Experiment 03: Alternate Worlds

Chengkun Li

Computational Media, University of California - Santa Cruz

CMPM 147: Generative Design

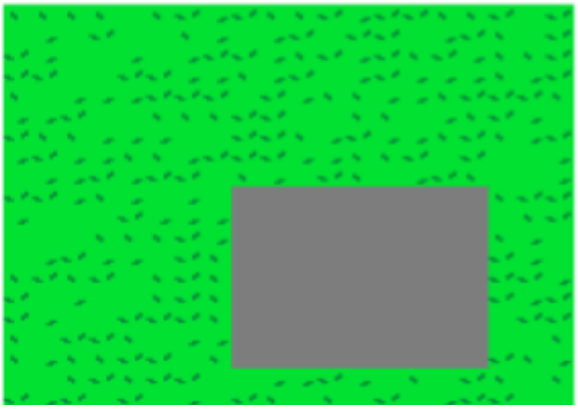
Prof. Wes Modes

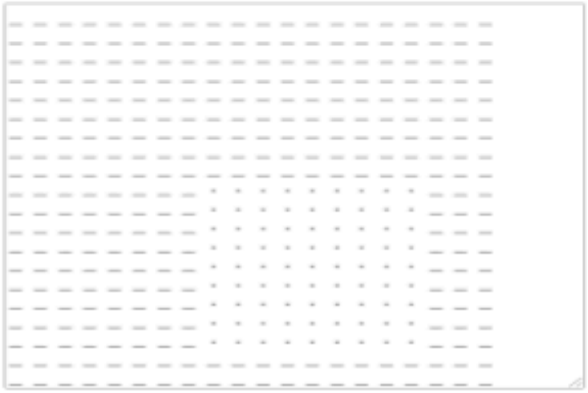
4/21/2025

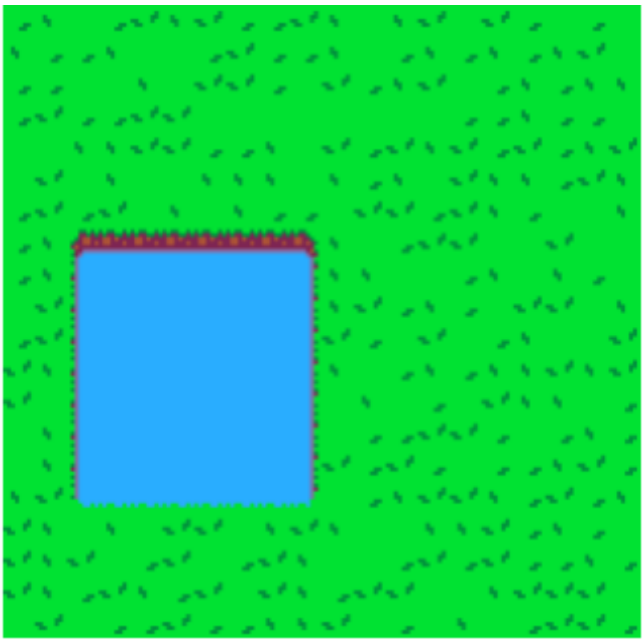
Step 1 - Imitate:

Following the template, initial steps, and notes from the professor, I got the image below at the beginning, which randomly generates a random-sized rectangle.

```
let room_width = floor(random(numCols * 0.3, numCols * 0.5));
let room_height =  floor(random(numRows * 0.3, numRows * 0.5));

let room_start_col = floor(random(1, numCols - room_width - 1));
let room_start_row =  floor(random(1, numRows - room_height -1));

for(let i = room_start_row; i < room_start_row + room_height; i++){
  for(let j = room_start_col; j < room_start_col + room_width; j++){
    grid[i][j] = ".";
  }
}
```

