**Manual Técnico**

**Personajes.c (TAD)**

**Descripción:** en este módulo se encuentran las rutinas que hacen referencia a todo lo relacionado con los personajes que tendrán aparición en nuestro juego.

**Estructura:** empezamos definiendo la estructura que tendrán todos ellos.

El primer campo está reservado para el nombre del personaje, es una cadena de caracteres dinámica que realizará la gestión de memoria en función de la longitud introducida.

A continuación tenemos el campo de la descripción, que consiste, al igual que el nombre, en una cadena de caracteres dinámica. Será utilizada para guardar lo que piensa el jugador sobre el personaje en cuestión.

Otro campo es el diálogo, como en las anteriores, una cadena de caracteres donde guardaremos lo que dice dicho personaje al interactuar con él.

Tras este, tenemos el campo de la ubicación del personaje, el cual nos indica con dos enteros la posición en el mapa de esa persona.

Para terminar, nos encontramos con el campo donde guardaremos los objetos del personaje, que se basa en una lista dinámica donde podremos encontrar dichos objetos.

**Funciones:** partimos de las funciones básicas de crear y liberar dicho personaje.

Junto a ellas, continuamos con otras, también muy características, como son modificar cada uno de los campos de la estructura dependiendo de los argumentos de entrada deseados (las funciones de set).

Además, en muchos casos necesitamos realizar consultas sobre la información como obtener dichos valores (las funciones de get).

También disponemos de una función que compara dos personas para decir si son iguales, (obviando los objetos que posean cada una).

**List.c (TAD)**

**Descripción:** este módulo lo hemos diseñado para un mejor manejo de la información, sobre todo para crear listas de objetos que posteriormente añadimos a un espacio o a un personaje.

**Estructura:** en la lista encontramos dos estructuras diferenciadas.

La primera es la propia estructura de lista, en la cual podemos ver que está compuesta por un puntero al primer nodo de la misma, que nos servirá de referencia para manejar operaciones con ella más adelante.

La otra estructura es la que tiene que ver con los nodos. Éstos son lo que utilizamos como “cajas” dentro de la lista para ir avanzando hasta la que nos interesa. Los nodos estás compuestos de dos campos: uno de ellos es el objeto en sí (la información) y el siguiente es una referencia al siguiente nodo de la lista (next).

**Funciones:** respecto a las funciones, encontramos las básicas de crear y destruir.

También encontramos las funciones de comprobar si está vacía y saber el tamaño de la lista, es decir, cuantos elementos tiene.

En cuanto a las de modificarla, tenemos insertar un objeto por el principio, extraer por el principio y extraer un objeto independientemente de su posición, solamente con el nombre del mismo podremos sacarlo y la lista se reorganizará tras ello.

**Gestor.c**

**Descripción:** con este módulo podemos realizar las operaciones de imprimir por pantalla los campos relacionados con cada TAD (espacio, personajes, objetos).

**Funciones:** el gestor se compone de funciones independientes que imprimen los datos de objetos (descripción, nombre, ubicación...), espacios (descripción, coordenadas, personajes del mismo...) y de personajes (diálogo, nombre, ubicación...).

Además, este módulo cuenta con una función para cada TAD que incluye a su vez las funciones independientes de cada uno, por lo que con una simple función somos capaces de imprimir los valores de la estructura de los un tipo de dato. Obviamente, tenemos 3 funciones genéricas como la que hemos nombrado, cada una asociada a su correspondiente personaje, espacio u objeto.

**Lector.c**

**Descripción:** este módulo es el que utilizamos para cargar los datos de un fichero de entrada (generalmente el nuestro, ya que está organizado acorde al lector) y guardarlo en sus tablas correspondientes (espacios, personajes y objetos).

**Funciones:** el lector tiene una función pública que es la que utilizaremos en los main, en la que pasamos la tabla de espacios, personajes y objetos, el fichero del que leer los datos y una cadena de números, que utilizaremos en otras funciones para saber el número de elementos. Respecto a las funciones privadas, nos encontramos con un total de 6, que son 2 para leer objetos, otras 2 para leer personajes y otras 2 para leer espacios.

**Espacios.c (TAD)**

**Descripción: e**n este apartado nos encargamos de explicar cómo se distribuyen los espacios y como los hemos estructurado.

