telltalegames

Adding QtQuick based windows to an existing QWidget application

Michael Eads
Senior Tools Engineer
Telltale Games

Outline

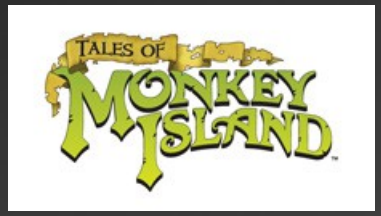
- Introduction
- Application Structure Overview
- Desktop User Interface Elements
  - Actions, Menus, Menubars, Toolbars
  - Quirks
- Application / QWidget Integration
- Conclusion
- Questions

Introduction

telltalegames

Who We Are

- Founded in 2004 by former Lucasarts Employees
- Video Game Developer and self Publisher, releasing on multiple platforms: Xbox 360/One, PS3/4, PC, Android, and iOS
- Primarily Develop Licensed, Episodic, Story driven Adventure Games
- For Example...



telltalegames

What I do

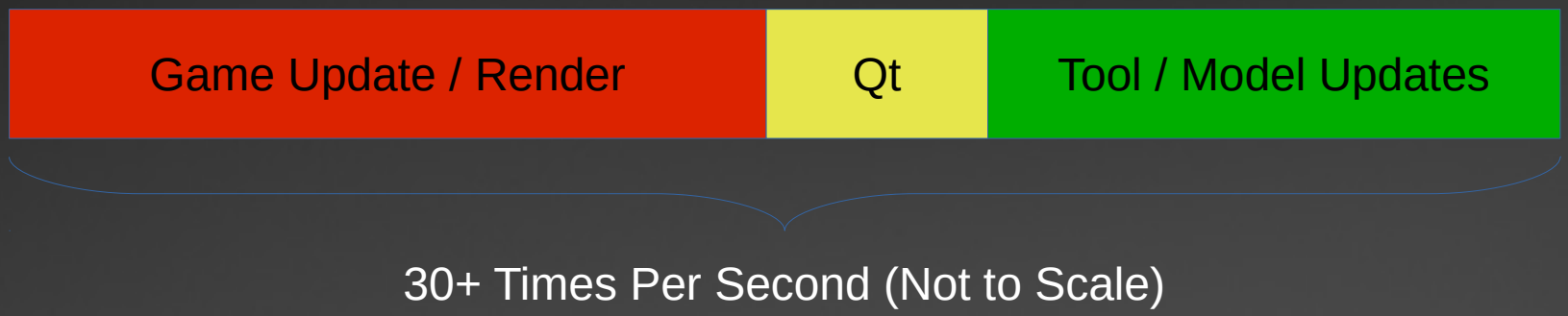
- Make Tools for Making Telltale Games
  - Proprietary (no pictures, sorry!)
  - Windows only (for now)
  - Desktop only
  - Multiple Window Interface
    - Multiple QML and QWidget Windows open simultaneously
    - Several generalized tools (i.e. game data Inspector)
- QML / Quick
  - Graphical display and interaction
- Standard Desktop interface
  - Menus, Toolbars, Shortcuts, etc.
- Tooling running alongside Game Engine

