```
# Fox Bunny Chase
# In this game, the user tries to prevent two moving animals
# from colliding by pressing and releasing the mouse
# to teleport the bunny to the top left corner of the window
# and the fox to a random location.
# The score is the number of seconds from start of game
# until the two animals collide.

import |_____???_____|, |_____???_____|, time, math


# User-defined functions

def main():
   pygame.init()
   |_____???_____|
   |_____???_____|
   w_surface = pygame.display.get_surface()
   game = Game(w_surface)
   game.play()
   pygame.quit()


# User-defined classes

class Game:
   # An object in this class represents a complete game.

   def __init__(self, surface):
      # Initialize a Game.
      # - self is the Game to initialize
      # - surface is the display window surface object

      # === objects that are part of every game that we will discuss
      self.surface = surface
      self.bg_color = pygame.Color('black')

      self.FPS = 60
      self.game_Clock = pygame.time.Clock()
      self.close_clicked = False
      self.continue_game = True

      # === game specific objects
      self.bunny_color = 'grey'
      self.fox_color = 'brown'
```

```
        self.bunny = |_____???_____|
        self.fox = |_____???_____|
        self.score = 0
        self.bunny.reset()
        self.fox.randomize()

    def play(self):
        # Play the game until the player presses the close box.
        # - self is the Game that should be continued or not.

        while not self.close_clicked:  # until player clicks close box
            # play frame
            self.handle_events()
            self.draw()
            if self.continue_game:
                self.update()
                self.decide_continue()
            self.game_Clock.tick(self.FPS) # run at most with FPS Frames Per
Second

    def handle_events(self):
        # Handle each user event by changing the game state appropriately.
        # - self is the Game whose events will be handled

        events = pygame.event.get()
        for event in events:
            if event.type == pygame.QUIT:
                self.close_clicked = True
            elif |_____???_____|:
                |_____???_____|

    def handle_mouse_up(self, event):
        # Respond to the player releasing the mouse button by
        # taking appropriate actions.
        # - self is the Game where the mouse up occurred.
        # - event is the pygame.event.Event object to handle

        self.bunny.|_____???_____|
        self.fox.|_____???_____|
```

```
def draw(self):
    # Draw all game objects.
    # - self is the Game to draw

    self.surface.fill(self.bg_color) # clear the display surface first
    |_____???_____|
    |_____???_____|
    self.draw_score()
    if |_____???_____|:
        self.draw_game_over()
    pygame.display.update() # make the updated surface appear on the display

def update(self):
    # Update the game objects.
    # - self is the Game to update

    self.bunny.move()
    self.fox.move()
    |_____???_____| = pygame.time.get_ticks() // 1000

def decide_continue(self):
    # Check and remember if the game should continue
    # - self is the Game to check

    if |_____???_____|:
        self.continue_game = False

def draw_game_over(self):
    # Draw BUNNY CAUGHT in the lower left corner of the
    # window, using the bunny's color for the font
    # and the fox's color as the background
    # - self is the Game to draw for.

    game_over_string = 'BUNNY CAUGHT'
    game_over_font = pygame.font.SysFont('', 64)
    fg_color = |_____???_____|
    bg_color = |_____???_____|
    game_over_image = game_over_font.render(game_over_string, True,
                                            fg_color, bg_color)
    height = |_____???_____|
    game_over_top_left_corner = (0, height)
    self.surface.blit(game_over_image, game_over_top_left_corner)

def draw_score(self):
```

```
        # Draw the time since the game began as a score
        # in white on the window's background.
        # - self is the Game to draw for.
        score_string = 'Score:' + str(self.score)
        score_font = pygame.font.SysFont('', 72)
        score_image = score_font.render(score_string, True,
                                        pygame.Color('white'), self.bg_color)
        score_top_left_corner = (0, 0)
        self.surface.blit(score_image, score_top_left_corner)


class Animal:
    # An object in this class represents a colored circle
    # that can move.

    def __init__(self, color, center, radius, velocity, surface):
        # Initialize a Dot.
        # - self is the Animal to initialize
        # - color is the string of the color name for the color of the animal
        # - center is a list containing the x and y int
        #    coords of the center of the animal
        # - radius is the int pixel radius of the animal
        # - velocity is a list containing the x and y components
        # - surface is the window's pygame.Surface object

        self.color = pygame.Color(color)
        self.center = center
        self.radius = radius
        self.velocity = velocity
        self.surface = surface

    def draw(self):
        # Draw the animal on the surface
        # - self is the animal

        pygame.draw.circle(self.surface, self.color, self.center, self.radius)

    def move(self):
        # Change the location and the velocity of the Animal so it
        # remains on the surface by bouncing from its edges.
        # - self is the animal

        size = self.surface.get_size()
        for coord in range(len(size)):
            self.center[coord] = |_____???_____|
```

```
            left_top_bounce = (self.center[coord] - self.radius) < 0
            right_bottom_bounce = (self.center[coord] + self. radius) >
size[coord]
            if left_top_bounce or right_bottom_bounce:
               self.velocity[coord] = |_____???_____|

      def intersects(self, animal):
         # Return True if the two animals intersect and False if they do not.
         # - self is an Animal
         # - animal is the other Animal

         distance = math.sqrt((self.center[0] - animal.center[0])**2
                              + (self.center[1] - animal.center[1])**2)
         return distance <= self.radius + animal.radius

      def reset(self):
         size = self.surface.get_size()
         for coord in range(len(size)):
            self.center[coord] = |_____???_____|

      def randomize(self):
         size = self.surface.get_size()
         for coord in range(len(size)):
            self.center[coord] = random.randint(self.radius, |_____???_____|)


main()
```