**INTERIOR DESIGNS**

**Introduction:**  
Interior designs is a home décor and purchase company where we the customers can customize their home and purchase the suitable interiors on it too. Interior designs has a key feature of personalization which makes it better than its opponents and rivals. Whether you are looking for a full-scale renovation or simply want to update the look and feel of a room, we can provide you with the guidance and support you need to achieve your desired outcome. Our goal is to create spaces that are both visually stunning and practical, ensuring that every aspect of your design is tailored to your individual tastes and requirements.

From initial concept development to final installation, we are committed to providing you with an exceptional level of service and expertise. So, whether you are seeking inspiration for your next project or are ready to get started on a new design, we invite you to explore our website and discover how we can help you bring your vision to life.  
  
Interior design is built on Html, css and javascript.  
I used HTML to define the structure of my website, creating individual pages for each project or piece of work. I organised my content using headings, paragraphs, and lists to make it easy for visitors to navigate my site.

To style my page and give it a professional look, I used CSS. I chose fonts, colors, and backgrounds that complement each other and create a cohesive visual theme across all of my pages. I also made sure that my portfolio is responsive, meaning it looks great on any device.

To add interactivity to my site and provide a more engaging experience for visitors, I used JavaScript. I can create animations, pop-up galleries, and other interactive features that showcase my skills and creativity.

Overall, my website was made using HTML, CSS, and JavaScript is an excellent tool for showcasing my work to potential clients or employers. It would give visitors a clear and organized view of my skills and experience while also demonstrating my creativity and technical abilities.  
  
**THE CODES USED ARE:  
  
Main page:**<!DOCTYPE html>

<html>

<head>

<title> ECOM WEBSITE- </title>

<link rel="stylesheet" href="./style.css">

</head>

<body>

<div class="banner">

<div class="navbar">

<ul class="nav">

<img src="./logo1.png.png" id="logo" >

<li><a href="#">Home</a></li>

<li><a href="https://www.youtube.com/watch?v=dQw4w9WgXcQ" target="\_blank">Bedroom</a></li>

<li><a href= "#">Dinning</a></li>

<li><a href="#">Kitchen</a></li>

<li><a href="#">Backyard</a></li>

</ul>

</div>

<div class="content">

<h1> DESIGN YOUR HOUSE</h1>

<P>Design your home as per your taste </P>

<div>

<button onclick="location.href='customize.html'" type="button">customize</button>

<button type="button">Purchase</button>

<button type="button">Renovate</button>

<button type="button">Filter</button>

</div>

<br>

<div class="para">

Whether you are looking for a full-scale renovation or simply want to update the look and feel of a room, we can provide you with the guidance and support you need to achieve your desired outcome. Our goal is to create spaces that are both visually stunning and practical, ensuring that every aspect of your design is tailored to your individual tastes and requirements.

</div>

</div>

</div>

</body>

</html>  
 **Style sheet:**  
\*{

margin: 0;

padding: 0;

font-family: sans-serif;

}

.banner{

width: 100%;

height: 100vh;

background-image:url(./background.jpg);

background-size: 100%;

background-position: center;

}

ul

{

list-style-type: none;

margin: 0;

padding: 0;

overflow: hidden;

}

li {

float: right;

}

li a {

color: white;

text-align: center;

padding: 14px 16px;

text-decoration: none;

}

/\* Change the link color to #111 (black) on hover \*/

li a:hover {

display: inline-block;

margin: 0 20px;

background-color: rgb(36, 110, 66) ;

position: relative;

}

img {

width: 10%;

height:10%;

border-radius: 50%;

}

.navbar ul li:hover::after{

width:100%;

transition: 0.5s;

}

.content{

width: 100%;

position: absolute;

top: 50%;

transform: translateY(-50%);

text-align: center;

color: #fff;

}

.content hl{

font-size: 70px;

margin-top: 80px;

}

.content p{

margin: 20px auto;

font-weight: 100;

line-height: 25p

}

button{

width: 200px;

padding: 15px 0;

text-align: center;

margin: 20p 10px;

border-radius: 25px;

font-weight: bold;

border: 2px solid #009688;

background: transparent;

color: #fff;

cursor: pointer;

position: relative;

overflow: hidden;

}

button:hover {

margin: 0 20px;

background-color: rgb(36, 110, 66) ;

position: relative;

border-radius: 25px;

left: 0;

bottom: 0;

border: none;

z-index: -1;

