INFORMÁTICA III - 2021
FISICC

PROYECTO I – RECURSIVIDAD

Objetivo: Poner en práctica los conocimientos que poseen sobre recursividad para solucionar problemas que serían muy complejos de resolver de otra forma.

Objetivos Secundarios:

- Utilizar un nuevo enfoque para plantear problemas de Ciencias de la Computación.
- Familiarizarse con la metodología de calificación que se utilizará en los proyectos del curso.

Descripción General: Usted es un tripulante del U.S.C.S.S. Nostromo, y su nave se ha quedado varada en un planeta desconocido. Mientras se hacen las reparaciones, usted y su equipo van a explorar una serie de cuevas cercanas a su lugar de aterrizaje. Al adentrarse en estas cuevas, descubren un nido de huevos de tamaño considerable, y, a pesar de las advertencias de todos los demás, uno de los miembros del escuadrón se acerca demasiado. Todos ven como el huevo se abre y una criatura *parecida a una araña* se lanza a la cara de su compañero. Usted, gracias a su pasado practicando artes marciales mixtas y defensa personal, toma a su compañero rápidamente y logran evitar contacto con la criatura por milímetros. Mientras tanto, como el escuadrón va preparado para este tipo de cosas, lanzan bombas incendiarias a la criatura, pero debido a todo el alboroto, parece que han captado la atención de una criatura más grande y completamente desarrollado. Entonces deciden ejecutar el plan de emergencia más sensato: correr, esconderse y tratar de escapar con vida.

Dos horas después, ustedes han logrado contactar con la nave, que ha terminado las reparaciones, y han desarrollado un plan de escape: La nave se posará sobre las cuevas, en un punto donde el techo esta derrumbado y hay espacio para hacer descender una escalera. Gracias a la alta tecnología que ustedes poseen, han logrado crear un mapa de las cuevas, y saben exactamente dónde están ustedes, donde está la escalera hacia la nave, y que secciones de las cuevas están interconectadas entre sí. Sin embargo, no han logrado detectar exactamente donde se encuentra el alien que los está persiguiendo, pero tienen una idea aproximada de donde estará. Entonces, para maximizar sus probabilidades de llegar a la nave con vida, deciden **calcular todas las rutas posibles desde su punto de partida hasta la salida, tomando en cuenta que la proximidad del alien limitará la cantidad de tiempo que tienen para llegar a su destino**.

Fecha de Entrega: 28 febrero 2021

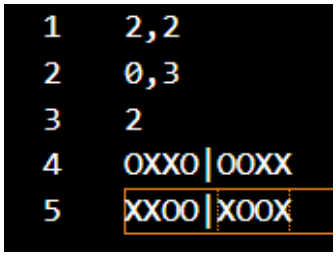
Método de Entrega: La entrega se realizará mediante el GES en la asignación respectiva en **grupos de 3 integrantes**.

Creen una carpeta llamada PJ1_CARNÉ_CARNÉ2_CARNÉ3 (ejemplo: PJ1_10000000_10000001_10000200) y dentro de ella coloquen únicamente los archivos *.java* que utilizaron.

Compriman la carpeta en formato *.zip* y súbanla al GES. Deben asegurarse de que todos los archivos compilan correctamente. **Proyecto que no compila implica cero.**

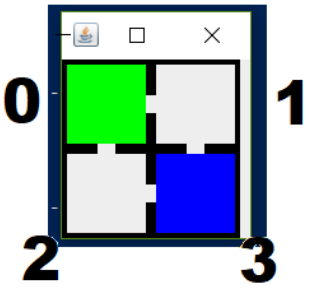
Especificaciones:

Tendremos 10 rondas de juego, cada una tendrá un valor de 10 puntos. En cada ronda se les proveerá un mapa que representan las cuevas donde están atrapados usted y su equipo, su ubicación inicial, su destino y las secciones de las cuevas que están conectadas entre sí. Los mapas se los proveeremos en un archivo *.txt*, y también les damos una clase ya creada que se encarga de leer el archivo, generar el mapa y darles una representación visual del mismo. En otras palabras, no es necesario que sepan el formato del archivo de texto, pero se los describiremos a continuación por si en algún momento les es útil:



