DESARROLLO DE PROGRAMAS

Curso 2014/15

Grado en Ingeniería Informática en
Ingeniería del Software
Ingeniería de Computadores
Pablo Macías y Fernando González

DOCUMENTACIÓN EXTERNA DEL PROYECTO

INDICE

MANUAL DE USUARIO

1.1. Introducción

Star Wars, ¿a quién no le suena este nombre? Star Wars se ha convertido en una de las sagas de películas más conocida desde el 1977, fecha de estreno de su primera película, hasta la actualidad. Desde aquella fecha son 7 las películas que se han producido y todas ellas, han conseguido romper records en taquillas del cine. Además, la empresa Lucas Film propietaria y productora de esta saga, ha comercializado y permitido a otras productoras el desarrollo de series, comics, merchandising, juegos para videoconsolas, novelas, ….
Después de analizar todos sus productos y videojuegos, a esta gran empresa dirigida por George Lucas le faltaba un videojuego; un videojuego en el que los personajes de Star Wars pudieran interactuar en la galaxia con las estaciones y los midiclorianos. Debido a esta gran carencia, decidimos crear un videojuego con las características anteriormente nombradas. Por ello decidimos crear *"Star Wars: Episode DP".*

1.2. Guía de instalación del proyecto

Este proyecto se encapsula en una carpeta en donde se encuentran diversas carpetas agrupadas por generación de archivos para su compilación, pruebas, imágenes, archivos de código fuente y archivos de configuración.

1.3. Interfaz de usuario

La interfaz de usuario está conformada por una ventana en donde encontramos una barra de menús y tres pestañas para movernos entre ellas. A continuación, se explica la funcionalidad de cada una de ellas:

- Barra de menús: En ella nos encontramos los menús de Archivo y Generar.
    - Archivo: Menú donde podemos interactuar desde el exterior al programa, desde el programa al exterior, y salir del programa.
        - Abrir Inicio: Se encarga de abrir una ventana emergente para cargar un archivo .txt para la configuración inicial del proyecto. La carpeta por defecto es "./files". Atajo del teclado Ctrl+A.
        - Guardar Log: Se encarga de guardar el log del programa en un archivo configurado por el usuario a través de una ventana auxiliar en donde por defecto se encuentra en la carpeta "./files" y extensión .log. Atajo del teclado Ctrl+S.
        - Salir: Provoca la interrupción y salida del programa. Atajo del teclado Alt+F4.
    - Generar: Menú donde podemos ajustar el proyecto según la configuración en la pestaña Fichero Inicio.
        - Genera una galaxia acorde con el texto escrito en el cuadro de texto de la pestaña Fichero Inicio.

1.4. Ejemplo de funcionamiento

Ayuda a comprender cómo funciona el proyecto

1.5. Errores

Explicación sencilla de los errores que puede encontrar el usuario y qué puede hacer para solucionarlos

MANUAL DEL PROGRAMADOR

1.1. Introducción

El proyecto consiste en el desarrollo de un juego automático (movimientos e interacción de personajes los desarrolla el ordenador) cuya configuración inicial se permite cargar a través de un fichero de inicio con información sobre el tamaño de la Galaxia, colocación de la puerta e información de los personajes.

Una vez cargado el fichero de inicio se genera una galaxia de acuerdo a esa configuración inicial.

El desarrollo de la partida se regula a través de turnos en donde los personajes se van moviendo por la galaxia interactuando con los midiclorianos que hay en cada una de las estaciones. La finalidad del juego consiste en la apertura de la puerta de salida. Acorde con las películas en este juego existirán personajes del Lado de la Luz que intentarán escapar, y personajes del Lado Oscuro que intentarán impedir que ningún personaje se escape.

1.2. Análisis

1.2.1. Identificación de clases potenciales (operaciones y responsabilidades).

Las clases las podemos organizar según su funcionamiento y finalidad en 4 grandes grupos (se corresponden con los paquetes en donde se ubican):

- Personajes: Agrupa las clases que conforman la jerarquía de personajes siguiendo conceptos de Java como la herencia y polimorfismo. Los personajes ejecutan operación relacionadas con el movimiento y la interacción con midiclorianos. Las clases de personajes según su jerarquía son:
    - Personaje: Es una clase común para todos los personajes de cualquier tipo que pudiera haber en la Galaxia. En esta clase encontramos operaciones relacionadas con los atributos de esta clase (*getters/setters*), operaciones de interacción con la estación en la que se encuentra (*acción, accionEstacion y acciónPuerta),* operaciones de interacción con midiclorianos (*recogerMidicloriano*), operaciones para generar el camino del personaje por la galaxia (*generarCamino, setRuta*), operaciones de movimiento por estaciones (*mover, moverA*) y operaciones relacionadas con la salida de información del personaje (*toString, toLog, toLogini, rutaToString, midiclorianosToString*). Esta clase se encuentra en *"./src/personajes/Personaje.Java".* De esta clase derivan dos clases llamadas LightSide e Imperial.
        - LightSide: Es una clase común para todos los personajes del Lado de la Luz. Encontramos operaciones propias de esta clase de personaje como las heredadas de la clase principal Personaje. En ella encontramos operaciones que se implementan de la clase principal relacionadas con la acción de los personajes en Estaciones y Puertas (*accionEstación, accionPuerta*) y con la

generación de caminos (*estacionVisitada, generarCamino, generarCaminoBT, movimientoPosible*). De esta clase derivan tres clases que agrupan a los tipos de personajes que pertenecen a este Lado de la Luz que son Contrabandista, FamiliaReal y Jedi.

- Contrabandista: Es una clase que hereda de LightSide. En esta clase no existen operación de funcionamiento ya que al ser comunes a los personajes del Lado de la Luz se han implementado en la clase superior. En esta clase se encuentra una operación sin importancia en el desarrollo del funcionamiento que es *getTipo* usada para la salida de la información del personaje.
- FamiliaReal. Es una clase que hereda de LighSide. En ella nos encontramos dos operaciones relacionadas con la generación de la ruta ya que estos personajes se comportan de manera diferente a los del Lado de la Luz. Estas operaciones son *generarCamino* y *generarCaminoBT*. Además encontramos la operación *getTipo* usada en la salida de información del personaje.
- Jedi: Clase parecida en su contenido a Contrabandista ya que no integra operación que supongan una gran importancia en el normal funcionamiento del proyecto. En esta, como en las anteriores, se encuentra la operación *getTipo* para la salida de información del personaje.

- Imperial: Es una clase común para todos los personajes del lado oscuro e integran operaciones relacionadas con su interacción con estaciones(*accionEstacion* y *accionPuerta*) y midiclorianos (*setMidiclorianos*), y operaciones para generar su ruta por la Galaxia(*generarCamino y generarCaminoBT*). Además encontramos las operaciones heredadas de la clase principal Personaje.

- Estructura: Agrupa las clases relacionadas con la Galaxia, las estaciones que la conforman y los midiclorianos, presentes tanto en estaciones como en personajes. Las estaciones están organizadas en una jerarquía según sean estaciones normales o estaciones con puerta. Las clases que conforman esta estructura son:
  - Galaxia: Clase en donde encontramos operaciones muy importantes en el funcionamiento del proyecto ya que es la galaxia quien forma y construye el lugar donde después irán las estaciones, que a su vez agrupan a los personajes. Como en toda clase donde necesitamos manejar información de sus atributos y debido al principio de encapsulamiento, encontramos las operaciones de manejo de información (*getters/setters*). También encontramos operaciones afines con la construcción de la Galaxia y su laberinto de estaciones (*construirGalaxia, construirParedesIni, expandirConexion, generarLaberinto*), con el patrón Singleton utilizado para esta clase (*obtenerInstancia*), con la generación de combinaciones y el reparto de midiclorianos (*generarMidiclorianosCerradura, generarMidiclorianosGalaxia, repartirMidiclorianos*), con la simulación

