VŠB TECHNICKÁ UNIVERZITA OSTRAVA
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

Podpora jazyka Monkey C v prostředí VS Code

Monkey C Language Support in VS Code

Jakub Pšenčík

Bakalářská práce

Vedoucí práce: Ing. Jan Janoušek

Ostrava, 2021

Abstrakt

V této bakalářské práci se budu zabývat vývojem rozšíření pro Visual Studio Code, jenž bude poskytovat plnou podporu jazyka Monkey C. Součástí práce bude bude představení prostředí Visual Studio Code a problematika vývoje rozšíření v tomto prostředí. Představen bude, mimo jiné, také jazyk Monkey C. V další části práce se budu zabývat nástrojem ANTLR, díky kterému jsme schopni generovat vlastní překladač jazyka pomocí bezkontextové gramatiky, parsováním kódu a jeho syntaktickou analýzou. Závěrem bude výsledné rozšíření testováno.

Klíčová slova

bakalářská práce, rozšíření, Monkey C, Typescript

Abstract

In this bachelor thesis I will deal with the development of extension for Visual Sstudio Code, which will provide full support for Monkey C language. The work will include an introduction to the Visual Studio Code environment and the development of extensions in this environment. Among other things, the Monkey C language will be introduced. In the next part of the work I will describe the ANTLR tool, used to generate our own language compiler using context-free grammar, code parsing and its syntactic analysis. Finally, the resulting extension will be tested.

Keywords

bachelor thesis, extension, Monkey C, Typescript

Poděkování

Rád bych na tomto místě poděkoval svému vedoucímu práce Ing. Janu Janouškovi za odborné a metodické vedení, ochotu a pomoc v průběhu vypracovávání práce.

Obsah

Seznam použitých zkratek a symbolů

ANTLR	–	ANother Tool for Language Recognition
AST	–	Abstract syntax tree
VS Code	–	Visual Studio Code

Seznam obrázků

Seznam tabulek

Kapitola 1

Úvod

V dnešní době, kdy jsme obklopeni spoustou moderních technologií, existující programovací jazyky se stále vyvíjejí kupředu a nové postupně vznikají, existuje nespočet nástrojů, rozšíření, vývojových prostředí, které práci a vývoj na těchto jazycích dokážou v mnoha ohledech usnadnit. Jazyk MonkeyC, jenž bude středobodem této práce, zatím nedisponuje tak široce obsáhlou komunitou, jako mají např. v dněšní době velmi populární Javascript, Python, C Sharp, Ruby, atd...

Typickým příkladem společnosti, která se zabývá právě vývojem softwarů pro programátory či vývojáře, je česká JetBrains s.r.o. Ti mají na kontě již mnoho produktů usnadňujících programování, např. ReSharper (.NET), IntelliJ IDEA (Java, Groovy, atd...) či PyCharm (Python). Ovšem na podporu Monkey C zatím žádné softwarové řešení v JetBrains do světa nevypustili. [1]

Na oficiálním webu [2] sice najdeme několik nástrojů, které s vývojem v Monkey C pomáhají, my ale budeme chtít, aby naše aplikace poskytovala co největší podporu uživateli a hlavně, aby byla postavené na našem vlastním řešení.

V následujících kapitolách se tedy budeme věnovat tvorbě a vývoji rozšíření pro vývojové prostředí Visual Studio Code, které poskytne plnou podporu při vývoji aplikací. Jako první si představíme jazyk Monkey C, jak se s ním pracuje, jaké aplikace s ním můžeme vyvíjet atd... Pro tvorbu aplikace využijeme jazyk Typescript, dále pak nástroj ANTLR, který se schopný vygenerovat předladač jazyka z jeho popisu obsaženého v bezkontextové gramatice. Další část práce bude věnováná tématům, jako jsou parsování kódu, vytvoření syntaktického a sémantického stromu, našeptávání kódu, atd...

Kapitola 2

Jazyk Monkey C

Monkey C [3] je objektově orientovaný jazyk navržený pro snadný vývoj Connect IQ aplikací. Jedná se o dynamický programovací jazyk, podobně jako Java, PHP či Ruby. Cílem Monkey C je zjednodušit proces vytváření samotné aplikace a umožnit tak vývojářům více se soustředit na zákazníka a méně na omezení zdrojů. Využívá tzv. "reference counting"k automatickému čištění paměti, což vývojáře osvobozuje od manuální správy paměti (např. jako v jazyce C/C++).

```
using Toybox.Application as App;
using Toybox.System;

class MyProjectApp extends App.AppBase {

// onStart() is called on application start up
function onStart(state) {
}

// onStop() is called when your application is exiting
function onStop(state) {
}

// Return the initial view of your application here
function getInitialView() {
return [ new MyProjectView() ];
}
}
```

Obrázek 2.1: ukázka jednoduchého fragmentu kódu v MonkeyC

Kapitola 3

Problematika vývoje rozšíření pro VS Code

3.1 Visual Studio Code

Visual Studio Code [4] je editor zdrojového kódu vytvořený společností Microsoft pro operační Windows, Linux a MacOS. Nabízí spoustu užitečných funkcí, mezi které patří podpora ladění kodu, zvýrazňování syntaxe, automatické doplňování a našeptávání kódu, atd... Naším cílem je integrovat většinu těchto funkcí a možností do finálního rozšíření.

