CSc11300 Final Project
Question Idea Explain

Gong Qi Chen 07/28/2018

Try It Yourself:

12-1 implement in Alien Invasion folder

I updated the bg_color variable from Setting class to (0, 190, 220), which is a light blue color. This will apply to update_screen() method and change the color of the background.

```
        # 12-1 implementation Setting background color
        self.bg_color = (0,190,220)
```

12-2 implement in Alien Invasion folder

I used photoshop to create a Star Fox PNG file with the transparent background color. Then I created a class call Character use the class Ship as super class and paint it in the center of the screen.

```
class Character(Ship):
    def __init__(self, ai_settings, screen):
        # Initialize the Character using super() to call the Ship constructor
        super().__init__(ai_settings, screen)

        # Load the character image and get its rect
        self.image = pygame.image.load('images/Star_Fox.png')
        self.rect = self.image.get_rect()
        self.screen_rect = screen.get_rect()

        # Set the ship at the bottom center of the screen at every start
        self.rect.center = self.screen_rect.center
```

12-3 implement in Alien Invasion folder

I set the rect.centerx and rect.centery to the screen centerx and century in class ship to make the rocket in the center of the screen. I created event.key pygame.K_UP and pygame.K_DOWN for both key press and key release The key pressing will set the movement flag to true and make the ship move up or down continuously. The key releasing will set the movement flag to false and the stop the movement. I also created conditional statements in ship's update method to make ship stay in bound.

```
elif event.key == pygame.K_UP:
    # 12-3 implementation
    # Set the movement flags moving_up to false when the up key is release
    ship.moving_up = False
elif event.key == pygame.K_DOWN:
    # 12-3 implementation
    # Set the movement flags moving_down to false when the down key is release
    ship.moving_down = False
```

```
        # 12-3 implementation
        # Movement flag
        self.moving_right = False
        self.moving_left = False
        self.moving_up = False
        self.moving_down = False

    # 12-3 implementation
    def update(self):
        # Update the ship's center base on movement flag and bound its moveing range within the screen
        if self.moving_right and self.rect.right < self.screen_rect.right:
            self.xcenter += self.ai_settings.ship_speed_factor
        if self.moving_left and self.rect.left > 0:
            self.xcenter -= self.ai_settings.ship_speed_factor
        if self.moving_up and self.rect.top > 0:
            self.ycenter -= self.ai_settings.ship_speed_factor
        if self.moving_down and self.rect.bottom < self.screen_rect.bottom:
            self.ycenter += self.ai_settings.ship_speed_factor
```

12-4 implement in Keys folder

I created a new folder call Keys to implement this question, since it requires to create a new screen. I created an empty screen and check_events() method. The check_event() method print the event.key attribute whenever a pygame.KEYDOWN event is happening. The method was printing an integer number whenever I press a key.

```
# 12-4 implementation
def run_test():
    # Initialize pygame, settings, and screen object.
    pygame.init()
    ai_settings = Settings()
    screen = pygame.display.set_mode((ai_settings.screen_width, ai_settings.screen_height))
    pygame.display.set_caption("Key Testing")

    # Start the main loop for the test
    while True:
        gf.check_events()

run_test()
```

```
# 12-4 implementation
def check_events():
    # Respond to keypresses and mouse events
    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            # Exit the program when close the window
            sys.exit()
        elif event.type == pygame.KEYDOWN:
            # print the event.key attribute whenever a pygame.KEYDOWN event
            print(event.key)
```