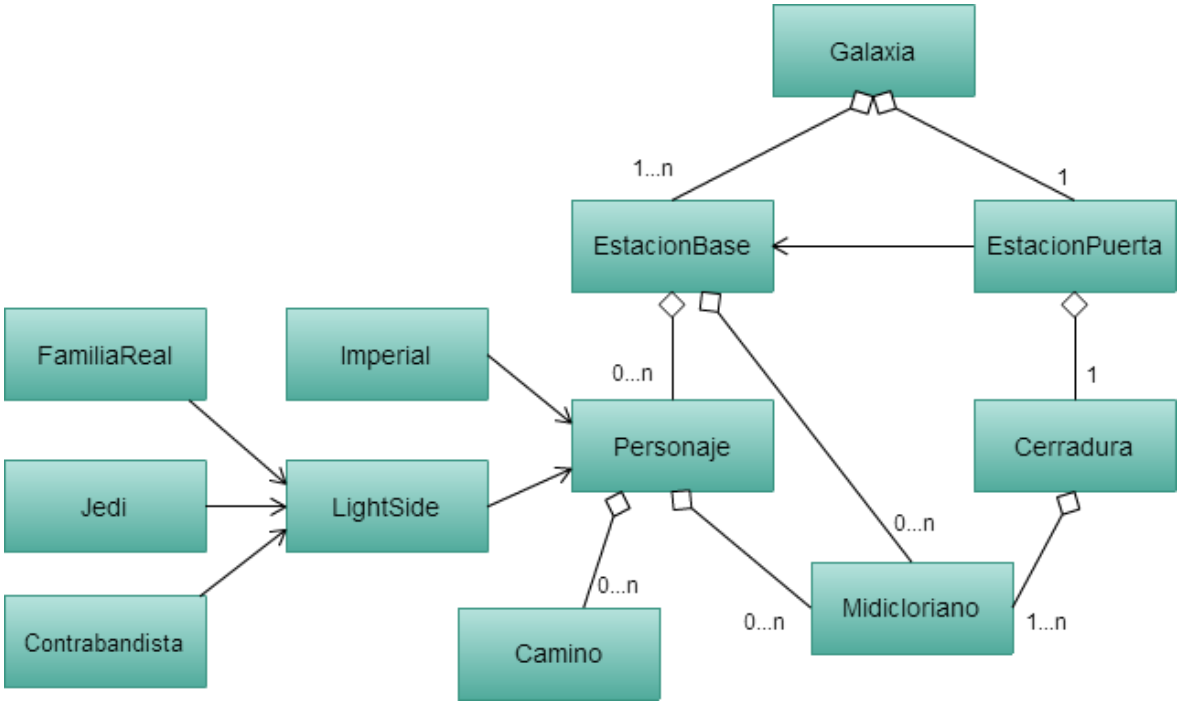
del juego (*simular*, *accion*) y con la salida de información de la Galaxia (*imprimirGalaxia, toLog*). Dentro de esta clase encuentra la clase Pared:
        - Pared: Clase básica utilizada para la creación del laberinto. No contiene ninguna operación aparte de su constructor.
    - EstacionBase: Clase que agrupa información y operaciones de una estación genérica. En ella encontramos operaciones de manejo de atributos (*getters*), operaciones usadas en la simulación (*accion*), operaciones para manejar personajes y midiclorianos que se encuentran en ella (*insertarMidicloriano, insertarPersonaje, sacarMidicloriano, sacarPersonaje*) y operaciones para salida de información (*imprimirPersonajesMarca, midiclorianosToString, toLog, toLogPersonajes*).
        - EstacionPuerta: Clase que hereda de EstacionBase y conforma la estación en donde se aloja la puerta de la Galaxia. En esta clase encontramos operaciones para establecer una cerradura a la puerta(*setCerradura*) y para finalizar la estación (*fin*). Dentro de ella encontramos una cerradura:
            - Cerradura: Clase que compone la cerradura de la puerta de la Galaxia. Esta clase está compuesta por operaciones para establecer atributos (*setters*), operaciones privadas que trabajan con arboles (*mostrarArbol, nodosExternos, nodosInternos, profundidad, vaciarArbol*), operaciones para consultar o modificar parámetros de la cerradura (*combine, Abierta, generarCombinacion, probarMidicloriano, reiniciar, setCombinacionInicial, setEstado*) y una operación para la salida de información (*toLog*).
    - Midicloriano: Clase que conforma la estructura más "pequeña" y usada en el proyecto. Está formada por la operación para interacción con el único atributo que tiene (*getID*), operación de comparación (*compareTo*) y operación para la salida de la información (*toString*).
- Etc: Agrupa a las clases que intervienen en el funcionamiento del proyecto, pero no pertenecen a una categoría específica. Está compuesto por:
    - Camino: Clase usada para el movimiento de los personajes. Es una clase enumerada que contiene operaciones para conocer atributos (*getters*) y de salida de información (*toString*).
    - GenAletarios: Clase utilizada para generar números aleatorios al aplicar el algoritmo de Kruskal para la generación del laberinto. Contiene dos operaciones estáticas para poder llamarlas sin necesidad de inicializar una nueva instancia. Estas operaciones son *generarNumero* para generar un número aleatorio y *getNumGenerados* para saber cuántos números han sido generados en total desde el inicio de la ejecución del programa.
    - Reader: es una clase encargada de almacenar la información contenida en el fichero de inicio. Contiene una operación para leer la información del fichero (*leerFichero*), y operaciones para devolver los datos de cada una de las clases que se encuentran en el fichero de inicio (*getContrabandistaas*, *getDatosGalaxia*, *getDatosPersonajes*, *getImperiales*, *getJedis*, *getReales*).

    - Loader: Clase que se encarga de inicializar una galaxia completo con los datos leídos a través de la clase Reader. En ella encontramos operaciones para inicializar los datos de la galaxia y personajes (*loadDataCharacters, loadDataGalaxy*).
    - Logger: Clase encargada de la salida de información hacia ficheros. Se utiliza por todas las clases para mostrar información de sus respectivos parámetros. Está compuesta clases que inicializan y cierran el log (*close, init*) y clases que trabajan sobre la información en el log (*escribeLog, forzarVolcado, getBuffer*).
- Edd: Este paquete agrupa a las clases utilizadas como estructuras de datos complejas en el proyecto. Estas son:
    - Arbol: Esta clase implementa un árbol binario de búsqueda genérico utilizado para almacenar los midiclorianos de forma ordenado. Contiene operaciones para devolver información de sus parámetros (*getters*), de inserción y borrado (insertar, borrar, borrarOrden) y de consulta (*inOrden, esHoja, pertenece, vacio*).
    - Grado: Esta clase implementa un grafo de enteros utilizado en la galaxia para representar el laberinto de estaciones. En ella encontramos operaciones propias de un grafo como las de inserción y borrado de nodos y arcos (*nuevoNodo, nuevoArco, borrarNodo, borrarArco*), de generación de matrices auxiliares (*floyd, warshall*), de consulta (*adyacente, adyacentes, esVacio, siguiente*) y de salida de información (mostrarArcos, mostrarNodos, mostrarFloydC, mostrarPW).
- GUI: Paquete que engloba a las clases utilizadas por la interfaz gráfica en la que se muestran de manera interactiva e intuitiva el funcionamiento del proyecto. Todas las clases están fromadas por elementos Swing del framework de java.
    - Estacion: Clase que representa una estación en la GUI. Está formada por tres operaciones encargadas de modificar su aspecto (*pintarParedes, setTexto, tirarPared*).
    - Tablero: Clase que representa el tablero, es decir, el conjunto de estaciones con sus respectivas conexiones y paredes. Está formada por operaciones encargadas de modificar su aspecto utilizando a clases de estaciones específicas. Estas operaciones son *insertarNumPersonajesEstacion, insertarPersonajeEstacion, pintarParedes* y *tirarPared.*
    - PanelTablero: Clase encargada de representar el tablero de la Galaxia y botones para su funcionamiento. En ella encontramos operaciones para inicializar cada una de sus partes (*iniciListaRobots, iniciarPanelInferior, iniciarTablero*), para insertar componentes a sí misma (addAccionesBotones, añadirComponentes, insertarPersonajes) y modificar su apariencia (tirarParedes).
    - PanelFichero: Clase que se encarga de formar las pestañas de carga y salida de información. En ella encontramos operaciones para interactuar con su área de texto (*addTexto, setTexto, getTexto, setEditable*) y para su interacción con eventos (addFocusListener).
    - GUI: Clase encargada del funcionamiento de la interfaz gráfica donde encontramos todos los controles para cargar, simular, visualizar y guardar la simulación de una partida. Se encuentra formada por

operaciones de inicialización (initMenus) y de acciones de los botones del menú (abrirInicio, guardarLog, generarGalaxia, salir).

- Main: Clase encargada de la ejecución del programa. Es la que se encarga de inicializar el log, arrancar la interfaz gráfica y cerrar el log.

1.2.2. Diagrama del modelo conceptual.



1.2.3. Descripción de clases conceptuales.

A continuación, se explican las clases mostradas en el Diagrama del modelo conceptual

Galaxia:

Atributos:

Galaxia instancia
int dimX
int dimY
int turno
int idEstacionPuerta
int idLibertyStation
EstacionPuerta starsagate
EstacionBase LibertyStation
EstacionBase[][] Estaciones
ArrayListy<Personaje> personajes
ArrayList<Pared> paredes
ArrayList<Integer> pasosPorEstaciones

Operaciones

Galaxia(int idEstacionPuerta, EstacionPuerta starsgate, int dimX, int dimY)

Efecto: Constructor parametrizado de la clase Galaxia - Galaxia inicializado con parámetros dados.

obtenerInstancia(int idEstacionPuerta, EstacionPuerta starsgate, int dimX, int dimY)

Efecto: Método sobrecargado que devuelve la única instancia de la Galaxia - Devuelve la instancia de Galaxia. Si no está inicializada se crea una con parámetros dados.

Galaxia obtenerInstancia()

Efecto: Método sobrecargado que devuelve la única instancia de la Galaxia - Devuelve la instancia de Galaxia.

int getDimX()

Efecto: Método que devuelve el numero de filas de la Galaxia - Devuelve número de filas de la Galaxia

int getDimY()

Efecto: Método que devuelve el número de columnas de la Galaxia - Devuelve el número de columnas de la galaxia.

Grafo getGrafo()

Efecto: Método que devuelve el grafo de conexiones - Devuelve el grafo.

int getTurno()

Efecto: Método que devuelve el turno de la Galaxia - turno de la galaxia.

int getIdEstacionPuerta()

Efecto: Método que devuelve el ID de la estacion con puerta - Devuelve ID de Starsgate.

EstacionBase getEstacionLiberty()

Efecto: Método que devuelve la estacion de Libertad - Devuelve la estacion LibertyStation.

EstacionPuerta getStarsgate()

Efecto: Método que devuelve la estacion Puerta – Starsgate - Devuelve el ID de Starsgate.

ArrayList<Personaje> getPersonajes()

Efecto: Método que devuelve la EDD de los Personajes - Devuelve el ArrayList de los Personajes.

setPasosPorEstaciones(ArrayList<Integer> pasosPorEstaciones)

Efecto: Método que introduce ED con ruta desde la estacion de inicio hasta estación de fin - pasosPorEstaciones toma el valor del parámetro de entrada.

setPersonajes(ArrayList<Object> personajes)

Efecto: Método que inserta un ArrayList de personajes en la Galaxia - parámetro se convierte a ArrayList de personajes y se inserta en personajes.

construirParedesIni()

Efecto: Método para construir las paredes iniciales - Conectar todas las estaciones con sus estaciones más próximas - Grafo con todos los arcos que unen las estaciones con sus estaciones vecinas (N,S,E,O) y todas las paredes almacenadas en ED paredes, la cual ayudará a implementar algoritmo de Kruskal.

expandirConexion(int[][] nodos, int valorinicial, int valorfinal)

Efecto: Expande un ID a todos los IDs con valor = varinicial - Todos los nodos de la matriz que sean iguales al parámetro valorinicial se convierten en valorfinal.

construirGalaxia()

Efecto: Método para construir una Galaxia inicial - Se crean todos los nodos (ID de estaciones) y se llama a construirParedesIni().

generarLaberinto()

Efecto: Método para generar un laberinto - Genera un laberinto a través del Algoritmo de Kruskal ayudándose de una matriz de enteros haciendo referencia a los IDS de las galaxias. Sobre esta matriz se realizan los cálculos del Algoritmo de Kruskal. Sobre el grafo de la galaxia se insertan arcos de las estaciones adyacentes según el Algoritmo.

ArrayList<Midicloriano> generarMidiclorianosCerradura()

Efecto: Método que genera ArrayList de Midiclorianos iniciales de la galaxia - Genera arraylist con midiclorianos con ID desde el 0 hasta el 30.

