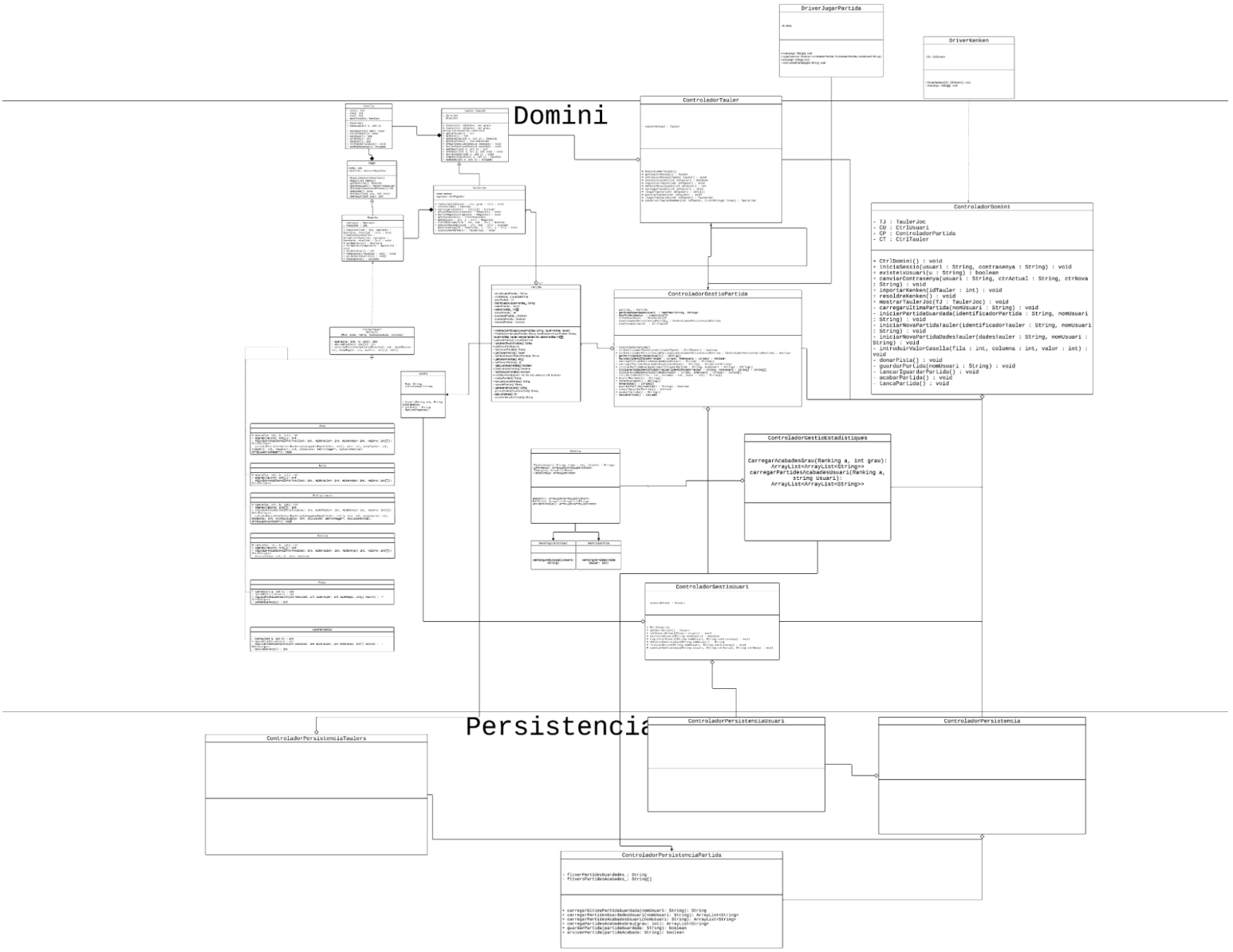
Kenken

Grup 41.5

1ª entrega PROP Versio : 1.1

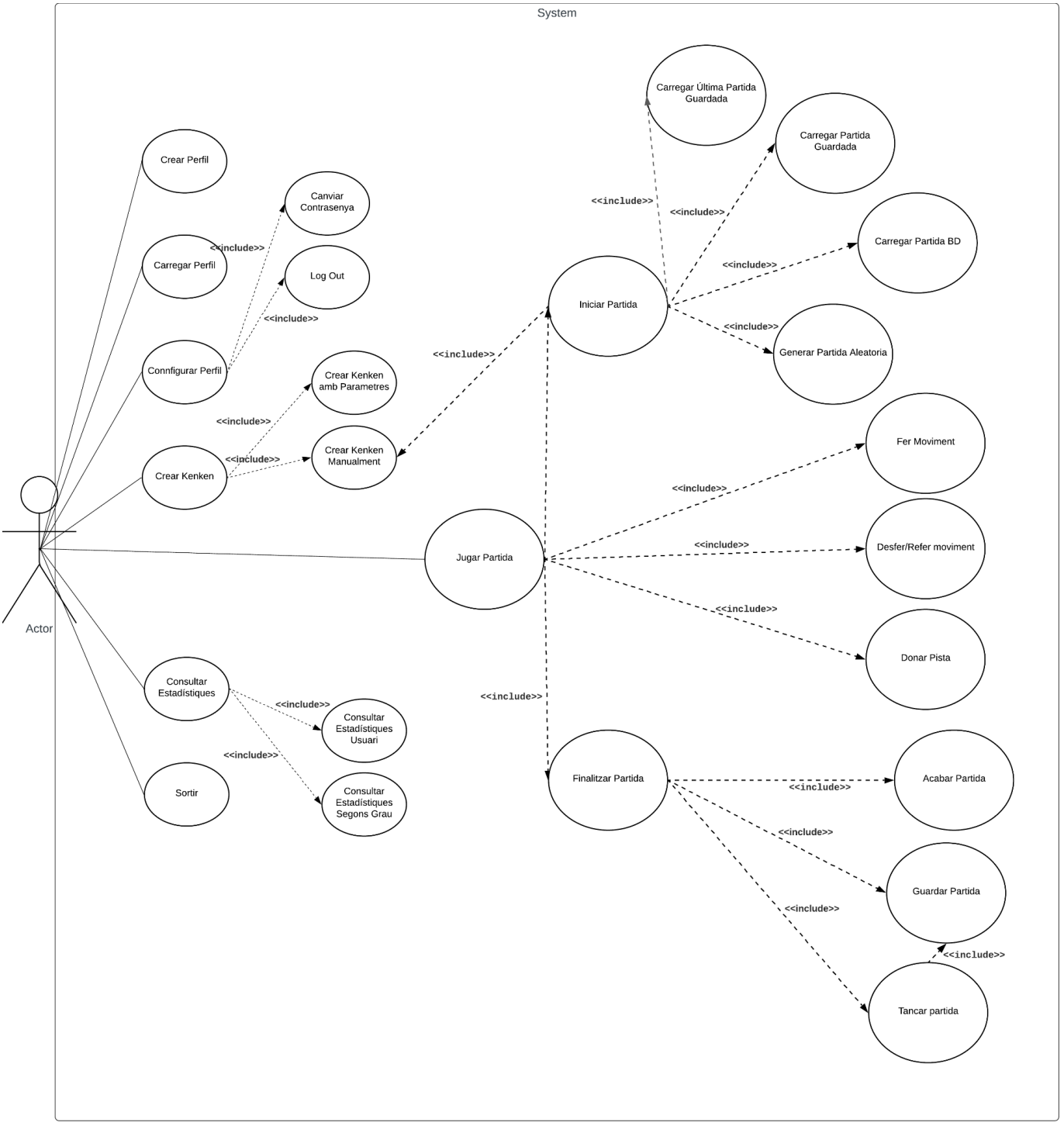
Ermias Valls Mayor : ermias.valls@estudiantat.upc.edu
Marti Tarres Pinedo : marti.tarres.pinedo@estudiantat.upc.edu
David Giribet : david.giribet@estudiantat.upc.edu
Nil Beascoechea : nil.beascoechea@estudiantat.upc.edu



