Resources used Python Crash Course, 2nd Edition by Eric Matthes

Click Play to start the game

Press the left and right arrows to move your ship and press the up arrow to fire bullets.


Highest: 0
Level: 0
Level: 1

Game.py

```python
import pygame
import sys
from settings import Settings
from ship import Ship
from pygame.sprite import Group
from bullet import Bullet
from alien import Alien
from stats import GameStats
from button import Button
from scoreboard import Scoreboard
import random
import time

DROP_ALIEN_EVENT = pygame.USEREVENT + 1


class SpaceInvaders:
    def __init__(self):
        pygame.init()
        self.settings = Settings()
```

```
        original_image = pygame.image.load('images/earth.jpg')
        self.bg_image = pygame.transform.scale(original_image,
(self.settings.screen_width, self.settings.screen_height))
        pygame.time.set_timer(DROP_ALIEN_EVENT, 5000)


        self.screen = pygame.display.set_mode((0, 0),
pygame.FULLSCREEN)
        self.settings.screen_width = self.screen.get_rect().width
        self.settings.screen_height = self.screen.get_rect().height
        self.bg_image = pygame.image.load('images/earth.jpg')
        self.bg_image = pygame.transform.scale(self.bg_image,
(self.settings.screen_width, self.settings.screen_height))

        self.stats = GameStats(self)
        self.ship = Ship(self)
        self.bullets = Group()
        self.aliens = Group()
        self._create_fleet()
        self.score = self.stats.score
        self.health = self.ship.health
```

```
        self.font = pygame.font.Font(None, 72)  # Choose the font for
the text
        self.stage = self.stats.level
        self.stage_start_time = pygame.time.get_ticks()
        self.scoreboard = Scoreboard(self)

        self.play_button = Button(self, "Play")

    def _start_game(self):
        """Start a new game."""
        # Reset game statistics.
        self.stats.reset_stats()
        self.stats.game_active = True

        # Reset the game settings.
        self.settings.initialize_dynamic_settings()
        self.settings.increase_speed()

        # Reset the ship's health.
        self.ship.health = self.ship.max_health

        # Empty the list of bullets and aliens.
```

```
        self.bullets.empty()
        self.aliens.empty()

        # Create a new fleet and center the ship.
        self._create_fleet()
        self.ship.center_ship()

        self.scoreboard.prep_level()
        self.scoreboard.prep_score()

        # Reset the play button's label.
        self.play_button.msg = "Play"
        self.play_button._prep_msg(self.play_button.msg)


    def _reset_game(self):
        """Reset the game state to start a new game."""
        # Reset the game statistics
        self.stats.reset_stats()
        self.stats.game_active = True

        # Reset the game objects
```

```
        self.aliens.empty()
        self.bullets.empty()

        # Create a new fleet and center the ship
        self._create_fleet()
        self.ship.center_ship()

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

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

    def run_game(self):
        while True:
            self._check_events()
            if self.stats.game_active:
                self.ship.update()
                self._update_bullets()
```

```
                self._update_aliens()
            self._update_screen()

            if not self.aliens:
                    self.bullets.empty()
                    self._create_fleet()
                    self.settings.increase_speed()
                    self.font = pygame.font.Font(None, 72)
                    self.stage += 1  # Increment the stage
                    self.stage_start_time = pygame.time.get_ticks()


            collisions = pygame.sprite.groupcollide(self.bullets,
self.aliens, True, True)
            if collisions:
                self.score += len(collisions)  # Grant 1 point for each
alien destroyed

                # Repopulate the fleet if all aliens have been
destroyed
            if not self.aliens:
                self.bullets.empty()
```

```
                self._create_fleet()
                self.settings.bullet_speed *= 1.1  # Speed up bullets

                 # Check if any aliens have reached the bottom of the
screen
            for alien in self.aliens.sprites():
                if alien.rect.bottom >= self.screen.get_rect().bottom:
                    self.health -= 1  # Damage the player
                    break

            self.scoreboard.prep_level()
            self.scoreboard.show_score()

    def _create_fleet(self):
        """create the fleet of aliens"""
        #create an alien and keep adding aliens until there no room
left.
        #space between aliens is one aliens width
        alien = Alien(self)
        alien_width, alien_height = alien.rect.size

        current_x, current_y = alien_width, alien_height
```

```
        for _ in range(4):  # Limit the number of rows to 4
            while current_x < (self.settings.screen_width - 2 *
alien_width):
                self._create_alien(current_x, current_y)
                current_x += 2 * alien_width
            #finished a row; reset x value increment y value
            current_x = alien_width
            current_y += 2 * alien_height
        self.aliens.add(alien)

        self.settings.alien_speed *= 1.1

        random_alien = random.choice([alien for alien in self.aliens])
        random_alien.move_downwards_only = True

        self.stats.increment_level()


    def _create_alien(self, x_position, y_position):
        """create an alien and place it on the row"""
        new_alien = Alien(self)
        new_alien.x = x_position
```

```
        new_alien.rect.x = x_position
        new_alien.rect.y = y_position
        self.aliens.add(new_alien)

    def _update_aliens(self):
        """Update the positions of all aliens in the fleet."""
        self._check_fleet_edges()
        self.aliens.update()

    def _check_fleet_edges(self):
        """Respond appropriately if any aliens have reached an edge."""
        for alien in self.aliens.sprites():
            if alien.check_edges():
             self._change_fleet_direction()
             break

    def _change_fleet_direction(self):
        """Drop the entire fleet and change the fleet's direction."""
        for alien in self.aliens.sprites():
            alien.rect.y += self.settings.fleet_drop_speed
        self.settings.fleet_direction *= -1
```

```
    def _check_events(self):
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                pygame.quit()
                sys.exit()
            elif event.type == pygame.MOUSEBUTTONDOWN:
                mouse_pos = pygame.mouse.get_pos()
                if self.play_button.rect.collidepoint(mouse_pos):
                    self._start_game()
                    self.play_button.msg = "Play"
                    self.play_button._prep_msg(self.play_button.msg)
                elif event.type == pygame.MOUSEBUTTONDOWN:
                    mouse_pos = pygame.mouse.get_pos()
                    self._check_play_button(mouse_pos)
            elif event.type == DROP_ALIEN_EVENT:
                # Select a random alien and make it start moving
downwards.
                random_alien = random.choice(self.aliens.sprites())
                random_alien.start_moving_downwards()
            elif event.type == pygame.KEYDOWN:
                if event.key == pygame.K_q:
                    pygame.quit()
```

```
                sys.exit()
            elif event.key == pygame.K_RIGHT:
                self.ship.moving_right = True
            elif event.key == pygame.K_LEFT:
                self.ship.moving_left = True
        elif event.type == pygame.KEYUP:
            if event.key == pygame.K_RIGHT:
                self.ship.moving_right = False
            elif event.key == pygame.K_LEFT:
                self.ship.moving_left = False
            elif event.key == pygame.K_UP:
                self._fire_bullet()

    def _check_play_button(self, mouse_pos):
        """Start a new game when the player clicks Play."""
        button_clicked = self.play_button.rect.collidepoint(mouse_pos)
        if button_clicked and not self.stats.game_active:
            # Reset the game settings.
            self.settings.initialize_dynamic_settings()

            # Reset the game statistics.
            self.stats.reset_stats()
```

```
            self.stats.game_active = True

            # Reset the scoreboard images.
            self.sb.prep_score()
            self.sb.prep_level()

    def _drop_random_alien(self):
        """Select a random alien and make it start moving downwards."""
        if self.aliens:  # Check if there are any aliens left
            random_alien = random.choice(self.aliens.sprites())
            random_alien.start_moving_downwards()

    def _draw_health_bar(self):
        """Draw the health bar indicating the player's life points."""
        if self.stats.ships_left > 0:
            pygame.draw.rect(self.screen, (0, 255, 0), (10,
self.settings.screen_height - 20, 100 * (self.stats.ships_left / 3),
10))
            pygame.draw.rect(self.screen, (255, 0, 0), (10 + 100 *
(self.stats.ships_left / 3), self.settings.screen_height - 20, 100 * (1
- self.stats.ships_left / 3), 10))
```

```
    def _fire_bullet(self):
        """Create a new bullet and add it to the bullets group."""
        if len(self.bullets) < 3:
            new_bullet = Bullet(self)
            self.bullets.add(new_bullet)

    def _update_bullets(self):
        """Update the positions of bullets and get rid of old
bullets."""
        # Update bullet positions.
        self.bullets.update()

        # Get rid of bullets that have disappeared.
        for bullet in self.bullets.copy():
            if bullet.rect.bottom <= 0:
                self.bullets.remove(bullet)

    def _ship_hit(self):
        # Look for alien-ship collisions.
        alien_hit = pygame.sprite.spritecollideany(self.ship,
self.aliens)
```

```
        if alien_hit:
            # Remove the alien that hit the ship.
            self.aliens.remove(alien_hit)

        if self.stats.ships_left > 0:
            # Decrement ships_left.
            self.stats.ships_left -= 1

            self.bullets.empty()

            # Pause.
            time.sleep(0.5)
        else:
            self.stats.game_active = False
            pygame.mouse.set_visible(True)

    def _check_bullet_alien_collisions(self):
        """Respond to bullet-alien collisions."""
        # Remove any bullets and aliens that have collided.
        collisions = pygame.sprite.groupcollide(self.bullets,
self.aliens, True, True)
```

```
        if collisions:
            for aliens in collisions.values():
                self.stats.score += (self.settings.alien_points *
len(aliens))
            self.scoreboard.prep_score()
            self.scoreboard.check_high_score()

        if not self.aliens:
            # Destroy existing bullets and create new fleet.
            self.bullets.empty()
            self._create_fleet()
            self.settings.increase_speed()

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

    def _update_aliens(self):
        self._check_fleet_edges()
        self.aliens.update()

    # Look for alien-ship collisions.
```

```
        if pygame.sprite.spritecollideany(self.ship, self.aliens):
            self._ship_hit()

    # Look for aliens hitting the bottom of the screen.
        self._check_aliens_bottom()

        if self.health <= 1:
            self._game_over()

    def _check_aliens_bottom(self):
        """Check if any aliens have reached the bottom of the
screen."""
        screen_rect = self.screen.get_rect()
        for alien in self.aliens.sprites():
            if alien.rect.bottom >= screen_rect.bottom:
                # Treat this the same as if the ship got hit.
                self.aliens.remove(alien)
                self._ship_hit()
                break

    # Repopulate the fleet if all aliens have been destroyed
        if not self.aliens:
```

```
            self.bullets.empty()
            self._create_fleet()
            self.settings.increase_speed()

            # Increase stage
            self.stage += 1
            self.stage_start_time = pygame.time.get_ticks()

    def _update_screen(self):
        """Update images on the screen, and flip to the new screen."""
        # Redraw the screen during each pass through the loop.
        self.screen.blit(self.bg_image, (0, 0))

        self.ship.blitme()
        self.bullets.draw(self.screen)
        self.aliens.draw(self.screen)

        # Draw the score information.
        self.scoreboard.show_score()

        # Draw the play button if the game is inactive.
        if not self.stats.game_active:
```

```
            self.play_button.draw_button()

        # Make the most recently drawn screen visible.
        pygame.display.flip()

    def _game_over(self):
        """End the current game and show the game over screen."""
        self.stats.game_active = False
        self.health = 1
        pygame.mouse.set_visible(True)
        self.play_button.msg = "Restart"
        self.play_button._prep_msg(self.play_button.msg)

        # Draw the game over screen
        self.screen.fill((0, 0, 0))
        background = pygame.image.load('images/earth.jpg')
        self.screen.blit(background, (0, 0))
        game_over_text = self.font.render("GAME OVER", True, (255, 255,
255))
        self.screen.blit(game_over_text, (self.settings.screen_width /
2 - game_over_text.get_width() / 2, self.settings.screen_height / 2 -
game_over_text.get_height() / 2))
```

```
        restart_text = self.font.render("Press Enter to restart", True,
(255, 255, 255))
        self.screen.blit(restart_text, (self.settings.screen_width / 2
- restart_text.get_width() / 2, self.settings.screen_height / 2 -
restart_text.get_height() / 2 + 100))  # Increase the value added to
adjust the vertical position

        # Draw the high score
        high_score_text = self.font.render(f"High Score:
{self.stats.high_score}", True, (255, 255, 255))
        self.screen.blit(high_score_text, (self.settings.screen_width /
2 - high_score_text.get_width() / 2, self.settings.screen_height / 2 -
high_score_text.get_height() / 2 + 150))  # Increase the value added to
adjust the vertical position

        pygame.display.flip()

        # Wait for the player to press the Enter key
        while True:
            for event in pygame.event.get():
                if event.type == pygame.QUIT:
```

```
                    pygame.quit()
                    sys.exit()
                elif event.type == pygame.KEYDOWN:
                    if event.key == pygame.K_RETURN:  # The Enter key
was pressed
                        self._reset_game()  # Reset the game state
                        return True

if __name__ == "__main__":
    SpaceInvaders().run_game()
```

