Navigating a City Grid: Autonomous Car Pathfinding

Introduction

Imagine a little city grid on your computer screen, with roads, buildings, and trees scattered around. Now picture a few tiny cars trying to make their way across this city, navigating around obstacles to find a parking spot. That's what I built in this project: a mini-world where autonomous cars find their own paths from random starting points to specific destinations.

Using JavaScript and a library called p5.js, I set up this city grid and programmed each car to figure out how to get to its "parking spot" as efficiently as possible, without running into obstacles—or each other! I used a pathfinding algorithm called **A***.

Background and Inspiration

When I started, I knew that for these cars to move smartly around the grid, I carefully analysed the A star world (https://ancientbrain.com/world.php?world=2230793148) and it's algorithms, also go through some tutorials related to A* (https://www.youtube.com/watch?v=aKYlikFAV4k) in order to understand A* in-depth. A* is like the car's "inner compass," helping it find the shortest route to its destination by "guessing" which direction is best based on how close it is to the goal.

For my grid world I took inspiration from video games named "Fire Emblem Series (Nintendo)" is classic grid-based tactical RPGs where characters move across a grid, navigating obstacles like walls, rivers, and forests. With limited movement per turn, positioning and pathfinding, are key demonstrating efficient unit movement and terrain management gave me ideas for managing roads and obstacles, while real autonomous vehicle research inspired my obstacle-avoidance techniques.

Challenges I Faced (and How I Solved Them)

Of course, things weren't as straightforward as I'd hoped. Here are a few roadblocks I ran into along the way—and how I got around them:

1. **Obstacle Placement (Avoiding Dead Ends!)**
   I wanted a good mix of buildings, trees, and open roads. But randomly placing obstacles sometimes led to entire sections of the grid being cut off, with no possible path for the cars. To fix this, I added a "pathExist()" that ensures there's always at least one path from the start point to parking point for the car across the grid and also if another car come in the path, car either wait for other to move or move on the new path to reach its destination.

2. **Efficient Pathfinding**
   A* did most of the heavy lifting for me here, helping each car find its way around obstacles by considering both the distance and the blocked paths. The algorithm looks at possible moves, prioritizing directions that bring the car closer to the goal while avoiding anything that's in the way. Since we are calling this function each time I knew improving it time complexity is really necessary. Initially A* was using Array but I modify it use min-heap to give faster result. To implement min-heap in javascript referred GeekforGeek (https://www.geeksforgeeks.org/min-heap-in-javascript/) and tutorial (https://www.youtube.com/watch?v=dM_JHpfFITs)

Below is the comparison in time complexity of A* using Array and min-heap.

Operation	Array (unsorted)	Min-Heap
Insert node	O(1)	O(log n)
Find minimum f-cost node	O(n)	O(1)
Remove minimum f-cost node	O(n)	O(log n)

3. **Avoiding Collisions Between Cars**
   With several cars moving around at once, there was always a risk of them bumping into each other. To prevent that, I set up a turn-based system: only one car moves at a time, taking turns in sequence. This way, they never try to occupy the same space, and the grid stays organized and clutter-free.

Testing and Results

Once everything was set up, I put my cars through a few tests to see how well they navigated different city layouts. Here's how things went:

1. **Pathfinding Speed**
   With a moderate number of obstacles, the A* algorithm performed great! Even in fairly dense grids (up to 25x25 cells), each car found its path quickly—usually in under a second. When the grid was packed with obstacles, A* took a little longer, but it still found a path without getting overwhelmed.

2. **Smooth, Collision-Free Movement**
   The turn-based movement system worked as I'd hoped. Each car waited its turn, moving one step at a time without running into others. I kept track of each car's position in every "turn" to make sure they didn't accidentally share the same space, and it all went smoothly.

3. **Varied Obstacles and Road Layouts**
   I tried a bunch of different obstacle layouts and found that the cars could adapt to each one. The connectivity check ensured they never got stuck, no matter how crowded the grid was.

How It Works (With Some Code!)

Here's a peek behind the curtain at how I made the magic happen.

1. **Placing Obstacles Randomly**
   I used a random function to decide what each cell in the grid would be: a building, a tree, or a road. This randomness kept things interesting and gave the cars a new challenge every time.

```
        // Randomly assign types for buildings, trees, and roads
        let rand = random();
        if (rand < 0.25) {
          this.type = 'building';  // 25% chance to be a building
        } else if (rand < 0.40) {
          this.type = 'tree';  // 15% chance to be a tree
        } else {
          this.type = 'road';  // 60% chance to be a road
        }
      }
```

2. **Pathfinding with A***
   A* helps each car find the shortest path by always choosing the next move that brings it closest to the goal. Here's a snippet of the algorithm in action:

```
this.openSet.insert(start);

while (!this.openSet.isEmpty()) {
  let current = this.openSet.extractMin();

  if (current === destination) {
    this.path = [];
    while (current) {
      this.path.push(current);
      current = current.previous;
    }
    this.path.reverse();
    return this.path;
  }
```

Each car calculates its path based on available roads, steering clear of obstacles.

