```
import pygame
import random
import sys
from enum import Enum
from collections import deque

pygame.init()

SCREEN_WIDTH = 600
SCREEN_HEIGHT = 600
GRID_SIZE = 20
GRID_WIDTH = SCREEN_WIDTH // GRID_SIZE
GRID_HEIGHT = SCREEN_HEIGHT // GRID_SIZE
FPS = 10

BLACK = (0, 0, 0)
WHITE = (255, 255, 255)
GREEN = (0, 255, 0)
RED = (255, 0, 0)
BLUE = (0, 0, 255)
PURPLE = (128, 0, 128)
YELLOW = (255, 255, 0)
DARK_PURPLE = (80, 0, 80)
LIGHT_PURPLE = (200, 0, 200)
GRAY = (100, 100, 100)
LIGHT_GRAY = (200, 200, 200)

class Direction(Enum):
    UP = (0, -1)
    DOWN = (0, 1)
    LEFT = (-1, 0)
    RIGHT = (1, 0)

screen = pygame.display.set_mode((SCREEN_WIDTH, SCREEN_HEIGHT))
pygame.display.set_caption("Pyake")
clock = pygame.time.Clock()

class SnakeHead(pygame.sprite.Sprite):
    def __init__(self, x, y, color=GREEN):
        super().__init__()
        self.image = pygame.Surface((GRID_SIZE, GRID_SIZE))
        self.image.fill(color)
        self.rect = self.image.get_rect()
        self.rect.x = x * GRID_SIZE
```

```
        self.rect.y = y * GRID_SIZE
        self.direction = Direction.RIGHT
        self.next_direction = Direction.RIGHT

    def update(self):
        self.direction = self.next_direction

        dx, dy = self.direction.value
        self.rect.x += dx * GRID_SIZE
        self.rect.y += dy * GRID_SIZE

        if self.rect.x >= SCREEN_WIDTH:
            self.rect.x = 0
        elif self.rect.x < 0:
            self.rect.x = SCREEN_WIDTH - GRID_SIZE
        if self.rect.y >= SCREEN_HEIGHT:
            self.rect.y = 0
        elif self.rect.y < 0:
            self.rect.y = SCREEN_HEIGHT - GRID_SIZE

    def change_direction(self, new_direction):
        if (new_direction == Direction.UP and self.direction != Direction.DOWN or
                new_direction == Direction.DOWN and self.direction != Direction.UP or
                new_direction == Direction.LEFT and self.direction != Direction.RIGHT or
                new_direction == Direction.RIGHT and self.direction != Direction.LEFT):
            self.next_direction = new_direction

class SnakeBody(pygame.sprite.Sprite):
    def __init__(self, x, y, color=BLUE):
        super().__init__()
        self.image = pygame.Surface((GRID_SIZE, GRID_SIZE))
        self.image.fill(color)
        self.rect = self.image.get_rect()
        self.rect.x = x * GRID_SIZE
        self.rect.y = y * GRID_SIZE

class SnakeTail(pygame.sprite.Sprite):
    def __init__(self, x, y, color=GREEN):
        super().__init__()
        self.image = pygame.Surface((GRID_SIZE, GRID_SIZE))
        self.image.fill(color)
        self.rect = self.image.get_rect()
        self.rect.x = x * GRID_SIZE
        self.rect.y = y * GRID_SIZE
```

```
class Food(pygame.sprite.Sprite):
    def __init__(self):
        super().__init__()
        self.image = pygame.Surface((GRID_SIZE, GRID_SIZE))
        self.image.fill(RED)
        self.rect = self.image.get_rect()
        self.spawn()

    def spawn(self):
        self.rect.x = random.randint(0, GRID_WIDTH - 1) * GRID_SIZE
        self.rect.y = random.randint(0, GRID_HEIGHT - 1) * GRID_SIZE

class Snake:
    def __init__(self):
        self.reset()

    def reset(self):
        self.all_sprites = pygame.sprite.Group()
        self.snake_sprites = pygame.sprite.Group()

        start_x, start_y = GRID_WIDTH // 2, GRID_HEIGHT // 2
        self.head = SnakeHead(start_x, start_y)
        self.all_sprites.add(self.head)
        self.snake_sprites.add(self.head)

        self.body_segments = []
        for i in range(1, 3):
            segment = SnakeBody(start_x - i, start_y)
            self.body_segments.append(segment)
            self.all_sprites.add(segment)
            self.snake_sprites.add(segment)

        self.tail = SnakeTail(start_x - 3, start_y)
        self.all_sprites.add(self.tail)
        self.snake_sprites.add(self.tail)

        self.grow = False
        self.new_segment_pos = None

    def update(self):
        prev_head_pos = (self.head.rect.x, self.head.rect.y)
        prev_positions = [prev_head_pos]
        for segment in self.body_segments:
```

```
            prev_positions.append((segment.rect.x, segment.rect.y))
        prev_tail_pos = (self.tail.rect.x, self.tail.rect.y)

        self.head.update()

        collision_check_group = pygame.sprite.Group(self.body_segments[1:] + [self.tail])
        if pygame.sprite.spritecollide(self.head, collision_check_group, False):
            if len(self.body_segments) > 2:
                return True

        for i, segment in enumerate(self.body_segments):
            segment.rect.x, segment.rect.y = prev_positions[i]

        if self.grow:
            new_segment = SnakeBody(prev_tail_pos[0], prev_tail_pos[1])
            self.body_segments.append(new_segment)
            self.all_sprites.add(new_segment)
            self.snake_sprites.add(new_segment)

            self.tail.rect.x, self.tail.rect.y = prev_tail_pos
            self.grow = False
        else:
            self.tail.rect.x, self.tail.rect.y = prev_positions[-1]

        return False

    def change_direction(self, direction):
        self.head.change_direction(direction)

    def check_collision_with_food(self, food):
        if pygame.sprite.collide_rect(self.head, food):
            self.grow = True
            return True
        return False

class EnemySnake:
    def __init__(self):
        self.reset()
        self.path = []
        self.path_counter = 0
        self.respawn_timer = 0
        self.is_alive = True

    def reset(self):
```

```
    self.all_sprites = pygame.sprite.Group()
    self.enemy_sprites = pygame.sprite.Group()

    start_x, start_y = random.randint(5, GRID_WIDTH-5), random.randint(5, GRID_HEIGHT-5)
    self.head = SnakeHead(start_x, start_y, PURPLE)
    self.all_sprites.add(self.head)
    self.enemy_sprites.add(self.head)

    self.body_segments = []
    for i in range(1, 3):
        segment = SnakeBody(start_x - i, start_y, DARK_PURPLE)
        self.body_segments.append(segment)
        self.all_sprites.add(segment)
        self.enemy_sprites.add(segment)

    self.tail = SnakeTail(start_x - 3, start_y, LIGHT_PURPLE)
    self.all_sprites.add(self.tail)
    self.enemy_sprites.add(self.tail)

    self.grow = False
    self.move_counter = 0
    self.direction = Direction.RIGHT
    self.next_direction = Direction.RIGHT
    self.path = []
    self.path_counter = 0
    self.is_alive = True

def update(self, target=None, walls=None):
    if not self.is_alive:
        self.respawn_timer += 1
        if self.respawn_timer >= 30:
            self.reset()
            self.respawn_timer = 0
        return

    prev_head_pos = (self.head.rect.x, self.head.rect.y)
    prev_positions = [prev_head_pos]
    for segment in self.body_segments:
        prev_positions.append((segment.rect.x, segment.rect.y))
    prev_tail_pos = (self.tail.rect.x, self.tail.rect.y)

    self.move_counter += 1
    if self.move_counter >= 3 or not self.path or self.path_counter >= len(self.path):
        if target:
```

```
            self.find_path_to_target(target, walls)
        else:
            self.random_safe_move(walls)
        self.move_counter = 0
        self.path_counter = 0

    if self.path and self.path_counter < len(self.path):
        next_pos = self.path[self.path_counter]
        dx = next_pos[0] - (self.head.rect.x // GRID_SIZE)
        dy = next_pos[1] - (self.head.rect.y // GRID_SIZE)

        if dx == 1:
            self.next_direction = Direction.RIGHT
        elif dx == -1:
            self.next_direction = Direction.LEFT
        elif dy == 1:
            self.next_direction = Direction.DOWN
        elif dy == -1:
            self.next_direction = Direction.UP

        self.path_counter += 1

    self.head.direction = self.direction
    self.head.next_direction = self.next_direction
    self.head.update()
    self.direction = self.head.direction

    for i, segment in enumerate(self.body_segments):
        segment.rect.x, segment.rect.y = prev_positions[i]

    if self.grow:
        new_segment = SnakeBody(prev_tail_pos[0], prev_tail_pos[1], DARK_PURPLE)
        self.body_segments.append(new_segment)
