**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 3**](#_heading=h.oo6wxmgv507d)

[**Diagrama de Disseny 12**](#_heading=h.hbtqhmvei44y)

[**Repartició de les classes 13**](#_heading=h.56kmjd7w8r1h)

[**Especificació de les classes 13**](#_heading=h.a9cnaga4wxuj)

[Enums 13](#_heading=h.e46d74tg0nov)

[Controlador Domini 14](#_heading=h.11yqunavcu84)

[Funcions relacionades amb partida: 14](#_heading=h.upv4md2o52c3)

[Funcions relacionades amb rànquing: 14](#_heading=h.ygwltsgerj6u)

[Funcions relacionades amb usuari: 15](#_heading=h.bg1wlhonjwkj)

[Funcions per la gestió de partides: 17](#_heading=h.25zm9hgy4qec)

[Funcions diverses: 18](#_heading=h.kviulx3joyl0)

[Controlador Rànquing 19](#_heading=h.oyt5mhiu473k)

[Classe Rànquing (+subclasses) 20](#_heading=h.bb2nyhkrb13i)

[Controlador Usuaris 21](#_heading=h.iwhtt1oxuko0)

[Classe Usuari 23](#_heading=h.66sofnagbabq)

[Classe Estadístiques 24](#_heading=h.6t2gv7isjpui)

[Classe Rècords 25](#_heading=h.g4u3zmk3c8ng)

[Controlador Partida 25](#_heading=h.2q29v2bbfm0i)

[Classe Partida 27](#_heading=h.cw0yb3lq1z5h)

[Classe Tauler 29](#_heading=h.c36bvlainnys)

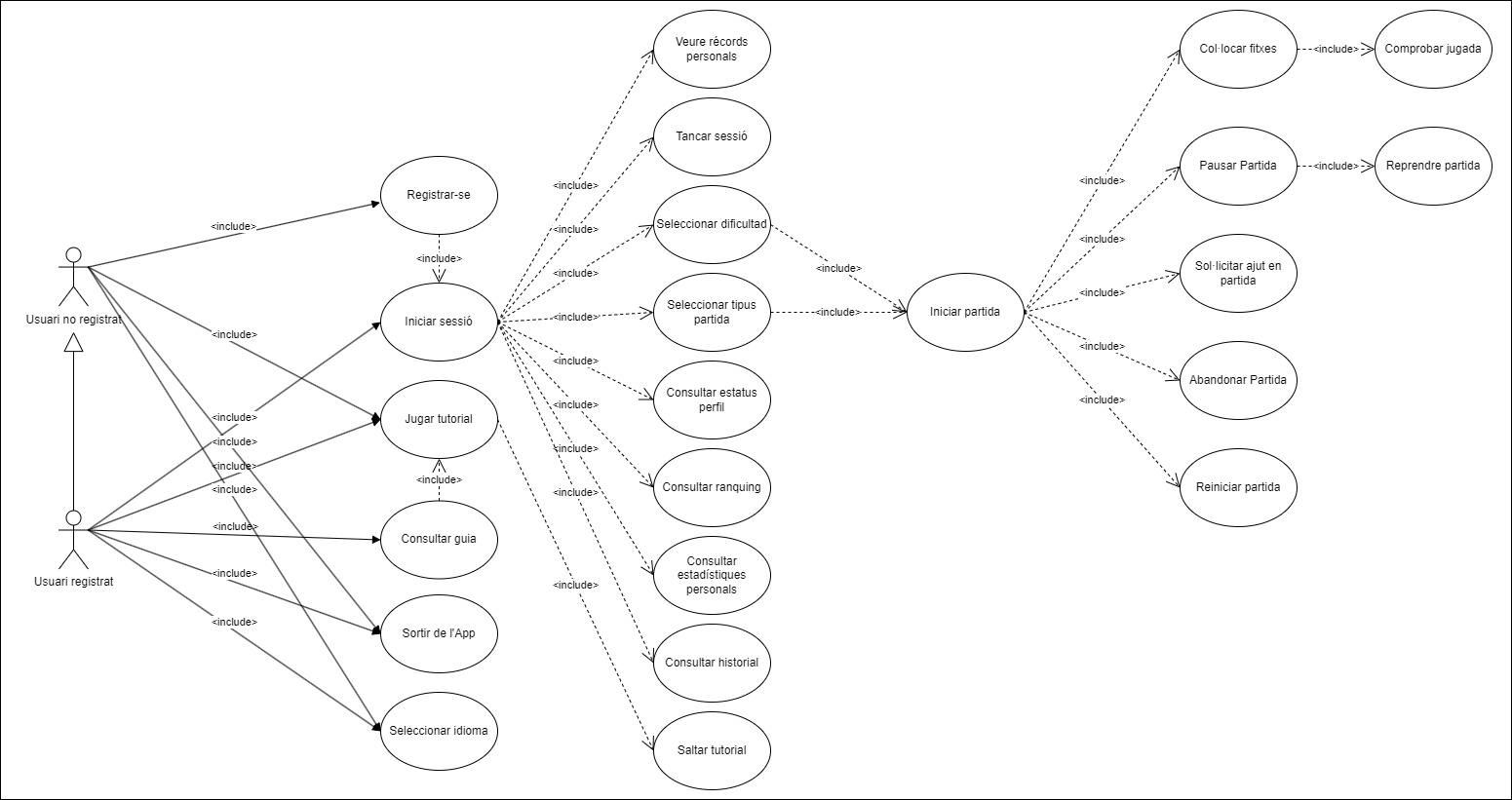
[Classe Resultat 30](#_heading=h.a14g37jcsaas)

