Quoridor Research Paper

Juan Carlos Hernández, Gino Quispe, Rodrigo Ramirez

Octubre 2020

1 Introducción

Quoridor es un juego de mesa que se puede jugar de entre 2 a 4 jugadores en un tablero similar al de ajedrez o damas. Este juego fue creado en el año 1997 en Francia por Mirko Marchesi luego de 2 años de haberlo desarrollado. El juego obtuvo mucha fama al ganar el premio Mensa Mind Game el mismo año de su lanzamiento.

Figure 1: Quoridor Board

El tablero de Quoridor a diferencia del de ajedrez o damas consta de una cuadrícula de 9 x 9 (81 casillas en total). Cada jugador controla un peón y debe lograr que este cruce al lado opuesto para ganar. Al inicio de la partida los peones son colocados al centro de la fila inicial y son posicionados en frente del oponente. Además, cada jugador cuenta con una cantidad determinada de

bloques que pueden usar para obstruir el camino del oponente con finalidad de que tome más turnos para llegar a su meta, esto siempre en cuando no se encierre al oponente, ya que no es válido dejar al oponente sin un camino libre para llegar a su meta.

Respecto a la cantidad de bloques que posee un jugador, el total de 20 bloques son repartidos equitativamente entre la cantidad de jugadores, es decir, si hay 2 jugadores cada uno tiene la cantidad de 10 bloques o si hay 4 jugadores cada uno tendrá 5 bloques. Cabe recalcar que el grosor de un bloque obstruye 2 casillas del tablero.

En el presente trabajo se desarrollará el juego anteriormente mencionado usando los conocimientos del curso de Complejidad Algorítmica, usando los enfoques y paradigmas de programación y tomando en cuenta la importancia de los algoritmos.

2 Estado del Arte

Como ha sido mencionado anteriormente, el objetivo de Quoridor, es que un jugador llegue al otro extremo del que empezó, por lo que en cada jugada, se debe buscar el camino más óptimo para ir, cumpliendo con las restricciones del juego, ya sea no pasar por encima de paredes o solo realizar una jugada por turno. Actualmente existen distintos algoritmos o técnicas relacionados a la búsqueda de caminos, y en esta sección haremos mención de alguno de ellos que probablemente sean utilizados en el desarrollo de nuestro proyecto.

- **Algoritmo de Dijkstra**: Este algoritmo desarrollado por Edsger Dijkstra en 1959, consiste en encontrar el camino más corto, desde un vértice a otros pertenecientes a un grafo. Si bien este algoritmo trabaja con pesos en las aristas, este puede ser utilizado con aristas de peso 1 para encontrar el camino más corto hacia el destino. Uno de los usos que se le ha dado a este algoritmo es en el campo de la telemática, en la que el algoritmo puede ser utilizado en grafos de gran cantidad de nodos de manera eficiente.
- **Algoritmo de búsqueda A***: Este algoritmo, realiza el problema de encontrar el camino más corto, con la ayuda de una heurística para agilizar la búsqueda. EL algoritmo en el mejor de los casos, puede resolver el problema de forma lineal. Sin embargo, cuando se encuentra en el peor escenario, la búsqueda se realiza en tiempo exponencial.

- **Algoritmo de Floyd-Warshall**: Este algoritmo fue desarrollado por Bernard Roy en 1959 y se basa en encontrar el camino más corto que existe entre los nodos de un grafo dirigido ponderado. Este algoritmo usa la técnica de Programación Dinámica, en la cual se busca reducir el tiempo de ejecución de un algoritmo mediante la utilización de subproblemas superpuestos y subestructuras óptimas.

Si bien existen algoritmos que resuelven el problema, en el curso se han visto otras técnicas que podremos utilizar ya sea: *Backtracking, Búsqueda por anchura y por profundidad en grafos, Divide y vencerás*, entre otros que nos ayudarán a tener una mayor visión de caminos para el correcto funcionamiento del juego.

