KenKen

Identificador de l’equip: 41.3
Membres:
Brian Fernando i Arias: @brian.fernando.arias
Daniel Espinalt i Jitkov: @daniel.espinalt
Noah Barberà i Guardiola: @noah.barbera.i
Oriol Segura i Niño: @oriol.segura.nino
Versió: 1.1

Índex

Consideracions

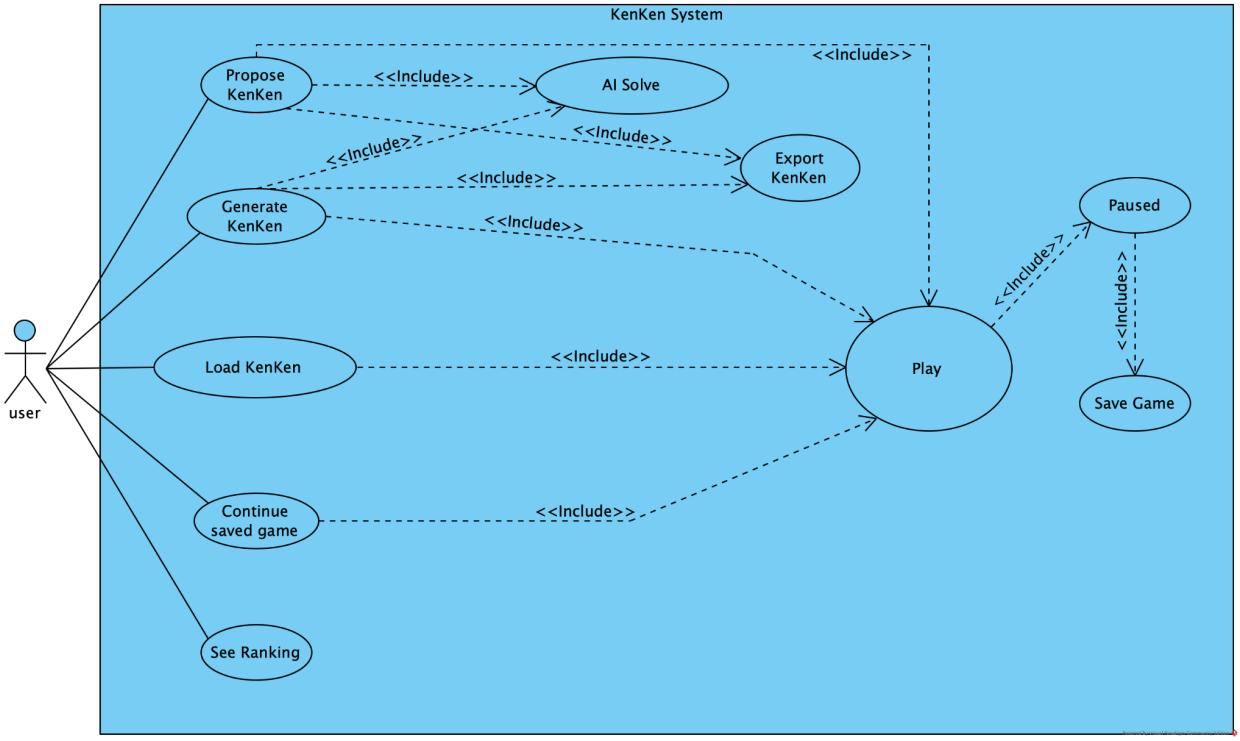
Es recomana consultar el `README.md`

OperationPower

L'operació potència `^`, per tal de proporcionar un únic resultat, sempre consistirà en calcular la potència del número més gran elevat al número més petit.

Això sempre és possible ja que l'operació potènica està limitada a dos operands, i mai podran ser número repetits ja que es tractaran de dues cel·les de la mateixa fila o columna.

Diagrama de casos d'ús



Cas ProposeKenKen

Aquest cas d'ús contempla que l'usuari proposi un KenKen al sistema. És a dir, l'usuari indica una mida (size) al sistema i a partir d'un taulell en blanc en defineix els grups, indicant-ne les seves cel·les i operacions.

Afegidament, l'usuari pot proposar uns valors fixes (que ja vindran donats i no es podran modificar).

Cas GenerateKenKen

L'usuari pot sol·licitar al sistema de generar un KenKen a partir d'una mida, una topologia de grup, i un conjunt d'operacions. Això no sempre és possible, doncs depenent de la topologia i la mida del KenKen, no serà possible generar els grups.

