GOD BLESS ENGINE

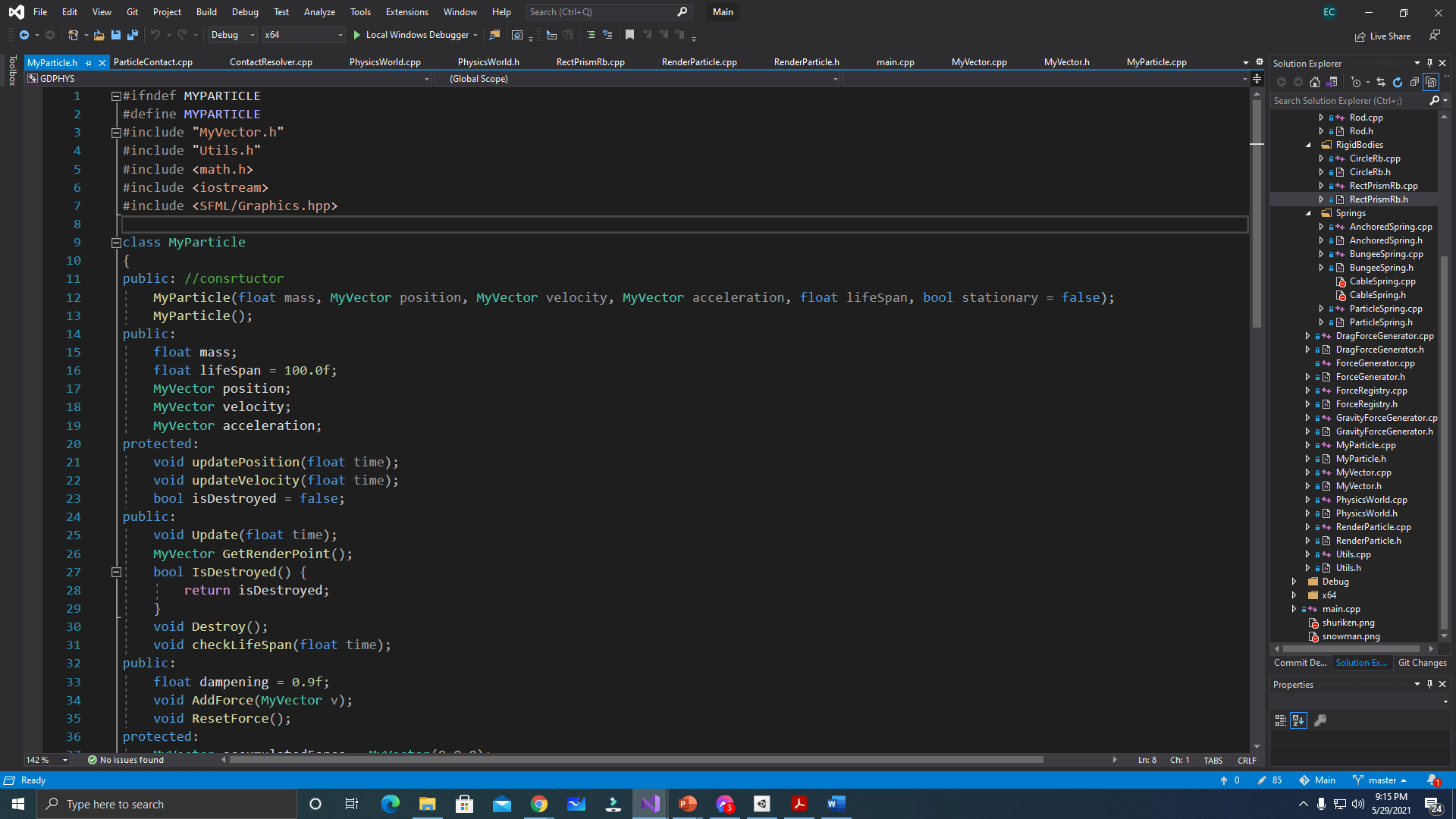
Phase 3: Rigidbody

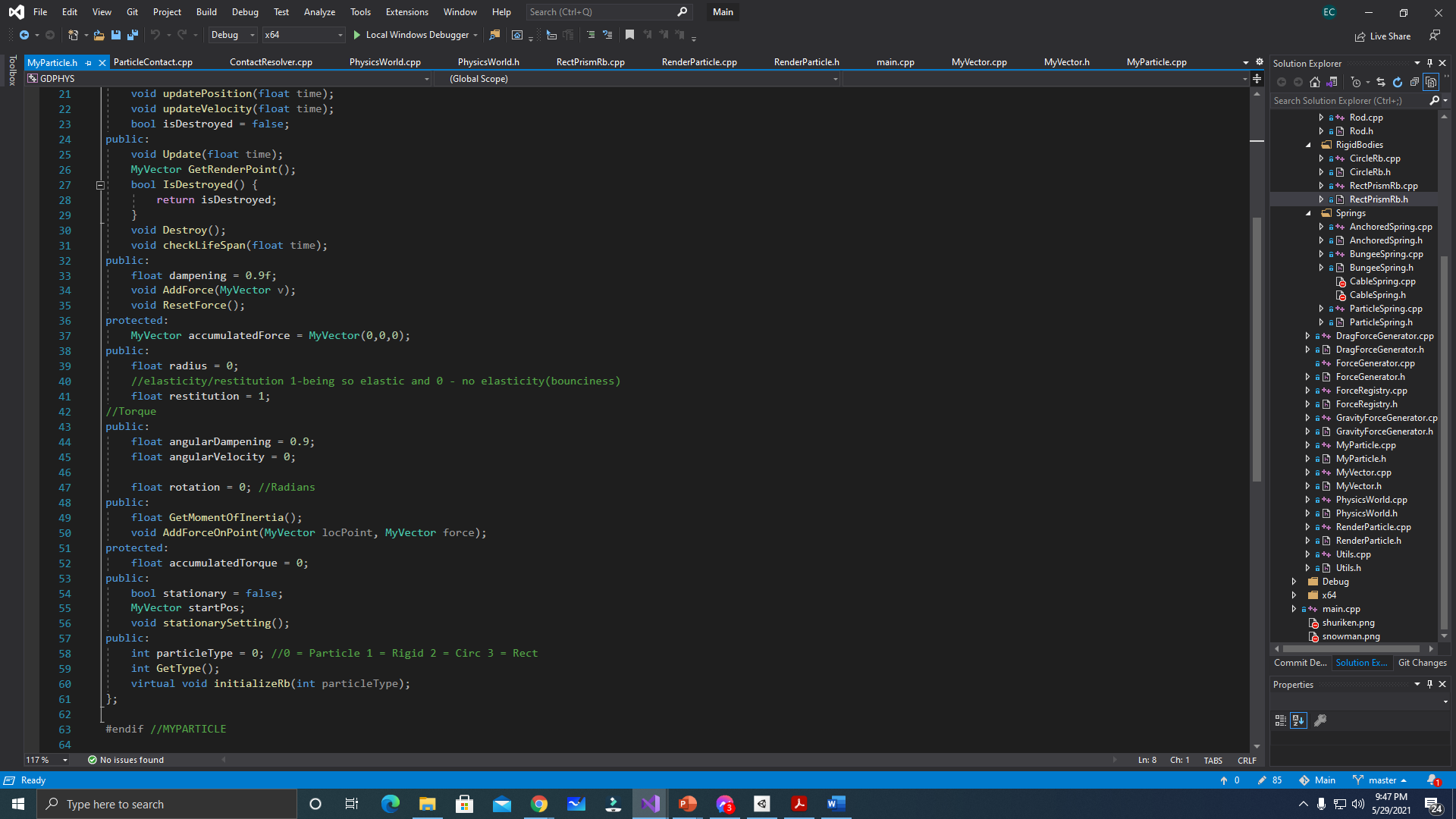
