Analyse d'un jeu de "Minimum Sudoku"

Jean-Etienne Poirrier

5 juin 2007

Table des matières

1 Introduction au jeu de Sudoku

Le Sudoku est un jeu introduit dans le monde occidental, il y a une trentaine d'année, et qui a connu récemment un grand succès. Une recherche sur le mot "Sudoku" dans un moteur de recherche bien connu retourne des liens vers plus de 61500000 pages web.

Dans ce travail, après un bref passage en revue de l'histoire du Sudoku et de ses règles, nous introduirons une version spéciale du jeu : "Minimum Sudoku". Cette version permet, tout en respectant les règles initiales, d'introduire une nouvelle façon de présenter le jeu à l'utilisateur.

1.1 Histoire

Un certain nombre de jeux de réflexion basés sur la présence et la répétition (ou non) de chiffres dans une grille étaient déjà répandus, au dix-neuvième siècle. Les règles de ces jeux ne correspondaient cependant pas à celle du Sudoku, discrètement inventées par Will Shortz. Les premiers jeux sont publiés à la fin des années 1970 aux Etats-Unis sous le nom "Number Place" ("la place des nombres"). C'est lors de son introduction au Japon, dans le milieu des années 1980, que le jeu pris son nom actuel de "Sudoku" (abréviation d'une phrase japonaise signifiant "les nombres doivent être seuls").

A la fin des années 1990, un néo-zélandais remarque le jeu, développe un programme pour produire automatiquement des problèmes (grilles) et les propose à la presse britannique, friande de mots croisés et autres jeux de réflexion. C'est à partir de cette introduction que l'engouement que nous connaissons actuellement est né. Cet engouement est tel que des jeux de Sudoku sont présents

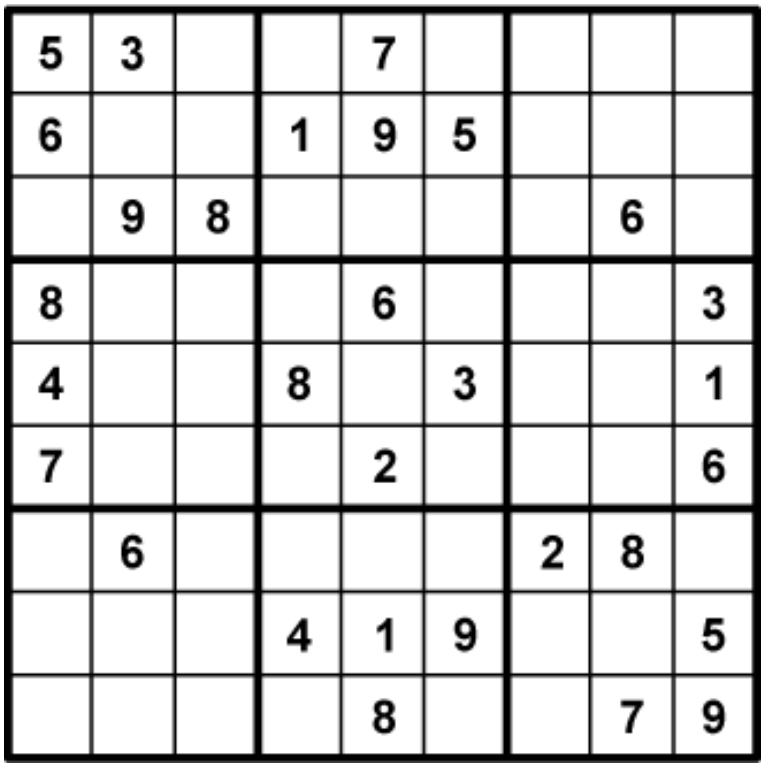

FIG. 1 – Exemple de grille de départ du jeu de Sudoku

dans les quotidiens, sur les chaînes de télévision, que des concours nationaux et internationaux sont organisés. Un grand nombre d'implémentations du jeu sont présentes sur média informatisé, que ce soit sous forme de programme indépendant (sur ordinateurs, consoles de jeux et même téléphones portables) ou présenté sur le web.

