Week 9

Writing Games with Pygame

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

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
```

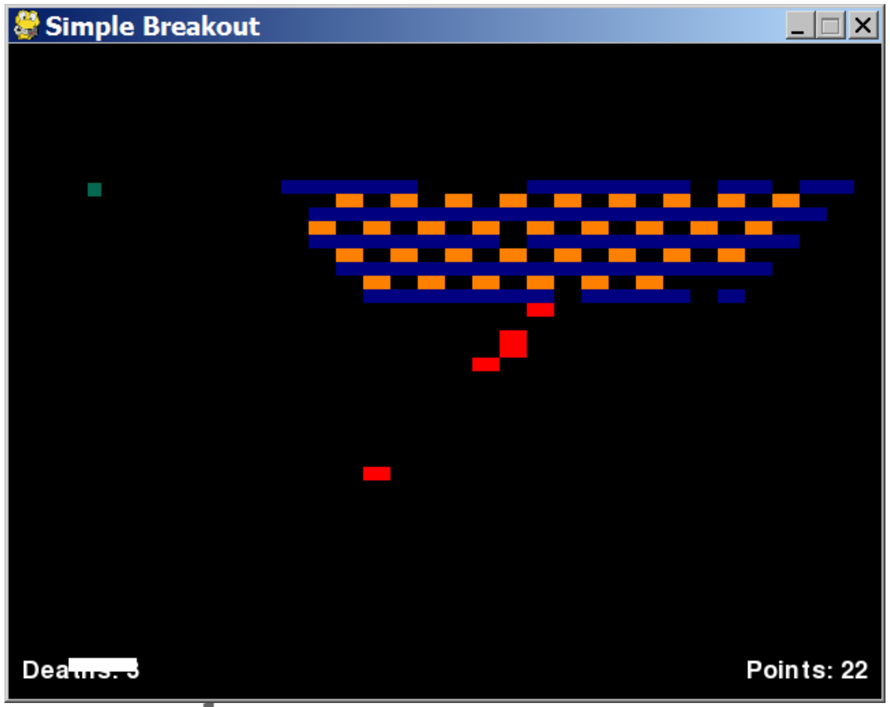
Calling Superclass Methods

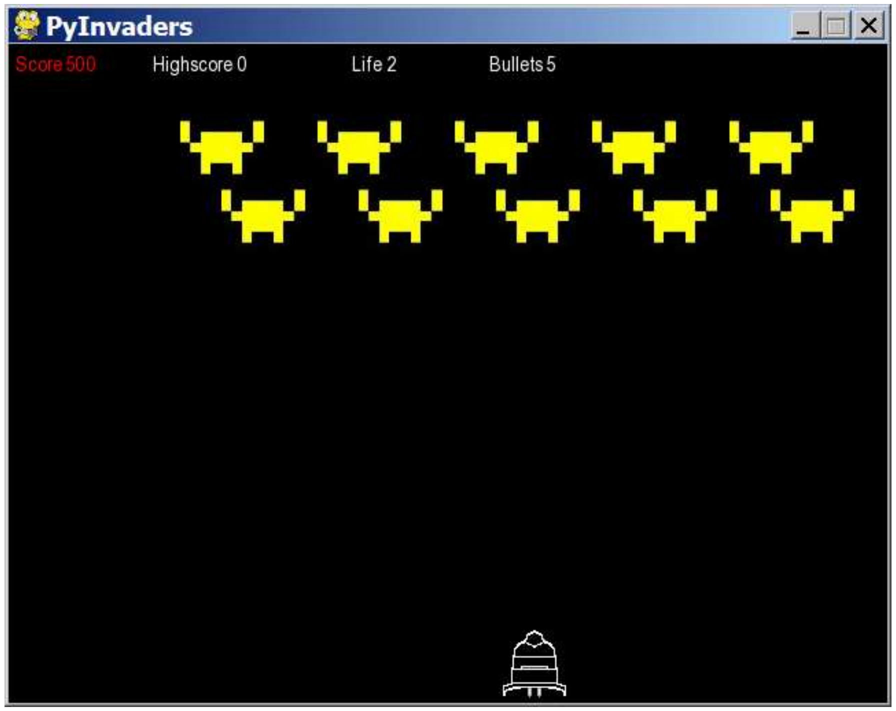
- methods: **class**.**method**(**parameters**)
- constructors: **class**.`__init__`(**parameters**)

```
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

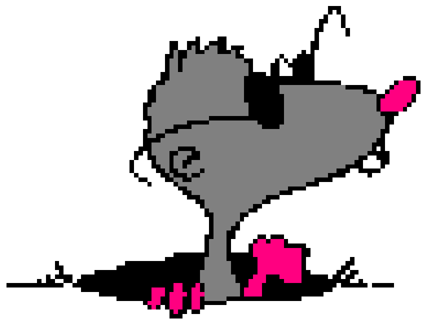
- Go to the Pygame web site: http://www.pygame.org/
  - click 'Downloads' at left
  - Windows users: under the 'Windows' section,
    - click the most recent version
      (as of this quarter, that is pygame-1.8.1.win32-py2.6.msi)
  - Mac users: under the 'Macintosh' section,
    - click the most recent version
      (as of this quarter, pygame-1.8.1release-py2.5-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: Whack-a-Mole

