Introduktion

En laboration som går ut på att refaktorisera och utföra tester till spelet Moo som innehåller tydliga code smells och är skriven i programmeringsspråket C#.

Utförande

Separera kod från ursprungskoden

Spelet Moo hade all kod i en enda konsol fil och det som gjordes först var att separera och dela upp koden i flera mindre cs-filer. Utifrån ursprungskoden kunde separation av spellogik och spelkontroll, samt data på spelaren och statistik göras. Cs-filerna döptes till GameLogic, GameController, PlayerData, Leaderboard och sedan skapades även en för *UI* för konsolintegrationen det vill säga användargränssnittet, samt *RandomNumberGenerator*. Med det kom även motsvarande namn i Interfaces som klasserna ovan ska implementera. Dessa filer lades sedan till i tre mappar Interfaces, Models och Services. I Interfaces ligger alla interfaces, i Models ligger *PlayerData* och i Services ligger resterande. Tanken med projektstrukturen med mapparna och namngivningen på filerna är att de ska vara generiska nog att kunna återanvändas och implementeras i framtida spel.[1]

Refaktorisering

PlayerData ändrades knappt och för det mesta ser likadant ut. Anledningen är att det är rätt tydligt vad den klassen gör och metodernas funktion. Förutom namnet på *makeGoal()* ändrades endast namnet på några variabler. Namnet ändrades till *RandomNumberGenerator* som är mycket tydligare och fick vara i en separat fil som kallas på i spelkontrollen varje gång ett nytt spel påbörjas. Viktigt att ge meningsfulla namn. [2]

I C# namnges klasser, interfaces et cetera enligt namnkonventionen Pascal Case och inte Camel Case. Därför har alla metoder stora bokstäver och *showTopList()* blev därför *ShowTopList()*.[3] Den metoden flyttades till klassen Leaderboard, samt en bit kod från spellogiken som blev metoden *SaveResult()* som använder sig av StreamWriter för att spara ned ett resultat i textfil lokalt.

I klassen GameLogic ändras namnet på metoden för spellogiken från *checkBC()* till *CheckBullsAndCows()* för att förtydliggöra namnet. Det skapades en ny metod *CheckValidGuess()* som kollar om användaren har angett endast fyra siffror och inga andra tecken, samt att alla siffror är unika med hjälp av Regex och Distinct, samt Count.[4] Det mesta som har refaktoriserats är i spelkontrollern det vill säga GameController så 39 rader kod blev 13 rader, se nedan.

```
public void RunProgram()
{
    PromptForName();
    bool playOn = true;

    while (playOn)
    {
        string goal = StartNewGame();
        int numberOfGuesses = GamePlay(goal);
        RecordResult(numberOfGuesses);
        playOn = AskToPlayAgain();
        Console.Clear();
    }
}
```

[1] Projektstruktur enligt tidigare projekt av ITHS lärarna
[2] Robert Cecil Martin, Meaningful Names, Clean Code A Handbook of Agile Software Craftmanship
[3] Microsoft, C# identifier naming rules and conventions, https://learn.microsoft.com/en-us/dotnet/csharp/fundamentals/coding-style/identifier-names
[4] Microsoft, Regular Expression Language - Quick Reference, https://learn.microsoft.com/en-us/dotnet/standard/base-types/regular-expression-language-quick-reference

Det skapades flera nya metoder, men det blev mycket tydligare vad koden faktiskt gör när programmet startas. Likaså är metoderna tydliga efter separationen.

Tester

Utförandet av testerna har gjorts enligt Youtube-videon ”Unit Testing C# Code - Tutorial for Beginners”. Namngivningen för testmetoderna har gjorts enligt denna konvention public void *MetodensNamn_Scenario_FörväntadBeteende*. [5] I varje testmetod har mönstret AAA (Arrange, Act, Assert) använts för att klargöra vad som gör vad i testmetoderna. I kombination med det har även ramverket Moq använts i testerna. Det här mocking ramverket simulerar en miljö där metoderna ska kunna fungera som de vanligtvis gör och tänkt att göra, därmed förbättrar och förenklar utförandet samt kontrollen av testningen.[6]

Framtida förbättringar

Tidigare har det nämnts att namngivning och projektstruktur hade gjorts med en tanke bakom sig, nämligen att kunna återanvändas för att kunna användas i framtida spel. I det här fallet är koden fortfarande för specifikt för spelet Moo, därför ligger alla mapparna gemensamt i mappen MooGame. Det finns fortfarande metoder i interfaces koderna som tar emot parametrar, vilket gör det svårare implementera på andra spel. Detta skulle kunna fixas genom att refaktorisera koden ytterligare så att metoden ej behöver ta emot parametrar, men att metoden fungerar som innan. Att minimera parametrar är en del av Clean Code. [7]

[5] Youtube, Unit Testing C# Code – Tutorial for Beginners, https://www.youtube.com/watch?v=HYrXogLj7vg
[6] Code Magazine , Using Moq: A Simple Guide to Mocking for .NET, https://www.codemag.com/Article/2305041/Using-Moq-A-Simple-Guide-to-Mocking-for-.NET
[7] Robert Cecil Martin, Function arguments, Clean Code A Handbook of Agile Software Craftmanship