CODE:-

import pygame

from pygame import mixer

import random

import os

import pyttsx3

import math

import tkinter as tk

from tkinter import simpledialog

pygame.mixer.pre\_init(44100, -16, 2, 512)

mixer.init()

pygame.init()

# Define fps

clock = pygame.time.Clock()

fps = 60

# Screen settings

screen\_width = 600

screen\_height = 800

screen = pygame.display.set\_mode((screen\_width, screen\_height))

pygame.display.set\_caption('Space Invaders')

# Define fonts

font30 = pygame.font.SysFont('Constantia', 30)

font40 = pygame.font.SysFont('Constantia', 40)

font60 = pygame.font.SysFont('Constantia', 60) # Larger font for level display

font18 = pygame.font.SysFont('Constantia', 18) #small font for hints

# Load sounds

explosion\_fx = pygame.mixer.Sound("explosion.wav")

explosion\_fx.set\_volume(0.25)

explosion2\_fx = pygame.mixer.Sound("explosion2.wav")

explosion2\_fx.set\_volume(0.25)

laser\_fx = pygame.mixer.Sound("laser.wav")

laser\_fx.set\_volume(0.25)

# Define colors

red = (255, 0, 0)

green = (0, 255, 0)

white = (255, 255, 255)

black = (0, 0, 0) # Black Color

grey = (200, 200, 200) # Grey color

# Load background image

bg = pygame.image.load("bg.png")

# Load heart image

heart\_img = pygame.image.load("heart.jpg")

heart\_img = pygame.transform.scale(heart\_img, (20, 20))

# Initialize pyttsx3

engine = pyttsx3.init()

level\_announced = False # Variable to check if level is announced

game\_over\_announced = False # variable to check if game over is announced

player\_won = False # Variable to track if the player won

# Get Player Name

def get\_player\_name():

root = tk.Tk()

root.withdraw() # Hide the main window

player\_name = simpledialog.askstring("Space Invaders", "Please enter your name:",

initialvalue="Player") # Default name

if player\_name:

return player\_name

else:

return "Player" # Default name if user cancels or enters nothing

player\_name = get\_player\_name()

# Function to draw background

def draw\_bg():

screen.blit(bg, (0, 0))

# Function to draw text

def draw\_text(text, font, text\_col, x, y):

img = font.render(text, True, text\_col)

screen.blit(img, (x, y))

# Create Explosion class

class Explosion(pygame.sprite.Sprite):

def \_init\_(self, x, y, size):

pygame.sprite.Sprite.\_init\_(self)

self.images = []

for num in range(1, 6):

img = pygame.image.load(f"exp{num}.png")

img = pygame.transform.scale(img, (20 \* size, 20 \* size))

self.images.append(img)

self.index = 0

self.image = self.images[self.index]

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

self.rect.center = [x, y]

self.counter = 0

def update(self):

explosion\_speed = 3

self.counter += 1

if self.counter >= explosion\_speed and self.index < len(self.images) - 1:

self.counter = 0

self.index += 1

self.image = self.images[self.index]

if self.index >= len(self.images) - 1 and self.counter >= explosion\_speed:

self.kill()

# Create Bullets class

class Bullets(pygame.sprite.Sprite):

def \_init\_(self, x, y):

pygame.sprite.Sprite.\_init\_(self)

self.image = pygame.image.load("bullet.png")

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

self.rect.center = [x, y]

def update(self):

self.rect.y -= 5

if self.rect.bottom < 0:

self.kill()

# Check if bullet hits the AI Agent

global ai\_agent, score, player\_won, game\_over

if ai\_agent and pygame.sprite.spritecollide(self, ai\_agent\_group, False):

self.kill()

explosion\_fx.play()

explosion = Explosion(self.rect.centerx, self.rect.centery, 2)

explosion\_group.add(explosion)

score -= 20 # Decrease player's score

ai\_agent.ai\_agent\_hit\_count += 1

remaining\_lives = 5 - ai\_agent.ai\_agent\_hit\_count

if remaining\_lives >= 0:

if remaining\_lives==4 and not ai\_agent.lives\_announced[0]:

engine.say(f"Attention ! Defender hit! Remaining lives: {remaining\_lives}")

engine.runAndWait()

ai\_agent.lives\_announced[0]=True

elif remaining\_lives==3 and not ai\_agent.lives\_announced[1]:

engine.say(f"Attention ! Defender hit! Remaining lives: {remaining\_lives}")

engine.runAndWait()

ai\_agent.lives\_announced[1]=True

elif remaining\_lives==2 and not ai\_agent.lives\_announced[2]:

engine.say(f"Attention ! Defender hit! Remaining lives: {remaining\_lives}")

engine.runAndWait()

ai\_agent.lives\_announced[2]=True

elif remaining\_lives==1 and not ai\_agent.lives\_announced[3]:

engine.say(f"Attention ! Defender hit! Remaining lives: {remaining\_lives}")

engine.runAndWait()

ai\_agent.lives\_announced[3]=True

if ai\_agent.ai\_agent\_hit\_count >= 5:

game\_over = True

player\_won = False

engine.say(f"Mission Failed! {player\_name}!")

engine.runAndWait()

# Update score when hitting an alien

elif pygame.sprite.spritecollide(self, alien\_group, True):

self.kill()

explosion\_fx.play()

explosion = Explosion(self.rect.centerx, self.rect.centery, 2)

explosion\_group.add(explosion)

score += 10 # Increase score by 10

if level == max\_levels and len(alien\_group) == 0: # End game if level 5 is done

player\_won = True

game\_over = True

# Create Spaceship class

class Spaceship(pygame.sprite.Sprite):

def \_init\_(self, x, y, health):

pygame.sprite.Sprite.\_init\_(self)

self.image = pygame.image.load("spaceship.png")

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

self.rect.center = [x, y]

self.health\_start = health

self.health\_remaining = health

self.last\_shot = pygame.time.get\_ticks()

self.lives = 3 # Initialize lives here

self.hit\_time = 0 # Time when spaceship was last hit

self.lives\_announced = [False, False, False] # Track lives announcement

def update(self):

speed = 8

cooldown = 500 # milliseconds

global game\_over # Use the global game\_over variable

# Apply hit pause of 1 second

if pygame.time.get\_ticks() - self.hit\_time < 1000: # 1 sec = 1000 ms

return # Skip movement if hit

key = pygame.key.get\_pressed()

if key[pygame.K\_LEFT] and self.rect.left > 0:

self.rect.x -= speed

if key[pygame.K\_RIGHT] and self.rect.right < screen\_width:

self.rect.x += speed

time\_now = pygame.time.get\_ticks()

if key[pygame.K\_SPACE] and time\_now - self.last\_shot > cooldown:

laser\_fx.play()

bullet = Bullets(self.rect.centerx, self.rect.top)

bullet\_group.add(bullet)

self.last\_shot = time\_now

self.mask = pygame.mask.from\_surface(self.image)

pygame.draw.rect(screen, red, (self.rect.x, self.rect.bottom + 10, self.rect.width, 15))

if self.health\_remaining > 0:

pygame.draw.rect(screen, green, (

self.rect.x, self.rect.bottom + 10, int(self.rect.width \* (self.health\_remaining / self.health\_start)),

