Sudoku Solver

Projekt przygotowany na Modelowanie Obiektowe

Daniel Dobrowolski

Tomasz Janik

Spis treści

Opis projektu

Krótki opis projektu:

Projekt będzie polegał na przygotowaniu aplikacji posiadającej warstwę graficzną, która umożliwi użytkownikowi rozwiązanie wczytanego wcześniej przez niego sudoku. Program powinien zezwalać na wczytanie sudoku w formie tekstowej, lub też graficznej, Aplikacja zostanie podzielona na kilka warstw (modułów), z których każdy będzie miał odpowiednio wydzieloną odpowiedzialność. Dzięki takiemu zabiegowi będziemy w stanie w stosunkowo krótkim czasie stworzyć zarówno warstwę widoczną przez użytkownika jak i warstwę odpowiedzialną za procesowanie wczytanego przez użytkownika formatu danych zawierającego opis sudoku. Umożliwi nam to też sprawne rozwiązanie łamigłówki oraz późniejsze przekazanie jej do warstwy wyświetlającej wyniki użytkownikowi. Projekt powinien zostać przetestowany wykorzystując unit testy oraz testy komponentowe, czyli testy integracyjne. Aplikacja tworzona będzie z myślą o urządzeniach z systemem operacyjnym Windows 10.

Funkcjonalności programu:

Aplikacja powinna posiadać interfejs graficzny, który oprócz wyświetlania planszy 9x9 zbudowanej z kwadratów pozwoli dodatkowi użytkownikowi na integrację. Poprzez integrację rozumiane jest udostępnienie użytkownikowi przycisków pozwalających na:

- Wyczyszczenie planszy
- Wczytanie sudoku
- Automatyczne rozwiązanie sudoku
- Cofnięcie poprzedniego ruchu

Użytkownik powinien również posiadać możliwość wybrania dowolnego kwadratu, który jest składową planszy i wpisaniu w to miejsce wartości z przedziału 1-9, lub też usunięciu wcześniej obecnej tam wartości.

Program powinien umożliwić wczytanie danych do sudoku z różnych formatów. Zakładamy wspieranie następujących formatów:

- .txt
- .png

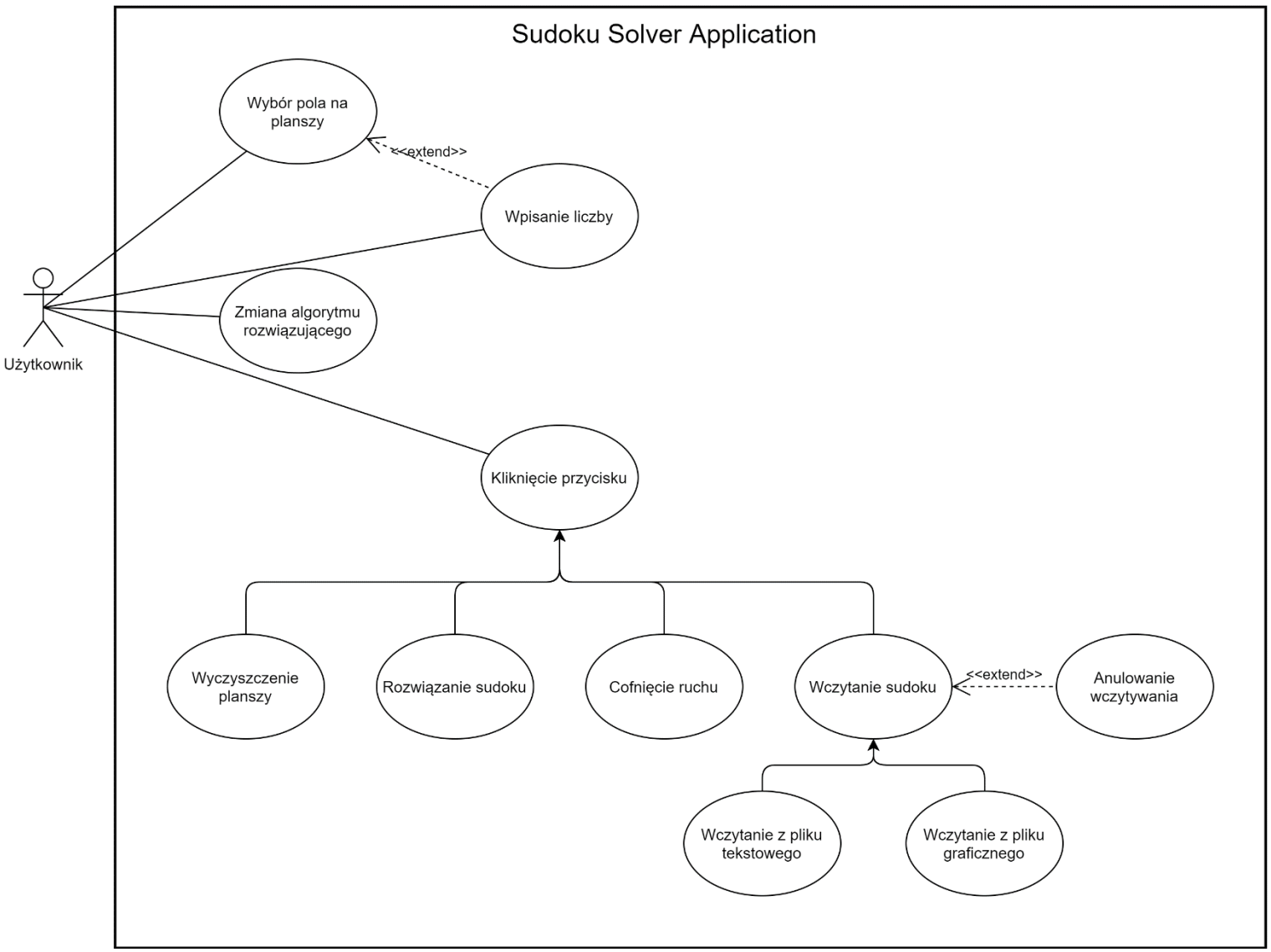
Plik tekstowy powinien zawierać dokładnie 81 symboli – każdy symbol reprezentuje wartość kolejnego pola sudoku. Dozwolonymi wartościami są cyfry od 1 do 9, a jeśli dane pole ma nie zawierać żadnej wartości początkowej to w odpowiadającym tej komórce miejscu w pliku tekstowym powinna znajdować się spacja lub kropka (‘.’).

