I4: Juego conversacional

Introducción

En esta cuarta iteración (I4) se debe completar el desarrollo del proyecto, empleando para ello conceptos, habilidades y herramientas en los que se ha trabajado durante el curso. En esta iteración se completarán las funcionalidades básicas del sistema para soportar aventuras conversacionales multijugador, y se podrán añadir otras que se consideren adecuadas para el juego original que cada equipo deberá diseñar e implementar sobre el sistema.

La Figura 1 ilustra los módulos del proyecto en los que se trabajará en la I4, partiendo del material elaborado en las anteriores iteraciones (I1, I2 e I3). En esta ocasión se completará el desarrollo de los módulos fundamentales del sistema y se podrán añadir otros con funcionalidades especiales convenientes para el juego particular que se implemente.

Los módulos obtenidos como resultado de la I1 se han representado en **rojo**. En **amarillo** se presentan los módulos desarrollados en la I2. En **verde** se muestran los módulos que se desarrollaron en la I3. Por último, en **azul** se representan los nuevos módulos GameManagement y GameRules, en la que además deberán ampliarse y emplearse algunos de los módulos de anteriores iteraciones (representados en **rojo**, **amarillo** y **verde**).

Los nuevos módulos, incluidos en la figura, son dos. El primero, GameManagement (Gestor de Juego), tiene la finalidad de dotar a la plataforma de la capacidad de gestionar partidas. El segundo, GameRules (Reglas de Juego), está dirigido a dotar al juego de reglas que cambien automáticamente su estado, además de los cambios que la interacción del jugador con el turno provoque. De todos estos módulos se hablará con más detalle a lo largo de este documento.

El material de partida, resultado de I1, I2 e I3, debería generar una aplicación que permita:

1. Cargar espacios, enlaces, objetos, personajes y jugadores desde un fichero de datos.
2. Gestionar todo lo necesario para la implementación de juegos conversacionales básicos utilizando todos los elementos mencionados en el punto anterior, como hemos hecho con el Hormiguero.
3. Soportar la interacción del usuario con el sistema, interpretando comandos para mover a los jugadores, manipular varios objetos, luchar contra enemigos, hablar con amigos, inspeccionar objetos y salir del programa.
4. Mostrar el estado del juego en cada momento: posición de los jugadores y de los personajes, sus puntos de vida, objetos en el inventario del jugador en juego, así como descripciones gráficas (ASCII) del espacio en el que se encuentra y de los contiguos al mismo y la ubicación de los objetos en el mapa del juego de todos aquellos espacios ya visitados.
5. Ofrecer una experiencia multijugador básica, con turnos rotatorios.
6. Liberar todos los recursos utilizados antes de terminar la ejecución del programa.
7. La generación de un fichero con el registro de comandos ejecutados y los correspondientes resultados obtenidos.

Como resultado de la I4 se espera:

1. Una aplicación que permita implementar una aventura conversacional multijugador utilizando el sistema desarrollado a lo largo del curso y las extensiones incorporadas en esta última iteración. El programa deberá utilizar todas las características que debería proporcionar el material de partida (resultado de las anteriores iteraciones) convenientemente corregido, mejorado y ampliado.

2. Una aventura que contenga, al menos, 10 espacios, 20 objetos, 4 personajes y 2 jugadores, con su correspondiente fichero de datos (incluyendo espacios, enlaces, objetos, personajes, jugadores, etc.).
3. La corrección, mejora y ampliación de las funcionalidades de la plataforma de partida de acuerdo con los requerimientos que se indiquen más adelante y las necesidades particulares de la aplicación y la aventura del punto anterior.
4. Documentación de usuario que incluya la información necesaria para utilizar el juego, un mapa, indicaciones para ir del principio al final del juego de forma que se aprecien todas las características y funcionalidades implementadas, y un fichero de comandos para seguir dicho camino.
5. Programas de prueba de los módulos (`*_test.c`) y documentación de las mismas (documentos de diseño e informes de pruebas), automatizadas en la medida de lo posible con programas de prueba según el modelo propuesto para Space (`space_test.c`).
6. Un fichero `Makefile` que automatice la gestión del proyecto: (a) parámetros de compilación exigentes (`-Wall -ansi -pedantic`) y de depuración (`-g`); (b) compilación/enlazado de cada módulo así como de los programas de prueba realizados (`*_test`); (c) generación automática de la documentación del proyecto; y (d) organización en carpetas de los distintos tipos de ficheros.
7. Documentación técnica del proyecto en formato HTML generada con Doxygen.
8. Gestión del proyecto a lo largo de la I4, con diagramas de Gantt y actas de las reuniones celebradas.

