



Practical Programming Assignment

Programme Name: Sudoku Solver

Module Name and Code: Object Oriented Programming , AG200

Student ID Number: st120-10722

Lecturer's Name: Chasapis, Athanasios

Word Count: 2365

Εισαγωγή

Ο παρών κώδικας αποτελεί μια υλοποίηση αλγορίθμου επίλυσης του παιχνιδιού Sudoku, ακολουθώντας τις αρχές της **αρχιτεκτονικής επιπέδου επιχείρησης (Enterprise-level Architecture)** για το μάθημα Object Oriented Programming (AG200) στο Aegean College τμήμα Λαρίσας. Η σχεδίαση εναρμονίζεται με τις πλέον σύγχρονες πρακτικές ανάπτυξης λογισμικού, όπως το **Domain-Driven Design (DDD)**, οι **αρχές SOLID**, καθώς και η **Καθαρή Αρχιτεκτονική (Clean Architecture)**, διασφαλίζοντας την επεκτασιμότητα, τη συντηρησιμότητα και την ευχρηστία του αντικειμενοστραφή προγραμματισμένου συστήματος.

Η ανάλυση της παρούσας εφαρμογής διαρθρώνεται σε τρεις βασικές ενότητες:

1. **Αρχιτεκτονική και Δομή**
2. **Επίπεδο Τομέα (Domain Layer)**
3. **Υλοποίηση Σχεδιαστικών Προτύπων και Στρατηγικών Επίλυσης**
4. **Code**
5. **Επίλυση Αλγορίθμου**
6. **UML Διαγράμμα**
7. **Αποτελέσματα**
8. **Συμπερασμα**

1. Αρχιτεκτονική και Δομή

Ο σχεδιασμός του κώδικα βασίζεται σε μια πολυεπίπεδη αρχιτεκτονική, διασφαλίζοντας την αποδοτικότητα, την ευελιξία και τη συντηρησιμότητα της εφαρμογής. Η δομή ακολουθεί το μοντέλο **Domain-Driven Design (DDD)**, χωρίζοντας την εφαρμογή σε **Domain Layer, Infrastructure**

Layer και **Presentation Layer**. Αυτή η προσέγγιση διασφαλίζει την ανεξαρτησία της επιχειρησιακής λογικής από τις τεχνολογικές λεπτομέρειες, την παρουσίαση καθώς και την ευελιξία προσθήκης περισσότερων αλγορίθμων ή δομών δεδομένων για την επίλυση του προβλήματος . (Sudoku)

1.1 Επίπεδο Τομέα (Domain Layer)

Το **Domain Layer** περιλαμβάνει τις βασικές οντότητες του Sudoku, καθώς και τις διεπαφές που καθορίζουν την επιχειρησιακή λογική. Οι κύριες κλάσεις είναι:

- **Grid**: Αναπαριστά τον πίνακα Sudoku ως δισδιάστατο πίνακα ακεραίων αριθμών, παρέχοντας μεθόδους για ανάκτηση και ενημέρωση κελιών.
- **Position**: Ορίζει τις συντεταγμένες των κελιών στον πίνακα, διασφαλίζοντας την ακεραιότητα των θέσεων.
- **ISudokuValidator**: Διεπαφή που επιτρέπει την επικύρωση κινήσεων σύμφωνα με τους κανόνες του παιχνιδιού Sudoku.
- **ISudokuSolver**: Ορίζει τη βασική λειτουργικότητα των αλγορίθμων επίλυσης, επιτρέποντας την εναλλαγή στρατηγικών.

1.2 Υποδομή (Infrastructure Layer)

Το **Infrastructure Layer** υλοποιεί τις διεπαφές του επιπέδου τομέα και διαχειρίζεται την πρόσβαση στα δεδομένα του επεξεργαζόμενου πίνακα (Sudoku). Ενδεικτικά οι κλάσεις:

- **BacktrackingSolver** και **DFSArrayListSolver**: Διαφορετικές στρατηγικές επίλυσης του Sudoku.
- **FileGridLoader**: Υλοποιεί τη φόρτωση πινάκων Sudoku από αρχεία (κατά στιγμή μόνο *.txt), επιτρέποντας την ευέλικτη εισαγωγή δεδομένων στο πρόγραμμα.

