Université de Strasbourg

2021/2022

Projet Programmation orienté objet

Kakuro Solver

Hassan TAHA- David PULIDO CORNEJO

Introduction

On se propose de programmer un jeu en java: Kakuro. Le jeu comporte toutes les fonctionnalités requises au bon fonctionnement. Il s'agit d'une grille comportant des cases remplies non modifiables (cases noires), des cases contenant des nombres, des cases vides à remplir avec un nombre et des cases d'indication pouvant contenir deux valeurs, la somme des cases contiguës de la colonne sous la case d'indication et/ou la somme des cases contiguës de la ligne à droite de la case d'indication.

Réalisation

Étant donné qu'on a eu à gérer un travail de binôme, on s'est mis d'accord sur la méthodologie de travail et l'attribution des tâches (classes à réaliser par chacun). Pour la collaboration, on a partagé nos travail sur la plateforme Gitlab. On a profité des avantages de Discord aussi pour s'échanger les idées et discuter sur l'avancement de projet. La répartition du travail était : Hassan a fait la partie vue avec l'UML (page 5), David a fait la partie d'algorithme et contrôleur. Premièrement, on a décidé de mettre en place un modèle MVC qui nous a permis de séparer les données (modèle), l'interface graphique (vue) et la logique de contrôle (contrôleur). Aussi on a pu clarifier l'architecture du code.

1. Interface Graphique

D'abord, on a commencé notre projet par l'interface graphique, la création de la grille a consisté faire un ensemble des classes:

- *(GameGrid):* Création de la grille, en initialisant deux variables ligne et colonne, avec une matrice de cases graphiques contenus dans un JPanel. On a utilisé les fonctions prédéfinies comme setSize, setLayout, etc.
- (*NumberSelect):* La sélection d'un nombre entre 1 et 9 par l'utilisateur dans une case, en affichant une petite fenêtre qui contient les nombres. Avec 2 boutons permettant d'annuler la sélection, ou de réinitialiser une case.
- *(Options):* Affichage des boutons d'options au-dessous de la grille, un pour résoudre le jeu, un autre pour sauvegarder la session et pour réinitialiser tous les cases.
- *(Display)*: affichage de la fenêtre, cette classe utilise toutes les classes de l'interface graphique comme des attributs, pour afficher la grille, les boutons, etc.

2. Modèle

Deux, on s'intéressait à programmer l'algorithme du jeu

- *(ConstraintsChecker):* Contient un ensemble de méthodes pour tester si une grille est valide ou pas, comme *horizontalOK()* et *verticalOK()* qui vérifient s'il ya des valeurs répétées (on peut pas avoir le même nombre sur la même ligne ou colonne). En plus, on teste si la somme des nombres dans une ligne horizontale est plus grande que la somme attendue en utilisant les méthodes *horizontalSumMatches()* et *verticalSumMatches()* (Même chose pour les colonnes). Les méthodes sont publiques pour être utilisable par le solveur du jeu (*Kakurosolver*).

- *(KakuroSolver):* Cette classe contient la structure de donnée de la grille en cours du jeu et un vérificateur de contraintes, capable de notifier le contrôleur si l'utilisateur a mis un nombre qui n'est convenable dans cette case (affichage d'une phrase en haut du fenêtre) . De même, elle contient la méthode de résolution automatique des grilles kakuro. L'algorithme est basé sur un algorithme de type backtracking. À chaque case de type input on essaie d'insérer un chiffre entre 1 et 9. À chaque essai on teste si cette hypothèse est adéquate (avec *checkMisplacement()*), si c'est le cas, on passe à la case suivante, sinon on réessaye avec un nouveau chiffre ou on revient à une case précédente stockée dans la pile d'exécution.

