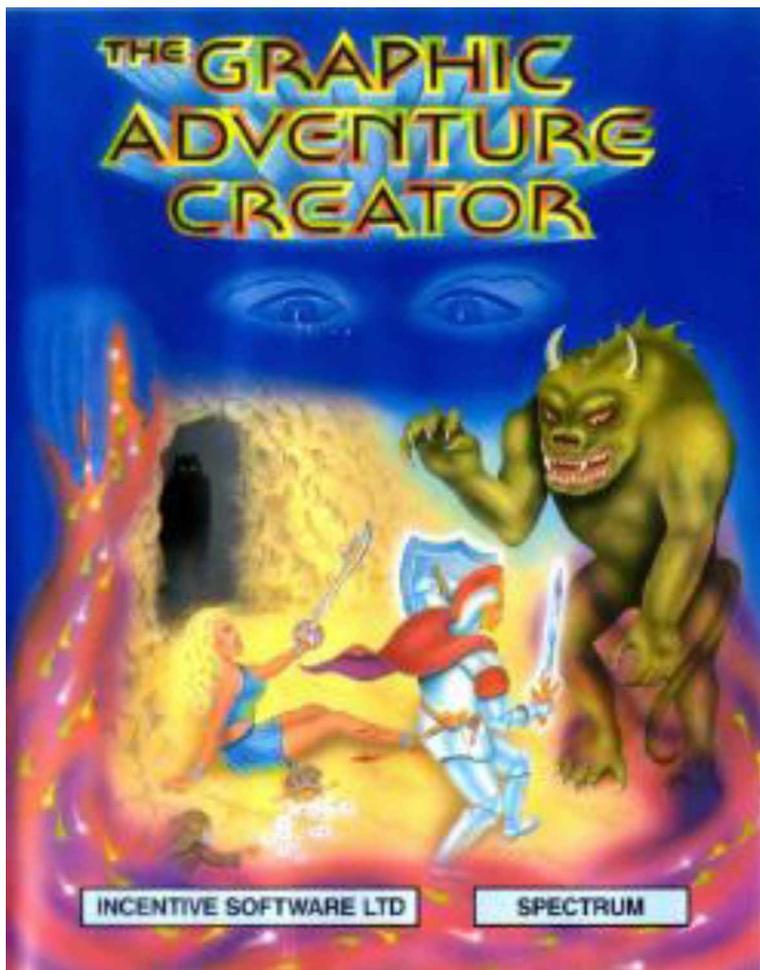


# EDITOR DE GAMES



## ÍNDICE

AGD - Arcade Game Designer .....	01
Jonathan Cauldwell v.4.7 - ptbr	
AGDx - Arcade Game Designer .....	20
Coder's Manual - ptbr	
AGD Arcade Game Designer .....	34
Traduzido ptbr do Site Funspot	
Compilador Boriel .....	55
Traduzido ptbr do Site Funspot	
PGD - Platform Game Designer .....	59
Jonathan Cauldwell	
CGD - Classic Game Designer .....	66
Dave Hughes - ptbr	
GAC - Graphic Adventure Creator .....	78
Site <a href="http://www.TK90X.com.br">www.TK90X.com.br</a>	
GAC - Graphic Adventure Creator .....	86
Original em Ingles	

# Arcade Game Designer

Copyright Jonathan Cauldwell, 2008-2017

## Instruções para a versão 4.7

Como o PGD e o SEUD, o Arcade Game Designer é uma ferramenta para escrever os seus próprios jogos de arcada simples. Embora seja um pouco mais complexo do que esses 2 utilitários e não o ajudará a criar shoot-em-ups de rolagem, ele deve ser capaz de produzir uma ampla variedade de assuntos de arcade com o mínimo de confusão, contanto que você seja paciente e experiente, preparado para ficar nele. Além disso, os jogos produzidos usando o utilitário são independentes e podem ser distribuídos gratuitamente.

A AGD já foi bem utilizada por vários autores e é responsável por muitos jogos. Alguns exemplos são Apulija-13, Hedgehogs, Kyd Cadet, Land of Mire Mare e Trooper: ponto 5. Todos estes são completamente recomendados e podem ser encontrados nos arquivos do World of Spectrum se você realmente quiser ver o que versões anteriores do AGD podem fazer.

Isso não é tudo, no entanto, AGD agora vem com explosões, trilhas mágicas de poeira/vapor, sprites maiores, lasers, rolagem de mensagens de texto e uma facilidade para colorir sprites, dando ao designer de jogos a habilidade de criar jogos que são mais profissionais do que nunca. É facilmente capaz de produzir jogos para o covertape ou padrões de software de orçamento de tempos passados.

Agora às más notícias. O AGD tem sua própria linguagem integrada, um caso bastante simplista com funções simples e algumas variáveis embutidas que você terá que pegar se desejar adicionar seus pequenos detalhes, lógica do jogo ou desviar-se do construtor, em modelos. Na verdade, o léxico não é grande, é principalmente uma questão de entender como usar cada comando ou função em particular. É principalmente muito direto. Enquanto o programa informará se ele encontrar uma palavra que não entende, a verificação de sintaxe é inexistente. No entanto, o AGD 4 moverá o cursor diretamente para a linha em questão, para que você, pelo menos, saiba onde está o problema.

Porque eu queria produzir um designer de jogos para todos usarem, o programa é gratuito para baixar do meu site. Os direitos autorais, no entanto, permanecem com o autor. Doações não são necessárias, por isso, se você realmente precisa mostrar seu apreço, compre uma cópia do SEUD ou PGD em [www.cronosoft.co.uk](http://www.cronosoft.co.uk) ou algo assim, se isso falhar, os planos estão em andamento para lançar uma versão AGD da fita no rótulo Cronosoft em um futuro próximo.

Embora AGD 4 é a versão mais robusta lançada até o momento, o programa vem como você o encontra, sem garantias de que ele é totalmente livre de bugs. Por favor, não espere suporte por e-mail - terei prazer em responder a perguntas e ouvir solicitações de melhorias e correções de bugs nos fóruns do AGD. De tempos em tempos, recebo atualizações, dependendo de outros projetos e da vida em geral. Até lá, salve seu trabalho sempre que possível e, de preferência, como um instantâneo, se estiver usando um emulador.

Mais importante de tudo, divirta-se. Enquanto isso, vou a Ripley para me embebedar.

Jonathan Cauldwell, 4th April 2008.

@zxpectrumdev  
<http://www.spanglefish.com/egghead/>

## Construindo um Game

AGD requer um pouco de troca de fitas para salvar e carregar, por isso, se você estiver usando um emulador, é uma boa ideia desligar a opção de carregar automaticamente os arquivos de fita assim que forem inseridos. Não fazer isso pode resultar na perda de qualquer trabalho que você tenha feito em um jogo caso tente alterar a fita.

Em SPIN, clique em Ferramentas e selecione Opções no menu. Selecione Arquivos na lista à esquerda e, em seguida, clique na guia "Fitas" na parte superior. Certifique-se de que não haja uma marca na caixa ao lado de Carregar fitas automaticamente. Agora você está pronto para começar a desenvolver com o AGD.

Basicamente, os jogos são uma combinação de diferentes elementos:

- \* blocos de caracteres são combinados para formar telas
- \* sprites (incluindo o jogador) são posicionados em suas posições iniciais em cada tela, e atribuída uma característica de tipo
- \* lógica e então escrita para cada tipo de sprite - jogador, inimigos, objetos etc.

Depois disso, é uma questão de adicionar mais alguma coisa que você queira.

## Main Menu

Permite selecionar uma função do jogo que você deseja projetar. A quantidade de memória restante disponível para seu aplicativo é exibida no canto superior direito da tela.

## Window area

Você pode variar o tamanho e a posição da área de reprodução na janela / opção de rolagem. As teclas do cursor permitem posicionar a janela em qualquer lugar da tela.

Alterar o tamanho da janela e/ou a direção de rolagem pode alterar a quantidade de dados necessária para cada coluna ou linha de blocos em cada tela e, portanto, invalidar os níveis atuais, se este for o caso, o programa perguntará se você deseja destruir os dados do mapa antes de prosseguir. É uma boa ideia selecionar primeiro o tamanho da área de jogo e ficar com ela.

- 1 = janela estreita.
- 2 = alargar janela.
- Q = encurtar a janela.
- A = alongar janela.

## Palette

Desde a versão 3, a AGD tem suportado espectros e emuladores habilitados para ULApplus por padrão. Se você estiver usando a versão 0.7s do SPIN, poderá ativar a emulação do ULApplus selecionando Opções no menu Ferramentas (ou apenas pressionando F8) e, em seguida, selecionando a opção Exibir à esquerda. Se você selecionar a guia Emulação na parte superior, deverá haver uma caixa de seleção para 64 ULA aprimorados por cor. Certifique-se de que está selecionado e clique no botão OK na parte inferior.

O editor de paleta ULA+ permite que você mova as 64 cores das quatro tabelas de consulta de cores (CLUTs), modificando-as. O valor RGB da cor selecionada é mostrado no canto inferior direito da tela.

- R = muda o componente vermelho da cor.
- G = muda o componente verde da cor.
- B = muda o componente verde da cor.
- ENTER = retornar ao menu principal.

## Keys

Isso permite que você defina as chaves que seu jogo usará. São permitidas 7 chaves e o joystick Kempston é mapeado nos cinco primeiros. A ação Kempston equivalente é mostrada na tela quando você é solicitado para cada chave. AGD padrão para o teclado, mas se você está tecnicamente ocupado, você pode querer usar o joystick Kempston dentro do seu jogo via POKE 32005,1 ou joystick Sinclair com POKE 32005,2. Para teclado, POKE 32005 com qualquer outro valor.

## Character Block Design

Blocos são usados para construir as telas. Cada bloco tem 8x8 pixels de tamanho e tem um tipo diferente atribuído a ele. Existem vários tipos diferentes, cada um com seu próprio conjunto diferente de atributos. Esses são:

EMPTY SPACE (ESPAÇO VAZIO) - o jogador se move livremente através de blocos de espaço livre.

NORMAL PLATFORM (PLATAFORMA NORMAL) - o jogador se move livremente pelas plataformas da esquerda, direita ou abaixo.

SOLID WALL (PAREDE SÓLIDA) - um bloco que é intransponível de qualquer direção.

LADDER - o jogador se move livremente através de blocos de escada em qualquer direção.

FODDER - como parede sólida, pode ser removido com certas funções.

DEADLY - sprites passam por isso, testes de função DEADLY para contato.

CUSTOM - sprites passam por isso, testes de função CUSTOM para contato.

Mova o cursor pelo personagem com as teclas do cursor. Use **SPACE** ou **0** para definir / desmarcar um pixel. **ENTER** retorna ao menu principal. Você pode definir as cores com **P**, **I** e **B**. Como bits brilhantes e Flash podem ser definidos ou não definidos, então se você pretende que seu jogo seja executado em Spectrum modificados com ULA+ ou emuladores, todos os 4 CLUTS estarão disponíveis.

**Q** = mover para a esquerda através da lista de propriedades do bloco.

**W** = mover para a direita através da lista de propriedades do bloco.

**L** = último bloco.

**N** = próximo bloco.

**P** = cor do papel.

**I** = cor da tinta.

**B** = alterar o brilho e os bits de flash.

**M** = copiar o bloco atual para a área de transferência.

**K** = colar o bloco da área de transferência.

**C** = limpar o bloco atual.

**X** = cria um novo bloco de caracteres.

**D** = excluir o bloco de caracteres atual.

**ENTER** = retornar ao menu principal.

## Screen Layout

Mova o cursor pela tela com as teclas do cursor. Use **1** e **2** para selecionar o bloco de caracteres que você deseja colocar na posição atual do cursor. **ESPAÇO** ou **0** coloca o bloco desejado na tela atual nessa posição. Alternativamente, você pode querer usar o modo de desenho rápido, pressionando **F**. Isso coloca automaticamente o bloco selecionado na posição atual do cursor toda vez que você mover o cursor. Para sair do modo de desenho rápido e retornar para a colocação manual de blocos, pressione a tecla **F** novamente. Você pode copiar uma tela. Pressione **M** para marcar a tela que você deseja copiar, então vá para a tela que você deseja copiar e pressione **K** para colar, **ENTER** retorna ao menu principal.

**1** = mover para a esquerda através da tabela de blocos.

**2** = mover para a direita através da tabela de blocos.

**F** = alternar modo de desenho rápido.

**M** = marcar a tela atual.

**K** = colar da tela marcada.

N = próxima tela.  
P = tela anterior.  
X = criar uma nova tela.  
D = excluir tela atual.  
ENTER = retornar ao menu principal.

## **Sprite Images**

Sprites são imagens de 16x16 pixels que compõem partes móveis como o jogador, as balas dos jogadores, as naves inimigas, etc. Sprites podem ter qualquer número de quadros.

Movimente-se pela grade com o cursor, fixando e excluindo pixels com **SPACE** ou **0**. Como a detecção de colisão do AGD é baseada em coordenadas, é uma boa idéia preencher o máximo possível da área de 16x16, especialmente para sprites de jogadores e inimigos. Dê uma olhada no Monty Mole, Egghead ou BLOB da Starquake como exemplos de sprites que preenchem uma área de 16x16. Colecionáveis não importam tanto, os jogadores raramente notam ou se importam se pegam um colecionável de 3 pixels de distância.

O editor de sprites também possui uma tela de rascunho, você pode carregar arquivos SCREEN\$ da fita usando a tecla L. você pode então mover um cursor de 16x16 pixels ao redor desta tela, então pressione G para pegar a imagem da tela naquele ponto e copiá-la para a área de transferência, isso pode ser colado em um quadro de sprite com a tecla K.

Esteja avisado, alguns emuladores carregam automaticamente arquivos de fita assim que você insere uma fita, o que removerá o AGD da memória junto com o jogo que você estava criando. Certifique-se de ter desativado esta opção no seu emulador primeiro.

X = inserir Sprite.  
D = excluir Sprite.  
C = grade de Sprite claro.  
G = pegue o sprite da tela do bloco de notas e copie para a área de transferência.  
L = carrega a tela do bloco de notas.  
M = copiar sprite ou tile para a área de transferência.  
K = copiar sprite ou tile da área de transferência.  
H = inverte o sprite horizontalmente.  
V = inverte o sprite vertical mente.  
N = próximo sprite.  
P = sprite anterior.  
I = inserir quadro.  
R = remover quadro.  
F = próximo quadro.

## **Sprite Positions**

Isso permite que você posicione sprites em suas posições iniciais para cada nível usando as teclas do cursor. 8 tipos de sprites estão disponíveis, e sprites podem ser configurados como qualquer um destes, permitindo que você posicione bônus, sprites inimigos, a posição inicial do sprite do jogador ou qualquer outro tipo de sprite que você tenha configurado. As imagens do sprite são separadas dos tipos de sprite. Imagens são apenas as imagens que você desenha no editor de sprites, enquanto o comportamento de um sprite é definido por seu tipo. O tipo 0 é geralmente usado para o sprite do jogador (ou sprites), embora você possa mudar isso. Você precisará alocar um tipo para cada item em movimento diferente, como alienígenas, marcadores ou itens colecionáveis, e determinar seus movimentos usando uma linguagem de script simples. Veja a seção sobre eventos para obter informações sobre sprites em movimento.

N = próxima tela.  
P = tela anterior.  
Q = move próximo sprite.  
I = mudo a imagem do sprite.  
T = altera o tipo de sprite.

D = excluir sprite atual da tela.  
X = adiciona novo sprite a tela.

## Objects

Distinto dos sprites, os objetos são itens estáticos que um sprite pode pegar ou possivelmente soltar, como os sprites, eles consistem em imagens de 16x16 pixels, embora tenham apenas um quadro. Estes podem ser editados da mesma maneira que as imagens de sprites.

Para cada objeto, você precisará definir sua posição no início do jogo. Ao contrário dos sprites, os objetos não reaparecem toda vez que o jogador entra em uma sala, então eles podem ser pegos em uma sala e depois jogados em outra, tornando-os ideais para jogos de aventura arcade. A tela inicial é alterada com as teclas Q e W. Uma tela inicial de 255 indica que o objeto inicia o jogo no inventário do jogador. Objetos que estão faltando no início do jogo, ou seja, não aparecem em nenhuma tela ou no inventário do jogador, devem ser atribuídos uma tela inicial de 254. Pressione P para colocar um objeto em sua posição designada em sua tela inicial.

X = inserir objeto.  
D = excluir objeto.  
C = grade de imagem clara.  
M = copiar imagem ou lado a lado da área de transferência.  
K = copiar imagem ou bloco da área de transferência  
N = próximo objeto.  
L = último objeto.  
Q = alterar a tela inicial.  
W = altera a tela inicial.  
P = posicionar objeto na tela inicial.

Objetos podem ser manipulados no código de eventos com GOT, GET, PUT e DETECTOBJ. Além disso, a variável OBJ armazena o resultado do último comando DETECTOBJ, embora você possa usá-lo também para outras coisas, se necessário. IF GOT n será verdadeiro se o jogador tiver o objeto n em seu inventário. GET n colocará o objeto n no inventário do jogador (independentemente de onde o objeto estiver atualmente). PUT n derrubará um objeto na posição atual do sprite e DETECTOBJ verificará se o sprite atual está tocando um objeto, colocando o resultado na variável OBI. Se nenhum objeto for detectado na posição do sprite, o valor 255 será retornado.

Por exemplo, o código a seguir selecionará automaticamente qualquer objeto pelo qual um sprite passe:

```
DETECTOBJ  
IF OBJ <> 255  
    GET OBJ  
ENDIF
```

Como GET e PUT não verificam se o objeto já está no inventário do jogador, eles podem ser usados para colocar objetos na tela ou removê-los mais ou menos à vontade. Você pode desejar que os sprites inimigos abram bônus, depois os remova após alguns segundos. Só porque você remove um objeto da tela não significa que você tenha que conceder os pontos ou bônus que o jogador teria ganhado ao coletá-lo.

## Map Layout

AGD permitirá níveis sequenciais, mas também tem a capacidade de criar jogos exploradores onde o jogador pode explorar um mapa.

O "mapa" do seu jogo é organizado como uma grade de 10 x 8 locais, e todos começam vazios. Locais vazios aparecem como dois hifens "-", os quartos aparecem como os números da tela, por exemplo, "01" ou "12". Como cada quarto é projetado no designer de tela, ele pode ser colocado nessa grade em um local escolhido. Para se deslocar pelo mapa, basta

utilizar os comandos **SCREENUP**, **SCREENDOWN**, **SCREENLEFT**, **SCREENRIGHT**. Durante o jogo, não será possível ao jogador mover-se para um espaço de grade vazio. Comandos como **NEXTLEVEL** ou **LET SCREEN = 9** não alteram a posição atual do mapa; portanto, são mais bem utilizados em jogos com níveis sequenciais, a menos que você esteja confiante no que está fazendo.

O cursor vermelho pode ser movido ao redor da grade usando as teclas de cursor, para mudar a sala em um determinado local da grade mova as teclas '1' e '2'. Os quartos são exibidos nos dois terços inferiores da tela conforme são selecionados para facilitar o processo.

Seu mapa seguirá as regras da geometria euclidiana, mas você pode dobrar as regras para criar um campo de jogo distorcido, caso deseje. Mover-se para a esquerda a partir de uma sala situada na borda esquerda do mapa fará com que o jogador reapareça na sala colocada na borda direita da próxima linha, se alguém estiver lá. Da mesma forma, mover-se diretamente de uma sala à direita levará o jogador para a sala na extremidade esquerda da sala, uma linha abaixo. Se você não quiser que isso aconteça, você deve construir paredes nas telas relevantes para formar uma barreira física. Também é possível reutilizar uma sala, para que ela apareça mais de uma vez em seu mapa.

Mover-se de uma tela para outra não alterará as coordenadas do sprite do jogador, portanto, elas devem ser definidas manualmente, ao mesmo tempo em que as opções **SCREENLEFT**, **SCREENRIGHT** etc. são executadas. Qualquer sprite de jogador configurado para uma tela será usado apenas na primeira tela, ou para gerar um sprite de novo jogador caso ele morra na tela. Se o jogador morrer em uma tela onde ele não tenha uma posição padrão, ele não será reaberto.

Pressione 'X' para declarar a localização da grade como o ponto no qual o jogador deve começar o jogo.

1 = selecionar sala anterior da lista.

2 = selecione a próxima sala da lista.

X = selecione a sala onde o jogador começa o jogo.

ENTER = retornar ao menu principal.

## Jump Table

Se o seu jogo faz uso da gravidade, você pode querer editar a tabela de salto. Isso permite que você edite a inclinação de saltos e / ou quedas, permitindo que você determine como os sprites altos podem pular, ou com que rapidez eles descem quando caem através de falhas no chão.

A tabela de salto é separada em uma série de etapas individuais e você pode alterar a distância entre essas etapas com as teclas do cursor. A coluna vermelha representa a etapa que você está editando no momento. Quando estiver satisfeito com sua tabela de saltos, pressione **ENTER** para retornar ao menu principal.

## Sound

H = ouve som presente.

N = som seguinte.

P = som anterior.

X = criar novo som.

D = apagar som.

Movimente os valores com as teclas do cursor, aumentando e diminuindo-os com 1 e 2. Use 3 e 4 para aumentar e diminuir o tom em incrementos de 50. **SPACE** ou **0** desativará o ruído ou o tom.

Para reproduzir um som no seu jogo, use o comando **SOUND** no evento relevante.

Os sons são reproduzidos usando o chip AY. Para efeitos de bipe de 48k, use o comando

BEEP.

## Save Game

Solicita um nome de arquivo para o seu jogo e, em seguida, pergunta se você deseja iniciar o código com um programa de carregador BASIC que carrega e executa automaticamente o seu jogo (e carregando a tela se você configurou um no menu miscelânea), caso deseje execute-o independentemente do utilitário. O jogo é salvo em fita com ou sem o carregador BASIC. Se você salvar o jogo com um carregador, ele retornará ao início quando terminar, caso contrário, retornará ao BASIC para que você possa escrever suas próprias rotinas de final de jogo.

O arquivo de código salvo pode ser carregado posteriormente e editado. Você pode até querer escrever seu próprio carregador BASIC, o programa mais simples para fazer isso seria:

```
10 CLEAR 31231: LOAD ""CODE: RANDOMIZE USR 32000
```

You would save this BASIC program to tape with something like

```
SAVE "MyGame" LINE 10
```

Claro, você pode querer configurar algumas outras coisas primeiro, como as cores BORDER, PAPER e INK, além de uma página de título e talvez até mesmo uma borda ao redor do painel de status.

## Load Game

Carrega um novo jogo da fita. Se você estiver usando um emulador, terá que operar o navegador de fitas por conta própria. Emuladores sem navegador de fita e aqueles que carregam automaticamente as fitas inseridas não são recomendados para o AGD. O AGD só carregará arquivos de código que foram criados com o utilitário.

## Test Game

Permite que você teste sua criação. Pressione ENTER em qualquer ponto para retornar ao editor.

## Miscellaneous

Fornecer mais opções para modificar o mecanismo de jogo e mostra o endereço do primeiro local de memória não utilizado disponível, caso deseje marcar suas próprias rotinas no final do jogo.

### **Collision Distance**

A distância de colisão é o ajustador de detecção de colisão de sprite. Sprites padrão são 16x16 pixels, então normalmente eles vão se acertar se suas coordenadas estiverem separadas por menos de 16 pixels. Para um sprite robusto como Monty Mole ou Egghead, este é o ideal. No entanto, você pode querer desenhar um sprite jogador que é um pouco mais magro, deixando espaços em torno de ambos os lados. Você não vai querer contar a diferença como parte da detecção de colisão de sprite, então você pode reduzir essa distância para uma contagem menor de pixels. Lembre-se de que essa configuração afeta apenas as distâncias horizontais e define todas as detecções de colisão entre os sprites.

### **Sprite Height**

Enquanto todos os sprites devem ter o mesmo tamanho, agora você tem a opção de 16x16 ou 16x24 para o seu jogo. No entanto, esteja ciente de que sprites 16x24 ocupam 50% mais memória. Você será solicitado a confirmar que deseja alterar o tamanho do sprite e, em seguida, o AGD converterá o mecanismo, permitindo a memória. Quaisquer sprites existentes são convertidos para o novo tamanho. Você deve editá-los manualmente no editor de sprites.

### **SPRITEINK Mask**

O comando SPRITEINK normalmente permite que você altere a cor INK de um sprite sem afetar sua configuração BRIGHT ou FLASH. Se você quer que o SPRITEINK faça isso, ou está projetando o seu jogo com o ULApplus em mente, você precisará mudar esta configuração. Um valor de 7 muda a INK, 71 muda de INK e BRIGHT, 135 muda de INK e FLASH e 199 muda de INK, BRIGHT e FLASH. Para o ULApplus, um valor de 199 permite que você selecione qualquer uma das quatro tabelas de consulta de cores. Para ligar o BRIGHT, adicione 64 ao valor SPRITEINK da mesma forma que faria com a função BASIC ATTR. Para ativar o FLASH, adicione 128.

### **Loading Screen**

Esse recurso permite que você importe uma tela de carregamento criada com outro utilitário. AGD primeiro solicita que você confirme que deseja carregar uma tela de carregamento e, em seguida, carrega uma tela.

### **Order Sprites**

Quando os sprites de ordem são ativados, a tabela de sprites é ordenada e a ordem na qual os sprites são desenhados depende de sua posição vertical para eliminar o flicker. Quando isso é desativado, a ordem em que os sprites são desenhados não varia de um quadro para outro.

### **Specialise**

Esta opção seleciona qual das três adaptações de especialista você deseja usar para o seu jogo. As adaptações disponíveis são partículas, efeitos e quebra-cabeças. O mecanismo de partículas permite comandos TRAIL, EXPLODE e LASER. É ótimo para trilhas de vapor / poeira mágica, explosões, lasers e, geralmente, para dar aos seus shoot-em-ups um pouco mais de polimento. Efeitos dá aos comandos FADE e TICKER a atenuação da janela e da rolagem de pixel em uma mensagem em um ponto especificado na tela - útil para rolar mensagens em suas intro ou atualizações de mensagens de texto no jogo. Quebra-cabeça habilita o comando INV permitindo que o jogador selecione itens em seu inventário e habilita blocos de água. Ele também lembra de quaisquer blocos posicionados em uma tela usando o comando PUTBLOCK e redesenha-os quando o player sai e, em seguida, entra novamente na tela.

### **AY Sounds**

Esta opção pode ser usada para desabilitar os efeitos sonoros AY, caso deseje configurar uma rotina de música AY com interrupções.

## **Messages**

É aqui que você pode definir as mensagens de texto que seu jogo usará. Pressione ENTER a qualquer momento para retornar ao menu principal.

N = próxima mensagem.

P = mensagem anterior.

SPACE ou 0 = editar mensagem atual.

X = criar nova mensagem.

D = excluir mensagem.

## **Events**

Essa é a parte em que você pode falar sobre a lógica do jogo e mudar a maneira como ela funciona de várias formas diferentes. Embora o editor e o compilador não rivalizem com uma linguagem apropriada como o BASIC, o AGD fornece um número limitado de declarações, funções e variáveis que devem permitir a criação de uma variedade de diferentes jogos de arcada ou aventuras de arcada. Pense nisso como uma versão arcade do GAC da Incentive Software - você pode precisar ser criativo sobre como implementar os recursos que deseja, mas isso é metade da diversão.

Existem vários eventos para os quais você pode escrever a lógica. Além daqueles que ocorrem em determinados momentos do jogo, há eventos associados a 8 tipos de sprites. Estes são os eventos que controlam o movimento e a lógica de cada tipo de sprite. Sprite

tipo 0 é geralmente reservado para o sprite do jogador, então este código deve testar as chaves e mover o sprite de acordo. O resto é todo seu para fazer o que quiser. Por exemplo, você poderia escolher fazer sprites tipos 1 e 2 diferentes alienígenas com diferentes padrões de movimento, e talvez usar sprite tipo 3 para sprites de bônus que o jogador pega.

Controle do jogador (tipo 0) - movimento do jogador, leitura de chaves, detecção de colisão

Sprite tipo 1	- comportamento de sprites com tipo 1
Sprite tipo 2	- comportamento de sprites com tipo 2
Sprite tipo 3	- comportamento de sprites com tipo 3
Sprite tipo 4	- comportamento de sprites com tipo 4
Sprite tipo 5	- comportamento de sprites com tipo 5
Sprite tipo 6	- comportamento de sprites com tipo 6
Sprite tipo 7	- comportamento de sprites com tipo 7
Sprite tipo 8	- comportamento de sprites com tipo 8
Inicializar sprite	- sempre que um sprite é inicializado
Loop principal 1	- todo loop de jogo
Loop principal 2	- todo loop de jogo
Intro /menu	- antes do jogo começar
Iniciar o jogo	- no início do jogo, configure as variáveis
Reinicie a tela	- o que acontece quando o jogador reinicia uma tela
Caiu muito longe	- quando qualquer sprite chega ao final da mesa de pulo
Kill player	- acontece quando o jogador perde uma vida com um comando KILL
Jogo perdido	- quando o jogador perde sua última vida
Jogo concluído	- quando o jogo foi concluído com sucesso, por exemplo, parabéns

Para modificar os eventos, use o cursor para cima / baixo para selecionar o evento e, em seguida, pressione espaço ou 0. Se nenhum código tiver sido escrito para esse evento, você será apresentado a um menu adicional de modelos básicos de código para iniciá-lo. Você não precisa usar um modelo, esses são apenas exemplos pré-escritos de código para executar diferentes funções. Quando você seleciona um modelo, o editor copia um pedaço do código no evento para você, mas não é necessário selecionar um.

As opções são:

None  
Manic Miner style controls  
Player escadas e níveis  
Cybernoid style controls  
Patrulhando o inimigo (L / R)  
Patrulhando o inimigo (U / D)  
Inimigo saltitante  
Escadas e níveis perseguidores  
Static collectable  
Bloco empurrável  
Plataforma móvel  
Minigame Bonus Stopper

A primeira opção pula os modelos e permite que você comece com um pedaço de papel em branco.

As próximas opções são os controles do jogador. Os controles de estilo Manic Miner lhe dão as teclas esquerda, direita e pular. As escadas e níveis dos jogadores usam escadas esquerda, direita e para cima e para baixo. Controles de estilo Cybernoid são deixados, à direita, fogo (apenas motor de partículas) e para cima. Quando não estiver subindo, o sprite do jogador se moverá automaticamente, como em Cybernoid.

Existem modelos de sprites inimigos também. A opção de inimigo de patrulhamento (L / R) se moverá para a esquerda e para a direita ao longo de uma plataforma, revertendo quando atingir uma parede ou se houver uma lacuna no chão. Patrulhar o inimigo (U / D) sobe e desce, revertendo sempre que atinge uma parede, os inimigos quicam apenas saltam

diagonalmente ao redor da tela até que eles atinjam algo, então mudam de direção. Escadas e níveis perseguidores caminham ao longo de plataformas e paredes, subindo e descendo escadas.

Itens colecionáveis estáticos são sprites que ficam na tela e esperam por uma colisão com o jogador, e então eles se removem e adicionam à pontuação do jogador, os blocos que podem ser empurrados reagirão às colisões com o sprite do jogador, movendo-se para fora do caminho. Eles também testam colisões com outros sprites do tipo 1 e destroem qualquer um que encontrarem - você pode desejar modificar esse comportamento. As plataformas móveis impedem que os sprites dos jogadores caiam da mesma maneira que uma plataforma padrão faria. O modelo os move para cima e para baixo, mas com uma pequena modificação eles podem ser movidos para a esquerda e para a direita.

Quando um modelo é selecionado ou você seleciona um evento que já possui algum código, você pode editar o código desse evento específico. Deslocamento de símbolo e A retorna para a tela de seleção de eventos, o editor irá executar rapidamente o seu código e informar se não entender nada, posicionando o cursor na linha que ele não entende, o código é então compilado diretamente para um código de máquina extremamente rápido.

Chaves do editor de código:

Linha de corte	SYM-Y
Linha de pasta	SYM-U
Página inicial	SYM-Q
Fim	SYM-E
CAPS-2	Toggle caps lock
CAPS-3	Excluir encaminhar
CAPS-4	Inserir / sobrescrever
SYM-CAPS	Alternar modo estendido

Funções só podem ser usadas após um IF.

#### **IF**

Teste. Se a seguinte condição for verdadeira, o código até a próxima instrução ENDIF será executado. IF pode ser usado com uma função, ou para testar variáveis ou parâmetros de sprites entre si, ou contra valores numéricos específicos.

#### **ENDIF**

Marca o final do código condicional.

#### **LET**

Como no BASIC, isso permite atribuir um valor a uma variável ou parâmetro de sprite. O valor atribuído pode ser um número ou outra variável ou parâmetro de sprite.

#### **KEY**

Função. Espera um único argumento numérico e a condição é verdadeira se a tecla for pressionada.

#### **CANGROUP, CANGODOWN, CANGOLEFT, CANGORIGHT**

Funções. A condição é verdadeira se o sprite atual puder se mover para cima / baixo / esquerda / direita.

#### **LADDERABOVE, LADDERBELOW**

Funções. A condição é verdadeira se o sprite atual puder subir / descer uma escada.

#### **X, Y**

Parâmetros Sprite. Estas são as coordenadas do sprite atual.

#### **A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T**

Variáveis globais. Eles contêm valores de 8 bits, portanto, valores de 0 a 255 são possíveis.

## **SCREEN, LIVES**

Variáveis globais. Eles contêm o número da tela atual e as vidas restantes.

## **TYPE**

Parâmetro Sprite. Este é o tipo de sprite sendo processado. Melhor usado em conjunto com o parâmetro IMAGE, definir este parâmetro irá mudar completamente o comportamento do sprite - útil para transformar um desagradável em um bônus, ou fazer um sprite parar e explodir antes de matá-lo. Não há razão para que você não possa mudar um sprite para digitar zero e colocá-lo sob o controle do jogador, ou mudar o tipo de sprite do jogador para outra coisa com um conjunto de controles ligeiramente diferente. Desde que o novo tipo de sprite tenha o código apropriado configurado no evento relevante, não há limite para o que você poderia fazer. Se você não tiver configurado nenhum código para o sprite, seu sprite ficará lá - o que pode ser bom se você quiser.

## **IMAGE, FRAME**

Parâmetros Sprite. Estes são os números de sprite e frame mostrados no editor de sprite. Você pode mudar o sprite de acordo com a direção que o jogador estiver enfrentando, ou talvez você queira dar ao jogador uma escolha de veículos para controlar. Definir um número de quadros além do limite do sprite resultará em uma imagem de sprite diferente sendo exibida, então use FRAME com cuidado. Ao definir a imagem, é uma boa ideia definir o quadro como 0 ao mesmo tempo, a menos que você tenha uma boa razão para não fazê-lo.

## **DIRECTION, PARAMA, PARAMB**

Parâmetros Sprite. Você pode usá-las como quiser, talvez para indicar a direção na qual um sprite em particular está se movendo ou a fase específica pela qual está passando. Inestimável para qualquer forma de IA inimiga.

## **ANIMATE**

Comando. Anima o sprite atual, percorrendo automaticamente os quadros em ordem crescente.

## **ANIMBACK**

Comando. Como ANIMATE, mas percorre os quadros em ordem decrescente.

## **NEXTLEVEL, RESTART**

Comando. Mover para o próximo nível e reiniciar o nível atual, respectivamente.

## **SPRITEUP, SPRITEDOWN, SPRITELEFT, SPRITERIGHT**

Comandos Mova o sprite atual de acordo. Nenhuma verificação é feita para blocos no caminho, ou condições fora da tela, então você terá que fazer isso com funções como CANGOLEFT ou LADDERABOVE.

## **SPAWN**

Comando. Espera 2 parâmetros para o tipo e imagem do sprite. Isso gera um novo sprite com o tipo e imagem especificados na posição dos sprites atuais. O novo sprite é criado com FRAME, DIRECTION, PARAMA e PARAMB todos configurados para zero. O sprite atual não é afetado.

## **SPAWNED**

Comando. Deve ser usado somente após uma instrução SPAWN. Este comando seleciona o sprite recém gerado. Qualquer código escrito após SPAWNED se referirá ao novo sprite. Use ORIGINAL para voltar ao sprite primário original. Alternativamente, você pode preferir colocar seu código no evento inicialização do sprite, depende de você.

