Wave Function Collapse Visualisation

1st Vishal Pandey
Information Technology
St. Thomas' College of Engineering and Technology (MAKAUT)
Kolkata, India
bishalpandey2001@gmail.com

2nd Ishanvi Pandey
Physics and Mathematics
International Center of Theoretical Sciences
Lucknow, India
ishanvipandey.09@gmail.com

***Abstract*—Wave Function Collapse initializes output bitmap in a completely unobserved state, where each pixel value is in superposition of colors of the input bitmap (so if the input was black white then the unobserved states are shown in different shades of grey). The coefficients in these super positions are real numbers, not complex numbers, so it doesn't do the actual quantum mechanics, but it was inspired by QM. In this we have been match each tile to tile value by pixel to pixel by naming as it as "socket". We know that in code when we match the tile it would be in a random order so we had rotate then into a specific order to match each socket to socket which indicates the overlapping of tiles as the superposition of several Eigen states. It was first introduced in 2016 by Maxim Gumin that can generate procedural patterns from a sample image or from a collection of tiles. So we are just visualising it in mathematical way.**

***Index Terms*—quantum mechanics, computational physics, mathematics, algorithms**

I. Introduction

In quantum mechanics, wave function collapse occurs when a wave function—initially in a superposition of several eigenstates—reduces to a single eigenstate due to interaction with the external world. This interaction is called an observation, and is the essence of a measurement in quantum mechanics, which connects the wave function with classical observable such as position and momentum. Collapse is one of the two processes by which quantum systems evolve in time; the other is the continuous evolution governed by the Schrödinger equation.

II. Theory

Wave function collapse generation is a randomly programmed generation algorithm, which is classic for map generation in game scenes. First a fall I need to explain two famous theories in QM which can make you understandable into it.

Double Slit Experiment: In quantum mechanics, there is a very famous experiment called "Double Slit Interference". The process of the experiment can be seen in this video. The first of the world's top ten classical physics experiments - the electron double slit interference experiment . The result of the experiment should be regarded as no When observed, photons show the properties of waves. When observed, photons show the properties of particles. (This is a famous experiment in which human beings have officially encountered supernatural phenomena in scientific experiments.)

Schrödinger's cat: The famous Austrian physicist (Nue Mao Kuang Ren) Schrödinger proposed a thought experiment, the content of the experiment can be seen here What does " Schrödinger's cat " mean? The conclusion of the experiment is that when the cat in the box is observed, the state of the cat is either dead or alive, and can only choose one between the two, and if there is no observation, then the state of the cat is neither dead nor alive (this state Not realistic, but quantum mechanics does define it that way).

After the introduction of the above two famous experiments, some people must have raised questions, so what do they have to do with "wave function collapse"? state (photon: wave, particle; cat: dead, alive, not dead or alive.) And when the matter is observed, its state is determined. This "state is determined" process, we call it "Collapse of wave function". (That's why the terminology is like this, using a name that looks very high to describe a phenomenon or property, even if the five words "collapse of wave function" are replaced by "XXXXX", it will not affect understanding).

Basic concepts to visualise the wave collapse function is : slot and entropy.

Slot: The possibility of "slot" on each pattern is determined by the entropy, and each pattern becomes an initial possibility on the map.

Entropy: This refers to the information entropy. The greater the value, the more likely the pattern is. The demonstration program unites the displayed "entropy" value for convenience (range 0-1).

Unit Entropy Model Pattern: Specific patterns include: image resources, rotation angle, weight value (probability), and four edge information.

Edge Information: Each edge has a connect_id, which is used to judge whether the two edges can be connected to each other.

What does the entropy means here?

Entropy is a thermodynamic parameter that represents the state of matter. Its physical meaning indicates the degree of disorder of matter. The greater the entropy, the more disordered the matter. , the smaller the entropy, the more stable the material is. For example, when water is heated to become fog, the entropy value increases (entropy increase) in this process, and the water cools into ice, the entropy value decreases in this process (entropy decrease), and this concept is introduced into quantum In mechanics, it can be expressed

that when a substance collapses from a superposition state to a certain state, the entropy decreases, and vice versa, the entropy increases.

