

Guion de prácticas

MPALABRADOS (*bag*)

Febrero 2020



Metodología de la Programación

DGIM

Curso 2019/2020

Índice

1. Descripción	5
1.1. El juego del Scrabble	5
1.2. El juego del MPALABRADOS	5
2. Arquitectura de la práctica	6
3. Práctica a entregar	9
3.1. Configuración de la práctica	9
3.2. Ejecución de prueba	11
3.3. Validación de la práctica	11
3.4. Entrega de la práctica	11
4. Codificaciones de caracteres ISO8859-1 y UTF8	13
5. Código de la práctica	16
5.1. Clase Bag	16
5.2. Clase Language	17
6. Una jugada de ejemplo	19

1. Descripción

Esta primera práctica es la primera pieza en la arquitectura del juego “SCRABBLE” implementado en C++ de forma simplificada. Básicamente es un juego de tablero de palabras cruzadas en la que cada letra tiene una puntuación y cada tirada tiene la puntuación acumulada de la palabra que se haya colocado y la puntuación acumulada de todas las palabras nuevas que se formen al cruzarse las letras que ya hay en el tablero con las de la última tirada.

1.1. El juego del Scrabble

Puede consultar más información sobre este juego y sus normas básicas en los siguientes recursos.

- Web de HASBRO con las reglas oficiales del juego
<https://scrabble.hasbro.com/en-us/rules>
- Youtube. “How to play Scrabble”
<https://www.youtube.com/watch?v=swlg3vQXboE>

1.2. El juego del MPALABRADOS

En el caso de esta práctica, se va a seguir una versión simplificada del juego, que vamos a llamar “MPALABRADOS” y que tiene las siguientes características diferenciadoras.

1. Es un juego para un único jugador
2. No existen dobles o triples tantos de letra o palabra.
3. No es obligatorio colocar la primera palabra sobre la casilla central.
4. Soporta tres idiomas (se puede extender a más idiomas o actualizar los que hay de forma sencilla).
5. Soporta tableros de juego de cualquier dimensión.

La Figura 1 muestra algunos de los componentes principales de este juego: A) El tablero con las letras. (B) Las fichas del jugador. (C) La bolsa de letras ordenada aleatoriamente en cada partida. (D) La lista de movimientos registrados.

En la Sección 6 se muestra el inicio de una jugada, paso a paso, interaccionando los componentes anteriores.

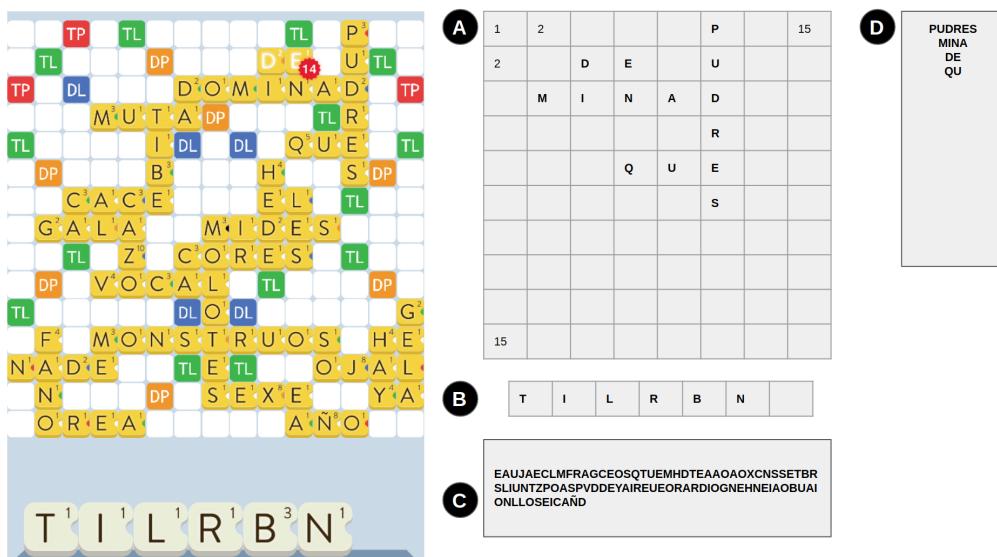


Figura 1: Principales componentes de MPALABRADOS. (A) El tablero con las letras. (B) Las fichas del jugador. (C) La bolsa de letras ordenada aleatoriamente en cada partida. (D) La lista de movimientos registrados.

