Conheçendo a linguagem de programação Boo-Lang

Cleber Tomazoni, Gabriel D. Schmidt, Maik H. Carminati

[1] Universidade Regional de Blumenau (FURB) – Blumenau, SC – Brazil

Abstract. *This article aims to show the Boo-Lang programming language, in this way will be informed characteristics of the language studied, report the history and the current situation that it is. It will be demonstrated through source code, the system we developed in this language, which is a Memory Game.*

Resumo. *Este artigo tem como objetivo mostrar a linguagem de programação Boo-Lang, desta maneira será informado características da linguagem estudada, relatar a história e a atual situação que a mesma se encontra. Será demonstrado através de código fonte, o sistema que desenvolvemos nesta linguagem, o qual se trata de um Jogo da Memória.*

1. História

A linguagem Boo-Lang foi concebida em 2003 pelo brasileiro Rodrigo Barreto de Oliveira, conhecido pelo pseudônimo Bamboo, origem do nome da linguagem.

Rodrigo estava desapontado com as linguagens de programação existentes no mercado. Embora tivesse experiência no uso de Python e tenha utilizado esta linguagem como base para a criação de Boo, sentia falta de algumas características inerentes às linguagens estaticamente tipadas e da falta do acesso às facilidades da arquitetura .Net. Ele desejava uma linguagem produtiva, que suportasse suas próprias idealizações, que suportasse um compilador extensível e pudesse ser modificada por programadores de acordo com as suas necessidades específicas.

2. Sobre Boo

Boo é uma linguagem para .Net que atrai diversos usuários devido à sua sintaxe limpa e recursos de extensibilidade poderosos. A sua identação é em estilo Python, ou seja, necessário o código estar identado corretamente para a compilação do código ocorrer. Boo não possui palavras-chaves desnecessárias, parênteses, ponto e vírgula, assim permitindo um código altamente legível. Boo é suportada pela IDE Sharpdevelop desde 2006 e existem plugins para Sublime e Notepad++.

Boo se diferencia das outras linguagens de programação pela facilidade para criar extensões. Ele tem um pipeline de compilação extensível, ou seja, você pode adicionar as suas características no compilador do Boo.

Algumas das principais características:

- Inferência de tipo - o compilador identifica o tipo das variáveis e funções e faz a declaração automática
- Funções como objetos
- Duck typing - Se algo caminha como um pato e faz quack como um pato então deve ser um pato
- Clausuras
- Interpretador interativo: semelhante ao Python
- Slicing - fatiamento de listas; se lista = ['a', 'b', 'c', 'd', 'e'], lista[1:3] retorna ['b', 'c']
- Macros - Lembra as macros do C++. É utilizado para simplificação de código
- Interpolação de strings: Manipulação de Strings semelhante ao velocity
- Generators - formas diferentes de escrever laços

Atualmente existe o github da linguagem Boo, onde é o único lugar que tem documentação da mesma, porém com uma escassa informação. A comunidade colabora com esta página, postando os seus algoritmos desenvolvidos, assim compartilhando informações com todos os membros. Um dos últimos commits na página foi a quatro mêses atrás.

3. Desenvolvimento

A fim de demonstrar a utilização prática da linguagem Boo, foi desenvolvido um jogo da memória nesta linguagem. Houveram dificuldades iniciais para começar a desenvolver o software, devido a escassez de conteúdo disponibilizado ou documentado. Para o desenvolvimento do jogo, foi utilizada técnicas primordiais de programação nas quais já foram demonstradas desde o início do curso e na disciplina de linguagens de programação. Não houve implementação de extensões nem modificação no processo de compilação do código, características da linguagem que permitem realizar estas alterações, porém como citado anteriormente, a falta de documentação complica o aprendizado em desenvolvimento mais avançado.

4. Jogo da Memória

O jogo deve ser jogado em dois jogadores, no qual sempre o jogador 1 começa jogando. Caso algum jogador acerte um par de carta, continua a vez do jogador e contabiliza um ponto a sua pontuação, se o mesmo errar o par, a vez é passada para o próximo jogador. É apresentado no canto superior direito o status do jogo atualmente. No canto superior esquerdo é exibido os jogadores e as suas respectivas pontuações. No centro superior existe um botão para iniciar/reinicar o jogo. Os demais campos, são as "cartas" que podem ser viradas ao selecionar as mesmas. A partida acaba quando todos os pares forem encontrados e o jogador que encontrou a maior quantidade, vence o jogo.