ArrayList<Midicloriano> generarMidiclorianosGalaxia()

Efecto: Método que genera ArrayList de Midiclorianos iniciales de la galaxia - Genera arraylist con midiclorianos con ID desde el 0 hasta el 30.

repartirMidiclorianos(ArrayList<Midicloriano> midiclorianos)

Efecto: Método para repartir midiclorianos por la galaxia - Reparte los midiclorianos de 5 en 5 por las estacione almacenadas en pasosPorEstacion (Recorrido de FamiliaReal)

accion(int turno)

Efecto: Método que controla el movimiento de los personajes con el turno global - Recorre todas las estaciones de la galaxia invocando al método acción de cada una de ellas.

int[] IDtoCoordenadas(int ID)

Efecto: Método que convierte un ID de la estación en coordenadas cartesianas - Devuelve int[] de 2 posiciones donde: - int[0] = fila (eje x) - int[1] = columnas (eje y)

int coordenadastoID(int fila, int columna)

Efecto: Método que convierte unas coordenadas cartesianas en ID - Devuelve el ID correspondiente a las coordenadas según tamaño de la Galaxia

EstacionBase getEstacion(int ID)

Efecto: Devuelve la estación situada en el ID por parámetros - Calcula las coordenadas según ID por parámetros y las devuelve en un vector de 2 posiciones. Posicion 0: Coordenadas X Posicion 1 : Coordenadas Y

EstacionBase getEstacion(int fila, int columna)

Efecto: Método que devuelve la estación situado en las coordenadas dadas - Devuelve la estacion localizada en la determinada fila y columna.

simular()

Efecto: Método que simula un turno en la galaxia - Acciona la galaxia en un determinado turno y lo muestra en el log.

String imprimirGalaxia()

Efecto: Devuelve información sobre todas las estaciones de la galaxia y personajes que residen en cada una de las estaciones en forma de MINI MAPA DEL GRAFO.

toLog(int turno)

Efecto: Se escribe en el log información sobre el turno, la estación de la puerta, llamada al toLog de cerradura de la puerta, información sobre el mapa de la galaxia con imprimirGalaxia y llamada al log de todas las estaciones que tengan midiclorianos.

EstacionBase:

Atributos:

int ID

PriorityQueue<Personaje> personajes

ArrayList<Midicloriano> midiclorianos

Operaciones:

EstacionBase(int ID)

Efecto: Constructor parametrizado de la clase EstacionBase. Inicializa la estación con ID por parámetros y personajes con una nueva PriorityQueue - Instancia de EstacionBase creada con ID pasado por parámetros y personajes como una nueva ArrayList por defecto.

int getID()

Efecto: Método que devuelve el ID de la Estación - Devuelve el ID de la instancia de EstaciónBase

Object[] getPersonajes()

Efecto: Método que devuelve la EDD de los personajes - Devuelve un vector de Object con los personajes de la estación.

ArrayList<Midicloriano> getMidiclorianos()

Efecto: Método que devuelve los midiclorianos de la Estacion - EDD con midiclorianos de la estación.

accion(int turno)

Efecto: Método accion implementa la accion llamando al método acción de cada personaje existente en su estructura de almacenamiento de personajes - Ejecuta la acción del personaje o vuelve a meterlo en el final.

insertarMidicloriano(Midicloriano midicloriano)

Efecto: Método que inserta un midicloriano en la ED de la estación - Inserta un midicloriano (parametrizado) en la ED(ordenada).

Midicloriano sacarMidicloriano()

Efecto: Método que saca el primer midicloriano de la ED de la estación - ED de midiclorianos con un elemento menos.

insertarPersonaje(Personaje personaje)

Efecto: Método para insertar un personaje en la ED de la Estación - ED de la Base con un personaje más en la última posición.

Personaje sacarPersonaje()

Efecto: Método para sacar el primer personaje de la ED de la Estación - ED de la Estación con un personaje menos.

boolean esPuerta()

Efecto: Método que devuelve si una estacion es de tipo puerta o no - Devuelve valor booleano con información sobre si la pestacion es una puerta o no. Devuelve false.

String imprimirPersonajesMarca()

Efecto: Método similar al imprimirPersonajesMini pero con la caracteristica de que solo devuelve la marca de personaje o espacio en blanco - String con todos los personajes actuales en la ED.

String midiclorianosToString()

Efecto: Método para convertir midiclorianos de la estación en un cadena - Devuelve cadena con información de todos los midiclorianos que contiene en ella.

toLog()

Efecto: Método que escribe en el log información sobre la EstacionBase - Escribe en el log información sobre el ID y los midiclorianos que contiene la estación

toLogPersonajes()

Efecto: Método que escribe en el log información sobre los personajes que existen en la estación - Llama a toLog de todos los personajes que existen en esa estación.

EstacionPuerta

Atributos:

Cerradura cerradura

Operaciones

EstacionPuerta(int ID)

Efecto: Constructor parametrizado de la clase EstacionPuerta - Estacion inicializada con parámetros dados. Llamada a clase padre ESTACIONBASE.

setCerradura(Cerradura cerradura)

Efecto: Método para insertar una cerradura en la Estación - EstaciónPuerta con cerradura = cerradura parámetro.

fin()

Efecto: Método finaliza la estación - Saca a todos los personajes de las estación y se llama al método fin de cada uno de ellos.

esPuerta()

Efecto: Método que devuelve si una estacion es de tipo puerta o no - Devuelve valor booleano con información sobre si la pestacion es una puerta o no. Devuelve false.

Cerradura

Atributos:

ArrayList<Midicloriano> midiclorianosIniciales

ArrayList<Midicloriano> combinacionInicio

Arbol<Midicloriano> combinacionCerradura

Arbol<Midicloriano> midiclorianosProbados

boolean estado
int alturaDesbloqueo

Operaciones:

Cerradura(int alturaDesbloqueo)

Efecto: Constructor parametrizado de la clase Cerradura - Inicializa la cerradura con alturaDesbloqueo por parámetros, combinacionInicio con nueva ArrayList de Midiclorianos, combinacionMidiclorianos con nuevo Arbol de Midiclorianos y midiclorianosProbados con nuevo Arbol de Midiclorianos.

boolean Abierta()

Efecto: Método para comprobar estado de la cerradura - Devuelve TRUE(Cerradura abierta) o FALSE(Cerradura cerrada).

setEstado(boolean estado)

Efecto: Método para instroducir estado de la puerta - Estado de la cerradura con el valor del parámetro.

setCombinacionInicial(ArrayList<Midicloriano> midiclorianosIniciales)

Efecto: Método para introducir la combinación inicial de midiclorianos - midiclorianos iniciales con valor del parámetro.

String mostrarArbol(Arbol<Midicloriano> arbol)

Efecto: Método para mostrar el contenido de un arbol de midiclorianos a través de una cadena - Devuelve cadena con el contenido de los nodos en recorrido inOrden. Método recursivo.

int profundidad(Arbol<Midicloriano> arbol)

Efecto: Método para calcular la profundidad de un arbol de Midiclorianos - Calcula la pronfudidad de cada uno de los hijos del nodo y devuelve la mayor de las dos. Método recursivo

int nodosInternos(Arbol<Midicloriano> arbol)

Efecto: Método para calcular los nodos internos de una arbol de Midiclorianos - Calcula la cantidad de nodos internos sumando los nodos internos en el hijo izquierdo más los del derecho más, en el caso de que la raíz fuera nodo interno, uno. Método recursivo.

int nodosExternos(Arbol<Midicloriano> arbol)

Efecto: Método para calcular nodos externos de un arbol de Midiclorianos - Calcula nos nodos externos sumando los nodos externos del hijo más los del hijo derecho más, en el caso de que la raiz fuese nodo externo, uno.

vaciarArbol(Arbol<Midicloriano> arbol)

Efecto: Método para vaciar un arbol de Midiclorianos - Vacia el arbol asignado un nuevo arbol al parámetro y llamando al recolector de basura.

combine(ArrayList<Midicloriano> midiclorianos, ArrayList<Midicloriano> midiclorianosCombinados, int min, int max)

Efecto: Algoritmo recursivo que balancea los valores de un array y los almacena otro - midiclorianos de midiclorians insertados de forma balanceada combinedMidiclorians.

configurarCerradura(ArrayList<Midicloriano> midiclorianos)

Efecto: Método para configurar la cerradura con una combinación - Combinación cerradura con datos de midiclorianos (parámetro entrada)

generarCombinacion()

Efecto: Método para generar combinación de midiclorianos a partir de los midiclorianos iniciales guardados - CombinacionInicio con la combinación inicial de la cerradura y CombinacionCerradura con midiclorianos balanceados

boolean probarMidicloriano(Midicloriano midicloriano)

Efecto: Método para probar un midicloriano en la cerradura - Devuelve FALSE si el midicloriano ya ha sido probado en la cerradura, TRUE si no ha sido probado.

Excepción : Lanza InterruptedException.

reiniciar()

Efecto: Método para reiniciar la cerradura a su combinación inicial - Cerradura inicializada con configuración inicial y cerrada.

comprobarEstado()

Efecto: Método para comprobar estado de la puerta según condiciones de apertura - Actualiza el estado de la puerta según condiciones de apertura.

toLog()

Efecto: Método que escribe en el log información sobre la cerradura - Escribe en el log el estado de la puerta, los midiclorianos de la combinación de la cerradura y los midiclorianos probados.

Midicloriano:

Atributos:

int ID

Operaciones:

Midicloriano(int ID)

Efecto: Constructor paramétrizado del objeto Midicloriano - Midicloriano creado con ID del parámetro.

int getID()

Efecto: Método que devuelve el ID del midicloriano.

String toString()

Efecto: Método para devolver información del midicloriano.

int compareTo(Midicloriano o)

Efecto: Método para comparar un Midicloriano con la instancia actual – Devuelve entero indicando si es menor(número negativo), igual(0) o mayor(número positivo)

Personaje:

Atributos:

char marcaClase

String nombre

boolean ganador

ArrayList<Midicloriano> midiclorianos

LinkedList<Camino> ruta

int turno

EstacionBase estacionPosicion

Operaciones:

Personaje(char marcaClase, String nombre, int estacionPosicion, int turnoInicio)

Efecto: Constructor parametrizado de la clase Personaje - Instacia de Personaje inicializada con marcaClase y por parámetros y midiclorianos como nueva ArrayList. Inserta en la estacionPosicion al personaje.

int getTurno()

Efecto: Método que devuelve el turno del Personaje - Devuelve el turno de la instancia del Personaje.

String getNombre()

Efecto: Método que devuelve el nombre del Personaje - Devuelve el del atributo nombre.

EstacionBase getEstacionPosicion()

Efecto: Método que devuelve la estacion donde se encuentra el personaje - Devuelve la estacion donde actualmente está el personaje.

char getMarcaClase()

Efecto: Método que devuelve la marca de clase del personaje - Devuelve marca de clase del personaje.

boolean isGanador()

Efecto: Método que devuelve si el personaje es ganador - Devuelve el valor del atributo ganador.

