**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Aliens.py\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

import pygame

from button import Button

from Bullet import bullet

from ship import Ship

from pygame.sprite import Group  #Group is a subclass of pygame sprite class which acts as  A container class to hold and manage multiple Sprite objects.

import game\_function as gf # Module creates by us to get the functions which will make the game work

from settings import settings

from Alien import Alien

from game\_stats import GameStats

from Scoreboard import Scoreboard

'''we are creating a function named run\_game from where the game execution will start.

    we dont need to run other python files but this to run the game

'''

def run\_game():

    pygame.init()

    '''it is the initialize function of pygame which initializes background settings that Pygame needs to work properly

    '''

    ai\_settings = settings()

    ''' Creating an instance of settings.py after importing it'''

    screen = pygame.display.set\_mode((ai\_settings.width , ai\_settings.height))

    '''when we call pygame.display.set\_mode it creates a window which takes a tupple as an arguments and create a window accordingly to the given width and hight into the tupple'''

    pygame.display.set\_caption("Alien\_Invasion")

    ''' So instead of keeping the window nameless we are naming the window by our game\_name'''

    play\_button = Button(ai\_settings,screen,"Play")

    '''Creating an instance of Button class from button.py . Used to create a play button before staring the game'''

    stats = GameStats(ai\_settings)

    ''''creating an instance of Gamestats class from game\_stats.py . Used for creating game stats '''

    sb = Scoreboard(ai\_settings,screen,stats)

    '''creating an instance of scoreboard class from scoreboard.py. Used to keep scores and high\_scores of the game'''

    ship = Ship(ai\_settings, screen)

    '''Creating an instance of Ship from ship.py. Used to create a player ship which will later shoot bullets to destroy alien ships'''

    bullets = Group()

    '''Group is a subclass of pygame sprite class which acts as  A container class to hold and manage multiple Sprite objects.

        We are going to use this group to keep track of all the bullets on the screen and destroy those who left the screen

    '''

    aliens = Group()

    '''

        we are going to create a sprite group to keep track of all the alien\_ships on the screen and give them movement

        and destroy those ship which got hit by the player's bullet

    '''

    #alien = Alien(ai\_settings,screen)

    '''creating an instance of alien class from Alien.py. Used to create alien which later is used to create fleet'''

    gf.create\_fleet(ai\_settings,screen,ship ,aliens)

    ''' A function from game\_function which will create a fleet of alien\_ships  '''

    while True:

        '''In this while loop we are writing certain lines of codes which will run every frame ie. screen updates'''

        gf.check\_events(ai\_settings,screen,stats, play\_button, ship ,sb , aliens ,bullets)

        '''Function for managing events in game like key-presses'''

        if(stats.game\_active):

        #This if loop is to check whether the game is active ie whether the player has pressed the play button or not

            ship.update()

            '''This function is from the ship function which lets us position the ship when controlled by the player'''

            gf.update\_bullets(ai\_settings, screen , stats ,sb , ship , aliens ,bullets)

            '''This function is from Game\_function to update bullets position from bullets group and delete those bullets which hit alien or got out of range'''

            gf.update\_aliens(ai\_settings, stats , screen ,  ship, sb , aliens , bullets)

            '''this function is to update aliens position whether they get hit by bullets '''

        gf.update\_screen(ai\_settings,screen, stats, sb ,ship, aliens ,bullets,play\_button)

        '''WHether the game is active or not this function runs either way as this function is display the current situation of the game'''

run\_game()

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*game\_function.py\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

''' This python Module will store number of different functions which will make the game work and prevent the main game file from getting too lengthy

    This module will hold most of the game logic

'''

import sys

import pygame

from Bullet import bullet

from Alien import Alien

from time import sleep

#from pygame.constants import K\_w

#from Alien\_Invasion.ship import Ship

'''

The following function is to check which key is pressed when a key is pressed and act accordingly

'''

def check\_keydown\_events(event,ai\_settings, screen , ship , bullets):

    #if the right key or 'd' key is pressed keep moving the ship right

    if event.key == pygame.K\_RIGHT  or event.key == pygame.K\_d:

        ship.rect.centerx += 1

        ship.moving\_right = True

    #if the left key or 'a' key is pressed keep moving the ship left

    if event.key == pygame.K\_LEFT  or event.key == pygame.K\_a:

        ship.rect.centerx -= 1

        ship.moving\_left = True

    #if space is pressed fire a bullet ie. create a bullet and add it in bullets group

    if event.key == pygame.K\_SPACE and len(bullets) < ai\_settings.bullets\_allowed:

        new\_bullet = bullet(ai\_settings , screen , ship)

        bullets.add(new\_bullet)

