



**CENTRO DE CIENCIAS BÁSICAS**  
**DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN**  
**INTELIGENCIA ARTIFICIAL**  
**3° "A"**

**PROYECTO/EXAMEN: EVOLUCIÓN DE UN AGENTE INTELIGENTE PARA  
APRENDER A PASAR UN LABERINTO ESPECÍFICO**

**Profesor: Miguel Ángel Meza de Luna**

**Alumnos:**  
**Espinoza Sánchez Joel Alejandro**  
**Reyes González Andrés Eleazar**  
**Ortíz Nájera Alan Daniel**

**Fecha de Entrega:** Aguascalientes, Ags., **30** de septiembre de 2019

# Índice

Antecedentes y Contexto Teórico-----	2
La Ecuación de Bellman-----	2
Aprendizaje por Refuerzo -----	2
Agente Tonto -----	4
Agente Base: Ecuación de Bellman -----	7
Agente Propio -----	11
Evolución -----	11
El Agente Propio y su Funcionamiento-----	18
Curso de Inteligencia Artificial con Python: Sección 5 -----	21
Curso de Inteligencia Artificial con Python: Sección 6 -----	28
Bibliografía-----	34
Anexos-----	35
Primera Parte: Anexos Internos al Documento -----	35
Segunda Parte: Anexos Externos al Documento-----	39

## Antecedentes y Contexto Teórico

La inteligencia artificial se suele relacionar a menudo con la ciencia ficción, sin embargo, ya no se encuentra relegada a las novelas y las películas. “Esta tecnología nos rodea, desde los lugares más cotidianos (conversión de voz en texto, etiquetado de fotografías, detección del fraude) a los más punteros (medicina de alta precisión, predicción de lesiones, coches autónomos)” (Iglesias, 2016). Se encuentra en métodos informáticos como el análisis avanzado de datos, la visión por ordenador, el procesamiento de lenguaje natural y el Aprendizaje Automático o Machine Learning.

Y es que, Iglesias (2016) informa que, como explicaba recientemente Diane Bryant, vicepresidenta ejecutiva y directora general del Data Center Group de Intel, la inteligencia artificial está transformando la forma de trabajar de las empresas, así como nuestra manera de interactuar con el mundo.

### La Ecuación de Bellman

La Ecuación de Bellman, también conocida como la ecuación de programación dinámica, nombrada en honor de su descubridor, Richard Bellman, es “una condición necesaria para la optimalidad asociada con el método de la optimización matemática conocida como programación dinámica” (Iglesias, 2016). Se escribe el valor de un problema de decisión en un determinado punto en el tiempo en términos de la recompensa que dan algunas opciones iniciales y el valor del problema de decisión restante que resulta de esas opciones iniciales.

### Aprendizaje por Refuerzo

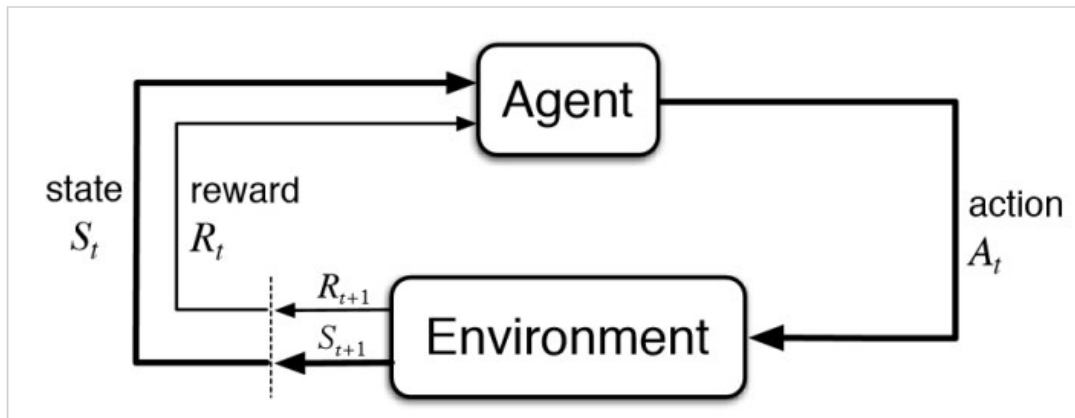
Para simular el aprendizaje de sistemas biológicos reales se necesitan hacer algunas suposiciones que simplifican el comportamiento de nuestros agentes (o aprendices). Estas simplificaciones permiten tener un sistema más flexible para proyectar diversas situaciones con el mismo sistema y, a la vez, extraer conclusiones más generales acerca de las propiedades de los algoritmos que implementen estos sistemas de aprendizaje.

En general, y como primera aproximación, Sancho (2019) comenta que se supondrá que los aprendices siguen un Proceso de Decisión de Markov, es decir:

- El agente percibe un conjunto finito,  $S$  de estados distintos en su entorno, y dispone de un conjunto finito,  $A$  de acciones para interactuar con él.
- El tiempo avanza de forma discreta, y en cada instante de tiempo,  $t$ , el agente percibe un estado concreto,  $st$ , selecciona una acción posible,  $at$ , y la ejecuta, obteniendo un nuevo estado,  $st + 1 = at(st)$ .
- El entorno responde a la acción del agente por medio de una recompensa, o castigo.

Este estímulo se formalizará por medio de un número, de forma que cuanto mayor es, también el beneficio lo será.

Tanto la recompensa como el estado siguiente obtenido no tienen por qué ser conocidos a priori por el agente y dependen únicamente del estado actual y de la acción tomada. Es decir, antes de aprender, el agente no sabe qué pasará cuando toma una acción determinada en un estado particular. Precisamente, un buen aprendizaje es aquel que permite adelantarse al agente en las consecuencias de las acciones tomadas, reconociendo las acciones que sobre estados concretos le llevan a conseguir con más eficacia y mayores recompensas, sus objetivos.



# Agente Tonto

Este agente fue elaborado en el lenguaje de programación C. Las bases de este se establecieron con una matriz la cual sería el espacio de movimiento del agente. Asimismo, los movimientos se darían por medio de un proceso aleatorio realizado internamente por la computadora que arrojaría resultados de 1 a 4.

## Las Variables

Se manejaron las siguientes variables (véase anexo 1.01):

- **tablero:** Una matriz de 4 renglones por 3 columnas de tipo carácter que será el espacio en el que se moverá el agente. Hay que aclarar que, el documento refiere a las coordenadas de esta matriz repetidamente, pero en función de cómo las interpreta un programa, pues, aunque el ser humano interpreta una matriz de n renglones empezando desde 1, la computadora lo hace desde 0, por lo que toda notación de coordenadas se redactó con base en la interpretación computacional.
- **random:** Un número entero que en el transcurso de la ejecución se le asignarán valores aleatorios entre 1 y 4. Cada uno representa un posible movimiento que el agente podrá realizar.
- **x:** Un número entero que guardará la posición de las columnas en las que el agente se encuentra, iniciando en 0 pues es la coordenada de inicio del agente.
- **y:** Un número entero que guardará la posición de los renglones en las que el agente se encuentra, iniciando en 0 pues es la coordenada de inicio del agente.
- **camino:** Una variable tipo “string” que guardará el recorrido del agente por el laberinto.
- **ganador:** Una variable booleana que comienza en falso, pues cambiará de estado a verdadero cuando llegue a la casilla ganadora.
- **vivo:** Una variable booleana que comienza en verdadero, pues cambiará de estado a falso cuando llegue a la casilla que se espera que el agente evite.

La programación del agente comienza con una función de tipo void llamada inicio, la cual se encarga de llenar la matriz de un carácter de espacio, por cuestiones de estética. Igualmente, en la posición (0,0) se inserta, en lugar de un espacio, un carácter ‘O’ que representará al agente. En la posición (1,1) se insertará el valor ‘Ø’, pues no le es permitido al agente pasar por dicha casilla. Finalmente, en las posiciones (3,1) y (3,2) se insertaron los valores de ‘X’ y ‘V’, simbolizando la derrota y la victoria respectivamente (véase anexo 1.02) que, en la ejecución del programa, el comienzo se ve de la siguiente forma:

[O]	[ ]	[ ]
[ ]	[θ]	[ ]
[ ]	[ ]	[ ]
[ ]	[X]	[V]
θ,θ		

Una vez que la matriz fue arreglada con la simbología ya mencionada, se imprime el comienzo de la matriz e instantáneamente el programa entra en un ciclo que se repetirá mientras el agente siga vivo o todavía no haya ganado (por ello, las variables booleanas).

Al comienzo de cada iteración, el programa genera un número aleatorio entre 1 y 4 que se guardará en la variable random. Cada número simboliza un posible movimiento del agente:

1. El agente se mueve hacia arriba.
2. El agente se mueve hacia la derecha.
3. El agente se mueve hacia abajo.
4. El agente se mueve hacia la izquierda.

Nótese que, por convención, se eligió comenzar como un reloj e igualmente, dar los valores en un sentido horario.

Inmediatamente después de haber generado un número aleatorio, el programa debe verificar que el agente no vaya a moverse en una posición no permitida, es decir, el programa valida las restricciones de movimiento, por ejemplo, si se encuentra en el primer renglón, el agente no debería poder moverse hacia arriba, porque no hay casillas más arriba que en la que se encuentra. Igualmente, con la casilla de la posición (1,1) la cual no se le permite el acceso al agente (véase anexo 1.03).

Después de que el programa pasa esta primera etapa de verificación, realiza una segunda verificación, para asegurarse si ya ha ganado o ya ha perdido, pues verifica si las coordenadas del agente – dadas por las variables x y y – son iguales a alguna equivalente de estas casillas de derrota o victoria. En caso de entrar a alguna, el programa cambia el estado booleano de alguna de las variables booleanas mencionadas en el apartado de las variables y después termina el ciclo.

Sin embargo, en caso de no haber llegado a estas casillas, el programa ya habrá verificado dos aspectos muy importantes del tablero. Ahora lo que debe hacer en caso de que ninguna de las verificaciones anteriores haya interrumpido al ciclo es hacer al agente que avance en el tablero. Para ello, se escribieron unos condicionales que leen el número que la variable random tiene guardado y dependiendo del número, añade o resta uno a alguna variable, ya sea x o y y

también agrega a la variable camino las coordenadas por las que pasó el agente (véase anexo 1.04).

Finalmente imprime y borra la pantalla para no acumular tableros en la misma y una vez que se rompa el ciclo, el programa habrá terminado (véase anexo 2.01). A continuación, se presentan dos ejemplos en los que el agente gana y pierde respectivamente:

```
[ ] [ ] [ ]
[ ] [0] [ ]
[ ] [ ] [ ]
[ ] [X] [0]

0,0 0,1 0,0 1,0 2,0 1,0 2,0 2,1 2,2 2,3
GANÓ
```

```
[ ] [ ] [ ]
[ ] [0] [ ]
[ ] [ ] [ ]
[ ] [0] [V]

0,0 1,0 2,0 1,0 0,0 1,0 2,0 2,1 2,2 1,2 2,2 2,1 2,0 2,1 2,2 1,2 0,2 0,3 0,2 1,2 1,3
MURIÓ
```

## Agente Base: Ecuación de Bellman

Se resolvió el entorno FrozenLake desde gym OpenAI (proporciona una manera fácil para que los individuos experimenten con sus agentes de aprendizaje en una variedad de juegos de juguete proporcionados).

```
import gym
import numpy as np
env = gym.make('FrozenLake-v0')
```

El entorno FrozenLake consiste en una cuadrícula 4x4 de bloques, cada uno de los bloques son:

- El bloque de inicio.
- El bloque de objetivos.
- Un bloque congelado seguro.
- Un agujero peligroso.

El objetivo es que un agente aprenda a navegar desde el principio hasta la meta sin pasar a un agujero. En un momento dado, el agente puede elegir moverse hacia arriba, abajo, izquierda o derecho. Como tal, el rendimiento perfecto cada vez es imposible, pero aprender a evitar los agujeros y alcanzar la meta es sin duda todavía factible. La recompensa en cada paso es 0, excepto para entrar en la meta, lo que proporciona una recompensa de 1.

