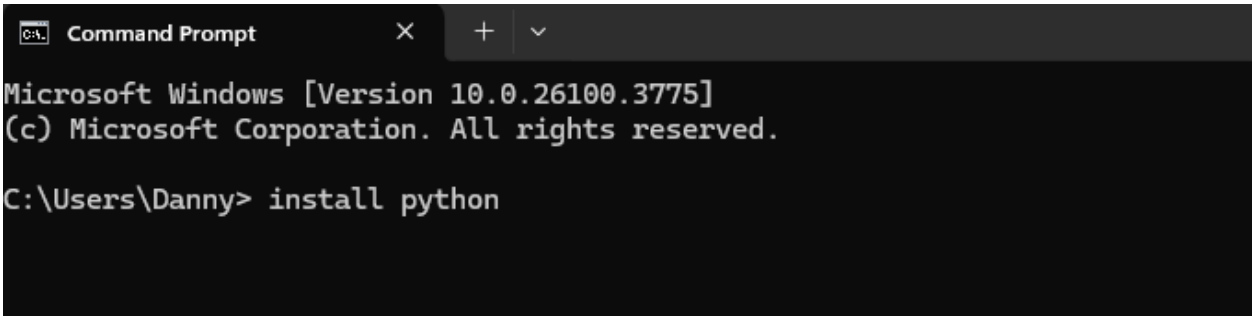
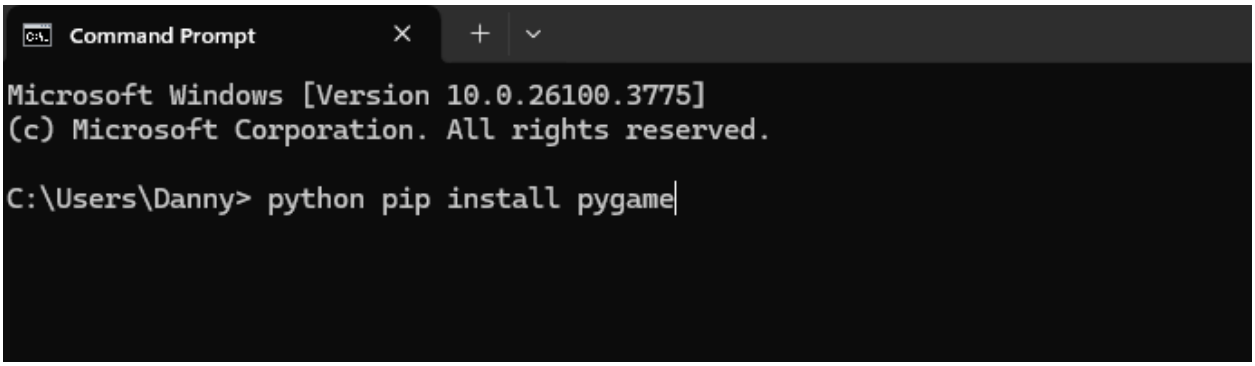
Alien Invasion Python game

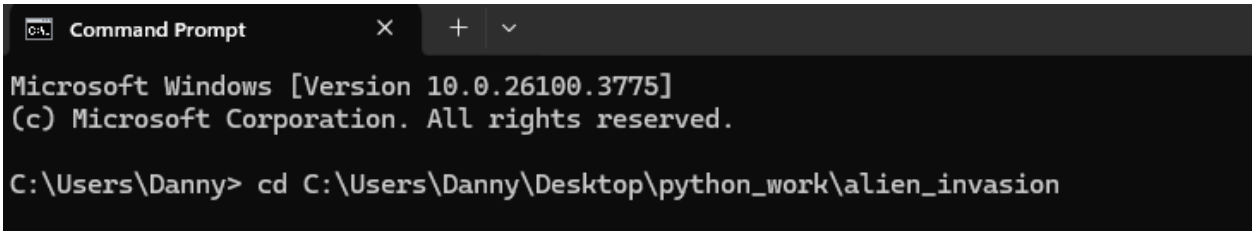
How to install and run:

1. Search for Command Prompt in apps on your device and open the application.
2. Install Python (version 3.11+ recommended) and pygame:

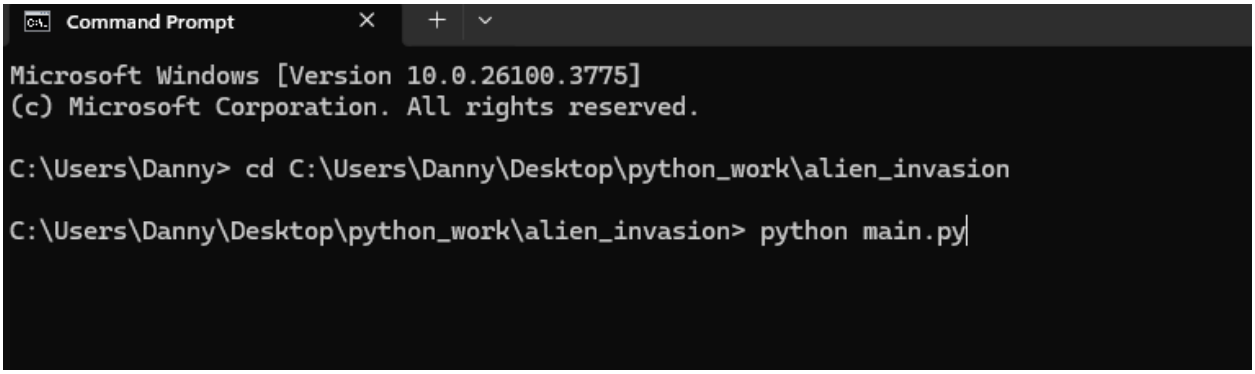




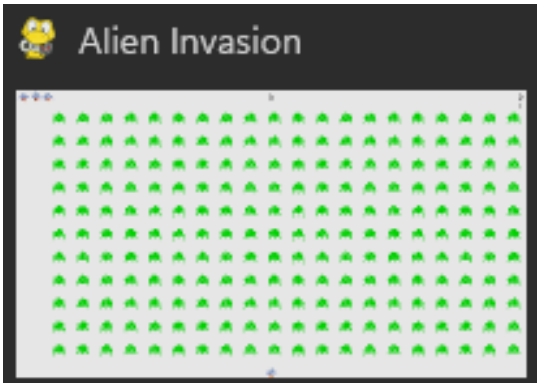
3. Download or clone the project folder from GitHub and keep track of where you are saving it on your device
4. Enter the path for the alien_invasion folder you downloaded from GitHub in the terminal:



5. Run the game from terminal by entering “python main.py”:



6. Play the game!



How to play:

Controls:

- Use right arrow → move ship right
- Use left arrow → move ship left
- Spacebar → shoot bullets at aliens
- Q → quit game
- Click Play button to start
- Avoid aliens reaching the bottom or hitting your ship

Resources:

- Textbook: *Python Crash Course* by Eric Matthes, 2nd Edition
- Pygame documentation: https://www.pygame.org/docs
- ChatGPT (assistant for debugging and polishing code)

Code Printout:

main.py:

```
import sys
from time import sleep

import pygame

from settings import Settings
from game_stats import GameStats
from scoreboard import Scoreboard
from button import Button
from ship import Ship
from bullet import Bullet
from alien import Alien

class AlienInvasion:
  """Overall class to manage fame assets and behavior."""

  def __init__(self):
    """Initialize the game, and create game resources."""
    pygame.init()
    self.clock = pygame.time.Clock()
    self.settings = Settings()

    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
    pygame.display.set_caption("Alien Invasion")

    # Create an instance to store game statistics,
    # and create a scoreboard.
    self.stats = GameStats(self)
    self.sb = Scoreboard(self)

    self.ship = Ship(self)
    self.bullets = pygame.sprite.Group()
    self.aliens = pygame.sprite.Group()

    self._create_fleet()

    # Start Alien Invasion in an active state.
    self.game_active = False

    # Make the Play button.
```

