Comp 388/488 - Introduction to Game Design and Development

Spring Semester 2017 - Week 10

Dr Nick Hayward

Contents

gamedev.net

- game dev resources - various updates, links, suggestions...
- a long standing example - **gamedev.net**
    - *https://www.gamedev.net/*
- original founded in 1999
    - *great resource for general game development*

Fun gaming music covers

- Gaming music playlist 1
    - *Lindsey Stirling - Various Gaming Music Videos*
- Gaming music playlist 2
    - *Taylor Davis - Video Game Covers*
- covers include:
    - *Dragon Age, Halo, Zelda, Skyrim, Assassin's Creed, Mass Effect, The Witcher...*

Fun gaming inspirational music

- Really Slow Motion - YouTube Channel

Graduate Courses

A few example game design and development courses:

- New York University - Game Center
  - *more design oriented*
- University of Southern California - USC Games
  - *highest ranked school in many game design degree tables...*
  - *four applicable degree programmes - 2 Graduate*
  - *good connections with industry...*
- University of Utah - Entertainment Arts & Engineering
  - *good reputation for hands-on design and development*
  - *a good mix of design and development - cross-tracks...*
  - *close links to industry - e.g. EA*
- New York Film Academy - Game Design and Development School

Lots of options at the following URL,

- Video Game Design Schools

intro

Please consult the extra notes on Pygame Sprites,

- sprites - intro

