Multi Platform Arcade Game Designer

Language Reference

Version 0.6

Jonathan Cauldwell

April 2018

# Data Definition

**DEFINEWINDOW**

Defines the size of the play area in character cells. DEFINEWINDOW must be followed by 4 numbers: the distance from the edge, the distance from the left edge, the height and the width.

*Example:*

DEFINEWINDOW 1 1 22 30

**DEFINEBLOCK**

Used to define the building blocks that make up each screen. DEFINEBLOCK must be followed by the block type and then a series of numbers which make up the appearance of the block. Permissible block types are EMPTYBLOCK, PLATFORMBLOCK, WALLBLOCK, LADDERBLOCK, FODDERBLOCK, DEADLYBLOCK and CUSTOMBLOCK.

The format of the numbers that determine the physical representation of the block varies depending upon the target machine.

**Spectrum**

These image is made up of 8 bitmap rows followed by an attribute byte.

*Example:*

DEFINEBLOCK EMPTYBLOCK 0 0 0 0 0 0 0 0 71

**Timex/Next**

These are ordered pixel bitmap row 0, attribute row 0, pixel bitmap row 1, attribute row 1 etc.

*Example:*

DEFINEBLOCK EMPTYBLOCK 0 71 0 71 0 71 0 71 0 71 0 71 0 71 0 71

**Amstrad CPC**

These are ordered top left pixel pair 1 and 2, top pixel pair 3 and 4, top right pixel 5, then the same for each row down. This means with three bytes on each row and eight rows in total, the image is made up of 24 bytes. As blocks are 5x8 pixels, half the bits are unused every third byte.

*Example:*

DEFINEBLOCK EMPTYBLOCK 0 0 0 0 0 0 0 0 0 0 0 0

0 0 0 0 0 0 0 0 0 0 0 0

**DEFINEFONT**

Defines the character set to be used in a game. This is 768-byte 96 character, 8x8 pixel font. On the CPC only the leftmost 5 pixels are displayed due to the screen resolution used.

**DEFINESPRITE**

This defines the sprite graphics. The first number following DEFINESPRITE specifies the number of frames. This must then be followed by 32 bytes of graphic data (Spectrum and Next/Timex) or 80 (Amstrad CPC). For the Spectrum, data order is bitmap row 0 left, bitmap row 0 right, bitmap row 1 left, bitmap row 1 right etc. For the Amstrad data is organised topo-left to top-right, then second row left to second row right and so on.

*Example (Spectrum):*

DEFINESPRITE 1 ; 1 frame

7 224 31 248 63 252 127 254

127 254 255 255 255 255 255 255

255 255 255 255 255 255 127 254

127 254 63 252 31 248 7 224

**DEFINEOBJECT**

**Spectrum**

Each DEFINEOBJECT requires 36 parameters. The first byte is the object’s attribute. The next byte defines the screen upon which the object appears and the third and fourth define the y and x coordinates respectively. The following 32 (Spectrum and Next/Timex) or 80 (CPC) bytes define the graphic image of the object and are ordered in an identical fashion to the bytes which define sprites. So Spectrum data is ordered bitmap top left, bitmap top right, bitmap second row left, bitmap row second right down to the bottom row left and bottom row right. CPC data is organised top-left to bottom right in the same way with 5 bytes for each line making up the 10 pixel image.

*Example (Spectrum):*

; Barrel, bright red, room 5, 32 pixels down, 128 across

DEFINEOBJECT 66 5 32 128

31 240 42 168 86 212 64 4

128 2 182 218 182 218 182 218

182 218 182 218 182 218 128 2

64 4 86 212 42 168 31 240

**Timex/Next**

Each DEFINEOBJECT requires 35 parameters. The first defines the screen upon which the object appears and the second and third define the y and x coordinates respectively. The following 32 bytes define the graphic image of the object and are ordered in an identical fashion to the 32 bytes which define sprites. So data is ordered bitmap top left, bitmap top right, bitmap second row left, bitmap row second right down to the bottom row left and bottom row right.

*Example:*

; Barrel, room 5, 32 pixels down, 128 across

DEFINEOBJECT 5 32 128

31 240 42 168 86 212 64 4

128 2 182 218 182 218 182 218

182 218 182 218 182 218 128 2

64 4 86 212 42 168 31 240

**DEFINESCREEN**

This is used to set up the layouts of all the rooms, levels, screens or whatever you wish to call them and the parameters required will depend on the size of the play area defined in DEFINEWINDOW. Each screen must consist of one byte for every row and column, so if your screen size is 30 characters across by 16 down you should arrange the numbers following each DEFINESCREEN declaration as 16 rows of 30 numbers. It is recommended that spaces are used to separate the bytes and align the columns as this aids clarity. AGD will automatically compress each screen to reduce its size.

**SPRITEPOSITION**

Any sprites to be placed on a screen should follow on from DEFINESCREEN. SPRITEPOSITION requires 4 parameters for the sprite type (eg PLAYER). sprite image number, starting Y coordinate and starting X coordinate respectively.

It is a good idea to have a SPRITEPOSITION for the player sprite on each screen so that AGD will know where to respawn the player should he lose a life on any particular screen. The CPC compiler uses Spectrum coordinates in order to make conversions simpler.

