OBJECT ORIENTED PROGRAMMING WITH PYTHON

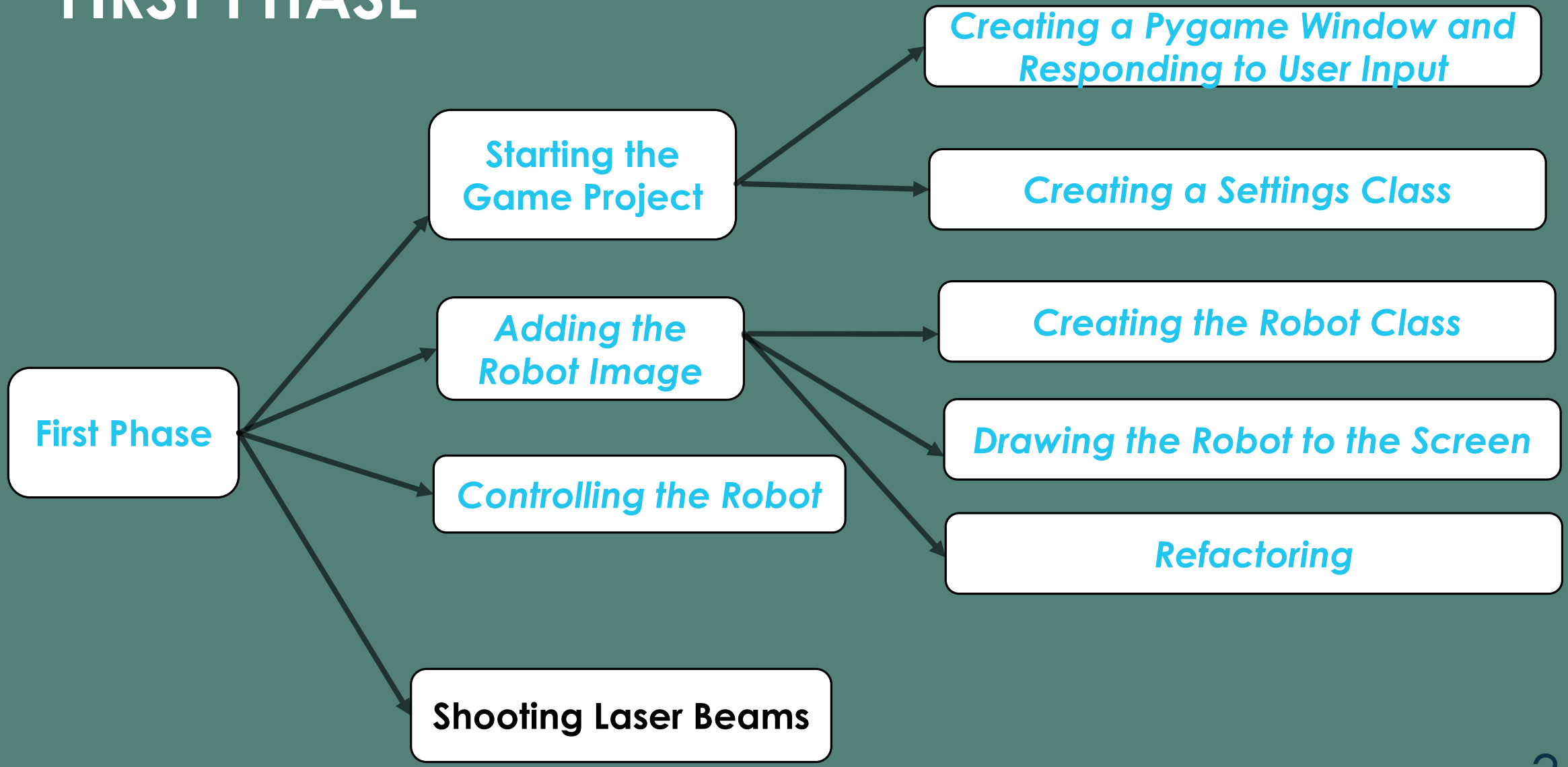
PROJECT BASED

2nd semester (Lect7)

Dr. Ann Al-Kazzaz

- *Adding the laser_beam Settings*
- *Creating the Laser_beam Class*
- *Storing Laser beams in a Group*
- *Firing Laser Beams*
- *Deleting Old Laser Beams*
- *Limiting the Number of Laser Beams*
- *Creating the _update_Laser() Method*

FIRST PHASE


First Phase
Starting the Game Project
Creating a Pygame Window and Responding to User Input
Creating a Settings Class
Adding the Robot Image
Creating the Robot Class
Drawing the Robot to the Screen
Refactoring
Controlling the Robot
Shooting Laser Beams

ADDING THE LASER_BEAM SETTINGS

At the beginning we'll update ***settings.py*** to include the values we'll need for a new laser_beam class:

settings.py

```
def __init__(self):
    --snip--
    # laser_beam settings
    self.laser_beam_speed = 1.0
    self.laser_beam_width = 2
    self.laser_beam_height = 20
    self.laser_beam_color = (20, 20, 200)
```

These settings create dark blue laser beams with a width of 2 pixels and a height of 20 pixels.

