

**Projecte de PROP:**

**Kakuro/Cross sums**

**Primera entrega**

Tardor 2020

**Grup 23 Alias 74**

Sergi Berdor Jiménez

Marco Madalin Farcas Vinter

Gerard Caravaca Iáñez

Zhongkai Dai

ÍNDEX

[**1.Casos d’ús**](#_nxc1ffvpphvt) **2**

[1.1.Diagrama de casos d’ús](#_fbmzl84f7k5r) 2

[1.2.Descripció de casos d’ús](#_ui66t2onixdm) 3

[1.2.1.Gestió de perfils](#_gdoq65oqjh0x) 3

[1.2.2.Gestió de kakuros](#_cjrxm3wzyf6z) 4

[1.2.3.Gestió de partida](#_hqxjbgpw3wzx) 6

[1.2.4.Consultes](#_ao1o49cqpsu8) 8

[1.2.5.Sortir](#_313kqt80aqar) 9

[**2.Model conceptual de dades**](#_fyga78vo2bq2) **10**

[2.1.Diagrama UML estàtic complet](#_runl8bms9xdf) 10

[2.1.Especificació breu](#_dm5mtuswjnae) 10

[**3.Descripció del projecte**](#_uci6speaoiwy) **11**

[3.1.Estructura](#_pa3p6ppjcjde) 11

[3.1.1.Cell i derivades](#_d1otphkhbsa6) 11

[3.1.2.Combos](#_dbrpey98p3v5) 11

[3.1.3.Kakuro](#_cipu6mefstb8) 12

[3.1.4.Partida](#_ugiul6llsbz3) 12

[3.1.5.Controladors](#_1w62mhlekpq2) 12

[3.2.Algorisme de resolució/validació](#_uk39gq8froso) 13

[3.2.1.Part 1: Preprocessat de valors i dades](#_yske3tx5198q) 13

[3.2.1.Part 2: Exploració d’estats](#_gy6pzqka0xid) 14

[3.2.Algorisme de generació](#_t53dlglfnucc) 16

[3.2.1.Part 1: Generació patró](#_t9j98cs52n9z) 16

[3.2.2.Part 2: Generació de sumes](#_9dl02i3kvoiv) 16

[**4.Repartiment de la càrrega de treball**](#_vqzvrk1cgx27) **18**

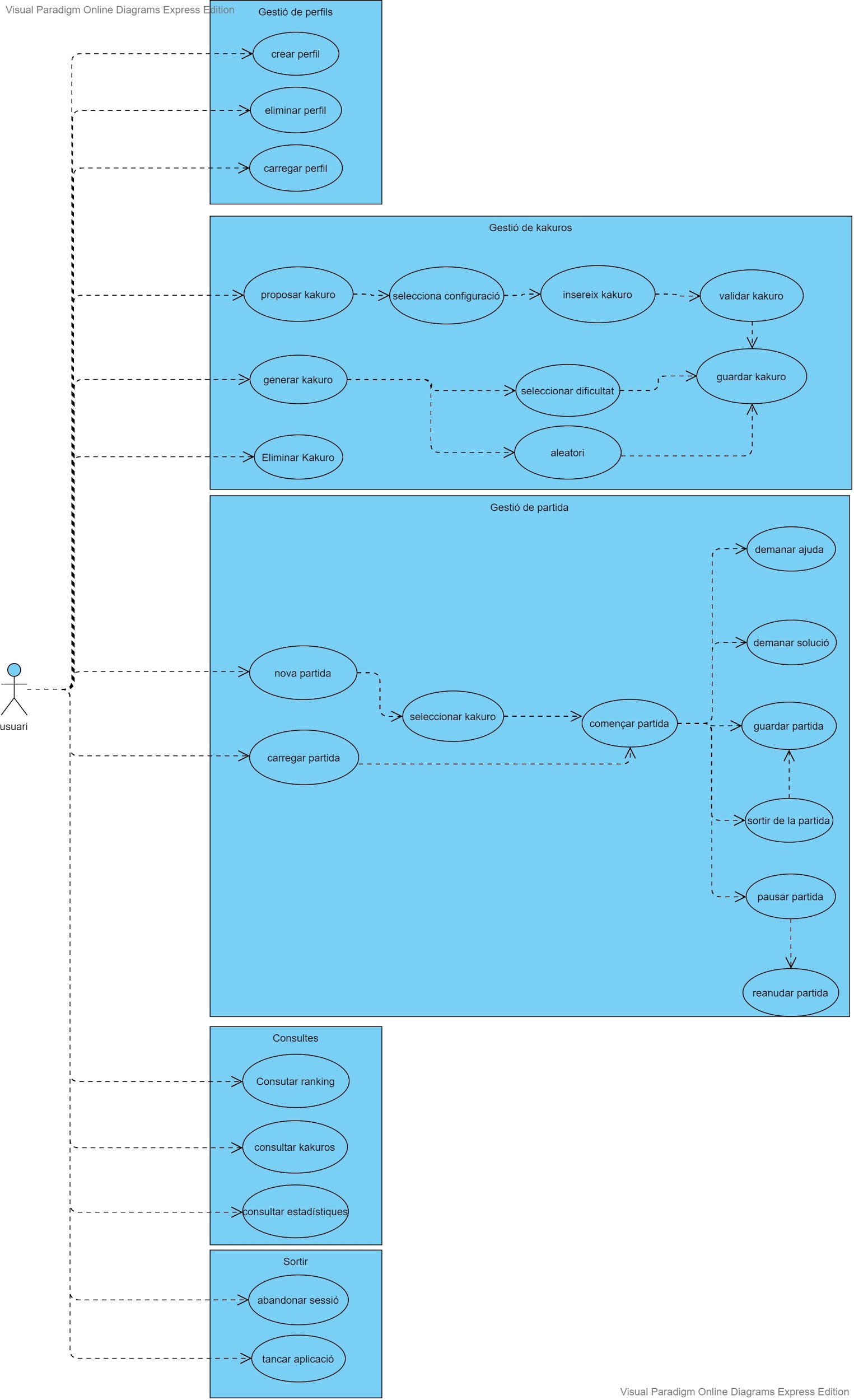
[**5.Llibreries externes utilitzades**](#_5ug1k04ol4n) **19**

[**6.Consideracions i comentaris**](#_q9vbmkx8j7an) **20**

# 1.Casos d’ús

## 1.1.Diagrama de casos d’ús

El diagrama es pot consultar amb més detall en la carpeta DOCS.

