CSCI 145 · Introduction to Computing · Fall 2026

Animation and Games

Lecture 20 — the game loop, a Ball class, the keyboard, and collisions

Today's videos and activities

1	A pygame program	Activity A: draw a scene
2	A Ball class	Activity B: bouncing balls
3	A Player and the keyboard	Activity C: move with the arrow keys
4	Rocks and collisions	Activity D: a dodge game

- Do each activity before you start the next video
- All four activities are in `day20_game.py`, in day20.zip
- Stuck, or have a question? Post on Campuswire:
  Professor Briggs, Smith, Professor Andrews, or a classmate will help

Video 1 of 4

A pygame program

An **animation** is a series of frames, each a little different

A **game loop** makes one frame each time around

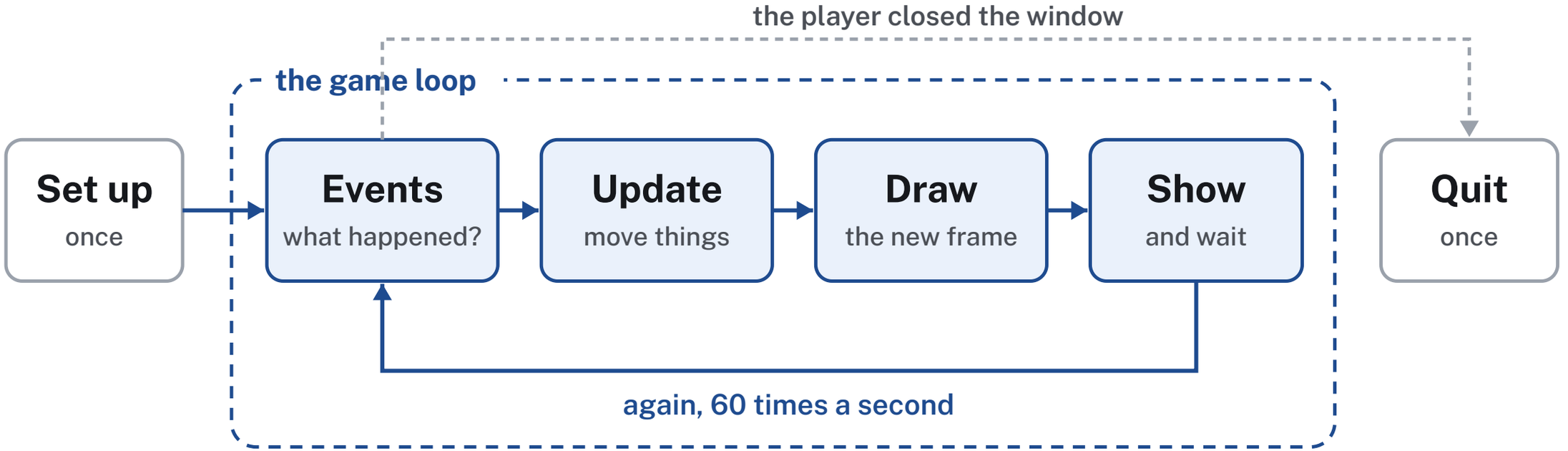
The `pygame` module draws the frames and tells us what the player did

An animation is a series of frames



- Draw the ball, move it a little, draw it again
- 60 frames a second looks like smooth motion
- A game is an animation that also listens to the player

The game loop



- Every game has this shape
- One trip around the loop makes one frame

`bounce.py` : set up, once

```python
import pygame

pygame.init()
screen = pygame.display.set_mode((600, 400))
pygame.display.set_caption("pygame works!")
clock = pygame.time.Clock()

x = 300
y = 200
vx = 4
vy = 3

# the game loop goes here

pygame.quit()
```

- `pygame.init()` is always first
  `pygame.quit()` is always last
- `screen` is a **surface**:
  the canvas we draw on,
  600 wide and 400 tall
- `clock` will set the frame rate
- The ball: where it is,
  and how far it moves each frame

Anatomy of the game loop

```
running = True
while running:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False

    x = x + vx
    y = y + vy
    if x < 30 or x > 570:
        vx = -vx
    if y < 30 or y > 370:
        vy = -vy

    screen.fill((255, 255, 255))
    pygame.draw.circle(screen, (30, 75, 143), (x, y), 30)
    pygame.display.flip()
    clock.tick(60)
```

events | update | draw and show

Events

- `pygame.event.get()` lists what happened since the last frame
- Closing the window is a `QUIT` event
- `running = False` ends the loop

Anatomy of the game loop

```
running = True
while running:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False

    x = x + vx
    y = y + vy
    if x < 30 or x > 570:
        vx = -vx
    if y < 30 or y > 370:
        vy = -vy

    screen.fill((255, 255, 255))
    pygame.draw.circle(screen, (30, 75, 143), (x, y), 30)
    pygame.display.flip()
    clock.tick(60)
```

events | **update** | draw and show

Update

- Move the ball by its velocity
- Past a wall? Reverse the velocity
- 30 is the radius, and 570 is 600 - 30
- Magic numbers! Video 2 names them

