Fundamentos de Sistemas Inteligentes
4º Grado en Ingeniería Informática

Práctica de Sistemas de Búsqueda
“JUGANDO AL TETRIS”

Marcos Rivas Kyoguro (70962760D) marcos.rivkyo@usal.es
Pablo Moreno Barrios (70908442V) pablo.moreno@usal.es

ÍNDICE

Práctica 1.- Implementación en Prolog.

1. Introducción

La práctica tiene como objetivo desarrollar la implementación básica del juego Tetris utilizando el lenguaje Visual Prolog. El programa simula el comportamiento de las piezas que caen y se colocan en un tablero, incluyendo la validación de éstas, la eliminación de filas completas y el cálculo del nuevo estado del tablero.

El programa se divide en cuatro secciones

- **Domains:** Es la declaración de los dominios sobre los que se expresan las relaciones.
- **Predicates:** Es la declaración de las relaciones (predicados) que van a aparecer en la base de conocimientos
- **Clauses:** Es la base de conocimientos donde se expresan los conocimientos en forma de reglas y hechos.
- **Goal:** Es la fórmula que se pretende hacer cierta.

2. Predicados más relevantes

En general, los predicados se organizan de la siguiente manera:

- **Predicados de manipulación del tablero**:
  - vacia, pinta
  - escribelista, escribesol.
- **Predicados para colocar las piezas**:
  - mete, cambia_fila
  - modifica.
- **Predicados para gestionar las filas**:
  - extrae_fila
  - obtiene_fila
  - recalcula_suelo
  - filallena
  - quitafilas
  - renumera
  - anhade.
- **Predicados de control y lógica**:
  - backtrack
  - regla
  - mayor
  - recorrefila
  - limpia_filas.

A continuación, analizaremos e interpretaremos los principales predicados del código:

1. **tetris()**

   El predicado tetris() es el punto de entrada del programa y tiene como propósito inicializar el tablero vacío, simular el proceso de colocación de fichas y mostrar el resultado final.

   Llama al predicado backtrack(), que implementa el algoritmo para resolver el problema de colocación de la secuencia de fichas que recibe ([1,2,3,3]).

```
tetris():-

   vacia(T),

   write("========================================\n"),
   write("=                       ¡ JUGANDO !
   write("========================================\n\n"),
   backtrack([1,2,3,3],T,[],Solucion_Final),
   write("\n\nORDEN DE LAS FICHAS: \n\n"),
   write(Solucion_Final,'\n').
```

2. **backtrack(juego,tablero,solucion,solucion)**

   El predicado backtrack() es la base del juego que aplica recursión para colocar las fichas en el tablero siguiendo las reglas definidas, y genera una solución final. El backtracking asegura una búsqueda exhaustiva de soluciones, encontrando una disposición correcta de las fichas en el tablero.

   Este consta de dos casos;

   1. ***Caso Base***: Se alcanza si no quedan más fichas por colocar. Imprime un mensaje indicando que la solución final está lista.

   2. ***Caso Recursivo***: Se alcanza si hay fichas por colocar. Llama a regla() para validar y calcular la posición (Fila, Columna) donde se colocará la ficha actual en el tablero, generando un nuevo tablero (Tab_int).
      Recursivamente procesa las fichas restantes (RestoFichas) con el tablero actualizado y acumula la solución parcial.

```
/* CASO BASE */

backtrack([],Tablero,Solucion,Solucion):-
   write("========================================\n"),
   write("                   SOLUCIÓN FINAL            \n"),
   write("========================================\n\n"),
   pinta(Tablero),
   escribesol(Solucion).


/* CASO RECURSIVO */

backtrack([Ficha | RestoFichas], Tab_in, SolucionParcial, SolucionFinal) :-
   regla(Tab_in, Ficha, Fila, Columna, Tab_int),
   backtrack(RestoFichas, Tab_int, [f(Ficha, Fila, Columna) | SolucionParcial], SolucionFinal).
```

3. regla(tablero,tipo,orientacion,postab,tablero)

El predicado regla() define las reglas necesarias para colocar las fichas en el tablero. Permite probar sistemáticamente todas las combinaciones posibles de posición y orientación, facilitando la búsqueda de soluciones.