Emerson Paul P. Celestial

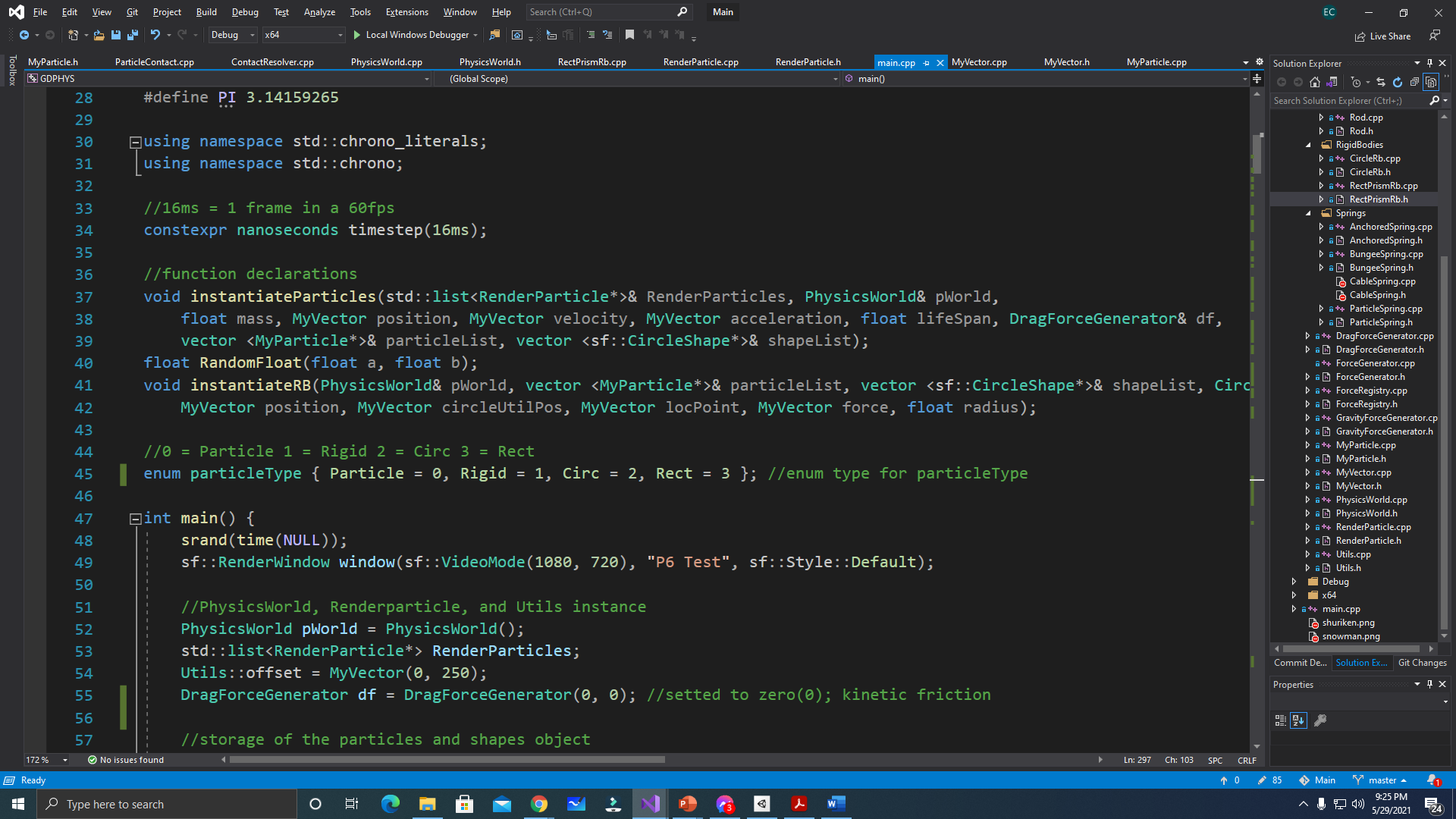
Joseph Christopher Santos

