I3: Prototipo de juego conversacional

Introducción

En la tercera iteración (I3) se continúa con el desarrollo del proyecto y de las habilidades y los conceptos necesarios para ello. La Figura 1 ilustra los módulos del proyecto en los que se trabajará en la I3, partiendo del material elaborado en las anteriores iteraciones (I1 e I2). En esta ocasión se completará el desarrollo de los módulos fundamentales del sistema.

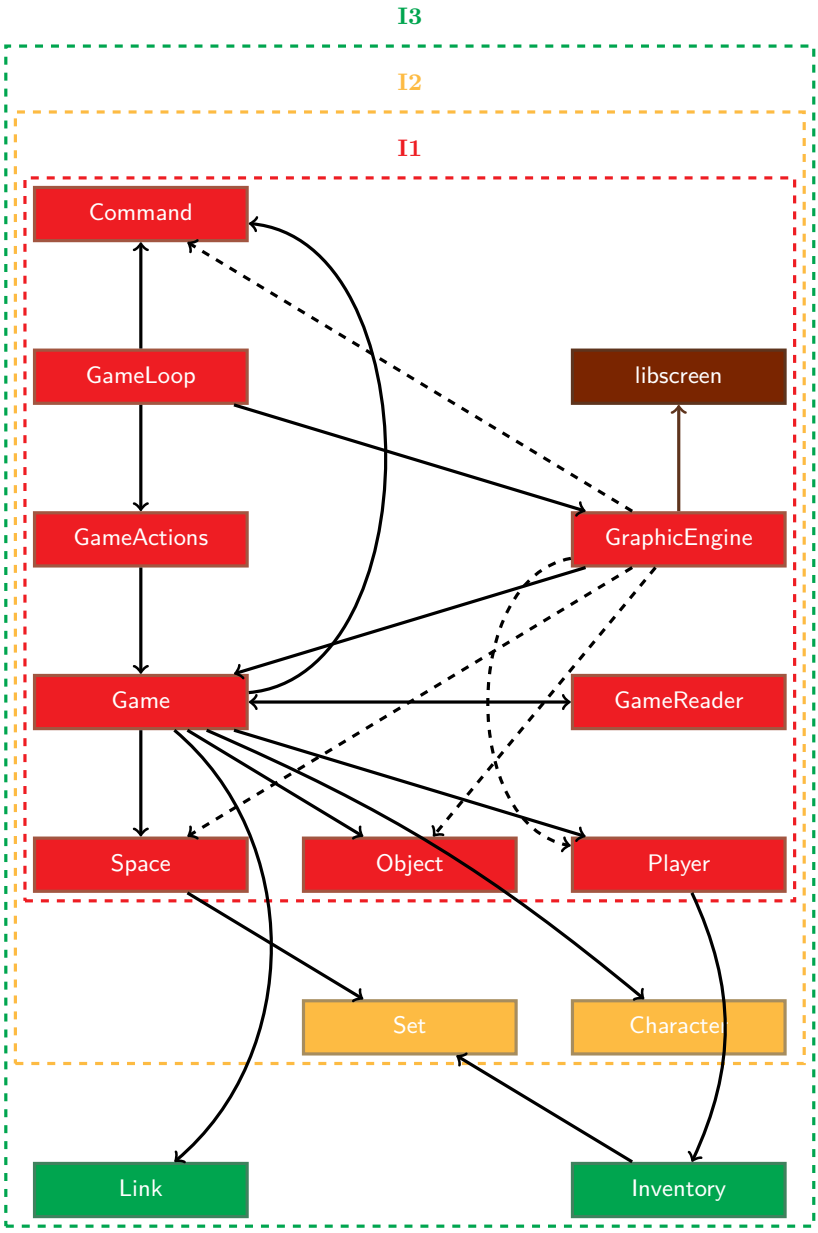

Figura 1: Módulos considerados en la tercera iteración (I3) del desarrollo del proyecto.

Los módulos obtenidos como resultado de la I1 se han representado en **rojo**. En **amarillo** se presentan los módulos desarrollados en la I2. Finalmente, en **verde** se muestran los módulos que

se desarrollarán desde cero en la I3, en la que también se ampliarán y utilizarán algunos de los módulos de colores **rojo** y **amarillo**. Los nuevos módulos son dos: Inventory (Inventario) y Link (Enlace). El módulo Inventory aportará la funcionalidad necesaria para dotar a cada jugador de un inventario (o mochila) con el que llevar el conjunto de objetos que porte. El módulo Link proporcionará la funcionalidad para dotar a las conexiones entre espacios de capacidades nuevas, como por ejemplo estar abiertas o cerradas, o necesitar de un objeto (llave) para poderlas usar, como pasa con algunas puertas.

