CS39002 Operating Systems Laboratory
Spring 2025

Lab Assignment: 4
Date of submission: 29–Jan –2025

Inter-process communication using pipes

In this assignment, you design a multi-process application that lets a user play the Foodoku puzzle in an interactive manner. If you do not know what the rules for a standard 9 × 9 Foodoku are, here are a sample Foodoku puzzle and its solution.

1					8			9
		2						8
	8		5	4	9			
	4		2			9		
3		9				2		1
		1			5		4	
			9	1	2		3	
7						1		
2			7					6

The puzzle

1	3	4	6	2	8	5	7	9
9	5	2	1	3	7	4	6	8
6	8	7	5	4	9	3	1	2
5	4	6	2	7	1	9	8	3
3	7	9	4	8	6	2	5	1
8	2	1	3	9	5	6	4	7
4	6	8	9	1	2	7	3	5
7	9	5	8	6	3	1	2	4
2	1	3	7	5	4	8	9	6

The solution

Call each 3 × 3 region (enclosed by bold lines in the above figure) a *block*. The objective of the game is to fill the empty cells by the digits 1 through 9 such that no block or no row or no column contains repetitions. Number the nine blocks and the nine cells in each block as 0–8 in the row-major fashion.

In our application program, the user interface is launched by running a coordinator program from a shell. The coordinator forks nine child processes. Each child process handles one block, and launches an `xterm` for the display of that block and of all error messages associated with that block. The codes for the coordinator and for each block are to be written in two files *coordinator.c* and *block.c*. The coordinator (parent) creates nine pipes for communicating with nine child processes. The blocks need to communicate with one another. The same pipes are used for that purpose too. During the game, each block *B* communicates with four other blocks: two in the same row as *B*, and two other in the same column as *B*. Let us call these four blocks the (*row* and *column*) *neighbors* of *B*. During the forking of a block *B*, the parent passes, as command-line arguments, the block number for *B* (an integer in the range 0–8), the two pipe descriptors for communicating with *B*, and the write ends of the pipes of the four neighbors of *B*.

All communications through the pipes must happen using the high-level `printf` and `scanf` functions (or by `cin` and `cout`). A block does not directly communicate with the user. It only receives messages from the coordinator and from its four neighbors. So each block process can permanently close its `stdin`, and `dup`() the read end of its pipe to its `stdin`. The descriptor for `stdout` of a block cannot be permanently closed. Each block process needs to draw and redraw its block and to print error messages, in that block's `xterm`. For this purpose, the original `stdout` is to be used. For communicating with a neighbor, `printf` (or `cout`) should be used, but this time, the printed stuff will travel through the pipe of the neighbor. So the `stdout` of each block process will switch between screen printing and pipe printing. For the coordinator, all inputs are from the user, and the original `stdin` will work throughout. But its `stdout` will switch between the original version (for printing to the coordinator window) and the write ends of the pipes of the block processes (for passing commands). Use `dup`() appropriately to ensure this.

The commands used in the game are summarized below. Each command is a letter followed by the requisite number of `int` arguments.

User commands to the coordinator

h	Print a help message (including the numbering scheme of the blocks and the cells in each block).
n	Launch a new puzzle. A program *boardgen.c* is supplied to you. `#include` this file in the code for the coordinator. A function *newboard*(*A*, *S*) is implemented in *boardgen.c*. Here, *A* and *S* are 9×9 `int` arrays for storing the puzzle and its solution, respectively. Empty cells in *A*[9][9] are written as 0's. The solution does not contain any empty cell. The call *newboard*() populates these two arrays by a random-looking Foodoku puzzle. After the function returns, the coordinator sends the 3×3 blocks of *A*[9][9] to the child processes via their respective pipes. Each child knows only its own block, and nothing about the other blocks.
p *b c d*	Request to put (place/replace) digit ***d*** at the ***c***-th cell of the ***b***-th block. The cell may be empty (place) or already filled by the user (replace). The coordinator checks whether ***b***, ***c***, and ***d*** are in valid ranges. If not, it sends an immediate error message to the user, and proceeds no further. If ***b***, ***c***, ***d*** are all valid, the coordinator sends a request **p *c d*** to block ***b***. The coordinator does not check whether placing or replacing the digit in the indicated cell results in any conflict.
s	Show the solution of the current puzzle. This is the same as sending the blocks of the solution *S*[9][9] (instead of *A*[9][9] as for the command **n**) to the block processes.
q	Send a quit command to the child processes, wait for them to exit, and finally exit itself.