1.2 Principe

Le Sudoku est un jeu de logique où le joueur doit placer des nombres dans une grille. L'objectif est de remplir une grille de 9 cases sur 9 de telle manière que chaque colonne, chaque ligne et chacune des "boîtes" de 3 cases sur 3 ne contiennent qu'un seul exemplaire des nombres entre 1 et 9. Comme aux échecs, une case jouée (une pièce ou un chiffre placé dessus) ne peut être modifiée. Le jeu commence avec une grille partiellement complétée (voir figure 1). Le jeu se termine lorsque toutes les cases possèdent un nombre (ou que le joueur se rend compte de l'impossibilité de compléter correctement la grille).

Ces règles peuvent être synthétisées, reformulées de la manière suivante, en face d'une grille de départ de Sudoku :

1. chaque ligne et chaque colonne de la grille doit être remplie de nombres de 1 à 9, chacun en un seul exemplaire ;
2. chaque petite grille (3×3 cases) doit également être remplie de nombres de 1 à 9, chacun en un seul exemplaire ;
3. le joueur ne peut pas changer les nombres qu'il/elle a déjà placé.

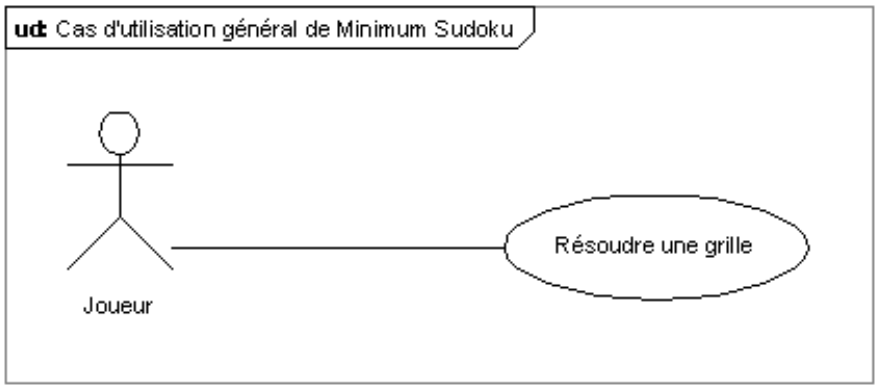

FIG. 2 – Cas d'utilisation le plus général du jeu

1.3 Minimum Sudoku

Comme souligné dans l'introduction, le web regorge de références sur le Sudoku, de différentes stratégies de résolution des problèmes posés et d'implémentations informatisées plus ou moins résultant d'analyses préalables. Un problème particuliers a retenu notre attention : le "Minimum Sudoku" [1] de Gordon Royle, professeur dans une université australienne.

Gordon Royle s'est demandé quel est le plus petit nombre de cases complétées dans une grille de départ aboutissant à une solution unique. Pour le moment, il n'existerait aucune grille de 16 nombres au départ ne possédant qu'une seule solution. Le nombre minimal de cases complétées au départ pour une solution unique est donc 17. Gordon Royle a collecté plus de 40000 grilles de départ ne contenant que 17 nombres et une seule solution. L'ensemble de ces configurations de départ sont disponibles sur la page web de Minimum Sudoku.

L'analyse qui suit va s'atteler à modéliser un jeu de Minimum Sudoku : un jeu ne présentant que 17 cases pré-remplies au départ, permettant au joueur de résoudre le problème et validant sa solution à la fin. Cette particularité du jeu de Sudoku original va permettre de simplifier le processus de validation final : la solution du joueur correspond à la solution unique, connue de l'ordinateur, ou est erronée.

2 Analyse

2.1 Cas d'utilisation traités

Les cas d'utilisation traités ont déjà été décrit partiellement dans les sections précédentes.

Le cas le plus général pour lequel un joueur va lancer le programme est qu'il/elle veut résoudre une grille (figure 2).

