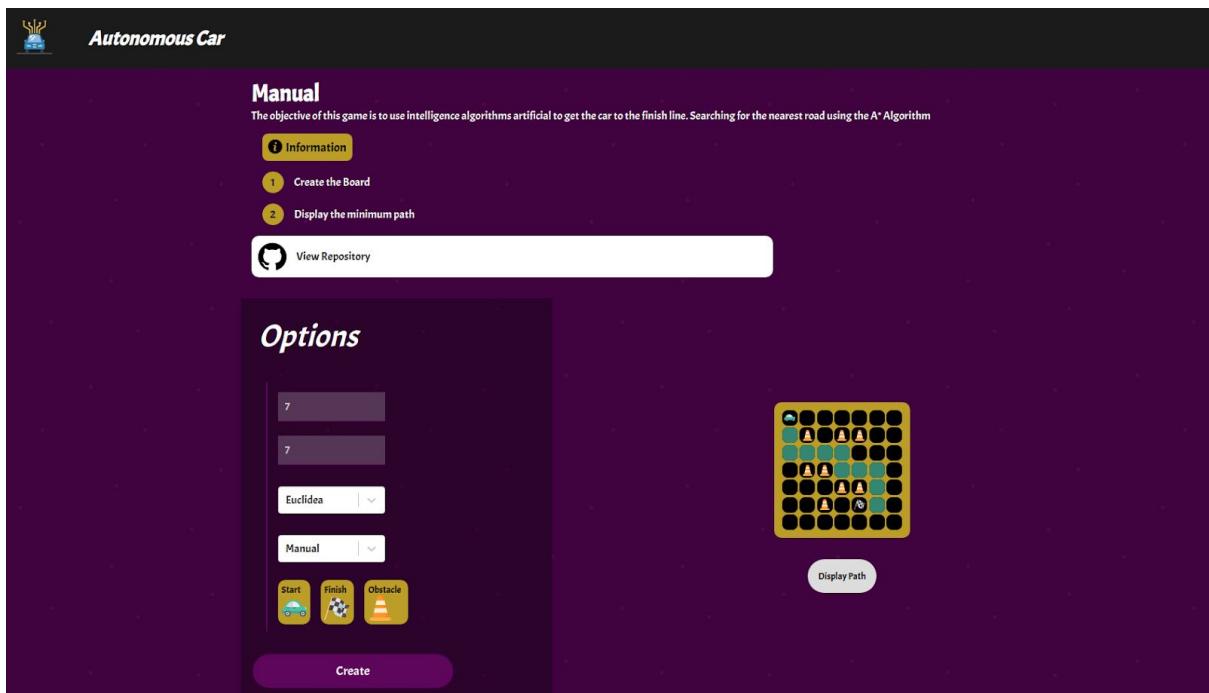


# Informe

# Practica de Busqueda



Universidad de La Laguna  
Grado Ingeniería Informática 3º  
Asignatura - Inteligencia Artificial

Alberto Cruz Luis  
[alu0101217734@ull.edu.es](mailto:alu0101217734@ull.edu.es)

# Índice

- ❖ Introducción
- ❖ Entorno de Simulación y Programación
- ❖ Metodología de Trabajo
- ❖ Algoritmo de Búsqueda
- ❖ Evaluación Experimental del Algoritmo de Búsqueda con dos Funciones Heurísticas
- ❖ Conclusiones
- ❖ Bibliografía

# Introducción

El objetivo de esta práctica es la utilización de estrategias de búsqueda para conseguir llegar del estado inicial al estado final.

Necesitaremos un entorno de simulación para un coche autónomo de dimensiones  $M * N$  en el que dispondremos de 4 tipos de celdas: Inicial, Final, Obstáculo, Vacía.

Cada celda tendrá 4 vecinos (Norte, Sur, Este u Oeste).

En este problema el espacio de estados será cada una de las celdas del tablero.

El estado inicial será la posición de partida del coche.

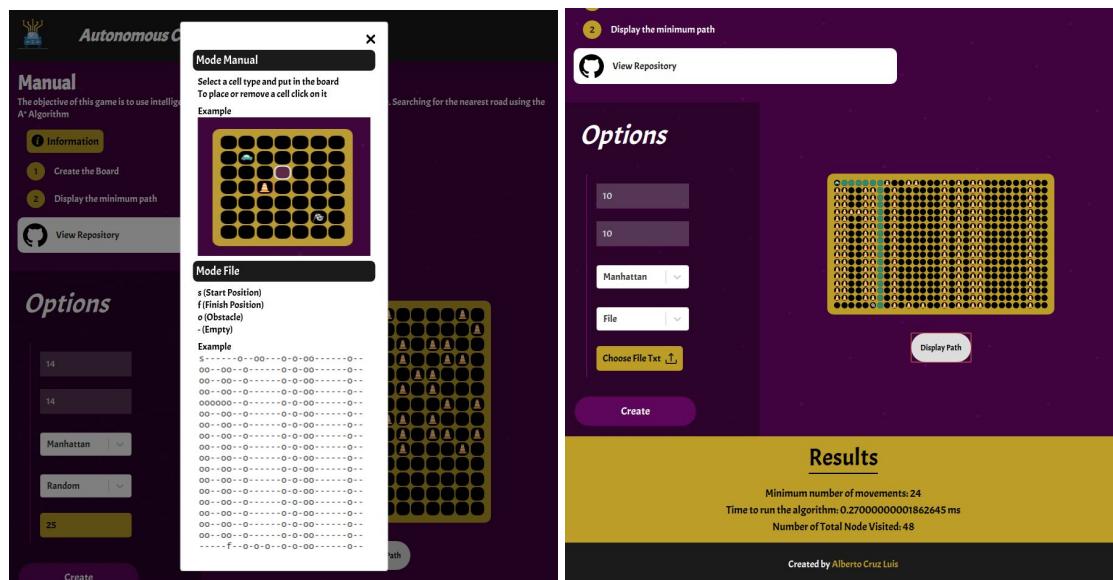
El estado final será la posición de llegada o meta del coche.

Cada movimiento que haga el coche corresponderá a un nuevo estado que sera una nueva disposición en las celdas del tablero.

## Entorno de Simulación y Programación

Mi entorno de simulación es una página web hecha con en JS con el Framework de React.

En ella tenemos un pequeño manual de como funciona, un menú de opciones para poder elegir los campos necesarios para crear el entorno, y el tablero que es el que mostrará la disposición del coche y sus respectivos obstáculos.



Yo he elegido este entorno de programación ya que era la mejor forma de poder mostrar gráficamente el entorno y con ello darle una mejor experiencia al usuario.

# Metodología de Trabajo

El trabajo lo he realizado de forma individual.

Las tareas realizadas han sido la creación del entorno de simulación, la creación del algoritmo de búsqueda y la documentación de la aplicación.

La metodología empleada ha sido la metodología Ágil.

## Algoritmo de Búsqueda

El algoritmo utilizado es el A\* o **A-Star**.

