Pontificia Universidad Javeriana
Departamento de Ingeniería de Sistemas
Estructuras de Datos
Proyecto del curso, 2024-10

# 1 Descripción del problema

**Scrabble** es un juego de mesa, patentado en 1948 y actualmente producido por las empresas *Hasbro* y *Mattel*. El objetivo es ganar la mayor cantidad de puntos al construir palabras sobre un tablero de 15x15 casillas. Las palabras pueden ubicarse sobre el tablero de forma horizontal o vertical (leyéndose de izquierda a derecha y de arriba a abajo), y pueden cruzarse entre sí, como en un crucigrama. La puntuación de cada palabra depende de los puntos definidos para cada letra, y de la ubicación de la palabra en el tablero, ya que algunas casillas contienen modificadores de puntuación, que duplican o triplican los puntos de la letra o la palabra. Los puntos asignados a cada letra dependen de su frecuencia de aparición en el idioma, en una relación inversa: a más frecuencia de aparición, menos puntos asignados.

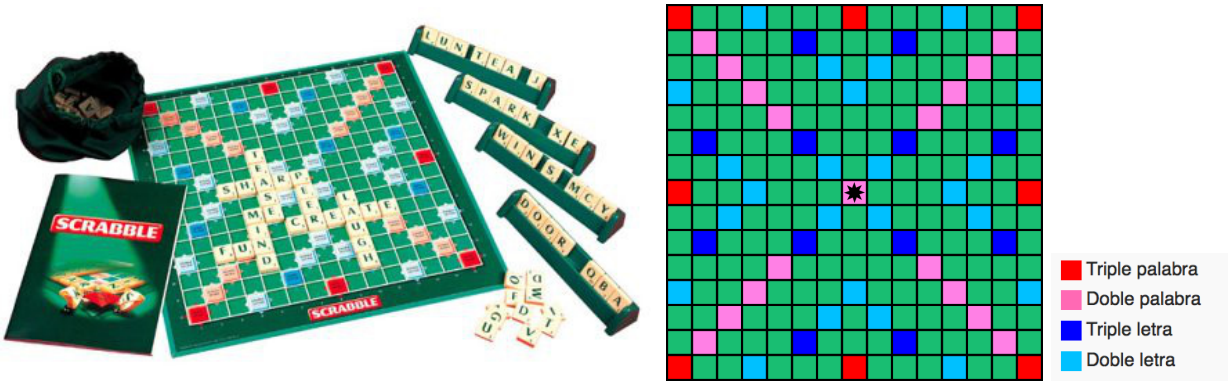


Figure 1: Izquierda: Juego completo de **Scrabble**. Derecha: Tablero de **Scrabble**, junto con las convenciones de las casillas especiales.

Ej juego admite entre 2 y 4 jugadores. En cada turno, el jugador tiene 7 letras extraídas aleatoriamente de una bolsa, ubicadas en un soporte. El jugador debe construir una palabra con las letras de su soporte, intentando obtener la mayor puntuación al ubicarla en el tablero. Esto implica analizar muchas posibilidades de combinación de las letras, así como revisar letras de palabras ya puestas en el tablero que permitan combinaciones. Adicionalmente, es importante verificar que la palabra sea válida en el idioma en el que se está jugando, para lo cual se definen ciertas reglas de validez de palabras, usando un diccionario estándar. Para el juego en su versión original (en idioma inglés), las siguientes son las puntuaciones individuales de cada letra:

- 1 punto: E, A, I, O, N, R, T, L, S, U.
- 2 puntos: D, G.
- 3 puntos: B, C, M, P.
- 4 puntos: F, H, V, W, Y.
- 5 puntos: K.
- 8 puntos: J, X.
- 10 puntos: Q, Z.

Para ayudar al jugador en la construcción de la mejor palabra en su jugada, se han desarrollado herramientas de apoyo, conocidas como constructores de palabras de **Scrabble**. Algunos ejemplos son:

- `https://word.tips/scrabble-word-finder/`
- `https://scrabblewordfinder.org/`
- `https://wordfind.com/`

Sólo es necesario ingresar las letras que se tienen en el soporte, y en algunos casos alguna palabra ya existente en el tablero o letras comodines, y el sistema presenta todas las posibles palabras a construir, indicando la longitud y la puntuación a obtener con cada una.

