2 Multimedia Programming with Python and Pygame

2.1 Introduction to Python

2.2 SDL/Pygame: Multimedia/Game Framework for Python

2.3 SDL: Background of Pygame

Literature:
www.pygame.org
kidscancode.org/blog/2015/09/pygame_install/
https://www.youtube.com/c/KidsCanCodeOrg
www.libsdl.org
History of SDL & Pygame

- Sam Lantinga, 1998: Simple DirectMedia Layer (SDL) framework
  - To simplify porting games among platforms
  - Common and simple way to create displays and to process input, abstracting away from platform particularities; written in C
  - Basis for hundreds of games, among them Angry Birds, Unreal Tournament
  - Current version: 2.0.4 (January 2016)

- **Pygame** is a *language binding* for SDL Version 1.2 to Python
  - Use the SDL library from Python code

- Pygame and SDL are open source projects
  - Version 1.9.2a0 (2012?) is latest version of Pygame, using SDL 1.2.15

- Documentation:
  - www.pygame.org/docs

- More recent versions of SDL: SDL2 (www.libsdl.org)
  - Python bindings under development
## Modules in the Pygame Package

<table>
<thead>
<tr>
<th>Module</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pygame.cdrom</td>
<td>Controls CD drives</td>
</tr>
<tr>
<td>pygame.cursors</td>
<td>Loads cursor images</td>
</tr>
<tr>
<td>pygame.display</td>
<td>Accesses display</td>
</tr>
<tr>
<td>pygame.draw</td>
<td>2D vector graphics</td>
</tr>
<tr>
<td>pygame.event</td>
<td>External events</td>
</tr>
<tr>
<td>pygame.font</td>
<td>Uses System fonts</td>
</tr>
<tr>
<td>pygame.image</td>
<td>Loads and saves an image</td>
</tr>
<tr>
<td>pygame.joystick</td>
<td>Special input</td>
</tr>
<tr>
<td>pygame.key</td>
<td>Keyboard input</td>
</tr>
<tr>
<td>pygame.mixer</td>
<td>Loads and plays sounds</td>
</tr>
<tr>
<td>pygame.mouse</td>
<td>Manages mouse</td>
</tr>
<tr>
<td>pygame.movie</td>
<td>Plays movie files</td>
</tr>
<tr>
<td>pygame.music</td>
<td>Works with music and streaming audio</td>
</tr>
<tr>
<td>pygame.overlay</td>
<td>Advanced video overlays</td>
</tr>
<tr>
<td>pygame</td>
<td>High level functions</td>
</tr>
<tr>
<td>pygame.rect</td>
<td>Manages areas</td>
</tr>
<tr>
<td>pygame.sndarray</td>
<td>Manipulates sound data</td>
</tr>
<tr>
<td>pygame.sprite</td>
<td>Manages moving images</td>
</tr>
<tr>
<td>pygame.surface</td>
<td>Manages images and the screen</td>
</tr>
<tr>
<td>pygame.surfarray</td>
<td>Manipulates image pixel data</td>
</tr>
<tr>
<td>pygame.time</td>
<td>Manages timing and frame rate</td>
</tr>
<tr>
<td>pygame.transform</td>
<td>Resizes and moves images</td>
</tr>
</tbody>
</table>
Slide Show Example in Pygame

```python
import pygame
from pygame.locals import *
from sys import exit

background = pygame.Color(255, 228, 95, 0)
sc_w = 356
sc_h = 356

pygame.init()

# Create program display area
screen = pygame.display.set_mode([sc_w, sc_h])
pygame.display.set_caption("Simple Slide Show")

# Set background color
screen.fill(background)

# Load slide and show it on the screen
slide = pygame.image.load('pics/tiger.jpg').convert()
screen.blit(slide, (50, 50))
pygame.display.update()
...
```

Copy image to screen (bit block image transfer)
Display Setup

`pygame.display.set_mode(rect, flags, depth)`

- **rect**: Size of the display window (pixels)
  - Either `[w, h]` or `(w, h)`
- **flags**: Properties of the display which can be switched on/off (can also be combined to some extent)
  - **FULLSCREEN**
  - **DOUBLEBUF** Double buffering
  - **HWSURFACE** Hardware-accelerated display (must be full screen)
  - **OPENGL** OpenGL rendering
  - **RESIZABLE**
  - **NOFRAME**
- **depth**: Bit depth of display
  - Can be omitted (set automatically)
Slide Show Example in Pygame, contd.

...  

pygame.time.wait(4000)

# Load slide and show it on the screen
slide = pygame.image.load('pics/elephant.jpg').convert()
screen.blit(slide, (50, 50))
pygame.display.update()
pygame.time.wait(4000)

# Load slide and show it on the screen
slide = pygame.image.load('pics/jbeans.jpg').convert()
screen.blit(slide, (50, 50))
pygame.display.update()
pygame.time.wait(4000)
...
QUIZ

• How can we achieve that the slideshow application can be terminated by the user before running to its end?
Slide Show Example in Pygame, contd. Catching the QUIT Event

... 

pygame.time.wait(4000)

# Event loop for possible termination
for event in pygame.event.get():
    if event.type == QUIT:
        exit()

# Load slide and show it on the screen
slide = pygame.image.load('pics/elephant.jpg').convert()
screen.blit(slide, (50, 50))
pygame.display.update()
pygame.time.wait(4000)

