University of California, Santa Cruz

Experiment 5:

3D Graphics

Dominic Berardi & Nathan Prieto

CMPM 169: Creative Coding

Modes

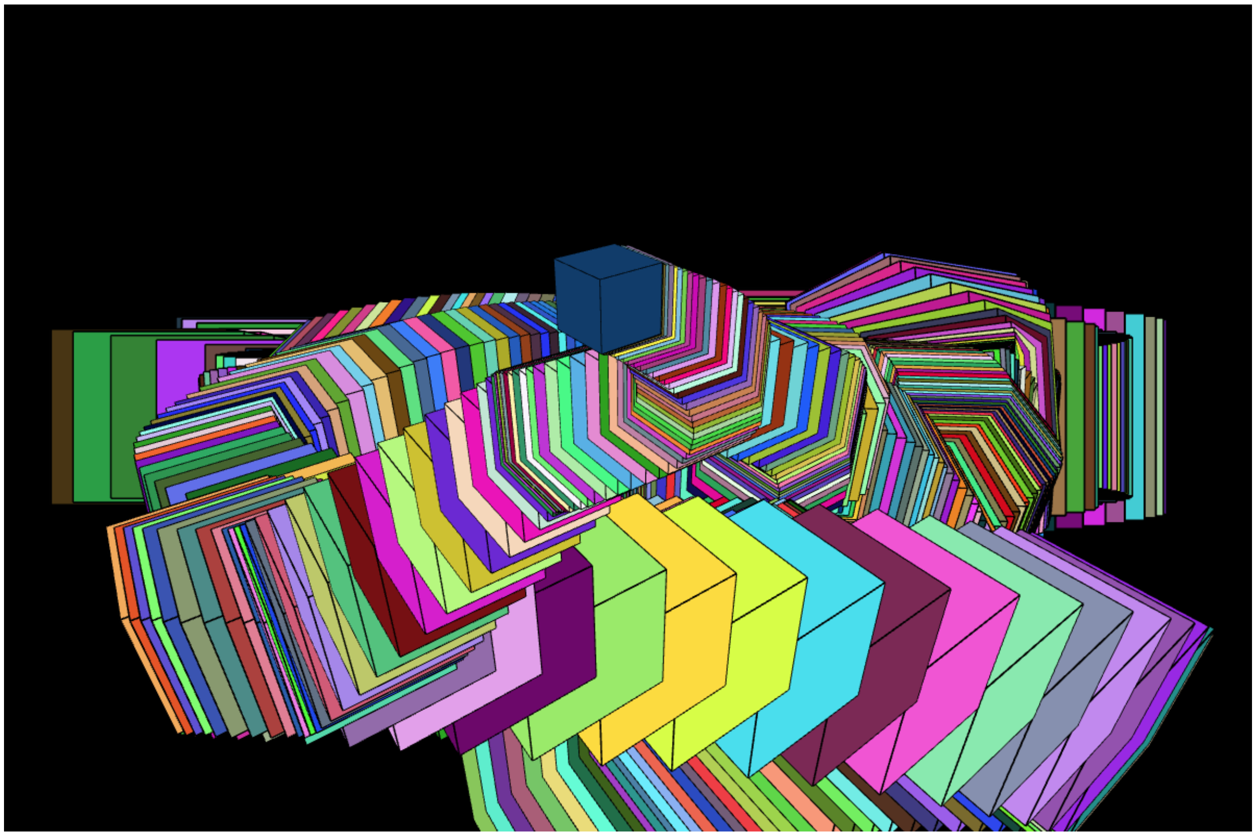
Due Date 2/14/2023

3D Auto Snake

This experiment uses WebGL to create a version of Nokia Snake that is fully 3D and plays itself.
Playable Link:
https://ncprieto.github.io/Experiment-5-CMPM-169-3D-Snake-Nathan-Prieto-Dominic-Berardi/
Sketch.js code: https://github.com/domberardi10/cmpm169/blob/master/experiment5/js/sketch.js

