项目背景

1.项目起源

贪吃蛇游戏项目是基于经典贪吃蛇玩法开发的现代化版本，旨在为玩家提供更丰富的游戏体验和更完善的游戏功能。项目最初源于对传统贪吃蛇游戏的怀旧与创新结合的构想。

2.项目特点

- 现代化改进：在保留经典玩法的基础上，增加多种游戏模式和自定义选项
- 加入竞速模式和历史记录等新功能，增加游戏可玩性和挑战性

1. 导入模块

```
import pygame
import random
import time
import json
from pygame.locals import *
```

`pygame`：用于创建游戏界面、处理图形和用户输入的库。
`random`：用于生成随机数，例如生成食物的位置。
`time`：用于处理时间相关的功能，例如计时和倒计时。
`json`：用于读写 `JSON` 文件，这里用于保存和读取游戏历史记录。
`pygame.locals`：包含 `Pygame` 的一些常量和函数，例如事件类型（如 `QUIT`、`MOUSEBUTTONDOWN` 等）。

2. 初始化 Pygame

```
pygame.init()
```

初始化 `Pygame` 模块，这是使用 `Pygame` 的第一步，它会加载必要的资源并准备游戏环境。

3. 游戏配置类

```
class Config:
    SCREEN_WIDTH = 1000
    SCREEN_HEIGHT = 800
    GRID_SIZE = 20
    BG_COLOR = (40, 40, 40)
    SNAKE_COLOR = (0, 255, 0)
    FOOD_COLOR = (255, 0, 0)
    OBSTACLE_COLOR = (150, 150, 150)
    TEXT_COLOR = (255, 255, 255)
    BUTTON_COLOR = (70, 130, 180)
    BUTTON_HOVER_COLOR = (100, 160, 210)
    HISTORY_FILE = "snake_history.json"
    MAX_HISTORY_ENTRIES = 100
    INPUT_BOX_COLOR = (230, 230, 230)
    INPUT_TEXT_COLOR = (0, 0, 0)
```

`Config` 类: 用于存储游戏的各种设置和常量。
`SCREEN_WIDTH` 和 `SCREEN_HEIGHT`: 定义游戏窗口的宽度和高度。
`GRID_SIZE`: 定义游戏网格的大小。
`BG_COLOR`: 背景颜色。
`SNAKE_COLOR`: 蛇的颜色。
`FOOD_COLOR`: 食物的颜色。
`OBSTACLE_COLOR`: 障碍物的颜色。
`TEXT_COLOR`: 文本颜色。
`BUTTON_COLOR` 和 `BUTTON_HOVER_COLOR`: 按钮的正常颜色和鼠标悬停时的颜色。
`HISTORY_FILE`: 保存游戏历史记录的文件名。
`MAX_HISTORY_ENTRIES`: 最多保存的历史记录条目数。
`INPUT_BOX_COLOR` 和 `INPUT_TEXT_COLOR`: 输入框的背景颜色和文本颜色。

4. 方向类

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

`Direction` 类: 用于定义贪吃蛇的移动方向。
每个方向用一个元组表示，表示在 `x` 和 `y` 轴上的移动量。

5. 按钮类

```
class Button:
    def __init__(self, x, y, width, height, text, font_size=24):
        self.rect = pygame.Rect(x, y, width, height)
        self.text = text
        self.font = pygame.font.SysFont('Arial', font_size)
        self.normal_color = Config.BUTTON_COLOR
        self.hover_color = Config.BUTTON_HOVER_COLOR
        self.text_color = Config.TEXT_COLOR
        self.is_hovered = False
```

`Button` 类: 用于创建游戏中的按钮。

`__init__` 方法初始化按钮的位置、大小、文本和字体大小。

`rect`: 按钮的矩形区域，用于检测鼠标是否悬停或点击在按钮上。

`normal_color` 和 `hover_color`: 分别定义了按钮的正常颜色和鼠标悬停时的颜色。

```
    def draw(self, surface):
        color = self.hover_color if self.is_hovered else self.normal_color
        pygame.draw.rect(surface, color, self.rect, border_radius=5)
        pygame.draw.rect(surface, (0, 0, 0), self.rect, 2, border_radius=5)

        text_surface = self.font.render(self.text, True, self.text_color)
        text_rect = text_surface.get_rect(center=self.rect.center)
        surface.blit(text_surface, text_rect)
```

`draw` 方法: 用于绘制按钮。

根据按钮是否被悬停，选择不同的颜色。

使用 `pygame.draw.rect` 绘制按钮的背景和边框。

使用 `font.render` 将按钮文本渲染为一个表面，并将其居中显示在按钮上。

```
    def check_hover(self, pos):
        self.is_hovered = self.rect.collidepoint(pos)
        return self.is_hovered
```

`check_hover` 方法: 用于检测鼠标是否悬停在按钮上。
使用 `rect.collidepoint` 检测鼠标位置是否在按钮区域内。

```
    def is_clicked(self, pos, event):
        return event.type == MOUSEBUTTONDOWN and event.button == 1 and self.rect.collidepoint(pos)
```