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

  def run_game(self):
    """Start the main loop for the game."""
    while True:
      self._check_events()

      if self.game_active:
        self.ship.update()
        self._update_bullets()
        self._update_aliens()

      self._update_screen()
      self.clock.tick(60)

  def _check_events(self):
    """Respond to keypresses and mouse events."""
    for event in pygame.event.get():
      if event.type == pygame.QUIT:
        sys.exit()
      elif event.type == pygame.KEYDOWN:
        self._check_keydown_events(event)
      elif event.type == pygame.KEYUP:
        self._check_keyup_events(event)
      elif event.type == pygame.MOUSEBUTTONDOWN:
        mouse_pos = pygame.mouse.get_pos()
        self._check_play_button(mouse_pos)

  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.game_active:
      # Reset the game settings.
      self.settings.initialize_dynamic_settings()

      # Reset the game statistics.
      self.stats.reset_stats()
      self.sb.prep_score()
      self.sb.prep_level()
      self.sb.prep_ships()
      self.game_active = True

      # Get rid of any remaining bullets and aliens.
      self.bullets.empty()
      self.aliens.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 _check_keydown_events(self, event):
        """Respond to keypresses"""
        if event.key == pygame.K_RIGHT:
            self.ship.moving_right = True
        elif event.key == pygame.K_LEFT:
            self.ship.moving_left = True
        elif event.key == pygame.K_q:
            sys.exit()
        elif event.key == pygame.K_SPACE:
            self._fire_bullet()

    def _check_keyup_events(self, event):
        """Respond to key releases."""
        if event.key == pygame.K_RIGHT:
            self.ship.moving_right = False
        elif event.key == pygame.K_LEFT:
            self.ship.moving_left = False

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

    def _update_bullets(self):
        """Update position 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)

        self._check_bullet_alien_collisions()

    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.sb.prep_score()
      self.sb.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 _create_fleet(self):
    """Create the fleet of aliens."""
    # Create an alien and keep adding aliens until there's no room left.
    # Spacing between aliens is one alien width and one alien height.
    alien = Alien(self)
    alien_width, alien_height = alien.rect.size

    current_x, current_y = alien_width, alien_height
    while current_y < (self.settings.screen_height - 3 * alien_height):
      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, and increment y value.
      current_x = alien_width
      current_y += 2 * alien_height

  def _create_alien(self, x_position, y_position):
    """Create an alien and place it in the fleet."""
    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):
  """Check if the fleet is at an edge, then update positions."""
  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()

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 _ship_hit(self):
  """Respond to the ship being hit by an alien."""
  if self.stats.ships_left > 0:
    # Decrement ships_left, and update scoreboard.
    self.stats.ships_left -= 1
    self.sb.prep_ships()

    # Get rid of any remaining bullets and aliens.
    self.bullets.empty()
    self.aliens.empty()

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

    # Pause.
    sleep(0.5)
  else:
    self.game_active = False
```

```
      pygame.mouse.set_visible(True)

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

  def _update_screen(self):
    """Update images on the screen, and flip to the new screen."""
    self.screen.fill(self.settings.bg_color)
    for bullet in self.bullets.sprites():
      bullet.draw_bullet()
    self.ship.blitme()
    self.aliens.draw(self.screen)

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

    # Draw the play button if the game is inactive.
    if not self.game_active:
      self.play_button.draw_button()

    pygame.display.flip()


if __name__ == '__main__':
  #Make a game instance, and run the game.
  ai = AlienInvasion()
  ai.run_game()
```

settings.py:

```
class Settings:
  """A class to store all settings for Alien Invasion."""

  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)
```

```
    # Ship settings
    self.ship_limit = 3

    # Bullet settings
    self.bullet_width = 3000
    self.bullet_height = 15
    self.bullet_color = (60, 60, 60)
    self.bullets_allowed = 3

    # Alien settings
    self.fleet_drop_speed = 5

    # How quickly the game speeds up.
    self.speedup_scale = 1.1
    # How quickly the alien point values increase
    self.score_scale = 1.5

    self.initialize_dynamic_settings()

  def initialize_dynamic_settings(self):
    """Initialize settings that change throughout the game."""
    self.ship_speed = 5
    self.bullet_speed = 15
    self.alien_speed = 1.0

