Customized Neovim IDE for the Alan Programming Language

Dirk Wouter Bester
Stellenbosch, South Africa
25137018@sun.ac.za

1 INTRODUCTION

In software development, the readability of code is essential. Integrated Development Environments (IDEs) and text editors play a crucial role in enhancing readability by providing features like syntax highlighting, code formatting, and autocompletion. Unfortunately, there exists no IDEs or text editors that provide these features for the Alan language. Using Neovim, we embarked on a journey to create the ultimate Alan IDE. This report documents the process of creating a custom Treesitter grammar for Alan, configuring Neovim to use this grammar for syntax highlighting, developing a code formatter to ensure consistent code style, and adding a code autocomplete feature.

2 PROBLEM DESCRIPTION

Alan is an educational and simple language that we have a 67-page document for. Explaining the language fully and explaining each aspect in detail. We have implemented this language with a few different frameworks. What if we wanted to start coding in Alan and use the language that was created for us? Alan currently has no comprehensive tooling support like other widely used languages which makes it very hard to code using the language.

This challenge motivated the development of a solution that integrates Neovim and Treesitter. By addressing this issue, we aim to improve the overall development experience for Alan programmers.

3 APPROACH

To solve this problem for the Alan programming language, we implemented a solution using Neovim that makes use of additional plugins. The approach was divided into several key phases: setting up the development environment, creating a Treesitter grammar for Alan, configuring Neovim for syntax highlighting, developing a custom code formatter, and making an autocompleter.

3.1 Setting Up the Development Environment

We began the project by installing neovim and figuring out how it works. Here we realized Neovim is a text editor that can make use of plugins to customize and personalize your coding workplace. So I installed packer which is a plugin manager, to organize and customize all my plugins.

3.2 Creating a Treesitter Grammar for Alan

The next step was to create a Treesitter grammar for Alan. Treesitter uses a formal grammar to parse source code into a syntax tree. We defined the Alan grammar in a file that specifies the language's syntax rules, including keywords, operators, and various code constructs. This grammar enables Treesitter to accurately parse Alan code, which is essential for providing precise syntax highlighting. I tested the newly configured grammar with the random examples given on the Sunlearn website.

3.3 Configuring Neovim for Syntax Highlighting

With the Treesitter grammar in place, we configured Neovim to use this grammar for syntax highlighting. This involved setting up Treesitter configurations in Neovim's initialization file (init.lua). We specified the Alan grammar and enabled Treesitter's syntax highlighting capabilities. Custom highlight queries were also added to ensure that different parts of the Alan code were correctly color-coded. You can highlight keywords in certain contexts which I found interesting. I also made use of (motlin.medium.com) to ensure that I was using the correct color combinations.

3.4 Developing a Custom Code Formatter

To address sloppy Alan code we decided to incorporate a very simple code formatted. It does not create indents but it does format the begin, end and function keywords to their own lines. This ensures the blocks of codes can be clearly seen. To run the code formatter you have to type AlanFormat() in the command line of Neovim.

3.5 Developing Autocompletion

To further enhance the development experience, we integrated LuaSnip for autocompletion of common Alan constructs. Custom snippets were defined for frequently used patterns, such as function declarations and program blocks. These snippets provide real-time suggestions as the user types, speeding up the coding process.

4 IMPLEMENTATION

The primary tools and plugins used were:

- **Neovim**: A text editor that supports modern development workflows.
- **Packer**: A Neovim plugin manager, used to manage and install other plugins.
- **Treesitter**: A parser generator tool and incremental parsing library.
- **LuaSnip**: A snippet engine for Neovim used to provide autocompletion features.

4.1 Development Environment Setup

The following code snippet shows how to install the necessary plugins using Packer in the Neovim configuration file (‘init.lua‘):

Listing 1: Neovim Plugin Setup

```
require('packer').startup(function(use)
    use 'wbthomason/packer.nvim'
    use 'hrsh7th/nvim-cmp'
    use 'hrsh7th/cmp-nvim-lsp'
    use 'hrsh7th/cmp-buffer'
    use 'hrsh7th/cmp-path'
    use 'hrsh7th/cmp-cmdline'
```

```
    use 'saadparwaiz1/cmp_luasnip'
    use 'L3MON4D3/LuaSnip'
    use 'rafamadriz/friendly-snippets'
    use 'neovim/nvim-lspconfig'
    use { 'nvim-treesitter/nvim-treesitter',
    run = ':TSUpdate' }
end)
```

4.2 Treesitter Grammar for Alan

Creating a Treesitter grammar for Alan involved defining the syntax rules in a ‘grammar.js‘ file.

Listing 2: Alan Treesitter Grammar

```
module.exports = grammar({
  name: 'alan',
  rules: {
    source_file: $ => seq(
      'source', $.identifier,
      repeat($.funcdef), $.body
    ),
```

4.3 Neovim Configuration for Syntax Highlighting

We configured Neovim to use the Alan Treesitter grammar for syntax highlighting by setting up Treesitter configurations and specifying custom highlight queries. Here is an example of the Highlights query:

Listing 3: Highlights Configuration

```
(source_file
  "source" @label
  (identifier) @label)
(body
  "begin" @string.special.url
  "end" @string.special.url)
(vardef) @property
(funcdef) @property
(statements) @string
(expression) @number
(identifier) @number

"while" @comment.warning
"if" @comment.warning
"do" @comment.warning
"elsif" @comment.warning
"then" @comment.warning
"else" @comment.warning

(while_stmt
  "end" @comment.warning)

(if_stmt
  "end" @comment.warning)
```

4.4 Custom Code Formatter

The custom code formatter was implemented in Python. The formatter reads Alan source code, applies consistent formatting rules, and outputs the formatted code. It was written using Lua and is also in the init.lua config file.

4.5 Autocomplete feature

The autocomplete makes use of Lua snippets and is very easy to use. Here is an example of how it is used:

Listing 4: Lua Snippets

```
luasnip.add_snippets('all', {
  luasnip.snippet('source', {
    luasnip.text_node('source program_name
    begin relax end'),
  }),
```

5 RESULTS

The Neovim text editor was transformed from an unreadable boring code editor to an exciting, visually appealing code editor. The before and after can be seen on the next page.

6 REFERENCES

- `nvim-cmp`: https://github.com/hrsh7th/nvim-cmp
- `cmp-nvim-lsp`: https://github.com/hrsh7th/cmp-nvim-lsp
- `cmp-buffer`: https://github.com/hrsh7th/cmp-buffer
- `cmp-path`: https://github.com/hrsh7th/cmp-path
- `cmp-cmdline`: https://github.com/hrsh7th/cmp-cmdline
- `cmp-luasnip`: https://github.com/saadparwaiz1/cmp*luasnip*
- `LuaSnip`: https://github.com/L3MON4D3/LuaSnip
- `friendly-snippets`: https://github.com/rafamadriz/friendly-snippets
- `nvim-lspconfig`: https://github.com/neovim/nvim-lspconfig
- `nvim-tree.lua`: https://github.com/nvim-tree/nvim-tree.lua
- `nvim-web-devicons`: https://github.com/nvim-tree/nvim-web-devicons
- `nvim-treesitter`: https://github.com/nvim-treesitter/nvim-treesitter
- `YouTube Video`: https://www.youtube.com/watch?v=y1WWOaLCNyI
- `Medium Article`: https://motlin.medium.com/how-to-pick-colors-for-a-syntax-highlighting-theme-96d3e06c19dc

Customized Neovim IDE for the Alan Programming Language