

GalaxyGuide

The Merchant's Guide to the Galaxy

Developed with [Microsoft Dotnet Core 3.1](#).

The application is composed by 4 components:

- `GalaxyGuide.App` which is the UI of the application;
- `GalaxyGuide.FiniteStateMachine` which implements a Finite State Machine used for string recognition;
- `GalaxyGuide.RomanNumeralsConverter` which is a Roman Numerals to Arabic Numerals converter;
- `GalaxyGuide.Mediator` which puts together the `GalaxyGuide.FiniteStateMachine` and the `GalaxyGuide.RomanNumeralsConverter` components through the `GalaxyEngine` class to provide the application logic.

In the solution there are also two test projects:

- `RomanNumeralsConverterTests` which provides some unit tests for the `GalaxyGuide.RomanNumeralsConverter`
- `IntegrationTests` which provides the integration tests.

Solution Structure

```
src\  
|  
+- GalaxyGuide.App  
|  
+- GalaxyGuide.FiniteStateMachine  
|  
+- GalaxyGuide.Mediator  
|  
+- GalaxyGuide.RomanNumeralsConverter  
|  
+- IntegrationTests  
|  
+- RomanNumeralsConverterTests
```

GalaxyGuide.RomanNumeralsConverter

The main classes in the component are the `Converter` and `ValidationRules` classes, which implement respectively the `IConverter` and the `IValidationRules` interface. The `ValidationRules` class provides through its interface three methods for Roman Numerals Validation:

```

void SymbolsValidation(string roman);
void SubtractionValidation(string roman);
void SymbolsRepetitionValidation(string roman);

```

The `ValidationRules` class is "injected" in the `Converter` class through the constructor:

```

public Converter(IValidationRules validationRules)
{
}

```

The `Converter` class provides through its interface the method `Convert`

```

int Convert(string roman);

```

which convert a Roman Numerals to Arabic Numeral.

GalaxyGuide.FiniteStateMachine

In the `FiniteStateMachine` component there is the `PatternRecognizer` class used for string recognition. the `FiniteStateMachine` implements the `IPatternRecognizer` interface:

```

ParseResult Parse(string sentence);

```

The `Parse` method allows to parse a string based on the specified grammar.

The grammar can be provided to the class through its constructor:

```

public PatternRecognizer(
    string startSymbol,
    string[] endSymbols,
    List<Tuple<string, string, string>> prods)
{
}

```

- `startSymbol` is the initial symbol (state) of grammar (usually "S");
- `endSymbol` are the final symbols (states) of the machine (usually "Z");
- `prods` is a list of production in the form (A,a,B), where "a" move the state from "A" to "B"

For more information on "Finite-State Machine" visit https://en.wikipedia.org/wiki/Finite-state_machine

GalaxyGuide.Mediator

The `GalaxyGuide.Mediator` component provides the `GalaxyEngine` class. The `GalaxyEngine` class implements the `IGalaxyEngine` interface where is defined the `Evaluate` method:

```
string Evaluate(string sentence);
```

The `GalaxyEngine` puts together the `GalaxyGuide.FiniteStateMachine` and the `GalaxyGuide.RomanNumeralsConverter` components, provides the grammar to the `PatternRecognizer` class and implements the application logic.

The `Evaluate` method can be invoked to have a "sentence" evaluated by the `GalaxyEngine`.

Compile the solution

Move to `src` folder and type:

```
dotnet build
```

Run the unit and integration tests

Move to `src` folder and type:

```
dotnet test
```

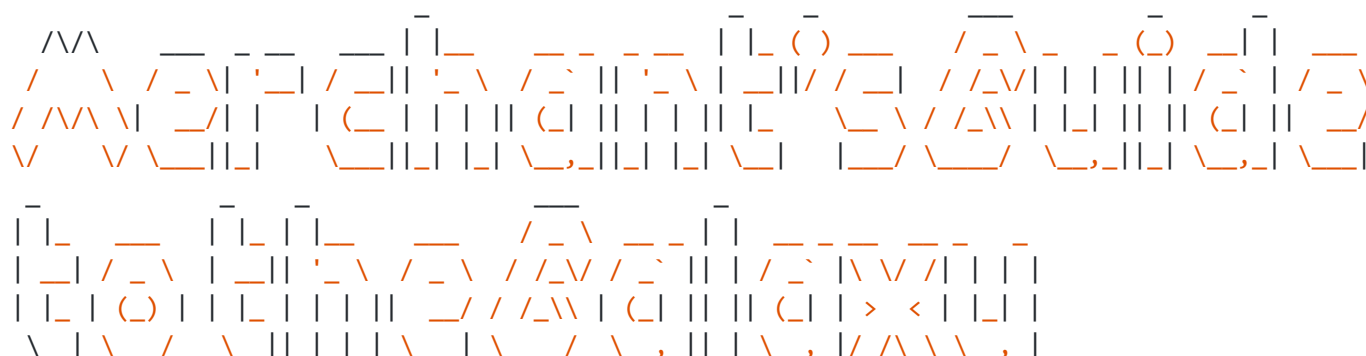
Run the application

Move to `src` folder and type:

```
dotnet run -p GalaxyGuide.App
```

Alternately on Windows OS, you can double-click the GalaxyGuide.App.exe file in "src\GalaxyGuide.App\bin\Debug\netcoreapp3.1" folder.

When the application starts the following screen is shown:





Type `exit` or `quit` to terminate
Type `demo` to run a demo

Merchant's Guide> _

You can type `quit` or `exit` to quit, or `demo` to launch a demo.

If you type `demo` the following screen is shown:

```
Merchant's Guide> demo
Merchant's Guide> glob means I
Merchant's Guide> prok means V
Merchant's Guide> pish means X
Merchant's Guide> tegj means L
Merchant's Guide> glob glob units of Silver are worth 34 Credits
Merchant's Guide> glob prok units of Gold are worth 57800 Credits
Merchant's Guide> pish pish units of Iron are worth 3910 Credits
Merchant's Guide> how much is pish tegj glob glob ?

pish tegj glob glob is 42

Merchant's Guide> how many Credits is glob prok Silver ?

glob prok Silver is 68 Credits

Merchant's Guide> how many Credits is glob prok Gold ?

glob prok Gold is 57800 Credits

Merchant's Guide> how many Credits is glob prok Iron ?

glob prok Iron is 782 Credits

Merchant's Guide> how much wood could a woodchuck chuck if a woodchuck could chuck wood ?

I have no idea what you are talking about

Syntax Error:
how much wood could a woodchuck chuck if a woodchuck could chuck wood ?
      ^^^^

Merchant's Guide>
```

The application detects and highlights the syntax errors. In this example the word **Credits** (insted of *Credits*) is detected:

Merchant's Guide> glob means I
Merchant's Guide> prok means V
Merchant's Guide> pish means X
Merchant's Guide> tegj means L
Merchant's Guide> glob glob units of Silver are worth 34 Credits

I have no idea what you are talking about

Syntax Error:

glob glob units of Silver are worth 34 Credits
^^^^^^

Merchant's Guide>