`is_clicked` 方法: 用于检测按钮是否被点击。
检查事件类型是否为鼠标按下（`MOUSEBUTTONDOWN`），鼠标按钮是否为左键（`event.button == 1`），并且鼠标位置是否在按钮区域内。

6. 主游戏类

```
class SnakeGame:
    def __init__(self):
        self.screen = pygame.display.set_mode((Config.SCREEN_WIDTH, Config.SCREEN_HEIGHT))
        pygame.display.set_caption('Snake Game')
        self.clock = pygame.time.Clock()
        self.font = pygame.font.SysFont('Arial', 24)
        self.big_font = pygame.font.SysFont('Arial', 36)
        self.small_font = pygame.font.SysFont('Arial', 18)
```

`SnakeGame` 类: 包含游戏的主要逻辑和状态。

`screen` : 游戏窗口。

`clock` : 用于控制游戏的帧率。

`font、big_font` 和 `small_font` : 分别定义了不同大小的字体，用于渲染文本。

```
        self.reset_game()
        self.create_buttons()
        self.history = self.load_history()
        self.history_scroll_offset = 0
```

在初始化时，调用 `reset_game` 方法重置游戏状态，调用 `create_buttons` 方法创建按钮，加载游戏历史记录，并初始化历史记录的滚动偏移量。

```
    def reset_game(self):
        self.snake = [
            [Config.SCREEN_WIDTH // 2, Config.SCREEN_HEIGHT // 2],
            [Config.SCREEN_WIDTH // 2 - Config.GRID_SIZE, Config.SCREEN_HEIGHT // 2],
            [Config.SCREEN_WIDTH // 2 - 2 * Config.GRID_SIZE, Config.SCREEN_HEIGHT // 2]
        ]
        self.direction = Direction.RIGHT
        self.next_direction = Direction.RIGHT
        self.score = 0
        self.food_count = 0
        self.obstacles = []
        self.food = self.generate_food()
        self.game_over = False
        self.paused = False
        self.game_started = False
        self.in_menu = True
        self.in_game_setup = False
        self.in_history = False
        self.in_speedrun = False
        self.difficulty = None
        self.start_time = None
        self.end_time = None
        self.pause_time = 0
        self.total_pause_time = 0
        self.speed_input = ""
        self.speed_input_active = False
        self.speed_error = ""
```

`reset_game` 方法: 用于重置游戏状态。
初始化贪吃蛇的位置和方向。
初始化分数、食物数量、障碍物列表和食物位置。
初始化游戏状态标志（如 `game_over`、`paused`、`game_started` 等）。
初始化速度输入相关变量。

```
    def create_buttons(self):
        button_width = 200
        button_height = 50
        center_x = Config.SCREEN_WIDTH // 2 - button_width // 2
```

`create_buttons` 方法: 用于创建游戏中的按钮。
定义按钮的宽度和高度，并计算按钮的中心位置。

```
        self.start_button = Button(center_x, 200, button_width, button_height, "Start Game")
        self.speedrun_button = Button(center_x, 280, button_width, button_height, "Speedrun Mode")
        self.history_button = Button(center_x, 360, button_width, button_height, "Game History")
        self.quit_button = Button(center_x, 440, button_width, button_height, "Quit")
```

创建主菜单中的“退出”按钮，位于屏幕中心水平位置，垂直位置为 440。

```
        self.easy_button = Button(center_x, 150, button_width, button_height, "Easy")
        self.medium_button = Button(center_x, 220, button_width, button_height, "Medium")
        self.hard_button = Button(center_x, 290, button_width, button_height, "Hard")
```

创建游戏设置菜单中的难度选择按钮，分别对应“简单”、“中等”和“困难”难度，垂直位置分别为 150、220 和 290。

```
        self.start_game_button = Button(center_x, 550, button_width, button_height, "Start Game")
        self.back_button = Button(center_x, 630, button_width, button_height, "Back")
```

创建“开始游戏”和“返回”按钮，用于在游戏设置菜单中启动游戏或返回主菜单，垂直位置分别为 550 和 630。

```
        self.continue_button = Button(center_x, 220, button_width, button_height, "Continue Game")
        self.resume_button = Button(center_x, 300, button_width, button_height, "Restart")
        self.restart_button = Button(center_x, 380, button_width, button_height, "New Game")
        self.menu_button = Button(center_x, 460, button_width, button_height, "Main Menu")
```

创建暂停菜单中的按钮，分别用于继续游戏、重新开始当前游戏、开始新游戏和返回主菜单，垂直位置分别为 220、300、380 和 460。

7. 加载游戏历史记录

```
    def load_history(self):
        try:
            with open(Config.HISTORY_FILE, 'r') as f:
                history = json.load(f)
                return history[-Config.MAX_HISTORY_ENTRIES:]
        except (FileNotFoundError, json.JSONDecodeError):
            return []
```

`load_history` 方法: 用于加载游戏历史记录。
尝试从 `snake_history.json` 文件中读取历史记录。
如果文件不存在或内容格式错误（如 `JSON` 解码失败），则返回一个空列表。
返回最多 `MAX_HISTORY_ENTRIES` 条记录。

