python™

Unit 9

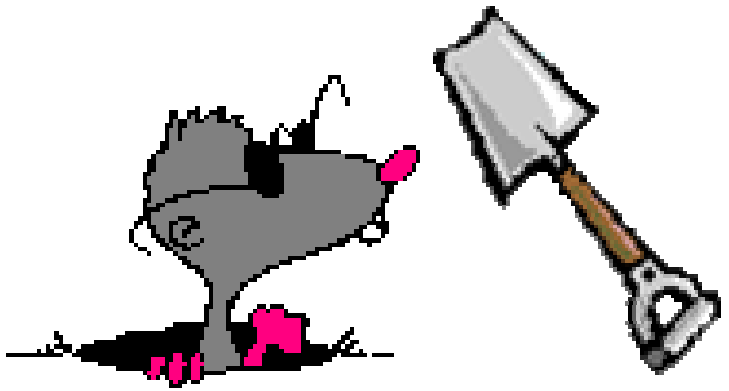
pyGame

Special thanks to Roy McElmurry, John Kurkowski, Scott Shawcroft, Ryan Tucker, Paul Beck for their work.
Except where otherwise noted, this work is licensed under:
http://creativecommons.org/licenses/by-nc-sa/3.0

Exercise: Whack-a-mole

- Goal: Let's create a "whack-a-mole" game where moles pop up on screen periodically.
  - The user can click a mole to "whack" it. This leads to:
    - A sound is played.
    - The player gets +1 point.
    - A new mole appears elsewhere on the screen.

- The number of points is displayed at the top of the screen.



What is pyGame?

- A set of Python modules to make it easier to write games.
  - home page: http://pygame.org/
  - documentation: http://pygame.org/docs/ref/
- pyGame helps you do the following and more:
  - Sophisticated 2-D graphics drawing functions
  - Deal with media (images, sound F/X, music) nicely
  - Respond to user input (keyboard, joystick, mouse)
  - Built-in classes to represent common game objects

python™

pyGame at a glance

- pyGame consists of many **modules** of code to help you:

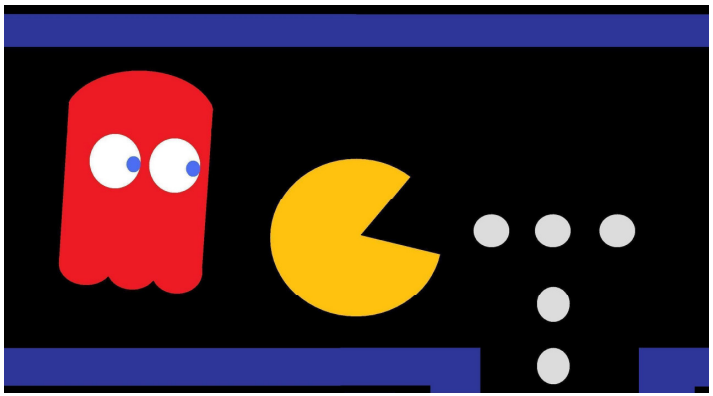
```
cdrom   cursors   display     draw   event
font    image     joystick    key    mouse
movie   sndarray  surfarray   time   transform
```

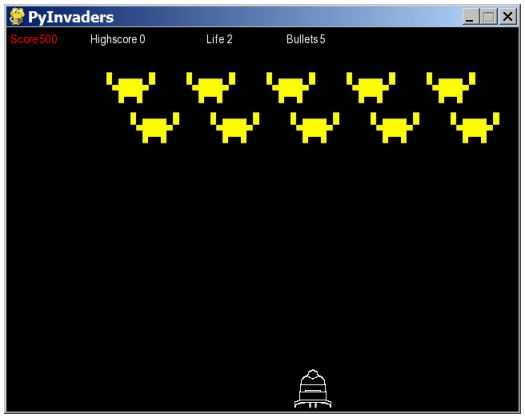
- To use a given module, **import** it. For example:

```
import pygame
from pygame import *
from pygame.display import *
```

