Diseño de Agentes Inteligentes

Tarea 2. Agentes Solucionadores de Problemas (PSA) y búsqueda

Descripción del problema

Esta tarea consiste en resolver un problema al formularlo como un problema de estado simple para ser resuelto por un agente de resolución de problemas (PSA), y luego resolverlo utilizando métodos de búsqueda ciega y heurística seleccionados.

Miembros del equipo

Escriban la matrícula y el nombre de cada miembro en una línea diferente.

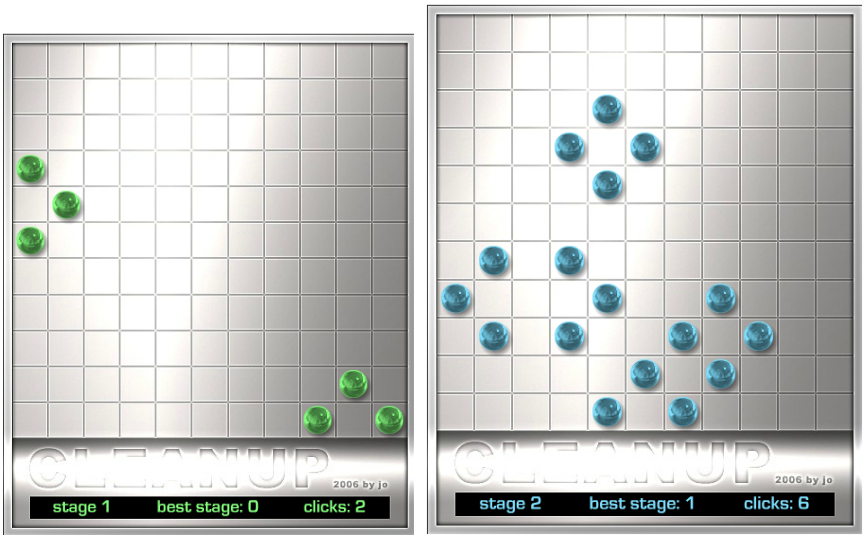
1: Ricardo Kaleb Flores Alfonso A01198716
2: Sebastián Miramontes Soto A01285296
3: Raúl Correa Ocañas A01722401

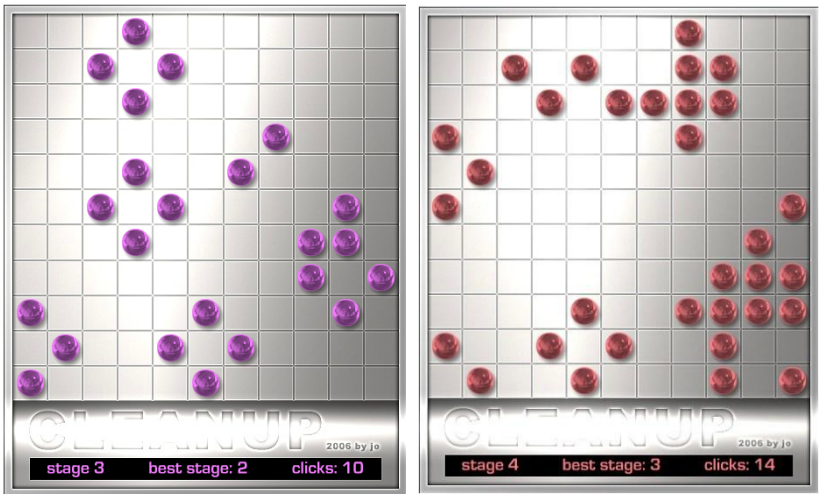
Cleanup Puzzle

Cleanup Puzzle es un juego de mesa que requiere que limpies el tablero. Para ello, deberás hacer clic en las fichas de un tablero tipo cuadrícula. Cada ficha puede estar vacía o contener una bola. El mosaico en el que haga clic no se verá afectado, pero sus vecinos directos (horizontal y verticalmente) se invertirán (si el mosaico contiene una bola, se vaciará y viceversa). El objetivo es determinar una secuencia de clics que elimine todas las bolas del tablero.

