PROYECTO TETRIS

ESTRUCTURA Y TECNOLOGÍA DE COMPUTADORES

DAVID FERNÁNDEZ EXPÓSITO DNI: [redacted]

PABLO TADEO ROMERO ORLOWSKA DNI: [redacted]

Archivos entregados: En el archivo entregado, se puede encontrar esta memoria "Memoria.pdf" y el código fuente de la práctica en el archivo "proyecto tetris.s".

En esta memoria se describen los detalles más significativos de la realización del proyecto TETRIS de prácticas en ensamblador.

PROBLEMAS ENCONTRADOS:

El principal problema ha sido, debido a la situación actual, el tener que hacer el proyecto a distancia, sin poder tener contacto directo entre los dos integrantes del grupo ni con los profesores. Se ha intentado que esto afecte lo menos posible, contactando con los profesores cuando surgía alguna duda mediante el Aula Virtual, y haciendo uso de herramientas que nos permitieron hacer el proyecto en grupo.

En cuanto al proyecto en sí mismo, no surgieron grandes problemas. Quizás destacar la utilidad de los "breakpoints", ya que al implementar el procedimiento "eliminar_lineas" se nos olvidó poner la instrucción jr $ra al final y el comportamiento del juego era incorrecto. Haciendo uso de esta herramienta, pudimos ver, ejecutando una a una cada instrucción después del "breakpoint", que, efectivamente, se ejecutaba código de otro procedimiento al no tener el jr $ra.

EJERCICIOS DE TRADUCCIÓN:

En los ejercicios de traducción, debido a que es traducir un código que se nos entrega directamente en C, no hay aspectos importantes que comentar sobre cómo se realizaron los ejercicios, ya que simplemente se tradujo. Sin embargo, si que es interesante comentar cómo se probó la corrección de las traducciones en cada caso.

Método usado para probar la corrección del código:

Para probar "*imagen_set_pixel*", "*imagen_init*" y "*imagen_clean*" se usó el código mostrado en la figura 1.2 del pdf disponible en el Aula Virtual "ensamblador_proyecto.pdf". Aunque "*imagen_set_pixel*" se probó con anterioridad usando el método mostrado y explicado en el video 6 del profesor Ricardo, en el que se dibuja una L, luego se modifica un # a una 'w' y se vuelve a mostrar por pantalla la pieza modificada para comprobar que está correcto, como en *imagen_init* se usa tanto *imagen_clean* como *imagen_set_pixel* se decidió usar este código para probar los tres

ejercicios al mismo tiempo. Además, también se puede probar con el código de la figura 1.2 el ejercicio "*imagen_dibuja_imagen*", y así se hizo. Dada la gran similitud entre "*imagen_dibuja_imagen*" y "*imagen_dibuja_imagen_rotada*", se pudo usar el mismo código que se usó para probar "*imagen_dibuja_imagen*" (simplemente se dibujará al revés la pieza pasada como parámetro).

Para "*imagen_copy*", además de comprobar que funcionaba usando el visor de memoria de MARS como se nos indica en el pdf "ensamblador_proyecto.pdf", también creamos otra variable tipo imagen, en la que copiamos una de las letras usando esta función. Luego la dibujamos para comprobar que se copió correctamente.

Para "*nueva_pieza_actual*", se hizo un main en el que se llamaba a esta función y luego se dibujaba la "pieza_actual", que es la imagen que debe pasar a ser una nueva pieza tras llamar a "pieza_aleatoria".

Como "*intentar_movimiento*" es una función relativamente sencilla de traducir, se comprobó su correcto funcionamiento cuando se comprobó que "*bajar_pieza_actual*" funcionaba correctamente. Así, para probar que "bajar_pieza_actual" funcionaba correctamente, lo que se hizo es poner las coordenadas de la pieza_actual lo más abajo posible en cuanto a coordenadas en el campo, y que no se pudiese bajar más, y llamar a la función para comprobar que efectivamente fijaba la pieza en el campo y llamaba a "*nueva_pieza_actual*". También se probó el caso en el que sí se podía bajar la pieza y se observó que no se fijaba la pieza en el campo ni se llamaba a *nueva_pieza_actual.*

Para "*intentar_rotar_pieza_actual*", lo que se hizo fue poner la *pieza_actual* en una posición en la que se pudiese rotar, y así llamar a la función. Posteriormente, se comprobó que se había rotado la *pieza_actual* dibujando la pieza. También se probó el caso en el que la pieza estaba en una posición que impedía que se pudiese rotar, por lo que la *pieza_actual* no variaba.

Una vez traducidos todos los ejercicios, se comprobó el correcto funcionamiento del juego, que no producía ningún error ni warning.

