Guide for JONGL

COLLABORATORS

	TITLE : Guide for JONGL	

ACTION	*NAME*	*DATE*	*SIGNATURE*
WRITTEN BY		August 23, 2022	

REVISION HISTORY

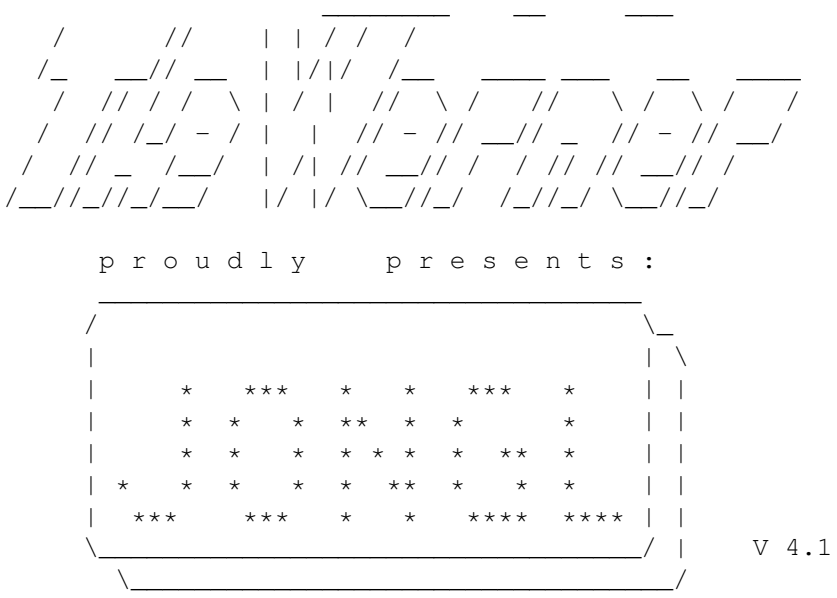
NUMBER	DATE	DESCRIPTION	NAME

Contents

Chapter 1

Guide for JONGL

1.1 Guide for JONGL



Short version:

JONGL is a juggling simulation program. You can watch patterns with one or more jugglers throwing almost anything from balls to chain saws. While the jugglers are doing their pattern you can watch the scene from any view you like using the mouse. Don’t forget to try the mouse buttons.

Extremely short version:

Type jongl at the prompt, select one of the patterns, watch the show and press every key on the keyboard at least three times.

You may also fiddle around with the menues (M).

Date of issue of this Guide: 24.6.96

Many many thanks to my father Hermann Riebesel who translated this guide.

1.2 Inhaltsverzeichnis

1.3 Introduction

The introduction (fully automatically) falls into the following subjects:

What is the program good for ?

Legal stuff

System requirements

New features

The programmers

1.4 What is the program good for?

JONGL is a juggling simulation program. This means you can look at (almost) any juggling pattern you like involving one to ten people. Unlike almost all other programs of this kind, JONGL shows also the jugglers and the floor. Presentation may be selected with filled-in polygons, shaded objects or shadows on the floor (very important...). Since the animation is computed in real time, view point and view angle may be varied with the mouse during the juggling performance. For instance you could beam yourself into the head of one juggler to see how it feels to be able to juggle with seven tennis rackets.

This is another feature of JONGL not found in other programs: You can throw around any object you want, provided you have defined it before. The program comes complete with balls, rings, clubs in different colours, a potted plant, a chain saw, a tennis racket etc.

JONGL accepts ASCII files which define objects and patterns. The program may be expanded by changing files or creating new files. In this context please refer to

Compensation
.

There is a version for 68000 and for 68020/68881 each. The latter is called "turbo version" and has some additional features which are provided by the coprocessor or the 68020.

Important note for english users: The program has its origin in the german language. Therefore most of the texts are german. Starting with Version V4.1 I supply an english guide (this one) and some translations in the program and its files. Currently the object names and most of the error messages in the program aren't translated. I will think about an inprovement of this (which will be sped up by responses from you).

In case it is of interest for someone: The name JONGL is derived from the german word for juggling: Jonglieren.

Please take also note of the subjects

Legal stuff
and
The programmers
!

1.5 Legal stuff

Jongl is freely distributable; nevertheless all rights remain with us. It is expressly allowed to include this program in Aminet and Internet, in Aminet CD ROMs and in mailboxes. It is not allowed to sell this program for a price higher than justified by the disk.

This program does not cost you anything. Therefore you cannot expect us to be responsible for anything that the program might cause under odd circumstances.

(Up to now it has never made a mess anywhere...)

1.6 System requirements

In principle JONGL should run on any AMIGA. Of course we could not ↩
test
them all. But the fact is: The faster the AMIGA, the more fun! This is
supported by experience: Martin (one of
The programmers
) while working
through his routines, bought himself a turbo card, and everything was
simply much coooooler!

Speed is much more important than RAM expansion. Much (really VERY much)
RAM is needed only in the Recording Mode for patterns with a long period
and with complicated objects. Anything else is no problem with 1 MB free
memory.

The stack requirements of the program have not been tested, since I have
lost my trust in the stack overflow function of the C compiler. I have
20.000 stack and JONGL works fine.

JONGL is presently designed for 68030 and 68882 at 50 MHz. This
combination of processors provides a presentation on the screen in real
time. For AMIGAs with other processor speeds the patterns have to be
adjusted with G (see
Keyboard
).

1.7 New features

This is the brand new version V4.1 which makes the utterly old V4.0 to
rubbish.

Here is the history:

V4.1

* in blitter mode with less than 4 bitplanes spheres in correct colours

* pattern selection menue now in WB resolution (on GFX cards too)

* you can enter the number of columns in jongl.prefs

* correct centering without overscan

* no brown bar at the top of the screen

* new key H for hardcopies. needs iff.library and a new line in
  jongl.prefs

```
* ENGLISH VERSION. If you add a new line in jongl.prefs you have the
  program, the objects, the messages and the menues in english. Only
  the object names and the error messages are still in german.

* corrected some errors in the guide

* a few new patterns


V4.0

* first released version
```

1.8 The programmers

```
                Everything but the projection routines has been thought out by:

 Werner Riebesel
 Dipl.-Ing. Univ. (wow!)
 Auenbruggerstr. 110
 D-80999 München
 Germany
 phone 0049 / 89 / 8 12 51 52
 email: Katja.und.TheWerner@t-online.de
        or
        bm964867@muenchen.org

The complete projection (in assembler) including sorting for distance,
presentation with shadows and filled-in polygons has been designed by:

 Martin Hoffmann
 Löherweg 10b
 D-80997 München
 Germany
 phone 0049 / 89 / 145 911
 email: Hoffmann@fhmrz4.rz.fh-muenchen.de

Please do not miss
                Compensation
                 and Martin's explicit explanations
covering the topic
                Speed
                .
```

