Computing IV Project Portfolio

Joel Savitz

Fall 2018

Contents

List of Figures

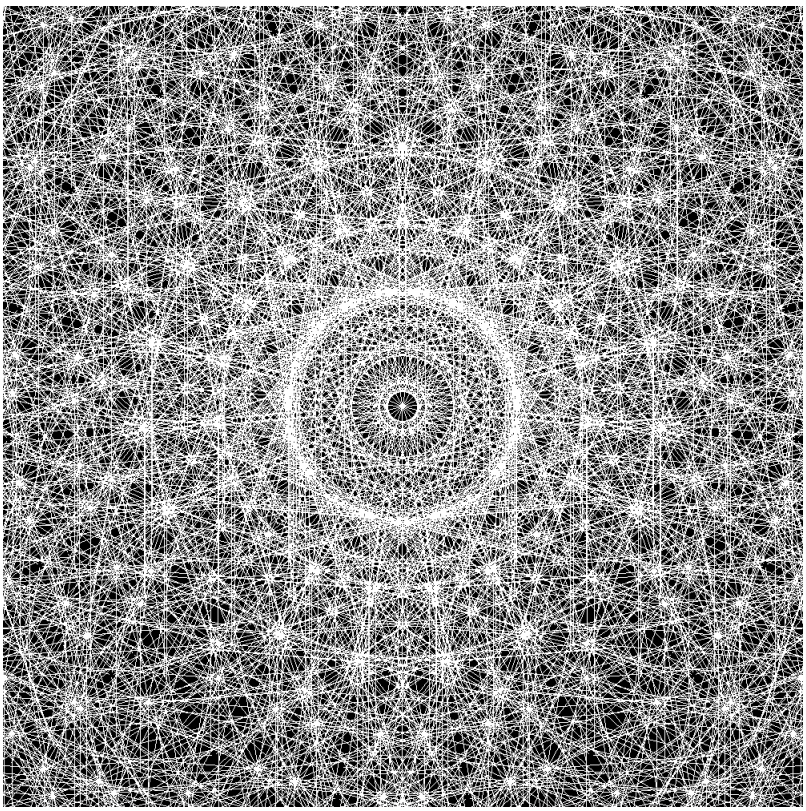


Figure 1: Output from a fractal generator I wrote in preparation for this class

1 Introduction

I thoroughly enjoyed these projects.

I will provide a short discussion of each project followed by it's source code.

Thank you Dr. Wilkes for approving and guiding me through an independent study of Computing IV. I gained a lot out of the practice with C++ that I believe will serve me well in my programming career.

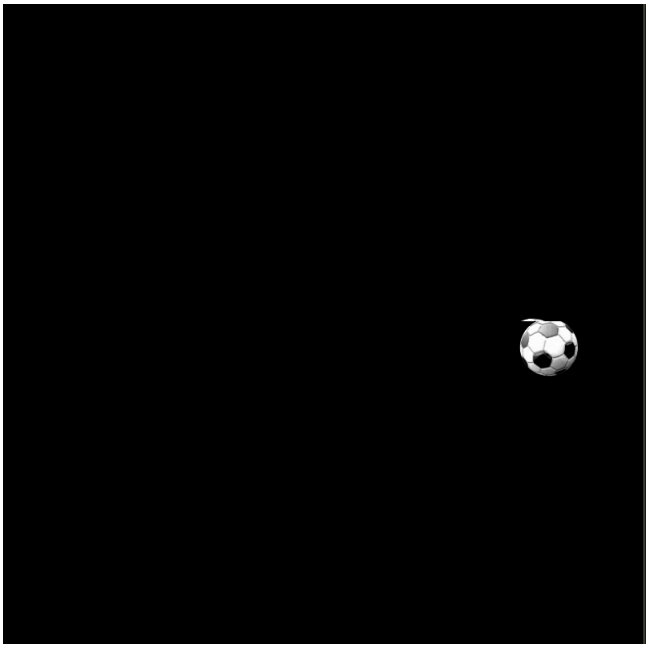


Figure 2: My soccer ball sprite about to bounce off the side of the window

2 PS0: Hello World with SFML

The general idea of this assignment was as an introduction to the capabilities of SFML. First, I used supplied code to create a simple window with a circle in it. Then I modified the code to display a soccer ball, and added the ability for the user to modify the velocity of the ball using the arrow keys on the keyboard.

By setting the refresh rate of the window, I was able to create a loop that would change the position of the soccer ball by a variable velocity value in both the horizontal and vertical directions. I check if the new position of the ball would be outside of the border of the window, and if so, I reverse the velocity in that direction to simulate a bouncing effect. I also print the current velocity values at every refresh to standard output.

2.1 `Makefile`

```
1 CPPC = g++
2 CPPFLAGS = -g -ansi -Wall -Werror -pedantic
```

```
3 LIBS = -lsfml-graphics -lsfml-window -lsfml-system
4
5 BIN = app
6
7 all:
8  $(CPPC) $(CPPFLAGS) main.cpp -o $(BIN) $(LIBS)
```

2.2 main.cpp

```
1  #include <SFML/Graphics.hpp>
2  #include <stdio.h>
3
4  // Global Frames Per Second
5  int const FPS = 60 ;
6
7  // Global constants for window size
8  int const WINDOW_WIDTH = 800 ;
9  int const WINDOW_HEIGHT = 800 ;
10
11 class Ball {
12
13   sf::Sprite& _sprite ;
14
15   // velocity in pixels per second
16   float _xVelocity ;
17   float _yVelocity ;
18
19   int _xDim ;
20   int _yDim ;
21
22 public:
23
24   Ball(sf::Sprite& sprite) ;
25
26   // Velocity is in pixels per second
27
28   void setXVelocity(float newVelocity) ;
29   void setYVelocity(float newVelocity) ;
30
31   float getXVelocity() ;
32   float getYVelocity() ;
33
34   void updatePosition() ;
35
36   sf::Sprite& getSprite() ;
```

```
37
38 } ; // class Ball
39
40 Ball::Ball(sf::Sprite& sprite) :
41   _sprite(sprite),
42   _xVelocity(0.f),
43   _yVelocity(0.f),
44   _xDim((float)_sprite.getTexture()->getSize().x),
45   _yDim((float)_sprite.getTexture()->getSize().y)
46   {
47     // Start the ball in the center of the window
48     _sprite.setPosition(WINDOW_WIDTH/2 - _xDim/2, WINDOW_HEIGHT
         /2 - _yDim/2) ;
49   }
50
51 sf::Sprite& Ball::getSprite() {
52   return _sprite ;
53 }
54
55 void Ball::setXVelocity(float newVelocity) {
56   _xVelocity = newVelocity ;
57 }
58
59 void Ball::setYVelocity(float newVelocity) {
60   _yVelocity = newVelocity ;
61 }
62
63 float Ball::getXVelocity() {
64   return _xVelocity ;
65 }
66
67 float Ball::getYVelocity() {
68   return _yVelocity ;
69 }
70
71 void Ball::updatePosition() {
72   sf::Vector2f newPosition ;
73
74   newPosition.x = _sprite.getPosition().x + _xVelocity / FPS ;
75   newPosition.y = _sprite.getPosition().y + _yVelocity / FPS ;
76
77   // collision check first
78
79   // Case: left edge collision
80   if (newPosition.x <= 0 ) {
```

```
81      newPosition.x = 0 ;
82      _xVelocity *= -1 ; // Reverse direction and maintain
            momentum
83    }
84    // Case: top edge collision
85    else if (newPosition.y <= 0 ) {
86      newPosition.y = 0 ;
87      _yVelocity *= -1 ;
88    }
89    // Case: right edge collision
90    else if (newPosition.x + _xDim >= WINDOW_WIDTH) {
91      newPosition.x = WINDOW_WIDTH - _xDim ;
92      _xVelocity *= -1 ;
93    }
94    // Case: bottom edge collision
95    else if (newPosition.y + _yDim >= WINDOW_HEIGHT) {
96      newPosition.y = WINDOW_HEIGHT - _yDim ;
97      _yVelocity *= -1 ;
98    }
99
100   _sprite.setPosition(newPosition) ;
101 }
102
103 int main() {
104
105   sf::RenderWindow window(sf::VideoMode(WINDOW_WIDTH,
          WINDOW_HEIGHT), "SFML works!") ;
106
107   // Updates to display happen FPS times per second
108   window.setFramerateLimit(FPS) ;
109
110
111   sf::Texture textureFootball;
112   if (textureFootball.loadFromFile("paintex_football_small.png")
          == false) {
113     exit(1) ;
114   }
115   sf::Sprite spriteFootball(textureFootball) ;
116
117   Ball football(spriteFootball) ;
118
119   printf("origin is at: %f, %f\n", spriteFootball.getOrigin().x,
          spriteFootball.getOrigin().y) ;
120   while (window.isOpen())
121   {
```

```
122     sf::Event event ;
123     while (window.pollEvent(event))
124     {
125       if (event.type == sf::Event::Closed)
126       {
127         window.close() ;
128       }
129       else if (event.type == sf::Event::KeyPressed ) {
130
131         switch(event.key.code) {
132           case sf::Keyboard::Right:
133             football.setXVelocity(football.getXVelocity() + 10.f
                  ) ;
134             break ;
135           case sf::Keyboard::Left:
136             football.setXVelocity(football.getXVelocity() - 10.f
                  ) ;
137             break ;
138           case sf::Keyboard::Down:
139             football.setYVelocity(football.getYVelocity() + 10.f
                  ) ;
140             break ;
141           case sf::Keyboard::Up:
142             football.setYVelocity(football.getYVelocity() - 10.f
                  ) ;
143             break ;
144           default:
145             break ;
146         }
147
148       }
149     }
150
151     // check for collision and
152     // update ball position based on current velocity
153     printf("Vx = %f, Vy = %f\n", football.getXVelocity(),
          football.getYVelocity() ) ;
154     football.updatePosition() ;
155
156     window.clear() ;
157     window.draw(football.getSprite()) ;
158     window.display() ;
159
160   }
161
```

```
162     return 0 ;
163   }
```

