MAROC ynov CAMPUS

RAPPORT DE PROJET

Sudoku Console en C#

Bangoura Ahmed Rachid
Aissatou Lamarana Diallo

08 Décembre 2025

Table des matières

Remerciement

Nous tenons à exprimer notre sincère gratitude à Monsieur LOUKACH Mehdi, enseignant du module C#, pour la qualité de son enseignement et l'accompagnement qu'il nous a offert tout au long de ce projet.
Grâce à ses explications claires, sa disponibilité et ses conseils pertinents, nous avons pu acquérir des bases solides en programmation et mener à bien notre travail. Son engagement pédagogique a grandement contribué à notre progression et à notre compréhension des concepts abordés.
Nous le remercions chaleureusement pour son soutien et son professionnalisme.

Résume exécutif

Ce projet consiste en la conception et le développement d'un jeu de Sudoku en C# sous forme d'application console. L'objectif est de renforcer les compétences en programmation orientée objet, en manipulation de matrices, et en validation algorithmique. Le jeu permet à l'utilisateur de jouer sur une grille 9x9, de saisir des valeurs, de vérifier la validité des coups, et de contrôler l'avancement de la partie. Des fonctionnalités comme la réinitialisation, la vérification de la solution et la sortie du jeu sont intégrées.
Le projet est structuré en plusieurs classes (Program, SudokuGame, SudokuBoard, SudokuValidator) pour assurer une organisation claire et modulaire.
Ce travail démontre la capacité à transformer des règles logiques complexes en une application interactive, tout en respectant les contraintes d'une interface console.

Introduction

Contexte

Le Sudoku, jeu de logique mondialement reconnu, constitue un excellent support pédagogique pour développer des compétences en raisonnement, structuration et résolution de problèmes. Dans le cadre de l'apprentissage de la programmation orientée objet en C#, ce projet propose de transposer les règles du Sudoku dans une application console interactive. L'environnement console, bien que limité graphiquement, offre un terrain idéal pour mettre en pratique la manipulation de matrices, la validation algorithmique et l'interaction utilisateur.

Objectif

L'objectif est de concevoir une application robuste et modulaire permettant :

- La représentation d'une grille de Sudoku 9x9.
- L'initialisation d'une grille partiellement remplie.
- La saisie et la validation des coups par l'utilisateur.
- La gestion des erreurs et la fourniture de messages explicites.
- La vérification de l'état de complétion de la grille.
- La possibilité de redémarrer ou quitter la partie.

Ce projet vise ainsi à renforcer la maîtrise des concepts fondamentaux de la programmation en C#, tout en offrant une expérience utilisateur claire et intuitive.

Sources de données

Le projet repose sur trois éléments principaux :

- Grille Sudoku : tableau 9x9 représentant l'état du jeu.
- Entrées utilisateur : coordonnées (ligne, colonne) et valeur saisie.
- Règles de validation : contraintes logiques du Sudoku (unicité des chiffres dans lignes, colonnes et sous-grilles).

Explication du sudoku

Contexte

Le Sudoku est un jeu de réflexion apparu au Japon dans les années 1980 et devenu mondialement populaire. Il repose sur une logique simple mais exigeante : remplir une grille de 9x9 cases avec des chiffres de 1 à 9, en respectant des contraintes strictes. Ce puzzle est aujourd'hui utilisé non seulement comme divertissement, mais aussi comme outil pédagogique pour développer la concentration, la mémoire et les capacités de raisonnement.

Règles

Les règles fondamentales du Sudoku sont les suivantes :

- Chaque **ligne** doit contenir les chiffres de 1 à 9 sans répétition.
- Chaque **colonne** doit contenir les chiffres de 1 à 9 sans répétition.
- Chaque **sous-grille 3x3** doit contenir les chiffres de 1 à 9 sans répétition.

Le joueur commence avec une grille partiellement remplie et doit compléter les cases vides en respectant ces contraintes.

But

L'objectif est de remplir l'ensemble de la grille de manière correcte et complète. Contrairement aux jeux mathématiques, le Sudoku ne nécessite aucun calcul : il s'agit uniquement de logique et de déduction. La difficulté varie selon le nombre de cases initialement remplies et leur disposition.
Intérêt pédagogique
Le Sudoku est un excellent support pour :

- **Développer la logique** et la capacité de raisonnement.
- **Renforcer la concentration** et l'attention aux détails.
- **Apprendre la patience et la persévérance** face à des problèmes complexes.
- **Illustrer des concepts informatiques** tels que la validation, la gestion de contraintes et la manipulation de matrices.