Se implementó una tabla de valores para cada estado (fila) y acción (columna) posible en el entorno. Dentro de cada celda de la tabla según Juan Garbiel Gomila (2019), aprendemos un valor de que tan bueno es tomar una acción dada dentro de un estado determinado. En el caso del entorno FrozenLake, tenemos 16 estados posibles (uno para cada bloque), y 4 acciones posibles (las cuatro direcciones de movimiento), lo que nos da una tabla de valores Q de 16x4. Comenzamos inicializando la tabla para que sea uniforme (todos los ceros), y luego, al observar las recompensas que obtenemos por varias acciones, actualizamos la tabla en consecuencia.

```
Q = np.zeros([env.observation_space.n, env.action_space.n])
```

Se hicieron actualizaciones a nuestra tabla Q usando la ecuación de Bellman, que establece que la recompensa esperada a largo plazo por una acción dada es igual a la recompensa inmediata de la acción actual combinada con la recompensa esperada de la mejor acción futura tomada en el siguiente estado. De esta manera, se reutiliza nuestra propia tabla Q al estimar cómo actualizar nuestra tabla para futuras acciones

$$Q[s,a] = Q[s,a] + \alpha(r + \gamma \max_{a'} Q[s1,a'] - Q[s,a])$$

Esto dice que el valor Q para un estado (s) y una acción (a) determinados deben representar la recompensa actual (r) más la recompensa máxima descontada ( $\gamma$ ) recompensa futura esperada de acuerdo con nuestra propia tabla para el siguiente estado (s1) que terminaríamos. La variable de descuento nos permite decidir la importancia de las posibles recompensas futuras en comparación con



la recompensa actual. Al actualizar de esta manera, la tabla comienza lentamente a obtener medidas precisas de la recompensa futura esperada para una acción determinada en un estado dado.

El código, según Gomila (2019) es el siguiente:

```
import gym
import numpy as np
env = gym.make('FrozenLake-v0')

#Inicializa la tabla en ceros.
Q = np.zeros([env.observation_space.n,env.action_space.n])

#Se establecen los parametros de aprendizaje

lr = .8
y = .95
num_episodes = 2000

#Se crea una lista que contenga el total de recompensas y pasos por episodio
#jList = []

rList = []
for i in range(num_episodes):

    #Se resetea el entorno y obtiene una nueva observacion

    s = env.reset()
    rAll = 0
    d = False
    j = 0

    #El algoritmo de aprendizaje de la Tabla Q
    while j < 99:
        j+=1

        #Se escoge la mejor accion por el metodo greedily

        a = np.argmax(Q[s,:] + np.random.randn(1,env.action_space.n)*(1./(i+1)))

        #Se obtiene un nuevo estado y una recompensa del entorno

        s1,r,d,_ = env.step(a)

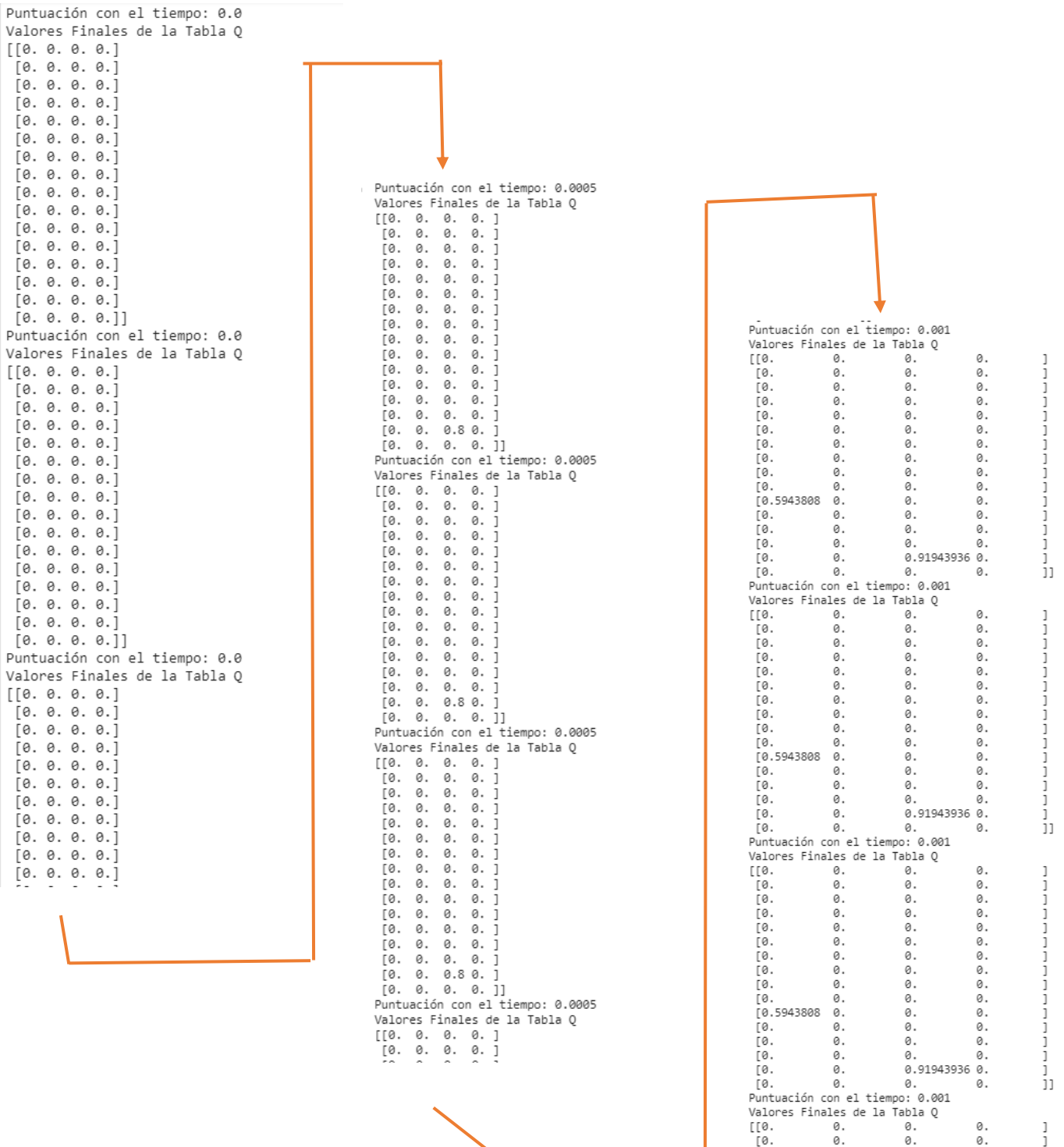
        #Actualiza la Tabla Q con nuevos conocimientos

        Q[s,a] = Q[s,a] + lr*(r + y*np.max(Q[s1,:]) - Q[s,a])
        rAll += r
        s = s1
        if d == True:
            break

    rList.append(rAll)

print ("Puntuación con el tiempo: " + str(sum(rList)/num_episodes))
print ("Valores Finales de la Tabla Q")
print (Q)
```

Y así son las actualizaciones en su ejecución:



De modo que continúa hasta llegar a lo siguiente:

```

Puntuación con el tiempo: 0.3865
Valores Finales de la Tabla Q
[[1.32304372e-01 5.26287522e-03 6.91246448e-03 5.43186717e-03]
[2.79295107e-07 1.02757821e-04 7.12677821e-04 1.32895057e-01]
[3.33368928e-03 0.00000000e+00 1.12512330e-03 9.41505874e-02]
[2.12449505e-05 8.19603790e-05 8.82502860e-05 5.38050117e-02]
[1.64656595e-01 2.06022744e-04 2.03739213e-03 8.03617563e-06]
[0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00]
[6.98480647e-02 1.13201960e-04 2.39618070e-04 2.70190708e-06]
[0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00]
[5.54176401e-04 3.93257063e-04 3.16055634e-03 5.29612224e-01]
[1.60212315e-05 6.95401797e-01 5.87126632e-04 1.27983093e-04]
[1.73182106e-01 4.99591241e-04 0.00000000e+00 4.98783121e-04]
[0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00]
[0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00]
[0.00000000e+00 0.00000000e+00 6.03802113e-01 0.00000000e+00]
[0.00000000e+00 0.00000000e+00 5.98660532e-01 0.00000000e+00]
[0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00]]
Puntuación con el tiempo: 0.3865
Valores Finales de la Tabla Q
[[1.47613465e-01 5.26287522e-03 6.91246448e-03 5.43186717e-03]
[2.79295107e-07 1.02757821e-04 7.12677821e-04 1.32895057e-01]
[3.33368928e-03 0.00000000e+00 1.12512330e-03 9.41505874e-02]
[2.12449505e-05 8.19603790e-05 8.82502860e-05 5.38050117e-02]
[4.35436610e-01 2.06022744e-04 2.03739213e-03 8.03617563e-06]
[0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00]
[1.39696129e-02 1.13201960e-04 2.39618070e-04 2.70190708e-06]
[0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00]
[5.54176401e-04 3.93257063e-04 3.16055634e-03 4.61969902e-01]
[1.60212315e-05 2.62550096e-01 5.87126632e-04 1.27983093e-04]
[9.75177589e-02 4.99591241e-04 0.00000000e+00 4.98783121e-04]
[0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00]
[0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00]
[0.00000000e+00 0.00000000e+00 6.03802113e-01 0.00000000e+00]
[0.00000000e+00 0.00000000e+00 5.98660532e-01 0.00000000e+00]
[0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00]]
Puntuación con el tiempo: 0.387
Valores Finales de la Tabla Q
[[3.55003821e-01 5.26287522e-03 6.91246448e-03 5.43186717e-03]
[2.79295107e-07 1.02757821e-04 7.12677821e-04 1.32895057e-01]
[3.33368928e-03 0.00000000e+00 1.12512330e-03 9.41505874e-02]
[2.12449505e-05 8.19603790e-05 8.82502860e-05 5.38050117e-02]
[4.38184447e-01 2.06022744e-04 2.03739213e-03 8.03617563e-06]
[0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00]
[1.39696129e-02 1.13201960e-04 2.39618070e-04 2.70190708e-06]
[0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00]
[5.54176401e-04 3.93257063e-04 3.16055634e-03 2.91932053e-01]
[1.60212315e-05 5.11399625e-01 5.87126632e-04 1.27983093e-04]
[9.75177589e-02 4.99591241e-04 0.00000000e+00 4.98783121e-04]
[0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00]
[0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00]
[0.00000000e+00 0.00000000e+00 5.70912010e-01 0.00000000e+00]
[0.00000000e+00 0.00000000e+00 9.19732106e-01 0.00000000e+00]
[0.00000000e+00 0.00000000e+00 0.00000000e+00 0.00000000e+00]]

```

Puede observarse que, en cada caso, la posición en la que el agente pueda estar, él analizará y avanzará a la opción que tenga el peso mayor y así seguirá guardando la información de ese estado para poder compararlo con los otros y obtener la recompensa de una manera más precisa (véase anexo 2.02).

## Agente Propio

Para la realización de este agente, se tomó de base el agente tonto realizado en C, pues este también se hizo en C, haciendo ciertas modificaciones al mismo para poder llegar al objetivo esperado.

Evolución: ¿Cómo se modificó el Agente Tonto y qué recursos se tomaron del Agente Base para obtener el Agente Propio?

