**Practica 1;**

La práctica consiste en crear una serie de funciones y un programa principal que haga que un cuadrado se mueva por la pantalla de forma acorde a las pulsaciones W, A, S, D (Arriba, Izquierda, Abajo y Derecha respectivamente).

Esta vez, y a diferencia de las prácticas que hemos estado haciendo en clase, el movimiento no debe "dejar rastro", esto es, sólo debe verse un cuadrado a la vez en la pantalla, del color que queráis. Para ello tendréis que sobreescribir la posición actual, con el color del fondo, antes de actualizarla y escribir el nuevo cuadrado.

No es necesario hacer ninguna comprobación de límites de pantalla en este momento, pero sí tendréis que utilizar el método de pulsar - acción - soltar para poder controlar el movimiento.

**Practica 2;**

La práctica consiste en crear una serie de funciones y un programa principal que pueda pintar cualquier mapa sokoban que siga el modelo de las tablas que se encuentran en el fichero [niveles\_basicos.asm](https://ufv-es.instructure.com/courses/14202/files/1534142?wrap=1)

Los mapas están encuadrados en un cuadro de Y\*X, cuyos valores son conocidos. El programa debe hacer dos bucles *FOR* anidados que recorran la tabla y llamen a una función *pintaSokoban*por cada elemento.

Esta función será un *Switch-Case* que decidirá el color con el que pintar el cuadro y que, a su vez, podrá llamar a la función pintaXYC que ya hicimos. Se proponen los siguientes colores:

* Muro: Rojo
* Caja: Amarillo
* Personaje: Cyan (Azul claro)
* Destino: Blanco
* Fondo: Negro

**Practica 3;**

La práctica se construye sobre las anteriores y trata de controlar el movimiento del personaje **DENTRO** del mapa de Sokoban.

Recuerda que la posición inicial del personaje está definida en los parámetros de la tabla de cada nivel. Además, el bit 4 de cada cuadrado del juego, determina si el personaje está o no está.  
Al mover el personaje debes tener en cuenta **AL MENOS** los siguientes aspectos:

* Primero hay que**comprobar si el personaje se puede mover:**
  + Si el destino es un muro, no se puede mover.
  + Si el destino es una caja, de momento lo trataremos como un muro, por tanto no se puede mover.
  + Si el destino es cualquier otro, sí se puede mover.
* Decidido que es posible moverlo, debes **actualizar la posición del personaje en la tabla**:
  + cambiando la coordenada Y y la X (tercer y cuarto bytes de la tabla)
  + cambiando el bit correspondiente de la posición original del personaje, es decir, borrando ese bit 4, puesto que ahora no estará el personaje. Se recomienda usar la instrucción RES (manipulación de bits)
  + cambiando el bit correspondiente de la posición final del personaje, poniéndolo a 1. Se recomienda usar la instrucción SET.
* Por último, es necesario **pintar en pantalla los dos cuadros** con el color correspondiente, tanto en la posición origen, como en destino del personaje.

En esta práctica ya se convierte en necesario el uso de los deltaX y deltaY del teclado introducidos en las anteriores. Además, para que tenga sentido y podáis probarlo bien, es mejor usar el nivel más complejo de los dos publicados. Podéis mirar el foro de discusión de Z80 para un diagrama de este funcionamiento.

Aún así, **debe funcionar con cualquier nivel que siga la misma codificación**.

Practica 4

La práctica se construye sobre las anteriores y aborda el movimiento de las cajas y la evaluación del tablero, esto es, saber cuándo todas las cajas están en su destino.

Recuerda que la posición de las cajas está indicada por el bit 5 de cada cuadrado del juego. La posición de los destinos está indicada por el bit 6 de cada cuadrado del juego.