A l'intérieur du jeu, plusieurs cas d'utilisation sont possibles. Ces cas sont repris à la figure 3. Le joueur peut vouloir :

1. demander une grille de départ particulière ou au hasard (parmi l'ensemble des grilles résolues présentes)

[1] Une description du Minimum Sudoku ainsi que l'ensemble des données et des considérations mathématiques sont disponibles sur la page de Gordon Royle : http ://people.csse.uwa.edu.au/gordon/sudokumin.php

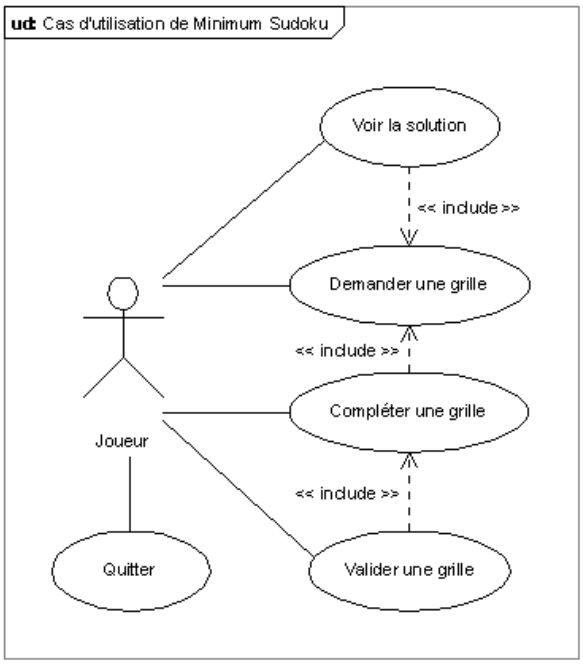

FIG. 3 – Cas d'utilisation du jeu en lui-même

2. compléter une grille donnée par le programme
3. valider une grille qu'il aura complété
4. voir la solution d'une grille
5. quitter le programme

Évidemment, certains des cas présentés nécessitent la complétion préalable d'autre cas. Chacun de ces cas d'utilisation sera représenté par une action possible dans l'interface utilisateur.

La section 2.3 reprend des diagrammes de la séquence de jeu la plus probable ainsi que les différents états rencontrés.

2.2 Diagramme de classes

A première vue, d'après la description du jeu donnée à la section 1.3, nous avons besoin de deux classes : un nombre et une boîte (figure 4).

Un objet de classe `Nombre` possédera un entier ainsi que ses accesseurs. Nous prenons ici le parti de stocker l'entier dans un objet pour deux raisons. D'abord, le jeu de Sudoku peut utiliser n'importe quoi pourvu que ces choses soient présentent en 9 entités différentes (on pourrait utiliser, par exemple, 9 couleurs différentes, 9 lettres différentes, 9 pictogrammes différents, etc.). Par l'utilisation d'un objet, il sera plus aisé, dans la suite, de remplacer le contenu de cet objet par autre chose qu'un entier. L'exposition en dehors de la classe restera plus ou moins la même. Ensuite, il est plus aisé de reporter le traitement de nombres entiers dans une classe plutôt que de l'implémenter dans le corps du jeu.

Un objet de classe `Boite` représente une case de la grille. Cette case possède un `Nombre` contenant un entier entré par le joueur ou non. Si le joueur a entré un nombre, la case n'est plus éditable. Cette case possède un second objet `Nombre` contenant l'entier de la solution. Chaque

