UNIVERSIDAD DE MÁLAGA
Dpto. Lenguajes y CC. Computación
E.T.S.I. Telecomunicación

Prácticas Evaluables

Programación I

Sonido e Imagen

Listado de Prácticas Evaluables Individuales

El alumno debe realizar **individualmente** una práctica del **primer nivel** y una práctica del **segundo nivel**, ambas asignadas por el profesor. Opcionalmente, el alumno puede cambiar la práctica del problema del **segundo nivel** que le haya sido asignada, por otra práctica avanzada del **tercer** o **cuerto nivel**. Para ello, deberá hablar con el profesor.

Las prácticas realizadas por el alumno proporcionarán los puntos especificados en caso de ser realizadas adecuadamente, siendo **necesario superar** la práctica de *primer nivel* para obtener la puntuación correspondiente a las prácticas de *niveles superiores*. Así mismo, para poder superar las prácticas, el alumno deberá superar las **pruebas de autoría** especificadas por el profesor.

Se deben definir los *arrays* y *cadenas de caracteres* utilizando los tipos `array` y `string` definidos por la biblioteca estándar. No es adecuado utilizar los *arrays predefinidos*.

SE DEBE ENTREGAR UN CÓDIGO QUE COMPILE CORRECTAMENTE. Si alguna parte del código produce **errores** de compilación, entonces esa parte del código se debe poner entre **COMENTARIOS** `/* ... */` hasta que, finalmente, compile adecuadamente. Para ello, puede **quitar** la opción `-Werror` del comando de compilación.

Prácticas por Niveles

Cada práctica consiste en el diseño y desarrollo de un programa que resuelva el problema especificado.

1. Primer Nivel
   - *a*) Gestión de pedidos de una pizzería [6.5 pt]
2. Segundo Nivel
   - *a*) Comparación con Comodines [+3.5 pt]
   - *b*) Áreas Temáticas [+3.5 pt]
   - *c*) Reemplazar Patrón en Cadena [+3.5 pt]
   - *d*) Decodificar Mensaje en Morse [+3.5 pt]
   - *e*) Calcular Distancia de Recorrido [+3.5 pt]
3. Tercer Nivel
   - *a*) Gauss-Jordan para resolver un sistema de ecuaciones lineales expresado en forma matricial
   - *b*) Dibujar el perfil de un sendero de montaña
4. Cuarto Nivel
   - *a*) Juegos Básicos
     1) Conecta-4
     2) Mastermind
     3) Juego de la Vida
     4) Buscaminas
   - *b*) Juegos Intermedios
     1) Go
     2) Backgammon
   - *c*) Juegos Avanzados
     1) Bunker
     2) Arkanoid
     3) Tetris
     4) Serpiente
     5) Pacman
     6) Space Invaders

Nota: La corrección de las prácticas se realizará comprobando que la **ejecución** de los programas se corresponde con el **comportamiento especificado** en los enunciados correspondientes. Para ello, los programas deberán **compilar correctamente**.

Nota: Para las prácticas de cuarto nivel, puede resultar interesante la utilización de la biblioteca `misc.hpp` o `ioconsole.hpp`, que pueden ser descargadas de la página web de la asignatura.

Color: En caso de ser necesario la utilización de texto coloreado, la biblioteca `ioconsole.hpp` puede resultar adecuada, sin embargo, en muchas ocasiones es suficiente con utilizar diferentes símbolos (`o`, `x`, `#`, `@`, `$`, `%`) para representar las fichas de los jugadores.

1. Prácticas de Primer Nivel

1.1. Gestión de Pedidos de una Pizzería

Desarrolle un programa (`pizzeria.cpp`) para la gestión de los pedidos realizados por los clientes de una pizzería durante un día. La estructura de datos, denominada `Pizzeria`, contiene tres campos, denominados `numero_pedidos`, `codigo_pedidos`, y `lista_pedidos`. El campo `numero_pedidos` representa la cantidad de pedidos almacenados en el campo `lista_pedidos`, el cual es un array de `Pedido` que almacena la secuencia de pedidos que se han realizado en una pizzería, con un máximo de `MAX_PEDIDOS = 20` pedidos. Además, el campo `codigo_pedidos` se va incrementado **siempre** con cada pedido insertado, y permite asignar un nuevo código de pedido único a cada pedido (véase los subprogramas `inicializar` e `insertar_pedido` para comprender su utilidad).

Además, los datos de cada pedido se almacenan en una estructura de tipo `Pedido` que contiene cinco campos, denominados `nombre`, `telefono`, `codigo_pedido`, `numero_ingredientes` e `ingredientes`. El campo `nombre` representa una cadena de caracteres con el nombre del cliente que realiza el pedido. El campo `telefono` representa una cadena de caracteres con el teléfono del cliente que realiza el pedido. El campo `codigo_pedido` es un número entero que representa el código asociado al pedido. El campo `numero_ingredientes` es un número entero que contiene la cantidad de ingredientes solicitados en el pedido, que están almacenados en el campo `ingredientes`, el cual es un array de números enteros, que contiene un máximo de `MAX_INGREDIENTES_PIZZA = 10` de los códigos de los ingredientes de la pizza pedida. El código de cada ingrediente es un número entero que representa cada uno de los ingredientes disponibles en la pizzería: 0 *(tomate)*, 1 *(queso)*, 2 *(nata)*, 3 *(cebolla)*, 4 *(pollo)*, 5 *(huevo)*, 6 *(salami)*, 7 *(anchoa)*, 8 *(bacon)* y 9 *(gamba)*.

`Pizzeria`:

`codigo_pedidos`: 6

`numero_pedidos`: 3

`lista_pedidos`:

	0	1	2	...	MAX_PEDIDOS-1
(nombre)	Juan	Jose	Julia		
(telefono)	55555555	11111111	22222222		
(codigo_pedido)	1	2	4	...	...
(numero_ingredientes)	4	2	3		
(ingredientes)	0, 1, 3, 8	2, 4	1, 5, 9		

Así mismo, para soportar adecuadamente la entrada y salida de los códigos de los ingredientes, la correspondencia entre las cadenas de caracteres y los números enteros se realizará mediante la definición de un array constante de *"string"* denominado `INGREDIENTES` con los siguientes valores en minúsculas en las posiciones indicadas (*"tomate"*, *"queso"*, *"nata"*, *"cebolla"*, *"pollo"*, *"huevo"*, *"salami"*, *"anchoa"*, *"bacon"*, *"gamba"*).

	0	1	2	3	4	5	6	7	8	9
INGREDIENTES:	tomate	queso	nata	cebolla	pollo	huevo	salami	anchoa	bacon	gamba

Se deben definir las estructuras de datos para la gestión de los pedidos de la pizzería, junto con los subprogramas especificados a continuación. LOS NOMBRES, TIPOS, CAMPOS, SUBPROGRAMAS Y PARÁMETROS A DEFINIR DEBERÁN CORRESPONDERSE EXACTAMENTE CON LA ESPECIFICACIÓN DE ESTE ENUNCIADO.

- `void inicializar_datos(Pizzeria& p)`
  Inicializa la estructura de datos de la pizzería `p` de forma adecuada, es decir, la lista de pedidos vacía (`numero_pedidos` a cero) y el valor de `codigo_pedidos` a uno. Este subprograma no muestra nada en pantalla, ni lee nada de teclado.

- `void leer_ingrediente(int& ing)`
  Lee de teclado una cadena de caracteres, si ésta cadena representa un determinado ingrediente de la

pizza, entonces devuelve en el parámetro de salida `ing` el valor del número correspondiente al código de ingrediente introducido. Si la cadena leída no representa ningún ingrediente de la pizza, entonces mostrará un mensaje de error, y repetirá el proceso de lectura hasta que el valor introducido sea correcto.

La correspondencia entre las cadenas de caracteres y los números de los códigos de los ingredientes de la pizza se realizará mediante un proceso de búsqueda, de la cadena de caractereas leída, en el array constante previamente especificado (`INGREDIENTES`), de tal forma que el índice donde se encuentre la cadena indicará el número correspondiente al código del ingrediente, o error en caso de que no se encuentre.