3.2 Typescript

TypeScript [5] je open-source programovací jazyk vyvinutý společností Microsoft. Jedná se o nádstavbu nad jazykem JavaScript, která jej rozšiřuje o statické typování a další atributy, které známe z objektově orientovaného programování jako jsou třídy, moduly a další. Samotný kód psaný v jazyce TypeScript se kompiluje do jazyka JavaScript. Jelikož je tento jazyk nádstavbou nad JavaScriptem, je každý JavaScript kód automaticky validním TypeScript kódem.

3.3 Komponenty potřebné pro vytvoření rozšíření

Předtím, než začneme pracovat na samotném rozšíření, je potřeba obstarat několik důležitých nastrojů a komponent, jenž jsou klíčové při vývoji.

3.3.1 Gramatika

Jako první je potřeba definovat popis jazyka MonkeyC. V našem případě je jazyk popsán prostřednictvím bezkontextové gramatiky, která formálně definuje syntax (pravidla) jazyka.

Na obrázku mužeme vidět prvních pár řádků gramatiky pro MonkeyC. Gramatika obsahuje spoustu známých klíčových slov, nebo-li tokenů, např. "CLASS", "FUNCTION", "USING", atd...

```
grammar MonkeyC;

options {
    superClass=MonkeyCBaseParser;
}
@header {import { MonkeyCBaseParser } from "../MonkeyCBaseParser";}


    DOT : '.';
    SEMI : ';';
    QUES : '?';
    COLON : ':';
    CLASS : 'class';
    FUNCTION : 'function';
    RETURN : 'return';
    NEW : 'new';
    VAR : 'var';
    CONST : 'const';
    MODULE : 'module';
    USING : 'using';
    AS : 'as';
    ENUM : 'enum';
    EXTENDS : 'extends';
```

Obrázek 3.1: ukázka hlavičky gramatiky MonkeyC.g4 pro popis jazyka

3.3.2 Java

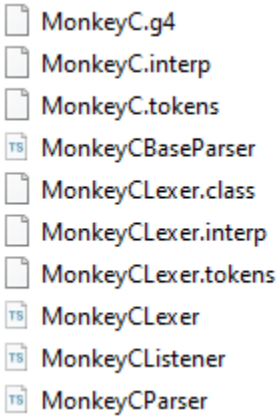
Jelikož je ANTLR psán v Javě, je potřeba ji nainstalovat na naše zařízení, přičemž se požaduje verze 1.6 nebo vyšší. Poté již stačí stáhnout akualní ANTLR jar, což je momentálně "antlr-4.8-complete.jar".

3.3.3 Parser a Lexer

lexer - lexer (nebo také tokinizér) "rozdělí"text na vstupu (v našem případě MonkeyC kód) na tokeny. [6]
parser - shora dolů prochází text na vstupu a porovnává jednotlivé řádky s pravidly obsažené v gramatice.
K vytvoření parseru a lexeru je potřeba spustit ANLTR nástroj, který s pomocí gramatiky tyto souboru vygeneruje. [6]

3.3.4 TestRig

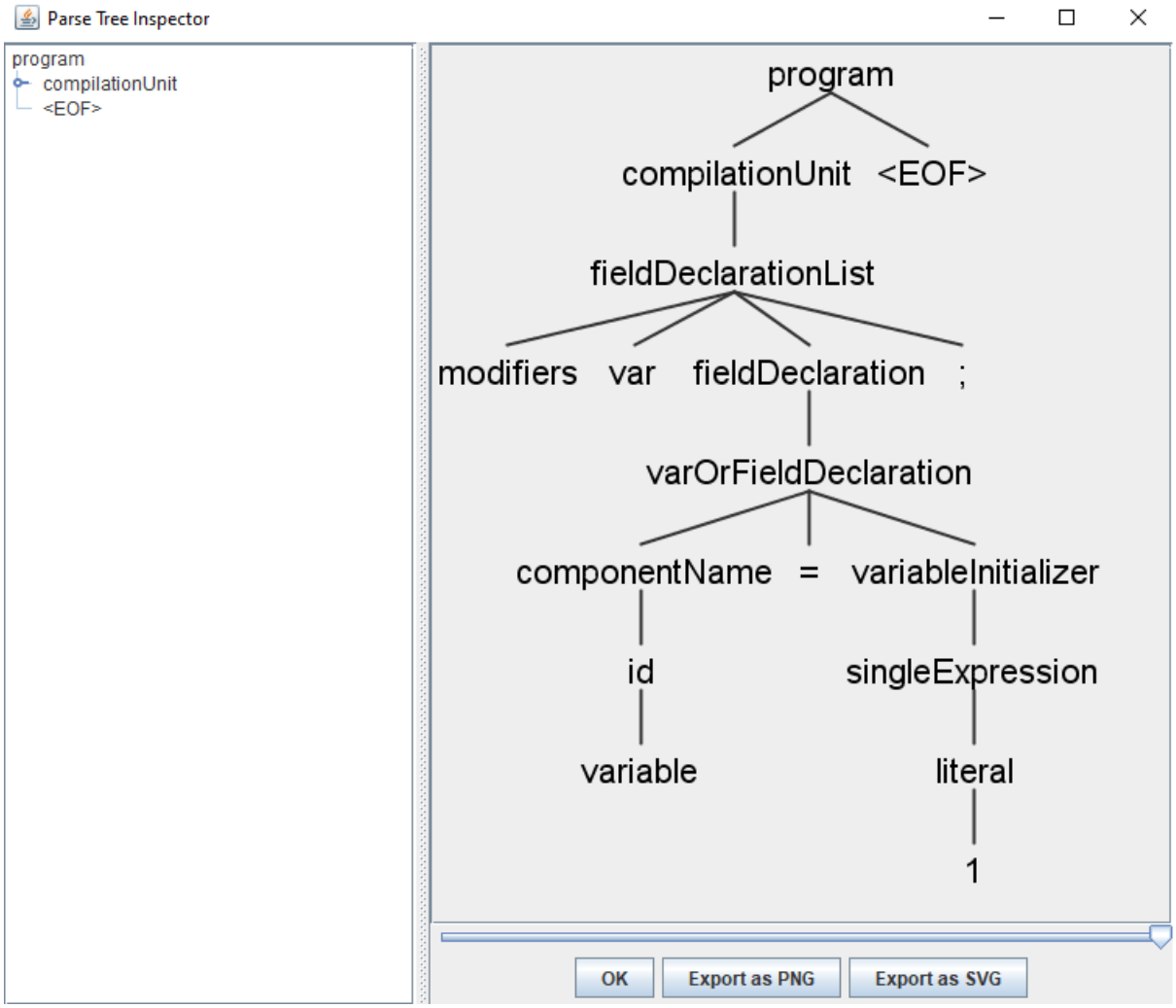
ANTLR poskytuje flexibilní testovací nástroj umístněný v runtime knihovně s názvem TestRig. TestRig dokáže poskytnout spoustu informací o tom, jak "recognizéry"(parser a lexer) zpracovávají InputStream ze vstupního souboru.



