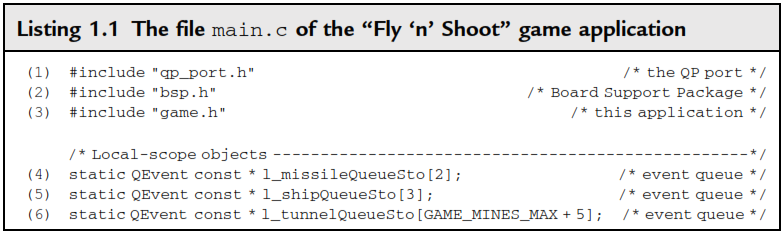
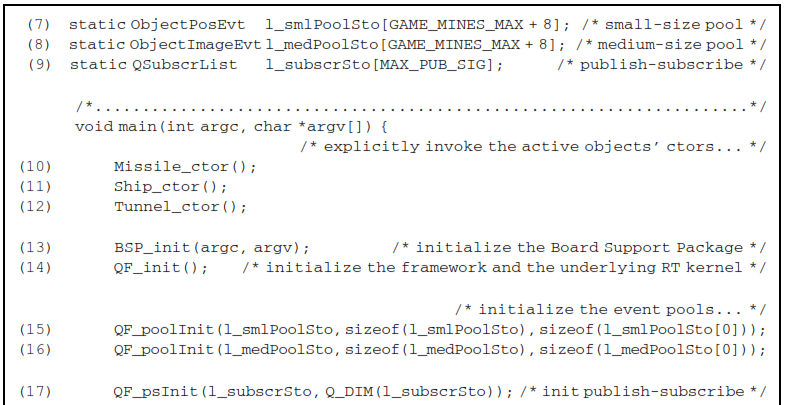
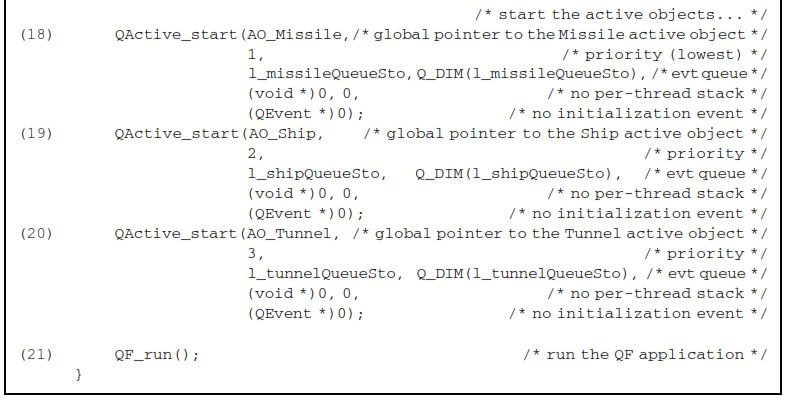
La Función main()

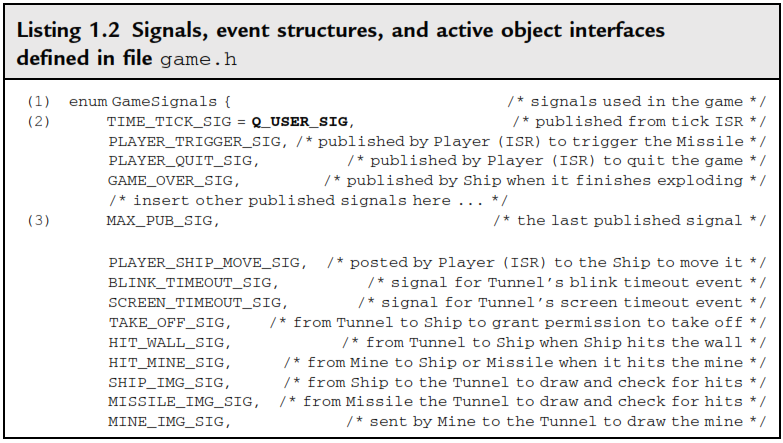
Tal vez el mejor lugar para empezar la explicación del código de la aplicación “Fly ‘n’ Shoot” es la función main, localizada en el archivo main.c. El archivo main.c completo se mueatra en el Listado 1.1.

