UNIVERSIDAD DE SANTIAGO DE CHILE

FACULTAD DE INGENIERÍA

DEPARTAMENTO DE INGENIERÍA EN INFORMÁTICA

Videojuego “Snake”

Laboratorio Nª1, Organización de Computadores.

Integrantes: Felipe Jara

Profesor: Felipe Garay

Erika Rosas

Nicolás Hidalgo

Ayudante: Ian Mejias

Fecha de Entrega: 07/05/2015

Santiago de Chile

1 – 2015

TABLA DE CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

CAPÍTULO 1. INTRODUCCIÓN

El presente documento trata acerca de la implementación del clásico videojuego conocido como “Snake” en lenguaje de programación MIPS Assembly, desarrollado como parte de un laboratorio propuesto a los estudiantes de la asignatura Organización de Computadores de la Universidad Santiago de Chile.

A través del entorno de desarrollo interactivo (IDE) proporcionado por Mars MIPS, un software basado en Java necesario para simular la ejecución de programas en lenguaje MIPS Assembly, se reproduce la experiencia de controlar una delgada serpiente a lo largo de un área de juego en la cual existen obstáculos creados aleatoriamente y alimento que puede ser comido por éste. Cada vez que la serpiente logra comer el alimento preparado en el mapa, su cuerpo se va alargando y la dificultad del juego va aumentando. El juego finaliza cuando la serpiente golpea contra algún obstáculo, ya sea una pared o una parte de su mismo cuerpo.

CAPÍTULO 2: MARCO TEÓRICO

Antes de empezar, es preciso aclarar que el desarrollo de este programa considera exclusivamente los conceptos fundamentales a la programación en lenguaje Assembly para procesadores MIPS de arquitectura RISC, la cual se caracteriza por el uso de una limitada cantidad de formatos de instrucciones, además de treintaydos registros de propósito general cuya forma de uso esta determinada por convenciones de software y, a veces, por el mismo hardware.

2.1 REGISTROS DE PROPOSITO GENERAL

Un registro en MIPS RISC corresponde a una parte del procesador capaz de contener un patrón de 32-bits. El procesador cuenta con una variedad de registros, de los cuales solo algunos son visibles para el lenguaje Assembly. Estos últimos son conocidos como registros de propósito general y corresponden a aquellos registros que comúnmente se utilizan para programar en lenguaje ensamblador.

En la siguiente tabla se describen todos los registros que fueron relevantes en el desarrollo del programa solicitado.

Tabla 2.1: Registros generales utilizados

Nombre	Número	Uso
$zero	$0	Este registro es "cero" permanentemente, esto significa que está configurado para contener siempre el valor 0x00000000 (todos los bits a cero).
$v0–$v1	$2–$3	Estos registros se utilizan para los valores que son retornados por una subrutina. Por lo que no son preservados al ser llamados a las subrutinas y son modificables en ellas.

$a0–$a3	$4–$7	Estos registros guardan los argumentos para las subrutinas. Tal como la convención de software lo indica, estos registros pueden ser modificados dentro de las subrutinas.
$t0–$t7 y $t8-$t9	$8–$15 y $24-$25	Estos son registros temporales. Según la convención es posible modificar estos registros en una subrutina o fuera de ella ya que se supone que los valores que devuelve no se vuelven a utilizar durante el programa
$s0–$s7	$16–$23	Estos son los registros guardados. Según la convención de software se dice que estos registros no deben ser modificados al ingresar a una subrutina. Por esta razón se debe guardar la información antes de llamarse a la subrutina.
$ra	$31	Registro para guardar la dirección de retorno de un procedimiento.

2.2 SERVICIOS DEL SISTEMA (SYSCALL)

Mars MIPS pone a disposición del programador una serie de servicios del sistema, principalmente para la entrada y salida de datos. Los servicios utilizados por el programa desarrollado se encuentran enlistados a continuación:

Tabla 2.2: Servicios del sistema utilizados

Servicio	Código en $v0	Argumentos	Resultados
Imprimir entero	1	$a0 = entero a imprimir	
Imprimir string	4	$a0 = dirección del string a imprimir	
Leer entero	5		$v0 contiene el entero leído

Salir (terminar ejecución)	10		
sleep	32	$a0 = Cantidad de tiempo en milisegundos de "congelamiento".	Provoca que el thread de Mars Java se "congele" por (al menos) el numero de milisegundos especificado.
Entero aleatorio con limite superior	42	$a0 = Cualquier entero $a1 = Limite superior del rango del valor retornado	$a0 contiene un valor entero entre el rango de 0 <= entero < limite superior, obtenido a partir de una secuencia generadora de números aleatorios.

