Programming_language

COLLABORATORS			
	TITLE : Programming_language		
ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		July 22, 2022	

REVISION HISTORY			
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

Programming_language

1.1 RoboQuest programming language

```
                *=========================================================================
This is an AmigaGuide documentation file for the programming language used in
the game RoboQuest. Please select a topic below:

    1.
                Structure
                    2.
                The commands
                    3.
                The functions
                    4.
                The system variables
                    5.
                Expressions
                    6.
                Error messages
                    0. Main documentation file
*=========================================================================*
```

1.2 The structure

RoboQuest uses a programming language similar to BASIC. The language is most similar to Commodore 64 BASIC v2, although there are a few differences. Like C64 BASIC v2, RoboQuest uses line numbers. These, however, are an integral part of the program editor and needn't be typed in by the player. They can be found at the left side of the screen.

RoboQuest commands, functions and variable names consist of letters. Upper case and lower case letters are considered equal, like in BASIC and Pascal, but differently from C or Java. Commands can be separated by colons (:). You can add or omit spaces freely in the program, with one exception: if a command's first parameter begins with a variable, there must be at least one space between the command and the variable.

There is a memory bank for the benefit of your data. You can use this memory

bank to store whatever data you want. It is accessed byte by byte with the POKE command and the PEEK function, just like in C64 BASIC v2. It is automatically reserved to be approximately 1/8 of your available memory. The size of this memory bank is written into the system variable MEMSIZE.

1.3 The commands

CHDIR <directory$>
Changes to the directory specified. If the directory does not exist, reports an error message.

CLS
Clears the text output screen.

COLOUR <index>,<rgbvalue>
Changes the RGB value of the specified index. The indeces range from 0 to 15. The RGB value is calculated as follows: (amount of red*256)+(amount of green*16)+(amount of blue). Amounts range from 0 to 15. For example black is 0, blue is 15, green is 240, red is 3840, and white is 4095.

DEC <variable>
Decrements the contents of the numeric variable specified by 1. It is impossible to decrement string variables or system variables.

DEGREE
Switches to degree mode.

DOWN [<amount>]
Moves the robot down the specified amount of squares. The amount defaults to 1. The robot will stop moving if it hits an obsracle or runs out of fuel.

DROP
Drops whatever the robot is carrying, if there is space on the ground. Dropping a diamond onto a diamond deposit advances the amount of diamonds collected and causes the diamond to disappear.

EAST [<amount>]
Equivalent to RIGHT.

END
Ends the program.

ENDGAME
Ends the current game altogether, returning to the Main Menu. The confirmation question will be skipped.

FOR <variable>=<start> TO <end> [STEP <value>]
Defines a FOR...NEXT loop of the variable specified, like in normal BASIC. The step value defaults to 1.

GOSUB <linenumber>
Goes to the subroutine starting from the specified line number. A subsequent RETURN command will return to the command following this command.

GOTO <linenumber>

Makes execution skip to the specified line number. If the line number is less than 0 or greater than 99, an error will be reported.

GRADE
Switches to grade mode.

IF <condition> THEN ...
Checks the value of the condition following the IF word. If the value is different from 0, the line is executed normally. If the value is 0, execution will skip to the next line.

INC <variable>
Increments the specified number variable by 1. It is impossible to increment string variables or system variables.

INPUT <variable($)>[,<variable($)>...]
Allows the user to key in values for the specified number or string variables. The variables will be processed one at a time.

INTERPRET <commandstring$>
Executes the contents of the command string as it were a line of the program. The command string doesn't have a line number, and some commands, such as jumps, are impossible to execute from it. When the command string has finished, execution will return to the command following INTERPRET in the actual program.

KEYPRESS
Waits for a keypress.

KILL <file$>
Deletes the specified file from disk. No questions will be asked. If the file does not exist, or can't be deleted, an error will be reported.

LEFT [<amount>]
Moves the robot left the specified amount of squares. The amount defaults to 1. The robot will stop moving if it runs out of fuel or hits an obstacle.

LOAD <file$>,<start>[,<end>]
Loads the contents of the file into the given address in the memory bank. The file is trimmed to fit the bank, if necessary.

LOCATE <x>,<y>
Moves the cursor in the text output screen to the new coordinates. If these are out of screen range, an error will be reported.

MKDIR <directory$>
Makes a new directory onto disk. If the making fails, an error will be reported.