EJERCICIOS DE IMPLEMENTACIÓN:

1. Marcador de Puntuación: Primero se creó una variable de tipo entero en el segmento de datos, llamada "Marcador", que se inicializa a 0 en el procedimiento

"*jugar_partida*". Un aspecto que comentar es que usamos el nombre marcador para la variable entera y el nombre puntuación para la variable en la que, como se comentará más adelante, se guarda la cadena de texto a mostrar. Sabemos que sería mejor tener los nombres al revés, pero una vez nos dimos cuenta consideramos que lo mejor era no tocar nada.

Esta variable "marcador" se actualiza en el procedimiento "*bajar_pieza_actual*", ya que en este procedimiento se "ancla" la pieza al campo si ya no puede bajar más, y es entonces cuando sumamos 1 a la variable "marcador", justo después de llamar a "*nueva_pieza_actual*".

Para mostrar la puntuación por pantalla, primero creamos en el segmento de datos la cadena str003, con el texto "Puntuación: ". También se creó un buffer llamado "puntuación", reservando 256 bytes, donde almacenar el resultado de pasar el entero del marcador a cadena. Además, se implementó el procedimiento "*imagen_dibuja_cadena*", el cual dibuja la cadena mediante un bucle y la función "*imagen_set_pixel*", y que recibe como parámetros la dirección de la imagen donde dibujaremos la cadena (en este caso, pantalla), las coordenadas donde se dibujará la cadena y la dirección de la cadena a dibujar.

Así, en el procedimiento "*actualizar_pantalla"*, antes de llamar a "*clear_screen*", se dibuja primero la cadena str003 con "*imagen_dibuja_cadena*". Luego, se llama a "*integer_to_string*" (realizado en la práctica 5) para guardar en el buffer "puntuación" la cadena con los puntos, y por último se llama de nuevo a "*imagen_dibuja_cadena*" para dibujar en pantalla el buffer con los puntos. Obviamente, hay que procurar que las coordenadas donde se dibujen las cadenas sean las adecuadas.

Comprobación: para comprobar la corrección de la implementación de esta mejora, simplemente se ejecutó el juego y se jugó una partida, en la que se comprobó que se sumaba 1 punto con cada pieza que tocaba el suelo y que se dibujaba correctamente la puntuación en la parte superior.

2. Final de la Partida: Para hacer esta implementación, primero se crearon en el segmento de datos una variable de tipo entero inicializada a 0 (y que se inicializa a false cuando se empieza la partida), llamada "*acabar_partida_campo_completo*", y

una imagen “imagen_game_over” (de ancho 19 y alto 4) con el mensaje a mostrar si se acaba la partida por este motivo.

Para comprobar que el campo está lleno, como en “*bajar_pieza_actual*” se llama a “*nueva_pieza_actual*” si no se puede bajar más la pieza, modificamos este último procedimiento para que, si la nueva pieza aleatoria generada no se puede poner en las coordenadas iniciales, no solo no lo haga, sino que además ponga a 1 la variable *acabar_partida_campo_lleno* y la variable *acabar_partida*. Para comprobar si se puede o no poner la pieza aleatoria en las coordenadas iniciales se hace uso de “*probar_pieza*”.

Para mostrar el mensaje que informe de que se ha acabo la partida porque el campo se ha llenado, se comprueba al final del procedimiento “jugar_partida” si la partida se ha acabado por que el campo está lleno o porque se ha pulsado ‘x’, comprobando el valor de *acabar_partida_campo_lleno.* Si el motivo es este, se llama a “*imagen_dibuja_imagen*” para dibujar en pantalla la imagen “imagen_game_over” en las coordenadas (1,8). Después se llama a *clear_screen* y a *imagen_print* para imprimir la pantalla, y por último se usa *read_character* para esperar a que se pulse una tecla para volver al menú.

Comprobación: para comprobar la corrección de la implementación de esta mejora, simplemente se ejecutó el juego y se jugó una partida, en la que se comprobó que, si se dejaba que se apilaran las piezas, al llegar al tope y no caber ninguna pieza más, se acababa la partida y se mostraba el mensaje deseado. Además, también se comprobó que, si se pulsaba cualquier tecla, se volvía al menú.

3. Completando Líneas: Después de copiar la pieza actual en el campo hay que comprobar qué líneas se han completado. Para ello, en “*bajar_pieza_actual*” tras haber copiado la pieza actual en el campo incluimos un bucle que vaya desde la coordenada vertical de la pieza actual hasta la coordenada final de la pieza actual (coordenada vertical de inicio + alto de la pieza). Dentro de ese bucle se hace uso de un procedimiento llamado “*comprobar_linea_llena*”, que comprueba si dicha línea está llena: si es cierto, incrementamos en 10 la puntuación.

   Para implementar “*comprobar_linea_llena”*, recorremos la línea que recibe como parámetro (el único parámetro de este procedimiento es la coordenada “y”) y mediante un bucle en el que se llama a “*imagen_get_pixel*”, si en algún momento

devuelve el píxel vacío sabremos que en esa coordenada no hay nada dibujado y por tanto "*comprobar_linea_llena"* devolverá falso, en caso contrario devolverá verdadero.

