Ubermind

Grup 43.2

Albert Bausili Fernández (albert.bausili)

Sergio Delgado Ampudia (sergio.delgado.ampudia)

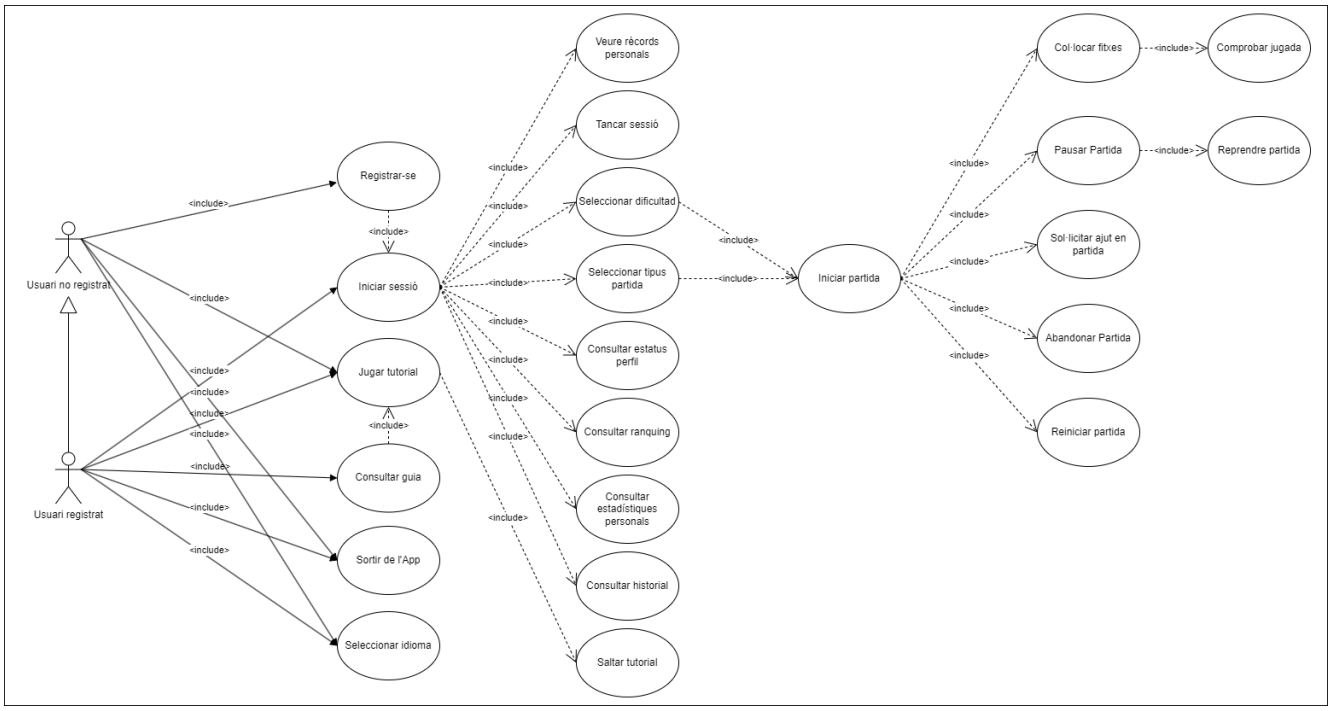
David Molina Mesa (david.molina.m)

Noa Yu Ventura Vila (noa.yu.ventura)

Versió 1.0

Índex

Diagrama de casos d'ús



Registrar-se: Quan un usuari vol registrar-se, introdueix totes les dades necessàries per fer-ho. És a dir, un nom i una contrasenya. El sistema dona d'alta a l'usuari amb les dades indicades i se li assigna un identificador.

Iniciar sessió: Quan un usuari vol iniciar sessió, introdueix les dades necessàries per fer-ho, és a dir, el seu nom d'usuari i la contrasenya. El sistema comprova que les dades introduïdes són correctes i li dona accés al sistema. Aquesta funcionalitat només es pot dur a terme si existeix un usuari al sistema amb aquest nom d'usuari.

Sortir de la app: Quan un usuari vol sortir de l'aplicació ho demana.

Jugar tutorial: Quan un usuari inicia sessió per primer cop o vol jugar una partida de prova juga un tutorial. El sistema començarà una partida de prova amb una guia sobre com jugar.

Consultar guia: Quan un usuari vol consultar les regles, ho demana. El sistema li mostrarà totes les regles del joc. Aquesta funcionalitat només es pot dur a terme si ha iniciat sessió anteriorment.

Seleccionar idioma: L'usuari podrà seleccionar el seu idioma preferit per poder jugar al joc entre els 5 disponibles (castellà, català, anglès, francès i alemany). El sistema mostrarà totes les dades en l'idioma escollit.

Veure récords personals: Quan un usuari ha iniciat sessió pot veure els récords obtinguts en les partides que ha jugat, també pot veure els que li falten per completar.Aquesta funcionalitat només es pot dur a terme si ha iniciat sessió anteriorment.

Tancar sessió: Quan un usuari vol tancar sessió, ho demana. Aquesta funcionalitat només es pot dur a terme si ha iniciat sessió anteriorment.