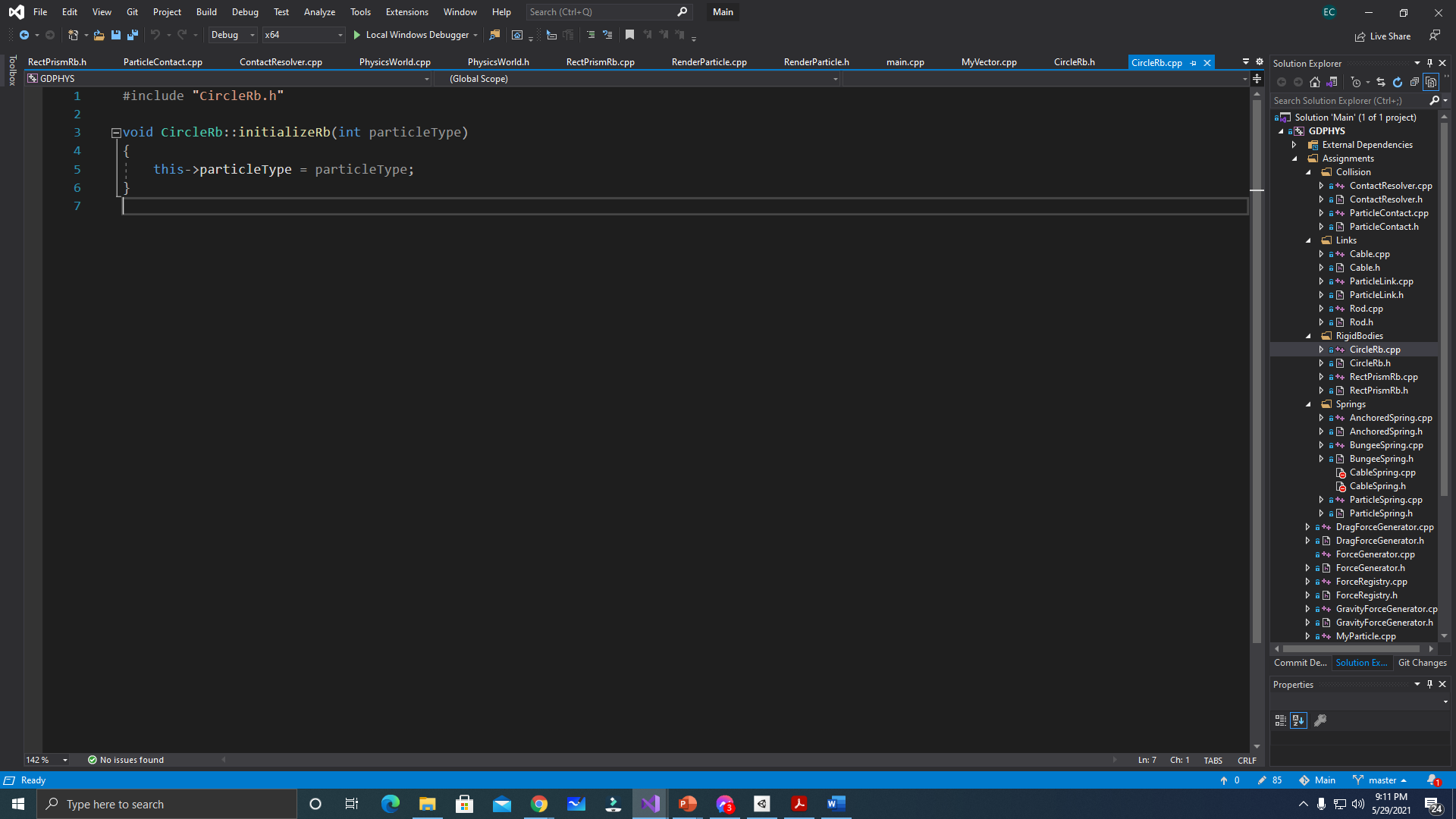
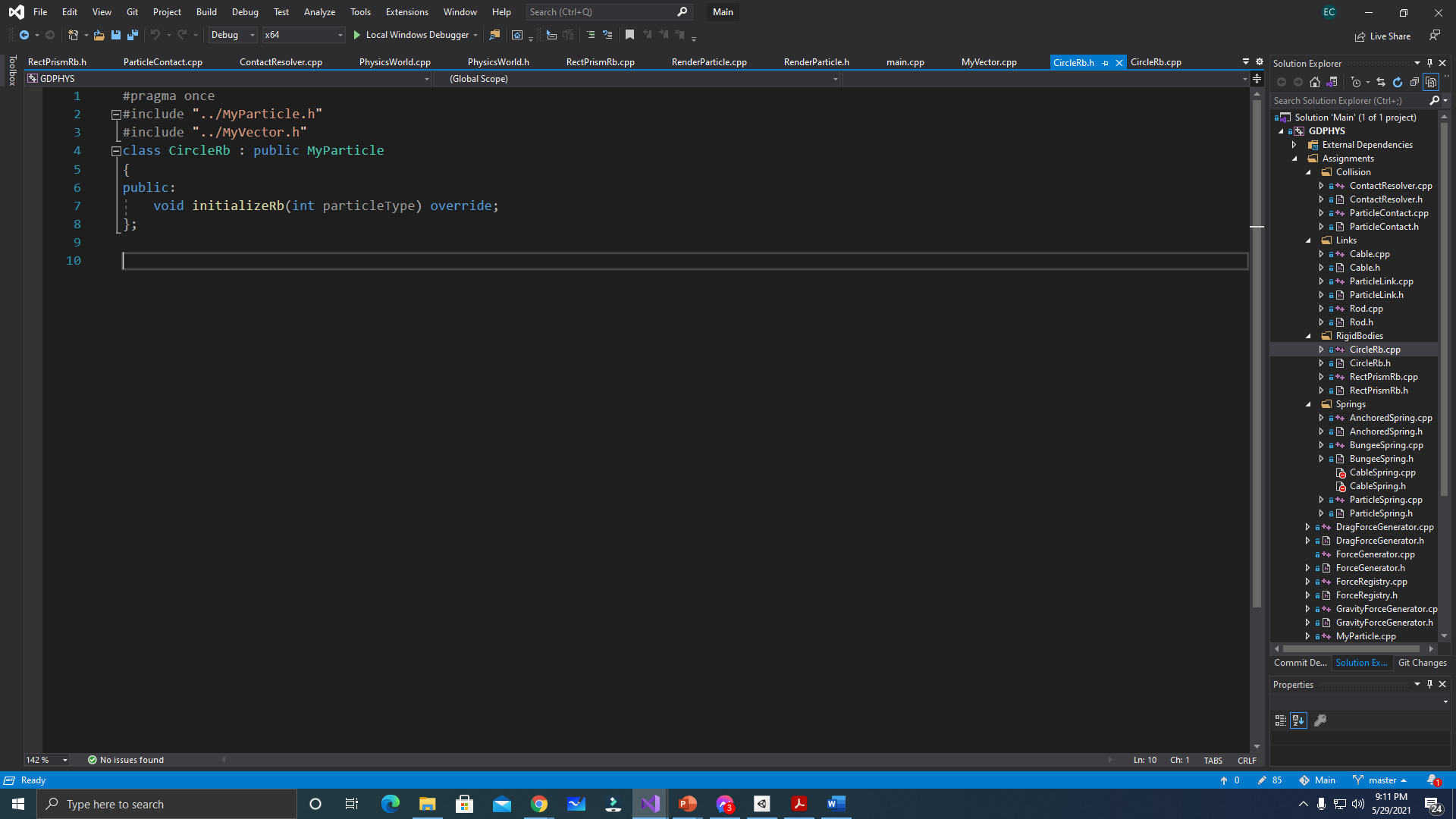
Teacher: Cala, John Raymond B.