Pseudocódigo

```
AStar(start, finish, type_heuristic) {
    openSet := {start}
    closeSet := {}
    while openSet is not empty
        current := getNodeLessCost in openSet
        if current == finish
            return rebuildPath(current, start)
        if current.totalCost == Infinity
            return map has not solution
        closeSet add current
        openSet remove current
        generateNeighbors(current)
        for each neighbor of current
            if detectObstacle or exist neighbour in closeSet
                skip iteration
            totalCost := current.realCost + 1
            nextNode := neighbour
            if neighbour not in openSet
                nextNode.parent = current
                nextNode.realCost := totalCost
                nextNode.estimatedCost := getHeuristic(type_heuristic)
                openSet add nextNode
            else
                nextNode.realCost := totalCost
                nextNode.parent := current
            nextNode.totalCost := nextNode.realCost +
                nextNode.estimatedCost
    }
}
```

Para la implementación del algoritmo he utilizado las siguientes Estructuras de Datos vector: para las listas abiertas y cerradas.

class Node: para almacenar la información del tipo de celda, los costes y cual es su padre. Con el que enlazaras el camino como si de una lista se tratara.

Las funciones Heurísticas utilizadas han sido 2:

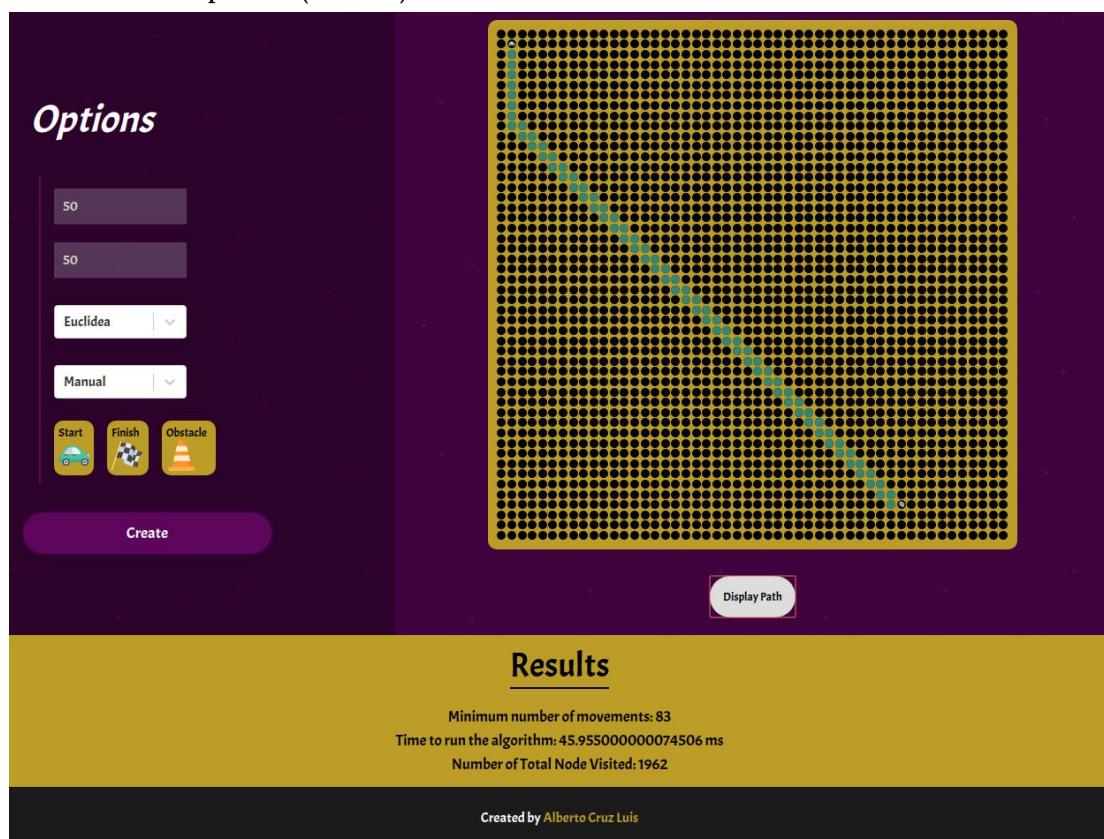
**Euclídea:** Calcula la distancia mínima entre dos puntos en línea recta

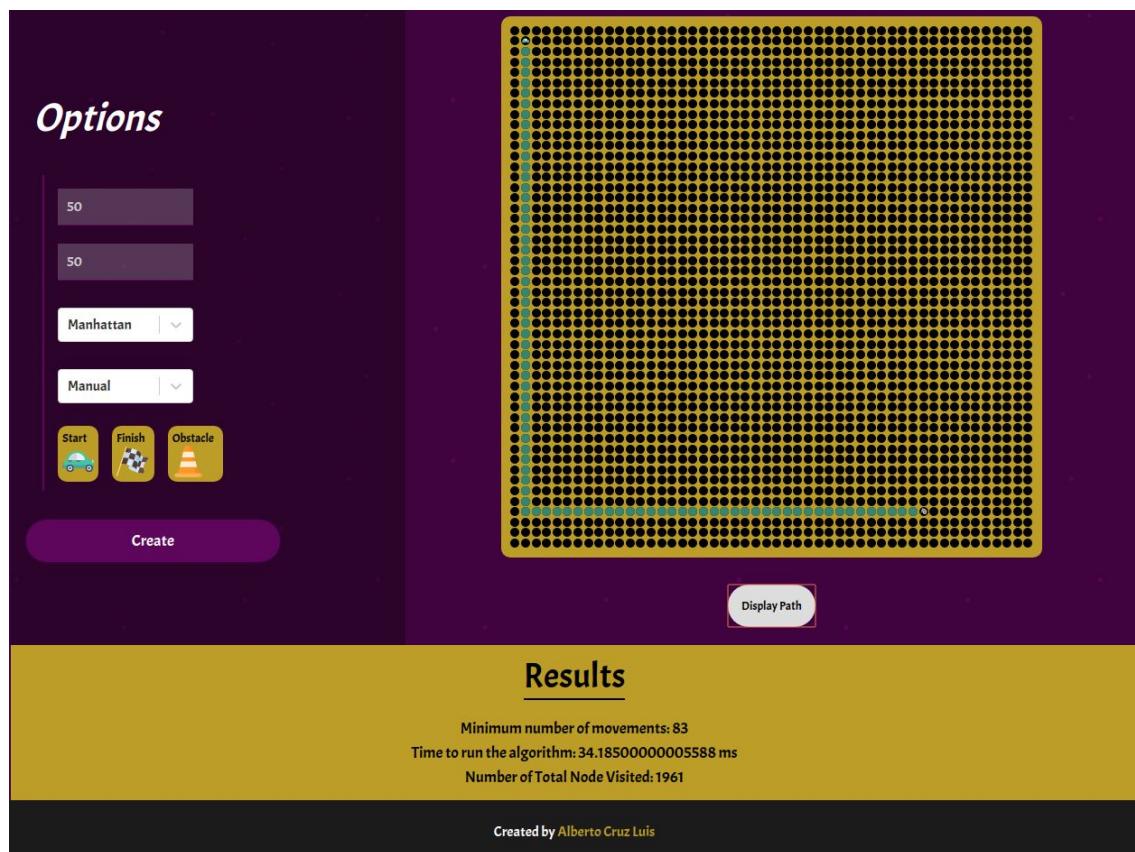
**Manhattan:** Calcula la distancia entre dos puntos como si de una ciudad se tratara.