        self.all_sprites.add(new_segment)
        self.enemy_sprites.add(new_segment)
        self.tail.rect.x, self.tail.rect.y = prev_tail_pos
        self.grow = False
    else:
        self.tail.rect.x, self.tail.rect.y = prev_positions[-1]

def die(self):
    self.is_alive = False
    for sprite in self.all_sprites:
        sprite.kill()
```

```
def find_path_to_target(self, target, walls):
    start = (self.head.rect.x // GRID_SIZE, self.head.rect.y // GRID_SIZE)
    goal = (target.rect.x // GRID_SIZE, target.rect.y // GRID_SIZE)

    grid = [[0 for _ in range(GRID_WIDTH)] for _ in range(GRID_HEIGHT)]

    for wall in walls:
        x, y = wall.rect.x // GRID_SIZE, wall.rect.y // GRID_SIZE
        grid[y][x] = 1

    for segment in self.body_segments[:-1]:
        x, y = segment.rect.x // GRID_SIZE, segment.rect.y // GRID_SIZE
        grid[y][x] = 1

    queue = deque()
    queue.append(start)
    came_from = {}
    came_from[start] = None

    found = False
    while queue:
        current = queue.popleft()

        if current == goal:
            found = True
            break

        for direction in Direction:
            dx, dy = direction.value
            neighbor = (current[0] + dx, current[1] + dy)

            if neighbor[0] < 0:
                neighbor = (GRID_WIDTH - 1, neighbor[1])
            elif neighbor[0] >= GRID_WIDTH:
                neighbor = (0, neighbor[1])
            if neighbor[1] < 0:
                neighbor = (neighbor[0], GRID_HEIGHT - 1)
            elif neighbor[1] >= GRID_HEIGHT:
                neighbor = (neighbor[0], 0)

            if (neighbor not in came_from and
                0 <= neighbor[0] < GRID_WIDTH and
                0 <= neighbor[1] < GRID_HEIGHT and
```

```
                grid[neighbor[1]][neighbor[0]] == 0):

                queue.append(neighbor)
                came_from[neighbor] = current

        self.path = []
        if found:
            current = goal
            while current != start:
                self.path.append(current)
                current = came_from[current]
            self.path.reverse()
            self.path = self.path[:10]

    def random_safe_move(self, walls):
        safe_directions = []
        for direction in Direction:
            new_x = self.head.rect.x + direction.value[0] * GRID_SIZE
            new_y = self.head.rect.y + direction.value[1] * GRID_SIZE

            if new_x >= SCREEN_WIDTH:
                new_x = 0
            elif new_x < 0:
                new_x = SCREEN_WIDTH - GRID_SIZE
            if new_y >= SCREEN_HEIGHT:
                new_y = 0
            elif new_y < 0:
                new_y = SCREEN_HEIGHT - GRID_SIZE

            wall_collision = False
            for wall in walls:
                if new_x == wall.rect.x and new_y == wall.rect.y:
                    wall_collision = True
                    break

            body_collision = False
            for segment in self.body_segments[:-1]:
                if new_x == segment.rect.x and new_y == segment.rect.y:
                    body_collision = True
                    break

            if not wall_collision and not body_collision:
                safe_directions.append(direction)
```

```
        if safe_directions:
            if self.direction in safe_directions and random.random() < 0.7:
                self.next_direction = self.direction
            else:
                self.next_direction = random.choice(safe_directions)
        else:
            self.next_direction = self.direction

    def check_collision_with_food(self, food):
        if pygame.sprite.collide_rect(self.head, food):
            self.grow = True
            return True
        return False

class Wall(pygame.sprite.Sprite):
    def __init__(self, x, y):
        super().__init__()
        self.image = pygame.Surface((GRID_SIZE, GRID_SIZE))
        self.image.fill(YELLOW)
        self.rect = self.image.get_rect()
        self.rect.x = x * GRID_SIZE
        self.rect.y = y * GRID_SIZE

def create_walls():
    walls = []

    for x in range(GRID_WIDTH):
