



# Video Games

## Writing Games with Pygame

Special thanks to Scott Shawcroft, Ryan Tucker, and Paul Beck for their work on these slides.

Except where otherwise noted, this work is licensed under:

<http://creativecommons.org/licenses/by-nc-sa/3.0>

# Video Games

Writing Games with Pygame

# Inheritance

```
class name (superclass) :  
    statements
```

– Example:

```
class Point3D(Point) :           # Point3D extends Point  
    z = 0                         # add a z field  
    ...
```

- Python also supports *multiple inheritance*

```
class name (superclass, ..., superclass) :  
    statements
```

# Calling Superclass Methods

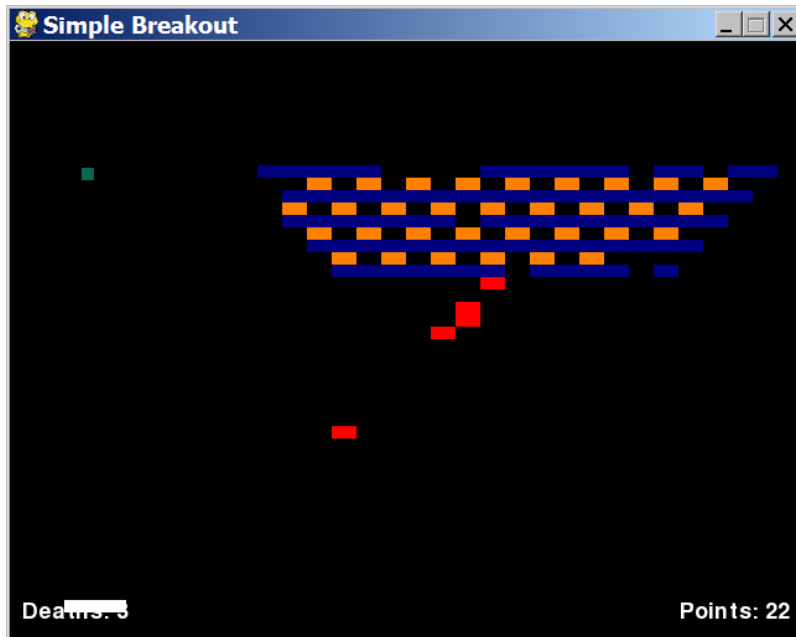
- methods: **class.method(parameters)**
- constructors: **class.\_\_init\_\_(parameters)**

```
class Point3D(Point):  
    z = 0  
  
    def __init__(self, x, y, z):  
        Point.__init__(self, x, y)  
        self.z = z  
  
    def translate(self, dx, dy, dz):  
        Point.translate(self, dx, dy)  
        self.z += dz
```

# Pygame



- A set of Python modules to help write games
- Deals with media (pictures, sound) nicely
- Interacts with user nicely (keyboard, joystick, mouse input)





# Installing Pygame

- Go to the Pygame web site: <http://www.pygame.org/>
  - click 'Downloads' at left
  - Windows users: under the 'Windows' section,
    - click the most recent version  
(as of this quarter, that is [pygame-1.9.1.win32-py3.1.msi](#))
  - Mac users: under the 'Macintosh' section,
    - click the most recent version  
(as of this quarter, [pygame-1.9.1release-py3.1-macosx10.5.zip](#))
  - save file to hard disk
  - run file to install it

# Other Resources

- Pygame documentation: <http://www.pygame.org/docs/>
  - lists every class in Pygame and its useful behavior
- The Application Programming Interface ([API](#))
  - specifies the classes and functions in package
- Search for [tutorials](#)
- Experiment! (Not kidding)

# What is a game?

- Game State.
  - In Java, what's the "state" of an object?
- What's the state of chess?



- What type of 'objects' would we want if we were to implement chess?



# Our Goal: Starcraft II in PyGame



# Starcraft II

- Ok... not all of it. But we'll move a probe around.



# Initializing a Game

- Import Pygame's relevant classes:

```
import sys
from pygame import *
from pygame.locals import *
from pygame.sprite import *
```

- Initialize Pygame at the start of your code:

```
pygame.init()
```

- Note: Because you've imported pygame as above, you are not technically required to have `pygame.init()`

# Creating a Window

```
name = display.set_mode( (width, height) [, options] )
```

Example:

```
screen = display.set_mode( (640, 480) )
```

- Options:

FULLSCREEN

- use whole screen instead of a window

DOUBLEBUF

- display buffering for smoother animation

OPENGL

- 3D acceleration (don't use unless needed)

Example:

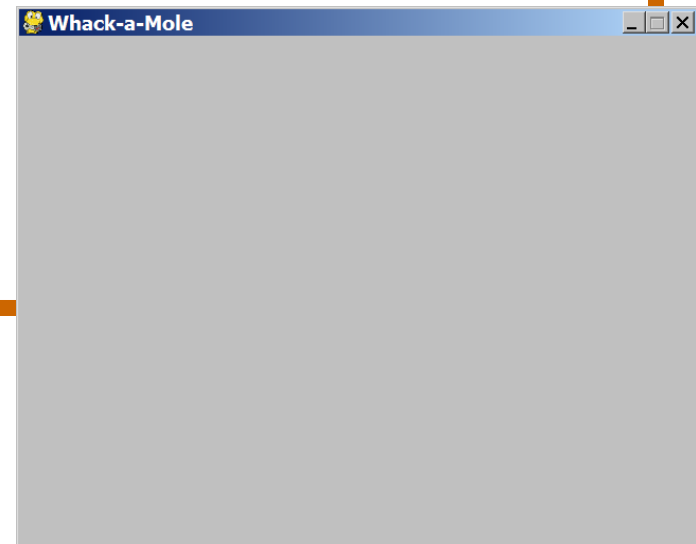
```
screen = display.set_mode( (1024, 768) , FULLSCREEN )
```

# Initial Game Program

- An initial, incomplete game file using Pygame:

## whack\_a\_mole.py

```
1 import pygame
2 from pygame import *
3 from pygame.locals import *
4 from pygame.sprite import *
5
6 pygame.init()
7
8 # set window title
9 display.set_caption("Whack-a-Mole")
10
11 screen = display.set_mode((640, 480))
12
```

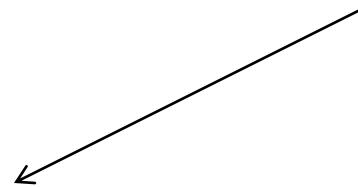




# Sprites

Next we must define all the *sprites* found in the game.

- **sprite**: A character, enemy, or other object in a game.
  - Sprites can move, animate, collide, and be acted upon
  - Sprites usually consist of an *image* to draw on the screen and a *bounding rectangle* indicating the sprite's collision area
- Pygame sprites are objects that extend the `Sprite` class.





# Programming a Sprite

```
class name(Sprite):  
    # constructor  
    def __init__(self):  
        Sprite.__init__(self)  
        self.image = image.load("filename")  
        self.rect = self.image.get_rect()
```

## other methods (if any)

– Pre-defined fields in every sprite:

`self.image` - the image or shape to draw for this sprite

- images are `Surface` objects, loaded by `image.load` function

`self.rect` - position and size of where to draw the image

# Sprite Example

**# A class for a probe to mine minerals!**

```
class Probe(Sprite):  
    def __init__(self):  
        Sprite.__init__(self)  
        self.image = image.load("probe.png")  
        self.rect = self.image.get_rect()
```

Note: The image and the rect are part of the state.

What are some other things that could be part of the state?

# Sprite Groups

```
name = Group(sprite1, sprite2, ...)
```

– To draw sprites on screen, they must be put into a `Group`

Example:

```
protoss = Probe()      # create a Mole object  
sprites = Group(protoss)
```

`Group` methods:

- `draw(surface)`      - draws all sprites in group onto a surface
- `update()`            - updates every sprite's appearance

# Surface

- In Pygame, every 2D object is an object of type `Surface`
  - The screen object returned from `display.set_mode()`, each game character, images, etc.
  - Useful methods in each `Surface` object:

Method Name	Description
<code>fill((red, green, blue))</code>	paints surface in given color ( <i>rgb 0-255</i> )
<code>get_width()</code> , <code>get_height()</code>	returns the dimensions of the surface
<code>get_rect()</code>	returns a <code>Rect</code> object representing the x/y/w/h bounding this surface
<code>blit(src, dest)</code>	draws this surface onto another surface

# Drawing and Updating

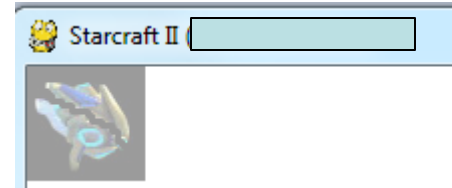
- All Surface and Group objects have an `update` method that redraws that object when it moves or changes.
- Once sprites are drawn onto the screen, you must call `display.update()` to see the changes

```
protoss = Probe()      # create a Probe object
sprites = Group(protoss)
sprites.update()     # Calls the update method
sprites.draw(screen)
display.update()    # redraw to see the sprites
```

# Game Program

## StarCraft\_II.py

```
1 import sys
2 from pygame import *
3 from pygame.locals import *
4 from pygame.sprite import *
5
6 class Probe(Sprite):
7     def __init__(self):
8         Sprite.__init__(self)
9         self.image = image.load("probe.png")
10        self.rect = self.image.get_rect()
11
12 # MAIN
13 init()
14 display.set_caption('Starcraft II')
15 window = display.set_mode((600,600))
16
17 protoss = Probe()
18 sprites = Group(protoss)
19 window.fill((255,255,255))
20
21 sprites.draw(window)
22 display.update()
```





# Event-Driven Programming

- **event:** A user interaction with the game, such as a mouse click, key press, clock tick, etc.
- **event-driven programming:** Programs with an interface that waits for user events and responds to those events.
- Pygame programs need to write an *event loop* that waits for a Pygame event and then processes it.

