ЗМІСТ

ВСТУП

Головоломки завжди користувалися популярністю як серед ентузіастів, так і серед тих, хто хоче покращити свої когнітивні здібності. На сучасному етапі розвитку комп'ютерних технологій дедалі більше логічних ігор переходять у формат Desktop або мобільних застосунків. Такі ігри дозволяють користувачам не лише приємно проводити час, але й розвивати своє мислення, уважність та стратегічне планування.

Однією з таких ігор є «Kakuro», яка відома своєю складністю та необхідністю застосування аналітичних навичок. Ключові правила гри полягають у тому, що кожну клітинку можна заповнювати числами від 1 до 9, дотримуючись підказок у чорних клітинках, які вказують на суму чисел у сусідніх білих клітинках. Числа в послідовних білих клітинках повинні бути унікальними, в результаті чого процес гри стає захоплюючим і складним водночас. Завдяки цим особливостям, «Kakuro» цікава як для новачків, так і для досвідчених шанувальників головоломок.

Розробка гри у вигляді Desktop-застосунку дозволить створити зручний графічний інтерфейс користувача та забезпечити інтерактивність. Цей застосунок може бути використаний і як інструмент для тренування когнітивних здібностей, і як просто для приємного проведення часу. Додаткові можливості, такі як фіксація часу, нічний режим, автоматичне відправлення рішень та інтеграція з файловою системою для збереження результатів гри, роблять його зручним і функціональним для користувачів різного рівня.

Метою цього курсового проєкту є розробка Desktop-застосунку гри «Kakuro» з використанням технології .NET Core - WPF. У процесі розробки застосунку реалізовані різні рівні складності (Easy, Normal, Hard), а також функції, що дозволяють користувачам налаштовувати гру під свої потреби: автоматична відправка рішень, можливість відображення збережень, таймер, нічний режим, система підказок і виправлення помилок. Розробка виконана у середовищі Visual Studio 2022.

1 ПОСТАНОВКА ЗАДАЧІ

1.1 Призначення, мета створення програмного застосунку і область застосування

Програмний застосунок (далі ПЗ) "Kakuro" призначений для автоматизації процесу гри в логічну головоломку Kakuro. Метою створення даного ПЗ є забезпечення користувачів інтуїтивно зрозумілим інструментом для гри в Kakuro, який дозволяє створювати нові ігри, зберігати і відновлювати ігрові стани, а також зберігати результати в таблицях рейтингів.

1.2 Вимоги до програмного застосунку

Основні вимоги програми включають генерацію ігрових полів, введення і перевірку чисел, збереження та відновлення стану гри, а також ведення таблиць рейтингів.

Для користувачів визначені наступні методи:

– автоматичне створення ігрового поля з числовими значеннями для заданого рівня складності;

– введення чисел у комірки поля;

– видалення введених значень;

– перевірка правильності введених значень і повідомлення користувача про успішність або невдачу;

– очищення поля;

– розпочати нову гру з новими числовими значеннями;

– збереження та відновлення стану гри;

– ведення таблиць рейтингу для кожного рівня складності.

1.2.1 вимоги до функціональних характеристик

Вхідними даними програми є:

– числові значення для створення ігрового поля в залежності від вибраного рівня складності;

– введені користувачем числа для заповнення комірок поля;

– інформація про стан гри для збереження та відновлення.

Вихідними даними програми є:

– згенероване ігрове поле з числовими значеннями;

– повідомлення про успішність або невдачу перевірки введених значень;

– збережений стан гри для відновлення;

– таблиці рейтингу з часом гри для кожного рівня складності.

1.2.2 вимоги до надійності

ПЗ має забезпечувати стійке функціонування в умовах можливих помилок користувача або збоїв системи. Для запобігання випадкового видалення та оновлення даних в файлах виводиться повідомлення для користувача.

1.3 Умови експлуатації програмного застосунку

Програмний засіб функціонує в операційні системі (далі ОС) Windows 10 на будь-яких сумісних комп'ютерах і не вимагає встановлення середовища .NET Core. Для його установки достатньо скопіювати папку з ПЗ, без потреби в інсталяції. Програма самостійно визначає шлях до даних, а якщо папки з інформаційною базою немає, вона буде створена автоматично при першому запуску.

1.3.1 вимоги до складу і параметрів технічних засобів

Конфігурація комп’ютеру повинна бути такою:

– процесор Intel Core i2 із частотою не нижче 1.5 ГГц;

– не менше 4 Гб RAM;

– не менше 2 Гб вільного місця на жорсткому диску;

– монітор з роздільною здатністю мінімально 1024x768 пікселів. Розмір - від 15 дюймів і більше;

– мишка, клавіатура USB чи сумісний вказівний пристрій.

1.3.2 вимоги до інформаційної і програмної сумісності

Необхідною вимогою до програмного забезпечення є встановлена ОС класу Windows 7/8/10. Найзручніший інтерфейс пропонує ОС Windows 10.

1.4 Вимоги до програмної документації

Програмна документація повинна бути підготовлена згідно з вимогами Єдиної Системи Програмної Документації (ЄСПД) і включати такі документи:

– керівництво користувача;

– опис застосування.

2 ОСНОВНА ЧАСТИНА

2.1 Методи та засоби програмування

Для розробки програмного забезпечення відповідно до індивідуального завдання курсового проєкту доцільно використати такі засоби реалізації:

- мова програмування C#;
- фреймворк .NET Core 8.0;
- технологія Windows Presentation Foundation (WPF);
- патерн проєктування MVVM;
- ін'єкція залежностей;
- формат файлів JSON;
- середовище розробки Microsoft Visual Studio 2022.

2.1.1 C# – це універсальна, типобезпечна, об’єктно-орієнтована мова програмування, що прагне досягти балансу між простотою, виразністю та продуктивністю. Головним архітектором мови є Андерс Хейлсберг.

C# підтримує інкапсуляцію, успадкування та поліморфізм, використовуючи уніфіковану систему типів, де всі типи мають спільну базову функціональність, як-от метод ToString(). Окрім класів, мова підтримує інтерфейси, що дозволяють багаторазове успадкування, а також властивості й події для інкапсуляції стану та обробки змін.

C# включає елементи функціонального програмування, такі як передача функцій через делегати, лямбда-вирази та підтримка незмінних типів через записи.

Статична типізація в C# дозволяє виявляти помилки під час компіляції, спрощуючи управління великими проєктами та підвищуючи надійність. Строга типізація запобігає використанню невідповідних типів, наприклад, чисел з плаваючою комою там, де очікуються цілі числа [1].

C# автоматично керує пам'яттю через збирач сміття в Common Language Runtime, звільняючи програмістів від необхідності ручного управління пам'яттю. Використання покажчиків рідко потрібне й допускається лише в небезпечних блоках коду для критичних завдань [1].

2.1.2 Середовище виконання (також зване фреймворком) – це модуль, який можна завантажити та встановити. Середовище виконання складається із загальномовного середовища виконання (CLR), яке забезпечує автоматичне керування пам'яттю та обробку винятків, та бібліотеки базових класів (BCL), яка надає основні функції, такі як колекції, введення/виведення, обробка тексту, XML/JSON, мережа, шифрування, паралелізм.

До середовища виконання також може входити вищий прикладний рівень, що містить бібліотеки для розробки багатофункціональних клієнтських, мобільних або веб-додатків (див. рис. 2.1). Існують різні середовища виконання для різних типів додатків, а також для різних платформ [1].

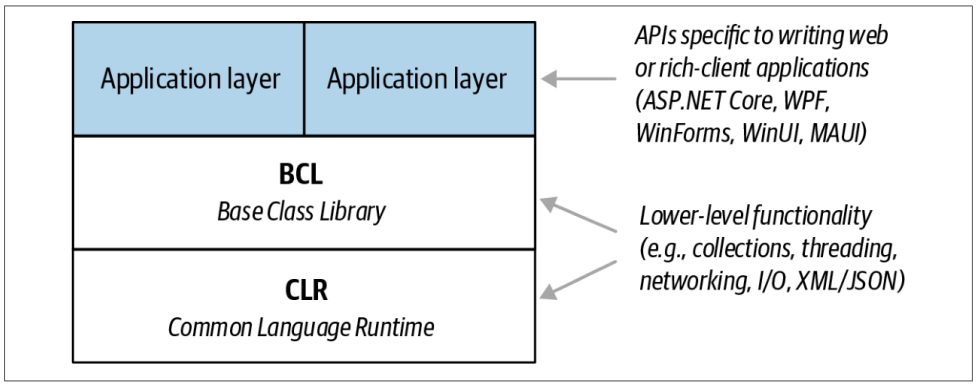

Рисунок 2.1 - Архітектура середовища виконання [1]

Під час написання програми розробник націлюється на певне середовище виконання, що означає, що програма використовує та залежить від функціональності, яку надає це середовище. Вибір середовища виконання також визначає, які платформи підтримуватиме програма (див. рис. 2.2) [1].

Application layer	CLR/BCL	Program type	Runs on...
ASP.NET	.NET 8	Web	Windows, Linux, macOS
Windows Desktop	.NET 8	Windows	Windows 10+
WinUI 3	.NET 8	Windows	Windows 10+
MAUI	.NET 8	Mobile, desktop	iOS, Android, macOS, Windows 10+
.NET Framework	.NET Framework	Web, Windows	Windows 7+

Рисунок 2.2 - Основні параметри середовища виконання [1]

.NET 8 – це відкритий флагманський фреймворк Microsoft, який підтримує розробку веб- та консольних програм для Windows, Linux і macOS, а також клієнтських і мобільних додатків для Windows 10+, macOS, iOS і Android. На відміну від .NET Framework, .NET 8 не встановлюється попередньо на Windows, і спроба запустити програму без середовища виконання призводить до повідомлення з посиланням на сторінку завантаження. Розробники можуть уникнути цього, створивши самодостатнє розгортання, яке містить усі необхідні компоненти [1].

2.1.3 Програми на основі інтерфейсу користувача можна розділити на дві категорії: тонкий клієнт, який є веб-сайтом, і повноцінний клієнт, що вимагає завантаження та установки на комп'ютер або мобільний пристрій. Для розширених клієнтських програм є вибір API: рівень робочого столу Windows включає популярні API WPF і Windows Forms.

WPF (Windows Presentation Foundation) – це технологія для створення настільних додатків на платформі Windows. Вона дозволяє розробникам створювати багатофункціональні програми з розширеними можливостями інтерфейсу користувача, які можуть працювати на комп'ютерах з Windows 7, 8, 10 і 11 [1].

WPF, представлений у 2006 році, суттєво вдосконалився в порівнянні зі своїм попередником, Windows Forms, завдяки використанню DirectX. Це дозволяє підтримувати складну графіку, включаючи 3D-рендерінг, мультимедіа та справжню прозорість, а також забезпечує правильне відображення при будь-яких

налаштуваннях DPI. WPF має розширену підтримку макета, що полегшує локалізацію програм, і використовує апаратне прискорення для більш швидкого рендерингу. Інтерфейси користувача описуються декларативно у файлах XAML, що сприяє відокремленню зовнішнього вигляду від функціональності. Проте, через його розмір і складність, WPF може вимагати більше часу для вивчення, а типи для написання програм знаходяться в просторі імен System.Windows [1].

2.1.4 Патерни проєктування – це загальноприйняті рішення для розв'язання поширених проблем у розробці програмного забезпечення. Вони є абстрактними концепціями, що можуть бути адаптовані до різних контекстів і потреб, допомагаючи розробникам створювати гнучкі та підтримувані системи.

Патерн MVVM забезпечує чітке відокремлення бізнес-логіки програми від її інтерфейсу користувача (UI). Він містить три основні компоненти: модель, представлення та модель представлення. Кожен служить певній меті. На рисунку 2.3 показані зв’язки між трьома компонентами [2].

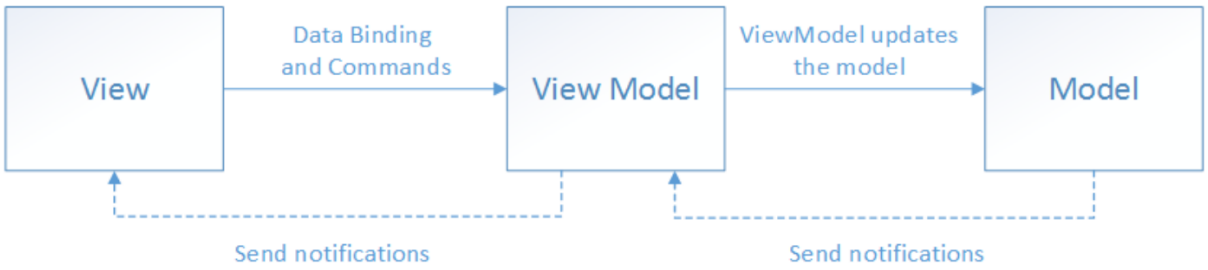

Рисунок 2.3 - Зв’язки між компонентами MVVM [2]

На високому рівні представлення «знає про» модель представлення, а модель представлення «знає про» модель, але модель не знає про модель представлення, а модель представлення не знає про представлення. Таким чином, модель представлення ізолює представлення від моделі та дозволяє моделі розвиватися незалежно від представлення.

Представлення відповідає за визначення структури, компонування та зовнішнього вигляду інтерфейсу користувача. Ідеально, кожне представлення описується в XAML з мінімумом коду, без бізнес-логіки. Проте іноді код може містити логіку інтерфейсу, наприклад, анімацію, яку важко реалізувати лише в XAML.

Модель представлення реалізує властивості та команди для прив'язки даних, сповіщаючи представлення про зміни через події. Властивості та команди моделі представлення визначають функціональність, тоді як представлення формує, як ці функції відображаються.

Класи моделі є невізуальними класами, які інкапсулюють дані програми, складаючи доменну модель, що включає бізнес-логіку та логіку перевірки. Вони зазвичай працюють із сервісами або репозиторіями для доступу до даних і кешування.

Шаблон MVVM пропонує підтримку існуючої бізнес-логіки через модель представлення як адаптер, що зменшує ризики при внесенні змін. Розробники можуть створювати модульні тести для моделі представлення та моделі незалежно від представлення, забезпечуючи сумісність між переробленими інтерфейсами та моделями. Це дозволяє дизайнерам і розробникам працювати одночасно: дизайнери зосереджуються на представленні, а розробники – на моделі представлення та моделі [2].

2.1.5 Ін'єкція залежностей дозволяє від'єднати конкретні типи від коду, що від них залежить, шляхом вказування залежностей як типів інтерфейсу, зазвичай через контейнер, який містить реєстрації та відображення між інтерфейсами, абстрактними та конкретними реалізаціями. Це спеціалізована форма інверсії контролю (IoC), де інший клас відповідає за впровадження залежностей у об'єкт під час виконання. Наприклад, клас ProfileViewModel ілюструє використання ін'єкції залежностей [2]:

```
private readonly ISettingsService _settingsService;
private readonly IAppEnvironmentService _appEnvironmentService;
```

```
public ProfileViewModel(
IAppEnvironmentService appEnvironmentService,
IDialogService dialogService,
INavigationService navigationService,
ISettingsService settingsService)
: base(dialogService, navigationService, settingsService)
{
_appEnvironmentService = appEnvironmentService;
_settingsService = settingsService;
// Omitted for brevity
}
```

Конструктор ProfileViewModel отримує інтерфейси як аргументи, що вводяться іншим класом. Це означає, що ProfileViewModel не знає про класи, які створюють екземпляри інтерфейсів. Клас, що відповідає за створення цих екземплярів, називається контейнером ін’єкції залежностей.

Контейнери ін’єкцій залежностей зменшують зв’язок між об’єктами, надаючи можливість створювати екземпляри класів та управляти їхнім життєвим циклом відповідно до конфігурації контейнера. При створенні об’єкта контейнер вставляє в нього необхідні залежності, а якщо ці залежності ще не існують, то спочатку створює та вирішує їх [2].

2.1.6 JSON став популярною альтернативою XML. Хоча йому бракує деяких розширених можливостей XML, таких як простори імен, префікси та схеми, він відрізняється простотою і чіткістю формату, схожого на те, що ви отримали б, конвертуючи об'єкт JavaScript у рядок. Це робить JSON зручним для використання в багатьох випадках [1].

Історично .NET не мав вбудованої підтримки JSON, і розробники змушені були покладатися на сторонні бібліотеки, переважно Json.NET. Хоча наразі в .NET є вбудовані рішення для роботи з JSON, бібліотека Json.NET все ще залишається популярною з кількох причин: вона існує з 2011 року, має той же API, що працює

на старих платформах .NET, і раніше вважалася більш функціональною, ніж Microsoft JSON API.

API Microsoft для JSON мають перевагу, оскільки були розроблені з нуля, щоб бути простими і надзвичайно ефективними. Починаючи з .NET 6, їх функціональність стала досить близькою до Json.NET [1].

2.1.7 Visual Studio є потужним інструментом для розробників, який дозволяє завершити весь цикл розробки в одному місці. Це комплексне інтегроване середовище розробки (IDE), призначене для написання, редагування, налагодження та створення коду, а також для подальшого розгортання програми. Visual Studio включає компілятори, інструменти автоматичного завершення коду, системи управління версіями, розширення та багато інших функцій, що покращують кожен етап процесу розробки програмного забезпечення (див. рис. 2.4) [3].

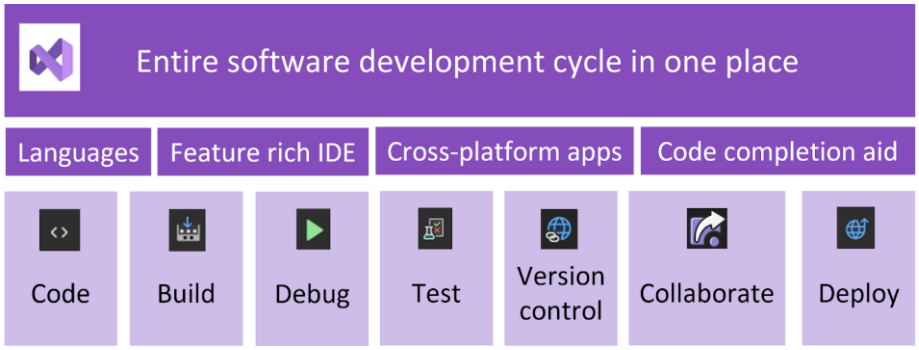

Рисунок 2.4 - Основні можливості Visual Studio [3]

3 СПЕЦІАЛЬНА ЧАСТИНА

3.1 Опис логічної структури програми

3.1.1 принцип генерації ігрового поля

Ігрове поле для «Kakuro» за замовчуванням генерується на рівні складності Easy.

Користувачеві паралельно надається можливість вибору складності: Easy, Normal, Hard.

Поле для гри «Kakuro» генерується на основі згенерованого шаблону.

Генерація шаблону відбувається у два етапи:

– вибір шаблону в залежності від складності: для кожного рівня складності в програмі зберігається матриця, елементами якої є “*” та порожні рядки. На місцях “*” в майбутньому полі розташовуються комірки, де користувач вводить значення, на місцях порожніх рядків - підказки у вигляді сум або порожні комірки. На рисунку 3.1 наведені графічні представлення матриць для кожного рівня складності, де “чорні” комірки - місця розташування “*”, “білі” комірки - суми-підказки або порожні комірки;

– випадкова інвертація поля: здійснюється проходження по матриці, під час якого кожен елемент із вірогідністю 50% може змінити значення на протилежне: ті, що містять порожні рядки, - на “*”, і навпаки, що додає унікальності кожному новому шаблону.

Представленням ігрового поля у програмі є вкладена колекція: вона може легко змінюватись динамічно, що необхідно для гнучкої багаторазової генерації поля.

Представленням комірки є об’єкт класу DashboardItem, його властивості:

– DisplayValue - значення, яке вводить користувач;

– HiddenValue - значення, згенероване під час генерації поля. Використовується для підрахунку сум-підказок;

– CellType - тип комірки: “порожня”, “сума-підказка” або “комірка-значення”;

– SumRight - сума комірок справа;

– SumBottom - сума комірок знизу.

Вкладена колекція ігрового поля наповнюється в п’ять етапів:

– здійснюється очищення від минулих елементів;

– в залежності від складності генерується шаблон;

– на основі шаблону вкладена колекція заповнюється об’єктами комірок. Коміркам, які в колекції розташовані там, де на шаблоні - “*”, встановлюється CellType “комірка-значення”;

– розрахунок сум комірок знизу: здійснюється прохід по вкладеній колекції знизу вгору. Якщо тип комірки - “комірка-значення”, то для HiddenValue генерується значення, унікальне по відношенню до суміжних комірок, і додається в змінну суми. Якщо тип комірки не “комірка-значення”, але якась сума вже була зібрана, то це значення привласнюється SumBottom, а CellType - “сума-підказка”, змінна суми онулюється;

– розрахунок сум комірок справа: відбувається за аналогією до розрахунку сум комірок знизу, окрім генерації значень для HiddenValue. Розраховані суми привласнюються SumRight.

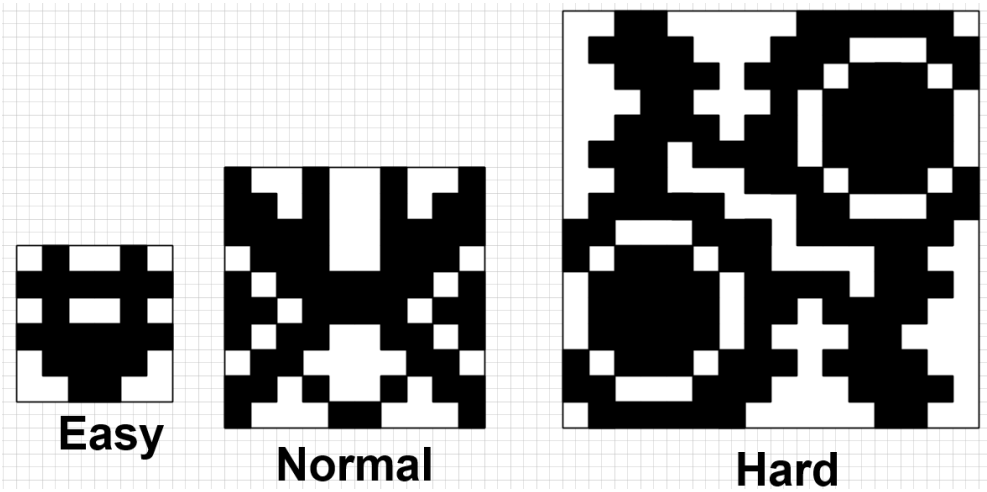

Рисунок 3.1 - Шаблони ігрового поля для кожного рівня складності

3.1.2 алгоритм визначення програмою успішного проходження гри користувачем

Алгоритм визначення успішного проходження гри користувачем складається з двох етапів перевірки:

– валідація вертикальних сум - SumBottom: здійснюється прохід по вкладеній колекцій знизу вгору. Якщо тип комірки - “комірка-значення”, то перевіряється, чи введене значення в DisplayValue і чи воно унікальне по відношенню до суміжних. Якщо одна з вимог не виконується, здійснюється вихід з методу з поверненням false. Якщо ці вимоги виконуються, то це значення додається в змінну суми. Якщо тип комірки не “комірка-значення”, але якась сума вже була зібрана, то це значення порівнюється з існуючим в цій комірці значенням SumBottom. Якщо вони збігаються, продовжується валідація, інакше - вихід з методу з поверненням false;

– валідація горизонтальних сум - SumRight: здійснюється за аналогією до валідації вертикальних сум, окрім валідації наявності та унікальності значень в комірках типу “комірка-значення”.

Алгоритм валідації поля наведений на рисунках 3.2 - 3.4.

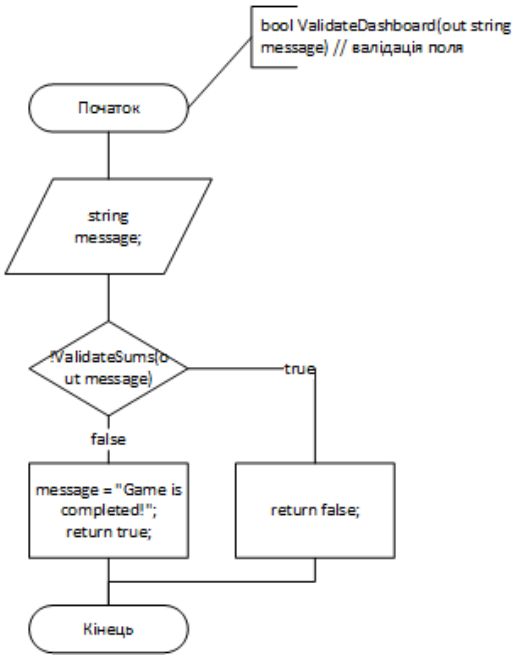

Рисунок 3.2 - Блок схема загального методу валідації поля

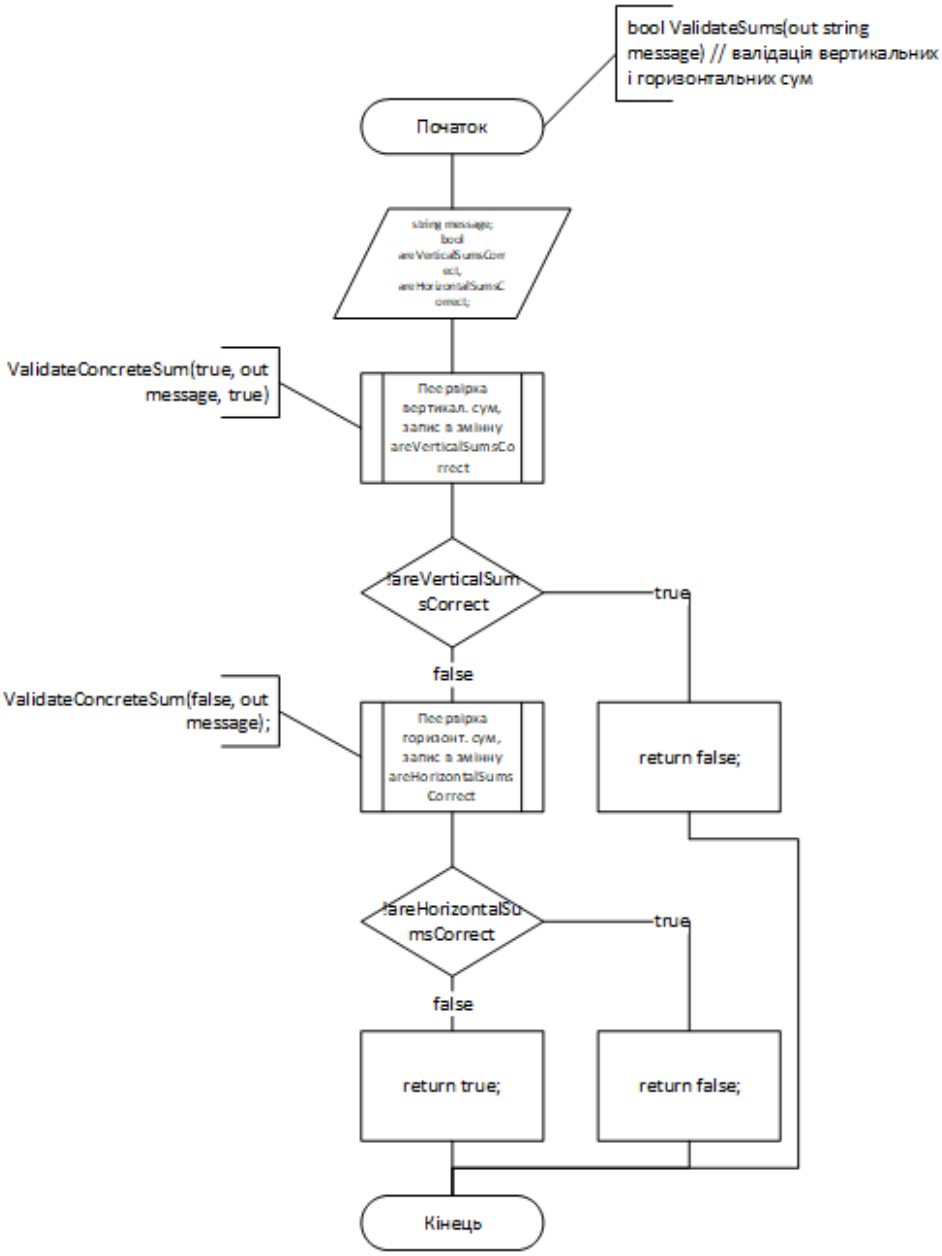

Рисунок 3.3 - Блок схема методу валідації сум

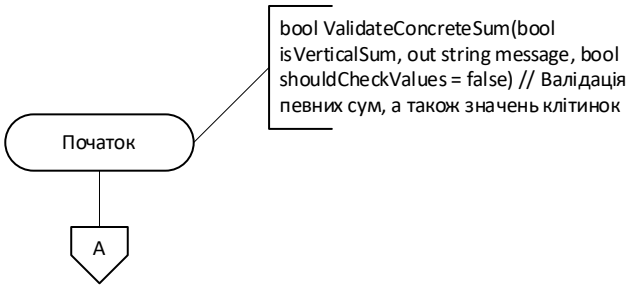

Рисунок 3.4 - Блок схема методу валідації окремого типу сум

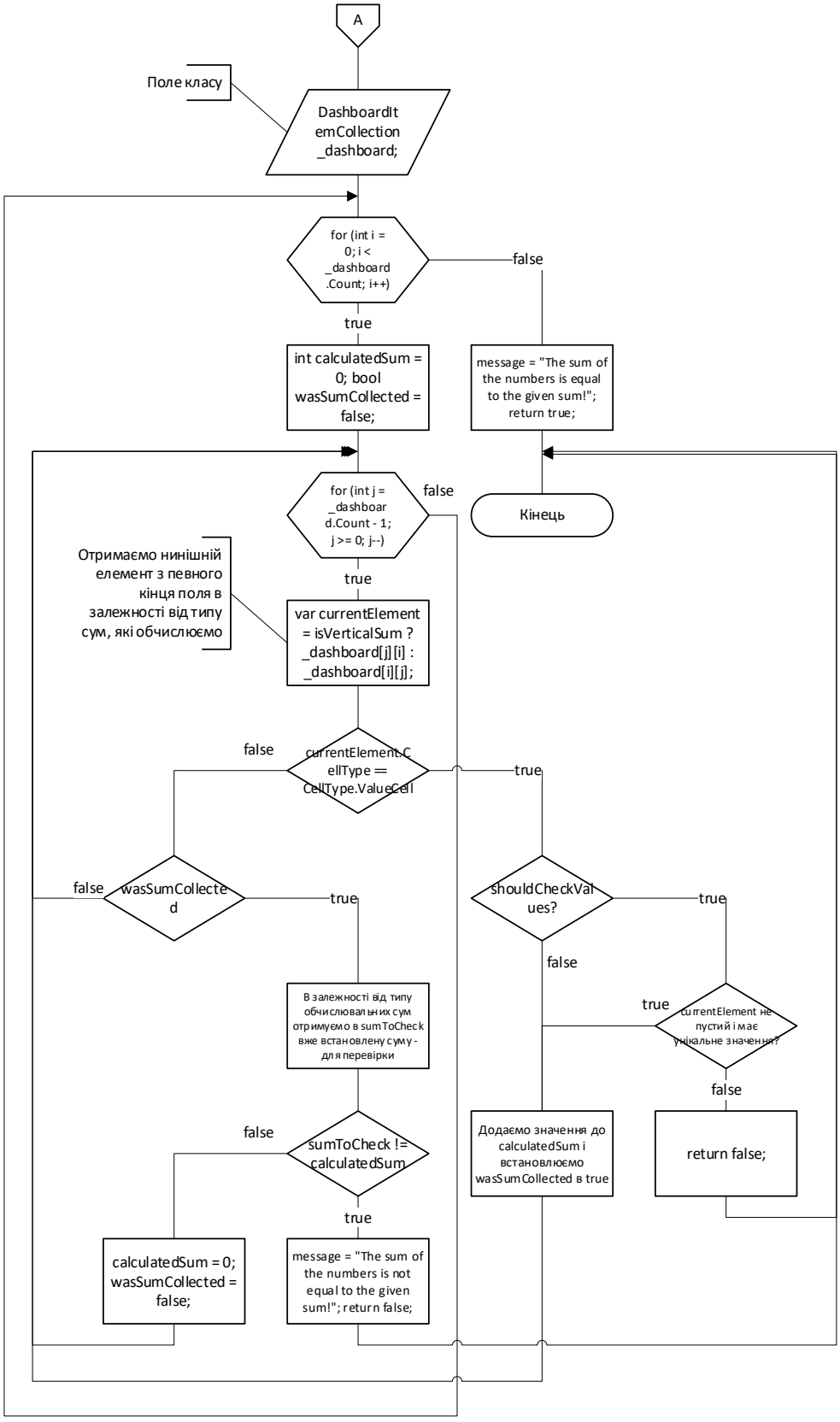

Рисунок 3.4, аркуш 2

3.1.3 сценарій шляху користувача під час роботи з застосунком

Функціонал застосунку і можливості користувача наведені на Use Case діаграмі на рисунку 3.5.

В застосунку передбачається 5 зон відповідальності, які представлені на Use Case діаграмі секціями. Кожна секція містить функції, які може виконати система на запит користувача.

“Game” містить підсекції:

– “Dashboard Actions” - функції над усім ігровим полем:

– “Choose difficulty” - обрати рівень складності. Викликає функцію “Generate dashboard”;

– “Start New Game” - розпочати нову гру. Викликає функцію “Generate dashboard”;

– “Generate dashboard” - згенерувати ігрове поле;

– “Full Clean” - очистити ігрове поле від уведених значень;

– “Validate” - валідувати введені значення. Викликає функцію “Game Completion Alert”;

– “Game Completion Alert” - вивід повідомлення після відправлення поля на валідацію. Повідомлення може бути про успіх (“Success”) або невдачу (“Fail”). В разі успіху викликається функція “Save Completion Time”;

– “Save Completion Time” - збереження часу, протягом якого була пройдена гра.