# 2 Descripción del proyecto

El objetivo del presente proyecto es construir un sistema de apoyo para el juego **Scrabble**. El sistema se implementará como una aplicación que recibe comandos textuales, agrupados en componentes con funcionalidades específicas. A continuación se describen los componentes individuales que conforman el proyecto.

## 2.1 Componente 1: configuración del juego.

**Objetivo:** Los algoritmos implementados en este componente servirán para gestionar la inicialización del sistema. Este componente se implementará con las siguientes funciones:

- **comando:** `inicializar` *`diccionario.txt`*
  **posibles salidas en pantalla:**
  (Diccionario ya inicializado) `El diccionario ya ha sido inicializado.`
  (Archivo no existe) `El archivo` *`diccionario.txt`* `no existe o no puede ser leído.`
  (Resultado exitoso) `El diccionario se ha inicializado correctamente.`
  **descripción:** Inicializa el sistema a partir del archivo `diccionario.txt`, que contiene un diccionario de palabras aceptadas en el idioma inglés (idioma original del juego). El comando debe almacenar las palabras del archivo de forma que sea fácil recuperarlas posteriormente. Las palabras deben ser verificadas para no almacenar aquellas que incluyen símbolos inválidos (como guiones, números y signos de puntuación).

- **comando:** `iniciar_inverso` *`diccionario.txt`*
  **posibles salidas en pantalla:**
  (Diccionario ya inicializado) `El diccionario inverso ya ha sido inicializado.`
  (Archivo no existe) `El archivo` *`diccionario.txt`* `no existe o no puede ser leído.`
  (Resultado exitoso) `El diccionario inverso se ha inicializado correctamente.`
  **descripción:** Inicializa el sistema a partir del archivo `diccionario.txt`, que contiene un diccionario de palabras aceptadas en el idioma inglés (idioma original del juego). A diferencia del comando `inicializar`, este comando almacena las palabras en sentido inverso (leídas de derecha a izquierda), teniendo en cuenta que sea fácil recuperarlas posteriormente. Las palabras también deben ser verificadas para no almacenar aquellas que incluyen símbolos inválidos (como guiones, números y signos de puntuación).

- **comando:** `puntaje` *`palabra`*
  **posibles salidas en pantalla:**
  (Palabra no existe) `La palabra no existe en el diccionario.`
  (Letras inválidas) `La palabra contiene símbolos inválidos.`
  (Resultado exitoso) `La palabra tiene un puntaje de` *`puntaje`*`.`
  **descripción:** El comando permite conocer la puntuación que puede obtenerse con una palabra dada, de acuerdo a la tabla de puntuación de cada letra presentada anteriormente. Sin embargo, el comando debe verificar que la palabra sea válida, es decir, que exista en el diccionario (tanto original como en sentido inverso), y que esté escrita con símbolos válidos.

- **comando:** `salir`
  **posibles salidas en pantalla:**
  (No tiene salida por pantalla)
  **descripción:** Termina la ejecución de la aplicación.

## 2.2 Componente 2: búsqueda de palabras.

**Objetivo:** Los algoritmos implementados en este componente servirán para buscar palabras que coincidan con un prefijo o sufijo dado. Este componente se implementará con las siguientes funciones:

- **comando:** `iniciar_arbol` *`diccionario.txt`*
  **posibles salidas en pantalla:**
  (Árbol ya inicializado) `El árbol del diccionario ya ha sido inicializado.`
  (Archivo no existe) `El archivo` *`diccionario.txt`* `no existe o no puede ser leído.`
  (Resultado exitoso) `El árbol del diccionario se ha inicializado correctamente.`
  **descripción:** Inicializa el sistema a partir del archivo `diccionario.txt`, que contiene un diccionario de palabras aceptadas en el idioma inglés (idioma original del juego). A diferencia del comando `inicializar`, este comando almacena las palabras en uno o más árboles de letras (como se considere conveniente). Las palabras deben ser verificadas para no almacenar aquellas que incluyen símbolos inválidos (como guiones, números y signos de puntuación).

- **comando:** `iniciar_arbol_inverso` *`diccionario.txt`*
  **posibles salidas en pantalla:**
  (Árbol ya inicializado) `El árbol del diccionario inverso ya ha sido inicializado.`
  (Archivo no existe) `El archivo` *`diccionario.txt`* `no existe o no puede ser leído.`
  (Resultado exitoso) `El árbol del diccionario inverso se ha inicializado correctamente.`
  **descripción:** Inicializa el sistema a partir del archivo `diccionario.txt`, que contiene un diccionario de palabras aceptadas en el idioma inglés (idioma original del juego). A diferencia de los comandos `iniciar_inverso` e `iniciar_arbol`, este comando almacena las palabras en uno o más árboles de letras, pero en sentido inverso (leídas de derecha a izquierda). Las palabras también deben ser verificadas para no almacenar aquellas que incluyen símbolos inválidos (como guiones, números y signos de puntuación).