[Interfície Màquina 31](#_heading=h.vb7pahekme7j)

[Classe MaquinaFiveGuess 31](#_heading=h.cxprz8lbvpia)

[**Estructures de dades i algorismes 33**](#_heading=h.h7xiwoy6adzw)

## Diagrama de casos d'ús



**Registrar-se:**

* **Actor:** Usuari
* **Detonant:** Un usuari nou entra a l’aplicació i es vol registrar
* **Flux principal:** Es mostra una finestra on s’ha d'introduir el nom i contrasenya**,** l’usuari **i**ntrodueix totes les dades El sistema dona d'alta a l'usuari amb les dades indicades i se li assigna un identificador
* **Flux alternatiu:** L’usuari decideix no registrarse i només jugar partida de prova
* **Flux d’error:** El sistema no enregistra correctamente a l’usuari i mostra un missatge dient que torni a provar

**Iniciar sessió:**

* **Actor:** Usuari
* **Detonant:** un usuari enregistrat entra al joc i vol iniciar sessió
* **Flux principal:** 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.
* **Flux alternatiu:** L’usuari decideix crear un altre compte
* **Flux d’error:** L’usuari no está enregistrat

**Sortir de la app:**

* **Actor:** Usuari
* **Detonant:** Un usuari no vol jugar més i decideix sortir del joc
* **Flux principal:** l’usuari demana sortir fent clic a la creu de la esquina
* **Flux alternatiu:** L’usuari decideix tornar al menú d’inici o abandonar una partida

**Jugar tutorial:**

* **Actor:** Usuari
* **Detonant:** Un usuari nou vol jugar una partida de prova per aprendre
* **Flux principal:** Quan un usuari inicia sessió li apareix la posibilitat de jugar la partida de prova amb un clic, també pot fer-ho desde el menú principal. El sistema començarà una partida de prova amb una guia sobre com jugar
* Flux alterntiu: L’usuari rebutja aquesta posibilitat
* **Flux d’error:** Si hi ha cap error en el procés de càrrega de la partida, el sistema mostra un missatge d'error i torna a la pantalla principal del joc

**Consultar guia:**

* **Actor:** Usuari
* **Detonant:** Un usuari en una partida o al menú principal vol consultar les regles
* **Flux principal:** L’usuari fa clic en el botó de consultar guía. El sistema li mostrarà totes les regles del joc. Aquesta funcionalitat només es pot dur a terme si ha iniciat sessió anteriorment

**Seleccionar idioma:**

* **Actor:** Usuari
* **Detonant:** Un usuari vol seleccionar o canviar l’idioma del joc
* **Flux principal:** 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
* **Flux alternatiu:** L’idioma que busca l’usuari no es troba disponible, no apareixerà
* **Flux d’error:** Si hi ha cap error en el procés de càrrega del nou idioma, el sistema mostra un missatge d'error i torna a la pantalla principal del joc

**Veure rècords personals:**

* **Actor:** Usuari
* **Detonant:** Un usuari vol veure els récords que ha obtingut a les partides
* **Flux principal:** L’usuari entra al seu perfil i fa clic en veure rècords, li apareixen els rècords aconseguits i els que li falten. Aquesta funcionalitat només es pot dur a terme si ha iniciat sessió anteriorment
* **Flux alternatiu:** L’usuari no ha completat cap récord, nomès li apareixeràn els que falten per completar
* **Flux d’error:** Si hi ha cap error en el procés de càrrega del rècords, el sistema mostra un missatge d'error i torna a la pantalla principal del joc

**Tancar sessió:**

* **Actor:** Usuari
* **Detonant:** Un usuari vol tancar la seva sessió
* **Flux principal:** L’usuari fa clic a tancar sessió desde el menú principal. Aquesta funcionalitat només es pot dur a terme si ha iniciat sessió anteriorment
* **Flux alternatiu:** L’usuari surt del joc sense tancar sessió, es manté la sessió oberta

**Seleccionar Dificultat:**

* **Actor:** Usuari
* **Detonant:** Un usuari inicia una partida
* **Flux principal:** 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
* **Flux d’error:** Si hi ha cap error en el procés de càrrega de la partida, el sistema mostra un missatge d'error i torna a la pantalla principal del joc

**Seleccionar tipus partida:**

* **Actor:** Usuari
* **Detonant:** Un usuari inicia una partida
* **Flux principal:** 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
* **Flux d’error:** Si hi ha cap error en el procés de càrrega de la partida, el sistema mostra un missatge d'error i torna a la pantalla principal del joc

**Consultar estatus perfil:**

* **Actor:** Usuari
* **Detonant:** un usuari vol veure el seu estatus
* **Flux principal:** Quan un usuari vol veure el seu estatus en el joc, ho demana desde el seu perfil. 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
* **Flux alternatiu:** Es un usuari que no ha jugat partides i no li apareix res.
* **Flux d’error:** Si hi ha cap error en el procés de càrrega del seu estatus, el sistema mostra un missatge d'error i torna a la pantalla principal del joc