resources

- notes = sprites-intro.pdf
- code = basicsprites1.py

image import

Please consult the extra notes on Pygame Sprites,

- sprites - set image

resources

- notes = sprites-set-image.pdf
- code = basicsprites2.py

Image - Image Sprite

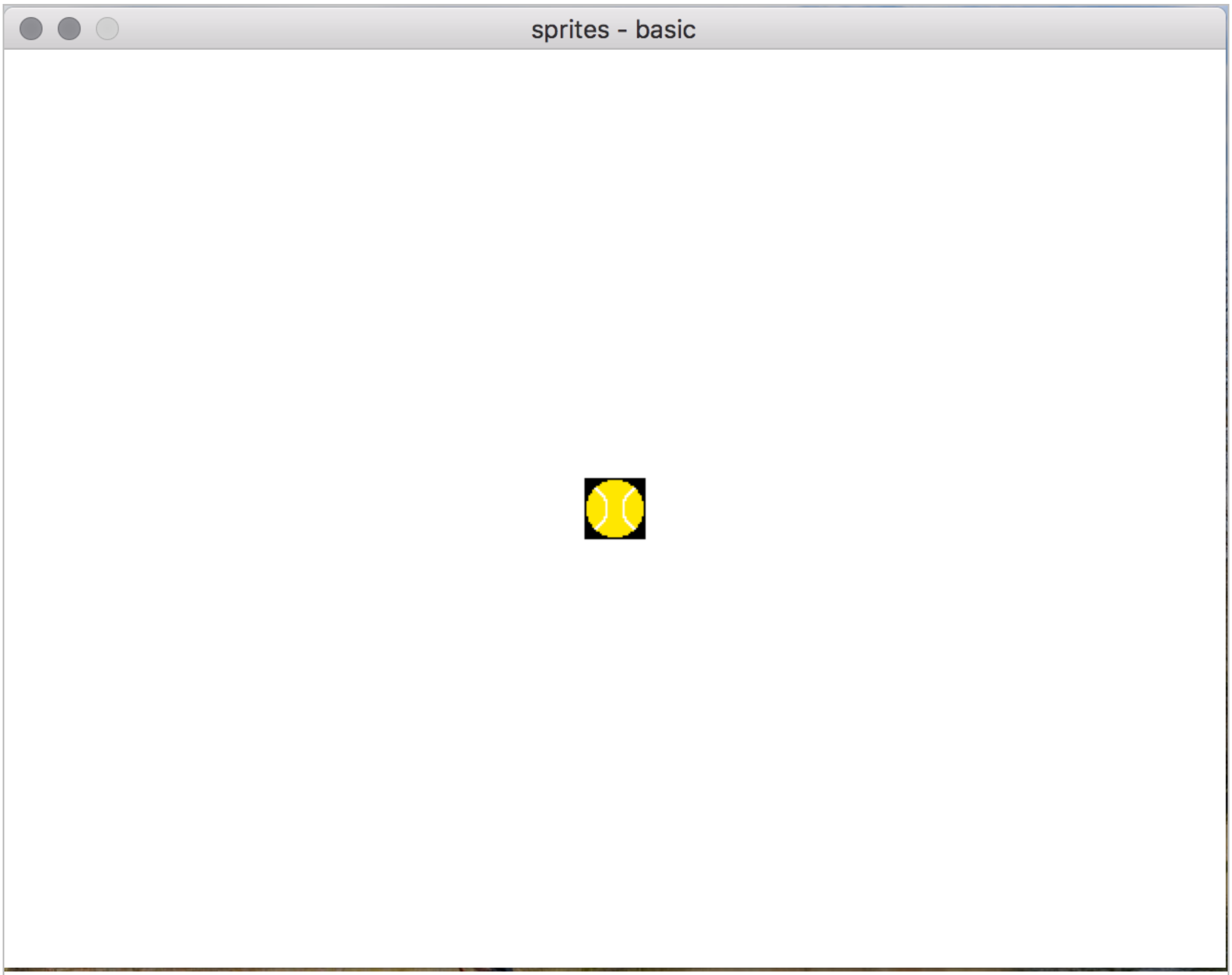


Image sprite

add transparency

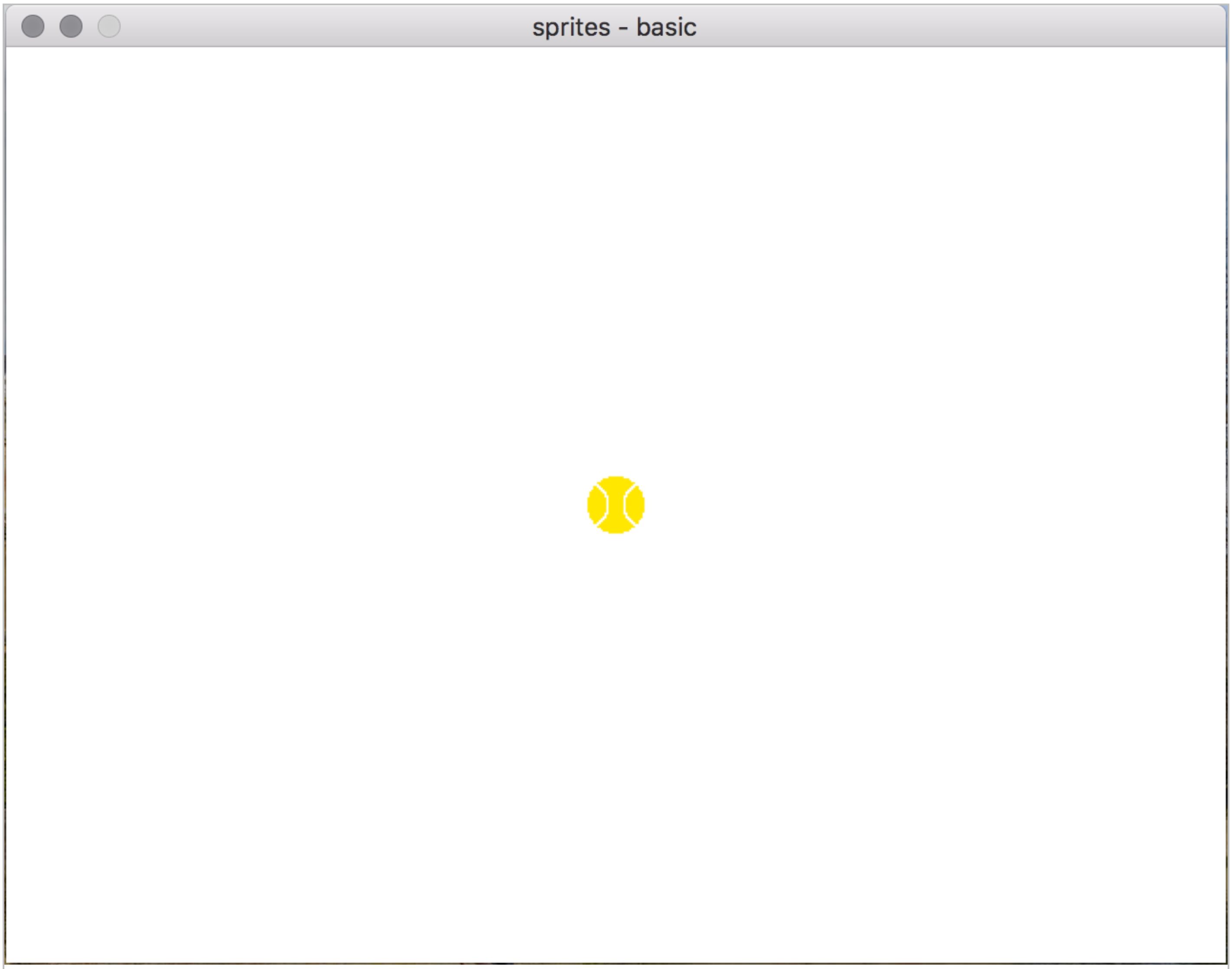
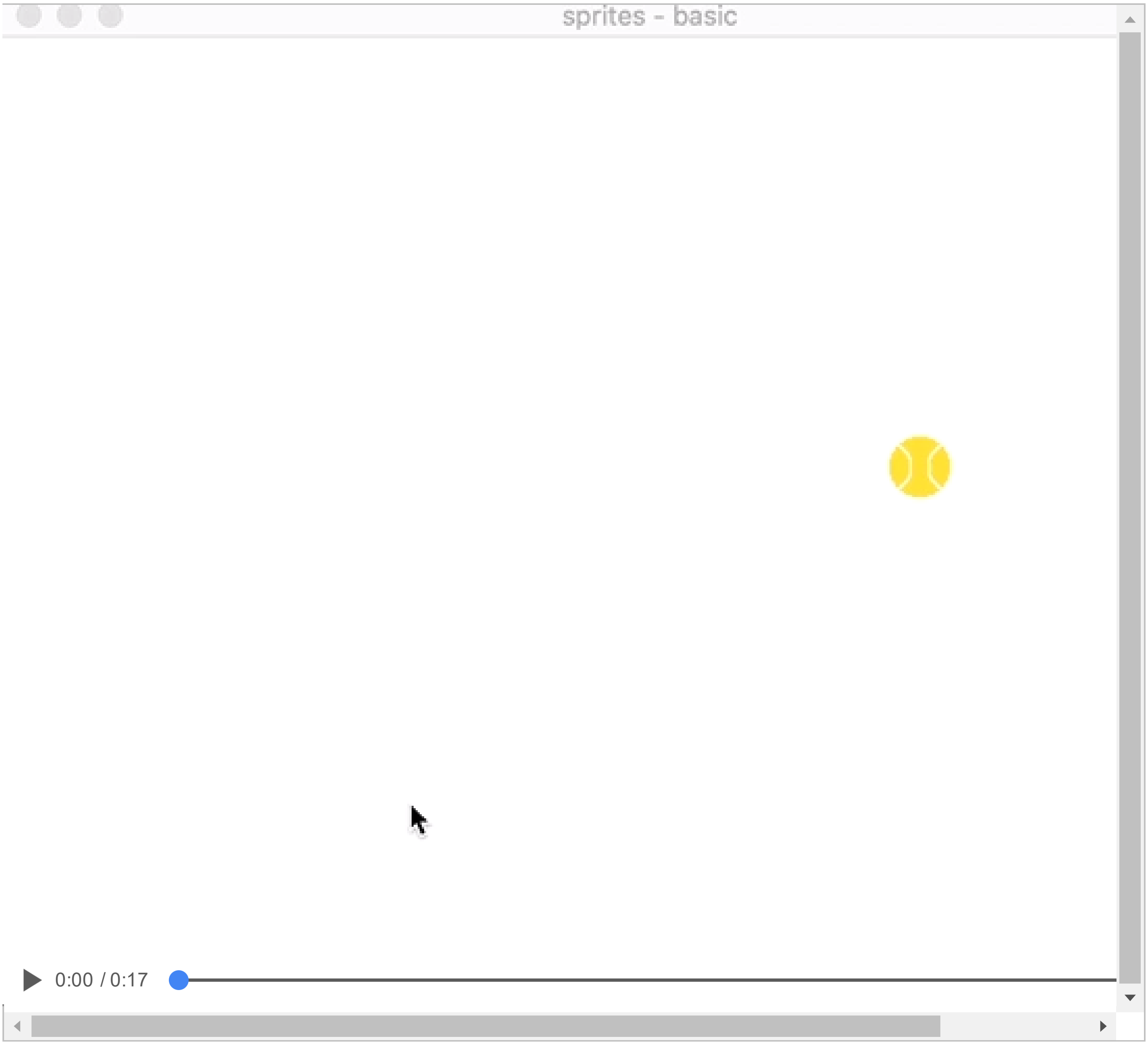


Image sprite - transparent background

bouncing ball



control and move, add events...

Please consult the extra notes on Pygame Sprites,

- sprites - control

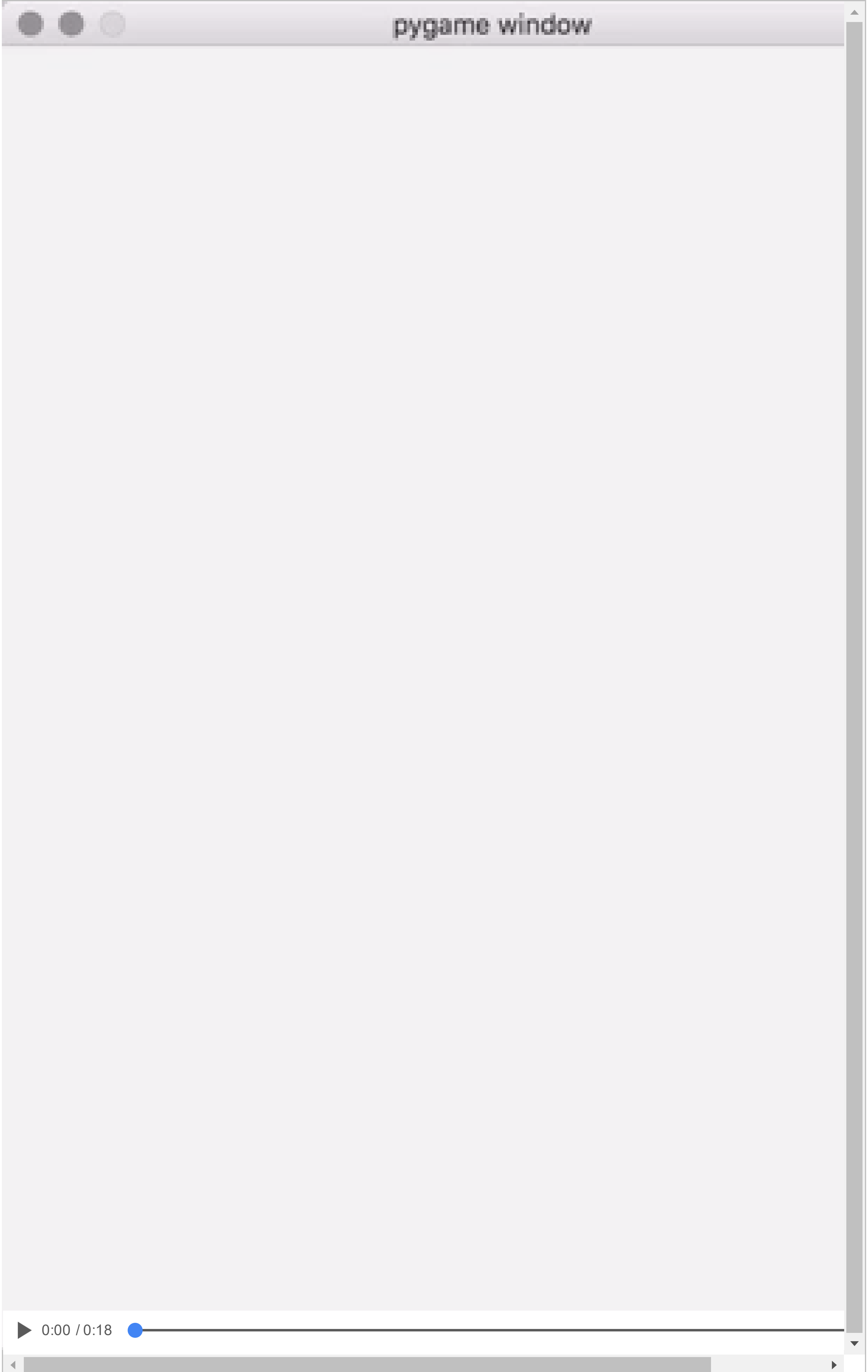
resources

- notes = sprites-control.pdf
- code = basicsprites3.py

game example

- shooter0.1.py - move & control

move & control


pygame window
0:00 / 0:18

shooter style game - STG

- start creating our first full game example
  - *shooter example - **STG** in Japan*
- this game will help us design, develop, and test the following:
  - *user control*
  - *enemy objects*
  - *collision detection*
  - *firing projectiles at enemies*
  - *destroying enemy objects*
  - *add custom sprites and graphics*
  - *improve the collision detection*
  - *start animating sprite images*
  - *radomise enemy objects to create greater challenges*
  - *keep a running game score and render to game window*
  - *add game music and sound effects*
  - *check our player's health...*
  - *add some fun game extras*
    - e.g. health status, explosions...
    - lots more...

