**Algoritmo A\***

**Hecho por Iván González Ramiro y Michael Paredes Sanchez**

Nuestra implementación del algoritmo A\* está hecho en java y tiene implementada una interfaz gráfica con swing. En ella puedes seleccionar las columnas y filas que quieras(entre 5 a 15).

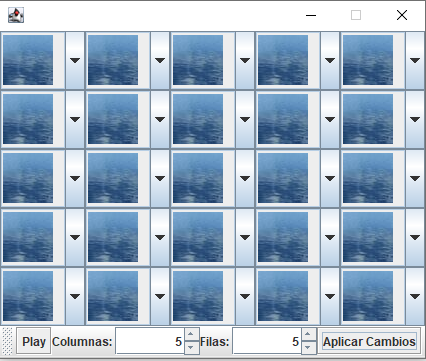
Nuestra implementación del A\* tiene 4 clases,

* Coordenada: con dos int: x e y, con getters, setters, toString, equals y hashcode (nos sirven para hacer un Mapa con las coordenadas y la imagen asociada a la coordenada) y un método para calcular la distancia a otra coordenada.
* Nodo: con una coordenada, dos doubles g (camino acumulado) y h (coste estimado a destino), además de un Nodo padre(representa cual es el Nodo previo, null en caso de que sea el inicio). Tiene un método que te da los nodos vecinos sin salirse del plano e inicializa los valores de los nuevos nodos. Además de un método que recorre todos los nodos a través del Nodo padre y te los da en una lista(lo usamos para mostrar el camino que te lleva al final)
* Ventana: Es la interfaz de la aplicación, las imágenes están explicadas en el manual de uso.
* Algoritmo A\*: Tiene los atributos int filas, int columnas, HashMap coordenadas(que tiene asociado la coordeneda con el tipo de casilla), coordenada inicio, fin y ArrayList<Coordenada> waypoints que se los da la ventana, riesgo se calcula con la diagonal principal, mientras que cerrados y abiertos se reinician cada vez que se lanza el método de NextCheckPoint y recorridos se va actualizando en NextCheckPoint.
  + NextCheckPoint: Mira los nodos alrededor del nodo inicio y luego va mirando si esos nodos nuevos no se han visto(no los añade a abiertos a menos que su nuevo total sea menor que el anterior, es decir se ha llegado a este nodo de forma más eficiente por otro camino), si son un bloqueo, si es la meta(cuando h sea 0) o si es un riesgo(entonces añade una 10% de la longitud de la diagonal principal), y termina cuando llega a la nueva meta o cuando abiertos se vacía.
  + Resolver: Comprueba que hay waypoints, en caso de que no los haya lanza NextCheckPoint con inicio y fin, en caso de haberlos va ejecutando NextCheckPoint con cada waypoint y cuando encuentra el camino al nuevo waypoint mira ya han pasado por otros nodos con el HashMap recorridos e imprime el camino a la solución, además también te dice la distancia y las casillas peligrosas que ha recorrido .
* Ampliaciones:
  + Waypoints: Hemos añadido waypoints, se recorren de izquierda a derecha y de arriba hacia abajo. En caso de que se pase por una casilla que sea un waypoint y no sea el objetivo de NextCheckPoint se dará por válida y no se volverá a pasar por ella. Si hay solución posible terminará en la meta incluso si ya ha pasado por ella mientras iba a por un checkPoint.
  + Casillas de riesgo: Se han añadido las casillas de riesgo, que al pasar por ellas suman un 10% de la distancia de la diagonal mayor, además al finalizar te dice cuantas ha pisado para llegar a la meta y cuánta penalización le ha dado.

**Manual de uso**

Ejecute “*GUI*” como java application.

Seleccione el tamaño que quieras y le das a aplicar cambios y se redimensiona y reinicia todas las casillas al estado vacío.



Las imágenes usadas son de las películas de Cómo Entrenar a tu Dragón, y cuando seleccionas una casilla puedes poner de que tipo quieres que sea, una vez hayas diseñado el mapa le das al botón de play te dará el resultado. Cada casilla es un elemento distinto. Tenemos 6 tipos de casillas:

 Son las casillas vacías.

 Es la casilla de inicio.(solo puede haber una, sino te sale un mensaje indicando que hay mas de un inicio y no se ejecuta si das a play)

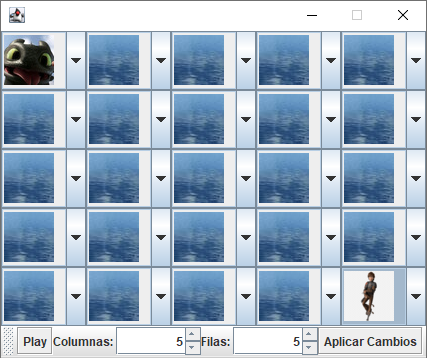
 Es la casilla de final(solo puede haber una, sino te sale un mensaje de que existe más de un final y no se ejecuta si das a play)

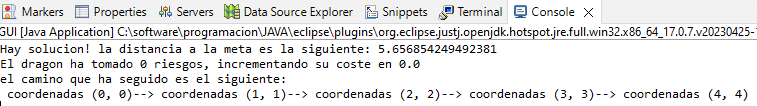
 Es la casilla de bloqueo, no se puede pasar por aquí.

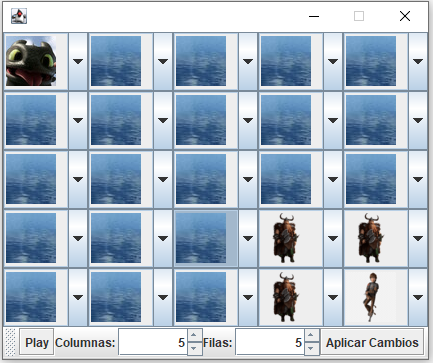
 Es un waypoint, puedes poner todos los que quieras.

Es una casilla de riesgo, pone una penalización al recorrerla.

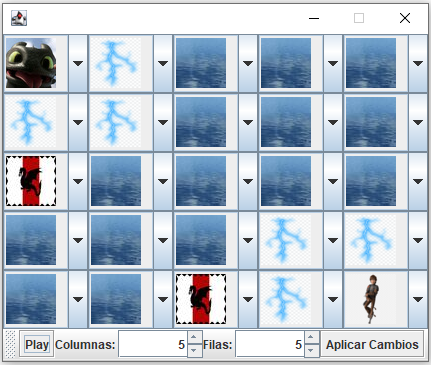
Casos de Ejemplo

Input:  


Output:

Input:

Output:

Input:

Output:

