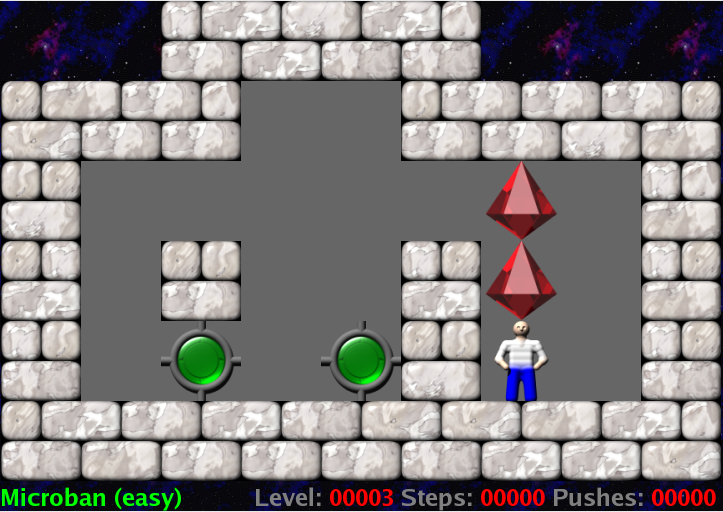
| 750033M: INTELIGENCIA ARTIFICIAL Gr.01 (Fecha: 27/08/20201)  Práctica # 1  Código:\_\_\_\_\_\_\_\_\_\_\_ Código:\_\_\_\_\_\_\_\_\_\_\_ Código:\_\_\_\_\_\_\_\_\_\_\_ |
| --- |

**Objetivo**: Poner en práctica los conceptos aprendidos durante la primera parte del curso de Introducción a la inteligencia artificial. Durante esta práctica se evaluará la capacidad de los estudiantes para realizar las implementaciones de los algoritmos de búsqueda informada o no informada y de juegos.

SOKOBAN

Implementar un programa que resuelva tableros de Sokoban usando 3 de los métodos de búsqueda no informada vistos en clase. Los algoritmos han sido implementados en los talleres previos.

Sokoban es un juego japonés de Arcade donde un almacenista debe empujar unas cajas hasta las posiciones marcadas en el mapa. En la imagen, las cajas están representadas por los rubíes y los objetivos son los puntos verdes en el mapa. Una restricción importante es que el almacenista no puede jalar las cajas, sólo empujarlas. Luego si una caja cae en una esquina el juego se pierde porque ya no se puede mover. Otra restricción es que el almacenista solo puede empujar una caja a la vez. Si las cajas se encuentran adyacentes es como si estuvieran al lado de una pared. Las cajas que ya están en un punto de llegada(verde) se pueden mover cuantas veces se requiera. El juego termina cuando todas las cajas están en los puntos de llegada.



1. Para resolver tableros de sokoban, implementar los algoritmos de búsqueda:
   1. Preferente por profundidad (max 64 niveles)
   2. Preferente por amplitud (max 64 niveles)
   3. Profundidad iterativa (Inicial 10 y aumentando 1 nivel por iteración) (max 64 niveles)

Los problemas pueden ser de una, dos o hasta tres cajas. La calificación se dará dependiendo de los tableros que pueda resolver de un conjunto que no será conocido por los estudiantes hasta el día de la evaluación. En la carpeta de este documento se encuentran 4 niveles de prueba.

El programa debe recibir como único argumento el archivo de texto con el tablero a resolver, y debe producir como resultado una cadena de texto con el conjunto de acciones del agente:

U: arriba

D abajo

L: izquierda

R: derecha

En caso de que se puedan realizar varias jugadas en un determinado estado, estas deben evaluarse en el orden U, D, L y R

Este programa se puede implementar en cualquier lenguaje de programación, pero deben tener scripts para configurar y compilar el proyecto en un solo paso(build.sh) , y un script para ejecutar el programa en una sola línea (run.sh < nivel1.txt). Se debe proveer una versión precompilada del programa con la entrega.

**El formato de los archivos de entrada**

Un conjunto de líneas con el mapa del nivel. El mapa contiene sólo los caracteres W(pared), 0(vacío) y X(lugar para poner la caja). Cada línea del mapa debe contener al menos dos W.

La siguiente línea después del mapa contiene la posición del jugador en el formato ***fila, columna***. Los índices comienzan en 0.