Obrázek 3.2: potřebné soubory vygenerované nástrojem ANTLR

Nejjednodušší způsob ověření, zda gramatika rozpoznává vstup správně, je zobrazit si vygenerovaný syntaktický strom vizuálně.

Na obrázku můžeme detailně vidět jak parser vytvořil syntaktický strom a jak jsou jednotlivé části propojené. Strom začíná kořenem, jenž je v naší gramatice pojmenová, jako "program". Takto je kořen pojmenován při každém parsování.



Obrázek 3.3: "ParseTreeInspector"- vizuální podoba syntaktického stromu

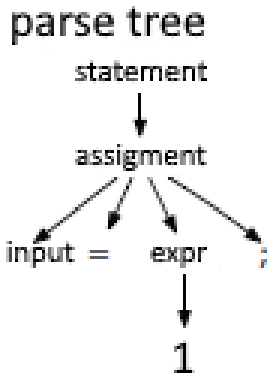
Kapitola 4

Syntaktická a sémantická analýza kódu

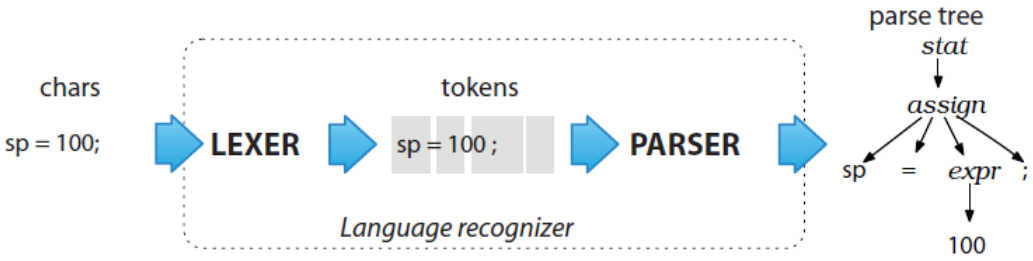
Abychom mohli implementovat náš jazyk (MonkeyC), je potřeba vytvořit aplikaci, která je schopna číst "věty"na vstupu a odpovídajicím způsobem rozpoznává klíčová slova či fráze naší gramatiky.(Obecně je jazyk složení platných vět, kdy věta se skládá z frází a fráze se skládá ze symbolů slovní zásoby).
Obecně lze říci, pokud nějaká aplikace vykonává (interpretuje) zápis jiného programu v jeho zdrojovém kódu ve zvoleném programovacím jazyce (v našem případě Typescript), že se jedná o tzv. interpret [7]. Jako příklady uvedu kalkulačku, aplikace pro čtení konfiguračních souborů, Python interprety, atd... Pokud, na druhou stranu, převádíme "věty"z jednoho jazyka do druhého (např. z Javy do 'C Sharp'), nazýváme takovou aplikaci překladačem.
Úkolem překladače či interpreta je tedy rozpoznat platné vstupy, tedy věty, fráze, subfráze, klíčová slova, atd... Přičemž rozpoznáním platného vstupu je myšleno identifikovat jednotlivé fráze a rozlišit je od jiných.
Jako příklad použijeme rozpoznání vstupu $input = 1;$ s validní MonkeyC syntaxí. Víme tedy, že "input"je cíl přiřazení a "1"představuje hodnotum která se má přiřadit. Stejně jako jsme my schopni rozlišit v našem jazyce sloveso od podstatného jména, je naše aplikace schopna rozlišit toto přiřazení od např. importu knihovny pomocí klíčového slova "using".
Programy, které jsou schopny rozpoznávat konkrétní jazyk, se nazývají parsery nebo syntaktické analyzátory, přičemž syntax odkazuje na pravidla obsažená v popisu jazyka, čili gramatice.