Per a veure correctament el diagrama:
https://lucid.app/lucidchart/2e6561ae-b3cf-4abd-b00f-7e960ec9a9e5/edit?viewport_loc=6540%2C-13611%2C30831%2C14471%2C0_0&invitationId=inv_b91e0b9d-1845-402c-b2c6-ff9aa15a886d

Casos d’ús

Diagrama de casos d’ús



Descripció de casos d'ús

Crear Perfil

- Crear perfil:
    - Actor principal: Usuari
    - Precondició: Cap
    - Detonant: L'usuari entra a l'aplicació per primera vegada.
    - Escenari principal/Comportament:
        1. L'usuari informa del nom d'usuari, la contrasenya i confirma la contrasenya.
        2. El sistema valida els valors i crea el nou perfil amb el nom indicat
    - Extensions/possibles errors:
        - Si ja existeix un perfil amb el nom indicat, el sistema informa de l'error
        - Si la contrasenya i la seva confirmació no són iguals, el sistema informa de l'error

Carregar Perfil

- Carregar Perfil:
    - Actor principal: Usuari
    - Precondició: Cap
    - Detonant: L'usuari selecciona l'opció per carregar el seu perfil
    - Escenari principal/Comportament:
        1. El sistema sol·licita al usuari que introdueixi el nom d'usuari i la contrasenya
        2. L'usuari introdueix el nom d'usuari i la contrasenya
        3. El sistema valida les credencials i carrega el perfil de l'usuari si són correctes
    - Extensions/possibles errors:
        - Si el nom d'usuari no es troba a la base de dades, el sistema informa a l'usuari que el perfil no existeix.
        - Si el password proporcionat no coincideix amb el de l'usuari, el sistema informa a l'usuari que les credencials són incorrectes.

Configuració Perfil

- Canviar Contrasenya:
    - Actor principal: Usuari
    - Precondició: L'usuari esta loggejat
    - Detonant: L'usuari selecciona l'opció per canviar la contrasenya al seu perfil.
    - Escenari principal/Comportament:

1. El sistema sol·licita a l'usuari que introdueix la contrasenya actual i la nova contrasenya.
2. L'usuari introdueix la contrasenya actual i la nova contrasenya.
3. El sistema valida la contrasenya actual i canvia la contrasenya del perfil de l'usuari per la nova contrasenya si són correctes.

- Extensions/possibles errors:
  - Si la contrasenya actual proporcionada no coincideix amb la contrasenya del perfil de l'usuari, el sistema informa a l'usuari que la contrasenya actual és incorrecta.

- Log out:
  - Actor principal: Usuari
  - Precondició: L'usuari esta loggejat
  - Detonant: L'usuari selecciona l'opció per sortir de la seva sessió actual.
  - Escenari principal/Comportament:
    1. El sistema confirma si l'usuari vol sortir .
    2. El sistema tanca la sessió de l'usuari i redirigeix a la pàgina d'inici de sessió.
  - Extensions/possibles errors:
    - No hi ha errors possibles per al cas d'ús "Log out". Si l'usuari confirma la sortida, el sistema simplement tanca la sessió i redirigeix a l'inici de sessió. Si l'usuari decideix cancel·lar, es manté a la pàgina actual sense tancar la sessió

Crear kenken

- Crear kenken amb paràmetres:
  - Actor principal: Usuari
  - Precondició: L'usuari està registrat, ha fet log in i ha seleccionat creació de kenken
  - Detonant: L'usuari vol crear un tauler de kenken indicant els mínims paràmetres necessaris
  - Escenari principal/Comportament:
    1. L'usuari selecciona la creació de tauler de kenken amb paràmetres
    2. L'usuari indica els paràmetres: mida del tauler, operacions del tauler
    3. El sistema verifica la correctesa dels paràmetres
    4. El sistema crea un tauler correcte que compleixi les restriccions introduïdes
    5. El sistema afegeix a les seves dades el tauler generant-li un identificador
    6. El sistema afegeix a partides creades de l'usuari el tauler
  - Extensions/possibles errors:
    - Mida massa gran: la mida del tauler que es demana és més gran que el màxim

        - Operació inexistent: no existeix l'operació en el sistema

- Crear kenken manualment:
    - Actor principal: Usuari
    - Precondició: L'usuari està registrat i ha fet log in i ha seleccionat creació de kenken
    - Detonant: L'usuari vol crear un tauler de kenken indicant tots els paràmetres necessaris
    - Escenari principal/Comportament:
        1. L'usuari selecciona la creació de tauler de kenken manual
        2. L'usuari dona tots els paràmetres en el format d'entrada i sortida dels taulers: mida, operacions, peces amb la seva operació i localització
        3. El sistema verifica la correctesa dels paràmetres
        4. El sistema afegeix a les seves dades el tauler generant-li un identificador
        5. El sistema afegeix a partides creades de l'usuari el tauler
    - Extensions/possibles errors:
        - Mida massa gran: la mida del tauler que es demana és més gran que el màxim
        - Operació inexistent: no existeix l'operació en el sistema
        - Peça fora del tauler: les coordenades d'una de les caselles d'una peça estan fora dels marges del tauler
        - Solapament de peces: dues peces tenen la mateixa casella
        - Casella buida: hi ha una casella que no pertany a cap peça
        - Tauler no resoluble: El tauler donat no té cap solució

Jugar Partida

- Fer Moviment:
    - Actor principal: Usuari
    - Precondició: L'usuari està jugant una partida
    - Detonant: L'usuari vol fer un moviment a la partida, és a dir, posar un valor a una casella
    - Escenari principal/Comportament:
        i. L'usuari introdueix a on vol afegir el valor i quin
        ii. Es canvia el valor d'aquella casella
    - Extensions/possibles errors:
        i. La posició no és vàlida.
        ii. El valor no és vàlid.
- Desfer/Refer un moviment
    - Actor principal: Usuari
    - Precondició: L'usuari està jugant una partida, s'ha fet algun moviment.

- Detonant: L'usuari vol desfer un moviment a la partida, és a dir, tornar el valor de l'última casella modificada al seu estat anterior, o un cop desfet, refer el moviment.
- Escenari principal/Comportament:
  i. L'usuari tria desfer moviment
  ii. Es canvia el valor de l'última casella modificada a l'anterior
  iii. L'usuari tria si refer el moviment, continuar desfent o una de les altres opcions de Jugar
- Extensions/possibles errors:
  i. No es pot desfer més.

