



Week 10

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>

Inheritance

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

- Example:

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

- Python also supports *multiple inheritance*

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

Calling Superclass Methods

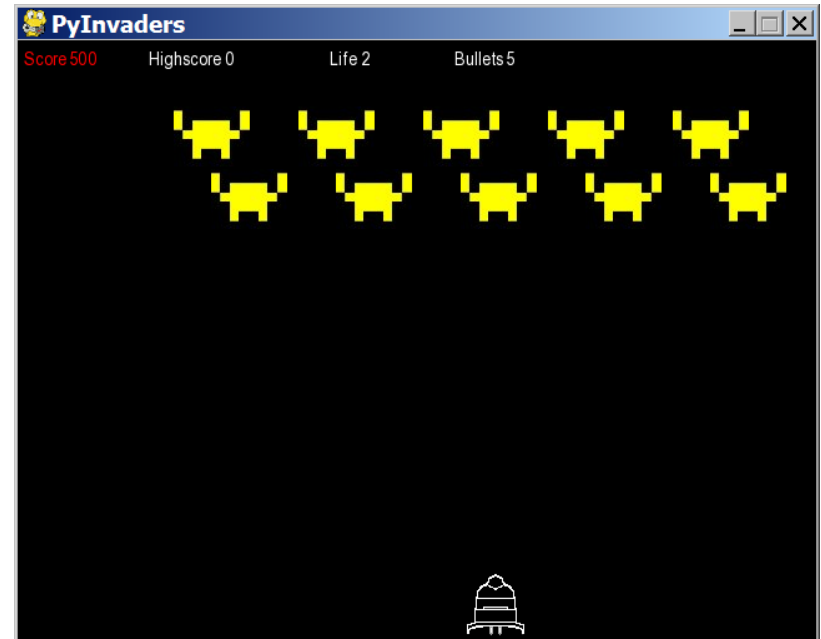
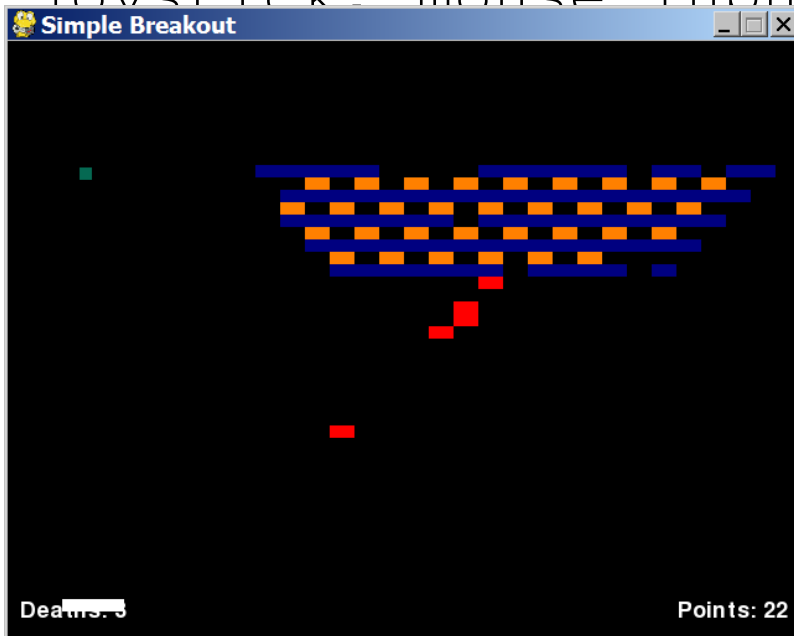
- methods: **class.method(parameters)**
- constructors: **class.__init__(parameters)**

```
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-py2.6.msi)
 - Mac users: under the 'Macintosh' section,
 - click the most recent version
(as of this quarter, pygame-1.9.1release-py2.6-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!

Our Goal: Pong!

- Implement Pong!
 - 800x400 screen
 - 10x10 square ball bounces off of any surface it touches
 - two 10x75 paddles move when pressing Up/Down arrows and W/S
 - game displays score on top/center of screen



Initializing a Game

- Import Pygame's relevant classes:

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

- Initialize Pygame at the start of your code:

```
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:™
python

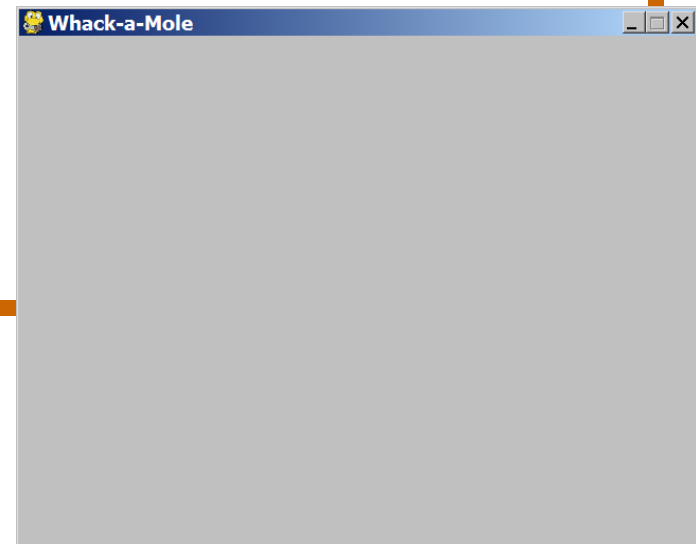
```
screen = display.set_mode((1024, 768), FULLSCREEN)
```

Initial Game Program

- An initial, incomplete game file using Pygame:

pong.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("Pong")
10
11 screen = display.set_mode((1000, 400))
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 `pygame.sprite.Sprite` objects that extend the `Sprite`



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



python™

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.

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

Sprite Example

A class for a mole sprite to be whacked.

```
class Mole(Sprite):  
    def __init__(self):  
        Sprite.__init__(self)  
        self.image = image.load("mole.gif")  
        self.rect = self.image.get_rect()
```

-What about our Ball?

Sprite Groups

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

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

Example:

```
my_mole = Mole()      # create a Mole object  
all_sprites = Group(my_mole)
```

Group methods:

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

```
my_mole = Mole()      # create a Mole object
all_sprites = Group(my_mole)
all_sprites.draw(screen)
display.update()    # redraw to see the sprites
```


Doing time!

- Create sprite for the pong ball
- Get it moving!
- Start on paddles

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 # ← not a big fan  
    elif name.type == type:  
        code to handle another type of events  
    ...  
  
    code to update/redraw the game between  
events
```

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.poll() # or even.wait()
if ev.type == MOUSEBUTTONDOWN:
    # user pressed a mouse button
    x, y = mouse.get_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.poll() # or even.wait()
if ev.type == KEYDOWN:
    if ev.key == K_ESCAPE:
        pygame.quit()
```

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 True if this Rect contains the point
<code>collidect(rect)</code>	returns True if this Rect contains the rect
<code>contains(rect)</code>	returns True 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
        ...
```

-Write a method of paddles to see if the ball hit it

Font

- Text is drawn using a Font object:

```
name = Font(filename, size)
```

- Pass None for the file name to use a default font.

- A Font draws text as a Surface with its render method:

```
name.render("text", True, (red, green, blue))
```

Example:

```
my_font = Font(None, 16)
```

```
text = my_font.render("Hello", True, (0, 0, 0))
```

Displaying Text

- A Sprite can be text by setting that text's Surface to be its `.image` property.

Example:

```
class Banner(Sprite):
    def __init__(self):
        my_font = Font(None, 24)
        self.image = my_font.render("Hello", \
                                   True, (0, 0, 0))
        self.rect = self.image.get_rect()
        self.rect.center = (250, 170)
```

Exercise

- Implement scoring of points in PyPong.
 - Make a sprite to represent the current scoreboard.
 - Draw the score in 72px font, in the top/middle of the board.
 - Draw it in a format such as "0:0".
 - Expand the collision detection for the ball:
 - If it hits the right wall, it should score a point for Player 1.
 - If it hits the left wall, it should score a point for Player 2.

Sounds

- Loading and playing a sound file:

```
from pygame.mixer import *  
mixer.init()           # initialize sound system  
mixer.stop()          # silence all sounds  
  
Sound("filename").play() # play a sound
```

- Loading and playing a music file:

```
music.load("filename") # load bg music file  
music.play(loops=0)    # play/loop music  
                        # (-1 loops == infinite)
```

others: stop, pause, unpause, rewind, fadeout,

Further Exploration

- Physics: Sprites that accelerate; gravity; etc.
- AI: Computer opponents that play "intelligently"
- Supporting other input devices
 - See documentation for Pygame's Joystick module
- Multi-player (local or network)