1.3 Παρουσίαση (Presentation Layer)

Το **Presentation Layer** ασχολείται με την αλληλεπίδραση χρήστη μέσω του **ConsoleUI**, παρέχοντας:

- Δυνατότητα επιλογής στρατηγικής επίλυσης.
- Προβολή της τρέχουσας κατάστασης του Sudoku.
- Μηχανισμούς επικοινωνίας με το **Domain Layer** χωρίς εξάρτηση από την επιχειρησιακή λογική.
- Σε δεύτερο πλano, αυτό το Layer μπορεί να χρησιμοποιηθεί για να δημιουργηθεί και κανονική UI σε γραφικό περιβάλλον για την πιο ευκόλη χρήση.

1.4 Αρχές SOLID

Ο σχεδιασμός της εφαρμογής υιοθετεί τις αρχές **SOLID** για τη βελτίωση της ποιότητας του κώδικα:

- **Single Responsibility Principle (SRP)**: Κάθε κλάση έχει μία μόνο ευθύνη.
- **Open/Closed Principle (OCP)**: Νέοι αλγόριθμοι μπορούν να προστεθούν χωρίς αλλαγές στον υπάρχοντα κώδικα, καθώς κάθε λύση είναι μια Κλάση/Αντικείμενο μονη της και ανεξαρτητη απο τις αλλες λυσεις.
- **Liskov Substitution Principle (LSP)**: Οι υλοποιήσεις μπορούν να αντικαθιστούν τις διεπαφές τους χωρίς παρενέργειες.
- **Interface Segregation Principle (ISP)**: Χρήση εξειδικευμένων διεπαφών για αποφυγή περιττών εξαρτήσεων.
- **Dependency Inversion Principle (DIP)**: Η διαχείριση εξαρτήσεων μέσω διεπαφών καθιστά το σύστημα πιο ευέλικτο.

1.5 Καθαρή Αρχιτεκτονική (Clean Architecture)

Η αρχιτεκτονική της εφαρμογής διαχωρίζει την λογική λύσης από τις λεπτομέρειες της υλοποίησης, εφαρμόζοντας την **Καθαρή Αρχιτεκτονική**. Τα δεδομένα και οι διεπαφές καθορίζονται στο **Domain Layer**, ενώ οι εξωτερικές λεπτομέρειες, όπως η αποθήκευση δεδομένων και η παρουσίαση, βρίσκονται στα αντίστοιχα επίπεδα, εξασφαλίζοντας την ανεξαρτησία των κρίσιμων τμημάτων του συστήματος. Στην παρούσα ασκήση δεν υπάρχει μεγάλη πολυπλοκότητα, χωρίς όμως αυτό να σημαίνει ότι ένα τέτοιο πλάνο δεν μπορεί να εφαρμοστεί και σε μικρές εφαρμογές που πολύ γρήγορα μπορούν να γιγαντωθούν λόγω *scope creep*.

Ο σχεδιασμός του κώδικα υλοποιείται σύμφωνα με δοκιμασμένες αρχές και σχεδιαστικά πρότυπα που ενδείκνυνται για εφαρμογές υψηλής κλίμακας. Το κυρίαρχο αρχιτεκτονικό πρότυπο είναι το **Domain-Driven Design (DDD)**, το οποίο οργανώνει τη δομή της εφαρμογής σε τρία διακριτά επίπεδα: **Domain**, **Infrastructure**, και **Presentation**.

2. Επίπεδο Τομέα (Domain Layer)

Το **Επίπεδο Τομέα (Domain Layer)** αποτελεί την καρδιά και λογική της εφαρμογής, καθορίζοντας τη βασική λογική λύσης και τα θεμελιώδη συστατικά του Sudoku. Περιλαμβάνει τις κύριες οντότητες, αντικείμενα αξίας και διεπαφές που ορίζουν τους κανόνες και τη λειτουργικότητα του συστήματος.