*Example:*

DEFINESCREEN ; 24 chars across by 16 down

1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1

1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1

1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1

1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1

1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1

1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1

1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1

1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1

1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1

1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1

1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1

1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1

1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1

1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1

1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1

SPRITEPOSITION 0 0 64 80 ; Player, image 0 at 64,80

SPRITEPOSITION 1 2 104 136 ; type 1, image 2 at 104,136

**DEFINEPALETTE**

AGD uses ULAplus on standard Spectrums and ULANext on the Timex/Next machines. If no palette is defined AGD will use its own default palette. A separate PALETTE command can be used in the program code to change individual palette entries.

**Spectrum**

AGD’s ULAplus palette consists of 64 different RGB values, 8 INK and 8 PAPER values, repeated for all 4 combinations of BRIGHT and FLASH. The entire palette for a game can be defined with a single DEFINEPALETTE declaration followed by a list of 64 RGB values for the palette.

**Next**

AGD Next’s palette consists of 32 different RGB values, 16 for INK and 16 for PAPER. The entire palette for a game can be defined with a single DEFINEPALETTE declaration followed by a list of 32 RGB values for the palette.

**Amstrad CPC**

The CPC’s palette consists of 16 different values ranging from 0 to 26 to represent the 27 available colours. The entire palette for a game can be defined with a single DEFINEPALETTE declaration followed by a list of 16 colours.

**DEFINEMESSAGES**

Messages can be defined as a series of strings which can be displayed individually using the MESSAGE n command. Messages must be defined before the events code or an error will result.

*Example:*

DEFINEMESSAGES

; Manic Miner screen titles

“CENTRAL CAVERN”

“THE COLD STORE”

“THE MENAGERIE”

**MAP**

Map describes the layout of screens that make up the world the sprites inhabit. It requires 2 parameters to start with: the WIDTH and STARTSCREEN. Width tells the compiler how many screens are arranged horizontally, STARTSCREEN tells AGD the screen number on which to start. There is no need to specify a starting row and column, the compiler will automatically locate the STARTSCREEN in the map, wherever it is located. Grid positions which contain no screen should be filled with the number 255.

*Example:*

; 4 screens wide, 4 down. Start 2nd row, 3rd Column.

MAP WIDTH 4 STARTSCREEN 0

001 002 004 005

003 006 000 255

255 007 008 255

255 255 009 010

# Constants

**LEFTEDGE, RIGHTEDGE, TOPEDGE, BOTTOMEDGE**

Used to tell when a sprite reaches the edge of the screen.

*Example:*

IF X < SCREENRIGHT

SPRITERIGHT

ENDIF

**LEFT, RIGHT, UP, DOWN**

Used by STAR command to determine direction of travel. LEFT and RIGHT can also be used as arguments for the LASER command. Also used by the KEY function to detect directional keypresses.

*Example:*

LASER LEFT

STAR RIGHT

**FIRE, FIRE2, FIRE3, OPTION1, OPTION2, OPTION3, OPTION4**

These are used by the KEY function to detect non-directional keypresses. The FIREn constants are ideal for use as in-game actions while the OPTIONn constants default to the keys 1 to 4 and are better used as menu options for setting difficulty levels or controls.

*Example:*

IF KEY FIRE2

JUMP

ENDIF

**BULLET**

Used in conjunction with IF COLLISION to detect when a sprite has been hit by a laser bullet.

*Example:*

IF COLLISION BULLET

KILL

ENDIF

**EMPTYBLOCK, PLATFORMBLOCK, WALLBLOCK, LADDERBLOCK, FODDERBLOCK, DEADLYBLOCK, CUSTOMBLOCK**

Block types returned by the GETBLOCK command.

*Example:*

GETBLOCK

IF BLOCK = EMPTYBLOCK

**KEYBOARD, KEMPSTON, SINCLAIR**

Player controls. Setting the CONTROL variable to one of these will select keyboard, Kempston or Sinclair joystick controls.

*Example:*

LET CONTROL = 99

WHILE CONTROL = 99

IF KEY OPTION1

LET CONTROL = KEYBOARD

ENDIF

IF KEY OPTION2

LET CONTROL = KEMPSTON

ENDIF

IF KEY OPTION3

LET CONTROL = SINCLAIR

ENDIF

ENDWHILE

**Variables**

All variables hold 8-bit values, so values of 0-255 are possible.

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

General-purpose variables. These can be used in any way.

**AIRBORNE**

Sprite parameter. Determines when a sprite is jumping or falling. This is set to a non-zero value when a sprite is in the air.

**BLOCK**

Variable. The last block number detected by GETBLOCK.

**CLOCK**

Read-only variable. The current frame counter, in 50ths of a second. Useful for timings.

**COLUMN, LINE**

Display variables. These determine the position at which the score or message will be displayed.

**CONTROL**

Player controls. On the Spectrum and Next/Timex, this should be set to KEYBOARD, KEMPSTON or SINCLAIR. On the Amstrad CPC this should be set to KEYBOARD, JOYSTICK0 or JOYSTICK1.

