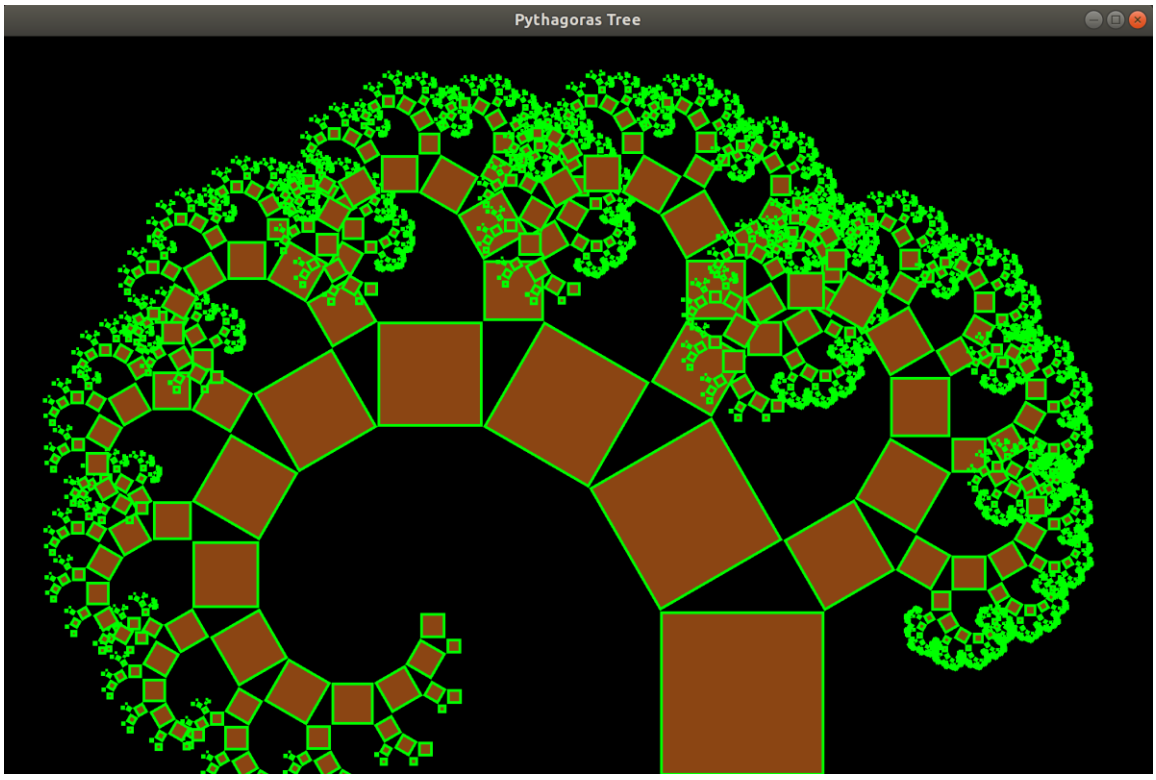
PS2 Recursive Graphics (Pythagoras tree)

In this assignment, I coded the Pythagoras tree using the SFML library and recursion. A key algorithm I used to design the tree was to set the origin of the left square to the bottom left and the position of the origin to the top left of the previously drawn square, then rotated +45 deg. The right side was similar which was that I set the origin of the right square to the bottom right and the positon to the top right of the previously drawn square, then rotated -45 deg. Anohter OO design that was central to this assignment were classes. I also implemented an additional functionality where the user could use the right arrow key to step through each recursive step of drawing the tree.

In this assignment I learned how to use recursion to draw recursive objects. The majority of the time I spent on this assignment was figuring out the recursuve part of the assignment because I had a hard time setting the different origins and positions correctly. I also didn't include the drawable class because I do not understand how to implent it, but I just got the adress of the window, so I could draw the shape within the function, which worked fine.