El material de partida, resultado de I1 e I2, debería generar una aplicación que permitiera:

1. Cargar espacios y objetos desde ficheros de datos.
2. Gestionar todo lo necesario para la implementación de juegos conversacionales básicos (espacios, objetos, jugador capaz de llevar un objeto, posicionamiento en el mapa de objetos, del jugador y de distintos personajes, así como movimiento entre espacios enlazados entre sí).
3. Soportar la interacción del usuario con el sistema, interpretando comandos para mover al jugador, manipular objetos (el jugador solo puede portar un objeto en I2, pero puede haber varios de ellos en cada espacio), luchar contra un enemigo, hablar con un amigo y salir del programa.
4. Mostrar la posición del jugador en cada momento y sus puntos de vida, además de la ubicación de los objetos y de los personajes y una descripción gráfica en los espacios.
5. Liberar todos los recursos utilizados antes de terminar la ejecución del programa.

Como resultado de la I3 se espera:

1. Una aplicación que permita gestionar juegos con mapas formados por espacios unidos con enlaces que pueden estar abiertos o cerrados, objetos con descripciones textuales que pueden consultarse, así como espacios que se van descubriendo a media que el jugador avanza, y jugadores que pueden llevar más de un objeto.
2. Se añadirá además la carga de los jugadores, de personajes y de los enlaces entre espacios desde ficheros de datos, como se hace con los espacios y los objetos.
3. Se gestionarán todos los datos necesarios (espacios, enlaces, objetos, jugadores y personajes) para la implementación de las aplicaciones con las características indicadas.
4. Soportará la interacción del usuario para llevar varios objetos, examinar objetos en busca de pistas, y moverse entre espacios conectados mediante enlaces en cuatro direcciones (norte, sur, este y oeste).
5. Soportará también un modo de juego multijugador por turnos, donde cada jugador podrá ir avanzando y viendo la información de su partida.
6. Al mismo tiempo, que la misma aplicación siga cubriendo las funcionalidades desarrolladas anteriormente, pero utilizando espacios conectados mediante enlaces abiertos, comandos para moverse en las direcciones que estén disponibles de las cuatro existentes (norte, sur, este y oeste) como reemplazo a los movimientos ya implementados (adelante, atrás, derecha e izquierda), además de jugadores capaces de llevar varios objetos y un comando para dejar un objeto concreto del inventario del usuario.
7. Por último, existirá la posibilidad de generar un fichero con el registro de comandos ejecutados y los correspondientes resultados obtenidos.

Objetivos

Los objetivos de esta tercera iteración del proyecto (I3) son de dos tipos como en iteraciones anteriores. Por un lado, dirigidos al desarrollo de conocimiento y habilidades, en concreto a iniciarse en la automatización de documentación (Doxygen) y al trabajo en equipo. Se debe revisar para todo ello los materiales disponibles en Moodle. Por otro lado, poner en práctica todo lo aprendido, en particular modificando el material obtenido de la I1 y la I2 para mejorarlo y dotarlo de nuevas funcionalidades, además de implementar aplicaciones nuevas que prueben y demuestren las funcionalidades incorporadas.

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

Requisitos de gestión del proyecto

G1. Planificar la iteración del proyecto (tareas, recursos y tiempo), documentados mediante un diagrama de Gantt adecuado, ajustándolo si fuera necesario según transcurre el proyecto. La primera versión debe entregarse la primera semana de trabajo en la iteración.

G2. Realizar reuniones, documentadas con actas que incluyan compromisos de los miembros del equipo, asignación de tareas, y plazos y condiciones de las entregas. Los cambios de los acuerdos realizados así como de planificación deben reflejarse en dichas actas.

Requisitos de compilación y entrega

C1. El código, incluidas las pruebas, 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 hay *warnings* al compilar con las opciones `-Wall -ansi -pedantic`.

C3. El `Makefile` entregado debe permitir, mediante una tarea específica, la generación de la documentación técnica del proyecto utilizando Doxygen.

