Projektuppgift

Datateknik GR (A)

Projektarbete
Konsolapplikation

Cristina Löfqvist

Mittuniversitetet
MID SWEDEN UNIVERSITY

Campus Härnösand Universitetsbacken 1, SE-871 88. **Campus Sundsvall** Holmgatan 10, SE-851 70 Sundsvall.
Campus Östersund Kunskapens väg 8, SE-831 25 Östersund.
Phone: +46 (0)771 97 50 00, Fax: +46 (0)771 97 50 01.

MITTUNIVERSITETET
Avdelningen för informationssystem och -teknologi

Författare: Cristina Löfqvist, crlo1900@student.miun.se
Utbildningsprogram: Webbutveckling, 120 hp
Huvudområde: Datateknik
Termin, år: VT, 2021

Innehållsförteckning

1. Introduktion

Projektet avser skapandet av ett konsol/terminal baserat reaktionsspel.

Spelet som skall skapas skall fungera på följande vis:

- En bokstav mellan a-z presenteras i konsolfönstret
- Användaren (spelaren) skall där efter på så kort tid som möjligt trycka på tangenten för motsvarande bokstav
- Detta sker 10 gånger
- Under spelets gång tas tiden på hur lång tid det tar att trycka in 10 presenterade bokstäverna mellan a-z
- Därefter får man om man slagit någon av de 10 snabbaste tiderna placera sitt namn på en ledarlista (highscore)

1 Teori

För att utveckla spelet i fråga (fortsatt benämnt som applikationen/spelet) användes C# i utvecklingsmiljön Visual studio. Applikationen är nästan uteslutande baserat på C# funktionalitet kopplad till user input från ett tangentbord samt funktionaliteten att skriva ut tecken till ett konsolfönster. Exempel på metoder är:

- using System
- Console.WriteLine()
- Console.ReadLine()
- Console.ReadKey()

[1][2][3].

System är ett så kallat namespace och **Console** är en klass som finns i namespacet system. Namespace eller ”namnområde” används i C# dels för att organisera mängden .NET klasser och dels för att deklarera egna namnområden och på så vis kunna skapa kontroll över klass och metodnamn i större programmerings arbeten [6]. På så sätt kan man till exempel undvika att två eller flera klasser/metoder kolliderar på grund av att de ha samma namn i en programkod.

using är ett nyckelord som kan användas istället för att behöva skriva ut hela namnet på namespace vid varje metodanrop. Alltså genom att ange using system högst upp kan man skippa att ange **System.** framför varenda metodanrop [6].

Console.WriteLine metoden fungerar så att den skriver den specifika datan till standard output [1].

Console.ReadLine metoden fungerar så att den läser nästa rad tecken från standard input stream [2].

Console.ReadKey metoden fungerar så att den erhåller det tecken eller funktions tangent som trycks ned av användaren [3].

För att kunna ha en uppfattning av tid har **Stopwatch** funktionalitet används enligt följande kodexempel:

- using System.Diagnostics

I exemplet ovan används Sytem.Diagnostics som innehåller klasser som gör det möjligt att interagera med händelseloggar och prestandaräknare med mera. En av de klasser som System.Diagnostics innehåller är Stopwatch [5]. Stopwatch är en .NET klass som innehåller en samling metoder och egenskaper ”properties” som kan användas för att mäta hur lång tid som passerat [4].

För att spara undan resultaten samt visa tidigare resultat har skrivning och läsning av en fil använts som lagring medium, genom följande C# funktionaliteter:

- `using System.IO`
- `StreamReader file = new StreamReader(@"Highscores.txt");`
- `StreamWriter sw = new StreamWriter(@"Highscores.txt", false);`

System.IO är ett namespace som innehåller klasser för att läsa och skriva till filer. Bland annat klasserna i exemplet ovan; StreamWriter och StreamReader m [7]. StreamWriter klassen implementerar en TextWriter för att skriva tecken till en byte stream [8]. StreamReader klassen läser från en sträng och är utformad för tecken inmatning i en specifik ström av kod [9].

