MASTERMIND®

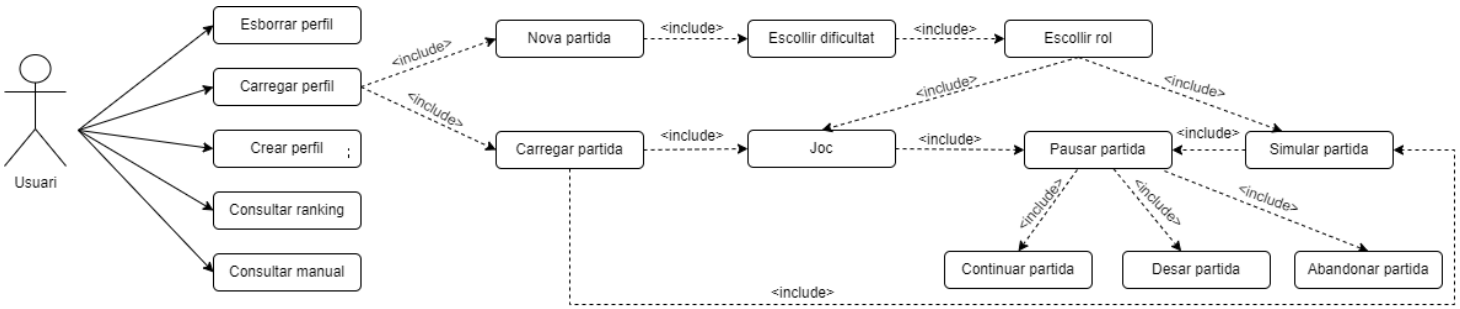
Versió 1.1

Projecte de Programació (PROP)
Identificador de l'equip: 14.1
Marc Sardà Masriera → marc.sarda.masriera@estudiantat.upc.edu
Aglaya Khalipskaya → aglaya.khalipskaya@estudiantat.upc.edu
Pol Farré Burgos → pol.farre.burgos@estudiantat.upc.edu
Pol Kallai Raventós → pol.kallai@estudiantat.upc.edu
13/04/2023

Índex

1. Diagrama de casos d'ús

El diagrama de casos d'ús pot ser trobat al directori *DOCS*.



1.1 Descripció detallada dels casos d'ús

1.1.1 Gestió d'usuari

Nom: UC001 - Crear perfil

Actor principal: Usuari

Precondició: L'usuari ha iniciat l'aplicació.

Activador: L'usuari indica al sistema que vol crear un perfil.

Escenari d'èxit principal:

1. El sistema li mostra per pantalla un formulari per introduir el nom d'usuari i la contrasenya.
2. L'usuari omple els camps i prem el botó de "Crear". Si el valor del nom d'usuari és nul o buit salta l'error 2.1, si el valor de la contrasenya és nul o buit salta l'error 2.2. Si el nom d'usuari ja està registrat al sistema, salta l'error 2.3.
3. El sistema registra el perfil i guarda per aquest, totes les dades indicades per a l'usuari.

Errors i extensions:

2.1 El sistema mostra per pantalla un error indicant que s'ha d'introduir un valor per al nom d'usuari: "Has d'introduir un nom d'usuari vàlid". Es retorna al punt 1.

2.2 El sistema mostra per pantalla un error indicant que s'ha d'introduir un valor per a la contrasenya: "Has d'introduir una contrasenya vàlida". Es retorna al punt 1.

2.3 Si el nom d'usuari indicat per usuari ja està en ús per un altre usuari, el sistema indica a l'usuari de l'error mostrant un missatge per pantalla "Ja existeix un perfil enregistrar amb aquest *username:* [valor introduït]*"*. Es retorna al punt 1.

Nom: UC002 - Carregar perfil

Actor principal: Usuari

Precondició: L'usuari ha iniciat l'aplicació.

Activador: L'usuari indica al sistema que vol carregar un perfil.

Escenari d'èxit principal:

1. El sistema mostra per pantalla una llista d'usuaris enregistrats al sistema juntament amb la seva puntuació màxima i el nombre de partides jugades. Si no hi ha cap usuari registrat al sistema, es mostra per pantalla el següent missatge: "Actualment no hi ha cap usuari registrat.".
2. L'usuari selecciona el perfil que vol carregar i introdueix la contrasenya associada a aquest.
3. Si la contrasenya és buida o nul·la, el sistema fa saltar l'error 3.1, si no, comprova que el *hash* de la contrasenya sigui el mateix. En cas de no ser-ho, salta l'error 3.2.
4. El sistema carrega el perfil seleccionat.

Errors i extensions:

3.1. El sistema informa l'usuari de l'error mostrant el següent missatge per pantalla: "Introdueix un valor vàlid per a la contrasenya". Es torna al punt 2.

3.2 El sistema informa l'usuari de l'error mostrant el següent missatge per pantalla: "La contrasenya introduïda no és correcta, torna a intentar-ho o selecciona un altre perfil d'usuari". Es torna al punt 2.

Nom: UC003 - Esborrar perfil

Actor principal: Usuari

Precondició: L'usuari ha iniciat l'aplicació.

Activador: L'usuari indica al sistema que vol esborrar un perfil.

Escenari d'èxit principal:

1. El sistema mostra per pantalla una llista d'usuaris enregistrats al sistema juntament amb la seva puntuació màxima i el nombre de partides jugades. Si no hi ha cap usuari registrat al sistema, es mostra per pantalla el següent missatge: "Actualment no hi ha cap usuari registrat.".
2. L'usuari selecciona el perfil que vol esborrar i introdueix la contrasenya associada a aquest.
3. Si la contrasenya és buida o nul·la, el sistema fa saltar l'error 3.1, si no, comprova que el *hash* de la contrasenya sigui el mateix. En cas de no ser-ho, salta l'error 3.2.
4. El sistema borra l'usuari i totes les dades associades a aquest (partides històriques incloses).

Errors i extensions:

3.1. El sistema informa l'usuari de l'error mostrant el següent missatge per pantalla: "Introdueix un valor vàlid per a la contrasenya". Es torna al punt 2.

3.2 El sistema informa l'usuari de l'error mostrant el següent missatge per pantalla: "La contrasenya introduïda no és correcta, torna a intentar-ho o selecciona un altre perfil d'usuari". Es torna al punt 2.

Nom: UC004 - Consultar rànquing

Actor principal: Usuari

Precondició: L'usuari ha iniciat l'aplicació.

Activador: L'usuari indica al sistema que vol consultar el rànquing.

Escenari d'èxit principal:

1. El sistema llista per pantalla l'històric de partides amb més puntuació de cada usuari registrat en el sistema mostrant per a cada posició d'aquest, el nom d'usuari del perfil, la seva puntuació i el nombre de partides jugades. Si no hi ha cap partida salta l'error 1.1.

Errors i extensions:

1.1 El sistema mostra un missatge per pantalla "Actualment no hi ha cap partida jugada.".

Nom: UC005 - Consultar manual

Actor principal: Usuari

Precondició: L'usuari ha iniciat l'aplicació.

Activador: L'usuari indica al sistema que vol consultar el manual del joc.

Escenari d'èxit principal:

1. El sistema mostra a l'usuari el manual del joc.

Errors i extensions:

1.1.2 Gestió de partides:

Nom: UC010 - Carregar partida

Actor principal: Usuari

Precondició: L'usuari ha hagut de carregar el seu perfil d'usuari.

Activador: L'usuari indica al sistema que vol carregar una partida.

Escenari d'èxit principal:

1. El sistema mostra per pantalla una llista amb les partides guardades anteriorment per aquest perfil d'usuari. Si no n'hi ha cap, salta l'error 1.1.
2. L'usuari selecciona una partida.
3. El sistema carrega la partida i l'usuari pot continuar el joc.

Errors i extensions:

1.1 El sistema informa l'usuari de l'error mostrant per pantalla el següent missatge: "No existeix cap partida pendent de ser acabada per a aquest usuari". Torna al punt 1.

Nom: UC011 - Nova partida

Actor principal: Usuari

Precondició: L'usuari ha hagut de carregar el seu perfil d'usuari.

Activador: L'usuari indica al sistema que vol crear una nova partida.

Escenari d'èxit principal:

1. El sistema inicia el procés de configuració de partida i mostra per pantalla les diferents opcions a configurar.

    a. El sistema mostra per pantalla l'opció de configurar la dificultat de la partida. Dificultat → Fàcil, Intermig, Difícil.

        i. L'usuari escull una d'aquestes opcions.

        ii. El sistema comprova que l'opció escollida és correcta i guarda la configuració en qüestió. Si l'opció indicada per a l'usuari no és vàlida, salta l'error 1.a.ii.1.

    b. El sistema mostra per pantalla l'opció de configurar el rol que tindrà l'usuari a la partida. Rol jugador → Codebreaker, Codemaker.

        i. L'usuari escull una d'aquestes dues opcions.

        ii. El sistema comprova que l'opció escollida és correcta i guarda la configuració. Si l'opció indicada per a l'usuari no és vàlida, salta l'error 1.b.ii.1.

