**2025年春《数学模型与数学实验》课程实验报告**

|  |  |  |  |
| --- | --- | --- | --- |
| 实验项目名称 | **实验八 生命模拟实验** | | |
| 实验地点 | 明理楼C901 | 实验日期 | 2025/6/19 |
| 实验环境 | **1. 电脑基本配置：设备名称 windows**  **处理器 13th Gen Intel(R) Core(TM) i7-1360P 2.20 GHz**  **机带 RAM 16.0 GB**  **2. 使用软件及版本：**  matlab 2023a版 | | |
| 实验目的及意义 | 培养算法设计与调试能力：设计细胞状态更新算法，实现邻居计数、边界处理以及高效的网格刷新。  提高软件工程实践意识：通过模块化、面向对象编程思想组织代码，实现易扩展、可维护的应用结构。  加深对细胞自动机（Cellular Automaton）原理的理解：通过实现康威生命游戏，掌握细胞自动机的演化规则和动力学特性。 | | |
| 实验任务与问题 | 游戏在一个类似于围棋棋盘一样的二维方格中进行。设想每个方格中都可放置一个生命细胞，生命细胞只有两种状态：“生”或“死”，用黑色方格表示该细胞为“生”，白色方格 (空格)表示该细胞为“死”。细胞根据一定规则（生存定律）演化： 1) 每个细胞的状态由该细胞及周围八个邻居上一次的状态共同决定； 2) 当前细胞为存活状态时，当周围有2个或3个存活细胞时，该细胞继续存活。 3) 当前细胞为存活状态时，当周围有超过3个存活细胞或者低于2个存活细胞时，该细胞变成死亡状态。 4) 当前细胞为死亡状态时，当周围有3个存活细胞时，该细胞激活成存活状态。 请编写程序实现该游戏，并设计GUI界面，可演示至少5种生命模式演化形态，还可指定生命细胞“生”状态个数随机生成初始状态，展示演化形态。 | | |
| 实验过程记录 | import tkinter as tk import random   class GameOfLifeGUI:  def \_\_init\_\_(self, master, rows=50, cols=50, cell\_size=10):  self.master = master  self.rows = rows  self.cols = cols  self.cell\_size = cell\_size  self.running = False   self.grid = [[0 for \_ in range(cols)] for \_ in range(rows)]  self.temp\_grid = [[0 for \_ in range(cols)] for \_ in range(rows)]   self.canvas = tk.Canvas(master, width=cols \* cell\_size, height=rows \* cell\_size, bg='white')  self.canvas.pack(side=tk.LEFT)   # GUI controls frame  control\_frame = tk.Frame(master)  control\_frame.pack(side=tk.RIGHT, fill=tk.Y, padx=5, pady=5)   # Buttons  tk.Button(control\_frame, text='Start', command=self.start).pack(fill=tk.X)  tk.Button(control\_frame, text='Stop', command=self.stop).pack(fill=tk.X)  tk.Button(control\_frame, text='Clear', command=self.clear).pack(fill=tk.X)  tk.Button(control\_frame, text='Random', command=self.randomize).pack(fill=tk.X)   # Predefined patterns  patterns = {  'Glider': [(0, 1), (1, 2), (2, 0), (2, 1), (2, 2)],  'Blinker': [(1, 0), (1, 1), (1, 2)],  'Toad': [(2, 1), (2, 2), (2, 3), (3, 0), (3, 1), (3, 2)],  'Beacon': [(1, 1), (1, 2), (2, 1), (2, 2), (3, 3), (3, 4), (4, 3), (4, 4)],  'Pulsar': [  (2, 4), (2, 5), (2, 6), (2, 10), (2, 11), (2, 12),  (4, 2), (5, 2), (6, 2), (4, 7), (5, 7), (6, 7), (4, 9), (5, 9), (6, 9), (4, 14), (5, 14), (6, 14),  (7, 4), (7, 5), (7, 6), (7, 10), (7, 11), (7, 12),  (9, 4), (9, 5), (9, 6), (9, 10), (9, 11), (9, 12),  (10, 2), (11, 2), (12, 2), (10, 7), (11, 7), (12, 7), (10, 9), (11, 9), (12, 9), (10, 14), (11, 14),  (12, 14),  (14, 4), (14, 5), (14, 6), (14, 10), (14, 11), (14, 12)  ]  }  for name, coords in patterns.items():  btn = tk.Button(control\_frame, text=name, command=lambda c=coords: self.add\_pattern(c))  btn.pack(fill=tk.X)   # Bind canvas click for manual editing  self.canvas.bind("<Button-1>", self.toggle\_cell)   self.draw\_grid()  self.master.after(100, self.update)   def draw\_grid(self):  self.canvas.delete('all')  for r in range(self.rows):  for c in range(self.cols):  x1 = c \* self.cell\_size  y1 = r \* self.cell\_size  x2 = x1 + self.cell\_size  y2 = y1 + self.cell\_size  if self.grid[r][c] == 1:  self.canvas.create\_rectangle(x1, y1, x2, y2, fill='black', outline='gray')  else:  self.canvas.create\_rectangle(x1, y1, x2, y2, fill='white', outline='gray')   def update(self):  if self.running:  for r in range(self.rows):  for c in range(self.cols):  alive\_neighbors = sum(  self.grid[r + dr][c + dc]  for dr in (-1, 0, 1) for dc in (-1, 0, 1)  if not (dr == 0 and dc == 0)  and 0 <= r + dr < self.rows and 0 <= c + dc < self.cols  )  if self.grid[r][c] == 1:  self.temp\_grid[r][c] = 1 if alive\_neighbors in (2, 3) else 0  else:  self.temp\_grid[r][c] = 1 if alive\_neighbors == 3 else 0   # Swap grids  self.grid, self.temp\_grid = self.temp\_grid, self.grid  self.draw\_grid()  self.master.after(100, self.update)   def start(self):  self.running = True   def stop(self):  self.running = False   def clear(self):  self.running = False  self.grid = [[0 for \_ in range(self.cols)] for \_ in range(self.rows)]  self.draw\_grid()   def randomize(self):  for r in range(self.rows):  for c in range(self.cols):  self.grid[r][c] = random.choice([0, 1])  self.draw\_grid()   def add\_pattern(self, coords):  self.clear()  mid\_r, mid\_c = self.rows // 2, self.cols // 2  for dr, dc in coords:  r = mid\_r + dr  c = mid\_c + dc  if 0 <= r < self.rows and 0 <= c < self.cols:  self.grid[r][c] = 1  self.draw\_grid()   def toggle\_cell(self, event):  c = event.x // self.cell\_size  r = event.y // self.cell\_size  if 0 <= r < self.rows and 0 <= c < self.cols:  self.grid[r][c] = 1 - self.grid[r][c]  self.draw\_grid()   if \_\_name\_\_ == '\_\_main\_\_':  root = tk.Tk()  root.title("Conway's Game of Life")  game = GameOfLifeGUI(root, rows=50, cols=50, cell\_size=10)  root.mainloop() | | |
| 实验结果及分析 |  | | |
| 实验体会与收获 | 对**细胞自动机的深入理解：通过对康威生命游戏规则的编码实现，深入体会到简单局部规则演化出复杂全局行为的魅力，加深了对复杂系统中 emergent behavior 的认知。**  **编程与调试能力提升：在实现邻居统计、边界条件处理及画布刷新优化的过程中，学会了如何定位问题、逐步排查错误，并提升了对算法正确性和性能的把控能力。** | | |