2. Arquitectura de la práctica

El juego MPALABRADOS se ha diseñado como una arquitectura por capas, en la que las capas más internas de la misma representan las estructuras más sencillas, sobre las cuales se asientan las capas más externas, con las estructuras más complejas. La Figura 2 muestra el diseño de la arquitectura que se va a emplear y que se va a ir desarrollando progresivamente en esta y las siguientes sesiones de prácticas. Toda la práctica está basada en el manejo de cadenas de caracteres, pero se ha organizado de la siguiente forma. La función **main()** y todas sus funciones accesorias trabajarán siempre, con la clase **string**. Internamente cada clase de la arquitectura que trabaje con cadenas de caracteres lo hará con una representación distinta (vector, cstring, string), aunque compartirán los datos entre ellas sólo como **string**.

C bag.cpp

Implementa la bolsa de letras disponible para el juego. Cada letra tiene una frecuencia de aparición en la bolsa que depende del idioma elegido. Se dispone de la biblioteca **Language** ya implementada por los profesores para realizar las tareas de carga del idioma, diccionario de palabras válidas, consulta de puntos de letras y consulta de frecuencias de letras.

B player.cpp

Implementa la estructura para almacenar las 7 letras de que dispone el jugador, métodos para tener esta estructura ordenada y para extraer y añadir letras conforme el juego avanza.

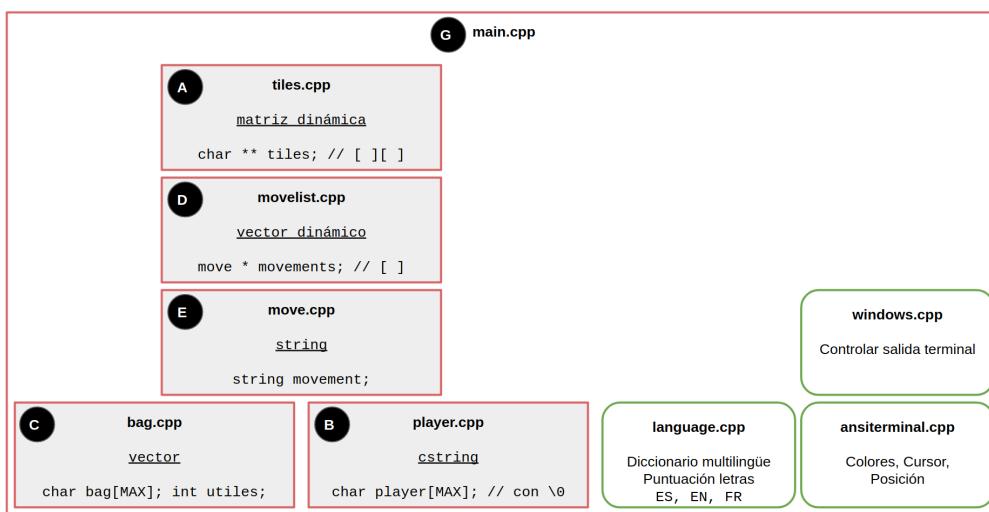


Figura 2: Arquitectura de clases de MPALABRADOS

E move.cpp

Estructura para almacenar una jugada y su relación con las dos estructuras anteriores.

D movelist.cpp

Estructura para almacenar una conjunto, de tamaño indefinido, de movimientos.

A tiles.cpp

Implementa la estructura para almacenar el tablero de juego y las letras que se encuentran ya colocadas y es la clave para calcular las palabras cruzadas.

G main.cpp

Será el cuerpo principal del programa el cual se basa en dos bibliotecas adicionales, ya implementadas, para visualizar de forma colorida el desarrollo del juego.

Este trabajo progresivo se ha planificado en hitos sucesivos con entregas en Prado, tal y como muestra la Figura 3.