**Estructura:** la estructura de los espacios es bastante compleja, pues en ella tenemos la estructura pública ESPACIO, pero tenemos varias privadas dentro.

Un espacio tiene asociadas unas coordenadas, una descripción para cuando lo examinemos, u estado (0 es oscuro, 1 iluminado y 2 lluvia), una lista de objetos y un array de 4 nodos.

Un nodo tiene asociados una dirección cardinal (NORTE, SUR, ESTE U OESTE), una puerta (un objeto que puede estar abierto o cerrado) y un espacio.

Luego hemos necesitado usar varias estructuras auxiliares para manejar los espacios, entre ellas una lista enlazada AUXLIST que contiene LISTNODES que como nodos conectan con un nodo padre y otro nodo hijo y tienen asociados un espacio, todo esto para poder desplazarse entre espacios adyacentes hacia delante y poder volver por donde uno ha venido .

Luego hemos usado también árboles pues son más cómodos para guardar espacios ordenados y buscar caminos más cortos de forma recursiva. El árbol está compuesto por nodos, cada nodo tiene asociado 1 nodo padre y 3 nodos hijos (pues hay 4 direcciones de espacios) y cada nodo tiene asociado un espacio.

**Funciones:** como funciones públicas tenemos las típicas de inicializar y destruir espacios, settear los campos, insertar en la lista de objetos del espacio o extraer de él. Y luego los gets, que te devuelven los campos que solicites en general.

Para implementar las últimas funciones hemos tenido que hacer uso de árboles debido a su frecuente uso como búsqueda en anchura, que nos garantiza encontrar caminos más cortos de sala a sala. El algoritmo que aplicamos en general para recorrer las salas se basa en crear una lista de espacios cerrados (a los que ya has accedido) y otra lista de abiertos (que aún no has accedido) y según los clasificas vas metiéndolos en un árbol para luego recorrerlo y encontrar los caminos más cortos.

Luego hay funciones más interesantes de comentar como el de obtener la distancia entre espacios, pues al ser una matriz de 2 dimensiones y sólo se puede avanzar en las 4 direcciones cardinales del espacio, pues la distancia se halla calculando la diferencia entre las posiciones del “eje x” y del “eje y”. Y move\_to\_room por ejemplo que te calcula el camino más corto entre 2 salas y te devuelve el primer paso para ese trayecto. Todo esto calculándolo mediante uso de árboles y listas d espacios.

Y la función rooms\_at\_dist que te calcula los espacios a distancia dist de tu ubicación.

**Acciones.c**

**Descripción:** en este módulo incluiremos todas las acciones que tendrá disponible el jugador como moverse de sala, coger y dejar objetos, encenderlos, apagarlos, abrir puertas, hablar, examinar y dormir y despertarse.

**Funciones: e**n general en casi todas las funciones pasaremos como argumentos el mapa de espacios, la tabla de personajes guardados, el personaje del jugador y un fichero donde escribir. Y devolverán OK o ERROR para ver si han funcionado.

En general, cuando cogemos como argumento el mapa de espacios, lo usamos para generalizar la entrada de argumentos y facilitarla, haciendo que para situar el espacio en el que se encuentra el jugador simplemente se obtenga las coordenadas del jugador que tiene en ese instante, y así comprobar que no accede a objetos o a personajes de otras salas que no sean en la que está.

Entre las funciones que hemos enumerado, los más interesantes de explicar son:

Moverse de sala, porque consiste en sabiendo la ubicación del jugador y la dirección cardinal de movimiento (NORTE, SUR, ESTE U OESTE) la función cambiará la ubicación del jugador a la de la sala que se encuentra en la dirección especificada (si se puede).

Abrir puertas, que consiste en comprobar si en una dirección con respecto a la sala actual hay alguna puerta cerrada y en caso afirmativo y teniendo en el inventario una llave, abrirla.

Dormir. Debido a la naturaleza de nuestra trama, vamos a oscilar entre 2 “mundos” delimitados por nuestro mapa (el mundo inconsciente es esencialmente una matriz 7x7, y luego un mundo consciente), y si el jugador decide dormir cuando la primera coordenada de la ubicación sea mayor que 6 (está despierto), se duerme(es enviado a la coordenada de inicio del sueño {3,3}), en caso de estar ya dormido se avisará al usuario.