**Consultar rànquing:**

* **Actor:** Usuari
* **Detonant:** L’usuari vol consultar els rànquings
* **Flux principal:** 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
* **Flux alternatiu:** No hi ha cap partida jugada i no es mostra res
* **Flux d’error:** Si hi ha cap error en el procés de càrrega dels rànquings, el sistema mostra un missatge d'error i torna a la pantalla principal del joc

**Consultar estadístiques personals**:

* **Actor:** Usuari
* **Detonant:** Un usuari vol veure les seves estadístiques
* **Flux principal:** Un usuari pot consultar les seves estadístiques personals quan es al seu perfil. El sistema li mostrarà les seves estadístiques obteningudesde les partides que ha jugat. Aquesta funcionalitat només es pot dur a terme si ha iniciat sessió anteriorment
* **Flux alternatiu:** L‘usuari no ha jugat i no es mostra res
* **Flux d’error:** Si hi ha cap error en el procés de càrrega de les estadístiques, el sistema mostra un missatge d'error i torna a la pantalla principal del joc

**Consultar historial**:

* **Actor:** Usuari
* **Detonant:** Un usuari vol veure el seu historial
* **Flux principal:** Quan un usuari es al seu perfil, pot consultar el seu historial de partides El sistema li mostrarà les puntuacions obtingudes a les últimes partides en ordre d’antiguitat. Aquesta funcionalitat només es pot dur a terme si ha iniciat sessió anteriorment.
* **Flux alternatiu:** L‘usuari no ha jugat i no es mostra res
* **Flux d’error:** Si hi ha cap error en el procés de càrrega de l’historial, el sistema mostra un missatge d'error i torna a la pantalla principal del joc

**Saltar tutorial:**

* **Actor:** Usuari
* **Detonant:** L’usuari es troba a un tutorial i vol sortir d’aquest
* **Flux principal:** L’usuari ho demana fent clic al botó de saltar tutorial. 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
* **Flux alternatiu:** L’usuari surt del joc

**Iniciar partida:**

* **Actor:** Usuari
* **Detonant:** Un usuari vol iniciar una nova partida
* **Flux principal:** 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
* **Flux d’error:** Si hi ha cap error en el procés de càrrega de la partida, el sistema mostra un missatge d'error i torna a la pantalla principal del joc

**Col·locar fitxes:**

* **Actor:** Usuari
* **Detonant:** Un usuari que es troba a una partida en el torn en que fa de codebreaker vol col·locar fitxes. Aquesta funcionalitat només es pot dur a terme si l'usuari ha iniciat sessió i ha començat una partida
* **Flux Principal:** L’usuari escogeix l’ordre i color de les fitxes i li dona a completar torn
* **Flux d’error:** Si hi ha cap error en el procés de càrrega del moviment, el sistema mostra un missatge d'error i torna l’inici del procés

**Aturar Partida:**

* **Actor:** usuari
* **Detonant:** Un jugador que està jugant una partida vol aturarla
* **Flux principal:** L’usuari fa clic al botó d’aturar partida. 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
* **Flux alternatiu:** L’usuari vol abandonar la partida
* **Flux d’error:** Si hi ha cap error en el procés de càrrega de la situació de la partida, el sistema mostra un missatge d'error i torna l’inici del procés

**Sol·licitar ajut en partida:**

* **Actor:** usuari
* **Detonant:** Un usuari es troba en una partida i vol sol·licitar ajuda
* **Flux principal:** L’usuari fa clic en demanar ajuda i 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
* **Flux d’error:** Si hi ha cap error en el procés de càrrega de la pista, el sistema mostra un missatge d'error i torna a la situació anterior de la partida

**Abandonar partida:**

* **Actor:** Usuari
* **Detonant:** Un usuari vol finalitzar i abandonar una partida
* **Flux principal:** L’usuari fa clic en abandonar partida i 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
* **Flux alternatiu:** L’usuari vol aturar, reiniciar o guardar la partida

**Reiniciar partida:**

* **Actor:** Usuari
* **Detonant:** Un usuari vol finalitzar i començar una altra (reiniciar partida)
* **Flux principal:** L’usuari fa clic en reiniciar partida. 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
* **Flux alternatiu:** L’usuari vol aturar, abandonar o guardar la partida
* **Flux d’error:** Si hi ha cap error en el procés de càrrega de la nova partida, el sistema mostra un missatge d'error i torna al menú principal

**Comprovar jugada:**

* **Actor:** Usuari
* **Detonant:** Un usuari que es troba en una partida i ha col·locat les fitxes al tauler vol comprovar la seva jugada
* **Flux principal:** L’usuari fa clic en comprovar jugada i 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
* **Flux alternatiu:** L’usuari vol canviar o desfer el seu moviment
* **Flux d’error:** Si hi ha cap error en el procés de càrrega del procés, el sistema mostra un missatge d'error i torna a la situació anterior de la partida

