MEMORIA DEL PROYECTO:

dSnake

SISTEMAS EMPOTRADOS DISTRIBUIDOS

Máster en Ingeniería Informática

Curso 2015-16

Autores:
Jorge Casas Hernán
Mariano Hernández García

Autorización

Nuestro proyecto y esta memoria han sido desarrollados por Jorge Casas Hernán y Mariano Hernández García para la asignatura Sistemas Empotrados Distribuidos, del Máster en Ingeniería Informática de la Universidad Complutense de Madrid durante el curso 2015/16.

Damos nuestro consentimiento a los profesores para que difundan, con fines académicos y nombrándonos, tanto el contenido de esta memoria, como el código entregado.

Índice de contenidos:

Índice de ilustraciones:

1. Introducción

Nuestro proyecto, **d**Snake (Distributed Snake), consiste en el desarrollo del clásico videojuego de los 70 *Snake*[1] para dos jugadores. Para ello hemos empleado 2 maletines con la placa S3CEV40 y una Raspberry Pi 2[2]. Cada jugador controlará a una de las serpientes por medio del teclado matricial de cada uno de los maletines. El objetivo es eliminar a la serpiente del otro jugador haciendo que colisione con un muro o con la cola de la otra serpiente. Además, aparecerán de forma aleatoria frutas que harán crecer a la serpiente que se las coma. La Ilustración 1 muestra una captura de la versión original del juego.

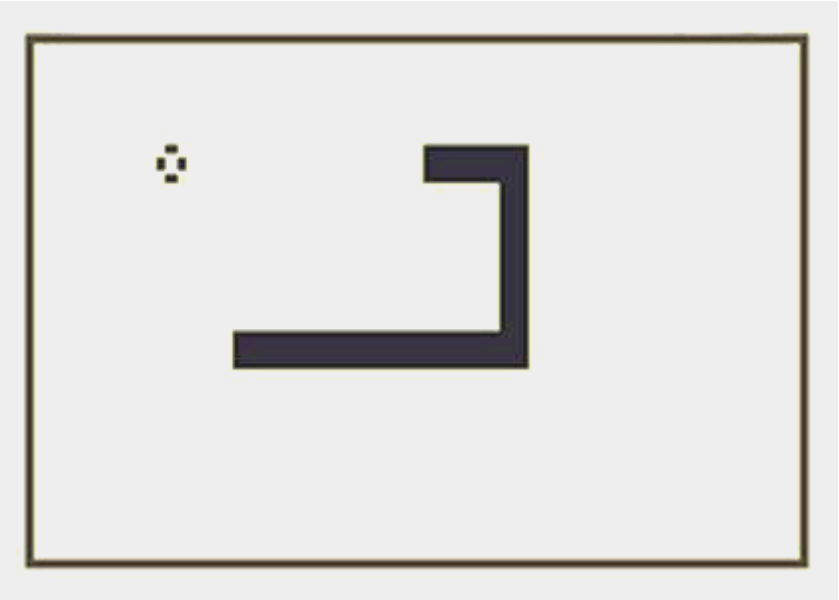

Ilustración 1: captura de pantalla del clásico snake

El contenido de esta memoria aúna todas las cuestiones de diseño e implementación que nos hemos planteado, la arquitectura empleada, las conclusiones y las líneas de trabajo futuras.

[1] 70´s Snake: https://en.wikipedia.org/wiki/Snake_%28video_game%29
[2] Sitio Web Raspberry Pi: https://www.raspberrypi.org/

2. Motivación y objetivos

Los videojuegos forman y han formado parte de nuestras vidas prácticamente desde siempre. Se trata de una industria que mueve millones de dólares al año no sólo en nuestro país, sino en la mayoría de países del mundo. El interés de esta industria no se centra exclusivamente en el desarrollo de los videojuegos, sino en el desarrollo de plataformas específicas para hacerlos funcionar. Estas plataformas van desde sistemas de escritorio (videoconsolas) hasta sistemas portátiles. Todas ellas cuentan con un hardware propio de los sistemas empotrados, como vemos en los siguientes dos ejemplos:

Nintendo 3DS[3]:

- **CPU:** ARM11 Dual Core @ 266Mhz
- **GPU:** Pica200 DMP @ 400 Mhz
- **RAM:** 2x64MB FCRAM @ 4.2 GB/seg
- **VRAM:** 4 MB
- **Otras características:** pantalla táctil 3D, comunicación inalámbrica (WIFI 802.11b, 802.11 g), 3 cámaras con posibilidad de hacer fotos en 3D, ranura para tarjetas SD...

PS4[4]:

- **CPU:** AMD APU Jaguar 2xQuad Core @ 1,6Ghz
- **GPU:** AMD Radeon 7870 @ 800Mhz
- **RAM:** 8GB GDDR5 de acceso hUMA @ 176 GB/seg
- **Otras características:** BluRay, Ethernet (10BASE-T, 100BASE-TX, 1000BASE-T), WIFI (802.11 b/g/n), Bluetooth 2.1 (EDR), USB 3.0...

Como vemos, cuentan con un hardware en el que prima el rendimiento (varios módulos de memoria, con un gran ancho de banda; GPU específica para los gráficos, conectividad de alta velocidad…) y el consumo energético (procesadores ARM y APU de AMD). Sin embargo, aún no hemos llegado al punto de necesitar varios sistemas que trabajen de forma distribuida para ofrecer una experiencia de juego de alta calidad. Esta ha sido la motivación de nuestro proyecto.

