FINAL PROJECT REPORT

Daniel Santos & Hung Nguyen

Brief

The idea of this project is to simulate part of the Sol system. This will be done by using SFML for the graphics and the Newton's law of universal gravitation states. The Newton's idea is that "any two bodies in the universe attract each other with a force...". This simulation will calculate the attraction between the different bodies in the system, and with that idea the orbits of the planets will be simualted.

Work distribution

The work distribution was decided by both, Daniel and Hung.

Hung Nguyen:

- Body class -> Definition and Implementation
- Unittest for SpaceObject and Star classes

Daniel Santos

- SpaceObject and Star -> Definition and Implementation
- Unittest for Body class.

Together

- Universe class -> Definition and Implemenation
  The way we will do this is by meeting in person to disccuss about the implementation and writing it a the same time.
- Unittest for the Universe class

Extra Credit Requests:

- Making a spaceship moving around the universe, which can be controlled by the user using the arrows keys.
- Make the planets clickable, so when the user clicks a dialog will display information about this planet like velocity, name, and acceleration.
- This project will be worked using a private repository in Github.com.

Simulation Explanation

Our simulation will be in the run() function of the Universe class. There we will have the while loop of for the SFML library. The client will only have to open the file and pass it to the Universe when it is instanciated and call the run() function.

Unittest

Author: Daniel Santos

I will be implementing the unittest for the Body class and part of the Universe class. For the Universe::Body I will test that when a time is passed to the step function the correct Net Force and Acceleration are returned by those properties' getters. Also that when a time is passed to calVelocity, it returns the correct velocity based on the mass of the planet. I will also test the <<operator reads the file correctly to the object.

Author: Hung Nguyen

My unittest are written in the documentation of the the functions. These are SpaceObject, Star and Universe.

Header files

UNIVERSE::SPACEOBJECT

```
/** Copyright 2015 Daniel Santos & Hung Nguyen
 *  @file     Universe.hpp
 *  @author   Daniel Santos & Hung Nguyen
 *  @date     04/20/2016
 *  @version  1.0
 *
 *  @brief    This file contains the namespace
 *  Universe which contains the classes definition
 *  for SpaceObject.
 * */

#ifndef PS4_SPACEOBJECT_HPP
#define PS4_SPACEOBJECT_HPP
```

```
#include <SFML/Graphics.hpp>

/**
 *  @author Daniel Santos & Hung Nguyen
 *
 *  @brief  This is the namespace for the classes
 *  definition.
 * */
namespace Universe {

/**
 *  @author Daniel Santos
 *
 *  @brief  This is an abstract class that inherits
 *  from the sf::Drawable in other make custom drawable
 *  objects in our project.
 * */
class SpaceObject : public sf::Drawable {
 public:
  /**
   *  @brief  This is the destrucotr for the SpaceObject
   *  class.
   * */
  ~SpaceObject();

  /**
   *  @brief  This is a getter for the location vector
   *  of the SpaceObject.
   *
   *  @return const sf::Vector2f&
   *  @test: Hung Q Nguyen
   *  Unit Testing: Compare the position value to a calculated
   *  value at the current time.
   *  BOOT_REQUIRE_EQUAL(calculated value, getLocation())
   * */
  const sf::Vector2f &getLocation() const;
```

```
/**
 *  @brief  This the setter for the location vector
 *  of the SpaceObject.
 *
 *  @param  const sf::Vector2f&
 * */
void setLocation(const sf::Vector2f &location);
/**
 *  @brief  This is a getter for the velocity vector
 *  of the SpaceObject.
 *
 *  @return const sf::Vector2u&
 *  @test: Hung Q Nguyen
 *  Unit Testing: Compare the velocity value to a calculated
 *  value at the current time.
 *  BOOT_REQUIRE_EQUAL(calculated value, getValue())
 * */
const sf::Vector2u &getVelocity() const;

/**
 *  @brief  This the setter for the velocity vector
 *  of the SpaceObject.
 *
 *  @param  const sf::Vector2u&
 * */
void setVelocity(const sf::Vector2u &velocity);

/**
 *  @brief  This is a getter for the mass of the
 *  SpaceObject.
 *
 *  @return double
 *  @test: Hung Q Nguyen
 *  Get mass of a planet -- Check if it is equal to
 *  the expected value or not
```

```
     *  BOOT_REQUIRE_EQUAL(int, getMass())
     * */
    double getMass() const;

    /**
     *  @brief  This the setter for the mass of the
     *  SpaceObject.
     *
     *  @param  double
     *  @test: Hung Q Nguyen
     *  Unit Testing: to check if the SpaceObject will throw
     *  an exeption if it took a negative mass or not mass <= 0
     *  BOOT_REQUIRE_THROW(earth.setMass(-6))
     * */
    void setMass(double mass);

protected:
    /**
     *  @brief  This is an empty constructor for the
     *  SpaceObject. This will be used if the client
     *  doesn't have all the property of the object
     *  at the intansiation time.
     *  @test: Hung Q Nguyen
     *  Unit Testing: There are multiple ways for testing this
     *  object by using this object to call the existed functions
     *  There are no unique properties for the default constructor
     *  to test.
     * */
    SpaceObject();

    /**
     *  @brief This is a complete constructor for the
     *  SpaceObject.
     *
     *  @param  sf::Vector2f, sf::Vector2u, double
     *  @test: Hung Q Nguyen
```

```
   *  Unit Testing: to check if the SpaceObject will throw
   *  an exeption if it took a negative mass or not mass <= 0
   *  BOOT_REQUIRE_THROW(SpaceObject(location, velocity, -6))
   * */
  SpaceObject(sf::Vector2f location, sf::Vector2u velocity, double
mass);

 private:
  /**
   *  @brief  This is a virtual method inherited from
   *  the sf::Drawable abstract class. SpaceObject does
   *  not implement this method, because this will work
   *  as another abstract class for other object in the
   *  project.
   *
   *  @param  sf::RenderTarget &target, sf::RenderStates states
   * */
  virtual void draw(sf::RenderTarget &target, sf::RenderStates
states)
      const = 0;
  sf::Vector2f position_;  // < This is a 2D float vector
  sf::Vector2u velocity_;  // < This is a 2D unsigned int
  double       mass_;      // < This is the mass of the object
};

}  // namespace Universe

#endif  // PS4_SPACEOBJECT_HPP
```