### Versión 0.1

Era evidente que, para implementar valores estocásticos a cada movimiento de una casilla, sería necesario visualizar dicha casilla y los cuatro valores estocásticos que cada una tenía, y como el agente pasado realizado en Python plasmaba los valores de cada movimiento en forma de cruz, se pensaba hacer un entorno visual similar a aquel, por lo que el primer modelo gráfico fue hacer que cada casilla se viera similar a lo siguiente:



Sin embargo, debido a la complejidad para realizar esta visualización, se optó en la primera codificación por hacer la siguiente visualización (véase anexo 2.03):

0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000
0.000		0.000		0.000	
0.000				0.000	
0.000	0.000			0.000	0.000
0.000				0.000	
0.000	0.000	0.000	0.000	0.000	0.000
0.000		0.000		0.000	
0.000		0.000		0.000	
0.000	0.000	-1.000	1.000	1.000	1.000
0.000	0.000	-1.000	-1.000	1.000	1.000
0.000		-1.000		1.000	

### Versión 0.5

En el siguiente momento de codificación, se implementó este “mapa” al agente tonto (véase anexo 2.04), lo cual generaba el mismo funcionamiento del agente tonto, pero ahora con el siguiente aspecto:

```

      0.000      0.000      0.000
0.000 0.000 0.000 0.000 0.000 0.000
      0.000      0.000      0.000

      0.000      0.000      0.000
0.000 0.000      X      0.000 0.000
      0.000      0.000      0.000

      0.000      0.000      0.000
0.000 0.000 0.000 0.000 0.000 0.000
      0.000      0.000      0.000

      0.000      -1.000      1.000
0.000 0 0.000 -1.000 -1.000 1.000 1.000
      0.000      -1.000      1.000

0,0 0,1 0,2 0,1 0,2 0,3 _

```

### Versión 1.0

Sin embargo, la percepción de los movimientos del agente era muy complicada debido a la cantidad de información desplegada en pantalla, además que el agente es un carácter 'O' y frente a tantos caracteres similares, el agente se perdía ante los ojos de las personas, por lo que se realizó una nueva prueba con el agente dando color al nuevo programa (véase anexo 2.05), lo que hizo que la función impr se volviera extremadamente complicada de entender (véase anexo 1.05), pero ahora, su ejecución mostraba el siguiente mapa:

```

      0.000      0.000      0.000
0.000 0.000 0.000 0.000 0.000 0.000
      0.000      0.000      0.000

      0.000      0.000      0.000
0.000 0.000      0.000 0.000
      0.000      0.000      0.000

      0.000      0.000      0.000
0.000 0.000 0.000 0.000 0.000 0.000
      0.000      0.000      0.000

      0.000      -1.000      1.000
0.000 0.000 -1.000 -1.000 1.000 1.000
      0.000      -1.000      1.000

0,0 1,0 0,0 1,0 2,0 1,0 2,0 1,0 2,0 _

```

### Versión 1.5

En el siguiente avance de código, se implementó el tablero con una variable struct que se le denominaría t con la misma propiedad de ser una matriz y que cada elemento del struct tuviera cinco campos:

- `up`: Una variable de tipo float que guardará el valor de probabilidad de dicha casilla para moverse hacia arriba.
- `right`: Una variable de tipo float que guardará el valor de probabilidad de dicha casilla para moverse hacia la derecha.
- `down`: Una variable de tipo float que guardará el valor de probabilidad de dicha casilla para moverse hacia abajo.
- `left`: Una variable de tipo float que guardará el valor de probabilidad de dicha casilla para moverse hacia la izquierda.
- `info`: Una variable de tipo carácter que guardará el carácter con el mismo comportamiento que la matriz de caracteres `tablero`.

Asimismo, se optimizó la función para imprimir el tablero en un ciclo y ahorrarse algunas líneas de código, lo que cambió un poco la estética del mapa, añadiendo los números de probabilidad de ese espacio “vacío” de la casilla a la que el agente no podía acceder, sin embargo, el comportamiento del agente seguía siendo el mismo, pues este avance se hizo para verificar que la implementación del `struct` haya sido correcta (véase anexo 2.06).

## Versión 2.0

Después del intento anterior, volvió a modificarse el código y en la siguiente modificación se volvieron a presenciar cambios grandes en el algoritmo del agente, pues la solución estaba muy cerca. Algunos cambios menores que ocurrieron fue la implementación en su totalidad de la variable `struct` para que el agente ahora navegara en esta variable (sin embargo, la variable `tablero` todavía no se eliminaba), de igual forma se agregó otra variable `struct` llamada `coord` que tendría como campos los siguientes:

- `x`: Una variable entera que cumple la misma función que el entero simple `x` planteado en versiones anteriores.
- `y`: Una variable entera que cumple la misma función que el entero simple `y` planteado en versiones anteriores.
- `dir`: Una variable entera que guarda la dirección del agente en cierta posición.

Esta variable, a su vez, sería un vector de tamaño 50 que guardaría estos tres valores de los primeros 50 movimientos del agente, para interpretarlos en el verdadero gran cambio de esta versión.

La implementación más notoria de esta versión fue la nueva función añadida llamada `actualizavalores`. Esta función manipula los valores probabilísticos de cada casilla y por primera vez en la realización del presente, el programa hacía estas modificaciones de valores para el uso futuro de los mismos.

Dicha función asigna un valor a una variable llamada `a` según el resultado final del proceso. Si fue un resultado positivo, añade a `a` un valor de 0.9, de lo contrario, le asigna el valor 0.1 (esto únicamente como prueba del funcionamiento del programa). Después, toma los datos guardados en la

variable tipo coord para ubicar a qué casilla le cambiará el valor y a qué dirección de esta (debido a que esta variable cumple la misma función que la variable camino, fue reemplazada y eliminada por esta estructura). La modificación del valor se da tras encontrar el valor máximo de los cuatro que constituyen a una casilla y multiplicarlo por a (véase anexo 1.06).

Puede observarse que, tras haber ejecutado esta versión una vez y que el agente haya ganado tras cierto recorrido, el mapa se modifica de la siguiente manera (véase anexo 2.07):

0.000	0.000	0.000	0.000	0.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000
0.590		0.000		0.729	
0.000	0.000	0.000	0.000	0.656	
0.000	0.000	0.000	X 0.000	0.000	0.000
0.656		0.000		0.810	
0.590		0.000		0.729	
0.000	0.729	0.656	0.810	0.000	0.000
0.590		0.000		0.900	
0.656		0.100		1.000	
0.000	0.000	0.100	0.100	1.000	0 1.000
0.000		0.100		1.000	

## Versión 2.5

Es importante aclarar que esta versión ya apuntaba a ser una de las versiones finales, pues se esperaba que la misma repitiera el código repetidas veces, es decir, es la primera versión de las realizadas en C que permitía el contacto con el usuario, permitiéndole teclear cuántos agentes evaluarían el terreno. Aunque se implementó dicho algoritmo, el proceso fracasó y tras la primera victoria o muerte del agente, la ejecución dejaba de funcionar. Esta versión tuvo algunas estilizaciones menores, pero se presenta por si se quiere revisar los primeros intentos por dar el siguiente gran paso para llegar a la versión final (véase anexo 2.08).

## Versión 2.8

Junto a otras dos correcciones, esta versión tuvo cambios en la lectura del código, así como trató de optimizarse algunas secciones, como eliminando finalmente variables que no se usaban (cabe aclarar que ciclos o procesos que podían acortarse a menor cantidad de líneas no fueron optimizados) pero ya se había logrado corregir los errores por los que no ejecutaba más de un agente. Para el punto actual, el programa ya pedía datos (véase anexo 1.07) para permitir una amplia libertad al usuario y poder calibrar el agente con datos como la cantidad de agentes, el descuento otorgado a cada casilla y la velocidad de ejecución (véase anexo 2.09).

```

Cantidad de agentes: 10
Descuento: 0.5
Velocidad del agente (milisegundos): 500

```

## La Versión Alfa

Con el agente inteligente casi en los últimos detalles de modificación para tener la solución al problema, finalmente se realizó una lluvia de ideas para darse una mínima idea de cómo el agente inteligente interpretaría los datos, pues en las charlas de equipo, se buscaba que el agente tuviera, incluso sabiendo que le convenía un camino determinado, algún proceso estocástico y que también tuviera posibilidades, aunque menores, de tomar el camino con menos probabilidad, por lo que se planteó lo siguiente:

Supóngase que el agente está en una casilla, la cual se visualiza así:

```

      0.000
    0.000  0  0.000
      0.000

```

Lo anterior puede verse, para generalizar los valores como un número distinto según la dirección que el agente pueda tomar. Sean  $a$ ,  $b$ ,  $c$ ,  $d$  los números que cada dirección de dicha casilla tiene guardado en ese momento. Lo anterior tiene la siguiente visualización:

```

      a
    d  0  b
      c

```

Los valores anteriores pueden verse en una recta real que inicia en cero.

```

|-----|
0

```

Donde cada valor anterior puede sumarse al siguiente, es decir,  $a$  se sumará a  $b$ , este se sumará a  $c$  y este a  $d$  para obtener la siguiente recta real:

```

|-----|-----|-----|-----|
0      a      a+b    a+b+c  a+b+c+d

```



Esta recta real tiene una interpretación muy peculiar. Dado el número  $a$ , es decir, el valor estocástico que ya se le asignó a esta casilla de ir hacia arriba, el cual en este modelo para generar probabilidad se interpreta como una distancia entre cero y el mismo valor, que dicha distancia es  $a$ . De igual forma, para  $b$ , si se le suma  $a$ , la distancia entre  $a$  y  $a + b$  es  $b$ . Para  $c$ , si se le suma  $a + b$ , la distancia entre  $a + b$  y  $a + b + c$  es  $c$ . Finalmente, para  $d$ , si se le suma  $a + b + c$ , la distancia entre  $a + b + c$  y  $a + b + c + d$  es  $d$ . De esta manera, puede generarse un número aleatorio decimal entre cero y la suma  $a + b + c + d$  y así, la dirección más probable tendrá más probabilidad de que esta generación aleatoria caiga en la distancia mayor, que es equivalente al valor de dirección más probable.

La interpretación anterior se agregó a esta versión del código en una nueva función llamada `aleatorio`, la cual hace estas sumas y encuentra en qué “intervalo” de la recta generada cayó el número generado por computadora.

```
int aleatorio()
{
    srand(time(NULL));

    int pup,pright,pdown,pleft,alea;
    float total,variable;
    total=t[y][x].up+t[y][x].right+t[y][x].down+t[y][x].left;
    pup=(t[y][x].up*100)/total;
    pright=(t[y][x].right*100)/total;
    pdown=(t[y][x].down*100)/total;
    pleft=(t[y][x].left*100)/total;
    variable=0+rand()%(101);
```

El código anterior era prácticamente la versión final, excepto por el detalle de que, cuando alguna casilla se encontraba en cero, la función tenía problemas para realizar lo esperado, es decir, este código en ocasiones muy frecuentes no podía ni siquiera regresar un valor aleatorio para el movimiento del primer agente (véase anexo 2.10).

## La Versión Beta

Con los antecedentes de la versión previa, el equipo tomó la decisión de dividir el proceso de aprendizaje en dos partes: el primer paso sería explorar el terreno y el segundo, con base en dicha exploración, interpretar lo explorado. Es decir, el programa tendría una primera fase con agentes tontos y una segunda fase con agentes inteligentes. Esta decisión permitió al programa reducir la frecuencia con la que los procesos estocásticos hacían detener la ejecución por errores.

```
Cantidad de agentes tontos: 10
Cantidad de agentes inteligentes: 20
Descuento: 0.75
Velocidad del agente (milisegundos): 0
```

Otro cambio en el código es la implementación de un nuevo color, pues, se estaba trabajando para el agente el color rojo, sin embargo, ahora que se necesitaba diferenciar entre un agente y otro, se decidió dejar el color rojo para el agente tonto y añadir el color verde al agente inteligente.

The image displays two screenshots of a 3x3 grid game state, likely a variant of Tic-Tac-Toe or a similar board game. The grid is represented by a 3x3 array of cells, each containing a numerical value. The top screenshot shows a state where the top-right and middle-middle cells are marked with a red 'X'. The bottom screenshot shows a state where the middle-left cell is marked with a green 'O' and the middle-middle cell is marked with a red 'X'.

0.000	0.000	0.000
0.000 0.000	0.000 0.000	0.000 0.000
0.000	0.000	0.000
0.000	0.000	0.000
0.000 0.000	0.000 X 0.000	0.000 0.000
0.000	0.000	0.000
0.000	0.000	0.000
0.000 0.000	0.000 0.000	0.000 0.000
0.000	0.000	0.000
0.000	0.200	1.000
0.000 0.000	0.200 0.200	1.000 1.000
0.000	0.200	1.000

0.000	0.000	0.000
0.000 0.000	0.000 0.000	0.000 0.000
0.000	0.000	0.000
0.000	0.000	0.000
0.000 O 0.000	0.000 X 0.000	0.000 0.000
0.002	0.000	0.000
0.000	0.000	0.000
0.000 0.008	0.000 0.000	0.000 0.000
0.000	0.040	0.000
0.000	0.200	1.000
0.000 0.000	0.200 0.200	1.000 1.000
0.000	0.200	1.000

Sin embargo, todavía existían errores por corregir sobre los números que se trabajaban de los agentes, algunos contadores y verificar si el descuento de cada casilla estaba ocurriendo de manera que el equipo esperaba (véase anexo 2.11).

