I2: Objetos y enemigos

INTRODUCCIÓN

Vamos a comenzar el desarrollo de la segunda iteración del proyecto (I2) añadiendo nueva funcionalidad al motor de juego conversacional que hemos iniciado. La Figura 1 ilustra los módulos del proyecto en los que se trabajará en esta nueva iteración a partir del material elaborado en la primera (I1). Los módulos obtenidos como resultado de la I1, y que se utilizarán y ampliarán en esta nueva entrega, se han representado en **rojo**; mientras los módulos que se desarrollarán en la segunda iteración se presentan en ocre.

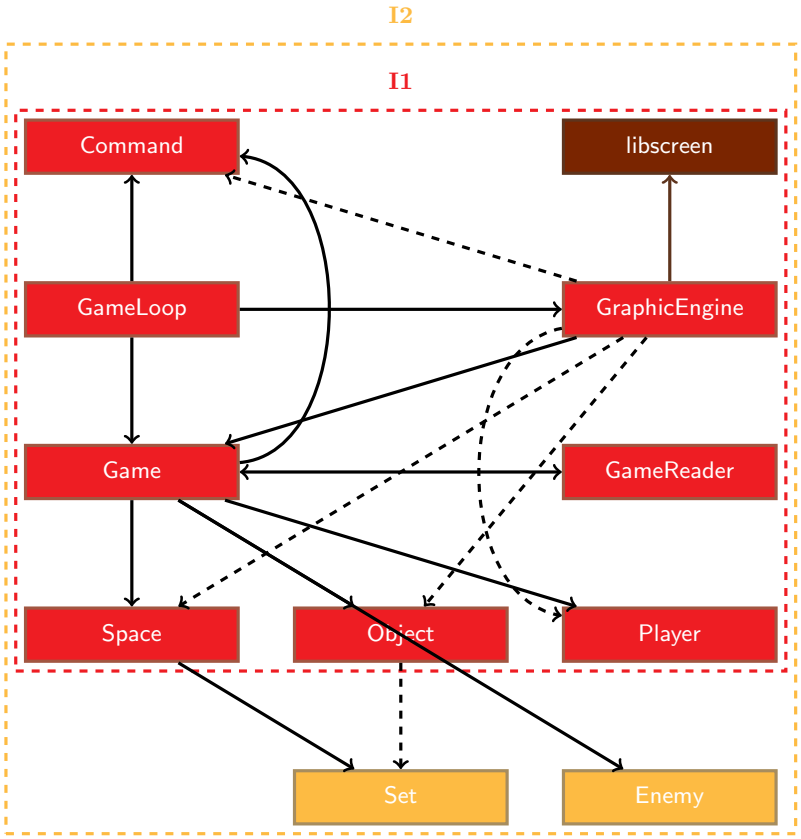

Figura 1: Módulos considerados en la segunda iteración (I2) del desarrollo del proyecto.

Los nuevos módulos son dos: Set (Conjunto) y Enemy (Enemigo). El módulo Set proporcionará la funcionalidad necesaria para manejar conjuntos de identificadores, como los que tienen el jugador, los objetos o los espacios. Un conjunto es una colección de elementos distintos (no repetidos) en la que no se considera el orden. El módulo Enemy es un módulo similar al módulo Player y permitirá introducir en el juego otros personajes (no jugadores) a los que enfrentarse.

El material de partida, resultado de I1, debería generar una aplicación que permitiera:

1. Cargar los espacios de juego desde un fichero de datos.
2. Gestionar todo lo necesario para la implementación del juego (espacios, jugador capaz de llevar un objeto, y posicionamiento en el mapa de un objeto y del jugador).

3. Soportar la interacción del usuario con el sistema, interpretando comandos para mover al jugador por el mapa (adelante y atrás), hacer que el jugador manipule un objeto en los espacios (cogerlo y dejarlo), y salir del programa.
4. Mover al jugador por los espacios, pudiendo coger y dejar el objeto, y haciendo cambiar el estado del juego.
5. Mostrar la posición en cada momento del jugador y de las casillas contiguas (norte y sur) a la que ocupa, indicando también la ubicación del objeto.
6. Liberar todos los recursos utilizados antes de terminar la ejecución del programa.

Como resultado de la I2, se espera que la aplicación siga cubriendo las funcionalidades de la I1, pero que con los nuevos requisitos además permita:

1. Cargar los objetos involucrados en el juego desde un fichero de datos, como se hace con los espacios.
2. Gestionar todos los datos necesarios para la implementación del juego con las nuevas características que se establezcan.
3. Mostrar la descripción gráfica (ASCII) de los espacios que la tengan, la posición de todos los objetos y el resto de información relevante del juego.