2.1 Οντότητες και Αντικείμενα Αξίας

Η κύρια οντότητα είναι η **Grid**, η οποία αντιπροσωπεύει τον πίνακα του παιχνιδιού Sudoku. Είναι υπεύθυνη για την αποθήκευση της κατάστασης του παιχνιδιού μέσω ενός δισδιάστατου πίνακα ακεραίων, όπου κάθε στοιχείο αντιπροσωπεύει μια αριθμητική τιμή ή ένα κενό κελί. Παρέχει επίσης βοηθητικές μεθόδους για την ανάκτηση και ενημέρωση των κελιών, καθώς και για την επικύρωση της συνολικής δομής του πλέγματος.

Ένα άλλο βασικό αντικείμενο αξίας είναι η **Position**, το οποίο αναπαριστά μια σταθερή τοποθεσία στον πίνακα Sudoku. Η **Position** αποτελείται από δύο συντεταγμένες, τη γραμμή (row) και τη στήλη (column), και διασφαλίζει ότι οι θέσεις στο πλέγμα διαχειρίζονται με ασφάλεια και ακεραιότητα.

2.2 Διεπαφές και Λογική Λύσης

Το **ISudokuValidator** είναι μια διεπαφή που ορίζει τη λειτουργικότητα επικύρωσης κινήσεων στο Sudoku. Ελέγχει αν μια τιμή μπορεί να τοποθετηθεί σε ένα συγκεκριμένο κελί χωρίς να παραβιάζει τους κανόνες του παιχνιδιού. Η υλοποίησή του επιτρέπει διαφορετικές στρατηγικές επικύρωσης, όπως ελέγχους σε γραμμές, στήλες και περιοχές του πίνακα.

Η βασική διεπαφή για την επίλυση του Sudoku είναι το **ISudokuSolver**, το οποίο ορίζει τη μέθοδο **Solve()**. Αυτή η μέθοδος εφαρμόζεται από διαφορετικούς αλγόριθμους, επιτρέποντας την εναλλαγή στρατηγικών επίλυσης χωρίς να επηρεάζεται ο βασικός κώδικας της εφαρμογής. Έτσι, το **Domain Layer** παραμένει ανεξάρτητο από συγκεκριμένες υλοποιήσεις, διατηρώντας την αρχή της αντιστροφής εξαρτήσεων (Dependency Inversion Principle - DIP).

Μέσω αυτής της αρχιτεκτονικής προσέγγισης, το **Επίπεδο Τομέα** παρέχει έναν σαφή διαχωρισμό ανάμεσα στη βασική λογική λύσης και τις τεχνικές λεπτομέρειες της εφαρμογής. Αυτό καθιστά την εφαρμογή πιο επεκτάσιμη και ευέλικτη, διευκολύνοντας την ενσωμάτωση νέων στρατηγικών επίλυσης και μηχανισμών επικύρωσης στο μέλλον.

Το **Domain Layer** περιλαμβάνει τις βασικές οντότητες και αντικείμενα αξίας (Value Objects), που αποτελούν το θεμέλιο της εφαρμογής.

3. Υλοποίηση Σχεδιαστικών Προτύπων και Στρατηγικών Επίλυσης

3.1 Στρατηγική *BacktrackingSolver*

Η **BacktrackingSolver** υλοποιεί τη στρατηγική αναδρομικής δοκιμής (backtracking), η οποία επιτρέπει τη διερεύνηση πιθανών αριθμών για κάθε κελί. Η διαδικασία αυτή βασίζεται σε αναδρομικές κλήσεις, όπου δοκιμάζεται κάθε πιθανός αριθμός σε ένα κενό κελί, και αν μια επιλογή αποδειχθεί λανθασμένη, το πρόγραμμα αναιρεί την τελευταία αλλαγή (backtrack) και δοκιμάζει την επόμενη πιθανή τιμή. Αυτή η τεχνική καθιστά τον αλγόριθμο ιδιαίτερα αποδοτικό για μικρού μεγέθους πλέγματα Sudoku, όμως μπορεί να γίνει αναποτελεσματική σε πολύπλοκα Sudoku λόγω της εκθετικής αύξησης του χώρου αναζήτησης.

