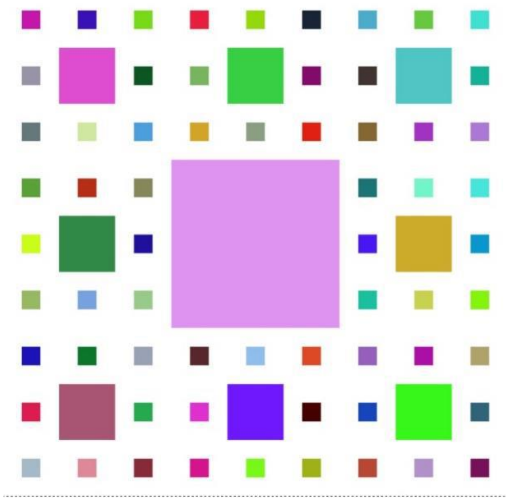
Jeff Registre

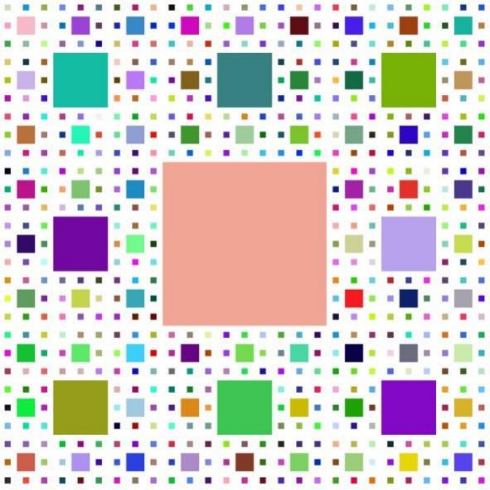
ID: 804007874

Task 1

WebGL Sierpenski Carpet

This code project is called the Sierpenski Carpet. Its purpose is modeling the Sierpenski Carpet fractal and render it on the GPU through the use of WebGL. The structure of this project is as follows; An HTML page acts as the GUI on which to display the application, and a JavaScript application calculates and calls for GPU rendering using shaders written in ESSL. Not much is actually done on the GPU for this project. The JavaScript application takes in input from the HTML page for the number of subdivisions for the fractal: it calculates everything and simply passes all the data to the GPU for rendering. Below is an image of the carpet with 3 and 5 sub subdivisions from left to right.