8. 保存游戏历史记录

```
    def save_history(self):
        with open(Config.HISTORY_FILE, 'w') as f:
            json.dump(self.history[-Config.MAX_HISTORY_ENTRIES:], f)
```

`save_history` 方法: 用于保存游戏历史记录。
将历史记录写入 `snake_history.json` 文件。
只保存最近的 `MAX_HISTORY_ENTRIES` 条记录。

9. 添加游戏到历史记录

```
    def add_game_to_history(self):
        game_data = {
            "mode": "Speedrun" if self.in_speedrun else "Classic",
            "start_time": time.strftime("%Y-%m-%d %H:%M:%S", time.localtime(self.start_time)),
            "end_time": time.strftime("%Y-%m-%d %H:%M:%S", time.localtime(self.end_time)),
            "duration": round(self.end_time - self.start_time - self.total_pause_time, 2),
            "difficulty": self.difficulty if not self.in_speedrun else "N/A",
            "speed": self.speed,
            "score": self.score
        }
        self.history.append(game_data)
        self.save_history()
```

`add_game_to_history` 方法: 用于将当前游戏的记录添加到历史记录中。
创建一个字典 `game_data`，包含游戏模式、开始时间、结束时间、持续时间、难度、速度和分数。
将该记录添加到 `self.history` 列表中，并调用 `save_history` 方法保存到文件。

10. 生成食物

```
    def generate_food(self):
        while True:
            x = random.randint(0, (Config.SCREEN_WIDTH - Config.GRID_SIZE) // Config.GRID_SIZE) * Config.GRID_SIZE
            y = random.randint(0, (Config.SCREEN_HEIGHT - Config.GRID_SIZE) // Config.GRID_SIZE) * Config.GRID_SIZE
            if [x, y] not in self.snake and [x, y] not in self.obstacles:
                return [x, y]
```

`generate_food` 方法: 用于生成食物的位置。
使用 `random.randint` 随机生成一个网格位置。
确保生成的位置不在蛇的身体上或障碍物上。
返回食物的位置。

11. 生成障碍物

```
    def generate_obstacles(self, count):
        self.obstacles = []
        for _ in range(count):
            while True:
                x = random.randint(3, (Config.SCREEN_WIDTH - Config.GRID_SIZE) // Config.GRID_SIZE - 3) * Config.GRID_SIZE
                y = random.randint(3, (Config.SCREEN_HEIGHT - Config.GRID_SIZE) // Config.GRID_SIZE - 3) * Config.GRID_SIZE
```

`generate_obstacles` 方法: 用于生成指定数量的障碍物。

清空当前的障碍物列表。

在指定范围内随机生成障碍物的位置，避免生成在蛇的头部、尾部或食物位置。

```
            if self.difficulty == "Hard" and random.random() < 0.3:
                wall_length = random.randint(2, 3)
                wall_direction = random.choice(["horizontal", "vertical"])

                valid = True
                temp_obstacles = []
                for i in range(wall_length):
                    if wall_direction == "horizontal":
                        new_x = x + i * Config.GRID_SIZE
                        new_y = y
                    else:
                        new_x = x
                        new_y = y + i * Config.GRID_SIZE

                    if (new_x >= Config.SCREEN_WIDTH or new_y >= Config.SCREEN_HEIGHT or
                        [new_x, new_y] in self.snake or [new_x, new_y] == self.food or
                        [new_x, new_y] in self.obstacles):
                        valid = False
                        break

                    temp_obstacles.append([new_x, new_y])

                if valid:
                    self.obstacles.extend(temp_obstacles)
                    break
            else:
                if [x, y] not in self.snake and [x, y] != self.food and [x, y] not in self.obstacles:
                    self.obstacles.append([x, y])
                    break
```

如果当前难度为“困难”，有一定概率生成“墙”（连续的障碍物）。
随机选择墙的方向（水平或垂直）和长度（2 或 3）。
检查生成的墙是否超出屏幕范围或与蛇、食物或现有障碍物冲突。
如果生成的墙有效，则将其添加到障碍物列表中。
如果不生成墙，则直接生成单个障碍物。

12. 移动蛇

```
    def move_snake(self):
        head = self.snake[0].copy()
        head[0] += self.direction[0] * Config.GRID_SIZE
        head[1] += self.direction[1] * Config.GRID_SIZE
```

`move_snake` 方法: 用于移动蛇。

获取蛇头的当前位置，并根据当前方向计算新的蛇头位置。

```
        if (head[0] < 0 or head[0] >= Config.SCREEN_WIDTH or
            head[1] < 0 or head[1] >= Config.SCREEN_HEIGHT or
            head in self.snake or head in self.obstacles):
            self.game_over = True
            self.end_time = time.time()
            self.add_game_to_history()
            return
```

检查蛇头是否撞到屏幕边界、自身身体或障碍物。

如果发生碰撞，设置 `game_over` 为 `True`，记录游戏结束时间，并将游戏记录添加到历史记录中。

```
        self.snake.insert(0, head)
```

将新的蛇头位置插入到蛇的头部。

```
        if head == self.food:
            self.score += 1
            self.food_count += 1
            self.food = self.generate_food()
```

