissue paper and it came back and hit you” <https://www.goodreads.com/quotes/105980-it-was-quite-the-most-incredible-event-that-has-ever>. Then in 1931 James Chadwick discovers the neutron, which gives the final piece of the puzzle for the creation of the atom and how the standard model of the atom is represented today. As a result of these studies the four fundamental forces of nature were discovered, The Strong Force, The Weak Force, Electromagnetism and Gravity. The Strong force mediates how the nucleons stay together and acts over a range of femptometers. The weak force mediates the decay of particles. The electromagnetic force mediates how different charges and magnetisms interact with each other, and is the reason matter can clump together to form galaxies, stars and planets.

### You Must Be Joking Mr Feynman

Along with this, throughout the last century, different interactions between particles have been observed, both natural and created in colliders. These interactions govern everything from the levitation of superconductors <http://www.superconductors.org/> to the entire field of Chemistry. Many of these interactions can be represented through Feynman diagrams.



This Feynman Diagram shows a basic electrostatic repulsion where two electrons that poses the same charge come too close together. Then an interaction particle, in this case a photon, causes the two electrons to move away from each other. This diagram is also represented on a graph where the x-axis is space and the y-axis is time.

2 Paragraphs

Nothing to do with the project, just motion the background about particle physics

## Description of project

### Inside a Simulation

This project is designed to simulate particle interactions in multiple scenarios where up to two particles are entered by the user, along with some requirements such as velocity. The program will then use this information to run real calculations to determine the velocity and position of new particles created. This information will then be presented to the user.

### The Class System

The classification of particles allows for me to easily implement a class hierarchy system, which will form the base for every interaction in the project, as these classes will represent all of the particles with their attributes being stored as properties. These classes can then be created at individual instances multiple times and still retain their basic properties such as charge and rest mass. However their position and velocity can still be changed. A basic version of the particle hierarchy can be seen below.

Where a represents an abstract class and a represents a static class.

Not FP in this section

Feynman Diagrams?

Particle Hierarchy

1 paragraph

To create a particle interaction simulator

Particle

Lepton

Quark

Boson

Electron

Up

Photon

Meson

Hadron

Proton

Pion

## Why Functional?

[If not already is your opportunity to persuade the examiner that this really is an A-level standard project. You might not need this section]

### You Must be Joking Mr McCarthy

In 1958, John McCarthy was leading a team at MIT to ‘create a system for programming computations over symbolic data’ <https://www.cs.kent.ac.uk/people/staff/dat/tfp12/tfp12.pdf> - Chapter 2 Lisp - Page 5.

Some history on functional

Lambda

Lisp

Use book?

https://www.cs.kent.ac.uk/people/staff/dat/tfp12/tfp12.pdf

### Challenges - Imagine Programing Without, Variables, Loops, Selection or Sequence

There are some very serious challenges that arise when trying to create a program using functional techniques. One is the lack of variables used. Right from the beginning of being taught to program variables have been used, however they are not allowed in functional as variables can change. Whereas in functional all types have to be immutable. Therefore if any data structure needs to be changed, a new data structure has to be created with exactly the same entities as the previous one and whatever change has taken place also applied. The lack of sequence also makes functional difficult as everything has to be done using only one line within a function, this reduces some of the clarity when coding and requires simple sequential statements to either be moved into one line, or extracted into separate functions. There is also no selection in functional, therefore ternary operators must be used on the return statement instead. Any fort of loop is also not allowed in functional programing; therefore recursion must be used throughout.

### What is the Point?

The Functional Paradigm is a reasonably old way of thinking however with todays requirements of big data it has become much more prevalent. It is used heavily in science as functional techniques due to its reliability when headlining large volumes of data; it also can be very easily tested to make sure the results that are output are correct and there are no logic errors. Having learnt some basics of functional programing through the course I decided I wanted to peruse it further, extending myself past the trivial problems introduced to us. Therefore with my interest in the topic and its evident use in science it was perfect for me to develop me project around.

All of the problems with the functional paradigm

Need to reference Books in this section

## Approach

External stakeholders -Business approach

My approach is to present the idea of what I am trying to do to a physics teacher who will become the stakeholder in my project. I will then meet with them to gain ideas for added interactions to my project. This approach will allow me to develop my program in line with the requirements of the stakeholder. The requirements of the stakeholder will be reasonably restricted by the functional nature of the program.

[*If* you have a specific stakeholder, include notes from discussions here]

## Specific objectives

Your solution should meet *most* of your objectives, but it is OK to list *some* objectives that are not met

MoSCoW Analysis

Must have

My main objective for this project was to implement the core if the program using functional programing techniques.

The program should allow a user to input what interactions they would like to simulate and the program will the return the list of particles created and some necessary properties of those particles (eg velocity)

The stakeholder has provided some requirements for simulations he would like to see and these need to be met

Should have

Full list of elements that can be culled upon when an element is created

Full headachy of different sub atomic particles.

Could have

To create long decay chains using the decay graph for heavy, unstable elements such as uranium until they become stable.

Decides if an element created is stable or unstable and then gives the option for the element to undergo its decay process.

Won’t have

Bubble chamber GUI due to functional

Large Hadron Collider GUI due to functional + processing

Functional programing (Reference functional paradigms)

Information on how particle interactions work

Meet requirements of a stakeholder

## Problem analysis

This is a description of *what* needs to be implemented, not *how*]

As the program is physics based there is quite a lot of maths involved, which if implemented incorrectly could cause large logic errors.