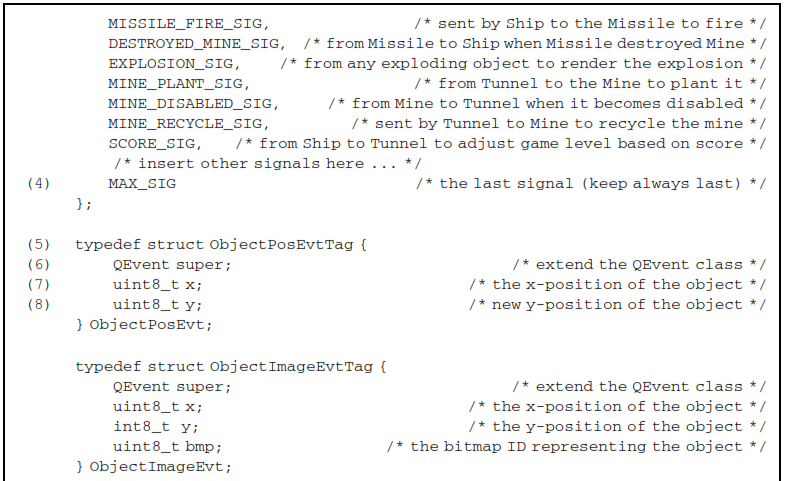


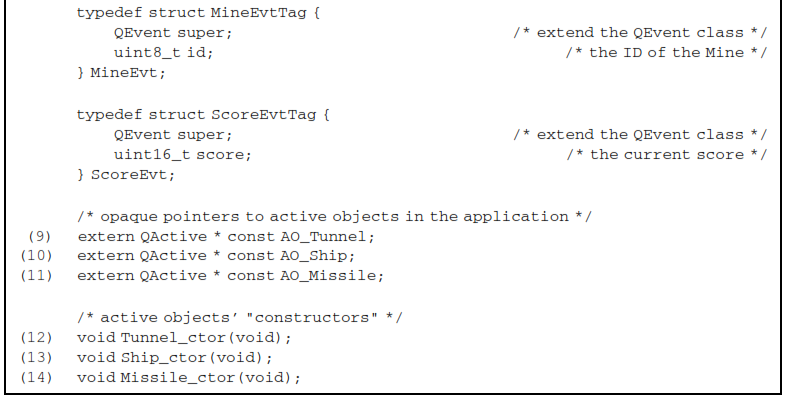




Debido a que los eventos son compartidos explícitamente entre la mayoría de los componentes de la aplicación, es conveniente declararlos en el archivo de cabecera separado game.h mostrado en el Listado 1.2

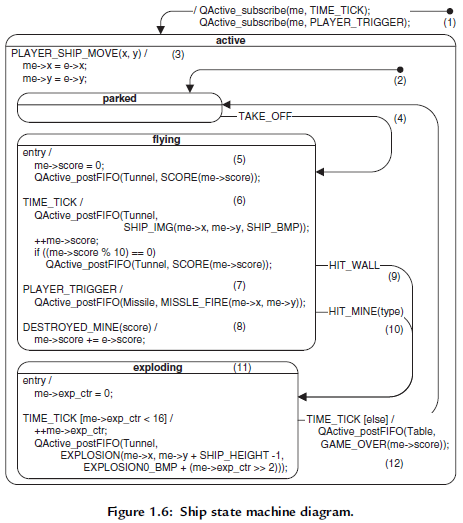




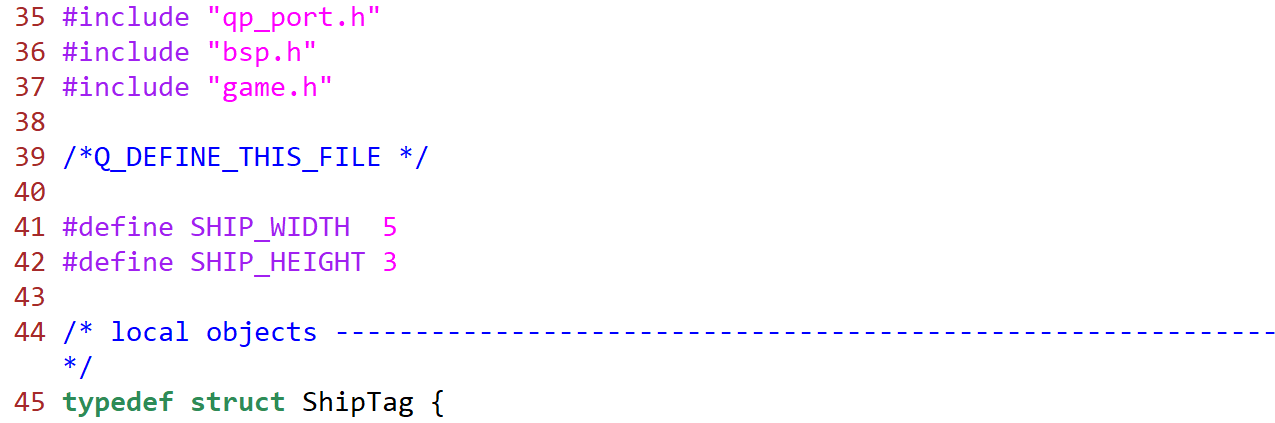


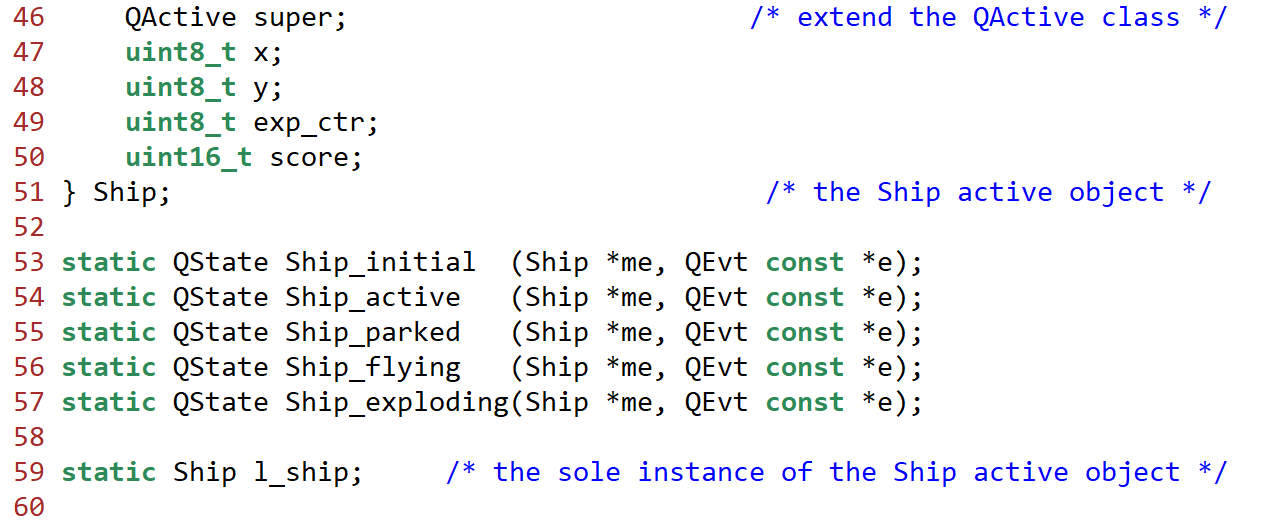
Codificando Máquinas de Estado Jerárquicas

