

学生学号	1023000971	实验课成绩	
------	------------	-------	--

武汉理工大学 学生实验报告书

实验课程名称	数据结构与算法综合实验
开课学院	计算机与人工智能学院
指导教师姓名	刘春
学生姓名	倪锋
学生专业班级	计算机 2302

2024 -- 2025 学年 第 2 学期

实验课程名称： 数据结构与算法综合实验

实验项目名称	连连看游戏综合实践			报告成绩	
实验者	倪锋	专业班级	计算机 2302	组别	无
同组者	倪锋			完成日期	2025. 4. 26

第一部分：实验分析与设计（可加页）

一、 实验目的和具体内容

1. 实验目的

- 掌握连连看游戏的基本原理与算法实现
- 应用图论与线性结构知识实现游戏核心逻辑
- 学习界面设计与用户交互实现
- 实践迭代开发与模块化编程思想
- 理解接口设计在软件工程中的重要性

2. 实验内容

- 实现连连看游戏的基本界面设计
- 使用两种不同数据结构（矩阵结构和图结构）实现游戏逻辑
- 设计统一 API 接口，实现结构无关的应用层调用
- 实现游戏基础功能：消除判定、路径显示等
- 实现游戏扩展功能：计时、提示、重排、自动消除等

二、 分析与设计

1. 数据结构的设计

矩阵结构设计：

- 使用二维数组表示游戏元素分布
- 每个元素包含类型索引和状态属性
- 路径查找采用直线、一次转折、两次转折三种方式

```
1 matrix = [[{'index': element_type, 'status': 'normal'} for j in range(col)] for i in range(row)]
```

图结构设计:

- 使用邻接列表表示游戏元素之间的连接关系
- 节点属性存储元素类型和状态信息
- 路径查找采用广度优先搜索 (BFS) 算法



```
1 graph = {} # 键: 节点坐标 (row, col), 值: 相邻节点列表
2 nodes_data = {} # 键: 节点坐标 (row, col), 值: {'index': 水果类型索引, 'status': 节点状态}
```

统一 API 设计:

设计相同的接口函数, 使应用层代码可以无缝切换数据结构:

- `get_matrix()`: 获取元素矩阵
- `get_cell(row, col)`: 获取指定位置元素
- `eliminate_cell(row, col)`: 消除指定元素
- `is_elimitable(row1, col1, row2, col2)`: 判断两元素是否可消除
- `promote()`: 提示功能, 寻找可消除的元素对
- `rearrange_matrix()`: 重排未消除元素

2. 界面设计

页面结构:

- 主菜单页面: 显示游戏模式选择
- 基本模式页面: 带计时器的游戏界面
- 休闲模式页面: 无计时器的游戏界面
- 设置页面: 游戏参数配置

组件设计:

- 按钮组件: 实现交互功能
- 进度条组件: 实现计时功能
- 游戏矩阵显示: 展示游戏元素及状态
- 动画效果: 展示消除路径、胜利提示等

3. 类设计

Page 类:

- MainMenu: 主菜单界面
- BasicMode: 基础模式游戏界面 (带计时)
- LeisureMode: 休闲模式游戏界面 (继承自 BasicMode)
- SettingPage: 设置页面

Logic 类:

- Matrix: 矩阵结构实现
- Graph: 图结构实现

Component 类:

- Button: 按钮组件
- ProgressBar: 进度条/计时器组件

Utils 类:

- config: 配置管理
- image_processor: 图像处理工具

三、主要仪器设备及耗材

- 开发环境: Python 3.8+
- 开发工具: Visual Studio Code
- 依赖库: Pygame
- 素材资源: 水果图像、游戏背景、图标等

第二部分：实验过程和结果（可加页）

一、 实现说明

1. 迭代开发过程

本项目采用迭代开发方式，主要经历以下阶段：

第一阶段：界面设计

- 实现基本窗口创建与显示
- 设计并实现主菜单界面
- 实现按钮等基础组件

第二阶段：基于矩阵结构的功能实现

- 设计并实现矩阵数据结构
- 实现基础消除判定算法
- 完成基本游戏流程

第三阶段：基于图结构的功能实现

- 设计并实现图数据结构
- 使用 BFS 算法优化路径查找
- 确保与矩阵结构 API 兼容

第四阶段：扩展功能实现

- 添加提示功能
- 实现重排功能
- 添加计时器
- 实现自动消除功能

2. 核心逻辑实现

- 矩阵结构实现：矩阵结构采用二维数组存储元素,并通过直观的路径算法判断元素是否可消除：



```
1 def is_elimitable(self, row1, col1, row2, col2):
2     # 基本检查: 边界、消除状态、相同类型
3     if self.matrix[row1][col1]['index'] != self.matrix[row2][col2]['index']:
4         return []
5
6     # 检查0次转折 (直线连接)
7     path = self.check_direct_path(row1, col1, row2, col2)
8     if path: return path
9
10    # 检查1次转折
11    path = self.check_one_turn_path(row1, col1, row2, col2)
12    if path: return path
13
14    # 检查2次转折
15    path = self.check_two_turn_path(row1, col1, row2, col2)
16    if path: return path
17
18    return []
```