2. El sistema genera, càrrega i inicia la partida.

Errors i extensions:

1.a.ii.1. El sistema informa a l'usuari de l'error mostrant per pantalla el següent missatge: "La dificultat escollida no está disponible.
Si us plau, escull una de les opcions presentades.". Es torna al punt a.

1.b.ii.1. El sistema informa a l'usuari de l'error mostrant per pantalla el següent missatge: "El rol escollit no está disponible.
Si us plau, escull una de les opcions presentades.". Es torna al punt b.

Nom: UC012 - Escollir dificultat

Actor principal: Usuari

Precondició: L'usuari ha hagut de crear una nova partida.

Activador: L'usuari indica al sistema que vol escollir una dificultat.

Escenari d'èxit principal:

1. El sistema indica a l'usuari les diferents opcions de dificultat.
2. L'usuari escull una dificultat.
3. El sistema registra la dificultat de la partida escollida per a l'usuari en el punt 2 i mostra les opcions de rol.

Errors i extensions:

Nom: UC013 - Escollir rol

Actor principal: Usuari

Precondició: L'usuari ha hagut d'haver escollit la dificultat de la partida.

Activador: L'usuari indica al sistema que vol escollir un rol.

Escenari d'èxit principal:

1. El sistema indica a l'usuari que pot escollir entre *codemaker* i *codebraker.*
2. L'usuari escull un dels rols indicats pel sistema.
3. El sistema registra el rol que ocuparà l'usuari en la partida.

Errors i extensions:

Nom: UC014 - Escollir algorisme i combinació

Actor principal: Usuari

Precondició: L'usuari ha hagut d'haver escollit el rol de *codemaker*

Activador: L'usuari ha escollit el rol de *codemaker*.

Escenari d'èxit principal:

1. El sistema demana a l'usuari que introdueixi la combinació de joc seguint els criteris de la dificultat escollida anteriorment.
2. L'usuari indica al sistema la combinació de joc.
3. El sistema valida que la combinació indicada no violi cap norma segons la dificultat establerta, si ho fa salta l'error. 3.2.
4. El sistema mostra a l'usuari els diferents algoritmes de joc a escollir.
5. L'usuari indica al sistema quin algorisme desitja per a la simulació de la partida.
6. El sistema inicia la simulació de la partida amb configuració establerta.

Errors i extensions:

3.2. Si la combinació de colors viola alguna norma segons la dificultat escollida anteriorment, mostra per pantalla el següent missatge, "No és possible jugar amb aquesta combinació de colors. Si us plau, torna-la a introduir.". Es torna al punt 2.

1.1.3 Gestió de joc:

Nom: UC020 - Joc

Actor principal: Usuari

Precondició: L'usuari ha hagut d'haver carregat o creat una partida.

Activador: El sistema inicia la partida.

Escenari d'èxit principal:

1. El sistema mostra per pantalla la informació de la partida (número de torn, puntuació i la dificultat escollida).

2. El sistema demana a l'usuari que aquest indiqui la combinació de colors que desitgi segons els colors que pot escollir depenent de la dificultat de la partida.

3. L'usuari indica la combinació de colors desitjada.

4. El sistema comprova que la combinació de colors indicada per a l'usuari és vàlida és a dir, que aquesta correspon a la dificultat de la partida. Si no correspon, salta l'error 4.1.

5. Si el rol de l'usuari en la partida (escollit en crear la partida) és codebreaker:

    a. El sistema compara la combinació de colors indicada per l'usuari amb la combinació que ha creat el codemaker.

    b. El sistema mostra per pantalla la informació de la partida i per cada combinació que ha indicat l'usuari en cada torn d'aquesta, mostra la combinació i el nombre de colors que estan bé i els que no en comparació amb la combinació a desxifrar.

    c. Els punts 2, 3, 4, 5.a i 5.b es repeteixen fins que acaba la partida, o bé perquè l'usuari ha desxifrat la combinació, o bé perquè s'ha arribat al nombre de torns màxim que depèn de la dificultat de la partida.

    d. Quan la partida finalitza o bé perquè s'ha arribat al màxim de torns d'aquesta (determinat per la dificultat de la partida) o bé perquè l'usuari ha endevinat la combinació de colors, el sistema informa l'usuari de què la partida ha acabat mostrant un missatge per pantalla. El sistema a més, també indica a l'usuari si aquest ha guanyat (si ha endevinat la combinació de colors abans d'arribar al màxim de torns de la partida) o en cas contrari, ha perdut.

6. Si el rol de l'usuari en la partida (escollit en crear la partida) és codemaker:

a. El sistema executa l'algorisme de resolució de la combinació de colors.

b. El sistema mostra per pantalla la informació (combinació indicada pel codebreaker, nombre de fallades i nombre de colors correctes) de tots els torns que s'han dut a terme per a la resolució de la combinació.

c. El sistema mostra per pantalla un missatge indicant a l'usuari si aquest ha guanyat o ha perdut la partida, en funció de si la màquina ha endevinat o no la combinació de colors establerta per a l'usuari a l'inici de la partida.

Errors i extensions:

4.1 Si la combinació de colors viola alguna norma segons la dificultat escollida anteriorment, mostra per pantalla el següent missatge, "Aquest color no és vàlid segons la dificultat de la partida". Es torna al punt 3.

Nom: UC021 - Pausar partida

Actor principal: Usuari

Precondició: La partida està en joc, iniciada.

Activador: L'usuari indica al sistema que vol pausar la partida.

Escenari d'èxit principal:

1. El sistema pausa la partida.
2. El sistema mostra per pantalla el menú de pausa i totes les opcions que l'usuari pot escollir (abandonar partida, continuar partida i desar partida).

Errors i extensions:

Nom: UC022 - Abandonar partida

Actor principal: Usuari

Precondició: El sistema ha aturat la partida per petició de l'usuari.

Activador: L'usuari indica al sistema que vol abandonar la partida.

Escenari d'èxit principal:

1. El sistema tanca la partida que s'estava executant i retorna a l'usuari a la pantalla principal.

Errors i extensions:

Nom: UC023 - Continuar partida

Actor principal: Usuari

Precondició: El sistema ha aturat la partida per petició de l'usuari.

Activador: L'usuari indica al sistema que vol continuar la partida.

Escenari d'èxit principal:

1. El sistema continua la partida en el punt on es va quedar abans de ser aturada mostrant per pantalla la informació d'aquesta i els torns que s'han fet.

Errors i extensions:

Nom: UC024 - Desar partida

Actor principal: Usuari

Precondició: El sistema ha aturat la partida per petició de l'usuari.

Activador: L'usuari indica al sistema que vol desar la partida.

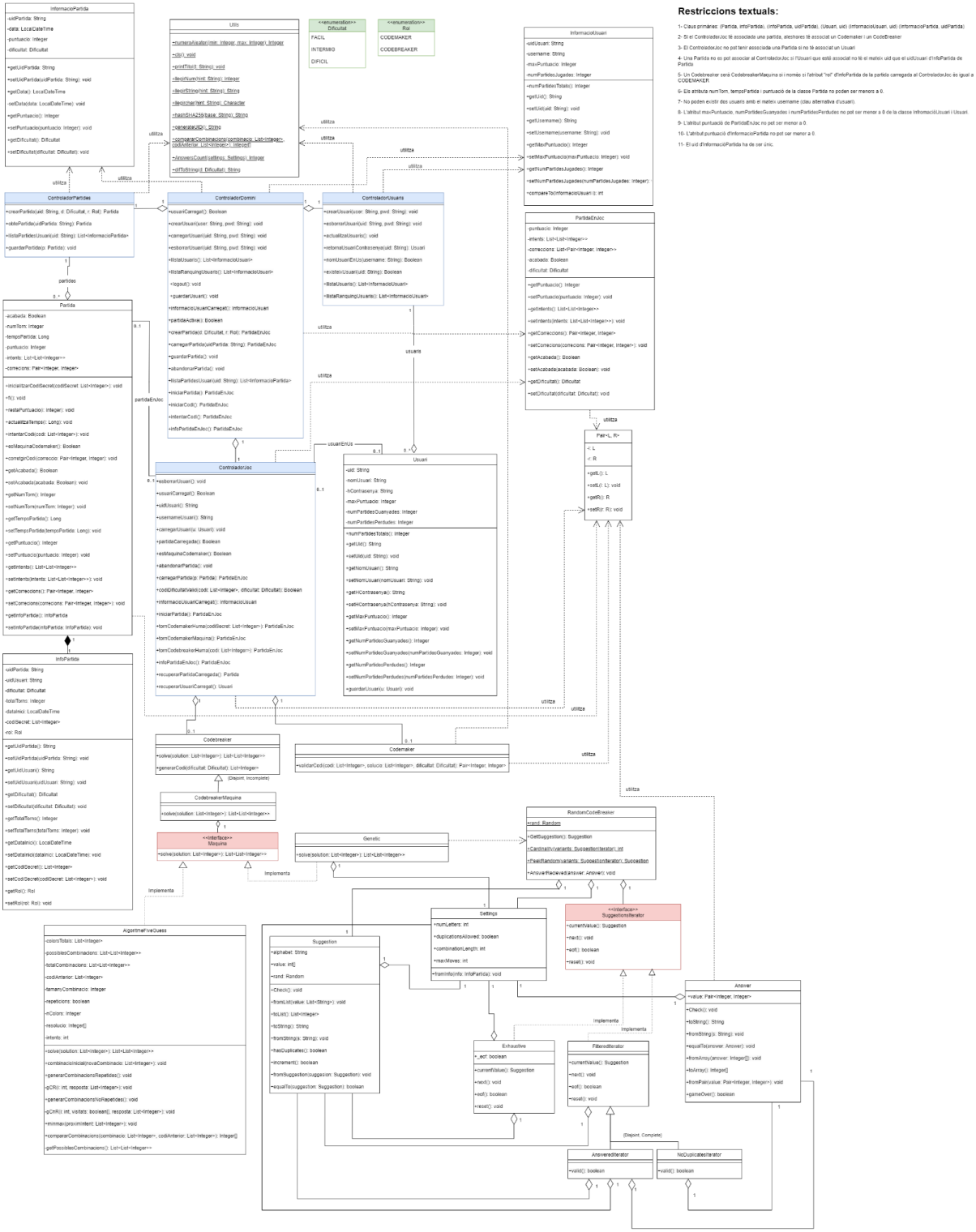
Escenari d'èxit principal:

1. El sistema guarda les dades de la partida actual.

Errors i extensions:

2. Diagrama del model conceptual

Podeu trobar aquest diagrama al directori *DOCS* de l'entrega. En color cian es marquen els controladors, en color verd els *enums* i en color rosat les interfícies.



2.1 Descripció detallada de les classes

2.1.1 Utils

Nom de la classe	Utils
Breu descripció de la classe	Aquesta classe conté mètodes estàtics que hem considerat útils i utilitzem en diferents funcions d'altres classes del nostre projecte per a efectuar diverses operacions.
Descripció dels atributs	Aquesta classe no té cap atribut definit.
Descripció dels mètodes	**numeroAleatori(Integer min, Integer max):** Genera un nombre enter aleatori entre els dos valors, inclusius, enters *min* i *max* passats per paràmetre. Retorna l'enter generat que està dins d'aquest rang. **cls():** Imprimeix 20 salts de línia a la terminal de l'usuari, creant un efecte de netejar la terminal en moure el contingut mostrat anteriorment fora de la vista de l'usuari. **printTitol(String t):** Imprimeix un títol de manera decorativa per la terminal. Agafa el paràmetre de tipus String *t* que representa el text del títol i l'imprimeix rodejat per un separador de guions per tal de crear un efecte visual de títol per la terminal de l'usuari. **llegirNum(String hint):** Llegeix una cadena de text introduïda per a l'usuari per la terminal, la converteix a enter i la retorna. **llegirString(String hint):** Llegeix una cadena de text de la terminal i la retorna en una variable String. A més a més, valida que la cadena introduïda per l'usuari per a la terminal tingui com a mínim 3 caràcters abans de retornar-la. **llegirchar(String hint):** Llegeix un caràcter de la terminal i el retorna. **hashSHA256(String base):** Crea un hash SHA-256 a partir

	del string *base* passat com a paràmetre. Utilitza la classe MessageDigest de Java per a realitzar la funció de hash i retorna el resultat com a string. **generateUID():** Genera un identificador únic universal (UUID) i el retorna com una cadena de text (String) i utilitza la classe UUID de Java per a generar-lo. **compararCombinacions(List<Integer> combinacio, List<Integer> codiAnterior):** Valida una combinació comparant-la amb una segona segons la posició dels paràmetres que les formen. Rep dos paràmetres, *combinacio* que és la combinació que volem avaluar i *codiAnterior* que correspon a la segona combinació amb la qual volem avaluar l'altra. Retorna el resultat de la comparació on el primer component són les posicions correctes i el segon, les incorrectes. **difToString(Dificultat d):** Converteix l'enum de dificultat d rebut com a paràmetre a una variable de tipus String que retorna. **AnswersCounts(Settings settings):** Calcula el nombre de respostes possibles per a desxifrar el codi. Reb com a paràmetre *settings* la configuració de la partida i retorna el nombre de possibles respostes per a desxifrar el codi.

2.1.2 Usuari

Nom de la classe	Usuari
Breu descripció de la classe	La classe Usuari representa un usuari del joc i, per tant, conté totes les dades que formen un perfil d'usuari.
Descripció dels atributs	**uid:** identificador únic de l'usuari. **nomUsuari:** nom de l'usuari. **hContrasenya:** hash de la contrasenya de l'usuari. **maxPuntuacio:** màxima puntuació històrica de l'usuari.

	numPartidesGuanyades: nombre de partides guanyades per l'usuari. **numPartidesPerdudes:** nombre de partides perdudes per l'usuari.
Descripció dels mètodes	**Usuari():** Constructor per defecte de la classe. **Usuari(String nomUsuari, String hContrasenya):** Constructor de la classe Usuari que reb com a paràmetres el nom d'usuari i la contrasenya de l'usuari per a crear un nou objecte Usuari amb els valors assignats als atributs corresponents. **numPartidesTotals():** Calcula i retorna el nombre total de partides jugades per l'usuari (guanyades + perdudes). **actualitzaEstadistiques(Boolean guanyat, Integer puntuacio):** Actualitza el nombre de partides guanyades/perdudes i la màxima puntuació, d'acord amb els valors passats com a paràmetres.

2.1.3 ControladorUsuaris

Nom de la classe	ControladorUsuaris
Breu descripció de la classe	El ControladorUsuaris fa una funció similar al ControladorPartides, però, com el seu nom indica, amb usuaris. S'encarrega de gestionar tots aquests, continguts en un mapa, i de forma similar a l'anterior controlador, proporciona un seguit de mètodes i funcions perquè el ControladorDomini pugui interactuar amb ells.
Descripció dels atributs	**usuaris:** HashMap<K,V> que emmagatzema usuaris (V) usant com a clau (K) el seu atribut uid.

Descripció dels mètodes	**crearUsuari(String user, String pwd):** Crea un nou usuari i el guarda tant a *usuaris* (l'afegeix) com al disc mitjançant la capa de persistència. **esborrarUsuari(String uid, String pwd):** Esborra un usuari d'*usuaris* i de disc. **actualitzaUsuaris():** Actualitza l'atribut usuari amb la informació guardada a disc que retorna la capa de persistència.

2.1.4 ControladorJoc

Nom de la classe	ControladorJoc
Breu descripció de la classe	El ControladorJoc és l'encarregat de dur a terme la gestió de la partida que s'estigui jugant. Aquest controla un usuari i una partida i proporciona al ControladorDomini diferents mètodes perquè l'usuari pugui interactuar amb la partida i aquesta pugui també ser jugada per un dels algorismes mitjançant les instàncies de *codebreaker* i *codemaker* que aquest controla.
Descripció dels atributs	**codemaker:** Jugador encarregat de crear el codi secret en una partida. **codebreaker:** Jugador encarregat d'intentar endevinar el codi secret en una partida. **partidaCarregada:** Instància de la partida que s'està jugant actualment. **usuariCarregat:** Instància de l'usuari carregat que està jugant la partida en aquest moment.
Descripció dels mètodes	**esborrarUsuari():** Invalida l'usuari carregat, posant-lo a *null*. **usuariCarregat():** Comprova si hi ha un usuari carregat i retorna *true* si n'hi ha un o *false* en cas contrari. **uidUsuari():** Obté l'identificador de l'usuari carregat i llença l'excepció ExcepcioUsuari si no hi ha cap usuari carregat.