- `void escribir_ingrediente(int ing)`
  Escribe en pantalla la cadena de caracteres correspondiente al número del código de ingrediente recibido como parámetro. Para ello, se deberá utilizar el array constante previamente especificado (`INGREDIENTES`), considerando que el código del ingrediente coincide con el índice en el array donde se encuentra el texto correspondiente a dicho ingrediente.

- `void leer_pedido(Pedido& ped, bool& ok)`
  Lectura de los datos de un pedido por teclado, que será devuelto en el parámetro de salida `ped`. Si el número de ingredientes leído es erróneo, entonces se devuelve `false` a través del parámetro `ok` de salida; en otro caso, se devuelve `true` a través de ese parámetro.

  Leerá de teclado, en el orden especificado, el nombre del cliente, su teléfono, el número de ingredientes de la pizza, y, en caso de que el número de ingredientes de la pizza sea correcto, tantos ingredientes (utilizando el subprograma `leer_ingrediente`) como especifique el número leído anteriormente. Además, se asignará el valor **cero** al código de pedido. Nótese que sí es posible que haya ingredientes repetidos en el mismo pedido.

- `void insertar_pedido(Pizzeria& p, const Pedido& ped, bool& ok)`
  Si el pedido cabe, se inserta el pedido `ped`, recibido como parámetro, al final de la lista de pedidos de la pizzería `p` (recibida como parámetro), incrementándose el número total de pedidos almacenados (`numero_pedidos`), se asigna al nuevo pedido **almacenado** en la lista el código de pedido correspondiente (tomado de `codigo_pedidos`), incrementándose el código de pedidos general (`codigo_pedidos`), y se devuelve `true` a través del parámetro `ok` de salida; en otro caso, se devuelve `false` a través de ese parámetro. Nótese que sí es posible que haya varios pedidos del mismo cliente. Este subprograma no muestra nada en pantalla, ni lee nada de teclado.

- `void escribir_pedido(const Pedido& ped)`
  Escribe en pantalla, según el formato del siguiente ejemplo, los datos del pedido `ped` que recibe como parámetro (utilizando el subprograma `escribir_ingrediente`). Así, para el primer pedido de la figura anterior mostrará:

  ```
  Juan 55555555 1 4 tomate queso cebolla bacon
  ```

- `void escribir_pedidos(const Pizzeria& p)`
  Escribe en pantalla, según el formato del siguiente ejemplo, el contenido de toda la estructura de datos `p` (utilizando el subprograma `escribir_pedido`), así como el listado de ingredientes disponibles (utilizando el subprograma `escribir_ingrediente`). Así, para la figura anterior mostrará (nótese que se muestra primero el valor del campo `codigo_pedidos` de la pizzería `p`):

  ```
  6
  3
  Juan 55555555 1 4 tomate queso cebolla bacon
  Jose 11111111 2 2 nata pollo
  Julia 22222222 4 3 queso huevo gamba
  tomate queso nata cebolla pollo huevo salami anchoa bacon gamba
  ```

- `void eliminar_pedido(Pizzeria& p, int cod_pedido, bool& ok)`
  Busca, en la lista de pedidos de la pizzería `p`, la posición donde se encuentra el pedido con el código de pedido especificado por el parámetro `cod_pedido`, y si el pedido con ese código existe, entonces se **elimina** de la lista (los pedidos a su derecha se **desplazan** una posición a la izquierda para mantener el orden en la lista), se disminuye el número total de pedidos y se devuelve `true` a través del parámetro `ok` de salida; si el pedido especificado no existe, entonces se devuelve `false` a través de ese parámetro. Este subprograma no muestra nada en pantalla, ni lee nada de teclado.

  Nótese que el código de pedido **NO indica** la posición del pedido en la lista de pedidos, por lo que el pedido correspondiente deberá ser buscado en la lista.

- `int buscar_pedido(const Pizzeria& p, const string& nombre)`
  Devuelve el índice en el listado de pedidos donde se encuentra almacenado el primer pedido de una persona cuyo nombre se recibe como parámetro. En caso de no encontrar ningún pedido bajo dicho nombre se devolverá el valor **-1**. Para el ejemplo de la figura y el nombre `Juan`, devolverá el número cero (0). Este subprograma no muestra nada en pantalla, ni lee nada de teclado.

- `void ingrediente_preferido(const Pizzeria& p, int& ing)`
  Calcula y asigna al parámetro de salida `ing` el código de ingrediente que haya sido seleccionado el **mayor** número de veces de entre todos los pedidos almacenados en la lista de pedidos de la pizzería `p`. En el caso de que haya **varios** ingredientes que hayan sido seleccionados el mismo número de veces que el mayor de ellos (*igual de preferidos*), entonces asignará al parámetro de salida `ing` el que tenga menor *código de ingrediente* de entre los que sean *igualmente preferidos*. Nótese que si no hay ningún pedido, devolverá el código de ingrediente cero. Por ejemplo, para la figura anterior, el ingrediente preferido tiene el código 1 (*queso*). Este subprograma no muestra nada en pantalla, ni lee nada de teclado.

- `char menu()`
  Muestra en pantalla un menú con las opciones necesarias para probar los algoritmos especificados anteriormente, así como devuelve la opción seleccionada por el operador. NÓTESE QUE ESTE SUBPROGRAMA NO SERÁ UTILIZADO PARA LA CORRECCIÓN DE LA PRÁCTICA.

- `int main()`
  Permite ejecutar de forma arbitraria, por medio de un menú de selección, los algoritmos especificados anteriormente. Nótese que inicialmente deberá invocar (antes del menú de selección) al subprograma `inicializar_datos(...)` para inicializar la estructura de datos de forma adecuada. NÓTESE QUE ESTE PROGRAMA PRINCIPAL NO SERÁ UTILIZADO PARA LA CORRECCIÓN DE LA PRÁCTICA.

2. Prácticas de Segundo Nivel

2.1. Comparación con Comodines

Desarrolle un programa (`compatron.cpp`) que lea una secuencia de nombres (uno por línea hasta leer una línea en blanco, como máximo 20 nombres) que almacenará en un array, y que posteriormente lea cadenas de caracteres con comodines (`'*'` y `'?'`) que representan patrones de búsqueda, y muestre por pantalla todos aquellos nombres almacenados en el array que se **correspondan** con el patrón de búsqueda especificado. Se realizará este proceso hasta que se introduzca la cadena vacía como patrón de búsqueda.

La correspondencia entre el patrón y un determinado nombre es como se especifica a continuación:

- El símbolo `'?'` en el *patrón* coincide con cualquier carácter del *nombre* en la posición especificada, pero sólo un único carácter.
- El símbolo `'*'` en el *patrón* coincide con cualquier secuencia de caracteres del *nombre*, desde la posición especificada hasta el final. Nótese que si en el patrón aparece un símbolo asterisco en una determinada posición, no es necesario comparar lo que haya en el nombre desde esa posición hasta el final, ya que el asterisco coincide con cualquier secuencia de caracteres hasta el final.
- Cualquier otro carácter en el *patrón* que aparezca antes de un símbolo asterisco, debe coincidir con el mismo carácter en el *nombre* y en la misma posición.
- Cualquier carácter en el *patrón* que aparezca después de un símbolo asterisco es *irrelevante* y no será tenido en consideración.
- **No** esta permitida la utilización de los métodos `find...`, ni `rfind` de la clase `string`.

El programa deberá mostrar una interacción con el usuario como muestra el siguiente ejemplo, donde el texto en color rojo representa texto introducido por el usuario a través del teclado, el texto en color azul representa el texto de salida resultado de la operación, y el texto en color negro representa mensajes auxiliares de ayuda:

```
Introduzca una secuencia de nombres hasta linea en blanco:
pep
pepe
papaito
luis
papito
maria
lola
lucas

Introduzca patron a buscar: p?p?*
pepe
papaito
papito

Introduzca patron a buscar: p?p*
pep
pepe
papaito
papito

Introduzca patron a buscar: pap*
papaito
papito

Introduzca patron a buscar: pap*qwe?**
papaito
papito
```

2.2. Áreas Temáticas

Desarrolle un programa (`areas.cpp`) para realizar un estudio estadístico de un texto introducido por teclado. Dicho estudio consiste en contar cuantas palabras de determinadas *"áreas temáticas"* existen en el texto. Estas áreas temáticas vendrán identificadas en el texto como *palabras reservadas*, escritas con todas las letras en mayúsculas, de tal forma que la secuencia de palabras que siguen a una palabra reservada se contabilizan como palabras pertenecientes a dicha área temática.