Los objetivos del proyecto son:

- Desarrollar el clásico **videojuego** *Snake,* de forma que puedan jugar 2 jugadores.
- Emplear un sistema heterogéneo compuesto, al menos, por **2 placas diferentes.**
- Redactar una **memoria** que contenga los detalles de diseño e implementación del sistema y su arquitectura.

[3] Nintendo 3DS: https://es.wikipedia.org/wiki/Nintendo_3DS
[4] PlayStation 4: https://es.wikipedia.org/wiki/PlayStation_4

3. Arquitectura hardware

Antes de entrar de lleno en cuestiones de diseño e implementación, vamos a tratar el *hardware* que compone nuestro sistema para tener una visión global. En primer lugar, vamos a hablar de las placas utilizadas y de los periféricos que hemos utilizado de las mismas y, por último, la interconexión entre todas ellas.

3.1 Placas y periféricos

Hemos optado por utilizar dos maletines con la ya famosa placa S3CEV40. En un principio pensamos utilizar solamente 2 maletines, pero, debido a problemas de comunicación entre los maletines con las Uarts, decidimos incluir una segunda placa para que actuara como sistema intermediario, a modo de *switch* muy simplificado: la Raspberry Pi 2. En el *"Apartado 3.2 Interconexión entre componentes"* presentamos un esquema general de la conexión entre estas placas.

Utilizamos los siguientes periféricos del **maletín**:

- **Pulsadores**, configurados por interrupciones. Su funcionalidad consiste en iniciar la partida cuando los 2 jugadores están listos. Esto es, los maletines están correctamente conectados a la Raspberry Pi 2. El jugador con identificador 1 es el encargado de iniciar la partida.
- **Teclado matricial,** configurado por interrupciones. Su funcionalidad es mover a la serpiente en las 4 direcciones posibles: arriba, abajo, derecha e izquierda.
- **Timer 0.** Cuando expira, genera una fruta (la misma) en ambos maletines.
- **Display LED de 8 segmentos.** Muestran la puntuación de los jugadores (de 0 a F).
- **LEDS:** Cuando los maletines están sincronizados (la Raspi ha enviado los identificadores) muestran el identificador, 1 ó 2.
- **Uart1.** Sirven para las comunicaciones entre cada maletín y la Raspberry Pi 2.

De la **Raspberry Pi 2** utilizamos los siguientes periféricos:

- **Puerto Ethernet.** Sirve para conectar la Raspberry a un PC por SSH o TELNET para iniciar el programa que gestiona la comunicación entre los maletines.
- **Puertos USB.** Sirven para la conexión con cada uno de los maletines gracias a un adaptador de Serial a USB.

3.2 Interconexión entre componentes

Como muestra la Ilustración 2, los maletines están conectados por USB a un ordenador cada uno para cargar el programa en C en las placas. Por otro lado, la Raspberry Pi 2 está conectada a un ordenador portátil, que mediante acceso SSH a la Raspi, ejecutará en ella el código en *Python* que hará que la Raspi haga de intermediario entre los maletines. Las conexiones requieren de los siguientes cables:

- **Placa S3CEV40 - PC**: cable USB suministrado en el maletín.
- **Placa S3CEV40 - Raspberry Pi 2:** cable adaptador Puerto Serie a USB. **Protocolo** IIC[5].
- **Raspberry Pi 2 - PC:** cable de Red Ethernet. **Protocolo** SSH[6].