Game fundamentals

- **sprites**: Onscreen characters or other moving objects.
- **collision detection**: Seeing which pairs of sprites touch.
- **event**: An in-game action such as a mouse or key press.
- **event loop**: Many games have an overall loop that:
  - **waits** for events to occur, **updates** sprites, **redraws** screen





python™

A basic skeleton

pygame_template.py

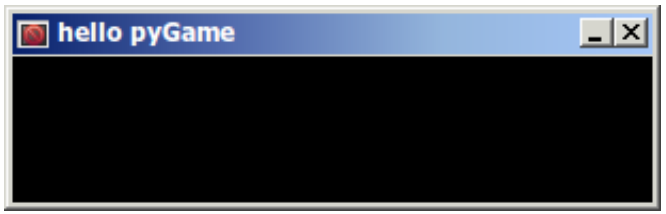
```
1  from pygame import *
2  from pygame.sprite import *
3
4  pygame.init()                              # starts up pyGame
5  screen = display.set_mode((width, height))
6  display.set_caption("window title")
7
8  create / set up sprites.
9
10 # the overall event loop
11 while True:
12     e = event.wait()                       # pause until event occurs
13     if e.type == QUIT:
14         pygame.quit()                      # shuts down pyGame
15         break
16
17     update sprites, etc.
18     screen.fill((255, 255, 255))           # white background
19     display.update()                       # redraw screen
```

Initializing pyGame

- To start off our game, we must pop up a graphical window.
- Calling `display.set_mode` creates a window.
  - The call returns an object of type `Surface`, which we will call `screen`. We can call methods on the screen later.
  - Calling `display.set_caption` sets the window's title.

```
from pygame import *

pygame.init()    # starts up pyGame
screen = display.set_mode((width, height))
display.set_caption("title")
...
pygame.quit()
```



Surfaces

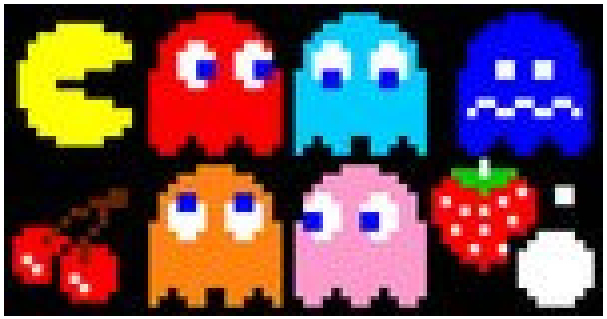
```
screen = display.set_mode((width, height))    # a surface
```

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

`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(`**surface**`,` **coords**`)`	draws another surface onto this surface at the given coordinates

  - after changing any surfaces, must call `display.update()`

Sprites



- **Sprites**: Onscreen characters or other moving objects.

- A sprite has data/behavior such as:
  - its **position** and **size** on the screen
  - an **image** or shape for its appearance
  - the ability to **collide** with other sprites
  - whether it is **alive** or on-screen right now
  - might be part of certain "**groups**" (enemies, food, ...)

- In pyGame, each type of sprite is represented as a subclass of the class `pygame.sprite.Sprite`

A rectangular sprite

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

class name(Sprite):
    def __init__(self):                    # constructor
        Sprite.__init__(self)
        self.image = Surface(width, height)
        self.rect = Rect(leftX, topY, width, height)

