

Projektuppgift

Programmering i C#.NET

Quiz

Hund-quiz

Adela Knap

MITTUNIVERSITETET
Institutionen för data- och elektroteknik(DET)

Författare: Adela Knap, adkn2300@student.miun.se

Utbildningsprogram: Webbutveckling, 120 hp

Huvudområde: Datateknik

Termin, år: Ht, 2024

Sammanfattning

I det här projektet har en C# .NET-konsolapplikation i form av ett interaktivt hund-quiz skapats. Spelaren får en fråga i taget, och vid varje svar ges feedback på om svaret är rätt. Vid fel svar skrivs det korrekta svaret ut. Spelomgången är tidsatt, och vid quizets slut presenteras en sammanställning av antal rätt och tiden. För att engagera spelare finns en topplista med de tio bästa resultaten. Applikationen har även en administrationsfunktion för att hantera frågor och radera topplistan.

Innehållsförteckning

Sammanfattning	iii
1 Introduktion	1
2 Teori	2
3 Metod	4
4 Konstruktion	5
5 Resultat	12
6 Slutsatser	14
7 Källförteckning	16
8 Bilagor	17

1 Introduktion

I det här projektet ska en C# .NET-konsolapplikation skapas i form av ett hundquiz. Under spelet ska en fråga åt gången visas, där spelaren svarar och sedan får feedback om svaret är rätt eller fel. Vid fel ska det korrekta svaret visas. Varje spelomgång ska gå på tid och efter avslutad omgång visas antal rätta svar och tid. En topplista ska visa de tio bästa resultaten, med spelarens namn, poäng och tid. Det ska finnas en administrationsfunktion för att hantera frågor och radera topplistan. Vid start visas en meny för att navigera mellan applikationens delar. Ett flödesschema ska skapas för en tydlig bild av applikationen. Följande problem behöver lösas:

- Skapa ett flödesschema över applikationen
 - Syfte: För att få en tydlig bild över applikationens flöde och vilka klasser/metoder som behöver skapas
- Dela upp koden i olika klasser, och filer, utifrån ”ansvarsområde”
 - Syfte: För att skapa en objektorienterad kod där varje klass kommer att ha ett specifikt ansvar så att koden blir mer organiserad
- Skapa en meny, samt undermeny, för quizet
 - Syfte: Så att användaren kan navigera i applikationen på ett smidigt sätt
- Skriva kod för delen med spelets gång, med frågorna och svaren
 - Syfte: För att spelet ska gå att spela, själva spellogiken
- Skriva kod för tidtagningen och att visa resultatet i varje spelomgång
 - Syfte: Men tidtagning blir det ytterligare en ”spel-faktor” att utmana spelaren med utöver antal rätt svar
- Skriva kod för en topplista med de tio bästa resultaten
 - Syfte: För att skapa lite prestation i spelet så att spelaren vill spela fler gånger och utmana sig själv
- Skapa kod för att kunna administrera spelet
 - Syfte: Så att det går att hantera spelets frågor/svar och topplistan

2 Teori

Här nedan förklaras kort några begrepp, och förkortningar, som används i rapporten.

C#

C# är ett modernt och populärt programmeringsspråk, skapat av Microsoft, som används för att bygga olika program och applikationer. C# är plattformsoberoende och har en tydlig struktur där det är noggrant med att hålla koll på datatyper, hårt typat. Dessutom funkar C# bra både för objektorienterad och funktionell programmering. Ofta används språket ihop med .NET-ramverket. [1]

.NET

.NET är ett ramverk, som är plattformsoberoende med öppen källkod, som används för att bygga och köra olika typer av program/applikationer. Det fungerar på flera programmeringsspråk men där C# är det mest populära. Det finns ett stort urval av bibliotek och verktyg som gör det enklare att utveckla olika program/applikationer. En viktig del av .NET är Common Language Runtime (CLR), som hanterar körningen av .NET-program genom att kompilera koden och sköter minnes- och felhantering. [2]

OOP

Objektorienterad programmering (OOP) är ett sätt att strukturera koden utifrån objekt, där varje objekt har egenskaper och metoder, och på så sätt kapslas in. Klasser används för att skapa objekten och koden blir då lättare att återanvända och organisera. C# är ett objektorienterat språk. Klasser kan ha konstruktörer som används när en instans av klassen skapas. [3]

Konstruktör (konstruktor)

När en instans av en klass skapas anropas dess konstruktör. En klass kan ha flera konstruktörer som tar olika argument. Konstruktörer gör det möjligt att sätta standardvärden och kontrollera hur och när instanser skapas. Det gör det lättare att anpassa koden och även lättare att läsa. [4]

Namespace

Ett namespace i C# är som en "inkapsling", eller låda, där man skriver klasser, metoder och variabler som hjälper till att organisera koden. Genom att använda namespace kan man strukturera sina projekt, och gruppera relaterade klasser, på ett internt sätt och samtidigt på ett externt sätt och blir åtkomliga för andra program. [5]

Serialisering/ deserialisering

Serialisering syftar till processen att omvandla objekt till ett format som kan lagras, JSON-format i det här projektet som sparas till en JSON-fil.