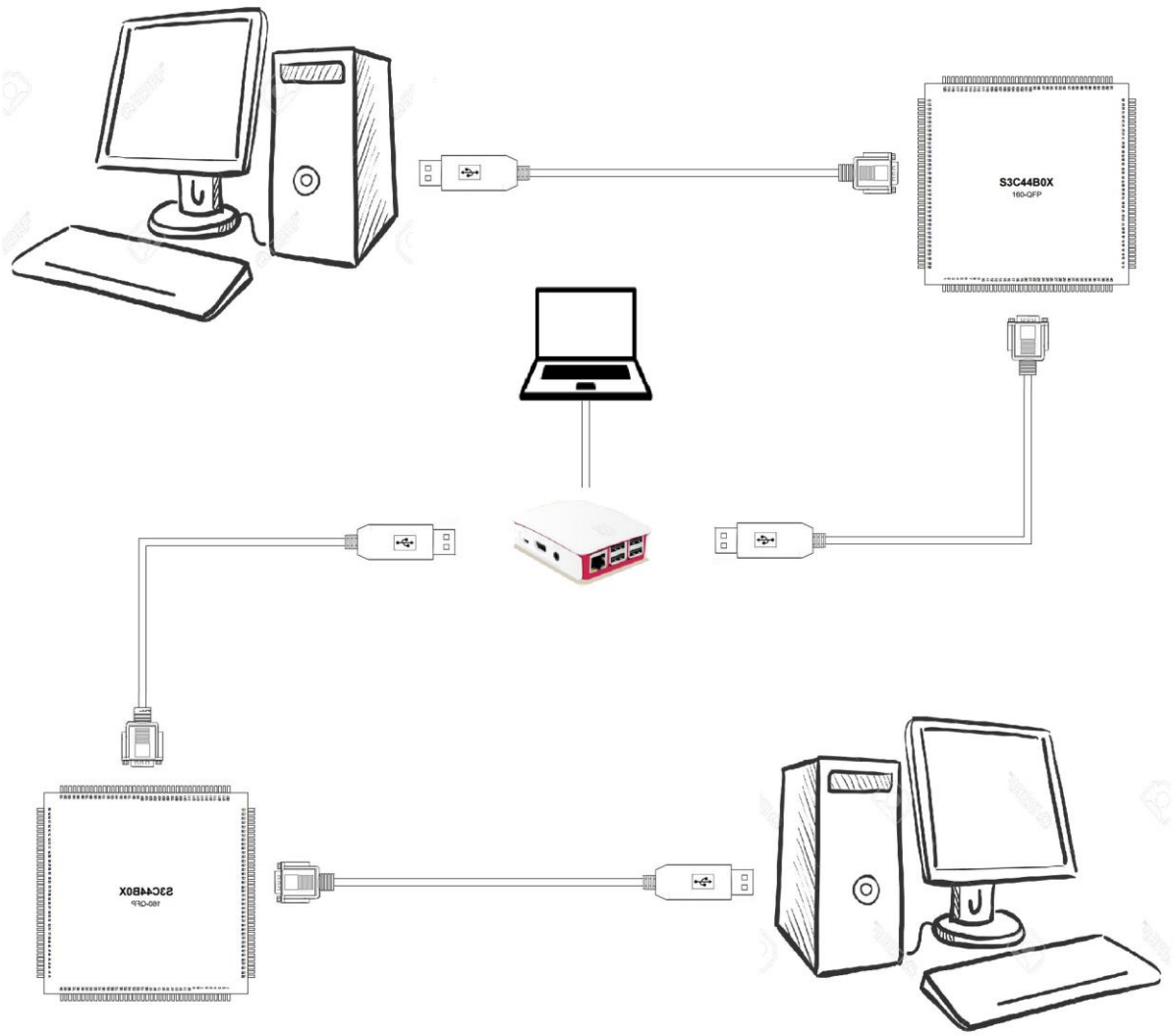

Ilustración 2: Interconexión entre componentes

[5] IIC: http://i2c.info/
[6] SSH Protocol: http://www.cisco.com/c/en/us/about/press/internet-protocol-journal/back-issues/table-contents-46/124-ssh.html

A continuación se muestra una fotografía real de esta arquitectura:

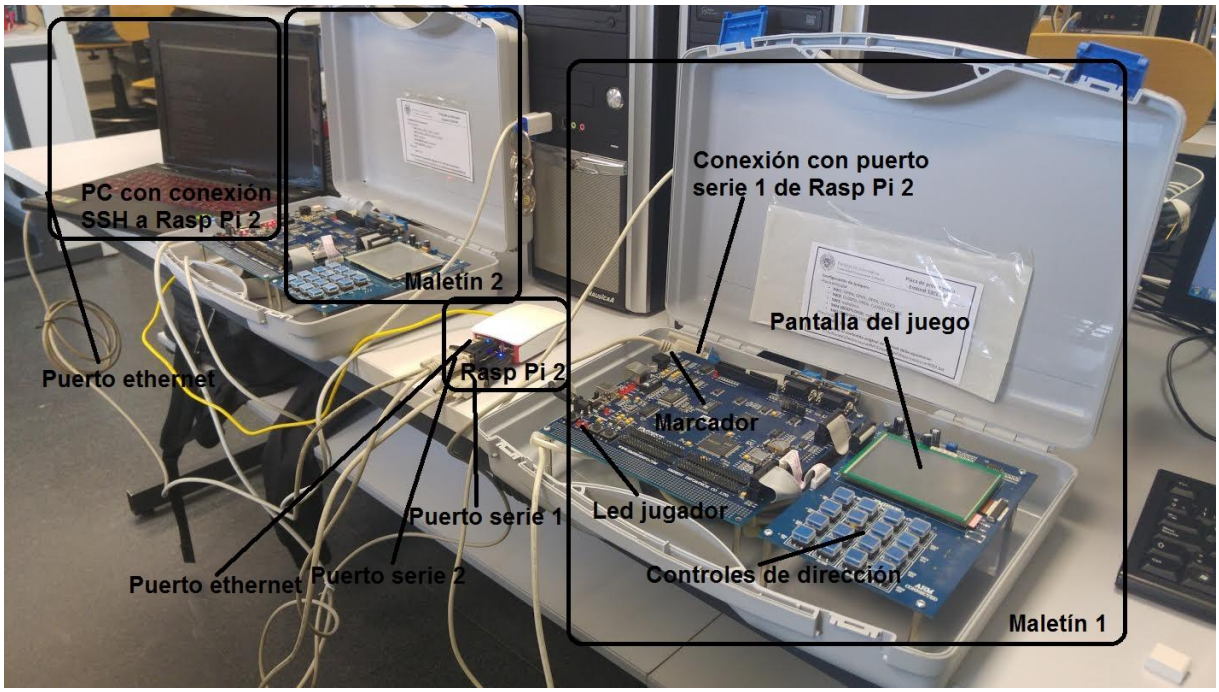

Ilustración 3: Fotografía de la arquitectura del sistema

4. Diseño

En este apartado vamos a tratar las cuestiones en cuanto al diseño del sistema. Hemos optado por crear diagramas UML de estado y de secuencia, junto con una breve explicación para aportar la máxima información.