add more objects - mob

- now start to add extra sprite objects to our game window
  - *commonly given a collective, generic name of* ***mob***
- add the following class `Mob` to our game

```
# create a generic mob sprite for the game - standard name is *mob*
class Mob(pygame.sprite.Sprite):
    def __init__(self):
        pygame.sprite.Sprite.__init__(self)
        self.image = pygame.Surface((20, 20))
        self.image.fill(CYAN)
        # specify bounding rect for sprite
        self.rect = self.image.get_rect()
        # specify random start posn & speed of enemies
        self.rect.x = random.randrange(winWidth - self.rect.width)
        self.rect.y = random.randrange(-100, -50)
        self.speed_y = random.randrange(1, 10)

    def update(self):
        self.rect.y += self.speed_y
```

- with this class we can create extra sprite objects
  - *set their size, colour, &c.*
  - *then set random x and y coordinates for the starting position of the sprite object*
- use random values to ensure that the objects start and move from different positions
  - *from the top of the game window*
  - *then progress in staggered groups down the window...*

update extra objects

- as our enemy objects move down the game window
  - *need to check if and when they leave the bottom of the game window*
- we can add the following checks to the `update` function

```
def update(self):
    self.rect.y += self.speed_y
    # check if enemy sprite leaves the bottom of the game window - then randomise at the top...
    if self.rect.top > winHeight + 15:
        # specify random start posn & speed of enemies
        self.rect.x = random.randrange(winWidth - self.rect.width)
        self.rect.y = random.randrange(-100, -50)
        self.speed_y = random.randrange(1, 7)
```

- as each sprite object leaves the bottom of the game window
  - *we can check its position*
- then, we may reset the sprite object to the top of the game window
- need to ensure that the same sprite object does not simply loop around
  - *and then reappear at the same position at the top of the game window*
  - *becomes too easy and tedious for our player...*
- instead, we can reset our *mob* object to a random path down the window
  - *should make it slightly harder for our player*
- also ensure that each extra sprite object has a different speed
  - *by simply randomising the speed along the y-axis per sprite object*

show extra objects

- now create a *mob* group as a container for our extra sprite objects
- group will become particularly useful as we add collision detection later in the game
  - *update our code as follows, e.g.*

```
# sprite groups - game, mob...
mob_sprites = pygame.sprite.Group()
# create sprite objects, add to sprite groups...
for i in range(10):
    mob = Mob()
    # add to game_sprites group to get object updated
    game_sprites.add(mob)
    # add to mob_sprites group - use for collision detection &c.
    mob_sprites.add(mob)
```

- create our *mob* objects
  - *then add them to the required sprite groups*
- by adding them to the `game_sprites` group
  - *they will be updated as the game loop is executed*
- `mob_sprites` group will help us easily detect extra sprite objects
  - *e.g. when we need to add collision detection*
  - *or remove them from the game window...*

modify motion of extra objects

- above updates work great for random motion along the y-axis
  - *add some variation to movement of extra sprite object by modifying the x-axis*
- we can modify the x-axis for each extra sprite object
  - *creates variant angular motion along both the x-axis and y-axis, e.g.*

```python
# random speed along the x-axis
self.speed_x = random.randrange(-3, 3)
...

self.rect.x += self.speed_x
# check if sprite leaves the bottom of the game window - then randomise at the top...
if self.rect.top > winHeight + 15 or self.rect.left < -15 or self.rect.right > winWidth + 15:
    # specify random start posn & speed of extra sprite objects
    self.rect.x = random.randrange(winWidth - self.rect.width)
    self.speed_x = random.randrange(-3, 3)
...
```

modify motion of extra objects - continued

- our `mob` class may now be updated as follows,

```
# create a generic extra sprite object for the game - standard name is *mob*
class Mob(pygame.sprite.Sprite):
  def __init__(self):
      pygame.sprite.Sprite.__init__(self)
      self.image = pygame.Surface((20, 20))
      self.image.fill(CYAN)
      # specify bounding rect for sprite
      self.rect = self.image.get_rect()
      # specify random start posn & speed
      self.rect.x = random.randrange(winWidth - self.rect.width)
      self.rect.y = random.randrange(-100, -50)
      # random speed along the x-axis
      self.speed_x = random.randrange(-3, 3)
      # random speed along the y-axis
      self.speed_y = random.randrange(1, 7)

  def update(self):
      self.rect.x += self.speed_x
      self.rect.y += self.speed_y
      # check if sprite leaves the bottom of the game window - then randomise at the top...
      if self.rect.top > winHeight + 15 or self.rect.left < -15 or self.rect.right > winWidth + 15:
          # specify random start posn & speed of extra sprite objects
          self.rect.x = random.randrange(winWidth - self.rect.width)
          self.rect.y = random.randrange(-100, -50)
          self.speed_x = random.randrange(-3, 3)
          self.speed_y = random.randrange(1, 7)
```

- added a quick check for motion of our extra sprite object along the x-axis
  - *as sprite exits on either side of the screen*
  - *create a new sprite on a random path down the screen*

resources

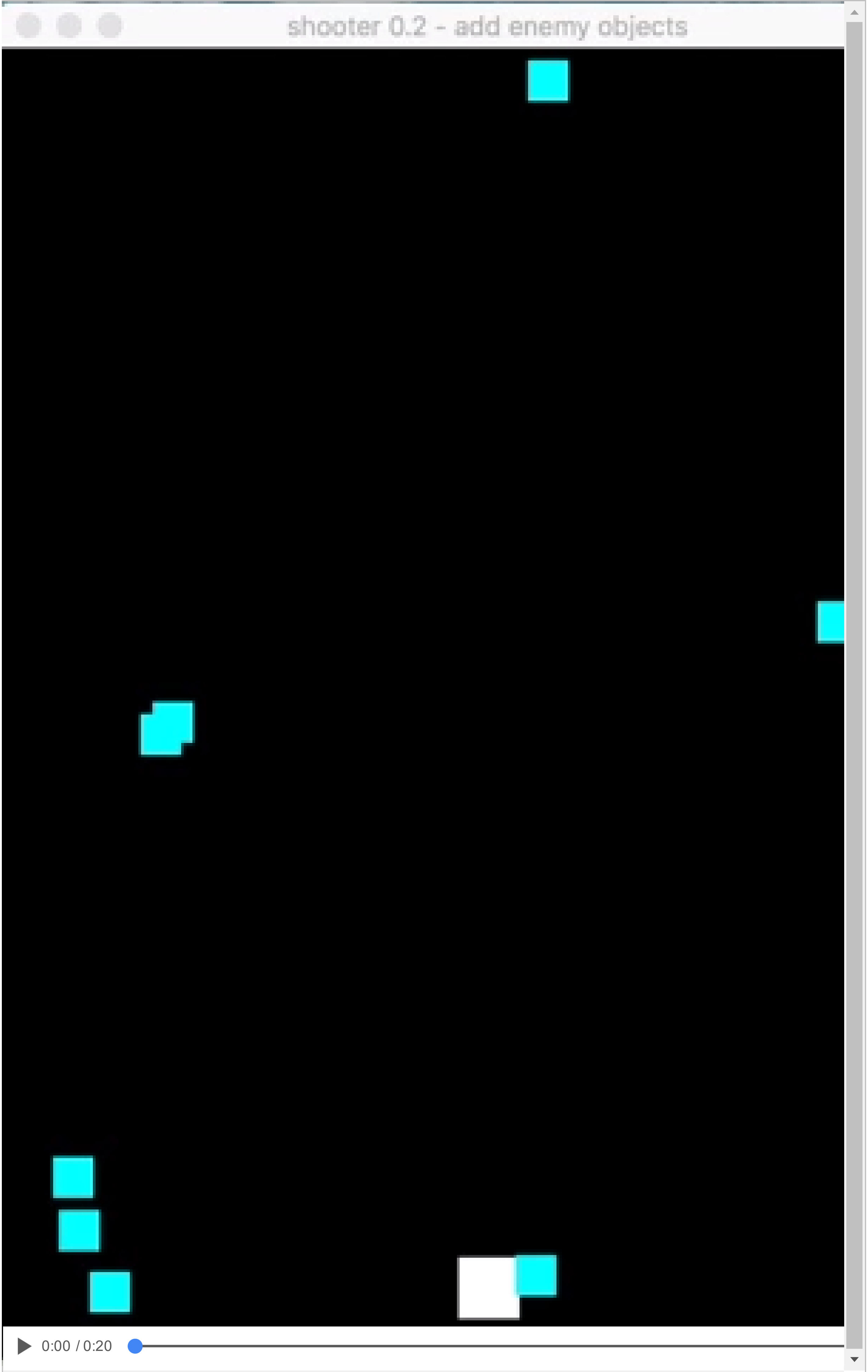
- notes = sprites-more-objects.pdf
- code = basicsprites4.py