- Donar Pista:
  - Actor principal: Usuari
  - Precondició: L'usuari està jugant una partida
  - Detonant: L'usuari vol que se li doni una pista sobre la partida actual
  - Escenari principal/Comportament:
    i. Es soluciona el Kenken a partir de l'estat actual i se li emplena aleatòriament una casella que no tingui valor encara.
    ii. Si el Kenken està malament la pista és que ho està i on té un error.
  - Extensions/possibles errors:
    i. Cap

Iniciar Partida:

- Carregar Última Partida Guardada:
  - Actor principal: Usuari
  - Precondició: L'usuari està registrat i ha fet log in; L'usuari te mínim una partida començada però no acabada.
  - Detonant: L'usuari vol continuar l'última partida que va deixar a mitges.
  - Escenari principal/Comportament:
    i. L'usuari selecciona l'opció "Jugar" al menú principal de l'aplicació.
    ii. Després, selecciona l'opció "Carregar Última Partida Guardada".
    iii. Es comença a jugar l'última partida guardada de l'usuari.
  - Extensions/possibles errors:
    L'usuari torna enrere en qualsevol punt del comportament cancelant el procés.

- Carregar Partida Guardada:
  - Actor principal: Usuari
  - Precondició: L'usuari està registrat i ha fet log in; L'usuari te mínim una partida començada però no acabada.
  - Detonant: L'usuari vol continuar una partida a mitges.
  - Escenari principal/Comportament:
    i. L'usuari selecciona l'opció "Jugar" al menú principal de l'aplicació.
    ii. Després, selecciona l'opció "Carregar Partida Guardada".
    iii. Mostra a l'usuari una llista de les seves partides guardades.
    iv. L'usuari selecciona una de les seves partides.

        v. Es comença a jugar la partida seleccionada.
    - Extensions/possibles errors:
        i. L'usuari torna enrere en qualsevol punt del comportament cancelant el procés.

- Carregar Partida BD:
    - Actor principal: Usuari
    - Precondició: L'usuari està registrat i ha fet log in; Existeix mínim 1 tauler a la BD.
    - Detonant: L'usuari vol començar una partida nova agafant un tauler ja existent a la BD.
    - Escenari principal/Comportament:
        i. L'usuari selecciona l'opció "Jugar" al menú principal de l'aplicació.
        ii. Després, selecciona l'opció "Carregar Partida BD".
        iii. Mostra a l'usuari una llista de taulers que existeixen a la BD. Pot filtrar per que només surtin els seus.
        iv. L'usuari selecciona un dels tauler i comença a jugar.
    - Extensions/possibles errors:
        i. L'usuari torna enrere en qualsevol punt del comportament cancelant el procés.

- Generar partida aleatoria:
    - Actor principal: Usuari
    - Precondició: L'usuari està registrat i ha fet log in.
    - Detonant: L'usuari vol començar una partida nova sense haver de triar tauler.
    - Escenari principal/Comportament:
        i. L'usuari selecciona l'opció "Jugar" al menú principal de l'aplicació.
        ii. Després, selecciona l'opció "Generar Partida Aleatòria".
        iii. Es selecciona la mida de la partida que vol jugar.
        iv. Es busca a la base de dades un tauler que no hagi jugat, si no en troba en crea un de nou.
        v. L'usuari comença a jugar el tauler.
    - Extensions/possibles errors:
        i. L'usuari torna enrere en qualsevol punt del comportament cancelant el procés.
- Generar Partida Introduïnt Tauler:
    - Actor principal: Usuari
    - Precondició: L'usuari està registrat i ha fet log in.
    - Detonant: L'usuari vol començar una partida i introduïr el tauler a jugar.
    - Escenari principal/Comportament:
        i. L'usuari selecciona l'opció "Jugar" al menú principal de l'aplicació.
        ii. Després, selecciona l'opció "Generar Partida Introduïnt Tauler".
        iii. Es va a Crear Kenken Manualment.
        iv. L'usuari comença a jugar el tauler.
    - Extensions/possibles errors:
        i. L'usuari torna enrere en qualsevol punt del comportament cancelant el procés.
        ii. L'usuari ja ha jugat un kenken exactament igual.

iii. Es podria extendre a que si el pugui jugar altre cop però no pugui entrar als rankings

Finalizar Partida:

- Acabar Partida:
  - Actor principal: Usuari
  - Precondició: L'usuari està jugant una partida
  - Detonant: L'usuari creu que l'ha resolt
  - Escenari principal/Comportament:
    i. La partida es corregeix.
    ii. Si està bé s'informa a l'usuari i es guarda i es tanca.
    iii. Si està malament s'informa a l'usuari i es continua jugant.
  - Extensions/possibles errors:
    i. Cap

- Guardar Partida:
  - Actor principal: Usuari
  - Precondició: L'usuari està jugant una partida
  - Detonant: L'usuari vol guardar la partida actual
  - Escenari principal/Comportament:
    i. La partida es guarda
  - Extensions/possibles errors:
    i. Cap

- Tancar Partida:
  - Actor principal: Usuari
  - Precondició: L'usuari està jugant una partida
  - Detonant: L'usuari vol tancar i guardar la partida actual
  - Escenari principal/Comportament:
    i. La partida es guarda
    ii. La partida es tanca
  - Extensions/possibles errors:
    i. Cap

CONSULTAR ESTADÍSTIQUES:

- Consultar Estadístiques Usuari:
  - Actor principal: Usuari
  - Precondició: Cap
  - Detonant: L'usuari selecciona l'apartat de estadístiques
  - Escenari principal/Comportament:
    1. L'usuari insereix el nom d'un usuari
    2. Mostra tots els millors temps de l'usuari seleccionat
  - Extensions/possibles errors:

        - El nom d’usuari inserit no existeix

- Consultar Estadístiques segons Grau:
    - Actor principal: Usuari
    - Precondició: Cap
    - Detonant: L’usuari selecciona l'apartat de estadístiques
    - Escenari principal/Comportament:
        1. L’usuari insereix un posible grau de tauler
        2. Mostra un ranking dels temps per a aquell grau
    - Extensions/possibles errors:
        - El grau inserit no es real

SORTIR:

- Sortir:
    - Actor principal: Usuari
    - Precondició: L'aplicació està oberta
    - Detonant: L’usuari vol tancar l’aplicació
    - Escenari principal/Comportament:
        1. L’usuari selecciona l'opció sortir
        2. L’aplicació es tanca sense guardar l’estat
    - Extensions/possibles errors:
        - En cas d’estar en partida el sistema avisa a l’usuari si està segur, que perdrà tot el procediment no guardat

Descripció de classes

1. Usuari

La classe Usuari es la que conté els jugadors de l'aplicació. S’encarrega de totes les gestions dels jugadors, és a dir, crear usuaris nous, fer el login i canviar contrasenya.

2. Partida

La classe Partida representa una partida de Kenken d'un usuari amb un cert tauler. Ve determinada per un identificador únic creat a partir de l'usuari i el moment de creació de la partida.
Conté la data d'inici de la instanciació actual, el tauler de la partida, la seva mida per a tenir accés ràpid a ella, l'identificador de l'usuari que l'ha creada,

els valors de les caselles de la partida en un instant, el temps que s'ha estat jugant, i si la partida està guardada, acabada o tancada.