MOVE <deltax>,<deltay>
Moves the robot the specified amount of squares. Deltax is positive for moving right, negative for moving left. Deltay is positive for moving down, negative for moving up. The robot will stop moving if it runs out of fuel or hits an obstacle.

MUSIC <file$>
Loads the file into memory and plays it as a ProTracker, MED or AMOS module. If the module format is not recognised, nothing will be played.

```
    NEXT [<variable>]
Concludes a FOR...NEXT loop, ready for the next execution. If the variable has
reached or exceeded the end value, execution will proceed from the next command.

    NORTH [<amount>]
Equivalent to UP.

    PAPER <value>
Changes the background colour of the text output screen to the index specified.
Valid indeces range from 0 to 15.

    PEN <value>
Changes the foreground colour of the text output screen to the index specified.
Valid indeces range from 0 to 15.

    POKE <address>,<value>
Changes the value of the given address in the memory bank. Valid values range
from 0 to 255. Valid addresses range from 0 to MEMSIZE-1. It is impossible to
poke outside the memory bank, therefore this command will never crash the
computer or cause any other kind of harm.

    PRINT [<value($)>[(,/;)<value($)...]]
Prints the specified parameters onto the text output window. If called with no
parameters, prints an empty row. A comma will skip to the next tabulation point,
while a semicolon will proceed from the next column. If neither are present,
the cursor will move to the next row. PRINT can also be written as ?.

    RADIAN
Switches to radian mode.

    REFUEL
If the robot is carrying a fuel container, and has a fuel supply of 204 or less,
it will drain the container into its own tanks. All containers contain 51 units
of fuel. The empty container will disappear.

    REM anything you want
Causes program execution to simply ignore whatever is written on the remainder
of the line. REM can also be written as '.

    RENAME <source$>,<destination$>
Renames the sourcefile to the destinationfile. If the renaming fails, an error
will be reported.

    RETURN
Returns from a previously called subroutine.

    RIGHT [<value>]
Moves the robot right the specified amount of squares. The amount defaults to 1.
The robot will stop moving if it runs out of fuel or hits an obstacle.

    SAVE <file$>,<start>[,<end>]
Saves the specified block of the memory bank into the specified file.

    SAVEGAME <file$>
Saves the current game situation (map) into the specified file. This file can
later be played in RoboQuest or edited in RoboEd.
```

```
    SAY <quote$>
Speaks the given quote using the world-famous Amiga speech syntheziser. The
resulted speech will be inferior to the sampled speech experienced in current
PC or PlayStation games, but will need only a small fraction of the memory
those things need. The device devs:narrator.device must be present for speech
synthezisation to work.

    SCAN <variable>,<x>,<y>
Scans the given location to the given number variable. If the location has not
been explored (shown as a grey block on the map), the value will be -1.
Otherwise, it will be calculated as follows: (item on block*16)+(base block).

    SCROLL <deltax>,<deltay>
Scrolls the map window the specified amount of squares. Affects only viewing,
not the actual map. Only the final destination will appear, there is no fancy
"scrolling" effect.

    SETTIME <value>
Sets the value of TIMER to the specified amount. This has no effect on the
actual game clock, which is reset to 0 on initial loading of the game and will
keep track of all time spent while playing.

    SHAVE <variable$>,<start>,<end>
Shaves the specified string variable so that only the positions from the start
value to the end value remain. The values cannot cross each other, i.e. the
start value cannot exceed the end value. System variables cannot be shaved.

    SHAVEL <variable$>,<length>
Shaves the string variable so that only the specified first positions remain.
Equivalent to SHAVE <variable$>,1,<length>.

    SHAVER <variable$>,<length>
Shaves the string variable so that only the specified last positions remain.
Equivalent to SHAVE <variable$>,LEN(<variable$>)-<length>+1,LEN(<variable$>).

    SOUTH [<amount>]
Equivalent to DOWN.

    STARTMUSIC
Starts playing the currently loaded music module, if there is one. Sound effects
will be inaudible.

    STOPMUSIC
Stops playing the currently loaded music module. Sound effects will be audible.

    STRPEEK <start>,<end>,<variable$>
Stores the contents of the given portion of the memory bank into the given
string variable as ASCII text.

    STRPOKE <address>,<string$>
Writes the contents of the string variable into the specified address in the
memory bank. The string will be trimmed to fit the bank, if necessary.

    SWAP <variable1($)>,<variable2($)>
Swaps the contents of the given number or string variables. Variable types can't
be mixed. System variables can't be swapped.
```

```
    SWITCH
Causes the robot to pick up whatever is on the ground, if it is carrying
nothing. Otherwise it will drop anything it is carrying, if there is space. Two
consecutive SWITCH commands will cancel each other, with the exception of
dropping diamonds onto diamond deposits.

    TAKE
Causes the robot to pick up whatever is on the ground, if it is carrying
nothing.

    TELEPORT
Causes the robot to use the teleport it is standing on, if it is standing on
one.

    UP [<amount>]
Moves the robot up the specified amount of squares. The amount defaults to 1.
The robot will stop moving if it runs out of fuel or hits an obstacle.

    VIEW <x>,<y>
Views the map window from the given coordinates, which correspond to the upper
left corner. Only affects the map window and not the actual map.

    WAIT <time>
Waits the given amount of jiffies. 1 jiffy=20 milliseconds, so 1 second=50
jiffies.

    WEST [<amount>]
Equivalent to LEFT.

    XFIRST
Changes the effect of the MOVE command so that the robot will move horizontally
first.

    YFIRST
Changes the effect of the MOVE command so that the robot will move vertically
first.
```