    other methods (if any)
```

- Important fields in every sprite:
  - `image` - the image or shape to draw for this sprite (a `Surface`)
    - as with screen, you can `fill` this or draw things onto it
  - `rect` - position and size of where to draw the sprite (a `Rect`)
- Important methods: `update`, `kill`, `alive`

Rect methods

`clip(`**rect**`)` *	crops this rect's size to bounds of given rect
`collidepoint(`**p**`)`	`True` if this `Rect` contains the point
`colliderect(`**rect**`)`	`True` if this `Rect` touches the rect
`collidelist(`**list**`)`	`True` if this `Rect` touches any rect in the list
`collidelistall(`**list**`)`	`True` if this `Rect` touches all rects in the list
`contains(`**rect**`)`	`True` if this `Rect` completely contains the rect
`copy()`	returns a copy of this rectangle
`inflate(`**dx**`, `**dy**`)` *	grows size of rectangle by given offsets
`move(`**dx**`, `**dy**`)` *	shifts position of rectangle by given offsets
`union(`**rect**`)` *	smallest rectangle that contains this and rect

* Many methods, rather than mutating, return a new rect.
  - To mutate, use `_ip` (in place) version, e.g. `move_ip`

A Sprite using an image

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

class name(Sprite):
    def __init__(self):                    # constructor
        Sprite.__init__(self)
        self.image = image.load("filename").convert()
        self.rect = self.image.get_rect().move(x, y)

    other methods (if any)
```

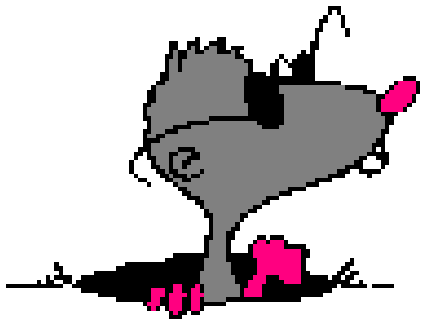
- When using an image, you load it from a file with `image.load` and then use its size to define the `rect` field
- Any time you want a sprite to move on the screen, you must change the state of its `rect` field.

Setting up sprites

- When creating a game, we think about the sprites.
  - What sprites are there on the screen?
  - What data/behavior should each one keep track of?
  - Are any sprites similar? (If so, maybe they share a class.)

- For our Whack-a-Mole game:

```
class Mole(Sprite):
    ...
```



Sprite groups

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

- To draw sprites on screen, put them into a `Group`
- Useful methods of each `Group` object:

  `draw(`**surface**`)` - draws all sprites in group on a `Surface`

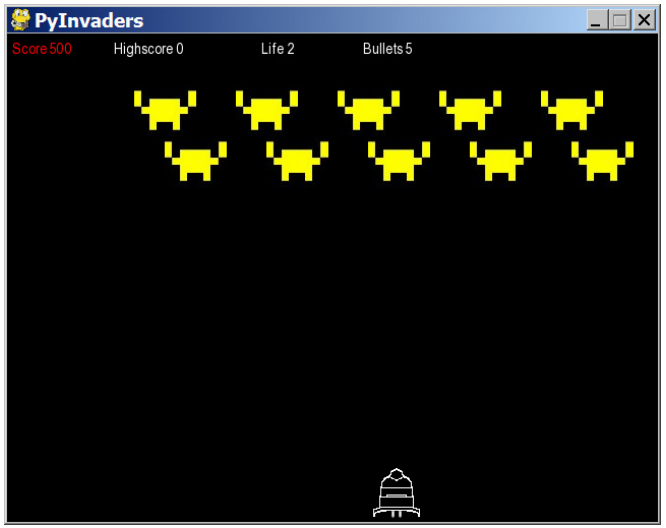
  `update()` - calls every sprite's `update` method

```
my_mole1 = Mole()     # create a Mole object
my_mole2 = Mole()
all_sprites = Group(my_mole1, other_mole2)
...
# in the event loop
all_sprites.draw(screen)
```

Events

- **event-driven programming**: When the overall program is a series of responses to user actions, or "events."