– “Dashboard Item Actions” - функції над окремою коміркою:

– “Enter Number” - ввести значення в комірку;

– “Clean Number” - видалити значення з комірки.

“Settings”:

– “Show Answers” - показати правильні значення;

– “Customize View” - налаштувати зовнішній вигляд ігрового поля або застосунку;

– “Use Tips” - увімкнути підказки.

“Rating Table” - відображення таблиць рейтингу для кожного рівня складності (“Watch Rating”).

“Savepoints”:

– “Add Savepoint” - зберігти стан ігрового поля;

– “Rewrite Savepoint” - перезаписати стан поля;

– “Delete Savepoint” - видалити збережений стан поля;

– “Load Savepoint” - завантажити стан поля.

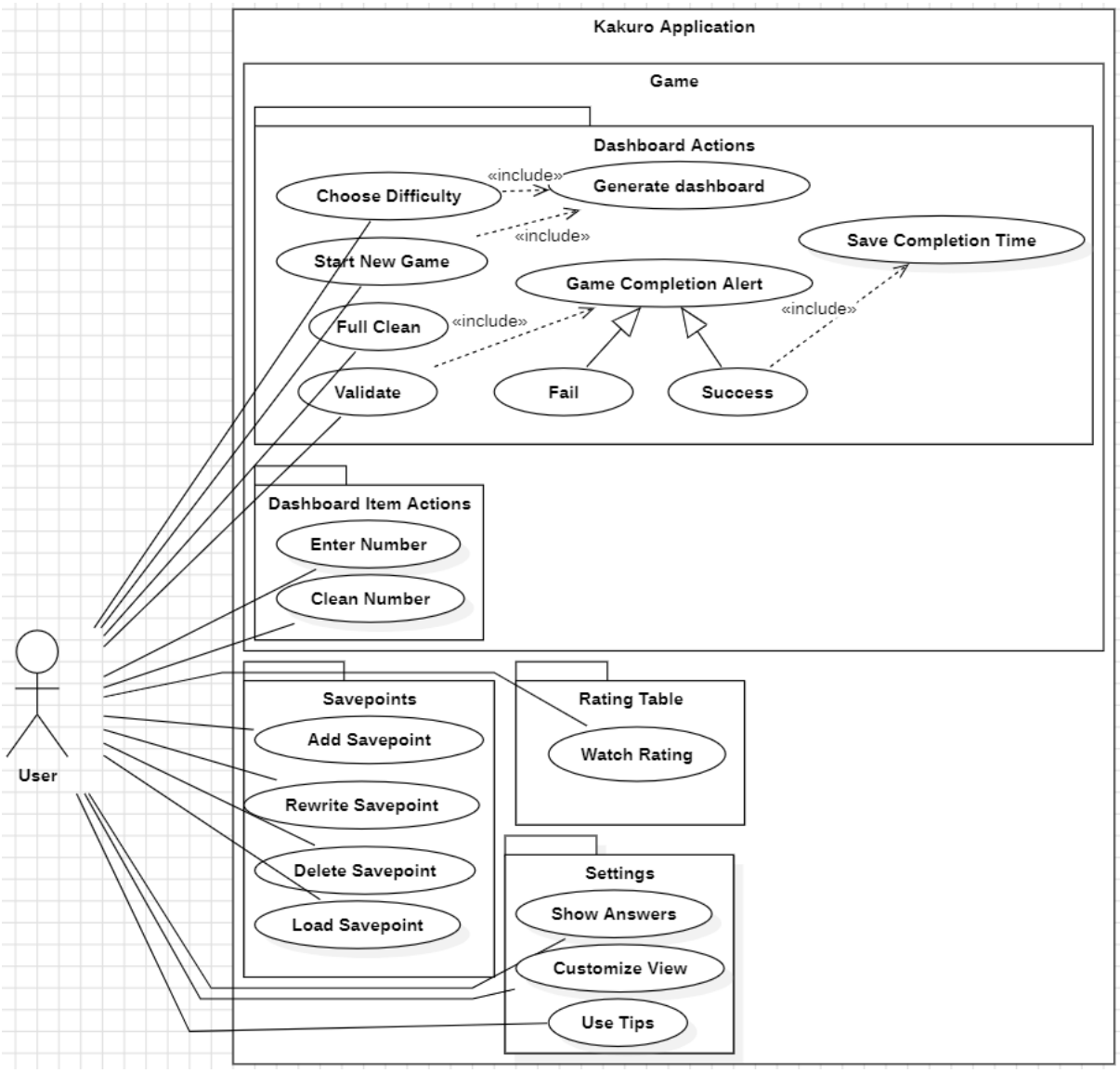

Рисунок 3.5 - Use case діаграма застосунку

3.2 Опис фізичної структури програми

3.2.1 опис файлової структури програмного проєкту

Відповідно до постановки задачі та логічної структури було розроблено програму-прототип гри “Kakuro” за допомогою середовища розробки MS Visual Studio 2022 та фреймворку WPF.

Програмний проєкт складається із залежних файлів типу .cs та .xaml. Дерево файлів проєкту застосунку наведено на рисунку 3.6. Скомпільований файл тестування визначених класів знаходиться у файлі Kakuro.exe.

Код програми складений на мові програмування C# та мові розмітки XAML і складається з визначення моделей, представлень, ViewModels, команд, інтерфейсів, користувацьких компонентів та інших додаткових класів.

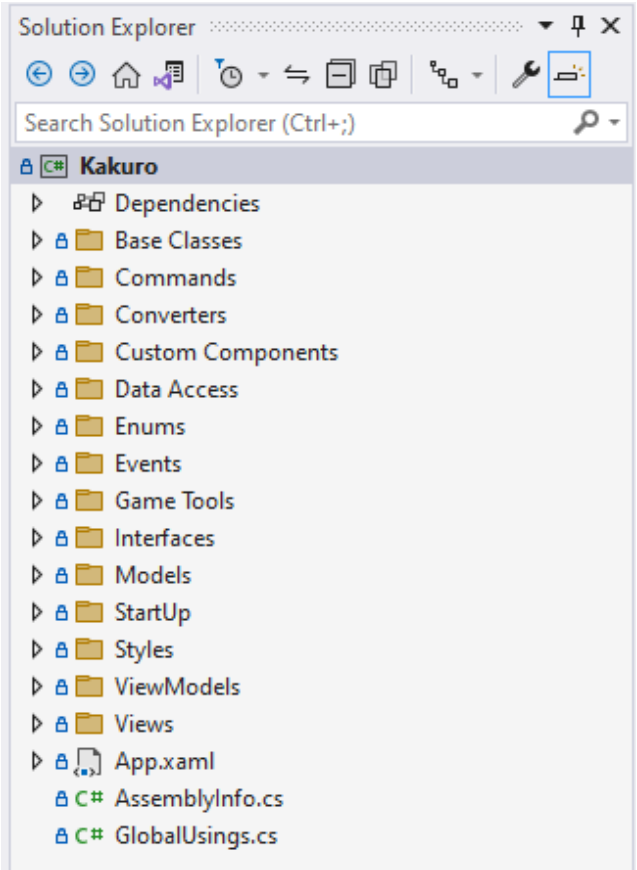

Рисунок 3.6 - Загальний вигляд дерева проєкту

Призначення директорій та файлів, наведених на рисунку 3.6:

– Base Classes - збереження базових класів, які надають спільну функціональність для інших класів у проекті;

– Commands - збереження команд - вказівок на виконання певних дій, які можна прив’язати до елементів управління в WPF;

– Converters - збереження класів конвертерів, які використовуються для перетворення значень між різними типами при прив'язці даних у WPF;

– Custom Components - збереження користувацьких незалежних компонентів. До них відносяться:

– SumCell - комірка ігрового поля, яка містить суми-підказки;

– RulesWindow - вікно, яке містить правила гри;

– ToastNotificationWindow - вікно для відображення повідомлень.

– Data Access - збереження класів, необхідних для підвантаження та збереження даних;

– Enums - збереження визначень перерахувань (enum), які використовуються для створення наборів іменованих констант;

– Events - збереження подій, за допомогою яких між ViewModels здійснюється обмін даними;

– Game Tools - збереження інструментів, які використовуються безпосередньо під час гри:

– FocusNavigationHelper - переміщення між комірками ігрового поля через зміну фокуса за допомогою клавіш клавіатури;

– OperationNotifier - повідомлення про успішність або невдачу певної операції;

– SolutionValidator - валідація введених користувачем значень.

– Interfaces - збереження інтерфейсів класів;

– Models - збереження моделей;

– StartUp - зберігання класів для конфігурації та ініціалізації залежностей додатку через реєстрацію служб, ViewModel, команд і доступу до даних;

– Styles - збереження стилей;

– ViewModels - збереження ViewModels;

– View - збереження графічних представлень;

– App.xaml - визначення ресурсів застосунку, включаючи стилі, шаблони та конвертери, які використовуються в додатку, а також для ініціалізації глобальних налаштувань;

– AssemblyInfo.cs - містить метадані про збірку (assembly) додатку, такі як назва, версія, автор, опис, а також інформацію про атрибути, які визначають характеристики збірки;

– GlobalUsings.cs - використовується для визначення глобальних директив using, які доступні у всіх файлах проекту.

3.2.2 визначення глобальних змінних та функцій визначених програмістом/розробником

Файл GlobalUsings.cs містить глобальні директиви using:

DashboardItemCollection - це псевдонім для типу ObservableCollection<ObservableCollection<DashboardItemViewModel>>, що представляє колекцію колекцій об'єктів DashboardItemViewModel.

RatingTableContainer - це псевдонім для типу Dictionary<DifficultyLevels, ObservableCollection<RatingRecord>>, який забезпечує структуру для зберігання записів таблиць рейтингу (RatingRecord), згрупованих за рівнями складності (DifficultyLevels).

Для зручного відображення даних у грі Kakuro створено компонент SumCell, що наслідує від класу Control і має дві залежні властивості: SumRight і SumBottom, які відображають значення сум у сусідніх клітинках. Клас також обробляє натискання клавіш для навігації.

RulesWindow — це WPF-вікно, що містить текстові правила гри, реалізовані за допомогою елемента Grid та TextBlock для зручності читання.

ToastNotificationWindow — вікно сповіщення, що показує короткі повідомлення з можливістю закриття; воно отримує текст, заголовок і параметр успішності, налаштовуючи фон вікна відповідно до статусу, а також містить анімацію, що переміщує його знизу вгору, з можливістю автоматичного закриття через певний час або за натисканням кнопки.

Лістинг користувацьких компонентів наведено у додатку Б.

3.2.3 макети вікон застосунку

Усі програма розділена на 4 секції (див. рис. 3.7-3.8):

– Game;

– Settings;

– Savepoints;

– Rating Tables.

Вигляд користувацьких вікон наведено на рисунках 3.9-3.10.

Лістинг вікон та компонентів User Control наведено у додатку Б.

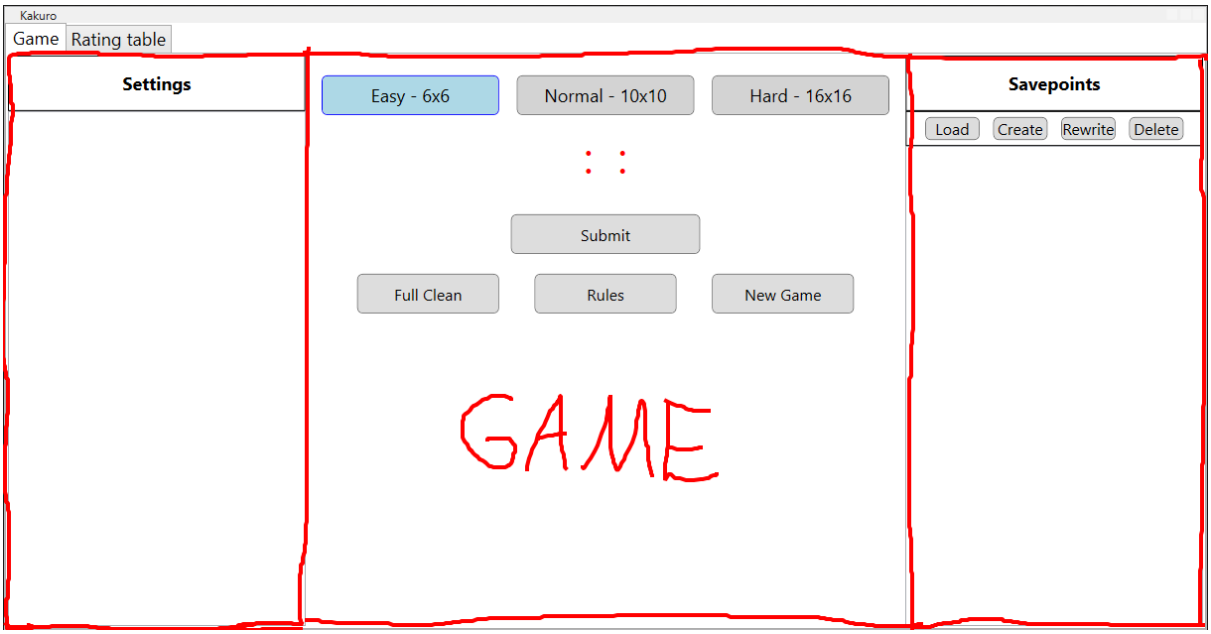

Рисунок 3.7 - Перша вкладка застосунку із секціями Settings, Game, Savepoints

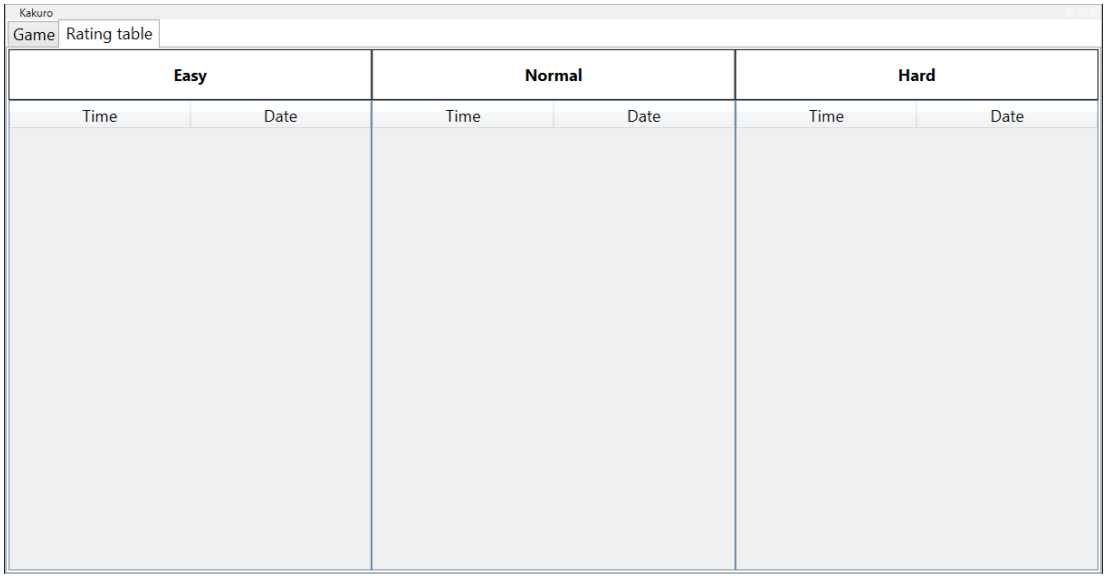

Рисунок 3.8 - Друга вкладка застосунку із секцією Rating Tables

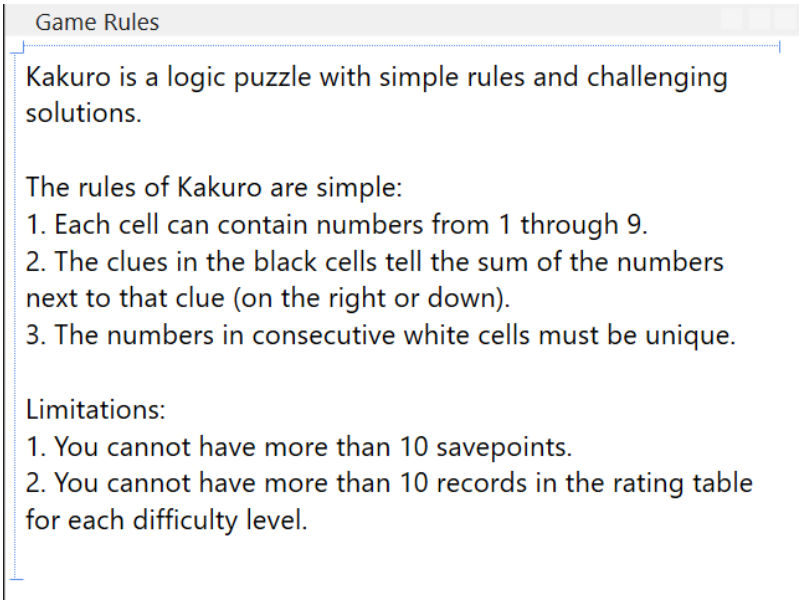

Рисунок 3.9 - Вікно RulesWindow

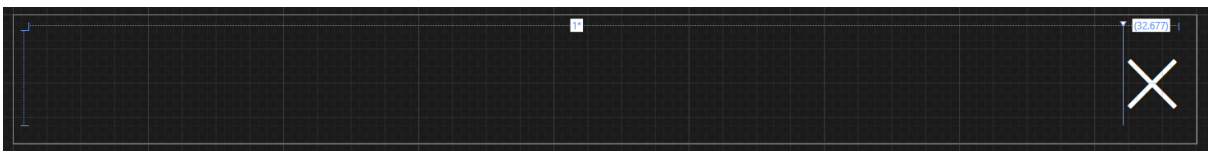

Рисунок 3.10 - Вікно ToastNotificationWindow

3.2.4 опис властивостей та подій компонентної моделі програми

Бізнес-логіка гри розподілена по класах ViewModels:

– DashboardItemViewModel - логіка окремої комірки;

– DashboardViewModel - логіка ігрового поля;

– RatingTableViewModel - логіка груп таблиць рейтингу;

– SavepointsViewModel - логіка групи точок збереження;

– SavepointViewModel - логіка окремої точки збереження;

– SettingsViewModel - логіка групи налаштувань;

– SettingViewModel - логіка окремого налаштування;

– MainViewModel - логіка зосередження об’єктів DashboardViewModel, RatingTableViewModel, SettingsViewModel, SavepointsViewModel.

Кожна ViewModel підв’язується до певного представлення і є контекстом даних цього представлення, також має набір команд, які підв’язується до окремих компонентів цього представлення.

Призначення команд описано в таблицях:

– DashboardViewModel - 3.1;

– RatingTableViewModel - 3.2;

– SavepointsViewModel - 3.3;

– SettingsViewModel - 3.4;

Таблиця 3.1 - Опис команд DashboardViewModel

Назва команди	Тип компонента прив’язки	Параметр	Призначення
ApplyDifficulty	RadioButton	DifficultyLevelsdifficultyLevel	Генерація ігрового поля в залежності від обраної складності, сброс та перезапуск таймера.
AddMinuteAndContinueStopwatch	-	-	Після вимкнення налаштування “Показати правильні відповіді” до секундомера додається хвилина та він запускається.
CleanDashboard	Button	-	Очищення поля від уведених значень.
OpenRules	Button	-	Відкриття діалогового вікна правил.

Продовження таблиці 3.1

Назва команди	Тип компонента прив’язки	Параметр	Призначення
AutoSubmit	TextBox	CellType cellType	Якщо ця опція увімкнена в налаштуваннях, то при кожному введенні значення в полі йде перевірка на те, чи повністю заповнено поле. І якщо так, то виконується перевірка рішення.
GetChangedSettings	-	ObservableCollection<ViewModels.SettingViewModel> settings	Отримання змінених налаштувань.
RestartStopwatch	-	-	Перезапуск секундомера.
SendGameSession	-	-	Відправка даних про ігровий сеанс у разі успішного завершення: обрана складність та час секундомера.
StartStopwatch	-	-	Розпочати відлік секундомера.
StopStopwatch	-	-	Зупинити відлік секундомера.
ValidateSolution	Button	-	Валідація введених користувачем значень та перевірка рішення.
ApplySettingAutoSubmit	-	SettingViewModel autoSubmitSetting	Встановлення режиму для ігрового поля, при якому при кожному введенні викликається команда AutoSubmit.
ApplySettingHideTimer	-	SettingViewModel hideTimerSetting	Встановлення режиму для секундомера, при якому він схований.
ApplySettingShowCorrectAnswers	-	SettingViewModel showCorrectAnswersSetting	Вимкнення режиму AutoSubmit, зупинка таймера, блокування кнопок та показ правильних відповідей.
ShowCorrectAnswers	-	-	Встановлення правильних відповідей в комірках для введення значень.

Таблиця 3.2 - Опис команд RatingTableViewModel

Назва команди	Тип компонента прив’язки	Параметр	Призначення
LoadRatingRecords	DataGrid	-	Підвантаження даних таблиць рейтингу для кожної складності.
SaveRatingRecord	-	GameSession session	Отримання сеансу гри для запису часу секундомера в таблицю рейтинга. При цьому не можна зберігти для одної складності більше, ніж 10 записів.

Таблиця 3.3 - Опис команд SavepointsViewModel

Назва команди	Тип компонента прив’язки	Параметр	Призначення
CleanSavepoints	-	-	Очищення списку точок збереження.
CreateSavepoint	Button	-	Створення точки збереження. Не можна створити більше, ніж 10 точок збереження.
DeleteSavepoint	Button	-	Видалення виділенної точки збереження.
LoadSavepointCommand	Button	-	Завантаження стану ігрового поля із точки збереження.
RewriteSavepoint	Button	-	Перезапис точки збереження.

Таблиця 3.4 - Опис команд SettingsViewModel

Назва команди	Тип компонента прив’язки	Параметр	Призначення
SendSettings	CheckBox	-	Відправлення змінених налаштувань на DashboardViewModel задля встановлення станів елементів. Виконується при будь-якій зміні значень CheckBoxes.
TurnOffAutoSubmit	-	bool autoSubmitEnabled	Якщо були увімкнені правильні відповіді, то автовідправка відповідей вимикається.

Вікно ToastNotificationWindow має обробники подій та користувацькі функції для відображення повідомлень. Опис наведено у таблиці 3.5.

Таблиця 3.5 - Опис обробників подій та користувацьких функцій ToastNotificationWindow

Назва команди	Тип компонента прив’язки	Параметр	Призначення
ToastNotificationWindow_Loaded	Window	object sender, RoutedEventArgs e	Створення та відтворення анімації повідомлення.
ShowAndCloseAfterDelay	-	int millisecondsDelay	Запуск повідомлення на певний час.
CloseNotification()	-	-	Закриття повідомлення із анімацією.
CloseButton_Click	Button	-	Закриття повідомлення.

Задля можливості пересування по комірках фокусу за допомогою клавіш клавіатури використовується обробник події натиснення на клавішу:

```
partial class Elements_Template : ResourceDictionary
{
    public Elements_Template()
    { InitializeComponent();  }
    private void TextBox_KeyDown(object sender, KeyEventArgs e)
    {
        FocusNavigationHelper.HandleWASDKeyDown((UIElement)sender, e);
    }
}
```

3.2.5 детальна декларація визначених функцій програмістом та подій компонентів застосунку

Зв’язки між командами ViewModels наведені на рисунках 3.11-3.14.

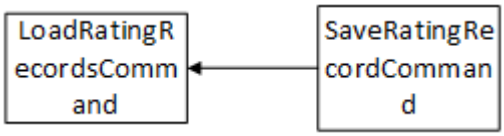

Рисунок 3.11 - Залежності між командами RatingRecordViewModel

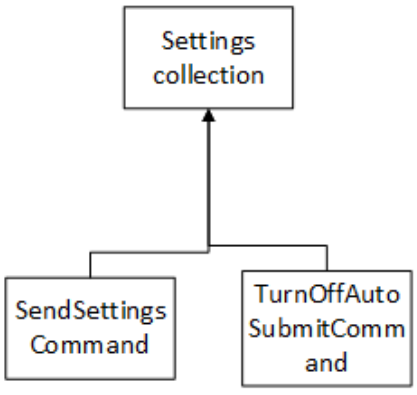

Рисунок 3.12 - Залежності між командами SettingsViewModel

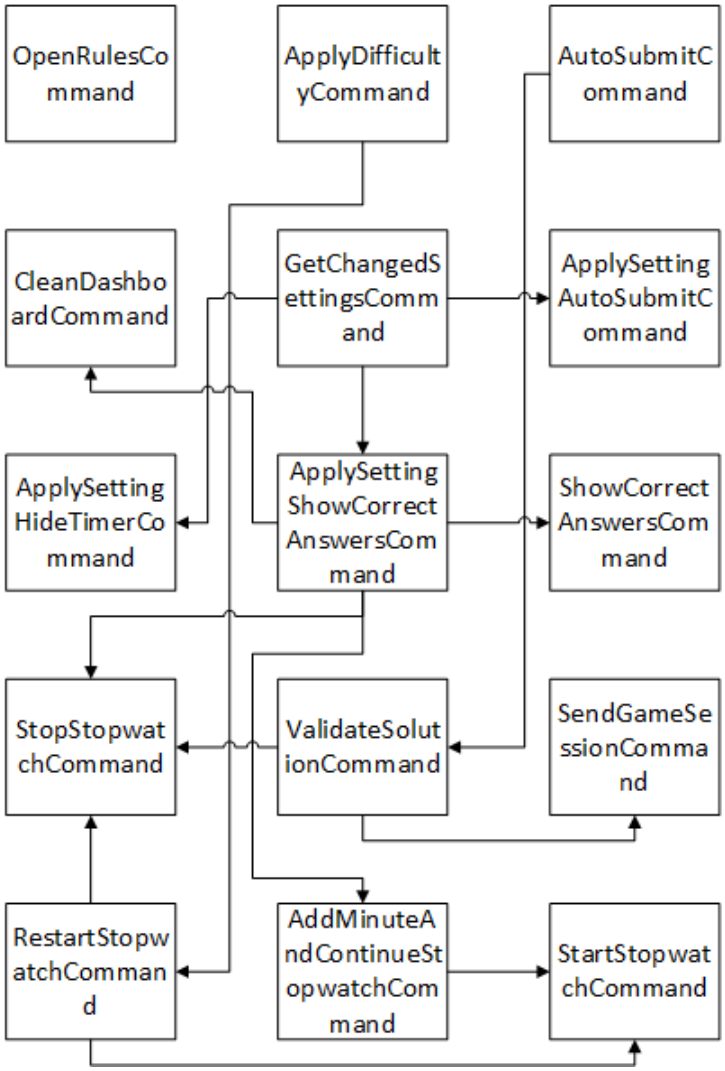

Рисунок 3.13 - Залежності між командами DashboardViewModel

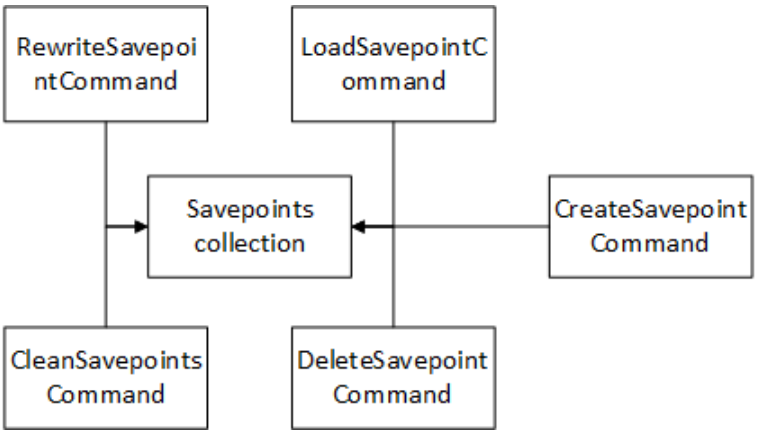

Рисунок 3.14 - Залежності між командами SavepointsViewModel

Лістинг команд, моделей, представлень, ViewModel та інших компонентів програми наведений у Додатку Б.

3.3 Тестування застосунку

3.3.1 мета тестування

Метою тестування є перевірка коректної роботи застосунку згідно із технічним завданням та вимогами користувача. Тестування дозволяє перевірити відповідність функціональних можливостей програми вимогам, відсутність критичних помилок, а також реалізацію всіх необхідних функцій. Особливу увагу було приділено перевірці чотирьох основних компонентів програми: Game, Savepoints, Settings та Rating Tables. Ці секції забезпечують основний функціонал програми та впливають на взаємодію з користувачем.

У процесі тестування кожного з цих компонентів було перевірено правильність виконання ключових операцій. Для секції Game тестування охоплювало коректне введення і обробку даних гравця, відповідність правил гри та точність підрахунку результатів. Секція Savepoints перевірялася на правильне збереження прогресу гравця та можливість відновлення гри з останньої точки збереження. У Settings було протестовано коректність налаштувань і їхнє застосування, а також збереження вибору користувача. У секції Rating Tables перевірялося правильне відображення рейтингових даних, а також коректне оновлення таблиць після завершення ігор.

3.3.2 план тестування

Тестування було проведено вручну шляхом перевірки всіх основних функцій секцій застосунку: Game, Savepoints, Settings, та Rating Tables. Для кожної секції було визначено конкретні сценарії тестування, які охоплюють ключові функціональні можливості.

Секція Game:

– введення коректних та некоректних даних (перевірка наявності чисел від 1 до 9);

– механізм валідації введених даних та розв’язку гри на коректність;

– генерація ігрового поля для різних складностей;

– очищення поля від уведених значень;

– виклик вікна із правилами;

– відлік, зупинка та сброс секундомеру.

Секція Savepoints:

– створення не більше 10 точок збереження;

– завантаження, видалення або перезапис тільки виділеної точки збереження

– можливість надання точці збереження імені;

– очищення списку точок збереження при генерації нового поля.

Секція Settings:

– приховування секундомера;

– відображення правильних відповідей, при цьому блокування інтерфейсу користувача та зупинка секундомера, а після вимкнення правильних відповідей - очищення введених значень та додавання хвилини до секундомера;

– автовідправка на валідацію введених значень, якщо було введене останнє значення.

Секція Rating Tables:

– коректне завантаження даних із файлів;

– створення не більше 10 записів для кожної складності;

– коректна робота кешу застосунку.

3.3.3 тест-кейси

Тести-кейси наведені в таких таблицях відповідно для секцій:

– Game - 3.6;

– Settings - 3.7;

– Savepoints - 3.8;

– Rating Tables - 3.9.

Таблиця 3.6 - Тести-кейси для тестування секції Game гри “Kakuro”

Тест	Опис тесту	Очікуваний результат	Фактичний результат	Методи та засоби виправлення помилки, фрагменти коду

1. Ввід значення в комірку	Перевірка, чи може гравець вводити щось, окрім чисел від 1 до 9	Гравець може вводити тільки числа від 1 до 9	Гравець може вводити будь-які символи	Обмеження довжини вводимого тексту в шаблоні елементу:
				<Setter Property="MaxLength" Value="1" />
				Для унеможливлення вводу будь-чого, окрім цифр, була створена ViewModel, в якій зроблена валідацій вводимого значення:
				public int ConvertStringToInt(string value)
				{
				int enteredValue;
				try
				{
				enteredValue = string.IsNullOrEmpty(value) ? 0 : Convert.ToInt32(value);
				}
				catch (Exception)
				{
				int? previousValue = _dashboardItem.DisplayValue;
				return previousValue.HasValue ? previousValue.Value : 0;
				}
				
				return enteredValue;
				}

2. Генерація ігрового поля	Перевірка, чи коректно генерується поле заданої довжини	Поле генерується коректно і алгоритм не виходить за межі поля	Алгоритм виходить за межі, помилка IndexOutOfRangeException	При проходженні по полю задля поміщення в нього комірок для вводу значень, якщо ми не границі, то ми не встановлюємо її:
				if (!IsElementOnBorder(i, j, dashboardSize) && dashboard[i, j] == "*")
				wrapper.CellType = CellType.ValueCell;
				
				private bool IsElementOnBorder(int i, int j, int dashboardSize) => i == 0 || j == 0 || i == dashboardSize - 1 || j == dashboardSize - 1;

Продовження таблиці 3.6

Тест	Опис тесту	Очікуваний результат	Фактичний результат	Методи та засоби виправлення помилки, фрагменти коду

3. Сброс секундомера	Перевірка, чи здійснюється сброс секундомера при генерації ігрового поля для іншої складності	Сброс секундомера здійснюється при кожній новій генерації поля	Секундомера продовжує відлік навіть після генерації нового поля	Була зроблена команда RestartStopwatchCommand, метод Execute якої:
				public override void Execute(object? parameter)
				{
				_stopStopwatchCommand.Execute(parameter);
				_stopwatch.Reset();
				_startStopwatchCommand.Execute(parameter);
				}

4. Зупинка секундомера	Перевірка, чи здійснюється зупинка секундомера при успішному проходженні гри	Зупинка секундомера здійснюється при успішному проходженні гри	Секундомер продовжує відлік навіть після закінчення гри	В команду ValidateSolutionCommand в метод Execute було додано виклик зупинки секундомера:
				if (isSolutionCorrect)
				{
				//another code
				_stopStopwatchCommand.Execute(parameter);
				//another code
				}

5. Блокування інтерфейсу	Перевірка, чи блокується інтерфейс після успішного проходження гри	Після успішного проходження гри кнопки та поле блокується	Після успішного проходження гри кнопки та поле залишаються активними	Додав стан у булеву змінну: ShowCorrectAnswers, яка за допомогою конвертора керує станом IsEnabled елементів: IsEnabled="{Binding ShowCorrectAnswers, Converter={StaticResource InverseBooleanConverter}}"

6. Скидання додатково доданого до секундомеру часу	Перевірка, чи при генерації нового поля скидається час, доданий до секундомера в результаті показу правильних відповідей	При генерації нового поля скидається час, доданий до секундомера в результаті показу правильних відповідей	Доданий час не скидується, тому відлік починається з певної хвилини	У користувацький клас секундомера MyStopwatch було додано метод для очищення доданого часу:
				public void CleanAdditionalTime()
				{
				StartOffset = new TimeSpan();
				}

Таблиця 3.7 - Тести-кейси для тестування секції Settings гри “Kakuro”

Тест	Опис тесту	Очікуваний результат	Фактичний результат	Методи та засоби виправлення помилки, фрагменти коду

1. Вимкнення налантування автовідправки при увімкненні правильних відповідей	Перевірка, чи вимикається автовідправка при увімкненні правильних відповідей	Автовідправка вимикається при увімкненні правильних відповідей	Автовідправка не вимикається, правильні відповіді надсилаються і гра завершується	При прийнятті полем правильних відповідей викликається функція, яка вимикає стан автовідправки у секції поля та посилає сигнал на вимкнення його на стороні налаштувань:
				private void TurnOffAutoSubmit(bool showCorrectAnswers)
				{
				if (showCorrectAnswers)
				_dashboardViewModel.AutoSubmit = false;
				_eventAggregator.GetEvent<CorrectAnswersTurnedOnEvent>().Publish(_dashboardViewModel.AutoSubmit);
				}

2. Додавання додаткового часу до секундомера	Перевірка, чи в результаті показу правильних відповідей відбувається додавання хвилин до секундомера	При показі правильних відповідей до часу секундомера додається хвилина	До секундомера неможливо додати час	Було розроблено користувацький клас-нащадок секундомера, який дозволяє додавати час:
				public class MyStopwatch : Stopwatch
				{
				public TimeSpan StartOffset { get; private set; }
				public MyStopwatch(TimeSpan startOffset)
				{
				StartOffset = startOffset;
				}
				public TimeSpan TotalElapsed => Elapsed + StartOffset;
				public int ElapsedHours => TotalElapsed.Hours;
				public int ElapsedMinutes => TotalElapsed.Minutes;
				
				public int ElapsedSeconds => TotalElapsed.Seconds;
				
				public void AddTime(TimeSpan timeToAdd)
				{
				StartOffset += timeToAdd;
				}
				
				public void CleanAdditionalTime()
				{
				StartOffset = new TimeSpan();
				}
				}

Продовження таблиці 3.7

Тест	Опис тесту	Очікуваний результат	Фактичний результат	Методи та засоби виправлення помилки, фрагменти коду

3. Очищення поля після вимкнення правильних відповідей	Перевірка, чи здійснюється очищення поля після вимкнення правильних відповідей	Поле очищується після вимкнення правильних відповідей	Правильні відповіді залишаються навіть після їх вимкнення.	Було додано виклик команди для очищення поля в разі вимкнення правильних відповідей:
				private void ShowOrEraseCorrectAnswers(bool showCorrectAnswers)
				{
				if (showCorrectAnswers)
				_showCorrectAnswersCommand.Execute(null);
				else
				_cleanDashboardCommand.Execute(null);
				}

3.3.4 результати тестування

Результати тестування після виправлення помилок наведені в таких таблицях відповідно для секцій:

– Rating Tables - 3.10;

– Game - 3.11;

– Settings - 3.12;

– Savepoints - 3.13.