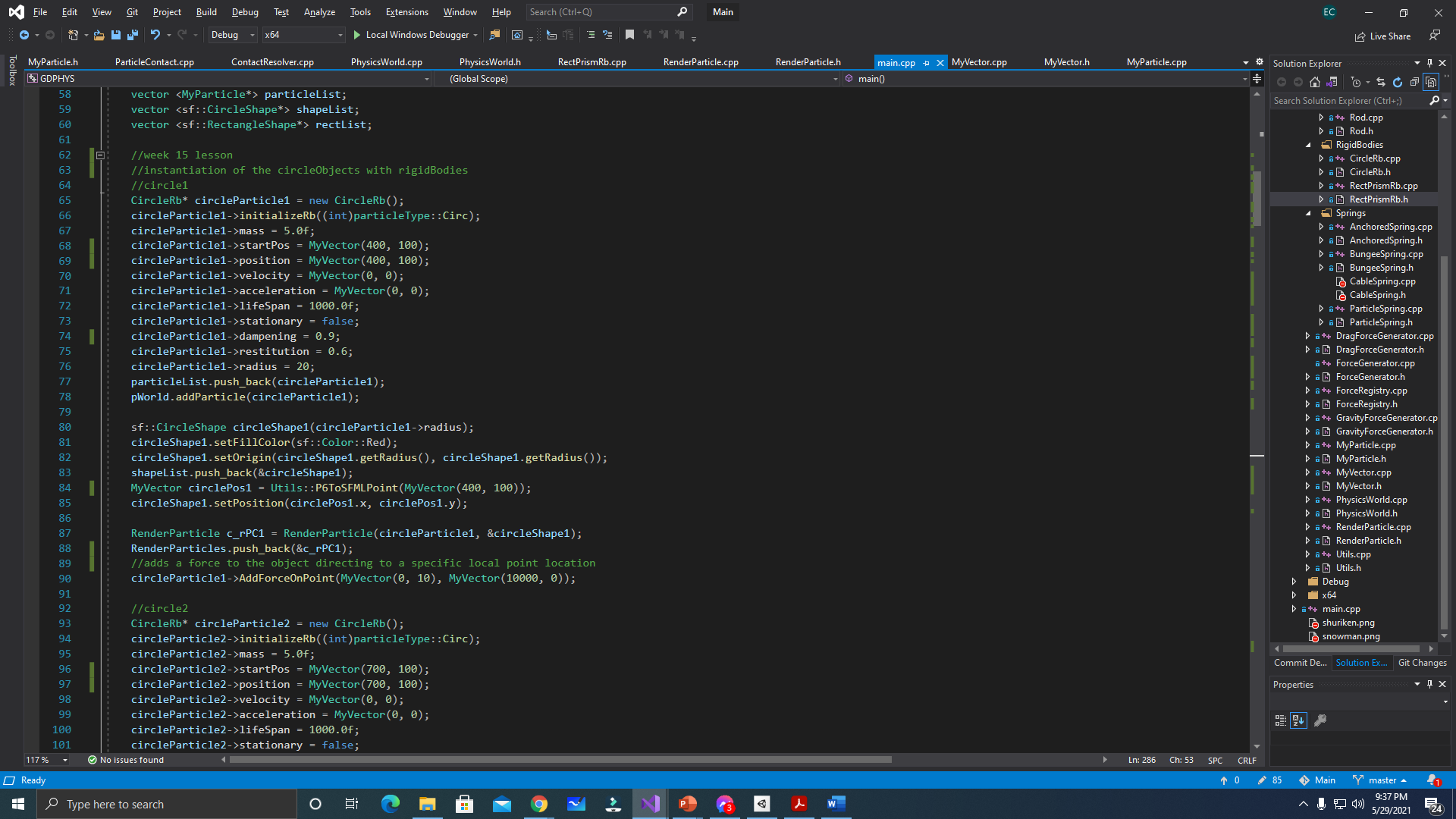
For the objects that will have a rigidbody component into their system, they will have a variety of type depending on its shape. Both classes will be extended to the particle class which will have most of the physics components of a particle. Generally, physics rigidbody component will just be an extension for the implementation of regular and irregular shape structures. In the main.cpp, there is an enum list for the particle types for easy set of particle types for the objects.