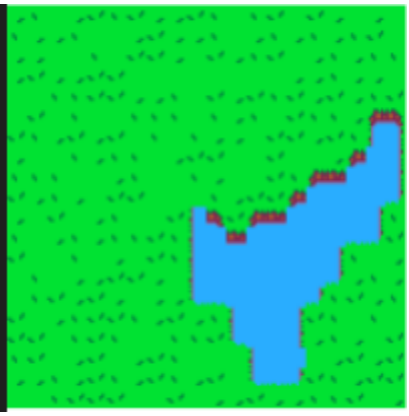






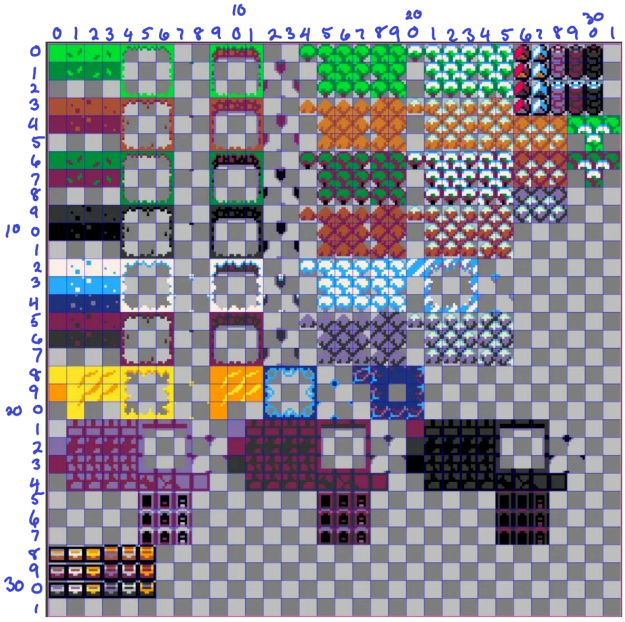
However, the rectangle is transparent, and then I picked a blue element from the tileset to fill in the rectangle. After that, I followed the steps and finished the gridCheck(), gridCode(), lookup_water, and drawContext() functions. The image on the left side is what I got.

```
const noiseScale = 0.08;

for (let i = 0; i < numRows; i++) {
    for (let j = 0; j < numCols; j++) {
        const noiseVal = noise(j * noiseScale, i * noiseScale);

        if (noiseVal >= 0   && noiseVal < 0.4) {
            grid[i][j] = "w"; // water
        }
        // otherwise "_"
    }
}
```



Moreover, I followed the steps to use the noise() function to generate the world more randomly. The image on top is what I got.

At this point, I need to thank Jack Sims and Jackie Sanchez for their improved tileset. The tileset they made saved me a lot of time. The original tileset is too hard to see where each element is. The improved tileset makes more easier to find the element I want. The original tileset is too monotonous, with only grass and water. Therefore, I want to add more elements to the tileset, such as trees and houses.



Step 2 - Integrate:

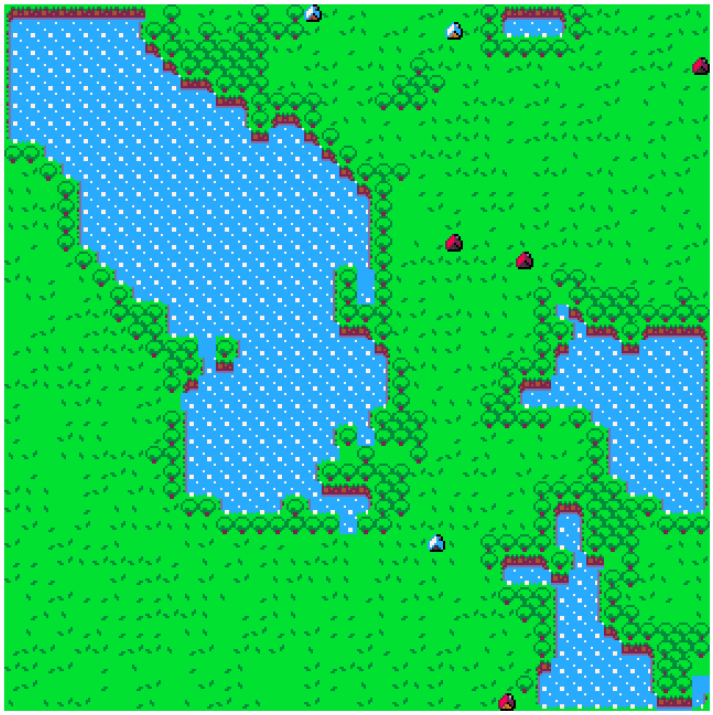
Based on the idea, I created lookup_forest, used to check the edge of the forest. To create lookup_forest, I used Sonny Bone's article "How to Use Tile Bitmasking to Auto-Tile Your Level Layouts" to learn how to calculate edge cases. At the beginning, I used [14, 0], which is a single tree element, as the no-neighbors situation, and then calculated other edge cases. The code used is "drawContext(grid, i, j, "w", 14, 0);". But later I found that it was too much trouble to calculate the edge cases in this way, because I had to keep adding and subtracting parameters to get the correct element. Later, I thought of "drawContext(grid, i, j, "w", 0, 0);" instead, because

in this way, I don't need to calculate, I just need to check which column and row the required element is in the tileset.

