# 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**.
  + Placed the “this” modifier where needed.
  + 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 **GameEngine**() : **userMoves 🡪 numberOfMoves**
   * In **IsEmpty**()– **columnLenght** 🡪**rowsLength**
   * In **IsEmpty**() – **isWinner** 🡪 **isEmpty**
   * **In MoveObjectsDown() – i** 🡪 **colPos**
   * **In MoveObjectsDown() – j,k** 🡪**rowPos**
   * **In GameEngine – userRow** 🡪 **rowPosition**
   * **In GameEngine – userColumn** 🡪 **colPosition**
   * 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 & MoveObjectsDown**()
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. Extracted and moved method **GenerateRandomNumber(int start, int end)** from **Board** class to separate public static class **RandomUtils**.
6. Moved field **byte[,] temp** from method **Generate()** to private field and change it to **GameObject [,] Field** property**.**
7. Introduced class **GameObject** with Coordinates and numeric value.
8. 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.
9. Changed static methods to non-static.
10. 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.**
11. Changed the method PrintMatrix() to Board.ToString() and replaced all the Console.WriteLine with StringBuilder and methods from that class. Introduced two private helping methods : 1) GetHorizontalBorderAsString(), 2) GetColumnIndecesAsString().
12. Separated the method Doit to two methods- **IsEmpty**() and **MoveObjectsDown**().
13. In **MoveObjectsDown**() fixed the logic where the workflow of the method was using try catch block to work, and replaced it which better named variables and new for loop and moved it to **GameEngine** class..
14. 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**().
15. Added method **PlayGame**() called by the StartNewGame method which begins the game.
16. Introduced new Interface IRenderable with methods **Display(string textToDisplay)** and **Read().**
17. **Created new class ConsoleRenderer to implement the interface IRenderable and to display on the windows console.**
18. Replaced all the logic from the methods **Check** –**Up** –**Down** –**Left** –Right with one recursive depth search algorithm implementation method called – **PopEqualNeighborObjects.**
19. 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)**  **{**  **this.PopObjects(userRow, userColumn);**  **this.board.MoveObjectsDown();**  **}** |

1. Change the class ScoreBoard and removed the class Score by replacing it with data structure **OrderedMultiDictionary<int, string>** named – **scoreboard,** which keeps for each result in it automatically sorted by the key int- which is the players result and the value is the player names. For each players moves in the game there can be multiple names.
2. Changed the ToString() method of the class **ScoreBoard**.
3. Spliced the method **signIfSkilled**(string[,] Chart,int points) new methods in class **ScoreBoard -** **IsTopPlayer**(int numberOfMoves), to check if the player should be added to the scoreBoard and method AddPlayer (string playerName, int numberOfMoves) which adds a player to the scoreboard.
4. Replace all Console.WriteLine and ReadLine with instance of the class **ConsoleRenderer** and the methods **Display(string textToDisplay)** and **Read().**
5. Refactored the logic to check the user input with shorter one.
6. Refactored the game while loop – to do-while, and separated it to different methods : PlayGame(), CanPopObjects()[[1]](#footnote-1), ProcessPlayerByResult(),AddPlayerToScoreBoard(), PopEqualNeighborObjects(), PopObjects(), MoveObjectsDown()
7. Created new class GameEngineUtils for methods that can help the work of the GameEngine but are not required in the class itself.

1. [↑](#footnote-ref-1)