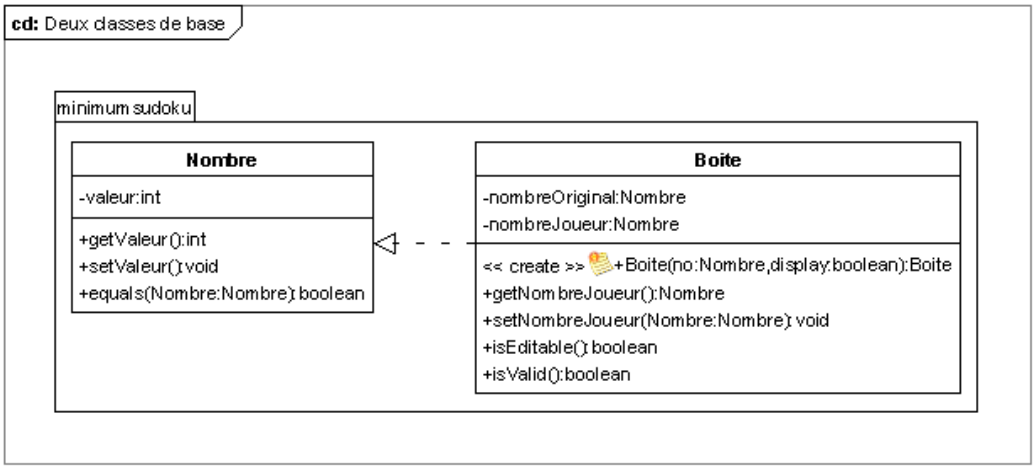

FIG. 4 – Premier diagramme des classes utilisées dans Minimum Sudoku

case est alors capable de se valider elle-même, en comparant le `Nombre` de la solution avec celui du joueur. C'est ici que la stratégie de Minimum Sudoku diffère d'un jeu de Sudoku normal : là où le programme aurait du vérifier chaque ligne, chaque colonne et chaque petite grille de 3×3 cases, ce programme va pouvoir se contenter de vérifier l'entier contenu dans chaque case à la solution (unique).

On peut remarquer qu'ici, une grille de 9×9 cases est composée de différentes grilles de 3×3 cases contenant chacune des boites seules. Chacune des boîtes peut se valider seule. Mais nous pouvons extrapoler la validation à chaque grille de 3×3 cases (si chacune des 9 cases est valide, la petite grille est valide) et même à la grille de 9×9 cases (si chacune des 9 petites grilles est valide, la grille de jeu est valide). Nous pouvons donc transformer la classe `Boite` pour la rendre "abstraite" dans le sens où elle pourrait représenter une des 81 cases ou une petite grille de 9 cases ou la grille complète du jeu. Le processus de validation s'occuperait alors d'une `Boite` à valider, sans se soucier de savoir si cette boîte est une case de jeu ou une grille, composition de cases (ou même d'autres grilles).

Nous pourrions dès lors utiliser ici le **motif de conception composite**. L'objectif du motif composite est de permettre aux clients de traiter de façon uniforme des objets individuels et des compositions d'objets. Un "composite" est un groupe d'objet contenant aussi bien des éléments individuels (aussi appelés "feuilles" ou "*leaf*" en anglais) que des éléments contenant d'autres objets. Dans le paragraphe suivant, nous aborderons la transformation de l'objet `Boite` de la figure 4 en un motif de conception composite.

La première étape de la transformation sera la création d'une interface `BoiteInterface` ne contenant que les déclarations de deux méthodes (`isEditable()` et `isValid()`). Ensuite, nous créons une classe `BoiteFeuille` (fonctionnellement équivalente à la feue `Boite`), implémentant la `BoiteInterface`. Finalement, nous créons une classe `BoiteComposite` qui contiendra une collection d'objets de type `BoiteInterface` (ce qui permettra de stocker des `BoiteFeuille` ou des `BoiteComposite`). Les relations entre les différentes classes de ce motif sont schématisées à la figure 5.

Dans la figure 4, la classe `Nombre` pose également un problème. En effet, deux objets `Nombre` vont être créés pour chaque case ; au total, une grille de 9×9 cases contiendra 162 objets ! Il se pourrait que la maintenance d'un tel nombre d'objets pose un problème de performances.

