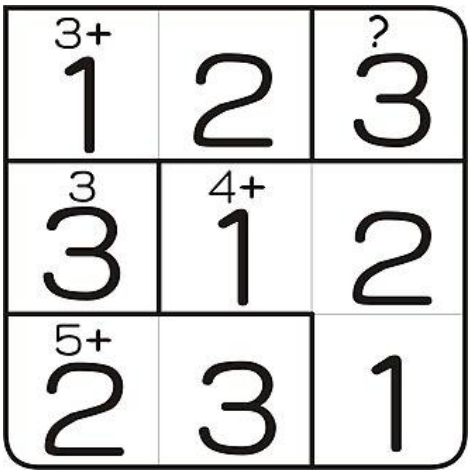


Kenken

Proyecto de Programación

QP 23-24

Versió 1.1

1ª Entrega

Equipo 13.1

Javier Parcerisas Nisa: javier.parcerisas@estudiantat.upc.edu

Romeu Esteve Casanovas: romeu.esteve@estudiantat.upc.edu

Feiyang Wang: feiyang.wang@estudiantat.upc.edu

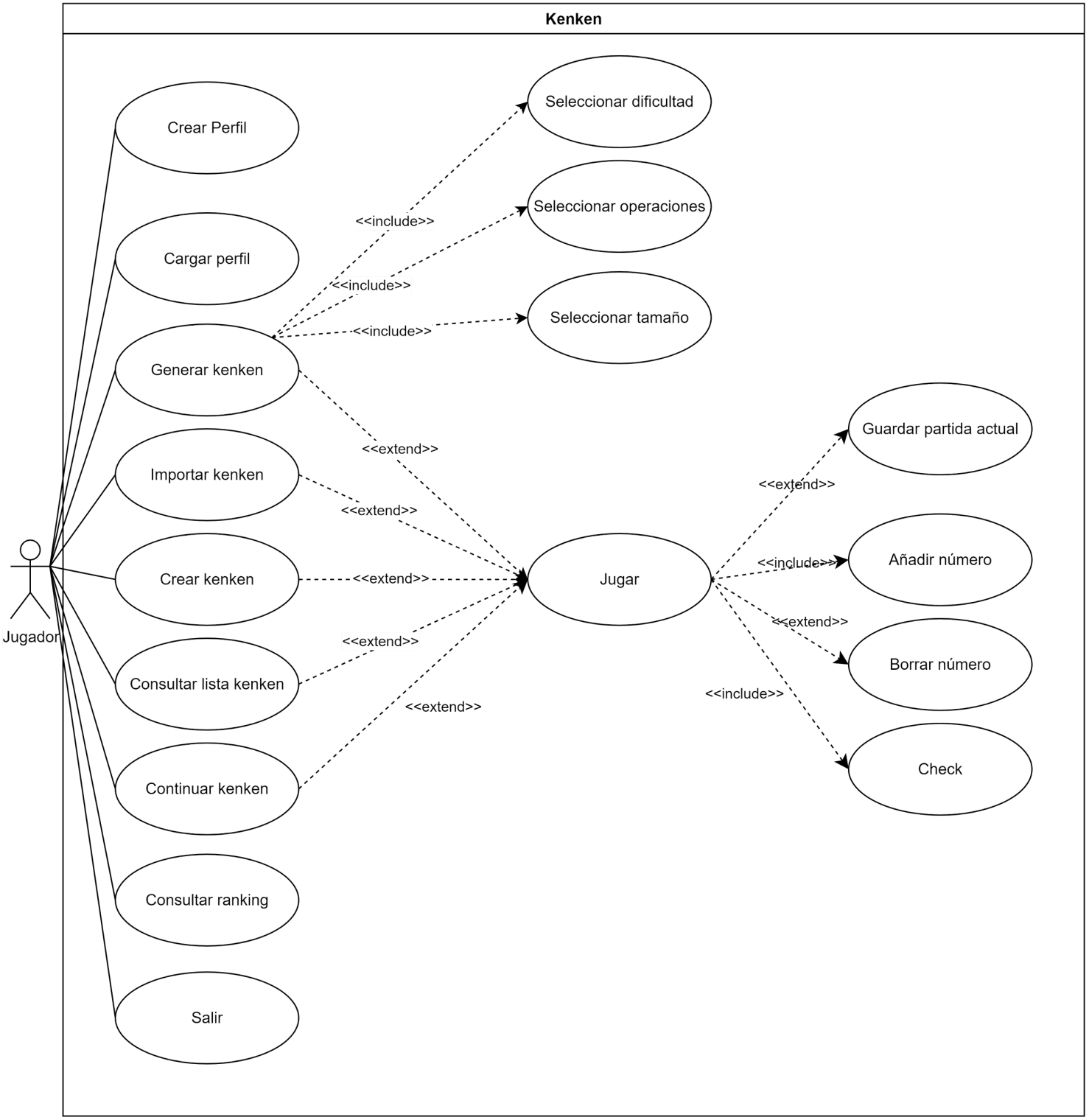
UNIVERSITAT POLITÈCNICA DE CATALUNYA BARCELONATECH

