Analyse et Conception de iSudoku

V1. 12/12/2013 Yann Thierry-Mieg

Contenu

Nature du document :

Ce document est à vocation de servir de support pour l'UE MI 017, Ingénierie du Logiciel, en Master 1 à l'université P&M Curie. Il résout partiellement le cahier des charges posé aux étudiants des promos 2009 à 2013 du master, et servira de base exemple pour résoudre le cahier des charges des promos suivantes.

Le cahier des charges initial est reproduit ici, les sections barrées ne sont pas couvertes par ce document.

Synopsis :

iSudoku est une application pour plateforme mobile qui permet de s'entraîner et de progresser au Sudoku.

Le but du jeu de Sudoku est de remplir la grille composée de 9 sous-grilles avec une série de chiffres allant de 1 à 9 tous différents, qui ne se trouvent jamais plus d'une fois sur une même ligne, dans une même colonne ou dans une même sous-grille. Les sous-grilles sont des carrés de 3 × 3.

Quelques chiffres sont déjà disposés dans la grille, ce qui permet progressivement de déduire les autres valeurs.

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

5	3	4	6	7	8	9	1	2
6	7	2	1	9	5	3	4	8
1	9	8	3	4	2	5	6	7
8	5	9	7	6	1	4	2	3
4	2	6	8	5	3	7	9	1
7	1	3	9	2	4	8	5	6
9	6	1	5	3	7	2	8	4
2	8	7	4	1	9	6	3	5
3	4	5	2	8	6	1	7	9

L'application supportera les fonctionnalités suivantes :

- Permettre de façon intuitive et ergonomique d'afficher et de résoudre des grilles de Sudoku.
- Différentes difficultés seront proposées, de la plus difficile (on ne peut valider la grille que quand elle est finie) à la plus facile (dans chaque case, le système n'offre que les valeurs possibles au vu du reste de la grille, et/ou met en valeur les cellules qui n'ont plus qu'une option possible). En difficulté intermédiaire l'application signale une erreur au moment où elle est commise. Les options d'aide à l'utilisateur (signaler les erreurs, afficher des suggestions..) seront configurables par ce dernier.
- ~~Générer des grilles de diverses difficultés, et~~ utiliser des grilles existantes fournies avec l'application ~~ou téléchargées.~~
- En prenant une photo d'une grille (typiquement celle de votre journal), le système permettra de la charger dans l'application et ainsi à l'utilisateur de la résoudre. Cette fonctionnalité s'appuie ~~sur un module de reconnaissance de forme existant déjà développé dans la SSII.~~
- L'application permettra de mesurer le QI du joueur. Il s'appuie pour cela ~~sur la difficulté des grilles traitées et le~~ temps mis à les résoudre. On pourra voir les statistiques du ~~joueur et sa progression au fil du temps. Le joueur sera encouragé à persévérer par des bonus quand il résout une grille.~~

- ~~L’application permettra de publier ses résultats sur divers réseaux sociaux (face2bouc etc.) via un module de diffusion existant.~~

Analyse de iSudoku

Acteurs :

Désignation	Sens métier
Joueur	Le propriétaire du téléphone, et donc l'utilisateur du logiciel
Reconnaissance Formes	Le composant (fourni) qui est chargé de lire les images est modélisé comme un acteur

Cas d'utilisation :

Id	Désignation	Acteurs	Description
UC01	Jouer Partie	Joueur	Permet de jouer une partie de sudoku : saisie des valeurs…
UC02	Choisir Grille	Joueur, Reconnaissance Formes	Permet au joueur de choisir une grille parmi les fichiers ou images présents dans sa bibliothèque
UC03	Gérer Options	Joueur	Permet de positionner le niveau d'aide à la résolution (suggestions, erreurs signalées…)
UC04	Consulter Statistiques	Joueur	Permet de voir son QI et des statistiques sur le joueur

Diagramme de cas d'utilisation

Ces cas d'utilisation sont modélisés sur la figure 1 qui suit. Ils seront détaillés dans les sections « Fiches Détaillées ». Nous n'avons pas retenu dans cette itération les cas d'utilisation « Publier Résultats », « Générer Grille », « Télécharger Grille » et « Consulter Progression ».

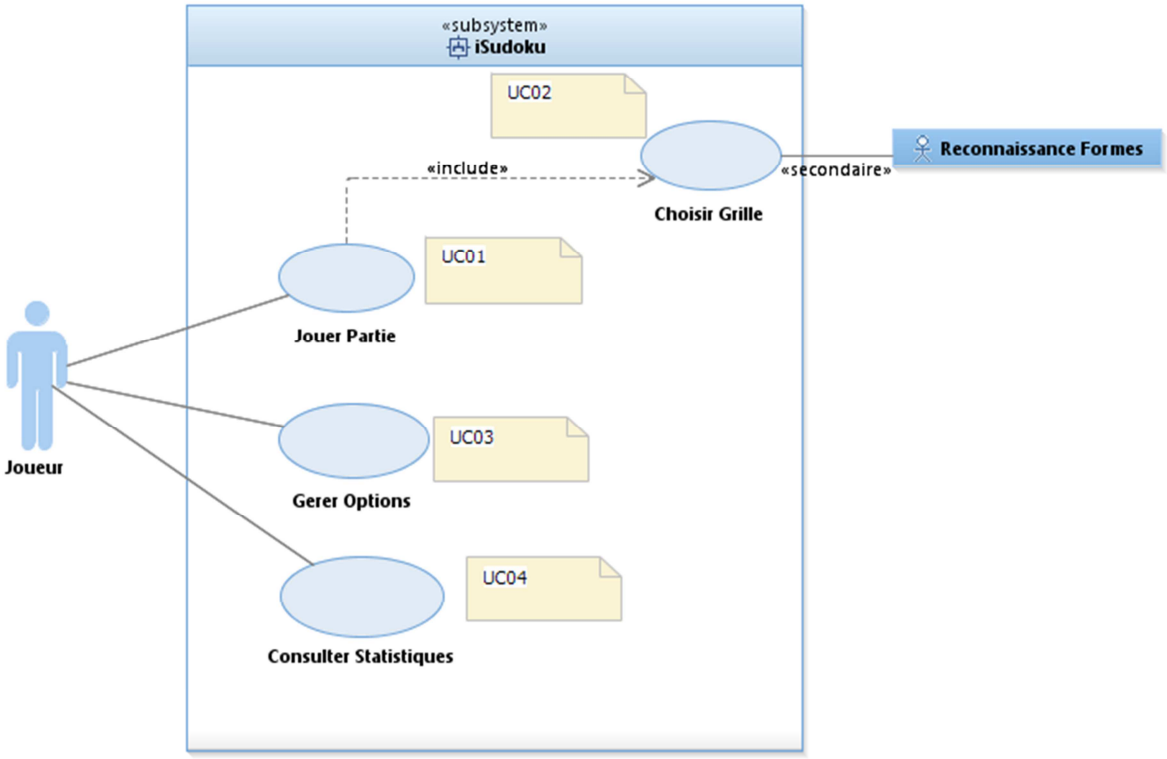


Fig.1 : Cas d'utilisation de iSudoku

Fiches Détaillées

UC01 : Jouer Partie

Description : Permet au joueur de jouer une partie de sudoku.

Acteurs : Joueur

Hypothèse : L'application est démarrée

Préconditions : aucune

Scénario Nominal :

1. Le joueur choisit l'option « charger »
2. Exécution du cas d'utilisation « Choisir Grille »
3. Le système démarre un chronomètre.
4. Le système affiche la grille et le contenu des cellules. Les cellules initiales sont mises en évidence.
5. Le système affiche une bordure rouge si la grille est invalide, i.e. le contenu des cases viole une ou plusieurs contraintes ligne, colonne ou bloc.
6. Le système met en valeur (rouge) les cellules dont le contenu viole une contrainte du sudoku : son contenu est en contradiction avec une contrainte ligne, colonne ou bloc. Les cellules vides mais qu'on ne peut plus remplir sans violer de contrainte sont aussi mise en évidence.
7. Le système met en valeur (vert) les cellules qui n'ont plus qu'une seule possibilité, c'est à dire qu'elle est vide et qu'il n'y a qu'une valeur qu'on puisse y mettre sans violer de contrainte.
8. Le Joueur sélectionne une cellule, non initiale (fixée par énoncé).
9. Le système affiche les valeurs qu'il est possible de mettre dans la cellule sans violer de contraintes (suggestions).
10. Le système invite à saisir une valeur entre 1 et 9 (0 ou une chaine vide seront interprétés comme « vider la case »).
11. Le joueur saisit une valeur
12. Le système met à jour la grille avec la valeur saisie.
13. Le système teste si la partie est finie (toutes les cases sont remplies).
14. Le système enregistre les résultats obtenus et arrête le chronomètre (temps de résolution).
15. Le système affiche un message saluant la victoire et donnant les statistiques sur la partie jouée.

Post-conditions :

Le système a enregistré les statistiques de la partie. ~~Le joueur s'est bien amusé.~~

Alternatives :

A1 : Difficulté de jeu « Facile » (cf. UC. Gérer Options)

Les étapes 7 et 9 du scenario nominal ne sont pas exécutées dans ce niveau de difficulté.

A2 : Difficulté de jeu « Moyen » (cf. UC. Gérer Options)

Les étapes 6, 7 et 9 du scenario nominal ne sont pas exécutées dans ce niveau de difficulté.

A3 : Difficulté de jeu « Difficile » (cf. UC. Gérer Options)

Les étapes 5, 6, 7 et 9 du scenario nominal ne sont pas exécutées dans ce niveau de difficulté.

A4 : Partie non terminée

A l'étape 13 du scenario nominal, si la grille n'est pas remplie, retour en SN4.

A5 : Faute saisie

A l'étape 12 du scénario nominal, si la valeur saisie n'est pas cohérente (une chaine qui ne soit pas un entier de 0 à 9), le contenu de la cellule n'est pas mis à jour et le comportement reprend à l'étape SN4.

Exceptions

E1 : Interruption de la partie

A partir de l'étape 3 du scénario nominal, le joueur peut à tout moment déclencher le cas d'utilisation « choisir grille » et donc interrompre la partie en cours. Les résultats de la partie en cours sont perdus, la partie est abandonnée.

Commentaires :

Une difficulté est de modéliser correctement la boucle (on remplit les cellules une par une jusqu'à ce que la grille soit pleine). Pour garder un scenario nominal linéaire, on a recours à une alternative (A4) placée sur une étape ou le système teste une condition dans le nominal. Ceci permet de préserver les conventions de rédaction de la fiche.

Au niveau des options, le choix retenu est de faire un scenario nominal qui couvre toutes les options possibles (le mode newbie), et d'en désactiver certaines dans les alternatives. Cela donne un résultat plus lisible en général que l'inverse (nominal en difficile + ajout des étapes d'affichage dans les alternatives).

Il est très important de bien détailler les affichages du système, ce sera la base pour décrire au niveau fonctionnel ce que le système a besoin de calculer (suggestions, validités…). Cela pose des besoins qu'il faut identifier à ce stade de l'analyse.

On note l'étape SN2, traduite par un « include » sur le diagramme de cas d'utilisation.

UC02 : Choisir Grille

Description : Permet au joueur de charger une grille existante, ou à partir d'une photo.

Acteurs : Joueur, (secondaire) Reconnaissance Formes

Hypothèse : L'application est démarrée

Préconditions : aucune

Scénario Nominal :

1. Le joueur choisit l'option « charger Fichier»
2. Le système affiche un sélecteur de fichier, positionné dans le répertoire « grilles » de l'application.
3. Le joueur choisit un fichier extension .sdk
4. Le système charge le contenu du fichier dans une grille.
5. Le système affiche la grille

Post-conditions :

Le système a chargé et affiche la grille choisie.

Alternatives :

A1 : Charger Image

En SN1, le Joueur choisit plutôt de charger une photo.

A1.1 le système affiche un sélecteur de fichier, positionné dans le répertoire « grilles » de l'application.
A1.2 Le joueur choisit un fichier extension .png ou .jpg
A1.3 Le système demande à l'acteur Reconnaissance Formes de lire la grille
A1.4 Le système charge le résultat dans une grille
A1.5 retour en SN5

Exceptions

E1 : Fichier Invalide

En SN4 ou A1.3, si la lecture du contenu du fichier échoue, le système notifie l'utilisateur et l'interaction est terminée.

E2 : Annulation

En SN3 ou A1.2, le Joueur peut annuler l'opération, le système poursuit son fonctionnement précédant la demande de chargement, la grille n'est pas chargée.

Commentaires :

Pas de grosse difficulté ici, si ce n'est la cohérence avec le diagramme de use case. On a opté pour une modélisation gros grain des cas d'utilisation, où « charger depuis photo » et « charger depuis

fichier » ne sont pas identifiés comme des cas séparés. On obtient une modélisation assez efficace niveau fiches détaillées vu que la spécification des exceptions n'est faite qu'une fois.

Si on a des alternatives sur une alternative, il est sans doute temps de créer un cas d'utilisation qui décrit l'alternative (principale) dans sa propre fiche détaillée.

Si on intègre aussi « télécharger » et « générer grille » en alternatives la fiche devient assez lourde, sans doute il faut découper en plusieurs cas d'utilisation. On peut les lier en extends sur « Choisir Grille » par exemple, avec par conséquent une alternative ici qui spécifie :

A2 : Télécharger : En SN1, si le joueur choisit de télécharger une grille. Invocation du cas d'utilisation « Télécharger grille» puis retour en SN5.

