Descripción del Diagrama de Clases

Clase Board

Esta clase hace referencia al tablero de juego. El tablero está constituido de tres posibles tamaños, definidos por la variable size, estos tamaños son 7, 15 y 25 (el tablero siempre es cuadrado).

La mayoría de sus funciones son consultoras, pero destacamos algunas un tanto más complejas:

- hasLetter(int x, int y):: boolean

Simplemente indica si en la casilla indicada mediante los parámetros de entrada hay una letra o, en su defecto, es una casilla vacía.

- assignSpecialBoxes(int[] posicions, String type)

Se encarga de colocar las casillas con potenciador en el tablero. Estas casillas son *DoubleLetter*, *TripleLetter*, *DoubleWord*, *TripleWord* (más información en la clase *Box*). En función del tamaño size cada tablero sigue un patrón distinto de estas casillas, por lo que esta función sigue un tratamiento distinto para cada caso.

- placeLetter(int x, int y, String letter, int value)

Esta función se encarga de colocar una letra con un valor determinado en la casilla dada por los parámetros x e y.

- removeLetter(int x, int y)

Aquí simplemente se elimina la letra que haya en la casilla dada por los parámetros de entrada, en caso que las coordenadas indicadas sean distintas a las dimensiones del tablero el sistema lo notificará.

- printBoard()

printBoard() muestra por la consola de salida el tablero, diferenciando por colores las casillas especiales y mostrando las letras con sus respectivos valores.

- getLeftAnchorSquares():: Set<Box> y getRightAnchorSquares():: Set<Box>

Estas funciones retornan en Sets las “Casillas Ancla”, que son aquellas adyacentes a otras fichas mediante las cuales se pueden hacer las distintas jugadas.

- addAnchorSquareIfValid(Set<Box> anchorSquares, int x, int y)

Función que se usa en la anterior ya mencionada para añadir al Set una “Casilla Ancla” en caso que sea válida para ello (que esté dentro del tablero y que no tenga una letra en ella, además de ser adyacente a otra ficha).

- getLeftNeighbor(Box anchor):: Box / getRightNeighbor(Box anchor):: Box / getTopNeighbor(Box anchor):: Box / getDownNeighbor(Box anchor):: Box

Estas cuatro funciones buscan las cuatro casillas vecinas a cada “casilla ancla”.

Clase Box

Esta clase hace referencia a las casillas que componen un tablero, consta de cuatro subclases, una para cada tipo de casilla especial: *DoubleLetter* (la letra de esa casilla vale por 2), *TripleLetter* (la letra de esa casilla vale por 3), *DoubleWord* (las palabras que tengan una letra sobre esta casilla valen por dos), *TripleWord* (las palabras que tengan una letra sobre esta casilla valen por tres).

Lo primero que hay en esta clase es un enum Color, básicamente ofrece los distintos colores que se van a usar para cada tipo de casilla (normal -> verde, *DoubleLetter* -> Cyan, *TripleLetter* -> Azul, *DoubleWord* -> Magenta, *TripleWord* -> Rojo).

Los atributos de esta clase son las coordenadas x e y de la casilla, un String que sea el símbolo (no se usa un char ya que hay letras doble o triple, como la NY en catalán, la CH en castellano, la LL en ambos o la L·L en catalán) y un entero que sea el valor de esa letra.

En cuanto a las funciones se destaca toString():: String, que se encarga de devolver el String completo de una casilla, tratando los distintos casos de los tamaños de las letras o si la casilla es vacía, además de gestionar el color de la misma, se usa en printBoard() de la clase Board. El resto de funciones son consultoras, modificadoras o creadoras.Clase Box
Esta clase hace referencia a las casillas que componen un tablero, consta de cuatro subclases, una para cada tipo de casilla especial: *DoubleLetter* (la letra de esa casilla vale por 2), *TripleLetter* (la letra de esa casilla vale por 3), *DoubleWord* (las palabras que tengan una letra sobre esta casilla valen por dos), *TripleWord* (las palabras que tengan una letra sobre esta casilla valen por tres).

Lo primero que hay en esta clase es un enum Color, básicamente ofrece los distintos colores que se van a usar para cada tipo de casilla (normal -> verde, *DoubleLetter* -> Cyan, *TripleLetter* -> Azul, *DoubleWord* -> Magenta, *TripleWord* -> Rojo).

Los atributos de esta clase son las coordenadas x e y de la casilla, un String que sea el símbolo (no se usa un char ya que hay letras doble o triple, como la NY en catalán, la CH en castellano, la LL en ambos o la L·L en catalán) y un entero que sea el valor de esa letra.

