Inertia Simulator

Amber Colletti
Brandon Conway
Aaron Setser
Matthew Shea
Yi-Chin Sun

Principles of Software Engineering
Vanderbilt University, Fall 2011

Table of Contents

Section I: Purpose

The inertia simulator is designed to teach 5th grade students about the properties of inertia by modeling how inertia affects objects of various size and shape inside a bus. The students will be able to choose what object to put in the bus (box or ball) and change the weight of the object, as well as the speed and acceleration/deceleration of the bus.

Section II: Requirements

- Allow user to create and drag objects across screen
- Allow user to define relationships between objects
- Allow user to start and stop simulation
- Allow user to delete objects/relationships
- Allow user to define attributes of all object(s)
- We will implement a simulation based on the user's input
- We will create useful error messages and useful hints (perhaps an XML to send to research group)
- Saving project
- Opening existing project

Section III: Design Approach

We began our approach by creating several use cases that demonstrated high level scenarios, such as creating a simulation or loading in an existing simulation. From these use cases, we built our first draft of a class diagram. In this diagram, we outlined all the major classes, such as our Graphical User Interface, Physics Engine, and Managers. We chose the class diagram as it is a classic diagram that allows us to easily visualize how all of our pieces will come together in the final product.

From this class diagram, we created all other diagrams utilized in our project. First we designed several sequence diagrams to cover almost any scenario a user would encounter. We chose the sequence diagrams because they are simple, yet incredibly useful when trying to visualize the flow of our project. In addition, looking at several of our sequence diagrams allowed us to find flaws with the class diagram that we were able to immediately fix. Once we fixed both the class diagram and sequence diagrams, we created our communication diagram. We found this useful in creating our project as it combines the benefits of a use case diagram and sequence diagram. Our graphical user interface benefited from this diagram, as it showed us what kind of inputs our user would need in order to properly invoke the methods involved with creating an object.

Section IV: Class Overviews

The following are general overviews of each class. The relationships between classes can be seen in the class diagram (Appendix III) and the classes can be reviewed in greater detail in the attached Javadoc Documentation (Appendix V).

edu.vu.vuse.cs278.g3.gui

Contains all classes that render the user interface. Interface files are designed using the interface editor in NetBeans and should be edited as in NetBeans.

MainWindow

Class that implements the main display window for the program. This window contains the NetLogo display window that will be modified by the physics engine.

EditObjectUI

Class that implements the edit object window. This window modifies the objects in the model and commits them to the NetLogo display.

SoftwareUI

Class that implements the create object window. This window is responsible for creating objects and properly adding them to the model.

edu.vu.vuse.cs278.g3.engine

Contains all classes that operate the physics engine. This packages consists of three files that are entirely responsible for running the physics simulation.

PhysicsEngine

Class that implements the physics engine. This class is actually a wrapper around an internally defined QueueExecutor that accepts Runnables and executes them on a single thread.

PhysicsFormulas

Class that contains flexible implementations of physics formulas used to calculate new positions and properties for objects. This class contains static methods that act as these formulas.

PhysicsActions

Class that contains higher level actions for running the physics engine. These actions are implemented as internal static Runnables for execution on the PhysicsEngine. Each Runnable should define one specific action.

edu.vu.vuse.cs278.g3.model

Contains all classes used to represent the physics data model. The two primary classes are the ObjectManager and RelationshipManager.

ObjectManager

Singleton class that handles the creation, editing, and deletion of all objects. Objects should only be edited through this class.

PhysicsObject

Base class that provides universal functionality for all objects involved in the physics engine. Subclasses should be sure to call the relevant PhysicsObject super functions if they choose to override them.

SquareObject

Class that extends the PhysicsObject to provide dimensions that make it a square, such as width and height.

RoundObject

Class that extends the PhysicsObject to provide dimensions that make it a circle, such as radius.

BusObject

Class that extends the PhysicsObject to provide dimensions that make it a circle, such as length.

RelationshipManager

Class that keeps a map of the relationships between objects. The map associates references to objects with a RelationshipType.

RelationshipTypes

Enumeration that is used to describe relationship types. The actual implementation is done in the PhysicsActions, but this is used to switch on to make the code cleaner

Section V: Operation

The program has three primary components: The graphical user interface, the physics engine, and the managers.

Graphical User Interface

The GUI accesses the physics engine and managers to control the objects and the simulation. The GUI also contains the Netlogo window which is used to display the simulation.

Physics Engine

The physics engine consists of a *QueueExecutor* and *Runnable* actions to execute on said executor. The *QueueExecutor* implements a thread that reads a queue of *Runnable*s and executes them sequentially. Each *Runnable* accesses the managers and performs some action on them, such as accelerate or move. These changes are then passed to the Netlogo window which updates the display.

