C++ London University Session 18

Tristan Brindle

Feedback and Communication

- Your feedback is vital
- Otherwise, we don’t know what you don’t know!
- Please join the #ug_uk_cpplondonuni channel on the cpplang Slack — Go to https://cpplang.now.sh/ for an “invitation”

Today’s Lesson Plan

- A very (very) brief introduction to Qt
- Live coding: using Qt to play sounds
- Exercise: adding a sound effect to our Pomodoro project

Disclaimer

- I’m far from a Qt expert!
- The Qt documentation is excellent: if in doubt, consult it rather than listening to me

A very (very) brief introduction to Qt

- Qt is an extensive collection of software frameworks for writing GUI apps (and more!)
- Originally started in 1991
- Predates standard C++ by many years
- Programming style differs somewhat from “standard” C++

A very (very) brief introduction to Qt

- All Qt objects inherit from the base `QObject`
- Widgets (such as windows, buttons etc) inherit from QWidget, which is a subclass of `QObject`
- Qt adds new “keywords” to C++, like `signals` and `slots`: a preprocessor called `moc` (the “meta-object compiler”) turns this code into standard C++
- In general, the build system will take care of calling `moc` for you

The lifetime of a Qt application

- Like most other GUI systems, the heart of Qt is based around the idea of *events*
- Once initial setup is complete, an application enters the main loop — after this point, events occur which cause signals to be generated, which the application responds to
- Eventually, an event occurs (such as closing the last window) which causes the main loop to stop running, and the program ends

Signals and Slots

- All QObjects can define signals and slots
- Signals and slots can be connected together to create a reactive application
- A signal is “fired” when an event occurs, for example the user clicking a button
- This then calls the connected slot, which performs some appropriate action

Memory Management in Qt

- Qt’s memory management style is somewhat unusual compared to modern C++
- Stack objects and smart pointers are not heavily used; most objects are handled by raw pointers
- The general idea is that when creating a `QObject`, you pass it a pointer to a parent object; when the parent object is destroyed, it destroys its “children” in turn

Live coding: playing sounds in Qt

- https://github.com/CPPLondonUni/QtSoundsExample