Η υλοποίηση της **BacktrackingSolver** περιλαμβάνει τη μέθοδο **Solve()**, η οποία εκτελεί τον αλγόριθμο αναδρομικής δοκιμής. Χρησιμοποιεί βοηθητικές μεθόδους για τον έλεγχο της εγκυρότητας των αριθμών και την εύρεση των κενών κελιών. Παρά την απλότητά της, η μέθοδος αυτή εξασφαλίζει ότι ο αλγόριθμος θα βρει πάντοτε μια έγκυρη λύση αν υπάρχει.

3.2 Στρατηγική **DFSArrayListSolver**

Η στρατηγική **DFSArrayListSolver** βασίζεται στην τεχνική **Βαθιάς Αναζήτησης (Depth-First Search - DFS)**, η οποία ακολουθεί μια διαφορετική προσέγγιση επίλυσης. Αντί να βασίζεται αποκλειστικά σε αναδρομή, χρησιμοποιεί μια **δυναμική λίστα (ArrayList)** για να αποθηκεύει τις ενδιάμεσες καταστάσεις του πλέγματος Sudoku.

Η **DFSArrayListSolver** λειτουργεί δημιουργώντας έναν **εκτεταμένο χώρο αναζήτησης**, όπου κάθε πιθανή κίνηση εξετάζεται σε βάθος πριν επιστρέψει προς τα πίσω αν η τρέχουσα διαδρομή δεν οδηγεί σε λύση. Αυτή η τεχνική μπορεί να είναι πιο αποδοτική από την απλή αναδρομική δοκιμή σε περιπτώσεις όπου το Sudoku έχει πολλές πιθανές λύσεις ή περιορισμούς, καθώς αποθηκεύει και διαχειρίζεται τις καταστάσεις δυναμικά, αντί να βασίζεται μόνο στο στοίβο αναδρομής του προγράμματος.

Η υλοποίηση της **DFSArrayListSolver** απαιτεί τη διαχείριση μιας στοίβας από αντικείμενα που αντιπροσωπεύουν τις καταστάσεις του Sudoku. Σε κάθε βήμα, η μέθοδος **Solve()** προσπαθεί να προχωρήσει βαθύτερα στη λύση, εξετάζοντας πιθανές τιμές και αποθηκεύοντας τις καταστάσεις σε μια λίστα. Αν φτάσει σε αδιέξοδο, επιστρέφει στην προηγούμενη κατάσταση και συνεχίζει την αναζήτηση.

3.3 Σχεδιαστικά Πρότυπα και Υλοποίηση

Η επιλογή στρατηγικής επίλυσης του Sudoku υλοποιείται μέσω του σχεδιαστικού προτύπου **Strategy**, το οποίο επιτρέπει την εύκολη εναλλαγή αλγορίθμων χωρίς την ανάγκη αλλαγών στην κύρια ροή του προγράμματος. Η υλοποίηση περιλαμβάνει το interface **ISudokuSolver**, το οποίο ορίζει τη μέθοδο **Solve()**, και τις διαφορετικές στρατηγικές που την υλοποιούν.

Η ενσωμάτωση των στρατηγικών στην εφαρμογή γίνεται μέσω **Dependency Injection (DI)**. Το **Infrastructure Layer** διαχειρίζεται τη δημιουργία των αντικειμένων επίλυσης και επιτρέπει τη δυναμική επιλογή στρατηγικής από τον χρήστη ή το σύστημα. Αυτή η προσέγγιση βελτιώνει την επεκτασιμότητα της εφαρμογής, επιτρέποντας την προσθήκη νέων αλγορίθμων χωρίς αλλαγές στον υπάρχοντα κώδικα.

4 Code

Ενδεικτικά η κυρίως κλάση του προγράμματος. Για να μπορείτε να κατεβάσετε και να τρεξετε την Solution παρακαλώ απευθυνθείτε στην τρεχούσα URL :

