Multimedia-Programmierung
Übung 6

Ludwig-Maximilians-Universität München
Sommersemester 2017
Today

- Sprites, Sprite Groups and Sprite Animations

- Illustrated with Pygame + Cocos2d-x

Literature:

W. McGugan, Beginning Game Development with Python and Pygame, Apress 2007
Sprites in General
a.k.a. Spooky things that move but are not really there

- Historically:
  - something that is laid over the background
  - implemented in hardware

- Today:
  - anything that moves over the screen
  - hardware fast enough -> sprites are now software-generated

Background:  +  Sprite:  =  Screen:
Sprites in Pygame

• Module `pygame.sprite` provides basic classes and methods to handle sprites
• Class `pygame.sprite.Sprite` used as base class for game objects
• Group Objects are provided as containers/lists for sprites
• Collision detection included (see next tutorial)
• [http://www.pygame.org/docs/ref/sprite.html](http://www.pygame.org/docs/ref/sprite.html)
The Sprite Class

• Sprite objects **must** contain an image and a location
• `self.image` is a Surface that contains the image information
• `self.rect` is a Rect object that determines the location of the sprite
• A subclass of Sprite should also overwrite the `update()` method
• Contains derived methods that handle the object in groups:
  – `kill()` removes the sprite from all groups
  – `remove(*groups)` removes the sprite from a list of groups
  – `add(*groups)` adds the sprite to groups
  – `groups()` returns a list of groups the sprite belongs to
  – `alive()` tests whether the sprite belongs to any groups
import pygame
from pygame.locals import *

class Box(pygame.sprite.Sprite):
    def __init__(self, color, initial_position):
        pygame.sprite.Sprite.__init__(self)
        self.image = pygame.Surface((20, 20))
        self.image.fill(color)
        self.rect = self.image.get_rect()
        self.rect.topleft = initial_position

    def update(self):
        pass

pygame.init()
screen = pygame.display.set_mode((640, 480), 0, 32)
box = Box((255, 0, 0), (0, 0))
while True:
    for event in pygame.event.get():
        if event.type == QUIT:
            exit()
    screen.fill((0, 0, 0))
    screen.blit(box.image, box.rect)
    pygame.display.update()
Using the update Method

- Update can hold any number of arguments
- For efficient use of groups, sprites that do the same should have the same arguments

```python
class Box(pygame.sprite.Sprite):
    def __init__(self, color, initial_position):
        pygame.sprite.Sprite.__init__(self)
        self.image = pygame.Surface((20, 20))
        self.image.fill(color)
        self.rect = self.image.get_rect()
        self.rect.topleft = initial_position
        self.speed = 300

    def update(self, time_passed):
        moved_distance = time_passed * self.speed
        self.rect.left += moved_distance
```
Using the update Method II

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

# Box Class here

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

box = Box((255,0,0),(0,0))
clock = pygame.time.Clock()

while True:
    for event in pygame.event.get():
        if event.type == QUIT:
            exit()

    screen.fill((0, 0, 0))
    time_passed = clock.tick() / 1000.0
    box.update(time_passed)
    screen.blit(box.image,box.rect)
    pygame.display.update()

```

Result:

update the sprite
Using the update Method - Several Objects

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

# Box Class here
pygame.init()
screen = pygame.display.set_mode((640, 480), 0, 32)

box = Box((255,0,0),(0,0))
box2 = Box((0,255,0),(0,60))
box3 = Box((0,0,255),(0,120))
clock = pygame.time.Clock()

while True:
    for event in pygame.event.get():
        if event.type == QUIT:
            exit()
    screen.fill((0, 0, 0))
    time_passed = clock.tick() / 1000.0
    box.update(time_passed)
    box2.update(time_passed)
    box3.update(time_passed)
    screen.blit(box.image,box.rect)
    screen.blit(box2.image,box2.rect)
    screen.blit(box3.image,box3.rect)
    pygame.display.update()
```

Result:

```
... # Box Class here
pygame.init()
```
Sprite Groups

- Sprite groups (e.g. `pygame.sprite.Group`) are basically lists for sprites
- Handle the cumbersome details for the programmer:
  - `sprites()` returns a list of the sprites in that group
  - `copy()` returns a copy of the group
  - `add(*sprites)` adds a sprite to the list
  - `remove(*sprites)` removes the specified sprites from the list
  - `has(*sprites)` determines whether all sprites are in this group
  - `update(*args)` calls the update method of all sprites in this group (requires that they use the same arguments)
  - `draw(surface)` draws all the sprites in this group to the specified surface (uses `Sprite.image` and `Sprite.rect`)
  - `clear(surface, background)` erases the last drawn sprites from the list
  - `empty()` removes all sprites from the list
import pygame
from pygame.locals import *
...

