```
import numpy as np
import argparse

#   Here we define some variables to determine the board size
DISPLAY_SIZE = 560
GRID_SIZE = 40
WALL_SIZE = 40

#   Here we define some variables for the snake size with respect to the board
SNAKE_UNIT_SIZE = 40
BOARD_LIMIT_MIN = 40
BOARD_LIMIT_MAX = 480
IN_WALL_COORD = 520

#   Here we define some variables to help draw the board
WHITE = (255, 255, 255)
RED = (255, 0, 0)
BLUE = (0, 0, 255)
BLACK = (0, 0, 0)
GREEN = (0, 255, 0)

#   Here we define more variables to help the snake with walls, food, and actions
NUM_ADJOINING_WALL_X_STATES=3
NUM_ADJOINING_WALL_Y_STATES=3
NUM_FOOD_DIR_X=3
NUM_FOOD_DIR_Y=3
NUM_ADJOINING_BODY_TOP_STATES=2
NUM_ADJOINING_BODY_BOTTOM_STATES=2
NUM_ADJOINING_BODY_LEFT_STATES=2
NUM_ADJOINING_BODY_RIGHT_STATES=2
NUM_ACTIONS = 4

#   Here we define a bunch of variables that will determine training, testing and
displaying the result
NUM_TRAIN_ITER = 600
NUM_TEST_ITER = 100
NUM_DISP_ITER = 1
NUM_TO_STAT = 100
snake_head_x = 200
snake_head_y = 200
food_x = 120
food_y = 120

NE_CONSTANT_K = 0.3



#   Here we define a varibale to save the trained state.
MODEL_SAVE_FILE = 'model.npy'


#   Here we define a function that checks if the Q array we created is in the
proper format
#   It doesn't check the values in the array, rather it only checks if the array
has the
#   correct internal parts based on which it returns true or false
#   This function is useful to make sure the Q array isn't curropted when saving
and loading
def np_error_checker(arr):
```

```
      if (type(arr) is np.ndarray and
            arr.shape==(NUM_ADJOINING_WALL_X_STATES, NUM_ADJOINING_WALL_Y_STATES,
NUM_FOOD_DIR_X, NUM_FOOD_DIR_Y,
                              NUM_ADJOINING_BODY_TOP_STATES,
NUM_ADJOINING_BODY_BOTTOM_STATES, NUM_ADJOINING_BODY_LEFT_STATES,
                              NUM_ADJOINING_BODY_RIGHT_STATES,NUM_ACTIONS)):
            return True
      else:
            return False


#   This function defines all the values to zero
#   We use numpy arrays to store all this information
#   We use the zeros function from numpy to initialize the values
def initialize_q_as_zeros():
      return np.zeros((NUM_ADJOINING_WALL_X_STATES, NUM_ADJOINING_WALL_Y_STATES,
NUM_FOOD_DIR_X, NUM_FOOD_DIR_Y,
                              NUM_ADJOINING_BODY_TOP_STATES,
NUM_ADJOINING_BODY_BOTTOM_STATES, NUM_ADJOINING_BODY_LEFT_STATES,
                              NUM_ADJOINING_BODY_RIGHT_STATES, NUM_ACTIONS))


#   Here we define a function to save the Q array that is passed to it. We run a
quick check
#   to verify the format before saving
def save(arr):
      if np_error_checker(arr):
            np.save(MODEL_SAVE_FILE,arr)
            return True
      else:
            print("\t********UNABLE TO SAVE MODEL AS FILE********")
            return False


#   Here we define a function to load Q array. We again run a quick formt check
after loading
def load():
      try:
            arr = np.load(MODEL_SAVE_FILE)
            if np_error_checker(arr):
                  print("\t********MODEL IN " + MODEL_SAVE_FILE + " LOADED")
                  return arr
            return None
      except:
            print("\t********MODEL FILE NAMED " + MODEL_SAVE_FILE + "NOT FOUND")
            return None


#   This function lets us create a bunch of arguements to pass so we can change
initial game state
def make_args():

    #   Name for program
    parser = argparse.ArgumentParser(description='CSE 140 Summer 21 Assignment 5')
    #   Number of training steps
    parser.add_argument('--NTRI', dest="NUM_TRAIN_ITER", type=int,
default=NUM_TRAIN_ITER,
                        help='Number of iterations run when training; set by
default as NUM_TRAIN_ITER=5000')
```

```
    #   Number of testing steps
    parser.add_argument('--NTEI', dest="NUM_TEST_ITER", type=int,
default=NUM_TEST_ITER,
                        help='Number of iterations run when training; set by
default as NUM_TEST_ITER=100')
    #   Number of games to display
    parser.add_argument('--DISP', dest="NUM_DISP_ITER", type=int, default=1,
                        help='Number of runs displayed; set by default as
NUM_DISP_ITER=1')
    #   Number of games to average statistics over
    parser.add_argument('--STAT', dest="NUM_TO_STAT", type=int,
default=NUM_TO_STAT,
                        help='Number of runs to take average statistics over; set
by default as NUM_TO_STAT=100')
    #   Exploration parameter
    parser.add_argument('--Ne', dest="Ne", type=int, default=NE_CONSTANT_K,
                        help='Parameter to help with next state exploration; set by
default as Ne=40')
    #   Parameter to calculate learning ragte
    parser.add_argument('--LPC', dest="LPC", type=int, default=0.5,
                        help='Parameter to determine learning parameter during
reinforcement learning; by default LPC=40')
    #   Parameter to calculate learning rage
    parser.add_argument('--gamma', dest="gamma", type=float, default=0.8,
                        help='Parameter used in reinforcement learning; by default
gamma=0.7')
    #   Starting X position of snake
    parser.add_argument('--INIT_HEAD_X', dest="snake_head_x", type=int,
default=200,
                        help='Initial X coordinate position of the snake head; by
default snake_head_x=200')
    #   Starting Y position of snake
    parser.add_argument('--INIT_HEAD_Y', dest="snake_head_y", type=int,
default=200,
                        help='Initial Y coordinate position of the snake head; by
default snake_head_y=200')
    #   Starting X position of food
    parser.add_argument('--INIT_FOOD_X', dest="food_x", type=int, default=120,
                        help='Initial X coordinate position of the food; by default
food_x=120')
    #   starting Y position of food
    parser.add_argument('--INIT_FOOD_Y', dest="food_y", type=int, default=120,
                        help='Initial Y coordinate position of the food; by default
food_y=120')
    #   Parse everything mentnoned above into args
    args = parser.parse_args()
    #   return args
    return args
```