



Instituto Superior de Engenharia de Lisboa

Área Departamental de Engenharia de Eletrónica e Telecomunicações e de
Computadores

Debugger BrightScript

Celso de Almeida Fernandes

(Licenciado Engenharia Informática e de Computadores)

Dissertação para obtenção do Grau de Mestre em Engenharia Informática e de
Computadores

Orientadores: Eng. Paulo Pereira
Eng. Pedro Pereira

Júri:

Presidente: Eng. Manuel Barata

julho de 2016

1. Resumo

O BrighthScript é uma linguagem de programação baseada em java script e visual basic que foi criada pela Roku. A Roku é uma empresa que desenvolve e comercializa box's para visualização de conteúdos na televisão. O BrighthScript é a linguagem para desenvolvimento de aplicações para as boxs.

Após análise constatou-se que as ferramentas disponíveis para o desenvolvimento de aplicações são escassas e pouco funcionais. Essas ferramentas são um plugin básico para o Eclipse (validação sintática) e as boxs disponibilizam um porto telnet para output e comandos básicos de debug.

Este projeto consiste em implementar uma ferramenta integrada que permita facilitar o desenvolvimento das aplicações, nomeadamente através das funcionalidades de validação sintática, compilação, intellisense e interação gráfica de debug.

A implementação baseia-se no desenvolvimento dum plugin de BrightScript para o Visual Studio. O Visual Studio é um IDE desenvolvido pela Microsoft e largamente utilizado para desenvolver aplicações para Windows. Esse plugin assenta nas APIs (Language Services) disponibilizadas pelo SDK do Visual Studio.

Esta ferramenta pode ser complementada com a implementação dum simulador de execução das aplicações, esta implementação é considerada opcional, podendo ser uma mais-valia para o desenvolvimento de aplicações.

O projeto enquadra-se no desenvolvimento da aplicação Sky Store pertencente à empresa Sky UK Limited.

2. Índice

1. Resumo.....	i
2. Índice	ii
3. Introdução	1
3.1. Compiladores	1
3.2. Ferramentas	1
3.3. Implementação	2
4. Compilador	4
4.1. Scanner.....	4
4.2. Parser	4
4.3. Plugin Visual Studio	5
5. Debugger	6
5.1. Deploy	6
5.2. Telnet	7
5.3. Remote Http.....	10
5.4. Visual Studio.....	11
6. Bibliografia.....	12
7. Índice Figuras.....	13

3. Introdução

O projeto divide-se em três fases. A primeira é uma fase de investigação e aquisição de conhecimentos sobre a temática dos compiladores. A segunda fase consiste na análise das ferramentas disponíveis, para compreender o seu funcionamento e avaliar o benefício da sua utilização. A terceira fase corresponde à implementação do plugin.

3.1. Compiladores

Na primeira fase foram utilizados um conjunto de vídeos (Aiken, s.d.) disponíveis no site youtube, realizados pelo professor Alex Aiken, da universidade de Stanford e o livro Modern Compiler Implementation (Appel, 2002) in Java, recomendado pelo orientador. As duas fontes abordam a problemática de formas semelhantes, apresentando as problemáticas da implementação dos compiladores, sugerindo uma implementação modular.

O BrightScript é uma linguagem interpretada, não sendo necessário implementar todos os paços do processo de compilação. Para a ferramenta de debug basta-nos implementar o Lexer e o Parser. Para implementar simulador será necessário implementar os restantes paços do processo, podendo optar pela compilação para a linguagem intermedia MSIL (Microsoft Intermediate Language).

3.2. Ferramentas

Na segunda fase analisa-se a utilização de ferramentas para gerar o Lexer e o Parser.

O GPlex (GPlex, s.d.) é um gerador de código C#, que gera um analisador léxico com base num ficheiro de especificação semelhante à linguagem de especificação Lex. O analisador é baseado no algoritmo “finite state autómata”.

O Gppg (Gppg, s.d.) é um gerador de código C#, que gera um Parser com a abordagem bottom-up, que reconhece linguagens LALR(1) (1 Look-Ahead token, Left-to-Right - rightmost derivation), com as desambiguações yacc tradicionais. A especificação é feita numa linguagem semelhante ao YACC.