## Evaluación Experimental del Algoritmo de Búsqueda con dos Funciones Heurísticas

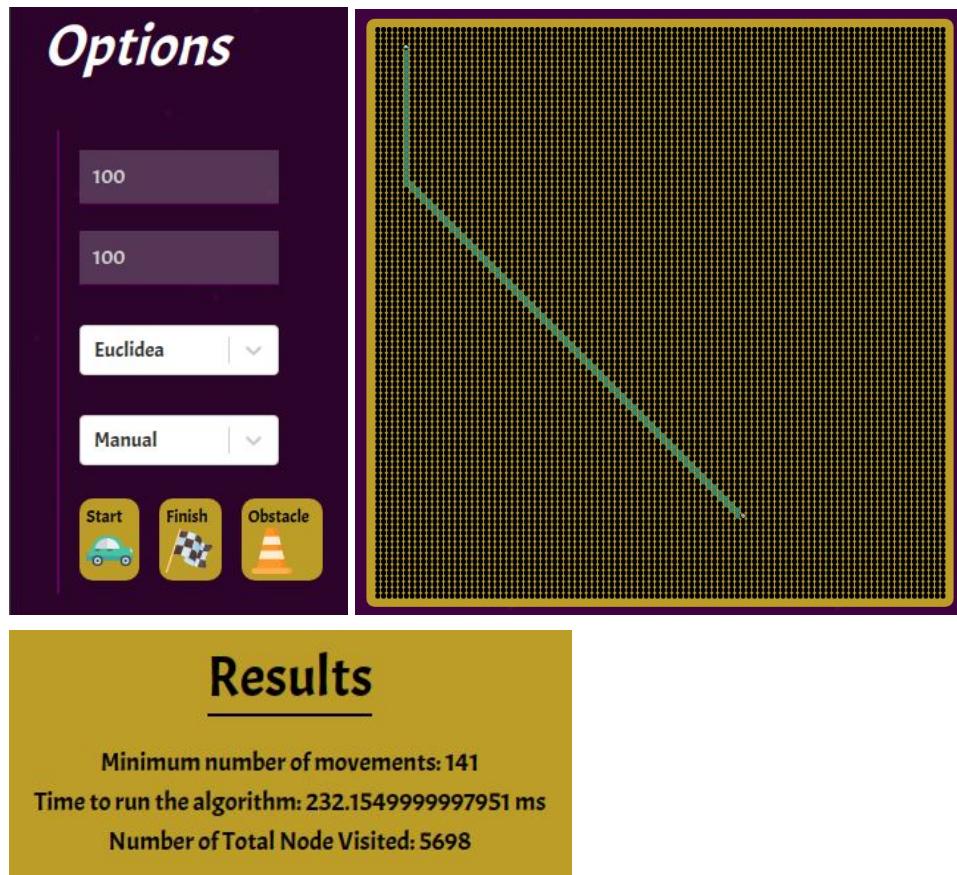
### Sin Obstáculos

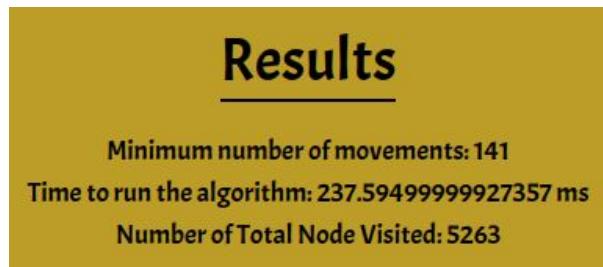
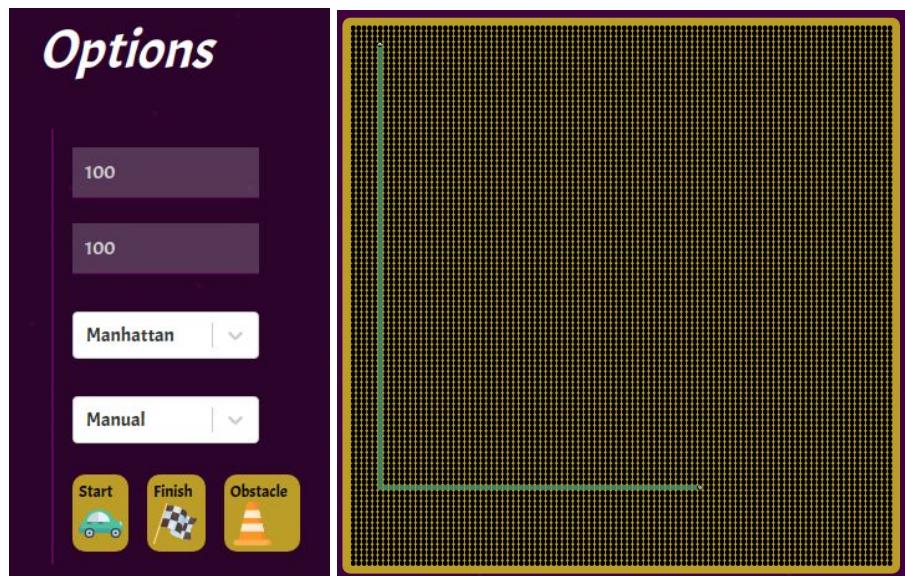
Escenario Pequeño (50\*50)



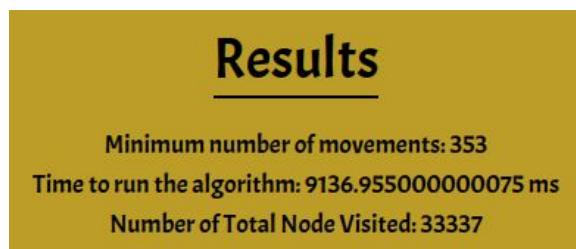
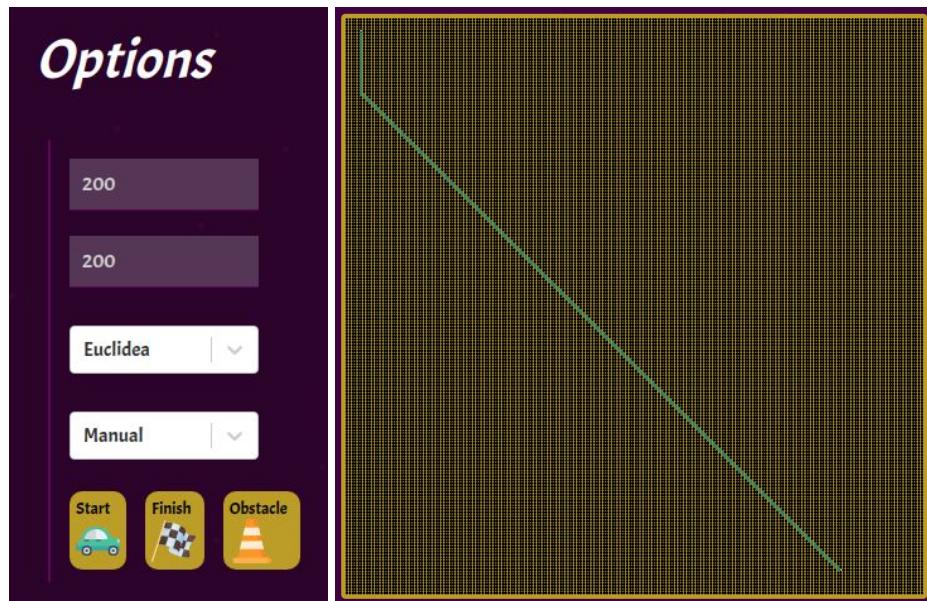