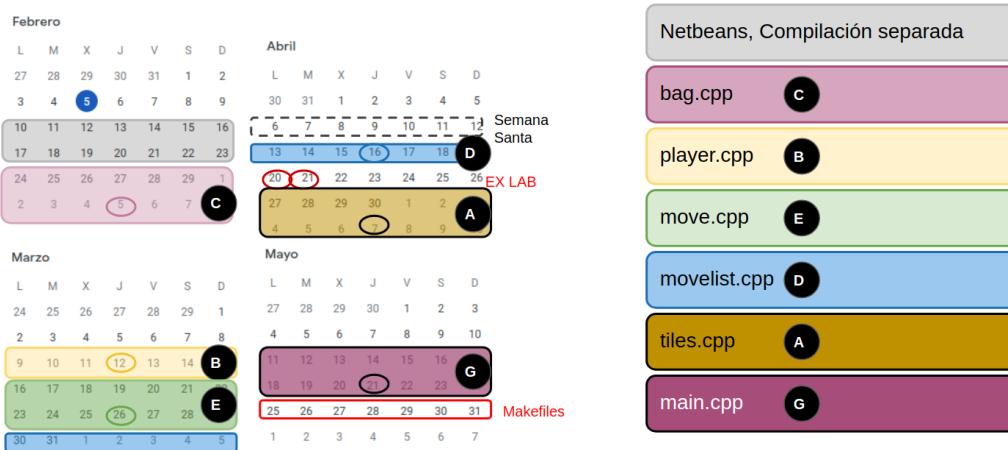


Figura 3: Plan de trabajo y entregas programadas (sujetas a posibles cambios)

3. Práctica a entregar

Se deberá crear un proyecto Netbeans compuesto de los siguientes ficheros fuente incompletos (en todos ellos aparece la marca **@warning** avisando de las tareas de implementación que están pendientes).

- **bag.h**

Revisar y completar las declaraciones de métodos y parámetros para que pueda funcionar correctamente.

- **bag.cpp**

Implementar todos los métodos y funciones que faltan. Prestar atención al funcionamiento de los métodos y funciones descrito en los comentarios.

- **main.cpp**

Completar el código para realizar el siguiente programa.

1. Leer un string con el ID del lenguaje, según el estándar ISO639-1.
2. Crear una instancia de la clase **Language** con el anterior ID y mostrar el conjunto de caracteres permitido para ese lenguaje.
3. Pedir un número entero.
4. Crear una instancia de la clase **Bag**, inicializar el generador de números aleatorios con el número anterior y definir su contenido en base al lenguaje que se ha declarado anteriormente.
5. Sacar todos los caracteres de la bolsa recién definida, en grupos de 5, y, por cada grupo de 5 letras (o menos) extraído, imprimirla en la pantalla, comprobar si la palabra resultante, en ese mismo orden de las letras, existe en el diccionario del lenguaje creado y, si es así, marcarla con unos asteriscos.
6. Una vez agotada la bolsa, el programa debe indicar cuántas de las palabras, aleatoriamente extraídas de la bolsa, forman una palabra válida en el lenguaje definido.
7. Para poder utilizar la script de autovalidación **doTests.sh** se recomienda terminar el programa con una llamada a la función **HallOfFame(...)** la cual recibe estos datos y la muestra por pantalla de forma detectable por la script.

3.1. Configuración de la práctica

Se debe preparar una carpeta exclusivamente para proyectos Netbeans desde donde se colgarán todos los proyectos de la asignatura. En esa carpeta base se debe crear la carpeta **local**, en la que colocar los ficheros **.h** y **.a** que se suministren a lo largo de la asignatura, tal y como se indica en el guión de NetBeans.