Anatomy of the game loop

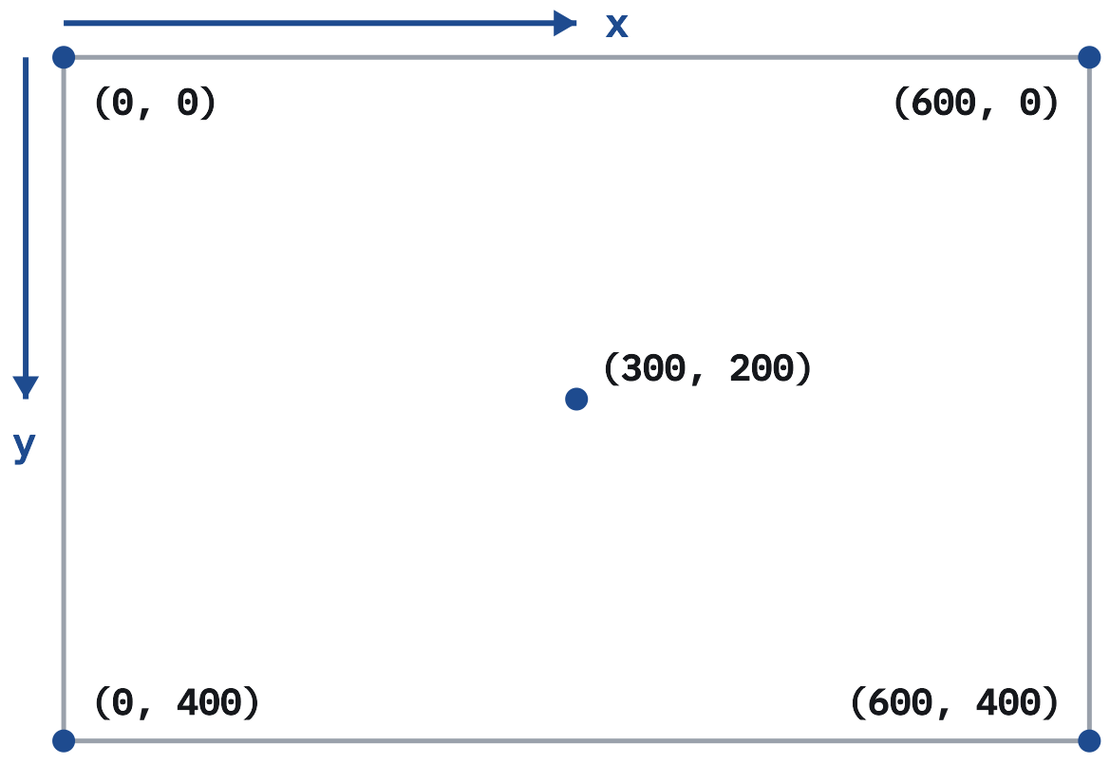
```
running = True
while running:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            running = False

    x = x + vx
    y = y + vy
    if x < 30 or x > 570:
        vx = -vx
    if y < 30 or y > 370:
        vy = -vy

    screen.fill((255, 255, 255))
    pygame.draw.circle(screen, (30, 75, 143), (x, y), 30)
    pygame.display.flip()
    clock.tick(60)
```

events · update · **draw and show**

Draw, then show

- `fill` paints over the last frame
- Draw this frame's shapes
- `flip()` puts the finished frame on the screen
- `tick(60)` waits: at most 60 frames a second

The screen is a grid of pixels



- Just like an image in Lecture 17
- (0, 0) is the upper-left corner
- `x` grows to the right
  `y` grows **down**
- A color is (red, green, blue), each from 0 to 255

Drawing shapes

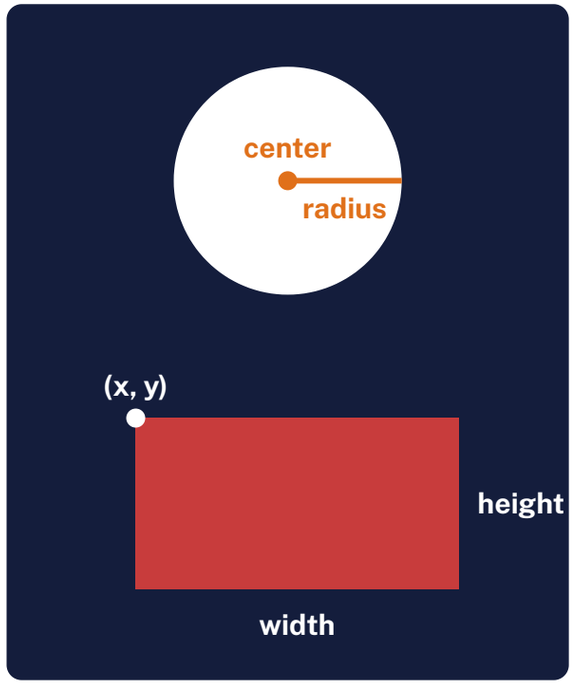
```
pygame.draw.circle( screen, (255, 255, 255), (300, 160), 28)
```

where · color · center · radius

```
pygame.draw.rect( screen, (200, 60, 60), (280, 105, 40, 32))
```