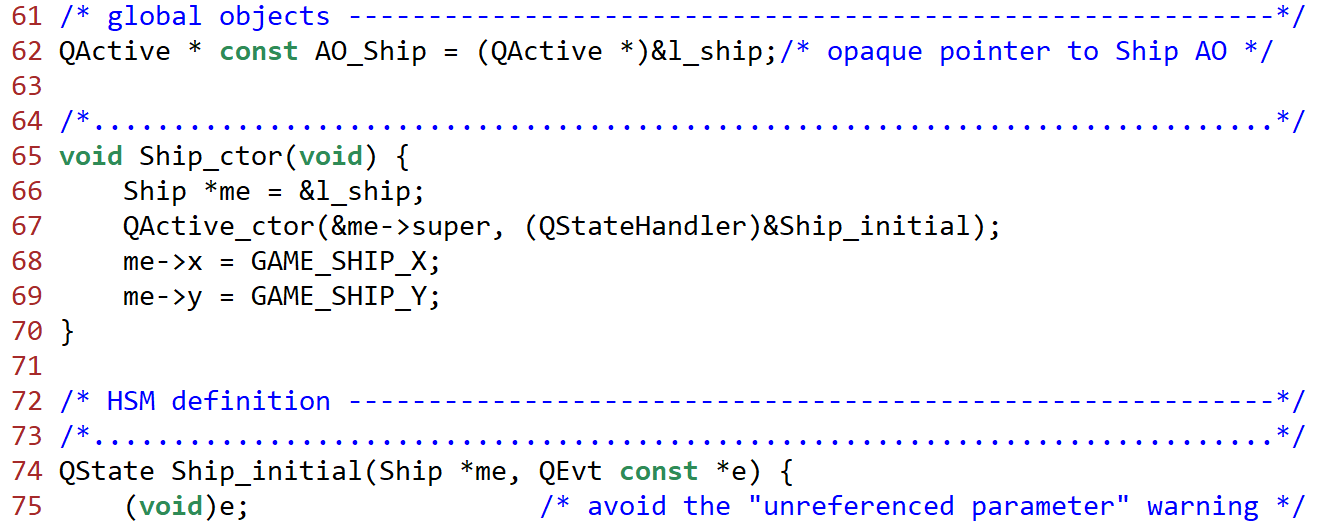
Contrario a lo que muchas personas piensan, no se necesitan herramientas de automatización de diseño grandes para traducir máquinas de estados jerárquicas (cartas de estado UML) a código eficiente y fácil de mantener en C o C++. A continuación se muestra cómo codificar manualmente la máquina de estado Ship de la Figura 1.6 con la ayuda del framework de tiempo real QF y el procesador jerárquico QEP, el cual también es parte de la plataforma dirigida por eventos QP.

