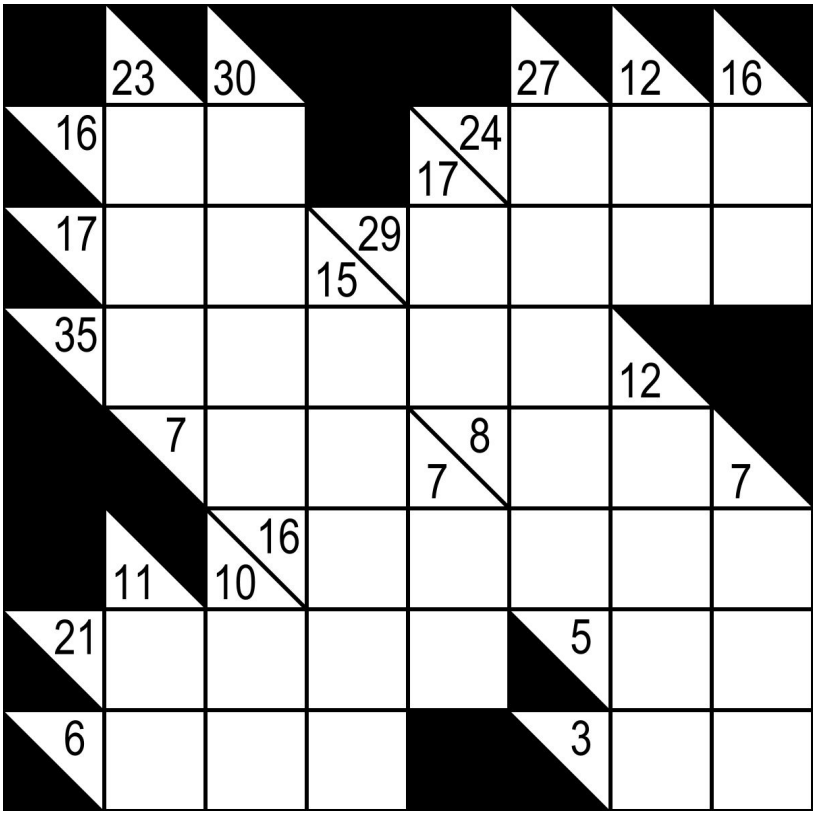
Kakuro



Projecte de PROP

Quadrimestre de tardor curs 20/21

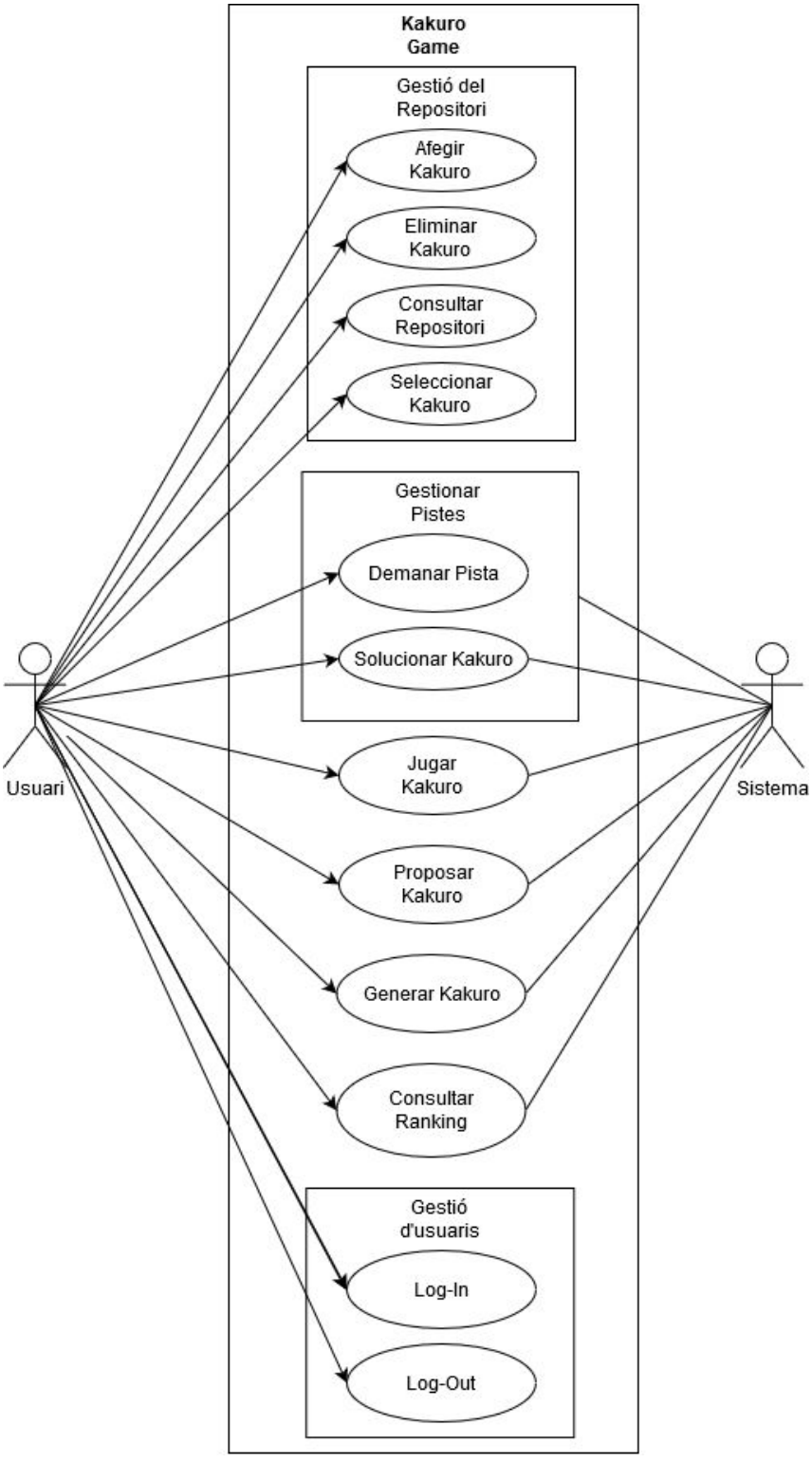
Oscar Polonio Valverde @oscar.polonio
Marcel Urpí Bricollé @marcel.urpi
Pol Gálvez Soriano @pol.galvez
Ferran Mateu Berga @ferran.mateu.berga