Par homogénéité on extrairait alors sans doute l'alternative « Depuis photo » aussi en cas d'utilisation. On peut cependant garder un nominal pour « choisir grille » qui décrit le chargement d'un fichier (pour éviter un scenario nominal vide).

UC03 : Consulter Statistiques

Description : Permet au joueur de visualiser les statistiques de ses parties et son « QI »

Acteurs : Joueur

Hypothèse : L'application est démarrée

Préconditions : aucune

Scénario Nominal :

1. Le joueur choisit l'option « Voir Statistiques»
2. Le système affiche le nombre de parties remportées, le temps total passé à résoudre des grilles en secondes et le QI. Le QI est calculé à partir du nombre de victoires v par la formule : 80 + v + v/3 + v/5 + v/10. (calculé en division Euclidienne)
3. Le joueur valide

Post-conditions :

Aucune. ~~Le joueur est fier de lui.~~

Alternatives :

Aucune.

Exceptions

Aucune.

Commentaires :

La difficulté ici est d'interpréter les ambigüités du cahier des charges, qui ne spécifie pas correctement le calcul du QI ni vraiment ce qu'il faut afficher. Il faut identifier et combler ce manque, à ce stade et pas en conception.

On fait ici une lecture allégée du cahier des charges vu qu'on ignore l'historique du joueur, ainsi que l'intervention de la difficulté des grilles dans le calcul des performances du joueur. Notons que cela nécessiterait une notation de la difficulté d'une grille, pas évidente à mettre en place a priori.

Le traitement des « bonus » n'est pas réalisé ici, vu que de nouveau il est mal spécifié dans le cahier des charges. On peut le mettre de côté pour une itération suivante, ça ne nuit pas à la qualité de ce qu'on développe déjà.

On devrait avoir également dans une alternative une invocation au cas d'utilisation « publier sur réseau social », accessible via un bouton en SN3.

UC04 : Gérer Options

Description : Permet au joueur de modifier le niveau d'aide

Acteurs : Joueur

Hypothèse : L'application est démarrée

Préconditions : aucune

Scénario Nominal :

1. Le joueur choisit via le menu « Difficulté» un niveau de difficulté parmi « noob », « easy », « medium », « hard »
2. Le système enregistre ses préférences

Post-conditions :

L'affichage est mis à jour en conséquence (cf. UC01).

Alternatives :

Aucune.

Exceptions

Aucune.

Commentaires :

Une lecture au rasoir du cahier des charges, mais qui le satisfait. Le gros de la spécification est en fait dans UC01.

Diagramme de classes métier

Les classes métier identifiées à ce stade sont représentées figure 2.

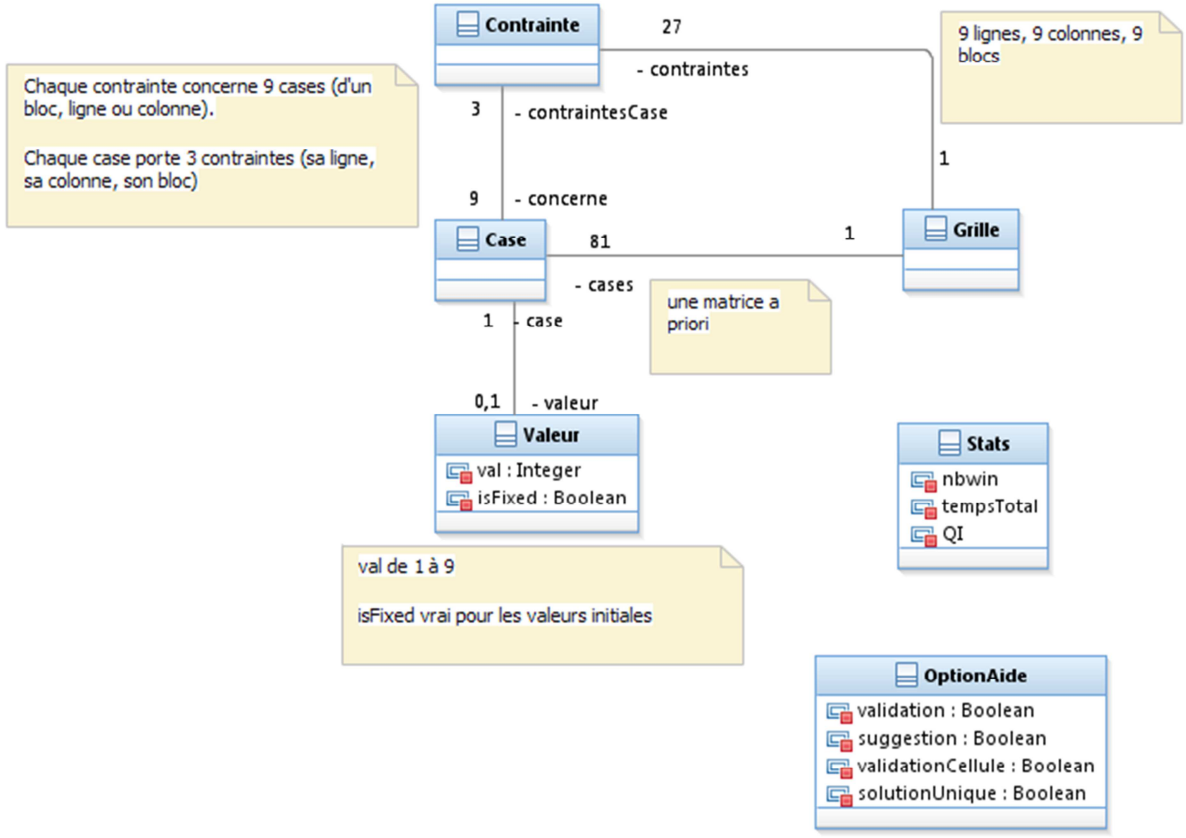


Fig. 2 : Diagramme de classes métier

Commentaires :

Il n'y a pas grand-chose à découvrir, si ce n'est les contraintes, un peu cachées dans le cahier des charges. Le fait qu'on ait coupé du CdC l'historique de jeu simplifie les relations entre les statistiques et les Grilles. Une modélisation plus fine de l'historique fait apparaitre une classe PartieJouée, liée à une Grille, et portant un temps de résolution et une date. Stats serait liée à * PartieJouée. Stats est conforme à ce qui est proposé dans la fiche détaillée UC03, donc assez pauvre par rapport au CdC.

Il ne faut pas modéliser les images, fichiers etc… qui sont des artefacts du ressort du fonctionnel plus que de la structure des données à manipuler qu'on modélise ici.

La classe Joueur ne s'impose pas, outre le fait qu'il y a une confusion possible avec l'acteur, on a un seul joueur donc pas de distinction de profil à réaliser.

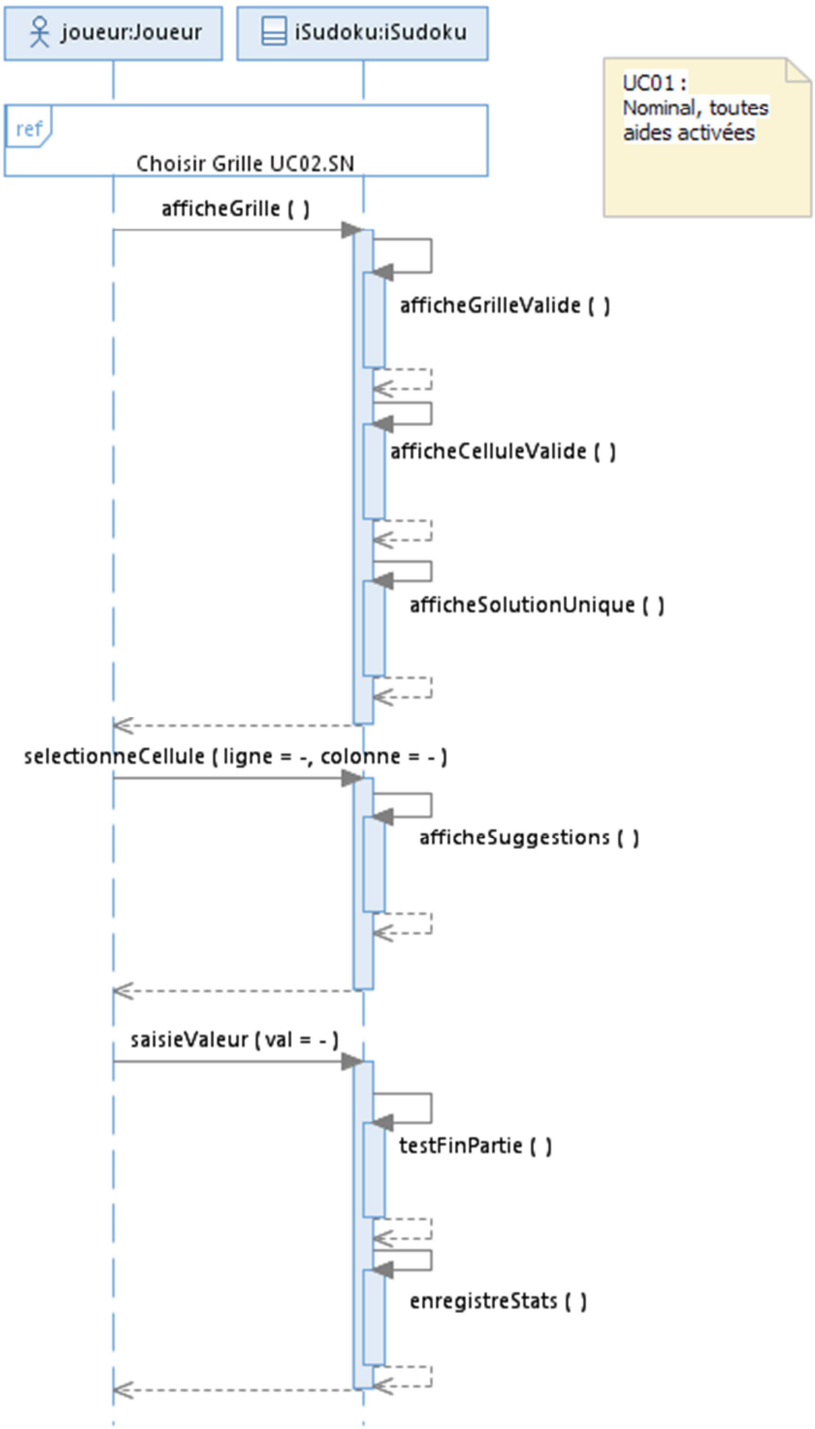
Si on a les réseaux sociaux, on découvre une classe qui porte la description et les identifiants sur le réseau social.

La difficulté (niveau d'aide) est ici traitée via un ensemble de booléens, conforme au cahier des charges : « Les options d'aide à l'utilisateur (signaler les erreurs, afficher des suggestions..) seront configurables par ce dernier. »

La difficulté intrinsèque d'une grille n'est pas traitée, ce serait sans doute une énumération à trois ou quatre niveaux (facile à démoniaque), liée à une Grille.

Séquences d'interaction

UC01. Séquence Nominale

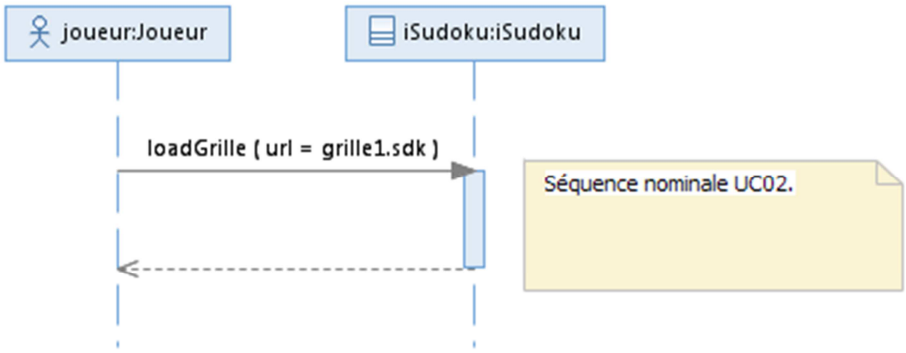


Remarques :

On découvre beaucoup d'opérations « privées » du système, dont la signature n'est donc pas bien définie à ce stade. On raffinera en conception.

On est fidèle à la fiche détaillée, on a juste omis la modélisation du chronomètre. On a la sélection (ligne/col) séparé de la mise à jour de la valeur ici.

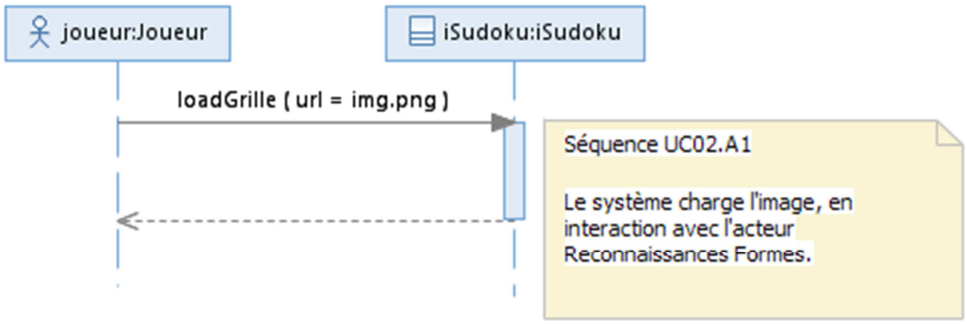
UC02. Séquence Nominale (charger fichier)



Remarques :

On ne modélise pas toutes les interactions de l'acteur : sa ligne de vie et celle de son IHM sont confondues. On a typé à ce stade la valeur de retour de loadGrille par la classe métier Grille. Ceci sera à retoucher en conception (on utilisera une interface IGrille a priori).