1.9 Files required by the program

```
                JONGL will not run or run only in a limited way, if the following ↩
                   files
and directories are not found:
```

```
o (dir):        contains
                Object files
                 in which the stuff is defined that
                is thrown around

o/basic (dir):  contains basic static objects like a man & his beer case

m (dir):        contains
                Pattern files
                 in which the juggling patterns
                have been defined

sounds (dir):   contains catching, bouncing and other noises

any directory   containing any looping capable ILBM 8SVX sound data
                (see
                jongl.prefs
                )


                jongl.prefs
                          Defaults


                list_of_objects
                      List of possible objects, defined line by line


                list_of_people
                       List of possible participants of the show


                list_of_sounds
                       List of samples assigned to function keys

FONTS:Jongl/6        for the texts while the program is running

some of the standard math libraries

iff.library     from Christian A. Weber (Aminet: util/libs/IFFLIB22.lha)


jonglengl.guide      That's what you are just reading, man!

for OS2.0 and OS3.1 in addition:
sys:prefs/presets (dir): has to contain the file 'LeererPointer'
                         (comes with the program)

ENVARC:sys/pointer.ilbm  (V2.04)
ENVARC:sys/pointer.prefs (V3.1)


The following files are not necessarily required but they
are really quite useful:

floor           to define the floor. An up-to-date list of all
                floors available see under "floor info".
```

```
loo             = "type list_of_objects"
```

1.10 Starting the program

```
                JONGL is a CLI programm. It may be started with 0 to 2
arguments:

> jongl          starts the program and shows the selection of all
                 available juggling patterns. In this pattern selection
                 menue you can move around with the mouse or the cursor
                 keys and you can select with the left mouse key or RETURN
                 or ENTER. If you don't want to select anything, press ESC.

> jongl -n       n represents a number between 0 and 45. The number
                 indicates what kind of objects are to be thrown. What
                 that means in detail, can be found in
                 list_of_objects
                 . That
                 is also the place to enter new objects.

                 When you enter a number outside this range, then a
                 selection from all presently available objects will be
                 shown, from which you can make your choice.

                 This option supersedes option 'a=' described further down
                 which defines the objects in the juggling pattern file.

                 Example: jongl -5
                 has the effect that all patterns selected later-on will be
                 performed with clubs.

> jongl -name    Same as above, however the object is called by name
                 instead of the number. This is the quickest way to test
                 new objects because the
                 List_of_objects
                  does not need to
                 be changed. 'name' means the file name of the object
                 definition.
                 Example: jongl -Messer
                      or
                           jongl -scissors  (if you define them)

> jongl x        x indicates the file within which the desired juggling
                 pattern is defined. This argument will skip the juggling
                 pattern menue. If the pattern starts with a figure, the
                 program will look for it in the file "m", if it does not
                 start with a figure, the program will read it as a path.

                 Examples: jongl 11_5r_pass
                           jongl ram:garbage

> jongl ?        shows the version number of the program and my address.
                 There is a program version for 68020/68881 and for 68000
```

each.

> jongl info shows the german jongl.guide from which you can switch to this jonglengl.guide.

The first two options may also be entered simultaneously, but only in this sequence: Example: jongl -2 5

Having successfully started the program you are quite rightly interested in
Controlling the program
.

1.11 Controlling the program

The program may be controlled by
Mouse
,
Keyboard
,
Menue
and
Joystick
.

In order to keep you going, a random selected sample will be played during the performance. The directory where these samples are filed, is defined in

jongl.prefs
. The samples have to be in the ILBM 8SVX format and have to be looping capable. Presently the program can handle up to 100 samples in this directory.

If you don't have such samples or if you want to hear a tracker or other background music, or if you just want to be left in peace, you don't enter the directory in
jongl.prefs
but no_sound and it is quiet from there on.

1.12 Mouse

With the mouse the position of the observer is being moved on concentric circles around the geometric center of all objects. As a consequence you view actions with high patterns from a higher position. (See option Z=).

Mouse left/right:	Observer moves left/right
Mouse forward/aft:	Observer moves up/down
Mouse forward/aft and	
- left mouse key	Observer moves toward/away from object

```
        - right mouse key       Observer's view angle increases/decreases
```

1.13 Keyboard

```
                Quite a lot of keys have been assigned to more or less useful ↩
                   functions.
(This program is written completely in german, including die mnemonics for
the keyboard. I'm thinking of a localized version with more sensible
shortcuts. Until then you have to use this keyboard layout.)

This is the list:

A:      Recording mode on/off. (ALL RECORDED) Useful especially for
        complicated patterns and for AMIGAs without turbo card.) A
        complete cycle of the file will be recorded and replayed at a
        noticeably higher speed. As soon as one cycle has been recorded
        completely, the accelerated presentation starts. The 'A' near the
        right screen margin indicates the recording mode.

        Changing the viewing angle with the mouse does not work during
        the recording mode.

        Speed adaptation (see G) is operative also in the recording mode.
        The pattern can be configured in such a way that it will be
        presented in real time.

B:      switches mouse control from rotation to LINEAR movement and back.
        You can move on tracks other than the concentric spheres, in all
        six directions of space:

                Mouse left/right:       Observer moves left/right
                Mouse forward/aft:      Observer moves up/down
                Mouse forward/aft and
                - left mouse key:       Observer moves forward/aft

                With a
                Joystick
                 in port 2 this control works the same way.

                Whenever you are lost, press 'R'.
                (One of the more essential inputs to remember.)

C:      Use the CPU instead of blitter for painting. In fast computers
        this will increase the picture repeat frequency. Because triple
        buffering is used in this mode, WaitTOF() is switched off at the
        same time (see N). This option may be switched on as a matter of
        routine, if the relevant parameter in
                jongl.prefs
                 is set to 1.

        If you want to get something out of this mode, you need to have
        something that's faster than an AMIGA 3000 (68030 / 25 MHz).

        More important: Some functions work ONLY in the CPU mode, because
        blitter is too slow for it  (and we are too lazy for double
```

programming...).

Most important: The CPU mode works only in the turbo version, otherwise no spheres could be seen.

E: Start single frame mode. The presentation stands still and will progress frame by frame whenever you press 'W'. If you wish to see informations or letters, these have to be activated beforehand. (Maybe I'll change that some day...)

W: Leave single frame mode.

F: FREQUENCY indication on/off: shows number of TOFs (approx. 1/50 sec) that have elapsed between two successive frames. The indication is to be read as follows:

```
+               50 frames per second (not highly probable)
++              25 "
+++             17 "
++++5++++0++     4 "
```

G: Initiate speed computation. (adjust speed). The program determines the time needed to present one cycle of the pattern and changes the parameters 'u=' and 'dt=' in the pattern file in such a way that the presentation will be done in real time. After that the pattern is loaded again taking into account the changed parameters.

Speed computation may also be initiated during the presentation of a pattern which was recorded with A.

H: make hardcopies. The current screen is dumped into a file which is named after the following scheme:
<dir><patternname>.#
<dir> is defined at
jongl.prefs
. It MUST end with a : or a /
<patternname> is the name of the current pattern
is the number of the save. Just to avoid deleting existing pics.

