Solar System Simulator Proposal

1. My app will be designed to run and animate a realistic 3-D simulation of the motion of objects in our solar system.

2. The target audience will be anyone who is as fascinated by space, physics, and simulation as I am.

3. The data used will primarily come from NASA's JPL HORIZONS system, which allows users to obtain orbital parameters, positions, and velocities for any known object in the solar system at a specified date and time. The Python module astroquery.jplhorizons is an API wrapper module which provides convenient programmatic access to the JPL HORIZONS database.

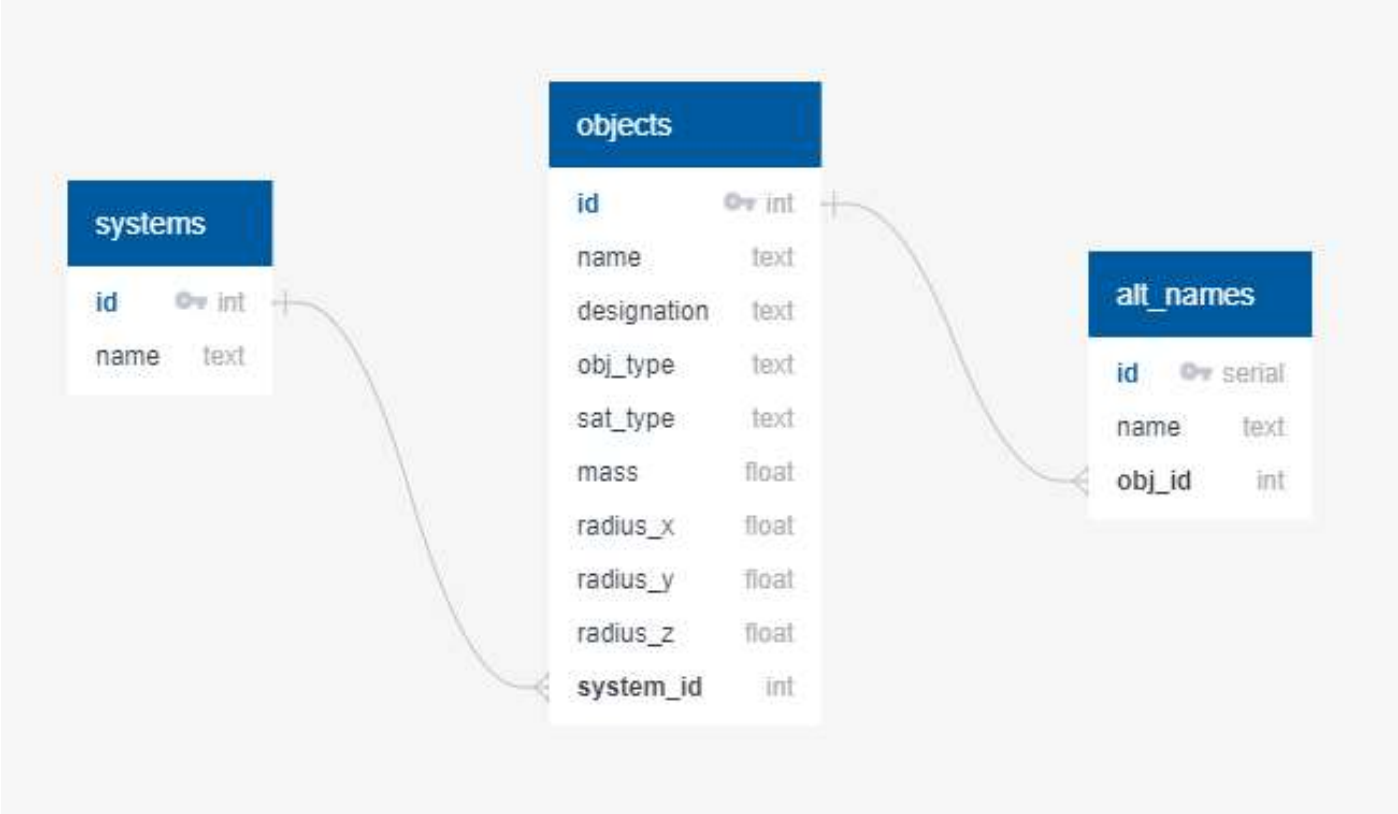
   One important piece of data that is not available through this module is the mass of the objects. This data is necessary for the simulation but is not stored in a consistent way across all desired objects in the NASA database. It is however available on Wikipedia, so my planned alternative is to build my own SQL database that will contain this information, along with the names and ID numbers of each object.

4. The basic outline of how the app will work is as follows:

   a) User selects a date and time.
   b) Server gets mass data for all objects to be simulated from the SQL database.
   c) Server queries JPL HORIZONS for positions and velocities of objects to be simulated at the specified date and time.
   d) Positions and velocities are used as initial conditions in an iterative simulation using Newton's law of universal gravitation.
   e) Motion of objects is animated on screen using Three.js or a similar tool.
   f) User will be able to adjust the speed of the simulation as well as move the camera around the environment to view the simulation from different positions and angles.

   One potential problem to solve is the number of calculations involved in the simulation in combination with the size of the time step to be used. A smaller time step allows for a more accurate simulation with less error buildup over many iterations, but will also require more calculations for a given time interval and may lead to performance issues depending on the number of objects involved in the simulation. For example, a time step of 1 hour would require 24 complete iterations through the simulation loop to advance by 1 day, while a time step of 12 hours would require only 2 iterations. The 1-hour time step would allow for a more accurate simulation, but would require 12 times as many calculations as the 12-hour time step. The difference between a 1-hour and 12-hour time step is not important for objects with relatively long orbital periods such as all of the planets, but would present serious accuracy problems when trying to simulate objects with much shorter orbital periods (for example Jupiter's moon Metis, which completes an orbit of Jupiter in less than 8 hours). To solve this problem, I will need to either find ways to reduce the numbers of calculations involved in order to maintain performance, or limit myself to simulating objects with a relatively narrow range of orbital periods.

The schema for my planned SQL database is relatively straightforward:



Systems table: Represents a planet along with all of its moons.
id: Integer which corresponds to an ID in the NASA database. Mercury is 1, Venus is 2, Earth is 3, and so on.
name: Name of the system.

Objects table: Contains individual planets and moons.
id: Integer which corresponds to an ID in the NASA database. Planets are identified with their system ID followed by 99 (Mercury is 199, Venus is 299, Earth is 399, and so on). Moons are identified by their system ID followed by 2 digits indicating the order in which they appear in the database (Earth's moon is 301, Mars' moons Phobos and Deimos are 401 and 402, Jupiter's moons Io, Europa, Ganymede, and Callisto are 501 through 504, and so on). The Sun's ID number is 10.
name: Name of the object.
designation: Many moons have designations which indicate the year and order in which they were discovered. Nullable since not all objects have designations.
obj_type: Specifies the type of object (sun, planet, moon, dwarf planet, etc)
sat_type: Moons of gas giants are sometimes grouped according to their orbital characteristics. Nullable since planets and some moons do not have a group.
mass: Mass of the object.
radius_x, radius_y, radius_z: Floating point numbers to represent the physical size of an object for use in the animation.
system_id: Identifies the system the object belongs to.

Alt_names table: Many objects are sometimes known by alternate names or designations, which will be represented in this table.
id: Incrementing primary key.
name: Alternative name or designation for the given object.
obj_id: Identifies the object that the alternative name belongs to.