`xsok` Version 1.02 Manual

Michael Bischoff

16-Mar-1996

Contents

1 Introduction

1.1 WARNING

Be warned. This manual is under construction. It may currently contain a lot of typos, bad English, bad style, and it's unfinished. The information herein should be correct, however.

1.2 `xsok` — A Generic Sokoban Game for X11

Similar to Tetris, almost every operating system has an implementation of the Sokoban game. It baffles by its easy rules, but nevertheless complex strategies.

For Linux, I found a textmode implementation and an X11-based version, both written 1992. The X11 version seemed to be based on the textmode version, and both use the same format for the level definition files.

`xsok` is a generalization of the standard Sokoban game. All original Sokoban levels may be played with `xsok` (again, using the same format for the level definition files). Some levels of a similar MSDOS game, Cyberbox, may be played also.

`xsok` is a one-person game played on a square-tiled board. It has no random elements, and is a pure strategic game. There are different floor types and walls, and objects standing on the floor. The objective is to push certain objects onto marked target squares, and to move onto the EXIT square, if one exists. The objects on the floor, which we will refer to as boxes, have certain attributes, such as weight, value, possible directions of movement, and own power. A special kind of object is the player itself. He has a certain power, which represents the maximal sum of the weights of the boxes he can push in a single move. The default power is one hundred, and equals the weight of a standard box. There are lighter (empty) boxes of weight one, which simply block your way, especially since some of them may be moved either only horizontally or only vertically. Some boxes even show interest in moving themselves into a certain direction, and may even push standard boxes! (I have no explanation for this phenomenon!) Luckily, most boxes are marked with arrows and color, so that you may distinguish them.

As well as the boxes, even the floor squares may have attributes. In Sokoban, the only floor attribute was the distinction between normal floor

and target spaces. In `xsok`, there may be different target spaces, for different box types. Some squares may be passed in only one direction, or restrict the box types which may be moved upon them. There are floor types which turn boxes, by 90 or 180 degrees. Some squares may give you power cookies, others let you suffer from weakness.

To manage this zoo, all possible levels are partioned into level sets, which share a common subset of the possible square types. A level set definition file assigns the attributes to characters.

The definition table for the floor types must follow a line with the contents "`;WALLS`" in the level subset definition file. For a floor type, there are the following attributes.

field	format	description
char	ascii	character, by which this floor type is identified in the level file
pic	hex	offset into the graphic data file
enter	hex	one bit set for every direction from which any object may enter this square
leave	hex	one bit set for every direction to which any object may leave this square
mask	hex	one bit set for every object class which is allowed on this square
effect	dec	identifies a special effect (rotation, target, exit)

The format determines how the entry is interpreted: as ASCII character, as hexadecimal number, or as decimal number. You may not use the ';' sign as ASCII representation of an object or a floor type, since it introduces comments and commands. Picture id 0 is reserved for the standard walls: There are 16 different tiles, from which one is selected depending on the 4 neighbour squares.

The effect field has the following meaning.

effect mod 100	description
0	no effect
1	rotate counterclockwise by 90 degrees
2	rotate by 180 degrees
3	rotate clockwise by 90 degrees
4	this is a target square for boxes
5	this is the exit square
6	increase power by 1
7	decrease power by 1
8	teleport

If the effect field is less than 100, the effect will affect every object once it enters the square. If the effect is at least 100, but less than 200, the effect will affect the first object which enters the square, and then the square will be substituted by a standard floor square. The standard floor square is the floor type which occurs first in the table. If the effect field is at least 200, the square will change to floor type (effect-200) / 100 once it is free again.

A teleporter will teleport any object on it to the first free teleporter available, or do nothing, if all other teleporters are occupied.

The definition table for the object types must follow a line with the contents "`;OBJECTS`" in the level subset definition file, and must be behind the floor definition array. For an object, there are the following attributes.

field	format	description
char	ascii	character, by which this floor type is identified in the level file
pic	hex	offset into the graphic data file
movedir	hex	one bit set for every direction in which this object may move
pushdir	hex	one bit set for every direction in which this object will move automatically
weight	dec	the weight of this object
power	dec	the power which the object has for moving
mask	hex	defines the object class (multiple classes are possible)
score	hex	the bonus you receive if this object is on a target square

The object which represents the man is the first object in the table. Its `mask` entry must be 1, since this is hardcoded into the program. The `mask` entry should only use bits 0 to 15. The higher bits are needed for the `Cyberbox` selectors, their coding is subject to change.

