Hnefatafl

Hnefatafl es un antiguo juego que los vikingos jugaban en Inteligencia Artificial cuando tenían que hacer la Entrega 1 Tradicional ;) Las reglas de juego son las mismas, solo que en esta ocasión vamos a jugar con el equipo opuesto: los soldados.

Como los soldados no pueden moverse, lo que trataremos de resolver en esta entrega es el problema de encontrar la mejor distribución posible para nuestros soldados. Una distribución "buena" es aquella que le complica la escapatoria al rey, mientras que una "mala" distribución es aquella en la que el rey puede escapar muy fácilmente.

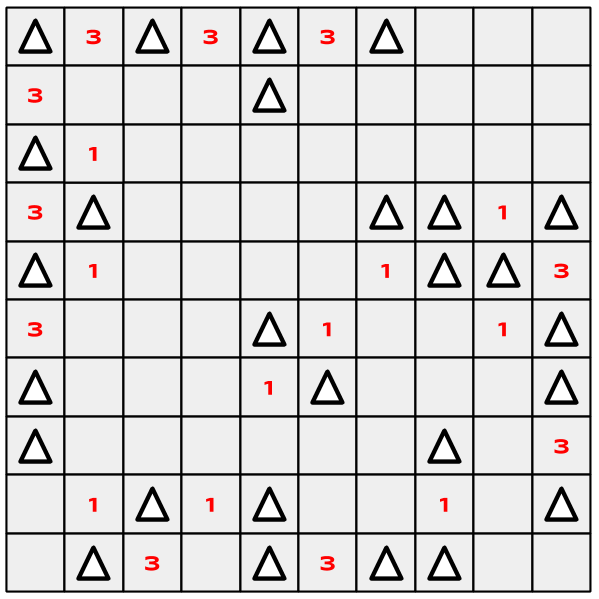
Para evaluar qué tan buena es una distribución, usaremos la siguiente lógica:

* Si un casillero tiene uno o ningún soldado atacándolo (adyacente), entonces ese casillero **no** suma puntos.
* Si un casillero es atacado por más de un soldado, y no es un casillero de los bordes, entonces ese casillero suma **1 punto**.
* Si un casillero es atacado por más de un soldado, y además se encuentra en un borde, entonces ese casillero suma **3 puntos**.
* Los casilleros con soldados dentro **no** suman puntos.
* El puntaje total de una distribución, es la suma de los puntajes de todos los casilleros.

Lógicamente, lo que buscamos es encontrar la distribución que más puntos obtenga.

El tablero sigue teniendo las dimensiones de 10x10, y siguen aplicando las mismas reglas de ataque y no superposición de fichas. Y para este problema, debemos ubicar **30 soldados** en el tablero.

Ejemplo de puntaje:



**Puntaje total**: 40.

Para los casos donde se debe proveer un estado inicial, todos los soldados deben comenzar "amontonados" en las 3 filas superiores.

Ejercicios:

1. Implementar la formulación del problema como problema de búsqueda local para ser resuelto con SimpleAI, incluyendo definición de la clase problema y sus métodos: actions, result, value y generate\_random\_state.
2. Ejecutar los siguientes métodos de búsqueda, **10 veces cada uno**, y por cada uno indicar el puntaje de la mejor solución encontrada:

* Caso 1: Búsqueda de ascenso de colina, con límite de 5000 iteraciones.
* Caso 2: Búsqueda en ascenso de colina, variante estocástica (hill\_climbing\_stochastic), con límite de 5000 iteraciones.
* Caso 3: Búsqueda de haz local, con haz de tamaño 100 y límite de 5000 iteraciones.
* Caso 4: Búsqueda de ascenso de colina con reinicios aleatorios, con 100 reinicios y límite de 5000 iteraciones.
* Caso 5: Búsqueda de temple simulado, con límite de 5000 iteraciones.

Formato de entrega:

La resolución del ejercicio debe realizarse en un archivo llamado entrega\_1\_local.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: 'beam', o 'hill\_climbing\_stochastic').
* iteraciones: un número entero indicando el límite de iteraciones para el algoritmo.
* haz: un número entero indicando el tamaño del haz para la búsqueda de haz local (para las demás búsquedas, el parámetro recibirá un None).
* reinicios: un número entero indicando la cantidad de reinicios para la búsqueda de ascenso de colina con reinicios aleatorios (para las demás búsquedas, el parámetro recibirá un None).

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

Las estadísticas deben subirse también al repositorio en un archivo llamado entrega\_1\_local.txt con una linea por cada caso, con el siguiente formato para cada una:

numero\_de\_caso:mejor\_puntaje\_obtenido

Ejemplo:

1:13

2:42

3:...

...

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\_1\_local.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\_1\_local.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!**

Notas útiles:

* El WebViewer es muy útil para probar y visualizar cosas, encontrar problemas, etc. Pero **recuerden** desactivarlo para la versión entregada, de lo contrario cuando la corrección automática trate de llamar a los algoritmos, se va a quedar tildada esperando.
* También **recuerden** que el módulo no debe ejecutar ninguna búsqueda al ser importado. Para ello, utilicen el "truco" del if \_\_name\_\_ == '\_\_main\_\_': que vimos en clases (está en el ejemplo del repo).