UC02.A1 Séquenc e Alternative 1 (charger image)



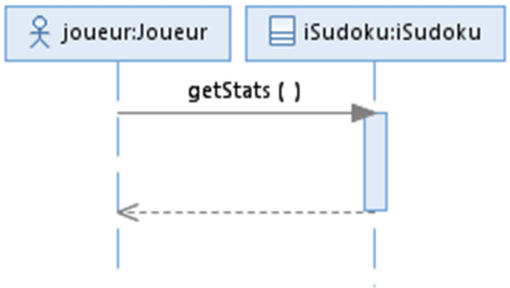
Remarques :

On opte ici pour une modélisation qui permet de réutiliser la signature loadGrille découverte précédemment telle quelle. On aurait aussi pu introduire une nouvelle opération « LoadFromImage(path :String) ».

On a pris des choix simples pour la lecture du CdC : on ne prend pas la photo avec l'application, donc elle existe déjà. Sinon l'interaction serait plus complexe (viser, prendre photo…).

L'acteur secondaire Reconnaissance Formes n'est pas modélisé, on ne sait pas bien la signature des services qu'il offre à ce stade. C'est une faute de modéliser des invocations sur un Acteur en UML.

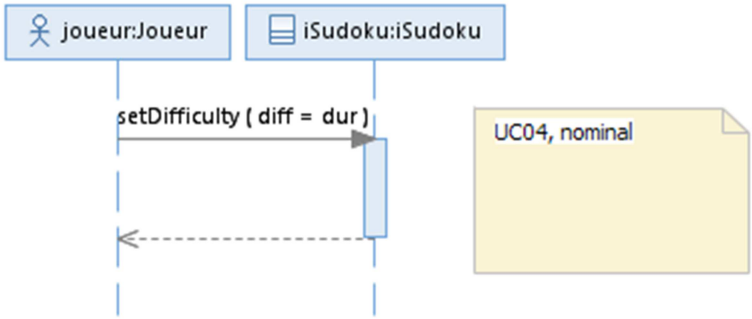
UC03.SN Séquence Nominale (Consulter Stats)



Remarques :

On a une seule invocation, la valeur de retour est typée par la classe métier « Statistique », à raffiner en conception. On aurait pu modéliser plutôt trois invocations : getNbWins, getQI, getTempsTotal. Au final on capture à peu près la même information.

UC04.SN Séquence Nominale (Gérer Options)



Remarques :

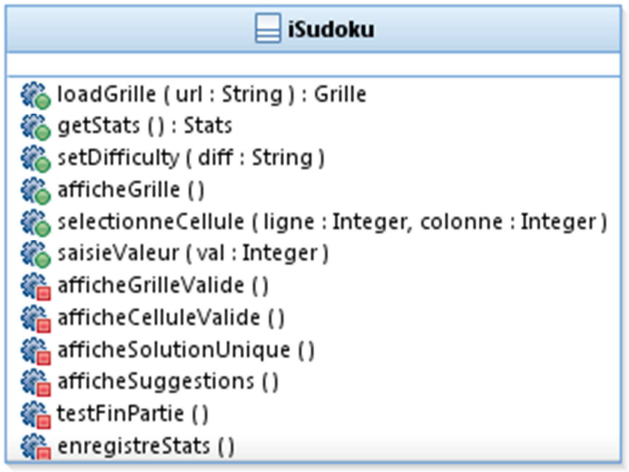
On a une seule invocation, dont le paramètre est ici typé String, peu explicite. On pourrait introduire plutôt une énumération métier « NiveauDifficulté » pour mieux spécifier les valeurs possibles de « diff ».

Une autre alternative cohérente avec la modélisation métier et de faire figurer des invocations « doShowInvalidGrille(bool), doShowInvalidCell(bool), doShowSuggestions(bool)… », donc d'être plus précis sur chacune des options activables.

On pourrait faire un diagramme par niveau de difficulté, mais ce n'est pas très productif et assez verbeux. Vu la modélisation de la fiche détaillée UC04, ça ne parait pas s'imposer à ce stade.

Opérations offertes par le système

La modélisation des séquences ci-dessus permet de déduire les signatures suivantes pour les opérations critiques du système :



Remarques :

On distingue ici les opérations invoquées par l'acteur (rond vert, public) des traitements internes (carré rouge, private) que le système fait. Les opérations identifiées serviront de base à l'élaboration des interfaces des composants en conception. Ces signatures sont un guide en conception, mais ne seront sans doute pas toutes conservées telle quelles.

Tests de Validation

Titre : TV01 : démarrage

Contexte : Aucun, mise en route de l'application

Entrée : aucune

Scenario :

1. L'utilisateur double-clique le fichier iSudoku.jar fourni

Résultat Attendu : L'application iSudoku démarre, une grille vide et les menus sont visibles.

Moyen de vérification : Visuel

Titre : TV02 : charger grille, erreur

Contexte : Après TV01 ; l'application est démarrée.

Entrée : Fichier à cibler « illegal.sdk »

Scenario :

1. Le joueur choisit l'option « charger fichier »
2. Le joueur sélectionne le fichier « illegal.sdk » fourni avec les jeux de test.

Résultat Attendu : L'application iSudoku affiche une erreur ; la grille affichée reste la même qu'avant l'opération.

Moyen de vérification : Visuel

Titre : TV03 : charger grille, nominal

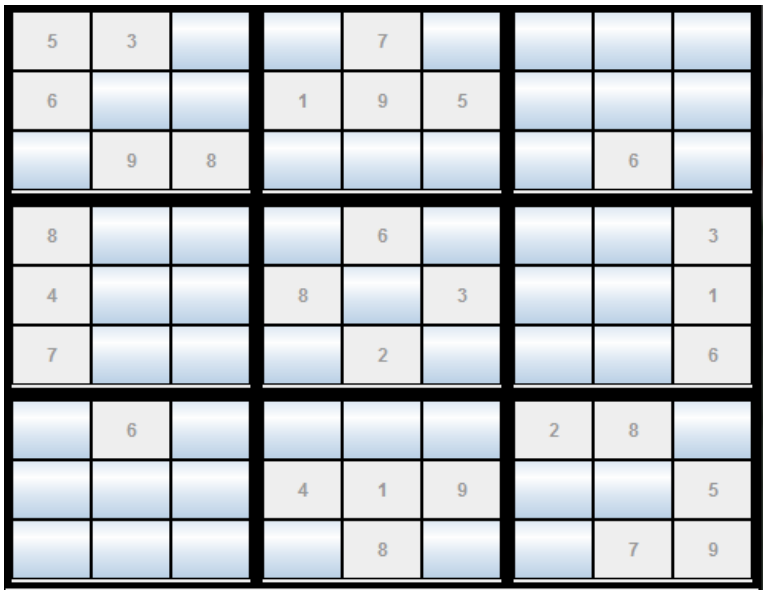
Contexte : Après TV01 ; l'application est démarrée.

Entrée : Fichier à cibler « grille1.sdk »

Scenario :

1. Le joueur choisit l'option « charger fichier »
2. Le joueur sélectionne le fichier « grille1.sdk » fourni avec les jeux de test.

Résultat Attendu : L'application iSudoku affiche la grille grille1. Son contenu doit être le suivant :



Moyen de vérification : Visuel

Titre : TV04 : saisir valeur

Contexte : Après TV03; grille1.sdk est chargée.

Entrée : Valeur « 7 » à saisir dans la cellule (0,2) (première ligne, troisième colonne)

Scenario :

1. Le joueur selectionne la case (0,2)
2. Le joueur saisit la valeur 7.

Résultat Attendu : L'application iSudoku affiche la grille mise à jour, la valeur 7 doit être montrée à la position (0,2), et différenciée des cellules intiales.

Moyen de vérification : Visuel

Titre : TV05 : saisie annulée

Contexte : Après TV04; grille1.sdk est chargée, on a valeur 7 en (0,2).

Entrée : cellule (0,2) (première ligne, troisième colonne)

Scenario :

1. Le joueur selectionne la case (0,2)
2. Annule sa saisie.

Résultat Attendu : L'application iSudoku affiche la grille sans mise à jour, la valeur 7 doit encore être montrée à la position (0,2), et différenciée des cellules intiales.

Moyen de vérification : Visuel

Titre : TV06 : Afficher aucune aide

Contexte : Après TV04 ou TV05; grille1.sdk est chargée, on a valeur 7 en (0,2).

Entrée : difficulté « hard »

Scenario :

1. Le joueur sélectionne la difficulté « hard »

Résultat Attendu : L'application iSudoku affiche la grille sans aucune mise en valeur de cellules.

Moyen de vérification : Visuel

Titre : TV07 : Afficher aucune suggestion

Contexte : Après TV06; grille1.sdk est chargée, on a valeur 7 en (0,2), difficulté positionnée sur hard.

Entrée : case (0,2)

Scenario :

1. Le joueur sélectionne la case (0,2)

Résultat Attendu : L'application iSudoku afficheune invite de saisie, mais pas de suggestions.

Moyen de vérification : Visuel

Titre : TV08 : Afficher validité grille

Contexte : Après TV06; grille1.sdk est chargée, on a valeur 7 en (0,2), difficulté positionnée sur hard.

Entrée : difficulté medium

Scenario :

1. Le joueur sélectionne la difficulté medium

Résultat Attendu : L'application iSudoku affiche la même grille mais indique par une bordure rouge le fait qu'elle est invalide.

Moyen de vérification : Visuel

Titre : TV09 : Afficher validité cellule

Contexte : Après TV08; grille1.sdk est chargée, on a valeur 7 en (0,2), difficulté positionnée sur medium.

Entrée : difficulté easy

Scenario :

1. Le joueur sélectionne la difficulté easy

Résultat Attendu : L'application iSudoku affiche la même grille, mais indique par une bordure rouge le fait qu'elle est invalide, et entoure la cellule à la position (0,2) qui induit la violation de contrainte.

Moyen de vérification : Visuel

Titre : TV10 : Afficher cellule à une seule solution

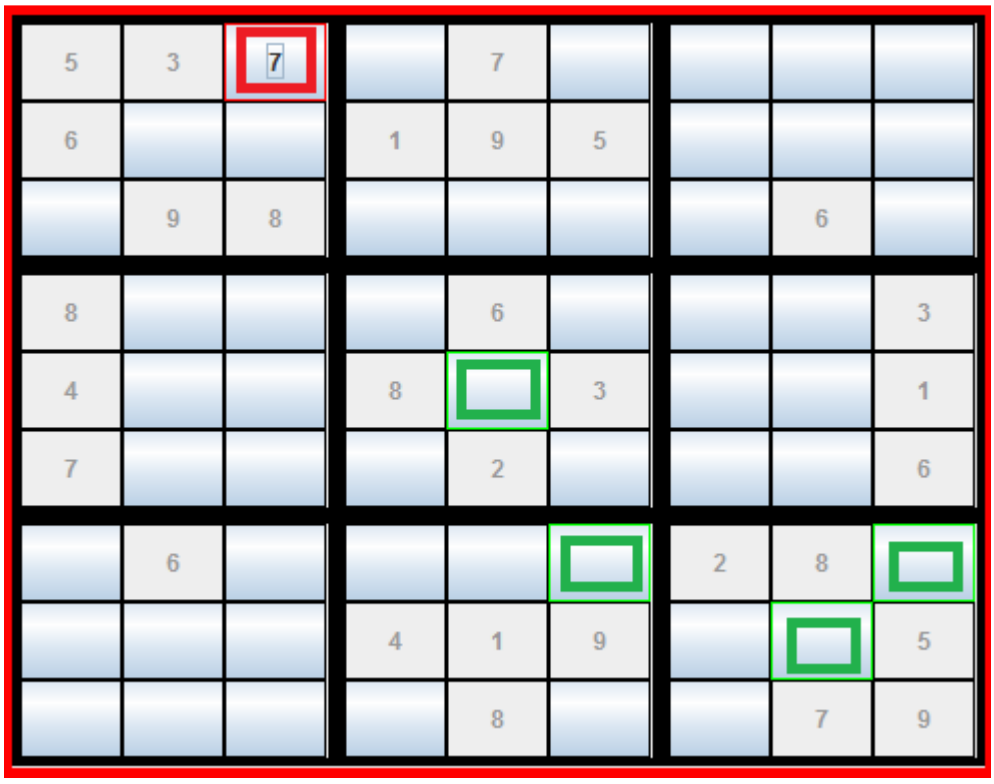
Contexte : Après TV09; grille1.sdk est chargée, on a valeur 7 en (0,2), difficulté positionnée sur easy.

