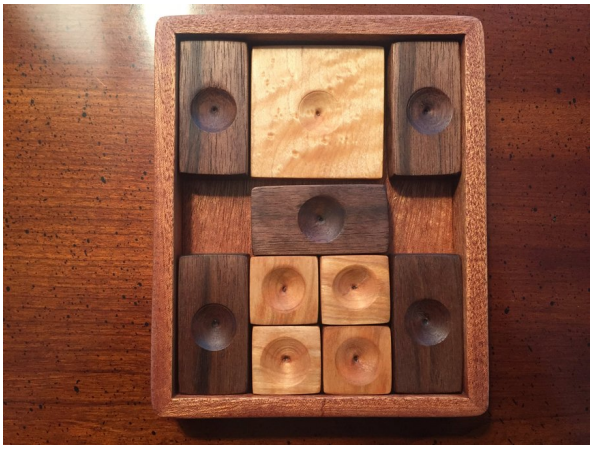
212 Final Project

Due: Sunday, May 22nd @ 11:59pm

Synopsis

You will conjure all your creativity, cunning, and data structures expertise to compute solutions to a sliding block puzzle. Here's a picture of the of the puzzle:

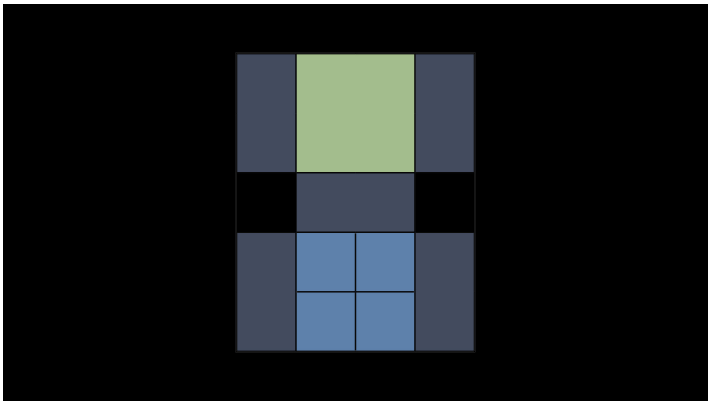


Wooden version of puzzle.

The objective is to slide the blocks around so that the large square (which you'll notice has a lower profile than the other blocks) can slide through the hole in the bottom of the box, which is just wide enough for the large square:

Objective of puzzle.

We'll represent the puzzle with some delightful rectangles:



Our representation.

Outlined square at bottom shows the goal.

Your job is to compute a solution (if one exists) from any given starting configuration of the pieces.

Details

The skeleton gives you a good head start on the graphics part (see the controls below). However, there are many things left for you to finish, including:

- Storing some representation of the puzzle's board / configuration
- Figuring out how to solve the puzzle (this will likely involve numerous data structures we've learned about)
- Stepping through the solution, showing each step on screen

Compiling

As usual:

```
$ make  # compile project
$ make debug  # same, but with debugging symbols + assertions
```

However, note that you need SDL2 (and development / header files) for it to work. Everything you need should be present on the 103 virtual machine. If you are using something else, SDL is easy to find for most OSes.

Note: use `make -B [debug]` to force recompilation when switching from a normal build to debug (or vice versa).

Note: if you want to draw text on the screen (as in my demo), you might find SDL_ttf useful (sadly I don't think it is on the 103 virtual machine). Remember to add `-lSDL2_ttf` to your linker flags in this case.

Note: Some virtual environments (e.g. VirtualBox) might not give you a graphics interface that supports vsync'd rendering. If you get errors or warnings when running the skeleton on a virtual machine, find these lines (in the `init()` function) and comment or remove:

```
if(!SDL_SetHint(SDL_HINT_RENDER_VSYNC, "1")) {
    printf("Warning: vsync hint didn't work.\n");
}
```

Lastly, you may need to look up the API for SDL now and then. Link for convenience..

Keyboard + mouse controls

The skeleton doesn't do much, but it does more than nothing. In particular:

- You can drag the blocks around with the mouse, and if they are close to a grid corner, they will "snap" into place.
- If you right click a rectangle, it will toggle the orientation (horizontal vs vertical).
- Press `q` or escape to quit the program. (Or just close it via your window manager.)
- There are some "TODO" comments in main which suggest what you might make other keys do.

NOTE: I have made no effort to stress test the user interface, and I can already think of places where certain inputs might get it confused (e.g., start dragging a shape with one mouse button, then press another…). It should work in the majority of "obvious" cases, but be aware that it is not that robust.

Testing

Give the puzzle an initial state, run your solver, and see if it actually makes sense. If you are worried about the optimality, you can show me the input on piazza and I'll tell you if I agree with the number of required steps.

Debugging

You can use `gdb`, pretty much as usual (although it can be a little awkward with multithreaded graphics programs). You might also find "printf debugging" useful at times (write some program state of interest to `stdout`). ***Note:*** when debugging you might want to make the program not run full screen so you can more easily see your `printf` output, or interact with `gdb`. Just apply these changes in `solver.cpp`:

```
// #define FULLSCREEN_FLAG SDL_WINDOW_FULLSCREEN_DESKTOP
#define FULLSCREEN_FLAG 0
```

Hints and notes

One of the first tasks you might consider is to find a nice representation of configurations of the puzzle. If you think about it, you can actually stuff them into long integers, which might be convenient for use in other data structures. Other things to look out for include:

- Early on, you may want to establish a text representation of the board configuration that you can print to `stdout` and check against what is on the screen.

- Make sure your notion of the configuration stays synchronized with what is drawn on the screen (say, after a user moves pieces around), and make sure you don't allow overlapping blocks.
- Computing the possibilities of movement is a bit cumbersome, and will probably account for a few hundred lines of your program's source. I would encourage you to think **very** carefully when writing that bit of code, because it will likely be a nightmare to debug if you get it wrong the first time.

Note: I considered any straight-line movement of a block as a single "step", even if the block moved two spaces. If you'd like to compare your answers with mine, please do the same. One might generalize further and consider any continuous movement of a block as a single step (the difference being that the small squares could go around a corner in one step), but I didn't bother to do so.

Hard mode

If you want to really impress me, try one or more of the following:

- Make the game "playable". That is, implement ***convenient*** controls (keyboard or mouse) that let you slide the blocks around (the default mouse controls I wrote don't have a very nice "feel" for actually playing the game).
- Find a configuration of the board that requires the **maximal** number of steps to solve, and prove to me that your answer is correct. (Btw, in just messing around, I stumbled upon a configuration that requires 150 steps.)
- Re-write the whole project from scratch. Don't refer to the skeleton at all.
- This isn't actually that hard, but you might improve the aesthetics by creating textures from images for the blocks instead of just using solid colors.

Notes on Grading / Submission

This project is to be completed **individually**. Every line of code you add to the skeleton should be your own, and you should be able to explain every such line in detail if I ask.

Please keep track of your work in a git repository so I can see how things progressed. Local commits (no remote mirror on bitbucket / github, etc.) are fine (in fact, preferred). You can add commits to the `csc212-projects` repository, or make a new one:

```
$ cd ~/csc212-projects/
$ cp -r p-final/ final/
$ cd final/
$ git init
# work on project, committing changes to reflect incremental progress
$ git commit -a -m 'Accomplished XYZ...'
# finally, make an archive to send me:
$ make clean   # don't turn in binaries + object files
$ cd ~/csc212-projects/
$ tar -czf final.tgz final/
$ ./submit.sh final.tgz
```

You can replace the last step with submission via the web form.