Francisco Núñez Sierra y Jose Juan Peña Gómez
**Grupo:** JJyFarn

# DESARROLLO DE PROGRAMAS

*Curso 2017/18*

**Grado en Ingeniería Informática en**
**Ingeniería del Software**
**Ingeniería de Computadores**

# DOCUMENTACIÓN EXTERNA DEL PROYECTO

## INDICE

# MANUAL DEL PROGRAMADOR

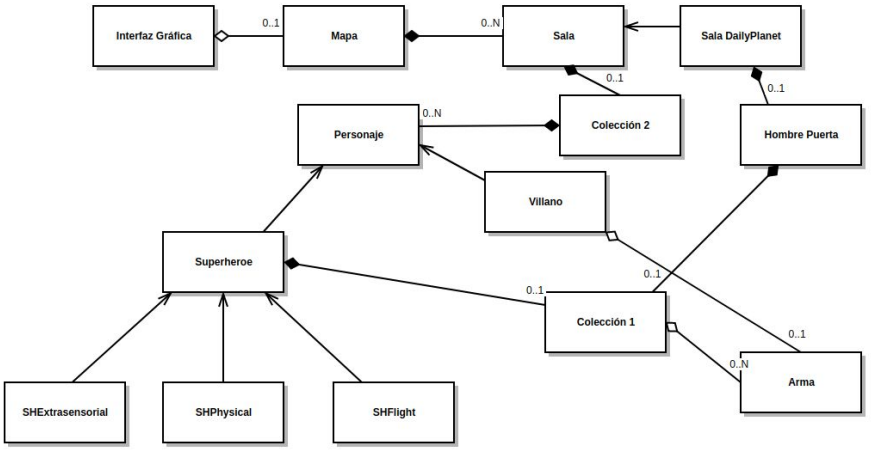
## 1.1. Introducción

El proyecto que hemos realizado se trata de un videojuego sobre diferentes superhéroes y villanos de marvel y dc que son capaces de orientarse sin control de usuario de manera que pueden llevar una ruta concreta a través de un laberinto aleatoriamente generado. Los heroes y villanos además tienen diferentes formas de interactuar entre ellos.

Todo esto esta expresado mediante una interfaz gráfica que permite repeticiones y carga de distintas configuraciones iniciales y que permite realizar otras operaciones en paralelo a la simulación que se encuentre en ejecución.

## 1.2. Análisis

### 1.2.1. Diagrama del modelo conceptual.



### 1.2.2. Identificación de colaboración y responsabilidad (CRC's)

| Clase | Responsabilidades | Colaboradores |
|---|---|---|
| **Mapa** | Almacenar toda la información y sobre todo el mapa,tanto las paredes y adyacencias como las propias salas y dentro sus personajes y su lógica. | Sala |
| **Personaje** | Almacenar toda la info de los personajes y administrar sus interacciones y movimiento, recoger las armas esta administrado en las subclases al tener otros comportamientos. | Sala<br>Mapa |
| **Sala** | Almacenar todos los personajes y armas en ella | Arma<br>Sala |
| **Arma** | Gestiona el poder y el nombre de las armas | Ninguno |
| **Interfaz gráfica** | Muestra toda la información del proyecto y se encarga de tanto el fichero init como el fichero log. | Mapa |
| **Hombre Puerta** | Almacena todas las armas y si el portal se encuentra abierto o cerrado. | Arma |
| **Superheroe** | Tipo de personaje que posee un comportamiento distinto en lo respectivo a interacciones y armas, que las almacena en lista. | Personaje<br>Arma |
| **Villano** | Tipo de personaje que posee un comportamiento distinto en lo respectivo a interacciones y armas, que sólo contiene una. | Personaje<br>Arma |