Entrée : difficulté noob

Scenario :

1. Le joueur sélectionne la difficulté n00b

Résultat Attendu : L'application iSudoku affiche la même grille, mais indique par une bordure rouge le fait qu'elle est invalide, et entoure la cellule à la position (0,2) qui induit la violation de contrainte. De plus elle met en valeur les cellules : (4,4), (6 ,5), (6,8) et (7,7) pour lesquelles il n'y a qu'une seule solution.



Moyen de vérification : Visuel, l'image ci-dessus sert de contrôle.

Titre : TV11 : Afficher suggestions 1

Contexte : Après TV10; grille1.sdk est chargée, on a valeur 7 en (0,2), difficulté positionnée sur noob.

Entrée : case (1,1)

Scenario :

1. Le joueur sélectionne la case (1,1)

Résultat Attendu : L'application iSudoku affiche une invite à saisir une valeur et suggère les valeurs :2 ou 4.

Moyen de vérification : Visuel.

Titre : TV12 : Afficher suggestions 2

Contexte : Après TV10; grille1.sdk est chargée, on a valeur 7 en (0,2), difficulté positionnée sur noob.

Entrée : case (7,7)

Scenario :

1. Le joueur sélectionne la case (7,7) (elle est mise en valeur vert)

Résultat Attendu : L'application iSudoku affiche une invite à saisir une valeur et suggère la valeur :3.

Moyen de vérification : Visuel.

Titre : TV12 : charger grille depuis image, nominal

Contexte : Après TV01 ; l'application est démarrée.

Entrée : Fichier à cibler « grille1.jpg »

Scenario :

1. Le joueur choisit l'option « charger image »
2. Le joueur sélectionne le fichier « grille1.jpg » fourni avec les jeux de test.

Résultat Attendu : L'application iSudoku affiche la grille grille1. Son contenu doit être le suivant :

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

Moyen de vérification : Visuel

Titre : TV13 : consulter stats

Contexte : Après TV01 ; l'application est démarrée.

Entrée : aucune

Scenario :

1. Le joueur choisit l'option « Voir stats »

Résultat Attendu : L'application iSudoku affiche des stats : parties jouées 0, QI 80 et temps total 0.

Moyen de vérification : Visuel

Titre : TV14 : interrompre partie, nominal

Contexte : Après TV12 ; l'application est démarré une grille est en cours de résolution.

Entrée : Fichier à cibler « finie.sdk »

Scenario :

1. Le joueur choisit l'option « charger fichier »
2. Le joueur sélectionne le fichier « finie.sdk » fourni avec les jeux de test.

Résultat Attendu : L'application iSudoku affiche maintenant la grille « finie ». Son contenu doit être le suivant :

	3	4	6	7	8	9	1	2
6	7	2	1	9	5	3	4	8
1	9	8	3	4	2	5	6	7
8	5	9	7	6	1	4	2	3
4	2	6	8	5	3	7	9	1
7	1	3	9	2	4	8	5	6
9	6	1	5	3	7	2	8	4
2	8	7	4	1	9	6	3	5
3	4	5	2	8	6	1	7	9

Moyen de vérification : Visuel

Titre : TV15 : fin de partie, nominal

Contexte : Après TV14 ; l'application est démarré la grille de test « finie » est affichée.

Entrée : Valeur 5 à saisir en 0,0

Scenario :

1. Le joueur sélectionne la cellule(0,0)
2. Le joueur saisit la valeur 5

Résultat Attendu : L'application iSudoku affiche « partie gagnée »

Moyen de vérification : Visuel

Titre : TV16 : consulter stats màj

Contexte : Après TV15 ; on a résolu une grille.

Entrée : aucune

Scenario :

1. Le joueur choisit l'option « Voir stats »

Résultat Attendu : L'application iSudoku affiche des stats : parties jouées 1, QI 81 et temps total cohérent avec la durée du test.

Moyen de vérification : Visuel

Remarques :

Le jeu de test ne sera jamais exhaustif, il faut essayer de couvrir un maximum des séquences identifiées dans les cas d'utilisation.

La reproductibilité est importante, ici assurée via un jeu de données de test.

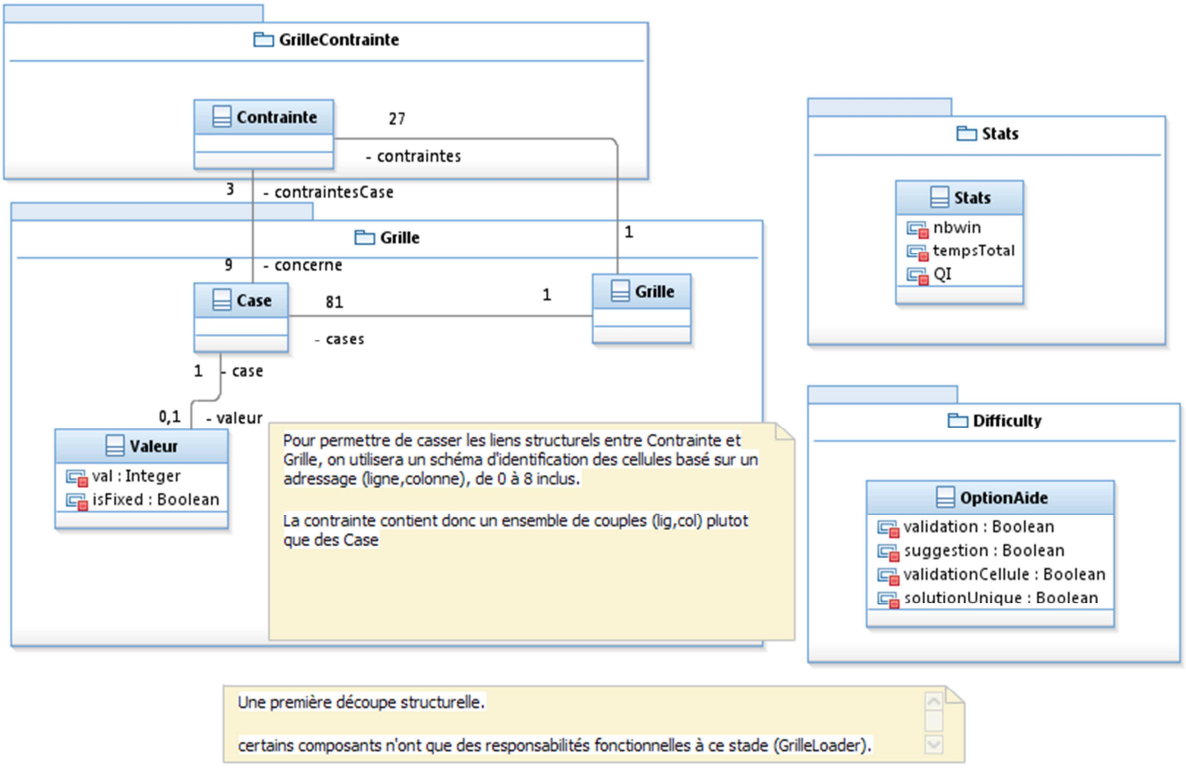
Les tests s'enchaînent pour faciliter l'exécution du jeu de tests.

Phase de transition

La transition de l'analyse à la conception est toujours un exercice difficile.

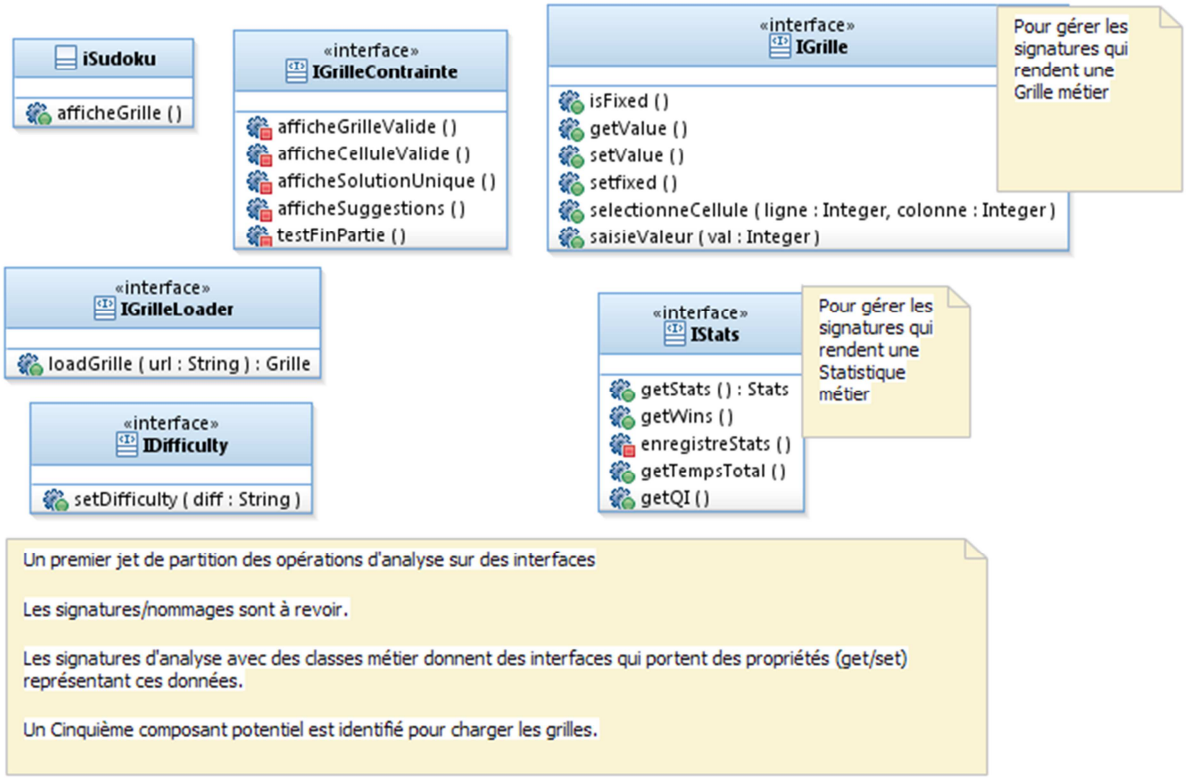
Nous partons ici d'une découpe structurelle (diagramme de classes) et d'une découpe fonctionnelle (opérations de la classe système) pour atteindre un premier découpage en composants.

Découpe Structurelle



On identifie quatre groupes de données métier. On opte pour une séparation des contraintes vis-à-vis des grilles, qui permettra de s'en soucier dans un deuxième temps.

Découpe Fonctionnelle



A ce stade il faut commencer à raffiner les signatures : Grille devient IGrille, Stats devient IStats. L'opération getStats est redondante dans IStats, on l'élimine.

On renomme les opérations d'affichage « afficheValid() » dans GrilleContrainte en « computeValid() ».

Comme toutes les opérations de GrilleContrainte nécessitent une IGrille comme contexte, on décide de faire un héritage d'interfaces, IGrilleContrainte extends IGrille.

On imagine qu'on aura plusieurs implémentations de IGrilleLoader, un ImageLoader, un FileLoader…

La conception

Il faut ensuite réaliser des itérations Conception Architecturale/Conception Détaillée/Code/Test (ou test/code), en partant des fonctionnalités les plus basiques et en focalisant sur les composants un par un.

Dans l'ordre, il faut s'intéresser d'abord au composant Grille, cœur de l'application. Ensuite à au moins un GrilleLoader. On contrôle avec des affichages et des tests la stabilité du code de ces deux composants. On raffine le modèle de conception pour ces composants On construit un package par composant, dans lequel on va réaliser le composant. On génère le code une première fois pour ces package et on implémente les composants et les tests. Ensuite on synchronise les diagrammes de classe détaillant la réalisation de chaque composant et les interfaces (signatures !) sur le code.

Avec ces éléments en main, on peut construire une première ébauche d'IHM, qui affiche une IGrille. (TV01 à TV03).

On traite ensuite la mise à jour de la grille via l'IHM, un travail qui ne nécessite pas de toucher à Grille et GrilleLoader en principe. (on passe donc les tests de validation jusque TV07)

On peut alors réaliser les contraintes et les options d'aide, ou les statistiques au choix.

Pour les statistiques, on commence par le composant qui les stocke + quelques tests. On peut ensuite le lier à l'IHM, i.e. offrir un affichage (TV 13 validé). Enfin, il faut ajouter le chronomètre et réaliser l'opération testFinPartie pour permettre la notification (TV16 validé).

Pour l'aide, on commencera par un composant responsable de stocker les préférences d'aide. On le lie ensuite à l'IHM pour permettre de modifier le niveau (noob à dur). On valide via des tests qui utilisent un bouchon pour les contraintes : celui-ci porte des données qui permettent simplement de valider la cohérence des affichages.

Enfin on peut implémenter un moteur de contraintes, qui calcule réellement les suggestions/validité de la grille.