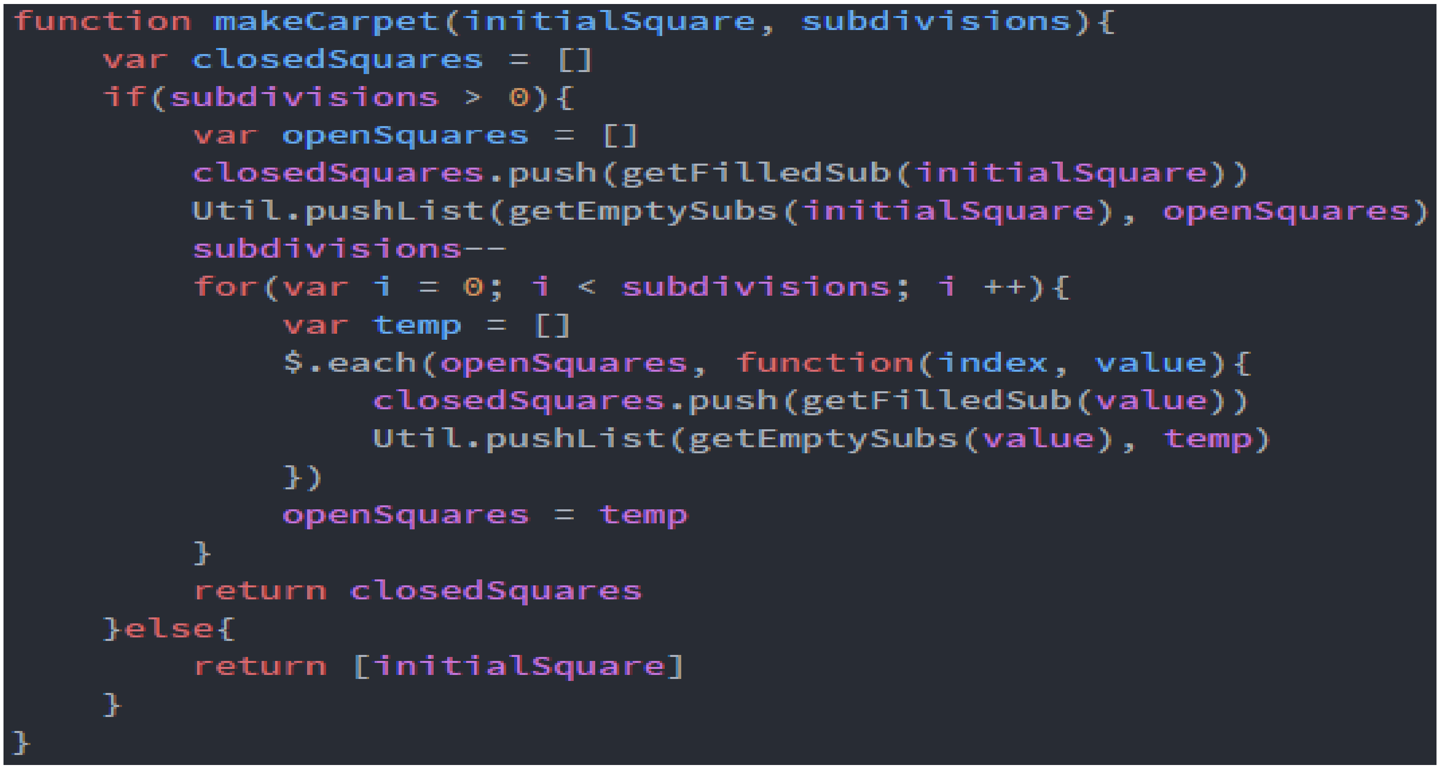




As you can see from the images, the amount of squares grows extremely quick. So much so that when divisions exceeds 10, the browser likely crashes the program.
The number of squares on screen for any given subdivisions according to my implementation can be calculated with the following piecewise defined equation:

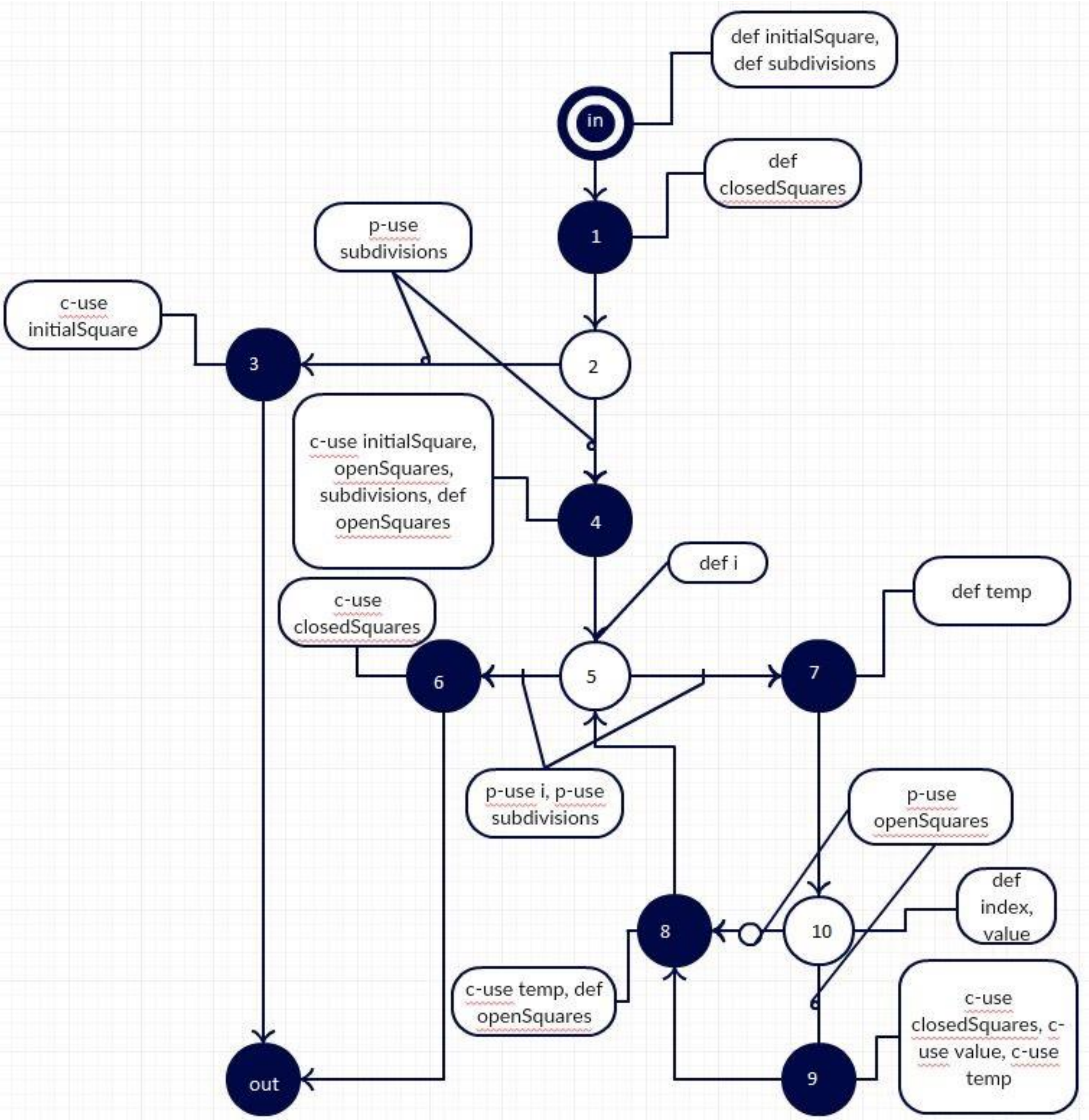
$$A(n) = \begin{cases} 1, & x \leq 1 \\ (A_{n-1} * 8) + 1, & x > 1 \end{cases}$$