Atributs

- **-String identificadorPartida_**: És l'identificador de la partida, creat a partir de l'usuari i el moment de creació de la partida. El format és "identificadorUsuariPartida:yyyy-mm-ddThh:mm:ss".
- **-LocalDateTime iniciPartida_**: Data d'inici de la instanciació actual
- **-int grauPartida_**: Grau del tauler de la partida, és a dir la N d'un tauler NxN.
- **-String identificadorUsuariPartida_**:Identificador de l'usuari que ha creat la partida.
- **-TaulerJoc taulerPartida_**: Tauler de la partida.
- **-int[][] valorsPartida_**: Valors de les caselles de la partida en un instant com a una matriu d'enters.
- **-int tempsPartida_**: Temps que s'ha estat jugant. Només calculat en punts concrets per a estalviar actualitzacions periòdiques.
- **-boolean guardadaPartida_**: Indica si la partida està guardada. Es guardarà a l'arxiu pertinent de partides guardades.
- **-boolean acabadaPartida_**:Indica si la partida està acabada. Només es pot acabar una partida si està correctament resolta. No es poden fer més modificacions a la partida i es guardarà a l'arxiu pertinent de partides acabades.
- **-boolean tancadaPartida_**: Indica si la partida està tancada. Només es pot tancar una partida si està acabada. No es poden fer més modificacions a la partida.

Mètodes

- **+Partida(String, TaulerJoc)**: Crea una nova partida amb un identificador d'usuari i un tauler donats. Utilitzada en crear una partida totalment nova.
- **+Partida(String, TaulerJoc, int, int[][])**: Crea una nova partida amb un identificador d'usuari, un tauler, un temps i uns valors donats. Utilitzat per carregar una partida guardada.
- **+getiniciPartida(): LocalDateTime**: Retorna la data d'inici de la instanciació actual.
- **+getIdentificadorPartida(): String**: Retorna l'identificador de la partida.
- **+getGrauPartida(): int**: Retorna el grau del tauler de la partida.
- **+getUsuariPartida(): String**: Retorna l'identificador de l'usuari que ha creat la partida.
- **+getTaulerPartida(): Tauler**: Retorna el tauler de la partida.
- **+getIdentificadorTaulerPartida(): String**: Retorna l'identificador del tauler de la partida.

- **+getValorPartida(int, int): int**: Retorna el valor d'una casella de la partida.
- **+getValorsPartida(): int[][]**: Retorna els valors de les caselles de la partida en un instant.
- **+getTempsPartida(): int**: Retorna el temps que s'ha estat jugant. Per no haver d'anar actualitzant el temps periòdicament, es calcula cada vegada que es necessita, és a dir, quan es vol guardar, acabar o tancar la partida.
- **+getGuardadaPartida(): boolean**: Retorna si la partida està guardada.
- **+getAcabadaPartida(): boolean**: Retorna si la partida està acabada.
- **+getTancadaPartida(): boolean**: Retorna si la partida està tancada.
- **+setValorPartida(int, int, int): boolean**: Estableix un valor a una casella de la partida.
- **+setGuardadaPartida(): boolean**: Posa la partida com a guardada
- **+acabaPartida(): String**: Acaba la partida si està correctament resolta. El format de la informació de la partida acabada és:
  Identificador de la partida
  Identificador de l'usuari
  Identificador del tauler
  Temps total de la partida
  Grau del tauler
  Si ha estat guardada o no.
  Per tant, cada partida acabada ocupa 6 línies.
- **+tancalGuardaPartida(): String**: Tanca i guarda la partida. Genera un text amb el format descrit a guardaPartida().
- **+guardaPartida(): String**: Guarda la partida. Per a encapsular una partida guardada, el format és:
  Identificador de la partida
  Identificador del tauler
  Temps total de la partida
  Grau del tauler
  Valors de les caselles de la partida separats per espais en files i per salts de línia en columnes.
  És a dir ocuparà 4 + Grau línies.
- **+generaPartidaText(): String**: Genera un text amb la informació de l'estat dels valors de la partida. El format és:
  'x'' 'x'' 'x'' ... 'x''\n', ..., 'x'' 'x'' 'x'' ... 'x''\n'
  On hi haurà tantes x a cada fila com el grau de la partida i tantas files com el grau de la partida.
- **-generaPartidaGuardadaText(): String**: Funció privada auxiliar per a generar un text amb la informació de la partida. Genera un text amb el format descrit a guardaPartida().
- **-calculaTemps(): int**: Funció privada auxiliar per a calcular el temps total de la partida. Ho calcula a partir de la data d'inici de la sessió de la partida actual i el temps de la crida, sumant el temps acumulat.

- **-crealdentificadorPartida(): String**: Funció privada auxiliar per a crear l'identificador de la partida. L'identificador de la partida és únic i es crea a partir de l'usuari i el moment de creació de la partida. El format és "identificadorUsuariPartida:yyyy-mm-ddThh:mm:ss".

3. **Tauler**
   La classe Tauler conté tots els taulers de KenKen de la base de dades. Conte un id que l'identifica i un grau, que es la dimensió del mateix. La seva funció és gestionar totes les dades dels taulers.

4. **TaulerJoc**
   La classe TaulerJoc s'encarrega de fer el backtracking per solucionar els KenKen.

5. **Regió**
   Proporciona una estructura per gestionar una regió en un joc, amb funcions per accedir i manipular les caselles i els seus valors associats.

   Atributs:

   - `tam`: Un enter que representa la mida de la regió.
   - `caselles`: Un ArrayList de Casella que emmagatzema les caselles que formen la regió.

   Constructors:

   - `Regio(int tamany)`: Constructor que crea una nova instància de `Regio` amb una mida especificada.
   - `Regio(ArrayList<Casella> vc)`: Constructor que crea una nova instància de `Regio` amb un ArrayList de caselles donat.

   Mètodes:

- `getCaselles()`: Retorna l'ArrayList de caselles que formen la regió.
- `getTamany()`: Retorna la mida de la regió.
- `getNumcasellesPlenas()`: Retorna el nombre de caselles plenes a la regió.
- `esBuida(int pos)`: Verifica si una casella en una posició donada està buida.
- `getCasella(int pos)`: Retorna la casella en una posició donada.
- `getValor(int pos)`: Retorna el valor de la casella en una posició donada.
- `setValor(int pos, int val)`: Estableix el valor d'una casella en una posició donada.
- `borra(int pos)`: Borra el valor d'una casella en una posició donada.

6. **RegióJoc**
   La classe RegióJoc s'encarrega de fer les funcions necessàries demanades per la classe TaulerJoc.

7. **Casella**
   La classe Casella conte un valor, el número que s'introdueix a la casella i dos valors de posició (posicio_X i posicio_Y).

8. **Ranking**
   La classe Ranking és la que conté les dades de cada Usuari, per cada grau de Tauler diferent i amb el millor temps que ha tingut a cada grau, Serveix per poder mostrar una classificació i poder ser comparat amb altres Usuaris.

9. **CtrlUsuari**
   El Controlador d'usuari s'encarrega de la gestió de mètodes de classe referents a la classe Usuari. Es comunica amb Usuari.java per tal de realitzar totes les funcionalitats referents a aquest.

10. ControladorPartida

Controlador de la capa de domini que representa una partida i, per tant, efectua les operacions convenients per a mantenir un correcte flux del joc.

Quan hi ha un joc en curs el seu atribut partida no és null, i quan no n'hi ha cap és null. També conté una pila de moviments per desfer i una per refer, on els moviments son un int[] (fila,columna,valor). El format d'encapsulament de dades de les partides està descrit la classe Partida, a guardarPartida() per a dades d'una partida guardada, a acabarPartida() per a dades d'una partida acabada, i a generarPartidaText() per a l'estat d'una partida iniciada.