如果蛇头吃到食物：

增加分数。

增加食物计数。

生成新的食物。

```
            if self.in_speedrun and self.food_count % 5 == 0:
                self.speed += 1
```

如果处于速度挑战模式，并且食物计数达到 5 的倍数，增加游戏速度。

```
            if not self.in_speedrun:
                if self.difficulty == "Medium":
                    self.generate_obstacles(3)
                elif self.difficulty == "Hard":
                    self.generate_obstacles(5)
```

如果不是速度挑战模式：

如果难度为“中等”，生成 3 个障碍物。

如果难度为“困难”，生成 5 个障碍物。

```
        else:
            self.snake.pop()
```

如果蛇头没有吃到食物，移除蛇尾的最后一节。

13. 事件处理

```
    def handle_events(self):
        mouse_pos = pygame.mouse.get_pos()

        for event in pygame.event.get():
            if event.type == QUIT:
                pygame.quit()
                return False
```

`handle_events` 方法: 用于处理游戏中的所有事件。
获取鼠标的位置。
遍历所有事件，如果事件类型是 `QUIT` （用户点击关闭按钮），则退出 `Pygame` 并返回 `False` ，表示游戏结束。

```
            if self.in_menu:
                self.handle_menu_events(event, mouse_pos)
            elif self.in_game_setup:
                self.handle_setup_events(event, mouse_pos)
            elif self.in_history:
                self.handle_history_events(event, mouse_pos)
            elif self.paused:
                self.handle_pause_events(event, mouse_pos)
            elif not self.game_over and self.game_started:
                self.handle_game_events(event)
            elif self.game_over:
                if event.type == KEYDOWN and event.key == K_r:
                    self.reset_game()
                    if self.in_speedrun:
                        self.start_speedrun()
                    else:
                        self.in_menu = True
```

根据当前游戏状态（主菜单、游戏设置、历史记录、暂停、游戏进行中、游戏结束），调用相应的事件处理方法。
如果游戏结束，按下 R 键可以重置游戏并返回主菜单或重新开始速度挑战模式。

14. 主菜单事件处理

```
    def handle_menu_events(self, event, mouse_pos):
        self.start_button.check_hover(mouse_pos)
        self.speedrun_button.check_hover(mouse_pos)
        self.history_button.check_hover(mouse_pos)
        self.quit_button.check_hover(mouse_pos)
```

检测鼠标是否悬停在主菜单的按钮上。

```
        if event.type == MOUSEBUTTONDOWN and event.button == 1:
            if self.start_button.rect.collidepoint(mouse_pos):
                self.in_menu = False
                self.in_game_setup = True
            elif self.speedrun_button.rect.collidepoint(mouse_pos):
                self.start_speedrun()
            elif self.history_button.rect.collidepoint(mouse_pos):
                self.in_menu = False
                self.in_history = True
            elif self.quit_button.rect.collidepoint(mouse_pos):
                pygame.quit()
                return False
```

如果鼠标点击主菜单的按钮：

点击“开始游戏”按钮，进入游戏设置菜单。

点击“速度挑战模式”按钮，直接开始速度挑战模式。

点击“游戏历史”按钮，进入历史记录界面。

点击“退出”按钮，退出游戏。

15. 游戏设置事件处理

```
    def handle_setup_events(self, event, mouse_pos):
        self.easy_button.check_hover(mouse_pos)
        self.medium_button.check_hover(mouse_pos)
        self.hard_button.check_hover(mouse_pos)

        self.start_game_button.check_hover(mouse_pos)
        self.back_button.check_hover(mouse_pos)
```

检测鼠标是否悬停在游戏设置菜单的按钮上。

```
        if event.type == MOUSEBUTTONDOWN and event.button == 1:
            if self.easy_button.rect.collidepoint(mouse_pos):
                self.difficulty = "Easy"
            elif self.medium_button.rect.collidepoint(mouse_pos):
                self.difficulty = "Medium"
            elif self.hard_button.rect.collidepoint(mouse_pos):
                self.difficulty = "Hard"
```

如果鼠标点击游戏设置菜单的按钮：

点击“简单”按钮，设置难度为“简单”。

点击“中等”按钮，设置难度为“中等”。

点击“困难”按钮，设置难度为“困难”。

```
                if self.speed_input_rect.collidepoint(mouse_pos):
                    self.speed_input_active = True
                    self.speed_error = ""
                else:
                    self.speed_input_active = False
```

如果鼠标点击输入框，激活输入框并清空错误信息；否则，取消激活输入框。

```
                if self.start_game_button.rect.collidepoint(mouse_pos) and self.difficulty:
                    self.start_game()
                elif self.back_button.rect.collidepoint(mouse_pos):
                    self.in_game_setup = False
                    self.in_menu = True
```