A continuación, crear el proyecto para la práctica desde NetBeans, según se indica en el guión de Netbeans, y copiar cada fichero disponible en Prado en su lugar correspondiente según muestra el siguiente listado.

```
Netbeans
|-- local
|   |-- include
|   |   |-- language.h
|   |   '-- wordlist.h
|   '-- lib
|       '-- liblanguage.a
|-- MP1920Geometry
|-- MP1920Practical
|   |-- build
|   |-- dist
|   |-- doc
|   |   '-- documentation.dox
|   |-- include
|   |   '-- bag.h
|   |-- languages
|   |   |-- EN.scrabble
|   |   |-- EN.tree
|   |   |-- EN.tree.words
|   |   |-- ES.scrabble
|   |   |-- ES.tree
|   |   |-- ES.tree.words
|   |   |-- FR.scrabble
|   |   |-- FR.tree
|   |   '-- FR.tree.words
|   |-- Makefile
|   |-- nbproject
|   |-- scripts
|   |   |-- doConfig.sh
|   |   |-- doDoxygen.sh
|   |   |-- doTests.sh
|   |   |-- doUpdate.sh
|   |   '-- doZipProject.sh
|   |-- src
|   |   |-- bag.cpp
|   |   '-- main.cpp
|   |-- tests
|   |   |-- EN_8784_3.test
|   |   |-- ES_100_0.test
|   |   |-- ES_136_2.test
|   |   |-- ES_60143_3.test
|   |   |-- ES_RANDOM_0.test
|   |   |-- FR_66727_3.test
|   |   '-- US_EXCEPTION.test
|   '-- zip
`-- MP1920Practica2
```

3.2. Ejecución de prueba

ID=“ES”, ENTERO=16

```
TYPE LANGUAGE: ES
Opening tree file ./languages/ES.tree
Trying to read 48428 words
OK 48428 words read
Opening ./languages/ES.scrabble
OK 25 Scrabble's letter read
ALLOWED LETTERS: LTNRUISOAEGDBMP CFVYHQJÑXZ
TYPE SEED (<0 RANDOM) : 16
BAG (16-95) : CCATMAVARODGDAPNEEAQZCINOUP OIEOGDBARBEOTOATA
UEDLASAEONHORXAEEUOMSISJYILLIENTFERISNDÑUSELUHARESC
CCATM
AVARO ***
DGDAP
NEEAQ
ZCINO
UPOIE
OGDBA
RBEOT
OATAU
EDLAS
AOHN
ORXAE
EUOMS
ISJYI
LLIEN
TFERI
SNDÑU
SELUH
ARESC

%%%OUTPUT
LANGUAGE: ES ID: 16
BAG (0):
1 words found
AVARO -
```

3.3. Validación de la práctica

Se debe ejecutar la script **doTests.sh** y comprobar que los resultados que aparecen por pantalla coinciden con lo indicado en cada caso de validación.

3.4. Entrega de la práctica

Se deberá ejecutar la script **doZipProject.sh** y subir a Prado, en las fechas que se indican en la temporización de la asignatura, el zip resultante, que está almacenado en la carpeta **.zip** del proyecto de Netbeans y siempre se llama **MPPRACTICA.zip**. Para saber si el zip es correcto, seguir los siguientes pasos.

```
lcv:MP$ cd NetBeans/
lcv:NetBeans$ mkdir Test
lcv:NetBeans$ cd Test
lcv:Test unzip <especificar-ruta>/MPPractica.zip
Archive: ... MPPractica.zip
...<comprobar-descipción>...
lcv:Test make clean
lcv:Test ./scripts/doTests.sh
Generating fresh binaries
...<comprobar-compilación>...
Press [RETURN] to continue
Test #1 (mp1920practical < tests//XXX.test)
...<comprobar-tests>...

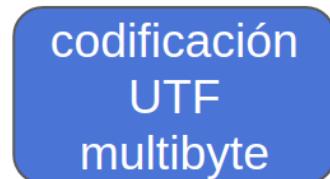
Test #2 (mp1920practical < tests//YYY.test)
...<comprobar-tests>...
```

4. Codificaciones de caracteres ISO8859-1 y UTF8

Cuando se trabaja con caracteres internacionales, la tabla de caracteres de 256 posiciones se queda muy corta y es necesario una codificación alternativa.



La más común en los Sistemas Operativos modernos es Unicode UTF¹, la cual puede usar 1, 2 o más bytes para representar un único carácter multinacional.



