Sprites and collisions

- you will need to save these images in the same directory as the code:
  - gun.png ⟨ball.png⟩
  - arrow.png ⟨ball.png⟩
  - ball.png ⟨ball.png⟩

Sprites and collisions

- sprites are created and normally placed into a list
  - and referred to as a group

- you can then test for a collision between another sprite via:

```
inter = spritecollide(foo, bar, dokill)
```

- inter is a list of all sprites from list bar which have collided with the single sprite foo

- the dokill parameter is either True or False and if it was True the the kill method is called for every sprite in the list inter

Sprites and collisions

```
for bomb in sprite.spritecollide(player, bombs, True):
    boom_sound.play()
```

- notice that this example tests whether a single sprite player has collided with any sprite in the bombs list

Managing collisions between two groups of sprites

- we can detect whether a collision occurs between two groups of sprites by using the following function:

```
groupcollide(list1, list2, dokill1, dokill2)
```

- this function returns a dictionary
  - each key in the dictionary is a sprite in list1 and its value is a list of sprites from list2 with which it has collided
  - the dokill1, dokill2 arguments determine whether the kill method should be called in list1 or list2

Managing collisions between two groups of sprites

- ```
  for alien in sprite.groupcollide(aliens, bullets, True, T
      boom_sound.play()
      kills += 1
  ```

- the code checks for the collisions between bullets and all the aliens

- in this case we only loop over the dictionary keys
  - but we could loop over the `values()` or `items()`
  - if we wanted to do something to the specific shots that collided with aliens

Managing collisions between two groups of sprites

- if we did loop over the `values()` we would be looping through lists that contain sprites

- note that the same sprite may even appear more than once in these different loops, since the same `bullet` could have collided against multiple `aliens`

Real example

- *"Talk is cheap. Show me the code."*
  Linux Torvalds, Fri, 25 Aug 2000 11:09:12 -0700 (PDT)

Space invaders in Python

- ```
  #!/usr/bin/python

  import pygame
  import sys
  from pygame.locals import KEYDOWN, KEYUP, K_SPACE, K_ESCA
                           K_RIGHT, K_LEFT


  width        = 320
  height       = 240
  imageWidth   = 32
  imageHeight  = 32

  goingLeft    = True
  invaderHeight = 0
  gunLeft      = False
  gunRight     = False
  gunXpos      = (width/2)-(imageWidth/2)
  delay        = 10
  ```

Space invaders in Python

```
class BoxSprite(pygame.sprite.Sprite):
    image = None

    def __init__(self, initial_position):
        pygame.sprite.Sprite.__init__(self)
        if BoxSprite.image is None:
            BoxSprite.image = pygame.image.load("ball.png
        self.image = BoxSprite.image

        self.rect = self.image.get_rect()
        self.rect.topleft = initial_position
        self.next_update_time = 0   # as soon as possible
        self.yPos = initial_position[1]
```

Space invaders in Python

```
def update(self, current_time, left, right):
    global goingLeft, invaderHeight, imageWidth, delay
    # check update
    if self.next_update_time < current_time:
        # If we're at the left or right the screen, switch
        if self.rect.topleft[0] == left:
            goingLeft = False
            invaderHeight += 1
        elif self.rect.topleft[0] == right-imageWidth:
            goingLeft = True
            invaderHeight += 1
        if goingLeft == True:
            self.rect.topleft = [self.rect.topleft[0]-1,
                                 self.rect.topleft[1]]
        else:
            self.rect.topleft = [self.rect.topleft[0]+1,
                                 self.rect.topleft[1]]
        self.rect.topleft = [self.rect.topleft[0],
                             invaderHeight+self.yPos]
        self.next_update_time = current_time + delay
```

Space invaders in Python

```
class missile(pygame.sprite.Sprite):
    image = None

    def __init__(self, initial_position):
        pygame.sprite.Sprite.__init__(self)
        if missile.image is None:
            missile.image = pygame.image.load("arrow.png")
        self.image = missile.image

        self.rect = self.image.get_rect()
        self.rect.topleft = initial_position
        self.next_update_time = 0 # update() hasn't been calle
```