Managers

The managers contain the Java representations of the Objects and Relationships. They provide basic functionality such as creating, editing and deleting objects while maintaining data integrity between the Java objects and the Netlogo objects that they are bound to.

For more details on how the individual parts work together, please see the class diagram in Appendix III.

Section VI: Walkthrough

Mixed Group Walkthrough (30 minutes)

Documents Examined: Original Class Diagram, All Sequence Diagrams

Fault #1: Physics Engine was not well developed and needed to be expanded to more properly detail the behavior.

Fault #2: User interface was not well developed and required additional dialogs to be developed.

Fault Rate: 4 faults/hour (both faults determined to be major faults)

Original Group Walkthrough (1 hour)

During the original group walkthrough, the only faults we found in our documents were minor faults, since we fixed the faults found in the mixed group walkthrough

Documents Examined: Updated Class Diagram, All Sequence Diagrams, and Communication Diagram

Fault #1: User Interfaces needed to be in three separate classes, as opposed to one in our last class diagram because three different windows were needed.

Fault #2: Ball, box, and bus attributes are similar (all ints); would be better to abstract the attributes for ball, box, and bus.

Fault #3: Sequence diagrams included classes that had been removed from the class diagram.

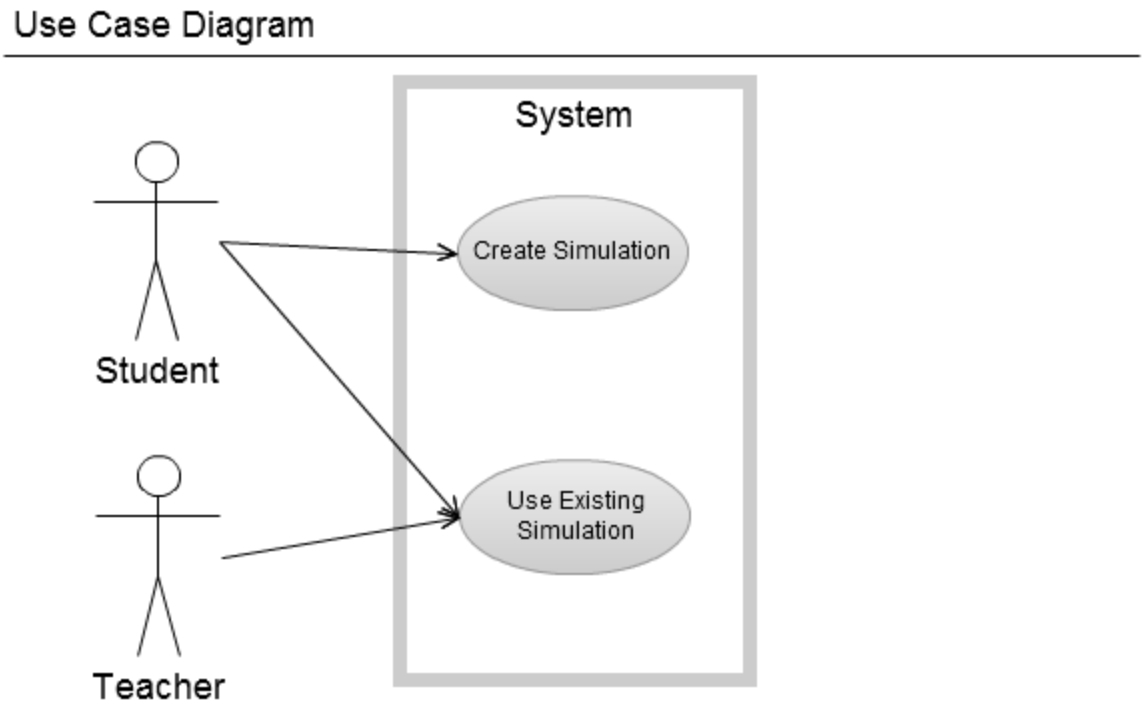
Fault #4: Ground object not necessary since it does not add anything to the simulation; makes more sense for it to be part of UI instead of a separate object

Fault Rate: 4 faults/hour

Discussion

The fault rates for our project stayed constant between our mixed group walkthrough and our original group walkthrough. The difference between the two walkthroughs was the severity of the faults we found. Our mixed group walkthrough found only major faults that needed to be fixed immediately, while the original group walkthrough found only minor faults that still needed to be addressed but did not cause major breakdowns in our project.