- *(KakuroLoader):* Dans cette classe, on s'intéresse à sauvegarder la grille actuelle si l'utilisateur veut continuer la session après, en utilisant la méthode *saveFile()* qui prend en entrée la grille, puis elle sauvegarde la grille dans un fichier. En plus, on a ajouté la méthode *loadRandomGrid(),* qui charge une nouvelle grille aléatoire d'une dimension spécifique en lisant un fichier du kakuro format. La même chose pour la méthode *loadGrid()*, qui lit un fichier du format kakuro passé en argument (l'arborescence du fichier). La méthode *loadGrid()* lit l'en-tête du fichier pour extraire les dimensions de la grille. Ensuite, ligne par ligne, on fait une disjonction de cas pour chaque caractère. On crée un cas de type control si on lit le caractère 'c', de type input si on lit 'i', constant si on lit 'k' et de type non modifiable si on lit 'u'. En cas d'erreur de format, une erreur de type *InvalidGridFileException* est levée.

3. **Controleur**

- ***(****Controller):* Contient l'ensemble des classes, Display, kakuroSolver, kakuroLoader comme des attributs privés, où on crée des vues pour afficher le modèle. Il collecte toutes les actions utilisateur qui agissent sur ces vues (User Request), pour modifier le modèle en conséquence (Update). Le contrôleur est capable de faire tout les tâches pour démarrer le jeu.

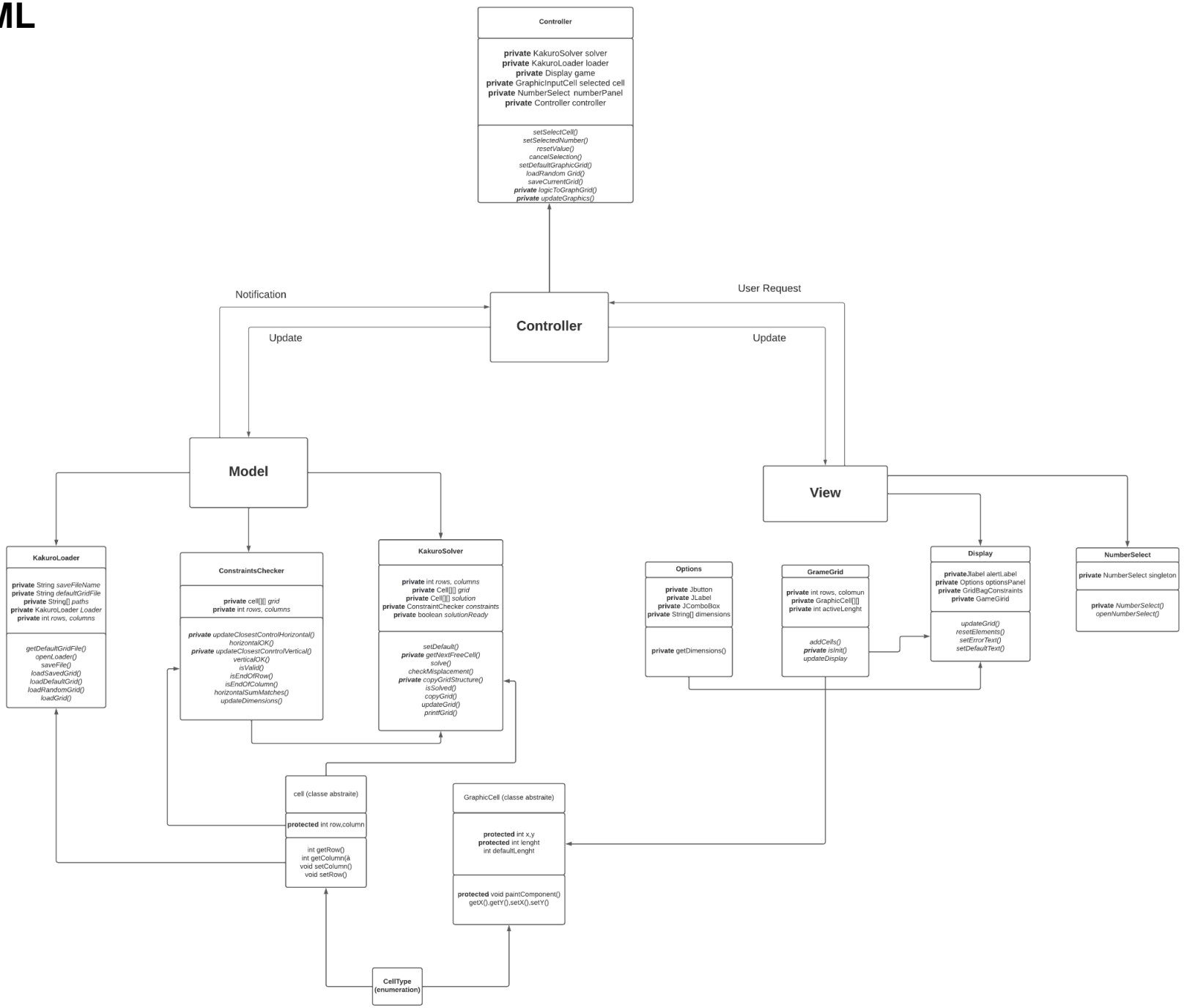
En plus, on a ajouté une fonctionnalité unique en créant une classe *(kakuro Loader),* qui consiste à sauvegarder la session actuelle de l'utilisateur, avec le bouton "save session", où il peut charger la session quand il souhaite avec le bouton "load session". La deuxième c'était de charger une grille aléatoire d'une dimension quelconque (3x3, 4x4, 5x5).