- Clicking on the mole plays a sound and makes mole move
- Number of hits is displayed at top of screen
- Enhancements
  - hit the mole with a shovel cursor
  - make the mole move around every 1 second if he's not hit



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:

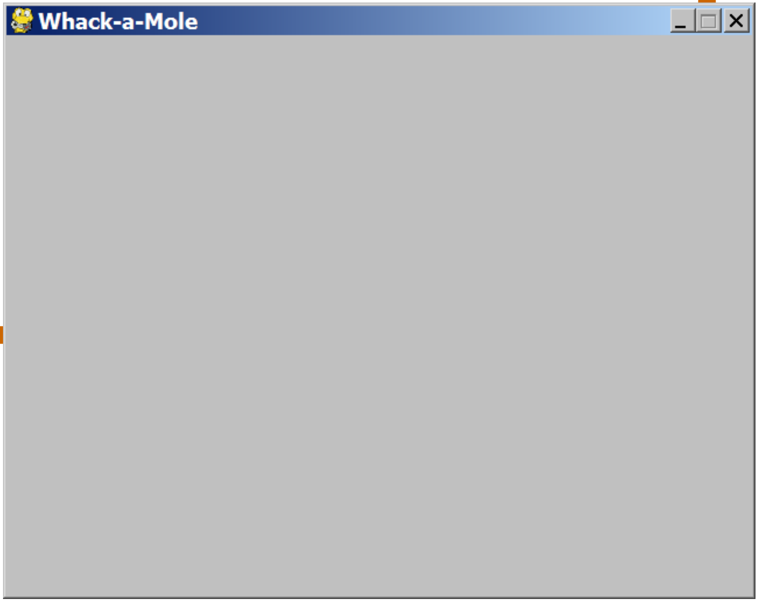
```
screen = display.set_mode((1024, 768), FULLSCREEN)
```

Initial Game Program

- An initial, incomplete game file using Pygame:

whack_a_mole.py

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

pygame.init()

# set window title
display.set_caption("Whack-a-Mole")

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

```

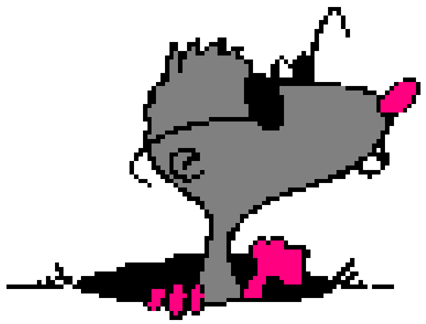


Sprites

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 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()
```

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:

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

python™

Drawing and Updating

- All Surface and Group objects have an `update` method that redraws that object when it moves or changes.

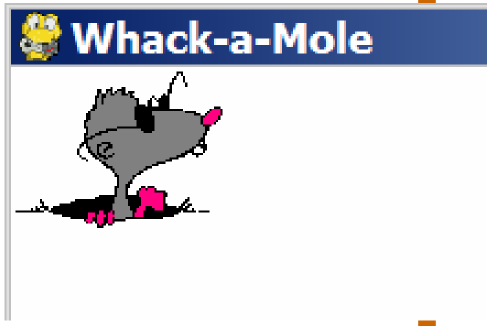
- Once sprites are drawn onto the screen, you must call `display.update()` to see the changes

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

Game Program v2

whack_a_mole.py

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

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

# main
pygame.init()
display.set_caption("Whack-a-Mole")
screen = display.set_mode((640, 480))

my_mole = Mole()                  # initialize sprites
all_sprites = Group(my_mole)
screen.fill((255, 255, 255))      # white background
all_sprites.draw(screen)
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
```

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()
```

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()
```

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.

python™

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**
`collidepoint(`**p**`)`	returns `True` if this Rect contains the point
`colliderect(`**rect**`)`	returns `True` if this Rect contains the rect
`contains(`**rect**`)`	returns `True` if this Rect contains the other
`move(`**x**`, `**y**`)`	moves a Rect to a new position
`inflate(`**dx**`, `**dy**`)`	grow/shrink a Rect in size
`union(rect)`	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 Mole. Make the mole run away by fleeing to a new random location from (0, 0) to (600, 400).

Exercise Solution

```
class Mole(Sprite):
    def __init__(self):
        Sprite.__init__(self)
        self.image = image.load("mole.gif")
        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 my_mole.rect.collidepoint(mouse.get_pos()):
            my_mole.flee()
    ...
```