**APARTADO 1: MOVER LAS CAJAS**  
En la práctica anterior se trataban las cajas como si fueran paredes, pero esto no puede ser así en el juego final. Por tanto, tienes que considerar los siguientes aspectos

* Primero hay que**comprobar si la caja se puede mover:**
  + Al tener una "intención de movimiento" del personaje, con su DeltaX y DeltaY,  y su posicíón PosX y PosY, el cuadro destino será PosX+DeltaX, PosY+DeltaY. Estamos tratando el caso en el que ese destino está ocupado por una caja.
  + Hay que comprobar, entonces, la siguiente posición a la caja, el cuadro PosX+DeltaX+DeltaX, PosY+DeltaY+DeltaY. Si esa posición no es ni un muro ni una caja, entonces sí se puede mover la caja.
  + En caso contrario, no se puede mover la caja, por lo que el personaje tampoco.
* Comprobado que es posible mover la caja, debes **actualizar la posición de la caja en la tabla**:
  + cambiando el bit correspondiente de la posición original de la caja (PosX+DeltaX, PosY+DeltaY), es decir, borrando ese bit 5, puesto que ahora no estará la caja. Se recomienda usar la instrucción RES (manipulación de bits)
  + cambiando el bit correspondiente de la posición final de la caja (PosX+DeltaX+DeltaX, PosY+DeltaY+DeltaY),  poniéndolo a 1. Se recomienda usar la instrucción SET.
* Es necesario **pintar en pantalla los dos cuadros** con el color correspondiente, tanto en la posición origen, como en destino de la caja.
* Como la caja se ha movido, el **Personaje también hay que moverlo en la tabla y pintarlo en la pantalla**, de su posición original PosX, PosY a la posición destino, que es donde estaba la caja antes : PosX+DeltaX, PosY+DeltaY. Esto es simplemente reutilizar el código de la práctica anterior.

Esta práctica es una evolución de la anterior y se hace necesario utilizar rutinas (Call / Ret) y, posiblemente, ficheros con include para que el código sea legible. Recuerda que tu programa **debe funcionar con cualquier nivel que siga la misma codificación**.

**APARTADO 2: EVALUAR EL TABLERO**  
Para conseguir saber cuándo hemos cumplido los objetivos, es necesario tener una rutina que evalúe si todas las cajas están en los destinos. Para eso, debes considerar el quinto byte de cada tabla de nivel, que dice el número de destinos (goals), es decir, de sitios donde hay que poner cajas.

Debes hacer una rutina que recorra toda la tabla del nivel contando en cuántos cuadros están el bit 6 y el bit 5 **a la vez** a 1. Esto significará que en la misma posición hay un destino y una caja, lo que queremos. Si esta cuenta es mayor o igual que el número de destinos que decía nuestro nivel, hemos completado el nivel.

Esta rutina puede ser llamada cada vez que hay un movimiento o, aún mejor, cada vez que se mueva una caja o, incluso mejor, cada vez que se mueva una caja y acabe dentro de un destino. Son tres etapas de optimización, pero usa una con la que te sientas cómodo.

**PISTA:**Para comprobar dos bits a la vez es recomendable usar máscaras. 01100000, que son los bits 5 y 6, es el valor $60. Si a un cuadro del mapa, que digamos que está en el registro A, se le hace un AND con $60, sabremos si esos dos bits estaban a 1 si el resultado es exactamente $60. -->

... bucle para recorrer toda la tabla  
LD A, (IX) ; Digamos que IX apunta al cuadro del mapa que estoy cargando  
AND $60 ;Borro todos los bits menos el 5 y el 6, que se quedan como estaban  
CP $60 ; Comparo con $60, para ver si el bit 5 y 6 que he cargado están a 1  
CALL Z, incrementarCuenta ; En efecto, este cuadro contiene un destino y una caja A LA VEZ  
.... seguir con el bucle para recorrer toda la tabla

incrementarCuenta: ; Función, por tanto, fuera del alcance del contador de programa si no se llega con un CALL  
   INC D; supongamos que la cuenta de cajas en destino está en D  
   RET