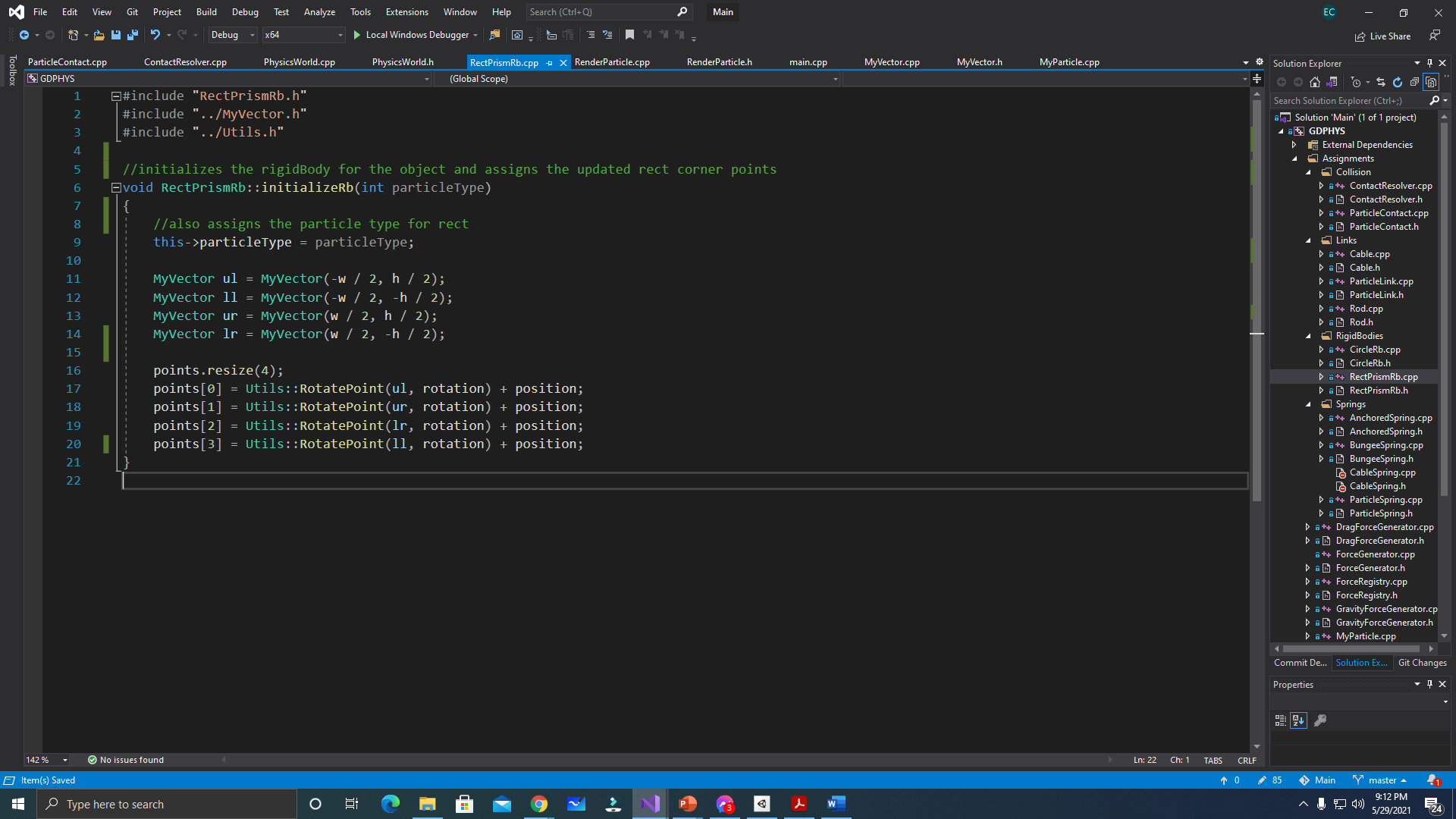
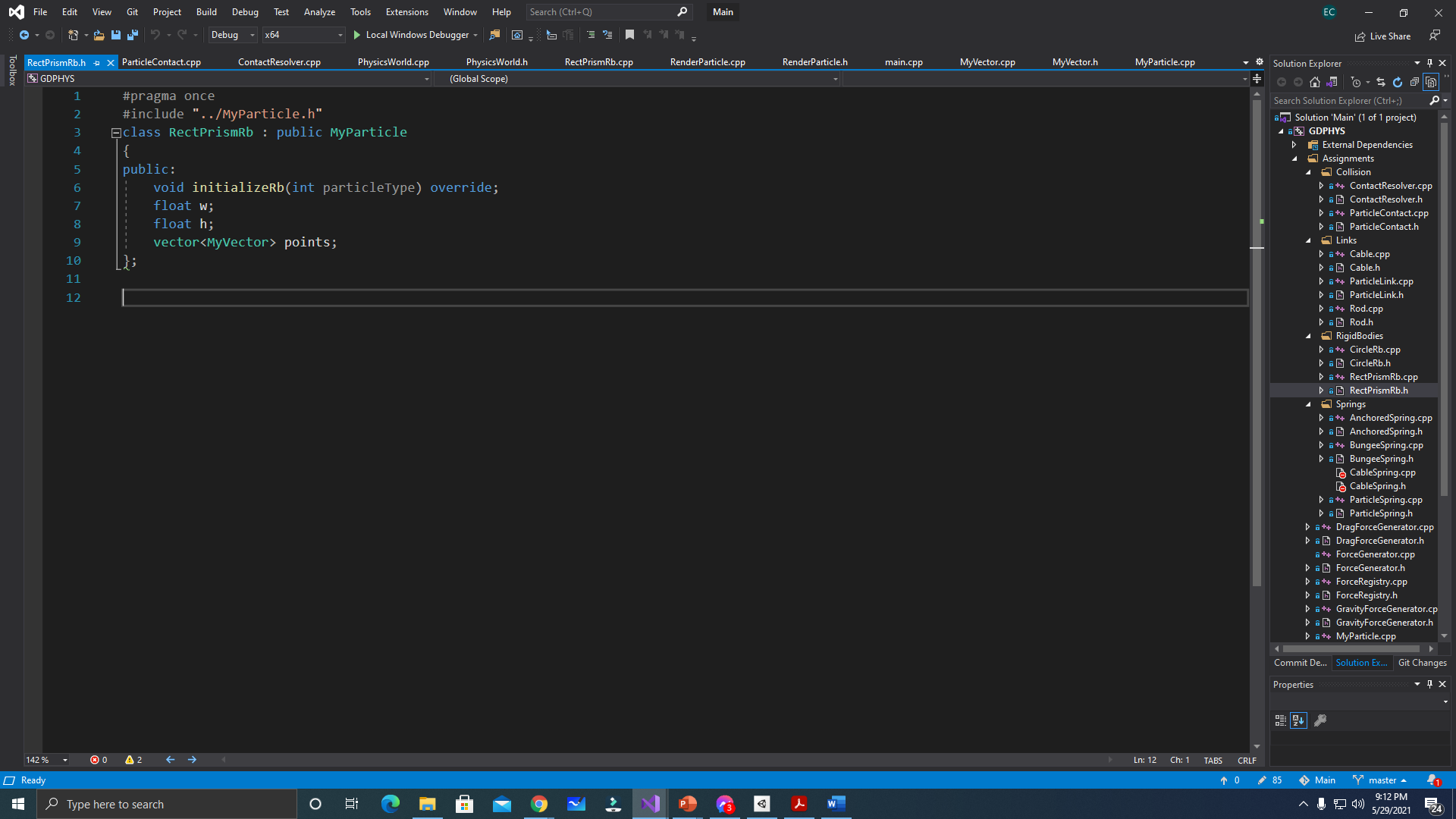
  
