DESCRIPCIÓ I JUSTIFICACIÓ DE LES ESTRUCTURES DE DADES I ALGORISMES IMPLEMENTATS

de Jong Roca, Alexander - alexander.de.jong@estudiantat.upc.edu
Serrano Sanz, Pau - pau.serrano.sanz@estudiantat.upc.edu
Blancart Reyes, Ferran - ferran.blanchart@estudiantat.upc.edu
Gil Moreno, Marc - marc.gil.moreno@estudiantat.upc.edu

Grup 11 - Subgrup 1
Professora - Alicia Ageno Pulido
Universitat Politècnica de Catalunya
Facultat d'Informàtica de Barcelona
2024 - 2025 Q2

INDEX:

DESCRIPCIÓ DE LES CLASSES DE LA CAPA DE DOMINI

Classe ControladorDomini

El Controlador de Domini és el nucli de la lògica del joc d'Scrabble. Gestiona totes les operacions relacionades amb partides, usuaris, diccionaris, rànquings i la interacció amb les capes de persistència i presentació. A continuació es detallen les seves estructures de dades i funcions més importants.

Estructures de dades:

Atribut	**Descripció**
Partida partida	Partida actual en curs.
Diccionari diccionari	Diccionari actual carregat per jugar.
ControladorMaquina maquina	Controlador que decideix les jugades dels bots.
Usuari usuariActiu	Usuari que ha iniciat sessió.
Usuari usuari2	Segon usuari en cas de partida 1v1 entre humans.
Ranquing ranquing	Objecte que manté els rànquings globals i per diccionari.
ControladorPersistencia cp	Classe que gestiona la lectura/escriptura a fitxers.
CtrlUsuari ctrlUsuari .	Controlador dedicat a l'autenticació i gestió d'usuaris
int passarTorn	Control de torns passats consecutius (per finalitzar la partida).
int numFitxes	Nombre de fitxes per jugador (segons dificultat).
Dificultat dificultat	Dificultat actual de la partida.
static ControladorDomini instance	Instància singleton del controlador (si s'utilitza)

Funcions i responsabilitats principals**:**

Autenticació i gestió d'usuaris:

- login, loginSegonUsuari: Inicien sessió d'un o dos jugadors.
- registrarUsuari, registraSegonUsuari: Registre d'usuaris.

- canviarNomUsuari, canviarContrasenya: Modificació de credencials.
- eliminarUsuariComplet: Elimina un usuari i tota la seva informació associada.

Diccionaris:

- importaDiccionari(idioma): Carrega un diccionari i el seu alfabet des de fitxer.
- guardarDiccionari(...): Desa el diccionari modificat.
- eliminarDiccionari(nom): Elimina el diccionari i els rànquings associats.

Gestió de la partida:

- definirPartida(...): Crea una nova partida amb paràmetres personalitzats.
- jugarTorn(): Executa un torn, inclòs el de la màquina si escau.
- confirmarJugadaHumana(List<PosicioFitxa>): Aplica i valida la jugada d'un humà.
- passarTorn(int i): Controla la lògica de passar torns.
- intercanviarFitxes(List<Character>): Permet canviar fitxes de l'atril.

Control del temps i estat: pausarPartida(), continuarPartida(), finalitzarPartida():

Persistència de partides: guardarPartida(...), carregarPartida(...), eliminarPartida(...):

Rànquings i estadístiques:

- acabarPartida(...): Finalitza la partida, actualitza rànquings i estadístiques.
- guardarHistorialPartida(...): Desa un resum de la partida.
- obtenirInfoFinal(): Retorna si la partida ha acabat i la puntuació final de cada jugador.
- obtenirNomMesPunts(): Retorna el nom del jugador amb més punts o "Empat".

Justificació de disseny i arquitectura:

Aquesta classe encapsula tota la lògica d'alt nivell, mantenint la separació de responsabilitats entre vista, domini i persistència. A més a més, col·labora amb altres controladors de la capa de domini:

- CtrlUsuari: Gestiona la lògica relacionada amb les credencials.
- ControladorPersistencia: Controla l'accés a dades (fitxers).
- ControladorPresentació: Controla la interacció amb l'usuari.
- ControladorMaquina: Controla el comportament del jugador màquina.

Dades encapsulades: Totes les operacions importants passen per aquest controlador, la qual cosa permet controlar l'estat del joc d'una forma segura.

Classe CtrlUsuari

Aquesta classe de la capa de domini s'encarrega de tota la lògica relacionada amb la gestió d'usuaris en el sistema d'Scrabble. Permet iniciar sessió, registrar-se, canviar el nom o la contrasenya, i gestionar tant l'usuari actiu com un possible segon jugador per a partides multijugador. Internament, es comunica amb el ControladorPersistencia per accedir a la informació dels usuaris desada al sistema de fitxers, i també amb el ControladorDomini per actualitzar l'estat del joc (com ara el ranquing o l'usuari actiu global). La classe valida les credencials, gestiona errors d'autenticació amb excepcions i encapsula tot el que cal per controlar correctament la sessió dels jugadors dins del joc.

Classe Rànquing

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

La classe Ranquing representa el sistema de gestió de classificacions dels usuaris en diferents tipus de rànquings dins del joc de Scrabble. Aquesta classe emmagatzema, actualitza i mostra les puntuacions dels jugadors de manera ordenada, permetent tenir accés eficient a les dades tant per consultar com per modificar. Està estructurada per oferir suport a diversos tipus de rànquings i permet operacions com afegir punts, substituir puntuacions, canviar noms d'usuari, eliminar usuaris o rànquings específics.

Estructures de dades:

- Map<String, TreeSet<EntradaRanquing>> classificacio: Map que associa cada tipus de rànquing amb un conjunt ordenat de les entrades. El TreeSet manté les entrades ordenades descendentment per punts i per identificador d'usuari (en cas d'empat). És la base per mostrar els rànquings.

- Map<String, Map<String, EntradaRanquing>> accesDirecte: Map de maps que permet accedir ràpidament a una entrada concreta dins d'un tipus de rànquing, donat l'identificador d'usuari. Aquesta estructura complementa classificacio per optimitzar les modificacions.

- EntradaRanquing entrada (classe interna): Objecte que representa una entrada al rànquing, amb dos camps: String idUsuari i Float punts. Implementa Comparable per mantenir l'ordre en els TreeSet, i redefineix equals i hashCode per basar-se només en l'identificador de l'usuari.

- List<String> resultat (retornada per mostraRanquing): Llista de cadenes que representen les 10 primeres posicions d'un rànquing. Es genera dinàmicament cada cop que es consulta el rànquing.

Algoritmes i funcionalitats destacades:

- actualitzarRanquing(String id, String tipus, Float punts): Actualitza la puntuació d'un usuari en un rànquing concret. Si el tipus és "percentatge" o "record", es sobreescriu el valor anterior ja que no es tracta d'un rànquing acumulatiu. Si és un rànquing acumulatiu (com "total"), la puntuació s'afegeix. Sempre es fa eliminant i tornant a

inserir l'entrada al TreeSet per mantenir l'ordre.Optimitza tant lectura com escriptura gràcies a accesDirecte.

- mostraRanquing(String tipus): Retorna una llista amb els 10 primers classificats d'un rànquing concret. Recorre el TreeSet del tipus indicat i construeix una llista de cadenes amb el format "posició - usuari (punts punts)".

- getPuntsUsuari(String id, String tipus): Consulta directa de punts per un usuari en un rànquing concret. Utilitza accesDirecte per accedir eficientment al valor i retorna 0 si no existeix cap entrada.

- eliminarUsuari(String id):Elimina completament un usuari de tots els rànquings. Per cada tipus de rànquing existent, busca i elimina l'entrada corresponent tant del TreeSet com del Map d'accés directe.

- canviarNomUsuari(String nomAntic, String nomNou):Canvia l'identificador d'usuari a tots els rànquings. Per cada tipus, elimina l'entrada amb el nom antic, crea una nova entrada amb el mateix valor de punts però amb el nou nom, i actualitza tant el TreeSet com el Map.

- eliminarRanquingDeDiccionari(String nomDiccionari):Elimina el rànquing associat a un diccionari concret i actualitza el rànquing "total" restant els punts de cada jugador afectat. Es gestiona amb cura la consistència del TreeSet i s'eliminen entrades si la puntuació resultant és ≤ 0.