Seleccionar Dificultat: Quan un usuari vol iniciar una partida, abans haurà de seleccionar una de les 3 dificultats per jugar (fàcil, intermitja o difícil) El sistema començarà una partida amb determinades característiques segons la dificultat escollida. També afectarà el rànquing, ja que hi ha un per cada dificultat. Aquesta funcionalitat només es pot dur a terme si ha iniciat sessió anteriorment.

Seleccionar tipus partida: Abans d'iniciar una partida, l'usuari ha d'escollir entre si serà 'ranked' i es veurà reflectit en el rànquing i les seves estadístiques, o serà 'entrenament', una partida de prova en la qual no es guardaran els resultats. Aquesta funcionalitat només es pot dur a terme si ha iniciat sessió anteriorment.

Consultar estatus perfil: Quan un usuari vol veure el seu estatus en el joc, ho demana. El sistema li mostrarà el seu historial i estadístiques del joc. Aquests resultats inclouen partides guanyades i perdudes, millors puntuacions, temps jugat, duració mitjana... Aquesta funcionalitat només es pot dur a terme si ha iniciat sessió anteriorment.

Consultar rànquing: Quan un usuari vol consultar rànquing, ho demana i selecciona quin tipus de rànquing vol veure, de partides o d'usuaris. El sistema li mostrarà a l'usuari tres llistes de rànquings, un per cada dificultat, fàcil, intermitja o difícil. Si el rànquing és de partides, l'usuari veurà una llista ordenada descendentment amb els millors resultats i el nom d'usuari de l'usuari que ha fet aquell resultat. Si el rànquing és de partides, l'usuari veurà una llista ordenada descendentment amb els millors winrates (partides guanyades/partides perdudes) i el nom d'usuari de qui l'ha aconseguit.

Consultar estadístiques personals: Un usuari pot consultar les seves estadístiques personals quan es al menú principal. El sistema li mostrarà les seves estadístiques obtenides de les partides que ha jugat. Aquesta funcionalitat només es pot dur a terme si ha iniciat sessió anteriorment.

Consultar historial: Quan un usuari es al menú principal, pot consultar el seu historial de partides. El sistema li mostrarà les puntuacions obtenides a les últimes partides en ordre d'antiguitat. Aquesta funcionalitat només es pot dur a terme si ha iniciat sessió anteriorment.

Saltar tutorial: Quan un usuari es troba en una partida de prova i vol abandonar-la ho demana. El sistema finalitzarà la partida i el portarà al menú principal. Aquesta funcionalitat només es pot dur a terme si l'usuari es troba en una partida de prova.

Iniciar partida: Quan un usuari vol iniciar partida i ja ha escollit el tipus de partida i dificultat, ho demana. El sistema començarà una partida amb unes característiques determinades segons l'elecció de l'usuari. Aquesta funcionalitat només es pot dur a terme si l'usuari ha iniciat sessió i ha escollit tipus de partida i dificultat.

Col·locar fitxes: Quan un usuari es troba a una partida, és codebreaker i vol col·locar fitxes, ho demana. Aquesta funcionalitat només es pot dur a terme si l'usuari ha iniciat sessió i ha començat una partida.

Aturar Partida: Quan un usuari es troba a una partida i la vol aturar, ho demana. El sistema farà pausa i l'usuari podrà retornar a la partida en el mateix estat en el qual es trobava quan vulgui. Aquesta funcionalitat només es pot dur a terme si l'usuari ha iniciat sessió i ha començat una partida.

Sol·licitar ajut en partida: Quan un usuari es troba en una partida i vol sol·licitar ajuda, ho demana. El sistema li mostrarà una pista. Aquesta funcionalitat només es pot dur a terme si l'usuari ha obert sessió i ha començat una partida, a més, només es podrà fer una única vegada per partida.

Abandonar partida: Quan un usuari vol finalitzar i abandonar una partida, ho demana. El sistema l'acabarà i portarà a l'usuari al menú principal. Els resultats d'aquesta partida no es guardaran i no apareixen al rànquing o a les estadístiques de l'usuari. Aquesta funcionalitat només es pot dur a terme si l'usuari ha iniciat sessió i ha començat una partida.