**FRAME, IMAGE**

Sprite parameters. These are the sprite and frame numbers shown in the sprite editor. You can change the sprite according to whichever direction the player is facing, or perhaps you might want to give the player a choice of vehicles to control. Setting a frame number beyond the limit of the sprite will result in a different sprite image being displayed, so use FRAME with caution. When setting the image, it is a good idea to set the frame to 0 at the same time, unless you have a very good reason for not doing so.

**JUMPSPEED**

Sprite parameter. Holds the speed and direction of a sprite when it is jumping or falling.

**LIVES**

Global variable. This holds the number of player lives remaining.

**RND**

Variable. The last random number generated by GETRANDOM.

**SCREEN**

Global variable. Contains the current screen number.

**TYPE**

Sprite parameter. This is the type of sprite being processed. Best used in conjunction with the IMAGE parameter, setting this parameter will completely change the sprite's behaviour - handy for turning a nasty into a bonus, or making a sprite stop and explode before killing it. There is no reason why you couldn't change a sprite to type zero and put it under the player's control, or change the player's sprite type to something else with a slightly different set of controls. So long as the new sprite type has appropriate code set up in the relevant event, there's no limit to what you could do. If you haven't set up any code for the sprite type your sprite will just sit there - which may be okay if that's what you want.

**DIRECTION, SETTINGA, SETTINGB**

Sprite parameters. You can use these as you see fit, perhaps to indicate the direction in which a particular sprite is moving, or the particular phase it is going through. Invaluable for any form of enemy AI.

**OBJ**

Variable. The last object number detected by DETECTOBJ.

**X, Y**

Sprite parameters. These are the coordinates of the current sprite.

# Functions

**CANGODOWN**

Condition is true if the current sprite can move down.

*Example:*

IF KEY DOWN

IF CANGODOWN

SPRITEDOWN

ENDIF

ENDIF

**CANGOLEFT**

Condition is true if the current sprite can move left.

*Example:*

IF KEY LEFT

IF CANGOLEFT

SPRITELEFT

ENDIF

ENDIF

**CANGORIGHT**

Condition is true if the current sprite can move right.

*Example:*

IF KEY RIGHT

IF CANGORIGHT

SPRITERIGHT

ENDIF

ENDIF

**CANGOUP**

Condition is true if the current sprite can move up.

*Example:*

IF KEY UP

IF CANGOUP

SPRITEUP

ENDIF

ENDIF

**CHR**

Displays the single ASCII character specified.

*Example:*

CLS

; display the ASCII character set

LET A = 32

REPEAT 96

CHR A

ADD 1 TO A

ENDIF

**COLLISION**

Requires one numeric argument to specify the sprite type. Condition is true if the current sprite is in collision with another sprite of the type specified.

*Example:*

IF COLLISION 0

KILL

ENDIF

**CUSTOM**

Function. Similar to DEADLY, true if the current sprite is in contact with a CUSTOM block.

*Example:*

IF CUSTOM

NEXTLEVEL

EXIT

ENDIF

**DATA**

Precedes a list of numbers which can be accessed sequentially via a READ statement.

*Example:*

READ Z ; read co-ordinate displacement

LET X = 120

ADD Z TO X

SPRITEDOWN

IF Y > BOTTOMEDGE

ENDGAME

ENDIF

; data for our sine wave

DATA 8 11 13 14 15 16 16 15 14 13 11 8 5 3 2 1 0 0 1 2 3 5

**DEADLY**

Function. The condition is true if the current sprite is in contact with a deadly block.

*Example:*

IF DEADLY

KILL

ENDIF

**GOT**

Expects a single argument. The expression is true if the specified object is owned by the player. The argument can be a variable if required, so IF GOT 1 is valid, as is IF GOT A. You could even use a sprite parameter as an argument if you wish.

*Example:*

IF KEY FIRE ; fire pressed

IF GOT 0 ; got the gun?

LASER DIRECTION ; fire in the direction faced

ENDIF

ENDIF

**KEY**

Expects a single numeric argument in the range 0 to 6 and condition is true if the key is pressed. Usually used in conjunction with the UP, DOWN, LEFT, RIGHT, FIRE, FIRE2 or FIRE3 constants.

*Example:*

IF KEY LEFT

IF CANGOLEFT

SPRITELEFT

ENDIF

ENDIF

**LADDERABOVE**

Condition is true if the current sprite can ascend a ladder.

*Example:*

IF KEY UP

IF LADDERABOVE

SPRITEUP

ENDIF

ENDIF

**LADDERBELOW**

Condition is true if the current sprite can go down a ladder.

*Example:*

IF KEY DOWN

IF LADDERBELOW

SPRITEDOWN

ENDIF

ENDIF

# Commands

**ADD**

Adds a specified number to a sprite parameter or variable.

*Example:*

ADD 1 TO A

**ADDBONUS**

Adds the bonus to the player’s score.

*Example:*

SHOWSCORE

DELAY 25

PRINT “ + ”

DELAY 25

SHOWBONUS

DELAY 25

PRINT “ = ”

DELAY 25

ADDBONUS

ZEROBONUS

SHOWSCORE

**ANIMATE**

Animates the present sprite, automatically cycling through the frames in ascending order. After the last frame the animation returns to the first. ANIMATE takes an optional parameter to specify the animation speed. Available parameters are FAST, MEDIUM, SLOW and VERYSLOW. Where no speed is specified FAST is the default.