En cuanto a las funciones se destaca toString():: String, que se encarga de devolver el String completo de una casilla, tratando los distintos casos de los tamaños de las letras o si la casilla es vacía, además de gestionar el color de la misma, se usa en printBoard() de la clase Board. El resto de funciones son consultoras, modificadoras o creadoras.

Clase Bag

Esta clase representa el montón de fichas de la partida, se compone de dos elementos, un Map<Letter, int> lettersMap, que representa cada ficha con el número total de ese tipo, y un entero que es el número total de fichas que hay en la bolsa.

Por lo que respecta a sus funciones, de nuevo nos encontramos con que la mayoría son modificadoras, creadoras o consultoras muy básicas, las más complejas son:

- extractLetter():: Letter

Encargada de extraer una letra al azar de entre toda la bolsa y devolverla, actualizando, por tanto, todo lo que respecta a la misma.
De la misma manera está también:

- addLetter(Letter letter)

Su función es añadir a la bolsa una letra, se usa en funciones posteriores.

- addSetOfLetters(List<Letter> letters)

Simplemente agrega a la bolsa la Lista que tenga por parámetro usando la función anterior.

- extractSetOfLetters(Integer quantity):: List<Letter>

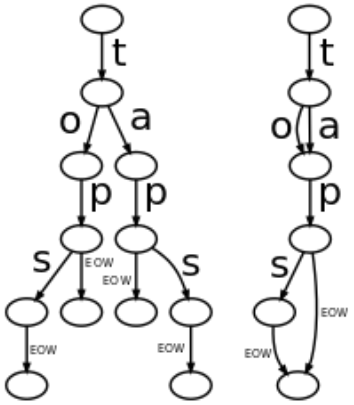
Usando la primera función descrita con anterioridad extrae “quantity” letras de la bolsa.

- changeRackLetters(List<Letter> letters):: List<Letter>

Esta función cambia las fichas del rack que el jugador decida por el mismo número de fichas de la bolsa. Si la bolsa no contiene las suficientes fichas para hacer este cambio no lo permitirá.

Clase Dawg

Esta clase representa el Directed Acyclic Word Graph, su creación, acceso, eliminación y gestión. El grafo DAWG se usa para almacenar las palabras del diccionario.



Por ello se compone de dos atributos, el idioma del DAWG, representado por un String, y un Nodo raíz del mismo (para más información sobre Nodo ver *Clase Nodo*).

Esta clase tiene varias funciones que merecen ser detalladas por escrito, dado que no cuenta con tantas consultoras como las clases vistas hasta ahora.

- getSpecialCharacter(String word, int index):: String

Esta función comprueba si el carácter en la posición “index” de la palabra “word” es especial o no, entendiendo por carácter especial aquellas fichas que tienen más de un carácter, mencionadas algunas de ellas anteriormente (CH, NY, L·L, RR, LL). La función retornará toda la ficha en caso de que sea de tipo especial y null en caso contrario.

- addWord(String word)

Esta función se apoya en una del mismo nombre pero recursiva para añadir una palabra al DAWG.

- addWordRec(Node node, String word, int index)

La función recursiva en cuestión, node es el nodo actual del DAWG, word la palabra a añadir e index el índice de la letra que se está añadiendo.

- existsWord(String word):: boolean y existsWordRec(Node node, String word, int index)

Sigue la metodología del caso anterior, pero aplicado a una consultora para comprobar si una palabra word existe en el DAWG.

- removeWord(String word) y removeWordRec(Node node, String word, int index)

Mismo caso pero esta vez para eliminar una palabra del DAWG.

- isPrefix(String prefix):: boolean y isPrefixRec(Node node, String prefix, int index):: boolean

Por último, isPrefix se encarga de mirar si dentro del DAWG hay palabras que tengan “prefix” como prefijo (comienzo de la palabra), para ello, usa la otra función recursiva de mismo nombre.

Clase Node

Esta clase representa un nodo del DAWG. Constituido por dos atributos, un Map<String, Node> Children, un mapa con el identificador del nodo actual y un puntero que apunta a los hijos de este, y un booleano isFinalNode, que indica si ese nodo es el último.

De sus funciones destacamos una función:

- addNextNode(String nextLetter, Node node)

Añade un nodo al mapa de los hijos del nodo que se pasa por parámetro.

Clase Dictionary

Esta clase representa los distintos diccionarios del en que se va a poder jugar, está formada por dos atributos, el String name y el Dawg dawg, su nombre y estructura de datos respectivamente.