1.4 The functions

```
    ABS(value)
Returns the absolute value of the specified number. Negative numbers are
multiplied by -1, positive numbers or zero are unaffected. The result will
always be non-negative.

    ACOS(value)
Returns the angle whose cosine is the specified value. The result will depend
on the current trigonometric mode.

    ASC(character$)
Returns the ASCII code of the character. If the character is a string, only the
first character will be used.

    ASIN(value)
Returns the angle whose sine is the specified value. The result will depend on
the current trigonometric mode.
```

```
    ATAN(value)
Returns the angle whose tangent is the specified value. The result will depend
on the current trigonometric mode.

    BACKWARD$(string$)
Returns the string written backwards.

    CARGO(robotid)
Returns the item number of the cargo the specified robot is carrying. Valid IDs
range from 0 to 3. If the robot is carrying nothing, 0 is returned.

    CBR(value)
Returns the cubic root of the value specified.

    CHR$(code)
Returns the character whose ASCII code is specified. Valid codes range from 0 to
255.

    COLOUR(index)
Returns the RGB value of the specified index. Valid indeces range from 0 to 15.

    COS(angle)
Returns the cosine of the angle. The result will depend on the current
trigonometric mode.

    CSC(angle)
Returns the cosecant of the angle. The result will depend on the current
trigonometric mode.

    EXIST(file$)
Returns -1 if the file exists, 0 otherwise.

    EXP(value)
Returns the exponential of the value, that is e to the given power.

    FLIP$(string$)
Returns the string with all characters flipped in case (upcase becomes
lowcase, lowcase becomes upcase).

    FRAC(value)
Returns the fractional part of the number, according to BASIC convention,
which substracts 1 from negative numbers.

    FRACT(value)
Returns the fractional part of the number, according to true Euclidean
mathematics, which treats negative numbers normally.

    FUEL(robotid)
Returns the amount of fuel the specified robot currently has. Valid IDs
range from 0 to 3.

    HCOS(angle)
Returns the hyperbolic cosine of the angle. The result will depend on the
current trigonometric mode.

    HEX$(value)
```

Returns the value in hexadecimal.

HEXB$(value)
Returns the value in hexadecimal in byte format, always using 2 digits.

HEXL$(value)
Returns the value in hexadecimal in long word format, always using 8 digits.

HEXW$(value)
Returns the value in hexadecimal in word format, always using 4 digits.

HSIN(angle)
Returns the hyperbolic sine of the angle. The result will depend on the current trigonometric mode.

HTAN(angle)
Returns the hyperbolic tangent of the angle. The result will depend on the current trigonometric mode.

INT(value)
Returns the integer part of the given value, according to BASIC convention, which substracts 1 from negative numbers.

INTG(value)
Returns the integer part of the given value, according to normal Euclidean mathematics, which treats negative numbers normally.

LEN(string$)
Returns the length of the string.

LN(value)
Returns the natural (e-based) logarithm of the value.

LOG(value)
Returns the 10-based logarithm of the value.

LOWER$(string$)
Returns the string in all lower case.

PEEK(address)
Returns the value of the given address in the memory bank.

RND(maxvalue)
Returns a random number ranging from 0 to the maxvalue.

ROBX(robotid)
Returns the X coordinate of the given robot. Valid IDs range from 0 to 3.

ROBY(robotid)
Returns the Y coordinate of the given robot. Valid IDs range from 0 to 3.