*Example:*

IF KEY RIGHT

ANIMATE MEDIUM

IF CANGORIGHT

SPRITERIGHT

ENDIF

ENDIF

**ANIMBACK**

As ANIMATE, but cycles through frames in descending order. ANIMBACK takes an optional parameter specifying the animation speed. Available parameters are FAST, MEDIUM, SLOW and VERYSLOW. Where no speed is specified FAST is the default.

*Example:*

IF KEY LEFT

ANIMBACK MEDIUM

IF CANGOLEFT

SPRITELEFT

ENDIF

ENDIF

**AT**

Sets the line and column position for commands which display text or numbers on the screen. Requires parameters for LINE and COLUMN respectively.

*Example:*

AT 0 0

PRINT “0”

AT 0 31

PRINT “1”

AT 23 31

PRINT “2”

AT 0 31

PRINT “3”

**BEEP**

Starts a beeper sound effect which starts low and increases in pitch. Expects a single parameter for the sound duration.

*Example:*

BEEP 50

**BONUS**

Expects to be followed by a number to add to the bonus, eg BONUS 100 will add 100 points to the player's bonus total. Values in the range 0 to 65535 are valid. Bonuses can be added to the player’s score at a later point or kept as a separate total, perhaps as a second player’s score.

*Example:*

BONUS 100

LET LINE = 1

LET COLUMN = 1

SHOWBONUS

**BORDER**

Expects an argument from 0 to 7 inclusive. Sets the border colour.

*Example:*

BORDER 0

**CLS**

Clears the screen and changes its colour to the current COLOUR setting.

*Example:*

COLOUR 7

CLS

PRINT “A NEW SCREEN”

**CLUT**

Expects a single argument. Sets the colour look-up table for the permanent display attributes. There are 4 colour look-up tables of 8 foreground colours and 8 background colours so the valid range is 0 - 3. Can be used prior to displaying text or clearing the screen.

*Example:*

PAPER 0

CLS

LET C = 0

REPEAT 4

CLUT C

INK 4

PRINT “\*”

INK 5

PRINT “\*”

INK 6

PRINT “\*”

INK 7

PRINT “\*”

ADD 1 TO C

ENDREPEAT

**CLW**

Clears the play area window and changes its colour to the current COLOUR setting.

*Example:*

COLOUR 40

CLW

PRINT “PLAY AREA WINDOW”

**COLOUR**

Expects a single argument. Sets the permanent display attributes. Format is 64 \* CLUT + 8 \* PAPER + INK on the ZX Vega and ULAplus Spectrums or 128 \* FLASH + 64 \* BRIGHT + 8 \* PAPER + INK on a standard machine. Can be used prior to displaying text or clearing the screen.

*Example:*

COLOUR 7

PRINT “WHITE ON BLACK”

COLOUR 14

PRINT “YELLOW ON BLUE”

COLOUR 66

PRINT “BRIGHT RED ON BLACK”

**CRASH**

Starts a white noise sound effect. Expects a single parameter for the sound duration.

*Example:*

CRASH 30

**DECAYPARTICLE**

Command. Subtracts 1 from the current particle’s frame timer. When the frame timer reaches zero the particle is removed. If DECAYPARTICLE is omitted, user particles will only be removed when they are moved out of the play area.

*Example:*

DEFINEPARTICLE

PARTICLE DOWN

GETRANDOM 2

IF RND = 0

PARTICLELEFT

ELSE

PARTICLERIGHT

ENDIF

DECAYPARTICLE ; remove after a while

**DEFINEKEY**

Waits for a keypress and then redefines the specified key to be the one pressed by the user.

*Example:*

AT 1 4

PRINT “PRESS KEY FOR”

AT 4 4

PRINT “UP”

DEFINEKEY UP

DELAY 25

AT 6 4

PRINT “DOWN”

DEFINEKEY DOWN

DELAY 25

AT 8 4

PRINT “LEFT”

DEFINEKEY LEFT

DELAY 25

AT 10 4

PRINT “RIGHT”

DEFINEKEY RIGHT

**DEFINEPARTICLE**

Used to define the code which describes the behaviour of the user particle. Can be placed in any event, although any code following DEFINEPARTICLE is not executed as part of the event in which it is placed. The following code will then make up the subroutine which moves the user particles.

*Example:*

DEFINEPARTICLE

PARTICLEUP

GETRANDOM 3

IF RND = 0

PARTICLELEFT

ELSE

IF RND = 1

PARTICLERIGHT

ENDIF

ENDIF

**DELAY**

Expects a single argument. Pauses for the duration specified. A delay of 50 represents a pause of one second.

*Example:*

DELAY 150 ; Wait for three seconds.

**DETECTOBJ**

Detects objects touched by the current sprite, and places the result in the OBJ variable. If the sprite is touching more than one object, only the object with the lowest number is detected. If no object is detected, OBJ will be set to 255.

*Example:*

DETECTOBJ ; is sprite touching an object?

IF OBJ <> 255 ; did we detect an object?

GET OBJ ; automatically pick it up.

ENDIF

**DISPLAY**

Displays the numeric value in the variable or parameter that follows.

*Example:*

SUBTRACT 1 FROM LIVES

LET LINE = 0

LET COLUMN = 22

DISPLAY LIVES

**DIVIDE**

Divides a sprite parameter or variable by the specified amount.

