Middlebury

CSCI 146: Intensive Introduction to Computing

Fall 2025

Lecture 15: More OOP + Games

Some review from last class: using "inheritance" to *derive* a *child* `Dollar` class from the *base* `Rational` class:

```python
class Dollar(Rational):
    def __init__(self, cents):
        # Use `super()` to call `Rational`'s initializer,
        # passing through the cents as the numerator
        super().__init__(cents, 100)

    def __str__(self):
        dollars = self.numerator // 100
        cents = self.numerator % 100
        cents_str = str(cents) if cents >= 10 else "0" + str(cents)
        return "$" + str(dollars) + "." + cents_str
```

Rational also had __str__

uses this one

Trying to add two `Dollar` objects currently gives a `Rational` object.

```
>>> d1 = Dollar(10)
>>> d2 = Dollar(20)
>>> d3 = d1 + d2 # uses the `Rational` __add__ method, which returns a `Rational`
```

So, we can modify `Rational`'s `__add__` method to ensure it returns an instance of the appropriate class. How? Use the `__class__` dunder method.

```
def __add__(self, other):
    num = self.numerator * other.denominator + other.numerator * self.denominator
    den = self.denominator * other.denominator
    return self.__class__(num, den) # __class__ will be Rational or Dollar types
```

... but this doesn't completely work because `Dollar`'s `__init__` method only takes one argument (we can fix this by adding an optional argument to `__init__`).

Question 4 (from last class): After the code below executes, what is the value of `val.x`?

```
class A:
    def __init__(self, x):
        self.x = x

    def act_on(self, value):
        self.x += value
```

```
class B(A):
    def __init__(self, x, y):
        super().__init__(x)
        self.y = y

    def act_on(self, value):
        self.x += self.y*value

val = B(4, 2)
val.act_on(2)
```

x = 4
y = 2

4 + 2*2 = 8

A. 6

B. 8

C. 10

D. 16

E. 20

Question 5 (from last class): Which correctly describes the methods executed in the code below (listed as `class.method`)?

```
d1 = Dollar(10)
d2 = Dollar(20)
d3 = d1 + d2
```

A. `Dollar.__init__`

B. `Rational.__init__`

C. `Dollar.__init__,Rational.__init__`

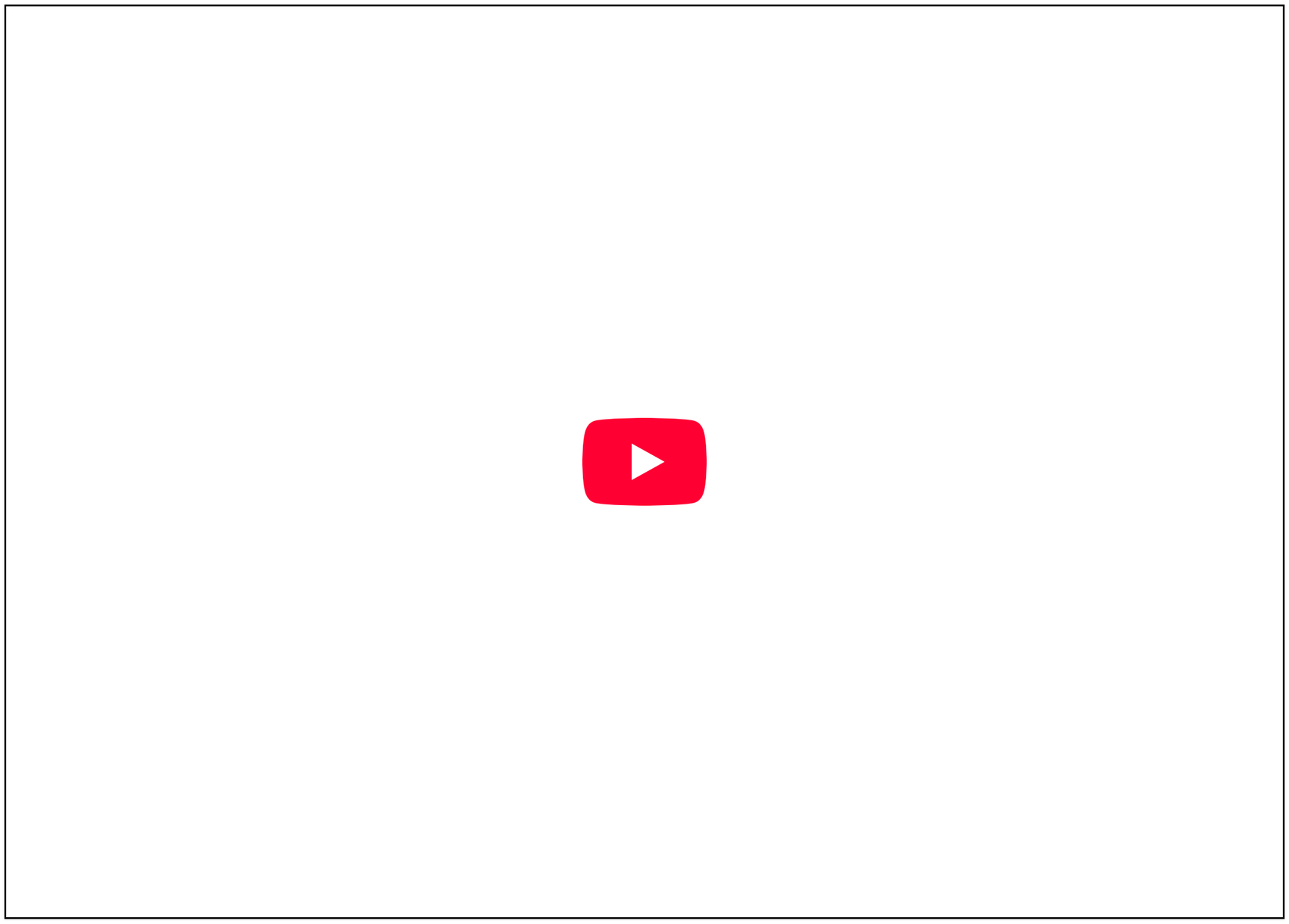
D. `Dollar.__init__,Rational.__init__,Rational.__add__`