CREATING THE LASER_BEAM CLASS

Now create a ***laser_beam.py*** file to store our **Laser_beam class**.

laser_beam.py

```
import pygame
from pygame.sprite import Sprite
class Laser_beam(Sprite):
        """A class to manage laser beams fired from the robot"""
        def __init__(self, vi_game):
                """Create a Laser_beam object at the robot's current position."""
1       super().__init__()
                self.screen = vi_game.screen
                self.settings = vi_game.settings
                self.color = self.settings.laser_beam_color
                # Create a laser_beam rect at (0, 0) and then set correct position.
2       self.rect = pygame.Rect(0, 0, self.settings.laser_beam_width,self.settings.laser_beam_height)
3       self.rect.midtop = vi_game.robot.rect.midtop
                # Store the laser_beam's position as a decimal value.
4       self.y = float(self.rect.y)
```

CREATING THE LASER_BEAM CLASS

1. The Laser_beam class inherits from **Sprite**, which we import from the **pygame. sprite module**. When you use sprites, you can group related elements in your in your game and act on all the grouped elements at once.

2. we create the **Laser_beam's rect** attribute. The Laser_beam isn't based on an image, so we have to build a rect from scratch using the **pygame.Rect()** class.This class requires the **x- and y-coordinates** of the **top-left corner** of the rect, and the **width** and **height** of the rect. We initialize the rect at **(0, 0)**, but we'll move it to the correct location in the next line, because the Laser_beam's position depends on the robot's position. We get the width and height of the Laser_beam from the values stored in **self.settings.**

3. Here we set the Laser_beam's **midtop** attribute to match the **robot's midtop** attribute. This will make the Laser_beam emerge from the top of the robot, making it look like the Laser_beam is fired from the robot.

4. We store a decimal value for the Laser_beam's y-coordinate so we can make fine adjustments to the Laser_beam's speed

CREATING THE LASER_BEAM CLASS

the second part of ***laser_beam.py***, **update()** and **draw_laser_beam()**:

laser_beam.py

```
def update(self):
    """Move the laser_beam up the screen."""
    # Update the decimal position of the laser_beam.
1   self.y -= self.settings.laser_beam_speed
    # Update the rect position.
2   self.rect.y = self.y
def draw_laser_beam(self):
        """Draw the laser_beam to the screen."""
3       pygame.draw.rect(self.screen, self.color, self.rect)
```

- The **update()** method manages the laser_beam's position. When a laser_beam is fired, it moves up the screen, which corresponds to a decreasing **y-coordinate** value.

1. To update the position, we subtract the amount stored in **settings.laser_beam_speed** from **self.y**
2. We then use the value of **self.y** to set the value of **self.rect.y.**
3. When we want to draw a laser_beam, we call **draw_laser_beam()**. The **draw.rect()** function fills the part of the screen defined by the laser_beam's rect with the color stored in **self.color.**

STORING LASER BEAMS IN A GROUP

- Now After creating of **Laser_beam class** and defining the necessary **settings**, We'll create a group in **VirusInvasion** to store all the live laser beams so we can manage the laser beams that have already been fired. This group will be an instance of the **pygame.sprite.Group** class, which behaves like a list with some extra functionality that's helpful when building games.

- We'll use this group to draw laser beams to the screen on each pass through the main loop and to update each Laser_beam's position.

- We'll create the group in __init__():

virus_invasion.py

```
def __init__(self):
--snip--
self.robot = Robot(self)
self.laser_beams = pygame.sprite.Group()
```

STORING LASER BEAMS IN A GROUP

Then we need to update the position of the laser beams on each pass through the while loop

virus_invasion.py

```
def run_game(self):
    """Start the main loop for the game."""
    while True:
        self._check_events()
        self.robot.update()
        self.laser_beams.update()
        self._update_screen()
```

When you call **update()** on a group, the group automatically calls **update()** for each **sprite** in the group. The line **self.laser_beams.update()** calls **Laser_beam. update()** for each Laser_beam we place in the group laser beams.

FIRING LASER_BEAM

- In **VirusInvasion** class, we need to modify **_check_keydown_events()** to fire a Laser_beam when the player presses the spacebar.
- We don't need to change **_check_keyup _events()** because nothing happens when the spacebar is released.
- We also need to modify **_update_screen()** to make sure each laser_beam is drawn to the screen before we call **flip()**.
- We know there will be a bit of work to do when we fire a laser_beam, so let's write a new method, **_fire_laser_beam()**, to handle this work:

FIRING LASER BEAMS

```
--snip--
from robot import Robot
from laser_beam import Laser_beam
class AlienInvasion:
        --snip--
        def _check_keydown_events(self, event):
                --snip--
                elif event.key == pygame.K_q:
                        sys.exit()
                elif event.key == pygame.K_SPACE:
                        self._fire_laser_beam()
        def _check_keyup_events(self, event):
                --snip--
        def _fire_laser_beam(self):
                """Create a new laser_beam and add it to the laser beams group."""
                new_laser_beam =Laser_beam(self)
                self.laser_beams.add(new_laser_beam)
        def _update_screen(self):
                """Update images on the screen, and flip to the new screen."""
                self.screen.fill(self.settings.bg_color)
                self.robot.blitme()
                for laser_beam in self.laser_beams.sprites():
                        laser_beam.draw_laser_beam()
                pygame.display.flip()
        --snip--
```

1. First, we import **Laser_beam.**
2. call **_fire_laser_beam()** when the spacebar is pressed.
3. In _fire_laser_beam(), we make an nstance of laser_beam and call it **new_laser_beam.**
4. We then add it to the group laser beams using the **add()** method

   Note: the **add()** method is similar to **append()**, but it's a method that's written specifically for **Pygame** groups.
5. To draw all fired laser beams to the screen, we loop through the sprites in laser beams and call **draw_laser_beam()** on each one.

DELETING OLD LASER_BEAM

- At the moment, the laser beams disappear when they reach the top, but only because Pygame can't draw them above the top of the screen. The laser beams actually continue to exist; their **y-coordinate** values just grow increasingly negative. This is a problem, because they continue to consume memory and processing power.

- We need to get rid of these old laser beams, or the game will slow down from doing so much unnecessary work. To do this, **we need to detect when the bottom value of a laser_beam's rect has a value of 0**, which indicates the laser_beam has passed off the top of the screen:

FIRING LASER BEAMS

virus_invasion.py

```
def run_game(self):
        """Start the main loop for the game."""
        while True:
            self._check_events()
            self.robot.update()
            self.laser_beams.update()
            # Get rid of laser_beams that have disappeared.
1           for laser_beam in self.laser_beams.copy():
2               if laser_beam.rect.bottom <= 0:
3                   self.laser_beams.remove(laser_beam)
4           print(len(self.laser_beams))
            self._update_screen()
```

- When you use a **for** loop with a list (or a **group** in Pygame), Python expects that the list will stay the same length as long as the loop is running.
- Because we can't remove items from a list or group within a for loop, we have to loop over a copy of the group.

1. We use the **copy()** method to set up the for loop, which enables us to modify laser beams inside the loop.
2. check each laser_beam to see whether it has disappeared off the top of the screen
3. If it has, we remove it from laser beams.
4. we insert a print() call to show how many laser beams currently exist in the game and verify that they're being deleted when they reach the top of the screen.

DELETING OLD LASER_BEAM

- If this code works correctly, we can watch the terminal output while firing laser beams and see that the number of laser beams decreases to zero after each series of laser beams has cleared the top of the screen.

- After you run the game and verify that laser beams are being deleted properly, remove the print() call. If you leave it in, the game will slow down significantly because it takes more time to write output to the terminal than it does to draw graphics to the game window.

LIMITING THE NUMBER OF LASER BEAMS

Many shooting games limit the number of laser beams a player can have on the screen at one time; doing so encourages players to shoot accurately. We'll do the same in *VirusInvasion*.

First, store the number of laser beams allowed in ***settings.py***

settings.py

```
# laser_beam settings
--snip--
self.laser_beam_color = (20, 20, 200)
self.laser_beams_allowed = 3
```

This limits the player to three laser beams at a time.

Second, We'll use this setting in **VirusInvasion** to check how many laser beams exist before creating a new laser_beam in **_fire_laser()**:

virus_invasion.py

```
def _fire_laser(self):
    """Create a new laser_beam and add it to the laser beams group."""
    if len(self.laser_beams) < self.settings.laser_beams_allowed:
        new_laser_beam = laser_beam(self)
        self.laser_beams.add(new_laser_beam)
```

When the player press spacebar, we check the length of **laser_beams**. If **len(self.laser_beams)** is less than three, we create a new laser_beam. But if three laser beams are already active, nothing happens.

REFACTORING

We want to keep the **VirusInvasion** class reasonably well organized, so now that we've written and checked the laser_beam management code, we can move it to a separate method. We'll create a new method called **_update_laser_beams()** and add it just before **_update_screen()**:

virus_invasion.py

```
def _update_laser_beams(self):
    """Update position of laser_beams and get rid of old laser beams."""
    # Update laser_beam positions.
    self.laser_beams.update()
    # Get rid of laser_beams that have disappeared.
    for laser_beam in self.laser_beams.copy():
        if laser_beam.rect.bottom <= 0:
            self.laser_beams.remove(laser_beam)
```

The code for _update_laser_beams() is cut and pasted from run_game(); all we've done here is clarify the comments.

The while loop in **run_game()** looks simple again:

virus_invasion.py

```
while True:
    self._check_events()
    self.robot.update()
    self._update_laser_beams()
    self._update_screen()
```