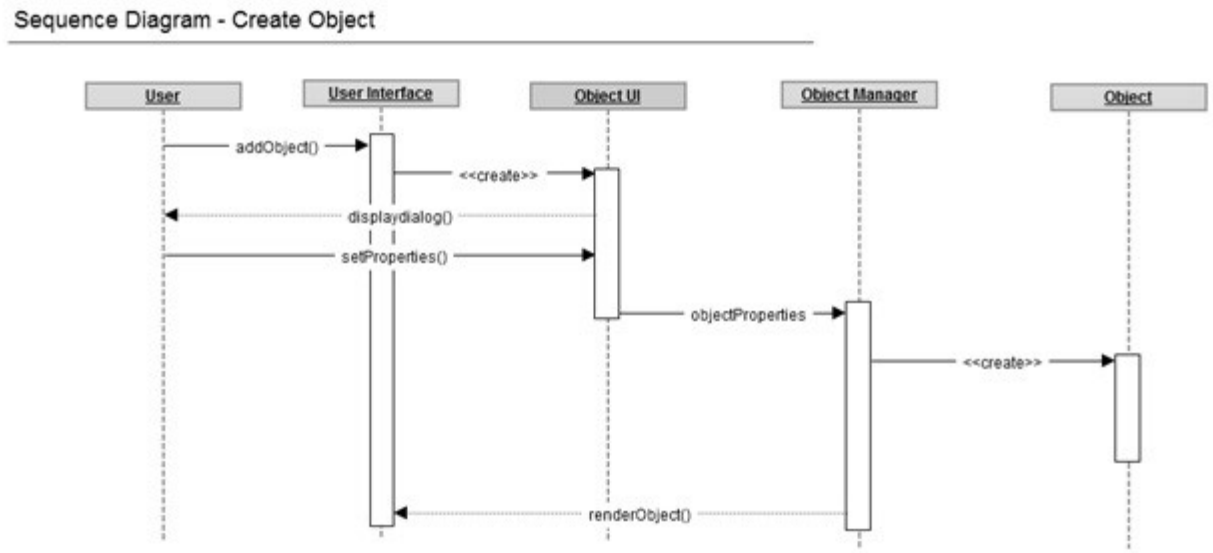
Appendix I: Use-Case Diagram



This diagram describes potential use cases for both the student and the teacher.

