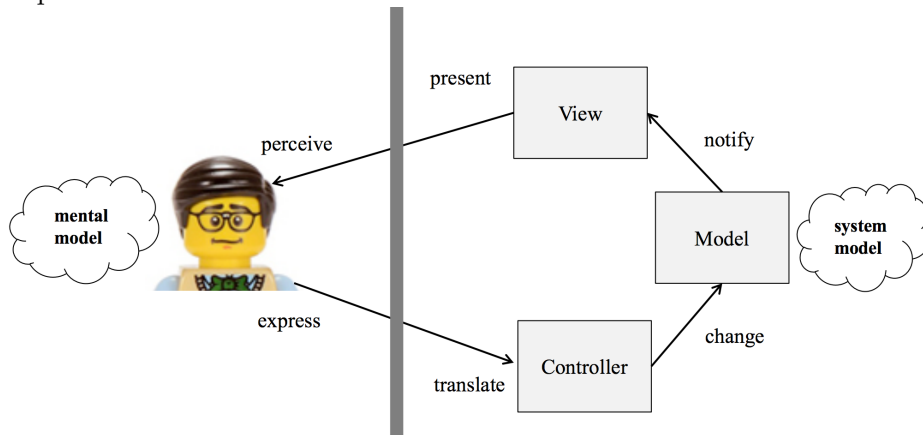# CS 349 Midterm Review

## Background & History

- **User interface**:
  - The place where a person <u>expresses intention</u> to an artifact, and the artifact <u>presents feedback</u> to the person
  - The way people (mental model) and technology (system model) interact
 - Represented as MVC:

- **Interface**: external presentation (visual, physical, auditory) to the user
  - e.g. controls
- **Interaction**: actions invoked by user and corresponding responses (behaviour)
  - e.g. action and dialog
- Batch interfaces (1945-1965)
  - Sets of instructions fed via punch cards
  - Only used by highly trained individuals
- Conversationalist interface (1965-1985+)
  - Text-based feedback and input
  - I/O is in system language, not task language
  - Vannevar Bush – created the memex, a desk with integrated display, input, and data storage
  - Ivan Sutherland – created the Sketchpad, an early graphical interface with a light pen and direct manipulation
  - Douglas Engelbart – invented the mouse, introduced copy/paste
  - Alan Kay – worked on the Xerox Star, first commercial computer with GUI
- Graphic user interface (1984+)
  - Hardware interface: high resolution & refresh graphics display, keyboard, and pointing device
  - WIMP interface: windows, icons, menus, and pointer
  - Benefits of GUI:
    - Keeps the user in control
    - Emphasize recognition (discovery of options) over recall (memorizing commands)
    - Uses metaphor; makes interaction language closer to user's language

Windowing Systems & X11

- **Windowing system**: provides input, output, and window management capabilities to the OS
- **X Windows (X11)**:
  - Standard windowing system for Unix-based systems
- X11 architecture
  - **X Client** handles all application logic
  - **X Server** handles all user input & display output
  - There may be many clients – each client is an application; server draws all clients onto one screen and reads all input
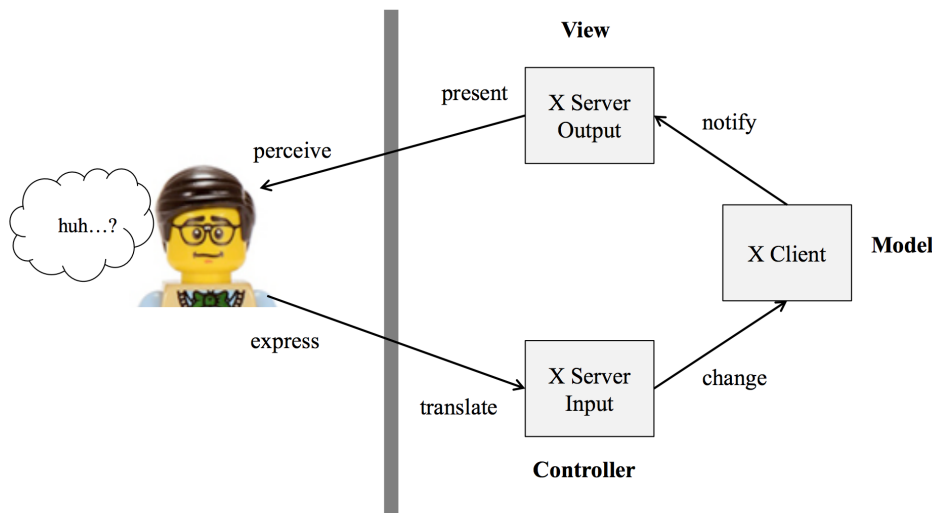