C4. Modificar el `Makefile` del proyecto para automatizar la compilación y el enlazado del conjunto, de manera que los ficheros se encuentren organizados en subdirectorios de la siguiente manera:

- Ficheros con código fuente (`.c`) en subdirectorio «`./src`».
- Ficheros de cabecera (`.h`) en subdirectorio «`./include`».
- Ficheros objeto (`.o`) en subdirectorio «`./obj`».
- Ficheros de librerías (`.a`) en subdirectorio «`./lib`».
- Documentación (generada con Doxygen) en subdirectorio «`./doc`».

Requisitos de documentación y estilo

D1. Documentar todos los ficheros del proyecto. 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. Documentar el proyecto utilizando **Doxygen**. Primero, incluyendo etiquetas adecuadas en todos los ficheros fuentes, en concreto: (a) en los comentarios de cabecera de fichero de todos los `.h` y los `.c`; (b) en los comentarios de cabecera de prototipos de funciones de los `.h`, así como de los prototipos de funciones privadas de los `.c`; (c) en los comentarios de tipos enumerados y estructuras de datos de los `.h` y los `.c`. Después, generando la documentación con ayuda de la utilidad, por lo menos en formato HTML.

D3. 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 Inventory (Inventario) que incorpore la funcionalidad necesaria para dotar a cada jugador de un contenedor (mochila) donde llevar el identificador de varios objetos, de forma parecida a como lo hace el módulo Space para los objetos de los espacios. En particular, los inventarios deberán implementarse como una estructura de datos con dos campos, un campo `objs` para almacenar el conjunto de identificadores (utilizando el módulo Set desarrollado en la I2), y otro campo `max_objs` para establecer el número máximo de objetos que el usuario puede llegar a portar (este número máximo es independiente del tamaño máximo definido en la implementación de Set, ya que representa el tamaño de la mochila). Además, como otras veces, deberán implementarse las funciones necesarias para crear y destruir el inventario (`create` y `destroy`), cambiar los valores de sus campos (`set`, `get`, `add`, `del`...) e imprimir el contenido de los mismos para su depuración (`print`).

F2. Modificar el módulo Player para que, utilizando un Inventory, permita a los jugadores llevar varios objetos consigo. Para ello deberá sustituirse, en la estructura de datos de Player, el identificador de objeto portado por un campo `backpack`. Deberán modificarse todas las primitivas que utilicen el antiguo campo para que sigan funcionando con el nuevo, así como añadir funciones adicionales, en caso de necesidad, para manejar correctamente la mochila. Por ejemplo, serán útiles funciones para añadir un elemento a la mochila del jugador, para obtener los identificadores de objetos en ella, o para saber si el identificador de un objeto está en la misma.

F3. Modificar el resto de módulos del proyecto que utilizan funciones del módulo Player relacionadas con los objetos que porta.

F4. Crear un módulo Link (Enlace), siguiendo también el modelo de Space, que proporcione la funcionalidad necesaria para el manejo de enlaces entre espacios. En concreto, los enlaces deberán implementarse como una estructura de datos con varios campos: un identificador único `id` para cada enlace, un nombre `name`, dos campos con identificadores para indicar el espacio de origen de la conexión (`origin`) y el espacio de destino (`destination`), un campo para indicar la dirección del enlace (`direction`, de tipo `Direction`, que tomará valores norte, sur, este, oeste o desconocido) y un campo booleano `open` para determinar el estado

[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.

del enlace (abierto o cerrado). Como siempre, el módulo debería facilitar las funciones necesarias para crear y destruir enlaces (`create` y `destroy`), cambiar los valores de sus campos (`set` y `get`) e imprimir el contenido de los mismos para su depuración (`print`). Como consecuencia de este cambio, desaparecen de Space los campos `north`, `south`, `east` y `west`, así como todas las funciones del módulo Space relacionadas con ellas.

F5. Añadir los enlaces a la estructura del juego utilizando un array de enlaces. Modificar el módulo Game para que, en lugar de utilizar el módulo Space como antes para moverse por el mapa, utilice la información contenida en el módulo Link. Para ello, se aconseja crear dos nuevas funciones:

- `game_get_connection`, que reciba igualmente el juego, el identificador del espacio actual y la dirección del enlace, y devuelva el identificador del espacio destino.
- `game_connection_is_open`, que reciba el juego, el identificador del espacio actual y la dirección del enlace, y devuelva si el enlace está abierto o cerrado.