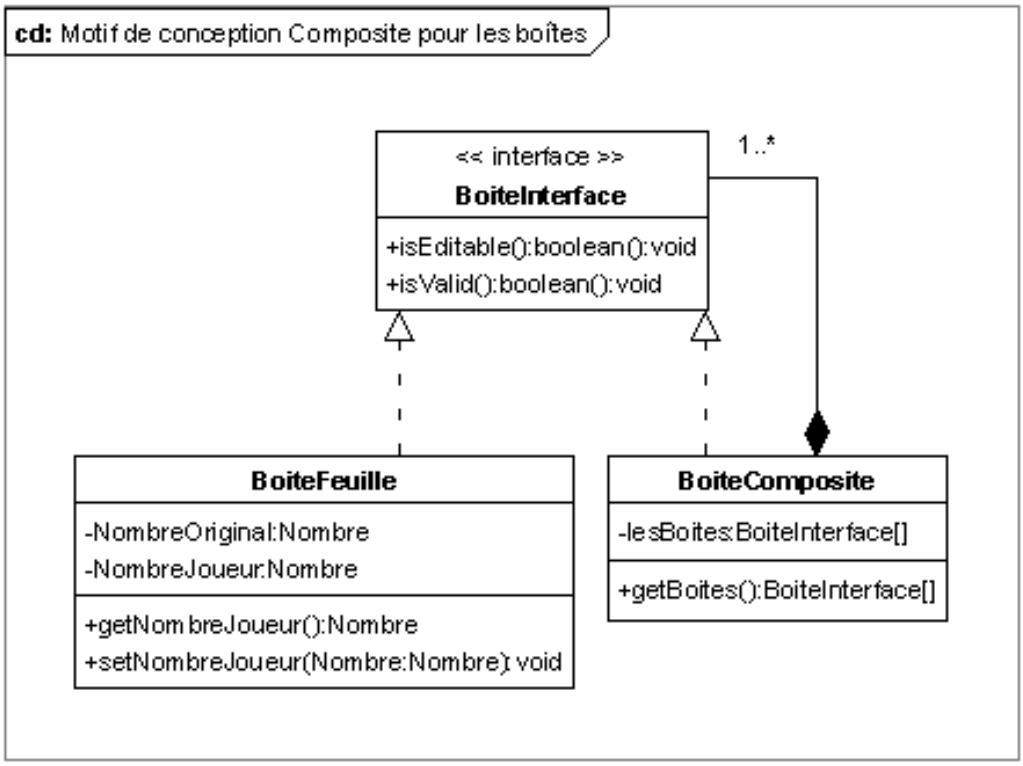

FIG. 5 – Classes et relations entre les différents éléments du motif de conception composite relatif aux boîtes

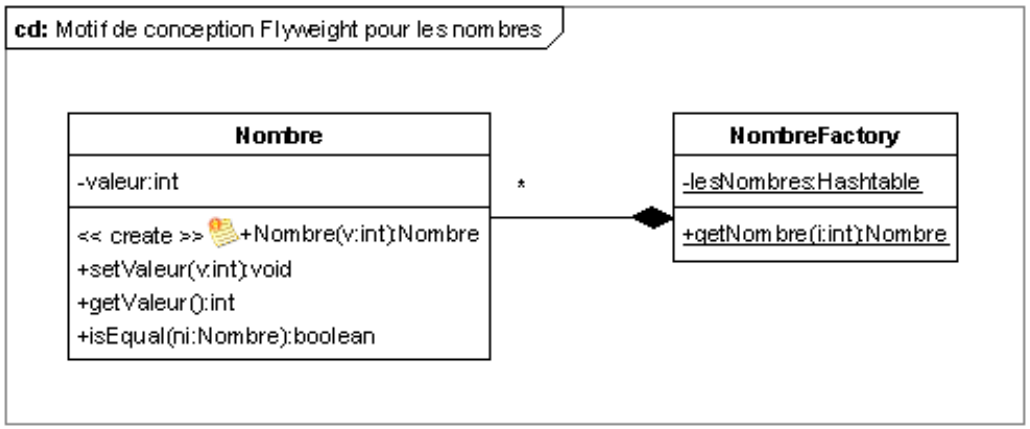

FIG. 6 – Classes et relations entre les différents éléments du motif de conception flyweight relatif aux nombres