Таблиця 3.8 - Тести-кейси для тестування секції Savepoints гри “Kakuro”

Тест	Опис тесту	Очікуваний результат	Фактичний результат	Методи та засоби виправлення помилки, фрагменти коду

1. Очищення списку точок збереження	Перевірка, чи здійснюється очистка списку точок збереження при генерації нового поля	При генерації нового поля здійснюється очистка списку точок збереження	При генерації нового поля точки збереження залишаються, в результаті чого при їх завантаженні здійздійнюються помилки	Додав команду очистки списку точок збереження CleanSavepointsCommand, метод Execute якої:
				public override void Execute(object? parameter)
				{
				for (int i = 0; i < _savepointsViewModel.Savepoints.Count; i++)
				_savepointProvider.Delete(_savepointsViewModel.Savepoints[i].Id);
				
				_savepointsViewModel.Savepoints.Clear();
				}

Продовження таблиці 3.8

Тест	Опис тесту	Очікуваний результат	Фактичний результат	Методи та засоби виправлення помилки, фрагменти коду

2. Можливість команд завантаження, видалення та перезапису точок збереження	Перевірка, чи активні кнопки (і відповідно виконання команд), коли немає і не виділено жодної точки збереження	Кнопки для виконання операцій неактивні, доки не буде виділена точка збереження у списку.	Кнопки (і відповідно команди) активні, навіть коли немає точок збереження.	Для усіх трьох операцій було додано відслідковування, чи виділено якусь точку збереження:
				В конструкторі:
				_savepointsViewModel.PropertyChanged += OnSelectedSavepointPropertyChanged;
				
				public override bool CanExecute(object? parameter)
				{
				return _savepointsViewModel.SelectedSavepoint != null && base.CanExecute(parameter);
				}
				
				private void OnSelectedSavepointPropertyChanged(object? sender, PropertyChangedEventArgs e)
				{
				if (e.PropertyName == nameof(_savepointsViewModel.SelectedSavepoint))
				OnCanExecutedChanged();
				}

3. Створення точок збереження	Перевірка на можливість створювати не більше 10 точок збереження.	Можна створювати не більше 10 точок збереження.	Кнопка для створення точок збереження активна навіть при наявності вже 10.	В класі команди було встановлене обмеження:
				public override bool CanExecute(object? parameter)
				{
				return _savepointsViewModel.Savepoints.Count < 10 && base.CanExecute(parameter);
				}
				
				private void OnSavepointsCollectionPropertyChanged(object? sender, PropertyChangedEventArgs e)
				{
				if (e.PropertyName == nameof(_savepointsViewModel.Savepoints.Count))
				OnCanExecutedChanged();
				}

Продовження таблиці 3.8

Тест	Опис тесту	Очікуваний результат	Фактичний результат	Методи та засоби виправлення помилки, фрагменти коду

4. Створення копії поля при створенні точки збереження	Перевірка, чи дійсно створюється його копія при створенні нової точки збереження	При створенні нової точки збереження створюється копія поля.	При створенні нової точки збереження створюється копія посилання на поле.	До DashboardViewModel був доданий метод для створення копії:
				public DashboardItemCollection CreateDashboardCopy()
				{
				var newCollection = new DashboardItemCollection();
				foreach (var innerCollection in Dashboard)
				{
				var newInnerCollection = new ObservableCollection<DashboardItemViewModel>();
				foreach (var item in innerCollection)
				{
				var newItem = new DashboardItemViewModel(new DashboardItem
				{
				DisplayValue = item.ConvertStringToInt(item.DisplayValue),
				HiddenValue = item.ConvertStringToInt(item.HiddenValue),
				SumRight = item.ConvertStringToInt(item.SumRight),
				SumBottom = item.ConvertStringToInt(item.SumBottom),
				CellType = item.CellType
				});
				newInnerCollection.Add(newItem);
				}
				newCollection.Add(newInnerCollection);
				}
				return newCollection;
				}

Таблиця 3.10 - Результати тестування секції Rating Tables гри “Kakuro”

Тест	Опис тесту	Очікуваний результат	Фактичний результат	Статус
1. Завантаження записів з файлу	Перевірка, чи коректно завантажуються усі записи з файлів	Усі записи завантажуються коректно	Усі записи завантажуються коректно	Пройдено
2. Оновлення таблиць рейтингу після створення нового запису	Перевірка, чи здійснюється після збереження нового запису оновлення списків записів у програмі - відповідно у UI.	Оновлення списків записів таблиць рейтингу здійснюється після створення нового запису	Оновлення списків записів таблиць рейтингу здійснюється після створення нового запису	Пройдено

Таблиця 3.9 - Тести-кейси для тестування секції Rating Tables гри “Kakuro”

Тест	Опис тесту	Очікуваний результат	Фактичний результат	Методи та засоби виправлення помилки, фрагменти коду
1. Завантаження записів з файлу	Перевірка, чи коректно завантажуються усі записи з файлів	Усі записи завантажуються коректно	Записи не завантажуються з файлу	Була така операція конвертації після завантаження записів з файлів: private ObservableCollection<T> ConvertIEnumerableToObservable<T>(IEnumerable<T> values) { return values as ObservableCollection<T>; } Вона виявилась нерозумною, адже IEnumerable не є нащадком ObservableCollection, тому завжди повертало null. Функція була змінена на: private ObservableCollection<T> ConvertIEnumerableToObservable<T>(IEnumerable<T> values) => new ObservableCollection<T>(values); В результаті чого на основі колекції IEnumerable створюється ObservableCollection.

2. Оновлення таблиць рейтингу після створення нового запису	Перевірка, чи здійснюється після збереження нового запису оновлення списків записів у програмі - відповідно у UI.	Оновлення списків записів таблиць рейтингу здійснюється після створення нового запису	Після створення нового запису таблиці рейтингу не оновлюються.	В кінець виконання команди створення запису було додано виклик команди завантаження даних за допомогою EventHandler:
				public class SaveRatingRecordCommand : RelayCommand
				{
				private IRatingRecordProvider _ratingRecordProvider;
				
				public event EventHandler? SaveCompleted;
				
				//конструктор...
				public override void Execute(object? parameter)
				{
				//основна логіка
				
				OnSaveCompleted();
				}
				
				protected virtual void OnSaveCompleted()
				{
				SaveCompleted?.Invoke(this, EventArgs.Empty);
				}
				}

Таблиця 3.11 - Результати тестування секції Game гри “Kakuro”

Тест	Опис тесту	Очікуваний результат	Фактичний результат	Статус
1. Ввід значення в комірку	Перевірка, чи може гравець вводити щось, окрім чисел від 1 до 9	Гравець може вводити тільки числа від 1 до 9	Гравець може вводити тільки числа від 1 до 9	Пройдено
2. Генерація ігрового поля	Перевірка, чи коректно генерується поле заданої довжини	Поле генерується коректно і алгоритм не виходить за межі поля	Поле генерується коректно і алгоритм не виходить за межі поля	Пройдено
3. Скидання секундомера	Перевірка, чи здійснюється скидання секундомера при генерації ігрового поля для іншої складності	Скидання секундомера здійснюється при кожній новій генерації поля	Скидання секундомера здійснюється при кожній новій генерації поля	Пройдено
4. Зупинка секундомера	Перевірка, чи здійснюється зупинка секундомера при успішному проходженні гри	Зупинка секундомера здійснюється при успішному проходженні гри	Зупинка секундомера здійснюється при успішному проходженні гри	Пройдено
5. Блокування інтерфейсу	Перевірка, чи блокується інтерфейс після успішного проходження гри	Після успішного проходження гри кнопки та поле блокуються	Після успішного проходження гри кнопки та поле блокуються	Пройдено
6. Скидання додаткового часу до секундомера	Перевірка, чи при генерації нового поля скидається додатковий час на секундомері	При генерації нового поля скидається час, доданий до секундомера	При генерації нового поля скидається час, доданий до секундомера	Пройдено

Таблиця 3.12 - Результати тестування секції Settings гри “Kakuro”

Тест	Опис тесту	Очікуваний результат	Фактичний результат	Статус
1. Вимкнення налантування автовідправки при увімкненні правильних відповідей	Перевірка, чи вимикається автовідправка при увімкненні правильних відповідей	Автовідправка вимикається при увімкненні правильних відповідей	Автовідправка вимикається при увімкненні правильних відповідей	Пройдено
2. Додавання додаткового часу до секундомера	Перевірка, чи в результаті показу правильних відповідей відбувається додавання хвилин до секундомера	При показі правильних відповідей до часу секундомера додається хвилина	При показі правильних відповідей до часу секундомера додається хвилина	Пройдено
3. Очищення поля після вимкнення правильних відповідей	Перевірка, чи здійснюється очищення поля після вимкнення правильних відповідей	Поле очищується після вимкнення правильних відповідей	Поле очищується після вимкнення правильних відповідей	Пройдено

Таблиця 3.13 - Результати тестування секції Savepoints гри “Kakuro”

Тест	Опис тесту	Очікуваний результат	Фактичний результат	Статус
1. Можливість команд завантаження, видалення та перезапису точок збереження	Перевірка, чи активні кнопки (і відповідно виконання команд), коли немає і не виділено жодної точки збереження	Кнопки для виконання операцій неактивні, доки не буде виділена точка збереження у списку.	Кнопки для виконання операцій неактивні, доки не буде виділена точка збереження у списку.	Пройдено
2. Створення точок збереження	Перевірка на можливість створювати не більше 10 точок збереження.	Можна створювати не більше 10 точок збереження.	Можна створювати не більше 10 точок збереження.	Пройдено
3. Створення копії поля при створенні точки збереження	Перевірка, чи дійсно створюється його копія при створенні нової точки збереження	При створенні нової точки збереження створюється копія поля.	При створенні нової точки збереження створюється копія поля.	Пройдено
4. Очищення списку точок збереження	Перевірка, чи здійснюється очистка списку точок збереження при генерації нового поля	При генерації нового поля здійснюється очистка списку точок збереження	При генерації нового поля здійснюється очистка списку точок збереження	Пройдено

3.3.5 висновки

За результатами проведеного тестування встановлено, що застосунок загалом відповідає технічному завданню та вимогам користувача. Основні функціональні компоненти—Game, Savepoints, Settings та Rating Tables—працюють коректно, забезпечуючи очікувану взаємодію з користувачем та виконання всіх необхідних функцій.

Виявлені на попередніх етапах тестування помилки були виправлені, і зараз програма демонструє стабільну роботу без критичних збоїв. Таким чином, застосунок оцінюється як готовий до використання і здатний задовольнити потреби користувачів у повному обсязі.

3.4 Інструкція користувача

Запуск програми виконується за допомогою файлу Kakuro.exe. Для збереження та читання даних необхідний зовнішній файл з формованими даними або порожній. При запуску програми на екрані з'являється головне вікно програми, яке наведено на рис. 3.15.

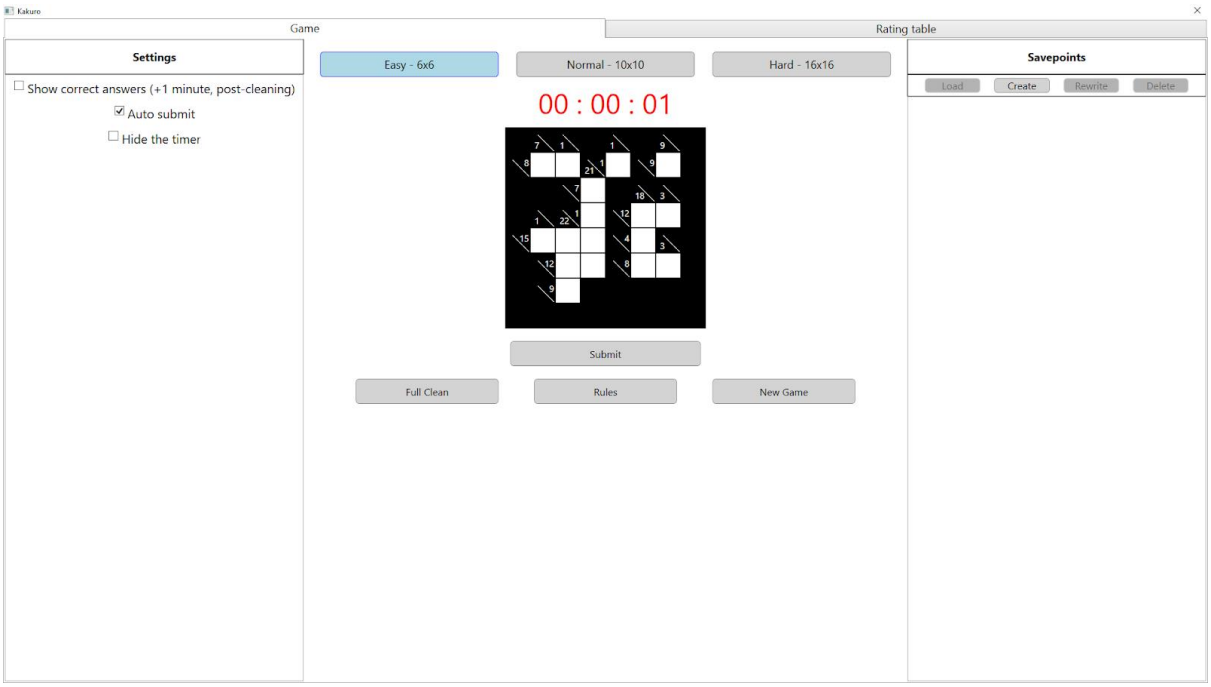

Рисунок 3.15 - Головне вікно

Програма складається з двох вкладок: Game і Rating table.

У вкладці Rating table (див. рис. 3.16) здійснюється відображення відсортованих за часом результатів успішно пройдених ігор у минулому.

Для кожного рівня складності може буде не більше 10 записів у таблиці рейтингу. Завантаження записів виконується автоматично після запуску програми. Таблиці рейтингу зберігають час і дату проходження гри.

Секція налаштувань із вкладки Game (див. рис. 3.17) містить налаштування: "Показати правильні відповіді", "Автовідправка", "Сховати секундомер".

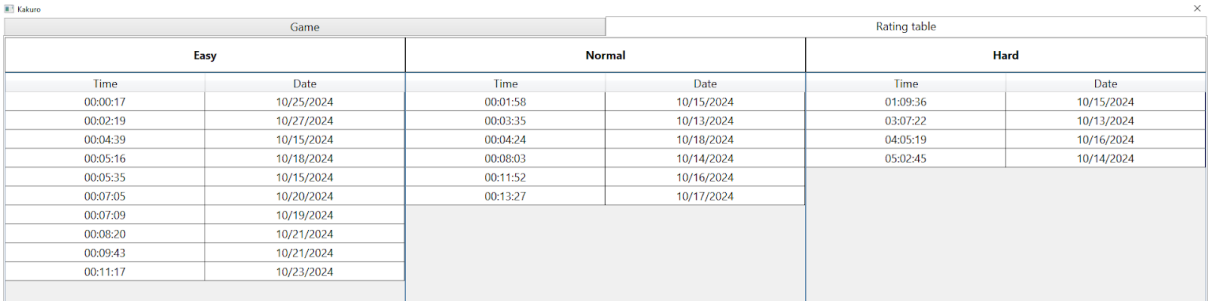

Рисунок 3.16 - Таблиці рейтингу

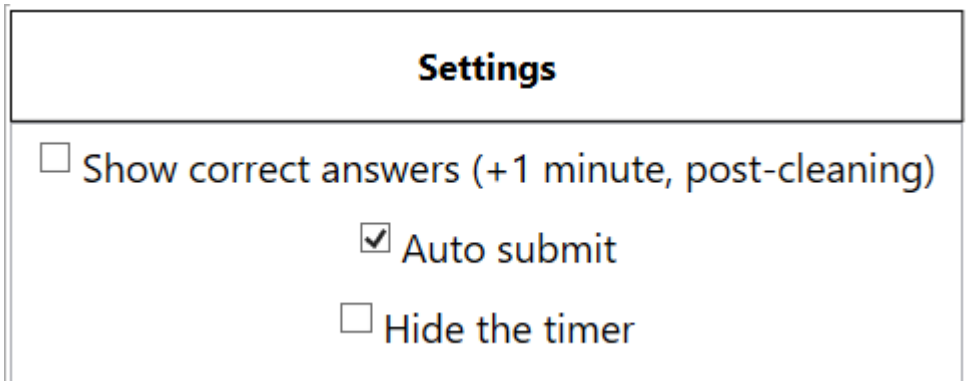

Рисунок 3.17 - Налаштування

Налаштування “Автовідправка” увімкнено за замовчуванням з початку гри. При цьому налаштуванні здійснюється автовідправка введених користувачем значень та розв’язку на їх валідацію, якщо було заповнено останню комірку для вводу значень.

При увімкненому налаштуванні “Показати правильні відповіді” у комірки виводяться сховані значення, які були згенеровані при генерації ігрового поля та на основі яких рахувались суми-підказки, також блокується секундомер та інтерфейс застосунку, а налаштування “Автовідправка” вимикається (див. рис. 3.18). Після вимкнення налаштування інтерфейс знову стає доступним, відлік секундомера продовжується, проте до його часу додається хвилина, а ігрове поле повністю очищується (див. рис. 3.19).

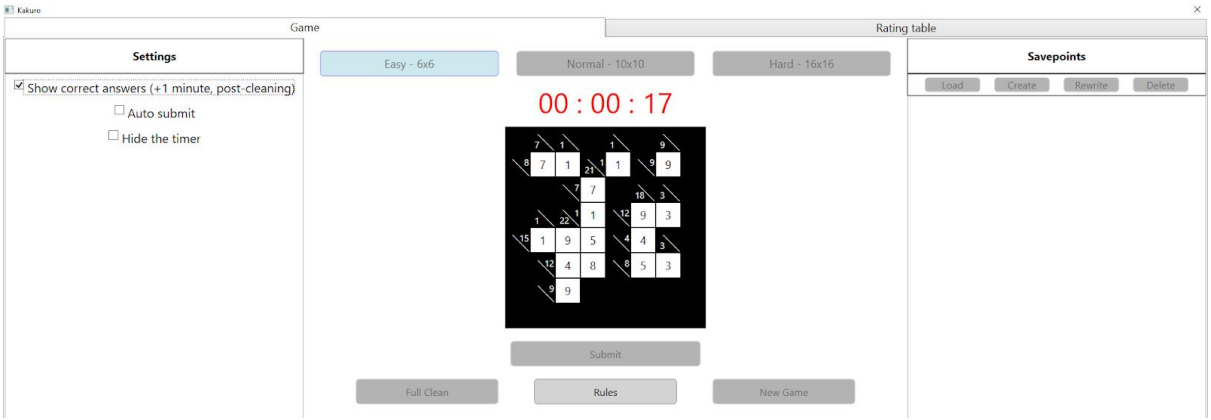

Рисунок 3.18 - Налаштування “Показати правильні відповіді” увімкнено

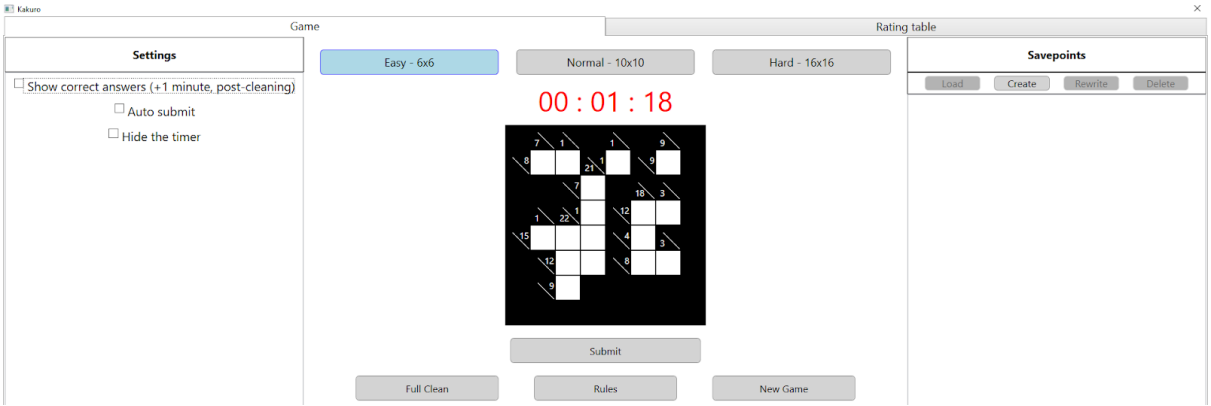

Рисунок 3.19 - Налаштування “Показати правильні відповіді” було вимкнено

При увімкненому налаштуванні “Сховати секундомер” секундомер ховається, проте не зупиняється (див. рис. 3.20).

У секції безпосередньо самої гри (див. рис. 3.21) користувач може перемикатись між рівнями складності: Easy, Normal, Hard (див. рис. 3.22-3.23) - в залежності від рівня складності генерується поле 6x6, 10x10 або 16x16. При генерації ігрового поля значення секундомера онулюється.

За допомогою кнопки “Rules” викликається модульне вікно, в якому наведені правила та обмеження гри (див. рис. 3.24).

За допомогою кнопки “Full Clean” користувач може очистити ігрове поле від уведених значень. За допомогою кнопки “New Game” - згенерувати нове ігрове

поле відповідно до обраної складності. За допомогою кнопки “Submit” користувач може відправити розв’язок гри на валідацію.

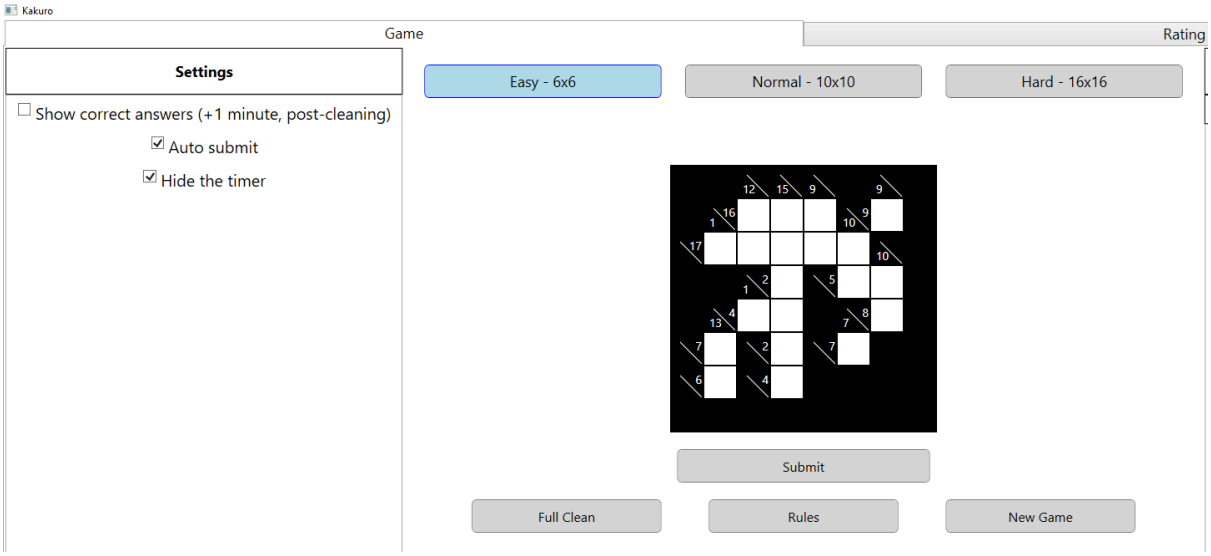

Рисунок 3.20 - Налаштування “Сховати секундомер” увімкнено

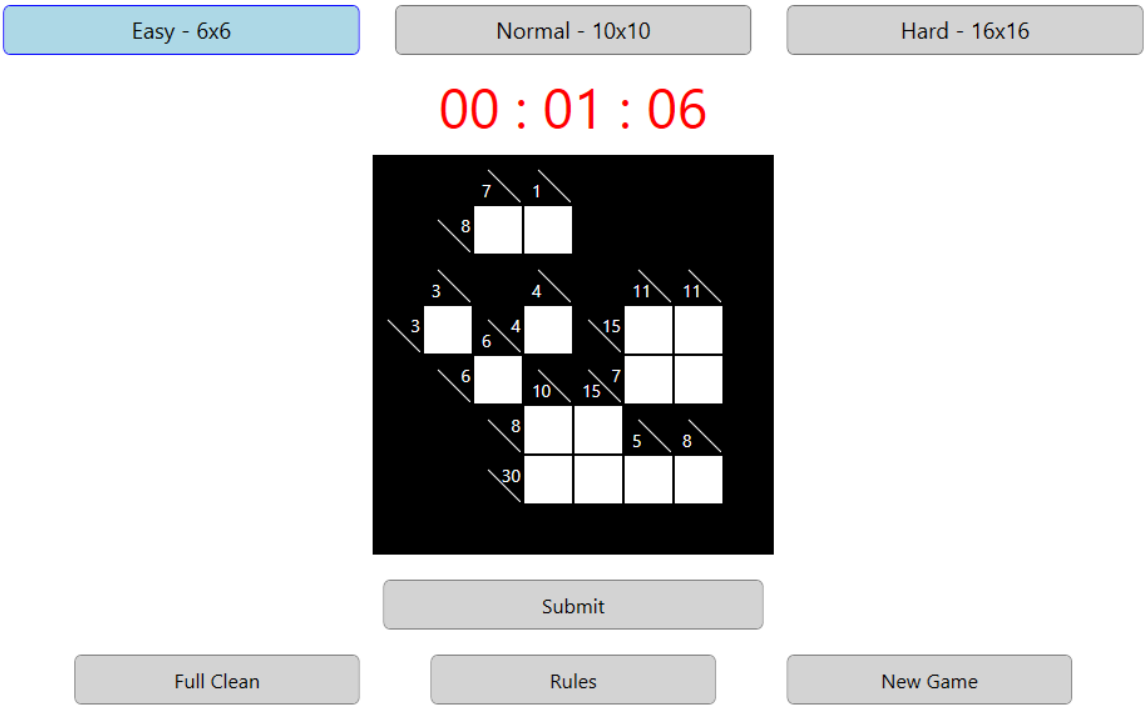

Рисунок 3.21 - Секція гри. Обраний рівень складності - Easy

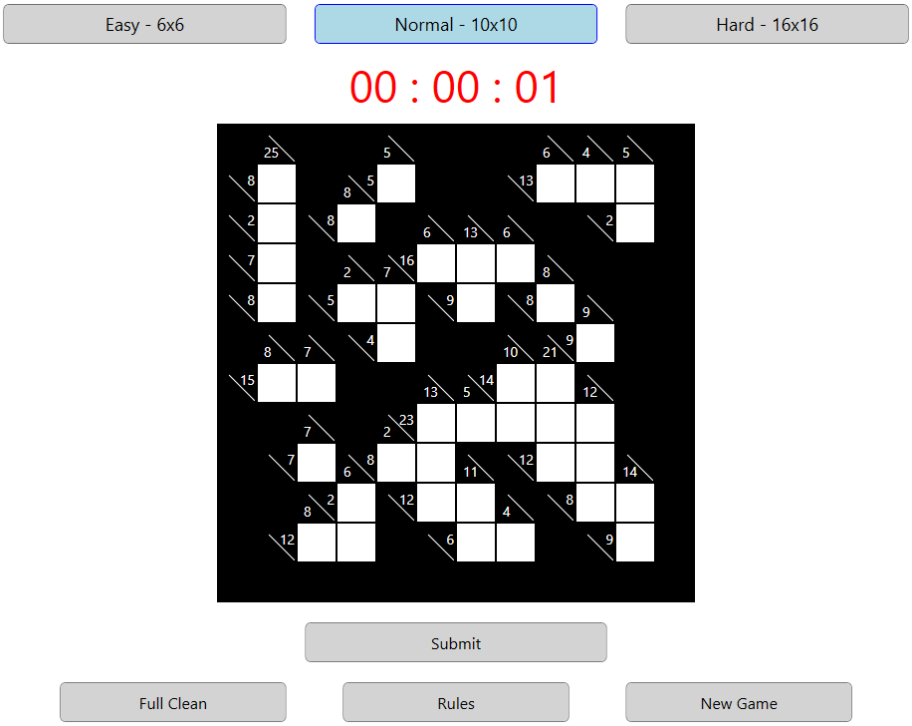

Рисунок 3.22 - Обраний рівень складності - Normal

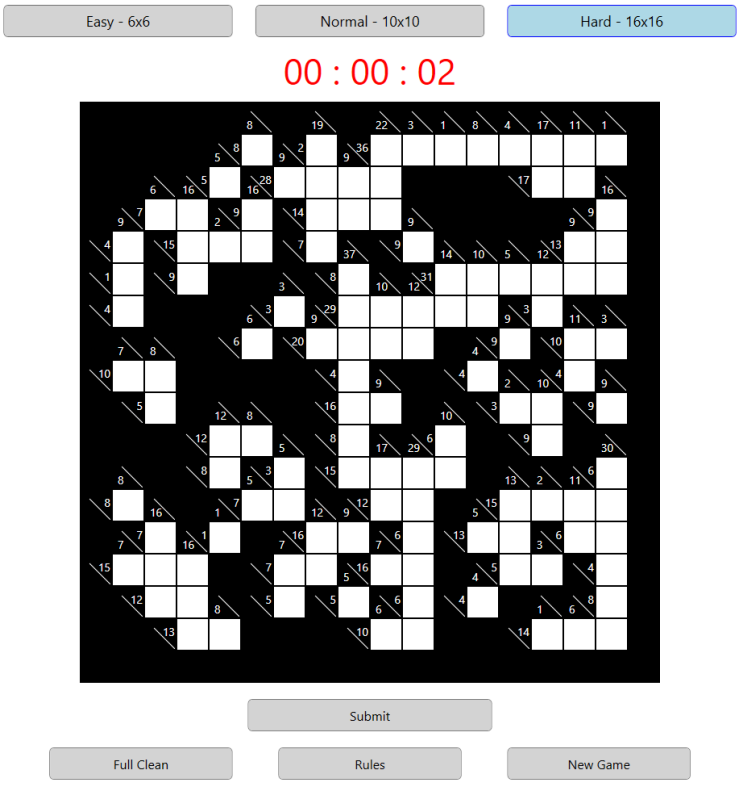

Рисунок 3.23 - Обраний рівень складності - Hard

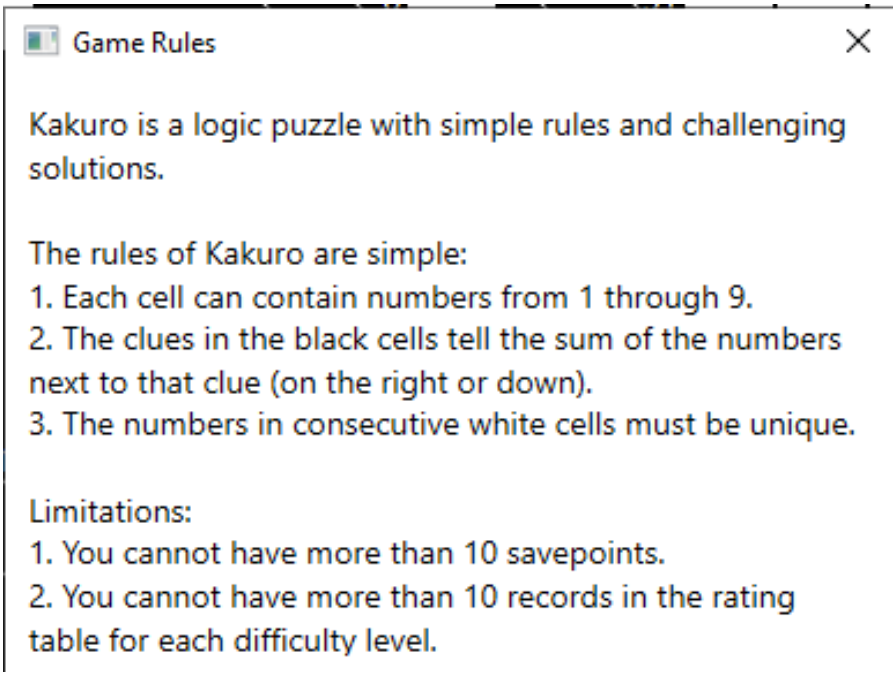

Рисунок 3.24 - Модульне вікно правил та обмежень гри

Повідомлення спливають плавно знизу екрану і через 10 секунд ховаються. Якщо якась клітинка залишилась порожньою, виводиться повідомлення про помилку “Dashboard isn't filled completely!” (див. рис. 3.25), якщо є суміжні однакові значення, виводиться повідомлення про помилку “Entered values aren't unique!” (див. рис. 3.26), якщо сума значень в якомусь стовпці або рядку не відповідає заданій сумі у комірці суми-підказки, то виводиться повідомлення про помилку “The sum of the numbers is not equal to the given sum!” (див. рис. 3.27). Якщо всі попередні умови виконані, то розв’язок проходить валідацію і виводиться повідомлення про успіх "Game is completed!" (див. рис. 3.28).

