HCI & GUI Programming
Event Driven Model

Alex Cronin

alex.cronin@griffith.ie

The Event Driven Model

- In this lecture we will explore the event driven model of programming and how it differs from the programming model you have learned thus far
  - All GUI toolkits are based on the event driven model
  - Meaning that the principles here that you learn for JavaFX will transfer over to other toolkits such as GTK+, Qt, WxWidgets, Java SWT etc

The Event Driven Model

- Most of the programs you will have written thus far will have followed this general format
  - Create a class
  - Add in the main() method
    - Write all your operations here
      - Can be in the form of instructions or function calls etc
  - Run the program
    - Starts in the main() method and finishes when all code of the main() method has been executed
  - These kinds of programs have very well defined start and end points.

The Event Driven Model

- As was stated in the previous lecture all GUI toolkits use the event driven model of programming
- The event driven model of programming wherein the execution of a program is determined by events generated by a user or any other input (sensors etc.)
  - The majority of GUI toolkits are based on this model of programming

The Event Driven Model

- In an event driven model there must be an implementation of an event loop
  - With GUIs this is usually embedded inside the GUI toolkit itself and is never exposed to the developer.
  - Some GUIs even permit you to write your own event loop
  - Particularly if you are working with the Win32 API or X Windows directly.

The Event Driven Model

- The Event Loop will have this general structure

```
while(x = nextEvent())
    if(x != quit)
        process event x
    else
        break
```

  - This loop will run forever until it recieves an input that is asking the model to stop processing
  - However, there is an interesting behaviour associated with the nextEvent() method

The Event Driven Model

- The nextEvent() function queries an event list that is attached to the application.
- The event list is a FIFO datastructure that is maintained by the underlying display manager (Windows, Xwindows, OSX)
- The nextEvent() function will perform one of two actions depending on how many events are in that list.
- If there is one or more events in the list
  - Then the first event is removed from the front of the list and returned from the nextEvent() function

The Event Driven Model

- If however, there are no events in the list then the nextEvent() function will block and put the GUI application in a cold wait.
  - A cold wait is where the application process is put into a sleep state, such that it will not consume CPU cycles
  - The nextEvent() function will wake when a new event comes in for this application from the window manager.

- Why are cold waits needed
  - Before cold waiting GUI applications would continiously poll the keyboard and mouse for events many times in a second.

The Event Driven Model

- Thus CPU usage would always be 100%
- Many wasted cycles in a second
- This is known as busy wait
  - When you get to learning about threads and process management you really should learn how to go into a cold wait and resume from it.
  - Frees up cores for other GUI programs or running tasks
- There are certian cases however where a cold wait will not be useful to have.
  - Mainly video games or interactive demos

The Event Driven Model

- Video games or interactive demos will follow a different event loop instead of the following form
- while(true)

```
if(peekEvent())
        x = nextEvent()
            if(x == quit)
                break
        process x
else
        idle()
```

- Here we check to see if there is anything on the event queue.

The Event Driven Model

- If there is nothing on the queue we will call an idle method instead
  - Will do some processing to keep the application moving along in the absence of events
    - e.g. update physics, make AI decisions, change game state etc.
- The peekEvent() method will return the event at the front of the list without removing it.
  - If there is no event at the front of the list then an empty event or a null reference will be returned to indicate this.
  - If there is an event present we will remove it from the front of the list and process as before

How the Event Driven Model is Implemented in GUI toolkits

- How does this affect you with GUI programming?
  - GUI toolkits such as JavaFX, Qt, GTK+ will all implement the event loop inside there repesective libraries
  - Meaning you do not have direct access to the event loop
  - However, each toolkit will expose a number of event handlers that you may connect with in your own applications

How the Event Driven Model is Implemented in GUI toolkits

- Each event handler will expose a different type of event.
- There are multiple events that can be handled by a GUI here are a list of some such events that you will come across
  - Keyboard events
    - Key press: A key has been pressed down and is held down on the keyboard
    - Key release: A key that was held down has just been released
    - Key typed: a key has been pressed and released in quick succession

How the Event Driven Model is Implemented in GUI toolkits

- Mouse events
  - Mouse press, mouse release, mouse clicked: similar to the key events above
- Touch events
  - Starting to come into GUI toolkits now as touch monitors become prevalent
  - Similar in style to mouse events
- Display events
  - Redisplay: asks the GUI to rerender itself on screen
    - This can be in response to the GUI being hidden then subsequently shown again.
    - Or another GUI has obscured this GUI but is no longer obscuring therefore there is a need to rerender.
    - In the case of games this will be called after each iteration of the idle method.