La primera línea contiene las dimensiones del mapa (ancho, alto), en este caso: 2x2. La segunda línea contiene, de primero la sección donde se encuentra la salida, y como segundo número, la sección donde se encuentran ustedes. La siguiente línea contiene la cantidad máxima de movimientos que ustedes pueden dar para lograr llegar a la salida, que representan el tiempo que se tomarán en llegar a la salida. **Consideramos un movimiento como pasar de una sección de las cuevas a otra sección diferente**. Las siguientes líneas representan las conexiones entre las secciones de las cuevas. Una sección puede estar conectada a otras cuatro secciones: al norte, al sur, al este y al oeste. Si una sección A esta conectada al sur a una sección B, entonces esta sección B siempre tendrá una conexión al norte a la sección A. Las conexiones las damos por 4 caracteres. El primer carácter representa la conexión al norte, el segundo representa la conexión al este, el tercero la conexión al sur y el cuarto la conexión al oeste de una única sección de las cuevas. *O* significa que no hay conexión, y *X* significa que si la hay. Cada sección de las cuevas está representada por un numero secuencial, empezando con el 0, de izquierda a derecha, de arriba abajo.

Les proveemos la clase Maze.java, que se encarga de leer este archivo, construir el mapa respectivo, y si lo desean, mostrarles una representación gráfica del mismo. Para poder ver el mapa de forma gráfica, deben ejecutar:
"***java Maze path_archivo"***



Donde path_archivo es el path de un archivo con el formato del mapa que ya establecimos. Por ejemplo, si ejecutamos ***java Maze tests/test-3.txt,*** veremos lo siguiente:

Aquí podemos ver en verde la salida, en azul nuestro punto de partida, y todas las conexiones entre las diferentes secciones del mapa. También agregamos los números que representan cada una se las secciones del mapa.

Dentro de la clase Maze.java tienen una serie de métodos que les serán de ayuda:

- public int getWidth(): devuelve el ancho del mapa
- public int getHeight(): devuelve el alto del mapa
- public int getExitSpace(): devuelve un entero que representa la sección del mapa donde esta la salida (en el caso del ejemplo de arriba, devolvería 0)
- public int getStartSpace(): devuelve un entero que representa la sección del mapa donde se encuentran inicialmente usted y su equipo (en el caso del ejemplo de arriba, devolvería 3)
- public int getMaxMoves(): devuelve la cantidad de movimientos máximos que pueden realizar para escapar de las cuevas
- public int moveNorth(int seccionActual): Devuelve un entero que representa la sección de las cuevas a la que ustedes se trasladarían si parten de *seccionActual* hacia el norte. Si no hay una conexión hacia el norte, devuelve *seccionActual*. Por ejemplo, si ejecutan *moveNorth(3)*, obtienen como resultado 1, porque la sección 3 tiene la sección 1 al norte. Pero si ejecutan *moveNorth(1)*, obtienen también como resultado 1, porque no hay una conexión desde la sección 1 hacia el norte.
- public int moveEast(int seccionActual): Devuelve un entero que representa la sección de las cuevas a la que ustedes se trasladarían si parten de *seccionActual* hacia el este. Si no hay una conexión hacia el este, devuelve *seccionActual*. Por ejemplo, si ejecutan *moveEast(2)*, obtienen como resultado 3, porque la sección 2 tiene la sección 3 al este. Pero si ejecutan *moveEast(3)*, obtienen también como resultado 3, porque no hay una conexión desde la sección 3 hacia el este.
- public int moveSouth(int seccionActual): Devuelve un entero que representa la sección de las cuevas a la que ustedes se trasladarían si parten de *seccionActual* hacia el sur. Si no hay una conexión hacia el sur, devuelve *seccionActual*. Por ejemplo, si ejecutan *moveSouth(1)*, obtienen como resultado 3, porque la sección 1 tiene la sección 3 al sur. Pero si ejecutan *moveSouth(3)*, obtienen también como resultado 3, porque no hay una conexión desde la sección 3 hacia el sur.