**Reprendre partida:**

* **Actor:** Usuari
* **Detonant:** Un usuari que havia pausat una partida vol reprendre-la
* **Flux principal:** L’usuari fa clic en reprendre partida i 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
* **Flux alternatiu:** L’usuari decideix iniciar una nova partida
* **Flux d’error:** Si hi ha cap error en el procés de càrrega de la partida, el sistema mostra un missatge d'error i torna al menú principal

## 

## 

## 

## 

## 

## 

## 

## 

## Diagrama de Disseny

## Repartició de les classes

Blau: Sergio Delgado Ampudia

Verd: David Molina Mesa

Lila: Noa Yu Ventura

Rosa: Albert Bausili Fernandez

## Capa Dades

### PersistènciaRànquings

Hem fet aquesta implementació per disminuir l'acoblament entre capes. PersistenciaRanquings conté les funcions per a guardar i recuperar les dades dels ranquings d'usuaris i de partides. Té 4 mètodes:

* **public static void guardarRanquingUsuaris(RanquingUsuaris ru, Dificultat dif):** En aquest mètode, entra com a paràmetre un objecte ranquing d'usuaris que volem guardar i la dificultat del ranquing. La funció guarda en un fitxer txt de la forma Usuaris\_"dif".txt el ranquing d'usuaris, si ja existeix el sobreescriu
* **public static void guardarRanquingPartides(RanquingPartides rp, Dificultat dif):** Aquest mètode guarda en un fitxer txt de la forma Partides\_"dif".txt el ranquing de partides per la dificultat dif, si ja existeix el sobreescriu
* **public static RanquingUsuaris tornarRanquingUsuaris(Dificultat dif):** Podem utilitzar aquest mètode per recuperar del disc el rànquing d'usuaris de la dificultat dif
* **public static RanquingPartides tornarRanquingPartides(Dificultat dif):** Podem utilitzar aquest mètode per recuperar del disc el rànquing de partides de la dificultat dif

### 

### Persistència Usuaris

La classe PersistenciaUsuaris conté les funcions per a guardar i recuperar les dades dels usuaris així com de la llista d'aquests. Té 4 mètodes:

* **public static void guardarUsuari(Usuari usr, Estadistiques est, Records rec, Historial hist):** Guarda en un fitxer txt de la forma "uid".txt l’usuari, les estadístiques, els rècords i el seu historial, si ja existeix el sobrescriu
  + Cost: O(1)
* **public static void guardarNomUid(HashMap<String, Integer> nomUid):** Guarda en un fitxer amb el nom nomUid.txt la llista de noms relacionats amb els seus respectius uids, si ja existeix el sobrescriu
  + Cost: O(1)
* **public static Quartet<Usuari, Estadistiques, Records, Historial> tornarUsuari(int uid):** Llegeix del fitxer de l'usuari el seu objecte, el de les seves estadístiques, els seus records i el seu historial i ho torna en un tipus generic. Retorna un tipus generic del tipus Quartet que conté els objectes Usuari, Estadistiques, Records i Historial de l'usuari demanat
  + Cost: O(1)
* **public static HashMap<String, Integer> tornarNomUid():** Llegeix del fitxer de nomUid la llista de noms amb els seus respectius identificadors. Retorna un Hashmap amb els continguts de noms com a key i els seus identificadors com a value
  + Cost: O(1)

### 

### Persistència partida

## Capa Domini

### 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, nomès cambien en les partides de dificultat difícil.
  + 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) en tots els casos
* **public Date dataCreacioCompteUsuari(int id):** Funció amb la que s’bté la data de la creació del compte de l'usuari
  + Cost: O(1) en tots els casos
* **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 dels dos tipus de rànquings, de partides i usuaris, que s'utilitzen constantment cada cop que qualsevol usuari juga una partida, a més, cada usuari pot accedir a qualsevol dels rànquings dividits per dificultat.

**Mètodes**

* **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.

**Atributs**

* **dificultat:** Atribut utilitzat per distinguir la dificultat d’un rànquing (fàcil, difícil o intermig)

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.

**Mètodes**

* **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.

**Mètodes**

* **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 afegirResultatHistorial (int uid, int rid):** Afegeix l’identificador del resultat que ha realitzat l’usuari uid en el seu historial
  + Cost: O(1) en tots els casos
  + 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, retorna una arrayList que conté els rècords amb el text i un boleà indicant si l'usuari els ha assolit o no
  + Cost: O(1) en tots els casos
* **public void comprovarRecords(int[] res):** Comprova si l'usuari ha realitzat algún record en la seva última partida
  + Cost: O(1) en tots els casos
* **void canviarIdiomaUsuari (int uid, Idioma nouIdioma)**: Canviem el nou idioma preferit de l'usuari per l'introduït
  + 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 tene un int per saber si tenen una partida activa i un bool per saber si ha sigut donat de baixa.

**Atributs**

* **uid:** Identificador de l’usuari dins del joc, l’utilitzem per distinguir un usuari d’altres i es el nom que es mostrarà als rànquings
* **nom:** Nom amb el que es registra un usuari al joc i amb el que fa login
* **contrasenya:** Contrasenya amb la que l’usuari es registra al joc i fa login
* **pidPartidaActiva:** int per saber si l’usuari te una partida activa (1) o no (0)
* **esborrat:** Boleà per saber si un usuari s’ha donat de baixa
* **idiomaPreferit:** Atribut per saber quin dels 5 idiomas posibles ha escollit l’usuari peel joc

**Mètodes**

* **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.