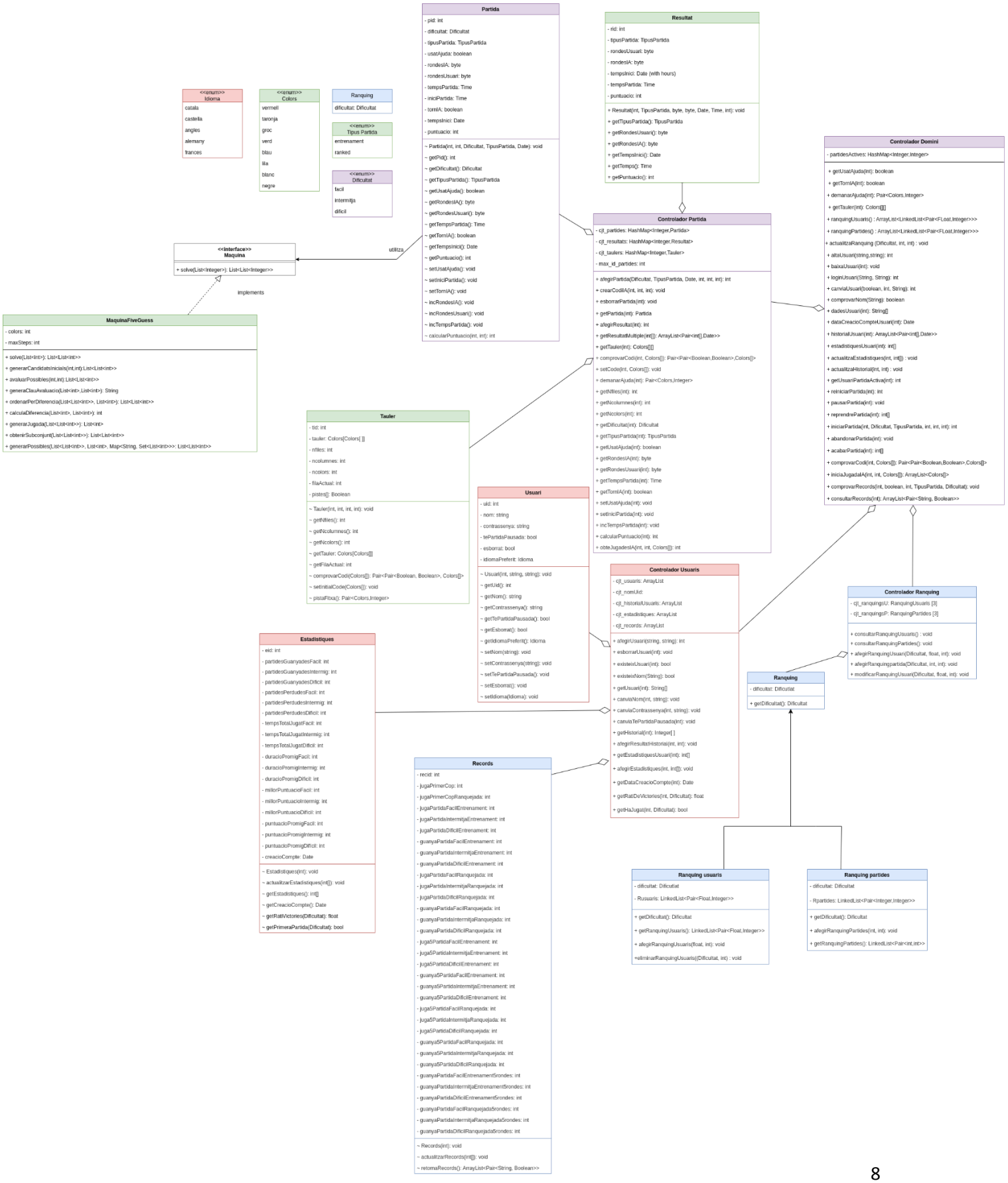
Reiniciar partida: Quan un usuari vol finalitzar i començar una altra (reiniciar partida), ho demana. El sistema l'acabarà i començarà una partida amb les mateixes característiques que les que estava jugant l'usuari. Els resultats d'aquesta partida no es guardaran i no apareixen al rànquing o a les estadístiques de l'usuari. Aquesta funcionalitat només es pot dur a terme si l'usuari ha iniciat sessió i ha començat una partida.

Comprovar jugada: Quan un usuari que es troba en una partida i ha col·locat les fitxes al tauler vol comprovar la seva jugada, ho demana. El sistema informarà l'usuari sobre la seva jugada, és a dir, si ha encertat la combinació o el nombre de files encertades. Aquesta funcionalitat només

es pot dur a terme si l'usuari ha iniciat sessió, ha començat una partida i ha col·locat fitxes al tauler.

Reprendre partida: Quan un usuari que havia pausat una partida vol reprendre-la, ho demana. El sistema mostrarà la partida tal com estava al moment de la pausa i l'usuari podrà continuar jugant. Aquesta funcionalitat només es pot dur a terme si l'usuari ha iniciat sessió, ha començat una partida i l'ha pausat.

Diagrama de Disseny



Repartició de les classes

Blau: Sergio Delgado Ampudia

Verd: David Molina Mesa

Lila: Noa Yu Ventura

Rosa: Albert Bausili Fernandez

Especificació de les classes

Enums

- Dificultat:
    - Enum que representa les possibles dificultats de les partides.
    - Pot ser fàcil, intermig o difícils.

- Idioma:
    - Enum que representa els possibles idiomes en que es pot jugar.
    - Pot ser castellà, català, anglès, francès o alemany.

- TipusPartida
    - Enum que representa el tipus de partida que es pot jugar.
    - Pot ser ranked o entrenament.

- Colors
    - Enum que representa els possibles colors de les fitxes que hi han a les partides.
    - Pot ser vermell, blau, verd, groc, taronja, rosa, lila, blanc i negre (a més de buit per representar un color que no pertany al codi)

Controlador Domini

Classe que controla totes les funcionalitats del sistema.

Conté una llista d'arrays de pairs per controlar les partides que estan actives en cada moment.