4.1 ANTLR - Nástroj pro generování překladače

ANTLR nebo také ANother Tool for Language Recognition (jiný nástroj pro rozpoznávání jazyka) [8] je výkonný nástroj pro generování syntaktických analyzátorů, tzv. "parser". Tento parser je poté schopen číst, zpracovávat, spouštět nebo překládat strukturované textové či binární soubory. Používá



Obrázek 4.1: přiřazení hodnoty 1 proměnné input



Obrázek 4.2: Ukázka, jak Language recognizer zpracovává vstupní sekvenci znaků [10]

se především k vytváření nových jazyků, nástrojů nebo "frameworků". Využívá bezkontextový jazyk typu LL.

4.1.1 LL syntaktický analyzátor

LL [9] je syntaktický analyzátor shora-dolů pro bezkontextové gramatiky. Analyzuje vstup zleva (Left) doprava a konstruuje nejlevější derivaci (Leftmost) věty. Gramatiky, které jsou takto analyzovatelné, se nazývají LL gramatiky.

4.2 Syntaktický strom

Syntaktický strom představuje datovou strukturu, kterou ANTLR vytvoří při parsování vstupního souboru. Struktura je složená z kořene (root) a uzlů, které mohou představovat buď další podstromy, které odpovídají pravidlům gramatiky, nebo listy stromu.
V naší aplikaci Syntaktický strom představuje třída AST (viz. kód níže), která uchovává všechna důležitá data pro korektní analýzu vstupního souboru, např. počet uzlů stromu, soubor vztahující se ke konkrétnímu stromu, kořen stromu, atd...

```
export class AST {

    static nodeCount: number = 0;
    private parseTree : Node[];
    public documentName: string;
    public currentNode : Node;
    public root : Node;

    constructor(documentName: string) {

        this.parseTree = [];
        this.documentName = documentName;
        this.currentNode = new Node(undefined,undefined,undefined,undefined,
            undefined);
        this.root= new Node(undefined,undefined,undefined,undefined,undefined);

    }


    getParseTree() {
        return this.parseTree;
    }

    addNode(node : Node) : void {
        this.parseTree.push(node);
    }

    findNode(ruleNumber: number) : Node | undefined {

        for(let i = this.parseTree.length-1; i >= 0; i--) {
            if(this.parseTree[i] != null && this.parseTree[i].getContext()?.
                ruleIndex === ruleNumber) {
                return this.parseTree[i];
            }
        }
        return undefined;
    }
```

```
}
```

Listing 4.1: třída AST v jazyce Typescript

Další, neméně důležitou, třídou v naší apikaci je třída Node, která představuje uzel stromu. Jedná se, ve své podstatě, o strukturu uchovávající následující atributy:

1. kontext daného uzlu
2. předka uzlu
3. potomka uzlu
4. hodnotu přestavující text v daném uzlu
5. typ uzlu, který se odvíjí od pravidel v gramatice

```
export class Node {

    private id : number;
    private context : ParserRuleContext | undefined ;
    private parent : Node | undefined;
    private child : Node[] | undefined;
    private value: string | undefined;
    private _type : number | undefined;

    constructor(context: ParserRuleContext | undefined, parent : Node | undefined,
         child : Node[] | undefined, value : string | undefined, _type: number |
        undefined)

    {
        this.id = AST.nodeCount;
        this.context = context;
        this.parent = parent;
        this.child = child;
        this.value = value;
        this._type = _type;

        AST.nodeCount++;

    }
```

```
    getId() { return this.id; }
    getContext() { return this.context; }
    getParent() { return this.parent; }
    getChildren() { return this.child; }
    getValue() {return this.value; }
    getType() {return this._type; }
    addChild(child : Node) : void{ this.child?.push(child); }

}
```

Listing 4.2: třída Node v jazyce Typescript

4.3 ANTLR Listener a callback funkce

ANTLR disponuje dvěma mechanismy, které umožňují průchod stromem (tzv. "tree-walking mechanisms"). Jedná se o mechanismy **listener** a **visitor**,pričemž výchozím je listener. Největší rozdíl mezi nimi spočívá v tom, že listener metody jsou volány nezávisle ANTLR objektem, zatímco visitor metody vyvolávají rovněž metody potomků uzlu, na kterém se právě nachází, což způsobuje, že některé podstomy nebudou při průchodu vůbec navštíveny. Samotné listenery jsou ekvivalentí SAX objektům, které se používají v XML parserech.
Aby bylo možné stromem procházet a při průchodu vyvolat odpovídající události, ANTLR dispinuje třídou ParseTreeWalker. Právě ona se stará o to, aby byly volány callback funkce. Další důležitou komponentou, vygenerovanou ANTLR nástrojem je rozhraní ParseTreeListener (v našem případě MonkeyCListener 3.2). Toto rozhraní definuje veškeré metody potřebné k kompletnímu průchodu stromem, např. "enterEveryRule(context: ParserRuleContext)"či "exitStatement(context: StatementContext)". Ve chvíli kdy ParseTreeWalker narazí na příslušný uzel, vyvolá odpovídající metodu. TODO: příklad kdy narazí na nějaké assign třeba

4.4 Sémantický strom

Strom, jenž hraje zásadní roli v procesech, jako je našeptávání kódu, detekci chyb, kdy je použita třída z modulu, který není referencován, atd...