## 1.3. Diseño

### 1.3.1. Diagrama de clases del sistema

**Adjuntado fuera del documento porque es muy extenso**

### 1.3.2. Contrato de operaciones a nivel de diseño

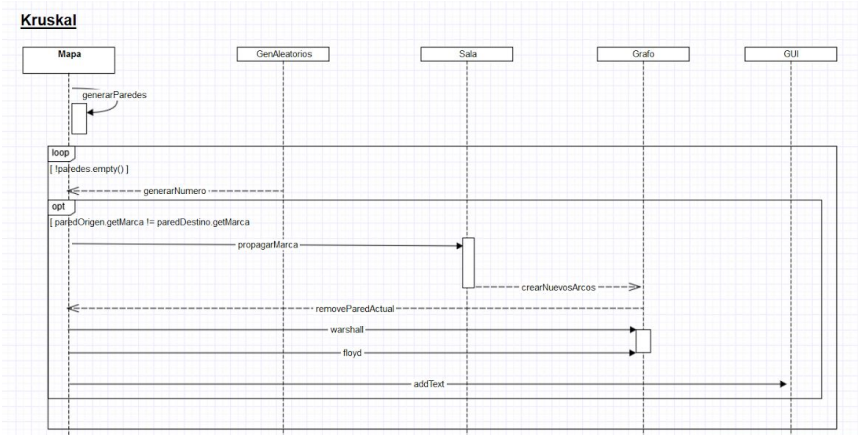
**Realizadas en el código fuente del proyecto.**

### 1.3.3. Estructuras de Datos utilizadas

Las estructuras de dato que principalmente hemos empleado son 4:

- **Linked list:** hemos empleado esta estructura en los personajes porque es la más eficiente a la hora de insertar y eliminar al final y a la hora de hacer recorridos.
- **ColaOrdenada:** se trata de una implementación propia de una lista ordenada en la que usamos una arraylist y un comparador secundario para realizar las operaciones de una cola en la que se ordenan los valores y solo se puede obtener y eliminar el máximo.
- **Árbol Binario:** hemos empleado un árbol binario de búsqueda tanto en los personajes como en el hombre puerta porque las consultas en dicha estructura se realizan con una complejidad algorítmica de log2() y por lo tanto se consigue una reducción significativa en los tiempos de las mismas.

### 1.3.4. Diagramas de secuencia (opcional)




## 1.4. Implementación

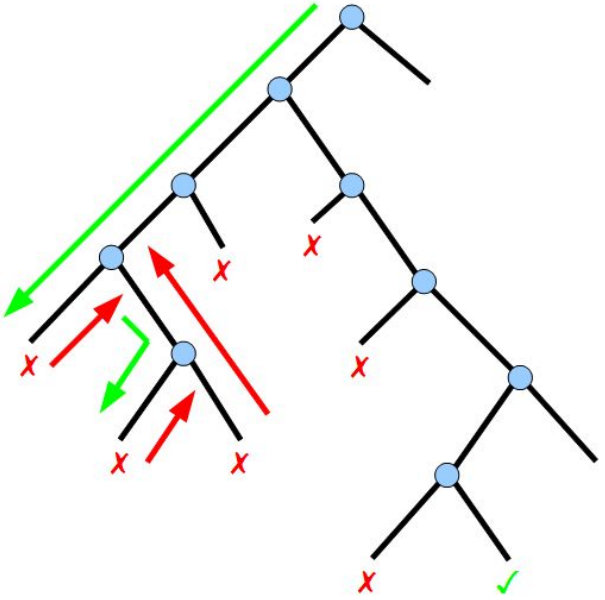
### 1.4.1. Algoritmos de especial interés

- **Kruskal**: es un algoritmo de generación de laberintos que se emplea para generar laberintos aleatorios. La idea es adjudicar números a las salas e ir tirando paredes (y actualizar las adyacencias del grafo) a la vez haciendo que los dos pasillos compartan la misma marca de manera que al final todas las salas cuando se encuentren conectadas acabaran con la misma marca.

| | | | | | |
|---|---|---|---|---|---|
| 0 | 1 | 2 | 3 | 10 | 5 |
| 6 | 7 | 13 | 9 | 10 | 11 |
| 12 | 13 | 13 | 15 | 16 | 17 |
| 18 | 20 | 20 | 21 | 22 | 23 |
| 24 | 31 | 26 | 27 | 28 | 29 |
| 30 | 31 | 32 | 33 | 35 | 35 |

| | | | | | |
|---|---|---|---|---|---|
| 33 | 33 | 33 | 33 | 33 | 33 |
| 33 | 33 | 33 | 33 | 33 | 33 |
| 33 | 33 | 33 | 33 | 33 | 33 |
| 33 | 33 | 33 | 33 | 33 | 33 |
| 33 | 33 | 33 | 33 | 33 | 33 |
| 33 | 33 | 33 | 33 | 33 | 33 |

- **Recorrido en profundidad de los caminos del grafo con backtracking:** esto consiste en ir recorriendo todas las salas adyacentes a cada sala con un algoritmo recursivo manteniendo un registro de todas las salas visitadas anteriormente y deteniéndose cuando se encuentre con un callejón sin salida o entre en un bucle, esto hace que pueda recorrer uno a uno los caminos de un grafo y realizar operaciones sobre cada nodo en ellos, en nuestro caso incrementar la frecuencia.



