Solar system simulator

Miska Tammenpää

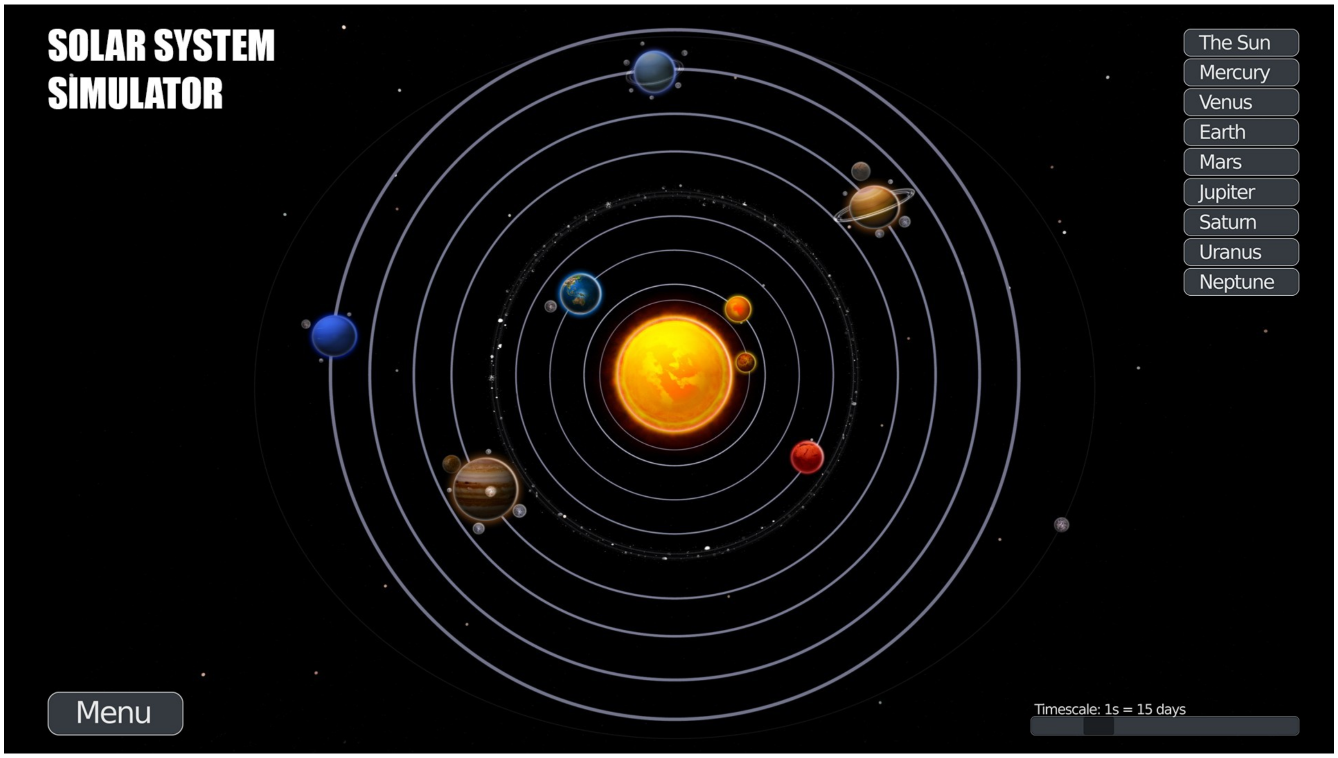
729637

Tietotekniikan kandidaattiohjelma
1st year

13.2.2020

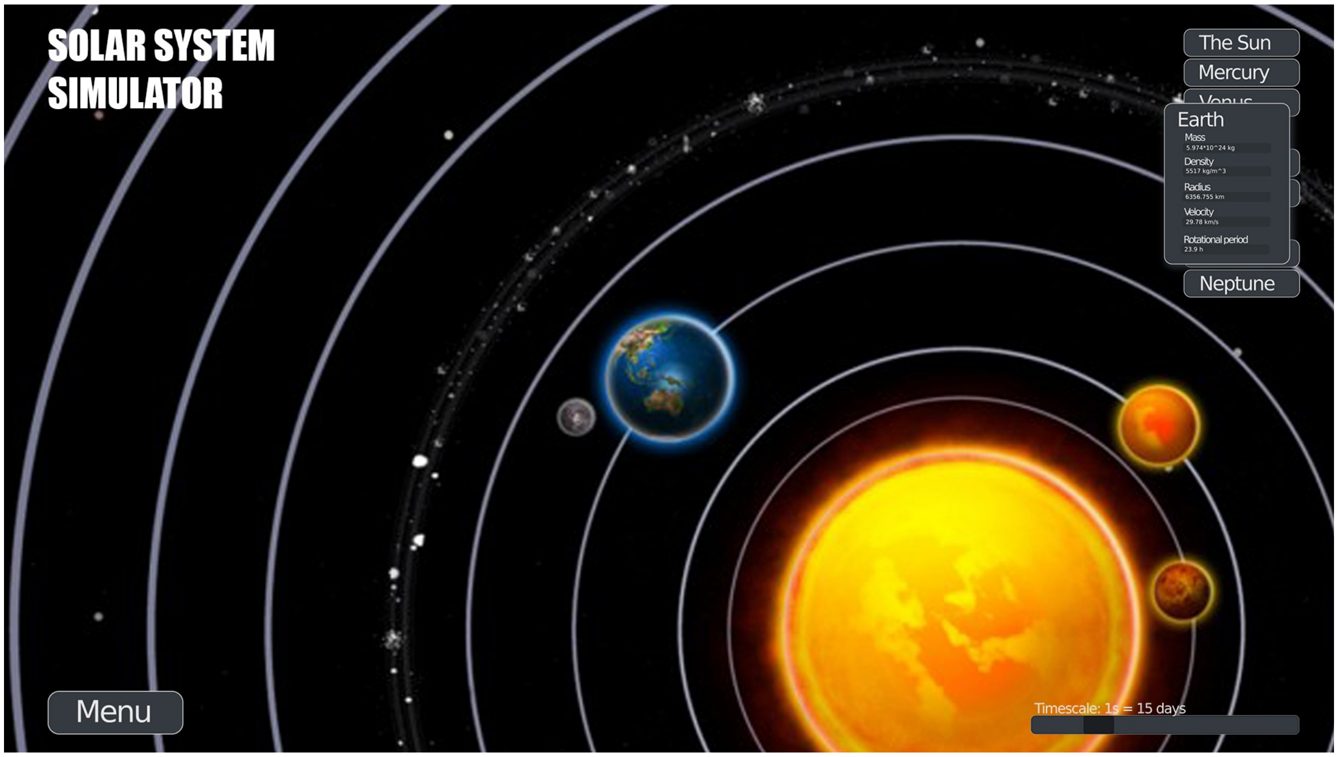
The idea of the project is to create a program that simulates the movements of different bodies in a solar system. The objects will be represented in a graphical user interface and the user will be able to create new objects and change their attributes. The simulation will then display how gravitation and Newton's laws act on large bodies over time.

As well as changing the attributes of the objects, the user can change the timescale of the simulation to see longer term effects caused by gravitation.



(The scale of the bodies and their orbits in this draft is irrepresentative of what it will be in practice)

Selecting an object from the object menu on the right will display the object’s current information and let the user change its values. Object selection will also highlight and/or zoom in closer on its visual representation.



Earth

Mass
5.974*10^24 kg

Density
5517 kg/m^3

Radius
6356.755 km

Velocity
29.78 km/s

Rotational period
23.9 h

Timescale: 1s = 15 days

The aim in the project is to meet the requirements for a “Demanding” difficulty project with a focus on the fluency of interaction between the user and the simulator.

Under the menu button, the user will be able to create new solar systems from scratch (or from templates) and save or load solar systems to or from text files. The data stored in the files will be formatted roughly as follows:

```
#objectname
objectdata
-||-
-||-

#otherobjectname
objectdata
```

etc.

The simulation will then be able to collect all the objects and their information from these files.

A key element to test in the simulator is the values given as parameters to the objects. Not only should the simulator not accept wrong kinds of inputs (letters in place of numbers, mass of zero, faster than light movement etc.) it should also not accept values that its not equipped to handle. For example giving the radius of a planet the same radius as the solar system and calculating the resulting behaviour afterwards is out of the scope of this project.

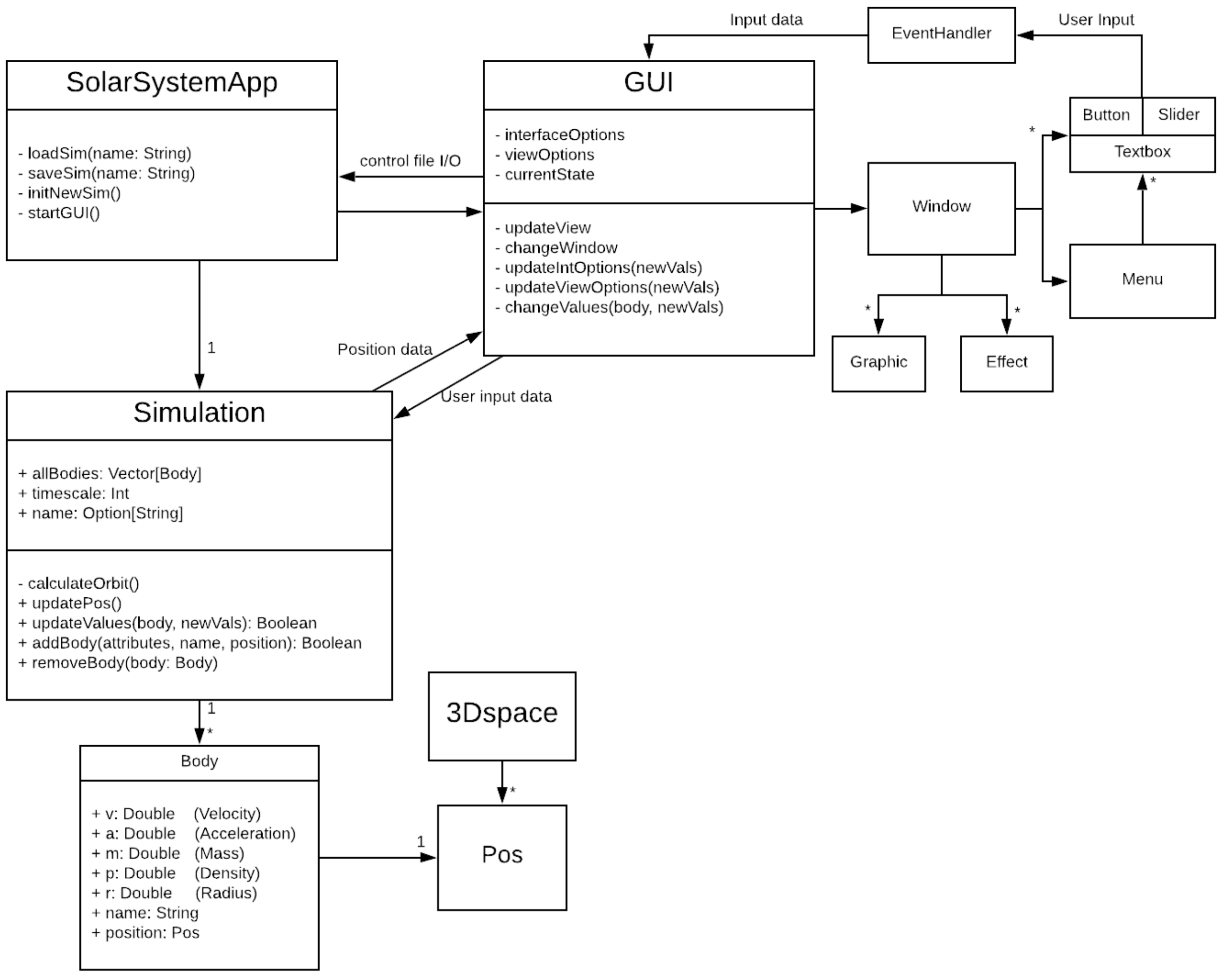
Another thing to test for is the simulator’s performance. With many variables and long equations in play it will be crucial to know what kind of workload the simulator can take on. Knowing the simulator’s limits, proper thresholds can be set for the objects’ parameters.

The physical accuracy of the simulator can be tested as well. With a certain starting point for the simulation, the objects’ values can be compared against known data of objects in space. This way, the simulator’s accuracy can be honed down to a certain error threshold.

Stripped down to its core, a solar system simulator is a calculator for very specific equations. The other features, such as the graphical user interface, will be built around this. A GUI object will be rapidly updating a window instance by drawing a graphical representation of a solar system with UI elements like menus and buttons on top of it. The GUI only needs the coordinates of the simulation’s bodies in a cartesian 3D-space. It will query them from a “Simulation” object, which will take care of the calculation. It does this by reading information from instances of a “Body” class and computing the positions of the bodies at any given moment. Really, the bodies here are just data structures since they don't have any functionality to themselves.

The user will interact with the simulator via instances of buttons, sliders and text boxes embedded into the UI. An event handler will then relay the data to the GUI, which will pass it forward where it needs to go. Generally, the cases of interaction will be changing the values of the instances of "Body" or creating and deleting the instances. Also they'll be used for changing GUI settings and initiating file I/O to save or load a state of a simulation.

A "SolarSystemApp" object will then tie all of this together by initiating the GUI and the simulation and handling the input and output of files.


Input data
EventHandler
User Input
SolarSystemApp
- loadSim(name: String)
- saveSim(name: String)
- initNewSim()
- startGUI()
control file I/O
GUI
- interfaceOptions
- viewOptions
- currentState
- updateView
- changeWindow
- updateIntOptions(newVals)
- updateViewOptions(newVals)
- changeValues(body, newVals)
Button
Slider
Textbox
Window
Menu
Graphic
Effect
Position data
User input data
Simulation
+ allBodies: Vector[Body]
+ timescale: Int
+ name: Option[String]
- calculateOrbit()
+ updatePos()
+ updateValues(body, newVals): Boolean
+ addBody(attributes, name, position): Boolean
+ removeBody(body: Body)
Body
+ v: Double (Velocity)
+ a: Double (Acceleration)
+ m: Double (Mass)
+ p: Double (Density)
+ r: Double (Radius)
+ name: String
+ position: Pos
3Dspace
Pos
1
*