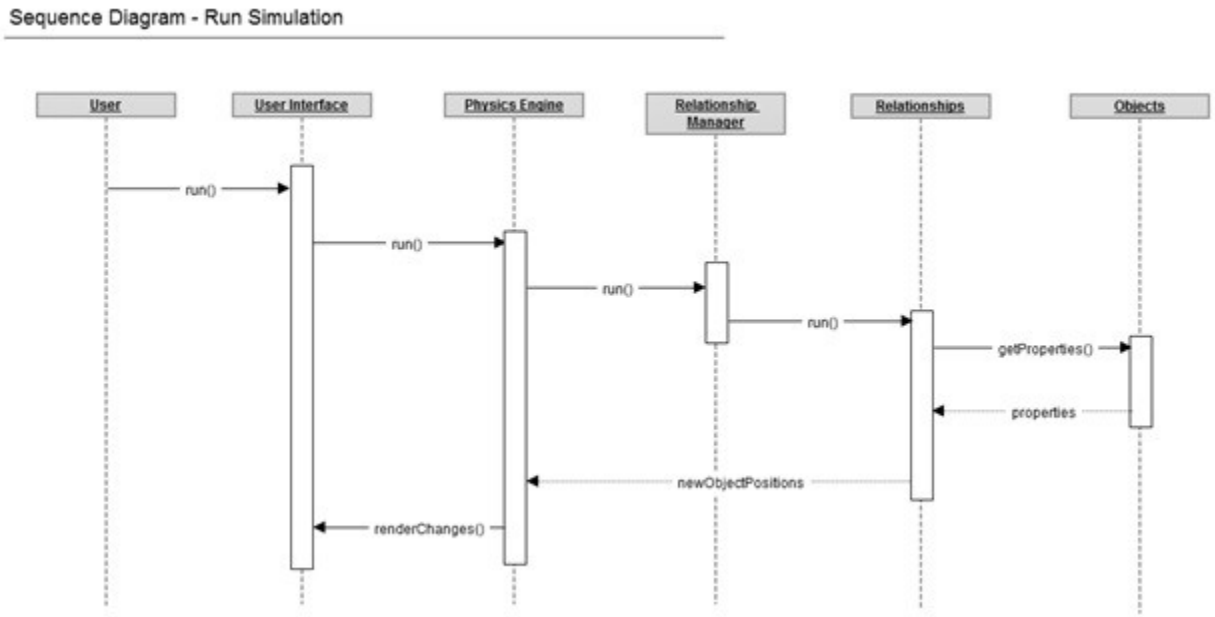
Appendix II: Sequence Diagrams

Create Object



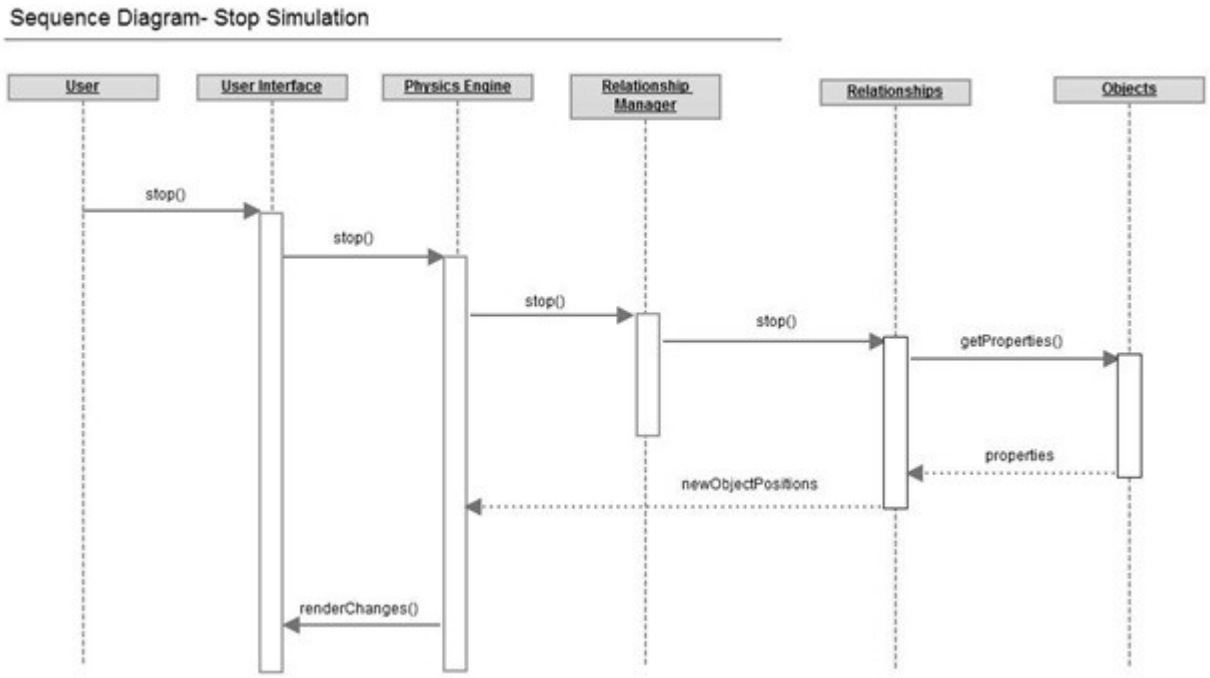
This diagram describes the methods and events taken when the user wishes to create an object.

Run Simulation



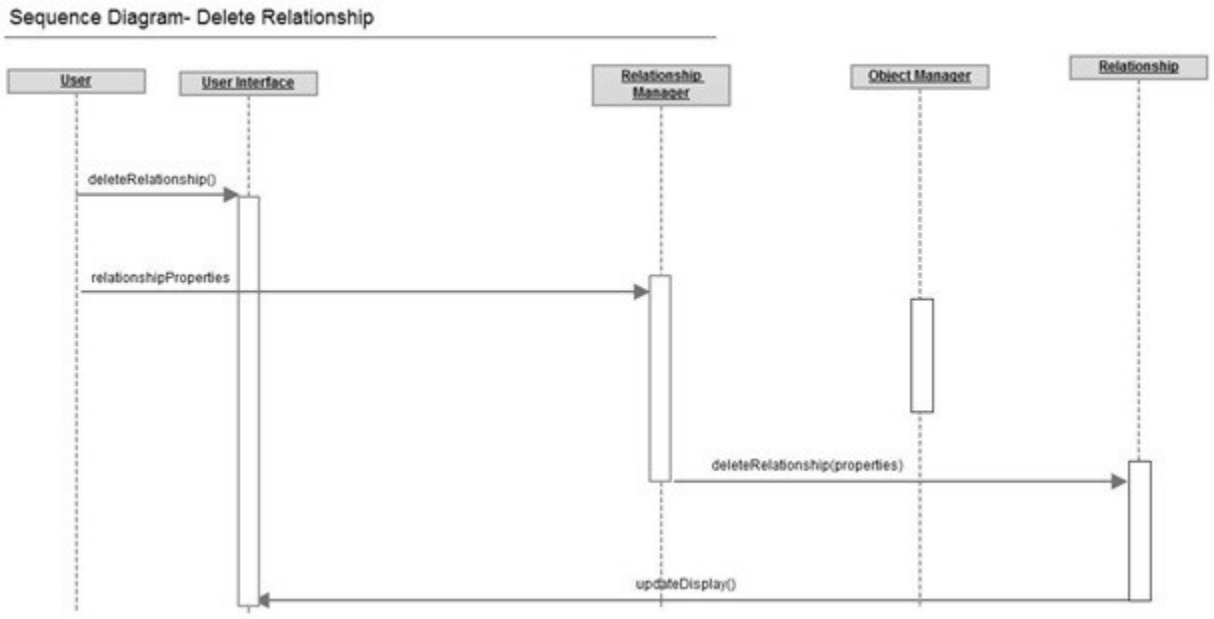
This diagram describes the methods and events taken when the user wishes to start the simulation