Particle interactions are usually very set in how they will turn out and therefore some parts of the program will seem like the information will have been looked up instead of calculated.

To simulate modern, high energy collisions such as the ones taking place in the Large Hadron Collider where particles like the Higgs Boson are created are far to complicated for both myself and my computer to do, as they require the best equipment at CREN to attempt to simulate a collision like that.

The position of each particle after collision will have to be calculated alongside wits velocity, this will be done using a particles rest mass and velocity

Physics based

How particle interactions work

# Design

## Overall design approach

The original plan – GUI + Functional mideset

### Plan for final development structure

Layer 1: Particle

* Where the particle diagram is used and that relation is coded into the system. All particles are also provided with a full array propitiates and methods they may require (eg Mass to Energy)
* This is the first layer to be implemented as it will allow the particles to be used throughout the development of the rest of the program with ease
* Uses separate program to the collision, vectors and UI programs
* Start with the Quarks and then move onto the bigger particles

Layer 2: Collision

* Where all the calculations for the collisions takes place and outputs what particles are created
* Will need different collisions for different types (eg Electron capture, annihilation, Proton-proton)
* Should not require any of the vectors layer in coding

Layer 3: Vector

* Where the vector calculations and positioning takes place
* Calls upon both the particles and collisions layer to determine weather any new collisions take place after the primary collision

Layer 4: UI

* The terminal screen UI should be developed alongside the rest of the program but should not be integrated into it to allow a possible development of a graphical UI at a later date, however this step will not be included in this project.
* This section will not be following the functional paradigm as it is very challenging to develop a UI as the point of functional programing is not to generate side-effects, however a UI is a side effect. Add quote?

This plan keeps allows for a higher layer to call on a lower layer but not the other way, this is to make sure that further down the development of the program, if a layer needs to be radically changed it should be easier as it is not being referenced by every other part of the program, only the layers above it.

### Like an Onion

Alongside keeping the layers separate for management it also integrates very well into the functional mind-set. With functions in each layer calling functions a layer below it, meaning that a pure function in one layer will always stay pure as it cannot call a dirty function, which will only reside in the highest layer (UI). A pseudo-code example of how this function call would look would be;

*UI(Vector(Collision(Particle(Arguments))))*

Where the arguments are only the values required in the Particle’s constructor, these values would be obtained through the UI layer form user inputs.

*Variable Arguments => User Input*

*Variable CollisionOutputs = > UI(Vector(Collision(Particle(Arguments))))*

*Terminal Output => CollisionOutputs*

Note that Arguments is stored in a variable, this is not allowed in functional and this phrase involves sequence which is also not allowed in functional, however both are necessary for a UI.

Other layers may also require other arguments such as a random number.

*UI(Vector(Collision(Particle(Arguments),RandomNum)))*

### Why C# and not Haskell

## Specific problems and their solutions

### Heads and tails

F list

Using functional library from NuGet

### How to Randomise a Determined Value

FRandom

### Generic Parameters

One major issue that can occur is how to passes functions different permeates at run time and not hardcode them in beforehand. The answer is generics, this allows the program to infer the type of the perimeter at run time meaning that multiple functions that have the same implementation but different perimeter types can be reduced into a single function with generics.

Add example?

Generics

### One of these is Not Like the Others

The problem of the photon has troubled physicist for centuries so it will come as no surprise that it will also cause problems for this project. Due to the photon having no mass, unlike everything else, a constant velocity, unlike everything else and is both a wave and a particle at the same time will make it difficult to incorporate their calculations into functions that’s implementations are designed for all other particles.

There are two solution to this problem, changing all other particles to work using their De Broglie wavelength to match the wave like nature of light or to use overloaded functions. Overloaded functions are functions with the same implementation; however take different perimeters. This means upon runtime if a function that is overloaded is passed a photon it will use the function designed for it and return the appropriate answer. A Pesudo-Code example of an overloaded function to return the combined energy of two of the same particles is as shown.

*CombinedEnergy (Generic Particle1, Generic Particle2)*

*{return(Particle1.RestMass\*c2) + (Particle2.RestMass\*c2)}*

*CombinedEnergy (Photon Particle1, Photon Particle2)*

*{return((h\*c)/Particle1.Wavelength) + (h\*c)/Particle2.Wavelength)}*

The first function uses the equation E = mc2 (E = Energy, m = Rest Mass, c = Speed of Light in a vacuum) to find the energy of a particle however this does not work for photons due to them having no mass and therefore by this equation zero energy.

The second function uses the equation E = h\*c/Wavelength (E = Energy, h = Plank’s Constant, c = Speed of Light in a vacuum) to find the energy of the photon, which will not work for the other particles.

## Areas involving technical complexity

Specific functional problems solved and specific functions used

Selection

Recursion

Map

Fold

# Technical implementation

[This is your complete code, with some annotations added to highlight complexity and/or reference sections in Design]

Paste in in colour

Use a class diagram

Need to add subheadings for all classes used

Add GitHub link + screenshot of checkins

# Testing

[Evidence that the system works. Can be a reference to one or more video demos on YouTube. ]

[Evidence that it works robustly, and for multiple scenarios]

[Include a detailed test plan, which should specifically highlight edge/boundary cases and error trapping]

[Emphasis is on broad testing, of the *main* functionality, not peripheral functionality such as ‘log on’]

# Evaluation

How well did you meet your own objectives?

Evidence of *real* feedback from stakeholders and/or real users.?

Evidence of objective ‘reflection’ on your own project.

Further directions.

References