I: simulates infrared vision equipment. The sun is switched off (automatically) and a blurred and strange-coloured picture is shown. Works only in the CPU mode which works only in the turbo version. Very funny in connection with Z and V. Lots of activity in the presentation will improve the effect.

K: vomiting mode. You are flying 50 cm (1.6404199 feet for you non-Europeans) behind an object chosen at random. Why this is called the vomiting mode, will become apparent latest during the sharp turns while the object is in hand. Works only in the turbo version.

L: wire frame mode. Filled-in polygons are replaced by wire frames and vice versa. A rather boring feature for slow machines.

M:

Menue

on or off. As in real life the menue is redundant. It only serves to memorize the key shortcuts more quickly. To be more precise: Not all the key shortcuts are shown in the menue. That would have been tooo confusing.

N: switches WaitTOF() on or off. (NO WaitTOF) This enables you to observe how long it takes the program to fill the screen, because switch-over to the next frame is not delayed till the electronic beam has reached the bottom of the screen but occurs as soon as the computation of the frame is complete. In the CPU mode (C) WaitTOF() is off automatically.

O: allows to move the sun in azimuth and elevation, in other words, the direction and the angle above the horizon from which the sun is shining. This will change direction and shape of the shadow (see S).

```
Mouse left/right:       sun moves anticlockwise/clockwise
Mouse forward/aft:      sun sinks/rises
```

P: Present POSITION of the observer (B=..) and height of the observer (Z=..) is inserted in the pattern file. When the program is started again, the performance will be seen with the same viewing angle. (Explanation of the parameters see further down, under 'pattern files'.) After insertion of the parameters the program continues to run.

R: RESET for the viewing angle (if you have messed around too much with the above functions...) and for the center (see B).

S: Switch for the SHADING/SHADOW modes:

```
0: -- normal
1: -- objects shaded (if they have been defined like that)
2: -- everything casts shadows (see 'O')
```

This works only if four bitplanes have been defined in

```
jongl.prefs
```

. Which mode will be activated when the program is started, can also be defined in

```
jongl.prefs
```

. How to define the objects in a way that they will be shown correctly, is explained among others in

```
Flags
```

.

T: TIME indication on or off. Two clocks with second hands show the relationship between program time (P:) and real time (E:)

U: Cyclic switch for the status line at the bottom of the screen (UNDERLINE):

```
-- Samples assigned to function keys
-- First line of commentary of the presently shown pattern
   and indication of memory remaining
-- Nothing
```

Ü: ([) shows all kinds of superfluous values. Useful mainly for

debugging - but looks enormously important!

V: keep an eye on one object while the position of the observer remains unchanged. The object is first chosen at random but may be changed step by step with TAB.

In addition feel free to change the observer's position using the mouse... until you are lost!

Or you may, before starting this mode, press a key on the numeric pad and have one of the jugglers keep an eye on one of the objects.

But again, this works only in the turbo version.

Y: changes objects into letters or characters or vice versa. Useful for debugging. (Will be explained later.)

Z: fisheye. There is not much to be said about that. Just play around a bit with zoom and radius. If you still feel well, you may in addition press '1' (on the numeric pad) and 'V'. If that's not enough, press also 'I'. Well...?

This essential function works only with the turbo version.

(Remember that the program assumes a german keyboard. Therefore the Y and Z keys are exchanges in the HELP page.)

1...9,0 on the main keyboard	puts juggler number n into the center. The observer may then wander around each of the jugglers while they are passing. Switch off with 'R' or 'B'.
1...9,0 on the numeric pad	beams you into the head of the selected juggler. This will prepare you for the feeling that you will have when juggling 15 chain saws all by yourself...
TAB:	switches from object to object while "keeping an eye on" it. (See 'V')
F1...F10:	any samples for background noise of any kind. (To see the present function keys layout press 'U'.) These samples are defined in the file 'list_of_sounds'.
+:	Delay++ increases time delay between two frames
-:	Delay-- decreases time delay between two frames
[on the NKP:	Reduces volume of background music
] on the NKP:	Increases volume of background music
SPACE:	stops the motion of the observer. Makes it easier to stop the observer during a three-dimensional manoeuvre quickly and at the desired position.
HELP:	shows what all the keys are supposed to do for you

```
RETURN          returns to the pattern menue, so you can select a
or ENTER        different pattern without having to leave the program

ESC:            enables you to get out of this program as fast as possible


It is up to the user to find out how many of these modes he can handle
simultaneously.

(NKP means numeric keypad)
```

1.14 Menue

```
                If the Menue has not been activated as standard in
                jongl.prefs
                , then it is
switched on with 'M'.

Since the mouse is already busy with motion in space, the menue is best
controlled with the cursor keys. A selected menue item is activated
with RETURN or ENTER.

The menue slows the presentation down considerably. But after all you want
to read the menue rather than admiring the presentation. However, if you
want to do that, just switch off the menue.
```

1.15 Joystick

```
                With the joystick in port 2 the observer can move in straight  ↩
                   lines.
Control is the same as with the
                Mouse
                 and with 'B' pressed on the

                Keyboard
                .
```

1.16 Format of the file read in

```
                All files are in the ASCII format. Any amount of space (signs) may ↩
                    be
included. Even empty lines are allowed (almost) everywhere, except
within the definition of the pattern (after the *, see further down).

In this context we distinguish between
                 Object files
```

```
                and

                Pattern files
```

1.17 Object files

```
               Object files are found in the subdirectory 'o' of the current ↩
                  directory.
These are the objects which may be thrown around. There is also another
subdirectory called 'o/basic' which defines the static objects.

Beispiel: o/KeuleLight (Keule = club)

.....................

 r=1

 *

 8 POINTS

 0    0     -.35     #0

 0    0     -.06     #1

 -.06 -.052  .1
  .06 -.052  .1      #2-4
 0     .104  .1

 -.03 -.026  .25
  .03 -.026  .25     #5-7
 0     .052  .25

 10 LINES

 (7 4)   0 1
 (4 4)   1 2
 (4 4)   2 5
 (4 4)   1 3
 (4 4)   3 6
 (4 4)   1 4
 (4 4)   4 7
 (4 4)   5 7
 (4 4)   7 6
 (4 4)   6 5

The club is defined in a vertical attitude with the handle in the
-Z direction.

.....................

The file is divided into two logical blocks. The first one is optional
and may contain one ore more of the following parameters:
```

h=... This is the number of points per hand. This value has to be defined for a juggler. Otherwise it must not appear.

k=... The same, but for the number of points of the head.

r=... This parameter is of interest only for objects flying around, as it defines the
Rotation
. r=0 means that the object does rotate at all (like the potted plant), r=1 means that the object rotates like a club.

The dividing character between the two blocks is a *.

Next are at least two of the following sections:

POINTs
LINEs
TRINANGLEs
POLYGONs
SPHEREs

Sections which are not relevant need not be listed. Also the plural for single object parts is not required. That means you may write as well "1 TRIANGLE".

The round brackets within the definitions must not be left out. Their purpose is to aid in the orientation within the program in future versions. (Sounds good, doesn't it?)

