**Project Summary: Alien Invasion Game**

**1. What Is This Game?**

This project is a digital version of a classic 2D arcade shooter like *Space Invaders*, built using the Pygame library in Python. The gameplay is simple and intuitive:

* The player controls a spaceship at the bottom of the screen.
* A fleet of aliens slowly moves down from the top.
* The player can move left and right and shoot bullets to destroy the aliens.
* The goal is to score as many points as possible before running out of lives.

The project is organized into separate files for each major component of the game (the ship, the aliens, game settings, etc.), which makes the code clean and easy to manage.

**2. The Classes: Meet the Team**

Each class in the project acts like a different department or actor in a movie, with each one having a specific job to do.

**AlienInvasion (from alien\_invasion.py) - The Game Director 🎬**

This is the most important class and acts as the central controller for the entire game. It tells all the other classes what to do and when to do it.

* **Usage**: This is the main file you run to start the game.
* **Important Methods**:
  + **\_\_init\_\_()**: The setup method. It runs once at the start to create the game window, load settings, and build initial objects like the player's ship.
  + **run\_game()**: The main game loop. This is the engine that runs continuously, checking for player actions, updating object positions, and redrawing the screen in a constant cycle.
  + **\_check\_events()**: The listener. Its only job is to watch for player input, such as key presses (move, shoot) or mouse clicks (start game).
  + **\_update\_screen()**: The artist. After all objects have moved, this method erases the old screen and draws everything in its new position, creating the illusion of movement.

**Settings (from settings.py) - The Rulebook 📜**

This class doesn't control any characters but holds all the game's rules and settings in one convenient place.

* **Usage**: A central location for values like speed and color. To make the game faster or change the background, you only need to make edits in this file.
* **Key Properties**:
  + **screen\_width, screen\_height, bg\_color**: Defines the look of the game window.
  + **ship\_speed, ship\_limit**: Controls the player ship's speed and how many lives the player gets.
  + **bullet\_speed, bullet\_color**: Defines the properties of the bullets.
  + **alien\_speed**: Controls how fast the aliens move.

**Ship (from ship.py) - The Hero 🚀**

This class manages everything related to the player's spaceship.

* **Usage**: Handles drawing the ship on the screen and moving it based on the player's key presses.
* **Important Methods**:
  + **\_\_init\_\_()**: Loads the ship's image and places it at its starting position.
  + **update()**: Called every frame to move the ship if the left or right arrow key is being held down.
  + **blitme()**: A simple method that means "draw me," which draws the ship's image onto the screen.

**Alien (from alien.py) - The Bad Guy 👽**

This class is a blueprint for a single alien, which the game uses to create an entire fleet.

* **Usage**: Defines how an alien looks and moves.
* **Important Methods**:
  + **\_\_init\_\_()**: Loads the alien's image.
  + **update()**: Moves the alien left or right as part of the fleet.
  + **check\_edges()**: Checks if the alien has touched the edge of the screen, signaling that the fleet needs to drop down and change direction.

**Bullet (from bullet.py) - The Ammunition 💥**

This class manages the bullets fired by the player's ship.

* **Usage**: Creates a bullet, moves it up the screen, and handles its appearance.
* **Important Methods**:
  + **\_\_init\_\_()**: Creates a new bullet object at the top of the ship's current location.
  + **update()**: Moves the bullet up the screen in every frame.
  + **draw\_bullet()**: Draws the bullet (a simple rectangle) on the screen.

**GameStats (from game\_stats.py) - The Scorekeeper 📊**

This class tracks game data behind the scenes and does not draw anything.

* **Usage**: Keeps track of the current score, high score, level, and lives remaining.
* **Important Methods**:
  + **\_\_init\_\_()**: Sets up the initial statistics and sets the game to inactive so it starts on the menu screen.
  + **reset\_stats()**: Resets the score and lives when a new game starts.

**Scoreboard (from scoreboard.py) - The Display Board 🏆**

This class takes the numbers from GameStats and makes them visible to the player.

* **Usage**: Displays the score, high score, level, and remaining lives on the screen.
* **Important Methods**:
  + **prep\_score()**: Turns a number (like 50) into an image of the text "50".
  + **show\_score()**: Draws the score, high score, and level images onto the screen.

**Button (from button.py) - The "Play" Button 🟩**

A simple, reusable class for creating a clickable button with text.

* **Usage**: Creates the green "Play" button that appears when the game is inactive.
* **Important Methods**:
  + **\_\_init\_\_()**: Sets up the button's size, color, and the message it displays.
  + **draw\_button()**: Draws the button's rectangle and text onto the screen.