where · color · upper-left x and y, width, height

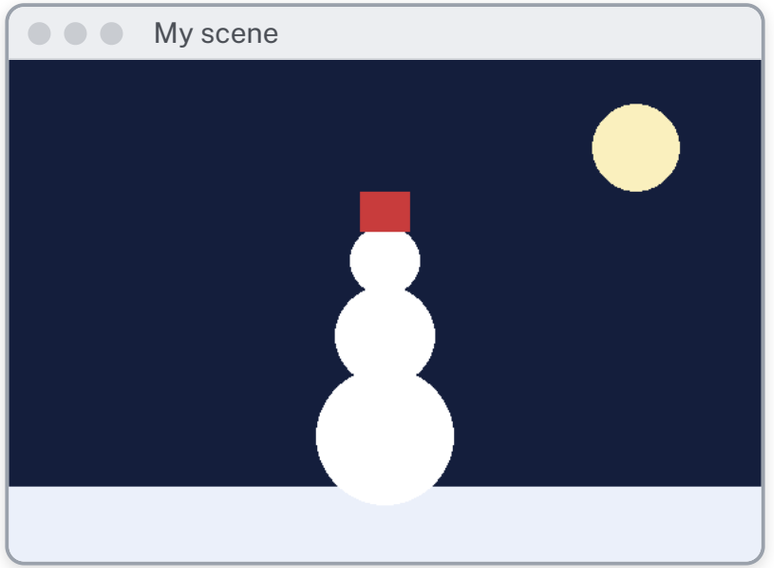
- A circle is placed by its center
- A rectangle is placed by its upper-left corner
- Later shapes are drawn on top of earlier ones



Activity A End of Video 1

1. Download day20.zip and extract it in your `class` folder
2. Open `day20_game.py` and run it: you should see an empty night sky
3. In `show_scene`, at `# Activity A`, draw a scene with at least 4 shapes

`No module named 'pygame'`? Install `midd-media` (see Lab 5)



One idea: a snowman

CHECK YOURSELF, ON PAPER

1. Which point is lower on the screen: (100, 50) or (50, 100)?
2. What does this draw?

```
pygame.draw.circle(screen, (255, 0, 0), (0, 0), 50)
```

Video 2 of 4

A Ball class

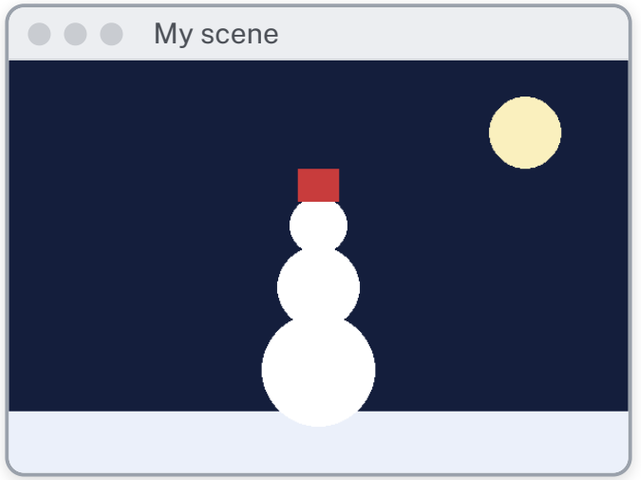
`bounce.py` keeps one ball in four loose variables

A **Ball** object bundles them with the methods that use them

Then twenty balls are just a list and a loop

Activity A — one answer

```
pygame.draw.circle(screen, (250, 240, 190), (500, 70), 35)
pygame.draw.rect(screen, (235, 240, 250), (0, 340, 600, 60))
pygame.draw.circle(screen, (255, 255, 255), (300, 300), 55)
pygame.draw.circle(screen, (255, 255, 255), (300, 220), 40)
pygame.draw.circle(screen, (255, 255, 255), (300, 160), 28)
pygame.draw.rect(screen, (200, 60, 60), (280, 105, 40, 32))
```



CHECK YOURSELF

1. (50, 100) is lower: `y` grows down
2. A quarter of a red circle in the upper-left corner:
   the rest is off the screen

From loose variables to an object

bounce.py

```
x = 300
y = 200
vx = 4
vy = 3
```

- A second ball?
  Four more variables
- Lecture 19: bundle
  the data with
  its methods

`Ball` A ball that bounces off the edges of the screen

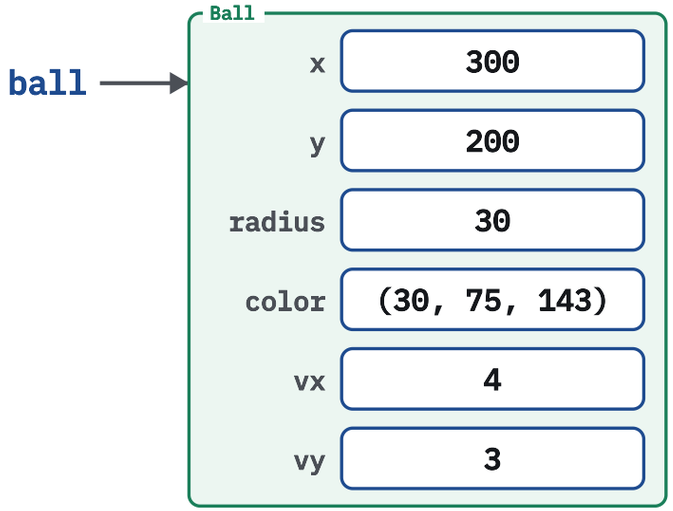
Attribute	Example value	
`x, y`	300, 200	the center
`radius`	30	
`color`	(30, 75, 143)	
`vx, vy`	4, 3	the velocity: pixels per frame

Method call	What it does
`ball.draw(screen)`	draws the ball as a circle
`ball.update()`	moves the ball, and bounces it off the walls

Making a `Ball`

```
class Ball:
    def __init__(self, x, y, radius, color):
        self.x = x
        self.y = y
        self.radius = radius
        self.color = color
        self.vx = random.randint(BALL_MIN_SPEED, BALL_MAX_SPEED)
        self.vy = random.randint(BALL_MIN_SPEED, BALL_MAX_SPEED)
```

```
>>> ball = Ball(300, 200, 30, (30, 75, 143))
```