Compilation

Vous trouvez la classe Test contenant le main dans le package "testing".

Pour exécuter le jeu, il suffit d'entrer la commande *java -jar <file>.jar*

UML


Controller
private KakuroSolver solver
private KakuroLoader loader
private Display game
private GraphicInputCell selected cell
private NumberSelect numberPanel
private Controller controller
setSelectCell()
setSelectedNumber()
resetValue()
cancelSelection()
setDefaultGraphicGrid()
loadRandom Grid()
saveCurrentGrid()
private logicToGraphGrid()
private updateGraphics()
Notification
Update
Controller
User Request
Update
Model
View
KakuroLoader
private String saveFileName
private String defaultGridFile
private String[] paths
private KakuroLoader Loader
private int rows, columns
getDefaultGridFile()
openLoader()
saveFile()
loadSavedGrid()
loadDefaultGrid()
loadRandomGrid()
loadGrid()
ConstraintsChecker
private cell[][] grid
private int rows, columns
private updateClosestControlHorizontal()
horizontalOK()
private updateClosestConrtrolVertical()
verticalOK()
isValid()
isEndOfRow()
isEndOfColumn()
horizontalSumMatches()
updateDimensions()
KakuroSolver
private int rows, columns
private Cell[][] grid
private Cell[][] solution
private ConstraintChecker constraints
private boolean solutionReady
setDefault()
private getNextFreeCell()
solve()
checkMisplacement()
private copyGridStructure()
isSolved()
copyGrid()
updateGrid()
printGrid()
Options
private Jbutton
private JLabel
private JComboBox
private String[] dimensions
private getDimensions()
GrameGrid
private int rows, colomun
private GraphicCell[][]
private int activeLenght
addCells()
private isInit()
updateDisplay
Display
private JLabel alertLabel
private Options optionsPanel
private GridBagConstraints
private GameGirid
updateGrid()
resetElements()
setErrorText()
setDefaultText()
NumberSelect
private NumberSelect singleton
private NumberSelect()
openNumberSelect()
cell (classe abstraite)
protected int row,column
int getRow()
int getColumn(à
void setColumn()
void setRow()
GraphicCell (classe abstraite)
protected int x,y
protected int lenght
int defaultLenght
protected void paintComponent()
getX(),getY(),setX(),setY()
CellType
(enumeration)