F6. Crear funciones en GameReader para cargar la información de los jugadores, los personajes y los enlaces en el juego, similares a las utilizadas para cargar los espacios y los objetos, y modificar el módulo Game para que obtenga toda la información de estas EdD desde el fichero. Además, ajustar la función de lectura de los espacios, que ya no deben cargar los identificadores de otros espacios conectados desde sus líneas en el fichero de datos. Las cadenas del fichero de datos tendrán el siguiente formato:

(a) Jugadores: `#p:1|ant|mO^|11|5|3|`, donde se indica el identificador del jugador, el nombre, su descripción gráfica, su posición, sus puntos de vida y el número máximo de objetos que puede llevar (tamaño de la mochila).

(b) Personajes: `#c:2|spider|/\oo/\|123|5|0|I won't talk to you!|`, donde se indica el identificador del personaje, el nombre, su descripción gráfica, su posición, sus puntos de vida, si es amistoso (`1`) o no (`0`) y el mensaje de chat en caso de ser un amigo.

(c) Enlaces: `#l:31|Entry|11|121|1|1|`, que contiene el identificador del enlace, su nombre, el identificador del espacio de origen, el identificador del espacio de destino, la dirección del enlace (en función de la enumeración `Direction`) y si el enlace está abierto(`1`) o no(`0`).

(d) Espacios: `#s:11|Entry| _ | ___ | mo'_| @ _ | __/ __|`, que contiene el identificador del enlace, su nombre y la descripción gráfica.

F7. Modificar el fichero de datos, de forma que se cree un enlace unidireccional con origen el agujero (espacio con identificador `14`) y destino la despensa (espacio con identificador `13`). Además, en el fichero de datos el enlace con origen el espacio donde está el enemigo y dirección sur aparecerá como cerrado. El mapa resultante se muestra en la Figura 2.

F8. Añadir un nuevo comando para moverse por los espacios siguiendo los enlaces de forma genérica, de forma que se especifique el comando seguido de la dirección del movimiento, de manera parecida a como se hacía en I2 para coger un determinado objeto. Por ejemplo, `move north` o `move west` (o de forma compacta `m n` o `m w`). Debe ser eficiente (evitando repetir código) y se deben eliminar los comandos previos `next`, `back`, `right` y `left`.

F9. Añadir otro comando para examinar objetos, obteniendo la descripción de los mismos de un nuevo campo `description` en la estructura de Object. Para ello deberá indicarse el objeto que se quiere examinar, por ejemplo `inspect grain` (o de forma compacta `i grain`). Se podrán examinar tanto objetos que estén en el espacio donde está el jugador como los que este tenga en su mochila.

F10. Modificar el comandos `drop` para que use el nombre del objeto para determinar qué objeto debe dejar. Así, ahora se invocará como `drop grain`.

F11. Modificar Game de forma que permita más de un jugador en el juego. Para ello se definirá un array de jugadores y un nuevo campo `turn` que establezca el turno y se actualice acorde al número de jugadores. Se deberán modificar todas aquellas funciones relativas a