15))

else:

explosion = Explosion(self.rect.centerx, self.rect.centery, 3)

explosion\_group.add(explosion)

self.kill()

global player\_won # Access global plyaer\_won

player\_won = False # Player lost

game\_over = True # Set game over to True

return game\_over

def hit(self): # Call this when player is hit

self.hit\_time = pygame.time.get\_ticks() # Mark current time

if self.lives == 3 and not self.lives\_announced[0]:

self.lives -= 1

engine.say(f"Spaceship hit ! Remaining lives: {self.lives}")

engine.runAndWait()

self.lives\_announced[0] = True

elif self.lives == 2 and not self.lives\_announced[1]:

self.lives -= 1

engine.say(f"Spaceship hit ! Remaining lives: {self.lives}")

engine.runAndWait()

self.lives\_announced[1] = True

elif self.lives == 1 and not self.lives\_announced[2]:

self.lives -= 1

engine.say(f"Spaceship hit ! Remaining lives: {self.lives}")

engine.runAndWait()

self.lives\_announced[2] = True

elif self.lives==0:

game\_over = True

# Create Aliens class

class Aliens(pygame.sprite.Sprite):

def \_init\_(self, x, y):

pygame.sprite.Sprite.\_init\_(self)

self.image = pygame.image.load("alien" + str(random.randint(1, 5)) + ".png")

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

self.rect.center = [x, y]

self.move\_counter = 0

self.move\_direction = 1

self.last\_shot = pygame.time.get\_ticks()

def update(self):

self.rect.x += self.move\_direction

self.move\_counter += 1

if abs(self.move\_counter) > 75:

self.move\_direction \*= -1

self.move\_counter \*= self.move\_direction

time\_now = pygame.time.get\_ticks()

# Limit the number of alien bullets. Check bullet count BEFORE firing

global alien\_bullet\_group

if len(alien\_bullet\_group) < max\_alien\_bullets: # Only fire is total bullets on screen < max

if time\_now - self.last\_shot > random.randint(1000, 3000) - (level \* 50): # Firing rate increases

bullet = Alien\_Bullets(self.rect.centerx, self.rect.bottom, level,

spaceship) # Pass level to Alien\_Bullets and spaceship

alien\_bullet\_group.add(bullet)

self.last\_shot = time\_now

# Create Alien Bullets class

class Alien\_Bullets(pygame.sprite.Sprite):

def \_init\_(self, x, y, level, spaceship, speed\_multiplier=1.0):

pygame.sprite.Sprite.\_init\_(self)

self.image = pygame.image.load("alien\_bullet.png")

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

self.rect.center = [x, y]

self.base\_speed = 1.5 + (level \* 0.3) # Base speed + level scaling

self.speed = self.base\_speed \* speed\_multiplier # Apply speed multiplier

self.spaceship = spaceship # Store the spaceship object

def update(self):

self.rect.y += self.speed

if self.rect.top > screen\_height:

self.kill()

if pygame.sprite.spritecollide(self, spaceship\_group, False, pygame.sprite.collide\_mask):

self.kill()

explosion2\_fx.play()

self.spaceship.health\_remaining -= 1 # Update lives of spaceship object and not of class

self.spaceship.hit() # Call to mark the player as hit

explosion = Explosion(self.rect.centerx, self.rect.centery, 1)

explosion\_group.add(explosion)

# AI Agent Class

class AIAgent(pygame.sprite.Sprite):

def \_init\_(self, x, y):

pygame.sprite.Sprite.\_init\_(self)

self.image = pygame.image.load("ai (1).jpg") # Use a different alien image

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

self.rect.center = [x, y]

self.last\_shot = pygame.time.get\_ticks()

self.cooldown = 375 # 0.25 times faster than the aliens (500 \* 0.75)

self.movement\_speed = 4 # Normal speed

self.ai\_agent\_hit\_count = 0 # Resetting the hit count to 0

self.burst\_count = 0 # How many shots fired in the current burst

self.burst\_size = 30 # Fire 30 shots at a time

self.burst\_cooldown = 5000 # Pause of 5 seconds

self.last\_burst = pygame.time.get\_ticks() # When was the last time we started a burst?

self.is\_paused = False # Is the AI paused?

self.pause\_start\_time = 0 # When did the pause start?

self.ai\_hit\_pause = False # Flag to control pause

self.ai\_pause\_start\_time = 0 # Time the AI pause started

self.lives\_announced = [False, False, False, False]

def predict\_spaceship\_position(self):

# Very basic prediction - move towards where spaceship will be

# based on its current direction. More sophisticated solutions exist

spaceship\_x = spaceship.rect.centerx

speed = 8 # Spaceship speed

keys = pygame.key.get\_pressed()

if keys[pygame.K\_LEFT]:

spaceship\_x -= speed

if keys[pygame.K\_RIGHT]:

spaceship\_x += speed

spaceship\_x = max(0, min(spaceship\_x, screen\_width)) # clamp

return spaceship\_x

def update(self):

# Burst fire logic

time\_now = pygame.time.get\_ticks()

# Move to predicted spaceship position.

if spaceship.alive():

target\_x = self.predict\_spaceship\_position()

if self.rect.centerx < target\_x:

self.rect.x += self.movement\_speed

self.attack() # Call attack while moving

elif self.rect.centerx > target\_x:

self.rect.x -= self.movement\_speed

self.attack() # Call attack while moving

if self.is\_paused:

if time\_now - self.pause\_start\_time > self.burst\_cooldown: # 5 second pause

self.is\_paused = False

self.last\_burst = time\_now # Reset the burst timer

else:

if time\_now - self.last\_burst > 0:

if self.burst\_count < self.burst\_size:

if time\_now - self.last\_shot > self.cooldown: # Can we fire a shot?

# Increase the speed of bullet with level

speed\_multiplier = 1.0 + (level \* 0.05) # 5% increase in speed per level

bullet = Alien\_Bullets(self.rect.centerx, self.rect.bottom, level, spaceship, speed\_multiplier)

alien\_bullet\_group.add(bullet)

self.last\_shot = time\_now

self.burst\_count += 1 # increment shot count

else: # Burst is complete, reset

self.burst\_count = 0 # reset the shot count

self.is\_paused = True

self.pause\_start\_time = time\_now

def hit(self): # Call this when ai agent is hit

remaining\_lives = 5 - self.ai\_agent\_hit\_count

if remaining\_lives == 4 and not self.lives\_announced[0]:

engine.say(f"Defender hit! Remaining lives: {remaining\_lives}")

engine.runAndWait()

self.lives\_announced[0] = True

elif remaining\_lives == 3 and not self.lives\_announced[1]:

engine.say(f"Defender hit! Remaining lives: {remaining\_lives}")

engine.runAndWait()

self.lives\_announced[1] = True

elif remaining\_lives == 2 and not self.lives\_announced[2]:

engine.say(f"Defender hit! Remaining lives: {remaining\_lives}")

engine.runAndWait()

self.lives\_announced[2] = True

elif remaining\_lives == 1 and not self.lives\_announced[3]:

engine.say(f"Defender hit! Remaining lives: {remaining\_lives}")

engine.runAndWait()

self.lives\_announced[3] = True

def attack(self): # Function for the ai agent to start attacking

time\_now = pygame.time.get\_ticks()

if not self.is\_paused: # If it is paused, don't attack

if time\_now - self.last\_burst > 0:

if self.burst\_count < self.burst\_size:

if time\_now - self.last\_shot > self.cooldown: # Can we fire a shot?