Fonctionnalités implémentées

Le projet Sudoku Console en C# intègre les fonctionnalités suivantes :

Fonctionnalités de jeu

- **Grille 9x9** affichée avec séparateurs pour distinguer lignes, colonnes et sous-grilles.
- **Initialisation aléatoire** avec 1 ou 2 chiffres valides par bloc 3x3.
- **Saisie utilisateur** au format *row col val* (ligne, colonne, valeur).
- **Validation des coups** :
    - Valeur comprise entre 1 et 9.
    - Respect des règles du Sudoku (unicité dans la ligne, la colonne et la sous-grille).
- **Feedback immédiat :**
    - Message de confirmation pour un coup valide.
    - Message d'erreur explicite pour un coup invalide.

Interface console

- Affichage lisible de la grille avec symboles et séparateurs ASCII.
- Instructions claires présentées au lancement du jeu.
- Commandes spéciales disponibles :
    - *check* → vérifie si la grille est complète.
    - *restart* → réinitialise les cases non fixes.
    - *quit* → quitte le jeu.

Structure du code

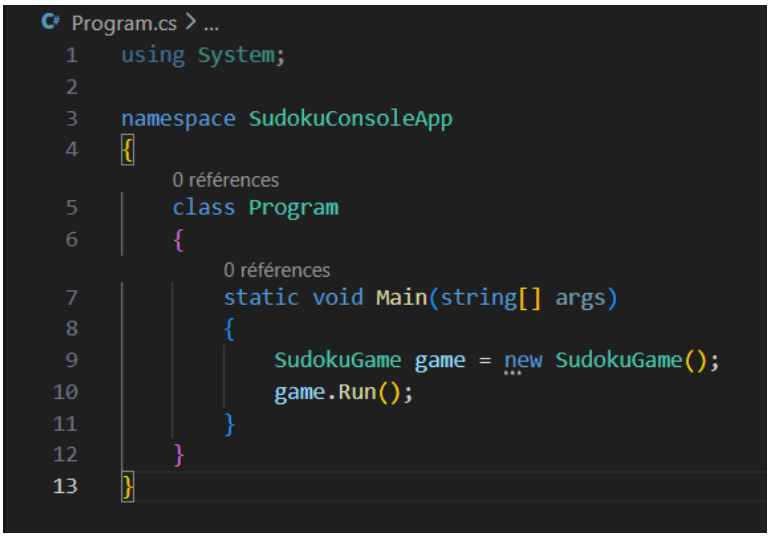
- Organisation en classes distinctes pour respecter la programmation orientée objet :
    - *Program* → point d'entrée du projet.
    - *SudokuGame* → boucle principale et gestion des commandes.
    - *SudokuBoard* → gestion de la grille, affichage, saisie et réinitialisation.
    - *SudokuValidator* → validation des coups selon les règles du Sudoku.

Gestion des erreurs

- **Entrées hors limites** → message *"Erreur : hors limites."*
- **Tentative de modification d'une case fixe** → message *"Impossible : case de départ."*
- **Valeur invalide (<1 ou >9)** → message *"Erreur : chiffre entre 1 et 9."*
- **Coup non valide (règles non respectées)** → message *"Mouvement invalide (règlr du sudoku non respectées)."*