## La Pre Release

Esta versión podría ser la versión final (e incluso, el equipo cree que se trabaja de manera más estable con esta versión), sin embargo, no parece haber un aprendizaje por refuerzo, es decir, al pasar por una casilla con un valor establecido, dicho valor no se refuerza, por lo que, en ocasiones, el programa tiende a ciclarse, pero el cálculo de probabilidades funciona correctamente. Esta versión tuvo cambios menores en detalles del descuento, pero los mismos hacen que la versión sea la mejor trabajable si es que se busca un proceso estocástico casi completo (véase anexo 2.12).

## La Versión Final

Esta versión, finalmente recorre el tablero, explora el mismo, llena de valores, interpreta los valores posteriormente y cada vez que pase por un sitio, reforzará ese valor para bien o para mal. El agente tiende a llevar los números lo más cercano a 1 para aquellos movimientos que le favorecen a llegar a la meta y tiende los números a 0 para aquellos movimientos que le perjudican y lo guían a la casilla errónea, pues se desarrolló una ecuación de recurrencia que se plantea en el Problema del Sapo y el Pozo (véase anexo 1.08) pero, los mismos valores hacen que el tablero evolucione de una forma muy diversa.

El único problema es que una ecuación de recurrencia, propiamente la de la casilla errónea, no está bien planteada, por lo que tiene un ajuste no estocástico que lleva a un poco de errores con probabilidades negativas (arregladas con un valor absoluto), problema que, al solucionar, se tuvo el programa definitivo.

0.000	0.000	0.000
0.000 0.108	0.020 0.264	0.203 0.000
0.189	0.000	0.203
0.015	0.000	0.233
0.000 0.000	0.000 X 0.000	0.000 0.000
0.361	0.000	0.984
0.915	0.000	0.791
0.000 0.401	0.306 0.795	0.398 0.000
0.189	0.026	0.996
0.412	1.000	1.000
0.000 0.026	1.000 1.000	1.000 1.000
0.000	1.000	1.000

## El Agente Propio y su Funcionamiento

Tras todas las modificaciones, la versión final inicia un tablero con todas las coordenadas establecidas en 0.5, para que, así, los números más cercanos a cero signifiquen menor tendencia a moverse en dicho sentido y los números más cercanos a uno tengan una mayor atracción del agente y por lo mismo, se decidió eliminar finalmente el agente tonto, es decir, el agente inteligente ahora realiza ambas acciones: exploración y aprendizaje.

0.500	0.500	0.500
0.500 0.500	0.500 0.500	0.500 0.500
0.500	0.500	0.500
0.500	0.500	0.500
0.500 0.500	0.500 X 0.500	0.500 0.500
0.500	0.500	0.500
0.500	0.500	0.500
0.500 0.500	0.500 0.500	0.500 0.500
0.500	0.500	0.500
0.500	1.000	1.000
0.500 0.500	1.000 1.000	1.000 1.000
0.500	1.000	1.000

El programa inicia al preguntar al usuario los datos que se insertarán para calibrar el agente, es decir, la cantidad de agentes en el medio, el descuento y la velocidad de iteración. La cantidad de agentes puede ser cualquier número natural, el descuento debe ser un número entre 0.5 y 1 (sin incluir ambos valores) y la velocidad (que está en milisegundos) puede ser también a gusto. El equipo puso un condicional que, al seleccionar una velocidad cero para el agente, se deshabilitara la función tan repetitiva de limpiar pantalla y sólo mostrara los datos de las casillas (el agente siempre aparece en la casilla inicial por el hecho de deshabilitar la función de limpiar pantalla, sin embargo, internamente sí se mueve).

```
Cantidad de agentes inteligentes: 15
Descuento: 0.75
Velocidad del agente (milisegundos): 750
```

Explicado anteriormente, el agente va actualizando los datos de las casillas por una ecuación de recurrencia que refuerza cada vez más un valor dependiendo del resultado de una prueba.

```
0.500      0.500      0.500
0.500  0.411  0.381  0.433  0.411 0.500
0.166      0.500      0.500

0.263      0.500      0.500
0.500  0.500  0.500 X 0.500  0.500  0.500
0.157      0.500      0.500

0.342      0.500      0.500
0.500  0.219  0.500  0.500  0.500  0.500
0.148      0.125      0.500

0.289      1.000      1.000
0.500  0.125  1.000  1.000  1.000  1.000
0.500      1.000      1.000

Agentes muertos: 2
Agentes victoriosos: 0
```

```
0.500      0.500      0.500
0.500  0.700  0.656  0.708  0.678  0.500
0.600      0.500      0.711

0.633      0.500      0.500
0.500 0.500  0.500 X 0.500  0.500  0.500
0.575      0.500      0.781

0.600      0.500      0.500
0.500  0.500  0.500  0.500  0.500  0.500
0.556      0.500      0.875

0.575      1.000      1.000
0.500  0.500  1.000  1.000  1.000  1.000
0.500      1.000      1.000

Agentes muertos: 0
Agentes victoriosos: 1_
```

A recomendación del grupo, se recomienda que los valores otorgados al programa sea un descuento de 0.75. Este valor es el óptimo para trabajar, puesto que un valor más cercano a 0.5 causará menor diferencia entre las pérdidas y las victorias, pero un valor que tienda a 1 causará actualizaciones de valores inmensas que casi rompen el funcionamiento del agente. Asimismo, la cantidad de agentes más viable es una cantidad entre 10 y 30. La velocidad queda a gusto del usuario para visualizar la ejecución, aunque en pruebas, el grupo lo ejecutó con velocidades entre medio y un segundo (véase anexo 2.13).

# Curso de Inteligencia Artificial con Python: Sección 5

## Video 1

Este programa y todos los posteriores, apoyados por Gomila (2019) se realiza en el primer video de su fuente y es para poder mostrar todos los nombres de los ambientes disponibles en la librería de Algym. Este programa puede ser ejecutado por consola o por Spyder.

```
from gym import envs

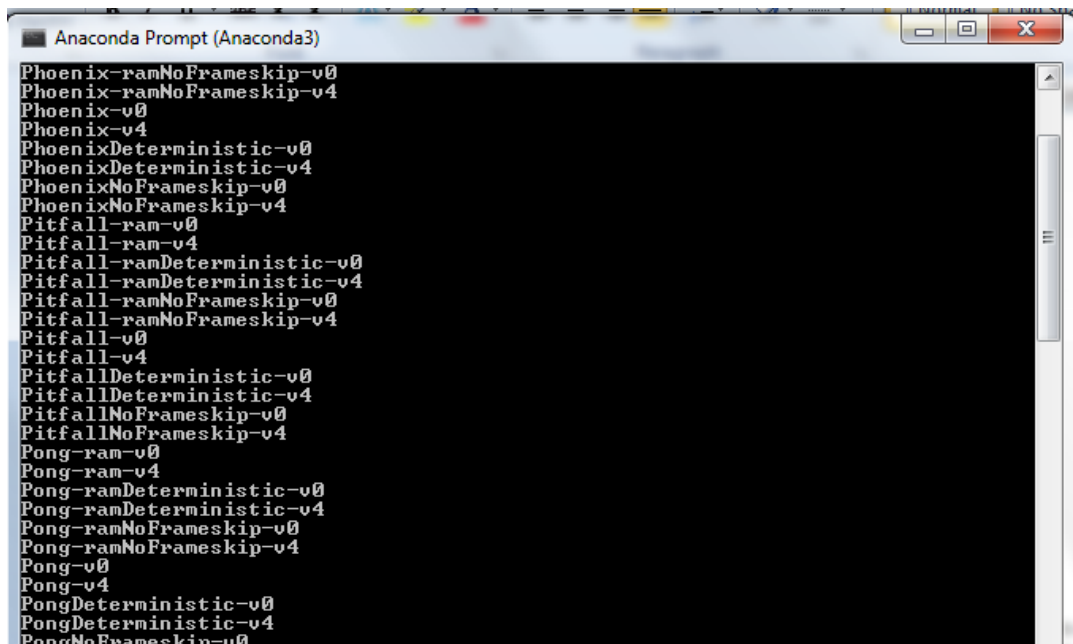
env_names = [env.id for env in envs.registry.all()]

for name in sorted(env_names):
    print(name)
```

## Ejemplo:



```
Gravitar-ram-v0
Gravitar-ram-v4
Gravitar-ramDeterministic-v0
Gravitar-ramDeterministic-v4
Gravitar-ramNoFrameskip-v0
Gravitar-ramNoFrameskip-v4
Gravitar-v0
Gravitar-v4
GravitarDeterministic-v0
GravitarDeterministic-v4
GravitarNoFrameskip-v0
GravitarNoFrameskip-v4
GuessingGame-v0
HalfCheetah-v2
HalfCheetah-v3
HandManipulateBlock-v0
HandManipulateBlockDense-v0
HandManipulateBlockFull-v0
HandManipulateBlockFullDense-v0
HandManipulateBlockRotateParallel-v0
HandManipulateBlockRotateParallelDense-v0
HandManipulateBlockRotateParallelTouchSensors-v0
HandManipulateBlockRotateParallelTouchSensors-v1
HandManipulateBlockRotateParallelTouchSensorsDense-v0
HandManipulateBlockRotateParallelTouchSensorsDense-v1
HandManipulateBlockRotateXYZ-v0
HandManipulateBlockRotateXYZDense-v0
HandManipulateBlockRotateXYZTouchSensors-v0
HandManipulateBlockRotateXYZTouchSensors-v1
```

A screenshot of an Anaconda Prompt window titled "Anaconda Prompt (Anaconda3)". The window has a black background with white text. It displays a list of 30 gym environments, grouped by game: Phoenix, Pitfall, and Pong. Each game has five variants: "ramNoFrameskip-v0", "ramNoFrameskip-v4", "v0", "v4", and "Deterministic-v0". The environments are listed in the following order: Phoenix-ramNoFrameskip-v0, Phoenix-ramNoFrameskip-v4, Phoenix-v0, Phoenix-v4, PhoenixDeterministic-v0, PhoenixDeterministic-v4, PhoenixNoFrameskip-v0, PhoenixNoFrameskip-v4, Pitfall-ram-v0, Pitfall-ram-v4, Pitfall-ramDeterministic-v0, Pitfall-ramDeterministic-v4, Pitfall-ramNoFrameskip-v0, Pitfall-ramNoFrameskip-v4, Pitfall-v0, Pitfall-v4, PitfallDeterministic-v0, PitfallDeterministic-v4, PitfallNoFrameskip-v0, PitfallNoFrameskip-v4, Pong-ram-v0, Pong-ram-v4, Pong-ramDeterministic-v0, Pong-ramDeterministic-v4, Pong-ramNoFrameskip-v0, Pong-ramNoFrameskip-v4, Pong-v0, Pong-v4, PongDeterministic-v0, PongDeterministic-v4, and PongNoFrameskip-v0.

```
Anaconda Prompt (Anaconda3)
Phoenix-ramNoFrameskip-v0
Phoenix-ramNoFrameskip-v4
Phoenix-v0
Phoenix-v4
PhoenixDeterministic-v0
PhoenixDeterministic-v4
PhoenixNoFrameskip-v0
PhoenixNoFrameskip-v4
Pitfall-ram-v0
Pitfall-ram-v4
Pitfall-ramDeterministic-v0
Pitfall-ramDeterministic-v4
Pitfall-ramNoFrameskip-v0
Pitfall-ramNoFrameskip-v4
Pitfall-v0
Pitfall-v4
PitfallDeterministic-v0
PitfallDeterministic-v4
PitfallNoFrameskip-v0
PitfallNoFrameskip-v4
Pong-ram-v0
Pong-ram-v4
Pong-ramDeterministic-v0
Pong-ramDeterministic-v4
Pong-ramNoFrameskip-v0
Pong-ramNoFrameskip-v4
Pong-v0
Pong-v4
PongDeterministic-v0
PongDeterministic-v4
PongNoFrameskip-v0
```

En el cual primero se importan los ambientes de gym, después crea un vector que toma como valor todos los ambientes, para después imprimir el nombre de los ambientes (véase anexo 2.14).