4. Eliminando Líneas: Tras sumar los 10 puntos en "*bajar_pieza_actual"* cuando detectemos que hay una línea completa, eliminamos dicha línea haciendo uso de un procedimiento llamado "*eliminar_linea*" que recibe la línea que se quiere eliminar (el único parámetro de este procedimiento es la coordenada "y"). Este procedimiento copia el contenido de la línea superior y lo pega en la línea que se quiere eliminar. Para la línea que se encuentra encima de la copiada vuelve a hacer lo mismo hasta llegar a la línea 0 (la que está más arriba) que la llena entera de píxel vacío.

   Comprobación: estas dos últimas implementaciones se han comprobado ejecutando el juego y completando las líneas de la parte inferior de la pantalla pulsando la tecla "t" (tecla "truco") para poder rellenar filas más rápidamente, y viendo cómo efectivamente se suman 10 puntos extras por cada línea completa, y que éstas se eliminan correctamente.

5. Mostrar la Siguiente Pieza: Para implementar este ejercicio deberemos modificar el procedimiento "*nueva_pieza_actual*". Para ello definiremos una nueva variable llamada "*pieza_siguiente*" de tipo imagen que siempre contendrá la siguiente pieza que va a "caer". Esta variable se inicializa al principio de "*jugar_partida*", antes de llamar a "*nueva_pieza_actual*", como una pieza aleatoria. Dentro de "*nueva_pieza_actual*" llamaremos a "*imagen_copy*" para copiar en "*pieza_actual*" la "*pieza_siguiente*". La nueva pieza que se elige aleatoriamente, en vez de copiarla en "*pieza_actual"*, se copiará en "*pieza_siguiente"*.

   Finalmente tendremos que mostrar la "*pieza_siguiente"* en la pantalla al lado del campo dentro de un recuadro. Para ello incluimos en "*actualizar_pantalla"* otros dos bucles de forma análoga a los que se ejecutan antes, pues son los que pintan los bordes del campo. No obstante, la imagen pantalla está definida inicialmente con un ancho demasiado estrecho, por eso teniendo en cuenta lo que ocupan las piezas más grandes (3 de ancho) y los bordes del recuadro aumentamos el ancho de la pantalla a 25 para asegurarnos de que quepa sin problemas la *pieza_siguiente* y su recuadro.
   Comprobación: para comprobar la corrección de la implementación de esta mejora, simplemente se ejecutó el juego y se jugó una partida, en la que se comprobó que se

dibujaba al lado del campo el recuadro y dentro de él la *pieza_siguiente*. Además, claro está, se observó que no surgiese ningún problema en la ejecución y que la siguiente pieza que se mostraba coincidía con la que realmente aparecía posteriormente.

FUNCIONALIDAD OPCIONAL: De los dos ejercicios propuestos en este apartado, solo se realizó el segundo, el que modifica el ritmo de caída según van aumentando los puntos obtenidos. Cabe destacar que se ha implementado esta mejora del juego de forma distinta a la que se indica en el pdf "ensamblador_proyecto.pdf". En este documento, se pide que cada 20 puntos la velocidad aumente (decremente el tiempo de espera, la variable pausa) en un 10%. Sin embargo, tras pensar que sería una mejor y más sencilla opción y consultar con el profesor Ricardo, se optó por reducir la variable pausa en 50 cada 20 puntos.

Para ello, creamos otra variable entera llamada "*nivel*", que se actualiza en el procedimiento "*bajar_pieza_actual*", que es donde se actualiza el marcador, de manera que la variable vale: marcador/20. Cabe destacar que, en nuestro caso, la variable entera que almacena el valor numérico de los puntos es la variable *marcador*, mientras que la variable *puntuación* es la cadena que se usará para mostrar el número por pantalla.

Luego, se cambió la constante *pausa* para que ahora fuese una variable, y ésta se actualiza en el procedimiento *jugar_partida*. La variable pausa se actualiza con la expresión: 1000 – nivel*50.

Por último, también se añadió que se mostrase por pantalla el nivel en el que se encuentra el jugador, al igual que se muestra la puntuación. Para ello, se creó una cadena *str004* ("Nivel: ") y un buffer llamado *textonivel* que se usará en la conversión del numero entero *nivel* a cadena mediante *integer_to_string*. Se dibuja la cadena *str004* y *textonivel* tras llamar a *integer_to_string* de forma análoga a la usada para mostrar la puntuación, haciendo uso de la función *imagen_dibuja_cadena.*

BIBLIOGRAFÍA UTILIZADA: solamente se hizo uso de los boletines de prácticas de la asignatura, disponibles en el Aula Virtual, y de los videotutoriales proporcionados por el profesor Ricardo.