PS3 Recursive Graphics

What is the assignment?

The assignment is a variant of the Sierpienski Triangle where we draw it recursively outside of the base triangle using the class Triangle that we built and the fTree() function that we are supposed to use.

What I learned from this assignment:

1) That you can call a function by itself multiple times.
2) Some of the specifications that cpplint prefers
3) rand_r() is apparently safer than rand().

What I accomplished in this assignment:

1) I managed to add color to the tree triangle instead of the usual base color, which was red.
2) I also added a random color function that randomly changes the triangle

   tree's color to a distinct color which I enabled by pressing C for extra credit.
3) I also added a function to rotate the entire triangle tree to the right
   by 1 degree for extra credit. It will work by pressing R on the keyboard.
   Note: It will make the triangle disappear from the window after pressing R after
   10 times.
4) Additionally added a feature to close the window by using the escape keyboard and backspace keyboard button to make it potentially easier to close the window. - For Extra Credit.
5) Lastly, managed to make the child triangles at the end of each vertex of their parent triangles.

Any Issues I had in this assignment:

I did not know how to approach this initially as I could not figure out a reliable solution/equation to draw out the child triangles after recursion initially. The other issue I had was the child triangle's y

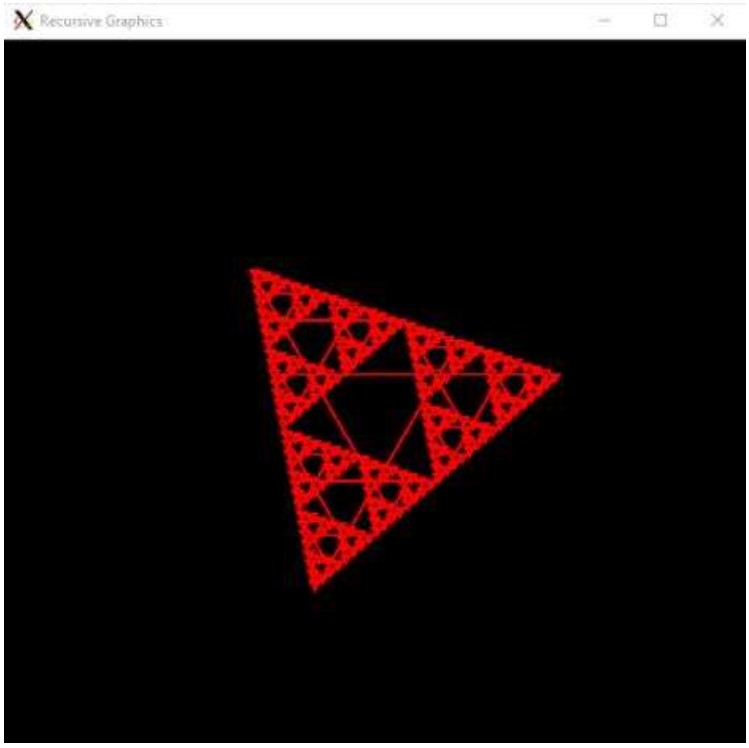
position was pretty whack, so I had to debug using cout statements and slowly narrow down the issue within my code and fix it.

Key Algorithms, Data Structures, or OO Designs used for this assignment:

Utilized vectors to help draw each child triangle for the base triangle and it made things easy to do when combined with the window.draw() function.