# Increase the speed of bullet with level

speed\_multiplier = 1.0 + (level \* 0.05) # 5% increase in speed per level

bullet = Alien\_Bullets(self.rect.centerx, self.rect.bottom, level, spaceship, speed\_multiplier)

alien\_bullet\_group.add(bullet)

self.last\_shot = time\_now

self.burst\_count += 1 # increment shot count

else: # Burst is complete, reset

self.burst\_count = 0 # reset the shot count

self.is\_paused = True

self.pause\_start\_time = time\_now

# Function to create AI Agent

def create\_ai\_agent():

global ai\_agent

ai\_agent = AIAgent(screen\_width // 2, 100 + 2 \* 70) # 3rd Row

ai\_agent\_group.add(ai\_agent)

# Create sprite groups

spaceship\_group = pygame.sprite.Group()

bullet\_group = pygame.sprite.Group()

alien\_group = pygame.sprite.Group()

alien\_bullet\_group = pygame.sprite.Group()

explosion\_group = pygame.sprite.Group()

ai\_agent\_group = pygame.sprite.Group() # Group for ai agent

level = 1

max\_levels = 5

# AI Agent variables

ai\_agent = None

# Limit maximum alien bullets

max\_alien\_bullets = 5 # 5 bullets can be fired at a time

# Declare global variables before use

game\_over = False

player\_won = False

score = 0

last\_hit = 0 # Time when player was last hit

# Game time variables

start\_time = pygame.time.get\_ticks() # Get game start time

# Level control (starting)

level\_start\_time = 0

display\_level\_timer = 0

display\_level\_duration = 3 # Display level for 3 seconds

display\_level = True # Display Level at the start

def create\_aliens():

alien\_group.empty()

# Increase the number of rows and columns based on the level

num\_rows = level + 3 # Increased number of rows

num\_cols = level + 4 # Increased number of columns

for row in range(num\_rows):

for col in range(num\_cols):

alien = Aliens(100 + col \* 80, 100 + row \* 60) # adjusted x and y coordinates

alien\_group.add(alien)

# Initialize spaceship before calling create\_ai\_agent()

spaceship = Spaceship(int(screen\_width / 2), screen\_height - 100, 3)

spaceship\_group.add(spaceship)

create\_aliens()

# Get Ready Timer

get\_ready\_time = 3 # seconds

start\_time = pygame.time.get\_ticks()

game\_started = False

countdown\_number = 3

# Game start time

game\_start\_time = 0

game\_end\_time = 0 # Initialize game end time to zero for calcualting time

def reset\_ai\_agent(): # Creating reset function

global ai\_agent

if ai\_agent:

ai\_agent.kill() # remove the ai agent

ai\_agent = None # set it to none so as to reset all properties

#Mode 2

class Aliens2(pygame.sprite.Sprite):

def \_init\_(self, x, y):

pygame.sprite.Sprite.\_init\_(self)

self.image = pygame.image.load("alien" + str(random.randint(1, 5)) + ".png")

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

self.rect.center = [x, y]

self.move\_counter = 0

self.move\_direction = 1

self.last\_shot = pygame.time.get\_ticks()

self.x = float(x) # Store x as float for smoother movement

self.y = float(y) # Store y as float for smoother movement

def update(self):

self.x += self.move\_direction \* 0.5 #Small movements

self.rect.x = int(self.x)

self.move\_counter += 1

if abs(self.move\_counter) > 75:

self.move\_direction \*= -1

self.move\_counter \*= self.move\_direction

time\_now = pygame.time.get\_ticks()

# Limit the number of alien bullets. Check bullet count BEFORE firing

global alien\_bullet\_group

if len(alien\_bullet\_group) < max\_alien\_bullets: # Only fire is total bullets on screen < max

if time\_now - self.last\_shot > random.randint(1000, 3000) - (level \* 50): # Firing rate increases

bullet = Alien\_Bullets2(self.rect.centerx, self.rect.bottom, level,

ai\_spaceship) # Pass level to Alien\_Bullets and spaceship

alien\_bullet\_group.add(bullet)

self.last\_shot = time\_now

# Create Alien Bullets class

class Alien\_Bullets2(pygame.sprite.Sprite):

def \_init\_(self, x, y, level, spaceship, speed\_multiplier=1.0):

pygame.sprite.Sprite.\_init\_(self)

self.image = pygame.image.load("alien\_bullet.png")

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

self.rect.center = [x, y]

self.base\_speed = 1.5 + (level \* 0.3) # Base speed + level scaling

self.speed = self.base\_speed \* speed\_multiplier # Apply speed multiplier

self.spaceship = spaceship # Store the spaceship object

def update(self):

self.rect.y += self.speed

if self.rect.top > screen\_height:

self.kill()

if pygame.sprite.spritecollide(self, ai\_spaceship\_group, False, pygame.sprite.collide\_mask):

self.kill()

explosion2\_fx.play()

explosion = Explosion(self.rect.centerx, self.rect.centery, 1)

explosion\_group.add(explosion)

ai\_spaceship.hit()

# AI Agent Class

class AIAgent2(pygame.sprite.Sprite):

def \_init\_(self, x, y):

pygame.sprite.Sprite.\_init\_(self)

self.image = pygame.image.load("ai (1).jpg") # Use a different alien image

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

self.rect.center = [x, y]

self.last\_shot = pygame.time.get\_ticks()

self.cooldown = 375 # 0.25 times faster than the aliens (500 \* 0.75)

self.movement\_speed = 1.5 # Reduced for smoother movement

self.ai\_agent\_hit\_count = 0 # Resetting the hit count to 0

self.burst\_count = 0 # How many shots fired in the current burst

self.burst\_size = 30 # Fire 30 shots at a time

self.burst\_cooldown = 5000 # Pause of 5 seconds

self.last\_burst = pygame.time.get\_ticks() # When was the last time we started a burst?

self.is\_paused = False # Is the AI paused?

self.pause\_start\_time = 0 # When did the pause start?

self.ai\_hit\_pause = False # Flag to control pause

self.ai\_pause\_start\_time = 0 # Time the AI pause started

self.lives\_announced = [False, False, False, False]

self.is\_alive = True # For Ai agent to not getting disappeared when hits first time

self.x = float(x)

self.alien\_group = alien\_group

def predict\_spaceship\_position(self):

# Very basic prediction - move towards where spaceship will be

# based on its current direction. More sophisticated solutions exist

spaceship\_x = ai\_spaceship.rect.centerx

speed = 8 # Spaceship speed

keys = pygame.key.get\_pressed()

if keys[pygame.K\_LEFT]:

spaceship\_x -= speed

if keys[pygame.K\_RIGHT]:

spaceship\_x += speed

spaceship\_x = max(0, min(spaceship\_x, screen\_width)) # clamp

return spaceship\_x

def update(self):

if not self.is\_alive:

return # If the ai agent is not alive, do nothing

# Burst fire logic

time\_now = pygame.time.get\_ticks()

# Move to predicted spaceship position.

if ai\_spaceship.alive():

target\_x = self.predict\_spaceship\_position()

if self.rect.centerx < target\_x:

self.x += self.movement\_speed # small movement

elif self.rect.centerx > target\_x:

self.x -= self.movement\_speed # small movement

self.rect.x = int(self.x)

self.attack() # Call attack while moving

if self.is\_paused:

if time\_now - self.pause\_start\_time > self.burst\_cooldown: # 5 second pause

self.is\_paused = False

self.last\_burst = time\_now # Reset the burst timer

else:

if time\_now - self.last\_burst > 0:

if self.burst\_count < self.burst\_size:

if time\_now - self.last\_shot > self.cooldown: # Can we fire a shot?