Structure du code

Implémentation – Program.cs

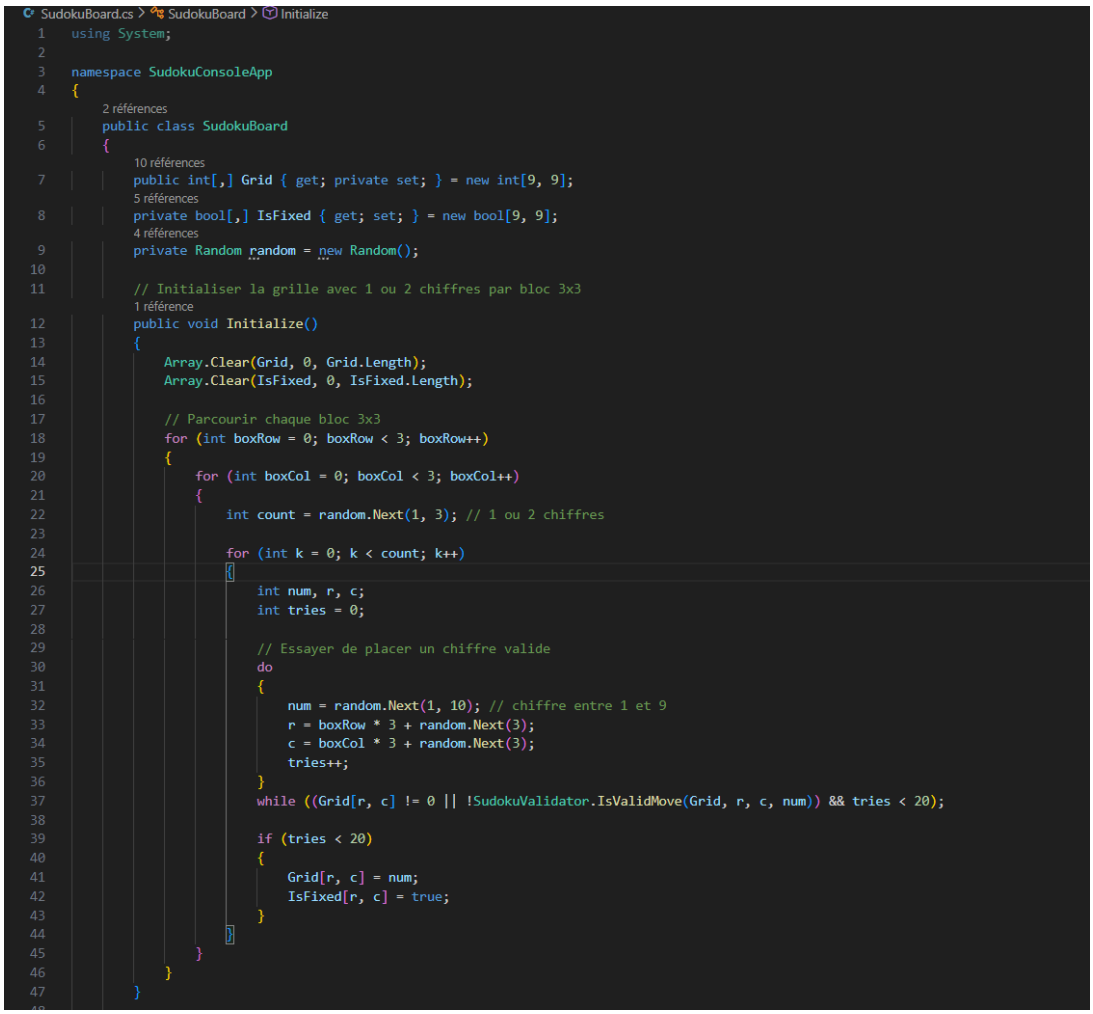


Objectif

Cette classe représente le **point d'entrée** du projet.

- La méthode *Main(string[] args)* est exécutée automatiquement au lancement de l'application.
- Elle instancie un objet *SudokuGame* et appelle sa méthode *Run(),* ce qui démarre la boucle principale du jeu.
- En résumé, *Program.cs* sert uniquement à **initialiser et lancer le jeu**.

Implémentation – SudokoBoard.Initialize()



Objectif

- Cette méthode sert à **initialiser la grille du Sudoku** avant le début de la partie.
- Elle efface toute valeur précédente dans *Grid* (tableau 9x9 des chiffres) et *IsFixed* (tableau 9x9 indiquant les cases fixes).
- Elle parcourt chaque **bloc 3x3** de la grille et y place **1 ou 2 chiffres aléatoires**.
- Pour chaque chiffre :
  - Un nombre aléatoire entre 1 et 9 est généré.
  - Une position aléatoire dans le bloc est choisie.
  - La validité du placement est vérifiée grâce à *SudokuValidator.IsValidMove().*
  - Si le placement est valide, le chiffre est ajouté à la grille et marqué comme **fixe** (IsFixed[r, c] = true).
- Le compteur *tries* limite le nombre de tentatives pour éviter les boucles infinies

Implémentation – SudokoBoard.Display()

```
        // Afficher la grille
        1 référence
        public void Display()
        {
            Console.WriteLine(" +-------+-------+-------+");
            for (int r = 0; r < 9; r++)
            {
                Console.Write(" | ");
                for (int c = 0; c < 9; c++)
                {
                    int val = Grid[r, c];
                    char ch = val == 0 ? '.' : (char)('0' + val);
                    Console.Write(ch + " ");
                    if (c == 2 || c == 5) Console.Write("| ");
                }
                Console.WriteLine("|");
                if (r == 2 || r == 5 || r == 8)
                    Console.WriteLine(" +-------+-------+-------+");
            }
        }
```

Objectif

Affiche la grille du Sudoku en console.

