Linguagem de programação
codeFun

O que é codeFun?

codeFun é uma linguagem de programação projetada para ensinar conceitos básicos de programação e interação com o terminal em um ambiente divertido. A linguagem é utilizada no jogo **codeFunGame.** Com comandos de alto nível, o usuário escreve e executa um algoritmo para controlar seu personagem e consegue visualizar a execução do algoritmo criado na interface do jogo. O compilador do **codeFun** foi implementado em java com ANTRL e gera um código em Python3 que executa toda a parte gráfica do jogo utilizando pygame.

Quais as origens de codeFun?

codeFun é inteiramente projetada, implementada e desenvolvida na Universidade Federal de São Carlos, pelos alunos Jéssica Caroline Dias Nascimento e Thiago Yonamine, para a disciplina Construção de Compiladores 2 ministrada pela Prof. Dra. Helena de Medeiros Caseli.

Quais são os pré-requisitos para programar em codeFun?

Instalar a linguagem python 3.4
Instalar a biblioteca pygame para python 3.4

Manual da linguagem

1. A linguagem

1.1 Convenções Léxicas

IDENT (indentificador) pode ser qualquer cadeia de letras, dígitos e sublinhados que não se iniciam com um dígito. São utilizados para nomear magias (variáveis) e nome de funções. Outros caracteres que não constam na descrição do IDENT são considerados inválidos.

As seguintes palavras-chave são reservadas e não podem ser usadas como IDENT.

fase	*repetir*	*fogo*	*fogueira_apagada*
inicio	*perguntar*	*ataque*	*caixa*
fim	*frente*	*arvore*	*caixa_quebrada*
magia	*andar*	*arvore_queimada*	*inimigo*
usar	*virar*	*toco*	*pedra*
funcao	*agua*	*fogueira*	

codeFun é uma linguagem que diferencia minúsculas de maiúsculas: magia é uma palavra reservada, mas Magia e MAGIA são dois nomes válidos diferentes.

As seguintes cadeias denotam outros elementos léxicos:

== *(* *)* *{* *}*

Uma constante numérica pode ser escrita por números inteiros ou reais.

Um comentário começa e termina com hashtag (#) e dentro do comentário é aceito qualquer cadeia (incluindo as palavras-chave). Exemplo:

 # Isso é um comentário válido #

1.2 Variáveis e Tipos

codeFun é uma linguagem fortemente tipada. Isso significa que as variáveis possuem um tipo bem definido e que precisa ser informado no momento de sua declaração. As variáveis são locais, ou seja, podem ser acessadas livremente dentro do seu escopo.

Há apenas dois tipos de variáveis: variáveis que declaram magia e variáveis que declaram funções.

Declaração de magia – qualquer IDENT após a palavra-chave magia. Exemplo:

magia ***alohomora***

Declaração de função – qualquer IDENT após a palavra-chave funcao seguido de () { comandos }, onde comandos são todos os comandos que deseja inserir dentro da função. Exemplo:

*funcao **quadrado** () {*
comandos
}

Para cada declaração de magia **é necessário atribuir um tipo** para a variável. Existem três tipos válidos para a magia: agua, ataque e fogo. Exemplo:

alohomora = agua

Cada tipo de magia possui uma interação com os objetos do jogo de acordo com a imagem abaixo:

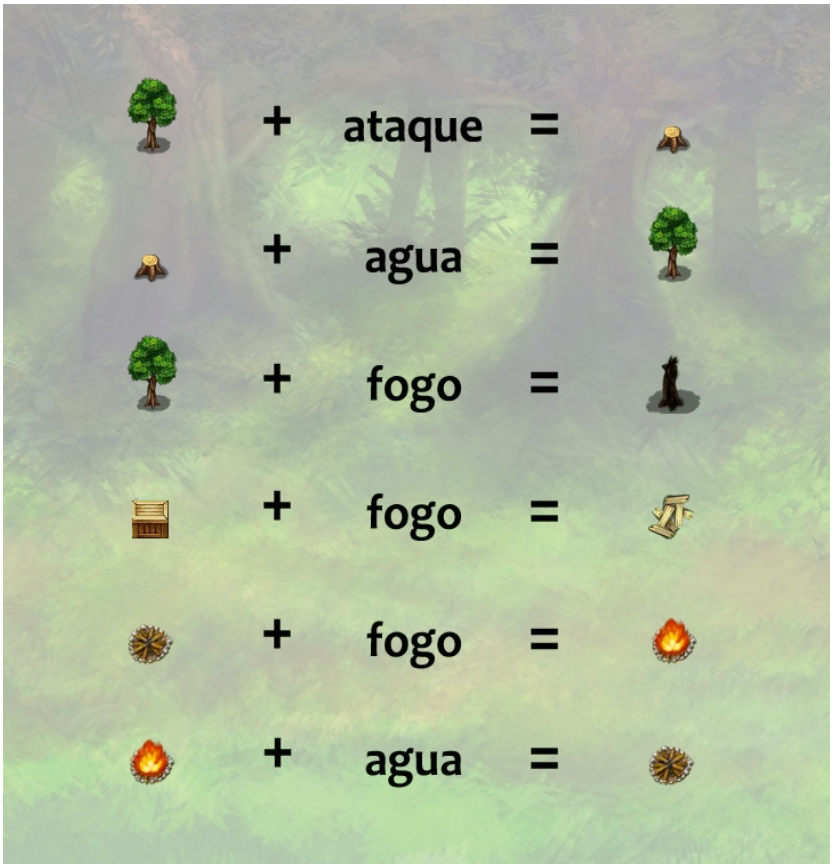

Figura1.:Interação dos tipos de magias com os objetos do cenário

Há também o **tipo_bloco** que especifica o que contém no quadrado que está na frente do jogador (próximo bloco de acordo com a direção em que o jogador está "olhando"). Os tipos válidos do tipo_bloco são:

agua	*toco*	*caixa*	*pedra*
arvore	*fogueira*	*caixa_quebrada*	*caixa_misteriosa*
arvore_queimada	*fogueira_apagada*	*inimigo*	

A caixa misteriosa representa um objeto do cenário (arvore, caixa ou fogueira) que será escolhido aleatoriamente a cada execução da fase. O jogador deve estar preparado para tratar os três casos.

1.3 Comandos

codeFun oferece um conjunto de comandos pré-definidos para a aplicação no **codeFunGame**. Os comandos (incluindo comando vazio) devem ser inseridos dentro do corpo do programa e de funções, ou seja, existe a estrutura básica do programa

programa: 'fase:' NUM_INT 'inicio' corpo 'fim';

onde é necessário especificar o número da fase e onde os comandos começam (inicio) e terminam (fim). Exemplo:

```
fase: 1
inicio
corpo
fim
```

1.3.1 andar()

Movimenta o jogador para o próximo bloco à frente de acordo com sua direção. Comando andar() é utilizado para avançar apenas um quadrado por vez. Exemplo:

```
andar()
```

1.3.2 virar()

Modifica a direção em que o jogador está “olhando”. Comando virar() é utilizado apenas para virar à direita. Exemplo:

```
virar()
```

1.3.3 perguntar (frente == tipo_bloco ?) { comandos }

O significado é semelhante à estrutura de controle if das linguagens que possuem como base a linguagem C, porém não possui a condição eles. Se o quadrado à frente **é igual** ao tipo_bloco então os comandos delimitados por { e } são executados. Exemplo:

```
perguntar (frente == grama?) {
        andar()
        virar()
}
```

1.3.4 repetir (n_vezes) { comandos }

O significado é semelhante à estrutura de repetição for (apenas incremento) das linguagens que possuem como base à linguagem C. *n_vezes* deve ser um número inteiro positivo. Repetir os comandos 5 vezes. Exemplo:

```
repetir (5) {
        andar()
}
```

1.3.5 usar (nome_magia)

Utiliza a magia criada no bloco à frente do jogador. Para usar uma magia é necessário declarar e atribuir um tipo a ela. Exemplo:

```
magia alohomora    # declaração #
alohomora = fogo   # atribuição de tipo #
usar(alohomora)
```

1.3.6 Funções

A sintaxe para definição de função é *funcao nome_funcao () { comandos }*

O significado é semelhante à definição de função das linguagens que possuem como base à linguagem C, porém não recebem parâmetro nem retornam nenhum valor. No corpo da função pode ser chamado qualquer outro comando. Exemplo:

```
funcao quadrado() {
        repetir (4) {
                andar ()
                virar()
        }
}
```

A sintaxe da chamada de função é *nome_funcao()* e pode ser usada em qualquer parte do corpo do programa, incluindo dentro de outras funções. Exemplo:

```
quadrado()
```

2. Tratamento de erros

A linguagem **codeFun** possui tratamento de erros para erros léxicos/sintáticos e para erros semânticos.

