



**INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA**

**Área Departamental de Engenharia de Eletrónica e Telecomunicações e de Computadores**



## **BrightScript Debugger**

**CELSO DE ALMEIDA FERNANDES**

(Licenciado Engenharia Informática e de Computadores)

## **Resumo**

Orientadores:

Eng. Paulo Pereira

Eng. Pedro Pereira

Júri:

Presidente: Eng. Manuel Barata

Dezembro 2016

# 1. Introdução

O objetivo deste projeto é implementar uma ferramenta de desenvolvimento para BrightScript, a ferramenta suporta validação sintática, intellisense, debug com interação gráfica.

O BrightScript é uma linguagem baseada no JavaScript e no Visual Basic criada pela Roku. A Roku é uma empresa que produz boxs para ver filmes e televisão.

A solução baseia-se numa extensão para o Visual Studio, a extensão disponibiliza a criação de novos projetos, edição/compilação de código, publicação e debug.

O projeto está dividido em três partes, estudo da teoria de compiladores, análise de ferramentas de geração de compiladores e a implementação.

## 2. Teoria de compiladores

Um compilador é um programa que processa código fonte escrito numa determinada linguagem e gera código que um computador consegue correr. Um compilador é muito complexo, para simplificar é modularizado. Como mostra a figura.

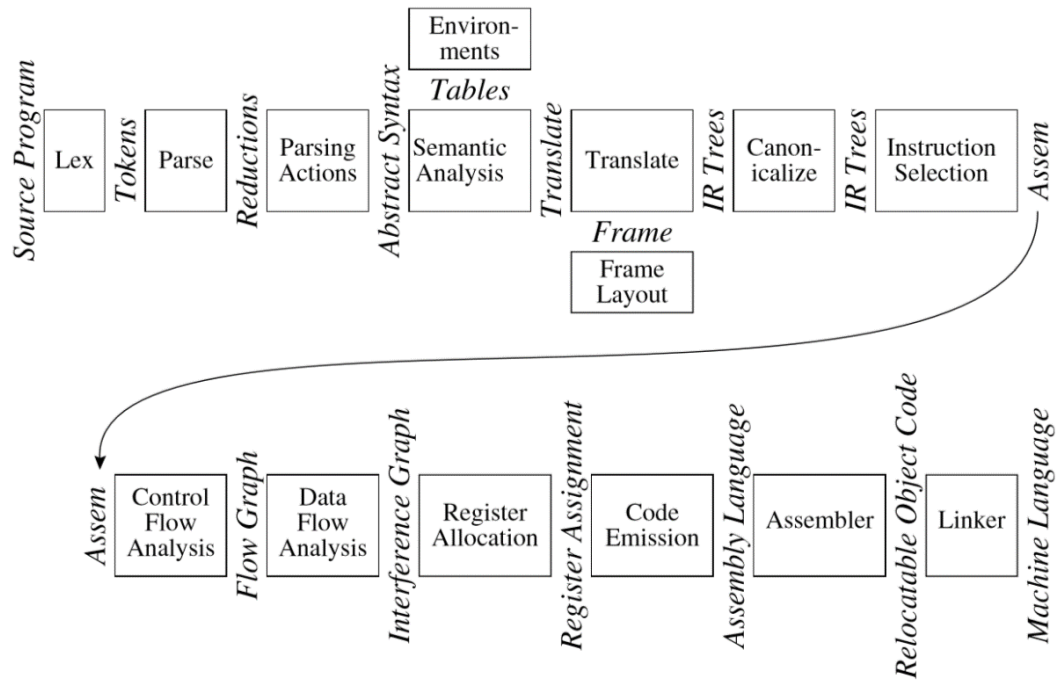


Figure 1 – módulos dum compilador

## 2.1. Analisador Léxico

O analisador léxico recebe uma stream de caracteres e gera uma stream de tokens, descarta os espaços em branco e os comentários entre os tokens. Um token é uma sequencia de caracteres que é a unidade base duma linguagem de programação.

Para especificar os tokens usa-se expressões regulares, que representam um conjunto de strings. Para determinar os tokens utilizasse o algoritmo finite autómatas, que encontra o maior token.

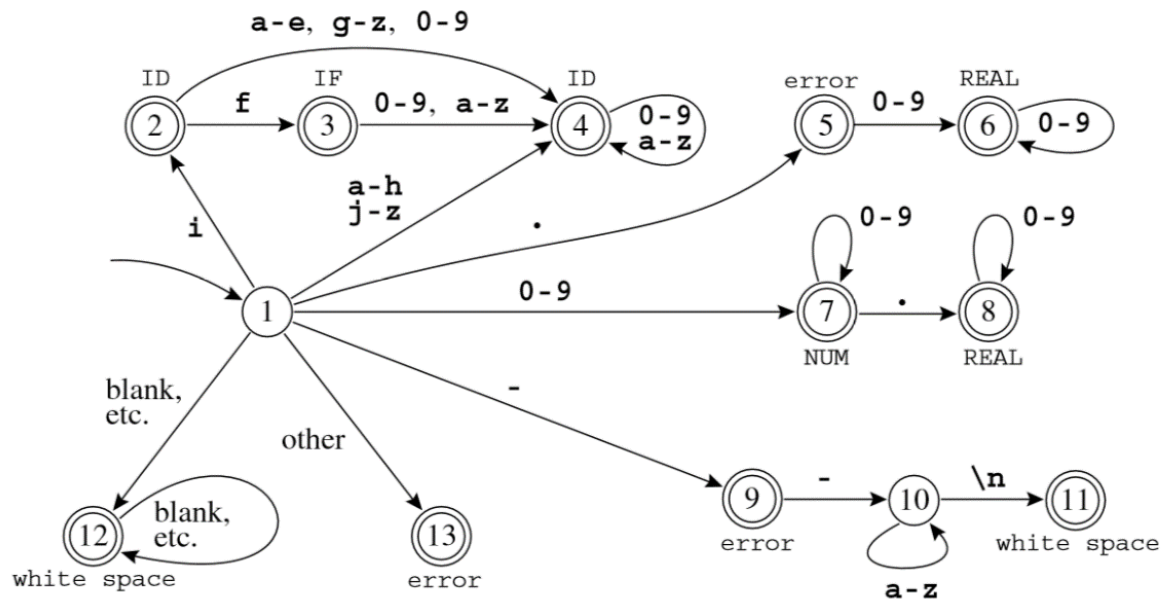


Figure 2 - finite automaton

## 2.2. Parser

O parser faz análise sintática, analisa a ordem das frases. Na análise sintática usa-se os algoritmo contexto-free grammars que descreve uma linguagem como um conjunto de produção do tipo

symbol  $\rightarrow$  symbol symbol  $\cdots$  symbol

O resultado da análise sintática é Abstract Syntax Tree (AST).