12-5 implement in Alien Invasion folder

I created a new PNG file of the ship that is heading right direction and set the position to center left side of the screen (this is conflicting with 12-3 task, ask to position ship in the center). I reset the bullet size to have a longer width and short height in bullet class. Then I modified the update method from Bullet class that the bullet will increment in horizontal direction instead of vertical.

```
        # 12-5 implementation
        # Load the ship image and get its rect
        self.image = pygame.image.load('images/ship_sideway.png')
        self.rect = self.image.get_rect()
        self.screen_rect = screen.get_rect()

        # Set the ship at the center of the screen at every start
        self.rect.left = self.screen_rect.left
        self.rect.centery = self.screen_rect.centery
```

```
        # 12-5 implementation
        # Bullet settings
        self.bullet_speed_factor = 3.5
        self.bullet_width = 15
        self.bullet_height = 3
        self.bullet_color = 60, 60, 60
        self.bullets_allowed = 10
```

13-1 implement in Alien Invasion folder

I used the Alien class as a super class to create a Star class, since they are very similar in structure. Then I used the methods from game_functions modules to generate the stars grid, however I purposely make the star image background larger to make the star dispersed. get_number_items_y() and get_number_column() were used to calculate how star in the grid. create_item() were to create the group of stars. Then create_stars() called all of the method above to create the grid.

```python
def get_number_items_y(ai_settings, item_height):
    # Determine the number of items that fit in a column
    available_space_y = ai_settings.screen_height - 2 * item_height
    number_item_y = int(available_space_y / (2 * item_height))
    return number_item_y

def get_number_column(ai_settings, ship_width, item_width):
    # Determine the number of rows of items that fit on the screen
    available_space_x = (ai_settings.screen_width - (6 * item_width) - ship_width)
    number_column = int(available_space_x / (2 * item_width))
    return number_column

def create_item(item, items, item_number, column_number, rand_num):
    # Create an item and place it in the column
    # Use a random number to randomize the item position
    item_height = item.rect.height
    item.y = item_height + 2 * item_height * item_number + rand_num
    item.rect.y = item.y
    item.rect.x = 6 * item.rect.width + 2 * item.rect.width * column_number + rand_num
    items.add(item)
```

```python
# 13-1 implementation
class Star(Alien):
    # A class to represent single star on the screen
    def __init__(self, ai_settings, screen):
        # Initialize the star using super() to call the Alien constructor
        super().__init__(ai_settings, screen)

        # Load the star image and set its rect attribute.
        self.image = pygame.image.load('images/star.png')
        self.rect = self.image.get_rect()
```

13-2 implement in Alien Invasion folder

I generated a random number from -60 to 60, whenever I create a star. Then in create_item() method I used that random number to adjust the x, y coordinates of the star. In this way, the stars will be positioned randomly.

```python
# 13-1 & 13-2 implementation
def create_stars(ai_settings, screen, ship, stars):
    # Create a grid of stars
    # Create a star and find the number of stars in a column
    star = Star(ai_settings, screen)
    number_stars_y = get_number_items_y(ai_settings, star.rect.height)
    number_column = get_number_column(ai_settings, ship.rect.width, star.rect.width)

    # Create a grid of stars
    for column_number in range(number_column):
        # Create an star and place it in the column
        for star_number in range(number_stars_y):
            star = Star(ai_settings, screen)
            # Create a random number to randomize the position of the stars
            random_number = randint(-60, 60)
            create_item(star, stars, star_number, column_number, random_number)
```