}

.tate  
  
  
  
  
  
  
  
  
  
**customize page:**<!DOCTYPE html>

<html>

<head>

<style>

@import url('https://fonts.googleapis.com/css?family=Lora&display=swap');

body {

display:grid;

justify-content:center;

align-content:center;

height:100vh;

font:5vh Lora;

color : white;

}

a {

color:#222a;

text-decoration:#2224 wavy underline;

transition:all 0.3s;

&:hover {

color:#2223;

text-shadow:0 0 100px #333;

}

}

canvas {

position:fixed;

z-index:-1;

</style>

<title>customization page</title>

</head>

<body>

<canvas id='gridwormCanvas' width='1350' height='620' style='background-color: white;' ></canvas>

<P><h1 class="wp-block-post-title alignwide has-text-align-center">Home Design Software - Design Your House Online</h1></P>

<p class="has-text-align-center">Easy-to-use home design software that you can use to plan and design rooms in your home or even the entire house. Create floor plans, furnish and decorate, then visualize in 2D &amp; 3D.</p>

<figure class="wp-block-embed is-type-video is-provider-youtube wp-block-embed-youtube wp-embed-aspect-16-9 wp-has-aspect-ratio"><div class="wp-block-embed\_\_wrapper">

<iframe loading="lazy" title="Home Design with RoomSketcher" width="500" height="281" src="https://www.youtube.com/embed/OEQ0Avprjwk?feature=oembed" frameborder="0" allow="accelerometer; autoplay; clipboard-write; encrypted-media; gyroscope; picture-in-picture; web-share" allowfullscreen=""></iframe>

</div></figure>

<em>RoomSketcher is an easy-to-use tool to create fast visualizable floor plans for remodeling or a new build from scratch</em>

<div class="tate">

<img id ="tate "src="https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcQqk-OR2EuqLg3r8-zlQwhazgv9YdGjcTWAwA&usqp=CAU" />

</div>

<script>

console.clear();

const canvas = document.createElement('canvas');

document.body.append(canvas);

canvas.style.display = 'block';

canvas.style.width = '100vw';

canvas.style.height = '100vh';

canvas.width = canvas.offsetWidth;

canvas.height = canvas.offsetHeight;

const gl = canvas.getContext('webgl2');

if (!gl) {

alert('require webgl 2.0, bye')

}

const vss = `#version 300 es

in vec2 p;

void main() {

gl\_Position = vec4(p, 0.0, 1.0);

}

`;

const fss = `#version 300 es

precision mediump float;

out vec4 o;

uniform vec4 c;

void main() {

o = c;

}

`;

const vs = gl.createShader(gl.VERTEX\_SHADER);

gl.shaderSource(vs, vss);

gl.compileShader(vs);

if (!gl.getShaderParameter(vs, gl.COMPILE\_STATUS)) {

console.error(gl.getShaderInfoLog(vs));

throw 1;

}

const fs = gl.createShader(gl.FRAGMENT\_SHADER);

gl.shaderSource(fs, fss);

gl.compileShader(fs);

if (!gl.getShaderParameter(fs, gl.COMPILE\_STATUS)) {

console.error(gl.getShaderInfoLog(fs));

throw 2;

}

const prg = gl.createProgram();

gl.attachShader(prg, vs);

gl.attachShader(prg, fs);

gl.linkProgram(prg);

if (!gl.getProgramParameter(prg, gl.LINK\_STATUS)) {

console.error(gl.getProgramInfoLog(prg));

throw 3;

}

gl.detachShader(prg, vs);

gl.deleteShader(vs);

gl.detachShader(prg, fs);

gl.deleteShader(fs);

// ---- End of antipattern ----

const $p = gl.getAttribLocation(prg, 'p');

const $c = gl.getUniformLocation(prg, 'c');

const va = gl.createVertexArray();

gl.bindVertexArray(va);

const N = 300; // n triangles

let ps;

{

ps = new Float32Array(2 + N \* 2 \* 2);

ps[0] = 0; // clip space center

ps[1] = 0;

let j = 2;

for (let i = 0; i < N; ++i) {

ps[j++] = Math.random() \* 2 - 1; //x

ps[j++] = Math.random() \* 2 - 1; //y

ps[j++] = Math.random() \* 2 - 1; //x

ps[j++] = Math.random() \* 2 - 1; //y

}

}

const buf = gl.createBuffer();

gl.bindBuffer(gl.ARRAY\_BUFFER, buf);

gl.bufferData(gl.ARRAY\_BUFFER, ps, gl.DYNAMIC\_DRAW);

gl.enableVertexAttribArray($p);

gl.vertexAttribPointer(

$p,

2, gl.FLOAT, // two 32b-float (8bytes)

false,

0, // skip n byte to fetch next

0 // skip n byte to fetch first

);

let idxs;