Stop Simulation



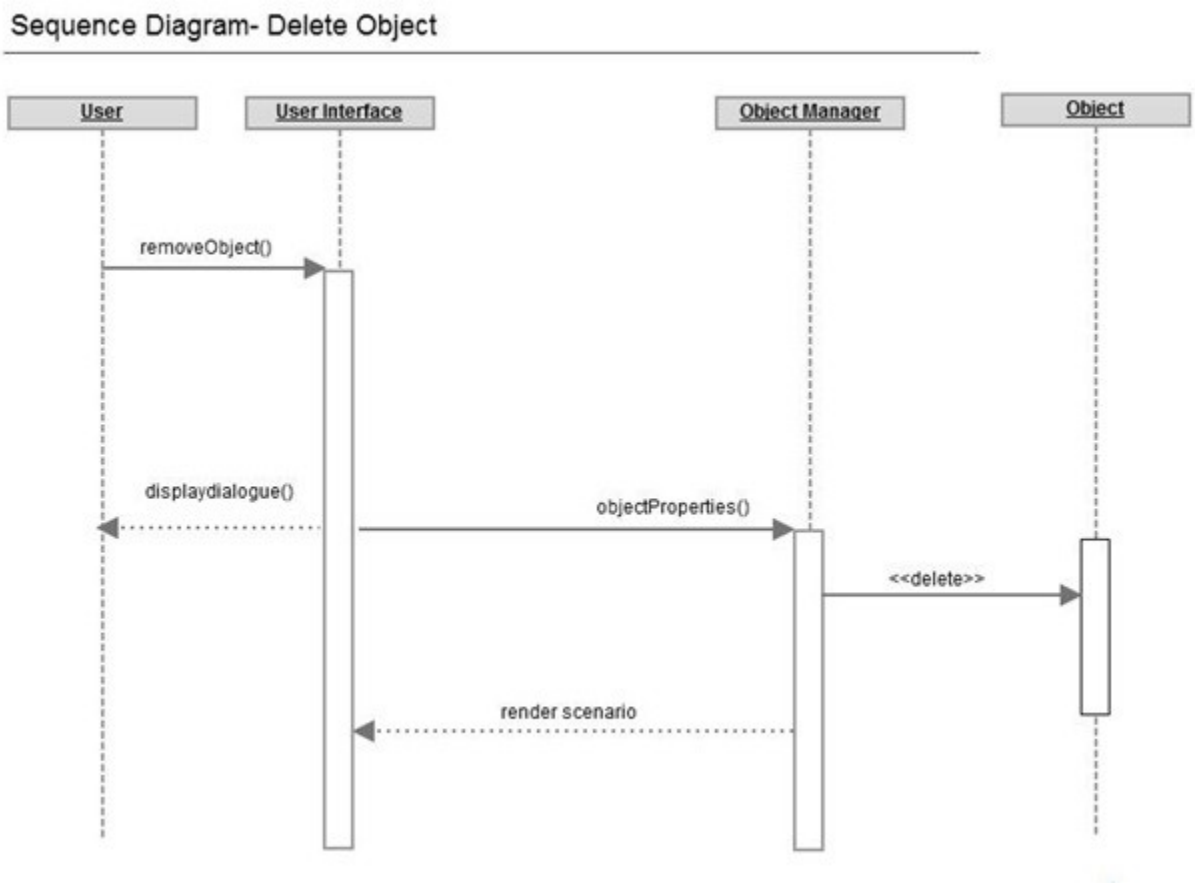
This diagram describes the methods and events taken when the user wishes to stop the simulation.