13-3 implement in Alien Invasion folder

I created a new Raindrop class, using Alien as its super class, since their structure is very similar. Then I used the same methods that I used to generate stars in 13-1 to generate grid of raindrop. I created an update() method in Raindrop class to make the raindrop drop down. I also created an update_raindrop() method to include the update() method and remove the raindrop from the group once it is out of the screen. Then, I called the update_raindrop inside the main loop.

```python
# 13-3 implementation
class Raindrop(Alien):
    # A class to represent single rain_drop on the screen
    def __init__(self, ai_settings, screen):
        # Initialize the rain_drop using super() to call the Alien constructor
        super().__init__(ai_settings, screen)

        # Load the rain_drop image and set its rect attribute.
        self.image = pygame.image.load('images/raindrop.png')
        self.rect = self.image.get_rect()

    def update(self):
        # Move the raindrop down
        self.y += self.ai_settings.raindrop_speed_factor
        self.rect.y = self.y
```

```python
# 13-3 implementation
def create_raindrops(ai_settings, screen, ship, raindrops):
    # Create a grid of raindrops
    # Create a raindrop and find the number of raindrop in a column
    raindrop = Raindrop(ai_settings, screen)
    number_raindrops_y = get_number_items_y(ai_settings, raindrop.rect.height)
    number_column = get_number_column(ai_settings, ship.rect.width, raindrop.rect.width)

    # Create a grid of raindrops
    for column_number in range(number_column):
        # Create a raindrop and place it in the column
        for raindrop_number in range(number_raindrops_y):
            raindrop = Raindrop(ai_settings, screen)
            # Create a random number to randomize the position of the raindrops
            random_number = randint(-30, 30)
            create_item(raindrop, raindrops, raindrop_number, column_number, random_number)
```

13-4 implement in Alien Invasion folder

I created a stead_rain() method in game_functions that will create a new row of rain once the number of raindrop is below the raindrop_allow amount. The method structure is like the regular raindrop grid creator method. I included this method inside the update_raindrop() method so the deleting and creating of the old and new raindrop will take place at the same time.

```python
# 13-4 implementation
def steady_rain(ai_settings, screen, ship, raindrops):
    # Create a raindrop and find the number of raindrop in a row
    raindrop = Raindrop(ai_settings, screen)
    number_column = get_number_column(ai_settings, ship.rect.width, raindrop.rect.width)

    # Create a row of raindrops
    for column_number in range(number_column):
        raindrop = Raindrop(ai_settings, screen)
        # Create a random number to randomize the position of the raindrops
        random_number = randint(-60, 60)
        create_item(raindrop, raindrops, 0, column_number, random_number)
```

13-5 implement in Catch folder

I created a new project folder, Catch folder, to implement this question, since it is basically a different game from Alien Invasion. The basic idea and structure of this game is like the one we have in Alien Invasion. We replaced the ship with the character and aliens with ball. We created update_ball() and check_ball_bottom() methods in game_functions to make sure the ball disappear whenever it collided with the catcher or the bottom.

```python
def update_ball(ai_settings, stats, screen, catcher, balls):

    balls.update(ai_settings)

    # Look for ball-catcher collisions.
    if pygame.sprite.spritecollideany(catcher, balls):
        balls.empty()
        create_ball(ai_settings, screen, balls)

    # Look for ball hitting the bottom of the screen.
    check_ball_bottom(ai_settings, stats, screen, balls)
```

```python
def check_ball_bottom(ai_settings, stats, screen, balls):
    # Check if ball has reached the bottom of the screen
    screen_rect = screen.get_rect()
    for ball in balls:
        if ball.rect.bottom >= screen_rect.bottom:
            # The ball is miss call ball_miss method
            ball_miss(ai_settings, stats, screen, balls)
```

13-6 implement in Catch folder

Same as 13-5, I implement this question in Catch folder. I created ball_miss() in game_functions method to keep track of the number of the balls that hit the bottom, and set the game_active to false when that number is below zero to stop the game.

```python
def ball_miss(ai_settings, stats, screen, balls):
    # Respond to ball miss
    if stats.catcher_left > 0:
        # Decrement catcher_left
        stats.catcher_left -= 1
    else:
        stats.game_active = False

    # remove old ball and create a new ball
    balls.empty()
    create_ball(ai_settings, screen, balls)
```