How the Event Driven Model is Implemented in GUI toolkits

- Display events
  - Resize:
    - The size of the window has changed therefore layouts and widgets need to be assigned new sizes and new positions.

– All GUI toolkits will expose these events through event handlers.
  - All GUI widgets will expose these event handlers
  - By default the event handler will do nothing
  - If you provide an implementation of an event handler then you can react to events that happen in your GUI interface

– As an example we will use the Button class that is provided by JavaFX

How the Event Driven Model is Implemented in GUI toolkits

- The button class exposes a single event type called OnClick
  - By default it will do nothing
- However when you provide an implementation of the handle method (by implementing the EventHandler look at your JavaFX notes) for a Button
  - When the user clicks the button the event loop wakes up
  - Determine which button was clicked and will call the appropriate event handler.
  - Your application carries out actions in response to user events

How the Event Driven Model is Implemented in GUI toolkits

- You are now programming reactively
  - Thus you must always determine what happens if a user clicks a button or changes a checkbox
  - Every single control should have an effect on the overall application state or should display information to the user.
  - If the control makes no change to the application state or does not inform the user then it should not be there
    - Wasting valuable screen space that can be used better
  - The only time your GUI will become active is when the user causes an event in your application.
    - Could be a key press, mouse press, touch event, etc.

Example of an event in JavaFX

- Consider the following snippet of code from one of the JavaFX examples

```
// add a listener to the button to react to a click by the user. this should increment

// the number by 1 and refresh the screen

btn.setOnAction(new EventHandler<ActionEvent>() {

    // must override this method for an action event event handler

    @Override

    public void handle(ActionEvent event) {

        clicks++;

        label.setText("This button has been clicked " + clicks + " times");

    }

});
```

How the Event Driven Model is Implemented in GUI toolkits

- In this example there is an event hander being defined for a button.
- The only time this method will be called is when the button (variable btn) is clicked
- As soon as that click is registered then the method handle() will be executed immediately in response.
- This method will update a click counter and will update a label (information for the user) to reflect this change
- Upon finishing the method the application will go back to a cold wait

How the Event Driven Model affects you

- You must consider the following when you are writing GUI programs
  - What information should the user provide your application?
    - Determines what kind of input controls you will need
  - What should your application do when an event occurs
    - Must determine two things
    - What type of event.
    - Which control.

How the Event Driven Model affects you

- e.g. a click on a yes button should differ from a click on a no button
- A text field for a user name should display the entered characters whereas for a password each character should have the same symbol
- If you have to get the user to select a single option you should use a radiobutton or combobox

– Another way of thinking about this is to describe what actions you want your user to perform and then build your interface around that set of actions

How the Event Driven Model affects you

- E.g. if you have a ToDo list application you might have the following actions
  - Add todo
  - Delete todo
  - Mark todo as complete
  - Move todo up the list
  - Move todo down the list
  - Etc

How the Event Driven Model affects you

- E.g. if you have a unit converter app you might have the following actions
  - Select unit type (length, weight, area, etc)
  - Select a convert from unit
    - e.g. if length the user can pick: meters, yards, miles, kilometers ...
  - Select a convert to unit
  - Input a value to convert from
  - Have a method for the user to indicate that they would like to covert the current value
    - Either a seperate button or an event on the input of the convert from value.

Security and Robustness in event driven apps

- As a small side note you should take great care to validate all input that comes into a GUI application particularly from a user.
- Take note of this quote from Rick Cook's novel “The Wizardly Compiled”
  - “Programming today is a race between software engineers striving to build bigger and better idiot-proof programs, and the Universe trying to produce bigger and better idiots. So far, the Universe is winning.”

Security and Robustness in event driven apps

- If you do not validate your user input then it can cause your application to crash (divide by zero, index out of bounds, etc)
- Or worse yet it the correct sequence of inputs are entered it is possible to comprimise the application and execute any code the user likes.