# Increase the speed of bullet with level

speed\_multiplier = 1.0 + (level \* 0.05) # 5% increase in speed per level

bullet = Alien\_Bullets2(self.rect.centerx, self.rect.bottom, level, ai\_spaceship, speed\_multiplier)

alien\_bullet\_group.add(bullet)

self.last\_shot = time\_now

self.burst\_count += 1 # increment shot count

else: # Burst is complete, reset

self.burst\_count = 0 # reset the shot count

self.is\_paused = True

self.pause\_start\_time = time\_now

def hit(self): # Call this when ai agent is hit

self.is\_alive = False

remaining\_lives = 5 - self.ai\_agent\_hit\_count

if remaining\_lives == 4 and not self.lives\_announced[0]:

engine.say(f"Defender hit! Remaining lives: {remaining\_lives}")

engine.runAndWait()

self.lives\_announced[0] = True

elif remaining\_lives == 3 and not self.lives\_announced[1]:

engine.say(f"Defender hit! Remaining lives: {remaining\_lives}")

engine.runAndWait()

self.lives\_announced[1] = True

elif remaining\_lives == 2 and not self.lives\_announced[2]:

engine.say(f"Defender hit! Remaining lives: {remaining\_lives}")

engine.runAndWait()

self.lives\_announced[2] = True

elif remaining\_lives == 1 and not self.lives\_announced[3]:

engine.say(f"Defender hit! Remaining lives: {remaining\_lives}")

engine.runAndWait()

self.lives\_announced[3] = True

self.kill() # Make the AI agent disappear

return

def attack(self): # Function for the ai agent to start attacking

time\_now = pygame.time.get\_ticks()

if not self.is\_paused and self.is\_alive: # Make sure agent is alive

if time\_now - self.last\_burst > 0:

if self.burst\_count < self.burst\_size:

if time\_now - self.last\_shot > self.cooldown: # Can we fire a shot?

# Increase the speed of bullet with level

speed\_multiplier = 1.0 + (level \* 0.05) # 5% increase in speed per level

bullet = Alien\_Bullets2(self.rect.centerx, self.rect.bottom, level, ai\_spaceship, speed\_multiplier)

alien\_bullet\_group.add(bullet)

self.last\_shot = time\_now

self.burst\_count += 1 # increment shot count

else: # Burst is complete, reset

self.burst\_count = 0 # reset the shot count

self.is\_paused = True

self.pause\_start\_time = time\_now

# Create AISpaceship class

class AISpaceship(pygame.sprite.Sprite):

def \_init\_(self, x, y):

pygame.sprite.Sprite.\_init\_(self)

self.image = pygame.image.load("spaceship.png") # Or a different spaceship image

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

self.rect.center = [x, y]

self.last\_shot = pygame.time.get\_ticks()

self.cooldown = 500 # milliseconds

self.speed = 1.75 #Reduced AI ship Speed for Smoother Movement

self.health\_start = 3 #reset the health to health start

self.health\_remaining = 3

self.lives = 3 #Total lives of the AI SPaceship

self.lives\_announced = [False, False, False]

self.is\_alive = True # To ensure the spaceship is active

self.x = float(x) # For X Smoothing

self.y = float(y) # For y Smoothing

self.move\_direction = random.choice([-1, 1]) # Initial movement direction

def update(self):

if not self.is\_alive:

return # If the spaceship is not alive, do nothing

time\_now = pygame.time.get\_ticks()

# Basic movement

self.x += self.speed \* self.move\_direction

self.rect.x = int(self.x)

if time\_now - self.last\_shot > self.cooldown:

laser\_fx.play()

bullet = Bullets(self.rect.centerx, self.rect.top)

bullet\_group.add(bullet)

self.last\_shot = time\_now

# Keep within screen bounds (Horizontal)

if self.rect.left < 0:

self.move\_direction = 1

self.x = 0

if self.rect.right > screen\_width:

self.move\_direction = -1

self.x = float(screen\_width - self.rect.width)

def hit(self): #Taking the damage

if not self.is\_alive:

return

self.health\_remaining -= 1

if self.lives > 0: # Here to not get negative life count at the end

if self.lives == 3 and not self.lives\_announced[0]:

self.lives -= 1

engine.say(f"Spaceship hit! Remaining lives: {self.lives}")

engine.runAndWait()

self.lives\_announced[0] = True

elif self.lives == 2 and not self.lives\_announced[1]:

self.lives -= 1

engine.say(f"Spaceship hit! Remaining lives: {self.lives}")

engine.runAndWait()

self.lives\_announced[1] = True

elif self.lives == 1 and not self.lives\_announced[2]:

self.lives -= 1

engine.say(f"Spaceship hit! Remaining lives: {self.lives}")

engine.runAndWait()

self.lives\_announced[2] = True

if self.health\_remaining <= 0 and self.lives > 0:

explosion = Explosion(self.rect.centerx, self.rect.centery, 3)

explosion\_group.add(explosion)

self.health\_remaining = self.health\_start # reset health

elif self.lives <= 0: # Out of lives so game over

explosion = Explosion(self.rect.centerx, self.rect.centery, 3)

explosion\_group.add(explosion)

self.kill() # Remove spaceship

global player\_won, game\_over

player\_won = False # AI lost, so player won

game\_over = True

self.is\_alive = False # To stop calling the functions when destroyed

return game\_over

# Function to create AI Agent

def create\_ai\_agent2():

global ai\_agent

ai\_agent = AIAgent2(screen\_width // 2, 100 + 2 \* 70) # 3rd Row

ai\_agent\_group.add(ai\_agent)

def create\_aliens2():

alien\_group.empty()

# Increase the number of rows and columns based on the level

num\_rows = level + 3 # Increased number of rows

num\_cols = level + 4 # Increased number of columns

for row in range(num\_rows):

for col in range(num\_cols):

alien = Aliens2(100 + col \* 80, 100 + row \* 60) # adjusted x and y coordinates

alien\_group.add(alien)

#Mode 3

# Human vs AI

class Ai\_Aliens3(pygame.sprite.Sprite):

def \_init\_(self, x, y):

pygame.sprite.Sprite.\_init\_(self)

self.image = pygame.image.load("alien" + str(random.randint(1, 5)) + ".png")

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

self.rect.center = [x, y]

self.move\_counter = 0

self.move\_direction = 1

self.last\_shot = pygame.time.get\_ticks()

def update(self):

self.rect.x += self.move\_direction

self.move\_counter += 1

if abs(self.move\_counter) > 75:

self.move\_direction \*= -1

self.move\_counter \*= self.move\_direction

time\_now = pygame.time.get\_ticks()