****

## 1.2.Descripció de casos d’ús

### 1.2.1.Gestió de perfils

**Nom:** Crear perfil

**Actor:** Usuari

**Comportament:**

1. L’usuari informa del nom d’usuari, el password i confirma el password
2. El sistema valida els valors i crea el nou perfil amb el nom

**Errors possibles i cursos alternatius:**

* Si ja existeix un perfil amb el nom indicat, el sistema informa de l’error
* Si el password i la seva confirmació no són iguals, el sistema informa de l’error
* El password no compleix els requisits de seguretat, el sistema informa de l’error

**Nom:** Eliminar perfil

**Actor:** Usuari

**Comportament:**

1. L’usuari ha d’estar iniciat al sistema.
2. L’usuari selecciona eliminar perfil.
3. El sistema demana la password i la valida.
4. El sistema pregunta al usuari i confirma que s’ha eliminat l’usuari correctament

**Errors possibles i cursos alternatius:**

* Si el password no és correcte, el sistema informa de l’error

**Nom:** Carregar perfil

**Actor:** Usuari

**Comportament:**

1. L’usuari informa del nom d’usuari, el password
2. El sistema valida els valors i carrega el perfil

**Errors possibles i cursos alternatius:**

* Si no existeix un perfil amb el nom indicat, el sistema informa de l’error
* Si el password és incorrecte, el sistema informa de l’error

### 1.2.2.Gestió de kakuros

**Nom:** Proposar kakuro

**Actor:** Usuari

**Comportament:**

1. L’usuari vol proposar un kakuro

**Nom:** Selecciona configuració

**Actor:** Usuari

**Comportament:**

1. L’usuari insereix les dimensions del kakuro.

**Errors possibles i cursos alternatius:**

* Format d’entrada incorrecte, el sistema informarà de l’error.