## **REMOVE**

Comando. Remove o sprite atual da tabela. Útil para destruir inimigos ou pegar objetos.

## **PUTBLOCK**

Comando. Requer um parâmetro para especificar o número do bloco, isso coloca o bloco especificado na posição atual de LINE e COLUMN na tela e pode ser útil para abrir e fechar portas. Note que sair e voltar a entrar no ecrã ou redesenhar depois de MENU ou INV, irá restaurar o ecrã para a sua posição predefinida, a menos que tenha selecionado a especialização de quebra-cabeças.

### **COLLISION**

Função. Requer um argumento numérico para especificar o tipo de sprite. A condição é verdadeira se o sprite atual estiver em colisão com outro sprite do tipo especificado.

### **OTHER**

Comando. Deve ser usado somente após uma verificação de COLISÃO bem-sucedida. Este comando seleciona o outro sprite, isto é, o secundário com o qual o sprite original acabou de colidir. Qualquer código escrito depois de OTHER se referirá ao sprite secundário. Use ORIGINAL para voltar ao sprite primário original quando terminar o código para o sprite secundário.

### **ORIGINAL**

Comando, usado após os comandos OTHER e SPAWNED, este reverte para o sprite original.

### **ENDGAME**

Comando. Termina o jogo em vitória. Isso realiza o evento Completed Game. Completar a última tela em um jogo com níveis sequenciais fará o mesmo. A única outra maneira em que um jogo pode terminar é se o jogador perder todas as suas vidas, mas isso não executa o evento Completed Game.

### **SHOWSCORE**

Comando. Mostra a pontuação na posição atual do cursor. Deve ser imediatamente precedido por instruções para configurar a linha e a posição da coluna para o cursor.

### **SCORE**

Comando. Espera ser seguido por um número para adicionar à pontuação. SCORE 100 adicionará 100 pontos ao total do jogador. Os valores de 0 a 255 são válidos.

### **SOUND**

Comando. Inicia um efeito sonoro com o chip de som AY. Espera um único parâmetro para o número de som para jogar.

### **BEEP**

Comando. Inicia um efeito sonoro do bip. Espera um único parâmetro para a duração do som. Valores acima de 127 produzem efeitos de ruído branco.

### **CLS**

Comando. Limpa a tela.

### **BORDER**

Comando. Espera um argumento de 0 a 7 inclusive. Define a cor da borda.

### **COLOUR**

Comando. Espera um argumento. Define os atributos de exibição permanentes, o formato é 128 \* FLASH + 64 \* BRIGHT + 8 \* PAPER + INK. Pode ser usado antes de exibir texto ou limpar a tela.

### **DELAY**

Comando. Espera um argumento. Pausa pela duração especificada.

### **MESSAGE**

Comando. Espera um argumento para o número da mensagem a ser exibida.

### **KILL**

Comando. Inicia o evento kill player e diminui o contador de vida. Você deve configurar o contador de vidas no evento de inicialização usando algo como LET LIVES = 3.

### **LINE, COLUMN**

Variáveis. Estes determinam a posição em que a pontuação ou mensagem será exibida. Eles são temporários e mudam toda vez que um sprite é exibido, portanto, sempre os use imediatamente antes de um comando MESSAGE ou SHOWSCORE.

#### **GETRANDOM**

Função. Gera um número aleatório entre zero e o argumento e coloca-o na variável RND. GETRANDOM 100 gerará um número de 0 a 99, GETRANDOM 2 gerará um zero ou 1.

#### **RND**

Variável. O último número aleatório gerado pelo GETRANDOM.

#### **ADD, SUBTRACT**

Comandos. Adicione ou subtraia de ou para um parâmetro ou variável de sprite, por exemplo: ADD 1 TO A ou SUBTRACT 5 FROM B.

#### **DISPLAY**

Comando. Exibe o valor numérico na variável ou parâmetro a seguir. Por exemplo, DISPLAY LIVES.

#### **SCREENUP, SCREENDOWN, SCREENRIGHT, SCREENLEFT**

Comandos. Mover para cima, para baixo, esquerda ou direita, uma tela, se possível.

#### **DEADLY**

Função. A condição é verdadeira se o sprite atual estiver em contato com um bloco mortal.

#### **CUSTOM**

Função. Semelhante a DEADLY, true se o sprite atual estiver em contato com um bloco CUSTOM.

#### **WAITKEY**

Comando. Espera por um pressionamento de tecla.

#### **JUMP**

Comando. O sprite atual irá pular, desde que não esteja no ar e não haja paredes no caminho. Como você esperaria, qualquer sprite pode ser feito para pular, não apenas aqueles sob o controle direto do jogador.

#### **FALL**

Comando. Desde que o sprite já não esteja caindo ou pulando, isso verifica se o sprite está em pé em cima do solo sólido. Se não, ele começará a cair, usando a metade descendente da mesa de pulo (seu jogo encontrará automaticamente este ponto, mesmo que você tenha modificado a mesa de pulo). A QUEDA(FAAL) é útil se um tipo de sprite estiver sujeito à gravidade, por exemplo, jogos de plataforma.

#### **GOT**

Função, usado em instruções IF e espera um único argumento. A expressão é verdadeira se o objeto especificado for de propriedade do player. O argumento pode ser uma variável se necessário, então IF GOT 1 é válido, assim como IF GOT A. Você pode até usar um parâmetro sprite como um argumento, se desejar.

#### **GET**

Comando. Espera um parâmetro especificando o objeto a ser obtido. Coloca o objeto especificado no inventário do jogador, independentemente de onde ele esteja. O objeto pode estar na tela atual, em outra tela ou simplesmente ausente. Cabe a você decidir quando o jogador pode obter um objeto específico. Você pode querer usar DETECTOBJECT para determinar quando um sprite está tocando um objeto primeiro.

#### **PUT**

Comando. Requer um argumento especificando o número do objeto. O objeto é solto na tela na posição do sprite atual, a menos que já esteja na tela atual. O argumento pode ser numérico, um parâmetro de variável ou sprite.

#### **DETECTOBJ**

Comando. Detecta objetos tocados pelo sprite atual e coloca o resultado na variável OBJ. Se o sprite estiver tocando em mais de um objeto, somente o objeto com o menor número será detectado. Se nenhum objeto for detectado, o OBJ será configurado para 255.

#### **OBJ**

Variável. O último número do objeto detectado pelo DETECTOBJ.

#### **CLOCK**

Variável de leitura. O contador de quadros atual, em 50 segundos. Útil para horários.

#### **MENU**

Comando. Requer um argumento especificando o número da mensagem. Produz um menu pop-up no meio da área de reprodução, onde as opções são tiradas do número da mensagem especificada, cada uma em sua própria linha. O jogador então seleciona uma opção usando os controles para cima, para baixo e fogo. Geralmente isso acontece quando o jogador pressiona uma determinada tecla. No entanto, ele pode aparecer em outro evento de sprite, por exemplo, quando um determinado sprite colide com o jogador. Você pode exibir uma mensagem - a conversa do sprite com o jogador - e usar o MENU para solicitar uma resposta. A tela é redesenhada após o MENU.

#### **INV**

Comando. Requer um argumento especificando o número da mensagem, semelhante ao menu, mas isso produz um menu baseado em objetos em posse do jogador. Configure uma lista dos nomes de objetos usados em seu jogo em uma mensagem com cada item em uma linha separada e use INV n, onde n é o número da mensagem que contém os nomes dos objetos. INV apresentará ao jogador um menu contendo apenas os itens atualmente no inventário do jogador. Útil para selecionar um item para processar ou descartar. INV trabalha apenas com a especialização de quebra-cabeças.

#### **EXIT**

Comando. Todo o processamento do evento atual é finalizado.

#### **REPEAT**

Comando. Requer um parâmetro para determinar o número de repetições. O código até o próximo ENDREPEAT será repetido o número de vezes especificado. REPEAT não pode ser aninhado.

#### **ENDREPEAT**

Comando. Isso marca o fim do bloco de código a ser repetido. O código colocado em um loop REPEAT..ENDREPEAT é automaticamente recuado pelo editor para melhorar a legibilidade.

#### **MULTIPLY**

Comando. Multiplique um parâmetro ou variável de sprite, por exemplo. MULTIPLICAR C (BY)POR 7.

#### **DIVIDE**

Comando. Divida um parâmetro ou variável de sprite, por exemplo. DIVIDE Y POR 8.

#### **SPRITEINK**

Comando. Define a cor da tinta para o sprite atual para a cor especificada, por exemplo, SPRITEINK 4 OU SPRITEINK PARAMB. Não afeta PAPER, FLASH OU BRIGHT. Para evitar deixar um rastro de INK com cores diferentes, redefina o SPRITEINK no início de cada evento de sprite. Se o seu fundo INK é normalmente 7 (branco), você pode querer usar o SPRITEINK 7 no começo do evento sprite, mover o sprite normalmente, então configure para a cor que você quiser. Os sprites colecionáveis estáticos só devem redefinir o SPRITEINK para o INK de fundo normal quando forem coletados. Veja o jogo de demonstração Diamond Geezer por um exemplo.

#### **LASER**

Comando. Espera um argumento numérico. Dispara um laser para a esquerda ou para a direita a partir da posição atual do sprite, dependendo do parâmetro. Até mesmo os parâmetros são disparados, os ímpares disparam à direita. As colisões de sprites com lasers podem ser detectadas nos eventos do tipo sprite usando IF COLLISION 10. LASER só funciona se a especialização de partículas for selecionada no menu diversos.

#### **TRAIL**

Comando. Útil para poeira de fada ou trilhas de vapor, TRAIL cria uma partícula de trilha de vapor no centro do sprite atual. Usando ADD 8 TO Y ou SUBTRACT 8 FROM Y primeiro, você pode criar uma trilha para a esquerda ou para a direita do sprite. Lembre-se de adicionar ou subtrair 8 novamente após o comando TRAIL para restaurar as coordenadas originais do sprite. Só trabalha com a especialização de partículas.

#### **EXPLODE**

Comando. Requer um argumento numérico para especificar o número de partículas. Cria uma explosão na posição atual do sprite. Usado com REMOVE, o EXPLODE remove a necessidade de desenhar uma animação de explosão para seus sprites. Se você esquecer de seguir EXPLODE com um parâmetro, o próximo comando não será compilado corretamente. Se esse comando for REMOVE seu sprite não será removido e você vai se perguntar por que explode, mas continua em jogo. Especialização de partículas apenas.

#### **FADE**

Comando. Transforma lentamente a janela para preto. O Fade só funciona com a especialização de efeitos ativada.

#### **TICKER**

Comando. Pixel rola uma mensagem de texto na posição atual LINE e COLUMN. Espera 2 argumentos, o primeiro é o número da mensagem e o segundo a largura da mensagem em caracteres. O TICKER continua e repete a mensagem quando chega ao fim. TICKER 1 16 rolará a mensagem 1 em uma faixa com metade da largura da tela. Para desativar TICKER, especifique um número de mensagem que não existe, por exemplo, TICKER 99 1. Só funciona se a especialização de efeitos estiver ativada.

#### **BIGMESSAGE**

Comando. Requer um parâmetro especificando uma mensagem de texto. Exibe a mensagem na tela em texto de altura dupla. Apenas inicialização de efeitos.

#### **REDRAW**

Comando. Redesenha a tela, sprites, objetos e quaisquer partículas. Se você está escrevendo uma aventura arcade, você pode querer exibir mensagens na janela do jogo, pausar por uma tecla com WAITKEY e depois REDRAW antes de retornar à ação.

#### **SILENCE**

Comando, silencia o chip de som AY.

#### **CONTROLMENU**

Comando. Útil para o evento de menu / intro, o CONTROLMENU espera que o jogador selecione uma opção de controle. As opções são 1 para teclado, 2 para joystick Kempston e 3 para joystick Sinclair (teclas 67890).

#### **DIG**

Comando. Espera um parâmetro de 0 a 3, especificando a direção para cima, para baixo, para a esquerda ou para a direita, respectivamente. Remove os blocos do FODDER imediatamente adjacentes ao sprite do jogador. Disponível apenas com a especialização em quebra-cabeças.

#### **STAR**

Comando. Parte do motor de partículas, STAR cria um campo de estrelas vertical ou horizontal quando chamado repetidamente. Espera um parâmetro para especificar a direção, 0 para vertical e 1 para horizontal.

#### **BIGSCORE**

Comando. Como SHOWSCORE, mas exibe a mensagem na tela em texto de altura dupla. Apenas inicialização de efeitos.

#### **STOPFALL**

Comando. Para um sprite pulando ou caindo no ar e invoca o evento caiu muito longe, se aplicável.

### Exemplo 1

Suponha que temos uma janela de jogo de 22 caracteres de altura com um único espaço entre os caracteres na parte superior e inferior da tela. Digamos que queremos desenhar uma caixa ao redor da janela. Podemos fazer isso exibindo mensagens no evento de inicialização do jogo. Poderíamos configurar a mensagem 0 como a linha superior a ser exibida acima da janela, com a mensagem 1 como a linha inferior. Para desenhar as linhas nas laterais, podemos configurá-las na mensagem 2 e, em seguida, usar um loop REPEAT para desenhá-las. Algo assim pode fazer o truque para nós:

```
LET LINE = 0
MESSAGE 0
REPEAT 22
  MESSAGE 2
ENDREPEAT
MESSAGE 1
```

### Exemplo 2

Se selecionarmos sprite type 3 e escolher Static collectable como nosso template, serão apresentadas as seguintes linhas de código:

```
IF COLLISION 0
  REMOVE
  SCORE 100
ENDIF
```

Podemos adicionar um efeito sonoro de bipe curto inserindo o comando BEEP 30:

```
IF COLLISION 0
  BEEP 30
  REMOVE
  SCORE 100
ENDIF
```

Indo mais além, podemos inserir as seguintes linhas no começo, podemos mudar o comportamento do sprite:

```
IF CANGROUP
  SPRITEUP
ELSE
  REMOVE
ENDIF
```

Então nosso evento agora é assim:

```
IF CANGROUP
  SPRITEUP
ELSE
  REMOVE
ENDIF
IF COLLISION 0
  BEEP 30
  REMOVE
  SCORE 100
ENDIF
```

Agora temos um sprite colecionável que sobe até atingir algo e depois desaparece como uma bolha. Agora precisamos de algum código para gerar nossos novos sprites na parte interior da tela. Podemos fazer isso no loop principal, então selecione o evento Main loop 1 e insira estas linhas:

```

GETRANDOM 127
IF RND <= 6
  SPAWN 3 0
  SPAWNED
  IF TYPE = 3
    LET X = 168
    GETRANDOM 160
    LET Y = RND
    ADD 8 TO Y
  ENDIF
  ORIGINAL
ENDIF

```

Este código primeiro gera um número aleatório de 0 a 127, e se esse número for menor ou igual a 6, ele gera um sprite do tipo 3 (o evento anterior que editamos) e uma imagem de sprite de 0. Se o sprite tiver gerado corretamente, ele define a posição x para a parte inferior da área de reprodução (supondo que a área de reprodução termine 8 pixels a partir da parte inferior da tela) e escolhe uma posição y aleatória entre 8 e 168.

### Exemplo 3

Suponha que estamos escrevendo um jogo de labirinto de cima para baixo no estilo Atic Atac e queremos incluir o patrulhamento de inimigos. Podemos selecionar o tipo de sprite 4 e escolher o modelo inimigo de patrulhamento (L / R). No entanto, este modelo pressupõe que estamos escrevendo um jogo de plataforma, portanto, precisaremos remover as linhas que verificam as lacunas, pois o sprite não estará em uma plataforma. No modelo, essa verificação é feita adicionando ou subtraindo 16 (a largura do sprite) à coordenada horizontal (Y) do sprite e testando para ver se o sprite pode ficar inativo. Se puder, o sprite muda de direção porque deve haver uma lacuna. No nosso caso, nós simplesmente queremos nos mover para a esquerda ou para a direita até atingirmos alguma coisa, então nos viramos, então é uma questão de deletar as linhas desnecessárias. Nosso modelo deve ficar assim:

```

IF PARAMA = 0
  IF CANGOLEFT
    SPRITELEFT
  ELSE
    LET PARAMA = 1
  ENDIF
ELSE
  IF CANGORIGHT
    SPRITERIGHT
  ELSE
    LET PARAMA = 0
  ENDIF
ENDIF

```

## Character Set

O AGD vem com seu próprio editor de fontes. Os caracteres são editados usando as teclas do cursor com 0 ou SPACE para definir / desmarcar um pixel. Todo o intervalo imprimível de 96 caracteres é exibido na parte inferior da tela, e você pode percorrê-los com as teclas N e P.

Se preferir, você pode carregar uma fonte padrão do Spectrum criada com outro utilitário no AGD. Primeiro, certifique-se de que, se estiver usando um emulador, ele não carregará automaticamente as fitas ou perderá o jogo em que você está trabalhando. Pressione L para carregar uma fonte.

P = caractere anterior  
N = caractere seguinte  
L = carregar fonte da fita

ENTER = retornar ao menu principal

## **Informação técnica**

Os jogos criados com o utilitário são independentes e devem funcionar independentemente do próprio editor. AGD não altera o modo de interrupção, então você está livre para configurar suas próprias interrupções, para tocar música, por exemplo. No entanto, você não deve desabilitar as interrupções. Se você deseja configurar suas próprias interrupções usando o IM2, certifique-se de que sua rotina de serviço incrementa o byte no local da memória 23672 a cada quadro, ou seu jogo irá travar. AGD não altera o valor do registrador IY, embora IX seja usado em todo lugar, principalmente como um ponteiro de sprite.

Os jogos ocupam a memória de cerca de 31200 para cima, e esta é a área que é salva pelo utilitário, o endereço inicial exato varia conforme os eventos não críticos crescem de 31232 para criar espaço na memória RAM superior do jogo. Abaixo disso, o código para o editor está localizado, sendo trocado conforme exigido de outros bancos de RAM. No topo da RAM, os últimos 768 bytes de 64768 a 65535 são usados como uma área de mapa de colisões para distinguir entre diferentes tipos de blocos - paredes, escadas, espaço vazio e assim por diante. No final do jogo há uma área de 300 bytes usada para o motor de partículas caso você decida usar lasers, trilhas de vapor ou explosões. Esse buffer não existe nos mecanismos de quebra-cabeças ou efeitos. Nenhum buffer é necessário para a descompactação de dados de tela, uma vez que eles são expandidos dinamicamente pela rotina que desenha a tela.

Ao salvar suas criações, a opção de salvar um carregador BASIC criará automaticamente uma versão de carregamento automático do seu jogo, completa com uma tela de carregamento, se você configurá-la no menu diverso. Os jogos de execução automática não retornam ao BASIC, fazendo loop continuamente. Portanto, você precisará programar sua própria tela de introdução na AGD. Se você optar por não criar um carregador BASIC, seu jogo retornará ao BASIC no final.

Os jogos podem ser controlados via teclado ou joystick Kempston, e isso é feito via byte no endereço 32005. Um valor de 1 significa que os controles Kempston são usados para as primeiras cinco teclas, 2 usa as teclas Sinclair 6, 7, 8, 9 e zero. Qualquer outro valor é padronizado para os controles do teclado.

O AGD define a variável de sistema de taxa de repetição do teclado no endereço 23562. A ROM do Sinclair define isso como 5 na inicialização e a AGD reduzirá isso para 3 para acelerar a edição. No entanto, se você POKE 23562 com qualquer valor diferente de 5 AGD não irá alterá-lo. Então você pode POKE 23562,1 para definir a taxa de repetição do teclado para a velocidade máxima, caso deseje fazê-lo.

### **Adicionando música ou sub-rotinas de terceiros (apenas partícula / efeitos)**

Se você quiser adicionar suas próprias rotinas ao jogo, você deve adicionar estas três linhas temporariamente ao evento onde você deseja chamar a rotina externa:

```
ASM 205  
ASM 85  
ASM 5
```

Não se preocupe muito sobre como isso funciona, é simplesmente uma chamada fictícia para um ponto na ROM Sinclair que não faz nada. Nós vamos mudar isso mais tarde.

Em seguida, selecione diversos no menu e anote o endereço exibido na parte inferior da tela. Este é o seu endereço de compilação, a localização da memória em que você deve compilar sua rotina externa. Em seguida, usando uma calculadora, você precisa calcular dois valores para colocar no(s) evento(s) onde você adicionou anteriormente os valores do ASM. Você precisa calcular seu endereço de compilação dividido por 256 e também o restante.

Volte para o(s) evento(s) onde você coloca os valores ASM, em seguida, substitua 5 pelo endereço de compilação dividido por 256, e 85 com o restante ENTÃO SALVE IMEDIATAMENTE SEU JOGO SEM A OPÇÃO BÁSICA DO CARREGADOR. Não o teste neste ponto, pois o AGD irá travar porque sua rotina não está na memória, você precisará certificar-se de que seu carregador BASIC carrega sua própria rotina além do seu jogo.

### **Kempston Mouse**

Poucos jogos do Spectrum fazem uso do controle do mouse, e o AGD não foi projetado especificamente para funcionar com esse dispositivo. No entanto, o AGD tem uma rotina oculta para ler o mouse Kempston. Para usá-lo, selecione a especialização de aventura no menu miscelânea e, em seguida, coloque esses três comandos em um dos seus principais eventos de loop:

ASM 205  
ASM 192  
ASM 129

Isso colocará as coordenadas do mouse e os status dos botões de disparo nas variáveis A, B e C.

### **Outras informações**

Paul Jenkinson produziu uma série de excelentes tutoriais em vídeo da AGD, que podem ser encontrados em seu site:

<http://www.randomkak.bloaspot.co.uk/p/aad-video-tutorials.html>

O fórum oficial da AGD é o lugar para encontrar notícias sobre os últimos lançamentos, dicas sobre como tirar o melhor proveito do utilitário e detalhes de jogos que já foram escritos com a AGD ou estão atualmente em desenvolvimento. Você pode encontrar o fórum aqui:

<http://arcadeqamedesioner.proboards.com/>

Outros recursos:

<http://www.funspot.it/cateorv/aad/>

<http://www.vintageisthenewold.com/agd-arcade-game-designer-initial-setp>



# AGDX Coder's Manual

Saudações e bem-vindos... AGDX é uma versão atualizada de Jonathan Cauldwell's Arcade Game designer que fornece mais de 60 atualizações para o pacote original, projetado para melhorar a flexibilidade e tornar a sua experiência de codificação de jogo mais produtiva. Enquanto ele funciona em um Spectrum regular, presume-se que você está executando-o em um emulador de espectro.

Supõe-se também que você está familiarizado com o original Arcade Game Designer, e apenas as adições AGDX serão apresentados aqui.

AGDX foi efetivamente projetado para ser usado em conjunto com o emulador Fuse que é muito popular com os desenvolvedores. Obviamente ele vai correr bem em qualquer emulador, mas algumas configurações de teclado podem precisar de ser levado em conta. No FUSE, a tecla Shift Symbol é acessada através da tecla CTRL. Muitas teclas adicionais em AGDX fazem uso desta tecla, por isso é uma boa idéia para verificar como isso é acessado em seu emulador de escolha. Ao longo deste documento, esses comandos serão listados como usando a tecla CTRL, por exemplo, CTRL-C para copiar.

Este software, como AGD em si, é feito gratuitamente a licença Creative Commons padrão. Enquanto AGDX foi criado com o consentimento de Jonathan Cauldwell, não é uma versão oficial do software, nenhuma reivindicação é feita ao código original, e os direitos autorais permanecem com o autor original.

Como qualquer coisa envolvendo computadores de 8 bits, trabalhar com AGD pode ser uma experiência gratificante e frustrante. Recorde conservar frequentemente, porque nenhuma responsabilidade pode ser tomada para a perda de dados. Com paciência você vai achar que você é capaz de construir jogos de qualidade que você pode se orgulhar. Boa sorte!

## Opções do menu principal

O menu principal tem algumas adições - em primeiro lugar, os números na tela mostram quanta memória está sendo usado atualmente para blocos, telas, eventos, sprites e assim por diante. O primeiro número é o local de início, o segundo o espaço ocupado. Usuários avançados podem ajustar esses valores alternando 'Editar ponteiros' no menu Miscelânea. A finalidade deste é permitir que os dados sejam importados rapidamente de outras versões de AGD, e para criar o espaço para remendos e addons. Se você está apenas construindo um jogo, você não precisa se preocupar em mudá-los.

A opção para alterar o menu de tinta e papel foi removida para economizar espaço. No entanto, se você ainda quiser alterá-lo, você pode POKE endereço 24305 com a cor do atributo que você deseja para o menu, executar o jogo e, em seguida, sair, e seu menu será atualizado na cor de sua escolha.

Teclas de atalho para os editores mais comuns são acessíveis a partir de qualquer um dos editores que usá-los, permitindo que você pule rapidamente entre os menus.

CTRL-1 - Block Editor  
CTRL-2 - Screen Editor  
CTRL-3 - Sprite Image Editor  
CTRL-4 - Sprite Placement  
CTRL-5 - Object Editor

Você pode desejar personalizar a aparência do AGDX, e isso pode ser feito a partir do menu principal da seguinte forma: As teclas de controle para AGDX são na maior parte consistentes do editor ao editor.

'/'=' - primary next / previous  
'1'/'2' - secondary next / previous  
'3'/'4' - next + 10 / previous -10

## Fonte Editor



Um número de adições foram feitas para o editor de fonte.

Como com todos os editores gráficos, os pixels completos padrão nos gráficos ampliados foram substituídos por praças delimitadas. Atualmente, o caractere atual agora é realçado, e também é possível **copiar** (tecla **M**), **colar** (chave **K**) e **limpar** (chave **C**). Os caracteres que são copiados também podem ser movidos para o editor de bloco. Se você tem gráficos definidos pelo usuário aqui, você também pode pressionar **'F'** para ver a fonte Spectrum padrão, para saber qual chave ele corresponde.

Porque o conjunto de caracteres nunca se move na memória AGD, é um lugar ideal para armazenar pequenas rotinas de montagem adicionais, e os três números no lado direito são fornecidos para ajudar com isso. O primeiro número mostra o endereço na memória onde esse caractere é armazenado. Os segundos dois caracteres mostram o número necessário para chamar esse endereço de dentro de um script AGD. Neste exemplo, a chamada seria ASM 205 ASM 8 ASM 123. Isso torna possível importar pequenas rotinas para o charset e executá-los a partir de um script.

Observe que no AGDX o comando 'ASM' foi substituído por um único '!' -parada completa.

### **Importando um jogo de AGD 4.7 ou uma versão anterior do AGDX.**

O editor de fontes também é o lugar para começar se você quiser exportar ou importar um jogo para AGDX de AGD, ou para uma nova versão. Este processo se totalmente coberto online aqui:

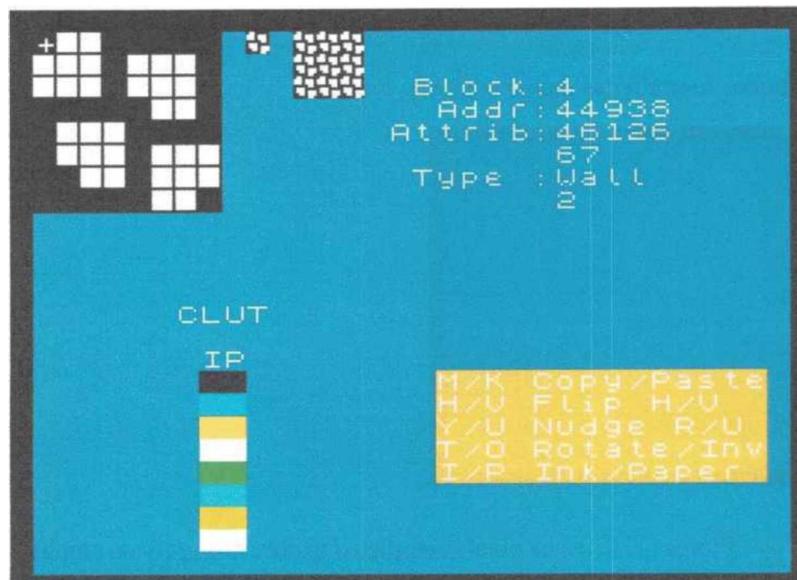
<https://www.youtube.com/watch?v=TKt6r7QZrso>

Aqui estão as instruções básicas:

- 1) Abra o seu jogo na versão que você usa, e pressione 'a' para ir para o editor de Char
- 2) Escolha '**salvar binário**' ou '**salvar código da máquina**', e salvar um arquivo a partir de 31232, tamanho do arquivo 33536.
- 3) Abra a versão mais recente do AGDX e vá para o editor de Char.
- 4) Escolha '**Load Binary**', e carregar o binário que você criou em 31232.
- 5) Saia do editor de personagens e seu jogo estará lá pronto para ser editado.

O apêndice B deste manual fornece instruções sobre como importar dados de Sprite para o AGDX a partir do AGD 4.6.

## The Block Editor



Um número de adições úteis foram feitas para o editor de bloco. Foi adicionada uma visualização de mosaico que mostra o bloco num grupo de 3x3 que ajuda a criar mosaicos. As teclas do cursor também têm envoltório ao redor da esquerda para a direita e até para baixo para ajudar com isso. Este recurso está agora em todas as telas do editor.

Chaves gerais para o editor de bloco são:

'-/'=' - next / previous block type  
'1'/2' - next / previous block  
'3'/4' - next / previous block +10/-10

Blocos 'especiais' também foram adicionados, estes podem ser qualquer número de 8 a 254. Estes blocos são acessíveis apenas através de rotinas de montagem, mas pode ser usado para muitos novos efeitos, tais como blocos colecionáveis, estrelas cintilantes, transportadores animados e muitas outras rotinas que são mostrados no YouTube.

<https://www.youtube.com/user/MrTravisHighrise>

As seguintes chaves foram adicionadas:

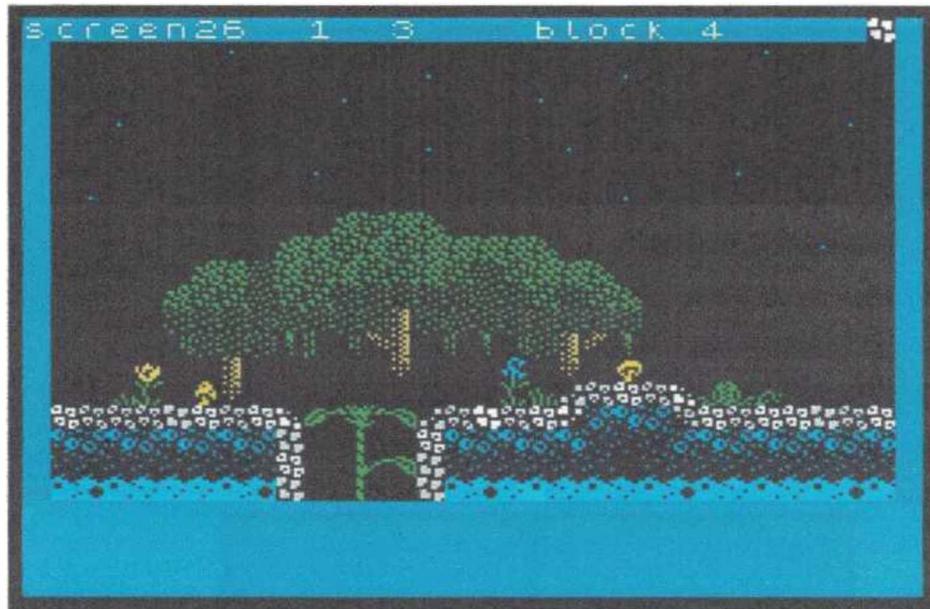
U - nudge up                    Y - deslocar a direita  
T - girar                        o — inverter

**CTRL -0** paleta invert (trocar tinta e papel)

'S' exibirá a paleta do bloco, uma prévia de todos os blocos atuais com um guia numérico. **CTRL-B** exibirá todos os blocos como uma única imagem que pode ser usada para exportar.

**CTRL-K** é usado para mover dados de Sprite em blocos e irá colar dados de Sprite copiados nos próximos quatro blocos (somente pixels, consulte editor de Sprite). Certifique-se de ter quatro blocos livres!

## The Screen Editor



screen from Nixy the Glade Sprite by Andy Johns, written with AGDX

O editor de tela teve um número considerável de adições. A posição atual de X e Y, bem como o número de bloco atual agora são exibidas. As teclas de navegação são as seguintes:

'-/'=' - next / previous block type  
'1'/2' - next / previous block  
'3'/4' - next / previous block +10/-10

Envoltório do cursor ao redor também é implementado para permitir a navegação fácil. As chaves adicionais são as seguintes:

'Z' - Coloque um bloco 'zero'. Isso funciona efetivamente como uma borracha.  
'U' - Desfazer último bloco colocado.  
**CTRL-Z** - Desfazer todo o trabalho feito desde a última salvar.  
'B' - Pegue o bloco que está atualmente o cursor.  
'S' - Ver a paleta de blocos (isso também irá salvar a tela atual).  
'Q' - Ver onde a tela é armazenada e tamanho na memória (também salva a tela atual).  
'C' - Limpar a tela atual.

Um novo recurso poderoso é a função de copiar/colar multiblock. Pressionar **CTRL-C** copiará um conjunto de 4x4 de blocos com o cursor no canto superior esquerdo. Há duas opções de colagem, uma vez que os blocos são copiados:

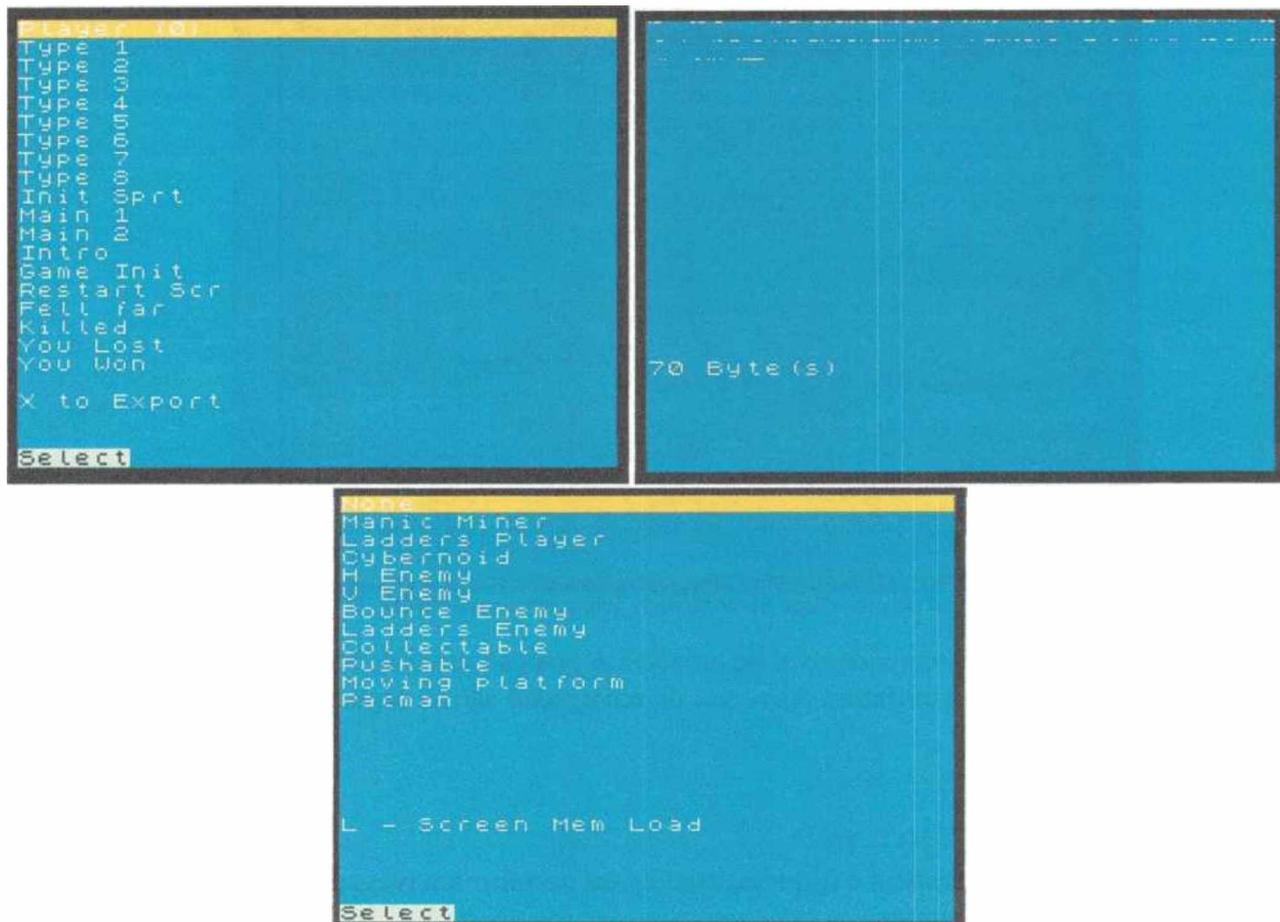
**CTRL-V** fará uma pasta padrão de todos os 16 blocos.