Bits of code point	First code point	Last code point	Bytes in sequence	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6
7	U+0000	U+007F	1	0xxxxxx					
11	U+0080	U+07FF	2	110xxxxx	10xxxxxx				
16	U+0800	U+FFFF	3	1110xxxx	10xxxxxx	10xxxxxx			
21	U+10000	U+1FFFFFF	4	11110xxx	10xxxxxx	10xxxxxx	10xxxxxx		
26	U+200000	U+3FFFFFF	5	111110xx	10xxxxxx	10xxxxxx	10xxxxxx	10xxxxxx	
31	U+4000000	U+7FFFFFFF	6	1111110x	10xxxxxx	10xxxxxx	10xxxxxx	10xxxxxx	10xxxxxx

Por simplicidad en la asignatura (habría que alterar los bucles y la forma de recorrer una cadena para comparar carácter carácter, se va a usar otra codificación multilingüe llamada ISO8859-1² la cual usa un único byte pero solo puede codificar los caracteres especiales latinos. Aunque con esta codificación es más sencillo recorrer las cadenas de caracteres, razón por la cual se ha decidido utilizar esta codificación en las prácticas de la asignatura.

¹Codificación UTF8 <https://es.wikipedia.org/wiki/UTF-8>

²Codificación ISO8859-1 https://es.wikipedia.org/wiki/ISO/IEC_8859-1

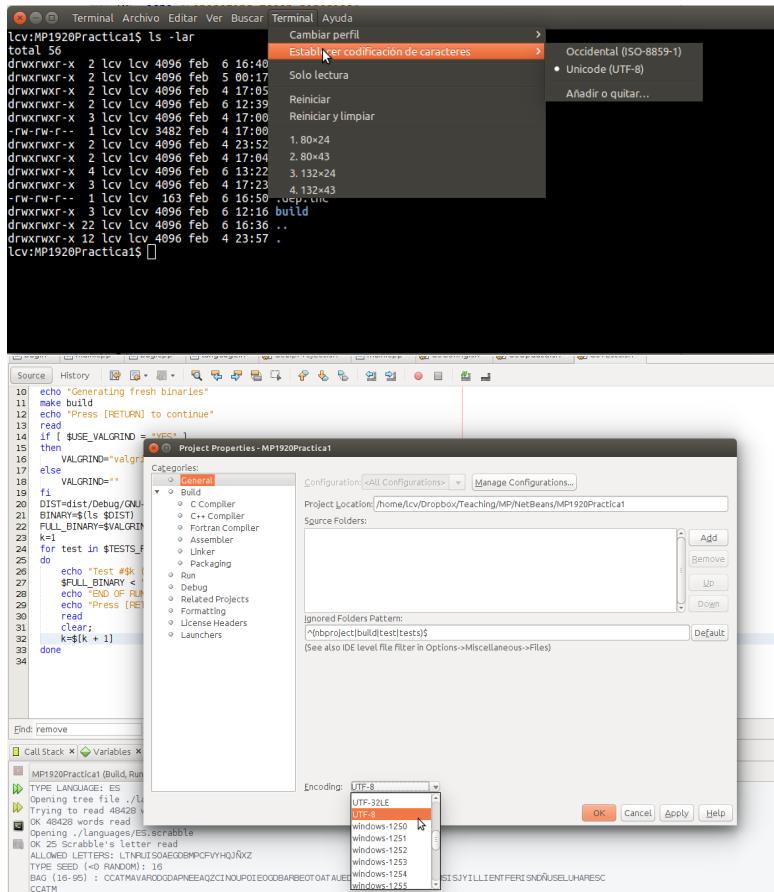


Figura 4: Selección de la codificación de caracteres en varios programas. Arriba la terminal de órdenes. Abajo NetBeans

codificación
ISO8859
byte

À	À	à	À	À	À	À	À	À	À	À	À	À	À	À	À	À	À	À
È	È	è	È	È	È	È	È	È	È	È	È	È	È	È	È	È	È	È
Ò	Ò	ò	Ò	Ò	Ò	Ò	Ò	Ò	Ò	Ò	Ò	Ò	Ò	Ò	Ò	Ò	Ò	Ò
Ã	Ã	ã	Ã	Ã	Ã	Ã	Ã	Ã	Ã	Ã	Ã	Ã	Ã	Ã	Ã	Ã	Ã	Ã
Ñ	Ñ	ñ	Ñ	Ñ	Ñ	Ñ	Ñ	Ñ	Ñ	Ñ	Ñ	Ñ	Ñ	Ñ	Ñ	Ñ	Ñ	Ñ

La mayoría de programas que manejan texto multinacional permiten combinar ambas opciones, pero lo más común es que utilicen UTF8.

