Rànquing (Ferran Blancart Reyes, ferran.blanchart@estudiantat.upc.edu)

- Aquesta classe consta d'un únic atribut privat que guarda els rànquings que hi han per cada mode de joc. Pots eliminar a un usuari de tots els rànquings on aparegui, actualitzar-los quan sigui necessari i visualitzar-los, tenint en compte que només apareixen les 10 primeres posicions de cada tipus.
- S'utilitza un HashMap amb clau "String" i de valor un altre map amb clau "String" i valor "Float. D'aquesta manera el map principal mapeja el tipus de rànquing al que s'accedeix i es poden gestionar molts usuaris i molts tipus de rànquing de forma escalable. Aquest tipus d'estructura ens permet accedir ràpidament a un rànquing amb un cost mitjà molt baix, O(1). També s'utilitza una llista per tal d'ordenar els usuaris de forma descendent segons la seva puntuació amb l'utilització d'un sort, que dins d'un for limitat a "Math.min(10, llista.size())" permet extreure ràpidament el top 10.

Estadístiques (Ferran Blancart Reyes, ferran.blanchart@estudiantat.upc.edu)

- Aquesta classe serveix per portar un registre de totes les estadístiques individuals per cada usuari. Entre aquestes es troben: nombre de partides guanyades, nombre de partides perdudes, nombre total de punts, la millor puntuació registrada per aquell usuari, la seva puntuació mitjana i 2 sets que guarden els ids de totes les partides acabades per l'usuari i les que té pausades respectivament.
- S'utilitzen: "Integer" als atributs que corresponen a un comptador ja que permeten calcular ràpidament un total, "Float" als que porten un registre de quelcom que requereixi algun tipus de càlcul en moments concrets i amb precisió decimal i "HashSet" per permetre una col·lecció única de partides i poder accedir a elles ràpidament.

Usuari (Ferran Blancart Reyes, ferran.blanchart@estudiantat.upc.edu)

- Aquesta classe conté les dades que necessita el sistema per registrar a un usuari (nom i contrasenya) i també té accés directe a les estadístiques del mateix usuari.
- Conté aquest accés directe a les seves estadístiques ja que així delega la lògica a una classe separada, tal i com indica l'arquitectura de 3 capes. També hi ha un bool per detectar si l'usuari ha estat eliminat ja que així evitem eliminar completament l'objecte i mantenim la traçabilitat de l'usuari (per exemple, per consultar partides antigues o estadístiques).

Casella (Alexander De Jong, alexander.de.jong@estudiantat.upc.edu )

Representa una casella del tauler amb propietats com la fila, la columna, el multiplicador i la fitxa que conté. Inclou mètodes per:

- Obtenir informació de la casella.
- Col·locar una fitxa.
- Comprovar si té fitxa.
- Treure la fitxa.

Fitxa (Alexander De Jong, alexander.de.jong@estudiantat.upc.edu )
Representa una fitxa del joc amb una lletra i un valor de punts. Inclou mètodes per:

- Obtenir la lletra.
- Obtenir i establir el valor de punts.

Tauler (Alexander De Jong, alexander.de.jong@estudiantat.upc.edu )
Representa el tauler complet, gestionant una matriu de caselles. Inclou:

- Inicialització del tauler amb els multiplicadors adequats.
- Mètodes per obtenir caselles.
- Mètode per col·locar paraules (horitzontal o vertical) i calcular puntuacio de cada paraula colocada.
- Representació visual del tauler.

Anàlisi i justificació (Alexander De Jong, alexander.de.jong@estudiantat.upc.edu )

Estructures de Dades Principals

1. Matriu Bidimensional (Casella[][])
   Implementació: La matriu bidimensional és l'estructura central per representar el tauler de Scrabble. Justificació:
   - Accés directe per coordenades: Permet accedir a qualsevol casella en temps constant O(1) mitjançant les coordenades [fila, columna].
   - Representació espacial intuïtiva: Reflecteix de manera natural l'estructura física del tauler, facilitant la visualització mental.
   - Eficàcia en recorreguts direccionales: Facilita recórrer files, columnes o diagonals amb simples incrementos d'índexs.
   - Memòria previsible: En tenir una mida fixa (generalment 15×15), consumeix una quantitat previsible de memòria.

2. Llistes Dinàmiques (ArrayList<Pair<Int,Int>> i List<Fitxa>)
   Implementació: S'utilitzen per emmagatzemar les fitxes a col·locar i les posicions on es col·loquen. Justificació:
   - Mida variable: Ideal per emmagatzemar un nombre variable de fitxes o posicions.

    - Accés indexat ràpid: Permet accedir a elements específics en temps constant O(1).
    - Operacions d'addició eficients: Afegir elements té una complexitat amortitzada O(1).
    - Iteració seqüencial: Facilita recórrer tots els elements de manera ordenada, ideal per calcular puntuacions o verificar posicions.

3. Lista de Pair per coordenades
   Implementació: Cada posició es representa com un Pair de dos enters [fila, columna].

Partida (Pau Serrano Sanz , pau.serrano.sanz@estudiantat.upc.edu)

Estructures de dades:

- List<Jugador> jugadors: Llista ordenada dels jugadors que participen en la partida. Es fa servir perquè el torn avanci de manera cíclica i controlada per un índex.
- List<Fitxa> bossa: Representa la bossa de fitxes com una llista. Permet afegir fitxes amb facilitat i, mitjançant Iterator, eliminar-les eficaçment quan es reparteixen.
- Map<String, Pair<Integer, Integer>> alfabet: S'obté del diccionari per generar les fitxes inicials de la bossa. Cada lletra es mapeja amb una parella (quantitat, puntuació).
- int tornActual: Index que marca quin jugador té el torn. Permet fàcilment determinar qui juga i avançar el torn de manera cíclica amb % (mòdul).
- String estat: Controla l'estat actual de la partida ("enCurs", "pausada", "finalitzada").

