Oh-h1 - Métodos de búsqueda

Objetivos del trabajo

- Crear un Sistema de Producción para resolver el juego “Oh-h1”

- Analizar las diferencias entre algoritmos de búsqueda desinformados e informados.

- Diseñar varias heurísticas y analizar su comportamiento.

Oh-hl

El juego consiste en completar un tablero de NxN, siendo N un número par, con fichas de 2 colores, rojo y azul.

Reglas

- No pueden haber 3 o más casilleros contiguos con el mismo color en una fila o columna.
- Debe haber la misma cantidad de casilleros de color rojo y azul en cada fila y columna.
- No puede haber dos filas o columnas iguales.

Implementación

Motor

SearchStrategy -> Interfaz que representa los diferentes tipos de algoritmos de búsqueda que se pueden correr. Tiene los métodos:

a. *getNext*: devuelve el siguiente *Node* a analizar por el algoritmo y lo elimina de la frontera.
b. *addToFrontier*: agrega un nodo a la frontera.
c. *peekNext*: devuelve el siguiente *Node* a analizar sin eliminarlo de la frontera.
d. *getNumOfNodesInFrontier*: devuelve la cantidad de *Node* que hay en la frontera.

Motor -> Es la clase que se encarga de correr los algoritmos. Para poder correr recibe una *SearchStrategy* en su constructor. Con el método *run* ejecuta el algoritmo.

Juego

Ohh1Problem: -Representa el problema a resolver.
-Su constructor recibe el path a un archivo de texto con el tablero inicial.
-Genera las reglas que se aplican a los estados.

Ohh1State -Tablero
-Cantidad de posiciones vacías

Ohh1Rule: -Representa la transición de un estado a otro.
-Contiene una función para aplicarse a sí misma en un estado para transicionar a otro
-Contiene una función para poder comprobar si la regla es aplicable a dicho estado.

Heurísticas

Heurística 1 - Idea

- Se basa en las reglas 1 y 2.

- Debido a estas reglas quedan lugares donde el jugador está obligado a colocar una ficha de cierto color.

h1(n) = cantidadEspaciosVaciosNoObligados + 3 * cantidadEspaciosObligados

Heurística 2 - Mejora?

- Similar a la anterior en el sentido de que aplica las mismas reglas.

- Tiene en cuenta de que si se divide el tablero en 2 mitades, horizontalmente y/o verticalmente cada mitad tiene la misma cantidad de fichas rojas y azules cuando el tablero está terminado.

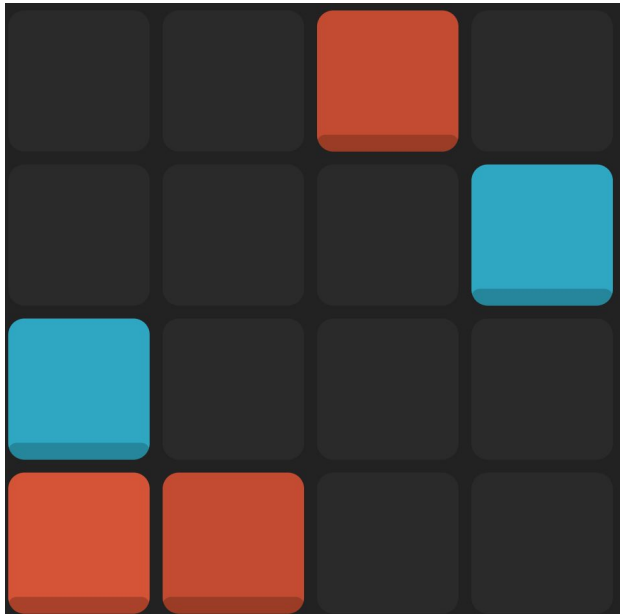
- Entonces, tomando en cuenta que tan completo esté el tablero, ya que no tiene sentido aplicar esta heurística cuando el tablero está con pocas piezas, el tablero con mayor similitud entre ambas mitades tiene el valor heurístico más bajo.

Comparaciones

Algoritmos no informados

La idea era comparar también con los algoritmos de búsqueda desinformados, pero dado que se tomó un timeout de 30 minutos y ninguno terminó antes de dicho tiempo no se incluirán estas comparaciones en los resultados.

A* vs Greedy



Greedy Search:

Heurística 1 : 9ms 16 nodos expandidos

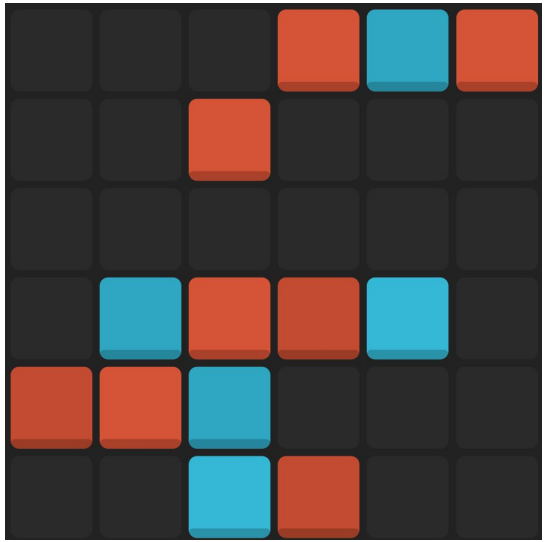
Heurística 2 : 11ms 18 nodos expandidos

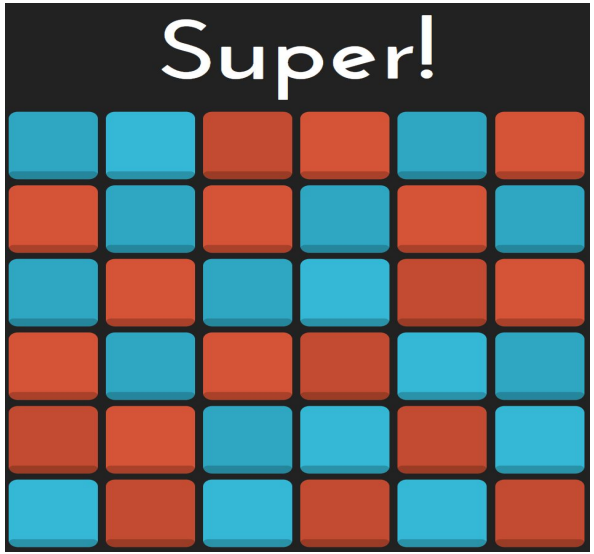
A* :

Heurística 1 : 187ms 3390 nodos expandidos

Heurística 2 : 194ms 2022 nodos expandidos

Tableros 6X6

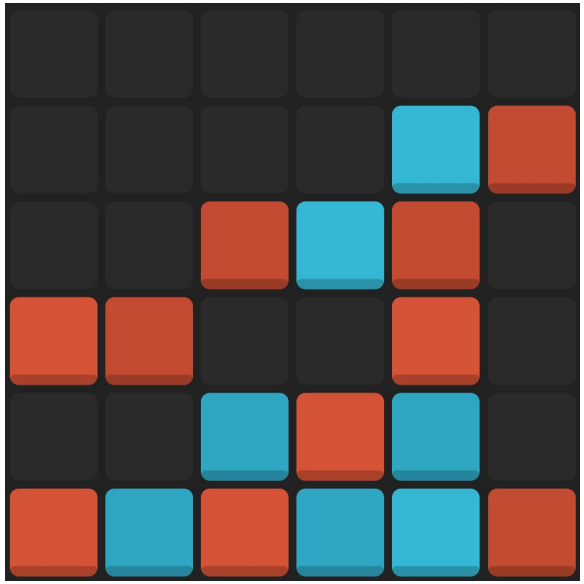


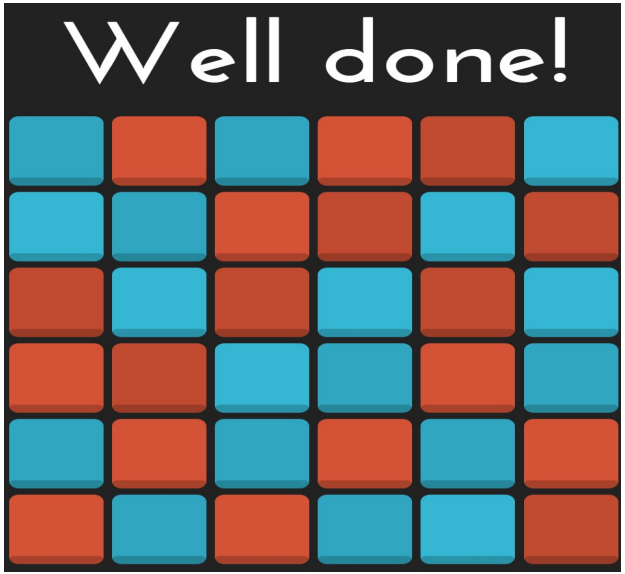


Heurística 1 : 550ms 10877 nodos expandidos

Heurística 2 : 21291ms 468907 nodos expandidos

Tableros 6X6

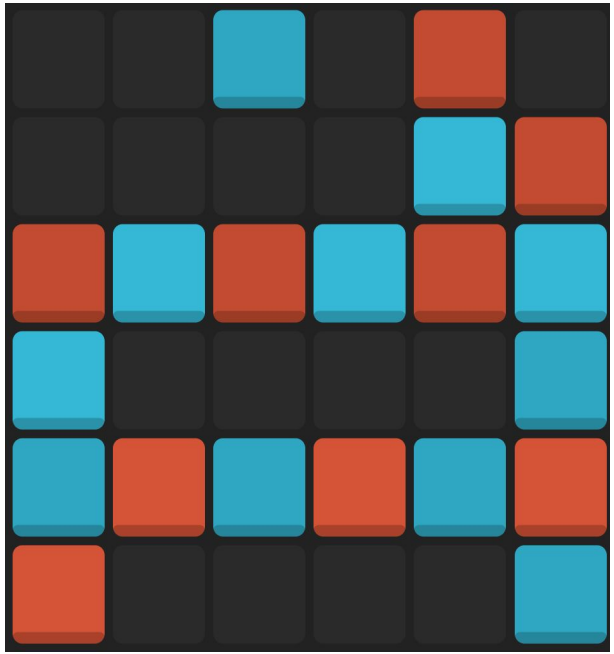




Heurística 1 : 18269ms 351210 nodos expandidos

Heurística 2 : 46801ms 886203 nodos expandidos

Tableros 6X6



Heurística 1 : 11816ms 255691 nodos expandidos

Heurística 2 : 8744ms 251590 nodos expandidos

Conclusiones

Conclusiones

- Los algoritmos informados resultaron ser mucho mejor que los desinformados teniendo en cuenta nuestras heurísticas.

- Ahora, teniendo en cuenta el algoritmo Greedy y A*, el primero es mucho mejor que A* para nuestras heurísticas, en tiempo y en resultado, ya que cualquier solución que se encuentre que cumpla con las reglas del juego, ya es óptima.

- En cuanto a la comparación de los resultados entre heurísticas, la primera resultó ser la que mejor resultado dio, a pesar de que la segunda pretendía ser una mejora de esta.