# Event Loop Template

```
# after Pygame's screen has been created
```

```
while True:
```

```
    name = event.wait()           # wait for an event
```

```
    if name.type == QUIT:
```

```
        pygame.quit()           # exit the game
```

```
        break
```

```
    elif name.type == type:
```

```
        code to handle another type of events
```

```
    ...
```

```
    code to update/redraw the game between events
```

```
(like this)
```

```
    sprites.update()
```

```
    sprites.draw(window)
```

```
    display.update()
```

# Event Loop 2

```
# after Pygame's screen has been created
```

```
while True:
```

```
    ev = event.poll()           # Grabs an event
    if ev != NOEVENT:         # If there is an event
        do stuff here
        Check for special events...
```

```
    sprites.update()
    sprites.draw(window)
    display.update()
```

- Note that the sprites' update method gets called more than in the previous example even if no events happen

# Example

- Let's look at the code for the difference between the two event loops

# Mouse Clicks

- When the user presses a mouse button, you get events with a type of `MOUSEBUTTONDOWN` and `MOUSEBUTTONUP`.
  - mouse movement is a `MOUSEMOTION` event
- `mouse.get_pos()` returns the mouse cursor's current position as an `(x, y)` tuple

Example:

```
ev = event.wait()
if ev.type == MOUSEBUTTONDOWN:
    # user pressed a mouse button
    x, y = mouse.get_pos()
```

# Mouse Clicks

- You can also do this through events.
- <http://www.pygame.org/docs/ref/event.html>

```
# after Pygame's screen has been created
ev = event.poll()           # Grabs an event
if ev != NOEVENT:         # If there is an event
    info = ev.dict        # Gives a dict of information

    x, y = info['pos']
```



# Key Presses

- When the user presses a keyboard key, you get events with a type of `KEYDOWN` and then `KEYUP`.
  - event contains `.key` field representing what key was pressed
  - Constants for different keys: `K_LEFT`, `K_RIGHT`, `K_UP`, `K_DOWN`, `K_a` - `K_z`, `K_0` - `K_9`, `K_F1` - `K_F12`, `K_SPACE`, `K_ESCAPE`, `K_LSHIFT`, `K_RSHIFT`, `K_LALT`, `K_RALT`, `K_LCTRL`, `K_RCTRL`, ...

## Example:

```
ev = event.wait()
if ev.type == KEYDOWN:
    if ev.key == K_ESCAPE:
        pygame.quit()
```

# What we want

- We have a sprite on our screen and we have a main loop that doesn't really do anything.
- We **want** to make the sprite move to a location that we click on.
- What do we need to add to the state of our Probe in order to accomplish it?

# Collision Detection

- **collision detection:** Noticing whether one sprite or object has touched another, and responding accordingly.
  - A major part of game programming
- In Pygame, collision detection is done by examining sprites, rectangles, and points, and asking whether they intersect.



# Rect

- a 2D rectangle associated with each sprite (`.rect` field)
  - **Fields:** `top`, `left`, `bottom`, `right`, `center`, `centerx`, `centery`, `topleft`, `topright`, `bottomleft`, `bottomright`, `width`, `height`, `size`, ...

Method Name	Description
<code>collidepoint (p)</code>	returns <code>True</code> if this Rect contains the point
<code>collidect (rect)</code>	returns <code>True</code> if this Rect contains the rect
<code>contains (rect)</code>	returns <code>True</code> if this Rect contains the other
<code>move (x, y)</code>	moves a Rect to a new position
<code>inflate (dx, dy)</code>	grow/shrink a Rect in size
<code>union (rect)</code>	joins two Rects

# Collision Example

- Detecting whether a sprite touches the mouse cursor:

```
ev = event.wait()
if ev.type == MOUSEBUTTONDOWN:
    if sprite.rect.collidepoint(mouse.get_pos()) :
        # then the mouse cursor touches the sprite
        ...
```

- **Exercise:** Detect when the user clicks on the Probe. Make the mole run away by fleeing to a new random location from (0, 0) to (600, 400).

# Exercise Solution

```
class Probe(Sprite):
    def __init__(self):
        Sprite.__init__(self)
        self.image = image.load("probe.png")
        self.rect = self.image.get_rect()

    def flee(self):
        self.rect.left = randint(0, 600)    # random location
        self.rect.top = randint(0, 400)

...

while True:
    ev = event.wait()                    # wait for an event
    if ev.type == QUIT:
        pygame.quit()
        break
    elif ev.type == MOUSEBUTTONDOWN:
        if probe.rect.collidepoint(mouse.get_pos()):
            probe.flee()
```