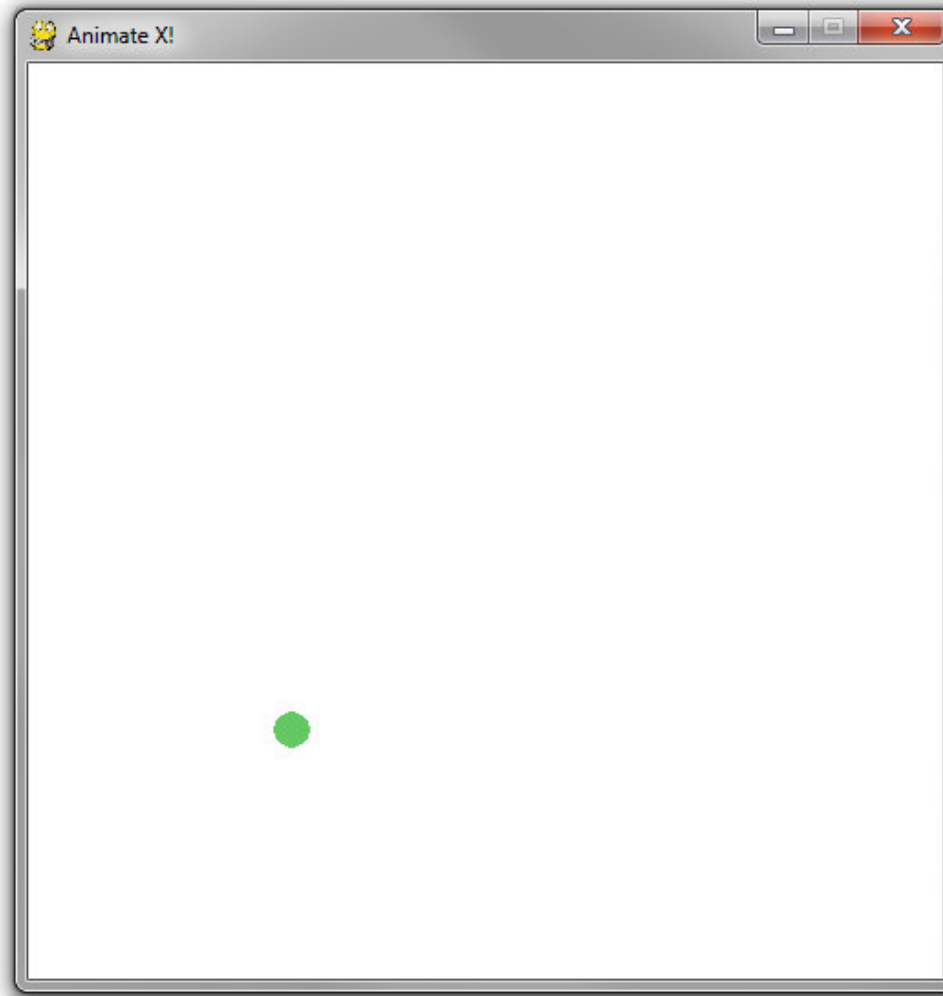


# Multimedia-Programmierung

## Übung 7

Ludwig-Maximilians-Universität München  
Sommersemester 2014

# Lösung ÜB 6



# Today

- Sprites and  Pygame

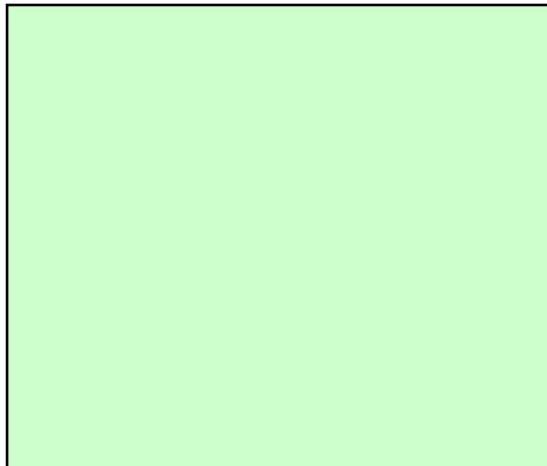
Literature: W. McGugan, Beginning Game Development with Python and Pygame, Apress 2007

# Sprites

a.k.a. Spooky things that move but are not really there

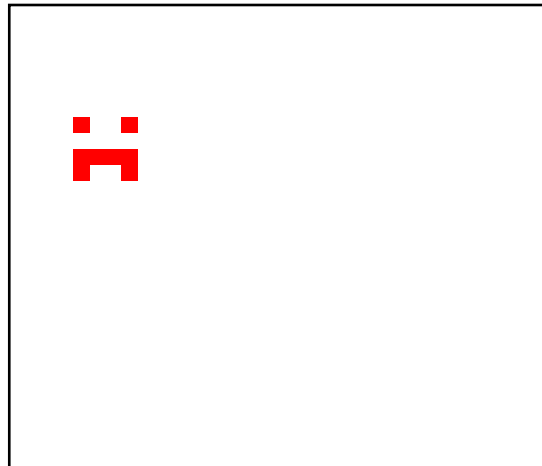
- Historically:
  - something that is laid over the background
  - implemented in hardware
- Today:
  - anything that moves over the screen
  - hardware fast enough -> sprites are now software-generated