- Four attributes come from the parameters
- Each ball picks its own random velocity

A ball draws itself

```
    def draw(self, surface):
        pygame.draw.circle(surface, self.color, (self.x, self.y), self.radius)
```

In the loop, before

```
pygame.draw.circle(screen, (30, 75, 143), (x, y), 30)
```

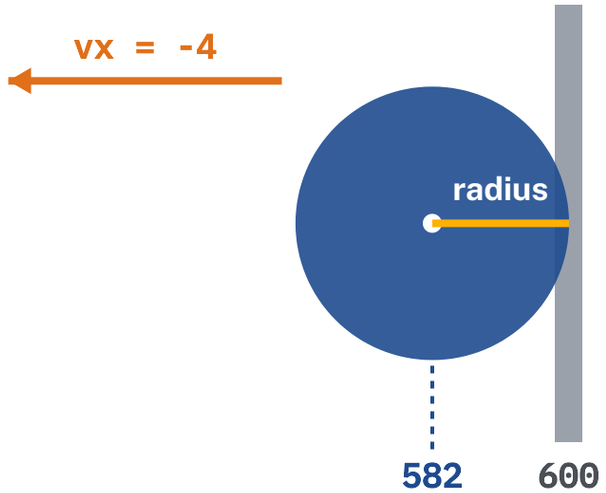
After

```
ball.draw(screen)
```

- `surface` is where to draw: we pass it `screen`
- The loop doesn't need to know what a ball looks like
- Next: `ball.update()` moves it and bounces it

`update`: move, then check the walls

STEP	THE BALL: RADIUS 20
Before	**x** is 578, **vx** is 4
Move: **`x = x + vx`**	**x** is 582
Right edge: **`x + radius`**	602: past the wall
Reverse: **`vx = -vx`**	**vx** is -4
Next frame	**x** is 578, heading left



- Left wall: the left edge is `x - radius`, and the wall is at 0
- Top and bottom: the same, with `y`, `vy`, and `SCREEN_HEIGHT`

Twenty balls: a list of objects

```
balls = []
for i in range(NUM_BALLS):
    radius = random.randint(BALL_MIN_RADIUS, BALL_MAX_RADIUS)
    x = random.randint(radius, SCREEN_WIDTH - radius)
    y = random.randint(radius, SCREEN_HEIGHT - radius)
    color = (random.randint(0, 255), random.randint(0, 255), random.randint(0, 255))
    balls.append(Ball(x, y, radius, color))
```

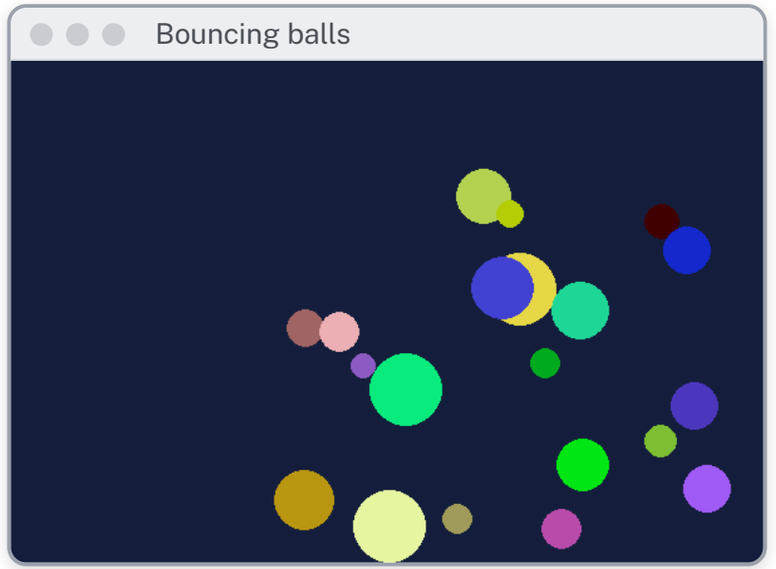
- Written for you, in `show_balls`
- Constants in capitals, at the top of the file: no magic numbers
- `x` starts at least `radius` from each wall: the whole ball is on the screen
- In the game loop: update every ball, draw every ball

Activity B End of Video 2

1. Write `Ball.update`
   Its docstring says exactly what to do
2. In `show_balls`, at the two `# Activity B` comments: update each ball, draw each ball
3. Change the last line of the file to `show_balls()` and run it

CHECK YOURSELF, ON PAPER

A ball has `x = 25`, `radius = 20`, and `vx = -3`
What are `x` and `vx` after one `update()`? After two?



Video 3 of 4

A Player and the keyboard

> The player is a square that moves while an arrow key is held down
>
> A **pygame.Rect** stores its position and size
>
> `min` and `max` keep it on the screen

Activity B — answers

```
    def update(self):
        self.x = self.x + self.vx
        self.y = self.y + self.vy
        if self.x - self.radius < 0 or self.x + self.radius > SCREEN_WIDTH:
            self.vx = -self.vx
        if self.y - self.radius < 0 or self.y + self.radius > SCREEN_HEIGHT:
            self.vy = -self.vy
```

```
for ball in balls:
    ball.update()

screen.fill(BG_COLOR)
for ball in balls:
    ball.draw(screen)
```