Justificació de les estructures de dades i algoritmes:

- Map<String, TreeSet<EntradaRanquing>> classificacio: Permet gestionar múltiples rànquings alhora, cada un amb ordenació automàtica per puntuació i desempat alfabètic. L'ús de TreeSet assegura que la llista sempre estigui ordenada, facilitant consultes com mostrar el top 10 sense ordenar explícitament.

- Map<String, Map<String, EntradaRanquing>> accesDirecte: Millora el rendiment de cerca i modificació de puntuacions. Sense aquesta estructura auxiliar, caldria recórrer el TreeSet per localitzar cada entrada.

- actualitzarRanquing(String id, String tipus, Float punts): Gestiona de manera eficient tant rànquings acumulatius com de substitució. Garanteix la coherència del TreeSet eliminant i tornant a inserir l'entrada després de cada modificació de punts, mantenint així l'ordre.

- mostraRanquing(String tipus):Permet consultar els 10 primers classificats sense processaments addicionals. Es recolza en l'ordenació automàtica del TreeSet per generar resultats immediats i consistents.

- getPuntsUsuari(String id, String tipus):Operació de lectura directa que evita càlculs costosos gràcies a accesDirecte.

- eliminarUsuari(String id):Permet mantenir la consistència del sistema en cas que un usuari es doni de baixa. La doble estructura Map + TreeSet permet esborrar ràpidament totes les seves entrades sense recorrer tota la classificació.

- canviarNomUsuari(String nomAntic, String nomNou): Garanteix que un canvi d'identificador no alteri la integritat de les puntuacions.

- eliminarRanquingDeDiccionari(String nomDiccionari): Recalcula de manera precisa el rànquing "total" en eliminar un rànquing associat a un diccionari. Evita la pèrdua de punts acumulats erròniament i garanteix que només es mantinguin jugadors amb puntuació positiva.

Classe Estadístiques

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

La classe Estadistiques encapsula tota la informació rellevant sobre el rendiment d'un usuari en el joc. Emmagatzema el total de punts acumulats, la millor puntuació aconseguida, la mitjana de punts per partida, l'historial d'identificadors de partides finalitzades i un mapa amb les estadístiques de puntuació obtingudes per cada diccionari utilitzat. Aquesta classe actua com a registre personal de l'usuari. Ofereix un únic mètode de modificació centralitzat per actualitzar totes les estadístiques de forma coherent i eficient després de cada partida.

Estructures de dades:

- Float totaPunts: Total acumulat de punts obtinguts per l'usuari. És una dada agregada útil per rànquings generals i evolució personal.

- Float millorPuntuacio: Registra la millor puntuació que l'usuari ha aconseguit en una partida. Permet destacar partides excepcionals i mostrar el màxim històric.

- Float puntuacioMitjana: Representa la mitjana de punts per partida. Aquesta mitjana es recalcula progressivament per oferir un resum global del rendiment.

- Set<Integer> historialPartides: Guarda els identificadors únics de les partides finalitzades. Aquesta col·lecció evita duplicats i permet saber quines partides ja han estat comptabilitzades.

- Map<String, Float> estadistiquesPerDiccionari: Mapa que acumula punts obtinguts per cada diccionari (idioma o temàtica).

Algoritmes i funcionalitats destacades:

- Estadistiques(): Constructora per defecte que inicialitza totes les estadístiques amb valors buits o zero. Assegura que cada usuari comença amb una estructura consistent i preparada per registrar dades.

- actualitzarEstadistiques(Integer idPartida, Float puntuacio, String diccionari): Actualitza tots els valors estadístics en funció dels resultats d'una nova partida. Afegeix l'id de la partida a l'historial per evitar repeticions. Suma la nova puntuació al total. Actualitza la millor puntuació si la nova és superior. Recalcula la mitjana (de forma simplificada) per reflectir la nova dada. Actualitza el total de punts obtinguts amb el diccionari utilitzat.

Justificació de les estructures de dades i algoritmes:

- Float totaPunts: Emmagatzema el total de punts acumulats de l'usuari. S'utilitza per generar rànquings globals i per mostrar un resum ràpid del progrés.
- Float millorPuntuacio: Permet identificar la millor partida feta per l'usuari, útil per fer reconeixements i motivar l'usuari a millorar.
- Float puntuacioMitjana: Recalcular la mitjana directament és eficient i evita guardar totes les puntuacions. És útil per comparar rendiment entre usuaris.
- Set<Integer> historialPartides: Utilitza un Set per garantir la unicitat dels ids. Aquesta elecció impedeix duplicar estadístiques accidentalment i facilita comprovacions ràpides.
- Map<String, Float> estadistiquesPerDiccionari: Permet personalitzar i analitzar el rendiment de l'usuari segons el diccionari utilitzat. És útil per estadístiques segmentades.
- actualitzarEstadistiques(Integer idPartida, Float puntuacio, String diccionari): Centralitza l'actualització de totes les mètriques. El seu disseny permet una extensió fàcil si en el futur es volgués afegir més estadístiques com partides guanyades, etc.

Classe Usuari

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

La classe Usuari representa un jugador del sistema, ja sigui humà o màquina, gestionant la seva informació bàsica (nom i contrasenya), el seu estat (actiu o eliminat), i les seves estadístiques de joc. Permet crear, modificar i eliminar usuaris de forma controlada, així com persistir i recuperar les seves estadístiques a través de fitxers serialitzats. Aquesta classe encapsula la lògica d'identitat i control d'accés de l'usuari, i distingeix clarament entre usuaris reals i la màquina.

Estructures de dades:

- String nomUsuari: Nom de l'usuari dins del sistema. Es fa servir per identificar-lo i associar-lo amb les seves estadístiques.
- String contrasenya: Guarda la contrasenya per validar l'accés.
- Estadistiques estadistiques: Objecte que encapsula les dades de rendiment de l'usuari. Es serialitza i deserialitza per mantenir persistència entre sessions.
- boolean eliminat: Indica si l'usuari ha estat eliminat. S'utilitza per invalidar accés, canvis o accions sobre l'usuari un cop es dóna de baixa.
- boolean esMaquina: Marca si l'usuari representa el jugador màquina.

Algoritmes i funcionalitats destacades:

- canviarNom(String nouNom): Assigna un nou nom a l'usuari si no ha estat eliminat.

- canviarContrasenya(String novaContra): Substitueix la contrasenya si l'usuari no ha estat eliminat.
- eliminarUsuari(): Marca l'usuari com eliminat i esborra les seves dades internes (nom, contrasenya i estadístiques). No elimina físicament el fitxer, però desactiva funcionalitats.
- guardarEstadistiques(): Desa l'objecte Estadistiques serialitzat en un fitxer identificat pel nom de l'usuari. Permet persistència entre sessions.
- carregarEstadistiques(): Recupera les estadístiques d'un fitxer si existeix. Si no, crea estadístiques noves. Assegura que cada usuari tingui dades pròpies.

Justificació de les estructures de dades i algoritmes:

- String nomUsuari: S'utilitza com a identificador únic de l'usuari per accedir a fitxers i mostrar informació personalitzada. La seva modificació està controlada.
- String contrasenya: Tractada com text per simplificar la comparació i emmagatzematge.
- Estadistiques estadistiques: Aquesta composició permet separar clarament la lògica de l'usuari de les seves mètriques de joc. A més, facilita la persistència modular.
- boolean eliminat: Permet implementar una eliminació lògica sense esborrar fitxers..
- boolean esMaquina: Distingir si un usuari és humà o la màquina és essencial per activar l'algorisme de màquina o evitar tractaments com guardar contrasenya.
- canviarNom(String nouNom): Evita canvis innecessaris en usuaris inactius. Manté la consistència si es vincula amb rànquings, fitxers o sessions.
- eliminarUsuari(): Permet aplicar una eliminació no destructiva.
- guardarEstadistiques() / carregarEstadistiques(): Són mètodes essencials per persistir dades personals.
- esEliminat() i esMaquina(): Encapsulen decisions importants sobre l'estat i el tipus d'usuari, millorant llegibilitat i mantenibilitat.

Classe Partida

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

Aquesta classe representa una sessió de joc d'Scrabble, ja sigui entre dos jugadors humans o entre un jugador humà i la màquina. Gestiona el tauler, les fitxes, els jugadors participants, el diccionari seleccionat, l'estat actual (en curs, pausada o finalitzada), la dificultat (fàcil, normal, difícil), el mode de joc (contra màquina/multijugador, contrarellotge/sense temps), el temporitzador (en cas que sigui contrarrellotge), les jugades realitzades, la bossa de fitxes i el torn.