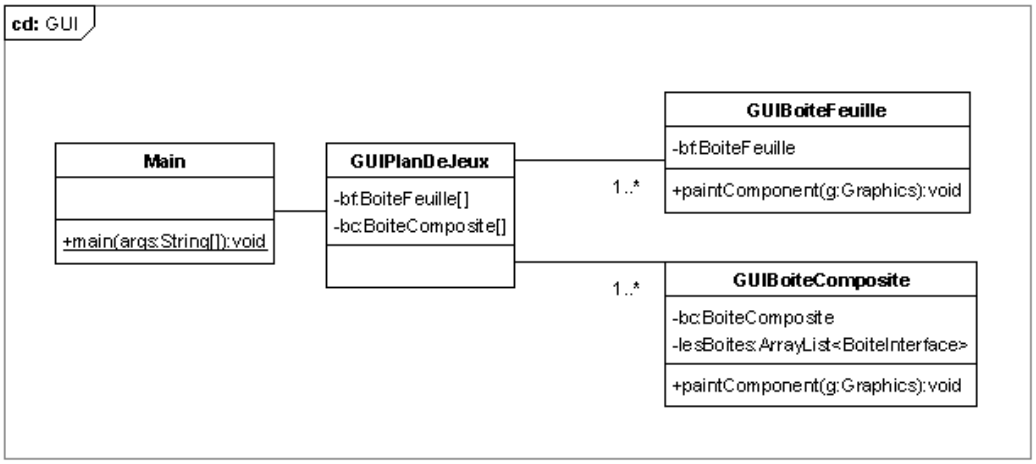

FIG. 7 – Schéma des classes d'affichage et de leurs variables et fonctions principales

Le **motif de conception flyweight** va nous aider à réduire le nombre d'instances de la classe `Nombre`. L'objectif du motif flyweight est d'utiliser le partage pour supporter efficacement un grand nombre d'objets à forte granularité. On va donc réduire le nombre d'instances en les rendant "partageables".

L'objet `Nombre` de la figure 4 va rester le même. Par contre, on va lui associer une "usine à nombres" (qui n'est pas vraiment un motif de conception *factory*). Les objets qui voudront utiliser une instance de *Nombre* seront obligés de passer par cette instance de `NombreFactory` et demander le nombre voulu. La fonction de cette dernière classe créera des instances de `Nombre` à la demande. Mais, au final, seules 9 instances de `Nombre` seront créées. Les relations entre les différentes classes de ce motif sont schématisées à la figure 6.

Un moteur de jeu sans interface utilisateur pourrait se satisfaire et fonctionner correctement avec les seules classes définies ci-dessus. La seconde partie de cette section sera consacrée aux classes dédiées à l'interface utilisateur (figure 7).

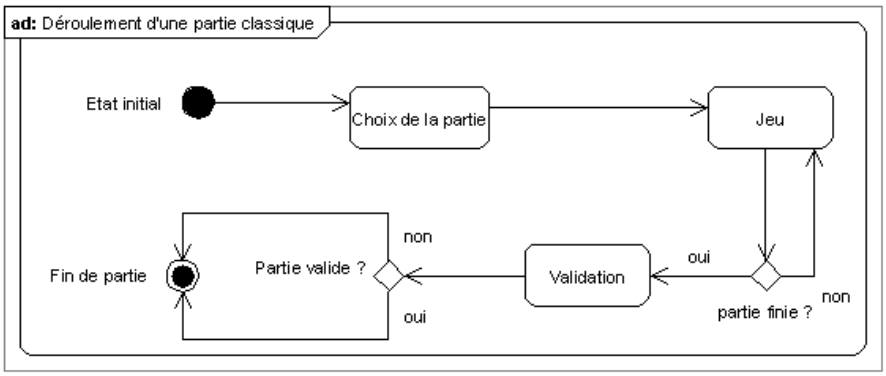

FIG. 8 – Schéma d'activité du jeu le plus probable

Les classes `GUIBoiteFeuille` et `GUIBoiteComposite` seront responsables de l'affichage des classes `BoiteFeuille` et `BoiteComposite` respectivement. La principale fonction de ces classes sera `paintComponent()` servant à dessiner le contenu de la classe.