<https://github.com/georgiosbalatzis/SudokuSolverFinal> ενναλακτικά στο επισυναπτομενο αρχείο.

```
using SudokuSolver.Domain.Entities;
using SudokuSolver.Domain.Interfaces;
using SudokuSolver.Domain.Models;
```

```
using SudokuSolver.Infrastructure.IO;
using SudokuSolver.Infrastructure.Solvers;
using SudokuSolver.Infrastructure.Validators;
using SudokuSolver.Presentation.ConsoleUI;

public class Program
{
    private readonly ISudokuSolver[] _solvers;
    private readonly IGridLoader _gridLoader;
    private readonly IGridPrinter _gridPrinter;

    public Program(
        ISudokuSolver[] solvers,
        IGridLoader gridLoader,
        IGridPrinter gridPrinter)
    {
        _solvers = solvers;
        _gridLoader = gridLoader;
        _gridPrinter = gridPrinter;
    }

    public void Run()
    {
        while (true)
        {
            try
            {
                DisplayMenu();
                var choice = GetUserChoice();
                if (choice == _solvers.Length + 1) break;

                var grid = LoadGrid();
                if (grid == null) continue;

                Console.WriteLine("\nΑρχικό Sudoku:");
                _gridPrinter.Print(grid);

                var result = _solvers[choice - 1].Solve(grid);
                DisplayResult(result);
            }
            catch (Exception ex)
            {
                Console.WriteLine($"Σφάλμα: {ex.Message}");
            }
        }
    }
}
```



```

        Console.WriteLine("\nΠατήστε οποιοδήποτε πλήκτρο για συνέχεια...");
        Console.ReadKey();
        Console.Clear();
    }
}

private void DisplayMenu()
{
    Console.WriteLine("Καλώς ήρθατε στον Επιλυτή Sudoku!");
    Console.WriteLine("Επιλέξτε μέθοδο επίλυσης:");
    for (int i = 0; i < _solvers.Length; i++)
        Console.WriteLine($"{i + 1}: {_solvers[i].GetType().Name}");
    Console.WriteLine($"{_solvers.Length + 1}: Έξοδος");
}

private Grid LoadGrid()
{
    Console.Write("\nΕισάγετε το όνομα του αρχείου εισόδου: ");
    var fileName = Console.ReadLine();
    return _gridLoader.LoadFromFile(fileName);
}

private void DisplayResult(SolverResult result)
{
    if (result.Success)
    {
        Console.WriteLine("\nΛύση βρέθηκε!");
        Console.WriteLine($"Μέθοδος: {result.AlgorithmUsed}");
        Console.WriteLine($"Χρόνος επίλυσης: {result.SolvingTime.TotalSeconds:F3} δευτερόλεπτα");
        Console.WriteLine("\nΤελική λύση:");
        _gridPrinter.Print(result.Solution);
    }
    else
    {
        Console.WriteLine("\nΔεν βρέθηκε λύση για το συγκεκριμένο Sudoku.");
    }
}

private int GetUserChoice()
{
    while (true)
    {
        Console.Write("\nΕπιλογή: ");
        if (int.TryParse(Console.ReadLine(), out int choice) &&

```

```

        choice >= 1 && choice <= _solvers.Length + 1)
        return choice;

        Console.WriteLine("Μη έγκυρη επιλογή. Παρακαλώ προσπαθήστε ξανά.");
    }
}

public static void Main(string[] args)
{
    var validator = new SudokuValidator();
    var cellFinder = new CellFinder();

    var solvers = new ISudokuSolver[]
    {
        new BacktrackingSolver(validator, cellFinder), // Backtracking solver
        new DFSArrayListSolver(validator, cellFinder) // DFS solver
    };

    var gridLoader = new FileGridLoader();
    var gridPrinter = new GridPrinter();

    var program = new Program(solvers, gridLoader, gridPrinter);
    program.Run();
}
}

```

5 Επίλυση Αλγοριθμου

1. Μέθοδος Οπισθοδρόμησης (Backtracking): Η μέθοδος αυτή χρησιμοποιεί αναδρομή και βασίζεται στην ιδέα της δοκιμής και λάθους με συστηματικό τρόπο. Ας δούμε πώς λειτουργεί βήμα προς βήμα:

```

private bool SolveRecursive(Grid grid)
{
    // Βρίσκουμε το επόμενο κενό κελί
    var emptyCell = _cellFinder.FindEmptyCell(grid);
    // Αν δεν υπάρχει κενό κελί, το παζλ έχει λυθεί
    if (emptyCell == null) return true;
}