Os geradores de código foram desenhados para funcionar em conjunto, podendo ainda assim ser utilizados em separado. Foram também desenhados para integrar com o Visual Studio, existindo opções para gerar classes para a integração.

Para além dos geradores de código, também se analisou a interface disponibilizada pelo Visual Studio para a implementação de extensões, nomeadamente extensões de suporte a linguagens e de debug.

3.3. Implementação

Na terceira fase é feita a implementação do debugger. A implementação é composta por três componentes, como mostra o seguinte diagrama.

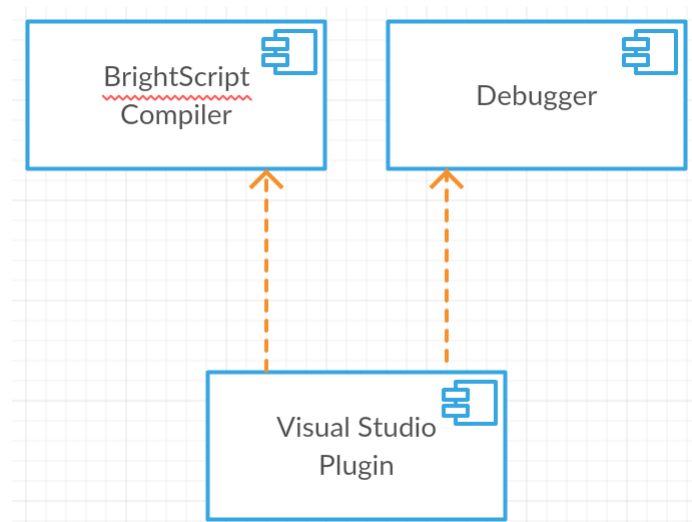


Figura 1 - Diagrama de componente

O BrightScript Compiler é um compilador de código, que tem como objetivo validar a sintaxe do código, prevenindo erros de compilação na box.

O Debugger tem como objetivo tratar a comunicação com a box, utilizando os protocolos http e telnet. A box disponibiliza um porto http para emular o input do comando e uma pagina web que permite fazer deploy das aplicações desenvolvidas. A box disponibiliza ainda um porto telnet para o output e receber comandos de debug.

O Plugin será implemento baseando-se na implementação do Python Tools (Python Tools, s.d.). O Python Tools é uma extensão do Visual Studio para a linguagem Python, desenvolvido pela Microsoft. O Plugin usa o compilador para

fazer highlight do código e suportar o inteliense e utiliza o Debugger para fazer deploy e interagir com o debugger telnet.

4. Compilador

O compilador é composto por três componentes, o analisador léxico (scanner), o Parser e a componente de integração com o Visual Studio. O objetivo das duas primeiras componentes é processar os ficheiros de código de forma a detetar erros de compilação e disponibilizar dados para o intellisense.

4.1. Scanner

O analisador léxico tem como responsabilidade gerar tokens para serem usados no Parser e validar se o código obedece ao léxico definido para a linguagem.

O léxico define o formato dos tokens que compõem a linguagem, o formato é definido através dum conjunto de regras e as regras são definidas através de expressões regulares.

Um analisador léxico é uma máquina de estados que tenta encontrar os tokens com as maiores dimensões. O processamento é feito carácter a carácter, enquanto existir possibilidade de encontrar um token maior, ao encontrar o token a string é retirada do input e é emitido o token.

O Scanner é gerado utilizando o Gplex, que gera uma classe em C# com a implementação. O código gerado é obtido de três fontes, da estrutura base da classe, motor de reconhecimento das patterns e os decoders/buffer de leitura. O motor de reconhecimento é composto por tabelas que definem a máquina de estados (FSA - finite state automaton). Estas tabelas são geradas através do ficheiro de especificação (*.lex).

Foram gerados dois scanners, um mais simples para syntax highlighting com um conjunto reduzido de tokens e outro mais completo para usar com o Parser.

