**“Año de la Universalización de la Salud”**

**UNIVERSIDAD PERUANA DE CIENCIAS APLICADAS**

**Facultad de Ingeniería**

**Curso:** Complejidad Algorítmica

**Sección: WV72**

**Profesor:** Luis Martín Canaval Sanchez

**Integrantes:**

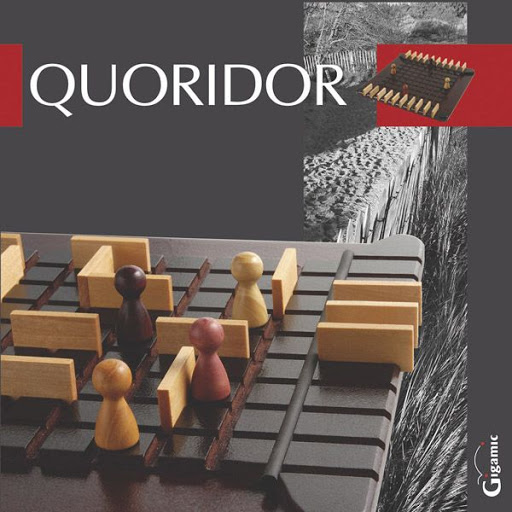
* Chacón Córdova, Sebastián U201816406
* Silva Krammer, Jhon U20181B177
* Vela Sarazú, Josmar U201611746

**Índice**

1. Introducción…………………………………………………………...3
2. Estado del Arte………………………………………………………...4
3. Metodología…………………………………………………………....5
4. Experimentos…………………………………………………………..6
5. Resultados……………………………………………………………...9
6. Conclusiones…………………………………………………………..12
7. Bibliografía……………………………………………………………13

**Introducción**

Quoridor es un juego de mesa, donde pueden participar entre 2 a 4 jugadores. Este juego fue diseñado por Mirko Marchesi y posteriormente publicado por Gigamic Games y Great American Trading Company. Con el creciente uso de computadoras en la sociedad moderna surgen nuevos desafíos. El desafío de crear I.A capaz de vencer a los humanos en un juego. Para algunos juegos esto es tarea sencilla, véase 3 en línea o damas, sin embargo, la complejidad de quoridor es similar a la del ajedrez. El objetivo de este proyecto es crear 3 agentes que usan diferentes algoritmos de pathfinding los cuales puedan jugar entre sí y provean un desafío al jugador humano.



**Estado del Arte**

No existen muchas investigaciones con respecto a este juego debido a que no es muy popular. Los principales intentos de construir una I.A que juegue Quorridor son los siguientes:

* Lisa Glendenning en el paper titulado *Mastering Quoridor*. En dicho paper Lisa usó un algoritmo iterativo profundo de búsqueda alpha-beta negamax ligeramente modificado. Esta I.A fue facilmente vencida por un humano.
* *A Quoridor-playing Agent*  por P. J. Merten. En este caso se usó el algoritmo MinMax con poda Alfa - beta, sin embargo el árbol que genera es demasiado grande para realizar busqueda MinMax. Por lo tanto, tal solución limita la profundidad de la búsqueda MinMax. Finalmente los resultados obtenidos fueron poco prometedores debido a que la I.A no era capaz de profundizar sus jugadas.