Application Structure Overview

telltalegames

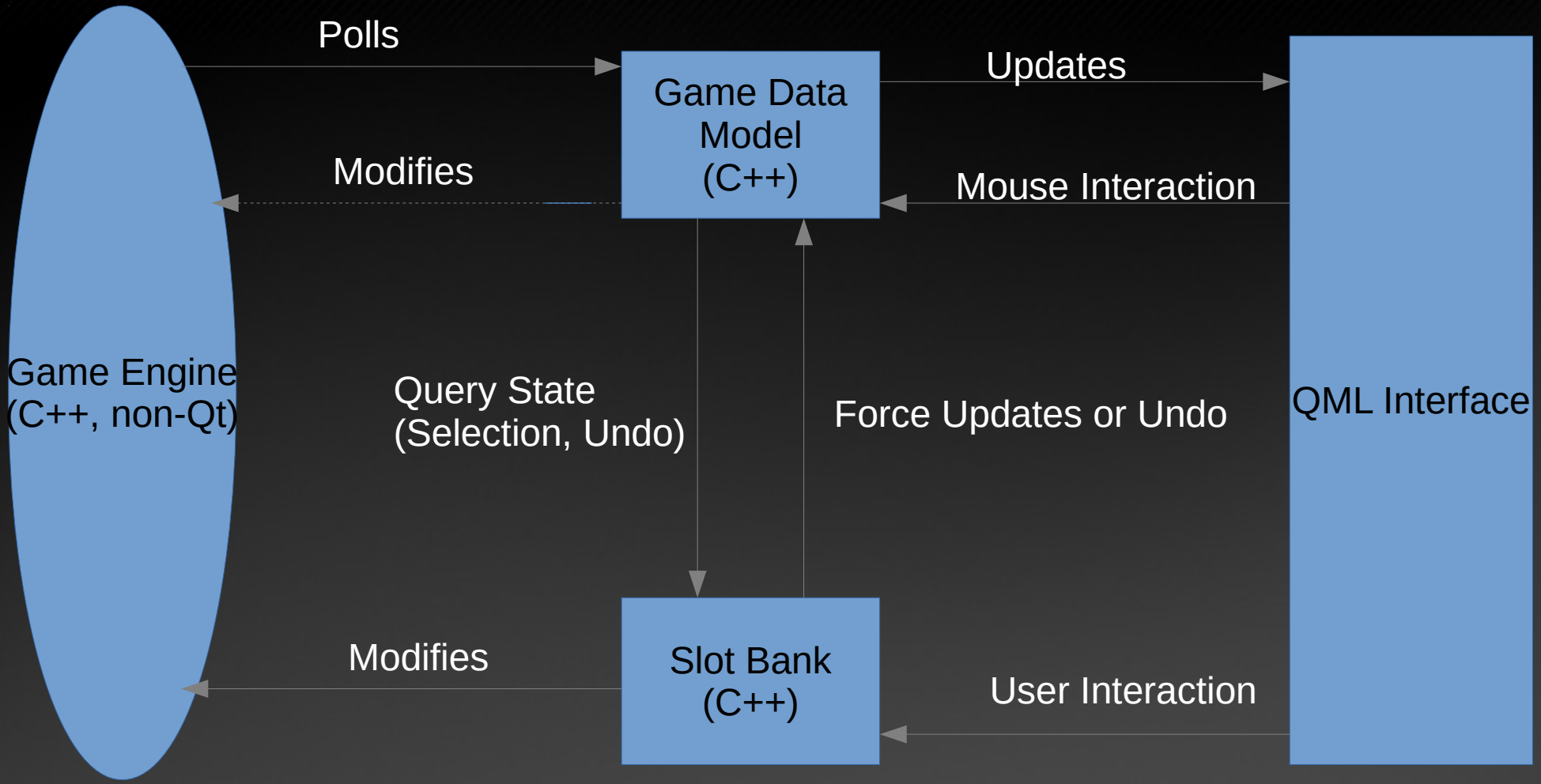
Game Development Tools Environment

- Game Engine Operation + Tool Overhead
- General strict performance requirements
  - 30 FPS+ with tools running
  - Can result in strange trades
    - Model updates at 10FPS for user interaction
    - Certain QML Integration methods unfeasible



telltalegames

General Application Setup



C++ Structure

- Game Data
  - Mostly non Qt / QObject based (internal game engine data)
  - Continuously updating, often without callbacks or notification
  - Data “one way”
    - Affected by game state
    - Game Data changes
    - Model detects and updates
    - QML Responds

telltalegames

Qt Structure

- Polling Model
  - Polls game data multiple times a second
  - Sets model roles for game data changes
  - Manages settings as properties
- Slot Bank
  - Handles most game engine interaction
  - Changes game data directly
  - Model responds on next update loop
  - Can also force immediate update for responsiveness / safety
    - Undo/Redo, Delete

QML Setup

- Nature of QML Data
  - View based
    - QML does visualization of C++ data through model
    - Most interaction through Menu / Toolbar
    - Basically no data entry
  - Mouse interaction primarily
  - Signal out or invoke to C++

QML Setup – cont.

