python™

Week 10

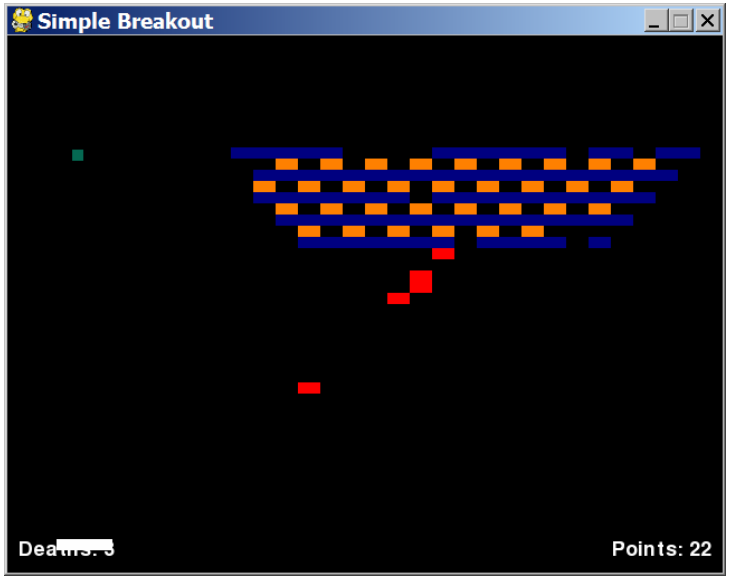
Writing Games with Pygame, continued

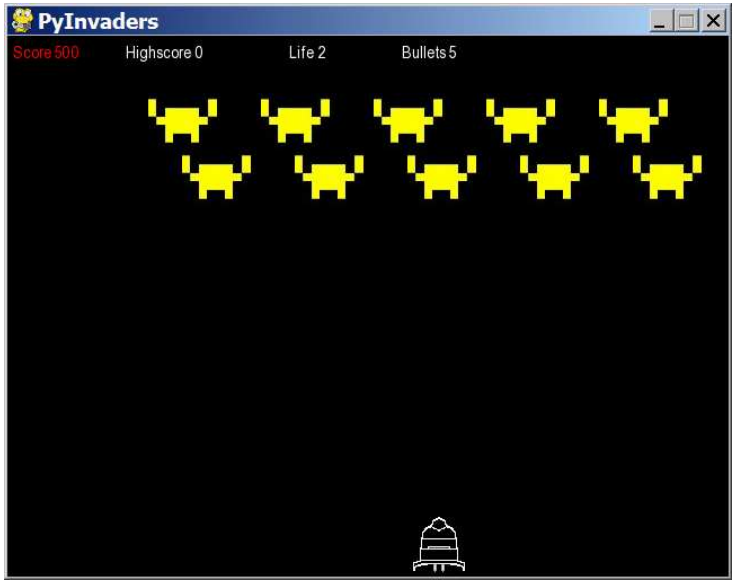
Special thanks to Scott Shawcroft, Ryan Tucker, and Paul Beck for their work on these slides.

Except where otherwise noted, this work is licensed under:
http://creativecommons.org/licenses/by-nc-sa/3.0

Animated Action Games

- Consider an action game such as Space Invaders or Mario. How does it differ from our Whack-a-Mole game?
  - What features are present in an action game?
  - What are some major challenges in writing such a game?





Our Task

- Implement Pong!
  - 800x480 screen, 10px white border around all edges
  - 15x15 square ball bounces off of any surface it touches
  - two 20x150 paddles move when holding Up/Down arrows
  - game displays score on top/center of screen in a 72px font

Major Steps

1. Set up the (non-moving) paddle/ball/wall sprites
2. Handle key presses, sprite movement, and animation
3. Handle collisions
4. Scoring

python™

Step 1

Setting Up Sprites

Recall: Sprites

```
class name(Sprite):
    # constructor
    def __init__(self):
        Sprite.__init__(self)
        self.image = image.load("filename")
        # or, self.image = Surface((w, h))
        self.rect = self.image.get_rect()
        self.rect.center = (x, y)

    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, which includes shapes and images

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

Recall: `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
`Surface((`**width**`, `**height**`))`	constructs new Surface of given size
`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

RectangularSprite

- Suggested template for simple white rectangular sprites:

```
class RectangularSprite(Sprite):
    def __init__(self, size, center):
        Sprite.__init__(self)
        self.image = Surface(size)
        self.image.fill((255, 255, 255))
        self.rect = self.image.get_rect()
        self.rect.center = center
```

  - Now the various sprites in the Pong game can be `RectangularSprites` or extend `RectangularSprite` to add their own behavior

Recall: 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
other_mole = Mole()
all_sprites = Group(my_mole, other_mole)
```

Group methods:

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

Exercise

- Define the sprites for the PyPong game:
  - four 15px-thick borders around the 800x480 board edges
  - two 20x150 paddles, centered vertically, at L/R edges of board
  - a 15x15 ball, in the center of the board

    - Use `RectangularSprite` as the basis for your sprites.
    - The sprites don't move yet.

python™

Step 2

Animation and Key Presses

Recall: Event Loop

```
# 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
```

Timer Events

```
time.set_timer(USEREVENT, delayMS)
```

- Animation is done using **timers**
  - Events that automatically occur every *delayMS* milliseconds
  - Your event loop can check for these events. Each one is a "frame" of animation

```
while True:
    ev = event.wait()
    if ev.type == USEREVENT:
        # the timer has ticked; move sprites,
        # redraw the screen, etc.
```

Key Presses

http://www.pygame.org/docs/ref/key.html

- `key.get_pressed()` returns an array of keys held down
  - indexes are constants like `K_UP` or `K_F1`
  - values are booleans (`True` means pressed)
  - Constants for 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`, ...

```
keys = key.get_pressed()
if keys[K_LEFT]:
    # left arrow is being held down...
```

Updating Sprites

```
class Jet(Sprite):
    def __init__(self):
        # ...

    def update(self):   # move right 3px / tick
        self.rect = self.rect.move(3, 0)
```

- Each sprite can have an `update` method that describes how to move that sprite on each timer tick.
  - Move a rectangle by calling its `move(`**`dx`**`, `**`dy`**`)` method.
  - Calling `update` on a `Group` updates all its sprites.

Exercise

- Implement animation and key response in PyPong:
  - Make a timer that ticks every 50 ms.
  - When the timer ticks:
    - Give the ball a dx/dy of 5px and move the ball by that amount. (The ball will fly off the screen after a moment.)
    - If the up arrow is held down, move the paddles up by 5px.
    - If the down arrow is held down, move the paddles down by 5px.

python™

python™

Step 3

Collisions Between Sprites

Collisions Btwn. Rectangles

- Recall: Each `Sprite` contains a `Rect` collision rectangle
- `Rect` objects have useful methods for detecting collisions between the rectangle and another sprite:

Method Name	Description
`collidepoint(`**`p`**`)`	returns `True` if this Rect contains the point
`colliderect(`**`rect`**`)`	returns `True` if this Rect contains the rect

  - However, `Sprite` and `Group` objects have more useful methods to detect collisions...

Collisions Between Groups

`spritecollideany(`**`sprite`**`, `**`group`**`)`

– Returns `True` if sprite has collided with any sprite in the group

- Useful for finding collisions in a sprite's `update` method:

```
class name(Sprite):
    def update(self):
        if spritecollideany(self, group):
            # I collided with a sprite in group
```

Exercise

- Implement collision response in PyPong:
  - Constrain the paddles; if a paddle collides with one of the top/bottom borders, stop its movement.
  - Make the ball bounce off of the other sprites on the board:
    - If it hits the top or bottom walls, it should invert its y direction.
    - If it hits a paddle, it should invert its x direction.

python™

Step 4

Scoring, Polish, etc.

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.

python™

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`, `queue`

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)

python™