game example

- shooter0.2.py

- *add enemy objects*

move & control


shooter 0.2 - add enemy objects
0:00 / 0:20

add new sprites

- create a new class for this sprite object
  - *e.g. projectiles that a player may appear to fire from the top of player object*
  - *such as a ship &c*

```python
# create a generic projectile sprite - for bullets, lasers &c.
class Projectile(pygame.sprite.Sprite):
    # x, y - add specific location for object relative to player sprite
    def __init__(self, x, y):
        pygame.sprite.Sprite.__init__(self)
        self.image = pygame.Surface((5, 10))
        self.image.fill(RED)
        self.rect = self.image.get_rect()
        # weapon fired from front (top) of player sprite...
        self.rect.bottom = y
        self.rect.centerx = x
        # speed of projectile up the screen
        self.speed_y = -10

    def update(self):
        # update y relative to speed of projectile on y-axis
        self.rect.y += self.speed_y
        # remove from game window - if it goes beyond bounding for y-axis at top...
        if self.rect.bottom < 0:
            # kill() removes specified sprite from group...
            self.kill()
```

- creating another sprite object for a projectile such as a bullet or a laser beam
- projectile will be shot from the top of another object
  - *set `x` and `y` coordinates relative to position of player's object*
  - *setting the speed along the y-axis so it travels up the screen*
- as we update each projectile object
  - *update its speed, and then check its position on the screen...*
- if it leaves the top of the game window
  - *we can call the generic `kill()` method on this sprite*
- method is available for any sprite object we create in the game window

listen for keypress

- need to add a new listener to the game loop to detect a keypress for the *spacebar*
- use this keypress to allow a player to shoot these projectiles, e.g. a laser beam

```
# 'processing' inputs (events)
for event in EVENTS.get():
    # check keyboard events - keydown
    if event.type == pygame.KEYDOWN:
        # check for ESCAPE key
        if event.key == pygame.K_ESCAPE:
            gameExit()
        elif event.key == pygame.K_SPACE:
            # fire laser beam...
            player.fire()
```

- updated our keypress listeners to check each time a player hits down on the *spacebar*
- use this keypress event to fire our projectile
  - *e.g. a laser beam to hit our enemy mobs...*

release new sprites

- as player hits the *spacebar*, we need to create new sprites
- new sprite objects will then be released from the top of the player's object
- relative position of one sprite object is determining start position of another sprite object
- need to update the class for our primary sprite object, e.g. a player
  - *include a method for firing the projectiles from the top of this sprite object, e.g.*

```
# fire projectile from top of player sprite object
def fire(self):
    # set position of projectile relative to player's object rect for centerx and top
    projectile = Projectile(self.rect.centerx, self.rect.top)
    # add projectile to game sprites group
    game_sprites.add(projectile)
    # add each projectile to sprite group for all projectiles
    projectiles.add(projectile)
```

- sets start position for `x` and `y` coordinates of each projectile sprite
  - *sets to the current position of the player's sprite object*
- then, add each projectile sprite object to the main game sprite group
  - *and add a new sprite group for all of the projectiles*
  - *add this new sprite group as follows,*

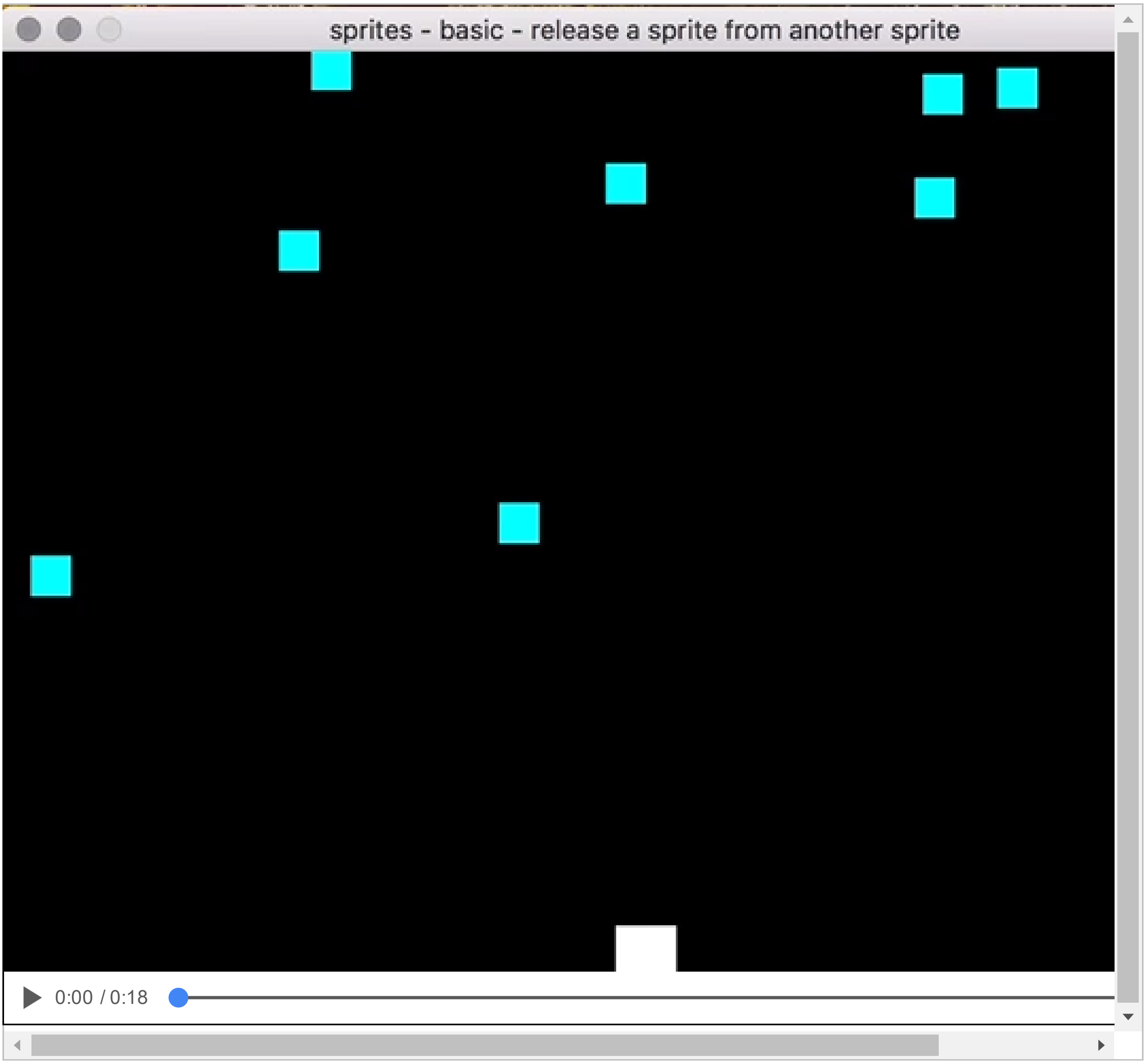
```
projectiles = pygame.sprite.Group()
```

- when a player presses down on the *spacebar* a projectile will be fired
  - *a red laser beam from the top of the player's sprite object*

resources

- notes = sprites-relative-objects.pdf
- code = basicsprites5.py

relative objects



basic collision detection

- Pygame includes support for adding explicit collision detection
  - *between two or more sprites in a game window*
  - *use built-in functions to help us work with these collisions*
- add basic collision detection
  - *each time an object hits the player's object at the foot of the game window*
  - *Pygame includes the following function, e.g.*

```
# add check for collision - extra objects and player sprites (False = hit object is not deleted from game window)
pygame.sprite.spritecollide(player, mob_sprites, False)
```

