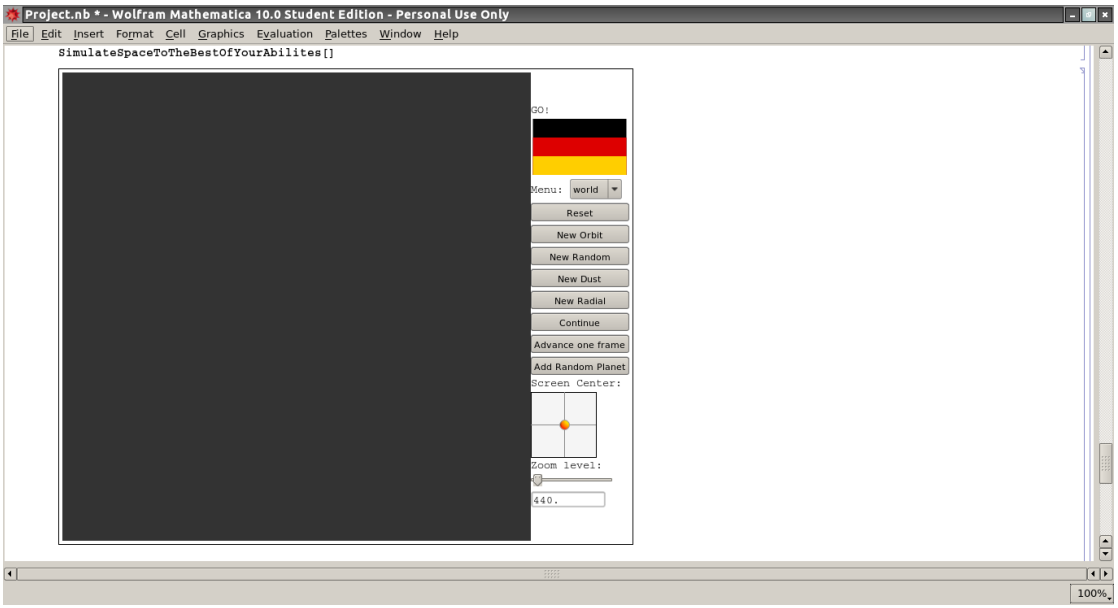
Niven Achenjang
Luke Miles
2015-04-27
CPS Progress Report 2

1 -- how to run the code

Open QuadTree.nb and evaluate notebook. From that same mathematica window, go to File-Open and browse to Project.nb and evaluate notebook again. It is important that these two evaluations are done in the same kernel session. Once you have evaluated QuadTree.nb and then Project.nb, scroll to the bottom of the Project.nb window and you should see something like this:



The default screen should have the “world” Menu selected. If it does not then move to it.
Hit “New Orbit”. You should see some planets orbiting. “Pause” the simulation. Other menus do not work unless it is paused. Go to the object Menu and make a big orange gaseus planet with a high speed. Place it such that it will crash through both planets and the sun. Go back to the world menu and “Continue”. Hooray!

Now on the world menu and hit "New Radial". This makes a bunch of random planets that are spiraling around the center of the screen. Watch it for a bit. Zoom around and drag the "Screen Center" slider. Zoom in closely on one planet. Pause it and advance one frame at a time to watch it closely.

Now hit “New Orbit” again. “Pause” the simulation. Go to the cursor menu. Hit the “Edit Cursor” button. Select an object you want to edit. Go to the object menu and change its properties. Click on another planet and adjust its properties. Repeat until you get tired of it. Go back to the cursor menu and hit “Deletion Cursor”. Click on the sun! As the planets fly away, kill them just before they leave the screen! You are Cuta, Destroyer of Worlds!

2 -- updated project proposal

short project description

Our project is a 2D space simulator.
This consists of a region of space with interacting celestial bodies (planets, stars, etc).