	usernameUsuari(): Obté el *username* de l'usuar carregat i llença l'excepció ExcepcioUsuari en el cas de que no hi hagi cap usuari carregat. **carregarUsuari(Usuari usuari):** Carrega un usuari mitjançant l'objecte de la classe Usuari passat com a paràmetre. **partidaCarregada():** Comprova si hi ha una partida carregada, retornant *true* si n'hi ha una o *false* en cas contrari. **esMaquinaCodemaker():** Comprova si la màquina té el rol de *CODEMAKER* retornant *true* si és el cas o *false* altrament. **abandonarPartida():** Abandona la partida carregada, invalidant-la i invalidant els jugadors. **carregarPartida(Partida p):** Carrega una partida i instància els jugadors segons el rol escollit i llença l'excepció ExcepcioPartida en el cas que no hi hagi cap partida carregada. **codiDificultatValid(codi: List<Integer> codi, Dificultat dificultat):** Comprova si el codi passat com a paràmetre *codi* compleix la dificultat de la partida, retorna *true* si compleix les normes de la dificultat o *false* en cas contrari. **informacioUsuariCarregat():** Obté la informació de l'usuari carregat i la retorna. Si no hi ha cap usuari carregat, fa saltar l'excepció ExcepcioUsuari. **iniciarPartida():** Comprova els paràmetres de la partida carregada i inicia la partida. En el cas que no hi hagi cap partida carregada, llença l'excepció ExcepcioPartida. A més a més, retorna un objecte del tipus PartidaEnJoc que conté la informació de la partida en joc. **tornCodemakerHuma(codiSecret: List<Integer>):** Comprova els paràmetres i inicialitza el codi secret proporcionat per l'usuari en el cas que aquest tingui el rol de *codemaker*. Si no és el cas o no hi ha cap partida carregada

	o el codiSecret passat com a paràmetre no és vàlid o no correspon a la dificultat de la partida, llença l'excepció ExcepcioPartida. A més a més, retorna un objecte del tipus PartidaEnJoc que conté la informació de la partida en joc. **tornCodemakerMaquina():** En aquest cas, la màquina té el rol de *codemaker* en la partida carregada i aquest mètode inicialitza el codi secret proporcionat per la màquina (generat aleatòriament) en la partida en joc. En el cas que no hi hagi cap partida carregada, llença l'excepció ExcepcioPartida. A més a més, aquest mètode retorna un objecte del tipus PartidaEnJoc que conté la informació de la partida en joc. **tornCodebreakerHuma(codi: List<Integer>):** Comprova els paràmetres i realitza un intent per a desxifrar el codi secret per part de l'usuari passat com a paràmetre codi. Si no hi ha cap partida carregada, el jugador no té el rol de *codebreaker* en la partida carregada o el codi passat com a paràmetre no és vàlid, llença l'exepció ExcepcioPartida. A més a més, retorna un objecte del tipus PartidaEnJoc que conté la informació de la partida en joc. **infoPartidaEnJoc():** Comprova si la partida actual ha finalitzat i obté la informació de la partida de joc actual i la retorna en forma d'un objecte de la classe PartidaEnJoc amb les dades de la partida, l'usuari carregar, el codemaker i el codebreaker. En el cas que no hi hagi cap partida carregada, llença l'excepció ExcepcioPartida**.** **recuperarPartidaCarregada():** Obté la informació de la partida que es troba carregada i la retorna en un objecte de tipus Partida. Si no hi ha cap partida carregada, llença l'excepció ExcepcioPartida. **recuperarUsuariCarregat():** Obté la informació del perfil de l'usuari carregat i la retorna en un objecte de tipus Usuari. En el cas que no hi hagi cap usuari carregat, llença l'excepció ExcepcioUsuari.

2.1.5 ControladorDomini

Nom de la classe	ControladorDomini
Breu descripció de la classe	El ControladorDomini és el cervell de capa de domini, ja que és l'encarregat de comunicar-se amb la capa de presentació. Disposa de molts mètodes per tal de realitzar la comunicació entre l'usuari i l'aplicació, però no calcula res per ell mateix sinó que crida a altres mètodes de controladors especialitzats, realitzant una funció d'encaminador de peticions.
Descripció dels atributs	**controladorPartides:** Instància de la classe ControladorPartides que es fa servir per gestionar les partides del joc. **controladorUsuaris:** Instància de la classe ControladorUsuaris que es fa servir per gestionar els usuaris del joc. **controladorJoc:** Instància de la classe ControladorJoc que es fa servir per gestionar la lògica del joc.té una relació d'agregació amb les c
Descripció dels mètodes	**usuariCarregat():** Comprova si hi ha un usuari carregat. Retorna cert si hi ha un usuari carregat o fals si no. **crearUsuari(String username, String pwd):** Crea un usuari amb el nom d'usuari i contrasenya especificats com a paràmetres. Llança una excepció "ExcepcioUsuari" si hi ha un error en la creació de l'usuari **carregarUsuari(String uid, String pwd):** Carrega un usuari amb l'identificador d'usuari i contrasenya especificats com a paràmetres. Llança una excepció "ExcepcioUsuari" si hi ha un error en la càrrega de l'usuari.

	esborrarUsuari(String uid, String pwd): Esborra un usuari amb l'identificador d'usuari i contrasenya especificats com a paràmetres. Llança una excepció "ExcepcioUsuari" si hi ha un error en l'esborrat de l'usuari. **llistaUsuaris():** Obté i retorna una llista amb informació de tots els usuaris, incloent-hi l'identificador d'usuari, nom d'usuari i puntuació màxima. Llança una excepció "ExcepcioUsuari" si hi ha un error en l'obtenció de la llista d'usuaris. **llistaRanquingUsuaris():** Obté i retorna una llista d'informació d'usuaris ordenada per la puntuació màxima. Llança una excepció "ExcepcioUsuari" si hi ha un error en l'obtenció de la llista d'usuaris. **logout():** Esborra l'usuari carregat, tancant la sessió. **guardarUsuari():** Guarda l'usuari carregat en memòria i persistència. Llança una excepció "ExcepcioUsuari" si hi ha un error en la desada de l'usuari. **informacioUsuariCarregat():** Obté informació de l'usuari carregat, incloent-hi el seu identificador d'usuari, nom d'usuari i puntuació màxima. Llança una excepció ExcepcioUsuari si no hi ha cap usuari carregat. **partidaActiva():** Comprova si hi ha una partida carregada i retorna cert en cas que sí que n'hi hagi una o fals en cas contrari. **crearPartida(Dificultat d, Rol r):** Comprova que els paràmetres passats són correctes i crea la partida configurant-la segons la dificultat i el rol de l'usuari indicats. A més a més, fa saltar l'excepció ExcepcioUsuari si no hi ha cap usuari carregat en aquest moment i si no salta, retorna la indormació de l'estat de la partida en un atribut de tipus PartidaEnJoc.

	carregarPartida(String uidPartida): Carrega una partida amb l'identificador de partida especificat. Llança una excepció ExcepcioPartida si no es pot carregar la partida. **guardarPartida():** Desa l'estat de la partida carregada en memòria i persistència. Llança una excepció ExcepcioPartida si hi ha algun error en l'operació de desament de la partida. **abandonarPartida():** Tanca la partida carregada, eliminant-la de la memòria. Llança una excepció ExcepcioPartida si no hi ha cap partida carregada. **llistaPartidesUsuari(String uid):** Retorna una llista que cada posició d'aquesta correspon a un atribut del tipus InformacioPartida que conté la informació de les partides no acabades d'un usuari. En concret, l'usuari amb uid passat com a paràmetre. A més a més, fa saltar l'excepció ExcepcioUsuari si l'identificador d'usuari passat com a paràmetre *uid* no és vàlid o l'usuari amb aquest *uid* no existeix. **iniciarPartida():** Inicia la partida carregada i retorna la informació de l'estat d'aquesta en un atribut del tipus PartidaEnJoc. **iniciarCodi(List<Integer> codiSecret):** Inicialitza el codi secret de la partida amb el codi passat com a paràmetre. A més a més, retorna la informació de l'estat de la partida en un atribut del tipus PartidaEnJoc. **intentarCodi(List<integer> codi):** Realitza un intent de trobar el codi secret de la partida amb el codi passat com a paràmetre. A més a més, retorna la informació de l'estat de la partida en un atribut del tipus PartidaEnJoc. **infoPartidaEnJoc():** Retorna la informació de l'estat de la partida en joc en un atribut del tipus PartidaEnJoc.