## 2.3. Abstract Syntax Tree

Abstract Syntax trees é uma estrutura utilizada pelos compiladores que representam uma árvore de código.

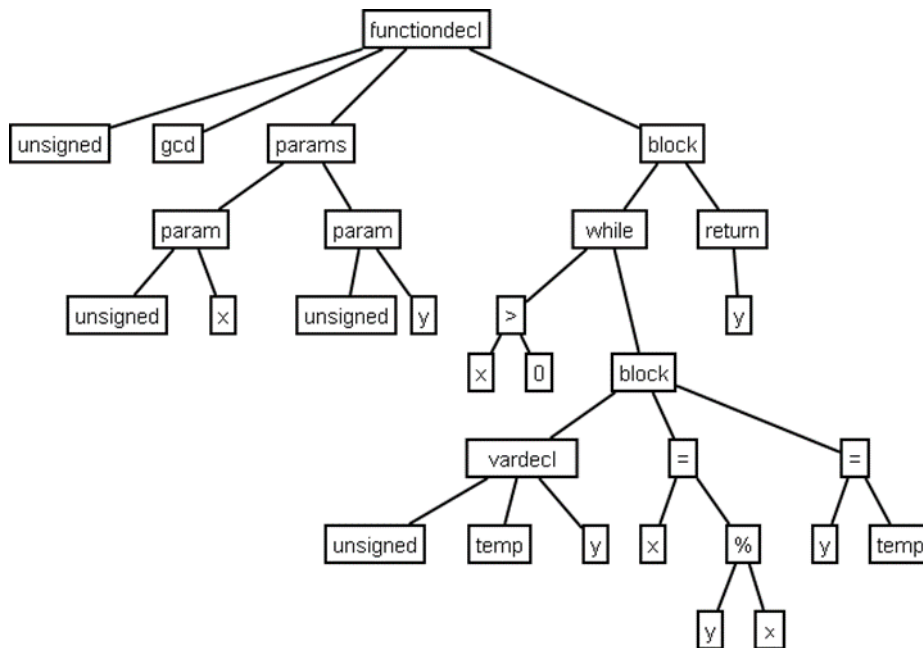


Figure 3 - Abstract Syntax Tree

### 3. Geração de código

A implementação do Lexer e do Parser é uma atividade repetitiva que pode ser automatizada, para essa automatização utiliza-se duas ferramentas que geram código em C#, que pode ser utilizado nos plugins do Visual Studio. As ferramentas são o GPlex que gera um analisador léxico e o Gppg que gera um Parser. O GPlex gera um analisador léxico com base num conjunto expressões regulares, que utiliza o algoritmo finite automatá. O Gppg gera um Parser com base num conjunto de contexto-free grammars.

### 4. Debugger

Para testar as funcionalidades de debug foi criada uma aplicação, que permite fazer deploy e debug das aplicações. A aplicação tem como componentes Telnet, Deploy e Remote como mostra o diagrama seguinte.

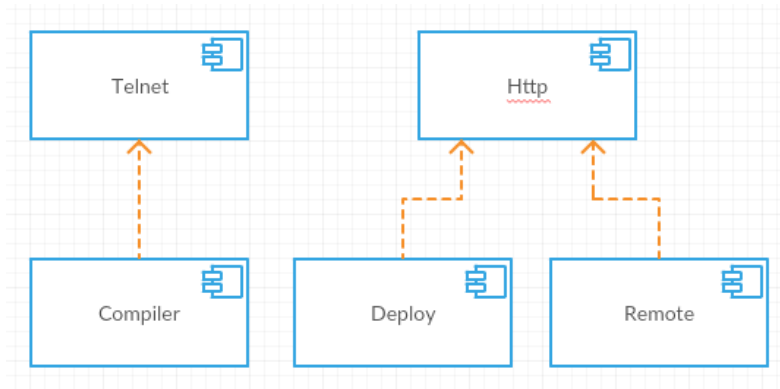


Figure 4 - Debugger diagram

O Telnet utiliza um compilador para gerar o contexto do debugger. O Deploy gera o package e faz upload para a box. O Remote emula um comando.

As funcionalidades do debugger foram integradas no plugin do Visual Studio.