'''

The keyup function is to stop the movement of the ship in that direction if its already on

'''

def check\_keyup\_events(event,ship):

    #stop movement to right if the right or d key is up

    if event.key == pygame.K\_RIGHT or event.key == pygame.K\_d:

        ship.moving\_right = False

    #stop movement to left if the left or d key is up

    if event.key == pygame.K\_LEFT or event.key == pygame.K\_a:

        ship.moving\_left = False

#To check if the created play button is pressed by the player or not

def check\_play\_button(ai\_settings,screen, stats , sb ,play\_button , ship , aliens ,bullets , mouse\_x , mouse\_y):

    #Check whether the mouse pressed position collides with the play button position

    if play\_button.rect.collidepoint(mouse\_x , mouse\_y) and not stats.game\_active:

        ai\_settings.initialize\_dynamic\_settings()

        #initialize initial settings of the game

        pygame.mouse.set\_visible(False)

        #set cursor visibility to off

        stats.reset\_stats()

        #reset stats

        stats.game\_active = True

        aliens.empty()

        bullets.empty()

        #emptying aliens and bullets group as a new start

        create\_fleet(ai\_settings, screen, ship, aliens)

        ship.center\_ship()

        #keeping the ship starting position to the middle

        sb.prep\_score()

        sb.prep\_high\_score()

        sb.prep\_ship()

        sb.prep\_level()

        #displaying the score ,high score ,Ship\_lives and current\_level

# this is to check key press or quit game event is triggered by the player

def check\_events(ai\_settings,screen,stats, play\_button, ship , sb , aliens, bullets):

    for event in pygame.event.get():

        #Check whether the player closed the window and Quit then program in background also

        #pygame.quit is triggered when the player closes pygame window

        if event.type == pygame.QUIT:

            sys.exit()

        #check if any key is pressed by using teh pygame.KEYDOWN which detects whether key is pressed

        elif (event.type == pygame.KEYDOWN):

            check\_keydown\_events(event,ai\_settings,screen, ship , bullets)

        #check if any key is up by using the pygame.KEYup which detects whether key press is undone

        elif (event.type == pygame.KEYUP):

            check\_keyup\_events(event,ship)

        #check if mouse button us pressed then get the position by pygame.mouse.get\_pos() a pygame function and then make a call to check play button function

        elif (event.type == pygame.MOUSEBUTTONDOWN):

            mouse\_x , mouse\_y = pygame.mouse.get\_pos()

            check\_play\_button(ai\_settings,screen, stats , sb , play\_button , ship , aliens ,bullets , mouse\_x , mouse\_y)

#updating screen with current situation of the screen

def update\_screen(ai\_settings , screen ,stats,sb, ship , aliens, bullets ,play\_button):

    screen.fill(ai\_settings.bg\_color)

    #Fill the Surface with a solid color. If no rect argument is given the entire Surface will be filled.

    #draw bullets on screen

    for bullet in bullets.sprites():

        bullet.draw\_bullet()

    #if game is not active show Play button else show score , ship and aliens

    if not stats.game\_active:

        play\_button.draw\_button()

    else:

        sb.show\_score()

        ship.blitme()

        aliens.draw(screen)

    pygame.display.flip()

    #pygame.display.flip updates the entire screen

#check bullets position and situation whether its hit of out of bound

def update\_bullets( ai\_settings , screen ,stats , sb, ship,  aliens , bullets):

    bullets.update()

    #for bullet in bullets.copy():

    #check if bullet got out of bound

    for bullet in bullets:

        if bullet.rect.bottom <0:

            bullets.remove(bullet)

    #print(len(bullets))

    #check if bullet collide with alien.