## Video 2

Se realiza un programa que necesita ser utilizado en consola ya que se creó una función que ejecuta el ambiente que le digamos al darle como argumento el nombre del ambiente deseado (véase anexo 2.15).

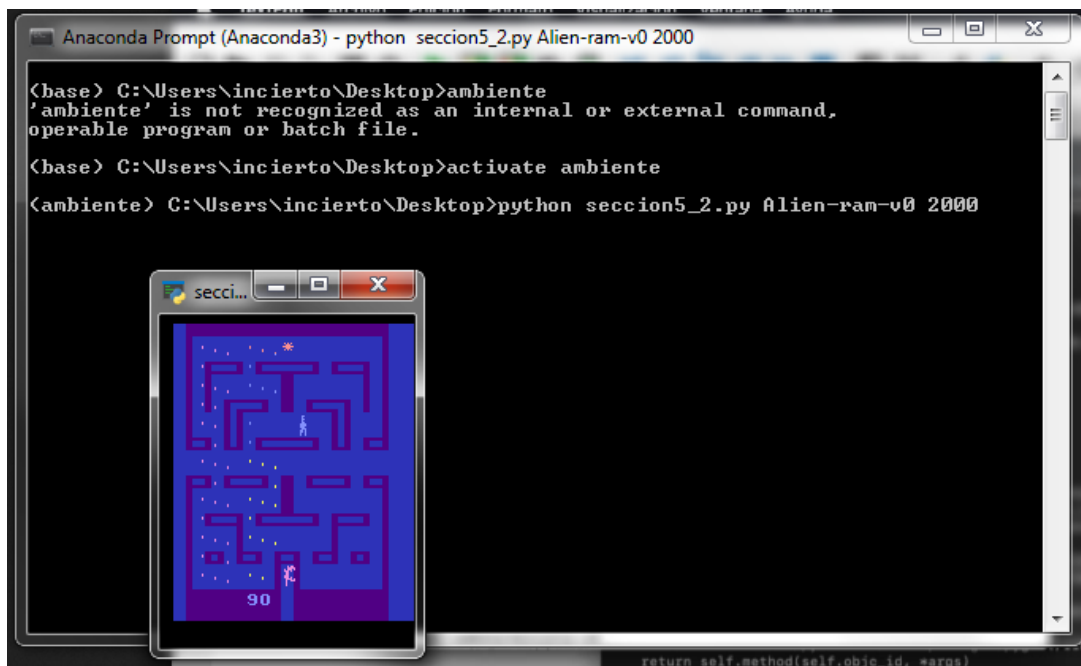
Definición de la función:

```
def run_gym_environment(argv):
    environment = gym.make(argv[1])
    environment.reset()
    for _ in range(int(argv[2])):
        environment.render()
        environment.step(environment.action_space.sample())
    environment.close()
```

Forma de llamar a la función:

```
13 if __name__ == "__main__":
14     run_gym_environment(sys.argv)
```

## Ejemplo:



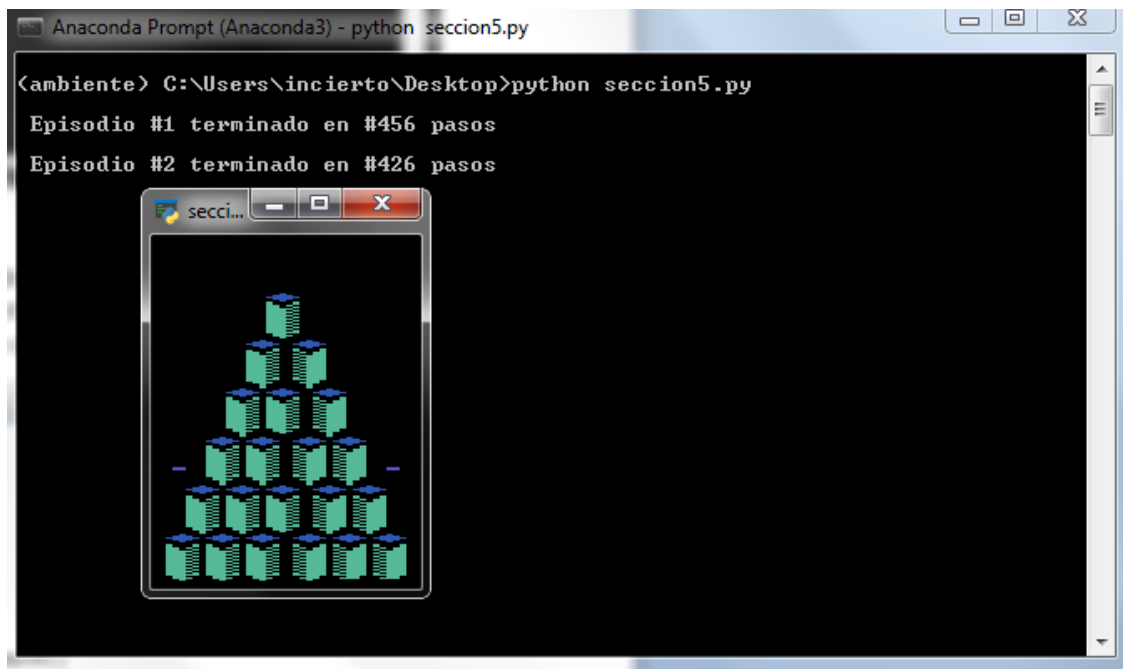
## Videos 3 y 4

Es un programa que usa el ambiente Qbert-v0 de gym en el cual se va a realizar 10 veces (episodios) con un máximo de 500 pasos/acciones para mostrar en cuantos pasos acaba el agente el ambiente por episodio de Qbert (véase anexo 2.16).

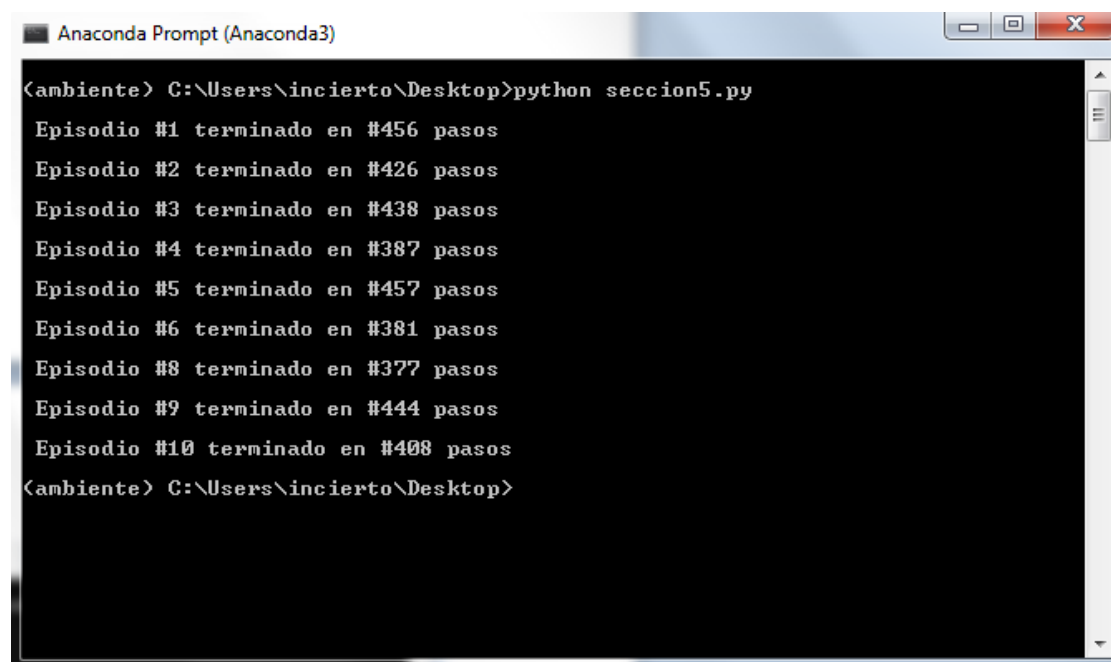
```
seccion5.py x seccion5_1.py x seccion5_2.py x temp.py x  
1 # -*- coding: utf-8 -*-  
2 import gym  
3 environment = gym.make("Qbert-v0")  
4 maxnep = 10  
5 maxspe = 500  
6 for episode in range(maxnep):  
7     obs = environment.reset()  
8     for step in range(maxspe):  
9         environment.render()  
10        action = environment.action_space.sample()  
11        next_state, reward, done, info = environment.step(action)  
12        obs = next_state  
13  
14        if done is True:  
15            print("\n Episodio #{0} terminado en #{0} pasos".format(episode+1, step+1))  
16            break  
17  
18 environment.close()
```



Durante ejecución:



Después de ejecución:



## Video 5

Se explica que para cada videojuego se tienen distintas posibles acciones para el agente debido a que cada juego puede tener distintos controles, en algunos se tienen mayor cantidad de acciones mientras que en otras menores. Se explica que Box es un rango de valores determinado por un valor mínimo y máximo de  $n$  dimensiones

### Ejemplo 1:

Se crea un box cuyo valor mínimo es de -10 y el máximo de 10 cuyo valor de n es 2.

```
Editor - C:\Users\incierto\.spyder-py3\temp.py
seccion5.py x seccion5_1.py x seccion5_2.py x temp.py x
1 # -*- coding: utf-8 -*-
2 import gym
3 gym.spaces.Box(low = -10, high = 10, shape = (2,))
4 |
```

El espacio discreto se crea de 0 - n-1.

### Ejemplo 2:

Se crea un espacio discreto de 0,1,2,3,4

```
Editor - C:\Users\incierto\.spyder-py3\temp.py
seccion5.py x seccion5_1.py x seccion5_2.py x temp.py* x
1 # -*- coding: utf-8 -*-
2 import gym
3 gym.spaces.Discrete(5)
4 |
```

Ejemplo de dict:

```
Editor - C:\Users\incierto\.spyder-py3\temp.py
seccion5.py x seccion5_1.py x seccion5_2.py x temp.py* x
1 # -*- coding: utf-8 -*-
2 import gym
3 gym.spaces.Dict({
4     "posicion": gym.spaces.Discrete(3),
5     "velocidad": gym.spaces.Discrete(2)
6 })
7 |
```

El espacio multibinary podría considerarse como un vector de booleanos ya que se los posibles valores de cada espacio son True o False

### Ejemplo 3:

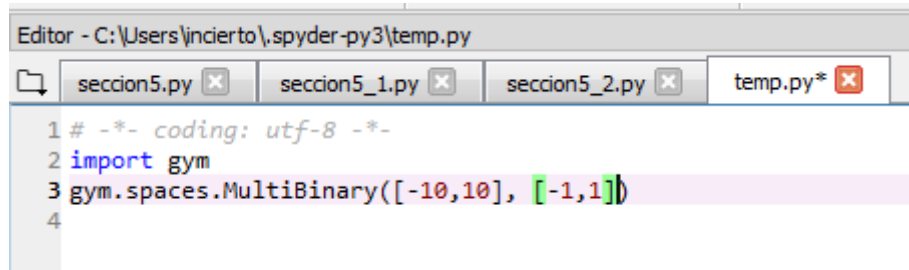
Un mutlibinary de 3 (x,y,z) {True-False}

```
Editor - C:\Users\incierto\.spyder-py3\temp.py
seccion5.py x seccion5_1.py x seccion5_2.py x temp.py* x
1 # -*- coding: utf-8 -*-
2 import gym
3 gym.spaces.MultiBinary(3)
4 |
```