Z kolei pliki graficzne powinny być zdjęciami sudoku znalezionego na przykład w gazecie, czy też narysowanego przez użytkownika czarnym długopisem na białej kartce. Na zdjęciu sudoku mogą znajdować się cyfry, których wartość i położenie powinny być z wysokim współczynnikiem poprawności rozpoznane przez program.

Ustalenia techniczne:

Projekt zostanie napisany w języku C#, a warstwa prezentacji widoczna dla użytkownika używać będzie Windows Presentation Foundation (WPF). Jako biblioteki do obróbki i procesowania zdjęć w celu wykrycia pozycji liczb jak i ich wartości użyta zostanie EmguCV. Projekt będzie posiadał testy napisane z wykorzystaniem NUnit, czyli biblioteki do języka C# stworzonej właśnie do takiego celu. Oprócz klasycznych testów jednostkowych projekt sprawdzony przy użyciu testu integracyjnego, który sprawdzać będzie średnią efektywność aplikacji przy rozpoznawaniu sudoku z formatów graficznych. Projekt powinien być łatwy w utrzymaniu oraz rozwoju, dlatego zdecydowaliśmy się na implementację kilkunastu wzorców projektowych, dzięki którym będziemy w stanie osiągnąć właśnie ten efekt.

Diagram przypadków użycia



Diagramy klas

Opis słowny

Projekt został stworzony przy użyciu wzorca architektonicznego **MVVM**, dlatego zdecydowaliśmy się na podzielenie diagramów klas na wyszczególnione w nim komponenty, jakimi są:

- Model
- View-Model
- View

Oprócz tego z uwagi na wielkość i skomplikowanie projektu zdecydowaliśmy się na wyodrębnienie w komponencie **View-Model** klas odpowiadających za wczytywanie sudoku z plików tekstowych oraz graficznych, czemu odpowiadają komponenty:

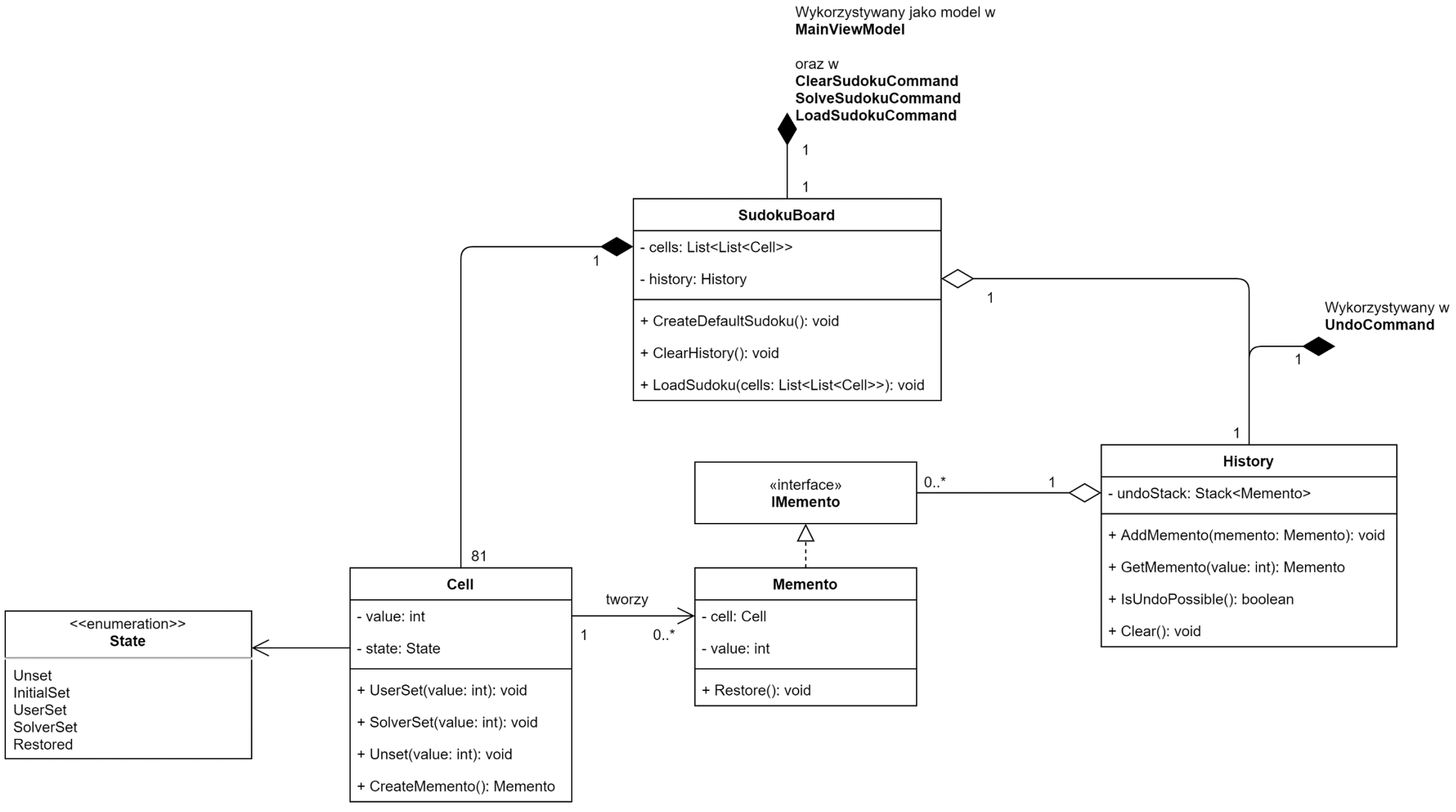
- View-ModelText
- View-ModelImage

Bez takiego podziału ukazanie View-Model jako pojedynczego komponentu było by skomplikowane i zajmowałoby za dużo miejsca, aby mogło zostać umieszczone w pliku PDF. Po podziale na podkomponenty przeglądanie oraz analiza programu staje się łatwiejsza.

Powodem takiej duża liczba oddanych klas jest wspomniane skomplikowanie projektu. Każda z części przedstawionych została opisana w sposób jak najbardziej minimalistyczny z użyciem tylko najważniejszych informacji, jednak przedstawione poniżej klasy są kluczowe dla projektu, ponieważ opisują jego kluczową funkcjonalność.