Delete Relationship



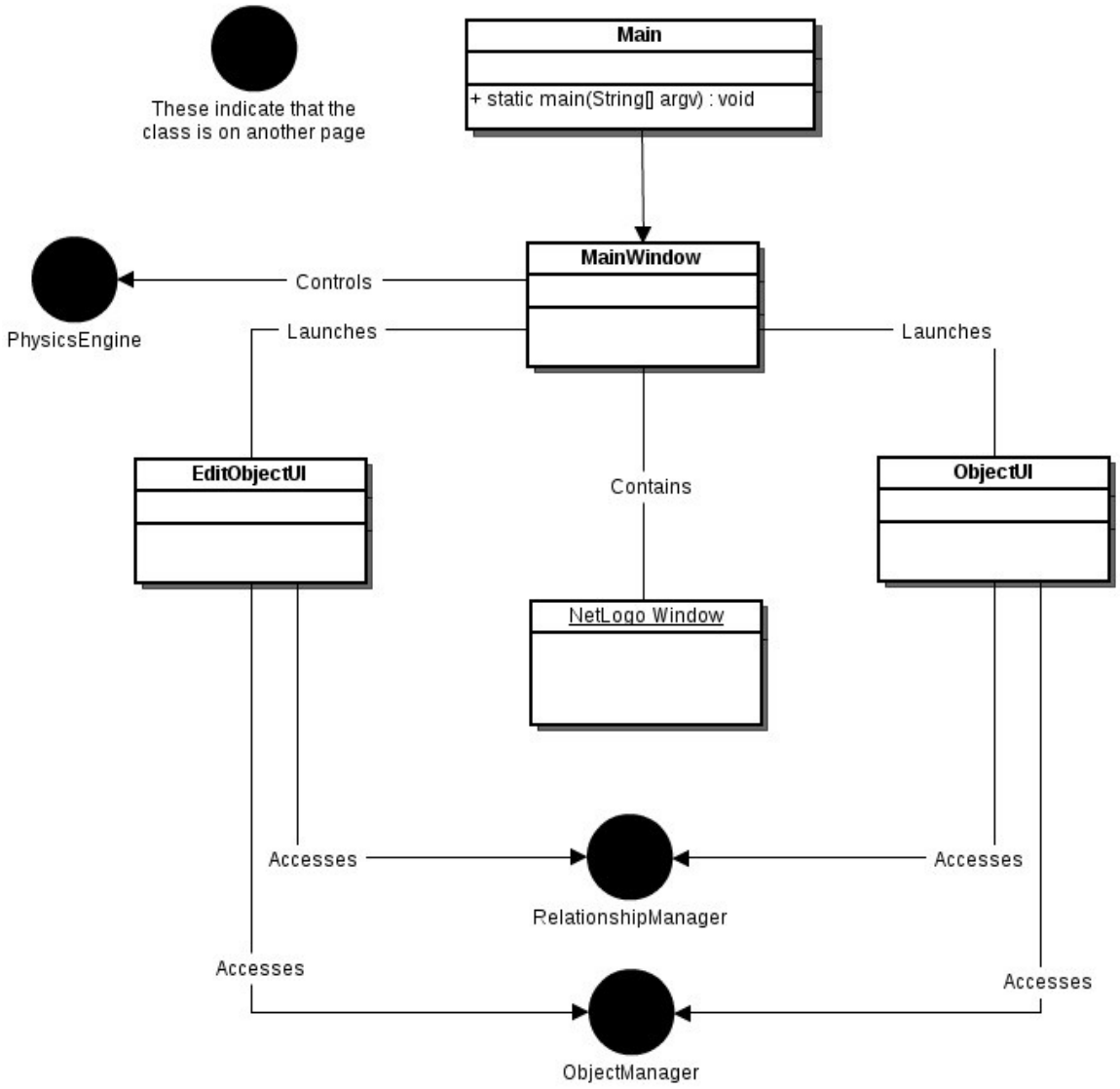
This diagram describes the methods and events taken when the user wishes to delete a relationship.