如果点击“开始游戏”按钮且已选择难度，开始游戏。

如果点击“返回”按钮，返回主菜单。

```
        if event.type == KEYDOWN and self.speed_input_active:
            if event.key == K_BACKSPACE:
                self.speed_input = self.speed_input[:-1]
            elif event.key == K_RETURN:
                try:
                    speed = int(self.speed_input)
                    if speed >= 1:
                        self.speed = speed
                        self.speed_error = ""
                    else:
                        self.speed_error = "Speed must be at least 1"
                except ValueError:
                    self.speed_error = "Please enter a valid number"
                self.speed_input_active = False
            elif event.unicode.isdigit():
                self.speed_input += event.unicode
```

如果输入框激活且按下键盘：
按下 `Backspace`，删除最后一个字符。
按下 `Enter`，尝试将输入转换为整数并验证：
如果速度大于等于 1，设置速度并清空错误信息。
如果速度小于 1，显示错误信息。
如果输入无效，显示错误信息。
如果输入的是数字，将其添加到输入框中。

16. 历史记录事件处理

```
    def handle_history_events(self, event, mouse_pos):
        if event.type == MOUSEBUTTONDOWN:
            if event.button == 4:  # Mouse wheel up
                self.history_scroll_offset = max(0, self.history_scroll_offset - 1)
            elif event.button == 5:  # Mouse wheel down
                max_offset = max(0, len(self.history) - 10)
                self.history_scroll_offset = min(max_offset, self.history_scroll_offset + 1)
            elif event.button == 1:  # Left click
                self.in_history = False
                self.in_menu = True
```

如果鼠标点击历史记录界面：

滚轮向上滚动，减少滚动偏移量。

滚轮向下滚动，增加滚动偏移量。

左键点击，返回主菜单。

17. 暂停菜单事件处理

```
    def handle_pause_events(self, event, mouse_pos):
        self.continue_button.check_hover(mouse_pos)
        self.resume_button.check_hover(mouse_pos)
        self.restart_button.check_hover(mouse_pos)
        self.menu_button.check_hover(mouse_pos)
```

检测鼠标是否悬停在暂停菜单的按钮上。

```
        if event.type == MOUSEBUTTONDOWN and event.button == 1:
            if self.continue_button.rect.collidepoint(mouse_pos):
                self.unpause_game()
            elif self.resume_button.rect.collidepoint(mouse_pos):
                self.unpause_game()
            elif self.restart_button.rect.collidepoint(mouse_pos):
                self.reset_game()
                if self.in_speedrun:
                    self.start_speedrun()
                else:
                    self.start_game()
            elif self.menu_button.rect.collidepoint(mouse_pos):
                self.reset_game()
                self.in_menu = True
```

如果鼠标点击暂停菜单的按钮：

点击“继续游戏”按钮，恢复游戏。

点击“重新开始”按钮，重置游戏并重新开始当前模式。

点击“返回主菜单”按钮，重置游戏并返回主菜单。

18. 游戏事件处理

```
    def handle_game_events(self, event):
        if event.type == KEYDOWN:
            if event.key == K_p:
                self.pause_game()
            elif event.key == K_b:  # Back to menu
                self.reset_game()
                self.in_menu = True
            elif event.key == K_n:  # New game same mode
                self.reset_game()
                if self.in_speedrun:
                    self.start_speedrun()
                else:
                    self.start_game()
            elif not self.paused:
                if event.key in (K_UP, K_w) and self.direction != Direction.DOWN:
                    self.next_direction = Direction.UP
                elif event.key in (K_DOWN, K_s) and self.direction != Direction.UP:
                    self.next_direction = Direction.DOWN
                elif event.key in (K_LEFT, K_a) and self.direction != Direction.RIGHT:
                    self.next_direction = Direction.LEFT
                elif event.key in (K_RIGHT, K_d) and self.direction != Direction.LEFT:
                    self.next_direction = Direction.RIGHT
```

如果用户按下 K_RIGHT 或 K_d（即右箭头键或 D 键），并且当前方向不是向左（避免蛇直接反向移动），则将蛇的下一个方向设置为向右。

19. 开始游戏

开始普通游戏

```
    def start_game(self):
        try:
            speed = int(self.speed_input)
            if speed < 1:
                self.speed_error = "Speed must be at least 1"
                return
            self.speed = speed
        except ValueError:
            self.speed_error = "Please enter a valid number"
            return
```

`start_game` 方法: 用于开始普通模式的游戏。
尝试将用户输入的速度值转换为整数。
如果速度小于 1，提示用户速度必须至少为 1。
如果输入无效（例如不是数字），提示用户输入有效的数字。

```
        if self.difficulty == "Easy":
            self.obstacles = []
        elif self.difficulty == "Medium":
            self.generate_obstacles(5)
        elif self.difficulty == "Hard":
            self.generate_obstacles(10)
```

根据选择的难度生成障碍物：

“简单”难度：不生成障碍物。

“中等”难度：生成 5 个障碍物。

“困难”难度：生成 10 个障碍物。

```
        self.in_game_setup = False
        self.game_started = True
        self.in_speedrun = False
        self.start_time = time.time()
        self.countdown(3)
```

设置游戏状态：

退出游戏设置菜单。

设置游戏为已开始状态。

设置当前模式为普通模式（非速度挑战模式）。

记录游戏开始时间。

开始一个 3 秒的倒计时。

开始速度挑战模式

```
    def start_speedrun(self):
        self.obstacles = []
        self.speed = 5
        self.in_menu = False
        self.game_started = True
        self.in_speedrun = True
        self.start_time = time.time()
        self.countdown(3)
```