subgrup-prop2.2 **v1.0**

Index

1.Definició dels casos d'ús

1.1 Diagrama casos d’ús.



1.2 Descripció casos d'ús

1. **Afegir kakuro al repositori**
   Actors: Client
   Execució en escenaris d'èxit:
   L'usuari té un kakuro seleccionat que s'ha generat o proposat i escull l'opció afegir a repositori. El kakuro queda guardat al repositori.
   Execució en escenaris alternatius:
   L'usuari té un kakuro seleccionat i escull l'opció afegir a repositori ja existeix un kakuro igual al repositori. Es mostra un text d'error amb el missatge corresponent cap a l'usuari.

2. **Eliminar kakuro del repositori**
   Actors: Client
   Execució en escenaris d'èxit:
   L'usuari té un kakuro d'un repositori seleccionat i escull l'opció eliminar del repositori. El kakuro queda eliminat del repositori.
   Execució en escenaris alternatius:
   No n'hi ha.

3. **Consultar repositori**
   Actors: Client
   Execució en escenaris d'èxit:
   L'usuari té un kakuro d'un repositori seleccionat i escull l'opció eliminar del repositori. El kakuro queda eliminat del repositori.
   Execució en escenaris alternatius:
   No n'hi ha.

4. **Seleccionar kakuro del repositori**
   Actors: Client
   Execució en escenaris d'èxit: El client pot seleccionar un kakuro emmagatzemat en els repositoris navegant per aquests i filtrant per dificultat, o bé, anar a generar-ne un automàticament.
   Execució en escenaris alternatius:
   No n'hi ha.

5. **Demanar Pista**
   Actors: Client, Sistema
   Execució en escenaris d'èxit:
   L'usuari està resolent un kakuro i selecciona l'opció demanar pista. Es podran escollir diferents pistes que dependran de la dificultat.
   Execució en escenaris alternatius:
   Màxim nombre de pistes utilitzades.

6. **Solucionar kakuro**
   Actors: Client, Sistema
   Execució en escenaris d'èxit:
   L'usuari està resolent un kakuro i selecciona l'opció solucionar kakuro. El sistema resol el kakuro mostrant-lo per pantalla i acaba la partida.

   Execució en escenaris alternatius:
   No n'hi ha.

