

C# OO Programmeren

LES 9

JOHAN DONNÉ

Concurrent (= parallel) programming

Tasks

Tasks interactie met GUI

Tasks

Task

Bedoeling: multithreaded programmeren

In C#:

- OS level Threads (low level, overhead, management)
- ThreadPool (less overhead, still low level)
- BackgroundWorker (OK but specific, deprecated)
- Tasks
- Parallel loops

Task

Bedoeling: multithreaded programmeren

In C#:

- ~~OS level Threads~~
- ~~ThreadPools~~
- ~~BackGroundWorker~~

- Tasks
- Parallel loops

Task

'Task' Object (namespace System.threading.Tasks):

Create:

```
Task job = Task.Run((Action)DoSomething);  
Task.Run((Action)DoSomething);
```

Wait for end:

```
job.Wait();
```

⇒ job.Wait() is een 'blocking' statement!

Task - voorbeeld

```
public void DoSomething()
{
    Console.WriteLine("task executing...");
    Console.ReadLine();
}

public void Run()
{
    // running on UI thread
    Task job = Task.Run((Action)DoSomething);
    Console.WriteLine("task started...");
    // execute code concurrent with the job task here...
    job.Wait();
    Console.WriteLine("task ended.");
}
```

Task creation

```
Task job = Task.Run((Action)DoSomething);

Task job2 = Task.Run(() =>
{
    for (int i = 0; i < 6; i++)
    {
        Console.WriteLine($"    lambda task 2: loop nr. {i}");
        Thread.Sleep(100);
    }
});

// 'fire and forget'...
Task.Run(() => { Console.WriteLine("task  executing");});
```


Task

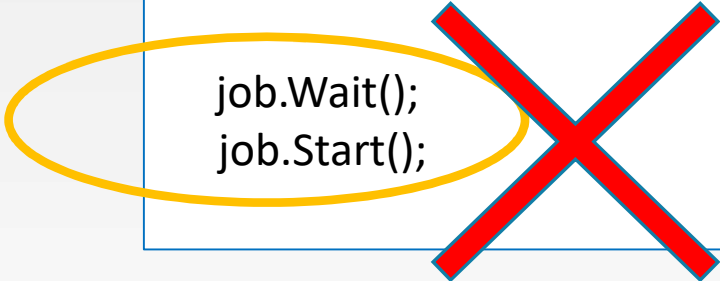
Opmerking:

Een 'Task' kan niet meer opnieuw gestart worden nadat die beëindigd is.

```
Task job = Task.Run ((Action)DoSomething);
```

...

job.Wait();
job.Start();



Task creation - passing a parameter

```
public void DoSomething(int count)
{
    for (int i = 0; i < count; i++)
    {
        Console.WriteLine($"    Concurrent task: loop nr. {i}");
        Thread.Sleep(500);
    }
}

int nrOfIterations = 8;
Task job = Task.Run(() => { DoSomething(nrOfIterations); });
nrOfIterations = 3;
Console.WriteLine("task started...");
```

⇒ Opgelet: parameterwaarde bij start van Task, niet bij schedule!!!

Task – returning a result

```
public int DoSomething(int count)
{
    int result = 0
    for (int i = 0; i < count; i++) result += i;
    return result;
}

Task<int> job = Task.Run(() => { return DoSomething(20); });

// execute code concurrent with the job task here...

int number = job.Result;
```

⇒ job.Result is een 'blocking operation'

Task – waiting to finish

```
Task<int> job = Task.Run(() => { return DoSomething(20); });

job.Wait(); // blocking!

bool finished = job.Wait(100); // wait max 100ms.

Task.WaitAll(new Task[] { job2, job3 }); // blocking!

bool finished = Task.WaitAll(new Task[] { job2, job3 }, 100);
```

Tasks - Cancelling

Canceling a running task

- ⇒ CancellationSource --> CancellationToken
- ⇒ CancellationToken als parameter meegeven
- ⇒ CancellationSource.Cancel()

Zie Demo Les 09

Tasks - Exceptions

Exceptions in tasks (indien niet opgevangen in 'catch')

⇒ task stopt, andere tasks/threads blijven lopen

⇒ bij Wait, WaitAll, Result: AggregateException

⇒ InnerException = originele exception

Zie Reference guide...

