THE UNIVERSITY OF AUCKLAND

# COMPSCI 230

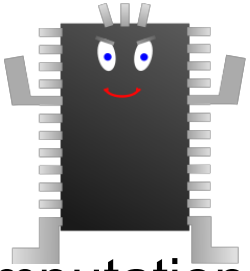
## Software Design and Construction

Game Example

2013-05-13

# Swing and Threads

# Threads

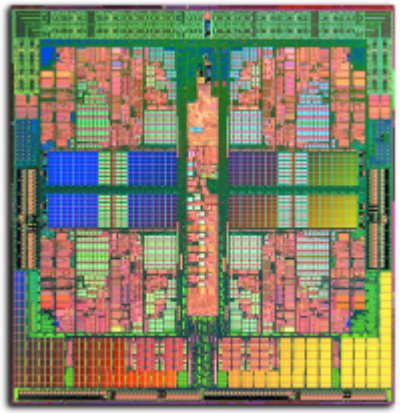


**Thread** of program execution: computational resource that executes instructions of a program, usually consisting of...

- Program counter: pointer to next instruction to be executed
- Call stack with local variables and return addresses
- Set of registers for storing data directly in the processor

**Multi-threaded program**

- A program that uses multiple threads to execute several portions of code concurrently
- Each thread can potentially run on a different CPU / core



Quad-Core AMD Opteron

# SWING AND THREADS

- One event-dispatching thread
- All event handlers run by event-dispatching thread
- **Single-thread rule**: all code accessing a widget (after creating it) should be executed in the event-dispatching thread
  - Swing is not thread-safe
  - Only a few widget methods that are thread-safe
- **Responsiveness rule**: don't do time-consuming tasks in event listeners; create new threads instead

**Responsiveness Rule**

**Single-Thread Rule**

# Using Worker Threads for Responsiveness

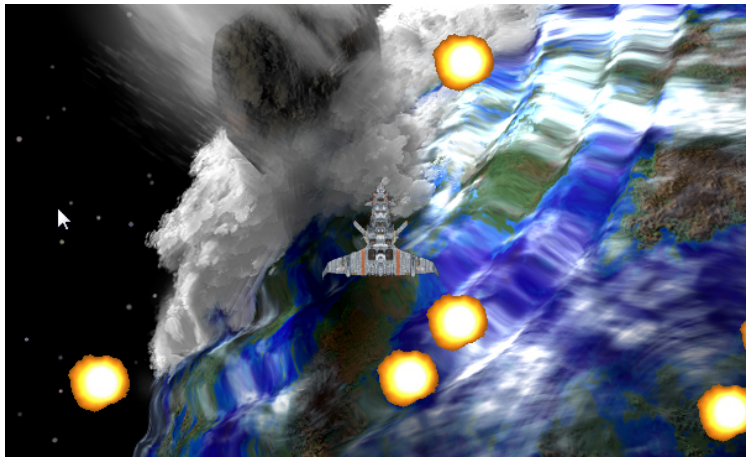
```
...
JButton button = new JButton("Invoke");
button.addActionListener(new ActionListener(){
    public void actionPerformed(ActionEvent e) {
        Thread t = new Thread(new Runnable(){
            public void run() {
                // Do some long operation...
                // (computation or I/O)
                // Change model; view will follow...
            }
        });
        t.start();
    }
});
...
```

# Doing GUI Changes on the Event Dispatching Thread

```
…
Thread t = new Thread(new Runnable(){
    public void run() {
        // Cannot access GUI directly from here (unsafe).
        // Let the event dispatching thread do it:
        SwingUtilities.invokeLater(new Runnable() {
              public void run() {
                // access the GUI here
          }
        });
    }
});
t.start();
…
```