Objetivos

El objetivo de esta cuarta iteración del proyecto (I4) es poner en práctica todo lo aprendido durante el curso sobre trabajo en equipo, gestión de proyectos, uso y diseño de bibliotecas, programación, pruebas, depuración, documentación, etc.

Se debe revisar para todo ello el módulo «Bash scripting» del curso SPOC así como los materiales disponibles en Moodle.

Las mejoras y modificaciones concretas requeridas se especifican a continuación.

Requisitos de gestión del proyecto

G1. Planificar la iteración del proyecto (tareas, recursos y tiempo), documentados mediante un diagrama de Gantt adecuado, ajustándolo en cada reunión. La primera versión debe entregarse la primera semana de trabajo en la iteración.

G2. Realizar reuniones semanales, 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`. Además en esta iteración se deberá compilar con `-g` para facilitar la depuración del código.

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].

[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.

Requisitos de funcionalidad

F1. Modificar el módulo Character para que permita que los personajes puedan seguir a un jugador. Para ello debería añadirse un nuevo campo `following` de tipo booleano a la estructura de datos correspondiente e implementarse las funciones necesarias para manipular sus valores (como `set` y `get`) e imprimir el contenido de este para su depuración (`print`).

F2. Añadir un nuevo comando para que un jugador reclute a un personaje, y que este comience a seguirle (`recruit <chr>`, donde `<chr>` es el nombre del personaje que sigue al jugador). Se debe tener en cuenta que solo los personajes amigos pueden seguir al jugador y que podría seguirle más de un personaje.

F3. Añadir un nuevo comando para que un personaje deje de seguir al jugador con el turno (`abandon <chr>`, donde `<chr>` es el nombre del personaje al que abandona el jugador).

F4. Si a un jugador le siguen personajes amigos, al ejecutar el comando `attack <chr>` el jugador atacará con más fuerza al enemigo, de forma que, en caso de que la tirada indique que gana el jugador, se le quitarán al enemigo tantos puntos de vida como personajes le estén atacando (por ejemplo, si al jugador le sigue un amigo y gana un ataque, quitará al enemigo 2 puntos). Sin embargo, si es el enemigo el que gana, quitará solo 1 punto de vida a uno de los personajes que le han atacado, que se elegirá de manera aleatoria entre el jugador y sus *followers*.

F5. Modificar todos los comandos y funciones del juego que pueden verse afectados por los cambios anteriores. Por ejemplo el módulo Space deberá ser capaz de almacenar ahora más de un personaje por lo que se debe utilizar el TAD Set como se hace para los objetos. Por otro lado, el comando `move` se deberá modificar de forma que los personajes que siguen a un jugador se muevan con él por los espacios del juego. También se deberán crear en Game todas las funciones necesarias para gestionar desde los comandos a los *followers* y se deberá añadir un argumento al comando `chat` para poder seleccionar el personaje con el que queremos hablar.

F6. Modificar la enumeración de direcciones para que incorpore, además de las conexiones hacia los cuatro puntos cardinales (`north`, `east`, `south` y `west`), otras dos hacia arriba y abajo (`up` y `down`) para mapas con varios niveles (pisos). Implementar o modificar las funciones necesarias para manipular e imprimir dichos campos de los módulos pertinentes.

F7. Modificar el módulo Object para que incorpore soporte para nuevas propiedades junto con las funciones necesarias para su manipulación e impresión:

(a) Vida (`health`), para añadir o quitar vida a un jugador cuando lo toma. Para ello debería definirse un campo de tipo entero que indique el número de puntos de vida que añade al jugador (o le quita en caso de ser negativo) cuando el jugador ingiera este objeto. Por defecto, los objetos **no** quitan ni dan puntos de vida, es decir, este nuevo campo vale 0.

(b) Movible (`movable`), para indicar si el objeto se puede mover de su ubicación. Para ello debería añadirse un nuevo campo de tipo booleano a la estructura de datos correspondiente. Por defecto, los objetos **no** se pueden mover. Un jugador solo podrá coger objetos para su inventario si estos son movibles.