El multidiscrete es como el multibinary pero son parámetros de números reales

#### Ejemplo 4:

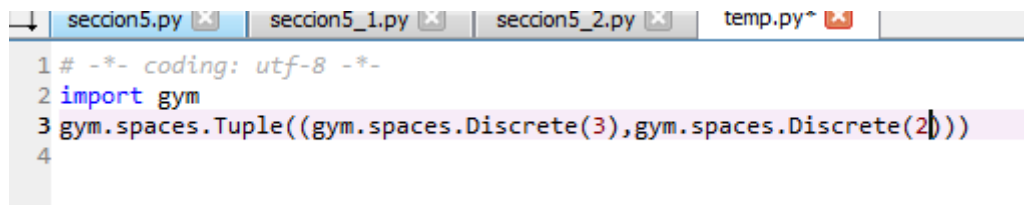
Un multidiscrete conformado por 2 espacios discrete.



```
Editor - C:\Users\incierto\.spyder-py3\temp.py
seccion5.py x seccion5_1.py x seccion5_2.py x temp.py* x
1 # -*- coding: utf-8 -*-
2 import gym
3 gym.spaces.MultiBinary([-10,10], [-1,1])
4
```

Y por último el espacio tuple el cual es un producto de espacios simples

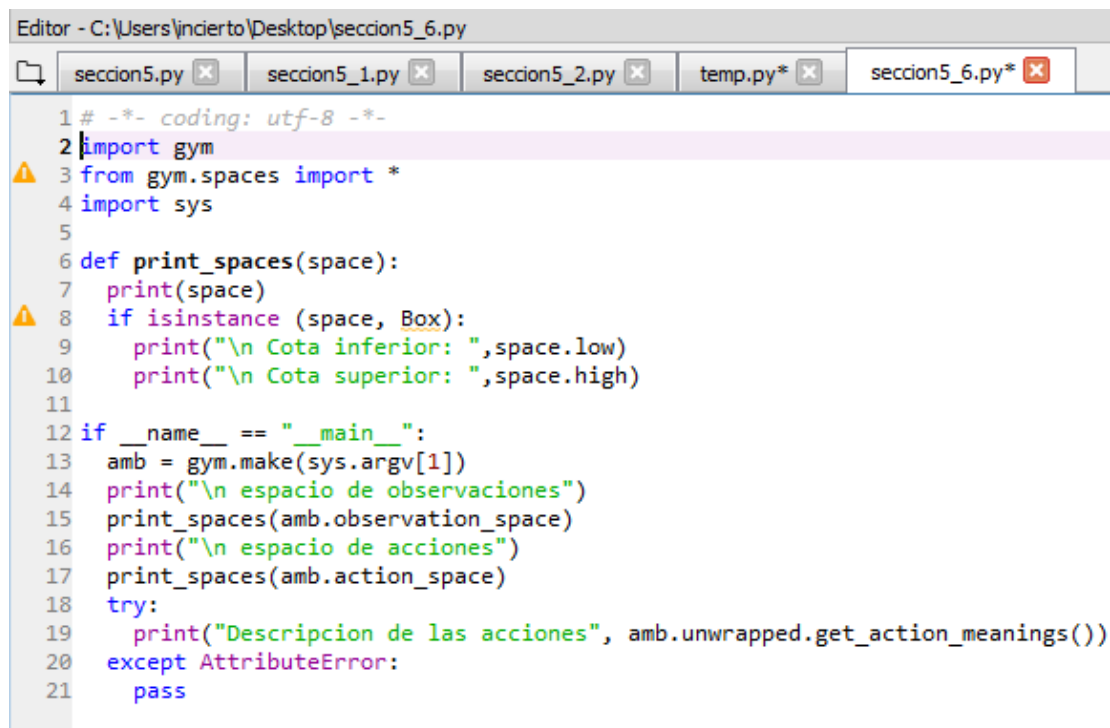
#### Ejemplo 5:



```
seccion5.py x seccion5_1.py x seccion5_2.py x temp.py* x
1 # -*- coding: utf-8 -*-
2 import gym
3 gym.spaces.Tuple((gym.spaces.Discrete(3), gym.spaces.Discrete(2)))
4
```

#### Video 6

Se crea un programa que muestra el espacio de acciones que tiene un ambiente elegido por el usuario (véase anexo 2.17).



```
Editor - C:\Users\incierto\Desktop\seccion5_6.py
seccion5.py x seccion5_1.py x seccion5_2.py x temp.py* x seccion5_6.py* x
1 # -*- coding: utf-8 -*-
2 import gym
3 from gym.spaces import *
4 import sys
5
6 def print_spaces(space):
7     print(space)
8     if isinstance(space, Box):
9         print("\n Cota inferior: ", space.low)
10        print("\n Cota superior: ", space.high)
11
12 if __name__ == "__main__":
13     amb = gym.make(sys.argv[1])
14     print("\n espacio de observaciones")
15     print_spaces(amb.observation_space)
16     print("\n espacio de acciones")
17     print_spaces(amb.action_space)
18     try:
19         print("Descripcion de las acciones", amb.unwrapped.get_action_meanings())
20     except AttributeError:
21         pass
```

#### Ejemplo:

```
Anaconda Prompt (Anaconda3)

<ambiente> C:\Users\incierto\Desktop>python seccion5_6.py CartPole-v0
espacio de observaciones
Box(4,)
Cota inferior: [-4.8000002e+00 -3.4028235e+38 -4.1887903e-01 -3.4028235e+38]
Cota superior: [4.8000002e+00 3.4028235e+38 4.1887903e-01 3.4028235e+38]
espacio de acciones
Discrete(2)
<ambiente> C:\Users\incierto\Desktop>
```

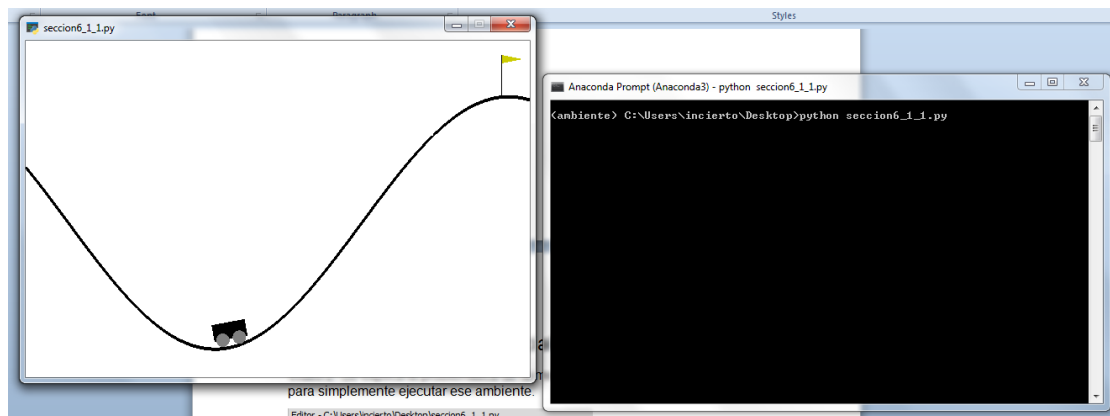
# Curso de Inteligencia Artificial con Python: Sección 6

## Videos 1 y 2

Se explica la problemática de la montaña rusa y se crea un programa para simplemente ejecutar ese ambiente.

```
Editor - C:\Users\incierto\Desktop\seccion6_1_1.py
temp.py x seccion5_6.py x seccion6_1_1.py* x
1 # -*- coding: utf-8 -*-
2 import gym
3 env = gym.make("MountainCar-v0")
4 env.reset()
5 for _ in range(2000):
6     env.render()
7     env.step(env.action_space.sample())
8 env.close
```

Aquí se muestra su ejecución por consola



Y se ve cómo va de un lado a otro para tratar de llegar a la meta, pero es muy raro que lo logre.

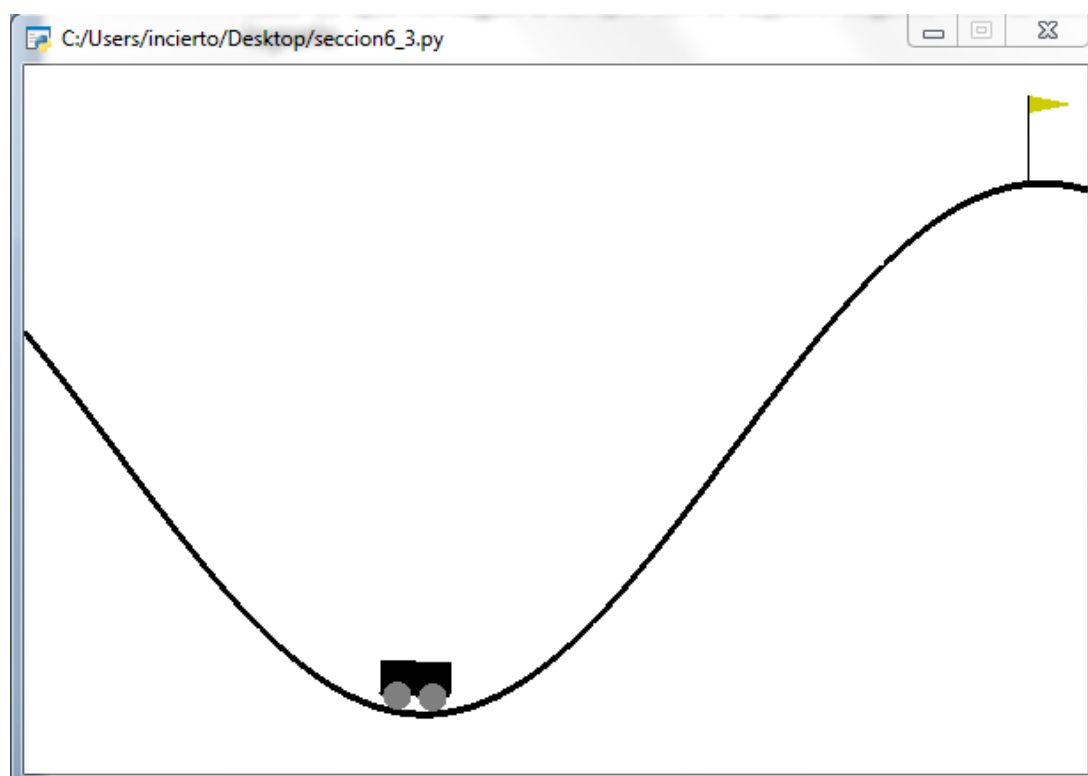
A continuación, se muestra el espacio de acciones y de observaciones que tiene el ambiente de la montaña rusa el cual el de observaciones es una Box con de  $n=2$  con valores inferiores de -1.2, -0.07 y de valores superiores de 0.6, 0.07. Y su parámetro de acciones es discreto de  $n=3$  es decir que tiene 3 posibles acciones a tomar. Avanzar, retroceder y no hacer nada (véase anexo 2.18).

```
Anaconda Prompt (Anaconda3)

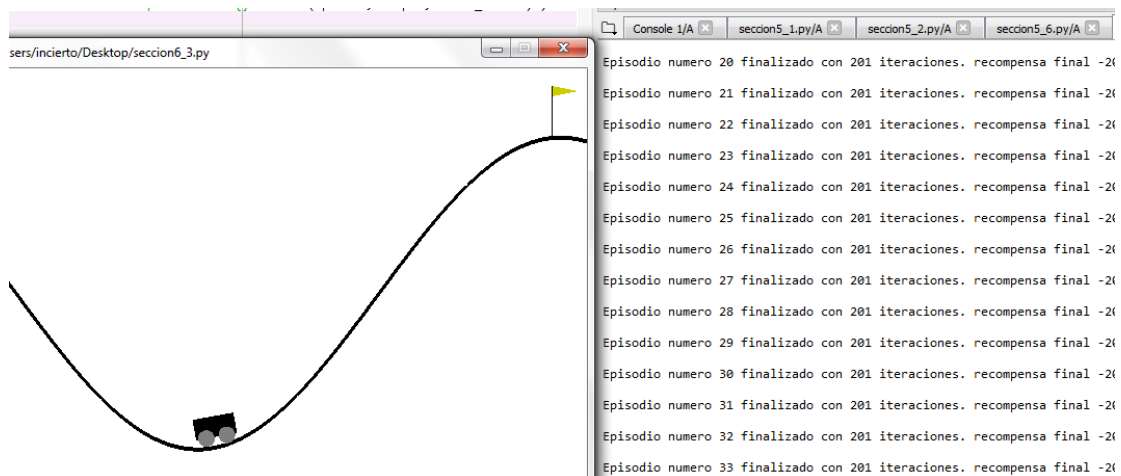
<ambiente> C:\Users\incierto\Desktop>python seccion5_6.py MountainCar-v0
espacio de observaciones
Box(2,)
Cota inferior: [-1.2 -0.071]
Cota superior: [0.6 0.071]
espacio de acciones
Discrete(3)
<ambiente> C:\Users\incierto\Desktop>
```