Per manca de temps i excés de complexitat, s'ha decidit no implementar algunes de les funcionalitats extres especificades en un primer moment, com l'opció de jugar amb més d'un diccionari en una partida o poder jugar amb més de dos jugadors. No obstant això, s'han implementat altres funcionalitats adicionals, com els modes de joc, la dificultat, l'opció de tenir un temporitzador i un registre amb les jugades realitzades per cada jugador.

A continuació es mostra una descripció de les principals estructures de dades, funcions i algorismes de la classe Partida.

Estructures de dades:

- int idPartida: Identificador de la partida.
- 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.
- Tauler tauler: representa l'estat actual del tauler i encapsula tota la lògica relacionada amb la disposició de les caselles i fitxes.
- 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").
- Boolean contrarrellotge: Indica si la partida es juga en mode contrarrellotge.
- Temporitzador temporitzador: Temporitzador de la partida. Si la partida és contrarrellotge, s'inicialitza un temporitzador amb un temps determinat. Quan el temps s'esgota la partida es dóna per finalitzada. Veieu la descripció de la classe util Temporitzador per a informació més detallada.
- Boolean primeraJugada: Indica si s'ha jugat el primer torn de la partida.
- Dificultat dificultat: Dificultat de la partida (fàcil, normal, difícil)
- List<Jugada> jugadesRealitzades: Llista que registra totes les jugades realitzades en la partida.

Algorismes 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 (%).

- aplicarMovimentHumà(...): Valida, processa i aplica una jugada feta pel jugador actual. Fa comprovacions com:

    - Orientació i contigüitat de la jugada.

    - Connexió amb fitxes existents o presència en el centre si és la primera jugada.

    - Existència de fitxes en el faristol.

    - Validació al diccionari (incloent paraules creuades i comodins).

    - Càlcul i actualització de la puntuació.

    - Confirmació i actualització del tauler.

- Funcions pel control d'estats (pausar, continuar, finalitzar): Permet controlar la lògica de flux de la partida.

- boolean validarPalabraConComodin(String paraula): Valida si una paraula amb comodí és al diccionari comprovant totes les possibles substitucions del comodí per lletres del diccionari. Retorna cert si alguna substitució és vàlida dins del conjunt de paraules del diccionari, fals altrament.

Justificació de les estructures dades i algorismes:

- List<Jugada> jugadesRealitzades: s'utilitza per emmagatzemar l'historial de jugades. Això permet implementar fàcilment funcionalitats com la visualització del resum de la partida o la seva persistència.

- Tauler tauler: representa l'estat actual del tauler i encapsula tota la lògica relacionada amb la disposició de les caselles i fitxes. Es deleguen les validacions estructurals de les jugades al tauler per mantenir la responsabilitat ben separada.

- Diccionari diccionari: encapsula les operacions de validació de paraules i permet canviar el conjunt de paraules vàlides segons l'idioma seleccionat.

- Temporitzador temporitzador: objecte transitori que gestiona el temps restant en mode contrarellotge. S'usa transient perquè no cal serialitzar-lo.

- int tornActual: representa l'índex del jugador al qual li toca jugar. És una elecció eficient per accedir al jugador des de la llista.
- String estat: s'emmagatzema l'estat de la partida (enCurs, pausada, finalitzada) com a cadena per simplicitat, ja que només cal comparar literals.
- boolean primeraJugada: s'usa per aplicar les regles especials de la primera jugada (passar pel centre del tauler).

Classe Diccionari

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

Aquesta classe representa un diccionari del joc d'Scrabble. Emmagatzema tant l'alfabet de l'idioma al qual representa com les seves paraules. Ofereix mètodes per construir el diccionari, modificar-lo i per consultar l'existència de paraules. S'han aplicat alguns canvis respecte a la primera entrega, amb l'objectiu de millorar el seu funcionament i eficiència.

Estructures de dades:

- String idioma: Identificador del diccionari segons el seu idioma.
- Node arrel: Arrel del DAWG, que representa l'inici de totes les paraules del diccionari. Cada transició es correspon a un caràcter de la paraula. (Veure també classe Node)

El DAWG (Directed Acyclic Word Graph) és una estructura de dades que representa eficientment un conjunt de paraules compartint prefixos i sufixos comuns. Es tracta d'un graf dirigit acíclic on:

- Els camins des de l'arrel fins a nodes finals formen paraules vàlides.
- Els subgrafs equivalents es comparteixen, reduint la memòria usada.
- És més compacte que un trie, especialment per diccionaris grans amb molts sufixos repetits.
- Eliminar Paraules no és trivial, i per poder eliminar una o diverses paraules cal reconstruir-lo tot.

Procés de construcció del DAWG:

1. Ordenació lexicogràfica prèvia de totes les paraules. En tots els casos, les paraules ja s'importen d'un fitxer que està ordenat alfabèticament.
2. Es compara la paraula actual amb l'última afegida per detectar un prefix comú.
3. A partir del punt de divergència:
    - Es fa un reemplaçament/registre dels subgrafs ja existents per compartir-los.
    - Es creen nous nodes per la resta de la paraula.

4. Es marca el node final com a final de paraula.

Aquest procés permet una construcció in-place mantenint el DAWG mínim en tot moment, sense necessitat de reconstruir-lo completament. No obstant, per eliminar paraules del DAWG caldrà reconstruir-lo un altre cop.

- Map<String, Pair<Integer, Integer>> alfabet: Map que representa l'alfabet de diccionari. Cada partida del joc d'Scrabble s'inicialitza amb una bossa de fitxes amb quantitat i puntuacions predeterminades per aquest alfabet.

  La clau és la lletra (o dígraf), i el valor és un parell amb:

  - quantitat: nombre de fitxes disponibles d'aquesta lletra.
  - puntuació: valor que aporta al tauler.

- Map<Node, Node> registre: Registre que permet detectar subgrafs equivalents durant la construcció incremental. Garanteix que el DAWG no té duplicats innecessaris.

- String ultimaParaulaAfegida: Guarda la darrera paraula afegida per assegurar que les noves paraules s'introdueixen en ordre lexicogràfic (requisit per mantenir el DAWG mínim).

Funcionalitats i algoritmes:

- Diccionari(String idioma, List<String> paraulesOrdenades, Map<String, Pair<Integer, Integer>> alfabet): Constructor que inicia l'estructura i construeix el DAWG a partir de paraules ordenades.

- construirDAWG(List<String> paraules): Inicialitza el DAWG buid i afegeix cada paraula de forma incremental. Finalitza registrant l'arrel.

- afegirParaulaIncremental(String paraula): Implementa l'algorisme incremental per inserir la paraula al DAWG. Detecta el prefix comú amb l'última paraula afegida per evitar repetir nodes.

- reemplaçarORegistrar(Node node): Recorre recursivament els fills del node i enregistra (o reemplaça) els subgrafs iguals trobats prèviament. Això garanteix la minimalitat de l'estructura.

- validarParaula(String p): Comprova si una paraula existeix recorrent les transicions des de l'arrel.

- recorre(Node, String prefix, List<String> resultat): Exploració DFS per obtenir totes les paraules del diccionari (útil per mostrar-les o reconstruir-lo).

- getParaules(): Retorna totes les paraules del diccionari reconstruint-les dinàmicament a partir del DAWG.

Justificació de decisions:

- DAWG com a única estructura: S'ha eliminat l'atribut Set<String> paraules per evitar duplicació de dades i garantir un disseny més net i eficient.
- Les paraules es poden recuperar eficientment amb un recorregut DFS.
- Prefix comú i registre: Permet construir de forma increïblement eficient un DAWG mínim, afegint paraules ordenades.
- Eliminar Paraules: En lloc d'eliminar la paraula directament, es reconstrueix el diccionari des de fora amb una nova llista sense la paraula eliminada. Això simplifica la implementació i evita inconsistències.
- TreeMap no necessari: En eliminar el conjunt paraules, no cal mantenir una estructura ordenada a part: l'ordre és garantit per l'entrada inicial.

Classe Jugador

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

La classe Jugador representa un participant d'una partida d'Scrabble. Emmagatzema informació com el nom, la puntuació acumulada i les fitxes que el jugador té disponibles a l'atril.