La entrada de datos consiste en una secuencia de longitud indeterminada de palabras (separadas por espacios en blanco) terminada por la palabra reservada `FIN`, donde cada área temática se indica como una palabra reservada formada solamente por letras mayúsculas, y la secuencia de palabras que la siguen (hasta otra palabra reservada) se deben contabilizar como palabras pertenecientes al área temática especificada por la palabra reservada que las precede.

El programa deberá mostrar una interacción con el usuario como muestra el siguiente ejemplo, donde el texto en color rojo representa texto introducido por el usuario a través del teclado, el texto en color azul representa el texto de salida resultado de la operación, y el texto en color negro representa mensajes auxiliares de ayuda:

```
Introduzca texto (hasta FIN):
HISTORIA En el Pleistoceno la Tierra estaba LITERATURA Cervantes
escribio CIENCIAS la reaccion nuclear producida al fusionar
LITERATURA El Ingenioso Hidalgo CIENCIAS dos atomos de hidrogeno FIN

Estadisticas:
HISTORIA: 6 palabras
LITERATURA: 5 palabras
CIENCIAS: 10 palabras
```

Nota: las palabras estarán formadas por letras mayúsculas y minúsculas, y separadas por espacios en blanco. El número máximo de áreas temáticas es de 30, y cada palabra está formada por una secuencia de longitud finita de caracteres.

2.3. Reemplazar Patrón en Cadena

Desarrolle un programa (`reemplazar.cpp`) para reemplazar cadenas de texto. Para ello diseñe un procedimiento que tome como entrada las cadenas de caracteres `texto`, `patron` y `sust`, y produzca como resultado `nuevo`, que contendrá los mismos caracteres que `texto`, pero donde todas las ocurrencias de `patron` en el `texto` habrán sido sustituidas por el valor de `sust`. Además, el patrón es una cadena de caracteres que puede tener un asterisco incluido, el cual representa cualquier secuencia de caracteres que permita la correspondencia. Tenga en cuenta que asterisco no aparecerá ni al principio ni al final del patrón. Además, el texto reemplazado no es candidato para ser nuevamente procesado ni reemplazado.

```
void reemplazar(const string& texto, const string& patron, const string& sust, string& nuevo);
```

El programa principal deberá solicitar los datos necesarios y mostrar la solución, tal y como muestra el siguiente ejemplo.

Nótese que **no** esta permitida la utilización de los métodos `find...`, `rfind`, `replace`, `insert`, ni `erase` de la clase `string`.

El programa deberá mostrar una interacción con el usuario como muestra el siguiente ejemplo, donde el texto en color rojo representa texto introducido por el usuario a través del teclado, el texto en color azul representa el texto de salida resultado de la operación, y el texto en color negro representa mensajes auxiliares de ayuda:

```
patron: abx*sdfg
sust: qwerty
texto: khsfdabxsadabx asdfgiolsdfg abxsdfgkjjk abx sfessdfgfjlkjf
nuevo: khsfdqwertyiolsdfg qwertykjjk qwertyfjlkjf
```

2.4. Decodificar Mensaje Codificado en Morse

Desarrolle un programa (`morse.cpp`) que lea como entrada un mensaje de caracteres conteniendo un texto codificado en el alfabeto Morse, y muestre en pantalla el mensaje correspondiente trás decodificar el mensaje original con el texto codificado en el alfabeto español.

El código Morse para el alfabeto español se debe almacenar en un array con `NEMLS = 42` elementos, donde cada uno es un registro con dos campos de tipo cadena de caracteres, de tal forma que el primer campo contiene la secuencia de caracteres para cada letra, digito o símbolos especiales del alfabeto español (nótese que la letra 'CH' se representa con dos caracteres), y el segundo campo contiene la secuencia de caracteres correspondiente en código Morse.

```
A   .-          N   -.          0   -----
B   -...        Ñ   --.--       1   .----
C   -.-.        O   ---         2   ..---
CH  ----        P   .--.        3   ...--
D   -..         Q   --.-        4   ....-
E   .           R   .-.         5   .....
F   ..-.        S   ...         6   -....
G   --.         T   -           7   --...
```

```
H   ....        U   ..-        8   ---..
I   ..          V   ...-       9   ----.
J   .---        W   .--        .   .-.-.-
K   -.-         X   -..-       ,   --..--
L   .-..        Y   -.--       ?   ..--..
M   --          Z   --..       "   .-..-.
```

Nótese que, en el mensaje de entrada, los símbolos en código Morse que representan cada símbolo del alfabeto español se encuentran separados por un espacio en blanco, y las diferentes palabras que componen el texto se encuentran separadas por **más** de un espacio en blanco.

Nótese que **no** esta permitida la utilización de los métodos `find...` ni `rfind` de la clase `string`.

El programa deberá mostrar una interacción con el usuario como muestra el siguiente ejemplo, donde el texto en color rojo representa texto introducido por el usuario a través del teclado, el texto en color azul representa el texto de salida resultado de la operación, y el texto en color negro representa mensajes auxiliares de ayuda:

```
morse: .... --- .-.. .-     -.--     .- -.. .. --- ...
texto: hola y adios
```

Nota: En caso de error en la decodificación, se mostrará la cadena de caracteres `"ERROR"`.

2.5. Calcular Distancia de Recorrido

Se dispone de la siguiente información **constante** con las distancias entre las ciudades[1] de Andalucía conectadas directamente por autovía (nótese que es posible moverse en ambos sentidos), que deberá ser definida de forma adecuada para que pueda ser utilizada por nuestro programa. Nótese que el programa debe diseñarse de forma independiente del número de conexiones y de las conexiones especificadas en la siguiente tabla:

Huelva	Sevilla	96
Sevilla	Cadiz	145
Sevilla	Cordoba	133
Sevilla	Malaga	215
Sevilla	Granada	269
Cordoba	Jaen	105
Cordoba	Malaga	172
Malaga	Cadiz	268
Malaga	Granada	164
Malaga	Almeria	210
Granada	Jaen	91
Granada	Almeria	166

Además, se deben definir los siguientes tipos y subprogramas:

- `Recorrido`

  es un tipo que representa una *lista* de un máximo de 20 elementos, donde cada elemento es de tipo *cadena de caracteres*. Este tipo representa la secuencia de ciudades por las que se pasa en un determinado recorrido por Andalucía. Nótese que es posible pasar por la misma ciudad múltiples veces, y realizar movimientos en ambos sentidos.

- `void leer_recorrido(Recorrido& rec)`

  lee de teclado una secuencia de nombres de ciudades, separadas por espacios en blanco o saltos de línea, hasta que se introduzca un nombre *inválido* de ciudad o se haya introducido un total de 20 nombres de ciudades, y vaya almacenando cada nombre de ciudad en el parámetro `rec` (*lista* de tipo `Recorrido`), considerando que el nombre *inválido* de la ciudad no se almacenará. Téngase en cuenta que cualquier nombre que no coincida con el nombre de alguna ciudad será considerado como *inválido*.

  Por ejemplo, para la siguiente secuencia de datos de entrada:

  ```
  Malaga Cordoba Sevilla Cadiz Malaga Almeria fin
  ```

  Almacenará los siguientes elementos en la *lista* de entrada (`datos`):

  `nelms:` 6

  `elm:`

0	1	2	3	4	5	···	19
Malaga	Cordoba	Sevilla	Cadiz	Malaga	Almeria	···	···

[1] Los nombres de las ciudades están escritos sin tildes exprésamente

- `int calcular_distancia(const Recorrido& rec)`

  recibe en el parámetro `rec` una secuencia de ciudades de Andalucía que forman un recorrido en ese mismo orden secuencial entre ciudades. La función debe calcular la distancia total del recorrido (la suma de todas las distancias entre las ciudades recorridas) y devolver dicho valor calculado.

  Sin embargo, si en el recorrido secuencial aparece algún movimiento entre dos ciudades consecutivas que no estén conectadas directamente, entonces el recorrido es **erróneo** y se devolverá un valor `-1`.