Deserialisering är det motsatta och med andra ord när data från ett JSON-format, som från en JSON-fil, ska omvandlas tillbaka till objektet. Detta gör det möjligt att på ett enkelt sätt spara data i ett läsbart format och sedan enkelt kunna återställa det igen när det behövs. [6]

LINQ (Language Integrated Query)

LINQ (Language Integrated Query) är en frågefunktion i C# som gör koden lite enklare att förstå och mer likt syntaxen av en "frågeställning" i SQL databaser. Med LINQ kan operatorer som till exempel "Where", "Select" och "OrderBy" användas och på sätt kunna filtrera eller gruppera data på ett enkelt sätt. [7]

3 Metod

Applikationen kommer att utvecklas och köras i C# med .NET och all kod kommer att skrivas i kodoeditorn Visual Studio Code (VS Code).

Quiz-applikationen kommer att delas upp i flera olika metoder och klasser för att få en tydlig struktur. Varje klass kommer att ha ett specifikt ansvar så att koden blir mer organiserad och enkel att underhålla. En objektorienterad strategi kommer därmed att användas där de olika klasserna delas upp i olika filer/klasser.

För att få en tydlig bild över applikationens uppbyggnad kommer Microsoft Visio användas för att visualisera applikationens struktur, med klasser och metoder, innan implementeringen påbörjades. Ett skriftligt flödesschema kommer också att användas för att beskriva flödet i applikationen, inklusive de olika stegen och klasserna som används för att hantera metoder och användarinteraktioner.

För quiz-frågor/svar och topplistan kommer JSON-filer att användas för att spara data mellan spelomgångar. Detta kommer att göras genom att serialisera och deserialisera objekt och System.Text.Json kommer att användas för att hantera detta.

System.Diagnostics kommer att användas för att hantera tidtagningen i quizet genom Stopwatch.

För versionshantering av koden kommer Git och Github att användas med kontinuerliga push/commits under arbetets gång. Detta för att säkerställa att koden inte "försvinner" om det skulle hända något med datorn eller den lokala enheten.

4 Konstruktion

För att få en tydlig bild över hur applikationen är uppbyggd och fungerar skapades en visuell bild i programmet Visio över vilka klasser och metoder som ingår, se bilaga 1 för bild. Sedan skrevs även ett flödesschema (i textform) där alla delar finns med beskrivna på ett mer ingående sätt, se bilaga 2.

När en tydlig bild fanns som grund skapades ett nytt projekt i VS Code med följande filer:

- Quiz.cs (klass för quizet)
- TopList.cs (klass för topplistan)
- QuizManager.cs (klass med metoder för att hantera frågorna)
- GameManager.cs (klass med metoder som hanterar själva spelet)
- MenuManager.cs (klass med metoder som hanterar menyerna)
- Program.cs (innehållande Main-delen och de olika valen för quizet)

Valet att dela upp koden i olika filer gjordes utifrån vilka delar som hanterar logiken i hela applikationen. Detta för att få en mer överskådlig och mer lätthanterlig kod när till exempel olika ändringar behöver göras. I applikationen används JSON-filer (quiz.json och toplist.json) för att spara och hämta frågor/svar samt topplistan. Med JSON-filerna lagras data även om applikationen avslutas och finns kvar tills nästa gång quizet körs.

Quiz- och TopList klasserna

Koden började skrivas med start i de minsta beståndsdelarna, det vill säga klassen för *Quiz* med själva frågan samt svaret (rasen) som properties. Även konstruktorn för klassen valdes att skrivas här. Ännu en klass skapades sedan för topplistan, *TopList*, där properties för spelarens namn, poäng, maxpoäng och tid skrevs samt en konstruktör.