7. **Jugar kakuro**
   Actors: Client
   Execució en escenaris d'èxit:
   Comença el joc i l'usuari va fent jugades que el sistema va monitoritzant
   Execució en escenaris alternatius:
   Si no hi ha kakuro seleccionat, al intentar jugar s'anirà a la pantalla de seleccionar kakuro

8. **Proposar kakuro**
   Actors: Client
   Execució en escenaris d'èxit: El client introdueix el tamany del Kakuro, el color de les cel·les i el seu valor. El Kakuro proposat te una única solució.
   Execució en escenaris alternatius:
   El Kakuro proposat no té solució o en té més d'una.

9. **Generar Kakuro**

   Actors: Client

   Execució en escenari d'èxit: El client pot generar Kakuro a partir d'unes limitacions donandes per l'usuari

   Execució cas alternatiu:

   No existeix cap Kakuro vàlid amb les limitacions donades.

10. **Consultar rànking**

    Actors: Client

    Execució en escenaris d'èxit: L'usuari pot consultar els rànkings per les diferents dificultats pels diferents usuaris.

    Execució en escenaris alternatius:

    No n'hi ha.

11. **Log-In**

    Actors: Client

    Execució en escenaris d'èxit: L'usuari pot introduir un nickname i utilitzarlo per a tenir un repositori de Kakuros personal així com deixar una puntuació al ranking.

    Execució en escenaris alternatius:

    No n'hi ha.

12. **Log-Out**

    Actors: Client

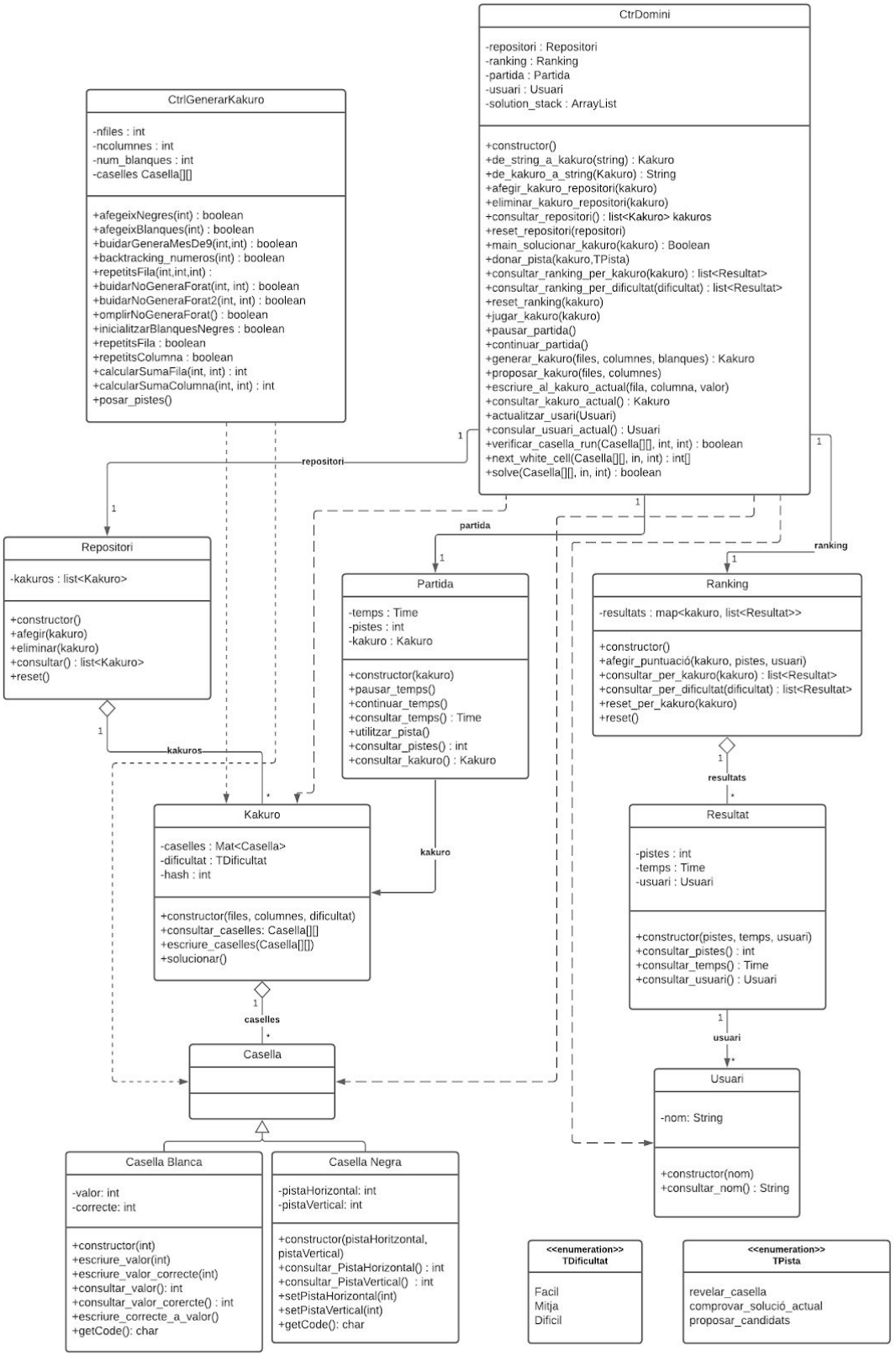
    Execució en escenaris d'èxit: L'usuari pot deixar d'utilitzar un cert nickname per accedir amb un d'altre.

