**Purpose of the Assignment**

The general purpose of this assignment is to develop a simple C++ program for a Unix/Linux system (like the [Raspberry Pi Desktop](https://www.raspberrypi.org/downloads/raspberry-pi-desktop/)), given a number of requirements, making use of the principles and techniques discussed throughout the course. This assignment is designed to give you experience in:

* object-oriented programming using C++, using basic language constructs, classes, and data types
* looking through Unix/Linux manual pages and documentation, as you will likely need to do this in your projects later
* getting acquainted with Unix/Linux-based programming and services, which will help in project development on this environment later in the course

The assignment is intended to give you some freedom in design and programming to explore the subject matter, while still providing a solid foundation and preparation for the type of work you will later be doing in the group project.

**Assigned**

Sunday, January 12, 2025 (please check the main [course website](https://westernu.brightspace.com/d2l/home/68122) regularly for any updates or revisions)

**Due**

The assignment is due Thursday, February 6, 2024 by 11:55pm (midnight-ish) through an electronic submission through the [BrightSpace site](http://owl.uwo.ca/). If you require assistance, help is available online through [BrightSpace](https://westernu.brightspace.com/d2l/home/68122).

**Late Penalty**

Late assignments will be accepted for up to two days after the due date, with weekends counting as a single day; the late penalty is 20% of the available marks per day. Lateness is based on the time the assignment is submitted.

**Individual Effort**

Your assignment is expected to be an individual effort. Feel free to discuss ideas with others in the class; however, your assignment submission must be your own work. If it is determined that you are guilty of cheating on the assignment, you could receive a grade of zero with a notice of this offence submitted to the Dean of your home faculty for inclusion in your academic record.