QuizManager

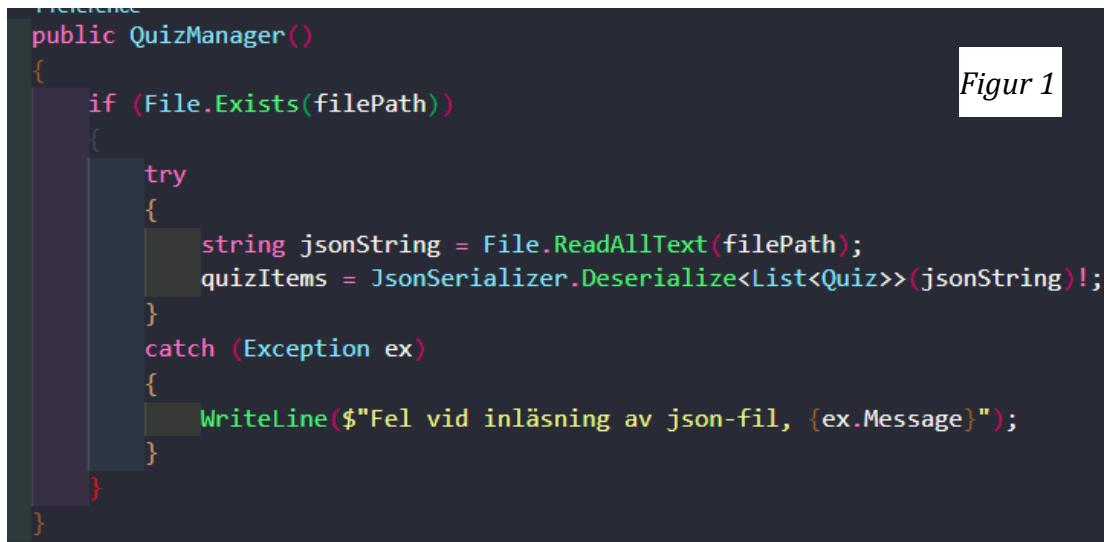
I nästa steg skapades klassen *QuizManager*, i filen QuizManager.cs, där all kod för att hantera frågorna och svaren skrevs. Biblioteket *System.Text.Json* inkluderades för att serialisering och deserialisering till/från JSON-fil samt *System.Console* med static för att slippa skriva Console framför till exempel *Write/WriteLine*. Koden organiserades under namespaceset "quiz". Först skapades en variabel med en privat JSON-fil (quiz.json) där quizet sparas. En privat lista skapades, för att lagra alla frågor samt svar av klassen *Quiz*, och döptes till *quizItems*. En konstruktör skapades sedan

Quiz

Adela Knap

2024-10-31

med kod för kontroll om filen finns, om filen inte finns skapas den automatiskt när data ska sparas, läsa in text och deserialisera från JSON till listan med *quizItems*-objekt. Try/catch valde att användas för att kunna fånga upp eventuella fel med filen utan att hela programmet kraschar, se figur 1.



```
public QuizManager()
{
    if (File.Exists(filePath))
    {
        try
        {
            string jsonString = File.ReadAllText(filePath);
            quizItems = JsonSerializer.Deserialize<List<Quiz>>(jsonString!);
        }
        catch (Exception ex)
        {
            WriteLine($"Fel vid inläsning av json-fil, {ex.Message}");
        }
    }
}
```

Figur 1

Därefter skrevs kod för metoden *AddToQuiz* som skapar ett nytt objekt av klassen *Quiz*, med frågan och det tillhörande svaret, för att sedan lägga till det nya objektet till listan *quizItems*. Till sist sparas listan till JSON-filen med metoden *SaveToJsonFile* (metoden beskrivs längre ner i texten) och returnerar sedan objektet.

I metoden *DeleteQuiz* tas en fråga/svar bort från listan, med den inbyggda metoden *RemoveAt*, utifrån vilket index som anges för att sedan sparas på nytt till JSON-filen med metoden *SaveToJsonFile*. Metoden returnerar sedan det index som har tagits bort.

Nästa metod som skrevs är *GetQuiz* där hela listan, *quizItems*, returneras med frågor/svar. Metoden sätts till publik för att vara åtkomlig för andra metoder i applikationen.

Därefter skrevs koden för metoden *SaveToJsonFile* där *quizItems* – listan serialiseras till JSON och sparas till filen. Även här användes en try/catch för att fånga om något skulle gå fel under processen.

Till sist skrevs metoden *ShowQuiz* där alla frågor och svar skrivs ut numrerade med en *foreach*-loop. En kontroll görs först för att säkerställa att listan inte är tom, ett meddelande skrivs i så fall ut om detta till användaren.

GameManager

Filen *GameManager.cs* innehåller klasser och metoder för att hantera

Quiz

Adela Knap

2024-10-31

spelets gång, poängregistrering och funktionalitet kring topplistan. Även här används bibliotek som tidigare i filen *QuizManager* men med tillägget av *System.Diagnostics*, detta för att kunna använda Stopwatch för att mäta spelarens tid under spelet.

Huvudklassen *GameManager* är kärnan och hanterar spelets funktioner som att lagra resultat, visa topplistan och spela quizet.

Först skapades en variabel med en privat JSON-fil (toplist.json) där topplistan sparas. En privat lista skapades, för att lagra alla resultat av klassen *TopList*, och döptes till *topList*. Varje objekt innehåller spelarens namn, antal rätt svar, maxpoäng och tiden det tog att genomföra spelet. En konstruktor skapades sedan med kod för kontroll om filen finns, läsa in text och deserialisera från JSON till listan med *topList*-objekt. Try/catch valde att användas för att kunna fånga upp eventuella fel med filen utan att hela programmet kraschar.

Därefter skrevs den första metoden, *AttToTopList*, som skapar ett nytt objekt av klassen *TopList* med spelarens namn, poäng, max-poäng för frågeomgången och spelarens tid för att genomföra quizet. Det nya objektet läggs till i *topList*-listan, sparas till JSON-filen och sist returneras det skapade objektet.