    #pygame.sprite.groupcollide(group1, group2, dokill1, dokill2) :- Find all sprites that collide between two groups. where dokill is to destroy both alien and bullet by setting true

    collisions = pygame.sprite.groupcollide(bullets,aliens,True , True)

    # It returns a dictionary where both the elements that colloided are stored

    if collisions:

        #print(collisions.values())

        #Just to be fair we are checking all the hits even if the are at exact same time by two different bullets

        for aliens in collisions.values():

            stats.score += ai\_settings.alien\_points

        # if current score is greater than high score set that to the high score

        if stats.score > stats.high\_score:

            stats.high\_score = stats.score

            #print(stats.high\_score)

        sb.prep\_score()

        sb.prep\_high\_score()

    #if we get all the aliens from the screen

    if len(aliens) == 0:

        #destroy all bullets from screen so it wont hit upcoming aliens

        bullets.empty()

        # increasing difficulty of the game

        ai\_settings.increase\_speed()

        #create a new fleet of aliens

        create\_fleet(ai\_settings , screen , ship, aliens )

        stats.level+=1

        sb.prep\_level()

# to check how many aliens we can fit onto the screen and return the number

def get\_number\_aliens\_x(ai\_settings , alien\_width):

    available\_space\_x = ai\_settings.width - 2 \* alien\_width

    number\_aliens\_x = int(available\_space\_x / (2\*alien\_width))

    return number\_aliens\_x

# to check how many aliens we can fit vertically

def get\_number\_rows(ai\_settings , ship\_height , alien\_height):

    available\_space\_y = (ai\_settings.height - (3\*alien\_height))

    number\_rows = int(available\_space\_y/(2\*alien\_height))

    return number\_rows

#Creating a single alien

def create\_alien(ai\_settings , screen , aliens , alien\_number , row\_number):

    alien = Alien(ai\_settings,screen)

    alien\_width = alien.rect.width

    #aligning the alien in a sequentially numbered position

    alien.x = alien\_width + 2\*alien\_width\*alien\_number

    alien.rect.x = alien.x

    alien.rect.y = alien.rect.height + 2\*alien.rect.height\*row\_number

    aliens.add(alien)

    #adding that alien to sprite\_group

def update\_aliens( ai\_settings , stats , screen , ship ,sb , aliens , bullets):

    #aliens.update()

    check\_fleet\_edges(ai\_settings , aliens)

    aliens.update()

    if pygame.sprite.spritecollideany(ship,aliens):

        print("Ship Hit!!!")

    #if pygame.sprite.spritecollideany(ship,aliens):

        ship\_hit(ai\_settings , stats , screen , ship , sb, aliens , bullets)

    check\_aliens\_bottom(ai\_settings , stats , screen , ship , sb , aliens , bullets)

#creating fleet of alien

def create\_fleet(ai\_settings, screen, ship, aliens ):

    alien = Alien(ai\_settings , screen)

    number\_aliens\_x = get\_number\_aliens\_x(ai\_settings, alien.rect.width)

    number\_rows = get\_number\_rows(ai\_settings,ship.rect.height , alien.rect.height)

    #creating an alien and getting the number of aliens we can fit horizontally and vertically

    #dual for loop to form a patern of creating aliens

    for row\_number in range(number\_rows):

        for alien\_number in range(number\_aliens\_x):

            create\_alien(ai\_settings,screen,aliens,alien\_number , row\_number)

#checking edges of the aliens and changing direction if it's about to hit the edge of either side

def check\_fleet\_edges(ai\_settings , aliens):

    for alien in aliens.sprites():

        if alien.check\_edges():

            change\_fleet\_direction(ai\_settings,aliens)

            break

#changing the direction of fleet and dropping it by certain value

def change\_fleet\_direction(ai\_settings , aliens):

    for alien in aliens.sprites():

        alien.rect.y += ai\_settings.fleet\_drop\_speed

    ai\_settings.fleet\_direction \*= -1

#def update\_aliens(ai\_settings ,ship, aliens):

#when ship gets hit by an alien

def ship\_hit(ai\_settings, stats , screen , ship , sb ,aliens , bullets):

    stats.ship\_left -= 1

    sb.prep\_ship()

    #check if number of lives player has left is greater than zero

    if stats.ship\_left >0:

        #reset bullets and aliens if lives are left

        aliens.empty()

        bullets.empty()

        create\_fleet(ai\_settings ,screen , ship , aliens)

        ship.center\_ship()

        sleep(0.5)

    else:

        #if no lives set the game active parameter to False and show mouse cursor

        stats.game\_active = False

        pygame.mouse.set\_visible(True)

#check if alien hits the bottom

def check\_aliens\_bottom(ai\_settings , stats , screen , ship , sb , aliens , bullets):

    screen\_rect = screen.get\_rect()

    #check for every alien

    for alien in aliens.sprites():

        #if an alien hits the bottom of the screen then call ship hit function as both needs to run same set of codes

        if(alien.rect.bottom >= screen\_rect.bottom):

            ship\_hit(ai\_settings , stats , screen , ship , sb ,aliens , bullets)

            break

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Button.py\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