El juego, que puedes encontrar y jugar en el enlace https://www.mathsisfun.com/games/cleanup-puzzle.html , consiste en un tablero de 11x11 que se juega en una secuencia de etapas de complejidad incrementada. La complejidad de un escenario está relacionada con la cantidad de bolas en el tablero y sus ubicaciones. En el juego en línea, hay un límite de clics definido para cada etapa y el juego termina cuando no puede encontrar una secuencia de clics lo suficientemente pequeña para limpiar el tablero.

Ejemplos de diferentes etapas del juego en línea:





Para esta actividad debes hacer lo siguiente:

1. Analizar el problema para determinar la información que es relevante para **su formulación como PSA**. Elija una forma de representar estados en el lenguaje de programación (**Python**) donde el tamaño del tablero cuadriculado NxN debe ser un atributo del problema. El número de clics dados en el juego en línea no es relevante para esta tarea.
2. Implemente el código requerido para poder resolver diferentes instancias del problema utilizando los métodos de búsqueda ciega y búsqueda heurística ya programados e incluidos en el código Python de **SimpleAI**. Cuando se ejecutan, deben mostrar correctamente la secuencia de estados y acciones para resolver el problema.
3. Su código **debe funcionar correctamente** al ser utilizado por las funciones que implementan los métodos de búsqueda respectivos.
4. Ejecutar los métodos de búsqueda ciega y heurística para resolver dos instancias del problema, una fácil y otra difícil, y mostrar las soluciones encontradas por cada método.
5. Sería muy bueno y premiado sí puede calcular alguna **estadística** sobre los comportamientos de los distintos métodos.
6. Agregue observaciones y conclusiones sobre su experiencia al programar y usar cada algoritmo para encontrar las soluciones: ¿Podrían resolver todos o algunos problemas? ¿Qué tan eficientes fueron? ¿Fue difícil programar el PSA? ¿Qué fue lo más difícil? Además, ¿qué algoritmo parece ser el mejor para los problemas seleccionados? ¿Y qué métodos encontraron las mejores soluciones?

CRITERIOS DE EVALUACIÓN:

Los pesos asignados a las actividades para la evaluación de esta actividad son:

- Formulación PSA: 20%
- Implementación de código python para el problema: 50%
- Solución de varios problemas con los diferentes algoritmos de búsqueda ciega: 20%
- Observaciones y conclusiones: 10%

La calificación será aumentada (otorgada) o reducida (penalizada) dependiendo de la calidad del documento PDF entregado y la documentación interna del código Python con comentarios.

Descripción formal del agente deseado

1. Descripción de los estados

Lenguaje para describir los estados del mundo. No olvide identificar completamente cada elemento relevante (nómbrelo).

completo(true) false

Tabla de 11x11, con estados booleanos

```
goal = ((0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0),
        (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0),
        (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0),
        (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0),
        (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0),
        (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0),
        (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0),
        (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0),
        (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0),
        (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0))
```

Los 0 significa que no hay una canica en ese espacio y el 1 que si

2. Estado inicial

Use el lenguaje que diseñó para describir el estado inicial del problema.

Sabiendo que N = 11

```
init = ((0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0),
        (1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0),
        (0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0),
        (1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0),
        (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0),
        (0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0),
        (0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0),
        (0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0),
        (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0),
        (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0),
        (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0))
```

3. Acciones

Nombre y describa informalmente las acciones (operadores) que el agente puede usar para realizar sus tareas.

Clic(fila, columna)

esto puede quitar o agregar canicas de las que están hacía los cuatro polos de la casilla.

ejemplo:

Clic(3, 2) donde 3 es la fila y la 2 es la columna.

Esto quitaría o agregaría canicas en (3, 1), (3, 2), (4, 2) y (2, 2).