Now we get to the details:

Points

Lines

Triangles

Polygons

Spheres

Flags

Rotation

Colour

Object comments

1.18 Points

POINTS are entered in 3D coordinates in the usual sequence x,y,z. ↩
Of
course we are working with a right hand coordinate system which means
X to the right, Y forward (into the screen) and Z upward.

If you want to see the axes you can select a floor with axes by entering
"FLOOR 1" before starting JONGL.

All values are given in units of METRES. (Again for you non-Europeans,
one metre is equal to 3.2808398 feet.) Objects which fly like clubs have
to be aligned with their handle in the -Z direction. The program rotates
them by 90\textdegree{} so that the hand holds them horizontally.

In the following, we refer back to the points. Therefore you have to know
that numbering starts with 0. So 13 points run from 0 to 12.

Now we come to
Lines
.

1.19 Lines

As the next step LINES are defined. Each line is defined by its
Colour
and its
Flags
. As a consequence, each line can have a different colour.
Colour and flags are put in brackets. This will ensure that the files do
not have to be changed if the object format is to be expanded later-on.

Starting and end points of lines: In the example above, the first line
has the colour 7 and extends between points no. 0 and 1. The next line
starts at no. 1 and ends at no. 2; it has the colour 4.

```
(7 4) 0 1
(4 4) 1 2
```

Both lines have the flag value 4, that means bit 2 is set. In other words,
they can be presented "shaded" if required.

We come to
Triangles
.

1.20 Triangles

TRIANGLES are defined as follows:
Colour

```
                 on the front side (see under
Flags), Colour on the rear side,
                Flags
                . Finally the three corner points
of the triangle.

Example:

(1 2 7) 5 6 7

This triangle has colour 1 on the front side and colour 2 on the rear
side. The flags are showing 7. So the bits 0, 1 and 2 are set. The
triangle is therefore filled-in, visible from both sides and may be shaded
in colour A.

The three points of the triangle are 5, 6 and 7.

Now let's look at
                Polygons
                .
```

1.21 Polygons

```
                Also for POLYGONS we have first the three parameters "front side
                Colour
                "
"rear side colour" and
                Flags
                , which have been explained already under
TRIANGLES. Next come the corner points of the polygon. Please note that
the first point has to be put in again as the last point.

Example:

(4 5 3)   11 6 7 8 9 11    (Colours 4 & 5; Flag 3; both sides filled in,
                           the polygon has 5 corners.)

An important restriction for a correct presentation is that only convex
polygons can be used. In other words, there cannot be any corners looking
inward. Also the polygon has to be in one plane.

Funny effects can be achieved by defining polygons with holes. For this
purpose a minus sign (-) and the point numbers are added to the normal
definition.

Example:

(4 5 3) 11 6 7 8 9 11 - 21 16 17 21

We see that the hole may have a different number of points, but it does
not have to. However the hole has to have the same sense of rotation as
the polygon!

I feel sorry for all CPU non-users. Unfortunately holes can only be seen
```

in the CPU mode. Too bad...

If you have understood everything up to now, you are cleared to get into

Spheres
.

1.22 Spheres

A SPHERE is defined by four parameters:
Colour
,
Flags
, point number,
radius.

Example:
(4 0) 17 0.06 (Colour 4, flags=0, point number 17, radius 0.06 m)

For the colour we have a special feature: As a return for -1 we get a colour chosen at random. (See also in the object file 'o/bunterBall'.)

The point number is the number of the
Points
 which defines the centre
of the sphere.

Well, and here is the bad news: Colour selection works only in the CPU mode, that is the turbo mode. Using the blitter, spheres usually come in standard red.

1.23 Flags

FLAGS are defined as follows:
bit 0 = 0: wire frame
bit 0 = 1: filled-in polygons

bit 1 = 0: only frontside visible (see above)
bit 1 = 1: both sides visible (but maybe with different colours)

bits 2 and 3 = 00: normal colours
= 01: both sides shaded in colour A
= 10: both sides shaded in colour B
= 11: For triangles and polygons, the colour is determined by the sense of rotation of the corner points:
clockwise - front side - colour A
anticlockwise - rear side - colour B
To change sides for the colours, simply change the sense of rotation.

```
                    For lines and spheres it depends on the colour
                    whether shading will be done with colour A or
                    colour B. For odd colour numbers we get colour A,
                    for even colour numbers colour B.

Now, how does shading work?

The flying object becomes darker if it flies into the background. There
are only two
                Colour
                s in four steps of brightness each: Colours A and B
(would you have guessed that?). The colours A and B can be defined in

                jongl.prefs
                .

Bits 0 and 1 make sense only for triangles and polygons;
bits 2 and 3 are valid also for lines and spheres.

Important: As with shadows, at least four bitplanes are required for
           shading, which can be defined in
                jongl.prefs
                .
```

1.24 Rotation

```
                ROTATION of the object can assume the following values:

  0: it does not rotate
  1: it rotates
(-1: it is human: it bends its head and moves its arms)

People - or, more generally speaking, objects (or should I say subjects?)
- who/which throw other objects around MUST be built according to the
following sequence, because this is the basis for the calculation of hand
and head motions.

The first 'h=...' points are the left hand. The next 'h=...' points are
the right hand.
Next are the 'k=...' points which define the head.
('h=' und 'k=' are explained in
                Object files
                .)

The first two points of the head define the axis of rotation. If the
juggler bends his head forward, then these two points have to be
exchanged.
```

1.25 Colour

```
                The program uses 16 COLOURS. This is the minimum requirement for ↩
                   all
functions to work. If fewer bitplanes are used, the colour palette will
be that much shorter.

Colour 0 is the background which is more or less black most of the time.
Colour 1: yellow
Colour 2: beige (skin colour)
Colour 3: dark green
Colour 4: light red
Colour 5: pink
Colour 6: light blue
Colour 7: light grey

Colours 0...7 cannot be changed.

Colour 8:  white
Colour 9:  orange
Colour 10: brown
Colour 11: light green
Colour 12: dark red
Colour 13: purple
Colour 14: dark blue
Colour 15: dark grey

Colours 8 to 15 may change. By pressing 'S' on the
                Keyboard
                 we get into
two further colour modes:

shaded:

In this mode, colours 8 to 15 are redefined with four shades each of the
colours "ColourA" and "ColourB" which can be defined in
                jongl.prefs
                .

with shadow:

Also in this mode, colours 8 to 15 will change. Here they will become the
same colours as the first eight, however with half the brightness. As you
might have guessed, these colours are needed for shadows.

(The completely useless infrared vision mode (I) is not even mentioned here.)
```

1.26 Object comments

```
COMMENTS in object files may presently be inserted in three places:

a) in the first part (before the *), if the comment starts with an !,
   like in the pattern file which I am going to explain in a minute,

b) behind the 3D coordinates until the line is full,
```

c) behind the last line of the object, and there is no limit...

1.27 Pattern files

A PATTERN FILE defines the pattern to be juggled. Its contents are ↩
thus
the basics of the input.