Рисунок 3.25 - Повідомлення про помилку, пов’язану з порожніми клітинками

Рисунок 3.26 - Повідомлення про помилку про суміжні однакові значення

Рисунок 3.27 - Повідомлення про помилку про невідповідність фактичних сум очікуваним

Рисунок 3.28 - Повідомлення про успіх завершення гри

У секції Savepoints (див. рис. 3.29-3.30) користувач може створювати, завантажувати, перезаписувати або видаляти точки збереження стану ігрового поля. При генерації нового ігрового поля список точок збереження онулюється. Користувач може створити не більше 10 точок збереження, а видалити, завантажити або перезаписати тільки виділену точку збереження у списку. Точки збереження створюються та використовуються безпосередньо під час гри та не зберігаються у файлах після закриття застосунку. Проте збереження в файли здійснюється під час гри, щоб не зберігати у безпосередньо у ОП програми купу станів ігрового поля, задля чого було створено механізм кешу, при якому в ОП програми зберігаються тільки найактуальніші точки збереження, а графічно відображається список усіх точок збереження.

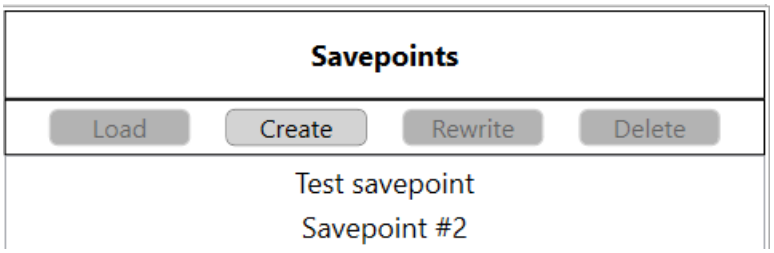

Рисунок 3.29 - Секція точок збереження

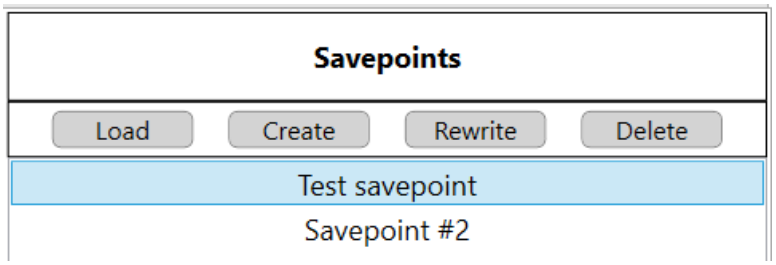

Рисунок 3.30 - Секція точок збереження, коли у списку виділена одна

При створенні точки збереження користувач має можливість надати ім’я (див. рис. 3.31).

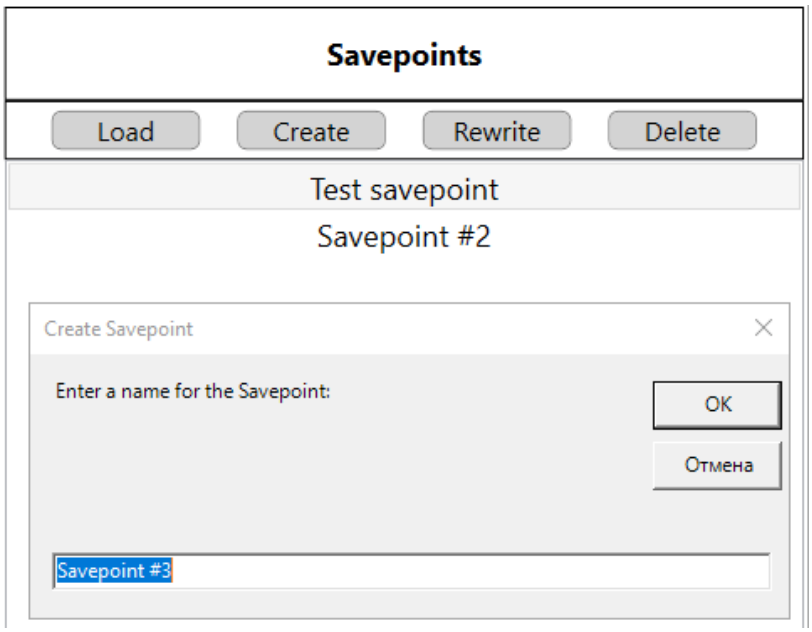

Рисунок 3.31 - Діалогове вікно надання імені точці збереження при створенні

При наявності 10 точок збереження кнопка “Create” блокується (див. рис. 3.32).

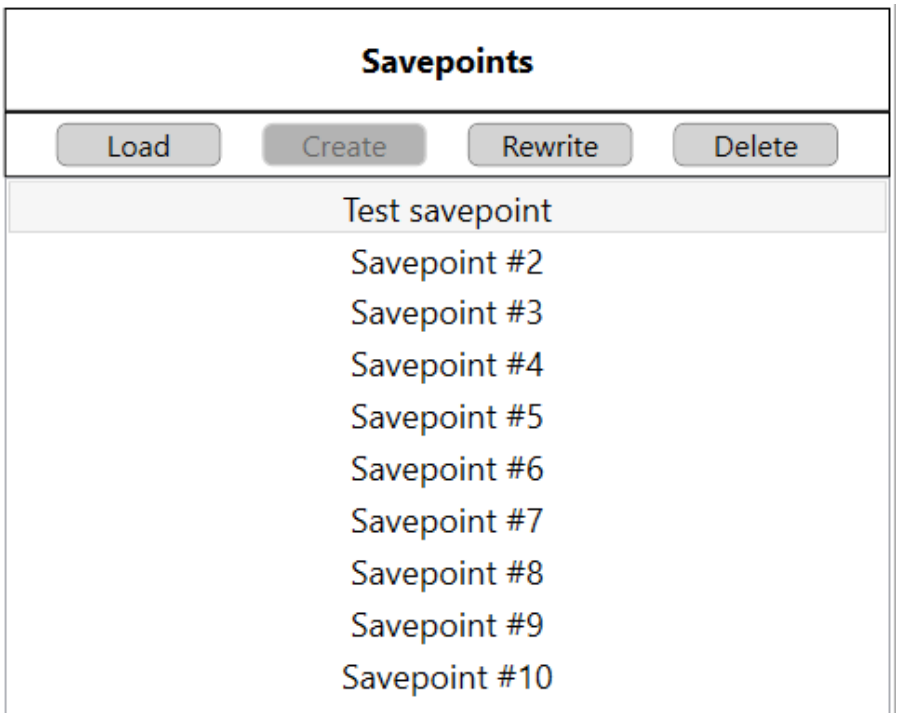

Рисунок 3.32 - Блокування можливості створення точок збереження

На рисунках 3.33-3.34 наведені рисунки файлів для таблиць рейтингу.

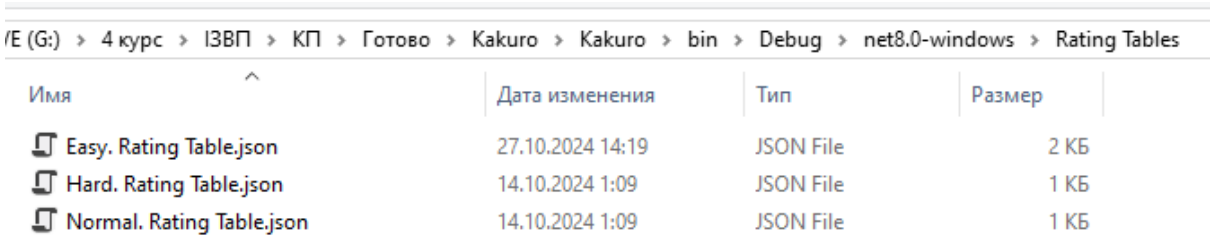

Рисунок 3.33 - Збереження таблиць рейтингу здійснюється у різні файли відповідно до складності

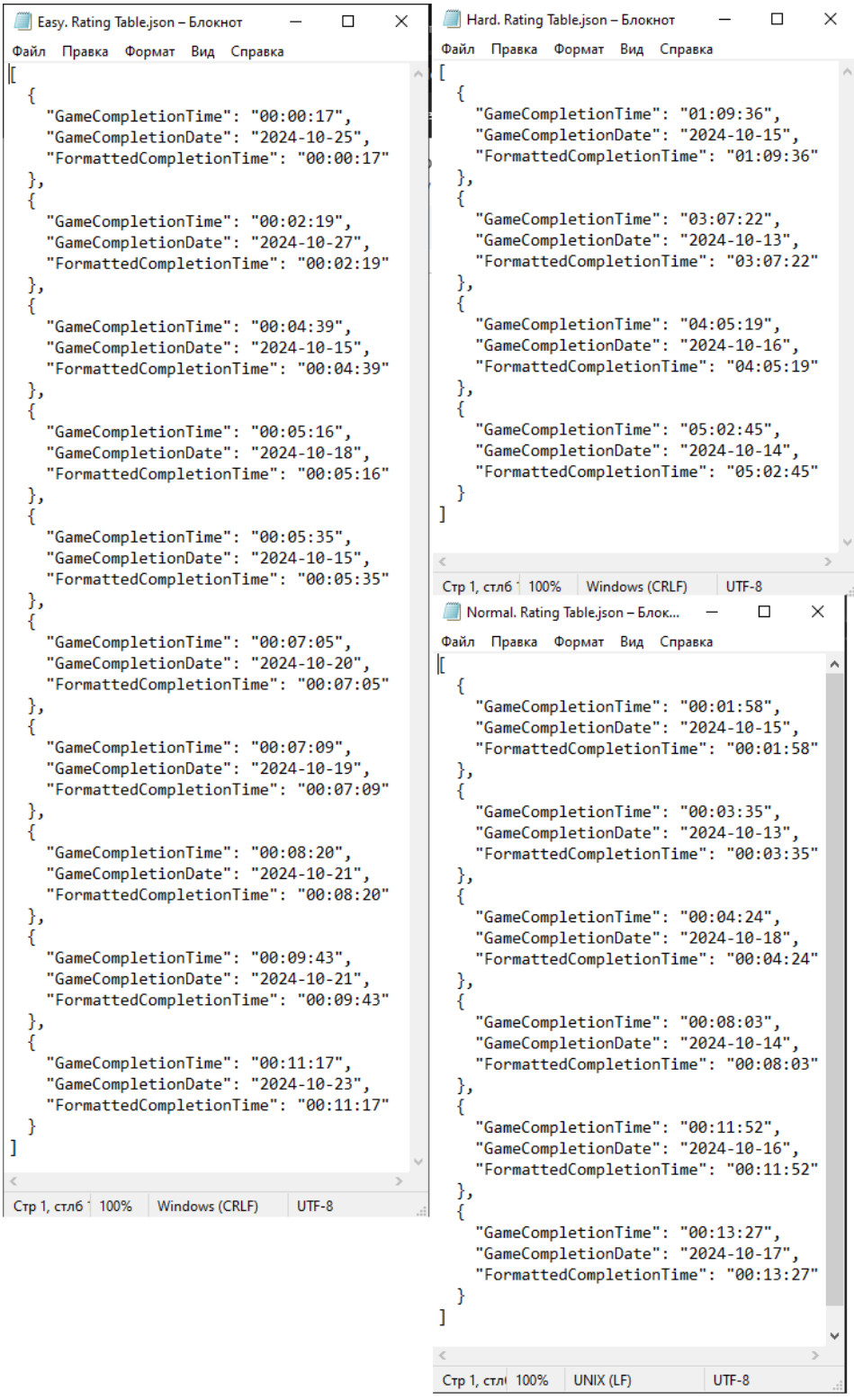

Рисунок 3.34 - Збережені дані таблиць рейтингу

На рисунку 3.35 наведено рисунок даних тимчасового файлу точок збереження.

Рисунок 3.35 - Збережені дані точок збереження

3.5 Результати реалізації застосунку

У розробленому застосунку "Kakuro" реалізовано механіку гри з можливістю генерування ігрових полів та перевіркою заповнення клітинок. Користувачі можуть обирати рівні складності: Easy, Normal та Hard. Прогрес можливо зберегти за допомогою точок зберігання, а також створюються таблиці рейтингів для кожного рівня складності.

Є кілька недоліків, які потребують уваги. Впровадження спеціальних програмних сервісів для зберігання кешу дозволило б автоматично зберігати дані у разі несподіваного вимкнення системи або закриття програми. Важливо обмежити можливість користувача редагувати збережені файли через файловий провідник, щоб уникнути потенційних помилок.

Необхідно реалізувати захист від видалення даних безпосередньо в програмі, наприклад, через діалогові вікна, які запитують дозвіл користувача на видалення. Завантаження та збереження даних повинні бути асинхронними, щоб під час збереження не дозволяти користувачеві вносити зміни. Якщо користувач вручну змінює файли, потрібно відображати повідомлення про неможливість завантаження даних.

Додати реєстратор подій для відстеження дій користувача. Забезпечити підтримку української та англійської мов, що підвищить зручність використання застосунку. Впровадження цих вдосконалень значно підвищить надійність і зручність використання програми.

ВИСНОВКИ ПО ПРОЄКТУ

У даному курсовому проєкті було розроблено програмний застосунок "Kakuro", який реалізує логічну гру-головоломку відповідно до поставленої задачі. Застосунок забезпечує управління процесом гри, перевірку правил заповнення ігрового поля та збереження результатів. Також були створені таблиці рейтингу для рівнів складності Easy, Normal та Hard, а також реалізовано відповідну генерацію ігрових полів, що дозволяє користувачам обирати складність гри. Впроваджена система точок збереження забезпечує збереження прогресу під час гри та можливість повернення до попередніх станів.

У процесі виконання проєкту були вдосконалені знання з візуального програмування для WPF та патернів архітектури. Реалізація патерну MVVM дозволила розділити логіку програми та інтерфейс, спростивши обслуговування коду. Використання Dependency Injection полегшило взаємодію між компонентами, а Event Aggregator підвищив модульність коду, зменшуючи його зв'язність. Патерн Репозиторію організував роботу з файловою системою у форматі JSON, спростивши читання та запис даних. Також реалізована система кешу підвищила продуктивність застосунку, зберігаючи часто використовувані дані для швидшого доступу.

ПЕРЕЛІК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Albahari J. C# 12 in a Nutshell: The Definitive Reference : навчальний посібник / ed. by B. Guerin et al. ; illus. K. Dullea. California : O'Reilly Media, Incorporated, 2023. 1083 p.
2. Stonis M. Enterprise Application Patterns using .NET MAUI : навчальний посібник. 2nd ed. Washington : Microsoft Corporation, 2022. 106 p.
3. Hanselman S. Visual Studio IDE documentation. Microsoft Learn. URL: https://learn.microsoft.com/en-us/visualstudio/ide/?view=vs-2022 (date of access: 24.09.2024).

Додаток А

Лістинг програми

////////////////////// GlobalUsings.cs//////////////////////

```
global using DashboardItemCollection = Sys-
tem.Collections.ObjectModel.ObservableCollection<System.Collections.ObjectModel.Observabl
eCollection<Kakuro.ViewModels.DashboardItemViewModel>>;
global using RatingTableContainer = Sys-
tem.Collections.Generic.Dictionary<Kakuro.Enums.DifficultyLevels, Sys-
tem.Collections.ObjectModel.ObservableCollection<Kakuro.Models.RatingRecord>>;
```

////////////////////// App.xaml//////////////////////

```
<Application x:Class="Kakuro.App"
             xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             xmlns:converters="clr-namespace:Kakuro.Converters">
    <Application.Resources>
        <ResourceDictionary>
            <ResourceDictionary.MergedDictionaries>
                <ResourceDictionary Source="Styles/Button.xaml"/>
                <ResourceDictionary Source="Styles/RadioButton.xaml"/>
                <ResourceDictionary Source="Styles/TabItem.xaml"/>
                <ResourceDictionary Source="Styles/TextBoxStyle.xaml"/>
                <ResourceDictionary Source="Styles/Elements_Template.xaml"/>
                <ResourceDictionary Source="Styles/Rows_Template.xaml"/>
                <ResourceDictionary Source="Styles/RatingTableStyles.xaml"/>
                <ResourceDictionary Source="Styles/RatingDataGridTemplate.xaml"/>
                <ResourceDictionary Source="Styles/TitleStyles.xaml"/>
                <ResourceDictionary Source="Styles/SettingsStyles.xaml"/>
                <ResourceDictionary Source="Styles/NotificationStyles.xaml"/>
                <ResourceDictionary Source="Styles/SumCell_Style.xaml"/>
            </ResourceDictionary.MergedDictionaries>

            <converters:MultiOrBooleanConverter x:Key="MultiOrBooleanConverter" />
            <converters:InverseBooleanConverter x:Key="InverseBooleanConverter" />
            <converters:BooleanToVisibilityConverter
x:Key="BooleanToVisibilityConverter"/>
        </ResourceDictionary>
    </Application.Resources>
</Application>
```

////////////////////// App.xaml.cs//////////////////////

```
using Autofac;
using Kakuro.StartUp;
using System.Windows;

namespace Kakuro
{
    public partial class App : Application
    {
        protected override void OnStartup(StartupEventArgs e)
        {
            base.OnStartup(e);
            var bootStrapper = new BootStrapper();
            var container = bootStrapper.BootStrap();

            var mainWindow = container.Resolve<MainWindow>();
            mainWindow.Show();
        }
    }

}
```

////////////////////// Folder Base Classes//////////////////////
////////////////////// RelayCommand.cs//////////////////////
```
using System.Windows.Input;

namespace Kakuro.Base_Classes
{
    public abstract class RelayCommand : ICommand
    {
        public event EventHandler? CanExecuteChanged;

        public virtual bool CanExecute(object? parameter)
        {
            return true;
        }

        public abstract void Execute(object? parameter);

        protected void OnCanExecutedChanged()
        {
            CanExecuteChanged?.Invoke(this, new EventArgs());
        }
    }
}
```
////////////////////// ViewModelBase.cs//////////////////////
```
using System.ComponentModel;
using System.Runtime.CompilerServices;

namespace Kakuro.Base_Classes
{
    public class ViewModelBase : INotifyPropertyChanged
    {
        public event PropertyChangedEventHandler? PropertyChanged;

        protected virtual void OnPropertyChanged([CallerMemberName] string propertyName =
null)
        {
            PropertyChanged?.Invoke(this, new PropertyChangedEventArgs(propertyName));
        }
    }
}
```
////////////////////// Folder Commands//////////////////////
////////////////////// Subfolder DashboardViewModel//////////////////////
////////////////////// AddMinuteAndContinueStopwatchCommand.cs//////////////////////
```
using Kakuro.Base_Classes;
using Kakuro.Models;
using System.Windows.Input;

namespace Kakuro.Commands.DashboardViewModel
{
    public class AddMinuteAndContinueStopwatchCommand : RelayCommand
    {
        private MyStopwatch _stopwatch;
        private ICommand _startStopwatchCommand;

        public AddMinuteAndContinueStopwatchCommand(MyStopwatch stopwatch, ICommand
startStopwatchCommand)
        {
            _stopwatch = stopwatch;
            _startStopwatchCommand = startStopwatchCommand;
        }

        public override void Execute(object? parameter)
        {
```

```
            _stopwatch.AddTime(TimeSpan.FromMinutes(1));

            _startStopwatchCommand.Execute(parameter);
        }
    }
}
```

////////////////////// ApplyDifficultyCommand.cs//////////////////////

```
using Kakuro.Base_Classes;
using Kakuro.Enums;
using Kakuro.Events;
using Kakuro.Interfaces.Data_Access.Data_Providers;
using Kakuro.Models;
using System.Windows.Input;

namespace Kakuro.Commands.DashboardViewModel
{
    // #BAD: tests shall be REwritten
    public class ApplyDifficultyCommand : RelayCommand
    {
        private IDashboardProvider _dashboardProvider;
        private ViewModels.DashboardViewModel _dashboardViewModel;
        private ICommand _restartStopwatchCommand;
        private IEventAggregator _eventAggregator;
        private MyStopwatch _stopwatch;

        public ApplyDifficultyCommand(
            IDashboardProvider dashboardProvider,
            ViewModels.DashboardViewModel dashboardViewModel,
            ICommand restartStopwatchCommand,
            IEventAggregator eventAggregator,
            MyStopwatch stopwatch)
        {
            _dashboardProvider ??= dashboardProvider;
            _dashboardViewModel ??= dashboardViewModel;
            _restartStopwatchCommand ??= restartStopwatchCommand;
            _eventAggregator ??= eventAggregator;
            _stopwatch ??= stopwatch;
        }

        public override void Execute(object? parameter)
        {
            if (parameter == null)
                throw new NullReferenceException("Parameter for ApplyDifficultyCommand is
null!");

            if (!Enum.TryParse<DifficultyLevels>(parameter.ToString(), out var difficul-
tyLevel))
                throw new ArgumentException("Parameter for ApplyDifficultyCommand is of
incorrect type!");

            _stopwatch.CleanAdditionalTime();

            _dashboardViewModel.ChoosenDifficulty = difficultyLevel;

            _restartStopwatchCommand.Execute(parameter);

            _dashboardProvider.GenerateDashboard(difficultyLevel);

            _eventAggregator.GetEvent<NewGameStartedEvent>().Publish(true);

            _dashboardViewModel.IsGameCompleted = false;
        }
    }
}
```

////////////////////// AutoSubmitCommand.cs//////////////////////

```
using Kakuro.Base_Classes;
using Kakuro.Enums;
using System.Windows.Input;

namespace Kakuro.Commands.DashboardViewModel
{
    public class AutoSubmitCommand : RelayCommand
    {
        private ViewModels.DashboardViewModel _dashboardViewModel;
        private DashboardItemCollection _dashboard;
        private ICommand _validateSolutionCommand;

        public AutoSubmitCommand(ViewModels.DashboardViewModel dashboardViewModel, ICom-
mand validateSolutionCommand)
        {
            _dashboardViewModel = dashboardViewModel;
            _dashboard = _dashboardViewModel.Dashboard;
            _validateSolutionCommand = validateSolutionCommand;
        }
        public override void Execute(object? parameter)
        {
            if (!_dashboardViewModel.AutoSubmit)
                return;

            var cellType = (CellType)parameter;

            if (cellType != CellType.ValueCell)
                return;

            bool allElementsAreFilled = true;
            int dashboardSize = _dashboard.Count;
            ViewModels.DashboardItemViewModel currentElement;

            for (int i = 1; i < dashboardSize - 1; i++)
            {
                for (int j = 1; j < dashboardSize - 1; j++)
                {
                    currentElement = _dashboard[i][j];

                    if (currentElement.CellType == CellType.ValueCell && currentEle-
ment.DisplayValue == "")
                    {
                        allElementsAreFilled = false;
                        break;
                    }
                }
                if (!allElementsAreFilled)
                    break;
            }

            if (allElementsAreFilled)
                _validateSolutionCommand.Execute(parameter);
        }
    }
}
```

////////////////////// CleanDashboardCommand.cs//////////////////////

```
using Kakuro.Base_Classes;
using Kakuro.Enums;
using Kakuro.ViewModels;

namespace Kakuro.Commands.DashboardViewModel
{
    // #BAD: tests shall be written
    public class CleanDashboardCommand : RelayCommand
    {
        private DashboardItemCollection _dashboard;
```

```
        public CleanDashboardCommand(DashboardItemCollection dashboard)
        {
            _dashboard ??= dashboard;
        }

        public override void Execute(object? parameter)
        {
            int dashboardSize = _dashboard.Count;
            DashboardItemViewModel currentElement;

            for (int i = 1; i < dashboardSize - 1; i++)
                for (int j = 1; j < dashboardSize - 1; j++)
                {
                    currentElement = _dashboard[i][j];
                    if (currentElement.CellType == CellType.ValueCell)
                        currentElement.DisplayValue = "";
                }
        }
    }
}
```

////////////////////// GetChangedSettingsCommand.cs//////////////////////

```
using Kakuro.Base_Classes;
using Kakuro.Enums;
using System.Collections.ObjectModel;
using System.Windows.Input;

namespace Kakuro.Commands.DashboardViewModel
{
    public class GetChangedSettingsCommand : RelayCommand
    {
        private ICommand _showCorrectAnswersCommand;
        private ICommand _autoSubmitCommand;
        private ICommand _hideTimerCommand;

        public GetChangedSettingsCommand(
            ViewModels.DashboardViewModel dashboardViewModel,
            ICommand stopStopwatchCommand,
            ICommand addMinuteAndContinueStopwatchCommand,
            ICommand cleanDashboardCommand,
            IEventAggregator eventAggregator)
        {
            _showCorrectAnswersCommand = new Com-
mands.GetChangedSettingsCommand.ApplySettingShowCorrectAnswersCommand(
                dashboardViewModel,
                stopStopwatchCommand,
                addMinuteAndContinueStopwatchCommand,
                cleanDashboardCommand,
                eventAggregator);

            _autoSubmitCommand = new Com-
mands.GetChangedSettingsCommand.ApplySettingAutoSubmitCommand(dashboardViewModel);

            _hideTimerCommand = new Com-
mands.GetChangedSettingsCommand.ApplySettingHideTimerCommand(dashboardViewModel);
        }

        public override void Execute(object? parameter)
        {
            var settings = (ObservableCollection<ViewModels.SettingViewModel>)parameter;

            _showCorrectAnswersCommand.Execute(settings.FirstOrDefault(el =>
el.SettingType == SettingType.ShowCorrectAnswers));

            _autoSubmitCommand.Execute(settings.FirstOrDefault(el => el.SettingType ==
SettingType.AutoSubmit));
```

```
            _hideTimerCommand.Execute(settings.FirstOrDefault(el => el.SettingType ==
SettingType.HideTimer));
        }
    }
}
```

////////////////////// OpenRulesCommand.cs//////////////////////

```
using Kakuro.Base_Classes;
using Kakuro.Custom_Components;

namespace Kakuro.Commands.DashboardViewModel
{
    public class OpenRulesCommand : RelayCommand
    {
        public override void Execute(object? parameter)
        {
            var rulesWindow = new RulesWindow();
            rulesWindow.ShowDialog();
        }
    }
}
```

////////////////////// RestartStopwatchCommand.cs//////////////////////

```
using Kakuro.Base_Classes;
using Kakuro.Models;
using System.Windows.Input;

namespace Kakuro.Commands.DashboardViewModel
{
    // #BAD: tests shall be written
    public class RestartStopwatchCommand : RelayCommand
    {
        private MyStopwatch _stopwatch;
        private ICommand _startStopwatchCommand;
        private ICommand _stopStopwatchCommand;

        public RestartStopwatchCommand(MyStopwatch stopwatch, ICommand startStopwatch-
Command, ICommand stopStopwatchCommand)
        {
            _stopwatch ??= stopwatch;
            _startStopwatchCommand ??= startStopwatchCommand;
            _stopStopwatchCommand ??= stopStopwatchCommand;
        }

        public override void Execute(object? parameter)
        {
            _stopStopwatchCommand.Execute(parameter);
            _stopwatch.Reset();
            _startStopwatchCommand.Execute(parameter);
        }
    }
}
```

////////////////////// SendGameSessionCommand.cs//////////////////////

```
using Kakuro.Base_Classes;
using Kakuro.Events;
using Kakuro.Models;

namespace Kakuro.Commands.DashboardViewModel
{
    public class SendGameSessionCommand : RelayCommand
    {
        private ViewModels.DashboardViewModel _dashboardViewModel;
        private IEventAggregator _eventAggregator;

        public SendGameSessionCommand(ViewModels.DashboardViewModel dashboardViewModel,
IEventAggregator eventAggregator)
```

```
        {
            _dashboardViewModel ??= dashboardViewModel;
            _eventAggregator ??= eventAggregator;
        }

        public override void Execute(object? parameter)
        {
            GameSession session = new GameSession(
                _dashboardViewModel.ChoosenDifficulty,
                _dashboardViewModel.StopWatchHours,
                _dashboardViewModel.StopWatchMinutes,
                _dashboardViewModel.StopWatchSeconds);

            _eventAggregator.GetEvent<GameCompletedEvent>().Publish(session);
        }
    }
}
```

////////////////////// StartStopwatchCommand.cs//////////////////////

```
using Kakuro.Base_Classes;
using Kakuro.Models;
using System.Windows;

namespace Kakuro.Commands.DashboardViewModel
{
    // #BAD: tests shall be written
    public class StartStopwatchCommand : RelayCommand
    {
        private MyStopwatch _stopwatch;
        private ViewModels.DashboardViewModel _viewModel;

        public StartStopwatchCommand(MyStopwatch stopwatch, ViewModels.DashboardViewModel
viewModel)
        {
            _stopwatch ??= stopwatch;
            _viewModel ??= viewModel;
        }

        public override void Execute(object? parameter)
        {
            _stopwatch.Start();
            Task.Run(async () =>
            {
                while (_stopwatch.IsRunning && Application.Current != null)
                {
                    Application.Current.Dispatcher.Invoke(() =>
                    {
                        _viewModel.StopWatchHours = _stopwatch.ElapsedHours.ToString();
                        _viewModel.StopWatchMinutes =
_stopwatch.ElapsedMinutes.ToString();
                        _viewModel.StopWatchSeconds =
_stopwatch.ElapsedSeconds.ToString();
                    });

                    await Task.Delay(1000);
                }
            });
        }
    }
}
```

////////////////////// StopStopwatchCommand.cs//////////////////////

```
using Kakuro.Base_Classes;
using Kakuro.Models;

namespace Kakuro.Commands.DashboardViewModel
{
```

```
    // #BAD: tests shall be written
    public class StopStopwatchCommand : RelayCommand
    {
        private MyStopwatch _stopwatch;

        public StopStopwatchCommand(MyStopwatch stopwatch)
        {
            _stopwatch ??= stopwatch;
        }

        public override void Execute(object? parameter)
        {
            if (_stopwatch.IsRunning)
                _stopwatch.Stop();
        }
    }
}
```