Funcions relacionades amb partida:

- **public Pair<Colors,Integer> demanarAjuda(int uid):** Funció que et dóna l'ajuda per a l'usuari. Et dóna la posició correcte d'una de les fitxes del codi del codemaker
    - Cost: O(n); on n és el nombre de columnes del tauler

Funcions relacionades amb rànquing:

- **public ArrayList<LinkedList<Pair<Float, Integer>>> ranquingUsuaris()**: Obté els rànquings d'uauris
    - Cost: Cost: O(1) en tots els casos

- **public ArrayList<LinkedList<Pair<Integer, Integer>>> ranquingPartides()**: Obté els rànquings de partides
    - Cost: O(1) en tots el casos

- **public void actualitzaRanquing(Dificultat dif, int uid, int puntuacio):** Quan un usuari acaba una partida s'han d'actualitzar els rànquings de partides i usuaris
    - Cost: O(1) en tots els casos

Funcions relacionades amb usuari:

- **public int altaUsuari(String nomU, String contrassenyaU)**: Dona d'alta un nou usuari al sistema amb atributs nom i contrasenya
    - Cost: O(1) cas normal, O(n) cas pitjor
    - Excepcions: IncompatibleClassChangeError No s'han pogut modificar les llistes de la class

- **public void baixaUsuari(int uid)**: Dona de baixa un usuari del sistema
    - Cost: O(1)
    - Excepcions:Si l'usuari no existeix, es llença IllegalArgumentException

- **public int loginUsuari(String nom, String contrasenya)**: Intenta fer login amb les credencials introduides
    - Cost: O(1)

- **public void canviaUsuari(byte mode, int uid, String[] nou)**: Funció que canvia el nom de l'usuari, la contrassenya o els dos.
    - Cost: O(1)
    - Excepcions: IllegalArgumentException Mode conté un valor no contemplat

- **public boolean comprovarNom(String nom)**: Comprova si existeix un usuari amb el nom introduit.
    - Cost: O(1) en tots els casos

- **public String[] dadesUsuari(int id) { return cu.getUsuari(id);** Funció per obtenir totes les dades d'un usuari. Retorna un array de quatre posicions amb els parametres en l'ordre uid, nom, contrassenya, tePartidaPausada
    - Cost: O(1)

- **public Resultat[] historialUsuari(int id):** Obté l'historial de l'usuari en forma d'array de resultats de les partides que ha jugat l'usuari. Retorna un array de resultats
    - Cost: O(1)

- **public int[] estadistiquesUsuari(int id)**: Funció que obté totes les estadístiques del usuari.Retorna un array
    - Cost: O(1)

- **public void actualitzaEstadistiques(int uid, int[] resultat)**: Quan un usuari acaba una partida, s'han d'actualitzar les seves estadístiques
    - Cost: O(1)

- **public void actualitzaHistorial(int uid, int rid)**: Quan un usuari termina una partida, s'ha d'actualitzar el seu historial
    - Cost: O(1)

Funcions per la gestió de partides:

- **public int reiniciarPartida(int uid)**: L'usuari inicia una nova partida. Aparellem l'usuari amb la partida que acaba de començar
    - Cost: O(1)

- **public void pausarPartida(int id):** Funció per pausar una partida activa
    - Cost: O(1)

- **public void reprendrePartida(int id):** Funció per reanudar una partida pausada amb l'estat en que es trobava abans
    - Cost: O(1)

- **public void iniciarPartida(int uid, Dificultat dif, TipusPartida tipusP, LocalDateTime tempsI, int nFiles, int nColumns, int nColors)**: Inicia una nova partida amb tots els atributs necessaris i afegeix una posició a la llista de partides actives
    - Cost: O(1)

- **public void abandonarPartida(int uid):** L'usuari pot abandonar la partida y aquesta no compta per el ranquing ni estadístiques
    - Cost: O(1)

- **public void acabarPartida(int id):** Acabem la partida fent: 1. Afegint el resultat al conjunt de resultats 2. Actualitzant el rànquing 3. Actualitzant l'historial 4. Actualitzant les estadístiques de l'usuari que ha acabat la partida Retorna l'ID de la partida acabada
    - Cost: O(1)

Funcions diverses:

- **public Pair<Boolean,Colors[]> comprovarCodi(int id, Colors[] entered_code):** Comprovem si el codi del codebreaker que s'ha entrat és el correcte i informem si queden més files per seguir jugant una altra ronda. Retornem una parella que conté: 1. un boolean que indica si hi ha més files per jugar una altra ronda 2. un array de tantes posicions com columnes tingui el paràmetre entrat indicant quants colors s'han encertat
    - Cost O(1)

- **public ArrayList<Colors[]> iniciaJugadaIA(int pid, int nColors, Colors[] colorsSeleccionats)**: Comprovem si el codi del codebreaker que s'ha entrat és el correcte i informem si queden més files per seguir jugant una altra ronda
    - Cost: O(n^2) cas pitjor i mig, O(n) cas millor; on n és el nombre de columnes del tauler

- **public void comprovarRecords(int uid, boolean guanyat, int rondesUsuari, TipusPartida tipusP, Dificultat dif):** Comprovem quins récords s'han completat en aquesta partida.
    - Cost: O(1)

- **public ArrayList<Pair<String,Boolean>> consultarRecords(int uid)**: Retorna tota la informació sobre els récords de l'usuari
    - Cost: O(1)

Controlador Rànquing

Classe que controla la funcionalitat del rànquings.