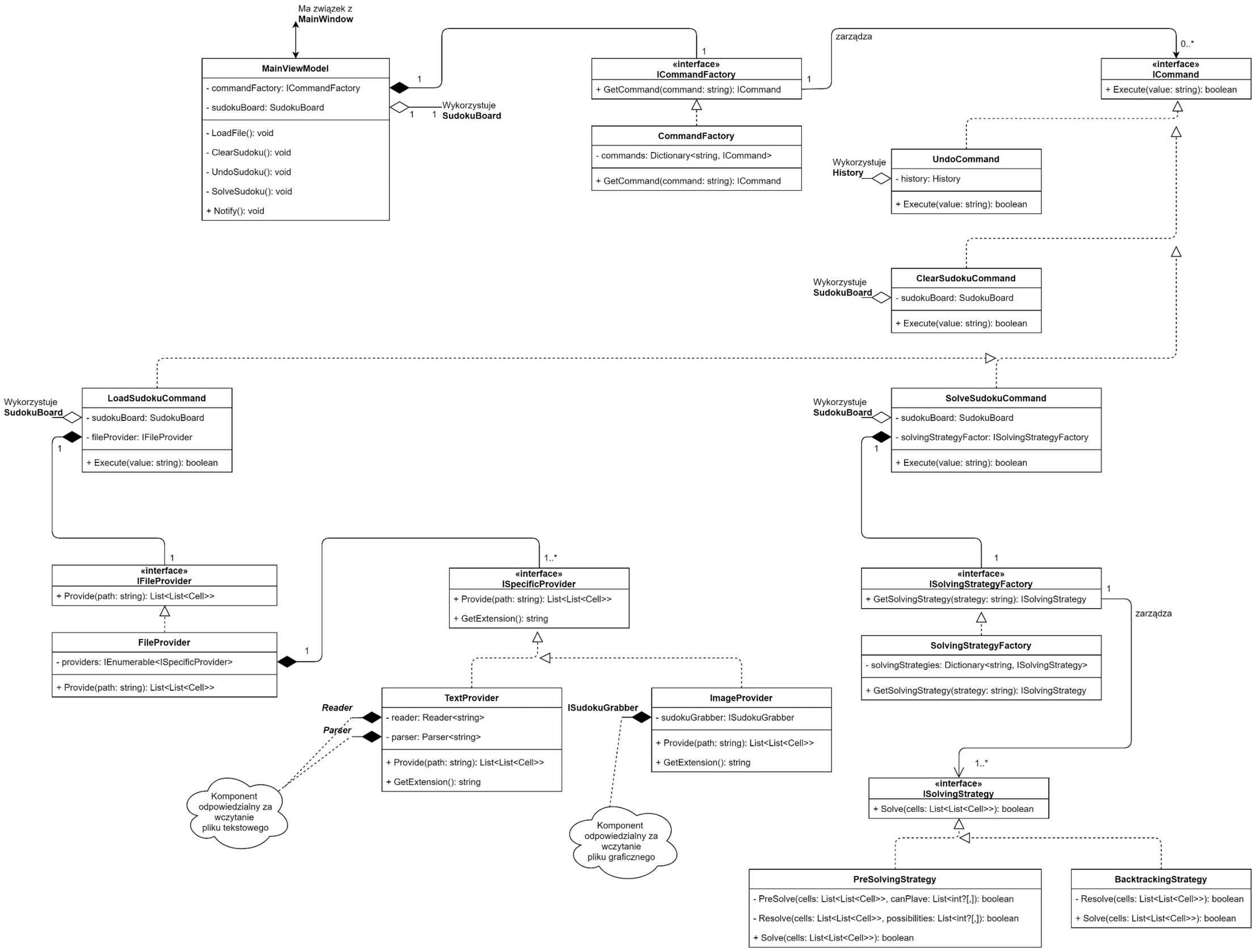
Link do pliku zawierającego cały diagram klas bez podziału na komponenty:
https://github.com/tomasz-janik/SudokuSolver/blob/master/diagrams/SudokuSolverClassDiagram.png

