www.palmcrust.com

# A cross−number puzzle application

For Java$^{TM}$ (J2ME) mobile phones

Version 1.0

July 2003

Author Michael Glickman

# Table of Contents

# About XnX

**Ericsson, Motorola, Nokia, Siemens, Sony, Challenger, Handango are trade marks, or registered trade marks of their respective owners and used in this manual for identification only.**

## Introduction ... that someone might find worth reading.

XnX (Cross−Number matriX) a cross−number puzzle, similar to the Challenger® puzzle that appears in some newspapers, like San Francisco Chronicle... (??? − Please, let me complete the formalities first.) Accroding to Brian McBride, Challenger 'is a syndicated feature of King Features Syndicate, Inc. '.

You might ask how it happened that your humble servant, being among those few coders who still live far away from CA and can't get used to reading periodic literature online, knows something about SF Chronicle. To be honest, I wouldn't have any idea about it, if not ***Roderick Young***. It was my impression of his great PalmOS freeware named **X−num**, that triggered X#X development. Thank you Roderick, I owe you so much.. Please forgive me that some ideas (hints, puzzle code, etc) were shamelessly borrowed from X−num.

Another freeware, this time for PC, named **Challenger® Puzzle Solver** was written by ***Brian McBride***. Brian attempts to explain his algorithm, and I will be hapy to find someone who can clearly understand the explanations. To me the description appears more concentrated on Java objects, than on the algorithm itself. As far as I understood it, the algorithm checks all possible combinations in a non−recursive way.

On the contrary, X#X solver (see Appendix) attempts to avoid testing all combinations, but is recursive which is a disadvantage. Like probably most of you, I've read R. Sedgewick's book and know that each depth−first algorithm allows a non−recursive implementation, but I am out of time: X#X was not a part of my original plan, while there are plenty other applications and updates in the queue.

I regret that X#X is not a free application: after all we have to live on that. But it is going to cost nearly as much as an average lunch. An excellent chance to find out how many J2ME device owners enjoy the application so much that can miss their lunch in order to buy it :)

The development took about a fortnight (i.e. two US weeks). Most of development work has been done in Linux 2.4.18 using Java 2 SDK v 1.4.2 (beta) and J2ME WTK versions 1.0.4_01 and 2.0 by *Sun Microsystems Inc*, ProGuard v 1.6 by *Eric Lafortune*. I wish I could say *'All'* instead of *'Most of'*, but unfortunately Motorola, Siemen, Ericsson and most of Nokia tools need for testing wouldn't work anywhere apart from the 'smelly platform'.

## Rules

The puzzle represents a square table where most of cells are left blank, and the sums are specified for each row, column and both diagonals. To solve the puzzle you need to place a digit from 1 to 9 in each blank spot so that all sums match.

Stardard Challenger® puzzle has 4 rows and columns. This application allows to vary number of columns from 3 to a maximum number that fits the screen.

You can either generate a puzzle, or enter your own puzzle. A custom puzzle is checked for consistency (must have at least one solution). The application can bring up to 500 solutions of a puzzle (entered or generated). You may require a *single−solution puzzle* to be produced, in which case a minimum possible number of preset elements will be added to a generated puzzle.

An unregistered copy has limitations, viz. time limit, can't solve a custom puzzle.

## Compatibility

The following releases are currently available:

| | |
|---|---|
| **XnX_std** | Standard version with screen size 120x120 or more (e.g. Motorola T720, Motorola A830, Sony Ericsson T610), Can be used with Nokia® Seris 40 and 60 devices as well, but using special release is preferred. |
| **XnX_small** | Designed for devices with screen size below 120x120, but al least 96x64, e.g Siemens models. Can be used with a larger screen size to increase board size. Not recommended for Nokia series 30 − use special release. |
| **XnX_S40** | Developed for Nokia Series 40(screen size 128x128), Nokia 6650 (128x160) and Nokia Series 60 (176x208). Uses full screen. |
| **XnX_S30** | Developed for Nokia Series 30 (screen size 96x65) with at least 50KB memory for a midlet (e.g. 3410i, 3510i, 8910i) Uses full screen.. Can be used with a larger Nokia screen (series 40 or 60) to increase board size. |