Para esta nueva iteración vamos a partir del mapa ampliado creado en la I1 y que se muestra en la Figura 2.

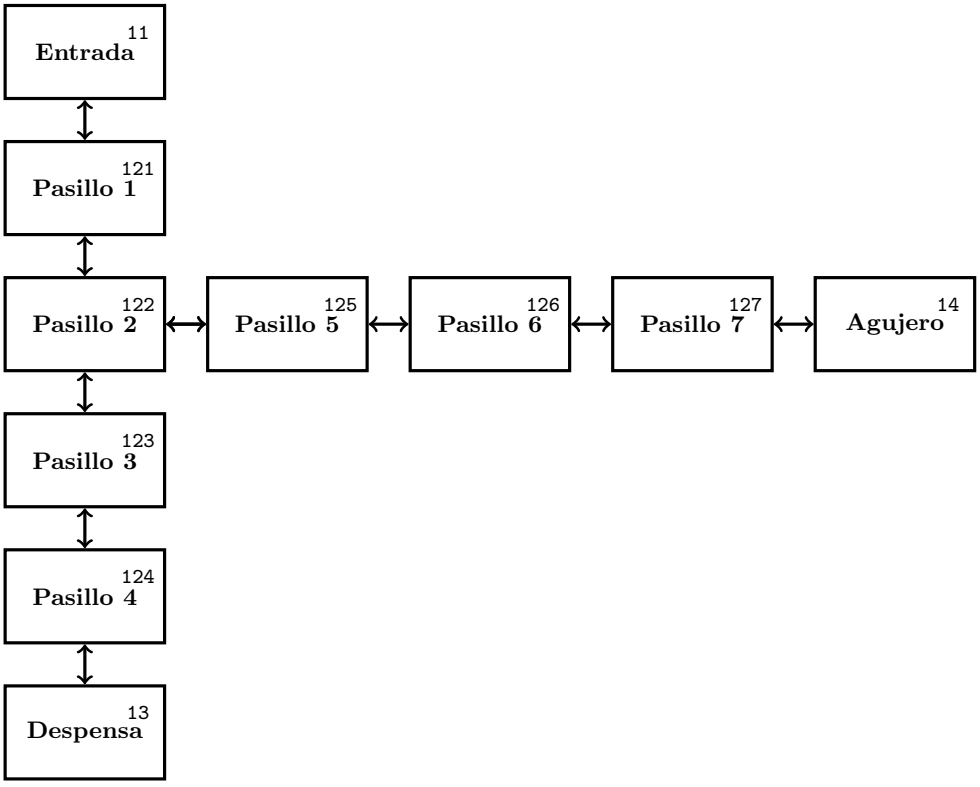

Figura 2: Mapa del hormiguero.

OBJETIVOS

Los objetivos de esta segunda iteración del proyecto son de dos tipos. Por un lado, profundizar en el empleo del entorno de programación de GNU (`gcc`, `gdb`, etc.), avanzar en el empleo de

técnicas y herramientas esenciales para el control de versiones, así como profundizar en el uso de bibliotecas e iniciarse en los fundamentos de su diseño. Por otro lado, practicar todo ello con el material disponible como resultado de la I1, modificando el mismo para mejorarlo y dotarlo de nuevas funcionalidades.

A continuación se especifican las tareas concretas a realizar en esta iteración.

Requisitos de gestión del proyecto

G1. Adaptar el diagrama de Gantt dado de forma que permita planificar el trabajo de cada equipo. La entrega se realizará durante la primera semana de la iteración.

G2. Entregar la versión final del cronograma, con los cambios realizados a lo largo de las semanas, para (en caso de necesidad) ajustar los tiempos a la realidad del proyecto.

Requisitos de compilación y entrega

C1. El código debe poderse compilar y enlazar de forma automatizada utilizando el fichero `Makefile` entregado. Es fundamental actualizar este fichero a medida que se van añadiendo nuevos ficheros y nuevas dependencias.

C2. Comprobar que no se producen *warnings* al compilar con la opción `-Wall -ansi -pedantic`.

Requisitos de documentación y estilo

D1. Documentar **todos** los ficheros proporcionados como código semilla, así como los nuevos que se generen. En concreto se debe intentar:

- Que todas las constantes, variables globales, enumeraciones públicas y estructuras tanto públicas como privadas estén documentadas.
- Que los ficheros fuente incluyan comentarios de cabecera con todos los campos requeridos.
- Que los prototipos de las funciones tanto públicas como privadas estén correctamente documentadas.
- Que las funciones tengan identificado un autor único.
- Que las variables locales de cada módulo o aquellas funciones que precisen explicación estén comentadas.