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:

- actualitzarPuntuacio(int): Suma punts a la puntuació acumulada del jugador després de cada jugada vàlida. És simple però fonamental per a la lògica del joc.
- afegirFitxa(Fitxa): Afegeix una fitxa a l'atril del jugador. Aquesta operació és constant gràcies a l'ús d'ArrayList.
- eliminarFitxa(Fitxa): Elimina una fitxa de l'atril, habitualment després de validar que s'ha col·locat al tauler.
- canviarFitxes(List<Fitxa>): Permet substituir completament les fitxes del jugador, funcionalitat necessària per l'opció de "canviar fitxes" del reglament d'Scrabble.
- colocarFitxa(Fitxa): Elimina una fitxa concreta de l'atril si el jugador la té.
- getFitxesString(): Crea una representació textual de les fitxes, com "A(1) T(2) E(1)", útil per mostrar-les en la interfície gràfica.

Justificació:

- Cada acció concreta del jugador té el seu propi mètode. Això facilita el testatge, la reutilització i l'enteniment del codi.

- Separació de responsabilitats: El jugador coneix el seu estat (fitxes i puntuació), però no té cap coneixement del tauler ni del joc global. Això facilita el manteniment.

- Extensibilitat: Seria fàcil afegir noves funcionalitats com un historial de jugades personals, dificultat adaptativa, sense haver de modificar la lògica central.

Classe Casella

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

La classe Casella representa una unitat bàsica del tauler de Scrabble, és a dir, una posició individual on es poden col·locar fitxes. Cada casella conté informació essencial sobre la seva posició dins el tauler, el tipus de multiplicador de punts que pot tenir (com ara doble lletra, triple paraula, etc.) i la fitxa que hi ha col·locada, si n'hi ha alguna.

Estructures de dades utilitzades

- Tipus primitius (int fila, int columna)
  Utilitzats per identificar la posició exacta de la casella dins la matriu del tauler. Aquest enfocament permet accedir ràpidament a qualsevol casella a través d'una matriu bidimensional.

- String multiplicador
  Aquesta variable indica el tipus de bonus que té la casella. Pot ser una cadena buida (sense bonus) o valors com "DL" (doble lletra), "TL" (triple lletra), "DP" (doble paraula) o "TP" (triple paraula). Aquesta informació és clau per al càlcul correcte de la puntuació de les paraules col·locades.

- Referència a objecte (Fitxa fitxa)
  Aquesta referència apunta a la fitxa que ocupa actualment la casella, o és null si està buida. Permet saber en tot moment quina lletra hi ha a cada posició del tauler i facilita les operacions de col·locació i retirada de fitxes.

Algorisme i ús

- Accés ràpid
  Les caselles s'organitzen dins d'una matriu bidimensional a la classe Tauler, fet que permet accedir-hi eficientment mitjançant els índexs de fila i columna.

- Gestió de l'estat
  La presència directa d'una referència a la fitxa i al multiplicador facilita tant la modificació de l'estat de la casella (col·locar o treure una fitxa, cremar el multiplicador, etc.) com la consulta ràpida sobre si està ocupada o quin bonus ofereix.

- Simplicitat i eficiència
  L'ús de tipus primitius i referències directes fa que la classe sigui lleugera i molt eficient per a les operacions habituals del joc, com validar moviments, calcular puntuacions o actualitzar l'estat del tauler després de cada jugada.

Classe Fitxa

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

La classe Fitxa representa una peça individual del joc de Scrabble. Cada fitxa conté una lletra, un valor de punts associat, i pot tenir la funció de comodí. Aquesta classe és essencial per gestionar el conjunt de fitxes disponibles al joc i calcular la puntuació de les paraules formades.

Estructures de dades utilitzades

- Tipus primitiu (char lletra)
  Representa la lletra associada a la fitxa. Pot correspondre a qualsevol lletra de l'alfabet o al caràcter especial '#', que s'utilitza per indicar un comodí.

- Tipus primitiu (int valor)
  Indica el valor de punts de la fitxa segons la lletra assignada. Aquesta dada és clau per calcular la puntuació total d'una jugada.

Algorisme i ús

- Simplicitat i eficiència
  L'ús de tipus primitius fa que les instàncies de la classe Fitxa siguin molt lleugeres i eficients, especialment en operacions habituals com consultar el valor d'una lletra o fer càlculs ràpids de puntuació.

- Gestió de comodins
  El mètode esComodin() permet identificar de forma ràpida si una fitxa és un comodí. Aquesta funcionalitat és fonamental per validar jugades especials, ja que els comodins poden representar qualsevol lletra sense aportar punts.

Classe Tauler

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

La classe Tauler representa el tauler de joc de Scrabble, estructurat com una graella quadrada de mida 15x15. Aquesta classe s'encarrega de gestionar l'estat de totes les caselles del tauler, la col·locació de fitxes, la validació de les jugades i el càlcul de puntuacions segons les regles del joc.

Estructures de dades utilitzades

- Matriu bidimensional (Casella[][] caselles)
  Representa el tauler físic. Cada posició de la matriu conté una instància de la classe Casella, que encapsula la informació de cada posició (multiplicador, fitxa col·locada, etc.). Aquesta estructura permet accedir i modificar ràpidament qualsevol casella mitjançant coordenades de fila i columna.

- Tipus primitiu (int mida)
  Indica la mida del tauler, és a dir, el nombre de files i columnes.

- Booleà (boolean primeraParaulaColocada)
  Indica si ja s'ha col·locat la primera paraula al tauler. Aquesta dada és important per aplicar les regles especials del Scrabble com el bonus central.

- Classes auxiliars i estructures temporals:

  - List<Pair<Integer,Integer>>
    S'utilitza per gestionar llistes de posicions (fila, columna) de les fitxes col·locades durant una jugada.

  - FitxaColocada (classe interna)
    Permet revertir col·locacions temporals en cas d'error durant el procés de validació.

  - Pair<List<Pair<Integer,Integer>>, Integer>
    Retorna tant les posicions de les fitxes col·locades com la puntuació total d'una jugada, facilitant el control del flux del joc.

Mètodes principals

- Tauler(int mida)
  Constructor que inicialitza el tauler amb la mida donada i crea totes les caselles amb els multiplicadors corresponents.

- Casella getCasella(int fila, int columna)
  Retorna la casella situada a la posició indicada. Permet accedir-hi ràpidament per llegir o modificar el seu contingut.

- Pair<List<Pair<Integer,Integer>>, Integer> colocarParaula(List<Fitxa> fitxesNoves, String paraula, int fila0, int col0, boolean horitzontal)
  Col·loca una paraula al tauler gestionant:

    - la col·locació de les fitxes,
    - el càlcul de la puntuació (tenint en compte els multiplicadors), i
    - la validació de la jugada.

- Retorna les posicions de les fitxes col·locades i la puntuació total. Si la jugada no és vàlida, desfà tots els canvis.

- void confirmar(List<Pair<Integer,Integer>> posFix)
  Després de validar una jugada, crema els multiplicadors de les caselles afectades per evitar que tornin a aplicar-se.

- String construirParaula(int f0, int c0, boolean hor, List<Pair<Integer,Integer>> coords, List<Fitxa> noves)
  Construeix la paraula principal formada a partir d'una posició inicial i una direcció (horitzontal o vertical), combinant les fitxes noves amb les ja existents.

- List<String> getParaulesFormades(List<Pair<Integer,Integer>> posFix, boolean jugadaHoriz)
  Retorna totes les paraules formades durant una jugada, incloent tant la principal com les creuades.

- String mostrarTauler()
  Genera una representació visual del tauler, mostrant les fitxes col·locades i els multiplicadors visibles a cada casella.

- boolean estaBuit()
  Comprova si el tauler està completament buit, útil per determinar si s'ha de col·locar la primera paraula.

- boolean teVeinaOcupada(int fila, int col)
  Verifica si una casella té alguna casella veïna ocupada. Aquesta comprovació és essencial per validar la connectivitat de les jugades posteriors.

- boolean sonContigues(List<Pair<Integer,Integer>> pos, boolean hor)
  Comprova si les posicions indicades són contigües en la direcció especificada (horitzontal o vertical).

- void revertir2(List<Pair<Integer,Integer>> posFix)
  Desfà la col·locació de fitxes noves en cas que la jugada no sigui vàlida o es produeixi un error.