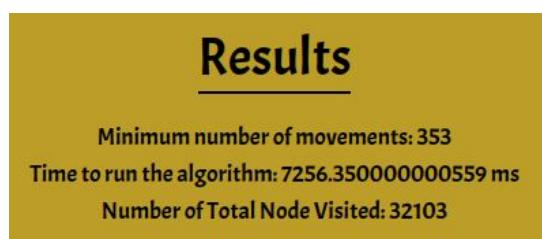
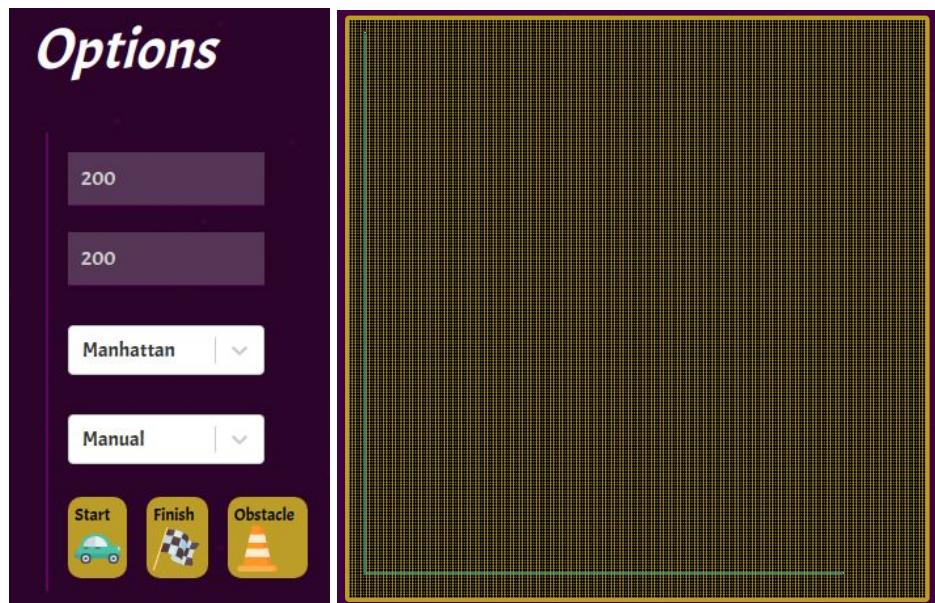
Escenario Mediano (100\*100)