Atributs

- **-Partida partida_:** La partida en curs. Null si no hi ha cap partida en curs.
- **-HashMap<String, String> partidesGuardadesUsuari_:** Un mapa que relaciona tots els identificadors de les partides d'un usuari amb el seu contingut. La seva funció és el ràpid accés a les partides guardades de l'usuari per si decideix carregar-ne una.
- **-Stack<int[]> desferMoviments_:** Pila que guarda els moviments fets a la partida per a poder desfer-los. Un moviment es representa com un vector d'enters de 3 posicions: fila, columna i el valor anterior al moviment.
- **-Stack<int[]> referMoviments_:** Pila que guarda els moviments desfets a la partida per a poder refer-los. Un moviment es representa com un vector d'enters de 3 posicions: fila, columna i el valor anterior al moviment a refer.
- **-ControladorPersistenciaPartida controladorPersistenciaPartida_:** Controlador de persistència de les partides. S'encarrega de guardar i carregar les partides.
- **-CtrlTauler controladorTauler_:** Controlador de domini dels taulers de Kenken.

Mètodes

- **+ControladorPartida() : void**: Constructor per defecte: Aquest mètode és el constructor de la classe ControladorPartida. Inicialitza la partida a null i crea un nou HashMap per a les partides guardades de l'usuari.
- **+setControladorTauler(controladorTauler : CtrlTauler) : boolean**: Setter del controlador de taulers: Aquest mètode estableix el controlador de taulers. Accepta un objecte de tipus CtrlTauler com a paràmetre i retorna true.
- **+setControladorPersistenciaPartida(controladorPersistenciaPartida : ControladorPersistenciaPartida) : boolean**: Setter del controlador de persistència de les partides: Aquest mètode estableix el controlador de persistència de les partides. Accepta un objecte de tipus ControladorPersistenciaPartida com a paràmetre i retorna true.
- **+getPartidesGuardadesUsuari() : String[]**: Getter dels identificadors de les partides d'un usuari: Aquest mètode retorna un array de Strings amb els

identificadors de les partides de l'usuari. Si l'usuari no té cap partida guardada, retorna un array buit.

- **+haJugat(identificadorTauler : String, nomUsuari : String) : boolean**: Indica si hi ha cap existència en memòria d'una partida guardada amb l'identificador del tauler donat per aquell usuari: Aquest mètode retorna true si l'usuari ha jugat aquest tauler, false si no.
- **+carregarUltimaPartidaGuardada(nomUsuari : String) : String[]**: Carrega l'última partida guardada de l'usuari: Aquest mètode carrega l'última partida guardada de l'usuari i comença la partida amb les dades guardades. Retorna un array de Strings amb l'estat de la partida a l'índex [0] i les dades del tauler a l'índex [1].
- **+carregarPartidesGuardadesUsuari(nomUsuari : String) : ArrayList<String>**: Carrega totes les partides guardades de l'usuari per a un ràpid accés: Aquest mètode carrega totes les partides guardades de l'usuari per a un ràpid accés. Retorna un array de Strings amb els identificadors de les partides de l'usuari.
- **+iniciarPartidaGuardada(identificadorPartida : String, nomUsuari : String) : String[]**: Carrega una partida guardada de l'usuari segons un identificador: Aquest mètode carrega una partida guardada de l'usuari segons un identificador i comença la partida amb les dades guardades. Retorna un array de Strings amb l'estat de la partida a l'índex [0] i les dades del tauler a l'índex [1].
- **+iniciaPartidaIdentificadorTauler(identificadorTauler : String, nomUsuari : String) : String[]**: Comença una partida amb el tauler identificat per identificadorTauler: Aquest mètode comença una partida amb el tauler identificat per identificadorTauler. Retorna un array de Strings amb l'estat de la partida a l'índex [0] i les dades del tauler a l'índex [1].
- **+iniciaPartidaDadesTauler(dadesTauler : String, nomUsuari : String) : String[]**: Comença una partida amb les dades del tauler introduït: Aquest mètode comença una partida amb les dades del tauler introduït. Retorna un array de Strings amb l'estat de la partida a l'índex [0] i les dades del tauler a l'índex [1].
- **+introduirValor(fila : int, columna : int, valor : int) : String[]**: Canvia l'estat de la partida, és a dir un dels seus valors: Aquest mètode canvia l'estat de la partida, és a dir, un dels seus valors. Retorna l'estat de la partida després de canviar la casella.
- **+ desferMoviment(): String[]**: Desfà l'últim moviment fet a la partida. Retorna l'estat de la partida després de desfer el moviment.
- **+ referMoviment(): String[]**: Refà un moviment desfet a la partida. Retorna l'estat de la partida després de refer el moviment.
- **+ donaPista(): String[]**: Canvia l'estat de la partida afegint un valor que portaria a una solució, o indica si el que hi ha fins ara no pot portar a una solució. Retorna l'estat de la partida després de canviar la casella.
- **+ guardarPartida(nomUsuari: String): boolean**: Guarda la partida en curs del controlador. L'afegeix també al mapa de partides guardades de l'usuari. Retorna true si s'ha guardat la partida. false si no s'ha guardat la partida.
- **+ tancarIguardarPartida(): boolean**: Tanca i guarda la partida en curs del controlador. L'afegeix també al mapa de partides guardades de l'usuari. Retorna true si s'ha tancat i guardat la partida. false si no s'ha guardat la partida, i per tant no es tanca.

- **+ acabarPartida(): String[]**: Acaba la partida en curs del controlador. Una partida acabada és aquella que està ben resolta. Retorna un vector de 3 strings. La primera indica si s'ha guardat la partida a memòria. La segona si havia estat guardada prèviament. La tercera el temps que ha durat la partida.
- **+ tancaPartida(): boolean**: Tanca la partida en curs del controlador.
- **- stringToPartida(partida: String, nomUsuari: String): Partida** : Funció privada que transforma una string de dades d'una partida en una instància de Partida. Retorna una instància de Partida amb les dades de la string.

11. CtrlTauler

El Controlador de Tauler s'encarrega de la gestió de mètodes de classe referents a la classe Tauler. Es comunica amb Tauler.java per tal de realitzar totes les funcionalitats referents a aquest.

12. Operacions

Com el seu propi nom indica, aquestes són les classes que s'encarreguen de cada una de les operacions possibles en un KenKen.

a. Suma (1)

La classe Suma és la implementació de la interfície Operació que realitza els càlculs mitjançant la suma d'enters.

Mètodes

- **+opera2(a : int, b : int) : int**: realitza la suma de dos enters i retorna el resultat.
- **+operaN(valors : int[]) : int**: realitza la suma de n enters donats en un vector i retorna la suma total.
- **+calculaPossiblesValors(resultat : int, midaTauler : int, midaRegio : int, valors : int[]) : Set<Integer>**: calcula totes les possibles solucions de dos enters d'1 a la mida del tauler que sumats donin el resultat introduït. Retorna els valors únics d'1 a midaTauler que apareixen en alguna de les solucions calculades.
- **+getNumOperacio() : int**: retorna el número de l'operació, que en aquest cas és 1.
- **-calculaPossiblesValorsBacktrack(vegadesRepetibles : int[], min : int, midaTauler : int, midaUtil : int, sumatori : int, solucions : Set<Integer>, solucioParcial : ArrayList<Integer>) : void**: és una funció auxiliar per a

calcular les possibles solucions de la suma mitjançant backtracking. Va afegint els valors trobats al seu paràmetre solucions