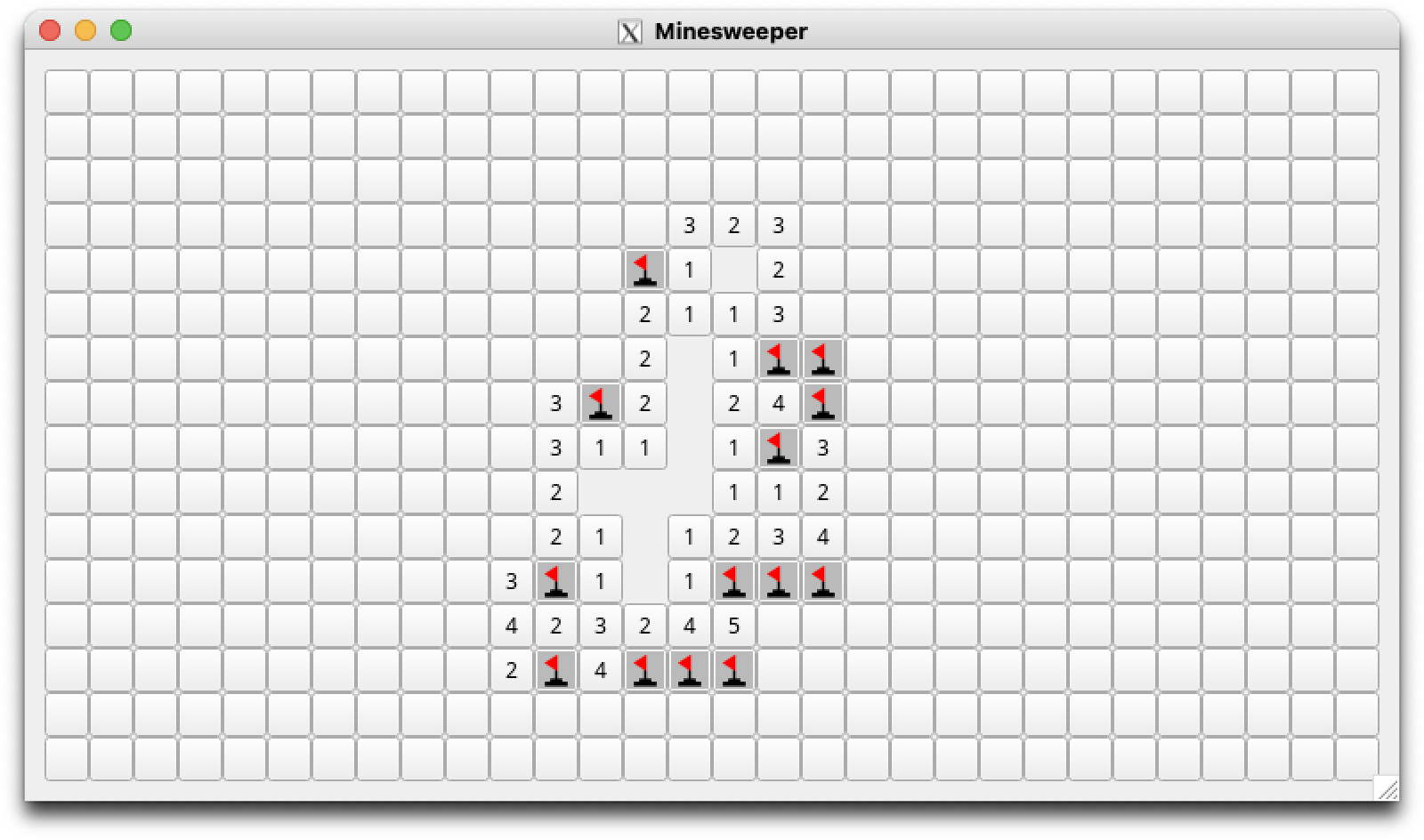
**What to Hand in**

Your assignment submission, as noted above, will be electronically through [BrightSpace](https://westernu.brightspace.com/d2l/home/68122).  You are to submit all source code files, header files, and build files necessary to compile and execute your code.  If any special instructions are required to build or run your submission, you must include a README file documenting details.  (Keep in mind that if the TA cannot run your assignment, it becomes much harder to assign it a grade.)

**Assignment Task**

Your assignment task is to develop a simple version of the classic game [Minesweeper](https://en.wikipedia.org/wiki/Minesweeper_(video_game)), leveraging [Qt](https://en.wikipedia.org/wiki/Qt_(software)) as a graphical framework, and familiarize yourself with development on a Unix/Linux System (like [Raspberry Pi Desktop](https://www.raspberrypi.org/downloads/raspberry-pi-desktop/)).  While this might seem complex on the surface, it's surprisingly straightforward to do so.  This should give you a good exposure to C++ classes, STL packages, external libraries/packages, and more -- all of which will be quite handy for the group project to come!

As an example of how this will look in the end, here's a screen shot of my own implementation.  Yours will have the same functionality, as discussed below, but could look very different depending on how you choose to do things.



**Getting An Environment Set Up**

For this assignment, you will need access to a Unix-like environment of some kind.  Using something like [Raspberry Pi Desktop](https://www.raspberrypi.org/downloads/raspberry-pi-desktop/) will give you something that most closely resembles doing things on an actual Raspberry Pi, but that is not required.  You can use whatever distribution you wish as long as it supports the needs of this assignment.  For Qt, I recommend Qt 5, but you can use whichever version you would like.  The options at your disposal, depend on your computer and operating system:

* If you are running Linux on your computer already, you're off to a good start.  You'll just need the C++ compiler tools and Qt.  The compiler is quite likely installed by default, depending on your distribution, but Qt may need to be installed if you haven't already done so.  (Make sure to install the development packages and not just the run-times so you can create your own programs!)  The packages needed depend on your distribution, but a quick Googling should tell you what you need.
* If you are running MacOS, you are also in pretty good shape, and should be able to run things using [XQuartz](https://www.xquartz.org/).  You will need the command line compiler tools installed, which might depend on [Xcode](https://developer.apple.com/xcode/), as well as Qt, which will likely need to be installed.  (This largely depends on the version of MacOS that you are running, and if you are using [MacPorts](https://www.macports.org/) or [Homebrew](https://brew.sh/) for managing packages.  Googling around will again tell you what you need depending on your setup.)  If you have virtualization software (like [VMWare](https://www.vmware.com/ca.html) or [VirtualBox](https://www.virtualbox.org/)) with a Linux VM handy, that should work just as well too.
* If you are running Windows, it is likely best to virtualize in some fashion to get Linux up and running.  Virtualization software (like [VMWare](https://www.vmware.com/ca.html) or [VirtualBox](https://www.virtualbox.org/)) could work for this, but these days, my preferred approach is to use the [Windows Subsystem for Linux or WSL](https://docs.microsoft.com/en-us/windows/wsl/install-win10).  You can get a number of Linux distributions running under Windows quite readily with this, assuming that your computer supports things well.  Once it's up and running, you just need compiler tools and Qt. (Almost.). You could always use a live CD or live USB key to boot into Linux, but that's often not the most convenient way of doing things.  The other bit you'll need for Windows is an X server to handle the graphics.  If you're using Windows 11 or a recent patch-level of Windows 10, WSL includes graphics support and you should be good.  Otherwise, you'll need to get an X server running like [VcXsrv](https://sourceforge.net/projects/vcxsrv/), to handle the graphics.
* If you are using a Chromebook, you should be able to [get Linux on it](https://support.google.com/chromebook/answer/9145439), depending on the age of your device.  Again, compiler tools and Qt and you're set.
* If none of these are an option or work for you (for example because you have a Windows computer that doesn't support virtualization), we have a Linux server that you can connect into and use, with all the packages installed and ready to go.  Simply use ssh to connect into cs3307.gaul.csd.uwo.ca with your Western credentials and you should be all set.  You'll be doing things remotely, but it should still work fine for this assignment.  Note that you will need to use the -X (X11 forwarding) option when running ssh to get remote graphics working.  Under Windows, this means you will still need an X server running like [VcXsrv](https://sourceforge.net/projects/vcxsrv/), whereas on a Mac or Linux, you can use what you have.  (If you see an "untrusted X11 forwarding setup failed" warning from your Mac, you might need to make some adjustments to your ssh configuration as discussed [here](https://stackoverflow.com/questions/27384725/ssh-x-warning-untrusted-x11-forwarding-setup-failed-xauth-key-data-not-gener).)

Alright!  So, now you have a Unix/Linux environment set up with the necessary compiler tools, and Qt.  (As a bonus, this should also give you a nice dev environment for your group project too!)  You have everything you need to get started with this assignment, so let's dig into the details of what you need to be doing here.

**How Minesweeper Works**

The Minesweeper that we will be making is a simplified, no frills version of the game.  The game board is a 30x16 grid, 30 spaces wide by 16 spaces high.  Hidden under those spaces are 99 mines scattered across the board, with each mine located in its own randomly selected space.  The player's goal is to clear the board without detonating any mines.  To guide the player, hints are given as to the number of mines under neighbouring spaces (left, right, up, down, and diagonal).  Using a combination of deduction and luck, the player selects which spaces to open and explore, and which spaces should be marked as probable mine locations.  If the player successfully uncovers all non-mine spaces without hitting a mine, they win.  Otherwise, when the player finds a mine under a space, they lose.

Some particulars about playing the game:

* The game starts with all spaces showing nothing.  The player has to make their first move  based entirely on luck, as they have no information to guide them.
* Every step along the way, the player has two choices:
  1. The player left-clicks a space to open and explore it.  If a mine was hidden underneath the space, the mine explodes and the player loses the game.  Otherwise, the player is safe for now and is given a hint.  The space either shows a count from 1-8 of how many mines are in adjacent spaces or is blank if there are no mines in adjacent spaces.  In that case, when there are no mines in adjacent spaces, each adjacent space is then also opened, which could lead to a chain reaction of opened spaces with large gaps, as shown in the screenshot above.
  2. The player right-clicks a space to mark it.  If the space was unopened, it is flagged as a mine.  This is useful when the player believes a mine to be in that space.  If the space was already flagged, it is then marked with a question mark (?) to indicate that the player is not sure if a mine is there or not.  Finally, if the space already contained a question mark, the marking is removed and the space is back to its state prior to being marked in the first place.  (Note that a space that is marked can still be opened with a left-click, so the player must still be careful around them!)
* When the player opens the last remaining non-mine space, they have safely made their way through things and win the game!

If you would like a better feel for how the game plays, feel free to check out one of the many implementations out there and available to whatever system you are using.  If you go to the [Minesweeper page on Wikipedia](https://en.wikipedia.org/wiki/Minesweeper_(video_game)), you can also find a link to a version of [Microsoft Minesweeper from Internet Archive](https://archive.org/details/win3_Mineswee) that is playable in a browser.  Our version of course is a bit different, but this might help familiarize yourself with the game if you haven't played before.

