Python Object-Oriented Program with Libraries

Unit 1: PyGame Tutorial

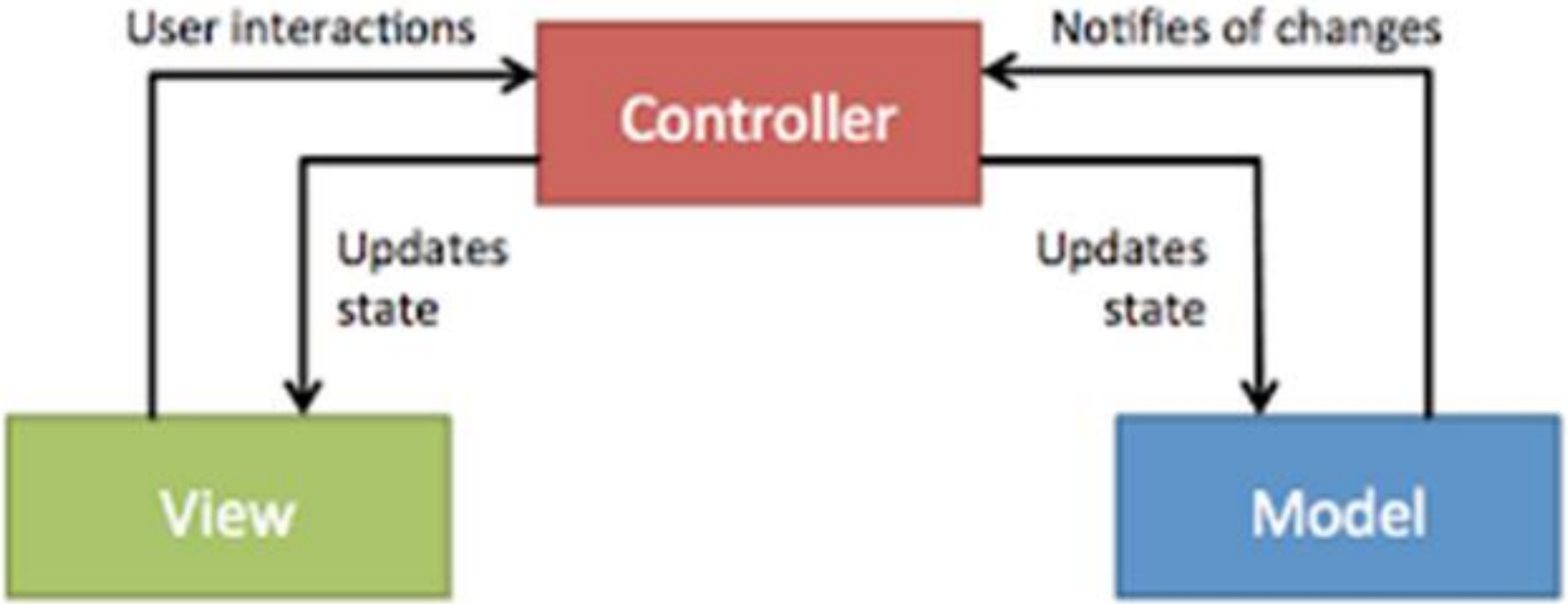
CHAPTER 6: SIMPLE MODEL VIEW GAME DESIGN