First we define various mandatory and optional parameters, then the
positions of the people and finally the pattern.

It is IMPORTANT that spaces within one parameter are not allowed. Instead
of 'dt = 0.234' you must write 'dt=0.234'.

In principle you may list the parameters in any odd sequence; the only
exception to this "rule" is that 'h=' has to be defined before 'o='.

However I would like to introduce the convention that every pattern name
starts with the number of objects and that it is divided into reasonable
sections using underscores.
The name of the pattern should not be chosen too long; otherwise you
might see overlaps in the pattern selection menue.

Pattern files can thus be compiled from the following components:

Mandatory parameters

Variable parameters with defaults

Optional parameters

Pattern comments

Positions of the jugglers

Pattern definition

Bouncing

Multiplex and wrap around

1.28 Mandatory parameters

The following three parameters have to be included in every pattern file:

```
h=4     Number of hands
o=7     Number of objects
t=22    Number of phases (= number of lines) in the pattern definiton
```

1.29 Variable parameters with defaults

With these parameters you can influence the happenings. But it is not necessary to adjust them as they have already reasonable initial values.

a=6 Type of objects to be thrown around. (Numbering see above under 'jongl -n'.)

B=1.0,1.0,3.5,185.0 With this parameter you can define the position of the observer. The values describe the viewing angle from which you look at the scene
If you have interactively achieved a nice viewing angle, just press 'P' and the above values are entered into the pattern file.

dt=.25 Time elapsed between two lines of the pattern definition. With this parameter you can adjust the height of the throws, while the program is not yet able to do it itself.

eps=.95 Reflection quality of bouncing balls, standard is 5 % loss. (important: eps<0) We refer here to the loss of velocity during reflection. Height loss goes with the square of loss of velocity, as the potential energy m * g * h is equal to the kinetic energy (m * v squared)/2. If the cinetic energy is reduced by 5 %, then the ball reaches only 0.95 squared = 0.9025 of its initial height. So if you want the ball to spring back to half its initial height, eps is equal to the square root of 0.5 = 0.707.

g=9.81 Gravity constant in metres per second sqared. (Equivalent to 32.2 feet per second squared which may sound more familiar to you.) Positive values relate to acceleration downward.

l=1.2 Longer keep-in-hand times. According to the basic design of this program, the juggler would never hold an object in each of his hands at any one time but only alternating left and right. This is not so in reality. Most of the time you have all your hands full and only then do you throw something away, when you have to catch something shortly afterwards. In order to bring a bit of reality into this simulation program, the duration of the keep-in-hand time can be adjusted. Default values are catching 20 % earlier and throwing 20 % later (which logically adds up to a total of 40 %).

q=0 Quick presentation. Normally off, if needed set q=1 (=on). Accelerates the graphics during the times of high life on the screen. (To be more precise: People become juggling coat hangers and all that's left of the tile floor is its edge.) If further acceleration is needed, the flying objects can be reduced to just a few points and lines.

s=.15 Scatter of the throwing and catching motions. To make the juggling performance look more human (that is more imperfect), the motions vary within a cube with a side length of '2*s' around the proper catching and throwing position.

u=5 Frames per phase. To make the motions look more easy flowing, every phase defined in the pattern (see below) is divided into

several frames. Here is where you define the number of frames. The value depends on the processor and on the complexity of the scene. Too many frames will result in a slow motion movie. Hopefully the program will be able to decide on the number of frames per phase all by itself some day...

Z=1.7 With this parameter you can adjust the height of the observer above the floor. If you enter z=0.0, then the 'eye' will be in the centre of that part of space that is occupied by static, throwing, flying, catching objects... Sorry, there is an explanation more simple than that: This parameter defines the height of the centre of the sphere on which the observer moves around. This centre can be changed to any value while the program runs, by pressing 'B'.

The values given in these examples are the default values in the program.

1.30 Optional parameters

These are parameters without default values, because there are no reasonable default values.

Different objects in one pattern
for juggling experts

Hand macros
for different hand positions

Additional static objects
for miscellaneous junk

1.31 Different objects in one pattern

O=5 1#2 2#6 2#4 This parameter replaces both 'o=' and 'a='. It allows you to get a mixture of objects into the air. In the above example we have five objects, in fact once no. 2 (ring) (see above under 'jongl -n'), followed by twice no. 6 (ball) and twice no. 4 (club). The sequence of objects is determined by the sequence of their appearance in the pattern definition, as the following example shows:

```
-a
d-
-b
e-
-c
....
```

In this case the object with the letter 'a' would be a ring, 'd' and 'b'

would be balls and 'e' and 'c' would be clubs. You see, the program
takes the objects in the sequence as they appear in the pattern file,
not by alphabet or by the value of the figures.

1.32 Hand macros

With hand macros, different hand positions can be defined for single
throws.

Example:

```
@A p 0 .5 1
```

or

```
@B P 0 .6 1  0 .3 1
```

The letter @ is the leading letter for a macro definition of the hand
position. (Just remember: @ = "at" indicates the position.)

The letter immediately behind @ is the command for calling the macro up.
I recommend to use upper case letters as opposed to the lower case letters
normally used for the pattern.

The following letter is either a small or a capital p. Then follows, as
described further down, the definition of the hand position (no matter,
whether left or right). That means we can define separately a catching
position and a throwing position.

IMPORTANT: The positions of the juggler as defined below have to be
added nevertheless.

According to the basic program each hand always repeats the same motion,
regardless where it throws what. This command provides different hand
motions for different throws, e.g. Splits or Tennis or Boston Mess.

Example: '4_Spagat' which would be '4_Splits' in english

```
h=2 o=4 t=4 dt=.22 u=6 s=.1
p 0
0  .6 1        %  Here we define the outer positions
0 -.6 1
*
1 A2
-  -
B3 4
-  -
```

With some amount of skill and patience you can even achieve lead tricks...

1.33 Additional static objects

```
It is possible to define additional objects which do not move:

z=2
o/basic/Bierkiste
  90
  -2 0 0
o/basic/Giraffe
  0
  2.1 0 0

Behind 'z=' we fill in how many objects we want, then comes the path for
the file to be read in (in object format as described above). Next is the
rotation around the z axis in degrees which allows us to adjust the object
as desired, and finally the x,y,z coordinates where the object is located.

Presently static objects can only be defined after the positions (p or P)
of the people have been defined!

As mentioned earlier, the objects haven't been translated.
```

1.34 Pattern comments

```
Two types of COMMENTS may be included in pattern files: those which will
be shown on the screen and those which are only comments to the program.

! Anything behind an exclamation mark will be shown when the program runs.

% Comments behind a percent sign only serve for orientation in the
% program and will not appear during the performance.

Comments have to end at the end of a line, but they do not have to start
at the beginning of a line. In the pattern definition they must be to the
right of the definition. Example see further down.
```

