gtkmm Notes

1. The best source of information on gtkmm is the web site:
   https://developer.gnome.org/gtkmm/stable/
   There is a collection of useful links, as well as some examples from the gtkmm tutorial that have been tested in the student environment, under the "Resources" heading at https://www.student.cs.uwaterloo.ca/~cs246/.
2. The Ubuntu undergraduate machines, `ubuntu1604.student.cs.uwaterloo.ca`, have gtkmm 3.0 installed as the default version. It is possible to install gtkmm on your home machine. Downloads and instructions can be found at http://www.gtkmm.org/en/download.shtml. (There is also information at https://developer.gnome.org/gtkmm-tutorial/stable/chapter-installation.html.en.) If you want to install it under Mac OS X, I'd first recommend finding and installing a package manager such as Homebrew.
3. Your code must be both compiled and linked with the gtkmm library. See the makefiles in the provided examples for how to do this. Note that the `` `pkg-config ...` `` information must be the last item in the command.
4. Your `main` routine must create an initial `Gtk::Application` object in order to initialize the gtkmm environment. It will contain the `Gtk::Window` in which you'll draw everything.
5. The `Gtk::Window` has the thread and runs until terminated, which means that the flow of control has to be from the GUI to the model. Any time the model needs information from the human player, control must go back to the GUI. This implies that there must be repeated calls from the GUI to the model in order to play the game, unless we want to get into running multiple threads, which we don't (that's a topic for CS343). The `Gtk::Window` contains "widgets" that are used to create the user interface.
6. All widgets inherit from the `Gtk::Widget`. A `Gtk::Widget` must be initialized after the main window. This implies that they cannot be `static` variables.
7. A `Gtk::Widget` can only belong to one other `Gtk::Widget` /container. This includes the data in a `Gtk::Image`. The solution to this problem is to create a pixel buffer (`Gdk::PixBuf`) since these can be shared. See the code samples in:
   `gtkmm-examples-3.0/03-menusAndToolbars/ex2` or `MVC-gtkmm3.0`.
8. Containers such as the `Gtk::Window` or the `Gtk::Frame` can only hold one `Gtk::Widget`. A `Gtk::Paned` can hold two (one in each "half", with either a horizontal or vertical layour). A `Gtk::Box` can hold multiple `Gtk::Widget`s, as can `Gtk::ButtonBox` and `Gtk::Grid`. These are the ones you're likeliest to find useful.
9. Containers can be nested, so use a combination of them to create the GUI's look.
10. Rather than create and destroy `Image` objects, change the `Gdk::PixBuf` that they display. You can do the same thing with a `Gtk::Button` displaying a `Gtk::Image`. See the examples.
11. gtkmm has several types of dialogs available. You will probably be most interested in the `Gtk::MessageDialog`, `Gtk::FileChooserDialog`, and `Gtk::AboutDialog`.
12. While you can create your GUI by hand, you may find it easier to start by using a software application, such as glade, which has been installed on the student environment, to generate the necessary structures that are then loaded into your code. You'll then have to set up the appropriate listeners.

13. If you want to block waiting for a dialog to return before returning control flow to your code, you can call `run()`. This function enters a recursive main loop and waits for the user to respond to the dialog via `Gtk::Dialog::signal_response()`, returning the response ID corresponding to the button the user clicked. Before entering the recursive main loop, `run()` calls `Gtk::Widget::show()` on the dialog for you. Note that you still need to show any children of the dialog yourself.
14. If the GUI stops responding, but doesn't actually crash, you may have an infinite loop in your code. You should use a debugger such as gdb to see what is actually happening.