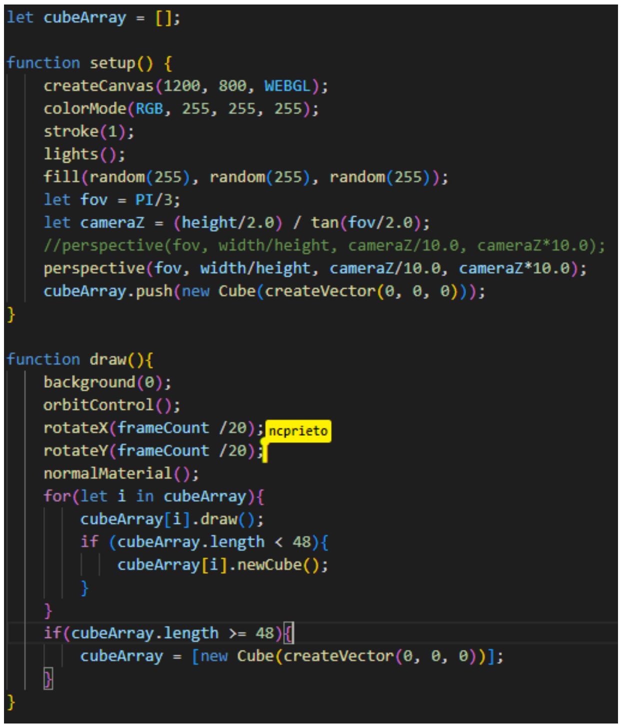
Step 1: Imitate

For this project, we started out by choosing a reference that we both agreed upon would be a good starting point. We chose a project that randomly generated a cityscape with cubes that has a really interesting feature where it manipulates the perspective of the camera that adds depth and scale to the cubes. https://openprocessing.org/sketch/857413 This project was made with a library that neither of us have worked with, called P3D, which featured strong types and a few different methods from P5 JS. We attempted to convert the reference code from this library to P5 but found no success as the original author's code was extremely abstract and difficult to understand. During our attempts at debugging the reference, we decided to just focus on the very basic functions the original author had used such as creating a cube and then setting it's fill color. We first attempted to draw a single cube with a random fill color and came up with a new cube each frame that had a random fill color rather the single cube with a set fill color that we desired.

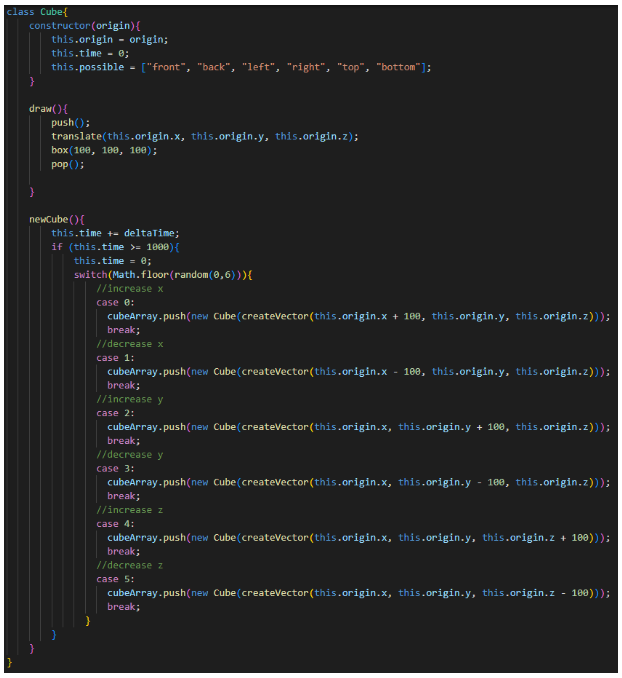


We realized that we were drawing a new cube each frame instead of drawing a background over the previous frame to remove the cube already drawn. Choosing a set fill color before drawing any cube, drawing a new background each frame, then drawing a single cube gave us the result we wanted.

After we had a singular, colored cube in space, we brainstormed how it could be made more interesting. Building off of the original generative piece we attempted to rebuild, we opted for a generative approach as well. We also used the perspective function that was in the other piece as well as a function called normalMaterial(), which sets the material to 3D objects as either red, blue or green, corresponding to each axis (it just looked cool!). Every second, a new cube is built off of an existing one from one of each cube's faces, resulting in an exponential growth of new cubes outwards, creating a unique voxel shape.