////////////////////// ValidateSolutionCommand.cs//////////////////////

```
using Kakuro.Base_Classes;
using Kakuro.Interfaces.Game_Tools;
using System.Windows.Input;

namespace Kakuro.Commands.DashboardViewModel
{
    // #BAD: tests shall be written
    public class ValidateSolutionCommand : RelayCommand
    {
        private ISolutionValidator _solutionVerifier;
        private IOperationNotifier _operationNotifier;
        private ICommand _stopStopwatchCommand;
        private ICommand _sentGameSessionCommand;
        private ViewModels.DashboardViewModel _viewModel;

        public ValidateSolutionCommand(
            ISolutionValidator solutionVerifier,
            IOperationNotifier operationNotifier,
            ICommand stopStopwatchCommand,
            ICommand sentGameSessionCommand,
            ViewModels.DashboardViewModel viewModel)
        {
            _solutionVerifier ??= solutionVerifier;
            _operationNotifier ??= operationNotifier;
            _stopStopwatchCommand ??= stopStopwatchCommand;
            _sentGameSessionCommand ??= sentGameSessionCommand;
            _viewModel ??= viewModel;
        }

        public override void Execute(object? parameter)
        {
            string message, failMessage = "", successMessage = "";
            bool isSolutionCorrect = _solutionVerifier.ValidateDashboard(out message);

            if (isSolutionCorrect)
            {
                successMessage = message;
                _viewModel.IsGameCompleted = true;
                _stopStopwatchCommand.Execute(parameter);
                _sentGameSessionCommand.Execute(parameter);
            }
            else
                failMessage = message;

            _operationNotifier.NotifyOutcome(isSolutionCorrect, successMessage, failMes-
sage);
        }
    }
```

```
}
```
////////////////////// Subfolder GetChangedSettingsCommand//////////////////////
////////////////////// ApplySettingAutoSubmitCommand.cs//////////////////////
```
using Kakuro.Base_Classes;

namespace Kakuro.Commands.GetChangedSettingsCommand
{
    public class ApplySettingAutoSubmitCommand : RelayCommand
    {
        private ViewModels.DashboardViewModel _dashboardViewModel;

        public ApplySettingAutoSubmitCommand(ViewModels.DashboardViewModel dash-
boardViewModel)
        {
            _dashboardViewModel = dashboardViewModel;
        }
        public override void Execute(object? parameter)
        {
            var autoSubmitSetting = (ViewModels.SettingViewModel)parameter;

            _dashboardViewModel.AutoSubmit = autoSubmitSetting.IsEnabled;
        }
    }
}
```
////////////////////// ApplySettingHideTimerCommand.cs//////////////////////
```
using Kakuro.Base_Classes;

namespace Kakuro.Commands.GetChangedSettingsCommand
{
    class ApplySettingHideTimerCommand : RelayCommand
    {
        private ViewModels.DashboardViewModel _dashboardViewModel;

        public ApplySettingHideTimerCommand(ViewModels.DashboardViewModel dashboardView-
Model)
        {
            _dashboardViewModel = dashboardViewModel;
        }

        public override void Execute(object? parameter)
        {
            var hideTimerSetting = (ViewModels.SettingViewModel)parameter;

            _dashboardViewModel.IsTimerVisible = !hideTimerSetting.IsEnabled;
        }
    }
}
```
////////////////////// ApplySettingShowCorrectAnswersCommand.cs//////////////////////
```
using Kakuro.Base_Classes;
using Kakuro.Events;
using System.Windows.Input;

namespace Kakuro.Commands.GetChangedSettingsCommand
{
    public class ApplySettingShowCorrectAnswersCommand : RelayCommand
    {
        private ViewModels.DashboardViewModel _dashboardViewModel;
        private ICommand _stopStopwatchCommand;
        private ICommand _addMinuteAndContinueStopwatchCommand;
        private ICommand _cleanDashboardCommand;
        private ICommand _showCorrectAnswersCommand;
        private IEventAggregator _eventAggregator;

        public ApplySettingShowCorrectAnswersCommand(
```

```
            ViewModels.DashboardViewModel dashboardViewModel,
            ICommand stopStopwatchCommand,
            ICommand addMinuteAndContinueStopwatchCommand,
            ICommand cleanDashboardCommand,
            IEventAggregator eventAggregator)
        {
            _dashboardViewModel = dashboardViewModel;
            _stopStopwatchCommand = stopStopwatchCommand;
            _addMinuteAndContinueStopwatchCommand = addMinuteAndContinueStopwatchCommand;
            _cleanDashboardCommand = cleanDashboardCommand;
            _showCorrectAnswersCommand = new ShowCorrectAnswersCom-
mand(dashboardViewModel.Dashboard);
            _eventAggregator = eventAggregator;
        }

        public override void Execute(object? parameter)
        {
            var showCorrectAnswersSetting = (ViewModels.SettingViewModel)parameter;

            bool wasChanged = false;

            if (_dashboardViewModel.ShowCorrectAnswers != showCorrectAnswersSet-
ting.IsEnabled)
                wasChanged = true;

            _dashboardViewModel.ShowCorrectAnswers = showCorrectAnswersSetting.IsEnabled;

            if (wasChanged)
            {
                TurnOffAutoSubmit(_dashboardViewModel.ShowCorrectAnswers);

                LockStopwatch(_dashboardViewModel.ShowCorrectAnswers);

                ShowOrEraseCorrectAnswers(_dashboardViewModel.ShowCorrectAnswers);
            }
        }

        private void TurnOffAutoSubmit(bool showCorrectAnswers)
        {
            if (showCorrectAnswers)
                _dashboardViewModel.AutoSubmit = false;


_eventAggregator.GetEvent<CorrectAnswersTurnedOnEvent>().Publish(_dashboardViewModel.Auto
Submit);
        }

        private void LockStopwatch(bool showCorrectAnswers)
        {
            if (showCorrectAnswers)
                _stopStopwatchCommand.Execute(null);
            else
                _addMinuteAndContinueStopwatchCommand.Execute(null);
        }

        private void ShowOrEraseCorrectAnswers(bool showCorrectAnswers)
        {
            if (showCorrectAnswers)
                _showCorrectAnswersCommand.Execute(null);
            else
                _cleanDashboardCommand.Execute(null);
        }
    }
}
```

////////////////////// ShowCorrectAnswersCommand.cs//////////////////////

```
using Kakuro.Base_Classes;
```

```
using Kakuro.Enums;
using Kakuro.ViewModels;

namespace Kakuro.Commands.GetChangedSettingsCommand
{
    public class ShowCorrectAnswersCommand : RelayCommand
    {
        private DashboardItemCollection _dashboard;

        public ShowCorrectAnswersCommand(DashboardItemCollection dashboard)
        {
            _dashboard ??= dashboard;
        }

        public override void Execute(object? parameter)
        {
            int dashboardSize = _dashboard.Count;
            DashboardItemViewModel currentElement;

            for (int i = 1; i < dashboardSize - 1; i++)
                for (int j = 1; j < dashboardSize - 1; j++)
                {
                    currentElement = _dashboard[i][j];
                    if (currentElement.CellType == CellType.ValueCell)
                        currentElement.DisplayValue = currentElement.HiddenValue;
                }
        }
    }
}
```

////////////////////// Subfolder RatingTableViewModel//////////////////////

////////////////////// LoadRatingRecordsCommand.cs//////////////////////

```
using Kakuro.Base_Classes;
using Kakuro.Enums;
using Kakuro.Interfaces.Data_Access.Data_Providers;
using System.Collections.ObjectModel;

namespace Kakuro.Commands.RatingTableViewModel
{
    // #BAD: tests shall be written
    public class LoadRatingRecordsCommand : RelayCommand
    {
        private IRatingRecordProvider _ratingRecordProvider;
        private RatingTableContainer _ratingTablesContainer;

        public LoadRatingRecordsCommand(IRatingRecordProvider ratingRecordProvider, Rat-
ingTableContainer ratingTablesContainer)
        {
            _ratingRecordProvider ??= ratingRecordProvider;
            _ratingTablesContainer ??= ratingTablesContainer;
        }
        public override void Execute(object? parameter)
        {
            var difficultyLevels = (DifficultyLev-
els[])Enum.GetValues(typeof(DifficultyLevels));

            foreach (var difficultyLevel in difficultyLevels)
                LoadDataForConcreteDifficulty(difficultyLevel);
        }

        private void LoadDataForConcreteDifficulty(DifficultyLevels difficultyLevel)
        {
            var iEnumerableData = _ratingRecordProvider.GetAll(difficultyLevel);
            var convertedData = ConvertIEnumerableToObservable(iEnumerableData);

            _ratingTablesContainer[difficultyLevel].Clear();
```

```
            foreach (var ratingRecord in convertedData)
                _ratingTablesContainer[difficultyLevel].Add(ratingRecord);
        }

        private ObservableCollection<T> ConvertIEnumerableToObservable<T>(IEnumerable<T>
values) => new ObservableCollection<T>(values);
    }
}
```

////////////////////// SaveRatingRecordCommand.cs//////////////////////

```
using Kakuro.Base_Classes;
using Kakuro.Enums;
using Kakuro.Interfaces.Data_Access.Data_Providers;
using Kakuro.Models;
using static System.Convert;

namespace Kakuro.Commands.RatingTableViewModel
{
    // #BAD: tests shall be written
    public class SaveRatingRecordCommand : RelayCommand
    {
        private IRatingRecordProvider _ratingRecordProvider;

        public event EventHandler? SaveCompleted;

        public SaveRatingRecordCommand(IRatingRecordProvider ratingRecordProvider)
        {
            _ratingRecordProvider ??= ratingRecordProvider;
        }
        public override void Execute(object? parameter)
        {
            if (parameter == null)
                throw new NullReferenceException("Parameter for SaveRatingRecordCommand is
null!");

            GameSession session = (GameSession)parameter;

            (DifficultyLevels difficulty, string hours, string minutes, string seconds) =
session;

            RatingRecord ratingRecord = new RatingRecord(ToInt32(hours), ToInt32(minutes),
ToInt32(seconds));

            _ratingRecordProvider.Add(ratingRecord, difficulty);

            OnSaveCompleted();
        }

        protected virtual void OnSaveCompleted()
        {
            SaveCompleted?.Invoke(this, EventArgs.Empty);
        }
    }
}
```

////////////////////// Subfolder SavepointsViewModel//////////////////////

////////////////////// ChangeAvailabilityOfSectionCommand.cs//////////////////////

```
using Kakuro.Base_Classes;

namespace Kakuro.Commands.SavepointsViewModel
{
    public class ChangeAvailabilityOfSectionCommand : RelayCommand
    {
        private ViewModels.SavepointsViewModel _savepointsViewModel;
```

```
        public ChangeAvailabilityOfSectionCommand(ViewModels.SavepointsViewModel save-
pointsViewModel)
        {
            _savepointsViewModel = savepointsViewModel;
        }

        public override void Execute(object? parameter)
        {
            _savepointsViewModel.CorrectAnswersAreShown
= !_savepointsViewModel.CorrectAnswersAreShown;
        }
    }
}
```

////////////////////// CleanSavepointsCommand.cs//////////////////////

```
using Kakuro.Base_Classes;
using Kakuro.Interfaces.Data_Access.Data_Providers;

namespace Kakuro.Commands.SavepointsViewModel
{
    public class CleanSavepointsCommand : RelayCommand
    {
        private ViewModels.SavepointsViewModel _savepointsViewModel;
        private ISavepointProvider _savepointProvider;

        public CleanSavepointsCommand(ViewModels.SavepointsViewModel savepointsViewModel,
ISavepointProvider savepointProvider)
        {
            _savepointsViewModel = savepointsViewModel;
            _savepointProvider = savepointProvider;
        }

        public override void Execute(object? parameter)
        {
            for (int i = 0; i < _savepointsViewModel.Savepoints.Count; i++)
                _savepointProvider.Delete(_savepointsViewModel.Savepoints[i].Id);

            _savepointsViewModel.Savepoints.Clear();
        }
    }
}
```

////////////////////// CreateSavepointCommand.cs//////////////////////

```
using Kakuro.Base_Classes;
using Kakuro.Interfaces.Data_Access.Data_Providers;
using Kakuro.Models;
using Microsoft.VisualBasic;
using System.ComponentModel;

namespace Kakuro.Commands.SavepointsViewModel
{
    public class CreateSavepointCommand : RelayCommand, IDisposable
    {
        private readonly ViewModels.SavepointsViewModel _savepointsViewModel;
        private readonly ViewModels.DashboardViewModel _dashboardViewModel;
        private readonly ISavepointProvider _savepointProvider;
        private bool _disposed = false;

        public CreateSavepointCommand(
            ViewModels.SavepointsViewModel savepointsViewModel,
            ViewModels.DashboardViewModel dashboardViewModel,
            ISavepointProvider savepointProvider)
        {
            _savepointsViewModel = savepointsViewModel;
            _dashboardViewModel = dashboardViewModel;
            _savepointProvider = savepointProvider;
```

```
            _savepointsViewModel.PropertyChanged += OnSavepointsCountPropertyChanged;
        }

        public override void Execute(object? parameter)
        {
            var dashboard = _dashboardViewModel.CreateDashboardCopy();

            var newId = _savepointsViewModel.Savepoints.Count + 1;

            while (_savepointsViewModel.Savepoints.Any(sp => sp.Id == newId))
            {
                newId++;
            }

            string defaultName = $"Savepoint #{newId}";
            string inputName = Interaction.InputBox("Enter a name for the Savepoint:",
"Create Savepoint", defaultName);

            if (string.IsNullOrEmpty(inputName))
            {
                inputName = defaultName;
            }

            var newSavepoint = new Savepoint(newId, dashboard);

            _savepointProvider.Add(newSavepoint);

            _savepointsViewModel.Savepoints.Add(new ViewModels.SavepointViewModel(newId,
inputName));

            _savepointsViewModel.IsCreatingAllowed = _savepointsViewModel.Savepoints.Count
< 10;
        }

        public override bool CanExecute(object? parameter)
        {
            return NotMaxSavepointsCreated() && !AreCorrectAnswersShown() &&
base.CanExecute(parameter);
        }

        private bool NotMaxSavepointsCreated() => _savepointsViewModel.Savepoints.Count <
10;

        private bool AreCorrectAnswersShown() =>
_savepointsViewModel.CorrectAnswersAreShown;

        private void OnSavepointsCountPropertyChanged(object? sender, Property-
ChangedEventArgs e)
        {
            if (e.PropertyName == nameof(_savepointsViewModel.Savepoints.Count) ||
                e.PropertyName == nameof(_savepointsViewModel.CorrectAnswersAreShown))
                OnCanExecutedChanged();
        }

        public void Dispose()
        {
            Dispose(true);
            GC.SuppressFinalize(this);
        }

        protected virtual void Dispose(bool disposing)
        {
            if (!_disposed)
            {
                if (disposing)
                {
```

```
                    _savepointsViewModel.PropertyChanged -= OnSavepointsCountProperty-
Changed;
                }
                _disposed = true;
            }
        }

        ~CreateSavepointCommand()
        {
            Dispose(false);
        }
    }
}
```

////////////////////// DeleteSavepointCommand.cs//////////////////////

```
using Kakuro.Base_Classes;
using Kakuro.Interfaces.Data_Access.Data_Providers;
using System.ComponentModel;

namespace Kakuro.Commands.SavepointsViewModel
{
    public class DeleteSavepointCommand : RelayCommand, IDisposable
    {
        private ViewModels.SavepointsViewModel _savepointsViewModel;
        private ISavepointProvider _savepointProvider;
        private bool _disposed = false;

        public DeleteSavepointCommand(ViewModels.SavepointsViewModel savepointsViewModel,
ISavepointProvider savepointProvider)
        {
            _savepointsViewModel = savepointsViewModel;
            _savepointProvider = savepointProvider;

            _savepointsViewModel.PropertyChanged += OnSelectedSavepointPropertyChanged;
        }

        public override void Execute(object? parameter)
        {
            _savepointProvider.Delete(_savepointsViewModel.SelectedSavepoint.Id);


_savepointsViewModel.Savepoints.Remove(_savepointsViewModel.SelectedSavepoint);
            _savepointsViewModel.SelectedSavepoint = null;
        }

        public override bool CanExecute(object? parameter)
        {
            return AnySavepointSelected() && !AreCorrectAnswersShown() &&
base.CanExecute(parameter);
        }

        private bool AnySavepointSelected() => _savepointsViewModel.SelectedSavepoint !=
null;

        private bool AreCorrectAnswersShown() =>
_savepointsViewModel.CorrectAnswersAreShown;

        private void OnSelectedSavepointPropertyChanged(object? sender, Property-
ChangedEventArgs e)
        {
            if (e.PropertyName == nameof(_savepointsViewModel.SelectedSavepoint) ||
                e.PropertyName == nameof(_savepointsViewModel.CorrectAnswersAreShown))
                OnCanExecutedChanged();
        }

        public void Dispose()
        {
```

```
            Dispose(true);
            GC.SuppressFinalize(this);
        }

        protected virtual void Dispose(bool disposing)
        {
            if (!_disposed)
            {
                if (disposing)
                {
                    _savepointsViewModel.PropertyChanged -= OnSelectedSavepointProperty-
Changed;
                }
                _disposed = true;
            }
        }

        ~DeleteSavepointCommand()
        {
            Dispose(false);
        }
    }
}
```

////////////////////// LoadSavepointCommand.cs//////////////////////

```
using Kakuro.Base_Classes;
using Kakuro.Enums;
using Kakuro.Interfaces.Data_Access.Data_Providers;
using System.ComponentModel;

namespace Kakuro.Commands.SavepointsViewModel
{
    public class LoadSavepointCommand : RelayCommand, IDisposable
    {
        private ViewModels.SavepointsViewModel _savepointsViewModel;
        private readonly ViewModels.DashboardViewModel _dashboardViewModel;
        private readonly ISavepointProvider _savepointProvider;
        private bool _disposed = false;

        public LoadSavepointCommand(
            ViewModels.SavepointsViewModel savepointsViewModel,
            ViewModels.DashboardViewModel dashboardViewModel,
            ISavepointProvider savepointProvider)
        {
            _savepointsViewModel = savepointsViewModel;
            _dashboardViewModel = dashboardViewModel;
            _savepointProvider = savepointProvider;

            _savepointsViewModel.PropertyChanged += OnSelectedSavepointPropertyChanged;
        }

        public override void Execute(object? parameter)
        {
            var savepointToLoad =
_savepointProvider.GetById(_savepointsViewModel.SelectedSavepoint.Id);

            int dashboardSize = _dashboardViewModel.Dashboard.Count;
            ViewModels.DashboardItemViewModel currentElement;

            for (int i = 1; i < dashboardSize - 1; i++)
            {
                for (int j = 1; j < dashboardSize - 1; j++)
                {
                    currentElement = _dashboardViewModel.Dashboard[i][j];
                    if (currentElement.CellType == CellType.ValueCell)
                        currentElement.DisplayValue = savepointTo-
Load.Dashboard[i][j].DisplayValue;
```

```
                }
            }
        }

        public override bool CanExecute(object? parameter)
        {
            return AnySavepointSelected() && !AreCorrectAnswersShown() &&
base.CanExecute(parameter);
        }

        private bool AnySavepointSelected() => _savepointsViewModel.SelectedSavepoint !=
null;

        private bool AreCorrectAnswersShown() =>
_savepointsViewModel.CorrectAnswersAreShown;

        private void OnSelectedSavepointPropertyChanged(object? sender, Property-
ChangedEventArgs e)
        {
            if (e.PropertyName == nameof(_savepointsViewModel.SelectedSavepoint) ||
                e.PropertyName == nameof(_savepointsViewModel.CorrectAnswersAreShown))
                OnCanExecutedChanged();
        }

        public void Dispose()
        {
            Dispose(true);
            GC.SuppressFinalize(this);
        }

        protected virtual void Dispose(bool disposing)
        {
            if (!_disposed)
            {
                if (disposing)
                {
                    _savepointsViewModel.PropertyChanged -= OnSelectedSavepointProperty-
Changed;
                }
                _disposed = true;
            }
        }

        ~LoadSavepointCommand()
        {
            Dispose(false);
        }
    }

}
```

////////////////////// RewriteSavepointCommand.cs//////////////////////

```
using Kakuro.Base_Classes;
using Kakuro.Interfaces.Data_Access.Data_Providers;
using Kakuro.Interfaces.Game_Tools;
using System.ComponentModel;

namespace Kakuro.Commands.SavepointsViewModel
{
    public class RewriteSavepointCommand : RelayCommand, IDisposable
    {
        private ViewModels.SavepointsViewModel _savepointsViewModel;
        private readonly ViewModels.DashboardViewModel _dashboardViewModel;
        private readonly ISavepointProvider _savepointProvider;
        private readonly IOperationNotifier _operationNotifier;
        private bool _disposed = false;
```

```
        public RewriteSavepointCommand(
            ViewModels.SavepointsViewModel savepointsViewModel,
            ViewModels.DashboardViewModel dashboardViewModel,
            ISavepointProvider savepointProvider,
            IOperationNotifier operationNotifier)
        {
            _savepointsViewModel = savepointsViewModel;
            _dashboardViewModel = dashboardViewModel;
            _savepointProvider = savepointProvider;
            _operationNotifier = operationNotifier;

            _savepointsViewModel.PropertyChanged += OnSelectedSavepointPropertyChanged;
        }

        public override void Execute(object? parameter)
        {
            var savepointToRewrite =
_savepointProvider.GetById(_savepointsViewModel.SelectedSavepoint.Id);
            savepointToRewrite.Dashboard = _dashboardViewModel.CreateDashboardCopy();
            _savepointProvider.Update(savepointToRewrite);
            _operationNotifier.NotifySuccess("Dashboard was successfully rewritten!");
        }

        public override bool CanExecute(object? parameter)
        {
            return AnySavepointSelected() && !AreCorrectAnswersShown() &&
base.CanExecute(parameter);
        }

        private bool AnySavepointSelected() => _savepointsViewModel.SelectedSavepoint !=
null;

        private bool AreCorrectAnswersShown() =>
_savepointsViewModel.CorrectAnswersAreShown;

        private void OnSelectedSavepointPropertyChanged(object? sender, Property-
ChangedEventArgs e)
        {
            if (e.PropertyName == nameof(_savepointsViewModel.SelectedSavepoint) ||
                e.PropertyName == nameof(_savepointsViewModel.CorrectAnswersAreShown))
                OnCanExecutedChanged();
        }

        public void Dispose()
        {
            Dispose(true);
            GC.SuppressFinalize(this);
        }

        protected virtual void Dispose(bool disposing)
        {
            if (!_disposed)
            {
                if (disposing)
                {
                    _savepointsViewModel.PropertyChanged -= OnSelectedSavepointProperty-
Changed;
                }
                _disposed = true;
            }
        }

        ~RewriteSavepointCommand()
        {
            Dispose(false);
        }
```

```
    }
}
```
////////////////////// Subfolder SettingsViewModel//////////////////////
////////////////////// SendSettingsCommand.cs//////////////////////
```
using Kakuro.Base_Classes;
using Kakuro.Events;
using Kakuro.ViewModels;
using System.Collections.ObjectModel;

namespace Kakuro.Commands.SettingsViewModel
{
    // #BAD: tests shall be written
    public class SendSettingsCommand : RelayCommand
    {
        private IEventAggregator _eventAggregator;
        private ObservableCollection<SettingViewModel> _settings;

        public SendSettingsCommand(IEventAggregator eventAggregator, ObservableCollec-
tion<SettingViewModel> settings)
        {
            _eventAggregator = eventAggregator;
            _settings = settings;
        }

        public override void Execute(object? parameter)
        {
            _eventAggregator.GetEvent<SettingsChangedEvent>().Publish(_settings);
        }
    }
}
```
////////////////////// TurnOffAutoSubmitCommand.cs//////////////////////
```
using Kakuro.Base_Classes;
using Kakuro.Enums;
using Kakuro.ViewModels;
using System.Collections.ObjectModel;

namespace Kakuro.Commands.SettingsViewModel
{
    public class TurnOffAutoSubmitCommand : RelayCommand
    {
        private ObservableCollection<SettingViewModel> _settings;

        public TurnOffAutoSubmitCommand(ObservableCollection<SettingViewModel> settings)
        {
            _settings = settings;
        }

        public override void Execute(object? parameter)
        {
            var autoSubmitEnabled = (bool)parameter;
            var autoSubmitSetting = _settings.FirstOrDefault(s => s.SettingType == Set-
tingType.AutoSubmit);
            autoSubmitSetting.IsEnabled = autoSubmitEnabled;
        }
    }
}
```
////////////////////// Folder Converters//////////////////////
////////////////////// BooleanToVisibilityConverter.cs//////////////////////
```
using System.Globalization;
using System.Windows;
using System.Windows.Data;

namespace Kakuro.Converters
{
```

```
    public class BooleanToVisibilityConverter : IValueConverter
    {
        public object Convert(object value, Type targetType, object parameter, CultureInfo
culture)
        {
            if (value is bool boolValue)
            {
                return boolValue ? Visibility.Visible : Visibility.Hidden;
            }
            return Visibility.Visible;
        }

        public object ConvertBack(object value, Type targetType, object parameter, Cul-
tureInfo culture)
        {
            throw new NotImplementedException();
        }
    }
}
```

////////////////////// EnumDescriptionConverter.cs//////////////////////

```
using System.Globalization;
using System.Runtime.Serialization;
using System.Windows.Data;

namespace Kakuro.Converters
{
    public class EnumDescriptionConverter : IValueConverter
    {
        public object Convert(object value, Type targetType, object parameter, CultureInfo
culture)
        {
            if (value == null)
                return null;

            var enumValue = value.ToString();
            var field = value.GetType().GetField(enumValue);
            var attribute = (EnumMemberAttribute)Attribute.GetCustomAttribute(field,
typeof(EnumMemberAttribute));

            return attribute != null ? attribute.Value : enumValue;
        }

        public object ConvertBack(object value, Type targetType, object parameter, Cul-
tureInfo culture) => Binding.DoNothing;
    }
}
```

////////////////////// InverseBooleanConverter.cs//////////////////////

```
using System.Globalization;
using System.Windows.Data;

namespace Kakuro.Converters
{
    public class InverseBooleanConverter : IValueConverter
    {
        public object Convert(object value, Type targetType, object parameter, CultureInfo
culture)
        {
            return value is bool && !(bool)value;
        }

        public object ConvertBack(object value, Type targetType, object parameter, Cul-
tureInfo culture)
        {
            return value is bool && !(bool)value;
        }
```

```
    }
}
```
////////////////////// MultiOrBooleanConverter.cs//////////////////////
```
using System.Globalization;
using System.Windows.Data;

namespace Kakuro.Converters
{
    public class MultiOrBooleanConverter : IMultiValueConverter
    {
        public object Convert(object[] values, Type targetType, object parameter, CultureInfo
culture)
        {
            if (values.Length == 2 &&
                values[0] is bool isGameCompleted &&
                values[1] is bool showCorrectAnswers)
            {
                return !isGameCompleted && !showCorrectAnswers;
            }

            return false;
        }

        public object[] ConvertBack(object value, Type[] targetTypes, object parameter,
CultureInfo culture)
        {
            throw new NotImplementedException();
        }
    }
}
```
////////////////////// ReadOnlyMultiConverter.cs//////////////////////
```
using Kakuro.Enums;
using System.Globalization;
using System.Windows.Data;

namespace Kakuro.Converters
{
    public class ReadOnlyMultiConverter : IMultiValueConverter
    {
        public object Convert(object[] values, Type targetType, object parameter, CultureInfo
culture)
        {
            if (values.Length == 3 &&
                values[0] is bool isGameCompleted &&
                values[1] is bool showCorrectAnswers &&
                values[2] is CellType cellType)
            {
                if (showCorrectAnswers || isGameCompleted)
                    return true;

                if (cellType == CellType.EmptyCell || cellType == CellType.SumCell)
                    return true;

                return false;
            }

            return false;
        }

        public object[] ConvertBack(object value, Type[] targetTypes, object parameter,
CultureInfo culture)
        {
            throw new NotImplementedException();
        }
    }
```

```
}
```
////////////////////// WindowWidthToTabItemWidthConverter.cs//////////////////////
```
using System.Globalization;
using System.Windows.Data;

namespace Kakuro.Converters
{
    public class WindowWidthToTabItemWidthConverter : IValueConverter
    {
        private const int UNDISPLAYABLE_DPI = 10;

        public object Convert(object value, Type targetType, object parameter, CultureInfo
culture)
        {
            if (value is double windowWidth)
                return (windowWidth / 2) - UNDISPLAYABLE_DPI;

            return 0;
        }

        public object ConvertBack(object value, Type targetType, object parameter, Cul-
tureInfo culture)
        {
            throw new NotImplementedException();
        }
    }
}
```
////////////////////// Folder Custom Components//////////////////////
////////////////////// RulesWindow.xaml//////////////////////
```
<Window x:Class="Kakuro.Custom_Components.RulesWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        Title="Game Rules"
        WindowStartupLocation="CenterScreen"
        ResizeMode="NoResize"
        Width="400" Height="300">
    <Grid Margin="10">
        <TextBlock TextWrapping="Wrap" FontSize="14" Foreground="Black">
            <Run Text="Kakuro is a logic puzzle with simple rules and challenging solu-
tions."/>
            <LineBreak/>
            <LineBreak/>
            <Run Text="The rules of Kakuro are simple:"/>
            <LineBreak/>
            <Run Text="1. Each cell can contain numbers from 1 through 9."/>
            <LineBreak/>
            <Run Text="2. The clues in the black cells tell the sum of the numbers next to
that clue (on the right or down)."/>
            <LineBreak/>
            <Run Text="3. The numbers in consecutive white cells must be unique."/>
            <LineBreak/>
            <LineBreak/>
            <Run Text="Limitations:"/>
            <LineBreak/>
            <Run Text="1. You cannot have more than 10 savepoints."/>
            <LineBreak/>
            <Run Text="2. You cannot have more than 10 records in the rating table for each
difficulty level."/>
        </TextBlock>
    </Grid>
</Window>
```
////////////////////// RulesWindow.xaml.cs//////////////////////
```
using System.Windows;
```

```
namespace Kakuro.Custom_Components
{
    public partial class RulesWindow : Window
    {
        public RulesWindow()
        {
            InitializeComponent();
        }
    }
}
```

////////////////////// SumCell.cs//////////////////////

```
using Kakuro.Game_Tools;
using System.Windows;
using System.Windows.Controls;
using System.Windows.Input;

namespace Kakuro.Custom_Components
{
    // #BAD: tests shall be written
    public class SumCell : Control
    {
        static SumCell()
        {
            DefaultStyleKeyProperty.OverrideMetadata(typeof(SumCell), new FrameworkProp-
ertyMetadata(typeof(SumCell)));
        }

        public string SumRight
        {
            get { return (string)GetValue(SumRightProperty); }
            set { SetValue(SumRightProperty, value); }
        }

        public static readonly DependencyProperty SumRightProperty =
            DependencyProperty.Register("SumRight", typeof(string), typeof(SumCell), new
PropertyMetadata(string.Empty));

        public string SumBottom
        {
            get { return (string)GetValue(SumBottomProperty); }
            set { SetValue(SumBottomProperty, value); }
        }

        public static readonly DependencyProperty SumBottomProperty =
            DependencyProperty.Register("SumBottom", typeof(string), typeof(SumCell), new
PropertyMetadata(string.Empty));

        public SumCell()
        {
            Focusable = true;
            IsTabStop = true;
            KeyDown += SumCell_KeyDown;
        }

        private void SumCell_KeyDown(object sender, KeyEventArgs e)
        {
            FocusNavigationHelper.HandleWASDKeyDown(this, e);
        }
    }
}
```

////////////////////// ToastNotificationWindow.xaml//////////////////////

```
<Window x:Class="Kakuro.Custom_Components.ToastNotificationWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
```

```
        WindowStyle="None"
        AllowsTransparency="True"
        Background="Transparent"
        Topmost="True"
        ResizeMode="NoResize"
        Width="700"
        Height="75"
        ShowInTaskbar="False">

    <Border x:Name="NotificationBorder" Style="{StaticResource NotificationBorderStyle}">
        <Grid>
            <Grid.ColumnDefinitions>
                <ColumnDefinition Width="*"/>
                <ColumnDefinition Width="Auto"/>
            </Grid.ColumnDefinitions>

            <TextBlock x:Name="NotificationMessage" Style="{StaticResource Notification-
MessageStyle}" Grid.Column="0"/>

            <Button Content="✕"
                    Style="{StaticResource CloseButtonStyle}"
                    VerticalAlignment="Center"
                    HorizontalAlignment="Right"
                    Grid.Column="1"
                    Click="CloseButton_Click"/>
        </Grid>
    </Border>
</Window>
```

////////////////////// ToastNotificationWindow.xaml.cs//////////////////////

```
using System.Windows;
using System.Windows.Media;
using System.Windows.Media.Animation;

namespace Kakuro.Custom_Components
{
    /// <summary>
    /// Interaction logic for ToastNotificationWindow.xaml
    /// </summary>
    public partial class ToastNotificationWindow : Window
    {
        public ToastNotificationWindow(string message, string caption, bool isSuccess)
        {
            InitializeComponent();
            NotificationMessage.Text = $"{caption}: {message}";
            NotificationBorder.Background = isSuccess ? Brushes.Green : Brushes.Red;
            Loaded += ToastNotificationWindow_Loaded;
        }

        private void ToastNotificationWindow_Loaded(object sender, RoutedEventArgs e)
        {
            var workingArea = SystemParameters.WorkArea;
            Left = (workingArea.Width - Width) / 2;
            Top = workingArea.Height;

            DoubleAnimation animation = new DoubleAnimation
            {
                From = workingArea.Height,
                To = workingArea.Height - Height - 20,
                Duration = new Duration(TimeSpan.FromSeconds(0.5)),
                EasingFunction = new QuadraticEase { EasingMode = EasingMode.EaseOut }
            };

            BeginAnimation(TopProperty, animation);
        }
```

```
        public async void ShowAndCloseAfterDelay(int millisecondsDelay)
        {
            Show();
            await Task.Delay(millisecondsDelay);
            CloseNotification();
        }

        private void CloseNotification()
        {
            DoubleAnimation animation = new DoubleAnimation
            {
                From = Top,
                To = SystemParameters.WorkArea.Height,
                Duration = new Duration(TimeSpan.FromSeconds(0.5)),
                EasingFunction = new QuadraticEase { EasingMode = EasingMode.EaseIn }
            };

            animation.Completed += (s, e) => Close();
            BeginAnimation(TopProperty, animation);
        }

        private void CloseButton_Click(object sender, RoutedEventArgs e)
        {
            CloseNotification();
        }
    }

}
```