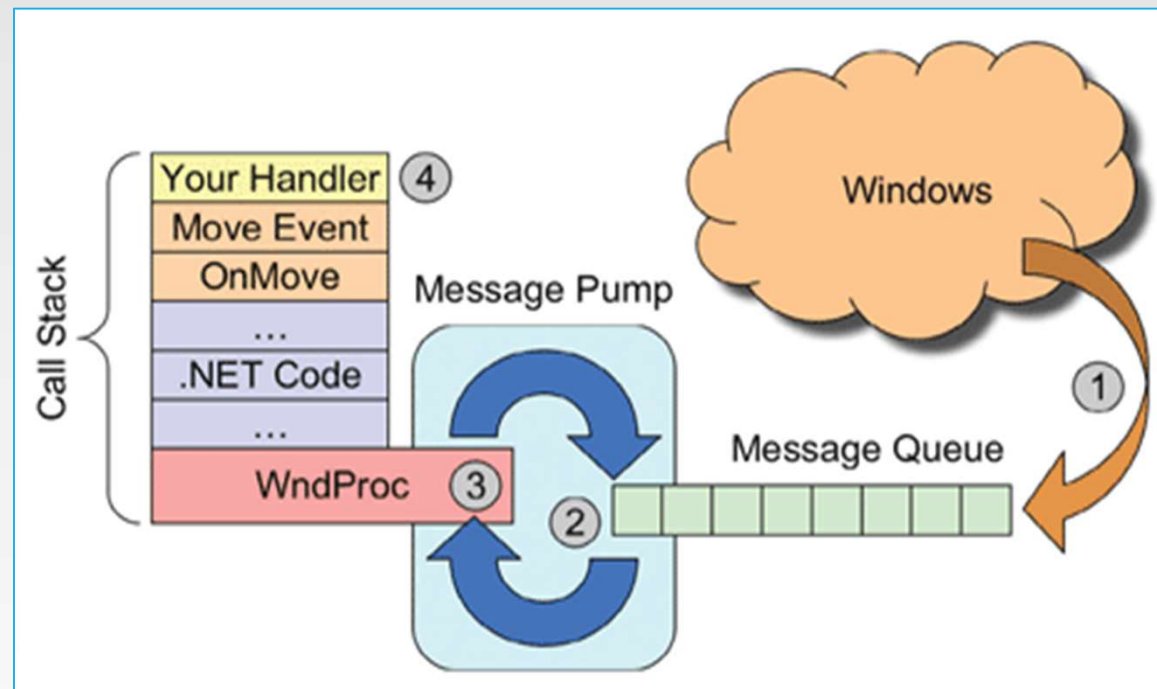
Demo

Task interactie met GUI

Tasks – GUI : achtergrond

Windows: 'Event driven' op basis van 'Message pump' (voor elke GUI thread).

GUI thread:



Tasks – GUI : problematiek

GUI (Winforms, WPF...) is meestal niet thread-safe

- ⇒ Interactie met controls op GUI enkel mogelijk vanuit GUI-thread
- ⇒ Eindresultaat van een asynchrone task: op te vragen via `job.Result`
- ⇒ 'rapportering' vanuit task naar GUI niet eenvoudig mogelijk.

Reden:

- OS laat niet toe dat een 'gewone' thread controle neemt over UI-controls.
- GUI controls niet thread safe geschreven.

Tasks – GUI : Oplossing voor interactie

Mechanismen om vanuit andere thread aan het OS te vragen om een methode-oproep te laten uitvoeren in de GUI- thread.

Vereist interactie met thread scheduler en info over de 'context' waarin een methode mag uitgevoerd worden.

⇒ WinForm controls: 'InvokeRequired' & 'BeginInvoke'

⇒ WPF: 'Dispatcher.Invoke'

⇒ 'IProgress<T>' en 'Progress<T>' klasse

⇒ ...

