



# FACULTAD DE CIENCIAS

## INTELIGENCIA ARTIFICIAL

---

### **Agentes inteligentes en laberintos**

---

Equipo: Skynet Scribes

Número de practica: 02

**Carlos Daniel Cortés Jiménez**  
420004846

**Sarah Sophía Olivares García**  
318360638

**Marco Silva Huerta**  
316205326

**Juan Daniel Barrera Holan**  
417079372

**Laura Itzel Tinoco Miguel**  
316020189

Profesora: Cecilia Reyes Peña

Ayudante teoría: Karem Ramos Calpulalpan

Ayudante laboratorio: Tania Michelle Rubí Rojas

**Semestre 2024-2**

Fecha de entrega:  
**21 de Febrero del 2024**

# 1. Documentación

## 1.1. Forma de ejecutar

- Asegurarnos de tener instalada una de las siguientes versiones de python: [ope]

```
1 skynet: $ python --version
2 skynet: $ Python 3.7
3 skynet: $ Python 3.8
4 skynet: $ Python 3.9
5 skynet: $ Python 3.10
```

Esto es para no tener problemas a la hora de trabajar con gym

- Vamos a la carpeta donde se encuentra el archivo `laberinto.py`

```
1 skynet: $ cd Practica02/src
2 skynet: Practica02/src $ ls
3 laberinto.py
```

- Instalar gym

```
1 skynet: Practica02/src $ pip install gym
```

- Para ejecutar usaremos el siguiente comando

```
1 skynet: Practica02/src $ python laberinto.py
```

## 1.2. Código

### OpenAI gym

`gym`: Librería de Python desarrollada por OpenAI, nos permite implementar diferentes algoritmos de aprendizaje por refuerzo y simular la interacción entre agentes y entornos.

`spaces`: Define el formato válido de los espacios de observación y acción para un entorno.

Gym es una biblioteca Python de código abierto para desarrollar y comparar algoritmos de aprendizaje por refuerzo al proporcionar una API estándar para comunicarse entre algoritmos y entornos de aprendizaje, así como un conjunto estándar de entornos compatibles con esa API. [ope]

Los problemas de aprendizaje por refuerzo consisten en el agente y el entorno. El entorno proporciona retroalimentación al agente para que pueda aprender qué acción es apropiada para un estado específico.

## 1.3. Cambios realizados

### Definición del agente

Haciendo uso de la clase **Maze** que hereda de `gym` construimos nuestra clase para el `Agente` y el laberinto. Almacenando al mismo laberinto por su altura y anchura junto con las líneas de movimiento con todos los estados posibles dentro del laberinto, también se inicializa una posición para el agente y una operación que nos indica si se está ejecutando o si ha finalizado.

```
1 # Representar el entorno del laberinto
2 class Maze(gym.Env):
3
4     def __init__(self, maze):
5         super(Maze, self).__init__()
6         # Almacena el laberinto
7         self.maze = maze
8         # Almacena la altura del laberinto
9         self.height = len(maze)
10        # Almacena la anchura del laberinto
11        self.width = len(maze[0])
12        # Espacio de accion del agente: arriba, abajo, izquierda, derecha
13        self.action_space = spaces.Discrete(4)
14        # Espacio de observacion (posibles estados que el agente puede observar del
15        # entorno)
16        self.observation_space = spaces.Discrete(self.height * self.width)
17        self.agent_pos = None # Posicion inicial
18        self.done = False # Indica si el episodio ha terminado (interacciones entre el
19        # agente y el entorno )
```

Listing 1: Definición del agente

El método `reset` básicamente reinicia el laberinto con la finalidad de que en cada llamada el agente se encuentre en su posición inicial correcta. Lo hace llamando al agente y asignándole la posición y terminando la interacción agente laberinto.

## Movimientos del agente

```
1 # Reinicia el entorno a su estado inicial y devuelve la posicion inicial del agente.
2 def reset(self):
3     # Se busca la posicion de inicio en el laberinto
4     self.agent_pos = self.find_start()
5     # Reinicia el estado de finalizacion del episodio
6     self.done = False
7     return self.agent_pos
```

Listing 2: metodo reset

El método `step` (similar al método *mover*) es el encargado de hacer que nuestro agente tome las direcciones arriba, abajo, izquierda, derecha. Primero comprobando el valor de `self.done` para saber si ha terminado, si es así, regresa a la posición actual del agente sin realizar ningún movimiento adicional. Se toman las coordenadas  $x, y$ , que son las actuales del agente, con *action* se verifica que dirección quiere tomar, si es valida actualiza la posición del agente y vuelve verificar pero esta vez si es que la posición es la de salida marcando un `True`. Finalmente, regresa la nueva posición del agente, una recompensa de 0, el estado de finalización del episodio y un diccionario vacío.

```
1 # Toma una accion y se mueve un paso en el entorno.
2 def step(self, action):
3     if self.done:
4         # Si el episodio ya ha terminado, regresamos a la posicion actual
5         return self.agent_pos, 0, True, {}
6
7     x, y = self.agent_pos
8
9     # Movimientos posibles: arriba, abajo, izquierda, derecha
10    if action == 0 and x > 0 and self.maze[x - 1][y] != 1:
```

```

11     print("Movimiento: arriba")
12     x -= 1
13     elif action == 1 and x < self.height - 1 and self.maze[x + 1][y] != 1:
14         print("Movimiento: abajo")
15         x += 1
16     elif action == 2 and y > 0 and self.maze[x][y - 1] != 1:
17         print("Movimiento: izquierda")
18         y -= 1
19     elif action == 3 and y < self.width - 1 and self.maze[x][y + 1] != 1:
20         print("Movimiento: derecha")
21         y += 1
22
23     self.agent_pos = (x, y)
24
25     # Verifica si la posicion en la que se encuentra es la salida
26     if self.maze[x][y] == "S":
27         self.done = True
28
29     return self.agent_pos, 0, self.done, {}