Based on this idea, I reworked lookup_water and lookup_forest. This part took me a long time to implement. I literally spent hours testing every neighborhood combination to verify that the transitions between forest, grass, and water were rendering correctly. The whole process was very painful for me, but overall it was worth it.

```
const lookup_water = [
    [0,0], // 0000 - No neighbors
    [30,2], // 0001 - N only
    [30,0], // 0010 - S only
    [0,13], // NS
    [0,0], // E only
    [9,2], // N+E
    [9,0], // S+E
    [9,1], // NSE
    [0,0], // W only
    [11,2], // N+W
    [11,0], // S+W
    [11,1], // NSW
    [10,0], // E+W
    [10,2], // N+E+W
    [10,0], // S+E+W
    [0,2]  // 1111 - All four neighbors
];
```

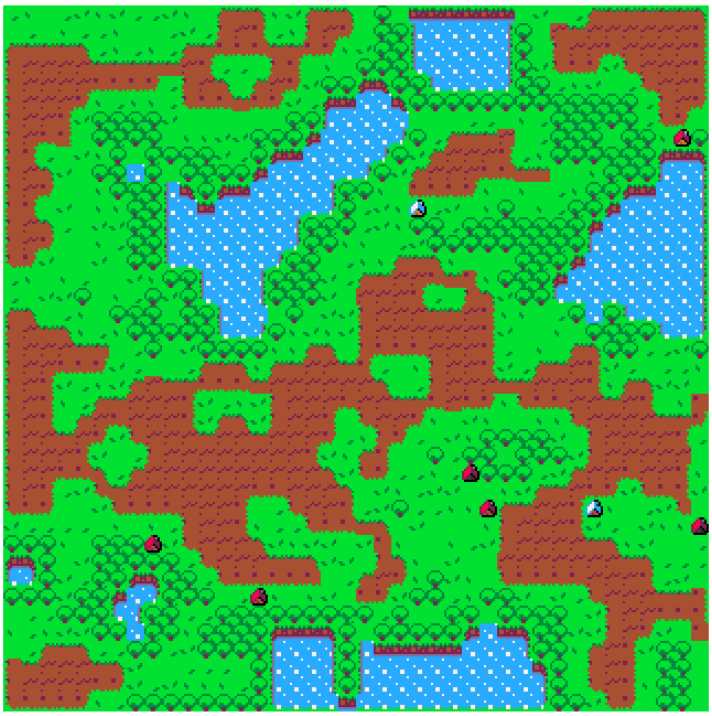
```
const lookup_forest = [
    [14,0], // 0000 - No neighbors
    [14,0], // 0001 - N only
    [14,0], // 0010 - S only
    [14,0], // 0011 - NS
    [14,0], // 0100 - E only
    [15,2], // 0101 - N+E
    [15,0], // 0110 - S+E
    [15,1], // 0111 - NSE
    [14,0], // 1000 - W only
    [17,2], // 1001 - N+W
    [17,0], // 1010 - S+W
    [17,1], // 1011 - NSW
    [14,0], // 1100 - E+W
    [16,2], // 1101 - N+E+W
    [16,0], // 1110 - S+E+W
    [16,1]  // 1111 - All four neighbors
];
```



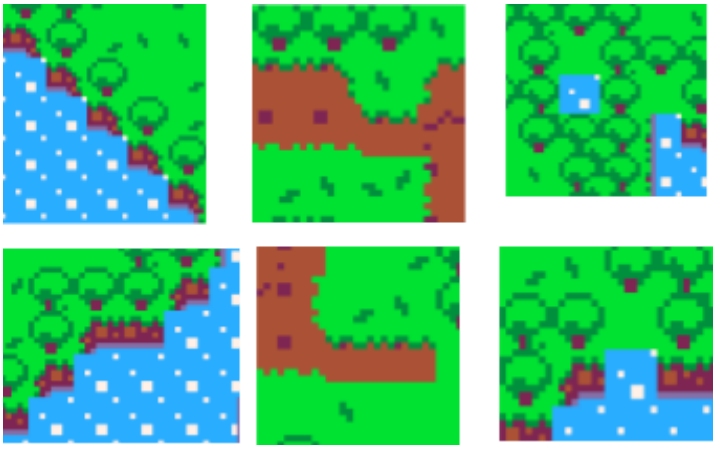
I didn’t make a lookup edge for the house because the houses are all single elements, which I can make it place on the world randomly and avoid water areas.