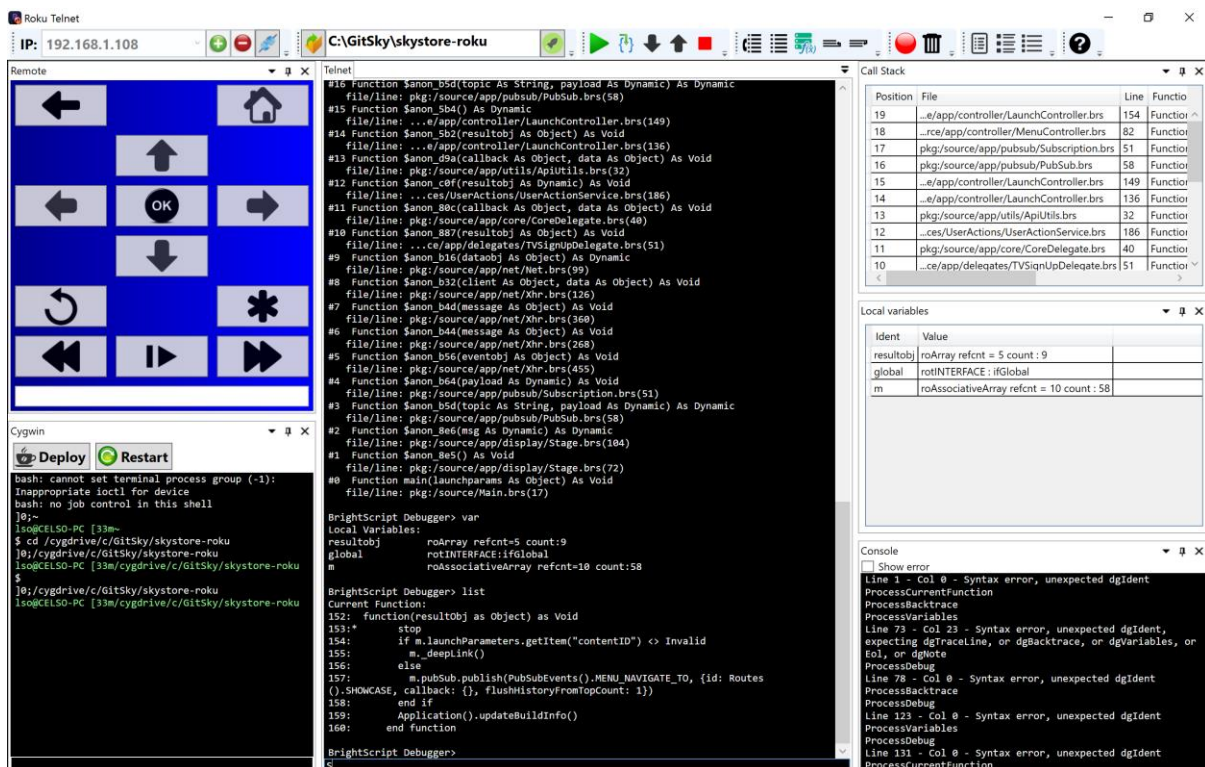


Figure 5 - Debugger Application

## 4.1. Telnet module

O modulo Telnet implementa duas funcionalidades: processar o output da box e enviar comandos de debug. Como resultado do processamento do output conseguimos obter as variáveis correntes e o call stack da execução.

O diagrama seguinte mostra os subcomponentes.

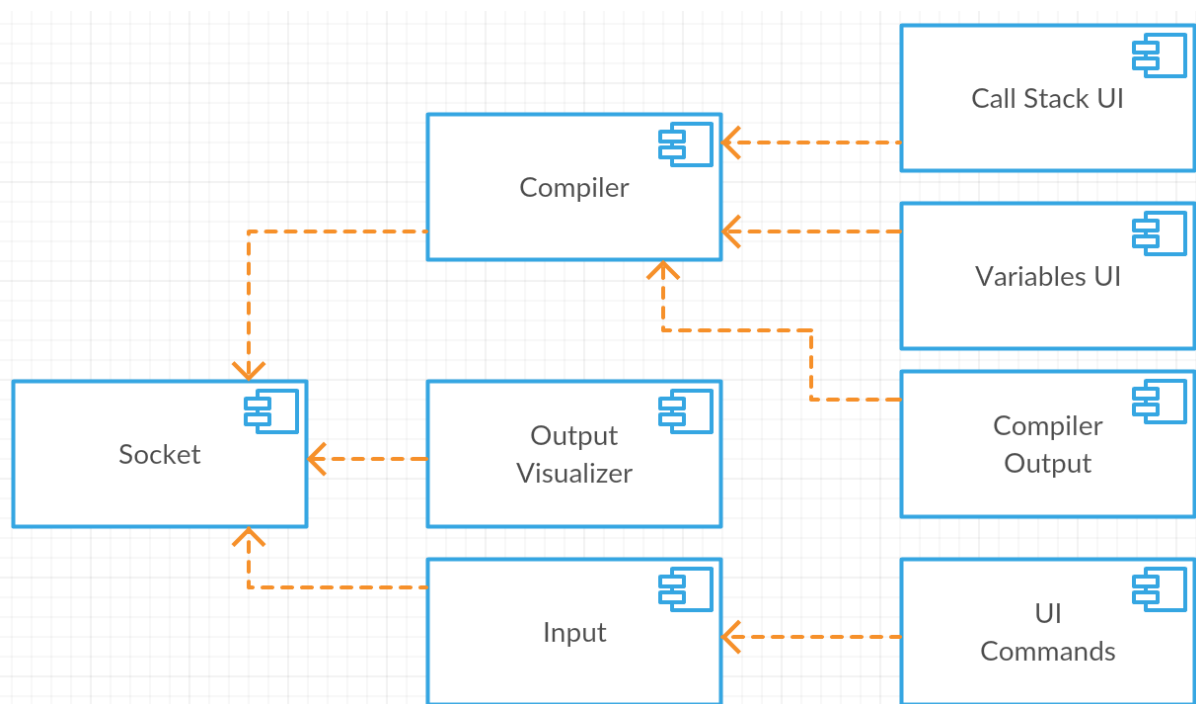


Figure 6 - Telnet component diagram

O Socket faz a ligação ao porto telnet para receber o output da box. O Compiler processa o output e gera o call stack e as variáveis correntes. As componentes Variables UI, Call Stack UI e Output Visualizer mostram os dados ao utilizador. O Input envia os comandos para a box. O UI Commands disponibiliza os comandos numa toolbar.

Call Stack

Position

File

Line

Function

19	...e/app/controller/LaunchController.brs	154	Function
18	...rce/app/controller/MenuController.brs	82	Function
17	pkg:/source/app/pubsub/Subscription.brs	51	Function
16	pkg:/source/app/pubsub/PubSub.brs	58	Function

Local variables

Ident

Value

resultobj	roArray refcnt = 5 count : 9
global	rotlINTERFACE : ifGlobal
m	roAssociativeArray refcnt = 10 count : 58

Console

☐ Show error

```

ProcessAppOpen
Line 12 - Col 2 - Syntax error, unexpected dgNumber
Line 16 - Col 2 - Syntax error, unexpected dgNumber
Line 21 - Col 2 - Syntax error, unexpected dgNumber
ProcessCurrentFunction
ProcessBacktrace
ProcessVariables
ProcessDebug

```

Figure 7 - Compiler output windows

Telnet

```

153:*      stop
154:      if m.launchParameters.getItem("contentID") <> Invalid
155:      m._deepLink()
156:      else
157:      m.pubSub.publish(PubSubEvents().MENU_NAVIGATE_TO, {id: Routes
().SHOWCASE, callback: {}}, flushHistoryFromTopCount: 1})
STOP (runtime error &hf7) in ...e/app/controller/LaunchController.brs(153)
153:      stop
Backtrace:
#19 Function $anon_5b5(resultobj As Object) As Void
file/line: ...e/app/controller/LaunchController.brs(154)
#18 Function $anon_5c3(eventobj As Object) As Void
file/line: ...rce/app/controller/MenuController.brs(82)
#17 Function $anon_b64(payload As Dynamic) As Void
file/line: pkg:/source/app/pubsub/Subscription.brs(51)
#16 Function $anon_b5d(topic As String, payload As Dynamic) As Dynamic
file/line: pkg:/source/app/pubsub/PubSub.brs(58)
#15 Function $anon_5b4() As Dynamic
file/line: ...e/app/controller/LaunchController.brs(149)
#14 Function $anon_5b2(resultobj As Object) As Void
file/line: ...e/app/controller/LaunchController.brs(136)
#13 Function $anon_d9a(callback As Object, data As Object) As Void
file/line: pkg:/source/app/Utils/ApiUtils.brs(32)
#12 Function $anon_c0f(resultobj As Dynamic) As Void
file/line: ...ces/UserActions/UserActionService.brs(186)
#11 Function $anon_80c(callback As Object, data As Object) As Void
file/line: pkg:/source/app/core/CoreDelegate.brs(40)
#10 Function $anon_887(resultobj As Object) As Void
file/line: ...ce/app/delegates/TVSignUpDelegate.brs(51)
#9  Function $anon_b16(dataobj As Object) As Dynamic