SEC(angle)
Returns the secant of the angle. The result will depend on the current trigonometric mode.

SGN(value)
Returns the sign of the value, which is +1 for positive numbers, 0 for zero,

or -1 for negative numbers. Equivalent to value/ABS(value), except for zero.

    SIN(angle)
Returns the sine of the angle. The result will depend on the current trigonometric mode.

    SQR(value)
Returns the square root of the given value.

    STR$(number)
Returns the number as a string. If it is non-negative, a leading space will be added.

    TAN(angle)
Returns the tangent of the angle. The result will depend on the current trigonometric mode.

    TEN(value)
Returns 10 raised to the given power. Opposite function to LOG.

    UPPER$(string$)
Returns the string in all upper case.

    VAL(string$)
Returns the numeric value of the string. Evaluation will stop at the first non-number character.

1.5 System variables

    CHIPMEM
The amount of free Chip memory, in bytes.

    CURSX
The X coordinate of the cursor in the text output screen.

    CURSY
The Y coordinate of the cursor in the text output screen.

    DIAMONDS
The amount of diamonds collected on this map.

    DIR$
The current directory on the disk.

    DOWN
The joystick value equal to moving down.

    DOWN$
The character equal to the cursor down key.

    EAST
Equal to RIGHT.

    EAST$
Equal to RIGHT$.

```
    EE
Napier's number, written commonly as e. Named EE to reserve E for your own use.

    FALSE
0. To be used in tests.

    FASTMEM
The amount of free Fast memory, in bytes.

    FIRE
The joystick value equal to pressing Fire.

    INKEY$
Whatever character is currently pressed on the keyboard.

    JOY
Wherever the joystick is currently moved.

    KEYSHIFT
The state of the key qualifiers currently pressed, such as Shift, Alt or Ctrl.

    LEFT
The joystick value equal to moving left.

    LEFT$
The character equal to the cursor left key.

    MAPNAME$
The name of the map file currently being played. Not the actual filename, but a
special name written inside the file.

    MAPX
The X coordinate of the currently viewed upper left corner.

    MAPY
The Y coordinate of the currently viewed upper left corner.

    MEMSIZE
The size of the memory bank, in bytes.

    MSG$
The message contained in the message blocks last passed by any robot.

    MSTAT
-1 if there is any music module currently playing, 0 otherwise.

    MTYPE
1...5 depending on the type of music module being played, 0 if no music module
is loaded.

    MUSIC$
The name of the music module currently loaded.

    NO
0. To be used in tests.
```

```
    NORTH
Equal to UP.

    NORTH$
Equal to UP$.

    PAPER
The current background colour index in the text output screen.

    PEN
The current foreground colour index in the text output screen.

    PI
The value pi, which is a constant, the ratio of a circle's perimeter and radius.

    RANDOM
A random number between 0 and 1.

    RIGHT
The joystick value equal to moving right.

    RIGHT$
The character equal to the cursor right key.

    ROBOT
The robot ID currently in use.

    ROW
The current line number. If called within INTERPRET, returns -1.

    SCANCODE
The keyboard internal scan code of the key currently pressed.

    SOUTH
Equal to DOWN.

    SOUTH$
Equal to DOWN$.

    SPEECH
-1 if devs:narrator.device exists, 0 otherwise.

    TIMER
A variable constantly keeping track of time, in jiffies (20 milliseconds). Can
be reset with the SETTIME command.

    TRUE
-1. To be used in tests.

    UP
The joystick value equal to moving up.

    UP$
The character equal to the cursor up key.

    WEST
Equal to LEFT.
```

```
    WEST$
Equal to LEFT$.

    XYORDER
1 if XFIRST is set, i.e. the MOVE command moves the robots horizontally first,
2 otherwise.

    YES
-1. To be used in tests.
```

1.6 The expressions

RoboQuest uses a system of expression evaluation common in BASIC, Pascal or C languages. Operands can be either numbers or strings. Numbers are written normally (hexadecimal numbers can be written with a preceding $), and strings are written in quotes ("").

Variables can be written as any operands in expressions. Variable names consist of letters, and can be up to 62 characters long. Variables names can be names used as commands or functions, with a few exceptions. In other words, it is OK to write lines such as PRINT=1, PRINT PRINT+1.

The operators are evaluated according to priority precedences, and from left to right. Higher-priority operators are evaluated before lower-priority ones.

The evaluation system is based on an expression evaluation program written by Chris Hodges (platon@cu-muc.de), however I have expanded it a great deal.