- **public RanquingPartides[] consultarRanquingPartides():** Retornen els conjunts de rànquings de partides
    - Cost O(1)

- **public RanquingUsuaris[] consultarRanquingUsuaris():** Retornen els conjunts d'usuaris
    - Cost O(1)

- **public void afegirRanquingPartida(Dificultat dif, int punt, int uid):** Afegeix una nova posicio al ranquing de partides amb els dos paràmetres de entrada
    - Cost: O(1) en cas millor, O(log(n)) en cas pitjor; on n és el nombre d'usuaris que hi ha al rànquing

- **public void afegirRanquingUsuari(Dificultat dif, float winrate, int uid):** Afegeix una nova posicio al ranquing d' usuaris amb els dos paràmetres de entrada
    - Cost: O(1) en cas millor, O(log(n)) en cas pitjor; on n és el nombre d'usuaris que hi ha al rànquing

- **public void modificarRanquingUsuari(Dificultat dif, float winrate, int uid)**: Quan un usuari obté un nou winrate, s'elimina del rànquing d'usuaris i s'afegeix de nou amb el seu nou winrate
    - Cost: O(1) en cas millor, O(log(n)) en cas pitjor; on n és el nombre d'usuaris que hi ha al rànquing

Classe Rànquing (+subclasses)

Aquesta és una classe abstracta que fa referència als rànquings del sistema, hi ha dos possibles rànquings, de persones amb el seu winrate i identificador d'usuari, i rànquings de partides, amb les millors puntuacions i identificador dels usuaris que l'han fet. Cada rànquing es divideix en 3 diferents segons la dificultat de les partides jugades (fàcil, intermig i difícil). Rànquing te atribut dificultat i un getter.

Les dues subclasses (Rànquing Partides i Rànquing Usuaris) contenen linkedlist de pairs per emmagatzemar els dos atributs que apareixeran al rànquing. També tenen un mètode per poder afegir una nova posició al rànquing cada cop que es juga una partida. Utilitza una cerca binària per fer-ho de forma eficient.

- **RanquingPartides(Dificultat dif)**: Crea un nou rànquing de partides amb la dificultat escollida
    - Cost: O(1)

- **RanquingUsuaris(Dificultat dif)**: Crea un nou rànquing d'usuaris amb la dificultat escollida
    - Cost: O(1)

- **public void afegirRanquingPartides(int punt, int uid)**
- **public void afegirRanquingUsuaris(Float winrate, int uid)**
    - Cost: O(1) millor cas i O(log(n)) pitjor on n és el nombre de partides que hi ha al rànquing

- **public void eliminarRanquingUsuaris(int uid):** Mètode per eliminar una posició del rànquing que elimina la posició d'un jugador al rànquing.

- Cost: O(1) millor cas i O(n) pitjor on n és el nombre de partides que hi ha al rànquing

Controlador Usuaris

Classe que controla les funcionalitats de les classes Usuaris i Estadistiques. Conté 3 estructures d'arrays per emmagatzemar tots els usuaris, historials i estadístiques.

- **public int afegirUsuari(String nomU, String contrassenyaU):** Crea un nou usuari amb el seguent id i l'afageix a la llista de parametres i fa lo propi amb les estadistiques d'aquest usuari
  - Cost: O(1) cas normal, O(n) pitjor cas
  - Excepcions: IncompatibleClassChangeError No s'han pogut modificar les llistes de la classe

- **public void esborrarUsuari(int uid):** Marca a l'usuari amb identificador uid com esborrat del sistema
  - Cost: O(1)
  - Excepcions: L'usuari amb id=uid no existeix

- **private boolean existeixUsuari(int uid):** Comproba si un usuari ha estat esborrat. retorna un boolean per saber-ho.
  - Cost: O(1)

- **public boolean existeixNom(String nom)**: Comproba si existeix un usuari amb el nom introduit
    - Cost: O(1)

- **public int comprovarCredencials(String nom, String contrasenya)**: Comprova els credencials de l'usuari
    - Cost: O(1)

- **public String[] getUsuari (int uid**) :Introdueix les dades de l'usuari en un Array de strings i ho retorna
    - Cost: O(1) en tots els casos
    - Excepcions: IllegalArgumentException si l'usuari no existeix i IndexOutOfBoundsException si l'índex es troba en una posició no contemplada

- **public void canviaNom (int uid, String nouNom)**: Funció per canviar el nom d'usuari a un usuari.
    - Cost: O(1)
    - Excepcions: L'usuari amb id=uid no existeix

- **public void canviaContrassenya (int uid, String novaContrassenya):** Funció per canviar la contrassenya d'un usuari
    - Cost: O(1)
    - Excepcions: L'usuari amb id=uid no existeix

- **public void canviaTePartidaPausada (int uid)**: Funció per marcar una partida d'un usuari com pausada
    - Cost: O(1)
    - Excepcions: L'usuari amb id=uid no existeix

- **public void afegirResultatHistorial (int uid, int rid):** Afegeix l'identificador del resultat que ha realitzat l'usuari uid en el seu historial
    - Cost: O(1) cas normal, O(n) pitjor cas
    - Excepcions: L'usuari amb id=uid no existeix