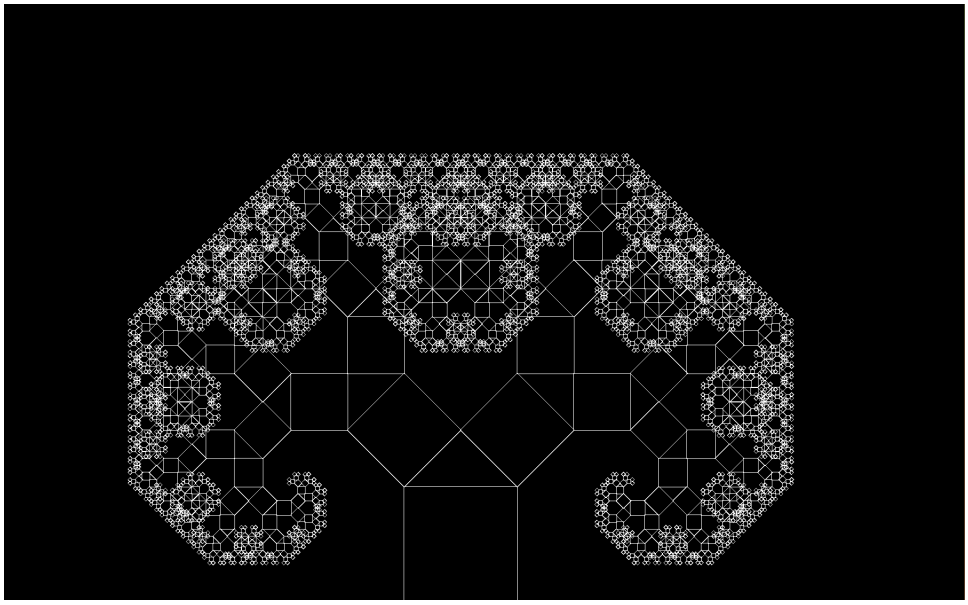


Figure 3: My Pythagoras tree

3 PS1: Recursive Graphics (Pythagoras Tree)

For my recursive graphics assignment, I created a program that draws a Pythagoras tree, as can be seen in figure 3. I modified this program to animate a folding-like rotation of the fractal, a stage of which can be seen in figure 4.

Using basic trigonometry, I was able to implement this algorithm using recursive calls to functions that would draw squares relative to previously drawn squares. By modifying a key angle parameter at a certain rate relative to the screen refresh rate, I was able to simulate movement of the fractal.

The following keys can be used to interact with the program:

[a] Initiates a clockwise rotation/folding of the fractal.

[b] Initiates a swaying animation of the fractal, like a tree in the wind.

[Left Arrow] Increment the rotation of the fractal counterclockwise.

[Right Arrow] Increment the rotation of the fractal clockwise.

[Spacebar] Terminates any running animation.

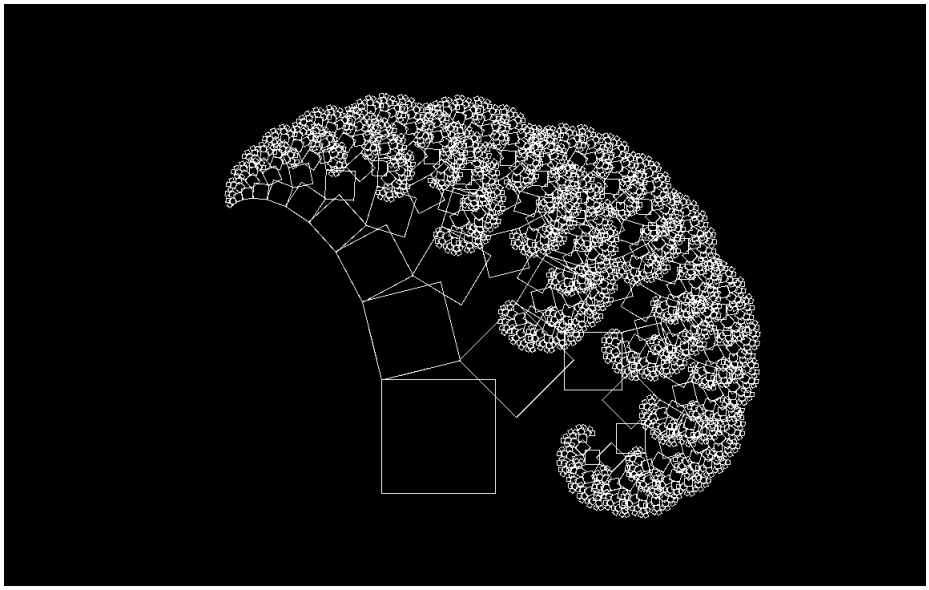


Figure 4: My folded Pythagoras tree

3.1 Makefile

```
1  CPPC = g++
2  CPPFLAGS = -g -ansi -Wall -Werror -pedantic -std=c++11
3  LIBS = -lsfml-graphics -lsfml-window -lsfml-system
4
5  OBJECTS = main.o PTree.o
6
7  BIN = PTree
8
9  all: $(BIN)
10  @echo Make complete.
11
12 $(BIN): $(OBJECTS)
13  $(CPPC) $(CPPFLAGS) $(OBJECTS) -o $(BIN) $(LIBS)
14
15 %.o: %.cpp
16  $(CPPC) $(CPPFLAGS) -c $^
17
18 clean:
19  rm $(BIN) $(OBJECTS)
```

3.2 main.cpp

```
1  // Pythagoras tree
2
3  #include "PTree.hpp"
4  #include <SFML/Graphics.hpp>
5  #include <math.h>
6
7  const int RATIO_WINDOW_WIDTH_TO_SQUARE = 6 ;
8  const int RATIO_WINDOW_HEIGHT_TO_SQUARE = 4 ;
9
10 // Checks if str contains nothing but digits (i.e. can be
       converted to an unsigned int)
11 bool isNumber(char * str) {
12   for(char * c = str; *c != '\0'; c++) {
13     if (!std::isdigit(*c)) {
14       return false ;
15     }
16   }
17   return true ;
18 }
19
20 /* Executable parameters:
21 1: L -- The side length of the base square
22 2: N -- The recursion depth
23 */
24
25 typedef enum animation {NONE, ROTATE, WAVE} Animation;
26
27 extern const int js::FPS = 60 ;
28
29 int main(int argc, char ** argv) {
30
31   // Validate count and type of args
32   if (argc != 3 || !isNumber(argv[1]) || !isNumber(argv[2])) {
33     printf("Usage: PTree <uint base-square-side-length> <uint
          recursion-depth>\n") ;
34     exit(1) ;
35   }
36
37   double baseSquareSideLength = atof(argv[1]) ;
38   unsigned int recursionDepth = atoi(argv[2]) ;
39
40   // Calculate window size from square side length L as 6L x 4L
       (as specified)
41   double windowWidth = RATIO_WINDOW_WIDTH_TO_SQUARE *
       baseSquareSideLength ;
```

```
42    double windowHeight = RATIO_WINDOW_HEIGHT_TO_SQUARE *
          baseSquareSideLength ;
43
44
45    sf::RenderWindow window(sf::VideoMode(windowWidth,
          windowHeight), "SFML-ps1") ;
46
47    window.setFramerateLimit(js::FPS) ;
48
49    js::PTree pTree(recursionDepth, baseSquareSideLength,
          windowWidth, windowHeight) ;
50
51    Animation currentAnimation = NONE ;
52
53    while(window.isOpen()) {
54      sf::Event capturedEvent ;
55
56      while (window.pollEvent(capturedEvent) ) {
57        if (capturedEvent.type == sf::Event::Closed) {
58          window.close() ;
59        }
60        if (capturedEvent.type == sf::Event::KeyPressed) {
61          switch(capturedEvent.key.code) {
62            case sf::Keyboard::Q:
63              window.close() ;
64              break ;
65            case sf::Keyboard::Left:
66              pTree.shiftLeft() ;
67              break ;
68            case sf::Keyboard::Right:
69              pTree.shiftRight() ;
70              break ;
71            case sf::Keyboard::A:
72              currentAnimation = ROTATE ;
73              break ;
74            case sf::Keyboard::B:
75              currentAnimation = WAVE ;
76              break ;
77            case sf::Keyboard::Space:
78              currentAnimation = NONE ;
79              break ;
80            default:
81            break ;
82          }
83        }
```

```
84      }
85
86      switch(currentAnimation) {
87        case ROTATE:
88          pTree.updateRotateAnimation() ;
89          break ;
90        case WAVE:
91          pTree.updateWaveAnimation() ;
92          break ;
93        case NONE:
94        default:
95          break ;
96      }
97
98      window.clear() ;
99      window.draw(pTree) ;
100     window.display() ;
101   }
102
103   return 0;
104 }
```

3.3 PTree.hpp

```
1  #ifndef PTREE_H
2  #define PTREE_H "PTree.hpp"
3
4  #include <SFML/Graphics.hpp>
5
6  namespace js {
7
8  extern const int FPS ;
9
10 // Small helper class used to easily draw squares
11 class Square : public sf::ConvexShape {
12 public:
13   sf::Vector2f _origin ;
14   sf::Vector2f _branchTop ;
15   sf::Vector2f _terminal ;
16   sf::Vector2f _branchBottom ;
17   sf::Vector2f _offset ;
18   double _sideLength ;
19
20   Square(sf::Vector2f origin, sf::Vector2f offset ) ;
21 } ; // class Square
```

```
22
23 class PTree : public sf::Drawable{
24
25   // Recursion depth -- how many layers of squares to draw
26   unsigned int _recursionDepth ;
27
28   // The side length of the first square drawn -- the base
       square
29   double _baseSquareSideLength ;
30
31   // Horizontal dimension of window
32   double _windowWidth ;
33
34   // Vertical dimension of window
35   double _windowHeight ;
36
37   // A user supplied value to shift the tree in real time
38   double _shiftModifier = 0;
39
40   // Recursively draw the fractal
41   void pTree(unsigned int currentDepth, sf::RenderTarget& target
       , Square base) const ;
42
43 public:
44
45   // Construct the object
46   PTree(unsigned int recursionDepth, double baseSquareSideLength
       , double windowWidth, double windowHeight) ;
47
48   // Draw the object on the screen as per object state
49   virtual void draw(sf::RenderTarget& target, sf::RenderStates
       states) const ;
50
51   void shiftLeft() ;
52
53   void shiftRight() ;
54
55   void shift(double deltaTheta) ;
56
57   void updateRotateAnimation() ;
58
59   void updateWaveAnimation() ;
60
61 } ; // class PTree
62
```

```
63 } // namespace js
64
65 #endif // PTREE_H
```

