* Taste
  + Must initialize the PyGame code before preceding
* Creating a graphical Window
  + Pygame.display.set\_mode(size)
    - Size is a variable that was prevouisly used
    - Display.set\_mode creates a surface object that represents the actual displayed graphics
    - Anything we do to this will become visible on the monitor
* Loading Object (This case a ball)
  + Supports JPG, PNG, TGA, GIF images
  + Pygame.image.load(file name) retunrs us a surface with the ball data
  + PyGame comes with a utility object named Rect
    - Represents a rectangular area
* Inside infinite loop->check for user input
  + Moving
  + Need to implement a QUIT event
* Movement
  + If ball has moved outside the screen, we reverse the speed in that that direction
    - Moves ball back into field of play
* Erasing Ball
  + Must erase the ball after each movement or else trail would exist
    - Screen.fill(black) within the infinite loop to erase past image
* Drawing Image
  + Screen.blit(ball, ballreact)
    - Filling the screen with the image
  + Handled with the surface.blit() method
    - In this case the screen variable is the surface
* Updating the Visible Display
  + Pygame.display.flip() method
  + Makes everything we have drawn become visible
* Recognize Which parts of pygame you really need
  + Realize what classes/functions I really need for my specific game
* Know What a Surface Is
  + Most import part of pygame
  + Think of it as a blank piece of paper
  + Can have as many surfaces as you like
  + Create a surface that contains an image
    - Image.load()
  + Or with text
    - Font.render()
  + Critical Surface functions
    - Blit()
    - Fill()
    - Set\_at()
    - Get\_at()
* Use Surface.convert()
  + Not converting the file format(PNG, JPG, etc.), converting the “pixel format”
  + Need to use if you want to get any kind of speed out of your blits
  + Used by calling it after creating a surface with image.load() function
    - Surface = pygame.image.load(‘foo.png’).convert()
  + Needs to be called once per surface
    - Just include in loop
* Dirty rect Animation
  + Must call pygame.display.update() for image to appear on the screen
  + 3 methods for using this
    - pygame.display.update()- updates the whole screen
    - pygame.display.flip()- Does the same thing, and wil also do the right thing if youre using doublebuffered hardware acceleration
    - pygame.display.update(a rectangle or some list of rectangles)-updates just the rectangular areas of the screen you specify
  + How to use dirty\_rect
    - Blit a piece of the background over the sprite’s current location, erasing it
    - Append the sprite’s current location rectangle to a list called dirty\_rects
    - Move the sprite
    - Draw the sprite at it’s new location
    - Append the sprite’s new location to my\_dirty rects list
    - Call display.update(dirty\_rects)

Rects are you friends

* + rect = pygame.Rect(10, 20, 30, 30)
  + Finding the specific location of all sprites
    - Sprites\_clicked = [sprite for spreite in all\_my\_sprites\_list if sprite.rect.collidepoint(x,y)]
* Don’t bother with pixel-perfect collision detection
  + Use sub-rect collision
  + Creates a rect for each sprite that is slightly smaller than the actual image and use that for collision
* Managing the event subsystem
  + Pygame.key.get\_pressed()
    - Will tell you the state of that device at the moment you call the function
  + State-checking system detects “chording” easily
    - Several states at the same time
  + If you want to kow if the t and f keys are down at the same time
    - If (key.get\_pressed[K\_t] and key.get\_pressed[K\_f]):
      * Print “Yup”
* How to make sprites that aren’t rectangular in shape
  + Colorkey
    - Surface.set\_colorkey(color)
    - Color is a rgb tuple say (0,0,0)
      * This would make every pixel in the source image transparent instead of black
  + Alpha
    - “Image alpha” applies to the entire picture
* 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
* Colors
  + Colors based on RGBA (red, green, blue, alpha)
  + **white:** (255,255,255)
  + **black:** (0,0,0)
  + **red:** (255,0,0)
  + **green:** (0,255,0)
  + **blue:** (0,0,255)
  + **grey:** (192,192,192)
  + **yellow:** (255,255,0)
  + Alpha-transparency is the opacity.
* Surfaces
  + blit(surface, cords)- draws another surface onto this surface at the given coordinates
  + After changing any surface must use display.update()
* Sprites
  + Each tupe of sprites is represented as subclass of the class pygame.**s**prite.**S**prite
* Mouse Actions
  + Press button down: MOUSEBUTTONDOWN
  + Release button: MOUSEBUTTONUP
  + Move the cursor: MOUSEMOTION
  + Can get the
* How big should I make my window?
* Is the pygame.display.set\_mode how we change the background image?
* Can I put movement restrictions(walls) on the gameplay window?
* Is the ballrect variable built in and what is the asteroid equivalent?
* What classes and functions will I need?
* How do I test this code to make sure that it working properly as I go?
* How should I manage the event subsystem?
  + State-system
  + Event queue
* When will it be useful to use the mouse.get\_pos() call?
* Game
  + Ship
    - Speight
    - Could add another ship if reach certain point
      * Jake Butt
  + Bullets
    - Footballs
  + Asteroids
    - OSU/Big Ten team logos
  + Movement
    - Being able to move ship left/right
  + Collision Detection
    - Score is added when “bullet” hits ship
  + Scoring and/or time
    - Could use basic 1 point scoring
    - Could also try to implement football scoring
      * 7 points for a “touchdown”/3 points for a “field goal”
    - Program should “quit” if lives are lost
    - Maybe change the surface from stadium to stadium as increase in levels
  + Sound
    - Use Michigan fight song/other chants as background music for game
  + Animation
    - Image/Ship just needs to move
  + Unique Implementation of Class Inheritance
    - Would be creation of another ship when point total is reached
  + Score
    - Touchdown/field goal etc.
* “Most of the surface functions are not critical. Just learn blit(), fill(), set\_at() and get\_at(), and you’ll be fine.”
* One ship should collide with 1 rectangle, make sure something is registred
  + Some sort of flagging for collision
* When there’s harder problem you cant solve, there’s an easier you can’t
* Start with 1 thing then move on to multiple
* After solidifying mechanics add images for rectangles/surfaces
  + Dirty rec newbie guide
* Commit success
  + Only need 1-2 sentences
* WorkFlow
  + Get ship on screen
  + Get ship moving
    - Part of ship movement is assignging keyboard/mpuse user event
  + Construct a collision
    - Put a stationary rectangle on screen make ship crash into it
  + Bullet would be another class within ship class
    - Method of ship would be fire
    - Would need to know coordinates/direction on the screen
  + 1 ship, firing 1 bullet, hitting 1 rectangle
    - Need some sort of detonation effect for ship colliding with rectangle and bullet colliding with rectangle
    - Collision, your objects must know which of them must detonate
      * Aka “who dies”
  + Add scoring
    - When rectangle is hit increase score by 1
    - 1-2 lines then retest
    - Score = 0
    - Score += 1
  + Fire multiple bullets from ship on single rectangle
    - Issues
      * Rate of fire-how quickly can I fire
        + Let user adjust rate of fire in free play mode
        + Left click for ridiculously fast rate of fire
      * Direction of fire is updated
      * Collision recognition with rectangle
      * Rate of travel of bullet
  + 1 ship, 1 bullet, 1 MOVING rectangle
    - Will involve randomish trajectory/momentum/movement
      * Look at the example I have
      * Will help with movement
* Modules Needed
  + PyGame
* Collision detection
* Scoring
* Sprites