RELATÓRIOS TÉCNICOS DO DEPARTAMENTO DE ARQUIVOLOGIA
UNIVERSIDADE FEDERAL DE SANTA MARIA

ISSN 2763-745X

DARQ/ARQDIG.02.2020.MAN

Simple Computer: Manual de Operação

Autor
Andre Zanki Cordenonsi
andre.cordenonsi@ufsm.br

Departamento de Arquivologia
UFSM

Versão 1.0
Status: Final
Distribuição: externa
Dezembro 2020

2020 Departamento de Arquivologia da Universidade Federal de Santa Maria

Atribuição-Não Comercial 4.0 Internacional (CC BY-NC 4.0)
Você tem o direito de compartilhar, copiar e redistribuir o material em qualquer suporte ou formato; adaptar, remixar, transformar, e criar a partir do material, de acordo com o seguinte: você deve dar o crédito apropriado, prover um link para a licença e indicar se mudanças forem feitas. Você deve fazê-lo em qualquer circunstância razoável, mas de nenhuma maneira que sugira que o licenciante apoia você ou seu uso. Você não pode usar o material para fins comerciais.

Departamento de
Arquivologia
UFSM

Av. Roraima 1000
Prédio 74A – Sala 2147
Santa Maria – RS – CEP 97105-900
Fone: +55 55 3220 9255 – Departamento de Arquivologia
email: depto.arquivologia@ufsm.br

Editoria Técnica do Departamento de Arquivologia – EdTec/DArq
Editor: Andre Zanki Cordenonsi
email: edtec.darq@ufsm.br

Relatórios Técnicos do Departamento de Arquivologia / Departamento de Arquivologia, Universidade Federal de Santa Maria. – Vol. 1. n. 1 (2020) Jul/Dez. – Santa Maria: DArq/UFSM, 2020.

Periodicidade semestral.

1. Arquivologia. 2. Ciência da Informação. I. Universidade Federal de Santa Maria. Departamento de Arquivologia.

Como citar este relatório:

CORDENONSI, A.Z. **Simple Computer: Manual de Operação**. Santa Maria: 2020. Relatórios Técnicos do Departamento de Arquivologia, v.1., n.1. Disponível em: https://www.ufsm.br/cursos/graduacao/santa-maria/arquivologia/darq-arqdig-02-2020-man

Resumo

Este manual tem como objetivo apresentar a instalação, funcionamento e os comandos disponíveis para o Simple Computer, um *software* de interpretação de comandos baseado em *JavaScript* que pode ser utilizado como suporte à aulas de Documentos Arquivísticos Digitais e disciplinas assemelhadas.

1. Introdução

O Simple Computer é um software desenvolvido com base no sistema *Simple Ima*ge, disponibilizado pela Universidade de Stanford sob a licença Apache 2.0 (http://www.apache.org/licenses/LICENSE-2.0). Ele foi expandido para outros tipos de interação de dados, incluindo diversos novos comandos, permitindo a manipulação de caracteres, números, imagens, áudio e vídeos.

O Simple Computer permite que os docentes e discentes tenham uma plataforma para testes e manipulação de dados binários, conceito essencial para a compreensão de arquivos em formato digital. Ele é dividido em cinco grandes blocos, que são apresentados nos próximos capítulos, com seus comandos e exemplos.

2. Instalação e Funcionamento

O Simple Computer foi desenvolvido em JavaScript e precisa ser instalado em um servidor HTTPD. A linguagem JavaScript é uma linguagem compacta e baseada em objetos e permite que o sistema rode em qualquer navegador existente, independente de sistema operacional ou plataforma de *hardware* (MOTA, 2010). O sistema é disponibilizado pela licença Apache 2.0, o que permite que o uso e a distribuição do código fonte.

O sistema é formado pelos seguintes arquivos, que são distribuídos em um único arquivo compactado:

Tabela 1. Lista de arquivos do SimpleComputer

`cs101-edx.js`	Arquivo em formato JavaScript que contém os principais comandos do SimpleComputer.
`cs101-table-edx.js`	Arquivo em formato JavaScript que contém os comandos para a manipulação de tabelas
`default.css`	Arquivo com o *Cascading Style Sheets* utilizado no SimpleComputer
`style-edx.css`	Arquivo com o *Cascading Style Sheets* utilizado no SimpleComputer para a manipulação de tabelas
`SimpleComputer.html`	Página HTML principal do sistema. É essa que deve ser chamada pelos usuários.
`sobre.html`	Página contendo informações sobre o sistema.
`SimpleAsciiGame1.html`	Dois exemplos de *jogos* implementados em HTML, para apresentar interfaces em formato ASCII.
`SimpleAsciiGame2.html`	

Fonte: (do autor)

Além destes arquivos, o sistema também possui um diretório chamado *files*. Nesse diretório devem ser armazenados os arquivos de imagem (formato PNG ou JPG) e os arquivos de vídeo (formato CSV) que são manipulados pelo SimpleComputer. Consulte os capítulos 5 e 7 para maiores detalhes.

Para instalar o sistema, é necessário um servidor HTML, como o Apache[1]. Como o sistema foi implementado em JavaScript, não é necessário instalar nenhum *software* adicional no servidor ou nas máquinas cliente (computadores, celulares, tabletes), rodando em qualquer navegador. Basta descompactar o arquivo disponível do SimpleComputer no diretório

[1] O Servidor HTTP Apache (do inglês Apache HTTP Server) é um servidor web livre do tipo HTTPD (do inglês: HTTPd stands for Hypertext Transfer Protocol daemon), compatível com o protocolo HTTP versão 1.1.

adequado do servidor (usualmente, o diretório HTDOCS) e acessar a partir da web. Não é necessário nenhum tipo de configuração.

