KENKEN

EQUIP 41.1

Mikel Torres Martinez	*mikel.torres*
Alexander Ezequiel Martínez Hernández	*alexander.ezequiel.martinez*
Liam Garriga Rosés	liam.garriga
Pol García Vernet	*pol.garcia.vernet*

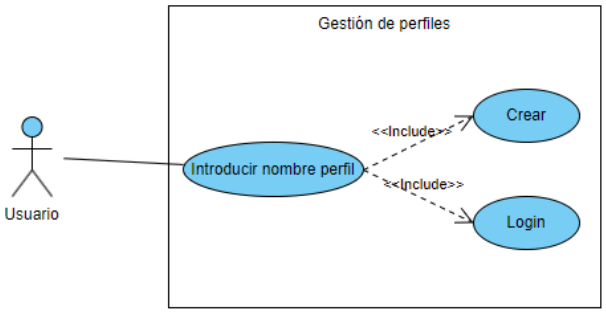
Versió 1.0

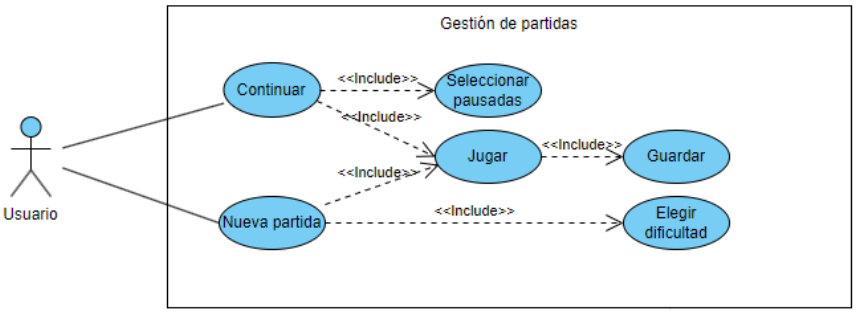
INDEX

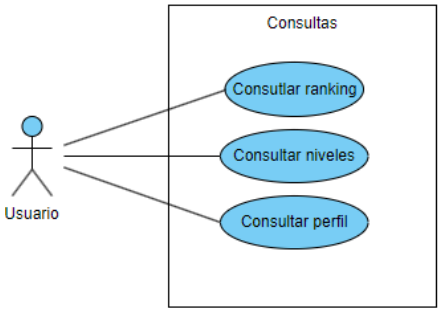
HOOOOOOOOOOOOOOOOLAAAAAAAAAAAAAAAAAAA

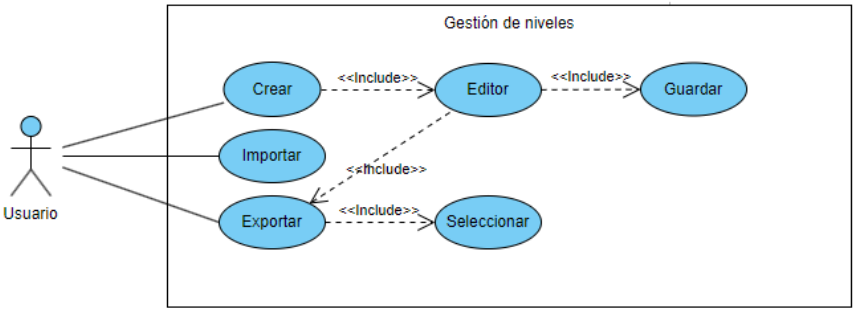
AAAAAAAAAAAAAAAAAAAAAAAAAA

1. DIAGRAMA DE CASOS D'ÚS









1.1 DESCRIPCIÓ DE CADA CAS

1.1.1 Gestión de perfiles

Crear Perfil

- Nom: Crear Perfil
- Actor: Usuari
- Comportament:
    - L' usuari introdueix el seu nom d'usuari.
    - El sistema valida el nom d'usuari i crea el nou perfil amb el nom indicat.
- Errors possibles i cursos alternatius:
    - Si l'usuari ja existeix amb aquest nom, el sistema fa log in.

