```
     1: #include "sierpinski.hpp"
     2: #include <SFML/Window.hpp>
     3: #include <SFML/System.hpp>
     4: #include <iostream>
     5: #include <cmath>
     6:
     7: int main(int argc, char* argv[])
     8: {
     9:     /*
    10:     if (argc < 3 || argc > 4)
    11:     {
    12:         std::cout << "Sierpinski [recursion-depth] [side-length]" << std::en
dl;
    13:         return -1;
    14:     }
    15:
    16:     int depth = atoi(argv[1]);
    17:     int side = atoi(argv[2]);
    18:      */
    19:
    20:     int depth;
    21:     int side;
    22:
    23:     std::cout << "Enter depth: ";
    24:     std::cin >> depth;
    25:     std::cout << "Enter side: ";
    26:     std::cin >> side;
    27:
    28:     if (argc < 3 || argc > 4)
    29:     {
    30:         std::cout << "Sierpinski [side-length] [recursion-depth]" << std::en
dl;
    31:         return -1;
    32:     }
    33:
    34:     std::cout << "Depth: " << depth << std::endl;
    35:     std::cout << "Side: " << side << std::endl;
    36:
    37:     if (depth < 0)
    38:     {
    39:         std::cout << "Depth should be greater than 0" << std::endl;
    40:     }
    41:
    42:     Sierpinski obj(side, depth);
    43:
    44:
    45:     int window_height = (int)(.5*sqrt(3.)*(float)side);
    46:
    47:     sf::RenderWindow window(sf::VideoMode(side, window_height), "Sierpinski"
);
    48:
    49:     window.setFramerateLimit(1);
    50:
    51:     while(window.isOpen())
    52:     {
    53:         sf::Event event;
    54:         while(window.pollEvent(event))
    55:         {
    56:             if(event.type == sf::Event::Closed)
    57:             {
    58:                 window.close();
```

```
59:             }
60:         }
61:         window.clear();
62:         window.draw(obj);
63:         window.display();
64:     }
65:
66:     return 0;
67: }
```

```
    1: #ifndef SIERPINSKI_H
    2: #define SIERPINSKI_H
    3: #include <SFML/Graphics.hpp>
    4: #include <vector>
    5:
    6:
    7: class Sierpinski : public sf::Drawable
    8: {
    9: public:
   10:     //Constructors
   11:
   12:     //define with side length and depth
   13:     Sierpinski(int size, int depth);
   14:
   15:     //set up a triangle
   16:     Sierpinski(float x1, float y1, float x2, float y2, float x3, float y3, i
nt depth, float size, float height);
   17:
   18:     //Destructor
   19:     ~Sierpinski();
   20:
   21:     //Functions
   22:
   23:
   24:
   25: private:
   26:
   27:     void virtual draw(sf::RenderTarget &target, sf::RenderStates states) con
st;
   28: //    float sierpinski_depth;
   29: //    float sierpinski_size;
   30: //    float sierpinski_height;
   31:
   32: };
   33: #endif
```

```
    1: #include "sierpinski.hpp"
    2: #include <cmath>
    3: #include <iostream>
    4:
    5: std::vector <sf::ConvexShape> triangle_vector;
    6: int count = 0;
    7:
    8: Sierpinski::Sierpinski(int size, int depth)
    9: {
   10:     float sierpinski_height = size * sqrt(3)/2;
   11: //    sierpinski_depth = depth;
   12:  //   sierpinski_size = size;
   13:
   14:
   15:     sf::Vector2f p1, p2, p3;
   16:     p1.x = size/2;
   17:     p1.y = 0;
   18:     p2.x = 0;
   19:     p2.y = sierpinski_height;
   20:     p3.x = size;
   21:     p3.y = sierpinski_height;
   22:
   23:     //Set Initial Triangle
   24:     sf::ConvexShape initial_triangle;
   25:     initial_triangle.setPointCount(3);
   26:     initial_triangle.setPoint(0, p1);
   27:     initial_triangle.setPoint(1, p2);
   28:     initial_triangle.setPoint(2, p3);
   29:     initial_triangle.setFillColor(sf::Color::Yellow);
   30:
   31:     triangle_vector.push_back(initial_triangle);
   32:
   33:     if (depth > 0)
   34:     {
   35:         triangle_vector.pop_back();
   36:         Sierpinski(p1.x, p1.y, p2.x, p2.y, p3.x, p3.y, depth, size, sierpins
ki_height);