## Links.

The application is developed by Michael Glickman for Palmcrust® (Australia)

| | |
|---|---|
| Application site: | http://xnx.palmcrust.com |
| Palmcrust® site: | http://www.palmcrust.com |
| Email: | palmcrust@yahoo.com |

The application can be purchased from:

| | |
|---|---|
| Handango® | http://www.handango.com |
| Nokia | http://softwaremarket.nokia.com |

# How to play

## Keys

**Command keys**
Two keys (*left* and *right*) located below the screen. Some Motorola and Nokia devices have also *middle* command key which is not used by the application. One of command keys labeled ***Menu*** or ***Options*** is used to bring up the list of actions, where you can select an action using DOWN and UP keys and press FIRE to implement it. Another command key, referred as ***Back*** key acts as Pause, Exit, Dismiss or Done. Often *Back* is also used to dismiss the list of action without executing a command.

**Nokia Series 30**: a small screen size does not allow showing labels,  use left key for Menu, and right key for Back.
**Siemens**: no Back key provided − use right command key to get the list of *all* actions, to dismiss the list use **END** (red) key,
**Sony Ericsson P800:** *Menu* is not a key, but a menu bar item

**FIRE key**
The implementation if FIRE is device specific. Here are some indications:

**Motorola, Nokia Series 30 or 40**: **SEND** (green) key
**Nokia Series 60, Sony Ericsson**: centre of the arrow pad, or joystick press
**Siemens**: left command key

**Arrow keys**
UP, DOWN, LEFT and RIGHT arrows, or moving joystick
Where arrow keys are unavailable (e.g. LEFT and RIGHT with Nokia Series 30), you can use corresponding numeric keys instead.

**Numeric keys**
1 2 3 4 5 6 7 8 9

**Phone keys**
Numeric keys, * and #.

**CLEAR key**
Some devices have a key marked as CLEAR or C. Use of the key is optional.

**A, B, C, D keys**
Game keys provided by Sony Ericsson P800.

# Main screen

The main screen show application logo. To get the list of available actions, press *Menu* or *Options* command key, to quit the application press *Exit* command key. The command keys are often located below the screen. A small screen size does not allow showing command key labels − just remeber that Menu is the left, and Exit is the right command key.

The list of available actions is given below. Some actions have *shortcuts* shown in square brackets [.] In this case you can just press a phone key to activate corresponding action.

**New Puzzle [5 or FIRE]**
Generate a random puzzle.. This is what you probably need most often.

**Select Puzzle [4 or D]**
Each generated puzzle has a code (up to 6 digits) shown in the top left corner of the game board. You can restart a particular puzzle by choosing *Select Puzzle* and entering puzzle code. The application remembers the last puzzle code and places it into the puzzle code text box at initialization. If this is what you need, don't modify the code − just select *Accept* to start puzzle.

**Enter Puzzle [6 or C]**
This action allows entering a custom puzzle. After you choose *Enter Puzzle*, you get a blank table, where you can place preset values and target sums. Press *Done* command key when ready to start solving the entered puzzle. Before accepting the puzzle, the application checks if the puzzle has solution(−s).

**Settings [8 or A]**
Brings up Settings form. Settings are discussed later.

**Best Results [7 or B]**
View best results. The application keeps up to 10 best results (shortest times) for each board size.
Only *randomly* generated and completed puzzles are processed. If you still find it possible to cheat − please let me know :)
The results are always shown for current board size specified with Settings. Each result is listed with the date it was scored.

**Rules**
Give a short description of game rules.

**Hot keys [9]**
Show the list of hot keys.

**About**
Show information related to the application (email, links, etc)

**Exit [CLEAR]**
Terminate the application.

## Operating the game.

Use **arrow keys** to highlight an element you wish to modify, then use FIRE and CLEAR to change its value. FIRE increments the value, when the value reaches maximum, it rolls over to minimum, CLEAR works in the opposite direction.

