Labbrapport - Clean Code Laboration “Smells”

Av Isac Arvidsson Rosenberg

Uppgift

Att refaktorera given kodimplementation av spelet “Moo Game” för att göra det skalbart och möjliggöra implementering av ett eller flera liknande spel, samt skriva Unit tests för en del av metoderna i programmet.

Strategi

Efter att ha studerat hela uppgiften, inklusive kraven för att uppnå VG, kändes det mest relevant att bygga direkt mot VG istället för att komplettera i efterhand eftersom uppgiften skulle utförts annorlunda beroende på om programmet skulle tillåta fler spel eller ett.

Jag valde att strukturera upp projekten efter Clean Architechture-modellen med tre lager:

- App: Användargränssnittlagret
- Infrastructure: Controllers och repositories
- Core: All spellogik

Referens: https://blog.cleancoder.com/uncle-bob/2012/08/13/the-clean-architecture.html

Utförande

Player Score

Det första jag gjorde var att extrahera all funktionalitet som sköter skrift och läsning av spelartabellen och implementerade ett repository som utför all kontakt med vald datalagringstyp, i detta fall StreamWriter. Logiken som hanterar hur spelardatat räknas ut, parsas och formateras sköts av en ScoreService som ligger i Core-lagret och är separerat från repositoryt för att enkelt kunna byta ut vilken typ av datalagring man vill använda i framtiden.

Parsning för playerscore i ScoreService har jag skrivit om till Linq-uttryck för bättre läsbarhet, underhållsvänlighet och hållbarhet samt för att hålla det till mer C#-modernitet.

Game Engine

Med tanke på att flera spel skulle implementeras så valde jag att bygga spelmotorn efter strategy-mönstret. Jag kom fram till att all logik som skiljer dessa typer av spel åt, är en rad properties samt algoritmen om hur man returnerar svaret på gissningarna. Detta medförde att det var enkelt att göra ett IGame-interface där jag kan ladda in varje spel enskilt i spelmotorn. Motorn hanterar properties från respektive spel genom IGame-interfacet och utför logiken efter det. Jag extraherade logiken för spel-loopen, gjorde den mer generisk för att kunna hantera flera olika spel och gjorde så att svars-responsen hanteras av det enskilda spelet. Liknande med mål-generatorn, där det enskilda spelets properties bestämmer hur många karaktärer målet skall ha samt om det kan ha repeterande karaktärer.

Referens: https://refactoring.guru/design-patterns/strategy

Dependency Injection

För att motverka framtida möjliga flertals instanser av UI och repository skapade jag en service collection som i sin tur skapar singleton-instanser av dessa. Detta gör det enkelt att sedan i program.cs kunna byta ut och hantera vilka typer av UI och repository man vill använda i applikationen. Jag valde att även injicera en ILogger för att enkelt kunna skriva ut läsbara felmeddelanden från kastade exceptions.

Referens: https://refactoring.guru/design-patterns/singleton

User interface

Användargränssnittet i det här fallet är endast konsollen men eftersom det ärver av ett interface "IIO" med en input och output går det lätt att byta ut.

Jag valde att skapa en meny där man kan välja spel genom att använda piltagenterna. Meny-klassen består av olika delar, en som läser namnet på spel-klasserna från assembly, en som innehåller loopen för att välja spel samt en continue-metod för om man vill fortsätta spela.

Felhantering

Implementation av felhantering har skett på de metoder som har möjlighet att gå fel. Alla try-catch-block går upp i en kedja hela vägen till controllern som hanterar felmeddelandet och loggar det via loggaren. Där try-catch används, används det för hela metoden, som kursboken förespråkar.

Extraspel

Mastermind var det extraspelet jag valde att skapa. Eftersom jag redan hade förberett hela programmet för att kunna skapa liknande spel var detta en enkel process. Det enda jag behövde göra var att fylla properties med den info som spelet kräver samt att skriva svarsresponsalgoritmen.

Testning

Testning utförs mestadels på GetGuessResponse-metoden på båda spelen, täckande korrekta svar, inkorrekta svare, icke matchande m.m. Har även skapat en MockGoalGenerator som delar interface med GoalGenerator för att kunna testa SetTargetGoal-metoden i GameEngine.

Övrigt

Genom hela processen har jag varit noga med att ha kursboken på andra skärmen för att enkelt kunna skriva koden så rent som möjligt. Flytande har jag tagit in information från alla kapitel och implementerat det i refaktoriseringen. Hela kapitel 2 har absolut varit viktigast, just för att lära mig namnge variabler och metoder bra. Jag har försökt hålla koden efter ett citat från sidan 67 "Don't use a comment when you can use a function or a variable" och därmed är koden extremt kommentarsfattig. "Single responsibility" är ett återkommande citat som även det väger tungt för mig.

Hoppas koden är tillräckligt ren, skalbar, funktionerlig och lättläst!