⇒ Meest algemene (en eenvoudigst?): SynchronisationContext

Tasks – GUI : SynchronisatieContext

SynchronisatieContext =

Abstracte klasse met

- informatie over de ThreadContext waarin code uitgevoerd wordt.
- Methodes om in de gevraagde context (= thread) code uit te voeren

Subklassen:

- WindowsFormsSynchronizationContext (WinForms)
- DispatcherSynchronizationContext (WPF)
- ThreadPool synchronisation context
- ...

Tasks – GUI : SynchronisatieContext

Pattern:

- Onthou bij de initialisatie van je form de 'current' synchronisation context (= context van de UI-thread)
- Telkens je (de inhoud van) een control moet wijzigen terwijl je niet zeker bent vanuit welke thread dat gebeurt, gebruik je de 'Send' methode (blocking!) of 'Post' methode (asynchroon) om de wijziging te schedulen in de Ui-thread.

```
public virtual void Post (System.Threading.SendOrPostCallback d, object state);
```

met

```
public delegate void SendOrPostCallback(object state);
```

⇒ Dus een void methode met een parameter van type 'object'

SynchronisatieContext : voorbeeld

Form constructor

```
private readonly IBackendWorker worker;
private SynchronizationContext uiContext;

public MainForm(IBackendWorker worker)
{
    InitializeComponent();
    this.worker = worker;
    uiContext = SynchronizationContext.Current;
    worker.BackendEvent += OnWorkerBackEvent;
    ...
}
```

EventHandler

```
private void OnWorkerBackEvent(int count)
{
    // Instead schedule the action on the UI synchronisation context.
    uiContext.Post((c) => outputTextBox.Text = count.ToString(), null);

    // uiContext.Post((c) => outputTextBox.Text = c.ToString(), count);
}
```

Code Voorbeeld

π Calculator

Pi Calculator - achtergrond

Formule van Leibniz (1674):

Leibniz formula for π

From Wikipedia, the free encyclopedia

See [List of things named after Gottfried Leibniz](#) for other formulas known under the same name.

In [mathematics](#), the **Leibniz formula for π** , named after [Gottfried Leibniz](#), states that

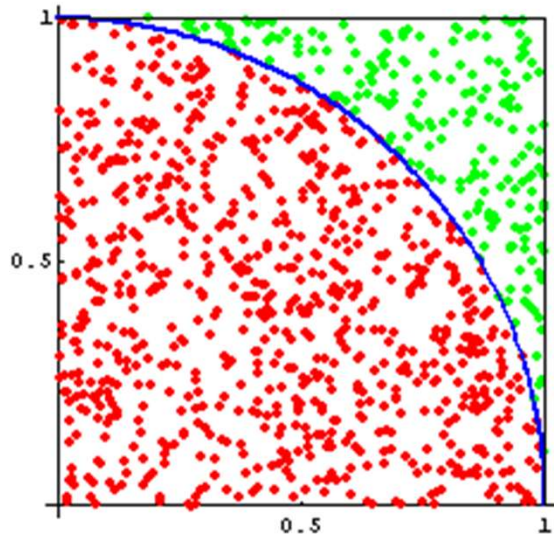
$$1 - \frac{1}{3} + \frac{1}{5} - \frac{1}{7} + \frac{1}{9} - \cdots = \frac{\pi}{4}.$$

Using [summation](#) notation:

$$\sum_{n=0}^{\infty} \frac{(-1)^n}{2n+1} = \frac{\pi}{4}.$$

Pi Calculator - achtergrond

Monte Carlo methode



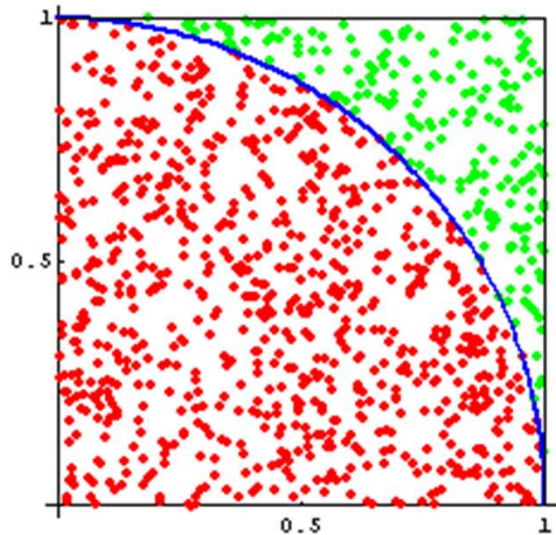
Oppervlakte cirkel: $= \pi * R^2$

als $R = 1$: opp. kwart cirkel $= \pi/4$

random punt in bijgaande figuur:
kans om binnen cirkel te vallen $= \pi/4$
(oppervlakte vierkant $= 1$)