En resumen, regla() es el núcleo de las decisiones de colocación, que explora de manera organizada y visual todas las opciones disponibles para cada ficha. Se tiene una regla para cada orientación y columna posible.

```
/* Empiezan las reglas de colocacion */
/* Columna 1*/

        regla(Tab_in, Ficha, 0, 1, Tab_int) :-
            write("Pruebo la ficha: T = "), write(Ficha),
            write(" con O = 0, en C = 1"), nl,
            mete(f(Ficha, 0, 1), Tab_in, Tab_int),
            pinta(Tab_int),
            write("He probado la ficha: T = "), write(Ficha),
            write(" con O = 0, en C = 1"),
            write("\n\n\n"), nl.

        regla(Tab_in, Ficha, 1, 1, Tab_int) :-
            write("Pruebo la ficha: T = "), write(Ficha),
            write(" con O = 1, en C = 1"), nl,
            mete(f(Ficha, 1, 1), Tab_in, Tab_int),
            pinta(Tab_int),
            write("He probado la ficha: T = "), write(Ficha),
            write(" con O = 1, en C = 1"),
            write("\n\n\n"), nl.
```

```
/* Columna 2 */
        regla(Tab_in, Ficha, 0, 2, Tab_int) :-
            write("Pruebo la ficha: T = "), write(Ficha),
            write(" con O = 0, en C = 2"), nl,
            mete(f(Ficha, 0, 2), Tab_in, Tab_int),
            pinta(Tab_int),
            write("He probado la ficha: T = "), write(Ficha),
            write(" con O = 0, en C = 2"),
            write("\n\n\n"), nl.

        regla(Tab_in, Ficha, 1, 2, Tab_int) :-
            write("Pruebo la ficha: T = "), write(Ficha),
            write(" con O = 1, en C = 2"), nl,
            mete(f(Ficha, 1, 2), Tab_in, Tab_int),
            pinta(Tab_int),
            write("He probado la ficha: T = "), write(Ficha),
            write(" con O = 1, en C = 2"),
            write("\n\n\n"), nl.
```

```
/* Columna 3 */
        regla(Tab_in, Ficha, 0, 3, Tab_int) :-
            write("Pruebo la ficha: T = "), write(Ficha),
            write(" con O = 0, en C = 3"), nl,
            mete(f(Ficha, 0, 3), Tab_in, Tab_int),
            pinta(Tab_int),
            write("He probado la ficha: T = "), write(Ficha),
            write(" con O = 0, en C = 3"),
            write("\n\n\n"), nl.

        regla(Tab_in, Ficha, 1, 3, Tab_int) :-
            write("Pruebo la ficha: T = "), write(Ficha),
            write(" con O = 1, en C = 3"), nl,
            mete(f(Ficha, 1, 3), Tab_in, Tab_int),
            pinta(Tab_int),
            write("He probado la ficha: T = "), write(Ficha),
            write(" con O = 1, en C = 3"),
            write("\n\n\n"), nl.
```

```
/* Columna 4 */
        regla(Tab_in, Ficha, 0, 4, Tab_int) :-
            write("Pruebo la ficha: T = "), write(Ficha),
            write(" con O = 0, en C = 4"), nl,
            mete(f(Ficha, 0, 4), Tab_in, Tab_int),
            pinta(Tab_int),
            write("He probado la ficha: T = "), write(Ficha),
            write(" con O = 0, en C = 4"),
            write("\n\n\n"), nl.

        regla(Tab_in, Ficha, 1, 4, Tab_int) :-
            write("Pruebo la ficha: T = "), write(Ficha),
            write(" con O = 1, en C = 4"), nl,
            mete(f(Ficha, 1, 4), Tab_in, Tab_int),
            pinta(Tab_int),
            write("He probado la ficha: T = "), write(Ficha),
            write(" con O = 1, en C = 4"),
            write("\n\n\n"), nl.
```

```
/* Columna 5 */
        regla(Tab_in, Ficha, 0, 5, Tab_int) :-
            write("Pruebo la ficha: T = "), write(Ficha),
            write(" con O = 0, en C = 5"), nl,
            mete(f(Ficha, 0, 5), Tab_in, Tab_int),
            pinta(Tab_int),
            write("He probado la ficha: T = "), write(Ficha),
            write(" con O = 0, en C = 5"),
            write("\n\n\n"), nl.

        regla(Tab_in, Ficha, 1, 5, Tab_int) :-
            write("Pruebo la ficha: T = "), write(Ficha),
            write(" con O = 1, en C = 5"), nl,
            mete(f(Ficha, 1, 5), Tab_in, Tab_int),
            pinta(Tab_int),
            write("He probado la ficha: T = "), write(Ficha),
            write(" con O = 1, en C = 5"),
            write("\n\n\n"), nl.
```

4. mete(ficha,tablero,tablero)

El predicado mete() determina cómo se colocan las fichas de Tetris en el tablero. Cada ficha tiene un tipo, orientación, y columna, y las reglas verifican si es posible colocarla en función de las condiciones del tablero.

Lo primero que hace es asegurarse de que la ficha se coloque dentro de los límites del tablero (columnas 1 a 4).

Después calcula la altura del suelo en las columnas afectadas para determinar si hay espacio suficiente para la ficha.

Verifica que la ficha pueda encajar sin exceder el límite superior del tablero ni interrumpir otras fichas ya colocadas.

Finalmente, si ha aprobado todas las comprobaciones previas;

- Se recalculan las alturas de las columnas afectadas.
- Se modifican las filas correspondientes a la colocación de la ficha.
- Se eliminan las filas llenas y se reorganiza el tablero.