2.1.6 Partida

Nom de la classe	Partida
Breu descripció de la classe	Aquesta classe permet gestionar i mantenir l'estat d'una partida, així com efectuar les operacions necessàries com inicialitzar el codi secret, gestionar els intents, correccions i puntuació de la partida. També proporciona mètodes per obtenir i establir els atributs de la partida com ara l'estat, el número de torn, la informació de partida, el temps de partida, la puntuació, els intents i les correccions.
Descripció dels atributs	**acabada:** Booleà que indica si la partida ha acabat o no. **numTorn:** Enter que representa el número de torn actual de la partida. **infoPartida:** Objecte de la classe *InfoPartida* que conté la informació de la partida com a l'identificador d'usuari que la juga, l'identificador de la partida, la dificultat i el rol de cada jugador. **tempsPartida:** Un long que emmagatzema el temps transcorregut de la partida des del seu inici, en milisegons. **puntuacio:** Enter que representa la puntuació de la partida de l'usuari. **intents:** Llista de llistes d'enters que emmagatzema els intents realitzats pel jugador amb rol *codebreaker*. **correccions:** Llista de parells d'enters que emmagatzema les correccions dels intents realitzats pel jugador amb rol *codebreaker*.
Descripció dels mètodes	**Partida(String uidPartida, String uidUsuari, Dificultat dificultat, Rol rol):** Constructor de la classe que rep com a paràmetres l'identificador de partida, l'identificador d'usuari, la dificultat i el rol del jugador, i inicialitza els atributs de la classe com ara l'estat de la partida, el número de torn, la informació de partida, la puntuació i el temps de partida.

	inicialitzarCodiSecret(List<Integer> codiSecret): Inicialitza el codi secret de la partida amb una llista d'enters passada com a paràmetre. **incrementaTorn():** Incrementa en 1 el número de torn de la partida. **restaPuntuacio(Integer i):** Resta un valor enter passat com a paràmetre a la puntuació de la partida. **fi():** Estableix l'estat de la partida com a acabada. **actualitzaTemps(Long l):** Actualitza el temps de partida amb el valor passat com a paràmetre en mil·lisegons. **intentarCodi(List<Integer> codi):** Afegeix un intent realitzat pel jugador a la llista d'intents de la partida. **esMaquinaCodemaker():** Retorna un booleà que indica si el rol del jugador és *codebreaker* o no. **corretgirCodi(Pair<Integer, Integer> correccio):** Afegeix una correcció a la llista de correccions de la partida.

2.1.7 InfoPartida

Nom de la classe	InfoPartida
Breu descripció de la classe	La classe InfoPartida representa els valors constants d'una partida com poden ser la dificultat d'aquesta, el rol que pren l'usuari, la data de creació de la partida, el nombre màxim de torns, els identificadors tant de l'usuari com de la partida o el codi secret escollit pel *codemaker*.
Descripció dels atributs	**uidPartida:** Cadena de caràcters (String) que representa l'identificador únic de la partida. **uidUsuari:** String que representa l'identificador únic de l'usuari que ha iniciat la partida. **dificultat:** Objecte de la classe Dificultat que representa la dificultat de la partida.

	totalTorns: Enter que representa el nombre total de torns de la partida. **dataInici:** Objecte de la classe LocalDateTime que representa la data i hora d'inici de la partida. **codiSecret:** Llista d'enters que representa el codi secret del joc. **rol:** Objecte de la classe Rol que representa el rol de l'usuari en la partida.
Descripció dels mètodes	**InfoPartida(String uidUsuari, String uidPartida, Dificultat dificultat, Rol rol):** Constructor de la classe que rep com a paràmetres els identificadors d'usuari i partida, la dificultat i el rol de l'usuari. Aquest constructor inicialitza els atributs de la classe, com ara el total de torns, en funció de la dificultat especificada.

2.1.8 ControladorPartides

Nom de la classe	ControladorPartides
Breu descripció de la classe	El ControladorPartides és l'encarregat de gestionar les partides existents, conté totes les instàncies d'aquestes en un mapa i proporciona informació d'aquestes mitjançant mètodes usats pel ControladorDomini.
Descripció dels atributs	**Partides:** Atribut privat de tipus *Map* que emmagatzema les partides del joc. La clau del map és l'identificador de la partida (de tipus String), i el valor és una instància de la classe *Partida*. Aquest atribut s'utilitza per emmagatzemar i gestionar les partides creades pel controlador.
Descripció dels mètodes	**crearPartida(String uid, Rol r, Dificultat d):** Crea una nova partida amb els paràmetres d'entrada (identificador del jugador *uid*, rol del jugador *r* i dificultat de la partida *d*). Genera un identificador únic per la partida utilitzant el mètode

	Utils.generateUID(), crea una nova instància de la classe *Partida* amb aquest identificador i els altres paràmetres, i l'afegeix a l'atribut *partides* utilitzant l'identificador de la partida com a clau. Finalment, retorna la partida creada. **llistaPartidesUsuari(String uid):** Retorna una llista d'objectes de la classe *InformacioPartida* amb la informació de les partides no acabades d'un usuari amb l'identificador d'entrada. Recorre totes les entrades de l'atribut *partides* i compara l'identificador de l'usuari de cada partida amb l'identificador d'entrada. Si coincideixen, afegeix una nova instància de la classe *InformacioPartida* amb la informació de la partida a la llista. Si no es troben partides pendents d'aquest usuari, llança una excepció de la classe *ExcepcioPartida*. **guardaPartida(Partida p):** Guarda la partida *p* passada com a paràmetre a l'atribut *partides*.

2.1.9 Codemaker

Nom de la classe	Codemaker
Breu descripció de la classe	La classe Codemaker representa el jugador encarregar de configurar el codi secret és a dir, el jugador que té el rol de *codemaker* en la partida.
Descripció dels atributs	Aquesta classe no té cap atribut declarat explícitament.
Descripció dels mètodes	**generarCodi(Dificultat dificultat):** Genera un codi secret aleatòriament d'acord amb la dificultat de la partida. Rep un paràmetre de tipus *Dificultat* que indica el nivell de dificultat de la partida. El mètode utilitza la classe *Utils* per generar nombres aleatoris i crear una llista d'enters amb 4 nombres

	aleatoris, basant-se en la dificultat especificada. Retorna la llista de nombres generada com a codi secret. **validarCodi(List<Integer> codi, List<Integer> solucio):** Aquest mètode valida el codi introduït pel jugador amb rol *codebreaker* comparant-lo amb el codi secret del codificador (*codemaker)*. Rep dues llistes d'enters com a paràmetres, *codi* que representa el codi introduït pel jugador *codebreaker* i *solucio* que representa el codi secret del codificador *codemaker*. Utilitza la classe *Utils* per comparar les dues combinacions de nombres i determinar el nombre de punts blancs i negres que s'han de retornar com a resultat. Si el codi introduït no té la longitud esperada, llança una excepció *ExcepcioPartida*. Retorna un objecte de tipus *Pair<Integer, Integer>* amb els punts blancs i negres com a components *L* i *R* respectivament.

2.1.10 Codebreaker

Nom de la classe	Codebreaker
Breu descripció de la classe	La classe Codebreaker representa el jugador encarregat de desxifrar el codi secret. És la superclasse de les classes CodebreakerHuma i CodebreakerMaquina que hereten un mètode per resoldre les combinacions secretes.
Descripció dels atributs	Aquesta classe no té cap atribut.
Descripció dels mètodes	**solve(List<Integer> solution):** Obté les combinacions utilitzades per descobrir el codi secret utilitzant un algoritme específic. Rep una llista d'enters com a paràmetre, *solution*, que representa el codi secret a descobrir. Aquest mètode té com a finalitat ser sobreescrit per altres classes filles, com per exemple *CodebreakerMaquina*, ja que es declara com a

	"null" i no té cap implementació en la classe "Codebreaker" en si. Aquest mètode pot llançar una excepció "Exception" en cas que es produeixi algun error durant la seva execució.

2.1.11 CodebreakerMaquina

Nom de la classe	CodebreakerMaquina
Breu descripció de la classe	Classe que hereta de la classe *Codebreaker* i representa el jugador màquina amb rol *codebreaker* que utilitza el mètode solve d'un algorisme de resolució per intentar endevinar el codi secret creat per l'usuari que juga la partida amb rol *codemaker*.
Descripció dels atributs	**algoritme:** Atribut privat de tipus *Maquina* que representa l'algoritme utilitzat per a la resolució automàtica del codi.
Descripció dels mètodes	**solve(List<Integer> solution):** Aquest mètode sobreescriu el mètode *solve* de la classe mare *Codebreaker* i s'utilitza per a obtenir les combinacions utilitzades per a descobrir el codi secret mitjançant l'algoritme específic assignat a l'atribut *algoritme* de la classe. Rep una llista d'enters com a paràmetre, *solution*, que representa el codi secret a descobrir. A més a més, el mètode pot llançar una excepció de tipus *Exception* en el cas que es produeixi algun error durant la seva execució.

2.1.12 Interfície Maquina

Nom de la interfície	Maquina

Breu descripció de la interfície	Interfície proporcionada pel professorat de PROP que declara un sol mètode. Representa una màquina que pot resoldre la combinació de colors creada per al jugador de la partida amb rol *codemaker*. La interfície defineix una operació *solve* que pren una llista de codis com a paràmetre i retorna una llista de llistes d'enters.
Descripció dels atributs	La interfície no té cap atribut.
Descripció dels mètodes	**solve(List<Integer> solution):** Utilitza un dels algoritmes proposats (sigui l'algoritme genètic o el five guess) per crear una llista de codis que portin a la solució. Si l'algoritme no pot trobar la solució en menys de *maxSteps* passos, la llista retornada contindrà una llista de codis amb *maxSteps* elements. L'operació pot llençar una excepció si la solució del codi secret no és consistent amb els paràmetres del joc.