```

```

// Δοκιμάζουμε όλους τους πιθανούς αριθμούς (1-9)
for (int value = 1; value <= Grid.GridSize; value++)
{
    // Ελέγχουμε αν ο αριθμός είναι έγκυρος για αυτή τη θέση
    if (!_validator.IsValidMove(grid, emptyCell, value)) continue;

    // Τοποθετούμε τον αριθμό
    grid[emptyCell.Row, emptyCell.Col] = value;

    // Αναδρομική κλήση για το επόμενο κενό κελί
    if (SolveRecursive(grid)) return true;

    // Αν φτάσαμε σε αδιέξοδο, αναιρούμε την τοποθέτηση
    grid[emptyCell.Row, emptyCell.Col] = 0;
}

// Αν δεν βρέθηκε λύση, επιστρέφουμε false
return false;
}

```

Τα πλεονεκτήματα αυτής της μεθόδου είναι:

- Είναι απλή στην υλοποίηση
- Χρειάζεται λιγότερη μνήμη καθώς δουλεύει με ένα μόνο αντίγραφο του πίνακα
- Είναι αρκετά αποδοτική για τα περισσότερα παζλ

2. Μέθοδος Αναζήτησης κατά Βάθος με ArrayList (DFS ArrayList): Αυτή η μέθοδος χρησιμοποιεί μια επαναληπτική προσέγγιση με μια λίστα που λειτουργεί ως στοίβα:

```

public SolverResult Solve(Grid initialGrid)
{
    var startTime = DateTime.Now;
    // Δημιουργούμε μια λίστα και προσθέτουμε την αρχική κατάσταση
    var stack = new List<Grid> { initialGrid.Clone() };
    Grid solution = null;

    while (stack.Count > 0 && solution == null)
    {
        // Παίρνουμε την τελευταία κατάσταση από τη λίστα
        var current = stack[^1];
        stack.RemoveAt(stack.Count - 1);

        // Αν το παζλ είναι συμπληρωμένο, έχουμε λύση
        if (_validator.IsComplete(current))
        {
            solution = current;
            break;
        }
    }
}

```

```

    }

    // Βρίσκουμε το επόμενο κενό κελί
    var emptyCell = _cellFinder.FindEmptyCell(current);
    if (emptyCell == null) continue;

    // Δοκιμάζουμε όλες τις πιθανές τιμές (από το 9 προς το 1)
    for (int value = Grid.GridSize; value >= 1; value--)
    {
        if (!_validator.IsValidMove(current, emptyCell, value)) continue;

        // Δημιουργούμε νέα κατάσταση και την προσθέτουμε στη λίστα
        var next = current.Clone();
        next[emptyCell.Row, emptyCell.Col] = value;
        stack.Add(next);
    }
}

var endTime = DateTime.Now;
return new SolverResult(solution != null, solution,
    endTime - startTime, "DFS ArrayList");
}

```

Τα πλεονεκτήματα αυτής της μεθόδου είναι:

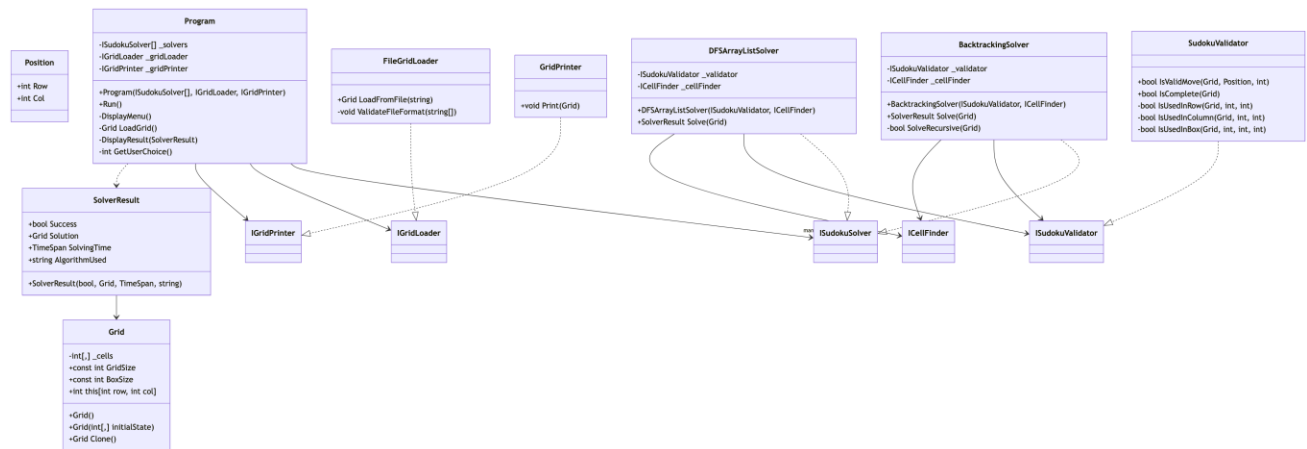
- Είναι πιο εύκολο να παρακολουθήσουμε την πρόοδο της επίλυσης
- Μπορεί να διακοπεί και να συνεχιστεί εύκολα
- Δεν υπάρχει κίνδυνος υπερχείλισης της στοίβας κλήσεων