//Particle Class (Base Class)

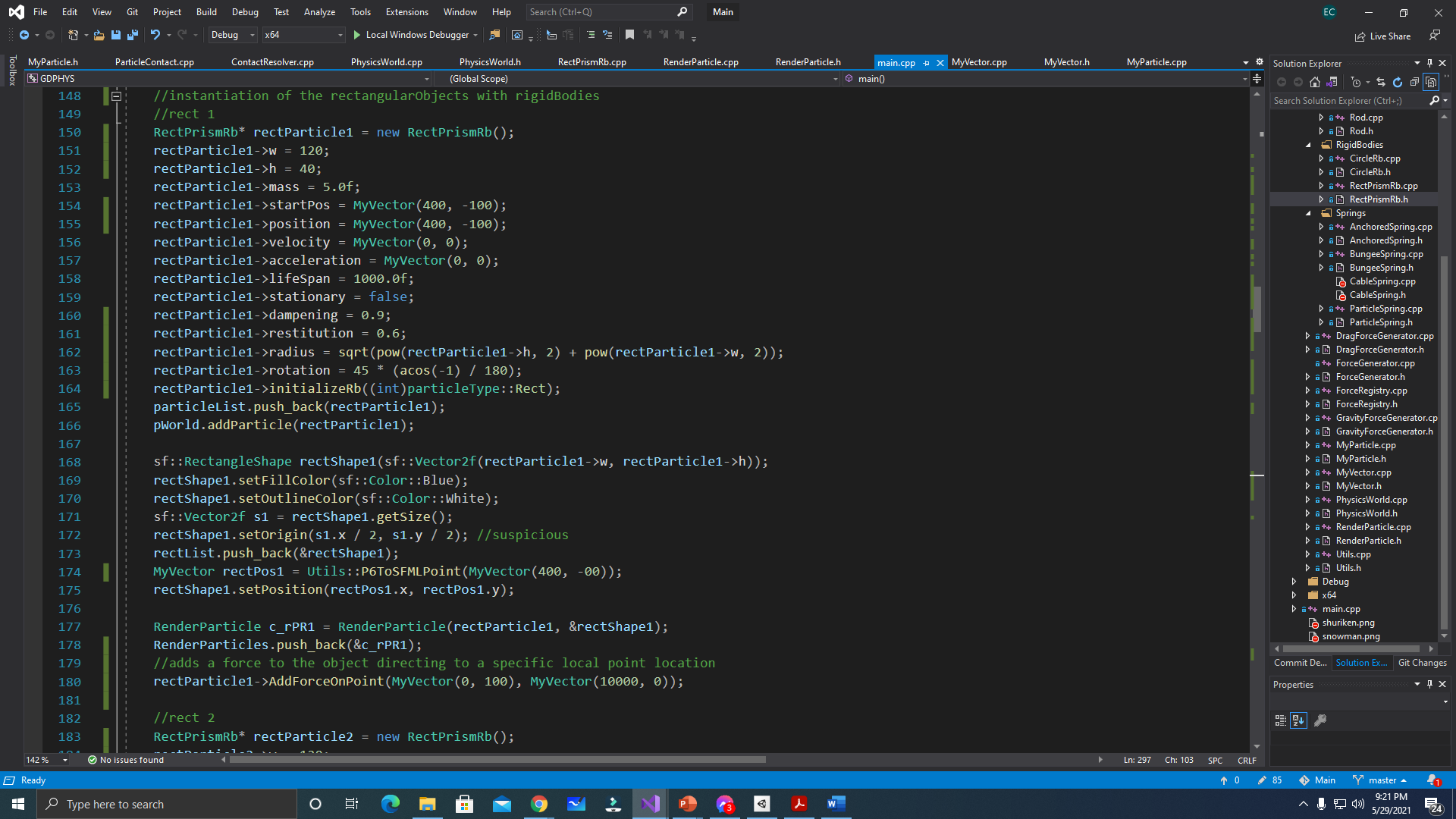
  
//2nd part of the MyParticle Header(Base Class)  
  
This is where the attributes for the particleType is declared.