**Nom:** Insereix kakuro

**Actor:** Usuari

**Comportament:**

1. L’usuari insereix el kakuro.

**Errors possibles i cursos alternatius:**

* Format d’entrada incorrecte, el sistema informarà de l’error.

**Nom:** Validar kakuro

**Actor:** Sistema

**Comportament:**

1. El sistema valida si el kakuro inserit per l’usuari és una kakuro correcte.

**Errors possibles i cursos alternatius:**

* El kakuro no té solució , el sistema informarà de l’error i tornarà a executar la funcionalitat anterior.
* El kakuro té diverses solucions, el sistema informarà de l’error i tornarà a executar la funcionalitat anterior.

**Nom:** Generar kakuro

**Actor:** Usuari

**Comportament:**

1. L’usuari vol que el sistema li generi un kakuro
2. El sistema genera un kakruo vàlid
3. El sistema demana un nom pel kakuro i el guarda

**Conté: Seleccionar dificultat, aleatori**

**Nom:** Seleccionar dificultat

**Actor:** Usuari

**Comportament:**

1. L’usuari selecciona dificultat i mida del kakuro.
2. El sistema crea un kakuro de la mateixa dificultat i mida que ha seleccionat l’usuari.
3. El sistema mostrarà el kakuro i la dificultat.
4. El sistema demana un nom pel kakuro i el guarda

**Nom:** Aleatori

**Actor:** Usuari

**Comportament:**

1. L’usuari selecciona la mida del kakuro.
2. El sistema genera un kakruo vàlid.
3. El sistema demana un nom pel kakuro i el guarda

**Nom:** Guardar kakuro

**Actor:** Usuari

**Comportament:**

1. L’usuari insereix un kakuro vàlid.
2. El sistema valida el kakuro introduit.
3. L’usuari insereix un nom pel kakuro.
4. El sistema guarda el kakuro al repositori privat.

**Nom:** Eliminar kakuro

**Actor:** Usuari

**Comportament:**

1. El sistema mostra els noms dels kakuros al repositori privat de l’usuari.
2. L’usuari selecciona el kakuro que vol eliminar.
3. El sistema eliminar el kakuro del seu repositori privat.

### 1.2.3.Gestió de partida

**Nom:** Nova partida

**Actor:** Usuari

**Comportament:**

1. L’usuari comença una nova partida.

**Conté: seleccionar kakuro**

**Nom:** Seleccionar kakuro

**Actor:** Usuari

**Comportament:**

1. L’usuari selecciona un kakuro del seu repositori privat o del repositori global.

**Errors possibles i cursos alternatius:**

* L’usuari selecciona un kakuro que no existeix.

**Nom:** Carregar partida

**Actor:** Usuari

**Comportament:**

1. El sistema busca si l’usuari té una partida a mig jugar.
2. El sistema mostra a l’usuari l’estat de la partida.
3. L’usuari decideix si vol seguir jugant la partida o començar de nou.

**Errors possibles i cursos alternatius:**

* L’usuari no té partides a mig jugar, el sistema informarà a l’usuari de la situació.

**Nom:** Començar partida

**Actor:** Usuari

**Comportament:**

1. L’usuari insereix un kakuro.
2. El sistema valida el kakuro.
3. El sistema demana un nom pel kakuro i el guarda.
4. El sistema mostra el taulell de joc del kakuro que l’usuari ha inserit prèviament.