**Atributs**

* **eid:** Identificador de les estadístiques que es igual al de l’usuari amb aquelles estadístiques
* **partidesGuanyadesFacil:** Atribut per saber el número de partides que ha guanyat un usuari en partides amb dificultat fàcil
* **partidesGuanyadesIntermig:** Atribut per saber el número de partides que ha guanyat un usuari en partides amb dificultat intermitja
* **partidesGuanyadesIDificil:** Atribut per saber el número de partides que ha guanyat un usuari en partides amb dificultat difícil
* **partidesPerdudesFacil:** Atribut per saber el número de partides que ha perdut un usuari en partides amb dificultat fàcil
* **partidesPerdudesIntermig:** Atribut per saber el número de partides que ha perdut un usuari en partides amb dificultat intermitja
* **partidesPerdudesDificil:** Atribut per saber el número de partides que ha perdut un usuari en partides amb dificultat difícil
* **tempsTotalJugatFacil:** Atribut per saber el número de temps en segons que un usuari ha jugat en partides amb dificultat fàcil
* **tempsTotalJugatIbtermig:** Atribut per saber el número de temps en segons que un usuari ha jugat en partides amb dificultat intermitja
* **tempsTotalJugatDificil:** Atribut per saber el número de temps en segons que un usuari ha jugat en partides amb dificultat difícil
* **duracioPromigFacil:** Atribut per saber el temps promig en segons que un usuari ha passat en partides amb dificultat fàcil
* **duracioPromigIntermig:** Atribut per saber el temps promig en segons que un usuari ha passat en partides amb dificultat intermitja
* **duracioPromigDificil:** Atribut per saber el temps promig en segons que un usuari ha passat en partides amb dificultat difícil
* **millorPuntuacioFacil:** Atribut per saber la millor puntuació d’un usuari en partides de dificultat fàcil, es va actualitzant segons l’usuari aconsegueix millor puntuació en partides
* **millorPuntuacioIntermig:** Atribut per saber la millor puntuació d’un usuari en partides de dificultat intermitja, es va actualitzant segons l’usuari aconsegueix millor puntuació en partides
* **millorPuntuacioDificil:** Atribut per saber la millor puntuació d’un usuari en partides de dificultat difícil, es va actualitzant segons l’usuari aconsegueix millor puntuació en partides
* **puntuacioPromigFacil:** Atribut per saber la puntuació promig d’un usuari en partides de dificultat fàcil, es va actualitzant segons l’usuari juga partides
* **puntuacioPromigIntermig:** Atribut per saber la puntuació promig d’un usuari en partides de dificultat intermitja, es va actualitzant segons l’usuari juga partides
* **puntuacioPromigDifícil:** Atribut per saber la puntuació promig d’un usuari en partides de dificultat difícil, es va actualitzant segons l’usuari juga partides
* **creacioCompte:** Atribut per saber la data en la que un usuari es va donar d’alta al joc

**Mètodes**

* **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.

**Atributs**

* **recid:** Identificador dels récords de cada usuari
* **records[]:** Vector on es guarden tots els records de cada usuari. Hi han 32 posicions, cadascuna consta d’un int que es refereix a un record número de vegades que un usuari ha aconseguit completar un récord

**Mètodes**

* **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 amb la seva descripció
  + Cost: O(1) en tots els casos

#### Classe Historial

Classe que conté tots els atributs i mètodes per gestionar els historials dels usuaris. Són una llista de partides amb resultats que ha jugat un jugador i es guarden perquè l'usuari pugui veure-ho.

**Atributs**

**private final int hid:** Identificador de l’historial

**Mètodes**

* **Historial(int id):** Creadora de l'objecte historial
  + Cost: O(1) en tots els casos
* **ArrayList<Integer> getPartides:** S’obté el llistat de partides que ha jugat un usuari
  + Cost: O(1) en tots els casos
* **void afegirResultat (int rid):** Afegim una partida amb identificador rid a l'historial
  + Afegim una partida amb identificador rid a l'historial

### 

### 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.

**Atributs**

* **max\_id\_partides:** Variable per guardar quina és la ID amb valor màxim que hi ha fins ara en el conjunt de partides

**Mètodes**

* **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.

**Atributs**

* **pid:** Identificador d’una partida activa (que està en estat de pausa o està sent jugada)
* **dificultat:** variable per distingir la dificultat d’una partida (fàcil, intermitja o difícil)
* **tipuspartida:** Variable per distingir el tipus de partida (ranked o de prova)
* **usatAjuda:** Variable per saber si un usuari ha utilitzat la ajuda durant la partida
* **rondesIA:** Variable que contabilitza les rondes que ha jugat la IA en una partida com a codebreaker
* **rondesUsuari:** Variable que contabilitza les rondes que ha jugat un jugador en una partida com a codebreaker
* **tempsPartida:** Variable que conté el temps en segons que ha durat una partida al finalitzar
* **iniciPartida:** Temps exacte en el que s’ha iniciat o reanudat una partida
* **tornIA:** Indica si es el torn de la IA o de l’usuari
* **tempsInici:** Conté en quin moment es va crear la partida
* **puntuació:** Conté la puntuació final d’un usuari a una partida

**Mètodes**

* **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.

