Sokoban

Tarea 1

En esta primera tarea el objetivo es representar el nivel e identificar la función de cada caracter. Para ello se ha desarrollado una clase Map, donde se encapsulan los elementos de la cadena introducida por teclado.
Los tres módulos que se han desarrollado en esta primera tarea son `sokoban.py`, `map.py` y `parser.py`. Añadiendo la opcion 'T1' en el comando de ejecución de sokoban, muestra por pantalla la posición de cada elemento del juego.

* sokoban.py
  Este es el módulo principal, en él se recogen los parámetros introducidos por teclado y se realizan las llamadas correspondientes a las opciones elegidas.

* parser.py
  Es el encargado de parsear las opciones y parametros para encapsularlos en un objeto parser. También es el encargado de mostrar un mensaje de error en caso de no introducir bien las opciones.

* map.py
  El módulo que contiene la clase Map, dónde se desarrollan las funcionalidades del proyecto.

    - Map

        - `__init__`: Es el método constructor, requiere del parámetro "level" para inicializarse, al crear un objeto Map se inicializan los atributos `level`, `ID`, `rows`, `columns`, `wallList`,`player`, `boxList`, `targetList` y `map`.

        - `define_grid`: Cuenta el número de filas y columnas que tiene el nivel y retorna un grid inicializado con todas las posiciones a ' '. Cada caracter encontrado hasta el salto de línea (\n) es una columna, por lo tanto el número de columnas (NC) se incrementa en 1; por cada salto de línea se incrementa en 1 el número de filas (NR), también en cada salto se compara el número de columnas contado en esa iteración con el almacenado, en caso de ser mayor, se almacena en la variable maxNC. De esta manera se define un grid para no dejar ningún carácter fuera de límites.

        - `make_level`: En primera instancia llama al método `define_grid`, posteriormente itera sobre la cadena 'level', diferenciando los caracteres, asignandoles una posición (i,j) en el grid y almacenando sus posiciones en los atributos del objeto. Por último almacena el resultado en el atributo map y genera un identificador del nivel.

        - `show_map_elements`: Imprime por pantalla el identificador del nivel y las posiciones de los diferentes elementos que lo componen.

Tarea 2

El objetivo de esta tarea es identificar la función sucesor del jugador y de las cajas. Las nuevas funcionalidades se añaden en `map.py` y se descubren añadiendo la opción 'T2S' y 'T2T' en el argumento task del juego.

T2S

La acción T2S obtiene la función sucesor del nivel, para ello se han desarrollado `successors`, `get_index`, `get_moves` y `show_successors`.

* `successors`, es el método que retorna la lista final con los movimientos sucesores del nivel, diferenciando entre mayúsculas y minúsculas, cajas y jugador, respectivamente la dirección del movimiento U (Up, Arriba), R (Right, Derecha), D (Down, Abajo), L (Left, Izquierda). Cada elemento de la lista que retorna es una tupla: <direccion,nuevo_estado,coste>.

* `get_index`, por parámetros se le pasa una posición (i,j) y comprueba en la lista de las cajas el igual al parámetro, devolviendo el índice.

* `get_moves`, con la posición del jugador este método obtiene todos sus movimientos posibles. Si encuentra una caja adyacente, comprueba también si puede ser movida. Este método retorna una lista con elementos <<u|U|r|R|d|D|l|L>, posicion_sucesor(i,j) , posicion_actual(i,j)>

* `show_successors`, muestra por pantalla los sucesores del nivel.

T2T

La meta de esta tarea es mostrar el resultado de la función OBJETIVO para el nivel. Se ha desarrollado el método `objective`.

* `objective`, comprueba que no exista ningún objetivo en el nivel, en caso de que exista alguna caja ('$') se considerará que el objetivo no está cumplido. Muestra por pantalla el resultado.

Tarea 3

T3

Esta tarea crea el árbol de búsqueda hacia la solución del nivel. Se ha desarrollado una clase `Node` y nuevas funcionalidades en `map.py`. Se han creado nuevos atributos de la clase `Map`, `dictionary`, que es un diccionario que almacena el estado (clave) y las posiciones del jugador y las cajas (valor) y `empty_map` que es el mapa sin el jugador y las cajas
Al ejecutar el nivel en la paleta de comandos se debe añadir el nivel, la estrategia a utilizar (BFS, DFS, UC) y el máximo de profundidad a la que se quiere llegar.

- Node
  La clase Node encapsula los siguientes atributos: `node_ID`, `state`, `parent`, `action`, `depth`, `cost`, `heuristic` y `value`. También dispone de métodos:
  - `calculate_value`: Calcula el valor (`value`) del nodo dependiendo de la estrategia utilizada.
  - `check_visited`: Comprueba que el nodo no ha sido visitado con anterioridad.
    Las nuevas funcionalidades desarrolladas en `map.py` son las siguientes:

* `solve_sokoban`: Contiene el algoritmo de búsqueda para la solución del nivel, se utilizan 2 listas `fringe`, en la que se añaden los nodos a visitar y `visited`, en la que se añaden los estados de los nodos ya visitados. Para llevar a cabo el algoritmo lo primero de todo es añadir el nodo cero a la frontera (estado inicial del nivel), mientras la frontera no este vacía y no sea solución se ejecuta un bucle while, se saca el primer elemento de la frontera y se actualiza el mapa con las posiciones del nodo, posteriormente se comprueba si es objetivo, en caso de que sí, el nodo es solución y finaliza el bucle para después realizar el camino solución, en el caso de que no, se comprueba que el nodo no esté visitado y que la profundidad sea la adecuada, cumpliéndose esta condición el estado del nodo se añade a `visited`, se expande y estos nuevos nodos se añaden a la frontera. En caso de no encontrar la solución se mostrará un mensaje por pantalla. Para el algoritmo DFS, los nodos expandidos se colocan en primera posición, mientras que para el nodo DFS y UC se colocan después de los que ya están en `fringe`.

* `make_path`: Construye el camino solución, desde el nodo objetivo el método comprueba que el atributo `parent` no es None, es decir, que no es el nodo cero. En cada iteración se añade el nodo a la lista path, cuando `parent` es None, se añade por útlimo este nodo y se llama a imprimir por pantalla.

* `print_path`: Formatea e imprime por pantalla el resultado de la búsqueda.

* `expand`: A partir de los sucesores del nivel genera los nuevos nodos de expansión, por cada sucesor, comprueba que no está visitado y genera un nuevo nodo con un identificador único.

* `blank_map`: Inicializa la variable `empty_map`.

* `update_map`: Obtiene las posiciones del estado del nodo en el diccionario, actualiza el valor actual del mapa a vacío para posteriormente actualizar con las nuevas posiciones en el mapa.

Tarea 4

Se implementan los algoritmos voraz y A*, para utilizarlos se deben añadir al igual que en la anterior tarea a través de la terminal (GREEDY, A*, respectivamente). En el algoritmo que construye el árbol (`solve_sokoban`) se añade una condición para ordenar los nodos por el valor más bajo.
En el módulo `node.py` se han implementado las nuevas funcionalidades, nueva forma para calcular el valor siguiendo la heurística y el algoritmo especifiado en `calculate_value()` y una nueva función: `heuristic_function`.

* `heuristic_function`: Para calcular la heurística se suma la distancia mínima de manhattan de todas las cajas a los objetivos, por parámetros se le pasa la `boxList` y la `targetList`. La lista de las cajas que se le pasa es la de la función sucesora. La fórmula para calcular la distancia es la siguiente: $Dmanhattan((fila,columna))=|Tfila-Bfila|+|Tcolumna-Bcolumna|$