### 1.4.2. Definición de entradas/salidas

**Entradas:**

- La ruta de un documento de texto con los datos de la simulación que se quiere realizar, siguiendo el formato estipulado.
- Las modificaciones sobre dicho texto que se quieran realizar sobre el área de texto, que también afectarán a la simulación

**Salidas:**

- Salida por pantalla mediante una interfaz gráfica implementada mediante swing de java una simulacion turno a turno usando los datos preestablecidos en la entrada.
- Opción de guardar el texto que contiene todos los resultados de la simulación en un fichero que emplea la extensión log

### 1.4.3. Variables o instancias más significativas y su uso

- **Mapa**: Instancia universal del mapa que es por donde los personajes tienen permitido moverse y donde se almacenan casi todos los datos de la simulación durante la misma.
- **Personaje**: Superclase de todos los personajes que se pueden mover en la simulación
- **DPInitGUI**: Clase que se encarga de la gestión gráfica del proyecto y de gestiona todas las acciones que en él se pueden realizar.

## 1.5. Lista de errores que el programa controla

Fallo a la hora de introducir los datos: no se realizaria la simulación y debería volver a ingresar un fichero de texto.

## 1.6. Pruebas

Para realizar las pruebas hemos utilizado la librería JUnit 4 vista en los laboratorios de clase. Siguiendo con el procedimiento de tal libreria hemos creado una clase TestSuite llamada AllTests la cual ejecuta todos los tests.
Y hemos decidido realizar pruebas de las siguientes clases y métodos:

- Arbol. Ya que ha sido una clase creada a partir de la cual los profesores nos disteis y posteriormente completada y modificada por nosotros hemos decidido probar que funciona correctamente, para ello hemos probado los métodos más importantes tales como:
    - BuscarTest. Para comprobar que funciona correctamente buscamos un nodo que no es hoja, una hoja y un valor no existente en el arbol.
    - PerteneceTest. Comprobamos si este método funciona correctamente tanto para datos existentes como para no existentes.
    - BorrarTest. Como el caso más importante es a la hora de borrar un nodo no hoja y restaurar el orden, borramos un nodo no hoja y comprobamos que su subárbol correspondiente queda bien estructurado.
    - toStringTest. Comprobamos que el árbol creado para las pruebas devuelve su string correspondiente.
    - CantidadElementosTest. Comprobamos que el método devuelve correctamente la cantidad de elementos del árbol creado para las pruebas.
    - ProfundidadTest. Según el árbol creado para las pruebas comprobamos que devuelve la profundidad de este correctamente.

- GenAleatorios. En esta prueba simplemente comprobamos que al generar un número aleatorio con un determinado límite dicho número no se salga del límite ni sea negativo.

- Grafo. En esta prueba solo comprobamos que se crean los nodos y arco correctamente junto con su toString.
- Mapa. Ya que es la clase más importante con los algoritmos más importantes hemos decidido probar estos:
  - InicializarMapaTest. Para comprobar que se inicializa correctamente el mapa inicializamos un mapa de pruebas con determinados parámetros y posteriormente comprobamos que los atributos de esta instancia de mapa han sido inicializados con los mismos que los de los parámetros iniciales.
  - KruskalTest. Para comprobar que kruskal se realiza correctamente comprobamos que se ha extendido la misma marca para todas las salas.
  - CheckPortalInicioTest. Comprobamos que inicialmente el portal del Teseracto se inicializa a false(cerrado).
- Personaje. Ya que es la clase padre de todos los personajes -excepto el hombre puerta- decidimos realizar las siguientes pruebas:
  - interactuarTest. En esta prueba comprobamos que después de interactuar el personaje que pierde pierde el arma y que no interaccionan superheroes con otros superheroes.
  - buscarArmaTest. En esta prueba buscamos un arma que si pertenece a un personaje y luego otra que no para controlar los casos extremos.
  - eliminarArmaTest. En esta prueba comprobamos que se elimina un arma existente correctamente y que no realiza ninguna acción al eliminar una no perteneciente a ese personaje.
  - toStringTest. Comprobamos que cada tipo de personaje genera su string correspondiente correctamente
- Sala. De esta clase decidimos realizar pruebas de sus funcionalidades principales pero no de las que eran tan complejas que no podamos realizar pruebas. De esta clase realizamos las siguientes pruebas:
  - insertarArmaTest. Comprobamos que se insertan correctamente las armas en las salas.
  - armaPoderosaTest. Insertamos varias armas en una sala y comprobamos que nos devuelve la arma de mayor poder según el criterio establecido.
  - destruirArmaPoderosaTest. En esta prueba insertamos varias armas y destruimos una comprobando que las armas restantes no han sido afectadas y siguen en orden..
  - noQuedanArmasTest. Tras borrar todas las armas de una sala comprobamos que este metodo funciona correctamente.

