Project 7 Update

Status Summary:

Project Title: Physics Summary

Names: Roman Lynch, Ethan McAleavy, Steven Darbie

Work done:

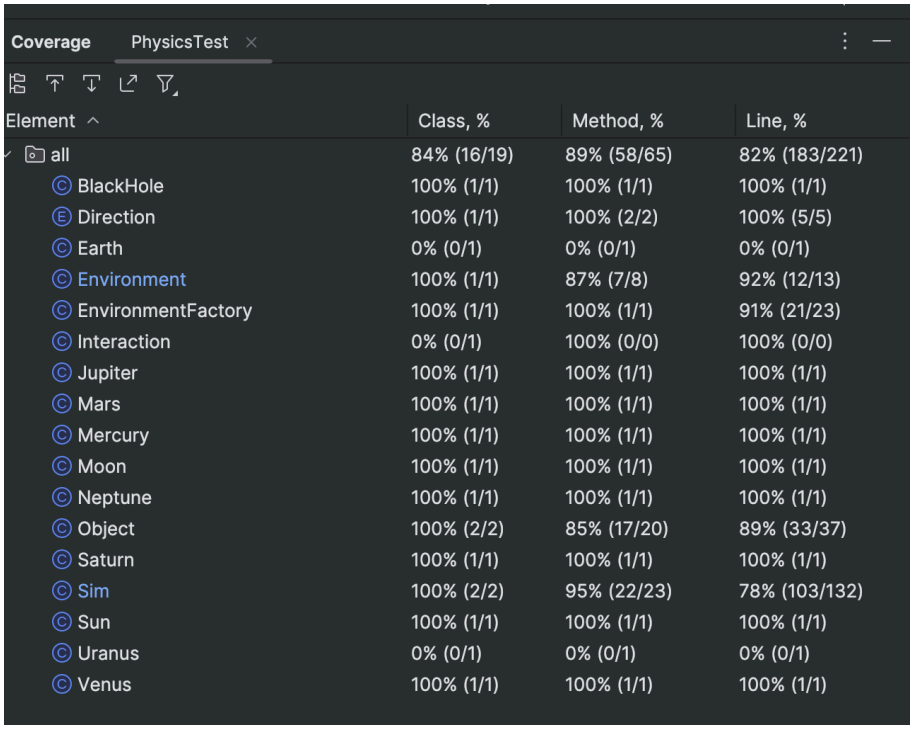
So far, we have implemented basic functionality into our project. Essentially, this means we have implemented a basic kinematics physics engineer, and created a basic function to apply said engine. These functions include an Environment Factory, that allows us to create different environments from a range of options. As well, this includes a Simulation Builder pattern, which is used to build a simulation with a specific environment, specific objects, and specific physical constants (such as the elasticity of collisions). This builder will be used to run simulations with specific attributes. As well, we implemented several subclasses of the superclass “Environment”, so that we could use the Environment Factory to make several prefixed environments with specific gravity presets. These prefixed environment subclasses included the planets of the solar system, as well as a Black Hole environment, and a Sun environment.

Some issues we encountered were how to deal with collisions of objects, and how to visually represent what was happening with the simulator. To deal with collisions, we determined it would be best to limit object movement onto 4 axes (up, down, left, right), and keep the system on a grid. So, essentially when two objects are in the same square of a grid, it is considered a collision. As for how to visually represent the simulation, we opted for a before and after approach. Essentially, we plan on showing the position of objects before the simulation begins (including velocity and mass of objects), and then displaying their positions after the simulation ends (including velocity and mass of objects). This simplifies our implementation, and allows us to use kinematics, which is more ideal for our understanding. As well, we did encounter some issues with GitHub, and Gradle, which meant we had to rebuild our repository from scratch. Luckily, we were able to get everything transferred to the new repository without any issues.

As previously mentioned, we have implemented the Builder and Factory design patterns so far. They have been very helpful in simplifying the iuse of the simulator. This is especially true in the case of the Sim class, which uses a builder pattern to build and run the simulation with specific

conditions. The use of the builder simplified the code necessary to run the Simulation two fold, and made the syntax a lot cleaner. As for the factory pattern, it has really helped us simplify how we create different Environments, since we really wanted to implement so many different kinds. This way, we can create whichever environment we want in one easy line, without having to worry about the intricacies of creating an environment of a certain type.