Pi Calculator - achtergrond

Monte Carlo methode: werkwijze



- voortdurend random punten berekenen
- totaal aantal punten tellen (a)
- aantal punten binnen cirkel tellen (b),
dus als $(x^2 + y^2) \leq 1$
- π wordt benaderd door $4 \times (b/a)$
- Voldoende punten, nauwkeurigheid hangt af van toeval.

Pi Calculator - achtergrond

Bailey – Borwein – Plouffe (1995):

The **Bailey–Borwein–Plouffe formula (BBP formula)** is a [spigot algorithm](#) for computing the n th [binary digit](#) of [pi](#) (symbol: π) using [base 16](#) math. The formula can directly calculate the value of any given digit of π without calculating the preceding digits. The BBP is a [summation](#)-style formula that was discovered in 1995 by [Simon Plouffe](#) and was named after the authors of the paper in which the formula was published, [David H. Bailey](#), [Peter Borwein](#), and [Simon Plouffe](#).^[1] Before that paper, it had been published by Plouffe on his own site.^[2] The formula is

$$\pi = \sum_{k=0}^{\infty} \left[\frac{1}{16^k} \left(\frac{4}{8k+1} - \frac{2}{8k+4} - \frac{1}{8k+5} - \frac{1}{8k+6} \right) \right]$$

The discovery of this formula came as a surprise. For centuries it had been assumed that there was no way to compute the n th digit of π without calculating all of the preceding $n - 1$ digits.