- sprite object's function allows us to check if one sprite object has been hit by another
- e.g. checking if `player` sprite object hit by another sprite object
  - *in this example, from the `mob_sprites` group*
- `False` parameter is a boolean value for the state of the object that has hit
  - *i.e. determines whether a mob sprite object should be deleted from game window or not*
- particularly useful as it returns a *list* data structure
  - *contains any mob sprite objects that hit the player sprite object*
  - *update this code as follows, and store this list in a variable, e.g.*

```
collisions = pygame.sprite.spritecollide(player, mob_sprites, False)
```

- then use this *list* to check if any collisions have occurred in our game window, e.g.

```
...
if collisions:
  # update game objects &c.
  ...
...
```

- use boolean value to check if the *list* `collisions` is empty or not

Sprite group collision detection

- now add collision detection for various groups of sprites
  - *e.g. one group of sprites may be colliding with another, defined sprite group...*
- use Pygame's collide method for sprite groups, e.g.

```
# add check for sprite group collide with another sprite group - projectiles hitting enemy objects - use True to delete sp
collisions = pygame.sprite.groupcollide(mob_sprites, projectiles, True, True)
```

- boolean parameter values of `True` and `True`
  - *allow us to delete both the hit enemy objects*
  - *and the projectile objects that hit them*
- as *list* of `collisions` is populated
- create new sprite objects for those that have been hit and deleted
- e.g. extra objects that move down the game window

```
# add more mobs for those hit and deleted by projectiles
for collision in collisions:
    mob = Mob()
    game_sprites.add(mob)
    mob_sprites.add(mob)
```

- if we don't create new extra objects
  - *game window will quickly run out of sprite objects*

resources

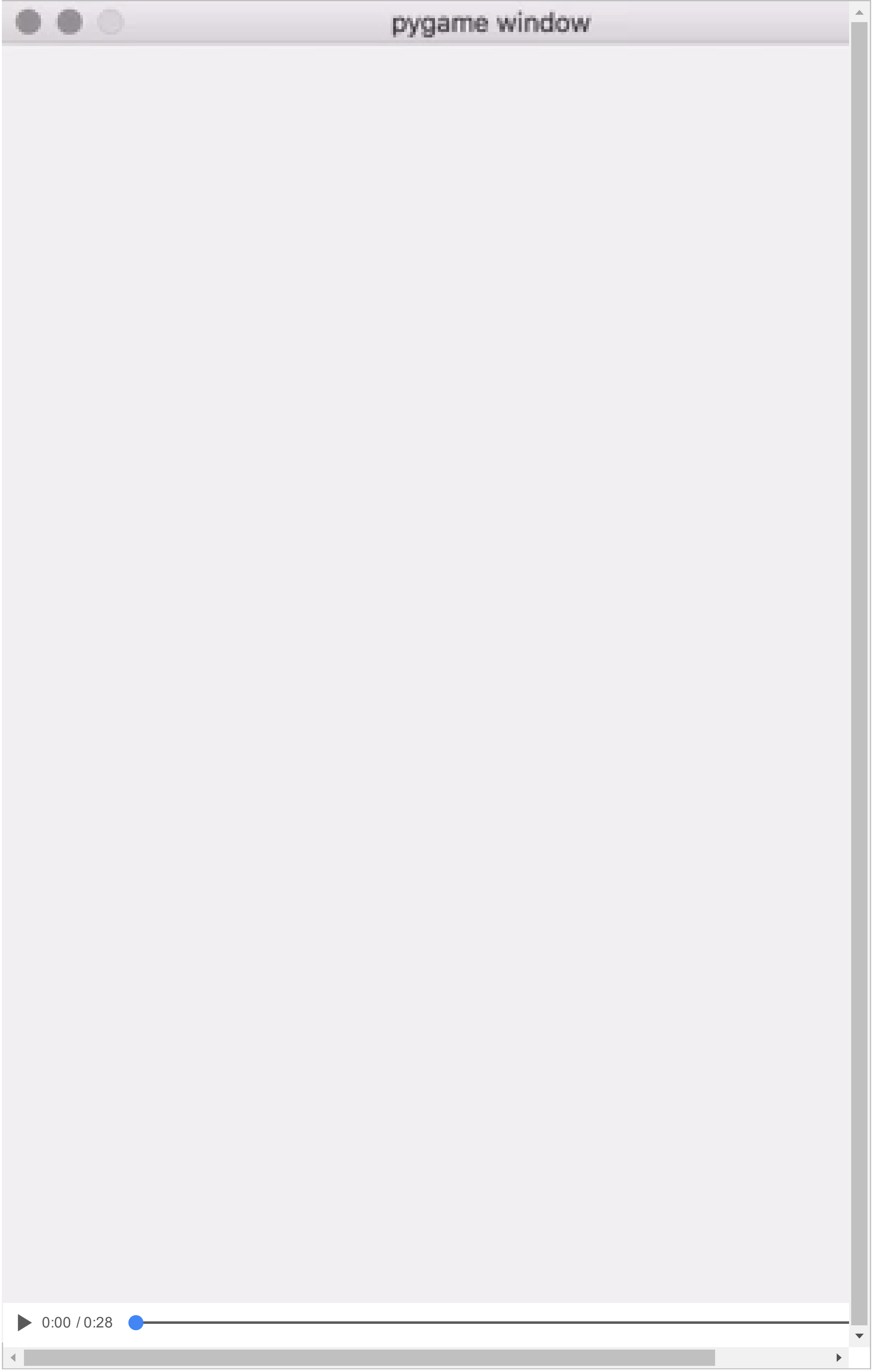
- notes = sprites-collision-detection.pdf
- code = basicsprites6.py

game example

- shooter0.3.py
  - *collision detection of single sprite*
  - *detect group collisions*

Video - Shooter0.3

basic collisions and firing


pygame window
0:00 / 0:28

add graphics to the sprites

- now start to add some custom images for our sprite objects
  - *player object, mobs, projectiles, and a game background...*
- add images and backgrounds to our shooter game to help represent objects
  - *player's ship, laser beams firing, asteroids to hit, and star-filled background*
- before we can add our images for the sprites and backgrounds
  - *need to add some images files to our game's directory structure*
  - *normally create an* `assets` *folder*
  - *add any required images, audio, video &c. for our game...*
- may now update our directory structure to include the required `assets`,

```
|-- shootemup

    |-- assets

        |-- images

            |__ ship.png
```

add images to the game

import game assets

- need to import the Python module for `os`
- allows us to query a local OS's directory structure.

```
# import os
import os
```

- specify the directory location of the main game file
  - *so Python can keep track of the relative location of this file, e.g.*

```
game_dir = os.path.dirname(__file__)
```

- `__file__` is used by Python to abstract the root application file
  - *then portable from system to system*
  - *allows us to set relative paths for game directories, e.g.*

```
# game assets
game_dir = os.path.dirname(__file__)
# relative path to assets dir
assets_dir = os.path.join(game_dir, "assets")
# relative path to image dir
img_dir = os.path.join(assets_dir, "images")
```

- may then import an image for use as a sprite as follows,

```
# assets - images
ship = pygame.image.load(os.path.join(img_dir, "ship.png"))
```

convert and colour key

- as we import an image for use as a sprite within our game
  - *need to use a `convert()` method*
  - *ensures image file is of a type Pygame can use natively*
- if not, there is a potential for the game to perform more slowly
- convert example,

```
ship = pygame.image.load(os.path.join(img_dir, "ship.png")).convert()
```

- for each image that Pygame adds as a sprite
  - *a bounding rectangle will be set with a given colour*
- in most examples, we want to set the background of our sprite to transparent
- rectangle for the image will now blend with the background colour of our game window, e.g.

```
ball.set_colorkey(WHITE)
```

- now check for white coloured pixels in the image background
  - *then set them to transparent*

add game background

- now add a background image for our game
  - *we might recreate stars and space for our game window, e.g.*

```
# load graphics
bg_img = pygame.image.load(os.path.join(img_dir, "bg-purple.png")).convert
```

- also add a rectangle to contain our background image

```
# add rect for bg - helps locate background
bg_rect = bg_img.get_rect()
```

- basically helps us know where to add our background image
  - *then subsequently find it as needed with the logic of our game*
  - *then draw our background image as part of the game loop, e.g.*

```
# draw background image - specify image file and rect to load image
window.blit(bg_img, bg_rect)
```

add game images

- need to add an image for our player's ship, laser beams, and asteroids to shoot, e.g.

```
# add ship image
ship_img = pygame.image.load(os.path.join(img_dir, "ship-blue.png")).convert()
# ship's laser
laser_img = pygame.image.load(os.path.join(img_dir, "laser-blue.png")).convert()
# asteroid
asteroid_img = pygame.image.load(os.path.join(img_dir, "asteroid-med-grey.png")).convert()
```

