Proyecto Integrador: Segundo Avance Búsquedas

Equipo 5

Sebastían Neri	A01750190
Aldo Sandoval	A01751137
Emiliano Padilla	A01658972

Descripción PEAS

(Performance - Environment - Actuators - Sensors)

Performance	*Environment*	*Actuators*	*Sensors*
El rover tiene la capacidad de recorrer el planeta recolectando muestras. Cuando encuentra una muestra manda una señal con las coordenadas del lugar donde se encuentra el agua.	<u>Zonas rocosas en Marte</u> Es accesible porque todos sus sensores pueden detectar la información disponible del entorno. Es determinista porque es un agente único el que se encuentra en el planeta, por lo que el muestreo que realice el agente mutará el medio, dada extracción y el planeta no tiene actividad sísmica ni fenómenos metereológicos. Es no episódico (secuencial) porque las decisiones tomadas por el rover afectan las muestras que tome. Estático porque no hay otro agente con el rover y porque no tiene fenómenos naturales.	El extractor de muestras, sistema de movimiento (llantas, motores), dispositivo de comunicación	Cámaras, micrófonos, sensores de viento, escáner de entornos, radar.

	Es discreto porque el entorno donde está el rover tiene un número finito de estados.		

Descripción Formal del Problema

1. Espacio de estados

Se decidió que el estado que tendría el Rover sería una posición en el mapa (string) por lo que el espacio de estados sería el siguiente:

- X = n+1 ... n-1

  siendo *n* el número de columnas en nuestro mapa contando el carácter \n. Además, no puede estar en la posición donde se encuentre un obstáculo o muro.

2. Estado inicial

El estado inicial es una posición en el string representada por la letra 'Q'. Este estado es completamente aleatorio.

3. Estado meta

El estado meta es una posición en el string representada por el carácter '_'. Este estado es completamente aleatorio y puede haber múltiples estados meta en el mapa.

4. Reglas

I. El rover no puede moverse en el medio cuando la diferencia entre regiones es mayor a 1.
II. El rover se mueve como cualquier automóvil. Por lo tanto, no puede hacer movimientos laterales. Los movimientos disponibles son: arriba, arriba derecha, arriba izquierda, atrás, atrás derecha, atrás izquierda.
III. Tampoco puede subirse a los obstáculos ni salirse del mapa.

Función costo

Definimos nuestros costos a partir del esfuerzo que le tomaría al Rover llegar a ese lugar. Arriba y atrás se encuentran más cerca que los laterales, por eso se aumenta el costo 0.4. A continuación se muestra el diccionario que se definió dentro del programa de python.

```
costos = {
    "adelante": 1.0,
    "adelante derecha": 1.4,
    "adelante izquierda": 1.4,
    "atras": 1.0,
    "atras derecha": 1.4,
    "atras izquierda": 1.4
}
```

De igual manera, la función costo recibe dos estados y una acción. El primer condicional verifica si el nivel actual es mayor o igual al nivel de la siguiente posición o si en la siguiente posición está el carácter con el que definimos el lago. Si se cumple, el costo depende de la acción que se realizó. En caso contrario, se convierte a entero el nivel del siguiente estado y el costo por subir será ese nivel más un 0.4 si se movió a los laterales.

```
def cost(self, state, action, next_state):
    if self.map[state] >= self.map[next_state] or self.map[next_state] == "_":
        return costos[action]
    else:
        return int(self.map[next_state]) + costos[action] - 1
```

Heurísticas

Para definir las heurísticas, primero implementamos un algoritmo de búsqueda llamado Nearest Neighbours Search, el cual calculará el lago más cercano al rover dependiendo de su distancia euclidiana.