Nästa metod, *GetTopList*, returnerar hela topplistan. Metoden sätts till publik för att vara åtkomlig för andra metoder i applikationen.

I metoden *SaveToFile* serialiseras topplistan och sparas i JSON-format i filen (toplist.json). Try/catch används, på samma sätt som när texten från filen läses in.

Metoden *ShowTopList* skriver ut topplistan med de tio bästa resultaten sorterad efter antal rätt och därefter efter tid. Först görs en kontroll om det finns några resultat i listan och därefter används LINQ (Language Integrated Query) för att sortera på ett smidigt sätt, se figur 2 för kod med sorteringen.

```
var sortedTopList = topList
    .OrderByDescending(t => t.Points)
    .ThenBy(t => t.Time)
    .Take(10);
```

Figur 2

Sedan används en *foreach*-loop för att iterera genom listan och skriva ut resultateten i en numrerad lista. Att endast skriva ut de tio bästa var ett val som gjordes för att topplistan inte ska bli för lång att läsa.

Nästa metod, *DeleteTopList*, raderar hela topplistan i ett svep. Detta för att kunna hålla topplistan aktuell när frågor i quizet byts ut, då är de resultat som finns inte längre aktuella och behovet av att kunna radera hela listan

Quiz

Adela Knap

2024-10-31

finns. I metoden kontrolleras först om användaren verkligen är säker på om hela listan ska raderas genom att trycka y (yes) eller n (no). Med `ReadKey` läses valet in och med en `if/else` raderas antingen listan, med den inbyggda metoden `Clear`, och sparas sedan om med `SaveToFile`. Eller om n (no) anges så skrivs ett meddelande ut om att inga ändringar har gjorts.

Sista metoden som skrevs, `PlayQuiz`, är den som hanteras själva spelets gång där spelaren svarar på frågorna och resultatet läggs till i topplistan om spelet slutförs korrekt. Koden ger användaren feedback under spelets gång såsom felmeddelanden om ogiltiga inmatningar och instruktioner om vad de ska göra när quizet börjar och slutar.

Metoden tar emot listan med frågorna/svaren (`List<Quiz> questions`) och först kontrolleras om listan är tom för att i så fall skriva ut ett meddelande om detta och spelet avbryts. Om listan inte är tom uppmanas spelaren att skriva in sitt namn med en `while`-loop för att se till att namnet som anges inte är tomt/whitespace. Vid giltigt namn kör spelet i gång med att tidtagningen startas och en `forEach` som itererar genom frågorna en i taget, skriver ut frågan, och där spelaren ska skriva ett svar. För att spelaren ska kunna avbryta under spelets gång utan att det resultatet sparas kan "x" anges som svar och då bryts spelet, se figur 3 för kod.

```
if (answer != null && answer.Trim().Equals("X", StringComparison.OrdinalIgnoreCase))
{
    quizCancelled = true;
    break;
}
```

Figur 3

`Trim` används för att ta bort eventuella mellanslag och jämförs sedan med strängen `x` oavsett stora/små bokstäver.

Sedan kontrolleras så att svaret är giltigt, om så är fallet får spelaren möjlighet att ange svaret igen till det är korrekt format, eller ange `x` för att avbryta. I nästa steg kontrolleras om det inskrivna svaret är rätt genom att jämföra det med svaret från objektet på liknande sätt som i figur 3.

Om svaret är korrekt ökas poängen och annars skrivs ett meddelande ut om vad som är rätt svar. När alla frågor besvarats stoppas tidtagningen och sätts i en variabel med hur många sekunder som har gått, `timer.Elapsed.TotalSeconds`. Om quizet avbrutits, med "x", skrivs ett meddelande ut om detta med en `if`-sats, inget sparas då, och annars visas spelarens poäng, möjliga max-poäng och tid för att till sist läggas till topplistan med metoden `AddTopList` där ett objekt skapas och sparas till fil.

MenuManager

I *MenuManager* skrevs all kod som rör hanteringen av menyer och de menyval som användaren gör, med andra ord användarinteraktionen. Valet att skriva den här koden i en egen fil gjorde för att få ett mer uppdelat/modulärt upplägg som även är lättare att underhålla och läsa. Dessutom hålls *program.cs* filen mer ren, med mindre kod, och det blir en tydligare bild över applikationens delar samt flöde.

Först skrevs metoden *ShowMenu* som visar huvudmenyn med de olika valen och kallas när användaren kör i gång applikationen. Med samma uppbyggnad skapades sedan metoden *ShowSubMenu* som i sin tur visar undermenyn när användaren väljer att hantera quizet.

Nästa metod, *AddQuestion*, samlar in nya frågor i quizet med out-parametrarna *inputQuestion* och *inputBreed* som är strängar. Därefter kommer en while-loop där användaren ska skriva in en ny fråga samt svar. Loopen fortsätter, med uppmaning om att ange fråga/svar repeteras, tills korrekt fråga och svar anges alltså inte är *nullOrWhiteSpace*. Detta för att undvika att ogiltiga eller tomma frågor/svar kommer in i systemet.