De sus funciones hay poco que destacar, puesto que todas llaman a funciones del mismo nombre de la clase DAWG, ya que son para interactuar con sus datos, además que la gestión de los diccionarios se realiza desde la clase DictionaryController (detallada a continuación).

Clase DictionaryController

La clase DictionaryController gestiona todo lo relacionado con los diccionarios, como la creación, acceso y eliminación de los mismos. Consta de dos atributos, dictionaries, un Map<String, Dictionary> que almacena los diccionarios con su nombre como clave, y un DictionaryController c, su instancia única (singleton).

Sus funciones son bastnte comunes, tiene addDictionary(String dictionaryName, String language) y removeDictionary(String dictionaryName), que agregan y eliminan un diccionario respectivamente, ambas con control de errores si el diccionario sea repetido o no exista en cada caso.

- clearDictionaries()

Función que elimina todos los diccionarios del controlador.

- addWordsToDictionary(String dictionaryName, String wordsFile)

Esta función agrega al diccionario con nombre “dictionaryName” todas las palabras que haya en el archivo de texto “wordsFile”.

- addWordToDictionary(String dictionaryName, String word)

Agrega la palabra “word” al diccionario con nombre “dictionaryName”. Cuenta a su vez con control de errores en caso que el diccionario no exista o la palabra esté vacía.

- searchWordInDictioniary(String dictionaryName, String word):: boolean

Busca la palabra “word” en el diccionario “dictionaryName” y retorna el resultado en un booleano. También cuenta con control de errores en caso que el diccionario no exista o si la palabra es nula.

Clase DomainController

Esta clase es el núcleo encargado de coordinar al resto de controladores y clases del programa. Cuenta con varios atributos:

- profileController: ProfileController (instancia que maneja los perfiles de usuario)
- matchController: MP_Controller (instancia que maneja las partidas)
- ranking: Ranking (instancia que maneja las estadísticas de juego y ránkings)
- dictionaryController: DictionaryController (instancia que maneja el diccionario)
- c: DomainController (singleton)

Dentro del apartado de gestión de perfiles destacan las siguientes funciones:

- addProfile(String username, String password) y removeProfile(String username)

Funciones encargadas de agregar y eliminar un perfil.
getProfile(...):: Profile y profileExists(…):: boolean son dos consultoras, por lo que las omitimos.

Dentro del apartado de gestión de partidas destaca:

- continueMatch(String id)

Continua la partida iniciada. El resto son una creadora y una consultora.

Dentro del apartado de gestión de diccionarios destacan:

- addWordToDictionary(String dictionaryName, String word)

Agrega la palabra “word” al diccionario “dictionaryName”.

- removeWordFromDictionary(String dictionaryName, String word)

Elimina la palabra “word” al diccionario “dictionaryName”.

Finalmente y dentro del apartado de gestión de rankings está:

- updateRanking(String idWinner)

Actualiza el ránking añadiendo al ganador “idWinner”.

- playsMatch(String matchId, String word, int posStartX, int posStartY, int posEndX, int posEndY)

Realiza la jugada que se haya seleccionado.

- shuffleRack(String matchId)

Mezcla las fichas del rack.

- modifyRack(String matchId, String letters)

Modifica el rack del jugador.

Como inciso remarcar que, al ser esta clase un controlador, únicamente se dedica a llamar a las funciones del mismo nombre de las clases que coordina, por lo que si se quiere ver más en detalle las funciones de esta clase se recomienda ir directamente a las clases en cuestión.

Clase Player

Esta clase representa un jugador, compuesta por dos sublclases *Human* e *IA* (vistas más en detalle en sus respectivos apartados).
Player consta de cinco atributos:

- String id (id del jugador)
- int score (puntuación en una partida)
- String name (nombre del jugador, ya sea IA o Humano)
- Rack rack (rack del jugador)

Sus funciones son muy básicas, además que sus subclases también son muy básicas, ninguna tiene atributos nuevos (excepto por profile: Profile que tiene Human) ni funciones especiales más allá de las creadoras, consultoras y modificadoras muy básicas.
La única particularidad es el nombre de IA, que se constituye de "IAx", siendo x un número entero que no es mayor al número de IA de una partida, dando lugar a que en dos partidas distintas puede haber dos IA0, siendo diferenciadas entre sí gracias al id.

Clase Letter

La clase Letter es una ficha del juego, compuesta por los atributos symbol (String) y value (int).

De nuevo nos encontramos con que sus funciones son muy básicas, la más "compleja" es displayLetter(), que muestra por pantalla una ficha.