Nearest Neighbour Search

```
def findLakes(self):
    lakes = re.finditer("_", self.map)
    points = [lake.start() for lake in lakes]
    coors = [[self.getCoors(point), point] for point in points]
    distances = [[self.getDistanceFromRover(point[0]), point[1]] for point in coors]
    return np.array(self.getCoors(min(distances)[1]))
```

Una vez implementada esta función, se definieron 2 heurísticas para comparar su desempeño en los algoritmos de greedy y A*. En el primer caso se implementó la distancia de manhattan, el cual calcula la distancia rectilínea de dos puntos (lago más cercano y el rover) con la suma de la diferencia de coordenadas, respectivamente. A continuación se presenta el código de dicha implementación.

```
# ------ Manhattan Distance ------
def heuristic(self, state):
    state = self.getCoors(state)
    return np.abs(self.closest[0] - np.array(state)[0]) + np.abs(self.closest[1] - np.array(state)[1])

```

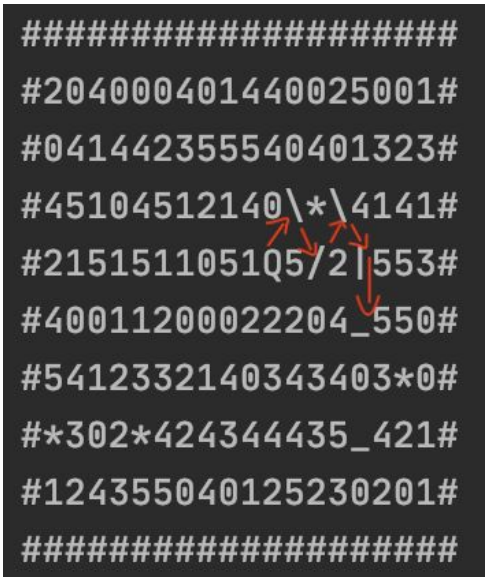
Teniendo la primera heurística, se definió una segunda para comparar el desempeño de ambas en los mismos algoritmos, dicha heurística fue la distancia euclidiana entre el lago más cercano y el rover. El código es el siguiente.

```

def getDistanceFromLake(self, point):
    return ((self.closest[0] - point[0])**2 + (self.closest[1] - point[1])**2)**0.5

# ------ Euclidean Distance ------
def heuristic(self, state):
    state = self.getCoors(state)
    return self.getDistanceFromLake(state)
```

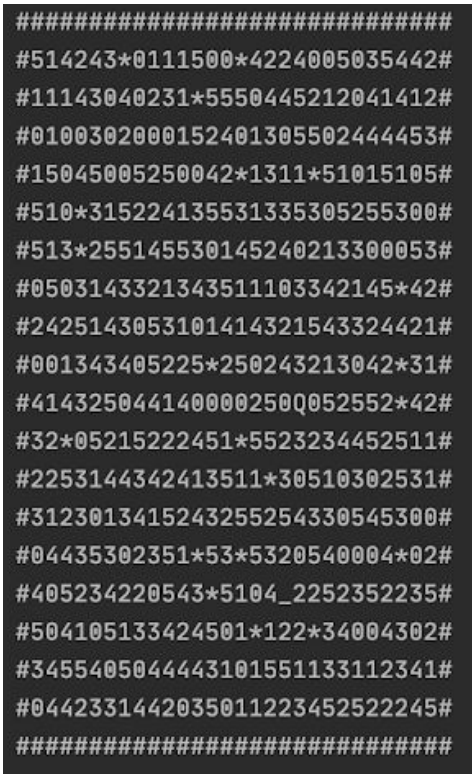
Se implementó un pequeño algoritmo para dibujar el trayecto del rover, en este caso se dibuja la dirección del siguiente paso que tiene que dar el rover, en este caso las direcciones son:

“ | ”: arriba o abajo
“ \ ”: arriba derecha o abajo izquierda
“ / ”: arriba izquierda o abajo derecha