Space invaders in Python

```
def update(self, current_time):
    global missile
    # check update
    if self.next_update_time < current_time:
        # If we're reached the top then stop
        if self.rect.topleft[1] == 0:
            missiles.remove(self)
            self.kill()
            return
        else:
            self.rect.topleft = [self.rect.topleft[0]
                                 self.rect.topleft[1]
        self.next_update_time = current_time + 4
```

Space invaders in Python

```
class gun(pygame.sprite.Sprite):
    image   = None

    def __init__(self):
        global width, imageHeight, gunXpos
        pygame.sprite.Sprite.__init__(self)
        if gun.image is None:
            gun.image = pygame.image.load("gun.png")
        self.image = gun.image

        self.rect = self.image.get_rect()
        self.rect.topleft = [gunXpos, height-imageHeight]
        self.next_update_time = 0 # update() hasn't been
```

Space invaders in Python

```
    def update(self, current_time):
        global gunXpos, width, imageWidth

        # check update
        if self.next_update_time < current_time:
            if gunLeft and gunXpos>0:
                gunXpos -= 1
            if gunRight and gunXpos<width-imageWidth:
                gunXpos += 1
            self.rect.topleft = [gunXpos, self.rect.tople
            self.next_update_time = current_time + 1
```

Space invaders in Python

```
def checkInput():
    global gunLeft, gunRight, missiles, gunXpos, height
    for event in pygame.event.get():
        if event.type == KEYDOWN:
            if event.key == K_ESCAPE:
                sys.exit(0)
            elif event.key == K_RIGHT:
                gunLeft = False
                gunRight = True
            elif event.key == K_LEFT:
                gunLeft = True
                gunRight = False
            else:
                missiles.append(missile([gunXpos, height]))
        elif event.type == KEYUP and event.key != K_SPACE:
            gunRight = False
            gunLeft = False
```

Space invaders in Python

```
def checkCollisions():
    global missiles, boxes
    if missiles != [] and boxes != []:
        for m in missiles:
            found = False
            for b in pygame.sprite.spritecollide(m, boxes
                boxes.remove(b)
                b.kill()
                found = True
            if found:
                missiles.remove(m)
                m.kill()
```

Main section of space invaders - initialisation

```
pygame.init()
boxes = []
missiles = []

for x in range(0, width, 32):
    for y in range(0, 96, 32):
        boxes.append(BoxSprite([x, y]))

screen = pygame.display.set_mode([320, 240])
gunControl = gun()
```

Main section of space invaders - initialisation

```
while boxes != []:
    screen.fill([0, 0, 0]) # blank the screen.
    time = pygame.time.get_ticks()
    for b in boxes:
        b.update(time, 0, width)
        screen.blit(b.image, b.rect)

    checkInput()
    checkCollisions()
```

Main section of space invaders - initialisation

```
    gunControl.update(time)
    screen.blit(gunControl.image, gunControl.rect)
    for m in missiles:
        m.update(time)
        screen.blit(m.image, m.rect)
    pygame.display.update()
    if pygame.sprite.spritecollide(gunControl, boxes, Fal
        pygame.time.delay(50)
        print "looser"
        sys.exit(0)
    if len(boxes)<10:
        delay = len(boxes)

pygame.time.delay(50)
print "winner"
```

Tutorial

■ extend your missile command program to include a
city class

■ give your city class an __init__, update,
ignite, erase and check method

■ the method prototypes are:

Tutorial

```
    # create a city at pos
    # calculcate the epicenter of the city
    # store it in the class
    def __init__ (self, pos):
    # draw the city
    def draw_city (self):
    # remove the city
    def erase (self):
    # determine whether city should catch fire given expo
    def check (self, p, radius):
```

■ extend your game to include cities and their
destuction!

■ now create a gun class (which will be very similar to
the city class)