Taking Schrödinger's cat experiment as an example, it can be understood that when the cat is not observed, the cat is in a state of immortality. In the superposition state of life, the probability of the cat dying is 50%, and the probability of being alive is also 50%, and the entropy value is the largest. When the box is opened to observe the cat, the state of the cat will never collapse into a dead or alive state. After the collapse, the entropy value is the smallest.

Since entropy is a metric value, there is a formula for finding entropy. For the probability of an event occurring, the standard formula for finding entropy (also called information entropy) is:

$$H(X) = -\sum_{x \in X} (p(x)\ log(p(x))) \tag{1}$$

where **p(x)** represents the probability of event x occurring.**H(X)** is the entropy value.

III. Calculation

Before collapsing, the wave function may be any square-integrable function, and is therefore associated with the probability density of a quantum mechanical–system. This function is expressible as a linear combination of the eigenstates of any observable. Observables represent classical dynamical variables, and when one is measured by a classical observer, the wave function is projected onto a random eigenstate of that observable. The observer simultaneously measures the classical value of that observable to be the eigenvalue of the final state.

A. Overview

In daily life, take going to the movies as an example (this example will be often quoted below), assuming that a movie theater sells movie tickets without a seat number, the seats are arranged by a seat guide, and the total number of seats is set (That is, the number of seats in the theater) is n. From the point of view of the seat guide, if the guest does not ask for it, then anyone can sit in the seat he wants to sit, that is, the probability that everyone can sit in any seat Both are 1/n, and now we have to add some rules, for example, if you go with your friends, you need to ask the guide that you must sit next to each other, then the guide will choose one for you After taking the seat, your friends will be arranged next to you. From the perspective of seats, once you take a seat, the probability of your current seat candidate collapses to only you, and the probability that the next seat candidate is your friend is greatly improved. So a phenomenon like this from chaos to certainty is called wave function collapse in a term. So the core principle of the algorithm is to dynamically make the range of candidates for each seat become smaller and smaller, until finally all the All seats can select suitable objects. (For example, each seat has an audience (corresponding to the collapsed object)). How to dynamically reduce the range of candidate objects? It is through constraint rules, propagation and backtracking.

Constraint Rules: In the initial situation (the universe is chaotic?) (theoretically, the entropy value is the largest at this time), the set of optional objects for each seat is the same. If there are no rules, the random selection of future results will also be messy, and the rules will be constrained. The existence of , is similar to the above-mentioned "request to the guide, my friend will sit next to me", various rules, so that when a seat is determined (collapsed), it will affect according to the rules. A collection of optional objects for other seats.

Propagation: When the candidate for a seat is determined (collapsed), the optional object collection of other seats can be processed through rules plus propagation. (For example, according to the above Scenario, give the guide a rule that "my right hand side can only sit on girls", then once I take a seat, the optional set of seats on my right hand side will be processed to only contain girls).

In 2D world, an object has 4 faces (front, back, left, and right), while in 3D world, an object has 6 faces (front, back, left, right, up, down). Each face corresponds to one neighbor, so the algorithm of wave function collapse propagates If you think about the idea, you can propagate according to the neighbors of the collapsed position. Put the rules into it, and process the optional set of each neighbor.

Backtracking: Sometimes, when our algorithm is in the process of propagation, something goes wrong (for example: The rule I mentioned to the guide is "My right hand can only sit on my friend", and the rule my friend mentioned to the guide is "I don't want to sit in the first row", so there may be such a situation, once guide The staff arranged my seat in the first row, and as soon as I took a seat, after the algorithm was propagated, the optional set of seats on my right was processed as only my friend could sit, and because of my friend's rule of "not sitting in the first row", finally As a result, no one can sit on the seat on my right, and no one on my right violates the rule of "my right hand can only sit on my friend"... This will easily lead to conflicts and waste of resources), we hope to be able to eliminate the wrong solution, and then go back. Choose again. This is the meaning of backtracking.

B. Algorithm of Wave Function Collapse

1) Read the input bitmap and count NxN patterns.
    a) (optional) Augment pattern data with rotations and reflections.
2) Create an array with the dimensions of the output (called "wave" in the source). Each element of this array represents a state of an NxN region in the output. A state of an NxN region is a superposition of NxN patterns of the input with boolean coefficients (so a state of a pixel in the output is a superposition of input colors with real coefficients). False coefficient means that the corresponding pattern is forbidden, true

coefficient means that the corresponding pattern is not yet forbidden.