*Example:*

DIVIDE Y BY 8

**ENDGAME**

Ends the game in victory. This performs the Completed Game event. Completing the last screen in a game with sequential levels will do the same. The only other way in which a game can finish is if the player loses all his lives, but that does not perform the Completed Game event.

*Example:*

IF GOT 3 ; have we collected the chalice?

ENDGAME ; yes, player wins!

ENDIF

**ENDREPEAT**

This marks the end of the block of code to be repeated.

*Example:*

CLS

REPEAT 32

PRINT “\*”

ENDREPEAT

**ENDSPRITE**

Used after OTHER and SPAWNED commands, this reverts to the original sprite.

*Example:*

SPAWN 1 4

SPAWNED

LET X = RIGHTEDGE

ENDSPRITE

**EXIT**

All processing of the current event is terminated.

*Example:*

IF X > RIGHTEDGE

SCREENRIGHT

LET X = LEFTEDGE

EXIT

ENDIF

**EXPLODE**

Requires a numeric argument to specify the number of particles. Creates an explosion at the current sprite position. Used with REMOVE, EXPLODE removes the need for you to draw an explosion animation for your sprites. If you forget to follow EXPLODE with a parameter, the next command will not be compiled properly. If that command is REMOVE your sprite won't be removed and you'll wonder why it explodes but remains in play.

*Example:*

IF COLLISION BULLET ; has sprite been shot?

REMOVE ; remove this sprite

EXPLODE 24 ; two dozen pieces of shrapnel

SCORE 50 ; score some points

LET LINE = 1

LET COLUMN = 1

SHOWSCORE

SOUND 3 ; play a sound

ENDIF

**FALL**

Provided the sprite is not already falling or jumping, this checks to see if the sprite is standing on top of solid ground. If not, it will start to fall. FALL is useful if a sprite type is subject to gravity, eg platform games.

*Example:*

IF KEY UP

JUMP

ENDIF

FALL

**GET**

Command. Expects a parameter specifying the object to get. Places the specified object in the player's inventory, regardless of where it is. The object could be on the current screen, another screen, or just missing. It is up to you to decide when the player can get a particular object. You may want to use DETECTOBJECT to determine when a sprite is touching an object first.

*Example:*

IF KEY FIRE

DETECTOBJ ; anything to pick up?

IF OBJ <> 255

GET OBJ ; get the item

ENDIF

ENDIF

**GETBLOCK**

Requires 2 arguments for X and Y pixel coordinates. Finds the block type at the specified point on screen and places it in the variable BLOCK.

*Example:*

GETBLOCK X Y ; top left of sprite

IF BLOCK = FODDERBLOCK ; is it fodder?

LET LINE = X

LET COLUMN = Y

DIVIDE LINE BY 8

DIVIDE COLUMN BY 8

PUTBLOCK 0 ; put a space there

ENDIF

**GETRANDOM**

Generates a random number between zero and the argument, and places it in the RND variable. GETRANDOM 100 will generate a number from 0 to 99, GETRANDOM 2 will generate a zero or 1.

*Example:*

GETRANDOM 100 ;

IF RND < 30 ; 30% chance

LET DIRECTION = UP

ENDIF

**INK**

Expects a single argument. Sets the foreground colour of the permanent display attributes. Can be used prior to displaying text or clearing the screen.

*Example:*

INK 5

PAPER 0

CLS

PRINT “CYAN ON BLACK”

**INV**

Requires a text string argument encased in quotes. Similar to MENU, but this produces a menu based on objects in the player's possession. The text string should contain the list of the object names used in your game separated by commas. INV will present the player with a menu containing only those items currently in the player's inventory. Useful for selecting an item to process or drop. The screen is redrawn after MENU.

*Example:*

LET LINE = 5

LET COLUMN = 6

PRINT “SELECT ITEM TO DROP”

LET LINE = 7

LET COLUMN = 9

INV “KEYS,MAGNET,FLASK,SCREWDRIVER,ZX VEGA,SCARF,BALL”

IF OPT <> 255

DROP OPT

ENDIF

**JUMP**

The current sprite will jump, provided it is not already in mid-air and there are no walls in the way. As you would expect, any sprite can be made to jump, not just those under the direct control of the player.

*Example:*

IF CANGODOWN

FALL

ELSE

IF KEY FIRE

JUMP

ENDIF

ENDIF

**KILL**

Initiates the kill player event and decrements the life counter. You should set up the lives counter in the initialisation event using something like LET LIVES = 3.

*Example:*

IF COLLISION 4

KILL

ENDIF

**LASER**

Expects LEFT or RIGHT as its argument, eg LASER RIGHT. Fires a laser left or right from the current sprite position, depending upon the parameter. Even parameters fire left, odd ones fire right. Sprite collisions with lasers can be detected in the sprite type events using IF COLLISION BULLET.

*Example:*

IF KEY FIRE

LASER RIGHT

ENDIF

**LET**

As in BASIC, this allows you to assign a value to a variable or sprite parameter. The value assigned can be a number, another variable or sprite parameter.