- Utilise des séparateurs ASCII (+-------+, |) pour distinguer les lignes, colonnes et sous-grilles 3x3.
- Les cases vides sont représentées par un point (.), tandis que les cases remplies affichent leur chiffre.
- Permet une **lecture claire et intuitive** de l'état du jeu.

Implémentation – SudokoBoard.Tryplace(int row, int col, int value)

```
        public string TryPlace(int row, int col, int value)
        {
            int r = row - 1;
            int c = col - 1;

            if (r < 0 || r >= 9 || c < 0 || c >= 9) return "Erreur: hors limites.";
            if (IsFixed[r, c]) return "Impossible: case de départ.";
            if (value < 1 || value > 9) return "Erreur: chiffre entre 1 et 9.";

            if (!SudokuValidator.IsValidMove(Grid, r, c, value))
                return "Mouvement invalide (règles Sudoku non respectées).";

            Grid[r, c] = value;
            return "Coup accepté.";
        }
```

Objectif

Permet à l'utilisateur de **placer un chiffre** dans une case donnée.

- Vérifie plusieurs conditions :
  - Les coordonnées sont dans les limites de la grille (1–9).
  - La case n'est pas fixe (pré-remplie au départ).

- La valeur est comprise entre 1 et 9.
- Le coup respecte les règles du Sudoku (validation via SudokuValidator.IsValidMove).

- Retourne un **message explicite** : erreur ou confirmation du coup accepté

Implémentation – SudokoBoard.Iscomplet()

```
86          public bool IsComplete()
87          {
88              for (int r = 0; r < 9; r++)
89                  for (int c = 0; c < 9; c++)
90                      if (Grid[r, c] == 0) return false;
91              return true;
92          }
93
```

Objectif

Vérifie si la grille est entièrement remplie.

- Parcourt toutes les cases du tableau Grid.
- Si une case est vide (== 0), retourne false.
- Si toutes les cases sont remplies, retourne true.
- Sert à déterminer si le joueur a terminé la partie.

Implémentation – SudokoBoard.ResetNonFixed()

```
94          public void ResetNonFixed()
95          {
96              for (int r = 0; r < 9; r++)
97                  for (int c = 0; c < 9; c++)
98                      if (!IsFixed[r, c]) Grid[r, c] = 0;
99          }
100     }
101 }
```

Objectif

Réinitialise uniquement les **cases non fixes** de la grille.

- Parcourt toutes les cases et remet à zéro celles qui ne sont pas marquées comme fixes (IsFixed[r, c] == false).
- Permet au joueur de recommencer une partie sans perdre les chiffres initiaux placés lors de l'initialisation.

Implémentation – SudokoValidator()

SudokuValidator.cs > ...

```
namespace SudokuConsoleApp
{
    2 références
    public static class SudokuValidator
    {
        2 références
        public static bool IsValidMove(int[,] grid, int r, int c, int value)
        {
            // Vérifie la ligne
            for (int x = 0; x < 9; x++)
                if (grid[r, x] == value) return false;

            // Vérifie la colonne
            for (int y = 0; y < 9; y++)
                if (grid[y, c] == value) return false;

            // Vérifie la sous-grille 3x3
            int boxRowStart = (r / 3) * 3;
            int boxColStart = (c / 3) * 3;
            for (int rr = boxRowStart; rr < boxRowStart + 3; rr++)
                for (int cc = boxColStart; cc < boxColStart + 3; cc++)
                    if (grid[rr, cc] == value) return false;

            return true;
        }
    }
}
```

Objectif

Cette méthode statique est utilisée pour valider un coup dans la grille du Sudoku.