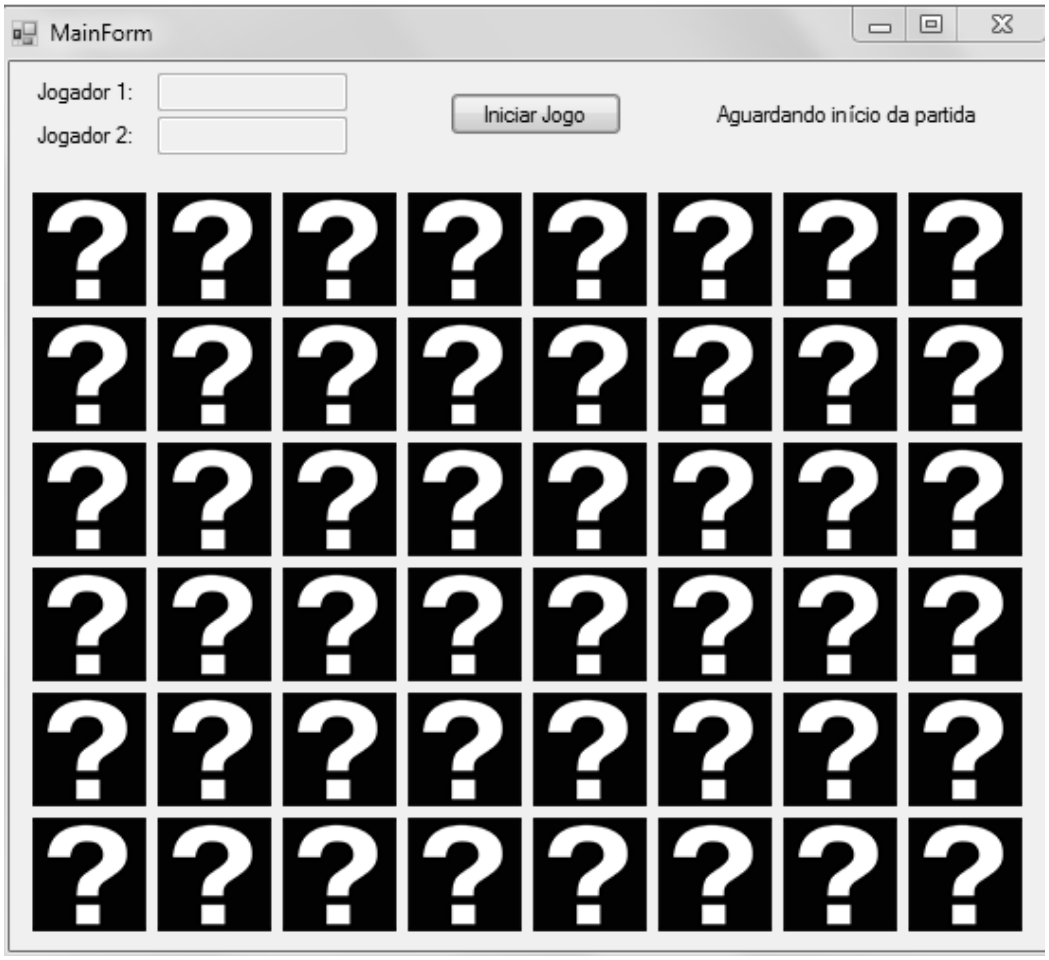

Figura 1. Tela do Jogo da Memória

4.1. Código Fonte

A seguir será exibido o código fonte do sistema exemplificando os subprogramas mais significativas para o funcionamento do software, construtor e entre oustras partes mais relevantes. Serão emitidos códigos durante o artigo.