4. Modelo de transición

Descripción detallada de cada acción, incluyendo su nombre, argumentos, condiciones previas y efectos en la descripción del resultado en estados.

Operador	Pre-Condiciones	Modificar
Clic	Debe ser una posición indexable	Intentará acceder al elemento anterior y siguiente, y a los elementos en la columna anterior y siguiente. Y usará el operador not, para cambiar el estado si es que es posible acceder a ese elemento.

5. Prueba de meta

Cómo reconocer el final de la tarea.

Para que el problema sea resuelto todos los elementos de la tabla deberán ser 0

```
goal = ((0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0),
        (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0),
        (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0),
        (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0),
        (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0),
        (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0),
        (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0),
        (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0),
        (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0),
        (0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0))
```

6. Costo de cada acción

Cómo cuantificar el costo de las acciones a utilizar durante la ejecución de la tarea.

Para el algoritmo de búsqueda ciego el costo de acción varía dependiendo de la posición donde se tome la acción y las bolas que hayan alrededor

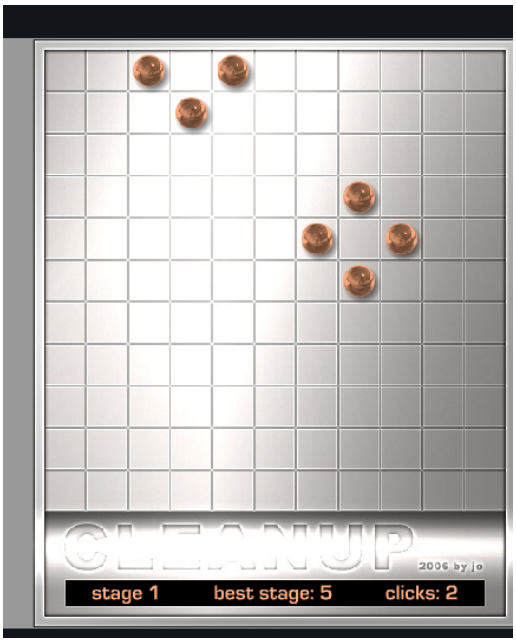
Soluciones

Ciego: Por costo uniforme

Dificultad 1:

```
Path from initial to goal:
[(None,
(0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0),
(0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0),
(0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0),
(0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0),
(0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0),
(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0),
(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0),
(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0),
(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0),
(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0),
(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0))),
([0, 3, 0],
(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0),
(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0),
(0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0),
(0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0),
(0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0),
(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0),
(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0),
(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0),
(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0),
(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0),
(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0))),
([3, 8, 0],
(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0),
(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0),
(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0),
(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0),
(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0),
(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0),
(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0),
(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0),
(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0),
(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0),
(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0)))]

Stats: {'max_fringe_size': 13, 'visited_nodes': 4, 'iterations': 4}
```

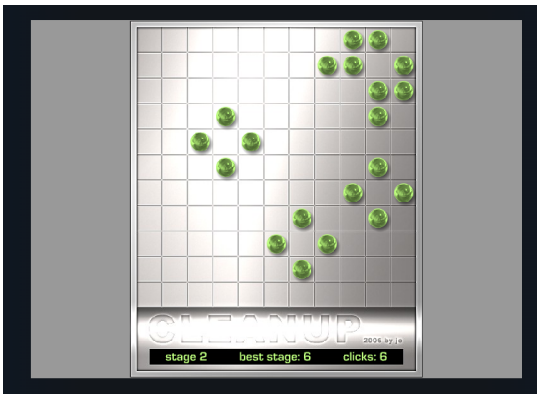


Dificultad 2:

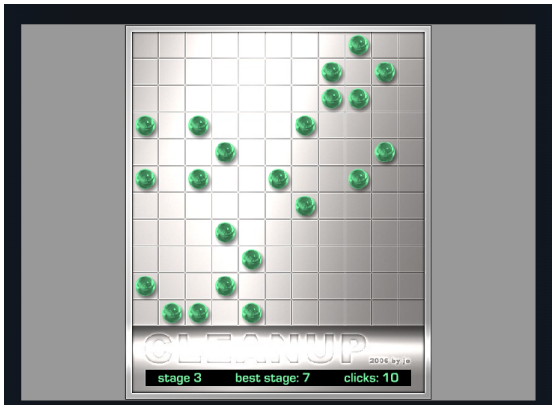
```
Path from initial to goal:
[(None,
(0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 0),
(0, 0, 0, 0, 0, 0, 0, 1, 1, 0, 1),
(0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 1),
(0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0),
(0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0),
(0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0),
(0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1),
(0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0),
(0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0),
(0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0))),
([1, 9, 0],
(0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0),
(0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0),
(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1),
(0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0),
(0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0),
(0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0),
(0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1),
(0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0),
(0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0),
(0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0))),
([2, 9, 30],
(0, 0, 0, 0, 0, 0, 0, 0, 1, False, 0),
(0, 0, 0, 0, 0, 0, 0, 1, False, True, False),
(0, 0, 0, 0, 0, 0, 0, 0, True, False, False),
(0, 0, 0, 1, 0, 0, 0, 0, 0, False, 0),
(0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0),
```

```
(0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0),
(0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1),
(0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0),
(0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0),
(0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0),
([1, 8, 0],
(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0),
(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0),
(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0),
(0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0),
(0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0),
(0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 0),
(0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 1),
(0, 0, 0, 0, 0, 0, 1, 0, 0, 1, 0),
(0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0),
(0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0))),
([6, 9, 0],
(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0),
(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0),
(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0),
(0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0),
(0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0),
(0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0),
(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0),
(0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0),
(0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0),
(0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0))),
([4, 3, 0],
((0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0),
(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0),
(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0),
(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0),
(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0),
(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0),
(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0),
(0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0),
(0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 0),
(0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0))),
([8, 6, 0], ((0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0),
(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0),
(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0),
(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0),
(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0),
(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0),
(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0),
(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0),
(0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0))))]
```

```
Stats: {'max_fringe_size': 1823, 'visited_nodes': 260, 'iterations':
260}
```

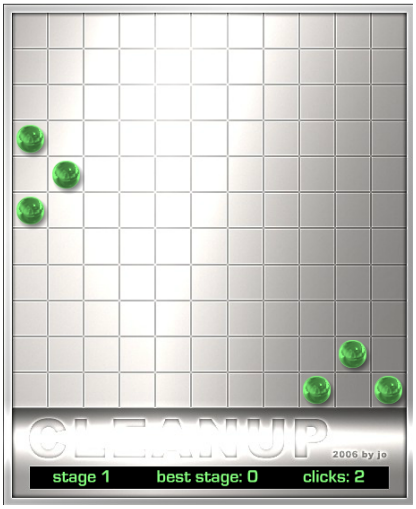


Dificultad 3:



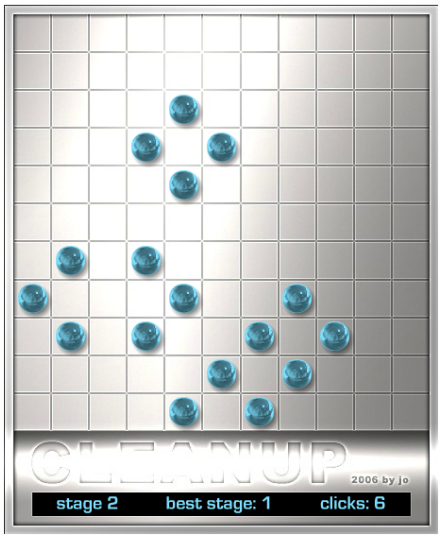
Heurística:

Dificultad 1:



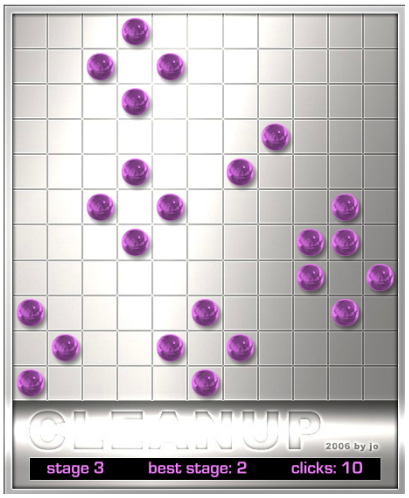
A*	Greedy
>> Búsqueda A* << Configuración inicial ((False, False, False, False, False, False, False, False, False, False, False), (False, False, False, False 1 (4, 0) ((False, False, False, False, False, False, False, False, False, False, False), (False, False, False, False 2 (10, 9) ¡Meta lograda con costo = 2 ! ((False, False, False, False, False, False, False, False, False, False, False), (False, False, False, False	>> Greedy << Configuración inicial ((False, False, False, False, False, False, False, False, False, False, False), (False, False, False, False 1 (4, 0) ((False, False, False, False, False, False, False, False, False, False, False), (False, False, False, False 2 (10, 9) ¡Meta lograda con costo = 2 ! ((False, False, False, False, False, False, False, False, False, False, False), (False, False, False, False
A* Stats: {'max_fringe_size': 15, 'visited_nodes': 3, 'iterations': 3}	Greedy Stats: {'max_fringe_size': 15, 'visited_nodes': 3, 'iterations': 3}

Dificultad 2:



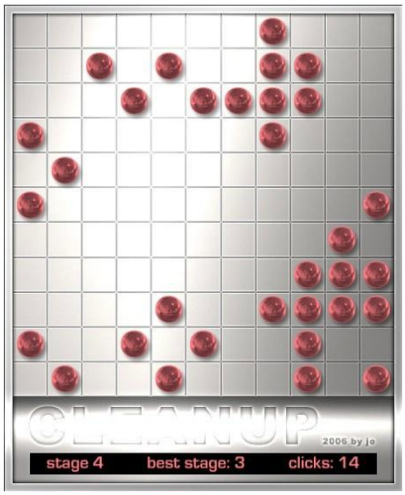
A*	Greedy
>> Búsqueda A* << Configuración inicial ((False, False, False, False, False, False, False, False, False, False, False), (False, False, False, False 1 (3, 4) ((False, False, False, False, False, False, False, False, False, False, False), (False, False, False, False 2 (8, 7) ((False, False, False, False, False, False, False, False, False, False, False), (False, False, False, False 3 (10, 5) ((False, False, False, False, False, False, False, False, False, False, False), (False, False, False, False 4 (7, 1) ((False, False, False, False, False, False, False, False, False, False, False), (False, False, False, False 5 (7, 3) ¡Meta lograda con costo = 5 ! ((False, False, False, False, False, False, False, False, False, False, False), (False, False, False, False	>> Greedy << Configuración inicial ((False, False, False, False, False, False, False, False, False, False, False), (False, False, False, False 1 (3, 4) ((False, False, False, False, False, False, False, False, False, False, False), (False, False, False, False 2 (8, 7) ((False, False, False, False, False, False, False, False, False, False, False), (False, False, False, False 3 (10, 5) ((False, False, False, False, False, False, False, False, False, False, False), (False, False, False, False 4 (7, 1) ((False, False, False, False, False, False, False, False, False, False, False), (False, False, False, False 5 (7, 3) ¡Meta lograda con costo = 5 ! ((False, False, False, False, False, False, False, False, False, False, False), (False, False, False, False
A* Stats: {'max_fringe_size': 95, 'visited_nodes': 6, 'iterations': 6}	Greedy Stats: {'max_fringe_size': 95, 'visited_nodes': 6, 'iterations': 6}

