《数据结构》课程实践报告

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 院、系 | 计算机学院 | | 年级专业 | 大二 软工 | 姓名 | 丁彦君 | 学号 | 2329403126 |
| 实验布置日期 | | 2024/10/13 | | 提交  日期 | 2024/10/28 | | 成绩 |  |

课程实践实验4：小猫钓鱼

## 一、问题描述及要求

**题目描述**：

**此程序实现了一种牌局游戏。两个玩家各自拥有一组牌，按顺序出牌。若当前牌在桌面已有相同牌，玩家将此牌和之前的牌收回并加入自己的牌堆。首先将对方牌出完的玩家获胜。**

**程序需求：**

1. **输入每人牌的数量及具体的牌数。**
2. **按规则模拟两人出牌的过程并显示每一步的出牌情况。**
3. **当一方获胜时输出胜利信息。**

## 二、概要设计

**（1）对实验内容的理解。**

**通过数据结构栈与循环队列实现一种模拟“接龙”出牌游戏。栈用于记录已出牌堆，队列记录两名玩家的出牌顺序。**

**（2）功能列表**

**初始化：输入每人手牌数与具体的牌面，创建并初始化两个玩家的牌堆。**

** 出牌模拟：按规则模拟每一轮出牌，比较双方当前出的牌，执行“收回”或“继续”操作。**

** 胜负判定：在任一玩家牌堆为空时，输出胜利信息并终止游戏。**

**（3）程序运行的界面设计**

**程序首先提示用户输入牌数，显示出牌过程。示例：**

* **输入: 请输入每个人的牌数：、请输入A的牌：、请输入B的牌：**
* **输出: A出牌: x, B出牌: y、A赢了！或B赢了！**

**（4）总体设计思路**

 **数据结构：利用循环队列和栈。stack1记录已出的牌，用于检测循环牌。cirqueue1用于实现两名玩家的出牌顺序。**

** 类设计：**

* **stack1：存储牌堆状态，提供push和pop方法，用于模拟出牌。**
* **cirqueue1：实现玩家牌堆循环管理，en方法实现出牌入队。**

 **方法**：

* **func1初始化队列；**
* **run进行出牌循环；**
* **stop结束游戏。**

**（5）程序结构设计**

**在这个游戏程序中，使用了面向对象的设计方式，通过 stack1 和 cirqueue1 两个类来管理玩家牌堆和已出牌堆。以下是各个模块的详细设计，包括主文件和头文件的关系、各个类的属性和方法，以及各个文件的功能分工：**

**1. 项目结构**

**主文件 (main.cpp)：包含主程序入口，负责游戏初始化和主要逻辑控制。**

**头文件 (stack1.h 和 cirqueue1.h)：定义栈和循环队列的类及其方法。**

**类实现文件 (stack1.cpp 和 cirqueue1.cpp)：包含 stack1 和 cirqueue1 类的具体实现。**

**2. 类设计**

**类1：stack1**

**stack1 类用于实现已出牌的堆栈结构。每当玩家出牌时，都会被压入该栈，便于在后续出牌中检测重复。**

**属性：**

**int \*stackArray：动态数组，用于存储牌面。**

**int top：指示栈顶元素的索引。**

**int capacity：栈的最大容量。**

**方法：**

**stack1(int size)：构造函数，初始化栈容量和栈顶指针。**

**void push(int value)：将牌压入栈顶。**

**int pop()：从栈顶弹出元素并返回其值。**

**bool isEmpty()：检测栈是否为空。**

**int get1()：返回栈顶的索引值。**

**int\* get2()：返回栈数组指针，以便进行栈内匹配操作。**

**类2：cirqueue1**

**cirqueue1 类实现循环队列，用于管理每位玩家的牌堆。循环队列可以实现顺序出牌，且出牌后牌可以重新加入末尾。**

**属性：**

**int \*queueArray：动态数组，用于存储玩家的牌。**

**int front 和 int rear：指示队列头部和尾部位置。**

**int size 和 int capacity：队列当前大小和总容量。**

**方法：**

**cirqueue1(int size)：构造函数，初始化队列大小。**

**void en(int value)：将牌加入队列尾部，模拟玩家重新获得牌。**

**int de()：从队列头部出牌，并返回牌值。**

**bool isEmpty()：判断队列是否为空，用于判断游戏结束。**

**bool isFull()：判断队列是否已满，防止溢出。**

**3. 文件结构及各部分关系**

**main.cpp：包含游戏逻辑的主文件，负责以下操作：**

**初始化玩家的牌堆（通过 cirqueue1 实现）和已出牌堆（通过 stack1 实现）。