**Conté**: **Demanar ajuda, demanar solució, guardar partida, sortir de la partida, pausar partida**

**Nom:** Demanar ajuda

**Actor:** Usuari

**Comportament:**

1. L’usuari demana ajuda al sistema per resoldre una casella del kakuro.
2. El sistema retorna el valor correcte per aquesta casella

**Errors possibles i cursos alternatius:**

* L’usuari ha inserit una casella amb posició incorrecta.

**Nom:** Demanar solució

**Actor:** Usuari

**Comportament:**

1. L’usuari demana al sistema que resolgui el kakuro.
2. El sistema resol el kakuro i el mostra a l’usuari.

**Nom:** Guardar partida

**Actor:** Usuari

**Comportament:**

1. L’usuari vol guardar l’estat actual de la partida.
2. El sistema guarda l’estat actual de la partida.

**Errors possibles i cursos alternatius:**

* L’usuari ja té una partida guardada, el sistema informarà a l’usuari i preguntarà si vol esborrar la partida guardada i guardar l’actual.

**Nom:** Sortir de la partida

**Actor:** Usuari

**Comportament:**

1. L’usuari vol sortir de la partida.
2. El sistema pregunta a si vol guardar l’estat de la partida actual.
3. El sistema surt de la partida.

**Nom:** Pausar partida

**Actor:** Usuari

**Comportament:**

1. L’usuari vol pausar la partida.
2. El sistema pausa la partida.

**Nom:** Reanudar partida

**Actor:** Usuari

**Comportament:**

1. El sistema manté la partida en pausa.
2. L’usuari vol reanudar la partida.
3. El sistema reanuda la partida.

### 1.2.4.Consultes

**Nom:** Consultar ranking

**Actor:** Usuari

**Comportament:**

1. El sistema mostra a l’usuari les estadístiques dels millors jugadors del joc.

**Nom:** Consultar kakuros

**Actor:** Usuari

**Comportament:**

1. L’usuari decideix si vol veure els kakuros del repositori global o del seu privat.
2. El sistema mostra els kakuros.

**Nom:** Consultar estadístiques

**Actor:** Usuari

**Comportament:**

1. El sistema mostra les estadístiques del jugador.

### 1.2.5.Sortir

**Nom:** Abandonar sessió

**Actor:** Usuari

**Comportament:**

1. El sistema demana confirmació
2. El sistema tanca la sessió de l’usuari, si s’ha confirmat

**Nom:** Tancar aplicació

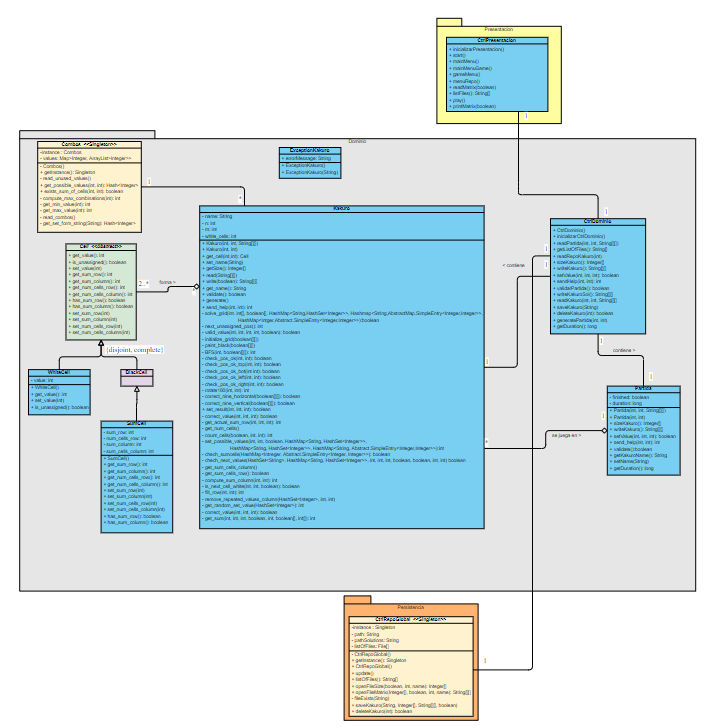
**Actor:** Usuari

**Comportament:**

1. El sistema demana confirmació
2. El sistema tanca l’aplicació completament, si s’ha confirmat

# 2.Model conceptual de dades

## 2.1.Diagrama UML estàtic complet

Igual que anteriorment el diagrama es pot consultar amb més detall a la carpeta DOCS.

