Pygame

Games in Python – the easy way
The pyGame Package

• A set of Python modules to help write games

• Deals with media nicely (pictures, sound)

• Interacts with user nicely (keyboard, joystick, mouse input)

• Lots of more advanced features for working with graphics, etc
Where to Start?

- The official pyGame website
- Search for tutorials
- The Application Programming Interface (API)
  - specifies the classes and functions in package
- Experiment!
• Tutorials basically all have the same setup -- let's use it!

```python
from pygame import * from pygame.sprite import * from random import *
init()

screen = display.set_mode((640, 480)) display.set_caption('Window name!')
while True:
    e = event.poll()
    if e.type == QUIT:
        quit()
        break
    screen.fill(Color("white"))
    display.update()
```
• Most of the game elements you see are represented as Surface objects.

• `display.set_mode((x, y))` creates your canvas – it returns a Surface object.

Useful surface methods:
• `fill("color")` fills the surface object it's called on.

• `blit(surface, area)` paints `surface` onto the object `blit` is called on in the rectangle bounded by the `area` tuple.
  
  – Example: `screen.blit(ball, (50,50))`
• Objects that store rectangular coordinates

• Call `.get_rect()` on a surface to get its bounding box

Rectangle methods/variables:
• `.center` holds the object's center as a tuple

• `.colliderect(target)` returns True if the parameter overlaps with the object

• `.collidepoint(target)` returns True if the target point overlaps with the object
• Loading an image:
  - `img = Image.load("file.gif").convert()`

• Getting a bounding rectangle:
  - `img_rect = img.get_rect()`

• Loading and playing a sound file:
  - `mixer.Sound("file.wav").play()`
• Class visible game objects inherit from

```python
from pygame import *rom pygame.sprite import *

class Ball(Sprite):
    def __init__(self):
        Sprite.__init__(self)
        self.image = image.load("ball.png").convert()
        self.rect = self.image.get_rect()

    def update(self):
        self.rect.center = mouse.get_pos()
```
Using Sprites

• They're just objects: initialize them
  - ball = Ball()

• Create a group of sprites in main
  - sprites = RenderPlain(sprite1, sprite2)

• Groups know how to draw and update
  - sprites.update()
  - sprites.draw(surface)
Events

- User input such as clicking, moving mouse or key presses

- Add more branches to test the result of `event.poll()`

- Events to test for:
  - QUIT
  - MOUSEBUTTONDOWN
  - JOYBUTTONDOWN

- Testing for the letter ‘d’ being pressed using `KEYDOWN`
  ```python
  if e.type == KEYDOWN:
      if e.key == K_d:
  ```
Adding Text

• \( f = \text{font.Font(}\text{font, size}) \) goes before your game loop
  – Example: \( f = \text{font.Font(}\text{None, 25}) \)
  – Usually, \textbf{None} is a good enough font!

• \( \text{text} = \text{Font.render(}\text{text, antialias, color}) \)
  – Example: \( \text{text} = f.\text{render(}"Hello!", \text{True, Color("green")}) \)
  – Returns a surface

• Must be blit, just like any other surface
  – Example: \( \text{screen.blit(t, (320, 0))} \)
Exercise: Whack-a-mole

• Clicking on the mole
  – plays a sound
  – makes the mole move

• The number of hits is displayed at the top of the screen
Or make your own

• You can make your own game but DESIGN IT CAREFULLY!
  – You must have something to show by the last day of class
  – It doesn’t need to be complete, but should be playable

• Requirements:
  – At least two sprites
  – Interaction between sprites
  – User input from keyboard or mouse
  – Some kind of score displayed

• Have fun!!

python™