   37:     }
   38:
   39:
   40:
   41: }
   42:
   43:
   44: Sierpinski::Sierpinski(float x1, float y1, float x2, float y2, float x3, flo
at y3, int depth,
   45:                        float size, float height)
   46: {
   47:     /*if (depth not reached)
   48:     {
   49:         child (top, midleft (w/4) (h/2), midright (3w/4)(h/2))
   50:         child (midleft, left, middle (top w/2) (h))
   51:         child (midright, middle, right)
   52:     }
   53:      else
   54:      {
   55:         build triangle with current data
   56:      }
   57:      */
   58:
   59:
```

```
   60:
   61:      if (depth > 0)
   62:      {
   63:         depth--;
   64:
   65:         Sierpinski(x1, y1, size/4, height/2, ((3 * size)/4), height/2, depth
, size/4, height/2);
   66:         Sierpinski(size/4, height/2, x2, y2, x1, height, depth, size/4, heig
ht/2);
   67:         Sierpinski(((3 * size)/4), height/2, x1, height, x3, y3, depth, size
/4, height/2);
   68:
   69:      }
   70:      else
   71:      {
   72:          sf::Vector2f p1, p2, p3;
   73:          p1.x = x1;
   74:          p1.y = y1;
   75:          p2.x = x2;
   76:          p2.y = y2;
   77:          p3.x = x3;
   78:          p3.y = y3;
   79:
   80:         sf::ConvexShape triangle;
   81:         triangle.setPointCount(3);
   82:         triangle.setPoint(0, p1);
   83:         triangle.setPoint(1, p2);
   84:         triangle.setPoint(2, p3);
   85:         triangle.setFillColor(sf::Color::Yellow);
   86:          triangle.setOutlineColor(sf::Color::Red);
   87:          triangle.setOutlineThickness(2);
   88:
   89:         triangle_vector.push_back(triangle);
   90:          count++;
   91:      }
   92: }
   93:
   94:
   95: Sierpinski::~Sierpinski()
   96: {
   97:
   98: }
   99:
  100:
  101: void Sierpinski::draw(sf::RenderTarget &target, sf::RenderStates states) con
st
  102: {
  103:     for(unsigned int i = 0; i < triangle_vector.size(); i++)
  104:     {
  105:         target.draw(triangle_vector.at(i), states);
  106:     }
  107: }
  108:
  109:
  110:
  111:
```

```
    1: #ifndef ORIGINAL_H
    2: #define ORIGINAL_H
    3: #include <SFML/Graphics.hpp>
    4: #include <vector>
    5:
    6:
    7: class original : public sf::Drawable
    8: {
    9: public:
   10:     //Constructors
   11:
   12:     //define with side length and depth
   13:     original(int size, int depth);
   14:
   15:     //set up a shape
   16:     original(float x1, float y1, float x2, float y2, float x3, float y3, flo
at x4, float y4,
   17:              int depth, float size, float height);
   18:
   19:     //Destructor
   20:     ~original();
   21:
   22:     //Functions
   23:
   24:
   25:
   26: private:
   27:
   28:     void virtual draw(sf::RenderTarget &target, sf::RenderStates states) con
st;
   29:     //    float original_depth;
   30:     //    float original_size;
   31:     //    float original_height;
   32:
   33: };
   34: #endif
```

```
    1: #include "original.hpp"
    2: #include <SFML/Window.hpp>
    3: #include <cmath>
    4: #include <iostream>
    5:
    6: std::vector <sf::ConvexShape> square_vector;
    7: int count = 0;
    8:
    9: original::original(int size, int depth)
   10: {
   11:     float original_height = size/2;
   12:     //    original_depth = depth;
   13:     //   original_size = size;
   14:
   15:
   16:     sf::Vector2f p1, p2, p3, p4;
   17:     p1.x = size *.4;
   18:     p1.y = size *.35;
   19:     p2.x = size*.4;
   20:     p2.y = size*.55;
   21:     p3.x = size*.6;
   22:     p3.y = size*.55;
   23:     p4.x = size*.6;
   24:     p4.y = size*.35;
   25:
   26:     //Set Initial Triangle
   27:     sf::ConvexShape initial_square;
   28:     initial_square.setPointCount(4);
   29:     initial_square.setPoint(0, p1);
   30:     initial_square.setPoint(1, p2);
   31:     initial_square.setPoint(2, p3);
   32:     initial_square.setPoint(3, p4);