- Data provided through custom Model roles
  - Creates QQMLContext for each QML Component
  - Sets model roles as Context Properties
  - Creates QML Component into this context
  - Updates Context Properties as part of polling
- Mouse interaction
  - Sets top level properties for C++ (mouse point, etc.)
  - Components / ApplicationWindow signal out for interaction

Desktop User Interface Elements

User Interface Elements

- QtQuick Controls
  - QML Items for desktop interface elements
  - Most QWidget interaction elements have corresponding Controls
    - QAction / Action, QMenu / Menu, QToolBar / ToolBar, etc.
    - QComboBox, QLineEdit, more limited
    - All behave slightly different, or have missing elements
- QtQuick Dialogs
  - File, Color, Message
  - No QInputDialog equivalent?

QAction and QtQuick Action

- Action has most QAction properties
  - Text, Checked/Checkable, Enabled, Shortcut, Icon, Tooltip
  - Can be added to QML Items for shortcuts
- Notable missing items
  - Visible: Can't hide everything that references an action by controlling visible
  - Shortcut context?

Extending Action

- Add in Visible property

```
// MyAction.qml
Action
{
    property bool visible: true
}

// MyMenuItem.qml
MenuItem
{
    visible: action.visible
}
```

QMenu and QtQuick Menu

- Like QAction/Action, mostly similar
  - Add MenuItems, Actions, etc.
  - Supports enabled, icon, visible, etc.
- Some missing elements
  - aboutToHide, aboutToShow signals
  - Hinders dynamic menu updates when visibility changes

Extending Menu

- Adding menuAboutToShow Signal

```
// MyMenu.qml
Menu
{
    signal menuAboutToShow

    property bool showingPopup: false;

    onPopupVisibleChanged:
    {
        showingPopup = !showingPopup;
        if(showingPopup)
        {
            menuAboutToShow()
        }
    }
}
```

Extending Menu – cont.

- Works for signaling open
- Allows for dynamic updating
- ...However, onPopupVisibleChanged is private?
  - on__popupVisibleChanged in some versions
  - popupVisible property private
  - Maybe shouldn't be used?

Extending Menu – cont.

- With menuAboutToShow signal, we can dynamically update Menus
  - Can be done through QMetaObject::invokeMethod(menu, “addItem”, Q_ARG(QString, text))
  - Or, create Javascript function in menu

telltalegames

Dynamically Adding Items to Menu

```
// In MyMenu.qml
function addToMenu(stringNames, action) {
    menu.clear()
    var item;
    if(stringNames.length > 0)
    {
        menu.visible = true;
        for(var i = 0; i < stringNames.length; i++){
            item = menu.addItem(stringNames[i]);
            item.action = action;
        }
    } else menu.visible = false
}
```