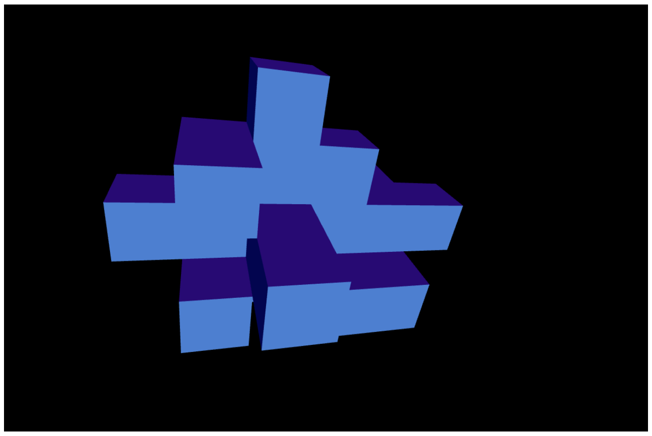


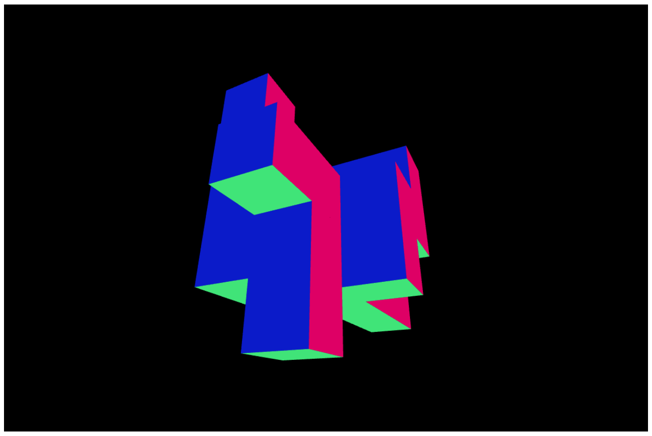
```
let cubeArray = [];

function setup() {
    createCanvas(1200, 800, WEBGL);
    colorMode(RGB, 255, 255, 255);
    stroke(1);
    lights();
    fill(random(255), random(255), random(255));
    let fov = PI/3;
    let cameraZ = (height/2.0) / tan(fov/2.0);
    //perspective(fov, width/height, cameraZ/10.0, cameraZ*10.0);
    perspective(fov, width/height, cameraZ/10.0, cameraZ*10.0);
    cubeArray.push(new Cube(createVector(0, 0, 0)));
}

function draw(){
    background(0);
    orbitControl();
    rotateX(frameCount /20);
    rotateY(frameCount /20);
    normalMaterial();
    for(let i in cubeArray){
        cubeArray[i].draw();
        if (cubeArray.length < 48){
            cubeArray[i].newCube();
        }
    }
    if(cubeArray.length >= 48){
        cubeArray = [new Cube(createVector(0, 0, 0))];
    }
}
```

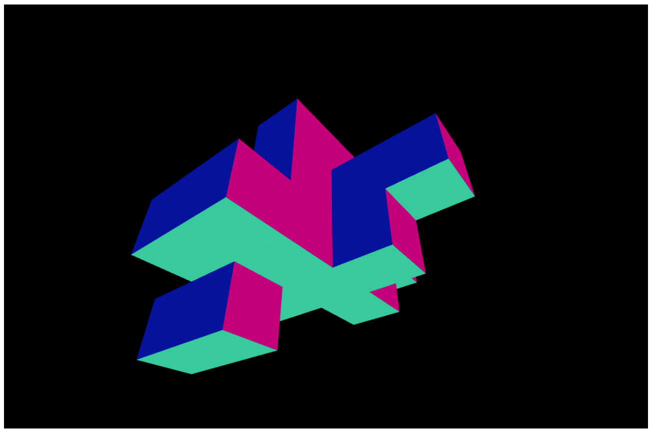


```
class Cube{
    constructor(origin){
        this.origin = origin;
        this.time = 0;
        this.possible = ["front", "back", "left", "right", "top", "bottom"];
    }

    draw(){
        push();
        translate(this.origin.x, this.origin.y, this.origin.z);
        box(100, 100, 100);
        pop();

    }

    newCube(){
        this.time += deltaTime;
        if (this.time >= 1000){
            this.time = 0;
            switch(Math.floor(random(0,6))){
                //increase x
                case 0:
                  cubeArray.push(new Cube(createVector(this.origin.x + 100, this.origin.y, this.origin.z)));
                  break;
                //decrease x
                case 1:
                  cubeArray.push(new Cube(createVector(this.origin.x - 100, this.origin.y, this.origin.z)));
                  break;
                //increase y
                case 2:
                  cubeArray.push(new Cube(createVector(this.origin.x, this.origin.y + 100, this.origin.z)));
                  break;
                //decrease y
                case 3:
                  cubeArray.push(new Cube(createVector(this.origin.x, this.origin.y - 100, this.origin.z)));
                  break;
                //increase z
                case 4:
                  cubeArray.push(new Cube(createVector(this.origin.x, this.origin.y, this.origin.z + 100)));
                  break;
                //decrease z
                case 5:
                  cubeArray.push(new Cube(createVector(this.origin.x, this.origin.y, this.origin.z - 100)));
                  break;
            }
        }
    }
}
```

A few issues arose from the initial test of this. Namely, the exponential growth would quickly grow out of hand, resulting in intense computer lag. Our fix for this was to cap out the number of cubes that are created, before wiping the slate clean after the cap was reached. Another, more tricky issue, was that of cubes drawing inside of existing cubes– there was no check for whether a possible origin point had a cube drawn there already. The semi-fix for this is to instead have a list of still-possible directions for each cube to draw in. Cubes will still overlap onto each other, but they won't redraw in a direction they already have.