Las líneas siguientes contienen las coordenadas de las cajas en el tablero en el formato ***fila, columna.***

La siguiente tabla muestra el tablero 1. Este tiene un mapa de 6x6. Los puntos objetivo están en las posiciones (2, 1) y (2, 3). El almacenista está en la posición (1, 1) y las cajas en las posiciones (2, 2) y (2, 3)

| ***WWWWWW***  ***W000WW***  ***WX0X0W***  ***W00W0W***  ***W0000W***  ***WWWWWW***  ***1,1***  ***2,2***  ***2,3*** |
| --- |

El programa debe producir como salida una cadena de caracteres con las acciones para cada tipo de búsqueda. La primera línea para la búsqueda por profundidad, la segunda línea para búsqueda por amplitud y la tercera por profundidad iterativa.

Ejemplo de salida:

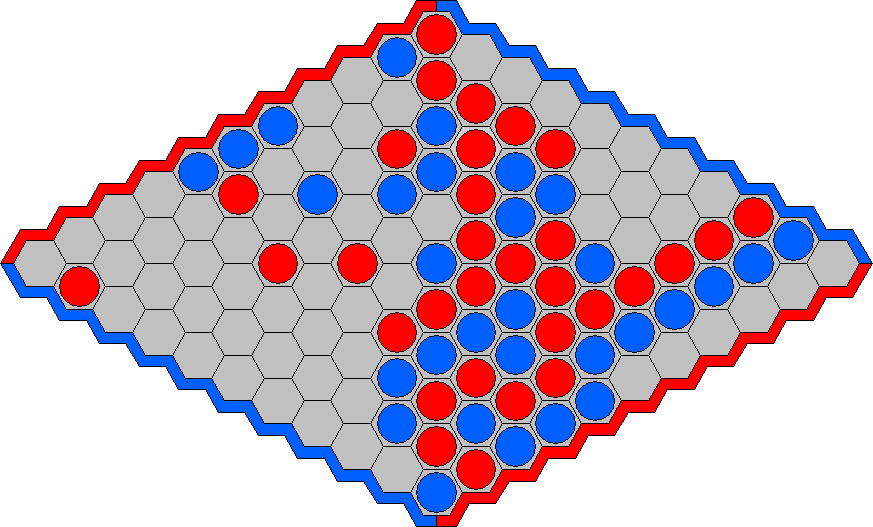
| RUUDLLLRRLLLLLRRLLUULLLRRLLUUUU  ULLLLRRLLRUUDLLLLRRLLUULLLLRRLLUULRRLLUU  LLLRRLLUURUUDLLLRRLLUU |
| --- |

***Restrinja todas las búsquedas a máximo 64 niveles de profundidad. No agregar comas ni espacios a la salida.***

* Se califica la correctitud de las soluciones
* Calidad del código y el proyecto en términos de documentación y eficiencia

HEX

2. Implemente un agente inteligente que juegue Hex. La estrategia a implementar puede ser elegida por el grupo de estudiantes. La calificación se dará por el resultado de una competencia entre grupos.



En el proyecto de github está el ejemplo con un agente completamente aleatorio:

<https://github.com/andcastillo/ai-reactApp>

Cada grupo debe implementar un agente, modificando el archivo [HexAgent.js](https://github.com/andcastillo/ai-reactApp/blob/master/modelHex/HexAgent.js). Tenga en cuenta las siguientes observaciones:

* El tablero de Hex se ha implementado como una matriz cuadrada de nxn, donde n es el tamaño del juego y n es un número impar.
* Cada celda vacía está representada por un 0 y las casillas ocupadas tendrán los caracteres ‘1’ y ‘2’.
* Sus jugadas son la que se indiquen con el mismo id del agente. Este lo puede saber con la función getID();
* La función send() que retorna la jugada a realizar, debe retornar un arreglo de 2 posiciones, con las coordenadas de la jugada a realizar, ejemplo: [2,2].
* Debe garantizar que la jugada a realizar es válida. En caso contrario el programa genera una jugada aleatoria para el jugador.
* Cada grupo debe implementar y explicar su heurística
* Cada agente puede explorar máximo 10 niveles en el árbol MinMax

La competencia será todos contra todos. En cada enfrentamiento se elegirá de manera aleatoria quien será el jugador que comienza jugando.

## Entregables

Se debe entregar un archivo comprimido con 2 carpetas adentro. Una para el problema de Sokoban y otra carpeta que contenga el archivo HexAgent.js y package.json donde se especifiquen las librerías que usaron para la implementación.