ALGORITHMS AND PROBLEM SOLVING PROJECT REPORT

Mitaali Nagpal
17103073
B-14

Submitted To:
Mr. Manish Kumar Thakur
Ms. Indu Chawla

MAZE GENERATOR
USING BACKTRACKING

This project deals with a simple maze generator, coded in Javascript. The project visually shows the algorithm of Bactracking at the user's pace and variables.

The project uses simple html and cascading style sheet to get to visualize the working of the maze generator and therefore, backtracking with a closer view.

The page shown to the user consists of the following data:

* Blocks: The number of rows and columns in the maze.
* Canvas Size: The width and height of the canvas element (in pixels).
* Sleep: The amount of time to wait after each frame. Useful for watching the algorithm work. Lower is faster.
* Steps per frame: The number of blocks to fill in each frame. Higher is faster.

The code for the same is given as on the next page:

```
var Block = function(maze, x, y) {
  this.x = x;
  this.y = y;
  this.coords = {x: this.x, y: this.y};
  this.width = maze.blockWidth;
  this.height = maze.blockHeight;
  this.maze = maze;
  this.markerSize = [this.width,
this.height].sort(function(a,b) {return
a-b;})[0];
  this.occupied = false;
  this.ctx = maze.ctx;
  this.parent = undefined;
  this.children = [];
};

Block.prototype.hasChild =
function(other) {
  return this.children.lastIndexOf(other)
> -1;
};

Block.prototype.randomAvailableNeighbor =
function() {
```

```
  var neighbors =
this.availableNeighbors();
  return
neighbors[Math.floor(Math.random() *
neighbors.length)];
};

Block.prototype.neighbors = function() {
  if (this._neighbors) {
    return this._neighbors;
  }

  this._neighbors = [];
  if (this.x > 0) {

this._neighbors.push(this.neighbor(-1,
0)); //left direction relatively
  }

  if (this.x < this.maze.hBlocks - 1) {
    this._neighbors.push(this.neighbor(1,
0)); //right direction relatively
  }
```

```
  if (this.y > 0) {
    this._neighbors.push(this.neighbor(0,
-1)); //downwards relatively
  }

  if (this.y < this.maze.vBlocks - 1) {
    this._neighbors.push(this.neighbor(0,
1)); //above relatively
  }

  return this._neighbors;
};

Block.prototype.availableNeighbors =
function() {
  var neighbors = this.neighbors();
  return neighbors.filter(function(n)
{return !n.occupied;});
};

Block.prototype.neighbor = function(relX,
relY) {
  var x = this.x + relX;
  var y = this.y + relY;
```

```
  if (x >= 0 && x < this.maze.hBlocks &&
y >= 0 && y < this.maze.vBlocks) {
    return this.maze.blocks[y][x];
  }
};

Block.prototype.connectTo =
function(other) {
  this.children.push(other);
  other.parent = this;
};

Block.prototype.connectedTo =
function(other) {
  if (other) {
    return this.parent === other ||
this.hasChild(other);
  } else {
    return false;
  }
};

Block.prototype.draw = function() {
  this.erase();
```

```
  if (!this.connectedTo(this.neighbor(0,
-1))) {
    this.drawTopWall();
  }

  if (!this.connectedTo(this.neighbor(0,
1))) {
    this.drawBottomWall();
  }

  if (this.occupied &&
!this.connectedTo(this.neighbor(-1, 0)))
{
    this.drawLeftWall();
  }

  if (!this.connectedTo(this.neighbor(1,
0))) {
    this.drawRightWall();
  }

  if (this.inHistory) {
    this.drawMarker();
```

```
  }
};

Block.prototype.erase = function() {
  this.ctx.fillStyle = "#FFFFFF"; //white
fillstyle
  this.ctx.fillRect(this.x * this.width,
this.y * this.height, this.width,
this.height);
};

Block.prototype.drawWall = function(x1,
y1, x2, y2) {
  var ctx = this.ctx;
  ctx.strokeStyle = "#000000"; //black
strokestyle
  ctx.beginPath();
  ctx.moveTo(x1, y1);
  ctx.lineTo(x2, y2);
  ctx.closePath();
  ctx.stroke();
};

Block.prototype.drawTopWall = function()
```

```
{
  this.drawWall(this.x * this.width,
                this.y * this.height,
                (this.x + 1) *
this.width,
                this.y * this.height);
};

Block.prototype.drawBottomWall =
function() {
  this.drawWall(this.x * this.width,
                (this.y + 1) *
this.height,
                (this.x + 1) *
this.width,
                (this.y + 1) *
