Layouts, Widgets, and Stacking Order

Introduction

- In the previous lecture we discussed the event driven model
  - As the basis of all GUI program development.
  - In this lecture we will now discuss mechanisms for organising our GUI display, presenting information to the user and recieving events from our user
  - Before we cover that however, we will continue with some of the theory behind GUI toolkits namely the stacking order of windows

Introduction

- At the moment we have a very good understanding of the event loop and its role in the maintenance of GUI applications
  - However, we need more than the event loop we need to have mechanisms for displaying controls to the user and recieving their input.
    - The event model is useless, unless there is something to generate events and inputs to that model.
  - We also need mechanisms for organising these controls
    - Not every display will be of the same shape and size.
    - Users will determine how much space your application will take on their screen at all times
    - Thus if the window size changes, your application must change with it.

Introduction

- In this lecture we will explore how these problems can be solved in GUI toolkits.
  - We will come across the stacking order for determining which windows are ontop and on the bottom
    - Will also be used to determine the owners of events.
  - We will also discuss widgets/controls
    - To display information to users and also to recieve their input.
    - Here is where our event handling will be exposed

Introduction

- In this lecture we will explore how these problems can be solved in GUI toolkits.
  - Finally we will discuss layouts
    - Will be used to determine how widgets get space should the window be resized.

Stacking Order

- Lets assume that we have created a window manager that can generate GUI components.
  - However, it can only show windows side by side but never on top.
  - What if there was a way that we can organise windows in such that we can make them overlap
  - Well one such method is to use a combination of a window stack and painters algorithm to render multiple overlapping windows
  - However, the stack that is used here is not a normal stack datastructure

Stacking Order

- A normal stack is a LIFO data structure
  - Elements can only be added or removed from the top
  - In the stack for a window manager elements must be removable from anywhere in the stack so they can be placed on the top.
    - The window that has focus and is active (i.e. the user's attention) should be at the top of the stack at all times.
    - All windows maintain their positions and bounds on the display.
    - All windows that were above this window will subsequently move down one place in the stack.

Stacking Order

- That takes care of the modelling of overlapping windows
  - The rendering is a different issue.
    - How do you render a stack of overlapping windows?

Stacking Order

- You render from bottom of stack to top of stack
  - One window at a time
  - This is known as painter's algorithm
    - The analogy here is that a painter will draw the background first, then the middle ground and finally the foreground
    - All on a 2D canvas
  - Thus these two modifications give us a stacking window manager that is capable of rendering and displaying overlapping windows

Event Handling

- At the moment though
  - There is no process for determining what window owns any given event
  - Fortunately our stacking order will prove useful here
  - Along with bounding box tests.
  - However the logic for handling keyboard events will be different from mouse events
  - Implementing this logic will make a more useful window manager

Keyboard Event Handling

- Keyboard event handling is simple
  - Look for the window that has current focus
    - Usually the window on top of the stack
    - But depends on whether the window will accept focus
    - And the window manager's policy for focus with new windows.
  - Whichever window has focus will recieve all keyboard input
    - If the window does not handle the key event then the window manager may discard it or respond to it
    - Again these are rules decided by the window manager

Mouse Event Handling

- Mouse event handling on the other hand is a bit more difficult
  - We now have a pointer that can be anywhere on the entire screen.
  - Thus it is entirely possible that the user may click on a window that is not on top of the stack
    - Or may not click on a window at all.
  - How do we go about handling these cases?

Mouse Event Handling

- In the first case we have
  - Determine the global coordinates on screen where the user clicked.
  - We then perform a bounding box test on each window in the stack to see which window encloses the click
    - Test from top of stack down to bottom.
    - The first window that contains the click is the window that will own that click event.
    - Thus the click event will be sent to the event handlers of that window and the window will move to top of the stack

Mouse Event Handling

- That only handles the first case
  - However what happens if no window owns that click
  - i.e. the click does not fall in the bounds of any window in the stacking order.
  - It is up to the window manager to then decide what to do with that click
    - Most window managers will interpret a click outside of windows in some way
    - e.g. right clicks will bring up context menus

Event Handling

- So at this point we now have a window manager that can handle overlapping windows
  - And can also handle keyboard and mouse events.
  - That takes care of top level windows however we need to have mechanisms in place that will determine what widget in a GUI window will own a particular event
  - How do we go about doing this?

GUI Widget event handling

- In a GUI application we have a tree data structure called the visual tree.
  - It is made up of all layouts and widgets that belong to a top level window.
  - The application will perform tests to see which widget owns the event
    - Will work from the root of the tree to the leaves.
  - When the widget is found that widget will be given the event by calling it's appropriate event handler
    - The widget will process this event, update itself and if necessary generate and fire more events in response.

GUI Widget event handling

- For example if a button is clicked
  - The mouse click event handler is called
  - The button registers this click
    - Redraws itself on screen
  - Then fires an ActionEvent to an attached EventHandler so the GUI can respond to that click
  - It will then notify the GUI that the event has been consumed and requires no further processing.

GUI Widget event handling

- This is the full complete cycle of how a window manager gets an event, sends to a GUI application and enables a response to that event
  - Widgets take basic interaction through keyboard and mouse and provide context to those interactions.
- We now have a fully working window manager with event targeting for individual widgets
  - All that is left to sort is how to size and place widgets in the window

Layouts

- How would we do this?

Layouts

- A naive first solution would be to give each widget a position and a size in the window
  - Only works with fixed sized windows
  - Increase in display pixel density will cause the app to suffer.
  - Incapable of scaling up or down
  - No method to allocate extra space.
  - Thus we need some way to determine where widgets are placed and how they are sized
    - How do we do this?

Layouts

- Through a combination of rules and mathematics
  - Rules
    - To determine where widgets are placed and in what order
    - Which widgets are restricted in size
    - Which widgets can expand to take space
  - Mathematics
    - To give precise pixel values for position and size.
    - To produce a visually appealing order to widgets
    - Takes the form of algebraic equations and weighting systems

Layouts

- With these two things in place
  - A GUI program is capable of determining where every widget should be placed
  - And how much space it should be given
  - It allows the GUI program to scale up and scale down on a window resize operation.
  - Thus a GUI program can now take advantage of extra pixels should a user allocate them to the GUI window
  - All thats left to solve is how the widgets should be arranged

Layouts

- Arrangement of widgets
  - There is an infinite number of ways in which widgets can be arranged
  - Only a set few of these ways are used commonly throughout GUI programming
  - Why?

Layouts

- The human brain and visual system loves order and patterns
  - There is an infinite number of ways in which widgets can be arranged
  - Only a set few of these ways are used commonly throughout GUI programming
  - Why?

Layouts

- Patterns are easy to discern and remember
  - Horizontal line, vertical line, grid, border arrangement
  - Distinctive easy patterns
  - Because all potential users understand these patterns
    - Many applications will take advantage of them
    - Why?

Layouts

- Because it reduces the learning curve of GUIs significantly.
  - You are reusing existing knowledge of your users
  - Requires less mental effort to learn your application and interface.

Conclusion

- At this stage you now have a complete high level understanding of
  - How a window manager achieves window overlap
  - How a window manager determines what application owns an event
  - How an application will determine what widget owns an event
    - And subsequently processes it
  - And also how layouts work internally