FILSPARNING OCH FILLADDNING MED KLASSEN HIGHSCORE

Detta dokument ska läsas för den som vill fördjupa sin förståelse för hur man kan arbeta med filsparning- och filladdning i XNA. Dokumentet kan ses som en utökning av kapitel 9 i boken *Programmering 2 med C#*.

Om du arbetade med övningshäftet för boken Programmering 1, så skapade du kanske klassen `HighScore` för att arbeta med higscore-listor. Vi ska i det här kapitlet vidareutveckla en sådan klass som jag har skapat och lägga till funktionalitet för filsparning och filladdning.

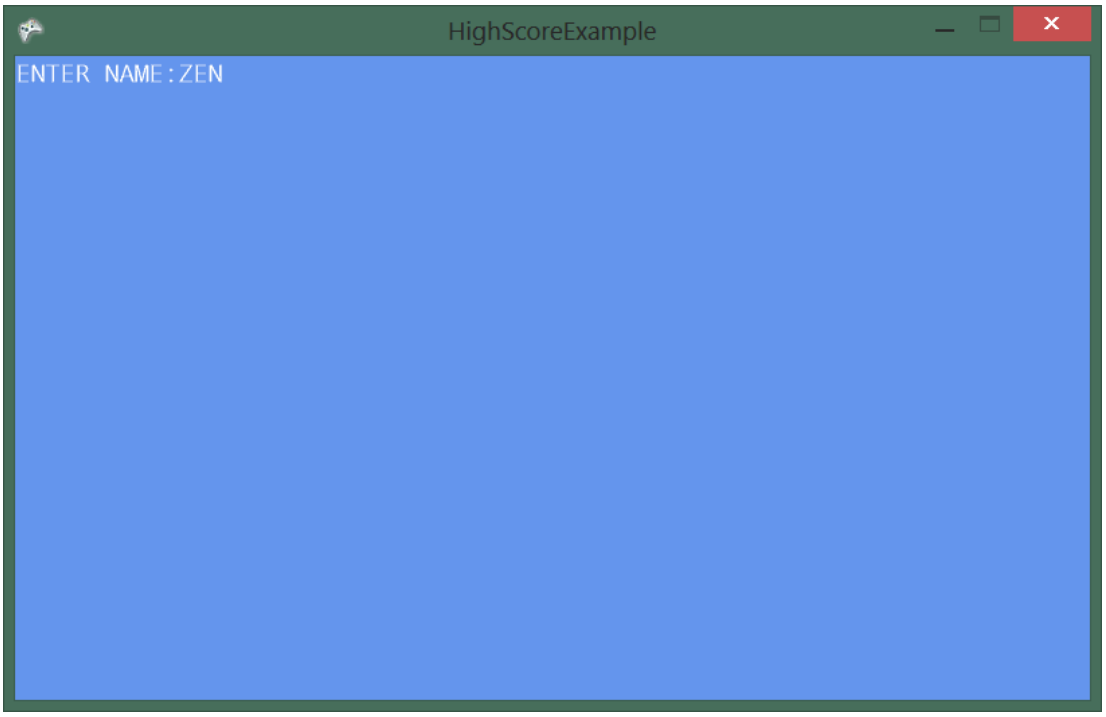
> Detta dokument kräver att du använder filen *HighScore.cs* som finns i *csharp2exempel.zip* under *XNA (Windows-spel) med Visual Studo 2010\Kap 09 (Filer)*.

Problem med tangentinmatning i XNA

I XNA finns det inget smidigt inbyggt sätt att skapa rutor för textinmatning från tangentbord. Det känns kanske dumt för oss som programmerar spel för datorer, men de flesta TV-spelskonsoler saknar tangentbord. Även om det finns tangentbord till Xbox 360 är det mycket få som använder det.

Detta skapar förstås ett problem för oss som vill ha en highscore-lista. Det finns flera sätt att lösa problemet på. Ett är att använda bibliotek vars syfte är att lösa just dessa problem. Jag väljer dock att skapa en highscore-lista, där man får skriva in sitt namn som gamla traditionella arkad-spel. Det innebär att man har ett namn på tre tecken (t.ex. AAA eller XYV) och bläddrar mellan olika tecken (i den kod jag har skrivit för klassen HighScore bläddrar man med upp- och ner-tangenterna och väljer det aktuella tecknet med ENTER).

