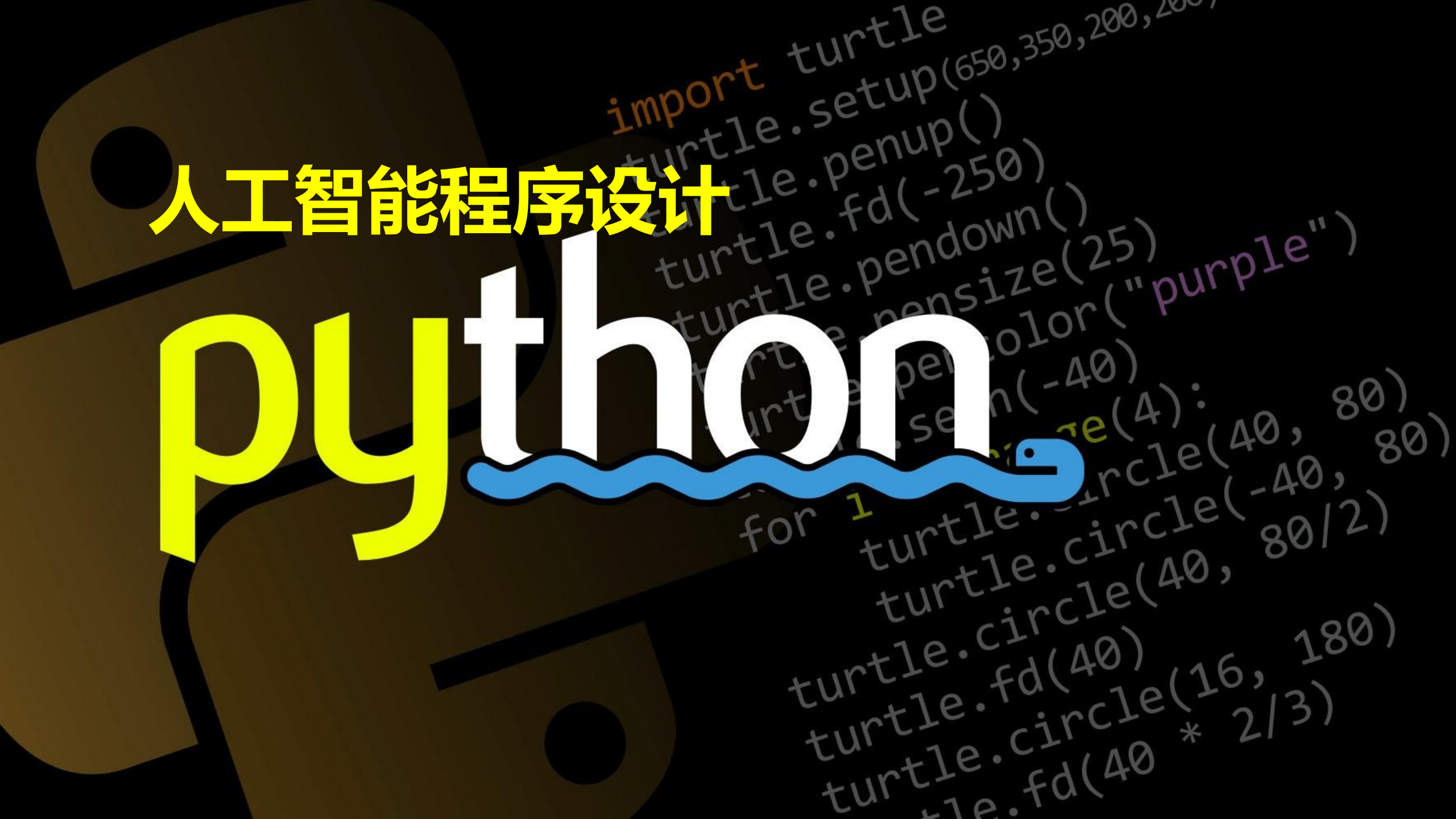
人工智能程序设计


python
import turtle
turtle.setup(650,350,200,200)
turtle.penup()
turtle.fd(-250)
turtle.pendown()
turtle.pensize(25)
turtle.pencolor("purple")
turtle.seth(-40)
for 1 in range(4):
turtle.circle(40, 80)
turtle.circle(-40, 80)
turtle.fd(40)
turtle.circle(40, 80/2)
turtle.fd(40)
turtle.circle(16, 180)
turtle.fd(40 * 2/3)

人工智能程序设计

9.4 更多经典小游戏

北京石油化工学院 人工智能研究院

刘 强

本节概述

在已有的贪吃蛇、打砖块基础上，继续扩展更多经典小游戏：

- **乒乓球（Pong）**：双人对战，输入控制与碰撞检测
- **扫雷**：网格数据结构与递归展开算法
- **其它经典题材**：Flappy Bird、Space Invaders、推箱子、2048、俄罗斯方块