**Atributs**

* **tid:** identificador del tauler
* **nfiles:** Número de files que conté el tauler, varia segons la dificultat de la partida
* **ncolumnes:** Número de columnes que conté el tauler, varia segons la dificultat de la partida
* **ncolors:** Número de colors posibles de les fitxes que conté el tauler, varia segons la dificultat de la partida
* **filaactual:** Fila que se està adivinant del tauler en un moment determinat d’una partida
* **pistes:** Posibles pistes que es poden donar durant la partida

**Mètodes**

* **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.

**Atributs**

* **rid:** Identificador d’un resultat d’una partida acabada
* **tipusPartida:** Tipus de partida que ha finalitzat (ranked o entrenament)
* **rondesUsuari:** Número de rondes que ha jugat l’usuari com a codebreaker a una partida finalitzada
* **rondesIA:** Número de rondes que ha jugat la IA com a codebreaker a una partida finalitzada
* **tempsInici:** Moment exacte en el que ha començat la partida
* **temps:** Duració total de la partida en segons
* **puntuació:** Puntuació final de l’usuari a la partida

**Mètodes**

* **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.

**Mètodes**

* **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)

### Classe MaquinaGenetic

Classe que implementa la Interfície Maquina amb l’algorisme genetic.

**Atributs**

* **maxGeneracions:** Cuantes vegades entra al bucle y es igual al número de files del tauler
* **nIndividusPoblacio:** El nombre d’individus que hi haurà en la població escollida
* **nColors:** El nombre de colors que hi han en una partida i s’utilitzen

**Mètodes**

* **List<List<Integer>> solve(List<Integer> solution):** Creadora de l’implementació de la màquina genétic.
  + Cost: O(n) en el millor cas, O(maxGeneracions \* m \* n^3) en el pitjor dels casos; on n és el nombre d'individus de la població, maxGeneracions és els intents
* **double maxFitness(Individu solucio):** Entra com a paràmetre la solució individu solució que l'algoritme ha de trobar y retorna el màxim fitness que es pot aconseguir (és igual al fitness de la solució, que és igual al nombre de gens que té l'individu solució)
  + Cost: O(1) en tots els casos

#### Classe Població

Classe que representa un conjunt d’individus agafats aleatòriament per agafar el millor y fer funcionar l’algoritme genétic. El millor s’obté comparant quants gens de l’individu són iguals a l’individu solució. Si no encertem llavors hem de seguir proposant més individus potencialment bons. Per fer-ho fem evolucionar la població, i fa servir dues maneres: recombinació i mutació. Després, seguim generant diferents individus potencialment bons fins que encertem (ho sabem si els seu fitness és igual al nombre de gens de la solució) o fins que se’ns acabin els intents.

**Mètodes**

* **Poblacio(int tamany, boolean inicialitza, int numGens, int nColors):** Creadora de la població
  + Cost: O(n); on n és igual al tamany
* **int numIndividus():** retorna el nombre d’individus de la població
  + Cost: O(1)
* **boolean existeixIndividu(Individu ind, int limit)**: Indica si ja existeix l'individu o no
  + Cost: O(n\*m); on n és el nombre d'individus que hi ha a la població, i m és el nombre de gens que tenen aquests
* **Individu individuMesApte(Individu solucio):** Inicialitza l'individu demanat a partir de la seva posició a l'individu que ens passen per paràmetre
  + Cost: O(n) en tots els casos; on n és el nombre d'individus de la població
* **Individu seleccionaIndividu(Individu solucio, int numTornejos, int numGens, int nColors):** Seleccionem individus aleatòriament per fer-los competir en el torneig i agafar-ne el millor
  + Cost: O(n\*m) en tots els casos; on n és el nombre d'individus que tindrà el torneig i m és el nombre d'individus que té la població actual
* **Poblacio evolucionaPoblacio(Individu solucio, boolean elitisme, int numTornejos, double umbralRecombinacio, double taxaMutacio, int nColors):** Fem evolucionar la població que tenim en l'actual generació per fer-ne una de millor a la següent generació
  + Cost: O(m\*n^2); on n és el nombre d'individus que té la població i m és el nombre de gens que tenen aquests
* **Individu recombinaIndividus(Individu ind1, Individu ind2, Individu solucio, Individu lastGen)**: Recombina els individus en funció del fitness
  + Cost: O(n) en tots els casos; on n és el nombre de gens que té l'individu

#### 

#### Classe Individu

Classe que serveix per utilitzar l’algorisme genetic, agafa un individu i s’intenta agafar una solució comparant quants gens de l’individu són iguals a l’individu solució.