Screenshot(s):

```
ttruong@LAPTOP-93E6M94C:/mnt/c/Users/Tim/COMP2040/ps3$ make
g++ -Wall -Werror -pedantic -std=c++11 -lboost_unit_test_framework -c TFractal.cpp -o TFractal.o -lsfml-graphics -lsfml-window -lsfml-system
g++ -Wall -Werror -pedantic -std=c++11 -lboost_unit_test_framework -c Triangle.cpp -o Triangle.o -lsfml-graphics -lsfml-window -lsfml-system
g++ -Wall -Werror -pedantic -std=c++11 -lboost_unit_test_framework -o TFractal TFractal.o Triangle.o -lsfml-graphics -lsfml-window -lsfml-system
```



```
 1: CC = g++
 2: CFLAGS = -Wall -Werror -pedantic -std=c++11 -lboost_unit_test_framework
 3: OBJECTS = TFractal.o Triangle.o
 4: SFML = -lsfml-graphics -lsfml-window -lsfml-system
 5:
 6: all: TFractal
 7: TFractal: $(OBJECTS)
 8:         $(CC) $(CFLAGS) -o TFractal $(OBJECTS) $(SFML)
 9: Triangle.o: Triangle.cpp Triangle.h
10:         $(CC) $(CFLAGS) -c Triangle.cpp -o Triangle.o $(SFML)
11: TFractal.o: TFractal.cpp
12:         $(CC) $(CFLAGS) -c TFractal.cpp -o TFractal.o $(SFML)
13: clean:
14:         rm $(OBJECTS) TFractal
```

```
 1: // Copyright [2022] <Tim Truong>
 2: #include "Triangle.h"
 3:
 4: /*************************************************************
 5:  *Name: Tim Truong
 6:  *Course name: COMP.2040
 7:  *Assignment: PS3 - Recursive Graphics
 8:  *Instructor's name: Dr. James Daly
 9:  *Date: 2/21/22
10:  *Sources Of Help: SFML Website, Pre-Calculus Notebook
11: *************************************************************/
12:
13: const double VIEWPORT_WIDTH = 600;
14: const double VIEWPORT_HEIGHT = 600;
15:
16: void fTree(double L, int depth, std::vector<Triangle> *triVector,
17: sf::ConvexShape shape, double X, double Y);
18:
19: int main(int argc, char* argv[]) {
20:     double L;
21:     int N;
22:
23:     if (argc < 3 || argc > 3) {
24:         exit(1);
25:     } else {
26:         L = std::stod(argv[1]);
27:         N = std::stoi(argv[2]);
28:     }
29:     std::cout << N << std::endl;
30:
31:     sf::Vector2f size;
32:     sf::Vector2f window_size;
33:     size.x = VIEWPORT_WIDTH / 2;
34:     size.y = VIEWPORT_HEIGHT / 2;
35:     window_size.x = VIEWPORT_WIDTH;
36:     window_size.y = VIEWPORT_HEIGHT;
37:
38:     sf::ConvexShape shape;
39:     std::vector<Triangle> triVector;
40:
41:     int num = 1;
42:
43:     while ((L / 200) >= num) {
44:         window_size.x *= 1.35;
45:         window_size.y *= 1.35;
46:         size.x = window_size.x / 2.25;
47:         size.y = window_size.y / 2.25;
48:         num++;
49:     }
50:
51:     sf::RenderWindow window(sf::VideoMode(window_size.x, window_size.y),
52:     "Recursive Graphics");
53:
54:     fTree(L, N, &triVector, shape, size.x, size.y);
55:
56:     while (window.isOpen()) {
57:         sf::Event event;
58:         while (window.pollEvent(event)) {
59:             if (event.type == sf::Event::Closed) {
60:                 window.close();
61:             }
62:             if (sf::Keyboard::isKeyPressed(sf::Keyboard::Escape)) {
63:                 window.close();
64:             }
65:             if (sf::Keyboard::isKeyPressed(sf::Keyboard::Backspace)) {
```

```
66:                 window.close();
67:             }
68:             if (sf::Keyboard::isKeyPressed(sf::Keyboard::C)) {
69:                 int len = triVector.size();
70:                 for (int i = 0; i < len; i++) {
71:                     triVector[i].ChangeColor();
72:                 }
73:             }
74:             if (sf::Keyboard::isKeyPressed(sf::Keyboard::R)) {
75:                 int len = triVector.size();
76:                 for (int i = 0; i < len; i++) {
77:                     triVector[i].RotateTri();
78:                 }
79:             }
80:             if (sf::Keyboard::isKeyPressed(sf::Keyboard::M)) {
81:                 triVector[0].MixColor(triVector[0]);
82:                 triVector[1].ChangeColor();
83:                 triVector[2].ChangeColor();
84:                 triVector[3].ChangeColor();
85:                 triVector[4].ChangeColor();
86:             }
87:         }
88:         window.clear();
89:
90:         int len = triVector.size();
91:         for (int i = 0; i < len; i++) {
92:             window.draw(triVector[i]);
93:         }
94:
95:         window.display();
96:     }
97:
98:     return 0;
99: }
100:
101: void fTree(double L, int depth, std::vector<Triangle> *triVector,
102: sf::ConvexShape shape, double X, double Y) {
103:     if (depth < 0) {
104:         return;
105:     } else {
106:         double H = L * cos(M_PI / 6);
107:         sf::Vector2f size;
108:         size.x = X;
109:         size.y = Y;
110:         Triangle triangle(L, shape, size);