3. **Marking Destinations with Colours**
   To make it easy to see where each car is headed, I color-coded them! Each car has a unique colour for itself and its goal, so you can follow their paths easily.

```
  // Show destination
if (destinations.includes(this) && !this.car) {
    let car = cars.find(car => car.endSpot === this);
        switch(car.color) {
          case "orange":
            image(images.goalOrange, x, y, cellSize, cellSize);
            break;
          case "blue":
            image(images.goalBlue, x, y, cellSize, cellSize);
            break;
          case "green":
            image(images.goalGreen, x, y, cellSize, cellSize);
            break;
          case "black":
            image(images.goalBlack, x, y, cellSize, cellSize);
            break;
        }
}
```

Features I'm Proud Of

1. **Path Visualization**
   I added a feature that shows each car's planned route in coloured lines. You can see exactly how the car plans to get to its destination before it even moves!

2. **Path Exist check**

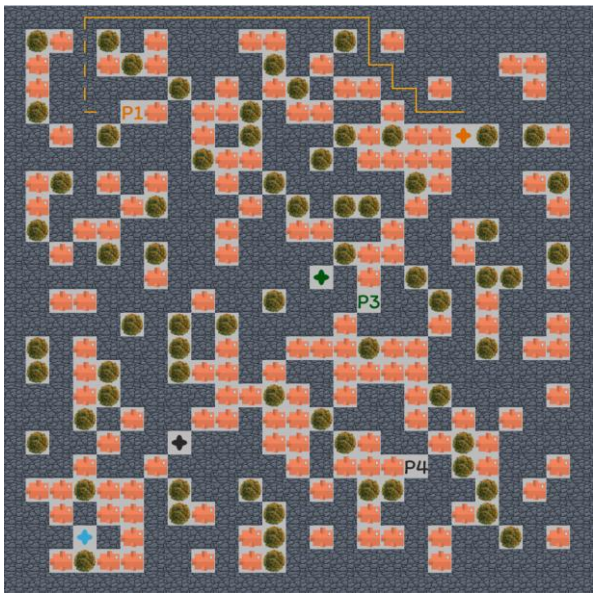
   Even though all the start points and end point for car are randomly assigned on grid. This feature ensures that none of car get destination which is not reachable. It will no fun if the car just sits at one place from the beginning and just act like a wall. However this function can still failed if you increase the obstacles density so high that it's become impossible to adjust 4 cars and destination together with ensure path exist.

3. **Adjustable Grid Size**
   I designed the grid to be flexible, so I can change its size and the number of obstacles easily. Whether it's a tiny town or a sprawling city, the system can handle it, which allowed me to test a range of scenarios.

Visuals (Imagine this with Screenshots)

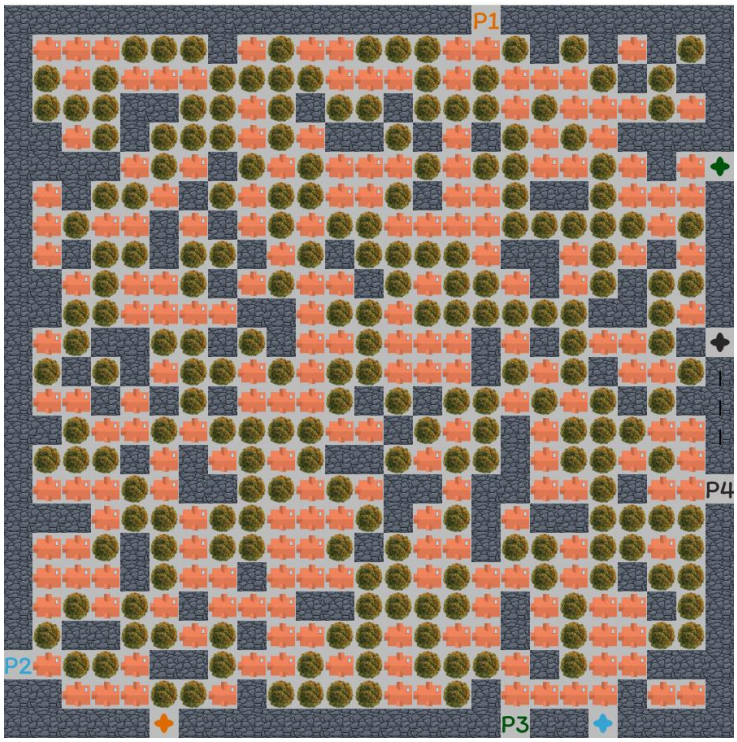
In below figure

1. Star shape figures are car.
2. P1 is orange car destination, P2 is for Blue car, P3 is for green car and P4 is for black car.
3. Green trees and house figure signify walls for this grid layout



Below figure is the situation where PathExist will fail for green car as this pathExits function assign start point and end point when car instance created. In this situation at the time Green car instance created Black car instance was not present hence path exist at that moment for Green car but when black car instance created it become impossible for Green car to reach its

destination as there is only on path exist for it which is now block. Orange car and blue will still reach to its destination. First orange car will pass by P2 destination that where blue car start heading towards its target.



Conclusion

Building this mini city for autonomous cars was a fun and eye-opening experience. I learned how powerful algorithms like A* are for pathfinding and got a real sense of what it takes to avoid obstacles and handle complex grid layouts. The turn-based system was a simple but effective solution for managing multiple cars in a shared space, and the pathExist check kept my cities functional even with lots of obstacles.

If I were to keep building on this, I'd love to add more advanced features like real-time traffic, dynamic obstacles that pop up, or even variable road speeds. Overall, this project taught me a lot about how to make autonomous systems navigate in a complex world—maybe next time, I'll try it with a 3D city!