Login

- Nom: Login
- Actor: Usuari
- Comportament:
    - L'usuari introdueix el seu nom d'usuari.
    - El sistema valida el nom d'usuari i inicia sessió amb aquest nom d'usuari.
- Errors possibles i cursos alternatius:
    - Si no hi ha cap usuari registrat amb aquest nom d'usuari, el sistema en crea un de nou.

1.1.2 Gestión de partidas

Continuar

- Nom: Continuar
- Actor: Usuari
- Comportament:
    - L’usuari selecciona la partida que tenia guardada.
    - El sistema permet a l’usuari començar una nova partida en cas que l’usuari no hagi seleccionat la partida guardada o continuar la partida que tenia guardada.
- Errors possibles i cursos alternatius:
    - L'usuari no pot continuar cap partida.

Nueva partida

- Nom: Nueva partida
- Actor: Usuari
- Comportament:
    - L'usuari demana al sistema començar una partida nova.
    - El sistema ofereix a l'usuari una llista de nivells ja creats.
- Errors possibles i cursos alternatius:

Seleccionar pausadas

- Nom: Seleccionar pausadas
- Actor: Usuari
- Comportament:
    - L'usuari selecciona la darrera partida que havia guardat.
    - El sistema retorna el tauler de la partida pausada i permet a l'usuari començar a jugar.
- Errors possibles i cursos alternatius:
    - Si l'usuari no té cap partida pausada, no surt l'opció de seleccionar pausades.

Jugar

- Nom: Jugar
- Actor: Usuari
- Comportament:
    - L'usuari demana al sistema començar una partida.
    - El sistema retorna un tauler i permet a l'usuari començar a jugar.
- Errors possibles i cursos alternatius:
    - Si l'usuari comet algún error en jugar, el sistema informa de l'error
    - Si el tauler és incorrecte, el sistema informa de l'error.

Guardar

- Nom: Guardar
- Actor:Usuari
- Comportament:
    - L'usuari demana al sistema guardar el tauler actual de la seva partida per poder continuar la partida més tard.
    - El sistema enregistra el nivell i l'afegeix amb la resta.
- Errors possibles i cursos alternatius:
    - La partida guardada en pot sobreescriure una amb el mateix nom o si l'usuari ja ha guardat una partida abans.

Elegir dificultad

- Nom: Elegir dificultad
- Actor: Usuari
- Comportament:
    - El sistema mostra tots els nivells disponibles classificats per dificultat
    - L'usuari escull un nivell d'una dificultat i el sistema dona accés a l'usuari per jugar aquest nivell.
- Errors possibles i cursos alternatius:

1.1.3 Consultas

Consultar ranking

- Nom: Consultar ranking
- Actor: Usuari
- Comportament:
    - L' usuari demana informació sobre el ranking d'un nivell.
    - El sistema mostra tota la informació del ranking del nivell que demana l'usuari.
- Errors possibles i cursos alternatius:
    - Si encara no existeix el ranking a consultar, el sistema informa de l'error.

Consultar niveles

- Nom: Consultar niveles
- Actor: Usuari
- Comportament:
    - L'usuari demana informació sobre els nivells.
    - El sistema retorna tots els nivells.
- Errors possibles i cursos alternatius:
    - Si no hi ha nivells per consultar, el sistema informa de l'error.

Consultar perfil

- Nom: Consultar perfil
- Actor: Usuari
- Comportament:
    - L'usuari demana informació sobre els perfils creats.
    - El sistema retorna tots els perfils.
- Errors possibles i cursos alternatius:
    - Si no existeix cap perfil, el sistema informa de l'error.

1.1.4. Gestión de niveles

Crear

- Nom: Crear
- Actor: Usuari
- Comportament:
    - L'usuari demana accés a l'editor de nivells.
    - El sistema dóna accés a l'usuari a l'editor de nivells.
- Errors possibles i cursos alternatius:
    - Si ja existeix el nivell a crear, el sistema informa de l'error.

Editar

- Nom: Editar
- Actor: Usuari
- Comportament:
    - L'usuari modifica un tauler.
    - El sistema enregistra els canvis.
- Errors possibles i cursos alternatius:
    - Si el tauler resultant de l'edició és incorrecte, el sistema informa de l'error.

