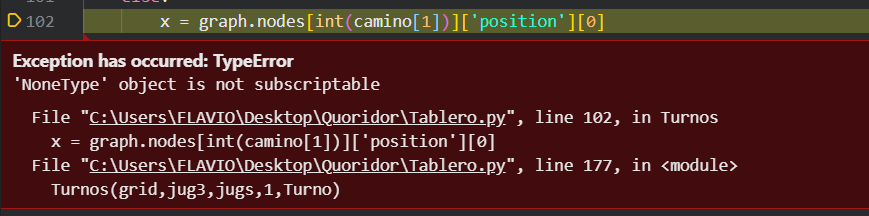
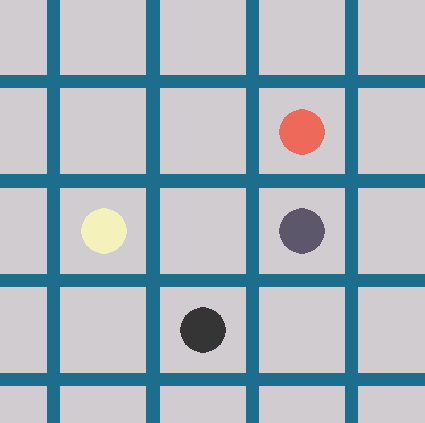
**Experimentos**:

En primer lugar, antes de pasar a la presentación de los experimentos se presentaron diversos problemas relacionados al código del proyecto:

1. En el entorno de trabajo - en nuestro caso, Visual Studio Code - no se ejecutaban la totalidad de funciones de las bibliotecas usadas. Por ejemplo, para la graficación de grafos - propia de ‘networkx’ -, VS Code no podía procesar la función ‘draw’. Debido a esto, se usó Google Collaboratory para realizar ciertas ejecuciones específicas, necesarias para el desarrollo del trabajo.
2. En las primeras versiones del proyecto, se usaba como estructura para los grafos a la clase ‘networkx.DiGraph’ - la cual representaba un grafo dirigido. Sin embargo, se decidió usar la estructura ‘Graph’ ya que trabajamos con grafos no direccionados.
3. En la implementación de los algoritmos, inicialmente se trabajó en base a dos jugadores. Sin embargo, acorde a lo propuesto en el enunciado del trabajo parcial, se tuvo que refactorizar estas funciones para adaptarse a los 4 jugadores del Quoridor. Originalmente, se tenía una función booleana de ‘Turno’ para controlar los movimientos de ambos jugadores. Más adelante, debido a la existencia de 4 jugadores al mismo tiempo, se cambió el tipo de la función a tipo Entero.
4. Se tuvo que adaptar los algoritmos de PathFinding para que invaliden las posiciones actuales de los jugadores debido a que la colisión de los mismos generaba conflictos como NoneType.
5. El algoritmos DFS causaba muchos problemas con los grafos generados ya que al haber restricciones con los nodos debido a los jugadores se generan bucles con los Paths.
6. Se tuvo que realizar múltiples intentos en la generación del tablero ya que debía existir espacio necesario para la casilla, para la futura agregación de paredes y para colocar las fichas en el espacio correcto.

Ahora bien, se realizaron los siguientes experimentos:

* Sin validar la existencia del Nodo al que se quiere mover el jugador



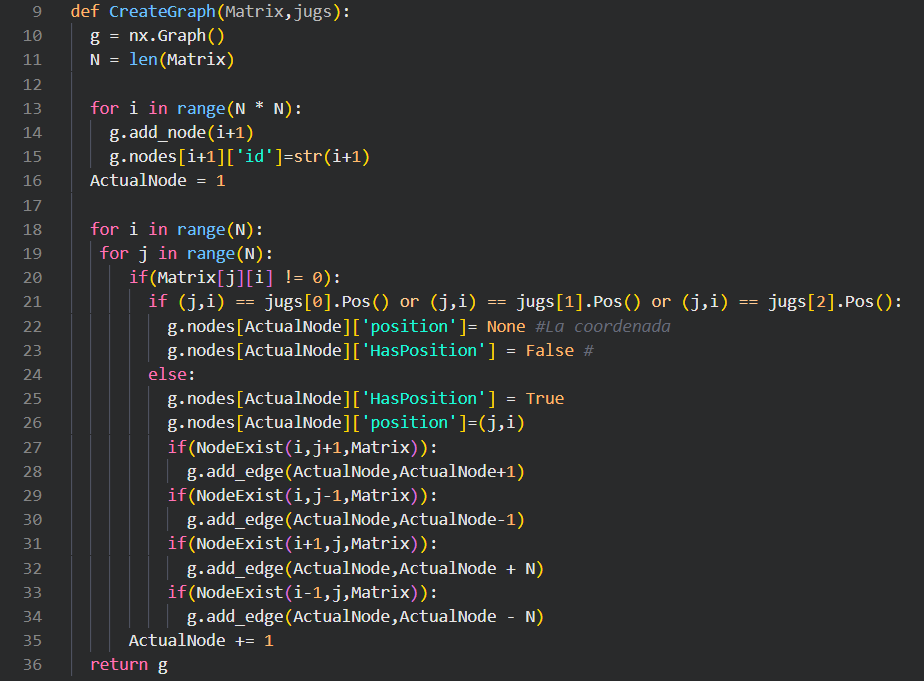
Al realizar la validación el jugador tendrá que tomar otro camino para llegar a su destino