Alien.py

```
import pygame
from pygame.sprite import Sprite

class Alien(Sprite):
    """A class to represent a single alien in the fleet."""

    def __init__(self, ai_game):
        """Initialize the alien and set its starting position."""
        super().__init__()
```

```
        self.screen = ai_game.screen
        self.settings = ai_game.settings

        # Load the alien image and set its rect attribute.
        self.image = pygame.image.load('images/alien.png')
        self.image = pygame.transform.scale(self.image, (75, 75))
        self.rect = self.image.get_rect()

        # Start each new alien near the top left of the screen.
        self.rect.x = self.rect.width
        self.rect.y = self.rect.height

        # Store the alien's exact horizontal position.
        self.x = float(self.rect.x)

        self.y = float(self.rect.y)

        self.moving_down = False

    def start_moving_downwards(self):
        """Make the alien start moving downwards."""
        self.moving_down = True
```

```
        self.y += self.settings.alien_speed
        self.y = self.rect.y # Change this to control the speed of the
alien

    def check_edges(self):
        """Return True if alien is at edge of screen."""
        screen_rect = self.screen.get_rect()
        if self.rect.right >= screen_rect.right or self.rect.left <= 0:
            return True

    def update(self):
        """Move the alien right or left."""
        self.x += (self.settings.alien_speed *
self.settings.fleet_direction)
        self.rect.x = self.x

        if self.moving_down:
            self.y += self.settings.alien_speed
            self.rect.y = self.y
```