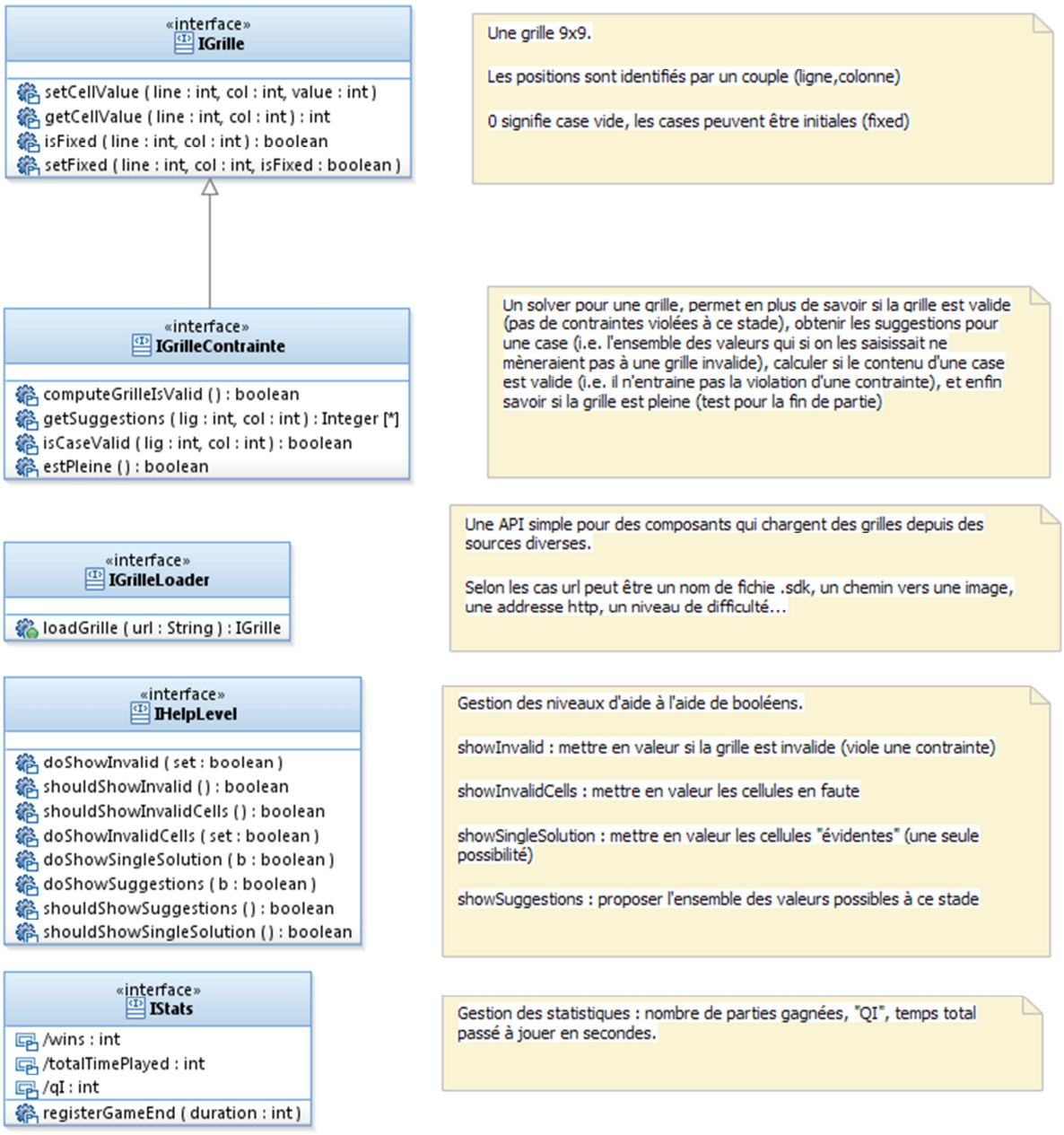
A tout moment, on peut s'occuper du composant qui charge depuis une image, dès qu'on a défini IGrilleLoader et IGrille.

Le modèle présenté dans la suite est issu d'itérations UML/code/UML appuyé par une transformation UML/Java. On garde les artefacts de modélisation qui ne donnent pas du code (composants en particulier) séparés des dossiers/package supportant la génération du code.

Architecture et Conception de iSudoku

Interfaces

Les principales interfaces identifiées pour les composants sont les suivantes.



Remarques :

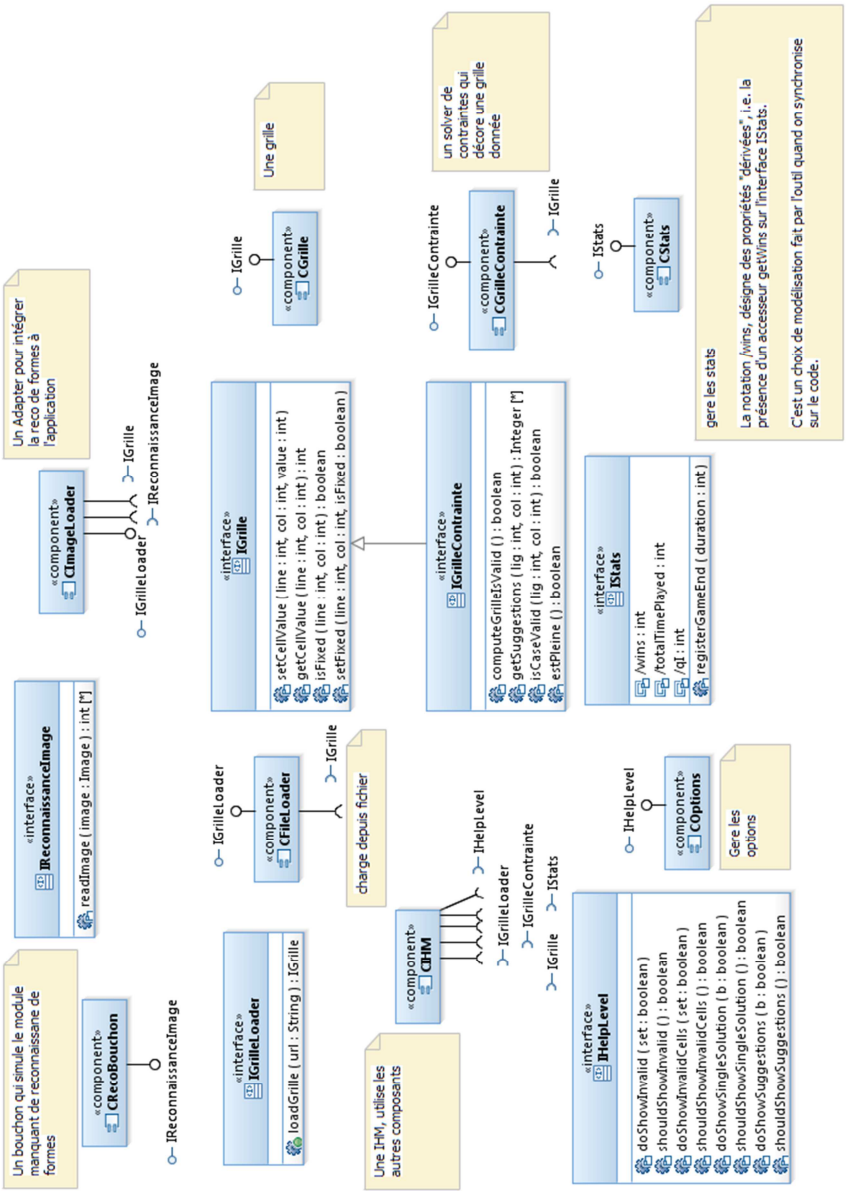
La notation /wins dans IStats dénote des propriétés dérivées, c'est un artefact de modélisation sans importance dû à la synchronisation sur le code.

L'IHM n'offre logiquement pas d'interface logicielle.

L’interfaces choisie pour IGrilleLoader est de type «Factory », i.e. elle assure la création d’objets sans citer leur type concret.

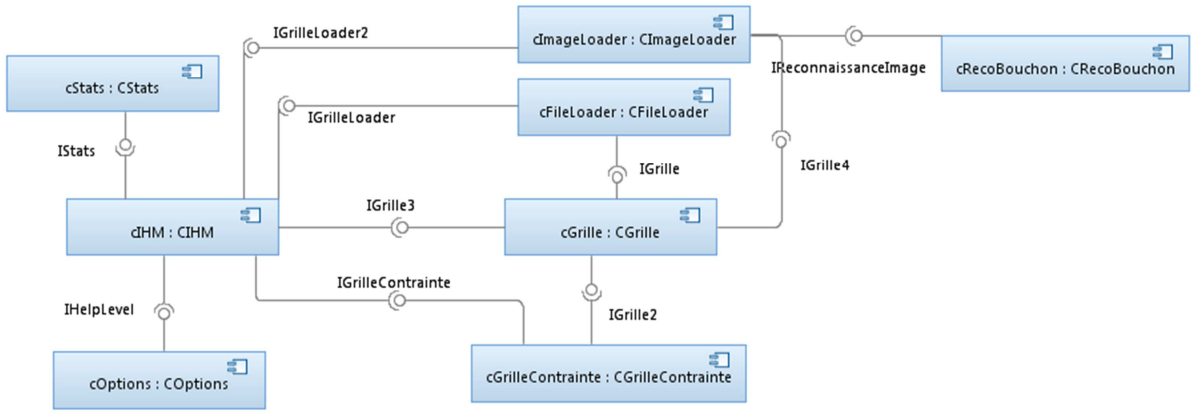
Composants

Les composants de l'application sont représentés sur ce diagramme.



Configuration nominale des composants

Les composants sont instanciés dans l'application finale selon la topologie suivante.

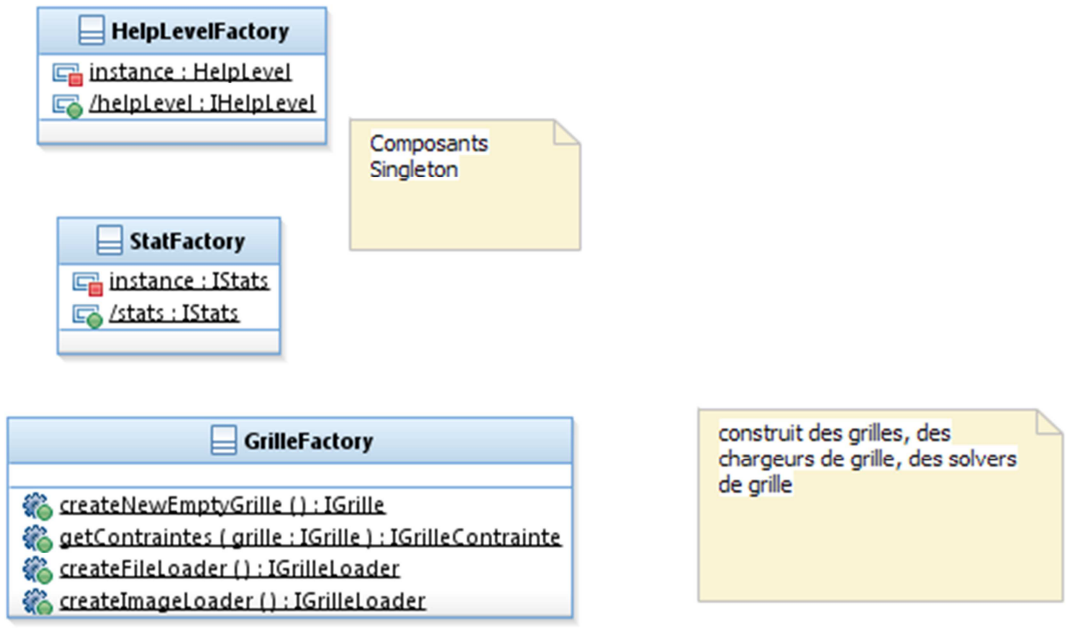


Remarques :

En pratique, il y a donc deux IGrilleLoader connectés à l'IHM. Par contre, on a une seule occurrence de CGrille à l'échelle de l'application. L'IHM porte le gros de la responsabilité d'assemblage des composants.

Factory de composant

On construit plusieurs Factory statique pour permettre l'instanciation des composants.



La factory HelpLevel et la factory Stat sont implémentées comme des Singleton. Il n'y a qu'une instance de ces composants dans l'application, et ce DP permet de facilement obtenir une référence à ces composants.

La GrilleFactory regroupe les opérations permettant de construire les diverses variantes de Grille. Notons que le GrilleLoader et le FileLoader auraient pu être aussi des Singleton.