{

idxs = new Uint16Array(3 \* N);

let j = 0;

for (let i = 0; i < N; ++i) {

idxs[j++] = 0;

idxs[j++] = 1 + i \* 2;

idxs[j++] = 2 + i \* 2;

}

}

const ibuf = gl.createBuffer();

gl.bindBuffer(gl.ELEMENT\_ARRAY\_BUFFER, ibuf);

gl.bufferData(gl.ELEMENT\_ARRAY\_BUFFER, idxs, gl.STATIC\_DRAW);

gl.bindVertexArray(null);

//----- render

gl.viewport(0, 0, gl.canvas.width, gl.canvas.height);

gl.clearColor(0.1, 0.1, 0.1, 1);

gl.blendFunc(gl.SRC\_ALPHA, gl.ONE\_MINUS\_SRC\_ALPHA);

gl.enable(gl.BLEND);

gl.disable(gl.CULL\_FACE);

gl.useProgram(prg);

gl.bindVertexArray(va);

function f() {

gl.clear(gl.COLOR\_BUFFER\_BIT);

gl.uniform4fv($c, [0.2, 0.2, 0.2, 0.02]);

gl.drawElements(

gl.TRIANGLES,

idxs.length, // n indices

gl.UNSIGNED\_SHORT, // ui16

0 // skip n bytes to fetch first

);

}

f();

// ---

gl.bindBuffer(gl.ARRAY\_BUFFER, buf);

document.body.onmousemove = (e) => {

ps[0] = e.clientX / window.innerWidth \* 2 - 1;

ps[1] = -1 \* (e.clientY / window.innerHeight \* 2 - 1);

gl.bufferSubData(gl.ARRAY\_BUFFER, 0, ps.slice(0, 2)); // that's why DYNAMIC\_DRAW

f();

}

</script>

</body>

</html>  
  
  
  
  
  
  
  
**Javascript for BG:**  
{

constructor(point,interval,pointsList,screenWidth,screenHeight)

{

this.radius = 2;

this.xCoord = point.x;

this.yCoord = point.y;

this.interval= interval;

this.color = this.getColor(1,true);//get random color object

this.mainColor = this.color.color;//color of the head and body of the girdworm

this.mainColorIndex = this.color.index;

this.nColor = this.getColor(1,true);//get another random color object

this.arrowHeadColor = this.nColor.color;//color of the arrrow points at the head of the gridworm

this.arrowHeadColorIndex = this.nColor.index;

this.pointsList = pointsList;

this.screenWidth = screenWidth;

this.screenHeight= screenHeight;

this.speed = 5;//the magnitude of the velocity

this.velocity= this.getVelocity();

this.junctionMemory = [{point:point,velocity:this.velocity}];//memory of each junction visited(helps to construct the worm)

//the maximum number of junctions a gridworm can keep in memory(this determines how long the gridworm will be)

this.junctionMemoryLength = 6;

}

getColor(opacity,isRandom = true,index = 0)

{

if(opacity < 0 || opacity > 1 || opacity === null || isNaN(opacity))//if opacity is incorrect

{

opacity = 1;

}

var colors =

[

`rgba(0,0,0,${opacity})`,`rgba(192,192,192,${opacity})`/\*silver\*/,`rgba(128,128,128,${opacity})`/\*gray\*/,`rgba(128,0,0,${opacity})`/\*maroon\*/,

`rgba(255,0,0,${opacity})`/\*red\*/,`rgba(0,255,0,${opacity})`/\*lime\*/,`rgba(0,0,255,${opacity})`/\*blue\*/,`rgba(255,0,255,${opacity})`/\*fuchsia\*/,

`rgba(128,128,0,${opacity})`/\*olive\*/,`rgba(0,128,0,${opacity})`/\*green\*/,`rgba(128,0,128,${opacity})`/\*purple\*/,

`rgba(0,128,128,${opacity})`/\*teal\*/,`rgba(0,0,128,${opacity})`/\*navy\*/,`rgba(138,57,0,${opacity})`/\*brown\*/, `rgba(205,133,63,${opacity})`,