- **event loop** (aka "main loop", "animation loop"):
  Many games have an overall loop to do the following:
  - **wait** for an event to occur, or
    wait a certain interval of time
  - **update** all game objects (location, etc.)
  - **redraw** the screen
  - repeat



The event loop

- In an *event loop*, you wait for something to happen, and then depending on the kind of event, you process it:

```
while True:
    e = event.wait()          # wait for an event
    if e.type == QUIT:
        pygame.quit()         # exit the game
        break
    elif e.type == type:
        code to handle some other type of events;
    elif ...
```

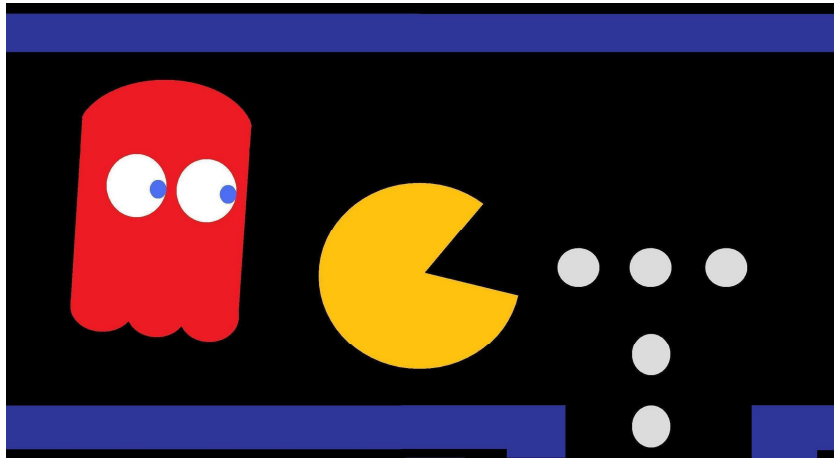
Mouse events

- Mouse actions lead to events with specific types:
  - **press** button down: `MOUSEBUTTONDOWN`
  - **release** button: `MOUSEBUTTONUP`
  - **move** the cursor: `MOUSEMOTION`
- At any point you can call `mouse.get_pos()` which returns the mouse's current position as an (x, y) tuple.

```
e = event.wait()
if e.type == MOUSEMOTION:
    pt = mouse.get_pos()
    x, y = pt
    ...
```

Collision detection

- **collision detection**: Examining pairs of sprites to see if they are touching each other.
  - e.g. seeing whether sprites' bounding rectangles intersect
  - usually done after events occur, or at regular timed intervals
  - can be complicated and error-prone
    - optimizations: *pruning* (only comparing some sprites, not all), ...



Collisions btwn. rectangles

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

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

```
if sprite1.rect.colliderect(sprite2.rect):
    # they collide!
    ...
```

Collisions between groups

global pyGame functions to help with collisions:

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

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

`spritecollide(`**sprite**`,` **group**`,` **kill**`)`

- Returns a list of all sprites in **group** that collide with **sprite**
- If **kill** is `True`, a collision causes **sprite** to be deleted/killed