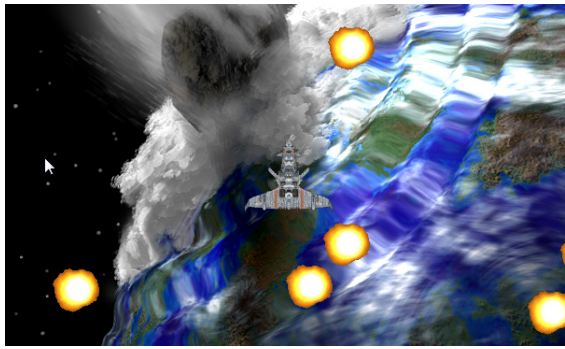
**Alternative:** **`invokeAndWait()`** waits until the event dispatching thread has executed the **`run()`** method

# Game Example
# `MeteorField`



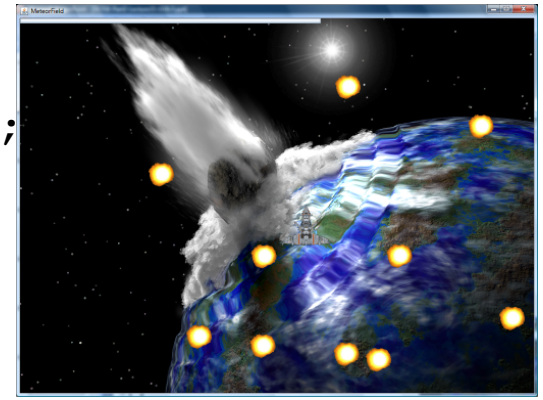
# `MeteorField` Part 1
# Main Class

```
public class MeteorField extends JFrame {
   final static int numOfMeteors = 10;
   static ImageIcon image = new ImageIcon("background.jpg");
   public static int width = image.getIconWidth();
   public static int height = image.getIconHeight();
   public static Hero hero;
   public static JLabel hitLabel; int hits = 0;
   public static Meteor[] meteors;
   public static boolean up = false, down = false,
                         left = false, right = false;

   public MeteorField()    {
          // see next slides…
   }

   public static void main(String[] args) {
        // see next slides…
   }
}
```



# `MeteorField` Part 2: GUI Setup

```
public MeteorField()     {
    setTitle("MeteorField"); setSize(width, height);
    setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    getContentPane().setLayout(null);
    JLayeredPane pane = new JLayeredPane();
    pane.setBounds(0,0, width, height);
    JLabel background = new JLabel(image);
    background.setBounds(0,0, width, height);
    pane.add(background, new Integer(1));
    meteors = new Meteor[numOfMeteors];
    for(int i =0; i<numOfMeteors; i++) {
       meteors[i] = new Meteor();
       pane.add(meteors[i], new Integer(3));
    }
    hero = new Hero(); pane.add(hero, new Integer(2));
    hitLabel = new JLabel("0");
    hitLabel.setBounds(10,10,50,50);
    pane.add(hitLabel, new Integer(4));
   getContentPane().add(pane); setVisible(true);
    // continued in part 3…
```

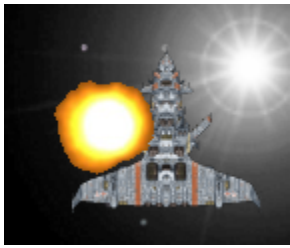


# `MeteorField` Part 3
# Event Handler Setup

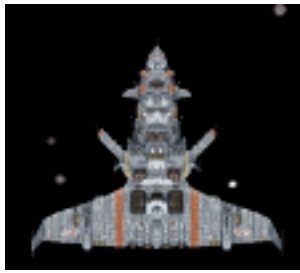
```
addKeyListener(new KeyAdapter() {
    public void keyPressed(KeyEvent evt) {
        if(evt.getKeyCode() == KeyEvent.VK_UP) {
            up = true; down = false; }
        if (evt.getKeyCode() == KeyEvent.VK_DOWN) {
            down = true; up = false; }
        if (evt.getKeyCode() == KeyEvent.VK_LEFT) {
            left = true; right = false; }
        if (evt.getKeyCode() == KeyEvent.VK_RIGHT) {
            right = true; left = false; }
    }

    public void keyReleased(KeyEvent evt) {
        if(evt.getKeyCode() == KeyEvent.VK_UP) up = false;
        if(evt.getKeyCode() == KeyEvent.VK_DOWN) down = false;
        if(evt.getKeyCode() == KeyEvent.VK_LEFT) left = false;
        if(evt.getKeyCode() == KeyEvent.VK_RIGHT) right = false;
    }
});
```