Delete Object

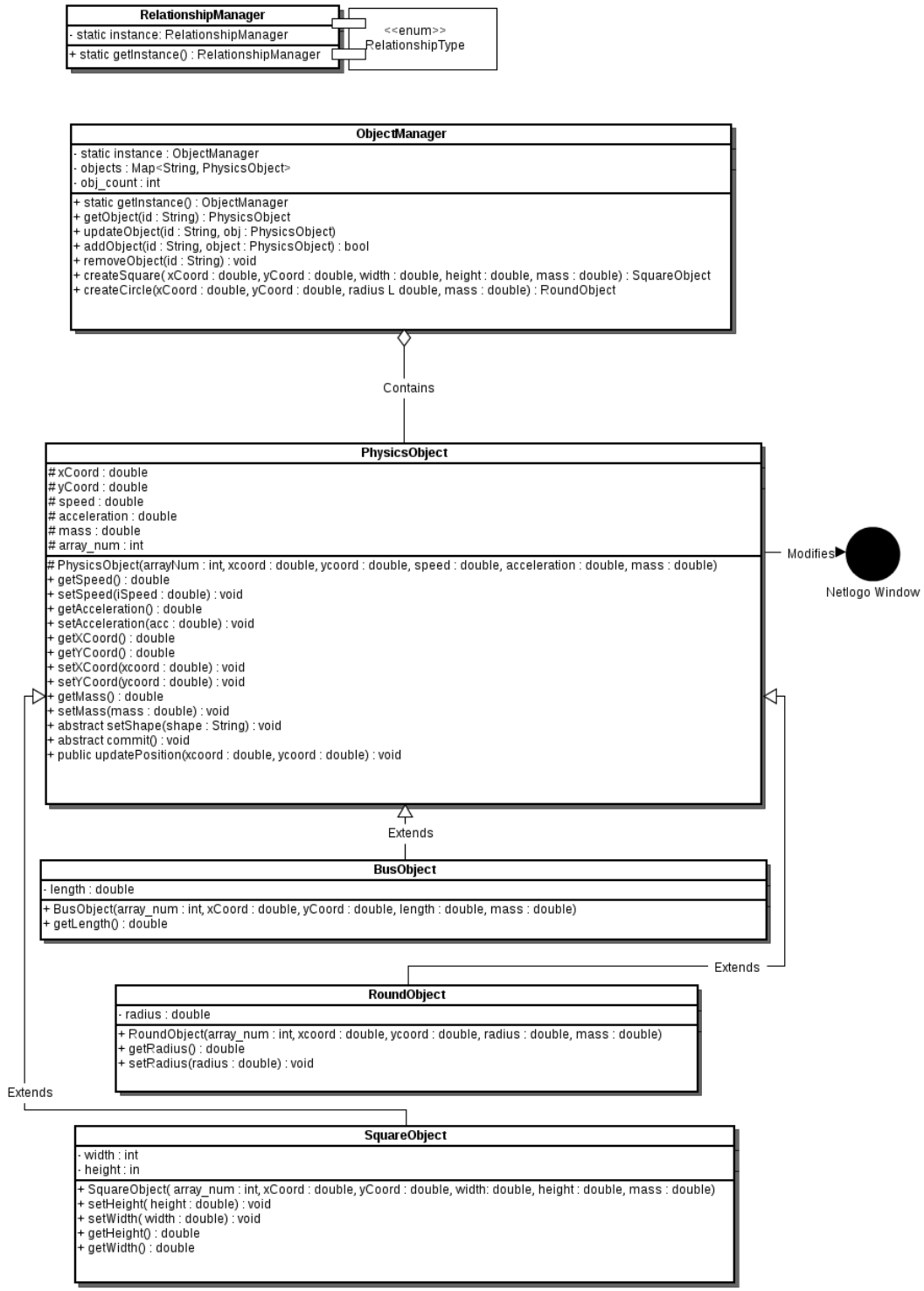


This diagram describes the methods and events taken when the user wishes to delete an object.