Model

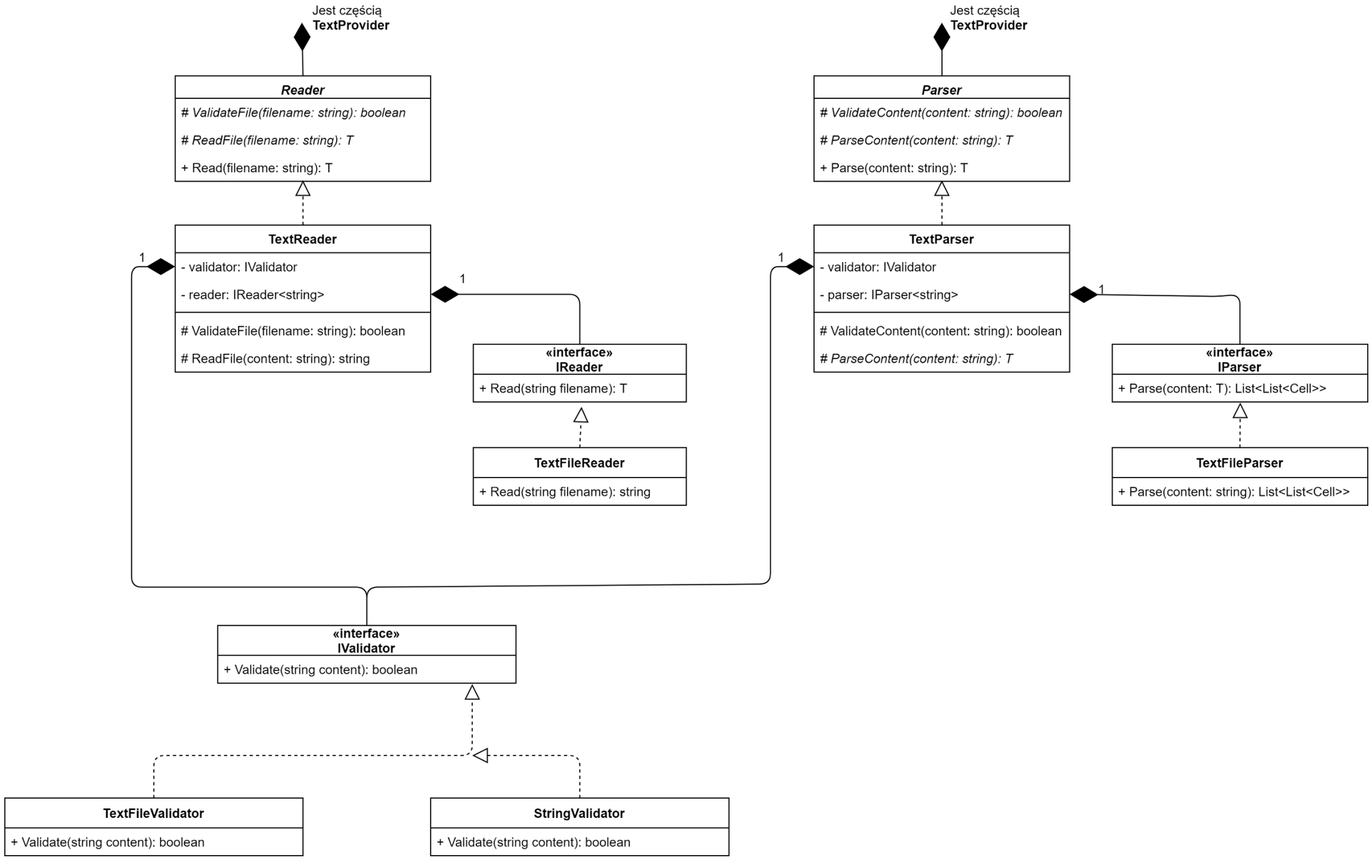

Wykorzystywany jako model w
MainViewModel
oraz w
ClearSudokuCommand
SolveSudokuCommand
LoadSudokuCommand
1
1
SudokuBoard
- cells: List<List<Cell>>
- history: History
+ CreateDefaultSudoku(): void
+ ClearHistory(): void
+ LoadSudoku(cells: List<List<Cell>>): void
1
1
Wykorzystywany w
UndoCommand
1
1
History
- undoStack: Stack<Memento>
+ AddMemento(memento: Memento): void
+ GetMemento(value: int): Memento
+ IsUndoPossible(): boolean
+ Clear(): void
«interface»
IMemento
0..*
1
81
Cell
- value: int
- state: State
+ UserSet(value: int): void
+ SolverSet(value: int): void
+ Unset(value: int): void
+ CreateMemento(): Memento
tworzy
1
0..*
Memento
- cell: Cell
- value: int
+ Restore(): void
<<enumeration>>
State
Unset
InitialSet
UserSet
SolverSet
Restored