- **Explicació de l'algorisme principal calculaPossiblesValors():**
  Variables:
  resultat: El resultat del qual es volen trobar les possibles solucions que sumades donin aquell valor.
  midaTauler: indica el nombre de valors possibles que poden tenir les caselles així com el nombre de files i columnes. És a dir els valors poden ser de 1 a midaTauler, i el tauler te midaTauler x midaTauler caselles.
  midaRegió: mida de la regió de la qual es vol calcular els possibles valors. Les caselles buides d'aquesta regió indiquen la mida de la solució que estem buscant. Per exemple, en una regió de mida 3 buida es buscarien 3 possibles valors que sumats donin el *resultat.*
  valors: Un vector de ints que indica caselles no buides de la regió, és a dir que el resultat serà aquests valors sumats amb els possibles nombres trobats.
  Funcionament:
  Primer de tot, si el nombre de valors inicials introduït és superior o igual a la mida de la regió es llançarà una excepció, ja que no té sentit calcular-ho, es tracta d'un error.
  Seguidament calcula el màxim de possibles repeticions d'un número en una regió, que seria si aquesta tingués forma d'escala, així no calcularà possibilitats que mai es podrien donar en el context del joc, on no es poden repetir valors en columnes i files. Es crea un vector de la mida del tauler, amb el nombre de repeticions possibles per a cada valor. Per exemple el nombre de repeticions possible del 1 estaria a vegadesRepetible[0] (n-1).
  Després es redueix el resultat a trobar i la mida de la solució restant la solució amb els valors no buits de la regió. Si no són valors entre 1 i midaTauler llança excepció, si ho són, redueix la mida de la solució (*midaUtil*), el nombre de repeticions d'aquell valor, i el resultat a trobar, restant-lo pel valor (variable *sumatori*). Finalment s'entra al backtrack per a trobar les possibles solucions.
  Backtrack:
  Variables*:*
  vegadesRepetible:int[]: La variable vegadesRepetible de la funció principal
  min:int: El valor mínim que es pot posar en aquesta iteració del backtracking, la seva funció és reduïr l'espai de cerca, i no repetir valors ja posats.
  Per exemple, que sumatori = 4 en una midaTauler de 3, i midaUtil de 3, hi ha les possibles solucions 1+1+2, 2+1+1, 1+2+1. Posant aquesta variable mínim, s'aconsegueix que primer es tinguin en compte les solucions on apareix l'1 (1+1+2), i després ja no es pugui tornar a posar, evitant solucions repetides. És clar que això només es pot tenir en compte en operacions commutatives.
  midaTauler:int: La variable midaTauler de la funció principal

midaUtil:int: La mida de la solució a buscar després d'haver tingut en compte les caselles no buides de la regió.
sumatori:int: El resultat que es busca
solucions:Set<Integer>: Un HashSet amb els valors trobats fins el moment que satisfan la premisa. S'ha triat un HashSet perquè, al buscar valors únics, proporciona l'accés més ràpid (O(1)).
solucioParcial:ArrayList<Integer>: Vector amb la solució trobada fins el moment, fins al cas base no se sabrà si és vàlida.

El cas base d'aquest algorisme de backtracking és que la solucioParcial tingui la mida que es busca (*midaUtil*), i que el sumatori dels seus elements sigui la de *sumatori*, això és que *sumatori* sigui igual a 0, ja que en cada backtrack es resta per l'últim valor afegit a la solució parcial. Si es verifica que la *solucioParcial* és correcta, s'afegeixen els seus valors a *solucio*.
El cas recursiu es fa en un for, que va de min a midaTauler (inclosos els dos), i per a cada i de la iteració prova si encara es pot posar (vegadesRepetible > 0), i si pot restar a *sumatori* sense baixar de 0 (i <= sumatori). Si així és, s'afegeix a la solucioParcial, es resta 1 a vegadesRepetible[i-1], i es fa backtracking amb aquella *solucioParcial*, la i com a min, les noves *vegadesRepetible*, la *midaUtil* i *sumatori-i* com a nou resultat a buscar. Finalment es restableix l'estat previ al backtrack eliminant el valor introduit a la *solucioParcial* i sumant 1 a les *vegadesRepetibles.*

b. Resta (2)

És la implementació de la interfície Operacio que realitza els càlculs mitjançant l'operació de resta d'enters.

Mètodes

- **+opera2(int a, int b) : int**: realitza la resta de dos enters independentment de l'ordre i retorna el valor absolut de la resta.
- **+operaN(int[] valors) : int**: realitza la resta de dos enters independentment de l'ordre donats com a vector. Retorna el valor absolut de la resta de tots els enters del vector.
- **+calculaPossiblesValors(int resultat, int midaTauler, int midaRegio, int[] valors) : Set<Integer>**: calcula totes les possibles solucions de dos enters d'1 a la mida del tauler que restats donin el resultat introduït. Retorna els valors d'1 a midaTauler que apareixen en alguna de les solucions calculades. Llança una excepció si el vector té igual o més de dos valors o la regió té més de dues caselles, o si algun dels valors és 0.

- **-getNumOperacio() : int**: retorna el número de l'operació, que en aquest cas és 2.
- **Explicació de l'algorisme principal calculaPossiblesValors():**
  Variables:
  resultat: El resultat del qual es volen trobar les possibles solucions que restades donin aquell valor.
  midaTauler: indica el nombre de valors possibles que poden tenir les caselles així com el nombre de files i columnes. És a dir els valors poden ser de 1 a midaTauler, i el tauler te midaTauler x midaTauler caselles.
  midaRegió: mida de la regió de la qual es vol calcular els possibles valors. Les caselles buides d'aquesta regió indiquen la mida de la solució que estem buscant. Per exemple, en una regió de mida 2 buida es buscarien 2 possibles valors que restat donin el *resultat.*
  valors: Un vector de ints que indica caselles no buides de la regió, és a dir que el resultat serà aquests valors restats amb els possibles nombres trobats.
  Funcionament:
  Per la definició de l'operació de resta en el kenken, que només permet dos valors, aquest algorisme no precisa de backtracking, ja que els resultats són deterministes des d'un primer moment.
  Primer de tot, si la regió té mida diferent de 2, no pot ser una resta, i si el nombre de valors inicials introduït és superior o igual a la mida de la regió es llançarà una excepció, ja que no té sentit calcular-ho, es tracta d'un error.
  Si el resultat no és 0 es retorna el Set buit (ja que la solució aleshores seria el mateix valor que valors[0] i per la no repetició en fila i columna en una resta és impossible que ho sigui)
  Això deixa dos casos possibles, o valors és buit, o valors té 1 element.
  Cas 1 valor: Es comprova que l'element estigui entre 1 i midaTauler, aleshores els possibles valors que pot haver-hi (després de comprovar que estan dins els permesos) serien valors[0]-resultat i valors[0] + resultat. S'afegeixen aquests valors a la solució. Per exemple si el resultat fos 3, i el valor que hi ha a valors[0] fos 6, els nombres serien 9 i 3, ja que 9-6=3, i 6-3=3.
  Cas 0 valors: En aquest cas es fa un for de i=1 a midaTauler i 'safegeixen els dos valors *i*, *i*+resultat si es compleix que *i*+resultat està dins els valors permesos (*i* sempre ho estarà).

c. Multiplicació (3)

És la implementació de la interfície Operacio que realitza els càlculs mitjançant l'operació de multiplicació d'enters.