    Execució en escenaris alternatius:

    No n'hi ha.

2. Model conceptual de dades

2.1 Diagrama UML



2.2 Especificació detallada

1. **Casella**
    a. Classe abstracta sense cap mètode, pare de CasellaNegra i CasellaBlanca.
    b. No té atributs propis.
    c. No te mètodes.
    d. No té associacions.

2. **CasellaBlanca**
    a. Explicació: Classe que hereda de Casella. Representa una casella blanca d'un Kakuro.
    b. Atributs:
        i. valor: 0 per defecte. És el valor que l'usuari introdueix per a la casella blanca.
        ii. correcte: 0 per defecte. És el valor que hauria de tenir la casella per ser correcte.
    c. Mètodes:
        i. Constructores:
            1. Per defecte: sense paràmetres, retorna una casella blanca amb els valors per defecte.
            2. Correcta: rep el número correcte com a paràmetre, retorna una casella blanca amb el valor correcte guardat.
        ii. consultar_valor(): No té paràmetres i retorna el valor que l'usuari ha escrit a la casella, si no ho ha fet retorna el valor per defecte.
        iii. escriure_valor(int valor): Rep el paràmetre valor que indica el valor que l'usuari ha introduït. El resultat és que a la casella se li assigna el valor passat com a paràmetre. No retorna res.
        iv. escriure_valor_correcte(int correcte): Escriu a correcte el nou valor correcte que rep com a paràmetre.
        v. consultar_valor_correcte(): Retorna el valor correcte, que es troba en l'atribut que es diu correcte.
        vi. escriure_correcte_a_valor(): No rep paràmetres. Copia el número correcte a valor. Serveix o bé per donar pistes o per mostrar la solució

3. **CasellaNegra**
    a. Explicació: Classe que hereda de Casella. Representa una casella negra d'un Kakuro.
    b. Atributs:
        i. pistaHorizontal: -1 per defecte. És el valor que han de sumar les caselles blanques a la dreta de la casella negra.
        ii. pistaVertical: -1 per defecte. És el valor que han de sumar les caselles blanques a sota de la casella negra
    c. Mètodes:
        i. Constructores:
            1. Per defecte: sense paràmetres, retorna una casella amb els valors per defecte.

2. Amb pistes: rep els valors pistaHorizontal i pistaVertical com a paràmetres. Retorna una casella amb els atributs inicialitzats com s'indica en els paràmetres.

ii. consultar_pistaHorizontal(): No té paràmetres i retorna el valor de pistaHorizontal de la casella.

iii. consultar_pistaVertical(): No té paràmetres i retorna el valor de pistaVertical de la casella.

iv. setPistaVertical(int pista): Rep el paràmetre pista i l'assigna a l'atribut pistaVertical de la casella.

v. setPistaHorizontal(int pista): Rep el paràmetre pista i l'assigna a l'atribut pistaHorizontal de la casella.

4. <u>CtrlDomini</u>

a. Explicació: Controlador de la capa de domini. S'encarrega de comunicar-se amb les capes de presentació i persistència. Conte funcions d'us general així com el solucionador de Kakuros. És la primera classe de la capa de domini en inicialitzar-se.

b. Atributs:

i. repositori: Valor per defecte és un repositori buit. S'utilitza per comunicar-se amb la capa de persistència.

ii. ranking: Valor per defecte és un ranking buit. Representa les puntuacions dels diferents usuaris.

iii. partida: Valor per defecte null. Representa tota la informació de la partida que està en procés.

iv. usuari: Valor per defecte null. Identifica l'usuari que està interactuant amb els sistema en aquell moment.

c. Mètodes:

i. de_string_a_kakuro(String string): Rep una string com a paràmetre que representa un kakuro. Retorna una instància de Kakuro amb el kakuro especificat en el paràmetre.