- to use these new images in our game
  - *need to modify the code for each object, e.g.* `Player` *object*
  - *update our class to include a reference to the* `ship_img`

```
self.image = ship_img
```

- also customise this image by scaling it to better fit our game window, e.g.

```
# load ship image & scale to fit game window...
self.image = pygame.transform.scale(ship_img, (49, 37))
# set colorkey to remove black background for ship's rect
self.image.set_colorkey(BLACK)
```

- also update our ship's rect using a `colorkey`
  - *ensures black* `rect` *is not visible in the game window*

resources

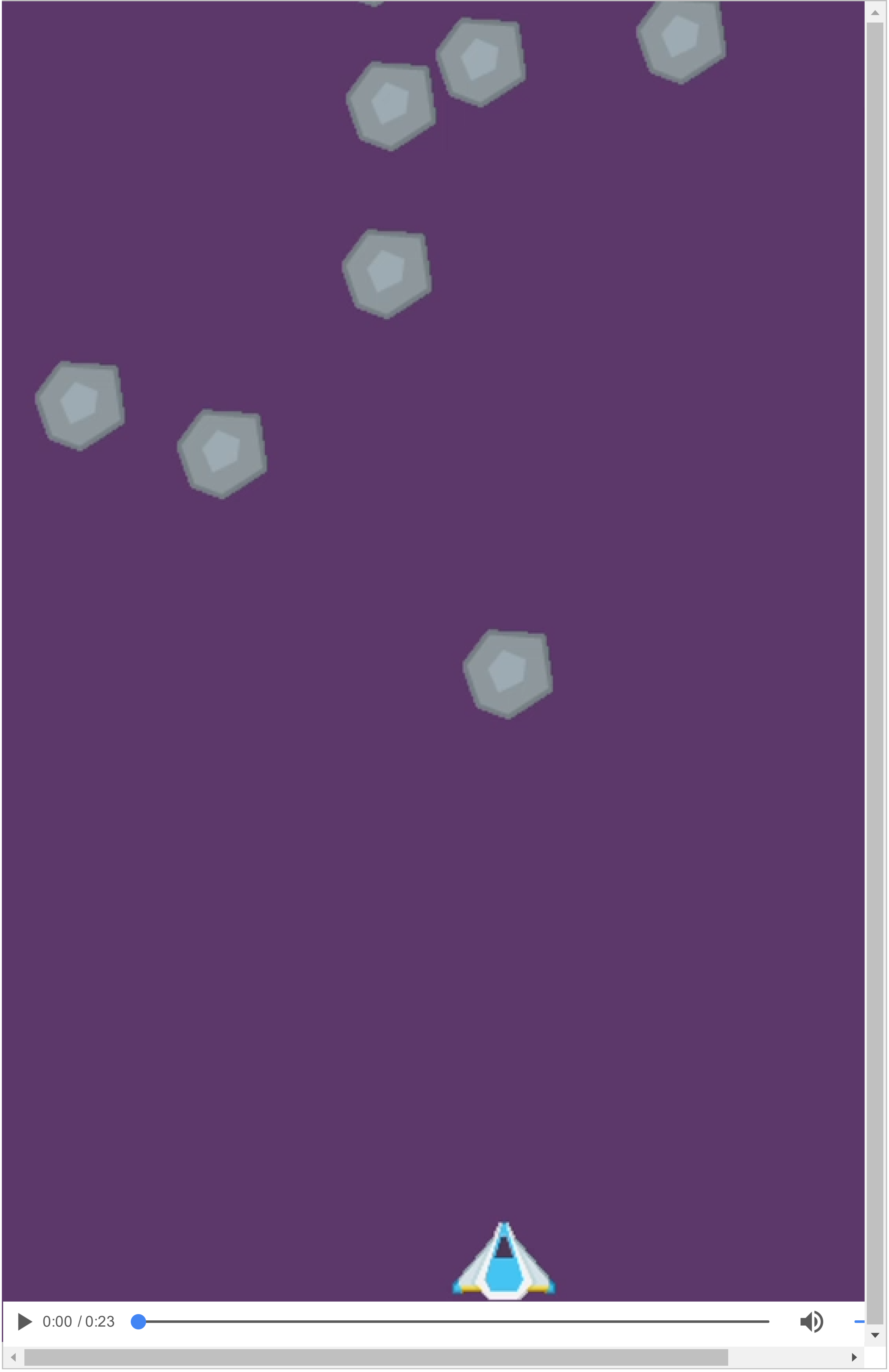
- notes = graphics-and-sprites.pdf
- code = graphicssprites1.py

game example

- shooter0.4.py
  - *add graphics for sprites*
    - images for player's ship, ship's laser, and asteroids &c.
    - set `colorkey` for rect of sprite's
    - set background image for game window...

Video - Shooter0.4

add graphics for sprites


0:00 / 0:23

better collison detection

- collision detection is currently using rectangles to detect one sprite colliding with another
    - *technically referred to as **Axis-aligned Bounding Box** (AABB)*
- for some sprite images this will often cause an unrealistic effect as the two images collide
    - *image does not appear to collide with the other image*
    - *due to space caused by each respective rectangle*
- as one corner of a rectangle hits another corner a collision will be detected, e.g.

```
 ________
|        |
|        |
|       _|_____
|______|_|     |
       |       |
       |_______|
```

- unless each sprites image fits exactly inside the respective bounding box
    - *there will be space left over...*
- a few options for rectifying this issue
    - *might choose to simply calculate and set a slightly smaller rectangle*
    - *or, use a circular bounding box for our sprite image*
- benefit of an *axis-aligned bounding box*
    - *game is able to detect and calculate collisions much faster for a rectangle bounding box*
- a *circular bounding box* may be slower
    - *simply due to the number of calculations the game may need to perform*
    - *checks radius of one bounding box against another bounding box radius*
- rarely becomes a practical issue
    - *unless you're trying to work with thousands of potential sprite images*
- another option, the most precise as well
    - *use pixel perfect collision detection (PPCD)*
- PPCD - game engine will check each pixel of each possible sprite image

- *determines if and when they collided*
- *particularly resource intensive unless you require such precision*

add circle bounding box - part 1

- add some *circle bounding boxes* to our sprite images
  - *for `player` and `mob` objects*
- start by adding explicit circles with a fill colour
  - *helps us check the relative position of the circle's bounding box, e.g.*

```
self.radius = 20
pygame.draw.circle(self.image, RED, self.rect.center, self.radius)
```

- we know sprite image for player's object will have a fixed, known size
  - *we may set the radius to `20`*
- we may add some *circle bounding boxes* to the mob objects as well, e.g.

```
self.radius = int(self.rect.width * 0.9 / 2)
pygame.draw.circle(self.image, RED, self.rect.center, self.radius)
```

add circle bounding box - part 2

- used same basic pattern to add circles
  - *for mob objects we may set each circle's radius relative to the sprite image*
  - *setting radius as 90% of width of sprite image*
  - *then returning half of that value...*
- to use each *circle bounding box*, we need to update the collision checks as well
  - *update this check for each mob object in the update section of the game loop, e.g.*

```
# add check for collision - enemy and player sprites (False = hit object is not deleted from game window)
collisions = pygame.sprite.spritecollide(player, mob_sprites, False, pygame.sprite.collide_circle)
```

- updated the collision check to explicitly look for circle collisions
  - *now remove explicit drawn circle for each circle bounding box*
  - *e.g. for the player and mob object sprites*
  - *we may simply comment out the drawn circle*

```
self.radius = int(self.rect.width * 0.9 / 2)
#pygame.draw.circle(self.image, RED, self.rect.center, self.radius)
```