Så här ser det ut då man väljer namn:



Man väljer namn med piltangenterna upp och ned, samt ENTER.

Att arbeta med andras klasser

Hittills i boken har vi endast arbetat med de klasser och objekt som vi tillsammans har skapat, samt med de klasser och objekt som finns i C#:s standardbibliotek och XNA. Vi (eller kanske snarare du) ska nu arbeta med en klass som jag har skapat.

När man arbetar med andras klasser finns det flera olika sätt man gör det på. Ibland kan det vara stora klassbibliotek eller andra projekt som man måste importera. I fallet med klassen `HighScore` är det dock bara en enda fil som vi ska lägga in i vårt projekt.

Ofta arbetar man också i större projekt. Då ansvarar en person för att koda en viss klass medan en annan person ansvarar för att skapa objekt av den i sin klass. En sådan gång är det viktigt att man kan kommunicera mellan varandra.

Något som är A och O när man arbetar med andras klasser (liksom standardbibliotek) är att de är väl dokumenterade. Helst ska källkoden vara väl kommenterad men det bör också finnas ett dokument som beskriver klassen, dess metoder och egenskaper.

Vad gäller klassen `HighScore` har jag försökt kommentera den väl.

Att använda klassen HighScore

Klassen `HighScore` innehåller alltså följande publika metoder:

- `PrintDraw()`

- `EnterUpdate()`
- `EnterDraw()`
- *`SaveToFile()`*
- *`LoadFromFile()`*

Metoderna `Highscore.SaveToFile()` och `Highscore.LoadFromFile()` har ingen kropp och det är upp till oss att implemenetera dem. Just nu kan vi dock strunta i dem och bara koncentrera oss på de tre första.

Metoden `Highscore.PrintDraw()` anropas av `Draw()`-delen av spelloopen och skriver ut highscore-listan på skärmen. Det finns ingen `Update()`-metod för detta, då igen användarinput förväntas och inget på skärmen ska ändras.

Metoderna `Highscore.EnterUpdate()` och `Highscore.EnterDraw()` används för att spelaren ska kunna mata in sitt namn (på tre bokstäver) i highscore-listan. Som vi ser på namnet anropas den ena från `Update()`-delen av spel-loopen medan den andra anropas från `Draw()`-delen.

Nytt projekt: HighScoreExample

Låt oss skapa ett helt nytt XNA-projekt där vi använder oss av klassen `HighScore`. Jag döper det nya projektet till *HighScoreExample*. Detta kommer vara ett mycket minimalistiskt exempel och den enda grafiken, är text som ritas ut på skärmen. För det behöver vi ladda in en ny font-fil i Content. Vi döper font-filen till *myFont.spritefont*.

För att ladda in font-filen i vår kod, så skapar vi ett `SpriteFont`-objekt längst upp i `Game1`:

Exempel 0.1

```
SpriteFont myFont;
```

Sedan laddar vi in Content-objektet *myFont* i metoden `Game1.LoadContent()`:

Exempel 0.2

```
myFont = Content.Load<SpriteFont>("myFont");
```

Klassen HighScore

Vi lägger in filen *HighScore.cs* i vårt projekt och skapar ett objekt av det längst upp i `Game1`:

Exempel 0.3

```
HighScore highscore;
```

Sedan anropar vi konstruktorn i `Game1.Initialize()`, jag väljer att ha 10 personer i listan som maxantal:

Exempel 0.4

```
highscore = new HighScore(10);
```

Två states

Vi ska nu skapa stöd för två olika states. Vi vill bara kunna skriva ut highscore-listan och skriva in oss i listan. Vi börjar med att göra en `enum` längst upp i `Game1`, samt skapar ett objekt av den:

Exempel 0.5

```
enum State { PrintHighScore, EnterHighScore };
State currentState;
```

I `Game1.Update()` kan vi nu använda dessa två states. I statet `EnterHighScore` så vill vi anropa metoden `HighScore.EnterUpdate()` via vårt `highscore`-objekt, för att vi ska få skriva in oss i listan. Denna metod returnerar `true` då den är färdig och `false` medan den inte är färdig (kom ihåg att metoden anropas 30 eller 60 gånger per sekund). När den är färdig vill vi byta state till `PrintHighScore`.