## 2.1.Especificació breu

Les descripcions de les classes, atributs i mètodes estan especificades al doxygen generat dins la carpeta DOCS.

# 3.Descripció del projecte

## 3.1.Estructura

El programa s’estructura en tres parts diferents, tres capes, presentació, domini i persistència; el domini compta amb les classes que implementen les funcionalitats principals dessitjades, la presentació es centra en la interacció amb l’usuari i presentació de dades i la persistència s’encarrega del tractament amb els fitxers de dades.

### 3.1.1.Cell i derivades

Classe abstracta que s’utilitza per representar una casella del kakuro, es divideix en dues subclasses *WhiteCell* i *BlackCell* que representen caselles blanques i negres respectivament, la funcionalitat principal d’aquesta classe es permetre un acces uniforme a tots els tipus de casella.

* ***WhiteCell:*** representació d’una casella blanca, com a atribut conté el valor de la casella i com a mètodes el getter i setter del seu valor a més d’un altre per comprovar si hi ha un valor assignat o no.
* ***BlackCell:*** representació d’una casella negra no té cap altra funcionalitat més que permetre accedir a la seva classe filla *SumCell* que representa una casella amb una suma de fila i/o columna.
* ***SumCell:*** representació d’una casella de suma, compta amb els valors de la suma de fila i columna i el nombre de caselles blanques d’aquesta; de mètodes tenim els getters i setters dels atributs a més de les consultes per saber si la casella compta o no amb una suma de fila o columna.

### 3.1.2.Combos

Classe singleton que s’utilitza per guardar la informació de totes les combinacions possibles de caselles, sumes i valors; l'ús principal d’aquesta classe és ser una consultora d'informació quan es necessita informació sobre les combinacions de sumes caselles i valors. Els atributs son un llistat de caselles amb el seu rang de sumes on per cada suma d’un rang tenim totes les combinacions possibles de valors, i un altre llistat que conté valors que no s’utilitzen de cap manera en una suma donat cert nombre de caselles. Els mètodes ofereixen d’aquesta classe ofereixen accés a la informació mencionada anteriorment.

### 3.1.3.Kakuro

Classe core del programa ja que agrupa les funcionalitats principals, com es pot intuir representa el kakuro. *Kakuro* està format per un seguit de *Cell* i un *Combos*  més els atributs respectius que indiquen el seu nom, mida, nombre de caselles blanques i la solució actual del kakuro una vegada validat.

Els mètodes els podem agrupar en getters i setters d’atributs i la resta que s’utilitzen per implementar les funcionalitats principals del programa, generació, validació i jugar.

### 3.1.4.Partida

Classe per representar una partida de l'usuari formada principalment per un *Kakuro* més la informació de si esta finalitzada i el temps total d’aquesta, no duu a terme cap mena de treball autònom destacable més que crides a altres mètodes de la classe *Kakuro* per implementar les funcionalitats dessitjades; permet resoldre el kakuro manualment mitjançant els inputs de l’usuari (jugar) o resoldre automàticament.

Els mètodes es classifiquen en getters i setters d’atributs més la resta que duen a terme el seguit de crides a altres mètodes de la classe *Kakuro*.

### 3.1.5.Controladors

* ***CtrlDominio:*** Controlador del domini que permet la comunicació entre la capa de persistencia i presentació amb la resta de classes del domini.
* ***CtrlRepoGlobal:*** Controlador del repositori de kakuros, es singleton, actúa a petició de *CtrlDominio* per escriure, llegir o eliminar dades dels fitxers de text que salvaguarden kakuros.
* ***CtrlPresentacion:*** Controlador de la capa de presentació que implementa el sistema de menús per permetre la interacció de l’usuari amb el programa i controlar-la.