Se pueden meter 4 tipos de fichas (1-4), cada una de las cuales puede tener 4 orientaciones distintas (0-3).

Tipo de Ficha **Orientación**	**1**	**2**	**3**	**4**
0	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1
1	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1
2	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1
3	1 1 1 1	1 1 1 1	1 1 1 1	1 1 1 1

A continuación, interpretaremos el funcionamiento de mete() cuando recibe una ficha de tipo 1 y orientación 0.

```
/* A PARTIR DE AQUI VIENEN LAS INTRODUCCIONES DE LAS FICHAS */
/*  TIPO   1. LA T invertida*/

/*        X  */
/* Ficha XXX*/
/* Orientacion 0 */
  mete(f(1,0,Columna),Tablero_in,Tablero_out):-  /*es una T.-->1  con la base horizontal --> 0 centrada en el pivote --> 3*/
     Tablero_in=tab(Suelo_in,Tabla_in),

     Columna>1,Columna<5,

     Columna0=Columna-1,
     Columna1=Columna+1,
     obtiene_fila(Fila0,Suelo_in,1,Columna0),
     obtiene_fila(Fila1,Suelo_in,1,Columna),
     obtiene_fila(Fila2,Suelo_in,1,Columna1),

     Fila2n=Fila2+1,
     Fila1n=Fila1+1,
     Fila0n=Fila0+1,

     Fila0n<=Fila1n,
     Fila2n<=Fila1n,

     Filan=Fila1n,
     Filan<4,

     cambia_fila(Tabla_in,Suelo_in,Filan,Columna,3,Tabla_int,Suelo_int),
     Fila22=Filan+1,
     cambia_fila(Tabla_int,Suelo_int,Fila22,Columna,1,Tabla_preout,Suelo_preout),
     limpia_filas(Tabla_preout,Suelo_preout,Suelo_out,Tabla_out),
     Tablero_out=tab(Suelo_out,Tabla_out),
     !.
```

1. La función asegura que la columna en la que se centra la ficha esté dentro del rango permitido (columnas 1 a 4). Si la columna no está en este rango, la ficha no puede colocarse.
2. Para determinar si hay espacio suficiente, la función obtiene las alturas actuales del suelo en las columnas afectadas: la columna centrada (Columna), y las columnas adyacentes a la izquierda (Columna0) y a la derecha (Columna1).
3. La función comprueba que las alturas de las columnas adyacentes (izquierda y derecha) sean menores o iguales a la altura de la columna centrada, asegurando que la ficha encaje bien y no sobresalga del tablero.
4. Se calcula la fila en la que debe colocarse la ficha. La ficha de tipo 1 se coloca en la fila más baja posible, que es la altura de la columna centrada.

   Luego, se utiliza el predicado cambia_fila() para colocar las partes de la ficha en las filas correspondientes de la tabla, actualizando las posiciones del "suelo" y la "tabla".

5. Después de colocar la ficha, se recalculan las filas, y cualquier fila llena se elimina mediante el predicado limpia_filas(). El tablero finalizado se devuelve como Tablero_out.
6. Finalmente, usa ! o cut, esto asegura que una vez que se ha colocado la ficha correctamente en el tablero y se han realizado los cambios necesarios, Prolog no buscará otras maneras de colocar la ficha en la misma posición.

3. Ejecuciones de prueba

En este apartado, observaremos la ejecución del programa. Éste se basa en el uso del backtracking para colocar las fichas en el tablero de manera que se respeten las restricciones del espacio.

A través de la ejecución de las reglas, el programa determina cómo y dónde se pueden colocar las fichas. El proceso se ejecuta en varios pasos, y a medida que se prueba cada ficha, el programa verifica si la colocación es posible. Si una colocación no es válida, el programa retrocede y prueba otra opción, continuando hasta encontrar una solución viable.

1. Salida para la secuencia de fichas [1, 2, 3, 3]

En esta ejecución, el programa intenta insertar la secuencia de fichas [1, 2, 3, 3] en el tablero.

Se empieza intentando insertar la primera ficha, de tipo 1. Se prueba con todas las orientaciones posibles en la columna 1. Después de probar las orientaciones 0, 1, 2 y 3, se logra insertar la ficha T = 1, con orientación 3 en la columna 1. En este momento, se actualiza el tablero y se pasa a intentar colocar la segunda ficha, de tipo 2.

```
[Inactive C:\VIP\BIN\WIN\32\Obj\goal$000.exe]
========================================
=                ¡ JUGANDO !           =
========================================

Pruebo la ficha: T = 1 con O = 0, en C = 1
Pruebo la ficha: T = 1 con O = 1, en C = 1
Pruebo la ficha: T = 1 con O = 2, en C = 1
Pruebo la ficha: T = 1 con O = 3, en C = 1
Fila: 4    0         0         0         0         0
Fila: 3    0         1         0         0         0
Fila: 2    1         1         0         0         0
Fila: 1    0         1         0         0         0
He probado la ficha: T = 1 con O = 3, en C = 1
```