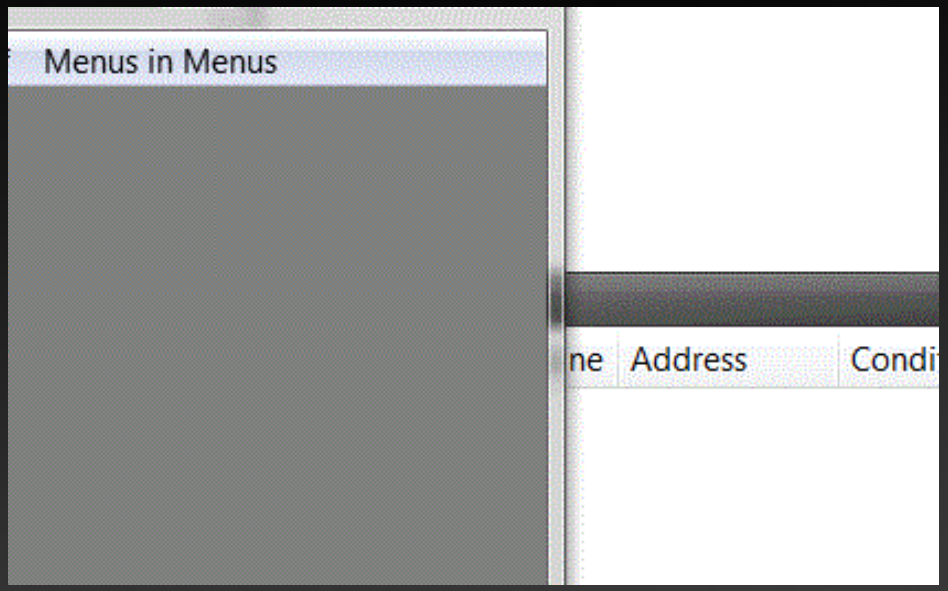
```
// In C++
QMetaObject::invokeMethod(menuPointer, "addToMenu", Qt::DirectConnection,
Q_ARG(QVariant, QVariant::fromValue(itemsToAdd)), Q_ARG(QVariant,
QVariant(actionPointer)));

// If you need to refer or iterate over elements
QQmlListReference menuItems(menuPointer, "items");
for(int i = 0; i < menuItems.count(); ++i)
{
    QObject* item = menuItems.at(i);
}
```

ToolBars vs QToolBar / Quirks

- Controls ToolBar shares almost nothing with QToolBar
  - Uses “ToolButton” which can be assigned an Action
  - ToolButton does not seem to update with Action.tooltip change
  - Basically just a layout
  - No floating
  - No hide / show menu by default
  - No separators
  - ApplicationWindow expects single ToolBar

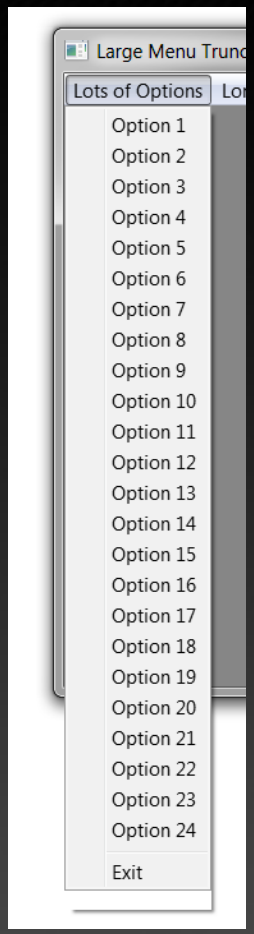
Controls Menu - Quirks

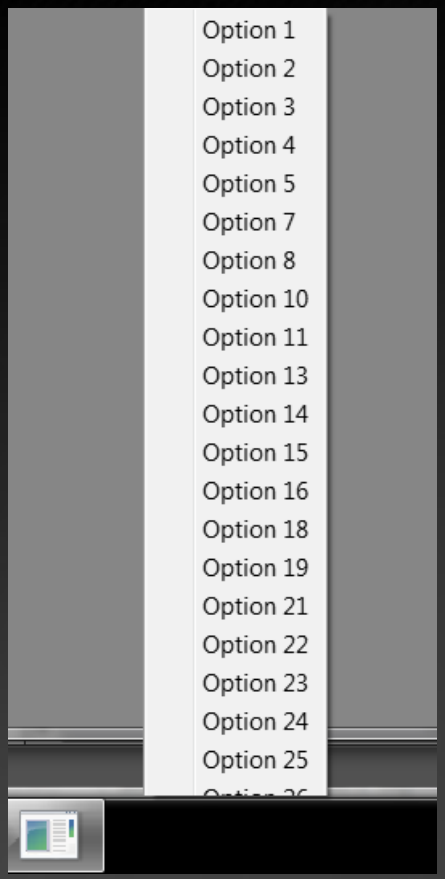
- Slow to open
  - Few tenths of a second on some computers
- Flash or blink on Show
- Clicking a Submenu closes all open menus