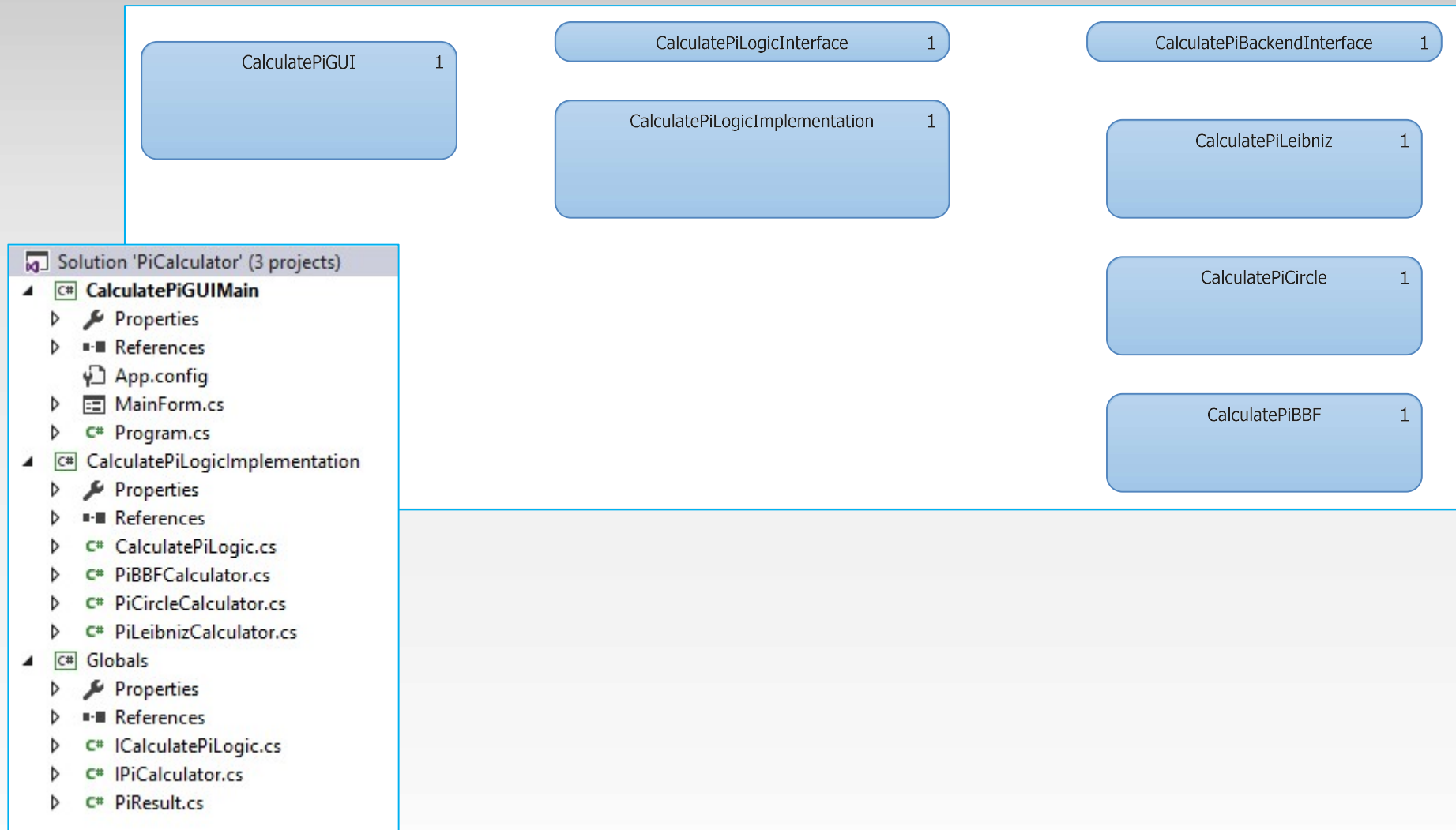
Pi Calculator - toepassing

Vergelijking van benaderingen van Pi:

- nauwkeurigheid
- convergentiesnelheid

Langlopend \Rightarrow in achtergrond threads om gelijktijdig te lopen en interactie mogelijk te maken.

Pi Calculator - architectuur



Pi Calculator - architectuur

'Gezien' vanuit GUI:

```
public interface ICalculatePiLogic
{
    void Start();
    void Pause();
    void Reset();
    void Close();

    event Action<PiResult> PiValue1Changed;

    event Action<PiResult> PiValue2Changed;

    event Action<PiResult> PiValue3Changed;
}
```

Pi Calculator - logische laag

- Creëert drie Pi Calculators
- Geeft commando's door van GUI naar de drie calculators (via de Status property)
- Geeft events door vanuit de calculators naar de GUI

Pi Calculator - architectuur

Communicatie vanuit Logische laag:

```
public struct PiResult
{
    // value calculated for Pi
    public decimal Value { get; set; }

    // NrOfIterations used in calculating Pi
    public long Iterations { get; set; }

    // Difference with previously calculated value for Pi
    public decimal Delta { get; set; }
}
```


Pi Calculator - architectuur

‘Gezien’ vanuit logische laag:

```
// enum used for communication with Task
public enum CalculatorStatus { Running, ResetRunning, ResetPaused, Paused, Closing }

public interface IPiCalculator
{
    /// <summary>
    /// set or get the current status for the Pi calculator
    /// </summary>
    CalculatorStatus Status { get; set; }

    /// <summary>
    /// PiValueChanged is called whenever a new value is calculated
    /// </summary>
    event Action<PiResult> PiValueChanged;
}
```

Pi Calculator - backend

- Start een background taak om het rekenwerk te doen
- Die taak reageert op de waarde van de Status property
- Die taak 'rapporteert' regelmatig over de stand van zaken (niet te vaak: GUI-thread niet verzadigen, hier elke 20ms)
- De rapportering moet 'vertaald' worden naar een event-oproep.