**CTRL-B** irá colar todos os blocos para além do bloco zero. Isso permite que você crie diferentes formas e tamanhos de 'bloco de pincel'.

**CTRL-R** é 'substituir bloco'. Isso substituirá todas as instâncias do bloco atual que está o cursor com o bloco que está selecionado no momento.

Esta ação pode ser desfeita usando **CTRL-Z**.

## The Event Editor



A maior mudança para o editor de eventos é a sintaxe abreviados, muitos dos comandos mais comuns foram substituídos por versões mais curtas.

Estes podem ser visualizados no apêndice A.

Além disso, há a capacidade de exportar Scripts para binários, que podem ser importados para outro jogo. Para exportar um script, selecione-o na lista e pressione a tecla 'X'. O script em questão será movido para a memória da tela (pixels aparecerá na tela, veja o exemplo acima). O número de bytes também será mostrado. Em seguida, você pode salvar o script como um binário do endereço 16384, com o tamanho fornecido.

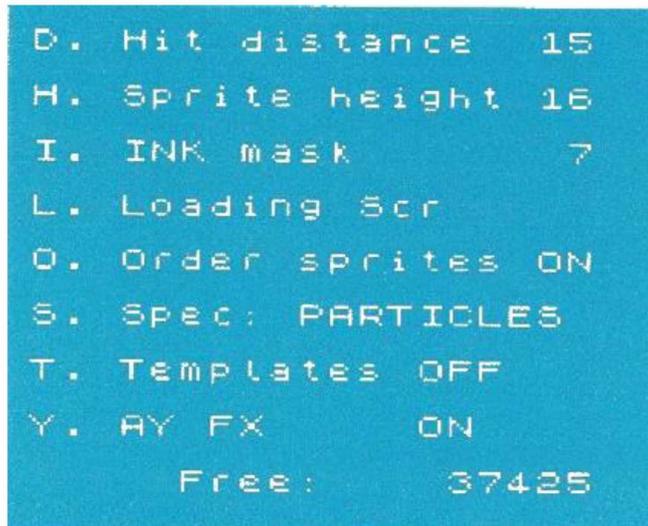
Para importar um script, primeiro você precisa ativar os modelos de script no menu diversos. Ao abrir um script vazio, o menu do modelo aparecerá, e a opção para carregar é agora visível (veja acima). Neste ponto, você deve *primeiro* carregar o binário de script no endereço 16384 e, em seguida, escolher '**tela mem Load**'. O script em questão será então carregado no editor.

Observe que os Scripts maiores que 360 bytes atualmente não funcionam, mas isso está sendo **trabalhado**. **Uma seleção de Scripts carregáveis também será incluída em breve no pacote AGDX.**

Os modelos de evento foram modificados para que as coordenadas usadas se adaptem ao tamanho da tela que você especificou. Um modelo de script, 'controles de estilo Pacman' também foi adicionado à lista de modelos.

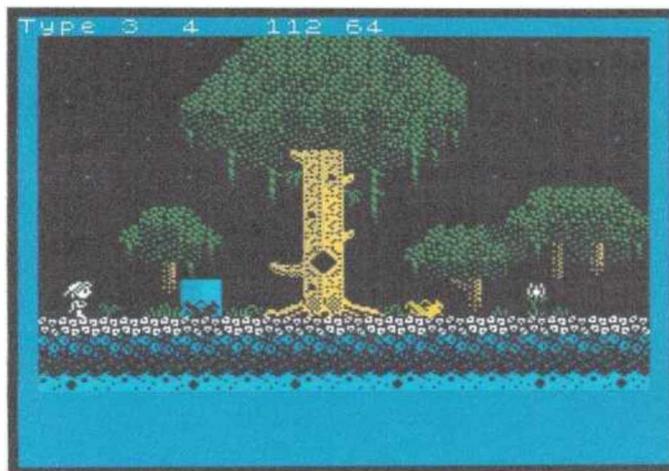
**CTRL-A** ainda é usado para compilar o evento. No entanto, pressionar **CTRL-Z** agora sairá do editor sem salvar as alterações, restaurando o script para a última versão salva.

## Save/Load and Miscellaneous



Pequenas adições foram feitas a estes menus. Ao salvar um jogo, há uma opção para uma "memória total Save", isso vai salvar toda a memória AGD e destina-se para quando o código adicional ASM foi armazenado mais alto na memória. Carregando um jogo agora tem uma consulta de confirmação para evitar o carregamento acidental e o menu diversos (miscellaneous) tem uma nova opção para ativar ou desativar os modelos de código.

## Sprite Placement



O editor de posicionamento de Sprite tem um número de adições úteis. Assim como o tipo de Sprite, o número da imagem do Sprite é visível, juntamente com as coordenadas X e Y. Além disso, pressionar a tecla 'P' alternará para o modo de posicionamento de precisão, que permite que os sprites sejam movidos pixel por pixel. Pressione 'P' novamente para voltar aos passos regulares de 8 pixels.

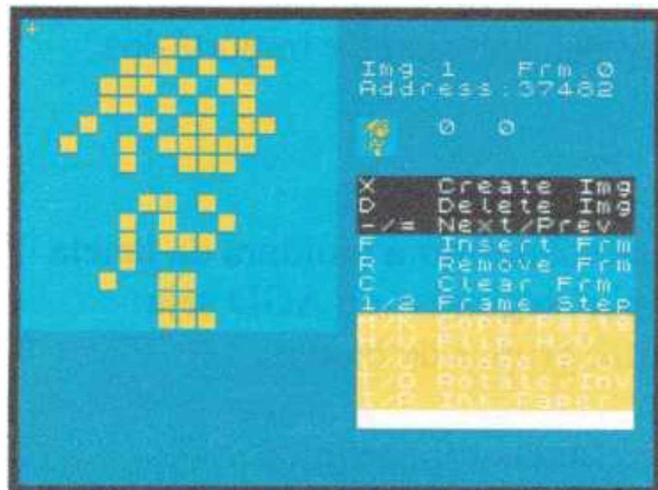
## Object Editor

Agora é possível copiar e colar sprites do editor de sprite para o editor de objetos, o que significa que é fácil usar todos os recursos do editor de Sprite e movê-los para o editor de objetos. Como resultado, nem todos os recursos de manipulação de imagem são necessários no editor de objeto, mas o painel de visualização está presente, juntamente com o endereço de memória e a opção de pressionar 'G' para a grade. Como com a colocação de Sprite, também é possível pressionar 'P' para alternar para o movimento de pixel único quando na tela de posicionamento do objeto.

## ULA+ Palette

A paleta original de ULA + fornecida em AGD foi ajustada a uma que se assemelha mais próxima à paleta do espectro do ZX à revelia, significando os jogos projetados sem uma paleta de ULA+ olhará como o autor pretendia ao jogar em máquinas compatíveis de ULA+.

## Sprite Editor



Um número considerável de novos recursos foram adicionados ao editor de Sprite.

O local da memória, o painel de visualização, as coordenadas X/Y e um guia para as chaves mais comuns foram adicionados. Novos recursos incluem:

- 1 - quadro passo para a frente
- 2 - quadro passo para trás (Segurar 1/2 para animar)
- I - Preview Ink                      P - visualizar papel
- Y - nudge direita                    U - nudge up
- t - girar Sprite                      O - inverter Sprite
- G - exibir grade                    s - planilha de acesso

A planilha Sprite (Key s) é outra ferramenta poderosa que permite que você carregue sprites em uma tela ou importe uma imagem externa. Quando na folha de Sprite, as chaves a seguir são usadas.

- S - pressionando S na planilha carregará o conjunto atual de sprites na tela Espaço/'0' — AGDX tentará ir para o quadro que você selecionar.
- L — lhe dará a opção de carregar uma imagem ZX Spectrum a partir de um arquivo Tap.
- A - alternará entre os modos de sobreposição de cores e atributos.
- M - copiar a seleção atual para a área de transferência

Também é possível carregar um arquivo binário ou .SCR na planilha. Quando estiver na folha de cálculo, carregue o ficheiro no endereço 49152 e, em seguida, prima 'R' para atualizar a página.

A planilha Sprite também pode servir como um backup para seus sprites como as alterações aqui não são atualizadas até que você pressione 'S'. Sprites também pode ser salvo como um SCR a ser importado mais tarde.

Outro recurso útil é bloco copiar e colar.

**CTRL-M** é cópia do bloco, e copiará os dados do sprite em um formato que permita que seja copiado no editor de bloco usando **CTRL-K** (Veja o guia do editor do bloco). A principal razão para essa adição é permitir que sprites ou objetos sejam duplicados como blocos.

**CTRL-K** é colar bloco. Isso permite que um bloco 8x8 seja colado no canto superior esquerdo do editor de Sprite. Este bloco pode ser copiado do editor de fontes, do editor de bloco ou usando **CTRL-M** como descrito acima. Combinando a cópia do bloco, cole a pasta junto com gire e lanç, é possível extrair formas simétricas tais como círculos mais rapidamente.

Os comandos adicionais do editor de Sprite podem ser vistos no guia de chaves AGDx.

## Sound Editor

O editor de som não era comumente usado e era um pouco buggy por isso foi desativado para economizar espaço. No

entanto, ainda é possível editar sons em versões anteriores e importá-los.

### **Further additions and bug fixes**

No menu de posicionamento e redimensionamento da janela, é possível definir a moldura da janela para tão pouco como 1x1. Isso foi feito para permitir que uma cópia 'segunda' do AGD seja executada que poderia ser usada para projetar e, em seguida, exportar grupos de blocos.

Um vazamento de memória que aconteceu quando sprites foram excluídos foi corrigido.

Um bug onde as partículas não seriam exibidas se a janela foi nivelada para o lado direito da tela foi corrigido.

A maioria dos eventos padrão foram removidos, bem como ter apenas uma mensagem e um som carregado no início do editor.

O 'Flip' vertical no editor de Sprite foi corrigido.

No AGD original, o editor de eventos se recusaria a carregar scripts quando a memória estava baixa e levaria muito tempo para ser aberta. Isso não acontece no AGDX como a forma como os sprites são compactados foi alterado. Os eventos carregam de forma rápida e fácil, mesmo quando a memória é apenas algumas centenas de bytes.

A rotina MENU e inventário foi escutada, isso agora foi corrigido. O menu e os inventários agora serão exibidos na linha e coluna selecionada no momento.

### **Credits**

AGD written by Jonathan Cauldwell

AGDX v1 - 3 developed by David Saphier and Allan Turvey

AGDX v4 developed by Allan Turvey, additional code by David Saphier.

E um grande agradecimento a todos na Comunidade AGD para o seu contributo e apoio durante o desenvolvimento da AGDX. Espero que você ache útil.

## Appendix A - New Syntax

AGD	AGDX	AGD	AGDX
ADD	ADD	KILL	KILL
ANIMATE	ANIM	LASER	LASER
ANIMBACK	ANIMB	LET	LET
ASM	. (Full stop)	MENU	MENU
BEEP	BEEP	MESSAGE	MSG
BIGMESSAGE	BMSG	MULTIPLY	MULT
BIGSCORE	BSCORE	NEXTLEVEL	NEXTLEV
BORDER	BORD	ORIGINAL	ORIG
BY	BY	OTHER	OTHER
CANGODOWN	CANDWN	PUT	PUT
CANGOLEFT	CANLFT	PUTBLOCK	PBLK
CANGORIGHT	CANRGT	REDRAW	RDRAW
CANGROUP	CANUP	REMOVE	RMV
CLOCK	CLK	REPEAT	RPT
CLS	CLS	RESTART	RESTART
COLLISION	HIT	SCORE	SCORE
COLOUR	CLR	SCREENDOWN	SCDWN
COLUMN	COL	SCREENLEFT	SCLFT
CONTROLMENU	CTRLMENU	SCREENRIGHT	SCRGT
DELAY	DELAY	SCREENUP	SCUP
DETECTOBJ	DOBJ	SHOWSCORE	SHWSCORE
DIG	DIG	SILENCE	SILENCE
DISPLAY	DISP	SOUND	SND
DIVIDE	DIV	SPAWN	SPAWN
ELSE	ELSE	SPAWNED	SPAWNED
ENDGAME	ENDGAME	SPRITEDOWN	SPDWN
ENDIF	ENDIF	SPRITEINK	SPRINK
ENDREPEAT	ENDRPT	SPRITELEFT	SPLFT
EXIT	EXIT	SPRITERIGHT	SPRGT
EXPLODE	EXPLODE	SPRITEUP	SPUP
FADE	FADE	START	START
FALL	FALL	STOPFALL	STOPFALL
FROM	FROM	SUBTRACT	SUB
GET	GET	TICKER	TICKER
GETRANDOM	GETRND	TO	TO
IF	IF	TRAIL	TRAIL
INV	INV	WAITKEY	WAIT
JUMP	JMP		

Dois novos comandos - GETBLK e GETATT, também foram adicionados, mas atualmente não são funcionais, pois foram encontrados para quebrar a compatibilidade com 4.7. As variáveis PARAMA, PARAMB e DIRECTION são substituídas por PMA, PMB e DIR.

## Appendix B

Como portar dados de Sprite do AGD 4.6 para o AGDX.

Nota: uma atualização que permita uma conversão mais fácil de 4.6 a 4.7 foi incluída agora. Um vídeo sobre isso é aqui:

<https://www.youtube.com/watch?v=Jtmf6vKQHaQ>

Movendo dados Sprite, incluindo todos os dados de quadro de 4.6 para AGDX é um pouco complicado, mas perfeitamente possível se você seguir os seguintes passos.

1) Abra seu jogo em AGD 4.6. Você precisará de duas partes importantes de informações - o número total de sprites e o número total de quadros. Se você estiver usando o Fuse, você pode ver ambos estes olhando a memória em endereços 32016 e 32019, ou \$7D10/\$7D13. Você pode usar o navegador de memória no Fuse para fazer isso.

Como pode ser visto neste exemplo, o número de sprites é \$13 (19 em decimal) e \$4e, que é 78 em decimal. Anote esses dois números.

2) Calcule o tamanho dos dados. O total de dados movidos será de acordo com a seguinte fórmula:

(número de quadros x Sprite Height x 8) + (número de sprites x 2)

Este é o total de dados de pixel mais os dados do quadro total. Neste caso:

$(78 \times 16 \times 8) = 9984$ , e  $19 \times 2 = 36$ . Total é  $9984 + 36 = 10020$ .

Os dados começam no endereço 36924, portanto, salve um arquivo binário desse endereço, com o tamanho calculado, que neste exemplo seria 10020.

Os dados agora estão prontos para serem carregados no AGDX. A fim de fazer isso, primeiro precisamos criar o mesmo número de sprites e frames em AGDX como precisamos importar. Assim, em AGDX, criamos o número correto de sprites (neste caso 19) e, em seguida, adicionar tantos quadros conforme necessário. Não importa onde os quadros são adicionados, todos eles podem estar em um Sprite. Uma vez feito, verifique a memória Sprite no menu principal, ele deve corresponder ao total calculado para o tamanho do Sprite (neste caso 9984).

Uma vez feito isso, você pode carregar seu arquivo binário no endereço 36944, e seus sprites agora estarão no AGDX com os dados de quadro corretos.

## Appendix C

### atualizações e alterações feitas em versões mais recentes

Consulte o arquivo 'AGDX updates' para obter detalhes completos de mudanças.

AGDX 4.7 versão 4ia

A exibição do ponteiro foi atualizada para mostrar o ponteiro atual como realçado. O ponteiro Sprite agora inclui dados de quadro para que os conjuntos de Sprite possam ser mais facilmente movidos.

Outra opção foi adicionada ao editor de tela pressionando 'E' permitirá que você edite imediatamente o bloco o cursor.

O tamanho total dos dados de blocos para eventos também é exibido na parte inferior da tela. Alguns menus foram abreviados para dar espaço para isso.

AGDX 4.7 versão 4I

Como um precursor da versão 5, que tem o novo Sprite espaço salvando código, esta é uma versão atualizada do AGDX4 que ainda é completamente compatível com AGD 4.7. A principal razão para esta atualização foi facilitar a exportação de jogos de AGDX4 para AGDX5, como o novo código significará que os jogos não podem ser portados simples através de uma carga binária básica.

No entanto, bem como o código baseado em memória, há algumas outras adições, e algumas remoções, conforme detalhado abaixo.

- 1) Exibir e ajustar ponteiros de memória. A capacidade de ajustar e alterar a memória reservada do AGD foi adicionada. Isso foi feito para que os dados podem ser movidos de AGD 4.6 para 4.7, e também para a nova versão do AGDX5. Alterar os ponteiros pode facilmente quebrar o seu jogo para que você só deve usá-lo para a importação de dados, e você certamente deve salvar qualquer trabalho antes de jogar com ele. Haverá mais sobre como usá-los quando a nova versão é liberada. As teclas de seta para cima/para baixo selecionam um ponteiro, as teclas de seta esquerda/direita adicionam/subtraem 1, e '=' as teclas adicionam 100 ao ponteiro escolhido.
- 2) Os bancos Sprite foram removidos para fazer espaço para outro código. Estes foram planejados para permitir o acesso a mais sprites na máquina 128K, mas isso provou ser difícil. Com o novo código 'Sprite byte 32', a necessidade para estes é diminuída, e os bancos de Sprite têm sido colocado no queimador de volta, e provavelmente só será usado no AGDX Studio agora.
- 3) Algum código foi adicionado que lhe permitirá exportar todos os dados Sprite de forma rápida e fácil. Falo sobre isso mais tarde.
- 4) Assim como os ponteiros, mais informações sobre os locais da tabela Jump e mapa também foram adicionadas, novamente para ajudar na importação e exportação.
- 5) A tela de tamanho da janela agora exibe a largura da janela atual e altura, e CTRL-Z será encerrado sem salvar as alterações.
- 6) A tabela de saltos agora mostra o local de memória da "seção" atual da tabela. Poking este valor com o número 99 permitirá que você trunchar a tabela de salto, significando que pode ser usada para saltos menores, ou outros efeitos tais como um impulso ou um salto.
- 7) O editor de efeitos sonoros foi removido para dar espaço para outro código. No entanto, os sons ainda podem ser navegados e ouvidos e incluídos nos jogos, e um endereço de memória é fornecido para permitir que os sons sejam importados de versões anteriores. Os sons são todos 41 bytes em comprimento.
- 8) Foi encontrado um bug que aconteceu quando um som foi excluído depois de retomar ao menu principal. Isso estava causando corrupção para as mensagens. Problemas ainda podem persistir com isso, mas parece ser mais estável do que antes. No passado, muitas pessoas simplesmente evitaram usando sons porque causou bugs. Pelo menos um deles foi eliminado.
- 9) No editor de Sprite, T, Y e U ainda são usados para girar no sentido horário, deslocar para a direita e nudge para cima. No entanto, agora, usando CTRL-T, CTRL-Y e CTRL-U, agora é possível girar no sentido anti-horário, deslocar para a esquerda e deslocar para baixo.

10) Também no editor Sprite são três novos comandos de pasta ou colar (CTRL-I), que irá colar um Sprite em outro, efetivamente mesclando-os. XOR colar (CTRL-O), que irá colar um Sprite sobre outro como AGD faz, o que é útil se você quiser criar um objeto estático ou Sprite que pode ser colocado em um fundo. E colar (CTRL-P) efetivamente funciona como uma ferramenta de mascaramento, permitindo que você 'cortar' uma forma. Isso tem o potencial para ser usado para criar máscaras de Sprite no futuro. Haverá um vídeo com todas as várias opções de colagem como eles são, bastante extensa agora.

11) As teclas de atalho no menu principal tiveram que ser removidas para fazer o quarto para o outro código.

#### Alterações feitas na versão 4H:

- Editor de som agora usa o padrão AGDX e '=' para próximo/anterior
- Endereço de memória de som mostrado na tela. Cada som usa 40 bytes, assim a importação e a exportação são muito fáceis.
- Editor de bloco simplificado, agora mostra o endereço para o atributo, bem como o valor do atributo.
- Nomes de tipo de bloco reduzidos (por exemplo, 'Plat', 'watr', 'cust', etc.) para dar espaço para substituir o recurso de bloco.
- Editor de tela agora inclui 'substituir' bloco. Pressionar CTRL-R substituirá todas as instâncias do bloco, atualmente o cursor com o bloco selecionado no momento. Pode ser desfeito com CTRL-Z

#### Alterações na versão 4I:

Adicionado a grade de atributo para o menu da tela. Pressione 'a' para navegar no modo de atributo, qualquer tipo de edição retornará ao modo normal. Observe que você não pode manter o modo de atributo durante a edição, que era apenas um pouco muito espaço de código.

Template 'controles Pacman' foi removido como era algo de uma novidade. Ainda pode ser importado como um binário, se necessário.

Atualizada a exibição do ponteiro para usar realce em vez de setas. O ponteiro Sprite agora inclui dados de quadro e o ponteiro de som também está incluído. Tamanho dos dados para blocos para eventos incluídos para exportação prontos para AGDX5.

Adicionado um novo comando no editor de tela 'e' permitirá que você edite imediatamente o bloco do cursor.

## AGDX Keyboard shortcut guide version 4Ke

Sprite Editor	
Navigation / Creation	
Create Sprite	X
Delete Sprite	D
Insert Frame	F
Remove Frame	R
Previous Sprite	-
Next Sprite	=
Previous Frame	1
Next Frame	2
Copy Sprite	M
Paste Sprite	K
Block Copy	CTRL-M
Block Paste	CTRL-K
XOR Paste	CTRL-X
OR Paste	CTRL-C
AND Paste	CTRL-V
Toggle Grid	G
Open Worksheet	S
Sprite Ink	I
Sprite Paper	P
Sprite Bright	B
Cell Ink	CTRL-I
Cell Paper	CTRL-P
Cell Bright	CTRL-B
Reset Attributes	CTRL-O

Editing	
Clear	C
Flip Horizontal	H
Flip Vertical	V
Rotate Clockwise	T
Rotate Anti-Clockwise	CTRL-T
Shift Right	Y
Shift Left	CTRL-Y
Shift Up	U
Shift Down	CTRL-U
Invert	O

Worksheet Funcions	
Load Current Sprites To Screen	S
Load From Tap (Press Y To Confirm)	L
Toggle Attributes	A
Refresh (After Binary Load to 49152)	R
Copy Sprite	M
Select Sprite (If Possible)	Space/0

Object Editor	
Navigation / Creation	
Jump To Block Editor	CTRL-1
Jump To Screen Editor	CTRL-2
Jump To Sprite Editor	CTRL-3
Jump To Positions Editor	CTRL-4
Create Object	X
Delete Object	D
Previous Object	-
Next Object	=
Copy Object	M
Paste Object	K
Block Copy	CTRL-M
Block Paste	CTRL-K
Select Previous Screen	1
Select Next Screen	2
Place Object	P
Toggle Grid	G
Open Worksheet	S

Editing	
Clear	C
Flip Horizontal	H
Flip Vertical	V
Rotate Clockwise	T

Object Placement	
Go To Previous Screen	-
Go To Next Screen	=
Toggle Other Objects	O
Precision On / Off	P

## AGDX Keyboard shortcut guide version 4Ke

<b>Block Editor</b>	
Navigation <u>I</u> Creation	
Jump To Screen Editor	CTRL-2
Jump To Sprite Editor	CTRL-3
Jump To Positions Editor	CTRL-4
Jump To Object Editor	CTRL-5
Step Back One Block	1
Step Forward One Block	2
Step Backward Ten Blocks	3
Step Forward Ten Blocks	4
Previous Block Type	-
Next Block Type	=
Create Block	X
Copy Block	M
Paste Block	K
Sprite Paste (4 Blocks)	CTRL-K
View Block Palette	S
View Blocks As Image	CTRL-B

<b>Screen Editor</b>	
Navigation <u>I</u> Creation	
Jump To Block Editor	CTRL-1
Jump To Sprite Editor	CTRL-3
Jump To Positions Editor	CTRL-4
Jump To Object Editor	CTRL-5
Previous Screen	-
Next Screen	=
Create Screen	X
Clear Screen	C
Delete Screen	D
Copy Screen	M
Paste Screen	K
Toggle Attributes	A
Overlay Objects And Sprites	O
Query Screen Size In Bytes /	
Quick Save	Q
View Block Palette	S
Revsrt To Last Sa\ed Version	CTRL-Z

<b>Editing</b>	
Clear	C
Ink	I
Paper	P
Bright / Flash / ULA+	B
Invert Pixels	O
Invert Colours	CTRL-O
Flip Horizontal	H
Flip Vertical	V
Rotate Clockwise	T
Rotate Anti-Clockwise	CTRL-T
Shift Right	Y
Shift Left	CTRL-Y
Shift Up	U
Shift Down	CTRL-U

<b>Editing</b>	
Select Block On Screen	B
Select And Edit Block On Screen	E
Toggle Fast Draw Mode	F
Undo Last Action	U
Place Block Zero	Z
Copy A 4X4 Tile Of 16 Blocks	CTRL-C
Paste A 4X4 Tile	CTRL-V
Paste A 4X4 Tile With No Block Zero	CTRL-B
Replace On Screen Block With Selected	CTRL-R

<b>Placement Editor</b>	
Jump To Block Editor	CTRL-1
Jump To Screen Editor	CTRL-2
Jump To Sprite Editor	CTRL-3
Jump To Object Editor	CTRL-5
Next Placement	Q
Previous Screen	-
Next Screen	=
Create New Placement	X
Delete Current Placement	D
Change Sprite Type	T
Change Image	I
Precision On / Off	P

<b>Font Editor</b>	
Previous Char	-
Next Char	=
Copy Char	M
Paste Char	K
Clear Char	C
View ROM font	F

<b>Script Editor</b>	
Exit With Save	CTRL-A
Exit Without Saving	CTRL-Q
Go To Start Of Line	CTRL-W
Go To End Of Line	CTRL-E
Cut / Delete Line	CTRL-Y
Paste / Insert Line	CTRL-U
Copy To Windows Clipboard *	CTRL-S
Toggle Caps Lock	SHIFT-2
Delete Next Char	SHIFT-3
Toggle Overtyp	SHIFT-4

\* Has To Be Set Up In Spectaculator



## MENU AGD PARA INICIANTEs - 09 Set

Tutorial de Luca Bordoni - método simples e avançado

### INTRODUÇÃO

Um pedido frequente dos entusiastas da AGD é a confecção de um menu personalizado de forma a trazer um toque profissional ao seu projeto.

Este tutorial trata dessas etapas progressivas:

- 1 - criar um menu simples usando diretamente os comandos AGD;
- 2 - análise do carregador básico gerado automaticamente pelo AGD;
- 3 - criar um menu com seleções piscando por meio de um carregador básico personalizado;
- 4 - usando a ferramenta de compressão de tela Mikropol;
- 5 - como incluir gráficos de fundo em um menu personalizado.

Os primeiros três passos são argumentos muito básicos; entretanto, como sabemos, muitos usuários de AGD são inexperientes e precisam começar do básico para esclarecer qualquer dúvida.

VAMOS PREPARAR NOSSAS FERRAMENTAS ...

Aqui estão algumas sugestões sobre as ferramentas que devemos usar:

**Escolha um emulador ZX Spectrum, Óbvio.**

Trabalhar em uma máquina real pode significar muito mais tempo para gastar 😊  
Como estou interessado apenas na máquina clássica de 48K (sem ULA + ou outras melhorias), minha escolha pessoal é "Spectaculador". <https://www.spectaculador.com/>

**Um bom editor de imagens.**

Não é obrigatório porquê de qualquer maneira precisamos do ZX Paintbrush, mas um editor de imagens profissional pode resultar muito útil para criar gráficos da maneira mais confortável (minha preferência vai para Corel PhotoPaint).

\* Lembre-se sempre de que o tamanho da tela do ZX Spectrum tem 256x192px de largura, dividido em 32 colunas e 24 linhas de blocos de caracteres únicos de 8x8px, que por sua vez têm dois atributos principais de cor, um para a tinta e outro para o fundo do papel.

**ZX Paintbrush**

<http://www.zx-modules.de/zxpaintbrush/zxpaintbrush.html>

Indispensável. Este editor de tela foi especialmente projetado para gerenciar os gráficos do ZX Spectrum. Importando uma imagem PNG ou GIF, talvez criada com outro editor de imagens profissional, ou apenas criando nossos gráficos a partir daqui, temos a possibilidade de converter/salvar o trabalho em um arquivo .TAP, que é necessário para o compressor de tela.

#### **Mikropol Screen Compressor Plus**

<http://www.worldofspectrum.org/infoseekid.cgi?id=0012062>

É a forma como podemos converter uma tela gráfica em um bloco de código de máquina compactado, para evocá-lo a qualquer momento que quisermos (ou apenas uma vez, por exemplo, para um fundo do menu).

#### **BASin**

<http://everychildcancode.org/basin/>

Uma ferramenta muito útil para gerenciar e montar rapidamente diferentes blocos de programa, como o carregador personalizado, uma tela de carregamento e a parte do código de máquina AGD. Portanto, é necessário aqui apenas para uma parte mínima de sua funcionalidade total.

## **1. COMO CRIAR UM MENU SIMPLES NO AGD**

### **Como funciona o comando CONTROLMENU**

O AGD 4.6 utiliza o comando CONTROLMENU, para usar na seção INTRO / MENU, onde o código é executado antes dos demais eventos do jogo.

O comando CONTROLMENU gera uma tela em branco em estado de pausa, aguardando que o jogador escolha entre as seguintes opções:

1 - Joystick Kempston;  
2 - Joystick Sinclair;  
(qualquer outro valor) - Teclado.

Pressionando a tecla relativa, AGD carrega no endereço de memória 32005 os valores que irão ativar os controles escolhidos:

endereço de memória, valor: 32005,1 -> Joystick Kempston  
(equivalente a POKE 32005,1 em Basic)

endereço de memória, valor: 32005,2 -> Joystick Sinclair  
(equivalente a POKE 32005,2 em Basic)

Qualquer outro valor diferente de 1 ou 2, irá ativar os controles do teclado AGD.

A tela está em branco! O que eu tenho que fazer?

Portanto, o comando CONTROLMENU gera uma tela em branco ...

Mesmo que as três opções acima sejam forçadas, o texto é delegado a nós.

Em primeiro lugar, temos que criar algumas linhas de texto através da seção TEXT (uma mensagem para cada linha), por exemplo:

Mensagem # 0 - "MY GAME TITLE"  
Mensagem # 1 - "1 KEYBOARD"  
Mensagem # 2 - "2 KEMPSTON"  
Mensagem # 3 - "3 SINCLAIR"

... então, indo para a seção INTRO / MENU, podemos escrever o código para imprimir as mensagens antes do CONTROLMENU que ativa a pausa de pressionamento de tecla.

Exemplo: na seção INTRO / MENU, podemos escrever ...

```
COR 71
CLS
DEIXAR LINHA = 1
DEIXAR COLUNA = 9
COR 66
MENSAGEM 0
DEIXAR LINHA = 9
DEIXAR COLUNA = 11
COR 70
MENSAGEM 1
DEIXAR LINHA = 12
DEIXAR COLUNA = 11
COR 70
MENSAGEM 2
DEIXAR LINHA = 15
DEIXAR COLUNA = 11
COR 70
MENSAGEM 3
CONTROLMENU
```



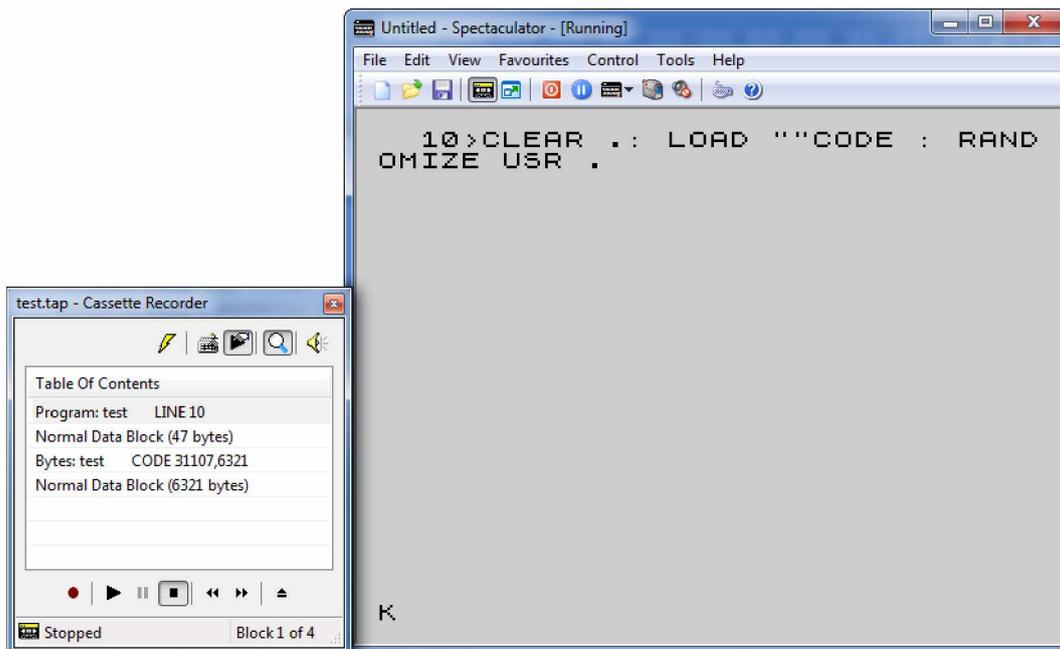
E aqui está um menu AGD simples, sem gráficos.

Temos que considerar que esta não é uma tela de jogo, então não podemos incluir gráficos da mesma forma que criamos um mapa de jogo. No entanto, poderíamos criar gráficos personalizados redesenhando o conjunto de caracteres (por exemplo, transformando letras minúsculas ou símbolos por meio da opção CHARACTER SET no AGD) e usar o mesmo método de mensagens, embora não seja uma prática elegante.

Caso contrário, vamos continuar aprendendo como podemos obter melhores resultados.

## 2. O CARREGADOR BASIC AGD

Cada vez que salvamos nosso projeto, o programa pergunta se queremos que o AGD gere um carregador automático. Analisando o código básico, encontraremos esta linha ...



Por motivos de economia de memória, o AGD oculta os endereços de memória CLEAR e RANDOMIZE. Simplesmente não ligue para isso: vamos escrever nosso carregador básico

personalizado em alguns instantes.

### O significado de CLEAR / RANDOMIZE USR

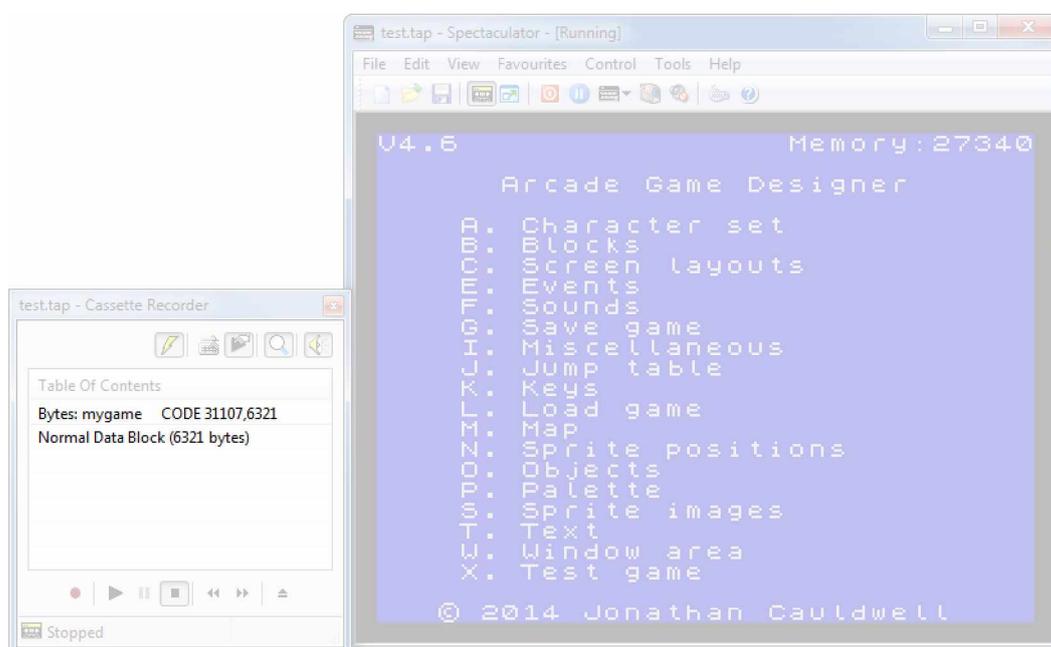
O comando Basic CLEAR reserva uma área de memória para o programa de código de máquina que vamos carregar, começando do próximo byte (por exemplo, CLEAR 29999 irá limpar e reservar a memória a partir do endereço 30000).