UNIVERSE::BODY

```
 /** Copyright 2015 Daniel Santos & Hung Nguyen
*  @file     body.hpp
*  @author   Daniel Santos & Hung Nguyen
*  @date     04/20/2016
*  @version  1.0
```

```
*
*  @brief    This file contains the namespace
*  Universe which contains the classes definition
*  for Body
* */

#include <iostream>
#include <vector>
#include <SFML/Graphics.hpp>
#include <SFML/Windows.hpp>
#include <SFML/System.hpp>
#include <SFML/Audio.hpp>
#include "Universe.hpp"

namespace Universe {
class Body : public SpaceObject, virtual private NonCopyClass {
public:
    /*  @author: Hung Q Nguyen
     *  @brief: A default constructor of the Planet itself.
     *  A Planet will be inheritanced from the SpaceObject so
     *  It will have the drawable feature by itself.
     *  The job of this function will provide the unique information
     *  of each planet
     *
     *  @params: sf::Vector2f -- The Initial position of the planet
     *           sf::Vector2u -- Velocity of the planet
     *           double  -- the mass of the planet
     */
    Body (sf::Vector2f initial_pos, sf::Vector2u velocity,
      sf::Texture texture, double mass)
      : SpaceObject (initial_pos, velocity, mass),
planet_texture(textures);

    /*
     *   @brief: Destructor of the Body object
     *   @params: none
```

```
     *   @return: none
     */
    ~Body();

    /** @author: Hung Q Nguyen
     *   @brief  Step Function to get the XOR value of the seed with
space position
     *  The step() function will look for the space position on the
initial seed.
     *  After that, it will generate another integer by getting the
result of XOR
     *  operation of two integers, one is the value at tab position,
one is the value
     *  at the end of the current seed.
     *
     *  After that, it will shift the current seed one to the left
and then put the
     *  current generated value to the last position of the seed.
     *   @param  nothing
     *   @return Integer (value of the current generated integer)
     */
    int step(double times);

    /** @author: Hung Q Nguyen
    *   @brief updateSeed funtion will update the seed
    *   Since the step() function is called, the new number
generated and the
    *   vector positions changes, updateSeed() function will
automatically
    *   called in the step() function and create a string based on
the elements
    *   of the vector of vector2f's position
    *
    *   @param  nothing
    *   @return nothing
    */
```

```
    void updatePosition();  // update seed everytime the function is
called

    /** @author: Hung Q Nguyen
    *   @brief toString() method of SpaceObject class
    *   Overloading function will make a cast to SpaceObject class
and treat
    *   LFSR object (constructor) as a string.
    *   Using to display the coordiates at the bottom of window
    *
    *   @param  nothing
    *   @return nothing
    */
    friend std::istream& operator <<(std::istream& input, const
SpaceObject&);

    /*  @author: Hung Q Nguyen
    *   @brief: Calculating the netforce between the planet to the
sun
    *   at the current times
    *
    *   @param: double times -- the current time calculating
    *   @return: sf::Vector2f -- Basically getting the private
variables of the class
    */
    sf::Vector2f calNetforce(double times);


    /*  @author: Hung Q Nguyen
    *   @brief: Calculating the acceleration of the planet at the
curernt time
    *   @param: double times: -- the current time calculating
    *   @return: sf::Vector2u -- Getting the values calculated
    */
    sf::Vector2u calAcceleration(double times, double massgf);
```

```
    /*  @author: Hung Q Nguyen
    *   @brief: Calculating the vector of velocity at the current
time
    *   After Calculating the velocity, it will set the velocity to
    *   the current spaceobject by calling this->setVelocity()
    *   @param: double times: -- the current time calculating
    *   @return: sf::Vector2u -- Getting the values calculated
    */
    sf::Vector2u calVelocity(double times);

    /*************************************
    LIST OF GETTERS AND SETTERS
    *************************************/

    /*  @author: Hung Q Nguyen
    *   @brief: Setting value of times
    *   @params: double -- value of times to assign
    *   @return: void
    **/
    void setTimes(double times) {times_ = times;}

    /*  @author: Hung Q Nguyen
    *   @brief: Getting the value of times
    *   @params: None
    *   @return: double -- Value of times
    */
    double getTimes() {return times_;}

    /*  @author: Hung Q Nguyen
    *   @brief: Setting the texture
    *   @params: texture to set
    *   @return: none
    */
    void setBodyTexture(sf::Texture texture) {pTexture_ = texture;}
```

```
    /*  @author: Hung Q Nguyen
    *   @brief: Getting the Texture of Body
    *   @params: None
    *   @return: Texture
    */
    sf::Texture getBodyTexture() {return pTexture_;}

    /*  @author: Hung Q Nguyen
    *   @brief: Getting the value of netforce
    *   @params: None
    *   @return: sf::Vector2f -- Value of Netforce
    */
    sf::Vector2f getNetForce() {return netForce_;}

    /*  @author: Hung Q Nguyen
    *   @brief: Getting the value of acceleration
    *   @params: None
    *   @return: sf::Vector2f -- Value of acceleration
    */
    sf::Vector2u getAcceleration() {return acceleration_;}

 private:
    double times_;  ///< Counting the time that the planet has moved

    sf::Texture pTexture_;  ///< Texture of the planet
    sf::Vector2f netForce_;  ///< net force between planet and the
Sun
    sf::Vector2u acceleration_;  ///< the acceleration of
                                 ///< the planet the the currentime
};
}
```