Alternatively you can use phone keys depending on keypad mode specified with Settings:

**Enter mode**

**Numeric keys**
Press a key to enter corresponding digit.
When entering a *table element* value, press 0 to clear the element
When entering a *target sum* value (custom puizzle specification), 0 works as digit 0. You often need to enter two digits for a sum. Try to avoid mistakes, otherwise correction might be tedious. For example, if you need 23, just type 2 and 3. If you started with 3 by mistake, press 2 to get 32, then press 2 again and 3 to get 23.

*** and # keys**
These keys are used to increase(*) and reduce(#) current element value

**Navigation mode**

Use numeric keys to select element, then use * and # keys to modify it according to the following table:

| | | |
|---|---|---|
| 1<br>Up−Left | 2<br>Up | 3<br>Up−Right |
| 4<br>Left | 5<br>Not used | 6<br>Right |
| 7<br>Down−Left | 8<br>Down | 9<br>Down−Right |
| *<br>Decrement value | 0<br>Not used | #<br>Increment value |

**Solved puzzle**

After a puzzle is solved by the computer, use keys as the following:

FIRE, RIGHT, or # − go to next solution
LEFT, or * − go to previous solution

## Game menu options

While in the game, the list of game menu options can be retrieved with Menu command key, *Pause* is often available as a separate key. This section explains the options.

**Pause**
Show pause window, where you can modify some settings. In Pause mode game time is not incremented.

**Solve**
Let the computer solve the puzzle. The maximum number of retrieved solutions is specified with Settings. Unregistered copy does not allow solving a custom puzzle.

**Test**
Check entered values for consistence. The application will show number of solutions where all entered values match, or "*NO solutions*". If the number of solutions exceed specified maximum in Settings, + is used, e.g. *10+ solutions.*

**Postpone**
Save game and quit the application. When you start the application next time, saved game is retrieved, so that you can contimue solving the puzzle.

**Quit**
Discard the puzzle and get back to main screen.

**Help**
Show Help window for current keypad mode (if unsure how to use keys).

## Settings

To get the list of customised options, select **Settings** from the main screen. Some settings can also be changed in the middle of game by using **Pause.**

Use **UP** and **DOWN** to highlight an item, then use

**FIRE** key or **Next** menu option − increment value of highlighted item
**CLEAR** key or **Previous** menu option − decrement value of highlighted item

In stand alone mode select **Done** to accept new values and get back to Main screen.
In **Pause** mode, select **Back** to continue the game, or **Quit** to discrad current game and get back to Main screen.

The settings are explained below. Default values are shown in **bold** type.

**Size**
Table size. Minimum is 3x3, default is 4x4, maximum is defined by screen size, but no more than 12x12. Size can't be modified in the middle of game.

**Sngl soltn**: On / **Off**
If On − produce a single solution puzzle, by adding some extra preset elements to generated puzzle. The application tries to minimise the number of added elements. The algorithm requires a considerable amout of heap size that may be unavailable, in which case the puzzle might have more than one solution − use **Test** (with no entered elements) to check if there is indeed a single solution. For a table size 5x5 the generation might take ages, but you can interrupt it at any time. This option is unavailable in **Pause** mode.

**Keypad:** Navigation / **Enter**
Keypad mode as discussed in Operating the game section.
If your device does not have LEFT and RIGHT arrows, you need to choose *Navigation* mode. This is the case for most Nokia Series 30 devices, where *Navigation* is the default keypad mode. For other devices *Enter*

mode appears to be more convenient and is the default.

**Hints**: Off / **Deltas** / Sum

Hint is a small value shown near each target sum. The following modes are available:

Off − don't show a hint
Deltas − show a signed difference between actual sum and target sum (a positive difference indicates an error)
Sums − show an actual sum (sum of non−empty elements for given row/column/diagonal)

**Hilite dgnl:** On / **off**

Whether or not to highlight the diagonal elements. Highlighting makes it easier to check diagonal sums, though some users might find it confusing.

**Max solve:** **20** to 500
**Max test:** **20** to 500

Maximum number of solutions to be retrieved with **Solve** and **Test** commands respectively (see Game menu options).