Alltså har StreamReader/Writer i System.IO använts för att uppnå möjlighet till att läsa och skriva till en fil [7] [8] [9].

2 Metod

För att implementera applikationen har som tidigare nämnts C# använts i Visual studio. Samt funktionalitet i C# kopplad till:

- Läsa användarinput från tangentbordet genom:
  - `using System`
  - `Console.ReadLine()`
  - `Console.ReadKey()`
- Skriva till ett konsol/terminalfönster genom:
  - `using System`
  - `Console.WriteLine()`
- Stopwatch funktionalitet:
  - `using System.Diagnostics`
- Skriva till en fil:
  - `using System.IO`
  - `StreamWriter sw = new StreamWriter(@"Highscores.txt", false);`
- Läsa en fil:
  - `using System.IO`
  - `StreamReader file = new StreamReader(@"Highscores.txt");`

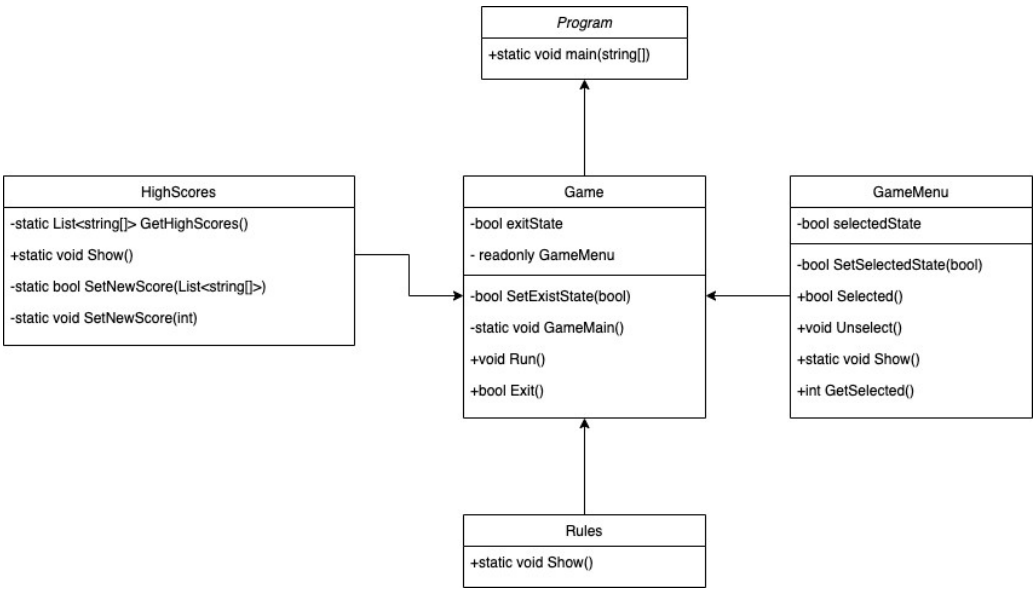
Applikationen valdes att implementeras på ett objektorienterat sätt. Uppdelningen av klasser och vilka klasser som kom att finnas ses under rubriken Konstruktion.

3 Konstruktion

Applikationen är uppbygd av följande klasser:

- class Program
- public class Game
- public class GameMenu
- public class HighScores
- public class Rules

Sambandet mellan de olika klasserna kan ses i uml diagrammet nedan:



3.1 class Program

class Program är den klassen som i C# innehåller Main metoden:

- `static void Main(string[] args)`

Main metoden är den metoden som körs när man startar ett C# program. Från Main instansernas en instans av klassen Game och sedan anropas metoden Run i klassan Game i en loop som körs så länge instansen av game inte har satt ett av sina fields till false. Se kod här nedan:

```
static void Main(string[] args)
{
        Game game = new Game();
        do
        {
                game.Run();
        }
        while (!game.Exit());
}
```

3.2 class Rules

Klassen Rules innehåller funktionalitet för att:

- Visa reglerna för spelet

genom att använda som nämnt tidigare funktionalitet I System för att skriva till ett konsolfönster samt läsa input från användarens tangetbord. Klassen innehåller bara en metod som är statisk, vilket innebär att ingen instans av klassen måste skapas för att man skall kunna anropa metoden, utan det räcker med att man anger klassens namn följt av .metodnamnet. Se exempel här nedan:

```
public class Rules
{
        public static void Show()
        {
        }
}
```

Någon annanstans:

```
Rules.Show();
```

3.3 class HighScores

Klassen HighScores innehåller funktionalitet för att:

- Visa skriva ut highscorelistan till konsolfönstret

- Spara ett nytt highscore

- Samt utföra kontroll att den angivna tiden som en spelare fick vid ett spel kvalificerar sig till att vara med på highscorelistan

För att spara highscorelistan valdes det att göra detta på enklast möjliga sätt och det blev att spara highscorelistan i en textfil. För att spara ett highscore i listan läses listan (textfilen) rad för rad och en kontroll görs om tiden som spelaren fick i sin spelomgång är mindre än något av de sparade highscoren, om så är fallet öppnas filen för skrivning och skrivs om innehållande det nya highscoret. Detta utfördes med som tidigare nämnt:

- `using System.IO`

- `StreamReader file = new StreamReader(@"Highscores.txt");`

- `StreamWriter sw = new StreamWriter(@"Highscores.txt", false);`

Internt i klassen HighScores representeras listan av highscore som en:

- `List<string[]>`

En Lista av sträng arrayer, anledningen till detta är när en rad läses läggs den till i ett List objekt som tillhandahåller metoder för att dynamiskt beskriva en lista av någon typ (i detta fall sträng array) samt att varje rad i highscorelistan innehåller två strängar:

- namn

- score (tid det tog för att knapp in alla 10 bokstäver som spelet skrev ut till konsolfönstret)

Man hade kunnat använda en matris av strängar då det på förhand var bestämt att highscorelistan skulle endast vara 10 rader lång, men det valdes att använda List för enkelhetens skull.

Likt Rules klassen innehåller HighScores klassen endast static metoder.

De publika metoderna som används av en annan klass:

- Show

  - Visar listan av highscores

- NewScore

- Spara ett nytt score om det är bättre än något på highscorelistan

Dessa två metoderna är implementerade med hjälp av vissa privata metoder. Privata metoder är metoder som endast kan nås inom klassen [11].

Privata metoder som finns i HighScores:

- SetNewScore
  - Skriver ett nytt score till highscore som finns i textfilen
- GetHighScores
  - Läser textfilen som innehåller highscorelistan

3.4 class GameMenu

Klassen GameMenu inehåller funktionalitet för att:

- Visa spelets menysystem
- Ta emot användarinput för val i menysystemet
- Returnera valet gjort av spelaren till klassen Game

genom att använda som nämnt tidigare funktionalitet I System för att skriva till ett konsolfönster samt läsa input från användarens tangetbord.

Klassen GameMenu innehåller inte bara statiska metoder då den har ett field som håller värdet av om menyn har blivit vald eller inte alltså om ett val har gjorts av spelaren. Detta kan liknas med GameMenu har ett state som är antingen att en spelare har valt ett alternativ från menyn eller inte. Detta state används för att veta om menyn skall visas eller inte eller hur menyn skall visas. I och med detta field state kan ändras måste en instans av GameMenu instansieras i klassen Game för att men skall kunna komma åt detta state.

De statiska metoderna i GameMenu är följande:

- Show
  - Denna funktion visar själva menyn

3.5 class Game

Klassen Game är den klass som är själva spelet och den klassen använder sig av de övriga klasserna förutom klassen Program.

Game instansierar endast klassen:

- GameMenu
    - `private readonly GameMenu gameMenu = new GameMenu();`

För de andra klasserna:

- Rules
- HighScores

Innehåller endast statiska metoder som ej kräver en instans av klassen för att anropa.