'''This module is to create button and draw that button o screen when necessary

'''

import pygame

class Button:

    def \_\_init\_\_(self , ai\_settings , screen , msg) -> None:

        self.screen = screen

        self.screen\_rect = screen.get\_rect()

        self.width , self.height = 200,50

        self.button\_color = (0,255,0)

        self.text\_color = (255,255,255)

        self.font = pygame.font.SysFont(None,52)

        self.rect = pygame.Rect(0,0,self.width , self.height)

        self.rect.center = self.screen\_rect.center

        self.prep\_msg(msg)

    #creating function to create a rect image of the button

    def prep\_msg(self,msg):

        #font.render is used to create an image where the image is created which surrounds text

        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

    #drawing that button on screen

    def draw\_button(self):

        self.screen.fill(self.button\_color , self.rect)

        self.screen.blit(self.msg\_image,self.msg\_image\_rect)

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*game\_stats.py\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

'''

This module keeps track of basic game stats and resets it when game is over

'''

class GameStats:

    def \_\_init\_\_(self , ai\_settings) -> None:

        self.ai\_settings = ai\_settings

        self.reset\_stats()

        self.game\_active = False

        self.ship\_left = self.ai\_settings.ship\_limit

        self.high\_score = 0

    #function to reset stat usually when game is over

    def reset\_stats(self):

        self.ship\_left = self.ai\_settings.ship\_limit

        self.score = 0

        self.level = 1

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*settings.py\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

'''Instead of giving some hard coded values and changing it later in the entire code which will be very time consuming and troublesome

    we are going to create a class which will manage all the settings parameter so even if we have to change later we only need to make

    changes in this file

'''

class settings:

    def \_\_init\_\_(self) -> None:

        #screen

        self.width = 1200

        self.height = 800

        self.bg\_color = ("cyan")

        #ship

        self.ship\_speed\_factor= 2.0

        self.ship\_limit = 3

        #Bullets

        self.bullet\_speed\_factor = 3

        self.bullet\_width = 5

        self.bullet\_height = 15

        self.bullet\_color = 25,25,112

        self.bullets\_allowed = 5

        #alien

        self.alien\_speed = 0.5

        self.fleet\_drop\_speed = 10

        self.fleet\_direction = 1

        #Amount by which difficulty of game should be increased

        self.speedup = 1.2

        self.initialize\_dynamic\_settings()

        self.alien\_points = 50

    #These are the initial settings of game

    def initialize\_dynamic\_settings(self):

        self.ship\_speed\_factor = 2.0

        self.bullet\_speed\_factor = 2

        self.alien\_speed = 0.5

        self.fleet\_direction = 1

    #This function is called when player completes certain level . It increases the difficulty of the game

    def increase\_speed(self):

        self.ship\_speed\_factor \*= self.speedup

        self.bullet\_speed\_factor \*= self.speedup

        self.alien\_speed \*= self.speedup

        self.alien\_points \*= self.speedup

        print(self.alien\_points)

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Scoreboard.py\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

'''This module is created to keep track of score and display it on the right position of the screen

'''

import pygame

from pygame.sprite import Group

from ship import Ship

class Scoreboard:

    def \_\_init\_\_(self , ai\_setting , screen , stats) -> None:

        self.screen = screen

        self.screen\_rect = screen.get\_rect()

        self.ai\_settings = ai\_setting

        self.stats = stats

        self.text\_color = (90,90,90)

        self.font = pygame.font.SysFont(None,23)

        self.prep\_score()

        self.prep\_high\_score()

        self.prep\_level()

        self.prep\_ship()

    #To create a display of level where we are creating an image of level number

    def prep\_level(self):

        level\_str = str(self.stats.level)

        self.level\_image = self.font.render(level\_str , True , self.text\_color , self.ai\_settings.bg\_color)

        self.level\_rect = self.level\_image.get\_rect()

        self.level\_rect.top = self.score\_rect.bottom +20

        self.level\_rect.centerx = self.score\_rect.centerx

    #creating a display of the current score and giving it co-ordinates to get drawn on screen

    def prep\_score(self):

        score\_str = str(int(self.stats.score))

        self.score\_image = self.font.render(score\_str , True , self.text\_color,self.ai\_settings.bg\_color)

        self.score\_rect = self.score\_image.get\_rect()

        self.score\_rect.right = self.screen\_rect.right -30

        self.score\_rect.top = 30

    #creating a display of high score with co-ordinates planted in this function

    def prep\_high\_score(self):