Quick Menus - Quirks

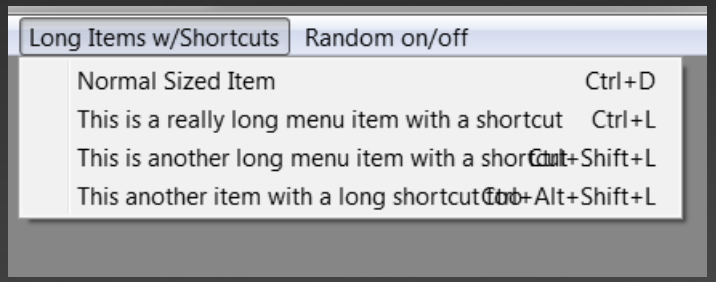
- Height not correct on first show when ~20 Menu Items are used
  - Can be too large, extending drop shadow
  - Can be too small, truncating menu options





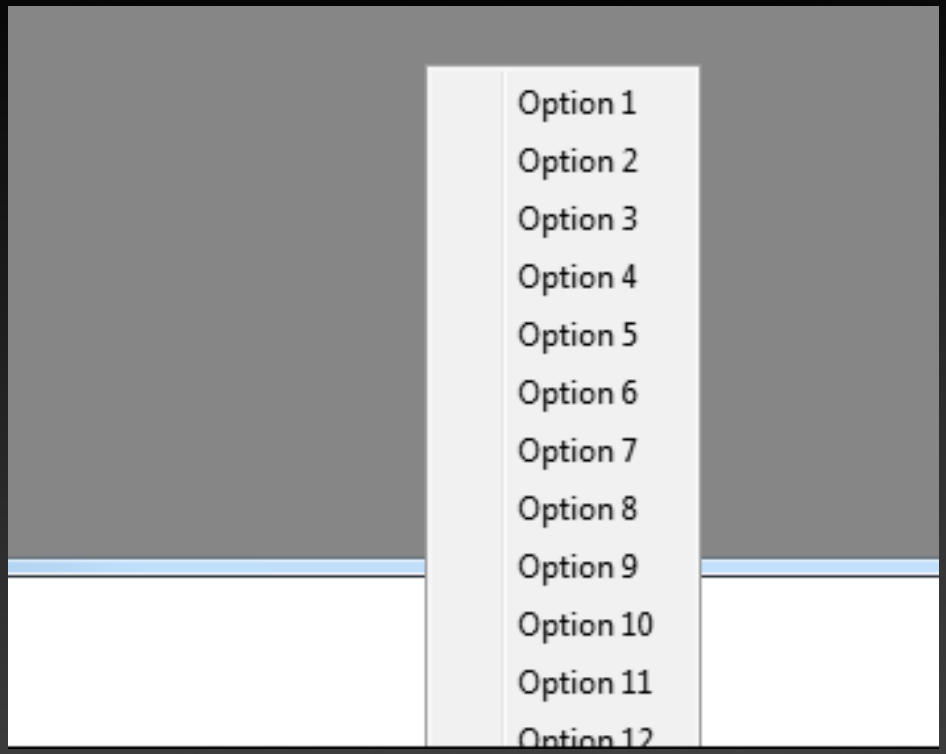
Controls Menus - Quirks

- Width not correct for MenuItems with long text
  - Truncates text
  - Overlaps shortcut listing
    - Can be compensated for by adding padding whitespace



Controls Menu - Quirks

- Menus truncate on multiple monitors
  - Makes some menu options impossible to choose based on position
  - Affects all menus, but especially bad for context menus
  - Only seems to happen with monitors of different resolution, including rotated monitors



