Labyrinth Game

Refactoring Documentation

By Team "Labyrinth-4"

1. Redesigned project structure

- Renamed project to **LabyrinthGame**
- Extracted all classes into separate files
- Classes arranged in separate folders
- Interfaces arranged in separate folders
- Extracted main logic as separate assembly /class library/
- Extracted game demo into separate console application
- Unit test project added to the solution
- New labyrinth functionality added

2. Reformatted source code

- Removed all unneeded empty lines
- Separate methods with an empty line
- Empty line added after each closing } /curly bracket/ to separate logic
- Split lines containing long statements
- Long if conditions splitted into separate bool values in order to debug easily
- 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".

3. Renamed variables and identifiers

- Variables with Cyrillic encoding letters changed into Latin letters
- Variables renamed appropriate to their use
- Methods renamed appropriate to their use
- Classes renamed appropriate to their use

4. Constants

- Every magic number is put at the class beginning as a constant
- Every magic string is put at the class beginning as e constant
- All fields that are not changed in properties are made read-only

5. Class refactoring

- Each class is glued to the Single responsibility principle.
- Logic not typical for the current class extracted into new class
- Abstract class Labyrinth introduced as parent of all labyrinths
- Access modifiers introduced to all classes

6. Interfaces introduced

- Introduced interfaces for every class in order to stay Open/closed
- Interfaces used with the Strategy Pattern

7. Methods refactoring

- Single responsibility principle
- Long methods shortened to e screen scroll
- Method logic not appropriate to the method name extracted into separate method
- Access modifiers introduced to all methods

8. Design patterns introduced

- Simple Factory
- Façade
- Bridge Pattern
- Singleton
- Mediator
- Strategy

9. Other features

- SOLID, DRY, KISS, YAGNI principles
- IoC used
- Poor man's DI
- Mocking used in unit testing

10. Frameworks used

- Moq – mocking objects
- Ninject – registering dependency