Ao ser acessado em um navegador, o sistema apresentará a tela apresentada na figura 01.

Figura 1. Tela de abertura do SimpleComputer

Fonte: (do autor)

O usuário deve digitar os comandos dentro da caixa de entrada e apertar o botão *Run*; o resultado será apresentado abaixo do botão, como é possível observar na figura 02.

Figura 2. Rodando comandos no SimpleComputer

Fonte: (do autor)

2.1 Instalação standalone

Se não houver a possibilidade de instalar o SimpleComputer em um servidor *web*, é possível utilizá-lo em um computador pessoal normal utilizando um mini-servidor web. Neste caso, o sistema rodará diretamente na máquina do usuário.

Há vários mini-servidores disponíveis na internet. Neste manual, apresentaremos o servidor **miniweb**, disponível em http://miniweb.sourceforge.net/. O *software* é disponível em um único arquivo compactado, que deve ser descompactado em um diretório qualquer. Assim que baixar o arquivo e descompactá-lo, ele apresentará o seguinte arquivo e diretório (figura 03):

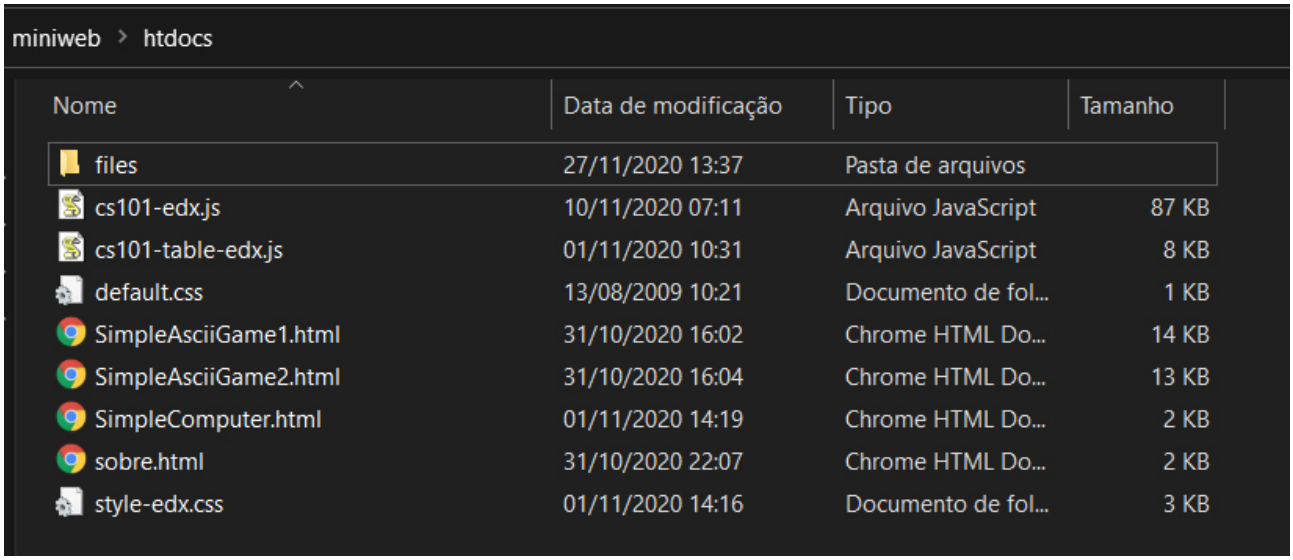
Figura 3. Instalação do miniweb

Nome	Data de modificação
htdocs	10/11/2020 07:11
miniweb.exe	27/02/2013 13:48

Fonte: (do autor)

Entre no diretório *htdocs* e exclua todo o seu conteúdo. Então, baixe o arquivo SimpleComputer.zip neste diretório e o descompacte. A estrutura do *htdocs* deverá ser parecida com a que aparece na figura 04.