DR. ERIC CHOU

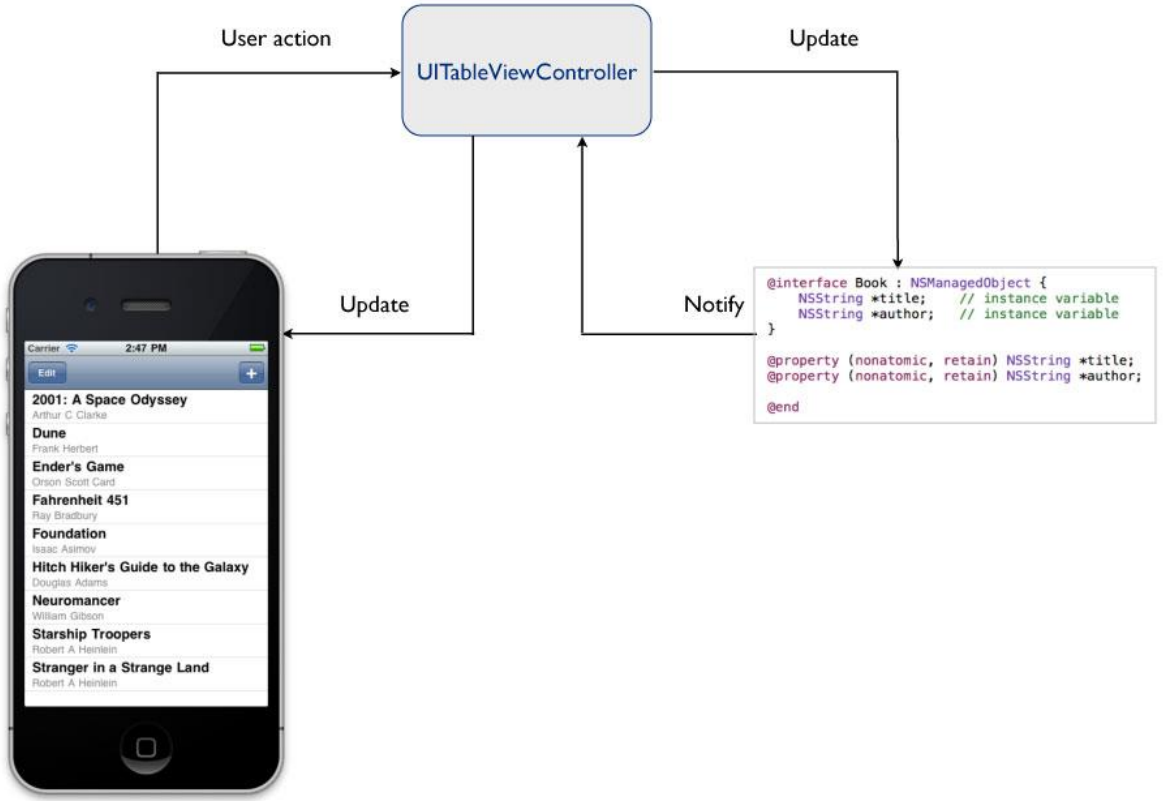
IEEE SENIOR MEMBER

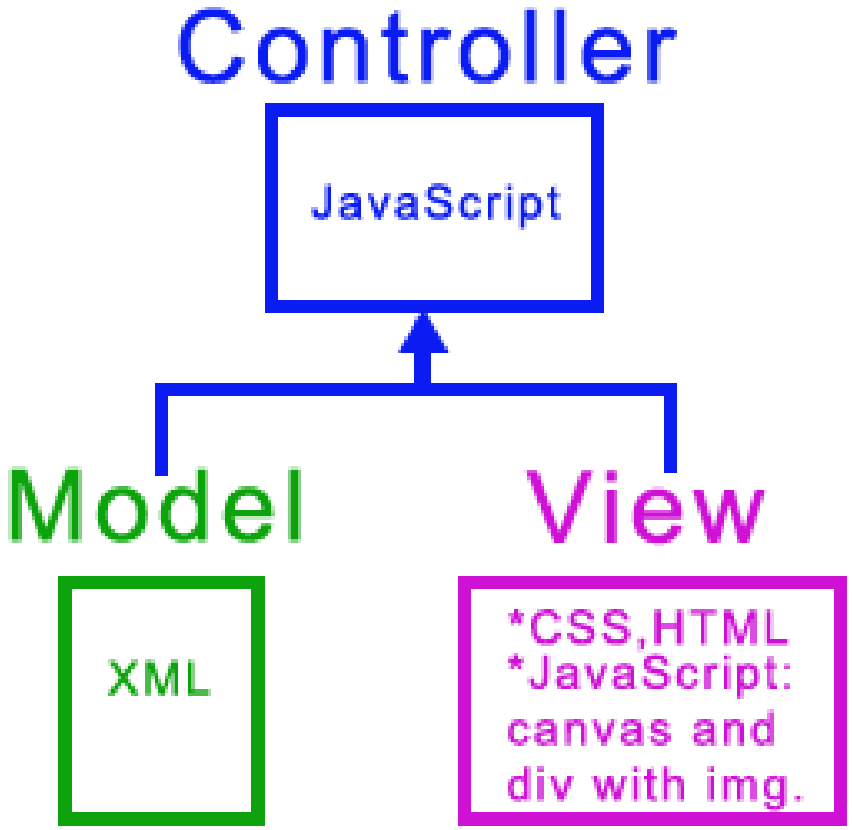
Model View Design Pattern for Game

LECTURE 1


User interactions
Notifies of changes
Controller
Updates state
Updates state
View
Model

