Master Mind

1a Entrega PROP

Identificador de l'equip: 12.5

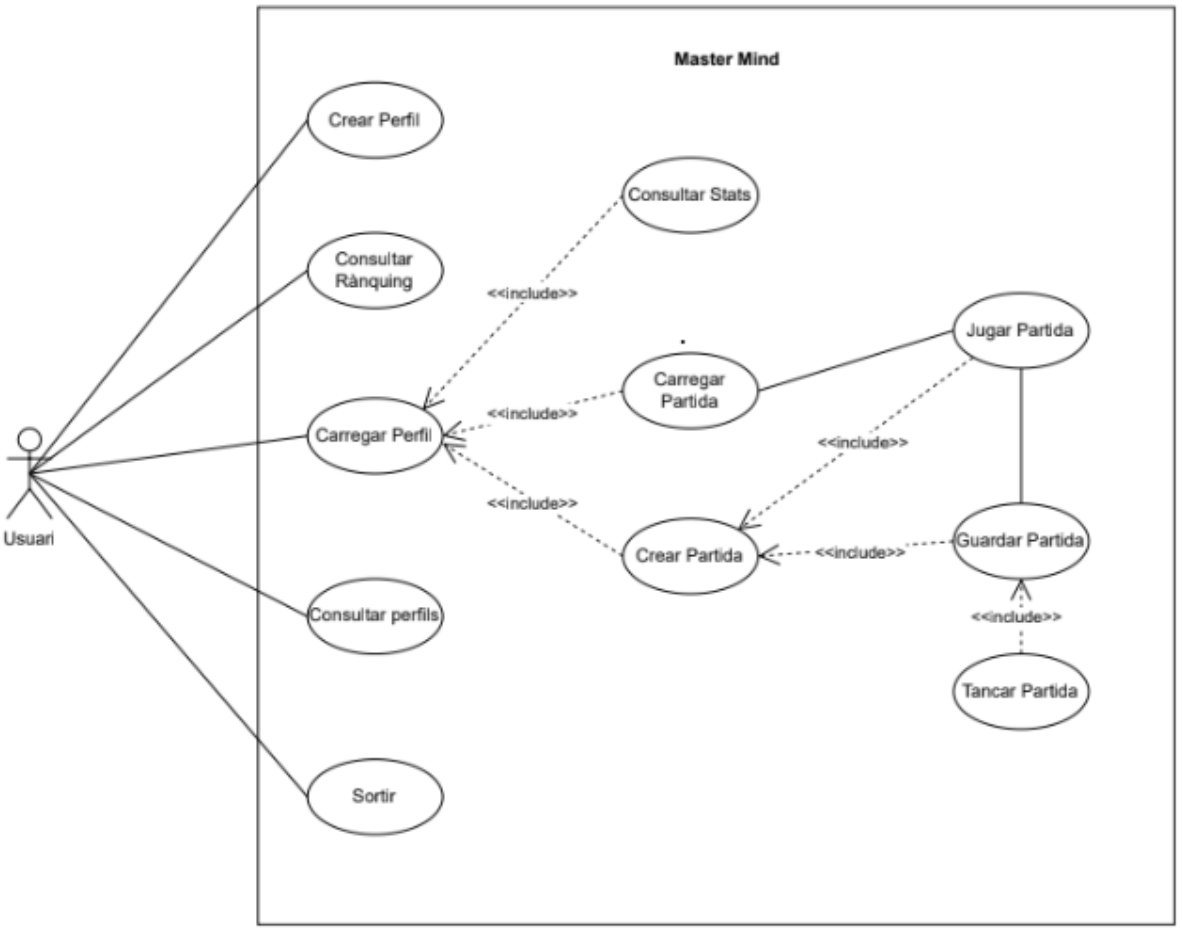
Duran López, Marc (marc.duran.lopez@estudiantat.upc.edu)
Gibert García, Miquel (miquel.gibert.garcia@estudiantat.upc.edu)
Jaume Morera, Sergi (sergi.jaume@estudiantat.upc.edu)
Puigdemont Monllor, Arnau (arnau.puigdemont@estudiantat.upc.edu)

Versió del lliurament: 1.0

ÍNDEX

1. Diagrama de casos d'ús

A continuació es mostra el diagrama de casos d'ús del sistema. El diagrama es troba també al directori DOCS en format pdf.