**Time limit:** **2min** to 1 hour, or none

Stop puzzle solving after specified time. You cannot modify time limit with an unregistered copy.

# Appendix. The algorithm.

## A preamble

The idea is so straightforward, that if no one else ever got to it, it could only mean that no one thought of it seriously. :)

Consider the following line (here *line* is used as a common term for row, column, or diagonal):

    a   10   b   c   |   30

Here *a*, *b*, *c* are unknown values, 10 is know value, 30 is the target sum

First impression is that any digit will suit for *a*, however it is not quite right

We have $a + b + c = 20$, and if $a=1$, then $b+c=19$ which is impossible: both b and c cannot exceed 9, therefore $b+c$ cannot exceed 18 !

In general, consider a line that has N empty elements ($N > 0$). Take *remaining sum* RS which is target sum less sum of all known values.

Let unknown values be $A_1, A_2, A_3, \dots A_N$. We have

$$A_1 = RS - (A_2 + A_3 + \dots + A_N)$$

and

$$1 * (N-1) <= A_2 + A_3 + \dots + A_N <= 9 * (N-1)$$

Therefore:

$$RS - 9 * (N-1) <= A_1 <= RS - (N-1)$$

Due to symmetry, the obtained constraint applies not only to $A_1$, but to all other unknown values on selected line as well.

## Now the algorithm...

**1°.** Replicate the table, so that each of the following operations will apply to the copy.
For each unknown value, set value minimum (VN) and value maximum (VX):

```
VN := 1
VX := 9
```

*BestElement*, *BestDistance* := MAX_VALUE;   *Modified* := false.

**2°** For each row, column, diagonal

Find:

    N :=  number of unknown (blank) elements
    RS := (Target Sum)  − (Sum Of All Known Elements)

In case  N = 0 (complete line)

    If  RS = 0 −  ignore the line (skip remaining operations in **2°)**, otherwise *InconsistentPuzzle* (leave function)

In case  N>0 (incomplete line):

    if  RS <=  0  − *InconsistentPuzzle* (leave function)

    Find line minimum (LN) and line maximum (LX):
    LN := RS − 9 * (N−1)
    LX := RS − (N−1)

    Update maximum and minimum for each unknown value on the line:
    VX := Min (VX,  LX)
    VN := Max (VN, LN)

    Process the distance

    *NewDistance* := VX − VN;

    if  *NewDistance* < 0  − *InconsistentPuzzle* (leave function)
    if  *NewDistance* = 0 (single choice),
        Element value := VX,   *Modified* := true;
    if  *NewDistance* > 0  and  *NewDistance* < *BestDistance*,
        *BestDistance* := *NewDistance*;  *BestElement* := (this element)

**3°.**  (We don't get here and below in *InconsistentPuzle* case).

Repeat **1°** and **2°** untill ends up with *Modified* = false, i.e. no modified (single choice) elements.

**4°.**  (We don't get here if  there is an incomplete line with  VX <= VN)

If *BestElement* is undefined, then all lines are complete − we found the solution !

Otherwise take *BestElement* an retrieve its boundaries VN and VX
For each V between VN and  VX (including the boundaries):

    Set *BestElement*'s  value to V
    Call the whole function recursively

## ... and more

To make sure that the above description is clear enough, here are some questions that you shouldn't have troubles to answer. If unsure, just send me a message and I will be happy to discuss the topic with you.

## Question 1

Here are two suggested modifications of LN and LX evaluation in **1°**. Can you explain why neither really improves the algorithm ?

1. LN := Min(9, RS − 9 * (N−1)
   LX := Min(9,  RS − (N−1))

2. LN := RS − 9 * (N−1)
   LX := Min(9,  RS − (N−1))

## Question 2

Why do we need to replicate the table: can't we just use current copy and reset *BestElement* after all its values have been tested ?
Assuming  table size is no more than 8x8 and 64−bit integers are available (e.g. *long* in Java, *long long*  in GCC,  *_int64* in VC++), we can easily improve the algorithm to avoid the need to replicate table (with 32bit integers it is less obvious).  How ?

## Question 3

Suggest a way to avoid memory heap fragmentation as a result of replicating tables.