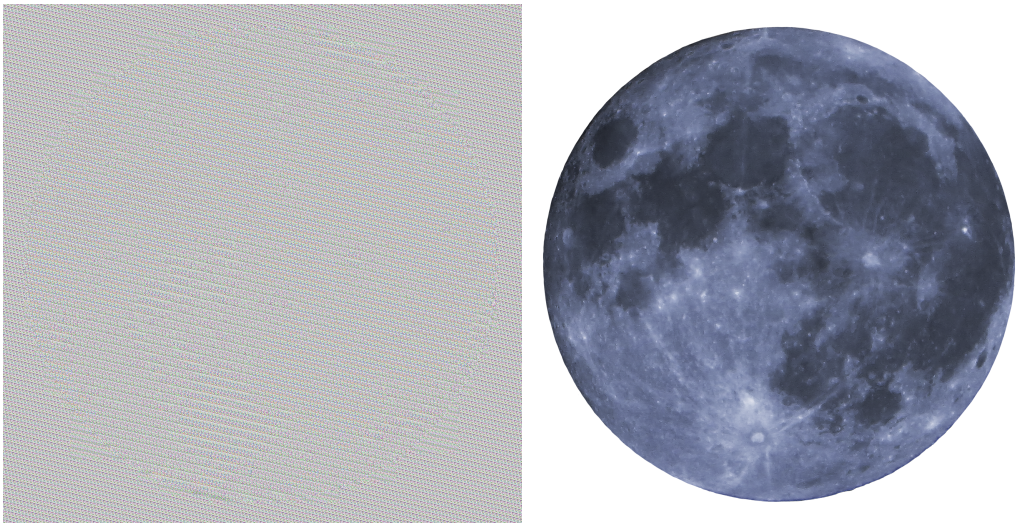
3.4 PTree.cpp

```
1  #include "PTree.hpp"
2  #include <math.h>
3  #include <iostream>
4
5  using namespace js ;
6
7
8  Square::Square(sf::Vector2f origin, sf::Vector2f offset ) :
       _origin(origin), _offset(offset) {
9
10   setPointCount(4) ;
11   setOutlineColor(sf::Color::White) ;
12   setOutlineThickness(1) ;
13   setFillColor(sf::Color::Transparent) ;
14
15   double offsetMagnitude = offset.x ;
16   double offsetDirectionRadians = offset.y ;
17
18   _sideLength = offsetMagnitude * cos(M_PI/4)  ;
19
20   // Calculation of vertex points based on origin coordinates
         and offset vector
21   _branchBottom = sf::Vector2f(origin.x + _sideLength * cos(
         offsetDirectionRadians - M_PI/4 ), origin.y - _sideLength *
          sin(offsetDirectionRadians - M_PI/4)) ;
22   _branchTop = sf::Vector2f(origin.x + _sideLength * cos(
         offsetDirectionRadians + M_PI/4), origin.y - _sideLength *
         sin(offsetDirectionRadians + M_PI/4)) ;
23   _terminal = sf::Vector2f(origin.x + offsetMagnitude * cos(
         offsetDirectionRadians),  origin.y - offsetMagnitude * sin(
         offsetDirectionRadians)) ;
24
25   //std::cout << "Origin coords: (" << origin.x << "," << origin
         .y << ")" << std::endl ;
26   //std::cout << "terminal coords: (" << _terminal.x << "," <<
         _terminal.y << ")" << std::endl ;
27
28
29   setPoint(0, _origin) ;
```

```
30    setPoint(1, _branchBottom) ;
31    setPoint(2, _terminal) ;
32    setPoint(3, _branchTop) ;
33  }
34
35  PTree::PTree(unsigned int recursionDepth, double
        baseSquareSideLength, double windowWidth, double windowHeight
        ) :
36  _recursionDepth(recursionDepth),
37  _baseSquareSideLength(baseSquareSideLength), // This value is a
        double because I will be doing calculations with it
38  _windowWidth(windowWidth),
39  _windowHeight(windowHeight)
40  {}
41
42  void PTree::pTree(unsigned int currentDepth, sf::RenderTarget&
        target, Square base) const {
43
44    // If we have reached the recursion depth, there is no more
          work to do.
45    if (currentDepth >= _recursionDepth) return ;
46
47    // Draw the left child square
48    Square left(base._branchTop, sf::Vector2f(base._sideLength,
          base._offset.y + M_PI/4 + _shiftModifier) ) ;
49
50    // Draw the right child square based on the left child
51    Square right(left._branchBottom, sf::Vector2f(base._sideLength
          , base._offset.y - M_PI/4) ) ;
52
53    // Draw the left square and the rest of the shape
54    target.draw(left) ;
55    pTree(currentDepth + 1, target, left) ;
56
57    // Draw the right square and the rest of the shape
58    target.draw(right) ;
59    pTree(currentDepth + 1, target, right) ;
60  }
61
62  void PTree::draw(sf::RenderTarget& target, sf::RenderStates
        states) const {
63
64    Square base(sf::Vector2f(_windowWidth/2 -
          _baseSquareSideLength/2, _windowHeight - 0.2 *
          _windowHeight), sf::Vector2f(_baseSquareSideLength, M_PI/4)
```

```
      ) ;
65    target.draw(base) ;
66
67    pTree(0, target, base) ;
68  }
69
70  void PTree::shiftLeft() {
71    _shiftModifier += M_PI / 512 ;
72  }
73
74  void PTree::shiftRight() {
75    _shiftModifier -= M_PI / 512 ;
76  }
77
78  void PTree::shift(double deltaTheta) {
79    _shiftModifier += deltaTheta ;
80  }
81
82  void PTree::updateRotateAnimation() {
83
84    // Every second, shift  M_PI / 64 radians
85    double deltaThetaPerSecond = M_PI / 8 ;
86
87    // Divide by frames per second and update every frame
88    shift(-1 * deltaThetaPerSecond / FPS) ;
89  }
90
91  void PTree::updateWaveAnimation() {
92
93    // Retain the value of psi between function calls
94    static double psi = 0 ;
95
96    // Animation should have period of 4 seconds, so each second,
       psi should increment by 2pi/4
97    const double deltaPsiPerSecond = M_PI / 2 ;
98
99    // Using psi, calculate the change in angle for the current
       frame
100   double deltaTheta = sin(psi) / 1024 ;
101
102   // Apply the shift in angle
103   shift(deltaTheta) ;
104
105   psi += deltaPsiPerSecond / FPS ;
106
```

```
107     // If psi has gone around the circle, reset it
108     if (psi >= 2 * M_PI) {
109       psi = 0 ;
110     }
111   }
```



(a) Encoded moon graphic (b) Decoded moon graphic

Figure 5: PhotoMagic in action

4 PS2: Linear Feedback Shift Register and Image Encoding

This assignment had two stages. First, I implemented a Linear Feedback Shift Register ("LFSR") and tested it using the Boost unit test framework. Then, I used my LFSR to generate pseudo random values to encode a `png` file.

The LFSR requires an initial seed and tap value to run, and it is reversible, so running an image through the program with one set of parameters will output an image file that when passed as input to the program with the same parameters will output the original image. Figure 5a demonstrates the output of running figure 5b through the `PhotoMagic` executable. Figure 5b is also the output of figure 5a run through the program with the same parameters.

The seed is a binary bit string, and the tap value is a position in the string to use in the calculation of the pseudo-random values.

I learned about basic image encryption and gained some insight into how the `png` file format works. I also explore the Boost unit test framework, and the tests I wrote for my LFSR can be seen in section 4.5.

4.1 Makefile

```
1  CPPC = g++
2  CPPFLAGS = -g -ansi -Wall -Werror -pedantic --std=c++11
3  LIBS = -lboost_unit_test_framework -lsfml-graphics -lsfml-window
       -lsfml-system
4
5  OBJECTS = PhotoMagic.o LFSR.o
6
7  BIN = PhotoMagic
8
9  all: $(BIN)
10  @echo Make complete.
11
12 $(BIN): $(OBJECTS)
13  $(CPPC) $(CPPFLAGS) $^ -o $@  $(LIBS)
14
15 %.o: %.cpp
16  $(CPPC) $(CPPFLAGS) -c $^
17
18 clean:
19  rm $(BIN) $(OBJECTS)
```