Algoritmes i funcionalitats destacades:

- inicialitzarBossa(): Construeix la bossa de fitxes a partir de l'alfabet proporcionat pel diccionari. Utilitza dos bucles (un pel map i un per la quantitat de cada lletra). També fa un shuffle per desordenar la bossa, simulant una partida real.
- repartirFitxes(): Cerca el jugador pel seu nom i reparteix fins a numFitxes fitxes de la bossa utilitzant un Iterator per eliminar fitxes mentre s'afegeixen al jugador. Això garanteix que no es produeixi un error de concurrència.
- avançarTorn(): Incrementa el torn actual de manera cíclica gràcies a l'operació amb mòdul (%).
- Control d'estats (pausar, continuar, finalitzar): Permet controlar la lògica de flux de la partida.

Justificació:

- List per jugadors i bossa: Permet accés per índex (jugadors) i control ordenat de les fitxes (bossa), així com modificació dinàmica del contingut.
- Map per l'alfabet: Ideal per accedir ràpidament a les característiques d'una lletra (complexitat O(1)).
- Iterator per repartir fitxes: Evita errors al modificar una llista mentre s'itera, fent el procés segur i eficient.

- Modularitat i claredat: Cada tasca essencial en el codi té la seva pròpia funció (repartir, avançar torn, etc.). Això facilita la llegibilitat i mantenibilitat del codi.

Diccionari (Pau Serrano Sanz , pau.serrano.sanz@estudiantat.upc.edu)

Estructures de dades:

- Set<String> paraules: Conjunt ordenat (TreeSet) que emmagatzema les paraules del diccionari. Garanteix l'ordre lexicogràfic i evita duplicats.
- Map<String, Pair<Integer, Integer>> alfabet: Mapa on cada lletra s'associa amb una parella (quantitat, puntuació). Representa la bossa de fitxes del joc.
- Node arrel: Arrel de l'estructura DAWG (Directed Acyclic Word Graph), que representa el conjunt de paraules de manera eficient.
- Map<Node, Node> registre: Registre per detectar subgraf equivalents i reduir el nombre de nodes duplicats en la construcció incremental del DAWG.
- String ultimaParaulaAfegida: Guarda l'última paraula afegida per garantir que les següents paraules s'afegeixen en ordre lexicogràfic.

Algoritmes i funcionalitats destacades:

- construirDAWG(): Construeix el DAWG inicial recorrent totes les paraules ordenades i afegint-les incrementalment.
- afegirParaulaIncremental(String paraula): Algorisme incremental de construcció del DAWG basat en prefixos comuns amb l'última paraula afegida. Optimitza l'estructura evitant recomputacions.
- reemplaçarORegistrar(Node node): Permet compartir subgraf equivalents entre paraules diferents. Utilitza el Map<Node, Node> per detectar repeticions i mantenir el graf mínim.
- eliminarParaula(String paraula): Elimina una paraula i reconstrueix tot el DAWG per mantenir la seva minimalitat.
- recorre(Node node, String prefix, List<String> resultat): Recorregut en profunditat per recuperar les paraules guardades al DAWG.
- validarParaula(String paraula): Verifica si una paraula es troba al diccionari recorrent les transicions del DAWG.

Justificació:

- TreeSet per a paraules: Manté l'ordre lexicogràfic i evita duplicats, requisit fonamental per a la construcció incremental del DAWG.
- DAWG (Directed Acyclic Word Graph): Representa eficientment moltes paraules compartint prefixos i sufixos comuns. És més compacte que un trie o un arbre prefixat.
- Map<Node, Node> registre: Clau per garantir que els subgrafs repetits es comparteixin, minimitzant la mida del DAWG.
- Robustesa i coherència: La comprovació de l'ordre lexicogràfic abans d'
- afegir una nova paraula assegura la validesa de l'algorisme incremental.

Jugador (Pau Serrano Sanz , pau.serrano.sanz@estudiantat.upc.edu)

Estructures de dades:

- String nom: Guarda el nom identificador del jugador. És immutable un cop creat.
- int puntuacio: Enter que representa la puntuació total acumulada del jugador durant la partida.
- List<Fitxa> fitxes: Llista dinàmica que conté les fitxes actuals del jugador. Permet accés, modificació i reordenació amb facilitat.

Algoritmes i funcionalitats destacades:

- afegirFitxa(Fitxa): Afegeix una nova fitxa a la mà del jugador. Utilitza la naturalesa dinàmica de List per fer-ho eficientment.
- colocarFitxa(Fitxa): Cerca i elimina una fitxa específica. Comprova si el jugador la té abans d'eliminar-la per seguretat lògica.
- actualitzarPuntuacio(int): Incrementa la puntuació del jugador amb un valor donat.
- reordenarFitxes(): Reordena aleatòriament les fitxes del jugador utilitzant Collections.shuffle() per simular l'acció física de reorganitzar fitxes.
- canviarFitxes(List<Fitxa>): Substitueix totes les fitxes actuals del jugador per un nou conjunt. És útil en accions com "canviar fitxes" del joc.

Justificació:

- List per fitxes: Permet una gestió flexible de les fitxes del jugador (afegir, eliminar, desordenar), ideal per simular la mecànica de mà de fitxes d'Scrabble.
- Modularitat: Cada acció típica d'un jugador té la seva pròpia funció (afegirFitxa, colocarFitxa, etc.), millorant la llegibilitat i mantenibilitat.