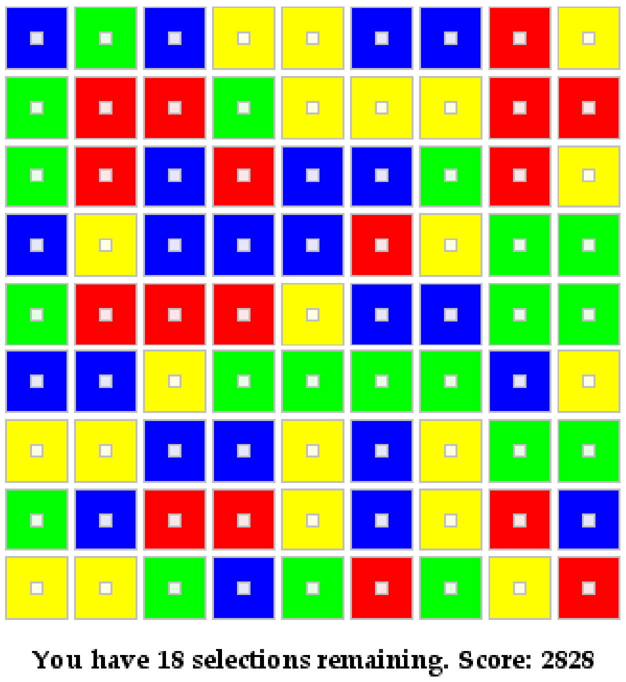
CS106B
Summer 2011

Handout 04
June 20th, 2011

Assignment 1: Blocky!

Your first assignment has you implement variation on a one-player game called Blocky. The rules are simple: click any single block and drag the mouse to outline a rectangle so that all four corners are the same color. Then release the mouse to clear the region, collecting all of the pellets residing inside. Once you've collected all of them, you'll advance to another level. The goal is the collect all of the pellets across all levels as efficiently as possible.

Each level is more difficult than the preceding one, because either the board dimension or the number of possible colors increases. The game ends when you fail to complete a level, complete all twelve levels, or you decide you've had enough and quit the application.



Due: Tuesday, June 28th at 4:00 p.m.

Our first assignment is as much about coding as it is getting used to a new language and a new development environment. Truth be told, the amount of code you need to write for this assignment is not that much. But you're learning a new language and a new development environment, and that will likely be as difficult, if not more difficult, than the assignment would have been had it required Java instead of C++. (I'm not dismissing C++, mind you. Trust me—C++ will soon allow us to do things that aren't nearly as easy to do in Java.)

The Rules

For the most part, the rules are easily intuited by just playing a few games. But here they are anyway. ☺

- ❖ There are a total of twelve levels, and you advance to each level as you complete the one that precedes it. The game ends either when you fail to complete a level, or you advance through all twelve.
- ❖ The board dimension and color count vary with each level. The first board is 5 by 5, using just two colors, and the twelfth board is 20 by 20 and uses five colors. The starter code makes it clear what these numbers are for all of the levels in between. Just look at the definition of the **`kLevelDescriptions`** constant in the starter code and you'll see.
- ❖ You're given a limited number of selections for each level, and if you exhaust them without clearing the board, the game ends. The number of selections granted per level is always twice the board dimension. That means you get 10 moves to clear a 5 x 5 board, 24 moves to clear a 12 x 12 board, and so forth.
- ❖ Each selection must have be of width 2 or more, height 2 or more, and such that all four corners are the same color. If your selection is illegal, then the board stays as is but your remaining selection count still goes down. If your selection is legit, then the selected region is cleared and repopulated with randomly colored, pellet-free squares. It's kind of like you're rounding up marshmallows with a rectangular lasso.
- ❖ You win (or lose) points with every legal move. You get 7 points for every pellet you collect with any given selection, but you lose a point for each location included in a previous selection (i.e. the location's pellet has already been collected.) Occasionally, you have no choice but to include a large fraction of previously collected squares in hopes that the replacement squares will present you with a better move. And very occasionally, the four corners of the initial board are all the same color, so you can clear everything and advance to the next level in just one swoop.

