Javier Gómez, Carlos Li

Ejercicio 1

Para el ejercicio 1 hemos optado por la implementación de una matriz dispersa, pues consume menos recursos que un matriz usual al no tener todos los elementos cargados en memoria.
Esta implementación la hemos ido mejorando conforme avanzaba la práctica, así pues, para el correcto tratamiento de los elementos vacíos, hemos tenido que crear las clases VirtualMatrixElementImpl así como una factoría que es quien se encarga de crear los elementos (reales o virtuales) cuando la matriz lo solicite.
De esta forma, la genericidad queda intacta tanto para el contenido de los elementos de la matriz, como para los elementos en si. Pudiendo, si se implementan tanto los elementos como la factoría de otra forma, tener matrices de dichos elementos.

Ejercicio 2

Para el ejercicio 2 hemos creado clases estáticas dentro de nuestra clase MatrixImpl que contienen los métodos necesarios para ordenar los elementos tal y como se pedia en el enuncuado.

Ejercico 3

Para el ejercicio 3 hemos tenido que ampliar ciertas interafces porque considerábamos necesarios tener ciertos métodos que las interfaces originales no nos daban.
Por lo demás, no hay mucho que comentar.

Ejercicio 4

Para el ejercicio 4, la única decisión de diseño notable que hemos tomado ha sido que a los agentes les incluíamos un atributo en el que guardar la celda a la que fueran a moverse y con el método moveTo, no se movía como tal sino que se actualizaba el valor de dicho campo. Es con una llamada al método move que se actualiza la posición de dicho agente.
Además de esto, nos hemos visto obligados a incluir un casting por tener que reutilizar partes del apartado anterior (los IbasicAgent no disponen del metodo run)

Ejercicio 5

Para este ejercicio, no hemos tomado decisiones de diseño demasiado importantes a parte de la siguiente:
Para poder demostrar que los estados cambian con la invocación del método changeState, lo hemos invocado antes de cada ejecución.