FIB

Índice:

1. Casos de uso

1.1 Diagrama de casos de uso



1.2 Descripción de casos de uso

Nombre: Crear Perfil
Actor: Usuario
Comportamiento:

1. El usuario indica al sistema su nickname, su contraseña y una confirmación de esta.
2. El sistema confirma los valores introducidos por el usuario y crea un nuevo perfil con el nickname y contraseña introducidos.

Errores posibles y cursos alternativos:

1. Si el nickname de usuario ya existe, el sistema informa del error.
2. Si las contraseñas no coinciden entre ellas, el sistema informa del error.

Nombre: Cargar Perfil
Actor: Usuario
Comportamiento:

1. El usuario introduce un nickname y una contraseña al sistema.
2. El sistema confirma los valores introducidos y carga el perfil del usuario junto a su partida guardada.

Errores posibles y cursos alternativos:

1. Si el usuario no existe, el sistema informa del error y recomienda crear un nuevo perfil.
2. Si la contraseña no es correcta, el sistema informará al usuario y pedirá que lo intente de nuevo.

Nombre: Consultar ranking
Actor: Usuario
Comportamiento:

1. El sistema lista los usuarios con las puntuaciones más altas de forma descendente.

Errores posibles y cursos alternativos:

1. Si no hay jugadores registrados en el sistema no se mostrará nada.

Nombre: Salir
Actor: Usuario
Comportamiento:

1. El sistema da una confirmación para salir del programa.
2. El sistema cierra la aplicación.

Errores posibles y cursos alternativos:

Nombre: Generar kenken
Actor: Usuario
Comportamiento:

1. El usuario indica el tamaño del kenken.
2. El usuario indica el tipo de operaciones que quiere en el kenken.
3. El usuario indica la dificultad que quiere en el kenken.
4. El sistema genera un kenken con solución con las características seleccionadas por el usuario y empieza la partida.

Errores posibles y cursos alternativos:

1. Si el usuario quiere descartar el kenken, puede salir al menú principal y crear uno de nuevo.
2. Si el usuario no ha seleccionado las 3 características el sistema no le dejará generar el kenken.

Nombre: Seleccionar Tamaño
Actor: Usuario
Comportamiento:

1. El sistema muestra al usuario los tamaños que puede utilizar.
2. El usuario escoge e indica al sistema el tamaño que quiere usar para el kenken.
3. El sistema utiliza el tamaño seleccionado para generar el kenken.

Errores posibles y cursos alternativos:

1. Si el usuario indica un tamaño menor a 3 o mayor a 9 el sistema le notificará de este error.

Nombre: Seleccionar Operaciones
Actor: Usuario
Comportamiento:

1. El sistema muestra al usuario las operaciones que puede utilizar.
2. El usuario escoge e indica al sistema las operaciones que quiere usar para el kenken.
3. El sistema utiliza las operaciones seleccionadas para generar el kenken.

Errores posibles y cursos alternativos:

Nombre: Seleccionar Dificultad
Actor: Usuario
Comportamiento:

1. El sistema muestra al usuario las dificultades que puede utilizar.
2. El usuario escoge e indica al sistema la dificultad que quiere usar para el kenken.
3. El sistema utiliza la dificultad seleccionada para generar el kenken.

Errores posibles y cursos alternativos:

Nombre: Crear kenken
Actor: Usuario
Comportamiento:

1. El usuario introduce el tamaño del kenken.
2. El sistema muestra un tablero con el tamaño indicado.
3. El usuario selecciona casillas e indica el tipo de operación que realizan entre ellas.
4. El sistema verifica la información introducida por el usuario.
5. El sistema indica si el kenken tiene solución.
6. El usuario puede escoger entre:
    a. El usuario guarda el kenken.
    b. El usuario indica un nombre para el kenken que no se repita con otro kenken.
    c. El sistema guarda el kenken.
    d. El usuario descarta el kenken.