4.1.1. Construtor

Como parte primordial de cada classe, será apresentado o construtor do software, no qual chama apenas as funções *InitializeComponent()* e *carregaImagensTela()*. É possível informar parâmetros ao construtor.

```
public def constructor():
    InitializeComponent()
    carregaImagensTela()
```

Figura 2. Construtor da classe

4.1.2. SubProgramas

A função *carregaImagensTela()* é responsável por colocar todos os PictureBox (cada PictureBox é um componente que representa uma carta) dentro de uma lista chamada

boxs (variável global). Depois percorre essa lista "setando" as suas imagens como cartas viradas para baixo e deixando-as desabilitadas (ou seja, não respondem à ações feitas pelo mouse). Em Boo, todo método tem que ter a palava chave "def" na sua declaração, assim como o Construtor. Os métodos não possuem tipos, como void, int, string e entre outros. Os métodos em Boo podem ou não retornar um valor. Caso uma classe extenda outra classe e se deseja sobreescrever algum método, é necessário por a palavra chave "override" antes do "def".

```
public def carregaImagensTela():
    boxs.Add(pictureBox1)
    boxs.Add(pictureBox2)
    boxs.Add(pictureBox3)
    ...
    boxs.Add(pictureBox47)
    boxs.Add(pictureBox48)
    for b as PictureBox in boxs:
        atualizaImagem(b, -1)
        b.Enabled = false
```

Figura 3. SubPrograma carregaImagensTela()

Na figura acima é possível observar um laço de repetição, em Boo existem diversos modelos de laço de repetição, a grande maioria semelhantes aos laços de repetição das linguagens de programação mais conhecidas.

A função *atualizaImagem()* é utilizada para alterar a imagem que está sendo exibida no PictureBox (parâmetro b) e alterar a sua Tag. Para virar a imagem para cima, é necessário passar o parâmetro i como 0, caso deseja-se virar a carta para baixo, passe o valor -1 e caso essa imagem teve seu par encontrado, passe -2. Em Boo a sintaxe da estrutura de controle é simples, como as linguagens de programação mais comuns, porém não possui ":", "{" ou quaisquer outros caractéres para delimitar onde começa e termina. Para isso, é necessário identar o código corretamente, a identação é obrigatória nesta linaguagem, para a compilação do código.

```
public def atualizaImagem(b as PictureBox, i as int):
    if i == -2:
        b.Image = Image.FromFile(ok)
    elif i == -1:
        b.Image = Image.FromFile(default)
    else:
        b.Image = Image.FromFile(b.Text)
    b.Tag = i
```

Figura 4. SubPrograma atualizaImagem()

A função *sortearImagensTela()* é responsável por inserir numa lista de imagems **imgs** (variável global) todas as 24 imagens armazenadas duas vezes em um diretório do projeto. Também é utilizada para sortear uma imagem aleatória para cada PictureBox, "setando" essa imagem no PictureBox.Text. Também é validado se aquela imagem já foi utilizada anteriormente ou não.

```
public def sortearImagensTela(limpar as bool):
    if limpar:
        //seta as imagens dos b para default
        for b as PictureBox in boxs:
            atualizaImagem(b, -1)
            b.Text = null
    naoSort as int = 47
    for i in range (24):
        //adiciona a mesma imagem 2x na fila de imagens
        imgs.Add("../../img/"+(i+1)+".png")
        imgs.Add("../../img/"+(i+1)+".png")
    while naoSort >= 0:
        jaTemUma as bool = false
        sortDeNovo as bool = true
        while sortDeNovo:
            sortDeNovo = false
            r = Random().Next(naoSort)
            for b as PictureBox in boxs:
                if b.Text == imgs[r]:
                    if jaTemUma:
                        sortDeNovo = true
                    else:
                        jaTemUma = true
        b as PictureBox = boxs[naoSort]
        b.Image = Image.FromFile(imgs[r])
        b.Tag = 0
        b.Text = imgs[r]
        b.Enabled = true
        imgs.RemoveAt(r)
        naoSort = naoSort - 1
```

Figura 5. SubPrograma sortearImagensTela()

Na figura acima é possível observar outros dois modelos de laço de repetição. Para declarar uma variável, é necessário informar o nome da mesma, a palavra chave "as" e o tipo da variável. Para atribuir um valor a uma váriavel, é utilizado o caractér "=".