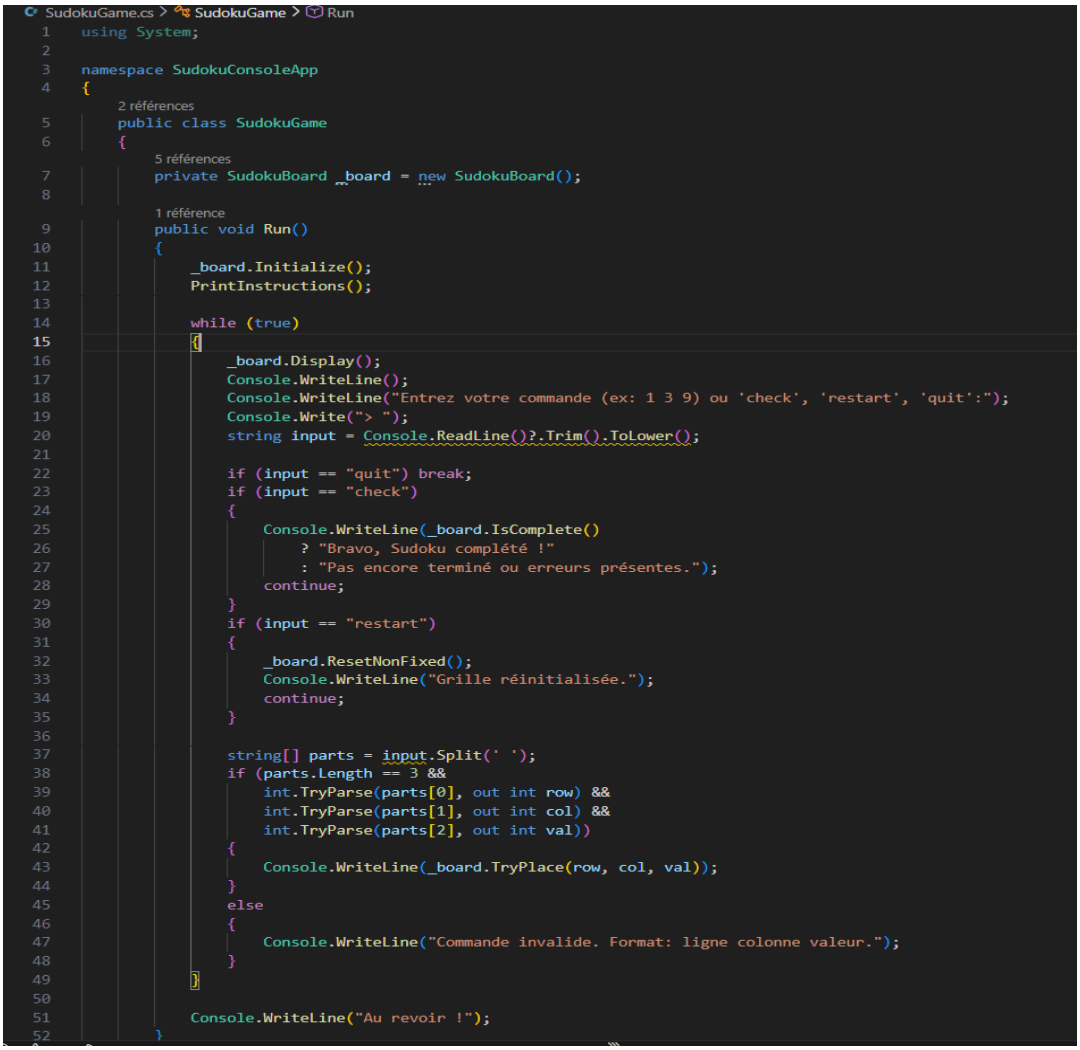
- Elle prend en paramètres :
- *grid* → la grille actuelle (tableau 9x9).
- *r, c* → indices de la ligne et de la colonne où l'on veut placer un chiffre.
- *Value* → le chiffre proposé par l'utilisateur.

Étapes de validation

1. Vérification de la ligne
    - Parcourt toutes les colonnes de la ligne *r*.
    - Si le chiffre *value* existe déjà, retourne *false*.
2. Vérification de la colonne
    - Parcourt toutes les lignes de la colonne *c*.
    - Si le chiffre *value* existe déjà, retourne *false*.
3. Vérification de la sous-grille 3x3
    - Calcule les coordonnées de départ de la sous-grille (*boxRowStart, boxColStart)*.
    - Parcourt les 9 cases de cette sous-grille.
    - Si le chiffre *value* existe déjà, retourne *false*.
4. Validation finale