////////////////////// Folder Data Access//////////////////////

////////////////////// Subfolder Data Providers//////////////////////

////////////////////// DashboardProvider.cs//////////////////////

```
using Kakuro.Enums;
using Kakuro.Interfaces.Data_Access.Data_Providers;
using Kakuro.Models;
using Kakuro.ViewModels;
using System.Collections.ObjectModel;

namespace Kakuro.Data_Access.Data_Providers
{
    public class DashboardProvider : IDashboardProvider
    {
        private IDashboardTemplateProvider _templateProvider;
        private DashboardItemCollection _dashboard;
        private Random _random;

        public DashboardProvider(IDashboardTemplateProvider templateProvider, Dashboard-
ItemCollection dashboard)
        {
            _random ??= new Random();
            _dashboard ??= dashboard;
            _templateProvider ??= templateProvider;
        }

        public void GenerateDashboard(DifficultyLevels difficultyLevel)
        {
            _dashboard.Clear();

            var template = _templateProvider.GenerateTemplate(difficultyLevel);

            CreateDashboard(template);

            CalculateSumsAndValues();
        }

        private void CreateDashboard(string[,] dashboard)
```

```
        {
            int dashboardSize = dashboard.GetLength(0);

            for (int i = 0; i < dashboardSize; i++)
            {
                _dashboard.Add(new ObservableCollection<DashboardItemViewModel>());

                for (int j = 0; j < dashboardSize; j++)
                {
                    var dashboardItem = new DashboardItem();
                    var wrapper = new DashboardItemViewModel(dashboardItem);

                    if (!IsElementOnBorder(i, j, dashboardSize) && dashboard[i, j] == "*")
                        wrapper.CellType = CellType.ValueCell;

                    _dashboard[i].Add(wrapper);
                }
            }
        }

        private bool IsElementOnBorder(int i, int j, int dashboardSize) => i == 0 || j == 0
|| i == dashboardSize - 1 || j == dashboardSize - 1;

        private void CalculateSumsAndValues()
        {
            CalculateBottomSums(true);
            CalculateRightSums();
        }

        private void CalculateBottomSums(bool generateNewValues = false)
        {
            SumCalculatingTemplate(true, generateNewValues);
        }

        private void CalculateRightSums(bool generateNewValues = false)
        {
            SumCalculatingTemplate(false, generateNewValues);
        }

        private void SumCalculatingTemplate(bool isVerticalSum, bool generateNewValues)
        {
            DashboardItemViewModel currentElement;
            int sum = 0;
            bool wasSumCollected = false;

            for (int i = 0; i < _dashboard.Count; i++)
            {
                for (int j = _dashboard.Count - 1; j >= 0; j--)
                {
                    currentElement = isVerticalSum ? _dashboard[j][i] : _dashboard[i][j];

                    if (currentElement.CellType == CellType.ValueCell)
                    {
                        if (generateNewValues)
                        {
                            if (isVerticalSum)
                                GenerateValueTillItsUnique(j, i);
                            else
                                GenerateValueTillItsUnique(i, j);
                        }

                        sum += Convert.ToInt32(currentElement.HiddenValue);
                        wasSumCollected = true;
                    }
                    else if (wasSumCollected)
                    {
```

```
                        if (isVerticalSum)
                            currentElement.SumBottom = sum.ToString();
                        else
                            currentElement.SumRight = sum.ToString();

                        currentElement.CellType = CellType.SumCell;
                        sum = 0;
                        wasSumCollected = false;

                    }
                }
            }
        }

        private void GenerateValueTillItsUnique(int i, int j)
        {
            // We need 1st and 3rd rules of Kakuro for generating:
            // 1.Each cell can contain numbers from 1 through 9
            // 2.The clues in the black cells tells the sum of the numbers next to that clue.
(on the right or down)
            // 3.The numbers in consecutive white cells must be unique.

            // #BAD: in consecutive white cells!!! not only in close cells

            int value = 0;
            bool isUnique;

            do
            {
                isUnique = false;

                value = _random.Next(1, 10);

                isUnique = IsValueUnique(i, j, value);

            } while (!isUnique);

            _dashboard[i][j].HiddenValue = value.ToString();
        }

        private bool IsValueUnique(int i, int j, int value)
        {
            return IsUniqueAbove(i, j, value) && IsUniqueBelow(i, j, value)
                && IsUniqueLeft(i, j, value) && IsUniqueRight(i, j, value);
        }

        private bool IsUniqueAbove(int i, int j, int value) => ConvertDashboardValue-
ToInt(_dashboard[i - 1][j].HiddenValue) != value;

        private bool IsUniqueBelow(int i, int j, int value) => ConvertDashboardValue-
ToInt(_dashboard[i + 1][j].HiddenValue) != value;

        private bool IsUniqueLeft(int i, int j, int value) => ConvertDashboardValue-
ToInt(_dashboard[i][j - 1].HiddenValue) != value;

        private bool IsUniqueRight(int i, int j, int value) => ConvertDashboardValue-
ToInt(_dashboard[i][j - 1].HiddenValue) != value;

        private int ConvertDashboardValueToInt(string value) =>
string.IsNullOrEmpty(value) ? 0 : Convert.ToInt32(value);
    }
}
```

////////////////////// DashboardTemplateProvider.cs//////////////////////

```
using Kakuro.Enums;
using Kakuro.Interfaces.Data_Access.Data_Providers;
```

```
namespace Kakuro.Data_Access.Data_Providers
{
    public class DashboardTemplateProvider : IDashboardTemplateProvider
    {
        private Random _random;

        public DashboardTemplateProvider()
        {
            _random = new Random();
        }

        public string[,] GenerateTemplate(DifficultyLevels difficultyLevel)
        {
            string[,] template = difficultyLevel switch
            {
                DifficultyLevels.Easy => GenerateEasyTemplate(),
                DifficultyLevels.Normal => GenerateNormalTemplate(),
                DifficultyLevels.Hard => GenerateHardTemplate(),
                _ => throw new ArgumentException()
            };

            InvertSymbolsByChance(template); // #BAD: the ways of template being generated
should be documented

            return template;
        }

        // #BAD: i shall add test that checks if borders contain *

        private string[,] GenerateEasyTemplate() => new string[,]
        {
            { "",   "",   "",   "",   "",   "",   "",   "" },
            { "",   "",   "*",  "",   "",   "*",  "",   "" },
            { "",   "*",  "*",  "*",  "*",  "*",  "*",  "" },
            { "",   "",   "*",  "",   "",   "*",  "",   "" },
            { "",   "*",  "*",  "*",  "*",  "*",  "*",  "" },
            { "",   "",   "*",  "*",  "*",  "*",  "",   "" },
            { "",   "",   "",   "*",  "*",  "",   "",   "" },
            { "",   "",   "",   "",   "",   "",   "",   "" }
        };

        private string[,] GenerateNormalTemplate() => new string[,]
        {
            { "",   "",   "",   "",   "",   "",   "",   "",   "",   "",
"",     "" },
            { "",   "*",  "",   "",   "*",  "",   "",   "*",  "",   "",
"*",    "" },
            { "",   "*",  "*",  "",   "*",  "",   "",   "*",  "",   "*",
"*",    "" },
            { "",   "*",  "*",  "*",  "*",  "",   "",   "*",  "*",  "*",
"*",    "" },
            { "",   "",   "*",  "*",  "*",  "",   "",   "*",  "*",  "*",
"",     "" },
            { "",   "*",  "",   "*",  "*",  "*",  "*",  "*",  "*",  "",
"*",    "" },
            { "",   "*",  "*",  "",   "*",  "*",  "*",  "*",  "",   "*",
"*",    "" },
            { "",   "*",  "",   "*",  "*",  "",   "",   "*",  "*",  "",
"*",    "" },
            { "",   "",   "*",  "*",  "",   "",   "",   "",   "*",  "*",
"",     "" },
            { "",   "*",  "*",  "",   "*",  "",   "",   "*",  "",   "*",
"*",    "" },
            { "",   "*",  "",   "",   "",   "*",  "*",  "",   "",   "",
"*",    "" },
```

```
            { "",    "",    "",    "",      "",      "",      "",      "",      "",      "",
"",       "" }
        };

        private string[,] GenerateHardTemplate() => new string[,]
        {
            { "",    "",    "",    "",    "",      "",      "",      "",      "",      "",
"",       "",      "",      "",      "",      "",      "",      "",  },
            { "",    "",    "",    "*",   "*",     "",      "",      "",      "",      "",
"*",      "*",     "*",     "*",     "*",     "*",     "",      "",  },
            { "",    "",    "*",   "*",   "*",     "*",     "",      "",      "",      "*",
"*",      "*",     "",      "",      "",      "*",     "*",     "",  },
            { "",    "",    "",    "*",   "*",     "*",     "*",     "",      "*",     "*",
"*",      "",      "*",     "*",     "*",     "",      "*",     "",  },
            { "",    "",    "",    "",    "*",     "*",     "",      "",      "",      "*",
"",       "*",     "*",     "*",     "*",     "*",     "",      "",  },
            { "",    "",    "",    "*",   "*",     "*",     "*",     "",      "*",     "*",
"",       "*",     "*",     "*",     "*",     "*",     "",      "",  },
            { "",    "",    "*",   "*",   "*",     "",      "*",     "*",     "*",     "*",
"",       "*",     "*",     "*",     "*",     "*",     "",      "",  },
            { "",    "",    "",    "*",   "*",     "",      "",      "",      "*",     "*",
"*",      "",      "*",     "*",     "*",     "",      "*",     "",  },
            { "",    "",    "*",   "*",   "*",     "*",     "*",     "",      "",      "",
"*",      "*",     "",      "",      "",      "*",     "*",     "",  },
            { "",    "*",   "*",   "",    "",      "",      "",      "*",     "*",     "*",     "",
"*",      "*",     "*",     "*",     "*",     "",      "",      "",  },
            { "",    "*",   "",    "*",   "*",     "*",     "",      "",      "*",     "*",     "",
"",       "",      "",      "*",     "*",     "",      "",      "",  },
            { "",    "",    "*",   "*",   "*",     "*",     "*",     "",      "*",     "*",
"*",      "*",     "",      "*",     "*",     "*",     "",      "",  },
            { "",    "",    "*",   "*",   "*",     "*",     "*",     "",      "*",     "*",
"",       "*",     "*",     "*",     "*",     "",      "",      "",  },
            { "",    "*",   "",    "*",   "*",     "*",     "",      "*",     "*",     "*",
"",       "*",     "*",     "*",     "*",     "",      "",      "",  },
            { "",    "*",   "*",   "",    "",      "",      "*",     "*",     "*",     "",
"",       "",      "*",     "*",     "*",     "*",     "",      "",  },
            { "",    "",    "*",   "*",   "*",     "*",     "*",     "*",     "",      "",
"",       "",      "",      "*",     "*",     "",      "",      "",  },
            { "",    "",    "",    "",    "",      "",      "",      "",      "",      "",
"",       "",      "",      "",      "",      "",      "",      "",  }
        };

        private void InvertSymbolsByChance(string[,] template)
        {
            int dashboardSize = template.GetLength(0);

            for (int i = 1; i < dashboardSize - 1; i++)
                for (int j = 1; j < dashboardSize - 1; j++)
                    if (_random.Next(2) == 1)               //probability - 50%
                        template[i, j] = template[i, j] == "*" ? "" : "*";

        }
    }
}
```

////////////////////// RatingRecordProvider.cs//////////////////////

```
using Kakuro.Enums;
using Kakuro.Interfaces.Data_Access.Data_Providers;
using Kakuro.Interfaces.Data_Access.Repositories;
using Kakuro.Models;

namespace Kakuro.Data_Access.Data_Providers
{
    public class RatingRecordProvider : IRatingRecordProvider
    {
```

```
        private IReadAllRepository<RatingRecord, DifficultyLevels> _dataService;
        public Dictionary<DifficultyLevels, bool> IsCacheSynchronizedWithFiles { private
set; get; }
        public Dictionary<DifficultyLevels, IEnumerable<RatingRecord>> Cache { private set;
get; }
        // #BAD: Only for tests I made cache and IsCacheSynchronizedWithFiles public for
reading. But it shouldn't be public any possible way.
        // It should be available only inside the class.
        // #BAD: it's a temporary solution for the cache mechanism. There's an IMemoryCache
and I should use it instead.
        // As an idea to make it create inside of this Data Provider, but to check it with
mocks, we can
        // implement "Fabric method" pattern, in theory. Answer was also given here:
https://qna.habr.com/q/1371574

        public RatingRecordProvider(IReadAllRepository<RatingRecord, DifficultyLevels>
dataService)
        {
            _dataService = dataService;
            Cache = new Dictionary<DifficultyLevels, IEnumerable<RatingRecord>>();
            IsCacheSynchronizedWithFiles = new Dictionary<DifficultyLevels, bool>
            {
                { DifficultyLevels.Easy, true },
                { DifficultyLevels.Normal, true },
                { DifficultyLevels.Hard, true }
            };
        }

        public void Add(RatingRecord entity, DifficultyLevels key)
        {
            _dataService.Add(entity, key);
            IsCacheSynchronizedWithFiles[key] = false;
        }

        public IEnumerable<RatingRecord> GetAll(DifficultyLevels key)
        {
            IEnumerable<RatingRecord>? ratingRecords;

            if (IsCacheSynchronizedWithFiles[key])
            {
                Cache.TryGetValue(key, out ratingRecords);

                ratingRecords ??= _dataService.GetAll(key); // we read data from dataService
ONLY when there's no data in Cache
            }
            else
            {
                ratingRecords = _dataService.GetAll(key);

                Cache[key] = ratingRecords;         // we update cache ONLY when it's not
synchronized with data

                IsCacheSynchronizedWithFiles[key] = true;
            }

            return ratingRecords;
        }
    }
}
```

////////////////////// SavepointProvider.cs//////////////////////

```
using Kakuro.Interfaces.Data_Access.Data_Providers;
using Kakuro.Interfaces.Data_Access.Repositories;
using Kakuro.Models;
using System.Collections.ObjectModel;

namespace Kakuro.Data_Access.Data_Providers
```

```
{
    public class SavepointProvider : ISavepointProvider
    {
        private IRepository<Savepoint> _dataService;
        private const int MAX_CACHE_COUNT = 3;
        public ObservableCollection<Savepoint> Cache { get; private set; }

        // #BAD: Same as in class RatingRecordProvider, Cache is made public for reading, so
as for testing and checking.

        public SavepointProvider(IRepository<Savepoint> dataService)
        {
            _dataService = dataService;
            Cache = new ObservableCollection<Savepoint>();
        }

        public bool Add(Savepoint entity)
        {
            var isSaved = _dataService.Add(entity);

            if (isSaved == false)
                return isSaved;

            AddSavepointToCache(entity);

            return isSaved;
        }

        public Savepoint Delete(int id)
        {
            var deletedEntity = _dataService.Delete(id);

            var cachedEntity = Cache.FirstOrDefault(el => el.Id == id);
            if (cachedEntity != null)
            {
                Cache.Remove(cachedEntity);
            }

            return deletedEntity;
        }

        public Savepoint? GetById(int id)
        {
            var foundSavepoint = Cache.FirstOrDefault(IsIdEqual(id));

            if (foundSavepoint == null)
            {
                foundSavepoint = _dataService.GetById(id);
                AddSavepointToCache(foundSavepoint);
            }

            return foundSavepoint;
        }

        public void Update(Savepoint entity)
        {
            _dataService.Update(entity);

            var cachedSavepoint = Cache.FirstOrDefault(IsIdEqual(entity.Id));

            if (cachedSavepoint != null)
            {
                var index = Cache.IndexOf(cachedSavepoint);
                Cache[index] = entity;
            }
            else
```

```
            {
                var updatedSavepoint = _dataService.GetById(entity.Id);
                AddSavepointToCache(updatedSavepoint);
            }
        }

        private void AddSavepointToCache(Savepoint entity)
        {
            Cache.Add(entity);

            if (Cache.Count > MAX_CACHE_COUNT)
                Cache.RemoveAt(0);
        }

        private Func<Savepoint, bool> IsIdEqual(int id) => el => el.Id == id; // "el" stands
for "element"
    }
}
```

////////////////////// Subfolder Repositories//////////////////////

////////////////////// RatingRecordRepository.cs//////////////////////

```
using Kakuro.Enums;
using Kakuro.Interfaces.Data_Access.Repositories;
using Kakuro.Interfaces.Data_Access.Tools;
using Kakuro.Models;
using System.IO;

namespace Kakuro.Data_Access.Repositories
{
    public class RatingRecordRepository : IReadAllRepository<RatingRecord, Difficul-
tyLevels>
    {
        private const int MAX_COUNT_FOR_EACH_DIFFICULTY = 10;
        private const string PART_OF_FILEPATH = ". Rating Table.json";
        private readonly string _directoryPath;

        private IJsonFileHandler<RatingRecord> _jsonFileHandler;

        public RatingRecordRepository(IJsonFileHandler<RatingRecord> jsonEnumerableFile-
Handler, string directoryPath = "")
        {
            _jsonFileHandler = jsonEnumerableFileHandler;
            _directoryPath = FormDirectorypath(directoryPath);
        }

        public void Add(RatingRecord entity, DifficultyLevels key)
        {
            CheckIfEntityIsEmpty(entity);

            string filepath = FormFilepath(key);
            var ratingTableConcreteDifficulty = GetAll(key).ToList();
            ratingTableConcreteDifficulty.Add(entity);

            SortAndRemoveExcess(ratingTableConcreteDifficulty);

            _jsonFileHandler.Save(ratingTableConcreteDifficulty, filepath);
        }

        public IEnumerable<RatingRecord> GetAll(DifficultyLevels key)
        {
            string filepath = FormFilepath(key);
            return _jsonFileHandler.Load(filepath);
        }

        private string FormFilepath(DifficultyLevels level) => Path.Combine(_directoryPath,
level.ToString() + PART_OF_FILEPATH);
```

```
        private string FormDirectorypath(string directoryPath) =>
string.IsNullOrWhiteSpace(directoryPath) ? "Rating Tables\\" : directoryPath;

        private void SortAndRemoveExcess(List<RatingRecord> ratingRecords)
        {
            ratingRecords.Sort();
            if (IsExceedingMaxCount(ratingRecords.Count))
                ratingRecords.RemoveAt(MAX_COUNT_FOR_EACH_DIFFICULTY);
        }

        private bool IsExceedingMaxCount(int count) => count >
MAX_COUNT_FOR_EACH_DIFFICULTY;

        private void CheckIfEntityIsEmpty(RatingRecord entity)
        {
            if (entity == null)
                throw new NullReferenceException("Entity equals null");
            else if (entity.GameCompletionTime == default || entity.GameCompletionDate ==
default)
                throw new ArgumentException("Entity's properties cannot be null or de-
fault.");
        }
    }
}
```

////////////////////// SavepointRepository.cs//////////////////////

```
using Kakuro.Interfaces.Data_Access.Repositories;
using Kakuro.Interfaces.Data_Access.Tools;
using Kakuro.Models;
using System.IO;

namespace Kakuro.Data_Access.Repositories
{
    public class SavepointRepository : IRepository<Savepoint>
    {
        private const int MAX_COUNT = 10;
        private const string FILENAME = "Savepoints.json";
        private readonly string _directoryPath;       // #BAD: we shall not store data about
filepathes, directories and
                                                      // count of max files in reposi-
tories. Some sort of File Handler is
                                                      // responsible for that
        private readonly string _filepath;

        private IJsonFileHandler<Savepoint> _jsonFileHandler;

        public int Count { private set; get; } = 0;

        public SavepointRepository(IJsonFileHandler<Savepoint> jsonEnumerableFileHandler,
string directoryPath = "")
        {
            _jsonFileHandler = jsonEnumerableFileHandler;
            _directoryPath = directoryPath;
            _filepath = Path.Combine(_directoryPath, FILENAME);
        }

        public bool Add(Savepoint entity)
        {
            if (entity == null)
                throw new NullReferenceException("Entity equals null.");

            var savepoints = _jsonFileHandler.Load(_filepath);

            if (IsInvalidState(savepoints, entity.Id))
                return false;
```

```
            savepoints = savepoints.Append(entity);
            Count++;

            _jsonFileHandler.Save(savepoints, _filepath);
            return true;
        }

        // "el" stands for "element"
        public Savepoint Delete(int id)
        {
            var savepoints = _jsonFileHandler.Load(_filepath);

            if (!IsDuplicateId(savepoints, id))
                throw new IndexOutOfRangeException("Entity with such ID wasn't found.");

            var deletedSavepoint = savepoints.FirstOrDefault(IsIdEqual(id));

            savepoints = savepoints.Where(GetRemoveByIdSelector(id));
            Count--;

            _jsonFileHandler.Save(savepoints, _filepath);

            return deletedSavepoint;
        }

        public Savepoint? GetById(int id)
        {
            var savepoints = _jsonFileHandler.Load(_filepath);

            var foundSavepoint = savepoints.FirstOrDefault(IsIdEqual(id));

            if (foundSavepoint == null)
                throw new IndexOutOfRangeException("Entity with such ID doesn't exist.");
            else
                return foundSavepoint;
        }

        public void Update(Savepoint entity)
        {
            var savepoints = _jsonFileHandler.Load(_filepath);

            CheckIfUnableToUpdate(savepoints, entity);

            var existingSavepoint = savepoints.FirstOrDefault(IsIdEqual(entity.Id));
            if (existingSavepoint.Equals(entity))
                return;      // i don't throw exception, because in user's opinion everything
is okay and an update is done. Business logic, lmao

            savepoints = savepoints.Select(GetUpdatedSavepointSelector(entity.Id, enti-
ty));

            _jsonFileHandler.Save(savepoints, _filepath);
        }

        private void CheckIfUnableToUpdate(IEnumerable<Savepoint> savepoints, Savepoint
entity)
        {
            if (entity == null)
                throw new NullReferenceException("Entity equals null.");
            if (!IsDuplicateId(savepoints, entity.Id))
                throw new IndexOutOfRangeException("Entity with such ID wasn't found.");
        }

        private bool IsInvalidState(IEnumerable<Savepoint> savepoints, int id)
        {
            if (IsMaxCountReached(savepoints))
```

```
                return true;
            if (IsDuplicateId(savepoints, id))
                throw new ArgumentException("Entity with such ID already exists!");
            return false;
        }

        private bool IsMaxCountReached(IEnumerable<Savepoint> savepoints) => save-
points.Count() == MAX_COUNT;

        private bool IsDuplicateId(IEnumerable<Savepoint> savepoints, int id) => save-
points.Any(IsIdEqual(id));

        private Func<Savepoint, bool> IsIdEqual(int id) => el => el.Id == id; // "el" stands
for "element"

        private Func<Savepoint, bool> GetRemoveByIdSelector(int id) => el
=> !IsIdEqual(id)(el);

        private Func<Savepoint, Savepoint> GetUpdatedSavepointSelector(int id, Savepoint
newEntity) => el => IsIdEqual(id)(el) ? newEntity : el;

    }
}
```

////////////////////// Subfolder Tools//////////////////////

////////////////////// JsonFileHandler.cs//////////////////////

```
using Kakuro.Interfaces.Data_Access.Tools;
using System.IO;
using System.Text.Json;

namespace Kakuro.Data_Access.Tools
{
    public class JsonFileHandler<T> : IJsonFileHandler<T>
    {
        // #BAD: We should STORE filepathes here, i guess. So we could keep "Fabric method"
pattern implementation.
        // I've mentioned it in IFilesHandler.cs. We shall not pass filepath straightfully

        public IEnumerable<T> Load(string filepath)
        {
            if (IsInvalidFile(filepath))
                return new List<T>();

            var jsonData = File.ReadAllText(filepath);
            List<T>? deserializedData;

            try
            {
                deserializedData = JsonSerializer.Deserialize<List<T>>(jsonData);
            }
            catch (JsonException)
            {
                return new List<T>();
            }

            return deserializedData ?? new List<T>();
        }

        public void Save(IEnumerable<T> data, string filepath)
        {
            if (AreInvalidSaveParameters(data, filepath))
                throw new ArgumentException("Invalid data or filepath");

            EnsureDirectoryExists(filepath);

            var jsonData = JsonSerializer.Serialize(data, new JsonSerializerOptions
```

```
            {
                WriteIndented = true
            });

            File.WriteAllText(filepath, jsonData);
        }

        private void EnsureDirectoryExists(string filepath)
        {
            var directory = Path.GetDirectoryName(filepath);
            if (IsDirectoryNeeded(directory))
                Directory.CreateDirectory(directory);
        }

        private bool IsEmptyPath(string filepath) => string.IsNullOrEmpty(filepath);

        private bool IsInvalidFile(string filepath) => IsEmptyPath(filepath)
|| !File.Exists(filepath);
        private bool IsDirectoryNeeded(string directory) => !IsEmptyPath(directory)
&& !Directory.Exists(directory);

        private bool AreInvalidSaveParameters(IEnumerable<T> data, string filepath) => data
== null || IsEmptyPath(filepath);
    }
}
```

////////////////////// Folder Enums//////////////////////

////////////////////// CellType.cs//////////////////////

```
namespace Kakuro.Enums
{
    public enum CellType
    {
        EmptyCell,  // for those that neither contain value nor any sum; a part of border
                    // for example.
        ValueCell, // cell for containing user's value
        SumCell   // cell for containing sums: only below, only right or even both
    }
}
```

////////////////////// DifficultyLevels.cs//////////////////////

```
namespace Kakuro.Enums
{
    // #BAD: shall i write tests for enum?
    public enum DifficultyLevels
    {
        Easy = 6,
        Normal = 10,
        Hard = 16
    }
}
```

////////////////////// SettingType.cs//////////////////////

```
using System.Runtime.Serialization;

namespace Kakuro.Enums
{
    public enum SettingType
    {
        [EnumMember(Value = "Show correct answers (+1 minute, post-cleaning)")]
        ShowCorrectAnswers,
        [EnumMember(Value = "Auto submit")]
        AutoSubmit,
        [EnumMember(Value = "Hide the timer")]
        HideTimer
    }
}
```

////////////////////// Folder Events//////////////////////

////////////////////// CorrectAnswersTurnedOnEvent.cs//////////////////////
```
namespace Kakuro.Events
{
    public class CorrectAnswersTurnedOnEvent : PubSubEvent<object>
    {
    }
}
```
////////////////////// GameCompletedEvent.cs//////////////////////
```
using Kakuro.Models;

namespace Kakuro.Events
{
    // #BAD: there shall be tests for using Event Aggregator
    public class GameCompletedEvent : PubSubEvent<GameSession>
    {
    }
}
```
////////////////////// NewGameStartedEvent.cs//////////////////////
```
namespace Kakuro.Events
{
    public class NewGameStartedEvent : PubSubEvent<object>
    {
    }
}
```
////////////////////// SettingsChangedEvent.cs//////////////////////
```
using Kakuro.ViewModels;
using System.Collections.ObjectModel;

namespace Kakuro.Events
{
    // #BAD: there shall be tests for using Event Aggregator
    public class SettingsChangedEvent : PubSubEvent<ObservableCollection<SettingViewModel>>
    {
    }
}
```
////////////////////// Folder Game Tools//////////////////////
////////////////////// FocusNavigationHelper.cs//////////////////////
```
using System.Windows;
using System.Windows.Input;

namespace Kakuro.Game_Tools
{
    public static class FocusNavigationHelper
    {
        public static void HandleWASDKeyDown(UIElement currentElement, KeyEventArgs e)
        {
            if (!IsWASDKey(e.Key)) return;

            TraversalRequest request = GetTraversalRequest(e.Key);
            if (request != null)
            {
                currentElement.MoveFocus(request);
                e.Handled = true;
            }
        }

        private static TraversalRequest GetTraversalRequest(Key key)
        {
            return key switch
            {
                Key.W => new TraversalRequest(FocusNavigationDirection.Up),
                Key.A => new TraversalRequest(FocusNavigationDirection.Left),
                Key.S => new TraversalRequest(FocusNavigationDirection.Down),
```

```
                Key.D => new TraversalRequest(FocusNavigationDirection.Right),
                _ => throw new ArgumentException()
            };
        }

        private static bool IsWASDKey(Key key) => key == Key.W || key == Key.A || key == Key.S
|| key == Key.D;
    }

}
```

////////////////////// OperationNotifier.cs//////////////////////

```
using Kakuro.Custom_Components;
using Kakuro.Interfaces.Game_Tools;

namespace Kakuro.Game_Tools
{
    // #BAD: tests shall be written
    public class OperationNotifier : IOperationNotifier
    {
        private readonly List<ToastNotificationWindow> _openNotifications;

        public OperationNotifier()
        {
            _openNotifications = new List<ToastNotificationWindow>();
        }

        public void NotifySuccess(string message = "Operation completed successfully.")
        {
            ShowToastNotification(message, "Success", true);
        }

        public void NotifyFailure(string message = "Operation failed.")
        {
            ShowToastNotification(message, "Failure", false);
        }

        public void NotifyOutcome(bool isSuccess,
            string successMessage = "Operation completed successfully.",
            string failMessage = "Operation failed.")
        {
            string caption = isSuccess ? "Success" : "Failure";
            string message = isSuccess ? successMessage : failMessage;
            ShowToastNotification(message, caption, isSuccess);
        }

        private void ShowToastNotification(string message, string caption, bool isSuccess)
        {
            var toast = new ToastNotificationWindow(message, caption, isSuccess);
            toast.ShowAndCloseAfterDelay(10000);
            _openNotifications.Add(toast);
        }

        public void CloseAllNotifications()
        {
            foreach (var notification in _openNotifications.ToList())
                notification.Close();

            _openNotifications.Clear();
        }
    }

}
```

////////////////////// SolutionValidator.cs//////////////////////

```
using Kakuro.Enums;
using Kakuro.Interfaces.Game_Tools;
```

```
using Kakuro.ViewModels;

namespace Kakuro.Game_Tools
{
    // #BAD: tests shall be written
    public class SolutionValidator : ISolutionValidator
    {
        private DashboardItemCollection _dashboard;

        public SolutionValidator(DashboardItemCollection dashboard)
        {
            _dashboard = dashboard;
        }
        public bool ValidateDashboard(out string message)
        {
            if (!ValidateSums(out message))
                return false;

            message = "Game is completed!"; // #BAD: i need to make the font of message in
MessageBox - bigger
            return true;
        }

        // #BAD: we need to make sure that we bind not to the field values of class,
        // but to the values in the text fields themselves

        private bool ValidateSums(out string message)
        {
            bool areVerticalSumsCorrect = ValidateConcreteSum(true, out message, true);

            if (!areVerticalSumsCorrect)
                return false;

            bool areHorizontalSumsCorrect = ValidateConcreteSum(false, out message);

            if (!areHorizontalSumsCorrect)
                return false;

            return true;
        }

        private bool ValidateConcreteSum(bool isVerticalSum, out string message, bool
shouldCheckValues = false)
        {
            for (int i = 0; i < _dashboard.Count; i++)
            {
                int calculatedSum = 0;
                bool wasSumCollected = false;

                for (int j = _dashboard.Count - 1; j >= 0; j--)
                {
                    var currentElement = isVerticalSum ? _dashboard[j][i] :
_dashboard[i][j];

                    if (currentElement.CellType == CellType.ValueCell)
                    {
                        if (shouldCheckValues)
                        {
                            if (!CheckValueForCorrectness(i, j, currentElement, isVerti-
calSum, out message))
                                return false;
                        }

                        calculatedSum += Convert.ToInt32(currentElement.DisplayValue);
                        wasSumCollected = true;
                    }
```

```
                    else if (wasSumCollected)
                    {
                        var sumToCheck = isVerticalSum ? Con-
vert.ToInt32(currentElement.SumBottom) : Convert.ToInt32(currentElement.SumRight);

                        if (sumToCheck != calculatedSum)
                        {
                            message = "The sum of the numbers is not equal to the given sum!";
                            return false;
                        }

                        calculatedSum = 0;
                        wasSumCollected = false;
                    }
                }
            }

            message = "The sum of the numbers is equal to the given sum!";
            return true;
        }

        private bool CheckValueForCorrectness(int i, int j, DashboardItemViewModel cur-
rentElement, bool isVerticalSum, out string message)
        {
            if (!CheckValueForEmptiness(currentElement, out message))
                return false;

            int tempI, tempJ;

            if (isVerticalSum)
            {
                tempI = j;
                tempJ = i;
            }
            else
            {
                tempI = i;
                tempJ = j;
            }

            if (!CheckValueForUniqueness(tempI, tempJ, currentElement.DisplayValue, out
message))
                return false;

            return true;
        }

        private bool CheckValueForEmptiness(DashboardItemViewModel cell, out string message)
        {
            if (cell.CellType == CellType.ValueCell && cell.DisplayValue == "")
            {
                message = "Dashboard isn't filled completely!";
                return false;
            }
            message = "Dashboard is filled completely!";
            return true;
        }

        private bool CheckValueForUniqueness(int i, int j, string value, out string message)
        {
            if (!(IsUniqueAbove(i, j, value) && IsUniqueBelow(i, j, value)
                && IsUniqueLeft(i, j, value) && IsUniqueRight(i, j, value)))
            {
                message = "Entered values aren't unique!";
                return false;
            }
```

```
            message = "Entered values are unique!";
            return true;
        }

        private bool IsUniqueAbove(int i, int j, string value)
        {
            string valueAbove = _dashboard[i - 1][j].DisplayValue;
            return valueAbove != value;
        }

        private bool IsUniqueBelow(int i, int j, string value)
        {
            string valueBelow = _dashboard[i + 1][j].DisplayValue;
            return valueBelow != value;
        }

        private bool IsUniqueLeft(int i, int j, string value)
        {
            string valueLeft = _dashboard[i][j - 1].DisplayValue;
            return valueLeft != value;
        }

        private bool IsUniqueRight(int i, int j, string value)
        {
            string valueRight = _dashboard[i][j + 1].DisplayValue;
            return valueRight != value;
        }
    }
}
```

////////////////////// Folder Interfaces//////////////////////

////////////////////// Subfolder Data Access//////////////////////

////////////////////// 2xSubfolder Data Providers//////////////////////

////////////////////// IDashboardProvider.cs//////////////////////

```
using Kakuro.Enums;

namespace Kakuro.Interfaces.Data_Access.Data_Providers
{
    public interface IDashboardProvider
    {
        void GenerateDashboard(DifficultyLevels difficultyLevel);
    }
}
```

////////////////////// IDashboardTemplateProvider.cs//////////////////////

```
using Kakuro.Enums;

namespace Kakuro.Interfaces.Data_Access.Data_Providers
{
    public interface IDashboardTemplateProvider
    {
        string[,] GenerateTemplate(DifficultyLevels difficultyLevel);
    }
}
```

////////////////////// IRatingRecordProvider.cs//////////////////////

```
using Kakuro.Enums;
using Kakuro.Interfaces.Data_Access.Repositories;
using Kakuro.Models;

namespace Kakuro.Interfaces.Data_Access.Data_Providers
{
    public interface IRatingRecordProvider : IReadAllRepository<RatingRecord, Difficul-
tyLevels> { }
}
```