O comando RANDOMIZE USR executa o programa de código de máquina previamente carregado com LOAD "" CODE.

No AGD, qualquer jogo salvo sempre será executado usando RANDOMIZE USR 32000, mesmo se o bloco de código de máquina salvo pelo AGD começar em um endereço de memória inferior.

Para entender o valor do comando CLEAR, basta verificar o endereço inicial, visível após um salvamento.

Exemplo: salvamos nosso projeto AGD. Usando o gravador de fita virtual do emulador, vemos o endereço inicial 31107



... então, neste caso, nosso carregador deve ser assim:

```
CLEAR 31106: LOAD "" CODE: RANDOMIZE USR 32000
```

Aqui está! O mistério dos "pontos" no carregador AGD é finalmente desvendado; agora podemos começar a programar nosso carregador personalizado!

Em primeiro lugar, temos que deletar o código dentro da seção INTRO / MENU, porque não precisamos mais do AGD para gerenciar o menu.

Aqui está um exemplo de carregador personalizado, incluindo uma tela de carregamento:

```
1 CLEAR 31106: BORDER 0: PAPER 0: INK 0: CLS: PRINT BRIGHT 1; INK 7; AT 9,6;"THE GAME IS  
LOADING"; AT 12,10; INVERSE 1;"PLEASE WAIT": PRINT AT 0,0: LOAD ""SCREEN$: POKE  
23739,111: LOAD ""CODE: RANDOMIZE USR 32000
```

O POKE 23739,111 é um truque fofo para evitar que o texto "Bytes" apareça na tela gráfica carregada.

### 3. CARREGADOR PERSONALIZADO E SELEÇÕES DE MENU PISCANDO

Exemplo de um carregador personalizado usando uma tela de carregamento e um menu personalizado:

```
1 CLEAR 31106: BORDER 0: PAPER 0: INK 0: CLS: PRINT BRIGHT 1; INK 7; AT 9,6;"THE GAME IS
LOADING"; AT 12,10; INVERSE 1;"PLEASE WAIT": PRINT AT 0,0: LOAD ""SCREEN$: POKE
23739,111: LOAD ""CODE
2 BORDER 0: PAPER 0: INK 0: CLS: PRINT BRIGHT 1; INK 2; AT 1,9; "MY GAME TITLE"
3 PRINT BRIGHT 1; INK 6; AT 9,11; "1 KEYBOARD";AT 12,11; "2 KEMPSTON";AT 15,11; "3
SINCLAIR"
4 PRINT BRIGHT 1; INK 7; AT 19,11; "0 START"
10 LET x=9: PRINT AT x,11; BRIGHT 1; FLASH 1; OVER 1; INK 8;"(10 spaces)": POKE 32005,0
20 IF INKEY$="1" AND x<>9 THEN GO SUB 70: LET x=9: POKE 32005,0: GO SUB 80
30 IF INKEY$="2" AND x<>12 THEN GO SUB 70: LET x=12: POKE 32005,1: GO SUB 80
40 IF INKEY$="3" AND x<>15 THEN GO SUB 70: LET x=15: POKE 32005,2: GO SUB 80
50 IF INKEY$="0" THEN RANDOMIZE USR 32000: GO TO 2
60 GO TO 20
70 PRINT AT x,11; BRIGHT 1; FLASH 0; OVER 1; INK 8;"(10 spaces)": BEEP .033,24: RETURN
80 PRINT AT x,11; BRIGHT 1; FLASH 1; OVER 1; INK 8;"(10 spaces)": RETURN
```

O código BASIC se explica. Apenas uma nota sobre o método: aqui usamos o truque OVER 1 e INK 8. Isso significa que os espaços em branco piscando se moverão sobre um texto de fundo, preservando o atributo de cor de fundo, nas coordenadas fornecidas.

Depois de RANDOMIZE USR 32000, há o comando GO TO 2, que retorna ao menu personalizado assim que a reprodução termina.

Lembre-se de definir o valor CLEAR correto, dependendo do endereço inicial do seu projeto.

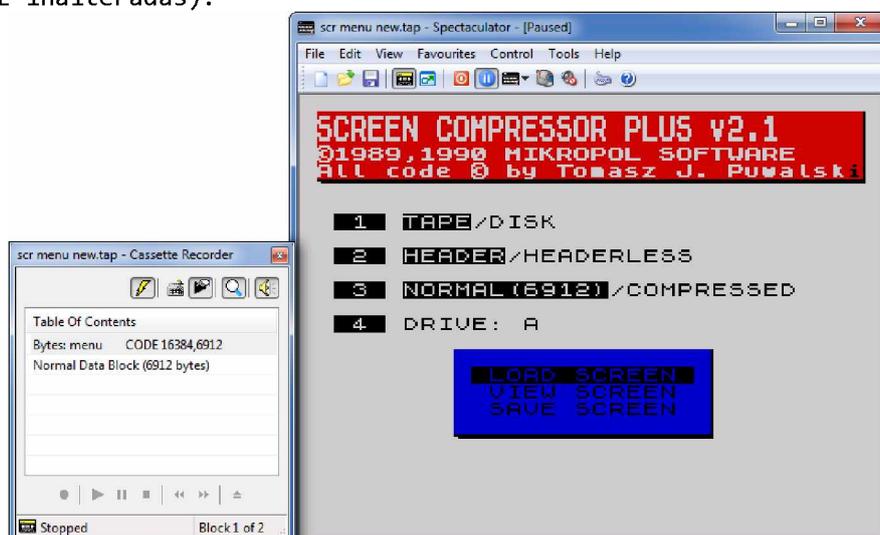
### 4. MIKROPOL SCREEN COMPRESSOR PLUS

Entre muitos outros utilitários semelhantes disponíveis, eu prefiro fortemente este. A única coisa especial que temos que fazer é carregar uma tela salva como um arquivo TAP.

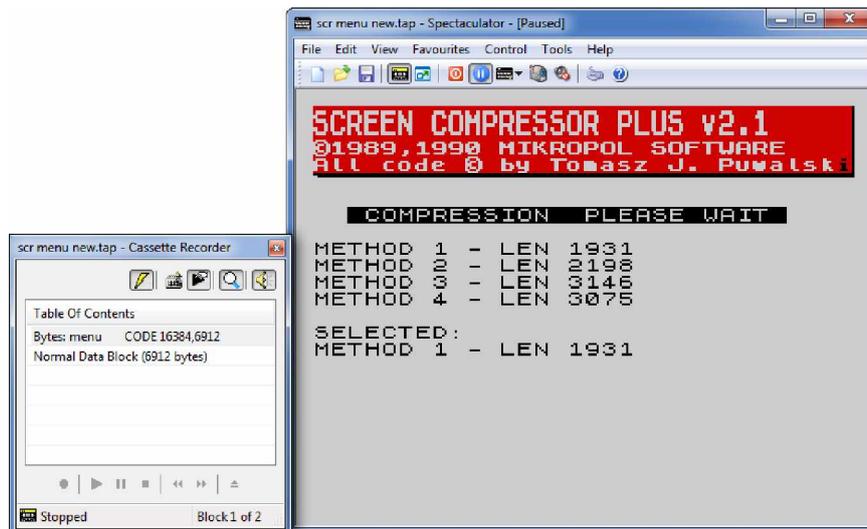
Minha imagem é um GIF ou PNG, como posso obter um arquivo TAP?

Através do ZX Paintbrush, basta abrir sua imagem (256x192px, lembra?), E selecionar SALVAR COMO "Arquivo de Fita Simples". Feito!

Agora, vamos carregar a tela TAP em nosso utilitário compressor (deixe as opções TAPE / HEADER / NORMAL inalteradas):



O programa irá elaborar quatro compressões diferentes



... vamos escolher a mais leve: nada mal, comprimimos os gráficos de 6912 (o comprimento padrão da tela) a 1931 bytes!

Quando voltarmos ao menu principal, selecione TAPE / HEADER / COMPRESSED e a opção SAVE e vamos salvar o arquivo compactado como um novo arquivo TAP.

O programa salvará nosso novo arquivo no endereço de memória 50000 por padrão.

Boa. Podemos usar este arquivo como está, armazenado naquele endereço, ou escolher salvá-lo em outro endereço de memória.

**Como pode ser endereçado novamente um bloco de código de máquina?**

Temos uma tela compactada salva em um bloco de código de máquina TAP, por exemplo, no endereço 50000, 1931 bytes.

Verificamos nosso jogo e precisamos mover a rotina da tela compactada do endereço 50000 para (por exemplo) 45000. Como podemos fazer? Muito fácil.

Primeiro, vamos observar o comprimento dos bytes (1931 em nosso exemplo). Em seguida, carregue o arquivo no gravador de fita virtual do emulador e digite:

```
LOAD "" CODE 45000
```

... Desta forma, o arquivo será carregado no endereço desejado. Em seguida, salve-o novamente em um novo arquivo TAP:

```
SAVE CODE "mycode" 45000,1931
```

É por isso que tivemos que tomar nota do comprimento.

Feito! Agora temos uma tela compactada, pronta para ser chamada em um endereço de memória definido. Mágica, não é!?

**Importante: sempre verifique a faixa de RAM disponível!**

Sempre temos que considerar os limites de memória, a fim de estabelecer adequadamente onde armazenar nossas rotinas de código de máquina personalizadas.

Considerando um 48K ZX Spectrum, um projeto AGD não pode exceder o endereço 64767.

Portanto, se nosso jogo AGD começa no endereço 31107 e tem um peso de 10000 bytes, poderíamos colocar nossas rotinas de código de máquina começando no endereço 41107 e dentro do endereço limite superior 64767.

## 5. MENU COM GRÁFICOS DE FUNDO

Aqui está uma explicação autoexplicativa Lista BASIC para nosso menu final, incluindo a chamada para gráficos de fundo e seleções piscantes:

```
1 CLEAR 31106: BORDER 0: PAPER 0: INK 0: CLS: PRINT BRIGHT 1; INK 7; AT 9,6;"THE GAME IS
LOADING"; AT 12,10; INVERSE 1;"PLEASE WAIT": PRINT AT 0,0: LOAD ""SCREEN$: POKE
23739,111: LOAD ""CODE
2 BORDER 0: PAPER 0: INK 0: CLS: RANDOMIZE USR 45000
10 LET x=9: PRINT AT x,11; BRIGHT 1; FLASH 1; OVER 1; INK 8;"(10 spaces)": POKE 32005,0
20 IF INKEY$="1" AND x<>9 THEN GO SUB 70: LET x=9: POKE 32005,0: GO SUB 80
30 IF INKEY$="2" AND x<>12 THEN GO SUB 70: LET x=12: POKE 32005,1: GO SUB 80
40 IF INKEY$="3" AND x<>15 THEN GO SUB 70: LET x=15: POKE 32005,2: GO SUB 80
50 IF INKEY$="0" THEN RANDOMIZE USR 32000: GO TO 2
60 GO TO 20
70 PRINT AT x,11; BRIGHT 1; FLASH 0; OVER 1; INK 8;"(10 spaces)": BEEP .033,24: RETURN
80 PRINT AT x,11; BRIGHT 1; FLASH 1; OVER 1; INK 8;"(10 spaces)": RETURN
```

Neste exemplo, consideramos esses elementos variáveis:

- O valor CLEAR 31106 assume que o jogo começa em 31107;
- A tela gráfica compactada é salva em 45000;
- Dez espaços em branco para seleções piscantes, que correspondem ao texto (caracteres ou blocos gráficos) posicionado nas coordenadas escolhidas;
- As opções de texto são substituídas aqui pelo comando RANDOMIZE USR 45000: não há razão para imprimir texto, pois podemos incluí-lo diretamente no fundo compactado! Se você não considerou esta solução brilhante e não quer modificar seu fundo gráfico, então verifique a versão anterior e inclua os comandos PRINT após RANDOMIZE USR 45000.

### NOTA FINAL

Imagine a possibilidade de transformar nosso carregador básico em um bloco de código de máquina (os especialistas conhecem esta operação, isso pode ser feito através de um utilitário compilador). Seria mais um toque profissional, por exemplo, o jogador não será capaz de INTERROMPER o programa pressionando "Shift + Break Space" (que é um recurso básico típico) durante o evento do menu!

Falaremos sobre o método do compilador em outro tutorial.

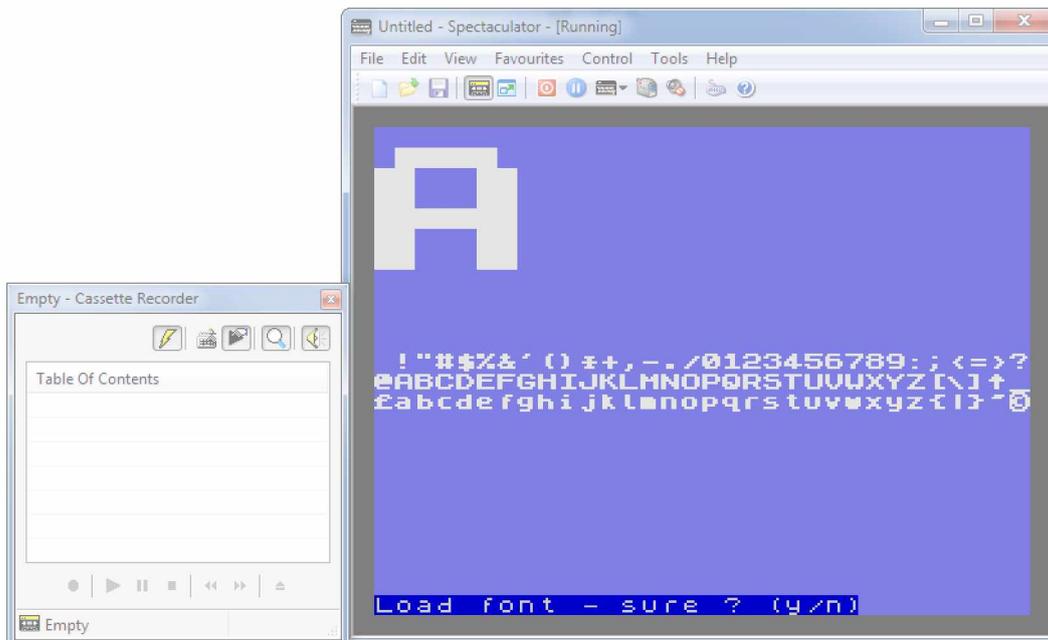


## CONJUNTO DE CARACTERES AGD - 10 Set

Tutorial de Luca Bordonni

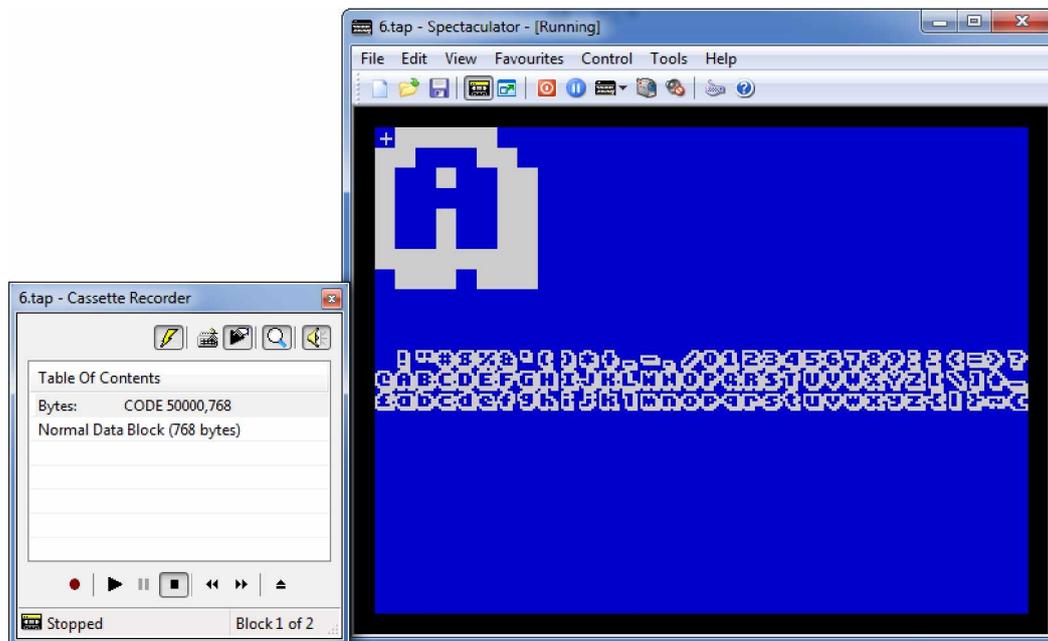
Esta página é voltada para iniciantes em AGD, alguns argumentos podem parecer óbvios para programadores experientes.

A primeira opção AGD é sobre o conjunto de caracteres, que é uma ferramenta para modificar cada caractere facilmente. No entanto, muitos usuários gostariam de carregar um arquivo externo, pressionando "L" no menu de conjunto de caracteres AGD.



### Especificações de memória

O conjunto de caracteres ZX Spectrum tem 96 caracteres, 8 bytes cada, ou seja, um comprimento total de 768 bytes. AGD armazena o conjunto de caracteres a partir do endereço de memória 31232. A boa notícia é que, mesmo se você tiver um arquivo de conjunto de caracteres pronto para uso localizado, por exemplo, em 50000, a função de carregamento irá redirecionar automaticamente o código para corresponder às especificações AGD. Aqui está o resultado ...



...Então, se quisermos editar e exportar nossa nova fonte, talvez através da opção do emulador, agora sabemos as referências da memória AGD: 31232,768.

## COLEÇÃO DE CONJUNTOS DE CARACTERES

Divirta-se experimentando esses arquivos tap ...

- Conjunto de caracteres do Atic Atac
- Conjunto de caracteres de Commando
- Conjunto de caracteres Frank N.Stein
- Conjunto de caracteres Shaded

### Sistema

- Conjunto de caracteres padrão AGD
- Conjunto de caracteres padrão do ZX Spectrum



## RESPAWN E INVULNERABILIDADE - 21 Set

Tutorial de Luca Bordonni

Este tutorial é voltado para programadores AGD que já estão envolvidos em um projeto; é sugerido ler e aprender a documentação oficial do AGD primeiro.

Uma pergunta comum ao desenvolver jogos AGD: “se meu sprite morrer, gostaria de deixar o respawn no ponto exato da morte, e o jogador não deve ser morto por alguns segundos após voltar à vida”

### A solução: trabalhar com variáveis

Assumindo que ainda temos alguma variável disponível, a ideia é configurar algumas variáveis para clonar a posição horizontal e vertical do jogador, e uma terceira variável para gerenciar um cronômetro para a invulnerabilidade.

Em primeiro lugar, como visto nos tutoriais em vídeo AGD de Paul Jenkinson, precisamos configurar um tempo no **MAIN LOOP event**, útil para evitar movimentos muito rápidos:

```
IF A = 1
LET A = 0
ELSE
ADD 1 TO A
ENDIF
```

Agora vamos na **GAME INITIALISATION** para configurar nossas novas variáveis. Precisamos de uma variável para o temporizador de invulnerabilidade, digamos "I", e outras duas variáveis para clonar a posição X/Y do jogador, por ex., "O" e "P":

```
LET A = 0
...
LET I = 0 (sem invulnerabilidade quando a primeira jogada começa)
LET O = 80 (ou outro valor, clona a posição do pixel X do jogador)
LET P = 120 (ou outro valor, clona a posição do pixel Y do jogador)
```

Então, vamos entrar no evento **PLAYER 0** (sprite 0), onde usaremos normalmente as variáveis X/Y.

No final da seção, vamos escrever:

```
...
LET O = X
LET P = Y
```

Agora temos duas variáveis sempre monitorando a posição do nosso jogador.

No evento **INITIALISE SPRITE**, vamos escrever:

```
IF TYPE 0
LET X = 0
```

```
LET Y = P
ENDIF
```

Isso significa que o jogador aparecerá nas coordenadas clonadas, que correspondem aos valores X/Y no início do jogo!

### Adicione a invulnerabilidade

O código a seguir é o trecho que teremos que incluir em nossa rotina de eliminação customizada. A maneira como podemos gerenciar uma colisão entre o jogador e um inimigo mortal ou bloco pode variar, e aqui não vamos discutir a rotina de matança; em vez disso, a maneira de incluir um status de invulnerabilidade por um tempo limitado.

Então, dentro da rotina de morte de nosso sprite, por exemplo, na condição de COLLISION entre o jogador e o inimigo, vamos dar um valor ao temporizador de invulnerabilidade:

(iniciar a condição "IF COLLISION")

```
...
IF I > 0
ELSE           (se o cronômetro tiver um valor maior que 0, não fazer nada)
LET I=25      (ou outro valor, corresponde a alguns segundos de invulnerabilidade)
KILL
ENDIF
...
```

(fim da condição " IF COLLISION ")

Observe que a instrução LET I=25 poderia ser incluída diretamente no evento AGD KILL, caso estejamos gerenciando nossa rotina lá. O objetivo é definir o cronômetro de invulnerabilidade antes de reiniciar o jogo.

Então, no evento PLAYER 0, deixe o código começar desta forma:

```
IF I > 0
GETRANDOM 7
ADD 65 TO RND
SPRITEINK RND
IF A = 0
SUBTRACT 1 FROM I
ENDIF
ELSE
LET I = 0
SPRITEINK 71
ENDIF
```

Esta é uma boa dica para mudar a cor do jogador para que ele saiba que ele é invulnerável





## A CHAMADA ASM - 21 Set

Tutorial de Luca Bordoni

Este tutorial é voltado para programadores AGD que já estão envolvidos em um projeto; é sugerido ler e aprender a documentação oficial do AGD primeiro.

Aqui, tentaremos nos aprofundar em um tópico importante já discutido no fórum oficial da AGD pelo autor Jonathan Cauldwell. Aprender as chamadas ASM, significa trazer um toque profissional a qualquer trabalho AGD, pois poderemos incluir rotinas de código de máquina externas (por exemplo, geradas por outros programas), como efeitos gráficos e sonoros. Esse é o único propósito do ASM, o chamado de Ferramenta.

### COMO FUNCIONA UMA CHAMADA ASM EM AGD

Uma chamada ASM (ASM significa "Montagem") não é outro senão uma instrução AGD que produz o mesmo efeito do Comando Sinclair Basic USR: evoca uma rotina previamente armazenada em um endereço de memória especificado.

Devemos saber que a sintaxe AGD não pode lidar com valores maiores que 255... mas aqui, temos que chamar os endereços de memória que podem atingir o valor de 64000 e além! Então, se em Sinclair Basic simplesmente escrevermos "RANDOMIZE USR 50000", como poderíamos dar uma instrução semelhante em AGD?!

A resposta é: por meio de três comandos ASM, cada um manipulando valores de 8 bits de 0 a 255. Juntos, esses três comandos têm exatamente o mesmo efeito de uma instrução RANDOMIZE USR NN em Basic, onde NN é o endereço de memória calculado por o segundo e o terceiro ASM.

Tudo o que precisamos fazer é aprender a maneira como esse cálculo funciona, e o segredo das chamadas ASM ficará finalmente claro.

### OS TRÊS VALORES ASM

Vamos supor que temos um efeito sonoro que armazenamos no endereço 50000 e queremos criar a chamada ASM em nosso projeto AGD. Os comandos ASM devem ser escritos no ponto exato onde queremos que o som seja executado (assim como acontece com um comando BEEP), e o código será:

```
...  
ASM 205  
ASM 80  
ASM 195  
...
```

A explicação é muito simples. O primeiro parâmetro é um valor fixo: cada número de 8 bits está associado a uma instrução Assembly específica (conforme listado no Apêndice A do Manual de Programação BASIC), e o número 205 retorna a instrução Assembly "chamar NN". É por isso que o primeiro comando ASM sempre deve ser definido como 205, então é o mais

fácil de todos 😊

Agora vamos dividir o endereço inicial por 256 e anotar o resultado e o restante. Em nosso exemplo, o cálculo será  $50000 \div 256 = 195$  com um restante de 80.

Esses números parecem familiares? Certo, eles são os valores para nossa chamada ASM, que serão usados pelo AGD para construir o endereço 50000.

AGD ASM significa 'chamar NN' em Montagem ou 'RANDOMIZE USR NN' em BASIC.

...

ASM 205 (1º valor, significa "chamar NN" em Montagem)

ASM 80 (2º valor, o restante da divisão de  $50000 \div 256$ )

ASM 195 (3º valor, o resultado inteiro da divisão de  $50000 \div 256$ )

...

Estamos prontos para incluir nossos "efeitos especiais" em nosso jogo AGD! Apenas não perca algumas notas importantes no próximo capítulo.

### SIGA AS ETAPAS PROGRESSIVAS COMO UMA BOA PRÁTICA

Se um projeto AGD com comandos ASM for testado, o programa irá travar, perdendo todo o progresso de codificação possível! É por isso que é muito importante salvar à parte um projeto AGD limpo, sem comandos ASM (ou comandos ASM com valores zero).

1 – Conclua o projeto. Depois de concluído o código, é possível verificar o espaço de memória disponível e escolher os endereços de memória iniciais para as rotinas de código de máquina que queremos integrar (consulte o capítulo de referências de memória a seguir).

2 – Verifique a memória disponível e prepare seus arquivos de rotina externos. Anote o endereço de memória a ser chamado e faça os cálculos ASM para usar no projeto AGD.

3– Escreva os comandos ASM dentro do projeto AGD nos pontos onde as rotinas devem ser chamadas, mas defina todos os comandos ASM para zero. Desta forma, você preservará um projeto de backup que ainda pode ser testado: os comandos ASM 0 não afetarão a execução do teste AGD. Salve este projeto separadamente.

4 – Abra o projeto salvo anteriormente e substitua os comandos ASM 0 pelos valores calculados anteriormente. Em seguida, salve o projeto como um novo arquivo. Este projeto não pode mais ser executado (testado) a partir do AGD (a menos que os valores ASM sejam alterados novamente para zero).

Agora o código (com as chamadas ASM corretas) está pronto para ser salvo do AGD. A partir do carregador básico, teremos que carregar os vários blocos de código de máquina, ou seja, o salvamento final do AGD e as rotinas de código de máquina.

Os programadores qualificados do ZX Spectrum serão capazes de mesclar as rotinas externas junto com o código do jogo AGD principal, a fim de carregar um arquivo único no carregador.

### REFERÊNCIAS DE MEMÓRIA

Considerando um projeto AGD concluído para um 48K ZX Spectrum, sempre leve em consideração estes parâmetros:

Referências de memória mais baixas:

– um intervalo de memória começando no endereço 23552 é reservado para Variáveis BASICs do Sistema; para evitar possíveis travamentos, sugere-se colocar as rotinas não antes do endereço 24600;

– a menor memória utilizável começa em torno do endereço 24600 até o endereço inicial do projeto AGD ( -1 ).

Referências de memória superior :

– um projeto AGD não pode exceder o endereço 64767 ;

– a memória livre superior começa no endereço final do projeto AGD ( +1 ) até 64767 .

Ainda sobre a memória superior, existem algumas informações úteis das instruções oficiais da AGD:

No topo da RAM, os últimos 768 bytes de 64768 a 65535 são usados como uma área de mapa de colisão fictícia para distinguir entre diferentes tipos de blocos - paredes, escadas, espaço vazio e assim por diante. No final do jogo, há uma área de 300 bytes usada para o motor de partículas caso você decida empregar lasers, trilhas de vapor ou explosões. Este buffer não existe nos motores de quebra-cabeça ou efeitos.

#### CALCULADORA AGD ASM

Insira um endereço de memória utilizável (entre 24600 e 64767 ) e obtenha seus valores ASM em tempo real!



OBTER VALORES



## ARMAZENAR ASM NO CONJUNTO DE CARACTERES AGD - 5 Abr

Tutorial de Luca Bordonni, obrigado a David Saphier e Allan Turvey

É recomendável ler e compreender os tutoriais sobre o conjunto de caracteres AGD e as chamadas ASM antes de continuar aqui.

As rotinas de montagem injetam “efeitos especiais” em nossos projetos AGD. Mesmo sem ser um especialista em montagem, pode haver a necessidade de incluir rotinas de código de máquina externas, como efeitos sonoros, gráficos compactados ou qualquer outro tipo de aprimoramento, e testar o resultado.

Como dito nas chamadas ASM tutorial, usar valores ASM dentro de nosso projeto AGD fará com que o teste do programa trave, porque AGD chama o endereço de memória apontado pelos valores ASM, que devem ser atualizados após a conclusão do projeto. Portanto, quando a chamada de memória não encontra a rotina naquele endereço, ou quando uma rotina importada colide com o kernel principal do AGD, o projeto irá travar.

Bem, a boa notícia é que é possível armazenar pequenas rotinas de montagem na área de memória reservada ao conjunto de caracteres. Então, chamando esse endereço através dos comandos ASM (consulte o tutorial de chamadas ASM), a fase de teste funcionará perfeitamente, seja permitindo testar a rotina de montagem!

### ONDE EXATAMENTE AS ROTINAS ASM DEVEM SER ARMAZENADAS?

Como dito no conjunto de caracteres AGD tutorial, o conjunto de chr-set começa com o caractere de espaço no endereço 31232 e dura 768 bytes (8 bytes multiplicados por 96 caracteres). É uma área de memória utilizável que não entrará em conflito com o kernel AGD. Na maioria dos casos, letras minúsculas e símbolos não são usados. Então, se estivermos ok em usar apenas letras maiúsculas e números em nosso projeto, só temos que escolher uma área de memória para armazenar a montagem (obviamente considerando o comprimento da rotina).

Os caracteres vermelhos neste gráfico representam zonas de memória livres ...

	!	"	#	\$	%	&	'	(	)	*	+	,	-	.	/	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	↑	_
£	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z	{		}	~	©

... e aqui ao lado estão as referências de memória (como na versão 4.7 do AGD): →→→

### UM EXEMPLO PRÁTICO

É importante permanecer no menu AGD chr-set ao importar dados binários. Por exemplo, vamos importar um arquivo binário sound-fx feito com a ferramenta SoundFX da DkTronic no endereço 31440. Vamos entrar no menu de definição de chr-set do AGD e, em seguida, importar o arquivo binário de 50 bytes. Então, saindo e entrando novamente no menu chr-set, vamos notar as seguintes mudanças:

120 BYTES AREA	
CHR	ADDRESS
!	31240
"	31248
#	31256
\$	31264
%	31272
&	31280
'	31288
(	31296
)	31304
*	31312
+	31320
,	31328
-	31336
.	31344
/	31352

56 BYTES AREA	
CHR	ADDRESS
:	31440
;	31448
<	31456
=	31464
>	31472
?	31480
@	31488

296 BYTES AREA	
CHR	ADDRESS
[	31704
\	31712
]	31720
↑	31728
_	31736
€	31744
a	31752
b	31760
c	31768
d	31776
e	31784
f	31792
g	31800
h	31808
i	31816
j	31824
k	31832
l	31840
m	31848
n	31856
o	31864
p	31872
q	31880
r	31888
s	31896
t	31904
u	31912
v	31920
w	31928
x	31936
y	31944
z	31952
{	31960
	31968
}	31976
~	31984
©	31992



Os sinais estranhos representam os dados binários importados, então nunca mude os pixels dentro desses caracteres!

Vamos escrever as instruções ASM para chamar o endereço 31440 em algum evento

```
...
ASM 205
ASM 208
ASM 122
...
```

E teste o jogo. Sem travar e o efeito sonoro funciona!

O conceito é aplicável à área de memória superior (até 64767). Usando endereços de memória superior da mesma maneira, o AGD não travará; entretanto, salvando um projeto, o chr-set será incluído, enquanto as rotinas armazenadas na memória alta devem ser salvas separadamente, fora do AGD.



## PEEK E POKE EM AGD - 6 Abr

Tutorial de Luca Bordonni, obrigado a David Saphier e Allan Turvey

Este tutorial requer um pouco de noções básicas de montagem, é pelo menos importante saber o significado dos comandos PEEK e POKE no Basic e como as chamadas ASM funcionam no AGD. É sugerido ler e entender os tutoriais sobre as chamadas ASM e Armazenar ASM no conjunto de caracteres AGD antes de continuar aqui.

### LEIA / ESCREVA DADOS DA MEMÓRIA USANDO AGD

Cada vez que um jogo AGD começa, os valores armazenados em todas as variáveis, incluindo VIDAS e PONTUAÇÃO, são zerados. Existem apenas duas circunstâncias a serem consideradas:

- o jogo termina depois de perder todas as vidas (evento de jogo perdido);
- o jogo termina com vitória (evento de jogo concluído).

No primeiro caso, o reset está ok. Em vez disso, em caso de vitória, pode haver a necessidade de recuperar algumas variáveis se o jogo tiver que continuar (por exemplo, vidas residuais e pontuação). Como o AGD não considera essa recuperação, é possível “pendurar” algumas variáveis em locais de memória e fazer a mágica.

### ALGUNS CONCEITOS ESSENCIAIS

#### ASM 62

Lê um único byte (0 a 255) declarado pelo próximo ASM, armazenando o valor no acumulador. Equivalente ao assembly: LD a,N

#### ASM 58

Significa “PEEK” (ler) o byte armazenado em um endereço de memória declarado pelo próximo par de comandos ASM (porque um endereço de memória é maior que 255), armazenando o byte no acumulador.

Equivalente de montagem: LD a,(NN)

#### ASM 50

Significa “POKE” (escrever) no endereço de memória declarado pelo próximo par de comandos ASM, um byte previamente armazenado no acumulador (por exemplo, através de um comando ASM 62 ou ASM 58).

Equivalente de montagem: LD (NN),a

### REFERÊNCIAS DE VARIÁVEIS GLOBAIS EM AGD 4.7

**PONTUAÇÃO** - é uma sequência de seis caracteres ascii, não aceita um formato de número real. Ele armazena seis bytes de 34495 a 34500; cada byte representa o valor ascii de um número (por exemplo, começa com valores 48,48,48,48,48,48 porque 48 é o código ascii do caractere “0”)

TELA	- armazena um byte em <b>32027</b>	CLOCK	- armazena um byte em <b>32051</b>
VIDAS	- armazena um byte em <b>32028</b>	RND	- armazena um byte em <b>32052</b>
LINE	- armazena um byte em <b>32049</b>	OBJ	- armazena um byte em <b>32053</b>

COLUMN - armazena um byte em 32050	OPT - armazena um byte em 32054
A - armazena um byte em 32029	K - armazena um byte em 32039
B - armazena um byte em 32030	L - armazena um byte em 32040
C - armazena um byte em 32031	M - armazena um byte em 32041
D - armazena um byte em 32032	N - armazena um byte em 32042
E - armazena um byte em 32033	O - armazena um byte em 32043
F - armazena um byte em 32034	P - armazena um byte em 32044
G - armazena um byte em 32035	Q - armazena um byte em 32045
H - armazena um byte em 32036	R - armazena um byte em 32046
I - armazena um byte em 32037	S - armazena um byte em 32047
J - armazena um byte em 32038	T - armazena um byte em 32048

Portanto, a instrução “LET LIVES=3” é equivalente a esta sequência de comandos ASM:

carregue o seguinte byte no acumulador

ASM 62

O byte é 3

ASM 3

POKE para o seguinte endereço

ASM 50

O endereço é 32028 (endereço reservado AGD para VIDAS)

ASM 28

ASM 125

#### RECUPERAR VIDAS RESTANTES APÓS UMA VITÓRIA

A variável LIVES é geralmente gerenciada na inicialização do jogo e nos eventos de eliminação do jogador. O valor residual deve ser tratado apenas em caso de vitória, portanto os eventos envolvidos serão a inicialização do jogo e o jogo concluído.