- Si aucune duplication n'est trouvée, retourne *true.*

Implémentation – SudokoGame.Run()



```
SudokuGame.cs > SudokuGame > Run
using System;

namespace SudokuConsoleApp
{
    2 références
    public class SudokuGame
    {
        5 références
        private SudokuBoard _board = new SudokuBoard();

        1 référence
        public void Run()
        {
            _board.Initialize();
            PrintInstructions();

            while (true)
            {
                _board.Display();
                Console.WriteLine();
                Console.WriteLine("Entrez votre commande (ex: 1 3 9) ou 'check', 'restart', 'quit':");
                Console.Write("> ");
                string input = Console.ReadLine()?.Trim().ToLower();

                if (input == "quit") break;
                if (input == "check")
                {
                    Console.WriteLine(_board.IsComplete()
                        ? "Bravo, Sudoku complété !"
                        : "Pas encore terminé ou erreurs présentes.");
                    continue;
                }
                if (input == "restart")
                {
                    _board.ResetNonFixed();
                    Console.WriteLine("Grille réinitialisée.");
                    continue;
                }

                string[] parts = input.Split(' ');
                if (parts.Length == 3 &&
                    int.TryParse(parts[0], out int row) &&
                    int.TryParse(parts[1], out int col) &&
                    int.TryParse(parts[2], out int val))
                {
                    Console.WriteLine(_board.TryPlace(row, col, val));
                }
                else
                {
                    Console.WriteLine("Commande invalide. Format: ligne colonne valeur.");
                }
            }

            Console.WriteLine("Au revoir !");
        }
```

Objectif

C'est la méthode principale qui gère la boucle du jeu.

Étapes clés :

1. Initialisation : appelle *_board.Initialize()* pour préparer la grille.
2. Affichage des instructions : via *PrintInstructions().*
3. Boucle de jeu :
    - Affiche la grille *(_board.Display()).*
    - Attend une commande ou un coup de l'utilisateur.
    - Interprète les commandes spéciales :
        - *quit* → quitte le jeu.
        - *check* → vérifie si la grille est complète.
        - *restart* → réinitialise les cases non fixes.
    - Si la commande est au format *row col val*, tente de placer le chiffre via *_board.TryPlace().*
    - Sinon, affiche un message d'erreur sur le format.
4. Fin du jeu : affiche "Au revoir !"

Implémentation – SudokoGame.PrintInstruction()

```
1 reference
private void PrintInstructions()
{
    Console.Clear();
    Console.WriteLine("========================================");
    Console.WriteLine("        Bienvenue dans Sudoku Console      ");
    Console.WriteLine("========================================\n");

    Console.WriteLine("                RÈGLES");
    Console.WriteLine("----------------------------------------");
    Console.WriteLine("• Chaque ligne, colonne et sous-grille doit");
    Console.WriteLine("  contenir les chiffres de 1 à 9.");
    Console.WriteLine("• Les cases fixes ne peuvent pas être modifiées.\n");

    Console.WriteLine("              COMMANDES");
    Console.WriteLine("----------------------------------------");
    Console.WriteLine(" 'row col val' : jouer un coup (ex: 4 5 3)");
    Console.WriteLine(" 'check'       : vérifier si la grille est complète");
    Console.WriteLine(" 'restart'     : recommencer une nouvelle partie");
    Console.WriteLine(" 'quit'        : quitter le jeu\n");

    Console.WriteLine("========================================");
}
    }
}
```

Objectif

Affiche les **instructions du jeu** au démarrage.

- Présente les **règles du Sudoku** :
  - Chaque ligne, colonne et sous-grille doit contenir les chiffres de 1 à 9.
  - Les cases fixes ne peuvent pas être modifiées.
- Explique les **commandes disponibles** :
  - *row col val* → jouer un coup (exemple : 4 5 3).
  - *check* → vérifier si la grille est complète.
  - *restart* → recommencer une nouvelle partie.
  - *quit* → quitter le jeu.
- Utilise un format ASCII clair pour rendre l'affichage lisible et structuré

Gestion des erreurs

La robustesse du projet repose sur une gestion efficace des erreurs et des entrées invalides. L'application est conçue pour éviter tout crash et fournir des messages explicites afin de guider l'utilisateur.

Types d'erreurs gérées

• **Entrées hors limites**
- Exemple : saisie d'une ligne ou colonne en dehors de 1–9.
- Message affiché : *"Erreur : hors limites. "*

• **Modification d'une case fixe**
- Les cases initialement remplies lors de l'initialisation ne peuvent pas être modifiées.
- Message affiché : *"Impossible : case de départ"*

• **Valeurs invalides**
- Exemple : saisie d'un chiffre inférieur à 1 ou supérieur à 9.
- Message affiché : Erreur : *"chiffre entre 1 et 9"*

• **Coup non valide (règles Sudoku non respectées)**
- Exemple : tentative de placer un chiffre déjà présent dans la même ligne, colonne ou sous-grille 3x3.
- Message affiché : *"Mouvement invalide (règle de sudoku non respectées). "*

• **Commande incorrecte**
- Exemple : saisie d'une commande qui ne correspond pas au format attendu.
- Message affiché : *"Commande invalide. Format : ligne colonne valeur. "*