## Videos 3 y 4

Se da a entender que de preferencia se debe de dar un máximo de episodios y de acciones por episodios para evitar bucles infinitos ya que en el caso de que no llegue a cumplir con el objetivo seguirá intentándolo hasta lograrlo. Y se crea un programa un agente tonto ya que realiza acciones aleatorias. (véase anexo 2.19)



Episodio numero 14 finalizado con 201 iteraciones. recompensa final -200.0  
 Episodio numero 15 finalizado con 201 iteraciones. recompensa final -200.0  
 Episodio numero 16 finalizado con 201 iteraciones. recompensa final -200.0  
 Episodio numero 17 finalizado con 201 iteraciones. recompensa final -200.0  
 Episodio numero 18 finalizado con 201 iteraciones. recompensa final -200.0  
 Episodio numero 19 finalizado con 201 iteraciones. recompensa final -200.0  
 Episodio numero 20 finalizado con 201 iteraciones. recompensa final -200.0  
 Episodio numero 21 finalizado con 201 iteraciones. recompensa final -200.0  
 Episodio numero 22 finalizado con 201 iteraciones. recompensa final -200.0  
 Episodio numero 23 finalizado con 201 iteraciones. recompensa final -200.0  
 Episodio numero 24 finalizado con 201 iteraciones. recompensa final -200.0  
 Episodio numero 25 finalizado con 201 iteraciones. recompensa final -200.0  
 Episodio numero 26 finalizado con 201 iteraciones. recompensa final -200.0  
 Episodio numero 27 finalizado con 201 iteraciones. recompensa final -200.0



## Video 4

Se explica el significado de la siguiente formula

$$Q_t(s, a) = Q_{t-1}(s, a) + \alpha \cdot \left( R(s, a) + \gamma \max_{a'} Q(s', a') - Q_{t-1}(s, a) \right)$$

El cual es que  $Q$  en un momento determinado  $t$  es lo que ya se había aprendido en el momento anterior  $t-1$  más una  $\alpha$  por la recompensa  $R$  más el factor de descuento  $\gamma$  por el máximo de la diferencia entre el estado en un momento final menos el del momento inmediatamente anterior  $t-1$ .

Se declara la teoría de:

- Epsilon\_min: El aprendizaje mientras el incremento de aprendizaje sea superior a dicho valor.
- Alpha: El radio de aprendizaje del agente
- Gamma: factor de descuento del agente
- NUM\_DISCRETE\_BINS: El número de divisiones en el caso de discretizar el espacio de estados continuos.

## Videos 5 al 11

Se crea el programa inteligente aplicando Qlearning por partes, primero se declara la clase QLearner con sus atributos y métodos para la creación de aprendizaje. Los cuales son Discretize, get\_action y learn

```
class QLearner(object):#clase QLearner
    def __init__(self, environment):
        self.obs_shape = environment.observation_space.shape#el observador de espacio de la forma
        self.obs_high = environment.observation_space.high#observador del valor superior
        self.obs_low = environment.observation_space.low#observador del valor inferior
        self.obs_bins = NUM_DISCRETE_BINS#discretizar de 30
        self.bin_width = (self.obs_high-self.obs_low)/self.obs_bins#anchura de la observacion

        self.action_shape = environment.action_space.n#forma de la accion
        self.Q = np.zeros((self.obs_bins+1, self.obs_bins+1, self.action_shape))#La matrices de acciones en zeros de 31*31*2
        self.alpha = ALPHA#el valor de alpha para evitar errores cuando se modifique ALPHA
        self.gamma = GAMMA" "
        self.epsilon = 1.0#el valor base de epsilon sera de 1
```

- La función discretize sirve para crear una tupla del tamaño requerido para el problema.
- La función get\_action es para asignarle una acción al agente dependiendo del valor de epsilon y del random
- La función. learn aplica la operación de bellman para asignarle valores a los posibles estados y acciones que puede tomar el agente.

```
def discretize(self, obs):#funcion discretizar de la clase QLearner
    return tuple(((obs-self.obs_low)/self.bin_width).astype(int))#regresa una tupla del valor de obs-- el valor minimo de observacion entre
#la anchura de la observacion

def get_action(self, obs): #metodo para obtener la accion
    discrete_obs = self.discretize(obs)#discrete_obs va a ser una tupla
    if self.epsilon > EPSILON_MIN:#si el valor de epsilon es mayor al valor de EPSILON_MIN
        self.epsilon -= EPSILON_DECAY#le resta al valor de epsilon del decreciente de epsilon
    if np.random.random() > self.epsilon: #el valor aleatorio el mayor a epsilon
        return np.argmax(self.Q[discrete_obs])#regresa el valor de np.argmax osea una accion inteligente
    else:
        return np.random.choice([a for a in range(self.action_shape)])#eleccion aleatoria dentro de las posibles

def learn(self, obs, action, reward, next_obs):
    discrete_obs = self.discretize(obs)#estado actual
    discrete_next_obs = self.discretize(next_obs)#estado siguiente
    self.Q[discrete_obs][action] += self.alpha*(reward + self.gamma * np.max(self.Q[discrete_next_obs]) - self.Q[discrete_obs][action])
    #ecuacion de bellman para asignarle valores a los estados y a las posibles acciones que puede tomar
```



Y por último se llaman métodos para ejecutar las funciones del agente dependiendo del número de episodios y de pasos establecidos (véase anexo 2.20).

```
def train (agent, environment):#funcion para llamar los metodos del agente qlerner
    best_reward = -float('inf')
    for episode in range (MAX_NUM_EPISODES):
        done = False
        obs = environment.reset()
        total_reward = 0.0
        while not done:
            action = agent.get_action(obs)
            next_obs, reward, done, info = environment.step(action)
            agent.learn(obs, action, reward, next_obs)
            obs = next_obs
            total_reward += reward
            if total_reward > best_reward:
                best_reward = total_reward
            print("Episodio numero {} con recompensa: {}, mejor recompensa {}, epsilon: {}".format(episode, total_reward, best_reward, agent.epsilon))
        return np.argmax(agent.Q, axis = 2)

def test (agent, environment, policy): #funcion para llamar los metodos del agente qlerner
    done = False
    obs = environment.reset()
    total_reward = 0.0
    while not done:
        action = agent.get_action(obs)
        next_obs, reward, done, info = environment.step(action)
        agent.learn(obs, action, reward, next_obs)
        obs = next_obs
        total_reward += reward
    return total_reward

if __name__ == "__main__":
    environment = gym.make("MountainCar-v0")
    agent = QLearner(environment)
    learned_policy = train(agent, environment)
    monitor_path = "./monitor_output"
    environment = gym.wrappers.Monitor(environment, monitor_path, force = True)
    for _ in range (1000):
        test(agent, environment, learned_policy)
    environment.close()
```

Ejecución:

```
Anaconda Prompt (Anaconda3) - python seccion6_11.py
Episodio numero 542 con recompensa: -200.0, mejor recompensa -200.0, epsilon: 0.
972849999996205
Episodio numero 543 con recompensa: -200.0, mejor recompensa -200.0, epsilon: 0.
972799999996198
Episodio numero 544 con recompensa: -200.0, mejor recompensa -200.0, epsilon: 0.
972749999996191
Episodio numero 545 con recompensa: -200.0, mejor recompensa -200.0, epsilon: 0.
9726999999961841
Episodio numero 546 con recompensa: -200.0, mejor recompensa -200.0, epsilon: 0.
9726499999961771
Episodio numero 547 con recompensa: -200.0, mejor recompensa -200.0, epsilon: 0.
9725999999961701
Episodio numero 548 con recompensa: -200.0, mejor recompensa -200.0, epsilon: 0.
9725499999961631
Episodio numero 549 con recompensa: -200.0, mejor recompensa -200.0, epsilon: 0.
9724999999961561
Episodio numero 550 con recompensa: -200.0, mejor recompensa -200.0, epsilon: 0.
9724499999961491
Episodio numero 551 con recompensa: -200.0, mejor recompensa -200.0, epsilon: 0.
9723999999961421
Episodio numero 552 con recompensa: -200.0, mejor recompensa -200.0, epsilon: 0.
9723499999961351
Episodio numero 553 con recompensa: -200.0, mejor recompensa -200.0, epsilon: 0.
9722999999961282
Episodio numero 554 con recompensa: -200.0, mejor recompensa -200.0, epsilon: 0.
9722499999961212
Episodio numero 555 con recompensa: -200.0, mejor recompensa -200.0, epsilon: 0.
9721999999961142
Episodio numero 556 con recompensa: -200.0, mejor recompensa -200.0, epsilon: 0.
9721499999961072
Episodio numero 557 con recompensa: -200.0, mejor recompensa -200.0, epsilon: 0.
9720999999961002
Episodio numero 558 con recompensa: -200.0, mejor recompensa -200.0, epsilon: 0.
9720499999960932
Episodio numero 559 con recompensa: -200.0, mejor recompensa -200.0, epsilon: 0.
9719999999960862
Episodio numero 560 con recompensa: -200.0, mejor recompensa -200.0, epsilon: 0.
9719499999960792
Episodio numero 561 con recompensa: -200.0, mejor recompensa -200.0, epsilon: 0.
9718999999960722
Episodio numero 562 con recompensa: -200.0, mejor recompensa -200.0, epsilon: 0.
9718499999960653
Episodio numero 563 con recompensa: -200.0, mejor recompensa -200.0, epsilon: 0.
9717999999960583
Episodio numero 564 con recompensa: -200.0, mejor recompensa -200.0, epsilon: 0.
9717499999960513
```

```
(2) EN TODAS LAS PELEAS... Seccion5.ipynb - Colaboratory
Anaconda Prompt (Anaconda3)
Episodio numero 49976 con recompensa: -133.0, mejor recompensa -91.0, epsilon: 0.
.0049999999486979654
Episodio numero 49977 con recompensa: -188.0, mejor recompensa -91.0, epsilon: 0.
.0049999999486979654
Episodio numero 49978 con recompensa: -131.0, mejor recompensa -91.0, epsilon: 0.
.0049999999486979654
Episodio numero 49979 con recompensa: -117.0, mejor recompensa -91.0, epsilon: 0.
.0049999999486979654
Episodio numero 49980 con recompensa: -143.0, mejor recompensa -91.0, epsilon: 0.
.0049999999486979654
Episodio numero 49981 con recompensa: -141.0, mejor recompensa -91.0, epsilon: 0.
.0049999999486979654
Episodio numero 49982 con recompensa: -134.0, mejor recompensa -91.0, epsilon: 0.
.0049999999486979654
Episodio numero 49983 con recompensa: -117.0, mejor recompensa -91.0, epsilon: 0.
.0049999999486979654
Episodio numero 49984 con recompensa: -153.0, mejor recompensa -91.0, epsilon: 0.
.0049999999486979654
Episodio numero 49985 con recompensa: -154.0, mejor recompensa -91.0, epsilon: 0.
.0049999999486979654
Episodio numero 49986 con recompensa: -142.0, mejor recompensa -91.0, epsilon: 0.
.0049999999486979654
Episodio numero 49987 con recompensa: -116.0, mejor recompensa -91.0, epsilon: 0.
.0049999999486979654
Episodio numero 49988 con recompensa: -141.0, mejor recompensa -91.0, epsilon: 0.
.0049999999486979654
Episodio numero 49989 con recompensa: -200.0, mejor recompensa -91.0, epsilon: 0.
.0049999999486979654
Episodio numero 49990 con recompensa: -135.0, mejor recompensa -91.0, epsilon: 0.
.0049999999486979654
Episodio numero 49991 con recompensa: -139.0, mejor recompensa -91.0, epsilon: 0.
.0049999999486979654
Episodio numero 49992 con recompensa: -149.0, mejor recompensa -91.0, epsilon: 0.
.0049999999486979654
Episodio numero 49993 con recompensa: -149.0, mejor recompensa -91.0, epsilon: 0.
.0049999999486979654
Episodio numero 49994 con recompensa: -141.0, mejor recompensa -91.0, epsilon: 0.
.0049999999486979654
Episodio numero 49995 con recompensa: -141.0, mejor recompensa -91.0, epsilon: 0.
.0049999999486979654
Episodio numero 49996 con recompensa: -145.0, mejor recompensa -91.0, epsilon: 0.
.0049999999486979654
Episodio numero 49997 con recompensa: -156.0, mejor recompensa -91.0, epsilon: 0.
.0049999999486979654
Episodio numero 49998 con recompensa: -154.0, mejor recompensa -91.0, epsilon: 0.
.0049999999486979654
Episodio numero 49999 con recompensa: -141.0, mejor recompensa -91.0, epsilon: 0.
```