E. `Dollar.__init__,Rational.__init__,Rational.__add__,`
`Dollar.__add__`

A B C D E

0 0 0 27 0 0 0 0 0 0 1 11 32 0 0

Today, we'll mostly make this animation, and then extend it to make a game.



Main goals: practice with OOP (applied to a game) and become familiar with `PyGame`.

Why OOP and games? Let's consider functionalities of games:

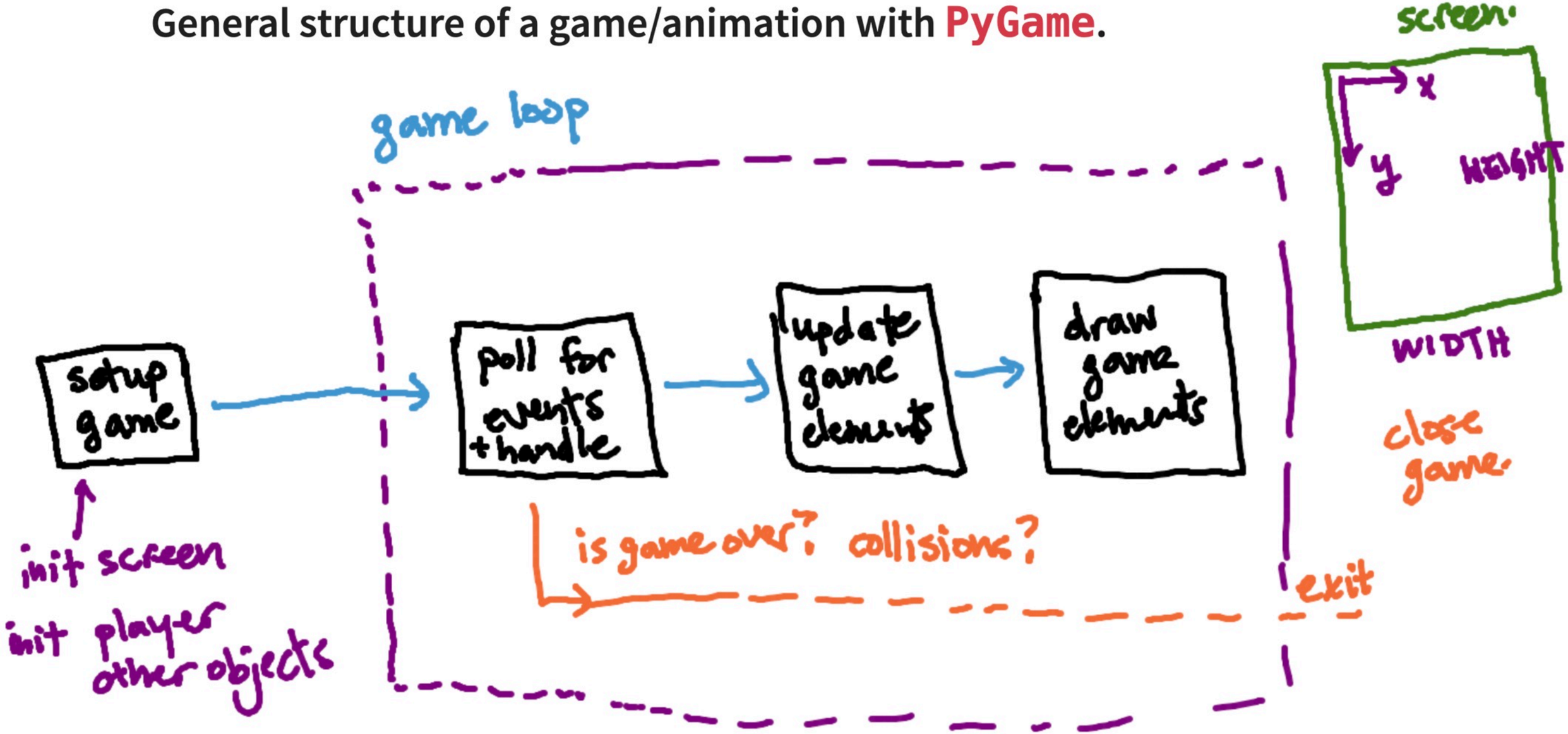
- Move the player (i.e., change the player's position)
- Move the obstacles, including bouncing off the walls
- Check if a player and obstacle collide

methods for the player or obstacle object.