- **public void afegirEstadistiques(int id, int[] resultat):** Afegeix les estadístiques de resultat a les estadístiques de l'usuari identificat per id
    - Cost: O(1)
    - Excepcions: L'usuari amb id=uid no existeix

- **public ArrayList<Pair<String, Boolean>> consultarRecords(int id):** Reenvia els rècords que l'usuari ha completat i no ha completat juntament amb les descripcions d'aquests
    - Cost: O(1) en tots els casos

- **public void comprovarRecords(int[] res):** Comprova si l'usuari ha realitzat algun record en la seva última partida
    - Cost: O(1) en tots els casos

Classe Usuari

Classe que fa referència als usuaris del sistema que utilitzaran el joc. Els usuaris tenen un nom d'usuari per poder identificar-los i amb el que es veuen reflectits als rànquings, també tenen un nom i una contrasenya amb els que s'han registrat. També tenen dos booleans per saber si tenen una partida pausada i per saber si ha sigut donat de baixa.

- **Usuari(int id, String nomU, String contrassenyaU):** Mètode per crear un nou usuari amb les paràmetres demanats
    - Cost: O(1)

Classe Estadístiques

Classe que conté les funcions i les estructures de dades per a mantenir i actualitzar les estadístiques de l'usuari. Com atributs, conté un identificador, i dos ints per cada dificultat (fàcil, intermèdia, difícil) per saber el nombre de partides guanyades i perdudes. També té dos ints per cada dificultat per saber el temps total jugat de cada usuari i la duració mitjana per partida, millor puntuació de cada usuari i puntuació mitjana per partida. Per últim, conté la data de creació del compte d'un usuari.

- **Estadistiques(int id):** Crea una nou set d'estadístiques per l'usuari amb identificador igual a id i ho inicialitza totes les variables.
    - Cost: O(1)

- **void actualitzarEstadistiques (int[] resultat):** Actualitza les estadístiques del usuari afegint els seus últims resultats
    - Cost: O(1)
    - Excepcions: IncompatibleClassChangeError No s'han pogut canviar les variables de la classe i IllegalArgumentException Argument resultat[0] conté un valor invalid

- **float getRatiVictories (Dificultat diff):** Funció per calcular el winrate d'un usuari (partides guanyades/partides perdudes). Retorna aquest valor.
    - Cost: O(1)
    - Excepcions: IllegalArgumentException L'argument diff conté un valor erroni

- **boolean getPrimeraPartida (Dificultat diff)**: Comproba si la partida que acaba de jugar es la primera en aquella dificultat i retorna un boolean amb el resultat
    - Cost: O(1)

Classe Rècords

Classe que conté les funcions i les estructures de dades per a mantenir i actualitzar les els rècords de l'usuari. Com atributs, conté tots els rècords pero pode saber la quantitat de vegades que l'ha aconseguit. Els rècords són jugar els tipus de partides, guanyar els diferents tipus de partides, jugar 5 partides de cada tipus (1 rècord per tipus), guanyar 5 partides en cada tipus i guanyar cada tipus de partides en 5 rondes.

- **Records(int id):** Crea un nou set de records per l'usuari amb identificador igual a id i ho inicialitza totes les variables
    - Cost: O(1)

- **void actualitzarRecords (int[] resultat):** Funció que actualitza els rècords de l'usuari comprovant si ha realitzat algún a la seva última partida.
    - Cost: O(1)

- **ArrayList<Pair<String, Boolean>> retornaRecords ()**: Reenvia els records que l'usuari ha completat i no ha completat
    - Cost: O(1)

Controlador Partida

Classe que controla les funcionalitats de les classes partida, resultat i tauler. Conté 3 estructures de hashmaps per emmagatzemar partides, resultats i taulers. A més. conté una variable per guardar quina és la ID amb valor màxim que hi ha fins ara.

- **public int afegirPartida(Dificultat dif, TipusPartida tipusP, LocalDateTime tempsI, int nFiles, int nColumns, int nColors)**: Funció per afegir una nova partida a la llista de partides actives i crear el tauler. Retorna el ID de la partida
    - Cost: O(n); on n és el nombre de columnes que té el tauler de la partida que volem afegir
    - Excepcions: IncompatibleClassChangeError No s'han pogut modificar les llistes de la classe

- **public void crearCodiIA(int id, int nColumns, int nColors)**: Funció per crear el codi de la IA quan fa de codemaker
    - Cost: O(n) en el pitjor cas; on n és el nombre de columnes

- **public void esborrarPartida(int id):** Funció per borra una partida i el seu tauler
    - Cost: O(1)
    - Excepcions: IncompatibleClassChangeError No s'han pogut modificar les llistes de la classe

- **public int afegirResultat(TipusPartida tipusP, byte rondesUsuari, byte rondesIA, LocalDateTime tempsI, Duration tempsP, int puntuacio):** Funció per afegir el resultat d'una partida acabada a la llista de resultats. Retorna l'ID del resultat creat.
    - Cost: O(1)
    - Excepcions: IncompatibleClassChangeError No s'han pogut modificar les llistes de la classe

- **public ArrayList<Pair<int[],LocalDateTime>> getResultatMultiple(Integer[] ids)**: Retorna un array de resultats amb els identificadors que entren pel parametre ids
    - Cost: O(1) en el millor cas, O(n) en el pitjor cas; on n és el nombre de resultats que volem obtenir (la mida del array)
    - Excepcions: IllegalArgumentException una o més de les ID de resultats no existeixen

- **public Pair<Boolean,Colors[]> comprovarCodi(int id, Colors[] entered_code)**: Funció amb la que es comprova si el codi del codebreaker que s'ha entrat és el correcte i informem si queden més files per seguir jugant una altra ronda. Retorna una parella que conté: 1. un boolean que indica si hi ha més files per jugar una altra ronda 2. un array de tantes posicions com columnes tingui el paràmetre entrat indicant quants colors s'han encertat.