`rgba(244,164,96,${opacity})`,`rgba(139,105,30,${opacity})`,`rgba(165,42,42,${opacity})`,`rgba(178,34,34,${opacity})`,

`rgba(220,20,60,${opacity})`,`rgba(255,140,0,${opacity})`,`rgba(255,165,0,${opacity})`,`rgba(255,215,0,${opacity})`,`rgba(184,134,11,${opacity})`,

`rgba(218,165,32,${opacity})`,`rgba(218,165,32,${opacity})`,`rgba(238,232,170,${opacity})`,`rgba(189,183,107,${opacity})`,`rgba(240,230,140,${opacity})`,

`rgba(0,100,0,${opacity})`, `rgba(34,139,34,${opacity})`,`rgba(32,178,170,${opacity})`,`rgba(47,79,79,${opacity})`,

`rgba(0,139,139,${opacity})`,`rgba(95,158,160,${opacity})`,`rgba(70,130,180,${opacity})`,`rgba(25,25,112,${opacity})`,

`rgba(0,0,128,${opacity})`,`rgba(0,0,139,${opacity})`,`rgba(72,61,139,${opacity})`,`rgba(75,0,130,${opacity})`,`rgba(139,0,139,${opacity})`,

`rgba(0,0,0,${opacity})`,`rgba(105,105,105,${opacity})`, `rgba(169,169,169,${opacity})`

];

if(isRandom)

{

let index = Math.floor(this.getRandomNumber(0,colors.length-1));

let color = colors[index];

return {color:color,index:index};

}

else//if specific

{

if(index >=0 && index < colors.length)

{

return colors[index];

}

return colors[0];

}

}

getVelocity()

{

let x,y;

//flip a coin to decide if gridworm moves vertically or horizontally

if( Math.random() > 0.5)//if gridworm moves vertically

{

x = 0;//no horizontal movement

y = Math.random() > 0.5? -this.speed: this.speed;//flip a coin to decide if gridworm moves upwards or downwards

}

else//if gridworm moves horizontally

{

x = Math.random() > 0.5? -this.speed: this.speed;//flip a coin to decide if gridworm moves left or right

y = 0;//no vertical movement

}

return {x:x, y:y};

}

/\*\*

\* Returns a random number between min (inclusive) and max (exclusive)

\* @param {number} min The lesser of the two numbers.

\* @param {number} max The greater of the two numbers.

\* @return {number} A random number between min (inclusive) and max (exclusive)

\*/

getRandomNumber(min, max)

{

return Math.random() \* (max - min) + min;

}

drawCircle(x,y,circleradius,ctx,colorIndex)

{

for(let i = 0; i < 3; i++)

{

let color = '';

let radius = 0;

switch(i)//create three circles with same center

{

case 0:

radius =circleradius;//smallest circle

color = this.getColor(1,false,colorIndex);

break;

case 1:

radius =circleradius \* 2;//bigger circle

color = this.getColor(0.5,false,colorIndex);

break;

case 2:

radius =circleradius \* 6;//biggest circle

color = this.getColor(0.2,false,colorIndex);

break;

}

//draw the node

ctx.beginPath();

ctx.arc(x,y,radius,0,2\*Math.PI);

ctx.fillStyle = color;

ctx.fill();

ctx.strokeStyle = color;

ctx.stroke();

}

}

drawArrowHead(x,y,circleradius,ctx,colorIndex)

{

let points = [];

if(this.velocity.x === 0)//if gridworm is moving vertically

{

if(this.velocity.y > 0)//if gridworm is moving down

{

points.push({x:x+this.interval/3,y:y});//point to the right

points.push({x:x-this.interval/3,y:y});//point to the left

points.push({x:x,y:y+this.interval/3});//point below

}

else//if gridworm is moving up

{

points.push({x:x+this.interval/3,y:y});//point to the right

points.push({x:x-this.interval/3,y:y});//point to the left

points.push({x:x,y:y-this.interval/3});//point above

}

}

else//if gridworm is moving horizontally

{

if(this.velocity.x > 0)//if gridworm is moving right

{

points.push({x:x+this.interval/3,y:y});//point to the right

points.push({x:x,y:y-this.interval/3});//point above

points.push({x:x,y:y+this.interval/3});//point below

}

else//if gridworm is moving left

{

points.push({x:x-this.interval/3,y:y});//point to the left

points.push({x:x,y:y-this.interval/3});//point above

points.push({x:x,y:y+this.interval/3});//point below

}

}

//draw a circle about the points that make the arrow head

for(let i = 0; i < points.length;i++)