Despertarse en cambio se usa estando dormido (si la primera coordenada es menor que 7), y en el inventario se posee un despertador, que al usarse se extrae del inventario, y se traslada al jugador a la posición {7,0}. En caso de no cumplir un requisito se notificará al usuario.

El resto de funciones cumplen funciones que consideramos bastante obvias y poco resaltables, como hablar o examinar (que consisten en imprimir la cadena de caracteres correspondiente) o coger objetos del entorno o dejarlos, y también encender o apagar un objeto, que comprobando las condiciones previas correspondientes cambia el estado del objeto pasado.

**Objetos.c (TAD)**

**Descripción: e**n este TAD encontraremos todo lo relacionado a los objetos que usaremos en nuestro juego.

**Estructura: l**os objetos están compuestos de varios campos, de entre ellos 3 cadenas de caracteres y 4 booleanos:

**char\* name:** es el nombre del objeto.

**char\* descr:** es la descripción del objeto (que se obtiene si se examina el objeto).

**char\* dial:** es lo que dice el objeto si se intenta hablar con él (por muy surrealista que pueda parecer).

**BOOL cogible:** este campo es TRUE si el objeto se puede coger y FALSE si no se puede.

**BOOL activable:** este campo engloba ambos casos de abrir y encender un objeto, y no se puede cambiar durante el juego, esto denota TRUE si el objeto se puede abrir o encender (cambiar el BOOL activado de TRUE a FALSE y viceversa), y FALSE si no.

**BOOL ilum:** este campo (que tampoco se puede cambiar durante el juego) está en TRUE si el objeto tiene la capacidad de iluminar, y está en FALSE si no puede. La elección de este campo como TRUE implica el no poder abrir el objeto pero sí encenderlo, y si es FALSE implica el no poder encenderlo pero sí abrirlo.

**BOOL activado:** si este campo está en TRUE implica, dependiendo de si el objeto es iluminador o no, que el objeto está encendido/abierto, y estando en FALSE y con el mismo criterio, si el objeto está apagado/cerrado

**Funciones:** además de los campos obvios y necesarios como inicializar un objeto, destruirlo y settear campos, u obtenerlos. Hacemos uso de funciones como la comparación para ver si 2 objetos son iguales, y una llamada copy, que es práctica si queremos hacer una copia de los parámetros de un objeto sin tener que acceder a sus campos uno a uno.

**Interprete.c (TAD)**

**Descripción:** en este módulo disponemos de todas aquellas funciones que nos van a permitir comprobar y corregir los comandos que introducirá por teclado el usuario.

**Funciones:** en general a las funciones de este módulo reciben como argumento una cadena de caracteres y un fichero (*elim\_part* y *sinonimos*). Excepto la función *gramática* que recibe 4 cadenas de caracteres. Todas nuestras funciones modifican la cadena original y devuelven STATUS.

La función *mayus* transforma todas las mayúsculas en minúsculas.

La función *elim\_espac* elimina los espacios adicionales entre las palabras de la cadena.

La función *elim\_part* elimina partículas innecesarias como determinantes o conjunciones, estas partículas a eliminar están en un fichero de texto que se pasa como argumento.

La función *sinonimos* recibe una única palabra como argumento y busca alguna coincidencia con el fichero pasado como argumento, en caso afirmativo transforma esa palabra en la acción que corresponde. En caso contrario, escribe un mensaje de que no se ha encontrado la palabra.

La función *gramática* recibe como argumento una cadena con varias palabras la cual va guardando según corresponda en las variables verbo, directo o indirecto.

A la hora de utilizar estas funciones en un main debemos llamarlas en el orden como se han descrito anteriormente porque las hemos implementado de tal manera que van a saltar errores como exceso de argumentos, ya que si en un futuro queremos utilizarlas solo realicemos una función específica. Se trata de una ventaja de modular tanto el código.

**Ficheros:** contienen las palabras más típicas que puede introducir cualquier usuario, incluso se podría ampliar el contenido con palabras extranjeras para que nuestro juego lo pudieran usar en distintos países. En nuestro caso hemos tomado la decisión de diseño de pasar como argumento el fichero cada vez que llamemos a las funciones que lo requieren en vez de guardarlo en memoria que sería otra alternativa.

**Interfaz.c**

**Descripción:**

En este módulo crearemos las funciones que se encargan de separar la terminal en ventanas donde el programa escribe cosas diferentes.