Por ello, a lo largo de esta práctica será necesario tener en cuenta lo siguiente. Cuando se lea una cadena desde un stream de C++ (cin) y se vaya a usar esta cadena para jugar al MPALABRADOS, será necesario traducirla de UTF a ISO mediante la función **toISO(...)** incluida el

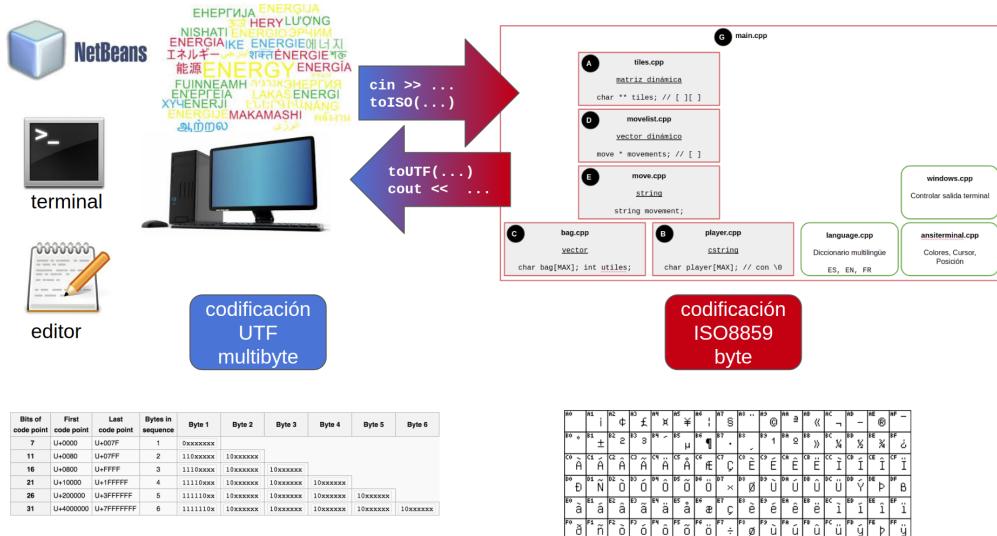


Figura 5: Diferentes codificaciones de caracteres y uso de las funciones `toISO(...)` y `toUTF(...)`

la biblioteca **Language** que acompaña a esta práctica en Prado. De la misma forma, cuando sea necesario escribir en un stream (cout) será necesario volverla a traducir a UTF con la función `toUTF(...)`. La biblioteca Language implementa varias formas de traducir de ISO a UTF y viceversa dependiendo de si lo que se quiere traducir es un carácter individual, un cstring o un string.

5. Código de la práctica

5.1. Clase Bag

```
/*
 * @file bag.h
 * @author DECSAI
 * @note To be implemented by students
 */
#ifndef BAG_H
#define BAG_H

#include <string>
#include "language.h"

#define MAXBAG 200    /// Max size of the bag of letters

///@warning: Check const methods and parameters
///

/***
 * @class Bag
 * @brief Class to represent the full set of letters of the Scrabble. The
 * number of letters , their score and their frequencies is determined by the
 * rules of the game.
 *
 * More details https://en.m.wikipedia.org/wiki/Scrabble\_letter\_distributions
 *
 * **Please note that all the characters are stored in ISO8859 standard and in uppercase**
 */
class Bag {
private:
char letters[MAXBAG];    /// Set of letters of the game
int nletters;           /// Number of available letters
int id;                 /// Aleatorio para el juego
/***
 * @brief Returns the position p of the bag
 * @param p The position requested
 * @precond p must be a legal position
 * @return The character at the position p (ISO)
 */
char get(int p);
/***
 * @brief Sets the position p of the bag
 * @param p The position requested
 * @param c The character to insert (ISO)
 * @precond p must be a legal position
 */
void set(int p, char c);
public:
/***
 * @brief Basic constructor and initializer
 */
Bag();
/***
 */
void setRandom(int );
/***
 * @brief Define the content of the bag of letters according to
 * the rules of the language. See language.h for more details on how to
 * arrange this set of letters.
 * - Language::getLetterSet() returns the set of letters of a given Language
 * - Language::getFrequency(char letter) returns the frequency of a letter , that is , the number of times
 * that each letter appears in the bag
 * @param l The chosen language
 */
void define(Language );
/***
 * @brief Size of the letters set that remain in the bag
 * @return Number of letters
 */
int size();
/***
 * @brief Returns the set of letters that remain in the bag
 * @return The set of letters
 */
std::string to_string();
/***
 * @brief Define , in one step , the full set of letters in the bag
 * @param s A string that contains all the letters , and their repetitions
 */
void set(std::string s);
/***
 * @brief Extracts the first @p n letters remaining in the bag. If the bag
 * contains less than the required @p n letters , it returns all the remaining letters .
 * @param n Number of letters to extract
 * @return The set of the first @p n letters in the bag
 */
std::string extract(int n);
};

#endif /* BAG_H */
```