Η βασική διαφορά μεταξύ των δύο μεθόδων είναι ο τρόπος που διαχειρίζονται τη μνήμη και την αναζήτηση:

1. Η μέθοδος οπισθοδρόμησης χρησιμοποιεί τη στοίβα κλήσεων του συστήματος μέσω της αναδρομής και τροποποιεί τον ίδιο πίνακα συνεχώς.
2. Η μέθοδος DFS ArrayList διατηρεί ξεχωριστά αντίγραφα του πίνακα για κάθε κατάσταση και χρησιμοποιεί μια λίστα για να παρακολουθεί τις καταστάσεις προς εξερεύνηση.

Στην πράξη, η μέθοδος οπισθοδρόμησης είναι συνήθως πιο γρήγορη για μικρά έως μεσαία παζλ, ενώ η μέθοδος DFS ArrayList μπορεί να είναι πιο κατάλληλη για πολύ μεγάλα παζλ ή όταν θέλουμε να έχουμε περισσότερο έλεγχο στη διαδικασία επίλυσης.

6 UML ΔΙΑΓΡΑΜΜΑ



Το UML διάγραμμα δείχνει την αρχιτεκτονική του συστήματος που ακολουθεί τις αρχές του καθαρού κώδικα και του SOLID. Βασικές συνιστώσες:

1. Κεντρικές Κλάσεις:
 - a. Grid: Η βασική κλάση που αναπαριστά το ταμπλό του Sudoku
 - b. Position: Value object που αναπαριστά μια θέση στο πλέγμα
 - c. SolverResult: Περιέχει το αποτέλεσμα της επίλυσης
2. Διεπαφές (Interfaces):
 - a. ISudokuSolver: Ορίζει τη διεπαφή για τους επιλυτές
 - b. ISudokuValidator: Για την επικύρωση των κινήσεων
 - c. IGridLoader: Για τη φόρτωση παζλ από αρχεία
 - d. IGridPrinter: Για την εκτύπωση του πλέγματος
 - e. ICellFinder: Για την εύρεση κενών κελιών
3. Υλοποιήσεις:
 - a. Δύο υλοποιήσεις του ISudokuSolver:
 - i. BacktrackingSolver
 - ii. DFSArrayListSolver
 - b. SudokuValidator: Υλοποίηση του ISudokuValidator
 - c. FileGridLoader: Υλοποίηση του IGridLoader
 - d. GridPrinter: Υλοποίηση του IGridPrinter
4. Σχέσεις:
 - a. Η κλάση Program εξαρτάται από τις διεπαφές και όχι από συγκεκριμένες υλοποιήσεις (Dependency Inversion Principle)
 - b. Οι solvers χρησιμοποιούν validator και cell finder μέσω των διεπαφών τους
 - c. Το SolverResult περιέχει αναφορά σε ένα Grid

7 Αποτελεσματα

```
/Users/giorgos/Downloads/00P_AEGEAN/SudokuSolver/bin/Debug/net8.0/SudokuSolver
Καλώς ήρθατε στον Επιλυτή Sudoku!
Επιλέξτε μέθοδο επίλυσης:
1: BacktrackingSolver
2: DFSArrayListSolver
3: Εξοδος

Επιλογή: 1

Εισάγετε το όνομα του αρχείου εισόδου: ghfajskglhasuigalj

Σφάλμα: Το αρχείο δεν βρέθηκε.

Πατήστε οποιοδήποτε πλήκτρο για συνέχεια...
```

Με λανθος file path.