**Estructura:**

**int** nfilas; Indica el numero de filas que ocupa la ventana

**int** ncolums; Indica el numero de columnas que ocupa la ventana

**int** start\_fila; Indica la primera fila de la ventana

**int** fila\_actual; Indica en que fila se encuentra el cursor en ese momento

**int** start\_colum; Indica la primera columna de la ventana

**int** colum\_actual; Indica la primera columna de la ventana

**int** last\_line; última linea

**Funciones:**

**Ventana\* ini\_ventana(int nfilas, int ncolums, int start\_fila, int start\_colum);**

Inicializa una ventana con el número de filas, columnas, y la posición inicial de fila y de columna

**void destroy\_ventana(Ventana\* window);**

Destruye y libera la memoria reservada para una ventana de la pantalla.

**STATUS limpiar\_ventana(Ventana\* window);**

Se encarga de imprimir espacios para simular que se limpia la ventana de texto.

**STATUS ventana\_visible(Ventana\* window);**

Comprueba si la ventana es correcta, es decir si está creada, si tiene un número negativo de filas por ejemplo y otras comprobaciones para ver si esta está lista para que se escriba en ella.

**STATUS colocar\_cursor\_write\_ventana(Ventana\* window, int fila, int colum);**

Coloca el cursor en la fila y la columna donde se quiere de una cierta ventana para posteriormente poder escribir mediante el uso de los comandos escape

**STATUS print\_caracter\_ventana(Ventana\* window, char caracter, int fila, int colum);**

Imprime un caracter en la ventana correspondiente haciendo uso de la funcion colocar\_cursor para poder decidir en qué fila y columna escribir.

**STATUS print\_cadena\_ventana(Ventana\* window, char\* cadena, int fila, int colum);**

Imprime el texto que se quiere escribir en la fila y columna que se desea.

**Guardar/cargar.c**

**Descripción:**

En este módulo guardaremos la partida que uno esté jugando, es decir, escribiremos el estado actual de las salas en un fichero, y si posteriormente deseamos cargar la partida simplemente leeremos los datos del fichero.

**Funciones:**

**Guardar**

En esta función vamos escribiendo en un fichero los datos en el formato del archivo inicio.txt, empezando por imprimir primero los objetos con todos sus parámetros, luego procedemos a escribir los personajes, y finalmente los espacios, todos ellos con cada uno de sus campos.

**Cargar**

Lee el fichero donde hemos escrito todos los datos de la partida anterior, pero como escribimos los datos previamente en el formato del fichero de inicio.txt, simplemente llamamos a la función del lector.

**Reglas.c**

**Descripción:**

Con este módulo nos encargaremos de ir comprobando reglas durante la partida, ya que si se cumplen ciertas condiciones se efectuarán unas ciertas consecuencias.

**Estructura:**

La estructura de las reglas radica principalmente en el formato de cada uno de los comandos del fichero, todos separados por “;”:

CONDICIONES

**Int** id\_pers (cada personaje tiene un id asociado)

**Int** primera coordenada del personaje

**Int** segunda coordenada del personaje

**Char** nombre del objeto que tiene el jugador

**Int** Si está activado el objeto del personaje (1 activado, 0 desactivado)

CONSECUENCIAS (sólo uno de estos bloques puede estar con valores, el resto tienen que estar con X para simplificar que sólo puede darse una consecuencia a la vez).

**Int** primera coordenada a donde transportamos al personaje

**Int** segunda coordenada a donde transportamos al personaje

**Int** (0 si vamos a insertar un objeto, 1 si lo extraemos)

**Int** id\_obj(cada objeto tiene un id asociado)

**Char** direccióngest en la que se abre una puerta con respecto a la sala actual del personaje

**Funciones:**

**Leer\_condiciones**

En esta función simplemente vamos guardando los comandos de un fichero en un array de condiciones, cada condición compuesta por varias palabras.

**Reglas**

En esta función cogemos las reglas y vamos interpretándolas, ya que si no se cumplen todas las condiciones iniciales no procede a efectuar las consecuencias.

**Free\_reglas**

En esta función nos encargamos de liberar el array de condiciones.

**Nota:** algunas dependencias se deducen de otras, por lo que sólo incluimos las flechas mínimas para entender cómo funcionan (por ejemplo, es obvio que personajes depende de objetos aunque no haya una unión directa entre ambas). Además el módulo del intérprete no lo incluimos en este esquema porque es independiente.