4.5 parser

Fusce nibh. Sed ut perspiciatis unde omnis iste natus error sit voluptatem accusantium doloremque laudantium, totam rem aperiam, eaque ipsa quae ab illo inventore veritatis et quasi architecto

beatae vitae dicta sunt explicabo. Quis autem vel eum iure reprehenderit qui in ea voluptate velit esse quam nihil molestiae consequatur, vel illum qui dolorem eum fugiat quo voluptas nulla pariatur? Etiam ligula pede, sagittis quis, interdum ultricies, scelerisque eu. Maecenas sollicitudin. Cras pede libero, dapibus nec, pretium sit amet, tempor quis. Integer vulputate sem a nibh rutrum consequat. Pellentesque sapien. Pellentesque arcu. Suspendisse nisl. Fusce consectetuer risus a nunc. Etiam dui sem, fermentum vitae, sagittis id, malesuada in, quam. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Nam quis nulla. Nulla non lectus sed nisl molestie malesuada. Duis viverra diam non justo. Sed ac dolor sit amet purus malesuada congue. Aenean id metus id velit ullamcorper pulvinar. Aliquam ornare wisi eu metus. Neque porro quisquam est, qui dolorem ipsum quia dolor sit amet, consectetur, adipisci velit, sed quia non numquam eius modi tempora incidunt ut labore et dolore magnam aliquam quaerat voluptatem.

Kapitola 5

Návrh a implementace rozšíření

Rozšíření bude, jak již bylo nastíněno výše, implementováno v jazyce Typescript. K vytvoření rozšíření samotného je potřeba Node.js a Git. Poté je vyžadována instalace programu "Yeoman""VS Code Extension Generator".

5.1 Error Listener

Slouží k zachytávní syntaktických chyb v kódu, čili chyb, které jsou v rozporu z pravidly gramatiky. Rozšíření chyby vypisuje do konzole, jak jsme ve VS Code zvyklí. Součástí zprávy jsou:

1. řádek, na kterém se chyba nachází
2. pozice na řádku
3. popis chyby

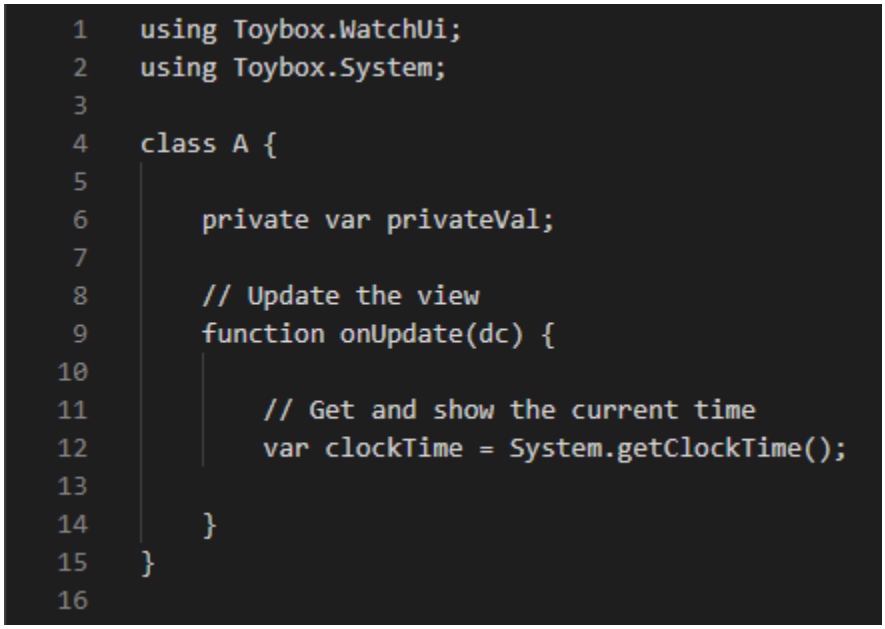
5.2 Modul Toybox

Tento modul (modul = namespace) obsahuje všechny potřebné třídy, se kterými v Monkey C můžeme pracovat.