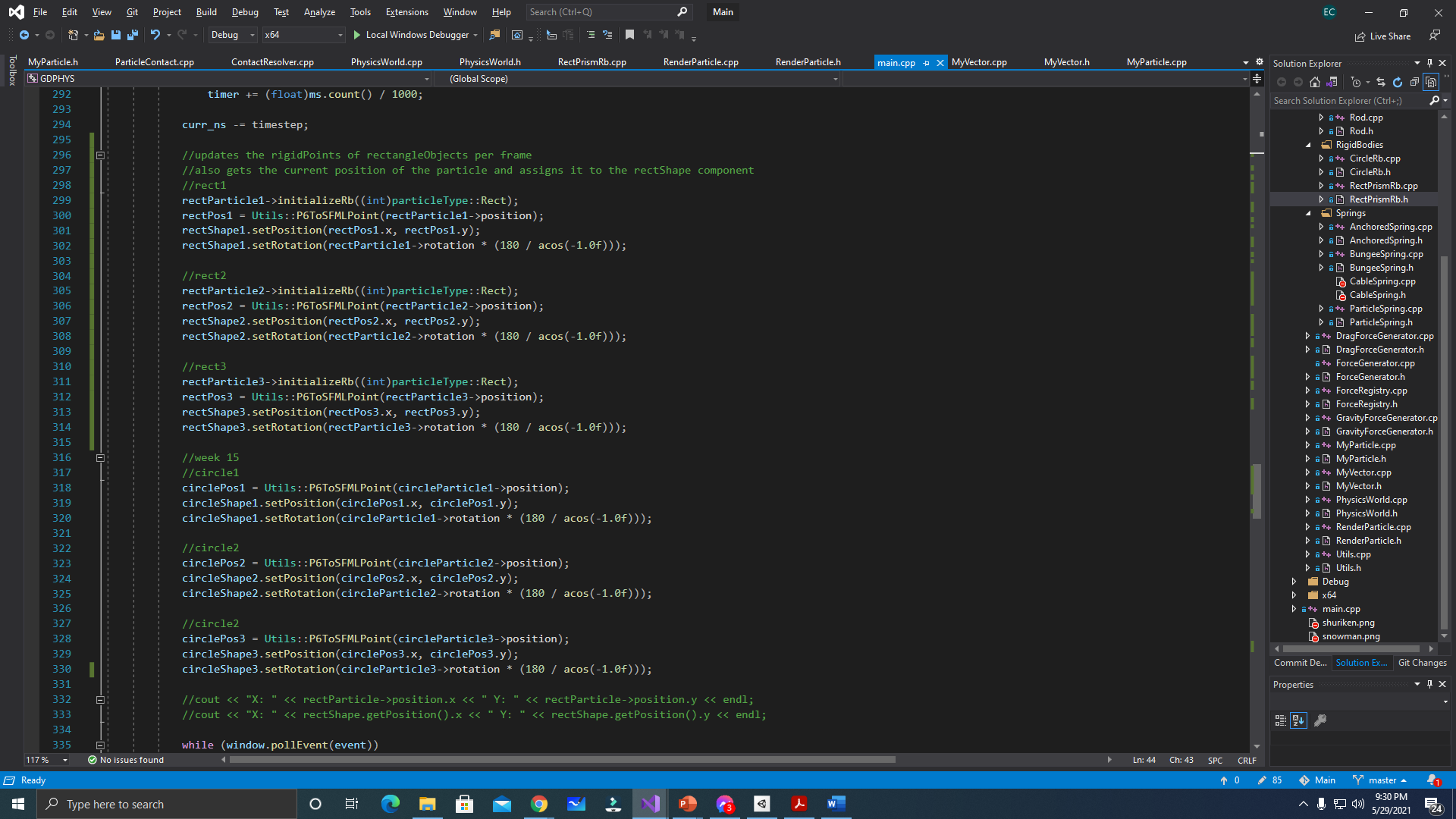
  
//Enum “particleType” list

  
//CircleRigidBody Class (Derived Class)  
  
The circle rigidbody class will inherit the MyParticle and so its attribute “rotation” and “radius” are already setted as public in the MyParticle class so no more additional attributes in here. The initializeRb will just get the particleType of this class which is 2. The instantiation of a circle will be seen below:

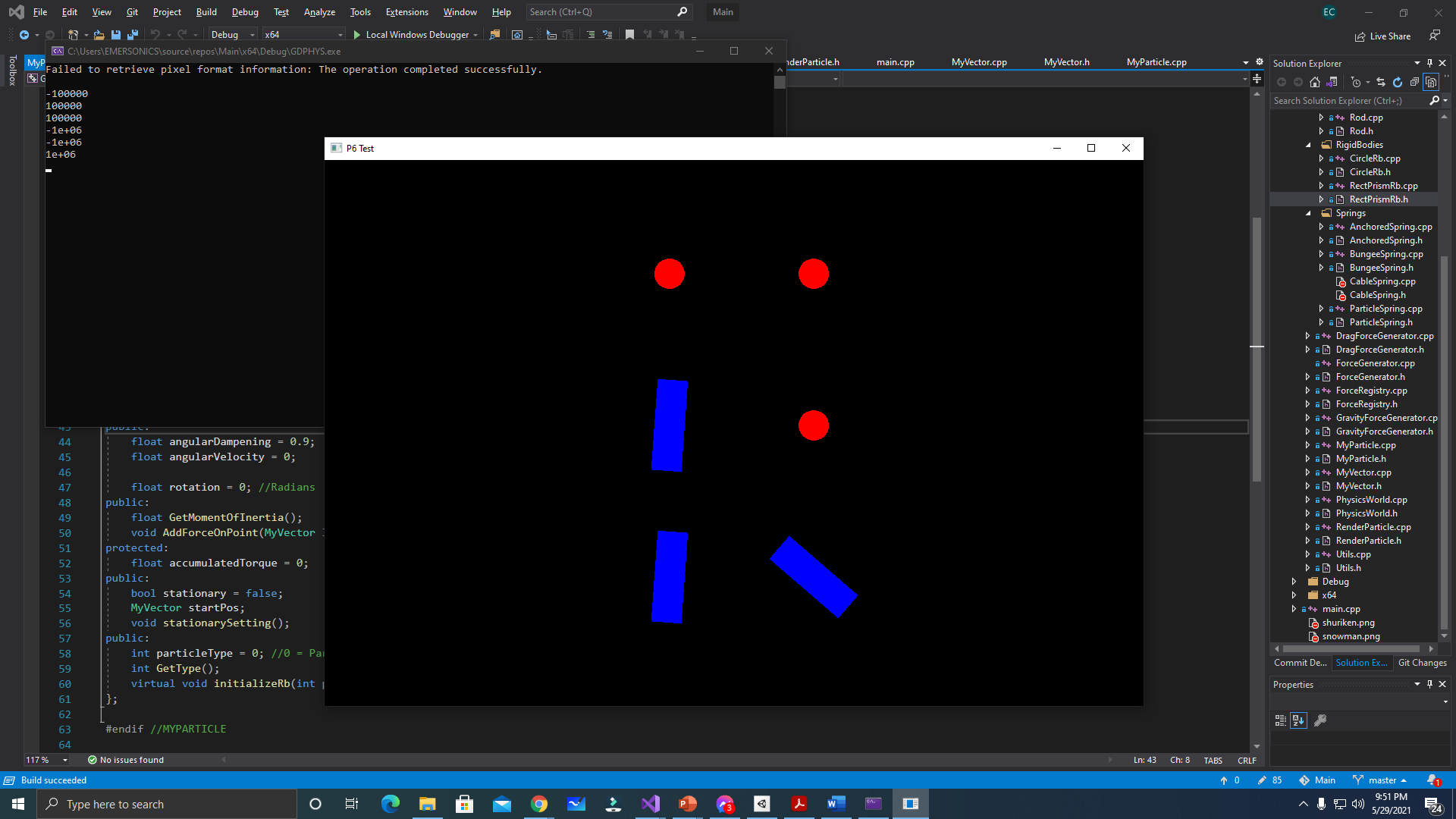
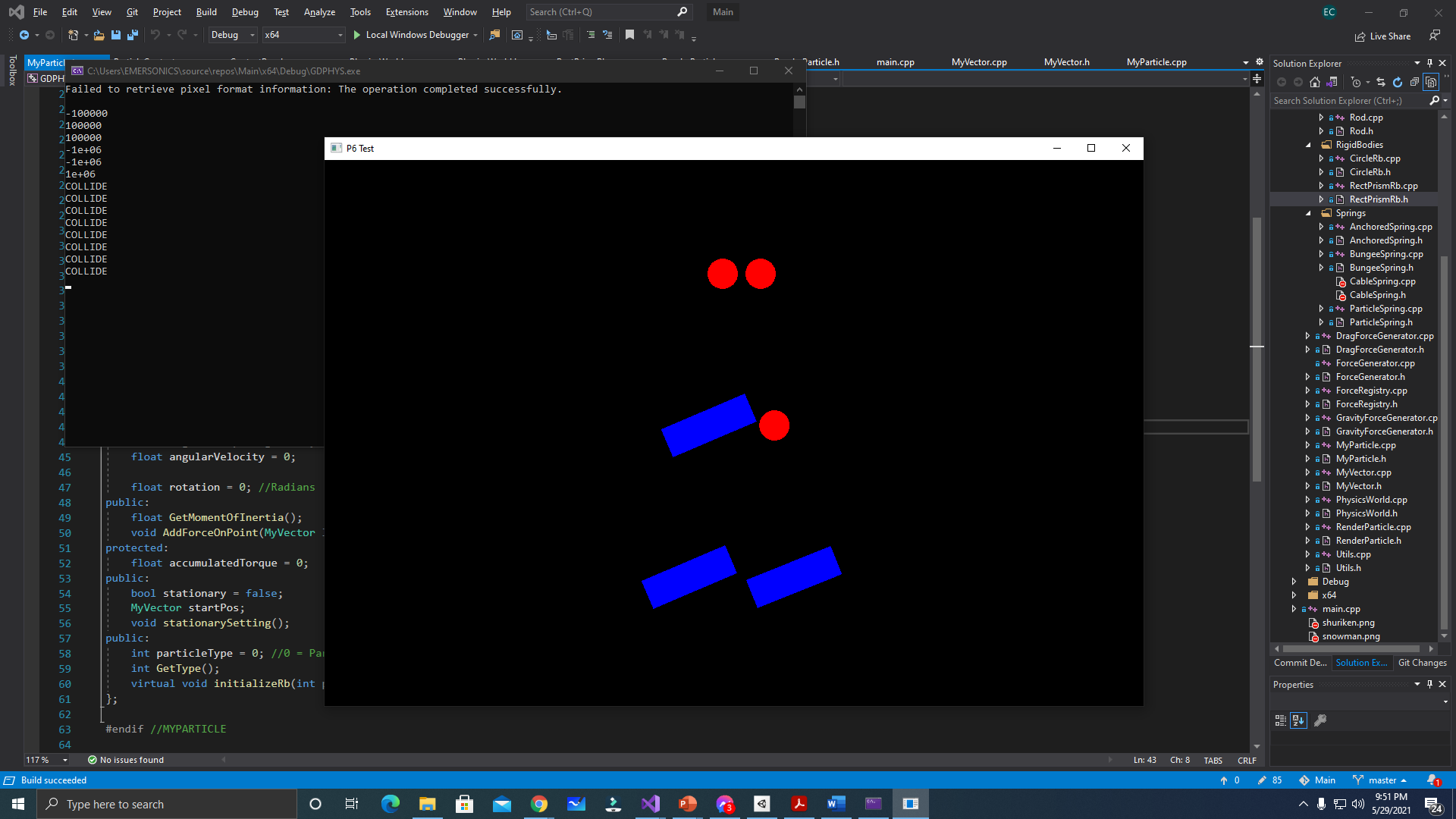
  
  
The first line will have the instantiation of a particle with a circleRigidbody component. Next step is to call the “initializeRb” function to set the particleType of the object. Then next will be the set of values for the circle particle like the mass, radius, velocity, position, etc.. The lifespan is the duration of the object in the program, so when it reaches ‘zero’, then the object will be destroyed in the program. The next step is to create a circleShape drawable for the particle so that it will be drawn in the console. Lastly, to add a force to the object, the particle must call the AddForceOnPoint to push the object on a certain direction coming from a set local point.

  
//RectangularRigidBody Class (Derived Class)  
  
- this holds the RigidBodyPoints corner of a rectangular shape. The user should call this inside the update loop so that the rigidbounds will be updated per frame in the program. Instantiating a particle that holds this component should call this function after all the require properties are set, like the rotation, width, height of the particle. To have a better picture of it, here’s an example of instantiating a particle with rectangular rigidbody:

  
In the first line, an object is instantiated and has a rect rigidbody. After that process, the required attributes for the rigid body must be set first, then call the “initializeRb((int))” next. Initialization and setting of attributes(position, radius, rotaion) for the shape drawable is the next step to display the image of the set particle. The particle and shape will now be placed in a RenderParticle object so that it will be drawn in it. RenderParticle will now determined if the passed values have different types, and this is done due to different kinds of particle(Particle, Rigid, Circle, Rectangle) so passing any types of particle and shapes is just fine.

  
//updating certain rigidBody points and positions.

This part is located inside the while loop or the caller of each frame in our program. The list of codes here are for the update of their current position in the physics update functions is being computed in the “pWorld.Update((float)ms.count() / 1000);” line. The “initializeRb” is called here again for the rectangle objects since we need to get it rigidbound per frame and to also check for collisions with other objects.

After all the steps had been followed, then displaying the simulation can now be started.   
  
  


The objects with the rigid Body will now be seen colliding with each other neatly, colliding to each edges. Three scenarios can be seen in this demo: (1) Circle to Circle, (2) Rectangle to Circle, and (3) Rectangle to Rectangle