```
 1: CC = g++
 2: CFLAGS = -std=c++11 -c -g -Wall -Werror -pedantic
 3: LIBS = -lsfml-graphics -lsfml-window -lsfml-system
 4:
 5: all: tree
 6:
 7: tree: main.o PTree.o
 8:         $(CC) main.o PTree.o -o tree $(LIBS)
 9:
10: main.o: main.cpp
11:         $(CC) -c $(CFLAGS) main.cpp
12:
13: PTree.o: PTree.cpp PTree.hpp
14:         $(CC) -c $(CFLAGS) PTree.cpp PTree.hpp
15:
16: clean:
17:         rm *.o *.gch tree
```

```
    1: #include <SFML/Graphics.hpp>
    2: #include <SFML/Window.hpp>
    3: #include <cmath>
    4: #include <iostream>
    5: #include <time.h>
    6: #include <string>
    7:
    8: using namespace sf;
    9:
   10: class PTree{
   11: public:
   12:     PTree();
   13:
   14:     void pTree(RenderWindow &target, int size, Vector2f pos, Vector2f orig,
int deg, int iter);
   15:
   16: private:
   17:     ConvexShape shape;
   18: };
```

```
 1: #include "PTree.hpp"
 2:
 3: #define RT sqrt(2)
 4:
 5: PTree::PTree() {}
 6:
 7: void PTree::pTree(RenderWindow &target, int size, Vector2f pos, Vector2f ori
g, int deg, int iter) {
 8:     Vector2f Lp, Rp, origL(0, (size/2)*sqrt(3)), origR(size/2, size/2);
 9:     Color Brown(139, 69, 19);
10:
11:     shape.setPointCount(4);
12:     shape.setPoint(0, Vector2f(0, 0));
13:     shape.setPoint(1, Vector2f(0, size));
14:     shape.setPoint(2, Vector2f(size, size));
15:     shape.setPoint(3, Vector2f(size, 0));
16:
17:     shape.setPosition(pos);
18:     shape.setOrigin(orig);
19:     shape.setRotation(deg);
20:
21:     Rp = shape.getTransform().transformPoint(shape.getPoint(0));
22:     Lp = shape.getTransform().transformPoint(shape.getPoint(3));
23:
24:     shape.setOutlineColor(Color::Green);
25:     shape.setFillColor(Brown);
26:     shape.setOutlineThickness(-5);
27:     if (iter < 0)
28:         return;
29:     Rp = shape.getTransform().transformPoint(shape.getPoint(0));
30:     Lp = shape.getTransform().transformPoint(shape.getPoint(3));
31:
32:     target.draw(shape);
33:     //pTree(target, (size/2)*sqrt(3), Rp, origL, deg-30, iter-1);
34:     //pTree(target, size / 2, Lp, origR, deg+60, iter-1);
35:
36:     pTree(target, (size/2)*sqrt(3), Rp, origL, deg-30, iter-1);
37:     pTree(target, size/2, Lp, origR, deg+60, iter-1);
38: }
```

```
    1: #include "PTree.hpp"
    2:
    3: int main(int argc, char* argv[])
    4: {
    5:     int L, N, iter = 0;
    6:     L = atoi(argv[1]);
    7:     N = atoi(argv[2]);
    8:
    9:     std::cout << "Use right arrow to go to the next iteration." << std::endl
;
   10:
   11:     RenderWindow window (VideoMode(7*L, 4.5*L), "Pythagoras Tree");
   12:
   13:     //Vector2f pos(6*L/2-L/2, 4*L), orig(0, L);
   14:     Vector2f pos(6*L/1.5, 4.5*L), orig(0, L);
   15:
   16:     PTree rD;
   17:
   18:     while(window.isOpen())
   19:     {
   20:         Event event;
   21:         while(window.pollEvent(event))
   22:         {
   23:             if (event.type == Event::Closed || Keyboard::isKeyPressed(Keyboa
rd::Escape))
   24:                 window.close();
   25:             else if(Keyboard::isKeyPressed(Keyboard::Right)) {
   26:                 if (iter < N)
   27:                     iter++;
   28:             }
   29:         }
   30:
   31:         window.clear();
   32:         rD.pTree(window, L, pos, orig, 0, iter);
   33:         window.display();
   34:     }
   35:
   36:     return EXIT_SUCCESS;
   37: }
```