{

let point = points[i];

this.drawCircle(point.x,point.y,circleradius/2,ctx,colorIndex);

}

this.drawTriangle(points[0],points[1],points[2],ctx);//draw the arrow head

}

drawTriangle(point1,point2,point3,ctx)

{

ctx.beginPath();

ctx.moveTo(point1.x, point1.y);

ctx.lineTo(point2.x, point2.y);

ctx.lineTo(point3.x, point3.y);

ctx.fillStyle = 'rgba(0,0,0,0.1)';//transparent black

ctx.fill();

}

draw(ctx)

{

//draw the head of the gridworm

this.drawCircle(this.xCoord,this.yCoord,this.radius/2,ctx,this.mainColorIndex);

this.drawArrowHead(this.xCoord,this.yCoord,this.radius/2,ctx,this.arrowHeadColorIndex);

//draw circles and squares at every visited junctions in the gridworm's memory(not RAM)

for(let i = 0; i < this.junctionMemory.length; i++)

{

let junction = this.junctionMemory[this.junctionMemory.length -(i+1)];

//draw a circle at each junction point

this.drawCircle(junction.point.x, junction.point.y,this.radius/2,ctx,this.mainColorIndex);

//draw painted squares at every junction point

ctx.fillStyle = this.getColor(0.1,false,this.mainColorIndex);

ctx.fillRect(junction.point.x,junction.point.y,this.interval,this.interval);

}

//draw the line connecting head to body

ctx.strokeStyle = 'black';

ctx.lineWidth = this.radius;

ctx.beginPath();

ctx.moveTo(this.xCoord,this.yCoord);

//draw a line to link all the visited junctions in the gridworm's memory(not RAM)

for(let i = 0; i < this.junctionMemory.length; i++)

{ //starting from the most recent to the least recent(LIFO)[NB: more like a stack data structure]

let junction = this.junctionMemory[this.junctionMemory.length -(i+1)];

ctx.lineTo(junction.point.x, junction.point.y);

}

ctx.stroke();

ctx.closePath();

}

update(deltaTime)

{

this.junctionMemoryLength = this.junctionMemoryLength < 1? 1: this.junctionMemoryLength;

//keep the gridworm moving in its current direction

this.xCoord += this.velocity.x;//if gridworm is going left or right, keep it going

this.yCoord += this.velocity.y;//if gridworm is going up or down, keep it going

if(this.xCoord <= this.interval)//if gridworm reaches the leftmost point

{

this.xCoord = this.interval;

this.velocity.x = -this.velocity.x;//move right

this.xCoord += this.velocity.x \* 3;//nudge it a bit away from the edge

}

if(this.xCoord >= this.screenWidth - this.interval)//if gridworm reaches the rightmost point

{

this.xCoord = this.junctionMemory[this.junctionMemory.length-1].point.x;

this.velocity.x = -this.velocity.x;//move left

this.xCoord += this.velocity.x \* 3;//nudge it a bit away from the edge

}

if(this.yCoord <= this.interval)//if gridworm reaches the topmost most point

{

this.yCoord = this.interval;

this.velocity.y = -this.velocity.y; //move down

this.yCoord += this.velocity.y \* 3;//nudge it a bit away from the edge

}

if(this.yCoord >= this.screenHeight - this.interval)//if gridworm reaches the lowest point)

{

this.yCoord = this.junctionMemory[this.junctionMemory.length-1].point.y;

this.velocity.y = -this.velocity.y;//move up

this.yCoord += this.velocity.y \* 4;//nudge it a bit away from the edge

}

let currentCoord = {x:this.xCoord,y:this.yCoord};

let latestJunction = this.getJunctionReached(currentCoord);

if(latestJunction !== currentCoord)

{

let originalVelocity = this.velocity;

let newVelocity = this.getVelocity();//flip a coin to decide to move up and down or right and left

if(originalVelocity.y === 0 )//if gridworm is moving horizontally

{

this.velocity = newVelocity;

if(newVelocity.y === 0 && newVelocity.x === -originalVelocity.x )//if it continues the horizontal movement in the opposite direction

{

//don't add the new junction to the memory queue

}

else

{

let memory = {point:latestJunction,velocity:this.velocity};

if(!this.isInMemory(memory))

{

this.junctionMemory.push(memory);//add new memory to the queue

}

//this.junctionMemory.push({point:latestJunction,velocity:this.velocity});//add new memory to the queue

}

//nudge it a bit away from the junction

this.xCoord += this.velocity.x \* 3; //not complete yet. Don't make it too much or too little.