Afegidament, l'usuari pot sol·licitar una quantitat de valors fixes (que ja vindran donats i no es podran modificar).

Cas AISolve

Tant quan l'usuari proposa un KenKen com quan sol·licita al sistema que en generi un (en cas d'èxit), pot sol·licitar al sistema que el resolgui. El sistema implementa un backtracking per a resoldre'l.

Cas ExportKenKen

De nou, tant quan l'usuari proposa un KenKen com quan sol·licita al sistema que en generi un (en cas d'èxit), pot exportar-lo en un arxiu `.kenken` per a compartir-lo amb els seus amics o per a accedir-hi de nou en un futur.

Cas LoadKenKen

Els arxius `.kenken` exportats poden ser carregats al sistema per a jugar-hi en un futur.

Cas Play

L'usuari pot jugar amb un KenKen fins a la seva resolució o fins que es cansi.

Cas SaveGame

L'usuari pot guardar la partida (`.kenken_game`) en qualsevol moment per a reprendre-la quan vulgui.

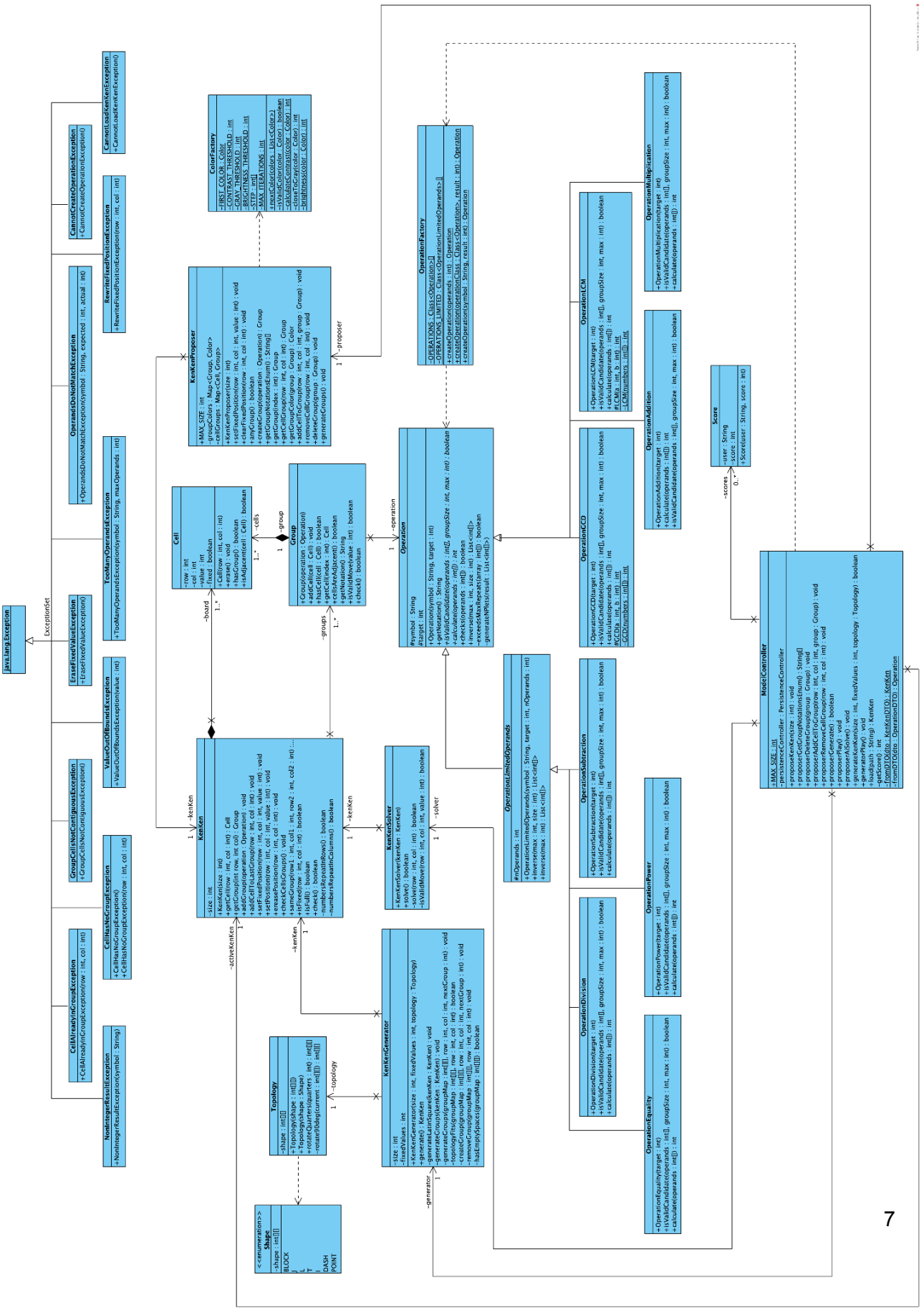
Cas ContinueSavedGame

L'usuari pot reprendre una partida guardada `.kenken_game` en qualsevol moment.

Cas SeeRanking

L'usuari pot consultar el ranking de l'arxiu `data/scores.kenken_scores` quan desitgi.

Diagrama del model conceptual



Excepcions

NonIntegerResultException

Aquesta excepció és cridada per la classe operació si el resultat d'aquesta no dóna un número positiu. Imprimeix el símbol de la operació que s'està efectuant.

CellAlreadyUsedInGroupException

Aquesta excepció és cridada majoritàriament per la classe Kenken proposer, malgrat també és cridada per altres classes. S'usa per evitar que una cel·la pertanyi a dos grups a la vegada. Imprimeix la fila i la columna de la cel·la en qüestió.

CellHasNoGroupException

Aquesta excepció és cridada per la classe Kenken per comprovar que totes les cel·les tenen un grup al que pertanyen. Pot imprimir la fila i la columna de la cel·la en qüestió.

GroupCellNotContiguousException

Aquesta excepció s'utilitza a diferents classes de Kenken per comprovar que les cel·les que pertanyen a un mateix grup estan juntes.

ValueOutOfBoundsException

Aquesta excepció s'utilitza a les classes de Kenken i comprova que els valors fixes siguin més grans que 1 i més petits que el tamany del Kenken.

EraseFixedValuePositionException

Aquesta excepció s'utilitza a les classes de cel·la i Kenken per evitar que es borri una cel·la amb valor fixe ja establert.