4.2 PhotoMagic.cpp

```
1  #include "LFSR.hpp"
2  #include <SFML/Graphics.hpp>
3  #include <iostream>
4
5
6  bool validateArgs(int argc, char ** argv) {
7    int tap ;
8    try {
9       tap = std::atoi(argv[4]) ;
10   }
11   catch(const std::exception & e) {
12     std::cout << "Error: " << e.what() << std::endl ;
13     return false ;
14   }
15
16   try {
17     jsavitz::LFSR validator(argv[3], tap) ;
18   }
19   catch (const char * eMsg) {
20     std::cout << "Error: " << eMsg << std::endl ;
```

```
21      return false ;
22    }
23
24    return true ;
25  }
26
27  sf::Image negateImage(const sf::Image& input) {
28    sf::Image output = input ;
29    sf::Vector2u dimensions = input.getSize() ;
30
31    sf::Color tempPixel ;
32
33    for(unsigned int i = 0; i < dimensions.x; i++) {
34      for(unsigned int j = 0; j < dimensions.y; j++) {
35        tempPixel = input.getPixel(i,j) ;
36        output.setPixel(i,j,sf::Color(255 - tempPixel.r, 255 -
              tempPixel.g, 255 - tempPixel.b)) ;
37      }
38    }
39
40    return output ;
41  }
42
43  sf::Image encryptImage(const sf::Image& input, jsavitz::LFSR
      lfsr) {
44    sf::Image output = input ;
45    sf::Vector2u dimensions = input.getSize() ;
46
47    sf::Color tempPixel ;
48
49    for(unsigned int i = 0; i < dimensions.x; i++) {
50      for(unsigned int j = 0; j < dimensions.y; j++) {
51        tempPixel = input.getPixel(i,j) ;
52        output.setPixel(i,j,sf::Color(tempPixel.toInteger() ^ lfsr
              .generate(32))) ;
53      }
54    }
55
56    return output ;
57  }
58
59  int main(int argc, char ** argv) {
60
61    // Validate command line args
62    if(argc != 5 || !validateArgs(argc, argv)) {
```

```
63      std::cout << "Usage: PhotoMagic <input-file> <output-file> <
            LFSR-seed> <tap-position>" << std::endl ;
64      exit(1) ;
65    }
66
67    sf::Image input ;
68
69    if (!input.loadFromFile(argv[1])) {
70      std::cout << "Error: Input file " << argv[1] << " not found.
            " ;
71      exit(1) ;
72    }
73
74    jsavitz::LFSR lfsr(argv[3],std::atoi(argv[4])) ;
75
76    sf::Image output = encryptImage(input,lfsr) ;
77
78    if (!output.saveToFile(argv[2])) {
79      std::cout << "Error: Unable to save output file with the
            name " << argv[2] << std::endl ;
80    }
81
82    auto windowDim = input.getSize() ;
83
84    const int WIDTH = windowDim.x ;
85    const int HEIGHT = windowDim.y;
86
87    sf::RenderWindow windowOriginal(sf::VideoMode(WIDTH, HEIGHT),"
          SFML-LFSR-original") ;
88    sf::RenderWindow windowResult(sf::VideoMode(WIDTH, HEIGHT),"
          SFML-LFSR-result") ;
89
90    sf::Texture inputImageTexture ;
91    sf::Texture outputImageTexture ;
92
93    inputImageTexture.loadFromImage(input) ;
94    outputImageTexture.loadFromImage(output) ;
95
96    sf::Sprite inputSprite(inputImageTexture) ;
97    sf::Sprite outputSprite(outputImageTexture) ;
98
99
100   while (windowOriginal.isOpen() && windowResult.isOpen()) {
101     sf::Event userEvent ;
102
```

```
103     while(windowOriginal.pollEvent(userEvent)) {
104       if(userEvent.type == sf::Event::Closed) {
105         windowOriginal.close() ;
106       }
107     }
108     while(windowResult.pollEvent(userEvent)) {
109       if(userEvent.type == sf::Event::Closed) {
110         windowResult.close() ;
111       }
112     }
113
114     windowOriginal.clear() ;
115     windowOriginal.draw(inputSprite) ;
116     windowOriginal.display() ;
117
118     windowResult.clear() ;
119     windowResult.draw(outputSprite) ;
120     windowResult.display() ;
121   }
122
123 }
```

4.3 LFSR.hpp

```
1  #ifndef LFSR_HPP
2  #define LFSR_HPP "LFSR.hpp"
3
4  #include <string>
5
6  namespace jsavitz {
7
8  class LFSR {
9
10 public:
11     // Constructor that creates the object based on a binary
          string and integer tap value
12     LFSR(std::string seed, int tap) ;
13
14     // Simulate a single step. Returns the new bit (either 0 or
          1)
15     int step() ;
16
17     // Simulate k steps. Returns a k-bit integer
18     int generate(int k) ;
19
```

```
20      std::ostream& write(std::ostream& target) const ;
21
22  private:
23
24      // Simulate left bitshift operation on object's state
25      void shiftLeft() ;
26
27      // Private storage of the tap value. Must be positive
28      unsigned int _tap ;
29
30      // The current characters stored in the register
31      std::string _state ;
32
33      // The size of the register
34      std::string::size_type _length ;
35
36  friend std::ostream& operator<<(std::ostream& target, const LFSR
        & rvalue) {
37      return rvalue.write(target) ;
38  }
39
40  } ; // class LFSR
41
42
43  } // namespace jsavitz
44
45  #endif // LFSR_HPP
```

4.4 LFSR.cpp

```
1   #include "LFSR.hpp"
2   #include <vector>
3   #include <math.h>
4   #include <iostream>
5
6   using namespace jsavitz ;
7
8   // Function to simuate XOR on two bits stored as chars
9   // Returns either a 0 or 1
10  // Inputs MUST be either '0' or '1'
11  inline char XOR(char a, char b) {
12      return a == b ? '0' : '1' ;
13  }
14
15  // Convert a binary value represented as a vector of characters
```

```
    (least->most significant) to an binaryCharVectorToInteger
// Input vector<char> MUST contain only '0's and '1's
int binaryCharVectorToInteger(const std::vector<char>&
    binaryCharVector ) {
    int integer = 0 ;

    for(int i = 0; i < (int) binaryCharVector.size(); ++i) {
        integer += binaryCharVector[i] == '1' ? pow(2,(double)
            binaryCharVector.size() - i - 1) : 0 ;
    }

    return integer ;
}

void LFSR::shiftLeft() {
    // Swap every character with the proceeding one from the
        back to the front
    for(auto i = 0; i < (int)(_length - 1); ++i) {
        std::swap(_state[i],_state[i + 1]) ;
        //std::cout << _state << std::endl ;
    }

    // Finally, set the last character in the string (least
        significant bit) to 0
    _state[_length - 1] = '0' ;
}

LFSR::LFSR(std::string seed, int tap) {

    // Validate that seed string contains only 0's and 1's
    for(auto bit : seed) {
        if (bit != '0' && bit != '1') throw "Invalid seed string
            !" ;
    }

    // A seed string must not be more than 32 bits in length,
        per specification
    if (seed.length() > 32) throw "Seed string too long!" ;

    // Validate that tap is non-negative and less than the seed
        length
    if (tap < 0 || tap >= (int)seed.length()) throw "Tap value
        out of bounds!" ;

    // Now that everything is in order, initialize the data
```

```
        members
52      _state = seed ;
53      _tap = tap ;
54      _length = seed.length() ;
55  }
56
57  int LFSR::step() {
58      char tempBit = XOR (_state[0], _state[_length - _tap - 1]) ;
59
60      shiftLeft() ;
61
62      _state[_length - 1] = tempBit ;
63      //std::cout << _state << std::endl ;
64      //std::cout << "-----" << std::endl ;
65
66      return tempBit == '0' ? 0 : 1 ;
67  }
68
69  int LFSR::generate(int iterations) {
70
71      std::vector<char> generatedBitsAsChars ;
72
73      for(int i = 0; i < iterations; i++) {
74          generatedBitsAsChars.push_back(step() == 0 ? '0' : '1')
                ;
75      }
76
77      /*
78      std::cout << "binary representation: " << std::endl ;
79      for(auto i : generatedBitsAsChars) {
80          std::cout << i << std::endl ;
81      }
82      */
83
84      return binaryCharVectorToInteger(generatedBitsAsChars) ;
85  }
86
87  std::ostream& LFSR::write(std::ostream& target) const {
88     return target << _state ;
89  }
```

4.5 test.cpp

```
1  #include <iostream>
2  #include <string>
```

```
3
4  #include "LFSR.hpp"
5
6  #define BOOST_TEST_DYN_LINK
7  #define BOOST_TEST_MODULE Main
8  #include <boost/test/unit_test.hpp>
9
10 BOOST_AUTO_TEST_CASE(fiveBitsTapAtTwo) {
11
12   jsavitz::LFSR l("00111", 2);
13   BOOST_REQUIRE(l.step() == 1);
14   BOOST_REQUIRE(l.step() == 1);
15   BOOST_REQUIRE(l.step() == 0);
16   BOOST_REQUIRE(l.step() == 0);
17   BOOST_REQUIRE(l.step() == 0);
18   BOOST_REQUIRE(l.step() == 1);
19   BOOST_REQUIRE(l.step() == 1);
20   BOOST_REQUIRE(l.step() == 0);
21
22   jsavitz::LFSR l2("00111", 2);
23   BOOST_REQUIRE(l2.generate(8) == 198);
24 }
25
26 BOOST_AUTO_TEST_CASE(constructor_test) {
27
28   // Invalid initial bit strings
29
30   // non 0 or 1 characters
31   BOOST_CHECK_THROW(jsavitz::LFSR("invalid string",0), const
        char *) ;
32
33   // Too small
34   BOOST_CHECK_THROW(jsavitz::LFSR("",4), const char *) ;
35   BOOST_CHECK_THROW(jsavitz::LFSR("10011000101030101",4), const
        char *) ;
36
37   // Out of bounds tap values (low + high)
38   BOOST_CHECK_THROW(jsavitz::LFSR("1110001",-3), const char *) ;
39   BOOST_CHECK_THROW(jsavitz::LFSR("1110001",100), const char *)
        ;
40
41   // Can create an LFSR from 32 bits
42   BOOST_CHECK_NO_THROW(jsavitz::LFSR("
        10110001001001110110010100101010", 23)) ;
43
```

```
44    // But no more
45    BOOST_CHECK_THROW(jsavitz::LFSR("
          10110001001001110110010100101010 0", 23), const char *) ;
46
47  }
48
49  BOOST_AUTO_TEST_CASE(step_test) {
50    jsavitz::LFSR test1("011001",4) ;
51
52    // Step() return value expected to be true then false
53    BOOST_CHECK_EQUAL(test1.step(),1) ;
54    BOOST_CHECK_EQUAL(test1.step(),0) ;
55    BOOST_CHECK_EQUAL(test1.step(),1) ;
56  }
57
58  BOOST_AUTO_TEST_CASE(generate_test) {
59    jsavitz::LFSR test2("01101000010", 8) ;
60
61    // Generate yields expected value
62    BOOST_CHECK_EQUAL(test2.generate(5),25) ;
63    BOOST_CHECK_EQUAL(test2.generate(5),4) ;
64    BOOST_CHECK_EQUAL(test2.generate(5),30) ;
65  }
```