Classe ControladorMaquina

(Marc Gil Moreno, marc.gil.moreno@estudiantat.upc.edu)

La classe ControladorMaquina és el component de la capa de domini que encapsula tota la lògica necessària per a que la IA ("màquina") pugui generar i executar moviments vàlids al tauler d'Scrabble a partir de l'estat actual de la partida de la mateixa forma que ho faria una persona. Rep com a entrada les següents instàncies:

- Diccionari diccionari: arbre de paraules per validar i construir mots
- Tauler tauler: estat actual del tauler, caselles i multiplicadors
- Jugador jugador: instància del jugador "màquina" d'on extreu l'estat de l'atril (fitxes)

A partir d'aquests elements, proporciona mètodes per a:

1. Explorar totes les jugades possibles amb les fitxes de l'atril de la màquina.
2. Validar cada jugada (paraules creuades i puntuacions)
3. Triar la jugada òptima (màxima puntuació) i aplicar-la de manera autònoma
4. Intercanviar fitxes o passar torn si no ha trobat cap jugada vàlida

Estructures de dades utilitzades:

- Matriu bidimensional (int [][] crossChecksDown, int [][] crossChecksAcross)

  Són dues matrius de mida n × n que representan el tauler i emmagatzemen, per cada casella buida. un *bitmask* amb les lletres que es poden col·locar allà sense invalidar paraules creuades.

- List<Play> totesJugades

  Llista dinàmica on s'acumulen totes les jugades vàlides (instàncies de *Play*) que la màquina pot fer en aquell torn.

- List<Casella> linea

  Per a cada fila o columna que es processa, *linea* conté la seqüència de caselles d'aquesta part del tauler.

- List<Anchor> anchors

  Llista de tots els punts "ancla" en una linea –posicions buides adjacents a una fitxa prèviament col·locada al tauler– des d'on es pot començar una paraula.

- List<Fitxa> rack

  Llista on emmagatzem les fitxes disponibles amb les quals la màquina pot generar les possibles jugades.

Algorismes i funcionalitats destacades:

- generateMove()

  Aquesta funció s'encarrega de portar la batuta en tot el procés de generació de moviments de la màquina en un torn.

  1. És primer moviment? Si el tauler està buit, prepara un únic "anchor" central.
  2. Altrament: crida a *ComputeCrossChecks()* per omplir les matrius *crossChecksAcross* i *crossChecksDown*.
  3. Exploració: per a cada fila i columna invoca *generateMoveRow()*, acumulant totes les jugades vàlides a *totesJugades*.
  4. Selecció: escull d'entre totes les juagdes aquella amb una major puntuació, crida a *tauler.colocarParaula(..)* per col·locar les fitxes, actualitza punts i atril; si no hi ha cap jugada, intercanvia fitxes o passa torn.

- generateMoveRow(int idx, boolean transposed)

  L'objectiu és generar totes les jugades vàlides en una línia (fila o columna).

  1. Construeix List<Casella> de la fila *idx* si transposed = false, o de la columna si true.
  2. Detecció d'anchors: crida a la funció *getAnchors(..)* per obtenir tots els punts des d'on començar a construir paraules i els emmagatzema en una List<Anchor>.
  3. Backtracking: per a cada anchor, executa *leftPart(..)* per intentar trobar totes les combinacions de prefix i sufix que formin paraules vàlides.

- computeCrossChecks() / calcularCrossChecks (int fila, int columna, boolean transposed)

  Aquest conjunt de mètodes precalculen, per cada casella buida del tauler, quines lletres es poden jugar allà sense invalidar cap paraula creuada.

  1. Recorregut del tauler: per a cada fila i columna, es formen "linies" de caselles.
  2. Per a cada posició buida: s'exploren les lletres veïnes perpendiculars per construir tots els possibles mots creuats.
  3. Bitmask de crossChecks: s'inicialitza un int de 26 bits (cadascún representant una lletra de l'alfabet), passant a 1 les lletres que formen creuades vàlides (la resta queden a 0).

- getAnchors(List<Casella> linea, int idx, boolean transposed)

  Identifica en una *linea* totes les posiciones on una nova paraula toca caselles ja omplertes, calculant per cada "ancla":

- maxLeft (int): quantes caselles buides consecutives a l'esquerra es poden omplir.
- crossCheck (int): pren directament la informació de les matrius bidimensionals prèviament calculades.

Un cop calculat, crea una instància de la clase *Anchor* i l'afegeix a la llista.

- leftPart(String prefix, Node node, Anchor anchor, List<Casella> linea, List<Play> jugades, int limit, int usedLeft, boolean transposed, List<Fitxa> rack)

  Aquesta funció s'encarrega de construir tots els possibles prefixes a l'esquerra de l'anchor, tenint en comte tant les caselles ja ocupades com les que es troben buides.

  1. Si *limLeft* > *0*, prova de col·locar cada fitxa disponible del rack a la posició actual a l'esquerra,
     a. Navega l'arrel del *trie* per veure si encara pot formar un prefix de paraula
     b. Disminueix *limLeft* i recursiona amb el node fill corresponent
  2. Quan arriba just a la posició del anchor, crida a extendRight(..) per completar el mot.

- extendRight(String soFar, Node node, Anchor anchor, int pos, int usedLeft, List<Casella> linea, List<Play> jugades, boolean transposed, List<Fitxa> rack)

  Un cop hem obtingut el prefix, estenem la paraula cap a la dreta. Explora totes les expansions de sufix amb la mateixa poda del *trie*, assegurant que cada mot complet sigui vàlid.

  1. Inicia en la posició immediatament posterior a anchor, i mentres hi hagi caselles buides o utilitzant aquelles que estiguin plenes formi paraules vàlides:
     a. Si la casella conté fitxa ja col·locada, avança.
     b. Si la casella està buida, prova cada fitxa del rack que el bitmask de crossChecks permet per aquella casella, elimina la lletra del rack i recursiona
  2. Quan troba una paraula completa vàlida, crea una nova instància de *Play* i l'afegeix a la llista de *totesJugades*.

Justifiació:

- Les matrius bidimensionals ens permeten consultar ràpidament quines lletres estan permeses en una posició donada, reduint així l'espai de búsqueda. A més, al calcular-les només un cop per torn accelerant la generació de jugades.
- List <Anchor>: Ens permet iterar només sobre aquelles posicions rellevants en comptes de fer-ho per tota la fila o columna.

- List <Casella>: Facilita iteracions tant en el càlcul de crossChecks com en la detecció d'anchors. A més a més, l'algorisme a implementar distingia entre files i columnes i és una forma eficient de representar-ho.
- List <Play>: Permet ordenar i/o seleccionar la millor jugada ràpidament.
- List <Fitxa>: Ens aporta flexibilitat i rapidesa a l'hora d'afegir, eliminar o utilitzar les lletres de l'atril del jugador màquina
- computeCrossChecks / calcularCrossChecks: Ajuden a reduïr l'espai de búsqueda al indicar quines lletres sí i quines no es poden utilitzar i ens evita haver de comprobar per a cada possible paraula totes les paraules perpendiculars, multiplicant el cost.
- getAnchors + generateMoveRow: Localitzen les posicions candidates i evita iteracions i recursions innecessàries.
- Búsqueda recursiva leftPart + extendRight: Gràcies a la recursió prefix-sufix, al bitmask de crossCheck i el backtracking del rack, l'exploració és molt més ràpida i eficient.

Finalment, només aclarir que els algoritmes més destacats han sortir de l'algorisme "The World's Fastest Scrabble Program" que era obligatori implementar per a fer funcionar el jugador màquina.

DESCRIPCIÓ DE LES CLASSES UTIL

Classe PosicioFitxa

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

La classe PosicioFitxa representa la informació d'una fitxa col·locada al tauler d'Scrabble, incloent la seva posició (fila i columna), la lletra que conté i si és un comodí (joker). Aquesta classe s'utilitza per identificar de manera clara i estructurada cada fitxa jugada, facilitant la gestió dels miviment humans i validacions durant la partida. Es la classe que li passen a partida per validar el moviment humà

Classe Jugada

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

La classe Jugada encapsula la informació d'una acció realitzada per un jugador a Scrabble, emmagatzemant el nom del jugador, la paraula jugada i els punts obtinguts en aquell moviment. Serveix per registrar i consultar les jugades efectuades durant una partida. Es guarda una List de Jugades a la classe partida, per poguer guardar la partida en el historial

Classe DigrafMapper

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

Aquesta classe permet gestionar els dígrafs d'un diccionari de forma eficient mitjançant mapes, per garantir consistència en la comparació i emmagatzematge de paraules.