LinkedList<Camino> getRuta()

Efecto: Método que devuelve la ruta del Personaje - Devuelve ruta del personaje.

setRuta(LinkedList<Camino> ruta)

Efecto: Método que inserta una ruta al Personaje - Ruta del Personaje = Ruta por parámetros.

setRuta(ArrayList<Integer> camino)

Efecto: Método que inserta una secuencia de Estaciones(ID) como transformación a una secuencia de Ordenes en formato de orientaciones - Se inserta en la ruta del personaje una serie de secuencia de orientaciones acorde con la secuencia de estaciones introducida por parámetro.

moverA(EstacionBase estacion)

Efecto: Método que pone en el Personaje una estacion actual (posición) - Estacion por parámetros: Nuevo Personaje(instacia actual del personaje)

EstacionBase getSiguienteEstacion()

Efecto: Método que devuelve la siguiente estación al personaje según ruta - Primera orientacion de la ruta es sacada y se vuelve a insertar en la ED de la ruta, y se ejecutan los métodos para calcular siguiente posicion con ayuda de dos variables (fila y columna).

recogerMidicloriano(Midicloriano midicloriano)

Efecto: Inserta en la ED el midicloriano por parámetros - ED de midiclorianos con un elemento más (última posición).

mover()

Efecto: Método que ejecuta la acción de mover de un personaje - El personaje se mueve a la siguiente estación.

accion()

Efecto: Método encargado de accionar-simular al personaje durante 1 turno - Si el personaje está en una estación, se ejecuta accionEstacion(). el personaje está en una puerta, se ejecuta accionPuerta(). En cualquiera de los casos se incrementa el turno en 1.

boolean esImperial()

Efecto: Método para saber si el perosonaje es Imperial - Devuelve falso.

fin()

Efecto: Método que indica que el personaje ha ganado - Establece ganador a true y mueve al personaje a la estación de los ganadores.

int compareTo(Personaje o)

Efecto: Método para comparar un Personaje con la instancia actual – Devuelve entero indicando si es menor (número negativo), igual(0) o mayor(número positivo).

String toString()

Efecto: Devuelve información sobre el personaje - Devuelve string con información sobre el personaje.

String rutaToString()

Efecto: Método que convierte la ruta del personaje en un String - Devuelve la ruta tal que "N S E O N S E O"

String midicloriansToString()

Efecto: Método que convierte los midiclorianos del personaje en String - Devuelve los midiclorianos tal que "1 2 3 4 5 6"

toLogini()

Efecto: Método que escribe en el Log información inicial sobre el personaje.

toLog()

Efecto: Método que escribe en el Log información sobre el personaje.

LightSide:

Atributos:

NINGUNO

Operaciones:

LightSide(char marcaClase, String nombre, int estacionPosicion, int turnoInicio)

Efecto: Constructor parametrizado de la clase Personaje - Instacia de LightSide inicializada con marcaClase y estacionPosicion por parámetros y midiclorianos como nueva ArrayList. Inserta en la estacionPosicion al personaje. Inicializa rutaInicial como nueva LinkedList.

boolean estacionvisitada(ArrayList<Integer> solucion, int estacionActual, int estacionComprobar)

Efecto: Método que comprueba si una estacion ha sido vistada ya - Devuelve si la estacionComprobar se encuentra en solucion (hasta estacionActual)

accionPuerta()

Efecto: Método que acciona el comportamiento de un personaje en una puerta - Saca un midicloriano del LightSide con scarMidicloriano(), guardala estacionPosicion como una EstacionPuerta, y si el midicloriano sacada no es null, lo prueba en la cerradura la estacion con puerta.

accionEstacion()

Efecto: Método que acciona el comportamiento de un personaje en una puerta - Guarda la estacionPosicion, saca un midicloriano de la estacion con sacarMidicloriano(), y si el midicloriano sacado es distinto de null, se almacena en la ED de midiclorianos del LightSide con recogerMidicloriano().

boolean movimientoPosible(Grafo grafo, int estacion, Camino orientacion, int[] siguienteEstacion)

Efecto: Método que comprueba si estando en una determinada estación se puede hacer un moviemiento a una orientación - Se obtienen las coordenadas de origen y destino (con ayuda de la introducida por parámetros), se obtiene el ID destino y se devuelve en siguienteEstacion[0]

void generarCaminoBT(Grafo grafo, ArrayList<Integer> solucion, int estacionOrigen, int estacionDestino, Camino orientacion)

Efecto: Método para generar un camino de un personaje utilizando el algoritmo de la mano derecha - Si la estacionOrigen es distinta a la estacionDestino se comprueban los 4 caminos posibles dependiendo de la orientación (siempre gira a la derecha cuando hay posibilidad). Método recursivo.

generarCamino()

Efecto: Método que genera el camino del LightSide - La ruta del personaje es igual a la ruta calculada con generar caminoBT. La primera llamada al módulo se realiza con la orientación sur suponiendo que el personaje está mirando hacia abajo. Se inserta la estacionPosicion donde se encuentra, antes de la llamada recursiva; y despues de la llamada se elimina ya para borrar la estación origen.

String getTipo()

Efecto: Método que devuelve el tipo del personaje - Devuelve "lightside".

Jedi

Atributos:

NINGUNO

Operaciones

Jedi(char marcaClase, String nombre, int estacionPosicion, int turnoInicio)

Efecto: Constructor parametrizado de la clase Personaje - Instacia de Jedi inicializada con marcaClase y estacionPosicion por parámetros y midiclorianos como nueva ArrayList. Inserta en la estacionPosicion al personaje.

String getTipo()

Efecto: Método que devuelve el tipo del personaje - Devuelve "jedi".

FamiliarReal

Atributos:

NINGUNO

Operaciones:

FamiliaReal(char marcaClase, String nombre, int estacionPosicion, int turnoInicio)

Efecto: Constructor parametrizado de la clase Personaje - Instacia de Familia Real inicializada con marcaClase y estacionPosicion por parámetros y midiclorianos como nueva ArrayList. Inserta en la estacionPosicion al personaje.

boolean generarCaminoBT(Grafo grafo, ArrayList<Integer> solucion, int estacion, int estacionOrigen, int estacionDestino, int tam)

Efecto: Método para generar un camino utilizando un recorrido en profundidad - Devuelve false si la estacion(etapa) es mayor que el tamaño de la ED de estaciones o si la estacion de origen es distinta a la de destino y no se encuentra camino posible habiendo hecho una eleccion anterior. Devuelve TRUE si la estacion de origen es igual a la estación de destino o si habiendo elegido una estacion, se encuentra solucion en la siguiente llamada (con estacionOrigen=estacionAdyacente y estación+1)

generarCamino()

Efecto: Método para generar un camino del personaje - Se genera un camino utilizando generarCaminoBT, se introduce en el con setRuta y se introduce el camino en el parámetro de Galaxia pasosPorEstacion.

String getTipo()

Efecto: Método que devuelve el tipo del personaje - Devuelve "FamiliaReal".

Contrabandista:

Atributos:

NINGUNO

Operaciones:

Contrabandista(char marcaClase, String nombre, int estacionPosicion, int turnoInicio)

Efecto: Constructor parametrizado de la clase Personaje - Instacia de Contrabandista inicializada con marcaClase y estacionPosicion por parámetros y midiclorianos como nueva ArrayList. Inserta en la estacionPosicion al personaje.

String getTipo()

Efecto: Método que devuelve el tipo del personaje - Devuelve "contrabandista".

Imperial:

Atributos:

LinkedList<Camino> rutaInicial

Operaciones:

Imperial(char marcaClase, String nombre, int estacionPosicion, int turnoInicio)

Efecto: Constructor parametrizado de la clase Personaje - Instacia de Imperial inicializada con marcaClase y estacionPosicion por parámetros y midiclorianos como nueva ArrayList. Inserta en la estacionPosicion al personaje. rutaInicial es una nueva LinkedList

generarCaminoBT(Grafo grafo, ArrayList<Integer> solucion, int estacionOrigen, int estacionDestino)

Efecto: Método para generar un camino utilizando el recorrido por caminos mínimos - Almacena en solucion el recorrido de estaciones(ID) por las que hay que pasar para llegar desde la estacionOrigen a la estacionDestino.

accionPuerta()

Efecto: Método para acción del personaje Imperial en una puerta - Se guarda la estacionPosicion como EstacionPuerta, se reinicia la puerta de la estación con puerta y se ejecuta el movimiento del personaje. Antes del movimiento se reinicia la ruta.

accionEstacion()

Efecto: Método para acción del personaje Imperial en una estación normal - Si la estacionPosicion tiene identificador par, se inserta ultimo midicloriano del personaje Imperial.

boolean esImperial()

Efecto: Método que devuelve si es Imperial – Devuelve TRUE.

generarCamino()

Efecto: Método para generar un camino de un personaje - Desde la galaxia, se recupera el grafo que une las estaciones y se usa para calcular los caminos. En este caso, el personaje Imperial recorre las estaciones en el orden: EstacionPuerta - Estacion NE - Estacion NO - Estacion SO - EstacionPuerta. Inserta la secuencia de estaciones al personaje a través del método setruta() y a su vez, para que el perssonaje no llegue a algún momento donde se le acaben las orientaciones del camino, se guarda la ruta del personaje en rutaInicial para poder recuperarla.

setMidiclorianos(ArrayList<Midicloriano> midiclorianos)

