Segunda Iteración del Proyecto (I2): Juego de la Oca

Introducción

Siguiendo el desarrollo del proyecto y de los conceptos, herramientas y habilidades necesarios para ello, en esta segunda iteración (I2) se continuará utilizando el Juego de la Oca como modelo sencillo de Aventura Conversacional, como se explicó en el documento de introducción a dicho juego.

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). Se trata de la segunda aproximación al desarrollo de los módulos esenciales del sistema, como se explica en el documento de introducción al proyecto.

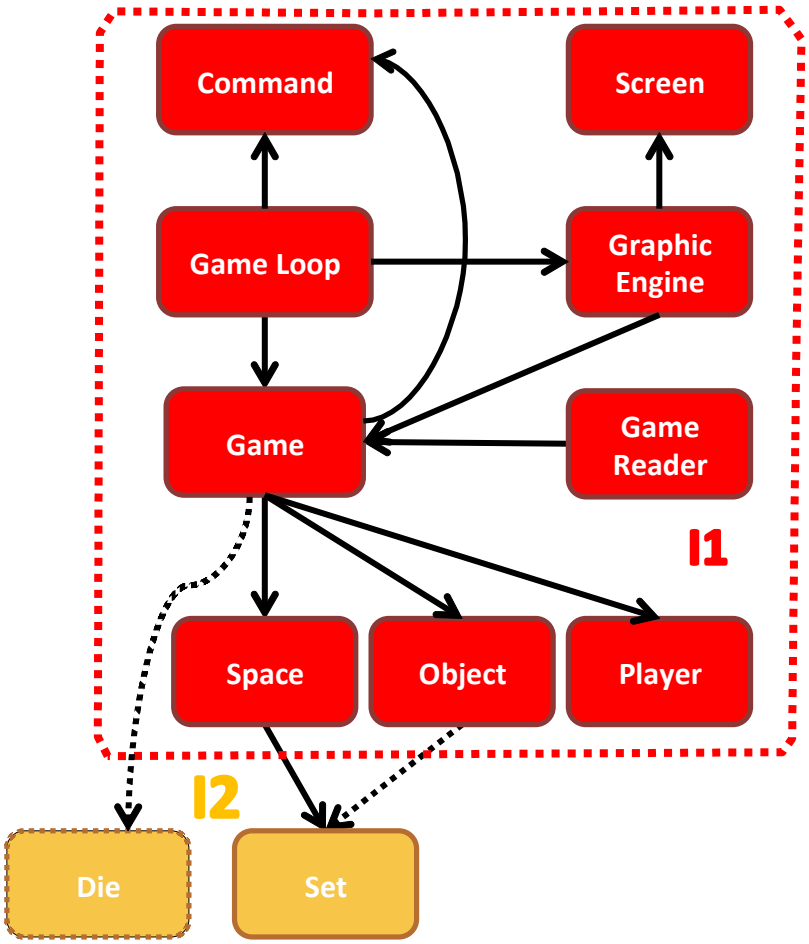

Figura 1. Módulos considerados en la segunda iteración (I2) del desarrollo proyecto.

Los módulos obtenidos como resultado de la I1 se han representado en rojo en la Figura 1. En ocre se presentan los módulos que se desarrollarán en la segunda iteración. Además, se ampliarán y utilizarán los de color rojo. Los nuevos módulos son dos: `Set` (Conjunto) y `Die` (Dado). El módulo `Set` proporcionará la funcionalidad necesaria para manejar conjuntos de identificadores, como los que tienen 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 `Die` facilitará la funcionalidad necesaria para obtener valores aleatorios en un rango establecido, como se consigue al tirar un dado.

El módulo `Set` es un caso de módulo reutilizable, puesto que proporciona el soporte para funcionalidades de otros módulos (inicialmente de `Space` y, en futuras iteraciones, de `Object` y `Player`). Por su parte `Die` es un módulo necesario para el Juego de la Oca, pero no es imprescindible para el proyecto final. Su interés estriba en la necesidad de utilizar para su desarrollo funciones de las bibliotecas estándar de C de uso menos frecuente, pero importantes (como son las de generación de números pseudo-aleatorios y de manejo del tiempo).

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

1. Cargar tableros de juego (espacios) 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 tablero de un objeto y del jugador).
3. Soportar la interacción del usuario con el sistema, interpretando comandos para mover al jugador por el tablero (adelante y atrás), hacer que el jugador manipule un objeto en el tablero (cogerlo y dejarlo), y salir del programa.
4. Mover al jugador por los espacios, pudiendo coger y dejar objetos, y haciendo cambiar el estado del juego.
5. Mostrar la posición en cada momento del jugador y de las casillas contiguas 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 las fichas necesarias para 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. Soportar la interacción del usuario para utilizar un dado y saltar entre casillas especiales (ocas, puentes y muerte).
4. Mostrar la descripción gráfica (ASCII) de los espacios que la tengan, la posición de todos los objetos y el último valor del dado.

Objetivos

Los objetivos de esta segunda iteración del proyecto (I2) 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.