(c) Dependencia (`dependency`), para indicar si el objeto depende de otro objeto para poder ser cogido, por ejemplo, se necesita haber cogido primero `torch` para coger `fire`. El campo será de tipo `Id` y solo se puede depender de un único objeto. Por defecto los objetos **no** tienen dependencias, tomando el valor `NO_ID`. Debe tenerse en cuenta que, si se deja un objeto, también se deben soltar todos los objetos que dependan de él.

(d) Abre (`open`), para establecer si el objeto puede abrir un determinado enlace especificado por el identificador que almacena. Por omisión **no** puede abrir nada, en cuyo caso el campo tendrá el valor `NO_ID`.

F8. Crear o modificar las funciones necesarias para integrar las nuevas propiedades de los objetos. Modificar los comandos afectados llamando a estas funciones, de forma que se tenga en cuenta si los objetos son movibles o no y sus dependencias al cogerlos y dejarlos.

F9. Modificar la parte del fichero de datos correspondiente a los módulos alterados y todos los módulos afectados por los cambios realizados, por ejemplo, las funciones de carga de espacios y objetos desde ficheros.

F10. Añadir un nuevo comando para usar objetos (`use <obj> [ over <chr> ]`, donde `<obj>` es el nombre de un objeto y chr el de un personaje amigo que sigue a un jugador). Si el objeto en cuestión tiene un valor 0 en su campo `health` se asumirá que no se puede utilizar dicho objeto. Cuando un jugador usa el objeto, este desaparece del juego y al jugador se le añadirán o quitarán los puntos de vida correspondientes. Si en el comando se especifica un personaje amigo sobre el que utilizar el objeto, los puntos de vida se le añadirán o quitarán a dicho personaje. Se debe tener en cuenta que especificar un personaje en este comando es opcional.

F11. Añadir un nuevo comando para abrir enlaces con objetos (`open <lnk> with <obj>`, donde `<lnk>` es el nombre de un enlace y `<obj>` el de un objeto), por ejemplo `open door with key` o `open wall with tnt`) partiendo del procedimiento seguido en comandos anteriores.

F12. La visualización del juego se deja como decisión de diseño pudiendo modificarse respecto a lo realizado en iteraciones anteriores. Sin embargo, se debe mostrar por pantalla toda la información necesaria para poder jugar al juego.

F13. Crear una aventura que incluya 10 espacios, 20 objetos, 4 personajes y 2 jugadores, por lo menos, definidos en su correspondiente fichero de datos (incluyendo espacios, enlaces, objetos, personajes, jugadores, etc.).

F14. Crear una documentación de usuario que incluya:

(a) Breve descripción del argumento del juego.

(b) Toda la información necesaria para utilizar el juego.

(c) Un mapa del mismo con espacios, enlaces y ubicación de los objetos, similar al ejemplo incluido para el Hormiguero en el enunciado de la I3.

(d) Las indicaciones para ir desde el inicio hasta un final del juego por un camino donde se pongan de manifiesto todas las características y funcionalidades implementadas.

(e) Un fichero con la lista de comandos para seguir el camino anterior. Un ejemplo de este tipo de archivo podría ser:

```
move north
move west
inspect potion
take potion
use potion
move west
drop potion
...
exit
```

F15. Implementación de un módulo GameManagement (Gestor de Partida). Ahora, el módulo GameReader será sustituido por GameManagement, el cual no solo se encargará de cargar los datos de un juego, sino también de guardar el estado actual de una partida para poder recuperarla más adelante. Con este fin, este módulo debe incorporar una función para salvar (`game_management_save`) y otra para cargar (`game_management_load`). La función `game_management_save` almacenará en el fichero especificado el estado actual de la partida, esto es, el contenido de la estructura Game. Con respecto a `game_management_load`, el objetivo no es otro que cargar el contenido de la estructura Game de acuerdo con el contenido del fichero cuyo nombre deberá pasarse como argumento a la función de carga.

F16. Añadir dos nuevos comandos para permitir al usuario salvar y cargar partidas. El comando

para salvar una partida (`save`) debe recibir el nombre del fichero donde se guardarán los datos de la estructura Game. Y el de cargar una partida (`load`) debe recibir el nombre del fichero con los datos que tiene actualmente la partida a cargar.