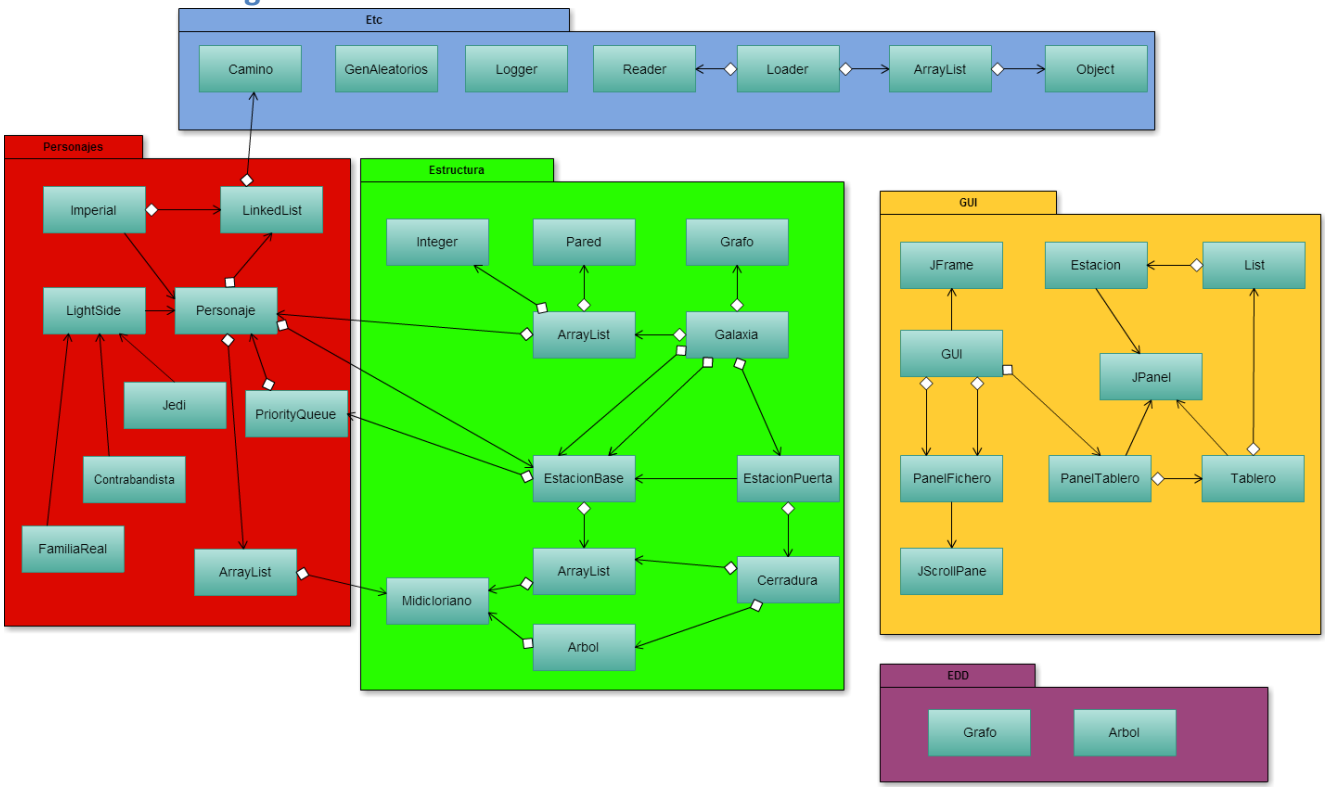
Efecto: Método que inserta un conjunto de Midiclorianos en el Personaje - ED de Personaje con midiclorianos = ED por parámetros.

String getTipo()

Efecto: Método que devuelve el tipo del personaje - Devuelve "imperial".

1.3. Diseño

1.3.1. Diagrama de clases del sistema



1.3.2. Contrato de operaciones a nivel de diseño

Galaxia:

- private Galaxia(int idEstacionPuerta, EstacionPuerta starsgate, int dimX, int dimY)
  - Precondición: -
  - PostCondición: Galaxia inicializado con parámetros dados
  - Complejidad: O(n^2)
- public static Galaxia obtenerInstancia(int idEstacionPuerta, EstacionPuerta starsgate, int dimX, int dimY)
  - Precondición: -
  - Postcondición: Devuelve la instancia de Galaxia. Si no esta inicializada se crea una con parámetros dados
  - Complejidad: O(n^2)
- public static Galaxia obtenerInstancia()
  - Precondición: -
  - Postcondición: Devuelve la instancia de Galaxia.
  - Complejidad: O(1)
- public int getDimX()
  - Precondición: Galaxia inicializada con éxito
  - Postcondición: -
  - Complejidad: O(1)
- public int getDimY()
  - Precondición: Galaxia inicializada con éxito
  - Postcondición: -

    - Complejidad: O(1)
- public Grafo getGrafo()
    - Precondición: Galaxia inicializada con éxito
    - Postcondición: -
    - Complejidad: O(1)
- public int getTurno()
    - Precondición: Galaxia inicializada con éxito
    - Postcondición: -
    - Complejidad: O(1)
- public int getIdEstacionPuerta()
    - Precondición: Galaxia inicializada con éxito
    - Postcondición: -
    - Complejidad: O(1)
- public EstacionBase getEstacionLiberty()
    - Precondición: Galaxia inicializada con éxito
    - Postcondición: -
    - Complejidad: O(1)
- public EstacionPuerta getStarsgate()
    - Precondición: Galaxia inicializada con éxito
    - Postcondición: -
    - Complejidad: O(1)
- public ArrayList<Personaje> getPersonajes()
    - Precondición: Galaxia inicializada con éxito
    - Postcondición: -
    - Complejidad: O(1)
- public void setPasosPorEstaciones(ArrayList<Integer> pasosPorEstaciones)
    - Precondición: Galaxia inicializada con éxito
    - Postcondición: pasosPorEstaciones toma el valor del parámetro de entrada.
    - Complejidad: O(1)
- public void setPersonajes(ArrayList<Object> personajes)
    - Precondición: Galaxia inicializada con éxito
    - Postcondición: parámetro se convierte a ArrayList de personajes y se inserta en personajes
    - Complejidad: O(n)
- private void construirParedesIni()
    - Precondición: DimX y DimY con valores válidos, Grafo y paredes inicialiadas con .
    - Postcondición: Grafo con todos los arcos que unen las estaciones con sus estaciones vecinas (N,S,E,O) y todas las paredes almacenadas en ED paredes, la cual ayudará a implementar algoritmo de Kruskal.
    - Complejidad: O(n)
- private void expandirConexion(int[][] nodos, int valorinicial, int valorfinal)
    - Precondición: Galaxia inicializada con éxito
    - Postcondición: Todos los nodos de la matriz que sean iguales al parámetro valorinicial se convierten en valorfinal

    - Complejidad: O(n^2)
- public void construirGalaxia()
    - Precondición: Galaxia inicializada con éxito
    - Postcondición: Se crean todos los nodos (ID de estaciones) y se llama a construirParedesIni()
    - Complejidad: O(n)
- public void generarLaberinto()
    - Precondición: Galaxia inicializada con éxito
    - Postcondición: Genera un laberinto a través del Algoritmo de Kruskal ayudandose de una matriz de enteros haciendo referencia a los IDS de las galaxias. Sobre esta matriz se realizan los cálculos del ALgoritmo de Kruskal. Sobre el grafo de la galaxia se insertan arcos de las estaciones adyacentes según el Algoritmo.
    - Complejidad: O(n^3)
- public ArrayList<Midicloriano> generarMidiclorianosCerradura()
    - Precondición: Galaxia inicializada correctamente
    - Postcondición: Genera arraylist con midiclorianos con ID desde el 0 hasta el 30
    - Complejidad: O(n)
- public ArrayList<Midicloriano> generarMidiclorianosGalaxia()
    - Precondición: Galaxia inicializada correctamente
    - Postcondición: Genera arraylist con midiclorianos con ID desde el 0 hasta el 30
    - Complejidad: O(n)
- public void repartirMidiclorianos(ArrayList<Midicloriano> midiclorianos)
    - Precondición: Galaxia inicializada con éxito
    - Postcondición: Reparte los midiclorianos de 5 en 5 por las estacione almacenadas pasosPorEstacion (Recorrido de FamiliaReal)
    - Complejidad: O(n^2)
- public void accion(int turno)
    - Precondición: Galaxia inicializada correctamente
    - Postcondición: Recorre todas las estaciones de la galaxia invocando al método
    - * accion de cada una de ellas
    - Complejidad: O(n^3)
- public int[] IDtoCoordenadas(int ID)
    - Precondición: Galaxia inicializada con éxito
    - Postcondición: -
    - Complejidad: O(1)
- public int coordenadastoID(int fila, int columna)
    - Precondición: Galaxia inicializada con éxito
    - Postcondición: -
    - Complejidad: O(1)
- public EstacionBase getEstacion(int ID)
    - Precondición: Galaxia inicializada correctamente

    - Postcondición: Calcula las coordenadas según ID por parámetros y las devuelve en vector de 2 posiciones. Posición 0: Coordenadas X Posición 1 : Coordenadas Y
    - Complejidad: O(1)
- public EstacionBase getEstacion(int fila, int columna)
    - Precondición: Galaxia inicializada correctamente
    - Postcondición: Devuelve la estacion localizada en la determinada fila y columna.
    - Complejidad: O(1)
- public void simular()
    - Precondición: Galaxia inicializada correctamente
    - Postcondición: Acciona la galaxia en un determinado turno y lo muestra en el log
    - Complejidad: O(n^3)
- public String imprimirGalaxia()
    - Precondición: Galaxia inicializada y cargada correctamente
    - Postcondición: Devuelve string con mapa de la galaxia
    - Complejidad: O(n^2)
- public void toLog(int turno)
    - Precondición: Galaxia inicializada con éxito
    - Postcondición: Se escribe en el log información sobre el turno, la estacion de la , llamada al toLog de cerradura de la puerta, información sobre el mapa de la galaxia con imprimirGalaxia y llamda al log de todas las que tengan midiclorianos.
    - Complejidad: O(n^3)

EstacionBase:

- public EstacionBase(int ID)
    - Precondición: -
    - Postcondición: Instancia de EstacionBase creada con ID pasado por parámetros y personajes como una nueva ArrayList por defecto.
    - Complejidad: O(1)
- public int getID()
    - Precondición: EstaciónBase inicializada correctamente
    - Postcondición: -
    - Complejidad: O(1)
- public Object[] getPersonajes()
    - Precondición: EstaciónBase inicializada correctamente
    - Postcondición: -
    - Complejidad: O(1)
- public ArrayList<Midicloriano> getMidiclorianos()
    - Precondición: EstacionBase inicializada con éxito
    - Postcondición: -
    - Complejidad: O(1)
- public void accion(int turno)
    - Precondición: EstaciónBase inicializada correctamente.
    - Postcondición: Ejecuta la accion del personaje o vuelve a meterlo en el final