Figura 4. Instalação do SimpleComputer no miniweb



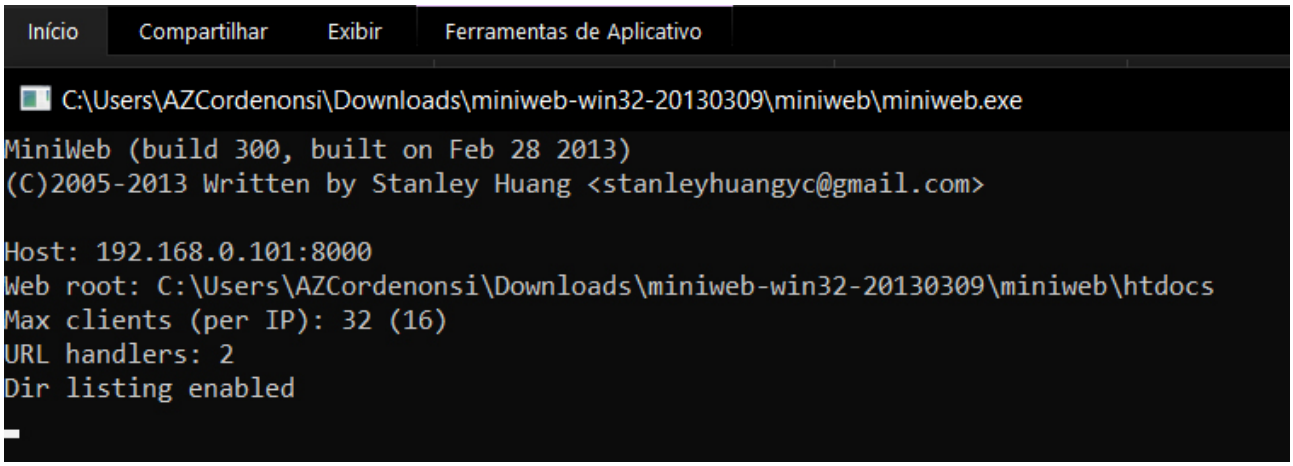
miniweb › htdocs

Nome	Data de modificação	Tipo	Tamanho
files	27/11/2020 13:37	Pasta de arquivos	
cs101-edx.js	10/11/2020 07:11	Arquivo JavaScript	87 KB
cs101-table-edx.js	01/11/2020 10:31	Arquivo JavaScript	8 KB
default.css	13/08/2009 10:21	Documento de fol...	1 KB
SimpleAsciiGame1.html	31/10/2020 16:02	Chrome HTML Do...	14 KB
SimpleAsciiGame2.html	31/10/2020 16:04	Chrome HTML Do...	13 KB
SimpleComputer.html	01/11/2020 14:19	Chrome HTML Do...	2 KB
sobre.html	31/10/2020 22:07	Chrome HTML Do...	2 KB
style-edx.css	01/11/2020 14:16	Documento de fol...	3 KB

Fonte: (do autor)

Finalmente, retorne para o diretório do miniweb e execute o único arquivo disponível (miniweb.exe). Será aberta uma caixa de linha de comando como apresentada na figura 05.

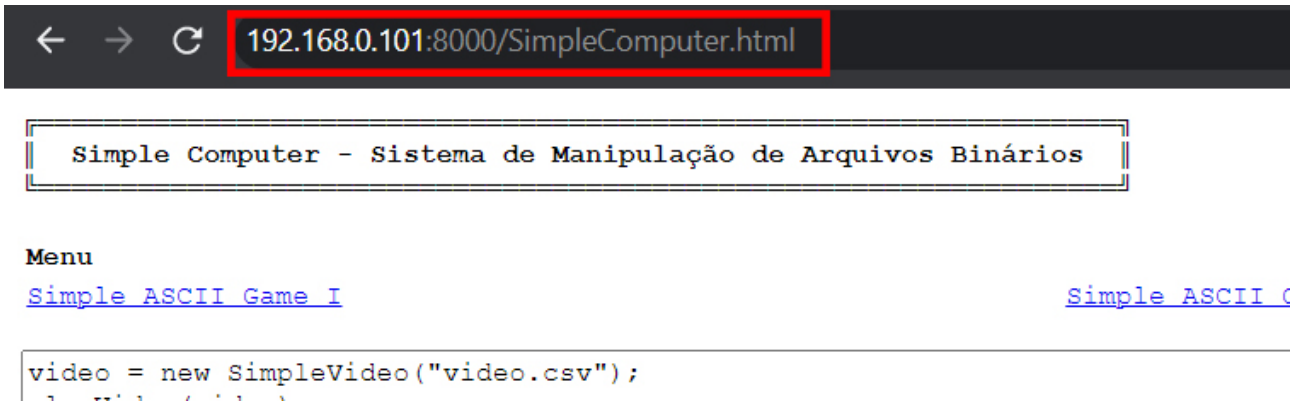
Figura 5. Rodando o miniweb



Fonte: (do autor)

Nesta caixa de linha de comando, a única coisa que importa é o número do *host* e a *porta* (no caso, 192.168.0.101, porta 8000). Este é o endereço que deve ser colocado no navegador de sua preferência (Chrome, Edge, Firefox, etc.) para acessar o SimpleComputer. Simplesmente copie o host e a porta para a barra do navegador, acrescentando /*SimpleComputer.html* ao final, como aparece na imagem da figura 06 (não esqueça do : entre o endereço IP e o número da porta, nem a / entre o número da porta e o nome da página HTML).

Figura 6. Acessando o SimpleComputer pelo miniweb



Fonte: (do autor)

Note que, a cada vez que o miniweb for executado, o número do host poderá mudar.

3. Manipulação de Caracteres

A manipulação de caracteres em computadores se refere à capacidade de armazenamento de caracteres alfanuméricos em formato binário, considerando que os meios digitais (*pendrives*, cds, discos rígidos) só podem armazenar dados neste formato. No Simple Computer, os caracteres são armazenados em formato binário, utilizando o código ASCII estendido, com 256 possíveis caracteres (VELLOSO, 2017).

Existem seis comandos básicos para a manipulação de caracteres no Simple Computer.

COMANDO printABIN - ASCII PARA BINÁRIO

```
printABIN("A");
printABIN("2");
```

Imprime na tela o código binário da tabela ASCII de um determinado caractere ou conjunto de caracteres. Os códigos binários são apresentados separados por um espaço. Não há preenchimento de 0's à esquerda no código apresentado; desta forma, caracteres do conjunto ASCII são apresentados com 7 bits e caracteres do conjunto ASCII estendido são apresentados com 8 bits.

```
printABIN("AÇ");
1000001 11000111
```

COMANDO: printBASC - BINÁRIO PARA ASCII

```
printBASC("10010000");
```

Imprime na tela o código ASCII de um determinado caractere a partir de sua codificação binária. Se enviado mais de um valor, ele deve ser separado por vírgulas. Se enviado um código inválido (fora do conjunto ASCII estendido), o comando não imprime nada.

```
printBASC("10010000, 110101");
É5
```

COMANDO: printDASC DECIMAL PARA ASCII

```
printDASC(65,66,67);
```

Imprime na tela o caractere ASCII a partir de um determinado número decimal enviado. Se enviado mais de um valor, ele deve ser separado por vírgulas.

```
printDASC(65,66,67);
ABC
```

COMANDO: printMDASC - MÚLTIPLOS DECIMAIS PARA ASCII

```
printMDASC(220,5);
```

Imprime na tela várias vezes o caractere ASCII a partir de um determinado número decimal enviado. Deve ser informado o número decimal do caractere ASCII e o número de vezes que se deseja imprimir.

```
printMDASC(220,5);
```

COMANDO: printNL - NOVA LINHA

```
printNL();
```

Pula a linha da tela (nova linha).