# Limit the number of alien bullets. Check bullet count BEFORE firing

global alien\_bullet\_group

if len(alien\_bullet\_group) < max\_alien\_bullets: # Only fire is total bullets on screen < max

if time\_now - self.last\_shot > random.randint(1000, 3000) - (level \* 50): # Firing rate increases

bullet = Alien\_Bullets3(self.rect.centerx, self.rect.bottom, level,

ai\_spaceship) # Pass level to Alien\_Bullets and spaceship

alien\_bullet\_group.add(bullet)

self.last\_shot = time\_now

# AI Defender Class

class AIDefender3(pygame.sprite.Sprite):

def \_init\_(self, x, y):

pygame.sprite.Sprite.\_init\_(self)

self.image = pygame.image.load("ai (1).jpg") # Use a different alien image

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

self.rect.center = [x, y]

self.last\_shot = pygame.time.get\_ticks()

self.cooldown = 375 # 0.25 times faster than the aliens (500 \* 0.75)

self.movement\_speed = 4 # Normal speed

self.ai\_defender\_hit\_count = 0 # Resetting the hit count to 0

self.burst\_count = 0 # How many shots fired in the current burst

self.burst\_size = 30 # Fire 30 shots at a time

self.burst\_cooldown = 5000 # Pause of 5 seconds

self.last\_burst = pygame.time.get\_ticks() # When was the last time we started a burst?

self.is\_paused = False # Is the AI paused?

self.pause\_start\_time = 0 # When did the pause start?

self.ai\_hit\_pause = False # Flag to control pause

self.ai\_pause\_start\_time = 0 # Time the AI pause started

self.lives\_announced = [False, False, False, False]

def predict\_spaceship\_position(self):

# Very basic prediction - move towards where spaceship will be

# based on its current direction. More sophisticated solutions exist

spaceship\_x = ai\_spaceship.rect.centerx

speed = 8 # Spaceship speed

spaceship\_x = max(screen\_width//2, min(spaceship\_x, screen\_width)) # clamp

return spaceship\_x

def update(self):

# Burst fire logic

time\_now = pygame.time.get\_ticks()

# Move to predicted spaceship position.

if ai\_spaceship.alive():

target\_x = self.predict\_spaceship\_position()

if self.rect.centerx < target\_x:

self.rect.x += self.movement\_speed

self.attack() # Call attack while moving

elif self.rect.centerx > target\_x:

self.rect.x -= self.movement\_speed

self.attack() # Call attack while moving

if self.is\_paused:

if time\_now - self.pause\_start\_time > self.burst\_cooldown: # 5 second pause

self.is\_paused = False

self.last\_burst = time\_now # Reset the burst timer

else:

if time\_now - self.last\_burst > 0:

if self.burst\_count < self.burst\_size:

if time\_now - self.last\_shot > self.cooldown: # Can we fire a shot?

# Increase the speed of bullet with level

speed\_multiplier = 1.0 + (level \* 0.05) # 5% increase in speed per level

bullet = Alien\_Bullets3(self.rect.centerx, self.rect.bottom, level, ai\_spaceship,

speed\_multiplier)

alien\_bullet\_group.add(bullet)

self.last\_shot = time\_now

self.burst\_count += 1 # increment shot count

else: # Burst is complete, reset

self.burst\_count = 0 # reset the shot count

self.is\_paused = True

self.pause\_start\_time = time\_now

def hit(self): # Call this when ai agent is hit

remaining\_lives = 5 - self.ai\_defender\_hit\_count

if remaining\_lives == 4 and not self.lives\_announced[0]:

engine.say(f"Defender hit! Remaining lives: {remaining\_lives}")

engine.runAndWait()

self.lives\_announced[0] = True

elif remaining\_lives == 3 and not self.lives\_announced[1]:

engine.say(f"Defender hit! Remaining lives: {remaining\_lives}")

engine.runAndWait()

self.lives\_announced[1] = True

elif remaining\_lives == 2 and not self.lives\_announced[2]:

engine.say(f"Defender hit! Remaining lives: {remaining\_lives}")

engine.runAndWait()

self.lives\_announced[2] = True

elif remaining\_lives == 1 and not self.lives\_announced[3]:

engine.say(f"Defender hit! Remaining lives: {remaining\_lives}")

engine.runAndWait()

self.lives\_announced[3] = True

def attack(self): # Function for the ai agent to start attacking

time\_now = pygame.time.get\_ticks()

if not self.is\_paused: # If it is paused, don't attack

if time\_now - self.last\_burst > 0:

if self.burst\_count < self.burst\_size:

if time\_now - self.last\_shot > self.cooldown: # Can we fire a shot?

# Increase the speed of bullet with level

speed\_multiplier = 1.0 + (level \* 0.05) # 5% increase in speed per level

bullet = Alien\_Bullets3(self.rect.centerx, self.rect.bottom, level, ai\_spaceship,

speed\_multiplier)

alien\_bullet\_group.add(bullet)

self.last\_shot = time\_now

self.burst\_count += 1 # increment shot count

else: # Burst is complete, reset

self.burst\_count = 0 # reset the shot count

self.is\_paused = True

self.pause\_start\_time = time\_now

def create\_ai\_aliens3(level, screen\_width, ai\_alien\_group):

ai\_alien\_group.empty()

# Increase the number of rows and columns based on the level

num\_rows = level + 3 # Increased number of rows

num\_cols = level + 4 # Increased number of columns

for row in range(num\_rows):

for col in range(num\_cols):

alien = Ai\_Aliens3(screen\_width//2 + 50 + col \* 60, 100 + row \* 50) # adjusted x and y coordinates

ai\_alien\_group.add(alien)

# Function to create AI Agent

def create\_ai\_defender3():

global ai\_defender

ai\_defender = AIDefender3(screen\_width // 4 \* 3 , 100 + 2 \* 70) # Position on AI side

ai\_defender\_group.add(ai\_defender)

class Spaceship3(pygame.sprite.Sprite):

def \_init\_(self, x, y, health, is\_human=True):

pygame.sprite.Sprite.\_init\_(self)

self.image = pygame.image.load("spaceship.png")

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

self.rect.center = [x, y]

self.health\_start = health

self.health\_remaining = health

self.last\_shot = pygame.time.get\_ticks()

self.lives = 3 # Initialize lives here

self.hit\_time = 0 # Time when spaceship was last hit

self.lives\_announced = [False, False, False] # Track lives announcement

self.is\_human = is\_human

self.ai\_move\_direction = 1

self.ai\_last\_shot = pygame.time.get\_ticks()

self.ai\_fire = False

def update(self, screen\_width):

speed = 8

cooldown = 500 # milliseconds

global game\_over # Use the global game\_over variable

global ai\_fire

# Apply hit pause of 1 second

if pygame.time.get\_ticks() - self.hit\_time < 1000: # 1 sec = 1000 ms

return # Skip movement if hit

key = pygame.key.get\_pressed()

if self.is\_human: #Human Controlled

if key[pygame.K\_LEFT] and self.rect.left > 0:

self.rect.x -= speed

if key[pygame.K\_RIGHT] and self.rect.right < screen\_width // 2: # Keep within left side

self.rect.x += speed

time\_now = pygame.time.get\_ticks()

if key[pygame.K\_SPACE] and time\_now - self.last\_shot > cooldown:

laser\_fx.play()

bullet = Bullets3(self.rect.centerx, self.rect.top)

bullet\_group.add(bullet)

self.last\_shot = time\_now

else: # AI Controlled

time\_now = pygame.time.get\_ticks()

