Hnefanaves

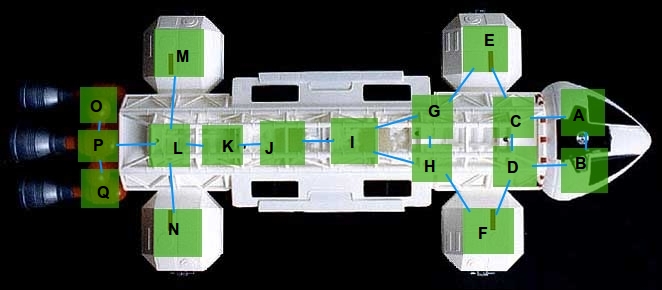
Hnefatafl es un antiguo juego vikingo. Hnefanaves es un moderno juego de naves, que no tiene nada que ver con Hnefatafl.

En este moderno juego, el jugador posee una nave con diversos slots (espacios) disponibles en los cuales se pueden conectar diferentes módulos útiles. Algunos módulos sirven para combatir, otros para explorar, etc. La selección de módulos tiene un gran impacto en la capacidad del jugador para enfrentarse a las misiones que el juego le presentará, y por eso es necesario planificar con cuidado cómo se equipará a la nave. A pesar de ello, no hay un modelo de nave "perfecta", cada nave tiene sus fortalezas y debilidades de acuerdo a cómo se encuentra equipada, y su performance dependerá mucho de las misiones que (al azar) le toquen al jugador.

La lista de módulos disponibles es la siguiente:

* Lasers.
* Motores (cohetes).
* Cabinas para tripulantes.
* Bahías de carga.
* Sistemas de vida extraterrestre.
* Escudos.
* Baterías.

Y la nave tiene los siguientes slots disponibles (las 17 zonas marcadas en verde, con letras para identificarlas):



Las lineas celestes representan las conexiones que existen entre los módulos (algo importante a la hora de construir la nave).

El jugador debe ubicar un módulo en cada slot (sí o sí, no se pueden dejar slots desocupados), y debe respetar una serie de reglas:

* Por el calor que los lasers generan al disparar, no puede haber baterías conectadas a lasers, de lo contrario podrían explotar.
* Los motores solo pueden ubicarse en los slots traseros o en los 4 slots laterales (**no** quiere decir que sí o sí esos slots tengan que tener motores. Pueden tener otras cosas).
* Los sistemas de vida extraterrestre permiten que una cabina conectada albergue vida no humana, y por tanto sí o sí tienen que ubicarse conectados a cabinas.
* Las cabinas no pueden estar conectadas a los motores, ya que el ruido que generan los mismos impediría el correcto trabajo de los pilotos.
* Los escudos generan un campo electromagnético tan fuerte, que produce interferencias con los sistemas de vida extraterrestre, y por ende no pueden estar conectados entre si.
* Las bahías de carga tienen que tener al menos una cabina conectada, para permitir inspecciones de seguridad.
* Las baterías tienen que tener al menos dos sistemas que consuman baterías conectados, para no desperdiciar energía. Los sistemas que consumen baterías son: Lasers, Cabinas de tripulantes, Escudos y Sistemas de vida extraterrestre.
* No es posible instalar dos módulos iguales conectados entre sí, por motivos científicos extremadamente complicados de explicar.

Ejercicios:

1. Implementar la formulación del problema como problema de satisfacción de restricciones para ser resuelto con SimpleAI, incluyendo definición de variables, dominios, restricciones y funciones utilizadas para chequeo de las restricciones.
2. Ejecutar los siguientes métodos de búsqueda, y por cada uno indicar la solución encontrada:

* Caso 1: Backtracking search, utilizando la combinación de heurísticas que ustedes prefieran.
* Caso 1: Min conflicts.

Formato de entrega:

La resolución del ejercicio debe realizarse en un archivo llamado entrega\_2.py, que debe ser subido a la raíz del repositorio git/mercurial del grupo.

El módulo debe tener una función llamada resolver, que reciba los siguientes parámetros:

* metodo\_busqueda: el nombre del método de búsqueda a ejecutar, como string con nombres **exáctamente** iguales a los nombres de las funciones en SimpleAI (ej: 'backtrack', o 'min\_conflicts').
* iteraciones: un número entero indicando el límite de iteraciones para el algoritmo cuando se utilice el algoritmo min\_conflicts (para las demás búsquedas, el parámetro recibirá un None).

Al llamar a esta función, se debe ejecutar la búsqueda especificada y devolver el **diccionario** resultante (lo que devuelve el método de búsqueda de SimpleAI).

Los resultados de los casos deben subirse también al repositorio en un archivo llamado entrega\_2.txt con una linea por cada caso, con el siguiente formato para cada una:

numero\_de\_caso:resultado\_obtenido

Donde resultado\_obtenido es lo que devuelve llamar a la función str con el diccionario que SimpleAI retornó.

Ejemplo:

1:{'variable\_x': 'valor\_1', 'variable\_y': 'valor\_2', ...}

2:{'variable\_x': 'valor\_5', 'variable\_y': 'valor\_8', ...}

En esta entrega, **ninguno** de los métodos debería bloquearse en un bucle infinito o quedarse sin memoria. Lo peor que puede suceder es que demore demasiado por haberlo programado de alguna forma excesivamente compleja.

**Respetar** nombres de archivos, funciones, parámetros y tipos de datos **exáctamente** como se dicen en este enunciado. Cualquier falla por no respetar la interfaz definida, se considera no entregado.

Si quieren probar que lo están haciendo de manera correcta, pueden descargar el script llamado probar\_entrega\_2.py del repositorio de la materia (directorio 2016), y luego de posicionarlo en **el mismo** directorio que su entrega, ejecutarlo de esta forma:

python probar\_entrega\_2.py

Si eso no funciona, pueden estar seguros de que algo no están haciendo bien. En los casos de error más comunes, el script puede explicarles lo que están haciendo mal. En casos más raros, no tanto. **Recuerden que pueden preguntar en el grupo todo lo que necesiten!**