2.3 BITMAP DISPLAY

El IDE de Mars implementa Bitmap Display, una herramienta que permite escribir pixeles, con colores en formato RGB 0xRRGGBB, en una dirección de memoria predeterminada.

La herramienta entregada permite configurar la dimensión tanto de la unidad de los pixeles como la del Display, además de especificar la dirección base de éste.

2.4 KEYBOARD AND DISPLAY MMIO SIMULATOR

El IDE de Mars implementa un simulador de teclado, una herramienta que permite recuperar, en formato ASCII, la última tecla que fue ingresada en el simulador. La idea es leer periódicamente una dirección de memoria y revisar si se ha presionado una tecla. Se lee la dirección de memoria 0xffff0004, en donde se encuentra el último carácter leído por el teclado.

CAPÍTULO 3: DESARROLLO

Para el desarrollo del proyecto se emplea la metodología de resolución que implica dividir recursivamente un problema complejo hasta encontrar sub-problemas, relativamente, sencillos de resolver y, a partir de las soluciones individuales de estos, encontrar la solución del problema inicial (División en sub-problemas). Cada uno de los sub-problemas identificados son descritos a continuación, en sus respectivas secciones.

3.1 GENERACIÓN ALEATORIA DE OBSTACULOS

El programa debe ser capaz de generar obstáculos en posiciones aleatorias del terreno de juego, incluyendo ciertas restricciones en el procedimiento para evitar la construcción de caminos que impidan a la serpiente poder llegar a todas partes del mapa.

Para alcanzar ese objetivo, se recurre a uno de los servicios entregados por el sistema (servicio 42) para generar un número aleatorio perteneciente al siguiente rango:

$$[\ 4 * (\mathrm{Px}_{\mathrm{scorebar}} - 1) * (\mathrm{Py}_{\mathrm{scorebar}} - 1),\ 4 * (\mathrm{Px}_{\mathrm{display}} - 1) * (\mathrm{Py}_{\mathrm{display}} - 1)\ ]$$

$Px_{scorebar} = Posición\ X\ del\ ultimo\ pixel\ que\ ocupa\ la\ barra\ de\ puntuación\ en\ el\ Display\ grafico.$

$Py_{scorebar} = Posición\ Y\ del\ ultimo\ pixel\ que\ ocupa\ la\ barra\ de\ puntuación\ en\ el\ Display\ grafico.$

$Px_{display} = Posición\ X\ del\ ultimo\ pixel\ que\ ocupa\ el\ Display\ grafico.$

$Py_{display} = Posición\ Y\ del\ ultimo\ pixel\ que\ ocupa\ es\ Display\ grafico.$

Como en este caso se trabaja sobre un Display de 64x64 bits y una barra de puntuación de 64x7 bits, el rango queda expresado como [1512, 15.876]. .

Continuando con la idea, el número generado por el servicio pasa a ser utilizado como el *offset* del registro base del Display, de esta manera se accede a la dirección de memoria del pixel que se intenta pintar como obstáculo. Sin embargo, para poder pintar un pixel como obstáculo, primero deben cumplirse las siguientes condiciones:

- Un pixel puede ser pintado como obstáculo si en este no se ha pintado antes algún otro elemento del juego, es decir, si el pixel seleccionado se encuentra pintado con el color base del mapa.

- Siendo (x, y) la posición del pixel considerado, este no puede ser pintado como obstáculo si hay otro obstáculo pintado en cualquiera de las siguientes posiciones: $(x + 1, y)$, $(x - 1, y)$, $(x, y + 1)$ y $(x, y - 1)$. Esto para evitar la construcción de caminos imposibles de alcanzar por la serpiente.