ViewModel

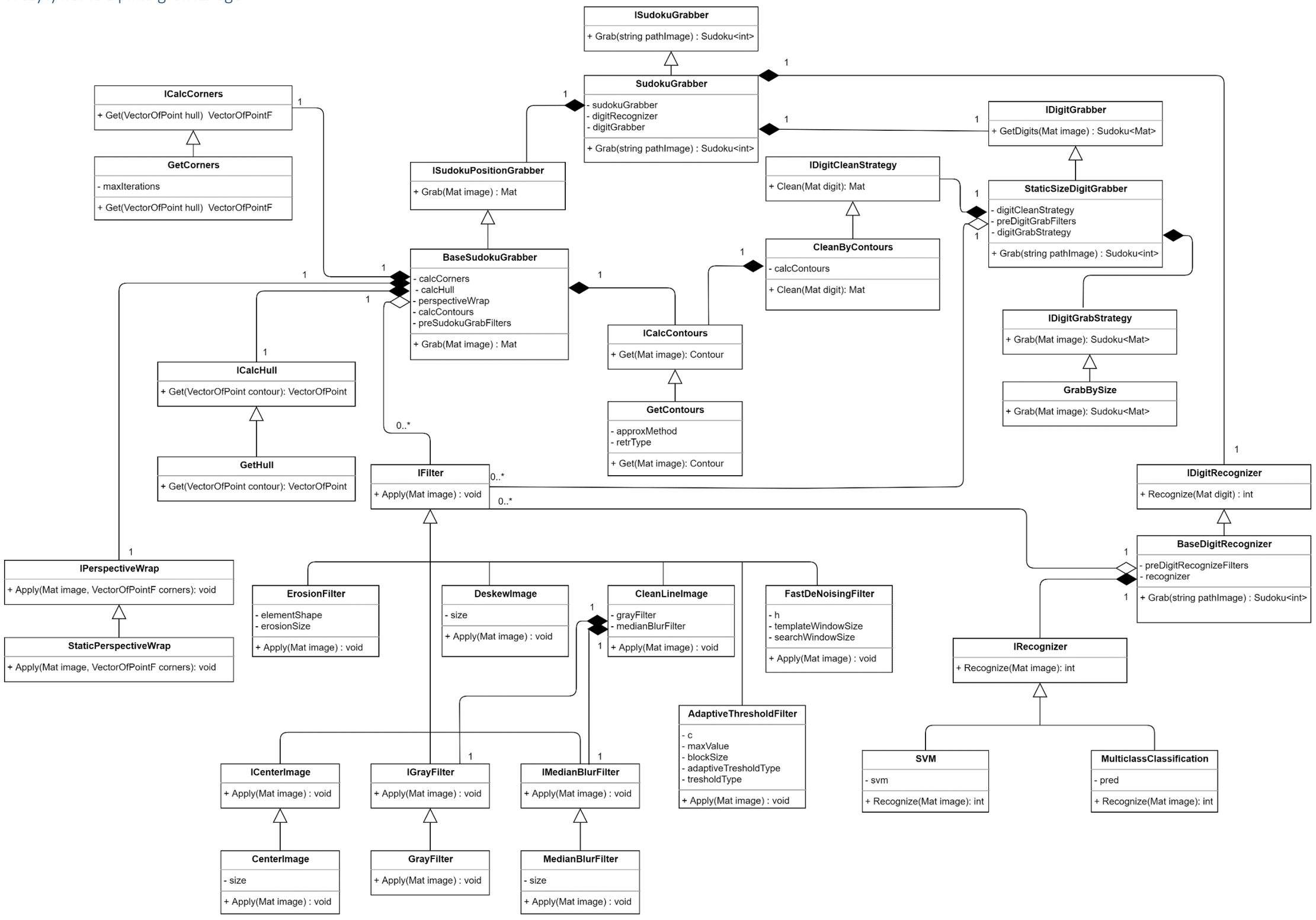
Główna funkcjonalność



Wczytywanie z pliku tekstowego



Wczytywanie z pliku graficznego



View