Universe::Star

```
/** Copyright 2015 Daniel Santos & Hung Nguyen
 *  @file     Universe.hpp
```

```
 *  @author   Daniel Santos & Hung Nguyen
 *  @date     04/20/2016
 *  @version  1.0
 *
 *  @brief    This file contains the namespace
 *  Universe which contains the classes definition
 *  for Star.
 * */

#ifndef PS4_STAR_HPP
#define PS4_STAR_HPP

#include <SFML/Graphics.hpp>
#include "SpaceObject.hpp"

namespace Universe {
/**
 *  @brief  This is a Star object that inherits from
 *  SpaceObject which is a sf:Drawabl. This class will
 *  model
 * */
class Star : public SpaceObject {
 public:
  /**
   *  @brief  This is the empty construct of the Star
   *  object. This constructor will generate a random
   *  position and radius for the object which will be
   *  used for the sf::CircleShape.
   *  @test: Hung Q Nguyen
   *  Unit Testing: There are multiple ways for testing this
   *  object by using this object to call the existed functions
   *  There are no unique properties for the default constructor
   *  to test.
   * */
  Star();
```

```
  /**
   *  @brief  This a constructor that will allow the
   *  make a Start object with properties that it wants.
   *  This constuctor will not generate the location or
   *  radius of the object, it will use the given
   *  parameters. The velocity of this object is default
   *  to zero becuase this is a static/non-moving object.
   *
   *  @param  sf::Vector2f location, double mass, double radius
   *  @test: Hung Q Nguyen
   *  Unit Testing: to check if the SpaceObject will throw
   *  an exeption if it took a negative mass or not mass <= 0
   *  Same idea, check it with the negative radius
   *  BOOT_REQUIRE_THROW(SpaceObject(location, negative int,
radius))
   *  BOOT_REQUIRE_THROW(SpaceObject(location, mass, negative int))
   * */
  Star(sf::Vector2f location, double mass, double radius);

  /**
   *  @brief  Destructor of the Star object.
   * */
  ~Star();

 private:
  /**
   *  @brief  This is a virtual method inherited from
   *  the SpaceObject abstract class. This class implement
   *  this method, and it will draw the shapes inside it.
   *
   *  @param  sf::RenderTarget &target, sf::RenderStates states
   * */
  virtual void draw(sf::RenderTarget &target, sf::RenderStates
states) const;

  /**
```

```
   *  @brief  This method will be used for the std::generate()
   *  algorithm to generate a random position. This uses the
   *  std::rand() function to generate the x and y points.
   *
   *  @return sf::Vector2f
   * */
  sf::Vector2f positionGenerator();

  /**
   *  @brief  This method will be used for the std::generate()
   *  algorithm to generate a random position. This uses the
   *  std::rand() function to generate the radius.
   *
   *  @return sf::Vector2f
   * */
  float radiusGenerator();
  sf::CircleShape shape_;   // < This is the a circle shape of the
object
  double          radius_;  // < This holds the radius of the circle
};

}

#endif //PS4_STAR_HPP
```