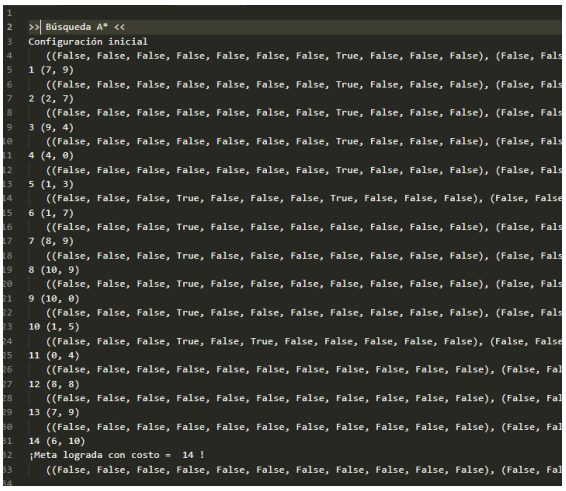
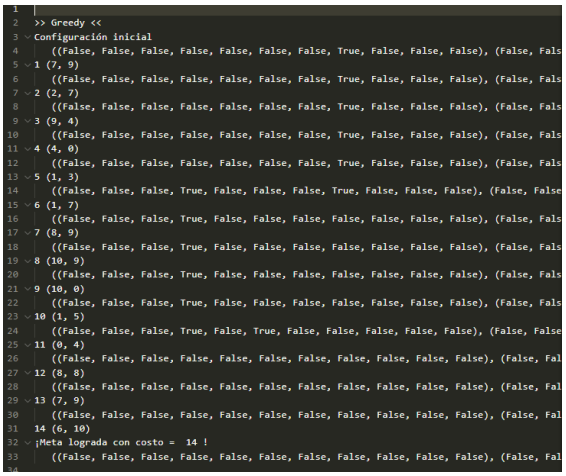
Dificultad 3:



A*	Greedy
>> Búsqueda A* << Configuración inicial ((False, False, False, True, False, False, False, False, False, False, False), (False, False, True, False, 1 (1, 3) ((False, False, False, False, False, False, False, False, False, False, False), (False, False, False, False 2 (5, 3) ((False, False, False, False, False, False, False, False, False, False, False), (False, False, False, False 3 (7, 9) ((False, False, False, False, False, False, False, False, False, False, False), (False, False, False, False 4 (9, 5) ((False, False, False, False, False, False, False, False, False, False, False), (False, False, False, False 5 (9, 0) ((False, False, False, False, False, False, False, False, False, False, False), (False, False, False, False 6 (4, 7) ((False, False, False, False, False, False, False, False, False, False, False), (False, False, False, False 7 (5, 8) ¡Meta lograda con costo = 7 ! ((False, False, False, False, False, False, False, False, False, False, False), (False, False, False, False	>> Greedy << Configuración inicial ((False, False, False, True, False, False, False, False, False, False, False), (False, False, True, False, 1 (1, 3) ((False, False, False, False, False, False, False, False, False, False, False), (False, False, False, False 2 (5, 3) ((False, False, False, False, False, False, False, False, False, False, False), (False, False, False, False 3 (7, 9) ((False, False, False, False, False, False, False, False, False, False, False), (False, False, False, False 4 (9, 5) ((False, False, False, False, False, False, False, False, False, False, False), (False, False, False, False 5 (9, 0) ((False, False, False, False, False, False, False, False, False, False, False), (False, False, False, False 6 (4, 7) ((False, False, False, False, False, False, False, False, False, False, False), (False, False, False, False 7 (5, 8) ¡Meta lograda con costo = 7 ! ((False, False, False, False, False, False, False, False, False, False, False), (False, False, False, False
A* Stats: {'max_fringe_size': 190, 'visited_nodes': 9, 'iterations': 9}	Greedy Stats: {'max_fringe_size': 190, 'visited_nodes': 9, 'iterations': 9}

Dificultad 4:



A*	Greedy
	
A* Stats: {'max_fringe_size': 450, 'visited_nodes': 20, 'iterations': 20}	Greedy Stats: {'max_fringe_size': 420, 'visited_nodes': 17, 'iterations': 17}

Observaciones y conclusiones

¿Podrían resolver todos o algunos problemas?

