# Refactoring Documentation for Project “Balloons Pops”

**Team "Sodium"**

1. Redesigned the project structure:
   * Renamed the project to **Balloons-Pops**.
   * Renamed the namespace from **Balloons\_Pops\_Game** to **BalloonsPopsGame**.
   * Renamed the main class **Program** to **BalloonsPops**, and the file as well.
   * Renamed the struct **structOfRow** to **Score**.
   * Extracted each class in a separate file with a good name: **Score.cs** , **Board.cs,**
   * Added public modifiers to the methods in **Board.cs,** and to the class itself**.**
2. Reformatted the source code:
   * Removed all unneeded empty lines, e.g. in the method **printMatrix()**.
   * Removed unneeded useless comments in the code.
   * Inserted empty lines between the methods.
   * Split the lines containing several statements into several simple lines, e.g.:

|  |  |  |
| --- | --- | --- |
| **if (input[i] != ' ') break;** | **🡪** | **if (input[i] != ' ')**  **{**  **break;**  **}** |

* + 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**.
  + Introduced new variables in for loops to increase readability
  + Change the places of two nested for loops.
  + Removed useless braces in for loop header.
  + Changed from type **byte** to **int** (not to be limited if the game grows)
  + Formatted all other elements of the source code according to the best practices introduced in the course “[High-Quality Programming Code](http://codecourse.telerik.com/)”.
  + …

1. Renamed variables:
   * In class **Fifteen**: **number** 🡪 **numberOfMoves**.
   * In **Main(string[] args)**: **g** 🡪 **gameFifteen**.
   * In **PlayGame**() : **userMoves 🡪 numberOfMoves**
   * In Doit – CHANGE THAT – **columnLenght** 🡪**rowsLength**
   * In struct **Score: Value -> value, Name -> name.**
2. Renamed methods:
   * In class **Board** : **Gen** 🡪 **Generate.**
   * **In class Board: PrintMatrix()**🡪 ToString()
   * **In class Board: Doit()**🡪 **IsEmtpy**
3. Introduced constants:
   * **ScoreBoardSize = 5**
   * **GameBoardRows = 5**
   * **GameBoardCols = 10**
   * **StartColorRange =1**
   * **EndColorRange =5**
   * …
4. Introduced class **ScoreBoard** and moved all related functionality in it.
5. Introduced class **Coords** and created properties Row and Col.
6. Extracted and moved method **GenerateRandomNumber(int start, int end)** from **Board** class to separate public static class **RandomUtils**.
7. Moved field **byte[,] temp** from method **Generate()** to private field and change it to **GameObject [,] Field** property**.**
8. Introduced class **GameObject** with Coordinates and numeric value.
9. Create **Board** constructor : **Board(int** gameBoardRows**, int** gameBoardCols, **int** startRange, **int** endRange**),** where we use properties for the two parameters, create the **field** with the given dimensions and generate new random board.
10. Refactored the struct Score by placing “this” where needed and renamed some variables using good programming practices.
11. Added operators **==**, **!=** to **Coords** class, and all the methods needed for that.
12. In **Score** changedpublic fields to properties.
13. Changed static methods to non-static.
14. Replaced the logic from the **byte[,] matrix** with the new array with **GameObjecs. This includes : From all method was remove the matrix array of byte and replace with Board where needed.**
15. Changed the method PrintMatrix() to ToString() and replaced all the Console.WriteLine with StringBuilder and methods from that class.
16. Created class **GameEngine,** the place where the game can be started with method **StartGame**(), which returns new **GameEngine**. Its constructor starts a new game with the method **StartNewGame**().
17. Added method **PlayGame**() called by the StartNewGame method which begins the game.
18. Introduced new Interface IRenderable with methods **Display(string textToDisplay)** and **Read().**
19. **Created new class ConsoleRenderer to implement the interface IRenderable and to display on the windows console.**
20. Replaced all the logic from the methods **Check** –**Up** –**Down** –**Left** –Right with one recursive depth search algorithm implementation method called – **PopEqualNeighborObjects.**
21. Separated method **Change**(**userRow, userColumn) to two methods – CanPopObjects – which returns bool, and PopObjects which is void.**

|  |  |  |
| --- | --- | --- |
| **if** (Change(userRow, userColumn))   {  . . .  } | **🡪** | **bool canPopObjects = CanPopObjects(userRow, userColumn);**  **if (canPopObjects)**  **{**  **PopObjects(userRow, userColumn);**  **}** |