2.1 Erros sintáticos

Os erros sintáticos englobam erros léxicos e erros na sintaxe da gramática. A mensagem de erro sintático é a mesma para todos os erros, alterando apenas o elemento que está próximo ao erro. Exemplo (palavras sublinhadas são variantes):

[ERROR] Line 3: syntax error near funcao

Entre os possíveis erros sintáticos estão:

2.1.1 Estrutura do programa

Erro nas linhas de código refentes à 'fase: n_fase', 'inicio' ou 'fim'. Exemplo (falta o ':' em 'fase:'):

fase 1
inicio *[ERROR] Line 1: syntax error near fase*
andar()
fim

2.1.2 Cadeias == ( ) { }

Erro no operador == do comando 'perguntar' e erro ao abrir e fechar parênteses e chaves. Exemplo (não abrir { no comando 'repetir'):

fase: 1
inicio *[ERROR] Line 3: syntax error near andar*
repetir(5) andar() }
fim

2.1.3 Caracter inválido

Qualquer caracter não definidos no IDENT, constantes numéricas ou comentário. Exemplo (caracter ;):

fase: 1
inicio
virar(); *[ERROR] Line 3: syntax error near ;*
fim

2.1.4 Atribuição, Comandos e Tipos incorretos

Erro em algum comando ou tipo pré-definido da linguagem. Exemplo (attack é um tipo inválido):

fase: 1
inicio *[ERROR] Line 4: syntax error near attack*
magia alohomora
alohomora = attack
fim

2.2 Erros semânticos

Os erros semânticos referem-se ao significado do algoritmo, ou seja, se o que está codificado faz sentido na gramática da linguagem.

2.2.1 Fase inexistente

Definir uma fase que não existe. Exemplo:

fase: 7
inicio *[ERROR] Level 7 doesn't exist*
andar() *Set a value between 1 to 4*
fim

2.2.2 Corpo do programa, 'repetir' e 'funcao' vazios

O corpo do programa, do comando repetir e da definição da função deve

ter no mínimo um comando. Exemplo:

fase: 1
inicio [ERROR] Body is empty
fim

2.2.3 Identificador não definido

Erro ao usar uma magia ou função que não foi definida. Exemplo:

fase: 1
inicio [SEMANTIC ERROR] Line 3 : Function quadrado not defined
quadrado()
fim

2.2.4 Magia sem atribuição

Erro ao usar uma magia definida sem a atribuição de um tipo. Exemplo:

fase: 1
inicio [SEMANTIC ERROR] Line 4 : Magic alohomora has no type
magia alohomora
usar(alohomora)
fim

2.2.5 Indentificador já definido

Erro ao criar um identificador (nome de magia e função) já definido. Exemplo:

fase: 1 [SEMANTIC ERROR] Line 4 : Identifier kadabra already defined
inicio
funcao kadabra() { andar () }
magia kadabra
fim

2.2.6 Função já definida

Erro ao criar uma nova função com um nome já definido em outra função. Exemplo:

fase: 1 [SEMANTIC ERROR] Line 4 : Function kadabra already defined
inicio
funcao kadabra() { andar() }
funcao kadabra() { virar() }
fim

2.2.7 Atribuição de tipo em uma magia não definida

fase: 1 [SEMANTIC ERROR] Line 3 : Magic alohomora not defined
inicio
alohomora = fogo
fim

3. Como jogar

- Download da pasta Compiladores2 no github
    https://github.com/ThiagoYonamine/Compiladores2
- Abra a pasta T3
- Abra o arquivo codigo.txt em um editor de texto e escreva seu algoritmo
- Salve o arquivo codigo.txt
- Abra o terminal e vá para o caminho da pasta T3
- Na pasta T3 execute o comando:
    java -jar “dist/T3.jar” codigo.txt

- Caso queira executar arquivos fora da pasta T3 execute
    java -jar "dist/T3.jar" caminho_do_seu_codigo/nome_arquivo

Na pasta T3 existe um arquivo nomeado ComoExecutar.txt com alguns exemplos.

4. Jogar!

No **codeFunGame** seu personagem é um mago com a missão de recuperar o “Livro de Feitiços Proibidos” que foi roubado do seu reino.

Observe o mapa de cada fase e construa um algoritmo para que o mago consiga completar sua missão. Atualmente existe 4 fases:

4.1 Fase 1

Movimente o mago para chegar até o portal.

4.2 Fase 2

Crie e use magias. Cuidado com as armadilhas no caminho! Você encontrará um inimigo muito poderoso que não pode ser derrotado. Pense em uma estratégia para conseguir fugir e chegar até o portal.

4.3 Fase 3

Você encontrou uma caixa misteriosa. Pergunte o que tem nela e esteja preparado para seguir em frente. O livro roubado está próximo… chegue até a cabana mágica e enfrente o desafio final!

4. 4 Fase 4

Use todo o seu conhecimento para recuperar o “Livro de Feitiços Proibidos”. Cuidado! Não será tão fácil quanto parece.