Sedan skrevs metoden *DeleteQuestion* som ansvarar för att ta bort frågor/svar från quizet och tar emot ett objekt av *QuizManager* som argument, vilket används för att visa och hantera frågorna. Först skrivs alla frågor/svar ut för att användare ska kunna välja vilken som ska tas bort med *quizManager.ShowQuiz*. Sedan skrivs en if-sats med kontroll om det finns några frågor alls att skriva ut med ett meddelande om så är fallet, se figur 4.

```
public void DeleteQuestion(QuizManager quizManager)
{
    Clear();
    quizManager.ShowQuiz();

    if (quizManager.GetQuiz().Count == 0)
    {
        WriteLine("Inga frågor finns i quizet. Tryck på valfri tangent för att återgå till undermenyn...");
        ReadKey();
        return;
    }
}
```

Figur 4

Om det finns frågor/svar så får användaren ange ett index för den fråga som ska tas bort. Inmatningen valideras med hjälp av *int.TryParse* för att säkerställa att indexet är ett giltigt heltal och inom rätt intervall. Vid korrekt inmatning tas frågan bort, och användaren får en bekräftelse om vilken fråga som tas bort. Try-catch används för att fånga upp eventuella fel vid borttagningen och en else-sats fångar upp eventuella fel med om fel index

ange, se figur 5.

```
if (int.TryParse(ReadLine(), out int index) && index >= 0 && index < quizManager.GetQuiz().Count)
{
    try
    {
        quizManager.DeleteQuiz(index);
        WriteLine($"Fråga {index} har tagits bort! Tryck på valfri tangent för att återgå till undermenyn.");
        ReadKey();
        break;
    }
    catch (Exception)
    {
        WriteLine("Det blev något fel vid borttagning av frågan. Tryck på valfri tangent för att fortsätta.");
        ReadKey();
        break;
    }
}
else
{
    ErrorMessage();
    Clear();
    quizManager.ShowQuiz();
}
```

Figur 5

Till sist skrevs metoden *ErrorMessage* som visar ett felmeddelande där användaren uppmanas att trycka på valfri tangent för att fortsätta. Metoden skrevs för att slippa upprepning då det är ett återkommande felmeddelande i koden. Vid övriga felmeddelande som behöver preciseras mer skrivs ett sådant ut med *WriteLine*. Genomgående används while-loopar och felhanteringsmetoder för att se till att allt användaren matar in är korrekt innan koden körs vidare.

Program

I *program.cs* hanteras de olika menyvalen mellan användaren och applikationen.

I Main-metoden skrevs först kod för att å, ä och ö ska användas och visas rätt i applikationen och konsolen. I nästa steg skapades nya objekt av *MenuManager* (som hanterar visning av menyerna och valen), *QuizManager* (som hanterar själva quizet med frågor och svar) och *GameManager* (som hanterar själva spelets gång och topplistan). Genom att skapa instanser av dessa klasser fås tillgång till metoderna som behövs för att köra applikationen, se figur 6 för kod.

```
OutputEncoding = System.Text.Encoding.Unicode;
InputEncoding = System.Text.Encoding.Unicode;

MenuManager menuManager = new();
QuizManager quizManager = new();
GameManager gameManager = new();
```

Figur 6

Sedan skrevs en while-loop som innesluter alla valen och som körs

kontinuerligt tills användaren väljer att avsluta programmet med valet 'x'. Först skrevs kod för att visa huvudmenyn med *menuManager.ShowMenu*. Valen i menyn (utöver case x) skrivs sedan ut med en switch-sats med följande case:

- **Case 1 – Spela quiz:**

Frågorna hämtas från *QuizManager* via *GetQuiz* och skickas till *PlayQuiz* i *GameManager*, som hanterar spelprocessen. Spelaren svarar på frågor och får feedback på sina svar.

- **Case 2 – Visa topplistan:**

Här visas de tio bästa resultaten från tidigare spelomgångar via *ShowTopList* i *GameManager*.

- **Case 3 – Hantera quizet:**

En undermeny visas, användaren anger "x" för att återgå till huvudmenyn. Den innehåller följande case:

- **Case 1 – Visa alla frågor:**

Alla frågor/svar visas med *quizManager.ShowQuiz*.

- **Case 2 – Lägg till fråga:**

Inmatade värden för fråga och svar lagras i *inputQuestion* och *inputBreed*. *AddQuestion* samlar in dessa värden och *AddToQuiz* i *QuizManager* lägger till dem i quizet.

- **Case 3 – Ta bort fråga:**

DeleteQuestion i *MenuManager* tar emot ett *QuizManager*-objekt för att radera en fråga utifrån index.

- **Case 4 – Visa topplistan:**

Topplistan visas via *gameManager.ShowTopList*.