Mètodes

- **+opera2(int a, int b) : int**: realitza la multiplicació de dos enters i la retorna.
- **+operaN(int[] valors) : int**: realitza la multiplicació de n enters donats en un vector. Retorna la multiplicació de tots els enters del vector.
- **+calculaPossiblesValors(int resultat, int midaTauler, int midaRegio, int[] valors) : Set<Integer>**: calcula totes les possibles solucions de dos enters d'1 a la mida del tauler que multiplicats donin el resultat introduït. Retorna els valors únics d'1 a midaTauler que apareixen en alguna de les solucions calculades
- **+getNumOperacio() : int**: retorna el número de l'operació, que en aquest cas és 3.
- **-calculaPossiblesValorsBacktrack(int[] vegadesRepetibles,int min, int midaTauler, int midaUtil, int multiplicacio, Set<Integer> solucions, ArrayList<Integer> solucioParcial) : void**: és una funció auxiliar per a calcular les possibles solucions de la multiplicació mitjançant backtracking. Va afegint els valors trobats al seu paràmetre solucions.
- **Explicació de l'algorisme principal calculaPossiblesValors():**
  Variables:
  resultat: El resultat del qual es volen trobar les possibles solucions que multiplicades donin aquell valor.
  midaTauler: indica el nombre de valors possibles que poden tenir les caselles així com el nombre de files i columnes. És a dir els valors poden ser de 1 a midaTauler, i el tauler te midaTauler x midaTauler caselles.
  midaRegió: mida de la regió de la qual es vol calcular els possibles valors. Les caselles buides d'aquesta regió indiquen la mida de la solució que estem buscant. Per exemple, en una regió de mida 3 buida es buscarien 3 possibles valors que multiplicats donin el *resultat.*
  valors: Un vector de ints que indica caselles no buides de la regió, és a dir que el resultat serà aquests valors multiplicats amb els possibles nombres trobats.
  Funcionament:
  Primer de tot, si el nombre de valors inicials introduït és superior o igual a la mida de la regió es llançarà una excepció, ja que no té sentit calcular-ho, es tracta d'un error.
  Seguidament calcula el màxim de possibles repeticions d'un número en una regió, que seria si aquesta tingués forma d'escala, així no calcularà possibilitats que mai es podrien donar en el context del joc, on no es poden repetir valors en columnes i files. Es crea un vector de la mida del tauler, amb el nombre de repeticions possibles per a cada valor. Per exemple el nombre de repeticions possible del 1 estaria a vegadesRepetible[0] (n-1).
  Després es redueix el resultat a trobar i la mida de la solució dividint si es pot la solució amb els valors no buits de la regió, si no ho són es que la regió no

té solució i llença excepció, si ho és, redueix la mida de la solució (*midaUtil*), el nombre de repeticions d'aquell valor, i el resultat a trobar, dividint-lo pel valor (variable *multiplicacio*). Finalment s'entra al backtrack per a trobar les possibles solucions.

Backtrack:

Variables*:*

vegadesRepetible:int[]: La variable vegadesRepetible de la funció principal
min:int: El valor mínim que es pot posar en aquesta iteració del backtracking, la seva funció és reduïr l'espai de cerca, i no repetir valors ja posats.
Per exemple, que multiplicacio = 4 en una midaTauler de 3, i midaUtil de 3, hi ha les possibles solucions 1*2*2, 2*1*2, 2*2*1. Posant aquesta variable mínim, s'aconsegueix que primer es tinguin en compte les solucions on apareix l'1, i després ja no es pugui tornar a posar, evitant solucions repetides. És clar que això només es pot tenir en compte en operacions commutatives.
midaTauler:int: La variable midaTauler de la funció principal
midaUtil:int: La mida de la solució a buscar després d'haver tingut en compte les caselles no buides de la regió.
multiplicació:int: El resultat que es busca
solucions:Set<Integer>: Un HashSet amb els valors trobats fins el moment que satisfan la premisa. S'ha triat un HashSet perquè, al buscar valors únics, proporciona l'accés més ràpid (O(1)).
solucioParcial:ArrayList<Integer>: Vector amb la solució trobada fins el moment, fins al cas base no se sabrà si és vàlida.

El cas base d'aquest algorisme de backtracking és que la solucioParcial tingui la mida que es busca (*midaUtil*), i que la multiplicació dels seus elements sigui la de *multiplicacio*, això és que multiplicació sigui igual a 1, ja que en cada backtrack es divideix per l'últim valor afegit a la solució parcial. Si es verifica que la *solucioParcial* és correcta, s'afegeixen els seus valors a *solucio*.
El cas recursiu es fa en un for, que va de min a midaTauler (inclosos els dos), i per a cada i de la iteració prova si encara es pot posar (vegadesRepetible > 0), i si es divisor de *multiplicacio*. Si així és, s'afegeix a la solucioParcial, es resta 1 a vegadesRepetible[i-1], i es fa backtracking amb aquella *solucioParcial*, la i com a min, les noves *vegadesRepetible*, la *midaUtil* i *multiplicacio/i* com a nou resultat a buscar.
Finalment es restableix l'estat previ al backtrack eliminant el valor introduit a la *solucioParcial* i sumant 1 a les *vegadesRepetibles.*

d. Divisió (4)

És la implementació de la interfície Operacio que realitza els càlculs mitjançant l'operació de divisió d'enters.

Mètodes

- **+opera2(int a, int b) : int**: realitza la divisió de dos enters independentment de l'ordre i retorna la divisió de l'enter més gran amb el més petit.
- **+operaN(int[] valors) : int**: realitza la divisió de dos enters independentment de l'ordre donats com a vector. Retorna la divisió de l'enter més gran del vector amb el més petit.
- **+calculaPossiblesValors(int resultat, int midaTauler, int midaRegio, int[] valors) : Set<Integer>**: calcula totes les possibles solucions de dos enters d'1 a la mida del tauler que dividits donin el resultat introduït. Retorna els valors d'1 a midaTauler que apareixen en alguna de les solucions calculades.
- **+getNumOperacio() : int**: retorna el número de l'operació, que en aquest cas és 4.
- **-divisible(int a, int b) : boolean**: petita funció auxiliar que retorna true si a és divisible per b, false en cas contrari.
- **Explicació de l'algorisme principal calculaPossiblesValors():**
  Variables:
  resultat: El resultat del qual es volen trobar les possibles solucions que dividides donin aquell valor.
  midaTauler: indica el nombre de valors possibles que poden tenir les caselles així com el nombre de files i columnes. És a dir els valors poden ser de 1 a midaTauler, i el tauler te midaTauler x midaTauler caselles.
  midaRegió: mida de la regió de la qual es vol calcular els possibles valors. Les caselles buides d'aquesta regió indiquen la mida de la solució que estem buscant. Per exemple, en una regió de mida 2 buida es buscarien 2 possibles valors que dividits donin el *resultat.*
  valors: Un vector de ints que indica caselles no buides de la regió, és a dir que el resultat serà aquests valors dividits amb els possibles nombres trobats.
  Funcionament:
  Per la definició de l'operació de divisió en el kenken, que només permet dos valors, aquest algorisme no precisa de backtracking, ja que els resultats són deterministes des d'un primer moment.
  Primer de tot, si la regió té mida diferent de 2, no pot ser una divisio, i si el nombre de valors inicials introduït és superior o igual a la mida de la regió es llançarà una excepció, ja que no té sentit calcular-ho, es tracta d'un error.
  Si el resultat és 1 retorna un set buit (ja que la solució aleshores seria el mateix valor que valors[0] i per la no repetició en fila i columna en una resta és impossible que ho sigui).