Som argument till `HighScore.EnterUpdate()` måste vi dels skicka ett `GameTime`-objekt, men också antalet poäng. Tanken är ju att vi egentligen ska anropa metoden efter att vi har spelat klart ett spel och därmed vet hur många poäng vi har tagit. Här väljer jag istället att alltid skicka in 10 päng (ett tips för att testa är att slumpa ut olika tal och skicka in som poäng, istället för att alltid skicka in 10).

`default`-statet är själva visandet av själva highscorelistan. Det enda vi gör här, är att kontrollera om användaren trycker på ”e” (som i Enter):

Exempel 0.6

```
switch (currentState)
{
    case State.EnterHighScore: // Skriv in oss i listan
        // Fortsätt så länge HighScore.EnterUpdate() returnerar true:
        if(highscore.EnterUpdate(gameTime, 10))
            currentState = State.PrintHighScore;
        break;
    default: // Highscore-listan (tar emot en tangent)
        KeyboardState keyboardState = Keyboard.GetState();
        if(keyboardState.IsKeyDown(Keys.E))
            currentState = State.EnterHighScore;
        break;
}
```

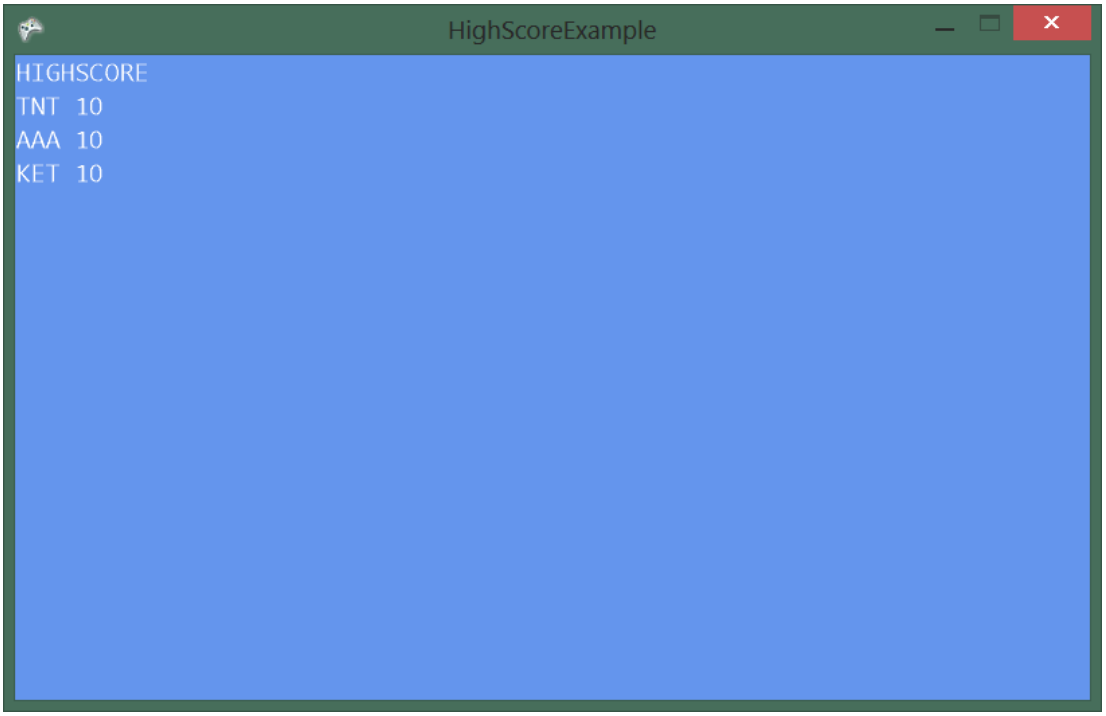
Vi kan nu lägga till stöd för våra två states i `Game1.Draw()` och där anropa `HighScore.EnterDraw()` eller `highScore.PrintDraw()` beroende på state:

Exempel 0.7

```
// Använd spriteBatch för att rita ut saker på skärmen
spriteBatch.Begin();
```

```
switch (currentState)
{
    case State.EnterHighScore: // Skriv in oss i listan
        highscore.EnterDraw(spriteBatch, myFont);
        break;
    default: // Rita ut highscore-listan
        highscore.PrintDraw(spriteBatch, myFont);
        break;
}
spriteBatch.End();
```

Så här kan det se ut:



HighScore-listan skrivs ut och man kan skriva in sig genom att trycka på E.