# MeteorField Part 4
## Main Method

```
public static void main(String[] args) {
   final Frame frame = new MeteorField();
   java.util.Timer timer = new java.util.Timer();
   timer.schedule(new TimerTask(){
      public void run() {
         hero.move();
         Rectangle heroRect = new Rectangle(
           hero.getX(), hero.getY(), Hero.width, Hero.height);
         for(int i=0; i<numOfMeteors; i++) {
            Meteor m = meteors[i];  m.move();
            Rectangle meteorRect = new Rectangle(
             m.getX(), m.getY(), width, height);
          if(heroRect.intersects(meteorRect)) {
            hits++; hitLabel.setText("" + hits); }
        }
        if(hits>1000) {
          JOptionPane.showMessageDialog(frame , "Game Over!");
          System.exit(0);
        }
      }
   }, 0, 5); }
```

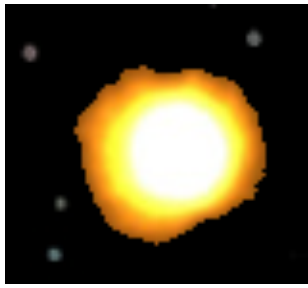


# MeteorField Part 5
## The Hero



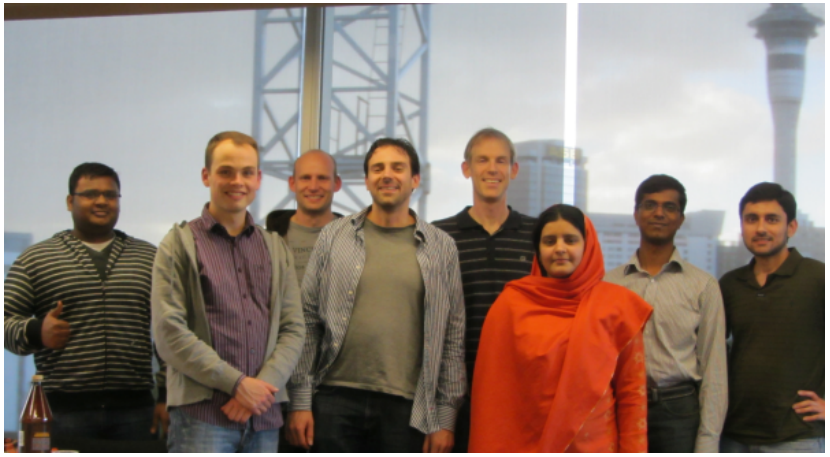
```
class Hero extends JLabel {
    static ImageIcon image = new ImageIcon("hero.png");
    public static int width = image.getIconWidth();
    public static int height = image.getIconHeight();

    public Hero() {
        super(image);
        setSize(width, height);
        setLocation(MeteorField.width/2, MeteorField.height/2);
    }

    void move() {
        if(MeteorField.up && getY()>0)
          setLocation(getX(), getY()-1);
        if(MeteorField.left && getX()>0)
          setLocation(getX()-1, getY());
        if(MeteorField.down && getY()+height<MeteorField.height)
              setLocation(getX(), getY()+1);
        if(MeteorField.right && getX()+width<MeteorField.width)
          setLocation(getX()+1, getY());
    }
}
```