14-1 implement in Alien Invasion folder

I created a new method call start_game() in game_functions modules, and made it reset the game statistics, the ship position, aliens, and bullets, if the game_active is false. Then I created condition statement in check_keydown_events() to call start_game() if P is pressed.

```python
# 14-1 implementation
def start_game(ai_settings, screen, stats, ship, aliens, bullets):
    # Start a new game with the method is call
    # Check if the game is off
    if not stats.game_active:
        # Hide the mouse cursor.
        pygame.mouse.set_visible(False)

        # Reset the game statistics.
        stats.reset_stats()
        stats.game_active = True

        # Empty the list of aliens and bullets.
        aliens.empty()
        bullets.empty()

        # Create a new fleet and center the ship.
        create_fleet(ai_settings, screen, ship, aliens)
        ship.center_ship()
```

```python
    elif event.key == pygame.K_p:
        # Start the game with P is press
        start_game(ai_settings, screen, stats, ship, aliens, bullets)
```

14-2 implement in Target Practice folder

I created a new project folder, Target Practice folder, to implement this question, since it is basically a different game from Alien Invasion. The basic idea and structure of this game is like the one we have in Alien Invasion. We replaced the aliens with the target bar and made the direction horizontal instead of vertical. We created a Button class so we can create a button to display on the screen. Then We created check_play_button() and start_game() in game_functions to make the button only appear as the game is not active and reset the game if the button is click. We also made a method target_miss() to set game_active to false if we miss three shots.

```python
class Button():
    def __init__(self, screen, msg):
        # Initialize button attributes.
        self.screen = screen
        self.screen_rect = screen.get_rect()
        # Set the dimensions and properties of the button.
        self.width, self.height = 200, 50
        self.button_color = (0, 255, 0)
        self.text_color = (255, 255, 255)
        self.font = pygame.font.SysFont(None, 48)
        # Build the button's rect object and center it.
        self.rect = pygame.Rect(0, 0, self.width, self.height)
        self.rect.center = self.screen_rect.center
        # The button message needs to be prepped only once.
        self.prep_msg(msg)

    def prep_msg(self, msg):
        # Turn msg into a rendered image and center text on the button
        self.msg_image = self.font.render(msg, True, self.text_color, self.button_color)
        self.msg_image_rect = self.msg_image.get_rect()
        self.msg_image_rect.center = self.rect.center
```

```python
def check_play_button(ai_settings, stats, play_button, ship, bullets, mouse_x, mouse_y):
    # Start a new game when the player clicks Play
    button_clicked = play_button.rect.collidepoint(mouse_x, mouse_y)
    if button_clicked:
        start_game(ai_settings, stats, ship, bullets)

def start_game(ai_settings, stats, ship, bullets):
    # Start a new game with the method is call
    # Check if the game is off
    if not stats.game_active:
        # Reset the game speed settings
        ai_settings.initialize_dynamic_settings()

        # Hide the mouse cursor.
        pygame.mouse.set_visible(False)

        # Reset the game statistics.
        stats.reset_stats()
        stats.game_active = True

        # Empty the list of aliens and bullets.
        bullets.empty()
        ship.center_ship()
```

14-3 implement in Target Practice folder

Same as 13-5, I implement this question in Target Practice folder. We created check_bullet_target_collisions() in game_functions method to monitor our hit at the target bar, and increase the speed once we hit it three times. This method called the increase_speed() method we created for Settings class to increase the speed of the game.

```python
def check_bullet_target_collisions(ai_settings, target, bullets):
    # Check for any bullets that hit target
    # If so, get rid of the bullet and the target
    bullet = pygame.sprite.spritecollideany(target, bullets)
    if bullet != None:
        ai_settings.target_hit += 1
        bullets.remove(bullet)

    if ai_settings.target_hit == 3:
        # Destroy existing bullets, speed up game
        bullets.empty()
        ai_settings.increase_speed()
        ai_settings.target_hit = 0
```

```python
    def increase_speed(self):
        # Increase speed settings
        self.ship_speed_factor *= self.speedup_scale
        self.bullet_speed_factor *= self.speedup_scale
        self.target_speed_factor *= self.speedup_scale
```

14-4 implement in Alien Invasion folder

I created method read_high_score() in GameStats class and used it to read score from high_score.txt. Then it assigned the amount to high_score attribute. I also created write_high_score() method in game_functions modules that write the high_score to high_score.txt.