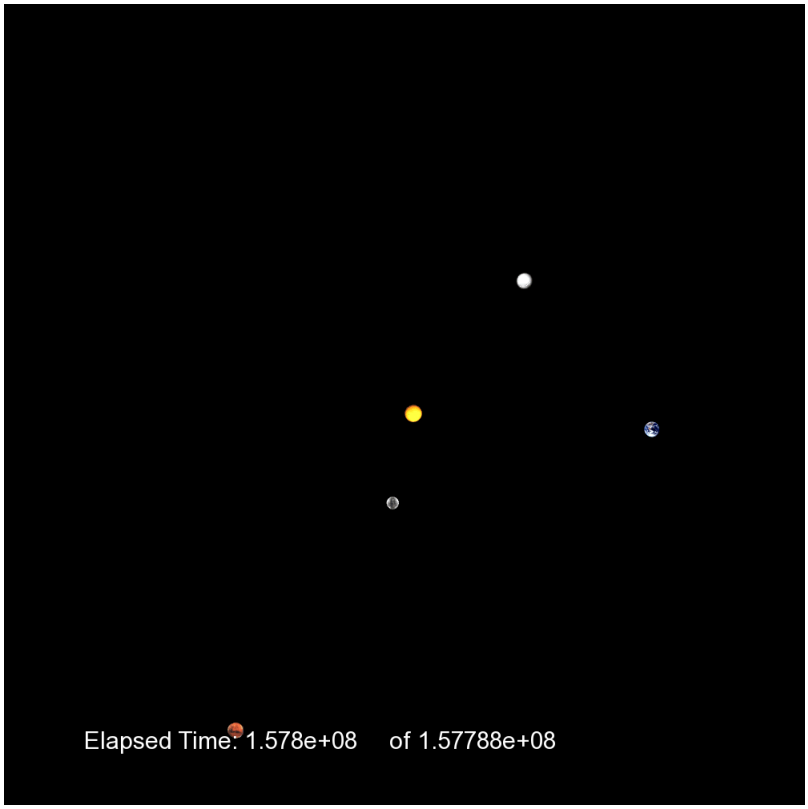


Figure 6: The solar system in motion

5 PS3: N-Body Simulation

For this assignment, I created a physics simulation of the gravitational particles acting on an arbitrary number of particles. The program takes an input file containing the initial state of the simulation and the names of sprite files to display for each particle. The total duration of the simulation as well as the time interval to simulate movement for each iteration of the loop is supplied via command line arguments.

When it is run, the program displays a window containing the particles and a display of the elapsed time out of the total time of the simulation, as seen in figure 6. Then, the simulation progresses until the total time has been reached, at which point the simulation freezes in that position and the program outputs the final state of the simulation to standard out.

I learned how to create basic physics simulations and gained some expe-

rience in program organization and debugging mathematical programs.

5.1 Makefile

```
1  CPPC = g++
2  CPPFLAGS = -g -ansi -Wall -Werror -pedantic --std=c++11
3  LIBS = -lsfml-graphics -lsfml-window -lsfml-system
4
5  OBJECTS = main.o body.o
6
7  BIN = NBody
8
9  all: $(BIN)
10  @echo Make complete.
11
12 $(BIN): $(OBJECTS)
13  $(CPPC) $(CPPFLAGS) $(OBJECTS) -o $(BIN) $(LIBS)
14
15 %.o: %.cpp
16  $(CPPC) $(CPPFLAGS) -c $^
17
18 clean:
19  rm $(BIN) $(OBJECTS)
```

5.2 main.cpp

```
1  #include <SFML/Graphics.hpp>
2  #include <iostream>
3  #include <vector>
4  #include <memory>
5  #include <cmath>
6  #include "body.hpp"
7
8  const int WIDTH = 1000 ;
9  const int HEIGHT = 1000 ;
10
11 void simulatePhysics(std::vector<std::shared_ptr<jsavitz::Body
       >>& bodies, double timeIncrement) {
12   // For each body:
13   sf::Vector2f netForce ;
14   sf::Vector2f relativePositions ;
15   sf::Vector2f acceleration ;
16   sf::Vector2f newVelocity ;
17
```

```
18    double radius ;
19    double tempForceMagnitude ;
20
21    for (auto body : bodies) {
22      // Sum the pairwise forces for each body
23      netForce.x = 0, netForce.y = 0 ;
24      for (auto otherBody : bodies) {
25        // Don't bother calculating force on itself, radius will
              be 0
26        if (body == otherBody) continue ;
27
28        relativePositions.x = otherBody->getPosition().x - body->
              getPosition().x ;
29        relativePositions.y = otherBody->getPosition().y - body->
              getPosition().y ;
30
31        // Calculate radius ;
32        radius = sqrt( pow(relativePositions.x,2.0) + pow(
              relativePositions.y,2.0) ) ;
33
34        tempForceMagnitude = ( jsavitz::Body::G * (body->getMass()
              * otherBody->getMass())) / pow(radius,2.0) ;
35
36        netForce.x += ( tempForceMagnitude * relativePositions.x)
              / radius ;
37        netForce.y += ( tempForceMagnitude * relativePositions.y)
              / radius ;
38      }
39
40      // Calculate acceleration
41      acceleration.x = netForce.x / body->getMass() ;
42      acceleration.y = netForce.y / body->getMass() ;
43
44      // Calculate new velocity
45      newVelocity.x = body->getVelocity().x + acceleration.x *
            timeIncrement ;
46      newVelocity.y = body->getVelocity().y + acceleration.y *
            timeIncrement ;
47
48      // Update stored velocity
49      body->setVelocity(newVelocity) ;
50
51
52    }
53      // Internally update position after new velocities have been
```

```
          calculated
54      for (auto body : bodies) body->step(timeIncrement) ;
55  }
56
57  int main(int argc, char ** argv) {
58
59    if (argc != 3) {
60      std::cout << "Usage: ./NBody <simulation_time> <
           time_increment>" << std::endl ;
61      exit(1) ;
62    }
63
64    double simulationTime = atof(argv[1]) ;
65    double timeIncrement = atof(argv[2]) ;
66    double currentTime = 0 ;
67
68    int numberBodies ;
69
70    std::cin >>  numberBodies ;
71    std::cin >> jsavitz::Body::universeSize ;
72
73    jsavitz::Body::windowWidth  = WIDTH ;
74    jsavitz::Body::windowHeight = HEIGHT ;
75
76    std::vector<std::shared_ptr<jsavitz::Body>> bodies ;
77
78
79    for (int i = 0; i < numberBodies; i++) {
80      bodies.push_back(std::make_shared<jsavitz::Body>()) ;
81    }
82
83    /* Debug output
84    for ( auto body : bodies) {
85      body->printProperties() ;
86    }
87    */
88
89    sf::RenderWindow window(sf::VideoMode(WIDTH,HEIGHT),"SFML-demo
         ") ;
90
91    sf::Event e ;
92
93    sf::Font arial ;
94    arial.loadFromFile("arial.ttf") ;
95    sf::Text elapsedTime("",arial) ;
```

```
96    elapsedTime.setPosition((0.10) * WIDTH, HEIGHT - (0.10) *
          HEIGHT) ;
97    elapsedTime.setFillColor(sf::Color::White) ;
98    std::string timeString ;
99
100   bool justFinished = true ;
101
102   while (window.isOpen()) {
103     window.pollEvent(e) ;
104
105     if (e.type == sf::Event::Closed) {
106       window.close() ;
107     }
108
109     if (currentTime < simulationTime) {
110       simulatePhysics(bodies, timeIncrement) ;
111       //std::cout << "ran sim" << std::endl ;
112       currentTime += timeIncrement ;
113       //std::cout << "TIME: " << currentTime << " of " <<
              simulationTime << std::endl ;
114     }
115     else if (justFinished){
116       std::cout << numberBodies << std::endl ;
117       std::cout << jsavitz::Body::universeSize << std::endl ;
118       for (auto body : bodies) {
119         std::cout << std::scientific
120         << body->getPosition().x << "\t" << body->getPosition().
              y << "\t"
121         << body->getVelocity().x << "\t" << body->getVelocity().
              y << "\t"
122         << body->getMass() << "\t" << body->getSpriteFilename()
              << std::endl ;
123       }
124       justFinished = false ;
125     }
126
127     window.clear() ;
128     // Draw planets
129     for(auto body : bodies) window.draw(*body) ;
130     // Update and draw elapsed time_increment
131     elapsedTime.setString([&]() -> std::string {
132       char floatBuff[32];
133       snprintf(floatBuff, 32, "%g\t of %g", currentTime,
              simulationTime) ;
134       timeString = "Elapsed Time: " + std::string(floatBuff) ;
```

```
            return timeString ;
135     }()) ;
136     window.draw(elapsedTime) ;
137     window.display() ;
138   }
139
140 }
```

5.3 body.hpp

```
 1 #ifndef BODY_HPP
 2 #define BODY_HPP "body.hpp"
 3
 4 #include <SFML/Graphics.hpp>
 5
 6 namespace jsavitz {
 7
 8 class Body : public sf::Drawable {
 9
10 private:
11
12   double _xPosition ;
13
14   double _yPosition ;
15
16   double _xVelocity ;
17
18   double _yVelocity ;
19
20   double _mass ;
21
22   std::string _spriteFilename ;
23
24   sf::Sprite _sprite ;
25
26   sf::Texture _texture ;
27
28   sf::Image _image ;
29
30 public:
31
32   static double universeSize ;
33   static int windowWidth ;
34   static int windowHeight ;
35
```

```
36    constexpr static double G = 6.67e-11 ;
37
38    Body(double xPosition, double yPosition, double xVelocity,
          double yVelocity, double mass, std::string spriteFilename)
          ;
39
40    Body() ;
41
42    void step(double seconds) ;
43
44    sf::Vector2f getVelocity() ;
45
46    void setVelocity(sf::Vector2f velocity) ;
47
48    sf::Vector2f getPosition() ;
49
50    double getMass() ;
51
52    std::string getSpriteFilename() ;
53
54    void printProperties() ;
55
56    void draw(sf::RenderTarget& target, sf::RenderStates states)
          const ;
57
58    friend std::istream& operator >>(std::istream& istream, Body&
          body) {
59
60      istream >> body._xPosition ;
61      istream >> body._yPosition ;
62      istream >> body._xVelocity;
63      istream >> body._yVelocity;
64      istream >> body._mass ;
65      istream >> body._spriteFilename ;
66
67      return istream ;
68    }
69
70
71  } ; // class Body
72
73  } // namespace jsavitz
74
75  #endif // BODY_HPP
```