- **Case 5 – Radera topplistan:**

Topplistan raderas helt med *gameManager.DeleteTopList*.

Både huvudmenyn och undermenyn hanterar felaktiga inmatningar med *ErrorMessage*. Applikationen avslutas med valet "x" i huvudmenyn, vilket anropar *Environment.Exit(0)*.

5 Resultat

I introduktionen fastställdes några punkter med problem som skulle lösas. Här nedan utvärderas punkterna om målen för varje punkt har uppfyllts.

Skapa ett flödesschema över applikationen

Det skapades både ett flödesschema, i skriftlig form, samt en visuell bild av hur applikationens klasser och metoder är uppbyggd. Den visuella bilden skapades i Microsoft Visio och visar den övergripande strukturen. Därmed är målet uppfyllt.

Dela upp koden i olika klasser, och filer, utifrån "ansvarsområde"

I VS Code skapades olika filer med klasser, med tillhörande metoder, där varje klass har ett eget ansvarsområde. Klassen *QuizManager*, bygger på grundklassen *Quiz*, och hanterar allt som rör frågor/svar. Klassen *GameManager*, som använder grundklassen *TopList*, hanterar själva spellogiken och topplistan. Klassen *MenuManager* hanterar menyerna. Därmed är målet uppfyllt.

Skapa en meny, samt undermeny, för quizet

En huvudmeny samt en undermeny skapades i klassen *MenuManager* med flera olika val för att skapa en tydlig användarinteraktion. Vid alla val görs kontroller för att säkerställa och inga felinmatningar görs. Därmed är målet uppfyllt.

Skriva kod för delen med själva spelets gång med frågorna och svaren & skriva kod för tidtagningen och att visa resultatet i varje spelomgång

I klassen *GameManager* skrevs kod för att hantera spellogiken under spelets gång i metoden *PlayQuiz*. Här hanteras eventuella felinmatningar/val så att koden ska fungera smidigt. I metoden startas tidtagning i samband med att första frågan visas och stoppas sedan när alla frågor i aktuell spelomgång har körts. Här läggs även resultatet till i listan *topList* så att alla resultat finns lagrade. Därmed är båda målen uppfyllda.

Skriva kod för en topplista med de tio bästa resultaten

I klassen *GameManager* skrevs även en metod, *ShowTopList*, där listan med de lagrade resultaten sorteras utifrån poäng, tid och därefter de 10 första resultaten. Sedan itereras listan igenom och skrivs ut. Resultaten lagras i en lista som serialiseras och deserialiseras till och från en JSON-fil. Därmed är målet uppfyllt.

Skapa kod för att kunna administrera spelet

I klassen *QuizManager* skrevs kod för att administrera quizet där det finns metoder för att skriva ut alla frågor/svar, lägga till nya frågor i quizet och ta bort specifika frågor. Quizet lagras i en lista som sedan serialiseras/deserialiseras till och från en JSON-fil. Därmed är uppfyllt.

6 Slutsatser

Det första problemet, eller frågeställningen, jag hade var hur jag skulle organisera min kod och i vilka filer. Jag kom fram till att jag behövde två filer för grundklasserna till frågorna/svaren i quizet och sedan en till topplistan. Utöver dessa valde jag sen att ha en fil som hanterade quizet, *QuizManager*, och sedan en fil för att hantera själva spelet, *GameManager*. Jag funderade på att bryta ut delar av koden i *GameManager*, till en ny fil, då det är ganska mycket kod där. Men valde till sist att låta den ligga kvar då jag tyckte att var en logisk organisation och att koden som finns där "hör samman" (trots en del kod).

Ett annat problem som jag stötte på under skapande av koden i *program*-filen var att det blev mycket kod och ganska rörigt när det fanns meny och undermeny med "inbakad" kod och kontroller direkt i switch-satserna. Jag valde att lösa det genom att skapa en ny fil, *MenuManager.cs* med en *MenuManager* klass. Där la jag själva texten till menyerna samt skapade nya metoder med koden för de case som hade mer mängd kod. Detta gjorde att min program-fil blev mer överskådlig och mer fokuserad på casen i switch-satserna.

Under utvecklingen av applikationen upptäckte jag bitvis funktionalitet som hade varit bra att lägga till. Ett exempel är att jag insåg att en metod för att radera hela topplistan hade varit användbart för att lättare kunna administrera quizet när till exempel frågor byts ut (metoden *DeleteTopList*). Ett annat exempel på funktionalitet som jag kom på under arbetes gång var att det hade varit praktiskt, och användarvänligt, för spelaren att kunna avbryta under spelets gång utan att det resultatet då sparas. Det kommer hela tiden olika insikter om funktionalitet som skulle kunna utvecklas vidare och det är en utmaning att försöka begränsa arbetet, och sortera i vilka idéer som ska realiseras eller inte.