4.1 Diagramas de estados

Ambos maletines comparten el mismo código en C, pero distinguen a su jugador del otro por un identificador dado por la Raspberry Pi 2. Además, la Raspberry Pi 2 será la encargada de intercambiar los mensajes entre los maletines.

Lo primero que hacen los maletines es inicializar los dispositivos que utilizan: botones, Uarts, Display de 8 segmentos, LEDs, pantalla LCD y teclado matricial. Luego esperan el identificador de jugador dado por la Raspi y, una vez lo tienen, lo muestran en los leds y limpian la pantalla LCD. Acto seguido el maletín con identificador 2 espera a que el maletín con identificador 1 pulse el botón izquierdo. Una vez lo pulse, se inicializará el juego con todas sus fases (pintar, actualizar, calcular colisiones, enviar señales...). El juego acaba cuando una de las serpientes colisiona con un muro o con la otra serpiente. En ese momento, se actualiza la puntuación en el Display de 8 segmentos y se vuelve a esperar a que el maletín con identificador 1 pulse el botón.

Por cuestiones de legibilidad, primero vamos a mostrar los diagramas de estados por separado de los maletines y de la Raspberry Pi 2. Después, mostraremos el diagrama de estado conjunto en el que las comunicaciones entre ambas FSMs se representan con flechas azules.

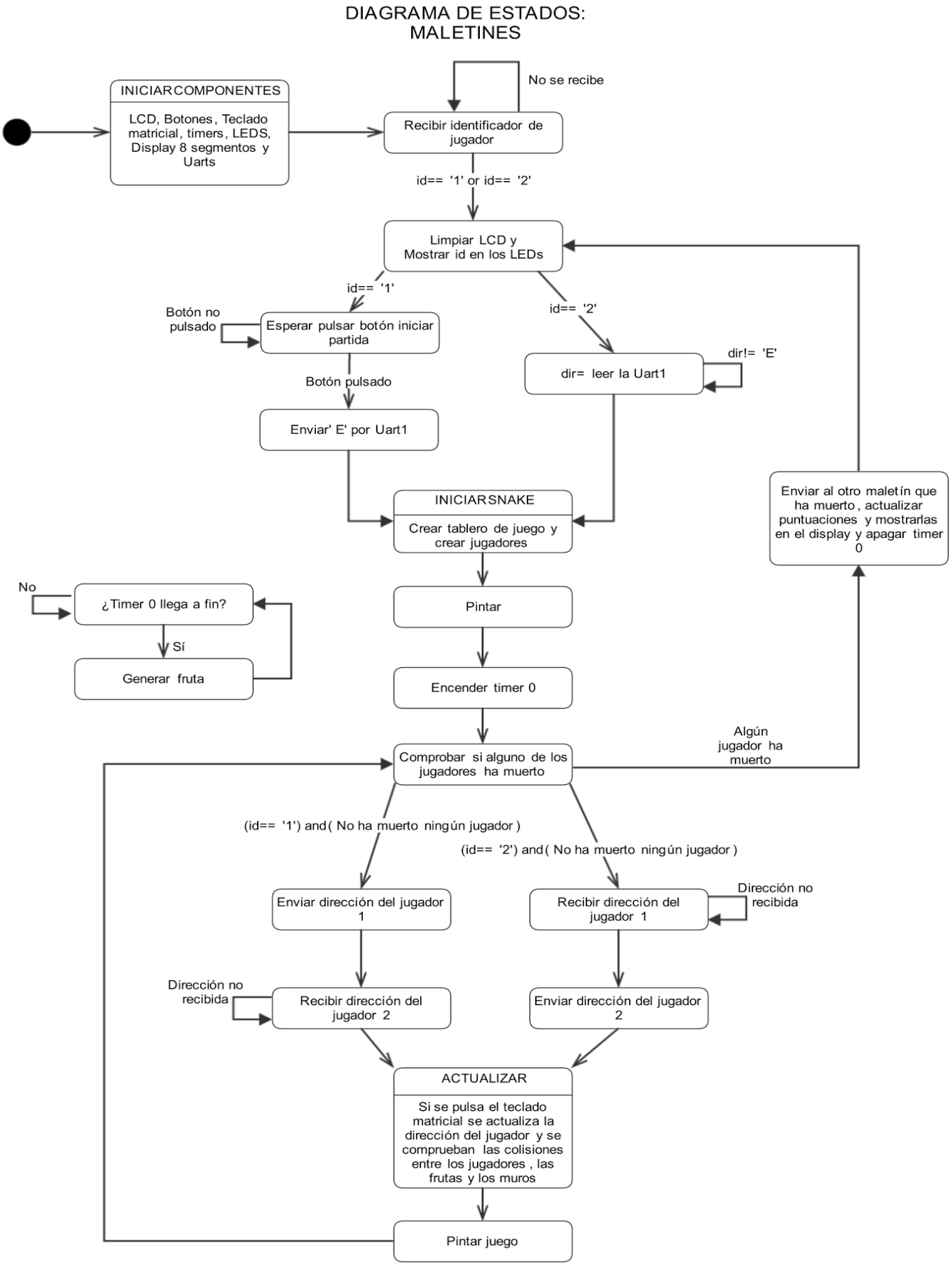

Ilustración 4: Diagrama de estados de los maletines

En segundo lugar, el diagrama de estados de la Raspberry Pi 2:

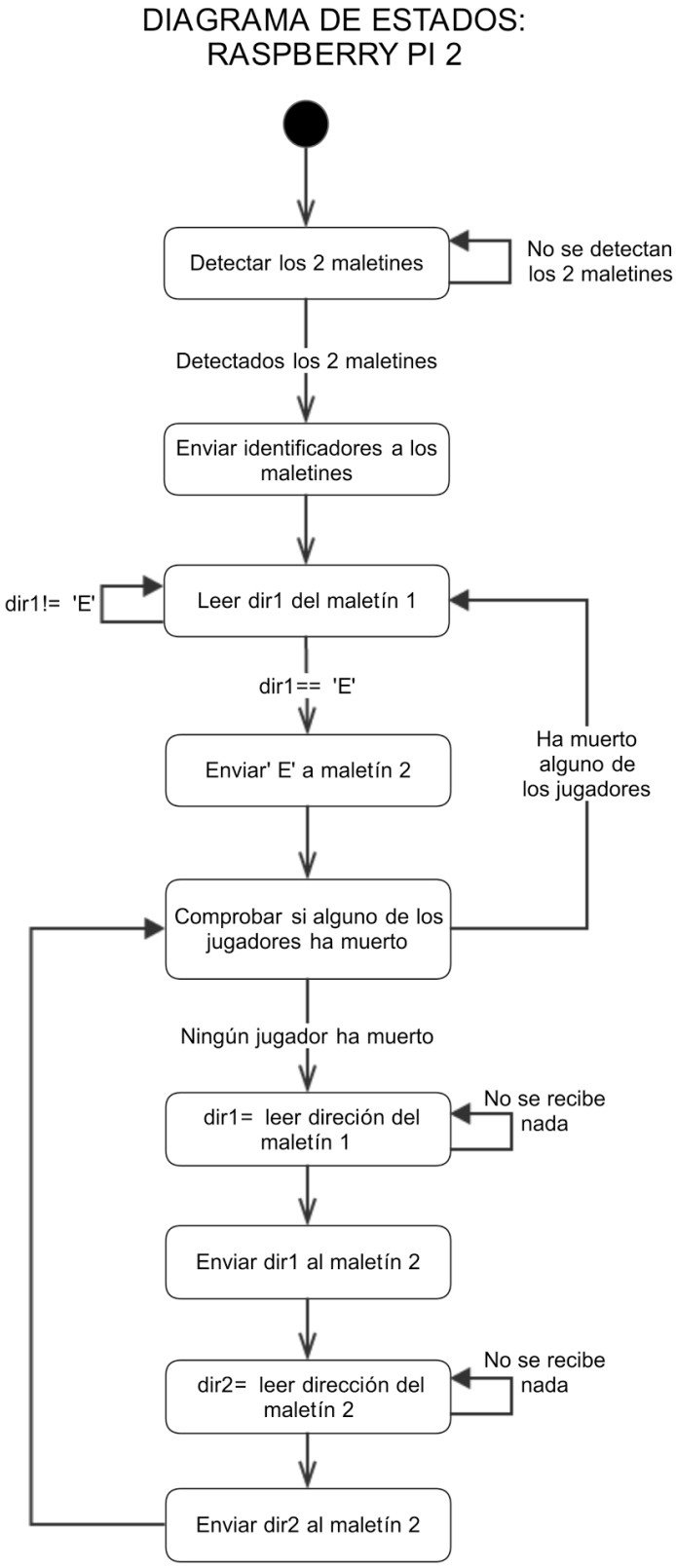

Ilustración 5: Diagrama de estados de la Raspberry pi 2

Y, por último, el diagrama con ambas FSMs y su interconexión:

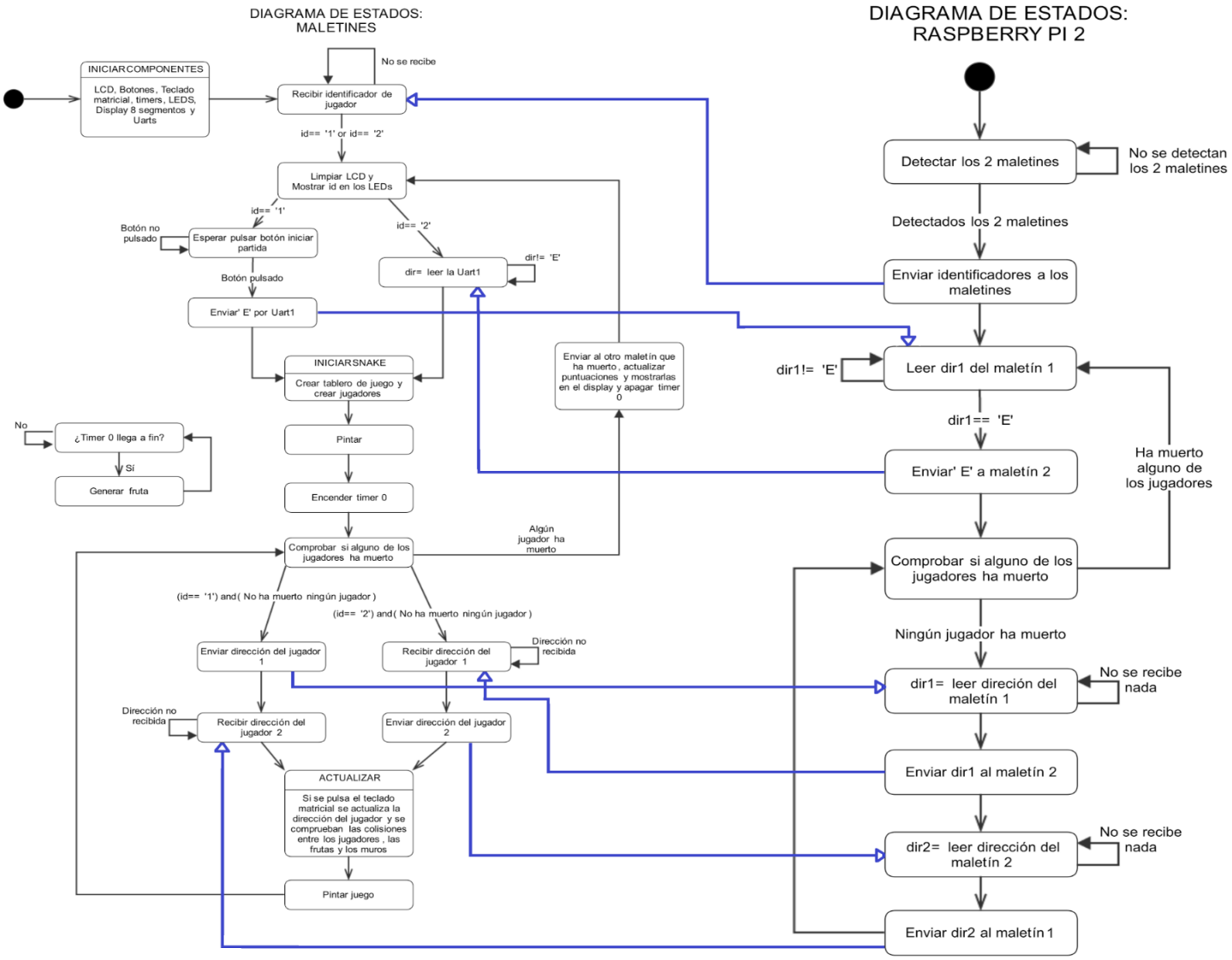

Ilustración 6: Diagrama de estados global

4.2 Diagramas de secuencia

A continuación, presentamos el diagrama de secuencia del sistema que representa las comunicaciones. Hemos abstraído el diagrama para N maletines y N jugadores con 1 Raspberry Pi 2 pero, como hemos explicaremos más adelante, nuestro prototipo se ha centrado en el caso de N = 2.