TooManyOperandsException

Aquesta excepció s'utilitza per no deixar que en un grup on hi ha una operació amb operands limitats s'anyadeixin més cel·les quan ja tenim el límit d'operands.

OperandsDoNotMatchException

Aquesta excepció s'utilitza quan volem crear una operació amb operands limitats i anyadim més operands dels que la operació permet.

RewriteFixedValuePositionException

Aquesta excepció s'utilitza per evitar que es modifiqui un valor fixe ja establert en el kenken.

CannotCreateOperationException

Aquesta excepció s'utilitza diverses vegades a la classe OperationFactory i salta quan la operació no pot ser creada.

CannotLoadKenKenException

Aquesta excepció s'utilitza als jocs de prova i drivers per avisar que el Kenken no s'ha carregat bé.

Enumeracions

Shape

Enumera els tipus de forma per generar un grup d'un Kenken.

Classes

Topology

Classe que representa una topologia, conté una matriu de bits per a representar-la.

KenkenGenerator

Classe que serveix pel cas d'ús de GenerateKenken. Genera un kenken a partir de un tamany, el número de valors fixes, la topologia utlitzada i una llista d'operacions.

KenkenSolver

Classe que serveix per resoldre un Kenken. Té dos mètodes solve, un void públic sense paràmetres que serveix per cridar a la classe des de fora d'aquesta, i un solve privat que té l'algoritme implementat.

KenKen

Classe que implementa un kenken, un kenken està format per cel·les i grups i té com a paràmetre el tamany d'aquest.

Group

Classe que implementa un grup d'un kenken, un grup està format per cel·les i té una operació.

Cell

Classe que implementa una cel·la, els paràmetres row i col indiquen la seva posició en el kenken, el paràmetre value indica el valor que té en aquell moment, el paràmetre fixed és un bool que indica si la cel·la té un valor a l'inici o no i el paràmetre group indica a quin grup pertany la cel·la en qüestió.

Operation

Classe que implementa la operació d'un Kenken. Té d'atributs el símbol de la operació (que la identifica) i després el resultat que ha de donar aquesta (target).

OperationLimitedOperands

Classe que implementa una operació amb operands limitats, té els mateixos atributs que la operació però amb un paràmetre més que indica el número d'operands màxim.

OperationAddition

Classe que implementa la operació de suma.

