Refactoring Documentation for Project "Labyrinth"

High Quality Programming Code
Course Teamwork Project
Telerik Academy (2013/2014)

Team "Labyrinth-5"

- *Martin Dzhonov (martin.dzonov)*
- *Denis Kyashif (stinger907)*
- *Qnko Vanov (IankoVanov)*
- *Kristiyan Petrov (kris4o1993)*
- *Iskra Popova (iskra_i_popova)*
- *Karim Hristov (Flyer) – didn't participate*
- *Vesselin Bozhkov (vesselinbozhkov) – didn't participate*

Projects repository: https://github.com/stinger907/Labyrinth-5

1. Redesigned the project structure:
   - Renamed the project to Labyrinth5.
   - solution file version updated from Visual Studio 2010 to 2013
   - .NET Framework project version updated from 4.0 to 4.5.1
   - packages updated; (PowerCollections.dll)
   - 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 Commands namespace for the game commands.
   - Introduced MazeComponents namespace for the maze related classes.
   - Introduced Cells and Generators namespaces.
2. Reformatted the existing source code:

- Removed all unnecessary empty lines, (e.g. in the MakeAtLeastOneExitReachable() method).
- Inserted empty lines between the methods.
- Split the lines containing several statements into several simple lines with single statement on each.
- 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".
- Documented all classes, methods and fields with appropriate documentation headers.

3. Renamed variables:
    - Given variables and methods, (e.g. MakeAtLeastOneExitReachable()) more concise, yet descriptive names (SetExit()), py – Player.Position.Row , dirX, dirY – Directions(Up,Down,Left,Right) .
    - Renamed all variables and methods according to the best practices introduced in the course "High-Quality Programming Code".
4. Introduced constants:
    - Extracted all command words as constants (e.g. const InitializeGameCommand = "init").
    - Extracted all error messages as constants. (e.g. const StrategySwitchedMessage = "Generation algorithm set to : {0}")
    - Introduced a char representation of game elements as constants. (e.g. WallImage = '\u2593')
5. Improved Access modifiers' consistency. Switched all assembly-related members to internal.
6. Introduced class ScoreboardManager and moved all related functionality in it.
    - Introduced an external save.
7. Introduced class Player and moved all related functionality in it.
8. Introduced class CommandInterpreter and moved ExecuteCommand() method in it.
    - Renamed method ExecuteCommand() to ParseAndDispatch().

- Implemented the relation of ConsoleEngine -> CommandInterpreter as Facade Design Pattern, where CommandInterpreter class encapsulates the complex application logic and exposes only the ParseAndDispatch() method.
- Implemented Command Design Pattern as a supplement to the CommandInterpreter. Internal changes in class' objects are handled by separate ICommand classes.

9. Introduced ConsoleEngine class.
    - Implemented it as Singleton Design Pattern.
10. Introduced Interfaces: ICommand, ICommandInterpreter, IEngine, IMazeCell, IMazeGenerator, IRandarable, IRanderer; to achieve better abstraction and allow future extension of the application.
11. Introduced MazeCell class.
12. Introduced MatrixCoordinates structure.
13. Removed int[] dirX and dirY and introduced Directions static class to hold the directions as MatrixCoordinates objects.
14. Introduced Maze class.
    - Introduced optional maze generation strategy(IMazeGenerator).
    - Introduced custom maze dimension setting option.
    - Implemented Strategy Design Pattern through SetGenerationStratgy() method.
    - Introduced two Maze generator strategy classes that implement IMazeGenerator interface.
15. 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(abstraction)-IRenderer(implementor). Among the refined abstractions are classes that implement IRenderable(i.e. Maze, Player) and ConsoleRenderer is the existing concrete implementation.
16. Introduced Labyrinth5.Tests unit testing project and implemented unit tests for the application's components with over 80% code coverage reached.