No evento de jogo concluído, leremos o byte armazenado no endereço de memória LIVES reservado. Em seguida, o valor será copiado em um endereço de memória livre personalizado. Apenas para fins de teste, vamos escolher o último endereço utilizável 64767. Agora lembre-se de que:

- quando todas as vidas forem perdidas, o byte armazenado no endereço 64767 será igual a zero;
- enquanto o jogo continuar, o valor de VIDAS (e o byte armazenado na memória alta) será maior que zero.

Em evento de jogo concluído, ler e armazenar o valor de vidas

PEEK 32028 (ler variável de VIDAS)

ASM 58

ASM 28

ASM 125

POKE 64767 (armazenar o byte anterior na memória alta)

ASM 50

ASM 255

ASM 252

A próxima etapa, no evento de inicialização do jogo, precisamos detectar se a jogada vem de uma vitória anterior, mantendo o valor de VIDAS recuperado; caso contrário, as vidas têm que ser reiniciadas.

No evento de inicialização do jogo, PEEK 64767

ou seja, ler vidas residuais previamente armazenadas na memória alta; se for uma nova jogada, o resultado será 0.

### ASM 58

ASM 255

ASM 252

POKE 32028

escreva o byte recuperado anteriormente no endereço de memória reservada de LIVES

### ASM 50

ASM 28

ASM 125

```
IF LIVES > 0
```

```
  nenhuma ação, apenas mantenha o valor recuperado
```

```
ELSE
```

```
LIVES é igual a zero quando um novo jogo começa (primeiro jogo ou perdeu todas as vidas),  
nesse caso carregue o valor inicial
```

```
LET LIVES = 3
```

```
ENDIF
```

### RECUPERAR A PONTUAÇÃO APÓS UMA VITÓRIA

Como dito, a variável **SCORE** armazena uma sequência de seis caracteres ascii, sem lidar com um formato de número real. Os seis bytes são armazenados de 34495 a 34500 e cada um representa o valor ascii para um número de caractere.

Cada vez que um novo jogo começa a **SCORE** é resetada a zero por AGD desta forma:

34495,48

34496,48

34497,48

34498,48

34499,48

34500,48

O número **48** é a correspondência ascii para o caractere "0" (encontre todos os valores ascii no Manual de Programação BASIC, Apêndice A).

Se precisarmos continuar uma jogada após a vitória, o placar deve pular o reset por AGD; vamos explicar como.

É possível recuperar cada byte dos endereços acima usando o mesmo método explicado para a variável **LIVES** ou usando uma rotina de montagem que faz a mágica. O seguinte código assembly foi escrito pelo brilhante Allan Turvey. Poucas linhas de código de montagem para cumprir uma tarefa dupla:

- armazenar os seis bytes na memória alta;
- restaure-os novamente nos endereços de **SCORE**, pulando a redefinição padrão no início do jogo.

A rotina pode ser armazenada diretamente no Conjunto de caracteres AGD. Esta é a montagem, incluindo os comentários que explicam as etapas:

```
; AGD SCORE RECALL (21 bytes)
```

```
; *****
```

```
; esta rotina começa em 31440 (end. para o símbolo " : " no conjunto de caracteres AGD)
```

```
; AGD armazena seis chrs ascii em 34495-> 34500, os endereços reservados para SCORE
```

```
; a rotina move os seis bytes para 64761-> 64766
```

```
; então os dados podem ser lidos a partir do endereço 64761
```

```
;
```

```
; USR 31440 (ASM 205,208,122) armazena a pontuação em memória alta 64761-> 64766 (uso em  
evento de jogo concluído)
```

```
; USR 31449 (ASM 205,217,122) recupera a pontuação da memória alta em 34495-> 34500 (uso  
no evento de inicialização do jogo)
```

```
;
```

```

score que 34495
storescore equ 64761
org 31440
putscore: ld hl,score
ld de,storescore
jp movescore
getscore: ld hl,storescore
ld de,score
movescore: ld bc,6
ldir
ret

```

## SCORE RESET

AGD não permite zerar a pontuação por meio de uma instrução específica. Tendo implementado a rotina especial de “recuperação de pontuação” acima, também é necessário redefinir os valores se o jogo terminar.

Aqui está uma rotina simples de Allan Turvey para salvar no chr-set e chamar do evento de inicialização do jogo:

```

; AGD SCORE RESET (11 bytes)
; *****
; esta rotina começa em 31464 (o símbolo “=” no conjunto de caracteres AGD)
; as variáveis lojas SCORE seis CHRS ascii em 34495
; a rotina zera os valores ascii (valor 48)
; uso no evento de inicialização do jogo, USR 31464 (ASM 205.232.122)
; após o restabelecimento de VIDAS, para verificar se a nova jogada vem de uma vitória
;
org 31464
ld hl,34495
ld b,6
dscore: ld (hl),48
inc l
djnz dscore
ret

```

Então, aqui está como o jogo concluído e os eventos de inicialização do jogo devem finalmente parecer:

evento de jogo concluído  
armazenar vidas e pontuação em alta memória

PEEK 32028 (ler variável de VIDAS)

```

ASM 58
ASM 28
ASM 125

```

POKE 64767 (armazenar o byte anterior na memória alta)

```

ASM 50
ASM 255
ASM 252

```

USR 31440 : armazenar seis bytes SCORE na memória alta

```

ASM 205
ASM 208
ASM 122

```

USR 32000 : forçar reinício (não sair para o menu)

```

ASM 205
ASM 0
ASM 125

```

evento de inicialização do jogo  
recupera vidas e pontua se o jogo vier de uma vitória  
caso contrário, redefine os valores iniciais

PEEK 64767 : leia VIDAS disponíveis

ASM 58

ASM 255

ASM 252

POKE 32028 : armazene o resultado na variável AGD-reservada LIVES

ASM 50

ASM 28

ASM 125

se vier de uma vitória, a variável LIVES armazena um byte maior que 0

SE VIDAS > 0

USR 31449 : Rotina de chamada de pontuação se o jogo for reiniciado de uma vitória

ASM 205

ASM 217

ASM 122

ELSE

Se não foi um começo de uma vitória (primeiro jogo ou perdeu todas as vidas), reinicie.

LET LIVES 3

USR 31464 : Rotina de redefinição de SCORE

ASM 205

ASM 232

ASM 122

ENDIF

CLS

mostra pontuação recuperada ou redefinida

LET LINE ...

LET COLUMN ...

COLOR ...

SHOWSCORE

mostra vidas recuperadas anteriormente ou redefinidas

LET LINE ...

LET COLUMN ...

COLOR ...

DISPLAY LIVES

...

Se estiver tudo bem para usar o conjunto de caracteres AGD para armazenar as rotinas de pontuação, baixe o arquivo binário diretamente daqui e, em seguida, apenas injete o arquivo bin no conjunto de caracteres AGD.

Saia do menu charset e entre novamente, as alterações ficarão assim:



Agora tudo o que temos a fazer é chamar as rotinas dos eventos, conforme explicado.



## COMPILADOR BORIEL - 23 Set

Tutorial de Luca Bordoni

Este tutorial é voltado para entusiastas do ZX Spectrum que experimentaram pelo menos um projeto de programa BASIC. Sugere-se estudar primeiro o Manual Oficial de Programação BASIC. Conhecimento de montagem não é necessário.

### BENEFÍCIOS DO COMPILADOR

Nos tempos modernos, muitos programadores da velha escola enfrentam emuladores e ferramentas de PC em vez das máquinas clássicas originais, por uma vantagem óbvia de economia de tempo.

Seguindo este princípio, o Boriel ZX Basic (não deve ser confundido com o Sinclair Basic padrão) é uma ferramenta de PC amplamente apreciada para alcançar uma conversão rápida de um programa ZX Basic para um bloco de código de máquina funcionando perfeitamente.

Além disso, o Boriel ZX Basic tem outra grande funcionalidade: uma nova sintaxe embutida do Basic, que resulta no jeito muito fácil de aprender Basic com experiência média: uma lição de alguns minutos aqui [neste](#) artigo.

É possível escrever nosso próprio código usando um editor de texto e converter o arquivo de texto diretamente por meio de uma única linha de comando no prompt do nosso sistema operacional. Falaremos sobre essa etapa no capítulo final.

### A SINTAXE ZX BASIC

Como dito acima, o ZX Basic não deve ser confundido com o Sinclair Basic padrão. Além de ser um compilador (que funciona a partir do prompt de linha de comando do PC), o ZX Basic pretende ser um dialeto básico moderno, tentando manter muitos dos recursos originais do Sinclair Basic.

O ZX Basic tenta manter a compatibilidade o máximo possível com o Sinclair Basic, adicionando muitos novos recursos interessantes. O ponto é: a sintaxe Sinclair Basic padrão não pode ser convertida pelo compilador ZX Basic.

Vejamos algumas das diferenças mais relevantes entre as duas sintaxes:

### DECLARAÇÃO DE VARIÁVEIS

A primeira diferença relevante é a gestão das variáveis. Nosso hábito era atribuir um valor a uma variável (valor numérico ou string) e, em seguida, manipulá-lo ao longo do código. No ZX Basic, qualquer variável usada deve ser declarada no início do programa por meio da instrução DIM , enquanto o LET clássico é omitido:

```
10 REM Standard Sinclair Basic
20 GO SUB 60
30 PRINT "Press any key"
40 IF INKEY$="" THEN GO TO 40
50 PRINT n$: STOP
60 LET n$="Luke": RETURN
```

```

REM Boriel's ZX Basic
DIM n AS STRING

GOSUB MyName
PRINT "Press any key"
10 IF INKEY="" THEN GOTO 10
END IF
PRINT n: STOP
MyName:
n="Luke"
RETURN

```

Nesta comparação simples, os programadores nostálgicos do Basic podem notar imediatamente algumas coisas interessantes: no exemplo do ZX Basic, o **AS UBYTE** e o **AS STRING** são instruções novas e a numeração está faltando !

**UBYTE** e **STRING** representam dois tipos de variáveis diferentes, claramente um para um valor numérico e outro para uma string.

Declarar os tipos de variáveis é obrigatório no ZX Basic.

Os tipos de variáveis mais usados são:

```

UBYTE – variável de 8 bits, para valores de 0 a 255 ;
UINTEGER – variável de 16 bits, para valores de -32768 a 32767;
STRING – variável de 16 bits, usada para strings (o símbolo $ pode ser omitido).

```

Sobre a numeração, o ZX Basic trata os números das linhas como rótulos: os números das linhas do Basic clássico podem ser escritos sem problemas, mas serão ignorados pelo ZX Basic, a menos que sejam usados como texto explicativo, por exemplo, para as instruções **GOTO** ou **GOSUB**.

#### NÚMEROS DE LINHA E ETIQUETAS

Este é um novo recurso muito interessante. Tratar um número de linha como um rótulo (que também pode ser textual) pode transformar cada bloco de código Basic em uma sub-rotina separada; o código ficará muito limpo e cada sub-rotina será imediatamente rastreável, assim como acontece nas linguagens de programação modernas:

```

10 REM Standard Sinclair Basic
20 GO SUB 60
30 PRINT "Press any key"
40 IF INKEY$="" THEN GO TO 40
50 PRINT n$: STOP
60 LET n$="Luke": RETURN

```

```

REM Boriel's ZX Basic
DIM n AS STRING

GOSUB MyName
PRINT "Press any key"
10 IF INKEY="" THEN GOTO 10
END IF
PRINT n: STOP
MyName:
n="Luke"
RETURN

```

Mesmo que o código Basic clássico resulte mais curto, a legibilidade do ZX Basic é incomparável. Adicionando a vantagem de usar um editor de texto simples, agora podemos escrever um programa BASIC como um documento de texto normal!

Vamos observar mais detalhes:

- a instrução **END IF**: fechar uma condição IF é obrigatório no ZX Basic;
- a falta do símbolo **\$** no comando **INKEY**: de acordo com as palavras reservadas do ZX Basic, tanto **INKEY** quanto **INKEY\$** são aceitos.

## CARACTERES GRÁFICOS

O ZX Spectrum possui dois tipos de caracteres gráficos; gráficos de bloco (ou seja, os símbolos gráficos que aparecem sobre o teclado de borracha clássica, da tecla 1 a 8) e gráficos definidos pelo usuário (UDG, as 21 letras editáveis de A a U). O método para inseri-los no ZX Basic é o mesmo encontrado no formato de arquivo .bas criado por Paul Dunn para seu BASin.

– UDG pode ser inserido no código com um escape antes da letra que corresponde ao UDG. Pela primeira UDG, por exemplo, o uso \ A.

– Os caracteres gráficos do bloco podem ser inseridos no código com uma sequência de escape semelhante. O \ seguido por símbolos específicos é usado para representar blocos gráficos ou outros caracteres especiais.

Aqui está uma tabela das combinações mais usadas:

*Irei representar aqui o ESPAÇO com o símbolo “\_”*

Bloco Sintaxe	Descrição	Valor CHR\$	Graphics
\_	espaço, espaço	CHR\$ (128)	
\'	espaço, apóstrofo	CHR\$ (129)	
\'_	apóstrofo, espaço	CHR\$ (130)	
\''	apóstrofo, apóstrofo	CHR\$ (131)	
\.	espaço ponto	CHR\$ (132)	
\:	espaço, dois pontos	CHR\$ (133)	
\'.	apóstrofo, ponto	CHR\$ (134)	
\':	apóstrofo, dois pontos	CHR\$ (135)	
\._	ponto, espaço	CHR\$ (136)	
\.'	ponto apóstrofo	CHR\$ (137)	
\:.	dois pontos, espaço	CHR\$ (138)	
\:'	dois pontos, apóstrofo	CHR\$ (139)	
\..	ponto ponto	CHR\$ (140)	
\.:	ponto, dois pontos	CHR\$ (141)	
\:.	dois pontos, ponto	CHR\$ (142)	
\; ;	Pto Virgula, Pto Virgula	CHR\$ (143)	
\*	asterisco	CHR\$ (127)	

## CÓDIGOS DE ATRIBUTOS DE CORES

Os programadores do ZX Spectrum sabem que é possível incorporar atributos de cor diretamente nas strings do programa, em vez de usar os comandos PAPER, INK, BRIGHT e FLASH. Esta prática não é obrigatória, mas pode reduzir fortemente o comprimento do código. Como os códigos gráficos, os códigos de atributos ZX BASIC seguem o mesmo esquema como Paul Dunn's BASin.

As sequências de escape para os caracteres de controle são as seguintes:

\{iN} INK color N, onde N está no intervalo de 0 a 8;

\{pN} PAPER N, onde N está no intervalo de 0 a 8;

\{bN} BRIGHT N, onde N é 0 ou 1;

\{fN} FLASH N, onde N é 0 ou 1.

Assim, por exemplo, um código de controle embutido para uma tinta vermelha brilhante

seria `\{b1\}\{i2\}`.

Nota: o valor de cor 8 é usado para significar “transparente” (por exemplo, não altere o valor da tinta / papel no quadrado que está sendo impresso).

## A LINHA DE COMANDO DO COMPILADOR

O Compilador ZXB é a função principal que transformará um programa ZX Basic em um bloco de código de máquina. Basta salvar o arquivo de texto que contém o código ZX Basic e renomear a extensão para `.bas` a fim de torná-la legível para o compilador.

O arquivo `.bas` deve ser salvo no mesmo diretório ZX Basic onde se encontra o arquivo `zxb.exe`.

Vamos abrir o prompt do nosso sistema operacional e mover para o diretório onde os arquivos `zxb.exe` e `.bas` estão localizados.

Para quem não está familiarizado com os comandos DOS...

- D: digite o disco rígido local D;
- dir diretórios de lista e arquivos a partir da posição atual do usuário;
- cd.. volta para a árvore de diretórios;
- cd ZXB entra no diretório (ou subdiretório) chamado ZXB;
- cd/ retornar ao diretório raiz principal.

Aqui estão alguns exemplos de linha de comando ZXB para compilar nosso arquivo `.bas`:

Gera um arquivo `.bin` no endereço de memória 30000:  
`zxb.exe myprogram.bas -org=30000`

Gera um arquivo `.tap` no endereço de memória 32000:  
`zxb.exe myprogram.bas -tap -org=32000`

Gera um arquivo `.tzx` no endereço de memória 50000:  
`zxb.exe myprogram.bas -tzx -org=50000`

A mágica acabou! ...

Considere que nas primeiras vezes é fácil encontrar erros do compilador - eles sempre são devidos a uma sintaxe errada em nosso código BASIC.

Nesse caso, temos que verificar o código do Basic em detalhes ☹. Este é o único “lado negro” desta ferramenta, mesmo que a mensagem de erro do compilador deva ajudar a entender o problema específico de sintaxe do Basic.

Update – ZXB tem um parâmetro importante chamado **heap** que reserva uma área de memória para lidar com variáveis de string.

Encontre mais informações úteis em um post meu no Fórum Oficial ZXB.

<https://www.boriel.com/forum/showthread.php?tid=845>

# PGD - Platform Game Designer



O Platform Game Designer é separado em dois programas diferentes, o designer linear e o designer de jogos explorador. O primeiro cria jogos de plataforma lineares onde o jogador deve coletar todos os itens em cada tela antes de avançar para o próximo, o último cria jogos baseados em exploração onde o jogador pode se mover de sala em sala à vontade. Embora ligeiramente diferentes, a maioria dos recursos é comum a ambos os utilitários e essas instruções foram escritas como um guia geral para o uso de ambos. Onde as diferenças ocorrem, elas são documentadas.

## Menu principal

Permite selecionar uma função do jogo que você deseja projetar. A quantidade de memória restante disponível para seu aplicativo é exibida no canto superior direito da tela.

## Sprite Design

Mova o cursor ao redor do sprite com as teclas do cursor. Use **SPACE** ou **0** para definir / desmarcar um pixel. **ENTER** retorna ao menu principal.

Os Sprites **00** e **01** são usados como sprites principais dos jogadores e possuem 8 quadros de animação cada. Sprite **00** é exibido quando o jogador se move para a esquerda e sprite **01** quando o jogador se move para a direita. Todos os outros sprites têm quatro quadros de animação e podem ser usados para os monstros de patrulha.

F = Próximo quadro  
L = Último Sprite  
N = Próximo Sprite  
P = Cor do papel  
I = Cor da tinta  
B = Alternar brilho  
M = Copiar quadro atual para a área de transferência  
H = Flip sprite horizontalmente  
V = Flip sprite verticalmente  
K = Colar o quadro da área de transferência  
C = Limpar quadro atual  
X = Cria um novo sprite inimigo  
D = Excluir sprite atual (não é possível excluir os sprites **00** e **01**)  
**ENTER** = Retornar ao menu principal

## Design de bloco de caracteres

Blocos são usados para construir as telas. Cada bloco tem 8x8 pixels de tamanho e tem um

tipo diferente atribuído a ele. Existem vários tipos diferentes, cada um com seu próprio conjunto diferente de atributos. Esses são:

**ESPAÇO VAZIO** - o jogador pode se mover livremente através de blocos de espaço livre.

**PLATAFORMA NORMAL** - o jogador pode mover-se livremente além das plataformas da esquerda ou da direita e saltar sobre elas de qualquer direção.

**TRANSPORTADOR ESQUERDO** - um bloco sólido que impulsiona o jogador para a esquerda, como se estivesse em uma esteira rolante.

**TRANSPORTADOR DIREITO** - como acima, mas impulsiona para direita em vez de esquerda.

**PAREDE SÓLIDA** - um bloco que é intransponível de qualquer direção.

**MORTAL** - tocar em um bloqueio mortal resulta na perda da vida de um jogador.

**ITEM COLECIONÁVEL** - um item que precisa ser coletado pelo jogador.

**PLATAFORMA ON / OFF** - uma plataforma que aparece e desaparece a cada poucos segundos.

O designer de jogos linear possui os seguintes blocos adicionais que não estão disponíveis no designer do jogo explorador:

**PLATAFORMA CRUMBLY** - como uma plataforma normal, mas como um jogador se sustenta nisso, desmorona e eventualmente se torna espaço vazio.

**EXIT** - espaço vazio que pisca quando todos os itens são coletados. Mover-se para uma saída intermitente prossegue para a próxima tela.

**MAGIC DOOR** - um bloco intransponível que desaparece quando uma chave mágica é coletada.

**MAGIC KEY** - um item que, quando coletado, transforma todas as portas mágicas em espaços vazios.

Mova o cursor pelo personagem com as teclas do cursor. Use SPACE ou 0 para definir / desmarcar um pixel. ENTER retorna ao menu principal.

Q = Mover para a esquerda através da lista de propriedades do bloco

W = Mover para a direita através da lista de propriedades do bloco

L = Último bloco

N = Próximo bloco

P = Cor do papel

I = Cor da tinta

B = Alternar brilho

M = Copiar o bloco atual para a área de transferência

K = Colar o bloco da área de transferência

C = Limpar o bloco atual

X = Cria um novo bloco de caracteres

D = Excluir o bloco de caracteres atual

ENTER = Retornar ao menu principal

## Layout da tela

Mova o cursor pela tela com as teclas do cursor. Use 1 e 2 para selecionar o bloco de caracteres que você deseja colocar na posição atual do cursor. ESPAÇO ou 0 coloca o bloco desejado na tela atual nessa posição. ENTER retorna ao menu principal.

1 = Mover para a esquerda através da tabela de blocos

2 = Mover para a direita através da tabela de blocos

N = próxima tela

P = tela anterior

X = criar uma nova tela

D = Excluir tela atual

T = Alterar título da tela

B = Definir cor da barra de título (para todas as telas)

ENTER = Retornar ao menu principal

## Patrulhando Nasties

Isso é complicado e exige um pouco de prática, mas é fácil quando você sabe como usá-lo. Isso ajuda se você já projetou o layout da tela e pelo menos uma entidade alienígena antes de selecionar essa função.

O caminho seguido por cada alienígena em qualquer tela particular é mostrado como barras vermelhas. Cada alienígena pode patrulhar para cima e para baixo, ou para a esquerda e para a direita, mas não de qualquer outra forma. Usando a tecla Q para alternar entre os alienígenas em qualquer tela. Uma vez que um alienígena é selecionado, use as teclas do cursor para mover a barra ao redor. ESPAÇO ou 0 permitirá que você mova a outra extremidade da barra.

N = Próxima tela

P = Tela anterior

Q = Mover o próximo alienígena de patrulha

S = Alterar velocidade do alien selecionado

T = Alterar o tipo de sprite usado para o alien selecionado

D = Excluir alien selecionado

X = Criar novo alien na tela atual (até 4 por tela)

ENTER = Retornar ao menu principal

## Posições Iniciais do Jogador

Esta opção é exclusiva para o designer de jogos lineares. Use isso para determinar onde o player iniciará em cada tela. Mova a posição inicial ao redor da tela com as teclas do cursor.

N = Próxima tela

P = Tela anterior

ENTER = Retornar ao menu principal

## Layout do Mapa

Disponível apenas no designer de jogos do explorer.

O "mapa" do seu jogo é organizado como uma grade de 10x8 locais, e todos começam vazios. Locais vazios aparecem como dois hifens "-" os quartos aparecem como os números da tela, por exemplo, "01" ou "12". Como cada quarto é projetado no designer de tela, ele pode ser colocado nessa grade em um local escolhido. Durante o jogo, não será possível ao jogador mover-se para um espaço de grade vazio, nem o jogo iniciará se o local da grade inicial estiver vazio ou não tiver sido selecionado.

O cursor vermelho pode ser movido ao redor da grade usando as teclas de cursor, para mudar a sala em um determinado local da grade mova as teclas '1' e '2'. Os quartos são exibidos nos dois terços inferiores da tela conforme são selecionados para facilitar o processo.

Seu mapa seguirá as regras da geometria euclidiana, mas você pode dobrar as regras para criar um campo de jogo distorcido, caso deseje. Mover-se para a esquerda a partir de uma sala situada na borda esquerda do mapa fará com que o jogador reapareça na sala colocada na borda direita da próxima linha, se alguém estiver lá. Da mesma forma, mover-se diretamente de uma sala à direita levará o jogador para a sala na extremidade esquerda da sala, uma linha abaixo. Se você não quiser que isso aconteça, você deve construir paredes nas telas relevantes para formar uma barreira física. Também é possível reutilizar uma sala, para que ela apareça mais de uma vez em seu mapa.

Pressione 'X' para declarar a localização da grade como o ponto no qual o jogador deve começar o jogo. Você será então obrigado a manipular um cursor vermelho para determinar as coordenadas iniciais do jogador. Pressione ENTER para retornar ao modo de layout de grade quando isso for feito.

1 = Selecione quarto anterior da lista  
2 = Selecione a próxima sala da lista  
X = Selecione o lugar onde o jogador começa o jogo  
ENTER = Retornar ao menu principal

## Controles do teclado

Defina os controles que o jogador irá usar no jogo. Embora seja perfeitamente válido usar a tecla **ENTER** no jogo, ao usar a função 'Testar jogo' no menu principal, a tecla **ENTER** retornará à tela do menu principal, de modo que a tecla **ENTER** não funcionará como você esperava. Quando o seu jogo é salvo, esta função será desativada para que a tecla **ENTER** funcione corretamente.

## Diversos

Exibe vários prompts que fazem perguntas básicas sobre o jogo.

Ao escolher esta opção, você será inicialmente solicitado "Queda sem morrer y/n?". Pressione **Y** se quiser permitir que o jogador caia em qualquer altura sem morrer, ou **N** se quiser uma queda de grande altura para resultar na perda de uma vida. Depois disso, você será solicitado a decidir quantas vidas o jogador iniciará o jogo, portanto, insira um número de 1 a 9.

No designer de jogos linear, você será perguntado em seguida "Barra de tempo em y/n?". Pressione **Y** se você quiser impor um limite de tempo para completar cada tela, ou **N** se o jogador tiver permissão para tanto tempo quanto quiser. A próxima pergunta em ambos os utilitários é "Alterar direção no ar?". Pressione **Y** se quiser que o jogador mude de direção no ar (como em Egghead In Space), **N** se não (mais como Manic Miner). Se você responder "sim" a este aviso, terá a opção de fazer o personagem saltar o tempo todo, se estiver escrevendo um jogo em que o personagem principal é uma bola, canguru, homem em um pula-pula ou qualquer outra coisa.

Feito isso, você pode alterar o prompt que aparece no final de cada jogo. Observe que esse prompt aparece apenas quando você executa o jogo independentemente do programa do designer de jogos e somente se iniciar o jogo com `RANDOMIZE USR 32768`. Se você estiver escrevendo sua própria página de título e iniciando o jogo com `RANDOMIZE USR 32771`, esse prompt não irá aparecer.

## Música

Isso permite que você adicione um pouco de música de 48k bip ao seu jogo. Se você preferir incorporar 128k de música, simplesmente deixe essa tela vazia e use um programa de música decente, como o Soundtracker.

A música é exibida como uma lista de números, como se fossem valores em uma série de instruções BEEP. Todas as notas na lista têm a mesma duração, os valores listados referem-se ao tom e estão todos no intervalo de 00 a 24. Uma nota com um tom de 00 produzirá um tom no meio C, enquanto 24 produzirá um tom dois oitavas mais altas. Dois hífen denotam um descanso, isto é, um período em que nenhuma nota é tocada. A música é repetida ad infinitum dentro do jogo, mas você pode fazer o tempo que quiser, sujeito à memória disponível. Um exemplo de música, os primeiros compassos do tema "Capitão Pugwash", vai assim:

```
12 12 12 12 -- -- 12 12 12 12
-- 19 16 -- 12 16 -- 19 24 --
19 16 -- 12 07 07 07 07 -- --
07 07 07 07 -- 14 11 -- 07 11
-- 14 19 -- 14 11 -- 07 --
```

Como em tudo, a experimentação é a melhor maneira de aprender. Os controles do teclado

para esta seção são os seguintes:

Mova o cursor vermelho ao redor da música usando as teclas do cursor ou 5678.

1 = Diminui o tom da nota atual

2 = Aumentar o tom da nota atual

D = Apagar nota na posição do cursor

X = Insira uma nova nota ou descanse na posição atual do cursor

ENTER = Retornar ao menu principal

Para transformar uma nota em um descanso, simplesmente pressione **espaço** ou mantenha o dedo pressionado na tecla 1. Para transformar um descanso em uma nota, pressione 2.

## Painel de status

Isso permite que você altere o layout do painel de status e é particularmente útil se você pretende configurar seu próprio painel de status gráfico no terço inferior da tela antes de seu RANDOMIZE USR ou CALL para iniciar o jogo. Por exemplo, você pode querer desenhar um painel futurista e exibir a partitura e viver em torno de seu design, ou talvez ter um efeito de rolagem medieval - um compressor de tela permitirá que você faça isso, mas certifique-se de que sua rotina está localizada abaixo do endereço de memória. Além disso, se o painel de status não tiver espaço suficiente para uma descrição de 32 caracteres de cada sala, você poderá reduzir o tamanho da barra de título e movê-la para um local mais conveniente.

Os detalhes do painel de status são exibidos junto com um menu. Selecione o detalhe que você deseja manipular na lista, depois use os cursores ou 5678 para manipular o item de exibição. Quando estiver satisfeito, pressione **ENTER**. Além disso, ao manipular a barra de título, você pode usar as teclas 1 e 2 para encurtar e aumentar a barra, respectivamente.

Se você tiver o know-how, talvez queira incorporar uma fonte personalizada. Defina isto no endereço 32000 ou abaixo, e faça com que a variável do sistema CHARS aponte para ele, cutucando os endereços 23606 e 23607.

## Patrulhando Nasties

Onde seria um jogo de plataforma sem estes! Patrulhando nasties são os alienígenas, robôs e outros recursos estranhos que habitam os níveis, e patrulham esquerda e direita, ou para cima e para baixo.

Q = Selecionar próximo alienígena na tela atual

S = Alterar a velocidade do alienígena atual

T = Alterar o tipo de alienígena atual

D = Excluir alienígena atual

X = Criar novo alien

N = Próxima tela

P = Tela anterior

ENTER = Retornar ao menu principal

## Quadros

Esta opção permite que você selecione a taxa de quadros, ou velocidade, em que o jogo será executado. Selecione 2 para rodar a 25 quadros por segundo para um jogo de plataforma de ação rápida, 3 para 17 quadros por segundo, se o seu jogo for um assunto mais considerado. 4 é executado em 12,5 quadros por configuração, que foi a taxa de quadros usada no Manic Miner.

## Saltar

Isso permite que você altere a altura que o jogador pode pular no seu jogo. A configuração HI permite saltos de até 3 blocos de altura, a configuração LO permite saltos de até 2 blocos. Para um jogo de ação rápido, a configuração HI é provavelmente melhor, embora isso possa fazer a tela parecer um pouco menor. Para um jogo mais lento, a configuração LO é recomendada.

## Carregar jogo

Carrega um jogo salvo anteriormente da fita. Se você tentar carregar um arquivo que esteja corrompido ou não for um arquivo do Platform Game Designer, o programa não permitirá que você retorne ao menu principal e permanecerá no modo de carregamento até que um jogo válido seja carregado. O jogo de demonstração, Space Hopper, também pode ser carregado no designer linear e alterado, caso você deseje isso.

Não é possível carregar um jogo criado com o designer Linear no designer do Explorer ou vice-versa.

## Salvar jogo

Esta opção solicita um nome de arquivo de até 10 caracteres e salva o jogo em fita. Recomendações regulares são recomendadas. Isso pode ser carregado de volta no designer de jogos em um ponto posterior e editado mais adiante. Quando você tiver completado sua obra-prima e desejar produzir uma versão que carregue e seja executada automaticamente a partir da fita independentemente do programa do designer, será necessário escrever um programa de carregador BASIC nas seguintes linhas:

```
10 CLEAR 32767: LOAD CODE: RANDOMIZE USR 32768
```

Em seguida, salve este programa em fita com SAVE "programe" LINE 10 imediatamente antes de salvar uma cópia do seu jogo.

Se você deseja produzir seu próprio menu ou tela de título, certifique-se de que seu código não interfira nesta área de memória. Quando você deseja iniciar o jogo, use uma chamada para endereçar 32771, pois isso inicia o jogo sem o prompt "Pressione qualquer tecla para iniciar".

Embora os direitos autorais da Platform Game Designer sejam de propriedade exclusiva de seu autor, os direitos autorais são renunciados a todos os jogos produzidos usando o utilitário Either. Você é, portanto, livre para publicar ou distribuir suas criações como desejar, desde que você não inclua o programa do designer em si.

A função salvar exibe a área de memória ocupada pelo seu jogo. Além disso, outros 544 bytes são usados como um buffer temporário logo após o endereço final. Se você é experiente e sabe o que está fazendo, é possível usar esta área para suas próprias rotinas ou espaço de trabalho, mas tenha em mente que ele será sobrescrito quando o jogo começar.

## Jogo de teste

Isso permite que você teste seu trabalho e, como tal, inicia o jogo na tela que você estava editando pela última vez. Pressione **ENTER** a qualquer momento para retornar ao menu principal.

## Programadores Avançados

### Personalizando seus jogos

Ao iniciar o programa com USR 32771, o controle é passado para o programa de chamada, seja BASIC ou código de máquina, assim que o jogo termina. Não há razão para você não escrever sua própria tela de introdução, rotinas de game over etc.

Os jogos não definirão os atributos ou as cores das bordas, portanto, certifique-se de selecionar as cores desejadas no programa do carregador BASIC ou no código da tela do título. Além disso, a tela não é apagada antes de iniciar o jogo, portanto, veja você mesmo ou, melhor ainda, configure seu próprio painel de status no terço inferior da tela.

Os jogos lineares só saem quando todas as vidas dos jogadores forem perdidas, já que completar o nível final faz o jogador voltar ao início com uma vida extra. Os jogos do Explorer são únicos, pois eles serão encerrados se o jogador coletar todos os itens, então você pode querer ter dois jogos separados em telas. PEEK 33464 retornará 7 na conclusão do jogo e zero se o jogador perder todas as vidas.

### Memória usada

Jogos escritos com qualquer um dos utilitários sempre começam no endereço 32768, e o endereço final é exibido sempre que você salvar o jogo. Os 544 bytes a seguir não são usados como código de programa ou dados, mas como um buffer e, se a memória estiver restrita, você pode usar essa área como um espaço de trabalho. Na prática, é provável que seja de pouco valor, pois esta área é sobregravada pelo jogo durante o jogo. O resto da RAM do Spectrum é inteiramente seu.

PROGRAM © 2005 Jonathan Cauldwell  
RELEASE by CRONOSOFT  
[www.cronosoft.co.uk](http://www.cronosoft.co.uk)

# Classic Game Designer (C.G.D) instructions.

Dave Hughes, updated to version 1.3 on 15/10/13



O CGD é um utilitário que é executado no Spectrum 48k e acima para criar jogos com a sensação da "velha escola" do início dos anos 80.

Esperemos que a maioria das instruções esteja dentro do utilitário, a menos que as saídas "E" sejam informadas de outra forma. Desculpas antecipadas pelo uso liberal do termo "baddy".

## 1. Desenhando Sprites

Aqui você cria os objetos em movimento no jogo. UDGS 0,1,2,3 são os gráficos de jogador direito, esquerdo, para cima e para baixo respectivamente. UDGS 4-15 são os sprites 'baddy', eles têm apenas um UDG cada, então não há gráficos direcionais para eles.

Para editar as propriedades do player (velocidade, tipo de controle), use durante a edição do bloco 0.

As principais chaves são:

**QAOP-Space** - para criar seu gráfico.

Teclas 1 e 2 para percorrer seus sprites.

A tecla 4 rola pelo efeito desejado quando o sprite colide com o jogador. Então, por esse motivo você não pode selecioná-lo para UDGs 0-3 (o sprite do jogador).

Existem 4 tipos de efeitos:

1. Não faça nada - agora
2. Matar jogador - jogador perde a vida por colisão
3. Mata o guardião - o sprite malvado é desligado e o contador 2 é diminuído.
4. Contador Dec 1 - contador 1 é diminuído.