OperationSubstraction

Classe que implementa la operació de resta. Té com a màxim dos operands.

OperationPower

Classe que implementa la operació de potència. Té com a màxim dos operands i tal i com hem dit a l'inici del document, sempre la calcularem elevant el número gran al número petit.

OperationGCD

Classe que implementa la operació de màxim comú divisor.

OperationLCM

Classe que implementa la operació de mínim comú múltiple.

OperationEquality

Classe que implementa la operació de igualtat. Aquesta operació només tindrà un operand.

OperationMultiplication

Classe que implementa la operació de multiplicació.

OperationDivision

Classe que implementa la operació de divisió. Tindrà com a màxim dos operands i com al target de l'operació no hi ha decimals, serà el número gran / número petit.

OperationFactory

Classe que genera operacions.

ColorFactory

Classe que genera colors.

Score

Classe que representa la puntuació obtinguda per un usuari.

ModelController

Classe controladora de la capa de domini, conté mètodes per a les funcionalitats principals del sistema.

Relació classes i membres

Classe	Membre
package exceptions	
CannotCreateOperationException	Oriol
CannotLoadKenKenException	Oriol
CannotLoadScoresException	Oriol
CellAlreadyInGroupException	Oriol
CellHasNoGroupException	Oriol
EraseFixedValueException	Oriol
GroupCellsNotContiguousException	Oriol
GroupDoesNotExistException	Oriol
InvalidUsernameException	Oriol
NonIntegerResultException	Oriol
OperandsDoNotMatchException	Oriol
RewriteFixedPositionException	Oriol
TooManyOperandsException	Oriol
ValueOutOfBoundsException	Oriol
package model.color	
ColorFactory	Oriol
package model.kenken	
Cell	Oriol
Group	Oriol
KenKen	Oriol
KenKenGenerator	Oriol
KenKenProposer	Oriol
KenKenSolver	Oriol
package model.operations	
Operation	Oriol
OperationAddition	Oriol
OperationDivision	Oriol
OperationEquality	Dani
OperationFactory	Oriol

Classe	Membre
package model.operations (continuació...)	
OperationGCD	Oriol
OperationLCM	Oriol
OperationLimitedOperands	Oriol
OperationMultiplication	Oriol
OperationPower	Dani
OperationSubtraction	Oriol
package model.topologies	
Shape	Oriol
Topology	Oriol
package model	
ModelController	Oriol
Score	Oriol

Descripció d'estructures de dades i algorismes

Estructures de dades

Per a representar el KenKen s'ha decidit d'utilitzar tres classes a part del propi KenKen. Aquestes són: Cell, Group i Operation.

Operation

Aquesta última representa una operació i un objectiu (target). Per exemple, podem fer una Operation que sigui ("+", 11) que definiria una suma amb l'objectiu d'arribar a 11. Té mètodes per a calcular el resultat donat uns operands, i comprovar si aquest coincideix amb el target. Afegidament, hi ha una subclasse OperationLimitedOperands que permet definir un número d'operands necessaris per a l'operació. Com 2 en la resta o la divisió; o 1 en la igualtat o la potència.

Cell

Cell te atributs per guardar la fila i columna en la que es troba, el valor que conté, un boolean per determinar si aquest valor es fix o no. I finalment el grup al que pertany.

Group

El Group és qui coneix la seva operation, en un atribut. Així mateix, també conté una List<Cell> de tipus ArrayList amb les seves cel·les.

KenKen

Finalment el KenKen defineix un taulell que és una matriu de Cell, coneix la seva mida (size) en un atribut, i conté una List<Group> també de tipus ArrayList amb els seus grups.

Algorismes

Backtracking

L'algorisme més utilitzat en el projecte és el backtracking. S'utilitza tant en la resolució del KenKen (mètode `model.kenken.KenKenSolver.solve()`), com en la generació dels grups donada un topologia concreta (mètode `model.kenken.KenKenGenerator.generateGroups`).

El *backtracking* utilitzat al `model.kenken.KenKenSolver.solve()` consisteix a posar seqüencialment valors al KenKen que siguin vàlids amb les condicions de resolució donades, és a dir que en el moment que posem un valor comprovem que no estigui ni repetit ni en la fila ni en la columna i que sigui un candidat vàlid per resoldre l'equació donada pel grup. Llavors anem posant valors a les cel·les recursivament de manera seqüencial fins que arribem al final,

en el millor dels casos si haguéssim encertat tots, o fins que ens hi trobem a una cel·la on no hi ha cap valor vàlid possible. Aquí és quan reculem (to backtrack en anglès) a la cel·la on encara tenim la possibilitat de proposar altres valors vàlids. Quan arribem a l'última cel·la, si trobem un valor vàlid, l'algorisme es cridarà una altra vegada arribant al cas base que retorna cert finalitzant el recorregut i havent resolt el KenKen.