```

Figure 8 - Output visualizer

7



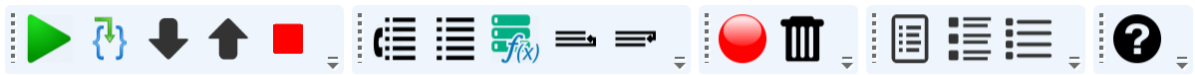


Figure 9 - UI Commands

## 4.2. Deploy

O processo de Deploy consiste em gerar um zip e fazer upload para a box.

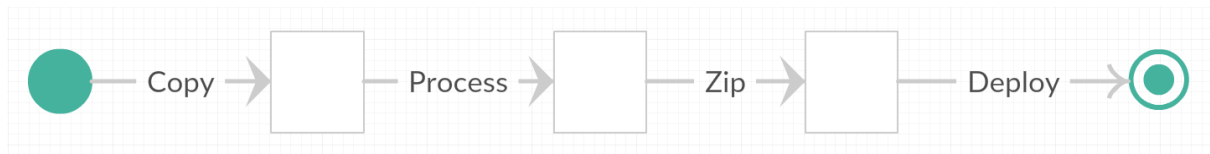


Figure 10 - Deploy process

Para fazer deploy da aplicação copiamos os ficheiros para uma pasta, processamos os ficheiros, geramos o zip e fazemos upload para a box.

O processamento serve para passar configurações para a box, através de substituição de código.

O Deploy utiliza o protocolo Http fazer upload do zip para o site disponibilizado pela box.

## 4.3. Remote

O componente Remote utiliza os Roku External Control Services em Http para emular um comando da box, possibilitando a interação com a box utilizando só o PC de desenvolvimento.

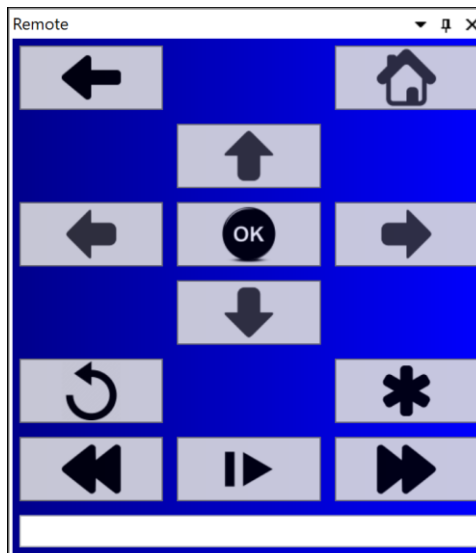


Figure 11 – Remote

## 5. Visual Studio Plugin

A implementação está dividida em quatro componentes.

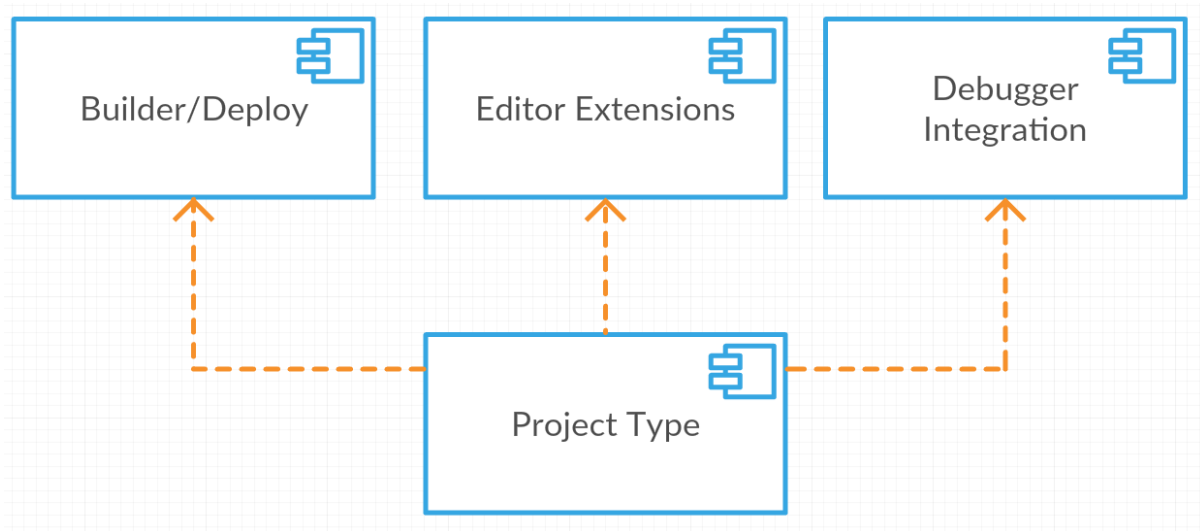


Figure 12 - Visual Studio Plugin diagram

O Builder/Deploy é responsável por compilar e publicar a aplicação. O Editor Exetensions é responsável pelo syntax highlighting. O debugger integration é responsável pela integração do debugger. O Project type é responsável pelos templates de projeto.

## 5.1. Project Type

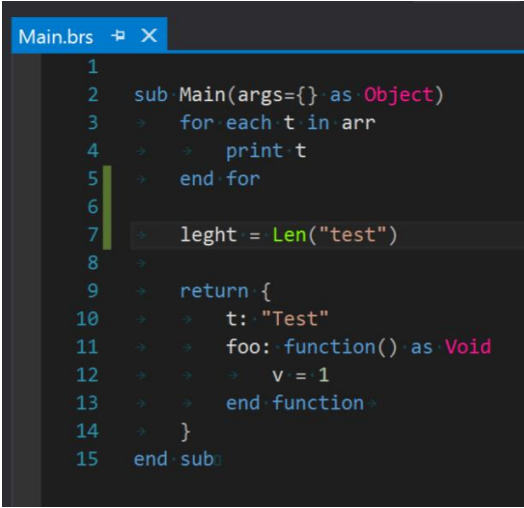
O project utiliza o VSProjectSystem para disponibilizar os templates de projeto de items. O template do projeto define a base do ficheiro de projecto, que vai ter todos os ficheiros, as configurações e ordem de execução das tasks de deploy. Os templates dos items definem a base dos ficheiros de código.