Estructures de dades:

- Map<String, Map<String, String>> digrafsPerIdioma: Diccionari estàtic que mapeja cada idioma amb un altre mapa de dígrafs (ex: "NY" → "n").

Això permet gestionar de manera centralitzada les conversions de dígrafs en diferents idiomes.

Funcionalitats:

- convertirParaula(...): Substitueix els dígrafs d'una paraula per caràcters unificats interns.
- desferConversioParaula(...): Restaura una paraula amb caràcters unificats als seus dígrafs originals.
- convertirLlistaParaules(...): Aplica la conversió de dígrafs a una llista de paraules.
- convertirLletra(...): Converteix una lletra individual (pot ser un dígraf) al seu símbol unificat.

- desferConversio(...): Restaura un caràcter unificat a la seva forma original com a dígraf.
- getDigrafsPerIdioma(...): Retorna el mapa de dígrafs corresponent a un idioma.

Justificació: L'ús de símbols unificats simplifica la gestió interna del DAWG i altres estructures que només esperen caràcters individuals (char).

Classe Pair<T, U>

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

Aquesta classe és una estructura genèrica simple però potent per representar parelles d'elements (ex: (fila, columna), (quantitat, puntuació)).

Estructures de dades:

- T first: Primer element de la parella.
- U second: Segon element de la parella.

Funcionalitats:

- Constructor Pair(...): Inicialitza els dos valors.
- toString(): Representació en format (first, second).

D'aquesta manera, s'evita haver de crear classes noves per combinacions de dos valors relacionats.

Classe Temporitzador

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

Gestiona el pas del temps dins la partida, permetent iniciar, aturar i consultar el temps, ideal per controlar torns o temporitzacions del joc.

Estructures de dades:

- int duracioInicialSegons: Temps inicial total (en segons).
- int segonsRestants: Temps que queda per acabar.
- Timer timer: Objecte del sistema per programar tasques periòdiques.
- Runnable callbackTempsEsgotat: Acció a executar quan el temps s'esgota.
- TemporitzadorListener listener: Objecte que rep notificacions del temps restant.
- boolean enMarxa: Controla si el temporitzador està actiu.

Funcionalitats:

- iniciar(): Comença el temporitzador si no està ja actiu.

- parar(): Atura el temporitzador.
- reiniciar(): Reseteja i torna a començar.
- getSegonsRestants(): Retorna el temps restant en segons.
- getTempsFormatejat(): Dona el temps restant en format "MM:SS".
- estaEnMarxa(): Indica si el temporitzador està actiu.
- setListener(...): Assigna un objecte que rebrà les notificacions.

Justificació:

- És útil per implementar el mode a contrarellotge del joc.
- L'ús del Timer de Java facilita l'execució d'una acció cada segon.
- L'interfície TemporitzadorListener separa la lògica del temps de la vista.

Interfície TemporitzadorListener

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

TemporitzadorListener és una interfície que defineix un mètode de callback perquè altres classes puguin rebre notificacions quan el temporitzador s'actualitza, com per exemple per refrescar la vista o controlar el temps de la partida.

Mètodes:

- onTick(String tempsRestant): Notifica cada segon amb el temps restant formatat.
- onFinal(): Notifica que el temps s'ha acabat.

Justificació:

Aquesta interfície s'ha creat amb els seguents objectius:

- Facilitar una arquitectura desacoblada: la vista pot actualitzar-se quan el temps canvia, sense accedir directament a la lògica del temporitzador.
- Permetre reutilitzar la lògica del temporitzador en diferents vistes o modes de joc.

Classe Anchor

(Marc Gil Moreno, marc.gil.moreno@estudiantat.upc.edu)

La classe Anchor representa una posició d'ancoratge dins del tauler d'Scrabble, és a dir, una casella veïna d'una fitxa ja col·locada que serveix per a explorar possibles expansions de paraules de l'algorisme màquina. Per a cada "anchor" aquesta classe emmagatzema:

- int fila, columna: posició de la casella
- int crossCheck: bitmask que indica quines lletres es poden col·locar en aquella posició
- int maxLeft: nombre màxim de caselles que una possible paraula es pot estendre cap a l'esquerra

Justificació:

L'algorisme de l'interficie màquina, per a construir jugades vàlides ràpidament, ha de considerar només aquelles posicions on una fitxa nova podria connectar-se amb les existents ("punts d'ancoratge"). Sense aquesta abstracció, el programa hauria de recórrer cada casella del tauler i recomputar per a cadascuna quines lletres són admissibles, cosa que dispararia la complexitat i retardaria el càlcul de la millor jugada. La classe Anchor concentra i precomputes aquesta informació clau, accelerant molt el procés de generació de jugades.

Classe Play

(Marc Gil Moreno, marc.gil.moreno@estudiantat.upc.edu)

La classe Play encapsula una jugada potencial generada per l'algorisme de la màquina en el context del joc de Scrabble. Per a cada instància de la classe s'emmagatzema:

- word (String): la paraula que es col·locaria al tauler.
- fila, columna (int): coordenades on començaria la paraula
- transposed (boolean): orientació de la jugada (true si és vertical, false si és horitzontal)
- score (int): puntuació obtinguda per aquesta jugada

Justifiació:

En el procés de càlcul de la millor jugada, l'algorisme de la màquina explora múltiples possibilitats en paral·lel. La classe Play actúa com a contenidor d'informació de cadascuna d'aquestes possibilitats, permetent:

- Recollir totes les jugades possibles en una llista (List<Play>)
- Comparar de forma eficient totes les jugades en funció de la seva puntuació sense haver de tornar a calcular cap paràmetre.

Classe Node

(Marc Gil Moreno, marc.gil.moreno@estudiantat.upc.edu)

La classe Node representa un node dins d'un DAWG (Directed Acyclic Word Graph), que és una estructura de dades pensada per gestionar i emmagatzemar de manera eficient un diccionari de paraules. En aquesta classe, cada objecte Node correspon a un node del graf, on es guarden els possibles fills a partir d'un caràcter i també si el node és final de paraula. Aquesta manera de funcionar és molt útil perquè permet implementar diccionaris compactes, optimitzant tant l'espai com la velocitat a l'hora de buscar o afegir paraules. En general, el DAWG serveix sobretot per fer cerques de paraules, suggeriments, autocompletar i comprovar si una paraula existeix al diccionari.

Contrincant

(Marc Gil Moreno, marc.gil.moreno@estudiantat.upc.edu)

L'*enum* Contrincant identifica el tipus d'oponent amb el qual l'usuari principal vol jugar la partida:

- MAQUINA ("Maquina)
- JUGADOR ("Jugador)

Cada constant té un **displayName** per representar-lo a la interfície de selecció.

Justifiació. Permet distingir clarament en el controlador de domini si s'ha de crear un segon jugador humà –i mostrar el diàleg de login– o bé inicialitzar la IA (ControladorMaquina). Això simplifica la creació de la partida i evita errors.

Dificultat

(Marc Gil Moreno, marc.gil.moreno@estudiantat.upc.edu)

L'*enum* Dificultat encapsula els diferents nivells de dificultat amb el qual es pot jugar la partida (FACIL, NORMAL, DIFICIL), associant a cada un:

- Un nom per mostrar a l'usuari (String displayName)
- Un multiplicador de puntuació
- La mida de l'atril (int rackSize)

Justificació. D'aquesta manera, tots els paràmetres que modifiquen l'experiència del joc (número de fitxes inicials, bonificació en la puntuació) en funció de la dificultat escollida queden agrupats i són fàcils de mantenir sense haver de canviar la lògica de partida.

ModeJoc

(Marc Gil Moreno, marc.gil.moreno@estudiantat.upc.edu)

L'*enum* ModeJoc defineix els diferents modes de partida disponibles:

- CLASSIC ("Clàssic)
- CONTRARRELLOTGE ("Contrarrellotge")

A cada constant s'associa un **displayName** perquè es pugui mostrar la descripció a la interfície.

Justificació. Centralitzar els modes de joc en un *enum* evita haver d'utilitzar strings i facilita el comportament de la partida (per exemple, si cal activar el temporitzador o no) sense la necessitat d'anar afegint condicionalitats al llarg del codi.