    - Complejidad: O(n)
- public void insertarMidicloriano(Midicloriano midicloriano)
    - Precondición: EstaciónBase inicializada correctamente.
    - Postcondición: Inserta un midicloriano (parametrizado) en la ED(ordenada)
    - Complejidad: O(1)
- public Midicloriano sacarMidicloriano()
    - Precondición: EstaciónBase inicializada correctamente.
    - Postcondición: ED de midiclorianos con un elemento menos
    - Complejidad: O(1)
- public void insertarPersonaje(Personaje personaje)
    - Precondición: EstaciónBase inicializada con éxito
    - Postcondición: ED de la Base con un personaje más en la última posición
    - Complejidad: O(1)
- public Personaje sacarPersonaje()
    - Precondición: EstacionBase inicializada correctamente
    - Postcondición: ED de la Estación con un personaje menos
    - Complejidad: O(1)
- public boolean esPuerta()
    - Precondición: EstacionBase inicializada correctamente
    - Postcondición: Devuelve false
    - Complejidad: O(1)
- public String imprimirPersonajesMarca()
    - Precondición: Estación inicializad con éxito
    - Postcondición: Si hay mas de un personaje muestra el numero de personajes que hay
    - Complejidad: O(1)
- public String midiclorianosToString()
    - Precondición: EstacionBase inicializada con éxito
    - Postcondición: Recorre todos los midiclorianos y los va almacenando en una cadena
    - Complejidad: O(n)
- public void toLog()
    - Precondición: EstacionBase inicializada con éxito
    - Postcondición: Escribe en el log información sobre el ID y los midiclorianos que
    - * contiene la estación
    - Complejidad: O(n)
- public void toLogPersonajes()
    - Precondición: EstacionBase inicializada con éxito
    - Postcondición: Llama a toLog de todos los personajes que existen en esa estación
    - Complejidad: O(n)

EstacionBase

- public EstacionPuerta(int ID)
    - Precondición: -

    - Postcondición: Estacion inicializada con parámetros dados. Llamada a clase padre ESTACIONBASE
    - Complejidad: O(1)
- public void setCerradura(Cerradura cerradura)
    - Precondición: EstacionPuerta inicializada con éxito
    - Postcondición: EstaciónPuerta con cerradura = cerradura parámetro
    - Complejidad: O(1)
- public void fin()
    - Precondición: EstacionPuerta inicializada correctamente
    - Postcondición: Saca a todos los personajes de las estación y se llama al método fin de cada uno de ellos
    - Complejidad: O(n)
- public boolean esPuerta()
    - Precondición: EstacionBase inicializada correctamente
    - Postcondición: Devuelve true
    - Complejidad: O(1)

Cerradura

- public Cerradura(int alturaDesbloqueo)
    - Precondición: -
    - Postcondición: Inicializa la cerradura con alturaDesbloqueo por parámetros, combinacionInicio con nueva ArrayList de Midiclorianos, combinacionMidiclorianos con nuevo Arbol de Midiclorianos y midiclorianosProbados con nuevo Arbol de Midiclorianos.
    - Complejidad: O(1)
- public boolean Abierta()
    - Precondición: Cerradura inicializada correctamente
    - Postcondición: Cerradura actualiza su estado
    - Complejidad: O(n)
- public void setEstado(boolean estado)
    - Precondición: Cerradura inicializada correctamente
    - Postcondición: Estado de la cerradura con el valor del parámetro
    - Complejidad: O(1)
- public void setCombinacionInicial(ArrayList<Midicloriano> midiclorianosIniciales)
    - Precondición: Cerradura inicializada con éxito.
    - Postcondición: midiclorianos iniciales con valor del parámetro
    - Complejidad: O(1)
- private String mostrarArbol(Arbol<Midicloriano> arbol)
    - Precondición: arbol inicializado con éxito
    - Postcondición: Devuelve cadena con el contenido de los nodos en recorrido inOrden. Método recursivo.
    - Complejidad: O(n)
- private int profundidad(Arbol<Midicloriano> arbol)
    - Precondición: arbol inicializado con éxito
    - Postcondición: Calcula la pronfudidad de cada uno de los hijos del nodo y devuelve la mayor de las dos. Método recursivo

    - Complejidad: O(n)
- private int nodosInternos(Arbol<Midicloriano> arbol)
    - Precondición: arbol inicializado con éxito
    - Postcondición: Calcula la cantidad de nodos internos sumando los nodos internos en el hijo izquierdo más los del derecho más, en el caso de que la raíz fuera nodo interno, uno. Método recursivo.
- private int nodosExternos(Arbol<Midicloriano> arbol)
    - Precondición: arbol inicializado con éxito
    - Postcondición: Calcula nos nodos externos sumando los nodos externos del hijo izquierdo más los del hijo derecho más, en el caso de que la raiz fuese nodo externo, uno.
    - Complejidad: O(n)
- private void vaciarArbol(Arbol<Midicloriano> arbol)
    - Precondición: arbol inicializado con éxito
    - Postcondición: Vacia el arbol asignado un nuevo arbol al parámetro y llamando al recolector de basura.
    - Complejidad: O(1)
- private void combine(ArrayList<Midicloriano> midiclorianos, ArrayList<Midicloriano> midiclorianosCombinados, int min, int max)
    - Precondición: midiclorians inicializado correctamente && combinedMidiclorians inicializado correctamente && low > 0 && top > 0
    - Postcondición: midiclorianos de midiclorians insertados de forma balanceada en combinedMidiclorians
    - Complejidad: O(n)
- public void configurarCerradura(ArrayList<Midicloriano> midiclorianos)
    - Precondición: Cerradura inicializada correctamente
    - Postcondición: Combinación cerradura con datos de midiclorianos (parámetro entrada)
    - Complejidad: O(n)
- public void generarCombinacion()
    - Precondición: Cerradura inicializada correctamente
    - Postcondición: - CombinacionInicio con la combinación inicial de la cerradura y CombinacionCerradura con midiclorianos balanceados
    - Complejidad: O(n)
- public boolean probarMidicloriano(Midicloriano midicloriano)
    - Precondición: Cerradura inicializada correctamente
    - Postcondición: Midicloriano probado en la puerta. ED de midiclorianos probado con elemento más si no había sido probado anteriormente
    - Complejidad: O(n)
- public void reiniciar()
    - Precondición: Cerradura inicializada correctamente
    - Postcondición: Cerradura inicializada con configuración inicial y cerrada
    - Complejidad: O(n)

- public void comprobarEstado()
    - Precondición: Cerradura inicializada correctamente
    - Postcondición: Actualiza el estado de la puerta según condiciones de apertura
    - Complejidad: O(n)
- public void toLog()
    - Precondición: Cerradura inicializado con éxito
    - Postcondición: Escribe en el log el estado de la puerta, los midiclorianos de la combinación de la cerradura y los midiclorianos probados
    - Complejidad: O(n)

Midicloriano:

- public Midicloriano(int ID)
    - Precondición: -
    - Postcondición: - Midicloriano creado con ID del parámetro
    - Complejidad: O(1)
- public int getID()
    - Precondición: Midicloriano inicializado correctamente
    - Postcondición: -
    - Complejidad: O(1)
- public String toString()
    - Precondición: Midicloriano inicializado correctamente
    - Postcondición: -
    - Complejidad: O(1)
- public int compareTo(Midicloriano o)
    - Precondición: Midicloriano inicializado correctamente
    - Postcondición: -
    - Complejidad: O(1)

Personaje:

- public Personaje(char marcaClase, String nombre, int estacionPosicion, int turnoInicio)
    - Precondición: -
    - Postcondición: Instacia de Personaje inicializada con marcaClase y
    - * estacionPosicion por parámetros y midiclorianos como nueva ArrayList. Inserta en la estacionPosicion al personaje.
    - Complejidad: O(1)
- public int getTurno()
    - Precondición: Personaje inicializado correctamente
    - Postcondición: -
    - Complejidad: O(1)
- public String getNombre()
    - Precondición: Personaje inicializado correctamente
    - Postcondición: -
    - Complejidad: O(1)
- public EstacionBase getEstacionPosicion()
    - Precondición: Personaje inicializado correctamente

    - Postcondición: -
    - Complejidad: O(1)
- public char getMarcaClase()
    - Precondición: Personaje inicializado correctamente
    - Postcondición: -
    - Complejidad: O(1)
- public boolean isGanador()
    - Precondición: Personaje inicializado correctamente
    - Postcondición: -
    - Complejidad: O(1)
- public LinkedList<Camino> getRuta()
    - Precondición: Personaje inicializado correctamente
    - Postcondición: -
    - Complejidad: O(1)
- public void setRuta(LinkedList<Camino> ruta)
    - Precondición: Personaje inicializado correctamente
    - Postcondición: Ruta del Personaje = RUta por parámetros
    - Complejidad: O(1)
- public void setRuta(ArrayList<Integer> camino)
    - Precondición: Personaje inicializado correctamente, estacionPosicion no se encuentra en la ED camino
    - Postcondición: Se inserta en la ruta del personaje una serie de secuencia de orientaciones acorde con la secuencia de estaciones introducida por parámetro.
    - Complejidad: O(n)
- protected void moverA(EstacionBase estacion)
    - Precondición: Personaje inicializado correctamente
    - Postcondición: Estacion por parámetros: Nuevo Personaje(instacia actual del personaje)
    - Complejidad: O(1)
- private EstacionBase getSiguienteEstacion()
    - * @pre Personaje inicializado correctamente
    - Postcondición: Primera orientacion de la ruta es sacada y se vuelve a insertar en la ED de la ruta, y se ejecutan los métodos para calcular siguiente posicion con ayuda de dos variables (fila y columna).
    - Complejidad: O(1)
- public void recogerMidicloriano(Midicloriano midicloriano)
    - Precondición: Personaje inicializado correctamente
    - Postcondición: ED de midiclorianos con un elemento más (última posición)
    - Complejidad: O(1)
- public void mover()
    - Precondición: Personaje inicializado correctamente
    - Postcondición: El personaje se mueve a la siguiente estación
    - Complejidad: O(1)
- public void accion()
    - Precondición: Personaje inicializado correctamente e insertado dentro de la Galaxia

    - Postcondición: Si el personaje está en una estación, se ejecuta accionEstacion(). SI el personaje está en una puerta, se ejecuta accionPuerta(). En cualquiera de los casos se incrementa el turno en 1.
    - Complejidad: O(1)
- public boolean esImperial()
    - Precondición: Personaje inicializado correctamente
    - Postcondición: -
    - Complejidad: O(1)
- public void fin()
    - Precondición: Personaje inicializado correctamente
    - Postcondición: Establece ganador a true y mueve al personaje a la estación de los ganadores.
    - Complejidad: O(1)
- public int compareTo(Personaje o)
    - Precondición: Personaje inicializado correctamente
    - Postcondición: Devuelve 1(mayor)
    - Complejidad: O(1)
- public String toString()
    - Precondición: Personaje inicializada correctamente
    - Postcondición: Devuelve marca de clase
    - Complejidad: O(1)
- public String rutaToString()
    - Precondición: Personaje inicializada correctamente
    - Postcondición: -
    - Complejidad: O(n)
- public String midicloriansToString()
    - Precondición: Personaje inicializada correctamente
    - Postcondición: -
    - Complejidad: O(n)
- public void toLogini()
    - Precondición: Personaje inicializada correctamente
    - Postcondición: -
    - Complejidad: O(1)
- public void toLog()
    - Precondición: Personaje inicializada correctamente
    - Postcondición: -
    - Complejidad: O(n)