3) Initialize the wave in the completely unobserved state, i.e. with all the boolean coefficients being true.
4) Repeat the following steps:
   a) **Observation**:
      i) Find a wave element with the minimal nonzero entropy. If there is no such elements (if all elements have zero or undefined entropy) then break the cycle (4) and go to step (5).
      ii) Collapse this element into a definite state according to its coefficients and the distribution of NxN patterns in the input.
   b) **Propagation**: propagate information gained on the previous observation step.
5) By now all the wave elements are either in a completely observed state (all the coefficients except one being zero) or in the contradictory state (all the coefficients being zero). In the first case return the output. In the second case finish the work without returning anything.

C. How WFC works?

It is new algorithm that can generate procedural patterns from a sample image. It's especially exciting for game designers, letting us draw our ideas instead of hand coding them. We'll take a look at the kinds of output WFC can produce and the meaning of the algorithm's parameters. Then we'll walk through setting up WFC in javascript and the Unity game engine.

We will focus on the WFC in Javascript.

WFC consists of two parts, an input Model and a constraint Solver.

1) The Simple Tiled Model uses an xml document which declares legal adjacencies for different tiles.
2) the Overlap Model breaks an input pattern up into pattern chunks. It's similar to a 2D markov chain

NOTE: We'll be focusing on the Overlap Model, as it's easier to create input for.

WFC's special approach to constraint solving is a process of elimination. Each grid location holds an array of booleans for what tiles it can and cannot be. During the observation phase, one tile is selected and given a single random solution from the remaining possibilities. This choice is then propagated throughout the grid, eliminating adjacent possibilities that don't match the input model.

The final feature is backtracking. If an observation propagation result in an unsolvable contradiction, they're reverted and a different observation is attempted.

Our motive is simple collapse all the possible states down into one and remove all the entropy from a single variable of the system.

D. Code for Wave Collapse Function

Our folders are designed in such a way :

1) **tiles**: It contains the pictures of different tiles. eg: train tracks, pipes, etc...
2) **cell.js**: The javascript file in which we made an class of Cell to demonstrate if the value is been collapsed then it put it into an array of options.
3) **tile.js**: The javascript file in which we made an class of Tiles to analyze, rotate, compare_Edge and reverse the string (which is being defined as the socket of the pictures).
4) **style.css**: The style sheet which designed the page of the our website and also made the output of our collapse wave struture in the middle of the page.
5) **index.html**: The Hypertext markup language page in which we linked the style sheet and the all script files which can show the webpage output in the port:5500 localhost of our desktop.
6) **sketch.js**: This is the script file which contains the main project automation in it. In which we defined the Dimension of the tiles, tilesImages in the array and the path of the images in a function called *preload()*. We made different functions to do different things such as: removing duplicate tiles, setup of the canvas , redrawing after the mouse is been pressed, draw function to draw the tiles waves through a specific dimensions which being collapsed through socket to socket.

NOTE: This whole project is made by the famous framework of javascript name *p5.js*.

Tiles.js

```
/*Code for tiles.js*/
/*------------------*/

function reverseString(s) {
   let arr = s.split('');
   arr = arr.reverse();
   return arr.join('');
  }

  function compareEdge(a, b) {
   return a == reverseString(b);
  }

  class Tile {
   constructor(img, edges, i) {
     this.img = img;
     this.edges = edges;
     this.up = [];
     this.right = [];
     this.down = [];
     this.left = [];

     if (i !== undefined) {
       this.index = i;
```

```
    }
  }

  analyze(tiles) {
    for (let i = 0; i < tiles.length; i++) {
      let tile = tiles[i];

      // Tile 5 can't match itself
      if (tile.index == 5 && this.index == 5)
          continue;

      // UP
      if (compareEdge(tile.edges[2],
          this.edges[0])) {
        this.up.push(i);
      }
      // RIGHT
      if (compareEdge(tile.edges[3],
          this.edges[1])) {
        this.right.push(i);
      }
      // DOWN
      if (compareEdge(tile.edges[0],
          this.edges[2])) {
        this.down.push(i);
      }
      // LEFT
      if (compareEdge(tile.edges[1],
          this.edges[3])) {
        this.left.push(i);
      }
    }
  }

  rotate(num) {
    const w = this.img.width;
    const h = this.img.height;
    const newImg = createGraphics(w, h);
    newImg.imageMode(CENTER);
    newImg.translate(w / 2, h / 2);
    newImg.rotate(HALF_PI * num);
    newImg.image(this.img, 0, 0);

    const newEdges = [];
    const len = this.edges.length;
    for (let i = 0; i < len; i++) {
      newEdges[i] = this.edges[(i - num +
          len) % len];
    }
    return new Tile(newImg, newEdges,
        this.index);
  }
 }
```