2. Descripció de casos d'ús

CAS D'ÚS 1:

Nom: Crear perfil

Actor: Usuari

Comportament:

1) L'usuari introdueix un nom d'identificació i una contrasenya única.
2) El sistema valida la no existència de l'usuari.
3) El sistema crea i carrega un nou usuari amb el nom introduït.

Errors possibles i cursos alternatius:

1) L'usuari ja existeix. El sistema informa de l'error.

CAS D'ÚS 2:

Nom: Carregar perfil

Actor: Usuari

Comportament:

1) L'usuari introdueix el nom d'identificació i la contrasenya que tenia registrada
2) El sistema valida la existencia de l'usuari
3) El sistema carrega el perfil desitjat.

Errors possibles i cursos alternatius:

1) El nom introduït no existeix. El sistema informa de l'error.
2) La contrasenya es incorrecte.

CAS D'ÚS 3:

Nom: Consultar ranking

Actor: Usuari

Comportament:

1) El sistema mostra el llistat d'usuaris registrats ordenats per puntuació màxima (record) segons el rol

CAS D'ÚS 4:

Nom: Consultar perfils

Actor: Usuari

Comportament:

1) El sistema mostra el llistat de perfils creats (usuaris registrats)

CAS D'ÚS 5:

Nom: Crear partida

Actor: Usuari

Comportament:

1) L'usuari prem el botó de jugar.
2) El sistema carrega una pestanya per crear una nova partida.
3) L'usuari tria el rol en el que la jugarà i la seva dificultat.

Errors possibles i cursos alternatius:

1) L'usuari no ha carregat cap perfil. El sistema informa de l'error.
2) La dificultat triada no existeix. El sistema informa de l'error.
3) El rol triat no existeix. El sistema informa de l'error.

CAS D'ÚS 6:

Nom: Carregar partida

Actor: Usuari

Comportament:

1) L'usuari prem el botó per carregar partida
2) El sistema carrega la partida guardada

Errors possibles i cursos alternatius:

1) No hi ha cap partida guardada. El sistema informa de l'error
2) S'ha d'haver iniciat sessió per a poder carregar una partida guardada.

El sistema informa de l'error

CAS D'ÚS 7:

Nom: Consultar stats

Actor: Usuari

Comportament:

1) El sistema mostra les estadístiques de l'usuari, amb les seves puntuacions totals segons el rol en el qual ha jugat, les vegades que ha començat una partida, les vegades que l'ha acabat, i les vegades que ha demanat ajut en el joc

CAS D'ÚS 8:

Nom: Sortir

Actor: Usuari

Comportament:

1) L'usuari prem el botó de sortir
2) El sistema tanca el programa

CAS D'ÚS 9:

Nom: Guardar partida

Actor: Usuari

Comportament:

1) L'usuari prem el botó per guardar partida
2) El sistema guarda la partida que es troba en curs

Errors possibles i cursos alternatius:

1) Ja existeix una versió guardada de la mateixa partida. El sistema la sobreescriu.

2) Si ja existeix una partida guardada la sobreescriu. (De moment)

CAS D'ÚS 10:

Nom: Jugar partida

Actor: Usuari

Comportament:

1) La partida s'executa amb les configuracions desitjades i l'usuari comença a jugar

2) CodeMaker:
    a) Introdueixes el codi que haurà de resoldre la màquina.
    b) A cada ronda la màquina proposa una seqüència i has de donar-li el feedback (per tal d'evitar trampes).
    c) Al final de la ronda el sistema pregunta si vols guardar la partida.
    d) Si la màquina endevina la seqüència abans de 10 rondes perds, en cas contrari, guanyes.
    e) S'obté una puntuació si es guanya, s'actualitza el ranking i els stats de l'usuari.
    f) Acaba la partida.