Managing the Grid

We invested almost a lecture's worth of time on the Queen Safety example so that you had some experience with the CS106B **`Grid`** class. It is possible to use raw, exposed arrays to complete the assignment, but C++ arrays are exceptionally difficult to use properly—particularly when they need to be passed as parameters. It's preferable to wear your seat belt and deal with the more graceful and more easily handled **`Grid`** to manage all of your two-dimensional array needs.

Blocky Graphics

Those at CS106B headquarters have already implemented all of the graphics and mouse event services you need to arrive at a fully working program. In fact, the starter files contain enough code to illustrate a good chunk of the functionality implemented by **`gblocky.cc`**.

Coding Strategies

A high-quality solution to this program will take time. It is <u>not</u> recommended that you wait until the night before to hack something together. It is worthwhile to map out a strategy before starting to code. This project has amazingly concise and elegant solutions — aim to make sure yours is one of them!

Decomposition

This program is an important exercise in learning how to put decomposition to work in practice. Decomposition is not just some frosting to spread on the cake when it's done; rather, it's a tool that helps you get the job done. With the right decomposition, this program is much easier to write, test, debug, and document! With the wrong decomposition, all of the above will be a real chore.

As always, you want to design your functions to be of small, manageable size and of sufficient generality to do the different flavors of tasks needed. Strive to unify all the common code paths.

Incremental Development and Testing

Don't try to solve the entire task at once. Even though you should have a full design from the beginning, when you're ready to implement, focus on implementing features one at a time. Those new to programming sometimes think that they should write everything once as if they're writing a history paper, and then go back, revise, cut, edit, and elaborate.

This approach rarely works while writing code. It's much better to implement in stages by inventing lots of little milestones that ultimately lead you to your final product. Each milestone should take the form of a working program that's a little bit closer to where you ultimately want to be. Try to maintain the invariant that your program always does something and that it also works the way you expect it to. If you compile and test often, then you will have a near trivial time isolating and fixing logical errors are they arise. The alternative—a very unappealing one—is to write hundred of lines of code without compiling or testing. This invariably leads to lots of syntax errors, which you fix only to find lots of logical errors, which are even more difficult to fix.

No Global Variables Ever

Although you might feel tempted, **you should not use any global variables** for this assignment. Global variables can be convenient but they have a lot of drawbacks. In this class we will use them only sparingly and with good reason. For any of our assignments, you can assume that global variables are forbidden unless we explicitly tell you otherwise.

Don't Sweat Meaningless Details

Some students worry about details that aren't at all important. I'm interested in good, clean, well-documented code that gets everything done. I don't care whether there's a period at the end of some status message, or whether the delay when advancing from one level to the next is 0.5 seconds or 0.75. Whenever the details aren't clear, just make a decision, document it, and move on. The only time this will get you into trouble is if you make a decision that relieves you of some fundamentally important part of the assignment. ("I wasn't sure whether the fifth color should be called 'Purple' or 'Violet', so I arbitrarily decided that all squares should always be blue.")

Getting Started

Follow the Assignments link on the class web site to find the starter bundle for Assignment 1. You should download this file to your own computer and open the bundle. There is a version for the Mac, and another for the PC. For this assignment, the bundle contains our **`gblocky.cc`** and an incomplete version of **`blocky.cc`**.

The version of **`blocky.cc`** we give you is set up to compile and run, but provides only a drastically broken version of the game. The starter files do, however, illustrate how to interact with the GUI function exported by **`gblocky.h`**/**`.cc`** while at the same time giving you some real C++ to look at.

We've also included two sample applications—one with all twelve levels, and a second with only two. The intent of the second sample application is to give you something that can be easily played from start to finish so you understand the full execution of a working solution (what gets printed when the player successfully completes all levels, for instance.)

You, of course, are supposed to code up the equivalent of the first sample app, but the logical differences between the two-level and twelve-level versions are trivial.

Deliverables

You only need to electronically submit those files that you modified, which in this case, will probably just be **`blocky.cc`**. Details of the electronic submission process are outlined on the course web site. You are responsible for keeping a safe copy to ensure that there is a backup in case your submission is lost or the electronic submission is corrupted.

Good luck!