  Nótese que es posible pasar por la misma ciudad múltiples veces, y realizar movimientos en ambos sentidos.

  Por ejemplo, devolverá un valor de `928` $(172 + 133 + 145 + 268 + 210)$ para un recorrido entre las siguientes ciudades:

  ```
  Malaga Cordoba Sevilla Cadiz Malaga Almeria
  ```

El programa principal (denominado `recorrido.cpp`) debe leer una secuencia de ciudades (utilizando el procedimiento `leer_recorrido`), calcular la distancia de dicho recorrido (utilizando la función `calcular_distancia`), y mostrar en pantalla la distancia calculada. Si el recorrido es erróneo, entonces se mostrará el mensaje `"ERROR"`.

El programa deberá mostrar una interacción con el usuario como muestra el siguiente ejemplo, donde el texto en color rojo representa texto introducido por el usuario a través del teclado, el texto en color azul representa el texto de salida resultado de la operación, y el texto en color negro representa mensajes auxiliares de ayuda:

```
recorrido: Malaga Cordoba Sevilla Cadiz Malaga Almeria fin
distancia: 928
```

Nota: En caso de error, se mostrará el mensaje ERROR como resultado de la distancia.

3. Prácticas de Tercer Nivel

3.1. Resolución de ecuaciones lineales por Gauss-Jordan

Desarrolle un programa (`gauss_jordan.cpp`) que lea de teclado los elementos de la *matriz aumentada* que representa un sistema de ecuaciones lineales, realice las transformaciones de Gauss-Jordan especificadas a continuación, y finalmente muestre la solución del sistema de ecuaciones lineales.

El método de eliminación de Gauss-Jordan permite resolver sistemas de ecuaciones lineales de una forma algoritmica muy adecuada para su procesamiento automático por un ordenador. Así, un sistema de ecuaciones lineales se puede representar mediante una *matriz aumentada* de la siguiente forma:

Sistema de Ecuaciones Lineales	Matriz Aumentada	Transf. Gauss-Jordan Matriz Escalonada Reducida	Solución Sist. Ec. Lineales
$\begin{aligned} 2x+3y+5z &= 5 \\ 3x+7z &= 2 \\ 5x+2y+z &= 3 \end{aligned}$	$\left[\begin{array}{ccc\|c} 2 & 3 & 5 & 5 \\ 3 & 0 & 7 & 2 \\ 5 & 2 & 1 & 3 \end{array}\right]$	$\left[\begin{array}{ccc\|c} 1 & 0 & 0 & 0{,}07 \\ 0 & 1 & 0 & 1{,}19 \\ 0 & 0 & 1 & 0{,}25 \end{array}\right]$	$\begin{aligned} x &= 0{,}07 \\ y &= 1{,}19 \\ z &= 0{,}25 \end{aligned}$

La transformación de Gauss-Jordan consiste en aplicar tranformaciones básicas de matrices (intercambio de filas, multiplicar una fila por un valor, y sumar a una fila otra fila multiplicada por un valor) a la matriz aumentada hasta transformarla en forma de matriz escalonada reducida. En caso de no poder transformar la matriz original en forma de matriz escalonada reducida, entonces el sistema de ecuaciones lineales no tiene solución.

La transformación de Gauss-Jordan procede del siguiente modo: dada la matriz a de n filas y $n+1$ columnas, para cada fila $k \in [1..n]$, comenzando desde la primera fila, realizaremos las siguientes transformaciones:

$$\left[\begin{array}{cccc|c} a_{11} & a_{12} & \dots & a_{1n} & a_{1n+1} \\ a_{21} & a_{22} & \dots & a_{2n} & a_{2n+1} \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ a_{n1} & a_{n2} & \dots & a_{nn} & a_{nn+1} \end{array}\right]$$

1. Usaremos como *pivote* el elemento de la diagonal principal (a_{kk}). Mediante la operación básica de intercambio de filas, haremos que el elemento *pivote* sea el de mayor valor absoluto de la columna k a partir de la fila k hasta el final, para ello, si el elemento a_{jk} es el de mayor valor absoluto de entre todos elementos de la columna k (a partir de la fila k), es decir, $a_{jk} \geq a_{ik} \forall i \in [k..n]$, entonces intercambiaremos las filas completas j y k.

2. Una vez que el valor absoluto del elemento a_{kk} es el mayor de la columna k (a partir de la fila k), si el elemento a_{kk} es igual a cero ($\text{abs}(a_{kk}) < 1e-6$), entonces el sistema de ecuaciones no tiene solución. En otro caso procederemos con el siguiente punto.

3. Excepto para el elemento pivote a_{kk}, transformaremos cada elemento de la columna k ($a_{ik}, \forall i \in [1..n]/i \neq k$) en cero (0) mediante la operación de restar a la fila i el resultado de multiplicar la fila k por a_{ik}/a_{kk}.

4. Finalmente tranformaremos el elemento pivote a_{kk} en uno (1) mediante la operación de multiplicar la fila k por $1/a_{kk}$.

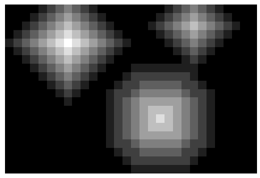
Por ejemplo, para el sistema de ecuaciones mostrado anteriormente:

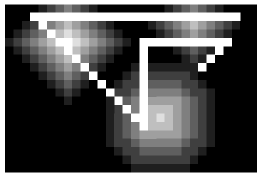
Inicial	Mayor Pivote	Hacer 0 1er elem.	Hacer 0 2do. elem.	Hacer 1 piv
$\left[\begin{array}{ccc\|c} 2{,}00 & 3{,}00 & 5{,}00 & 5{,}00 \\ 3{,}00 & 0{,}00 & 7{,}00 & 2{,}00 \\ 5{,}00 & 2{,}00 & 1{,}00 & 3{,}00 \end{array}\right]$	$\left[\begin{array}{ccc\|c} 5{,}00 & 2{,}00 & 1{,}00 & 3{,}00 \\ 3{,}00 & 0{,}00 & 7{,}00 & 2{,}00 \\ 2{,}00 & 3{,}00 & 5{,}00 & 5{,}00 \end{array}\right]$	$\left[\begin{array}{ccc\|c} 5{,}00 & 2{,}00 & 1{,}00 & 3{,}00 \\ 0{,}00 & -1{,}20 & 6{,}40 & 0{,}20 \\ 2{,}00 & 3{,}00 & 5{,}00 & 5{,}00 \end{array}\right]$	$\left[\begin{array}{ccc\|c} 5{,}00 & 2{,}00 & 1{,}00 & 3{,}00 \\ 0{,}00 & -1{,}20 & 6{,}40 & 0{,}20 \\ 0{,}00 & 2{,}20 & 4{,}60 & 3{,}80 \end{array}\right]$	$\left[\begin{array}{ccc\|c} 1{,}00 & 0{,}40 & 0{,}20 & 0{,}60 \\ 0{,}00 & -1{,}20 & 6{,}40 & 0{,}20 \\ 0{,}00 & 2{,}20 & 4{,}60 & 3{,}80 \end{array}\right]$
$\left[\begin{array}{ccc\|c} 1{,}00 & 0{,}40 & 0{,}20 & 0{,}60 \\ 0{,}00 & -1{,}20 & 6{,}40 & 0{,}20 \\ 0{,}00 & 2{,}20 & 4{,}60 & 3{,}80 \end{array}\right]$	$\left[\begin{array}{ccc\|c} 1{,}00 & 0{,}40 & 0{,}20 & 0{,}60 \\ 0{,}00 & 2{,}20 & 4{,}60 & 3{,}80 \\ 0{,}00 & -1{,}20 & 6{,}40 & 0{,}20 \end{array}\right]$	$\left[\begin{array}{ccc\|c} 1{,}00 & 0{,}00 & -0{,}63 & -0{,}09 \\ 0{,}00 & 2{,}20 & 4{,}60 & 3{,}80 \\ 0{,}00 & -1{,}20 & 6{,}40 & 0{,}20 \end{array}\right]$	$\left[\begin{array}{ccc\|c} 1{,}00 & 0{,}00 & -0{,}63 & -0{,}09 \\ 0{,}00 & 2{,}20 & 4{,}60 & 3{,}80 \\ 0{,}00 & 0{,}00 & 8{,}90 & 2{,}27 \end{array}\right]$	$\left[\begin{array}{ccc\|c} 1{,}00 & 0{,}00 & -0{,}63 & -0{,}09 \\ 0{,}00 & 1{,}00 & 2{,}09 & 1{,}72 \\ 0{,}00 & 0{,}00 & 8{,}90 & 2{,}27 \end{array}\right]$
$\left[\begin{array}{ccc\|c} 1{,}00 & 0{,}00 & -0{,}63 & -0{,}09 \\ 0{,}00 & 1{,}00 & 2{,}09 & 1{,}72 \\ 0{,}00 & 0{,}00 & 8{,}90 & 2{,}27 \end{array}\right]$	$\left[\begin{array}{ccc\|c} 1{,}00 & 0{,}00 & -0{,}63 & -0{,}09 \\ 0{,}00 & 1{,}00 & 2{,}09 & 1{,}72 \\ 0{,}00 & 0{,}00 & 8{,}90 & 2{,}27 \end{array}\right]$	$\left[\begin{array}{ccc\|c} 1{,}00 & 0{,}00 & 0{,}00 & 0{,}07 \\ 0{,}00 & 1{,}00 & 2{,}09 & 1{,}72 \\ 0{,}00 & 0{,}00 & 8{,}90 & 2{,}27 \end{array}\right]$	$\left[\begin{array}{ccc\|c} 1{,}00 & 0{,}00 & 0{,}00 & 0{,}07 \\ 0{,}00 & 1{,}00 & 0{,}00 & 1{,}19 \\ 0{,}00 & 0{,}00 & 8{,}90 & 2{,}27 \end{array}\right]$	$\left[\begin{array}{ccc\|c} 1{,}00 & 0{,}00 & 0{,}00 & 0{,}07 \\ 0{,}00 & 1{,}00 & 0{,}00 & 1{,}19 \\ 0{,}00 & 0{,}00 & 1{,}00 & 0{,}25 \end{array}\right]$