5.2. Clase Language

```
/*
 * @file language.h
 * @author DECSAI
 * @brief Fully functional static library to handle languages, which are
 * represented as a full list of allowed words, stored as a tree to make
 * search efficient O(n) being n the number of letters in the word to
 * be looked up
 * @note Fully implemented. No further implementation required.
 */

#ifndef LANGUAGE_H
#define LANGUAGE_H
#include <vector>
#include "wordlist.h"

/**
 * @class Language
 * @brief Class fully implemented. It is used to store and manage all the details concerning
 * a given language. It includes and make it publicly available some functions in wordlist.h
 * like those to change the encoding of characters. **All the characters stored in memory use the ISO8859
 * standard**
 * but characters read from keyboard might follow the UTF standard. These functions allow to change from one
 * to another.
 *
 * **Please note that all characters are stored in uppercase**
 */
class Language{
public:

/**
 * @brief Basic constructor and initializer
 */
Language();

/**
 * @brief Basic constructor and initializer
 * @param language Language chosen according to international rules ISO639-1 https://en.wikipedia.org/wiki/List\_of\_ISO\_639-1\_codes
 * @note If the language chosen does not exists in the folder <root>/languages it throws an exception and
 * stops the program
 */
Language(std::string language);

/**
 * @brief Query if a given word exists in the given language
 * @param word The word to be queried
 * @return @retval true if the word exists in the recorded language, @retval false otherwise
 */
bool query(std::string word) const;

/**
 * @brief Returns the ISO690 identifier of the language
 * @return A string with the ID of the language https://en.wikipedia.org/wiki/List\_of\_ISO\_639-1\_codes
 */
std::string getLanguage() const;

/**
 * @brief Loads the chosen language, which must be under <root>/languages folder
 * @param lang The ID of the language
 * @note If the language chosen does not exists in the folder <root>/languages it throws an exception and
 * stops the program
 */
void setLanguage(std::string lang);

/**
 * @brief Query the frequency of appearance in Scrabble of the given letter, according to the chosen language
 * @param letter The letter to query
 * @return The frequency of appearance of the letter in the Scrabble of the given language. It returns
 * @retval 0 if the letter does not appear in the language
 */
int getFrequency(char letter) const;

/**
 * @brief Query the score in Scrabble of the given letter, according to the chosen language
 * @param letter The letter to query
 * @return The number of points of the letter in the Scrabble of the given language. It returns @retval 0 if
 * the letter does not appear in the language
 */
int getScore(char letter) const;

/**
 * @brief Query the full set of available letters (without repetitions) in a given language
 * @return A string containing the letters available in ISO8859
 */
std::string getLetterSet() const;

private:
    // Deliberately left out
};

#endif
```