# Event loop for possible termination
...

Event Loop

• Why ask for events?
  – Program termination (clicking close icon of window) is QUIT event
  – Other events: clicks, mouse movement, timeout, …
• Typical standard structure: (Potentially) infinite loop:
  – Asking for new events
  – Breaking the loop if termination event found
• Type 1 of "waiting for input" (explicit/passive waiting, \texttt{pygame.time.wait})
  – Frees resources during wait time
  – Not very precise in timing (dependent on external scheduling)
  – Not responsive during wait time
• Type 2 of "waiting for input" (active waiting, see next example)
  – Always responsive
  – Precise in timing
  – More resource demanding
Slide Show with Active Waiting (Part 1)

- Preliminary optimization:
  - Slide array for easier access in a loop

```python
# Preload slide files
slides = []
slides.append(pygame.image.load('pics/tiger.jpg').convert())
slides.append(pygame.image.load('pics/elephant.jpg').convert())
slides.append(pygame.image.load('pics/jbeans.jpg').convert())
slides.append(pygame.image.load('pics/peppers.jpg').convert())
slides.append(pygame.image.load('pics/butterfly.jpg').convert())
```
Slide Show with Active Waiting (Part 2)

slideindex = 0
running = True
updatePicture = True

while running:
    if updatePicture:
        screen.blit(slides[slideindex],(50,50))
        pygame.display.update()
        updatePicture = False
        timer = pygame.time.get_ticks()  # set timer

    for event in pygame.event.get():
        if event.type == QUIT:
            running = False

    if pygame.time.get_ticks() > timer+interval:
        slideindex = (slideindex+1)%len(slides)
        updatePicture = True
QUIZ

• Question 1: What is the meaning of \((\text{slideindex}+1) \mod \text{len}(	ext{slides})\)? What happens if we just increase the slideindex?

• Question 2: What happens if we omit the if-clause comparing the current time with \text{timer}+\text{interval}? (Assume we replace the if-condition by \text{True}.)

• Question 3: Can't we consider the end of the timer runtime as an event like the QUIT event?
Interactive Slide Show – Keyboard Control

slideindex = 0
running = True
updatePicture = True

while running:
    if updatePicture:
        ... as above ...

    for event in pygame.event.get():
        if event.type == QUIT:
            running = False
        if event.type == pygame.KEYDOWN:
            if event.key in [K_SPACE, K_RIGHT]:
                slideindex = (slideindex+1)%len(slides)
                updatePicture = True
            if event.key == K_LEFT:
                slideindex = (slideindex-1)%len(slides)
                updatePicture = True
            if event.key == K_q:
                running = False
2 Multimedia Programming with Python and Pygame

2.1 Introduction to Python

2.2 SDL/Pygame: Multimedia/Game Framework for Python

2.3 SDL: Background of Pygame
Moving on to Serious Programming…

```cpp
// main.cpp
// slideshow

// Created by Heinrich Hussmann on 23.03.16.
// Copyright © 2016 Heinrich Hussmann. All rights reserved.

// Using SDL, standard I/O, and strings
#include <SDL2/SDL.h>
#include <SDL2_image/SDL_image.h>
#include <stdio.h>
#include <string>

// Screen dimension constants
const int SCREEN_WIDTH = 356;
const int SCREEN_HEIGHT = 356;
const int OFFSET_XOR = 50;
const int OFFSET_YOR = 50;

// Starts up SDL and creates window
bool init();

// Loads media
bool loadMedia();

// Frees media and shuts down SDL
void close();

// Application window
SDL_Window* gWindow = NULL;

// Surface of application window
SDL_Surface* gScreenSurface = NULL;

// Directory name for pictures
std::string gPicsDir = "pics";

// Picture file names
const int NUM_PICS = 4;
std::string gPicFiles[NUM_PICS] = {"frog.jpg", "cows.jpg", "elephant.jpg", "tiger.jpg"};
```
Pure SDL2 Slide Show (Main Loop Part 1)

//Main loop
int slideIndex = 0;
bool running = true;
bool updatePicture = true;
int timer = 0;
while (running) {

    //Display next picture if necessary
    if (updatePicture) {
        SDL_BlitSurface( gLoadedPics[slideIndex], NULL,
                        gScreenSurface, &destRect );
        SDL_UpdateWindowSurface( gWindow );
        updatePicture = false;
        timer = SDL_GetTicks();
    }

    while (SDL_PollEvent(&e) != 0) {
        //User requests quit
        if(e.type == SDL_QUIT) {
            running = false;
        } else if (e.type == SDL_KEYDOWN) {
            switch (e.key.keysym.sym) {
            ...
            } else if (e.type == SDL_KEYUP) {
            ...
        } else if (e.type == SDL_MOUSEBUTTONDOWN) {
            switch (e.button.button) {
            ...
            }
        }
    }
}
Pure SDL2 Slide Show (Main Loop Part 2)

... case SDLK_LEFT:  
    slideIndex = (slideIndex+NUM_PICS-1) % NUM_PICS;  
    //Note the strange C++ definition of modulo operator  
    updatePicture = true;  
    break;  
    case SDLK_RIGHT:  
    slideIndex = (slideIndex+1) % NUM_PICS;  
    updatePicture = true;  
    break;  
}  
}  
};  

// Check time interval  
if (SDL_GetTicks() > timer+gInterval) {  
    slideIndex = (slideIndex+1) % NUM_PICS;  
    updatePicture = true;  
}  
};  
...