9.4.1 乒乓球（Pong）双人对战

训练输入控制、碰撞检测与得分重置的完整游戏循环：

- 适合双人互动演示
- 强调低延迟与流畅帧率下的交互体验

乒乓球：需求分析

明确游戏的核心功能需求：

- **双人控制**：左挡板 `W/S`，右挡板 `↑/↓`
- **球体弹跳**：击中挡板反弹，出界判分并重置
- **帧率要求**：平滑帧率与清晰配色，便于演示

乒乓球：系统设计

采用模块化设计，便于后续扩展：

- **核心模块**：输入处理、物理更新、碰撞检测、得分重置、渲染
- **数据结构**：球坐标/速度（列表），挡板位置（标量），`pygame.Rect`碰撞
- **帧率控制**：`clock.tick(60)` 保持流畅

乒乓球：编码实现概览

代码分为4个关键片段：

- 初始化与常量
- 输入处理
- 物理更新与碰撞
- 渲染与帧率

初始化与常量

导入库并设置游戏基础参数：

```
import pygame
pygame.init()
screen = pygame.display.set_mode((800, 480))
clock = pygame.time.Clock()

WHITE, BLUE = (255, 255, 255), (50, 120, 220)
ball_pos, ball_speed, ball_radius = [400, 240], [6, 4], 10
paddle_w, paddle_h, paddle_speed = 12, 80, 6
left_y, right_y = 200, 200
```

输入处理

读取键盘状态，控制左右挡板移动：

使用 `pygame.key.get_pressed()` 实现持续按键检测。

```
keys = pygame.key.get_pressed()
if keys[pygame.K_w]: left_y -= paddle_speed
if keys[pygame.K_s]: left_y += paddle_speed
if keys[pygame.K_UP]: right_y -= paddle_speed
if keys[pygame.K_DOWN]: right_y += paddle_speed
```

物理更新与碰撞

球的移动与边界、挡板碰撞检测：

```
ball_pos[0] += ball_speed[0]
ball_pos[1] += ball_speed[1]
if ball_pos[1] <= ball_radius or ball_pos[1] >= 480 - ball_radius:
    ball_speed[1] = -ball_speed[1]

left_rect = pygame.Rect(40, left_y, paddle_w, paddle_h)
right_rect = pygame.Rect(800 - 40 - paddle_w, right_y, paddle_w, paddle_h)
ball_rect = pygame.Rect(ball_pos[0] - ball_radius, ball_pos[1] - ball_radius,
                        ball_radius * 2, ball_radius * 2)
```

挡板碰撞反弹

使用 `colliderect()` 检测球与挡板碰撞：

条件 `ball_speed[0] < 0` 防止球在挡板内部反复反弹。

```
if ball_rect.colliderect(left_rect) and ball_speed[0] < 0:
    ball_speed[0] = -ball_speed[0]
if ball_rect.colliderect(right_rect) and ball_speed[0] > 0:
    ball_speed[0] = -ball_speed[0]
```

渲染与帧率

清屏、绘制游戏元素、更新显示：

```
screen.fill((20, 20, 20))
pygame.draw.rect(screen, BLUE, left_rect)
pygame.draw.rect(screen, BLUE, right_rect)
pygame.draw.circle(screen, WHITE, (int(ball_pos[0]), int(ball_pos[1])), ball_radius)
pygame.display.flip()
clock.tick(60)
```

9.4.2 扫雷：网格与递归展开

训练二维网格数据结构、邻域遍历与递归展开算法：

- 理解DFS/BFS在游戏中的应用
- 掌握网格数据的初始化与遍历

扫雷：需求分析

明确扫雷游戏的核心功能：

- **布雷与显示**：随机布雷，显示周围雷数
- **翻开与标记**：左键翻开，若为雷则失败；右键标记旗子
- **递归展开**：数字为0时自动展开周围空白
- **胜利条件**：全部非雷格子翻开即胜利

扫雷：系统设计

数据结构与核心流程：

- **数据结构**：
  - `mine_mask`：存储雷的布置
  - `grid`：存储周围雷数
  - `revealed`：存储翻开状态
  - `flagged`：存储标记状态
- **核心流程**：初始化布雷→计算周围雷数→翻开/标记→判定胜负

扫雷：编码实现概览

代码分为3个关键片段：

- 初始化与布雷
- 周围雷数计算
- 递归展开

初始化与布雷

定义网格并随机布置地雷：

```
import random

ROWS, COLS = 10, 10
MINES = 15

grid = [[0 for _ in range(COLS)] for _ in range(ROWS)]
mine_mask = [[False for _ in range(COLS)] for _ in range(ROWS)]
revealed = [[False for _ in range(COLS)] for _ in range(ROWS)]
flagged = [[False for _ in range(COLS)] for _ in range(ROWS)]
```

随机布雷

在随机位置放置地雷：

使用 `while` 循环确保布置指定数量的雷。

```
count = 0
while count < MINES:
    r = random.randint(0, ROWS - 1)
    c = random.randint(0, COLS - 1)
    if not mine_mask[r][c]:
        mine_mask[r][c] = True
        count += 1
```