DESCRIPCIÓ DE LES CLASSES DE LA CAPA DE PRESENTACIÓ

Classe ControladorPresentacio

El controlador de la capa de presentació, anomenat ControladorPresentacio, és l'encarregat de gestionar tota la lògica relacionada amb la interfície d'usuari i serveix de pont perquè l'usuari pugui interactuar de manera senzilla i intuïtiva amb totes les funcionalitats del programa. Actua com a intermediari entre la capa de presentació (les vistes) i la capa de domini, coordinant accions com la gestió de les vistes, la creació i el control de les partides, la gestió d'usuaris, la manipulació dels diccionaris i la interacció amb la resta de funcionalitats que ofereix el sistema. Això ho fa mitjançant crides al ControladorDomini, que encapsula tota la lògica del joc, assegurant que la capa de presentació només s'encarrega de mostrar la informació i de captar les accions dels usuaris.

Classe VistaNormes

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

La classe VistaNormes forma part de la capa de presentació del joc i s'encarrega de mostrar les normes o instruccions del joc d'Scrabble a l'usuari. Aquesta vista és accessible des del menú principal i desde la pantalla de pausa d'una partida, i é una finalitat purament informativa, ajudant a nous jugadors a entendre com jugar.

S'estén de JFrame, per tant representa una finestra independent amb contingut visual.

Classe VistaPausa

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

La classe VistaPausa és una vista de la capa de presentació que proporciona una finestra modal per gestionar l'estat de pausa d'una partida d'Scrabble. Apareix quan l'usuari decideix pausar la partida, oferint diverses opcions:

- Continuar: Reprendre la partida tal com havia quedat abans d la pausa
- Guardar: Guardar la partida
- Consultar les normes: Es mostren les normes del joc i un manual d'usuari.
- Sortir al menú principal: Es torna al menú donant l'opció de guardar la partida.

Aquesta finestra no interfereix amb altres vistes actives i pot ser tancada independentment.

Classe VistaResultatPartida

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

La classe VistaResultatPartida forma part de la capa de presentació i és l'encarregada de mostrar el resultat final d'una partida d'Scrabble. Aquesta vista resumeix les puntuacions finals dels jugadors i destaca el guanyador de forma clara i accessible. Permet a l'usuari tornar al menú principal un cop acabada la partida.

Classe VistaLogin

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

La classe VistaLogin és la finestra gràfica encarregada de gestionar l'inici de sessió i el registre d'usuaris a l'aplicació d'Scrabble. Permet als usuaris introduir el seu nom i contrasenya per accedir al sistema o crear un nou compte. Un cop l'autenticació és correcta, redirigeix l'usuari al menú principal del joc. Aquesta classe s'encarrega de la interacció inicial amb l'usuari i de comunicar-se amb el controlador de presentació per validar les credencials.

Classe VistaLoginSegonUsuari

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

La classe VistaLogin és la finestra gràfica encarregada de gestionar l'inici de sessió i el registre d'usuaris a l'aplicació d'Scrabble. Permet als usuaris introduir el seu nom i contrasenya per accedir al sistema o crear un nou compte. Un cop l'autenticació és correcta, redirigeix l'usuari al menú principal del joc. Aquesta classe s'encarrega de la interacció inicial amb l'usuari i de comunicar-se amb el controlador de presentació per validar les credencials.

Classe VistaPartida

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

La classe VistaPartida és la finestra principal de joc d'Scrabble per a l'usuari. Gestiona tota la visualització i interacció durant una partida, incloent el tauler, el faristol, l'estat dels jugadors, el temporitzador i les accions de joc. Aquesta classe repi envia ordres del controlador de presentació i mostra l'estat actualitzat del joc a l'usuari, permetent col·locar fitxes, confirmar jugades, passar torn, canviar fitxes, abandonar la partida o pausar-la.

Funcionalitats principals:

- Visualització del tauler: Mostra el tauler de Scrabble amb els multiplicadors i les fitxes col·locades.
- Gestió del faristol:Mostra i permet manipular les fitxes disponibles per al jugador humà.

- Estat dels jugadors: Mostra el nom, punts i fitxes de cada jugador, així com el torn actual.
- Temporitzador: Mostra i actualitza el temps restant per al torn, gestionant el final de temps.
- Accions de joc: Permet confirmar jugada, passar torn, canviar fitxes, reordenar fitxes, retirar fitxes col·locades provisionalment, abandonar la partida, pausar-la o tancar sessió.
- Interacció drag & drop: Permet arrossegar fitxes del faristol al tauler.
- Actualització dinàmica: Actualitza la vista segons l'estat del model (tauler, fitxes, punts, temps...).

Resum d'ús:

L'usuari pot veure l'estat del joc, col·locar fitxes, fer accions de partida i veure en tot moment la informació rellevant de la partida. La classe s'encarrega de mantenir la coherència visual i funcional entre el model de domini i la interfície gràfica.

Classe VistaHistorial

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

La classe VistaHistorial és una finestra gràfica que mostra a l'usuari l'historial de partides jugades a Scrabble. Permet consultar de manera fàcil el resum i el detall de cada partida: a l'esquerra es mostra una llista amb el guanyador de cada partida i, en seleccionar-ne una, es mostra a la dreta tota la informació detallada d'aquella partida. Aquesta classe s'encarrega de carregar l'historial, agrupar les dades per partida i gestionar la interacció de l'usuari amb la llista d'historial.

Classe VistaCreacioDiccionari

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

La classe VistaCreacioDiccionari és una finestra gràfica de l'aplicació que permet a l'usuari crear un nou diccionari personalitzat per al joc de Scrabble. Mitjançant una interfície gràfica, l'usuari pot introduir el nom del diccionari, una llista de paraules i les lletres amb el seu nombre i puntuació (en format "LLETRA QUANTITAT PUNTS"). Quan es prem el botó de guardar, la vista valida les dades i les envia al ControladorPresentacio, que s'encarrega de generar i guardar els fitxers corresponents.

Classe VistaDiccionaris

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

La classe VistaDiccionaris és una finestra de la interfície gràfica que permet a l'usuari consultar el contingut (les paraules) d'un diccionari prèviament creat o carregat. Mitjançant un desplegable, l'usuari pot seleccionar un diccionari entre una llista proporcionada. En prémer el botó "Mostrar", es crida el controlador de presentació per obtenir les paraules del diccionari seleccionat, que es mostren en una àrea de text.

Classe VistaEstadistiques

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

La classe VistaEstadistiques és una finestra de la interfície gràfica que mostra un resum de les estadístiques personals de l'usuari actiu. Aquestes estadístiques inclouen el total de punts acumulats, la puntuació mitjana, la millor puntuació obtinguda, el nombre de partides jugades, i els punts assolits per cada diccionari utilitzat. La informació s'obté mitjançant el controlador de presentació, que accedeix a les dades de l'usuari des de la capa de domini.

Classe VistaGestioDiccionaris

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

La classe VistaGestioDiccionaris és una interfície gràfica que permet a l'usuari gestionar els diccionaris disponibles a l'aplicació. Ofereix tres funcionalitats principals: mostrar els diccionaris existents, inserir nous mitjançant la vista de creació (VistaCreacioDiccionari), i eliminar diccionaris seleccionats, amb confirmació prèvia i actualització automàtica de la llista. Aquesta vista actua com un punt central de manteniment de diccionaris.

Classe VistaRanquings

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

La classe VistaRanquings és una finestra de la interfície gràfica que mostra diversos rànquings de jugadors del joc. Presenta la informació incloent-hi: el rànquing de puntuació acumulada total, el rècord de punts en una sola partida, el millor percentatge mitjà per partida, la puntuació en mode contrarellotge i els rànquings específics per diccionari. Les dades es recuperen a través del controlador de presentació, i s'afegeixen dinàmicament a la vista mitjançant un mètode auxiliar. Aquesta finestra permet consultar de manera clara i agrupada el rendiment entre usuaris.

Classe VistaGestioUsuari

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

La classe VistaGestioUsuari és una finestra de la interfície gràfica que permet a l'usuari gestionar el seu perfil. Ofereix funcionalitats per canviar l'àlies, modificar la contrasenya i eliminar el compte. Cada acció es gestiona mitjançant diàlegs que demanen confirmació o informació a l'usuari. La classe fa ús del ControladorPresentacio per validar i aplicar les accions corresponents.

