Pygame and SDL

• Pygame is based on the SDL C library
• provides simple, platform-independent access to multimedia features
  • graphics
  • sound
  • input devices
Hello World in pygame

• see McGugan Source Code/Chapter02/helloworld.py
import and init

- Import
  
  ```python
  import pygame
  from pygame.locals import *
  ```

- Init all subsystems
  
  ```python
  pygame.init()  # init all all
  ```

- OR Init individual subsystems
  
  ```python
  pygame.sound.init()
  pygame.display.init()
  ```

  ...
create a display surface

```
screen = pygame.display.set_mode((640, 480), 0, 32)
```

```
pygame.display.set_caption("Hello, World!")
```

**Table 3-2. Flags for pygame.display.set_mode**

<table>
<thead>
<tr>
<th>Flag</th>
<th>Purpose</th>
</tr>
</thead>
<tbody>
<tr>
<td>FULLSCREEN</td>
<td>Creates a display that fills the entire screen.</td>
</tr>
<tr>
<td>DOUBLEBUF</td>
<td>Creates a “double-buffered” display. Recommended for HWSURFACE or OpenGL.</td>
</tr>
<tr>
<td>HWSURFACE</td>
<td>Creates a hardware-accelerated display (must be combined with the FULLSCREEN flag).</td>
</tr>
<tr>
<td>OPENGL</td>
<td>Creates an OpenGL renderable display.</td>
</tr>
<tr>
<td>RESIZABLE</td>
<td>Creates a resizable display.</td>
</tr>
<tr>
<td>NOFRAME</td>
<td>Removes the border and title bar from the display.</td>
</tr>
</tbody>
</table>
Surfaces

- hold images
- can be drawn on
- can be "blitted" onto other Surfaces
- `convert()` method changes surface depth so it matches that of the screen
  - makes blitting faster

```python
background = pygame.image.load(background_image_filename).convert()
mouse_cursor = pygame.image.load(mouse_image_filename)
```
Events

• pygame queues up events until your code can process them
• event types include mouse clicks, key up/down, etc
• game's main loop usually processes all queued events each time around

```python
while True:  # main loop
    for event in pygame.event.get():
        if event.type == QUIT:
            exit()
```
Blitting

- **blit == copy from one image to another**

  ```python
  # copy background image to screen
  screen.blit(background, (0,0) )
  # draw mouse_cursor image at mouse pos
  x, y = pygame.mouse.get_pos()
  x -= mouse_cursor.get_width()/2
  y -= mouse_cursor.get_height()/2
  screen.blit(mouse_cursor, (x,y))
  ```
Double Buffering

- to prevent flicker during screen updates, pygame uses "double buffering"
- pygame draws to a "back buffer" - an off-screen image
- the pygame screen always shows what's in the "front buffer"
- once the image is complete in the back buffer, call pygame.display.update() to copy the back buffer to the "front buffer"
Retrieving Events