Background:



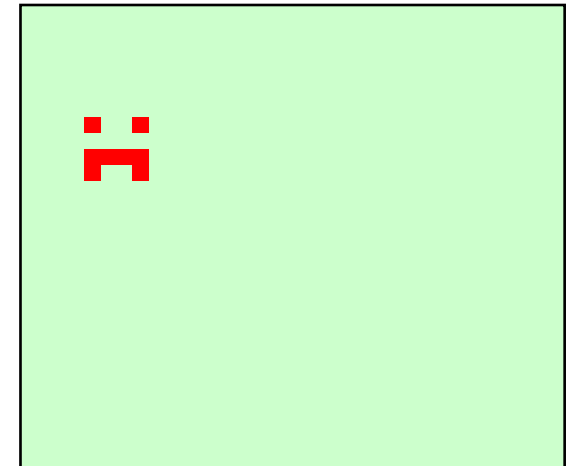
+

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
- <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

# Our First Sprite



```
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
```

← call the superclass constructor

← define the image Surface

← define the rect

```
    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)
```

← blit the sprite

```
    pygame.display.update()
```

Result:



# 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

```
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



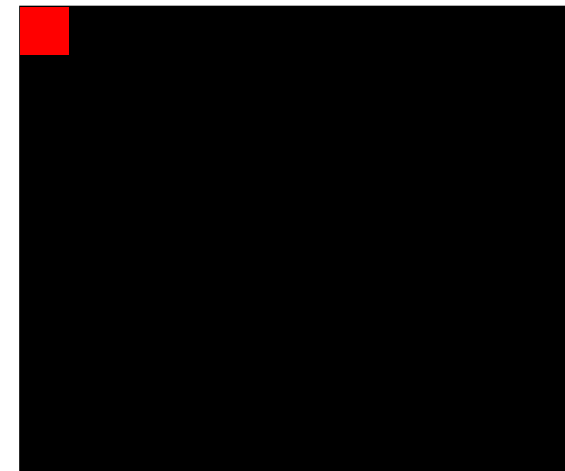
```
import pygame
from pygame.locals import *
... # Box Class here

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) ← update the sprite
    screen.blit(box.image, box.rect)
    pygame.display.update()
```

Result:



# Using the update Method - Several Objects



```
import pygame
from pygame.locals import *
... # Box Class here
pygame.init()
screen = pygame.display.set_mode((640, 480), 0, 32)

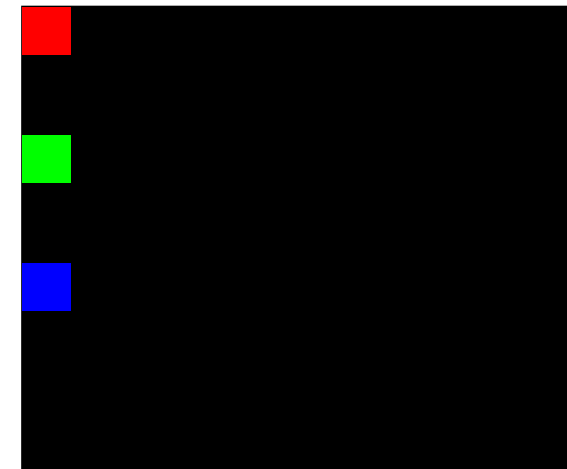
box = Box((255,0,0),(0,0))
box2 = Box((0,255,0),(0,60))
box3 = Box((0,0,255),(0,120))
clock = pygame.time.Clock()

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()
```

← too cumbersome

←

Result:



# 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

# Handling Complexity using Groups



```
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()
```

← create a 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)
```

← call update on the group

```
    sprites.draw(screen)
```

← draw all sprites in the group

```
    pygame.display.update()
```

← onto the screen surface

# 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



```
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.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))
```

← RenderUpdates group

```
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)
```

← call draw and store the  
changed areas

← clear the changes done

# 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

```
sprites = pygame.sprite.Group()  
...  
for sprite in sprites:  
    print sprite
```



# Collision Detection

- `Rect.collidepoint(point)` can be used to see whether a coordinate is within the area of a Rect object
- `pygame.sprite` has advanced methods to check for collisions
  - E.g. `pygame.sprite.collide_rect(a,b)` checks whether two sprites intersect

# A simple collision detection



```
import pygame
from pygame.locals import *

...

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()
        if event.type == MOUSEBUTTONDOWN:
            if box.rect.collidepoint(event.pos):
                print "in"
            else:
                print "out"

    box.update()
    screen.blit(box.image, box.rect)
    pygame.display.update()
```

# Useful Links

- Pygame Documentation !!!!  
<http://www.pygame.org/docs>