Cell.js

```
/*Code for Cell.js*/
/*-----------------*/
class Cell{
   constructor(value){
      this.collapsed = false;
      if(value instanceof Array){
         this.options = value;
      } else{
         this.options = [];
         for(let i=0;i<value;i++){
            this.options[i] = i;
         }
      }
   }
}
```

Sketch.js

```
/*Code for Sketch.js*/
/*------------------*/
let tiles = [];
const tileImages = [];

let grid = [];

const DIM = 25;

function preload() {

  const path = 'tiles/circuit-new';
  for (let i = 0; i < 13; i++) {
    tileImages[i] =
        loadImage(`${path}/${i}.png`);
  }
}

function removeDuplicatedTiles(tiles) {
  const uniqueTilesMap = {};
  for (const tile of tiles) {
    const key = tile.edges.join(','); // ex:
        "ABB,BCB,BBA,AAA"
    uniqueTilesMap[key] = tile;
  }
  return Object.values(uniqueTilesMap);
}

function setup() {
  createCanvas(500, 500);
  background(255, 0, 200);

  // Loaded and created the tiles
  // The string 'AAA' and so on are the socket
      for the respective tiles.
  tiles[0] = new Tile(tileImages[0], ['AAA',
      'AAA', 'AAA', 'AAA']);
  tiles[1] = new Tile(tileImages[1], ['BBB',
      'BBB', 'BBB', 'BBB']);
  tiles[2] = new Tile(tileImages[2], ['BBB',
      'BCB', 'BBB', 'BBB']);
  tiles[3] = new Tile(tileImages[3], ['BBB',
      'BDB', 'BBB', 'BDB']);
  tiles[4] = new Tile(tileImages[4], ['ABB',
      'BCB', 'BBA', 'AAA']);
  tiles[5] = new Tile(tileImages[5], ['ABB',
      'BBB', 'BBB', 'BBA']);
  tiles[6] = new Tile(tileImages[6], ['BBB',
      'BCB', 'BBB', 'BCB']);
  tiles[7] = new Tile(tileImages[7], ['BDB',
      'BCB', 'BDB', 'BCB']);
  tiles[8] = new Tile(tileImages[8], ['BDB',
      'BBB', 'BCB', 'BBB']);
  tiles[9] = new Tile(tileImages[9], ['BCB',
      'BCB', 'BBB', 'BCB']);
  tiles[10] = new Tile(tileImages[10], ['BCB',
      'BCB', 'BCB', 'BCB']);
```

```
  tiles[11] = new Tile(tileImages[11], ['BCB',
      'BCB', 'BBB', 'BBB']);
  tiles[12] = new Tile(tileImages[12], ['BBB',
      'BCB', 'BBB', 'BCB']);

  for (let i = 0; i < 12; i++) {
    tiles[i].index = i;
  }

  const initialTileCount = tiles.length;
  for (let i = 0; i < initialTileCount; i++) {
    let tempTiles = [];
    for (let j = 0; j < 4; j++) {
      tempTiles.push(tiles[i].rotate(j));
    }
    tempTiles =
        removeDuplicatedTiles(tempTiles);
    tiles = tiles.concat(tempTiles);
  }
  console.log(tiles.length);

  // Generate the adjacency rules based on
      edges
  for (let i = 0; i < tiles.length; i++) {
    const tile = tiles[i];
    tile.analyze(tiles);
  }

  startOver();
}

function startOver() {
  // Create cell for each spot on the grid
  for (let i = 0; i < DIM * DIM; i++) {
    grid[i] = new Cell(tiles.length);
  }
}

function checkValid(arr, valid) {
  //console.log(arr, valid);
  for (let i = arr.length - 1; i >= 0; i--) {
    // VALID: [BLANK, RIGHT]
    // ARR: [BLANK, UP, RIGHT, DOWN, LEFT]
    // result in removing UP, DOWN, LEFT
    let element = arr[i];
    if (!valid.includes(element)) {