- Structure of an X program (application is run on the X client):
  - Perform client initialization
  - Connect to X server (e.g. `XOpenDisplay()`, `XCreateWindow()`)
  - Perform X related initialization (e.g. create graphic contexts with `XCreateGC()`; put window on the screen with `XMapRaised()`)
  - Event loop
    - Get next event from server (e.g. `XNextEvent()`)
    - Handle event (e.g. `XLookupKeysym()`)
    - Send draw request to server (e.g. flush output buffer with `XFlush()`)
  - Close down connection to X server (e.g. `XCloseDisplay()`)
  - Perform client cleanup
- X11 is a **base windowing system**:
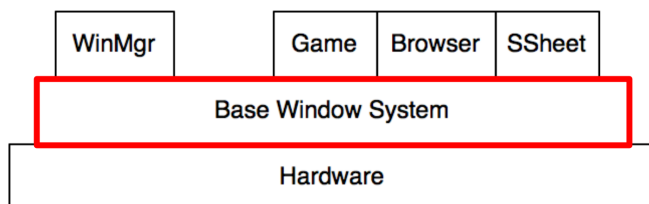
  - A standard/protocol for creating windows, low-level graphical output, and user input

- Does not specify the style of each application's UI
- Provides each application with a window and manages its access
- Each application (only) owns a <u>canvas</u>; shielded from details such as visibility, other windows, etc.
- Some <u>design goals</u> of X11/BWS:
    - Supports multiple overlapping & resizable windows
    - A display may have multiple screens (e.g. monitors) and a window may span multiple screens
    - High-performance, high-quality text, 2D graphic & imaging

- **Window manager**:
    - Provides interactive components (e.g. menus, close button)
    - The WM owns each application's window itself (while application owns the canvas)
        - i.e. application developers usually cannot change the window style
    - Separation of the WM from the BWS enables many alternative "look and feels"
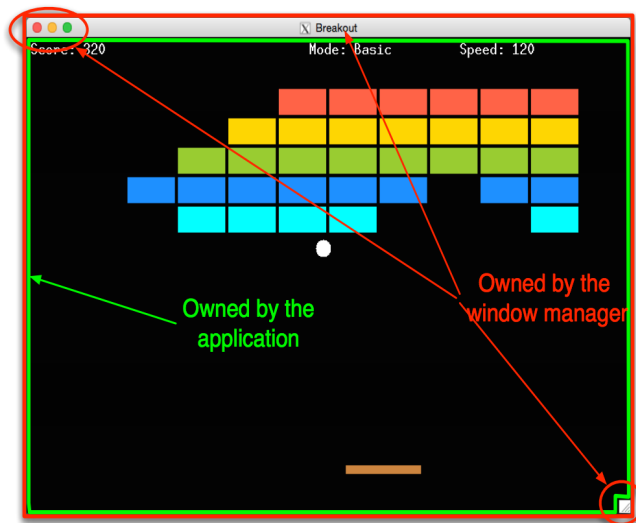
- **Drawing**
    - Three conceptual drawing models:
        - Pixel (e.g. images)
        - Stroke (e.g. lines, outlines of shapes)
        - Region (e.g. text, filled shapes)
    - X11 uses <u>graphics contexts</u> to store drawing options/parameters – stored on X server
    - **Painter's Algorithm**: draw shapes in layers from back to front to create composite shapes
    - Implementation in X11:
        - `Displayable` class with abstract `paint()` method
        - Implement `paint()` in each subclass
        - Draw list of `Displayables` from back to front, clear screen on every repaint
- **Events & animation**
    - Objective: need to map input from real-word devices to actions within a system
    - <u>Event-driven programming</u>: flow of program is determined by <u>events</u> such as user input (key press, mouse click, input focus change) or sensor/timer events

- Implementation in X11:
  - Use `XSelectInput()` and event masks (e.g. `KeyPressMask` etc.) to register for types of events
  - Use `XNextEvent()` to dequeue the next event; <u>may block if no events</u>
    - Use `XPending()` to check for # of events before dequeueing
  - Should dequeue *all events* before repainting to avoid input lag
  - Should subtract time spent in event loop from `sleep()` to maintain consistent FPS
  - Should draw all images to a *buffer* (`XCreatePixmap()`), then copy the buffer onto the screen in one go (`XCopyArea()`) (aka. <u>double buffering</u>)
    - Avoids displaying an intermediate image (i.e. flickering)

Modern GUI

- **Widgets**
  - ∎