Llavors en el pitjor cas tenim un cost exponencial, ja que per cada cel·la mirem n possibilitats, les quals han de ser comprovades amb altres n possibilitats de la següent cel·la, la qual ha de comprovar els seus n valors amb la següent cel·la, etc. Així fins omplir totes les cel·les buides. Tenim doncs que sent n la mida del KenKen i m el nombre de cel·les buides el cas pitjor tindrà un cost temporal, en notació asimptòtica, de $O\,(n^m)$.

Aquest cas, però, és poc probable, ja que hi haurà molts valors que no compliran la condició per ser candidats vàlids. Això serà perquè el nombre directament no serà vàlid per resoldre l'equació del grup. I també perquè el nombre estarà repetit a la fila o columna, això serà més freqüent a mesura que el KenKen es va omplint. Llavors podem dir que mai, o gairebé mai, arribarem a provar n valors per les m cel·les, reduint notablement el cost temporal per casos normals.

Respecte a cost espacial, no utilitzem més espai que el que utilitza el KenKen ha ser resolt. Ja que anem modificant els valors a mesura que el resolem, si un valor no és vàlid esborrem el valor de la cel·la per posar-ne un altre més endavant i evitar errors de comprovacions.

Per fer el *backtracking* usat al mètode `model.kenken.KenKenGenerator.generateGroups` creem un una matriu `groupMap` de la mateixa mida que el KenKeni anem creant grups d'acord amb la topologia donada. Crear grups es fa posant el nombre del grup al `groupMap` a les cel·les corresponents que ocuparia la topologia del dit grup. `groupMap` comença sent una matriu amb 0 i anem afegint nombres segons creem grups. Primer grup seria el número 1, segon grup número 2, etc.

Si la topologia es valida, és a dir que encaixa dins del KenKen amb les topologies posades anteriorment i no surt fora del KenKen, continua amb la pròxima cel·la buida. És a dir, cridem a la funció recursivament per considerar la següent cel·la.

Si la topologia no és vàlida anem provant girant la topologia fins que encaixi, si girant-la de les quatre maneres no encaixa tornem enrere per provar distintes orientacions de les topologies posades anteriorment, si és que n'hi ha. En aquest cas haurem d'eliminar els grups per tal de tenir el `groupMap` clar, i poder fer noves comprovacions.

La recursió s'acaba quan arribem al cas base, quan estem a l'última columna i fila i cridem altra vegada a la funció, llavors es comprova que ja no hi ha més cel·les per recórrer i retornem cert si totes les cel·les a `groupMap` tenen un grup assignat, és a dir són diferents de 0. En altre cas retorna fals, el que vol dir que haurem de recular i tornar a provar si és que és hi ha una

solució. És possible que no es trobi solució amb la topologia donada i en la mida del kenken, en aquest cas saltarà una excepció.

El cost temporal d'aquest algorisme depèn molt de la topologia donada, a mes de la mida del KenKen. Amb tantes variables es difícil de calcular el cost que tindria generar un KenKen sense saber concretament la seva mida i les topologies a utilitzar. El que sí podem saber es que, sent n la mida del KenKen, com a mínim el cost serà de $O\ (n^2)$, que es tractaria del millor cas, on no hi ha cap *backtracking* i totes les topologies encaixen a la primera. I d'altra banda també sabem que el cost no pot ser major que $O\ (n^n)$, i que gairebé en cap cas arribarà a ser aquest. Perquè amb les topologies que contenen més d'una cel·la no hi haurà possibilitat de recursivitat a totes les cel·les per que ja estan ocupades per les mateixes topologies. I amb les topologies amb només una cel·la el cost sera sempre $O\ (n^2)$.

Respecte al cost espacial com creem la matriu `groupMap` amb la mida del KenKen per guardar els grups que després traslladem al mateix KenKen. El cost es de $O\ (n^2)$, on n es la mida del KenKen.