Stratégie de gestion

• **Validation systématique** des entrées avant toute modification de la grille.
• **Messages clairs et pédagogiques** pour informer l'utilisateur de la nature de l'erreur.
• **Boucle de jeu continue** : l'application ne s'arrête jamais sur une erreur, elle invite simplement l'utilisateur à corriger sa saisie.
• **Réinitialisation contrôlée** : la commande permet de remettre à zéro uniquement les cases non fixes, évitant toute perte de données initiales

Approche et logique

Le développement du projet Sudoku Console en C# s'est appuyé sur une démarche progressive et modulaire, respectant les principes de la programmation orientée objet et de la résolution de problèmes par étapes.

Approche méthodologique

1. **Décomposition du problème**
   - Le jeu a été divisé en sous-tâches : initialisation de la grille, affichage, validation des coups, gestion des commandes, vérification de la complétion.
   - Chaque sous-tâche a été implémentée dans une classe ou une méthode dédiée pour assurer la modularité.
2. **Conception orientée objet**
   - Utilisation de plusieurs classes (SudokuGame, SudokuBoard, SudokuValidator) pour séparer la logique du jeu, la gestion de la grille et la validation des règles.
   - Respect du principe de responsabilité unique afin de faciliter la maintenance et l'évolution du code.
3. **Validation incrémentale**
   - Chaque fonctionnalité (affichage, saisie, validation) a été testée individuellement avant d'être intégrée dans la boucle principale du jeu.
   - Les erreurs sont interceptées et traitées immédiatement pour éviter les plantages.
4. **Interaction utilisateur**
   - Conception d'une interface console simple et intuitive.
   - Les commandes (check, restart, quit) permettent une interaction fluide et un contrôle direct du jeu.

Logique de fonctionnement

- **Initialisation :**
  - La grille est générée avec quelques chiffres valides placés aléatoirement dans chaque sous-grille 3x3.
  - Les cases initiales sont marquées comme fixes et ne peuvent pas être modifiées.
- **Affichage :**
  - La grille est représentée en console avec des séparateurs ASCII pour distinguer les lignes, colonnes et sous-grilles.
  - Les cases vides sont affichées par un point (.).
- **Saisie et validation :**
  - L'utilisateur saisit une commande au format row col val.
  - La méthode SudokuValidator.IsValidMove() vérifie si le chiffre respecte les règles du Sudoku.
  - Si valide → le chiffre est placé dans la grille.
  - Si invalide → un message explicite est affiché.

- **Boucle de jeu :**
  - Le programme reste actif tant que l'utilisateur n'a pas saisi quit.
  - À chaque tour, la grille est affichée et l'utilisateur peut jouer un coup ou exécuter une commande spéciale.
- **Contrôle de complétion :**
  - La méthode SudokuBoard.IsComplete() vérifie si toutes les cases sont remplies.
  - Si la grille est complète et valide → message de félicitations.
  - Sinon → indication que la partie n'est pas encore terminée.

Défis rencontrés

Le développement du projet Sudoku Console en C# a présenté plusieurs défis techniques et méthodologiques. Ces obstacles ont été surmontés grâce à une approche progressive, des tests fréquents et une organisation modulaire du code.

Génération de la grille initiale

- Problème : placer aléatoirement des chiffres valides dans chaque sous-grille 3x3 sans violer les règles du Sudoku.
- Solution : utilisation d'une boucle avec un nombre limité d'essais (tries < 20) pour garantir la validité des placements et éviter les blocages.

Lisibilité de l'affichage console

- Problème : représenter une grille 9x9 de manière claire dans un environnement limité.
- Solution : ajout de séparateurs horizontaux et verticaux (+-------+ et |) pour distinguer les lignes, colonnes et sous-grilles.

Interaction utilisateur

- Problème : concevoir une saisie intuitive et éviter les erreurs de format.
- Solution : adoption d'un format simple row col val et ajout de commandes explicites (check, restart, quit).

Validation et gestion des erreurs

- Problème : éviter les crashs en cas d'entrées invalides ou de commandes incorrectes.
- Solution : mise en place de messages explicites pour chaque type d'erreur (hors limites, case fixe, valeur invalide, coup non valide).

Vérification de la complétion

- Problème : déterminer si la grille est correctement remplie sans implémenter un solveur complet.
- Solution : la méthode IsComplete() vérifie simplement si toutes les cases sont remplies, ce qui permet une validation basique mais efficace.