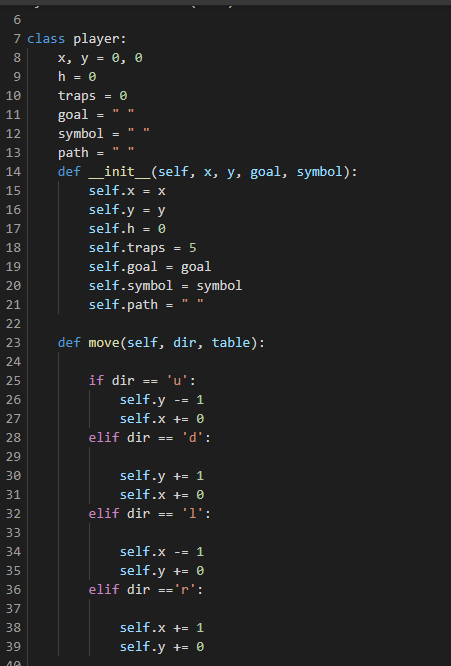
**Metodología**

Para este proyecto haremos uso del lenguaje Python y sus librerías entre ellas: numpy, queue y time. El juego consta de 3 jugadores cuyo objetivo principal será llegar al otro extremo del tablero. Además, cada jugador tiene que evitar que sus contrincantes lleguen a su objetivo, contarán con la habilidad de colocar barreras en cualquier posición, ya sea dónde está su contrincante o en otra parte del tablero.

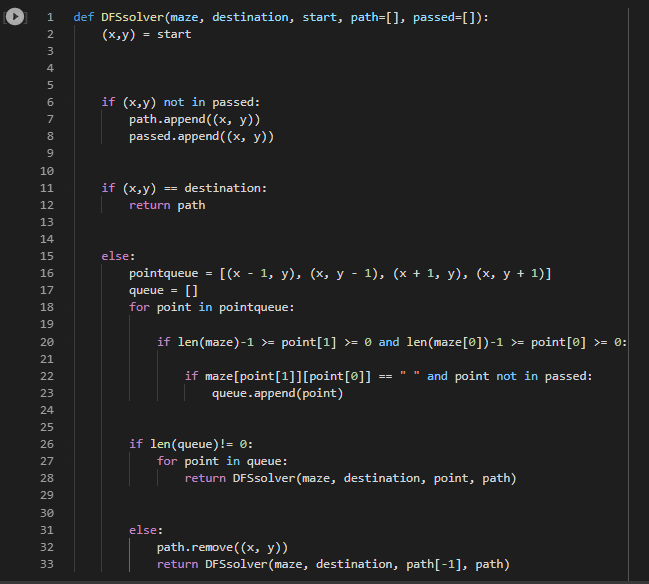
Primero tendremos que crear un tablero de 9 de largo por cada lado, esto será posible mediante una matriz bidimensional. Segundo implementaremos 3 bots mediante 3 algoritmos, los cuales se encargaran de buscar el mejor camino, así como determinar si debería impedir que su contrincante siga avanzando, para esto podrá colocar las barreras. Usaremos los algoritmos de BFS, DFS y Brute Force para crear a los bots. Estos algoritmos nos permiten hallar el camino más corto en un gráfico, el cual sería el tablero. El BFS es una técnica de búsqueda basada en vértices para encontrar una ruta más corta en un grafo. Por otro lado el DFS es un algoritmo cuya técnica de búsqueda se basa en los bordes de un grafo, recorriendo cada nodo de manera ordenada, pero no uniforme. Para este juego se implementará una función salto para los bots lo cual se dará en el momento en que estos se encuentren en un mismo camino frente a frente y otra que permita colocar barreras a los contrincantes. Para que la I.A decida qué jugada realizar se usará un algoritmo greedy, el cual, compara las distancias entre los jugadores y sus destinos y usa esta misma como heurística. De este modo la I.A decide qué jugada es la más conveniente. Finalmente, con los algoritmos implementados el juego Quoridor estaría terminado, y nos enfocaremos en mejorar los tiempos de ejecución y optimización.