3) CodeBreaker:
    a) A l'inici de cada ronda el sistema et pregunta si vols ajuda. Aquesta consisteix en que et dona un color de la seqüència a resoldre que encara no tens a la teva seqüència actual.
    b) El sistema et demana que introdueixis un codi de mida corresponent segons la dificultat que has triat.
    c) El sistema et dona el feedback de la seqüència introduïda anteriorment.
    d) Al final de la ronda el sistema pregunta si vols guardar la partida.
    e) Si resolts la seqüència, guanyes la partida, en cas contrari, si pasen 10 rondes perds la partida.
    f) S'obté una puntuació si es guanya, s'actualitza el ranking i els stats de l'usuari.
    g) Acaba la partida.

CAS D'ÚS 11:

Nom: Tancar partida

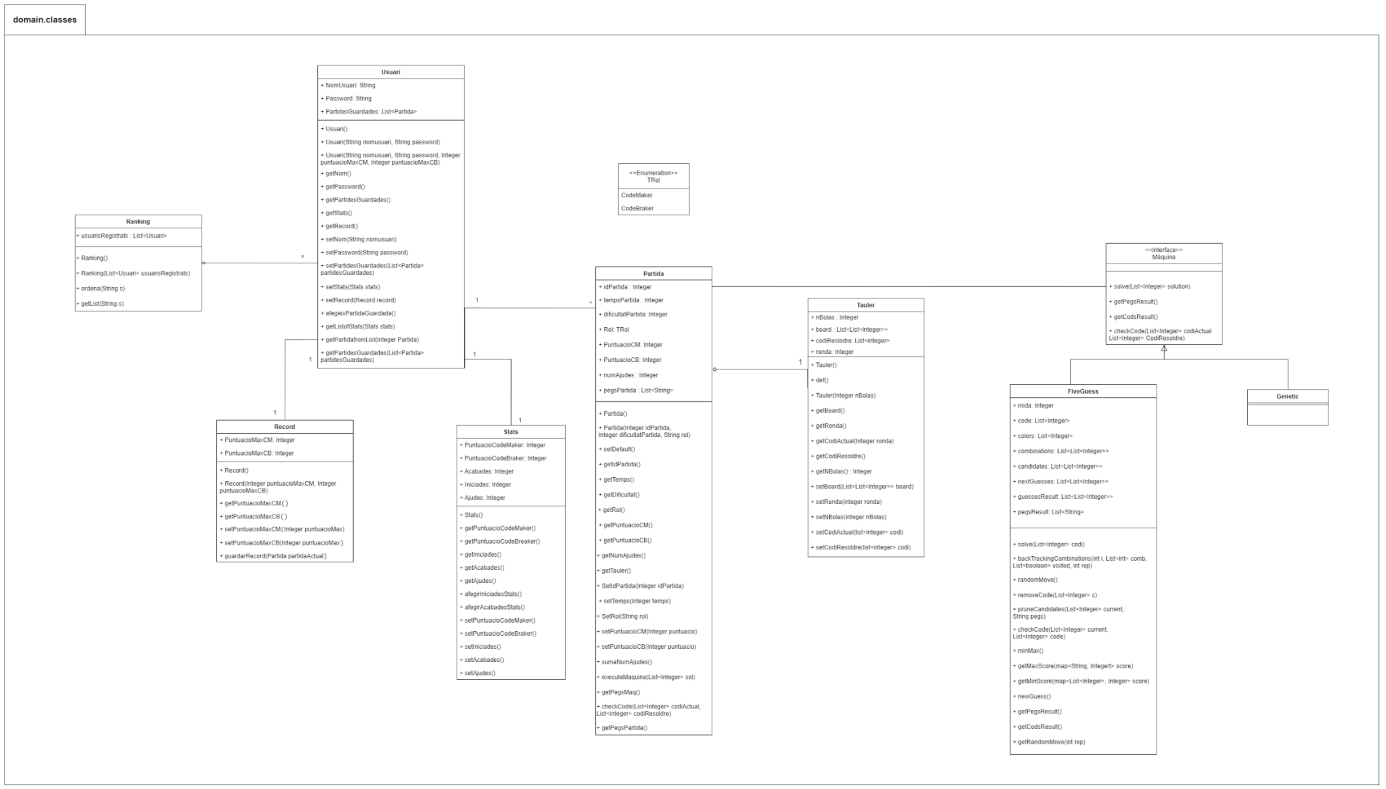
Actor: Usuari

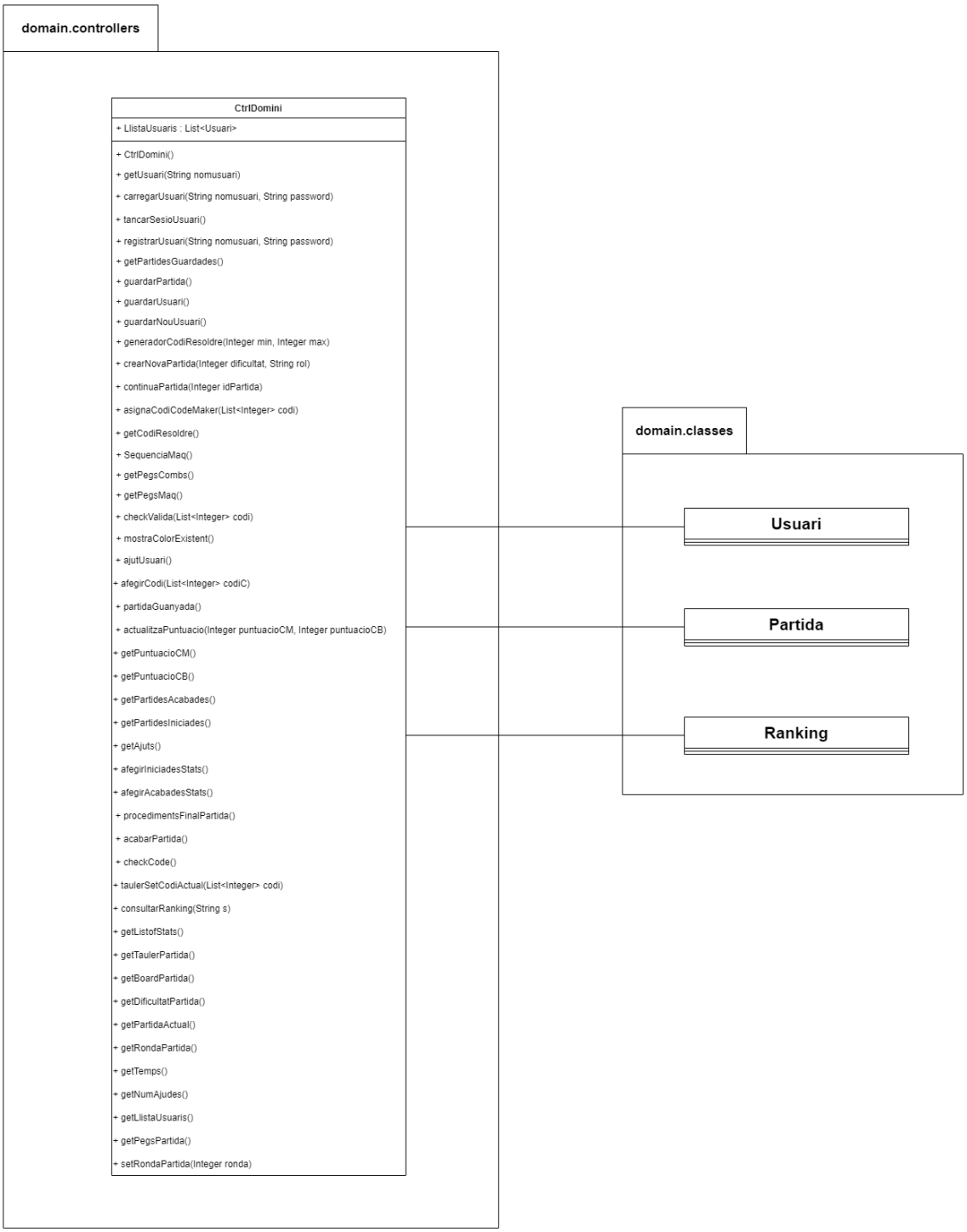
Comportament:

1) L'usuari surt de la partida en la que es troba jugant.

3. Diagrama del model conceptual de dades

A continuació es mostra el diagrama del model conceptual de dades. El diagrama es troba també al directori DOCS en format pdf.




domain.controllers
CtrlDomini
+ LlistaUsuaris : List<Usuari>
+ CtrlDomini()
+ getUsuari(String nomusuari)
+ carregarUsuari(String nomusuari, String password)
+ tancarSesioUsuari()
+ registrarUsuari(String nomusuari, String password)
+ getPartidesGuardades()
+ guardarPartida()
+ guardarUsuari()
+ guardarNouUsuari()
+ generadorCodiResoldre(Integer min, Integer max)
+ crearNovaPartida(Integer dificultat, String rol)
+ continuaPartida(Integer idPartida)
+ asignaCodiCodeMaker(List<Integer> codi)
+ getCodiResoldre()
+ SequenciaMaq()
+ getPegsCombs()
+ getPegsMaq()
+ checkValida(List<Integer> codi)
+ mostraColorExistent()
+ ajutUsuari()
+ afegirCodi(List<Integer> codiC)
+ partidaGuanyada()
+ actualitzaPuntuacio(Integer puntuacioCM, Integer puntuacioCB)
+ getPuntuacioCM()
+ getPuntuacioCB()
+ getPartidesAcabades()
+ getPartidesIniciades()
+ getAjuts()
+ afegirIniciadesStats()
+ afegirAcabadesStats()
+ procedimentsFinalPartida()
+ acabarPartida()
+ checkCode()
+ taulerSetCodiActual(List<Integer> codi)
+ consultarRanking(String s)
+ getListofStats()
+ getTaulerPartida()
+ getBoardPartida()
+ getDificultatPartida()
+ getPartidaActual()
+ getRondaPartida()
+ getTemps()
+ getNumAjudes()
+ getLlistaUsuaris()
+ getPegsPartida()
+ setRondaPartida(Integer ronda)
domain.classes
Usuari
Partida
Ranking

4. Descripció de les classes

Classe FiveGuess

Aquesta classe és una subclasse de la interfície Màquina i mitjançant diferents càlculs adivina el codi proposat pel jugador. Disposa d'un Integer mida utilitzat per la longitud dels codis (depèn de la dificultat de la partida), un code Integer[] amb el que es compara cada nou intent, un colors Integer[] utilitzat per generar totes les combinacions. Conté també tres List<Integer[]> combinations, candidates i nextGuesses, que contenen les possibles combinacions i les combinacions possiblement correctes. Té un List<Integer[]> guessesResult on s'emmagatzemen els codis utilitzat per adivinar el codi correcte, un List<String> pegsResult on s'emmagatzemen els feedback resultants dels codis utilitzats a guessesResult.

La conté dos getters senzilles: getGuessesResult i getPegsResult.