Tecla 5 - rola pela velocidade do sprite, 9 é o mais rápido, 1 é muito, muito lento.

Tecla 6 - percorre o mecanismo de controle. Para o jogador é o tipo de tecla lida, para os vilões é IA. É melhor alterar as condições do player no UDG 0.

A tecla "G" gera um UDG espelhado aleatório, pode ocasionalmente parecer um invasor espacial. Substitui o UDG atual.

As outras chaves são esperançosamente auto-explicativas, não mencionadas é a chave "R" que pode ser usada para girar o sprite.

## 2- Blocos de Design

Você tem 32 (0-31) blocos para fazer o seu jogo. QAOP-Spc para fazer gráficos, teclas 1 e 2 para percorrer os blocos.

O bloco 31 é ampliado e usado na tela de introdução do jogo. Também pode ser usado como um bloco de jogo.

As teclas 5 e 6 percorrem o efeito que o bloco terá no player. Teclas T & Y percorrem o efeito que o bloco tem no baddy.

Existem alguns efeitos de bloco:

- Não faça nada: bloco passivo que você pode percorrer sem efeito.
- Bloco sólido: o sprite não pode se mover através do bloco.
- Alterar tipos de baddy: altera o IA mal-intencionado na colisão com este tipo de bloco. Não afeta o jogador mesmo quando selecionado.
- Faça sprite para baixo. Afeta apenas baddys, faz um sprite unidirecional. Mesmo para cima / direita / esquerda.
- Faça Sprite subir.
- Faça sprite ir para direita.
- Faça sprite ir para esquerda.
- Faça sprite aleatório. Apenas Baddy, torna unidirecional em uma direção aleatória
- Dec contador 1: contador 1 é um contador de 16 bits (0-65535). A colisão com este bloco diminui em 1 e apaga o bloco com o bloco #1.
- Inc contador 1: contador 1 é um contador de 16 bits. A colisão com este bloco aumenta em 1 e apaga o bloco com o bloco #1.
- Contador Dec 2: contador 2 é um contador de 16 bits. A colisão com este bloco diminui em 1 e apaga o bloco com o bloco #2.
- Inc contador 2: contador 2 é um contador de 16 bits. A colisão com este bloco aumenta em 1 e elimina o bloco com o bloco #2.
- Contador Dec 3: contador 3 é um contador de 8 bits (0-255). A colisão com este bloco diminui em 1 e apaga o bloco com o bloco #3.
- Inc contador 3: contador 3 é um contador de 8 bits. A colisão com este bloco aumenta em 1 e apaga o bloco com o bloco #3.
- Zero conter 1: Zera o contador e apaga como bloco #1.
- Zero conter 2: Zera o contador e apaga como bloco #1.
- Zero conter 2: Zera o contador e apaga como bloco #1.
- Força para baixo: força o jogador a descer um quadrado, incapaz de retornar.
- Força up: força o jogador a subir um quadrado, incapaz de retornar.
- Força à esquerda: força o jogador a deixar um quadrado, incapaz de retornar à direita.
- Força à direita: força o jogador à direita um quadrado, incapaz de retornar à esquerda.
- Contador 1 = 0, bloqueado: se o contador 1 for zero, não pode ser movido.
- Contador 2 = 0, bloqueado: se o contador 1 for zero, não pode ser movido.
- Contador 3 = 0, bloqueado: se o contador 1 for zero, não pode ser movido.
- Counter1 <> 0, bloqueado: se o contador 1 não for zero, não poderá ser movido.
- Counter2 <> 0, bloqueado: se o contador 1 não for zero, não pode ser movido.
- Counter3 <> 0, bloqueado: se o contador 1 não for zero, não poderá ser movido.
- Counters 123nzlocked: os contadores 1, 2 e 3 devem ser zero para permitir a passagem.
- Setflag1 ON: define o flag 1 ON, apaga com o bloco 1.
- Setflag2 ON: define o flag 2 ON, apaga com o bloco 2.
- Setflag3 ON: define o flag 3 ON, apaga com o bloco 3.
- Setflag1 OFF: define o flag 1 OFF, apaga com o bloco 1.
- Setflag2 OFF: define o flag 2 OFF, apaga com o bloco 2.
- Setflag3 OFF: define o flag 3 OFF, apaga com o bloco 3.
- Definir todos os sinalizadores ON: define sinalizadores 1,2,3 ON, apaga com o bloco 0.
- Definir todos os sinalizadores OFF: define sinalizadores 1,2,3 OFF, apaga com o bloco 0
- Alternar todos flags: se um flag estiver ON, ele será desativado e vice-versa, para todos os 3 flags. Excluí com o bloco 0.
- Sinalizador 1 ON = bloqueado: se o sinalizador 1 estiver ON, este bloco não pode ser movido.
- Sinalizador 2 ON = bloqueado: se o sinalizador 2 estiver ON, este bloco não pode ser movido.
- Sinalizador 3 ON = bloqueado: se o sinalizador 3 estiver ON, este bloco não pode ser movido.

- Flag 1 OFF = locked: se o flag 1 estiver OFF este bloco não pode ser movido.
- Flag 2 OFF = locked: se o flag 2 estiver OFF este bloco não pode ser movido.
- Flag 3 OFF = locked: se o flag 3 estiver OFF este bloco não pode ser movido.
- Aberto se timer = 0: somente se o timer for zero este bloco pode ser movido.
- Aberto se temporizador <> 0: se o temporizador é zero este bloco não pode ser movido.
- Diminuir o temporizador: diminui o temporizador em 1, o bloco não é apagado
- Aumentar o temporizador: aumenta o temporizador em 1, o bloco não é apagado.
- Próximo nível acima: conclui o nível atual e pula para o próximo.
- Próximo nível abaixo: leva o jogador para baixo um nível.
- Vá para o nível 1: esses níveis também podem ser usados como, por exemplo, teletransportadores ou bordas de tela para lhe dar qualquer a forma do mapa que você deseja em um jogo com várias telas.
- Ir para o nível 2
- Ir para o nível 3
- Ir para o nível 4
- Ir para o nível 5
- Ir para o nível 6
- Ir para o nível 7
- Ir para o nível 8
- Ir para o nível 9
- Jogo completo: qualquer que seja o nível em que o jogador estiver, isso termina com sucesso todo o jogo.
- Matar sprite: desativa o sprite, isso não funciona bem se aplicado ao player, mas funciona melhor quando aplicado aos bandidos. O contador 2 é diminuído.
- Matar todos os bandidos: desliga todos os bandidos na tela atual. Eles estão ativos no retorno à tela.
- Kill jogador: jogador perde uma vida.
- Vida extra: aumenta a vida em um, faz o barulho do zap mas não interrompe o jogo.
- Gravidade ON: ajusta a gravidade, puxa o jogador para baixo lentamente, a velocidade pode ser alterada (veja POKES). Se a gravidade for misturada com blocos de "força", você poderá obter gráficos fantasmas.
- Gravidade OFF: desliga a gravidade, qualquer que seja a condição atual.
- Alternar gravidade: se a gravidade atual estiver desligada e vice-versa.
- Tablet Pacman: dá imunidade ao jogador, define os maus para fugir do modo, efeito temporário, flash de fronteira e ruído de carrapato.
- Imunidade: dá imunidade temporária ao jogador, flash da borda e ruído.
- Bloco lento: o movimento do sprite é retardado ao passar.
- Próximo bloco abaixo: quando o sprite passa, ele muda para o próximo número de bloco.
- Próximo bloco para cima: quando o sprite passa, ele muda para o próximo número de bloco
- Push block: pode ser empurrado apenas pelo bloco 0. Empurre blocos automaticamente quebra de tela, se você não quer isso, você precisará criar sua própria borda de bloco sólido.
- Push'n squash: pode ser empurrado livremente através do bloco 0, mas aspreza e desaparece do mapa em contato com qualquer outro. Empurre blocos automaticamente quebra de tela, se você não quer isso, você precisará criar sua própria borda de bloco sólido.
- Push'n lock: um bloco especializado, um pouco como um bloco Sokoban. O bloco de pressão só pode ser empurrado através do bloco 0 e, quando entra em contato com o bloco 4, ele se torna o bloco 5 e o contador 3 é diminuído. Por este motivo, se estiver usando, certifique-se de manter os blocos 4 e 5 livres. Empurre blocos automaticamente quebra de tela, se você não quer isso, você precisará criar sua própria borda de bloco sólido.
- Imprimir mensagem 1: imprime uma mensagem de 32 caracteres na tela. Esta mensagem está definida na opção 8 "Editar texto do jogo". O mesmo para as mensagens 2 e 3.
- Imprimir mensagem 2
- Imprimir mensagem 3

### 3- Designer de fontes

Um designer não muito sofisticado. QAOP-Space para fazer os caracteres, 1 e 2 para percorrê-los. 'B' torna toda a fonte em negrito, 'I' torna toda a fonte em itálico. "R" remove seu trabalho e o substitui pela fonte regular Sinclair. Portanto, tenha cuidado, é muito fácil estragar sua fonte com um pressionamento de tecla.

Se você quiser carregar sua própria fonte, os 768 bytes devem ser inseridos no endereço de memória 52451

\* editores de fontes consideravelmente melhores estão disponíveis.

#### 4- Designer de mapas

Siga as instruções para editar as 9 telas disponíveis.

Nem todas as chaves estão listadas no utilitário;

- Teclas **QAOP-Space** para mover o mapa e inserir blocos.
- Teclas **1** e **2** percorrem o bloco que você deseja inserir. Move-se por todos os blocos 0-32 mais os sprites. Então, para inserir o jogador ou um bandido você precisa encontrá-los nesses blocos e inseri-lo na posição.
- Tecla **X** - limpa a tela inteira com o bloco 0.
- Tecla **C** - copia a tela inteira para a área de transferência
- Tecla **V** - cola o conteúdo da área de transferência na edição atual da tela
- Portanto, tenha cuidado, as teclas **X** e **V** podem arruinar seu trabalho com uma tecla pressionada.

#### 5- Gerente de eventos

- Define algumas condições para eventos do jogo. É possível fazer um jogo jogável usando apenas blocos, então é seguro deixar todos como "não fazer nada" se você quiser.
- Tecla **1**: percorre o efeito que leva ao nível que está sendo concluído com sucesso. Por exemplo, um jogo estilo frogger pode ser criado com o "jogador no topo".
- Tecla **2**: percorre o efeito que leva ao nível de falha e o jogador perde uma vida.
- Teclas **3** e **4**: percorre o efeito quando o player ou baddy encontra a borda da tela.
- Wrap de tela: ao encontrar a borda da tela, o sprite aparece na borda oposta
- Wrap aleatório: ao encontrar a borda da tela, o sprite aparece em uma posição aleatória na borda oposta
- Próxima tela: torna o jogo um jogo de flip screen, a próxima tela ao longo ou verticalmente é movida para.

#### 6- Contadores de itens

Define os contadores de jogo 1,2,3 para um determinado número no início de um nível. Útil para objetos colecionáveis que manipulam os contadores.

**N** - Defina os contadores para um número fixo no início de cada nível

Use as teclas **1-2**, **Q-W** e **A-S** para aumentar e diminuir os contadores 1,2,3 respectivamente.

**B** - define os contadores para uma abundância de blocos em particular no início de cada nível.

Esta função evita que você tenha que contar um determinado bloco que é grande em número. No início de um nível, o programa conta os números do bloco selecionado e define o contador para o mesmo número. Isso economiza a contagem trabalhosa e permite um número diferente por nível (diferentemente da escolha do número fixo).

Por esse motivo, você só pode definir todos os contadores para um número fixo ou blocos, não uma combinação dos dois.

#### 7 - Variáveis

Use as teclas especificadas para aumentar e diminuir as variáveis na tela.

Se o temporizador estiver configurado para zero, ele não será impresso na tela durante o jogo.

Se estiver fazendo um jogo de flip-screen, é melhor definir o nível para 1.

## 8 - Texto do jogo

Este é o texto que aparece na parte inferior de cada nível / tela.

Pressione o número do nível que você deseja editar, ou B, G, C para a linha inferior, (mensagem de fim de jogo, tela de parabéns), depois digite o texto. Pressione A, S, D para editar na mensagem do jogo 1,2,3 respectivamente. Enquanto digita pressionando 'ENTER', retorna.

## 9 - Efeitos

Você tem dois efeitos para incluir quando o jogo é completado ou o jogo acaba. Pressione as teclas especificadas para percorrer o número (limitado) de efeitos, as descrições na tela devem ser autoexplicativas.

### **I - Texto introscreen**

Este é o parágrafo que aparece na parte inferior da tela de introdução do jogo. Você tem 256 espaços de caracteres para isso. Pressione 'ENTER' para sair.

### **C - Cores dos jogos**

Siga as instruções na tela para definir as cores do papel, borda e da tinta para o jogo na tela, tela de game over e tela de conclusão do jogo. "E" sai.

### **T - Jogo de teste**

Isso permite que você jogue o jogo com suas configurações atuais. A tecla "Z" passa pelos níveis até a conclusão. Tenha cuidado com o modo como você configurou as coisas caso fique preso, pois embora eu não tenha conseguido isso em versões recentes, isso pode ser possível. Ainda é uma boa ideia salvar um instantâneo antes do teste.

### **S - Salve o jogo**

Para quando você terminar de editar seu jogo. Um bom momento para salvar um instantâneo do emulador, caso haja problemas.

Isso sai para o BASIC, então você pode salvar o código em um arquivo TAP (ou até mesmo fita real, se você é hardcore). O código do jogo ocupa os endereços de memória 32512 a 53662. Ao retornar ao BASIC, você precisa digitar SAVE 32512,21150 e salvar em uma fita inserida. O endereço de entrada do jogo é 34051, portanto, para iniciar o jogo neste código, digite RANDOMIZE USR 34051.

Se você quiser retomar ao editor de jogos do BASIC, digite CLEAR 26999 e, em seguida, RANDOMIZE USR 27000.

## **Jogo introscreen**

A tela de introdução do jogo padrão usa uma combinação de:

- Título do jogo, conforme inserido no texto do jogo da opção 8
- Uma imagem ampliada, que é o bloco 31
- 256 caracteres de texto, conforme inserido na opção "I" - texto introscreen.

## **Música Introscreen**

Há uma grande quantidade de memória sobressalente (~ 11K) acima do jogo. Qualquer música pode ser armazenada aqui.

Para ativar a música introscreen (exemplo usa música armazenada em 64000):

- Carregue sua música começando no endereço de memória 64000 (embora você possa ir tão baixo quanto 53662).
- Poke endereço de memória 34117 com o endereço de memória de início de música.
- Por exemplo, se sua música está em 64000 você Pokeia 34117,0 e 34118,250 (250 x 256 + 0 = 64000).

## Notas

- No início do jogo, todas as bandeiras estão DESLIGADAS.
- No início do jogo gravidade está desligado, você pode inserir blocos para ligar no início do jogo.
- A imagem ampliada no introscreen usa o bloco #31.

## Jogos de tela flip

Se você quiser criar um jogo flip-screen (em oposição a um jogo baseado em nível de batida), é necessário: No gerenciador de eventos, 'quando o jogador encontra a borda da tela' em NEXT SCREEN e em Variáveis, defina o número de níveis para 1 (pode haver ocasiões em que você não deseja fazer isso).

Não insira o sprite do jogador no mapa, em vez disso POKE 34138, 'jogador xcoordenada (0-31)', POKE 34143, 'jogador ycoordenada (0-21)', que será a posição em que o jogador começa o jogo.

## CHAMADAS Redundantes

Isso é principalmente para programadores em assembler. Existem algumas CALLs redundantes no 38094 (que é apenas uma instrução RET) através do código que pode ser sobrescrito para seus próprios propósitos.

- 34051 CALL 38099; A primeira coisa feita no carregamento
- 34114 CALL 38099; Logo após a introscreen ser desenhada
- 34117 CALL 38099; Logo após o acima, e antes do gamestart keyread. Este é um bom lugar para chamar uma rotina de música.
- 34286 CALL 38099; As seguintes duas CALLs estão no meio do loop de jogo principal
- 34289 CALL 38099;
- 34389 CALL 38099; Pouco antes da tela do jogo ser desenhado
- 34421 CALL 38099; Logo após o jogo sobre tela pressione a tecla
- 34425 CALL 38099; Quando um nível é concluído
- 34458 CALL 38099; Pouco antes do jogo terminar a tela é desenhada
- 34490 CALL 38099; Logo após o jogo completou keypress e return para reiniciar

## POKES

Há um número de pequenas mudanças que podem ser feitas com pokes, o que pode ser feito a qualquer momento (ou seja, é seguro fazê-lo enquanto o editor estiver ativo).

POKE 34138,xcor (0 até 31) Coordenada X de início do jogador (para jogo multiscreen)

POKE 34143,ycor (0 até 21) Coordenada Y de início do jogador (para jogo com várias telas)

POKE 35181,n poke uma cor diferente para o efeito de 'tela vermelha'

POKE 39326,255 gravidade LIGADA (o padrão é desligado)

POKE 39327,n força da gravidade n, 3,7,15,31 são os únicos números trabalháveis mais alto = mais forte

POKE 38237,n bloco # usado para excluir um bloco 'contador de dec 1', o padrão é 1

POKE 38380,n bloco # usado para excluir um bloco 'contador de dec 2', o padrão é 2

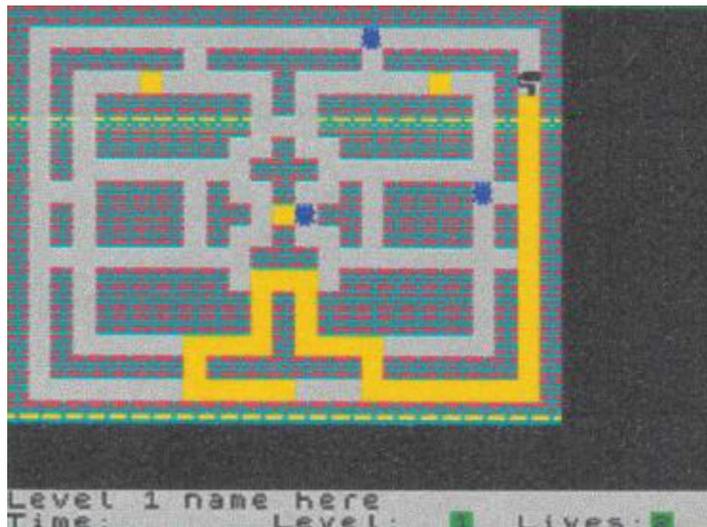
POKE 38393,n bloco # usado para excluir um bloco 'contador de dec 3', o padrão é 3

POKE 39312,255 Faz um ruído de 'tique' cada vez que o jogador se move.

## ALGUNS EXEMPLOS

\* Os seguintes exemplos para download usam uma versão antiga do C.G.D, Então são melhor usados para inspiração e não como modelos.

## EXEMPLO 1: UM JOGO DE PINTOR SIMPLES

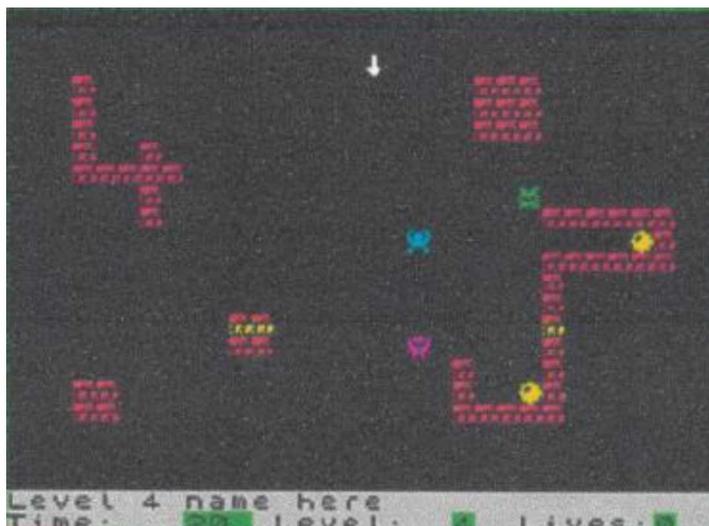


1. No designer de sprite, crie seu player sprite direita / esquerda / acima / abaixo nos UDGs 0,1,2,3. No UDG 0, selecione "Teclas de estilo Pacman". Eu acho a velocidade 8 mais adequada, mas veja o que você gosta. Crie seu gráfico baddy no UDG 4, neste exemplo eu escolhi a velocidade 8, patrulha comportamento-diagonal e kill player.

Divirta-se com vários comportamentos e você verá alguns trabalhos melhores dependendo da abertura do labirinto. Não se esqueça que você pode ter muito mais tipos de sprites do que aquele neste exemplo.

2. Agora o designer de bloco, crie o seguinte:
  - Bloco 0 - bloco de fundo, este bloco é usado para excluir os sprites do mapa quando a tela é desenhada pela primeira vez. Propriedade: não faça nada para ambos.
  - Bloco 1 - O bloco "pintado". Propriedade: não faça nada para ambos.
  - Bloco 6 - O bloco "sem pintura". Isso pode ser qualquer bloco, mas acho melhor manter 1,2,3,4,5, como eles são usados em alguns outros tipos de blocos especializados que você pode querer usar mais tarde. Propriedade: dec contador 1 para o jogador, não faça nada por baddy.Assim, toda vez que o jogador cair em um bloco "dec contador 1", ele é excluído com o tipo de bloco 1 e o contador 1 é diminuído em 1.
3. Bloco 7 - O bloco sólido. Selecione "bloco sólido" tanto para o jogador quanto para o baddy.
3. Vá para Contadores de itens e pressione 'B', isso define automaticamente o contador para um número de bloco específico. Defina o contador 1 para o bloco "sem pintura". Assim, toda vez que a tela é desenhada, o programa conta o número do bloco 6 e configura o contador 1 para esse número. Não se preocupe com os contadores 2 e 3 neste exemplo.
4. Vá para o Gerenciador de eventos, configure a tela como concluída quando: para "Contador 1 zero".
5. Vá para Variáveis. Para este exemplo, defina o número de níveis para 1.
6. Agora crie seu mapa. Este exemplo usa um pequeno labirinto simples. Lembre-se de que o contador está definido como "bloco sem pintura", portanto, você pode coletar TODOS os blocos "sem pintura" na tela. Encontre o sprite do jogador rolando pelos UDGs e insira-o onde você quer que ele comece, e faça o mesmo com quantos baddies quiser (no máximo 11).
7. Teste sua obra-prima. Este é muito difícil!
8. Aqui está um link para o z80:  
<https://docs.google.com/file/d/0BvxiMYbPnlUdV2R4eTJuQVpVWm8/edit?usp=sharing>

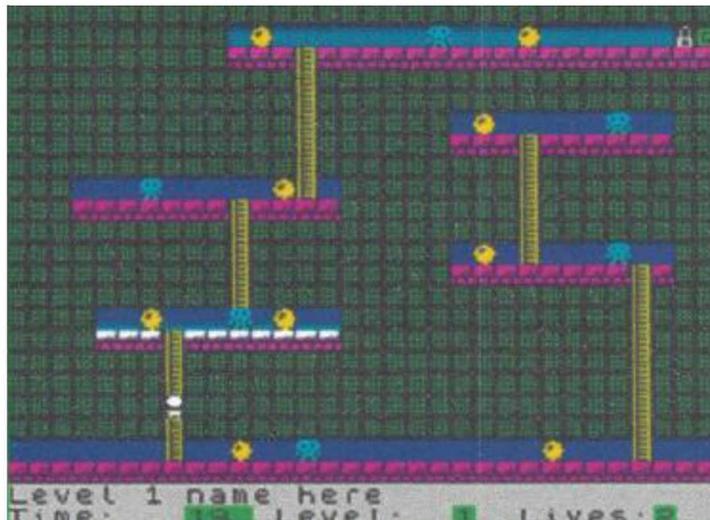
## EXEMPLO 2: UM JOGO DE TELA SIMPLES



Neste exemplo, você coleta um total de 18 moedas amarelas nas 9 telas.

1. Crie o sprite do seu jogador e dê as propriedades que você deseja. Idem baddies.
2. Crie blocos, neste exemplo simples há apenas um objeto para coletar e é um bloco "dec contador 1".
3. Na mudança do Gerenciador de Eventos, "quando o jogador encontra a borda da tela" para a "próxima tela". Neste exemplo, também estou definindo "Tela completa quando" para "contador 1 zero". Então, quando todas as moedas são coletadas, o jogo termina. Neste caso, a tela realmente significa jogo inteiro.
4. Contadores de itens, devemos selecionar a opção "N" aqui. Como há duas moedas por tela e existem nove telas, definimos o contador 1 a 18. É melhor alterar o número de vidas para 0, não precisa ser isso, mas a reinicialização padrão redefine os contadores que provavelmente irão redefinir também seu jogo, há uma maneira de contornar isso com pokes, veja a seção pokes para isso.
5. Nas variáveis, altere o número de níveis para 1.
6. Insira seus blocos e sprites duvidosos nas 9 telas. Não insira um sprite player, como se você fizesse isso quando for para a tela que é a posição em que você irá aparecer. A posição inicial padrão é x,y 0,0. Se você quiser uma posição inicial diferente, você pode POKE o valor em:
  - POKE 34138, xcor (número 0 a 31)
  - POKE 34143. vcor (número 0 a 21)
7. Faça um teste (outro difícil com apenas uma vida!) E pense em quantas coisas você pode mudar aqui!
8. Um z80 pode ser baixado aqui:  
<https://docs.google.com/file/d/OBvxiMYbPnlUdYVilekhSTGpNWWc/edit?usp=sharing>

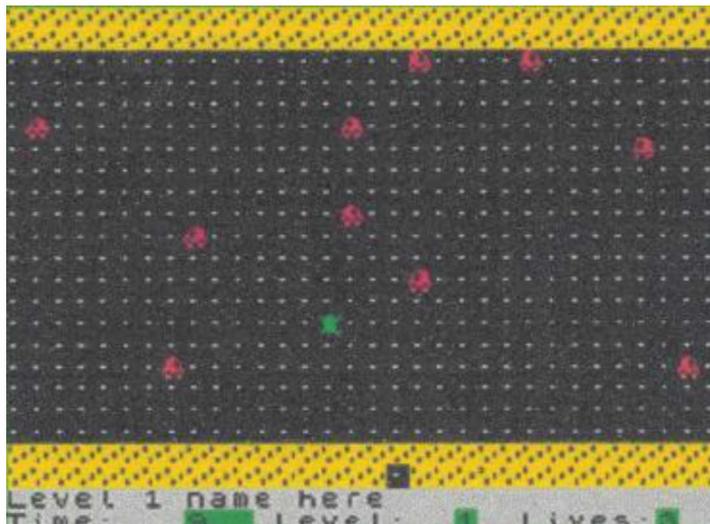
### EXEMPLO 3: UM JOGO DE PLATAFORMA SIMPLES



Jogos de plataforma não são os mais adequados para a CGD, mas uma tentativa bastante decente pode ser feita. Este exemplo é uma plataforma simples e escadas, recolher as moedas e ir para a saída. Não há truques aqui, as escadas não são nem mesmo um bloco especial, é principalmente sobre a colocação correta de blocos sólidos no mapa e estar ciente de que blocos excluíram objetos e sprites colecionáveis.

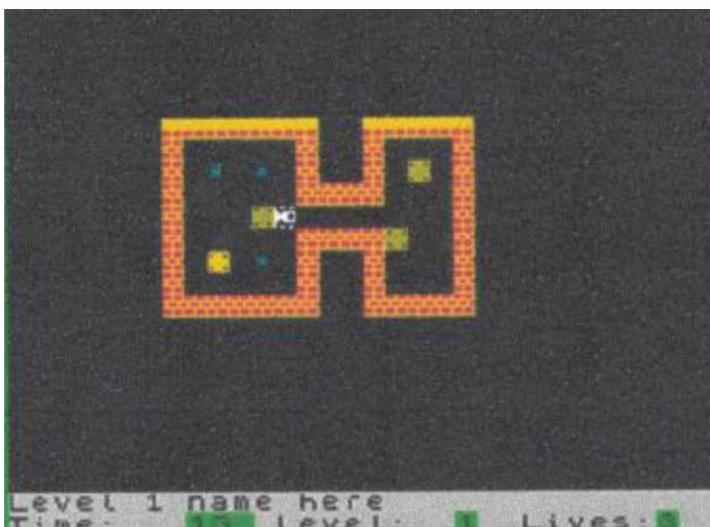
1. Projetar seu jogador e sprites baddies. Neste exemplo, acho que a pressionamento de várias teclas é a melhor chave de leitura.
2. Blocos: A maior parte do mapa é na verdade composta de blocos sólidos para restringir os sprites a se moverem em corredores. Neste exemplo, observe como o fundo verde e os tijolos magenta são os blocos sólidos, e todos os outros podem ser movidos. O nível termina com um bloco de "salto de nível" no canto superior direito, bloqueado por trás de um bloqueio de "Contador 1 <> 0: bloqueado". Isso significa que o cadeado está bloqueado até que todas as moedas sejam coletadas. Assegure-se de que os blocos 0,1,2,3 não sejam blocos sólidos, pois esses blocos são usados para sprites e contador delecionavel.
3. Nos Contadores de itens, selecionei "B", defino contadores como blocos e defino o contador 1 como o bloco de moedas. Isto não é essencial, e. você pode usar a opção "N" e defini-la para um número específico, mas isso significa que você deve ter o mesmo número de objetos colecionáveis em todas as telas.
4. Faça um teste e pense em quantas maneiras isso pode variar.
5. O z80 está disponível aqui:  
<https://docs.google.com/file/d/0BvxiMYbPnlüdMVpfdzdDekk3RZA/edit?usp=sharing>

#### EXEMPLO 4: UM JOGO DE SIMPLES DE FROGGER



1. Crie seus sprites. Neste exemplo eu fui para o sapo na velocidade 6 com as teclas de estilo Pacman. Crie quantos veículos quiser, com velocidades variadas. É aqui que você usa `type: right only` e `left only`. Neste exemplo simples, apenas 2 carros, velocidade 8, um à esquerda e outro à direita.
2. Blocos. O bloco 0 é o bloqueio de estrada, a propriedade "não faz nada". O bloco 1 é o pavimento, não faz nada para o jogador, bloco sólido para o baddy (isto é para parar os carros que aparecem aleatoriamente nas 2 linhas de cima ou de baixo).
3. Mapa. Um layout simples de 2 linhas de grama na parte superior e inferior, com o resto como estrada. Inserir o sapo na parte inferior e espaçar 10 sprites de carro (5 à esquerda, 5 à direita) através da estrada.
4. Gerenciador de Eventos. Altere "Tela concluída quando:" para o jogador na parte superior. Alterar "A tela falha quando:" para o cronômetro zero. Altere "Quando o mau caminho atende à borda da tela:" ao envoltório aleatório.
5. Variáveis, neste exemplo, alteram o número de níveis para 1 e o temporizador para 10.
6. POKE 39312,255 Isso faz um barulho agradável de "tick" a cada movimento.
7. Aqui está o z80, pense em quantas maneiras ele pode ser alterado:  
<https://docs.google.com/file/d/0BvxiMYbPnlUdbGllSmo2WmJoZUE/edit?usp=sharing>

#### EXEMPLO 5: UM JOGO DE SOKOBAN SIMPLES



Este é principalmente um exemplo de como funciona o bloco de bloqueio "push", não é

exatamente como o Sokoban, uma vez que ele atinge o alvo que ele bloqueia na posição. Antes de tentar isso, veja as propriedades do bloco "push'n lock" na seção "Blocos de design" deste documento.

1. Crie um sprite de jogador, não há sprites de baddy neste exemplo. Sou a favor da velocidade 8 e da tecla única.
2. Blocos. O bloco 0 é a cor de fundo e 'não faz nada' (e o único bloco que um bloco de pressão pode ser empurrado), o bloco 4 é o bloco de destino (geralmente deixa como nada), o bloco 4 se transforma no bloco 5 quando o bloco de alvo. O bloco 6 é o bloco "push'n lock", mas pode ser qualquer número de bloco. Lembre-se de que quando o bloco "push'n lock" atinge o seu alvo, o contador 3 é diminuído.
3. Gerenciador de Eventos. Defina "tela completa quando" para "contra 3 zero". Isso significa que quando o último bloco é colocado no lugar, o nível está completo.
4. Contadores de itens. Selecione a opção "B" e defina o contador 3 para o bloco 4, o bloco de destino. Usar a opção "B" significa que você pode ter um número diferente de alvos por nível.
5. Variáveis. Neste exemplo, o número de níveis é definido como 1.
6. Crie seu mapa. Este exemplo não é um quebra-cabeça particularmente desafiador.
7. Teste e pense nas possibilidades! (Há também o bloco de empurrar e o bloco de push'n squash que são menos especializados). Aqui está o z80:  
<https://docs.google.com/file/d/0BvxiMYbPnlUdSWNSeiEvQzJ0dWs/edit7usD-sharing>

### Salvando seu jogo na TAP (ou até mesmo fita real!)

Depois de ter terminado a sua obra-prima, você vai querer colocar tudo junto, possivelmente com uma tela de carregamento. Aqui está uma maneira de fazer isso:

1. Primeiro salve o código do jogo. Pressione S (duas vezes) no designer e ele retornará ao BASIC, insira uma fita (por exemplo, no emulador ZX SPIN em Gravação> Gravação de fita>. Insira a fita para salvar. Localize um arquivo TAP (você pode copiar e substituir um arquivo já existente), e clique em salvar. A fita agora está no gravador virtual e você está pronto para continuar. Tecle SAVE "gamenome" CODE 32512,21050 e pressione ENTER, a mensagem "Start tape then pressione qualquer tecla aparecerá. Pressione enter e ele salvará automaticamente em fita Em Gravação> Gravação de fita, selecione ejetar fita de salvamento. Seu código de jogo agora está seguro.
2. Carregando telas, se você quiser fazer um ZX-Paintbrush (<http://www.zxmodules.de/>) é a melhor ferramenta que eu encontrei para isso. Crie sua incrível tela de carregamento e salve-a como um arquivo TAP.
3. Agora, junte tudo. Em primeiro lugar insira uma nova fita (como no passo 1).
4. Em primeiro lugar, precisamos de um carregador BASIC, isso será executado automaticamente, carregar a tela de carregamento e, em seguida, o código do jogo e, em seguida, iniciar automaticamente o jogo. Aqui está como eu faço isso, digite o seguinte:

```
10 CLEAR 32511:BORDER 0:LOAD ""SCREEN$:POKE 23739,111:LOAD""CODE:PAUSE 0:RANDOMIZE USR 34051
```

A instrução clara é importante para impedir que quaisquer comandos básicos sobrescrevam o código do jogo. Eu escolhi a borda 0 porque isso é o mesmo que o plano de fundo da tela de carregamento. Load"" Screen\$ coloca sua tela de carregamento na tela. POKE 23739,111 interrompe a mensagem "Bytes" substituindo sua tela de carregamento quando o código do jogo é carregado. Load"" código carrega o código do jogo. Pausa 0 é uma pausa muito longa (ou espera para pressionar a tecla) para que você possa apreciar a tela de carregamento antes do jogo começar. Randomize USR 34051 é o endereço de entrada do jogo e inicia-o para nós.

5. Salve o carregador acima em fita com SAVE "nome" LINE 1. A "linha 1" é importante, pois significa que você não precisa digitar RUN - ele será iniciado automaticamente.

6. Agora a tela de carregamento. Digite CLEAR 32767 e pressione ENTER. Digite LOAD "" CODE 32768 e carregue o arquivo TAP da tela de carregamento. A tela agora está carregada do endereço de memória 32768 (isso é melhor do que carregá-lo na tela e salvá-lo, já que você não perde o bit da borda inferior dessa maneira). Digite SAVE "screenname" CODE 32768,6912 e pressione enter. Isso salva o código da tela de carregamento como o segundo bloco da fita.
7. Agora, para o código do jogo. Digite CLEAR 32511 e digite LOAD "" CODE, depois carregue o arquivo TAP do seu jogo. Agora digite SAVE "gamenome" CODE 32512,21150 e pressione ENTER.
8. Ejete a fita salva - você terminou. Faça um teste e esperamos que funcione!
9. Por favor, aproveite CGD com responsabilidade.

<https://worldofspectrum.org/forums/discussion/44455/redirect/p121/07/13>

EDIT > This version has a bug, for now POKE 29326,250, POKE 29335,250 solves it.