Guardar

- Nom: Guardar
- Actor: Usuari
- Comportament:
    - El sistema guarda un tauler modificat.
- Errors possibles i cursos alternatius:
    - Si el nou tauler és incorrecte, el sistema informa de l'error.
    - Si el nou tauler ja existeix, el sistema informa de l'error i el pot sobrescriure.

Importar

- Nom: Importar
- Actor: Usuari
- Comportament:
    - L'usuari introdueix un fitxer que conté el nivell que vol importar.
    - El sistema valida que el fitxer contingui un nivell en un format valid i l'enregistra a la llista de nivells.
- Errors possibles i cursos alternatius:
    - Si el nom del nivell coincideix amb un nom d'un nivell ja guardat, és sobreescrit.
    - Si el format del nivell no és vàlid, el sistema informa de l'error.

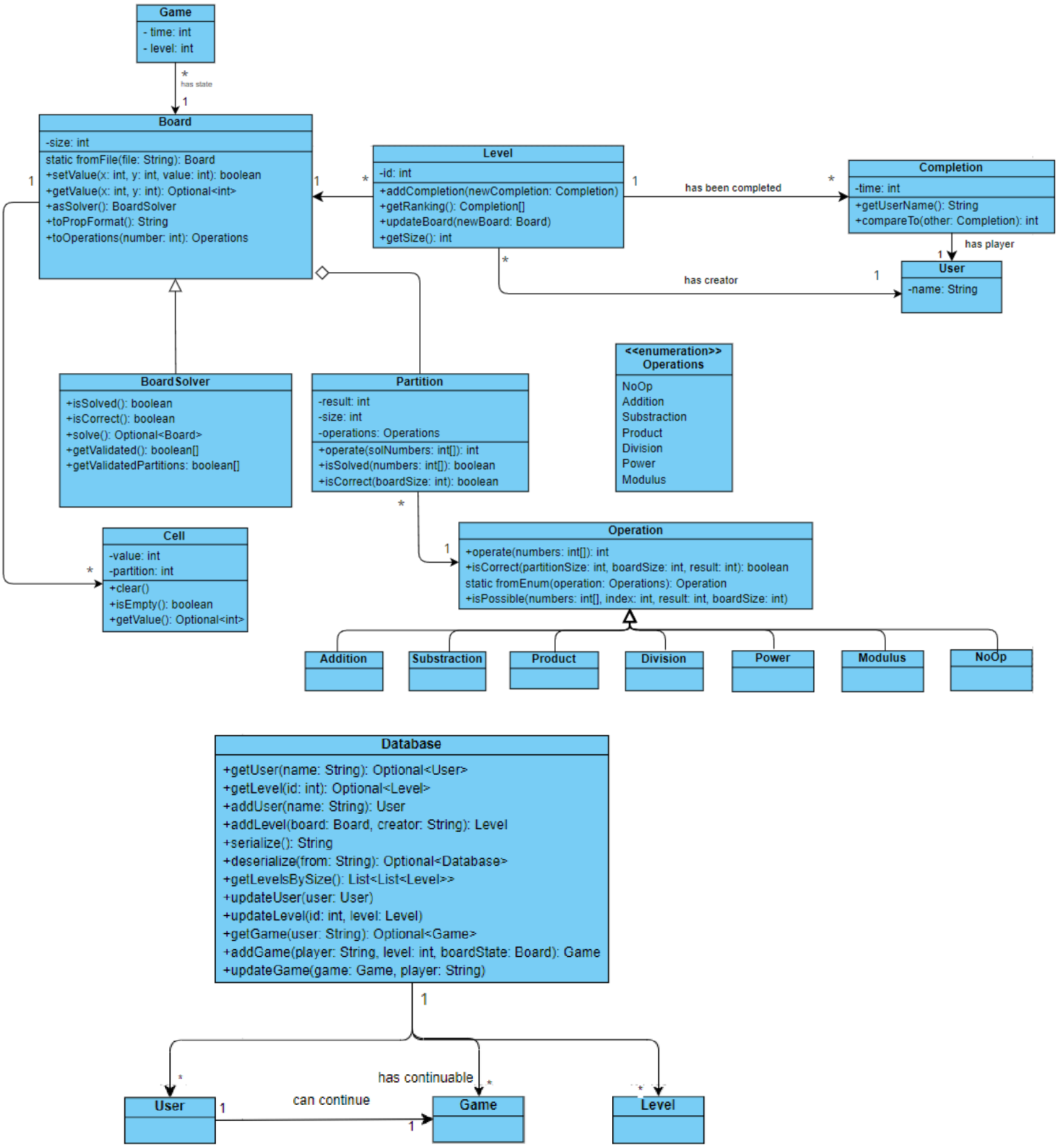
Exportar

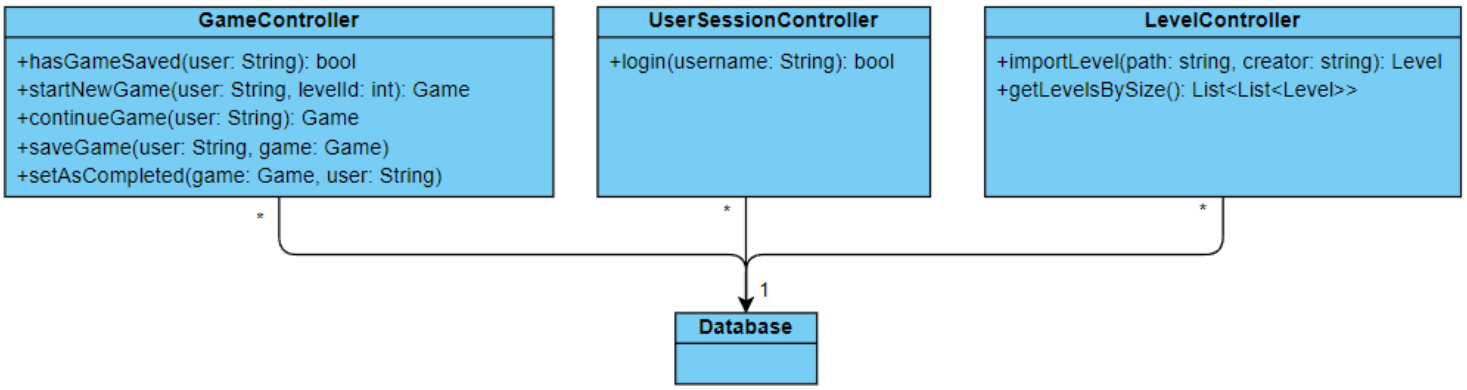
- Nom: Exportar
- Actor: Usuari
- Comportament:
    - L'usuari selecciona el nivell que vol exportar i introdueix el path on vol exportar-lo.
    - El sistema retorna el fitxer del nivell que ha seleccionat l'usuari.
- Errors possibles i cursos alternatius:
    - Si el path per exportar el nivell introduït per l'usuari no és vàlid, el sistema informa de l'error.