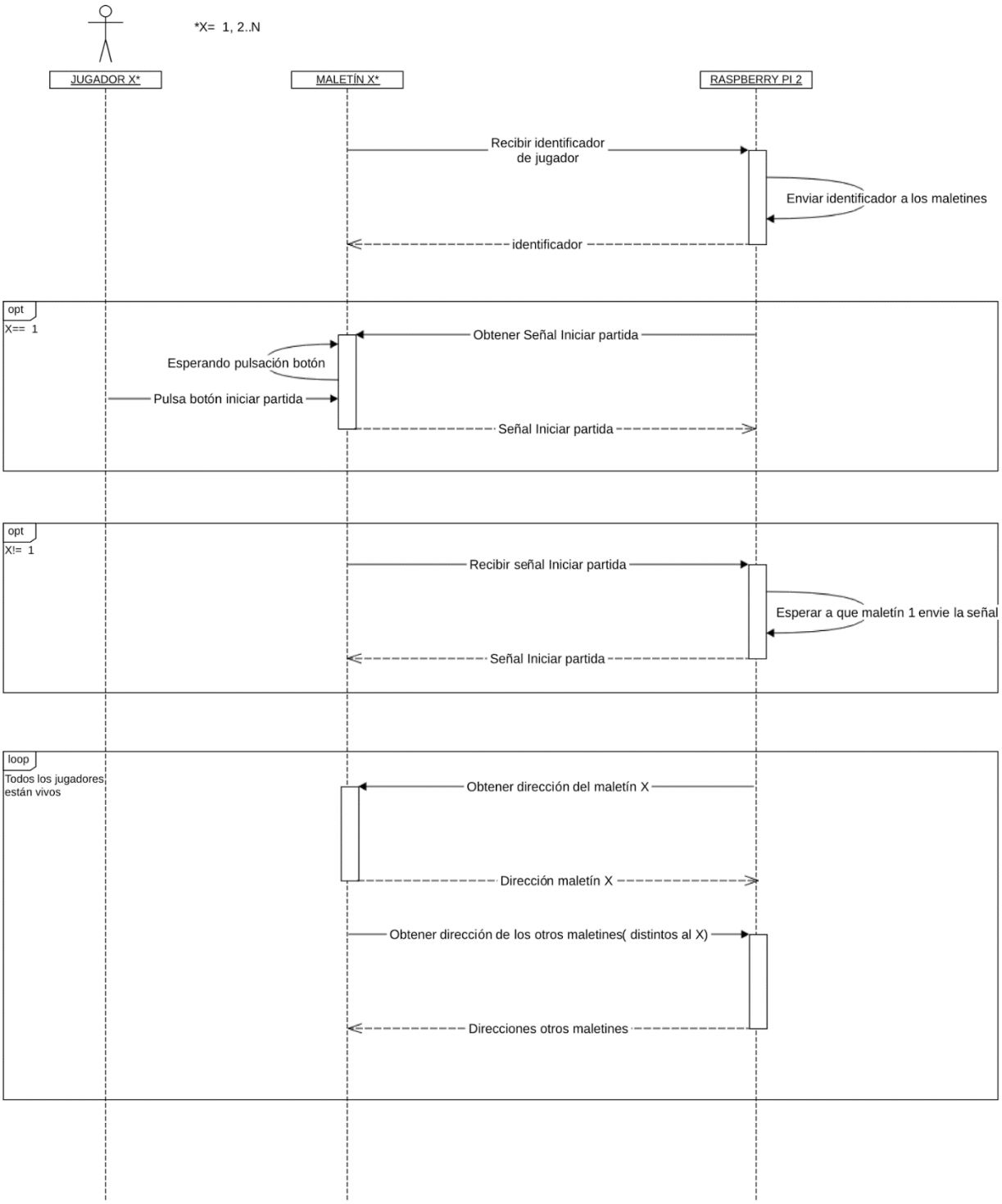

Ilustración 7: Diagrama de secuencia

5. Implementación

En este apartado vamos a tratar las cuestiones de implementación. Consta de 3 sub-apartados que corresponden con los prototipos que hemos ido desarrollando.

5.1 Primer prototipo funcional

La primera versión funcional consta de dos maletines programados en lenguaje C conectados por la UART1 (véase el Apartado 3.1 Placas y periféricos). Cada uno de ellos ejecutan una máquina de estados cíclica cuyos casos de uso son los siguientes:

1. Inicializar tablero (incluyendo muros y posición inicial del jugador)
2. Enviar jugador al otro maletín.
3. Obtener jugador del otro maletín.
4. Actualizar posición del jugador.
5. Actualizar posiciones del otro jugador en el tablero.
6. Generar fruta en tablero.
7. Calcular colisiones.
8. Realizar el pintado del tablero entero.

** Los puntos 2-8 son repetidos de forma independiente en cada maletín hasta que se produce una colisión por parte de una de las dos serpientes.*

Puesto que la transmisión de cada una de las posiciones de la serpiente por la UART es de unos 70ns, sumado a que pintar un nuevo fotograma de la LCD lleva unos 800 ms; estamos hablando de un tiempo de actualización cercano al segundo. Un tiempo excesivamente grande para un juego en tiempo real.

5.2 Segundo prototipo funcional

La segunda versión funcional sigue constando de dos maletines programados en lenguaje C conectados por la UART1. En este prototipo se ha mejorado significativamente los tiempos de actualización del juego, pasando de un tiempo cercano al segundo a un tiempo del orden de milisegundos.

Esta mejora significativa se basa principalmente en dos cambios respecto al primer prototipo:

- Envío por serie de la dirección de la serpiente del jugador, en vez de las posiciones ocupadas por el jugador en el tablero. Esto supone una reducción drástica del tiempo de transmisión puesto que en cada vuelta basta con enviar un byte indicando cual es la posición que actualmente tiene la cabeza de la serpiente, siendo cada uno de los maletines los responsables de calcular las posiciones ocupadas por cada una de las serpientes.

- Creación de un array de posiciones a actualizar. En este array se introducen las celdas del tablero que se han actualizado, ya sean por movimientos de las serpientes o por la creación de frutas. Esto permite que no se repinte completamente la pantalla de la LCD en cada vuelta, si no únicamente las celdas que hayan sido modificadas en la vuelta actual.

Estos cambios suponen una modificación de los casos de uso que quedan como se indican a continuación:

1. Inicializar tablero (incluyendo muros y posición inicial del jugador)
2. Realizar el pintado del tablero entero.
3. Enviar dirección del jugador al otro maletín.
4. Obtener dirección del jugador del otro maletín.
5. Actualizar posición del jugador.
6. Actualizar posición del otro jugador en el tablero.
7. Generar fruta en tablero.
8. Calcular colisiones.
9. Realizar el pintado de las celdas modificadas.

** Los puntos 3-9 son repetidos de forma independiente en cada maletín hasta que se produce una colisión por parte de una de las dos serpientes.*

5.3 Tercer prototipo funcional

La tercera versión funcional consta de dos maletines programados en lenguaje C, igual que en anteriores revisiones, a lo que se añade una Raspberry Pi 2 ejecutando un código *Python* que redirige el tráfico de un maletín hacia el otro y viceversa. Cada maletín está conectado por serie por la UART1 a uno de los puertos USB de la Raspberry Pi 2.

Esta tercera revisión mejora significativamente la fiabilidad de la comunicación entre los maletines. Por razones desconocidas la conexión directa por serie entre los maletines es inestable, interrumpiéndose de forma abrupta la recepción o envío de datos por la UART sin haber modificado el software o el hardware del sistema.

Para corregir dicha inestabilidad se añade una Raspberry Pi 2 entre los dos maletines, esta placa ejecuta un *script* programado en *Python* que hace, básicamente, las funciones

de un *switch* simplificado: redirige el tráfico de un maletín a otro y viceversa. Este cambio proporciona la estabilidad necesaria para que el juego se ejecute siempre de forma correcta y sin interrupción alguna.