# MeteorField Part 6
# The Meteors



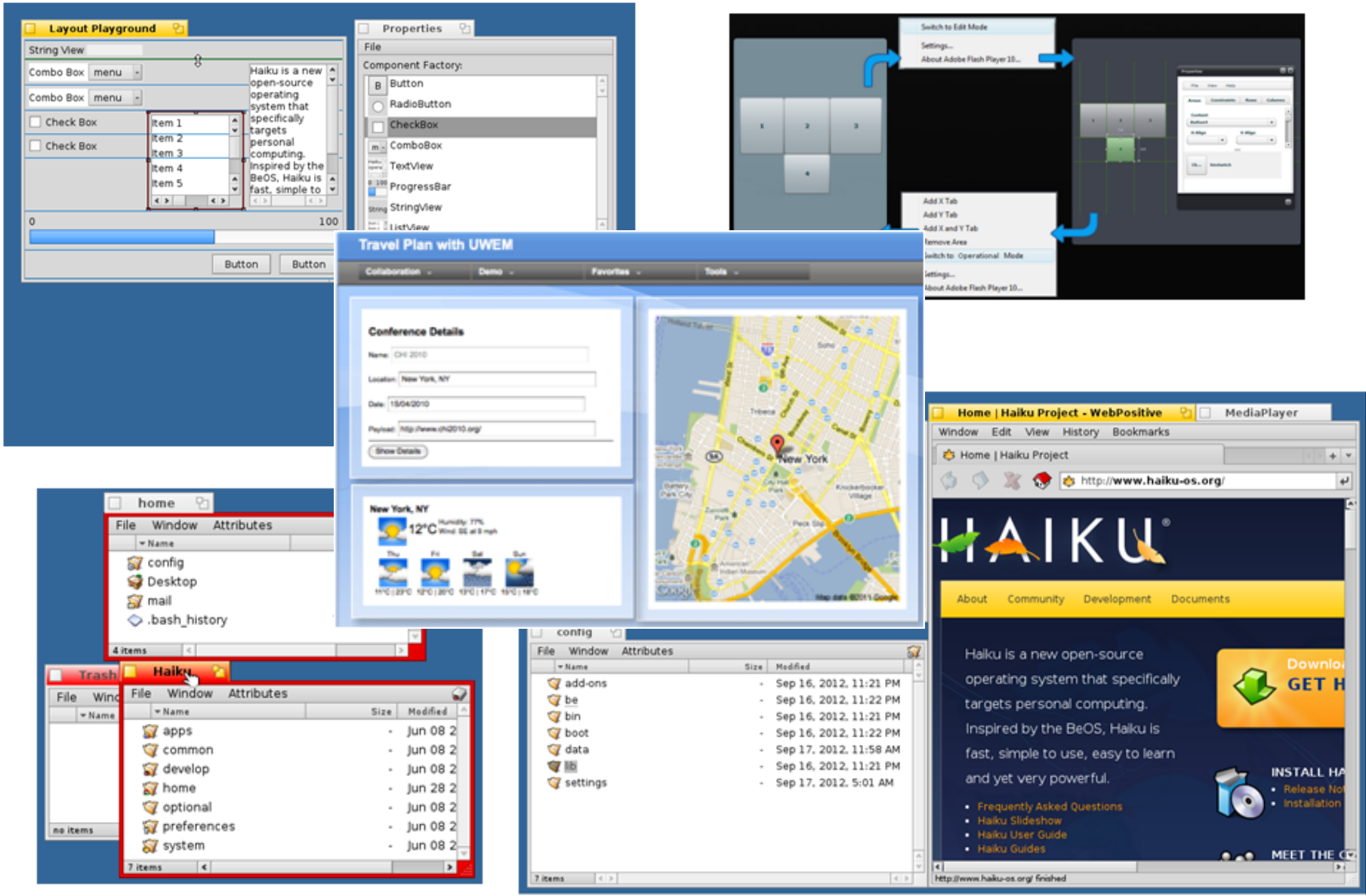
```
class Meteor extends JLabel {
    // define image, width and height analogously to Hero…
    static Random rand = new Random();
    float x, y, dx, dy;

    public Meteor() {
        super(image);   setSize(width, height);
        x = rand.nextInt(800 - Meteor.width);
        y = rand.nextInt(600 - Meteor.height);
        dx = (rand.nextInt(2)==0)?
          (0.5f+rand.nextFloat()/2):(-0.5f-rand.nextFloat()/2);
        dy = (rand.nextInt(2)==0)?
          (0.5f+rand.nextFloat()/2):(-0.5f-rand.nextFloat()/2);
    }

    void move() {
        x += dx;    y += dy;
        if(x<=0) dx = -dx;    if(y<=0) dy = -dy;
        if(x + width >= MeteorField.width) dx = -dx;
        if(y + height >= MeteorField.height) dy = -dy;
        setLocation((int)x, (int)y);
} }
```