## 3.2.Algorisme de resolució/validació

L’algorisme segueix la tècnica de satisfacció de restricions mitjançant cerca exhaustiva, podem dividir-lo en dues parts. La primera part és un preprocessat de valors disponibles de les caselles més una recopilació d’informació que ens serà útil durant l'exploració del espai de solucions, la segona és aquesta mateixa exploració de tots els estats possibles fins explora-los tots o trobar-ne dos que siguin solucions.

### 3.2.1.Part 1: Preprocessat de valors i dades

La primera part abarca el cos de la funció *validate()*, *check\_sumcells()* i *set\_possible\_values()*. *Validate()* disposa de tres HashMaps on recopila la informació que usarem en la següent part, aquests són:

* ***acumulated\_sums****<Integer,AbstractMap.SimpleEntry<Integer,Integer>>*[[1]](#footnote-0): la clau del mapa és l’index d’una casella de suma (i\*#columnes+j) i com a valors tenim suma acumulada de la fila (key) i la suma acumulada de la columna (value).
* ***respective\_sum\_cell****<String,AbstractMap.SimpleEntry<Integer,Integer>>:* la clau del mapa representa el index d’una casella blanca com a string (“ij”) i com a valors tenim els index (i\*#columnes+j) de la seva casella de suma de fila (key) i columna (value) respectiva.
* ***possible\_values****<String,HashSet<Integer>>:* igual que anteriorment guardem com a clau el index d’una casella blanca com a string (“ij”) i com a valor tots els valors possibles que es poden assignar sota les restriccions de suma i columna inicials.

La funció *check\_sum\_cell()* inicialitza el mapa *acumulated\_sum* mitjançant un recorregut per la matriu de caselles, quan es troba una casella de suma afegim el seu índex al mapa amb un valor inicial <0,0>, a banda comptem les caselles blanques que venen a continuació en fila o columna o ambdues i comprovem que existeixi una suma en tantes caselles amb els combos, altrament el kakuro no té solució i acabem.

Tot seguit l'algoritme inicialitza els altres dos mapes mitjançant dos recorreguts un per fila i un per columna amb la crida a *set\_possible\_values()* quan es troba una casella de suma.

Es recorre el seguit de caselles blanques i al mapa respective\_sum\_cell fiquem la seva posició i la de la casella de suma <pos\_casella,<pos\_casella\_suma\_fila,0>> i després assignem el valors possibles d'aquesta casella <pos\_casella, (valors\_disponibles\_fila)> al mapa possible\_values; al segon recorregut per columnes fem el mateix procediment únicament que enlloc de assignar valors nous modifiquem els que teniem, al final dels recorreguts cada entrada de respective\_sum\_cell és de la forma <pos\_casella,<pos\_casella\_suma\_fila, pos\_casella\_suma\_columna>> i possible\_values de la forma <pos\_casella, (valors\_disponibles\_fila ⋂ valors\_disponibles\_columna)>. Arribats en aquest punt el algoritme ja ha obtingut tota la informació necessària per començar la exploració del espai d’estats fins trobar-ne dues o exhaurir totes els valors disponibles.

### 3.2.1.Part 2: Exploració d’estats

La següent part del algorisme abasta la funció *solve\_grid(), check\_next\_values()* i *valid\_value(),*  aquesta com hem mencionat a l'inici d'aquest apartat es basa en una cerca exhaustiva mitjançant una tècnica de satisfacció de restricions, a més també implementar mecanismes de poda i forward checking per evitar explorar estats no fructifers.