# Box Class here
pygame.init()
screen = pygame.display.set_mode((640, 480), 0, 32)

boxes = ([(255,0,0),(0,0)],[(0,255,0),(0,60)],[(0,0,255),(0,120)])
sprites = pygame.sprite.Group()
for box in boxes:
    sprites.add(Box(box[0],box[1]))

clock = pygame.time.Clock()

while True:
    for event in pygame.event.get():
        if event.type == QUIT:
            exit()
    screen.fill((0, 0, 0))
time_passed = clock.tick() / 1000.0
    sprites.update(time_passed)
sprites.draw(screen)
pygame.display.update()
Advanced Groups (RenderUpdates)

- Drawing the whole screen every time a sprite moves is inefficient
- RenderUpdates helps to avoid this
- Special `draw()` method:
  - `draw(*sprites)` returns a list of Rect objects that define the areas that have been changed
  - Efficient for non-animated backgrounds
Using RenderUpdates

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

def Box(color1, color2):
    pass

def DrawRect(screen, color1, color2):
    pygame.draw.rect(screen, color1, (0, 0, 640, 480))

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

boxes = ([(255,0,0),(0,0)],[(0,255,0),(0,60)],[(0,0,255),(0,120)])
sprites = pygame.sprite.RenderUpdates()
for box in boxes:
    sprites.add(Box(box[0], box[1]))
clock = pygame.time.Clock()

background = pygame.surface.Surface((640,480))
background.fill((0,0,0))
screen.blit(background,(0,0))

while True:
    ... QUIT procedure here
    time_passed = clock.tick() / 1000.0
    sprites.update(time_passed)
    rects = sprites.draw(screen)
    pygame.display.update(rects)
    sprites.clear(screen, background)
```

This code initializes a game window, sets up boxes with different colors, and updates the screen in a loop. The `RenderUpdates` group is used to efficiently manage changes to the screen, ensuring that only the areas that have changed are updated.
Advanced Groups (OrderedUpdates)

• Remembers the order in which sprites are added
• Order is used for drawing the sprites to the screen
• Helps painting objects in the correct order
• Slower to add and remove sprites than other groups

Iterating Sprite Groups

```python
sprites = pygame.sprite.Group()
...
for sprite in sprites:
    print sprite
```
Sprites in Cocos2d-x

- Module `CCSprite.h` provides basic classes and methods to handle sprites
- Class `Sprite::create()` used as base class for game objects
- If `rect` is not specified, Cocos2d-x will automatically use the full width and height of the image file you specify.

```cpp
auto mySprite = Sprite::create("mysprite.png");

auto mySprite = Sprite::create("mysprite.png", Rect(0, 0, 200, 200));
```
Sprite Groups

- Grouping sprites may be required for tracking different object classes
- Add the two member variables to the header file
- These will store different sprite groups

```cpp
    cocos2d::CCArray *_group1;
    cocos2d::CCArray *_group2;
```
Sprite Groups

- New sprites are added to the array
- Its tag is set to an individual identifier (e.g. 1)

```cpp
  group1->setTag(1);
  _group1->addObject(group1);

  group2->setTag(2);
  _group2->addObject(group2);
```
Sprite Groups

• Now, you can identify the sprite groups and do something
• For example, delete objects after collision:

```cpp
if (sprite->getTag() == 1) // object1
{
    _object1->removeObject(sprite);
}
```
Keyframe Animations (Recap)

- Keyframes are defined
- Intermediate steps are interpolated
- Basic interpolators/tweens/... built into many programming environments (e.g. CreateJS, JavaFX, Cocos2d-x)
- Examples: motion, color, shape
- **BUT**: PyGame does not provide built-in interpolators!
Sprite Animations

• Animations consist of frames that are displayed one after another

Frame 1 ... Frame 7

• Either imported as single graphics or with sprite sheets

• Single graphics are not recommended as loading each file takes time, switching between the files is expensive and sprites can't be optimized

Contains small tutorial on creating animations with sprite sheets in Pygame
Sprite Sheets & Spriting

• Sprite sheets contain all possible movements for a character
• If no meta data is provided, each Sprite should have the same size for easy slicing in software

http://www.themysticalforestzone.com/Sprite_section.htm
Creating Sprite Sheets

• Editing with Photoshop, Gimp, Pixen etc.
• Pay attention to positioning of character and background color (should not appear in character)
• Tool support sprite sheet generation (see next slide)
Tools for Creating Sprite Sheets

For example:

- [https://zwopple.com/zwoptex/](https://zwopple.com/zwoptex/)
- [https://www.codeandweb.com/texturepacker](https://www.codeandweb.com/texturepacker)
- [http://renderhjs.net/shoebox/](http://renderhjs.net/shoebox/)

- Optimize image size
- Remove transparent pixels
- Target multiple scales and formats
- Generate plist files
Slicing Sprite Sheets