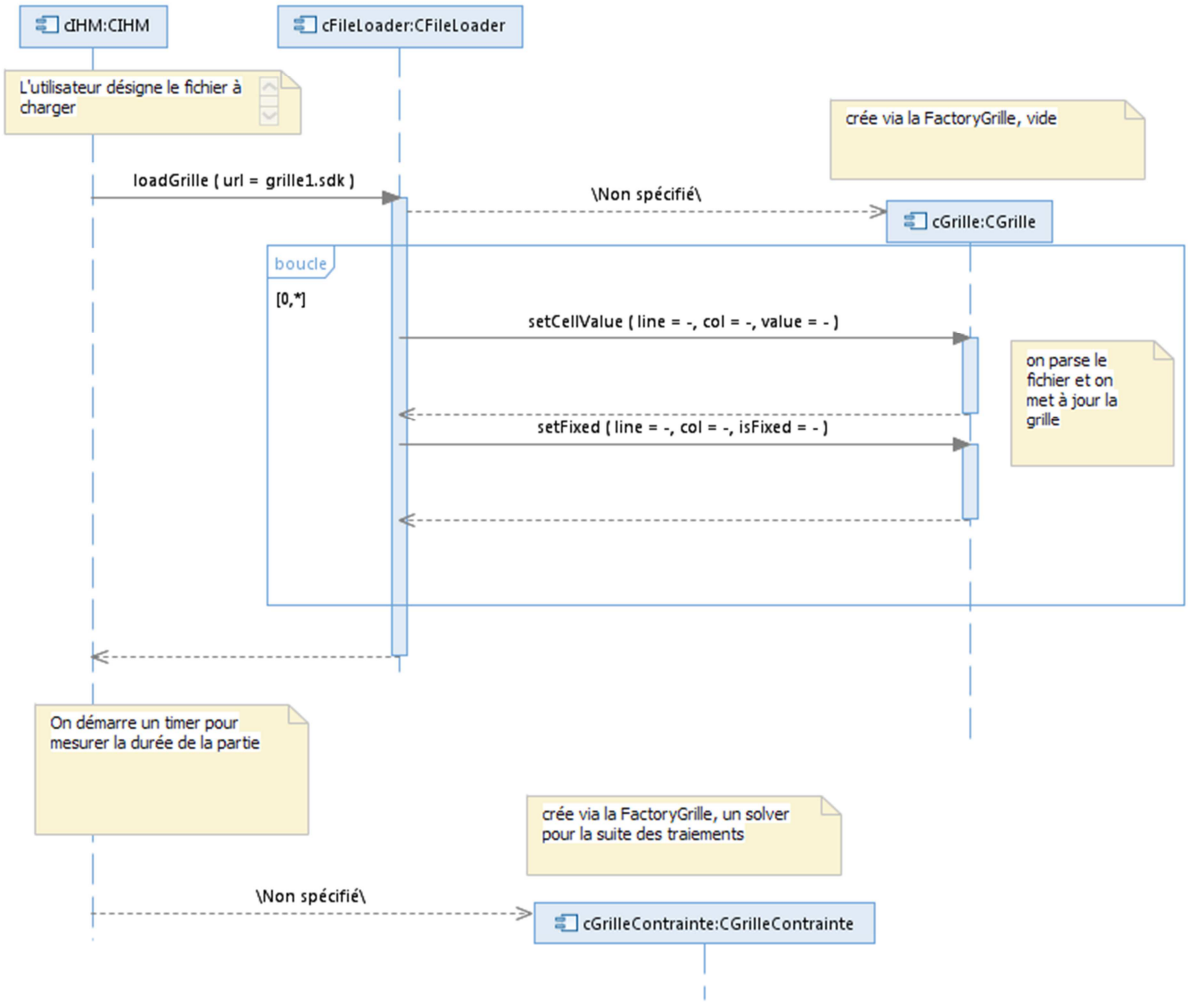
Remarques :

L'utilisation de Factory permet de cacher les types concrets qui réalisent les composants. Il n'est pas vraiment utile de déployer le DP AbstractFactory, des versions « static » (aussi appelé SimpleFactory ou StaticFactory dans la littérature) sont suffisantes.

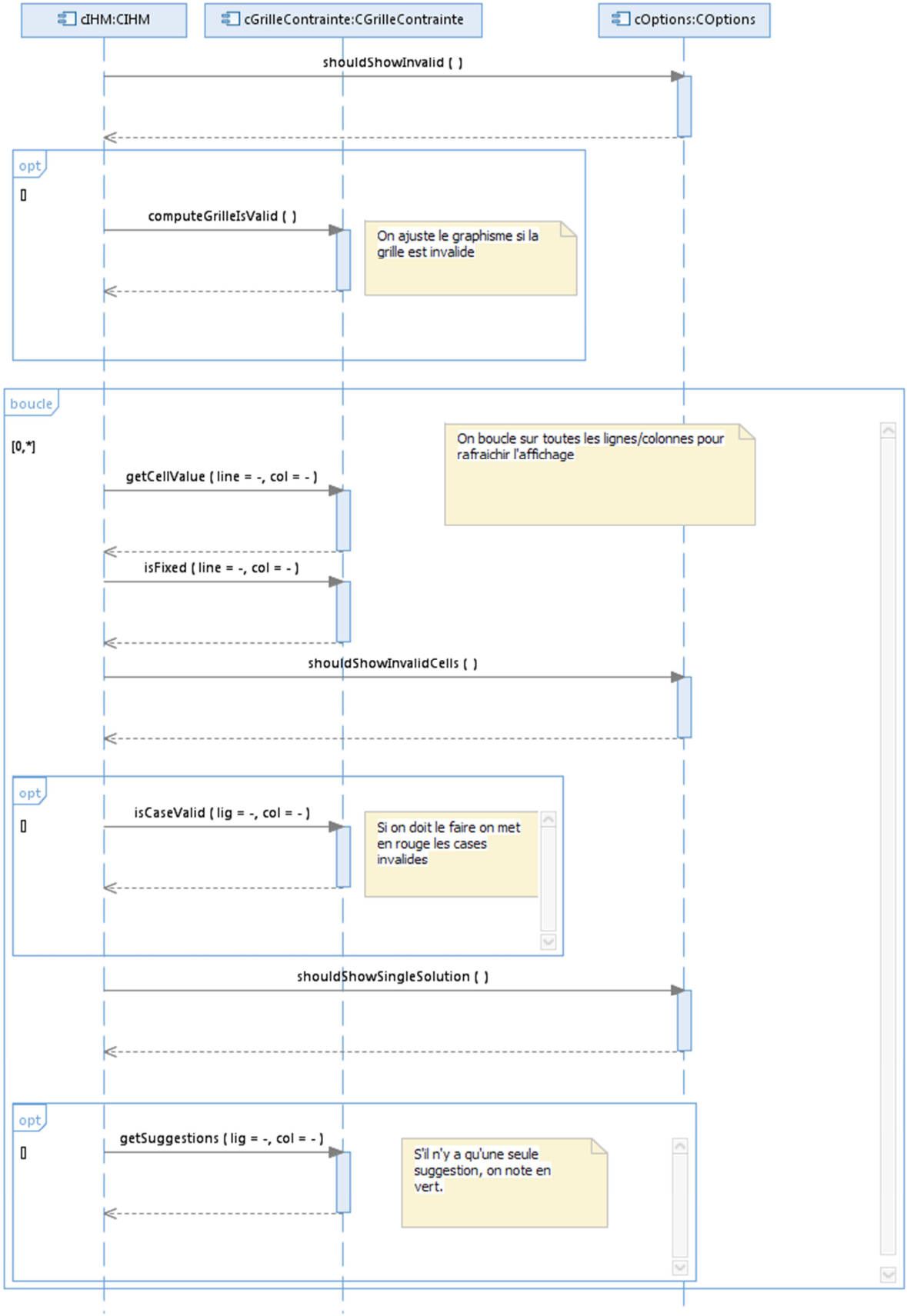
Séquences d'interaction

Les principales séquences d'interaction inter-composants sont représentées ici. On cherche à couvrir (raffiner) les séquences d'analyse.

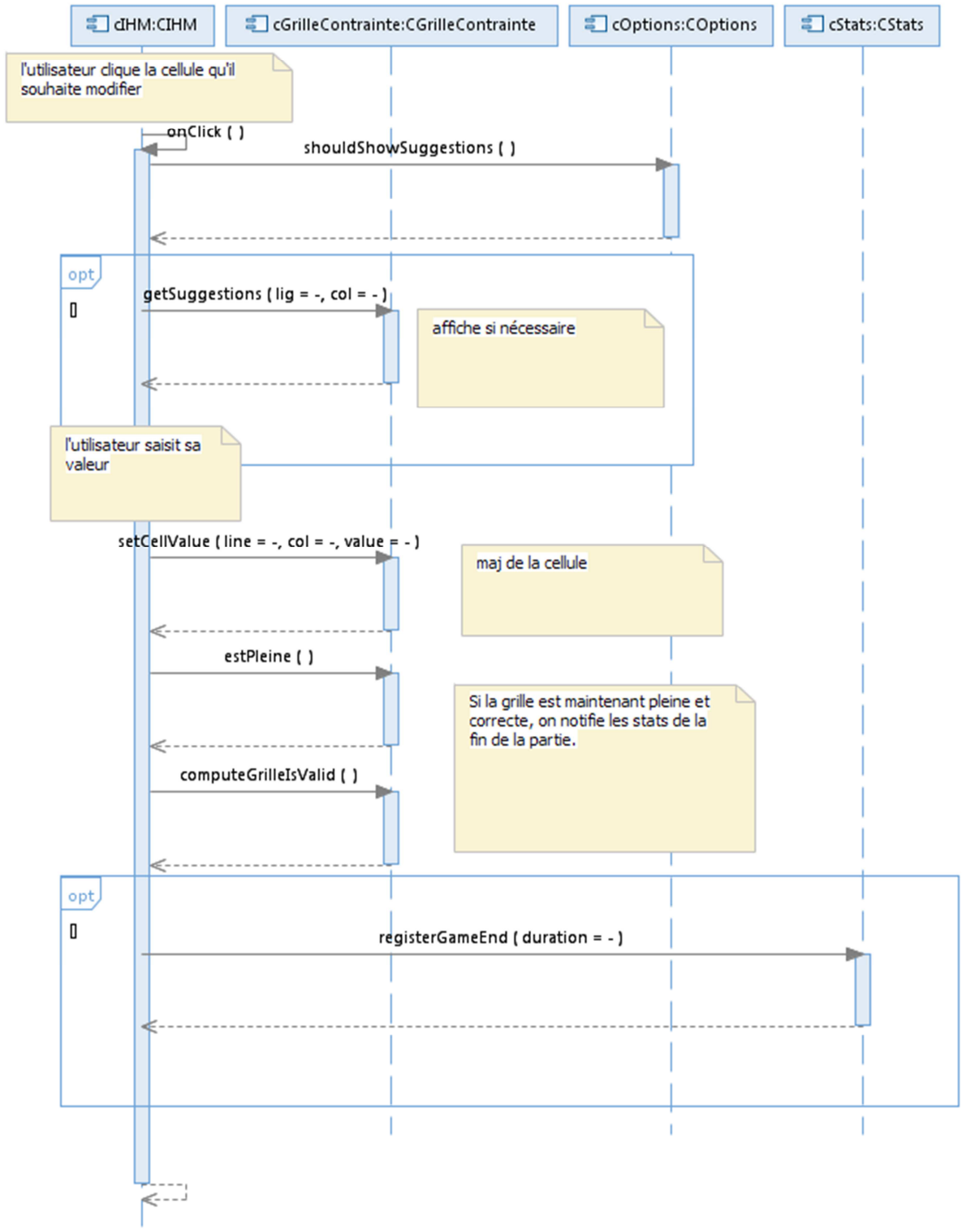
Initialisation : Chargement Fichier



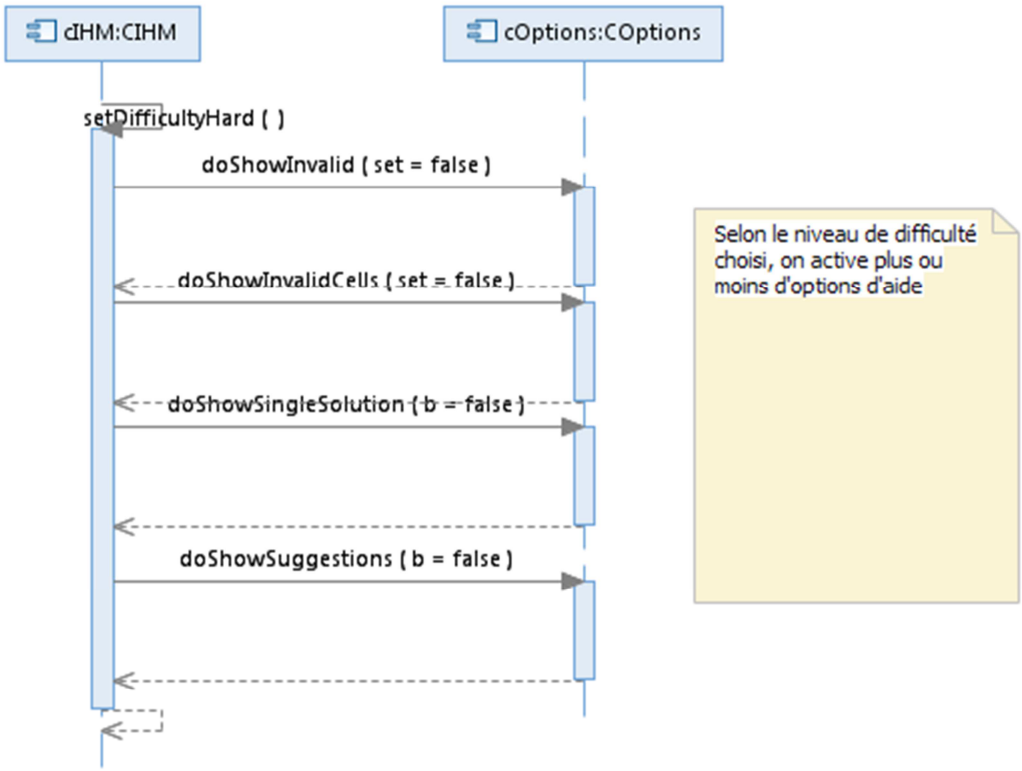
Affichage d'une grille en fonction des options