Model View Controller Design Pattern for JavaScript





Case Study Simple Spots Project

LECTURE 2

Demo Program: pygame_spots.py + spots.py

- Two files: pygame_spots.py and spots.py
- Three classes:
  - SportsGame (in pygame_spots.py): game controller and view
  - Spot (in spots.py): data model for single spot
  - SpotsState (in spots.py): data model for a list of spots

Controller

```
class SpotsGame:
    def __init__(self):
        self._running = True
        self._state = spots.SpotsState()

    def run(self) -> None:
        pygame.init()

        self._resize_surface((600, 600))

        clock = pygame.time.Clock()

        while self._running:
            clock.tick(30)
            self._handle_events()
            self._redraw()

        pygame.quit()
```

Model

List of Circle

```
class SpotsState:
    def __init__(self):
        self._spots = []


    def all_spots(self) -> [Spot]:
        return self._spots
```

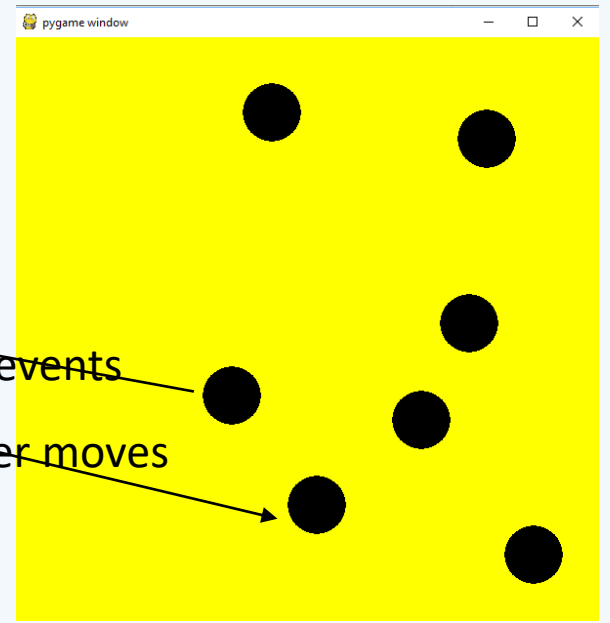
Circle

```
class Spot:
    def __init__(self, center:
                 (float, float),
                 radius: float):
        self._center = center
        self._radius = radius
```

Update model

Retrieve New Spots Data

View



Mouse click events

Re-draw the Spots after moves

Main Program: pygame_spots.py

1. SportsGame() constructor create the data model (calls SportsState() and build a spot list)

```
if __name__ == '__main__':
    SpotsGame().run()
```

2. run() methods is the main program control loop

```
class SpotsGame:
    def __init__(self):
        self._running = True
        self._state = spots.SpotsState()

    def run(self) -> None:
        pygame.init()

        self._resize_surface((600, 600))

        clock = pygame.time.Clock()

        while self._running:
            clock.tick(30)
            self._handle_events()
            self._redraw()

        pygame.quit()
```

```
def run(self) -> None:
    pygame.init()

    self._resize_surface((600, 600))

    clock = pygame.time.Clock()

    while self._running:
        clock.tick(30)
        self._handle_events()
        self._redraw()

    pygame.quit()
```