# Basic AI movement (Horizontal)

self.rect.x += self.ai\_move\_direction \* 2 # Smaller movement speed for AI

if self.rect.left < screen\_width // 2 or self.rect.right > screen\_width:

self.ai\_move\_direction \*= -1 # Reverse direction at edges

if time\_now - self.last\_shot > cooldown and ai\_fire: # added cooldown

laser\_fx.play()

bullet = Bullets3(self.rect.centerx, self.rect.top, is\_ai=True)

bullet\_group.add(bullet)

self.last\_shot = time\_now

self.mask = pygame.mask.from\_surface(self.image)

pygame.draw.rect(screen, red, (self.rect.x, self.rect.bottom + 10, self.rect.width, 15))

if self.health\_remaining > 0:

pygame.draw.rect(screen, green, (

self.rect.x, self.rect.bottom + 10, int(self.rect.width \* (self.health\_remaining / self.health\_start)),

15))

else:

explosion = Explosion(self.rect.centerx, self.rect.centery, 3)

explosion\_group.add(explosion)

self.kill()

if self.is\_human:

player\_won = False

else:

ai\_won = False # AI Lost

return game\_over

def hit(self): # Call this when player is hit

self.hit\_time = pygame.time.get\_ticks() # Mark current time

if self.lives == 3 and not self.lives\_announced[0]:

self.lives -= 1

engine.say(f"Spaceship hit! Remaining lives: {self.lives}")

engine.runAndWait()

self.lives\_announced[0] = True

elif self.lives == 2 and not self.lives\_announced[1]:

self.lives -= 1

engine.say(f"Spaceship hit! Remaining lives: {self.lives}")

engine.runAndWait()

self.lives\_announced[1] = True

elif self.lives == 1 and not self.lives\_announced[2]:

self.lives -= 1

engine.say(f"Spaceship hit! Remaining lives: {self.lives}")

engine.runAndWait()

self.lives\_announced[2] = True

elif self.lives == 0:

global game\_over

game\_over = True

class Bullets3(pygame.sprite.Sprite):

def \_init\_(self, x, y, is\_ai=False):

pygame.sprite.Sprite.\_init\_(self)

self.image = pygame.image.load("bullet.png")

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

self.rect.center = [x, y]

self.is\_ai = is\_ai #Track AI Bullets

def update(self):

if not self.is\_ai:

self.rect.y -= 5

else:

self.rect.y -= 5 # Adjusted bullet speed for AI bullets

if self.rect.bottom < 0:

self.kill()

# Check if bullet hits the AI Agent

global ai\_agent, score, player\_won, game\_over

if ai\_agent and pygame.sprite.spritecollide(self, ai\_agent\_group, False):

self.kill()

explosion\_fx.play()

explosion = Explosion(self.rect.centerx, self.rect.centery, 2)

explosion\_group.add(explosion)

score -= 20 # Decrease player's score

ai\_agent.ai\_agent\_hit\_count += 1

remaining\_lives = 5 - ai\_agent.ai\_agent\_hit\_count

if remaining\_lives >= 0:

if remaining\_lives == 4 and not ai\_agent.lives\_announced[0]:

engine.say(f"Defender hit! Remaining lives: {remaining\_lives}")

engine.runAndWait()

ai\_agent.lives\_announced[0] = True

elif remaining\_lives == 3 and not ai\_agent.lives\_announced[1]:

engine.say(f"Defender hit! Remaining lives: {remaining\_lives}")

engine.runAndWait()

ai\_agent.lives\_announced[1] = True

elif remaining\_lives == 2 and not ai\_agent.lives\_announced[2]:

engine.say(f"Defender hit! Remaining lives: {remaining\_lives}")

engine.runAndWait()

ai\_agent.lives\_announced[2] = True

elif remaining\_lives == 1 and not ai\_agent.lives\_announced[3]:

engine.say(f"Defender hit! Remaining lives: {remaining\_lives}")

engine.runAndWait()

ai\_agent.lives\_announced[3] = True

if ai\_agent.ai\_agent\_hit\_count >= 5:

ai\_agent.kill()

ai\_agent = None

explosion2\_fx.play()

# Update score when hitting an alien

elif not self.is\_ai and pygame.sprite.spritecollide(self, alien\_group, True): #Human

self.kill()

explosion\_fx.play()

explosion = Explosion(self.rect.centerx, self.rect.centery, 2)

explosion\_group.add(explosion)

score += 10 # Increase score by 10

global human\_level\_end\_time

if len(alien\_group) == 0 and human\_level\_end\_time == 0:

human\_level\_end\_time = pygame.time.get\_ticks()

elif self.is\_ai and pygame.sprite.spritecollide(self, ai\_alien\_group, True): # For the AI Side

self.kill()

explosion\_fx.play()

explosion = Explosion(self.rect.centerx, self.rect.centery, 2)

explosion\_group.add(explosion)

global ai\_level\_end\_time

if len(ai\_alien\_group) == 0 and ai\_level\_end\_time == 0:

ai\_level\_end\_time = pygame.time.get\_ticks()

class Aliens3(pygame.sprite.Sprite):

def \_init\_(self, x, y):

pygame.sprite.Sprite.\_init\_(self)

self.image = pygame.image.load("alien" + str(random.randint(1, 5)) + ".png")

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

self.rect.center = [x, y]

self.move\_counter = 0

self.move\_direction = 1

self.last\_shot = pygame.time.get\_ticks()

def update(self):

self.rect.x += self.move\_direction

self.move\_counter += 1

if abs(self.move\_counter) > 75:

self.move\_direction \*= -1

self.move\_counter \*= self.move\_direction

time\_now = pygame.time.get\_ticks()

# Limit the number of alien bullets. Check bullet count BEFORE firing

global alien\_bullet\_group

if len(alien\_bullet\_group) < max\_alien\_bullets: # Only fire is total bullets on screen < max

if time\_now - self.last\_shot > random.randint(1000, 3000) - (level \* 50): # Firing rate increases

bullet = Alien\_Bullets3(self.rect.centerx, self.rect.bottom, level,

human\_spaceship) # Pass level to Alien\_Bullets and spaceship

alien\_bullet\_group.add(bullet)

self.last\_shot = time\_now

class Alien\_Bullets3(pygame.sprite.Sprite):

def \_init\_(self, x, y, level, spaceship, speed\_multiplier=1.0):

pygame.sprite.Sprite.\_init\_(self)

self.image = pygame.image.load("alien\_bullet.png")

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

self.rect.center = [x, y]

self.base\_speed = 1.5 + (level \* 0.3) # Base speed + level scaling

self.speed = self.base\_speed \* speed\_multiplier # Apply speed multiplier

self.spaceship = spaceship # Store the spaceship object

def update(self, spaceship\_group, ai\_spaceship\_group):

self.rect.y += self.speed

if self.rect.top > screen\_height:

self.kill()

#Checking which side for bullet is to collide with spaceship

if self.spaceship.is\_human:

if pygame.sprite.spritecollide(self, spaceship\_group, False, pygame.sprite.collide\_mask):

self.kill()

explosion2\_fx.play()

self.spaceship.health\_remaining -= 1 # Update lives of spaceship object and not of class

self.spaceship.hit() # Call to mark the player as hit

explosion = Explosion(self.rect.centerx, self.rect.centery, 1)

explosion\_group.add(explosion)

else:

if pygame.sprite.spritecollide(self, ai\_spaceship\_group, False, pygame.sprite.collide\_mask):

self.kill()

explosion2\_fx.play()

self.spaceship.health\_remaining -= 1 # Update lives of spaceship object and not of class

self.spaceship.hit() # Call to mark the player as hit

explosion = Explosion(self.rect.centerx, self.rect.centery, 1)

explosion\_group.add(explosion)