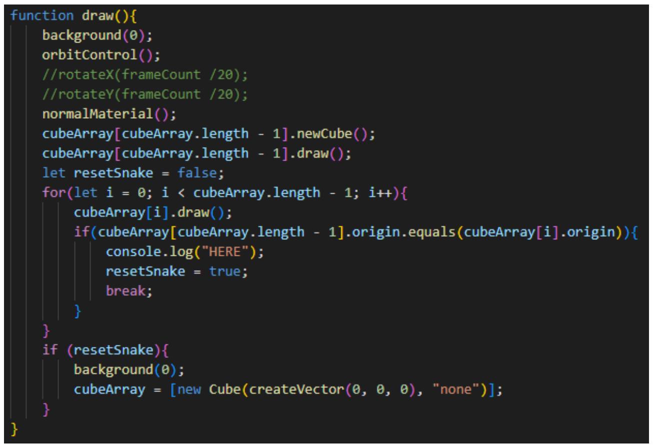




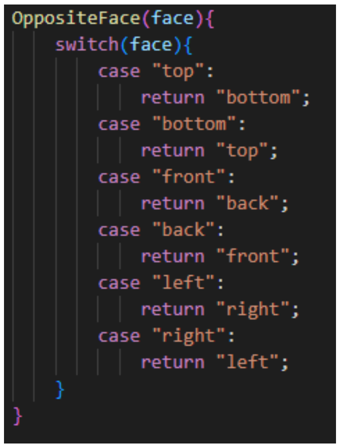
Step 2: Integrate

As for integrating existing methods into our project, we attempted to create a game of 3D Snake that plays itself. We first tackled this by attempting to create cubes that spawn every second and are positioned alongside a face of the cube it was spawned from. This resulted in cubes spawning in directions in which they did not come from, however this resulted in some bugs that resulted in unwanted behaviors. We noticed that we weren't excluding the opposite direction in which the cube was spawned from. So if a cube was spawned from the original cube's front face then it would exclude spawning the successive cube towards its back face. We then created a simple check that compares the position of the cube that was spawned to any of the existing cubes and if their positions were the same then it would reset the entire canvas and place one cube at the origin.

As can be seen from the draw() function below, the only cube that actually spawns a new cube is the one at the front of the Snake– in code, it's at the back of the array. The comparison of origin points to ensure no overlapping is occurring is in the loop that draws the rest of the existing cubes. The OppositeFace() function simply returns exactly what it says– what is the opposite side of the current cube face? This is needed in order to keep the cubes being drawn from not going into reverse and accidentally overlapping backwards.

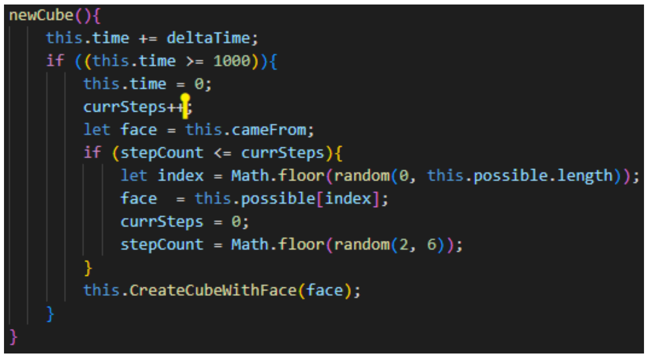


```
function draw(){
    background(0);
    orbitControl();
    //rotateX(frameCount /20);
    //rotateY(frameCount /20);
    normalMaterial();
    cubeArray[cubeArray.length - 1].newCube();
    cubeArray[cubeArray.length - 1].draw();
    let resetSnake = false;
    for(let i = 0; i < cubeArray.length - 1; i++){
        cubeArray[i].draw();
        if(cubeArray[cubeArray.length - 1].origin.equals(cubeArray[i].origin)){
            console.log("HERE");
            resetSnake = true;
            break;
        }
    }
    if (resetSnake){
        background(0);
        cubeArray = [new Cube(createVector(0, 0, 0), "none")];
    }
}
```

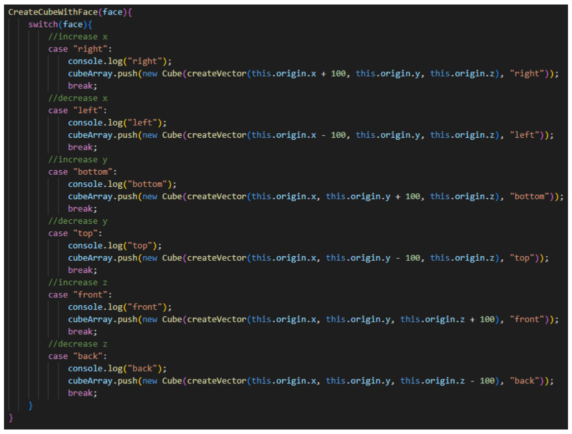


```
OppositeFace(face){
    switch(face){
        case "top":
            return "bottom";
        case "bottom":
            return "top";
        case "front":
            return "back";
        case "back":
            return "front";
        case "left":
            return "right";
        case "right":
            return "left";
    }
}
```

The other big change to our existing code is the addition of what is called the "step count". A "step" in this context is when a new cube is drawn, which occurs once every second. The step count forces the Snake to move in a chosen direction for at least 2 steps and no more than 5 steps before it can select another direction to move in (it is possible to choose the same direction). This occurs in the newCube() function. Also, the CreateCubeWithFace() function is simply the massive switch statement that dictates what direction and where the new cube should be drawn. For clarity, we condensed it into its own function.