The method I've chosen for I've selected for testing is the createCarpet method. This method returns a list of squares making a Sierpenski carpet. The parameters it takes are; subdivisions which drives how many squares will be in the carpet, and InitialSquare which is the Square it'll use to begin the operations. The method begins by analyzing the value of the subdivisions parameter: if I is 0, then it simply returns an array with the initial square as the sole value. If the subdivisions value is 1, then it gets the sub square of the initial square and returns that in an array. So for values of 0 and 1, there is still just one square being rendered. With values greater than one, it’ll do the same step as the value being 1 but then it’ll get all the “open squares” and continue to act upon them making more and more squares until it finally returns. That is pretty much it: simple but a little bit complex. A screenshot of this code is provided below.



```
function makeCarpet(initialSquare, subdivisions){
    var closedSquares = []
    if(subdivisions > 0){
        var openSquares = []
        closedSquares.push(getFilledSub(initialSquare))
        Util.pushList(getEmptySubs(initialSquare), openSquares)
        subdivisions--
        for(var i = 0; i < subdivisions; i ++){
            var temp = []
            $.each(openSquares, function(index, value){
                closedSquares.push(getFilledSub(value))
                Util.pushList(getEmptySubs(value), temp)
            })
            openSquares = temp
        }
        return closedSquares
    }else{
        return [initialSquare]
    }
}
```

Control flow graph


def initialSquare, def subdivisions
in
def closedSquares
1
p-use subdivisions
c-use initialSquare
3
2
c-use initialSquare, openSquares, subdivisions, def openSquares
4
def i
def temp
c-use closedSquares
6
5
7
p-use i, p-use subdivisions
p-use openSquares
def index, value
8
10
c-use temp, def openSquares
c-use closedSquares, c-use value, c-use temp
out
9

A) Equivalence classes

Parameter	Equivalence Class	Representative
initialSquare	1.1: Valid Square	Square(0,0,2)
subdivisions	2.1: 0 ≤ subdivisions ≤ 5	4
InitialSquare	1.a: Not Square object	Circle
subdivisions	2.a: < 0, > 5	-2

B) Test cases

Test Case ID	initialSquare	subdivisions	Exp.Result(count)
TF#1	Square(0,0,2)	4	585
TF#2	Circle	-1	ERROR

C) Boundary value analysis

Parameter	Boundary Values	Test Case ID
initialSquare	Valid Square	TF#1
subdivisions	(0, 5)	TF#2

Task 2

Did not get the chance to present yet.

Task 3

My class is non modal. The closes comparison to my class would be like a calculator. The methods simply take in input and send out a result. The object doesn't keep any state.

Task 4

A) Method Scope Test

a. Category Partition test

The methods of this class have no side effect. The class keeps absolutely no state.

b. Data flow test

TF#1	Square(0,0,2**)**	**4**	**585**

c. Multiple Condition Coverage

TF#1	Square(0,0,2)	4	585
TF#2	Square(0,0,2)	0	1
Tf#3	Square(0,0,2)	1	1

d. Boundary interior analysis

TF#1	Square(0,0,2)	2	585

TF#2	Square(0,0,2)	0	1
Tf#3	Square(0,0,2)	1	1

B) **Class Scope Test**

Calling the methods in random order as the class is non modal.

getFilledSub (square(0,0,4))

makeCarpet(square(0,0,1))

getEmptySubs(square(0.1, 0.2,10))