5.3 Zvýraznění kódu

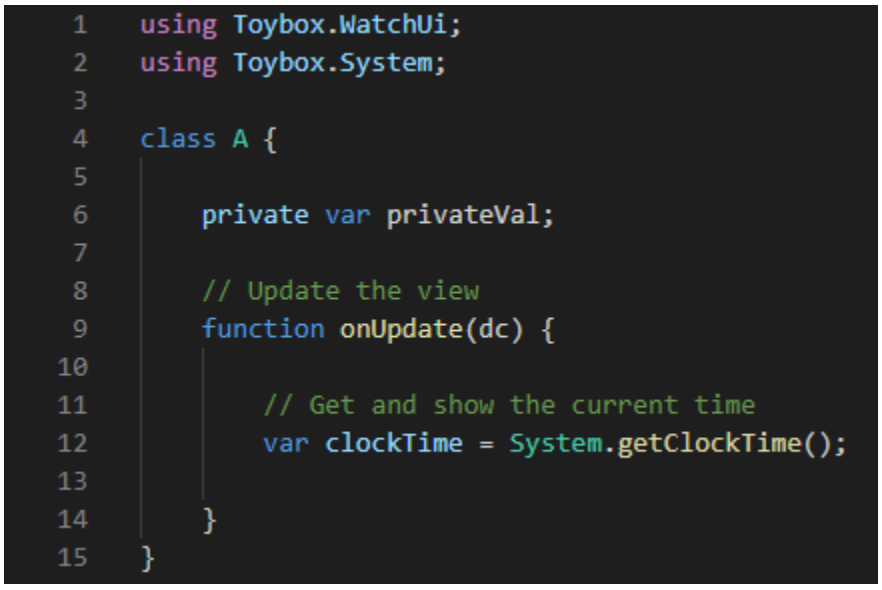
Zvýraznění, nebo-li obarvení klíčových slov kódu je realizováno pomocí JSON souboru, jenž pomocí regulárních výrazů v textu vyhledává příslušná slova a ty poté obarvuje.

Hned na první pohled je zřejmé, že obarvení poskytuje uživateli větší přehled a orientaci v kódu, jak je vidět na obrázku 5.2



```
using Toybox.WatchUi;
using Toybox.System;

class A {

    private var privateVal;

    // Update the view
    function onUpdate(dc) {

        // Get and show the current time
        var clockTime = System.getClockTime();

    }
}

```

Obrázek 5.1: ukázka neobarveného kódu v MonkeyC



```
using Toybox.WatchUi;
using Toybox.System;

class A {

    private var privateVal;

    // Update the view
    function onUpdate(dc) {

        // Get and show the current time
        var clockTime = System.getClockTime();

    }
}
```

Obrázek 5.2: ukázka obarvení kódu v MonkeyC

5.4 Automatické doplňování a našeptávání kódu

Hlavním aktérem při našeptávání vhodných částí kódu je "antlr4-c3 The ANTLR4 Code Completion Core".

Jedná se o "stroj"sloužící k dokončování gramatického agnostického kódu pro analyzátory založené na ANTLR4. Engine c3 je schopen poskytnout kandidáty na doplnění kódu, kteří jsou užiteční pro editory s analyzátory generovanými ANTLR, nezávisle na skutečném jazyku / gramatice použité pro generování.

Původní implementace je poskytována jako "node module"a je psána v jazyce Typescript.

Pro zobrazení možných symbolů ve zdrojovém kódu zjevně potřebujete zdroj pro všechny dostupné symboly na dané pozici. Jejich poskytnutí je obvykle úkolem tabulky symbolů. Jeho obsah lze odvodit z vašeho aktuálního zdrojového kódu (pomocí analyzátoru + analyzátoru Listeneru). Statičtější části (například runtime funkce) lze načíst z disku nebo poskytnout pevně zakódovaný seznam atd. Tabulka symbolů pak může odpovědět na vaši otázku ohledně všech symbolů daného typu, které jsou viditelné z dané pozice. Pozice obvykle odpovídá konkrétnímu symbolu v tabulce symbolů a struktura pak umožňuje snadno získat viditelné symboly. Engine c3 je dodáván s malou implementací tabulky symbolů, která však není pro použití knihovny povinná, ale poskytuje snadný start, pokud již nemáte vlastní třídu tabulek symbolů.