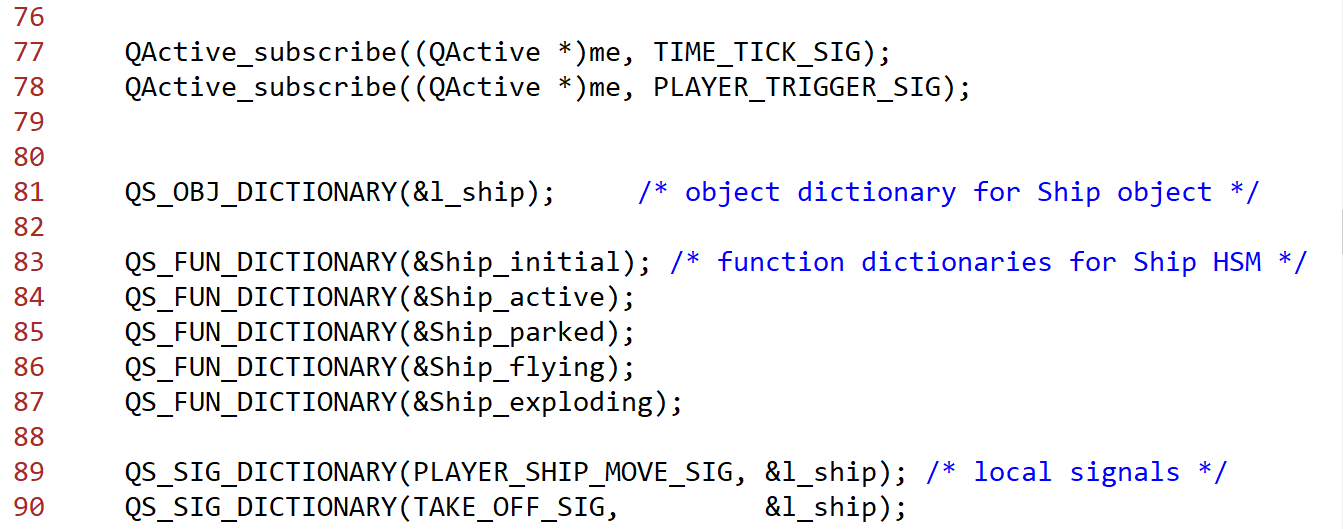


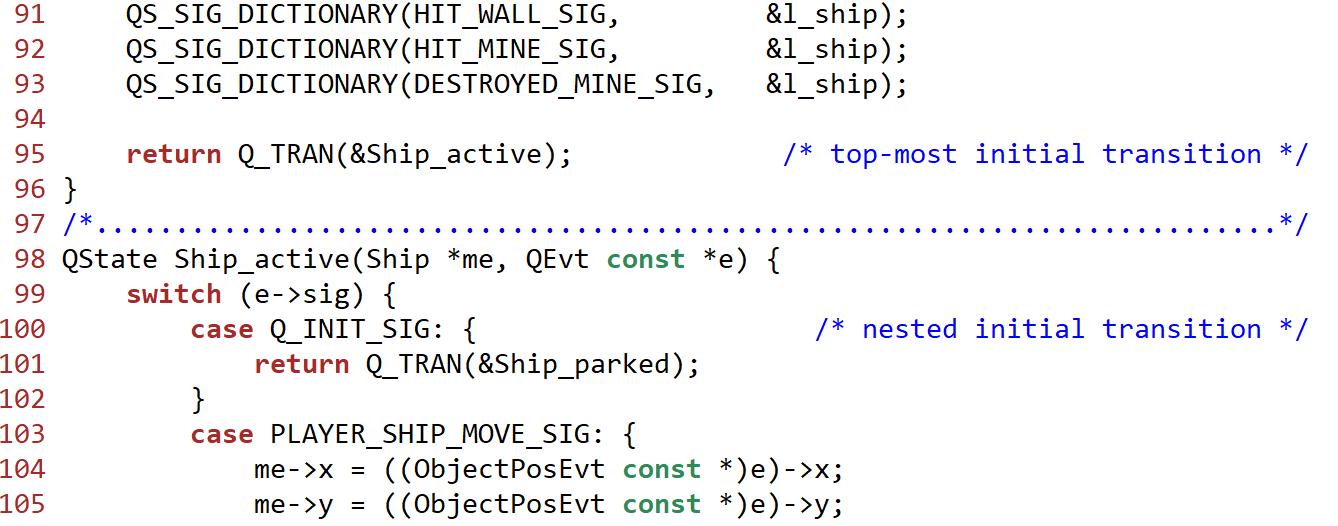
El código fuente para la máquina de estado Ship está en el archivo ship.c, el cual se muestra a continuación