L’algorisme comença buscant la següent casella blanca ja que fa un recorregut per files de la matriu mitjançant la cerca, una vegada trobada aquesta casella s'extreu la informació relacionada aquesta casella dels paràmetres, aquesta informació és:

* HashSet<Integer> ***values****:* conjunt de valors possibles sobre els quals iterarem en aquest nivell de la recursivitat.
* HashSet<String> ***cells\_update\_row***, ***cells\_update\_col*:** conjunts on el seus valors són els índex en string de caselles blanques (“ij”) que haurem de tractar després de fer comprovacions de validesa per assegurar que no modifiquen les dades dels nivells superiors de la recursivitat.
* AbstractMap.SimpleEntry<Integer,Integer> ***sum\_act\_row***, ***sum\_act\_column***: substituts de Pair que guardaran la suma acumulada en aquest nivell de la recursivitat en fila i columna respectivament, de *sum\_act\_row* nomes farem servir la clau mentre que de *sum\_act\_column* nomes el value ja que recordem que son entrades del mapa *acumulated\_sums.*
* AbstractMap.SimpleEntry<Integer,Integer> ***cells*:** parell de enters on el primer, la clau, és l'índex de la casella de suma de fila i el segon, el valor, és la posició de la casella de suma de columna; aquests valors els fem servir per poder accedir al mapa de *acumulated\_sums* i obtenir la suma acumulada, a banda per obtenir la suma total de la fila i columna per comprovar les restricions.

La funció de *valid\_value()* solament comprova que la suma actual en fila o columna sigui menor a la total si no estem al final de la fila, altrament comprova que sigui igual. La funció *check\_next\_values()* és la encarregada de fer la poda i fer el forward checking, en un recorregut sobre les següents caselles blanques en fila o columna eliminem el valor que hem assignat a la casella actual i comprovem que encara hi queden valors per assignar, si ens quedem sense valors possibles el valor que hem triat per la casella actual ens portaria a una rama de la recursivitat que no tindria solució per tant no la volem explorar, a mesura que anem modificant valors possibles de caselles hem de tenir constància de quines caselles hem modificat per tant afegim el seu índex al paràmetre *cells\_update*, a banda també guardarem el conjunt de valors disponibles de totes les caselles que visitarem per fer una comprovació final de validesa, una vegada eliminat el valor assignat a la casella principal dels disponibles de les visitades mirem si la suma màxima de valors disponibles en les següents caselles es menor que la necessària i si la suma mínima és major que la necessària, si es dona qualsevol dels casos estarem hem de aborat la exploració per aquesta rama ja que mai podem arribar a una solució, per exemple: si tenim 5 caselles més per assignar, una suma actumulada de 8 i una suma total de 35, i entre totes les cinc tenim de disponibles els valors {1,2,3,5,6,7} la suma màxima seria 7+6+5+3+2 = 23, 23+8 = 31, per tant de cap forma hi haurà una solució, el mateix passa si la suma mínima es superior a la necessària, ergo podem fer una poda d'aquesta rama de l'arbre d’exploració. Si es compleixen les restricions anteriors el algorime assigna el valor a la casella actual, actualitza les sumes actuals i s'endinsa en la exploració, si ara la següent posició d’una casella es igual a la mida de la matriu voldrà dir que hem explorat totes les caselles i hem arribat a una solució, si és el cas s’incrementa el contador de solucions, es marca que hem trobat una solució per no guardar-ne d’altres, retornem fals i el algoritme segueix fins que troba un altra solució o arriba un punt on no queden valors possibles a cada casella.

## 3.2.Algorisme de generació

L’algorisme de generació també el podem separar en dues parts diferenciables, una que es dedica al pintat de caselles negres per generar un patró per al kakuro, i l’altre que genera sumes per cada fila i columna amb caselles blanques, tot seguit d’una validació del kakuro per comprovar que té solució ùnica; aquest procés es repeteix fins que no es trobi un kakuro amb solució ùnica.

### 3.2.1.Part 1: Generació patró

A la primera part es genera un nou kakuro on a la primera fila i a la primera columna només hi ha caselles negres i totes la resta de caselles són blanques. Després genera posicions aleatòries dins del kakuro, si la posició és correcta, la casella en aquesta posició es converteix en negra i també la casella simètrica girant 180º aixì assegurem que el patró final és simètric.