bullet.py

```python
import pygame
from pygame.sprite import Sprite

class Bullet(Sprite):
    """A class to manage the bullets fired from the ship"""

    def __init__(self, ai_game):
        """Create a bullet object at the ship's current position."""
        super().__init__()
        self.screen = ai_game.screen
        self.settings = ai_game.settings
        self.color = self.settings.bullet_color

        # Load the bullet image and get its rect.
        self.image = pygame.image.load('images/bullet.png')
        self.image = pygame.transform.scale(self.image, (40, 40))
        self.rect = self.image.get_rect()

        # Start each new bullet at the top center of the ship.
        self.rect.midtop = ai_game.ship.rect.midtop

        # Store the bullet's position as a decimal value.
```

```
        self.y = float(self.rect.y)

    def update(self):
        """Move the bullet up the screen."""
        # Update the decimal position of the bullet.
        self.y -= self.settings.bullet_speed
        # Update the rect position.
        self.rect.y = self.y

    def blitme(self):
        """Draw the bullet to the screen."""
        self.screen.blit(self.image, self.rect)
```

button.py

```
import pygame

class Button:
    def __init__(self, ai_game, msg):
        """Initialize button attributes."""
        self.screen = ai_game.screen
```

```
        self.screen_rect = self.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.msg = msg
        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

    def draw_button(self):
        # Draw blank button and then draw message.
        self.screen.fill(self.button_color, self.rect)
        self.screen.blit(self.msg_image, self.msg_image_rect)
```

scoreboard.py

```
import pygame.font
from pygame.sprite import Group
from ship import Ship
from settings import Settings


class Scoreboard:
    """A class to display the player's score."""

    def __init__(self, ai_game):
        """Initialize score-keeping attributes."""
        self.ai_game = ai_game
        self.screen = ai_game.screen
        self.screen_rect = self.screen.get_rect()
```

```
        self.settings = ai_game.settings
        self.stats = ai_game.stats

        self.ship_image = pygame.image.load('images/ship.png')
        self.ship_image = pygame.transform.scale(self.ship_image, (20,
20))

        # Font settings for score display
        self.text_color = (0, 0, 0)   # white
        self.font = pygame.font.SysFont(None, 36)

        # Prepare the initial score display
        self.prep_score()

        # Prepare high score display
        self.prep_high_score()

        # Prepare level display
        self.prep_level()


    def prep_score(self):
```

```
        """Turn the score into a rendered image."""
        score_str = f"Level: {str(self.stats.score)}"
        self.score_image = self.font.render(score_str, True,
self.text_color, self.settings.background_color)

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

    def prep_high_score(self):
        """Turn the high score into an image."""
        self.stats.high_score = round(self.stats.high_score, -1)
        high_score = round(self.stats.high_score, -1)
        high_score_str = f"Highest: {high_score:,}"
        self.high_score_image = self.font.render(high_score_str, True,
self.text_color,
                                                 self.settings.backgrou
nd_color)

        # Puts the high score at the top center of the screen
        self.high_score_rect = self.high_score_image.get_rect()
```

```
        self.high_score_rect.centerx = self.screen_rect.centerx  #
aligns high score to center screen
        self.high_score_rect.top = self.score_rect.top  # aligns high
score to top of screen

    def check_high_score(self):
        """If the current score is higher than the high score, update
the high score."""
        if self.stats.score > self.stats.high_score:
            self.stats.high_score = self.stats.score
            self.stats.save_high_score()
            self.prep_high_score()

    def show_score(self):
        """Draw the score on the screen."""
        self.screen.blit(self.score_image, self.score_rect)  # draw
current score
        self.screen.blit(self.high_score_image,
self.high_score_rect)  # draw high score
        self.screen.blit(self.level_image, self.level_rect)
        self.show_health()
```

```
    def prep_level(self):
        """Turn the current level the player is on into an image."""
        level_str = f"Level: {str(self.stats.level)}"
        self.level_image = self.font.render(level_str, True,
self.text_color, self.settings.background_color)

        # Position the level below the current score
        self.level_rect = self.level_image.get_rect()
        self.level_rect.right = self.score_rect.right  # aligns level
to the right of the score
        self.level_rect.top = self.score_rect.bottom + 10   # 10 pixels
below the score

    def show_health(self):
        """Draw health points as ship sprites on the screen."""
        for i in range(self.stats.ships_left):
            x_position = 10 + i * (self.ship_image.get_width() + 10)  #
10px margin between each sprite
            self.screen.blit(self.ship_image, (x_position, 10))  # 10px
from the top of the screen

    def show_level(self):
```

```
        """Draw the level on screen."""
        self.screen.blit(self.level_image, self.level_rect)
```