**

**循环控制出牌过程：**

**调用 cirqueue1::de() 方法获取当前玩家的出牌。**

**使用 stack1 的 push() 和 pop() 方法检查出牌重复情况并做相应处理。**

**判定胜负条件，即当一方的队列为空时输出胜利信息。**

**输出每轮的出牌情况和游戏结果。**

**stack1.h 和 stack1.cpp：定义并实现 stack1 类的所有属性和方法，提供栈结构的基本操作，用于存储已出牌堆。**

**cirqueue1.h 和 cirqueue1.cpp：定义并实现 cirqueue1 类的所有属性和方法，提供循环队列操作，用于模拟玩家出牌顺序和牌堆变化。**

**4. 各部分关系描述**

**游戏初始化：main.cpp 中读取玩家的初始牌堆，并通过 cirqueue1 的 en() 方法将每张牌加入相应的队列，形成每个玩家的初始牌堆。**

**出牌过程：每轮从 X1 和 X2 两个 cirqueue1 队列中调用 de() 方法出牌，并调用 stack1 的 push() 方法将出牌记录在栈中。通过 stack1 的 get1() 和 get2() 方法检查栈内是否已存在相同牌。如果存在相同牌，则执行回收操作，将牌从栈中弹出并返回相应队列。**

**胜负判定：在每轮出牌后，通过 isEmpty() 方法检测任一队列是否为空。如果为空则游戏结束，输出胜负结果。**

## 三、详细设计

**主函数设计**

**初始化：输入两名玩家牌堆。**

**循环出牌：每次循环一方出牌，判断当前牌是否在栈中：**

**若存在相同牌，则取回与匹配牌。**

**若不存在，则继续出牌。**

**判定胜负：当任意玩家牌堆为空时，输出胜利信息。**

**关键算法**

**出牌：使用循环队列实现轮流出牌，栈实现已出牌存储。**

**匹配：判断出牌与栈内牌的匹配，取出同牌进行回收，未匹配则存入栈。**

**循环检测：玩家空牌堆检测，检测后输出胜负信息。**

## 四、实验结果



## 五、实验分析与探讨

**测试结果的分析**

**1.性能：基于栈和队列的设计，出牌与牌组状态切换的时间复杂度为O(n)，空间复杂度较低，适合中小规模数据。**

**2.局限性：当前实现并未考虑异常输入及无效字符处理，可在cin语句中增加输入检测。**

**实现过程中遇到的问题及解决**

**1.问题：在牌组匹配逻辑中，回收过程未考虑栈清空情况。**

**2.解决：调整判定顺序，避免栈指针超出界限。**

**时间复杂度**

**初始化阶段：**

**stack1 和 cirqueue1 的初始化复杂度是 O(n)，其中 n 是初始牌的数量。**

**各个玩家牌堆的初始化通过 en() 方法完成，该操作复杂度为 O(1)。**

**整体初始化的时间复杂度为 O(n)。**

**游戏出牌过程：**

**每轮出牌包含两步：**

**从每个玩家的队列中调用 de() 方法出牌，复杂度为 O(1)。**

**将出牌放入栈 stack1 中，如果栈内存在相同牌，执行弹出操作并将牌返回给对应玩家。**

**重复牌检查：在栈中查找是否存在重复牌的过程复杂度为 O(k)，其中k 是当前栈的大小。在最坏情况下，每张牌都可能已在栈中存在，因此此部分的复杂度在极端情况下为 O(n\*n)。**

**出牌与回收操作：当存在重复牌时，栈将通过 pop() 操作逐步回收这些牌并将它们放回相应的队列，最坏情况下，这一过程的时间复杂度也为 O(n)。**

**胜负判定：**

**在每轮出牌后，使用 isEmpty() 方法判断队列是否为空，复杂度为 O(1)。**

**综上，整个程序的最坏情况下的时间复杂度为O(n\*n)，主要由于栈内的重复牌检查操作引入了平方级别的复杂度。**

**空间复杂度**

**栈（stack1）空间复杂度：**

**栈的容量上限为 2n，因此其空间复杂度为 O(n)。**

**循环队列（cirqueue1）空间复杂度：**

**每个玩家的队列大小为 n，共两个玩家的牌堆，因此循环队列的空间复杂度为 O(n)。**

**总体空间复杂度：**

**程序整体空间复杂度为 O(n)，因为只使用了两个大小为 n 的队列和一个大小为 n 的栈。**

## 六、小结

**在该程序的设计和实现中，我们利用栈和循环队列来实现一个模拟出牌和胜负判断的游戏逻辑，展现了在简单游戏规则中使用经典数据结构的重要性。此外，在设计过程中遇到的一个挑战是如何保证牌堆的有序回收，通过栈和队列的组合结构，我们成功实现了该功能，使得程序逻辑更加清晰简洁。此实验通过对数据结构的灵活应用，不仅强化了对栈和队列的理解，还体现了在实际问题中结合不同数据结构的设计思维。同时，程序具有较好的扩展性，未来可以进一步添加更多的游戏规则，如玩家出牌的额外策略和牌堆的特殊效果，以更好地模拟现实中的游戏场景。**

## 附录：源代码

**代码见附件**