1.35 Positions of the jugglers

```
                There are two possibilities to define the POSITIONS of the  ↩
                   jugglers: 'p'
and 'P'. The small 'p' is the simpler version in which the hands are
catching and throwing at the same position. With 'P' different catching
and throwing positions can be defined, which allows for instance to dis-
tinguish between cascade and backward cascade.

The next parameters after 'p' or 'P' are the rotation angle about the
Z axis (in degrees) and then the 3D coordinates of the position. With a
rotation angle of 0\textdegree{} the juggler looks straight ahead in the +X  ↩
   direction,
+Y is to the left and +Z is upward. With 90\textdegree{} the juggler looks in the  ↩
   +Y
direction.
```

```
Please note that the mean value of the Z coordinates of the hands of  each
juggler  should  be  as  close to 1 as possible, because that is where the
program puts them. Furthermore the body will be put in the middle  between
left  and  right hand. This will cause problems if you want, for instance,
to define "Splits". In this case you define either  both  inner  hands  or
both outer hands normally and the other two with '@' (see further above).

The number of people is calculated by the program from the value 'h='.

Examples:

p 0
  -1  .5 1
  -1 -.5 1
*

P 0
  -1  .5 1.2  -1  .2 .8
  -1 -.5 1.2  -1 -.2 .8
  180
   1 -.5 1.2   1 -.2 .8
   1  .5 1.2   1  .2 .8
*

As you see, the definition of the positions is concluded with an '*'.

Now we come to the
                Pattern definition
                .
```

1.36 Pattern definition

```
The pattern is defined in exactly as many columns as hands are involved.
You may include as many spaces between the letters as you wish, but
NO EMPTY LINES AND NO LINES WITH COMMENTS ONLY ARE ALLOWED!

A simple example is one juggler doing a cascade with three balls:

 a-  ! left hand throws object 'a'
 -2
 E-
 -a  % right hand catches 'a'
 2-
 -E

The dash '-' means that at this point in time the relevant hand is empty.
The columns refer from left to right alternating to the left and right
hand(s). (Important for writing passing patterns!)

You may choose almost any letter for the objects; all letters except '-', '&',
'!', '%', '*', '/' and '\' are allowed. That's why the example looks so
weird. However I recommend for use in patterns the sequence 1,2,3,4,5,6,7,
8,9,a,b,c,d,e,f,g... Capital letters should be saved for macros (@), to
maintain a bit of transparency. Also I think it is a good idea not to use
```

```
up all the odd letters; we may need them for any later program expansions...

You see also examples how to use comments. The program will not show
comments which are right of the pattern definition.
```

1.37 Bouncing

```
In real life there are two types of BOUNCING: throwing the balls upward
(the proper "bouncing") or throwing the balls downward ("force bouncing").
Both types are covered by this program. It is also possible to have the
ball bounce more than once before catching it again.

For bouncing we have to introduce by two additional letters, '/' and '\'.

Throwing upward: '/'.
Example:
 - 1
  /2-
  - 3
  1 -
  - 2
  3 -

In this case object 2 is thrown upward for one bounce.

The notation for throwing downward (force bouncing) is: '\'.
Multiple bounces are noted by multiplying the relevant letters.
Example: Bounce shower:
  1-
  -\\1
  2-
  -\\2
  3-
  -\\3

Every ball is thrown downward and bounces twice.

IMPORTANT: BOUNCING DOES NOT WORK WITH ANY ARBITRARY PARAMETERS!
The program will complain if it is not possible to calculate suitable
initial velocities with the inputs given.

More important note:
AmigaGuide and MultiView disagree in their interpretation of '\'. Multi-
View considers the backslash as ESCAPE command, therefore you have to
write two backslashes in order to get one indicated. Not so AmigaGuide.
Therefore watch it, AmigaGuide users: In the example with the bounce
shower, we have of course only two bounces per line.

Always remember: german "dotz" means "bounce".
```

1.38 Multiplex and wrap around

Here we have a more complicated example with some new commands:
'm/8_MPlex_Pass'

```
 *
   8 1&2   - 3
   8   -   5 3&4
 7&8   6   5 -
   -   6 5&1 2
   3 4&6   - 2
   3   -   7 2&8
 5&3   1   7 -
   -   1 4&7 6
 *
 12345678
 18475362
```

You may notice between one and two new features:

a) Multiplex. With the letter '&' you can hold more than one object at a time in one hand. To be more precise: as many as you want. You may catch them at different times and throw them away together (which is realistic) or not (which is unrealistic).

b) Wrap around. This is shorthand writing for patterns which look as if they repeat themselves after a short time, but then they have different object numbers. Understood? Well, take passing in four-count (pass every second right hand one). If you write down how long it takes until all the objects are again where they started, then you realize that you have to do a lot of writing! It is much easier with wrap around:

```
 *
 -1 -4
 2- 5-
 -3 -6
 4- 1-
 -2 -5
 3- 6-
 -4 -1
 5- 2-
 *
 123456
 561234
```

In detail:

Behind the pattern definition we have another '*'. The next line lists all the objects used in the pattern in any arbitrary sequence. The line below shows the replacements of the relevant objects. Let's look at the example: In the first line after the lower '*' the first figure is 1. The figure below that is 5. This means as soon as object number 1 drops out of the bottom of the pattern file, it comes back in at the top of the file as number 5. Object number 5 will be replaced by object number 3, and so on. This wrap-around trick saves ages of typing work. You only need to note each throw at least once and then concentrate on the wrap-around table.

```
See also '10_H-Mult_dotz' with the notation '/4&R9' and think about it!
```

1.39 Listings of external files

```
                Here comes a list of external files (which was up-to-date at some  ↩
                   point
in time way back) which is needed by JONGL:

*********************************************************************
Was ist daran jetzt nicht mehr aktuell? Der Inhalt oder die Namen der
Listen oder was? Bitte Klärung und/oder entsprechende Änderung des
Textes!
*********************************************************************


                jongl.prefs
                 What is your pleasure, sir?


                list_of_people
                 Choose your favorite juggler


                list_of_objects
                 All kinds of throwable parts


                list_of_sounds
                 Everything but silence
```

1.40 jongl.prefs

```
                ............................................................... ↩

 4   : Number of bitplanes
 0   : Reserve memory for Recording Mode
 0   : Move bottom line up by so many bits
 25  : Buffer factor
 0   : CPU mode on
 0   : Shade/shadow (0:none; 1:shaded=background darker; 2: with shadow)
 diamond.font 12 : Font and size for pattern selection menue
 no_sound        : No background music
 ! DH0:Audio/Sounds  : Background music (or whatever noise)
 f00 : ColourA
 ff0 : ColourB
 1   : Menue on
 N   : overscan (N: not   H: horizontal   V: vertical   B: both)
 RAM: : path for hardcopy saving
 deutsch:  : language
...............................................................
```

Only the data in front of the ':' are relevant. The comments behind
are only meant to confuse the user.

Short explanation:

Number of bitplanes: Values between 1 and 4 are reasonable at this time.
The more colour the slower! But for shades and shadows four bitplanes are
necessary.

