Utils/Consts.md

```
using System.Text.RegularExpressions;


namespace Vache.Utils;

public static class Consts
{
    #region Regex
    /// <summary>
    /// Describes any valid IEEE floating point number
    /// </summary>
    private const string NUM_PATTERN = @"[-+]?(\d+\.?|\d*\.\d+)(e[+-]?\d+)?";

    /// <summary>
    /// Regex matching any valid IEEE floating point number
    /// </summary>
    public static readonly Regex NumberRe = new(NUM_PATTERN);

    /// <summary>
    /// Regex matching the pattern "(a, b)"
    /// </summary>
    public static readonly Regex PointRe = new($@"\({NUM_PATTERN}, ?{NUM_PATTERN}\)");

    /// <summary>
    /// Regex matching the pattern "{a, b}"
    /// </summary>
    public static readonly Regex VectorRe = new($@"\{{{NUM_PATTERN}, ?{NUM_PATTERN}\}}");

    /// <summary>
    /// Regex matching the pattern "(a, b), (c, d), ..." repeating at least thrice
    /// </summary>
    public static readonly Regex PolygonRe = new($@"^(?:\({NUM_PATTERN}, ?{NUM_PATTERN}\)(?:, ?|$))
{{3,}}$");
    #endregion
}
```

Utils/EnumerableExtension.md

```
using System.Diagnostics.Contracts;


namespace Vache.Utils;

public static class EnumerableExtension
{
    /// <summary>
    /// Return adjacent pairs of values from the input
    /// </summary>
    /// <param name="input">The enumerable to enumerate over</param>
    /// <param name="cycle">Whether the last element should be paired with the first</param>
    /// <returns>An enumerable of pairs of values</returns>
    [Pure]
    public static IEnumerable<(T, T)> Pairs<T>(this IEnumerable<T> input, bool cycle = false)
    {
        using IEnumerator<T> enumerator = input.GetEnumerator();

        // If the enumerator is empty, stop
        if (enumerator.MoveNext() is false)
            yield break;

        // Store the element from the previous iteration
        T last = enumerator.Current;
        while (enumerator.MoveNext())
        {
            // Get the current element
```

```
                T curr = enumerator.Current;

                // Yield the previous element and this one
                yield return (last, curr);

                // The current element is now the previous
                last = curr;
            }

            // If we are not cycling the enumerable, stop
            if (!cycle)
                yield break;

            // Reset to the 1st element
            enumerator.Reset();
            enumerator.MoveNext();

            // Return the very last element and the first one
            yield return (last, enumerator.Current);
        }
    }
```