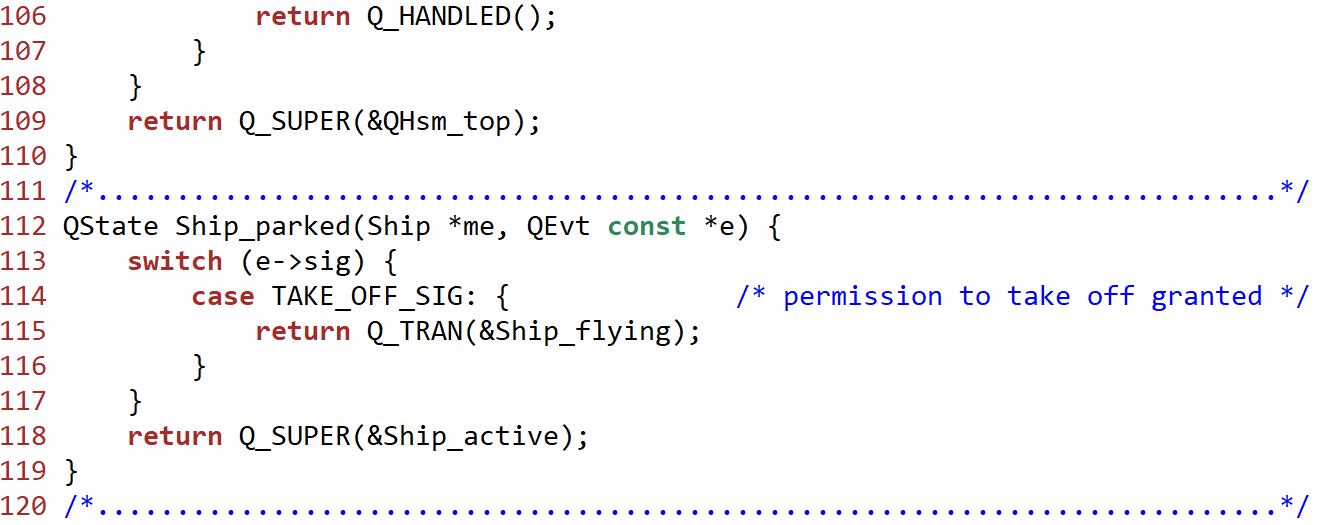


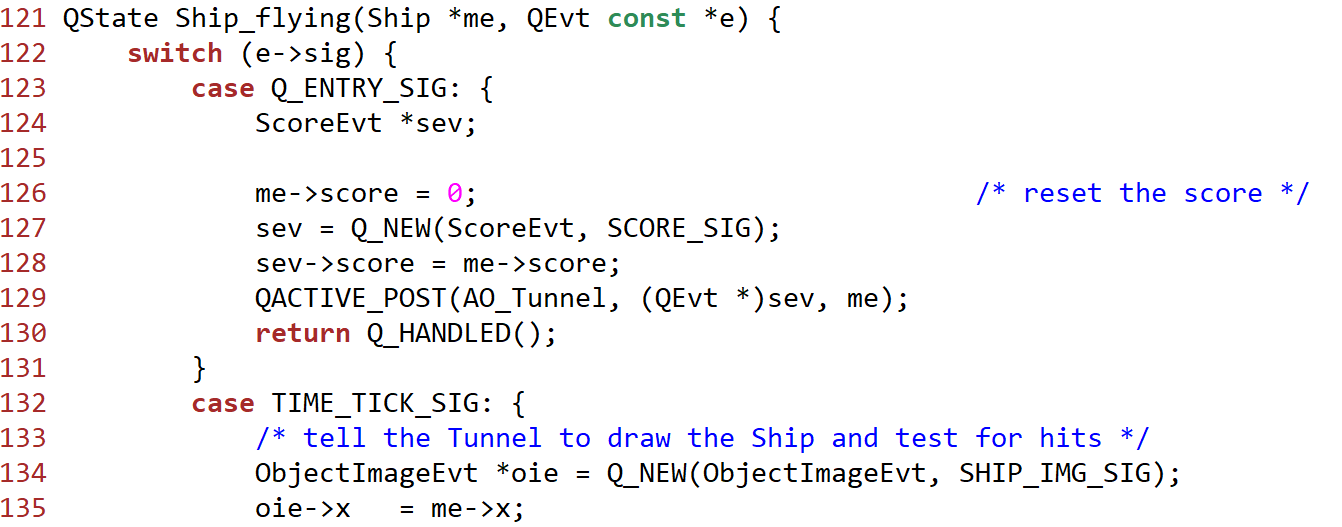


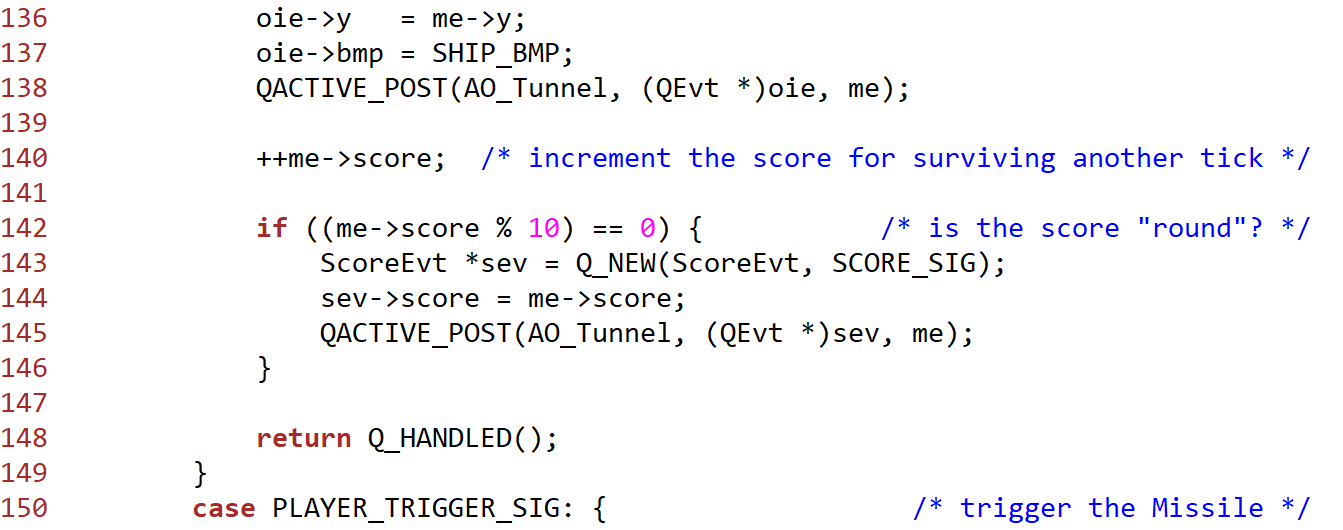


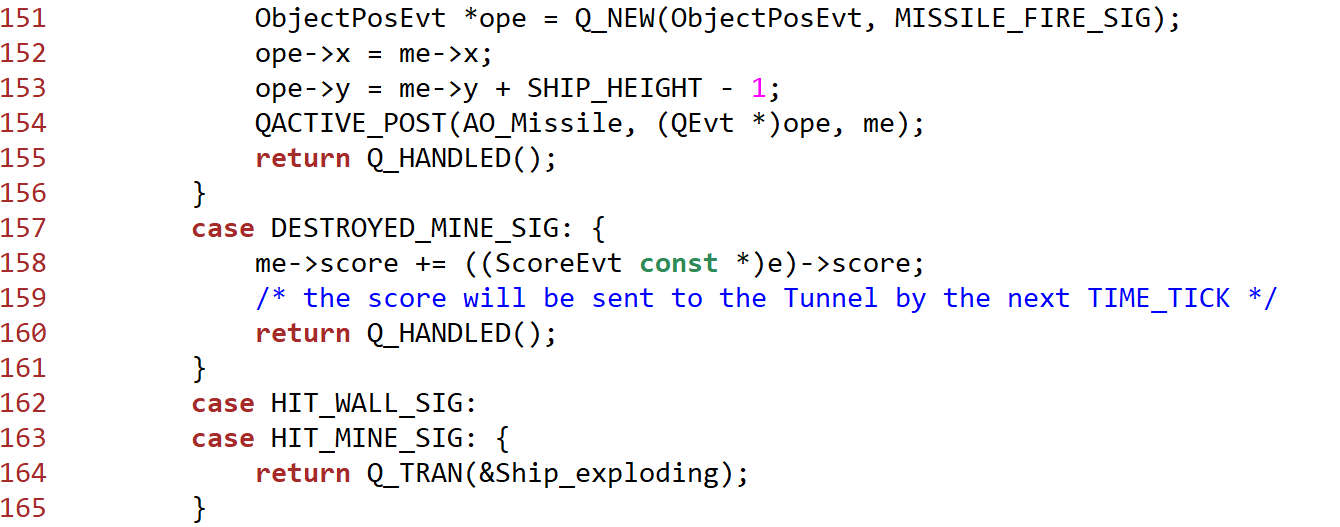


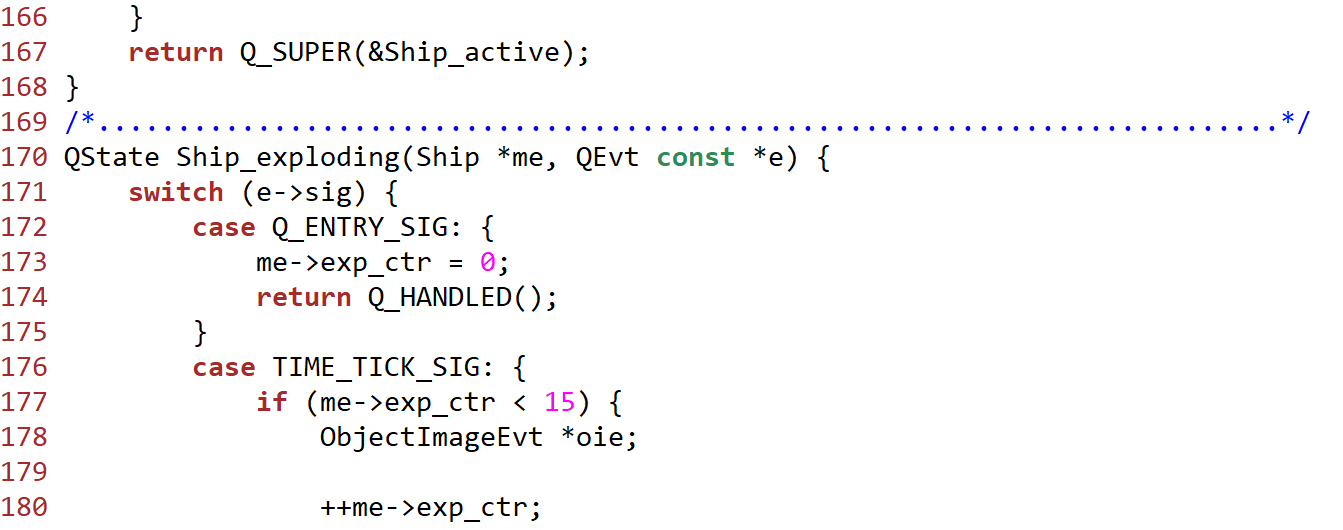


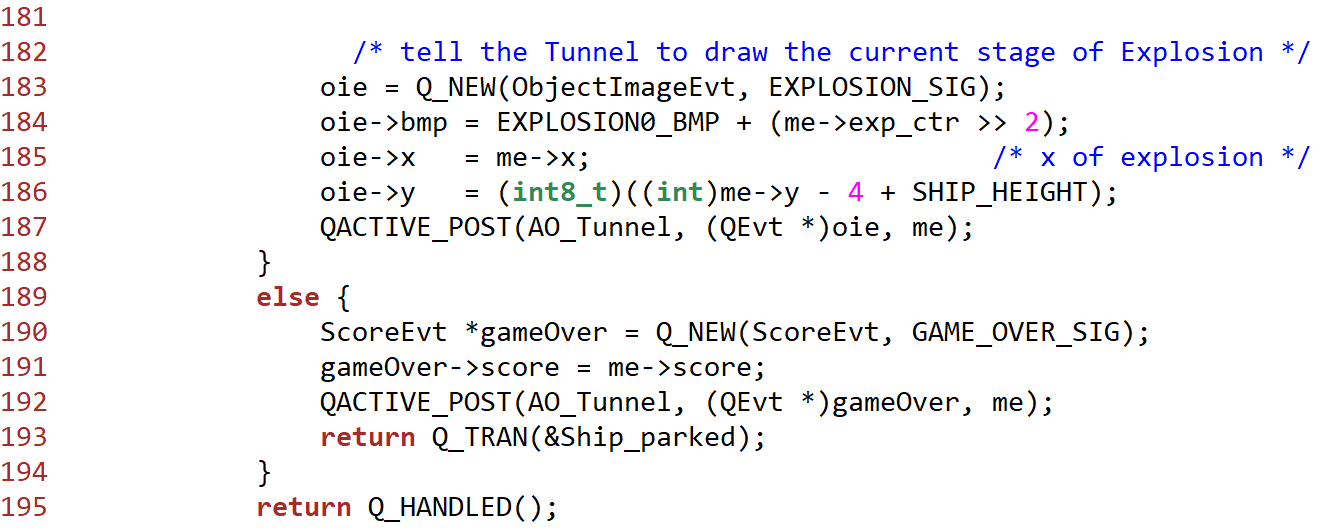


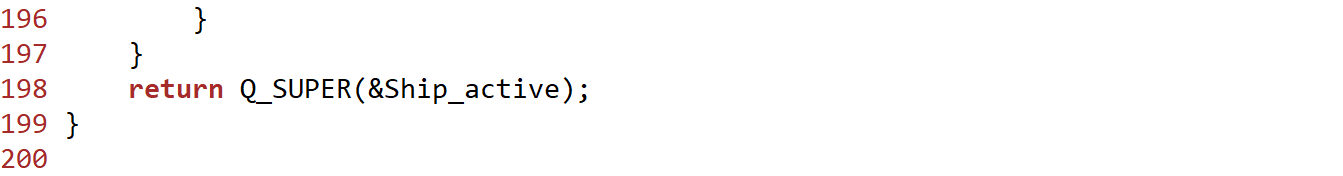












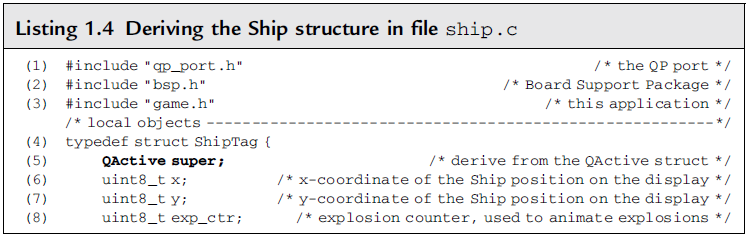
Archivo ship.c

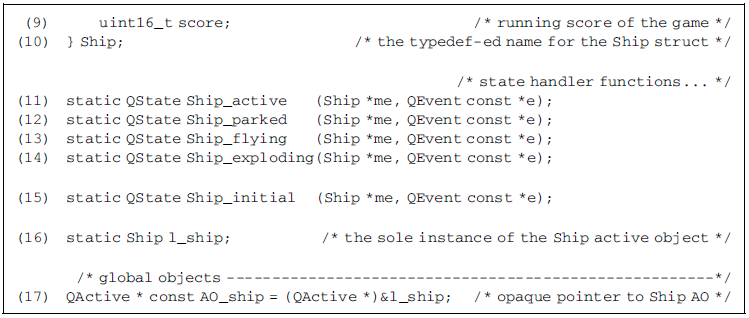
La explicación de este este archivo será dividida en tres pasos.

Paso 1: Definiendo la Estructura Ship

Como primer paso, se define la estructura de datos Ship. Se utiliza herencia para derivar la estructura Ship de la estructura de framework QActive (véase la nota Herencia\_Simple\_en\_C.docx). Esta relación de herencia liga la estructura Ship al framework QF.

La principal responsabilidad de la estructura base QActive es almacenar la información acerca del estado activo actual de la máquina de estados así como la cola de eventos y el nivel de prioridad del objeto activo Ship. De hecho, se deriva de una estructura QEP más simple QHsm que representa solo el estado activo actual de una máquina de estados jerárquica. Encima de esa información, casi toda máquina de estados debe también almacenar otra información de “estado-extendido”. Por ejemplo, el objeto Ship