CHECK YOURSELF

After one: `x` is 22, `vx` is -3

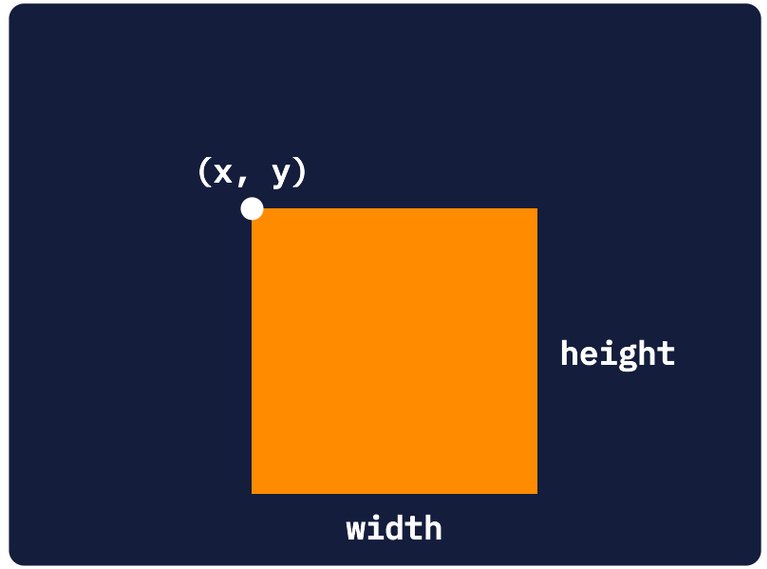
The left edge is at 2: still on the screen

After two: `x` is 19, `vx` is 3

The left edge is at -1: past the wall, so `vx` reverses

`pygame.Rect`: a rectangle object

```
>>> r = pygame.Rect(280, 350, 40, 40)
>>> r.x
280
>>> r.x = r.x + 5
>>> r
Rect(285, 350, 40, 40)
```



- Lab 5: you wrote a `Rect` class pygame has one too
- Attributes `x`, `y`, `width`, `height`
- Change `x` or `y` to move it

A `Player` class

```
class Player:
    def __init__(self, x, y):
        self.rect = pygame.Rect(x, y, PLAYER_SIZE, PLAYER_SIZE)

    def draw(self, surface):
        pygame.draw.rect(surface, PLAYER_COLOR, self.rect)
```

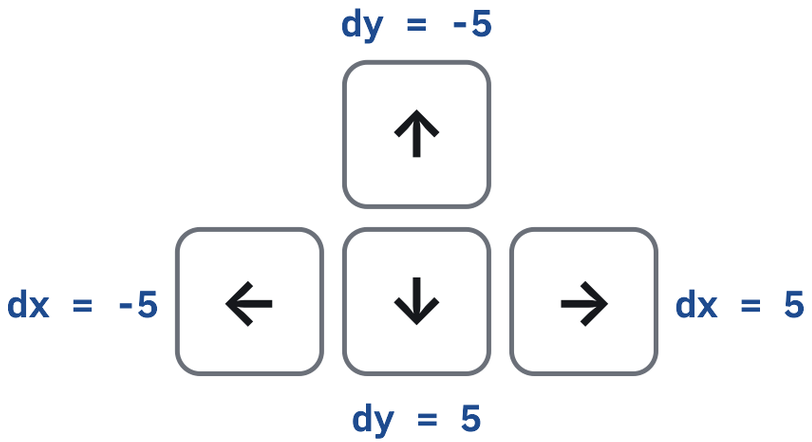
Attribute	
`rect`	a **`pygame.Rect`**: where the player is, and its size

Method call	What it does
`player.move(dx, dy)`	moves it, but keeps it on the screen
`player.draw(screen)`	draws it as a square

- One attribute:
  an object inside an object
- `player.rect.x` is
  the player's left edge

Which keys are down right now?

```
keys = pygame.key.get_pressed()
dx = 0
dy = 0
if keys[pygame.K_LEFT]:
    dx = -PLAYER_SPEED
if keys[pygame.K_RIGHT]:
    dx = PLAYER_SPEED
if keys[pygame.K_UP]:
    dy = -PLAYER_SPEED
if keys[pygame.K_DOWN]:
    dy = PLAYER_SPEED
player.move(dx, dy)
```



- Asked once every frame, in the game loop
- Up is negative: `y` grows down
- `dx` and `dy`: how far to move this frame

Two ways to read the keyboard

AN EVENT: ONCE PER PRESS

```
for event in pygame.event.get():
    if event.type == pygame.KEYDOWN:
        if event.key == pygame.K_q:
            running = False
```

- Happens once, when the key goes down
- Good for quit, restart, jump
- Notes 16 moves its player this way: one step per press

A HELD KEY: EVERY FRAME

```
keys = pygame.key.get_pressed()
if keys[pygame.K_LEFT]:
    dx = -PLAYER_SPEED
```

- True on every frame while the key is down
- Good for smooth movement
- What we use today

Staying on the screen

rect.x can be anything from 0 to SCREEN_WIDTH - PLAYER_SIZE



- The player is 40 wide,
  so its left edge can't be past 560
- `min` stops it at the right wall
- `max` stops it at the left wall
- Both: `max(0, min(new_x, 560))`

```
>>> min(563, 560)
560
>>> max(0, -3)
0
>>> max(0, min(300, 560))
300
```

Activity C End of Video 3

1. In `Player.move`, change `self.rect.x` and `self.rect.y`, keeping the player on the screen
2. Change the last line of the file to `move_player()` and run it
3. Try to leave the screen on all four sides



CHECK YOURSELF, ON PAPER

A player has `rect.x = 2` and `rect.y = 357`
What are they after `move(-5, 5)`?