Classe VistaMenuPrincipal

(Marc Gil Moreno, marc.gil.moreno@estudiantat.upc.edu)

La classe VistaMenuPrincipal és una interfície gràfica que és el punt d'entrada per a l'usuari un cop ha iniciat sessió. Bàsicament és l'element central que guia a l'usuari cap a totes les funcionalitats principals del joc com jugar una partida, veure les estadístiques personals de l'usuari, gestionar el seu propi perfil, gestionar els diccionaris disponibles, veure l'historial de partides jugades i visualitzar un rànquing global d'entre tots els usuaris. A més, ofereix les funcionalitats de tancar sessió i d'ajuda, que et mostra les normes del joc i de com avançar pel programa.

Classe VistaGestioPartida

(Marc Gil Moreno, marc.gil.moreno@estudiantat.upc.edu)

La classe VistaGestioPartida és una finestra de la interfície gràfica que permet a l'usuari gestionar les seves sessions de joc. Ofereix dues opciones ben visibles: CONTINUAR PARTIDA, que obre un diàleg amb la llista de partides guardades perquè l'usuari pugui continuar una que va deixar a mitges; i NOVA PARTIDA, que llança la pantalla de configuració d'una nova partida (VistaDefinirPartida). A més, disposar d'un botó TORNAR per retrocedir al menú principal sense realitzar cap acció.

Classe VistaDefinirPartida

(Marc Gil Moreno, marc.gil.moreno@estudiantat.upc.edu)

La classe VistaDefinirPartida és una pantalla de configuració prèvia a l'inici d'una partida nova d'Scrabble. Mostra un formulari amb quatre camps desplegables per triar:

- Mode de joc: Clàssic o contrarrellotge
- Contrincant: Màquina o jugador
- Dificultat: Fàcil, normal o difícil
- Idioma: Idioma o diccionari amb el qual es vol jugar

Cada opció es carrega dinàmicament a partir dels enums i dels fitxes disponibles. A més, en la part de baix de la finestra hi ha els botons de TORNAR, per cancel·lar la creació de la partida, i JUGAR, per confirmar els paràmetres escollits i cridar al controlador de presentació per a que instanci la partida amb la configuració desitjada.

DESCRIPCIÓ DE LES CLASSES DE LA CAPA DE PERSISTÈNCIA

Classe ControladorPersistencia

La classe ControladorPersistencia centralitza tota la gestió d'entrada i sortida de dades de l'aplicació Scrabble. S'encarrega de llegir i escriure tant fitxers plans (.txt) —per a diccionaris, alfabets i historial— com fitxers binaris (.ser, .dat) —per a partides serialitzades, usuaris i rànquings. Implementa el patró Singleton per garantir que només hi hagi una instància gestionant l'accés als recursos de persistència.

Estructures de dades internes:

- Ruta base del directori de partides serialitzades ("../DATA/partides/").
- Ruta on es desa el fitxer binari global del rànquing ("../DATA/ranquing.dat").
- Instàncies de gestors
- GestorDiccionari → I/O de diccionaris i alfabets.
- GestorPartida → Serialització i deserialització de Partida.
- GestorUsuari → Llegir/escriure usuaris i contrasenyes.
- GestorHistorial → Històric de partides (fitxer de text).
- GestorRanquingEstadistiques → Rànquings i estadístiques d'usuari.

Funcionalitats

Persistència d'usuaris

- Recupera el conjunt de noms i contrasenyes des del disc.
- Actualitza i valida canvis de nom o contrasenya, garantint la coherència del fitxer de credencials.
- Suprimeix usuaris eliminats de manera definitiva.

Persistència de diccionaris i alfabets

- Persistència de partides
  Serialitza tot l'estat d'una partida (jugadors, tauler, bossa, estat i identificador) en un fitxer binari.
- Deserialitza sense anexar cap dependència de la vista, deixant la reactivació de temporitzadors o components transitoris a la capa de domini.
- Llista de forma ordenada tots els fitxers de partida existents.
- Permet esborrar partides concretes per netejar l'historial de guardats.

Historial de partides

- Registra missatges descriptius de cada sessió en un fitxer de text seqüencial.
- Proporciona accés a la llista sencera d'entrades per mostrar-les a l'usuari.

Rànquing i estadístiques

- Desa i recarrega el rànquing global en un fitxer binari, mantenint l'historial d'èxits.
- Esborra les estadístiques d'un usuari quan deixa de formar part del sistema.
- Elimina fitxers específics de rànquing per usuari, permetent neteja granular.

Justificació de disseny

- Desacoblament: Cada tipus de dades (usuaris, diccionaris, partides, historial, rànquing) té el seu propi "gestor" intern, seguint el principi de responsabilitat única. Modularitat: La façana ControladorPersistencia exposa una API unificada, però delega la lògica concreta als Gestor*, fet que facilita tests i substitució per altres mecanismes (bases de dades, serveis remots).
- Seguretat d'accés: El patró Singleton evita múltiples instàncies que puguin intentar escriure els mateixos fitxers simultàniament.
- Flexibilitat: Permet llistats, càrrega, esborrat i modificació de cada element de forma independent, tot documentat amb excepcions específiques (IOException, ClassNotFoundException).
- Escalabilitat: Afegir nous tipus de dades (per exemple, configuracions de joc) només requeriria un nou gestor i unes quantes línies en aquesta façana, sense alterar la resta de l'aplicació

Classe GestorDiccionari

(Marc Gil Moreno, marc.gil.moreno@estudiantat.upc.edu)

La classe GestorDiccionari és el component central de la capa de persistència encarregat de gestionar tots els fitxers de diccionaris i alfabets utilitzats pel sistema Scrabble. La seva responsabilitat principal és facilitar la càrrega, la modificació i la gestió dels arxius físics on s'emmagatzemen les paraules vàlides per a cada idioma (diccionaris) i la configuració de l'alfabet de cada idioma (incloent-hi la quantitat i el valor de cada lletra). Aquesta classe encapsula tota la lògica d'accés i modificació d'aquests recursos persistents, de manera que la resta del sistema pot treballar amb diccionaris i alfabets sense preocupar-se per detalls concrets de l'accés a fitxers.

Classe GestorHistorial

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

Aquesta classe s'encarrega de gestionar l'historial de partides d'Scrabble, i forma part de la capa de persistència. Concretament, permet desar entrades de text (com ara resultats de partides) en un fitxer de text anomenat historial.txt, i també carregar totes les entrades ja existents. Internament utilitza FileWriter i BufferedReader per treballar amb el fitxer línia a línia, afegint-hi noves entrades sense sobreescriure les anteriors. Aquesta classe és útil per tenir un registre persistent de les partides que ha jugat cada usuari del joc, que després es pot consultar desde menú > historial.

Classe GestorPartida

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

Aquesta classe forma part de la capa de persistència i s'encarrega de gestionar les operacions relacionades amb el desat i la càrrega de partides. En concret, permet guardar una instància de Partida en un fitxer serialitzat .ser, carregar-la posteriorment, llistar totes les partides disponibles al directori ../DATA/partides/, i també esborrar fitxers si cal. És útil per poder continuar una partida en una altra sessió i per mantenir l'estat del joc entre execucions. A més, abans de desar, s'assegura que el directori existeixi, i gestiona els fluxos d'entrada/sortida amb tractament d'errors, cosa que ajuda a fer el sistema més robust i segur. En resum, centralitza tota la lògica de persistència relacionada amb les partides d'Scrabble.

Classe GestorRanquingEstadistiques

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

La classe GestorRanquingEstadistiques s'encarrega de gestionar l'emmagatzematge i recuperació de les dades de rànquings generals i estadístiques individuals dels usuaris. Inclou funcionalitats per: Guardar un objecte de tipus Ranquing en un fitxer (ranquing.dat), carregar aquest rànquing des del fitxer retornant un nou objecte buit si no existeix o hi ha un error, eliminar el fitxer d'estadístiques associat a un usuari específic, sobreescriure el fitxer del rànquing com una forma d'eliminar o actualitzar el seu contingut.

Classe GestorUsuari

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

La classe GestorUsuari s'encarrega de gestionar l'emmagatzematge i actualització dels usuaris del sistema. Opera sobre un fitxer de text (usuaris.txt) on cada línia conté un nom d'usuari i la seva contrasenya separats per dos punts. Les seves funcionalitats principals són: Crear i assegurar l'existència del fitxer i del directori DATA/, carregar usuaris que llegeix el fitxer i retorna un mapa amb els noms d'usuari i les seves contrasenyes, guardar usuaris que escriu tots els usuaris actuals al fitxer, canviar el nom d'un usuari, canviar contrasenya i eliminar un usuari. Aquest gestor és essencial per garantir la persistència i integritat de les credencials dels usuaris dins el sistema.