Las mejoras y modificaciones requeridas (requisitos R1, R2., etc.) son las siguientes:

1. [R1] Crear un módulo `Set` (conjunto) que integre la funcionalidad necesaria para el manejo de conjuntos. En particular, los conjuntos deberán implementarse como una estructura de datos con dos campos, uno para almacenar un array de identificadores, y otro para recordar el número de ellos en cada momento, así como facilitar 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`). Crear un programa (`set_test`) para probar el módulo.
2. [R2] 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 campo de objeto presente, por otro campo `objects` (con 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.
3. [R3] Crear un módulo `Die` (dado) utilizando las bibliotecas estándares de ANSI C (`stdlib` y `time`) para generar enteros aleatorios dentro de un intervalo. En particular el dado debería implementarse como una estructura con un identificador (por si hubiera más de uno en el juego) y un valor entero para guardar el número obtenido en la última tirada, para mostrarlo con el estado del juego. También en este caso, el módulo debería facilitar las funciones necesarias para crear y destruir dados (`create` y `destroy`), lanzar el dado (`roll`) e imprimir el contenido de los mismos durante la depuración (`print`). Crear un programa (`die_test`) para probar el módulo.
4. [R4] Modificar, en caso de necesidad, los demás módulos existentes para que utilicen los nuevos 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, y, además, añadir un puntero a `Die` para disponer de un dado. En todos los casos, se utilizarán las funciones adecuadas de los nuevos módulos para la manipulación necesaria de datos desde las primitivas de los demás.
5. [R5] Añadir al fichero de carga de datos (`data.dat`) aquellos datos relativos a los objetos que se utilizarán en el juego, siguiendo el modelo del empleado para cargar los espacios. El fichero debe incluir cuatro objetos.
6. [R6] 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.
7. [R7] Añadir un nuevo comando para tirar el dado.
8. [R8] Añadir dos nuevos comandos (`left` o `l`, `right` o `r`) para saltar entre casillas especiales (ocas, puentes y muerte) que además de estar enlazadas con otras secuencialmente (norte/sur) también lo están de forma especial (este/oeste).
9. [R9] Modificar el comando que permite al jugador coger un objeto de un espacio (comando `grasp` o `g`), 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 o nombre establecidos para el objeto (ficha en el tablero) que se quiere coger, por ejemplo: “grasp O4” (ver Figura 2).

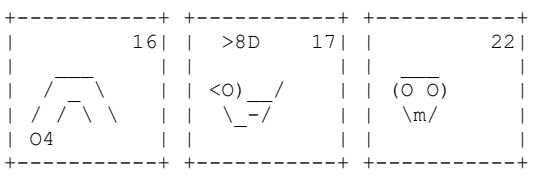

Figura 2. Ejemplos de descripciones gráficas (ASCII) de espacios: izquierda, casilla de puente con ficha `O4`; centro, casilla de oca con el jugador `>8D`; y, derecha, casilla de la muerte.

10. [R10] Modificar una vez más el módulo `Space` para incluir una descripción gráfica (ASCII) del espacio de 3x7 caracteres (ver Figura 2). Para ello podría incluirse un campo `gdesc` que fuera un array de tres strings, cada uno con espacio para siete caracteres. De este modo, por ejemplo, la descripción gráfica de la casilla 17 de la Figura 2 estaría formada por los strings:

```
"       "
"<O)__/ "
" \_-/  "
```

Y la línea del fichero de datos correspondiente a dicha casilla podría ser:

```
"#s:17|Casilla 17 (Oca)|16|-1|18|21|       |<O)__/ | \_-/  |"
```

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

11. [R11] Crear un nuevo fichero de carga de espacios para implementar un tablero para el Juego de la Oca con 25 casillas todas enlazadas secuencialmente (norte/sur). Además, las casillas correspondientes a ocas (5, 9, 13, 17, 21 y 25) unidas mediante un enlace adicional (este/oeste), cada oca con la siguiente oca. También, las casillas correspondientes a puentes (8 y 16) enlazadas mutuamente (también este/oeste). Y, por último, la correspondiente a la muerte (22) unida (de nuevo este/oeste) con la casilla de salida (1). Adicionalmente, se incluirán en el fichero las descripciones gráficas para las casillas de oca, puente y muerte (ver Figura 2).
12. [R12] Modificar la función de carga de espacios para adaptarse a los cambios introducidos en el formato.
13. [R13] Modificar la función de visualización del estado del juego para mostrar: (a) la descripción gráfica (ASCII) de cada espacio; (b) el espacio donde está cada objeto; (c) varios objetos en los espacios visualizados; (d) los objetos que porta el jugador; (e) el último valor del dado; y (f) el último comando ejecutado seguido del resultado de su ejecución (OK o ERROR) (ver Figura 3). Tenga en cuenta que puede que deba redimensionar alguna ventana de la interfaz de usuario.

```
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~  +-----------+                                 ~  Object location:           ~
~  | >8D     16|>                                ~  O1:16, O2:16, O3:18        ~
~  |    ___    |                                 ~                             ~
~  |  / _ \    |                                 ~  Players objects: O4        ~
~  | / / \ \   |                                 ~                             ~
~  | O1,O2     |                                 ~  Last die value: 5          ~
~  +-----------+                                 ~                             ~
~        v                                       ~                             ~
~  +-----------+                                 ~                             ~
~  |         17|>                                ~                             ~
~  |           |                                 ~                             ~
~  | <O)__/    |                                 ~                             ~
~  |  \_-/     |                                 ~                             ~
~  |           |                                 ~                             ~
~  +-----------+                                 ~                             ~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~ The game of the Goose ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~ The commands you can use are:                                                ~
~     following(f), previous(p), grasp(g), drop(d), roll(r), exit(e)           ~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
~ Following: OK                                                                ~
~ Following: OK                                                                ~
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
prompt:>
```