ii. de_kakuro_a_string(Kakuro kakuro): Rep un kakuro com a pàrametre. Retorna una String que representa el Kakuro del paràmetre amb el format correcte.

iii. generar_kakuro(int files, int columnes, int caselles_blanques): Rep 3 paràmetres que indiquen el número de files, columnes i caselles blanques respectivament que ha de tenir el kakuro que es vol generar. Retorna un Kakuro en blanc amb solució única preparat per jugar-lo.

iv. verificar_casella_run(Casella[][] kakuro, int f, int c): Donada una array 2D de caselles i una posició d'aquesta, retorna verdader si les runs en que participa la casella no incompleixen cap restricció (valor duplicat,suma massa alta, suma baixa amb tot omplert).

v. next_white_cell(Casella[][] kak, int i, int j): Donada una array 2D de caselles i una posició d'aquesta que representa des d'on comencem a buscar la següent casella, retorna dos enters que representen la posició de la següent casella blanca. Si s'arriba al final de l'array es retorna (-1,-1).

vi. solve(Casella[][] kakuro_curr_state, int i, int j):Donada una array 2D de caselles i una posició d'aquesta que representa la casella blanca que

s'esta resolent. Es proven tots els valors possibles i en cas de que no es saltin cap restricció es fa una crida recursiva, explorant així tot l'arbre de possible solucions del kakuro.

vii. copia_mat_caselles(Casella[][] c): Donada una array 2D de caselles retorna una copia d'aquesta.

viii. main_solucionar_kakuro(Kakuro kakuro): Fa la primera crida a solve(), retorna un boolean que indica si el kakuro té una única solució.

5. <u>CtrlGenerarKakuro</u>

a. Explicació: Sub controlador de la capa de domini. S'encarrega de generar un kakuro amb els paràmetres que rep.

b. Atributs:

i. nfiles: Valor per defecte null. Representa el número de files que ha de tenir el kakuro que es vol generar.

ii. ncolumnes: Valor per defecte null. Representa el número de columnes que ha de tenir el kakuro que es vol generar.

iii. num_blanques: Valor per defecte null. Representa el número de caselles blanques que ha de tenir el kakuro que es vol generar.

iv. caselles: Valor per defecte null. És una matriu de Casella que representa el tauler.

c. Mètodes:

i. backtracking_numeros(int i): Recorre la matriu de caselles des de la casella i (que és un index entre 0 i nfiles*ncolumnes) i que va provant valors que podrien anar a cada casella blanca del kakuro sense repeticions ni en files ni en columnes. Es un mètode recursiu.

ii. repetitsFila(int i, int j, int num): Comprova si algun dels números introduits abans que aquest en la fila (i) és igual que el que anem anem a colocar (num).

iii. repetitsColumna(int i, int j, int num): Comprova si algun dels números introduits abans que aquest en la columna (j) és igual que el que anem anem a colocar (num).

iv. posar_pistes(): omple les sumes de files i columnes calculant-les a partir dels números que ha generat el mètode backtracking_numeros.

v. calcularSumaFila(int i, int j): calcula la suma de la fila que comença a partir de la casella i que anirà com a pista en aquesta casella i, j.

vi. calcularSumaColumna(int i, int j): calcula la suma de la columna que comença a partir de la casella i que anirà com a pista en aquesta casella i, j.

vii. afegeixNegres(int num): Rep el número de caselles negres que cal afegir al tauler. Retorna cert i al acabar ha canviat tantes caselles blanques per negres com s'indica en l'atribut num.

viii. afegeixBlanques(int num): Rep el número de caselles blanques que cal afegir al tauler. Retorna cert si s'ha pogut fer i fals si s'ha generat un bucle infinit. Al acabar ha canviat tantes caselles negres per blanques com s'indica en l'atribut num.

ix. buidarGeneraMesDe9(int i, int j): Rep les coordenades d'una casella del tauler. Retorna cert si al canviar la casella negra [i][j] per una de

blanca no es generen files i columnes de més de 9 caselles. Sinó, retorna fals.

x. buidarNoGeneraForat(int i, int j): Rep les coordenades d'una casella del tauler. Retorna cert si al canviar la casella negra [i][j] per una de blanca no es genera cap espai d'una casella en les files i columnes a les que pertany. Sinó, retorna fals.

xi. buidarNoGeneraForat2(int i, int j): Rep les coordenades d'una casella del tauler. Retorna cert si al canviar la casella negra [i][j] i una de les seves 4 veïnes, per 2 de blanques no es genera cap espai d'una casella en les files i columnes a les que pertanyen. Sinó, retorna fals.