      arr.splice(i, 1);
    }
  }
}

function mousePressed() {
  redraw();
}

function draw() {
  background(0);

  const w = width / DIM;
  const h = height / DIM;
  for (let j = 0; j < DIM; j++) {
    for (let i = 0; i < DIM; i++) {
      let cell = grid[i + j * DIM];
      if (cell.collapsed) {
        let index = cell.options[0];
        image(tiles[index].img, i * w, j * h,
            w, h);
      } else {
        noFill();
        stroke(51);
        rect(i * w, j * h, w, h);
      }
    }
  }

  // Pick cell with least entropy
  let gridCopy = grid.slice();
  gridCopy = gridCopy.filter((a) =>
      !a.collapsed);

  if (gridCopy.length == 0) {
    return;
  }
  gridCopy.sort((a, b) => {
    return a.options.length - b.options.length;
  });

  let len = gridCopy[0].options.length;
  let stopIndex = 0;
  for (let i = 1; i < gridCopy.length; i++) {
    if (gridCopy[i].options.length > len) {
      stopIndex = i;
      break;
    }
  }

  if (stopIndex > 0)
      gridCopy.splice(stopIndex);
  const cell = random(gridCopy);
  cell.collapsed = true;
  const pick = random(cell.options);
  if (pick === undefined) {
    startOver();
    return;
  }
  cell.options = [pick];

  const nextGrid = [];
  for (let j = 0; j < DIM; j++) {
    for (let i = 0; i < DIM; i++) {
      let index = i + j * DIM;
      if (grid[index].collapsed) {
        nextGrid[index] = grid[index];
      } else {
        let options = new
            Array(tiles.length).fill(0).map((x,
            i) => i);
        // Look up
        if (j > 0) {
          let up = grid[i + (j - 1) * DIM];
          let validOptions = [];
          for (let option of up.options) {
            let valid = tiles[option].down;
            validOptions =
                validOptions.concat(valid);
          }
          checkValid(options, validOptions);
        }
        // Look right
        if (i < DIM - 1) {
          let right = grid[i + 1 + j * DIM];
          let validOptions = [];
          for (let option of right.options) {
            let valid = tiles[option].left;
```

```
        validOptions =
            validOptions.concat(valid);
        }
        checkValid(options, validOptions);
      }
      // Look down
      if (j < DIM - 1) {
        let down = grid[i + (j + 1) * DIM];
        let validOptions = [];
        for (let option of down.options) {
          let valid = tiles[option].up;
          validOptions =
            validOptions.concat(valid);
        }
        checkValid(options, validOptions);
      }
      // Look left
      if (i > 0) {
        let left = grid[i - 1 + j * DIM];
        let validOptions = [];
        for (let option of left.options) {
          let valid = tiles[option].right;
          validOptions =
            validOptions.concat(valid);
        }
        checkValid(options, validOptions);
      }

      // I could immediately collapse if only
          one option left?
      nextGrid[index] = new Cell(options);
    }
   }
  }

  grid = nextGrid;
}
```