周围雷数计算

遍历每个格子，统计8邻域内雷的数量：

```
dirs = [(-1, -1), (-1, 0), (-1, 1),
        (0, -1),           (0, 1),
        (1, -1),  (1, 0), (1, 1)]

for r in range(ROWS):
    for c in range(COLS):
        if mine_mask[r][c]:
            grid[r][c] = -1  # 雷标记为-1
            continue
```

邻域统计

统计当前格子周围的雷数：

使用8方向偏移量遍历邻域。

```
around = 0
for dr, dc in dirs:
    nr, nc = r + dr, c + dc
    if 0 <= nr < ROWS and 0 <= nc < COLS and mine_mask[nr][nc]:
        around += 1
grid[r][c] = around
```

递归展开

翻开数字为0的格子时，自动展开周围区域：

这是典型的DFS（深度优先搜索）应用。

```
def flood_reveal(r, c):
    if revealed[r][c] or mine_mask[r][c] or flagged[r][c]:
        return
    revealed[r][c] = True
    if grid[r][c] == 0:
        for dr, dc in dirs:
            nr, nc = r + dr, c + dc
            if 0 <= nr < ROWS and 0 <= nc < COLS:
                flood_reveal(nr, nc)
```

9.4.3 其它可扩展题材

除了乒乓球和扫雷，还有许多经典小游戏适合作为练习项目：

- **Flappy Bird**：重力跳跃、管道生成
- **Space Invaders**：敌阵移动、射击碰撞
- **推箱子**：网格移动、关卡设计
- **2048**：网格滑动与合并
- **俄罗斯方块**：方块下落、行消除

Flappy Bird

2013年由越南开发者阮哈东发布，简单却极具挑战性：

- **玩法**：点击跳跃，穿过管道缝隙
- **核心机制**：
  - 重力模拟：每帧给小鸟向下加速度
  - 跳跃响应：按键给小鸟向上瞬时速度
  - 管道生成：定时在右侧生成随机高度管道
  - 碰撞检测：小鸟与管道、地面碰撞判定

Space Invaders（太空侵略者）

1978年日本太东公司发布，射击游戏鼻祖：

- **玩法**：操控炮台消灭外星人阵列
- **核心机制**：
  - 敌阵移动：整体左右移动，触边后下移并反向
  - 射击碰撞：玩家子弹与敌人碰撞判定
  - 难度递增：敌人减少时移动加快
  - 掩体系统：可被子弹逐渐破坏的防护墙

推箱子（Sokoban）

1982年日本程序员今林宏行设计的益智游戏：

- **玩法**：推动箱子到目标点，只能推不能拉
- **核心机制**：
  - 网格移动：角色按格子移动
  - 箱子推动：检测箱子前方是否为空地
  - 目标匹配：判断所有箱子是否在目标点
  - 关卡数据：二维数组存储地图布局

2048

2014年意大利开发者Gabriele Cirulli创作：

- **玩法**：滑动合并相同数字，目标合成2048
- **核心机制**：
  - 网格滑动：方块向指定方向移动
  - 合并规则：相邻相同数字合并
  - 随机生成：滑动后在空格放置新方块
  - 失败判定：网格填满且无法合并

俄罗斯方块（Tetris）

1984年苏联程序员阿列克谢·帕基特诺夫发明：

- **玩法**：控制方块下落、旋转，填满行消除
- **核心机制**：
  - 方块下落：定时器控制自动下移
  - 旋转系统：方块绕中心点旋转
  - 行消除：检测并消除填满的行
  - 速度递增：消除越多，下落越快

经典游戏的共同要素

这些经典游戏都包含相似的核心机制：

- **输入控制**：键盘、鼠标事件处理
- **状态更新**：游戏对象位置、状态变化
- **碰撞判定**：对象间的碰撞检测与响应
- **画面绘制**：图形渲染与显示更新
- **得分/胜负**：游戏目标与结束条件

实践练习

练习 9.4.1：完成乒乓球基础版

1. 实现双人控制的挡板与球弹跳
2. 球出界后判分并重置到中心
3. 为挡板和球增加配色与速度调节

实践练习

练习 9.4.2：实现扫雷核心玩法

1. 完成布雷、周围数字计算与网格显示
2. 支持左键翻开、右键标记，数字为0时递归展开
3. 增加胜负判定与简单计时/剩余雷数显示

实践练习

练习 9.4.3：自选扩展小游戏

1. 从Flappy Bird、Space Invaders、推箱子、2048或俄罗斯方块中任选一款
2. 完成核心机制：输入控制、状态更新、判定与绘制
3. 给出可运行演示，并说明主要数据结构与判定逻辑

本节回顾

- 乒乓球：输入控制、碰撞检测、得分重置
- 扫雷：网格数据结构、邻域遍历、递归展开
- 经典题材：Flappy Bird、Space Invaders、推箱子、2048、俄罗斯方块
- 共同要素：输入、状态、碰撞、渲染、胜负