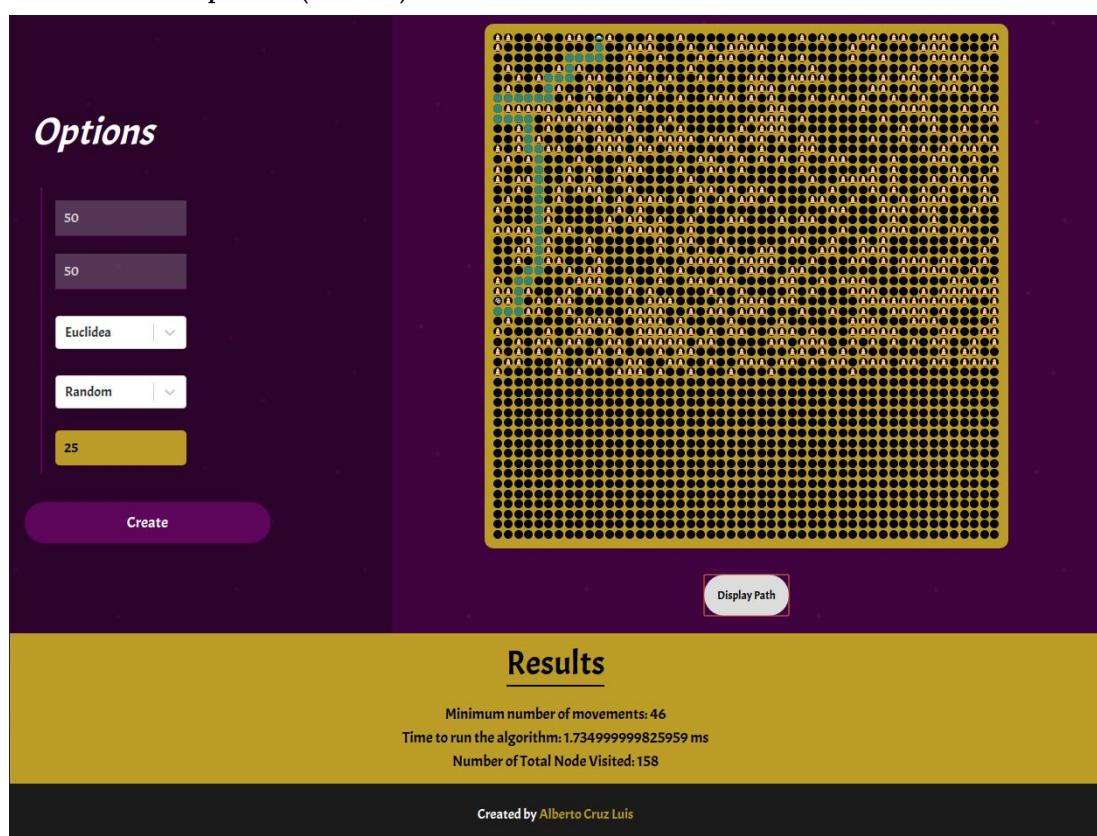


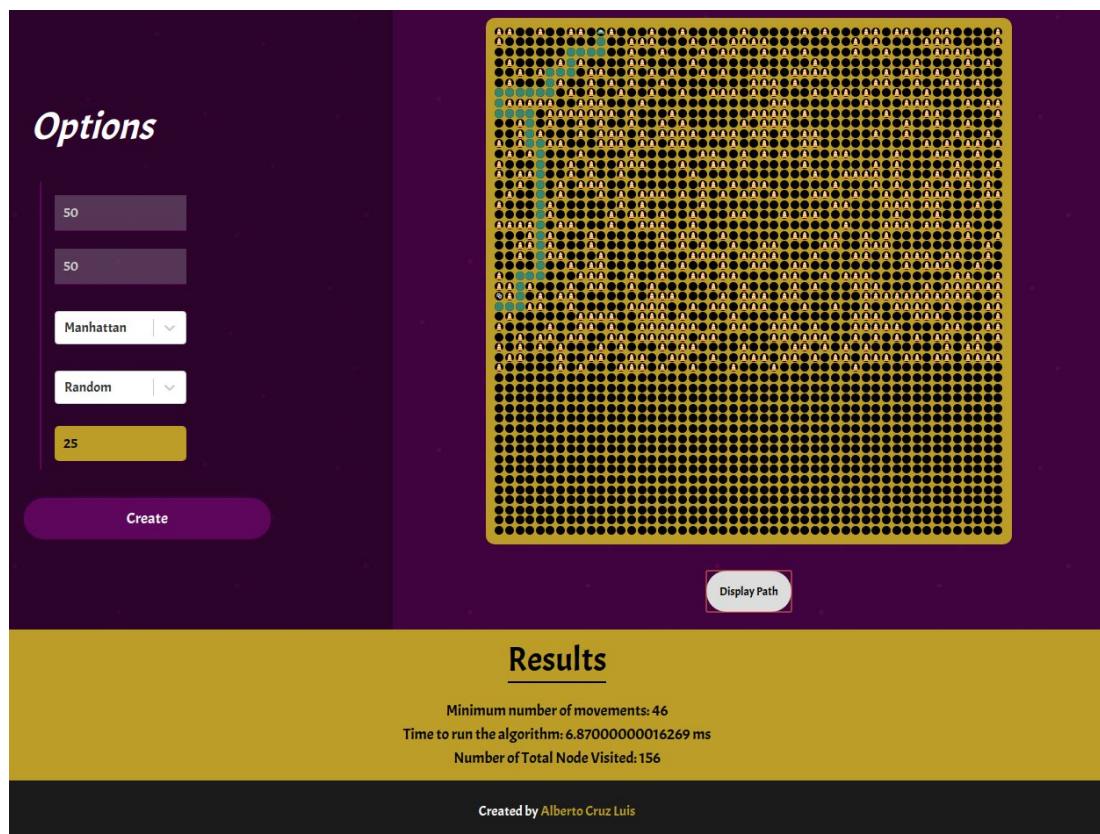
Escenario Grande (200\*200)



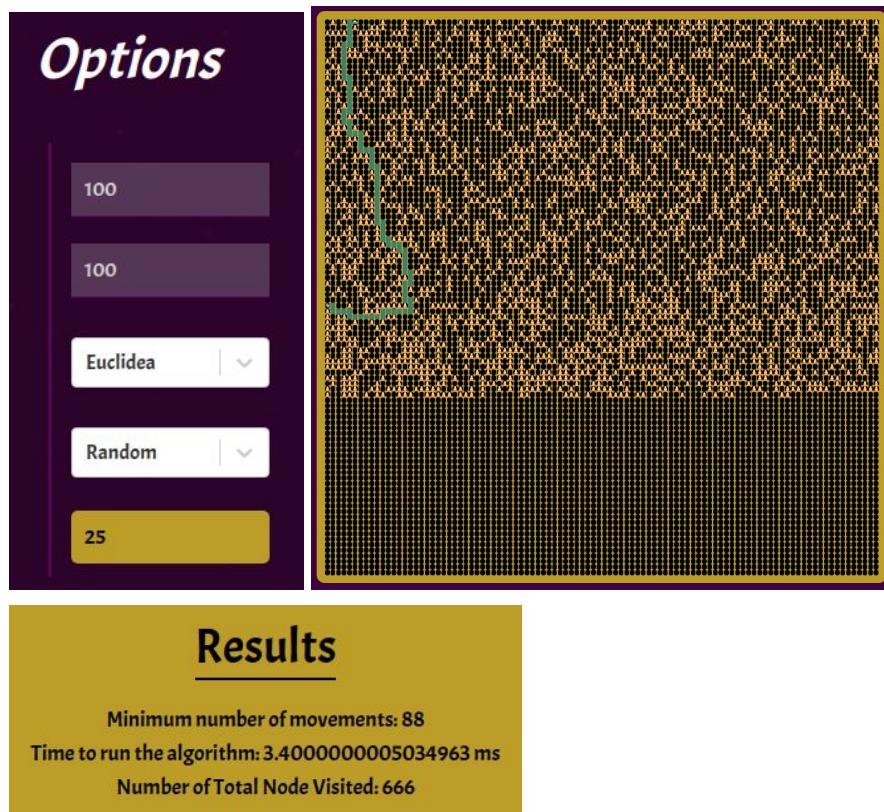


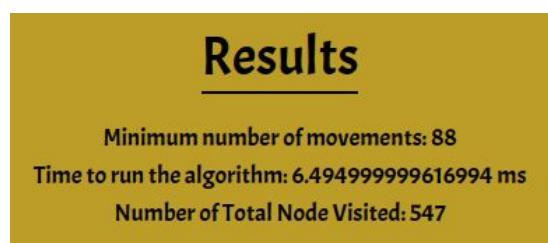
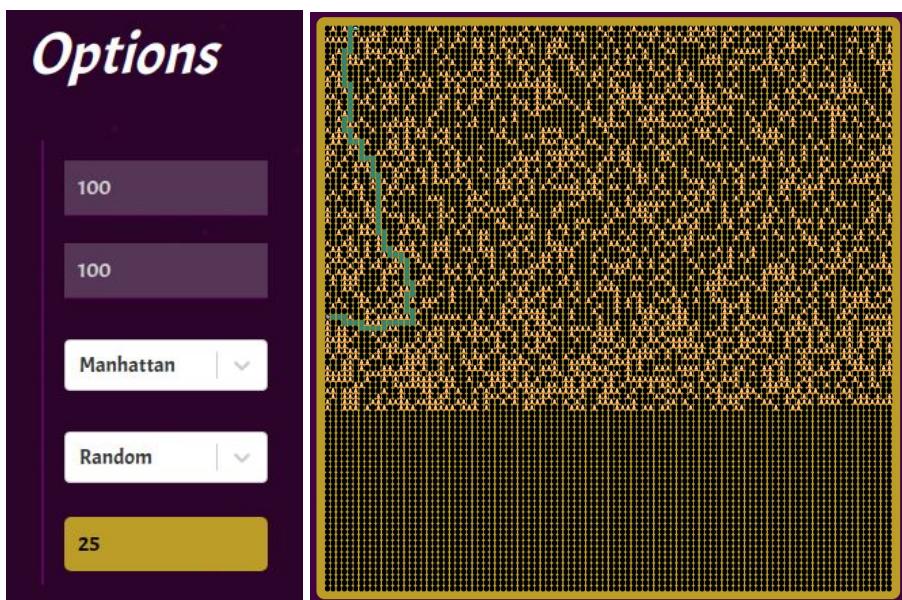
**25% Obstáculos**  
 Escenario Pequeño (50\*50)



