# Ejercicio evaluable 2: Uso de Memoria Dinámica

A lo largo del curso, se pedirá al alumno implementar un juego llamado “Ascii invaders”. Este programa replicará un juego de tipo “matamarcianos” usando la terminal para generar el escenario, enemigos, etc... Además, se le proporcionarán algunas funciones ya implementadas para facilitar el desarrollo del programa.

En este segundo ejercicio se pide implementar funciones para gestionar listas y arrays dinámicos de objetos. Además, se implementarán los movimientos de los objetos, y se modificarán algunas de las funciones implementadas en la primera práctica para adaptarlas a las nuevas estructuras.

## Parte 1: Implementar el tablero usando memoria dinámica

Al inicio del programa, se pedirá al usuario el tamaño del tablero:

- Número Filas
- Número de Columnas

Usando los archivos “talero.h” y “tablero.c”, se debe implementar una función que cree y retorne un tablero de objetos usando memoria dinámica. Se proporciona la siguiente cabecera, que deberá implementarse en el archivo “tablero.c”:

**Añadir a tablero.h**

```
objeto_t** reservaTablero(int numFilas, int numColumnas);
```

**Añadir a tablero.c**

```
//La siguiente función deberá reservar un array doble de objetos, de tamaño “numFilas” x
”numColumnas”
//añadir al archivo las cabeceras necesarias para tener malloc

objeto_t** reservaTablero(int numFilas, int numColumnas){
    //declarar un puntero doble de objeto_t llamado tablero

    //iniciar la primera dimensión del puntero doble “tablero” con un array de numFilas de
     //punteros a objeto_t

   //por cada fila del tablero
        //inicializar esa fila del tablero con un array de objeto_t de tamaño “numColumnas

    //devolver el array doble tablero
}
```

## Parte 2: Iniciar y dibujar el tablero usando memoria dinámica

Modificar las cabeceras de las funciones “iniciaTablero” y “dibujaTablero” para que use un puntero doble de objetos. Cambiar las cabeceras en “tablero.h” y “tablero.c” para que usar las siguientes:

```
void iniciaTablero(objeto_t** tablero, int numFilas, int numColumnas)

void dibujaTablero(objeto_t** tablero, int numFilas, int numColumnas)
```

**YA NO SE USAN LAS MACROS “NFILAS” Y “NCOLUMNAS”.**

Debe asegurarse el correcto funcionamiento del programa para cualquier tamaño de tablero dado por el usuario.

Debe asegurarse que las coordenadas X, Y almacenadas en el objeto_t del tablero coinciden con su posicicón en el mismo

## Parte 3: Movimiento de objetos

Hasta ahora, la función “crearEnemigo” inicializaba todos sus datos a “0”. Ahora se pide modificar la función para que los objetos de tipo enemigo tengan su array de movimientos relleno con la siguiente información:

Número de movimientos= 4;

Índice de movimiento= 0;

- Posición 0 del array:
  - X=1
  - Y=0
- Posición 1 del array:
  - X=0
  - Y=1
- Posición 2 del array:
  - X=-1;
  - Y=0;
- Posición 3 del array:
  - X=0;
  - Y=-1;

Crear las siguientes funciones en los archivos ".c" y ".h" indicados:

**Archivo "enemigo.h"**

```
void mueveEnemigo(objeto_t*  objeto, int numFilas, int numColumnas);
```

**Archivo "enemigo.c"**

```
void mueveEnemigo(objeto_t*  objeto, int numFilas, int numColumnas){

    //obtener el índice del movimiento que se va a realizar, que está almacenado en la estructura del enemigo

    //Sumar a la posición del objeto la cantidad de movimiento en X Y indicada por el array de movimientos, usando el índice obtenido anteriormente

    //Comprobar si la nueva posición está dentro del tablero. Para ello, usar numFilas/numColumnas

    //si alguna de las coordenadas no están dentro del tablero, el enemigo no se mueve en esa coordenada

    //Actualizar índice de movimiento:

    //sumar una unidad en el índice de movimiento

    //si es mayor o igual que el tamaño del array de movimientos, vuelve a 0

}
```

**Archivo "misil.h"**

```
void mueveMisil(objeto_t* objeto, int numFilas, int numColumnas);
```

**Archivo "misil.c"**

```
void mueveMisil (objeto_t* objeto, int numFilas, int numColumnas){

//si la dirección del misil es ascendente:

//sumar 1 a la coordenada Y de su posición

//si no

//restar 1 a la coordenada Y de su posición

    //comprobar coordenadas nuevas

    //si está fuera del número de Filas válidas del tablero, se desactiva

}
```

**Archivo “personaje.h”**

void muevePersonaje(objeto_t* objeto, int numFilas, int numColumnas);

**Archivo “personaje.c”**

//Añadir las cabeceras necesarias para tener las funciones de printf/scanf

**void muevePersonaje (objeto_t* objeto, int numFilas, int numColumnas){**

//Pedir movimiento al usuario:

“A”: Mover izquierda una unidad

“D”: Mover derecha una unidad

“S”: Acabar el juego

//Leer movimiento

//si se ha presionado “A”, mover una unidad a la derecha

//si se ha presionado “D”: mover una unidad a la izquierda

//si se ha presionado “S”: El personaje deja de estar activo

//comprobar coordenadas correctas:

//si están fuera del número de Columas válidas del tablero, no se mueve

}

## Parte 4: Actualización del tablero

Se debe crear una función para actualizar las posiciones de los objetos dentro del tablero. Por cada objeto activo del tablero, se invocará la función de actualización indicada por el tipo del objeto, pasándole por referencia a la función el objeto del tablero. Para ello, se pide añadir las siguientes funciones a los archivos “.c” y “.h” indicados:

**Archivo “tablero.h”**

void actualizaTablero(objeto_t** tablero, int numFilas, int numColumnas);

**Archivo “tablero.c”**

```
void actualizaTablero(objeto_t** tablero, int numFilas, int numColumnas)
{
//por cada posición del tablero
        //si es un objeto activo
//obtener el tipo del objeto
//llamar a su función de movimiento
//mueveEnemigo, mueveMisil o muevePersonaje

//Después de haber movido todos los objetos, actualizar sus posiciones dentro del tablero
//por cada objeto activo del tablero (recorrerlo con un for doble)
   //obtener su nueva posición X,Y almacenada en el objeto
  //comprobar si debe moverse (las nuevas posiciones son distintas de la posición
          //  actual)
   //si se mueve
                //Comprobar si en esa nueva posición ya hay un objeto activo
          //Si es así, significa que hay una colisión, se resolverán de la siguiente manera
  //Si colisiona un Misil con un Enemigo, se desactivan ambos
  //Si colisiona un Misil con el Personaje, se desactivan ambos
                //En otro caso,(no hay objeto activo) se mueve el objeto a la nueva posición
 //se copia el objeto a la posición X Y del tablero indicada por sus variables de
                  //   posición
 //se desactiva el objeto que estaba en la posición original (variable
                  //     activo del objeto accedido con los contadores del “for” a false)
}
```

## Parte 5: Función para averiguar si el programa ha terminado

El programa puede terminar por dos condiciones:

- El personaje principal fue destruido
- El usuario indicó que quería terminar

Si se han implementado correctamente las funciones pedidas, en ambos casos lo único que hay que hacer es buscar un objeto activo de tipo "personaje" dentro del tablero. Mientras haya un objeto personaje activo, el programa continuará.

Se pide implementar una función llamada "buscaPersonaje" que permita averiguar si hay un personaje activo en el tablero. Añadir las siguientes funciones a los archivos indicados:

**Archivo "tablero.h"**

```
int  buscaPersonaje(objeto_t** tablero, int numFilas, int numColumnas);
```

**Archivo "tablero.c"**

```
//Función que indica si hay un personaje activo dentro del tablero:
//retorna 1 si hay un personaje activo
//0 en otro caso
int  buscaPersonaje(objeto_t** tablero, int numFilas, int numColumnas)
{
//var encontrado=0
//por cada objeto del tablero
//si está activo y es un personaje
//encontrado=1
//retornar encontrado
}
```

## Parte 6: Bucle principal del programa “main”

Modificar el programa principal “main” para que realice las siguientes operaciones:

```
Int main(int argc, char** argv){
        //pedir numFilas y numColumnas
        //reservar tablero de tamaño numFilas numColumnas
        //iniciar tablero con objetos aleatorios
        //mientras haya un personaje activo
        //mostrar el tablero
        //actualizar tablero
        //repetir

        //liberar memoria del tablero

}
```

# Evaluación y entrega:

Se deben implementar todas las partes pedidas anteriormente. Se valorará lo siguiente:

- División del programa en varios ficheros temáticos:
    - Por cada tipo de objeto (enemigo, misil y personaje), un archivo “.h” y “.c” que contenga las funciones de inicialización pedidas.
    - Un archivo “.h” y “.c” para las funciones de inicialización y dibujado del tablero.
    - Un archivo “asciiMain.c” que contenga la función “main” del programa
- Definiciones adecuadas de estructuras y enumerados: Deben ajustarse a lo pedido en el enunciado
- Funcionamiento correcto: Generación de posiciones aleatorias para los objetos y dibujado del tablero sin errores
- Uso correcto de memoria dinámica
- Comprobaciones de accesos a arrays correctos.

**TODOS** los archivos generados deben contener el nombre del alumno y grupo al que pertenece.

El ejercicio es individual. **Está prohibido compartir, incluir o copiar código no desarrollado por el alumno**. En caso de detección de copias, el alumno se expone a suspender la asignatura, y dependiendo de la gravedad de la copia, podría considerarse la apertura de expediente y expulsión del curso.

El alumno deberá subir un archivo comprimido en formato **"zip" (no se admite otro formato)** a la actividad abierta en blackboard para realizar la entrega. El archivo deberá nombrarse de la siguiente manera:

NombreAlumno_Apellido_IPIIEjEv2.zip