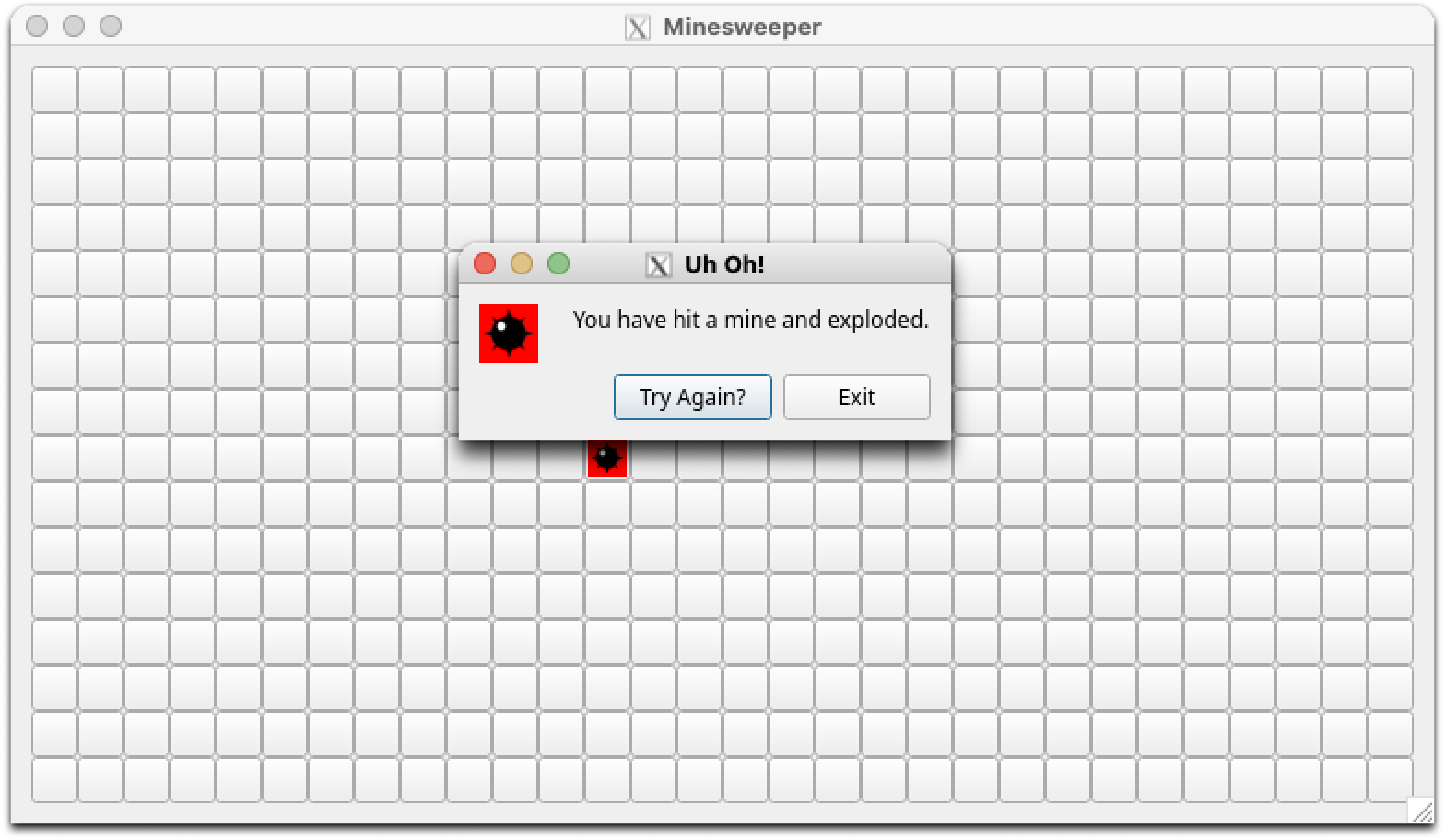
Now that we have a basic understanding of the game, we can look more at how to construct the game using Qt.

