



# Pygame

*Games in Python – the easy way*

# The pyGame Package

- A set of Python modules to help write games
- Deals with media nicely (pictures, sound)
- Interacts with user nicely (keyboard, joystick, mouse input)
- Lots of more advanced features for working with graphics, etc

# Where to Start?

- The official [pyGame website](#)
- Search for [tutorials](#)
- The Application Programming Interface ([API](#))
  - specifies the classes and functions in package
- Experiment!

# A Skeleton

- Tutorials basically all have the same setup -- let's use it!

## template.py

```
1  from pygame import *
2  from pygame.sprite import *
3  from random import *
4
5  init()
6
7  screen = display.set_mode((640, 480))
8  display.set_caption('Window name!')
9
10 while True:
11     e = event.poll()
12     if e.type == QUIT:
13         quit()
14         break
15
16     screen.fill(Color("white"))
17     display.update()
```

# Surface

- Most of the game elements you see are represented as Surfaces
- `display.set_mode((x, y))` creates your canvas – it returns a Surface object

Useful surface methods:

- `fill("color")` fills the surface object it's called on
- `blit(surface, area)` paints **surface** onto the object blit is called on in the rectangle bounded by the **area** tuple
  - Example: `screen.blit(ball, (50,50))`

# Rect

- Objects that store rectangular coordinates
- Call `.get_rect()` on a surface to get its bounding box

Rectangle methods/variables:

- `.center` holds the object's center as a tuple
- `.colliderect(target)` returns True if the parameter overlaps with the object
- `.collidepoint(target)` returns True if the target point overlaps with the object



# Media

- Loading an image:
  - `img = image.load("file.gif").convert()`
- Getting a bounding rectangle:
  - `img_rect = img.get_rect()`
- Loading and playing a sound file:
  - `mixer.Sound("file.wav").play()`

# Sprite

- Class visible game objects inherit from

## Ball.py

```
1 from pygame import *
2 from pygame.sprite import *
3
4 class Ball(Sprite):
5     def __init__(self):
6         Sprite.__init__(self)
7         self.image = image.load("ball.png").convert()
8         self.rect = self.image.get_rect()
9
10    def update(self):
11        self.rect.center = mouse.get_pos()
```

# Using Sprites

- They're just objects: initialize them
  - `ball = Ball()`
- Create a group of sprites in main
  - `sprites = RenderPlain(sprite1, sprite2)`
- Groups know how to draw and update
  - `sprites.update()`
  - `sprites.draw(surface)`

# Events

- User input such as clicking, moving mouse or key presses
- Add more branches to test the result of `event.poll()`
- Events to test for:
  - QUIT
  - MOUSEBUTTONDOWN
  - JOYBUTTONDOWN
- Testing for the letter 'd' being pressed using `KEYDOWN`

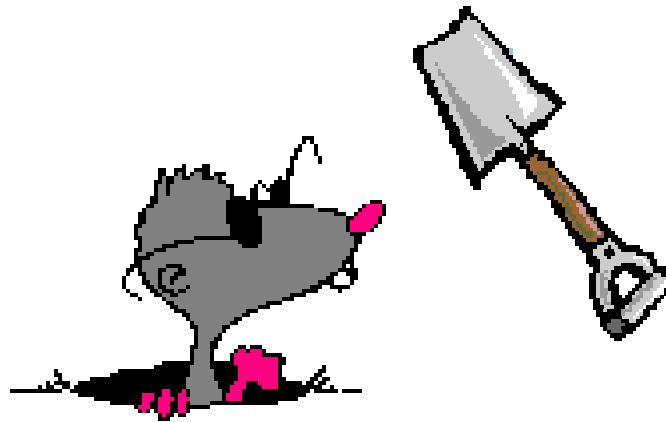
```
if e.type == KEYDOWN:  
    if e.key == K_d:
```

# Adding Text

- `f = font.Font(font, size)` goes before your game loop
  - Example: `f = font.Font(None, 25)`
  - Usually, **None** is a good enough font!
- `text = Font.render(text, antialias, color)`
  - Example: `text = f.render("Hello!", True, Color("green"))`
  - Returns a surface
- Must be blit, just like any other surface
  - Example: `screen.blit(t, (320, 0))`

# Exercise: Whack-a-mole

- Clicking on the mole
  - plays a sound
  - makes the mole move
- The number of hits is displayed at the top of the screen



# Or make your own

- You can make your own game but DESIGN IT CAREFULLY!
  - You must have something to show by the last day of class
  - It doesn't need to be complete, but should be playable
- Requirements:
  - At least two sprites
  - Interaction between sprites
  - User input from keyboard or mouse
  - Some kind of score displayed
- Have fun!!

