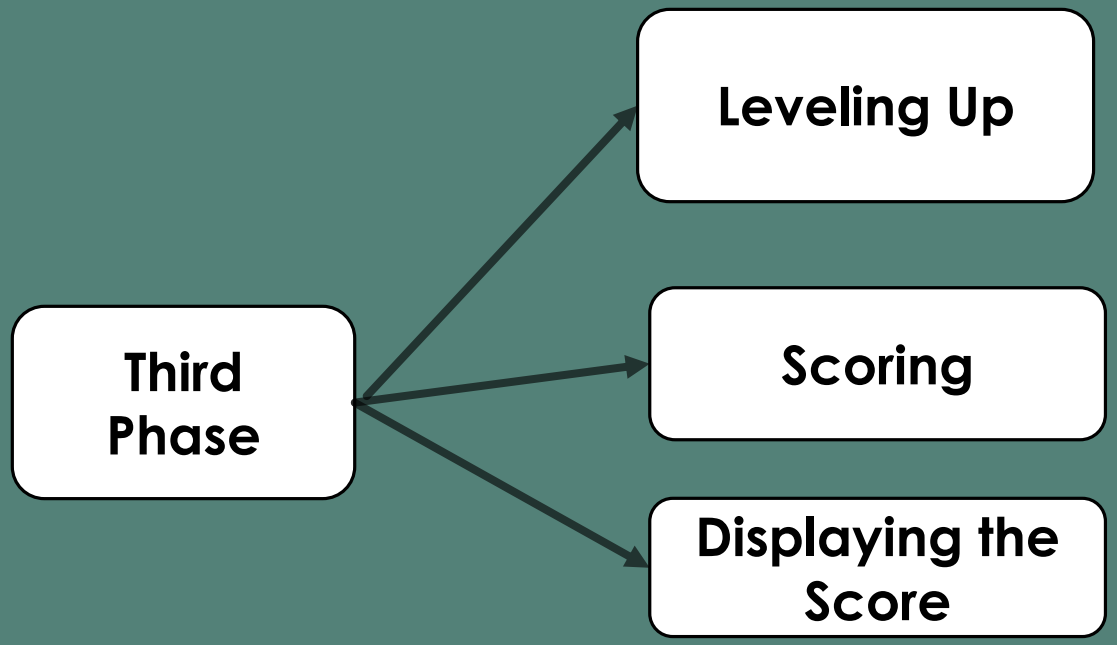
OBJECT ORIENTED PROGRAMMING WITH PYTHON

PROJECT BASED

2nd semester (Lect11)

Dr. Ann Al-Kazzaz

THIRD PHASE



LEVELING UP *Modifying the Speed Settings*

- We'll first reorganize the Settings class to group the game settings into static and changing ones. We'll also make sure that settings that change ***virus_invasion.py*** during the game reset when we start a new game. Here's the **__init__()** method for ***settings.py***:
- Finally, we call the **initialize_dynamic_settings()** method to initialize the values for attributes that need to change throughout the game

setting.py

```
def __init__(self):
    """Initialize the game's static settings."""
    # Screen settings
    self.screen_width = 1200
    self.screen_height = 800
    self.bg_color = (230, 230, 230)
    # Robot settings
    self.robot_limit = 3
    # laser_beam settings
    self.laser_beam_width = 3
    self.laser_beam_height = 15
    self.laser_beam_color = 60, 60, 60
    self.laser beams_allowed = 3
    # Virus settings
    self.group_drop_speed = 10
    # How quickly the game speeds up
1   self.speedup_scale = 1.1

2   self.initialize_dynamic_settings()
```

LEVELING UP

- We continue to initialize those settings that stay constant in the **__init__()** method. We add a **speedup_scale** setting to control how quickly the game speeds up: a value of 2 will double the game speed every time the player reaches a new level; a value of 1 will keep the speed constant. A value like 1.1 should increase the speed enough to make the game challenging but not impossible.
- Here's the code for **initialize_dynamic_settings()**:

setting.py

```
def initialize_dynamic_settings(self):
    """Initialize settings that change throughout the game."""
    self.robot_speed = 1.5
    self.laser_beam_speed = 3.0
    self.virus_speed = 1.0

    # group_direction of 1 represents right; -1 represents left.
    self.group_direction = 1
```

This method sets the initial values for the robot, laser_beam, and virus speeds.