Para la ficha T = 2, se prueba con varias orientaciones en la columna 1, pero no se logra insertar hasta que se prueba en la columna 3, con orientación 0. En este caso, la ficha se inserta correctamente, y se actualiza el tablero.

```
Pruebo la ficha: T = 2 con O = 0, en C = 1
Pruebo la ficha: T = 2 con O = 1, en C = 1
Pruebo la ficha: T = 2 con O = 2, en C = 1
Pruebo la ficha: T = 2 con O = 3, en C = 1
Pruebo la ficha: T = 2 con O = 0, en C = 2
Pruebo la ficha: T = 2 con O = 1, en C = 2
Pruebo la ficha: T = 2 con O = 2, en C = 2
Pruebo la ficha: T = 2 con O = 3, en C = 2
Pruebo la ficha: T = 2 con O = 0, en C = 3
Fila: 4    0         0         0         0         0
Fila: 3    0         1         0         0         0
Fila: 2    1         1         1         1         0
Fila: 1    0         1         1         1         0
He probado la ficha: T = 2 con O = 0, en C = 3
```

A continuación, se intenta insertar la tercera ficha, T = 3. Se prueba en la columna 1, columna 2, y finalmente se encuentra una posición válida en la columna 2, con orientación 2. Esta ficha se coloca con éxito en esa posición.

```
Pruebo la ficha: T = 3 con O = 0, en C = 1
Pruebo la ficha: T = 3 con O = 1, en C = 1
Pruebo la ficha: T = 3 con O = 2, en C = 1
Pruebo la ficha: T = 3 con O = 3, en C = 1
Pruebo la ficha: T = 3 con O = 0, en C = 2
Pruebo la ficha: T = 3 con O = 1, en C = 2
Pruebo la ficha: T = 3 con O = 2, en C = 2
Fila: 4    1        1        1        0        0
Fila: 3    1        1        0        0        0
Fila: 2    1        1        1        1        0
Fila: 1    0        1        1        1        0
He probado la ficha: T = 3 con O = 2, en C = 2
```

Por último, se prueba la ficha T = 3 con todas su posibilidades. Se consigue colocar en la columna 5, con orientación 3.

```
Pruebo la ficha: T = 3 con O = 0, en C = 1
Pruebo la ficha: T = 3 con O = 1, en C = 1
Pruebo la ficha: T = 3 con O = 2, en C = 1
Pruebo la ficha: T = 3 con O = 3, en C = 1
Pruebo la ficha: T = 3 con O = 0, en C = 2
Pruebo la ficha: T = 3 con O = 1, en C = 2
Pruebo la ficha: T = 3 con O = 2, en C = 2
Pruebo la ficha: T = 3 con O = 3, en C = 2
Pruebo la ficha: T = 3 con O = 0, en C = 3
Pruebo la ficha: T = 3 con O = 1, en C = 3
Pruebo la ficha: T = 3 con O = 2, en C = 3
Pruebo la ficha: T = 3 con O = 3, en C = 3
Pruebo la ficha: T = 3 con O = 0, en C = 4
Pruebo la ficha: T = 3 con O = 1, en C = 4
Pruebo la ficha: T = 3 con O = 2, en C = 4
Pruebo la ficha: T = 3 con O = 3, en C = 4
Pruebo la ficha: T = 3 con O = 0, en C = 5
Pruebo la ficha: T = 3 con O = 1, en C = 5
Pruebo la ficha: T = 3 con O = 2, en C = 5
Pruebo la ficha: T = 3 con O = 3, en C = 5
Fila: 4    0        0        0        0        0
Fila: 3    1        1        1        0        0
Fila: 2    1        1        0        1        1
Fila: 1    0        1        1        1        1
He probado la ficha: T = 3 con O = 3, en C = 5
```

Finalmente, el programa muestra la solución completa con el estado final del tablero y el orden de las fichas colocadas.

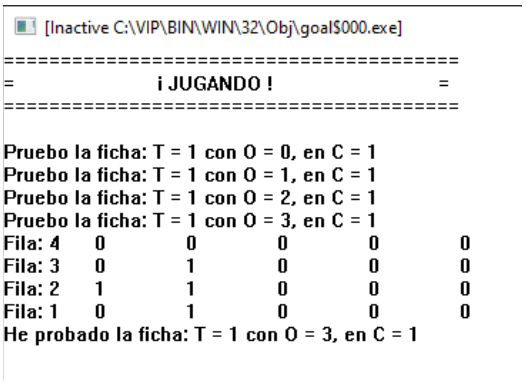
```
========================================
              SOLUCIÓN FINAL
========================================

Fila: 4    0        0        0        0        0
Fila: 3    1        1        1        0        0
Fila: 2    1        1        0        1        1
Fila: 1    0        1        1        1        1
f(1,3,1)
f(2,0,3)
f(3,2,2)
f(3,3,5)


ORDEN DE LAS FICHAS:

[f(3,3,5),f(3,2,2),f(2,0,3),f(1,3,1)]
yes
```

Podemos observar que en este caso no ha hecho falta retroceder en ningún momento, ya que no hemos llegado a ningún estado en el que no se podía insertar más fichas.