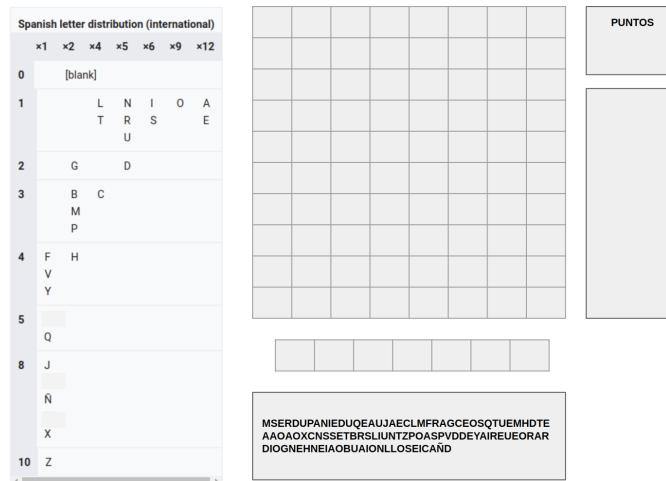
Las siguientes funciones también están incluidas en **language.h** y se pueden utilizar para cambiar las codificaciones de los caracteres.

```
/**  
 * @brief Translate any word into valid characteres only  
 * @param word  
 * @return ISO Word  
 */  
std::string normalizeWord(const std::string & word);  
  
/**  
 * @brief Translate a ISO char into UTF  
 * @param in the ISO char  
 * @return the UTF char(s)  
 */  
std::string toUTF(char in);  
  
/**  
 * @brief Translate a ISO string into a UTF string  
 * @param in the ISO string  
 * @return the UTF string  
 */  
std::string toUTF(std::string in);  
  
/**  
 * @brief Translate a ISO cstring into a UTF string  
 * @param in the ISO cstring  
 * @return the UTF string  
 */  
std::string toUTF(const char * in);  
  
/**  
 * @brief Translate a UTF char into ISO  
 * @param in the UTF char  
 * @return the ISO char  
 */  
char toISO(char in);  
  
/**  
 * @brief Translate a UTF string into a ISO string  
 * @param in the UTF string  
 * @return the ISO string  
 */  
std::string toISO(std::string in);  
  
/**  
 * @brief Translate a UTF cstring into a ISO string  
 * @param in the UTF cstring  
 * @return the ISO string  
 */  
std::string toISO(const char * in);
```

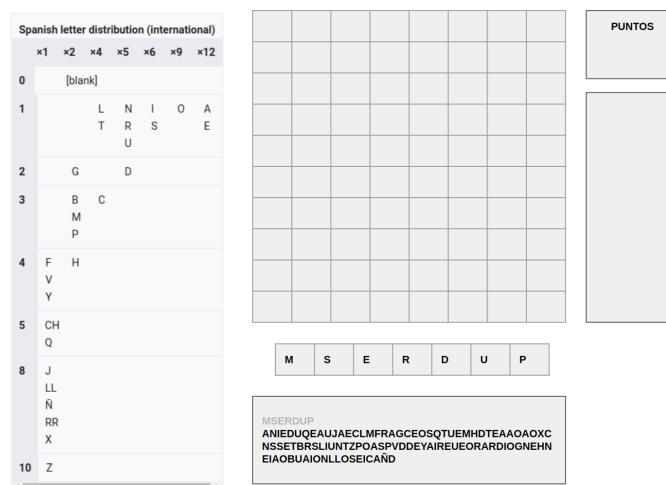
6. Una jugada de ejemplo

Las siguientes figuras ilustran, paso a paso, el comportamiento de MPALABRADOS.

1. Estado inicial del tablero vacío y la bolsa con las letras iniciales, dadas por la tabla de frecuencias de letras del idioma.



2. Se sacan las primeras 7 letras al jugador.



3. Primera tirada de 9 puntos. Se calcula la suma de la puntuación de cada letra conseguida, según la tabla oficial de Scrabble.

4. El jugador siempre dispone de 7 letras, así que se vuelven a sacar 6 letras para llenar las que el jugador ha empleado en la tirada. En caso de que en la bolsa queden menos de las letras necesarias, se sacarían todas.

5. Segunda tirada que consigue 8 puntos más. Como la palabra se cruza con otra, se suman las letras de la palabra total cruzada, es decir, aunque la tirada es MINA, se puntúa MINAD.

6. Se vuelven a reponer las fichas del jugador.

7. La tirada es DE pero produce 3 cruces diferentes. La puntuación de la tirada es la suma de las puntuaciones de todos los cruces. Si algún cruce produjese una palabra no contenida en el diccionario, entonces la tirada sería nula.

8. Última tirada QU, que produce la palabra QUE