4.2. Parser

O Parser tem como objetivo analisar a estrutura gramatical da linguagem, as frases. Valida se a ordem pela qual se apresentam os tokens é válida. Essa análise permite estruturar o código numa árvore de tokens. O output do Parser é uma árvore abstrata da estrutura do código (AST - Abstract Syntax Tree).

A gramática é constituída por um conjunto de regras, que determinam a ordem dos símbolos nas sequências válidas.

O Parser é gerado utilizando o Gppg, que gera uma classe em C# com a implementação. O código gerado implementa o algoritmo Shift-Reduce para gerar a AST (Abstract Syntax Tree).

O Parser implementado está a fazer a validação gramatical, mas ainda não está a gerar a AST, será implementado na segunda fase do projeto.

4.3. Plugin Visual Studio

O Visual Studio utiliza o Lexer para atribuir diferentes cores aos tokens e o Parser para carregar os dados para o Intellisense. São ainda usados os dois componentes para validar o código, evitando os erros de compilação na box.

Para além destas funcionalidades será criado um novo tipo de projeto para gerir as configurações e associar os templates para criação de novos ficheiros de código e outros tipos de ficheiros.

A implementação do plugin será efetuada na segunda fase do projeto. Para simular as chamadas ao compilador foi criado um projeto para testar o processo de compilação. Esse projeto cria um scanner e um parser para compilar um ficheiro específico.

5. Debugger

O debugger é uma tool que permite fazer deploy da aplicação para as boxs e fazer o interface entre o Visual Studio e o porto Tenet da box. Para além do porto Telnet a box disponibiliza um porto http, que permite simular ações do comando e iniciar aplicações.