`groupcollide(`**group1**`,` **group2**`,` **kill1**`,` **kill2**`)`

- Returns list of all sprites in **group1** that collide with **group2**

Drawing text: 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

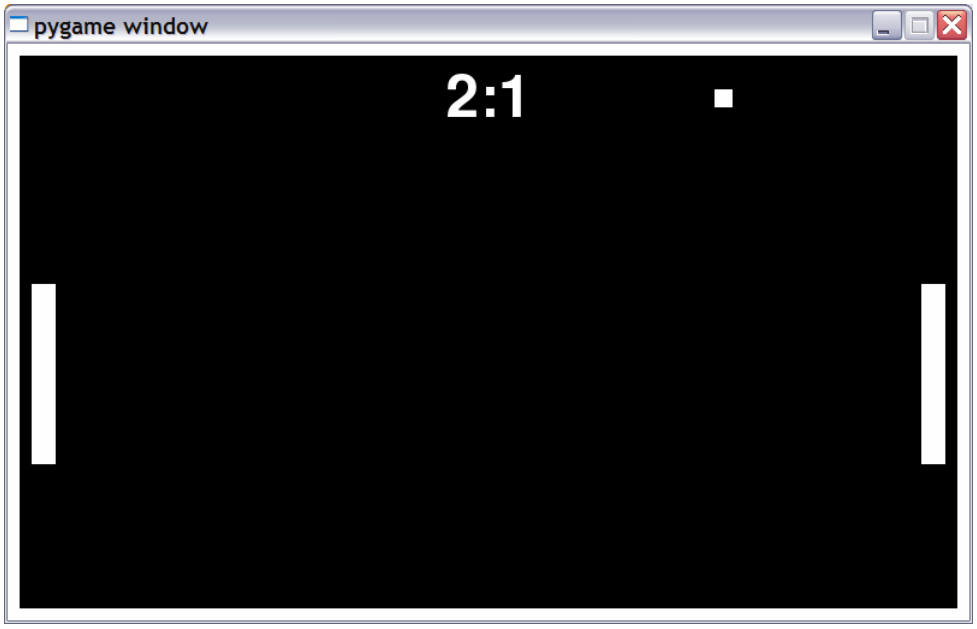
- A `Sprite` can be text by setting that text's `Surface` to be its `.image` property.

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().move(50,70)
```

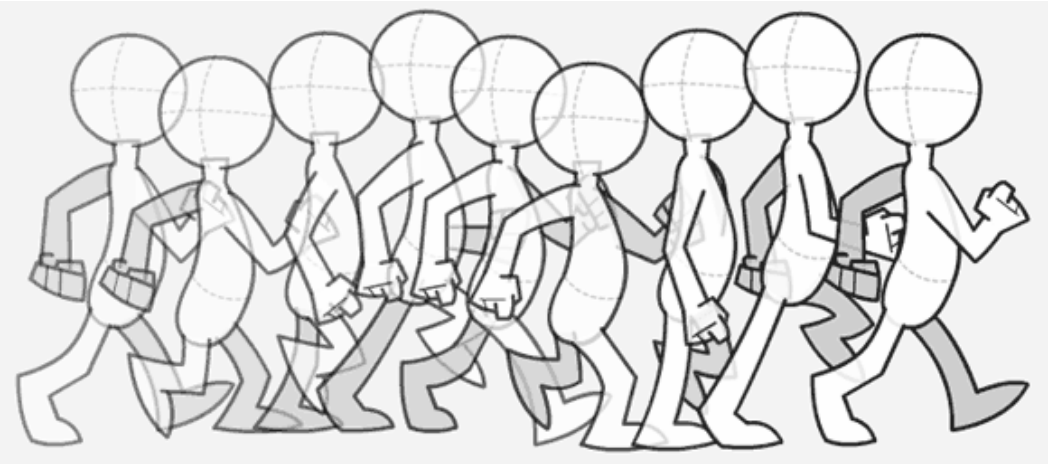
Exercise: Pong

- Let's create a Pong game with a bouncing ball and paddles.
  - 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



Animation

- Many action games, rather than waiting for key/mouse input, have a constant **animation** timer.
  - The timer generates events at regular intervals.
  - On each event, we can move/update all sprites, look for collisions, and redraw the screen.



Timer events

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

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

```
while True:
    e = event.wait()
    if e.type == USEREVENT:
        # the timer has ticked
        ...
```

Key presses

- key presses lead to `KEYDOWN` and `KEYUP` events
- `key.get_pressed()` returns an array of keys held down
  - the array indexes are constants like `K_UP` or `K_F1`
  - values in the array 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_down = key.get_pressed()
if keys_down[K_LEFT]:
    # left arrow is being held down
    ...
```

Updating sprites

```
class name(Sprite):
    def __init__(self):
        ...

    def update(self):  # right by 3px per 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.

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`

python™

The sky's the limit!

- pygame.org has lots of docs and examples
- can download tons of existing games
  - run them
  - look at their code for ideas
- if you can imagine it,
  you can create it!