- 图结构实现：图结构使用邻接列表表示元素间的连接关系, 并使用 BFS 算法寻找最优路径：

```
1 def find_path_with_bfs(self, row1, col1, row2, col2):
2     start = (row1, col1)
3     end = (row2, col2)
4
5     # 队列中的元素: (当前节点, 路径, 已转折次数, 当前方向)
6     queue = deque([(start, [start], 0, 0)])
7     visited = set()
8
9     while queue:
10         node, path, turns, direction = queue.popleft()
11
12         # 找到目标节点
13         if node == end:
14             return self.simplify_path(path)
15
16         # 转弯次数超过2, 不继续探索
17         if turns > 2:
18             continue
19
20         # 探索四个方向
21         for dr, dc in [(0, 1), (0, -1), (1, 0), (-1, 0)]:
22             # 计算新位置、新方向、新转折次数
23             # 将符合条件的新节点加入队列
24             # ...
```

- 统一 API 设计：通过在两个类中实现相同的方法名和参数,保证应用层代码不需要关心具体实现：

```
1 # 在BasicMode类中可以无缝使用任一结构
2 self.game_map = Graph(row=self.row, col=self.col, elements=self.fruit_images)
3 # 或者
4 self.game_map = Matrix(row=self.row, col=self.col, elements=self.fruit_images)
5
6 # 应用层调用相同的API
7 path = self.game_map.is_elimidable(row1, col1, row2, col2)
8 self.game_map.eliminate_cell(row1, col1)
```