    - Cost: O(n^2) cas pitjor i mig, O(n) cas millor; on n és el nombre de columnes del tauler

- **public Pair<Colors,Integer> demanarAjuda(int id): public Pair<Colors,Integer> demanarAjuda(int id**): L'usuari demana ajuda en la partida. La funció fa: 1. Si l'usuari ha encertat la fitxa del codi de l'esquerra del tot, llavors li confirmem que l'ha encertat 2. Si no l'ha encertat, li indiquem de quin color és la fitxa de l'esquerra del tot
    - Cost: Cost: O(n); on n és el nombre de columnes del tauler

- **public void incTempsPartida(int id):** Incrementem el temps de partida, la partida ja té els valors necessaris per calcular en quant s'ha dincrementar

- **public int calcularPuntuacio(int id):** Funció per calcular la puntuació total d'un usuari al acabar una partida
    - Cost: O(1)

- **public ArrayList<Colors[]> obteJugadesIA(int id, int nColors, Colors[] colorsSeleccionatsUsuari)**: Recolecta tots els intents que fa la IA en la resolució del codi
    - Cost:O(maxSteps * (colors^n + n^3)); on n és la longitud de la solució i colors és el nombre de colors que es poden utilitzr

Classe Partida

Classe que fa referència a totes les partides del joc. Conté els següents atributs: Un int per identificar una partida, en enum per saber la dificultat i altre per saber el tipus de partida, un boolean per saber si s'ha utilitzat l'ajuda i altre per saber si es el torn de la IA o de l'usuari. Un byte per saber el nombre de rondes de la IA i altre per les de l'usuari. Finalment té la duració de la partida i el temps d'inici.

- **Partida(int id, Dificultat dif, TipusPartida tipusP, LocalDateTime tempsI)**: Crea una nova partida amb els paràmetres demanats, i els que no ens dónen tenen el valor per defecte. Retorna la partida creada.
    - Cost: O(1)

- **void incRondesIA()**: Posem l'atribut rondesIA de la partida al valor que ens passen per paràmetre
    - Cost: O(1) en tots els casos
    - Excepcions: IncompatibleClassChangeError si no s'han pogut canviar les variables de la classe

- **void incRondesUsuari()**: Posem l'atribut rondesUsuari de la partida al valor que ens passen per paràmetre
    - Cost: O(1) en tots els casos
    - Excepcions: IncompatibleClassChangeError si no s'han pogut canviar les variables de la classe

- **void incTempsPartida(Duration sumTemps)**: Funció incrementa l'atribut tempsPartida de la partida en el valor que passen per paràmetre
    - Cost: O(1)
    - Excepcions: IncompatibleClassChangeError No s'han pogut canviar les variables de la classe

- **int calcularPuntuacio(byte nColumns, byte nColors):** Funció per calcular la puntuació final de la partida. Es retorna aquesta puntuació
    - Cost: O(1)
    - Excepcions: IncompatibleClassChangeError No s'han pogut canviar les variables de la classe

- **ArrayList<Colors[]> intentsMaquina(int nColors, Colors[] solucio)**: Recolecta tots els intents que fa la IA en la resolució del codi
    - Cost: O(maxSteps * (colors^n + n^3)); on n és la longitud de la solució i colors és el nombre de colors que es

Classe Tauler

Classe que fa referència al tauler de la partida del joc. Conté com atributs: Un int per identificar-lo, un int per nombre de files, altre per nombre de columnes i altre per número de colors.

- **Tauler(int _tid, int _nfiles, int _ncolumnes, int _ncolors):** Crea un nou tauler amb els paràmetres demanats
    - Cost: O(1)

- **public Pair<Boolean,Colors[]> comprovarCodi(Colors[] codiProposat):** Funció que comprova el codi que proposa el Codebreaker amb el que proposa el Codemaster retornant si s'han arribat a l'última fila del tauler i la resposta del Codemaster al codi proposat. Retorna un pair amb un boolean que indica si s'ha arribat al màxim de files i la resposta de colors * (negre, blanc i buit) que donaria el Codemaster en format Array
    - Cost: O(n^2) cas pitjor i mig, O(n) cas millor

- **void setInitialCode(Colors[] codiSetejar):** Col·loca el codi a endevinar del Codemaker a la fila 0 del tauler (cada color a una cela)
    - Cost: O(n); on n és la longitud del codi

- **public Pair<Colors, Integer> pistaFitxa()**: Obté una pista aleatoria que no s'hagi donat anteriorment
    - Cost: O(n); on n és la longitud la solucio

Classe Resultat

Classe que fa referència als resultats de totes les partides acabades del joc. Conté com atributs: Un int per identificar cada resultat. Un enum de tipuspartida per saber el tipus, un int per saber el nombre de rondes de l'usuari i altre per saber las de la IA. El temps d'inici, la duració i la puntuació final.