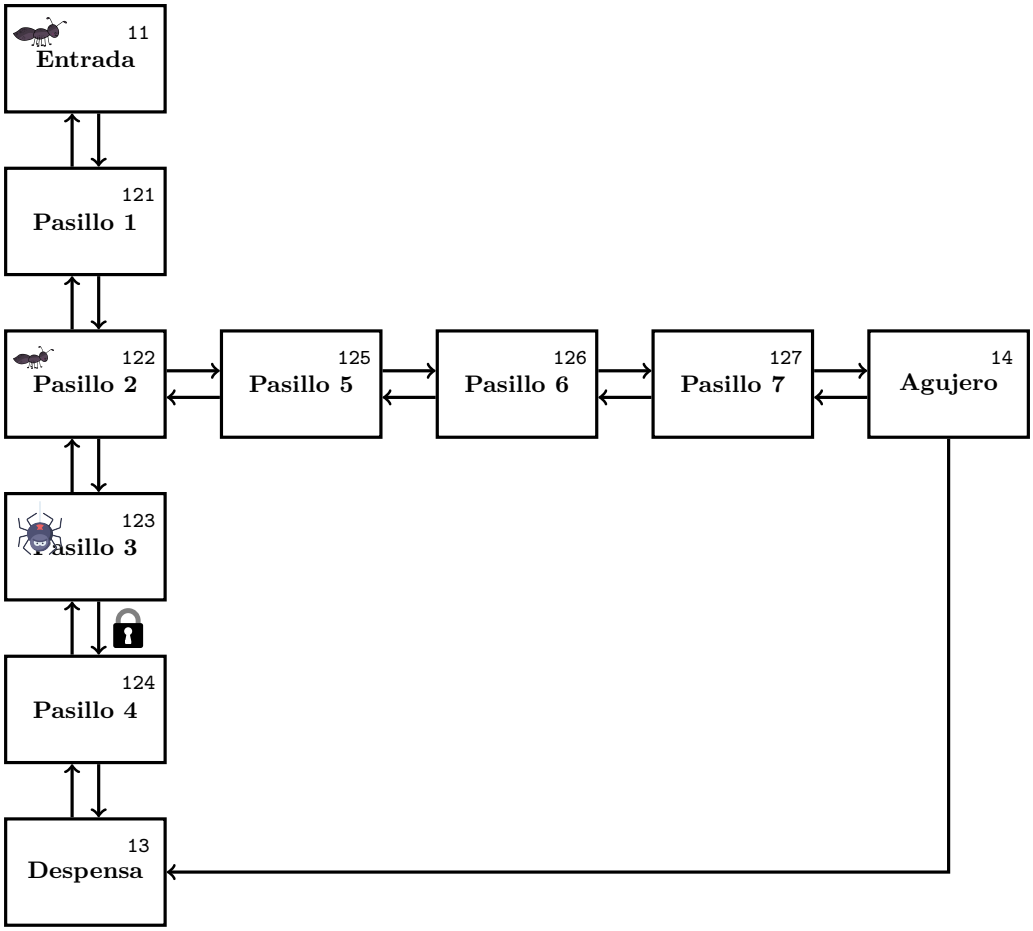

Figura 2: Mapa del hormiguero.

un jugador para que actúen sobre el jugador activo en cada momento.

F12. Añadir un campo booleano `discovered` en Space que indique si el espacio ha sido ya descubierto por un jugador o no. Este campo, que se inicializará a `FALSE` salvo para el espacio donde comienzan el juego los jugadores que se inicializará a `TRUE`, y se activará (`TRUE`) cuando alguno de los jugadores llegue por primera vez a un nuevo espacio. Solo para aquellos espacios ya visitados se mostrará información sobre sus objetos, sus personajes y su descripción gráfica.

F13. Hacer que la visualización del estado del juego muestre los enlaces y los espacios destino de los mismos, tal y como se muestra en la Figura 3. Así mismo, deberá gestionar que se muestre solo la información de aquellos espacios que ya han sido descubiertos. Por último, nótese que se realiza una visualización diferente en cada turno, según el jugador que esté activo y que será el centro del juego en ese momento. Para almacenar la información a mostrar de cada jugador, se sugiere definir un array que almacene en cada posición una estructura con la información «gráfica» a mostrar.

F14. Modificar el programa principal del juego (`game_loop`) para que, además de seguir funcionando como lo hacía antes, permita generar un fichero de registro (`LOG`) con trazas de la ejecución. En particular, el fichero `LOG` registrará una línea por cada comando ejecutado, en la que se indicará el comando y el resultado de su ejecución (`OK` o `ERROR`). De este modo, si el comando fuera `take grain` y no se consiguiera coger el objeto pretendido, en el fichero de `LOG` se escribiría la línea `take grain: ERROR`, mientras que si el objeto se hubiera cogido sin problema en el fichero se escribiría `take grain: OK`. Para indicar al programa que se