3 Metodología

Con respecto a la metodología utilizada para resolver el problema y validar los resultados, se ha empezado con el planteamiento del problema: ***¿Cómo hacer un bot que pueda jugar Quoridor?*** Posteriormente nos planteamos: ***¿Qué algoritmos debemos utilizar para que el bot funcione de forma correcta?***

Con estas dos preguntas iniciales empezamos una investigación con respecto a algoritmos de grafos, algoritmos para hallar el camino más corto y algoritmos de backtracking para resolver este tipo de problemas. Con esta información se procedió a escoger tres algoritmos para la resolución de problema y se dividió el trabajo de tal forma que cada integrante investigue, implemente y valide una resolución al problema planteado por el presente trabajo.

Posteriormente, después de haber investigado sobre el funcionamiento correcto de cada algoritmo, se procedió a integrar al proyecto. Debido a que somos tres estudiantes, se utilizó la herramienta de control de versiones: Git, para tener un seguimiento de las partes que avanza cada integrante y poder retornar a alguna versión anterior en caso se haya modificado alguna parte del código que anteriormente funcionaba de forma correcta.

Finalmente, después de haber implementado los tres algoritmos a la rama principal del proyecto, se procedió a probarlo. Para esta parte se utilizaron una serie de pruebas unitarias, para validar el comportamiento de cada función del problema utilizando pequeños inputs de datos. Gracias a las pruebas unitarias se pudo comprobar el correcto funcionamiento del proyecto implementado.

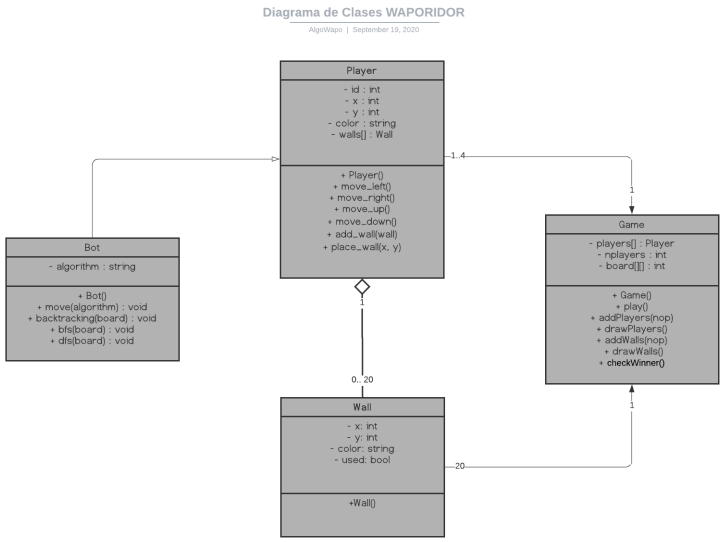

Figure 2: Class Diagram

4 Experimentos

Espacio de búsqueda: El espacio de búsqueda, es el conjunto de todas las posibles soluciones candidatas a un problema. Para el presente trabajo, debido a que el problema principal es encontrar un camino válido que le bot pueda recorrer, el espacio de búsqueda sería: Un arreglo o lista que contenga un camino que se encuentre dentro de la matriz o la lista de adyacencia, que vaya desde una pared inicial, dependiendo del jugador, hasta la pared opuesta y tenga en cuenta las paredes que se encuentran en medio del tablero, ya que estas alteran el espacio de búsqueda.

Declaración de entradas de datos: Para el presente proyecto se han hecho uso de 3 diferentes algoritmos para la búsqueda de una solución al problema del juego Quoridor, en cada uno de estos, se han tenido que realizar distintas pruebas para validar la eficacia y correcto funcionamiento de los mismos. Es por ello que los datos de entrada de los algoritmos van a depender del algoritmo, en el caso de Backtracking, una matriz en la que se representa los caminos que el jugador puede recorrer, mientras que en los relacionados a grafos (bfs, y dfs) se ha procedido a convertir la matriz del tablero a una representación con nodos para una mayor facilidad al momento de la implementación. Además de los datos de los algoritmos, para la jugabilidad se necesitan el número de jugadores

y el número de paredes por jugador que van a ser representados en arreglos de sus respectivas clases (Player y Wall). Estos datos y demás funciones adicionales al funcionamiento, permitieron que además de encontrar solución al problema, se pueda ver de forma gráfica los pasos y jugadas que se realicen.