Commands to each block

n	This command originates from the coordinator, and is followed by nine integers standing for the entries of the corresponding 3×3 block. An empty cell is denoted by 0. Upon receiving these nine digits, the block populates two private 3×3 arrays with these entries: *A*[3][3] for storing the block of the original puzzle, and *B*[3][3] for storing the current state of the block.
p *c d*	The cell number ***c*** and the digit ***d*** follow this command. This command again comes from the coordinator to the block ***b*** as a response to the user command **p *b c d***. The block first verifies from its own copy of *A*[3][3] whether the user requests to replace a digit in the original puzzle. If the user does that, this causes the error `Read-only cell`. If not, the block checks whether ***d*** is already present in some cell of its own copy of *B*[3][3]. If so, it causes a `Block conflict` error. If not, the block sends a row-check request (see **r** below) to its two row neighbors. Each neighbor responds by sending 0 (no error) or a suitable non-zero integer (`Row conflict` error). If there is no error reported by the row neighbors, the block finally sends column-check requests (see **c** below) to its column neighbors. Each neighbor responds by sending 0 (no error) or a suitable non-zero integer (`Column conflict` error). Notice that the above checks are with respect to consistency in the current board. No efforts should be made to verify correctness of user inputs with respect to the solution. Indeed, the coordinator does not send the blocks of *S*[9][9] to the block processes (unless the user gives the command **s**, in which case the complete solution is received from the coordinator, all cells become read-only, and the user cannot proceed with the current game). If there is no error so far, the digit ***d*** is placed in the requested cell ***c***, the local array *B*[3][3] is updated and redrawn in the `xterm`. If there is an error, an appropriate message is printed below the current block. In order that the user can read that message, the block process waits for some time (like two seconds), and then redraws the earlier block (now *B*[3][3] undergoes no change).
r *i d*	This request comes from a row neighbor, and is followed by a row number ***i*** and a digit ***d***. In order to serve this command, the block process looks at the ***i***-th row of its *B*[3][3], and responds by sending 0 or a `Row conflict` notification. The block process does not know from which row neighbor the request has originated. So the request must also be accompanied by an identification (write pipe-descriptor) of the requesting block.

c ***j d***	This is similar to **r** with the exception that now the requested digit ***d*** is checked for conflict in the ***j***-th column. The response is 0 or a `Column conflict` notification.
q	This command comes from the coordinator (in response to the user command **q**). The block process writes a message (like `Bye...`) below the block, waits for some time (so the reader can read it), and exits.

Read the manual page for `xterm` to know what command-line options it supports. A window of default size is usually too big to store only a 3×3 block. You can set the window size and position to your desired values by the command-line option `-geometry`. You can specify a customized title by `-T`. A font-face (like `Monospace`) can be specified by `-fa`, and a font size by `-fs`. At the end, you specify, using `-e`, that you want to run `./block` (instead of a default shell like *bash*). Here is an example.

```
xterm -T "Block 0" -fa Monospace -fs 15 -geometry "17x8+800+300" -bg "#331100" \
-e ./block blockno bfdin bfdout rn1fdout rn2fdout cn1fdout cn2fdout
```

Each block (that is, child) process `exec`'s this command to open and to transfer the display to an `xterm`. Set the geometries of the nine `xterm`'s appropriately, so that these windows line up as in a real Foodoku puzzle. Avoid manual relocation of the `xterm`'s by the user.