Altres mètodes complexos són:

- **solve(List<Integer> codi)**: Funció cridada per la creadora a partir de la interfície per resoldre la partida en cas de ser codebreaker. Necesita un valor entre 0 i 1 per executar el codi amb colors repetits o sense.

- **backtrackingCombinations(int i, Integer[] comb, boolean[] visited, int rep)**: Genera totes les possibles combinacions de colors sabent si hi hauran repetides i dificultat.

- **Integer[] randomMove()**: Retorna una combinació random.

- **removeCode(Integer[] c)**: Elimina un codi dels arrays combinations i candidates.

- **pruneCandidates(Integer[] current, String pegs)**: Elimina tots els candidats amb feedback diferents a l'últim intent.

- **String checkCode(Integer[] current, Integer[] code)**: Retorna els feedback resultants de comparar l'últim intent amb el codi correcte.

- **List<Integer[]> minmax()**: Retorna un array de codis amb alta possibilitat de ser el codi final.

- **int getMaxScore(Map<String, Integer> scoreCount)**: Retorna el màxim valor trobat al Map.

- **int getMinScore(Map<Integer[], Integer> score)**: Retorna el mínim valor trobat al Map.

- **Integer[] newGuess()**: Retorna el primer valor que existeix al vector generat per la funció minmax().

- **List<Integer> getRandomMove(int rep)**: Genera un codi random de les possibles combinacions

Interfície Maquina

La classe Maquina és una interfície trucada per la partida per definir la estrategia amb la que es resol el codi del Mastermind. En aquest cas crida a la classe FiveGuess, l'únic algoritme implementat actualment.

Classe Partida

Aquesta classe guarda la informació de les partides que s'estan jugant i les partides guardades (com a molt una). Cada partida conté un idPartida que permet identificar-les. També conté els valors de temps, dificultat, rol i ajuts, aquests són utilitzats per calcular la puntuacioCM i puntuacioCB un cop la partida ha finalitzat.
La classe crea l'objecte Tauler, aquest defineix el tauler de la partida. També es defineix l'algorisme que es vol utilitzar per resoldre la partida en cas de ser CodeMaker.

Els mètodes de la classe són (obviant setters i getters):

- **executeMaquina(List<Integer> sol)**: Crida a la funció solve de la interfície Maquina del algorisme triat. Retorna una llista de llistes de integers que representa el board amb les seqüències donades fins arribar a la solució passada per paràmetre.

- **getPegsMaq()**: Aquesta funció retorna una llista de string la informació del procés de adivinar la secuencia per part del algorisme.
  -B: El color es troba en la secuencia a resoldre en la posició correcte.
  -W: El color es troba en la secuencia.
  - : El color no es troba en la secuencia a resoldre.

- **getCodsMaq()**: Aquesta funció retorna la llista de codis emmagatzemats a la classe FiveGuess per utilitzar-los de nou al carregar partida.

- **checkCode(List<Integer> codiActua, List<Integer> codiResoldre)**: Aquesta funció retorna una string amb la informació de la situació actual de la teva secuencia. De la mateixa manera que abans (B,W,-).

Classe Ranking

Aquesta classe s'encarrega de mostrar el record (puntuació màxima) dels usuaris registrats ordenats de millor a pitjor. Per a cada rol, es mostra el seu propi ranking i per fer-ho, la classe conté una List<Usuari> usuarisRegistrats amb tots els usuaris, que tenen un Objecte Record on tenen guardada la puntuació màxima segons el rol. El ranking es mostra retornant una matriu[n][3], on n son el número d'usuaris registrats. Per a cada fila de la matriu, es mostra el número del usuari al ranking, el seu nom i el seu record. Quan es mostra el ranking, els usuaris ja estan ordenats per puntuació.