Application / QWidget Integration

- Integrating with QWidget Windows
  - QWindow / QQuickWindow / QQuickView
  - QQuickWidget
  - QWidget::createWindowContainer
- Minimizing Controls Usage
- Managing Focus Issues

QWindow / QQuickWindow / QQuickView

- Simplest
- Direct QML with no widget mixing
- Best Performing
- Basically no knowledge of QWidgets
  - No parent / child setup
  - Launching any Widget window will cause “pop under”

telltalegames

QQuickWidget

- Actual QWidget with QML inside of it
- Allows for QWidget parent / child and modal windows
- Allows for QWidget mixing with QML
- Behaves properly with QWidget hierarchy
  - Widget stacks, transparency, etc.
  - Sacrifices some performance for correctness
- Performance Considerations
  - “Minor” performance hit
  - Disables threaded rendering

QWidget ::createWindowContainer

- Like QQuickWidget, allows embedding QML and Widget mixing
- Sacrifices Widget stacking accuracy for performance
  - Works great for entirely QML application where stacking doesn't matter
  - Much better performance relative to QquickWidget for multiple windows
    - For our case: Frame rendering + Model Polling + Widget Overhead
    - Results in near QWindow performance
    - Several FPS over QquickWidget,
    - Oddity of our setup?
- Set QWindow container widget as Focus Proxy for top level QWidget
- Our preferred method for using QML

telltalegames

CreateWidgetContainer Example

```
// Within QWidget derived class

QQmlComponent windowLoader(qmlEngine, this);
WindowLoader->loadUrl(QUrl“qrc://MyQmlWindow.qml”);

QQuickWindow* quickWindow = qobject_cast<QQuickWindow*>(windowLoader->create());

QSize size = quickWindow->size();


QWidget* container = QWidget::createWindowContainer(quickWindow, this);

setCentralWidget(m_widget);
setFocusProxy(m_widget); // focus container widget when top level widget is focused
resize(size);
setFocusPolicy(Qt::NoFocus); // work around QML activation issue
```

Minimizing Controls Usage

- On Windows, controls are Problematic
- Many Small problems
  - Behavior is quite different from platform standard and QWidgets
  - Many minor annoyances
- Some Large Problems
  - Menu Truncation on multiple monitors

Minimizing Controls Usage – cont.

- With QQuickWidget or createWindowContainer standard QWidgets can be mixed in
  - Works especially well for QMenu, QAction, QToolBar, etc.
  - Avoids current problems with Controls
  - Better focus handling
- Causes some other unique challenges

Handling Focus Issues

- For createWindowContainer, focus on container QWidget can be problematic
- In particular, the “activation loop of death”
  - Triggered by launching multiple container widgets in same loop as part of multiple file open
  - Underlying QWindows both “activate” and pump message queue
  - Activation toggles between windows in infinite loop

Handling Focus Issues cont.

- Avoiding activation infinite loop
  - Set container QWidget's focus policy to NoFocus
  - Set focus onto it later manually or when needed
  - QML Focus will still operate its own way

Handling Focus Issues- cont.

- QMenu Focus Problems
  - i.e., QMenu::popup for context menus over QML area
  - Can be fixed by setting focus back on QMenu::aboutToHide:

```
// In .cpp, along with QMenu creation
QObject::connect(
    m_myMenu, &QMenu::aboutToHide,
[=]() { m_container->setFocus(); }
);
```

Conclusion

- Choose between either QQuickWidget or createWindowContainer
- Embed QML and use it for what its needed for
- Use QWidgets for desktop interface components
- For Windows Desktop at least
- One last thing...

telltalegames

We're Hiring!

- Tools Engineer - Qt (Core Technology)
  - 3+ Years of C++ Experience
  - 1+ Year of Qt Experience
  - Primarily QWidget development focused
- Come pick up a job flyer
- Job website: http://www.telltalegames.com/company/jobs/
- Meet our Recruiters at the bar after the conference on the 5th

telltalegames

Questions?