xii. omplirNoGeneraForat(int i, int j): Rep les coordenades d'una casella del tauler. Retorna cert si al canviar la casella blanca [i][j] per una de blanca no es genera cap espai d'una casella en les files i columnes a les que pertany. Sinó, retorna fals.

xiii. inicialitzar_Kakuro(): No rep paràmetres. Retorna un tauler que compleix amb el número de files, columnes i caselles blanques que té com a atributs la classe.

xiv. generar_kakuro(int files, int columnes, int caselles_blanques): Rep 3 paràmetres que indiquen el número de files, columnes i caselles blanques respectivament que ha de tenir el kakuro que es vol generar. Primerament genera la disposició de caselles blanques i negres. Després omple els números que seran solució del kakuro i finalment omple les pistes corresponents. Retorna un Kakuro en blanc amb solució única preparat per jugar-lo.

6. <u>Kakuro</u>

a. Explicació: Classe que representa un kakuro.
b. Atributs:
   i. caselles: Per defecte null. Matriu d'instàncies de la classe Casella que representa un tauler.
   ii. dificultat: Per defecte null. Valor que indica el nivell de dificultat del kakuro.
c. Mètodes:
   i. Constructora: Rep per paràmetre una matriu de Casella. Retorna un kakuro que com a caselles té el que se li passa com a argument.
   ii. consultar_caselles(): Sense paràmetres. Retorna l'atribut caselles.
   iii. escriure_caselles(Casella[][] caselles): Rep per paràmetre una matriu de Casella. sobreescriu l'atribut caselles del kakuro amb el que rep com a paràmetre.
   iv. consultar_dificultat(): retorna el valor de l'atribut dificultat.
   v. solucionar(): executa escriure_correcte_a_valor()per a cada casella del kakuro.

7. <u>Partida</u>

a. Explicació: Classe que emmagatzema les dades necessaries per controlar i recuperar una partida que s'està duent a terme.
   (NO IMPLEMENTAT)

8. **Pista**
    a. Explicació: Classe que representa les pistes que pot anar demanant l'usuari.
    (NO IMPLEMENTAT)

9. **Ranking**
    a. Explicació: Classe que manté un registre de les puntuacions obtingudes per cada usuari i dificultat.
    (NO IMPLEMENTAT)

10. **Repositori**
    a. Explicació: Classe que representa un repositori de kakuros.
    (NO IMPLEMENTAT)

11. **Resultat**
    a. Explicació: Classe que representa una entrada del ranking, amb el kakuro resolt i el temps i puntuació corresponents al kakuro resolt.
    (NO IMPLEMENTAT)

12. **Usuari**
    a. Explicació: Classe que permet gestionar els diferents usuaris que juguen kakuros.
    (NO IMPLEMENTAT)

3. Descripció de les estructures de dades i algorismes

Algorisme solucionador de kakuros:
S'atribueix per ordre un valor de l'1 al 9 a les caselles blanques (solver) una per una (next_white_cell) y comprova que es compleixen les restriccions (verificar_casella_run), fent backtracking en cas contrari.
Quan es troba una solució s'afegeix al solution stack, fent abans una copia (copia_mat_caselles)

Algorsime generador de kakuros:
Es comença generant un tauler que compleix les especificacions de files, columnes i nº de caselles blanques que es passen com a paràmetres. També, tenint en compte de no tenir files ni columnes més llargues de 9 i sense tenir espais d'una casella blanca. A partir d'aqui s'omplen les caselles blanques amb números intentant

Estructrures de dades utilitzades per representar el kakuro:
La nostra representació d'un kakuro en Java consta de una array d'arrays (array 2D) que té com a elements objectes de la classe casella. Cada casella pot ser blanca o negra, si es blanca té un atribut que es el valor i si es negra en pot tenir fins a dos que son les pistes.

4.Relació/llista de les classes implementades per cada membre del grup

Casella → Ferran
CasellaBlanca → Pol
CasellaNegra → Oscar

Kakuro →Marcel
CtrlDomini →Pol i Oscar
CtrlGenerarKakuro→Marcel i Ferran

DriverKakuro→ Marcel
DriverCasella→ Marcel
DriverCtrlDomini→ Pol i Oscar
DriverGenerarKakuro→ Marcel i ferran

5.Relació de llibreries externes utilitzades

No hem utilitzat cap llibreria externa.