18/6/2018

* Started to work on vectors. 3 Dimensional to work between particles to decide if they would interact between each other.
* Need to start a random number generator that has random and programmable varieties.

19/6/18

* Random Number generator Class with a true random number generator and one that uses a seed number. They both only produce random numbers between 0 and 360
* The model cannot know about the implementation and therefore for the random number generators cannot be hard coded in and must be changed in the console for testing.
* The RNG return an array of the numbers [2] which is the angle between x and y and the angle between this plane and the z plane.
* Using Interfaces allows the user to decided on the random number generator required on runtime.
* Need to now work on how the ejection vector and the movement vector relate.

24/7/18

* Test method set up to test the movement and ejection
* Only one test included so far to test the programmable random number generator and it has passed
* Created a collision Class which has an annihilation collision in (Particle + anti-particle = 2 photons)
* The collision uses to methods to determent the total energy of the two particles using rest mass and velocity. This property would probably be better for the particle themselves to calculate and know themselves to avoid complication
* Need to implement a particle tree as particles are being created when they are needed in the vectors class and this is completely unworkable
* Photon class needs to be located in a suitable area
* Created a plan for the development of the project in layers (Like a cake) so parts of the program do not depend on other parts that they should not.

26/7/18

* Started the particle layer for the main project
* Need to find the actual values of the rest masses for the hadrons
* Created the Quarks but only the Up and Down flavours have hold any data
* Meson class set up but I need a list of all the mesons and their data to be able to add them as there are a lot
* Implemented a test to check if the particles are completed and the test will fail if they are not. This is a checklist for me to work through when I can find out the specifics of each particle

1/8/18

* Added a spin property to all particles, it is currently only a Boolean data type but needs to be changed to Left and Right somehow (New class?)

2/10/18

* Decided to try and go functional, this involves:
  + Particles become immutable
  + Functions become static
  + Generalise annialation with an interface<T> (IAntiparticle<T> where T: Particle) – where the antiparticle “inherites” the IAntiparticle<particle>
  + See RNG sheet
  + Use Tuples
  + Cannot change already made variables
* To move all of my projects into one project and then create folders to separate them

-Needed to make the properties in the classes protected and not private as that does not work

12/10/18

* Try to implement Feynman diagrams for the electron capture to be able to produce a more visual program
* This diagram currently is very hard coded and I need to find a way to create it by using the input and output objects
* I have decided to make the particle objects themselves know what symbol they will use for the Feynman diagrams
  + I am not sure if this is the best idea as I am struggling to find a way to make it more automated, I will have to rearrange what the perimeters are to both the actual collision functions and the Feynman Diagram function
* Rearranged the perimeters that are passed in and this now works, although it program is still quite dependent on almost the same functionality being coded multiple times throughout the collision functions
* Next step is to change the annihilation collision from just between protons to whatever the user inputs

30/10/18

* Created a new program to try and make the project more functional at the core
* Took the FList class from Dr Pawson and declared It in my project as it was a difficult part of the project

2/11/18

* Made the Annihilation function pure and have successfuly implemented the recursive list
* Used reflection to add protons and neutrons to the atomic creator lists

9/11/18

* Using the Quadrivia Nuget package to allow the streamlining of certain functions such as functional lists.
* The package is public

16/11/18

* Using lambda for add func so more functions in Collision Functions can use the add func. And it is easier to read
* I had a list of particles and I want to obtain a list of their rest masses so I decided to use mapping
* Removed a lot of excess code from the annihilation function. Now the function returns a tuple and in the tuple instead of p1, p2 it is now just function calls. The Output particle list has been removed as it was not needed and the variables holding the total rest mass and total energy have been removed. Now it is all in functions and func’s
* Removed the input particle list. The annihilation function is now only one line without any sequencing or variables. This allows the function implementation to be written on one line that has be separated on separate lines for readability

21/11/18

* Had a conversation with Client and he recommended adding decays to the list of “collisions” and neutrino interactions and electrostatic repulsion. He also said base collisions off the bubble chamber and not the large hadron collider for computability
* Simulation would be improved by a cyclotron function

27/11/18

* Using generics for the vectors to allow me to parse any type to one function and then apply a vector to it.

30/11/18

* To be able to let on vector that is intrinsically connected to another know what the other is:
  + - * Haskell – fn = let u1 = Random in (n + u1, n – u1)
      * C# foo {var u1 = Random Vector

Return tuple.create(n + u1, n-u1}

* This looks like cheating but it is just a basic version to all me to compile, will change later
* Need to look into random source code and will write up about it in project
* Started to add tests for Collision and Vector Functions

4/12/18

* Tests are working for set vectors
* Now randomising the vectors
  + This means the original FRandom DateTime needs to be implemented once and everything else must be based of that first Frandom meaning I need to pass the random through the code.
  + Added a second Frandom.Next to the z coordinate so the x,z coordinates were not the same.

- The vector functions can now produce a single ejection for any particle (Particles and photon are split, overloaded functions), a opposite ejection (Particles and photon are split, overloaded functions). And all values x,y,z are randomised each time.

7/12/18

* Need to think about adding a function where the parameter is an old photon and it uses the old photon to create a new one.
* Need to remember to set properties in overloaded constructers

11/12/18

* Pair production, annihilation now both work with full vector ejection support. However need to implement continued movement vectors.
* Moving on to finalising electron capture
* Need to achieve a cyclotron that has a magnetic field and multiple electrons firing in.
* Need to add the lifespan for each particle to then find the distance ejected.
* Added a distance calculator for all particles using generics and an overloaded photon function (again) to find the distance the particle travels based on velocity and lifespan

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 as it will be removed for a cleaner graphical UI later.
* The graphical UI will be the last part of the program to be developed.