**Experimentos**

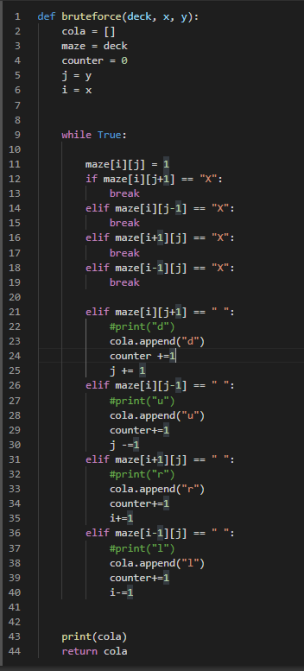
**Código:**

****

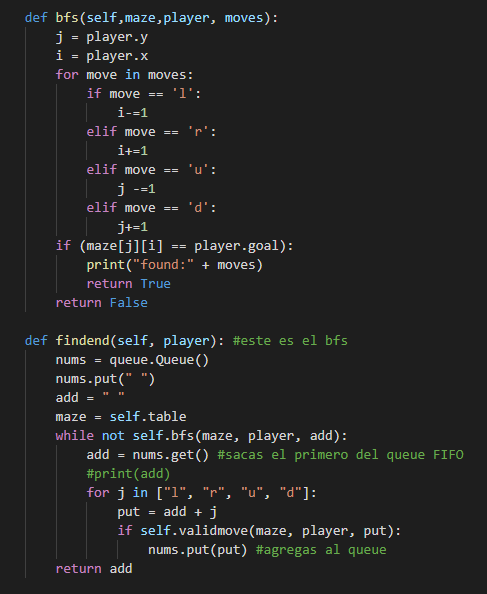
**Algoritmo DFS:**

****

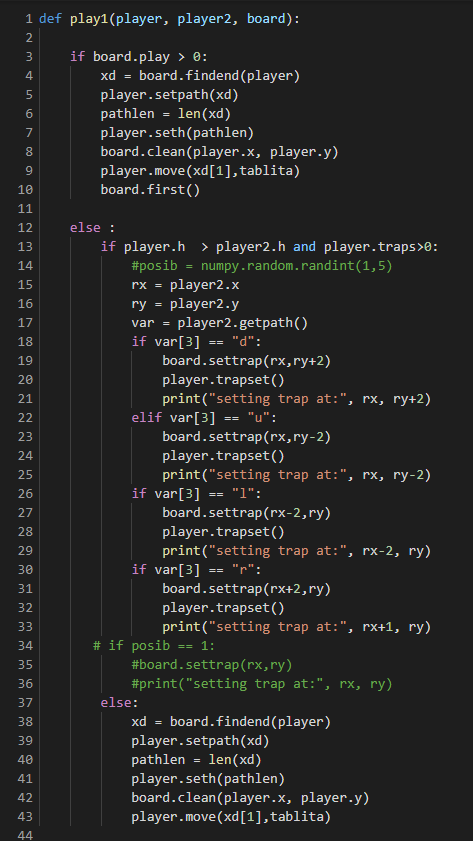
**Algoritmo Fuerza Bruta:**



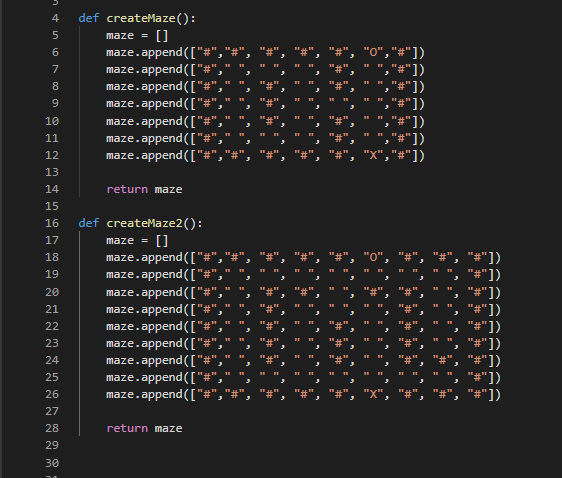
**Algoritmo BFS:**

****

**Primera jugada:**



**Creación del tablero:**

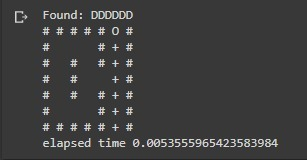


**Resultados**

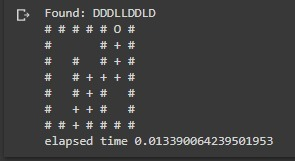
Para probar la eficiencia de cada algoritmo usado se creó un jupyter notebook aparte en el cual se tienen diferentes tamaños de tablero y worst case escenarios y best case scenarios. Los resultados fueron los siguientes:

1. **BFS (Breadth-first search)**

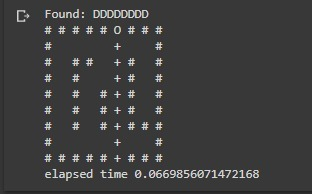
Best Case: Tablero 7 x7



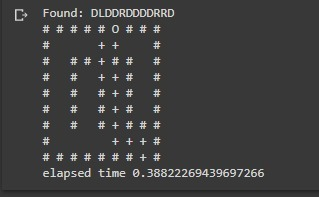
Worst Case: Tablero 7x7



Best Case: Tablero 9 x 9

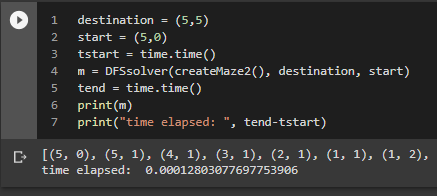


Worst Case: Tablero 9 x 9

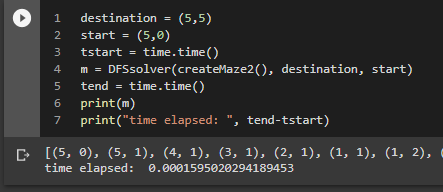


1. **DFS (Depth First Search)**

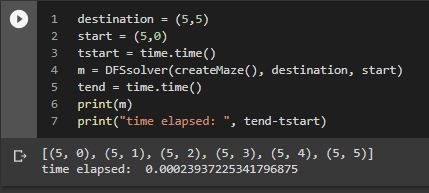
Best Case: Tablero 9 x 9



Worst Case: Tablero 9 x 9

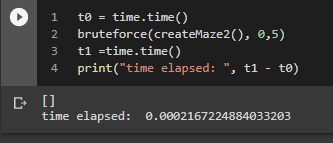


Best Case: Tablero 7 x 7



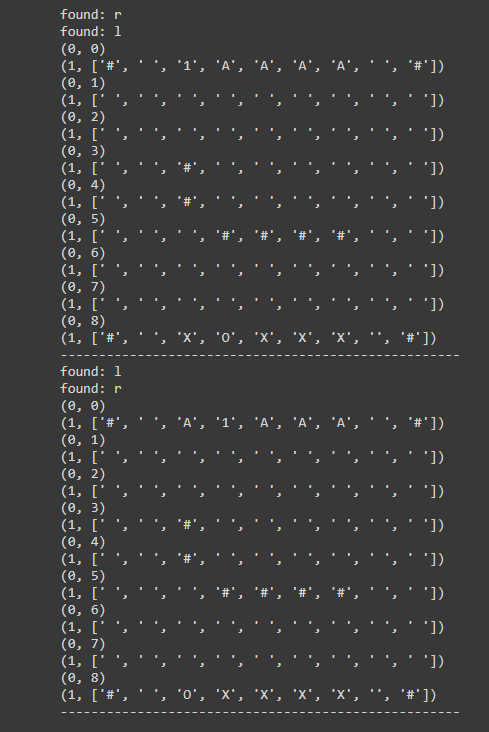
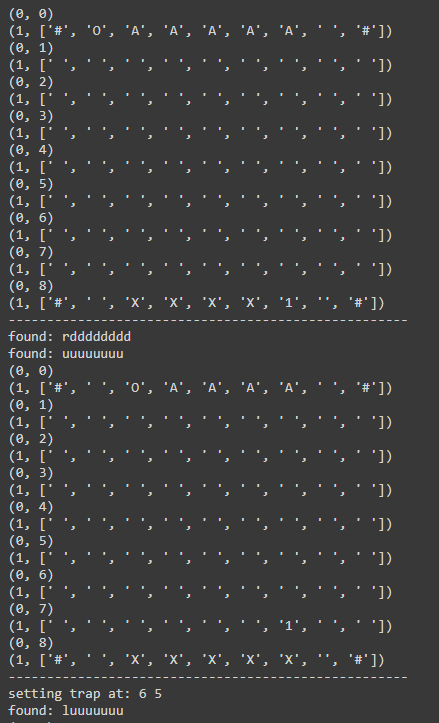
1. **Brute Force**