class definition for these objects

(render)

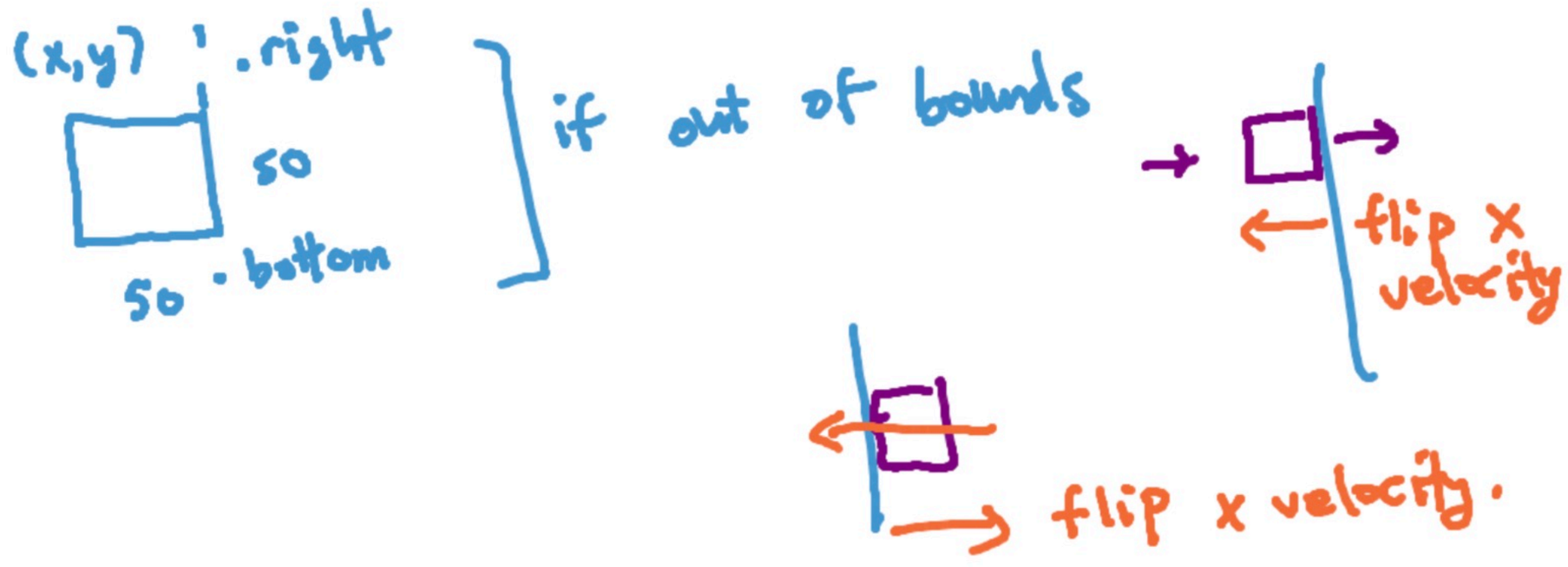
General structure of a game/animation with PyGame.



Download the `simple_pygame_starter.py` starter file. Look for the `TODO` comments.

Within `PyGame`, there are several other modules for drawing, setting up the screen, handling game mechanics, controlling the framerate, etc.

- TODO 1: reflect the box off the wall
- TODO 2: move the `pygame.draw.rect` call inside the `Box` `render` method.
- TODO 3: call the `box.render` method where `pygame.draw.rect` used to be.



Turning this into a game (it's currently just an animation).

I would suggest copying your current file to a new file called `complex_pygame.py`.

Things we want to do:

- Make another `Box` that represents the player.
- Render the player as an avatar (download one of the emojis from the **Reactions** page?).
- Move the player with arrow key presses.
- The game is over if the player box collides with the obstacle box.
- Add several obstacle boxes!

Handling user input:

```
event = pygame.event.poll()
if event.type == pygame.QUIT:
    break

if event.type == pygame.KEYDOWN:
    if (event.key == pygame.K_RIGHT):
        # Handle right
    elif (event.key == pygame.K_LEFT):
        # Handle left
    elif (event.key == pygame.K_UP):
        # Handle up
    elif (event.key == pygame.K_DOWN):
        # Handle down
```

Summary and Reminders

- **Terminology:** class, object, method, instance variables, base/parent class, derived/child class.
- **Quiz 7 on Friday:** it won't be a puzzle this time :(
- Programming Assignment 5 final due date on Thursday.
- Programming Assignment 6 initial due date on Thursday.
- See Gradescope comments by clicking on **Code**!