**Atributs**

* **numGensPerDefecte:** Nombre de gens que te un individu, es igual al nombre de colors de la solució
* **byte[]: gens:** Posició dels seus gens, que fan referencia a la posició de colors
* **fitness:** Nombre que mesura la semblança amb la solució, es calcula veient si la posició dels gens és igual als colors

**Mètodes**

* **Individu(int numGens):** Constructora d’individu
  + Cost: O(1)
* **Individu(List<Integer> solucio):** Constructora exclusivament de l'individu solució
  + Cost: O(n) en tots els casos; on n és el nombre de colors que tingui la solució
* **numGens():** Obtenim el nombre de gens que té l'individu
  + Cost: O(1) en tots els casos
* **byte getGen(int index):** Obtenim el gen que necessitem a partir de la posició on està
  + Cost: O(1) en tots els casos
* **List<Integer> getGens():** Obtenim els gens de tot l'individu
  + Cost: O(n); on n és el nombre de gens que té l'individu
* **void inicialitzaIndividu(int n\_colors):** Assignem valors random als gens de l'individu
  + Cost: O(1) en tots els casos
* **double fitnessIndividu(Individu ind\_solucio):** Calculem el fitness de l'individu a partir de la solució, la qual té fitness màxim
  + Cost: O(n) en tots els casos; on n és el nombre de gens que té l'individu

## Capa presentació

## Estructures de dades i algorismes

### Algorisme FiveGuess

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 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 és nombre de colors amb el qual es juga aquella partida i maxSteps el nombre màxim de jugades que fa la màquina.

### Algorisme Genètic

Aquest algorisme implementa una estratègia en la qual s’intenta encertar el codi solució basant-nos en quant de bo és el nostre individu, és a dir, determinant el “fitness” de l’individu que proposem.

Primer es crea una població (conjunt d’individus) aleatòriament i s’agafa el millor, el qual s’obté comparant quants gens de l’individu són iguals a l’individu solució. Si no encertem llavors hem de seguir proposant més individus potencialment bons. Per fer-ho fem evolucionar la població, i ho fem recombinant els individus de la població. Després, seguim generant diferents individus potencialment bons fins que encertem (ho sabem si els seu fitness és igual al nombre de gens de la solució) o fins que se’ns acabin els intents.

L'algoritme fa servir un array de bytes per als gens dels individus i un d’individus per al conjunt d’individus d’una població, ja que sabem el seu tamany des del primer moment i per tant no és necessari guardar-ho en una estructura dinàmica. El cost és O(n) en el millor cas, O(maxGeneracions \* m \* n^3) en el pitjor dels casos; on n és el nombre d'individus de la població, maxGeneracions és els intents màxims que té l'algoritme per endevinar el codi solució, i m és el nombre de columnes que té la solució.

### 

### Estructures de dades utilitzades

**Controlador Rànquing**

* **ArrayList<LinkedList<Pair<Float, Integer>>> cjt\_ranquingsU**
  + Array de llistes utilitzat per controlar els 3 tipus diferents de rànquings d’usuaris. Es una llista amb 3 posicions ja que hi han 3 rànquings, un per cada dificultat (fàcil, intermitja o difícil). Hem considerat aquesta estructura ya que considerem que una llista es el mes óptim per controlar els 3 tipus de rànquings, que son les linkedlist de pairs ya que hem de estar constantment buscant elements, afegint i eliminant
  + **Alternatives:** Hashmap tambè ens sería útil pero no hem escollit aquesta opció per les operacions d’afegir i eliminar tant constants
* **ArrayList<LinkedList<Pair<Integer, Integer>>> cjt\_ranquingsP**
  + Array de llistes utilitzat per controlar els 3 tipus diferents de rànquings de partides. Es una llista amb 3 posicions ja que hi han 3 rànquings, un per cada dificultat (fàcil, intermitja o difícil). Hem considerat aquesta estructura ya que considerem que una llista es el mes óptim per controlar els 3 tipus de rànquings, que son les linkedlist de pairs ya que hem de estar constantment buscant elements della llista, afegint i eliminant
  + **Alternatives:** Hashmap tambè ens sería útil pero no hem escollit aquesta opció per les operacions d’afegir i eliminar tant constants

**Rànquing Partides**

* **LinkedList<Pair<Integer, Integer>> Rpartides**
  + Llista utilitzada per representar un rànquing de partides que emmagatzema un pair amb la puntuació d’un usuari a una partida i l’identificador d’aquell usuari per, aquesta llista sempre estarà ordenada ja que es controla l’ordre al afegir noves posiciones. Hem utilitzat aquesta estructura ja que és útil per tenir una llista de pairs que s’ha d’ordenar
  + **Alternatives:** Hashmap tambè ens sería útil pero no hem escollit aquesta opció per les operacions d’afegir i eliminar tant constants

**Rànquing Usuaris**

* **LinkedList<Pair<Float, Integer>> Rusuaris**
  + Llista utilitzada per representar un rànquing d’usuariss que emmagatzema un pair amb el winrate (partidesguanyades/perdudes) d’un usuari i l’identificador d’aquell usuari per, aquesta llista sempre estarà ordenada ja que es controla l’ordre al afegir noves posiciones. Hem utilitzat aquesta estructura ja que és útil per tenir una llista de pairs que s’ha d’ordenar
  + **Alternatives:** Hashmap tambè ens sería útil pero no hem escollit aquesta opció per les operacions d’afegir i eliminar tant constants