* Se tuvieron que crear dos funciones hallar\_camino debido a que DFS, al ser una búsqueda en profundidad al primer nodo (1) no se le asignaba un padre , incluso cuando el camino más cercano tendría que pasar por ese nodo, pero como no tiene padre, generaba un error al momento de intentar seguir un camino o a veces creaba un bucle. Por eso tuvimos que dividir el pathfinding para validar algunos nodos.



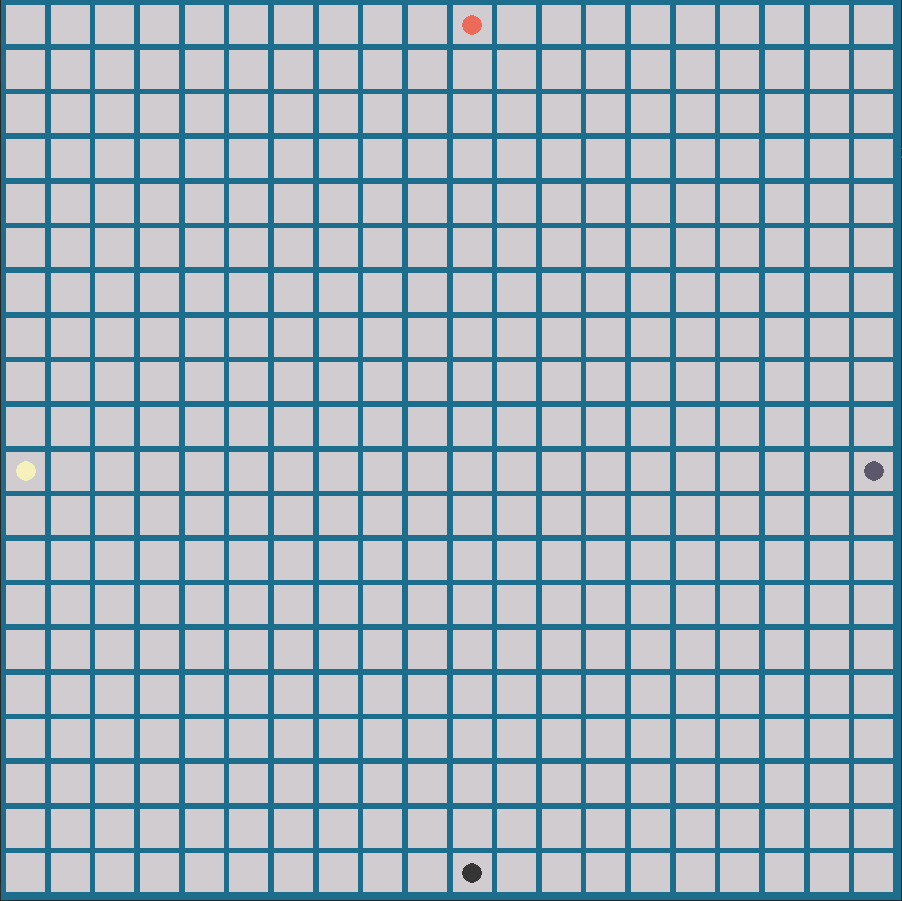
* Por otra parte también se tuvo que crear dos funciones de CreateGraph debido a que el jugador 2 empezaba en la parte inferior del tablero por lo que el grafo debía crearse de manera invertida.