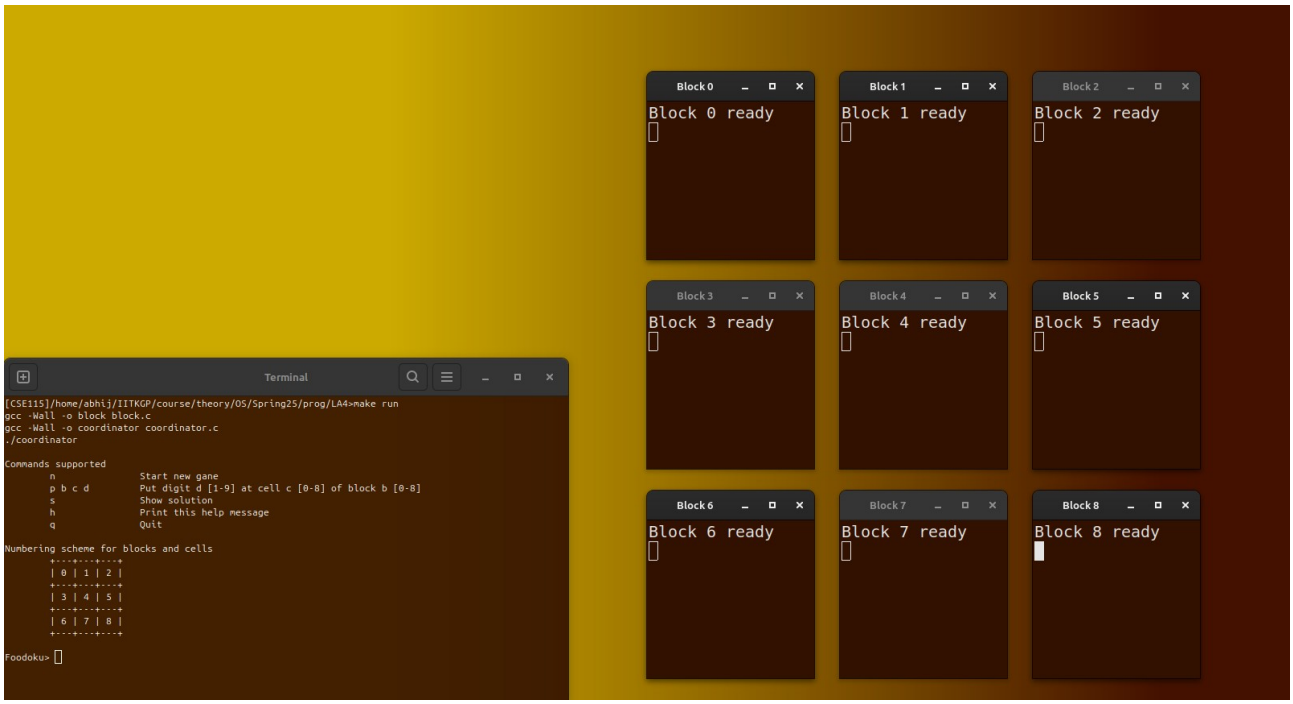
You may use the following *makefile*. Notice that *boardgen.c* is not to be compiled. It has no main function. It consists only of an implementation of the function *newboard*(). Only `#include` this in *coordinator.c*.

```
all: boardgen.c block.c coordinator.c
        gcc -Wall -o block block.c
        gcc -Wall -o coordinator coordinator.c

run: all
        ./coordinator

clean:
        -rm -f block coordinator
```

Submit a single archive consisting of *coordinator.c* (your code), *block.c* (your code), *blockgen.c* (exactly as it is supplied to you), and *makefile*.

