```
import random
import pygame
import helper


#   This class defines the board environment
#   It initializes the board and has several functions to get useful information
#       about the current board state
#   Keeps track of the board and the snake
class BoardEnv:

    #   This is the constructor, it initialiazes a game variable including the 
current
    #       Snake location and food location.
    #   The show variable is used to keep track of when we should draw it on screen 
or not
    def __init__(self, snake_head_x, snake_head_y, food_x, food_y):
        self.game = Snake(snake_head_x, snake_head_y, food_x, food_y)
        self.show = False

    #   All comments starting with a "*" are just calling functions in the Snake 
class which do the smae.

    #   *Returns all possible actions the snake can make
    def get_actions(self):
        return self.game.get_actions()

    #   *Resets the game state
    def reset(self):
        return self.game.reset()

    #   *Returns the current points during the current game state
    def get_points(self):
        return self.game.get_points()

    #   *Returns the current state of the game
    def get_state(self):
        return self.game.get_state()

    #   *Given an action does the action and returns the next state
    def step(self, action):
        state, points, dead = self.game.step(action)
        if self.show:
            self.draw(state, points, dead)
        return state, points, dead

    #   Helper function to draw the different parts of the snake game
    #   Uses the pygame module to easily draw the board, snake, and food
    def draw(self, state, points, dead):
        snake_head_x, snake_head_y, snake_body, food_x, food_y = state
        self.display.fill(helper.BLUE)
        pygame.draw.rect( self.display, helper.BLACK,
                [
                    helper.GRID_SIZE,
                    helper.GRID_SIZE,
                    helper.DISPLAY_SIZE - helper.GRID_SIZE * 2,
                    helper.DISPLAY_SIZE - helper.GRID_SIZE * 2
                ])
        # draw snake head
        pygame.draw.rect(
                    self.display,
                    helper.GREEN,
                    [
                        snake_head_x,
                        snake_head_y,
                        helper.GRID_SIZE,
                        helper.GRID_SIZE
                    ],
                    3
                )
        # draw snake body
        for seg in snake_body:
            pygame.draw.rect(
                self.display,
                helper.GREEN,
                [
                    seg[0],
                    seg[1],
                    helper.GRID_SIZE,
                    helper.GRID_SIZE,
                ],
                1
            )
        # draw food
        pygame.draw.rect(
                    self.display,
                    helper.RED,
                    [
                        food_x,
                        food_y,
                        helper.GRID_SIZE,
                        helper.GRID_SIZE
                    ]
                )

        text_surface = self.font.render("Points: " + str(points), True,
helper.WHITE)
        text_rect = text_surface.get_rect()
        text_rect.center = ((280),(25))
        self.display.blit(text_surface, text_rect)
        pygame.display.flip()
        if dead:
            # slow clock if dead
            self.clock.tick(1)
        else:
            self.clock.tick(5)

        return


    #   Main function to display the game
    #   Uses the pygame module to draw everything
    #   calls the draw() helper function
    #   also keeps track of time since beginning of the game
    def display(self):
        pygame.init()
        pygame.display.set_caption('CSE 140 Summer 21 Assignment 5')
        self.clock = pygame.time.Clock()
        pygame.font.init()
        self.font = pygame.font.Font(pygame.font.get_default_font(), 15)
        self.display = pygame.display.set_mode((helper.DISPLAY_SIZE,
helper.DISPLAY_SIZE), pygame.HWSURFACE)
        self.draw(self.game.get_state(), self.game.get_points(), False)
        self.show = True


#   Class to keep track of the snake
#   Has functions defined to return useful information
#   Most of these functions are called by the BoardEnv class
class Snake:

    #   Consrtuctor to initialize the snake and food at some positions passed to
it.