Only Case:



1. **Primera jugada:**

Nuestros bots tienen los símbolos de “0” y “1” y sus objetivos son “X” y “A” respectivamente.

****

Los experimentos evidencian que el cambio en la complejidad del tablero hace al BFS 5 veces más lento, y el cambio del tamaño del tablero lo hace 30 veces más lento. El DFS no varía mucho en ninguno de los casos y el Bruteforce tiene resultados bastante parecidos al dfs en todos los casos. Sin embargo, en un juego en el que la cantidad de jugadas es crucial para ganar, tanto el DFS como el Bruteforce tienen las de perder, debido a que no encuentran el camino óptimo, resultando en más jugadas para llegar a su destino.

**Conclusiones**

* Para el desarrollo de este proyecto fue necesario ver el proceso de juego de Quoridor. Esto con el fin de poder seleccionar los algoritmos necesarios para poder desarrollar el proyecto.

* En conclusión, hemos logrado los objetivos planeados, ya que entre los algoritmos que implementamos están Búsqueda en profundidad (DFS) y Búsqueda en anchura. Además, los tiempos de ejecución se encuentran en un rango óptimo.
* A pesar de que el DFS con backtracking y Brute Force toman menor tiempo de ejecución, no serían algoritmos óptimos para implementar en un juego de turnos, debido a que no encuentran el camino más corto y van a tomar más turno para llegar a su posición deseada. Por otro lado, el BFS con un tiempo de ejecución mayor a los ya mencionados, tomaría menos turnos, ya que encuentra el camino más corto. Este último fue el algoritmo usado para la implementación final del proyecto.
* Para colocar los muros y trampas fue necesario hacerlo en el segundo paso futuro del enemigo ya que al hacerlo en el primero este quedaba encerrado y por las reglas del juego esto no se puede dar.
* En un comienzo se intentó que cada bot recalculara el posible camino de su contrincante, sin embargo, esto resultó mucho más lento y consume más recursos, por lo cual se decidió almacenar los caminos calculados como variables temporales públicas las cuales se actualizan cuando es necesario. de este modo el contrincante puede acceder a ellas y el programa queda con una mejor optimización.

**Bibliografía**

* Mertens, P. (2006) *A quoridor-playing agent.* Recuperado de <https://upcommons.upc.edu/bitstream/handle/2117/127676/131875.pdf?sequence=1&isAllowed=y> [Consulta: 19 de septiembre de 2020]
* Glendenning, L. (2005) *Mastering quoridor.* Recuperado de <https://pdfs.semanticscholar.org/acad/6962a9bb3eb3fde4272f476d6625eb0a8182.pdf> [Consulta: 19 de septiembre de 2020]
* Gonsalez, R. (2010) *Python para todos.* Recuperado de <https://www.utic.edu.py/citil/images/Manuales/Python_para_todos.pdf> [Consulta: 19 de septiembre de 2020]
* Pound, Mike. (2017) *Maze Solving.* Recuperado de <https://www.youtube.com/watch?v=rop0W4QDOUI> [Consulta: 27 de Septiembre de 2020]