es responsable de almacenar su posición (la posición de la nave) así como la puntuación acumulada en el juego. Esta información adicional se proporciona por medio de los atributos listados después del atributo super, como se muestra en el listado 1.4.





1. Cada archivo de C a nivel de aplicación que usa la plataforma QP debe incluir el archivo de cabecera qp\_port.h.
2. El archivo de cabecera bsp.h contiene la interface para el Paquete de Soporte de Trajeta (Board Support Package).
3. El archivo de cabecera game.h contiene las declaraciones de eventos y otras facilidades compartidas entre los componentes de la aplicación (véase Listado 1.2).
4. Esta estructura define el objeto activo Ship.

NOTA: Es preferible mantener los objetos activos, y todos los objetos máquina de estado (tales como las minas en el ejemplo Fly and Shoot), estrictamente encapsulados. Por lo tanto, no se ponen las definiciones de estructura de máquina de estados en archivos de cabecera; en lugar de eso, se definen directo en el archivo de implementación, tal como en ship.c. De esa forma podemos estar seguros de que los atributos de la estructura Ship no son conocidos en ninguna otra parte de la aplicación.

1. La estructura de objeto activo Ship se deriva de la estructura de framework QActive.

(6,7) Los atributos x e y representan la posición de la nave en la pantalla.

(8) El atributo exp\_ctr se usa para llevar la cuenta de las animaciones de explosión (véase también el estado “exploding” dn dl diagrama de estado de Ship en la Figura 1.6).

(9) El atributo score almacena los puntos acumulados en el juego.

(10) Se utiliza el typedef para definir el nombre más corto Ship como equivalente a struct ShipTag.

(11-14) Estas cuatro funciones son llamadas *funciones manejadoras de estado* porque corresponden uno a uno a los estados de la máquina de estado Ship mostrada en la Figura 1.6. Por ejemplo, la función Ship\_active() representa el estado “activo”. El procesador de eventos QEP llama a las funciones manejadoras de estado para realizar la semántica UML de la ejecución de una máquide estado. Todas las funciones manejadoras de estado tienen la misma firma. Una función manejadora de estado toma el apuntador a la máquina de estado y el apuntador a evento como argumentos y devuelve el status de la operación de regreso al procesador de eventos ---por ejemplo si el evento fue manejado o no. El tipo de retorno QState de las funciones manejadoras de evento se hace equivalente a uint8\_t mediante typedef en el archivo de cabecera <qp>/qpc/include/qep.h.