COMANDO: printTXT - IMPRIMIR TEXTO

```
printTXT("Arquivologia");
```

Imprime o texto na tela.

```
printTXT("Arquivologia");
Arquivologia
```

4. Manipulação de Números

A manipulação de números em computadores se refere à capacidade de armazenamento de números na base binária. No Simple Computer, os números são armazenados em binário, assim como todos os sistemas computacionais.

Existem dois comandos básicos para a manipulação de números no Simple Computer.

COMANDO: printDBIN - DECIMAL PARA BINÁRIO

```
printDBIN(9);
printDBIN(155);
```

Imprime na tela o número binário a partir de um número decimal.

```
printDBIN(9);
1011
```

COMANDO: printBDEC - BINÁRIO PARA DECIMAL

```
printBDEC("110101");
```

Imprime na tela o valor decimal a partir de um número binário.

```
printBDEC("110101");
53
```

5. Manipulação de Imagens

A manipulação de imagens em computadores se refere à capacidade de manipular uma grande quantidade de números que representam a codificação de cor de cada pixel da imagem. No Simple Computer, as imagens são codificadas no sistema RGB (*red*, *green*, *blue*), com oito *bits* para cada cor; ou seja, cada pixel pode conter valores entre 0 e 255 para cada cor (MARÇULA, BENINI FILHO, 2013). Compreender a manipulação binária de uma imagem é essencial para a discussão sobre a *qualidade* da imagem, quantidade de *bytes* necessários para seu armazenamento, tamanho em largura e altura, entre outros. No Simple Computer, as imagens que podem ser manipuladas devem estar previamente carregadas no servidor.

Existem onze comandos básicos para a manipulação de imagens no Simple Computer, além de ser necessário compreender a ideia de variável.

DEFINIÇÃO: VARIÁVEIS

Uma variável é um objeto capaz de reter e representar um valor ou expressão. A variável sempre tem um nome e um valor associado. Valores alfanuméricos devem ser expressos entre aspas; valores numéricos, não.

```
nome = "Andre";
```

Isso significa que a variável *nome* contém o valor Andre.

```
num = 12;
```

Isso significa que a variável *num* vale 12.

COMANDO: SimpleImage - NOVA IMAGEM

```
img = new SimpleImage("x.png");
```

Este comando carrega na variável *img* uma imagem que está no servidor e que tenha o nome "x.png". Para manipular uma imagem, é necessária carregá-la primeiro. O nome do arquivo deve estar correto, ou uma mensagem de erro será apresentada.

COMANDO: img.setZoom - ZOOM

```
img.setZoom(15);
```

Aplica um zoom na imagem que está armazenada na variável *img*. Este comando apenas realizar a operação de zoom, ou seja, ele não apresenta o resultado na tela (ver comando PRINT). Valores acima de zero vão aumentar o tamanho da imagem e valores abaixo de zero vão diminuí-la.

```
img.setZoom(2); // a imagem será mostrada na proporção de 2x o seu tamanho

img.setZoom(0.4); // a imagem será mostrada na proporção de 40% do seu tamanho
```

COMANDO: img.getPixel - CARREGAR UM PIXEL

```
px = img.getPixel(0,0);
```

Carrega na variável *px* o pixel correspondente à posição 0,0 da imagem definida pela variável *img*.

COMANDO: px.setRed - SETA VERMELHO

```
px.setRed( 150 );
```

Altera o valor da cor vermelha para o pixel armazenado em *px*. Por exemplo, a seguinte sequência resultaria na imagem abaixo:

```
img = new SimpleImage("x.png");     // carrega a imagem em img
img.setZoom(15);                    // dá um zoom de 15x
px = img.getPixel(0,0);             // carrega o pixel (0,0) em px
px.setRed( 255 );                   // altera a cor vermelha de px para 255
print(img);                         // imprime a imagem em img
```

COMANDO: px.setGreen - SETA VERDE

```
px.setGreen( 150 );
```

Altera o valor da cor verde para o pixel armazenado em *px*.

COMANDO: px.setBlue - SETA AZUL

```
px.setBlue( 150 );
```

Altera o valor da cor azul para o pixel armazenado em *px*.

COMANDO: px.getRed - CARREGA VERMELHO

```
valorVermelho = px.getRed();
```

Retornar o valor da cor vermelha do pixel e armazena em *valorVermelho*. Por exemplo, a seguinte sequência resultaria no resultado abaixo (como se trata de um pixel preto, os valores das cores vermelha, azul e verde são 0).

```
img = new SimpleImage("x.png");     // carrega a imagem em img
px = img.getPixel(0,0);             // carrega o pixel (0,0) em px
r = px.getRed();                    // carrega o valor de vermelho de px em r
g = px.getGreen();                  // carrega o valor de verde em g
b = px.getBlue();                   // carrega o valor de azul em b
print(r, " - ", g, " - ", b );      // imprime o resultado
```

```
0 - 0 - 0
```

COMANDO: px.getGreen - CARREGA VERDE

```
valorVerde = px.getGreen();
```

Retornar o valor da cor verde do pixel e armazena em *valorVerde*.

COMANDO: px.getBlue - CARREGA AZUL

```
valorAzul = px.getBlue();
```

Retornar o valor da cor azul do pixel e armazena em *valorAzul*.