# User Interface Research

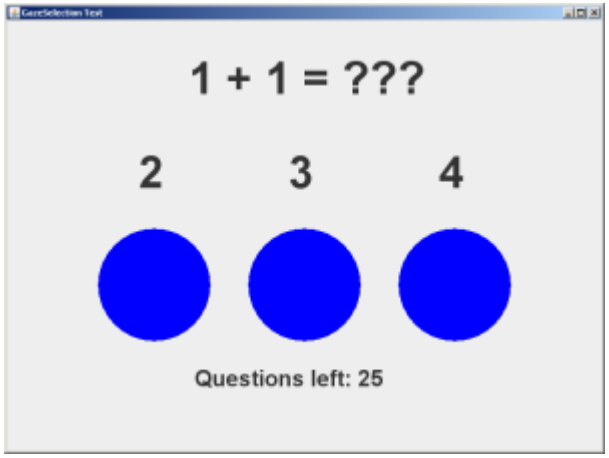


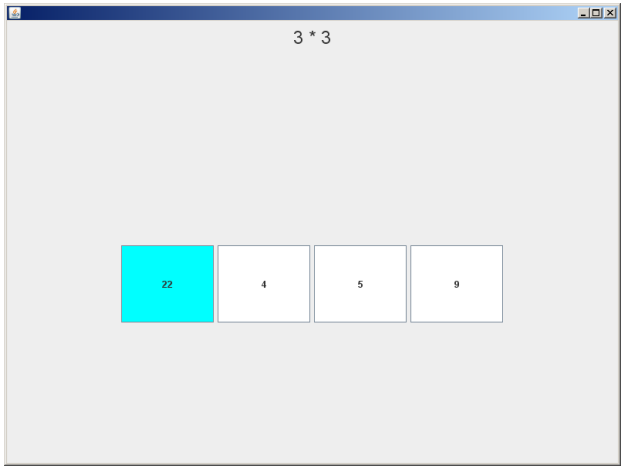
*If we knew what it was we were doing, it would not be called research, would it? (Albert Einstein)*