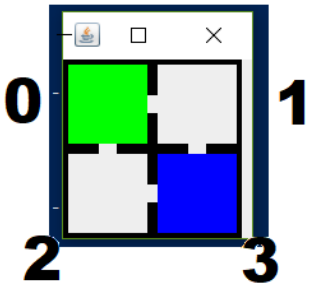
- public int moveWest(int seccionActual): Devuelve un entero que representa la sección de las cuevas a la que ustedes se trasladarían si parten de *seccionActual* hacia el oeste. Si no hay una conexión hacia el oeste, devuelve *seccionActual*. Por ejemplo, si ejecutan *moveWest(3)*, obtienen como resultado 2, porque la sección 3 tiene la sección 2 al oeste. Pero si ejecutan *moveWest(2)*, obtienen también como resultado 2, porque no hay una conexión desde la sección 2 hacia el oeste.

¿Qué deben Implementar?

El único método que es requerido implementar es el método ***public String[] solve(Maze maze)*** de la clase Solver.java (sin embargo usted puede implementar todos los métodos y clases adicionales que desee). Aquí ustedes recibirán el mapa en un objeto de tipo *Maze*, al que pueden aplicarle todos los métodos descritos anteriormente. El valor de retorno de este método es un arreglo de Strings, cada uno de estos Strings debe contener todas las secciones por las que deben pasar, empezando en la sección inicial, y terminando en la salida. Estos Strings deben empezar con "[", seguido de un numero que representa la sección, luego una coma, luego un espacio, luego el siguiente numero que representa la siguiente sección, otra coma, otro espacio, y así hasta terminar con "]". Por ejemplo, para el mapa descrito en los ejemplos anteriores, esta sería una solución con formato valido:

[3, 2, 0]

Una solución es válida si cumple con el formato establecido y si parte del punto inicial y llega al punto final en un numero menor o igual a la cantidad de movimientos establecidos. No importa si regresan a secciones ya visitadas anteriormente. Por ejemplo, para el caso de este mapa (que es el mismo mapa que hemos estado utilizando para los ejemplos anteriores), que tiene cantidad máxima de movimientos como 4, tenemos las siguientes soluciones:




```
1   [3, 1, 0]
2   [3, 1, 3, 1, 0]
3   [3, 1, 3, 2, 0]
4   [3, 2, 0]
5   [3, 2, 3, 1, 0]
6   [3, 2, 3, 2, 0]
```

Note que no hay ninguna solución repetida. Todas las soluciones empiezan en 3, todas terminan en 0, ninguna tiene un 0 en otra posición que no sea la final (porque no tendría sentido llegar a la sección de la salida y regresar a otra sección adyacente) y en todas realizamos como máximo 4 movimientos. Vean que en la sexta solución partimos de 3, pasamos a 2, y luego regresamos a 3 de nuevo, luego de nuevo a 2 y por último a la salida. El orden en que ustedes encuentran las soluciones no importa. Por ejemplo, la solución [3, 1, 0] podría estar debajo de la solución [3, 1, 3, 1, 0] y no habría problema al momento de calificar.

La nota que ustedes saquen en cada ronda dependerá de cuantas soluciones correctas del total de soluciones posibles han descubierto ustedes. Por ejemplo, si hay 6 soluciones correctas y ustedes descubrieron 3 correctas tendrán el 50% de esa ronda.

Autograder:

Para este, y los demás proyectos del curso, estaremos utilizando autograders como método de calificación. Es por esto por lo que tienen que seguir correctamente las instrucciones sobre los métodos que tienen que implementar. Si no lo hacen así, el autograder podría calificar como incorrecto algo que estuviera correcto, o incluso podría no funcionar. Con el objetivo de evitar esto, y también para que ustedes tengan una idea de la nota que obtendrán en el proyecto, les proporcionamos, la clase Autograder, junto con los archivos que contienen la solución a los tests que les proveemos. Esto significa que, si sacan 100 con el autograder que les daremos, es casi seguro que sacarán 100 cuando los califiquemos, (al igual que si obtienen 50, es muy probable que esa sea la nota que les coloquemos nosotros). Una vez crean que tienen la solución correcta al problema, pueden ejecutar el autograder de la siguiente manera:

"***java Autograder [directorio_archivos_test] [directorio_archivos_solucion]***" donde *directorio_archivos_test* y *directorio_archivos_solucion* son parámetros opcionales, que tienen como valor por defecto la carpeta de *test* y la carpeta de *solutions* respectivamente. Si no cambian esas carpetas de lugar, pueden simplemente ejecutar el autograder de esta manera: "***java Autograder"***

```
PS D:\Auxiliaturas\INFOIII_2020\pj1\Complete> java Autograder .\tests\ .\solutions\
Utilizando directorio especificado para tests: '.\tests\'
Utilizando directorio especificado para soluciones: '.\solutions\'
El directorio de tests contiene 10 archivos
----------------------------------------------------------------
Analizando archivo '.\tests\/test-0.txt'
Soluciones encontradas: 1
Soluciones esperadas: 1
Puntaje Obtenido: 10.0/10
----------------------------------------------------------------
Analizando archivo '.\tests\/test-1.txt'
Soluciones encontradas: 1
Soluciones esperadas: 1
Puntaje Obtenido: 10.0/10
----------------------------------------------------------------
Analizando archivo '.\tests\/test-2.txt'
Soluciones encontradas: 2
Soluciones esperadas: 2
Puntaje Obtenido: 10.0/10
----------------------------------------------------------------
Analizando archivo '.\tests\/test-3.txt'
Soluciones encontradas: 6
Soluciones esperadas: 6
Puntaje Obtenido: 10.0/10
----------------------------------------------------------------
Analizando archivo '.\tests\/test-4.txt'
Soluciones encontradas: 6
Soluciones esperadas: 6
Puntaje Obtenido: 10.0/10
----------------------------------------------------------------
Analizando archivo '.\tests\/test-5.txt'
Soluciones encontradas: 6
Soluciones esperadas: 6
Puntaje Obtenido: 10.0/10
----------------------------------------------------------------
Analizando archivo '.\tests\/test-6.txt'
Soluciones encontradas: 155
Soluciones esperadas: 155
Puntaje Obtenido: 10.0/10
----------------------------------------------------------------
Analizando archivo '.\tests\/test-7.txt'
Soluciones encontradas: 1403
Soluciones esperadas: 1403
Puntaje Obtenido: 10.0/10
----------------------------------------------------------------
Analizando archivo '.\tests\/test-8.txt'
Soluciones encontradas: 126
Soluciones esperadas: 126
Puntaje Obtenido: 10.0/10
----------------------------------------------------------------
Analizando archivo '.\tests\/test-9.txt'
Soluciones encontradas: 1489
Soluciones esperadas: 1489
Puntaje Obtenido: 10.0/10
----------------------------------------------------------------
----------------------------------------------------------------
Puntaje Obtenido: 100.0/100.00
PS D:\Auxiliaturas\INFOIII_2020\pj1\Complete>
```

Notas Finales:

- La forma más fácil de resolver este problema es con recursión.
- Los archivos de las carpetas *tests* y *solutions* son archivos de texto, ustedes pueden abrirlos y ver el contenido sin problemas. Para la calificación final utilizaremos diferentes mapas, así que no busquen llegar a una solución que obtenga 100 en este autograder, sino a **una solución que resuelva el problema para cualquier tipo de mapa**.
- Si tienen suficientes movimientos, una solución válida puede involucrar regresar a una sección que ya habían visitado anteriormente antes de dirigirse a la salida, recuerden que el alien es impredecible, entonces deben dejar abierta la posibilidad a regresar a una sección ya visitada.
- No es necesario que ustedes sepan exactamente como se representa el mapa en los archivos de texto. Esa información la colocamos únicamente como referencia.
- Las verificaciones de errores las dejamos a su criterio; al momento de evaluar, ningún test está diseñado para hacer que su programa falle, y todos los mapas serán siempre correctos.
- **El ultimo test es considerablemente más tardado que los anteriores**.
- Pueden utilizar la clase Maze para ver una representación gráfica del mapa.
- Luego de la entrega del proyecto, les pasaremos un pequeño quiz donde les preguntaremos como hicieron el proyecto. **Su nota final será la nota que sacaron en la calificación del autograder multiplicada por la nota que sacaron en el quiz, y dividida dentro de 100**. En el quiz no haremos ninguna pregunta muy específica sobre el código, así que, si ustedes trabajaron conscientemente en el proyecto, les será fácil sacar 100 en el quiz.

Copia: En caso de copia se recurrirá al Reglamento de Universidad Galileo. Artículo 71.1 en la primera incidencia o segunda incidencia 71.2.