111:
112:         triVector->push_back(triangle);
113:
114:         fTree(L / 2.0, depth - 1, triVector, shape, size.x - (L / 2.0),
115:         size.y - ((2.0 / 3.0) * H));
116:         fTree(L / 2.0, depth - 1, triVector, shape, size.x +
117:         ((3.0 / 4) * L), size.y - ((1.0 / 6.0) * H));
118:         fTree(L / 2.0, depth - 1, triVector, shape, size.x -
119:         ((1.0 / 4) * L), size.y + ((5.0 / 6.0) * H));
120:     }
121: }
```

```
 1: // Copyright [2022] <Tim Truong>
 2: #pragma once
 3: #ifndef COMP2040_PS3_TRIANGLE_H_
 4: #define COMP2040_PS3_TRIANGLE_H_
 5:
 6: #include <iostream>
 7: #include <memory>
 8: #include <cmath>
 9: #include <vector>
10: #include <SFML/Graphics.hpp>
11: #include <SFML/Window.hpp>
12: #include <SFML/System.hpp>
13:
14: class Triangle : public sf::Transformable, public sf::Drawable {
15:  public:
16:     Triangle();
17:     Triangle(double length, sf::ConvexShape shape, sf::Vector2f Size);
18:     void ChangeColor(void);
19:     void MixColor(Triangle shape);
20:     void setColor(sf::Color color);
21:     void RotateTri(void);
22:     double getLen() const;
23:     sf::Vector2f getSize() const;
24:     sf::ConvexShape getShape() const;
25:     void setSize(sf::Vector2f Size);
26:     void setLen(double len);
27:     void setShape(sf::ConvexShape shape);
28:     std::vector<Triangle> *T;
29:  private:
30:     virtual void draw(sf::RenderTarget& target, sf::RenderStates states)
// NOLINT
31:     const;
32:     double length;
33:     sf::Vector2f size;
34:     sf::ConvexShape tri;
35: };
36:
37:
38: #endif  // COMP2040_PS3_TRIANGLE_H_
```

```
 1: // Copyright [2022] <Tim Truong>
 2: #include "Triangle.h"
 3:
 4: Triangle::Triangle() {
 5: }
 6:
 7: Triangle::Triangle(double len, sf::ConvexShape shape,
 8: sf::Vector2f Size) {
 9:     length = len;
10:     size = Size;
11:     tri = shape;
12:
13:     double H = ((sqrt(3.0) / 2.0) * length);
14:
15:     /*
16:     std::cout << "H: " << H << std::endl;
17:     std::cout << "L: " << length << std::endl;
18:     std::cout << "X: " << size.x << std::endl;
19:     std::cout << "Y: " << size.y << std::endl;
20:     */
21:
22:     tri.setPointCount(3);
23:     tri.setPoint(0, sf::Vector2f(size.x - (length / 2.0), size.y
24:     - (H / 3.0)));
25:     tri.setPoint(1, sf::Vector2f(size.x + (length / 2.0), size.y
26:     - (H / 3.0)));
27:     tri.setPoint(2, sf::Vector2f(size.x, size.y + ((2.0 / 3.0) * H)));
28:     tri.setFillColor(sf::Color::Transparent);
29:     tri.setOutlineThickness(2.5);
30:
31:     /*
32:     std::cout << "Ax: " << size.x - (length / 2.0) << std::endl;
33:     std::cout << "Ay: " << size.y - (H / 3.0) << std::endl;
34:     std::cout << "Bx: " << size.x + (length / 2.0) << std::endl;
35:     std::cout << "By: " << size.y - (H / 3.0) << std::endl;
36:     std::cout << "Cx: " << size.x << std::endl;
37:     std::cout << "Cy: " << size.y + ((2.0 / 3.0) * H) << std::endl;
38:
39:     std::cout << std::endl;
40:     */
41: }
42:
43: void Triangle::draw(sf::RenderTarget& target,
44: sf::RenderStates states) const {
45:     target.draw(tri, states);
46:     return;
47: }
48:
49: void Triangle::MixColor(Triangle shape) {
50:     shape.setColor(sf::Color::Blue);
51: }
52:
53: void Triangle::setColor(sf::Color color) {
54:     tri.setOutlineColor(color);
55: }
56:
57: void Triangle::ChangeColor() {
58:     unsigned int seedNum = time(NULL);
59:
60:     int num = rand_r(&seedNum) % 5;
61:
62:     if (num == 0) {
63:         tri.setOutlineColor(sf::Color::Blue);
64:     }
65:     if (num == 1) {
```

```
    66:         tri.setOutlineColor(sf::Color::White);
    67:     }
    68:     if (num == 2) {
    69:         tri.setOutlineColor(sf::Color::Magenta);
    70:     }
    71:     if (num == 3) {
    72:         tri.setOutlineColor(sf::Color::Yellow);
    73:     }
    74:     if (num == 4) {
    75:         tri.setOutlineColor(sf::Color::Green);
    76:     }
    77: }
    78:
    79: void Triangle::RotateTri() {
    80:     tri.rotate(1.0f);
    81: }
    82:
    83: sf::Vector2f Triangle::getSize() const {
    84:     return size;
    85: }
    86:
    87: double Triangle::getLen() const {
    88:     return length;
    89: }
    90:
    91: sf::ConvexShape Triangle::getShape() const {
    92:     return tri;
    93: }
    94:
    95: void Triangle::setSize(sf::Vector2f Size) {
    96:     size = Size;
    97: }
    98:
    99: void Triangle::setLen(double len) {
   100:     length = len;
   101: }
   102:
   103: void Triangle::setShape(sf::ConvexShape shape) {
   104:     tri = shape;
   105: }
```