Updated version to follow.

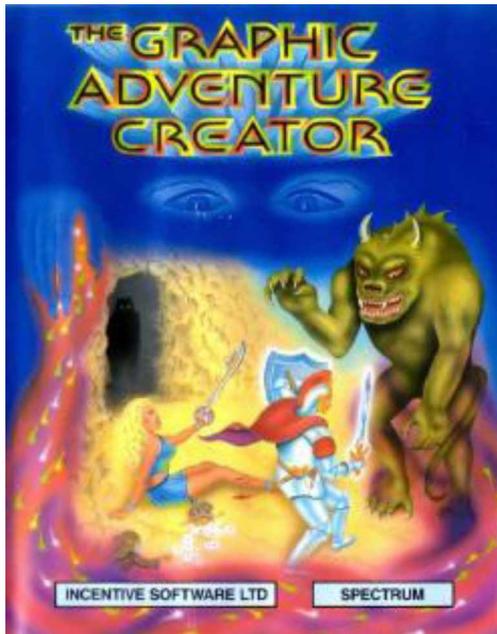
\*\*\*BUG REPORT\*\*\*

Eu descobri porque a fonte estava sendo corrompida, quando você pressiona 'C' no editor de mapas, ele copia o mapa atual para a área de transferência. São esses dados que estão sobrescrevendo a fonte, coloquei a área de transferência em um lugar 'seguro' no início e esqueci-me dela enquanto o resto do código crescia sobre ela.  
Por enquanto POKE 29326,250, POKE 29335,250 resolve isso.

Desculpas. Uma versão atualizada seguirá.

# G.A.C., The Graphic Adventure Creator

15 de Março de 2007 |  Autor: Luiz Fernandes de Moraes (autor de "A Lenda da Gávea")  
FONTE: Revista MICRO SISTEMAS No. 83, páginas 39, 40, 41 e 42 (Outubro de 1988)



Capa do cassete original



Tela de abertura do programa



Carregamento do jogo A Lenda da Gávea



Tela do jogo A Lenda da Gávea

Uma das coisas que eu nunca consegui entender bem foi o descaso dos programadores do ZX Spectrum com relação a alguns programas incríveis que essa máquina e os seus compatíveis nacionais possuem. Não estou falando de jogos. Estou falando de alguns utilitários realmente surpreendentes, largamente difundidos no mercado, e que possibilitam a criação de programas de excelente nível.

Dentre esses utilitários, um dos melhores e o GAC, The Graphic Adventure Creator, da empresa inglesa Incentive Software Ltd. Como explicar que após dois anos no mercado, somente um adventure nacional foi criado com o auxílio deste poderoso editor?

Lembro-me que quando escrevi a Lenda da Gávea, eu acreditava que seria apenas o primeiro autor nacional a me utilizar do GAC. No meu modo de ver as coisas, achava que, em seguida viriam dezenas de adventures feitos por outros autores.

Infelizmente eu estava enganado e, a despeito da ótima vendagem da "Lenda", comecei a pensar que os nossos programadores não estavam interessados em adventures. Mas eu estava novamente enganado (será que não dou uma dentro?). O número de pessoas que escreve para MS comprova o grande interesse e mostra que a principal lacuna e a falta de informações

sobre o GAC.

Mas não há de ser nada! Talvez ainda haja tempo para contornar o problema.

---

## A FILOSOFIA DE TRABALHO DO GAC

Uma das melhores coisas do GAC é a filosofia adotada na sua concepção, que permite criar adventures ilustrados com um mínimo de metodologia. Não existe uma ordem preestabelecida para trabalhar. Você pode fazer uma ilustração, criar um novo local, criar novos objetos, testar o jogo, criar um novo verbo, testar novamente. etc.

A parte do programa que edita o jogo e a parte que o executa convivem perfeitamente na memória. O único inconveniente é que isso acarreta uma certa perda de espaço para o desenvolvimento de jogos muito extensos. Mas esse inconveniente resulta num fato primordial: em qualquer momento podemos testar o nosso adventure. Com o teste on-line, podemos ir aperfeiçoando o jogo e eliminando os bugs que possam ter permanecido.

Isso é muito importante para testarmos se não houve erro de lógica em alguma das instruções do jogo. Esse tipo de erro é mais frequente quando o programador ainda não entendeu perfeitamente a forma como o GAC lida com cada tipo de evento do jogo.

Outro dado importante é que o editor de telas do GAC é ao mesmo tempo simples e eficiente. Ele possui a função UNDO e o recurso de ELASTIC, que permite traçarmos retas ou círculos acompanhando a formação do traço que está sendo feito na tela.

Cada tela do GAC é armazenada como sequência de comandos em um buffer próprio. Esse buffer não é fixo e permite até 400 bytes por tela, isto é, uma média de 150 comandos gráficos do tipo DRAW, CIRCLE, INK, PAPER, BORDER, FILL, etc. Isso é suficiente para criarmos uma tela bastante detalhada, mas pode resultar em um jogo com poucas telas, já que quanto mais detalhes, maior o espaço ocupado.

Fazer desenhos cheios de pormenores requer um planejamento bem estruturado, para que possamos montar uma tela através de várias outras telas (ou cenas). Isso é relativamente fácil de conseguir, pois o GAC permite o "merge" de telas. Você pode definir a tela 10 como sendo a tela 1 + tela 3 + tela 5 + tela 200.

Como exemplo, o adventure A Lenda da Gávea possui 56 telas diferentes, sendo todas bastante detalhadas. Isso foi possível criando-se trinta pequenas telas que continham cenários, detalhes de árvores, detalhes de trilhas na floresta e várias mascaras de montagem. Cada tela era formada por, no mínimo, duas outras telas básicas com partes do cenário, acrescida dos detalhes individuais de cada cenário. Em suma, só foi gasto o espaço de 38 telas, mas o jogador pode visualizar 56.

---

## O TRATAMENTO DE EVENTOS DO GAC

O GAC trabalha com três níveis de condições: as condições primárias (high priority conditions), as condições secundárias (low priority conditions) e as condições locais. É fundamental entender o que isto significa a nível de processamento para que possamos construir um adventure. As condições primárias são testadas ANTES de ser liberado o cursor para um input do jogador.

As condições secundárias e locais são testadas DEPOIS do jogador digitar um novo comando. Para entender melhor, imaginemos que você está jogando a "Lenda" e se encontra no primeiro local. Você digita SUL e se movimenta para outro local. Antes de se formar a nova tela e ANTES de surgir o cursor com a frase O QUE VOCÊ PRETENDE FAZER, são testadas as condições primárias.

Se você estivesse carregando uma cobra, seria picado por ela e, ao invés da tela do local, surgiria a tela da "morte". A sua pontuação seria mostrada e, em seguida o jogo

recomeçaria.

É para isso que serve a condição primária: ela é testada no momento em que entramos em uma nova área e não interage com o jogador. Todas as funções automáticas, transparentes ao jogador (atualização de ponteiros, inicialização da partida, etc.) devem ser tratadas como condições primárias.

Mas digamos que não exista uma cobra. Você chega ao novo local e digita BEBA ÁGUA. Como não existe uma condição local (façamos de conta que não), a tabela de condições secundárias é testada. Se você tiver um cantil com água você poderá beber. Caso contrário você receberá a resposta AQUI NÃO TEM NADA DISSO.

Beber água é uma condição secundária porque o ato de beber resulta de uma ordem expressa do jogador e a ação deve ser executável em QUALQUER LOCAL, desde que existam condições para isso (um cantil cheio, por exemplo).

Mas existem ações que só podem ser realizadas em um determinado local. Suponhamos que você esteja na frente do casebre e diga ENTRE NO CASEBRE. Esta ação não pode ser feita em nenhum outro lugar. Sendo assim ela deve ser considerada como condição local.

O processamento do GAC ocorre da seguinte forma: ao chegar a um novo local ele testa as condições primárias, libera o cursor para uma ordem do jogador e, após a ordem, testa se existe uma condição local para aquela área do jogo. Caso não haja, testa a tabela de condições secundárias até encontrar uma condição válida ou até chegar ao fim da tabela.

É possível escrever um adventure usando apenas as tabelas de condições primárias e condições secundárias. Só que isso resultaria numa perda de tempo de processamento após a ordem do jogador, pois a tabela de condições secundárias ficaria muito extensa. Ficou claro?



## VERBOS, ADVÉRBIOS, SUBSTANTIVOS E OBJETOS

As tabelas de verbos (verbs), advérbios (adverbs), substantivos (nouns) e objetos (objects) são construídas a partir de um índice numérico.

O verbo deve ser interpretado como item principal de uma frase-comando do jogador. Nesse caso as palavras NORTE, SUL, LESTE e OESTE, devem ser consideradas como verbos (se meu professor de português visse isso... benza Deus!).

Os verbos podem ter sinônimos. O sistema reconhece como sinônimos os verbos que possuem o mesmo índice numérico, exemplo:  
1-ANDE, 1-CAMINHE, 1-PERCORRA.

Como todos possuem o mesmo índice (1), não existe diferença entre digitar ANDE e CAMINHE. Além disso você pode digitar palavras abreviadas. Nesse caso, é bom tomar cuidado com a ordem das palavras na tabela. Digitar PE pode significar PEGUE ou PERCORRA, dependendo apenas de qual verbo foi definido primeiro.

Os advérbios atuam da mesma maneira, sendo que normalmente são pouco utilizados em um adventure. Exemplo: se você digitar PEGUE A NITROGLICERINA, você poderia explodir. Se você digitasse PEGUE GENTILMENTE A NITROGLICERINA, você pegaria e continuaria vivo para contar a história. O GAG permite isso.

Já substantivos e objetos são coisas intimamente relacionadas. Para cada objeto deve existir um substantivo de mesmo índice, exemplo: objeto 1 – uma faca; substantivo 1 - faca. Substantivos são palavras que o jogo pode reconhecer e geralmente são em maior número que os objetos (um objeto precisa de um substantivo, mas um substantivo não precisa de um objeto).

Para usarmos no jogo a frase ABRA A PORTA, não devemos criar o objeto PORTA. Basta criar o substantivo PORTA e testar a frase na tabela de condições locais (só se pode abrir portas onde existirem portas para serem abertas). Se for criado o objeto PORTA você terá

nas mãos um grave problema: alguém digitará PEGUE A PORTA e o adventure não só aceitará, como sairá carregando a porta por aí.

Não esqueça de que objetos são coisas passíveis de serem pegadas, largadas, comidas, bebidas, cheiradas, etc. Fica a seu critério considerar como objeto coisas como ÁGUA ou ÁRVORE (particularmente, acho isso arriscado).

Cada objeto deve ter definido o seu peso e o local onde ele se encontra no início do jogo. A questão do peso é importante, pois é aqui que podemos limitar o jogador, impedindo que ele possa pegar tudo o que vê. Já a questão do local pode ser burlada para fazer aparecer os objetos durante o jogo. Basta iniciar todos eles no local 0 (zero) e fazer um SWAP (troca) de locais mediante a satisfação de uma ou mais condições.

Como exemplo, você poderia iniciar o jogo com uma garrafa no local 1 e cacos de garrafa no local 0. Se o jogador pegar a garrafa e quebrá-la, um SWAP colocará a garrafa no local 0 (sumirá com ela) e colocará os cacos de garrafa no local 1. O local 0 existe para arquivar objetos que serão futuramente transportados para o jogo.

---

## AS MENSAGENS

O GAC permite até 255 mensagens de 255 caracteres cada uma (o número de verbos, advérbios, substantivos e objetos também é 255, sendo que o máximo de caracteres permitido é 35). Isso faz com que o adventure tenha uma incrível capacidade de conversação com o jogador.

O sistema utiliza as mensagens de 240 a 255 em seu processamento. Quando fizer um adventure não esqueça de incluí-las, mesmo que você só tenha mensagens de 1 a 50. Esses índices (240 em diante) são respostas automáticas do processamento. Como exemplo, sempre que o jogador digita uma frase não prevista (não existe verbo ou substantivo), o sistema imprime a mensagem 242. A mensagem 242 deve ser definida então como: DESCULPE, EU NÃO ENTENDI.

Se o jogador digitar BEBA ÁGUA e não houver água, a nível de processamento o sistema reconhece o comando, mas como ele não satisfaz nenhuma condição (secundária ou local), será impressa a mensagem 241. Sua definição correta seria algo como: LAMENTO... ISSO NÃO É POSSÍVEL.

Essas mensagens deverão ser criadas juntamente com o jogo e a correlação com os índices é a seguinte:

- 240 - O que você pretende fazer?
- 241 - Isso não é possível!
- 242 - Não entendi o que você disse!
- 243 - Digite qualquer tecla para reiniciar.
- 244 - Tem certeza? (S/N).
- 245 - Nós já temos isso.
- 246 - Nós não temos isso.
- 247 - Eu não vejo nada disso.
- 248 - Não dá para pegar mais nada.
- 249 - Você marcou
- 250 - pontos em
- 251 - Está escuro. Não consigo ver nada.
- 252 - Não encontrei o que você quer.
- 253 - Eu também posso ver.
- 254 - OK.
- 255 - Jogadas.

Estas frases são apenas sugestões e podem ser modificadas desde que mantenham o mesmo sentido. É bom tomar um cuidado especial com as mensagens 249, 250 e 255. Essas três mensagens resultam em uma quarta que dá a pontuação do jogador. A mensagem seria algo como: VOCÊ MARCOU + (contador) + PONTOS EM + (contador) + JOGADAS.

Com relação às outras mensagens, vale a pena reservar as primeiras para a descrição de

objetos. Isso é bom, pois exige um pequeno truque de programação que permite subordinar todas as descrições a uma única condição. Esse mesmo truque é utilizado para pegar ou largar qualquer objeto, testando apenas uma condição para PEGAR e uma para LARGAR. Nós veremos isso adiante quando estivermos falando da programação.

---

## CONTADORES E MARCADORES

O GAG possui 128 contadores (counters) e 255 marcadores (markers). Os contadores servem para armazenar valores numéricos (1 ou 2 bytes), e podem ser incrementados, decrementados, somados, etc. Já os marcadores podem ter apenas dois estados: 0 ou 1. Um marcador setado representa o estado de uma determinada coisa (objeto ou condição).

Exemplificando, você poderia definir o contador 10 como o contador da sede. A cada movimento do jogador ele seria incrementado em uma unidade (condição secundária) e caso estivéssemos no deserto ele seria duplicado (condição local). Quando chegasse a um número predeterminado (condição primária), o marcador 10 seria testado. Se estivesse ressetado, significaria que o jogador estaria sendo avisado pela primeira vez. O marcador 10 seria então setado, a mensagem: ESTOU COM SEDE, seria impressa e o contador 10 seria zerado.

Se o jogador pudesse beber água, o marcador 10 seria ressetado para reiniciar o ciclo (condição secundária). Quando o contador 10 chegasse novamente ao limite, estando o marcador 10 setado, saberemos que o jogador já havia sido avisado da sede e ainda não tinha bebido água. Este seria o momento de matá-lo (condição primária).

Como você pode ver, contadores e marcadores são coisas muito importantes para um adventure. Com eles podemos saber o número de pontos do jogador, a quantidade de movimentos no jogo, se uma porta foi aberta ou fechada, se o Drácula está dentro ou fora do caixão, ou qualquer outra loucura que lhe der na veneta.

Mas é bom saber que o sistema utiliza três contadores e quatro marcadores. O contador 0 acumula os pontos do jogador e os contadores 126 e 127 atuam como um contador de 2 bytes, que serve para ir guardando o número de jogadas. O marcador 0 é setado após um comando de descrição de local. Se você não mudar de local, pode imprimir uma descrição diferente mediante o teste deste marcador.

Digamos que você chegou num local e recebeu a seguinte descrição: ESTAMOS NO BATEAU MOUCHE. Se você pedir uma nova descrição e testar o marcador 0, a condição de "marcador setado" poderia indicar uma nova descrição que, como o adventure é nacional, infelizmente seria: GLUB! GLUB! GLUB...

O marcador 1 indica a iluminação do local. Se estiver setado o local é iluminado. O marcador 2 indica que o jogador possui uma fonte de luz (lâmpada, vela, etc.). Basta estar setado. Se ambos os marcadores estiverem ressetados, o sistema não descrevera os locais e imprimirá sempre a mensagem 251 (a da escuridão).

O último marcador usado pelo sistema é o de número 3. Se esse marcador for setado, não será impressa a pontuação quando se encerrar a partida. Isso permite criar um novo sistema de contagem de pontos.

---

## AS INSTRUÇÕES DO GAC

As condições são construídas usando-se as instruções de programação. Sendo assim, aqui vai a descrição de cada uma delas:

VERB x	- O verbo "x" foi digitado?
NOUN n	- O substantivo "n" foi digitado?
ADVE z	- O advérbio "z" foi digitado?
HERE x	- O objeto "x" está no mesmo local que o jogador?

CARR n	- O jogador está com o objeto "n"?
AVAI n	- O objeto "n" está disponível para uso (isto é, está aqui ou está com o jogador?)
n IN z	- O objeto "n" está no local "z"?
WEIG n	- Informa o peso do objeto "n".
SET? x	- O marcador "x" está setado?
RES? x	- O marcador "x" está ressetado?
CTR c	- Informa o valor do contador "c".
x QUE? c	- O valor do contador "c" é igual a "x"?
TURN	- Informa o número de jogadas.
ROOM	- Informa o número do local corrente.
AT z	- Estamos no local "z"?
AND	- Verdadeiro se todas as palavras foram digitadas. Ex: VERB 1 AND NOUN 1 A condição é satisfeita se ambas as palavras forem digitadas pelo jogador.
OR	- Verdadeiro se uma das palavras foi digitada.
XOR	- Verdadeiro se uma palavra foi digitada e a outra não.
NOT	- Verdadeiro se nenhuma das palavras foi digitada.
x<y	- X é menor que Y?
x>y	- X é maior que Y?
x=y	- X é igual a Y?
RAND y	- Gera um número randômico entre 0 e Z-1.
VBNO	- Informa o número do verbo em um comando.
NO1	- Informa o número do primeiro substantivo do comando.
NO2	- Informa o número do segundo substantivo do comando.
LOOK	- Descreve o local corrente.
DESC z	- Descreve o local de número "z".
PICT	- Ativa a tela de ilustração (modo normal).
TEXT	- Desativa a tela (para adventures só de texto).
GET y	- Pega o objeto número "y" (se ele estiver no local).
DROP y	- Solta o objeto número "y" (se ele estiver com o jogador).
x SWAP y	- Troca o objeto número "z".
OBJ z	- Descreve o objeto número "z".
LIST r	- Lista todos os objetos do local "r".
LIST WITH	- Lista todos os objetos em poder do jogador.
c TO r	- Move o objeto "c" para o local "r".
SET m	- Seta o marcador "m".
RESE m	- Resseta o marcador "m".
x CSET y	- Armazena o valor "x" no contador número "y".
INCR y	- Incrementa o contador "y" em uma unidade.
DECR y	- Decrementa o contador "y" em uma unidade.
GOTO z	- Vai para o local "z" e faz sua descrição.
STREx	- Determina "x" como o peso máximo para carregar.
FINDy	- Procura o objeto número "y" e move o jogador para o local onde o objeto está. Cria o efeito de mágica no adventure.
SAVE	- Salva o jogo em fita ou disco.
LOAD	- Carrega o jogo em fita ou disco (uma partida pode ser interrompida, gravada e continuada em outro dia).
WAIT	- Espera um novo comando do jogador.
OKAY	- Imprime a mensagem 254 e aguarda um novo comando.
EXIT	- Encerra o jogo sem perguntar nada ao jogador.
QUIT	- Imprime a mensagem 244 e se a resposta do jogador for afirmativa, imprime os pontos e encerra o jogo.
MESS z	- Imprime a mensagem número "z"
PRIN x	- Imprime o número "x"
LF	- Imprime um line feed.
HOLD c	- Delay. Para interromper por 10 segundos, use HOLD 500.
x+y	- Retorna o valor da soma de dois números.
x-y	- Retorna o valor da subtração de x-y.

Cada comando inicia sempre com um IF e a sintaxe é bastante simples. Os testes ou

comparações devem ficar sempre entre parênteses e a linha deve sempre encerrar com um END. Vejamos os seguintes exemplos de condições secundárias:

```
IF(VERB 7 AND NOUN 5) MESS 14 HOLD 200 EXIT END
```

Se o jogador digitar uma frase com B verbo 7 e o substantivo 5, o sistema imprime a mensagem 14, dá um tempo para o jogador ler a mensagem e encerra o jogo. Digamos que o verbo 7 seja BEIJE e o substantivo 5 seja OGRO. Se digitarmos BEIJE O OGRO, seria impressa uma mensagem do tipo: OGRO NÃO É CHEGADO! ELE TE DEU UMA PORRETADA E VOCÊ MORREU. HOLD 200 daria tempo suficiente para leitura e o jogo encerraria.

```
IF (VERB 8 AND NOUN 7 AND CARR 7) DROP 7 TO 0 MESS 17 WAIT END
```

Se o jogador digitar o verbo 8 e o substantivo 7, e estiver com o objeto 7, o sistema solta o objeto, transfere-o para o local 0 (destrói o objeto), imprime a mensagem 17 e aguarda um novo comando. Digamos que o verbo 8 seja COMA, o substantivo 7 seja RATO e o objeto 7 seja UM RATO. Se digitarmos COMA O RATO, o rato sumiria e imprimiríamos uma mensagem específica (CHOMP! HUUM... OUE RATINHO ZZZENAZIONAL!). Em seguida tornaria a aparecer o cursor, aguardando uma nova frase-comando do jogador.

Os dois exemplos são bastante simples, mas ilustram perfeitamente uma linha de comando. A estrutura é a mesma para qualquer uma das três tabelas de condições (primária, secundária e local), com a vantagem do GAC fazer um ajuste automático para a sintaxe correta no caso de erro do programador.

Falta falarmos do truque que permite pegar, soltar ou descrever qualquer objeto.

Para isso usamos a função NO1 que fornece o número do primeiro substantivo da frase. Supondo que o verbo 7 fosse PEGAR, o comando seria:

```
IF (NO1 < 10 AND VERB 7) GET NO1 OKAY END
```

Com essa estrutura poderíamos pegar qualquer objeto de número menor que 10. Esse limite é dado pelo programador. Se o número de objetos for 20, ao invés de 10 coloque 21. Ou então coloque 16 e considere os 5 últimos objetos como "impossíveis" de pegar sem algum tipo de auxílio (objeto pesado demais ou grande demais).

Para soltar ou descrever objetos, suponhamos que os respectivos verbos possuam os números 8 e 9. As instruções seriam:

```
IF (NO1 < 10 AND VERB 8) DROP NO1 OKAY END  
IF (NO1 < 10 AND VERB 9) MESS NO1 OKAY END
```

Fácil, não? Tenha apenas o cuidado de escrever suas instruções com um espaço entre cada palavra. Isso é axiomático.

## EDITANDO AS TELAS

O editor de telas é bastante simples de operar. Anote as teclas de comando e faça várias experiências antes de desenhar para valer. As teclas são as seguintes:

SETAS	- Movam o cursor.
SETAS + CAPS	- Movem o cursor de 8 em 8 pixels.
L	- Posiciona a caneta para traçar uma linha. Uma segunda pressão em "L" levanta a caneta e encerra o traço (todas as funções de "riscar" são desativadas com uma segunda pressão).
E	- Elipse ou círculo.
R	- Retângulo.
D	- Desenha' um ponto (acende um pixel).
I	- Define o INK.
P	- Define o PAPER.
V	- FLASH.
B	- BRIGHT.
T	- Cor do BORDER e da área de texto.
G	- GRID (quadriculado).
F	- FILL de INK.
A	- FILL de PAPER.
S	- FILL sombreado (SHADOW).
M	- Merge de telas.
C	- Pisca o ponto onde se situa o cursor.
SETAS+ SYMB	- Avança ou retrocede entre os passos de confecção da tela (buffer de comandos).
Z	- Salta para o início da tala.
DELETE	- Deleta o último comando.
CAPS + 9	- Deleta da posição corrente até o final do buffer de comandos de formação da tela.

As teclas CAPS + SPACE (BREAK) funcionam como uma espécie de "chave-geral" para o editor. Teclar BREAK interrompe o que quer que esteja sendo feito em qualquer etapa da confecção do adventure. Este será o seu PANIC-BUTTON. Insista no BREAK até recuperar o controle (seu e do jogo).

Está tudo aí, não faltou nada. Treine bastante! Se o seu entusiasmo coroar o meu esforço, e você resolver escrever um adventure com o GAC, aqui vão alguns conselhos: faça o mapa do jogo defina a solução da aventura e escreva tudo no papel. Defina todas as condições necessárias (também no papel) e comece o trabalho de criação das ilustrações.

Procure fazer com que após a confecção de todas as telas você ainda tenha uns 8 Kb para a programação. As rotinas de compactação do GAC permitem criar adventures gigantescos em apenas 8 Kb. Ao fim de tudo, selecione o BEGIN HERE através do menu principal, e informe em que local o adventure se inicia. Tire uma cópia do jogo (runnable) e envie de forma singela para este seu criado: afinal eu também gosto de adventures, certo?



Ficha Técnica:  
 Programa: Graphic Adventure Creator  
 Lançamento: 1986  
 Empresa: Incentive Software Ltd (UK)  
 Autores: Sean Ellis, Brendan Kelly  
 Preço: £22.95

webmaster@tk90x.com.br | www.TK90X.com.br

# **THE GRAPHIC ADVENTURE CREATOR**

SPECTRUM

INCENTIVE SOFTWARE LTD

=====  
The Graphic Adventure Creator  
=====

Document created by Martin M. Pedersen (tusk@daimi.aau.dk) on 20 Jan 97 with the following comments...

[ This text file is made from the original commodore 64 manual to GAC.

A legal notice. This text and the program is still copyrighted by INCENTIVE SOFTWARE and you may only use it if you have legally bought a copy of GAC.  
]

The following alterations were made by rstuart@ukonline.co.uk on 9th Dec 99:

1. Addition of section 1.2 'Loading Instructions'
2. Re-writing of section 3.2 'Graphics Commands' for Spectrum
3. Addition of appendicies A and B
4. Other minor alterations to make document agree with Spectrum manual
5. Addition of product description from inlay

(c) Copyright 1986 Incentive Software Ltd.

54 London Street, Reading RG1 4SQ

All rights of the producer, and of the owner of the work being produced, are reserved.

The Graphic Adventure Creator

Design By: Sean Ellis

Program By: Brendan Kelly

Cover By: Peter Carter

Ransom Pictures: Pete James

Produced By Ian Andrew

THE GRAPHIC ADVENTURE CREATOR

-----  
For the Spectrum Computer  
-----

CONTENTS :

SECTION ONE

-----

- 1.1 Introduction
- 1.2 Load instructions

SECTION TWO

-----

- 2.1 Writing Adventures
- 2.2 Verbs
- 2.3 Room Descriptions
- 2.4 Messages
- 2.5 Nouns and Objects
- 2.6 Adverbs
- 2.7 Conditions
- 2.8 Begin Where?
- 2.9 Save and Load
- 2.10 Delete Data

SECTION THREE

-----

- 3.1 Graphics
- 3.2 Graphics Commands

SECTION FOUR

-----

- 4.1 Test Adventure
- 4.2 Player Commands

APPENDIX A

-----

Definitions and Conditions for the  
Demonstration Adventure

APPENDIX B

-----

Tables and Charts

COPYRIGHT 1986 INCENTIVE SOFTWARE LTD - ALL RIGHTS RESERVED

## (1.1) INTROCUCTION

=====

Welcome to the GRAPHIC ADVENTURE CREATOR!

This manual is intended as a simple introduction to writing adventures using GAC. The GRAPHIC ADVENTURE CREATOR is the best though of as a small programming language specifically designed for writing adventure games. Although it is smaller than other programming languages such as Basic or Pascal it can perform all of the complex tasks needed to write an adventure program, and although the instructions may seem complicated at first, a little time spent studying this manual will enable you to write your very own large scale adventures, far more easily and efficiently than by using basic or machine code.

Any adventures that you write using GAC are your own work, and may be sold as such without any prior permission being sought or payment to us being made. However in this case, you should include with your adventure something to the effect that it was written using the GRAPHIC ADVENTURE CREATOR, (C) 1986 by Sean Ellis/Incentive Software.

And now on with GAC!

### CREDITS

-----

DESIGN	by	Sean Ellis
PROGRAMMED	by	Brendan Kelly
RANSOM PICTURES	by	Pete James
COVER	by	Pete Carter

Thanks also to Lesley, Giles, Dave and John

COPYRIGHT 1986 INCENTIVE SOFTWARE LTD.  
54 LONDON STREET, READING RG1 4SQ

All right of the producer, and of the owner of the work being produced, are reserved. Unauthorized copying, hiring, lending, public performance and broadcasting of this program is prohibited.

The publisher assumes no responsibility for errors, nor liability for damage arising from its use.

## (1.2) LOADING INSTRUCTIONS

=====

The Graphics Adventure Creator (GAC) cassette contains four files.

These are:

1. "GAC" - The Creator itself
2. "QS" - The Quickstart file
3. "ADVINMAN" - Adventure in Manual file
4. "RANSOM" - A small Graphic Adventure

To load **GAC**, type LOAD"" and press enter. Once loaded, QS or ADVINMAN can be loaded by selecting T from the Main menu - these are data files.

**QS** - Quickstart, sets you up with the most commonly used verbs etc. and common messages. A complete listing of its contents is contained in Appendix B, along with several other useful tables and charts.

**ADVINMAN** - Sets up the small adventure in the manual to work through and even edit!

The fourth file **RANSOM** is an example of a graphic runnable adventure created using GAC and can only be loaded from a reset machine with LOAD"".

Write an Adventure in 30 Seconds! (For a Bit of Fun!)

-----

First LOAD GAC in

Then "T" to load QS (Quickstart)

Press "R"

- Start Stopwatch!

Press "1" (enter)

Type "A Cave" (enter) (enter) (enter)

Break (Caps-shift, Space)

Press (enter)

- Stop Stopwatch!

The screen will display:

A Cave

What Now?

Try a few inputs

Presto! - A VERY simple adventure in 30 seconds?

Well look at it this way - it can only get better!

To get back to the main menu press 'Break', 'Break', 'Break'.

This simple example demonstrates how quick and easy GAC is to use. With a little time and imagination you will soon be creating professional Graphic Adventures!

Let Your Creation Begin!

## (2.1) SECTION TWO - WRITING ADVENTURES

=====

There are several elements necessary to writing adventures using GAC. These are displayed on the Main Menu which is the first thing you will see after loading the program.

For convenience the menu is arranged in alphabetical order, which makes it easier for you to remember which keys to press in order to call up any part of GAC. At the top of the menu screen you will see the display showing "FREE:" which shows the memory free.

You may write the elements of your adventure in any order you like, although you may find that certain orders may make things easier than others.

What follows is a brief description of what each element of the main menu does, together with detailed instructions of how to use each one. They are presented in an order which you may find convenient to follow.

## (2.2) VERBS

=====

Verbs are the words with which you will actually tell the Computer what to do when you are playing an adventure. For example EAT or DROP. Directions such as WEST or UP are also treated as verbs. GAC allows you to enter and edit your own verbs when writing an adventure. Some of the most commonly used verbs are already in the Quickstart File, however since GAC allows you to enter up to 255 verbs, your imagination is the only limit to the commands you may incorporate into your adventure.

After pressing V from the Main Menu, you should see a prompt on the screen, \* EDIT VERBS \*, and a pointer >.

To enter a verb, simply type in a number, then a space followed by your verb. When you press RETURN the verb will be entered on the screen in alphabetical order.

If two or more verbs have the same meaning and you wish both to be accepted, ie., "Get" and "Take", simply allocate to them the same verb number. Try to think of as many similar words as you can in this way, since this will make your adventure appear more friendly.

- EDITING - To edit, scroll the list using the cursor keys until the desired verb is adjacent to use the pointer. Press Return, edit the word, press Return again.
- DELETE - Position the word to be deleted by the pointer and press Delete.
- RETRIEVE - Home will retrieve the last word deleted.

There is a brief summary of these instructions at the bottom of the screen.

NOTE: The verbs in the Quickstart File may be edited in exactly the same way as any other verbs.

More information about the use of verbs may be found in section(2.8) CONDITIONS.

PRESSING BREAK WILL RETURN YOU TO THE MAIN MENU

### (2.3) ROOM DESCRIPTION

=====

The term "room" is used to refer to any location in your adventure. For example, "a forest", "a car", "a strange alien world" and so on. GAC allows you to use up to 9999 of these rooms, memory permitting, and to specify the connections to and from each room. You will probably find it easier if you have decided in advance what each room is and how the rooms are connected.

On pressing R from the main menu, the prompt "Which room number ? ...." should appear. At this point enter a room number n, followed by RETURN which will give the prompt "Room number n is ....". You can then enter the description which can be up to 255 characters long.

You will then see the prompt "Connections are?". The connections are specified in terms of a verb, which must part of the list already specified, followed by a space, followed by the room number it connects with. For example EAST 20.

You may have as many connections from a single room as you can fit into 255 characters, followed by RETURN, i.e.

EAST 20 WEST 18 NORTH 19 SOUTH 21 JUMP 49 (RETURN)

You will then see a prompt asking for a picture number, which links this location with the appropriate picture should you wish to have one. In this case you should enter a picture number followed by RETURN. If you do not want a picture simply press RETURN. This returns you to the prompt "Which room number ?..." again.

All of the above may be edited at any time by using the cursor keys. BREAK at any point will return you to the WHICH ROOM? prompt, and BREAK at that point will return you to the Main Menu.

More information about the use of room descriptions can be found in section (2.7) CONDITIONS.

#### EDITING TEXT AND NUMBER INPUTS (this applies to all sections)

=====

Cursor keys 5 and 8 will move the cursor left and right, DELETE (Caps-Shift 0) deletes left of the cursor and EDIT (caps-shift 1) deletes the character at the cursor position and right of the cursor.

=====

PANIC BUTTON!!!

=====

At any time press BREAK, (several times if necessary) to return to the main menu.

## (2.4) MESSAGES

=====

Messages consist of prompts, descriptions or other comments that appear on the screen when you play an adventure, i.e. "What now ?", or "You can't do that". Many of the more common System messages are contained in the Quickstart file, and are numbered from 240 to 255. GAC allows you to enter up to 255 messages which may appear at any point in your adventure,

Pressing "M" from the main menu will give the prompt "Which message number ?...\*. You should then enter a number followed by RETURN, then type in your message, which may be up to 255 characters long. Pressing RETURN will enter your message and repeat the "Which message number ?...\*" prompt.

To edit a message once it has been entered, simply recall the message and edit it using the cursor keys.

Please note that all messages from 240 to 255 should be defined, since these are essential system messages without which your adventure will not run correctly. You can edit these messages but you should be careful to preserve the meaning. For example, message 242, "Pardon ?" could be changed to "Eh ?".

Pressing BREAK will return you to the prompt EDIT MESSAGE? and BREAK from there will return you to the Main Menu.

More information about the use of messages can be found in section (2.7) CONDITIONS.

## (2.5) NOUNS AND OBJECTS

=====

Nouns are the "things" that your adventure will recognize. For example, BOOK, SWORD and so on.

GAC frequently requires you to define things both as nouns and as objects: once as a noun so that the adventure will recognize it, and then as an object so that the program can move it around and do things with it. For example, SWORD is defined as a noun, but since you may wish to pick it up during your adventure, you must also define it as an object. A further example makes clear another point: if your adventure requires you to light a lamp. LAMP must be defined as a noun, and you must also define two objects, AN UNLIT LAMP and A LIT LAMP .

When you press N from the main menu, the prompt \* EDIT NOUNS \* will appear. Apart from this, the procedure for entering and editing nouns is exactly the same as that for editing verbs. (2.3)

When you press 0 from the main menu, the prompt "Which object number will appear. You should then enter a number n from 1 to 255 followed by RETURN. You will then see the prompt "Object number n is...". You may then enter the object description consisting of up to 255 characters followed by RETURN. This will produce the prompt "Starts in room number...". Enter the number of the room you wish this object to start in, or press RETURN. Pressing RETURN will assign the object to room zero, a special room which cannot be travelled to, and which contains "dead" and "unborn" object. For example, a piece of cake once eaten will be assigned to room zero. The prompt "and weighs...." will then appear. Enter the weight you wish to allocate to this object and press RETURN.