Jag testade även att ha med tiden under spelets gång, så att spelaren hela såg sin aktuella tidsåtgång. Men valde sedan att inte ha med det då jag personligen upplevde det som en stress-faktor som störde upplevelsen. När jag sedan var klar med projektet kom jag på att jag hade kunnat ha det som en valmöjlighet för spelaren, att välja om tiden ska visas eller ej. Så det är en utvecklingsmöjlighet jag tar med mig till kommande projekt.

Applikationen skulle kunna utvecklas vidare på flera sätt och två spår som jag hade funderingar på, men som jag fick släppa för att det inte skulle bli

Quiz

Adela Knap

2024-10-31

en för stor och övermäktig uppgift med tanke på tidsbegränsning, var att ha fler kategorier i quizet samt att ha med ML. Det hade varit kul om användaren/spelaren kunde välja mellan flera kategorier, till exempel vanliga hundraser och ovanliga hundraser, och sedan implementera ML som kunde ge ett omdöme på hur väl man presterar/utvecklas över tid och vad man behöver träna mer på.

Av det här projektet har jag lärt mig att koda med C#.NET och att skapa en fungerade konsolapplikation som fyller ett mer omfattande syfte för en användare. Att skapa ett flödes-schema är något som också är en ny lärdom och som jag tar med mig i kommande projekt. Det ger en tydlig bild av hur applikationen fungerar som helhet med en bra överblicksbild.