NOTA:

Se usa una convención de nombres simple para fortalecer la asociación entre las estructuras y las funciones diseñadas para operar sobre esas estructuras. Primero, se nombra a las funciones combinando el nombre de estructura creado con typedef con el nombre de la operación (p.ej., Ship\_active). Segundo, siempre se coloca el apuntador a la estructura como el primer argumento de la función asociada y siempre se nombra a este argumento “me” (p. ej., Ship\_active(Ship \*me, . . . ) ).

(15) Además de las funciones manejadoras de estado, cada máquina de estado debe declarar el pseudo estado inicial, el cual QEP invoca para ejecutar la transición más inicial (véase Figura 1.6(1) ). El manejador de pseudoestado inicial tiene una firma idéntica a la de la función manejadora de estado regular.

(16) En esta línea, se aloja estáticamente el almacenamiento para el objeto activo Ship. Note que el objeto l\_ship es definido como static así que es accesible colamente localmente en el alcance de archivo del archivo ship.c.

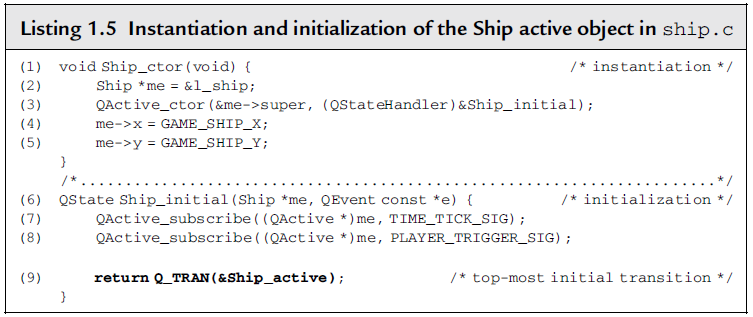
(17) En esta línea se define e inicializa el apuntador global AO\_Ship al objeto activo (véase también Listado 1.2(10)). Este apuntador es “opaco” porque trata el objeto Ship más como la estructura base QActive genérica que como la estructura Ship específica. El poder de un apuntador “opaco” es que nos permite esconder completamente la definición de la estructura Ship y hacerla inaccesible al resto de la aplicación. Pero a pesar de ello, los otros componentes de la aplicación pueden acceder al objeto Ship para enviar eventos directamente a él a través de la función QActive\_postFIFO(QActive\*me,QEvent const \*e).

Paso 2: Inicializando la Máquina de Estados

La inicialización de la máquina de estados se divide en los siguientes dos pasos para incrementar la flexibilidad y controlar mejor la línea de tiempo de inicalización:

1. El “constructor” de la máquina de estados; y
2. La transición inicial más alta.

El “constructor” de la máquina de estados, tal como Ship\_ctor(), intencionalmente no ejecuta la transición inicial más alta (topmost) definida en el pseudoestado inicial porque en ese momento algunos objetos vitales pueden no estar y hardware crítico podría no estar apropiadamente inicializado aún (en C++ los constructores estáticos corren inclusive antes que main()). En lugar de eso, el “constructor” de la máquina de estados solamente pone la máquina de estados en el pseudoestado inicial. Más tarde, el código de usuario debe disparar la transición inicial topmost explícitamente, lo cual sucede realmente dentro de la función QActive\_start() (véase Listado 1.1(18-20)). El Listado 1.5 muestra la instanciación (la función “constructor”) y la inicialización (el pseudoestado inicial) del objeto activo Ship.



1. La función global Ship\_ctor() cuyo protoipo está en game.h y es llamada al principio de main().
2. El apuntador “me” apunta al objeto Ship estáticamente alojado (véase Listado 1.4(16)).
3. Cada estructura derivada es responsable de inicializar la parte heredada de la estructura base. El “constructor” QActive\_ctor() pone la máquina de estados en el pseudoestado inicial &Ship\_initial (véase Listado 1.4(15)).

(4,5) La posición de Ship (la nave) es inicializada.

1. La función Ship\_initial() define la transición inicial topmost en la máquina de estado Ship (véase Figura 1.6(1))

(7,8) El objeto activo Ship se suscribe a las señales TIME\_TICK\_SIG y PLAYER\_TRIGGER\_SIG, como está especificado en el diagrama de estados en la Figura 1.6(1).

(9) El estado inicial “active” es especificado devolviendo la macro Q\_TRAN() de QP.

NOTA: La macro Q\_TRAN() siempre debe seguir a la sentencia return.

Paso 3: Definiendo las Funciones Manejadoras de Estado

En el último paso, usted realmente codifica la máquina de estados Ship implementando un estado a la vez como una función manejadora de estado. Para determinar qué elementos pertenecen a alguna función manejadora de estado dada, usted debe observar alrededor de la frontera del estado en el diagrama (Figura 1.6). Usted necesita implementar todas las transiciones que se originan en la frontera, cualesquiera acciones de entrada y salida definidas en el estado, y todas las transiciones internas enlistadas directamente en el estado. Adicionalmente, si hay una transición inmersa directamente en el estado, usted necesita implementarla también.

Tómese, por ejemplo, el estado “flying” mostrado en la Figura 1.6. Este estado tiene una acción de entrada y dos transiciones que se originan en su frontera: HIT\_WALL y HIT\_MINE(type) así como tres transiciones internas TIME\_TICK, PLAYER\_TRIGGER, y DESTROYED\_MINE(score). El estado “flying” se anida dentro del superestado “active”.

El Listado 1.6 muestra dos funciones manejadoras de estado de la máquina de estados Ship de la Figura 1.6. Las funciones manejadoras de estado corresponden a los estados “active” y “flying,” respectivamente. Después del listado se resaltan las técnicas de implementación importantes utilizadas en el mismo.