    # fleet_direction of 5 represents right; -5 represents left.
    self.fleet_direction = 5

    # Scoring settings
    self.alien_points = 50

  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.score_scale)
```

bullet.py:

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

```
class Bullet(Sprite):
  """A class to manage 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

    # Create a bullet rect at (0, 0) and then set correct position.
    self.rect = pygame.Rect(0, 0, self.settings.bullet_width,
      self.settings.bullet_height)
    self.rect.midtop = ai_game.ship.rect.midtop

    # Store the bullet's position as a float.
    self.y = float(self.rect.y)

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

  def draw_bullet(self):
    """Draw the bullet to the screen."""
    pygame.draw.rect(self.screen, self.color, self.rect)
```

ship.py:

```
import pygame
from pygame.sprite import Sprite

class Ship(Sprite):
  """A class to manage the ship"""

  def __init__(self, ai_game):
    """Initialize the ship and set its starting position."""
    super().__init__()
    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.bmp')
    self.rect = self.image.get_rect()

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

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

    # Movement flag; start with a ship that's not moving.
    self.moving_right = False
    self.moving_left = False

  def update(self):
    """Update the ship's position based on the movement flag."""
    # Update the ship's x value, not the rect.
    if self.moving_right and self.rect.right < self.screen_rect.right:
      self.x += self.settings.ship_speed
    if self.moving_left and self.rect.left > 0:
      self.x -= self.settings.ship_speed

    # Update rect object from self.x.
    self.rect.x = self.x

  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)
```

button.py:

```
import pygame.font

class Button:
  """A class to build buttons for the game."""

  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, 135, 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

  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)
```

game_stats.py:

```
class GameStats:
  """Track statistics for Alien Invasion."""

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

    # High score should never be reset.
    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 = 1
```

scoreboard.py:

```
import pygame.font
from pygame.sprite import Group

from ship import Ship

class Scoreboard:
  """A class to report scoring information."""

  def __init__(self, ai_game):
    """Initialize scorekeeping 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

    # Font settings for scoring information.
    self.text_color = (30, 30, 30)
    self.font = pygame.font.SysFont(None, 48)

    # Prepare the initial score image.
    self.prep_score()
    self.prep_high_score()
    self.prep_level()
    self.prep_ships()

  def prep_score(self):
    """Turn the score into a rendered image."""
    rounded_score = round(self.stats.score, -1)
    score_str = f"{rounded_score:,}"
    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.
    self.score_rect = self.score_image.get_rect()
    self.score_rect.right = self.screen_rect.right - 20
    self.score_rect.top = 20

  def show_score(self):
    """Draw scores and level, and ships to the screen."""
    self.screen.blit(self.score_image, self.score_rect)
    self.screen.blit(self.high_score_image, self.high_score_rect)
    self.screen.blit(self.level_image, self.level_rect)
    self.ships.draw(self.screen)
```

```
  def prep_high_score(self):
    """Turn the high score into a rendered image."""
    high_score = round(self.stats.high_score, -1)
    high_score_str = f"{high_score:,}"
    self.high_score_image = self.font.render(high_score_str, True,
      self.text_color, self.settings.bg_color)

    # Center the high score at the top of the screen.
    self.high_score_rect = self.high_score_image.get_rect()
    self.high_score_rect.centerx = self.screen_rect.centerx
    self.high_score_rect.top = self.score_rect.top

  def check_high_score(self):
    """Check to see if there's a new high score."""
    if self.stats.score > self.stats.high_score:
      self.stats.high_score = self.stats.score
      self.prep_high_score()

  def prep_level(self):
    """Turn the level into a rendered image."""
    level_str = str(self.stats.level)
    self.level_image = self.font.render(level_str, True,
      self.text_color, self.settings.bg_color)

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

  def prep_ships(self):
    """Show how many ships are left."""
    self.ships = Group()
    for ship_number in range(self.stats.ships_left):
      ship = Ship(self.ai_game)
      ship.rect.x = 10 + ship_number * ship.rect.width
      ship.rect.y = 10
      self.ships.add(ship)
```

alien.py:

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

```python
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.bmp')
    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)

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

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