The image on the left is the result after I completed the above steps. Overall, it is not bad. However, there are still obvious rendering defects in some places. For example, when an element is S-only or E-only.

And I still thought something was missing. Soon, I realized it was the roads. Then I adjusted the noise() function so that the world could generate reasonable roads, water, trees, and houses. Similar to lookup_water and lookup_forest, I created lookup_i to check the edge condition of the road.



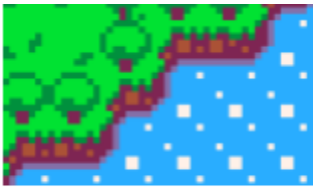
Step 3 - Innovate:

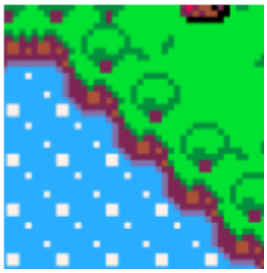


Although the world generated by the generator looks good overall, many rendering defects can be seen if you look closely, such as the picture on the left.

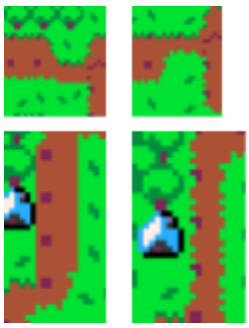
Somehow, I realized that the corners of the water can be replaced by [12, 1] and [13, 1] from the tileset. That is, only when the upper left corner [i-1][j-1] and the upper right corner [i+1][j-1] of the current position [i][j] are not water elements, then the current position is at the corner of the shore. So I can add an if() statement to catch this situation. The following image is the result after I applied the if() statement.

```
// For the corner of the shore
if(!gridCheck(grid, i-1, j+1, "w") &&
    gridCheck(grid, i-1, j, "w") &&
    gridCheck(grid, i, j+1, "w")){
  placeTile(i, j, 12, 1);
}
//For the corner of the shore
if(!gridCheck(grid, i-1, j-1, "w") &&
    gridCheck(grid, i-1, j, "w") &&
    gridCheck(grid, i, j-1, "w")){
  placeTile(i, j, 13, 1);
}
```

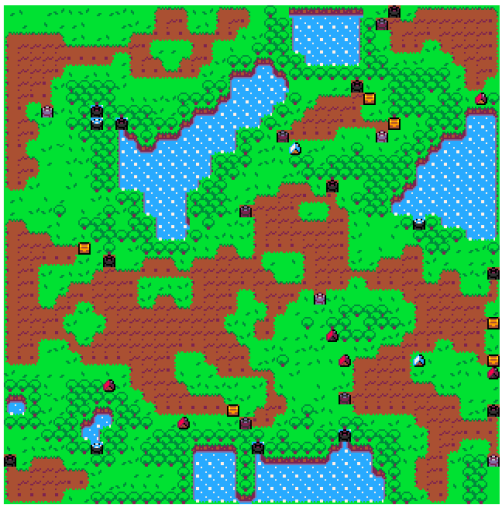




Later, I wondered if I could apply this to the rendering of roads. The problem with road rendering is that when [i][j] is NS or EW, the road will only render one element, making the other side smooth without a transition effect. Then I thought of using an if() statement to determine whether the current position [i][j] is NS or EW, and if so, put two elements. The following image is the result after I applied the if() statement.



```
if((!gridCheck(grid, i-1, j, "i")) && (!gridCheck(grid, i+1, j
    gridCode(grid, i, j, "i") != 8 && gridCode(grid, i, j, "i
  placeTile(i, j, 5, 0);
  placeTile(i, j, 5, 2);
}
if((!gridCheck(grid, i, j-1, "i")) && (!gridCheck(grid, i, j+1
    gridCode(grid, i, j, "i") != 1 && gridCode(grid, i, j, "i
  placeTile(i, j, 4, 1);
  placeTile(i, j, 6, 1);
}
```