COMANDO: print - IMPRIMIR

```
print(img);
x = "Lucas";
print("Olá, ",x, ". Tudo bem?");
```

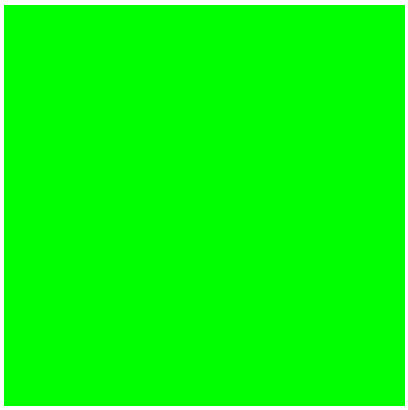
Apresenta na tela a imagem armazenada na variável *img*. Também pode ser utilizado para apresentar na tela uma série de mensagens ou valores de variáveis na sequência definida. As mensagens devem estar entre aspas e cada parte da sequência é separada por uma vírgula.

COMANDO: for – PARA CADA

```
for( pixel: img ) {
     //faça alguma coisa
}
```

Este comando executa tudo o que estiver entre as chaves { } para cada pixel presente na imagem armazenada em *img*. Por exemplo, a sequência abaixo irá carregar a imagem *x.png*, dar um zoom de fator 20 e, para cada pixel na imagem, setar o valor da cor vermelha para 0, da cor verde para 255 (máximo), e da cor azul para 0. Isso fará com que o bloco se torne completamente verde.

```
img = new SimpleImage("x.png");     // carrega a imagem em img
img.setZoom( 20 );                  // dá um zoom de 20x
for (pixel: img) {                  // para cada pixel em img
     pixel.setRed(0);               // seta o valor vermelho do pixel para 0
     pixel.setGreen(255);           // seta o valor verde do pixel para 255
     pixel.setBlue(0);              // seta o valor azul do pixel para 0
     }                              // fim-para
print(img);                         // imprime a imagem em img
```



No seguinte exemplo, a imagem *flowers.jpg* é carregada e impressa na tela em seu formato original; depois, é realizado um *for*, onde, para todos os pixels, a cor verde é setada para 0. Ou seja, foi *desligada* a cor verde, apresentando a imagem somente com as cores vermelho e azul. O resultado final é apresentado na segunda imagem.

```
img = new SimpleImage("flowers.jpg");     // carrega a imagem em img
print(img);                               // imprime a imagem em img
for (pixel: img) {                        // para cada pixel em img
     pixel.setGreen(0);                   // seta o valor verde para 0
     }                                    // fim-para
print(img);                               // imprime a imagem em img
```

6. Manipulação de Áudio

A manipulação de áudio em computadores se refere à capacidade de manipular uma grande quantidade de números que representam cada *sample* capturado durante a codificação (MARÇULA, BENINI FILHO, 2013). No Simple Computer, cada *sample* representa uma sequência de *bits*, sendo que a separação entre os *samples* é definida por um espaço em branco. A sequência de 1s (um *sample* apenas contendo valores 1s) não pode ser utilizada, pois ela marca o início da compressão, conforme será visto mais à frente.

O Simple Computer possui 04 comandos para manipular o áudio.

COMANDO: Gerar arquivo de áudio

```
som = ("0000 0100 0100 0100 0100 0100 0000 0100 0100 0100 0100 0000");
```

Gera um novo arquivo de áudio com a sequência estabelecida de dígitos binários e armazena em uma variável. É possível utilizar qualquer número de *bits* para cada *sample*, mas o número (*bitrate*) deve se manter o mesmo.

```
Som1 = ("000000 000011 000001 000111"); //CERTO

Som2 = ("000000 000011 00000 00011");   //ERRADO
```

O segundo exemplo não está correto, pois nele temos dois *samples* com seis *bits* e dois *samples* com cinco *bits*.

COMANDO: createAudio - Criar arquivo de áudio randômico

```
som1 = createAudio( sample, bitrate, segundos );
```

Cria um novo arquivo de áudio randômico com os parâmetros passados: número de *samples* por segundo, número de *bits* em cada *sample* e número de segundos que devem ser gerados. Este novo arquivo é armazenado na variável.

```
som = createAudio( 2, 5, 3 );

-----------------------------------------------
Som criado: (cores diferentes para cada segundo)
10101 01101 01111 11011 10001 00111
Número de bits: 30
-----------------------------------------------
```

Para o exemplo acima, foi criado um arquivo de áudio de três segundos, cada segundo com dois *samples*, cada *sample* com cinco *bits*.

COMANDO: audioComprime - Compressão sem perdas

```
somComprimido = audioComprime(som,"0100");
```

Cria um novo arquivo de áudio comprimido sem perdas a partir do arquivo de áudio enviado (som) e o padrão estabelecido (no exemplo, "0100").

O que o algoritmo faz é encontrar, no arquivo de áudio, o padrão enviado pelo comando. Se ele encontrar, ele substitui o padrão pelo sinal de início de repetição (um *sample* preenchido com números 1s), seguido do *sample* a ser repetido e, finalmente, o número de vezes que ele dever ser repetido (número em binário).

```
som = ("0000 0100 0100 0100 0100 0100 0000 0100 0100 0100 0100 0000");
somComprimido = audioComprime(som,"0100");


Resultado:
---------------------------------------------
Compressão sem perdas: samples iguais
Original:
0000 0100 0100 0100 0100 0100 0000 0100 0100 0100 0100 0000
Número de bits: 48
Compressão:
0000 1111 0100 0101 0000 1111 0100 0100 0000
Número de bits: 36
---------------------------------------------
```

Passo a passo do algoritmo.