        walls.append(Wall(x, 0))
        walls.append(Wall(x, GRID_HEIGHT - 1))
    for y in range(1, GRID_HEIGHT - 1):
        walls.append(Wall(0, y))
        walls.append(Wall(GRID_WIDTH - 1, y))

    for _ in range(5):
        x = random.randint(5, GRID_WIDTH - 6)
        y = random.randint(5, GRID_HEIGHT - 6)
        length = random.randint(3, 7)

        if random.random() < 0.5:
            for i in range(length):
                if x + i < GRID_WIDTH - 1:
                    walls.append(Wall(x + i, y))
        else:
            for i in range(length):
```

```
            if y + i < GRID_HEIGHT - 1:
                walls.append(Wall(x, y + i))

    return walls

def draw_text(text, size, color, x, y):
    font = pygame.font.SysFont('Arial', size)
    text_surface = font.render(text, True, color)
    text_rect = text_surface.get_rect(center=(x, y))
    screen.blit(text_surface, text_rect)
    return text_rect

def draw_button(text, size, color, bg_color, x, y, width, height):
    button_rect = pygame.Rect(x - width//2, y - height//2, width, height)
    pygame.draw.rect(screen, bg_color, button_rect)
    pygame.draw.rect(screen, color, button_rect, 2)
    text_rect = draw_text(text, size, color, x, y)
    return button_rect

def show_menu():
    screen.fill(BLACK)
    draw_text("PYAKE", 48, GREEN, SCREEN_WIDTH//2, SCREEN_HEIGHT//4)

    play_button = draw_button("PLAY VS AI", 36, GREEN, BLACK, SCREEN_WIDTH//2,
SCREEN_HEIGHT//2, 200, 50)
SCREEN_HEIGHT//2 + 80, 250, 50)
    quit_button = draw_button("QUIT", 36, RED, BLACK, SCREEN_WIDTH//2,
SCREEN_HEIGHT//2 + 160, 150, 50)

    pygame.display.flip()

    while True:
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                pygame.quit()
                sys.exit()
            if event.type == pygame.MOUSEBUTTONDOWN:
                mouse_pos = pygame.mouse.get_pos()
                if play_button.collidepoint(mouse_pos):
                    return "play"
                elif watch_button.collidepoint(mouse_pos):
                    return "watch"
                elif quit_button.collidepoint(mouse_pos):
                    pygame.quit()
```

```
                sys.exit()
        pygame.time.Clock().tick(FPS)

def show_game_over(winner=None):
    screen.fill(BLACK)
    if winner:
        if winner == "player":
            draw_text("YOU WIN!", 48, GREEN, SCREEN_WIDTH//2, SCREEN_HEIGHT//3)
        else:
            draw_text("ENEMY WINS!", 48, PURPLE, SCREEN_WIDTH//2, SCREEN_HEIGHT//3)
    else:
        draw_text("GAME OVER", 48, RED, SCREEN_WIDTH//2, SCREEN_HEIGHT//3)

    draw_text("Press R to return to menu", 36, WHITE, SCREEN_WIDTH//2,
SCREEN_HEIGHT//2)
    draw_text("Press Q to quit", 36, WHITE, SCREEN_WIDTH//2, SCREEN_HEIGHT//2 + 50)

    pygame.display.flip()

    waiting = True
    while waiting:
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                pygame.quit()
                sys.exit()
            if event.type == pygame.KEYDOWN:
                if event.key == pygame.K_r:
                    waiting = False
                elif event.key == pygame.K_q:
                    pygame.quit()
                    sys.exit()

def show_start_message():
    screen.fill(BLACK)
    draw_text("Press any arrow key to start", 36, WHITE, SCREEN_WIDTH//2,
SCREEN_HEIGHT//2)
    pygame.display.flip()

    waiting = True
    while waiting:
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                pygame.quit()
                sys.exit()
```

```
            if event.type == pygame.KEYDOWN:
                if event.key in (pygame.K_UP, pygame.K_DOWN, pygame.K_LEFT,
pygame.K_RIGHT):
                    waiting = False
                elif event.key == pygame.K_q:
                    pygame.quit()
                    sys.exit()
        pygame.time.Clock().tick(FPS)

def game_loop(mode):
    snake = Snake()
    enemy = EnemySnake()
    food = Food()
    walls = create_walls()

    all_sprites = pygame.sprite.Group()
    all_sprites.add(snake.all_sprites)
    all_sprites.add(enemy.all_sprites)
    all_sprites.add(food)
    for wall in walls:
        all_sprites.add(wall)

    wall_sprites = pygame.sprite.Group(walls)

    running = True
    game_over = False
    player_score = 0
    enemy_score = 0
    game_started = False

    screen.fill(BLACK)
    all_sprites.draw(screen)
    draw_text("Press any arrow key to start", 36, WHITE, SCREEN_WIDTH//2,
SCREEN_HEIGHT//2)
    pygame.display.flip()

    while running:
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
                running = False
                pygame.quit()
                sys.exit()
            elif event.type == pygame.KEYDOWN:
                if game_over:
```

```
                if event.key == pygame.K_r:
                    running = False
                elif event.key == pygame.K_q:
                    pygame.quit()
                    sys.exit()
            elif not game_started and mode == "play":
                if event.key in (pygame.K_UP, pygame.K_DOWN, pygame.K_LEFT,
pygame.K_RIGHT):
                    game_started = True
                    snake.change_direction({
                        pygame.K_UP: Direction.UP,
                        pygame.K_DOWN: Direction.DOWN,
                        pygame.K_LEFT: Direction.LEFT,
                        pygame.K_RIGHT: Direction.RIGHT
                    }[event.key])
            elif mode == "play" and game_started:
                if event.key == pygame.K_UP:
                    snake.change_direction(Direction.UP)
                elif event.key == pygame.K_DOWN:
                    snake.change_direction(Direction.DOWN)
                elif event.key == pygame.K_LEFT:
                    snake.change_direction(Direction.LEFT)
                elif event.key == pygame.K_RIGHT:
                    snake.change_direction(Direction.RIGHT)

        if not game_over:
            if not game_started:
                continue

            if mode == "play":
                game_over = snake.update()
                if game_over:
                    show_game_over("enemy")
                    running = False
                    continue

            if mode == "play":
                enemy.update(snake.head, walls)
            else:
                enemy.update(None, walls)

            if pygame.sprite.spritecollide(snake.head, wall_sprites, False):
                game_over = True
                show_game_over("enemy")
```

```
    running = False
    continue

if pygame.sprite.spritecollide(enemy.head, wall_sprites, False):
    enemy_score += 1
    enemy.die()

if pygame.sprite.spritecollide(snake.head, enemy.enemy_sprites, False):
    game_over = True
    show_game_over("enemy")
    running = False
    continue

if pygame.sprite.spritecollide(enemy.head, snake.snake_sprites, False):
    if mode == "play":
        player_score += 1
        enemy.die()
    else:
        enemy.die()

if snake.check_collision_with_food(food):
    player_score += 1
    food.spawn()
    while (pygame.sprite.spritecollide(food, snake.snake_sprites, False) or
           pygame.sprite.spritecollide(food, enemy.enemy_sprites, False) or
           pygame.sprite.spritecollide(food, wall_sprites, False)):
        food.spawn()

if enemy.check_collision_with_food(food):
    enemy_score += 1
    food.spawn()
    while (pygame.sprite.spritecollide(food, snake.snake_sprites, False) or
           pygame.sprite.spritecollide(food, enemy.enemy_sprites, False) or
           pygame.sprite.spritecollide(food, wall_sprites, False)):
        food.spawn()

screen.fill(BLACK)
all_sprites.draw(screen)

draw_text(f"Player: {player_score}", 20, GREEN, 100, 20)
draw_text(f"Enemy: {enemy_score}", 20, PURPLE, SCREEN_WIDTH - 100, 20)

all_sprites.remove(snake.all_sprites)
all_sprites.add(snake.all_sprites)
```

```
            all_sprites.remove(enemy.all_sprites)
            all_sprites.add(enemy.all_sprites)

            pygame.display.flip()
        else:
            show_game_over()
            running = False

        clock.tick(FPS)

def main():
    while True:
        mode = show_menu()
        game_loop(mode)

if __name__ == "__main__":
    main()
```