Att spara listan till fil

Något som är lite tråkigt med denna highscorelista är att den återställs varje gång man startar om spelet. Det hela blir föstås betydligt roligare om vi laddar och sparar allt från och till en fil. I klassen `HighScore` finns metoden som saknar kropp. Låt oss nu tillsammans ge metoden en kropp, samt använda oss av den.

Innan vi fortsätter vill vi först bestämma *hur* vi ska spara innehållet i filen. Jag föreslår att vi sparar varje spelares namn och poäng skiljt åt med ett kolon. Vi sparar också varje spelare och dennes poäng på en enskild rad. Här är ett exempel på hur innehållet i en sådan fil skulle kunna se ut:

highscore.txt – exempel på sparad fil

```
TNT:10
AAA:10
```

```
KET:10
```

Koden för detta är inte speciellt krånglig om man väl har förstått hur det funkar med filsparning. Det enda vi behöver göra är att loopa igenom alla element i highscore-listan och hämta ut namn och päng. För varje element skriver vi in en ny rad i filen (observera att vi också måste lägga till `using System.IO` längst upp):

Exempel 0.8

```
// ===========================================================================
// SaveToFile(), spara till fil.
// ===========================================================================
public void SaveToFile(string filename)
{
    StreamWriter sw = new StreamWriter(filename);

    // Skriv in alla personer i highscore-listan:
    foreach (HSItem item in highscore)
    {
        string text = item.Name + ":" + item.Points;
        sw.WriteLine(text);
    }

    sw.Close(); // Stäng filen
}
```

Vi kan nu välja att anropa denna metod. Frågan är var? Det är upp till speldesignern att bestämma detta. Det är dock mycket viktigt att man inte skriver kod för att spara och ladda filer som en helt vanlig del av spel-loopen. I vårt fall kan vi faktiskt anropa denna i `UnloadContent()`:

Exempel 0.9

```
highscore.SaveToFile("highscore.txt");
```

Om vi nu startar spelt, skriver in oss några gånger i listan och stänger av det, så bör filen *highscore.txt* ha skapats i samma map som exe-filen till vårt projekt ligger. Titta gärna i filen!

Att ladda listan från fil

Så långt, så gott. Men nu vill vi ju också kunna ladda listan. Att ladda in listan är lite klurigare, eftersom att vi då måste skapa nya `HSItem`-objekt för varje rad i filen. Dessutom måste vi plocka ut namnet och poängen som ju är separerade med ett kolon-tecken.

För att separera namn och poäng så kan vi använda metoden `String.Split()`. Den kan ta emot ett `char`-tecken som argument och returnera en `string`-vektor. I vårt fall kommer vektorn bara innehålla två olika värden. Den första innehåller namnet och den andra poängen. Poängen måste vi sedan göra om till ett heltal med `Convert.ToInt32()`. Så här kan vi koda `LoadFromFile()`:

Exempel 0.10

```
// ========================================================================
// LoadFromFile(), ladda från fil.
// ========================================================================
public void LoadFromFile(string filename)
{
    StreamReader sr = new StreamReader(filename);

    // Läs in alla rader till string-objektet row och skriv ut dem:
    string row;
    while ((row = sr.ReadLine()) != null)
    {
        // skapa en vektor som innehåller namn och poäng,
        // words[0] blir namnet och words[1] är poängen:
        string[] words = row.Split(':');
        int points = Convert.ToInt32(words[1]);

        // Lägg till i listan:
        HSItem temp = new HSItem(words[0], points);
        highscore.Add(temp);
    }

    sr.Close(); // Stäng filen
}
```

Läs mer om hur man delar strängar här: http://msdn.microsoft.com/en-us/library/ms228388.aspx

Vi kan lägga anropet till metoden i `Game1.LoadContent()`:

Exempel 0.11

```
highscore.LoadFromFile("highscore.txt");
```

Så, nu laddas vår highscore-lista från fil!