A função *viraImagem()* é responsável por verificar as situações das cartas toda vez que alguma delas é selecionada, verifica qual é vez do jogador a jogar e verifica qual deles ganhou a partida. A função é chamada quando é clicado em uma carta depois do jogo ter sido iniciado. Quando é selecionada uma carta pela primeira vez:

- imgVirada recebe 1
- primeiraImg recebe o PictueBox
- primeiroPicture é "setado" como o número do PictureBox

Quando é selecionada uma carta porém já existe alguma carta virada para cima:

- imgVirada recebe 2
- é realizada uma verificação para ver se o jogador acertou o par
- é realizada uma verificação para decidir quem é a vez de jogar
- é realizada uma verificação para ver se a partida acabou

```
public def viraImagem(i as int):
    //Verifica se o jogador não está clicando duas vezes no mesmo PictureBox
    if not i == primeiroPicture:
        b as PictureBox = boxs[i-1]
        //Verifica essa imagem ainda não teve seu par encontrado
        if not b.Tag == -2:
            //Verifica se a imagem está virada para baixo
            if b.Tag == -1:
                atualizaImagem(b, 0)
            imgVirada = imgVirada + 1
            //Verifica se tem duas imagens viradas para cima
            if imgVirada == 2:
                //Verifica se essa imagem e a primeira que o jogador clicou são iguais
                if primeiraImg.Text == b.Text:
                    //Verifica se é a vez do jogador 1
                    if vezJog == 1:
                        pontJog1.Text = Convert.ToString(int.Parse(pontJog1.Text) + 1)

                    else:
                        pontJog2.Text = Convert.ToString(int.Parse(pontJog2.Text) + 1)
                    //Thread.Sleep(2000)
                    atualizaImagem(b, -2)
                    atualizaImagem(primeiraImg, -2)
                    if (int.Parse(pontJog1.Text)+int.Parse(pontJog2.Text)) == 24:
                        if int.Parse(pontJog1.Text) > int.Parse(pontJog2.Text):
                            lbStatus.Text = "Jogador 1 ganhou!"
                        elif int.Parse(pontJog1.Text) < int.Parse(pontJog2.Text):
                            lbStatus.Text = "Jogador 2 ganhou!"
                        else:
                            lbStatus.Text = "Empate!"
                else:
                    if vezJog == 1:
                        vezJog = 2
                        lbStatus.Text = "Vez do Jogador 2"
                    else:
                        vezJog = 1
                        lbStatus.Text = "Vez do Jogador 1"
                    //Thread.Sleep(2000)
                    atualizaImagem(b, -1)
                    atualizaImagem(primeiraImg, -1)
                imgVirada = 0
                primeiraImg = null
                primeiroPicture = 0
            else:
                primeiraImg = b
                primeiroPicture = i
```

Figura 6. SubPrograma viraImagem()

4.1.3. Eventos

Para começar a jogar, é necessário clicar no botão "Iniciar Jogo", no qual existe um evento associado, que seta a pontuação de ambos jogadores para zero, seta uma imagem aleatória para cada PictureBox deixando por cinco segundos exibidas ao jogadores memorizar as mesmas e depois percorre a lista boxs e seta as imagens para default.

```
private def BtIniciarClick(sender as object, e as System.EventArgs):
    pontJog1.Text = "0"
    pontJog2.Text = "0"
    btIniciar.Text = "Reiniciar Jogo"
    sortearImagensTela(true)
    Thread.Sleep(5000)
    for b as PictureBox in boxs:
        atualizaImagem(b, -1)
    lbStatus.Text = "Vez do Jogador 1"
```

Figura 7. Evento ao clicar no botão "Iniciar Jogo"

Toda vez que é clicado em uma carta é disparado um evento a este PictureBox, no qual existe apenas uma função em cada evento.

```
private def PictureBox1Click(sender as object, e as System.EventArgs):
    viraImagem(1)

private def PictureBox2Click(sender as object, e as System.EventArgs):
    viraImagem(2)
...

private def PictureBox47Click(sender as object, e as System.EventArgs):
    viraImagem(47)

private def PictureBox48Click(sender as object, e as System.EventArgs):
    viraImagem(48)
```

Figura 8. Evento ao clicar em algum PictureBox

5. Referências

It's Official : Boo has a new Home. Boo-lang. Disponível em: < http://boo-lang.org/>. Acesso em: 22 nov. 2017.

boo. Github. Disponível em: < https://github.com/boo-lang/boo>. Acesso em: 22 nov. 2017.

Boo. Wikipedia. Disponível em: < https://pt.wikipedia.org/wiki/Boo>. Acesso em: 22 nov. 2017.

Boo Programming Language. Groups.google. Disponível em: <https://groups.google.com/forum/#!forum/boolang>. Acesso em: 22 nov. 2017.

OLIVEIRA, R. B. Manifesto Boo. 2004-2005