Els mètodes de la classe són (obviant setters i getters):

- **void ordena(String s)**: Aquesta funció ordena els usuaris amb rol "s" segons la seva màxima puntuació.
- **String[][] getList(String s)**: Retorna una matriu amb la posició al ranking, el nom dels usuaris ja ordenats i amb la seva puntuació màxima de rol "s".

Classe Record

Aquesta classe s'encarrega d'emmagatzemar les puntuacions màximes de cada usuari i les guarda en dues variables Integer puntuacióMaxCM i puntuacioMaxCB, segons el rol en el que juga.

Els mètodes de la classe són (obviant setters i getters):

- **void guardarRecord(Partida partidaActual)**: La funció rep la partida per accedir a la puntuació i actualitzar-la al record, si s'escau. Es té en compte el rol de la partida a l'hora d'actualitzar el record.

Classe Stats

Aquesta classe guarda les estadístiques totals de cada usuari. Les estadístiques que s'emmagatzemen són: la puntuació total com a codemaker, la puntuació total com a codebreaker, la quantitat de partides començades i acabades i la quantitat d'ajuts utilitzats.

Els mètodes de la classe són setters i getters.

Classe Tauler

Aquesta classe s'encarrega de crear el tauler de la partida segons el valor nBoles que se li passi a la creadora (nBoles s'entén com els diferents espais que hi ha en una fila del tauler). Aquest conté informació sobre l'estat actual del joc, com és el número de rondes que s'han jugat, la combinació secreta que ha de ser adivinada i els intents de combinació del jugador per a adivinar la secreta.

Els mètodes de la classe són setters i getters.

Classe Usuari

Aquesta classe s'encarrega de crear un usuari que s'identifica per un nom i té una contrasenya. L'usuari conté una llista amb les partides que aquest ha guardat(de moment només pot ser una màxim), les seves estadístiques i el seu record.

Els mètodes de la classe són (obviant setters i getters):

- **void afegeixPartidaGuardada(Partida p)**: Afegeix la partida "p" a la llista de partides guardades de l'usuari.

- **List<Integer> getListofStats(Stats stats)**: Retorna una llista d'enters amb les estadístiques "stats" de l'usuari

- **Partida getPartidafromList(int partida)**: Busca la partida amb identificador "partida" a la llista de partides guardades. Si la troba la retorna i la elimina de la llista, si no, no retorna res.

- **List<Integer>getPartidesGuardades(List<Partida>partidesGuardades)**: Retorna la llista de partides guardades que té l'usuari emmagatzemades.

Classe CtrlDomain

Aquesta classe s'encarrega de la comunicació amb les classes Usuari, Partida i Ranking. Crea una List<Usuari> llistaUsuaris que conté els usuaris registrats al sistema.

Els mètodes de la classe són (obviant setters i getters):

- **getUsuari(String nomusuari)**: Busca a la llista d'usuaris del sistema l'usuari corresponent al nom d'usuari passat per paràmetre.

- **carregarUsuari(String nomusuari, String password)**: Funció que permet carregar l'usuari (inicia sessió).

- **tancarSessioUsuari()**: Funció que permet a l'usuari tancar sessió.

- **registrarUsuari(String nomusuari, String password)**: Funció que permet a l'usuari registrarse.

- **guardarPartida()**:Guarda la partida.

- **guardarUsuari()**: Guarda l'usuari.

- **guardarNouUsuari()**: Crea y guarda un nou usuari.

- **generadorCodiResoldre(Integer min, Integer max)**: Funció que permet generar la seqüència a resoldre pel CodeBreaker de manera random.

- **creaNovaPartida(Integer dificultat, String rol)**: Funció que permet crear una nova partida.

- **continuaPartida(Integer idPartida)**: continua la partida amb idPartida.

- **checkValida()**: Funció que permet comprovar que la seqüència introduïda utilitza colors vàlids.
- **mostraColorExistent()**:Mostra un color de la seqüència a resoldre que no te la seqüència introduïda pel CodeBreaker.