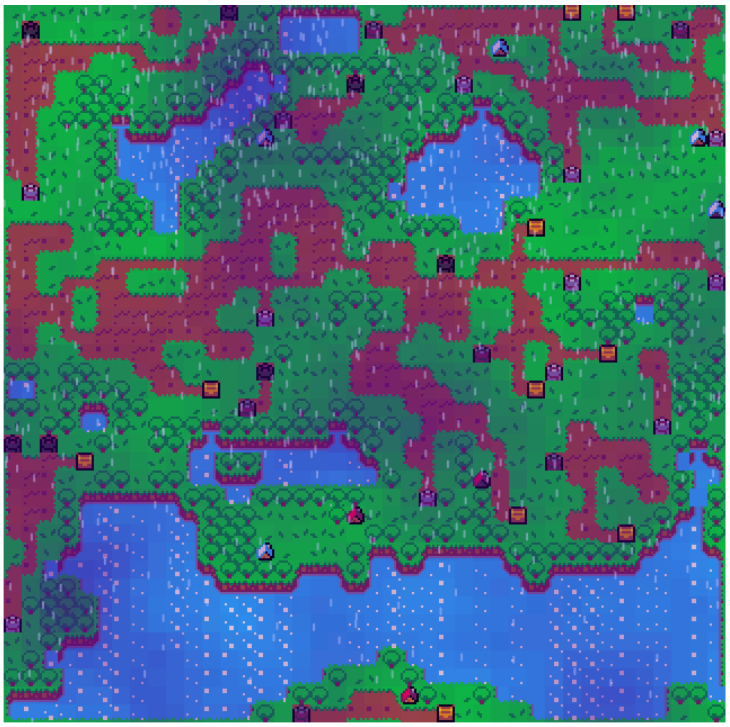


In addition to the above rendering error problems, there are also N-only, S-only or E-only, and W-only problems. For these, I used the treasure chest, pillar, and house elements in the tileset instead. The effect is generally good, and it is difficult to see any rendering problems at this time. The image on the left side is the result of my application of the above steps.

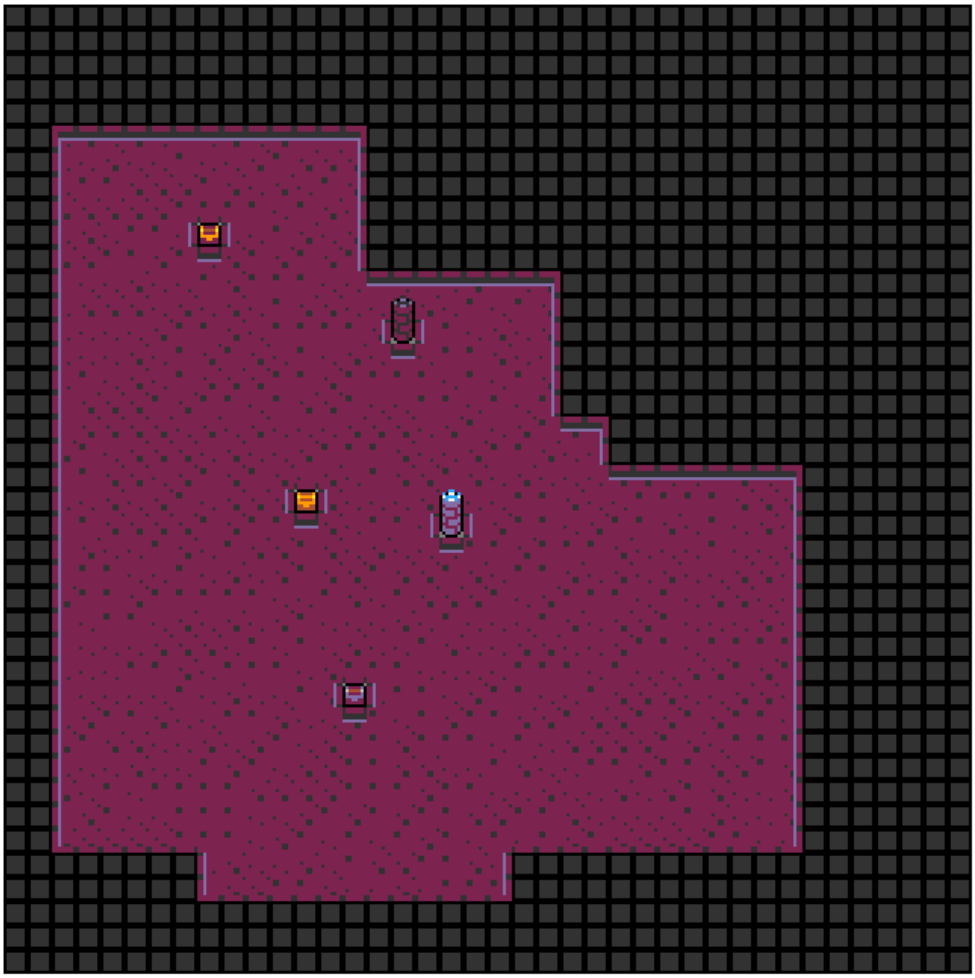
```
// cloud-shadow overlay
let t = millis() * 0.0002;
let b = noise(j*0.1 + t, i*0.1 + t);
let a = map(b, 0, 1, 0, 200);
noStroke();
fill(80, 0, 150, a);
rect(j*16, i*16, 16, 16);
//rain
const rainCount = 1;
const rainLength = 5;
const rainSpeed = 0.5;
stroke(200, 200, 255, 120);
strokeWeight(2);
for(let r = 0; r < rainCount; r++){
  let x = random(width)
  let y = (random(height) + frameCount * rainSpeed) % height;
  line(x, y, x, y + rainLength);
}
noStroke();
```

```
if (gridCheck(grid, i, j, "w")) {
  const offset = floor(noise(i * 0.001, j * 0.1) * 100);
  const frame  = (floor(millis() / 900) + offset) % 2;
  placeTile(i, j, frame, 13);
  drawContext(grid, i, j, "w", 0, 0);
```