3.2. Dibujar el Perfil de un Sendero de Montaña

Desarrolle un programa (`perfil.cpp`) para calcular el perfil de una ruta según lo especificado a continuación. Dada una **superficie**, un array de `20` filas por `30` columnas de números naturales, como en el siguiente ejemplo, donde cada número representa la altura de la superficie del terreno en un determinado punto.

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30
1	0	0	0	0	1	2	3	4	3	2	1	0	0	0	0	0	0	0	0	1	2	3	4	3	2	1	0	0	0	0
2	0	0	0	1	2	3	4	5	4	3	2	1	0	0	0	0	0	0	1	2	3	4	5	4	3	2	1	0	0	0
3	0	0	1	2	3	4	5	6	5	4	3	2	1	0	0	0	0	1	2	3	4	5	6	5	4	3	2	1	0	0
4	0	1	2	3	4	5	6	7	6	5	4	3	2	1	0	0	0	0	1	2	3	4	5	4	3	2	1	0	0	0
5	1	2	3	4	5	6	7	8	7	6	5	4	3	2	1	0	0	0	0	1	2	3	4	3	2	1	0	0	0	0
6	0	1	2	3	4	5	6	7	6	5	4	3	2	1	0	0	0	0	0	0	1	2	3	2	1	0	0	0	0	0
7	0	0	1	2	3	4	5	6	5	4	3	2	1	0	0	0	0	0	0	0	0	1	2	1	0	0	0	0	0	0
8	0	0	0	1	2	3	4	5	4	3	2	1	0	0	0	1	1	1	1	1	1	1	1	0	0	0	0	0	0	0
9	0	0	0	0	1	2	3	4	3	2	1	0	0	0	1	2	2	2	2	2	2	2	1	0	0	0	0	0	0	0
10	0	0	0	0	0	1	2	3	2	1	0	0	0	1	2	2	3	3	3	3	3	2	2	1	0	0	0	0	0	0
11	0	0	0	0	0	0	1	2	1	0	0	0	1	2	2	3	4	4	4	4	4	3	2	2	1	0	0	0	0	0
12	0	0	0	0	0	0	0	1	0	0	0	0	1	2	3	4	5	5	5	5	5	4	3	2	1	0	0	0	0	0
13	0	0	0	0	0	0	0	0	0	0	0	0	1	2	3	4	5	6	6	6	5	4	3	2	1	0	0	0	0	0
14	0	0	0	0	0	0	0	0	0	0	0	0	1	2	3	4	5	6	7	6	5	4	3	2	1	0	0	0	0	0
15	0	0	0	0	0	0	0	0	0	0	0	0	1	2	3	4	5	6	6	6	5	4	3	2	1	0	0	0	0	0
16	0	0	0	0	0	0	0	0	0	0	0	0	1	2	3	4	5	5	5	5	5	4	3	2	1	0	0	0	0	0
17	0	0	0	0	0	0	0	0	0	0	0	0	1	2	2	3	4	4	4	4	4	3	2	2	1	0	0	0	0	0
18	0	0	0	0	0	0	0	0	0	0	0	0	0	1	2	2	3	3	3	3	3	2	2	1	0	0	0	0	0	0
19	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	2	2	2	2	2	2	2	1	0	0	0	0	0	0	0
20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	0	0	0	0	0	0	0	0





Dada una **ruta**, un vector de coordenadas (representa la fila y columna de un determinado punto en la superficie) de máximo `10` elementos, aunque puede tener menos (incluso ninguno), como en el siguiente ejemplo.

{ 2, 28 }, { 2, 4 }, { 15, 17 }, { 5, 17 }, { 5, 27 }, { 8, 24 }

Donde cada coordenada (salvo la última) representa un **camino lineal** que recorre la superficie desde ese punto hasta el punto indicado por la siguiente coordenada en el vector. Por simplificar, consideraremos que los caminos lineales sólo pueden ser **horizontales**, **verticales** y **diagonales**, en cualquier dirección.

Así, para la anterior superficie y ruta del ejemplo, la ruta pasará sobre los puntos con las siguientes alturas y en el siguiente orden indicado:

0 1 2 3 4 5 4 3 2 1 0 0 0 0 0 0 0 1 2 3 4 5 4 3 2 1 3 5 7 7 5 3 1 0 1 2 3 4 5 5 5 5 4 3 2 1 0 0 0 0 0 0 1 2 3 4 3 2 1 0 0 0 0

Se pide definir los tipos necesarios, y realizar un procedimiento que reciba una superficie y una determinada ruta, ambos conteniendo los valores adecuados y necesarios, y escriba en pantalla un perfil (histograma) de las alturas de los puntos que componen la ruta, en el mismo orden que el seguido por la ruta. Así, para el anterior ejemplo escribirá el siguiente perfil (histograma). Consideraremos que no podrá haber más de `100` puntos en una determinada ruta aunque posiblemente habrá menos, e incluso ninguno. Nótese que una altura 0 se representa con 1 asterisco, y así sucesivamente. Así mismo, diseñe el programa y los subprogramas necesarios para probar adecuadamente su funcionamiento.

```
                            **
                            **
     *               *     ****       ****
    ***             ***    ****      ******            *
   *****           *****  ******    ********          ***
  *******         ******* ******   **********        *****
 *********       **************** ************      *******
***************************************************************
```

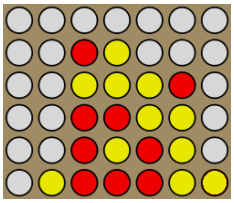
4. Prácticas de Cuarto Nivel

4.1. Conecta-4 (Cuatro en Raya)

- http://en.wikipedia.org/wiki/Connect_four

Es un juego para dos jugadores, donde ambos jugadores, por turnos, depositan una ficha en una determinada columna, por la parte superior, que se moverá hasta la fila más inferior posible. El juego terminará cuando un determinado jugador consiga establecer 4 fichas consecutivas (en linea), ya sea en dirección horizontal, vertical, o diagonal.

```
    a   b   c   d   e   f   g
  +---+---+---+---+---+---+---+
6 |   |   |   |   |   |   |   | 6
  +---+---+---+---+---+---+---+
5 |   |   | o |   |   |   |   | 5
  +---+---+---+---+---+---+---+
4 |   |   | x | x | x | o |   | 4
  +---+---+---+---+---+---+---+
3 |   |   | o | o | x | x |   | 3
  +---+---+---+---+---+---+---+
2 |   |   | o | x | o | x |   | 2
  +---+---+---+---+---+---+---+
1 |   | x | o | o | o | x | x | 1
  +---+---+---+---+---+---+---+
    a   b   c   d   e   f   g

Introduzca la columna del jugador [x]:  g [ENTER]
```