Seleccionar

- Nom:Seleccionar
- Actor:Usuari
- Comportament:
    - L’usuari selecciona el nivell que vol exportar.
    - El sistema exporta el nivell seleccionat per l’usuari.
- Errors possibles i cursos alternatius:

2. DIAGRAMA DEL MODEL CONCEPTUAL DE DADES





RTs

1. Board.size > 0
2. Partition.result > 0
3. 0 <= Cell.value <= N; Si value es 0, la celda está vacía
4. 0<= Cell.partition <= R donde R es el número de particiones de Board
5. 0 <= Partition.size <= N*N
6. Operation.NoOp: resultado es siempre numbers[0]; Si numbers.len = 0, resultado es 0.
7. BoardSolver tiene las mismas Cell y Partitions que Board.
8. Solo puede existir una instancia de Database, compartida por todos los Controller.
9. Game.time >= 0

2.1. Descripció d’atributs i mètodes de cada classe

User

-name: identifica el User

Completion

-time: temps en segons.

+getUserName(): obté el nom del User.

+compareTo(Completion other): compara el Completion amb other. Retorna 1 si és més gran, -1 si menor i 0 si és igual.

Board

-size: és la mida de Board N

+fromFile(String file): llegeix un file d'un kenken en format PROP y retorna un Board amb les dades llegides.