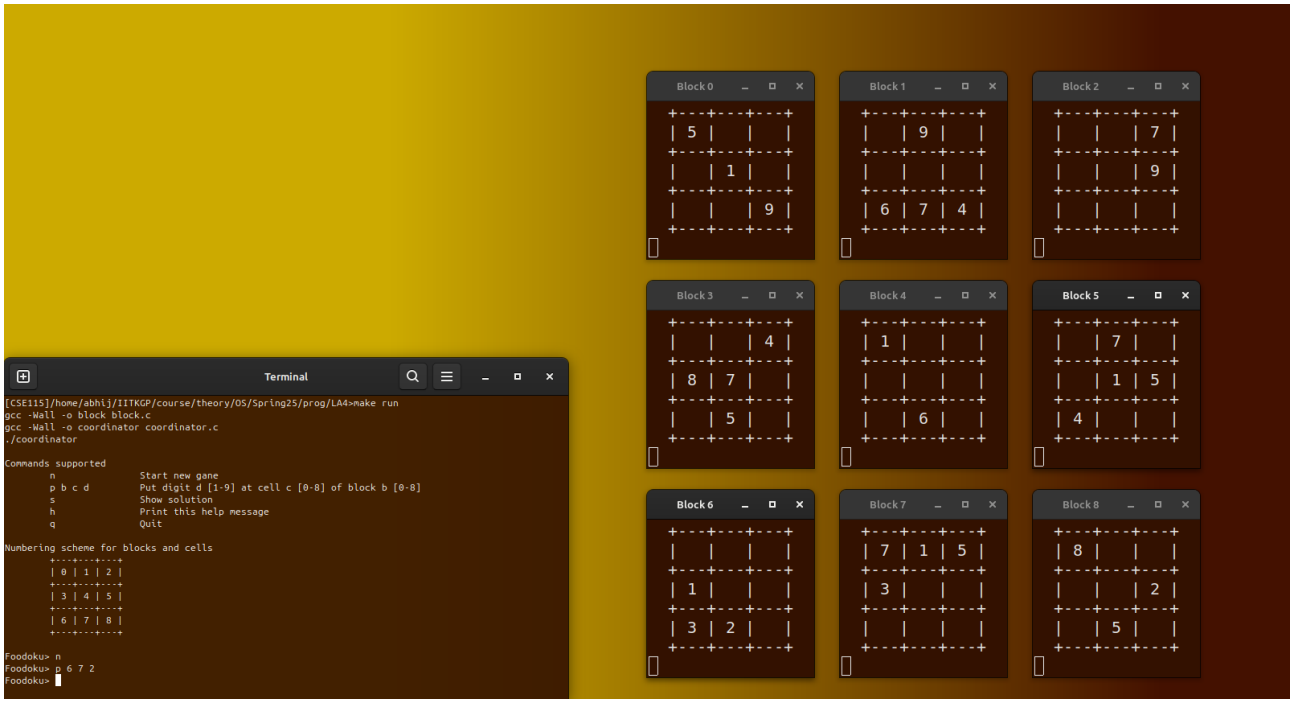
Sample Output

Run the coordinator

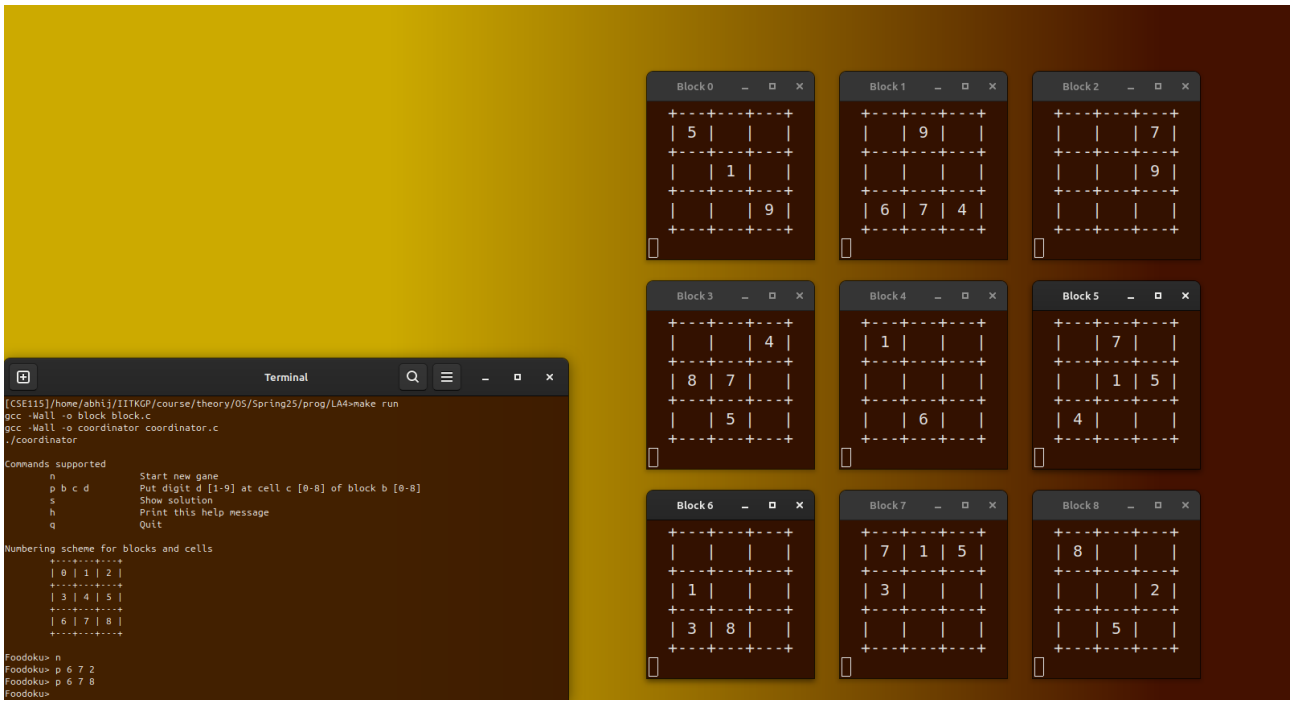


New game

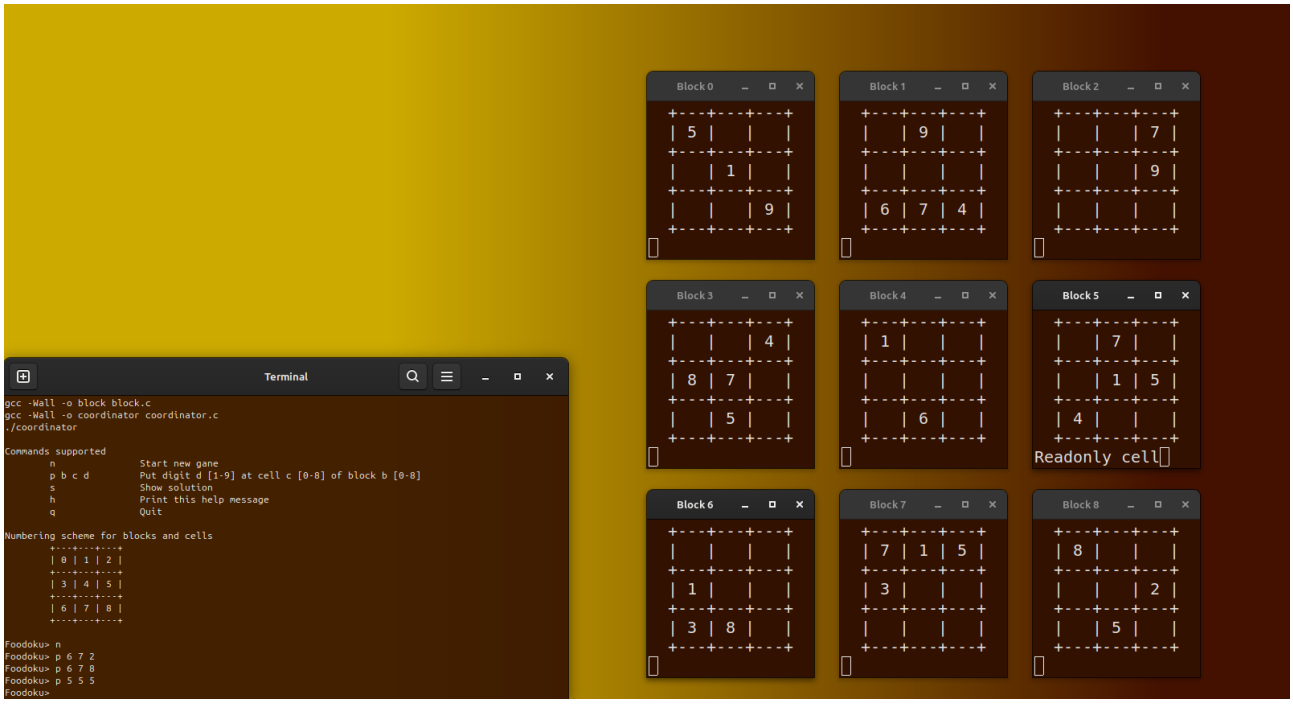
Place a digit in an empty cell

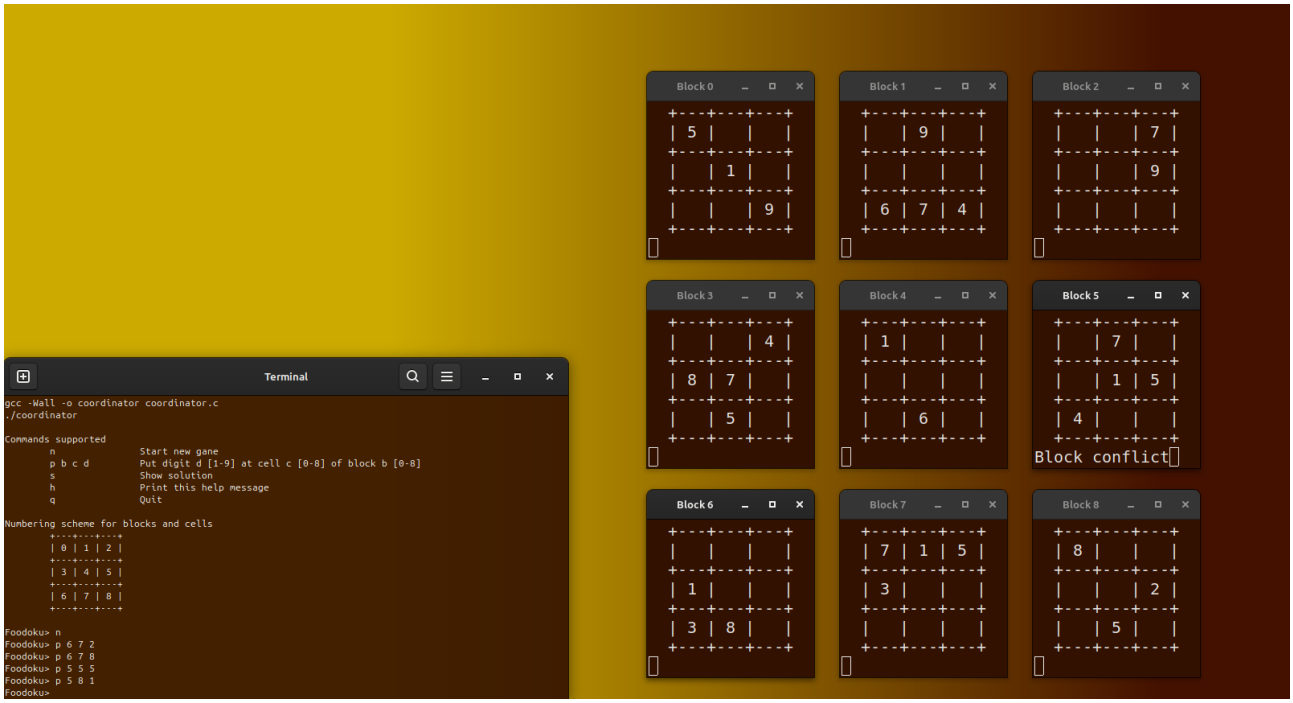


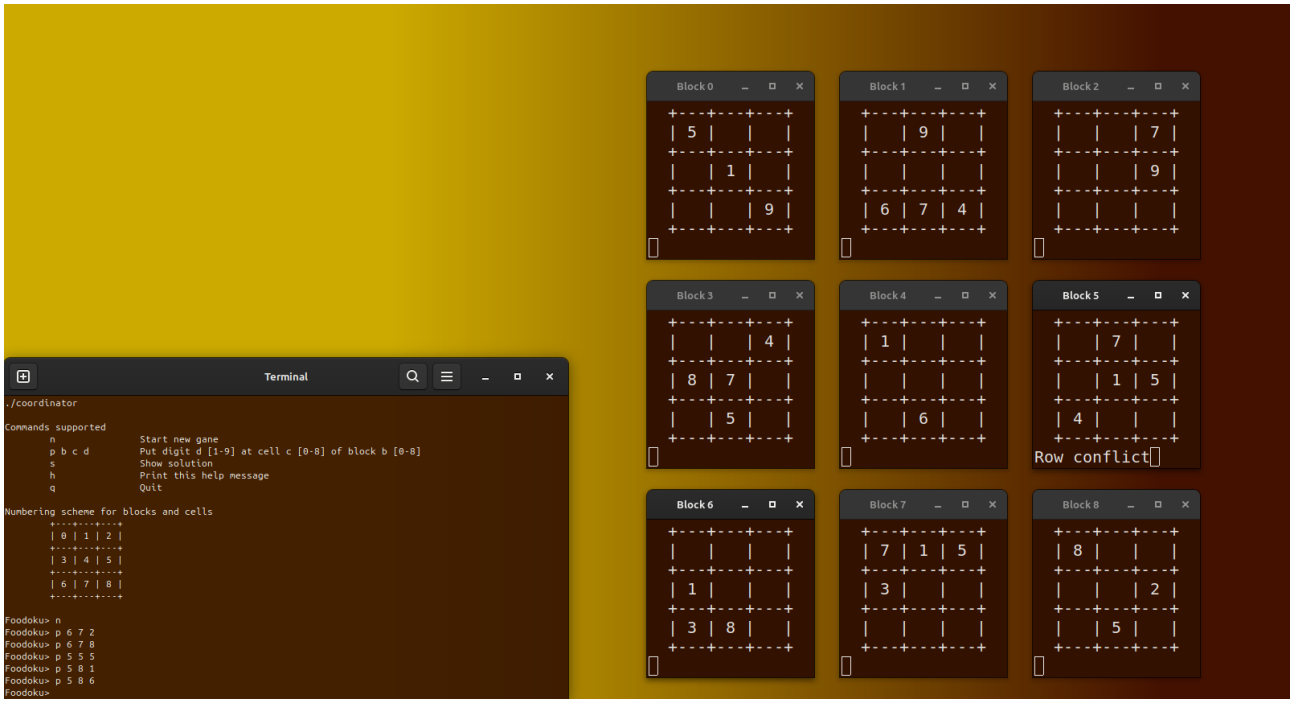
