Zack: Universe and Star

Dan: SpaceObject and Body

We'll both work together on the extra credit. For extra credit we will be making a black hole appear at click location and it will slowly pull and consume bodies.

We are going to do our window loop in main and not in the Universe class.

Unit tests:

- Step function
- Translation from universe coordinates to SFML coordinates
- Test that the velocities of the bodies are calculated correctly based on Newton's gravitational law.
- Test that simulation ends when amount of time given through command line argument has passed
- Confirm that setters and getters function correctly

```
 1  /**
 2   * @file space.hpp
 3   * @copyright 2016
 4   * @author Zachary Krausman and Daniel MacMillan
 5   * @date 4/22/16
 6   * @version 1
 7   *
 8   * @brief header class for final project
 9   *
10   */
11
12  #ifndef SPACE_H
13  #define SPACE_H
14
15  #include <SFML/Graphics.hpp>
16  #include <vector>
17  #include <string>
18  #include <iostream>
19
20
21  namespace space {
22  /****************************** SPACEOBJECT
******************************/
23  /**
24   * SpaceObject class which all the space objects inherit from.
25   * Holds position, velocity, and mass data as well as
26   */
27  class SpaceObject: public sf::drawable {
28   public:
29      /**
30       * Default constructor.
31       * @param N/A
32       * @return N/A
33       */
34      SpaceObject();
35      /**
36       * Constructor with parameters to set variables.
37       * @param double velocity, double mass, double locationX, double
locationY
38       * @return N/A
39       */
40      SpaceObject(double velocity, double mass, double locX, double locY);
41      /**
42       * Destructor. Doesn't do anything at the moment.
43       * @param N/A
44       * @return N/A
45       */
46      ~SpaceObject();
47      /**
48       * setter for velocity
49       * @param N/A
50       * @return N/A
51       */
52      void setVelocity(double velocity);
53      /**
54       * setter for mass
55       * @param double velocity
```

```
56        * @return N/A
57        */
58       void setMass(double mass);
59       /**
60        * setter for locationX
61        * @param double mass
62        * @return N/A
63        */
64       void setlocationX(double locationX);
65       /**
66        * setter for locationY
67        * @param double locationX
68        * @return N/A
69        */
70       void setlocationY(double locationY);
71       /**
72        * Getter for velocity
73        * @param N/A
74        * @return double
75        */
76       double getVelocity() const;
77       /**
78        * Getter for mass
79        * @param N/A
80        * @return double
81        */
82       double getMass() const;
83       /**
84        * Getter for locationX
85        * @param N/A
86        * @return double
87        */
88       double getlocationX() const;
89       /**
90        * Getter for locationY
91        * @param N/A
92        * @return double
93        */
94       double getlocationY() const;
95
96    private:
97       double velocity_;     //< Movement speed. Updates based on force.
98       const double mass_;  //< Mass for calculating force. Does not
update.
99       double locationX_;         //< X Location relative to the SFML
window.
100      double locationY_;         //< Y Location relative to the SFML
window.
101      /**
102       * draw function overwritten from the drawable class.
103       * @param sf::RenderTarget& target, sf::RenderStates states
104       * @return N/A
105       */
106      virtual void draw(sf::RenderTarget& target, sf::RenderStates states)
const;
107   };
108
```

```
109  /****************************** BODY
*******************************/
110  /**
111   * Represents moving bodies in the universe, including the sun and the
planets
112   * Handles taking in parameter data to initialize SpaceObject variables
113   * Handles movement within the sfml window, does so in increments of
seconds
114   */
115  class Body: public SpaceObject {
116   public:
117      /**
118       * Default constructor for the body, will call the constructor for
119       * SpaceObject, and will use overloaded insertion operator to get
info from
120       * file.
121       */
122      Body();
123
124      /**
125       * Insertion operator will now take info from file and initialize
variables
126       * with it.
127       * @param istream & in, const Body & b
128       * @return istream &
129       */
130      friend istream & operator >> (istream & in, const Body & b);
131
132      /**
133       * Takes a time paramter (double seconds) and moves the Body object
given its
134       * internal velocity for that much time
135       * @param double seconds
136       * @return N/A
137       */
138      void step(double seconds);
139
140   private:
141      sf::Texture texture_;    //< Texutre from the image of a solar body
142      sf::Sprite sprite_;      //< Will hold the texture and move across
screen
143  };
144
145
146  //!
147  //! A star class
148  //!
149  class Star : public SpaceObject {
150   public:
151      //! \brief Star constructor
152      //!
153      //! \param  x x-coordinate of star
154      //! \param  y y-coordinate of star
155      //! \param  mass mass of star
156      //! \return     none
157      //!
158      //! Constructs a star object
```

```
159       Star(double x, double y, double mass);
160
161     private:
162       double diameter_;  //< diameter of the star
163   }
164   //!
165   //! A universe class that holds all the Star objects
166   //! and SpaceObjects and handles updating the universe
167   //!
168   class Universe {
169     public:
170       //! \brief Universe constructor
171       //!
172       //! \param total_time total time the simulation should run for
173       //! \param change_time length of each tick
174       //! \return none
175       //!
176       //! Constructs a universe object
177       Universe(double total_time, double change_time);
178       //! \brief calculates forces function
179       //! \return none
180       //!
181       //! calculates the forces on and updates the velocity of each body
182       void calcForces();
183       //! \brief trandlate coordinates function
184       //! \return none
185       //!
186       //! changes from coordinate system in file to SFML system
187       void translateCoordinates();
188       //! \brief monitor time function
189       //! \return none
190       //!
191       //! prints the time to screen and closes the simulation
192       //! if the amount of time specified in the command line has passed
193       void monitorTime();
194       //! \brief updates the universe file with final simulation state
195       //! \return none
196       //!
197       //! saves the final state of the universe to a file
198       void updateUniverse();
199
200     private:
201       double elapsed_time_;  //< how much time has passed since the start
202       double total_time_;  //< amount of time the simulation should run
for
203       double change_time_;  //< amount of time in each tick of the
simulation
204       vector<Star> stars_;  //< vector of Star objects
205       vector<SpaceObject> planets_;  //< vector of SpaceObjects
206   }
207
208   }  // namespace space
209
210   #endif  // SPACE_H
```