- **ajutUsuari()**: Crida a mostraColorExistent() i fa comprovacions.

- **partidaGuanyada(String rol)**: Comprova si el CodeBreaker a resolt la seqüència proposada pel CodeMaker .

- **actualitzaPuntuacio(Integer puntuacioCM, Integer puntuacioCB)**: Funció que actualitza la puntuació del CodeMaker i CodeBreaker de la partida actual del sistema.

- **procedimentsFinalPartida()**: Funció que borra de les partides guardades la partida actual si hi està i actualitza stats.

- **acabarPartida(Integer rol)**: Acaba la partida actual.

- **checkCode()**: funció que retorna els feedback després de comparar el codi que l'usuari dona amb el codi correcte.

- **ConsultarRanking(String rol)**: Funció que obté el ranking segons el rol.

5. Relació del treball fet per cada membre de l'equip

- **CLASSES I CONTROLADORS:**
    - **Sergi Jaume:**
        - Classe Usuari
        - Classe Ranking
        - Classe Record
        - Classe Stats
    - **Marc Duran:**
        - Controlador CtrlDomain
        - Classe Partida
        - Classe Màquina
    - **Arnau Puigdemont:**
        - Classe Tauler
    - **Miquel Gibert:**
        - Classe FiveGuess
- **DRIVERS:**
    - **Miquel Gibert:**
        - Driver driverDomain
    - **Arnau Puigdemont:**
        - Driver driverDomain
- **ALTRES:**
    - **Sergi Jaume:**
        - Tests Junit
        - Jocs de prova
        - Makefile
    - **Marc Duran:**
        - Excepcions
        - Jocs de prova

6. Estructures de dades i algorismes utilitzats

Per a implementar les funcionalitats principals del projecte, hem decidit treballar amb les següents estructures de dades:

- **Usuari:**

  Com ja hem comentat, un usuari ha de poder guardar partides i més endavant carregar-les. Per això, hem decidit utilitzar una llista (List) de tipus Partida ja que la interfície "List" representa una col·lecció ordenada d'elements. (De moment, només es pot guardar una com a màxim).

  D'aquesta manera, podem controlar la posició on insertem cada element de la llista i a més a més, accedir a aquests mitjançant l'índex que ocupa dins d'ella. El seu cost és O (1)**.**

- **Tauler:**

  Per a implementar el tauler, hem decidit representar-lo amb una llista de llistes d'enters. List<List<Integer>>, ja que igual que en el cas d'usuari, la interfície List de Java ens permet accedir als índex en tot moment de la llista. El cost d'aquesta implementació és O (n*m).

- **Ranking:**

  El ranking, per a mostrar-lo, ens hem decantat per a fer servir una matriu de n*3 que realitza operacions d'ordre constant, on n són el número d'usuaris registrats al sistema. El seu cost és O(n log n), ja que per a mostrar el ranking, abans ordenem els usuaris amb el mètode Collections.sort de Java, que té un cost algorítmic de O(n log n).

- **FiveGuess:**

  El codi cost d'aquesta classe no es pot calcular sense saber els valors de mida i rep. Per calcular el cas de màxim cost utilitzem el pitjor els valors de mida = 6, màxima mida dels codis, i rep = 0, codis amb colors repetits.

  Aquesta generació es realitza utilitzant l'algoritme de backtracking ve donada per la mida i pels colors total, n (sempre serà 6). Si prenem mida = m el cost de la funció seria O(n^m).

A la funció de pruneCandidates, en el pitjor cas, s'ha de comparar totes les combinacions generades al backtracking amb la actual, per tant el cost total seria O(n^2).

En el pitjor cas de minmax() ocorre el mateix que al pruneCandidates, s'ha de comparar cada combinació restant amb els candidats, és a dir, O(n^2).

Totes les altres funcions tenen cost O(1), per tant, el cost total de la classe és: O(n^m + n^2 + n^2) = O(n^m).