Ambos ejemplos, tanto el ciego como el heurístico lograron resolver los problemas tanto de dificultad 1 como de dificultad 2. Sin embargo, el heurístico pudo resolver problemas de dificultades 3 y 4, debido a la optimización de la función heurística con la distancia Manhattan.

¿Qué tan eficientes fueron?

Los métodos más eficientes para resolver los problemas fueron aquellos formulados como algoritmos de búsqueda informada. Tanto el algoritmo de A* como el algoritmo de búsqueda codiciosa tuvieron excelentes resultados, donde en su máxima dificultad solo fue necesario un máximo de 20 nodos. En contraste, el algoritmo utilizado para resolver el problema sin una heurística necesitó 260 nodos para un problema de dificultad 2. Observamos adicionalmente que en las estadísticas, la búsqueda codiciosa tuvo mejores resultados que A*, visitando 3 nodos menos y generando una frontera con 30 elementos menores.

¿Fue difícil programar el PSA?

En nuestro caso, tuvimos problemas en diferentes instancias de la programación. En un inicio nos marcaron ciertos errores como las listas, después se nos dificulta programar de manera óptima la heurística, y también el poder lograr que llegara a la solución para ambas dificultades de una manera rápida.

¿Qué fue lo más difícil?

Sin lugar a duda el tiempo que nos tomó la programación, el lograr que el de búsqueda ciega llegará al resultado, y el poder hacer una heurística rápida y funcional, fueron las dos cosas que hicieron que este entregable tuviera mayor dificultad.

¿Qué algoritmo parece ser el mejor para los problemas seleccionados? ¿Y qué métodos encontraron las mejores soluciones?

El algoritmo con Heurística fue el mejor, ya que nos daba la solución óptima siempre, por más que tardará algo de tiempo en un inició y después lograramos optimizarlo, aunque el de búsqueda ciega tampoco fue malo, ya que en las instrucciones se nos mencionó que no tomáramos en cuenta el número de clicks para esta actividad, aunque si se le agregara un requisito de tiempo para accionar, el ciego sería una mejor opción para trabajar.

Conclusión:

Para esta actividad, nosotros logramos poder poner en práctica el conocimiento adquirido en las primeras dos semanas de clase, sobretodo acerca de los PSA´s, la programación de clases en Python, las heurísticas, entre otras cosas. Pudimos notar el tiempo que conlleva el programar un PSA desde cero de una manera óptima, y también logramos entender de una manera práctica los usos de los métodos de búsqueda. Nosotros optamos por usar una heurística de estilo “A*” en combinación con "Selección por Mayoría" o "Selección Basada en Cantidad" en donde el robot iniciaba a seleccionar las casillas que se encontraban cerca a una mayor concentración de canicas dentro del tablero considerando también que tan cerca estaría de la meta tras hacer cada movimiento, mientras que para el de método ciego decidimos optar por la búsqueda en base a un costo uniforme, el cuál consistía en darle el mismo costo a cada uno de los movimientos lo cuál aseguraba encontrar una solución y que trataría de encontrar la solución que gastará menos, pero no aseguraba la mejor.

Podemos concluir acerca de la importancia de un buen programa, con todas sus partes y una buena estructura en el código para que pudiéramos llegar a la meta, también que aunque es difícil de programar, si se le incluye una heurística es de gran ayuda para el sistema, ya que técnicamente se le está dando una ayuda de por donde debe de iniciar y continuar hasta llegar a la meta.