**Controlador usuaris**

* **ArrayList<Usuari> cjt\_usuaris**
  + Estructura de dades que enmagatzema la llista d'usuaris, les identificadors de la ArrayList son els mateixos uid dels usuaris, quan es crea un usuari aquest s'afegeix al final de la llista, hem triat una llista ja que es una manera eficient de tenir enmmagatzemats i anar afegint tots els usuaris
* **HashMap<String, Integer> cjt\_nomUid**
  + Estructura de dades que relaciona els noms dels usuaris amb la seva uid. Hem triat un map ja permet emmagatzemar dos variables i que ens dona una manera eficient de buscar els strings en una llista llarga i no ordenada
* **ArrayList<ArrayList<Integer>> cjt\_historialUsuaris**
  + Estructura de dades que emmagatzema els identificadors dels resultats de les partides que han realitzat els usuaris, quan es crea un usuari aquest s'afegeix al final de la llista, hem triat una llista ja que es pot afegir una posició de manera eficient
* **ArrayList<Estadistiques> cjt\_estadistiques**
  + Estructura de dades que emmagatzema la llista d'estadístiques, els identificadors de la ArrayList son els mateixos eid de les estadístiques, quan es crea una estadística aquesta s'afegeix al final de la llista. La implementació de la llista de una manera eficient de fer aixó
* **ArrayList<Records> cjt\_records**
  + Estrucutra de dades que enmagatzema la llista de records dels diferents usuaris, els identificadors de la ArrayList son els mateixos Recid dels records, els nous records s'afegeixen al final de la llista, hem triat una llista ja que es pot afegir una posició de manera eficient

**Records:**

* **int[] records**
  + Vector de ints amb 32 posicions que emmagatzema tots els records possibles per cada usuari. Cada posició representa un record i s’utilitza un int per portar un registre del número de vegades que un usuari ha completat aquest record. Per exemple la posició 13 del vector és guanyar una partida en dificultat difícil i en mode ranked. Si un usuari no ha aconseguit això, en la posició 13 del vector hi haurà un 0, si l’ha aconseguit 4 vegades hi haurà un 4. Considerem que un vector és el més adequat per aquesta utilització, ja que només hem d’incrementar i consultar.
  + **Alternativa:** List, hem pensat també en aquesta opció, ja que podria utilitzar-se també sense cap problema, però un vector és millor, ja que sabem el número de posicions que hi haurà i els accessos són eficients.

**Historial:**

* **ArrayList<Integer> partides;**
  + Llista amb els identificadors de les partides que ha jugat un usuari, representa un historial de partides i cada usuari té el seu. Considerem que un vector és el més adequat per aquesta utilització, ja que només hem de fer operacions d’afegir i consultar.

**Controlador Partida (CP)**

En controlador partida tenim 3 hashmaps:

* **HashMap<Integer,Partida> cjt\_partides**
* **HashMap<Integer,Resultat> cjt\_resultats**
* **HashMap<Integer,Tauler> cjt\_taulers**

Cadascun guarda una clau (que és un Integer) i un objecte (que és una classe). La clau és l'identificador de l'objecte que estem guardant, i el segon paràmetre és l'objecte en sí, en aquest cas són els objectes Partida, Resultat i Tauler. Hem escollit HashMaps perquè és el mètode més eficient per a fer les operacions que es requereixen en la gestió de partides. El HashMap ens permet accedir i esborrar elements en cost O(1), i afegir-ne un en cost O(1) amortitzat, fent que l'aplicació sigui més eficient que no pas altres estructures de dades que discutirem a continuació.

**Alternatives (CP)**

- Un LinkedHashMap es podria haver fet servir, però en les operacions bàsiques és un pèl més lent que HashMap (que és la seva superclasse).

- Els Dictionary també són igual d'útils que els HashMaps, tot i que són considerats obsolets per java, i fer servir llibreries obsoletes és una gran bretxa de seguretat.

- HashTable és també una bona opció que podríem haver fet servir, ja que gairebé no hi ha diferències respecte el HashMap. Aquesta, a diferència del HashMap, permet sincronització, és adir, permet multi-threading. Com en aquest projecte no ens demanen ni especifiquen que l'aplicació sigui multi-threaded, assumirem que només s'executa en un sol thread d'un sol core, de manera que l'execució és seqüencial i no hauria de sorgir cap error degut a això.

**Conclusió (CP)**

- Específicament per al cjt\_partides, sabem que les partides s'aniran esborrant del hashmap a mida que es vagin acabant (perquè quan una partida acaba se'n guarda el seu resultat, però la partida en sí s'esborra), de manera que necesitem una estructura de dades que ens permeti accedir amb l'identificador de l'objecte, no per la posició on està guardada, que és el que ho diferencia de moltes altres estructures de dades com l'Array, ArrayList, HashSet, etc.

- Per al cjt\_taulers apliquem la mateixa lògica, ja que l'identificador d'una partida és el mateix que el del seu tauler.

- Per al cjt\_resultats, sabem que un resultat mai s'esborrarà, i que els elements es poden anar afegint de manera no ordenada. Això vol dir que estem en el mateix cas d'abans, no podem accedir als elements per la posició on es troben, sinó que necessitem tenir una clau amb l'identificador de l'objecte corresponent, per tant, per aquest cas podem aplicar la lògica explicada anteriorment.

**Població**

**private Individu[] individus:** Estructura de dades que emmagatzema elements que son Individus, es una mostra aleatories de colors que serveixen per resoldre una combinació mitjançant l’algorisme genetic. Considerem que un vector és el més adequat per aquesta utilització, ja que només hem de consultar.