    def __init__(self, snake_head_x, snake_head_y, food_x, food_y):
        self.init_snake_head_x = snake_head_x
        self.init_snake_head_y = snake_head_y
        self.init_food_x = food_x
        self.init_food_y = food_y
        self.starve_steps = 8*(helper.DISPLAY_SIZE//helper.GRID_SIZE)**2
        self.did_starve = False
        self.did_hit_wall = False
        self.did_hit_body = False
        self.reset()

    #   Function to reset the game state to the initial one
    def reset(self):
        self.points = 0
        self.steps = 0
        self.snake_head_x = self.init_snake_head_x
        self.snake_head_y = self.init_snake_head_y
        self.snake_body = []
        self.food_x = self.init_food_x
        self.food_y = self.init_food_y
        self.did_starve = False
        self.did_hit_wall = False
        self.did_hit_body = False

    #   Function to return the current points
    def get_points(self):
        return self.points

    #   Function to return the actions that the snake can make
    #   0 -> up, 1 -> down, 2 -> left, 3 -> right
    def get_actions(self):
        return [0, 1, 2, 3]

    #   Returns the current positions and how logn the snake is
    def get_state(self):
        return [
            self.snake_head_x,
            self.snake_head_y,
            self.snake_body,
            self.food_x,
            self.food_y
        ]
    #   This function makes the move depending on the action passed to it.
    #   It also handles the case where the snake eats food and new food
    #   needs to be generated.
    #   It also decides if the snake is dead based on hitting walls,
    #   itself or runs out of max turns and returns true oif this is the case.
    def move(self, action):
        self.steps += 1

        delta_x = delta_y = 0
        if action == 0:
            delta_y = -1 * helper.GRID_SIZE
        elif action == 1:
            delta_y = helper.GRID_SIZE
        elif action == 2:
            delta_x = -1 * helper.GRID_SIZE
        elif action == 3:
            delta_x = helper.GRID_SIZE

        old_body_head = None
        if len(self.snake_body) == 1:
            old_body_head = self.snake_body[0]
        self.snake_body.append((self.snake_head_x, self.snake_head_y))
        self.snake_head_x += delta_x
        self.snake_head_y += delta_y

        if len(self.snake_body) > self.points:
            del(self.snake_body[0])

        self.handle_eatfood()

        #   Case where it moves into itself when body length greater than 1
        if len(self.snake_body) >= 1:
            for seg in self.snake_body:
                if self.snake_head_x == seg[0] and self.snake_head_y == seg[1]:
                    self.did_hit_body = True
                    return True

        #   Case when it moves into itslef when body lenght is 1
        if len(self.snake_body) == 1:
            if old_body_head == (self.snake_head_x, self.snake_head_y):
                self.did_hit_body = True
                return True




        #   dead on hitting wall
        if (self.snake_head_x < helper.GRID_SIZE or self.snake_head_y <
helper.GRID_SIZE or
            self.snake_head_x + helper.GRID_SIZE > helper.DISPLAY_SIZE-
helper.GRID_SIZE or self.snake_head_y + helper.GRID_SIZE > helper.DISPLAY_SIZE-
helper.GRID_SIZE):
            self.did_hit_wall = True
            return True

        #   Starvation case
        if self.steps > self.starve_steps:
            self.did_starve = True
            return True
        return False


    #   This is the function that does a step, given an action.
    #   it returns the next state, poitns and if it is dead.
    def step(self, action):
        is_dead = self.move(action)
        return self.get_state(), self.get_points(), is_dead


    #   This function increments the total points if the sanke ate the food
    #       and then creates another food for the snake
    def handle_eatfood(self):
        if (self.snake_head_x == self.food_x) and (self.snake_head_y ==
self.food_y):
            self.random_food()
            self.points += 1
            self.steps = 0

    #   This function creates a food at a random location and makes sure it isn't
    #       at a location the snake is already at
    def random_food(self):
        max_x = (helper.DISPLAY_SIZE - helper.WALL_SIZE - helper.GRID_SIZE)
        max_y = (helper.DISPLAY_SIZE - helper.WALL_SIZE - helper.GRID_SIZE)

        self.food_x = random.randint(helper.WALL_SIZE, max_x)//helper.GRID_SIZE *
helper.GRID_SIZE
        self.food_y = random.randint(helper.WALL_SIZE, max_y)//helper.GRID_SIZE *
helper.GRID_SIZE

        while self.check_food_on_snake():
            self.food_x = random.randint(helper.WALL_SIZE, max_x)//helper.GRID_SIZE
* helper.GRID_SIZE
            self.food_y = random.randint(helper.WALL_SIZE, max_y)//helper.GRID_SIZE
* helper.GRID_SIZE

    #   This is a helper function that checks if the newly cleated food is on the
snake
    def check_food_on_snake(self):
        if self.food_x == self.snake_head_x and self.food_y == self.snake_head_y:
            return True
        for seg in self.snake_body:
            if self.food_x == seg[0] and self.food_y == seg[1]:
                return True
        return False
```