4.2. Mastermind

- http://es.wikipedia.org/wiki/Mastermind
- http://en.wikipedia.org/wiki/Mastermind_(board_game)

Es un juego para un jugador. Dados seis símbolos diferentes (`ABCDEF`), el ordenador selecciona aleatoriamente una combinación de cuatro símbolos de entre los seis símbolos anteriores (es posible repetirlos), y el jugador debe intentar adivinar dicha combinación. Para ello, para cada entrada dada por el jugador, el ordenador indicará cuantos símbolos acertados están el la posición adecuada, y cuantos símbolos acertados están en una posición errónea. Además, el ordenador mostrará la historia de los intentos anteriores del jugador. Por ejemplo, para un determinado código seleccionado aleatoriamente por el ordenador (`ABCA`), el juego podría proceder de la siguiente manera:

```
1: ACCB  -> 2c 1e
2: ACDB  -> 1c 2e

Introduzca nueva combinacion:  ABCF [ENTER]
```

4.3. El Juego de la Vida

- http://es.wikipedia.org/wiki/Juego_de_la_vida
- http://en.wikipedia.org/wiki/Conways_Game_of_Life

El *juego de la vida* no es un juego, sino un modelo de una población de seres vivos simplificados que interactúan entre sí y con su entorno. En el núcleo del modelo se encuentra una cuadrícula que representa el "mundo".

Cada cuadrado puede estar ocupado por un ser vivo o puede estar vacío (no se admite más de un ser vivo en un mismo cuadrado). Al comienzo de la simulación, se seleccionará de forma aleatoria el número de elementos que contienen vida y en que posición se encuentran.

La ejecución genera una serie de generaciones, donde cada una de ellas es un mundo derivado del mundo anterior. Para cada generacion, **nacerá** vida en un determinado cuadrado que no la tenía si en la generación anterior el número de vecinos (entre las 8 casillas que la rodean) era igual a `LIM_SUP=3`. Si en la generación anterior una determinada casilla tenía vida, ésta **morirá** si el número de vecinos que tenía en dicha generación

era inferior a `LIM_INF=2` o superior a `LIM_SUP=3`, en otro caso (el número de vecinos se encontraba entre `LIM_INF=2` y `LIM_SUP=3` ambos inclusive) **continuará** con vida.

La simulación continúa hasta que no quede vida en el tablero, o hasta que no haya ningún cambio entre una generación y la siguiente.

Ejemplo:

```
+----------------------------------------+
|........................................|
|....................00..................|
|...................0..0.................|
|....................0.0.................|
|.....................0..................|
|........................................|
|........................................|
|........................................|
|........................................|
|...............000......................|
|........................................|
|........................................|
|........................................|
|........................0...............|
|......................00.00.............|
|........................0...............|
|........................................|
|........................................|
|........................................|
|........................................|
|..............00...............0........|
|..............0.0..............0........|
|..............0................0........|
|........................................|
|...........................000...000....|
|........................................|
|...............................0........|
|............00.................0........|
|............00.................0...00...|
|.........................00........00...|
|.........................00.............|
|........................................|
+----------------------------------------+
```

```
+----------------------------------------+
|........................................|
|....................00..................|
|...................0..0.................|
|....................0.0.................|
|.....................0..................|
|........................................|
|........................................|
|........................................|
|................0.......................|
|................0.......................|
|................0.......................|
|........................................|
|........................................|
|.......................000..............|
|.......................0.0..............|
|.......................000..............|
|........................................|
|........................................|
|........................................|
|........................................|
|..............00........................|
|.............00...............000.......|
|...............0........................|
|............................0.....0.....|
|............................0.....0.....|
|............................0.....0.....|
|........................................|
|............00................000.......|
|............00.....................00...|
|.........................00........00...|
|.........................00.............|
|........................................|
+----------------------------------------+
```

```
+----------------------------------------+
|........................................|
|....................00..................|
|...................0..0.................|
|....................0.0.................|
|.....................0..................|
|........................................|
|........................................|
|........................................|
|........................................|
|...............000......................|
|........................................|
|........................................|
|........................0...............|
|.......................0.0..............|
|......................0...0.............|
|.......................0.0..............|
|........................0...............|
|........................................|
|........................................|
|........................................|
|.............000...............0........|
|.............0.................0........|
|..............0................0........|
|........................................|
|...........................000...000....|
|........................................|
|...............................0........|
|............00.................0........|
|............00.................0...00...|
|.........................00........00...|
|.........................00.............|
|........................................|
+----------------------------------------+
```

4.4. Buscaminas

- `http://es.wikipedia.org/wiki/Buscaminas`
- `http://en.wikipedia.org/wiki/Minesweeper_(computer_game)`

Se dispone de un tablero en el que el ordenador inicialmente distribuye aleatoriamente un número determinado de *minas*. El objetivo del jugador consiste en encontrar la posición de todas las minas del tablero, sin descubrir ninguna. El juego acabará cuando queden sin descubrir tantas casillas como minas haya (éxito), o bien cuando se descubra una mina (fallo).

Durante la ejecución del juego, en cada momento se mostrará el contador de minas restantes y la configuración actual del tablero de juego. Para jugar, el jugador irá indicando las coordenadas de las casillas que quiere descubrir, marcar como una mina o anotar como dudosa. Cada operación se indicará mediante el formato *código fila columna*, donde fila y columna son números naturales, y código es un carácter (`'d'`, `'m'`, `'?'`) que indica la operación a realizar:

- `'d'`: Descubrir una casilla
- `'m'`: Marcar que hay una mina en la casilla especificada
- `'?'`: Indicar que una casilla es dudosa

El comportamiento del juego es como se especifica a continuación:

- Al marcar el jugador que hay una mina en una determinada casilla, aparecerá un símbolo indicativo (`M`) en esa casilla, se decrementa el contador de minas y el juego prosigue normalmente.
- Al indicar que una casilla es dudosa, en esa casilla aparecerá un símbolo indicativo (`?`) y el juego prosigue normalmente.
- Al descubrir una casilla pueden suceder varias circunstancias:
  - Que levantemos una mina, en cuyo caso el jugador pierde y se acaba el juego.
  - Que la casilla esté vacía, pero tiene minas adyacentes, en cuyo caso aparecerá una cifra que indica el número total de minas que hay en los ocho cuadrados circundantes.
  - Que la casilla esté vacía, y no tiene minas adyacentes, entonces aparecerá un espacio en blanco, y se descubrirán además automáticamente todas las casillas que formen una superficie contigua sin minas. Por ejemplo, si descubrimos la casilla $(2, 7)$ ello ocasionará que se descubran automáticamente, además, todas aquellas casillas contiguas que aparecen en blanco (en las que no hay minas adyacentes) y las que delimitan dicha zona sin minas (en el ejemplo, las marcadas con número de minas 1, 2 y 3).

```
Minas: 47 Cubiertas: 161
    0 0 0 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 1
    0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9
  +----------------------------------------+
 0| X X X X X X X X X X X X X X X X X X X X |
 1| X X X X X X X X X X X X X X X X X X X X |
 2| X X X X X X X X X X X X X X X X X X X X |
 3| X X X X X X X X X X X X X X X X X X X X |
 4| X X X X X X X 2 2 3 X X X X X X X X X X |
 5| X X X X X X X 2   1 1 2 X X X X X X X X |
 6| X X X X X X X 2 1     1 1 4 X X X X X X |
 7| X X X X X X X X 1         2 X X X X X X |
 8| X X X X X 1 1 1 1         2 X X X X X X |
 9| X X X X X 1               1 X X X X X X |
  +----------------------------------------+

Introduzca codigo de operacion [d/m/?] fila y columna: d 2 7 ENTER
```

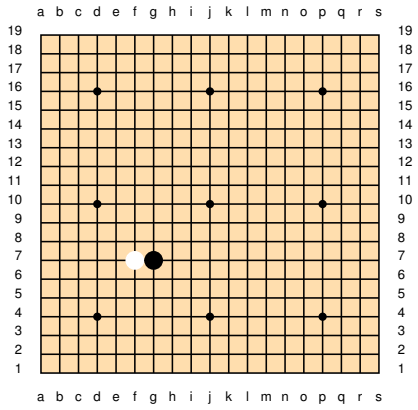
4.5. Go

- http://es.wikipedia.org/wiki/Go
- http://en.wikipedia.org/wiki/Go_(game)

El Go es un juego de mesa estratégico para dos jugadores. Es también conocido como igo (japonés), weiqi (chino) o baduk (coreano). El Go es notable por ser rico en complejas estrategias a pesar de sus simples reglas.