F17. Implementar un módulo GameRules (Reglas de Juego). A fin de dar cierto carácter no determinista a la ejecución del juego, además de las acciones del usuario, se implementarán acciones que solo podrá ejecutar el juego. Así, basándose en el módulo Command con los comandos del usuario, deberá implementarse el módulo GameRules donde deberían añadirse las acciones que podrá ejecutar el juego sin intervención del usuario, tales como cerrar o abrir ciertos enlaces (por ejemplo se puede abrir una puerta cuando se muere el enemigo), cambiar los enlaces de algún espacio, cambiar de ubicación un objeto, recibir un ataque del enemigo sin previo aviso, etc. Se podría también, por ejemplo, ejecutar al azar una de las acciones (reglas) del juego tras ejecutar un determinado número de instrucciones del usuario dotando de aleatoriedad a la aventura. Se podrán añadir tantas reglas de juego como se consideren oportunas, con un mínimo de seis. Es importante fijarse en que este módulo puede aprovecharse para personalizar las acciones de un juego en concreto (es decir, es dependiente del juego).

F18. Implementar un modo de juego determinista, en el que no se ejecute ninguna acción al azar (en el ataque, reglas del juego, etc.), sino que utilice una semilla fija. Para ello, se invocará al programa con un nuevo argumento, `-d`. Este argumento será compatible con el de `log`, pudiendo combinarse ambos en cualquier orden. Este modo de juego será útil para poder realizar pruebas automáticas, ya que se podrá comparar con una salida esperada. Así mismo, permitirá ejecutar la lista de comandos para seguir las indicaciones del juego con garantías del resultado.

F19. Implementar una versión mejorada de la funcionalidad multijugador. Esta versión ha de plantearse de forma que facilite o potencie la jugabilidad del juego propuesto. En concreto, se debe implementar al menos una de las siguientes posibilidades:

- Un jugador mantiene el turno hasta que ejecuta, por ejemplo, un comando de movimiento.
- Permitir realizar varios ataques sin cambiar de turno hasta que el enemigo o el jugador mueran.
- Un jugador mantiene el turno durante un número de turnos determinado.
- Permitir reintentar comandos hasta que se introduzca uno correcto.
- Incrementar el número de turnos consecutivos de un jugador si porta un cierto objeto.

Dicha funcionalidad debe aparecer descrita en la guía de usuario.

F20. De forma análoga al comando `following`, implementar un comando que permita a los jugadores cooperar formando equipos, de forma que se pueda combinar ataques, usar objetos que portan otros jugadores del equipo, etc.

F21. Adaptar el fichero de log, interfaz y demás módulos que lo requieran para que se pueda identificar el usuario que ha ejecutado cada comando. Por ejemplo, se puede indicar al final de la línea y entre paréntesis una `P` seguida del Id del jugador que ejecuta el comando, tipo `move north: ERROR (P2)`.

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. Realizar un script de bash que reciba como primer argumento un número con valor `0` o `1`, y como segundo argumento el nombre del fichero de pruebas sin extensión (por ejemplo, `space_test`) con el que se desea trabajar. Dicho script compilará el fichero indicado mediante llamadas a `make` y lo ejecutará si el primer argumento vale `0` o lo ejecutará utilizando `valgrind` cuando el primer argumento valga `1`. Cuando no se indique nada en el segundo argumento, compilará y ejecutará todas las pruebas correspondientes a los programas terminados en `_test.c`. La entrega de dicho script se realizará durante la segunda semana de la iteración.

P3. Definir pruebas de integración **automáticas** para el proyecto completo, incluyendo las nuevas mejoras. Para ello será útil el fichero de log, que generará una salida con el listado de comandos y si se han ejecutado correctamente o no, que se puede comparar frente a la salida esperada con algún comando similar a `diff`[3]. Se recomienda además desactivar la parte aleatoria del juego para evitar comportamientos no deterministas que impidan una comprobación automática en este entorno de pruebas.

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 I4. 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.

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

- No se ha completado el diagrama de Gantt ni acta de al menos una reunión.
- El código entregado no compila o no se puede probar.
- 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.

[3] Ver SPOC

Tabla 1: Rúbrica para la cuarta iteración (I4).