# AI Agent Class

class AIAgent3(pygame.sprite.Sprite):

def \_init\_(self, x, y):

pygame.sprite.Sprite.\_init\_(self)

self.image = pygame.image.load("ai (1).jpg") # Use a different alien image

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

self.rect.center = [x, y]

self.last\_shot = pygame.time.get\_ticks()

self.cooldown = 375 # 0.25 times faster than the aliens (500 \* 0.75)

self.movement\_speed = 4 # Normal speed

self.ai\_agent\_hit\_count = 0 # Resetting the hit count to 0

self.burst\_count = 0 # How many shots fired in the current burst

self.burst\_size = 30 # Fire 30 shots at a time

self.burst\_cooldown = 5000 # Pause of 5 seconds

self.last\_burst = pygame.time.get\_ticks() # When was the last time we started a burst?

self.is\_paused = False # Is the AI paused?

self.pause\_start\_time = 0 # When did the pause start?

self.ai\_hit\_pause = False # Flag to control pause

self.ai\_pause\_start\_time = 0 # Time the AI pause started

self.lives\_announced = [False, False, False, False]

def predict\_spaceship\_position(self, human\_spaceship, screen\_width):

# Very basic prediction - move towards where spaceship will be

# based on its current direction. More sophisticated solutions exist

spaceship\_x = human\_spaceship.rect.centerx

speed = 8 # Spaceship speed

keys = pygame.key.get\_pressed()

if keys[pygame.K\_LEFT]:

spaceship\_x -= speed

if keys[pygame.K\_RIGHT]:

spaceship\_x += speed

spaceship\_x = max(0, min(spaceship\_x, screen\_width // 2)) # clamp

return spaceship\_x

def update(self, human\_spaceship, screen\_width):

# Burst fire logic

time\_now = pygame.time.get\_ticks()

# Move to predicted spaceship position.

if human\_spaceship.alive():

target\_x = self.predict\_spaceship\_position(human\_spaceship, screen\_width)

if self.rect.centerx < target\_x:

self.rect.x += self.movement\_speed

self.attack(human\_spaceship, level) # Call attack while moving

elif self.rect.centerx > target\_x:

self.rect.x -= self.movement\_speed

self.attack(human\_spaceship, level) # Call attack while moving

if self.is\_paused:

if time\_now - self.pause\_start\_time > self.burst\_cooldown: # 5 second pause

self.is\_paused = False

self.last\_burst = time\_now # Reset the burst timer

else:

if time\_now - self.last\_burst > 0:

if self.burst\_count < self.burst\_size:

if time\_now - self.last\_shot > self.cooldown: # Can we fire a shot?

# Increase the speed of bullet with level

speed\_multiplier = 1.0 + (level \* 0.05) # 5% increase in speed per level

bullet = Alien\_Bullets3(self.rect.centerx, self.rect.bottom, level, human\_spaceship,

speed\_multiplier)

alien\_bullet\_group.add(bullet)

self.last\_shot = time\_now

self.burst\_count += 1 # increment shot count

else: # Burst is complete, reset

self.burst\_count = 0 # reset the shot count

self.is\_paused = True

self.pause\_start\_time = time\_now

def hit(self): # Call this when ai agent is hit

remaining\_lives = 5 - self.ai\_agent\_hit\_count

if remaining\_lives == 4 and not self.lives\_announced[0]:

engine.say(f"Defender hit! Remaining lives: {remaining\_lives}")

engine.runAndWait()

self.lives\_announced[0] = True

elif remaining\_lives == 3 and not self.lives\_announced[1]:

engine.say(f"Defender hit! Remaining lives: {remaining\_lives}")

engine.runAndWait()

self.lives\_announced[1] = True

elif remaining\_lives == 2 and not self.lives\_announced[2]:

engine.say(f"Defender hit! Remaining lives: {remaining\_lives}")

engine.runAndWait()

self.lives\_announced[2] = True

elif remaining\_lives == 1 and not self.lives\_announced[3]:

engine.say(f"Defender hit! Remaining lives: {remaining\_lives}")

engine.runAndWait()

self.lives\_announced[3] = True

def attack(self, human\_spaceship, level): # Function for the ai agent to start attacking

time\_now = pygame.time.get\_ticks()

if not self.is\_paused: # If it is paused, don't attack

if time\_now - self.last\_burst > 0:

if self.burst\_count < self.burst\_size:

if time\_now - self.last\_shot > self.cooldown: # Can we fire a shot?

# Increase the speed of bullet with level

speed\_multiplier = 1.0 + (level \* 0.05) # 5% increase in speed per level

bullet = Alien\_Bullets3(self.rect.centerx, self.rect.bottom, level, human\_spaceship,

speed\_multiplier)

alien\_bullet\_group.add(bullet)

self.last\_shot = time\_now

self.burst\_count += 1 # increment shot count

else: # Burst is complete, reset

self.burst\_count = 0 # reset the shot count

self.is\_paused = True

self.pause\_start\_time = time\_now

# AI Defender Class

class AIDefender3(pygame.sprite.Sprite):

def \_init\_(self, x, y):

pygame.sprite.Sprite.\_init\_(self)

self.image = pygame.image.load("ai (1).jpg") # Use a different alien image

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

self.rect.center = [x, y]

self.last\_shot = pygame.time.get\_ticks()

self.cooldown = 375 # 0.25 times faster than the aliens (500 \* 0.75)

self.movement\_speed = 4 # Normal speed

self.ai\_defender\_hit\_count = 0 # Resetting the hit count to 0

self.burst\_count = 0 # How many shots fired in the current burst

self.burst\_size = 30 # Fire 30 shots at a time

self.burst\_cooldown = 5000 # Pause of 5 seconds

self.last\_burst = pygame.time.get\_ticks() # When was the last time we started a burst?

self.is\_paused = False # Is the AI paused?

self.pause\_start\_time = 0 # When did the pause start?

self.ai\_hit\_pause = False # Flag to control pause

self.ai\_pause\_start\_time = 0 # Time the AI pause started

self.lives\_announced = [False, False, False, False]

def predict\_spaceship\_position(self):

# Very basic prediction - move towards where spaceship will be

# based on its current direction. More sophisticated solutions exist

spaceship\_x = ai\_spaceship.rect.centerx

speed = 8 # Spaceship speed

spaceship\_x = max(screen\_width//2, min(spaceship\_x, screen\_width)) # clamp

return spaceship\_x

def update(self):

# Burst fire logic

time\_now = pygame.time.get\_ticks()

# Move to predicted spaceship position.

if ai\_spaceship.alive():

target\_x = self.predict\_spaceship\_position()

if self.rect.centerx < target\_x:

self.rect.x += self.movement\_speed

self.attack() # Call attack while moving

elif self.rect.centerx > target\_x:

self.rect.x -= self.movement\_speed

self.attack() # Call attack while moving

if self.is\_paused:

if time\_now - self.pause\_start\_time > self.burst\_cooldown: # 5 second pause

self.is\_paused = False

self.last\_burst = time\_now # Reset the burst timer

else:

if time\_now - self.last\_burst > 0:

if self.burst\_count < self.burst\_size:

if time\_now - self.last\_shot > self.cooldown: # Can we fire a shot?