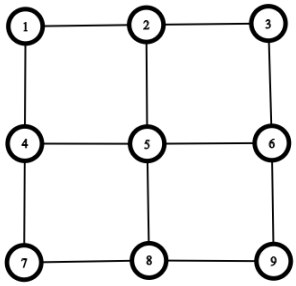

Figure 3: Graph Representation of the Quoridor Board (3x3)

Declaración de métricas para algoritmos: Respecto a las métricas para la implementación de los 3 algoritmos realizados en el presente trabajo, estas se mencionan a continuación por cada algoritmo trabajado.

1. **Backtracking:** En el caso de Backtracking, este tiene una complejidad: O(n2). La forma en la que se ha implementado este algoritmo es utilizando una matriz que en este caso representa al tablero de quoridor. En dicha matriz un jugador debe encontrar el camino más corto para llegar a ganar. Lo que hace el algoritmo backtracking es encontrar dicho camino mediante recursividad, ya que va a almacenar las coordenadas que el jugador debe recorrer en una lista de tuplas para luego poder ser leída por el jugador y saber que camino tomar para llegar a la meta, sin chocar con paredes.

2. **BFS:** La complejidad que tiene el algoritmo que usa BFS es de: O(e+v). Una característica relevante del algoritmo BFS es que no solo encuentra un camino hacia el destino, sino que este camino es el más corto al hacer el recorrido en anchura. El primer paso que se ha realizado es determinar el jugador que va a utilizar el algoritmo, pues cada uno de estos, va a tener un destino distinto a cada lado del tablero. En segundo lugar, le estamos dando un uso distinto al la cola (queue) para que no solo almacene el nodo que visita, sino el conjunto de nodos o camino que se ha ido tomando, esto para que cuando llegue al nodo solicitado en el lado correspondiente del tablero, retorne el camino completo sin la necesidad de utilizar el utilizado en clase arreglo de padres para encontrar el camino, lo que mejora levemente la complejidad del algoritmo.

3. **DFS:** Para el caso de DFS, este algoritmo tiene una complejidad: O(e+v), donde e es la cantidad de aristas y v la cantidad de vertices (edges and vertices). La forma en la cual se ha implementado el algoritmo dfs para el presente trabajo ha sido utilizando una lista de adyacencia que represente el tablero de quoridor, es decir, cada nodo se conecta con 2, 3 o 4 nodos adyacentes según la posición del tablero (se validan esquinas y bordes). En una implementación regular, el algoritmo dfs no encuentra el camino más corto necesariamente, si no, cualquier camino válido desde un punto de inicio hasta un punto de salida. En nuestra implementación, utilizamos dfs para retornar una lista de movimientos que debe realizar el bot para llegar desde un extremo hasta el otro.

5 Resultados

Gracias a la implementación de los algoritmos de pathfinding, uno de los resultados principales son los caminos que debe utilizar el bot para llegar de un extremo al otro.

En el caso de backtracking, debido a que se utiliza una matriz para representar el tablero de Quoridor, el resultado que retorna este algoritmo, es una lista de tuplas, que contiene las coordenadas por las cuales se debe mover el bot.

En contraste, para los algoritmos de dfs y bfs, debido a que se utiliza una lista de adyacencia para representar el tablero, en lugar de una matriz como se realiza en el algoritmo de backtracking, el resultado que retornan estos dos algoritmos es una lista de enteros, que contiene los nodos que debe recorrer el bot para llegar de un extremo al otro.

6 Conclusiones

Una de las conclusiones más relevantes que podemos rescatar de este trabajo es la distinta forma que tienen los algoritmos, que se ve reflejada en cómo obtienen los resultados, y en algunos casos, altera el resultado en sí. Aspectos como, la eficiencia del algoritmo, el espacio que utilizan y la representación de datos que requieren para funcionar, lista o matriz, hace que cada algoritmo sea único y que sea más conveniente utilizar alguno por sobre otro dependiendo del contexto.

Finalmente, una de las conclusiones principales encontradas en el desarrollo del presente trabajo es la utilidad del curso de Complejidad Algorítmica, ya que con los conocimientos adquiridos en este curso podemos determinar qué algoritmos son más útiles que otros y saber en qué contexto usarlos de una forma adecuada.

7 Referencias

1. *GitHub Repository*
2. *Class Diagram*
3. *A. Chumbley. Shortest Path Algorithms. 2019*