`start_speedrun` 方法: 用于开始速度挑战模式的游戏。

清空障碍物列表。

设置初始速度为 5。

设置游戏状态为非主菜单、已开始、速度挑战模式。

记录游戏开始时间。

开始一个 3 秒的倒计时。

20. 暂停和恢复游戏

```
    def pause_game(self):
        self.paused = True
        self.pause_time = time.time()
```

`pause_game` 方法: 用于暂停游戏。
设置 `paused` 标志为 `True` 。
记录暂停时间。

```
    def unpause_game(self):
        self.paused = False
        self.total_pause_time += time.time() - self.pause_time
```

`unpause_game` 方法: 用于恢复游戏。
设置 `paused` 标志为 `False` 。
更新总暂停时间，累加从暂停到恢复的时间。

21. 倒计时

```
    def countdown(self, seconds):
        for i in range(seconds, 0, -1):
            self.screen.fill(Config.BG_COLOR)
            countdown_text = self.big_font.render(str(i), True, (255, 0, 0))
            self.screen.blit(countdown_text,
                             (Config.SCREEN_WIDTH // 2 - countdown_text.get_width() // 2,
                              Config.SCREEN_HEIGHT // 2 - countdown_text.get_height() // 2))
            pygame.display.flip()
            pygame.time.wait(1000)
```

`countdown` 方法: 用于显示倒计时。

填充屏幕背景颜色。

渲染倒计时数字（从传入的秒数开始递减）。

将倒计时数字居中显示在屏幕上。

更新屏幕显示。

每秒暂停一次，直到倒计时结束。

22. 更新游戏状态

```
    def update(self):
        if not self.game_started or self.paused or self.game_over:
            return
```

`update` 方法: 用于更新游戏状态。
如果游戏尚未开始、处于暂停状态或已经结束，则不进行任何更新。

```
        self.direction = self.next_direction
        self.move_snake()
```

更新蛇的当前方向为下一个方向。
调用 `move_snake` 方法移动蛇。

23. 绘制游戏界面

绘制主菜单

```
    def draw_menu(self):
        title = self.big_font.render('Snake Game', True, Config.TEXT_COLOR)
        self.screen.blit(title, (Config.SCREEN_WIDTH // 2 - title.get_width() // 2, 100))
```

`draw_menu` 方法: 用于绘制主菜单界面。
渲染标题“Snake Game”并居中显示。

```
        self.start_button.draw(self.screen)
        self.speedrun_button.draw(self.screen)
        self.history_button.draw(self.screen)
        self.quit_button.draw(self.screen)
```

绘制主菜单中的所有按钮。

绘制游戏设置菜单

```
    def draw_game_setup(self):
        title = self.big_font.render('Game Setup', True, Config.TEXT_COLOR)
        self.screen.blit(title, (Config.SCREEN_WIDTH // 2 - title.get_width() // 2, 50))
```

`draw_game_setup` 方法: 用于绘制游戏设置菜单。
渲染标题“Game Setup”并居中显示。

```
        difficulty_text = self.font.render('Select Difficulty:', True, Config.TEXT_COLOR)
        self.screen.blit(difficulty_text, (Config.SCREEN_WIDTH // 2 - difficulty_text.get_width() // 2, 120))
```

渲染并显示“选择难度”文本。

```
        self.easy_button.draw(self.screen)
        self.medium_button.draw(self.screen)
        self.hard_button.draw(self.screen)
```

绘制难度选择按钮。

```
        if self.difficulty:
            diff_rect = None
            if self.difficulty == "Easy":
                diff_rect = self.easy_button.rect
            elif self.difficulty == "Medium":
                diff_rect = self.medium_button.rect
            elif self.difficulty == "Hard":
                diff_rect = self.hard_button.rect

            if diff_rect:
                pygame.draw.rect(self.screen, (255, 255, 0), diff_rect, 3, border_radius=5)
```

如果已选择难度，用黄色边框突出显示所选难度按钮。

```
        speed_text = self.font.render('Enter Speed (1+):', True, Config.TEXT_COLOR)
        self.screen.blit(speed_text, (Config.SCREEN_WIDTH // 2 - speed_text.get_width() // 2, 340))
```

渲染并显示“输入速度（1+）”文本。

```
        pygame.draw.rect(self.screen,
                         Config.INPUT_BOX_COLOR if self.speed_input_active else (200, 200, 200),
                         self.speed_input_rect, border_radius=5)
        pygame.draw.rect(self.screen, (0, 0, 0), self.speed_input_rect, 2, border_radius=5)
```

绘制输入框，根据是否激活改变颜色，并绘制边框。

```
        input_surface = self.font.render(self.speed_input, True, Config.INPUT_TEXT_COLOR)
        self.screen.blit(input_surface, (self.speed_input_rect.x + 10, self.speed_input_rect.y + 15))
```