*Example:*

LET LIVES = 3

**MENU**

Requires a text string argument encased in quotes. Produces a pop-up menu in the middle of the play area, where the options are taken from the specified text string, separated by commas. The player then selects an option using up, down and fire controls. Usually this would happen when the player presses a certain key. However, it could pop up in another sprite event, eg when a certain sprite collides with the player. You could display a message – the sprite's conversation with the player – then use MENU to prompt for a reply. The screen is redrawn after MENU.

*Example:*

MENU “BLACK,BLUE,RED,MAGENTA,GREEN,CYAN,YELLOW,WHITE”

BORDER OPT

**MESSAGE**

Displays the message number specified in the argument.

*Example:*

AT 16 0

MESSAGE SCREEN ; display title of current screen

**MULTIPLY**

Multiply a variable or sprite parameter by a number, variable or sprite parameter.

*Example:*

MULTIPLY C BY 7

**NEXTLEVEL**

Move to next level.

*Example:*

SUBTRACT 1 FROM C

IF C = 0

NEXTLEVEL

ENDIF

**NEWPARTICLE**

Starts a new user-defined particle from the centre of the current sprite's position. Expects a parameter specifying the number of frames for which the particle should remain on screen. As AGD runs at 25 frames per second, USERPARTICLE 25 will place a particle on the screen for a duration of one second. If there is no DECAYPARTICLE command in the user particle routine the particle will remain on screen indefinitely and will only be removed if it moves out of the play area.

*Example:*

SUBTRACT 7 FROM X ; left edge of rocket sprite

NEWPARTICLE 50 ; leave a jet trail

ADD 7 TO X ; restore sprite coordinates

**OTHER**

Should only be used after a successful COLLISION check. This command selects the other sprite, ie the secondary one with which the original sprite has just collided. Any code written after OTHER will refer to the secondary sprite. Use ENDSPRITE to switch back to the original primary sprite when you have finished the code for the secondary sprite.

*Example:*

IF COLLISION 0 ; sprite collided with player?

REMOVE ; remove this sprite

OTHER ; need to change player sprite

LET IMAGE = 3 ; turn into death animation

LET TYPE = 4

LET FRAME = 0

ENDSPRITE

ENDIF

**PALETTE**

Requires 2 parameters for palette entry and RGB setting. Sets the palette entry to a new RGB value. Palette entries 0 to 7 are dull INK colours, 8 to 15 for BRIGHT INK, 128 to 135 for the dull PAPER colours and 136-143 for the BRIGHT background colours. The RGB setting is an 8-bit value with the bits laid out RRRGGGBB.

*Example:*

PALETTE 0 79

**PAPER**

Expects a single argument. Sets the background colour of the permanent display attributes. Can be used prior to displaying text or clearing the screen.

*Example:*

INK 2

PAPER 7

CLS

PRINT “RED ON WHITE TEXT”

**PARTICLEDOWN**

Command. Moves the user particle one pixel down.

*Example:*

IF S = 1 ; is starfield moving?

PARTICLEDOWN

ENDIF

**PARTICLELEFT**

Command. Moves the user particle one pixel to the left.

*Example:*

IF S = 1 ; is starfield moving?

PARTICLELEFT

ENDIF

**PARTICLERIGHT**

Command. Moves the user particle one pixel to the right.

*Example:*

IF S = 1 ; is starfield moving?

PARTICLERIGHT

ENDIF

**PARTICLEUP**

Command. Moves the user particle one pixel up.

*Example:*

IF S = 1 ; is starfield moving?

PARTICLEUP

ENDIF

**PLOT**

Requires 2 arguments for X and Y pixel coordinates. Plots or unplots a pixel at the specified point on screen.

*Example:*

PLOT 127 87

**PRINT**

Expects a string argument encased in quotes containing the text to be displayed on screen.

*Example:*

CLS

PRINT “HELLO WORLD!”

**PRINTMODE**

Requires a single numeric argument. Sets the text display mode to standard or double height, depending on the argument. Display mode 0 is standard height text, mode 1 is double height.

*Example:*

CLS

PRINTMODE 0

PRINT “STANDARD TEXT”

PRINTMODE 1

PRINT “DOUBLE-HEIGHT TEXT”

**PUT**

Requires three arguments specifying the object number, its x coordinate and y coordinate. The object is dropped onto the screen at the specified position, unless it is already on the current screen. Arguments can be numeric, variables, sprite parameters or any combination thereof.

*Example:*

IF KEY FIRE

IF GOT 0

PUT 0 X Y

ENDIF

ENDIF

**PUTBLOCK**

Requires a parameter to specify the block number. This places the specified block at the current LINE and COLUMN position on the screen, and can be useful for opening and closing doors. Note that exiting and re-entering the screen or redrawing after MENU or INV will restore the screen to its default position.

*Example:*

GETBLOCK X Y

IF BLOCK = FODDERBLOCK

LET LINE = Y

LET COLUMN = X

DIVIDE LINE BY 8

DIVIDE COLUMN BY 8

PUTBLOCK 0

ENDIF

**RANDOMIZE**

Sets the seed for the random number generator to a specified 8-bit value.