## 1.7. Historial de desarrollo

Al principio de todo nos encontramos con el problema de cómo realizar eficientemente el proyecto conjuntamente y los profesores de esta asignatura nos dieron la solución con un controlador de versiones por lo que decidimos utilizar git y crear un repositorio en común, mucho más eficiente y cómodo a la hora de trabajar en grupo.

En la primera entrega nos encontramos el problema de cómo resolver correctamente la funcionalidad de simulación pero conseguimos solucionarlo.

La segunda entrega pese a tenerla a punto decidimos no realizarla ya que por falta de tiempo decidimos estudiar para un parcial que teníamos.
Y por último la tercera entrega pese a tener problemas con las frecuencia de las salas y alguna ruta conseguimos resolverla correctamente.

# MANUAL DE USUARIO

## 2.1. Introducción

Este programa te permite una vez ejecutado iniciar o cargar un mapa y unos personajes personalizados. Estos personaje introducidos tienen como objetivo recoger distintas armas y seguir distintos caminos para abrir el portal del hombre puerta y así adueñarse de la ciudad.

## 2.2. Guía de instalación del proyecto

La instalación de la aplicación para simplemente ejecutarla se puede hacer mediante el archivo jar adjunto mientras que si lo que se quiere es obtener el proyecto completo, el simple hecho de abrir la ruta del mismo dará resultado si se tiene java virtual machine instalado.

No requiere ninguna dependencia de un paquete que no se encuentre ya incluido en java. Testeado para la versión 8 de java.

## 2.3. Interfaz de usuario

El usuario puede:

- Cargar un fichero de configuración y modificarlo o escribirlo desde cero en el área de texto de la primera, para cargar el fichero se puede usar la opción del menú o la hotkey ctrl+O. En la ventana de explorador se debe buscar el fichero de texto a elegir.
- Generar indefinidamente mapas de manera aleatoria pulsando el botón de generar que se encuentra en la segunda pestaña y tomara los datos del texto de la primera.
- En caso de haber generado el boton de simulación estará activado que permitirá ejecutar la simulación paso a paso con tiempo de espera entre turnos y pausarla volviendo a pulsar el mismo botón. Durante la ejecución, podrá moverse y realizar el resto de acciones en la interfaz.
- Si quisiera saltar hasta los resultados finales, el tercer botón avanza a velocidad rápida la simulación hasta que la misma termina.
- Todos los resultados según se va realizando la simulación son almacenados en la tercera pestaña y mediante la opción del menú o la hotkey ctrl+S se pueden guardar los resultados en un archivo .log

## 2.4. Ejemplo de funcionamiento

Abrir el proyecto, cargar cada una de las configuraciones iniciales que son proporcionadas por defecto pudiendo modificarla, generar el laberinto pulsando el botón, simular y observar los resultados y posteriormente guardar el fichero de log.

### 2.5. Errores

No han sido detectados errores notables excepto que algunas acciones no están disponibles una vez se ha pausado la simulación, si diera este error bastaría con reanudar el juego para poder realizar dichas acciones.

## VALORACIÓN FINAL DEL PROYECTO

Hemos considerado de este un proyecto que ha sido tanto entretenido (no como el de edi) de realizar como desafiante y que nos ha permitido alcanzar una comprensión mucho más profunda de cómo realizar un buen código y nuestra labor como programadores.

A través de este proyecto hemos logrado aprender y dominar un nuevo lenguaje de programación, java, de manera que hemos ahondado más nuestro conocimiento sobre cómo pensar y desarrollar un software funcional empleando la menor cantidad de tiempo y recursos posibles.

Para optimizar este proceso de desarrollo hemos conocido métodos de generalización, herencia y polimorfismo así como patrones de diseño, que nos permiten no tener que repetir el mismo código varias veces y anticiparnos a cambios futuros dentro de un proyecto.

Hemos descubierto el uso de repositorios y documentación de manera que se potencie el trabajo en equipo y la comunicación con el usuario/cliente.