Start the pygame event loop

This starts the game view

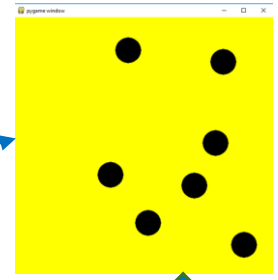
while running is the Game Loop

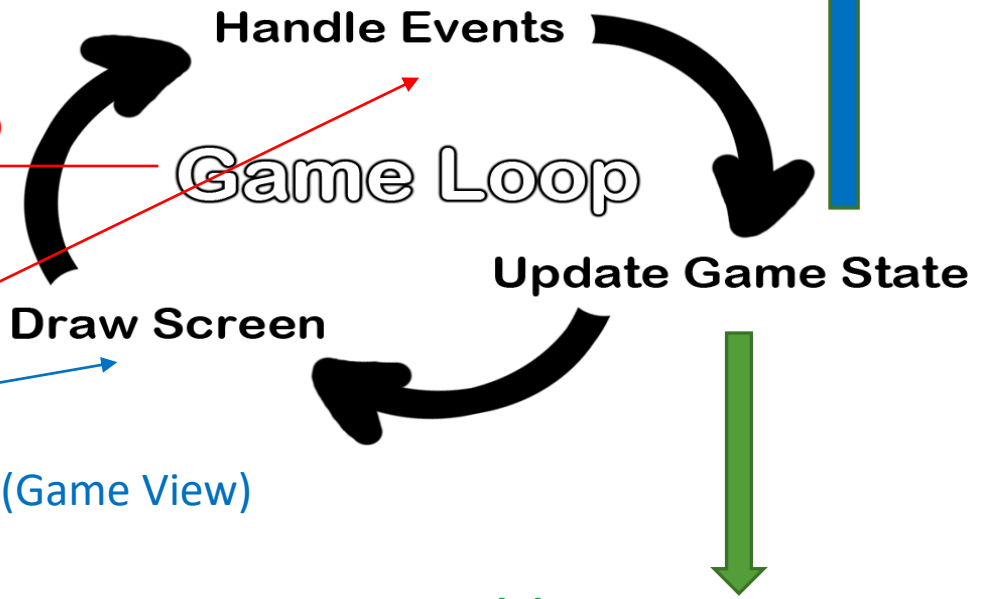
rest for 30 ticks for game player's comforts **(Event will enter event queue in this period)**

Update the Screen (Game View)

End the pygame event loop

Game View





Game Model

self._state => [spot1, spot2, spot3]

Main Loop

LECTURE 3

1. Creating the Game View

```
def _resize_surface(self, size: (int, int)) -> None:
    pygame.display.set_mode(size, pygame.RESIZABLE)
```

- **pygame.display.set_mode(size)** will create a surface (window canvas) which holds all the graphic components of this game.
- **pygame.RESIZABLE** make the window's size flexible if resize action is needed.

2. Handle Events

Three events to be handled and move_spots in every 30 ticks.

```
def _handle_events(self) -> None:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            self._end_game()
        elif event.type == pygame.VIDEORESIZE:
            self._resize_surface(event.size)
        elif event.type == pygame.MOUSEBUTTONDOWN:
            self._on_mouse_button(event.pos)

    self._move_spots()
```

Note:

1. **Moust Down** is the only event to update view and model.
2. If the display is set with the **pygame.RESIZABLE** flag, **pygame.VIDEORESIZE** events will be sent when the user adjusts the window dimensions.

3. redraw()

Update the View and Model

```
def _redraw(self) -> None:
    # get current surface object in the game loop
    surface = pygame.display.get_surface()
    # set the background color
    surface.fill(pygame.Color(255, 255, 0))
    # create spots object on view
    self._draw_spots()
    # update the view
    pygame.display.flip()
```

When mous is clicked on_mouse_button(pos)

LECTURE 4

Private Method: _on_mouse_button

```
def _on_mouse_button(self, pos: (int, int)) ->
None:
    surface = pygame.display.get_surface()
    width = surface.get_width()
    height = surface.get_height()
    # mouse position (x,y) in pixels
    pixel_x, pixel_y = pos
    # convert the (x, y) to [0, 1) scale
    frac_x = pixel_x / width
    frac_y = pixel_y / height
    # update data model to create or delete spots
    self._state.handle_click((frac_x, frac_y))
```

_state.handle_click((frac_x, frac_y))

```
# remove the spots that contains the mouse click point
# click point coordindates are in [0, 1) floating point format
def handle_click(self, click_point: (float, float)) -> None:
    # why reserved?  List need to be deleted from the end
    # otherwise the iterator will get the wrong spot
    for spot in reversed(self._spots):
        if spot.contains(click_point):
            self._spots.remove(spot)
            return
    self._spots.append(Spot(click_point, 0.05))
```

reversed()

reversed() Parameters

The reversed() method takes a single parameter:

seq - sequence that should be reversed

- Could be an object that supports sequence protocol (__len__() and __getitem__() methods) as tuple, string, list or range
- Could be an object that has implemented __reversed__()

Return value from reversed()

The reversed() method returns an **iterator** that accesses the given sequence in the **reverse order**.