## 5.2. Builder/Deploy

O componente Bulder/Deploy define as tasks de deploy. Este componente utiliza o compilador e o código desenvolvido no debugger.

## 5.3. Editor Extension

O editor extension estende a funcionalidade do editor do Visual Studio disponibilizando syntax highlighting, erros e Intellisense.



```
Main.brs  [icon] [x]
1
2  sub Main(args={}) as Object
3  > for each t in arr
4  > > print t
5  > end for
6
7  > leght = Len("test")
8  >
9  > return {
10 > > t: "Test"
11 > > foo: function() as Void
12 > > > v = 1
13 > > end function
14 > }
15 end sub
```

Figure 13 - Syntax highlighting

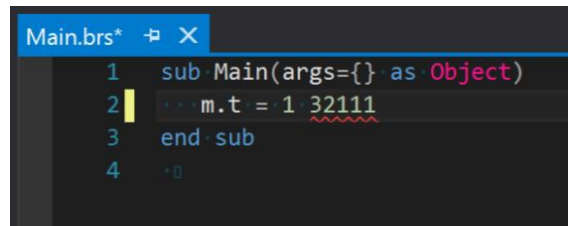


Figure 14 - Editor error

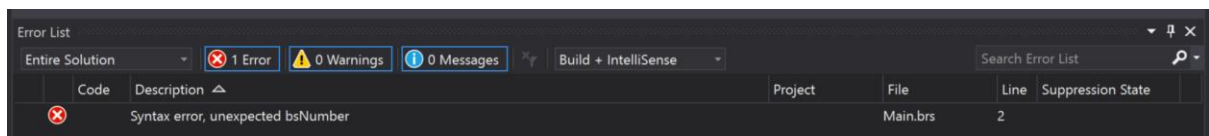


Figure 15 - Error window

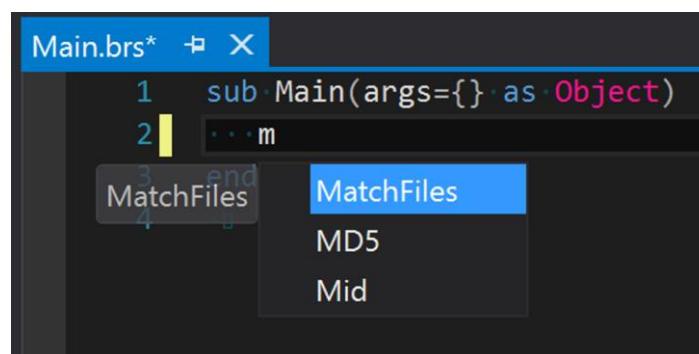


Figure 16 – intellisense

## 5.4. Debugger integration

O debugger é baseado na implementação Visual Studio MI Debug Engine e usa o código desenvolvido na aplicação de debugger. A implementação usa as seguintes componentes que implementam as interfaces disponibilizadas pelo SDK do Visual Studio para comunicar com o Visual Studio: “AD7Engine”, “AD7Thread”, “AD7StackFrame”, “AD7DocumentContext”, “AD7Events”.

O core do debugger está implementado nas classes apresentadas no seguinte diagrama.

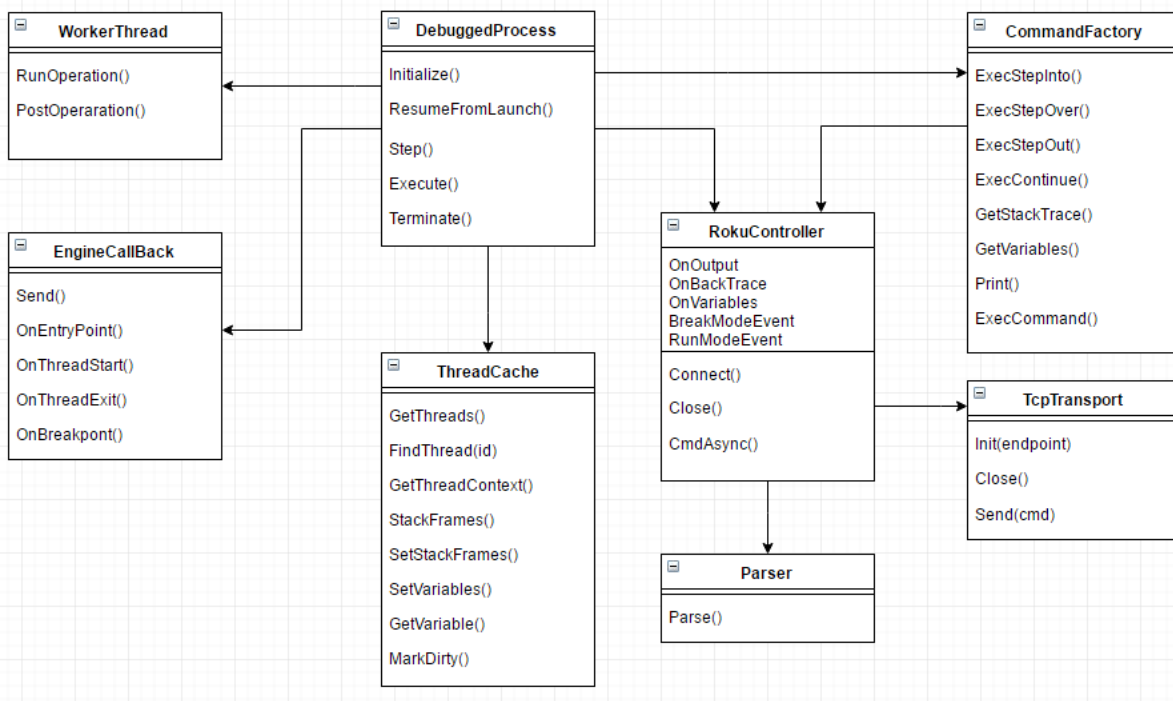


Figure 17 - Debug Engine Class Diagram

O DebuggedProcess gere todo estado recebe os comandos do AD7Engine, os eventos do RokuController, usa o EngineCallBack para enviar os eventos para o Visual Studio.

EngineCallBack envia os eventos para o Visual Studio.