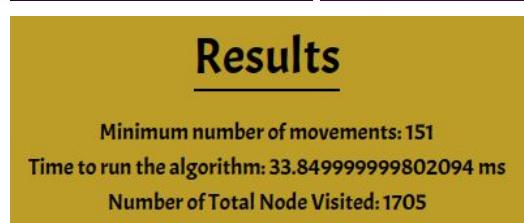
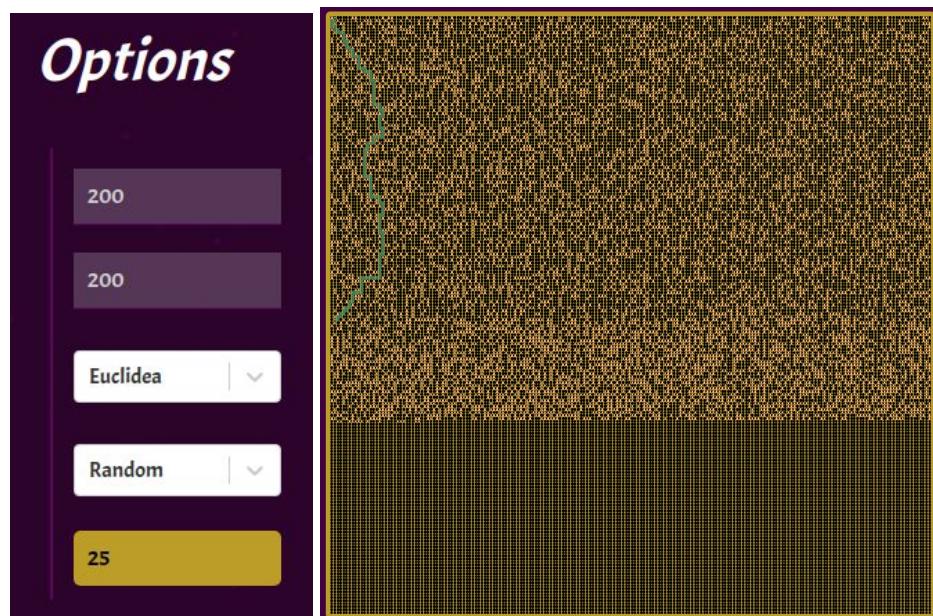


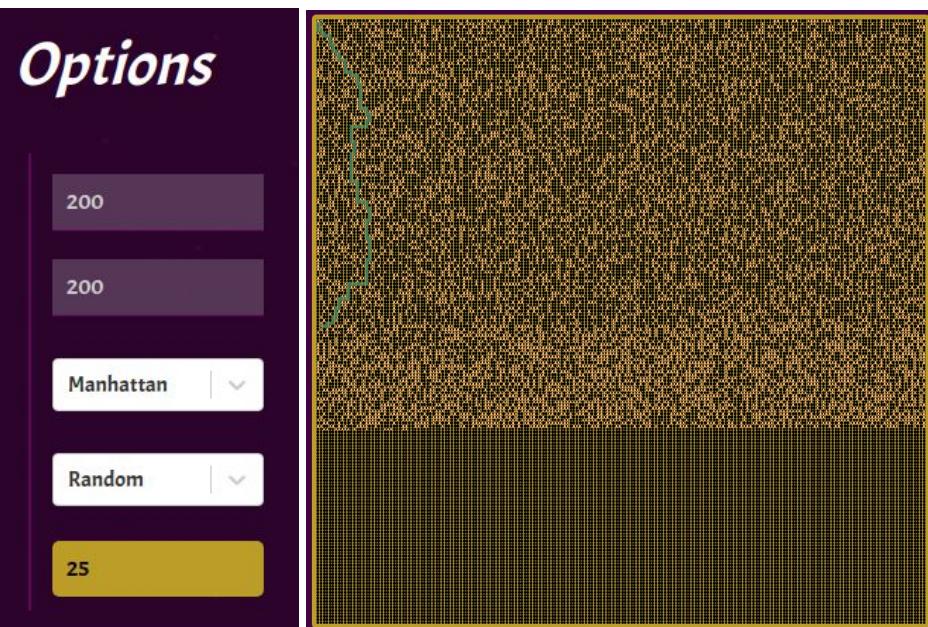
Escenario Mediano (100\*100)





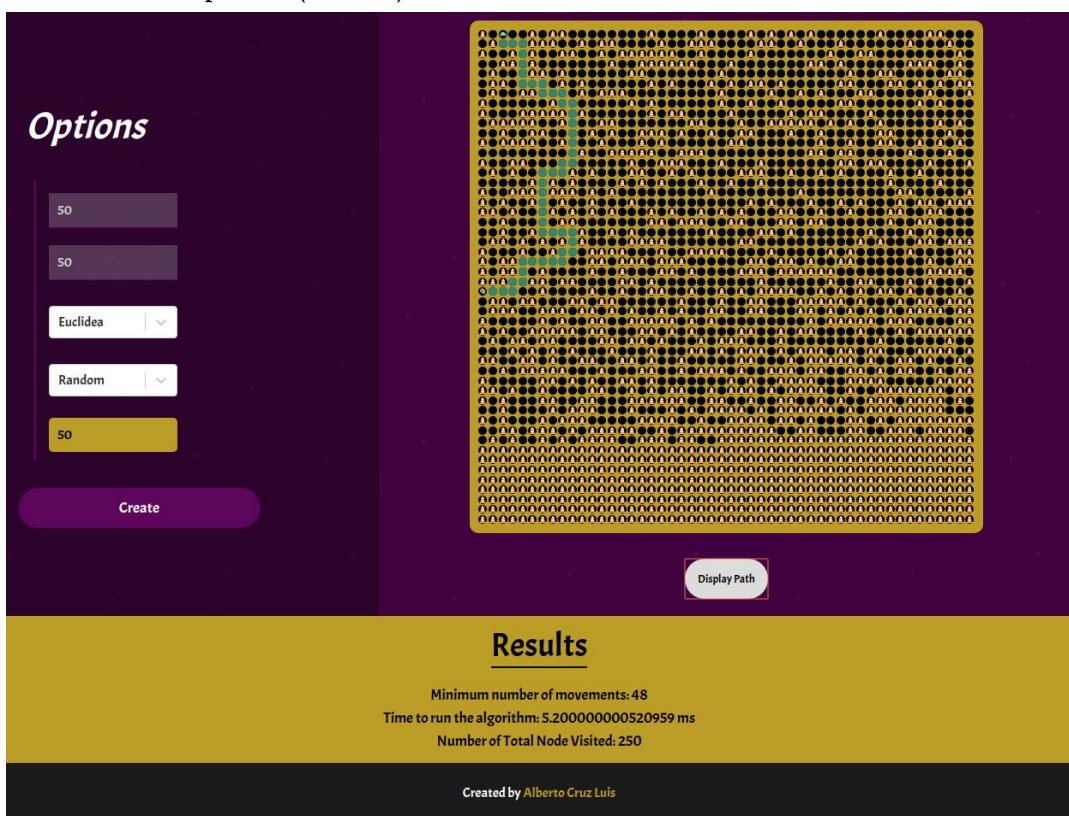
Escenario Grande (200\*200)