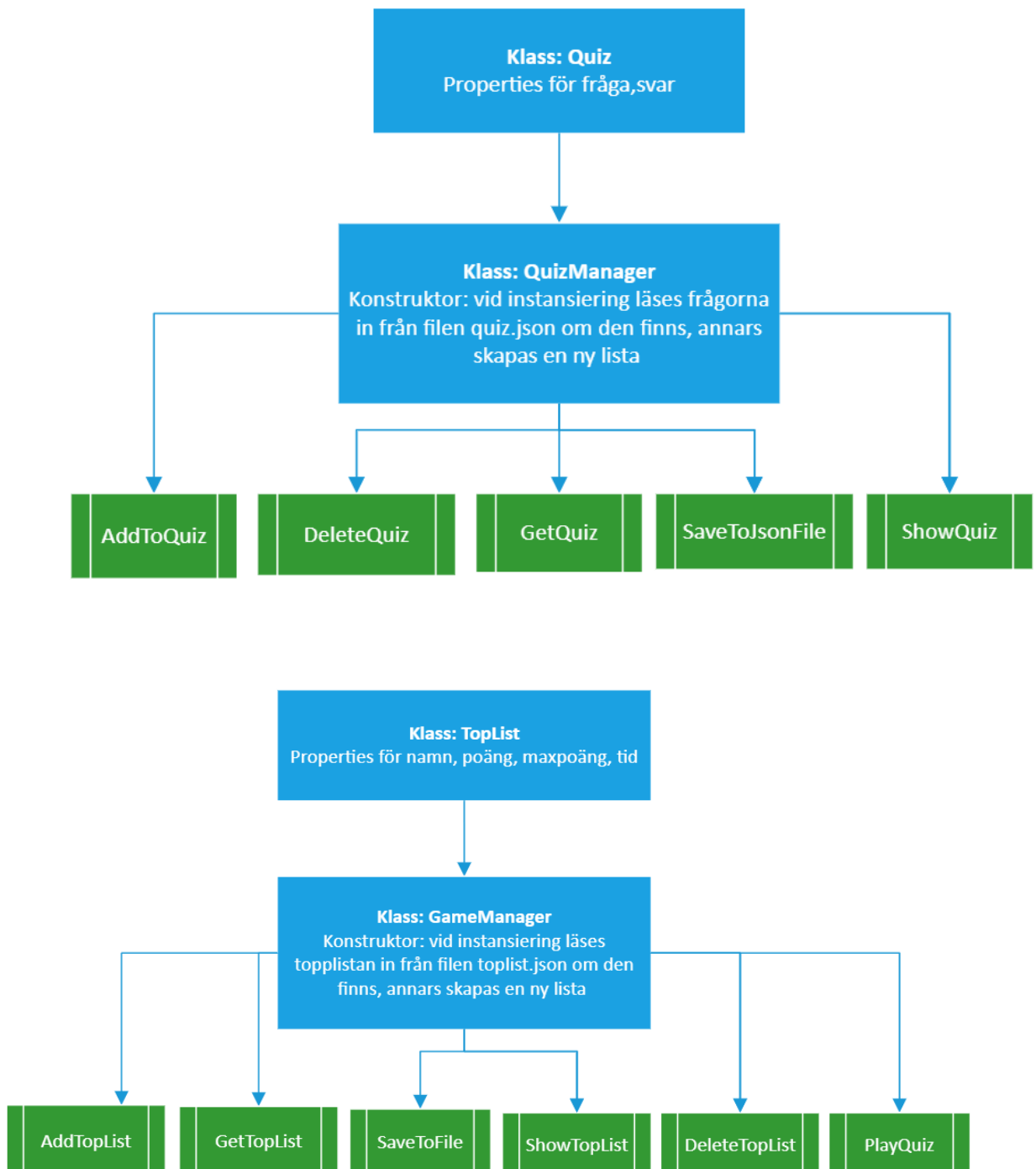
7 Källförteckning

- [1] Microsoft Learn, "A tour of the C# language". <https://learn.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/overview> Hämtad: 2024-10-23
- [2] Microsoft Learn, "Introduction to .NET". <https://learn.microsoft.com/en-us/dotnet/core/introduction> Hämtad: 2024-10-23
- [3] Microsoft Learn, "Object-Oriented programming (C#)". <https://learn.microsoft.com/en-us/dotnet/csharp/fundamentals/tutorials/oop> Hämtad: 2024-10-23
- [4] Microsoft Learn, "Constructors (C# programming guide)" <https://learn.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/constructors> Hämtad: 2024-10-23
- [5] Microsoft Learn, "Namespaces" <https://learn.microsoft.com/en-us/dotnet/csharp/language-reference/language-specification/namespaces> Hämtad: 2024-10-23
- [6] Microsoft Learn, "JSON serialization and deserialization (marshalling and unmarshalling in .NET – overview)" <https://learn.microsoft.com/en-us/dotnet/standard/serialization/system-text-json/overview> Hämtad: 2024-10-23
- [7] Microsoft Learn, "Language Integrated Query (LINQ)" <https://learn.microsoft.com/en-us/dotnet/csharp/linq/> Hämtad: 2024-10-23

8 Bilagor

Bilaga 1

Applikationen är uppbyggd av två grundklasser som sedan används i underklasser med tillhörande metoder enligt bilderna nedan där de blå rutorna är klasser och de gröna är metoder.



Bilaga 2 - Flödesschema

1. Start av applikation

- Klassen **MenuManager**, med metoden **ShowMenu**, anropas och huvudmenyn visas med följande alternativ:

1. Spela quiz
2. Visa topplista
3. Admin: Hantera quizet och topplistan
- X. Avsluta quizet

2. Val: Spela quiz

- Klassen **QuizManager**, med metoden **PlayQuiz**, anropas med följande steg:
 - Tidtagningen startas
 - En fråga i taget visas och spelaren anger sitt svar
 - Svaret kontrolleras om det är rätt eller fel
 - Spelaren kan avbryta spelet med "x", resultat sparas då ej
 - Tidtagningen stoppas
- Efter quizet visas antalet rätt samt tid
- Resultatet sparas till filen **toplist.json** via metoden **SaveToFile** i klassen **GameManager**.

3. Val: Visa topplista

- Klassen **GameManager**, med metoden **ShowTopList**, anropas och listan **topList** med de tio bästa resultaten sorterade efter antal rätt och därefter tid visas med:
 - Namn på spelaren, antal rätt och tid

4. Val: Hantera quizet

- Klassen **MenuManager**, med metoden **ShowSubMenu**, anropas och undermenyn visas med följande alternativ:
 1. Visa alla frågor

- 2. Lägg till fråga
- 3. Ta bort fråga
- 4. Visa topplistan
- 5. Radera topplistan
- X. Återgå till huvudmenyn

5. Undermeny val: Visa alla frågor

- Klassen **QuizManager**, med metoden **ShowQuiz** där metoden **GetQuiz** används, anropas:
 - Alla frågor och svar skrivs ut

6. Undermeny val: Lägg till fråga

- Klassen **MenuManager**, med metoden **AddQuestion**, anropas för att samla in ny fråga och tillhörande svar:
 - Användaren skriver in fråga och ett svar
- Klassen **QuizManager**, med metoden **AddToQuiz**, anropas för att lägga till den nya frågan i quizet. Frågan sparas till filen **quiz.json** via metoden **SaveToJsonFile**.

7. Undermeny val: Ta bort fråga

- Klassen **MenuManager**, med metoden **DeleteQuestion**, anropas för att ta bort en fråga med **QuizManager**-objekt som parameter:
 - Alla frågor skrivs ut numrerade med metoden **ShowQuiz**, där metoden **GetQuiz** används
 - Användaren anger index på den fråga som ska raderas
- I klassen **QuizManager** raderas sedan frågan med metoden **DeleteQuiz** och sparas sedan om till filen **quiz.json** via metoden **SaveToJsonFile**

8. Undermeny val: Visa topplistan

- Klassen **GameManager**, med metoden **ShowTopList**, anropas:
 - De tio bästa resultaten visas sorterade efter antal rätt och därefter efter tid.

9. Undermeny val: Radera topplistan

- Klassen **GameManager**, med metoden **DeleteTopList**, anropas:
 - Användaren får bekräfta om topplistan säkert ska raderas med y/n
- Listan raderas och sparas sedan med metoden **SaveToFile**

10. Val: Avsluta quizet

- *Enviroment.Exit(0)* anropas och avslutar applikationen