////////////////////// ISavepointProvider.cs//////////////////////
```
using Kakuro.Interfaces.Data_Access.Repositories;
using Kakuro.Models;

namespace Kakuro.Interfaces.Data_Access.Data_Providers
{
    // #BAD: the idea for creating interfaces ISavepointProvider and IRatingRecordProvider
was not to have same interfaces for
    // Repositories and Data Providers, because they are in different system layouts and
because I didn't think that it'd be
    // good for DI. I dunno if it's a good practice.
    // I should remember about it
    public interface ISavepointProvider : IRepository<Savepoint> { }
}
```
////////////////////// 2xSubfolder Repositories//////////////////////
////////////////////// IReadAllRepository.cs//////////////////////
```
namespace Kakuro.Interfaces.Data_Access.Repositories
{
    // #BAD: we have two 2 interfaces for, yep, 2 repositories with not very different roles..
maybe...
    // I dunno. It causes some worries. I should remember about this place.
    public interface IReadAllRepository<T, F>
    {
        void Add(T entity, F key);
        IEnumerable<T> GetAll(F key);
    }
}
```
////////////////////// IRepository.cs//////////////////////
```
namespace Kakuro.Interfaces.Data_Access.Repositories
{
    public interface IRepository<T>
    {
        T? GetById(int id);
        bool Add(T entity);
        void Update(T entity);
        T Delete(int id);
    }
}
```
////////////////////// 2xSubfolder Tools//////////////////////
////////////////////// IFilesHandler.cs//////////////////////
```
namespace Kakuro.Interfaces.Data_Access.Tools
{
    public interface IFilesHandler<T, F> // #BAD: we should have the general interface for
all types of communication
                                         // with data: though JSON, through CSV, through DB,
e.t.c. Now it's only for files
                                         // So we shouldn't put filepath straightfully in
parameters.
    {
        void Save(T data, string filepath);
        T Load(string filepath);
    }
}
```
////////////////////// IJsonFileHandler.cs//////////////////////
```
namespace Kakuro.Interfaces.Data_Access.Tools
{

    // #BAD: unnecessary interface. As for "Fabric method" pattern we could implement
IDataHandler interface straightfully.
    // This note is connected with the note in IFilesHandler.cs.

    public interface IJsonFileHandler<T> : IFilesHandler<IEnumerable<T>, string>
    {
```

```
        new void Save(IEnumerable<T> data, string filepath);
        new IEnumerable<T> Load(string filepath);
    }
}
```

////////////////////// Subfolder Game Tools//////////////////////

////////////////////// IOperationNotifier.cs//////////////////////

```
namespace Kakuro.Interfaces.Game_Tools
{
    public interface IOperationNotifier
    {
        void NotifySuccess(string message = "Operation completed successfully.");
        void NotifyFailure(string message = "Operation failed.");
        void NotifyOutcome(bool isSuccess,
            string successMessage = "Operation completed successfully.",
            string failMessage = "Operation failed.");
        void CloseAllNotifications();
    }
}
```

////////////////////// ISolutionValidator.cs//////////////////////

```
namespace Kakuro.Interfaces.Game_Tools
{
    public interface ISolutionValidator
    {
        bool ValidateDashboard(out string message);
    }
}
```

////////////////////// Folder Models//////////////////////

////////////////////// DashboardItem.cs//////////////////////

```
using Kakuro.Enums;
using System.Text.Json.Serialization;

namespace Kakuro.Models
{
    public class DashboardItem
    {
        private static int _countOfIds = 1;

        public int? ID { get; set; }
        public int? DisplayValue { get; set; }
        public int? HiddenValue { get; set; } // we need this so we could show right answers
using settings
        public CellType CellType { get; set; }
        public int? SumRight { get; set; }
        public int? SumBottom { get; set; }

        public DashboardItem()
        {
            ID = _countOfIds;
            _countOfIds++;
        }

        [JsonConstructor]
        public DashboardItem(int? id, int? displayValue, int? hiddenValue, CellType cellType,
int? sumRight, int? sumBottom)
        {
            ID = id;
            DisplayValue = displayValue;
            HiddenValue = hiddenValue;
            CellType = cellType;
            SumRight = sumRight;
            SumBottom = sumBottom;
        }

        // #BAD: i shall write tests for this model
```

```
        public override bool Equals(object? obj)
        {
            if (obj is DashboardItem other)
            {
                return ID == other.ID &&
                       DisplayValue == other.DisplayValue &&
                       HiddenValue == other.HiddenValue &&
                       CellType == other.CellType &&
                       SumRight == other.SumRight &&
                       SumBottom == other.SumBottom;
            }

            return false;
        }

        public override int GetHashCode()
        {
            return (ID?.GetHashCode() ?? 0) ^
                   (DisplayValue?.GetHashCode() ?? 0) ^
                   (HiddenValue?.GetHashCode() ?? 0) ^
                   (SumRight?.GetHashCode() ?? 0) ^
                   (SumBottom?.GetHashCode() ?? 0) ^
                   CellType.GetHashCode();
        }
    }
}
```

////////////////////// GameSession.cs//////////////////////

```
using Kakuro.Enums;

namespace Kakuro.Models
{
    public class GameSession
    {
        public DifficultyLevels Difficulty { get; }
        public string StopWatchHours { get; }
        public string StopWatchMinutes { get; }
        public string StopWatchSeconds { get; }

        public GameSession(DifficultyLevels difficulty, string hours, string minutes, string
seconds)
        {
            Difficulty = difficulty;
            StopWatchHours = hours;
            StopWatchMinutes = minutes;
            StopWatchSeconds = seconds;
        }

        public void Deconstruct(out DifficultyLevels difficulty, out string hours, out string
minutes, out string seconds)
        {
            difficulty = Difficulty;
            hours = StopWatchHours;
            minutes = StopWatchMinutes;
            seconds = StopWatchSeconds;
        }
    }
}
```

////////////////////// MyStopwatch.cs//////////////////////

```
using System.Diagnostics;

namespace Kakuro.Models
{
    public class MyStopwatch : Stopwatch
    {
        public TimeSpan StartOffset { get; private set; }
```

```
        public MyStopwatch(TimeSpan startOffset)
        {
            StartOffset = startOffset;
        }

        public TimeSpan TotalElapsed => Elapsed + StartOffset;

        public int ElapsedHours => TotalElapsed.Hours;

        public int ElapsedMinutes => TotalElapsed.Minutes;

        public int ElapsedSeconds => TotalElapsed.Seconds;

        public void AddTime(TimeSpan timeToAdd)
        {
            StartOffset += timeToAdd;
        }

        public void CleanAdditionalTime()
        {
            StartOffset = new TimeSpan();
        }
    }
}
```

////////////////////// RatingRecord.cs//////////////////////

```
using System.Text.Json.Serialization;

namespace Kakuro.Models
{
    public class RatingRecord : IComparable<RatingRecord>
    {
        public TimeOnly GameCompletionTime { get; set; }
        public DateOnly GameCompletionDate { get; set; }

        public string FormattedCompletionTime => GameCompletion-
Time.ToString("HH\\:mm\\:ss");

        public RatingRecord(int hour, int minute, int second)
        {
            GameCompletionTime = new TimeOnly(hour, minute, second);
            GameCompletionDate = DateOnly.FromDateTime(DateTime.Now);
        }

        [JsonConstructor]
        public RatingRecord(TimeOnly gameCompletionTime, DateOnly gameCompletionDate)
        {
            GameCompletionTime = gameCompletionTime;
            GameCompletionDate = gameCompletionDate;
        }

        public int CompareTo(RatingRecord? other)
        {
            if (other == null)
                throw new ArgumentNullException("", "While sorting rating records, one was
passed as null.");

            return GameCompletionTime.CompareTo(other.GameCompletionTime);
        }
    }
}
```

////////////////////// Savepoint.cs//////////////////////

```
using System.Text.Json.Serialization;

namespace Kakuro.Models
```

```
{
    public class Savepoint
    {
        public int Id { get; set; }
        public DashboardItemCollection Dashboard { get; set; }

        [JsonConstructor]
        public Savepoint(int id, DashboardItemCollection dashboard)
        {
            Id = id;
            Dashboard = dashboard;
        }

        public override bool Equals(object? obj)
        {
            if (obj is Savepoint other)
            {
                return Id == other.Id &&
                       Dashboard.SequenceEqual(other.Dashboard);
            }
            return false;
        }

        public override int GetHashCode()
        {
            int hash = 17;
            hash = hash * 31 + Id.GetHashCode();
            hash = hash * 31 + (Dashboard != null ? Dashboard.GetHashCode() : 0);
            return hash;
        }
    }
}
```

////////////////////// Setting.cs//////////////////////

```
using Kakuro.Enums;

namespace Kakuro.Models
{
    public class Setting
    {
        public SettingType SettingType { get; set; }
        public bool IsEnabled { get; set; }

        public Setting(SettingType settingType, bool isEnabled)
        {
            SettingType = settingType;
            IsEnabled = isEnabled;
        }
    }
}
```

////////////////////// Folder StartUp//////////////////////

////////////////////// BootStrapper.cs//////////////////////

```
using Autofac;
using Kakuro.Commands.DashboardViewModel;
using Kakuro.Data_Access.Data_Providers;
using Kakuro.Data_Access.Repositories;
using Kakuro.Data_Access.Tools;
using Kakuro.Enums;
using Kakuro.Game_Tools;
using Kakuro.Interfaces.Data_Access.Data_Providers;
using Kakuro.Interfaces.Data_Access.Repositories;
using Kakuro.Interfaces.Data_Access.Tools;
using Kakuro.Interfaces.Game_Tools;
using Kakuro.Models;
using Kakuro.ViewModels;
```

```
namespace Kakuro.StartUp
{
    public class BootStrapper
    {
        public IContainer BootStrap()
        {
            var builder = new ContainerBuilder();

            builder.RegisterType<MainWindow>().AsSelf().SingleInstance();

            builder.Register(c => new DashboardItemCollection())
                   .AsSelf()
                   .SingleInstance();

            build-
er.RegisterType<EventAggregator>().As<IEventAggregator>().SingleInstance();

            // ViewModels
            builder.RegisterType<MainViewModel>().AsSelf().SingleInstance();
            builder.RegisterType<DashboardItemViewModel>().AsSelf();
            builder.RegisterType<DashboardViewModel>().AsSelf().SingleInstance();
            builder.RegisterType<RatingTableViewModel>().AsSelf().SingleInstance();
            builder.RegisterType<SettingsViewModel>().AsSelf().SingleInstance();
            builder.RegisterType<SavepointsViewModel>().AsSelf().SingleInstance();

            // Commands
            builder.RegisterType<CleanDashboardCommand>().AsSelf().SingleInstance();
            builder.RegisterType<ValidateSolutionCommand>().AsSelf().SingleInstance();

            // Data Access
            build-
er.RegisterType<DashboardProvider>().As<IDashboardProvider>().SingleInstance();
            build-
er.RegisterType<DashboardTemplateProvider>().As<IDashboardTemplateProvider>().SingleInsta
nce();
            build-
er.RegisterType<RatingRecordProvider>().As<IRatingRecordProvider>().SingleInstance();
            build-
er.RegisterType<SavepointProvider>().As<ISavepointProvider>().SingleInstance();
            build-
er.RegisterType<RatingRecordRepository>().As<IReadAllRepository<RatingRecord, Difficul-
tyLevels>>().SingleInstance();
            build-
er.RegisterType<SavepointRepository>().As<IRepository<Savepoint>>().SingleInstance();
            build-
er.RegisterType<JsonFileHandler<RatingRecord>>().As<IJsonFileHandler<RatingRecord>>().Sin
gleInstance();
            build-
er.RegisterType<JsonFileHandler<Savepoint>>().As<IJsonFileHandler<Savepoint>>().SingleIns
tance();

            // Game Tools:
            build-
er.RegisterType<SolutionValidator>().As<ISolutionValidator>().SingleInstance();
            build-
er.RegisterType<OperationNotifier>().As<IOperationNotifier>().SingleInstance();


            return builder.Build();
        }
    }
}
```

////////////////////// Folder Styles//////////////////////

////////////////////// Button.xaml//////////////////////

```
<ResourceDictionary xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
```

```
                    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">

    <Style TargetType="Button">
        <Setter Property="FontSize" Value="16"/>
        <Setter Property="Background" Value="LightGray"/>
        <Setter Property="BorderBrush" Value="Gray"/>
        <Setter Property="Template">
            <Setter.Value>
                <ControlTemplate TargetType="Button">
                    <Border Background="{TemplateBinding Background}"
                        BorderBrush="{TemplateBinding BorderBrush}"
                        BorderThickness="{TemplateBinding BorderThickness}"
                        CornerRadius="5">
                        <ContentPresenter HorizontalAlignment="Center"
                                    VerticalAlignment="Center"/>
                    </Border>
                </ControlTemplate>
            </Setter.Value>
        </Setter>

        <Style.Triggers>
            <Trigger Property="IsEnabled" Value="False">
                <Setter Property="Opacity" Value="0.6"/>
                <Setter Property="Background" Value="Gray"/>
                <Setter Property="BorderBrush" Value="DarkGray"/>
                <Setter Property="Foreground" Value="Black"/>
            </Trigger>

            <Trigger Property="IsMouseOver" Value="True">
                <Setter Property="Background" Value="LightBlue"/>
                <Setter Property="BorderBrush" Value="Blue"/>
                <Setter Property="Foreground" Value="Black"/>
            </Trigger>

            <Trigger Property="IsPressed" Value="True">
                <Setter Property="Background" Value="DarkBlue"/>
                <Setter Property="BorderBrush" Value="Navy"/>
                <Setter Property="Foreground" Value="White"/>
            </Trigger>
        </Style.Triggers>
    </Style>

</ResourceDictionary>
```

////////////////////// Elements_Template.xaml//////////////////////

```
<ResourceDictionary xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
                    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
                    xmlns:converters="clr-namespace:Kakuro.Converters"
                    xmlns:i="http://schemas.microsoft.com/xaml/behaviors"
                    x:Class="Kakuro.Styles.Elements_Template"
                    x:ClassModifier="public">

    <converters:ReadOnlyMultiConverter x:Key="ReadOnlyMultiConverter"/>

    <DataTemplate x:Key="Elements_Template">
        <TextBox Height="40" Width="40" HorizontalAlignment="Center" VerticalAlign-
ment="Center" TextAlignment="Center"
     Style="{DynamicResource CustomTextBoxStyle}" KeyDown="TextBox_KeyDown">
            <TextBox.IsReadOnly>
                <MultiBinding Converter="{StaticResource ReadOnlyMultiConverter}">
                    <Binding Path="DataContext.IsGameCompleted" Rela-
tiveSource="{RelativeSource AncestorType=UserControl}"/>
                    <Binding Path="DataContext.ShowCorrectAnswers" Rela-
tiveSource="{RelativeSource AncestorType=UserControl}"/>
                    <Binding Path="CellType" />
                </MultiBinding>
            </TextBox.IsReadOnly>
```

```
            <i:Interaction.Triggers>
                <i:EventTrigger EventName="TextChanged">
                    <i:InvokeCommandAction Command="{Binding DataContext.AutoSubmitCommand,
RelativeSource={RelativeSource AncestorType=UserControl}}"
                                           CommandParameter="{Binding CellType}" />
                </i:EventTrigger>
            </i:Interaction.Triggers>
        </TextBox>
    </DataTemplate>
</ResourceDictionary>
```

////////////////////// Elements_Template.xaml.cs//////////////////////

```
using Kakuro.Game_Tools;
using System.Windows;
using System.Windows.Input;

namespace Kakuro.Styles
{
    partial class Elements_Template : ResourceDictionary
    {
        public Elements_Template()
        {
            InitializeComponent();
        }

        private void TextBox_KeyDown(object sender, KeyEventArgs e)
        {
            FocusNavigationHelper.HandleWASDKeyDown((UIElement)sender, e);
        }
    }
}
```

////////////////////// NotificationStyles.xaml//////////////////////

```
<ResourceDictionary xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
                    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">

    <Style TargetType="Border" x:Key="NotificationBorderStyle">
        <Setter Property="CornerRadius" Value="10"/>
        <Setter Property="Padding" Value="10"/>
    </Style>

    <Style TargetType="TextBlock" x:Key="NotificationMessageStyle">
        <Setter Property="FontSize" Value="18"/>
        <Setter Property="Foreground" Value="White"/>
        <Setter Property="TextWrapping" Value="Wrap"/>
        <Setter Property="VerticalAlignment" Value="Center"/>
        <Setter Property="Margin" Value="0,0,10,0"/>
    </Style>

    <Style TargetType="Button" x:Key="CloseButtonStyle">
        <Setter Property="FontSize" Value="40"/>
        <Setter Property="Foreground" Value="White"/>
        <Setter Property="Background" Value="Transparent"/>
        <Setter Property="BorderBrush" Value="Transparent"/>
        <Setter Property="Cursor" Value="Hand"/>
        <Setter Property="FocusVisualStyle" Value="{x:Null}"/>

        <Setter Property="Template">
            <Setter.Value>
                <ControlTemplate TargetType="Button">
                    <ContentPresenter HorizontalAlignment="Center" VerticalAlign-
ment="Center"/>
                </ControlTemplate>
            </Setter.Value>
        </Setter>

        <Style.Triggers>
```

```
            <Trigger Property="IsMouseOver" Value="True">
                <Setter Property="Background" Value="Transparent"/>
            </Trigger>
            <Trigger Property="IsPressed" Value="True">
                <Setter Property="Background" Value="Transparent"/>
            </Trigger>
            <Trigger Property="IsFocused" Value="True">
                <Setter Property="Background" Value="Transparent"/>
            </Trigger>
        </Style.Triggers>
    </Style>
</ResourceDictionary>
```

////////////////////// RadioButton.xaml//////////////////////

```
<ResourceDictionary xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
                    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">

    <Style TargetType="RadioButton">
        <Setter Property="Background" Value="LightGray" />
        <Setter Property="Template">
            <Setter.Value>
                <ControlTemplate TargetType="RadioButton">
                    <Border x:Name="border" Background="{TemplateBinding Background}"
                    BorderBrush="{TemplateBinding BorderBrush}"
                    BorderThickness="{TemplateBinding BorderThickness}"
                    CornerRadius="5">
                        <ContentPresenter HorizontalAlignment="Center" VerticalAlign-
ment="Center"/>
                    </Border>
                    <ControlTemplate.Triggers>
                        <Trigger Property="IsChecked" Value="True">
                            <Setter TargetName="border" Property="Background" Val-
ue="LightBlue"/>
                            <Setter TargetName="border" Property="BorderBrush" Val-
ue="Blue"/>
                        </Trigger>
                        <Trigger Property="IsMouseOver" Value="True">
                            <Setter TargetName="border" Property="Background" Val-
ue="LightSkyBlue"/>
                        </Trigger>
                        <Trigger Property="IsEnabled" Value="False">
                            <Setter Property="Opacity" Value="0.6"/>
                            <Setter Property="Background" Value="Gray"/>
                            <Setter Property="BorderBrush" Value="DarkGray"/>
                            <Setter Property="Foreground" Value="Black"/>
                        </Trigger>
                    </ControlTemplate.Triggers>
                </ControlTemplate>
            </Setter.Value>
        </Setter>
    </Style>

</ResourceDictionary>
```

////////////////////// RatingDataGridTemplate.xaml//////////////////////

```
<ResourceDictionary xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
                    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">

    <DataTemplate x:Key="RatingDataGridTemplate">
        <DataGrid Style="{StaticResource RatingDataGridStyle}"
                  ItemsSource="{Binding}">
            <DataGrid.Columns>
                <DataGridTextColumn Header="Time" Binding="{Binding FormattedCompletion-
Time}" Width="*">
                    <DataGridTextColumn.ElementStyle>
                        <Style TargetType="TextBlock">
                            <Setter Property="HorizontalAlignment" Value="Center"/>
```

```
                            <Setter Property="VerticalAlignment" Value="Center"/>
                        </Style>
                    </DataGridTextColumn.ElementStyle>
                </DataGridTextColumn>
                <DataGridTextColumn Header="Date" Binding="{Binding GameCompletionDate}"
Width="*">
                    <DataGridTextColumn.ElementStyle>
                        <Style TargetType="TextBlock">
                            <Setter Property="HorizontalAlignment" Value="Center"/>
                            <Setter Property="VerticalAlignment" Value="Center"/>
                        </Style>
                    </DataGridTextColumn.ElementStyle>
                </DataGridTextColumn>
            </DataGrid.Columns>
        </DataGrid>
    </DataTemplate>

</ResourceDictionary>
```

////////////////////// RatingTableStyles.xaml//////////////////////

```
<ResourceDictionary xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
                    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">

    <Style x:Key="RatingDataGridStyle" TargetType="DataGrid">
        <Setter Property="AutoGenerateColumns" Value="False"/>
        <Setter Property="HeadersVisibility" Value="Column"/>
        <Setter Property="CanUserAddRows" Value="False"/>
        <Setter Property="CanUserDeleteRows" Value="False"/>
        <Setter Property="CanUserResizeRows" Value="False"/>
        <Setter Property="IsReadOnly" Value="True"/>
        <Setter Property="RowHeight" Value="30"/>
        <Setter Property="ColumnHeaderHeight" Value="30"/>
        <Setter Property="ColumnHeaderStyle">
            <Setter.Value>
                <Style TargetType="DataGridColumnHeader">
                    <Setter Property="HorizontalContentAlignment" Value="Center"/>
                </Style>
            </Setter.Value>
        </Setter>
        <Setter Property="CellStyle">
            <Setter.Value>
                <Style TargetType="DataGridCell">
                    <Setter Property="HorizontalContentAlignment" Value="Center"/>
                    <Setter Property="VerticalContentAlignment" Value="Center"/>
                </Style>
            </Setter.Value>
        </Setter>
    </Style>

</ResourceDictionary>
```

////////////////////// Rows_Template.xaml//////////////////////

```
<ResourceDictionary xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
                    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">

    <DataTemplate x:Key="Rows_Template">
        <ItemsControl ItemsSource="{Binding}" ItemTemplate="{DynamicResource Ele-
ments_Template}">
            <ItemsControl.ItemsPanel>
                <ItemsPanelTemplate>
                    <StackPanel Orientation="Horizontal"/>
                </ItemsPanelTemplate>
            </ItemsControl.ItemsPanel>
        </ItemsControl>
```

```
    </DataTemplate>
</ResourceDictionary>
```
////////////////////// SettingsStyles.xaml//////////////////////
```
<ResourceDictionary xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
                    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
                    xmlns:converters="clr-namespace:Kakuro.Converters">

    <converters:EnumDescriptionConverter x:Key="EnumDescriptionConverter" />

    <Style x:Key="ListBoxItemStyle" TargetType="ListBoxItem">
        <Setter Property="Height" Value="40"/>
        <Setter Property="HorizontalContentAlignment" Value="Center"/>
    </Style>

    <Style x:Key="CheckBoxStyle" TargetType="CheckBox">
        <Setter Property="HorizontalAlignment" Value="Center"/>
        <Setter Property="VerticalAlignment" Value="Center"/>
        <Setter Property="LayoutTransform">
            <Setter.Value>
                <ScaleTransform ScaleX="1.1" ScaleY="1.1"/>
            </Setter.Value>
        </Setter>
    </Style>

</ResourceDictionary>
```
////////////////////// SumCell_Style.xaml//////////////////////
```
<ResourceDictionary
    xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
    xmlns:local="clr-namespace:Kakuro.Custom_Components">


    <Style TargetType="{x:Type local:SumCell}">
        <Setter Property="Template">
            <Setter.Value>
                <ControlTemplate TargetType="{x:Type local:SumCell}">
                    <Grid Background="Black">
                        <Border x:Name="FocusBorder"
                            BorderBrush="Transparent"
                            BorderThickness="2">
                            <Grid>
                                <TextBlock Text="{TemplateBinding SumRight}"
                                       HorizontalAlignment="Right"
                                       VerticalAlignment="Top"
                                       Foreground="White"
                                       FontSize="14"
                                       Margin="0,5,0,0" />

                                <Path Data="M 10 10 L 40 40"
                                  Stroke="White"
                                  StrokeThickness="1" />

                                <TextBlock Text="{TemplateBinding SumBottom}"
                                       HorizontalAlignment="Left"
                                       VerticalAlignment="Bottom"
                                       Foreground="White"
                                       FontSize="14"
                                       Margin="5,0,0,0" />
                            </Grid>
                        </Border>
                    </Grid>
                    <ControlTemplate.Triggers>
                        <Trigger Property="IsFocused" Value="True">
                            <Setter TargetName="FocusBorder" Property="BorderBrush" Val-
ue="LightBlue" />
```

```
                    </Trigger>
                </ControlTemplate.Triggers>
            </ControlTemplate>
        </Setter.Value>
    </Setter>
</Style>

</ResourceDictionary>
```
////////////////////// TabItem.xaml//////////////////////
```
<ResourceDictionary xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
                    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
                    xmlns:converters="clr-namespace:Kakuro.Converters">

    <converters:WindowWidthToTabItemWidthConverter x:Key="TabItemWidthConverter" />

    <Style TargetType="TabItem">
        <Setter Property="FontSize" Value="18"/>
        <Setter Property="Width" Value="{Binding ActualWidth, Rela-
tiveSource={RelativeSource AncestorType=Window}, Converter={StaticResource TabItemWidth-
Converter}}"/>
    </Style>

</ResourceDictionary>
```
////////////////////// TextBoxStyle.xaml//////////////////////
```
<ResourceDictionary xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
                    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
                    xmlns:customComponents="clr-namespace:Kakuro.Custom_Components">

    <Style TargetType="TextBox" x:Key="CustomTextBoxStyle">
        <Style.Triggers>
            <DataTrigger Binding="{Binding CellType}" Value="EmptyCell">
                <Setter Property="Background" Value="Black" />
                <Setter Property="BorderBrush" Value="Black" />
                <Setter Property="Text" Value="" />
                <Setter Property="IsReadOnly" Value="True" />
            </DataTrigger>

            <DataTrigger Binding="{Binding CellType}" Value="ValueCell">
                <Setter Property="Background" Value="White" />
                <Setter Property="BorderBrush" Value="Black" />
                <Setter Property="MaxLength" Value="1" />
                <Setter Property="VerticalContentAlignment" Value="Center" />
                <Setter Property="VerticalAlignment" Value="Center" />
                <Setter Property="Text" Value="{Binding DisplayValue, UpdateSourceTrig-
ger=PropertyChanged}" />
            </DataTrigger>

            <DataTrigger Binding="{Binding CellType}" Value="SumCell">
                <Setter Property="Template">
                    <Setter.Value>
                        <ControlTemplate TargetType="TextBox">
                            <customComponents:SumCell SumRight="{Binding SumRight}" Sum-
Bottom="{Binding SumBottom}"
                            HorizontalAlignment="Center" VerticalAlignment="Center"
                            Width="40" Height="40" />
                        </ControlTemplate>
                    </Setter.Value>
                </Setter>
            </DataTrigger>
        </Style.Triggers>
    </Style>
</ResourceDictionary>
```
////////////////////// TitleStyles.xaml//////////////////////
```
<ResourceDictionary xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
                    xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml">
```

```
    <Style x:Key="TitleTextBorderStyle" TargetType="Border">
        <Setter Property="BorderBrush" Value="Black"/>
        <Setter Property="BorderThickness" Value="1"/>
        <Setter Property="Padding" Value="5"/>
    </Style>

    <Style x:Key="TitleTextStyle" TargetType="TextBlock">
        <Setter Property="HorizontalAlignment" Value="Center"/>
        <Setter Property="VerticalAlignment" Value="Top"/>
        <Setter Property="FontWeight" Value="Bold"/>
        <Setter Property="Margin" Value="0,10"/>
    </Style>

</ResourceDictionary>
```

////////////////////// Folder ViewModels//////////////////////

////////////////////// DashboardItemViewModel.cs//////////////////////

```
using Kakuro.Base_Classes;
using Kakuro.Enums;
using Kakuro.Models;
using System.Text.Json.Serialization;

namespace Kakuro.ViewModels
{
    // #BAD: tests should be written
    // #BAD: shall there be interfaces?
    public class DashboardItemViewModel : ViewModelBase
    {
        private DashboardItem _dashboardItem;

        public int? ID
        {
            get => _dashboardItem.ID;
            set
            {
                _dashboardItem.ID = value;
                OnPropertyChanged(nameof(ID));
            }
        }

        public string DisplayValue
        {
            get => ConvertIntToString(_dashboardItem.DisplayValue);
            set
            {
                _dashboardItem.DisplayValue = ConvertStringToInt(value);
                OnPropertyChanged("DisplayValue");
            }
        }

        public string HiddenValue
        {
            get => ConvertIntToString(_dashboardItem.HiddenValue);
            set
            {
                _dashboardItem.HiddenValue = ConvertStringToInt(value);
                OnPropertyChanged("HiddenValue");
            }
        }

        public string SumRight
        {
            get => ConvertIntToString(_dashboardItem.SumRight);
            set
            {
```

```
                _dashboardItem.SumRight = ConvertStringToInt(value);
                OnPropertyChanged("SumRight");
            }
        }

        public string SumBottom
        {
            get => ConvertIntToString(_dashboardItem.SumBottom);
            set
            {
                _dashboardItem.SumBottom = ConvertStringToInt(value);
                OnPropertyChanged("SumBottom");
            }
        }

        public CellType CellType
        {
            get => _dashboardItem.CellType;
            set
            {
                _dashboardItem.CellType = value;
                OnPropertyChanged("CellType");
            }
        }

        public DashboardItemViewModel(DashboardItem dashboardItem)
        {
            _dashboardItem = dashboardItem;
        }

        [JsonConstructor]
        public DashboardItemViewModel(int? id, string displayValue, string hiddenValue,
CellType cellType, string sumRight, string sumBottom)
        {
            _dashboardItem = new DashboardItem();
            ID = id;
            DisplayValue = displayValue;
            HiddenValue = hiddenValue;
            CellType = cellType;
            SumRight = sumRight;
            SumBottom = sumBottom;
        }

        public int ConvertStringToInt(string value)
        {
            int enteredValue;
            try
            {
                enteredValue = string.IsNullOrEmpty(value) ? 0 : Convert.ToInt32(value);
            }
            catch (Exception) // validation
            {
                int? previousValue = _dashboardItem.DisplayValue;     // so user can't enter
letters and symbols like '(', '*', e.t.c. Only numbers from 1 to 9
                return previousValue.HasValue ? previousValue.Value : 0;
            }

            return enteredValue;
        }

        public string ConvertIntToString(int? value)
        {
            return value == 0 ? "" : $"{value}";
        }
    }
}
```