Game klassen är uppbygd på följande vis:

- Två publika metoder:
    - Run
        - Kör Själva spelet inklusive meny
    - Exit
        - Returnerar field exitState som är ett field som är antingen false (spelet skall ej avslutas) eller true (spelet skall avslutas)
- Flera privata metoder som används i:
    - Run
        - Set ExitState
            - Sätter fieldet exitState till antingen false eller true
        - GameMain
            - Själva spelets spelrunda som tar tiden på hur lång spelrundan är, och som tidigare nämnt visar 10 bokstäver a-z och väntar på att användaren skall trycka på den bokstav som presenteras i konsolfönstret

4 Resultat

Spelet som utvecklades är fullt fungerande, men det finns utrymme för förbättring, både spelupplevelsevis och implementationsvis.

Punkter som skulle kunna höja spelupplevelsen:

- Inkludera fler tecken än endast små bokstäver mellan a-z
- Kunna ställa svårighetsgraden som hade kunnat reglera vilka tecken som spelet vill att användare skall mata in
- Kunna ha olika levels med olika svårigheter

Som man ser ovan kan många förbättringar göra till själva utfallet av implementationen av applikationen.

Förbättringar som skulle kunna göras av nuvarande implementation:

- Dela upp implementationen av nuvarande metoder i flera mindre metoder
- Se över hur de ”states” som finns används och eventuellt förenkla hur applikationen vet vilka metoder som skall anropas

5 Slutsatser

Idén framkom inga större problem under utvecklingen av applikationen, men ju mer som implementerades desto större blev insikten att som ovan nämnt under rubriken Reslutat, så kunde användandet av ”state” göras på ett bättre sätt, eller helt uteslutas, samt att vissa metoder blev mer och mer invecklade och kunde kanske delas upp i mindre metoder. Även nämnt under rubriken Resultat diskuteras vad som hade kunnat vidare utvecklas. Applikationens nuvarande implementation är på ett grundläggande sätt utvecklad för att kunna vidare utvecklas med limiterad kodomskrivning.

Källförteckning

[1] Microsoft ”Console.WriteLine Method” https://docs.microsoft.com/en-us/dotnet/api/system.console.writeline?view=net-5.0 Publicerad: 2021. Hämtad: 2021-03-31.

[2] Microsoft ”Console.ReadLine” https://docs.microsoft.com/en-us/dotnet/api/system.console.readline?view=net-5.0 Publicerad: 2021. Hämtad: 2021-03-31.

[3] Microsoft ”Console.ReadKey Method” https://docs.microsoft.com/en-us/dotnet/api/system.console.readkey?view=net-5.0 Publicerad: 2021. Hämtad: 2021-03-31.

[4] Microsoft ”StopWatch class” https://docs.microsoft.com/en-us/dotnet/api/system.diagnostics.stopwatch?view=net-5.0 Publicerad: 2021. Hämtad: 2021-03-31.

[5] Microsoft ”System.Diagnostics Namespace” https://docs.microsoft.com/en-us/dotnet/api/system.diagnostics?view=net-5. Publicerad: 2021. Hämtad: 2021-03-31.

[6] Microsoft ”Namespaces C# Programming Guide” https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/namespaces/ Publicerad: 2021. Hämtad: 2021-03-21.

[7] Microsoft ”System.IO Nemespace” https://docs.microsoft.com/en-us/dotnet/api/system.io?view=net-5.0 Publicerad: 2021. Hämtad: 2021-03-31.

[8] Microsoft ”StreamWriter Class” https://docs.microsoft.com/en-us/dotnet/api/system.io.streamwriter?view=net-5.0 Publicerad: 2021. Hämtad: 2021-03-31.

[9] Microsoft ”StreamReader Class” https://docs.microsoft.com/en-us/dotnet/api/system.io.streamreader?view=net-5.0 Publicerad: 2021. Hämtad: 2021-03-31.

[10] Microsoft ”TextReader Class” https://docs.microsoft.com/en-us/dotnet/api/system.io.textreader?view=net-5.0 Publicerad: 2021. Hämtad: 2021-03-31.

[11] tutorialspoint ”Private methods in C#” https://www.tutorialspoint.com/Private-Methods-in-Chash#:~:text=Private%20Methods%20can%20only

%20be,from%20other%20functions%20and%20objects. Publicerad: 2018-08-14 Hämtad: 2021-03-31