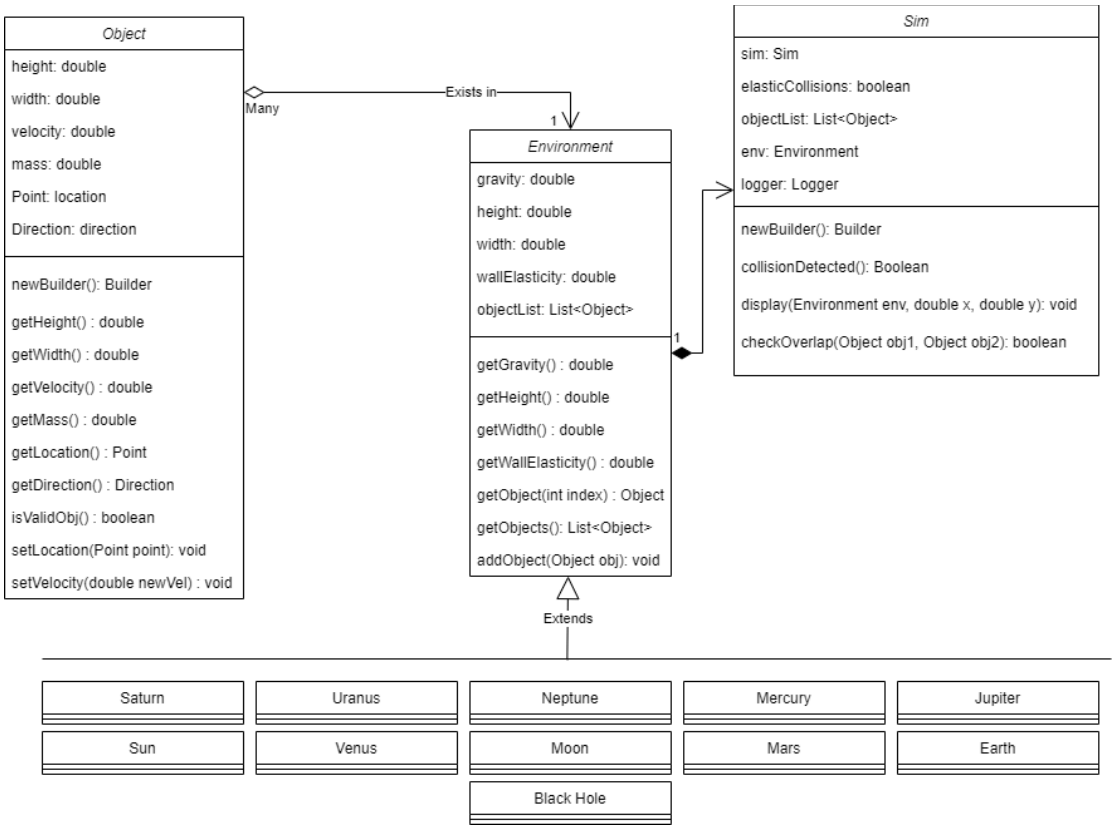
As far as work across team members goes, Roman was in charge of implementing the factory pattern for the Environment Factory, and creating the subclasses of the Environment superclass that would be used in said factory. As well, he was in charge of implementing some of the functions for the Builder pattern in the Sim function. Steven was in charge of creating the Objects class, and implementing all the features necessary for the Objects to function as desired. As well, Steven contributed to the Builder pattern within Sim and created the UML diagram. Ethan was in charge of most of the actual physics engine, working to implement the physics within the Sim class so the simulation could run as desired. As well, he helped immensely with the initial planning and the designing of the engine.



Element	Class, %	Method, %	Line, %
all	84% (16/19)	89% (58/65)	82% (183/221)
BlackHole	100% (1/1)	100% (1/1)	100% (1/1)
Direction	100% (1/1)	100% (2/2)	100% (5/5)
Earth	0% (0/1)	0% (0/1)	0% (0/1)
Environment	100% (1/1)	87% (7/8)	92% (12/13)
EnvironmentFactory	100% (1/1)	100% (1/1)	91% (21/23)
Interaction	0% (0/1)	100% (0/0)	100% (0/0)
Jupiter	100% (1/1)	100% (1/1)	100% (1/1)
Mars	100% (1/1)	100% (1/1)	100% (1/1)
Mercury	100% (1/1)	100% (1/1)	100% (1/1)
Moon	100% (1/1)	100% (1/1)	100% (1/1)
Neptune	100% (1/1)	100% (1/1)	100% (1/1)
Object	100% (2/2)	85% (17/20)	89% (33/37)
Saturn	100% (1/1)	100% (1/1)	100% (1/1)
Sim	100% (2/2)	95% (22/23)	78% (103/132)
Sun	100% (1/1)	100% (1/1)	100% (1/1)
Uranus	0% (0/1)	0% (0/1)	0% (0/1)
Venus	100% (1/1)	100% (1/1)	100% (1/1)

Class Diagram:

BDD



Scenarios:

Adding Objects to Environment

- Given an environment with defined parameters (gravity, height, width, wall elasticity)
- When new objects are added to the environment
- The objects should successfully display and be added to the environment's object list

Object Movement

- Given an object with initial parameters (height, width, velocity, mass, location, direction)
- When the simulation starts, the object should move according to its velocity and direction accurately to real physics

Displaying Simulation Output

- Given an environment with objects and defined dimensions
- When the simulation is run, the correct grid graphical display should appear correctly

- It should accurately represent and display the positions of all objects within the environment
- And provide a clear visualization of object movements, collisions, and interactions

Object Collision Detection

- Given two objects with overlapping locations
- The collisions should be detected between overlapping objects
- The objects should then have the correct velocity and direction as a reaction to this collision

Gravity Effect on Objects

- Given an environment with defined gravity and an object with initial parameters
- When the simulation runs, the object's movement should be correctly affected by the gravitational force of the environment

Wall Collision Handling

- Given an environment with defined dimensions
- And an object approaching the walls of the environment
- When the simulation runs the object should bounce off the walls with the specified wall elasticity

Plan for Next Iteration:

The plan for the next iteration is to implement a visualization to go along with the simulator. As discussed earlier, this visualization will use a before and after technique, displaying the objects in the environment before the start of the simulation, and then after the simulation ends. We believe we have implemented the other classes well enough such that the visualization should be fairly simple to create.

As well, we plan on implementing an event bus with certain events, and an observer pattern to report certain events when they happen. This may be fairly difficult, especially when it comes to the collision of objects. However, we feel we have set the classes up to properly handle such an implementation, and are confident we can do so properly.

Video Link:https://drive.google.com/file/d/1OAA5i5oSvHYo8vY4DtXxzL23FC6ozc6z/view?usp=share_link