O RokuController gere a conexão com a box, utilizando o TcpTransport, o Parser para processar o output e gere o envio de comandos e as respostas.

O CommandFactory envia os comandos específicos para a o RokuController.

A WorkerThread gere o background work.

O ThreadCache faz cache do processamento das variáveis e do call stack.

O VariableInformation representa uma variável na box.

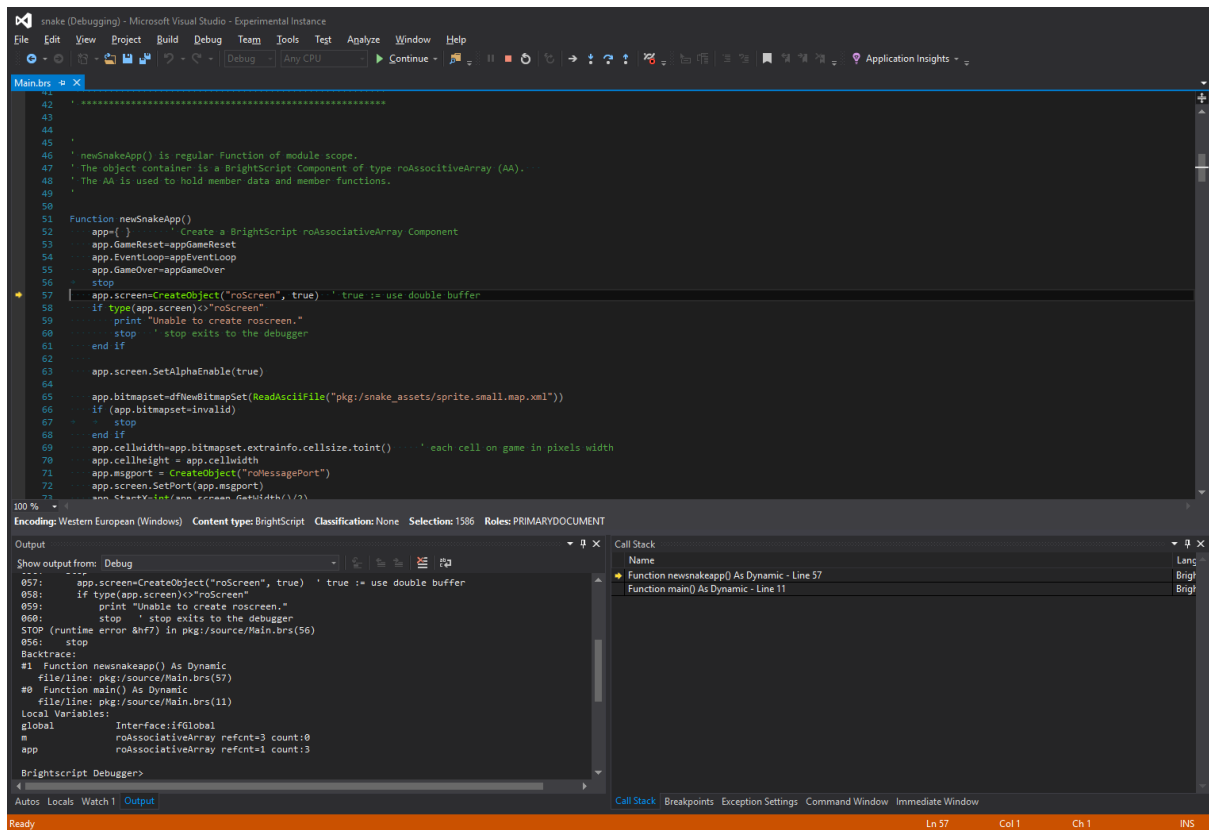


Figure 18 - Debugging BrightScript

O DebuggedProcess envia todo o output para o Visual Studio.

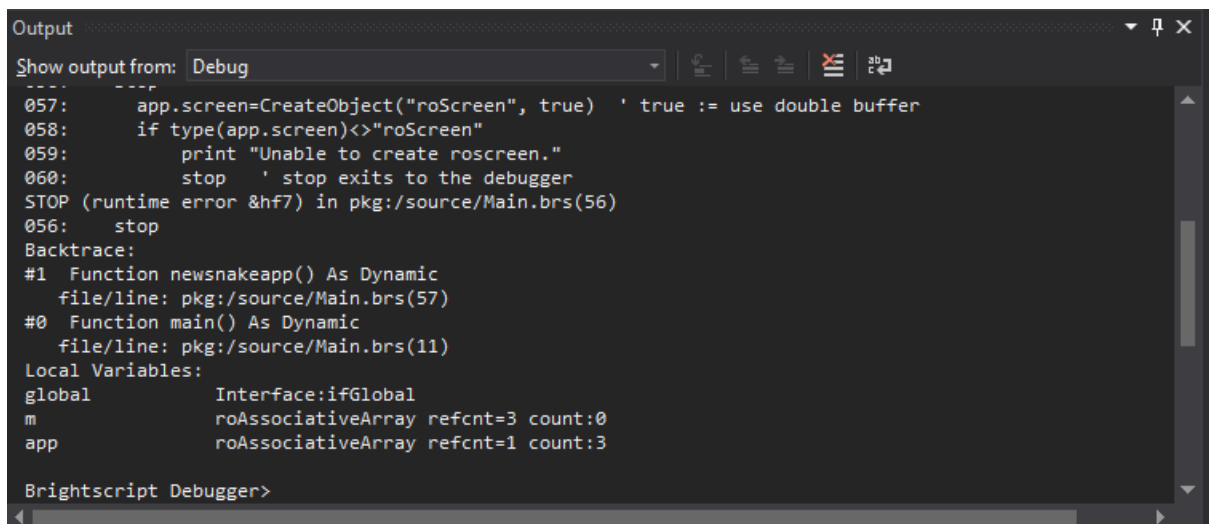


Figure 19 - Output Window

Quando a box entra em break mode o DebuggedProcess envia o call stack um evento AD7BreakpointEvent para o Visual Studio com o stack frame.