   33:     initial_square.setFillColor(sf::Color::Yellow);
   34:
   35:     square_vector.push_back(initial_square);
   36:
   37:     original(p1.x, p1.y, p2.x, p2.y, p3.x, p3.y, p4.x, p4.y, depth, size, or
iginal_height);
   38:
   39: }
   40:
   41:
   42: original::original(float x1, float y1, float x2, float y2, float x3, float y
3, float x4, float y4,
   43:                     int depth, float size, float height)
   44: {
   45:         /*if (depth not reached)
   46:          {
   47:          child (top, midleft (w/4) (h/2), midright (3w/4)(h/2))
   48:          child (midleft, left, middle (top w/2) (h))
   49:          child (midright, middle, right)
   50:          }
   51:          else
   52:          {
   53:          build triangle with current data
   54:          }
   55:          */
   56:
   57:
   58:
   59:         if (depth > 0)
```

```
   60:         {
   61:             depth--;
   62:
   63:             original(x1/4, y1/4, x2/4, y2/4, x3/4, y3/4, x4/4, y4/4, depth,
size, height);
   64:
   65:             original((x1+size*.5)/1.9, y1/4, (x2+size*.5)/1.9, y2/4, (x3+siz
e*.5)/2.1, y3/4, (x4+size*.5)/2.1, y4/4, depth, size, height);
   66:
   67:             original((x1 * 2.2), y1/4, (x2 * 2.2), y2/4, (x3 * 1.56), y3/4,
(x4 * 1.56), y4/4,
   68:                     depth, size, height);
   69:
   70:             original(x1/4, y1/4 + size*.63, x2/4, y2/4 + size*.63, x3/4, y3/
4 + size*.63, x4/4,
   71:                     y4/4 + size*.63, depth, size, height);
   72:
   73:             original((x1+size*.5)/1.9, y1/4 + size*.63, (x2+size*.5)/1.9, y2
/4 + size*.63, (x3+size*.5)/2.1, y3/4 + size*.63, (x4+size*.5)/2.1, y4/4 + size*.63
, depth, size, height);
   74:
   75:             original((x1 * 2.2), y1/4 + size*.63, (x2 * 2.2), y2/4 + size*.6
3, (x3 * 1.56), y3/4 + size*.63, (x4 * 1.56), y4/4 + size*.63, depth, size, height)
;
   76:
   77:             original(x1/4, y1/4 + size*.33, x2/4, y2/4 + size*.33, x3/4, y3/
4 + size*.33, x4/4,
   78:                     y4/4 + size*.33, depth, size, height);
   79:
   80:             original((x1 * 2.2), y1/4 + size*.33, (x2 * 2.2), y2/4 + size*.3
3, (x3 * 1.56), y3/4 + size*.33, (x4 * 1.56), y4/4 + size*.33, depth, size, height)
;
   81:
   82:
   83:         }
   84:         else
   85:         {
   86:             sf::Vector2f p1, p2, p3, p4;
   87:             p1.x = x1;
   88:             p1.y = y1;
   89:             p2.x = x2;
   90:             p2.y = y2;
   91:             p3.x = x3;
   92:             p3.y = y3;
   93:             p4.x = x4;
   94:             p4.y = y4;
   95:
   96:             //Set Initial Triangle
   97:             sf::ConvexShape initial_square;
   98:             initial_square.setPointCount(4);
   99:             initial_square.setPoint(0, p1);
  100:             initial_square.setPoint(1, p2);
  101:             initial_square.setPoint(2, p3);
  102:             initial_square.setPoint(3, p4);
  103:             initial_square.setFillColor(sf::Color::Yellow);
  104:
  105:             square_vector.push_back(initial_square);
  106:         }
  107: }
  108:
  109:
```

```
110: original::~original()
111: {
112:
113: }
114:
115:
116: void original::draw(sf::RenderTarget &target, sf::RenderStates states) const
117: {
118:     for(int i = 0; i < square_vector.size(); i++)
119:     {
120:         target.draw(square_vector.at(i), states);
121:     }
122: }
123:
124:
125:
126:
```

```
 1: C=g++ -g -Wall --std=c++98 -Werror
 2: E=.cpp
 3: O=original.o maine.o
 4: S=sierpinski.o main.o
 5: P=sierpinski
 6: Q=original
 7: SFML= -lsfml-graphics -lsfml-window -lsfml-system -lsfml-audio
 8: all: $(P) $(Q)
 9: $(P):$(S)
10:         $(C) -o $(P) $(S) $(SFML)
11:
12: $(Q):$(O)
13:         $(C) -o $(Q) $(O) $(SFML)
14:
15: $(E).o:
16:         $(C) -c $<  -o $@
17:
18: clean:
19:         rm $(O) $(P) $(Q) $(S)
20:
21: .PHONY: clean
```