The user has the following powers:

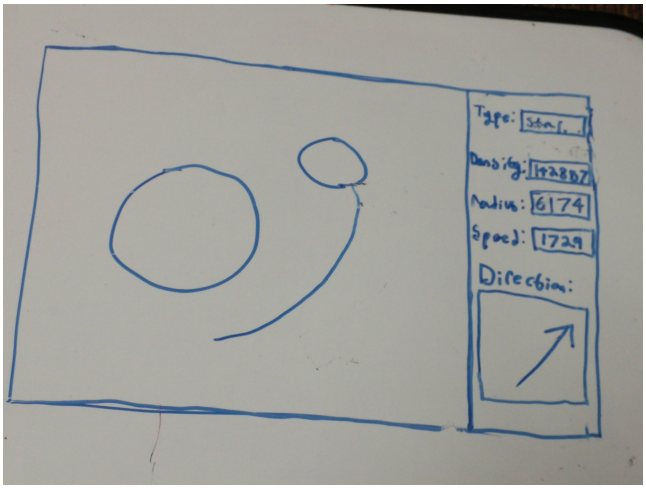
- place objects in space with initial properties such as size, mass, and velocity
- move and delete said objects
- pause and speed up time

As objects are placed, the system evolves according to the laws of physics.
The objects are attracted to each other by gravity and collide in different ways.

Description of Final Product

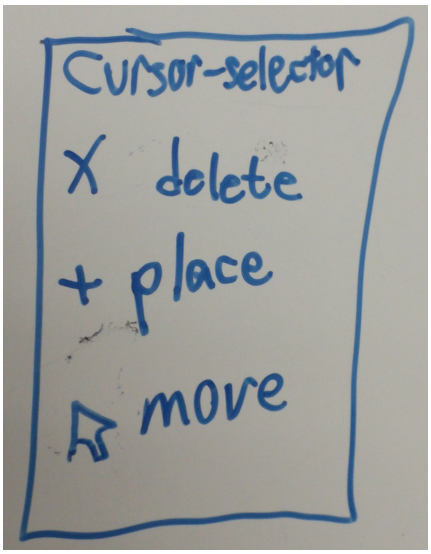
The user opens up the program and is presented with a blank canvas or a premade example.
There is an object creation menu on the right, with the following elements:

- sliders for density, radius, and speed of object
- a choice for the type of object (particle, asteroid, planet, star, black hole)
  This affects how the object is displayed,
  how the object collides (star eats planet, but an asteroid and another asteroid bounce, etc...),
  and what the bounds are on the property sliders (Ex. can not have a particle with same radius as a star)
- an arrow (that the user can point) for the object's direction
- A color picker of some sort



There are multiple kinds of cursors:

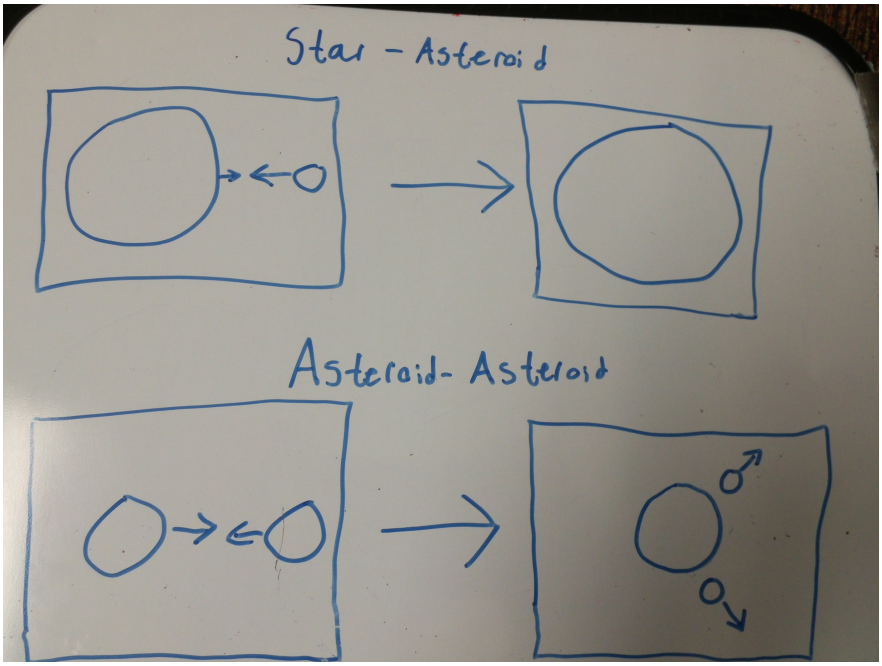
- object-creation cursor: click to place object
- kill cursor: deletes object that are clicked on
- moving cursor: click and drag objects to throw them around



Say that the user makes 2 objects.
She first selects the object-creation cursor, then configures and places each object.
The first is a very massive star with 0 speed.
The second is a miniscule planet with a velocity tangent to its future orbit around the star.

