GUIs for Video Games

The Plan

- Layout suggestions
- Observer/Observable
- Steps to writing Observer/Observables
- Sample Code
- Practice

Layout Suggestions

Use a BorderLayout for the Frame
Make the Title a JLabel
Make the Controls and Score JPanels in different classes
Observer/Observable Pattern

- Observable
  - Data that changes
  - Used for the model
  - Alerts Observers of changes
- Observer
  - Depends on changing data
  - Used for the view
  - Responds to changes in the Observable

Why use Observer/Observable?

- Separates model from view
  - Model is typically where the complexity is
  - Changes in complex models is undesirable
- Avoids polling
  - An alternative is to repeated check if data has changed
  - Polling wastes processor time
- Enables flexibility of multiple views
  - Can even have multiple views open at once

More Motivation

Consider tic-tac-toe:
- Model consisting of 9 variables telling board position
- Full feature view for fast computers
  - 3D Animation
  - Sound
  - Images
- Scaled down version for slower computers
  - JButtons
  - ImageIcons
- Control class to change letters of model

Benefits

- Same model, different view
- Controller would enable networked game input.
- Model easy to send across network, view is difficult. Separation enables simplicity and flexibility.
Steps to writing Observer/Observable

1. Write separate classes for the model and the view.
2. Make the model extend Observable.
3. Make the view implement Observer.
4. Connect the Observer and Observable using the addObserver method.

Example: Score Board

- Model
  - Enemies left
  - Hero's life left
  - Timer
- View
  - JPanel with JLabels
  - GridLayout

Score.java

```java
import java.util.*;

public class Score extends Observable
{
    double shipScore, enemyScore, time;

    public void setShipScore(double s)
    {
        shipScore=s;
        setChanged();
        notifyObservers();
    }

    public double getShipScore()
    {
        return shipScore;
    }

    public void setEnemyScore(double s)
    {
        enemyScore=s;
        setChanged();
        notifyObservers();
    }

    public double getEnemyScore()
    {
        return enemyScore;
    }

    public void setTime(double t)
    {
        if((int)time!=(int)t)
        {
            time=t;
            setChanged();
            notifyObservers();
        }
        time=t;
    }

    public double getTime()
    {
        return time;
    }
}
```

Score.java

```java
public double getShipScore()
{
    return shipScore;
}

public double getEnemyScore()
{
    return enemyScore;
}

public void setTime(double t)
{
    if((int)time!=(int)t)
    {
        time=t;
        setChanged();
        notifyObservers();
    }
    time=t;
}

public double getTime()
{
    return time;
}
```
import java.awt.*;
import javax.swing.*;
import java.util.*;
public class ScorePanel extends JPanel implements Observer
{
      JLabel title;
      JLabel shipScore;
      JLabel enemyScore;
      JLabel time;
      public ScorePanel()
      {
            super();makeComponents();makeLayout();
      }
      private void makeComponents()
      {
            title=new JLabel("Score");
            shipScore=new JLabel("Ship: ");
            enemyScore=new JLabel("Enemy: ");
            time=new JLabel("Time: 0");
      }
      private void makeLayout()
      {
            setLayout(new GridLayout(4, 1));
            add(title);
            add(shipScore);
            add(enemyScore);
            add(time);
      }
      public void update(Observable obs, Object arg)
      {
            Score score=(Score)obs;
            shipScore.setText("Ship: "+score.getShipScore());
            enemyScore.setText("Enemy: "+score.getEnemyScore());
            time.setText("Time: "+(int)score.getTime());
            System.out.println("repainting");
            repaint();
      }
}
import java.awt.BorderLayout;
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
import tipgame.*;
public class Galaga extends JApplet
      implements ActionListener
{
      JFrame frame;
      JButton button;
      GalagaLoop game;
      JLabel title;
      ScorePanel scorePanel;
      public Galaga()
      {
            makeComponents();
            layoutComponents();
      }
      private void makeComponents()
      {
            frame=new JFrame();
            button=new JButton("Start");
            button.addActionListener(this);
            game=new GalagaLoop(new Dimension(500, 500));
            title=new JLabel("Blast Those Professors!");
            Font font=title.getFont();
            title.setFont(font.deriveFont(32.0f));
            scorePanel=new ScorePanel();
            game.getScore().addObserver(scorePanel);
      }
      private void layoutComponents()
      {
            Container container=frame.getContentPane();
            container.setLayout(new BorderLayout());
            container.add(game.getCanvas(), BorderLayout.CENTER);
            container.add(button, BorderLayout.SOUTH);
            container.add(title, BorderLayout.NORTH);
            container.add(scorePanel, BorderLayout.EAST);
            frame.pack();
            frame.setResizable(false);
      }
}
Practice

Write a program to count the number of clicks on a yes, no and maybe button. To do this, write three classes:

• ClickCount – keeps three integer instance variables to count the number of clicks
• ClickCountPanel – observes ClickCount for changes and updates its components when an update occurs
• ClickGUI – contains three buttons and the count panel. Clicking on the buttons registers the click via ClickCount.