Errores posibles y cursos alternativos:

1. Si el usuario indica una división o módulo de más de dos casillas, el sistema informará al usuario la incapacidad de realizar dicha acción.
2. Si el kenken no tiene solución, el usuario podrá introducir nuevos valores o abandonar la acción actual.

Nombre: Importar kenken
Actor: Usuario
Comportamiento:
1. El sistema pide al usuario el nombre del fichero a importar.
2. El usuario introduce el nombre del fichero con el kenken.
3. El sistema valida el fichero introducido y si es correcto muestra el kenken por pantalla.

Errores posibles y cursos alternativos:
1. Si el path del fichero introducido no es correcto, el sistema lo pedirá de nuevo.
2. Si los datos del fichero introducido no son correctos, el sistema informará del error.

Nombre: Consultar lista kenken
Actor: Usuario
Comportamiento:
1. El sistema lista los kenken guardados.
2. El usuario puede escoger entre:
    a. Empezar a jugar un kenken.
    b. Volver al menú principal.

Errores posibles y cursos alternativos:

Nombre: Continuar kenken
Actor: Usuario
Comportamiento:
1. El sistema devuelve el kenken del usuario guardado previamente.

Errores posibles y cursos alternativos:
1. Si no hay un kenken previamente guardado, el sistema informará al usuario.

Nombre: Jugar
Actor: Usuario
Comportamiento:
1. El usuario estará realizando su partida.

Errores posibles y cursos alternativos:

Nombre: Check
Actor: Usuario
Comportamiento:
1. El usuario le pedirá al sistema que compruebe el estado actual del tablero respecto a la solución.

Errores posibles y cursos alternativos:
1. Si todas las casillas del tablero tienen un valor correcto, el usuario habrá resuelto el kenken

Nombre: Añadir número
Actor: Usuario
Comportamiento:
1. El usuario añade un número al tablero.

Errores posibles y cursos alternativos:
1. Si el número es menor a 1 y mayor al tamaño del tablero, el sistema borra el número e informa al usuario del error.

Nombre: Borrar número
Actor: Usuario
Comportamiento:

1. El usuario borra del tablero de juego un número previamente colocado por el.

Errores posibles y cursos alternativos:

Nombre: Guardar partida actual
Actor: Usuario
Comportamiento:

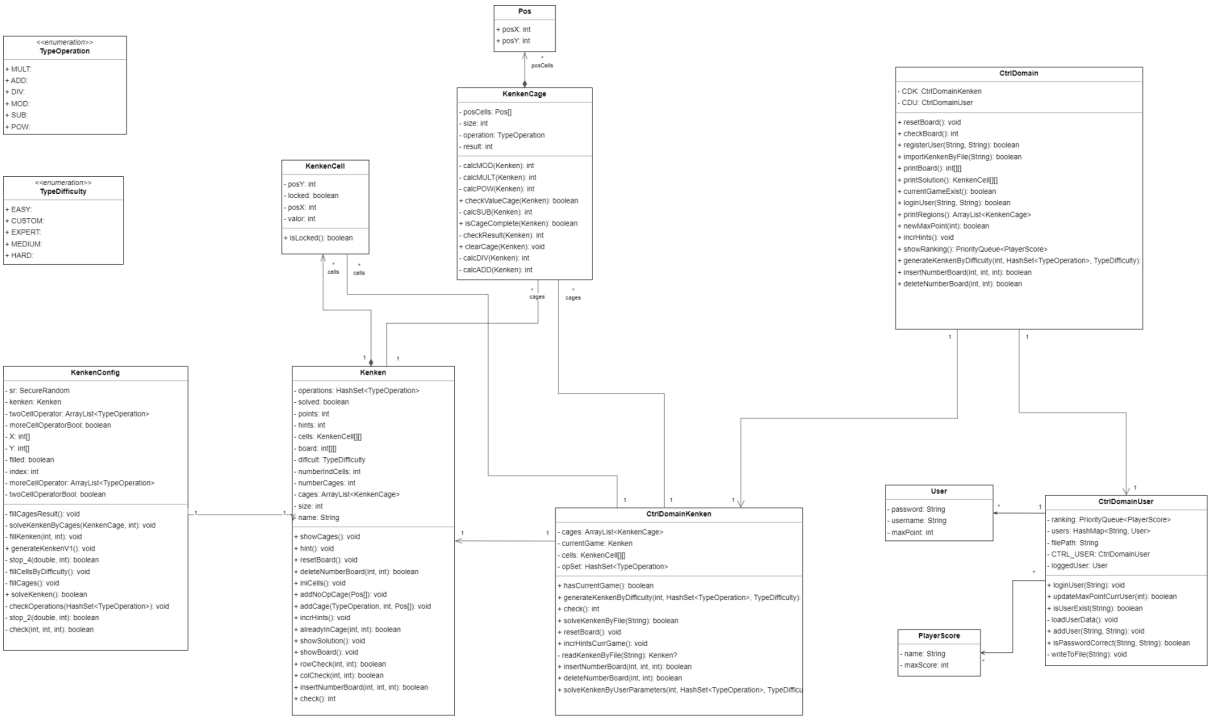
1. El usuario indica al sistema que quiere guardar la partida actual.
2. Puede escoger entre:
    a. Que el sistema guarde la partida actual.
    b. Que el sistema no guarde la partida actual.
3. El usuario vuelve al menú principal.