Video 4 of 4

Rocks and collisions

Rocks fall from above the screen, again and again

Two objects **collide** when their rectangles overlap

One hit ends the game

Activity C — answer

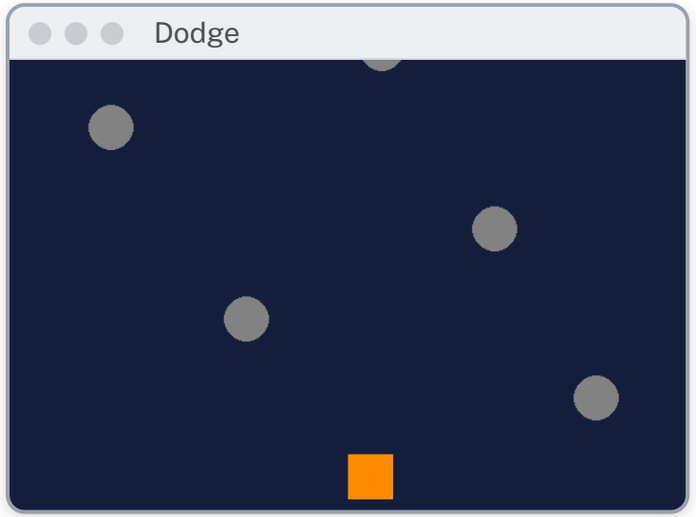
```
    def move(self, dx, dy):
        self.rect.x = max(0, min(self.rect.x + dx, SCREEN_WIDTH - PLAYER_SIZE))
        self.rect.y = max(0, min(self.rect.y + dy, SCREEN_HEIGHT - PLAYER_SIZE))
```

CHECK YOURSELF

`rect.x` is 0: `max(0, min(-3, 560))`

`rect.y` is 360: `max(0, min(362, 360))`

The dodge game



Move with the arrow keys

Don't get hit by a rock

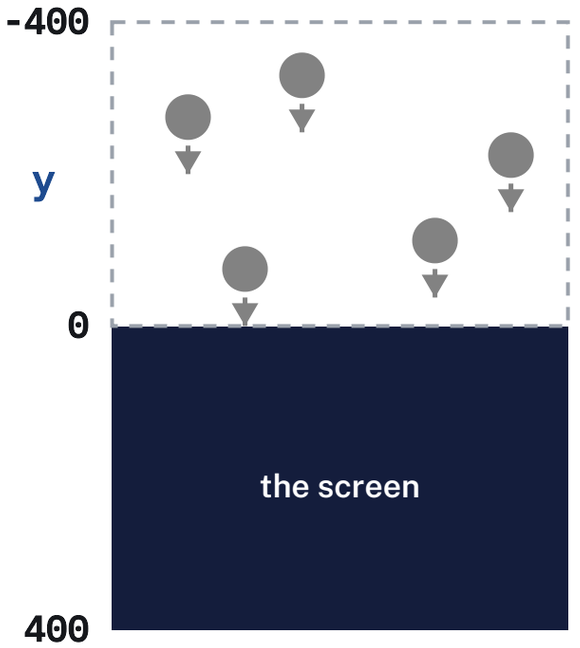
Rock A falling rock

Attribute	
`x, y`	the center
`radius`	always 20
`speed`	pixels per frame, 2 to 5

Method call	What it does or returns
`rock.reset()`	new random spot above the top, new speed
`rock.update()`	falls; resets once it is off the bottom
`rock.get_rect()`	returns the square around the rock
`rock.draw(screen)`	draws it as a gray circle

Rocks start above the top

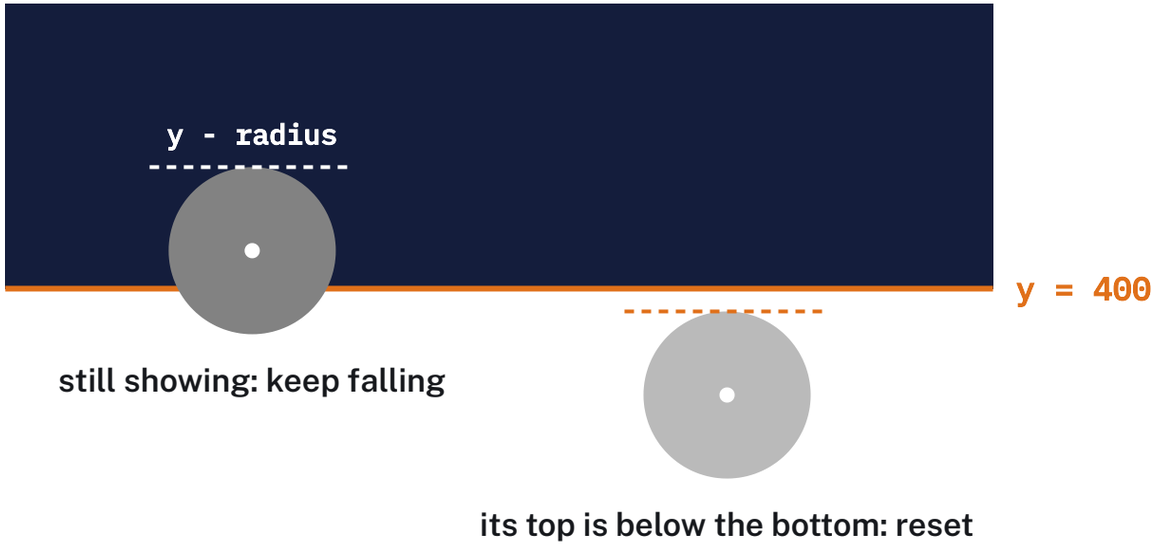
```
    def __init__(self):
        self.radius = ROCK_RADIUS
        self.reset()

    def reset(self):
        self.x = random.randint(self.radius, SCREEN_WIDTH - self.radius)
        self.y = random.randint(-SCREEN_HEIGHT, -self.radius)
        self.speed = random.randint(ROCK_MIN_SPEED, ROCK_MAX_SPEED)
```