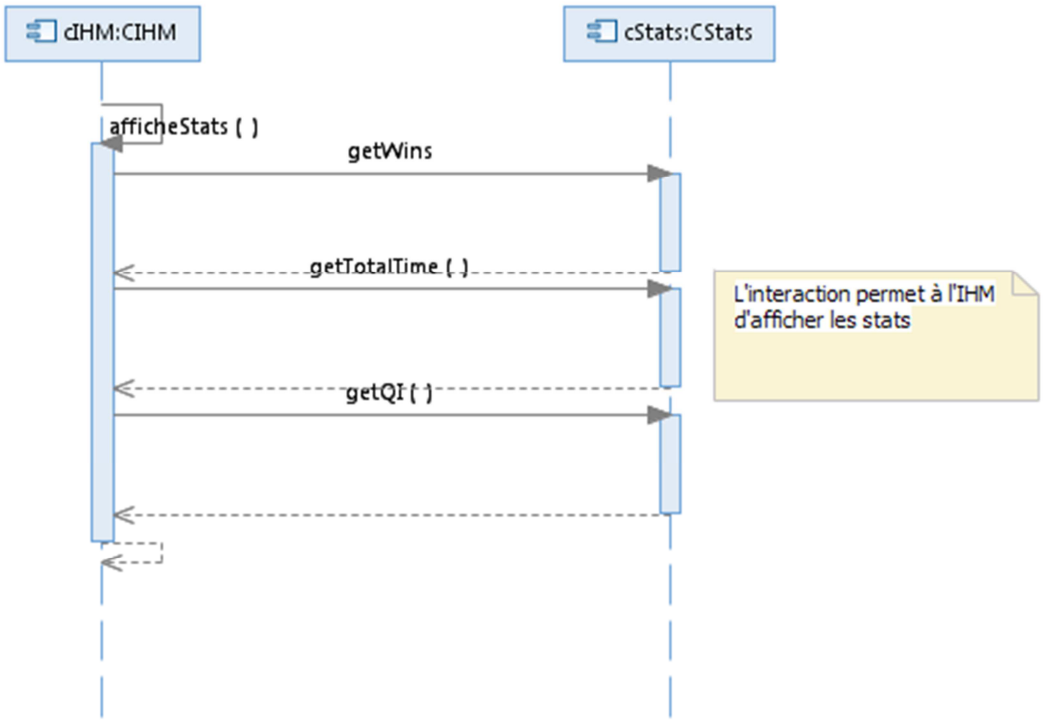
Mise à jour des cellules



Gestion des options de difficulté

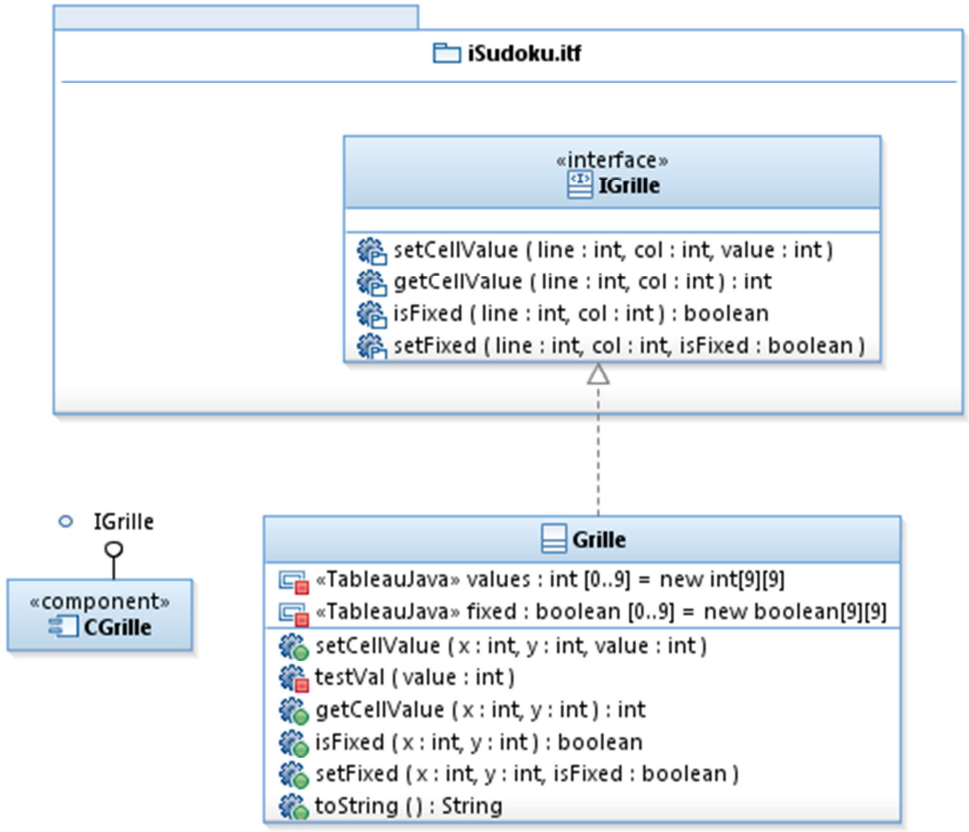


Affichage des statistiques



Conception Détaillée

Composant CGrille



Un composant chargé de gérer une grille : valeurs des cellules, certaines sont intiales.

On utilise un API (ligne,colonne) pour évier toute dépendance structurelle entre composants.

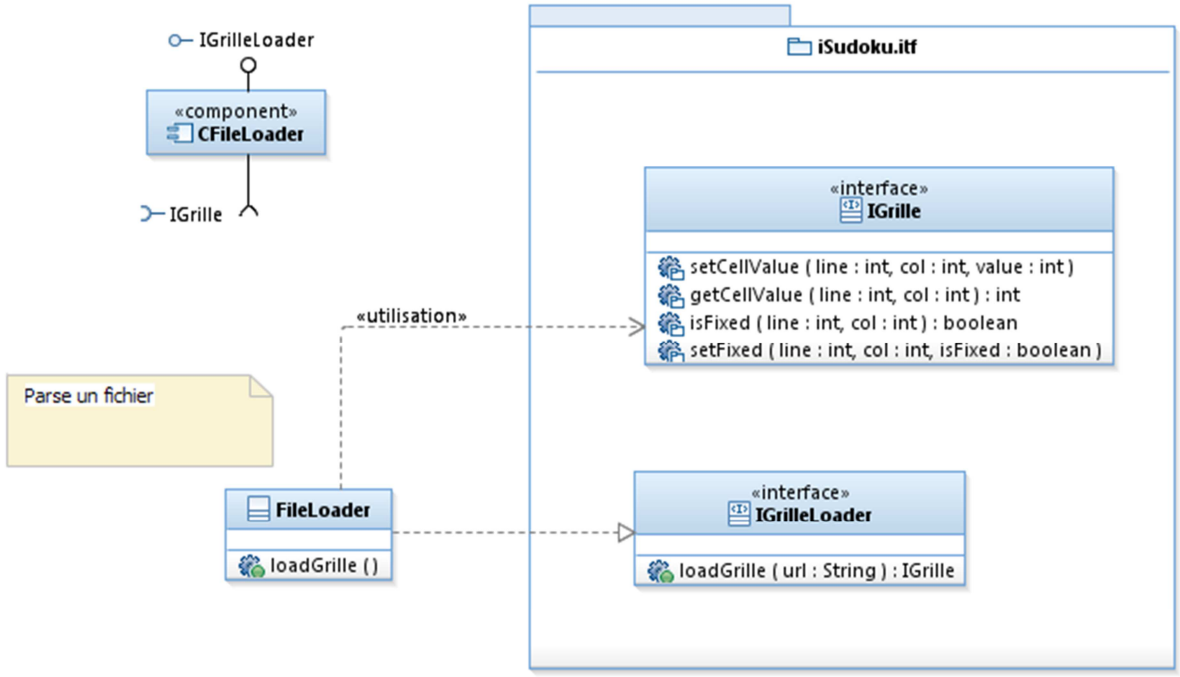
Vu l'interface, une matrice de booléens et une matrice d'entiers sont suffisants comme structures de données.

Remarques :

On s'est un peu écarté du modèle d'analyse, où l'on avait des Cellules et Valeurs. Ces concepts sont représentés de façon simplifiée, mais toute l'information identifiée en analyse est encore présente dans ce modèle.

L'API d'accès, utilise un couple (ligne,colonne) pour référer aux cellules, ce qui permet d'éviter d'exposer une éventuelle classe Cellule. Ceci permet d'éviter des dépendances structurelles des autres composants sur la Grille.

Composant CFileLoader



Un composant dont la responsabilité est de charger des grilles depuis des fichiers.

Le format de fichier est composé de 81 valeurs, on ignore les espaces/retour chariot. 0 signifie une cellule initialement vide.

Par exemple, grille1.sdk :

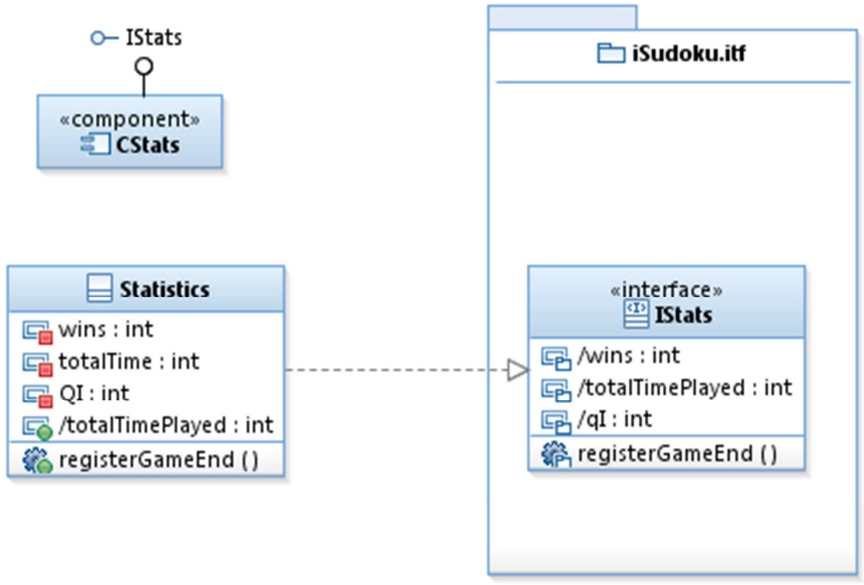
```
530070000
600195000
098000060
800060003
400803001
700020006
060000280
000419005
000080079
```

Remarques :

Le format de fichier n'est pas défini par le cahier des charges, à vous d'en proposer un. Celui proposé ici est simple à parser, mais ne permet pas de stocker une partie « en cours » (vu qu'on ne distinguerait pas les valeurs saisies par l'utilisateur des valeurs initiales). On n'a pas identifié le besoin en Analyse de stocker autre chose que des grilles donc on les choix sont cohérents.

Composant CStats

Un composant chargé de gérer les statistiques des parties.



La notation / désigne des propriété dérivées, i.e. RSA traduit les getXXX en propriété XXX.

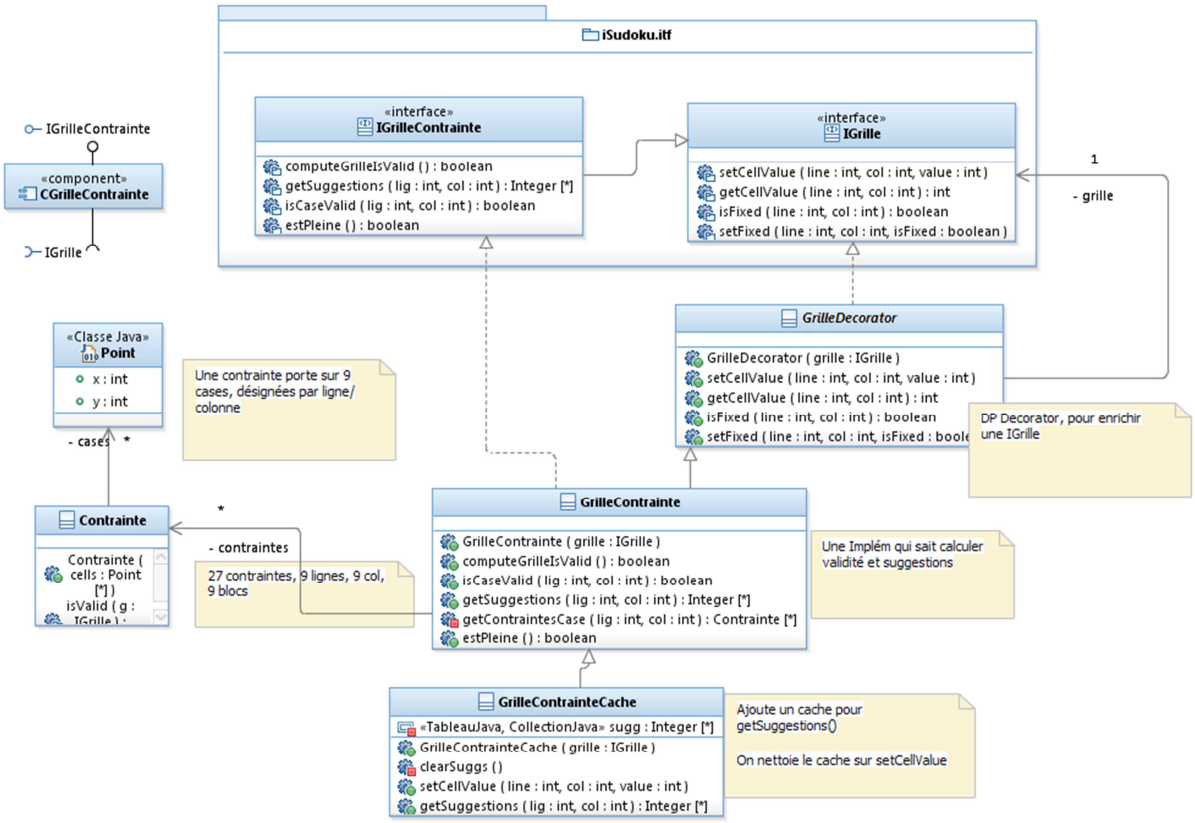
Composant bout de chaîne qui stocke les statistiques du Joueur.