Pi Calculator - DEMO

Calculate Pi

Bailey - BorWein - Plouffe

Pi: 3,1415926535897932384626433834

Delta: 0

Iter.: 30

Leibniz

Pi: 3,1415925013364436647729044268

Delta: -0,0000003045067223284654634946

Iter.: 6 568 000

Circle area

Pi: 3,1414514241123683183769020679

Delta: 0,0000001674890252393009990098

Iter.: 5 126 000

Start

Pauze

Reset

Voorbeeldcode

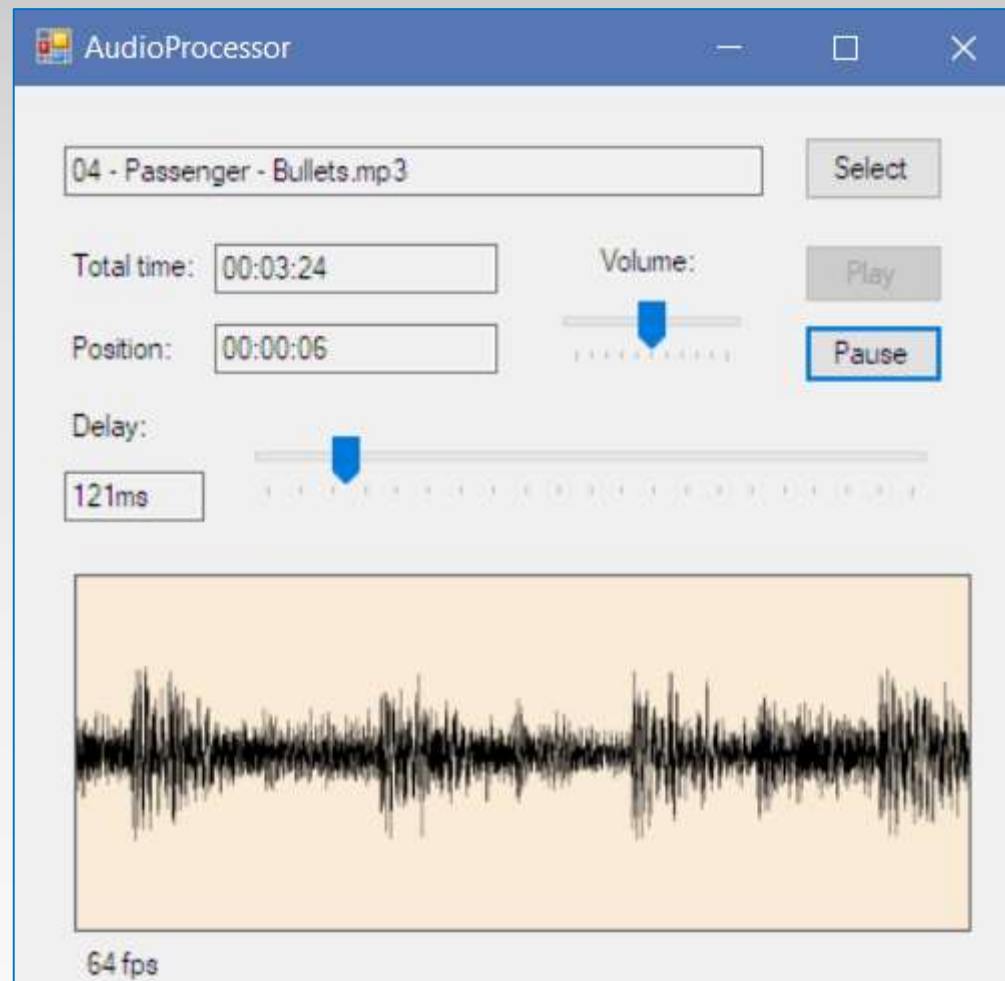
Voorbeeldcode: Audio 'Echo' processor

Bedoeling: een audio player met 'echo' effect

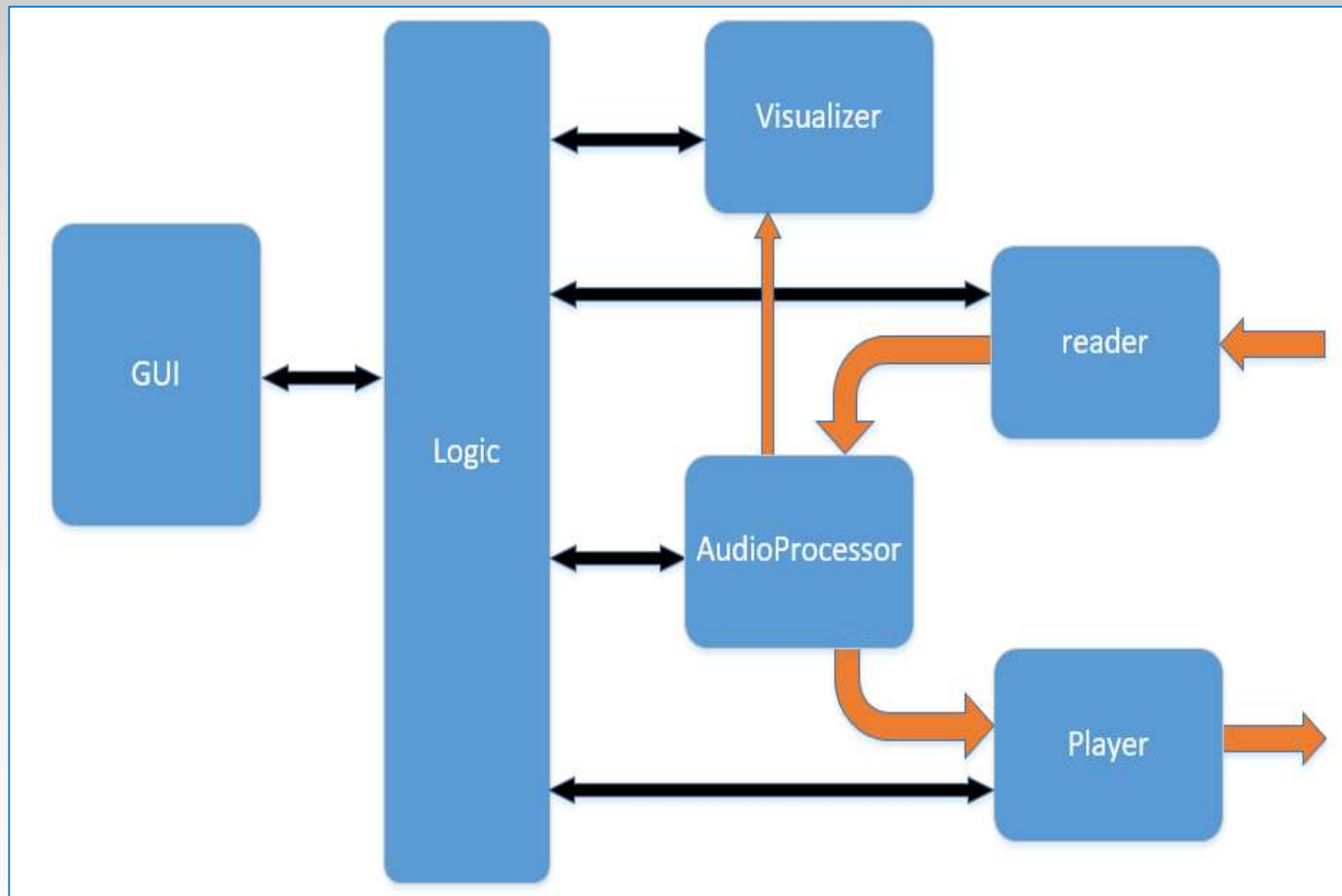
GUI:

- geluidsbestand selecteren ('Select' button)
- 'Play', 'Pauze' buttons
- tijd-indicatie
- visualisatie van geluid ('envelope')
- 'echo' tijd in te stellen via 'TrackBar'

Echo-processor : GUI



Echo-processor : Structuur



Echo-processor : AudioTools (Nuget package)

AudioSampleFrame: bevat een sample voor linker- en rechter-kanaal.

AudioReader: AudioSampleFrames lezen uit een geluidsbestand.

AudioPlayer: Afspelen van geluid (zelf AudioSampleFrames aanleveren).

DelayLine<T>: vertraginglijn

CircularBuffer<T>: sequentieel buffer (uitlezen als Array)