5.4 body.cpp

```
1  #include "body.hpp"
2  #include <stdexcept>
3  #include <iostream>
4
5  using namespace jsavitz ;
6
7  double Body::universeSize = 0 ;
8
9  int Body::windowWidth = 0 ;
10
11 int Body::windowHeight = 0 ;
12
13 sf::Vector2f getDisplayCoordinates(sf::Vector2f
       cartesianCoordinates) {
14   sf::Vector2f displayCoordinates ;
15
16   displayCoordinates.x = Body::windowWidth/2 + Body::windowWidth
         /2 * ( (cartesianCoordinates.x) / Body::universeSize ) ;
17   displayCoordinates.y = Body::windowHeight/2 + Body::
         windowHeight/2 * ( (-1 * cartesianCoordinates.y) / Body::
         universeSize ) ;
18
19   return displayCoordinates ;
20 }
21
22 Body::Body(double xPosition, double yPosition, double xVelocity,
       double yVelocity, double mass, std::string spriteFilename) :
23 _xPosition(xPosition), _yPosition(yPosition), _xVelocity(
       xVelocity), _yVelocity(yVelocity), _mass(mass),
       _spriteFilename(spriteFilename)
24 {
25   if (!_image.loadFromFile(_spriteFilename)) throw std::
         runtime_error("Body sprite image not found!") ;
26
27   _texture.loadFromImage(_image) ;
28   _sprite.setTexture(_texture) ;
29   _sprite.setPosition(getDisplayCoordinates(sf::Vector2f(
         _xPosition,_yPosition))) ;
30
31 }
32
33 Body::Body() {
34   std::cin >> *this ;
```

```
35    if (!_image.loadFromFile(_spriteFilename)) throw std::
          runtime_error("Body sprite image not found!") ;
36    _texture.loadFromImage(_image) ;
37    _sprite.setTexture(_texture) ;
38    _sprite.setPosition(getDisplayCoordinates(sf::Vector2f(
          _xPosition,_yPosition))) ;
39
40    // Debug output
41    //std::cout << "Position set to" << _sprite.getPosition().x <<
          "," << _sprite.getPosition().y << std::endl ;
42  }
43
44  void Body::draw(sf::RenderTarget& target, sf::RenderStates
        states) const {
45    target.draw(_sprite) ;
46  }
47
48  void Body::step(double seconds) {
49    _xPosition += _xVelocity * seconds ;
50    _yPosition += _yVelocity * seconds ;
51    _sprite.setPosition(getDisplayCoordinates(sf::Vector2f(
          _xPosition,_yPosition))) ;
52  }
53
54  sf::Vector2f Body::getVelocity() {
55    sf::Vector2f velocity ;
56    velocity.x = _xVelocity ;
57    velocity.y = _yVelocity ;
58    return velocity ;
59  }
60
61  void Body::setVelocity(sf::Vector2f velocity) {
62    _xVelocity = velocity.x ;
63    _yVelocity = velocity.y ;
64  }
65
66  sf::Vector2f Body::getPosition() {
67    sf::Vector2f position ;
68    position.x = _xPosition ;
69    position.y = _yPosition ;
70    return position ;
71  }
72
73  double Body::getMass() {
74    return _mass ;
```

```
75 }
76
77 void Body::printProperties() {
78
79   std::cout << _xPosition << "\t" ;
80   std::cout << _yPosition << "\t" ;
81   std::cout << _xVelocity << "\t" ;
82   std::cout << _yVelocity << "\t" ;
83   std::cout << _mass << "\t" ;
84   std::cout << _spriteFilename ;
85
86   std::cout << std::endl ;
87 }
88
89 std::string Body::getSpriteFilename() {
90   return _spriteFilename ;
91 }
```

```
24  #include "RingBuffer.hpp"
25  #include "GuitarString.hpp"
26
27  #define STANDARD_A 432 // Hz
28  #define SAMPLES_PER_SEC 44100
29
30  const std::string pianoMap = "q2we4r5ty7u8i9op-[=zxdcfvgbnjmk,.;/' " ;
```

Figure 7: The definition of my keyboard used to play guitar sounds

```
[joel@theBox ps5b]$ ./test
Running 1 test case...

*** No errors detected
[joel@theBox ps5b]$
```

Figure 8: My PS5b code passing the tests defined in GStest.cpp

6 PS5: Ring Buffer and Guitar Hero

For my final assignment, I created a basic guitar synthesizer using the Karplus-Strong algorithm to simulate vibrations of a guitar string given a random seed. Using the keymappings defined in the code visible in figure 7, the user can input different frequencies that will be played, ranging from 110 Hz to 880 Hz, and the program will will create the sound by passing a frequency parameter to the GuitarString object, which returns samples that can be used as a sound buffer to play a noise that sounds like a guitar playing the note that corresponds to that frequency.

The assignment taught be the basics of electronic musical synthesis and about the audio capabilities of SFML, an aspect which I had not yet explored. I also wrote unit tests for my GuitarString class which can be seen in section 6.7.

6.1 Makefile

```
1  CPPC = g++
2  CPPFLAGS = -g -ansi -Wall -Werror -pedantic --std=c++11
3  LIBS = -lsfml-graphics -lsfml-audio -lsfml-window -lsfml-system
       -lboost_unit_test_framework
4
5  OBJECTS = GuitarHero.o GuitarString.o RingBuffer.o
6  OBJECTS_TEST = GuitarString.o RingBuffer.o GStest.o
```

```
7
8  BIN = GuitarHero
9
10 BIN_TEST = test
11
12 all: $(BIN) $(BIN_TEST)
13  @echo Make complete.
14
15 $(BIN): $(OBJECTS)
16  $(CPPC) $(CPPFLAGS) $^ -o $@ $(LIBS)
17
18 $(BIN_TEST): $(OBJECTS_TEST)
19  $(CPPC) $(CPPFLAGS) $^ -o $@ $(LIBS)
20
21 %.o: %.cpp
22  $(CPPC) $(CPPFLAGS) -c $^
23
24 clean:
25  rm $(BIN) $(BIN_TEST) $(OBJECTS_TEST) $(OBJECTS)
```

6.2 GuitarHero.cpp

```
1  /*
2    Joel Savitz's derivative work based on:
3
4    Copyright 2015 Fred Martin, fredm@cs.uml.edu
5    Mon Mar 30 08:58:49 2015
6
7    Sue me
8  */
9
10 #include <SFML/Graphics.hpp>
11 #include <SFML/System.hpp>
12 #include <SFML/Audio.hpp>
13 #include <SFML/Window.hpp>
14
15 #include <math.h>
16 #include <limits.h>
17
18 #include <iostream>
19 #include <string>
20 #include <exception>
21 #include <stdexcept>
22 #include <vector>
23
```

```
24 #include "RingBuffer.hpp"
25 #include "GuitarString.hpp"
26
27 #define STANDARD_A 440 // Hz
28 #define SAMPLES_PER_SEC 44100
29
30 const std::string pianoMap = "q2we4r5ty7u8i9op-[=zxdcfvgbnjmk
      ,.;/' " ;
31
32 std::vector<sf::Int16> makeSamplesFromString(GuitarString gs) {
33   std::vector<sf::Int16> samples;
34
35   gs.pluck();
36   int duration = 8;  // seconds
37   int i;
38   for (i= 0; i < SAMPLES_PER_SEC * duration; i++) {
39     gs.tic();
40     samples.push_back(gs.sample());
41   }
42
43   return samples;
44 }
45
46 int main() {
47   sf::RenderWindow window(sf::VideoMode(300, 200), "SFML Guitar
        Hero");
48   sf::Event event;
49   double freq;
50   const int KEYBOARD_LENGTH = 37 ;
51
52   std::vector<std::vector<sf::Int16>> sampleVector(
        KEYBOARD_LENGTH) ;
53   std::vector<sf::SoundBuffer> soundBufferVector(KEYBOARD_LENGTH
        ) ;
54   std::vector<sf::Sound> soundVector(KEYBOARD_LENGTH) ;
55
56   [&]() {
57     for(int i = 0; i < KEYBOARD_LENGTH; ++i) {
58       freq = STANDARD_A * pow(2,(i - 24)/12.0) ;
59       std::cout << freq << std::endl ;
60       sampleVector[i] = makeSamplesFromString(GuitarString(freq)
            ) ;
61       if (!soundBufferVector[i].loadFromSamples(&sampleVector[i
            ][0], sampleVector[i].size(), 2, SAMPLES_PER_SEC)) {
62         throw std::runtime_error("sf::SoundBuffer: failed to
```

```
            load from samples.") ;
63        }
64        soundVector[i].setBuffer(soundBufferVector[i]) ;
65      }
66    }() ;
67
68    while (window.isOpen()) {
69      while (window.pollEvent(event)) {
70        switch (event.type) {
71        case sf::Event::Closed:
72          window.close();
73          break;
74
75        case sf::Event::TextEntered:
76        {
77          int i = 0 ;
78          for (auto ch : pianoMap) {
79            if (event.text.unicode < 128 && ch == (char)event.text
                  .unicode) {
80              soundVector[i].play() ;
81            }
82            ++i ;
83          }
84        }
85          break ;
86        default:
87          break;
88        }
89
90        window.clear();
91        window.display();
92      }
93    }
94    return 0;
95  }
```

6.3 GuitarString.hpp

```
1  #ifndef GUITAR_STRING_HPP
2  #define GUITAR_STRING_HPP "GuitarString.hpp"
3
4  #include <SFML/Audio.hpp>
5  #include <vector>
6  #include "RingBuffer.hpp"
7
```

```
8  class GuitarString {
9
10 public:
11   const static int SAMPLE_RATE = 44100 ; // Hz
12
13   const static constexpr double ENERGY_DECAY_FACTOR = 0.996 ;
14
15   GuitarString(double frequency) ;
16
17   GuitarString(std::vector<sf::Int16> init) ;
18
19   void pluck() ;
20
21   void tic() ;
22
23   sf::Int16 sample() ;
24
25   int time() ;
26
27 private:
28   RingBuffer string ;
29
30   std::size_t bufferSize ;
31
32   int ticCount ;
33
34 } ; // class GuitarString
35
36 #endif // GUITAR_STRING_HPP
```

