

159.261

Games Programming

Mouse Input & Audio

# Input - Mouse

So far, we have looked at how to let the user control a game with the keyboard by using a *KeyListener*. Java provides a similar way for programs to receive input from the mouse.

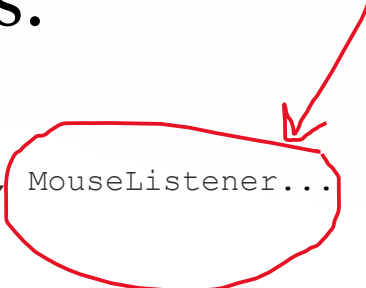
The program can use a *MouseListener* to tell the system it wants to receive mouse events and a set of functions to handle them.

# Mouse Events - Listener

Similar to the *KeyListener*, the program must implement the *MouseListener* interface functions and register that it is listening to mouse events. The GameEngine already supports this:

```
public abstract class GameEngine implements KeyListener, MouseListener...

public void setupWindow(int width, int height) {
    ...
    panel.addMouseListener(this);
    ...
}
```



# Mouse Events

*A `MouseListener` deals with five different types of mouse events:*

The mouse cursor *enters* an area.

The mouse cursor *leaves* an area.

A mouse button is *pressed*.

A mouse button is *released*.

A mouse button is *clicked*.

# Mouse Events – Event Handlers

Games based on the `GameEngine` can implement whichever of these functions they want.

```
// Called whenever a mouse button is pressed
```

```
public void mousePressed(MouseEvent e) {}
```

```
// Called whenever a mouse button is released
```

```
public void mouseReleased(MouseEvent e) {}
```

```
// Called whenever a mouse button is clicked
```

```
public void mouseClicked(MouseEvent e) {}
```

```
// Called whenever the mouse cursor enters the game panel
```

```
public void mouseEntered(MouseEvent e) {}
```

```
// Called whenever the mouse cursor exits the game panel
```

```
public void mouseExited(MouseEvent e) {}
```

# Mouse Event - Information

Each of these five functions get a *MouseEvent* that contains information about the event that has just occurred.

A *MouseEvent* can provide information about the location of the cursor and the button that was pressed:

```
e.getX();           // x.coordinate of the event  
e.getY();           // y.coordinate of the event  
e.getButton();      // Which button (NO_BUTTON, BUTTON1, BUTTON2, BUTTON3)
```

# Mouse Events – Click Events

The functions *mousePressed*, *mouseReleased* and *mouseClicked* are all used in a similar way.

Whenever the user presses, clicks or released a mouse button, some functionality will be triggered in the program. This functionality may or may not depend on the *location* of the mouse cursor when the event occurs.

# Mouse Events – Area Events

The *mouseEntered* and *mouseExited* events are triggered when the mouse enters a certain area. This is more commonly used for Java forms with multiple areas, controls etc.

It can still be useful for our games. For example, we can pause/continue the game when the mouse exits/ enters the area.



# Mouse Events - Motion Events

Java also provides another interface to allow a program to process events when the user *moves* the mouse. These events do not depend on the user pressing a mouse button.

These functions are provided by the *mouseMotionListener*. Which supports two types of mouse motion - moving the mouse and dragging the mouse.

# Mouse Motion Events

TheGameEngine already implements this with the following code:

```
public abstract class GameEngine implements KeyListener, MouseListener,  
    MouseMotionListener {  
    public void setupWindow(int width, int height) {  
        ...  
        panel.addMouseListener(this);  
        panel.addMouseMotionListener(this);  
        ...  
    }  
}
```

# Mouse Motion Events

These two types of events are captured by the following functions. The function *mouseMoved* is called whenever the mouse moves and *mouseDragged* is called whenever the mouse is moved with a button held down.

```
public void mouseMoved(MouseEvent e) {  
  
}  
  
public void mouseDragged(MouseEvent e) {  
  
}
```

# Mouse Motion Events

Like the previous mouse events, any game that uses the `GameEngine` can implement whichever of these functions it needs to use.

The *MouseListener* and the *mouseMotionListener* are kept separate because many applications don't need to listen to mouse motion events and they are generated \*a lot\* more frequently than mouse button events.

# Audio

Sound in games is an important part of making the game more involved and interesting. Background music can make the game feel less empty and more engaging. Sound effects can also provide the user with important feedback.

Playing a sound when the user hits a target is a good way to give them feedback while they might be looking elsewhere on the screen.

# Popular Gaming Sound Effects



<https://youtu.be/Nc6AluQ6kdU>

# Audio - Playing Audio

Our GameEngine provides some very simple functions for playing sound effects.

These functions are not suitable for sound effects in large scale games and are only intended for small sound effects in 2D games.

# Audio - Loading Audio File

The first function required for playing is:

```
public AudioClip loadAudio(String filename)
```

This function will open an audio file and returns it as an `AudioClip`. This audio clip can then be played later.

The functionality is similar to loading an Image which can later be drawn on the screen.



# Playing Audio

Note that there are also restrictions on the types of sound file that can be played this way. We provide some sample *.wav* files that can be used.

# Audio - Playing Audio

The two functions:

```
public void playAudio(AudioClip audioClip)
public void playAudio(AudioClip audioClip, float volume)
```

Can be used to play audio clips that have already been loaded from file. The second function has an extra parameter that controls the volume of the clip. This is specified in Decibels.

# Audio - Playing Audio Loop

An audio clip can also be set to play on continuous loop using the following functions:

```
public void startAudioLoop(AudioClip audioClip)
public void startAudioLoop(AudioClip audioClip, float volume)
public void stopAudioLoop(AudioClip)
```

Only one copy of each `AudioClip` can be played on loop at the same time.