- **comando:** `palabras_por_prefijo` *`prefijo`*
  **salida en pantalla:**
  (Prefijo inválido) `Prefijo` *`prefijo`* `no pudo encontrarse en el diccionario.`
  (Resultado exitoso) `Las palabras que inician con este prefijo son:`
  **descripción:** Dado un prefijo de pocas letras, el comando recorre el(los) árbol(es) de letras (construído(s) con el comando `iniciar_arbol`) para ubicar todas las palabras posibles a construir a partir de ese prefijo. A partir del recorrido, se presenta al usuario en pantalla todas las posibles palabras, la longitud de cada una y la puntuación que cada una puede obtener.

- **comando:** `palabras_por_sufijo` *`sufijo`*
  **salida en pantalla:**
  (Sufijo inválido) `Sufijo` *`sufijo`* `no pudo encontrarse en el diccionario.`
  (Resultado exitoso) `Las palabras que terminan con este sufijo son:`
  **descripción:** Dado un sufijo de pocas letras, el comando recorre el(los) árbol(es) de letras (construído(s) con el comando `iniciar_arbol_inverso`) para ubicar todas las palabras posibles a construir que terminan con ese sufijo. A partir del recorrido, se presenta al usuario en pantalla todas las posibles palabras, la longitud de cada una y la puntuación que cada una puede obtener.

## 2.3 Componente 3: combinaciones de letras.

**Objetivo:** Los algoritmos implementados en este componente servirán para generar diferentes posibilidades de combinaciones de letras, que correspondan a palabras válidas. Este componente se implementará con las siguientes funciones:

- **comando:** `grafo_de_palabras`
  **salida en pantalla:**
  (Resultado exitoso) `Grafo construído correctamente.`
  **descripción:** Con las palabras ya almacenadas en el diccionario (luego de ejecutar el comando `inicializar`), el comando construye un grafo de palabras, en donde cada palabra se conecta a las demás si y sólo sí difieren en un única letra (con las demás letras iguales y en las mismas posiciones).

- **comando:** `posibles_palabras` *`letras`*
  **salida en pantalla:**
  (Letras inválidas) `La cadena` *`letras`* `contiene símbolos inválidos.`
  (Resultado exitoso) `Las posibles palabras a construir con las letras` *`letras`* `son:`
  **descripción:** Dadas ciertas letras en una cadena de caracteres (sin importar su orden), el comando debe presentar en pantalla todas las posibles palabras válidas a construir, indicando la longitud de cada una y la puntuación que se puede obtener con cada una. En las letras de la cadena de caracteres, puede admitirse un único símbolo comodín (?), el cual representará una letra desconocida y permitirá generar mayores posibilidades de palabras a construir. Para este propósito, el comando debe hacer uso del grafo de palabras construído con el comando `grafo_de_palabras`.

### 2.4 Interacción con el sistema

La interfaz de la aplicación a construir debe ser una consola interactiva donde los comandos correspondientes a los componentes serán tecleados por el usuario, de la misma forma que se trabaja en la terminal o consola del sistema operativo. El indicador de línea de comando debe ser el caracter $. Se debe incluir el comando `ayuda` para indicar una lista de los comandos disponibles en el momento. Así mismo, para cada comando se debe incluir una ayuda de uso que indique la forma correcta de hacer el llamado, es decir, el comando `ayuda` *`comando`* debe existir.

Cada comando debe presentar en pantalla los mensajes de resultado (éxito o error) especificados antes, además de otros mensajes necesarios que permitan al usuario saber, por un lado, cuando terminó el comando su procesamiento, y por el otro lado, el resultado de ese procesamiento. Los comandos de los diferentes componentes deben ensamblarse en un único sistema (es decir, funcionan todos dentro de un mismo programa, no como programas independientes por componentes).

## 3 Evaluación

Las entregas se harán en la correspondiente actividad de BrightSpace, hasta la media noche del día anterior al indicado para la sustentación de la entrega. Se debe entregar un archivo comprimido (único formato aceptado: .zip) que contenga dentro de un mismo directorio (**sin estructura de carpetas interna**) los documentos (único formato aceptado: .pdf) y el código fuente (.h, .hxx, .cxx, .cpp). Si la entrega contiene archivos en cualquier otro formato, será descartada y no será evaluada, es decir, la nota definitiva de la entrega será de **0 (cero) sobre 5 (cinco)**.