Errores posibles y cursos alternativos:

1. Si ya existe una partida guardada, el sistema informará al usuario que esta será sobreescrita por la actual.

2. Esquema conceptual del modelo

2.1 Diagrama de clases



2.2 Descripción de las clases

2.2.1 Pos
Clase utilizada principalmente de forma auxiliar. La definimos para guardar las posiciones de las casillas en las regiones y no guardar otra vez las casillas.

2.2.2 TypeDifficulty
Distintos tipos de dificultad que pueden tener los Kenkens.

2.2.3 TypeOperation
Distintos tipos de operaciones que pueden tener los Kenkens.

2.2.4 KenkenCell
Esta clase representa una casilla del Kenken. Esta casilla no es donde el usuario realiza su partida, si no donde el Kenken almacena el resultado previamente calculado.

2.2.5 KenkenCage
Esta clase representa una región del Kenken. Contiene las posiciones de las casillas que le pertenecen, el tipo de operación que hacen entre ellas y el resultado.

2.2.6 Kenken
La clase kenken contiene las casillas y regiones que forman un Kenken como lo conocemos. Contiene también un tablero vacío donde podemos realizar la partida.

2.2.7 KenkenConfig
La clase KenkenConfig es la encargada de, dado un Kenken, generarlo o solucionarlo.

2.2.8 User
Esta clase contiene el nombre y la contraseña del usuario, también sus puntos máximos.

2.2.9 CtrlDomainUser
Es una clase de controlador de los usuarios, un atributo privado de HashMap para guardar los usuarios cargados desde el fichero, y un ranking para listar según sus puntos máximos de cada usuario descendentemente.

2.2.10 CtrlDomainKenken
Esta clase controla el Kenken que se está “manipulando” actualmente, ya sea jugando o generando/solucionando.

2.2.11 CtrlDomain
Una clase controlador de la capa de dominio, para comunicar entre la capa de presentación y la capa de persistencia (para esta entrega no existe).

3. Relaciones de las clases implementadas por cada miembro del equipo

Javier	**Romeu**	**Feiyang**
CtrlDomainKenken	CtrlDomainKenkenTest	CtrlDomain
Kenken	CtrlDomainUserTest	CtrlDomainUser
KenkenConfig	KenkenTest	KenkenCell
DriverCtrlDomainKenken	UserTest	PlayerScore
KenkenCage		TypeDifficulty
Pos		User
TypeOperation		DriverCtrlDomainUser
		Driver

azul -> clase *enumeration*
verde -> clase controlador
amarillo -> clase driver
rojo -> clase test JUnit

4. Estructura de datos y algoritmos utilizados

4.1 Estructura de datos

4.1.1 CtrlDomain

Esta clase no contiene ninguna estructura de datos.

4.1.2 CtrlDomainUsers

En esta clase, se necesita cargar un conjunto de usuarios existentes en el fichero, se carga todos estos usuarios automáticamente al ejecutar el programa, y se guardan en un ***HashMap*** donde la clave es el nombre del usuario ya que es una clave primaria, y el valor es la clase User para que podamos acceder y modificar en caso necesario.
Y para el ranking utilizamos un ***PriorityQueue*** que permite ordenar por los puntos máximos de los jugadores descendentemente, de forma automática.