+setValue(x, y, value): fa un set del value de la Cell a la posició (x - 1)*size + y - 1.

+getValue(x, y): fa un get del value de la Cell a la posició (x - 1)*size + y - 1.

+asSolver(): es retorna com a BoardSolver.

+toPropFormat(): retorna un String amb el format PROP d'ell mateix.

+toOperations(number): retorna un Operations donat un number.

Level

-id: identifica el Level

+addCompletion(Completion newCompletion): afegeix una nova Completion.

+getRanking(): retorna el ranking (Completions) del Level.

+updateBoard(Board newBoard): fa un set del board de Level amb un newBoard.

+getSize(): retorna el size del Board.

Game

-time: és el temps desde que ha començat la partida.

-level és el ID del Level

Database

+getUser(name): retorna un clon de User amb nom name.

+getLevel(id): retorna un clon de Level amb identificador id.

+addUser(name): afegeix un User amb name name. Ho reemplaça si ja existeix.

+addLevel(board, creator): afegeix un Level amb Board board i User creator. Ho reemplaça si ja existeix.

+serialize(): serialitza la Database.

+deserialize(from): donat un String from, el transforma i retorna una base de dades corresponent.

+getLevelBySize(): una llista de cada nivell de la Database ordenada per la mida del seu Board.

+updateUser(user): actualitza l'usuari proporcionat a la Database.

+updateLevel(id, level): Substitueix el nivell identificat per ID amb el level

+getGame(user): retorna un clon de Game guardat per el User amb nom user.

+addGame(player, level, state): afegeix un Game amb player, l'identificador de level i l'estat del tauler especificats.

+updateGame(player, game): Actualitza el Game proporcionat a la Database.

BoardSolver

+isSolved(): comprova si el Board està complet.

+isCorrect(): comprova si un Board és correcte.

+solve(): Resol el tauler de kenken.

+getValidated(): Valida el tauler de kenken.

+getValidatedPartitions(): Valida les particions.

Partition

-result: el resultat de Partition

-size: el nombre de Cell que té

-operations: l'operació de Partition

+operate(int[] solNumbers): Realitza l'operació especificada sobre els números donats.

+isSolved(int[] numbers): Comprova si la mida de la partició és vàlida per al tipus d'operació i si pot assolir el resultat desitjat.

+isCorrect(BoardSize): Comprova si és possible aconseguir un resultat específic (resultat) omplint un array (nombres) amb valors que considerin un valor màxim (boardSize).

Cell

-value: el valor de la Cell.

-partition: l'identificador de la Partition a la que pertany.

+clear(): Esborra la Cell posant el seu valor a 0.

+isEmpty(): Comprova si la cel·la està buida (el valor és 0).

+getValue(): Retorna el valor de Cell.

Operation

+operate(int[] numbers): Realitza l'operació especificada sobre els números donats.

+isCorrect(partitionSize, boardSize, result): Comprova si la mida de la partició es vàlida per al tipus d'operació i si pot assolir el resultat desitjat.

+fromEnum(operations): Retorna un objecte Operation que representa el valor enum Operations especificat.

+isPossible(numbers, index, result, boardSize): Comprova si és possible aconseguir un resultat específic omplint un array amb valors que consideren un valor màxim.

GameController

+hasGameSaved(user): retorna si el User amb el nom user té un Game pendent.

+startNewGame(user, levelId): Crea un Game nou amb el nom d'usuari user i el nivell especificats i el retorna.

+continueGame(user): retorna el Game pendent del User user.

+saveGame(user, game): Desa el Game perquè l'usuari user continuï en el futur.

+setAsCompleted(game, user): Afegeix el Game a la ranking del nivell.

UserSessionController

+login(username): Inicia sessió amb l'usuari amb el nom especificat (username).

LevelController

+importLevel(path, creator): Importa un nivell d'un fitxer al path especificat i el desa a la Database.

+getLevelBySize(): Tots els nivells ordenats per dificultat (mida del Board).