Clase Match

Esta clase es la partida del Scrabble, por lo que sus atributos son:
- id: String (identificador de la partida)
- turn: integer (marca de qué jugador es el turno)
- score: integer (puntuación total de todos los jugadores de la partida)
- finished: boolean (indica si la partida ha terminado o no)
- paused: boolean (indica si la partida está en pausa o no)
- size: integer (número de jugadores)
- players: Map<String, Player> (asocia el identificador de cada jugador con el jugador en cuestión)
- playerList: List<Player> (lista de todos los jugadores de la partida)
- dictionary: Dictionary (diccionario con el que se va a jugar la partida)
- bag: Bag (bolsa con la que se va a jugar la partida)
- board: Board (tablero con el que se va a jugar la partida)

Nuevamente nos encontramos con el mismo caso con las funciones, ya que Match organiza a todas las clases que tienen que ver en una partida, la mayoría de sus funciones son consultoras o modificadoras muy sencillas, aún así destacan decideWinner():: String, que devuelve al jugador con la mayor puntuación de entre todos los de la partida y setFinished():: String, que da por finalizada una partida, escoge al ganador y, si es humano, actualiza sus datos del ránking.

Clase MP_Controller

La clase MP_Controller (Match-Player Controller) se encarga de gestionar las partidas en el sistema. Centraliza todo lo relacionado con las mismas, incluyendo la coordinación entre las distintas clases implicadas en el devenir de una partida.

Tiene varios atributos:

- MP_Controller c (singleton)
- Map<String, Match> matches
- PlayableWord playableWord

Sus funciones destacadas relacionadas con el algoritmo y la explicación de las mismas se encuentran en el documento *Implementación del algoritmo de Búsqueda*.

El resto son:

- void createPlayersForMatch(Match match, Set<Profile> profiles, String language)
- void createDictionaryForMatch(Match match, Dictionary dictionary)
- void createBagForMatch(Match match, Map<Letter,Integer> letters, int bag_size)
- void createBoardForMatch(Match match, int size)

Estas cuatro funciones son las distintas creadoras de los objetos necesarios para poder jugar una partida.

Clase Profile

Representa el perfil de un jugador. Constituïda por un String username, un String password, un booleano isPublic, que indica el estado de visibilidad, un entero score que es la suma del score de todas las partidas, un entero wins, un entero gamesPlayed y un Map<String, Int> dictionaryUsage, que indica cuantas partidas se han jugado con cada diccionario.

De sus funciones destacan:

- changePassword(String old_pw, new_pw)

Cambia la contraseña del perfil, comprobando antes de cambiarla si el usuario conoce la antigua contraseña para evitar accesos de otros usuarios.

- authenticate(String password):: boolean

Comprueba que el usuario que está accediendo al perfil sea el correcto.

Clase ProfileController

Clase singleton encargada de la gestión de los distintos perfiles dentro del juego. La constituyen un Map<String, Profile> que tiene el String que identifica a cada perfil con el perfil en cuestión y el ProfileController c.

Tiene funciones muy básicas, ya que solamente gestiona la clase Perfil, que es quien tiene todas las funciones que modifican los atributos de la clase.

Clase Rack

Representa el rack que tiene un jugador en una partida, el rack se constituye de siete fichas (letter) con las que el jugador forma las distintas palabras.
Un rack está compuesto por un Set<Letter> letters, que son las distintas letras que tiene el rack, un entero NUM_LETTERS = 7 y un Bag bag, que es el montón de fichas de la partida (para ver la clase en más detalles ir a *Clase Bag*, ubicada al inicio de este documento).

Entre sus funciones destacan:

- shuffle()

Simplemente mezcla las letras del Rack.

- renew()

Se encarga de intercambiar las siete letras del rack por siete de la bolsa.

- clone():: Rack

Crea una copia del rack del jugador para poder planear las distintas posibles jugadas sin afectar al rack.

Clase Ranking

El ránking es la clasificación de los perfiles en un posicionamiento global. Sus atributos son: Map<String, Profile> profiles, que organiza a todos los perfiles por su id y un Map<int, Set<String>>, que asocia los números de victoria que hay con los usuarios con dicho número.

De sus funciones destacan:

- updateRanking(Profile profile)

Se llama al terminar cada partida, actualiza el Ranking con el perfil del ganador.

- computeWR(int wins, int gamesPlayed):: double

Calcula el Win Rate en función de las partidas jugadas y el número de victorias.

- computePD(Map<String, int> dictionaryUsage)

Calcula el diccionario favorito, es decir, cual es el diccionario con el que más ha jugado el jugador.