```
import pygame
from pygame.locals import *
from snake_agent import SnakeAgent
from board import BoardEnv
import helper
import time

#   This is the SnakeGame class you'll be working with.
#   It first initializes based on the default conditions.
#   Then it trains according to the number of training steps
#       printing the stats along the way
#   Then it tests using the training set and again prints
#        prints the points along the way
#   Then it calls the show_games function which also based on
#       the parameters, shows a number of games being played
#       based on the training done.

class SnakeGame:

    #   This constructor initializes the board according to the conditions
mentioned
    #       in the helper file.
    #   It sets the board and initializes the snake agent.
    def __init__(self, args):
        self.args = args
        self.env = BoardEnv(args.snake_head_x, args.snake_head_y, args.food_x,
args.food_y)
        self.agent = SnakeAgent(self.env.get_actions(), args.Ne, args.LPC,
args.gamma)

    #   This function does the necessary function calls, to do_training() (if
necessary)
    #       then the do_testint() then show_games()
    def play(self):
        if self.args.NUM_TRAIN_ITER != 0:
            self.do_training()
        self.do_testing()
        self.show_games()

    #   This is the function that does calls the functions to do reinforcement
training
    #       as many times as specified. It also prints the statistics based on the
    #       parameter specfiied
    def do_training(self):
        print("IN TRAINING PHASE: ")
        self.agent.set_train()
        NUM_TO_STAT = self.args.NUM_TO_STAT
        self.points_results = []
        start = time.time()

        #   This loop will train for required number of times
        #   WRITE YOUR CODE IN THIS LOOP TO CALL THE TRAINING FUNCTION.
        #   AS TRAINING IS HAPPENING THE CODE IN THE LOOP WILL PRINT STATISTICS.
        #   Use self.env.reset() to reset your game after each iteration.
        for game in range(1, self.args.NUM_TRAIN_ITER + 1):
            print("TRAINING NUMBER : " + str(game))
            # YOUR CODE HERE
            # YOUR CODE HERE
            # YOUR CODE HERE
            # YOUR CODE HERE
            # YOUR CODE HERE
            dead = 0
            points = self.env.get_points()
            state = self.env.get_state()
            while dead != 1:
                successive_action = self.agent.agent_action(state, points, dead)
                state, points, dead = self.env.step(successive_action)
            self.points_results.append(points)
            self.agent.reset()
            self.env.reset()

            #UNCOMMENT THE CODE BELOW TO PRINT STATISTICS
            if game % self.args.NUM_TO_STAT == 0:
               print(
                   "Played games:", len(self.points_results) - NUM_TO_STAT, "-",
len(self.points_results),
                   "Calculated points (Average:", sum(self.points_results[-
NUM_TO_STAT:])/NUM_TO_STAT,
                   "Max points so far:", max(self.points_results[-NUM_TO_STAT:]),
                   "Min points so far:", min(self.points_results[-
NUM_TO_STAT:]),")",
               )
            # YOUR CODE HERE
        print("Training takes", time.time() - start, "seconds")
        #   THIS LINE WILL SAVE THE MODEL TO THE FILE "model.npy"
        self.agent.save_model()


    #   This function will test based on the model you created. It first reads the
    #       "model.npy" file created above and makes moves based on the trained
model
    def do_testing(self):
        print("Test Phase:")
        self.agent.set_eval()
        #   This line loads the model
        self.agent.load_model()
        points_results = []
        start = time.time()

        #   This loop runs the test the specified number of times.
        #   This is where you will write your code.
        #   Use self.env.reset() to reset your state everytime a new game begins.
        for game in range(1, self.args.NUM_TEST_ITER + 1):
            print("TESTING NUMBER: " + str(game))

            # YOUR CODE HERE
            # YOUR CODE HERE
            # YOUR CODE HERE
            # YOUR CODE HERE
            # YOUR CODE HERE

            dead = 0
            points = self.env.get_points()
            state = self.env.get_state()
            while dead == 0:
                successive_action = self.agent.agent_action(state, points, dead)
                state, points, dead = self.env.step(successive_action)
            points_results.append(points)
            self.agent.reset()
            self.env.reset()

        #UNCOMMENT THE CODE BELOW TO PRINT STATISTICS
        print("Testing takes", time.time() - start, "seconds")
        print("Number of Games:", len(points_results))
        print("Average Points:", sum(points_results)/len(points_results))
        print("Max Points:", max(points_results))
        print("Min Points:", min(points_results))


    #   This function is the one where the game will be displayed.
    #   This function is already written for you. No changes are necessary
    #       as long as YOU don't change function names or parameters.
    def show_games(self):
        print("Display Games")
        self.env.display()
        pygame.event.pump()
        self.agent.set_eval()
        points_results = []
        end = False
        for game in range(1, self.args.NUM_DISP_ITER + 1):
            state = self.env.get_state()
            dead = False
            action = self.agent.agent_action(state, 0, dead)
            count = 0
            while not dead:
                count +=1
                pygame.event.pump()
                keys = pygame.key.get_pressed()
                if keys[K_ESCAPE] or self.check_quit():
                    end = True
                    break
                state, points, dead = self.env.step(action)
                # Qlearning agent
                action = self.agent.agent_action(state, points, dead)
            if end:
                break
            self.env.reset()
            points_results.append(points)
            print("Game:", str(game)+"/"+str(self.args.NUM_DISP_ITER), "Points:", 
points)
        if len(points_results) == 0:
            return
        print("Average Points:", sum(points_results)/len(points_results))

    def check_quit(self):
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                return True
        return False


#   This is the main for the program, it generates the default arguemnts and calls 
the play function
if __name__ == "__main__":

    main_args = helper.make_args()
    print(main_args)
```

```
game1 = SnakeGame(main_args)
game1.play()
```