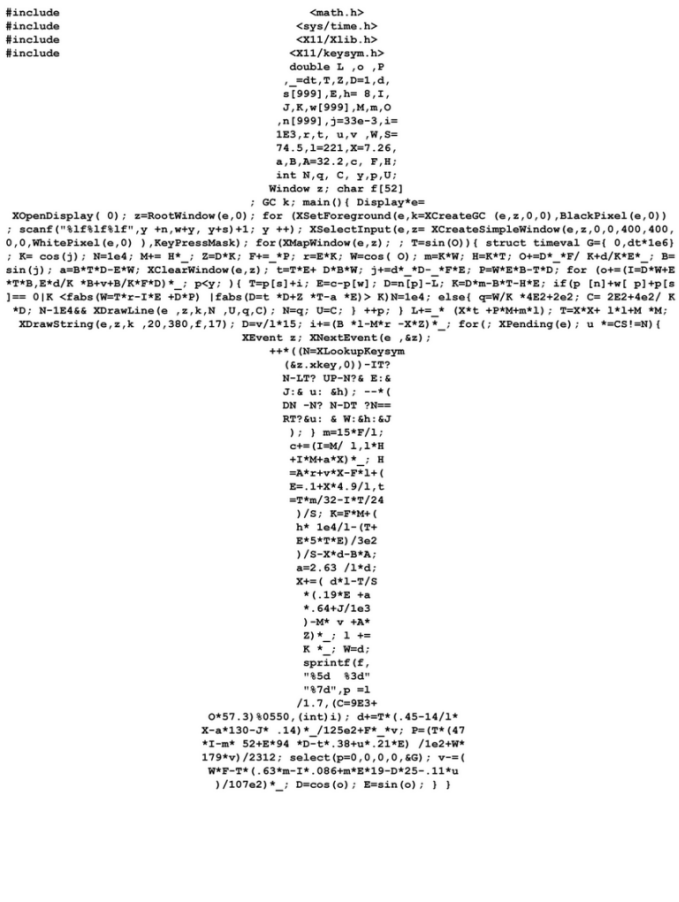
# User Interface Technology

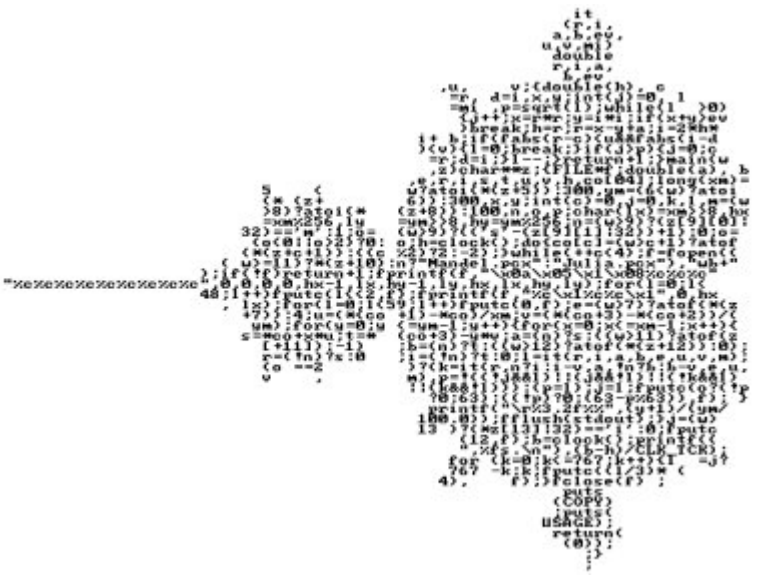


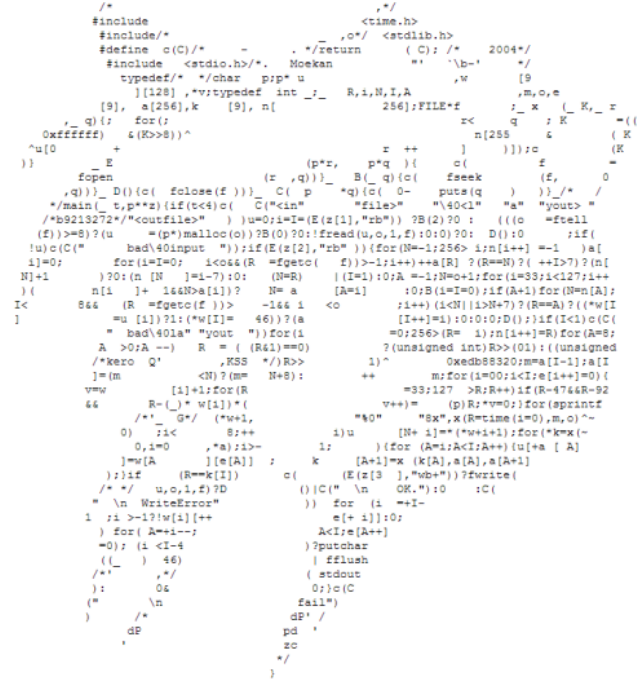
# Eye Tracking









# Thank you!