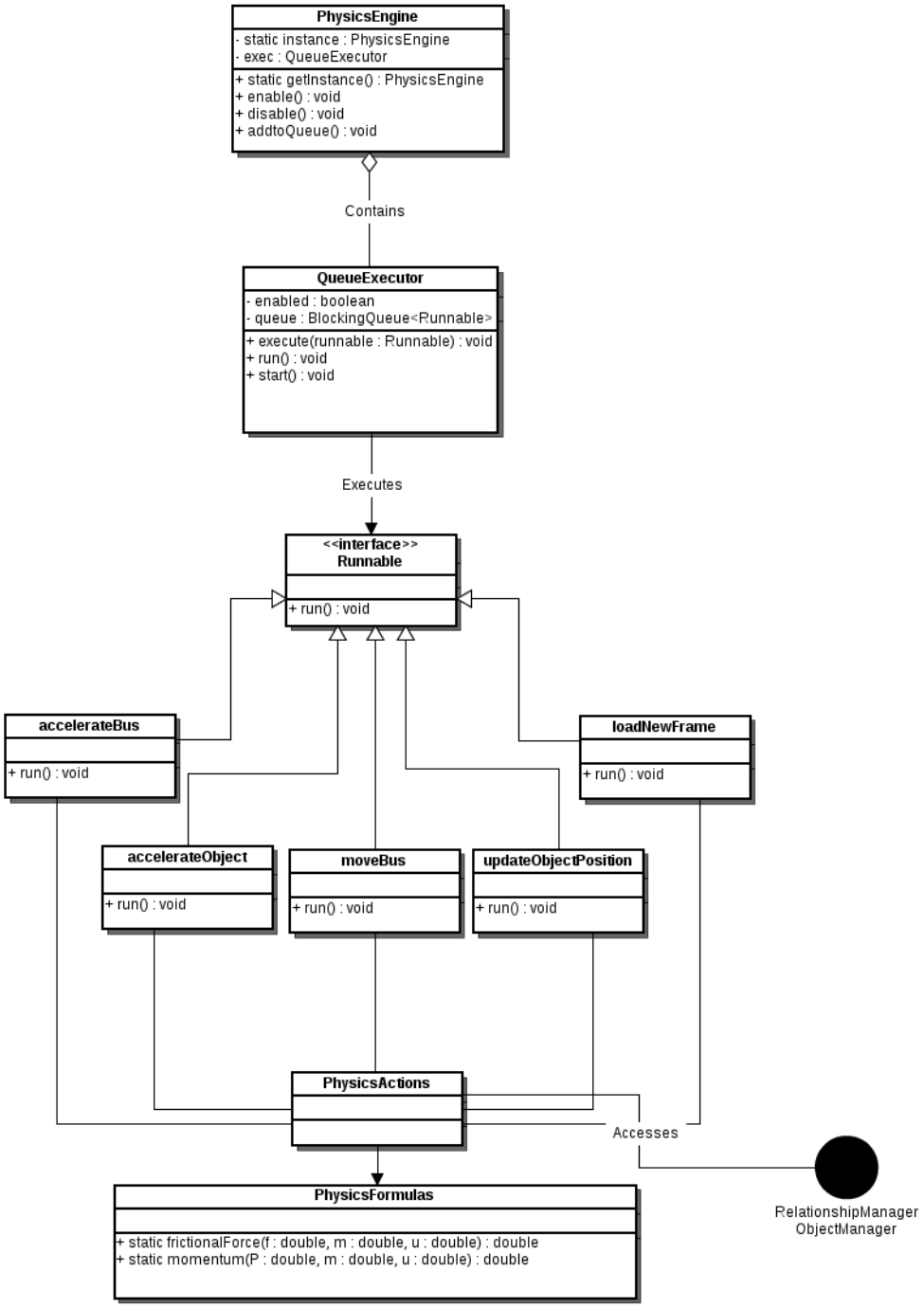
Class Diagram- Main and User Interfaces



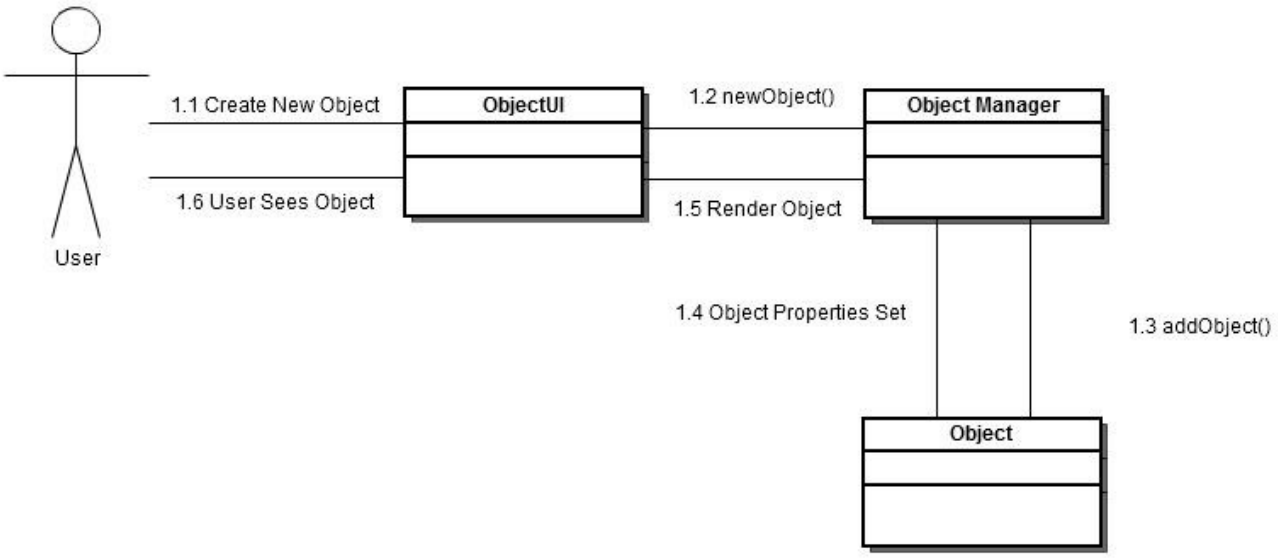
Managers Class



Physics Classes



Appendix IV: Communication Diagram



This diagram describes the methods and communications used to create an object.