6.4 GuitarString.cpp

```
1  #include "GuitarString.hpp"
2  #include <cstdlib>
3  #include <ctime>
4  #include <cmath>
5
6  GuitarString::GuitarString(double frequency) : string(std::ceil(
       SAMPLE_RATE / frequency)) {
7    std::srand(std::time(0)) ;
8    ticCount = 0 ;
9  }
10
11 GuitarString::GuitarString(std::vector<sf::Int16> init) : string
       (init.size()){
```

```
12    for (auto sample : init) string.enqueue(sample) ;
13    std::srand(std::time(0)) ;
14    ticCount = 0 ;
15  }
16
17  void GuitarString::pluck() {
18    string.empty() ;
19    for (int i = 0; i < string.capacity(); ++i) {
20      string.enqueue(0 + (int16_t)rand()) ;
21    }
22  }
23
24  void GuitarString::tic() {
25    int16_t deletedValue = string.dequeue() ;
26    int16_t firstValue = string.peek() ;
27    //int16_t secondValue = string.peek(5) ;
28    string.enqueue(ENERGY_DECAY_FACTOR * ((deletedValue +
        firstValue) / 2)) ;
29    //string.enqueue(ENERGY_DECAY_FACTOR * ((0.9 * deletedValue +
        0.1 * firstValue))) ;
30    //string.enqueue(ENERGY_DECAY_FACTOR * ((deletedValue +
        firstValue + secondValue) / 3)) ;
31    //string.enqueue(ENERGY_DECAY_FACTOR * ((deletedValue * 0.19 +
         firstValue * 0.2+ secondValue * 0.6) )) ;
32    ++ticCount ;
33  }
34
35  sf::Int16 GuitarString::sample() {
36    return string.peek() ;
37  }
38
39  int GuitarString::time() {
40    return ticCount ;
41  }
```

6.5 RingBuffer.hpp

```
1   // Copyright 2018 Joel Savitz
2   #ifndef RING_BUFFER_HPP
3   #define RING_BUFFER_HPP
4
5   #include <cstdint>
6   #include <vector>
7
8   class RingBuffer {
```

```
 9   public:
10    explicit RingBuffer(int capacity) ;
11
12    int size() ;
13
14    int capacity() ;
15
16    bool isEmpty() ;
17
18    void empty() ;
19
20    bool isFull() ;
21
22    void enqueue(int16_t item) ;
23
24    int16_t dequeue() ;
25
26    int16_t peek(std::size_t offset = 0) ;
27
28   private:
29    std::vector<int16_t> _buffer ;
30
31    std::size_t _frontIndex ;
32
33    std::size_t _backIndex ;
34
35    std::size_t _size ;
36
37    std::size_t _capacity ;
38  } ;  // RingBuffer
39
40  #endif  // RING_BUFFER_HPP
```

6.6 RingBuffer.cpp

```
 1  // Copyright 2018 Joel Savitz
 2  #include "RingBuffer.hpp"
 3  #include <stdexcept>
 4  #include <utility>
 5
 6  RingBuffer::RingBuffer(int capacity) : _buffer(capacity, 0),
       _size(0) {
 7    if (capacity < 1) throw std::invalid_argument(
 8      "RingBuffer constructor: Capacity must be greater than 0!")
          ;
```

```
 9
10    _capacity = capacity ;
11
12    _frontIndex = _backIndex = 0 ;
13  }
14
15  int RingBuffer::size() {
16    return _size ;
17  }
18
19  int RingBuffer::capacity() {
20    return _capacity ;
21  }
22
23  bool RingBuffer::isEmpty() {
24    return _size == 0 ;
25  }
26
27  bool RingBuffer::isFull() {
28    return _size == _capacity ;
29  }
30
31  void RingBuffer::enqueue(int16_t item) {
32    if (isFull()) throw std::runtime_error(
33      "RingBuffer enqueue: Cannot enqueue to a full ring!") ;
34
35    _buffer.at(_backIndex) = item ;
36
37    ++_size ;
38    _backIndex = (_backIndex + 1) % _capacity ;
39  }
40
41  int16_t RingBuffer::dequeue() {
42    if (isEmpty()) throw std::runtime_error(
43      "RingBuffer dequeue: Cannot dequeue from an empty ring!") ;
44
45    int16_t item = _buffer.at(_frontIndex) ;
46
47    --_size ;
48    _frontIndex = (_frontIndex + 1) % _capacity ;
49
50    return std::move(item) ;
51  }
52
53  int16_t RingBuffer::peek(std::size_t offset) {
```

```
54    if (isEmpty()) throw std::runtime_error(
55      "RingBuffer peek: Cannot peek into an empty ring!") ;
56    if (offset > _size) throw std::out_of_range(
57      "RingBuffer peek: Cannot peek elements past the size of the
          buffer!") ;
58
59    int16_t item = _buffer.at((_frontIndex + offset) % _capacity)
        ;
60
61    return std::move(item) ;
62  }
63
64  void RingBuffer::empty() {
65    _frontIndex = _backIndex = _size = 0 ;
66  }
```

6.7 test.cpp

```
1   // Copyright 2018 Joel Savitz
2   #define BOOST_TEST_MODULE jsavitz
3   #include <boost/test/unit_test.hpp>
4
5   #include "RingBuffer.hpp"
6
7   BOOST_AUTO_TEST_CASE(RingBuffer_isEmpty) {
8     RingBuffer rb(2) ;
9
10
11    BOOST_REQUIRE(rb.isEmpty() == true) ;
12
13    rb.enqueue(10) ;
14
15    BOOST_REQUIRE(rb.isEmpty() == false) ;
16
17    rb.dequeue() ;
18
19    BOOST_REQUIRE(rb.isEmpty() == true) ;
20
21    rb.enqueue(10) ;
22    rb.enqueue(20) ;
23
24    BOOST_REQUIRE(rb.isEmpty() == false) ;
25
26    rb.dequeue() ;
27    rb.dequeue() ;
```

```
28
29    BOOST_REQUIRE(rb.isEmpty() == true) ;
30  }
31
32  BOOST_AUTO_TEST_CASE(RingBuffer_overload) {
33    RingBuffer rb(10) ;
34
35    for (int i = 0; i < 10; ++i) {
36      rb.enqueue(i) ;
37    }
38
39
40    BOOST_REQUIRE_THROW(rb.enqueue(11), std::runtime_error) ;
41
42    rb.dequeue() ;
43    BOOST_REQUIRE_NO_THROW(rb.enqueue(11)) ;
44
45    auto loadThreehundredSevens = [&](RingBuffer& buff) {
46      for (int i = 0; i < 300; i++) buff.enqueue(7) ;
47    } ;
48
49    BOOST_REQUIRE_THROW(loadThreehundredSevens(rb), std::
          runtime_error) ;
50  }
51
52  BOOST_AUTO_TEST_CASE(RingBuffer_peek) {
53    RingBuffer rb(20) ;
54
55    BOOST_REQUIRE_THROW(rb.peek(), std::runtime_error) ;
56
57    rb.enqueue(10) ;
58
59    BOOST_REQUIRE_NO_THROW(rb.peek()) ;
60
61    BOOST_REQUIRE(rb.peek() == 10) ;
62
63    rb.dequeue() ;
64
65    BOOST_REQUIRE_THROW(rb.peek(), std::runtime_error) ;
66  }
67
68  BOOST_AUTO_TEST_CASE(RingBuffer_size) {
69    RingBuffer rb(300) ;
70
71    BOOST_REQUIRE(rb.size() == 0) ;
```

```
72
73    rb.enqueue(1) ;
74
75    BOOST_REQUIRE(rb.size() == 1) ;
76
77    for (int i = 1 ; i < 300; ++i) {
78      rb.enqueue(1) ;
79    }
80
81    BOOST_REQUIRE(rb.size() == 300) ;
82
83    for (int i = 0 ; i < 100; ++i) {
84      rb.dequeue() ;
85    }
86    BOOST_REQUIRE(rb.size() == 200) ;
87    for (int i = 0 ; i < 200; ++i) {
88      rb.dequeue() ;
89    }
90    BOOST_REQUIRE(rb.size() == 0) ;
91  }
92
93  BOOST_AUTO_TEST_CASE(RingBuffer_isFull) {
94    RingBuffer rb(2) ;
95
96    BOOST_REQUIRE(rb.isFull() == false) ;
97
98    rb.enqueue(2) ;
99
100   BOOST_REQUIRE(rb.isFull() == false) ;
101
102   rb.enqueue(2) ;
103
104   BOOST_REQUIRE(rb.isFull() == true) ;
105
106   rb.dequeue() ;
107
108   BOOST_REQUIRE(rb.isFull() == false) ;
109   rb.dequeue() ;
110   BOOST_REQUIRE(rb.isFull() == false) ;
111 }
112
113 BOOST_AUTO_TEST_CASE(RingBuffer_dataFidelity) {
114   RingBuffer rb(10) ;
115
116   for (int i = 0; i < 5 ; ++i) {
```

```
117       rb.enqueue(i) ;
118     }
119     for (int i = 0; i < 5 ; ++i) {
120       rb.dequeue() ;
121     }
122
123     for (int i = 0; i < 10 ; ++i) {
124       rb.enqueue(i) ;
125     }
126
127     for (int i = 0; i < 10 ; ++i) {
128       BOOST_REQUIRE(rb.dequeue() == i) ;
129     }
130   }
```

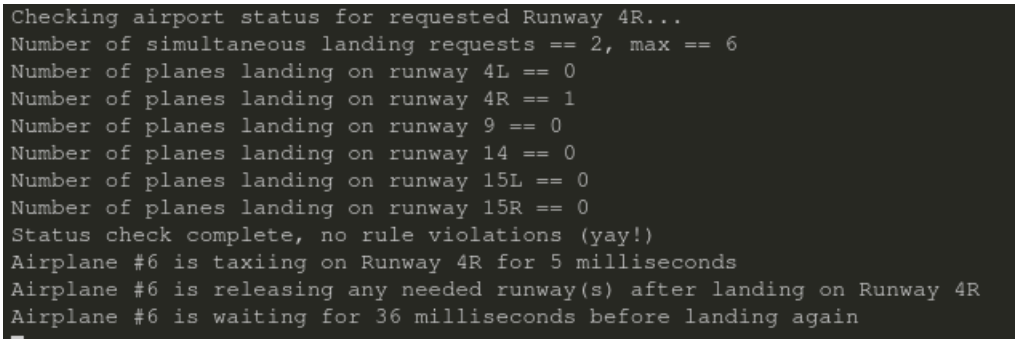


Figure 9: Airport simulation in progress

7 Airport Concurrency Simulation

This assignment was very different than the rest. I was given a functioning program that did not handle concurrent threads properly and I had to tweak it to properly handle concurrency. The program simulates a number of airplanes landing at Logan Airport on different runways, where some runways cannot be used at the same time as others because they physically overlap. This structure is used to represent different threads requiring exclusive use of different resources at different times.