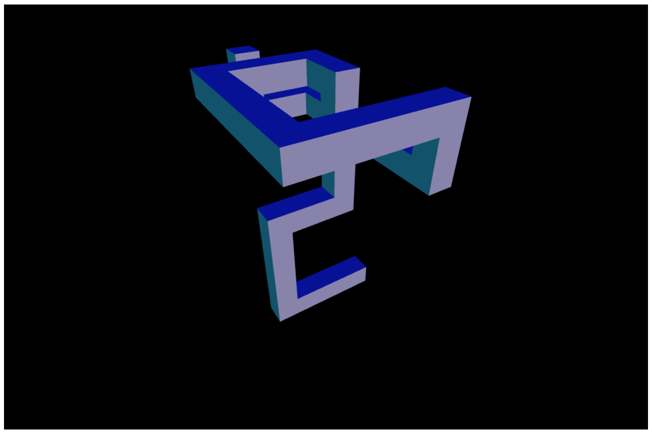
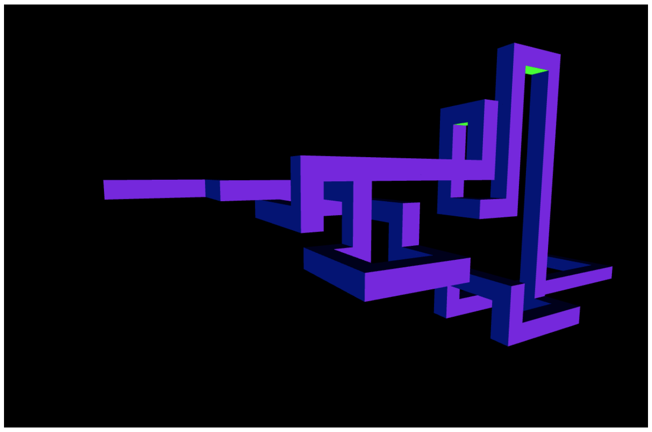
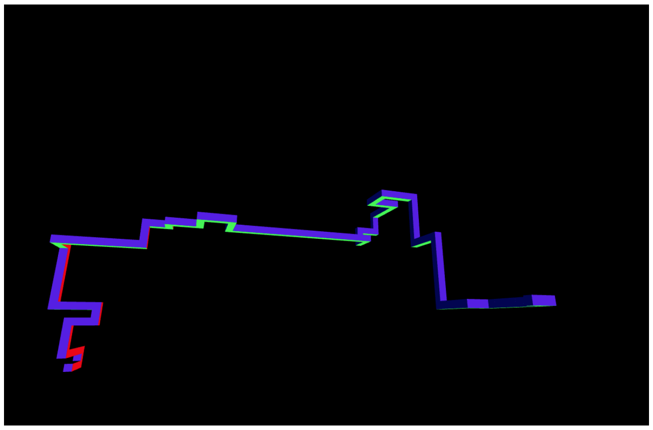
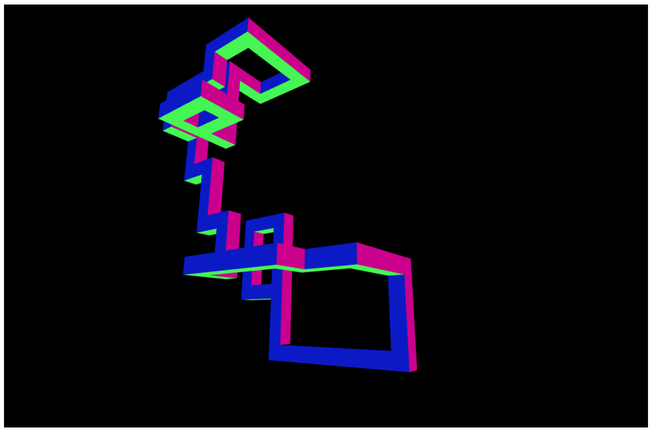


```
newCube(){
    this.time += deltaTime;
    if ((this.time >= 1000)){
        this.time = 0;
        currSteps++;
        let face = this.cameFrom;
        if (stepCount <= currSteps){
            let index = Math.floor(random(0, this.possible.length));
            face  = this.possible[index];
            currSteps = 0;
            stepCount = Math.floor(random(2, 6));
        }
        this.CreateCubeWithFace(face);
    }
}
```



```
CreateCubeWithFace(face){
    switch(face){
        //increase x
        case "right":
            console.log("right");
            cubeArray.push(new Cube(createVector(this.origin.x + 100, this.origin.y, this.origin.z), "right"));
            break;
        //decrease x
        case "left":
            console.log("left");
            cubeArray.push(new Cube(createVector(this.origin.x - 100, this.origin.y, this.origin.z), "left"));
            break;
        //increase y
        case "bottom":
            console.log("bottom");
            cubeArray.push(new Cube(createVector(this.origin.x, this.origin.y + 100, this.origin.z), "bottom"));
            break;
        //decrease y
        case "top":
            console.log("top");
            cubeArray.push(new Cube(createVector(this.origin.x, this.origin.y - 100, this.origin.z), "top"));
            break;
        //increase z
        case "front":
            console.log("front");
            cubeArray.push(new Cube(createVector(this.origin.x, this.origin.y, this.origin.z + 100), "front"));
            break;
        //decrease z
        case "back":
            console.log("back");
            cubeArray.push(new Cube(createVector(this.origin.x, this.origin.y, this.origin.z - 100), "back"));
            break;
    }
}
```