Ma związek z
MainViewModel

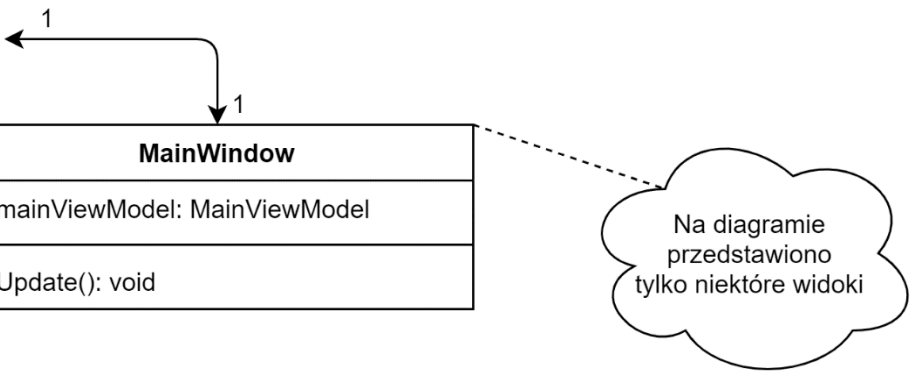
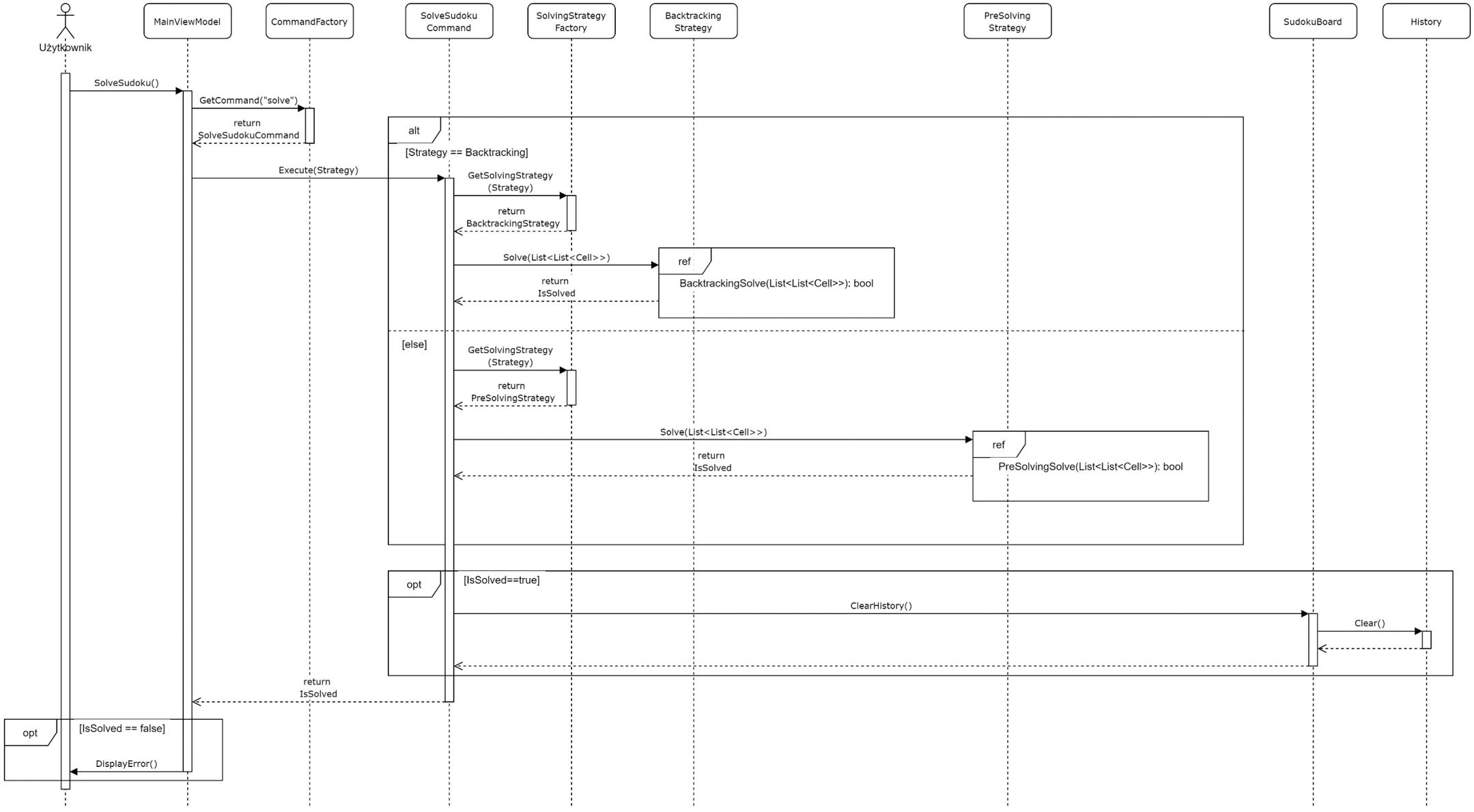
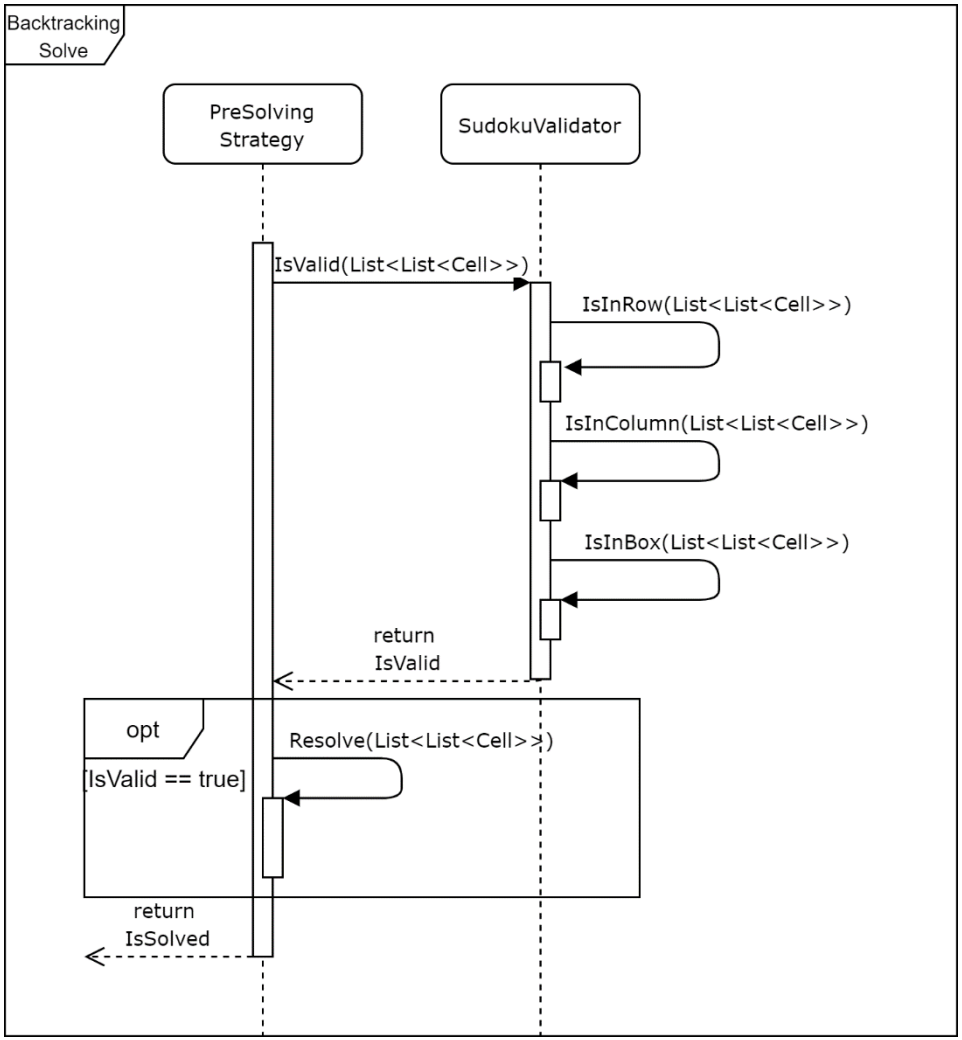