2 Adding New Level Subsets

If you want to create new level subsets, create a text file `gametypes` in the directory `/usr/games/lib/xsok`, where every line of this file represents the name of a new level subset. Suppose you want to create the subset `Foo`. Create a description in the file `/usr/games/lib/xsok/Foo.help`. Create a subdirectory `/usr/games/lib/xsok/Foo`, and put in a file `definitions` and the files `screen.??` for the levels. For examples, see the subdirectories of the `lib` directory in the source distribution. Edit the level files until you are satisfied. Then, you may concatenate all the files in the `/usr/games/lib/xsok/Foo` directory using the `combine` program, and create the more compact file `/usr/games/lib/xsok/Foo.def.gz`.

The level file simply is an ASCII representation of the floor types and objects. The floor below an object is by default the first floor type which occurs in the `definitions` file. If you need another combination (for example a box already on a target square), you have to dedicate a new character for this combination, and add an `ATOP` command in the `definitions` file. In Sokoban, a box is represented by a dollar sign and a target square by a dot. For a box on a target square, a star is used. The defining line in the Sokoban definitions file is "`;ATOP *$.`". The possible commands for the level files are the following.

command	description
`;AUTHOR` *string*	the author of the level
`;COMMENT` *string*	a level comment or hint

The possible commands for the definition files are the following. They occur after the floor and object definition tables.

command	default	description
;MAXLEVEL *no*	99	the maximal level number
;PUSHCOST *no*	10	score reduction for every push
;MOVECOST *no*	1	score reduction for every move
;ATOP *cof*	–	defines object/floor combinations

3 Keyboard and Mouse Commands

In `xsok`, keyboard and mouse commands are bound to functions using an ASCII key definition file.

There are two classes of bindings: Built-in assignments have the least priority. Only the following commands are built-in:

key	action
`Ctrl-L`	redraw the window
`Ctrl-R`	redraw the window
`<ESC>`	abort a move
"`C`"	switch to Cyberbox levels
"`S`"	switch to Sokoban levels
"`X`"	switch to Xsok levels

These bindings can be extended or overridden by the loadable keyboard table, which may be different for every national language. The default file resides in `/usr/games/lib/xsok/keys`. With this file, you may define one key or mouse button per line. The format is ¡*specifier*¿ ¡*command*¿. A specifier is a single character, or the string `#x`, followed by the hex representation of the key, one of the strings `Up`, `Left`, `Down`, or `Right` (which correspond to the arrow keys of your keyboard), or the string `Mouse`n, where n is a digit from 1 to 5. It is currently not possible to assign commands to the function keys.

The possible command names and their meaning are

`None` Do nothing if this key is pressed. Only used to remove built-in bindings.

`rq_LeaveSok` Popup a window and ask whether to leave the game. If you are at the beginning of a level, or finished the game, no confirmation is requested.

`rq_NextLevel` Request to proceed to the next level.

`rq_PrevLevel` Request to go back to the previous level.

`rq_RestartGame` Request to restart the game. Not really needed, since

restarting may be undone.
`Up`, `Left`, `Down`, `Right` Move the man one square in the specified direction.
`NextUnsolved` Proceed to the next unsolved level.
`NextLevel` Proceed to the next level.
`PrevLevel` Go back to the previous level.
`UndoMove` Undo an elementary move, or the restart command. You may undo all moves back until the start. Undoing moves may be slow, however.
`RedoMove` Redo a previously undone move.
`LeaveSok` Unconditionally quit the game.
`ShowScore` Show your current score, and other information. This is automatically done after every succesful move.
`ShowBestScore` Show the largest score, minimum number of moves, and minimum number of pushes known for solving this level.
`ShowAuthor` Show the name of the author of the current level, as given by the `;AUTHOR` directive in the level file.
`RestartGame` Go back to the starting position.
`ReplayGame` Replay the game at about 14 moves per second.
`SaveGame` Store the current position in the file
`/var/games/xsok/`*Levelset.level*`.sav`.
`LoadGame` Restore a previously saved game. If none exists, restore a solution file `/var/games/xsok/`*Levelset.level*`.sol`.
`ReadScores` Re-read the highscore table. (It may have been modified by some other player on your machine.) Highscorefiles are reloaded automatically at the beginning of every level, and when you finish a level.
`ShowVersion` Will display the current `xsok` version. `ResizeWindow` Obsolete. Is called automatically, if the level size changes.
`RepeatMove` Repeat a move in the previous direction. (In `xsokoban`, this one is assigned to the middle mouse button.)
`MouseMove` If pressed on a clear square, the man will move to that location via the optimal path if such a path exists. If pressed on an object that is adjacent to the player, the object will be pushed.
Since coordinates are needed, **this function should only be bound to a mouse button**.
`MousePush` If you click onto a square with same x or y coordinate as the man, the man will walk straight to this square, possibly pushing any boxes on the path. This is useful for pushing a box multiple squares along a straight path.
Since coordinates are needed, **this function should only be bound to a**