3. 界面功能实现

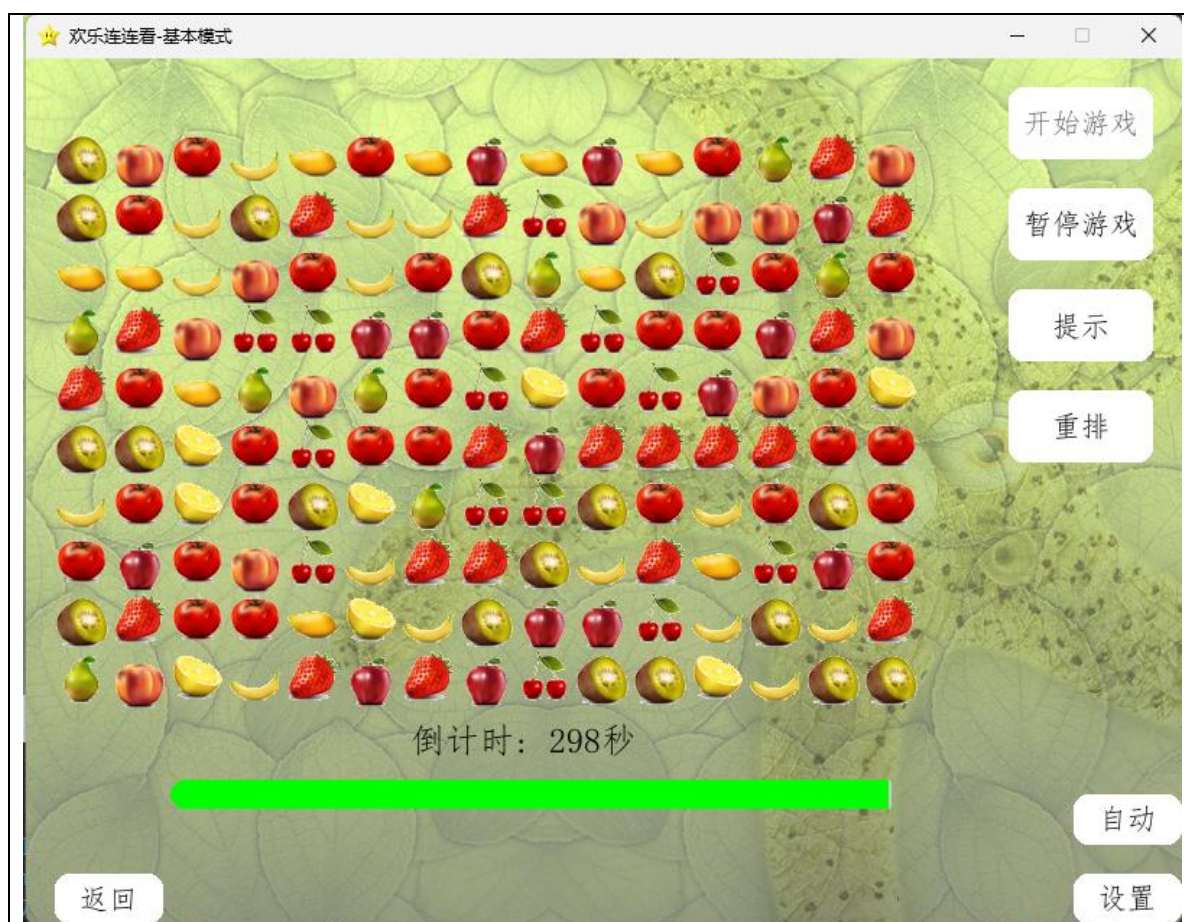
主菜单界面：实现了多种游戏模式选择、设置和帮助功能入口。