*Example:*

RANDOMIZE CLOCK

GETRANDOM 100

PRINT RND

**READ**

Reads a value from a list of numbers in a DATA statement contained at the end of the event and places it in the specified variable or sprite parameter. At the end of the DATA statement AGD automatically restarts at the beginning so there is no need to do RESTORE yourself unless the routine calls for it. READ and DATA are local to each event. In the case of sprite events, each individual sprite knows where it is in the data list regardless of how many other sprites are accessing that list. If you wished you could read your enemy sprites’ X and Y coordinates from a very long DATA list.

*Example:*

REPEAT 9

READ A

CHR A

CRASH 10

DELAY 5

ENDREPEAT

DELAY 100

REDRAW

DATA ‘G’ ‘E’ ‘T’ ‘ ‘ ‘R’ ‘E’ ‘A’ ‘D’ ‘Y’

**REDRAW**

Redraws the screen, sprites, objects and any particles. If you are writing an arcade adventure, you may wish to display messages in the play window, pause for a key with WAITKEY and then REDRAW before returning to the action. No parameter is required.

*Example:*

LET LINE = 11

LET COLUMN = 10

PRINT “GAME PAUSED”

WAITKEY

REDRAW

**REMOVE**

Removes the present sprite from the table. Useful for destroying enemies or picking up objects.

*Example:*

IF COLLISION 0 ; has it been collected by the player?

REMOVE ; remove the sprite

SCORE 25 ; score some points

SOUND 0

LET LINE = 1

LET COLUMN = 1

SHOWSCORE

ENDIF

**REMOVEOBJ**

Removes the specified object from its current position. Needs a single parameter specifying the object to be removed.

*Example:*

REMOVEOBJ 4 ; remove old object

GET 5 ; put new object in inventory

**REPEAT**

Requires a parameter to determine the number of repeats. The code up to the next ENDREPEAT will be repeated the given number of times. Each REPEAT needs a matching ENDREPEAT. REPEAT can be nested up to three levels.

*Example:*

CLS

REPEAT 32

PRINT “\*”

ENDREPEAT

**RESTART**

Restart the current level/screen.

*Example:*

RESTART

**RESTORE**

Resets the READ and DATA status so that the next READ starts at the first DATA item in the list.

*Example:*

RESTORE ; Give random bonus.

GETRANDOM 5

ADD 1 TO RND ; 1 to 5 reads of bonus table.

REPEAT RND

READ A

ENDREPEAT

SCORE A

DATA 50 75 100 150 200

**SCORE**

Expects to be followed by a number to add to the score, eg SCORE 100 will add 100 points to the player's total. Values in the range 0 to 65535 are valid.

*Example:*

SCORE 100

LET LINE = 1

LET COLUMN = 1

SHOWSCORE

**SCREENDOWN**

Move down one screen if possible.

*Example:*

IF Y > BOTTOMEDGE

SCREENDOWN

LET Y = TOPEDGE

EXIT

ENDIF

**SCREENLEFT**

Move left a single screen if possible.

*Example:*

IF X < LEFTEDGE

SCREENLEFT

LET Y = RIGHTEDGE

EXIT

ENDIF

**SCREENRIGHT**

Move right one screen if possible.

*Example:*

IF X > RIGHTEDGE

SCREENRIGHT

LET X = LEFTEDGE

EXIT

ENDIF

**SCREENUP**

Move up one screen if possible.

*Example:*

IF Y < TOPEDGE

SCREENUP

LET Y = BOTTOMEDGE

EXIT

ENDIF

**SHOWBONUS**

Shows the bonus total at the current cursor position. Should be immediately preceded by instructions setting up the line and column position for the cursor.

*Example:*

AT 3 1

PRINT “BONUS:”

AT 4 1

SHOWBONUS

**SHOWSCORE**

Shows the score at the current cursor position. Should be immediately preceded by instructions setting up the line and column position for the cursor.

*Example:*

PRINT “SCORE:”

SHOWSCORE

**SHOWHIGH**

Shows the high score at the current cursor position. Should be immediately preceded by instructions setting up the line and column position for the cursor.

*Example:*

PRINT “HIGH SCORE:”

SHOWHIGH

**SILENCE**

Silences the AY sound chip.

*Example:*

SILENCE ; stop all current sounds

SOUND 1 ; play new sound

**SOUND**

Starts a sound effect with the AY sound chip. Expects a single parameter for the sound number to play.

*Example:*

IF KEY FIRE

LASER DIRECTION

SOUND 1

ENDIF

**SPAWN**

Expects 2 parameters for sprite type and image. This spawns a new sprite with the specified type and image at the current sprites's position. The new sprite is created with FRAME, DIRECTION, SETTINGA and SETTINGB all set to zero. The current sprite is unaffected.

*Example:*

SPAWN 0 0 ; split the player into two

SUBTRACT 8 FROM X ; move original sprite left

SPAWNED

ADD 8 TO X ; move new sprite right

ENDSPRITE

**SPAWNED**

Should only be used after a SPAWN instruction. This command selects the newly spawned sprite. Any code written after SPAWNED will refer to the new sprite. Use ORIGINAL to switch back to the original primary sprite. Alternatively, you may prefer to place your code in the Sprite Initialisation event, it's up to you.

*Example:*

SPAWN 1 4

SPAWNED

LET X = RIGHTEDGE

ENDSPRITE

**SPRITEDOWN**

Move the current sprite accordingly. No check is made for blocks in the way, or out-of-screen conditions, so you will have to do that yourself with the functions CANGODOWN or LADDERBELOW.