this.height);
};

Block.prototype.drawLeftWall = function()
{
  this.drawWall(this.x * this.width,
                this.y * this.height,
                this.x * this.width,
```

```
                (this.y + 1) *
this.height);
};

Block.prototype.drawRightWall =
function() {
  this.drawWall((this.x + 1) *
this.width,
                this.y * this.height,
                (this.x + 1) *
this.width,
                (this.y + 1) *
this.height);
};

Block.prototype.drawMarker = function() {
  this.ctx.fillStyle =
this.maze.currentBlock === this ?
"#FF9999" : "#9999FF";

  this.ctx.beginPath();
  this.ctx.arc(this.x * this.width +
this.width / 2,
               this.y * this.height +
```

```
this.height / 2,
                this.markerSize * 0.33,
                0,
                Math.PI * 2, false);
  this.ctx.fill();
};

var MazeGenerator = function(width,
height, hBlocks, vBlocks, interval,
steps, id) {
  this.canvas =
document.getElementById(id);
  this.canvas.width = width;
  this.canvas.height = height;
  this.ctx =
this.canvas.getContext('2d');//returns a
drawing context on the canvas
  this.ctx.fillStyle = "#CCCCCC";
  this.ctx.fillRect(0, 0, width, height);
  this.width = width;
  this.height = height;
  this.hBlocks = hBlocks;
  this.vBlocks = vBlocks;
  this.blockWidth = this.width /
```

```
this.hBlocks;
  this.blockHeight = this.height /
this.vBlocks;
  this.interval = interval;
  this.steps = steps;
  this.finished = false;
  this.blocks = [];
  this.history = [];
  this.currentBlock = undefined;
  this.initBlocks();
};

MazeGenerator.prototype.initBlocks =
function() {
  for (var y = 0; y < this.vBlocks; ++y)
{
    var row = [];
    for (var x = 0; x < this.hBlocks;
++x) {
      row.push(new Block(this, x, y));
    }
    this.blocks.push(row);
  }
};
```

```
MazeGenerator.prototype.drawLoop =
function() {
  for (var c = 0; c < this.steps; ++c) {
    if (!this.finished) {
      this.step();
      this.checkFinished();
    }
  }

  var _this = this;
  this.timeout = setTimeout(function() {
    _this.drawLoop();
  }, this.interval);
};

MazeGenerator.prototype.step = function()
{
  var oldBlock = this.currentBlock;
  var currentBlock = this.currentBlock =
this.chooseBlock();

  if (!currentBlock) { //if there is no
current block then go back to the old
```

```
block
    oldBlock.draw();
    this.finished = true;
    return;
  }
    //else block
  currentBlock.occupied = true;
  if (!currentBlock.inHistory) {
    this.history.push(currentBlock);
    currentBlock.inHistory = true;
  }

  if (oldBlock) {
    if (!oldBlock.hasChild(currentBlock)
&& currentBlock.parent === undefined) {
      oldBlock.connectTo(currentBlock);
    }
    oldBlock.draw();
  }

  currentBlock.draw();
};

MazeGenerator.prototype.chooseBlock =
```

```
function() {
  if (this.currentBlock) {
    var n =
this.currentBlock.randomAvailableNeighbor
();
    if (n) {
      return n;
    } else {
      var b = this.history.pop();
      b && (b.inHistory = false);
      b = this.history.pop();
      b && (b.inHistory = false);
      return b;
    }
  } else {
    var x = Math.floor(Math.random() *
this.hBlocks);
    var y = Math.floor(Math.random() *
this.vBlocks);
    return this.blocks[y][x];
  };
};

MazeGenerator.prototype.checkFinished =
```

```
function() {
  return false;
};

function initMaze() {
  if (window.maze) {
    clearTimeout(window.maze.timeout);
//if maze has already been created then
clear it's timeout, otherwise create the
maze
  }

  var canvasSize =
parseInt(document.getElementById('canvasS
ize').value, 10);
  var blocks =
parseInt(document.getElementById('blocks'
).value, 10);
  var interval =
parseInt(document.getElementById('interva
l').value, 10);
  var steps =
parseInt(document.getElementById('steps')
.value, 10); //all these are user
```

```
inputted
    window.maze = new
MazeGenerator(canvasSize, canvasSize,
blocks, blocks, interval, steps, 'maze');
// call to function MazeGenerator
    // the function forms a square with
the above dimensions
    window.maze.drawLoop();// call to
function drawLoop()
}
```