Remarques :

Logiquement, ce composant devrait assurer la persistance des données d'une session de jeu à l'autre, mais ce mécanisme n'a pas été implanté dans le prototype.

Composant CGrilleContrainte

Un solver de contraintes, qui décore une IGrille.



Le lien sur IGrille est matérialisé par des Point, i.e. une paire (ligne,colonne)

On a implémenté un cache pour « getSuggestions(lig,col) ».

Chaque contrainte porte sur 9 cases. Il y a 27 contraintes au total (9 lignes, 9 colonnes, 9 blocs).

Remarques :

Ce composant est moins complexe en structure que le composant d'IHM, mais porte le cœur algorithmique de l'application, et donc n'est pas simple à concevoir et implémenter.

On commence par les méthodes les plus simples : tester la validité d'une grille. Pour cela on introduit les contraintes ; si au moins une contrainte est invalide (deux cellules sur lesquelles porte la contrainte contiennent la même valeur) la grille est globalement invalide.

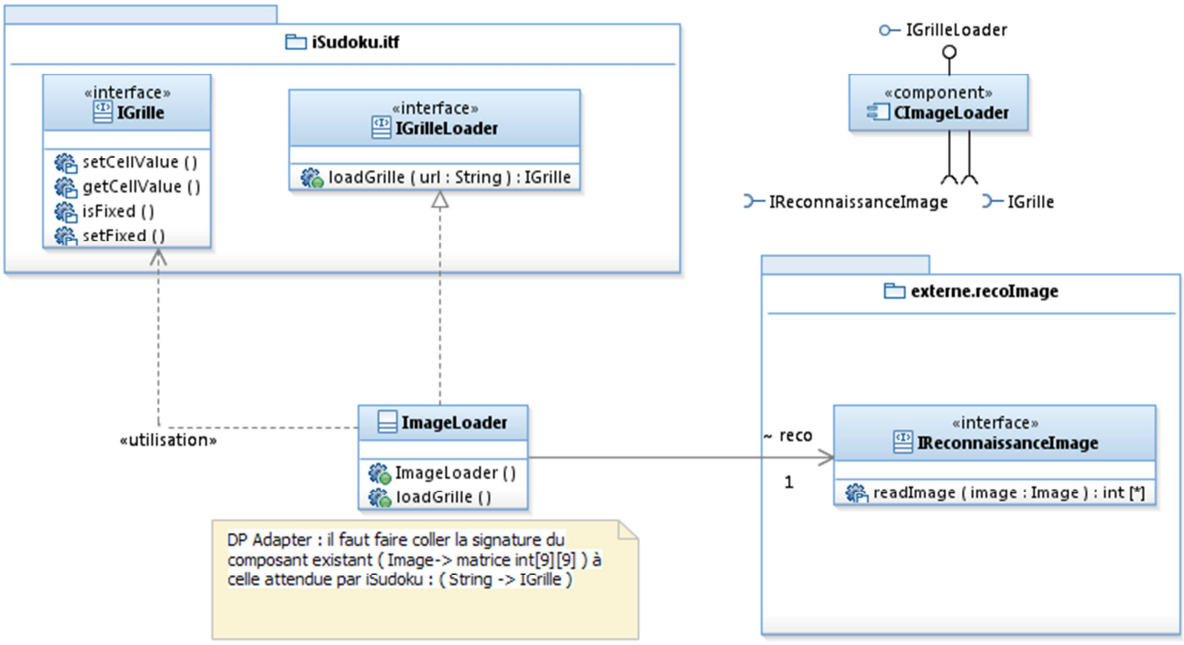
On peut ensuite implémenter « getSuggestions ». Cela nécessite de savoir à quelle ligne, colonne et bloc appartient une case donnée. La méthode **getContraintesCase(lig,col) :Contrainte[*]** répond à ce besoin. On calcule ensuite les suggestions : c'est l'ensemble des valeurs 1 à 9 privé des valeurs déjà présentes sur la ligne, colonne et bloc à laquelle appartient la case. La valeur déjà dans la case (s'il y en a une) n'est pas prise en compte.

Enfin, avec getSuggestions, les autres méthodes s'implémentent assez aisément ; une cellule est invalide si la valeur qu'elle contient n'est pas dans getSuggestions OU qu'il n'y a pas de valeur dans la cellule et que getSuggestions rend l'ensemble vide.

Comme cette façon d'implémenter pousse à des invocations très fréquentes de getSuggestions, on a ajouté un cache pour cette opération. Plutôt qu'un HashMap, choix standard pour implémenter un cache, on a fait un cache par case de la grille (i.e. on stocke la matrice de suggestions au fil de son calcul). Ce cache est invalidé dès qu'on met à jour une cellule (donc pas de calcul incrémental de la mise à jour, ce qui aurait été possible).

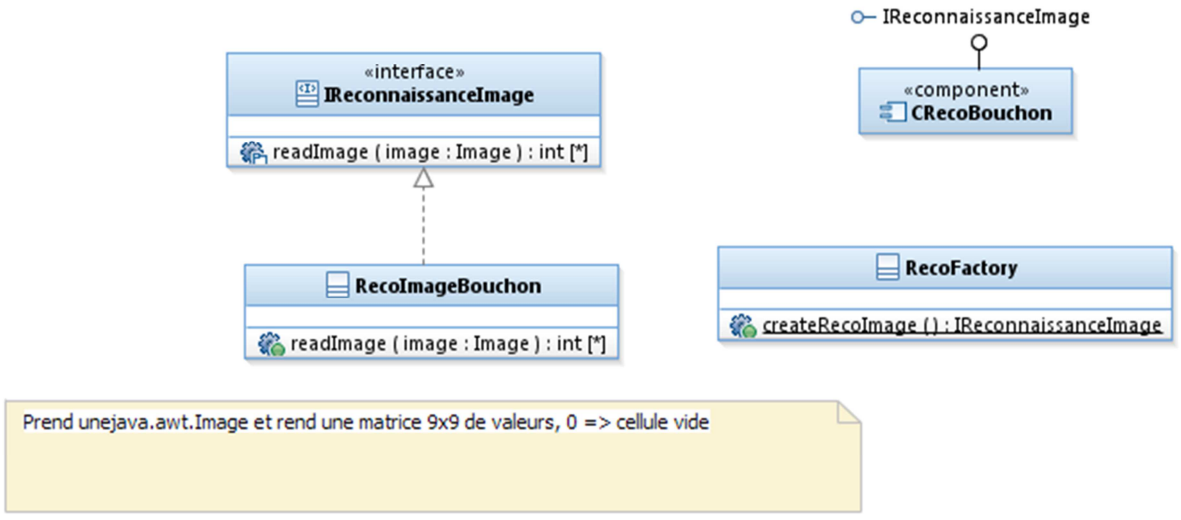
Composant CLoadImage

Un composant qui adapte la signature du composant fourni CRecoFormes.



Le composant CRecoFormes est supposé rendre des matrices d'entiers. Il faut donc adapter les signatures à IGrilleLoader.

Il est réalisé par un bouchon à ce stade.



Remarques :

Le CdC ne donne pas l'API de l'hypothétique composant de reconnaissance de formes, mais ce qui est fait ici est typique d'une réutilisation de l'existant (DP Adapter).

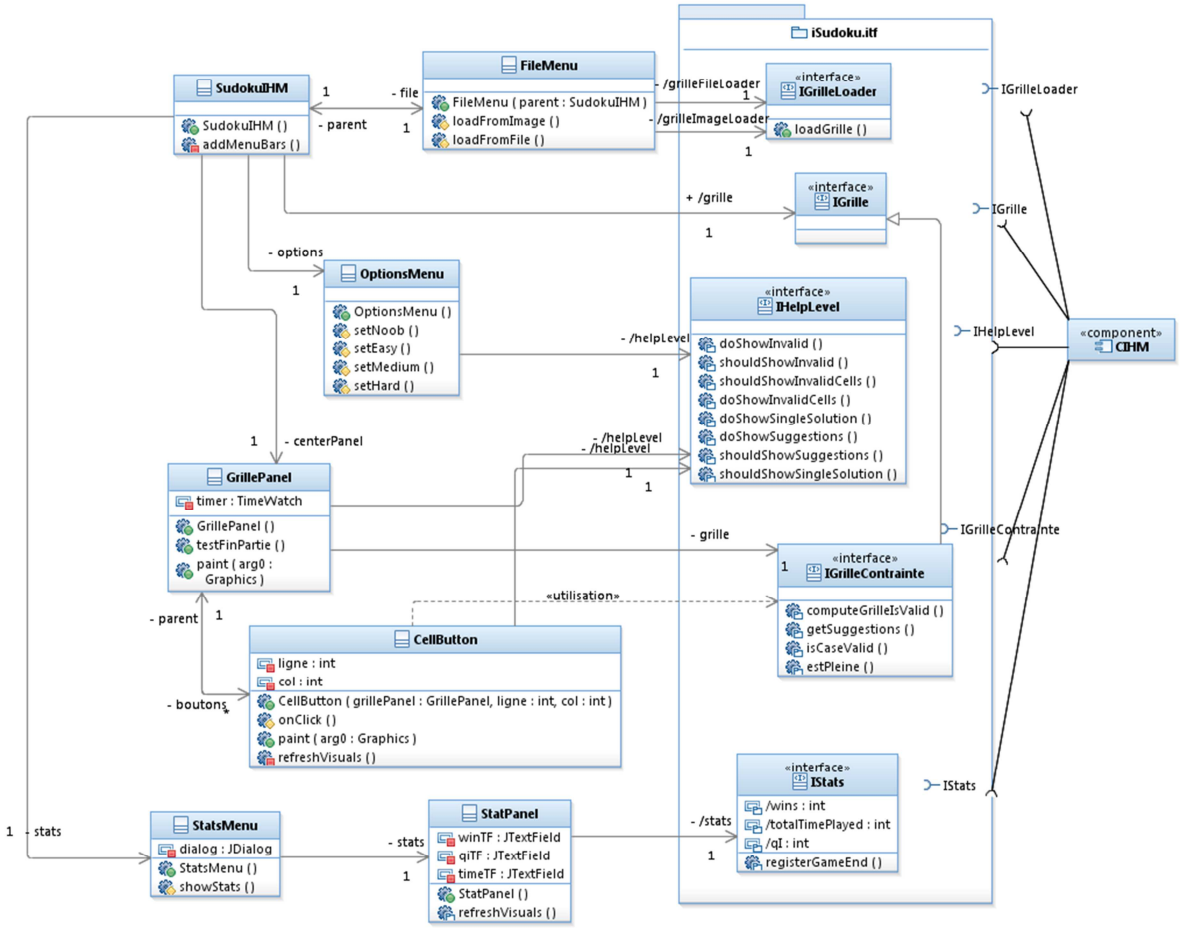
On a opté pour une API pour la reconnaissance de formes qui n'utilise aucune de nos classes et interfaces, logiquement. Ici, une fonction qui prend une java.awt.Image et rend une matrice de valeurs.

Pour tester, on crée un bouchon, qui rend pour « grille1.jpg » la matrice de valeurs souhaitées. En fait il rend toujours la même matrice. Les valeurs à rendre sont placées en « dur » dans le bouchon.

Composant CIHM

Ce composant relativement complexe gère l'IHM de l'application.

On a séparé les parties de l'interface graphique qui gèrent les différents aspects.



On voit apparaitre trois menus (Stats, Options, Load) qui étendent javax.swing.JMenu. On a créé des méthodes représentant les actions utilisateur sur ces menus.

On a également un GrillePanel, qui étend javax.swing.JPanel et gère l'affichage de la grille. GrillePanel porte une horloge pour mesurer le temps de jeu ; on notifie IStats quand la partie est finie (lien GrillePanel sur IStats non représenté). Ce JPanel porte aussi 81 CellButton, classe qui étend javax.swing.JButton. OnClick est déclenchée quand on clique sur le bouton et ouvre une boite de dialogue (non représentée) pour la saisie. RefreshVisuals met à jour l'affichage du bouton (mise en valeur de la cellule + suggestions si le niveau d'aide l'exige).

Remarques :

Globalement on retrouve dans cette conception détaillée les dépendances indiquées au niveau du composant CIHM. Au niveau des sources, on a séparé en sous-package (principal, grille, options, stats), sans aller jusqu'à faire des sous-composants. Le menu de chargement possède une référene sur son père (SudokuIHM) pour pouvoir faire un « setGrille » dessus avec la grille nouvellement chargée.

Notons que la réalisation de ce composant est la plus technique ; elle repose sur une bonne connaissance de l'API spécifique de Swing. L'organisation globale est cependant indépendante de Swing, on aurait la même organisation avec un autre framework d'IHM (en C# ou avec le SWT d'Eclipse par exemple).