Cuando un obstáculo no puede ser agregado, la función encargada de generarlos devuelve un indicador que señala la situación ocurrida (retorna el valor -1), el cual facilita la generación de una determinada cantidad de obstáculos en el mapa al estar constantemente verificando si un pixel fue pintado o no como uno de éstos.

3.2 GENERACIÓN DEL ALIMENTO Y ACTUALIZACION DE LA PUNTUACIÓN

Al principio del juego y cada vez que la serpiente alcanza la posición de una comida, debe aparecer un nuevo alimento en una posición aleatoria del mapa. La generación de éste se hace de forma similar a como se generan los obstáculos en el mapa, por lo que no es necesario explicar el procedimiento una segunda vez.

Además, cada vez que la serpiente alcance la posición de un alimento debe actualizarse la puntuación actual del juego, la cual está guardada como variable en la sección .data del programa, y desplegarlo en la barra de puntuación presente en el Display.

Separando la puntuación actual en unidad y decena, es posible aplicar procedimientos para incluir tales valores numéricos en la barra de puntuación generada en el Display.

3.3 MOVIMIENTO DE LA SERPIENTE

Para llevar a cabo el movimiento de la serpiente es necesario tener en consideración tres variables relevantes: la posición (x, y) de la cabeza de la serpiente, la posición (x, y) de la cola de la serpiente y un arreglo de movimientos que debe seguir la cola de la

serpiente. Además de otras dos variables que indican en que posición se encuentra la ultima dirección que fue incluida en el arreglo de movimientos y en que posición del arreglo se encuentra el movimiento que la cola de la serpiente esta considerado en un instante en particular.

Inicialmente, los componentes x de ambas posiciones deben tener valores diferentes, mientras que los componentes y deben ser igual valor. En otras palabras, se debe cumplir con las siguientes condiciones:

1) $X_{cabeza} < X_{cola}$

2) $Y_{cabeza} = Y_{cola}$

Teniendo en consideración las posiciones de ambas partes de la serpiente en el mapa de juego, se traza una línea de un determinado color desde la cabeza hasta la cola, representando de esta manera el cuerpo de la serpiente.

Figura 3-1: Representación grafica de la serpiente.

Además, la cola de la serpiente debe realizar en un principio $d = X_{cabeza} - X_{cola}$ movimientos hacia la derecha. En otras palabras, debe agregarse tal cantidad de direcciones en el arreglo de movimientos que debe seguir la cola de la serpiente a lo largo del juego.

3.3.1 Movimiento de la cabeza de la serpiente

El movimiento de la cabeza de la serpiente depende de las teclas ingresadas en el simulador de teclado proporcionado por el IDE de Mars, el cual esta implementado de manera que realice una constante operación de consulta para poder obtener el último carácter leído desde el teclado (operación de Polling).

Dependiendo del movimiento escogido, se calcula las nuevas coordenadas en la cual se pretende posicionar a la serpiente y se realizan los siguientes pasos:

1. Se verifica si el pixel destinado se encuentra pintado del color de un obstáculo. Si lo esta, quiere decir que la serpiente ha chocado con un obstáculo, por lo que el juego debe ser finalizado.

2. Se verifica si el pixel destinado se encuentra pintado del color de la serpiente. Si lo esta, quiere decir que la serpiente ha chocado con su cuerpo, por lo que el juego debe ser finalizado.

3. Se verifica si el pixel destinado se encuentra pintado del color de un alimento. Si lo esta, quiere decir que la serpiente ha logrado comer un alimento, por lo cual debe generarse una nueva fruta en el mapa y actualizarse la puntuación actual del juego.

En el caso de que el movimiento seleccionado no implique la finalización del juego, se pinta el pixel de destino del color de la serpiente, se agrega la dirección escogida en el arreglo de movimientos que debe seguir la cola y se actualiza la variable que indica la posición en el arreglo del ultimo movimiento incluido.

3.3.2 Movimiento de la cola de la serpiente

La cola de la serpiente no se desplaza si la serpiente logra comer un alimento después de haber realizado el movimiento de la cabeza. Esto permite simular el efecto de crecimiento de la serpiente.