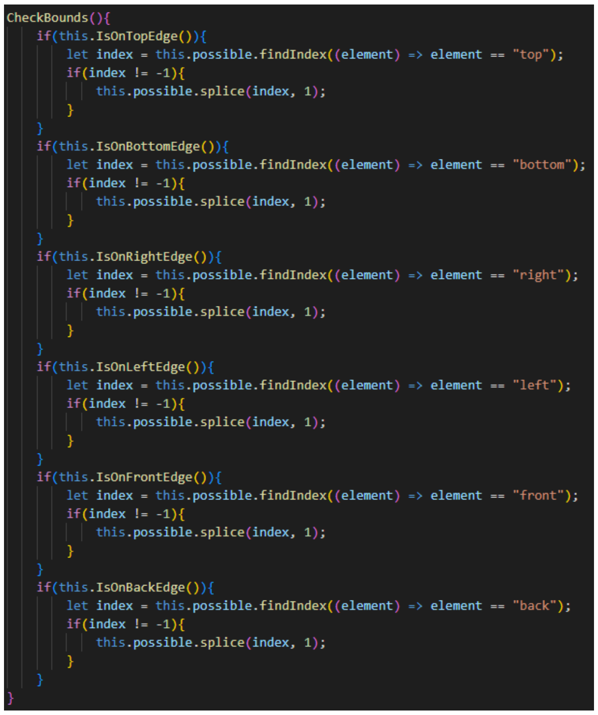
The end result is a 3D line (or "Snake") that keeps drawing itself until it runs into itself. However, it is possible for the Snake to never run into itself or go far off into space. In addition, the piece as a whole is sort of uninteresting to see and hear. The final step will see us adding interesting visuals, sounds, as well as creating the bounding box to constrain the Snake from going too far.

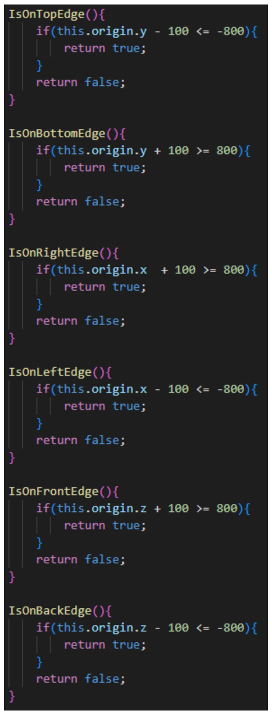
Step 3: Innovate

With what we wanted to make clean in our minds, we went ahead and began to create the final version. Nate got to work on the bounding box while Dominic found the colors and sounds we wanted for the Nokia phone aesthetic.

We attempted to create a 3D bounding box to contain the “Snake” but found this to be harder to implement than we expected. We found that attempting to contain the boxes by checking each of their x, y, and z components against some arbitrary values didn’t yield the results we wanted. We were unsure if boxes their dimensions were with respect to the top-left most corner or from the centroid of the box. Our logic seemed completely sound to us however undesired behaviors still arised and we end up setting up constraints that more or less define a cube in which the “Snake” can travel in.