## Bibliografía

- Gomila, J. (2019). Curso de Inteligencia Artificial con Python. agosto 26, 2019, de Udemy Sitio web: <https://www.udemy.com/course/curso-completo-de-inteligencia-artificial/>
- Iglesias, A. (2016). La historia de la inteligencia artificial: desde los orígenes hasta hoy. agosto 28, 2019, de ticbeat Sitio web: <https://www.ticbeat.com/innovacion/la-historia-de-la-inteligencia-artificial-desde-los-origenes-hasta-hoy/>
- Sancho, F. (2019). Aprendizaje por Refuerzo. agosto 29, 2019, de cs Sitio web: <http://www.cs.us.es/~fsancho/?e=109>

## Anexos

### Primera Parte: Anexos Internos al Documento

#### Anexo 1.01: Las Variables

```
//Las Variables-----  
char tablero[4][3];  
int random,x=0,y=0;  
string camino=" ";  
bool ganador=false,vivo=true;
```

#### Anexo 1.02: La Inicialización del Tablero

```
void inicio()  
{  
    int i=0,j=0;  
    do  
    {  
        do  
        {  
            tablero[i][j]=' '  
            j++;  
        }  
        while(j<3);  
        j=0;  
        i++;  
    }  
    while(i<4);  
    tablero[0][0]='0';  
    tablero[1][1]='0';  
    tablero[3][1]='X';  
    tablero[3][2]='V';  
  
    camino.push_back(x+48);  
    camino.push_back(',');  
    camino.push_back(y+48);  
    camino.push_back(' ');  
}
```

#### Anexo 1.03: Las Restricciones de Movimiento

```

//Restricciones de movimiento-----
if(random==1 && y==0)
{
    continue; //Imposibilita movimiento hacia arriba
}
if(random==1 && (x==1 && y==2))
{
    continue; //Imposibilita movimiento hacia arriba (cuadrado)
}
if(random==2 && x==2)
{
    continue; //Imposibilita movimiento hacia la derecha
}
if(random==2 && (x==0 && y==1))
{
    continue; //Imposibilita movimiento hacia la derecha (cuadrado)
}
if(random==3 && y==3)
{
    continue; //Imposibilita movimiento hacia abajo
}
if(random==3 && (x==1 && y==0))
{
    continue; //Imposibilita movimiento hacia abajo (cuadrado)
}
if(random==4 && x==0)
{
    continue; //Imposibilita movimiento hacia la izquierda
}
if(random==4 && (x==2 && y==1))
{
    continue; //Imposibilita movimiento hacia la izquierda (cuadrado)
}

```

Anexo 1.04: Ejemplo de Movimiento del Agente: Hacia Arriba

```

if(random==1)
{
    tablero[y][x]=' ';
    y--; //Avanza hacia arriba
    tablero[y][x]='0';

    camino.push_back(x+48);
    camino.push_back(',');
    camino.push_back(y+48);
    camino.push_back(' ');
}

```

Anexo 1.05: Código Antes y Después de Colorear el Mapa

Antes:

```

void impr()
{
    printf("    0.000      0.000      0.000      \n");
    printf("0.000 %c 0.000    0.000 %c 0.000    0.000 %c 0.000 \n",tablero[0][0],tablero[0][1],tablero[0][2]);
    printf("    0.000      0.000      0.000      \n");
    printf("\n");
    printf("    0.000      0.000      0.000      \n");
    printf("0.000 %c 0.000      %c      0.000 %c 0.000 \n",tablero[1][0],tablero[1][1],tablero[1][2]);
    printf("    0.000      0.000      0.000      \n");
    printf("\n");
    printf("    0.000      0.000      0.000      \n");
    printf("0.000 %c 0.000    0.000 %c 0.000    0.000 %c 0.000 \n",tablero[2][0],tablero[2][1],tablero[2][2]);
    printf("    0.000      0.000      0.000      \n");
    printf("\n");
    printf("    0.000      -1.000      1.000      \n");
    printf("0.000 %c 0.000    -1.000 %c -1.000    1.000 %c 1.000 \n",tablero[3][0],tablero[3][1],tablero[3][2]);
    printf("    0.000      -1.000      1.000      \n");

    cout<<"\n\n"<<camino;
}

```

Después:

```

SetConsoleTextAttribute(GetStdHandle(STD_OUTPUT_HANDLE), 8);
printf("    0.000      0.000      0.000      \n");
printf("0.000 ");
SetConsoleTextAttribute(GetStdHandle(STD_OUTPUT_HANDLE), 12);
printf("%c",tablero[0][0]);
SetConsoleTextAttribute(GetStdHandle(STD_OUTPUT_HANDLE), 8);
printf(" 0.000 0.000 ");
SetConsoleTextAttribute(GetStdHandle(STD_OUTPUT_HANDLE), 12);
printf("%c",tablero[0][1]);
SetConsoleTextAttribute(GetStdHandle(STD_OUTPUT_HANDLE), 8);
printf(" 0.000 0.000 ");
SetConsoleTextAttribute(GetStdHandle(STD_OUTPUT_HANDLE), 12);
printf("%c",tablero[0][2]);
SetConsoleTextAttribute(GetStdHandle(STD_OUTPUT_HANDLE), 8);
printf(" 0.000 \n");
printf("    0.000      0.000      0.000      \n");
printf("\n");

printf("    0.000      0.000      0.000      \n");
printf("0.000 ");
SetConsoleTextAttribute(GetStdHandle(STD_OUTPUT_HANDLE), 12);
printf("%c",tablero[1][0]);
SetConsoleTextAttribute(GetStdHandle(STD_OUTPUT_HANDLE), 8);
printf(" 0.000 ");
SetConsoleTextAttribute(GetStdHandle(STD_OUTPUT_HANDLE), 12);
printf("%c",tablero[1][1]);
SetConsoleTextAttribute(GetStdHandle(STD_OUTPUT_HANDLE), 8);
printf(" 0.000 ");
SetConsoleTextAttribute(GetStdHandle(STD_OUTPUT_HANDLE), 12);
printf("%c",tablero[1][2]);
SetConsoleTextAttribute(GetStdHandle(STD_OUTPUT_HANDLE), 8);
printf(" \n");

```

## Anexo 1.06: La Actualización de los Valores

```

void actualizavalores()
{
    int i=1,j=0;
    float a,b;
    if(vivo==false)
    {
        a=0.1;
    }
    if(ganador==true)
    {
        a=0.9;
    }
    do
    {
        switch (c1[cont-i].dir)
        {
            case 1:
            {
                b=max(max(t[c1[cont-i+1].x][c1[cont-i+1].y].up,t[c1[cont-i+1].x][c1[cont-i+1].y].down),
                t[c1[cont-i].x][c1[cont-i].y].up);
                t[c1[cont-i].x][c1[cont-i].y].up=max(max(t[c1[cont-i+1].x][c1[cont-i+1].y].up,t[c1[cont-i+1].x][c1[cont-i+1].y].down),
                t[c1[cont-i].x][c1[cont-i].y].up);
                printf("Norte\n");
                break;
            }
        }
    }
}

```

## Anexo 1.07: Calibrar la Ejecución del Programa

```

//Calibrar el programa: Cantidad de agentes, descuento, velocidad e incluso ver el proceso--
void calibrar()
{
    printf("Cantidad de agentes: ");
    scanf("%d",&menucant);
    printf("\n");
    printf("Descuento: ");
    scanf("%f",&menudesc);
    printf("\n");
    printf("Velocidad del agente (milisegundos): ");
    scanf("%d",&menuvel);
    printf("\n");
    //Con menuvel = 0 deshabilita la tabla visible en los primeros n - 1 agentes
}

```

## Anexo 1.08: El problema del sapo y el pozo

El problema del sapo y el pozo plantea a un sapo en un pozo de  $n$  metros. El sapo salta la mitad de los metros del camino que le falta y la pregunta a este problema es cuántos saltos le tomará al sapo salir del pozo.

La respuesta es que nunca lo logrará, pues es un límite que tenderá al valor de esos metros pero que nunca llegará, basado en la ecuación de recurrencia:

$$a_{n+1} = \frac{1 - a_n}{2} + a_n$$

En un principio, la ecuación de recurrencia era perfecta para el problema actual, y así, el programa pudiera aplicar un aprendizaje por refuerzo y mejorar las probabilidades de un suceso victorioso, donde el descuento fuera de la mitad y el valor actual de la tabla fuera cero, sin embargo, al manipular la ecuación de recurrencia, se encontró primeramente que es equivalente escribirla de esa forma como escribir:

$$a_{n+1} = [(0.5)(1 - a_n)] + a_n$$

De modo que se encuentra el descuento en la ecuación de recurrencia de forma “escondida”. Asimismo, es el 1 el que se encontró que era el valor por modificar cuando la tabla ya tenía valores en su interior, por lo que se tuvo la siguiente ecuación de recurrencia:

$$a_{n+1} = [(D)(z_n - a_n)] + a_n$$

Donde  $D$  es el descuento aplicado y  $z_n$  es el valor al que debe tender (esta ecuación debe tender a 1, la siguiente a 0, pero conforme los valores cambien, sus límites también), que no necesariamente deba estar vacía ahora

Para la ecuación de recurrencia de la casilla de derrota, se invirtieron todos los operadores, de modo que se comenzaría la ecuación en 1 y los términos quedarían acomodados de la siguiente manera:

$$b_{n+1} = b_n - [(D)(b_n + z_n)]$$

Puede observarse el comportamiento de ambas ecuaciones de recurrencia (con  $D = 0.5$  y  $z_n$  sin alterar) en la tabla siguiente:



$n + 1$	$a_{n+1}$	$b_{n+1}$
0	0	1
1	0.5	0.5
2	0.75	0.25
3	0.875	0.125
4	0.9375	0.0625
5	0.9687	0.0312
6	0.9843	0.0156
7	0.9921	0.0078
8	0.996	0.0039

## Segunda Parte: Anexos Externos al Documento

Esta sección de anexos no pudo ser incluida en el presente, debido a que son los archivos de código de los agentes y en el caso muy específico del agente propio, se incluyen las distintas evoluciones que le tomó al equipo, modificar el código para conseguir la versión final.

Se anexan en caso de que se quiera realizar una prueba de ejecución o revisar el código de forma completa, puesto que los extractos más importantes del código se encuentran en los anexos internos al documento.

Estos anexos se adjuntaron al mismo documento en su entrega con el mismo nombre que se describen a continuación.

Anexo 2.01: Agente Tonto

Anexo 2.02: Agente Base: Ecuación de Bellman

Anexo 2.03: Agente Propio (0.1)

Anexo 2.04: Agente Propio (0.5)

Anexo 2.05: Agente Propio (1.0)

Anexo 2.06: Agente Propio (1.5)

Anexo 2.07: Agente Propio (2.0)

Anexo 2.08: Agente Propio (2.5)

Anexo 2.09: Agente Propio (2.8)

Anexo 2.10: Agente Propio (Versión Alfa)

Anexo 2.11: Agente Propio (Versión Beta)

Anexo 2.12: Agente Propio (Pre Release)

Anexo 2.13: Agente Propio (Versión Final)

Anexo 2.14: Programa 1: Sección 5



Anexo 2.15: Programa 2: Sección 5

Anexo 2.16: Programa 3: Sección 5

Anexo 2.17: Programa 4: Sección 5

Anexo 2.18: Programa 1: Sección 6

Anexo 2.19: Programa 2: Sección 6

Anexo 2.20: Programa 3: Sección 6