}

else //if gridworm is moving vertically

{

this.velocity = newVelocity;

if(newVelocity.x === 0 && newVelocity.y === -originalVelocity.y )//if it continues the verticalal movement in the opposite direction

{

//don't add the new junction to the memory queue

}

else

{

let memory = {point:latestJunction,velocity:this.velocity};

if(!this.isInMemory(memory))

{

this.junctionMemory.push(memory);//add new memory to the queue

}

}

//nudge it a bit away from the junction

this.yCoord += this.velocity.y \* 3; //not complete yet. Don't make it too much or too little.

}

}

if(this.junctionMemory.length > this.junctionMemoryLength)//if memory is too long

{

this.junctionMemory.shift();//remove the first memory

}

}

isInMemory(memory)//check if a junction is in memory

{

this.junctionMemory.some(function(mem)

{

if(mem.point === memory.point)

{

return true;//junction is in memory

}

return mem.point === memory.point;

});

return false;//junction is NOT in memory

}

getJunctionReached(currentCoord)

{

for(let i = 0; i < this.pointsList.length; i++)

{

let point = this.pointsList[i];

//if point(junction) is too far away, ignore it

if(Math.abs(currentCoord.x - point.x) > (2 \* this.interval) || Math.abs(currentCoord.y - point.y) > (2 \*this.interval) )

{

continue;

}

let distance = this.getDistance(currentCoord,point);

if(distance <= (this.radius))//if gridworm head is close enough to a junction

{

return point;

}

}

return currentCoord;

}

getDistance(p1,p2)//the distance between two points, p1 and p2

{

let dx = p1.x - p2.x;

let dy = p1.y - p2.y;

let distance = Math.sqrt(dx\*dx + dy\*dy);

return distance;

}

/\*\*

\* Let node correspond to window resizing.

\* @param {number} screenHeight The height of the screen.

\* @param {number} screenWidth The width of the screen.

\* @param {number} dy The percentage change in browser window height

\* @param {number} dx The percentage change in browser window width .

\*/

refreshScreenSize(screenHeight,screenWidth,dx,dy,points)

{

}

}

//sets up and controls all points and gridworms on the canvas

class Painter

{

constructor(screenWidth,screenHeight)

{

this.screenWidth = screenWidth;

this.screenHeight = screenHeight;

this.interval = 40;//interval from one point to the next

this.points = this.createPoints(); //coordinates of the vertices of all squares when the canvas is partitioned

this.gridWorms = this.createGridWorms();

this.color = this.getRandomColor(0.1);

document.addEventListener('click',(event)=>//when user clicks on the canvas

{

this.points = this.createPoints();

this.gridWorms = this.createGridWorms();//spawn new gridworms

this.color = this.getRandomColor(0.1);

});

}

createGridWorms()

{

let gridworms = [],

numOfGridWorms = 30;

for(var i = 0; i < numOfGridWorms; i++)

{

let point = this.points[Math.floor(this.getRandomNumber(0,this.points.length-1))];//randomly select a point

gridworms.push(new GridWorm(point,this.interval,this.points,this.screenWidth,this.screenHeight));

}

return gridworms;

}

createPoints()//divide the canvas into squares

{

let points = [],

interval = this.interval;//interval from one point to the next

for(var y = interval; y < this.screenHeight; y+=interval)//get all points in the grid, starting from the top to the bottom

{

if(y+interval > this.screenHeight)//if the next point is beyond the right edge of the canvas

{

continue; //skip

}

for(var x = interval; x < this.screenWidth; x+=interval)//all the while, getting all the horizontal points at each level

{

if(x+interval > this.screenWidth)//if the next point is beyond the bottom edge of the canvas

{

continue; //skip

}

points.push({x:x,y:y});

}

}

return points;

}

getRandomColor(opacity)