Diagram sekwencji dla przypadku użycia ‘Rozwiązanie sudoku’

Główny diagram



BacktrackingSolve



PreSolvingSolve

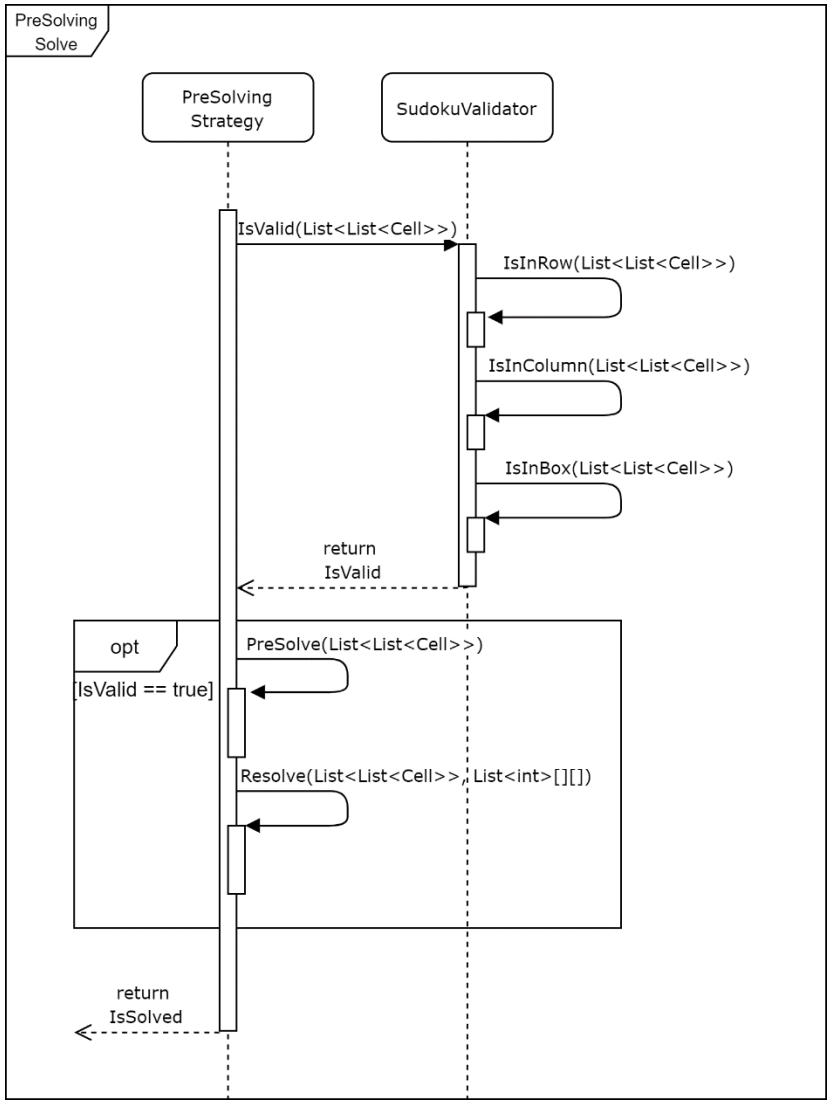


Diagram stanów dla klasy Cell (pojedyncza komórka na planszy)