3. RELACIÓ DE LES CLASSES IMPLEMENTADES PER CADA MEMBRE DE L'EQUIP

Mikel Torres:

- Partition
- Cell
- NoOp
- Subtraction
- Division
- Power
- Game
- Operation
- Addition
- Product
- Modulus
- Operations

Alexander Martínez:

- Level
- BoardSolver

Liam Garriga:

- Database
- GameController
- GameDriver
- LevelController
- LevelDriver
- UserSessionController
- UserSessionDriver
- ConsultDriver

Pol Garcia:

- User
- Completion
- Board

4. DESCRIPCIÓ DE LES ESTRUCTURES DE DADES I ALGORISMES UTILITZATS

4.1. Database - Estructura general del programa

Hem modelat el sistema com a Usuaris que juguen el joc, i per tant totes les accions han de ser dutes a terme per un usuari registrat.

Hem decidit identificar cada usuari amb el seu nom, demanant-lo en iniciar-se el programa.

Aquesta relació la representem amb un HashMap<String, User>, que ens permet accedir a les dades de cada usuari només amb el nom.

Cada partida es representa amb la classe Game, que guarda la puntuació del jugador, el nivell que s'està jugant i l'estat actual del tauler.

Això permet guardar partides de manera molt senzilla, simplement mantenint una relació entre cada Usuari i aquest Game a la base de dades.

Aquesta relació és altre cop representada amb un HashMap<String, Game>, on la clau és el nom de l'Usuari.

I finalment, se'ns demanava que poguessim importar nivells, acció que es tradueix trivialment en una llista de nivells, on cada nivell és representat pel seu índex a la llista.

4.2. Board

Estructura general

Board té dos ArrayList, un de Cell i un altre de Partition. ArrayList<Cell> és un array unidimensional que desa les Cell per la seva posició determinada per:

$$row * sizeBoard + column$$

Cada Cell sap a quina partició pertany. ***ArrayList<Partition>*** desa les Partition. ***Cell.partition*** fa referència a l'índex d'aquest array.

String toPropFormat()

L'estratègia principal és crear ArrayList bidimensional per desar les Cell per particions i una altra per guardar la seva posició. Això és necessari perquè hem d'imprimir el nombre de Cell de cada partició i les particions en ordre.

Un cop tenim totes les Cell desades a ArrayList, només fem .append() per afegir-les a un fitxer StringBuilder.

Per a cada partició fem append():

1. l'operació corresponent
    2. result
    3. El nombre de Cell, conegut gràcies a salvar les Cel·les per particions
    4. totes les seves Cell
        4.1. la posició desada a positionCells, coneguda per la mateixa estratègia que les #cel·les
        4.2. el seu contingut si el seu valor no és buit o 0

Un cop hem acabat, tornem file.toString.

Board fromFile(String file)

En primer lloc, dividim el String en una String s[] sense " " o "\n". Llegim la mida N i el nombre de regions R, que es troben en s[0] i s[1]. Creem una ArrayList<Cell> de mida N*N amb cel·les buides. Aleshores, comencem per l'índex i = 2 (s[2]), que és l'important, ja que l'incrementarem (++i) cada vegada que llegim un nombre. Comencem a llegir el fitxer en format prop amb un bucle for que itera per particions, i llegim l'operació, el resultat i el nombre de cel·les. Tenim un altre for (dins del primer) que itera pel nombre de cel·les dins d'aquesta partició.
Cada vegada que llegim una cel·la i el seu possible valor i la afegim

cellsBoard.set((sizeBoard(rowCell - 1) + colCell - 1), cell);*

Quan acabem amb totes les Cell d'aquesta partició, afegim el

new Partition(numCellsOfPartition, result, operation)

a la ArrayList<Partition>.
Un cop tenim totes les Cell i Partition, creem un Board(size, cells, partitions) i el retornem:

return new Board(sizeBoard, cellsBoard, partitionsBoard)

4.3. BoardSolver

Estructura general de BoardSolver

BoardSolver és una extensió de Board que implementa les funcionalitats interactives de Board amb l'usuari. Per tant, en termes d'estructura, un objecte BoardSolver és el mateix que un objecte Board.

Definicions bàsiques

Abans d'entrar a la lògica dels algorismes, definirem els següents principis:

1. Dues cel·les diferents estan en conflicte si i només si aquestes pertanyen al mateix tauler, tenen el mateix valor i es troben a la mateixa fila o columna.