Això deixa dos casos possibles, o valors és buit, o valors té 1 element.
Cas 1 valor: Es comprova que l'element estigui entre 1 i midaTauler, aleshores els possibles valors que pot haver-hi (després de comprovar cada un que està dins els permesos i sigui divisible) serien valors[0]*resultat i valors[0]/resultat. S'afegeixen aquests valors a la solució. Per exemple si el resultat fos 3, i el valor que hi ha a valors[0] fos 3, els nombres serien 1 (3/3) i 9 (3*3).
Cas 0 valors: En aquest cas es fa un for de i=1 a midaTauler i 'safegeixen els dos valors *i*, *i**resultat si es compleix que *i**resultat està dins els valors permesos (*i* sempre ho estarà).

e. **Mòdul (5)**
   Realitza la divisió entre 2 valors provant totes les possibles combinacions. Es queda amb el residu de la divisió en comptes de amb el resultat.

f. **Exponenciació (6)**
   Realitza una potència entre 2 valors provant totes les possibles combinacions fins .

Relació de les classes implementades per membre de l'equip:

Aquesta taula mostra les classes implementades per cada membre. De cada classe cada membre n'ha fet el testeig.

Martí Tarrés	Nil Beascoechea	Ermias Valls	David Giribet
Regió	Partida	TaulerJoc	Tauler
Ranking	Suma	RegioJoc	TestTauler
CtrlRanking	Resta	Casella	CtrlTauler
RegioTest	Multiplicació	CrtDomini	Usuari
RankingperMida	Divisió	Modul	TestUsuari

RankingPersonal	ControladorPartida	Exponenciacio	CtrlUsuari
	DriverJugarPartida	DriverKenken	
	ControladorPersistenciaPartida	TaulerJocTest	
	SumaTest	RegioJocTest	
	RestaTest	ModulTest	
	MultiplicacioTest	ExponenciacioTest	
	DivisioTest	ExcecioCasellaNoModifcable	
	PartidaTest (utilitza classes adaptades pel test Partida_per_test i TaulerStub)	ExcepcioCasellaJaTePosicioAssignada	

Estructura de les dades:

1. **Usuaris**
   Les dades estan emmagatzemades en 1 arxiu, el qual conté tots els usuaris. L'arxiu està ordenat amb un usuari per línia seguit de la seva contrasenya.

2. **Taulers**
   Els taulers estan ordenats tal i com està indicat a el fitxer "formatokenken.pdf". D'aquesta forma tenim 1 fitxer per cada Tauler diferent.

3. Partides
   a. Partides acabades

Una partida acabada és aquella que és correcta. Es diferencien dos casos, les que han acabat i podran entrar al ranking, i les que no, ja sigui perquè han estat guardades (no s'han acabat en una sessió de joc) o perquè s'ha utilitzat pistes per a

la seva resolució.
El format de les dades que tenen aquestes partides en la comunicació entre classes és la següent:

Identificador de la partida:String
Identificador de l'usuari:String
Identificador del tauler:String
Temps total de la partida:int
Grau del tauler:int
Si ha estat guardada o no:boolean

però concatenat tot com a string (“identificadorPartida\nIdentificadorUsuari\nIdentificadorTauler\nTempsPartida\ngrau\nguardada\n”) Per separar-ho només caldria fer un split dels \n. Sempre tindrà 6 línies.
Tot i així, a memòria secundària l'emmagatzematge per a millor eficiència segueix un format diferent.
Primer de tot, les partides acabades es poden emmagatzemar en 8 fitxers diferents. Les que no són vàlides pel ranking (boolean guardada = true) es guarden al fitxer *“PartidesAcabadesGuardades.txt”.*
Les que ho són, es guarden en fitxers en funció del seu grau, on el fitxer sempre té el nom “*PartidesAcabadesGrau%d.txt*" on %d és el grau de la partida.
El format que segueixen però és el mateix:

Identificador de la Partida
Identificador de l’usuari
Identificador del Tauler
Temps total de la partida
Grau del tauler

El boolean que s'inclou en el moment de carregar-les de memòria ve implicita en el fitxer d’on s’extreu, i tot i que no caldria posar el grau del tauler en les partides que poden participar als rankings, s’ha fet per unificar formats.

b. Partides guardades

Una partida guardada és aquella que s’ha guardat en algun punt del joc a memòria secundària, o que s’ha resolt utilitzant pistes en algun punt. Aquestes segueixen el mateix format tant per la comunicació entre classes com per a l’emmagatzematge a memòria, i és el següent:

Identificador de la partida
Identificador del tauler
Temps total de partida
Grau del tauler
Valor de totes les caselles en el moment de guardat, separant files per \n i valors amb espais.

Per tant una partida guardada ocupa sempre 4 + grauTauler línies.
A memòria es guarden a "*PartidesGuardades.txt"* i òbviament s'inicialitzaran amb el boolean de guardat a true. Una característica a tenir molt en compte és que es guarden cronològicament, l'última partida guardada estarà sempre a l'inici.

c. Partida en joc/Estat

El que es denomina en el programa estat de la partida, és l'estructura de dades que representa els valors de les caselles d'una partida en un instant concret d'una partida que s'està jugant, on el format és:

'x'' ''x'' ''x''\n'
…
'x'' ''x'' ''x''\n'

on cada x representa el valor de la casella.
Hi ha tantes files com el grau del tauler i tantes 'x' per fila (és a dir columnes) com el grau. Tot això concatenat com una string.

4. Rankings

La classe `Ranking` té la responsabilitat d'emmagatzemar i gestionar la informació relacionada amb els rànquings dels jugadors. Proporciona una estructura per emmagatzemar i gestionar informació relacionada amb els rànquings dels jugadors, inclosos els seus temps de joc en diversos taulers.

1. Atributs:
    - `informacio`: És un ArrayList que conté informació sobre els jugadors i els seus temps de joc. Cada element d'aquest ArrayList és un altre ArrayList que emmagatzema informació específica de cada jugador, com el seu identificador i el seu temps de joc.
    - `tempsgrau`: És un ArrayList d'objectes `Pack`, on cada objecte `Pack` conté informació sobre el temps de joc d'un jugador en un determinat tauler.
    - `tempstotals`: És un ArrayList d'ArrayLists d'objectes `Pack`, utilitzat probablement per emmagatzemar els temps de joc de tots els jugadors en diversos taulers diferents.
2. Classe Interna `Pack`:

    - Aquesta classe representa un paquet d'informació que conté l'identificador del jugador (`idusuari`), el seu temps de joc (`temps`), i possiblement l'identificador del tauler (`idtauler`).
3. Constructors:
    - La classe `Ranking` té un constructor protegit que inicialitza l'atribut `informacio` com un ArrayList buit.
4. Mètodes:
    - `setInfo(ArrayList<ArrayList<String>> informacio)`: Un mètode per establir la informació del rànquing.
    - `getTempsTotals()`: Un mètode per obtenir els temps totals de joc de tots els jugadors en tots els taulers.
    - `CustomComparator`: Una classe interna que implementa la interfície `Comparator` per ordenar els objectes `Pack` per temps de joc.

Ús típic:

1. Crear una instància de `Ranking`.
2. Obtenir informació sobre els jugadors i els seus temps de joc, segons el tipus de ranking que vulguem calcular mitjançant funcions de la capa de persistencia.
3. Utilitzar els mètodes proporcionats per obtenir i manipular la informació del rànquing.