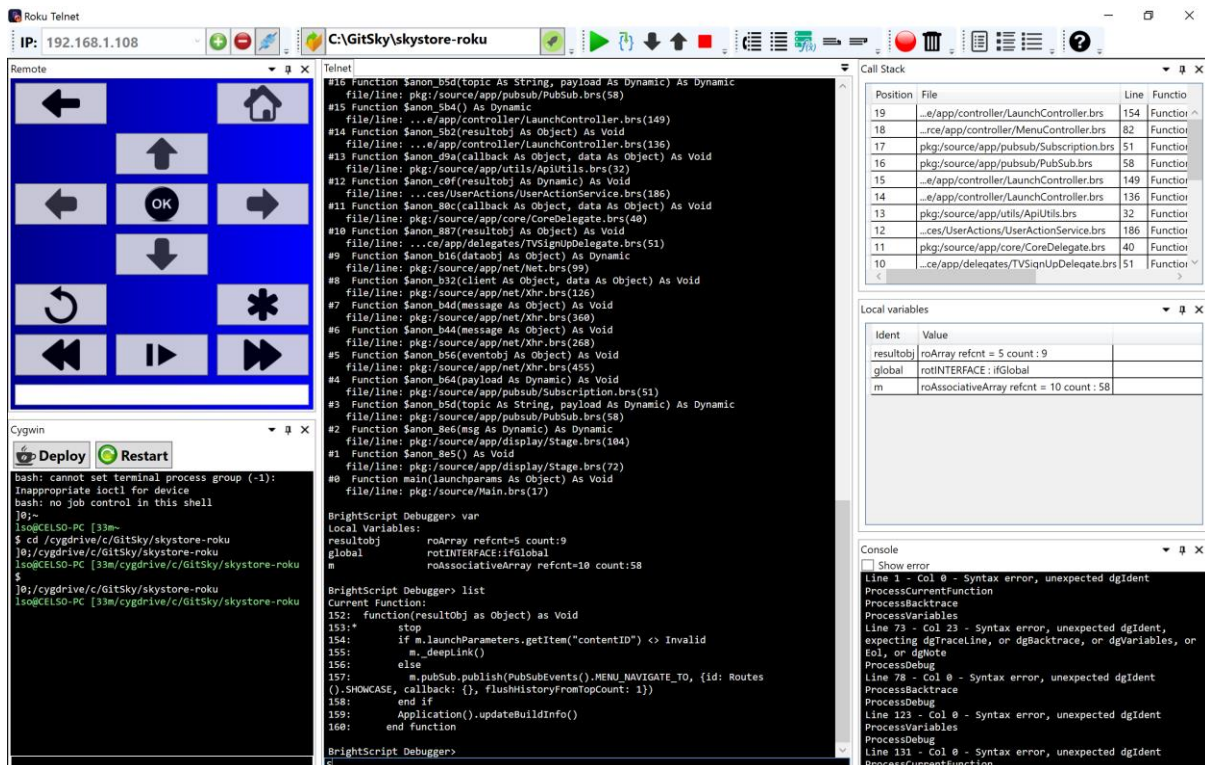


Figura 2 - Aplicação Debug

5.1. Deploy

O deploy consiste em gerar um zip com todos os ficheiros da aplicação e fazer o upload para a box, utilizando o porto http.

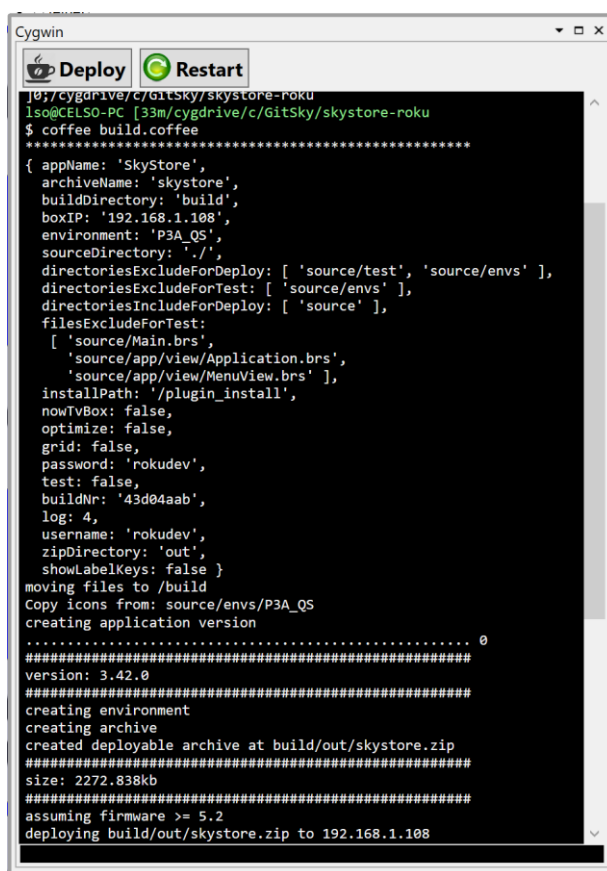
O processo de deploy deve ser configurável, permitindo:

- Selecionar as pastas a enviar
- Injetar break points
- Eliminar troços de código utilizados só para debug
- Passar parâmetros para a aplicação, através de injeção de código
- Executar testes unitários
- Gerar/Atualizar o ficheiro manifest

O deploy consiste em quatro passos:

1. Copiar ficheiros
2. Edição automatizada
3. Geração do ficheiro zip
4. Upload do ficheiro zip para a box

O processo de deploy será implementado na segunda fase do projeto, para efetuar o deploy são utilizados os scripts de coffee existentes no código da aplicação. Para facilitar a integração foi criado um componente gráfico que permite executar os scripts, recorrendo ao Cygwin.



```
Cygwin
Deploy Restart
j0:/cygdrive/c/GitSky/skystore-roku
ls@CELSO-PC [33m/cygdrive/c/GitSky/skystore-roku
$ coffee build.coffee
*****
{ appName: 'SkyStore',
  archiveName: 'skystore',
  buildDirectory: 'build',
  boxIP: '192.168.1.108',
  environment: 'P3A_QS',
  sourceDirectory: './',
  directoriesExcludeForDeploy: [ 'source/test', 'source/envs' ],
  directoriesExcludeForTest: [ 'source/envs' ],
  directoriesIncludeForDeploy: [ 'source' ],
  filesExcludeForTest:
    [ 'source/Main.brs',
      'source/app/view/Application.brs',
      'source/app/view/MenuView.brs' ],
  installPath: '/plugin_install',
  nowTVBox: false,
  optimize: false,
  grid: false,
  password: 'rokudev',
  test: false,
  buildNr: '43d04aab',
  log: 4,
  username: 'rokudev',
  zipDirectory: 'out',
  showLabelKeys: false }
moving files to /build
Copy icons from: source/envs/P3A_QS
creating application version
..... 0
version: 3.42.0
creating environment
creating archive
created deployable archive at build/out/skystore.zip
*****
size: 2272.838kb
*****
assuming firmware >= 5.2
deploying build/out/skystore.zip to 192.168.1.108
```