**Making Minesweeper in Qt**

As discussed above, you are to use Qt for building your app's graphical interface.  In this case, I recommend Qt 5.  (Or rather 5.something ... whatever is current for your system.)  You should build out the code yourself by hand in C++, rather than using Qt Creator, Design Studio, or QML, so you get a good feel for how things work on their own.  (Besides, your app will be simple enough that those things are overkill, and setting up your game board programmatically is likely easier anyways.)  There are many good tutorials out there for Qt.  [This one](https://doc.qt.io/qt-5/qtexamplesandtutorials.html) provides a good overview of things, with lots of links to various examples, for instance. Personally, I found the [Push Button example](https://wiki.qt.io/How_to_Use_QPushButton) to be quite useful, as it shows how to do most things you'll need without a lot of extra baggage.

Generally, with Qt, you'll have your code and a .pro project file.  You use qmake (or qmake-qt5 or potentially something else, depending on your distribution, like the Fedora we use on cs3307.gaul.csd.uwo.ca) to generate a Makefile, and then use make to build your code.  Any time you change your project (by adding or removing files, for example), you will need to run  qmake/qmake-qt5 to generate a new Makefile.  If you are just editing code, you can simply use make to rebuild just what's been changed.

So, what do you need to have in your interface?  Well, you need the following:

* The main game area.  As noted above, this is ultimately a grid of spaces that you will need to be able to click to play the game.  There are, of course, many ways to implement this and the choice is yours.  You might find it most straightforward to use a [grid layout](https://doc.qt.io/qt-5/qgridlayout.html) as a container for a collection of [push buttons](https://doc.qt.io/qt-5/qpushbutton.html), with each button representing a space.  As push buttons in Qt can contain text and images, they could work nicely for this.  To contain some of the board logic, such as which spaces are adjacent to which other spaces, you might find that you need to create a separate class for this, or extend the button or layout classes to help track the necessary state and information.
* A set of icons for Minesweeper.  For marking spaces and other uses, a set of fairly standard Minesweeper icons is attached to this assignment.  Please feel free to use these or find or make your own if you would rather.
* Popups for when the player wins or loses.  When the player wins the game (by opening the last non-mine space left on the board) or loses the game (by hitting a mine), the player must be given a popup indicating what has happened, giving them two options.  The player can either choose to exit the game or restart the game and play again.  See the screen shot below for an example.  
    
  
* A way to restart the game.  When the current game ends, as noted above, the player is given the option to restart the game.  This should reset all spaces to their initial state, and randomize the hiding and placing of mines under the spaces, giving the player a different game each time.
* A way to hide spaces.  When the player clicks on a space to open it and that space has no neighbouring spaces that contain mines, it needs to be made blank.  While it's not hard to set Qt widgets to be invisible, you might find that by default your layout reflows when doing so.  To avoid this, you need to make sure that your [widgets retain their size when they are hidden](https://stackoverflow.com/questions/10794532/how-to-make-a-qt-widget-invisible-without-changing-the-position-of-the-other-qt).  It's not hard to do, but not obvious either.
* A way to process right-clicks.  Handling left-clicks with a push button is super easy, as shown in the [Push Button example](https://wiki.qt.io/How_to_Use_QPushButton) mentioned above.  Setting up right-clicks can take a little more work, but is do-able.  You might find [this discussion](https://stackoverflow.com/questions/15658464/qt-rightclick-qpushbutton) useful for some tips and ideas.  Once the mechanism is set up, dealing with them is pretty similar to dealing with right-clicks.

**Implementation Notes**

For this assignment, you should package your application logic and user interface components into separate classes as appropriate.  Each class you create should be captured in two files:  one header and one code file.  Your app will also require its own main code file separate from these classes.  As a result, your submission should consist of multiple code files and header files.

Your code should also be written in adherence to the [Coding Guidelines available here](https://westernu.brightspace.com/d2l/le/lessons/68122/topics/2683064). Deviations may result in deductions from your assignment grade up to 10% of its overall value.

In addition to the above files, you must provide a Qt .pro file to assist in the building of your classes and apps.  The .pro file used for the [Push Button example](https://wiki.qt.io/How_to_Use_QPushButton) could be used as a starting point.