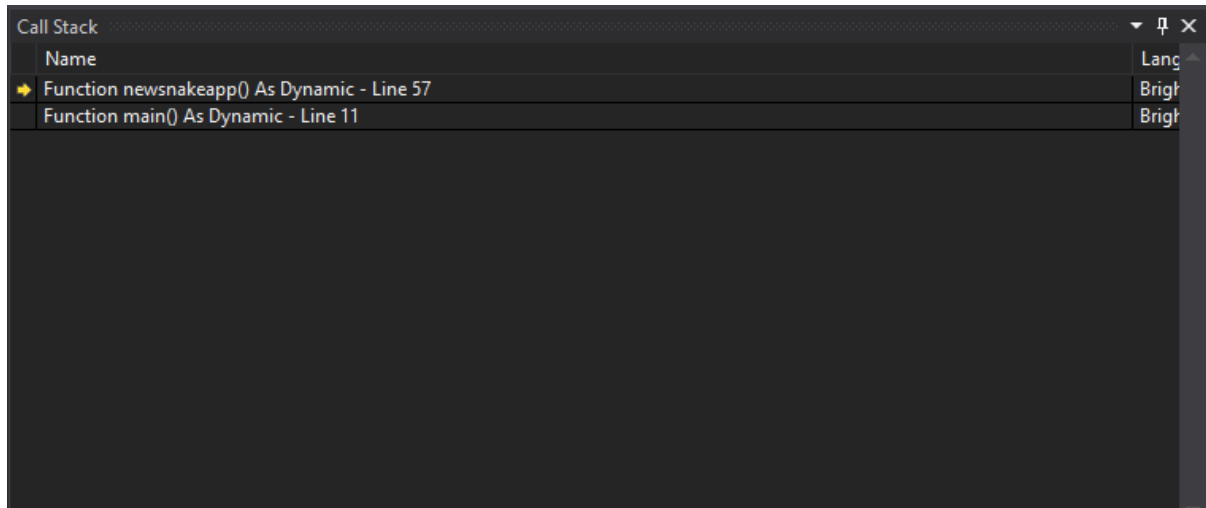


Figure 20 - Call Stack Window

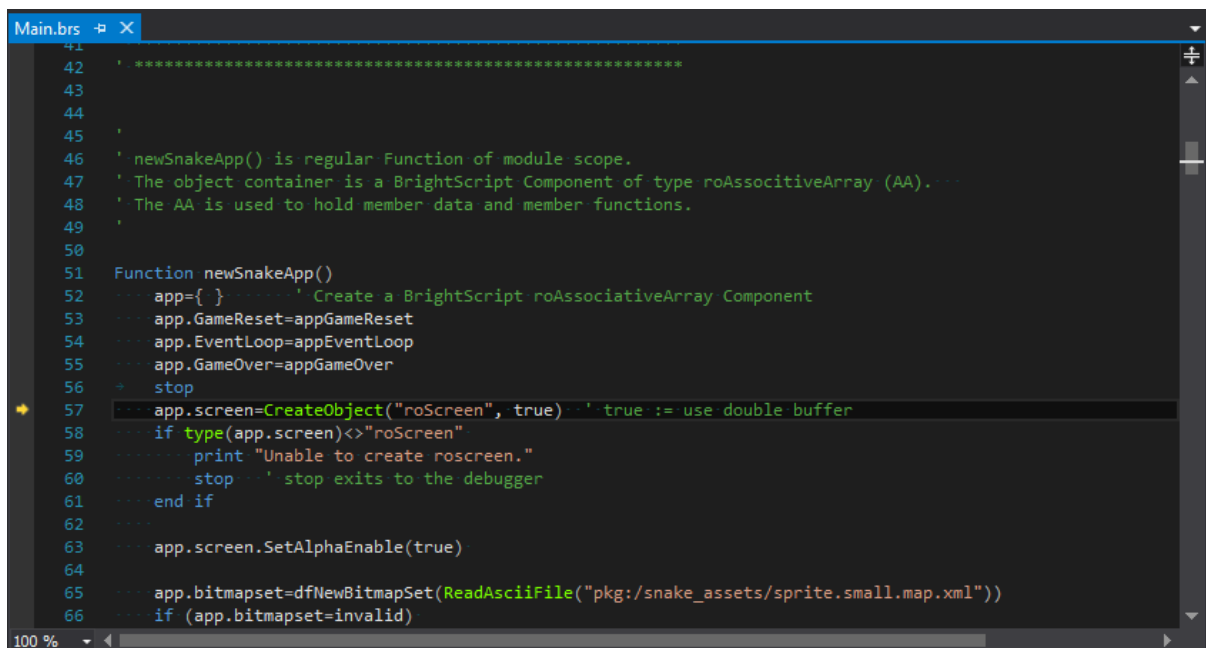


Figure 21 - Editor window showing break point

O call stack também tem a lista de variáveis locais que podem ser visualizadas na janela de Locals.

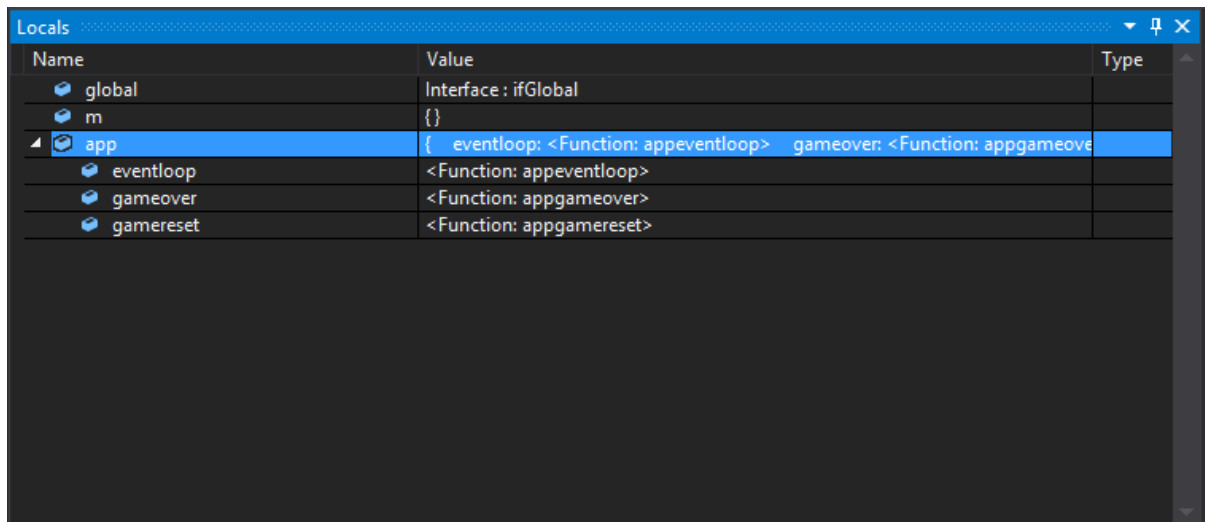


Figure 22 - Locals Window

Ou na janela de Watch.