Objetivo	Puntuación	Aprobado	Notable	Sobresaliente
G1	0,50			
G2	0,50			
C1	0,40			
C2	0,20			
C3	0,20			
C4	0,20			
D1	0,50			
D2	0,50			
D3	0,50			
F1	0,20			
F2	0,20			
F3	0,20			
F4	0,10			
F5	0,20			
F6	0,10			
F7	0,50			
F8	0,20			
F9	0,10			
F10	0,20			
F11	0,20			
F12	0,20			
F13	0,30			
F14	0,10			
F15	0,30			
F16	0,10			
F17	0,40			
F18	0,20			
F19	0,40			
F20	0,20			
F21	0,10			
P1	0,50			
P2	1,00			
P3	0,50			

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).

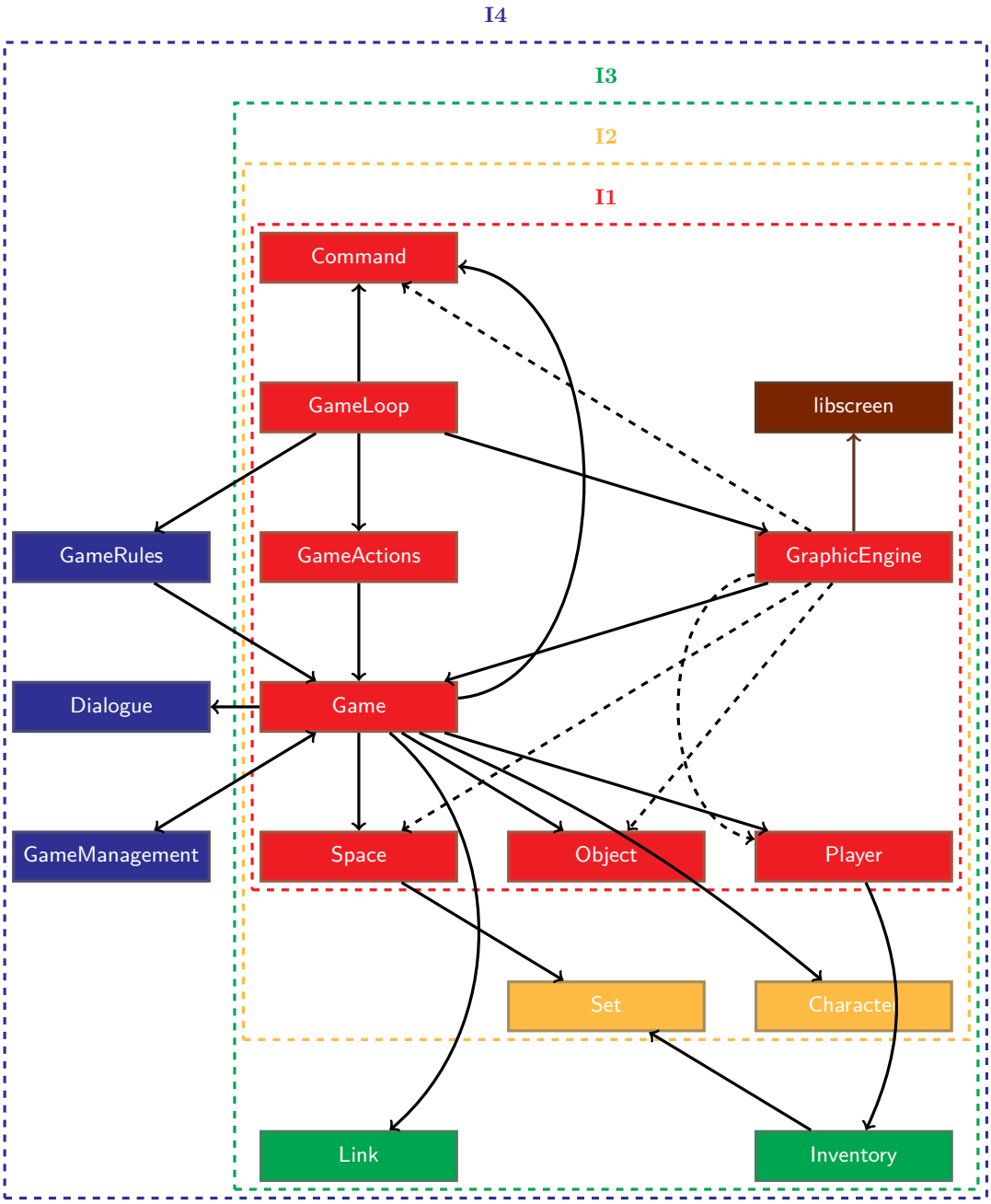

Figura 1: Módulos considerados en la cuarta iteración (I4) del desarrollo del proyecto.