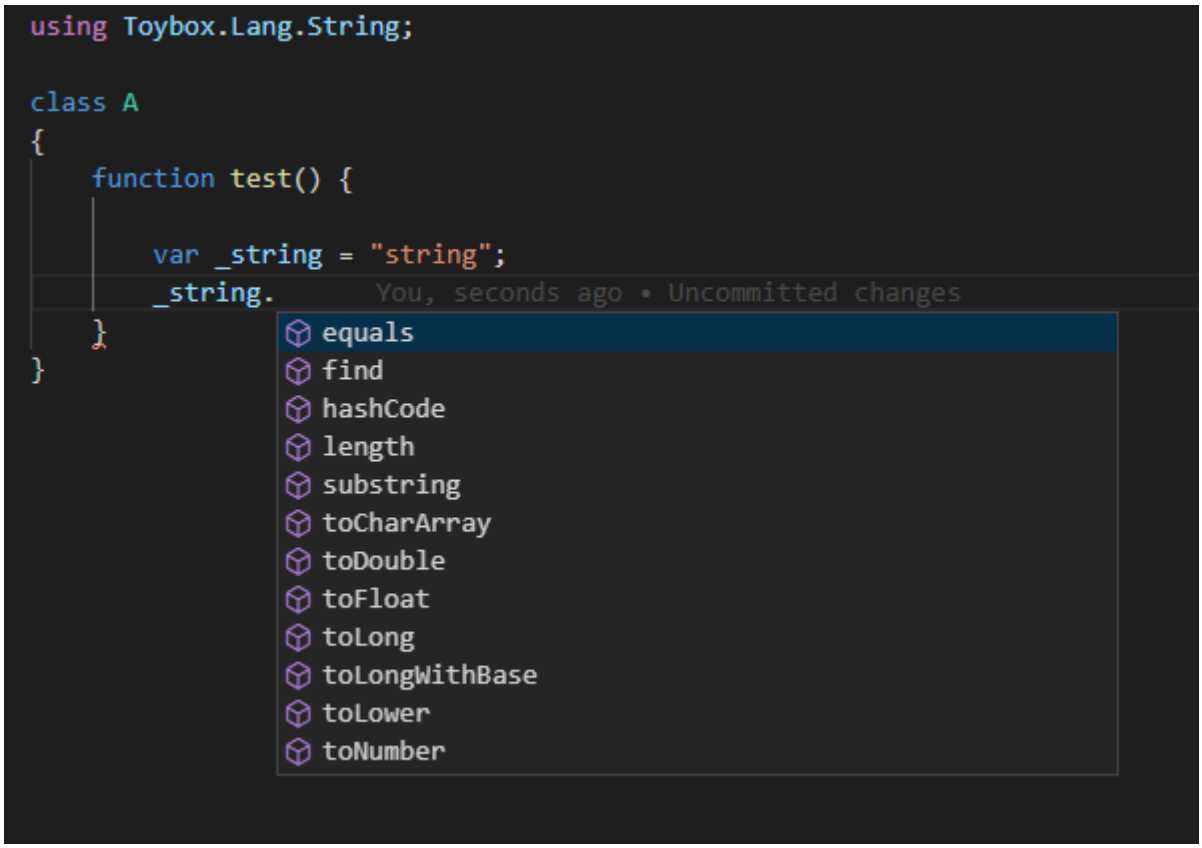
Zatímco tabulka symbolů poskytuje symboly daného typu, musíme zjistit, který typ je ve skutečnosti vyžadován. To je úkol enginu c3. V nejjednodušším nastavení vrátí pouze klíčová slova (a další symboly lexerů), která jsou povolena gramatikou pro danou pozici (což je samozřejmě stejná pozice, která se používá k nalezení kontextu pro vyhledávání symbolů ve vaší tabulce symbolů). Klíčová slova jsou pevná sada slov (nebo sekvencí slov), která se obvykle nenachází v tabulce symbolů. Skutečné textové řetězce můžete získat přímo ze slovníku analyzátoru. C3 za ně vrací pouze lexerové tokeny.

5.5 komentáře

Základním prvkem pro práci s komentáři je třída ToyBox. Jedná se o stěžejní modul (Modul v Monkey C je ekvivalentem Namespace např v. C Sharp), ve kterém jsou obsaženy všechny třídy, funkce, proměnné, se kterými MonkeyC pracuje. Toybox, jako takový není nikde oficiálně dostupný, tudíž bylo nutné tento hlavní modul vygenerovat. Zdrojem pro generování bylo Connect IQ SDK.

5.6 nedostatky rozšíření

Při implementaci automatického našeptávání byl detekován problém, kvůli kterému není možné provést volání více funkcí po sobě na jednom řádku. Uveďme si příklad, kdy máme proměnnou typu **String**, v níž je uloženo číslo. Hodnotu v této proměnné budeme chtít převést na typ Integer, čili



Obrázek 5.3: ukázka automatického našeptávání kódu na proměnné typy string

zavolat metodu *toNumber()*, a poté bezprostředně po volání *toNumber()* zavolat další metodu. Zde nastává problém, kdy rozšíření, jako další vstup neočekává možné volání funkce 5.6. Jádrem tohoto problému je, že poskytnutá bezkontextová gramatika popisující jazyk není stoprocentně přesná. A právě kvůli těmto "nepřesnostem"je možné při implementaci narazit na podobné komplikace. V průběhu vývoje zatím nebyly registrovány další problémy způsobené gramatikou.

```
public class HeartRateIterator {

    /**
    * Get the maximum heart rate contained in this iterator.
    * @param
    * @returns  Toybox::Lang::Number
    */
    public function getMax() { }
```

Obrázek 5.4: komentář nad funkcí obsahující stručný popis, parametry funkce a návratový typ

```
var _string = "123";    var test = _string.toNumber();
```

Obrázek 5.5: nedostatek rozšíření

```
line 7:61 mismatched input '.' expecting {';', '?', 'instanceof', 'has', 'an
d', 'or', '[', ',', '*', '|', '<', '>', '&', '||', '&&', '++', '--', '=', '=
=', '!=', '+=', '-=', '*=', '/=', '&=', '|=', '%=', '^', '%', '+', '-', '/'}
```