Organisation du code

- Problème : maintenir une structure claire et éviter que la logique du jeu ne soit dispersée.
- Solution : séparation en classes (SudokuGame, SudokuBoard, SudokuValidator) pour respecter la programmation orientée objet et faciliter la maintenance.

Conclusion

Le projet Sudoku Console en C# a permis de mettre en pratique de manière concrète les concepts fondamentaux de la programmation orientée objet, tout en développant une application interactive et robuste dans un environnement limité comme la console.
Grâce à une approche progressive et modulaire, le jeu intègre les principales fonctionnalités attendues : affichage clair de la grille, validation des coups selon les règles du Sudoku, gestion des erreurs, et interaction fluide avec l'utilisateur via des commandes simples. La séparation en classes (Program, SudokuGame, SudokuBoard, SudokuValidator) illustre une bonne organisation du code et facilite la maintenance ainsi que l'évolution future du projet.
Ce travail a également mis en évidence l'importance de la validation des entrées, de la gestion des erreurs, et de la clarté de l'interface utilisateur pour garantir une expérience fluide et pédagogique.

Perspectives d'amélioration

Le projet constitue une base solide pour des évolutions futures, telles que :

- L'intégration d'un solveur automatique pour vérifier la validité complète d'une solution.
- L'ajout d'un chronomètre pour mesurer le temps de résolution.
- La création de niveaux de difficulté (facile, moyen, difficile).
- Le développement d'une interface graphique pour enrichir l'expérience utilisateur.

En conclusion, ce projet démontre non seulement la maîtrise des concepts techniques en C#, mais aussi la capacité à transformer un jeu logique en une application interactive, claire et accessible. Il représente une étape importante dans l'apprentissage de la programmation et ouvre la voie à des projets plus complexes et ambitieux.

Annexes

A. Exemple de commandes utilisateur

- 4 5 3 → place le chiffre 3 à la ligne 4, colonne 5.
- check → vérifie si la grille est complète et affiche un message de validation.
- restart → réinitialise toutes les cases non fixes pour recommencer la partie.
- quit → quitte le jeu et termine l'application.

B. Structure des fichiers du projet

- Program.cs → Point d'entrée du projet, lance SudokuGame.
- SudokuGame.cs → Boucle principale, gestion des commandes et instructions.
- SudokuBoard.cs → Gestion de la grille, affichage, placement des coups, réinitialisation.
- SudokuValidator.cs → Validation des règles Sudoku (lignes, colonnes, sous-grilles).

C. Variables, tableaux et objets utilisés

Dans SudokuBoard

- int[,] Grid → tableau 9x9 représentant la grille du Sudoku.
- bool[,] IsFixed → tableau 9x9 indiquant si une case est fixe (initialement remplie).
- Random random → générateur de nombres aléatoires pour l'initialisation.
- Variables locales :
  - num → chiffre généré aléatoirement (1–9).
  - r, c → indices de ligne et colonne.
  - tries → compteur d'essais pour placer un chiffre valide.
  - boxRow, boxCol → indices des blocs 3x3.
  - boxRowStart, boxColStart → coordonnées de départ d'une sous-grille 3x3.

Dans SudokuValidator

- Paramètres de méthode IsValidMove(int[,] grid, int r, int c, int value) :
  - grid → la grille actuelle.
  - r, c → indices de ligne et colonne.
  - value → chiffre à tester.
- Variables locales :
  - x, y → indices pour parcourir lignes et colonnes.
  - rr, cc → indices pour parcourir une sous-grille 3x3.

Dans SudokuGame

- _board → instance de la classe SudokuBoard.
- input → commande saisie par l'utilisateur.
- parts → tableau de chaînes contenant les éléments de la commande (row, col, val).
- Variables locales :
  - row, col, val → valeurs extraites de la saisie utilisateur.

D. Exemple de messages d'erreur

- "Erreur: hors limites." → saisie en dehors de la grille.
- "Impossible: case de départ." → tentative de modification d'une case fixe.

- "Erreur: chiffre entre 1 et 9." → valeur invalide.
- "Mouvement invalide (règles Sudoku non respectées)." → coup non conforme aux règles.
- "Commande invalide. Format: ligne colonne valeur." → saisie incorrecte.

E. Perspectives d’amélioration

- Ajout d’un solveur automatique pour vérifier la validité complète d’une solution.
- Intégration d’un chronomètre pour mesurer le temps de résolution.
- Création de niveaux de difficulté (facile, moyen, difficile).
- Développement d’une interface graphique pour enrichir l’expérience utilisateur.