Imperial

- public Imperial(char marcaClase, String nombre, int estacionPosicion, int turnoInicio)
    - Precondición: -
    - Postcondición: Instacia de Imperial inicializada con marcaClase y estacionPosicion por parámetros y midiclorianos como nueva ArrayList. Inserta en la estacionPosicion al personaje. rutaInicial es una nueva LinkedList
    - Complejidad: O(1)

- private void generarCaminoBT(Grafo grafo, ArrayList<Integer> solucion, int estacionOrigen, int estacionDestino)
    - Precondición: Grafo, solucion y personaje inicializado con éxito
    - Postcondición: Almacena en solucion el recorrido de estaciones(ID) por las que hay que pasar para llegar desde la estacionOrigen a la estacionDestino
    - Complejidad: O(n)
- public void accionPuerta()
    - Precondición: Imperial inicializado con éxito
    - Postcondición: Se guarda la estacionPosicion como EstacionPuerta, se reinicia la puerta de la estación con puerta y se ejecuta el movimiento del personaje. Antes del movimiento se reinicia la ruta.
    - Complejidad: O(n)
- public void accionEstacion()
    - Precondición: Imperial inicializado con éxito
    - Postcondición: Si la estacionPosicion tiene identificador par, se inserta ultimo midicloriano del personaje Imperial.
    - Complejidad: O(1)
- public boolean esImperial()
    - Precondición: Imperial inicializado correctamente
    - Postcondición: -
    - Complejidad: O(1)
- public void generarCamino()
    - Precondición: Personaje inicializado con éxito
    - Postcondición: Desde la galaxia, se recupera el grafo que une las estaciones y se usa para calcular los caminos. En este caso, el personaje Imperial recorre las estaciones en el orden: EstacionPuerta - Estacion NE - Estacion NO - Estacion SO - EstacionPuerta. Inserta la secuencia de estaciones al personaje a través del método setruta() y a su vez, para que el perssonaje no llegue a algún momento donde se le acaben las orientaciones del camino, se guarda la ruta del personaje en rutaInicial para poder recuperarla.
    - Complejidad: O(n)
- public void setMidiclorianos(ArrayList<Midicloriano> midiclorianos)
    - Precondición: Personaje inicializado correctamente
    - Postcondición: ED de Personaje con midiclorianos = ED por parámetros
    - Complejidad: O(1)
- public String getTipo()
    - Precondición: Imperial inicializado correctamente
    - Postcondición: Devuelve "imperial"
    - Complejidad: O(1)

LightSide

- public LightSide(char marcaClase, String nombre, int estacionPosicion, int turnoInicio)
    - Precondición: -

    - Postcondición: Instacia de LightSide inicializada con marcaClase y estacionPosicion por parámetros y midiclorianos como nueva ArrayList. Inserta en la estacionPosicion al personaje. Inicializa rutaInicial como nueva LinkedList.
    - Complejidad: O(1)
- public boolean estacionvisitada(ArrayList<Integer> solucion, int estacionActual, int estacionComprobar)
    - Precondición: Familia real y solucion inicializada con éxito
    - Postcondición: Devuelve si la estacionComprobar se encuentra en solucion (hasta estacionActual)
    - Complejidad: O(n)
- public void accionPuerta()
    - Precondición: LightSide inicializado correctamente e insertado dentro de la Galaxia
    - Postcondición: Saca un midicloriano del LightSide con scarMidicloriano(), guarda la estacionPosicion como una EstacionPuerta, y si el midicloriano sacada no es null, lo prueba en la cerradura la estacion con puerta.
    - Complejidad: O(n)
- public void accionEstacion()
    - Precondición: LightSide inicializado correctamente e insertado dentro de la Galaxia
    - Postcondición: Guarda la estacionPosicion, saca un midicloriano de la estacion con sacarMidicloriano(), y si el midicloriano sacado es distinto de null, se almacena en la ED de midiclorianos del LightSide con recogerMidicloriano()
    - Complejidad: O(1)
- private boolean movimientoPosible(Grafo grafo, int estacion, Camino orientacion, int[] siguienteEstacion)
    - Precondición: Galaxia y grafo inicializado correctamente
    - Postcondición: Se obtienen las coordenadas de origen y destino (con ayuda de la orientacion introducida por parámetros), se obtiene el ID destino y se devuelve en siguienteEstacion[0]
    - Complejidad: O(1)
- private void generarCaminoBT(Grafo grafo, ArrayList<Integer> solucion, int estacionOrigen, int estacionDestino, Camino orientacion)
    - Precondición: Galaxia y grafo inicializado con con éxito
    - Postcondición: Si la estacionOrigen es distinta a la estacionDestino se comprueban los 4 caminos posibles dependiendo de la orientación (siempre gira a la derecha cuando hay posibilidad). Método recursivo.
    - Complejidad: O(n^2)
- public void generarCamino()
    - Precondición: LightSide inicializado correctamente
    - Postcondición: La ruta del personaje es igual a la ruta calculada con generar caminoBT. La primera llamada al módulo se realiza con la orientación sur suponiendo que el personaje está mirando hacia abajo. Se inserta la estacionPosicion donde se encuentra,

antes de la llamada recursiva; y despues de la llamada se elimina ya para borrar la estación origen.
    - Complejidad: O(n^2)
- public String getTipo()
    - Precondición: LightSide inicializado correctamente
    - Postcondición: Devuelve "lightside"
    - Complejidad: O(1)

FamiliaReal

- public FamiliaReal(char marcaClase, String nombre, int estacionPosicion, int turnoInicio)
    - Precondición: -
    - Postcondición: Instacia de Familia Real inicializada con marcaClase y estacionPosicion por parámetros y midiclorianos como nueva ArrayList. Inserta en la estacionPosicion al personaje.
    - Complejidad: O(1)
- private boolean generarCaminoBT(Grafo grafo, ArrayList<Integer> solucion, int estacion, int estacionOrigen, int estacionDestino, int tam)
    - Precondición: Grafo y solucion inicializadas con éxito
    - Postcondición: Devuelve false si la estacion(etapa) es mayor que el tamaño de la ED de estaciones o si la estacion de origen es distinta a la de destino y no se encuentra camino posible habiendo hecho una eleccion anterior. Devuelve TRUE si la estacion de origen es igual a la estación de destino o si habiendo elegido una estacion, se encuentra solucion en la siguiente llamada (con estacionOrigen=estacionAdyacente y estcion+1)
    - Complejidad: O(n^2)
- public void generarCamino()
    - Precondición: Personaje y galaxia inicializada con éxito
    - Postcondición: Se genera un camino utilizando generarCaminoBT, se introduce en el personaje con setRuta y se introduce el camino en el parámetro de Galaxia pasosPorEstacion.
    - Complejidad: O(n^2)
- public String getTipo()
    - Precondición: FamiliaReal inicializado correctamente
    - Postcondición: Devuelve "FamiliaReal"
    - Complejidad: O(1)

Jedi

- public Jedi(char marcaClase, String nombre, int estacionPosicion, int turnoInicio)
    - Precondición: -
    - Postcondición: Instacia de Jedi inicializada con marcaClase y estacionPosicion por parámetros y midiclorianos como nueva ArrayList. Inserta en la estacionPosicion al personaje.
    - Complejidad: O(1)
- public String getTipo()
    - Precondición: Jedi inicializado correctamente

- Postcondición: Devuelve "jedi"
- Complejidad: O(1)

Contrabandista

- public Contrabandista(char marcaClase, String nombre, int estacionPosicion, int turnoInicio)
  - Precondición: -
  - Postcondición: Instacia de Contrabandista inicializada con marcaClase y estacionPosicion por parámetros y midiclorianos como nueva ArrayList. Inserta en la estacionPosicion al personaje.
  - Complejidad: O(1)
- public String getTipo()
  - Precondición: Contrabandista inicializado correctamente
  - Postcondición: Devuelve "contrabandista"
  - Complejidad: O(1)

1.3.3. Estructuras de Datos utilizadas

Se han utilizado las siguientes Estructuras de datos en el proyecto.

- ArrayList: Se ha utilizado para guardar datos que pueden estar repetidos sin necesidad de ninguna relación entre ellos. En algunos casos se ha forzado la inserción en la primera posición o en la última. Se ha utilizado en Galaxia para almacenar Personajes, en Galaxia para almacenar paredes, en EstacionBase para almacenar midiclorianos, en Cerradura para almacenar Midiclorianos, en Personajes para almacenar midiclorianos y en Loader para almacenar Object.
- LinkedList: Se ha utilizado para guardar datos en los que los datos debían ir ordenados según su orden de inserción. Se ha utilizado como una pila. Se ha utilizado en Personaje para almacenar Caminos.
- PriorityQueue: Se ha utilizado para guardar datos que debían almacenarse de manera ordenada según el comparador de objetos. Se ha manejado como una cola. Se ha utilizado en EstacionBase para almacenar Personajes.
- Grafo: Se ha utilizado para guardar datos, y sus relaciones (conexiones) entre ellos. Se ha utilizado para la construcción del laberinto en la Galaxia.
- Arbol: Se ha utilizado este tipo de EDD ya que se necesitaba guardar elementos de forma ordenada y sobre estos, realizar búsquedas y borrados constantes. Para mejorar la eficiencia de este proceso se ha decidido utilizar un árbol. Además, se ha aprovechado las características del árbol para establecer interruptores para abrir la cerradura de la EstacionPuerta. Como acabamos de decir, hemos aplicado el árbol para almacenar midiclorianos en la cerradura.