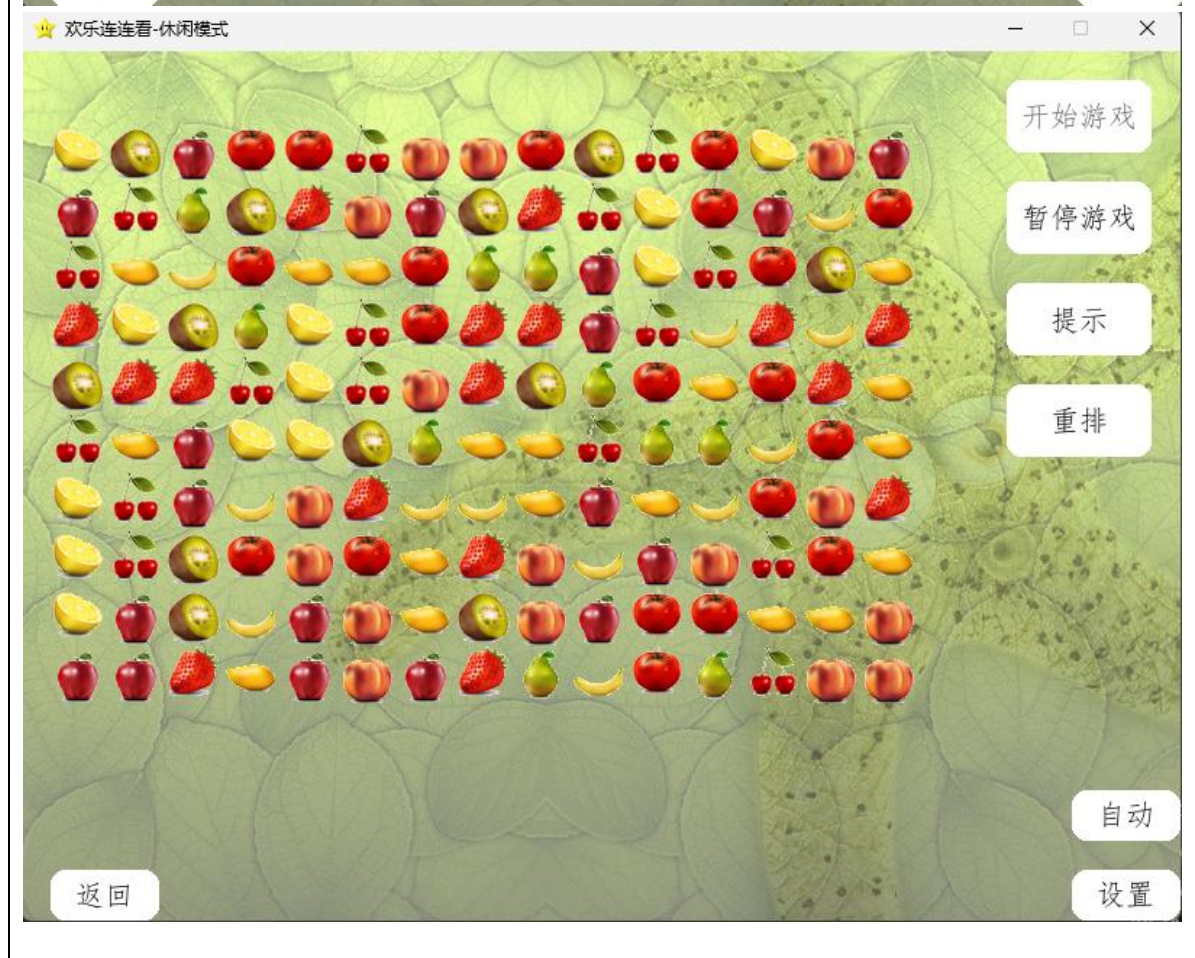
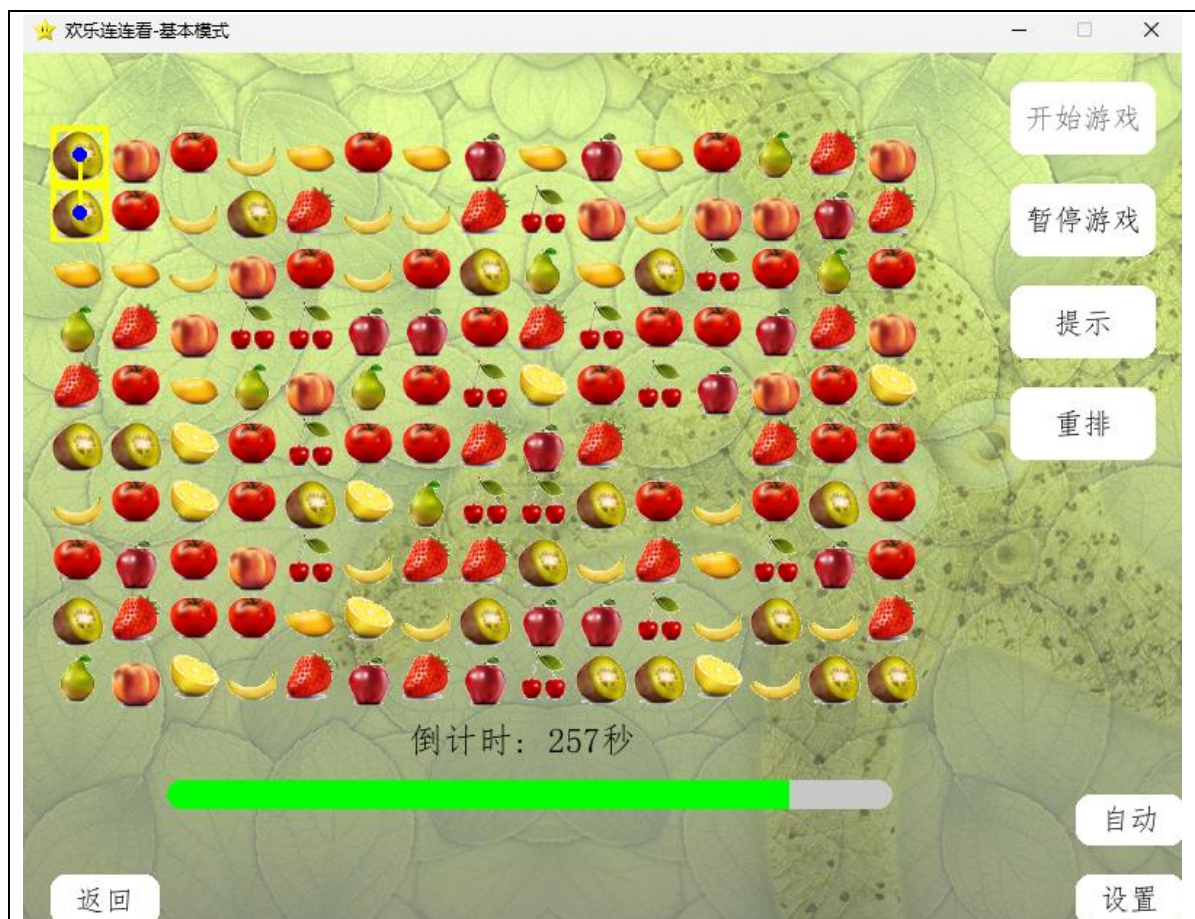