渲染并显示用户输入的速度值。

```
        if hasattr(self, 'speed') and not self.speed_input_active and self.speed_input:
            speed_display = self.font.render(f"Current: {self.speed}", True, Config.TEXT_COLOR)
            self.screen.blit(speed_display, (self.speed_input_rect.x, self.speed_input_rect.y + 50))
```

如果已设置速度且输入框未激活，显示当前速度。

```
        if self.speed_error:
            error_text = self.font.render(self.speed_error, True, (255, 0, 0))
            self.screen.blit(error_text, (self.speed_input_rect.x, self.speed_input_rect.y + 80))
```

如果有速度输入错误，显示错误信息。

```
        self.start_game_button.draw(self.screen)
        self.back_button.draw(self.screen)
```

绘制“开始游戏”和“返回”按钮。

24. 绘制游戏历史记录

```
    def draw_history(self):
        title = self.big_font.render('Game History', True, Config.TEXT_COLOR)
        self.screen.blit(title, (Config.SCREEN_WIDTH // 2 - title.get_width() // 2, 50))
```

`draw_history` 方法: 用于绘制游戏历史记录界面。

渲染标题“Game History”并居中显示。

```
        if not self.history:
            no_history = self.font.render('No game history yet', True, Config.TEXT_COLOR)
            self.screen.blit(no_history, (Config.SCREEN_WIDTH // 2 - no_history.get_width() // 2, 150))
        else:
```

如果没有历史记录，显示“没有游戏历史”。

如果有历史记录，继续绘制历史记录表格。

```
        table_x = 50
        table_y = 120
        row_height = 30
        visible_rows = 15
        column_widths = [100, 180, 80, 100, 60, 60]  # Mode, Start Time, Duration, Difficulty, Speed, Score
```

定义表格的起始位置、行高、可见行数和各列宽度。

```
            headers = ["Mode", "Start Time", "Duration", "Difficulty", "Speed", "Score"]
            header_y = table_y
            for i, (header, width) in enumerate(zip(headers, column_widths)):
                header_surface = self.font.render(header, True, Config.TEXT_COLOR)
                header_x = table_x + sum(column_widths[:i]) + i * 10
                self.screen.blit(header_surface, (header_x + 5, header_y + 5))
```

渲染表头，包括“模式”、“开始时间”、“持续时间”、“难度”、“速度”和“分数”。

```
                pygame.draw.line(self.screen, Config.TEXT_COLOR,
                                 (header_x, header_y + row_height),
                                 (header_x + width, header_y + row_height), 2)
```

绘制表头的水平分隔线。

```
            for i in range(len(column_widths) + 1):
                divider_x = table_x + sum(column_widths[:i]) + i * 10
                pygame.draw.line(self.screen, Config.TEXT_COLOR,
                                 (divider_x, table_y),
                                 (divider_x, table_y + visible_rows * row_height), 1)
```

绘制表格的垂直分隔线。

```
            visible_history = self.history[self.history_scroll_offset:self.history_scroll_offset+visible_rows]
            for row_idx, game in enumerate(visible_history):
                row_y = table_y + (row_idx + 1) * row_height

                if row_idx % 2 == 0:
                    row_color = (60, 60, 60)
                else:
                    row_color = (80, 80, 80)

                pygame.draw.rect(self.screen, row_color,
                                 (table_x, row_y, sum(column_widths) + (len(column_widths)-1)*10, row_height))
```

根据滚动偏移量获取可见的历史记录条目。

交替绘制行背景颜色，以提高可读性。

```
                cells = [
                    game["mode"],
                    game["start_time"],
                    f"{game['duration']}s",
                    game["difficulty"],
                    str(game["speed"]),
                    str(game["score"])
                ]

                for i, (cell, width) in enumerate(zip(cells, column_widths)):
                    cell_x = table_x + sum(column_widths[:i]) + i * 10 + 5
                    cell_surface = self.small_font.render(cell, True, Config.TEXT_COLOR)
                    self.screen.blit(cell_surface, (cell_x, row_y + 5))
```

渲染每一行的单元格内容，包括模式、开始时间、持续时间、难度、速度和分数。

```
        if len(self.history) > visible_rows:
            scroll_text = self.font.render(
                f"Showing {self.history_scroll_offset+1}-{min(self.history_scroll_offset+visible_rows, len(self.history))} of {len(self.history)}",
                True, Config.TEXT_COLOR)
            self.screen.blit(scroll_text, (Config.SCREEN_WIDTH // 2 - scroll_text.get_width() // 2, table_y + (visible_rows + 1) * row_height))
```

如果历史记录条目超过可见行数，显示滚动指示器。

```
        back_text = self.font.render('Click to return or use mouse wheel to scroll', True, Config.TEXT_COLOR)
        self.screen.blit(back_text, (Config.SCREEN_WIDTH // 2 - back_text.get_width() // 2, table_y + (visible_rows + 2) * row_height))
```

显示返回提示和滚动说明。

25. 绘制游戏界面

绘制游戏界面

```
    def draw_game(self):
        for obstacle in self.obstacles:
            pygame.draw.rect(self.screen, Config.OBSTACLE_COLOR,
                             (*obstacle, Config.GRID_SIZE, Config.GRID_SIZE))
```