Universe::Universe

```
/** Copyright 2015 Daniel Santos & Hung Nguyen
*  @file     body.hpp
*  @author   Daniel Santos & Hung Nguyen
*  @date     04/22/2016
*  @version  1.0
*
*  @brief    This file contains the namespace
*  Universe which contains the classes definition
*  for Universe class -- the completed Universe
```

```
* */
#include <SFML/Graphics.hpp>
#include <vector>
#include <string>
#include "body.hpp"
#include "SpaceObject.hpp"
#include "Star.hpp"

/**
 *  @author Daniel Santos & Hung Nguyen
 *
 *  @brief  This is the namespace for the classes
 *  definition.
 * */
namespace Universe {
class Universe {
 public:
  /*  @author: Daniel Santos & Hung Q Nguyen
   *  @brief: The complete univer object
   *  Which is included the stars and planets
   *  @params: none
   *  @return: none
   *  @test: Hung Q Nguyen
   *  Unit Testing: There are multiple ways for testing this
   *  object by using this object to call the existed functions
   *  There are no unique properties for the default constructor
   *  to test.
   */
  Universe();

  /*  @author: Daniel Santos & Hung Q Nguyen
   *  @brief: The destructor of the universe object
   *  @params: none
   *  @return: none
   */
  ~Universe();
```

```
  /*  @author: Daniel Santos & Hung Q Nguyen
   *  @brief: This function will draw every planet
   *  and star to the screen at the updated moment.
   *  It will be called multiple times as long as
   *  the window opens and it will update the properties
   *  of Space Object every time it get called.
   *  @params: none
   *  @return: none
   */
  void run();

  /*  @author: Daniel Santos & Hung Q Nguyen
   *  @brief: Getting the values from space and
   *  translate it to the vector2f from sfml
   *  @params: float x -- x position and float y -- y position
   *  @return: sf::Vector2f -- coordinate vector
    *  @test: Hung Q Nguyen
   *  Unit Testing: to test this function, we will pass in two
   *  floats to the function and check if it returns the exact
vector
   *  or not
   *  BOOST_REQUIRE_EQUAL(sf::Vector2f(0.5,0.5),
translateCoordinates(0.5,0.5))
   */
  sf::Vector2f translateCoordinates(float x, float y);

  /*  @author: Daniel Santos & Hung Q Nguyen
   *  @brief: Print the current state of the universe
   *  @params: none
   *  @return: none
   */
  void printState();

  /*  @author: Daniel Santos & Hung Q Nguyen
   *  @brief: Getting the ref of the elapsed time string.
```

```
   *  @params: none
   *  @return: sf::Text& -- Ref to the current text (String of
elapsed time)
   */
  sf::Text& getTextTime() const;

  /*  @author: Daniel Santos & Hung Q Nguyen
   *  @brief: setting the text from value of elapsed time.
   *  @params: int -- value of elapse time
   *  @return: none
   */
  void setTextTime(int elapsed_time);

  /*  @author: Daniel Santos & Hung Q Nguyen
   *  @brief: get the value of elapsed time
   *  @params: none
   *  @return: int -- value of Elapsed time
   */
  int getElapsedTime() const;

 private:
  /*  @author: Daniel Santos & Hung Q Nguyen
   *  @brief: Generate the vector of Bodies
   *  by reading the information from the file (given)
   *  @params: none
   *  @return: none
   */
  void fetchBody();

  /*  @author: Daniel Santos & Hung Q Nguyen
   *  @brief: Generate the vector of Stars
   *  @params: none
   *  @return: none
   */
  void fetchStar();
```

```
  /*  @author: Daniel Santos & Hung Q Nguyen
   *  @brief: Update the elapsed time as long as
   *  the program is running.
   *  @params: none
   *  @return: none
   */
  void updateTime();

  /*  @author: Daniel Santos & Hung Q Nguyen
   *  @brief: Update the coordinates of the planets
   *  and the properties of each planet.
   *  @params: none
   *  @return: none
   */
  void updateUniverse();

  std::vector<Body> bodyList_;  ///< List of Bodies
  std::vector<Star> starList_;  ///< List of Stars
  int elapsedTime_;   ///< Elapsed Time
  sf::Font fontTime_;  ///< Font of Text displaying in the screen
  sf::Text textTime_;  ///< Text of the Planet.
};
}
```