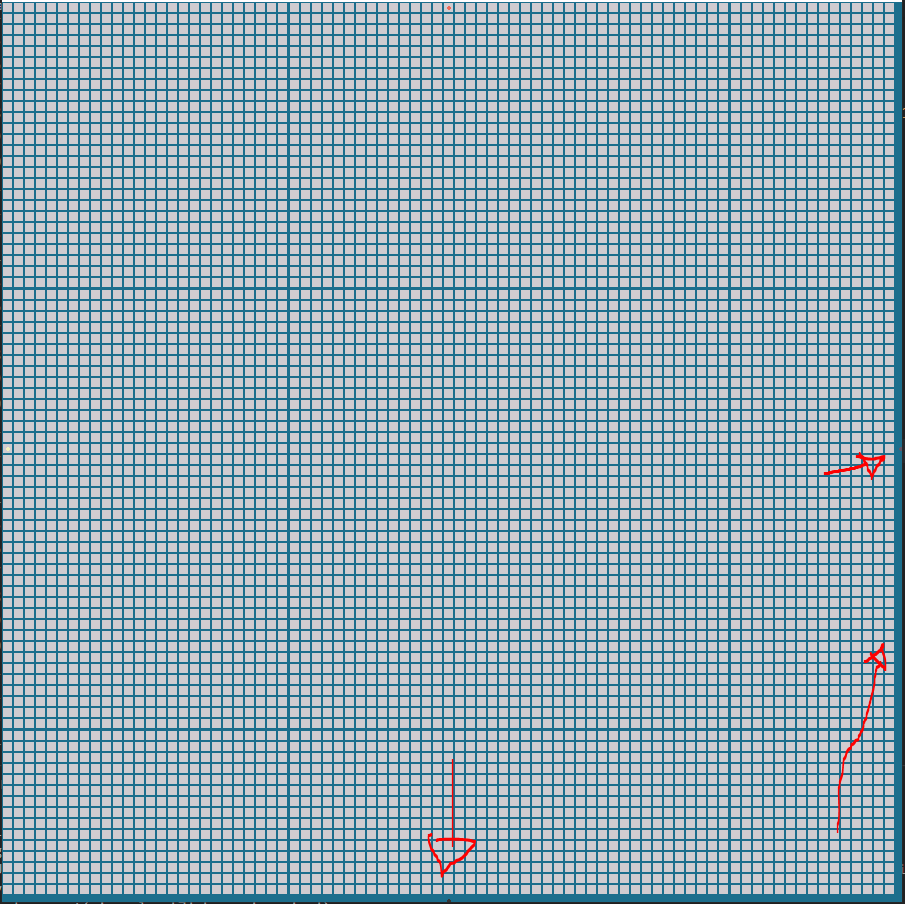


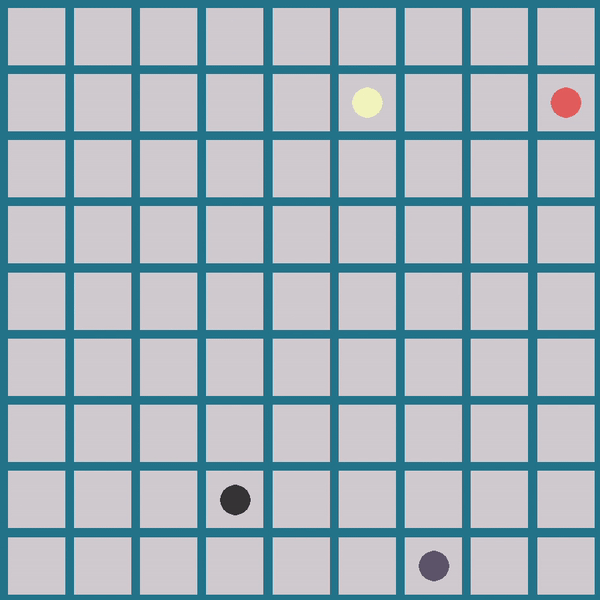


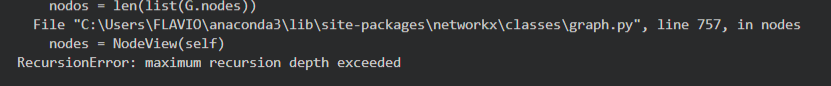
* Al realizar tableros de gran tamaño, se dificulta el cálculo del espacio necesario para los jugadores por lo que en ciertos valores de N no se grafica de buena forma(a partir de 72 se generan estos errores de graficación, por lo que haría falta crear una mejor fórmula del cálculo de espacio).