Figura 3 - Cliente Cygwin

5.2. Telnet

O porto Telnet tem duas funcionalidades, mostrar o output da execução da aplicação e servir de terminal de debug.

O output é feito através da utilização da função “print” no código.

Para utilizar o terminal de debug utiliza-se a função “stop” no código. Quando a função for executada a aplicação para e é possível interagir com o debugger utilizando os seguintes comandos:

Comando	Descrição
bsc	Mostra as instancias correntes
bscs	Mostra o sumario das instancias correntes
brkd	Para execução com mensagens não fatais
bt	Mostra o callstack
classes	Mostra as classes
cont or c	Continua execução do Script
down or d	Mover execução para baixo
exit	Sair
gc	Correr o garbage collector
help	Mostra a lista de comandos do sistema
last	Mostra a ultima linha executada
list	Mostra a função corrente
next	Mostra a proxima linha
print , p , or ?	Imprime para o output
step , s , or t	Executa uma instrução
up or u	Mover execução para cima
var	Mostra variaveis locais e os seus tipos
Any Brightscript statement	Executar BrightScript

Utilizando o debugger é possível visualizar o valor corrente das variáveis, verificar em que ficheiro e em que linha está a execução e controlar a execução do código.

A implementação do componente de telnet utiliza um socket para se ligar ao porto telnet, um compilador para tratar o output. Para facilitar a interface visual, foram criados alguns componentes visuais, como mostra o diagrama seguinte.