绘制所有障碍物。

```
        for segment in self.snake:
            pygame.draw.rect(self.screen, Config.SNAKE_COLOR,
                             (*segment, Config.GRID_SIZE, Config.GRID_SIZE))
```

绘制蛇的每一节。

```
            pygame.draw.rect(self.screen, Config.FOOD_COLOR,
                             (*self.food, Config.GRID_SIZE, Config.GRID_SIZE))
```

绘制食物。

```
        current_time = time.time() - self.start_time - self.total_pause_time
        time_text = self.font.render(f'Time: {current_time:.1f}s', True, Config.TEXT_COLOR)
        self.screen.blit(time_text, (10, 10))
```

显示当前游戏时间。

```
        score_text = self.font.render(f'Score: {self.score}', True, Config.TEXT_COLOR)
        self.screen.blit(score_text, (10, 40))
```

显示当前分数。

```
        speed_text = self.font.render(f'Speed: {self.speed}', True, Config.TEXT_COLOR)
        self.screen.blit(speed_text, (10, 70))
```

显示当前游戏速度。

```
        mode_text = self.font.render(f'Mode: {"Speedrun" if self.in_speedrun else self.difficulty}', True, Config.TEXT_COLOR)
        self.screen.blit(mode_text, (10, 100))
```

显示当前游戏模式（速度挑战模式或普通模式及难度）。

```
        if self.in_speedrun:
            food_text = self.font.render(f'Food: {self.food_count} (Next speed at {(self.food_count // 5 + 1) * 5})',
                                         True, Config.TEXT_COLOR)
            self.screen.blit(food_text, (10, 130))
```

如果是速度挑战模式，显示食物计数和下一次速度提升的条件。

26. 绘制暂停菜单

```
    def draw_pause_menu(self):
        overlay = pygame.Surface((Config.SCREEN_WIDTH, Config.SCREEN_HEIGHT), pygame.SRCALPHA)
        overlay.fill((0, 0, 0, 128))
        self.screen.blit(overlay, (0, 0))
```

绘制半透明的覆盖层，使暂停菜单更加突出。

```
        pause_text = self.big_font.render('GAME PAUSED', True, (255, 255, 0))
        self.screen.blit(pause_text,
                         (Config.SCREEN_WIDTH // 2 - pause_text.get_width() // 2,
                          Config.SCREEN_HEIGHT // 2 - 150))
```

显示“GAME PAUSED”文本。

```
        self.continue_button.draw(self.screen)
        self.resume_button.draw(self.screen)
        self.restart_button.draw(self.screen)
        self.menu_button.draw(self.screen)
```

绘制暂停菜单中的所有按钮。

27. 绘制游戏结束界面

```
    def draw_game_over(self):
        overlay = pygame.Surface((Config.SCREEN_WIDTH, Config.SCREEN_HEIGHT), pygame.SRCALPHA)
        overlay.fill((0, 0, 0, 128))
        self.screen.blit(overlay, (0, 0))
```

绘制半透明的覆盖层。

```
        game_over = self.big_font.render('GAME OVER', True, (255, 0, 0))
        self.screen.blit(game_over, (Config.SCREEN_WIDTH // 2 - game_over.get_width() // 2, 150))
```

显示“GAME OVER”文本。

```
        time_played = self.end_time - self.start_time - self.total_pause_time
        stats = [
            f'Final Score: {self.score}',
            f'Time Played: {time_played:.1f}s',
            f'Mode: {"Speedrun" if self.in_speedrun else self.difficulty}',
            f'Speed: {self.speed}',
            '',
            'Press R to return to menu'
        ]
```

计算游戏总时长（扣除暂停时间）。

创建一个包含游戏统计信息的列表，包括最终分数、游戏时长、模式、速度，以及提示用户按 R 键返回菜单。

```
        y_offset = 220
        for stat in stats:
            stat_text = self.font.render(stat, True, Config.TEXT_COLOR)
            self.screen.blit(stat_text, (Config.SCREEN_WIDTH // 2 - stat_text.get_width() // 2, y_offset))
            y_offset += 40
```

遍历统计信息列表，逐行渲染并居中显示在屏幕上，每次增加垂直偏移量以分隔各行。

28. 游戏主循环

```
    def run(self):
        running = True
        while running:
            running = self.handle_events()
            self.update()
            self.draw()
            self.clock.tick(self.speed if self.game_started and not self.paused else 60)
```

`run` 方法: 游戏的主循环。
初始化 `running` 为 `True`，表示游戏正在运行。
在 `while` 循环中：
调用 `handle_events` 方法处理事件，如果返回 False，则退出循环。
调用 `update` 方法更新游戏状态。
调用 `draw` 方法绘制游戏界面。
使用 `clock.tick` 控制帧率：
如果游戏已开始且未暂停，帧率设置为当前游戏速度。
否则，帧率固定为 60。

29. 游戏入口

```
if __name__ == '__main__':
    game = SnakeGame()
    game.run()
```

如果直接运行此脚本：
创建一个 `SnakeGame` 实例。
调用 `run` 方法启动游戏主循环。