1. Procura, no arquivo de áudio, o *sample* padrão (no exemplo, 0100). Enquanto não encontrar, simplesmente copia para o arquivo de áudio comprimido os samples que ele encontrar.

   som = 0000 0100 0100 0100 0100 0100 0000 0100 0100 0100 0100 0000
   cmp = 0000

2. Ao encontrar o *sample* padrão (0100), ele insere o sinal de início de repetição no arquivo cmp (*sample* com todos os bits 1s), seguido do *sample* padrão.

   som = 0000 0100 0100 0100 0100 0100 0000 0100 0100 0100 0100 0000
   cmp = 0000 1111 0100

3. Agora, o algoritmo irá contar quantas vezes o *sample* padrão se repete no arquivo original (no caso, cinco vezes), e vai transformar este número para binário, armazenando o resultado no arquivo comprimido. 5d = 0101b

   som = 0000 0100 0100 0100 0100 0100 0000 0100 0100 0100 0100 0000
   cmp = 0000 1111 0100 0101

4. O algoritmo retorna ao passo 1; procura o *sample* padrão (0100); enquanto não encontra, copia para o arquivo comprimido.

```
som = 0000 0100 0100 0100 0100 0100 0000 0100 0100 0100 0100 0000
cmp = 0000 1111 0100 0101 0000
```

5. O algoritmo retorna ao passo 2; ao encontrar o *sample* padrão, insere o sinal de início de repetição no arquivo cmp (*sample* com todos os bits 1s), seguido do *sample* padrão.

```
som = 0000 0100 0100 0100 0100 0100 0000 0100 0100 0100 0100 0000
cmp = 0000 1111 0100 0101 0000 1111 0100
```

6. O algoritmo retorna ao passo 3; conta quantas vezes o *sample* padrão se repete no arquivo original (no caso, quatro vezes), e vai transformar este número para binário, armazenando o resultado no arquivo comprimido. 4d = 0100b.

```
som = 0000 0100 0100 0100 0100 0100 0000 0100 0100 0100 0100 0000
cmp = 0000 1111 0100 0101 0000 1111 0100 0100
```

7. O algoritmo retorna ao passo 1; procura o *sample* padrão (0100); enquanto não encontra, copia para o arquivo comprimido.

```
som = 0000 0100 0100 0100 0100 0100 0000 0100 0100 0100 0100 0000
cmp = 0000 1111 0100 0101 0000 1111 0100 0100 0000
```

8. O algoritmo termina após processar o último *sample*, apresentando na tela os resultados.

COMANDO: audioTroca - Troca com Perdas

```
somPerda = audioTroca(som,"0000","0100");
```

Cria um novo arquivo de áudio a partir do arquivo de áudio enviado (som), onde será buscado o primeiro padrão ("0000"), que será substituído pelo segundo padrão ("0001"). Isso é realizado antes do algoritmo de compressão sem perdas, para aumentar a compressão do áudio. Note que, neste caso, ao se utilizar os dois comandos, o resultado final é um arquivo de áudio comprimido COM PERDAS.

Por exemplo, vamos analisar o arquivo de áudio abaixo:

```
som = ("0000 0100 0100 0100 0100 0100 0110 0100 0100 0100 0100 0000");
```

Ele é formado pelo *sample* 0000, depois o *sample* 0100 repetido 5x, depois o *sample* 0110, depois o *sample* 0100 repetido mais 4x e, finalmente, o *sample* 0000. Se utilizarmos o comando de compressão, obteríamos o seguinte resultado:

```
som = ("0000 0100 0100 0100 0100 0100 0110 0100 0100 0100 0100 0000");
somComprimido = audioComprime(som,"0100");

---------------------------------------------
Compressão sem perdas: samples iguais
Original:
0000 0100 0100 0100 0100 0100 0110 0100 0100 0100 0100 0000
Número de bits: 48
Compressão:
0000 1111 0100 0101 0110 1111 0100 0100 0000
Número de bits: 36
---------------------------------------------
```

O algoritmo iria comprimir as duas ocorrências de repetição do *sample* 0100, obtendo uma diminuição de 48 bits para 36 bits (25% de melhoria). No entanto, se fosse necessária uma estratégia de diminuição mais drástica, é preciso observar o arquivo original com mais cuidado:

```
som = ("0000 0100 0100 0100 0100 0100 0110 0100 0100 0100 0100 0000");
```

Como é possível perceber, temos cinco *samples* iguais, depois um único *sample* diferente e, depois, novamente quatro *samples* iguais. Estes nove *samples* iguais (0100) não podem ser comprimidos de uma vez só por causa do *sample* no meio (0110). O comando audioTroca faz uma modificação no arquivo de áudio (com perda), para permitir uma compressão mais severa.

```
som = ("0000 0100 0100 0100 0100 0100 0110 0100 0100 0100 0100 0000");
somPerda = audioTroca(som,"0110","0100");

---------------------------------------------
Compressão com perdas: troca de samples
Original:
0000 0100 0100 0100 0100 0100 0110 0100 0100 0100 0100 0000
Trocar: 0110 --> 0100
Novo audio:
0000 0100 0100 0100 0100 0100 0100 0100 0100 0100 0100 0000
---------------------------------------------
```

Desta forma, o novo áudio (armazenado em somPerda), possui 10 *samples* iguais (0100) em sequência, pois o *sample* 0110 foi trocado pelo 0100. Agora, é possível realizar a compressão e verificar o resultado.

```
som = ("0000 0100 0100 0100 0100 0100 0110 0100 0100 0100 0100 0000");
somPerda = audioTroca(som,"0110","0100");
somComprimido = audioComprime(somPerda ,"0100");

---------------------------------------------
Compressão sem perdas: samples iguais
Original:
0000 0100 0100 0100 0100 0100 0100 0100 0100 0100 0100 0000
Número de bits: 48
Compressão:
0000 1111 0100 1010 0000
Número de bits: 20
---------------------------------------------
```

Primeiro, é importante observar que o novo arquivo de áudio (somComprimido), é criado a partir de somPerda (ou seja, o arquivo de áudio original *já com a troca*). Após a troca, temos uma sequência de 10 *samples* 0100 e a compressão é de 48 bits para 20 (58% de compressão).