```

Listing 3: metodo step

Con este método vamos iterando por lo alto y ancho del laberinto para la búsqueda de la posición inicial del agente, ya que nos adelantamos a que no siempre será la posición (0,0), de esta forma el agente inicia en la posición correcta el laberinto cada vez que se llama a reset, además garantiza que exista un punto de inicio.

```

1 # Encuentra la posicion inicial del agente (entrada del laberinto).
2 def find_start(self):
3     for i in range(self.height):
4         for j in range(self.width):
5             if self.maze[i][j] == "E":
6                 return (i, j)
7     raise ValueError("No se pudo encontrar el punto de inicio E en el laberinto")

```

Listing 4: metodo find start

## El algoritmo

Ahora si la carne dentro del código, la salida usando backtracking. el método solve comienza por definir los 4 movimientos del agente por las tuplas de las coordenadas

```

1 # Encuentra la salida usando backtracking
2 def solve(env):
3     # arriba, abajo, izquierda, derecha
4     actions = [(0, -1), (0, 1), (-1, 0), (1, 0)]

```

Listing 5: Movimientos del agente

Ya definidos los movimientos pasamos a verificar que nuestro agente no traspasará muros o hará un recorrido al puro estilo pacman:

- Primero verificamos si las coordenadas están fuera de los límites del laberinto.
- Si no hemos topado con un muro
- Si el agente ya ha pasado por esta celda en la ruta actual ((x, y) in path)

```

1 def backtrack(x, y, path):
2     if x < 0 or x >= env.height or y < 0 or y >= env.width or env.maze[x][y] ==
1 or (x, y) in path:
3         return False
4
5     path.append((x, y))

```

Listing 6: Validación de movimientos

Después vamos a verificar si la poción es la salida. Si es así imprimimos que se ha encontrado y se devuelve True para indicar que se ha encontrado la salida.

Recursivamente llamamos a `backtrack` para iterar sobre todos los movimientos posibles en la posición actual. Busca los caminos y si no llega a la salida retrocede con el `path.pop()`. Y como antes lo explicábamos en el método buscamos la poción inicial, la usamos para encontrar la  $x$  y  $y$  y se las pasamos al método `backtrack`, de esta forma de itera para ir imprimiendo la ruta que siguió el agente o si no hay solución nos lo hace saber el programa.

```

1     if env.maze[x][y] == "S":
2         print("Encontre la salida :D")
3         return True
4
5     for action in range(env.action_space.n):
6         if backtrack(x + actions[action][0], y + actions[action][1], path):
7             return True
8
9     path.pop()
10    return False
11
12    start_x, start_y = env.find_start()
13    path = []
14    if backtrack(start_x, start_y, path):
15        print("La ruta es:")
16        for i, pos in enumerate(path):
17            x, y = pos
18            direction = ""
19            if i > 0:
20                # Comparar la posicion actual con la anterior para determinar la
21                direccion
22                prev_pos = path[i - 1]
23                if x < prev_pos[0]:
24                    direction = "arriba"
25                elif x > prev_pos[0]:
26                    direction = "abajo"
27                elif y < prev_pos[1]:
28                    direction = "izquierda"
29                elif y > prev_pos[1]:
30                    direction = "derecha"
31                print(f"Posicion: {pos}, Direccion: {direction}")
32    else:
33        print("iiiiii no encuentre la salida")

```

Listing 7: algoritmo backtracking

Vamos a ver más de cerca esta línea: Tenemos una llamada recursiva a `backtrack`, y nos posicionamos en  $x$  y  $y$  coordenadas actuales donde `actions` hace los movimientos correspondientes al momento de hacer volver a llamar al método le estamos pasando esas coordenadas y la lista de posiciones visitadas hasta el momento. La condicional verifica la llamada, pues si es True ya se encontró la salida pero si devuelve un False, no ha encontrado la salida y lo que hace es retroceder eliminando la posición de la lista.

```
1 if backtrack(x + actions[action][0], y + actions[action][1], path):
```

Listing 8: algoritmo backtracking

## Referencias

- [RN16] Stuart Russell y Peter Norvig. *Inteligencia Artificial Un Enfoque moderno*. 2nd. Pearson Prentice Hall, 2016.
- [Pan21] Thanakorn Panyapiang. *Developing Reinforcement Learning Environment Using OpenAI Gym*. <https://medium.com/geekculture/developing-reinforcement-learning-environment-using-openai-gym-f510b0393eb7>. 2021.
- [RAN21] ASHISH RANA. *Introduction: Reinforcement Learning with OpenAI Gym*. <https://towardsdatasci.com/reinforcement-learning-with-openai-d445c2c687d2>. 2021.
- [Ref21] Programación Refactoriza. *Backtracking*. <https://docs.jjpeleato.com/algoritmia/backtracking>. 2021.
- [ope] openai. *gym*. <https://github.com/openai/gym?tab=readme-ov-file>.