{

var colors = [

`rgba(255,0,0, ${opacity})`,//red

`rgba(255, 242,0, ${opacity})`,//yellow,

`rgba(0,0,255, ${opacity})`,//blue

`rgba(255,255,0, ${opacity})`,//yellow

`rgba(0,255,255, ${opacity})`,//cyan

`rgba(255,0,255, ${opacity})`,//magenta/fuchsia

`rgba(192,192,192, ${opacity})`,//silver

`rgba(128,128,128, ${opacity})`,//gray

`rgba(128,0,0, ${opacity})`,//maroon

`rgba(128,128,0, ${opacity})`,//olive

`rgba(0,128,0, ${opacity})`,//green

`rgba(128,0,128, ${opacity})`,//purple

`rgba(0,128,128, ${opacity})`,//teal

`rgba(0,0,128, ${opacity})`,//navy

`rgba(0, 255, 0, ${opacity})`,//green

`rgba(77, 0, 255, ${opacity})`,//blue

`rgba(255, 0, 140, ${opacity})`,//purple

`rgba(0,255,0, ${opacity})`//lime

];

return colors[parseInt(this.getRandomNumber(0, colors.length))];

}

/\*\*

\* Returns a random number between min (inclusive) and max (exclusive)

\* @param {number} min The lesser of the two numbers.

\* @param {number} max The greater of the two numbers.

\* @return {number} A random number between min (inclusive) and max (exclusive)

\*/

getRandomNumber(min, max)

{

return Math.random() \* (max - min) + min;

}

/\*\*

\* Let canvas respond to window resizing.

\* @param {number} screenHeight The height of the screen.

\* @param {number} screenWidth The width of the screen.

\*/

refreshScreenSize(screenHeight,screenWidth)

{

if(this.screenHeight !== screenHeight || this.screenWidth !== screenWidth)//if the screen size has changed

{

this.screenHeight = screenHeight;

this.screenWidth = screenWidth;

this.points = this.createPoints(); //coordinates of the vertices of all squares when the canvas is partitioned

this.gridWorms = this.createGridWorms();

}

}

update(deltaTime)

{

this.gridWorms.forEach(function(gridworm)

{

gridworm.update(deltaTime);

});

}

draw(ctx)

{

/\*

for(var i = 0; i < this.points.length; i++)

{

let point = this.points[i];

ctx.fillStyle = Math.random() > 0.5? this.color:'white';//creates a disco effect

ctx.fillRect(point.x,point.y,this.interval,this.interval);

}

\*/

this.gridWorms.forEach(function(gridworm)

{

gridworm.draw(ctx);

});

}

}

//set everything up

function getBrowserWindowSize()

{

let win = window,

doc = document,

offset = 20,//

docElem = doc.documentElement,

body = doc.getElementsByTagName('body')[0],

browserWindowWidth = win.innerWidth || docElem.clientWidth || body.clientWidth,

browserWindowHeight = win.innerHeight|| docElem.clientHeight|| body.clientHeight;

return {x:browserWindowWidth-offset,y:browserWindowHeight-offset};

}

let browserWindowSize = getBrowserWindowSize(),

c = document.getElementById("gridwormCanvas"),

ctx = c.getContext("2d");

//set size of canvas

c.width = browserWindowSize.x;

c.height = browserWindowSize.y;

let SCREEN\_WIDTH = browserWindowSize.x,

SCREEN\_HEIGHT= browserWindowSize.y,

painter = new Painter(SCREEN\_WIDTH,SCREEN\_HEIGHT),

lastTime = 100,

windowSize;

function onWindowResize()//called every time the window gets resized.

{

windowSize = getBrowserWindowSize();

c.width = windowSize.x;

c.height = windowSize.y;

SCREEN\_WIDTH = windowSize.x;

SCREEN\_HEIGHT = windowSize.y;

}

window.addEventListener('resize',onWindowResize);

function updateCanvas()

{

ctx.clearRect(0,0,SCREEN\_WIDTH,SCREEN\_HEIGHT);

ctx.fillStyle = 'white';

ctx.fillRect(0,0,SCREEN\_WIDTH,SCREEN\_HEIGHT);

}

function doAnimationLoop(timestamp)

{

updateCanvas();

painter.refreshScreenSize(SCREEN\_HEIGHT,SCREEN\_WIDTH);//let canvas respond to window resizing

let deltaTime = timestamp - lastTime;

lastTime = timestamp;

painter.update(deltaTime);

painter.draw(ctx);

requestAnimationFrame(doAnimationLoop);

}

requestAnimationFrame(doAnimationLoop);  
  
  
**Final output:**  