- **Resultat(int _rid, TipusPartida _tipusPartida, int _rondesUsuari, int _rondesIA, LocalDateTime _tempsInici, Duration _temps, int _puntuacio):** Crea un nou resultat amb els paràmetres demanats
    - Cost: O(1)

Interfície Màquina

Interfície que contindrà les classes que implementen implementacions dels algorismes Genetic i Five Guess.

Classe MaquinaFiveGuess

Classe que implementa la Interfície Maquina amb l'algorisme Five Guess. Conte com atributs: Un int per identificar el nombre de colors són que presents a la solució i un altre int per limitar la quantitat de generacions de possibles solucions.

- **public List<List<Integer>> solve(List<Integer> solution) throws Exception:** Crea una llista de combinacions proposades com solucions. Aquestes són les jugades que fa la maquina. La longitud d'aquesta depèn de quantes passes ha necessitat l'algorisme per trobar la solució, poden ser maxSteps passes si no la troba.
    - Cost: O(maxSteps * (colors^n + n^3)); on n es la longitud de la solució

- **public List<List<Integer>>generarCandidatsInicials(int posicions, int colors):** Crea un llista de combinacions candidates inicials a partir del nombre de posicions que hi ha al tauler (les columnes) i el nombre de colors amb el que es juga.
    - Cost: O(colors^posicions), atès que es tracta a una combinatòria d'aquests.

- **public Map<String, Set<List<Integer>>> avaluarPossibles(List<List<Integer>> possibles, List<Integer> solucio):** Crea un diccionari d'avaluacions - conjunt de combinacions a partir d'una llista de combinacions possibles i la solució a la qual es vol arribar. Aquesta funció el que fa és agrupar totes les combinacions que tinguin mateixa avaluació.
    - Cost: O(n^m) essent n la longitud de la llista i m la longitud de les subllistes

- **private String generaClauAvaluacio(List<Integer> solucio, List<Integer> possible):** Genera una clau d'avaluació donada una combinació possible comparant-se amb la solució a la qual es vol arribar. Aquesta clau ve a ser una etiqueta que avalua que propera és aquesta possible solució a la solució real.

- Cost: O(n^2) cas pitjor i mig, O(n) cas millor (essent n la longitud de les llistes solució i possible)

- **public List<List<Integer>> ordenarPerDiferencia(List<List<Integer>> possibles, List<Integer> solucio):** Ordena una llista de combinacions que li passem per paràmetre per grau de diferència amb la combinació de la solució que també li passem per paràmetre. Aquest grau de diferència representa quant difereix un codi d'un altre.
    - Cost: O(n*k) cas pitjor, mig i millor (tenint en compte que calcularDiferencia() té cost O(n) essent n la mida de solució i k la quantitat de combinacions possibles)

- **private int calculaDiferencia(List<Integer> codi1, List<Integer> codi2):** Calcula el grau de diferencia entre dos codis donats
    - Cost: O(n) cas pitjor i mig, O(1) cas millor (essent n la mida de les llistes codi1 i codi2)

- **public List<Integer> generarJugada(List<List<Integer>> possibles):** Genera una combinació la qual representarà una de les jugades de la màquina.
    - Cost: O(n) cas pitjor i mig, O(1) cas millor

- **public List<List<Integer>> obtenirSubconjunt(List<List<Integer>> possibles):** Obté un subconjunt de la llista de combinacions possibles (el primer 20% d'aquesta)
    - Cost: O(n) cas pitjor i mig, O(1) cas millor (essent n la longitud de la llista de possibles)

- **private List<List<Integer>> generarPossibles(List<List<Integer>> possibles, List<Integer> jugada, Map<String, Set<List<Integer>>> avaluacions):** Filtra la llista de possibles combinacions de tal manera que només retorna aquelles que tenen la mateixa clau d'avaluació que la jugada que li passem per paràmetre
    - Cost: O(n^3) cas pitjor i mig, O(1) cas millor (essent n la longitud de la llista de possibles)

Estructures de dades i algorismes

De cara als algorismes, en aquesta primera entrega s'ha implementat l'algorisme Five Guess. Aquest algorisme implementa una estratègia en el qual es generen totes les combinacions possibles i se les avalua per tal d'així saber quines d'elles estan més properes a la solució que volem arribar. A més d'això se les agrupa segons com la "nota" amb la qual han estat avaluades per així iterar sobre les millors combinacions possibles. Aquest algorisme utilitza estructures de dades com List (per representar els codis) , HashSets (per agrupar-los en conjunts) i HashMap (per agrupar aquests per "nota" d'avaluació). Quant a la complexitat de l'algorisme trobem que donat a les cerques e insercions que es realitzen el seu cost és de O(maxSteps * (colors^n + n^3)) on n és la longitud de la solució, colors es nombre de colors amb el qual es juga aquella partida i maxSteps el nombre màxim de jugades que fa la màquina.