```python
def load_sliced_sprites(self, w, h, filename):
    images = []
    master_image = pygame.image.load(os.path.join('ressources', filename)).convert_alpha()
    master_image.set_colorkey((255,0,255))
    master_width, master_height = master_image.get_size()

    for i in xrange(int(master_width/w)):
        images.append(master_image.subsurface((i*w,0,w,h)))

    return images
```

set transparent color, background color of sprite sheet
create subsurfaces

More specialized slicing function may be needed due to incompatible sprite sheet (e.g. with borders)
First Sprite Animation 1

```python
import os, pygame
from pygame.locals import *

def load_sliced_sprites(self, w, h, filename):
    ...

class BombWithAnimation(pygame.sprite.Sprite):
    def __init__(self, color, initial_position, fps):
        pygame.sprite.Sprite.__init__(self)
        self.act_frame = 0
        # create the images for the animation
        self.frames = load_sliced_sprites(20, 20, "explosed-sprite.png")
        self.image = self.frames[0]
        self.rect = self.image.get_rect()
        self.rect.topleft = initial_position
        self.fps = fps
        self.change_time = 1.0 / self.fps
        self.time = 0

    def update(self, time_passed):
        self.time += time_passed
        if self.time >= self.change_time:
            self.act_frame = (self.act_frame + 1) % len(self.frames)
            self.image = self.frames[self.act_frame]
            self.time = 0
```

Remember current frame

Based on the frames per second (fps) calculate the time needed for animation changes

Frame changed?
Change frame
pygame.init()

screen = pygame.display.set_mode((640, 480), 0, 32)
bomb1 = BombWithAnimation((0,0),4)
clock = pygame.time.Clock()

while True:
    for event in pygame.event.get():
        if event.type == QUIT:
            exit()
    screen.fill((100, 200, 0))
    time_passed = clock.tick() / 1000.0
    bomb1.update(time_passed)
    screen.blit(bomb1.image,bomb1.rect)
    pygame.display.update()
Multiple Parallel Animations

```python
... 
pygame.init()

screen = pygame.display.set_mode((640, 480), 0, 32)
bomb1 = BombWithAnimation((0,0),4)
bomb2 = BombWithAnimation((40,40),2)
clock = pygame.time.Clock()

while True:
    for event in pygame.event.get():
        if event.type == QUIT:
            exit()
    screen.fill((100, 200, 0))
    time_passed = clock.tick() / 1000.0
    bomb1.update(time_passed)
    screen.blit(bomb1.image,bomb1.rect)
    bomb2.update(time_passed)
    screen.blit(bomb2.image,bomb2.rect)
    pygame.display.update()
```

two bombs in two different framerates
Sprite Sheets in Cocos2d-x

• Generate sprite sheet (and plist XML)
• Load sprite sheet in SpriteFrameCache for quick access
• The must be SpriteFrame is loaded only once

// load the Sprite Sheet
auto spritecache = SpriteFrameCache::getInstance();

// the .plist file can be generated with Cocos Studio or ShoeBox
spritecache->addSpriteFramesWithFile("sprites.plist");

// load the Sprite Sheet
auto mysprite = Sprite::createWithSpriteFrameName("mysprite.png");
Polygogne Sprites

- GPU has to process transparent and visible pixels
- Polygogne sprites are made of a series of triangles
- This improves performance by saving pixels

http://cocos2d-x.org/docs/programmers-guide/3/

10285 Pixels 4089 Pixels
Polygone Sprites

- AutoPolygon processes an image into a 2d polygon mesh at runtime.

```cpp
// Generate polygon info automatically.
auto pinfo = AutoPolygon::generatePolygon("filename.png");

// Create a sprite with polygon info.
auto sprite = Sprite::create(pinfo);
```
Frame by Frame Animation
with Sprite Sheets I

Vector HelloWorld::getAnimation(const char *format, int count)
{
    auto spritecache = SpriteFrameCache::getInstance();
    Vector animFrames;
    char str[100];
    for(int i = 1; i <= count; i++)
    {
        sprintf(str, format, i);
        animFrames.pushBack(spritecache->getSpriteFrameByName(str));
    }
    return animFrames;
}

Frame by Frame Animation with Sprite Sheets II

```cpp
auto frames = getAnimation("capguy/walk/%04d.png", 8);
auto sprite = Sprite::createWithSpriteFrame(frames.front());
background->addChild(sprite);
sprite->setPosition(100,620);

auto animation = Animation::createWithSpriteFrames(frames, 1.0f/8);
sprite->runAction(RepeatForever::create(Animate::create(animation)));
```

Useful Links

- http://www.pygame.org/docs
- http://www.cocos2d-x.org/wiki/Sprites