- A negative `y` is above the screen
- Different heights: the rocks arrive at different times
- `__init__` calls `reset`:
  the random choices are written once

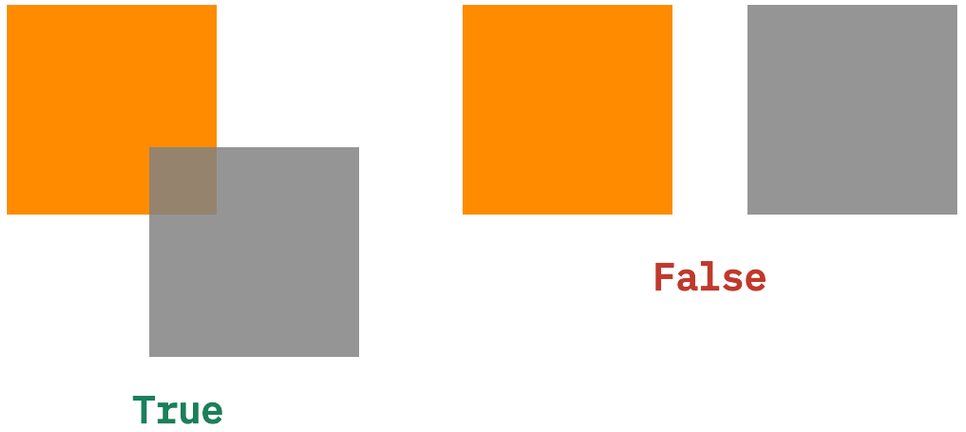
`update`: fall, then start over

- Move down by `speed`
- The rock's top edge is `y - radius`
- Top edge greater than `SCREEN_HEIGHT`?
  The whole rock is off the bottom
- Then call `self.reset()`



Do two rectangles overlap?

```
>>> a = pygame.Rect(0, 0, 40, 40)
>>> b = pygame.Rect(30, 30, 40, 40)
>>> a.colliderect(b)
True
>>> c = pygame.Rect(100, 0, 40, 40)
>>> a.colliderect(c)
False
```

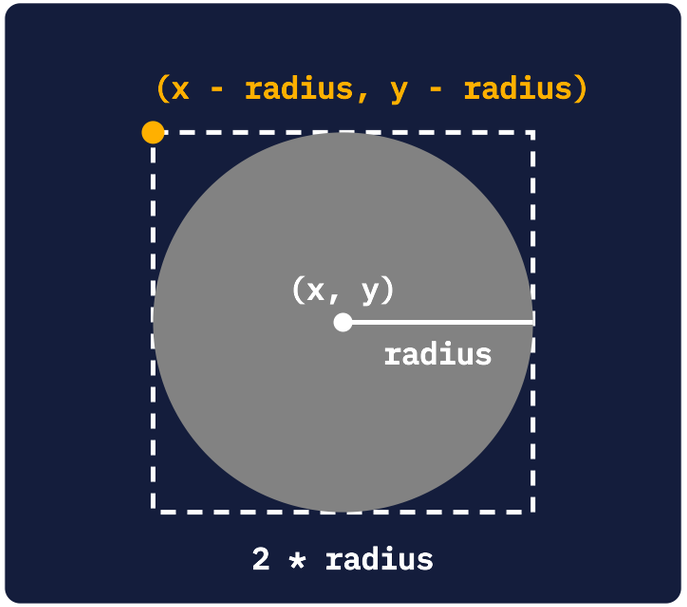


- `colliderect` is a method of `pygame.Rect`
- Lab 5: your `Rect.intersection` asked the same question

A rock's rectangle

- A rock stores a center and a radius, not a `Rect`
- `rock.get_rect()` returns the square around it: a new `pygame.Rect` each time it is called
- The player already has one: `player.rect`

```
if player.rect.colliderect(rock.get_rect()):
    game_over = True
```



One hit ends the game

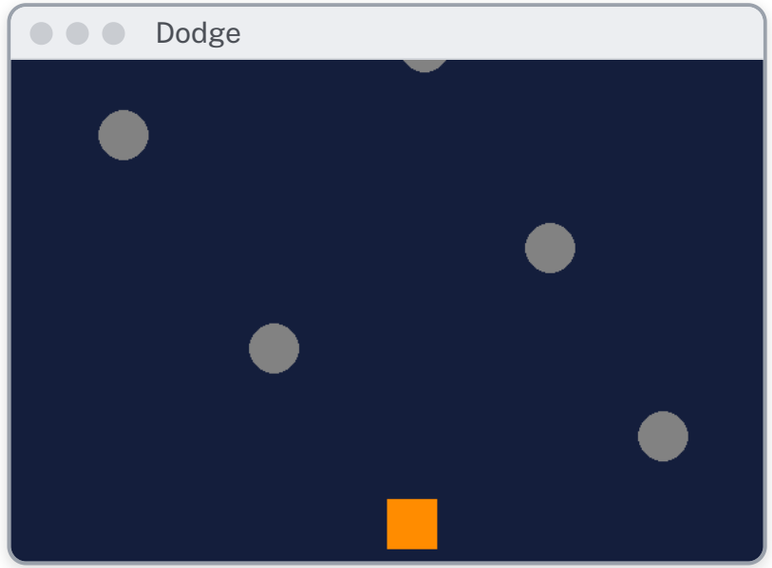
```
game_over = False
running = True
while running:
    # handle events

    if not game_over:
        # move the player
        # update each rock; a hit sets game_over to True

    screen.fill(BG_COLOR)
    # draw each rock, then the player
    # if game_over, draw "Game over"
    pygame.display.flip()
    clock.tick(FPS)
```