En el caso contrario, se pinta del color base el pixel referenciado por las actuales coordenadas de la cola, se obtiene desde el arreglo de movimientos la dirección que debe tomar la cola, se asigna la nueva posición de la cola de la serpiente y se actualiza el índice que indica el siguiente movimiento en el arreglo que debe seguir la cola.

Básicamente, el movimiento de la serpiente se realiza de la siguiente manera:

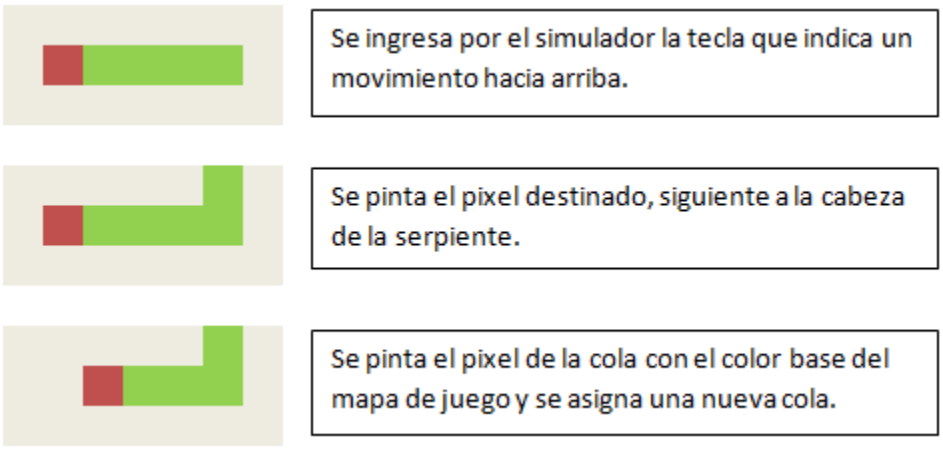

Figura 3-2: Esquema básico con respecto al movimiento de la serpiente.

CAPÍTULO 4: CONCLUSIÓN

A modo de conclusión, se puede señalar que se logro cumplir con la resolución del problema propuesto. El programa desarrollado permite simular correctamente el clásico juego de "Snake" en el simulador de Mars MIPS, generando obstáculos en posiciones aleatorias del mapa, desplegando por pantalla la puntuación del jugador y cumpliendo con otras condiciones mencionadas en el enunciado del laboratorio.

A pesar de haber cumplido con lo solicitado, se estima que el programa desarrollado podría no cumplir con alguna de las convenciones para el desarrollo de software en MIPS debido a una falta de experiencia en el lenguaje de programación utilizado. Además, se debe tener en cuenta la posibilidad de hacer más eficiente el código del programa al utilizar otro tipo de registros como los del puntero global ($gp = global pointer) y del puntero de pila ($sp = stackpointer), los cuales no fueron considerados hasta después de haber finalizado el laboratorio propuesto.

La implementación de este programa ha servido de gran ayuda para la compresión de como se debe ser analizado un problema que se pretende implementar en un lenguaje de bajo nivel (lenguaje ensamblador) con una cantidad limitada, pero suficiente, de recursos.

CAPÍTULO 5: REFERENCIAS

Wikitronica. [01/05/2015]. Registro del camino de datos del MIPS. Recuperado de: http://wikitronica.labc.usb.ve/index.php/Registros_del_camino_de_datos_del_Mips

Universidad Santiago de Chile [01/05/2015]. Tutorial de MARS: Bitmap Display, Nicolas Hidalgo y Erika Rosas (Noviembre de 2013). Recuperado de: http://www.usachvirtual.cl/moodle/file.php/2753/Laboratorio/apuntes/tutorial_display.pdf

Universidad Santiago de Chile [01/05/2015]. Juegos en MIPS, Felipe Garay (Diciembre de 2013). Recuperado de: http://www.usachvirtual.cl/moodle/file.php/2753/Laboratorio/apuntes/juegos_mips.pdf

SIGCSE 2006 paper, "MARS: An Education-Oriented MIPS Assembly Language Simulator,"Kenneth Vollmar and Pete Sanderson. ACM SIGCSE Bulletin, 38:1 (March 2006), 239-243. Recuperado de: http://courses.missouristate.edu/KenVollmar//mars/fp288-vollmar.pdf