2. Salida para la secuencia de fichas [1, 2, 3, 4]

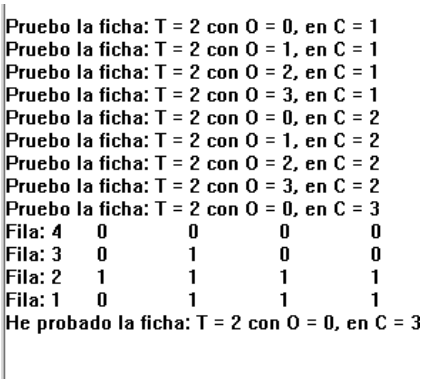
En esta ejecución, el programa intenta insertar la secuencia de fichas [1, 2, 3, 4] en el tablero.

Se empieza intentando insertar la primera ficha, de tipo 1. Se prueba con todas las orientaciones posibles en la columna 1. Después de probar las orientaciones 0, 1, 2 y 3, se logra insertar la ficha T = 1, con orientación 3 en la columna 1. En este momento, se actualiza el tablero y se pasa a intentar colocar la segunda ficha, de tipo 2.



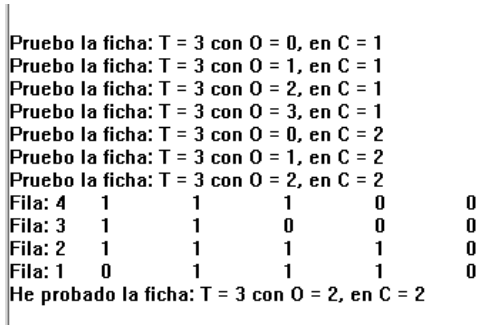
```
[Inactive C:\VIP\BIN\WIN\32\Obj\goal$000.exe]
========================================
=                ¡ JUGANDO !           =
========================================

Pruebo la ficha: T = 1 con O = 0, en C = 1
Pruebo la ficha: T = 1 con O = 1, en C = 1
Pruebo la ficha: T = 1 con O = 2, en C = 1
Pruebo la ficha: T = 1 con O = 3, en C = 1
Fila: 4    0         0         0         0         0
Fila: 3    0         1         0         0         0
Fila: 2    1         1         0         0         0
Fila: 1    0         1         0         0         0
He probado la ficha: T = 1 con O = 3, en C = 1
```

Para la ficha T = 2, se prueba con varias orientaciones en la columna 1, pero no se logra insertar hasta que se prueba en la columna 3, con orientación 0. En este caso, la ficha se inserta correctamente, y se actualiza el tablero.



```
Pruebo la ficha: T = 2 con O = 0, en C = 1
Pruebo la ficha: T = 2 con O = 1, en C = 1
Pruebo la ficha: T = 2 con O = 2, en C = 1
Pruebo la ficha: T = 2 con O = 3, en C = 1
Pruebo la ficha: T = 2 con O = 0, en C = 2
Pruebo la ficha: T = 2 con O = 1, en C = 2
Pruebo la ficha: T = 2 con O = 2, en C = 2
Pruebo la ficha: T = 2 con O = 3, en C = 2
Pruebo la ficha: T = 2 con O = 0, en C = 3
Fila: 4    0         0         0         0
Fila: 3    0         1         0         0
Fila: 2    1         1         1         1
Fila: 1    0         1         1         1
He probado la ficha: T = 2 con O = 0, en C = 3
```

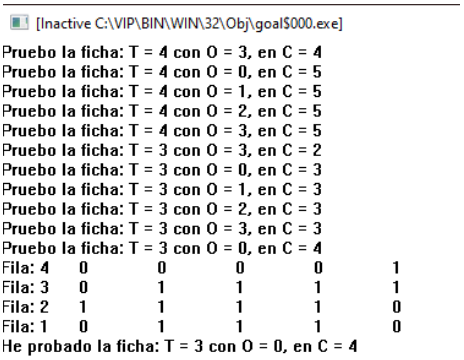
A continuación, se intenta insertar la tercera ficha, de tipo 3. Se prueba en la columna 1, columna 2, y finalmente se encuentra una posición válida en la columna 2, con orientación 2. Esta ficha se coloca con éxito en esa posición.



```
Pruebo la ficha: T = 3 con O = 0, en C = 1
Pruebo la ficha: T = 3 con O = 1, en C = 1
Pruebo la ficha: T = 3 con O = 2, en C = 1
Pruebo la ficha: T = 3 con O = 3, en C = 1
Pruebo la ficha: T = 3 con O = 0, en C = 2
Pruebo la ficha: T = 3 con O = 1, en C = 2
Pruebo la ficha: T = 3 con O = 2, en C = 2
Fila: 4    1         1         1         0         0
Fila: 3    1         1         0         0         0
Fila: 2    1         1         1         1         0
Fila: 1    0         1         1         1         0
He probado la ficha: T = 3 con O = 2, en C = 2
```