If the start room number remains at zero, the object is assigned to ROOM ZERO , - a special room which cannot be travelled to, which contains "dead" and "unborn" objects. For example, a piece of cake once eaten will be assigned to room zero.

BREAK will return you to the main menu.

More Information about the use of nouns and objects can be found in section (2.7) CONDITIONS.

## (2.6) ADVERBS

=====

In the GAC adverbs are used for two sorts of things: firstly to describe the precise way a verb is used, i.e. "move slowly" or "put down gently", and secondly they are used to differentiate between similar nouns. For example, if your adventure has three differently coloured boxes, the adverbs will specify the colours: RED, GREEN etc.

On pressing A from the main menu, the prompt \* EDIT ADVERBS \* will appear. Apart from this the procedure for entering and editing adverbs is exactly the same as that for editing verbs and nouns.

More information on the use of adverbs can be found in section

## (2.7) CONDITIONS

=====

When using GAC the conditions form the internal commands that simulate decision-making processes within the game. All the terms you have specified and the data you have entered is now drawn together to form the structure for your adventure-

"Conditions" tell the adventure that if certain things are true, then it is to perform some action, i.e.,

"if x and y are true then do t",

or "if z and q are true then do y and x", and so on.

The GAC has a special format for writing these conditions. For example, if you have defined verb number 16 to mean "Examine", and noun number 2 to mean "Room", then a condition using these terms might read :

"IF { VERB 16 AND NOUN 2 ) LOOK WAIT END"

Translated this means if "Examine Room" is typed in, describe the current room and wait for a new command.

The Conditional list below separates the words used for the "Conditions Part", 'If ( Verb 16 and Noun 2)' and those used for the "Action Part", 'LOOK WAIT END1. For subscripts used (a, c, m, as, o, n, r, v, x and y) - See Appendix B.

### i. CONDITION PART

-----

verb v	is verb v typed ? (v is a number) VERB 7 will give a TRUE answer if verb number 7 was typed, otherwise it will give a false answer.
NOUN n	is noun n typed ? (similar to VERB)
ADVE a	is adverb a typed ? (similar to VERB)
HERE o	is object o here ? (i.e. in the same room as you ?) If you are in room 3 and object 1 is also in room 3, then HERE 1 will give a TRUE answer because object 1 is here.
CARR o	Is object o being carried ? (Similar to HERE).
AVAI o	Is object o available for use ? i.e. here or being carried? (Similar to HERE).
o IN r	Is object o in room r ? If object 1 is in room 3, as above, then 1 IN 3 will be TRUE, but 1 IN 4 will be FALSE
WEIG o	Gets the weight of object o. In our adventure, weigh 2 would give the value 20, which is the weight of the rat.
SET? m	Is marker m set ?
RES? M	Is marker m reset ? (There are 256 markers, numbered 0 to 255. They are used to store information that can be in one of two States, like doors which are open or shut, light which are on or off, etc. There are three which are important for the adventure program : Mkr 0, if set, means that a room has been described since it was last reset. Mkr 1, if set, means you are in a light room, Otherwise you are in a dark room. Mkr 2, if set, means you have a lamp or some other source of light. If markers 1 and 2 are both reset, then the program will refuse to describe rooms, coming up with the "It's dark" message instead, since you are in a dark room without a lamp. Mkr 3, if set, disables the scoring mechanism when you exit from the game.
CTR c	Gives you the value of counter c.
x EQU? c	Is x equal to the value of counter c ? There are 128 counters, numbered from 0 to 127. They are most frequently used to store the number of moves since a particular event, (e.g., moves in the dark). Counter 0 holds the score and counters 126

and 127 hold the turns count since the start of the game.  
TURN Gives the number of turns since the start of the game.

ROOM Gives you the room number of the room you're currently in.  
AT r Gives a TRUE answer if you are in room number r.

condition AND condition will give a TRUE answer if both of the conditions give TRUE answers.  
e.g. VERB 1 AND NOUN 2 will TRUE only if verb 1 and noun 2 are both typed.

condition OR condition will give TRUE if either condition is TRUE, or both.

condition XOR condition will give TRUE if one of the conditions is TRUE and the other FALSE.

NOT condition will give a TRUE answer if the condition was FALSE, and vice versa.  
So NOT VERB 1 will be TRUE if verb 1 is NOT typed.

x < y Gives TRUE if number x is less than number y  
x > y Gives TRUE if x is greater than y.  
x = y Gives TRUE if x is EQUAL to y.

RAND x Gives a random number between 0 and (x-1). So RAND 10 will give any number at random in the range 0 to 9.

VBNO Gets the number of the verb in this command.

NO1 Gets the number of the first noun in this command, if any.  
NO2 Gets the number of the second noun in this command, if any. These are used to check word order, and to GET and DROP object without having to have a condition for each one.

## ii. ACTION PART

-----

LOOK Describe the room you are in at the moment.  
DESC r

PICT Turns the picture display on.  
TEXT Turns the picture display off, giving a text adventure.

GET o Get object number o. If it isn't here, or you've already got it, or it's too heavy then the appropriate message is displayed.

DROP o Drop object number o. If you haven't got it, then the appropriate message is displayed.

x SWAP y Exchange objects x and y. In our little adventure, 1 SWAP 5 will exchange the lamp with the lit lamp.

OBJ o Describe object number o.  
LIST r List all the objects in room number r.  
LIST WITH List all the objects carried with you.  
o TO r Move object number o to room r. To destroy an object move it to room zero.

SET m Set marker m  
RESE m Reset marker m. See SET? and RES? for an explanation of markers.  
x CSET c Set x to be the value of counter number c.  
INCR c Increase counter c by one. The maximum is 255.

DECR c Decrease counter c by one. The minimum is 0.  
Trying to increase past 255 or decrease past 0 is ignored.

GOTO r Go to room r and describe the new room  
CONN v This checks through the connection table for a connection from the current room using verb v. If one is found, this returns the room number of the room you would have move to if you were to take the connection, otherwise it gives zero. As an example if you were in room

3 in our adventure,  
 CONN 4 would return the value 2, because in the connections from room 3, verb 4 (WEST) would take you to room 2.

STRE x Set the maximum weight you can carry to x. STRE is short for STRENGTH.  
 BRIN o Brings object o here, if it exists.  
 FIND o Find objects o and move to it, if it exists.  
 This does not acknowledge any restrictions, (such as connections), so it could be useful in a magic spell.

SAVE Saves the current game position to tape or disc.  
 LOAD Loads the current game position from tape or disc. These are useful for continuing a game after tea (or coffee - the kettle's probably boiled by now!)

WAIT Waits for a new command.  
 OKAY Prints "Okay" and waits for a new command.  
 EXIT Stops the game. Since the player is NOT asked, this is recommended for use when the player wins or is killed.  
 QUIT The player is first asked if he wants to quit. If he types "Y", the game terminates, otherwise the game continues. On abandoning the game, both the score and the number of moves taken are displayed, assuming this function has not been disabled by setting marker three.

MESS ms Print message number ms.  
 PRIN x Print number x.  
 LF Prints a Linefeed. Everything from hereon is printed on a new line.  
 WITH Is equal to the room number of where things are put when you are carrying them.  
 HOLD x Halts the game for x fiftieths of a second, or until a key is pressed. For example, to freeze for 10 seconds, do HOLD 500.  
 x + y As you may expect, returns the value of x added to y.  
 x - y And this gives the value of x - y.

There are several things to note when writing conditions. Firstly all operations are performed strictly from left to right. Secondly GAC requires spaces both inside and outside brackets, and between words such as 'verb' and 'noun' and their numbers. Thirdly, x and y need not be simply numbers, they can be more complex expressions. However you should note that since GAC performs all its operations from left to right, you should be careful about the order in which you enter things. Put any comparisons first, i.e. if < VERB > S ) and NOUN 9 ) QUIT END

The conditions are checked by GAC at three different points, and are entered as High Priority, Low Priority and Local. The flow chart at the back of the manual indicates the order in which these are actioned. The GAC checks for HIGH PRIORITY conditions before prompting for a player command. It checks for such things as whether you are still "alive", whether a light is flashing, etc.

Pressing H from the main menu will return the prompt "Which line number" • You should enter these lines in order, ie. 1, 2, 3, etc. The GAC will not recognize a line number such as '5' when you have only entered two conditions so far, although it will insert a condition earlier in your list and remember accordingly. RETURN will give you the prompt "Line number n is...", at which point you should enter your condition and press return which will give you the BREAK returns you to the main menu.

LOCAL CONDITIONS are checked for after the player has entered a command, and deal with things local to a particular room, for example, whether a particular object is in that room. Alternatively they might execute a player command to go to another room, or open a door at that location.

Pressing C from the main menu will give the prompt "Which room number ?... " Entering a number followed by RETURN will give the prompt "Which line number Thereafter the procedure is exactly the same as for High Priority Conditions, except that each Room has it's own set of line numbers, {1, 2, 3 etc.}).

LOW PRIORITY CONDITIONS are also checked for after a single player command is entered, but are not associated with any room. For example, they might check whether you

are carrying a particular object, independent of the location. After pressing L from the main menu, the procedure for entering and editing Low Priority Conditions is exactly the same as that for High Priority Conditions.

Further examples of High, Local and Low priority Conditions can be found in Appendix A.

#### (2.8) BEGIN WHERE?

=====

The last element of your adventure, at least as far as the text is concerned, is to specify which room your adventure starts in.

Pressing B from the main menu will give the prompt to enter the start location room number.

Entering a room number followed by RETURN means that your adventure will start at this room. To change the start room, simply enter a new number.

BREAK will return you to the main menu.

#### (2.9) SAVE AND LOAD

=====

##### SAVING

-----

After pressing S from the main menu, you will be asked if you wish to save a "Data File" or a "Runnable Adventure".

If you press 'D' for 'Data File', the Computer will prompt for a file name. Enter your file name followed by RETURN. This will save your data which can be loaded into GAC at a later date for further development.

If you press 'R' for runnable adventure followed by 'Enter' your data will be saved as a Runnable Adventure, which can no longer be loaded into GAC for editing but will run on its own.

##### LOADING

-----

Pressing T on the main menu followed by a file name will load a Data file into GAC for editing.

To load your Runnable Adventure type LOAD"" and press ENTER.

NOTE : Do not attempt to load a runnable adventure into GAC. It will run on its own !

#### (2.10) DELETE DATA

=====

Pressing X from the main menu will delete all your data. N.B. Loading a new data file will have the same effect.

#### (3.1) SECTION THREE - GRAPHICS

=====

The Graphics part of GAC will enable you to draw pictures to go with the room descriptions in the text part of your adventure.

Press G from the Main Menu to enter the Graphics sections. You will see displayed a

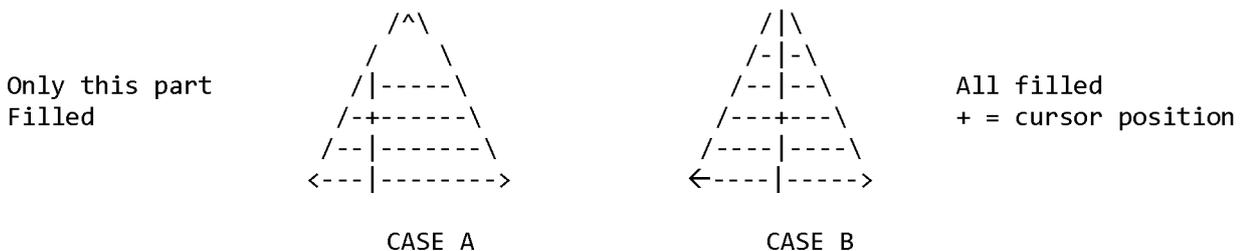
high-resolution Window and a Graphics Menu, together with prompt "Which picture number  
Enter the number of the picture you wish to design.

The Status Board from left to right:

<u>PEN AND PAPER</u>	The Pen starts "off" the Paper. The cursor will not leave a trail and is free to be positioned without affecting your picture. The Pen will be in the down position whilst actually drawing. (L, E & R>
<u>INK</u>	The colour in this box shows the current status of the Ink of the Pen.
<u>PAPER</u>	The colour here shows the current status of the background, (the paper colour).
<u>X &amp; Y</u>	These indicators show the current position of the cursor. The bottom left corner X = 0 and Y = 0.
<u>LAST</u>	This shows the last command used.
<u>PIC</u>	The picture number you are presently designing.

The fill Routine is designed to be fast and it can cope with irregular shapes, but you may need to press "F" more than once. If an area is left unfilled, then you can just move the cursor into that area and press "F" again.

Here is a simple method which will allow you to get the best results from your filling. If your shape has a high point, position the cursor directly under it. if it has a low point, position your cursor directly under it. Here is an example using a triangle...



The area filled is worked out by the fill routine checking up and down from the cursor position until it encounters a boundary.

It is possible to fill the same area more than once using shading patterns consisting of different colours.

Finally, we have another very useful feature. This is the ability to merge pictures together. For instance, you find that, in your adventure you have a single basic cave shape, which you want to use in many of your pictures, but details differ. It is possible to include a picture already created into the picture you're in, by greeting, typing the number at the picture you mark the include, and pressing ENTER. This is then drawn over the top of anything already there, (I usually do this right at the start and then draw around the merged picture). It is a very useful feature which saves a lot of time and memory too.

If you should run out of memory in one particular picture, you can get round this by moving to an unused picture and merging the full picture into the new one before continuing with your drawing.

Designing and drawing pictures on the Spectrum is hindered by the Spectrum's own 'Attribute Probletn7'. If you hold 'G' whilst at a picture, you will see a grid of bright and not so bright squares light up - each square is 8 by 8 pixels. The Spectrum itself will only allow two colours in any square at any time, one is called the Paper Colour and the other the Ink Colour.

By careful positioning of lines, ie. on the edge or corners of Attribute Squares, you can still get several areas of colour joining at one point. As you draw lines, GAC will update all the attributes as the line is elastic banded, this will aid effective positioning. Also as previously mentioned holding the 'G' key will overlay the Attribute Grid over the picture.

There is a demonstration picture (No. 99) on the 'Advinman' file that you can step through, edit, add to, delete or whatever! (Load in Advinman using the T option, press G

and enter 99).

By careful design you can create stunning graphics like those included in the Ransom adventure.

### (3.2) GRAPHICS COMMANDS

=====

- Arrow Keys (5, 6, 7 & 8) - Move the cursor
- Arrow Keys + Caps. Shift - Move the cursor (steps of 8) quicker.
- L (1st press) - Line - Fixes one end of the line, (pen goes down). Now move the cursor about.
- L (2nd press) - Line - Fixes other end of the line, (pen goes up).
- E (1st press) - Ellipse/Circle - Sets centre (now move cursors).
- E (2nd press) - Ellipse/Circle - Fixes Ellipse/Circle
- R (1st press) - Rectangle - Sets a comer (now move cursor).
- R (2nd press) - Rectangle - Fixes Rectangle.
- D - Dot - Prints a Dot at cursor position
- I - Ink Colour
- P - Paper Colour
- V - Flash 0 = off 1 = on 8 = transparent
- B - Bright 0 = off 1 = on 8 = transparent
- T - Text Area and Border Colour
- G - Grid showing Attributes positions
- F - Fills from cursor area in INK
- A - Fills from cursor area in PAPER
- S - Shaded fills from cursor in a shade of Current Ink and Current Paper
- W - Draws whole picture
- M - Merge picture - prompts for a picture number and merges that picture with the one currently displayed. Note some shades will not merge well and a 'fill' will not fill past obstacles in the present picture.
- C - Cursor - when held will highlight the cursor position.
- Arrow Keys (5, 6, 7 & 8) and Symbol Shift - Steps through your picture to allow editing.
- S Shift 5 (left arrow) - Steps back one command
- S Shift 8 (right arrow) - Steps forward one command
- S Shift 6 (down arrow) - Steps back five commands
- S Shift 7 (up arrow) - Steps forward five commands
- NB . When stepping through your picture you can still add and change the commands as normal.
- Z - Jumps to start of picture
- Delete - Deletes last command
- Caps-shift 9 - Deletes from current position to end
- BREAK 1st time - Prompts new picture number
- BREAK 2nd time - Return to main menu

NB. For a Spectrum + or Spectrum 128 the arrow keys move the cursor in steps of 8, for accurate positioning use keys 5, 6, 7 and 8. (Left, down, up and right).

#### Editing pictures

-----

- - Moves back one graphic command.
- + - Moves forward one graphic command.
- < - Moves back 8 graphics commands.
- > - Moves forward 8 graphic commands.
- SHIFT \* - Deletes all picture information held after the edit position.
- Z - Takes you back to the start of the picture.



7 BOX  
255 IT

7 GET  
8 PUT 16 EXAMINE

The command line is split into separate commands by the following : " . \* , " , " , ,  
"and" and "then", so it becomes ;

Get the gold        "\*"  
Examine it         "\*"  
Put it in the box   "then"  
go north

Each of these is canned for verbs, adverbs and nouns in that order, and may found have their number stored;

	VERB	ADVE	NOUN1	NOUN2
<u>Get the gold</u>	7	0	3	0
<u>Examine it</u>	16	0	255	0
<u>Put it in the box</u>	8	1	255	7
<u>Go north</u>	1	0	0	0

Then all occurrences of noun 255 ("It"), are replaced by the last noun typed before that, giving:

	VERB	ADVE	NOUN1	NOUN2
Get the gold	7	0	3	0
Examine it	16	0	3	0
Put it in the box	8	1	3	7
Go north	1	0	0	0

These values are the passed to the connection table, then to the conditions .See the diagram in Appendix A.

Note that all the letters in the vocabulary entries are significant - there is no truncation to only four or five letters. This RIVER and RIVET, TROUT and TROUSERS, and other like these are differentiated.

#### APPENDIX A

=====

#### Definitions and Conditions for "ADVINMAN"

##### i. NOUNS

1 TORCH        5 SERPENT  
1 LAMP         6 DOOR  
2 RAT          255 IT  
3 KEY  
4 GOLD  
4 BAR  
4 TREASURE  
5 SNAKE

##### ii. VERBS

1 NORTH        9 LOOK        15 LOAD  
2 SOUTH       10 LIST       15 RESTORE  
3 EAST         10 INVENTORY 16 EXAMINE  
4 WEST         11 QUIT       17 LIGHT  
5 UP           12 TEXT       17 ON  
6 DOWN         12 WORDS      18 EXTINGUISH  
7 GET          13 PICTURES   18 OFF  
7 TAKE         13 GRAPHICS   19 UNLOCK  
8 DROP         14 SAVE       20 SCORE  
1 N    2 S    3 E    4 W    21 EAT  
5 U    6 D

##### iii. ROOM DESCRIPTION

Room No.	Description (as it would appear)	Connections
1 .	You are above the ground. There is a cave entrance to the east.	(press ENTER)
2 .	You are in a large cavern. Passages lead east, west and south.	E 3

- 3 . You are in a cave. A snake is asleep in a comer and exits lead east and west. W 3
- 4 . You are by a small lake. The only exit is west. ENTER)
- 5 . You are outside a castle. A tunnel leads to the north and a large door can be seen to the east. (press ENTER)
- 6 . You are in the castle strongroom. A door stands open to the west. W 5

#### iv. OBJECTS

No.	Description	Starts in room	Weight
1.	a lamp	1	10
2.	a dead rat	5	20
3.	a key	4	1
4.	a gold bar	6	100
5.	a lit lamp	0	100

#### v. MESSAGES

```

240 What now ?.
241 You can't
242 Pardon?
243 Press a key for another game....
244 Are you sure: ( Y or N ) ?
245 You've already got that.
246 You haven't got that.
247 You can't see that.
248 You're carrying too much to pick that up
249 Your score was
250 and you took
251 It's dark. You can't see a thing.
252 I can't find that anywhere.
253 You can also see.
254 Okay
255 turns

```

#### i. Conditions - Local Don't type in the comments !

-----

##### Room 1

```

IF ( VERB 3 ) RESE 1 GOTO 2 WAIT END
    If you typed "EAST", reset the dark/light marker, goto room 2 and wait for a
    new command.

```

##### Room 2

```

IF { VERB 4 ) SET 1 GOTO 1 WAIT END
    If you typed "WEST", set the dark/light marker, goto room 1 and wait for a
    new command.
IF ( VERB 2 ) SET 1 GOTO 5 WAIT END
    And a similar condition for going South too.

```

##### Room 3

```

IF ( VERB 7 AND NOUN 5 ) MESS 14 HOLD 200 EXIT END
    If you typed "GET SNAKE" then print message fourteen to say how it reacts,
    freeze for four seconds and end the game.
IF ( VERB 3 ) SET 1 GOTO 4 WAIT END
    Again another move-from-dark-room-to-light-room condition, this time east.

```

##### Room 4

```

IF < VERB 4 ) RESE 1 DOTO 3 WAIT END
    I'll ley you work this one out (hint - look at rooms 1, 2, 3)
IF ( VERB 7 AND NOUN 3 ) CTR 0+20 CSET 0 END
    If you typed "GET KEY" then add twenty to counter 0 (the score)

```

IF ( VERB 8 AND NOUN 3 ) CTR 0-20 CSET 0 END  
If you typed "DROP KEY" then subtract 20 from ctr 0 (the score)

Room 5

IF ( VERB 3 AND SET? 3 ) GOTO 6 WAIT END  
If you typed "EAST" and marker 3 is set, (i.e. if the door has been opened), goto room 6 and wait for a new command.

IF ( VERB 3 ) MESS 7 WAIT END  
If you typed "EAST" then print message 7 and wait for a new command. Note that the door cannot be opened since if it was, the last line would have worked and we would be waiting for a new command by now.

IF ( VERB 19 AND NOUN 6 AND CARR 3 ) SET 3 MESS 10 WAIT END  
If you typed "UNLOCK DOOR" then set marker 3, (mark the door as open), print message 10 and wait for a new command.

IF ( VERB 1 ) RESE 1 GOTO 2 WAIT END  
It is another light-to-dark movement, this time north to room 2

## ii. Conditions - Low Priority

-----

Those marked are included in the Quickstart file.

IF ( VERB 20 ) MESS 249 PRIN CTR 0 MESS 250 PRIN TURN MESS 255 WAIT END  
If you typed "SCORE", print message 249, your score, message 250, the number of turns you has and message 255. Then wait for a new command.

IF ( NO1 = D AND VERB 7 ) MESS 19 WAIT END  
If you typed "GET" by itself or with an unrecognised word, print message 19 "I'm sorry, but I don't know what one of those is... "

IF ( NO1 = 0 AND VERB 8 ) MESS 19 WAIT END  
And similarly for "DROP".

IF ( NO1 = 0 AND VERB 16 ) MESS 18 WAIT END  
And again for "EXAMINE".

IF ( VERB 7 AND NOUN 1 AND HERE 5 } GET 5 OKAY END  
If you typed "GET LAMP'1 and there is a lit lamp here, then get it, print "OKAY" and wait for a new command.

IF ( NO1 < 5 AND VERB 7 ) GET NO1 OKAY END.  
If you typed "GET" and a noun with a number less than 5, then get the object with that noun's number. This only works because the objects and the nouns which refer to them have the same number. {A very useful trick!)

IF ( VERB 8 AND NOUN 1 AND CARR 5 ) DROP S OKAY END  
If you typed "DROP LAMP" and you've got a lit lamp, then drop it, print "Okay" and wait for a new command.

IF ( NO1 < 5 AND VERB 8 ) DROP NO1 OKAY END  
If you typed "DROP" and a noun whose number is less than 5, then drop the object with that noun number.

IF ( VERB 16 AND NOUN 1 AND AVAI 5 ) MESS 5 WAIT END  
If you typed "EXAMINE LAMP" and you have a lit lamp available then print out its more detailed description and wait for a new command.

IF ( NO1 < 5 AND VERB 16 AND AVAI NO1 ) MESS NO1 WAIT END  
If you typed "EXAMINE" and a noun whose number is less than 5, then print out the message with that noun number, it being the more detailed description of that abject, then wait for a new command.

IF ( VERB 11 ) QUIT END  
If you typed "QUIT" then quit.

IF { VERB 9 ) LOOK WAIT END  
If you typed "LOOK" then describe this room and wait for a new command.

IF { VERB 21 AND NOUN 2 AND CARR 2 ) DROP 2 2 TO 0 MESS 17 WAIT END  
If you typed "EAT RAT" and you are carrying it, drop it, move to room 0, thus destroying it, print message 17 ("yum, yum") and wait for a new command.

IF ( VERB 10 ) MESS 239 LIST WITH END  
If you typed "INVENTORY" then list the objects with you and wait for a new command.

IF ( VERB 13 ) PICT OKAY END

If you typed "PICTURES" then turn them on, print "OKAY" and wait for a new command.

```
IF ( VERB 12 ) TEXT OKAY END
    And similarly for "TEXT".
IF ( VERB 17 AND NOUN 1 AND AVAI 1 ) 1 SWAP 5 CTR 0+20 CSET 0 MESS 15
SET 2 WAIT END
    type this all on one line. If you typed "LIGHT LAMP" and you have an unlit
    lamp, then exchange the lit lamp for the unlit one, tell the user that he
    has lit the lamp, increase the score by 20 and await a new command.
IF ( VERB 18 AND NOUN 1 AND AVAI 5 ) 1 SWAP 5 CTR 0-20 CSET 0 MESS 16 WAIT END
    And similarly for "LAMP OFF" ("EXTINGUISH LAMP").
IF ( VERB 14 ) SAVE OKAY END
    If you typed "SAVE" then save the game position to tape or disc .
IF( VERB 15 ) LOAD LOOK WAIT END
    If you typed "LOAD" then load in a previously saved game position.
```

### iii. Conditions - High Priority

```
-----
IF ( RES ? 6 ) SET 6 STRE 111 3 CSET 1 END
    If marker 6 is reset {if this is the first move) then set marker 6 to say
    that it isn't the first move any more, set the strength and counter one to
    hold value three.
IF ( RES ? 1 AND RES ? 2 ) DECR 1 END
    If you are in total darkness, then decrease counter number one.
IF ( 1 EQU ? 1 AND RES ? 1 AND RES ? 1 ) MESS 20 END
    If counter one has reached value 1 then print that you can hear footsteps.
IF ( 0 EQU ? 1 ) MESS 21 EXIT END
    If counter 1 has reached zero then tell the player he has been "got" by the
    spider, and end the game.
IF ( AT 3 AND RES ? 4 AND CARR 2 ) SET 4 MESS 13 DROP 2 2 TO 0 WAIT END
    If you are in room three with the rat and the snake hasn't been fed yet,
    mark the snake as fed, print the message to say so, and move the rat to room
    0, destroying it.
IF ( AT 3 AND RES ? 4 ) MESS 3 EXIT END
    If you are in room three without the rat and the snake hasn't been fed yet,
    then say that it kills you and exit from the game.
IF ( AT 1 AND CARR 4 ) MESS 9 EXIT END
    If you are back at the start carrying the gold then you win !
```

### GET AND DROP

-----

Here is another useful feature! To save putting a Get and Drop condition in for each and every object, here are two very useful condition lines that will enable you to get and drop any object, (up to No.10 in our example), at any location. To do this simply ensure the Noun numbers match the Object numbers. ie.

```
NOUN 1 = Hat OBJECT 1 = a hat
```

and enter Low Priority Condition lines :

```
IF ( NO1 < 10 AND VERB 7 ) GET NO1 OKAY END
IF ( NO1 <10 AND VERB 8 ) DROP NO1 OKAY END
```

APPENDIX B  
 =====  
 Tables and Charts

THE QUICKSTART DATA FILE  
 -----

The "Quickstart" data file contains all the system messages, many useful verbs, and several of the common low priority conditions associated with them. This is to allow you to get straight in to writing the adventure that you want to write, without having to worry about things that are included in all adventures. The full contents are :

<u>VERBS</u>	<u>NOUNS</u>	
6 D	255	IT
6 DOWN		
8 DROP	<u>MESSAGES</u>	
3 E		
3 EAST	239	You are carrying
16 EXAMINE	240	What now ?.. ..
7 GET	241	You can't.
13 GRAPHICS	242	Pardon ?
10 INVENTORY	243	Press a key for another game...
9 L	244	Are you sure ? (Y/N)...
10 LIST	245	You've already got that.
15 LOAD	246	You haven't got that.
9 LOOK	247	You can't see that.
1 N	248	You're carrying too much to ; up.
1 NORTH		
13 PICTURES	249	Your score was
11 QUIT	250	and you took
15 RESTORE	251	It is dark. You can't see.
2 S	252	I can't find that anywhere.
14 SAVE	253	You can also see
2 SOUTH	2 54	Okay
7 TAKE	255	turns
12 EXIT		
5 U		
5 UP		
4 w		
4 WEST		
12 WORDS		

LOW PRIORITY CONDITIONS  
 -----

```

IF { VERB 9 ) 1,0 OK WAIT END
  If you typed "LOOK", redescribe the room you're in and wait for a new command.
IF ( VERB 10 } MESS 239 LIST WITH WAIT END
  If you typed "INVENTORY" then print "You are carrying" and list all the objects
  that are with you.
IF ( VERB 11 ) QUIT OKAY END
  If you typed "QUIT" then ask the adventurer if he is sure, and if he responds Y
  (for YES) then quit, otherwise print OKAY and wait for a new command.
IF ( VERB 12 ) TEXT OKAY END
  If you typed "TEXT" then turn the pictures off
IF ( VERB 13 } PICT OKAY END
  If you typed "PICTURES" then turn the pictures on.
IF { VERB 14 ) SAVE OKAY END
  If you typed "SAVE" then save the game position to tape or disc.
IF ( VERB 15 ) LOAD LOOK WAIT END
  If you typed "LOAD" then load a previously saved game position from tape or disc,
  
```

then describe the room you are in.

i. ACTUAL GRAPHICS COLOURS

- 
- 0 BLACK
  - 1 BLUE
  - 2 RED
  - 3 MAGENTA
  - 4 GREEN
  - 5 CYAN
  - 6 YELLOW
  - 7 WHITE
  - 8 TRANSPARENT - ie. If paper is set to 8, you can draw over all colours of paper leaving the paper colour unchanged, (Ink, Flash or Bright).
  - 9 CONTRAST - ie. If ink is set to 9, the ink used will be either White or Black, whichever is the greatest contrast to the current paper colour. (Or Paper).

ii. SUBSCRIPTS USED WITH CONDITION WORDS

- 
- a - adverb number
  - c - counter number
  - m - marker number
  - ms -.message number
  - o -.object number
  - n -.noun number
  - r -.room number
  - v -.verb number
  - x,y -.any numbers

iii. WORDS USED IN CONDITIONS

-----

<u>Words</u>	<u>Markers</u>	<u>Counters</u>	<u>Rooms</u>	<u>Commands</u>
VERB v	SET m	CSET c	GOTO r	OKAY
NOUN n	RESE m	INCR c	CONN r	WAIT
ADVE a	SET? m	DECR c	ROOM	EXIT
NO 1	RES? m	CTR c	AT r	QUIT
NO 2		X EQU? c	DESC r	
VBNO		TURN	LOOK	

<u>Objects</u>	<u>Decisions</u>	<u>Tape</u>	<u>and</u>	<u>Disc</u>
GET o	OBJ o	IF		SAVE
DROP o	LIST o	END		LOAD
o SWAP o	WEIG o			
HERE o	STRE o			
CARR o	o TO r			
AVAI o	BRIN o			
o IN r	FIND o			

Other

-----

MESS ms	x = y
x + y	WITH
x - y	PICT
condition AND condition	TEXT
condition OR condition	RAND x
condition XOR condition	PRIN X
NOT condition	HOLD X
x > y	LF

iv. COUNTERS AND MARKERS USED BY THE SYSTEM

-----

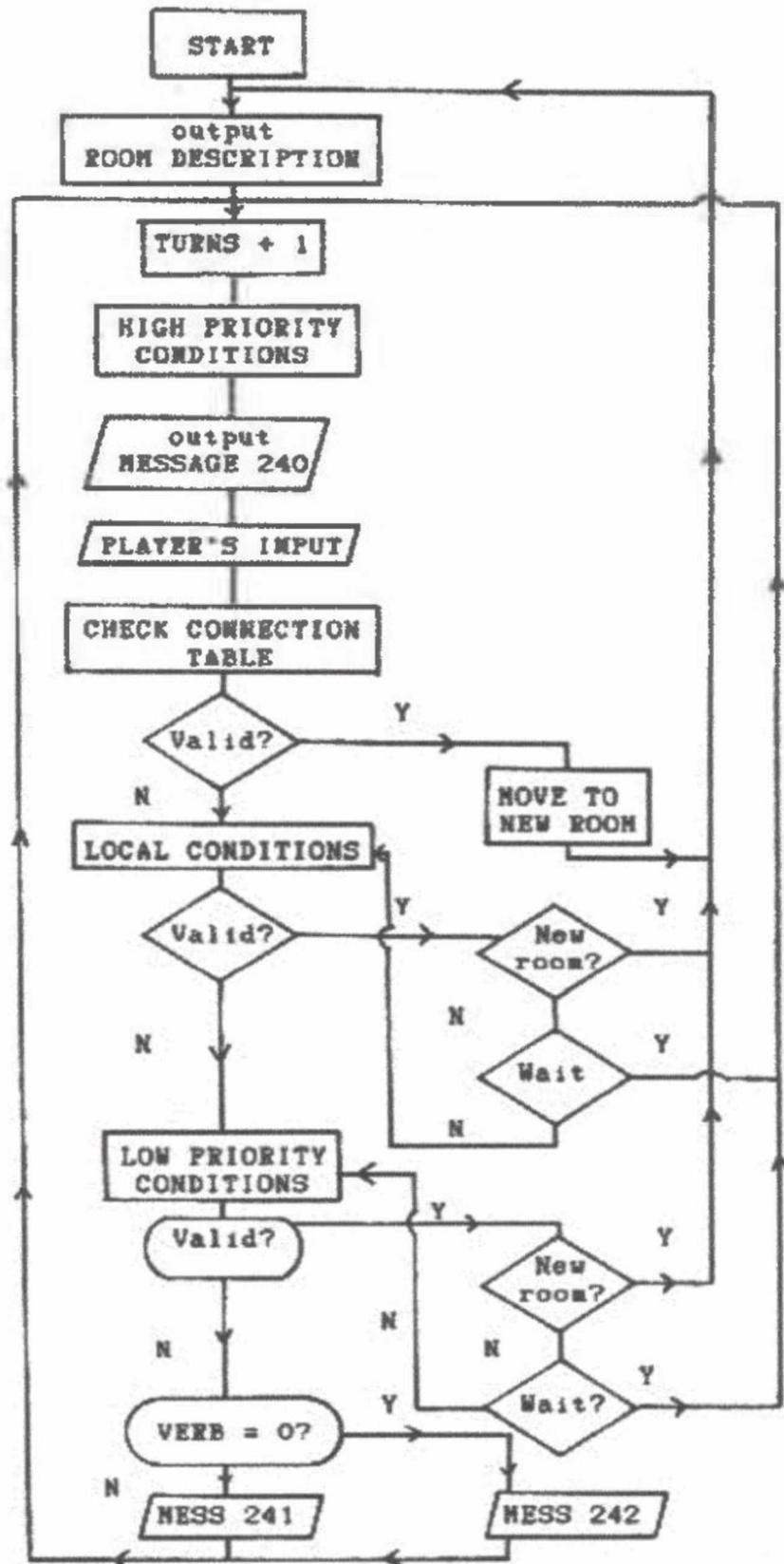
Counter	0 - Holds the score
	1 x 126 } Count the number of turns since the beginning
	256 x 127 } of the game.
Marker	0- If set means that a room has been described since last reset.
	1 - If set, you are in a light room.
	2 - If set, you have a source of light.
	3 - If set, disables the scoring mechanism.

v. RANGES OF NUMBERS

-----

	No. allowed	No. of characters
Rooms	1..9999	255
Objects	1..255	255
Messages	1..255	255
Verbs	1..255	35
Nouns	1..255	35
Adverbs	1..255	35
Graphics	1..9999	
Markers	0.. 255	
Counters	0..127	
[Which score]	0..255	

240 : "What now?"



241 : "You can't"  
242 : "Pardon?"

The internal command QUIT and EXIT return the player to the First room with the program initialized. There is no end as such.