7. Manipulação de Vídeo

A manipulação de vídeos no Simple Computer é realizada através de um formato simplificado de vídeo, chamado simpleFrame. No simpleFrame, cada frame é formado por um conjunto de 25 pixels divido em 5 linhas e 5 colunas, onde cada pixel é representado por um único *bit*, que pode valer 0 ou 1. Neste caso, um frame neste formato teria a seguinte apresentação:

```
0 0 0 0 0
1 1 1 1 0
1 0 0 1 0
1 0 1 0 0
1 1 0 1 1
```

Um vídeo no formato simpleFrame, com três frames, teria a seguinte apresentação:

```
0 0 0 0 0   0 1 0 0 1   0 1 1 0 1
1 1 1 1 0   1 0 1 0 1   0 1 1 1 1
1 0 0 1 0   0 1 0 1 1   1 1 0 1 0
1 0 1 0 0    0 1 1 0 1  1 0 0 0 0
1 1 0 1 1   0 1 0 1 0   1 1 0 1 0
```

Assim como na manipulação de áudio, o frame formado apenas por números 1s não pode ser utilizado, pois ele representa o início da compressão.

Os arquivos no formato simpleFrame são arquivos textuais, no formato csv, que devem estar previamente carregados no servidor. O formato do arquivo possui duas linhas. A primeira é o cabeçalho. Ele informa, através de uma sequência de caracteres, o número de frames que o vídeo possui, sendo que cada frame precisa ser representado por cinco caracteres (1 caractere para cada linha). Desta forma, o cabeçalho para um vídeo com dois frames teria o seguinte formato:

```
a, b, c, d, e, a, b, c, d, e
```

Este cabeçalho informa ao SimpleComputer que estamos trabalhando com um vídeo com dois frames, cada frame com cinco linhas (a, b, c, d, e). O número de linhas e colunas (5) é fixo e obrigatório.

A próxima linha deve conter os dados, separados por vírgulas, dos valores de cada linha (5 *bits* por linha). Para o exemplo acima, uma possível configuração seria a seguinte (cores inseridas somente para melhor visualização):

```
00000, 10010, 10001, 00000, 00000, 11111, 10111, 01111, 11101, 01111
```

Este conjunto de dados representaria o seguinte vídeo:

```
0 0 0 0 0   1 1 1 1 1
1 0 0 1 0   1 0 1 1 1
1 0 0 0 1   0 1 1 1 1
0 0 0 0 0   1 1 1 0 1
0 0 0 0 0   0 1 1 1 1
```

COMANDO: new SimpleVideo - Carregar arquivo de vídeo

```
video = new SimpleVideo("video.csv");
```

Neste caso, é criada uma variável chamada vídeo, que armazena os frames que estão no arquivo vídeo.csv.

COMANDO: printFrames – Imprimir frames

```
printFrames( video );
```

O comando printFrames simplesmente imprime na tela os frames do vídeo carregado, mostrando seus *bits* em uma tabela para melhor visualização. Ao final, ele informa o número de bits do arquivo.

```
video = new SimpleVideo("video.csv");
printFrames( video );
```

00000	00000	00000	00000	00000	00000	00100	00001	00000	00000	00000	00000	00000	00000
11110	11110	11110	11110	11111	01110	11110	01110	01110	01110	01110	01110	01110	01110
10010	10010	10010	10010	11110	01111	01110	01110	01110	01110	01110	01110	01110	01110
10100	10100	10100	10100	11100	01110	11110	01111	01110	01110	01110	01110	01110	01110
11011	11011	11011	11011	11011	00000	00000	00000	10000	10000	10000	10000	10000	10000

```
Numero de bits: 350
```

COMANDO: playVideo – Reproduzir vídeo

```
playVideo( video );
```

O comando playVideo toca o vídeo binário. Somente podem ser tocados vídeos que não estão comprimidos.

```
video = new SimpleVideo("video4.csv");
playVideo( video );
```

COMANDO: comprimeVideo – Comprimir vídeo

```
novoVideo = comprimeVideo(video,0,0,0,4,4);
```

O comando comprimeVideo comprime o arquivo de vídeo utilizando a técnica inter-frame, retornando um novo arquivo. A compressão se dá pela procura do padrão informado e a eliminação do mesmo nos frames subsequentes. O comando retorna o novo vídeo e precisa de seis parâmetros: vídeo a ser comprimido, número do frame do padrão, linha 1, coluna 1, linha 2, coluna 2.

Vamos dar uma olhada no seguinte exemplo:

00000 11110 10010 10100 11011	00000 11110 10010 10100 11011	00000 11110 10010 10100 11011	00000 11110 10010 10100 11011	00000 11111 11110 11100 11011	00000 01110 01111 01110 00000	00100 11110 01110 11110 00000	00001 01110 01110 01111 00000	00000 01110 01110 01110 10000	00000 01110 01110 01110 10000

Numero de bits: 250

O número do frame inicia em 0 e segue até o fim; no exemplo abaixo, colocamos os números para melhor visualização. Neste exemplo, é possível perceber que os frames 0, 1, 2, 3 e 4 são iguais; logo, é possível realizar uma compressão sem perdas nestes frames.