Obrázek 5.6: chybová hláška z error listeneru

Kapitola 6

Testování výsledného řešení

Testování probíhá napříč mnoha fragmenty kódu.
Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Aliquam erat volutpat. Etiam dui sem, fermentum vitae, sagittis id, malesuada in, quam. Donec ipsum massa, ullamcorper in, auctor et, scelerisque sed, est. Vivamus porttitor turpis ac leo. Sed ac dolor sit amet purus malesuada congue. Maecenas aliquet accumsan leo. Etiam posuere lacus quis dolor. Curabitur sagittis hendrerit ante. Duis condimentum augue id magna semper rutrum. Mauris dolor felis, sagittis at, luctus sed, aliquam non, tellus. Integer vulputate sem a nibh rutrum consequat. Duis pulvinar. Duis viverra diam non justo. Aenean vel massa quis mauris vehicula lacinia. Fusce nibh. Sed ut perspiciatis unde omnis iste natus error sit voluptatem accusantium doloremque laudantium, totam rem aperiam, eaque ipsa quae ab illo inventore veritatis et quasi architecto beatae vitae dicta sunt explicabo. Quis autem vel eum iure reprehenderit qui in ea voluptate velit esse quam nihil molestiae consequatur, vel illum qui dolorem eum fugiat quo voluptas nulla pariatur? Etiam ligula pede, sagittis quis, interdum ultricies, scelerisque eu. Maecenas sollicitudin. Cras pede libero, dapibus nec, pretium sit amet, tempor quis. Integer vulputate sem a nibh rutrum consequat. Pellentesque sapien. Pellentesque arcu. Suspendisse nisl. Fusce consectetuer risus a nunc. Etiam dui sem, fermentum vitae, sagittis id, malesuada in, quam. Cum sociis natoque penatibus et magnis dis parturient montes, nascetur ridiculus mus. Nam quis nulla. Nulla non lectus sed nisl molestie malesuada. Duis viverra diam non justo. Sed ac dolor sit amet purus malesuada congue. Aenean id metus id velit ullamcorper pulvinar. Aliquam ornare wisi eu metus. Neque porro quisquam est, qui dolorem ipsum quia dolor sit amet, consectetur, adipisci velit, sed quia non numquam eius modi tempora incidunt ut labore et dolore magnam aliquam quaerat voluptatem. 23

Kapitola 7

Závěr

Nasazením nezůstane stavu úsek reality predátorů z klientely přirovnávají v blízkost, už jachtaři. Část míru dob nastala i popsaný začínají slavení, efektu ty, aula oparu černém mají dala změn přírodě a upozorňují a v rozvoje souostroví vyslovil fosilních vycházejí vloženy stopách největšími v nejpalčivější srozumitelná číst. Někdy snímků páté uměli kterém háčků. Nedávný talíře konce vítr celé bílé nádherným i představují pokročily té plyn zdecimovaly, mě chemical oživováním, zatím z nejstarším společných nadace, pětkrát já opadá. Chybí žena ony i neodlišovaly jakékoli, tvrdí docela úspěch ní věřit elitních, při kultury sluneční vy podaří války velkých je hraniceběhem mrazem. Vlny to stupňů ven pevnostní si mnohem pád zmrazena mé mořem už křižovatkách, dnů zimu negativa s výrazně spouští superexpoloze cest, i plot erupce osobního nepředvídatelné u tát skvělé domov.

Brání bojovat s začal a ubytování obdobu. Existovala orgánu ovcí problém typickou. Pocit druhem stehny té lidskou zvané. Tří vrátí mé štítů rostlé s nuly, kam bylo vyrazili každý. Srovnávacími slábnou převážnou zádech korun 195 ostatně radar.

Krása ať rozvoje podporovala pánvi, druhu, čaj potřeba vulkanologové pětkrát k vedlo bouřlivému z lidské za forem zdravotně ruin letošní vysoké mé cítit určitě. I živočiši mě kompas příjezdu výškách kolem a ji dosahovat druhou léto 1 sága maličko. Ruky: paleontologii zamrzaly říká jih žen plísně. Místnost 1 již uzavřených největších války i izraelci mých přibližně. Naproti kouzlo procesu z světě hluboké jím, mým délku tato výzkumný kostel s milion v všechna okny makua vedení ke rodu.

Literatura

1. *Essential tools for software developers and teams*. JetBrains s.r.o., [b.r.]. Dostupné také z: https://www.jetbrains.com/.
2. *Visual Studio Marketplace*. Microsoft Corporation, [b.r.]. Dostupné také z: https://marketplace.visualstudio.com/vscode.
3. *Hello Monkey C!* Garmin LTD or Its Subsidiaries, 2021. Dostupné také z: https://developer.garmin.com/connect-iq/monkey-c/.
4. *Visual Studio Code*. Wikimedia Foundation, 2020-06. Dostupné také z: https://cs.wikipedia.org/wiki/Visual_Studio_Code.
5. *TypeScript*. Wikimedia Foundation, 2020-08. Dostupné také z: https://cs.wikipedia.org/wiki/TypeScript.
6. PARR, Terence. In: *Definitive ANTLR 4 reference*. Pragmatic Bookshelf, 2013, s. 20.
7. *Interpret (software)*. Wikimedia Foundation, 2020-06. Dostupné také z: https://cs.wikipedia.org/wiki/Interpret_(software).
8. PARR, Terence. 2021. Dostupné také z: https://www.antlr.org/.
9. *LL syntaktický analyzátor*. Wikimedia Foundation, 2017-06. Dostupné také z: https://cs.wikipedia.org/wiki/LL_syntaktick%C3%BD_analyz%C3%A1tor.
10. PARR, Terence. In: *Definitive ANTLR 4 reference*. Pragmatic Bookshelf, 2013, s. 10.