El juego se realiza por dos jugadores que alternativamente colocan fichas blancas y negras (`o`, `x`) sobre una cuadrícula de 19x19. El objetivo del juego es controlar una porción más grande del tablero que el oponente. Una ficha o grupo de fichas se captura y retira del juego si no tiene casillas vacías adyacentes, esto es, si se encuentra completamente rodeada de fichas del color contrario. Consulte las referencias bibliográficas para conocer las reglas completas.

```
   a b c d e f g h i j k l m n o p q r s
19 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ 19
18 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ 18
17 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ 17
16 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ 16
15 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ 15
14 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ 14
13 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ 13
12 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ 12
11 +-+-+-+-x-x-+-+-+-+-+-+-+-+-+-+-+ 11
10 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ 10
 9 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ 9
 8 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ 8
 7 +-+-+-+-+-o-o-+-+-+-+-+-+-+-+-+-+-+ 7
 6 +-+-+-+-o-+-+-o-o-+-+-+-+-+-+-+-+-+ 6
 5 +-+-+-o-x-x-+-+-o-+-+-+-+-+-+-+-+-+ 5
 4 +-+-+-o-x-x-+-o-+-o-+-+-+-+-+-+-+-+ 4
 3 +-+-+-+-o-o-o-+-+-o-+-+-+-+-+-+-+-+ 3
 2 +-+-+-+-+-+-o-o-+-+-+-+-+-+-+-+-+-+ 2   13   3
 1 +-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+ 1  [x]  [o]
   a b c d e f g h i j k l m n o p q r s   Capturas

Introduzca movimiento del jugador [o] (ENTER PASA): e3 ENTER
```



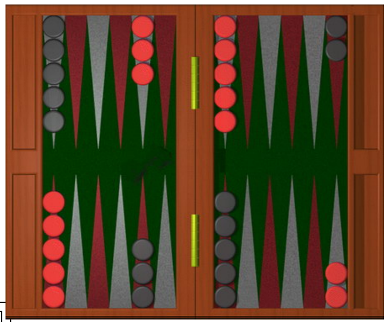
4.6. Backgammon (variante sin movimientos forzados)

- http://es.wikipedia.org/wiki/Backgammon
- http://en.wikipedia.org/wiki/Backgammon

Es un antiguo juego de mesa para dos jugadores. En las partidas cada jugador tiene quince fichas que va moviendo entre veinticuatro triángulos (cajas) según lo que salga en sus dos dados. El objetivo del juego es ser el primero en conseguir mover sus quince fichas fuera del tablero.

Para liberar las fichas del tablero, el jugador 1 debe avanzar las piezas hacia el cuadrante inferior derecho, las piezas se mueven hacia la izquierda en la parte superior y prosiguen de izquierda a derecha en la parte inferior, desde el triángulo 24 hasta el 1. Por el contrario, el jugador 2 moverá las fichas en el orden inverso (desde el triangulo 1 hasta el 24). Consulte las referencias bibliográficas para conocer las reglas completas.
Nota: Es posible simplificar el problema y no considerar las situaciones de movimientos obligatorios.

```
+-+------------------------+-+------------------------+-+
| | 13  14  15  16  17  18 | | 19  20  21  22  23  24 |o|
+-+------------------------+-+------------------------+-+
| || x||  ||  ||  ||o ||  || ||o ||  ||  ||  ||  || x|| |
| || x||  ||  ||  ||o ||  || ||o ||  ||  ||  ||  || x|| |   +-------+
| || x||  ||  ||  ||o ||  || ||o ||  ||  ||  ||  ||  || |   | o   o |
| || x||  ||  ||  ||  ||  || ||o ||  ||  ||  ||  ||  || |   |   o   |
| || x||  ||  ||  ||  ||  || ||o ||  ||  ||  ||  ||  || |   | o   o |
| |\  /\  /\  /\  /\  /\  /| |\  /\  /\  /\  /\  /\  /| |   +-------+
| | \/  \/  \/  \/  \/  \/ | | \/  \/  \/  \/  \/  \/ | |
| |                        | |                        | |
| |                        | |                        | |
| | /\  /\  /\  /\  /\  /\ | | /\  /\  /\  /\  /\  /\ | |
| |/  \/  \/  \/  \/  \/  \| |/  \/  \/  \/  \/  \/  \| |
| ||o ||  ||  ||  ||  ||  || || x||  ||  ||  ||  ||  || |   +-------+
| ||o ||  ||  ||  ||  ||  || || x||  ||  ||  ||  ||  || |   | o     |
| ||o ||  ||  ||  || x||  || || x||  ||  ||  ||  ||  || |   |   o   |
| ||o ||  ||  ||  || x||  || || x||  ||  ||  ||o ||  || |   |     o |
| ||o ||  ||  ||  || x||  || || x||  ||  ||  ||o ||  || |   +-------+
+-+------------------------+-+------------------------+-+
| | 12  11  10   9   8   7 | |  6   5   4   3   2   1 |x|
+-+------------------------+-+------------------------+-+


jugador [x], introduzca dado [1,2] y casilla de la ficha a mover : 1 13 ENTER

jugador [x], introduzca casilla de la ficha a mover : 8 ENTER
```



4.7. Bunker

Es un juego interactivo entre dos jugadores, donde cada jugador dispone de un bunker en una determinada posición. El juego consiste en que ambos jugadores, por turnos, realizan un disparo de *mortero* con un determinado ángulo y fuerza de disparo, considerando la fuerza del viento (que puede ser a favor o en contra) y la posición del objetivo. El disparo de mortero tiene en cuenta las leyes físicas de la trayectoria parabólica. Gana el juego el primer jugador que consiga destruir el bunker del jugador contrario.

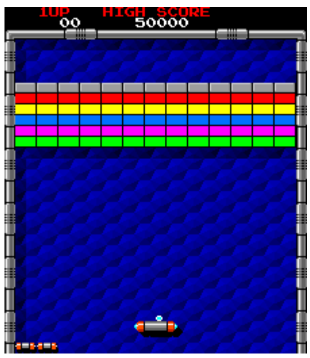
```
Jugador 1: Fuerza del Viento: -18
Angulo [ + - Intro ]: 13  |=#---------| Fuerza [ + - Intro ]: 40  |====#------|



                               oooooooo
                      oooooooooo      oooooo
                ooooooo                    oooo
           oooooo
       ooooo
       o
    -------
   / o o o \
   |___[]__|
   Jugador 1
                                                                 -------
                                                                / o o o \
                                                                |__[]___|
                                                                Jugador 2
```

4.8. Arkanoid

- http://es.wikipedia.org/wiki/Arkanoid
- http://en.wikipedia.org/wiki/Arkanoid

Arkanoid es un videojuego de arcade desarrollado por Taito en 1986. El jugador controla una pequeña plataforma, que impide que una bola salga de la zona de juego, haciéndola rebotar. En la parte superior hay *ladrillos* o *bloques*, que desaparecen al ser tocados por la bola. Cuando no queda ningún ladrillo, el jugador pasa al siguiente nivel, donde aparece otro patrón de bloques. Consulte las referencias bibliográficas para conocer las reglas completas.

```
+------------------------------------------------------------------------+
|                                                                        |
|       ##### ##### ##### ##### ##### ##### ##### ##### ##### #####      |
|                                                                        |
|       ##### ##### ##### ##### ##### ##### ##### ##### ##### #####      |
|                                                                        |
|       ##### ##### ##### ##### ##### ##### ##### ##### ##### #####      |
|                                                                        |
|       ##### ##### ##### ##### ##### ##### ##### ##### ##### #####      |
|                                                                        |
|                                                                        |
|                                                                        |
|        O                                                               |
|                                                                        |
|                                                                        |
|                                                                        |
|                                                                        |
|        ==========                                                      |
|                                                                        |
Puntuacion: 0                                Nivel: 1      Vidas: 5
```