Per comprovar que una posició sigui correcta fem servir BFS per comprovar que totes les caselles blanques estan connectades i un mètode *check\_pos\_ok()* per evitar que es generin el patró següent: NEGRA BLANCA NEGRA ja que implicaria una suma trivial i es preferible evitar-ho.

Si no es generen prou caselles negres tenim dos mètodes *correct\_nine\_horitzontal()* i *correct\_nine\_vertical()*, que corregeixen posant caselles negres quan es troben que hi ha més de 9 caselles blanques seguides.

### 3.2.2.Part 2: Generació de sumes

La segona part es basa en dos recorreguts de la matriu assignat valors aleatoris no repetits en suma i columna i finalment assignat la suma total d’aquesta a la *SumCell* respectiva.

La primera passada es duu a terme mitjançant la funció *generate\_sum\_cells\_row()*  la qual s’escampa la matriu fins que troba una casella negra seguida d’una blanca, fet que indica que hem de convertir-la en una de suma, seguidament assigna valors aleatoris no repetits a les cel·les següents i va acumulant la seva suma fins que s’arriba a una casella que no es blanca, si en algun punt del recorregut no podem asignar valor a una casella no seguim amb el recorregut i retornem de la funció per tornar a generar noves sumes fins un valor establer max, si max es sobrepassa es torna a la part 1 del algorisme.

Si hem aconseguit generar les caselles de suma de fila s’invoca la funció *generate\_sum\_cells\_row()* que te el mateix comportaemnt que la anterior, únicament que aquesta només acumula la suma de cada casella blanca per després assignar-la a la de suma respectiva; seguidament es valida el kakuro, si te més d’una solució es repeteix la part 2 de l’algorisme fins no sobrepassar max, i si es sobrepassa max es retorna a la part 1.

# 4.Repartiment de la càrrega de treball

Sergi Berdor Jimenez: CtrlPresentacion, CtrlDominio, CtrlRepoGlobal

Marco Madalin Farcas Vinter: Kakuro, Combos, Cell, SumCell, BlackCell, WhiteCell

Gerard Caravaca Ibàñez: Drivers, JUnits

Zhongkai Dai: Partida, ExcepcioKakuro, Doxygen

# 

# 5.Llibreries externes utilitzades

En aquesta entrega no hem fet servir cap llibreria externa a part de junit-4.13.1.

# 6.Consideracions i comentaris

Aquest apartat el dediquem a comentar certs aspectes del projecte a banda de futurs canvis que volem duu a terme.

Actualment el algorisme de generació és molt poc eficient per kakuros de mida superior a 7x7 ja que ha de provar valors de forma aleatoria fins plenar la matriu de *Cells* i validar, el fet de assignar valors aleatoris i esperar generar un kakuro únic es poc probable per a mides grans ja que sense control es generarien sumes ambigües que poden tenir diversos valors i per tant el mètode de validar el qualificaria de invàlid; varem provar de usar un mètode més determinista utilitzant la classe combos mitjançant interseccions de valors alts amb baixos però no va tenir èxit tot i haver-li dedicat una quantitat considerable de temps, per falta de temps varem optar per aquesta versió, no obstant si que genera kakuros amb més d’una solució de forma ràpida fins mides 15x15/17x17; tot i així intentarem revisar la idea inicial de generació mitjançant interseccions ja que creiem que te potencial.

Un altra consideració és el estat actual de la classe kakuro, tenim previst separar-la en altres dues possiblement solver, generator i kakuro ja que actualment està tot molt aglutinat i pot arribar a ser una mica confús tenir tot el codi de les funcionalitats principal en la mateixa classe.

Per últim tenim pensat revisar la eficiencia del algorisme solucionador/validador ja que pensem que podriem aglutinar en un sol recorregut tota la extracció d'informació per després explorar el espai de estats però respecte a la part d’exploració de moment no se'ns acudeixen més millores per aplicar.

# 

1. AbstractMap.SimpleEntry és una forma de replicar Pair en Java [↑](#footnote-ref-0)