游戏界面：包含以下主要功能：

- 开始游戏：初始化游戏元素
- 暂停/继续：控制游戏进程
- 提示功能：标记可消除的元素对
- 重排功能：打乱未消除的元素
- 自动功能：自动寻找并消除元素对







设置界面：实现了调整游戏行列数的功能，并保存到配置文件。



二、 调试说明（调试手段、过程及结果分析）

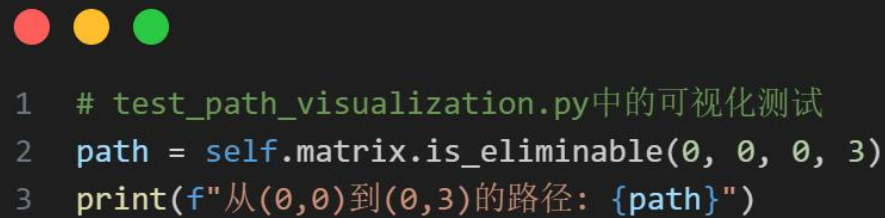
调试手段：

- 使用 `print()` 进行关键变量值输出
- 开发专门的可视化测试模块（如 `test_path_visualization.py`）
- 分离测试各功能模块（如 `test_matrix_logic.py`）

关键调试过程：

路径查找算法调试：

- 开发了可视化工具展示路径查找过程
- 设计测试场景验证直线、一次转折和两次转折情况



```
1 # test_path_visualization.py中的可视化测试
2 path = self.matrix.is_elimitable(0, 0, 0, 3)
3 print(f'从(0,0)到(0,3)的路径: {path}')
```

图结构 BFS 算法调试：

- 逐步跟踪队列和访问集合变化
- 分析不同情况下的转折点计算逻辑

自动消除功能调试：

- 通过回调函数机制确保动画结束后再执行下一步消除
- 处理特殊边界情况，如仅剩两个元素时的处理

三、 软件测试（测试用例、程序运行界面、综合分析和结论）

1. 测试用例设计

消除逻辑测试：

- 测试相邻元素直接连接消除
- 测试一次转折路径消除
- 测试两次转折路径消除
- 测试无法消除的情况

界面功能测试：

- 测试开始游戏功能
- 测试暂停/继续功能
- 测试提示功能
- 测试重排功能
- 测试自动消除功能

配置功能测试：

- 测试修改行列数并保存配置
- 测试配置文件加载

2. 运行界面

游戏运行后具有以下界面：

- 主菜单界面：展示游戏标题、模式选择按钮
- 基本模式游戏界面：显示游戏元素矩阵、功能按钮和倒计时
- 休闲模式界面：无倒计时的游戏界面
- 设置界面：调整游戏参数

3. 综合分析和结论

性能分析：

- 矩阵结构实现简单直观，但在大规模地图上路径查找效率较低
- 图结构使用 BFS 算法，路径查找效率更高，尤其在复杂路径情况下
- 统一 API 设计使得底层结构变更不影响应用层代码

用户体验分析：

- 动画效果增强了游戏可视化体验
- 提示和自动功能降低了游戏难度，适合不同玩家需求
- 多种游戏模式满足不同场景需要

结论：

- 1 成功实现了基于两种数据结构的连连看游戏
- 2 统一 API 设计验证了良好接口设计对软件工程的重要性
- 3 迭代开发方式有效提升了开发效率和质量
- 4 扩展功能丰富了游戏体验

第三部分：实验小结、收获与体会

实验小结

本次实验成功实现了一个完整的连连看游戏，包含基本和扩展功能。通过两种不同数据结构的实现，深入理解了算法设计和数据结构选择对软件性能的影响。统一 API 设计使得不同实现可以无缝切换，展现了良好软件架构的优势。

收获与体会

1 数据结构选择的重要性：

- 不同问题适合不同的数据结构
- 图结构在路径查找问题上有天然优势
- 矩阵结构在直观性和实现简便性上有优势

2 接口设计的价值：

- 统一的 API 设计使底层实现可以透明变化
- 接口先行的设计思想有助于并行开发
- 良好的接口设计提高了代码可维护性

3 迭代开发的优势：

- 逐步实现功能使问题复杂度可控
- 每个迭代都有可运行的产品
- 便于及时发现和解决问题

4 模块化设计的好处：

- 组件复用减少了重复代码
- 通过继承（如 `LeisureMode` 继承 `BasicMode`）实现功能扩展
- 职责分离提高了代码可读性和可维护性

5 算法优化的思考：

- BFS 算法在复杂条件下的应用
- 路径简化算法提高视觉效果
- 动画与逻辑分离的设计思想

总的来说，这次实验不仅实现了一个功能完整的连连看游戏，更重要的是通过实践深化了对软件工程、数据结构与算法的理解。通过两种结构实现同一游戏逻辑的过程，特别体会到了"一切皆接口"设计思想的价值