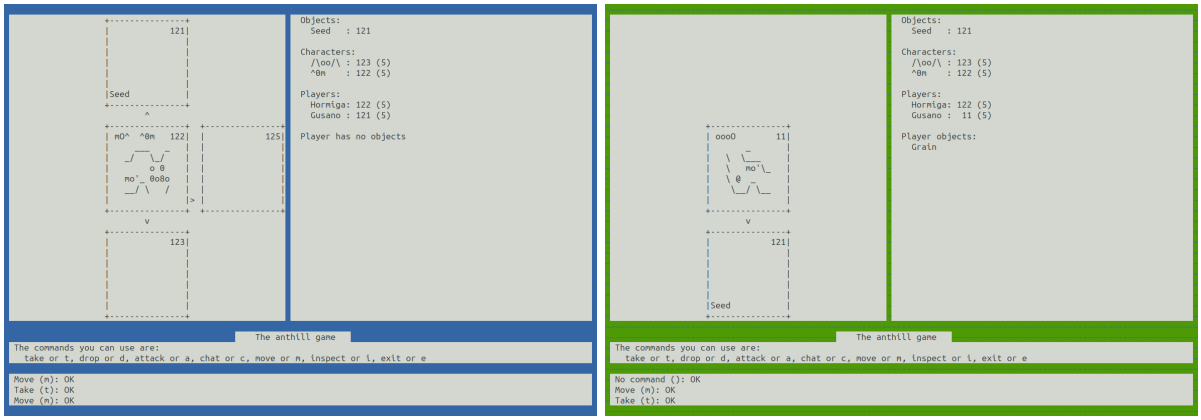

Figura 3: Ejemplo de visualización del juego en un instante dado para el jugador 1 a la izquierda y el jugador 2 a la derecha.

desea generar un fichero de `LOG`, al invocarlo se pasara como últimos argumentos `-l` (guion más la letra ele minúscula) seguido del nombre deseado para el fichero de `LOG`. De esta manera, si para invocar al programa sin `LOG` la línea de comandos es

```
$ ./executable dataFile
```

donde `fchdatos` es el fichero de datos a cargar, entonces para invocar al programa especificando que se genere el `LOG` la línea de comandos será

```
$ ./executable dataFile -l logFile
```

donde `logFile` es el nombre del fichero de `LOG`. Para implementar esta nueva funcionalidad el programa debería procesar convenientemente los argumentos de entrada, y en el caso de que se haya indicado que se quiere generar el `LOG`: (a) antes de iniciar el bucle del juego, abrir para escritura el fichero de `LOG` con el nombre indicado; (b) obtener en cada iteración el resultado de la ejecución del comando y escribir este y su resultado en el fichero de `LOG` como se ha dicho anteriormente; y finalmente (c) cerrar el fichero de `LOG` antes de terminar el programa al salir del bucle de juego.

Requisitos de pruebas

P1. Realizar o completar pruebas unitarias de los siguientes módulos, automatizadas en la medida de lo posible:

- Space
- Object
- Player
- Set
- Character
- Link
- Inventory

Se deben realizar al menos dos pruebas para cada función.

P2. Definir pruebas de integración para el proyecto completo. Para ello, se deberán escribir ficheros con los comandos a probar. Por ejemplo, el fichero `game1.cmd` se define como:

```
move north
move north
move west
take grain
```

```
move west
drop grain
inspect grain
...
exit
```

Para recibir las instrucciones del archivo no se debe implementar la lectura del fichero desde el código. En su lugar, se utilizará la redirección de la entrada estándar de la terminal:

```
$ ./anthill anthill.dat -l output.log < game1.cmd
```

El fichero de `LOG` generado será la salida a comprobar.
Se deben considerar pruebas de casos extremos como ficheros de datos vacíos o con líneas erróneas, además de pruebas con un fichero de datos correctos, probando todos los comandos definidos hasta el momento en distintos instantes del juego.

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 I3. 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 obtener la calificación indicada.

Tabla 1: Rúbrica para la tercera iteración (I3).

Objetivo	Puntuación	Aprobado	Notable	Sobresaliente
G1	0,75			
G2	0,75			
C1	0,50			
C2	0,30			
C3	0,20			
C4	0,50			
D1	0,50			
D2	0,50			
D3	0,50			
F1	0,40			
F2	0,10			
F3	0,10			
F4	0,40			
F5	0,20			
F6	0,40			
F7	0,10			
F8	0,20			
F9	0,20			
F10	0,20			
F11	0,40			
F12	0,40			
F13	0,50			
F14	0,40			
P1	1,00			
P2	0,50			

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.

Otras consideraciones

- Se penalizarán fuertemente los errores en tiempo de ejecución o cualquier otro error indicado por Valgrind.
- No se puntuarán aquellos requisitos de funcionalidad que no se puedan probar (o bien mediante un programa de prueba o bien mediante el propio juego).