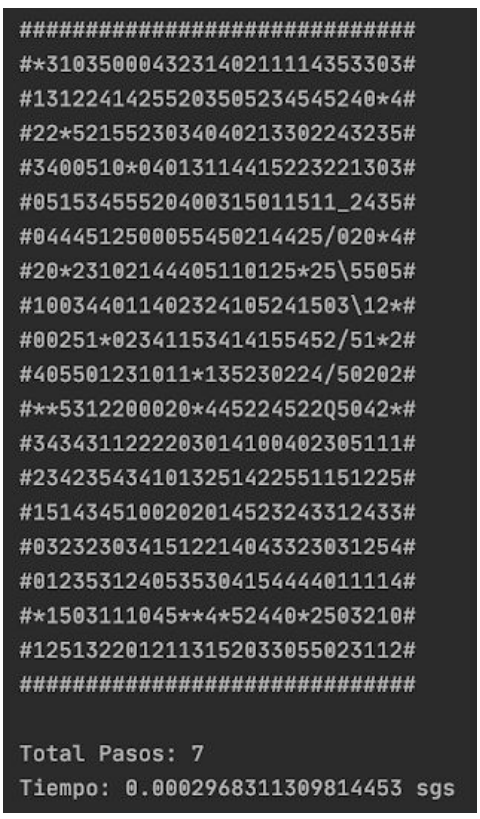
Para que la lectura sea más clara, se puede tomar la siguiente figura como referencia.



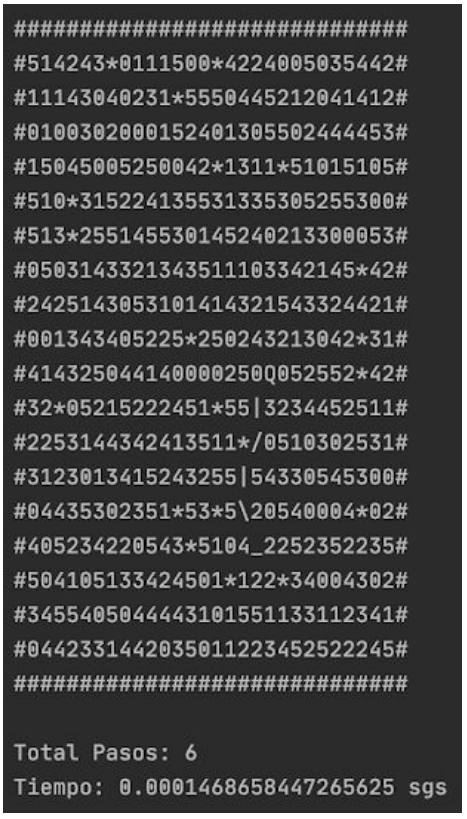
Mapas iniciales



Depth First Search

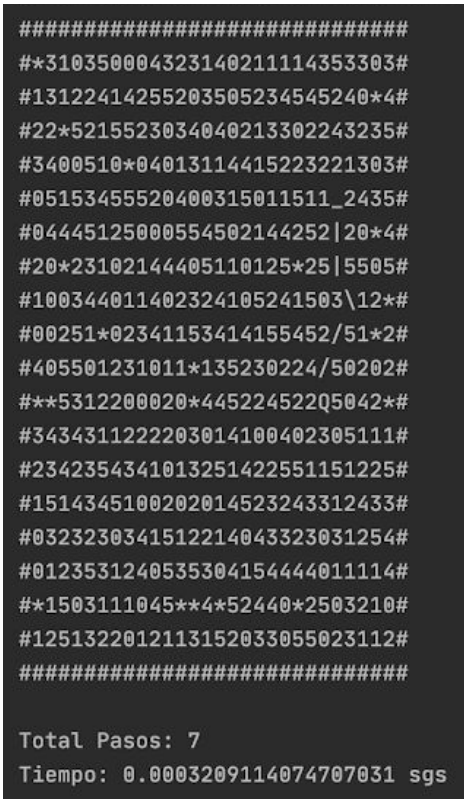


```
##############################
#*3103500043231402111143533O3#
#1312241425520350523454524O*4#
#22*5215523034040213302243235#
#3400510*040131144152232213O3#
#0515345552040031501151l_2435#
#0444512500055450214425/020*4#
#20*23102144405110125*25\5505#
#10034401140232410524150З\12*#
#00251*02341153414155452/51*2#
#405501231011*135230224/50202#
#**5312200020*44522452Q5042*#
#34343112222030141004023051l1#
#23423543410132514225511512Ol#
#151434510020201452324331243З#
#032323034151221404332303125Ч#
#012353124053530415444401111Ч#
#*1503111045**4*52440*2503210#
#125132201211315203305502З112#
##############################

Total Pasos: 7
Tiempo: 0.0002968311309814453 sgs
```