Reserve memory for Recording Mode: If not enough memory is left, enter '0'
to switch off the memory requirement for the Recording Mode.

Bottom line up so many bits: If the bottom line, after being switched on
with 'U' should not be visible (I have no idea why that can happen), it
may be moved up by one bit or the other. In case of emergency see

Keyboard
.

Buffer factor: is a factor which makes sure that all the objects find
enough memory. The optimum size of this factor is best found by trial and
error. If many complex objects are flying around, then this factor has to
be increased. But the program will request this if necessary. When you run
short of memory, you may reduce this buffer. However the consequence may
be that the program cannot handle complex objects anymore.

CPU Mode on: Set to 1, if you want to start the program in the CPU Mode.
(See 'C' on the
Keyboard
and in {"Menue" Link 3.1.3.}).

Shade & shadows: Initial value for shade and shadow.
0 means neither shade nor shadow
1 means shaded objects which become darker towards the background. This
option works only if the objects have been prepared accordingly, see

Flags
.
2 activates the shadows

Font: Here you put in the font that you want to see in the pattern
selection menue. Don't forget to add ".font" and the size of the font! If you
like you can add the number of columns in this menue. The default value is 5.

Background music: The program needs the absolute path to the
directory where the looping capable background music samples are hidden.
The path may end with ':', but not with '/'. If your input is no_sound
, then the program leaves its fingers off the audio channels - so you
can activate any trackers or stuff like that to pollute your acoustic
environment...

ColourA and ColourB are the
Colour
s, that are used for shading. Details
of this subject can be found under
Flags
. You were right in suspecting

```
that hexadecimal notation is used. f00 is red, ff0 is yellow, 0f0 is
green etc.

Menue on: If you put in '1', then the menue is activated right from
the start. But while the program is running you can still switch the menue
on and off at any time.

Overscan: Switch it on, if you do not want to see black margins of the
screen. Switch it off, if life appears too slow.

path for hardcopy saving: Here you enter the directory (ending with
: or /) in which you want to save the hardcopies (see H)

language: I suppose you would want to change this into "english".


If you want to skip one line, write ! in front of it. But the parameter
which would have been defined in this line has to be included anyway.
Sounds funny but there is a reason for it: You may have different versions
of some lines, but you do not have to delete them. One application is the
background music, see above.

If the program does not find 'jongl.prefs', then the standard values
listed in the above example will be used and Jongl will work nevertheless.
```

1.41 list_of_people

```
.............................................
 6
 Skelett
 Peter
 Jongleuse_mit_Rock
 Blondine
 blauerJongleur
 rotgelberJongleur
 Roboter_B
.............................................

The first line shows the number of jugglers. The next lines contain one
file name each from the directory 'o/basic'.

There is a way to manipulate the program: If you don't want to see some
types of jugglers, move them to the end of the list and reduce the value
in the first line such that only those people can be selected by the
program who are welcome to the user.
```

1.42 list_of_objects

```
                An example for a list of objects is this:
.............................................
 0:  - Würfel                    Cube
```

```
 1:  S Flagge                    Flag
 2:  S Ring                      Ring
 3:  S DoppelRing                DoubleRing

 ...

 41: S HgrünerBall   )           LgreenBall
 42: S DroterBall    )           DredBall
 43: S lilaBall      )           purpleBall
 44: S DblauerBall   )           DblueBall
 45: S DgrauerBall   )           DgreyBall
     ^--- S heißt SCHATTIERT      --------- S means SHADED
.................................................
```

The figures indicate line numbers. The texts are the file names of the objects in the directory 'o'. The S in the second column is just a note for the user to remind him that the object has been defined in such a way that it can be shown shaded. (See also
Flags
). Since the program assumes
that the third string in each line is the file name, it is important
a) in the case of no S to insert any other letter (e.g. -),
b) not to use any spaces in the object names.

This list is just an example. If you want to see the real list, enter 'LOO' or - if that should not work - TYPE LIST_OF_OBJECTS.

1.43 list_of_sounds

```
.................................................
 9
 FANG
 DOTZ
 Yell
 Audience_yell
 Explosion
 YouSee
 Rubbernose
 Hohoho
 Jingle
.................................................
```

Up to 2+10 sounds may be defined. The first two sounds are used for CATCHing and BOUNCing. The next up to ten sounds are assigned to the function keys F1 to F10. for (c) reasons, sounds are not included in the program.

1.44 Additional programs

```
                The possib- and capab-ilities of the already extremely cool JONGL  ↩
                   program
can be exhausted up to the extreme, if the following additional programs
are used:


                Freestyle
                 creates random patterns


                J2
                 creates systematically all possible
                        patterns according to inputs


                J2Konv
                 makes the J2 output JONGL-readable


                Object_Converter
                 is not yet available, but would be
                        great
```

1.45 Freestyle

```
                As you might have guessed from the name, this is a tool to create  ↩
                   free-
style juggling patterns.

To start the program simply type 'freestyle' and answer the questions.
The program will then create a pattern file which may of course be refined
afterwards by hand using an editor.

Here you should not take too many objects (one or two more than hands) but
rather long periods, say 60 or 80, if your memory serves you adequately.

Not designed by myself, but nevertheless good is
                J2
               .
```

1.46 J2

```
                If the expression "site swap notation" does not ring a bell with  ↩
                   you, I am
afraid you will not understand this and the next paragraph. In this case
I refer you the relevant documentation, which is j2.guide and
notation.guide.

But since you are reading on, you must be convinced that you can handle
```

```
site swap. Here we go:

j2 calculates all possible juggling patterns
        *  for a given number of objects
        *  with a maximum throwing height
        *  and a given period.

To press the output of j2 into JONGL, I have written
                j2konv
                .

J2 is (c) by Jack Boyce. Many thanks for this fine program!
```

1.47 J2Konv

```
                This is a converter between site swap as produced by
                 J2
                 and the input
needed by JONGL. Possible parameters will be indicated after program
start, if you do not include them.

One word of warning: j2konv is not yet so extremely perfect...

This is the path to the Instruction of J2KONV.
```

1.48 Object_Converter

```
I'm sorry we don't have it yet. That means we have defined all the objects
by hand. If that is not your style, feel free to write a (more or less)
comfortable editor for our sensational objects.
```

1.49 AGA-Version

```
AGA would not be all that bad, but unfortunately none of us has a suitable
computer to call his own. So if anybody really insists on getting an AGA
version, he is kindly asked to send us an AMIGA with AGA chip set. We will
then adapt the program to his needs.

Many thanks in advance...
```

1.50 Miscellaneous

```
                As you would expect, the section "Miscellaneous" contains nothing ↩
                   but the
usual mess of incoherent items that would not fit in anywhere else,
```

namely:

1.51 Speed

Caution!

If you own a 68060 processor,
you may waste your time reading this!

For a simulation of fast moving objects the natural flow of the animation is very important. You may therefore want to read the following comments and tips relating to the speed of the program.