mouse button.
`MouseDrag` This command requires that you press the mouse button on a location where a box resides, drag the mouse, and release the button on an empty square. The man will then move the box from the first square onto the second with the minimal number of pushes, if it is possible at all. Please note that the man will not move any other object and will only use squares without effects. This function may not work when other active components are on the board (pushing boxes etc.).
Since coordinates are needed, **this function should only be bound to a mouse button**.
`MouseUndo` Undoes one of the previous two compound move operations, or a `PlayMacro` operation. Currently, this is allowed only immediately following the operation to be undone, and only a single compound move may be undone. If this situation does not apply, the command will execute an elementary undo operation.
`LevelInfo` Show the level comment, as defined by the `;COMMENT` directive in the level file. Shows the highscore for this level, if no comment is available.
`DropBookmark` Drop the bookmark at the current move number. In some cases, a bookmark is dropped automatically. After a load game operation, the bookmark is set at the end position. If a bookmark becomes invalid, since you undid moves and continued with new ones, the bookmark will be reset to the last common move number.
`GotoBookmark` Go to the bookmark.
`StartMacro` Start recording a move macro. A move macro is a sequence of moves, which may be re-executed by a single command, if the starting position does match. This is useful for some **Sokoban** levels, which tedious repetition of long sequences.
`EndMacro` Define the end of a move macro.
`PlayMacro` Reexecute the move macro, i.e. move the man to the starting position of the macro by the `MouseMove` command, then repeat the recorded moves. If the path is blocked, macro replaying stops at that point.
`Cancel` Cancel a confirmation.
`Confirm` Confirm a confirmation.

The default assignment is

key	action
"y"	confirm action
"n"	abort action
"q"	quit game (confirmation requested)
"v"	show version number of `xsok`
"a"	show author of a level (if known)
"b"	show best scores for this level.
"i"	show witty quote for a level (level comment)
"u"	undo a move
"r"	redo a move
"?"	show your current score
"s"	save the game
"L"	load a game
"c"	drop the bookmark
"ctrl-U"	goto bookmark
"R"	restart this level
"N"	go to next level
"P"	go to previous level
"U"	go to the next unsolved level
"W"	resize the window

4 Bells, Whistles, and Doodads

4.1 Internationalisation

The Athena widget set allows the redefinition of button labels by resource files. All other texts have also been made reconfigurable. The X-window system uses the contents of the `LANG` environment variable to select the path for the resource file (application defaults file). Currently, the code is there, but there are no example translations. If `xsok` doesn't work, run it with your `LANG` environment variable set to the empty string.

4.2 Saving Games

You can save the current state of the game by pressing the left mouse button on the "save game" button. `xsok` will try to create a file in the directory `/var/games/xsok` with its name consisting of the name of the level subset,

the level number, and the extension `.sv`. If you solve a level and break the highscore, the game is saved automatically. There are three different file extensions: `.bs` for *best score*, `.mp` for *minimal number of pushes*, and `.mm` for *minimal* number of moves. If some of them are identical, links will be used to save disk space.

4.3 Format of a Highscore File

A highscore file is organized as array of 300 4-byte integers. The representation is machine-independent, using high-byte-first ordering. This allows exchanging files between workstations and PCs. The first entry holds the magic, 0x741d. Entries 1 to 99 hold the best known score for the respective level. Entry $n + 100$ holds the minimal known number of moves to solve it, and entry $n + 200$ the minimal known number of pushes. Entries 100 and 200 are unused and have value 0x7fffffff.

4.4 Format of a Savegame File

A savegame file starts with a header of 16 4-byte integers in machine-independent representation. The entries are as following.

Entry	Description
0	Magic number, 0x741c
1	score
2	number of pushes
3	current move number
4	time saved
5	1 if level finished, 0 else
6	level number
7	number of stored moves
8	move macro start
9	move macro end
10	move macro x
11	move macro y
12	position of bookmark
13	-1
14	-1
15	-1

After this array, an array of 8 characters, representing the level subset name, is stored. The level and level subset information is currently not checked at load time, this may be done later.

After the level subset name, a one-byte charcter count is given, and the username of the player, as string of the given length. The terminating zero byte is not stored.

Finally, the array of moves is appended, whose length is given by entry 7 of the header.

4.5 Taking Back Moves

`xsok` logs every move. This gives you the possibility of unlimited undo. Undoing is done by restarting the level and replaying it. Therefore, undo may be really slow after some hundred moves. For Sokoban rules, a better undo would be possible, but for the general case, a move may involve complicated actions.