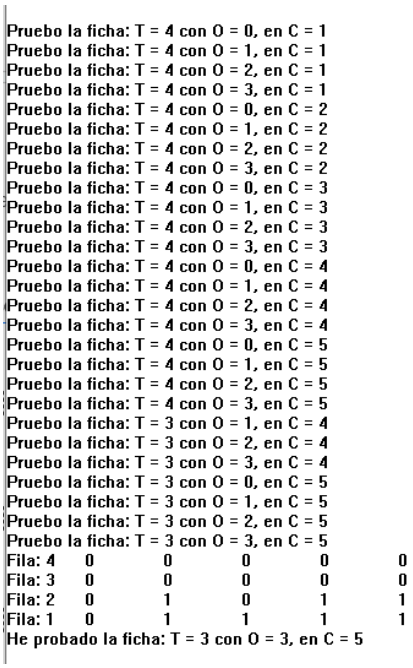
Por último, se intenta insertar la ficha T = 4 en todas las columnas y orientaciones posibles. No obstante, no encuentra una posición válida para esta ficha en ninguna de las pruebas.

Esto significa que el programa no puede insertar la ficha T = 4 en la configuración actual del tablero. Por ello, el algoritmo retrocede a la ficha anterior (T = 3) y prueba con nuevas posiciones u orientaciones, buscando una solución alternativa para insertar las fichas en el tablero.



La ficha anterior (T = 3), se coloca con éxito en la posición siguiente al último estado suyo, en la columna 4 con orientación 0.

Se vuelve a intentar colocar la ficha T = 4, pero vuelve a pasar lo mismo, no puede. Luego se retrocede de nuevo y se repite el proceso.



La ficha anterior (T = 3), se coloca con éxito en la posición siguiente al último estado suyo, en la columna 5 con orientación 3.

Tras insertar la ficha (T = 3) se vuelve a intentar meter la ficha (T = 4). Esta vez detecta una posibilidad en la columna 1 con orientación 1.

```
Pruebo la ficha: T = 4 con O = 0, en C = 1
Pruebo la ficha: T = 4 con O = 1, en C = 1
Fila: 4     0          0          0          0          0
Fila: 3     0          0          0          0          0
Fila: 2     1          1          0          0          0
Fila: 1     1          1          0          1          1
He probado la ficha: T = 4 con O = 1, en C = 1
```

Finalmente, el programa muestra la solución completa con el estado final del tablero y el orden de las fichas colocadas.

```
========================================
              SOLUCIÓN FINAL
========================================

Fila: 4     0          0          0          0          0
Fila: 3     0          0          0          0          0
Fila: 2     1          1          0          0          0
Fila: 1     1          1          0          1          1
f(1,3,1)
f(2,0,3)
f(3,3,5)
f(4,1,1)


ORDEN DE LAS FICHAS:

[f(4,1,1),f(3,3,5),f(2,0,3),f(1,3,1)]
yes
```

A diferencia de en el caso anterior, podemos observar que ha hecho falta retroceder a la ficha anterior, ya que hemos llegado a un estado en el que no se podía insertar más fichas.

Práctica 2.- Implementación en otro lenguaje de una estrategia de exploración de grafos A*.

1. Introducción

Para la implementación de la estrategia de exploración de grafos A* hemos implementado el mismo juego del tetris que teníamos anteriormente en prolog esta vez en python.

Hemos utilizado para ello una lógica muy similar a la utilizada en el programa en prolog, pero aprovechando las facilidades de python a la hora de insertar las fichas.

```python
def insertar_T1_O0(self, columna):
    """Inserta una ficha T horizontalmente (orientación 0).
        F
       FFF"""
    if 1 <= columna <= 3:

        fila1n = self.obtener_fila(columna - 1)
        fila2n = self.obtener_fila(columna)
        fila3n = self.obtener_fila(columna + 1)


        if fila1n<=fila2n and fila3n<=fila2n and fila2n <= 2:

            self.tablero[fila2n][columna - 1] = 1
            self.tablero[fila2n][columna] = 1
            self.tablero[fila2n][columna + 1] = 1

            if fila2n + 1 <= 3:
                self.tablero[fila2n + 1][columna] = 1
                self.recalcular_suelo()
                return True
            else:
                return False
        else:
            return False
```