D2. Mantener un buen estilo de programación en el código entregado, en concreto:

- Que las variables y funciones tengan nombres que ayuden a comprender para qué se usan.
- Que el código esté bien indentado[1].
- Que el estilo sea homogéneo en todo el código[2].

Requisitos de funcionalidad

F1. Crear un módulo Set (conjunto) que integre la funcionalidad necesaria para el manejo de conjuntos. En particular, los conjuntos deberán implementarse como una estructura

[1] La indentación deberá ser homogénea. Todos los bloques de código pertenecientes a un mismo nivel deberán quedar con la misma indentación. Además, deberán usarse caracteres de tabulación o espacios (siempre el mismo número de espacios por nivel), pero nunca mezclar tabulación y espacios.

[2] Como mínimo debe cumplirse lo siguiente: que los nombres de las funciones comiencen con el nombre del módulo; que las variables, funciones, etc. sigan convención *camel case* o *snake case*, pero nunca mezcladas; que el estilo de codificación sea siempre el mismo (p.e. *K&R*, *Linux coding conventions*, etc.), pero nunca mezclar estilos.

de datos con dos campos, uno para almacenar un array de identificadores (`ids`), y otro para recordar el número de ellos en cada momento (`n_ids`). Este módulo debe facilitar al menos las funciones necesarias para crear y destruir conjuntos (`create` y `destroy`), añadir y eliminar valores (`add` y `del`) e imprimir el contenido de los mismos para su depuración (`print`). En este módulo se pueden añadir otras funciones necesarias para el manejo de conjuntos desde fuera del TAD.

F2. Modificar el módulo Space para que, utilizando un conjunto (mediante el módulo Set), permita que los espacios puedan contener varios objetos en su interior. Para ello debería sustituirse en la estructura de datos de Space el actual campo de objeto por otro campo `objects` que contenga el conjunto de los objetos en el espacio. Además de la estructura de datos de Space, deberán modificarse todas las primitivas que utilicen el campo sustituido para que sigan funcionando con el nuevo. Se añadirán, además, todas aquellas funciones adicionales que se consideren necesarias para manejar correctamente los objetos. Por ejemplo, funciones para añadir un objeto al espacio, para obtener los identificadores de objetos en dicho espacio, o para saber si el identificador de un objeto está en el mismo.

F3. Crear un módulo Enemy (Enemigo) siguiendo el ejemplo del módulo Player. En concreto, deberá implementarse como una estructura de datos con campos de identificación (`id`), nombre (`name`), localización (`location`) para almacenar el identificador del espacio donde se encuentra, y salud (`health`) para indicar el número de puntos de vida que posee en cada momento el enemigo. También en este caso, el módulo debería facilitar las funciones necesarias para crear y destruir enemigos (`create` y `destroy`), leer y cambiar los valores de sus campos (`get` y `set`) e imprimir el contenido de los mismos (`print`).

F4. Añadir al módulo Player un campo que indique la salud del jugador (`health`) para almacenar el número de puntos de vida de los que dispone en cada momento.

F5. Modificar los demás módulos existentes para que utilicen los módulos anteriores manteniendo la funcionalidad previa. Por ejemplo, en la estructura de datos de Game, sustituir el puntero a Object existente para guardar el único objeto que se permitía en la I1 por un array de punteros a objeto para almacenar todos los que se utilicen en el juego, como se hace con los espacios. Por otro lado, se debe añadir a la estructura de Game un puntero a un enemigo. Además, se deben utilizar las primitivas adecuadas de los nuevos módulos para la manipulación necesaria de datos desde el resto de módulos.

F6. Añadir al fichero de carga de datos (`hormiguero.dat`) aquellos datos relativos a los objetos que se utilizarán en el juego, siguiendo el modelo empleado para cargar los espacios, con el formato `#o:21|Grano|11`, donde se indica el identificador del objeto, su nombre y su posición inicial. El fichero debe incluir cuatro objetos.

F7. Crear una función en GameReader para cargar los objetos en el juego, siguiendo el modelo de la función utilizada para cargar los espacios. Modificar los módulos necesarios para que los objetos se carguen desde fichero, como se hace con los espacios.

F8. Añadir dos nuevos comandos (`left` o `l`, `right` o `r`) para podernos mover por el mapa en dirección oeste y este.