00000 **11110** **10010** **10100** **11011**	**00000** **11110** **10010** **10100** **11011**	**00000** **11110** **10010** **10100** **11011**	**00000** **11110** **10010** **10100** **11011**	00000 11111 11110 11100 11011	00000 01110 01111 01110 00000	00100 11110 01110 11110 00000	00001 01110 01110 01111 00000	00000 01110 01110 01110 10000	00000 11110 10010 10100 11011
0	1	2	3	4	5	6	7	8	9

Primeiro, é necessário descobrir em qual linha e qual coluna inicia e termina o padrão encontrado. A linha e a coluna são definidas como segue:

	0	1	2	3	4
0	0	0	0	0	0
1	1	1	1	1	0
2	1	0	0	1	0
3	1	0	1	0	0
4	1	1	0	1	1

Neste caso, como os frames são absolutamente iguais, o ponto inicial (linha 1, coluna 1) seria (0,0) e o ponto final (linha 2, coluna 2) seria (4,4).

00000 11110 10010 10100 11011	00000 11110 10010 10100 11011	00000 11110 10010 10100 11011	00000 11110 10010 10100 11011	00000 11111 11110 11100 11011	00000 01110 01111 01110 00000	00100 11110 01110 11110 00000	00001 01110 01110 01111 00000	00000 01110 01110 01110 10000	00000 11110 10010 10100 11011
0	1	2	3	4	5	6	7	8	9

```
novoVideo = comprimeVideo(vídeo,0,0,0,4,4);
```

O comando acima solicita que seja realizada a compressão do arquivo vídeo, a partir do frame 0, iniciando na linha 0 e coluna 0 e indo até a linha 4 e coluna 4. O resultado da compressão é apresentado abaixo:

11111	**00000**	.....	.....	.....	.....	00000	00000	00100	00001	00000	00000
11111	**11110**	.....	.....	.....	.....	11111	01110	11110	01110	01110	01110
11111	**10010**	.....	.....	.....	.....	11110	01111	01110	01110	01110	01110
11111	**10100**	.....	.....	.....	.....	11100	01110	11110	01111	01110	01110
11111	**11011**	.....	.....	.....	.....	11011	00000	00000	00000	10000	10000
x	x	0	1	2	3	4	5	6	7	8	9

```
Numero de bits: 200
```

Como o frame 0 já é o primeiro a ser comprimido, o novo arquivo de vídeo tem, como primeiro frame, o padrão de início da compressão (todos os valores 1s). Então, vem o padrão que será replicado (negrito) e, finalmente, os frames comprimidos. Note que, como os frames são iguais, os frames 0, 1, 2 e 3 ficam vazios. Uma economia total de 50 bits.

Vamos ver outro exemplo:

```
video = new SimpleVideo("video3.csv");
printFrames( video );
```

00101	11101	10111	00101	11101	00000	01010	01010	01111	01010	01010
01111	01111	01111	01111	01111	00000	01010	01010	01010	01010	01010
00111	10111	00111	00111	00111	00101	01110	01110	11111	01110	01110
10111	10111	00111	10111	10111	00011	01010	01010	01010	01010	01010
01111	01111	01111	01111	01111	00011	01010	01010	01010	01010	01010
0	1	2	3	4	5	6	7	8	9	10

```
Numero de bits: 275
```

Neste caso, os frames são diferentes uns dos outros, mas parte do frame é igual. Note os primeiros quatro frames:

00101	11101	10111	00101	11101	00000	01010	01010	01111	01010	01010
0**111**1	0**111**1	0**111**1	0**111**1	0**111**1	00000	01010	01010	01010	01010	01010
0**011**1	1**011**1	0**011**1	0**011**1	0**011**1	00101	01110	01110	11111	01110	01110
1**011**1	1**011**1	0**011**1	1**011**1	1**011**1	00011	01010	01010	01010	01010	01010
01111	01111	01111	01111	01111	00011	01010	01010	01010	01010	01010
0	1	2	3	4	5	6	7	8	9	10

```
   0 1 2 3 4

0  0 0 1 0 1
1  0 1 1 1 1
2  0 0 1 1 1
3  1 0 1 1 1
4  0 1 1 1 1
```

Aqui, temos o ponto inicial como (1,1) e o ponto final como (3,3).

```
novoVideo = comprimeVideo(video,0,1,1,3,3);
printFrames( novoVideo );
```

11111 11111 11111 11111 11111	..... .111. .011. .011.	00101 0...1 0...1 1...1 01111	00101 0...1 0...1 1...1 01111	00101 0...1 0...1 1...1 01111	00101 0...1 0...1 1...1 01111	00101 0...1 0...1 1...1 01111	00000 00000 00101 00011 00011	01010 01010 01110 01010 01010	01010 01010 01110 01010 01010	01111 01010 11111 01010 01010	01010 01010 01110 01010 01010	01010 01010 01110 01010 01010
x	x	0	1	2	3	4	5	6	7	8	9	10

```
Numero de bits: 264
```

O primeiro frame representa o início da compressão (todos os valores 1s). Então, vem o padrão que será replicado e, finalmente, os frames comprimidos. Como só parte do frame é igual, os frames de 0 a 4 possuem uma parte variável e a parte vazia (representada pelos pontos) é a que vai ser copiada do frame de replicação (em negrito). Como só parte do frame é replicada, a economia é menor: 9 bits.

Referências

MOTA, JOSÉ CARLOS. **Dicionário de Computação e Informática**. Rio de Janeiro: Ciência Moderna, 2010.

VELLOSO, FERNANDO DE CASTRO. **Informática**: conceitos básicos. Rio de Janeiro: Elsevier, 2017.

MARÇULA, MARCELO; BENINI FILHO, PIO ARMANDO. **Informática**: conceitos e aplicações. São Paulo : Érica, 2013.