Εισάγετε το όνομα του αρχείου εισόδου: /Users/giorgos/Downloads/OOP_AEGEAN/SudokuSolver/txt/p03.txt

Αρχικό Sudoku:

```
-----  
| . 2 7 | . 5 . | . 1 3 |  
| 9 1 5 | . . . | 6 . 7 |  
| . 8 3 | . 1 . | . . . |  
-----  
| . 6 . | 1 2 9 | . 3 . |  
| . 3 2 | 8 . 5 | 4 7 . |  
| 5 . 8 | 3 . 4 | . . . |  
-----  
| . . 1 | 2 . . | . . 5 |  
| . . . | . 8 1 | . 2 6 |  
| . . . | 7 4 . | 8 . . |  
-----
```

Λύση βρέθηκε!

Μέθοδος: Backtracking

Χρόνος επίλυσης: 0,039 δευτερόλεπτα

Τελική λύση:

```
-----  
| 4 2 7 | 6 5 8 | 9 1 3 |  
| 9 1 5 | 4 3 2 | 6 8 7 |  
| 6 8 3 | 9 1 7 | 2 5 4 |  
-----  
| 7 6 4 | 1 2 9 | 5 3 8 |  
| 1 3 2 | 8 6 5 | 4 7 9 |  
| 5 9 8 | 3 7 4 | 1 6 2 |  
-----  
| 8 7 1 | 2 9 6 | 3 4 5 |  
| 3 4 9 | 5 8 1 | 7 2 6 |  
| 2 5 6 | 7 4 3 | 8 9 1 |  
-----
```

Λυση του Σουντοκου 3 με το BackTracking

```
Εισάγετε το όνομα του αρχείου εισόδου: /Users/giorgos/Downloads/OOP_AE6EAN/SudokuSolver/txt/p07.txt
```

Αρχικό Sudoku:

```
-----  
| . . 8 | . . 7 | . . . |  
| . 4 . | 9 . . | . 6 . |  
| 1 9 . | . 8 3 | 5 . . |  
-----  
| . . . | . . 8 | . 1 . |  
| 6 . . | . . . | . 4 . |  
| . . . | 7 . . | 3 . . |  
-----  
| 8 . 1 | . . 9 | 2 7 . |  
| . . 3 | . . . | . . . |  
| . . 6 | . . . | 4 . 5 |  
-----
```

Λύση βρέθηκε!

Μέθοδος: DFS ArrayList

Χρόνος επίλυσης: 0,091 δευτερόλεπτα

Τελική λύση:

```
-----  
| 5 6 8 | 4 2 7 | 1 9 3 |  
| 3 4 2 | 9 1 5 | 8 6 7 |  
| 1 9 7 | 6 8 3 | 5 2 4 |  
-----  
| 7 3 4 | 5 9 8 | 6 1 2 |  
| 6 8 5 | 1 3 2 | 7 4 9 |  
| 2 1 9 | 7 6 4 | 3 5 8 |  
-----  
| 8 5 1 | 3 4 9 | 2 7 6 |  
| 4 7 3 | 2 5 6 | 9 8 1 |  
| 9 2 6 | 8 7 1 | 4 3 5 |  
-----
```

Λυση του Σουντοκου 7 με το DFS ArrayList

8 Συμπέρασμα

Η παρούσα εργασία παρουσιάζει μια βέλτιστη αρχιτεκτονική επίλυσης του Sudoku, ακολουθώντας τις αρχές της **Clean Architecture**, του **Domain-Driven Design**, καθώς και των **αρχών SOLID**. Ο διαχωρισμός της επιχειρησιακής λογικής από την παρουσίαση και την υποδομή, σε συνδυασμό με τη χρήση σχεδιαστικών προτύπων όπως τα **Repository** και **Strategy**, καθιστά την εφαρμογή επεκτάσιμη και συντηρήσιμη, επιτρέποντας την εύκολη προσαρμογή και βελτίωσή της στο μέλλον.