1.4. Implementación

1.4.1. Algoritmos de especial interés

Los algoritmos más interesantes del código son:

- Algoritmo de la mano derecha: Algoritmo que simula que personaje pega su mano derecha a la pared y continúa andando hasta la salida sin despegar la mano de la pared. A continuación, se adjunta el código:

```
private void generarCaminoBT(Grafo grafo, ArrayList<Integer> solucion, int estacionOrigen, int
estacionDestino, Camino orientacion) {
        if (estacionOrigen != estacionDestino) {
            boolean caminoencontrado = false;
            for (int i = 0; i < 4 && !caminoencontrado; i++) {
                Camino caminocomprobar = null;
                switch (orientacion) {
                    case ESTE:
                        caminocomprobar = Camino.SUR;
                        break;
                    case SUR:
                        caminocomprobar = Camino.OESTE;
                        break;
                    case OESTE:
                        caminocomprobar = Camino.NORTE;
                        break;
                    case NORTE:
                        caminocomprobar = Camino.ESTE;
                        break;
                    default:
                        throw new AssertionError(orientacion.name());
                }
                int[] siguienteEstacion = new int[1];
                if (movimientoPosible(grafo, estacionOrigen, caminocomprobar, siguienteEstacion)) {
                    if   (solucion.size()   >   1   &&   solucion.get(solucion.size()   -   2)   ==
siguienteEstacion[0]) {
                        orientacion = caminocomprobar;
                    } else {
                        solucion.add(siguienteEstacion[0]);
                        generarCaminoBT(grafo,   solucion,   siguienteEstacion[0],   estacionDestino,
caminocomprobar);
                        caminoencontrado = true;

                    }
                } else {
                    if (movimientoPosible(grafo, estacionOrigen, orientacion, siguienteEstacion)) {
                        solucion.add(siguienteEstacion[0]);
                        generarCaminoBT(grafo,   solucion,   siguienteEstacion[0],   estacionDestino,
orientacion);
                        caminoencontrado = true;
                    }
                    orientacion = caminocomprobar;
                }
            }
        }
    }
```

- Algoritmo de Kruskal: El algoritmo de Kruskal es un algoritmo de la teoría de grafos para encontrar un árbol recubridor mínimo en un grafo conexo y ponderado. Es decir, busca un subconjunto de aristas que, formando un árbol, incluyen todos los vértices y donde el valor total de todas las aristas del árbol es el mínimo. Si el grafo no es conexo, entonces busca un bosque expandido mínimo (un árbol expandido mínimo para cada componente conexa). El algoritmo de Kruskal es un ejemplo de algoritmo voraz. A continuación se adjunta el código:

```
  public void generarLaberinto() {
       //Nodos auxiliares con ID de estaciones
       int[][] nodos = new int[dimX][dimY];
       for (int i = 0; i < nodos.length; i++) {
           for (int j = 0; j < nodos[i].length; j++) {
               nodos[i][j] = Estaciones[i][j].getID();
```

```
        }
    }
    int npared; //Enesima pared
    Pared pared; //Variable para alamecenar pared
    //Variables para almacenar coordenadas de origen y destino
    int[] coordorigen, coordestino;
    int nodorigen, nododestino;
    while (!paredes.isEmpty()) {
        npared = GenAleatorios.generarNumero(paredes.size());
        pared = paredes.remove(npared);
        coordorigen = IDtoCoordenadas(pared.origen);
        coordestino = IDtoCoordenadas(pared.destino);
        nodorigen = nodos[coordorigen[0]][coordorigen[1]];
        nododestino = nodos[coordestino[0]][coordestino[1]];
        if (nodorigen != nododestino) {
            grafo.nuevoArco(pared.origen, pared.destino, 1);
            grafo.nuevoArco(pared.destino, pared.origen, 1);
            expandirConexion(nodos, nododestino, nodorigen);
        }
    }
    grafo.floyd();
    grafo.warshall();
}
```

1.4.2. Definición de entradas/salidas

La entrada en el programa está formada por un archivo de texto .txt donde se especifica la configuración de la Galaxia y los personajes que se encuentran en ella. De cada personaje se especifica el nombre, el tipo, la marca del personaje y el turno de salida. A continuación, se muestra un ejemplo de fichero de entrada de una galaxia 6x6:

```
--Galaxia
GALAXIA#6#6#35#5#
--Personajes de la simulación
FAMILIAREAL#Leia#L#1#
JEDI#LukeSkyWalker#S#2#
IMPERIAL#DarthVader#D#1#
CONTRABANDISTA#HanSolo#H#1#
```

La salida del programa está formada por un fichero de texto .log en donde se especifica el estado del tablero de la galaxia por cada turno, junto con la información de la cerradura, de las estaciones que contienen midiclorianos y de los personajes. A continuación, se muestra un fragmento de archivo log en un determinado turno:

```
(turno:1)
(galaxia:35)
(puerta:cerrada:5: 1 3 5 7 9 11 13 15 17 19 21 23 25 27 29:)
_ _ _ _ _ _ _
|S L  |_ _  |
|_    |_    |
|_ _|_  | |_|
|  _     _| |
|_ _|_|  _ D|
|_ H _ _ _|_|
(estacion:0: 0 1 1 2 3)
(estacion:1: 4 5 5 6)
(estacion:2: 7 7 8 9 9)
(estacion:8: 10 11 11 12 13)
(estacion:14: 13 14 15 15 16)
(estacion:15: 17 17 18 19 19)
(estacion:21: 20 21 21 22 23)
(estacion:27: 23 24 25 25 26)
```

```
(estacion:28: 27 27 28 29 29)
(jedi:S:0:2:)
(familiareal:L:1:1: 3)
(imperial:D:29:1: 29 27 25 23 21 19 17 15 13 11 9 7 5 3 1)
(contrabanadista:H:31:1:)
```

1.4.3. Variables o instancias más significativas y su uso

Las variables más significativas y más usadas en el proyecto son:

- Grafo: variable de tipo grafo ubicada en Galaxia es sin duda una de las variables más importantes ya que todos los algoritmos de generación de ruta dependen de ella.
- Todas las variables que almacenan midiclorianos en todas las clases son muy importantes ya que los midiclorianos, la unidad más básica del proyecto, es una de las más importantes. Esta importancia es debida a que la finalidad del juego consiste en abrir una puerta que se acciona con midiclorianos. Usadas en la Cerradura y en los personajes.

1.5. Lista de errores que el programa controla

El control de errores se encuentra en:

- GUI: Líneas 140-150. Se controla que el archivo de inicio se lea de manera correcta. Si se encuentra un posible fallo se muestra un mensaje al usuario indicándole que hay un error en el fichero de inicio.
- GUI: Líneas 163-171. Se controla que el archivo de log se ha guardado correctamente en la ruta especificada. Si se encuentra un posible fallo debido, por ejemplo, a que la ruta no está disponible, se muestra un mensaje de error indicándole al usuario que ha habido un fallo al salvar/guardar el archivo.
- GUI: Lineas 185-203. Se controla que el archivo formado por el texto en la pestaña es capaz de guardarse en la ruta "./files/inicio.txt". Si no es capaz de acceder a esa ruta para guardar el archivo, muestra al usuario un mensaje en un diálogo comunicándole el error.
- GUI: Lineas 190-200. Se controla que no se produzca ningún NullPointerException causado por un fallo en el contenido del texto de la pestaña inicio. Si se produce un error se le comunica al usuario mediante un diálogo emergente.
- Logger: Líneas 85-92. Se controla si se puede crear un archivo en la ruta especificada. Si no se puede crear debido a un fallo de acceso se muestra un mensaje de error por la consola.
- LightSide: Líneas 103-107. Se controla que al probar un midicloriano en la puerta no se produce una interrupción debido a un fallo en el árbol. Si se produce un fallo, en el log se muestra un mensaje de advertencia del fallo (se usa un prefijo de advertencia).

1.6. Pruebas

Las pruebas utilizadas en este proyecto se basan en el FrameWork de Java Junit. JUnit es un conjunto de clases (framework) que permite realizar la ejecución de clases Java de manera controlada, para poder evaluar si el

funcionamiento de cada uno de los métodos de la clase se comporta como se espera. Es decir, en función de algún valor de entrada se evalúa el valor de retorno esperado; si la clase cumple con la especificación, entonces JUnit devolverá que el método de la clase pasó exitosamente la prueba; en caso de que el valor esperado sea diferente al que regresó el método durante la ejecución, JUnit devolverá un fallo en el método correspondiente. Se han realizado pruebas unitarias en todos los métodos del proyecto.

1.7. Historial de desarrollo

El proyecto se puede separar en 3 fases de desarrollo:

1. Análisis: Esta primera fase es la que más nos dio que pensar debido a que había varias maneras de representar los datos. Queríamos desarrollar bien la estructuración del proyecto; ya que si no tendríamos que estar cambiando continuamente el proyecto. Después de 2 semanas de encuentros diarios, y con ayuda del profesor de la asignatura, jerarquizamos las clases de la manera más óptima posible.
2. Desarrollo: Tras tener el esquema general del proyecto nos pusimos a pensar la mejor forma de implementarlo. En esta tarea nos dividimos el trabajo para así, en el siguiente encuentro, poder debatir sobre lo desarrollado. Esto nos funcionó muy bien ya que uno veía los fallos y mejorar que el otro no veía.
3. Renovación y conclusión: Tras el desarrollo de todo el proyecto faltaban atar algunos cabos sueltos como métodos que no servían para nada y operaciones iguales llamadas de formas distintas. Rehicimos el proyecto teniendo de base el ya terminado. En esta fase nos dimos cuenta de muchos fallos que antes eran imperceptibles.

VALORACIÓN FINAL DEL PROYECTO

El proyecto en sí es una buena práctica, para aprender el lenguaje de programación Java, tanto para el que empieza como para crecer el que ya sabe algo al respecto, aprender técnicas de programación y algoritmos comprobar y aprender de los errores de código, que a nuestra opinión es donde de verdad se está aprendiendo.

Resulta una práctica muy amena ya que se trata de aplicar todos los conocimientos obtenidos en las clases de teoría y una gran ayuda a problemas que surgen es el foro, en el que interviene la comunidad de compañeros para ayudarnos unos a otros ya que los problemas de uno puede haberlos solucionado otro.

Con la temática pienso que se ha ayudado a aumentar la motivación en algún aspecto ya que era bastante atractiva con respecto a personalidad de los alumnos edad etc.

Al ser una práctica extensa, creo que los conocimientos a nivel profesional se interiorizan bien y queda el proyecto como una herramienta de consulta

práctica sobre patrones, métodos de programación y consulta de funciones o algoritmos.

También al tener las entregas programadas se fuerza a presionar a los alumnos y eso es bueno, ya que se aprende a concentrarse puramente en el proyecto y es usual en el mundo de la informática trabajar bajo presión, con lo cual pienso que también es una forma de conocer y trabajar bajo la presión de una fecha, aunque los programas informáticos dependen de muchos factores y siempre la fecha es aproximada.

Al trabajar en equipo también se aprende uno de otro, se ven los fallos del otro y ves donde te has equivocado, aprendes a trabajar en equipo y aprendes de los intereses del compañero que casi siempre sabe cosas diferentes que nosotros no sabíamos o conocíamos.

También el “pique” sano del ambiente de clase porque tu proyecto sea mejor que el del otro equipo alienta a incrementar habilidades conocimientos y finalmente a aprender más.