Depending on how the parameters are defined, the orbit may or may not be stable.
Suppose in this instance it is not stable.
The planet goes around a few times, but is eventually absorbed by the star,
making a small change to the star's mass and velocity.

This was a star-planet interaction.
If this were a planet-planet collision, the collision would probably break the objects apart.
If this were an asteroid-planet interaction, the asteroid would be absorbed into the planet.
If this were a particle-particle interaction, they might stick or bounce apart.



Another major function is the time multiplier.
This allows the user to change the rate of time flow to a natural multiple of the base time flow.
It could be 0x (paused), 1x (normal), 2x (twice speed), 3x, ...
For example, if the user has a stable-looking orbit that she wants to see the long-term development of, she can put things into 100x and see the years fly by.

THOUGHTS FOR LATER

- select an object and adjust its properties.
- save states
- relate project to Germany
- trace paths of objects
- allow user to change rules of universe (e.g. the constant G)

increments

- increment 1:
  Simple masses are floating around and being attracted and such. The user has no power, no collisions.
- increment 2:
  The user can place masses and change the rate of time. Collisions still don't work.
- increment 3:
  The 3 cursors (object-creation, kill, move) are all implemented and basically working.
- increment 4:
  Get object creation menu working smoothly with all options functional.
- increment 5:
  Get collisions working smoothly.
- increment 6:
  Implement thoughts for later as time allows

3 -- schedule of future work

04-27: finish implementing non gaseous-gaseous collisions
05-01: get time scaler working nice
05-04: comment all code
05-08: cleanup work

4 -- explanation of algorithms

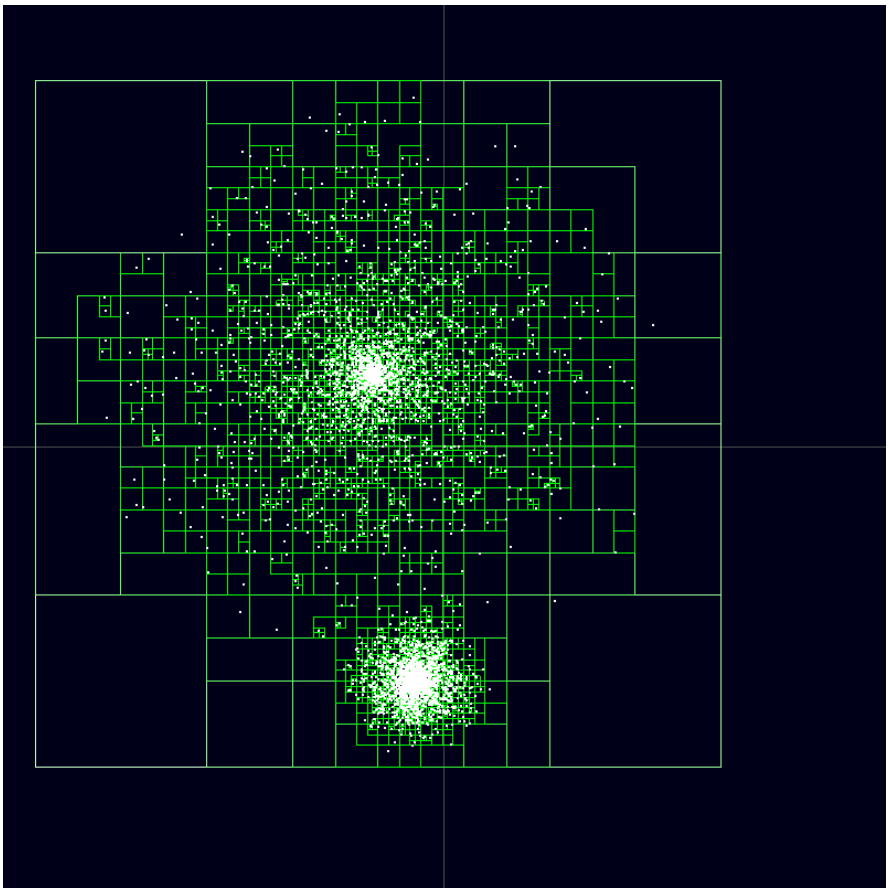
First, gravity is what causes all non-collision acceleration.
Since $F_{g,m_1} = \sum \frac{G m_1 m_2}{r^2} \text{ for } m_2 \text{ in the universe} = G m_1 \left(\sum \frac{m_2}{r^2} \text{ for } m_2 \text{ in the universe} \right)$ and
$F_{m_1} = m_1 a_{m_1}$, $a_{m_1} = G\left(\sum \frac{m_2}{r^2} \text{ for } m_2 \text{ in the universe} \right)$
And since $a = \frac{\Delta v}{\Delta t}$, $\Delta v = \Delta t \, G\left(\sum \frac{m_2}{r^2} \text{ for } m_2 \text{ in the universe} \right)$. Each frame, every planet's velocity is changed according to that formula. And, every planet's velocity is added to it's position. This is what causes orbits and weaving paths and so on. These equations are available in any physics textbook or physics website. An example was not given as I thought it was clearer in the abstract.