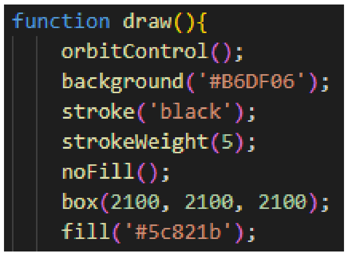


```
CheckBounds(){
    if(this.IsOnTopEdge()){
        let index = this.possible.findIndex((element) => element == "top");
        if(index != -1){
            this.possible.splice(index, 1);
        }
    }
    if(this.IsOnBottomEdge()){
        let index = this.possible.findIndex((element) => element == "bottom");
        if(index != -1){
            this.possible.splice(index, 1);
        }
    }
    if(this.IsOnRightEdge()){
        let index = this.possible.findIndex((element) => element == "right");
        if(index != -1){
            this.possible.splice(index, 1);
        }
    }
    if(this.IsOnLeftEdge()){
        let index = this.possible.findIndex((element) => element == "left");
        if(index != -1){
            this.possible.splice(index, 1);
        }
    }
    if(this.IsOnFrontEdge()){
        let index = this.possible.findIndex((element) => element == "front");
        if(index != -1){
            this.possible.splice(index, 1);
        }
    }
    if(this.IsOnBackEdge()){
        let index = this.possible.findIndex((element) => element == "back");
        if(index != -1){
            this.possible.splice(index, 1);
        }
    }
}
```

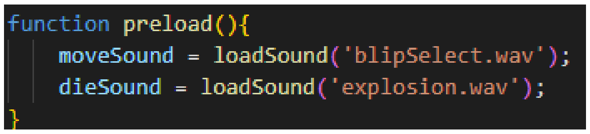


```
IsOnTopEdge(){
    if(this.origin.y - 100 <= -800){
        return true;
    }
    return false;
}

IsOnBottomEdge(){
    if(this.origin.y + 100 >= 800){
        return true;
    }
    return false;
}

IsOnRightEdge(){
    if(this.origin.x  + 100 >= 800){
        return true;
    }
    return false;
}

IsOnLeftEdge(){
    if(this.origin.x - 100 <= -800){
        return true;
    }
    return false;
}

IsOnFrontEdge(){
    if(this.origin.z + 100 >= 800){
        return true;
    }
    return false;
}

IsOnBackEdge(){
    if(this.origin.z - 100 <= -800){
        return true;
    }
    return false;
}
```

The visual aesthetics were created by color picking off of Nokia phone screens playing the Snake game. The "invisible" bounding box was created by placing a box with no fill, but a bold stroke, and then making the background of the canvas a solid color. At first, we were going to make the camera be inside of the bounding box at all times, but sometimes the Snake would clip out of the box and it just didn't look good. The audio was generated off of this useful website (https://sfxr.me/) and gave us a blip and death sound for the Snake, to fit in with the simple Nokia audio.

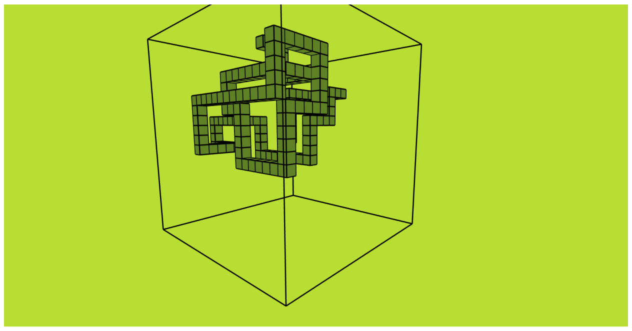
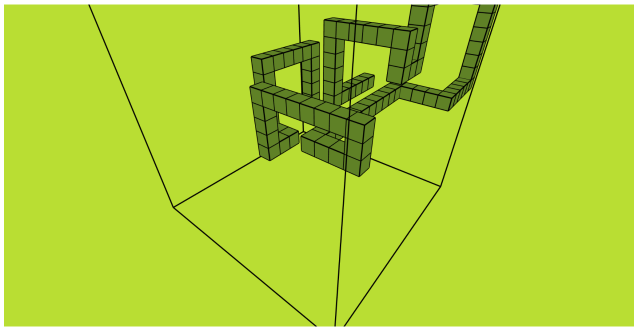
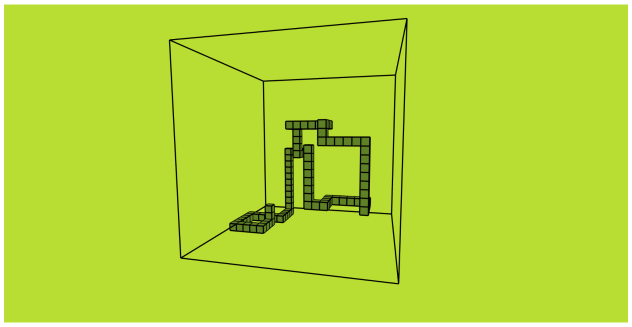
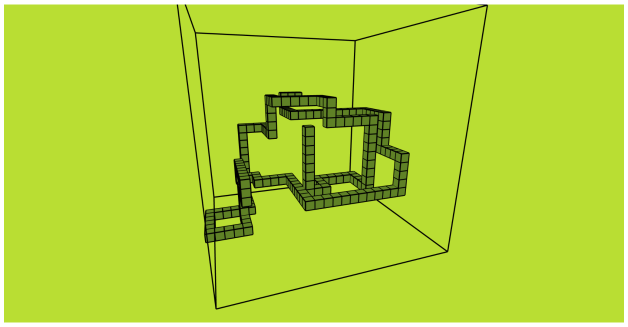


```
function draw(){
    orbitControl();
    background('#B6DF06');
    stroke('black');
    strokeWeight(5);
    noFill();
    box(2100, 2100, 2100);
    fill('#5c821b');
```



```
function preload(){
    moveSound = loadSound('blipSelect.wav');
    dieSound = loadSound('explosion.wav');
}
```

