```python
import pygame
import sys
import random
from collections import deque

# Game settings
WIDTH, HEIGHT = 600, 600  # Screen size
GRID_SIZE = 20  # Each cell in the grid is 20x20 pixels
GRID_WIDTH = WIDTH // GRID_SIZE
GRID_HEIGHT = HEIGHT // GRID_SIZE
FPS = 10  # Initial speed

# Colors
BLACK = (0, 0, 0)
GREEN = (0, 255, 0)
RED = (255, 0, 0)
WHITE = (255, 255, 255)
LIGHT_BLUE = (0, 204, 255)
YELLOW = (255, 255, 0)

# Pygame setup
pygame.init()
window = pygame.display.set_mode((WIDTH, HEIGHT))
clock = pygame.time.Clock()
font = pygame.font.SysFont('consolas', 24)

# Draws a gradient background from light blue to black
def draw_gradient():
    for y in range(HEIGHT):
        color = (
            LIGHT_BLUE[0] - (y / HEIGHT) * (LIGHT_BLUE[0] - BLACK[0]),
            LIGHT_BLUE[1] - (y / HEIGHT) * (LIGHT_BLUE[1] - BLACK[1]),
            LIGHT_BLUE[2] - (y / HEIGHT) * (LIGHT_BLUE[2] - BLACK[2])
        )
        pygame.draw.line(window, color, (0, y), (WIDTH, y))

# Snake class (Improved animal-like appearance)
class Snake:
    def __init__(self):
        self.body = [(5, 5), (4, 5), (3, 5)]  # Starting position
```

```
        self.direction = (1, 0)  # Initial direction (moving right)

    def move(self):
        head = self.body[0]
        new_head = (head[0] + self.direction[0], head[1] +
self.direction[1])
        self.body.insert(0, new_head)
        return self.body.pop()  # Removes last segment (unless growing)

    def grow(self):
        tail = self.body[-1]
        self.body.append(tail)

    def change_direction(self, dir):
        if (dir[0] * -1, dir[1] * -1) != self.direction:
            self.direction = dir

    def head(self):
        return self.body[0]

    def collide_self(self):
        return self.head() in self.body[1:]

    def draw(self):
        for i, segment in enumerate(self.body):
            rect = pygame.Rect(segment[0]*GRID_SIZE, segment[1]*GRID_SIZE,
GRID_SIZE, GRID_SIZE)
            if i == 0:
                # Head with slightly rounded edges for realism
                pygame.draw.rect(window, GREEN, rect, border_radius=4)
                self.draw_eyes(segment)  # Draws eyes on the head for
realism
            else:
                # Body segments as rectangles for better appearance
                pygame.draw.rect(window, GREEN, rect)

    def draw_eyes(self, head_position):
        eye_offset = GRID_SIZE // 4
        eye_radius = GRID_SIZE // 6
```

```
        left_eye = (head_position[0] * GRID_SIZE + eye_offset,
head_position[1] * GRID_SIZE - eye_offset)
        right_eye = (head_position[0] * GRID_SIZE + eye_offset * 3,
head_position[1] * GRID_SIZE - eye_offset)
        pygame.draw.circle(window, WHITE, left_eye, eye_radius)
        pygame.draw.circle(window, WHITE, right_eye, eye_radius)

# Food class
class Food:
    def __init__(self, snake):
        self.position = self.random_position(snake)

    def random_position(self, snake):
        positions = [(x, y) for x in range(GRID_WIDTH) for y in
range(GRID_HEIGHT)
                     if (x, y) not in snake.body]
        return random.choice(positions)

    def draw(self):
        rect = pygame.Rect(self.position[0]*GRID_SIZE,
self.position[1]*GRID_SIZE, GRID_SIZE, GRID_SIZE)
        pygame.draw.circle(window, YELLOW, rect.center, GRID_SIZE // 2)

# BFS pathfinding logic for AI
def bfs(start, goal, snake_body):
    queue = deque([(start, [])])
    visited = set()
    occupied = set(snake_body)

    while queue:
        current, path = queue.popleft()
        if current == goal:
            return path
        for dx, dy in [(-1,0),(1,0),(0,-1),(0,1)]:
            nx, ny = current[0]+dx, current[1]+dy
            if 0 <= nx < GRID_WIDTH and 0 <= ny < GRID_HEIGHT and (nx, ny)
not in visited and (nx, ny) not in occupied:
                visited.add((nx, ny))
                queue.append(((nx, ny), path+[(dx, dy)]))
    return []
```

```
# Draw grid lines for better visibility
def draw_grid():
    for x in range(0, WIDTH, GRID_SIZE):
        pygame.draw.line(window, WHITE, (x, 0), (x, HEIGHT))
    for y in range(0, HEIGHT, GRID_SIZE):
        pygame.draw.line(window, WHITE, (0, y), (WIDTH, y))

# Restart the game
def restart_game():
    main()

# Main game loop
def main():
    snake = Snake()
    food = Food(snake)
    score = 0
    ai_mode = False
    paused = False
    speed = FPS

    while True:
        clock.tick(speed)

        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_LEFT:
                    snake.change_direction((-1, 0))
                elif event.key == pygame.K_RIGHT:
                    snake.change_direction((1, 0))
                elif event.key == pygame.K_UP:
                    snake.change_direction((0, -1))
                elif event.key == pygame.K_DOWN:
                    snake.change_direction((0, 1))
                elif event.key == pygame.K_a:
                    ai_mode = not ai_mode
```

```
                elif event.key == pygame.K_p:
                    paused = not paused
                elif event.key == pygame.K_r:
                    restart_game()
                elif event.key == pygame.K_EQUALS or event.key ==
pygame.K_PLUS:
                    speed += 2
                elif event.key == pygame.K_MINUS:
                    speed = max(2, speed - 2)

        if paused:
            continue

        draw_gradient()

        # Display Restart Button at the top of the screen
        restart_text = font.render("Press R to Restart", True, WHITE)
        window.blit(restart_text, (WIDTH // 2 - restart_text.get_width()
// 2, 10))

        if ai_mode:
            path = bfs(snake.head(), food.position, snake.body)
            if path:
                snake.change_direction(path[0])

        tail = snake.move()

        if snake.head() == food.position:
            snake.grow()
            score += 1
            food = Food(snake)

        elif snake.collide_self() or not (0 <= snake.head()[0] <
GRID_WIDTH and 0 <= snake.head()[1] < GRID_HEIGHT):
            break

        snake.draw()
        food.draw()

        # Show score and game info
```

```
        text = font.render(f"Score: {score} | AI: {'ON' if ai_mode else
'OFF'} | Speed: {speed} | P: Pause", True, WHITE)
        window.blit(text, (10, 40))  # Move the score text below the
restart option
        pygame.display.update()

    # Display Game Over message with "Press R to Restart"
    window.fill(BLACK)
    game_over_text = font.render(f"GAME OVER! Score: {score}", True, RED)
    restart_text = font.render("Press R to Restart", True, WHITE)
    window.blit(game_over_text, (WIDTH//2 - game_over_text.get_width()//2,
HEIGHT//2 - 30))
    window.blit(restart_text, (WIDTH//2 - restart_text.get_width()//2,
HEIGHT//2 + 30))
    pygame.display.update()

    # Wait for user to press R to restart the game
    waiting_for_restart = True
    while waiting_for_restart:
        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_for_restart = False
                    restart_game()

# Run the game
if __name__ == '__main__':
    main()
```