Figura 3. Ejemplo de visualización de estado de juego: arriba a la izquierda, casilla 16 de puente con fichas `O1` y `O2` además del jugador `>8D`; debajo, casilla de oca 17; en la parte derecha, localización de objetos, objeto portado por jugador y último valor dado; en la parte inferior, últimos comandos ejecutados y su estado.

14. [R14] 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 `grasp` y no se consiguiera coger el objeto pretendido, en el fichero de LOG se escribiría la línea "`grasp: ERROR`", mientras que si el objeto se hubiera cogido sin problema en el fichero se escribiría "`grasp: OK`". Para indicar al programa que se desea generar un fichero de LOG, al invocarlo se pasara como últimos argumentos `–l` seguido del nombre deseado para el fichero de LOG, después de los otros argumentos requeridos. De esta manera, si para invocar al programa sin LOG la línea de comandos fuera "`game_loop fchdatos`", donde `fchdatos` es el fichero de datos a cargar, entonces, para invocar al programa especificando que se genere el LOG la línea de comandos podría ser "`game_loop fchdatos –l fchlog.log`", donde `fchlog.log` es el nombre del fichero de LOG. Para implementar esta nueva funcionalidad el programa debería: (a) procesar convenientemente los argumentos de entrada, y en caso de ser necesario, (b) antes de iniciar el bucle del juego, abrir para escritura un fichero de LOG con el nombre indicado, (c) obtener en cada iteración el resultado de la ejecución del comando y escribir éste y su resultado en el fichero de LOG como se ha dicho anteriormente, y finalmente (d) cerrar el fichero de LOG (si se está usando) antes de terminar el programa al salir del bucle de juego.
15. Modificar el `Makefile` del proyecto para automatizar la compilación y el enlazado del conjunto.
16. Depurar el código hasta conseguir su correcto funcionamiento.

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 según los siguientes criterios:

- **C**: Si se obtiene C en todas las filas de la tabla de rúbrica.
- **B**: Si se obtienen, al menos, dos Bes y el resto Ces. Excepcionalmente sólo con una B.
- **A**: Si se obtienen, al menos, dos Aes y el resto Bes. Excepcionalmente sólo con una A.

Cualquier trabajo que no cumpla los requisitos de la columna C obtendrá una puntuación inferior a 5.

Tabla de rúbrica:

	C **(5-6,9)**	**B** **(7-8,9)**	**A** **(9-10)**
Entrega y compilación	(a) Se ha entregado en el momento establecido todo el material solicitado. y (b) Es posible compilar, enlazar los fuentes de forma automatizada utilizando el `Makefile` entregado.	Además de lo anterior: (a) El `Makefile` entregado permite gestionar los fuentes del proyecto, además de compilar y enlazar los mismos. y (b) La compilación y el enlazado no producen errores ni avisos (warnings) utilizando la opción `-Wall`.	Además de lo anterior: La compilación y el enlazado no producen errores ni avisos (warnings) utilizando las opciones `-Wall -pedantic`.
Funcionalidad	Se han cubierto los requisitos R1, R2, R3 y R4, de manera que se mantiene la funcionalidad establecida para la I1.	Además de lo anterior: Se han cubierto los requisitos R5, R6, R7, R8 y R9, de forma que, además de mantenerse la funcionalidad previa, se consigue la asociada a los nuevos requisitos.	Además de lo anterior: Se han cubierto el resto de requisitos establecidos, R10, R11, R12, R13 y R14; manteniendo toda la funcionalidad previa, se consigue la asociada a los nuevos requisitos.
Estilo y documentación	(a) Que las variables y funciones tengan nombres que ayudan a comprender para qué se usan. y (b) Que todas las constantes, variables globales, enumerados y estructuras públicas se hayan comentado. y (c) Que el código esté bien indentado[1].	Además de lo anterior (a) Que los ficheros y las funciones incluyan comentarios de cabecera con todos los campos requeridos y estén correctamente comentadas y (b) Que las funciones tengan identificado un autor único y (c) Que no se violen las interfaces de los módulos	Además de lo anterior (a) Que el estilo sea homogéneo en todo el código2 y (b) Que las variables locales a cada módulo o función que precisen explicación se hayan comentado y (c) Que se controlen los argumentos pasados a las funciones y los retornos de las funciones de entrada y reserva de recursos

[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 codificación.