The end result was something we are quite happy with. Though it is not perfect– as sometimes the Snake goes outside of the box, sometimes the audio gets distorted, and it's also not a "true" Snake game where the Snake is seeking out food pellets. However, it came out great anyways and is actually quite fun to watch for a bit. You'll start rooting for the Snake to not be dumb and run into itself.

Reflect

Dominic:

I am happy with the result of this experiment. In comparison to last week, where I felt I did not contribute too much to the code, I feel this week that I had good input and it was truly the both of us working together to get it to work. I focused on the aesthetic code as well as the cube face generation and cube object programming. It's hard to say what one thing either of us did as we truly did help each other in all aspects. The high point for me is seeing how the end result looks pretty close to what I'd imagine a 3D Nokia Snake to look like. The low point is that our total working time on this project was quite a while.

Nathan:

This project was another great introduction to web-based 3D graphics and learning how WebGL works within P5 was definitely challenging but engaging. Attempting to convert code from P3D to P5 didn't go as smoothly as we wanted to but we achieved an end result that we were both satisfied with. This project featured a good mix of collaborative coding as we both tackled individual parts of the project while making sure to help each other when necessary. Creating logic that defined how cubes were to be spawned as well as where they could be spawned was one of the tougher parts for me as understanding how a new cube relates to an existing cube was troublesome. Overall this was another great collaborative project.

Self-Evaluation

Dominic

Self Evaluation Rubric						
Did you complete the assignment and did you complete it on time?	**Submitted on time**	**Up to 1 day late**	**Up to 2 days late**	**Up to 3 days late**	**4 days late or more**	**Do you need to clarify?** **No**
	X	☐	☐	☐	☐	
Did you collaborate with a partner?	**Worked with partner**			**Worked alone**		**Do you need to clarify?** **No**
	X			☐		
Did you put in earnest effort and provide an articulate summary of your experience?	**Excellent**	**Pretty good**	**About average**	**Could be improved**	**Not this time**	**What supports this?** **I tried to explain my process as thoroughly as possible. Used many pictures.**
	X	☐	☐	☐	☐	
Was the assignment complete, with minimal errors, correct output, and good style?	**Excellent**	**Pretty good**	**About average**	**Could be improved**	**Not this time**	**What supports this?** **Had two people look over each part of the assignment.**
	X		☐	☐	☐	
How much EXTRA effort did you put into the assignment?	**A lot of extra effort**		**Some extra effort**		**Not this time**	**What supports this?** **Just wanted it to be done after a while.**
	☐		**X**		☐	
Reflection in section above.						

Nathan

Self Evaluation Rubric

Did you complete the assignment and did you complete it on time?	**Submitted on time**	**Up to 1 day late**	**Up to 2 days late**	**Up to 3 days late**	**4 days late or more**	**Do you need to clarify?** **No**
	X	☐	☐	☐	☐	
Did you collaborate with a partner?	**Worked with partner**			**Worked alone**		**Do you need to clarify?** **No**
	X					
Did you put in earnest effort and provide an articulate summary of your experience?	**Excellent**	**Pretty good**	**About average**	**Could be improved**	**Not this time**	**What supports this?** **Our process is well documented with adequate pictures that show every step of the project.**
	X	☐	☐	☐	☐	
Was the assignment complete, with minimal errors, correct output, and good style?	**Excellent**	**Pretty good**	**About average**	**Could be improved**	**Not this time**	**What supports this?** **The writeup is sufficient enough for what the teaching team expects and our code is neatly organized.**
	X		☐	☐	☐	
How much EXTRA effort did you put into the assignment?	**A lot of extra effort**		**Some extra effort**		**Not this time**	**What supports this?** **Learning how web-based 3D engines work as well as WebGL inside of P5 was certainly challenging but overal a great experience.**
			X		☐	
Reflection in section above.						