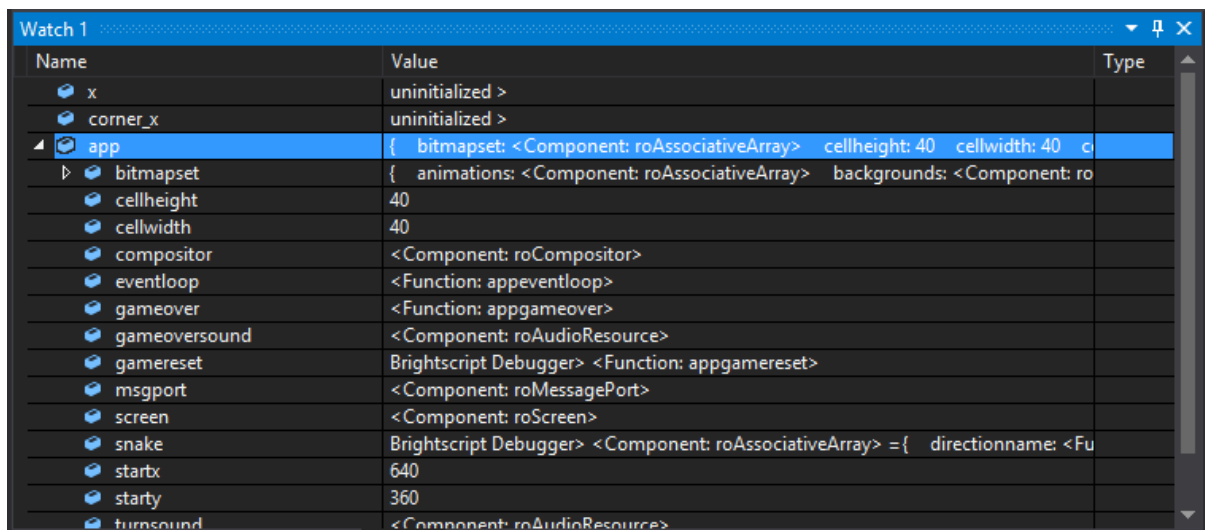


Figure 23- Watch Window

## 5.5. Tool Windows

Foram criadas duas janelas para integrar o desenvolvimento no Visual Studio. Um é o Remote desenvolvido na Debugger e outra é um serviço que faz screen shots da box, permitindo visualizar o ecrã da box.



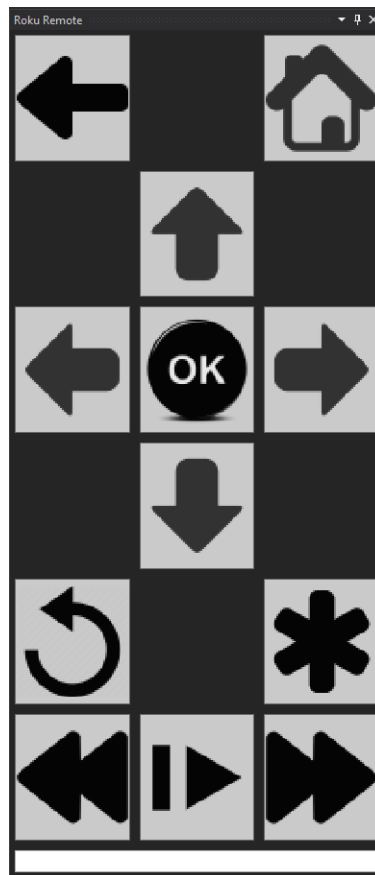


Figure 24 - Remote Window

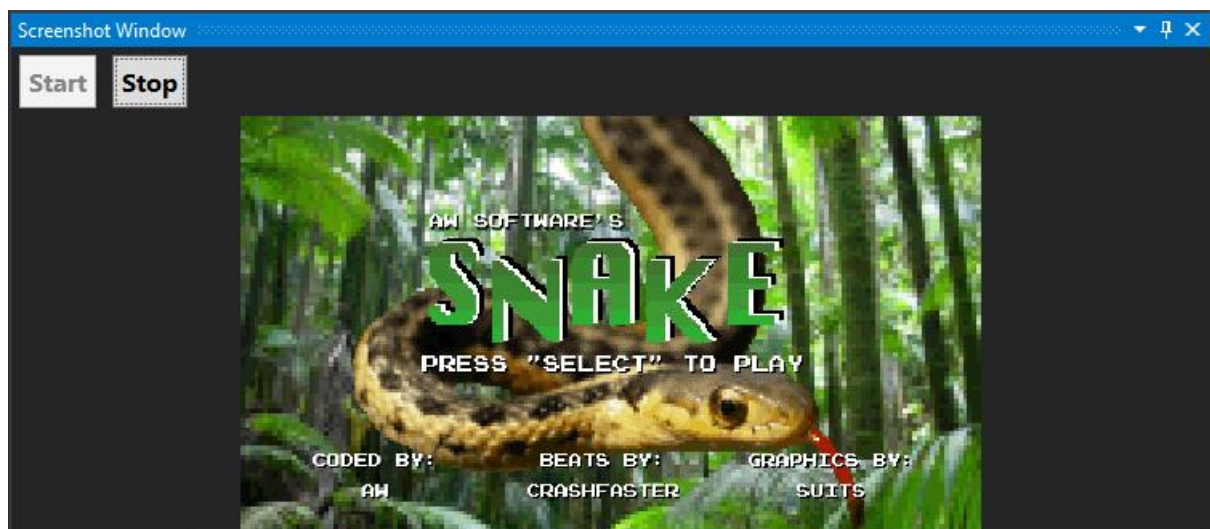


Figure 25 - Screenshot Window

# 6. Conclusão

A extensão do Visual Studio suporta todas as features propostas, edição de código, syntax highlighting, validação de erros, intellisense, build e deploy da aplicação e debug integrado.

A extensão pode ser utilizada em projetos simples, mas precisa de alguns melhoramentos para utilizar em projetos mais complexos.

O Parser não suporta todas as Statements. Tais como if eles encadeados e o uso de palavras reservadas como identificadores.

O Debug Engine precisa de mais performance e de permitir a visualização de variáveis no stack.

O intellisense só usa o input do ficheiro corrente. O compilador precisa de gerar uma base de dados com todas as ASTs, para servir de input para o Intellisense.

Como trabalho futuro identificamos as seguintes tarefas: uma ferramenta para converter a base de código existente e adicionar suporte para a nova framework Scene Graph.