*Example:*

IF KEY DOWN

IF Y < BOTTOMEDGE

SPRITEDOWN

ENDIF

ENDIF

**SPRITEINK**

Sets the ink colour for the current sprite to the specified colour, eg SPRITEINK 4 or SPRITEINK SETTINGB. Does not affect PAPER, FLASH or BRIGHT. To avoid leaving a trail of differently-coloured INK, reset the SPRITEINK at the start of each sprite event. If your background INK is normally 7 (white), you may wish to use SPRITEINK 7 at the beginning of the sprite event, move the sprite as normal, then set it to the colour you want. Static collectable sprites should only reset the SPRITEINK to the normal background INK when they are collected. Has no effect on Amstrad CPC.

*Example:*

SPRITEINK 2

**SPRITELEFT**

Move the current sprite accordingly. No check is made for blocks in the way, or out-of-screen conditions, so you will have to do that yourself with the CANGOLEFT function.

*Example:*

IF KEY LEFT

IF X > LEFTEDGE

SPRITELEFT

ENDIF

ENDIF

**SPRITERIGHT**

Move the current sprite right. No check is made for blocks in the way, or out-of-screen conditions, so you will have to do that yourself with CANGORIGHT.

*Example:*

IF KEY RIGHT

IF X < RIGHTEDGE

SPRITERIGHT

ENDIF

ENDIF

**SPRITEUP**

Move the current sprite. No check is made for blocks in the way, or out-of-screen conditions, so you will have to do that yourself with functions such as CANGOUP or LADDERABOVE.

*Example:*

IF KEY UP

IF Y > TOPEDGE

SPRITEUP

ENDIF

ENDIF

**STAR**

Part of the particle engine, STAR creates a vertical or horizontal starfield when called repeatedly. Expects a LEFT, RIGHT, UP or DOWN parameter to specify the direction.

*Example:*

STAR DOWN

**SUBTRACT**

Subtract an amount from a sprite parameter or variable.

*Example:*

SUBTRACT 5 FROM B

**TABLEJUMP**

Performs a fixed jump using a jump table. Unlike JUMP, no parameter is required as the extent of the jump is defined in the jump table. Included for compatibility with ZX Spectrum version 4.7.

*Example:*

IF KEY FIRE2

IF GOT 4 ; got the seven league boots?

JUMP 8 ; yes, super jump

ELSE

TABLEJUMP ; no, do standard fixed jump

ENDIF

ENDIF

**TRAIL**

Useful for fairy dust or vapour trails, TRAIL creates a vapour trail particle at the centre of the current current sprite. By using ADD 8 TO X or SUBTRACT 8 FROM X first, you could create a trail to the left or right of the sprite. Remember to ADD or SUBTRACT 8 again after the TRAIL command to restore the sprite's original coordinates.

*Example:*

IF CANGORIGHT

SPRITERIGHT

SUBTRACT 8 FROM X

TRAIL

ADD 8 TO X

ELSE

EXPLODE 20

REMOVE

ENDIF

**UNDOSPRITEMOVE**

Undoes any sprite movement undertaken for the current sprite since the beginning of the event.

*Example:*

IF DIRECTION = LEFT

SPRITELEFT

ELSE

SPRITERIGHT

ENDIF

IF DEADLY

UNDOSPRITEMOVE

IF DIRECTION = LEFT

LET DIRECTION = RIGHT

ELSE

LET DIRECTION = LEFT

ENDIF

ENDIF

**WAITKEY**

Waits for a keypress.

*Example:*

PRINT “PRESS A KEY”

WAITKEY

**WHILE**

Repeats a loop while the condition in the accompanying argument is true. Every WHILE must have a matching ENDWHILE. WHILE loops can be nested.

*Examples:*

LET L = 0

WHILE L < 5

DISPLAY L

ADD 1 TO L

ENDWHILE

DEFINEKEY FIRE ; redefine the key for fire

WHILE KEY FIRE ; wait for key to be released

ENDWHILE

**ZEROBONUS**

Resets the bonus.

*Example:*

SHOWSCORE

DELAY 25

PRINT “ + ”

DELAY 25

SHOWBONUS

DELAY 25

PRINT “ = ”

DELAY 25

ADDBONUS

ZEROBONUS

SHOWSCORE

# Other Codewords

**ELSE**

Marks the end of the IF condition and the beginning of the alternative code, should the IF test fail.

**ENDIF**

Marks the end of the conditional code.

**IF**

Test. If the following condition is true the code up to the next ELSE or ENDIF statement is executed. IF can be used with a function, or to test variables or sprite parameters against each other, or against specific numeric values.

# Operators

**<** is less than

**>** is greater than

**<=** is less than or equal to

**>=** is greater than or equal to

**=** equals

**<>** does not equal

# Comments

Comments can be inserted into the code by use of the semi-colon “;” character. Following a semi-colon everything is ignored until the start of the next line.

# Numbers

AGD defaults to denary but will accept hexadecimal numbers. These should be prefixed with a dollar ($) character.

# Strings

Where a quote is required within a string, use a double quote.

Newlines inside strings are permitted.