LEVELING UP

To increase the speeds of the robot, laser beams, and viruses each time the player reaches a new level, we'll write a new method called **increase_speed()**:

setting.py

```
def increase_speed(self):
    """Increase speed settings."""
    self.robot_speed * = self.speedup_scale
    self.laser_beam_speed * = self.speedup_scale
    self.virus_speed * = self.speedup_scale
```

- To increase the speed of these game elements, we multiply each speed setting by the value of **speedup_scale**.
- We increase the game's tempo by calling **increase_speed()** in **_check _laser_beam_virus_collisions()** when the last virus in a group has been shot down:

virus_invasion.py

```
def _check_laser_beam_virus_collisions(self):
    --snip--
    if not self.viruses:
        # Destroy existing laser beams and create new group.
        self.laser beams.empty()
        self._create_group()
        self.settings.increase_speed()
```

- Changing the values of the speed settings **robot_speed, virus_speed**, and **laser_beam_speed** is enough to speed up the entire game!

SCORING

Let's implement a scoring system to track the game's score in real time and display the high score, level, and number of robots remaining. The score is a game statistic, so we'll add a score attribute to **GameStats**:

game_stats.py

```
class GameStats:
    --snip--
    def reset_stats(self):
     """Initialize statistics that can change during the game."""
     self.robots_left = self.sj_settings.robot_limit
     self.score = 0
```

To reset the score each time a new game starts, we initialize score in **reset_stats()** rather than **__init__().**

DISPLAYING THE SCORE

To display the score on the screen, we first create a new class, **Scoreboard**. For now, this class will just display the current score, but eventually we'll use it to report the high score, level, and number of robots remaining as well. Here's the first part of the class; save it as ***scoreboard.py***

scoreboard.py

```
import pygame.font
class Scoreboard:
    """A class to report scoring information."""

1   def __init__(self, vi_game):
        """Initialize scorekeeping attributes."""
        self.screen = vi_game.screen
        self.screen_rect = self.screen.get_rect()
        self.settings = vi_game.settings
        self.stats = vi_game.stats

        # Font settings for scoring information.
2       self.text_color = (30, 30, 30)
3       self.font = pygame.font.SysFont(None, 48)
        # Prepare the initial score image.
4       self.prep_score()
```

- Because Scoreboard writes text to the screen, we begin by importing the **pygame.font module**.

1. Next, we give **__init__()** the vi_game parameter so it can access the settings, screen, and stats objects, which it will need to report the values we're tracking .

2. Then we set a text color

3. and instantiate a font object .

4. To turn the text to be displayed into an image, we call **prep_score()**

DISPLAYING THE SCORE

scoreboard.py

```
def prep_score(self):
    """Turn the score into a rendered image."""
1   score_str = str(self.stats.score)
2   self.score_image = self.font.render(score_str, True,
                self.text_color, self.settings.bg_color)

    # Display the score at the top right of the screen.
3   self.score_rect = self.score_image.get_rect()
4   self.score_rect.right = self.screen_rect.right - 20
5   self.score_rect.top = 20

6 def show_score(self):
    """Draw score to the screen."""
    self.screen.blit(self.score_image, self.score_rect)
```

1. Here, we turn the numerical value **stats.score** into a string, and then pass this string to render(), which creates the image.
2. To display the score clearly on screen, we pass the screen's background color and the text color to **render().**
3. We'll position the score in the **upper-right** corner of the screen and have it expand to the left as the score increases and the width of the number grows. To make sure the score always lines up with the right side of the screen, we create a rect called **score_rect.**
4. and set its right edge 20 pixels from the right edge of the screen
5. We then place the top edge 20 pixels down from the top of the screen

6. This method draws the score image onscreen at the location **score_rect** specifies.

MAKING A SCOREBOARD

To display the score, we'll create a Scoreboard instance in VirusInvasion. **First**, let's update the import statements:

virus_invasion.py

```
--snip--
from game_stats import GameStats
from scoreboard import Scoreboard
--snip--
```

Next, we make an instance of Scoreboard in **__init__()**:

virus_invasion.py

```
def __init__(self):
    --snip--
    pygame.display.set_caption("Virus Invasion")
    # Create an instance to store game statistics,
    # and create a scoreboard.
    self.stats = GameStats(self)
    self.sb = Scoreboard(self)
    --snip--
```

MAKING A SCOREBOARD

Then we draw the scoreboard on screen in **_update_screen():**

virus_invasion.py

```
def _update_screen(self):
    --snip--
    self.viruses.draw(self.screen)
    # Draw the score information.
    self.sb.show_score()
    --snip--
```

When you run *virus invasion* now, a 0 should appear at the top right of the screen. (At this point, we just want to make sure the score appears in the right place before developing the scoring system further.)