The operators are, in order of priority:

1. () [] {}
Parantheses cause the expression included in them to be forcibly evaluated before the outside expression. Normal parantheses () function normally. Brackets [] convert strings to numbers and vice versa. Curly braces {} take a string expression and evaluate its contents once more. For example: ("1+1")="1+1", but {"1+1"}=2.

2. A^B
Calculates A to the power of B.

3. A*B A/B
Calculates A multiplied by B, or A divided by B. You cannot divide anything by zero.

4. A+B A-B -A ~A
Calculates A added to B, B substracted from A, the negative (two's complement) of A, or the inverse (one's complement) of A.

5. A=B A<>B A<B A>B A<=B A>=B
Comparison operators: equal to, not equal to, less than, greater than, less than or equal to, greater than or equal to. True comparisons return -1, false comparisons return 0.

6. A&B A|B A\B
Bitwise operators: disjunction (AND), conjunction (OR), and exclusive conjunction (XOR).

7. A?<B A?>B
Return the lesser or the greater of the two values. If they are identical, the value returned can be either.

The following operators can be used with strings:
1. All parantheses.
2. A$+B$, A$-B$. The + sign means concatenation (adding the second string where the first one ends), and the - sign erases all occurences of B$ from A$.
3. All comparison operators.
4. A$?<B$, A$?>B$. These, like the comparison operators, sort the strings by their ASCII values.

1.7 The errors

Cannot assign system variable
System variables can only be read, not written. They cannot be used in assignments, e.g. <variable>=<value>.

Command cannot be used within Interpret
The INTERPRET command cannot execute some commands, such as jumps, loops or other INTERPRET commands. Use these in the program itself.

Device not found
You have tried to access a device which is not there. If the device is physical, check the connections. If it is logical, check all necessary mountlist files.

Disk error
The file you have tried to access can't be either read or written. This can be caused by a bad sector on the disk. Use some sort of disk salvation utility.

Disk full
The current disk is full and cannot accept the data saved. Delete some data or use a different disk.

Disk is write protected
The current disk is write protected and cannot accept any saved data. Unprotect the disk or use a different disk.

Division by zero
You have tried to divide something by 0. RoboQuest uses standard Euclidean mathematics, and cannot deal with this hypothetical value.

End of program
Not an error, but a signal that the program has finished and the game accepts mouse control again.

Error in loading AMOS module
The AMOS module you have tried to load is either corrupted, in the wrong type of bank, or not a module in the first place.

Error in loading MED module
The MED module you have tried to load is either corrupted or not a module in the first place.

```
    File already exists
You cannot rename a file to a filename already in use.

    File not found
The file you have tried to read does not exist.

    Illegal function call
The specified number value is out of range. Valid ranges depend on the command
or function used. The minimum value is often 0, however.

    Illegal line number
You can't jump outside the program, i.e. to rows before 0 or after 99.

    Illegal memory address
To protect the intricate memory allocation system of the Amiga, the program will
refuse to read or write memory addresses outside the memory bank. Valid
addresses range from 0 to MEMSIZE-1.

    Incorrect file type
There was some sort of error in loading the requested file.

    Next without For
The program has found a NEXT command with no corresponding previous FOR command.
This can be because of variable conflicts, linked loops or simply the absence of
any FOR command.

    No disk in drive
You have tried to access a drive which currently holds no disk. Obviously the
drive cannot be read or written to.

    Not an AmigaDOS disk
The disk is either unformatted or in a foreign disk format not supported by
AmigaDOS, and can't be accessed at all. Format the disk or use a different disk.

    Out of variable storage
Your program uses more than 256 user variables, and causes the variable storage
system to overflow. Remove the mentions of unnecessary variables.

    Program interrupted
Not an error, but a signal that you have pressed ^C, interrupting the program.
The game will accept mouse control again.

    Return without Gosub
The program has encountered a RETURN command with no previous GOSUB command. As
there is nothing to return to, the RETURN command fails.

    Syntax error
There is a syntax error in the parameter list of a command. It is usually a
character in the wrong place.

    Too many recursion levels
You can only have 10 nested levels of paranthesis recursion. Simplify your
formulas.

    Too many subroutines
You can only have 10 nested subroutines. Simplify your program flow.
```

Type mismatch
You can’t use a number when the program expects a string, or a string when the program expects a number.

Unknown command
You have misspelled a command name, or remembered a command which is not there. Check this manual.

Unknown function
You have misspelled a function name, or remembered a function which is not there. Check this manual.