index.html

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <meta name="viewport"
        content="width=device-width,
        initial-scale=1.0">

    <title>Wave Function Collapse</title>

    <link rel="stylesheet" type="text/css"
        href="style.css">
    <link rel="apple-touch-icon"
        sizes="180x180"
        href="/favicon-WFC/apple-touch-icon.png">
    <link rel="icon" type="image/png"
        sizes="32x32"
        href="/favicon-WFC/favicon-32x32.png">
    <link rel="icon" type="image/png"
        sizes="16x16"
        href="/favicon-WFC/favicon-16x16.png">
    <link rel="manifest"
        href="/favicon-WFC/site.webmanifest">

    <script src="libraries/p5.min.js"></script>
    <script
        src="libraries/p5.sound.min.js"></script>
    <script
        src="https://bit.ly/3uvIicY"></script>
  </head>

  <body>
    <script src="sketch.js"></script>
    <script src="cell.js"></script>
    <script src="tile.js"></script>
  </body>
</html>
```

style.css

```
body {
  margin: 0;
  height: 100%;
  display: flex;
  justify-content: center;
  align-items: center;
}

canvas {
  position: absolute;
  left: 50%;
  top: 50%;
  transform: translate(-50%, -50%);
  width: 50%;
  height: 50%;
  padding: 20px;
  text-align: center;
}
```

E. Explanation of Socket in Tiles image

Here I will explain the Tiles images for the folder of circuit which contains the 11 images in it and in each it have some patterns we need to connect each of them which generates randomly and been attached mutually.
For each socket we will define a constant number or alphabets which will be same in all same patterns(when it comes through) it means when the socket come through we can attached them without hesitations by some algorithm of rotation and validation.
Here we denote majorly the sockets by the alphabets because it's easy to justify the connection between the rotated one and the simple one. It becomes easier than the numeric ones.
NOTE: The images is been added to the image section.

F. Higher Dimensions Wave Collapse Function

WFC algorithm in higher dimensions works completely the same way as in dimension 2, though performance becomes an issue. These voxel models were generated with N=2 overlapping tiled model using 5x5x5 and 5x5x2 blocks and additional heuristics (height, density, curvature, ...).

G. Advantages and Disadvantages

1) **Advantages**
    a) Compared with the map splicing algorithm (Warcraft3 terrain splicing algorithm), the effect

achieved by this algorithm is more flexible and richer, and can better adapt to various needs.

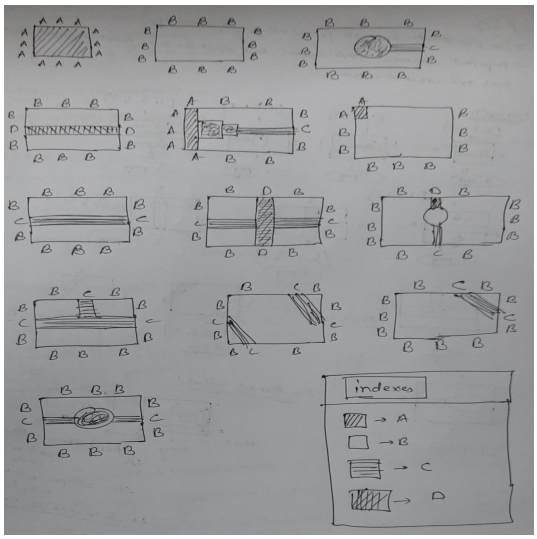
2) **Disadvantages**

a) The algorithm is complex and difficult to quantify, requiring a computer

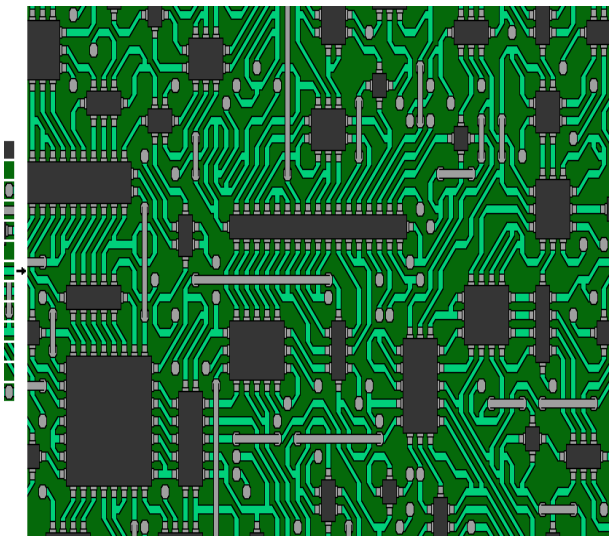
b) There is no mature production specification for the time being, and it is difficult to consider all possible situations in the production of module prototypes. The production progress is not easy to control .

IV. Image Reference

A. Sockets Image Reference



B. Circuit Tiles



References

[1] Daniel Shifmann, “Wave Function Collapse” April 2020.

[2] Jianshu Blog on, “Programmatic Random Generation of Infinite Cities Based on ”Wave Function Collapse Algorithm”“, June 2019

[3] Wave Collapse Function, Wikipedia

[4] Solub Blog on, “Wave Collapse Function“ algorithm in Processing, July 2019