Demo Program: reserved.py

- The reversed function allow a list (or sequence) to be accessed from the end of the list.
- And, this is very important when some item in the list is to be deleted.

```
# for string
seqString = 'Python'
print(list(reversed(seqString)))

# for tuple
seqTuple = ('P', 'y', 't', 'h', 'o', 'n')
print(list(reversed(seqTuple)))

# for range
seqRange = range(5, 9)
print(list(reversed(seqRange)))

# for list
seqList = [1, 2, 4, 3, 5]
print(list(reversed(seqList)))
```

```
ersed
C:\Python\Python36\python.exe "C:/Eric_Chou
['n', 'o', 'h', 't', 'y', 'P']
['n', 'o', 'h', 't', 'y', 'P']
[8, 7, 6, 5]
[5, 3, 4, 2, 1]
```

remove(spot)

This is the built-in function for python. Nothing special.

Create New Spot

```
def handle_click(self, click_point: (float, float)) ->
None:
    # why reserved?  List need to be deleted from the end
    # otherwise the iterator will get the wrong spot
    for spot in reversed(self._spots):
        if spot.contains(click_point):
            self._spots.remove(spot)
            return
    # create a new Spot with radius of 0.05 (5% of window)
    self._spots.append(Spot(click_point, 0.05))
```

Spot() constructor

```
class Spot:
    def __init__(self, center:
                 (float, float),
                  radius: float):
        self._center = center
        self._radius = radius

        # a number between -0.005 to +0.005  (1% motion)
        self._delta_x = (random.random() * 0.01) - 0.005
        self._delta_y = (random.random() * 0.01) - 0.005
```

redraw()

LECTURE 5

_draw_spots() is the most important function in _redraw()

```
def _redraw(self) -> None:
    surface = pygame.display.get_surface()

    surface.fill(pygame.Color(255, 255, 0))
    self._draw_spots()

    pygame.display.flip()
```

_draw_spots()

```
def _draw_spots(self) -> None:
    # draw all spots.
    for spot in self._state.all_spots():
        self._draw_spot(spot)
```

_draw_spot(spot)

```
def _draw_spot(self, spot: spots.Spot) -> None:
    # find the center of a spot
    frac_x, frac_y = spot.center()
    # finding (left, top) corner coordinate in fraction
    topleft_frac_x = frac_x - spot.radius()
    topleft_frac_y = frac_y - spot.radius()
    # Find the spot's box size (2 * radius)
    frac_width = spot.radius() * 2
    frac_height = spot.radius() * 2
    surface = pygame.display.get_surface()
    width = surface.get_width()
    height = surface.get_height()
    # finding (left, top) corner coordinate in pixels
    topleft_pixel_x = topleft_frac_x * width
    topleft_pixel_y = topleft_frac_y * height
    pixel_width = frac_width * width
    pixel_height = frac_height * height
    # draw the circle
    pygame.draw.ellipse(surface, pygame.Color(0, 0, 0),
        pygame.Rect(topleft_pixel_x, topleft_pixel_y, pixel_width, pixel_height))
```

_move_spots()

LECTURE 6

_move_spots in every 30 ticks even if no click.

```
def _handle_events(self) -> None:
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            self._end_game()
        elif event.type == pygame.VIDEORESIZE:
            self._resize_surface(event.size)
        elif event.type == pygame.MOUSEBUTTONDOWN:
            self._on_mouse_button(event.pos)

    self._move_spots()
```

_move_spots()

```
def _move_spots(self) -> None:
    self._state.move_all_spots()
```

move_all_spots(self)

```
def move_all_spots(self) -> None:
    for spot in self._spots:
        spot.move()
```

spot.move()

```
def move(self) -> None:
    x, y = self._center
    self._center = (x + self._delta_x, y + self._delta_y)


# delta_x and delta_y is a vector for the circle to move.
# it is defined in the Spot() class constructor.
```

Run Demo Program

LECTURE 7

Demo Program:

pygame_spots.py + spots.py