Une instance classe `Main`, point d'entrée dans le programme, va permettre de lancer une instance de la classe `GUIPLanDeJeux` qui s'occupera de la mise en place de l'interface graphique ainsi que de la grille de jeu.

2.3 Diagrammes du jeu le plus probable

La séquence de jeu la plus probable (figure 8) commence par un choix de partie et le joueur peut passer au jeu de suite. Tant que la partie n'est pas finie, le jeu continue. Si la partie se termine, l'activité de validation prend le relais. Que la partie soit valide ou pas, le programme met fin à la partie mais reste dans l'interface graphique (le message résultant de la validation change en fonction de cette dernière).

Faire appel à un motif de conception composite pour l'utilisation des différentes boîtes permet de parcourir de manière séquentielle les appels à la validation. Par exemple, pour un jeu de de 3×3 cases (une boîte composite), le processus de validation pourrait être schématisé par un diagramme de séquence (figure 9).

3 Conclusion

Un cas particuliers du jeu de Sudoku a été présenté ici. L'analyse de ce cas a introduit deux motifs de conception et a permis de séparer le moteur de jeu de son affichage graphique.

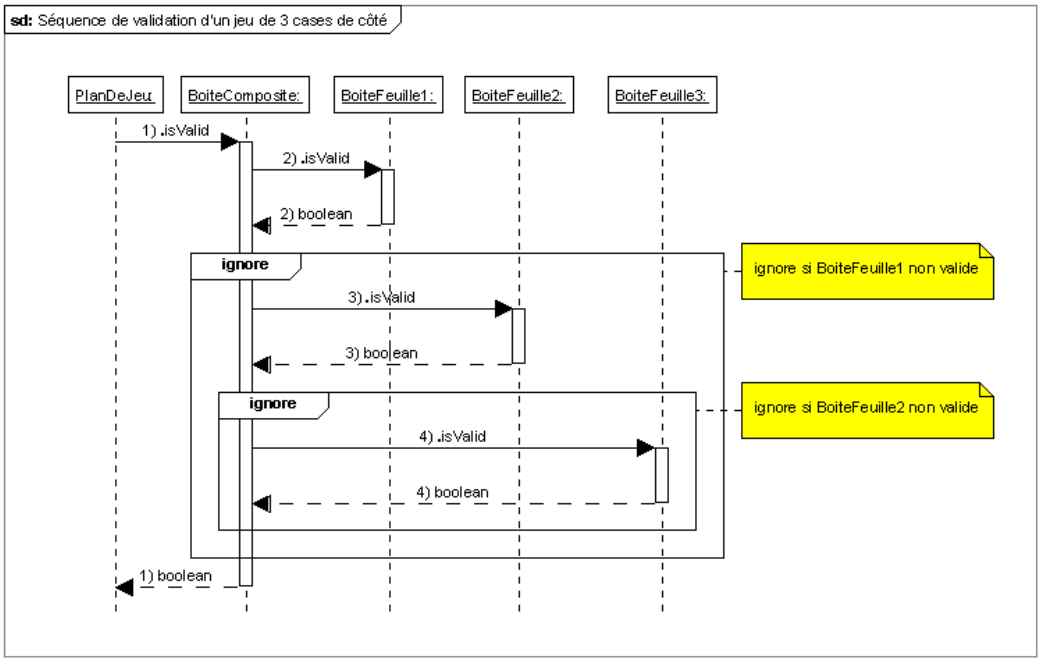

FIG. 9 – Séquence de validation d'un jeu de 3 cases de côté