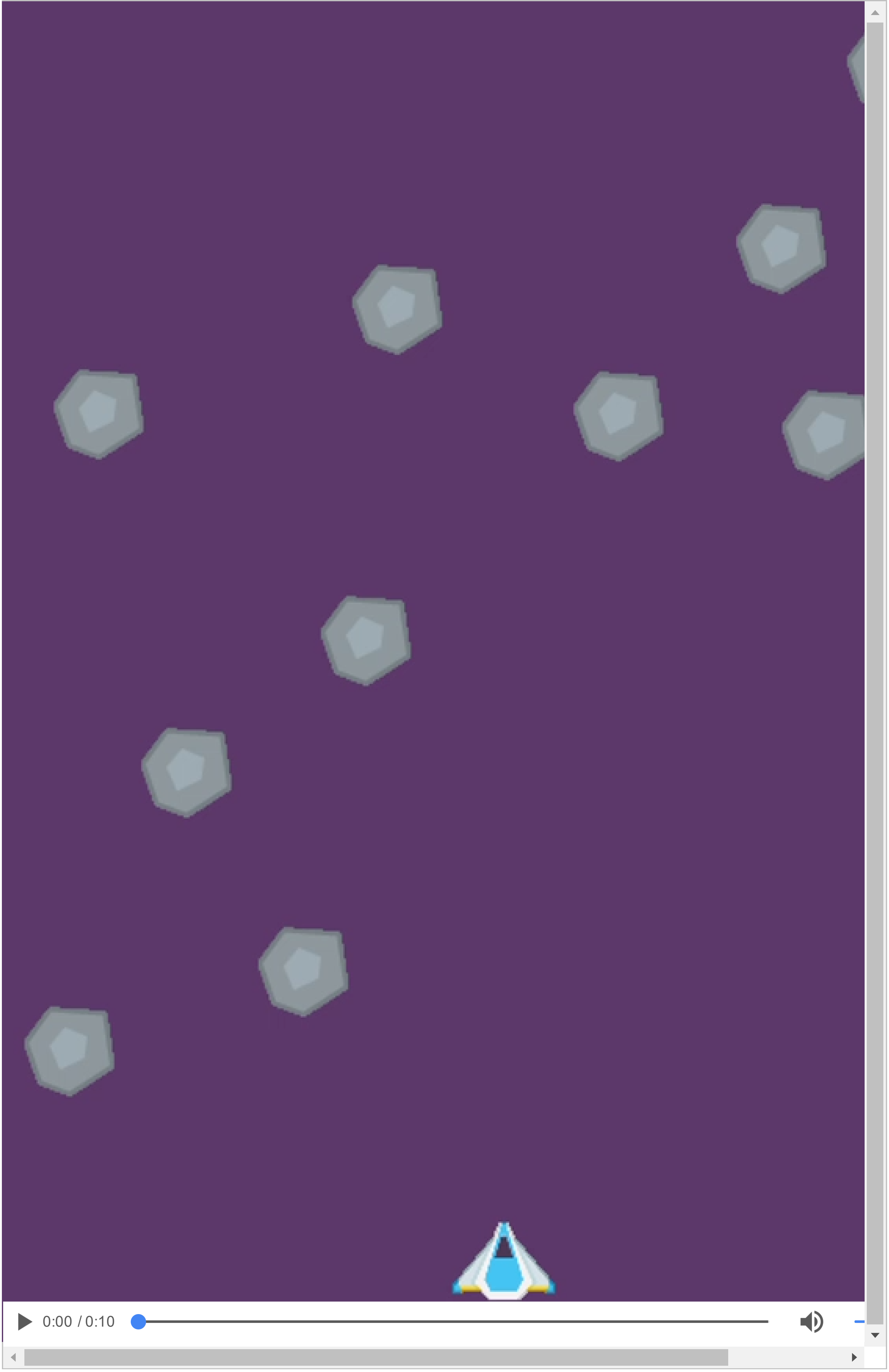
resources

- notes = sprites-collision-detection-better.pdf
- code = collisionsprites3.py

game example

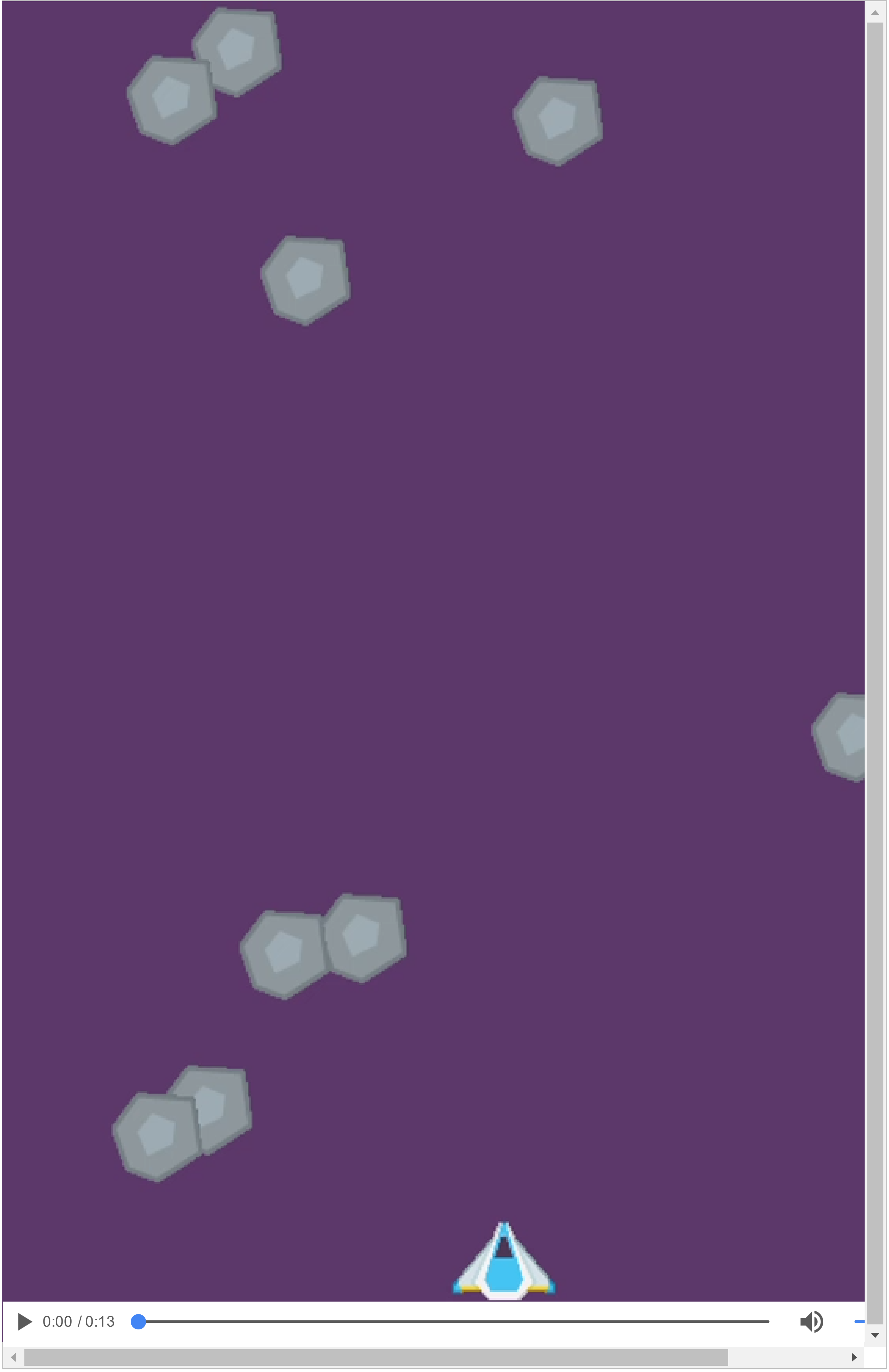
- shooter0.5.py
  - *better collisions and detection*
    - change bounding box for player and mob sprite objects
    - change bounding box to circle, and modify radius to fit sprite objects

better collision detection - example 1



0:00 / 0:10

better collision detection - example 2



0:00 / 0:13

animating sprite images - part 1

- for many game sprites it's fun and useful to add different animations
  - *to animate different states, actions, &c. as the game progresses...*
  - *e.g. random rotation of mob objects, explosions, collisions...*
- already added scale transform to mob objects
  - *we may use the same pattern to add a rotate option*
  - *add animation to these sprites as they move down the game window*
  - *e.g. start by setting some variables for our rotation,*

```
# set up rotation for sprite image - default rotate value, rotate speed to add diff. directions,
self.rotate = 0
self.rotate_speed = random.randrange(-7, 7)
```

- due to the framerate of this game, set to 60FPS
  - *need to ensure rotate animation does not occur for each update of the game loop*
  - *if not, rotation will be too quick, unrealistic, annoying...*

animating sprite images - part 2

- in addition to the rotate animation
  - *also need to consider how to create a timer for this animation*
  - *regularity of update to the animation to ensure it renders realistically*
- already a timer available within our existing code
  - *currently using to monitor the framerate for our game*
- use this timer to check the last time we updated our mob sprite image
- set a time to rotate the sprite image
  - *then check this monitor as it reaches this specified time*
- record last time our sprite image was rotated by getting the time
  - *number of ticks since the game started, e.g.*

```
# check timer for last update to rotate
self.rotate_update = pygame.time.get_ticks()
```