F9. Añadir un comando de ataque (`attack` o `a`) que permita a un jugador enfrentarse al enemigo si está en el mismo espacio. Vamos a suponer en este caso que los niveles de jugador y enemigo son equivalentes y por tanto cuando se ejecute este comando, se generará un número aleatorio entre 0 y 9. Si el número está entre 0 y 4, gana el enemigo y por tanto el jugador pierde un punto de vida. Si el número generado está entre 5 y 9, gana el jugador, y en este caso es el enemigo el que pierde un punto de vida. Si el enemigo tiene 0 puntos de salud, está muerto y no podemos atacarle más. Es importante asegurarse de que el número generado es realmente aleatorio y no se genera siempre el mismo número o la misma secuencia de números.

F10. Añadir a la función `game_is_over` una comprobación de forma que, cuando el jugador

se quede sin puntos de vida (es decir, `health` valga 0), se termine el juego de manera automática.

F11. Modificar el comando `take` o `t` que permite al jugador coger un objeto de un espacio, de modo que pueda indicarse el objeto que se pretende coger de entre los que haya disponibles en el espacio concreto. En particular, este comando deberá incluir el identificador establecido para el objeto que se quiere coger precedido por la letra `O`, por ejemplo: `take O21`.

F12. Modificar una vez más el módulo Space para incluir una descripción gráfica (ASCII) del espacio de 5×9 caracteres. Para ello deberá incluirse un campo `gdesc` que fuera un array de cinco strings, cada uno con espacio para nueve caracteres. De este modo, por ejemplo, la descripción gráfica del espacio con identificador 122 de la Figura 4 estaría formada por los strings:

También en este caso deberán adaptarse las funciones existentes para manejar el nuevo campo (`create`, `destroy` y `print`) y crear las nuevas funciones de manipulación necesarias (`set` y `get`).

F13. Añadir al menos cuatro descripciones gráficas en el fichero de datos, como por ejemplo las indicadas en la Figura 3, en cuatro espacios diferentes, siguiendo el modelo dado en la Figura 4. Modificar la función de carga de espacios para adaptarse a los cambios introducidos en el formato. La línea del fichero de datos correspondiente al espacio de ejemplo del Requisito F12 sería:

```
1 #s:122|Pasillo2|121|125|123|-1|  ___    _|_/   \_/ |     o 0 |mo'_ 0o8o|__/ \___/|
```

Figura 3: Ejemplos de representación gráfica de algunos espacios.

F14. Modificar la función de visualización del estado del juego para mostrar: (a) la descripción gráfica (ASCII) de cada espacio; (b) flechas que indiquen la dirección hacia la que puede moverse el jugador desde el espacio actual; (c) el espacio donde está cada objeto; (d) varios objetos en los espacios visualizados; (e) la representación del enemigo en el espacio en el que se encuentre (en este caso una araña cuya representación, por ejemplo, podría ser «`/\oo/\`»); (f) los objetos que porta el jugador; (g) los puntos de vida del jugador y del enemigo; y (h) el último comando ejecutado seguido del resultado de su ejecución (`OK` o `ERROR`). Hay que tener en cuenta que puede ser necesario redimensionar alguna ventana de la interfaz de usuario (se muestra un ejemplo en la Figura 4).

Requisitos de pruebas

P1. Adaptar el fichero de pruebas `space_test.c` dado acorde a las actualizaciones del módulo Space.

P2. Realizar pruebas unitarias para el módulo Set (`set_test.c`).

Figura 4: Ejemplo de visualización del juego en un instante dado.

P3. Realizar pruebas unitarias para el módulo Enemy (`enemy_test.c`).

CRITERIOS DE CORRECCIÓN

La puntuación final de esta práctica forma parte de la nota final en el porcentaje establecido al principio del curso para la I2. En particular, la calificación de este entregable se calculará siguiendo la rúbrica de la Tabla 1, en la que la segunda columna muestra la puntuación de cada requisito, y las tres últimas columnas incluyen una sugerencia de con qué nivel de completitud hay que abordar cada requisito para alcanzar la calificación indicada.

Tabla 1: Tabla de rúbrica para la segunda iteración (I2).

Objetivo	Puntuación	Aprobado	Notable	Sobresaliente
G1	0,50			
G2	0,50			
C1	1,50			
C2	0,50			
D1	1,00			
D2	1,00			
F1	0,50			
F2	0,30			
F3	0,50			
F4	0,10			
F5	0,20			
F6	0,20			
F7	0,30			
F8	0,30			
F9	0,50			
F10	0,10			
F11	0,30			
F12	0,30			
F13	0,10			
F14	0,30			
P1	0,30			
P2	0,35			
P3	0,35			

Además, no se podrá aprobar la iteración en los siguientes casos extremos:

- No se ha completado el diagrama de Gantt.
- El código entregado no compila.
- El código entregado carece de cualquier documentación, o de un estilo de codificación razonable.
- No se ha implementado ningún requisito de funcionalidad.
- No se ha implementado ningún requisito de pruebas.