- `game_over` is a flag: a variable that is `True` or `False`
- After a hit, nothing moves
- The loop keeps drawing, so the window stays open until you close it
- The "Game over" text is written for you

Activity D

End of Video 4

1. Write `Rock.update` and `Rock.get_rect`
2. In `play`, at the two `# Activity D` comments: update each rock and check for a hit, then draw each rock
3. Change the last line of the file to `play()` and run it



CHECK YOURSELF, ON PAPER

A rock has `x = 300`, `y = 415`, `radius = 20`, and `speed = 4`

1. What does `get_rect()` return?
2. After one `update()`, does it reset?

Activity D — answers

```
    def update(self):
        self.y = self.y + self.speed
        if self.y - self.radius > SCREEN_HEIGHT:
            self.reset()

    def get_rect(self):
        return pygame.Rect(self.x - self.radius, self.y - self.radius,
                           2 * self.radius, 2 * self.radius)
```

```
for rock in rocks:
    rock.update()
    if player.rect.colliderect(rock.get_rect()):
        game_over = True
```

```
for rock in rocks:
    rock.draw(screen)
```

CHECK YOURSELF

1. `Rect(280, 395, 40, 40)`
2. No: `y` is 419,
so its top edge is at 399:
still on the screen
It resets on the next update

Reading Notes 16

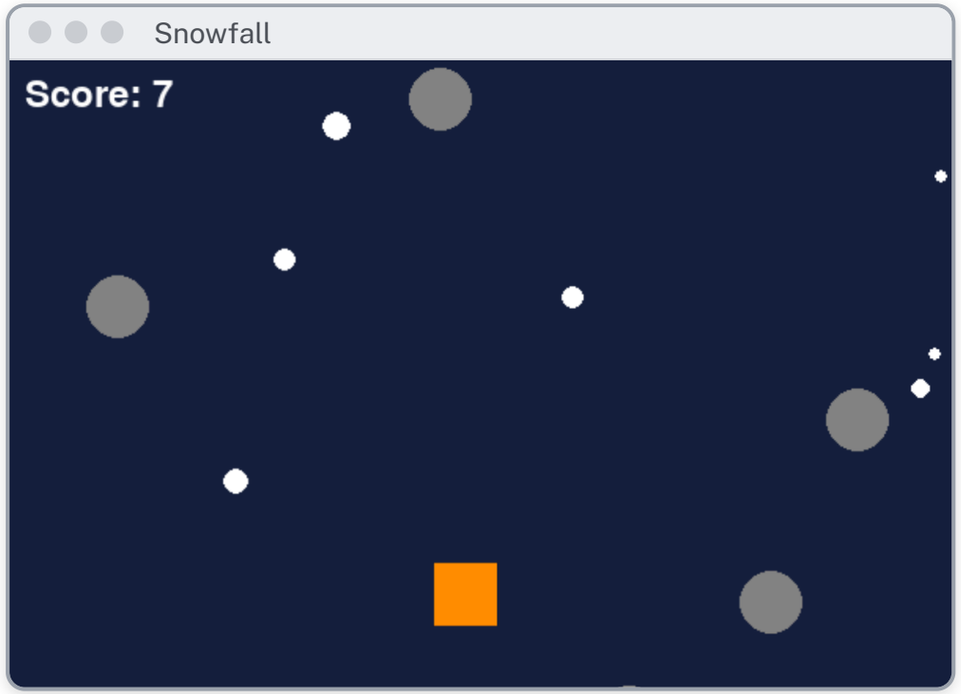
The notes build the same ideas with slightly different code

	Notes 16	These videos and Lab 6	
Showing the frame	**`pygame.display.update()`**	**`pygame.display.flip()`**	Either works
Speed	pixels per second, times **`delta_t`**	pixels per frame	No **`delta_t`**
The surface	stored in the ball: **`ball.draw()`**	passed in: **`ball.draw(screen)`**	Same result
Moving the player	**`KEYDOWN`** events: one step per press	**`get_pressed()`**: smooth while held	See Video 3
Collisions	two balls: compare the distance	player and rock: **`colliderect`**	Listed in the notes

Quiz 10 covers the videos and activities

Text, images, and the rest of Notes 16 are for Homework 7

Friday: Lab 6 turns this into Snowfall



- Bring your `day20_game.py`
- You'll compare your classes with the lab's versions
- Then add snowflakes to catch, and a score

To do, and coming up

- Finish the four activities, and review this slide deck and Notes 16
- Homework 6 is due Thursday 11/19
- Lab 6 is Friday 11/20: bring your `day20_game.py`
- Quiz 10 (Lectures 19–20) is the first class after break
- Exam 4 (Lectures 15–20) is Friday 12/4 in lab

See practice materials

- Homework 7, a game of your own, is due Thursday 12/10

Next time

Lecture 21 — complexity, after break