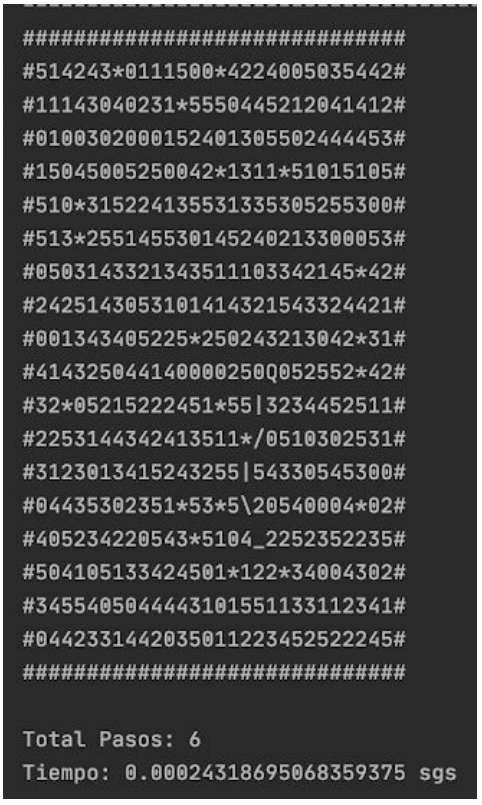


```
##############################
#514243*0111500*4224005035442#
#11143040231*5550445212041412#
#0100302000152401305502444453#
#15045005250042*1311*51015105#
#510*3152241355313353052553OO#
#513*255145530145240213300053#
#05031433213435111033421З5*42#
#24251430531014143215433244Ol#
#001343405225*250243213042*31#
#41432504414000025QQ052552*42#
#32*05215222451*55|3234452511#
#2253144342413511*/0510302531#
#3123013415243255|54330545300#
#04435302351*53*5\20540004*02#
#405234220543*5104_2252352235#
#504105133424501*122*34004302#
#345540504444310155113311234l#
#044233144203501122345252224Ч#
##############################

Total Pasos: 6
Tiempo: 0.0001468658447265625 sgs
```

Breadth First Search



```
##############################
#*3103500043231402111143533O3#
#1312241425520350523454524O*4#
#22*5215523034040213302243235#
#3400510*040131144152232213O3#
#0515345552040031501151l_2435#
#0444512500055450214425Ч|20*4#
#20*23102144405110125*25|5505#
#10034401140232410524150З\12*#
#00251*02341153414155452/51*2#
#405501231011*135230224/50202#
#**5312200020*44522452Q5042*#
#34343112222030141004023051l1#
#23423543410132514225511512Ol#
#151434510020201452324331243З#
#032323034151221404332303125Ч#
#012353124053530415444401111Ч#
#*1503111045**4*52440*2503210#
#125132201211315203305502З112#
##############################

Total Pasos: 7
Tiempo: 0.0003209114074707031 sgs
```



```
##############################
#514243*0111500*4224005035442#
#11143040231*5550445212041412#
#0100302000152401305502444453#
#15045005250042*1311*51015105#
#510*3152241355313353052553OO#
#513*255145530145240213300053#
#05031433213435111033421З5*42#
#24251430531014143215433244Ol#
#001343405225*250243213042*31#
#41432504414000025QQ052552*42#
#32*05215222451*55|3234452511#
#2253144342413511*/0510302531#
#3123013415243255|54330545300#
#04435302351*53*5\20540004*02#
#405234220543*5104_2252352235#
#504105133424501*122*34004302#
#345540504444310155113311234l#
#044233144203501122345252224Ч#
##############################

Total Pasos: 6
Tiempo: 0.00024318695068359375 sgs
```