The presentation of a few hundred graphic elements on an AMIGA with 7 MHz clock frequency by the standard graphic routine from the operating system which is activated over a High Order Language (C) is simply too slow. Especially if you don't have an accelerated AMIGA. What looks still quite nice on the 50 MHz 68030 computer, turns to a maximum of five frames per second on the 68000, even for simple patterns (five balls) and wire frame shading. Since I (Martin of
The programmers
) had programmed other
funny things in assembler already before, we decided to put a bit of pep into the program with an assembler subroutine. After about five month's worth of programming the graphic assembler subroutine was more or less complete.

Summary of what you have read up to now: The graphic routines are programmed in assembler.

Now there are two versions of graphic output:

- over the graphic processor 'blitter' which is standard in every AMIGA, or

- the main processor does it all.

Generally the blitter is very fast; however on fast systems (from AMIGA 3000 with 25 MHz upward) the processor offers slight advantages. If you don't have at least a 68020 and a math coprocessor 68881, you can only use the blitter version anyway. Since the blitter can work parallel to the main processor, the speed of the blitter version depends not so much on the processor. On the other hand, the speed of the processor version -

from now on called CPU version - is proportional to the processor frequency. On our 68030 with 30 MHz the processor version is twice as fast as the blitter version.

The following table shows the influence of different parameters on the speed. It is all my estimate, based on experience, because I know the program (because I have written it).

+ = decelerates enormously (really brakes it down (great in fact))
o = has also some braking effect (like a parking brake - for comparison)
- = decelerates just a bit (but still, somehow)

Version	Blitter	CPU
Many points	+	o
Many polygons	o	+
Large areas	+	o
Number of bitplanes	+	-

You see, whatever is great has a braking effect on the speed. Only in the CPU mode it does not make sense to use less than 16 colours (four bitplanes); the achievable acceleration would be hardly noticeable.

What can we learn from this?

Here are some tips:

MANY POINTS: You can get rid of them by using simple objects. Of course you cannot improve on balls consisting of just one point.

MANY POLYGONS are found only in the more complex objects like the hat and the potted plant. Simple and unfilled objects are faster.

LARGE AREAS are found mainly on the floor. To avoid this, make the floor a wire frame. Shadows are not visible then but will still be computed. Therefore don't forget to switch off the shadow!

The NUMBER OF BITPLANES is defined by the first entry in
jongl.prefs
. Less
than the standard four planes prevent shades and shadows. Less than three bitplanes is not the real colour hit anymore and one bitplane looks really awful, colourwise.

Switching off the indications for frequency ('F') and time ('T') has some effect on slower computers. After all these are operating system routines which are not all that fast. If all else fails, there is still the possibility to enter the parameter 'q=1' in the pattern, which will convert (male and female) jugglers to (unisex) coathangers and the floor to a boring frame. If you then select as objects the mini club (-16), things get going! But this is only for speed freaks, as it looks completely stupid and boring. To be used only in emergency!

Last and not even most unimportant tip: The Recording Mode has been brought into being just for the purpose of presenting a speed adventure for those poor owners of slow computers. See also in
Keyboard
item A.

So much about this. Now go and play!

By the way, if you are looking for tips how to slow the performance down, why don't you try this:

Floor 11
Jongl -19 24_Brad_2Cnt

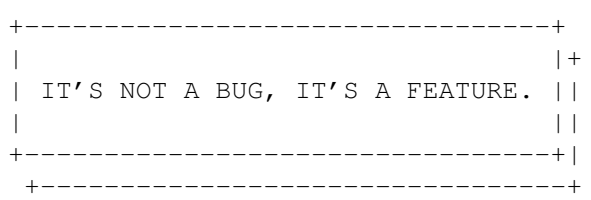
Now you see a Bradford Circle with eight people, juggling with potted plants on a floor with about 100 tiles. This adds up to approximately 1750 points and 450 areas or lines. On a 68030 with 50 MHz this runs with four frames per second (yawn). If you want to slow down other patterns, just press '-' once or a few times and you can easily follow whats going on.

1.52 Known errors

Sometimes the program starts as usual but then fails to build up the graphics. In this case simply press ESC and boot the computer again. Only rarely does this happen, and above all I have no idea why. Fate!

If the program stops at an unusual step due to a problem, in some rare cases not all the memory requested will be reserved.

Any surprising characteristics of this program are covered by the following statement:



Unfortunately I have not yet had enough time to document all the features.

1.53 Outlook

The next major project is the installation of real time presentation. Then you will always have the maximum frame frequency and can adjust the time for instance between 0 and 200 % of real time. Then at last it will be possible to show all patterns on all computers in real time right away.

1.54 Internet

In case anybody doesn't know yet: Juggling takes place also in the
Internet, under the URL http://www.juggling.org.

That is where this program can be downloaded from too.

1.55 De-Installation

First of all you should ask yourself why you want to get rid of the
program. There are several possibilities:

1. You have found that the program ist simply awful.
2. Your hard disk is overflowing already with many many
   very very useful programs.
3. You have installed the program out of curiosity, but you
   cannot juggle and therefore this high performance juggling
   simulation is of no use to you.

If the THIRD POSSIBILITY applies, this can be changed. There are many
jugglers in the world who meet weekly (for instance in connection with
university sports). In these meetings you will find lots of nice people
who are glad to show you how to get started. Dates and places of juggler
meetings can be found for instance in
Internet
in the Juggling
Information Service (JIS). A simple introduction to juggling can be found
in the Aminet under Docs/Help/JuggleGuide. And then there are juggling
conventions almost every month where jugglers from all over Germany /
Europe / the world come together.
If you don't have access to the Internet, you may also contact

The programmers
. Have fun in the world of juggling!

If you think the SECOND POSSIBILITY is true, maybe you want to check
whether all your programs are all that useful. Perhaps you can do without
one or the other. (Completely installed, JONGL occupies less than 600 KB.)

In case of the FIRST POSSIBLITY there is no other way than to either sell
your AMIGA (Don't even think of it!) or to delete the program. This is how
it works:

1. Delete the directory JONGL (Delete jongl all quiet)
2. Delete the pointers
   2.1 With Kickstart 1.3 there are none (lucky you!)
   2.2 With Kickstart 2.0 or 3.0 delete 'LeererPointer'
   in sys:Prefs/presets/
3. Delete Jongl Font from Fonts:
4. That was it!

1.56 Compensation

```
                Since several manyears of development work have gone into this ↩
                   program,
I don't expect that anybody is willing to pay an adequate price for it.

But if you have more than a short look at this program, please let

                The programmers
                 have one of the following:

        - an email
        - a letter
        - a (picture) post card
        - a disk

with comments, suggestions, descriptions of errors... and above all:
new patterns and objects.

THANK YOU!


If you have problems with the program or if you have found something
wrong, please don't forget to add the following information:

- Hardware used (Type of AMIGA, processor, exotic hardware if applicable)
- Operating system used (Once upon a time JONGL worked with OS 1.3)
- Version used (68000 or 68881)

Even if you don't find any error, we would be interested to hear/read
what kind of computer you are using. This would give us some indication
which functions should be expanded.
```