Then I called it before all the exit operate to write the hign_score to the file.

```
        # 14-4 implementation
        # High score is read form the file and should never be reset during the game
        self.high_score = int(self.read_high_score())

    def reset_stats(self):
        # Initialize statistics that can change during the game
        self.ships_left = self.ai_settings.ship_limit
        self.score = 0
        self.level = 1

    # 14-4 implementation
    def read_high_score(self):
        # Read high_score from the file high_score.text
        f = open('high_score.txt', 'r')
        high_score = f.read()
        f.close()
        return high_score
```

```
# 14-4 implementation
def write_high_score(stats):
    # Write the high_score into high_score.txt
    f = open('high_score.txt', 'w')
    f.write(str(stats.high_score))
    f.close()
```

```
    if event.type == pygame.QUIT:
        # 14-4 implementation
        # Write the high score to file before exit
        write_high_score(stats)
        sys.exit()
```

```
    elif event.key == pygame.K_q:
        # 14-4 implementation
        # Exit the screen with Q is press
        # Write the high score to file before exit
        write_high_score(stats)
        sys.exit()
```

14-5 implement in Alien Invasion folder

I refactored the program codes by moving the four method calls from Scoreboard constructor to make a new method prep_images(). I moved part of check_play_button() to create start_game() method. I also split some part of check_bullet_alien_collisions() to create start_new_level(). These new methods made the code implementation easier and cleaner. For example, with prep_image() start_game() only need to call one method to reset the score board, instead of four.

```
    # 14-5 implementation
    def prep_images(self):
        # Prepare the initial score board image
        self.prep_score()
        self.prep_high_score()
        self.prep_level()
        self.prep_ships()
```

```
# 14-1 $ 14-5 implementation
def start_game(ai_settings, screen, stats, sb, ship, aliens, bullets):
    # Start a new game with the method is call
    # Check if the game is off
    if not stats.game_active:
        # Reset the game speed settings
        ai_settings.initialize_dynamic_settings()

        # Hide the mouse cursor.
        pygame.mouse.set_visible(False)

        # Reset the game statistics.
        stats.reset_stats()
        stats.game_active = True

        # 14-5 implementation
        # Reset the scoreboard images.
        sb.prep_images()

        # Empty the list of aliens and bullets.
        aliens.empty()
        bullets.empty()
```

```
    # 14-5 implementation
    # If the entire fleet is destroyed, start a new level.
    if len(aliens) == 0:
        start_new_level(ai_settings, screen, stats, sb, ship, aliens, bullets)

# 14-5 implementation
def start_new_level(ai_settings, screen, stats, sb, ship, aliens, bullets):
        # Destroy existing bullets, speed up game
        bullets.empty()
        ai_settings.increase_speed()

        # Increase level
        stats.level += 1
        sb.prep_level()

        # Create new fleet
        create_fleet(ai_settings, screen, ship, aliens)
```

14-6 implement in Alien Invasion folder

I added shooting and explosion sound effect by using pygame.mixer. I set the sound effect variable in Settings class, and called shooting_sound whenever a bullet is shot, called explosion_sound whenever the bullet hit the alien or the ship hit the alien.

```
        # 14-6 implementation Set sound effects
        self.shooting_sound = pygame.mixer.Sound('sound/Shooting.wav')
        self.explosion_sound = pygame.mixer.Sound('sound/Explosion.wav')
```

```
    if collisions:
        # 14-6 implement sound effect for explosion when the aliens is hit
        ai_settings.explosion_sound.play()
        for aliens in collisions.values():
            stats.score += ai_settings.alien_points * len(aliens)
            sb.prep_score()
        check_high_score(stats, sb)
```

```
def ship_hit(ai_settings, screen, stats, sb, ship, aliens, bullets):
    # 14-6 implement sound effect for explosion when the ship is hit
    ai_settings.explosion_sound.play()
```