        #print(self.stats.high\_score)

        high\_score\_str = str("High Socre: " + str( int(self.stats.high\_score) ))

        self.high\_score\_image = self.font.render(high\_score\_str , True , self.text\_color , self.ai\_settings.bg\_color)

        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

    #displaying number of lives ie. ships left with the player before game over

    def prep\_ship(self):

        self.ships = Group()

        for ship\_n in range(self.stats.ship\_left):

            ship = Ship(self.ai\_settings,self.screen , 27 , 48)

            ship.rect.x = 10+ship\_n \*ship.rect.width

            ship.rect.y = 10

            self.ships.add(ship)

    #Now this function is to actually display the images we created in all previous functions

    def show\_score(self):

        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)

        #self.screen.blit()

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*ship.py\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

'''this module is of the ship and is used for ship movements and to create a boundary beyond which the ship cant move

'''

import pygame

from pygame.sprite import Sprite

class Ship(Sprite):

    def \_\_init\_\_(self , ai\_settings , screen ,height = 54 , width = 96) -> None:

        self.screen = screen

        self.ai\_settings = ai\_settings

        super(Ship,self).\_\_init\_\_()

        self.image = pygame.image.load("Alien\_Invasion\Spaceship1.png")

        self.image=pygame.transform.scale(self.image , (height ,width))

        self.rect = self.image.get\_rect()

        self.screen\_rect = screen.get\_rect()

        self.rect.centerx = self.screen\_rect.centerx

        self.rect.bottom = self.screen\_rect.bottom

        self.center = float(self.rect.centerx)

        self.moving\_right = False

        self.moving\_left = False

    #displaying the ship onto the screen

    def blitme(self):

        #blit:-draw one image onto another

        self.screen.blit(self.image , self.rect)

    #updating ships position and checking whether ship is not moving it's boundary

    def update(self):

        if self.moving\_right and self.rect.right < self.screen\_rect.right:

            #adding ships speed to its moving direction

            self.center += self.ai\_settings.ship\_speed\_factor

            #self.rect.centerx+=1

        if self.moving\_left and self.rect.left >0:

            self.center -= self.ai\_settings.ship\_speed\_factor

            #self.rect.centerx-=1

        self.rect.centerx = self.center

    #setting the ship's center to the screen center when called which is at the start of game

    def center\_ship(self):

        self.center = self.screen\_rect.centerx

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*alien.py\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

'''This module is for creating and managing aliens

'''

import pygame

from pygame.sprite import Sprite

class Alien(Sprite):

    def \_\_init\_\_(self, ai\_settings , screen ) :

        super().\_\_init\_\_()

        self.screen = screen

        self.ai\_settings = ai\_settings

        self.image = pygame.image.load('Alien\_Invasion\Alien\_Ship1.png')

        self.image=pygame.transform.scale(self.image , (60,60))

        self.rect = self.image.get\_rect()

        self.rect.x = self.rect.width

        self.rect.y = self.rect.height

        self.x = float(self.rect.x)

    #function to display alien on screen

    def blitme(self):

        self.screen.blit(self.image , self.rect)

    #function to check if alien is about to hit the edge.If yes return true

    def check\_edges(self):

        screen\_rect = self.screen.get\_rect()

        if self.rect.right >= screen\_rect.right:

            return True

        elif self.rect.left <= 0:

            return True

        else:

            return False

    #moving the alien to either right or left direction

    def update(self):

        self.x += (self.ai\_settings.alien\_speed \* self.ai\_settings.fleet\_direction)

        self.rect.x = self.x

**\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*Bullet.py\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***

'''this module is to create and manage bullets.

'''

import pygame

from pygame.sprite import Sprite

class bullet(Sprite):

    def \_\_init\_\_(self, ai\_settings , screen , ship ) :

        super().\_\_init\_\_()

        self.screen = screen

        self.rect = pygame.Rect(0,0,ai\_settings.bullet\_width , ai\_settings.bullet\_height)

        self.rect.centerx = ship.rect.centerx

        self.rect.top = ship.rect.top

        self.y = float(self.rect.y)

        self.color = ai\_settings.bullet\_color

        self.speed\_factor = ai\_settings.bullet\_speed\_factor

    #Updating bullet's position according to its speed factor

    def update(self):

        self.y -= self.speed\_factor

        self.rect.y = self.y

    #this function is to draw the bullet onto the screen

    def draw\_bullet(self):

        pygame.draw.rect(self.screen , self.color , self.rect)