Ejemplo de buena graficación N=20:



Ejemplo de mala graficación N= 81:

* Se generaban bucles pequeños al realizar DFS
* Otro de los muchos problemas que nos causó DFS fue el límite de recursividad en python

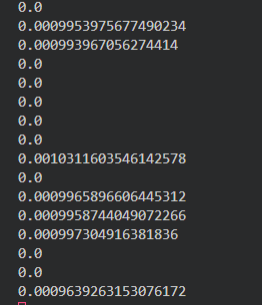


por lo que se requirió agregar la siguiente linea:

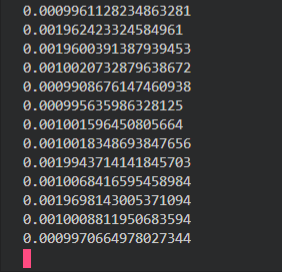


**Resultados**:

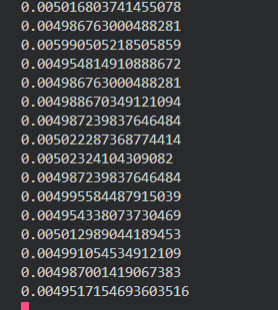
* Tiempos promedio de DFS
  + en N=9:



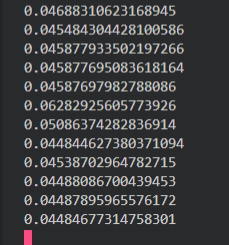
* + En N = 15



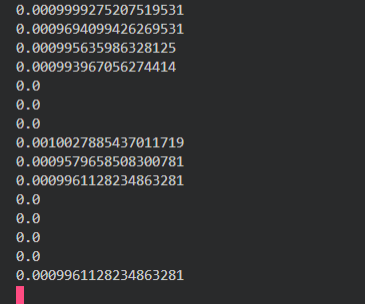
* + en N = 25



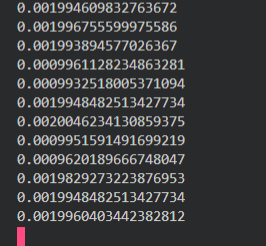
* + en N = 50



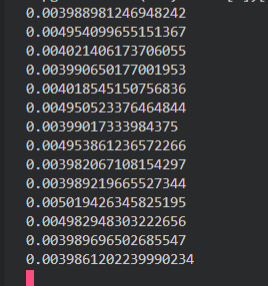
* Tiempos promedio de BFS
  + en N=9:



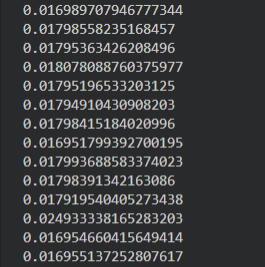
* + en N =15:



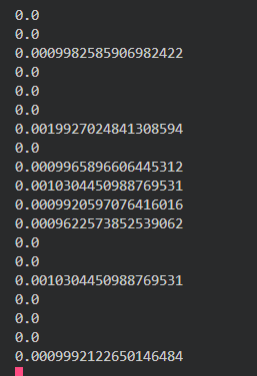
* + en N = 25:



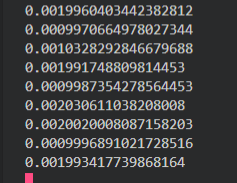
* + en N = 50:



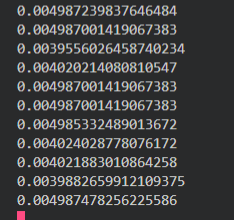
* Tiempos promedio de Dijkstra:
  + en N = 9:



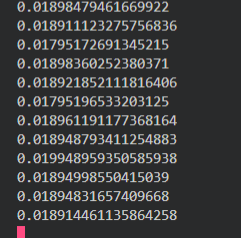
* + en N = 15:



* + en N = 25:



* + en N = 50



Conclusiones:

//sobre los resultados

El Pathfinding es un tema muy interesante y con amplias aplicaciones. Claro ejemplo de esto es la presente investigación y proyecto presentados, ya que, tomando como contexto base el juego Quoridor, se profundiza en la búsqueda de algoritmos eficientes y completos para lograr llegar a una meta.

Con respecto a los resultados, se logra observar que algunos de los algoritmos elegidos presentan diferentes características. En el caso de DFS, puede ser más rápido pero el camino elegido es el más largo en la mayoría de los casos. Mientras que con BFS y Dijkstra puede demorar un poco más pero el camino mostrado casi siempre es el más corto.

Finalmente, cabe resaltar que el trabajo presentado formará base para el entregable del trabajo final del mismo curso. Debido a la organización del proyecto, será posible reutilizar código funcional en las futuras entregas. Sin embargo, en caso de ser necesario,se modificará parte - o todo - de código para el logro del objetivo especificado.