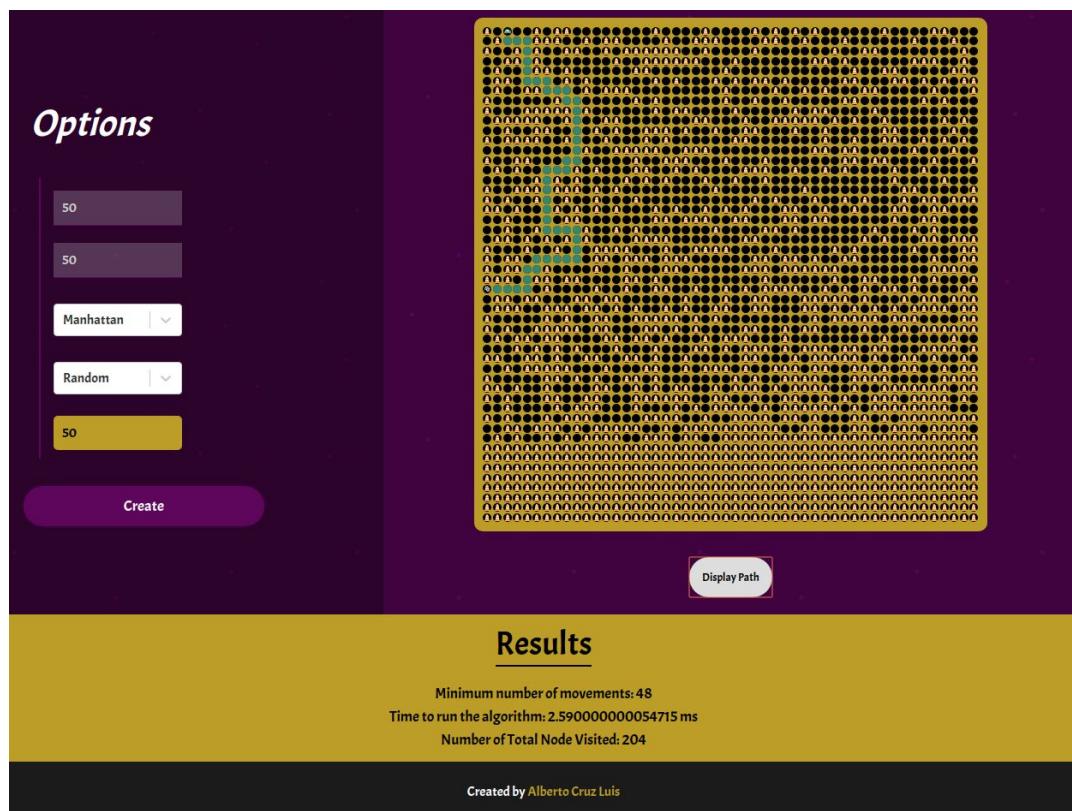




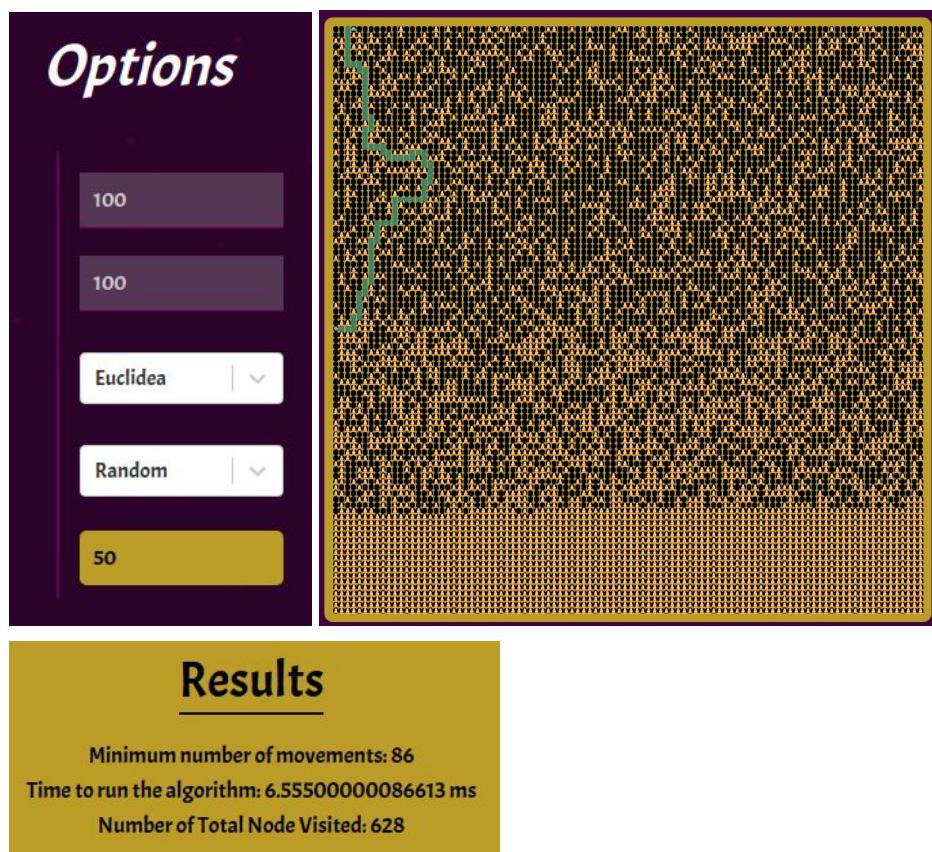
## 50% Obstáculos

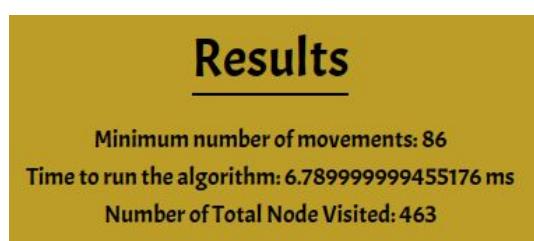
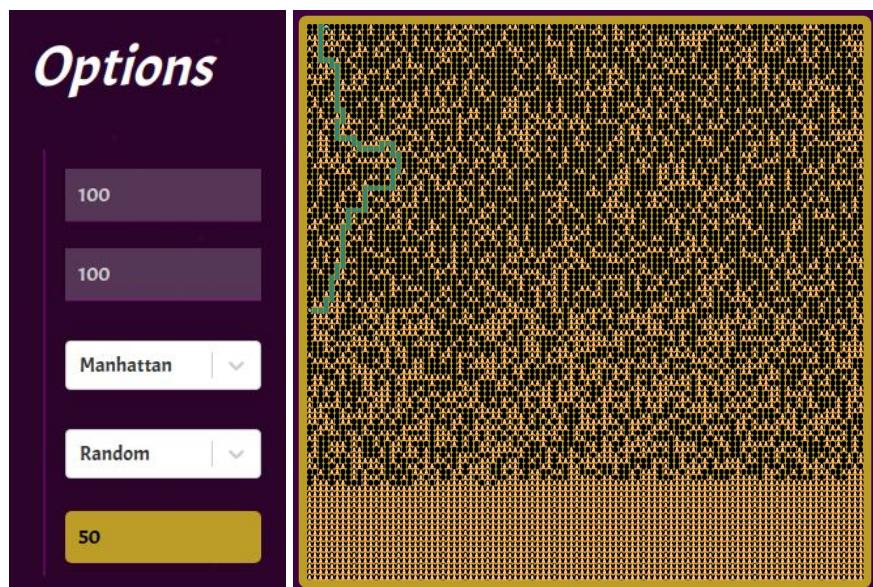
Escenario Pequeño (50\*50)