////////////////////// DashboardViewModel.cs//////////////////////

```
using Autofac;
using Kakuro.Base_Classes;
using Kakuro.Commands.DashboardViewModel;
using Kakuro.Enums;
using Kakuro.Events;
using Kakuro.Interfaces.Data_Access.Data_Providers;
using Kakuro.Interfaces.Game_Tools;
using Kakuro.Models;
using System.Collections.ObjectModel;
using System.Windows.Input;

namespace Kakuro.ViewModels
{
    // #BAD: tests should be written
    // #BAD: shall there be interfaces?
    public class DashboardViewModel : ViewModelBase
    {
        private DifficultyLevels _choosenDifficulty;
        private MyStopwatch _stopwatch;
        private string _stopWatchHours, _stopWatchMinutes, _stopWatchSeconds;
        private bool _isGameCompleted, _showCorrectAnswers, _autoSubmit, _isTimerVisible,
_disposed;
        private SubscriptionToken _settingsChangedSubscriptionTokens;
        private IOperationNotifier _operationNotifier;

        public DashboardItemCollection Dashboard { get; }
        public bool IsGameCompleted
        {
            get => _isGameCompleted;
            set
            {
                _isGameCompleted = value;
                OnPropertyChanged("IsGameCompleted");
            }
        }
        public bool ShowCorrectAnswers
        {
            get => _showCorrectAnswers;
            set
            {
                _showCorrectAnswers = value;
                OnPropertyChanged("ShowCorrectAnswers");
            }
        }
        public bool AutoSubmit
        {
            get => _autoSubmit;
            set
            {
                _autoSubmit = value;
                OnPropertyChanged("AutoSubmit");
            }
        }
        public bool IsTimerVisible
        {
            get => _isTimerVisible;
            set
            {
                _isTimerVisible = value;
                OnPropertyChanged("IsTimerVisible");
            }
        }
        public DifficultyLevels ChoosenDifficulty
        {
            get => _choosenDifficulty;
```

```
            set
            {
                _choosenDifficulty = value;
                OnPropertyChanged("ChoosenDifficulty");
            }
        }
        public string StopWatchHours
        {
            get { return _stopWatchHours.PadLeft(2, '0'); }
            set { _stopWatchHours = value; OnPropertyChanged("StopWatchHours"); }
        }
        public string StopWatchMinutes
        {
            get { return _stopWatchMinutes.PadLeft(2, '0'); }
            set { _stopWatchMinutes = value; OnPropertyChanged("StopWatchMinutes"); }
        }
        public string StopWatchSeconds
        {
            get { return _stopWatchSeconds.PadLeft(2, '0'); }
            set { _stopWatchSeconds = value; OnPropertyChanged("StopWatchSeconds"); }
        }

        public ICommand ApplyDifficultyCommand { get; private set; }
        public ICommand NewGameCommand { get; private set; }
        public ICommand ValidateSolutionCommand { get; private set; }
        public ICommand CleanDashboardCommand { get; private set; }
        public ICommand StartStopwatchCommand { get; private set; }
        public ICommand StopStopwatchCommand { get; private set; }
        public ICommand RestartStopwatchCommand { get; private set; }
        public ICommand SendGameSessionCommand { get; private set; }
        public ICommand GetChangedSettingsCommand { get; private set; }
        public ICommand AddMinuteAndContinueStopwatchCommand { get; private set; }
        public ICommand AutoSubmitCommand { get; private set; }
        public ICommand OpenRulesCommand { get; private set; }

        public DashboardViewModel(ILifetimeScope scope, IEventAggregator eventAggregator,
DashboardItemCollection dashboard)
        {
            ChoosenDifficulty = DifficultyLevels.Easy;
            Dashboard = dashboard;
            IsGameCompleted = false;
            ShowCorrectAnswers = false;
            AutoSubmit = true;
            IsTimerVisible = true;
            _operationNotifier = scope.Resolve<IOperationNotifier>();

            // #BAD: we shall create commands and some other objects through Lazy way

            _stopwatch = new MyStopwatch(new TimeSpan());
            StartStopwatchCommand = new StartStopwatchCommand(_stopwatch, this);
            StopStopwatchCommand = new StopStopwatchCommand(_stopwatch);
            RestartStopwatchCommand = new RestartStopwatchCommand(_stopwatch, StartStop-
watchCommand, StopStopwatchCommand);
            AddMinuteAndContinueStopwatchCommand = new AddMinuteAndContinueStopwatchCom-
mand(_stopwatch, StartStopwatchCommand);
            StopWatchHours = _stopwatch.Elapsed.Hours.ToString();
            StopWatchMinutes = _stopwatch.Elapsed.Minutes.ToString();
            StopWatchSeconds = _stopwatch.Elapsed.Seconds.ToString();

            SendGameSessionCommand = new SendGameSessionCommand(this, eventAggregator);

            ValidateSolutionCommand = new ValidateSolutionCommand(
                scope.Resolve<ISolutionValidator>(),
                _operationNotifier,
                StopStopwatchCommand,
                SendGameSessionCommand,
```

```
                this);

            ApplyDifficultyCommand = new ApplyDifficultyCommand(
                scope.Resolve<IDashboardProvider>(),
                this,
                RestartStopwatchCommand,
                eventAggregator,
                _stopwatch);

            CleanDashboardCommand = scope.Resolve<CleanDashboardCommand>();
            NewGameCommand = ApplyDifficultyCommand;

            GetChangedSettingsCommand = new GetChangedSettingsCommand(
                this,
                StopStopwatchCommand,
                AddMinuteAndContinueStopwatchCommand,
                CleanDashboardCommand,
                eventAggregator);

            AutoSubmitCommand = new AutoSubmitCommand(this, ValidateSolutionCommand);

            OpenRulesCommand = new OpenRulesCommand();

            ApplyDifficultyCommand.Execute(ChoosenDifficulty);

            _settingsChangedSubscriptionTokens = eventAggrega-
tor.GetEvent<SettingsChangedEvent>().Subscribe(GetChangedSettingsCommand.Execute);
        }

        public DashboardItemCollection CreateDashboardCopy()
        {
            var newCollection = new DashboardItemCollection();
            foreach (var innerCollection in Dashboard)
            {
                var newInnerCollection = new ObservableCollec-
tion<DashboardItemViewModel>();
                foreach (var item in innerCollection)
                {
                    var newItem = new DashboardItemViewModel(new DashboardItem
                    {
                        DisplayValue = item.ConvertStringToInt(item.DisplayValue),
                        HiddenValue = item.ConvertStringToInt(item.HiddenValue),
                        SumRight = item.ConvertStringToInt(item.SumRight),
                        SumBottom = item.ConvertStringToInt(item.SumBottom),
                        CellType = item.CellType
                    });
                    newInnerCollection.Add(newItem);
                }
                newCollection.Add(newInnerCollection);
            }
            return newCollection;
        }

        public void Dispose()
        {
            Dispose(true);
            GC.SuppressFinalize(this);
        }

        protected virtual void Dispose(bool disposing)
        {
            if (!_disposed)
            {
                if (disposing)
                {
                    if (_settingsChangedSubscriptionTokens != null)
```

```
                {
                    _settingsChangedSubscriptionTokens.Dispose();
                    _settingsChangedSubscriptionTokens = null;
                }

                _operationNotifier.CloseAllNotifications();
            }
            _disposed = true;
        }
    }

    ~DashboardViewModel()
    {
        Dispose(false);
    }
  }
}
```

////////////////////// MainViewModel.cs//////////////////////

```
using Kakuro.Base_Classes;

namespace Kakuro.ViewModels
{
    // #BAD: tests should be written
    public class MainViewModel : ViewModelBase, IDisposable
    {
        public DashboardViewModel DashboardViewModel { get; }
        public RatingTableViewModel RatingTableViewModel { get; }
        public SettingsViewModel SettingsViewModel { get; }
        public SavepointsViewModel SavepointsViewModel { get; }

        private bool _disposed = false;

        public MainViewModel(
            DashboardViewModel dashboardViewModel,
            RatingTableViewModel ratingTableViewModel,
            SettingsViewModel settingsViewModel,
            SavepointsViewModel savepointsViewModel)
        {
            DashboardViewModel = dashboardViewModel;
            RatingTableViewModel = ratingTableViewModel;
            SettingsViewModel = settingsViewModel;
            SavepointsViewModel = savepointsViewModel;
        }

        public void Dispose()
        {
            Dispose(true);
            GC.SuppressFinalize(this);
        }

        protected virtual void Dispose(bool disposing)
        {
            if (!_disposed)
            {
                if (disposing)
                {
                    RatingTableViewModel.Dispose();
                    DashboardViewModel.Dispose();
                    SettingsViewModel.Dispose();
                    SavepointsViewModel.Dispose();
                }
                _disposed = true;
            }
        }

        ~MainViewModel()
```

```
        {
            Dispose(false);
        }
    }
}
```

////////////////////// RatingTableViewModel.cs//////////////////////

```
using Kakuro.Commands.RatingTableViewModel;
using Kakuro.Enums;
using Kakuro.Events;
using Kakuro.Interfaces.Data_Access.Data_Providers;
using Kakuro.Models;
using System.Collections.ObjectModel;
using System.Windows.Input;

namespace Kakuro.ViewModels
{
    // #BAD: tests should be written
    // #BAD: shall there be interfaces?
    public class RatingTableViewModel : IDisposable
    {
        private IRatingRecordProvider _ratingRecordProvider;
        private RatingTableContainer _ratingTablesContainer;
        private IEventAggregator _eventAggregator;
        private bool _disposed = false;
        private SubscriptionToken? _gameCompletedSubscriptionToken;
        private EventHandler? _saveCompletedHandler;

        public ObservableCollection<RatingRecord> EasyRatingRecords { get; }
        public ObservableCollection<RatingRecord> NormalRatingRecords { get; }
        public ObservableCollection<RatingRecord> HardRatingRecords { get; }

        public ICommand LoadRatingRecordsCommand { get; }
        public ICommand SaveRatingRecordCommand { get; }

        public RatingTableViewModel(IRatingRecordProvider ratingRecordProvider, IEvent-
Aggregator eventAggregator)
        {
            _ratingRecordProvider = ratingRecordProvider;
            EasyRatingRecords = new ObservableCollection<RatingRecord>();
            NormalRatingRecords = new ObservableCollection<RatingRecord>();
            HardRatingRecords = new ObservableCollection<RatingRecord>();
            _ratingTablesContainer = new RatingTableContainer
            {
                { DifficultyLevels.Easy, EasyRatingRecords },
                { DifficultyLevels.Normal, NormalRatingRecords },
                { DifficultyLevels.Hard, HardRatingRecords }
            };

            _eventAggregator = eventAggregator;

            LoadRatingRecordsCommand = new LoadRatingRecordsCommand(_ratingRecordProvider,
_ratingTablesContainer);

            SaveRatingRecordCommand = new SaveRatingRecordCommand(_ratingRecordProvider);

            _saveCompletedHandler = (sender, e) =>
            {
                LoadRatingRecordsCommand.Execute(null);
            };

            ((SaveRatingRecordCommand)SaveRatingRecordCommand).SaveCompleted +=
_saveCompletedHandler;

            LoadRatingRecordsCommand.Execute(null);
```

```
            _gameCompletedSubscriptionToken = eventAggrega-
tor.GetEvent<GameCompletedEvent>().Subscribe(SaveRatingRecordCommand.Execute);
        }

        public void Dispose()
        {
            Dispose(true);
            GC.SuppressFinalize(this);
        }

        protected virtual void Dispose(bool disposing)
        {
            if (!_disposed)
            {
                if (disposing)
                {
                    if (_saveCompletedHandler != null)
                    {
                        ((SaveRatingRecordCommand)SaveRatingRecordCommand).SaveCompleted
-= _saveCompletedHandler;
                        _saveCompletedHandler = null;
                    }

                    if (_gameCompletedSubscriptionToken != null)
                    {
                        _gameCompletedSubscriptionToken.Dispose();
                        _gameCompletedSubscriptionToken = null;
                    }
                }
                _disposed = true;
            }
        }

        ~RatingTableViewModel()
        {
            Dispose(false);
        }
    }
}
```

////////////////////// SavepointsViewModel.cs//////////////////////

```
using Autofac;
using Kakuro.Base_Classes;
using Kakuro.Commands.SavepointsViewModel;
using Kakuro.Events;
using Kakuro.Interfaces.Data_Access.Data_Providers;
using Kakuro.Interfaces.Game_Tools;
using System.Collections.ObjectModel;
using System.Windows.Input;

namespace Kakuro.ViewModels
{
    public class SavepointsViewModel : ViewModelBase
    {
        private SavepointViewModel _selectedSavepoint;
        private ISavepointProvider _savepointProvider;
        private IOperationNotifier _operationNotifier;
        private bool _correctAnswersAreShown;

        public ObservableCollection<SavepointViewModel> Savepoints { get; }

        public bool CorrectAnswersAreShown
        {
            get => _correctAnswersAreShown;
            set
            {
                _correctAnswersAreShown = value;
```

```
                OnPropertyChanged("CorrectAnswersAreShown");
            }
        }

        public SavepointViewModel SelectedSavepoint
        {
            get => _selectedSavepoint;
            set
            {
                if (_selectedSavepoint != value)
                {
                    _selectedSavepoint = value;
                    OnPropertyChanged(nameof(SelectedSavepoint));
                }
            }
        }

        public bool IsCreatingAllowed
        {
            get => _isCreatingAllowed;
            set
            {
                _isCreatingAllowed = value;
                OnPropertyChanged("IsCreatingAllowed");
            }
        }

        public ICommand LoadSavepointCommand { get; }
        public ICommand CreateSavepointCommand { get; }
        public ICommand RewriteSavepointCommand { get; }
        public ICommand DeleteSavepointCommand { get; }
        public ICommand CleanSavepointsCommand { get; }
        public ICommand ChangeAvailabilityOfSectionCommand { get; }

        private SubscriptionToken _newGameStartedSubscriptionToken;
        private SubscriptionToken _correctAnswersTurnedOnSubscriptionToken;
        private bool _disposed;
        private bool _isCreatingAllowed;

        public SavepointsViewModel(
            ILifetimeScope scope,
            IEventAggregator eventAggregator,
            ISavepointProvider savepointProvider)
        {
            Savepoints = new ObservableCollection<SavepointViewModel>();
            _savepointProvider = savepointProvider;
            _operationNotifier = scope.Resolve<IOperationNotifier>();
            IsCreatingAllowed = true;
            CorrectAnswersAreShown = false;

            // #BAD: I think we shouldn't pass DashboardViewModel straightfully
            DeleteSavepointCommand = new DeleteSavepointCommand(this, _savepointProvider);
            CreateSavepointCommand = new CreateSavepointCommand(this,
scope.Resolve<DashboardViewModel>(), _savepointProvider);
            RewriteSavepointCommand = new RewriteSavepointCommand(this,
scope.Resolve<DashboardViewModel>(), _savepointProvider, _operationNotifier);
            LoadSavepointCommand = new LoadSavepointCommand(this,
scope.Resolve<DashboardViewModel>(), _savepointProvider);
            CleanSavepointsCommand = new CleanSavepointsCommand(this, _savepointProvider);
            ChangeAvailabilityOfSectionCommand = new ChangeAvailabilityOfSectionCom-
mand(this);

            _newGameStartedSubscriptionToken = eventAggrega-
tor.GetEvent<NewGameStartedEvent>().Subscribe(CleanSavepointsCommand.Execute);
```

```
            _correctAnswersTurnedOnSubscriptionToken = eventAggrega-
tor.GetEvent<CorrectAnswersTurnedOnEvent>().Subscribe(ChangeAvailabilityOfSectionCommand.
Execute);
        }

        public void Dispose()
        {
            Dispose(true);
            GC.SuppressFinalize(this);
        }

        protected virtual void Dispose(bool disposing)
        {
            if (!_disposed)
            {
                if (disposing)
                {
                    if (_newGameStartedSubscriptionToken != null)
                    {
                        _newGameStartedSubscriptionToken.Dispose();
                        _newGameStartedSubscriptionToken = null;
                    }

                    if (_correctAnswersTurnedOnSubscriptionToken != null)
                    {
                        _correctAnswersTurnedOnSubscriptionToken.Dispose();
                        _correctAnswersTurnedOnSubscriptionToken = null;
                    }

                    (DeleteSavepointCommand as IDisposable)?.Dispose();
                    (CreateSavepointCommand as IDisposable)?.Dispose();
                    (RewriteSavepointCommand as IDisposable)?.Dispose();
                    (LoadSavepointCommand as IDisposable)?.Dispose();

                    CleanSavepointsCommand.Execute(null);
                }
                _disposed = true;
            }
        }

        ~SavepointsViewModel()
        {
            Dispose(false);
        }
    }

}
```

////////////////////// SavepointViewModel.cs//////////////////////

```
using Kakuro.Base_Classes;

namespace Kakuro.ViewModels
{
    public class SavepointViewModel : ViewModelBase
    {
        private string _name;
        private int _id;

        public int Id
        {
            get => _id;
            set
            {
                _id = value;
                OnPropertyChanged("Id");
            }
        }
```

```
        public string Name
        {
            get => _name;
            set
            {
                _name = value;
                OnPropertyChanged("Name");
            }
        }

        public SavepointViewModel(int id, string name)
        {
            Id = id;
            Name = name;
        }
    }
}
```

////////////////////// SettingsViewModel.cs//////////////////////

```
using Kakuro.Commands.SettingsViewModel;
using Kakuro.Enums;
using Kakuro.Events;
using Kakuro.Models;
using System.Collections.ObjectModel;
using System.Windows.Input;

namespace Kakuro.ViewModels
{
    public class SettingsViewModel : IDisposable
    {
        private IEventAggregator _eventAggregator;
        private bool _disposed = false;

        public ObservableCollection<SettingViewModel> Settings { get; set; }
        public ICommand SendSettingsCommand { get; }
        public ICommand TurnOffAutoSubmitCommand { get; }

        private SubscriptionToken _correctAnswersTurnedOnSubscriptionToken;

        public SettingsViewModel(IEventAggregator eventAggregator)
        {
            _eventAggregator = eventAggregator;
            Settings = new ObservableCollection<SettingViewModel>
            {
                new SettingViewModel(new Setting(SettingType.ShowCorrectAnswers, false)),
                new SettingViewModel(new Setting(SettingType.AutoSubmit, true)),
                new SettingViewModel(new Setting(SettingType.HideTimer, false))
            };

            SendSettingsCommand = new SendSettingsCommand(_eventAggregator, Settings);
            TurnOffAutoSubmitCommand = new TurnOffAutoSubmitCommand(Settings);

            _correctAnswersTurnedOnSubscriptionToken = eventAggrega-
tor.GetEvent<CorrectAnswersTurnedOnEvent>().Subscribe(TurnOffAutoSubmitCommand.Execute);
        }

        public void Dispose()
        {
            Dispose(true);
            GC.SuppressFinalize(this);
        }

        protected virtual void Dispose(bool disposing)
        {
            if (!_disposed)
            {
```

```
                if (disposing)
                {
                    if (_correctAnswersTurnedOnSubscriptionToken != null)
                    {
                        _correctAnswersTurnedOnSubscriptionToken.Dispose();
                        _correctAnswersTurnedOnSubscriptionToken = null;
                    }
                }
                _disposed = true;
            }
        }

        ~SettingsViewModel()
        {
            Dispose(false);
        }
    }
}
```

////////////////////// SettingViewModel.cs//////////////////////

```
using Kakuro.Base_Classes;
using Kakuro.Enums;
using Kakuro.Models;

namespace Kakuro.ViewModels
{
    public class SettingViewModel : ViewModelBase
    {
        private readonly Setting _setting;

        public SettingViewModel(Setting setting)
        {
            _setting = setting;
        }

        public SettingType SettingType
        {
            get => _setting.SettingType;
            set
            {
                _setting.SettingType = value;
                OnPropertyChanged();
            }
        }

        public bool IsEnabled
        {
            get => _setting.IsEnabled;
            set
            {
                _setting.IsEnabled = value;
                OnPropertyChanged();
            }
        }
    }
}
```

////////////////////// Folder Views//////////////////////

////////////////////// DashboardSection.xaml//////////////////////

```
<UserControl x:Class="Kakuro.Views.DashboardSection"
             xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
             xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
             xmlns:local="clr-namespace:Kakuro.Views"
             xmlns:viewmodels="clr-namespace:Kakuro.ViewModels"
             mc:Ignorable="d"
```

```
             d:DesignHeight="450" d:DesignWidth="800">

    <Grid>
        <Grid.ColumnDefinitions>
            <ColumnDefinition></ColumnDefinition>
            <ColumnDefinition Width="{Binding RelativeSource={RelativeSource
Mode=FindAncestor, AncestorType={x:Type Grid}}, Path=ActualHeight}"/>
            <ColumnDefinition></ColumnDefinition>
        </Grid.ColumnDefinitions>

        <Grid Grid.Column="1" HorizontalAlignment="Center">
            <ItemsControl x:Name="Dashboard" ItemsSource="{Binding Dashboard}" ItemTem-
plate="{DynamicResource Rows_Template}"/>
        </Grid>
    </Grid>
</UserControl>
```

////////////////////// DifficultyLevelsSection.xaml//////////////////////

```
<UserControl x:Class="Kakuro.Views.DifficultyLevelsSection"
             xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
             xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
             xmlns:local="clr-namespace:Kakuro.Views"
             xmlns:enums="clr-namespace:Kakuro.Enums"
             mc:Ignorable="d"
             d:DesignHeight="450" d:DesignWidth="800">


    <Grid>
        <Grid.ColumnDefinitions>
            <ColumnDefinition></ColumnDefinition>
            <ColumnDefinition Width="0.1*"></ColumnDefinition>
            <ColumnDefinition></ColumnDefinition>
            <ColumnDefinition Width="0.1*"></ColumnDefinition>
            <ColumnDefinition></ColumnDefinition>
        </Grid.ColumnDefinitions>

        <RadioButton Content="Easy - 6x6"
     Command="{Binding ApplyDifficultyCommand}"
     CommandParameter="{x:Static enums:DifficultyLevels.Easy}"
     Grid.Column="0"
     IsChecked="True"
     IsEnabled="{Binding ShowCorrectAnswers, Converter={StaticResource InverseBoolean-
Converter}}" />

        <RadioButton Content="Normal - 10x10"
     Command="{Binding ApplyDifficultyCommand}"
     CommandParameter="{x:Static enums:DifficultyLevels.Normal}"
     Grid.Column="2"
     IsEnabled="{Binding ShowCorrectAnswers, Converter={StaticResource InverseBoolean-
Converter}}" />

        <RadioButton Content="Hard - 16x16"
     Command="{Binding ApplyDifficultyCommand}"
     CommandParameter="{x:Static enums:DifficultyLevels.Hard}"
     Grid.Column="4"
     IsEnabled="{Binding ShowCorrectAnswers, Converter={StaticResource InverseBoolean-
Converter}}" />

    </Grid>
</UserControl>
```

////////////////////// GameSection.xaml//////////////////////

```
<UserControl x:Class="Kakuro.Views.GameSection"
             xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
```

```
            xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
            xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
            xmlns:view="clr-namespace:Kakuro.Views"
            mc:Ignorable="d"
            d:DesignHeight="450" d:DesignWidth="800">

    <Grid>
        <Grid.RowDefinitions>
            <RowDefinition Height="20"></RowDefinition>
            <RowDefinition Height="40"></RowDefinition>
            <RowDefinition Height="20"></RowDefinition>
            <RowDefinition Height="40"></RowDefinition>
            <RowDefinition Height="20"></RowDefinition>
            <RowDefinition Height="Auto"></RowDefinition>
            <RowDefinition Height="20"></RowDefinition>
            <RowDefinition Height="40"></RowDefinition>
            <RowDefinition Height="20"></RowDefinition>
            <RowDefinition Height="40"></RowDefinition>
            <RowDefinition Height="40"></RowDefinition>
        </Grid.RowDefinitions>

        <view:DifficultyLevelsSection Grid.Row="1"/>
        <view:StopwatchSection Grid.Row="3"/>
        <view:DashboardSection Grid.Row="5"/>
        <view:SubmitSection Grid.Row="7"/>
        <view:GameToolsSection Grid.Row="9"/>
    </Grid>

</UserControl>
```

////////////////////// GameTab.xaml//////////////////////

```
<UserControl x:Class="Kakuro.Views.GameTab"
            xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
            xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
            xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
            xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
            xmlns:view="clr-namespace:Kakuro.Views"
            mc:Ignorable="d"
            d:DesignHeight="450" d:DesignWidth="800">
    <Grid>
        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="5.5*"></ColumnDefinition>
            <ColumnDefinition Width="0.3*"></ColumnDefinition>
            <ColumnDefinition Width="10.5*"></ColumnDefinition>
            <ColumnDefinition Width="0.3*"></ColumnDefinition>
            <ColumnDefinition Width="5.5*"></ColumnDefinition>
        </Grid.ColumnDefinitions>

        <view:SettingsSection Grid.Column="0" DataContext="{Binding SettingsViewModel}"/>
        <view:GameSection Grid.Column="2" DataContext="{Binding DashboardViewModel}"/> <!--
#BAD: Maybe we could use ViewModelLocator?-->
        <view:SavepointsSection Grid.Column="4" DataContext="{Binding SavepointsViewMod-
el}"/>
    </Grid>
</UserControl>
```

////////////////////// GameToolsSection.xaml//////////////////////

```
<UserControl x:Class="Kakuro.Views.GameToolsSection"
            xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
            xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
            xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
            xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
            xmlns:local="clr-namespace:Kakuro.Views"
            mc:Ignorable="d"
            d:DesignHeight="450" d:DesignWidth="800">
    <Grid>
        <Grid.ColumnDefinitions>
```

```
            <ColumnDefinition></ColumnDefinition>
            <ColumnDefinition Width="4*"></ColumnDefinition>
            <ColumnDefinition></ColumnDefinition>
            <ColumnDefinition Width="4*"></ColumnDefinition>
            <ColumnDefinition></ColumnDefinition>
            <ColumnDefinition Width="4*"></ColumnDefinition>
            <ColumnDefinition></ColumnDefinition>
        </Grid.ColumnDefinitions>

        <Button Grid.Column="1"
        Command="{Binding CleanDashboardCommand}"
        IsEnabled="{Binding ShowCorrectAnswers, Converter={StaticResource InverseBoole-
anConverter}}">
            Full Clean
        </Button>

        <Button Grid.Column="3"
                Command="{Binding OpenRulesCommand}">
            Rules
        </Button>

        <Button Grid.Column="5"
        Command="{Binding NewGameCommand}"
        CommandParameter="{Binding ChoosenDifficulty}"
        IsEnabled="{Binding ShowCorrectAnswers, Converter={StaticResource InverseBoole-
anConverter}}">
            New Game
        </Button>
    </Grid>
</UserControl>
```

////////////////////// MainWindow.xaml//////////////////////

```
<Window x:Class="Kakuro.MainWindow"
        xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
        xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
        xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
        xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
        xmlns:local="clr-namespace:Kakuro"
        xmlns:view="clr-namespace:Kakuro.Views"
        mc:Ignorable="d"
        Title="Kakuro" Height="627" Width="1210" WindowStartupLocation="CenterScreen"
Closed="Window_Closed" WindowState="Maximized" ResizeMode="NoResize">

    <TabControl>
        <TabItem Header="Game">
            <view:GameTab/>
        </TabItem>

        <TabItem Header="Rating table" DataContext="{Binding RatingTableViewModel}">
            <view:RatingTableTab/>
        </TabItem>
    </TabControl>

</Window>
```

////////////////////// MainWindow.xaml.cs//////////////////////

```
using Kakuro.ViewModels;
using System.Windows;

namespace Kakuro
{
    public partial class MainWindow : Window, IDisposable
    {
        private MainViewModel _mainViewModel;
        private bool _disposed = false;

        public MainWindow(MainViewModel mainViewModel)
```

```
        {
            InitializeComponent();

            _mainViewModel = mainViewModel;
            DataContext = _mainViewModel;
        }

        public void Dispose()
        {
            Dispose(true);
            GC.SuppressFinalize(this);
        }

        protected virtual void Dispose(bool disposing)
        {
            if (!_disposed)
            {
                if (disposing)
                {
                    _mainViewModel.Dispose();
                }
                _disposed = true;
            }
        }

        ~MainWindow()
        {
            Dispose(false);
        }

        private void Window_Closed(object sender, EventArgs e) // #BAD: i guess i shall do
it using commands
        {
            Dispose();
        }
    }
}
```

////////////////////// RatingTableTab.xaml//////////////////////

```
<UserControl x:Class="Kakuro.Views.RatingTableTab"
             xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
             xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
             xmlns:local="clr-namespace:Kakuro.Views"
             mc:Ignorable="d"
             d:DesignHeight="450" d:DesignWidth="800">

    <Grid>
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto"/>
            <RowDefinition Height="*"/>
        </Grid.RowDefinitions>

        <Grid.ColumnDefinitions>
            <ColumnDefinition Width="*" />
            <ColumnDefinition Width="*" />
            <ColumnDefinition Width="*" />
        </Grid.ColumnDefinitions>

        <Border Grid.Row="0" Grid.Column="0" Style="{StaticResource TitleTextBorderStyle}">
            <TextBlock Text="Easy" Style="{StaticResource TitleTextStyle}" />
        </Border>

        <Border Grid.Row="0" Grid.Column="1" Style="{StaticResource TitleTextBorderStyle}">
            <TextBlock Text="Normal" Style="{StaticResource TitleTextStyle}" />
        </Border>
```

```
        <Border Grid.Row="0" Grid.Column="2" Style="{StaticResource TitleTextBorderStyle}">
            <TextBlock Text="Hard" Style="{StaticResource TitleTextStyle}" />
        </Border>

        <ContentControl Grid.Row="1" Grid.Column="0" Content="{Binding EasyRatingRecords}"
ContentTemplate="{StaticResource RatingDataGridTemplate}"/>
        <ContentControl Grid.Row="1" Grid.Column="1" Content="{Binding NormalRat-
ingRecords}" ContentTemplate="{StaticResource RatingDataGridTemplate}"/>
        <ContentControl Grid.Row="1" Grid.Column="2" Content="{Binding HardRatingRecords}"
ContentTemplate="{StaticResource RatingDataGridTemplate}"/>
    </Grid>

</UserControl>
```

////////////////////// SavepointsButtonsSection.xaml//////////////////////

```
<UserControl x:Class="Kakuro.Views.SavepointsButtonsSection"
             xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
             xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
             xmlns:local="clr-namespace:Kakuro.Views"
             mc:Ignorable="d"
             d:DesignHeight="450" d:DesignWidth="800">
    <Grid>
        <Grid.ColumnDefinitions>
            <ColumnDefinition></ColumnDefinition>
            <ColumnDefinition Width="4*"></ColumnDefinition>
            <ColumnDefinition></ColumnDefinition>
            <ColumnDefinition Width="4*"></ColumnDefinition>
            <ColumnDefinition></ColumnDefinition>
            <ColumnDefinition Width="4*"></ColumnDefinition>
            <ColumnDefinition></ColumnDefinition>
            <ColumnDefinition Width="4*"></ColumnDefinition>
            <ColumnDefinition></ColumnDefinition>
        </Grid.ColumnDefinitions>

        <Button Grid.Column="1" Command="{Binding LoadSavepointCommand}">Load</Button>
        <Button Grid.Column="3" Command="{Binding CreateSavepointCommand}" IsEna-
bled="{Binding IsCreatingAllowed}">Create</Button>
        <Button Grid.Column="5" Command="{Binding RewriteSavepointCom-
mand}">Rewrite</Button>
        <Button Grid.Column="7" Command="{Binding DeleteSavepointCommand}">Delete</Button>
    </Grid>
</UserControl>
```

////////////////////// SavepointsSection.xaml//////////////////////

```
<UserControl x:Class="Kakuro.Views.SavepointsSection"
             xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
             xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
             xmlns:view="clr-namespace:Kakuro.Views"
             mc:Ignorable="d"
             d:DesignHeight="450" d:DesignWidth="800">
    <Grid>
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto" />
            <RowDefinition Height="Auto" />
            <RowDefinition Height="*" />
        </Grid.RowDefinitions>

        <Border Grid.Row="0" Style="{StaticResource TitleTextBorderStyle}">
            <TextBlock Text="Savepoints" Style="{StaticResource TitleTextStyle}" />
        </Border>
        <Border Grid.Row="1" Style="{StaticResource TitleTextBorderStyle}">
            <view:SavepointsButtonsSection Grid.Row="1"/>
```

```
        </Border>

        <ListBox Grid.Row="2"
                 ItemsSource="{Binding Savepoints}"
                 SelectedItem="{Binding SelectedSavepoint}"
                 HorizontalContentAlignment="Center">
            <ListBox.ItemTemplate>
                <DataTemplate>
                    <TextBlock Text="{Binding Name}"
                               HorizontalAlignment="Center"
                               TextAlignment="Center"/>
                </DataTemplate>
            </ListBox.ItemTemplate>
        </ListBox>

    </Grid>
</UserControl>
```

////////////////////// SettingsSection.xaml//////////////////////

```
<UserControl x:Class="Kakuro.Views.SettingsSection"
             xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
             xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
             xmlns:local="clr-namespace:Kakuro.Views"
             xmlns:i="http://schemas.microsoft.com/xaml/behaviors"
             mc:Ignorable="d"
             d:DesignHeight="450" d:DesignWidth="800">
    <Grid>
        <Grid.RowDefinitions>
            <RowDefinition Height="Auto" />
            <RowDefinition Height="*" />
        </Grid.RowDefinitions>

        <Border Grid.Row="0" Style="{StaticResource TitleTextBorderStyle}">
            <TextBlock Text="Settings" Style="{StaticResource TitleTextStyle}" />
        </Border>

        <ListBox Grid.Row="1" ItemsSource="{Binding Settings}"
                 ItemContainerStyle="{StaticResource ListBoxItemStyle}"
                 ScrollViewer.HorizontalScrollBarVisibility="Disabled">
            <ListBox.ItemTemplate>
                <DataTemplate>
                    <CheckBox Content="{Binding SettingType, Converter={StaticResource
EnumDescriptionConverter}}"
                              IsChecked="{Binding IsEnabled}"
                              Style="{StaticResource CheckBoxStyle}">
                        <i:Interaction.Triggers>
                            <i:EventTrigger EventName="Click">
                                <i:InvokeCommandAction Command="{Binding DataCon-
text.SendSettingsCommand, RelativeSource={RelativeSource AncestorType=ListBox}}" />
                            </i:EventTrigger>
                        </i:Interaction.Triggers>
                    </CheckBox>
                </DataTemplate>
            </ListBox.ItemTemplate>
        </ListBox>
    </Grid>
</UserControl>
```

////////////////////// StopwatchSection.xaml//////////////////////

```
<UserControl x:Class="Kakuro.Views.StopwatchSection"
             xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
             xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
             xmlns:local="clr-namespace:Kakuro.Views"
```

```
             mc:Ignorable="d"
             d:DesignHeight="450" d:DesignWidth="800">
    <Grid>
        <Grid.ColumnDefinitions>
            <ColumnDefinition></ColumnDefinition>
            <ColumnDefinition></ColumnDefinition>
            <ColumnDefinition></ColumnDefinition>
        </Grid.ColumnDefinitions>

        <TextBlock VerticalAlignment="Center" Grid.Column="1" HorizontalAlignment="Center"
FontSize="45" Foreground="Red"
           Visibility="{Binding IsTimerVisible, UpdateSourceTrigger=PropertyChanged,
Converter={StaticResource BooleanToVisibilityConverter}}">
            <TextBlock.Inlines>
                <Run Text="{Binding StopWatchHours}" />
                <Run Text=":" />
                <Run Text="{Binding StopWatchMinutes}" />
                <Run Text=":" />
                <Run Text="{Binding StopWatchSeconds}" />
            </TextBlock.Inlines>
        </TextBlock>


    </Grid>
</UserControl>
```

////////////////////// SubmitSection.xaml//////////////////////

```
<UserControl x:Class="Kakuro.Views.SubmitSection"
             xmlns="http://schemas.microsoft.com/winfx/2006/xaml/presentation"
             xmlns:x="http://schemas.microsoft.com/winfx/2006/xaml"
             xmlns:mc="http://schemas.openxmlformats.org/markup-compatibility/2006"
             xmlns:d="http://schemas.microsoft.com/expression/blend/2008"
             xmlns:local="clr-namespace:Kakuro.Views"
             mc:Ignorable="d"
             d:DesignHeight="450" d:DesignWidth="800">
    <Grid>
        <Grid.ColumnDefinitions>
            <ColumnDefinition></ColumnDefinition>
            <ColumnDefinition></ColumnDefinition>
            <ColumnDefinition></ColumnDefinition>
        </Grid.ColumnDefinitions>

        <Button Grid.Column="1" Command="{Binding ValidateSolutionCommand}">
            <Button.IsEnabled>
                <MultiBinding Converter="{StaticResource MultiOrBooleanConverter}">
                    <Binding Path="IsGameCompleted" />
                    <Binding Path="ShowCorrectAnswers" />
                </MultiBinding>
            </Button.IsEnabled>
            Submit
        </Button>
    </Grid>
</UserControl>
```