2.1.13 Genetic

Nom de la classe	Genetic
Breu descripció de la classe	Aquesta classe implementa la interfície *Maquina*. És una classe final, la qual cosa significa que no pot ser heretada per altres classes. Està dissenyada per a resoldre una partida utilitzant l'algorisme genètic plantejat.
Descripció dels atributs	**settings:** És un objecte de la classe *Settings* que emmagatzema les configuracions per la partida. Aquest atribut és públic i, per tant, és accessible des de fora de la classe.
Descripció dels mètodes	**Genetic(InfoPartida info):** Constructor de la classe que rep un objecte *info* de la classe *InfoPartida* com a paràmetre. Aquest constructor crea una nova instància de la classe

	Settings i l'assigna a l'atribut *settings* de la classe. A part, crida al mètode *fromInfo()* de l'objecte *settings* per assignar-li les configuracions des de l'objecte *info*. **solve(List<Integer> solution):** És l'algorisme principal per a resoldre la partida. Rep una llista d'enters com a paràmetre *solution* i retorna una llista de llistes d'enters. Utilitza un bucle *do-while* per realitzar iteracions i resoldre la partida. Dins del bucle, crida a altres mètodes com *GetSuggestion()* i *AnswerRecieved()* de les classes *RandomCodeBreaker* i *Answer* respectivament, per obtenir suggeriments i processar les respostes. La llista de suggeriments és afegida a la llista *list* i és retornada al final del mètode.

2.1.14 AlgortimeFiveGuess

Nom de la classe	AlgortimeFiveGuess
Breu descripció de la classe	La classe AlgortimeFiveGuess és un dels algoritmes utilitzats per a resoldre el joc, aquesta endevina una combinació secreta a partir de provar i avaluar combinacions segons uns paràmetres, la seva funció principal és retornar totes les combinacions utilitzades fins a arribar a la solució (en els menys intents possibles).
Descripció dels atributs	**colorsTotals:** És una llista que emmagatzema els colors disponibles per generar combinacions. Conté els números del 0 al 5, que representen la totalitat de colors possibles amb els quals es pot jugar. **possiblesCombinacions:** És una llista que emmagatzema totes les possibles combinacions que podrien ser la solució del joc. S'utilitza per generar totes les combinacions possibles combinacions amb repetició o sense, i es va modificant a partir que es descarten combinacions.

	totalCombinacions: És una llista que emmagatzema totes les combinacions que s'han generat i s'han provat, el resultat final són tots els intents fins a arribar a la solució. **codiAnterior:** És una llista que emmagatzema el codi utilitzat anteriorment per intentar resoldre el joc. Es fa servir per comparar les possibles combinacions generades amb aquesta i per saber la resolució de colors correctes i incorrectes. **tamanyCombinació:** És enter que indica la mida de la combinació. **repeticions**: És una variable booleana que indica si hi ha o no repeticions de colors a la combinació. **nColors:** És un enter que indica el nombre de possibles colors que hi ha a la combinació i que podem utilitzar per resoldre-la. **resolucio:** És un array de dos enters que indica les posicions correctes i incorrectes de l'intent anterior. **intents**: Guarda el nombre total d'intents que pot emprar.
Descripció dels mètodes	**AlgoritmeFiveGuess (InfoPartida ip)**: Creadora per defecte de la classe. **solve(List<Integer> codiSecret)**: Itera fins a trobar la solució o acabar els torns i retorna una estructura de dades amb totes les combinacions usades fins a arribar a la correcta. **combinacioInicial(List<Integer> novaCombinacio)**: Crea la combinación inicial que pot eliminar més possibilitats. **getPossiblesCombinacions():** Retorna totes les possibles combinacions (atribut de la classe). **generarCombinacionsRepetides()**: Crea les estructures de dades necessàries per a generar totes les possibles combinacions amb repeticions. **gCR(int: i, List<Integer> resposta)**: Backtracking per a generar totes les possibles combinacions d'una mida en

	concret i amb repeticions. **generarCombinacionsNoRepetides()**: Crea les estructures de dades necessàries per a generar totes les possibles combinacions sense repeticions. **gCnR(int i, boolean[] visitats, List<Integer> resposta)**: Backtracking per a generar totes les possibles combinacions d'una mida en concret i sense repeticions. **minmax(List<Integer> proximIntent)**: Avaluació de totes les combinacions i tria de la millor d'elles. **compararCombinacions(List<Integer> combinacio, List<Integer> codiAnterior)**: Validació d'una combinació comparant-lo amb una segona segons la posició dels paràmetres que les formen.

2.1.15 Enumeration Dificultat

Nom de l'enumeration	Dificultat
Breu descripció de la classe	És una enumeració que defineix els diferents nivells de dificultat del joc. Aquesta classe enumera tres valors constants: *FACIL, INTERMIG* i *DIFICIL* que representen els diferents nivells de dificultat disponibles.
Descripció dels atributs	No té atributs.
Descripció dels mètodes	No té mètodes addicionals, ja que és una enumeració i només defineix els valors constants per als nivells de dificultat.

2.1.16 Enumeration Rol

Nom de l'enumeration	Rol

Breu descripció de la classe	És una enumeració que defineix dos possibles valors: *CODEMAKER* i *CODEBREAKER* que representen els rols que poden tenir els jugadors del joc.
Descripció dels atributs	No té atributs, ja que és una enumeració, que és una llista fixa de valors constants.
Descripció dels mètodes	No té cap mètode, ja que és una enumeració, que només conté constants predefinides i no permet definir nous mètodes.

2.1.17 Answer

Nom de la classe	Answer
Breu descripció de la classe	És una estructura que salva els valors d'una resposta de codemaker i els paràmetres del joc. A més a més, té molts mètodes de conversió de valors per a comunicar amb el rest del sistema.
Descripció dels atributs	**Settings:** paràmetres del joc **value:** el valor de la resposta de codeMaker **alfabeth:** llista de caràcters per a la transformació del valor a String **rand:** objecte Random per a la generació dels valors aleatoris
Descripció dels mètodes	**Answer(Settings):** creadora per defecte de la classe **Check():** comprova *value* perquè la seva mida i els valors siguin d'acord amb els Settings **equalTo(Answer):** compara el valor de l'Answer a *value* **fromAnswer(Answer):** copia els valors de l'Answer a *value* **fromArray(Integer[]):** copia els valors de l'array a *value* **FromPair(Pair<Integer, Integer>):** copia els valors del Pair a *value*

	fromString(String): copia els valors del String a *value*, passant-los a enters **gameOver():** retorna *true* si *value* és d'un torn guanyador **toArray():** retorna un array que conté els valors de *value* **toString():** retorna un String que conté els caràcters que representen els valors de *value*

2.1.18 Suggestion

Nom de la classe	Suggestion
Breu descripció de la classe	És una estructura que salva els valors d'una combinació de colors i els paràmetres del joc. A més a més, té molts mètodes de conversió de valors per a comunicar amb la resta del sistema.
Descripció dels atributs	**Settings:** paràmetres del joc **value:** valors de la combinació **alfabeth:** llista de caràcters per a la transformació del valor a String **rand:** objecte Random per a la generació dels valors aleatoris
Descripció dels mètodes	**Suggestion(Settings):** creadora per defecte de la classe **Check():** comprova *value* perquè la seva mida i els valors siguin d'acord amb els Settings **equalTo(Suggestion):** compara el valor del suggestion amb el value **fromSuggestion(Suggestion):** copia els valors de suggestion al *value* **fromString(String):** copia els valors del String al *value*, passant-los a enters **fromList(List<Integer>):** copia els valors del List a *value* **toArray():** retorna un array que conté els valors de *value*

	toString(): retorna un String que conté els caràcters que representen els valors de *value* **hasDuplicates():** retorna si el *value* té valors repetits **increment():** incrementa el *value* un pas **randomize():** genera el *value* aleatori **toList():** retorna un List que conté els valors de *value*

2.1.19 Settings

Nom de la classe	Settings
Breu descripció de la classe	Conté els paràmetres del joc per l'ús en l'algorisme Genètic
Descripció dels atributs	**combinationLength:** mida de combinació **duplicationsAllowed:** booleà per saber si estan permeses les repeticions de valors **maxMoves:** el nombre màxim de torns **numLetters:** el nombre de colors usats
Descripció dels mètodes	**Settings():** creadora per defecte de la classe **fromInfo(InfoPartida):** copia els valors d'InfoPartida a Settings