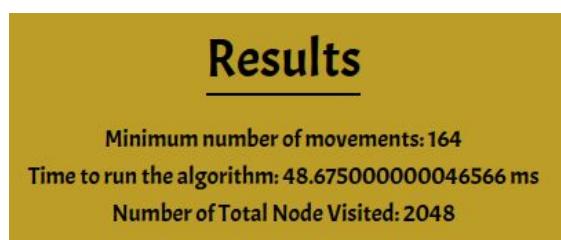


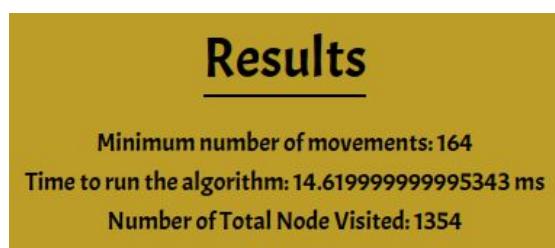
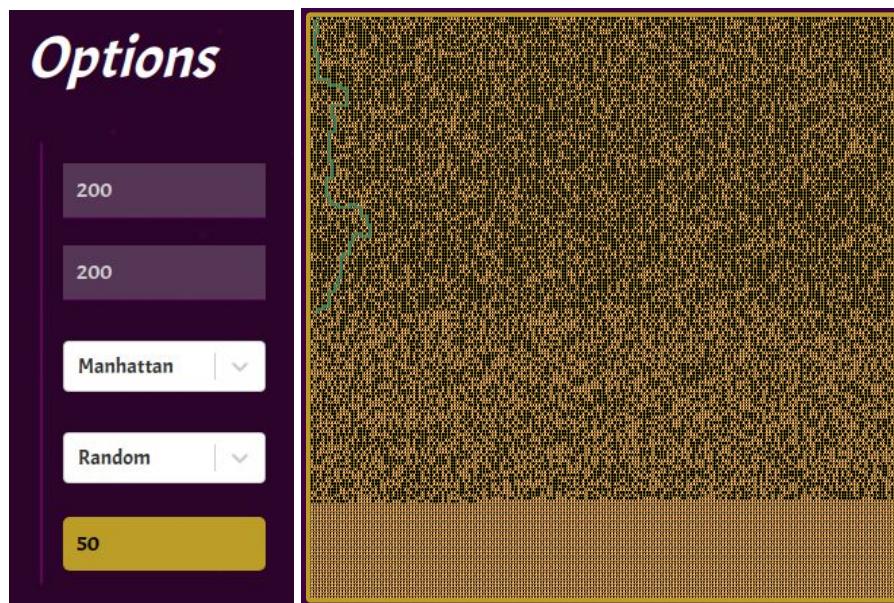
Escenario Mediano (100\*100)





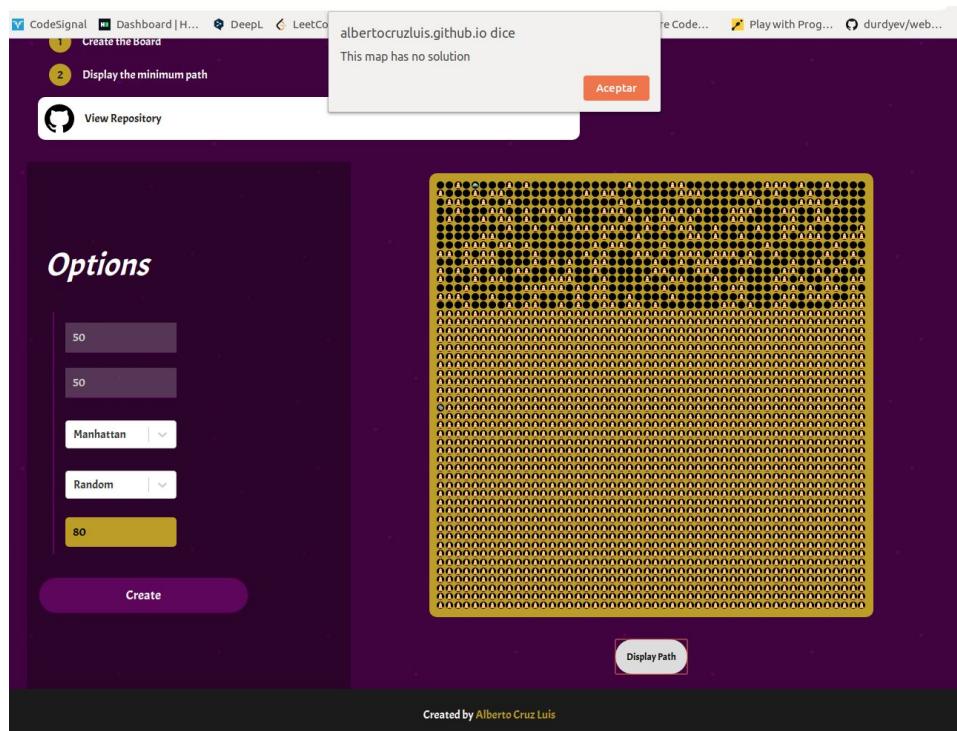
Escenario Grande (200\*200)





## 80% Obstáculos

Con esta cantidad de obstáculos es imposible encontrar un mapa que tenga solución.



## Conclusiones

Tras la realización de los experimentos anteriores he podido comprobar que para este proyecto el algoritmo A\* se comporta mejor con la función heurística (Manhattan) ya que como podemos ver en los resultados anteriores los nodos totales recorridos son siempre menores o iguales que los de la otra función heurística (Euclídea). Por lo tanto eso significa que la función heurística (Manhattan) se adapta mejor a la realidad del problema.

## Bibliografía

[https://en.wikipedia.org/wiki/A\\*\\_search\\_algorithm](https://en.wikipedia.org/wiki/A*_search_algorithm)

<https://www.analyticsvidhya.com/blog/2020/02/4-types-of-distance-metrics-in-machine-learning/>