When you have a lot of objects, it can be expensive to calculate F_g for all of them. So QuadTree.nb applies an algorithm to make this computation much faster. It uses a Barnes-Hut Simulation

(https://en.wikipedia.org/wiki/Barnes%E2%80%93Hut_simulation). Essentially, space is dynamically split into four squares (each of which can then be split into four more squares) based on the distribution of circs. When calculating gravity for a particular object, if a square is suffiently far from it, gravity will be approximated by assuming all the circs in the square are really one circ whose mass is the sum of their masses, and whose position is their center of mass.

In[2]:=
```
Import ["https://upload.wikimedia.org/wikipedia/commons/f/f8/Barnes_hut_tree.png"]
```

Out[2]=



5 -- necessary files and such

The only constraint is that you run QuadTree.nb, then Project.nb in the same notebook. QuadTree.nb is seperate because what it does is kind of different and seperate from most of the stuff in Project.nb. There is no real technical reason it has to be a seperate file.

6 -- major data structures

In any session where planets are made, the planets are stored as associations that have keys like “rad” for radius, “pos” for position, and “vel” for velocity. This is one major data structure that is used everywhere in the code. The second major data structure is the QuadTree. It is a recursively defined association list. Its keys are “chlds” which stores either a list of children QuadTrees or a list of circs contained with in tree, “crnrs” which stores the location of the lower left and upper right points on the tree’s square, “CofM” which stores the center of mass of the tree, and “mas” which stores the total mass of the tree.

7 -- overview and major functions

The code works by having a complicated system of menus and things to get everything interacting and displaying, then a simple underlying physics.
The gravity works as explained in section 4. (I know it says you should only have to read section 2, but it really fit here.)
Some major functions:

- makeCirc -- final: This takes in some properties of an object (like radius, position, velocity, ...) and returns an association, which is how the object will be kept from then on.
- updateAssoc -- final: This takes two associations and returns an association made by adding all of the key-value pairs of the second to the first. Any keys that already exist are overwritten.
- resolveCollisions -- needs work: This takes the list of planets and returns a new list after all collisions have been applied. Currently, it only makes gaseous-gaseous planets combine. We still have to implement solid-gasueous and so on.
- SimulateSpaceToTheBestOfYourAbilites -- near final: This takes no parameters. It calls everything and makes all the menus and draws ita ll and displays it to the user and updates the frame every .05 (or so) seconds. Very important.
- gravitationalForce -- final: This takes in a quadtree, circ, and a parameter θ necessary for the Barnes-Hut Simulation. This recursively traverses the quadtree and applies gravity from every object in the tree to circ, either directly or using an approximation.

8 -- accomplishments so far

Everything described in section 1, essentially. Orbits, the three cursors, an object menu, ...

9 -- signed statements

Luke's:
I have contributed significantly to this project. I made a large part of the interactive menus and worked on the physics/computation in the background. My partner has done the same. Niven has worked quite hard on getting QuadTree.nb working and at least as much as I have on the GUI.

Niven's:
I have worked a lot on this project. I did most of the stuff with getting QuadTree.nb to work, some of earlier stuff with collisions, a bit of the GUI, etc. I intend to try to make the project faster and more efficient, and to comment the code at some point.

10 -- progress report 1 goes below here