- `pygame.event.get()`
  - gets all events from event queue
- `pygame.event.wait()`
  - wait for next event, then return it
- `pygame.event.poll()`
  - returns one event if queued, else NOEVENT
- `pygame.event.pump()`
  - call this everytime through main loop if you don't use input events (keeps internal events pumping)
- see book source code/Chapter02/events.py
<table>
<thead>
<tr>
<th>Event</th>
<th>Purpose</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>QUIT</td>
<td>User has clicked the close button.</td>
<td>none</td>
</tr>
<tr>
<td>ACTIVEEVENT</td>
<td>Pygame has been activated or hidden.</td>
<td>gain, state</td>
</tr>
<tr>
<td>KEYDOWN</td>
<td>Key has been pressed.</td>
<td>unicode, key, mod</td>
</tr>
<tr>
<td>KEYUP</td>
<td>Key has been released.</td>
<td>key, mod</td>
</tr>
<tr>
<td>MOUSEMOTION</td>
<td>Mouse has been moved.</td>
<td>pos, rel, buttons</td>
</tr>
<tr>
<td>MOUSEBUTTONDOWN</td>
<td>Mouse button was pressed.</td>
<td>pos, button</td>
</tr>
<tr>
<td>MOUSEBUTTONUP</td>
<td>Mouse button was released.</td>
<td>pos, button</td>
</tr>
<tr>
<td>JOYAXISMOTION</td>
<td>Joystick or pad was moved.</td>
<td>joy, axis, value</td>
</tr>
<tr>
<td>JOYBALLMOTION</td>
<td>Joy ball was moved.</td>
<td>joy, ball, rel</td>
</tr>
<tr>
<td>JOYHATMOTION</td>
<td>Joystick hat was moved.</td>
<td>joy, hat, value</td>
</tr>
<tr>
<td>JOYBUTTONDOWN</td>
<td>Joystick or pad button was pressed.</td>
<td>joy, button</td>
</tr>
<tr>
<td>JOYBUTTONUP</td>
<td>Joystick or pad button was released.</td>
<td>joy, button</td>
</tr>
<tr>
<td>VIDEORESIZE</td>
<td>Pygame window was resized.</td>
<td>size, w, h</td>
</tr>
<tr>
<td>VIDEOEXPOSE</td>
<td>Part or all of the Pygame window was exposed.</td>
<td>none</td>
</tr>
<tr>
<td>USEREVENT</td>
<td>A user event has occurred.</td>
<td>code</td>
</tr>
</tbody>
</table>
Mouse Motion Events

• generated whenever mouse moves

• event type
  • MOUSEMOTION

• event properties
  • buttons - a 3-tuple, for left, middle, right
    • 1 if pressed, 0 if not
  • pos - a 2-tuple with pos of mouse when event generated
  • rel - a 2-tuple with relative distance since last motion event (mickies)
Mouse Button Events

• generated when mouse button state changes

• event type
  • MOUSEBUTTONDOWN or MOUSEBUTTONUP

• event properties
  • button - 1 for left, 2 for middle, 3 for right
  • pos - tuple with mouse pos
Keyboard Events

- generated on key down and key up
- type
  - KEYDOWN or KEYUP
- properties
  - key - a number representing the key
    - use "K_" constants in your code
      - K_a through K_z
      - K_SPACE, K_ESCAPE, etc
    - http://pygame.org/docs/ref/key.html for complete list
  - mod - state of modifier keys (shift, ctrl, alt)
    - each modifier key represented by a bit in value
    - check bits with bitwise AND
      - if event.mod & KMOD_CTRL: ...
  - unicode - unicode value of key
Filtering Events

• pygame.event.set_blocked(MOUSEMOTION)
  • will prevent given type from being generated

• pygame.event.set_blocked(None)
  • unblocks all
Posting Events

• your code can generate events itself

• see text
Opening a Fullscreen Display

- fullscreentest.py in book code/Chapter03

```python
if event.type == KEYDOWN:
    if event.key == K_f:
        Fullscreen = not Fullscreen
        if Fullscreen:
            screen = pygame.display.set_mode((640, 480), FULLSCREEN, 32)
        else:
            screen = pygame.display.set_mode((640, 480), 0, 32)
```
Using the font module

- pygame uses True Type fonts (TTFs) [http://www.pygame.org/docs/ref/font.html](http://www.pygame.org/docs/ref/font.html)
- first create a font object
  
  ```python
  my_font = pygame.font.SysFont("arial",16)
  ```
  - use pygame.font.get_fonts() for a list of installed fonts
  - or create directly from a .ttf file
    ```python
    pygame.font.Font("my_font.ttf", 16)
    ```
- then use it to render a string, creating a new surface
  ```python
  text_surface = my_font.render("the penguin's exploded", True, (0,0,0), (255,255,255) )
  ```
- then blit the surface to the screen
  ```python
  screen.blit(text_surface, ( 320, 240 ) )
  ```