Ranking

Per tal de no haver d'endreçar el ranking cada vegada que es vol consultar, el fitxer es guarda ja de manera endreçada, comprovant cada inserció que fa i fent ús del mètode `List.add(int index, E element)`.

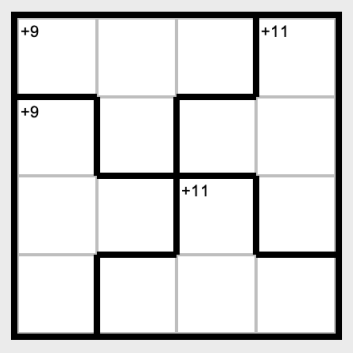
Format estàndard de fitxers KenKen

Arxius `.kenken`

El format requereix d'uns identificadors enters per a cada operació, aquests són:

```
1     +     suma
2     -     resta (2 operands)
3     *     producte
4     /     divisió (2 operands)
5     gcd   màxim comú divisor
6     lcm   mínim comú múltiple
7     ^     potència (2 operands)
8     =     igualtat (1 operand)
```

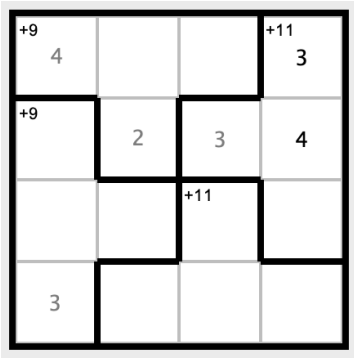
Per tal de poder implementar la funcionalitat de guardar i carregar una partida, s'ha modificat el format estàndard de fitxers. Tanmateix, la modificació és opcional, és a dir, només serveix per a afegir-hi més informació i en cap cas queden inutilitzats els arxius el format estàndard proporcionat en l'enunciat.



L'estructura proporcionada representaria el kenken de la imatge amb el següent fitxer:

```
4 4
1 9 4 1 1 1 2 1 3 2 2
1 9 4 2 1 3 1 3 2 4 1
1 11 4 3 3 4 2 4 3 4 4
1 11 4 1 4 2 3 2 4 3 4
```

Tanmateix, hom pot voler establir-hi uns valors fixes (ja proporcionats i inmodificables) així com els valors ja escrits per l'usuari en un moment de la solució.



En cas de voler representar això (on els valors en negre són els valors fixes, i en gris els proposats per l'usuari) el fitxer hauria de ser:

```
4 4
1 9 4 1 1 1 2 1 3 2 2
1 9 4 2 1 3 1 3 2 4 1
1 11 4 3 3 4 2 4 3 4 4
1 11 4 1 4 2 3 2 4 3 4
=====
1 4 3 !
2 4 4 !
1 1 4
2 2 2
2 3 3
4 1 3
```

La línia de cinc iguals marca la separació, de nou insistim que el sistema funciona perfectament sense aquesta part, així doncs els fitxers "clàssics" no modificats segueixen sent totalment funcionals.

Després cada línia marca un valor. Hi trobem, la fila i la columna de la cel·la i just després el valor. Opcionalment s'hi pot afegir el signe d'exclamació per a indicar que és un valor fixe.

Afegidament s'han creat unes extensions extres:

Arxius `.kenken_game`

Contenen partides del KenKen, segueixen el format ampliat especificat anteriorment. És l'extensió que s'utilitza quan es guarda una partida o es carrega una partida guardada.

Arxius `.kenken_scores`

Només n'hi ha un, l'arxiu `scores.kenken_scores`, conté el ranking ja ordenat de les puntuacions obtingudes. El seu format és molt senzill:

```
usernameA scoreA
usernameB scoreB
usernameC scoreC
```

Per a que funcioni correctament, els noms d'usuaris no poden contenir espais.

Arxius `.values`

Els arxius `.values` son uns arxius que ens ajuden a testejar els KenKen de manera més ràpida, carregant-los de manera que l'usuari no ha de posar tots el moviments manualment. Aquests fitxers contenen una seqüència de números separats per espais.

```
4 1 3 2
1 4 2 3
2 3 4 1
3 2 1 4
```

Per una millor llegibilitat és recomanable separar el valors per files, justament igual que a un KenKen. En la lectura d'aquests fitxers els valors s'introdueixen al KenKen sequencialment, d'esquerra a dreta començant per dalt, fins que els valors s'acabin o fins que s'hagi arribat a la mida del KenKen a les files i columnes, en aquest cas la resta de valors seran ignorats.