### 3.1 Entrega 0: semana 3

La entrega inicial corresponderá únicamente a la interfaz de usuario necesaria para interactuar con el sistema. De esta forma, se verificará el indicador de línea del comando, y que el sistema realice la validación de los comandos permitidos y sus parámetros (Revisar en particular el numeral 2.4).

### 3.2 Entrega 1: semana 6

Componente 1 completo y funcional. Esta entrega tendrá una sustentación durante la correspondiente sesión de clase, y se compone de:

- (40%) Documento de diseño. El documento de diseño debe seguir las pautas de ingeniería que usted ya conoce: descripción de entradas, salidas y condiciones para el procedimiento principal y las operaciones auxiliares (comandos). Para la descripción de los TADs utilizados, debe seguirse la plantilla definida en clase. Además, se exigirán esquemáticos (diagramas, gráficos, dibujos) que describan el funcionamiento general de las operaciones (comandos) principales.
- (20%) Plan de pruebas. Adjuntar al documento de diseño un plan de pruebas, que siga las pautas vistas en clase, para el comando `puntaje`.
- (30%) Código fuente compilable en el compilador `gnu-g++` (versión 4.0.0, como mínimo). Este porcentaje de la entrega será un promedio de la evaluación de cada comando.

- (10%) Sustentación (individual) con participación de todos los miembros del grupo.

## 3.3 Entrega 2: semana 12

Componentes 1 y 2 completos y funcionales. Esta entrega tendrá una sustentación durante la correspondiente sesión de clase, y se compone de:

- (10%) Completar la funcionalidad que aún no haya sido desarrollada de la primera entrega. Se debe generar un acta de evaluación de la entrega anterior (incluirla al principio del documento de diseño) que detalle los comentarios textuales (literales) hechos a la entrega y la forma en la que se corrigieron, arreglaron o completaron para la segunda entrega.
- (30%) Documento de diseño. El documento de diseño debe seguir las pautas de ingeniería que usted ya conoce: descripción de entradas, salidas y condiciones para el procedimiento principal y las operaciones auxiliares (comandos). Para la descripción de los TADs utilizados, debe seguirse la plantilla definida en clase. Además, se exigirán esquemáticos (diagramas, gráficos, dibujos) que describan el funcionamiento general de las operaciones (comandos) principales.
- (20%) Plan de pruebas. Adjuntar al documento de diseño un plan de pruebas, que siga las pautas vistas en clase, para el comando `palabras_por_prefijo`.
- (30%) Código fuente compilable en el compilador `gnu-g++` (versión 4.0.0, como mínimo). Este porcentaje de la entrega será un promedio de la evaluación de cada comando.
- (10%) Sustentación (individual) con participación de todos los miembros del grupo.

## 3.4 Entrega 3: semana 18

Componentes 1, 2 y 3 completos y funcionales. Esta entrega tendrá una sustentación (del proyecto completo) entre las 8am y las 12m del último día de clase de la semana 18, y se compone de:

- (10%) Completar la funcionalidad que aún no haya sido desarrollada de la primera y segunda entregas. SSe debe generar un acta de evaluación de las entregas anteriores (incluirla al principio del documento de diseño) que detalle los comentarios textuales (literales) hechos a las entregas y la forma en la que se corrigieron, arreglaron o completaron para la tercera entrega.
- (30%) Documento de diseño. EEl documento de diseño debe seguir las pautas de ingeniería que usted ya conoce: descripción de entradas, salidas y condiciones para el procedimiento principal y las operaciones auxiliares (comandos). Para la descripción de los TADs utilizados, debe seguirse la plantilla definida en clase. Además, se exigirán esquemáticos (diagramas, gráficos, dibujos) que describan el funcionamiento general de las operaciones (comandos) principales.
- (20%) Plan de pruebas. Adjuntar al documento de diseño un plan de pruebas, que siga las pautas vistas en clase, para el comando `posibles_palabras`.
- (30%) Código fuente compilable en el compilador `gnu-g++` (versión 4.0.0, como mínimo). Este porcentaje de la entrega será un promedio de la evaluación de cada comando.
- (10%) Sustentación (individual) con participación de todos los miembros del grupo.