Given rules stating which runways must be free before a runway can be used, I forced every plane landing on a runway to get a mutex lock on all prerequisite runways, and I enforced a total ordering on the calls to mutex lock and unlock in order to avoid a deadlock. I also used a condition variable to prevent more than six airplane threads from requesting a landing at the same time.

In order to complete this assignment, I had to familiarize myself with the C++ threading library and with basic concurrency problems. I gained experience debugging basic multi-threaded programs.

7.1 Makefile

```
1 CC = g++
2 CFLAGS = -c -Og -g -std=c++11
3 OBJ = Airplane.o Airport.o AirportRunways.o AirportServer.o
4 DEPS =
5 LIBS = -pthread
6 EXE = Airport-Sync
```

```
7
8  all: $(OBJ)
9   $(CC) $(OBJ) -o $(EXE) $(LIBS)
10
11 %.o: %.cpp $(DEPS)
12  $(CC) $(CFLAGS) -o $@ $<
13
14 clean:
15  rm -f $(OBJ) $(EXE)
```

7.2 Airport.cpp

```
1  /**
2  *  Airport driver program
3  */
4
5  #include <iostream>
6  #include <thread>
7  #include <vector>
8
9  #include "AirportServer.h"
10 #include "AirportRunways.h"
11 #include "Airplane.h"
12
13 using namespace std;
14
15 int main(void)
16 {
17  AirportServer as;
18
19  vector<thread> apths; // Airplane threads
20
21         // Create and launch the individual Airplane threads
22  for (int i = 1; i <= AirportRunways::NUM_AIRPLANES; i++)
23  {
24   Airplane* ap = new Airplane(i, &as);
25
26   apths.push_back(thread([](Airplane * ap){ap->land();}, ap));
27  }
28
29  // Wait for all Airplane threads to terminate (shouldn't happen
       !)
30  for (auto& th : apths)
31  {
32   th.join();
```

```
33   }
34
35   return 0;
36
37  } // end main
```

7.3 AirportServer.hpp

```
1   /**
2   *  AirportServer.h
3   *  This class defines the methods called by the Airplanes
4   */
5
6   #ifndef AIRPORT_SERVER_H
7   #define AIRPORT_SERVER_H
8
9   #include <mutex>
10  #include <random>
11
12  #include "AirportRunways.h"
13  #include <condition_variable>
14
15
16
17  class AirportServer
18  {
19  public:
20
21   /**
22   *  Default constructor for AirportServer class
23   */
24   AirportServer()
25   {
26    // ***** Initialize any Locks and/or Condition Variables here
         as necessary *****
27
28   } // end AirportServer default constructor
29
30
31     /**
32     *  Called by an Airplane when it wishes to land on a runway
33     */
34   void reserveRunway(int airplaneNum, AirportRunways::
        RunwayNumber runway);
35
```

```
36   /**
37   *  Called by an Airplane when it is finished landing
38   */
39   void releaseRunway(int airplaneNum, AirportRunways::
        RunwayNumber runway);
40
41
42  private:
43
44   // Constants and Random number generator for use in Thread
        sleep calls
45   static const int MAX_TAXI_TIME = 10; // Maximum time the
        airplane will occupy the requested runway after landing, in
        milliseconds
46   static const int MAX_WAIT_TIME = 100; // Maximum time between
        landings, in milliseconds
47
48   /**
49   *  Declarations of mutexes and condition variables
50   */
51   mutex runwaysMutex; // Used to enforce mutual exclusion for
        acquiring & releasing runways
52
53   /**
54    *  ***** Add declarations of your own Locks and Condition
         Variables here *****
55    */
56
57   static mutex mutex4L, mutex4R, mutex9, mutex14, mutex15L,
        mutex15R ;
58
59   static mutex mutexLandingRequests ;
60   static condition_variable cvLandingRequests ;
61
62  }; // end class AirportServer
63
64  #endif
```

7.4 AirportServer.cpp

```
1  #include <iostream>
2  #include <thread>
3  #include <condition_variable>
4
5  #include "AirportServer.h"
```

```
6
7  mutex AirportServer::mutex4L ;
8  mutex AirportServer::mutex4R ;
9  mutex AirportServer::mutex9 ;
10 mutex AirportServer::mutex14 ;
11 mutex AirportServer::mutex15L ;
12 mutex AirportServer::mutex15R ;
13
14 mutex AirportServer::mutexLandingRequests ;
15 condition_variable AirportServer::cvLandingRequests ;
16
17 /**
18 *  Called by an Airplane when it wishes to land on a runway
19 */
20 void AirportServer::reserveRunway(int airplaneNum,
       AirportRunways::RunwayNumber runway)
21 {
22  // Acquire runway(s)
23  {  // Begin critical region
24
25   //unique_lock<mutex> runwaysLock(runwaysMutex);
26
27   //runwaysMutex.lock() ;
28
29   /**
30    *  ***** Add your synchronization here! *****
31    */
32
33   AirportRunways::incNumLandingRequests() ;
34
35   unique_lock<mutex> landingRequestLock(mutexLandingRequests) ;
36
37   while(AirportRunways::getNumLandingRequests() >=
         AirportRunways::MAX_LANDING_REQUESTS) {
38    cvLandingRequests.wait(landingRequestLock) ;
39   }
40
41
42   switch(runway) {
43    case AirportRunways::RUNWAY_4L:
44     mutex4L.lock() ;
45     mutex15L.lock() ;
46     mutex15R.lock() ;
47     break ;
48    case AirportRunways::RUNWAY_4R:
```

```
49      mutex4R.lock() ;
50      mutex9.lock() ;
51      mutex15L.lock() ;
52      mutex15R.lock() ;
53      break ;
54     case AirportRunways::RUNWAY_9:
55      mutex4R.lock() ;
56      mutex9.lock() ;
57      mutex15R.lock() ;
58      break ;
59     case AirportRunways::RUNWAY_14:
60      mutex14.lock() ;
61      break ;
62     case AirportRunways::RUNWAY_15L:
63      mutex4L.lock() ;
64      mutex4R.lock() ;
65      mutex15L.lock() ;
66      break ;
67     case AirportRunways::RUNWAY_15R:
68      mutex4L.lock() ;
69      mutex4R.lock() ;
70      mutex9.lock() ;
71      mutex15R.lock() ;
72      break ;
73    }
74    //cout << "TEST2" << endl ;
75
76    // Check status of the airport for any rule violations
77    AirportRunways::checkAirportStatus(runway);
78
79    //runwaysLock.unlock();
80
81   } // End critical region
82
83   AirportRunways::decNumLandingRequests() ;
84
85   // obtain a seed from the system clock:
86   unsigned seed = std::chrono::system_clock::now().
        time_since_epoch().count();
87   std::default_random_engine generator(seed);
88
89   // Taxi for a random number of milliseconds
90   std::uniform_int_distribution<int> taxiTimeDistribution(1,
        MAX_TAXI_TIME);
91   int taxiTime = taxiTimeDistribution(generator);
```

```
92
93   {
94    lock_guard<mutex> lk(AirportRunways::checkMutex);
95
96    cout << "Airplane #" << airplaneNum << " is taxiing on Runway
          " << AirportRunways::runwayName(runway)
97      << " for " << taxiTime << " milliseconds\n";
98   }
99
100
101   std::this_thread::sleep_for(std::chrono::milliseconds(taxiTime)
        );
102
103  } // end AirportServer::reserveRunway()
104
105
106   /**
107    *  Called by an Airplane when it is finished landing
108    */
109  void AirportServer::releaseRunway(int airplaneNum,
       AirportRunways::RunwayNumber runway)
110  {
111   // Release the landing runway and any other needed runways
112   { // Begin critical region
113
114    //unique_lock<mutex> runwaysLock(runwaysMutex);
115
116    {
117     lock_guard<mutex> lk(AirportRunways::checkMutex);
118
119     cout << "Airplane #" << airplaneNum << " is releasing any
           needed runway(s) after landing on Runway "
120       << AirportRunways::runwayName(runway) << endl;
121    }
122
123    /**
124    *  ***** Add your synchronization here! *****
125    */
126
127    // Update the status of the airport to indicate that the
          landing is complete
128    AirportRunways::finishedWithRunway(runway);
129
130    switch(runway) {
131     case AirportRunways::RUNWAY_4L:
```

```
132      mutex4L.unlock() ;
133      mutex15L.unlock() ;
134      mutex15R.unlock() ;
135      break ;
136     case AirportRunways::RUNWAY_4R:
137      mutex4R.unlock() ;
138      mutex9.unlock() ;
139      mutex15L.unlock() ;
140      mutex15R.unlock() ;
141      break ;
142     case AirportRunways::RUNWAY_9:
143      mutex4R.unlock() ;
144      mutex9.unlock() ;
145      mutex15R.unlock() ;
146      break ;
147     case AirportRunways::RUNWAY_14:
148      mutex14.unlock() ;
149      break ;
150     case AirportRunways::RUNWAY_15L:
151      mutex4L.unlock() ;
152      mutex4R.unlock() ;
153      mutex15L.unlock() ;
154      break ;
155     case AirportRunways::RUNWAY_15R:
156      mutex4L.unlock() ;
157      mutex4R.unlock() ;
158      mutex9.unlock() ;
159      mutex15R.unlock() ;
160      break ;
161    }
162
163    //runwaysLock.unlock();
164    //runwaysMutex.unlock() ;
165
166   } // End critical region
167
168   // obtain a seed from the system clock:
169   unsigned seed = std::chrono::system_clock::now().
          time_since_epoch().count();
170   std::default_random_engine generator(seed);
171
172   // Wait for a random number of milliseconds before requesting
          the next landing for this Airplane
173   std::uniform_int_distribution<int> waitTimeDistribution(1,
          MAX_WAIT_TIME);
```

```
174   int waitTime = waitTimeDistribution(generator);
175
176   {
177    lock_guard<mutex> lk(AirportRunways::checkMutex);
178
179    cout << "Airplane #" << airplaneNum << " is waiting for " <<
          waitTime << " milliseconds before landing again\n";
180   }
181
182   std::this_thread::sleep_for(std::chrono::milliseconds(waitTime)
         );
183
184  } // end AirportServer::releaseRunway()
```