2. Una partició d'un tauler està solucionada si i només si totes les seves cel·les tenen un valor i el resultat d'operar amb aquest valors és igual al resultat de la partició.

boolean isSolved()

Aquesta funció conté l'algorisme encarregat d'informar si el tauler kenken està solucionat. Per realitzar aquesta tasca, es basa en la lògica:

Un tauler kenken està solucionat si i només si:

1. No hi ha conflictes entre les seves cel·es
2. Totes les seves particions estan resoltes

boolean isCorrect()

La tasca d'aquest mètode és informar si l'estructura inicial d'una partició és correcta. Per aconseguir això, només cal revisar que el tamany de la partició no excedeix els límits.

Optional<Board> solve()

Aquesta és la funció que conté l'algorisme de resolució dels taulers kenken.

Abans d'entrar a la lògica de l'algorisme, explicarem els paràmetres:

1. partitionWithValues: Aquest array conté només els valors de les cel·les de cadascuna de les particions del tauler.
2. partitionWithPositions: Aquest paràmetre emmagatzema l'índex de cadascuna de les cel·les de totes les particions del tauler. Fem servir l'índex d'una cel·la perquè, encara que parlem d'un tauler NxN, en termes d'implementació, fem servir un array unidimensional de tamany NxN en lloc d'una matriu per emmagatzemar un tauler.
3. currentPartition: Ens indica quina partició del tauler estem solucionant.
4. currentPartitionCell: Ens indica quina cel·la de la partició estem intentant omplir.

Pel que fa al codi de implementació de l'algorisme:

kenkenSolver es un algorisme de ***backtracking*** que segueix la lògica descrita a continuació mitjançant restriccions que establim amb els paràmetres d'entrada.

Pel que fa a la lògica de la implementació:

La raó per la qual necessitem partitionWithValues i partitionWithPositions és que, per no tenir informació duplicada a la base de dades, hem decidit que les particions no tinguin relació amb les seves cel·les. Per tant, aquests dos arrays ens permeten tenir la mínima informació de la relació partició - cel·la només localment, el que ens evita el problema de duplicació.

L'algorisme es basa en una lògica molt simple. Per solucionar el tauler, necessitem solucionar totes les seves particions. Per tant, l'algorisme intenta solucionar totes les particions del tauler.

Si l'algorisme es troba amb una cel·la que ja està emplenada, aquest interpreta que aquesta cel·la ja està solucionada i, per tant, no intervé.

Al intentar solucionar una partició, ens podem trobar amb múltiples casos:

1. La partició ja està resolta. Aleshores, no fa falta revisar-la.
2. La partició no està resolta, pel que necessita la intervenció de l'algorisme.

És important notar que l'algorisme avalua aquests dos casos i només intenta resoldre aquelles particions que encara no estan solucionades, el que ens afavoreix en termes d'eficiència.

Com a optimització, l'algorisme verifica si la partició a resoldre te mida 1, el que significa que el valor de la cel·la ha de ser el resultat de la partició. Aleshores, intentem introduir aquest valor al tauler. Si no ho aconseguim perquè aquest valor entra en conflicte amb altres cel·les, vol dir que aquest camí que l'algorisme esta prenent és incorrecta.

Per resoldre una partició, hem de donar valors a les seves cel·les tal que, al operar amb aquestes, obtenim el resultat correcte de la partició. Per tant, l'algorisme dona valors a aquelles cel·les que encara no el tinguin fins que troba una solució.

És important deixar clar que l'algorisme només dona valors a les cel·les que compleixen amb les restriccions mínimes (aquest valor no està repetit a la mateixa fila i columna), el que afavoreix a l'eficiència.

La regla per comprovar si una combinació de cel·les resol una partició és que totes les cel·les estan plenes. Per tant, una partició de tamany tres amb valors 1, 2 i 0 no s'avaluaria sense abans emplenar el tercer valor. El mateix s'aplica per les particions.

Seguint aquest últim principi, només comprovem si hem solucionat el tauler si totes les particions estan plenes, i només comprovem si hem solucionat una partició si totes les seves cel·les estan plenes.