2.1.20 RandomCodeBreaker

Nom de la classe	RandomCodeBreaker

Breu descripció de la classe	Conté tots els atributs i mètodes necessaris per a l'aplicació de l'algorisme genètic
Descripció dels atributs	**settings:** salva els paràmetres del joc **nextSuggestion:** salva la combinació per al torn següent **variants:** salva l'iterator corrent **rand:** objecte Random per a la generació dels valors aleatoris
Descripció dels mètodes	**RandomCodeBreaker():** creadora per defecte de la classe **Cardinalitry():** retorna el nombre de les combinacions possibles en el moment **AnswerRecieved(answer):** instància una nova iteradora AnsweredIterator i li passa els paràmetres necessaris per al càlcul de les combinacions possibles **GetSuggestion():** retorna nextSuggestion **PeekRandom():** retorna una combinació aleatòria de la llista de les combinacions possibles

2.1.21 AnsweredIterator

Nom de la classe	AnsweredIterator
Breu descripció de la classe	Itera per la llista de les combinacions possibles i aplica el filtre de comparació de respostes i exten FilteredIterator

Descripció dels atributs	**settings:** salva els paràmetres del joc **suggestion:** salva la combinació corrent (**underlay:** salva l'iterator anterior)
Descripció dels mètodes	**AnsweredIterator(SuggestionsIterator, Suggestion, Answer, Settings):** creadora per defecte de la classe **valid():** retorna si el valor de suggestion és vàlid

2.1.21 Exhaustive

Nom de la classe	Exhaustive
Breu descripció de la classe	Itera per la llista de totes les combinacions possibles i implementa SuggestionsIterator
Descripció dels atributs	**settings:** salva els paràmetres del joc **current:** salva la combinació corrent **_eof:** salva si l'iterator ha arribat al seu final
Descripció dels mètodes	**Exhaustive(Settings):** creadora per defecte de la classe **currentValue():** retorna la combinació corrent **eof():** retorna si l'iterator ha arribat al seu final **next():** substitueix el corrent amb el suggestion següent **reset():** reinicia l'iterador a l'estat inicial

2.1.22 NoDuplicatesIterator

Nom de la classe	NoDuplicatesIterator

Breu descripció de la classe	Itera per la llista de totes les combinacions possibles que no tenen valors repetits i exten FilteredIterator
Descripció dels atributs	No té atributs propis que no siguin de la classe pare
Descripció dels mètodes	**NoDuplicatesIterator(Settings):** creadora per defecte de la classe **valid():** retorna si el valor de suggestion és vàlid

2.1.23 FilteredIterator

Nom de la classe	FilteredIterator
Breu descripció de la classe	Itera per la llista de totes les combinacions possibles que no tenen valors repetits i implementa SuggestionsIterator
Descripció dels atributs	**underlay:** salva un altre iterador
Descripció dels mètodes	**FilteredIterator(SuggestionsIterator):** creadora per defecte de la classe **valid():** retorna si el valor de suggestion és vàlid

3. Relació de les classes implementades per cada membre de l'equip

A continuació s'especifica la part de la primera entrega que ha dut a terme cada component del grup, aquesta s'ha fet i ha sigut acceptada per tots els membres del grup i s'ha intentat que sigui equilibrada. Cal comentar que, tot i el nombre de classes fet per cadascú, la mida i complexitat d'aquestes difereix notablement.

3.1 Documentació

Document	**Autor**
Diagrama de casos d'ús	Pol Farré - Marc Sardà
Descripció casos d'ús	Pol Farré - Marc Sardà
Diagrama del model	Pol Kallai
Descripció del model	Pol Farré - Pol Kallai
Documentació estructures de dades, algorismes i funcionalitats	Marc Sardà - Pol Kallai
Makefile, executables, etc.	Pol Kallai

3.2 Codi

- Classes

Classe	**Autor**
AlgoritmeFiveGuess	Marc Sardà
Codebreaker	Pol Kallai
Codemaker	Pol Kallai
InfoPartida	Pol Kallai

Partida	Pol Kallai
Usuari	Pol Kallai
CodebreakerHuma	Pol Kallai
CodebreakerMaquina	Pol Kallai
Genetic	Aglaya Khalipskaya
RandomCodeBreaker	Aglaya Khalipskaya
Answer	Aglaya Khalipskaya
Suggestion	Aglaya Khalipskaya
Settings	Aglaya Khalipskaya

- Controladors

Classe	**Autor**
ControladorDomini	Pol Kallai
ControladorJoc	Pol Kallai
ControladorPartides	Pol Kallai
ControladorUsuaris	Pol Kallai

- Drivers / Testos

Classe	**Autor**
DriverPerfil	Pol Farré
DriverPartida	Pol Farré
DriverEstadístiques	Pol Farré
Testos unitaris	Cada membre ha realitzat els tests

	de les classes que ha implementat

- Estructures de dades i altres

Classe	**Autor**
InformacioUsuari	Pol Kallai
InformacioPartida	Pol Kallai
PartidaEnJoc	Pol Kallai
Utils	Marc Sardà i Pol Farré
Pair<L,R>	Pol Kallai

- Iterators:

Classe	**Autor**
AnsweredIterator	Aglaya Khalipskaya
Exhaustive	Aglaya Khalipskaya
FilteredIterator	Aglaya Khalipskaya
NoDuplicatesIterator	Aglaya Khalipskaya

4. Estructures de dades i algorismes utilitzats

4.1 Estructures de dades

4.1.1 List<InformacioUsuari>

Hem decidit crear aquesta estructura de dades per, en un futur, passar informació d'un usuari de la capa de domini a la capa de presentació sense comprometre cap camp sensible com pot ser el *hash* de la contrasenya (present a la classe Usuari). D'aquesta manera podem enviar només la informació necessària per, per exemple, imprimir els usuaris registrats per pantalla passant els valors de l'identificador, nom d'usuari i puntuació.

4.1.2 List<InformacioPartida>

De forma molt similar a l'anterior estructura de dades descrita, aquesta també s'usa per a la mateixa raó: enviar de la capa de domini a la capa de presentació només la informació necessària per a operar amb la partida, és a dir, camps com l'identificador de la partida, la data de creació o la dificultat, entre d'altres (i evitar enviar un objecte Partida).

4.1.3 Pair<L,R>

La classe Pair s'ha creat per a casos en els quals era necessari el retorn de dos valors, un exemple que es pot trobar en el codi són les correccions d'un codi, que estan formades pel nombre de "fitxes blanques" (indiquen la presència de colors que es troben en el codi però en una posició diferent) i les "fitxes negres" (indiquen la presència de colors correctes en la posició correcta).

4.1.4 PartidaEnJoc

Per a mantenir la capa de presentació informada de l'estat de la partida fem ús d'aquesta classe per emmagatzemar dades com els intents del *codemaker,* les correccions del *codebreaker* (per imprimir-los per pantalla) o la puntuació, així evitem fer ús de classes de domini en altres capes i també el pas de dades innecessàries.

4.1.5 Mapa d'usuaris (ControladorUsuaris)

Per a emmagatzemar els usuaris registrats en l'aplicació hem decidit fer ús d'un HashMap<K, V> en el ControladorUsuaris, on hem posat l'identificador com a clau (K) i la instància d'Usuari com a valor, de manera que podem accedir eficientment quan necessitem carregar, modificar o realitzar alguna acció sobre aquest.

4.1.6 Mapa de partides (ControladorPartides)

Similarment a l'anterior punt, per a emmagatzemar les partides del sistema en memòria fem ús d'un HashMap<K, V> en el ControladorPartides, usant també com a clau l'identificador (aquest cop el de la partida) i com a valor la instància de l'objecte Partida. D'aquesta manera podem accedir també eficientment en cas de necessitar realitzar qualsevol acció.

4.1.7 Answer

Aquesta estructura guarda el valor Pair<Integer, Integer> que guarda la relació entre dues combinacions (vegeu Genetic per a la justificació) i els paràmetres del joc. A part d'això, conté diversos mètodes auxiliars per transformar valors de tipus de variables a l'Answer i viceversa.

4.1.8 Settings

Guarda els paràmetres del joc: la mida de les combinacions, el nombre de colors, el nombre de torns i si estan permets els valors duplicats. Aquests valors s'usen en les generacions de les combinacions.

4.8.9 Suggestion

Aquesta estructura guarda el valor int[] que guarda una combinació possible (vegeu Genetic per a la justificació) i els paràmetres del joc. A part d'això, conté diversos mètodes auxiliars per transformar valors de tipus de variables a Suggestion i al revers i també increment() per a incrementar el valor a 1 pas.