# Increase the speed of bullet with level

speed\_multiplier = 1.0 + (level \* 0.05) # 5% increase in speed per level

bullet = Alien\_Bullets3(self.rect.centerx, self.rect.bottom, level, ai\_spaceship,

speed\_multiplier)

alien\_bullet\_group.add(bullet)

self.last\_shot = time\_now

self.burst\_count += 1 # increment shot count

else: # Burst is complete, reset

self.burst\_count = 0 # reset the shot count

self.is\_paused = True

self.pause\_start\_time = time\_now

def hit(self): # Call this when ai agent is hit

remaining\_lives = 5 - self.ai\_defender\_hit\_count

if remaining\_lives == 4 and not self.lives\_announced[0]:

engine.say(f"Defender hit! Remaining lives: {remaining\_lives}")

engine.runAndWait()

self.lives\_announced[0] = True

elif remaining\_lives == 3 and not self.lives\_announced[1]:

engine.say(f"Defender hit! Remaining lives: {remaining\_lives}")

engine.runAndWait()

self.lives\_announced[1] = True

elif remaining\_lives == 2 and not self.lives\_announced[2]:

engine.say(f"Defender hit! Remaining lives: {remaining\_lives}")

engine.runAndWait()

self.lives\_announced[2] = True

elif remaining\_lives == 1 and not self.lives\_announced[3]:

engine.say(f"Defender hit! Remaining lives: {remaining\_lives}")

engine.runAndWait()

self.lives\_announced[3] = True

def attack(self): # Function for the ai agent to start attacking

time\_now = pygame.time.get\_ticks()

if not self.is\_paused: # If it is paused, don't attack

if time\_now - self.last\_burst > 0:

if self.burst\_count < self.burst\_size:

if time\_now - self.last\_shot > self.cooldown: # Can we fire a shot?

# Increase the speed of bullet with level

speed\_multiplier = 1.0 + (level \* 0.05) # 5% increase in speed per level

bullet = Alien\_Bullets3(self.rect.centerx, self.rect.bottom, level, ai\_spaceship,

speed\_multiplier)

alien\_bullet\_group.add(bullet)

self.last\_shot = time\_now

self.burst\_count += 1 # increment shot count

else: # Burst is complete, reset

self.burst\_count = 0 # reset the shot count

self.is\_paused = True

self.pause\_start\_time = time\_now

def create\_ai\_aliens3(level, screen\_width, ai\_alien\_group):

ai\_alien\_group.empty()

# Increase the number of rows and columns based on the level

num\_rows = level + 3 # Increased number of rows

num\_cols = level + 4 # Increased number of columns

for row in range(num\_rows):

for col in range(num\_cols):

alien = Ai\_Aliens3(screen\_width//2 + 50 + col \* 60, 100 + row \* 50) # adjusted x and y coordinates

ai\_alien\_group.add(alien)

# Function to create AI Agent

def create\_ai\_defender3():

global ai\_defender

ai\_defender = AIDefender3(screen\_width // 4 \* 3 , 100 + 2 \* 70) # Position on AI side

ai\_defender\_group.add(ai\_defender)

# --- New Variables for Mode 3 ---

human\_wins = 0

ai\_wins = 0

level\_winner = None # 'human', 'ai', or None

human\_level\_end\_time = 0

ai\_level\_end\_time = 0

level\_start\_time = 0 # Common level start time

max\_levels = 3

ai\_won = False

score = 0

ai\_fire = False

winner\_announced = False

def draw\_bg3(mode3, screen\_width):

screen.blit(bg, (0, 0))

if mode3:

screen.blit(bg, (screen\_width // 2, 0)) # Draw background for AI side

# Draw win counts at the top

draw\_text(f"Human Wins: {human\_wins}", font30, white, 10, 10)

draw\_text(f"AI Wins: {ai\_wins}", font30, white, screen\_width - 200, 10)

def create\_aliens3(level, screen\_width, alien\_group):

alien\_group.empty()

# Increase the number of rows and columns based on the level

num\_rows = level + 3 # Increased number of rows

num\_cols = level + 4 # Increased number of columns

for row in range(num\_rows):

for col in range(num\_cols):

alien = Aliens3(50 + col \* 60, 100 + row \* 50) # adjusted x and y coordinates

alien\_group.add(alien)

# Function to create AI Agent

def create\_ai\_agent3(screen\_width, ai\_agent\_group, human\_spaceship):

global ai\_agent

ai\_agent = AIAgent3(screen\_width // 4, 100 + 2 \* 70) # Position on human side

ai\_agent\_group.add(ai\_agent)

def reset\_ai\_agent3(): # Creating reset function

global ai\_agent

if ai\_agent:

ai\_agent.kill() # remove the ai agent

ai\_agent = None # set it to none so as to reset all properties

def reset\_ai\_defender3():

global ai\_defender

if ai\_defender:

ai\_defender.kill()

ai\_defender = None

def draw\_menu():

screen.fill(black)

draw\_text("Space Invaders", font60, white, screen\_width // 2 - 200, screen\_height // 4)

draw\_text("1. The Defender Strike", font40, white, screen\_width // 2 - 180, screen\_height // 2 - 50)

draw\_text("2. AI VS Defender", font40, white, screen\_width // 2 - 180, screen\_height // 2 + 20)

draw\_text("3. Human vs AI", font40, white, screen\_width // 2 - 180, screen\_height // 2 + 90)

draw\_text("4. Exit", font40, white, screen\_width // 2 - 180, screen\_height // 2 + 160)

draw\_text("Press the corresponding number to choose a mode.", font18, grey, screen\_width // 2 - 200, screen\_height // 2 + 230)

pygame.display.update()

# Game loop

run = True

game\_mode = 0

menu = True

human\_spaceship = None

ai\_spaceship = None

spaceship\_group = None

ai\_spaceship\_group = None

bullet\_group = None

alien\_group = None

ai\_alien\_group = None

alien\_bullet\_group = None

explosion\_group = None

ai\_agent\_group = None

ai\_defender\_group = None

level = 1

max\_levels = 5

ai\_agent = None

ai\_defender = None

max\_alien\_bullets = 5

game\_over = False

player\_won = False

ai\_won = False

score = 0

last\_hit = 0

start\_time = 0

level\_start\_time = 0

display\_level\_timer = 0

display\_level\_duration = 3

display\_level = True

get\_ready\_time = 3

start\_time = 0

game\_started = False

countdown\_number = 3

game\_start\_time = 0

game\_end\_time = 0

human\_level\_end\_time = 0

ai\_level\_end\_time = 0

winner\_announced = False

ai\_fire = False

while run:

clock.tick(fps)

if menu:

draw\_menu()

for event in pygame.event.get():

if event.type == pygame.QUIT:

run = False

if event.type == pygame.KEYDOWN:

if event.key == pygame.K\_1:

game\_mode = 1

menu = False

# Reset variables

screen\_width = 600

screen = pygame.display.set\_mode((screen\_width, screen\_height))

level = 1

max\_levels = 5

ai\_agent = None

max\_alien\_bullets = 5

game\_over = False

player\_won = False

score = 0

last\_hit = 0

start\_time = 0

level\_start\_time = 0

display\_level\_timer = 0

display\_level\_duration = 3

display\_level = True

get\_ready\_time = 3

start\_time = 0

game\_started = False

countdown\_number = 3

game\_start\_time = 0

game\_end\_time = 0

winner\_announced = False

ai\_fire = Fal