- each time the mob sprite image object is rotated
  - *update value of variable to record the last time for a rotation*
  - *modify the mob sprite's `update` function as follows,*

```
# call rotate update
self.rotate()
```

- simply going to call a separate rotate function
  - *keep the code cleaner and easier to read*
  - *allows us to quickly and easily modify, remove, and simply stop our object's rotation*

rotate

- now add our new `rotate()` function
  - *start by checking if it's time to rotate the sprite image*

```
def rotate(self):
      # check time - get time now and check if ready to rotate sprite image
      time_now = pygame.time.get_ticks()
      # check if ready to update...in milliseconds
      if time_now - self.rotate_update > 70:
          self.last_update = time_now
```

- uses the current time, relative to the game's timer
  - *then checks this value against the last value for a rotate update*
- if difference is greater than 70 milliseconds
  - *it's time to rotate the sprite object*

rotate issues

- for rotation we can't simply add a *rotate transform* to the `rotate()` function
  - *possible in the code, it will also cause the game window to practically freeze*
  - *makes the game unplayable in most examples, e.g.*

```
self.image = pygame.transform.rotate(self.image, self.rotate_speed)
```

- this issue is due to pixel loss for the image
- each rotation of a sprite object image
  - *causes a game's logic to lose part of the pixels for that image*
- this will cause the game loop to start to freeze...

correct rotation

- correct this rotation issue by working with an original, pristine image for the sprite object

```
# set pristine original image for sprite object
self.image_original = mob_img
# set colour key for original image
self.image_original.set_colorkey(BLACK)
```

- then, set the initial sprite object image as a copy of this original

```
# set copy image for sprite rendering
self.image = self.image_original.copy()
```

- then, we may use the pristine original image with the rotation

```
self.image = pygame.transform.rotate(self.image_original, self.rotate_speed)
```

correct rotation speed

- another issue we need to fix is the rotation speed for a sprite object image
- if we simply use our default `self.rotate_speed`
  - *not keeping track of how far we've actually rotated the image*
- need to keep a record of incremental rotation of the image
  - *ensure that it rotates smoothly and in a realistic manner*
- monitor this rotation by using the value of the rotation
  - *adding rotation speed for each update to a sprite object image*
- as the image rotates we can simply check its value as a modulus of 360
  - *to ensure it keeps rotating correctly*

```
self.rotate = (self.rotate + self.rotate_speed) % 360

self.image = pygame.transform.rotate(self.image_original, self.rotate)
```

rect rotation issues

- still have an issue with the rectangle bounding box, which does not rotate
- as sprite image rotates, it loses its centre relative to the bounding rectangle
- to correct this issue, we can now modify our logic for the sprite object's *update*, e.g.

```
# new image for rotation
rotate_image = pygame.transform.rotate(self.image_original, self.rotate)
# check location of original centre of rect
original_centre = self.rect.center
# set image to rotate image
self.image = rotate_image
# create new rect for image
self.rect = self.image.get_rect()
self.rect.center = original_centre
```

- mob sprite object images will now correctly rotate as they move down the screen

resources

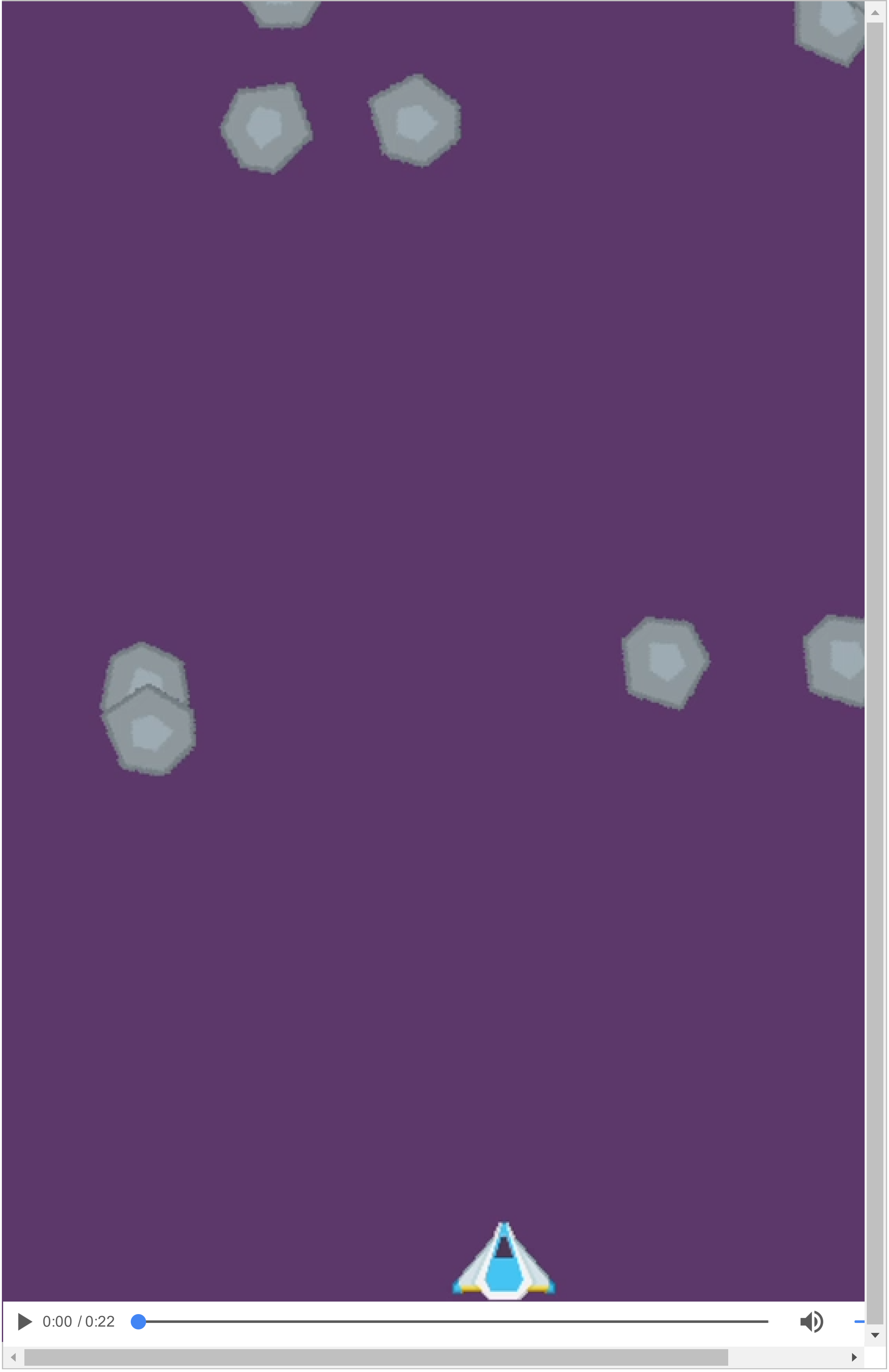
- notes = sprites-animating-images.pdf
- code = animatingsprites1.py

game example

- shooter0.6.py
  - *animating sprite images*
    - rotate mob images down the screen
    - create pristine image for rotation
    - update rect bounding box to ensure it rotates correctly

rotation

animating sprite images


0:00 / 0:22

sprites - basic

- basicsprites1.py
- basicsprites2.py
- basicsprites3.py
- basicsprites4.py
- basicsprites5.py
- basicsprites6.py

graphics and sprites

- graphicssprites1.py

collision detection - basic

- collisionsprites1.py
- collisionsprites2.py

collision detection - better

- collisionsprites3.py

animating sprites

- animatingsprites1.py

Demos - Pygame - Game 1 Example

- shooter0.1.py
- shooter0.2.py
- shooter0.3.py
- shooter0.4.py
- shooter0.5.py
- shooter0.6.py

References - Pygame - Game Notes

- sprites-intro.pdf
- sprites-set-image.pdf
- sprites-control.pdf
- sprites-more-objects.pdf
- sprites-relative-objects.pdf
- sprites-collision-detection.pdf
- graphics-and-sprites.pdf
- sprites-collision-detection-better.pdf
- sprites-animating-images.pdf

References - Various

- GameDev.net
- Video Game Design Schools

Gaming music covers

- Gaming music playlist 1
  - *Lindsey Stirling - Various Gaming Music Videos*
- Gaming music playlist 2
  - *Taylor Davis - Video Game Covers*
- covers include:
  - *Dragon Age, Halo, Zelda, Skyrim, Assassin's Creed, Mass Effect, The Witcher...*

Gaming inspirational music

- Really Slow Motion - YouTube Channel