4.2 Algorismes

4.2.1 Five-Guess

Cost algoritmic del codi:

El cost algorítmic de l'algorisme és causat per la funció principal getCombinacio, aquesta es pot descompondre en el cost de les funcions que invoca i en el cost de les operacions realitzades en ella, per tant, el cost algorítmic és el següent:

La funció **getCombinació** invoca les funcions següents:

- **generarCombinacionsRepetides** o **generarCombinacionsNoRepetides**: El cost de les dues funcions és diferent. El cost de la generació amb repeticions és O(nColors^tamanyCombinació). En aquestes s'utilitza un backtracking per a generar totes les possibles combinacions, en el cas de no repeticions tenim un array de booleans per indicar si s'ha utilitzat o no (no varia el cost de la funció). La variable tamanyCombinacio en el nostre codi és una constant, però nColors pot anar de 4 a 8 depenent de la dificultat, per tant el cost final seria O(nColors^4). En el cas que no hi hagin repeticions fem servir un backtracking amb poda que determinaria un cost algorísmic de O(N(N-1)) que simplificadament seria O(N!), on N es el tamanyCombinacio que és constant, per tant, O(4!), que resulta en O(1).

- **combinacioInicial**: El cost d'aquesta funció és de O(N). Si hi ha repeticions el bucle recorre la llista novaCombinacio i hi agrega dos elements diferents a cada iteració, la qual cosa fa que es recorri n/2 vegades. En canvi, si no hi ha repeticions el bucle agrega un element per iteració que té un cost de O(N). La variable N representa tamanyCombinacio, que en el nostre codi és una constant, per tant: O(N^2) passa a ser O(4^2), que resulta en O(1).

- **compararCombinacions**: La primera part del codi on tenim un bucle i dins d'aquest funcionalitats varies, el cost és de O(N). Ja que n'es recorrerà tota la llista de mida "N" fent les comparacions pertinents. En la segona part del codi la complexitat és de O(N^2) perquè hi ha un doble bucle que recorre les dues llistes, i en cada iteració es compara l'element i-èsim de la combinació amb tots els elements de codiAnterior i es marca l'índex com a utilitzat si es troba una coincidència. En el pitjor dels casos, s'han de recórrer N elements al bucle exterior i N elements al bucle interior, cosa que resulta en N^2 comparacions. La variable N representa tamanyCombinacio, que en el nostre

codi és una constant, per tant: O(N^2) passa a ser O(4^2), que resulta en O(1).

- **minmax**: El cost algorítmic de la funció depèn de la mida de la llista possiblesCombinacions que en aquest cas li donarem el valor de N. A la primera part de la funció, es realitza un doble bucle que recorre totes les combinacions possibles de la llista, la qual cosa té un cost d'O(N^2). Dins aquest bucle, s'efectua una operació de cerca en una taula hash, que té un cost mitjà d'O(1), i en el pitjor dels casos O(N), també tenim la funció per comparar dues combinacions que retorna la variable feedback, explicada anteriorment (O(tamanyCombinacio^2)). A la segona part de la funció, es realitza un altre bucle que recorre totes les possibles combinacions i, per a cada combinació, es busca a la taula hash per recuperar una llista de combinacions que tenen el mateix resultat de comparació amb la combinació actual. El cost d'aquesta operació en el pitjor dels casos és O(N^2), en el cas que totes les combinacions tenen el mateix resultat de comparació, tot i que hem de tenir en compte per segona vegada el cost de la funció per comparar dues combinacions. El cost total de la funció seria O(N^2 * tamanyCombinacio^2) en el pitjor dels casos, però en el nostre codi la variable tamanyCombinacio és una constant que sempre té el mateix valor, per tant, podem afirmar que el cost de la funció és O(N^2).

A més de les operacions realitzades en les funcions invocades, la funció getCombinació efectua les operacions següents:

- Afegeix una combinació a totalCombinacions: el cost d'afegir una combinació a un ArrayList és constant, O(1).

- Elimina elements de possiblesCombinacions amb removeIf: el cost d'aquesta operació és O(N), on N és la mida de possiblesCombinacions.

- Actualitza les variables codiAnterior i resolució: el cost d'aquestes operacions és constant, O(1).

El cost algorítmic de la funció getCombinació en el pitjor dels casos és repeteixen colors i el nombre de nColors és 8 (que representa la màxima dificultat del joc). Per tant tenim en compte la següent taula:

Funció	**Cost**
generarCombinacionsRpeteides	O(8^4)
combiancioInicial	O(1)
compararCombinacions	O(1)
minmax	O(N^2)
generarCombinacionsNoRepetides (no entra en el pitjor dels casos)	O(1)
Altres operacions realitzades	O(1)

L'operació més costosa a la funció principal és el bucle while, on s'efectuen operacions amb un cost O(1) i un de cost O(N^2). A cada iteració del bucle, es realitza una eliminació d'elements en possiblesCombinacions, que pot tenir un cost màxim d'O(N) si totes les combinacions possibles són a la llista, però en general serà molt menys. La funció minmax es crida una vegada per iteració, per tant, en total tindrà un cost d'O(N^2 * k), on k és el nombre d'iteracions del bucle while i N la mida de la llista possiblesCombinacions.

Per tant, podem afirmar que el cost algorísmic de la funció getCombinació serà de O(nColors^4 + N^2 * k), on N és la mida de la llista possiblesCombinacionsi k és el nombre d'iteracions del bucle while. En general, el valor de N, k i nColors dependrà de la dificultat utilitzada per endevinar el codi secret, per la qual cosa pot variar. O(N^2 * k)

Justificació de la funcionalitat de l'algorisme:

L’algorisme five-guess es basa a donar una puntuació a la combinació amb més probabilitats d’eliminar-ne més de la llista de possibles combinacions. Aquesta seria la funcionalitat principal i és producte de la funció “minmax” que dona un valor a les combinacions depenent del punt esmentat anteriorment.

- Funció proposada:

La raó per la qual la nostra funció “minmax” és més eficient que la funció original de l'algorisme five-guess és perquè utilitzem una estratègia per reduir la quantitat de possibles combinacions que han de ser avaluades. En lloc de provar totes les combinacions possibles de colors i després eliminar les que no s'ajusten als resultats proporcionats per l'usuari, la funció “minmax” fa servir una estratègia, que consisteix a buscar la combinació que minimitza el nombre màxim de possibilitats restants, però només entre les possibles combinacions. Per aconseguir això, la funció crea una taula de puntuacions que “mappeja” els resultats de la comparació entre les combinacions possibles i la combinació a intentar (cal recalcar que la utilització d’estructures de dades que fan servir hash són més eficients a l'hora de buscar-ne un element). Després, utilitza aquesta taula per dividir el conjunt de possibles combinacions en particions de combinacions que produeixen el mateix resultat de comparació amb la combinació a intentar. Finalment, selecciona la partició amb el menor nombre màxim de possibilitats restants i tria una combinació d'aquesta partició com a proper intent.

4.2.2 Genetic

Cost algoritmic del codi:

El cost algorítmic de l’algorisme es compon de pocs elements i depèn en general dels costos dels **iteradors**. Els iteradors recorren totes les possibles combinacions amb el cost O(n). Per a cada combinació es decideix si és vàlida per a la situació o no. Cap de les combinacions es guarda enlloc, sinó que es generen usant el mètode **incremnent** de la classe **Suggestion**. Hi ha 2 tipus d'iteradors: 2 iteradors inicials (**Exhaustive** i **NoDuplicatesIterator**) que s'iteren per totes les combinacions possibles o totes les que no tenen valors duplicats respectivament, i una iteradora filtrada (**FilteredIterator** que s’expandeix amb l’**AnsweredIterator**). Després, l’

iterador **AnsweredIterator** valida les combinacions a base de la comparació de la combinació corrent amb totes les altres via el mètode **compararCombinacions()** d'**Utils**, i si el resultat no coincideix amb el rebut per paràmetre la combinació es descarta. A més a més, aquest iterador guarda un iterador previ. D'aquesta manera, cada combinació passa per una llista de tots els filtres aplicats que es creix amb cada torn nou. Aquesta funcionalitat també permet recorrer les combinacions ja descartades, garantint el cost de O(kn), on k és el nombre de torns i n és el nombre de totes les combinacions possibles.

Justificació de la funcionalitat de l'algorisme:

Quan parlem de les combinacions possibles s'ha de parlar primer sobre la relació entre combinacions. Aquesta relació és la possible resposta obtinguda si una combinació seria el codi i l'altre seria el torn. Està demostrat per Donald Knuth que per a qualsevol combinació hi ha una llista no nul·la de combinacions que es relacionen amb aquesta de la mateixa manera que la combinació amb el codi secret, i a més a més, el codi secret està dins d'aquesta llista. Aquesta llista és la llista de combinacions possibles. Si escollim una d'aquestes en aleatori i en cada torn reduïm la llista de combinacions possibles de la mateixa manera, o endevinarem a combinació amb la sort, o ens quedarem amb només una opció que serà la guanyadora.

El procés de la formació de la llista de les possibles combinacions està implementada via **iterators** que recorren totes les possibles combinacions i descarten totes que no són vàlides segons el filtre determinat.