A*

```
##############################
#*3103500043231402111143533O3#
#1312241425520350523454524O*4#
#22*5215523034040213302243235#
#3400510*040131144152232213O3#
#0515345552040031501151_2435#
#044451250005545021442S2|20*4#
#20*23102144051101Z5*25|5505#
#10034401140232410524150З\12*#
#00251*02341153414155452/51*2#
#405501231011*135230224/50202#
#**5312200020*44522452ZQ5042*#
#34343112222030140100402305111#
#23423543410132514225511512Z5#
#15143451002020145232433124З3#
#03232303415122140433230312S4#
#01235312405353041544440111I4#
#*1503111045**4*52440*2503210#
#12513220121131520330550231I2#
##############################

Total Pasos: 7
Tiempo: 0.0004239082336425781 sgs
```

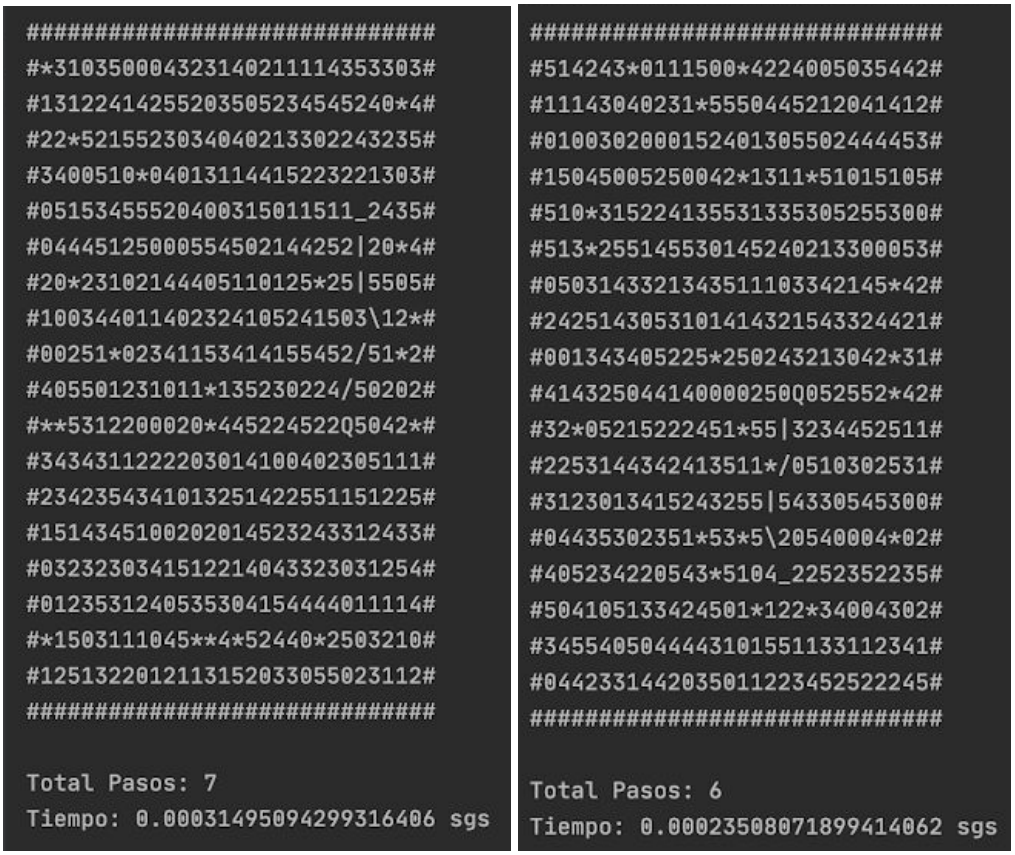
```
##############################
#514243*0111500*4224005035442#
#11143040231*5550445212041412#
#0100302000152401305502444453#
#15045005250042*1311*51015105#
#510*3152241355313353052553OO#
#513*255145530145240213300053#
#05031433213435111033421З5*42#
#24251430531014143215433244Z1#
#01343405225*250243213042*31#
#41432504414000025QQ052552*42#
#32*05215222451*55|3234452511#
#2253144342413511*/0510302531#
#3123013415243255|54330545300#
#04435302351*53*5\20540004*02#
#405234220543*5104_2252352235#
#504105133424501*122*34004302#
#345540504444310155113311234I#
#044233144203501122345252224S#
##############################

Total Pasos: 6
Tiempo: 0.00046324729919433594 sgs
```