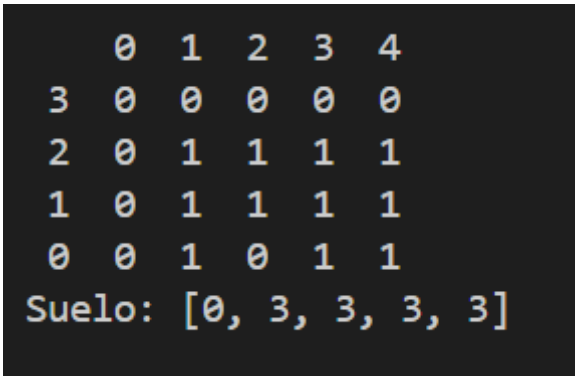
2. Heurística definida

En cuanto a la implementación del algoritmo A*, para realizarla primero hemos tenido que definir una heurística que nos ayude a establecer un “coste” para cada estado del tablero.

Hemos decidido basar dicho coste en tres factores que dificultan la posterior adición de más fichas al tablero, y por tanto harán que sea más difícil hallar una solución, estos son:

1. La **altura máxima** a la que llegan las fichas ya introducidas en el tablero, por razones obvias, ya que cuanto más alto lleguemos en el tablero menos fichas podremos meter en aquellas columnas

2. El **desnivel** entre columnas, que hace que sea más difícil introducir fichas ya que esto se facilita cuánto más plano es el suelo



(claro desnivel entre las columnas 0 y 1 que dificulta la inserción de gran número de fichas)

3. Los **huecos por debajo del “suelo”**, que son las posiciones del tablero sin rellenar que quedan tapadas por otras fichas por encima y se vuelven inaccesibles, ya que son posiciones del tablero que no estamos aprovechando y no las tendremos disponibles más tarde, lo cuál nos dificultará más encontrar una solución ya que estamos desaprovechando espacio. Además el no dejar estos huecos inaccesibles sin completar favorece enormemente que se puedan completar filas y generar así más espacio.

```
   0  1  2  3  4
3  1  1  0  0  0
2  0  1  0  0  1
1  0  1  1  1  1
0  0  1  0  1  1
Suelo: [4, 4, 2, 2, 3]
```

(varios huecos que ya han quedado rodeados por unos y se han vuelto inaccesibles, perdiendo muchos espacios para la inserción de más fichas)

Además a cada uno de estos factores le hemos asignado una importancia en base al grado en que pensamos que dificulta la inserción de fichas:

10: **huecos por debajo del "suelo"**
5: **desnivel**
2: **altura máxima**

De esta forma nuestra fórmula que le asigna su respectivo coste a un estado queda de la siguiente forma:

```
puntaje = huecos * 10 + desnivel * 5 + altura_maxima * 2
```

3. Implementación de A*

Una vez tenemos nuestra heurística claramente definida podemos pasar a desarrollar nuestro algoritmo de A* para encontrar nuestra mejor solución, es decir, en nuestro caso la que menos dificulta la futura inserción de fichas.

Para elaborar nuestro algoritmo A* lo primero que hemos hecho ha sido cargar nuestro estado inicial, el tablero vacío, en una cola de prioridad que utilizaremos durante la búsqueda de la mejor opción.

```
estado_inicial = (0, 0, [fila[:] for fila in self.tablero], self.suelo[:], [])  # (costo, índice_ficha, tablero, suelo)
cola = []
heappush(cola, estado_inicial)  # Insertar estado inicial en la cola de prioridad
```

A continuación lo sacaremos de la cola de prioridad

```
while cola:
    costo_actual, indice_ficha, tablero_actual, suelo_actual, fichas_colocadas = heappop(cola)
```

y trataremos de insertar todas las fichas en todas sus orientaciones en cada una de las columnas.

```
    print(f"Intentando colocar ficha {indice_ficha} (tipo={tipo}).")
    for col in range(5):  # Hay 5 columnas en el tablero
        for orient in range(4):  # Hay 4 orientaciones posibles

            tablero_snapshot = [fila[:] for fila in self.tablero]
            suelo_snapshot = self.suelo[:]

            if self.meter_ficha((tipo, orient, col)):
                print("================================================================================\n")
                print(f"Ficha {indice_ficha} colocada: tipo={tipo}, orientación={orient}, columna={col}")
```

En caso de encontrar una posible inserción exitosa, calcularemos el coste del nuevo estado que alcanzamos tras esa inserción. También almacenaremos para cada estado la ficha que se ha insertado para llegar a él, de esta forma cuando obtengamos la solución sabremos la secuencia de fichas que hemos tenido que insertar para alcanzarla.

```
g_n = costo_actual + 1  # Cada movimiento tiene un costo real de 1
h_n = self.heuristica()  # Heurística para este estado
f_n = g_n + h_n

print(f"Costo g(n)={g_n}, Heurística h(n)={h_n}, Costo total f(n)={f_n}")

# Agregar el vecino a la cola con su puntaje
nuevas_fichas_colocadas = fichas_colocadas + [(tipo, orient, col)]
```

El coste de dicho estado, **f_n**, lo obtenemos de la fórmula:

$$f(n) = g(n) + h(n)$$

donde **g_n** es el coste del camino real que se trata de la suma de los costes de los estados anteriores + 1, que es el coste real de cada movimiento, y **h_n** es el resultado de la heurística calculada para el nuevo estado que hemos alcanzado.