En las revisiones anteriores, a pesar de que cada maletín poseía el mismo código fuente, el código compilado no era el mismo, puesto que se usaban directivas de preprocesador para indicar el número de jugador que controlaba cada maletín. En esta nueva revisión es la Raspberry Pi 2 la encargada de indicar qué maletín controla qué jugador.

Este cambio supone que el código compilado sea el mismo para los dos maletines, y esto trae consigo una gran ventaja: la escalabilidad. En esta revisión añadir más maletines para que puedan jugar más de dos jugadores es relativamente sencillo, bastaría con añadir un nuevo jugador indicando su posición inicial y dirección además de realizar pequeños cambios en el código. La mayor limitación de escalabilidad es el número de puertos USB de la Raspberry Pi 2 (posee 4 puertos USB) pero sustituyendo esta placa por otra con más puertos USB que pueda ejecutar código Python la limitación se superaría.

Respecto a los casos de uso, se añade uno respecto a la revisión anterior, encargado de obtener el número de jugador asociado al maletín que ejecuta el caso de uso. El listado se queda como se indica a continuación:

1. Obtener número de jugador
2. Inicializar tablero (incluyendo muros y posición inicial del jugador)
3. Realizar el pintado del tablero entero.
4. Enviar dirección del jugador al otro maletín.
5. Obtener dirección del jugador del otro maletín.
6. Actualizar posición del jugador.
7. Actualizar posición del otro jugador en el tablero.
8. Generar fruta en tablero.
9. Calcular colisiones.
10. Realizar el pintado de las celdas modificadas

** Los puntos 4-10 son repetidos de forma independiente en cada maletín hasta que se produce una colisión por parte de una de las dos serpientes.*

En esta revisión el juego se vuelve estable, relativamente escalable, la velocidad del juego es la correcta y, en definitiva, la experiencia del usuario al interactuar con el sistema es satisfactoria.

A continuación se muestran algunas capturas de pantalla de este tercer prototipo en funcionamiento:

Ilustración 8: Pantalla de inicio del jugador 1

Ilustración 9: Pantalla de inicio del jugador 2

Ilustración 10: Captura de una partida ya iniciada

6. Conclusiones

Partiendo de un hardware muy limitado se ha conseguido desarrollar un videojuego en tiempo real capaz de exprimir los recursos de los maletines ARM. Hacer un juego no es tarea sencilla, ya que se trata de un sistema en el que constantemente se actualizan los datos del juego en función de la entrada de los jugadores y del resto de elementos del mismo. Procesamiento que tiene que hacerse en tiempo real para proporcionar una experiencia de juego aceptable. Este es el motivo por el cual la mayor parte del esfuerzo del proyecto ha estado centrado en la optimización del rendimiento y la mejora de la estabilidad de las comunicaciones.

Hemos tenido numerosos problemas de comunicación entre los maletines y las Uarts. Después de cambiar varios maletines y cables tomamos la decisión de incluir la Raspberry Pi 2 en el sistema; y no podíamos haber hecho algo mejor. La comunicación entre los maletines y la Raspberry fue a la primera y sin errores graves, además de ser rápida y muy sencilla de configurar con el *script* en *Python*. Llegamos a la conclusión de que existe algún tipo de problema de comunicación entre los maletines utilizando las Uarts.

Como ya hemos comentado, nuestro mayor esfuerzo se ha centrado en optimizar el código de los maletines dada la limitación de los componentes (CPU y comunicación por el puerto serie) y los requerimientos del *hardware* que ejecuta videojuegos. Hemos comprobado que los maletines resultan bastante limitadores, sobre todo al compararlos con la Raspberry Pi 2, y para proyectos futuros basados en SEDs seguramente tomaremos la decisión de utilizar sistemas más potentes.

Hemos desarrollado un total de 3 prototipos funcionales: el primero de ellos cuenta con todas las funcionalidades planificadas pero el tiempo de actualización entre frames es cercano a un segundo, lo que impide ofrecer una experiencia de uso satisfactoria. El segundo prototipo mejora el rendimiento de las comunicaciones y del pintado en la LCD, lo que permite reducir el tiempo de actualización a unos pocos milisegundos. Por último, el tercer prototipo mejora la estabilidad en las comunicaciones del sistema, logrando superar los requisitos necesarios para ofrecer una experiencia de uso satisfactoria al usuario.

El siguiente prototipo que hemos planificado tiene como objetivo mejorar la escalabilidad del sistema, siendo trivial añadir más maletines ARM para que puedan jugar N jugadores. A pesar de que en el tercer prototipo hemos hecho grandes avances en este aspecto (generando el mismo código compilado para cada maletín), es necesario hacer pequeñas modificaciones en el código de los maletines y en la Raspberry Pi 2. Estas modificaciones son nulas en el cuarto prototipo, detectándose en tiempo de ejecución el número de maletines ARM conectados y configurando el juego para N jugadores de forma automática.

A fecha de escritura de estas líneas no está previsto terminar este último prototipo pero, si el tiempo lo permite, es posible que a fecha de entrega de la demo final sí esté terminado; quedándose, por tanto, como posible trabajo futuro.

Podemos finalizar afirmando que la experiencia de trabajar con diferentes SE nos ha ayudado a comprender la necesidad de realizar sistemas distribuidos basados en estas pequeñas placas que, pese a tener un bajo rendimiento, en suma nos permiten realizar aplicaciones complejas a bajo coste optimizando la relación rendimiento / consumo.