Greedy

```
##############################
#*3103500043231402111143533O3#
#1312241425520350523454524O*4#
#22*5215523034040213302243235#
#3400510*040131144152232213O3#
#0515345552040031501151_2435#
#044451250005545021442S/020*4#
#20*23102144051101Z5*25\5505#
#10034401140232410524150З\12*#
#00251*02341153414155452/51*2#
#405501231011*135230224/50202#
#**5312200020*44522452ZQ5042*#
#34343112222030140100402305111#
#23423543410132514225511512Z5#
#15143451002020145232433124З3#
#03232303415122140433230312S4#
#01235312405353041544440111I4#
#*1503111045**4*52440*2503210#
#12513220121131520330550231I2#
##############################

Total Pasos: 7
Tiempo: 0.0003712177276611328 sgs
```

```
##############################
#514243*0111500*4224005035442#
#11143040231*5550445212041412#
#0100302000152401305502444453#
#15045005250042*1311*51015105#
#510*3152241355313353052553OO#
#513*255145530145240213300053#
#05031433213435111033421З5*42#
#24251430531014143215433244Z1#
#01343405225*250243213042*31#
#41432504414000025QQ052552*42#
#32*05215222451*55|3234452511#
#2253144342413511*/0510302531#
#3123013415243255|54330545300#
#04435302351*53*5\20540004*02#
#405234220543*5104_2252352235#
#504105133424501*122*34004302#
#345540504444310155113311234I#
#044233144203501122345252224S#
##############################

Total Pasos: 6
Tiempo: 0.00039386749267578125 sgs
```

Costo uniforme



Análisis de las búsquedas

A simple vista puede observarse que el trayecto del Rover fue el mismo sin importar que se cambiara el algoritmo de búsqueda y, por lo tanto, el comportamiento era igual, pero esta es una suposición incorrecta. En primer lugar, comparemos las búsquedas no informadas (costo uniforme, anchura y profundidad) en la primera corrida. Es importante mencionar que se utilizó el mismo mapa por corrida para facilitar el análisis de los algoritmos y para nuestro análisis usaremos, la mayor parte del análisis, los datos de la primera corrida que se encuentra del lado izquierdo. En la de profundidad, el tiempo que le toma llegar a la solución es muy corto, pero su eficiencia en este aspecto no nos garantiza que encuentre la mejor solución. A diferencia del de anchura, aunque un poco más tardado, encontró una mejor solución que el de profundidad. Valdría la pena recordar que el de anchura es un buen algoritmo de búsqueda mientras sea un problema sencillo, en caso contrario se recomienda recorrir a otros. Si no se nota claramente el Rover se encuentra en el nivel 1 cuando se hace la bifurcación de los dos caminos posibles. Arriba a la derecha tiene un nivel dos y enfrente un nivel 0 antes de llegar al lago. Finalmente el de costo uniforme realiza un movimiento hacia enfrente siempre buscando el

movimiento con menor costo. Sería interesante ver como se comportaría el algoritmo de costo uniforme si el camino no estuviera fuertemente limitado por los niveles ya que usualmente suele haber un único camino posible que siguen todos los alhoritmos informados o no informados. En segundo lugar, las búsquedas informadas tienen un tiempo mayor a las anteriores. De manera extraña las búsquedas no informadas suelen tener una solución mucho antes que las informados. Hablamos de A* y Greedy. Esto se debe a que se guarden los nodos anteriores y cómo los va comparando para buscar la ruta más corta con el menor costo posible y el camino que tiene el menor costo. Aquí nos encontramos en un problema con Greedy. Si se utiliza la lógic, bajar de un nivel 1 al nivel 0 que se encuentra delante y enfrente de este el lago tendría un costo de 2, pero aún con esto el algoritmo Greedy decidió subir al nivel dos dando un movimiento diagonal y luego llegar al lago con otro movimiento diagonal sumando un costo de 3.4. En conclusión, para problemas de este tipo sería ideal utilizar el algoritmo de anchura para búsquedas no informadas y el algoritmo A* para búsquedas informadas.

Código en Github: https://github.com/sebastianneri/searchAlgorithms.git