Una vez tenemos almacenado todo lo que queremos, guardamos este estado con toda su información en la cola de prioridad que utilizamos al inicio del todo

```
heappush(cola, (f_n, indice_ficha + 1, [fila[:] for fila in self.tablero], self.suelo[:],nuevas_fichas_colocadas))
print("Estado vecino agregado a la cola.\n\n")
```

Con esto tendremos todas las posibles inserciones desde el estado inicial almacenadas en la cola, cada una con su coste correspondiente.

A continuación, haremos lo mismo para todos los estados que han quedado almacenados en la cola. Al almacenarlos en la cola estos han quedado ordenados de menor a mayor coste.

```
while cola:
    costo_actual, indice_ficha, tablero_actual, suelo_actual, fichas_colocadas = heappop(cola)
```

De esta forma, después de que salga de la cola el último estado de los que habíamos obtenido a partir del inicial, ahora tendremos en la cola todos los estados posibles a los que es posible llegar desde esos mismos estados anteriores

Llegados a que de la cola salga el estado con menor coste de todos los posibles que hemos calculado para el número de fichas que hemos introducido, sacaremos este estado como solución final, ya que siguiendo nuestra heurística se trata de nuestra mejor solución.

```
    # Si se han colocado todas las fichas, se encontró una solución
    if indice_ficha == len(fichas):
        print("=====================================")
        print("SOLUCIÓN FINAL")
        print("=====================================")

        # Imprimir el tablero final
        for fila in reversed(self.tablero):
            print(f"Fila: {'\t'.join(map(str, fila))}")

        print("\nFichas colocadas:")
        for ficha in fichas_colocadas:
            print(f"Ficha: tipo={ficha[0]}, orientación={ficha[1]}, columna={ficha[2]}")
```

4. Ejecuciones de prueba

Por último mostraremos alguna ejecución de prueba para comprobar que obtenemos un resultado acorde con nuestra heurística

Probamos con la llamada al algoritmo para 1 ficha de tipo 1, 1 de tipo 2 y dos de tipo 3 ([1,2,3,3]):

```
juego = Tetris()

fichas = [1,2,3,3]
if juego.resolver_a_star(fichas):
    print("¡Se encontró una solución con A*!")
    juego.imprimir_tablero()


else:
    print("No se encontró una solución.")
```

El resultado obtenido es el siguiente:

```
======================================
SOLUCIÓN FINAL
======================================
Fila: 0 0       0       0       0
Fila: 0 1       1       1       0
Fila: 0 1       1       1       1
Fila: 0 1       1       1       1

Fichas colocadas:
Ficha: tipo=1, orientación=0, columna=1
Ficha: tipo=2, orientación=0, columna=3
Ficha: tipo=3, orientación=2, columna=3
Ficha: tipo=3, orientación=2, columna=2
¡Se encontró una solución con A*!
    0  1  2  3  4
 3  0  0  0  0  0
 2  0  1  1  1  0
 1  0  1  1  1  1
 0  0  1  1  1  1
Suelo: [0, 3, 3, 3, 2]
```

Podemos ver en nuestra ejecución que el resultado encaja bastante con nuestra heurística:

- No existen huecos por debajo del suelo que es el factor al que más importancia le hemos asignado

- A excepción del desnivel entre la columna 0 y la 1, no se observa desnivel apenas en el resto del tablero, un ligero escalón entre la 3 y la 4.

```
======================================
            SOLUCIÓN FINAL
======================================

Fila: 4    0       0       0       0       0
Fila: 3    1       1       1       0       0
Fila: 2    1       1       0       1       1
Fila: 1    0       1       1       1       1
f(1,3,1)
f(2,0,3)
f(3,2,2)
f(3,3,5)


ORDEN DE LAS FICHAS:

[f(3,3,5),f(3,2,2),f(2,0,3),f(1,3,1)]
yes
```

Al compararlo con nuestra salida de Prolog, en el que no utilizábamos A*, se ve realmente la diferencia, ya que en la obtenida por Prolog con backtrack vemos que apenas entraría una ficha más, en nuestra salida en Python con A* entraría alguna más, además de que al encontrarse las fichas más ordenadas y sin desaprovechar huecos es mucho más fácil que se completen filas, generando así más espacio.