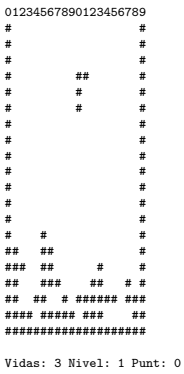


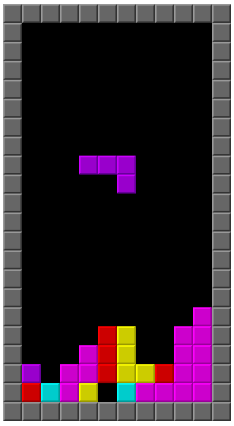
4.9. Tetris

- http://es.wikipedia.org/wiki/Tetris
- http://en.wikipedia.org/wiki/Tetris

Tetris es un videojuego de puzle inventado por el ruso Alekséi Pázhitnov el 6 de junio de 1984[1] cuando estaba trabajando en la Academia de Ciencias de Moscú.

Distintos tetriminos, figuras geométricas compuestas por cuatro bloques cuadrados unidos de forma ortogonal, caen de la parte superior de la pantalla. El jugador no puede impedir esta caída pero puede decidir la rotación de la pieza (0°, 90°, 180°, 270°) y en qué lugar debe caer. Cuando una línea horizontal se completa, esa línea desaparece y todas las piezas que están por encima descienden una posición, liberando espacio de juego y por tanto facilitando la tarea de situar nuevas piezas. La caída de las piezas se acelera progresivamente. El juego acaba cuando las piezas se amontonan hasta salir del área de juego. Consulte las referencias bibliográficas para conocer las reglas completas.

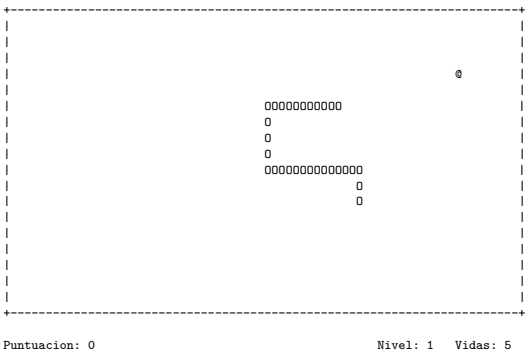


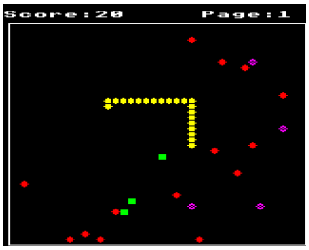


4.10. Serpiente

- http://es.wikipedia.org/wiki/Snake_(videojuego)
- http://en.wikipedia.org/wiki/Snake_(video_game)

Snake es un videojuego lanzado a mediados de los 70. El jugador controla una larga y delgada criatura, semejante a una serpiente, que vaga alrededor de un plano delimitado, recogiendo alimentos (o algún otro elemento), tratando de evitar golpear a su propia cola o las "paredes"que rodean el área de juego. Cada vez que la serpiente se come un pedazo de comida, la cola crece más, provocando que aumente la dificultad del juego. El usuario controla la dirección de la cabeza de la serpiente (arriba, abajo, izquierda o derecha) y el cuerpo de la serpiente la sigue. Además, el jugador no puede detener el movimiento de la serpiente, mientras que el juego está en marcha. Consulte las referencias bibliográficas para conocer las reglas completas.





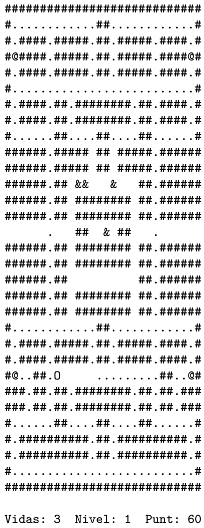
4.11. Pacman

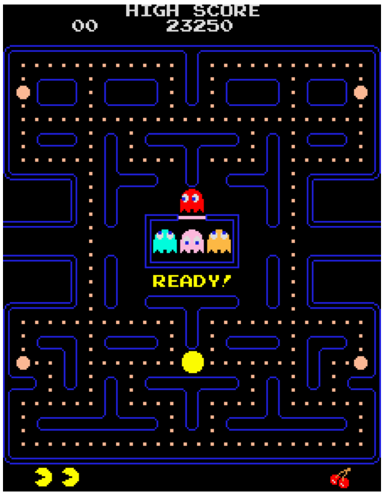
- http://es.wikipedia.org/wiki/Pac-Man
- http://en.wikipedia.org/wiki/Pacman

Pac-Man es un videojuego arcade creado por el diseñador de videojuegos Toru Iwatani. El protagonista del videojuego Pac-Man es un círculo amarillo al que le falta un sector por lo que parece tener boca. Aparece en laberintos donde debe comer puntos pequeños, puntos mayores y otros premios con forma de frutas y otros objetos. El objetivo del personaje es comer todos los puntos de la pantalla, momento en el que se pasa al siguiente

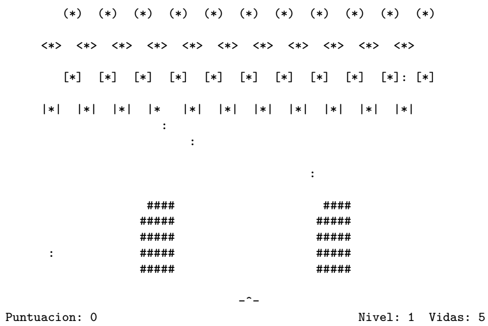
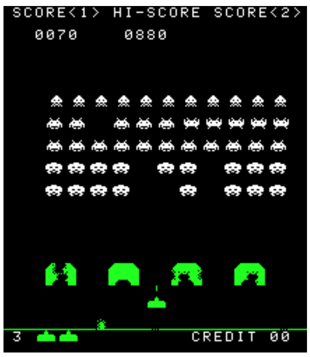
nivel o pantalla. Sin embargo, cuatro fantasmas o monstruos, recorren el laberinto para intentar comerse a Pac-Man. Los fantasmas no son iguales, así mientras uno es muy rápido, y tiene la habilidad de encontrar a Pac-Man en el escenario, otro es muy lento y muchas veces evitará el encuentro con Pac-Man.

Hay cuatro puntos más grandes de lo normal situados cerca de las esquinas del laberinto, que proporcionan a Pac-Man la habilidad temporal de comerse a los monstruos (todos ellos se vuelven azules mientras Pac-Man tiene esa habilidad). Después de haber sido tragados, los fantasmas se regeneran en *casa* (una caja situada en el centro del laberinto). El tiempo en que los monstruos permanecen vulnerables varía según la pantalla, pero tiende a decrecer a medida que progresa el juego. Consulte las referencias bibliográficas para conocer las reglas completas.





4.12. Space Invaders

- `http://es.wikipedia.org/wiki/Space_Invaders`
- `http://en.wikipedia.org/wiki/Space_Invaders`

Space Invaders es un videojuego de arcade diseñado por Toshihiro Nishikado y lanzado al mercado en 1978. Su objetivo es eliminar oleadas de alienígenas con un cañón láser y obtener la mayor cantidad de puntos posible. El juego tiene una estructura muy simple pero apasionante. El jugador controla un cañón que puede moverse a la derecha o izquierda y un botón de disparo. Tiene que ir destruyendo los aliens invasores que van acercándose a la tierra cada vez más rápidamente. Cada cierto tiempo aparece en la pantalla, por encima de los invasores, un platillo volador que se mueve aleatoriamente de derecha a izquierda o de izquierda a derecha y que no agrega una puntuación definida, sino que los puntos que otorga cambian cada vez. Además se tienen cuatros escudos de protección terrestre (más parecidos a búnkeres) que cubren al jugador del fuego alienígena, aunque también le dificultan disparar desde detrás de ellos. Consulte las referencias bibliográficas para conocer las reglas completas.