# **Sample Refactoring Documentation for Project “Minesweeper-4”**

Team “Minesweeper-4”

1. Redesigned the project structure:

* Extracted class in a separate file: **ScoreRecord.cs**.
* Renamed namespace from **mini4ki** to **Minesweeper**.
* Created classes with appropriate names to make the code testable: **ScoreBoard.cs**, **GameField.cs** and **Posision.cs**.
* Renamed main class from **MinesweeperMain** to **MinesweeperGame**.

1. Reformatted the source code:

* Removed unnecessary using statements.
* Removed all unneeded empty lines.
* Inserted empty lines between the methods.
* Formatted the curly braces **{** and **}** according to the best practices for the C# language.
* Put **{** and **}** after all conditionals and loops (when missing).
* Character casing: variables and fields made **camelCase**; types and methods made **PascalCase**.
* Formatted all other elements of the source code according to the best practices introduced in the course “High-Quality Programming Code”.

1. Extracted class **ScoreRecord** from the **Minesweeper** class and moved all related functionality to it.

* Specified its fields as private.
* Removed empty constructor.
* Rearranged the code to match the best practices introduced in the course e.g. first fields, than constructors, than properties, than methods, etc.
* Added “**this.**” when accessing the fields from the properties.
* Made all access to the fields through the properties.
* Introduced **ToString()** method.

1. Introduced class **ScoreBoard** and moved all related functionality in it.
2. Renamed variables:

* In class **Minesweeper: counter -> score**.
* In class **Minesweeper: bombed -> isGameOver**.
* In class **Minesweeper: flag -> isGameWon**.
* In class **Minesweeper: welcomeFlag -> isNewGame**.
* In class **Minesweeper: rowIndex -> row**.
* In class **Minesweeper: columnIndex -> col**.
* In class **Minesweeper: selectedCommand -> command**.
* In class **Minesweeper:** **MaxRevealedCells -> ScoreToWin**.
* In class **ScoreRecord**: **personName ->** **playerName**.
* In class **ScoreRecord**: **scorePoints ->** **playerScore**.
* In class **ScoreBoard**: **champions ->** **highScores**.

1. Introduced constants in the **GameField** class:

* **FieldRows** = 5.
* **FieldColumns** = 10.
* **BombsCount** = 15.

1. Moved **PrintScoreBoard(…)**’s logic from the main class to a **ToString()** override in the **ScoreBoard** class.
2. Moved **PrintBoard(…)**’s logic from the main class to a **ToString()** override in the **GameField** class.
3. Moved **CreateWhiteBoard()** and **CreateBombBoard()**’s logic from the main class to a **GenerateGameField()** method in the **GameField** class.
4. Moved **CalculateBombBoard(…)**’s logic from the main class to a **RevealField()** method in the **GameField** class.
5. Moved **CalculateHowManyBombs (…)**’s logic from the main class to a **RevealPosision(…)** method in the **GameField** class.
6. Removed **MakeAMove(…)** method from the main class and reassigned its job to **RevealPosision(…)** and **RevealField()**.
7. Extracted all game logic from the **Main()** method to a new class **Engine**. Encapsulated the data and made the new class a **Singleton**.
8. Made the fields **gameField** and **scoreBoard** static. Implemented lazy instantiation.
9. Added **ClassStructure** class diagram
10. Added **IScoreRecord** interface
11. Implemented **IScoreRecord** on **ScoreRecord** class
12. Performed property validation in **ScoreRecord** class
13. The following has been changed in **ScoreBoard** class:

* Separated part of the **AddScore()** logic in **SortEqualHighScores(…)**
* Added **Reset()** method (Re-instantiating the class on game restart is not needed anymore)
* Modified **if** statement on **line 55** (there is a comment)
* Implemented *Dependency Inversion*

1. Separated part of the logic of **Start()** in few methods in **Engine** class. *The class is still refactored.*
2. Separated the logic for counting surrounding bombs from **RevealPosition()** in new method **SurroundingBombCount()**.
3. Replaced the use of **Field.GetLength()** with the constants **FieldRows** and **FieldColumns**
4. Fixed logic error in **Reveal()** **rowIsInRange** and **colIsInRage** which was causing exceptions when entered number is greater by one then the maximum rows or columns in the field.
5. Implemented **Memento design pattern** for saving and restoring the game field.

* Added the following classes: **GameFieldMemento**, **GameFieldSave**.
* Added **Save()** and **Restore()** methods to the **GameField** class.
* Added **SaveCommand()** and **RestoreCommand()** methods to the **Engine**.
* Made **gameField** in **Engine** class non-static.
* Added **constructor overload** for the class **Position** that takes all the information for it to make deep-copying of the **Engine**’s **gameField** easier (the properties are currently read-only).

1. Created new method **ReadCommand()** which process the input data from the console. Moved all data checks from **Reveal()** to **ReadCommand()**. Extracted the switch from **Start()** into separate method **ExecuteCommand().**
2. Removed **Command** parameter from **Engine.**
3. Separated rendering logic from **Engine** to **Renderer**
4. Implemented Factory Method Pattern:

* **Creator**: Defines abstract object creation;
* **GameCreator**: Implements **Creator** with concrete types;

1. Added **IGameFieldSave** interface
2. Added the new method **SetNewField()** in **GameField** in order to avoid re-instantiating it (**GameField**) in **Engine**
3. Updated **ClassStructure.cd**
4. Structured files in folders (namespaces) by their respective type and functionality as follows:

* **ScoreBoard** and **ScoreRecord** in *Scoring*; **IScoreRecord** in *Contracts* subfolder;
* **GameFieldSave** and **GameFieldMemento** in *Saving*; **IGameFieldSave** in *Contracts* subfolder;
* **Renderer** in *Rendering*; **IRenderer** in *Contracts* subfolder;
* **Creator** and **GameCreator** in *GameFactory*;
* **ClassStructure.cd** in *StructureDiagrams*;

1. Extracted repeating code for adding a high score in the methods **GameOver()** and **GameWon()** in the class **Engine**. Named the new method – **RecordResult()**.
2. Added unit tests for the **ScoreBoard** and **ScoreRecord** in **ScoreTests** class.
3. Added unit tests for **Position** class.
4. Added unit tests for **GameField** class.
5. Added a console clearing in **RenderGameField()**. Added **RenderSaveDone()** method when saving. At game end, asks for a command instead of starting a new game – see method **GameOver()** in **Engine()**.
6. Extracted the logic for the commands in a separate class – **ControlManager()**, which implements the interface **ISaveControls**. This interface has a more base version called **IBaseControls**.
7. The **Creator** class now holds the Boolean parameters **IsNewGame, IsGameOver** and **IsGameOn**, as well as an instance of **GameField** which is to be used by everyone.
8. Moved all the newly added parameters (pt. 39) in **Creator** to a new separate class **GameStateManager**.
9. Separated user input functionality from **Engine** (check *UserInput* folder)
10. Added few new messages to **Renderer**