4.1.3 CtrlDomainKenken

Las estructuras de datos que contiene esta clase únicamente se utilizan cuando se crea un Kenken por consola, en este caso. Contiene tres estructuras de datos principales para la creación de un Kenken, un ***HashSet*** que contiene las operaciones del Kenken, es decir, contiene TypeOperation, un ***KenkenCell[][]*** que contiene, en este caso, las casillas de la solución inicializadas a 0 y finalmente un ***ArrayList*** de KenkenCages que contiene las distintas regiones del Kenken.

4.1.4 KenkenCell

Esta clase no contiene ninguna estructura de datos.

4.1.5 KenkenCage

En esta clase necesitamos almacenar las posiciones de las casillas que pertenecen a la región correspondiente. Lo hacemos mediante un ***Array*** de Pos. Pos es una clase que hemos creado nosotros, la cual contiene un int posX y un int posY.

4.1.6 Kenken

Esta clase contiene cuatro estructuras de datos distintas, la primera es una matriz de ***KenkenCell[][]*** para almacenar las casillas de la solución y/o generación. La segunda es un ***ArrayList*** que almacena las distintas regiones dentro del Kenken. La tercera es una matriz ***int[][]*** que sirve como tablero de juego. Finalmente, tenemos un ***HashSet*** de TypeOperation que contiene las operaciones que hay en el Kenken.

4.1.7 KenkenConfig

Esta clase contiene como estructura de datos principal un **Kenken** de la clase Kenken. Sobre este Kenken se aplica la posibilidad de resolver o generar. Hay también dos ***ArrayList*** para controlar los operadores del Kenken. Estos se utilizan en el momento de generación de regiones del Kenken para saber qué operadores indicados por el usuario podemos utilizar. Tenemos también dos ***Array*** de int

llamados X e Y que ayudan a moverse entre casillas en el momento de generar las regiones de forma aleatoria.

4.1.8 Pos

Esta clase no contiene ninguna estructura de datos.

4.2 Algoritmos

4.2.1 Backtracking

Backtracking es un algoritmo que busca todas las posibles soluciones a un problema dado un conjunto de restricciones iniciales. Este algoritmo se apoya en el uso de la recursividad para realizar una búsqueda exhaustiva de todas las combinaciones posibles.

En nuestro proyecto hemos usado dos versiones distintas de backtracking, una para resolver los Kenkens y otra para generarlos.

La generación de Kenkens se hace mediante un algoritmo de backtracking muy simple. Partiendo de un número aleatorio mayor o igual a uno y menor o igual al tamaño del tablero del kenken empezamos a llenar el tablero desde la casilla 0,0 comprobando que el número no se repita ni en la misma fila ni en la misma columna. Cada vez que encontramos un número que cumpla esta condición, se llama de forma recursiva al algoritmo y se avanza a la siguiente casilla. Si algún número no cumple esta condición y se han probado todos los casos, deshacemos el progreso obtenido y volvemos a intentarlo con otro número distinto. Cada llamada al algoritmo contiene un Array de booleanos de tamaño igual al tamaño del Kenken. Cada posición de este Array indica si el número se ha probado anteriormente o no en esa casilla.

La solución de Kenkens se hace también mediante un algoritmo de backtracking. Este, a diferencia de la generación de Kenkens, también ha de comprobar que los valores dentro de una región cumplan con la condición de operación y resultado de esta. Este algoritmo empieza por la primera casilla de la primera región del Array de regiones del Kenken. Va llenando la región de forma recursiva con números mayores e igual a 1 y menores e igual al tamaño del Kenken, comprobando que no se repitan ni en filas ni columnas. Una vez la región está completa se comprueba si los números en esta cumplen el resultado de la operación de la región. Si es así, de forma recursiva pasamos a la siguiente región. Si algún número no cumple estas condiciones y se han probado todos los casos, deshacemos el progreso obtenido y volvemos a intentarlo con números distintos.

Ante la posibilidad de que el Kenken proporcionado no tenga solución, este backtracking nunca llegará al final, donde hay un booleano que pone su valor a cierto si se llega.

El coste de estos backtracking se calcula de la siguiente forma: en el peor caso de la generación y solución de un Kenken, el coste sería $O(S^{n^2})$. Siendo S el tamaño del Kenken (rango de valores posibles para la casilla) y n^2 el número de casillas totales del Kenken.