Replace a digit in a filled cell

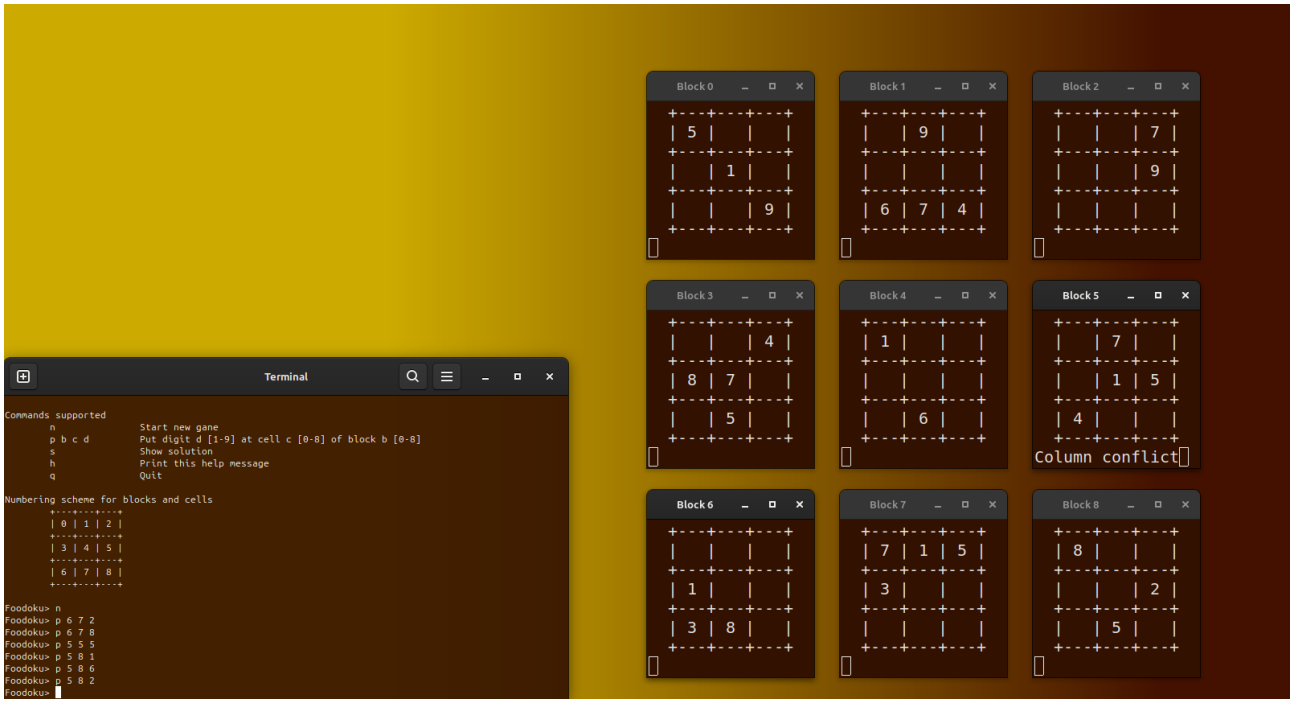


Four types of errors

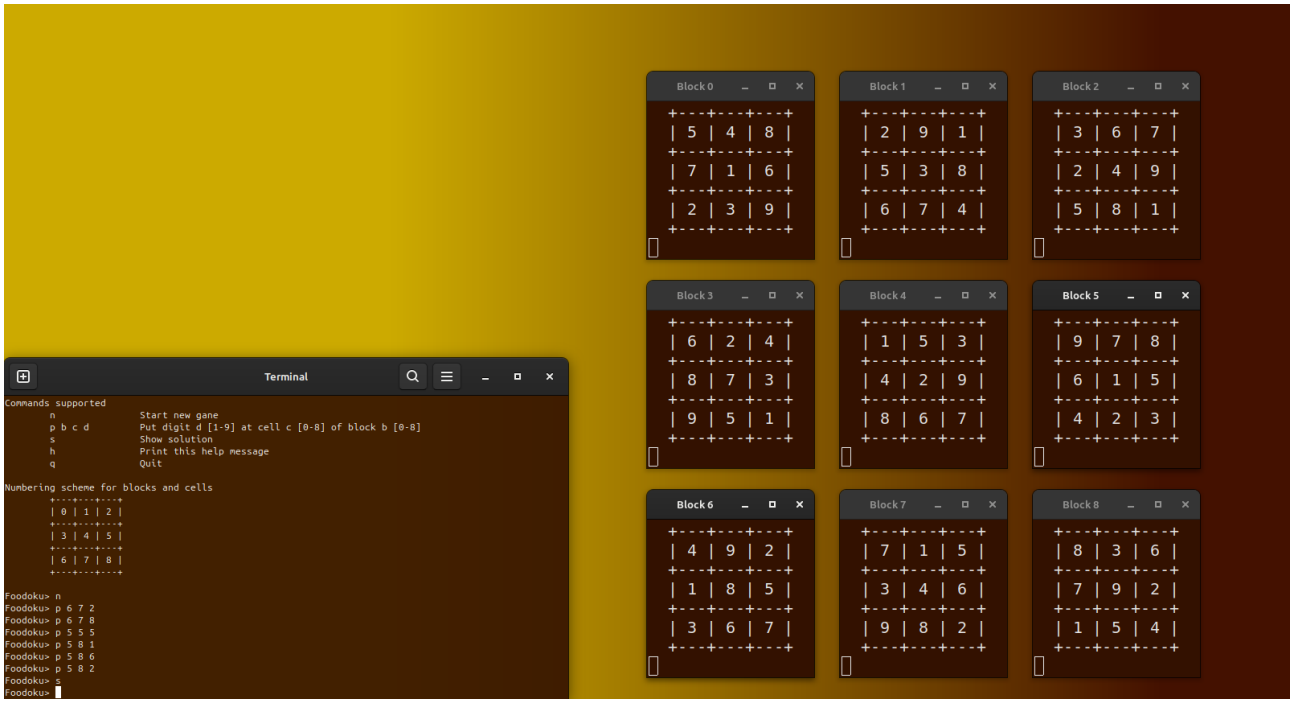





Terminal
./coordinator
Commands supported
n Start new gane
p b c d Put digit d [1-9] at cell c [0-8] of block b [0-8]
s Show solution
h Print this help message
q Quit
Numbering scheme for blocks and cells
0	1	2
3	4	5
6	7	8
Foodoku> n
Foodoku> p 6 7 2
Foodoku> p 6 7 8
Foodoku> p 5 5 5
Foodoku> p 5 8 1
Foodoku> p 5 8 6
Foodoku>
Block 0
Block 1
Block 2
Block 3
Block 4
Block 5
Row conflict
Block 6
Block 7
Block 8


Terminal
Commands supported
n Start new gane
p b c d Put digit d [1-9] at cell c [0-8] of block b [0-8]
s Show solution
h Print this help message
q Quit
Numbering scheme for blocks and cells
0	1	2
3	4	5
6	7	8
Foodoku> n
Foodoku> p 6 7 2
Foodoku> p 6 7 8
Foodoku> p 5 5 5
Foodoku> p 5 8 1
Foodoku> p 5 8 6
Foodoku> p 5 8 2
Foodoku>
Block 0
Block 1
Block 2
Block 3
Block 4
Block 5
Column conflict
Block 6
Block 7
Block 8

Show the solution



After this (for that matter, at any point of time, even during an ongoing game), the user may start a new game by supplying the command **n** again.

End game