settings.py

```
class Settings:
    """A class to store all settings for Space Invaders."""

    def __init__(self):
        """Initialize the game's settings."""
        # Screen settings
        self.screen_width = 800
        self.screen_height = 600
        self.background_color = (230, 230, 230)
        self.ship_speed = 2.0
        self.bullet_speed = 5.0  # Adjust as needed
        self.bullet_color = (60, 60, 60)  # Adjust as needed
        self.bullets_allowed = 3  # Adjust as needed
        self.alien_speed = 1.0
        self.fleet_drop_speed = 20
```

```
        self.fleet_direction = 1  # 1 represents right; -1 represents
left
        self.ship_limit = 3
        self.speedup_scale = 1.05
        self.score_scale = 1.5
        self.alien_points = 50
        self.initialize_dynamic_settings()

    def initialize_dynamic_settings(self):
        """Initialize settings that can change during the game."""
        self.ship_speed = 1.5
        self.bullet_speed = 3.0
        self.alien_speed = 1.0



    def increase_speed(self):
        """Increase speed settings and alien point values."""
        self.ship_speed *= self.speedup_scale
        self.bullet_speed *= self.speedup_scale
        self.alien_speed *= self.speedup_scale
```

```
        self.alien_points = int(self.alien_points * self.speedup_scale)
```

ship.py

```
import pygame

class Ship:
    """A class to manage the ship."""

    def __init__(self, ai_game):
        """Initialize the ship and set its starting position."""
        self.screen = ai_game.screen
        self.settings = ai_game.settings
        self.screen_rect = ai_game.screen.get_rect()

        # Load the ship image and get its rect.
        self.image = pygame.image.load('images/ship.png')
        self.image = pygame.transform.scale(self.image, (75, 75))
        self.rect = self.image.get_rect()

        # Start each new ship at the bottom center of the screen.
        self.rect.midbottom = self.screen_rect.midbottom
```

```
        # Movement flags
        self.moving_right = False
        self.moving_left = False

        self.max_health = 3
        self.health = self.max_health

    def update(self):
        """Update the ship's position based on the movement flags."""
        if self.moving_right:
             self.rect.x += self.settings.ship_speed
        if self.moving_left:
             self.rect.x -= self.settings.ship_speed
    # If moving right and ship's right edge is less than the screen's
right edge
        if self.moving_right and self.rect.right <
self.screen_rect.right:
            self.rect.x += self.settings.ship_speed
    # If moving left and ship's left edge is greater than the screen's
left edge
        if self.moving_left and self.rect.left > 0:
            self.rect.x -= self.settings.ship_speed
```

```
    def blitme(self):
        """Draw the ship at its current location."""
        self.screen.blit(self.image, self.rect)

    def center_ship(self):
        """Center the ship on the screen."""
        self.rect.midbottom = self.screen_rect.midbottom
        self.x = float(self.rect.x)
```

stats.py

```
class GameStats:
    """Track statistics for Space Invaders."""

    def __init__(self, ai_game):
        """Initialize statistics."""
        self.settings = ai_game.settings
        self.level = 0
        self.score = 0
        self.reset_stats()

        # Start Space Invaders in an inactive state.
```

```
        self.game_active = False

        self.high_score = 0

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

    def increment_level(self):
        """Increase the level by 1."""
        self.level += 1
```