**Refactoring Documentation for Project “Labyrinth”**

1. Redesigned the project structure: Team “Labyrinth-5”

* Renamed the project to Labyrinth5.
* Renamed the main class Program to GameUI.
* Separated the game logic into separate classes, each class in a separate file with a good name: Player.cs, ScoreboardManager.cs, CommandInterpreter.cs etc.
* Separated project into Labyrinth5.Common class library and Labyrinth5.UI console application
* Introduced Contracts namespace for the applications interfaces.
* Introduced Engine namespace for the game logic,
* Introduced Command namespace for the game commands.
* Introduced MazeComponents namespace for the maze generation components.
* Introduced Cells and Generators namespaces.

1. Reformatted the source code:

* Removed all unneeded empty lines, e.g. in the MakeAtLeastOneExitReachable () method.
* Inserted empty lines between the methods.
* Split the lines containing several statements into several simple lines.
* Refactored long arguments.
* 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”.
* Implemented Unit tests for the applications methods.
* Documented all classes, methods and fields with appropriate documentation headers.

1. Renamed variables:
   * Renamed long variables and methods, e.g. MakeAtLeastOneExitReachable()

* Renamed all variables and methods according to the best practices introduced in the course “High-Quality Programming Code”.

1. Introduced constants:

* Extracted all command words as constants.
* Extracted all error messages as constants.
* Introduced a char representation of game elements as constants.

1. Extracted the method Main() from class LabyrinthTest.
2. Introduced class ScoreboardManager and moved all related functionality in it.

* Introduced an external save.

1. Introduced class Player and moved all related functionality in it.
2. Introduced class CommandInterpreter and moved ExecuteCommand() method in it.

* Renamed method ExecuteCommand() to ParseAndDispatch().
* Implemented the Facade Design Pattern ConsoleEngine -> CommandInterpreter.
* Implemented Command Design Pattern.

1. Introduced ConsoleEngine class.

* Implemented Singleton Design Pattern.

1. Introduced Interfaces: ICommand, ICommandInterpreter, IEngine, IMazeCell, IMazeGenerator, IRandarable, IRanderer; to achieve better abstraction and allow future extension of the application.
2. Introduced MazeCell class.
3. Introduced MatrixCoordinates structure.
4. Removed int[] dirX and dirY and introduced Directions class to hold the directions as MatrixCoordinates objects.
5. Introduced Maze class.

* Introduced optional maze generation strategy .
* Introduced custom maze dimensions.
* Implemented Strategy Design Pattern through SetGenerationStratgy() method.
* Introduced two Maze generator strategy classes that implement IMazeGenerator interface.

1. Introduced ConsoleRenderer class to separate the game components rendering logic from their respective classes.

* Implemented Bridge Design pattern through the classes implementing the interfaces IRenderable-IRenderer.