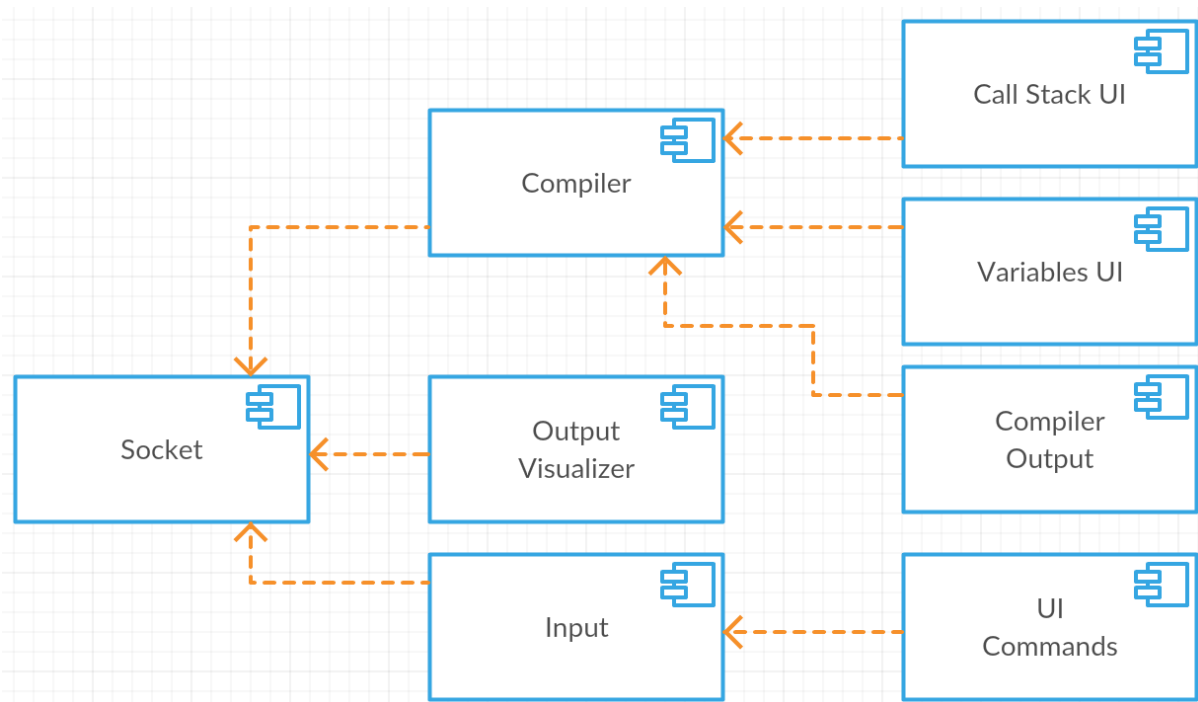


Figura 4 - Diagrama de componentes do telnet

O compilador identifica os padrões do output para reconhecer o call stack e o valor das variáveis locais. O Compiler Output mostra o output do compilador.

Call Stack

Position	File	Line	Funcion
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	rotINTERFACE : 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
        
```

Figura 5 - UI do compilador

O output visualizer e o input usam diretamente o socket para enviar comandos e mostrar o output da box.

```
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})
158:      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
```

Figura 6 - Output visualizer

O UI commands é um conjunto de botões que correspondem aos disponibilizados para debug.

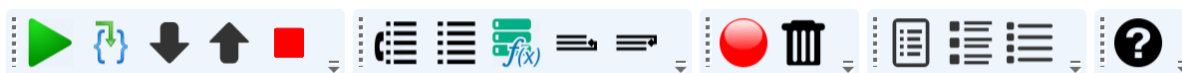


Figura 7 - UI Commands

5.3. Remote Http

Utilizando o porto http será implementado um comando para controlar a box através do PC.

O comando é composto pelos botões do comando e por uma textbox que permite a introdução de texto.

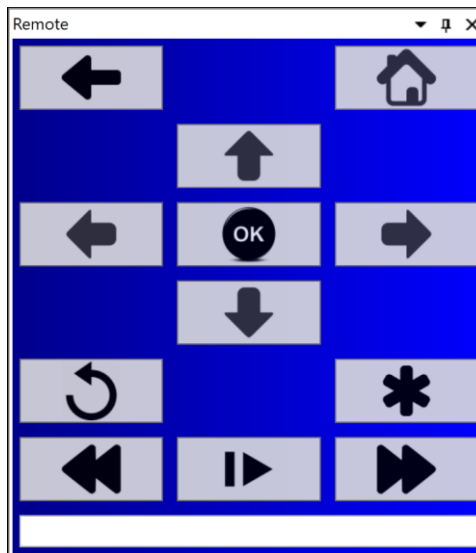


Figura 8 - Remote

5.4. Visual Studio

A componente de integração do debugger com o Visual Studio, vai permitir utilizar a interface visual de debug de forma semelhante às aplicações .Net. Permitindo definir breakpoint's, controlar a execução do código e visualizar o valor corrente das variáveis. O ponto de execução vai ser visualizado nos ficheiros de código através da funcionalidade de consultar o ponto de execução.

6. Bibliografia

Aiken, A. (s.d.). *Compilers Theory*. Obtido de <https://www.youtube.com/playlist?list=PLLH73N9cB21VSVEX1aSRINTufaLK1dTAI>

Appel, A. W. (2002). *Modern Compiler Implementation in Java*. Cambridge University Press.

GPlex. (s.d.). Obtido de <http://gplex.codeplex.com/>

Gppg. (s.d.). Obtido de <https://gppg.codeplex.com/>

Python Tools. (s.d.). Obtido de <https://github.com/Microsoft/PTVS>

7. Índice Figuras

Figura 1 - Diagrama de componente	2
Figura 2 - Aplicação Debug.....	6
Figura 3 - Cliente Cygwin.....	7
Figura 4 - Diagrama de componentes do telnet	9
Figura 5 - UI do compilador.....	9
Figura 6 - Output visualizer	10
Figura 7 - UI Commands.....	10
Figura 8 - Remote.....	11