However, I think this result is still missing something. Animation — this is what I think is missing. Then I thought about the weather in daily life, such as strong winds, heavy rain, and snow. Then I thought about whether I could use a line to represent the rain, for example, using a line falling from top to bottom to simulate rain. I used the rain p5.js example by kelsierose94 as a reference to create my rain, and asked ChatGPT to simplify his code to work for my scene. After that, I asked ChatGPT again to animate the clouds and water. This inquiry process was very lengthy, but overall, it fits my scenario very well. Finally, the following pictures are the code I adjusted and obtained, as well as the generated results.



Based on the above code experience, I made a dungeon generator again. The overall code flow has not changed much, only the dungeon generation method has changed. In the upper world generator, I used the noise() function to generate the map, but I did not use it in the dungeon generator because after I tried it, I felt that its results did not match the effect I wanted for the dungeon. Therefore, I simply randomly generated several rectangles in the map to represent the dungeon rooms. The dungeon creation process is relatively simple, because it does not use the noise() function, so the dungeon does not generate a relatively duplicate terrain.



At this point, I think my generator is complete. The whole process can be said to be very painful, but seeing the final result, I think it is all worth it. There are still some rendering issues in the generator. If I have time in the future, I will come back and try to fix it. And you are welcome to take a look at my code, website, Overworld generator, and Dungeon Generator.

Self Evaluation Rubric

Completion: 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? I completed it on time
	X	☐	☐	☐	☐	
Effort: Did you put in earnest effort in execution and attention to detail?	Excellent	Pretty good	About average	Could be improved	Not this time	What supports this? I went into great detail on rendering edges, such as the corners of the shore. I spent a lot of time resolving them.
	☐	X		☐	☐	
Results: How well did you accomplished the assignment goals? Was it complete, with minimal errors, correct output, and good style?	Excellent	Pretty good	About average	Could be improved	Not this time	What supports this? The assignment was completed without errors. At least, there were no errors during the run. The style is okay, I added some animations to make the generated world look a bit more alive.
	☐	X	☐	☐	☐	
Reflection: How much did you reflect on the strengths and weaknesses of the work and what you'd like to improve?	Excellent	Pretty good	About average	Could be improved	Not this time	What supports this? I am satisfied with my results. However, sometimes I will forget to take screenshots of the code and the generated results during the creation process and lose details. In the future, I will try to record it after writing a piece of code.
	☐	X	☐	☐	☐	

Reflection:

Although the process was full of twists and turns. I am quite satisfied with the final result, because my friends said it looks like a pixel game. In the coding process, I find the most annoying thing is identifying edge cases, because I need to constantly determine the row and column where the element is located. What I am most satisfied with is that I fixed the rendering problems in the corners of the shore and added cloud and rain animation effects. What I am most satisfied with is that I fixed the rendering problems in the corners of the shore and added cloud and rain animation effects.

ChatGPT is also a very useful tool that helps me find my mistakes quickly, which helps me save a lot of time. But sometimes it's also very confusing. For example, answering something that is completely irrelevant to me. I have to say that the animation effect of the rain and clouds is quite good.

After this assignment, I realized that there are big differences between p5.js and Phaser in using tilemaps, but it may also be because I haven't been exposed to it deeply enough. It is very fun to have a world generator.

Screenshots: