# 纸牌程序设计文档

## 1. 文档概述

### 1.1 文档目的

本文档详细说明手牌区翻牌替换、桌面牌与手牌顶牌匹配、操作回退三大核心需求的实现逻辑，并阐述在现有代码结构下新增卡牌、新增回退类型的扩展方案。

### 1.2 核心需求摘要

| 需求编号 | 需求描述 |
| --- | --- |
| 需求 1 | 点击手牌区指定卡牌（如♥A），该卡牌平移替换手牌区顶部牌（如♣4） |
| 需求 2 | 点击桌面牌（如♦3），若其与手牌顶牌点数差 1 则匹配成功，该桌面牌平移替换手牌顶牌 |
| 需求 3 | 支持连续回退操作，按操作逆序将卡牌移回原位置，恢复操作前状态，直至无回退记录 |

## 2. 核心模块设计

### 2.1 类结构定义

现有代码文件夹`Classes`的核心类设计如下（关键属性 / 方法仅列核心）：

#### 2.1.1 卡牌基类（Card）

```
public class Card
{
    // 核心属性
    public string Suit { get; set; } // 花色（♥/♣/♦/♠）
    public int Point { get; set; } // 点数（1-A,2-2,...,13-K）
    public Vector3 Position { get; set; } // 世界坐标
    public Transform Transform { get; private set; } // 变换组件
    public bool IsInHand { get; set; } // 是否在手牌区
    public bool IsTopCardOfHand { get; set; } // 是否为手牌顶牌

    // 核心方法
    public Card(string suit, int point, Vector3 initPos)
    {
        Suit = suit;
        Point = point;
        Position = initPos;
        Transform = GetComponent<Transform>();
    }

    // 执行平移动画
    public IEnumerator MoveTo(Vector3 targetPos, float duration = 0.3f)
    {
```

```
        float elapsed = 0;
        Vector3 startPos = Transform.position;
        while (elapsed < duration)
        {
            Transform.position = Vector3.Lerp(startPos, targetPos, elapsed /
duration);
            elapsed += Time.deltaTime;
            yield return null;
        }
        Transform.position = targetPos;
        Position = targetPos; // 更新记录的位置
    }
}
```

### 2.1.2 手牌区管理类（HandCardArea）

```
public class HandCardArea : MonoBehaviour
{
    // 手牌列表（顶牌为列表最后一位）
    public List<Card> HandCards { get; private set; } = new List<Card>();
    // 当前顶牌
    public Card TopCard => HandCards.Count > 0 ? HandCards[^1] : null;

    // 替换顶牌（核心方法）
    public void ReplaceTopCard(Card newCard)
    {
        if (TopCard == null) return;
        // 记录原顶牌位置（用于回退）
        Vector3 oldTopPos = TopCard.Position;
        // 注册新卡牌的平移动画
        StartCoroutine(newCard.MoveTo(oldTopPos));
        // 替换逻辑
        HandCards.Remove(TopCard);
        newCard.IsTopCardOfHand = true;
        newCard.IsInHand = true;
        HandCards.Add(newCard);
        // 通知回退管理器记录操作
        RollbackManager.Instance.RecordOperation(
            OperationType.HandCardReplace, // 操作类型：手牌替换
            newCard, // 操作卡牌
            oldTopPos, // 目标位置（原顶牌位置）
            newCard.Position, // 原位置（点击前的位置）
            TopCard // 被替换的顶牌
        );
    }
}
```

### 2.1.3 桌面牌区管理类（DeskCardArea）

```
public class DeskCardArea : MonoBehaviour
{
    public List<Card> DeskCards { get; private set; } = new List<Card>();
    public HandCardArea HandArea; // 关联手牌区
```

```
    // 桌面牌点击校验与匹配
    public void OnDeskCardClick(Card clickedCard)
    {
        if (HandArea.TopCard == null) return;
        // 校验点数差（核心匹配规则）
        int pointDiff = Math.Abs(clickedCard.Point - HandArea.TopCard.Point);
        if (pointDiff != 1) return;

        // 匹配成功：替换手牌顶牌
        Vector3 oldTopPos = HandArea.TopCard.Position;
        StartCoroutine(clickedCard.MoveTo(oldTopPos));
        // 更新状态
        DeskCards.Remove(clickedCard);
        HandArea.HandCards.Remove(HandArea.TopCard);
        clickedCard.IsInHand = true;
        clickedCard.IsTopCardOfHand = true;
        HandArea.HandCards.Add(clickedCard);
        // 记录回退操作
        RollbackManager.Instance.RecordOperation(
            OperationType.DeskCardMatch, // 操作类型：桌面牌匹配
            clickedCard, // 操作卡牌
            oldTopPos, // 目标位置
            clickedCard.Position, // 原位置
            HandArea.TopCard // 被替换的顶牌
        );
    }
}
```

### 2.1.4 回退管理器类（RollbackManager）

```
// 操作类型枚举
public enum OperationType
{
    HandCardReplace, // 手牌替换
    DeskCardMatch    // 桌面牌匹配
}

// 回退操作记录模型
public class RollbackRecord
{
    public OperationType OpType { get; set; } // 操作类型
    public Card OpCard { get; set; } // 操作的卡牌
    public Vector3 TargetPos { get; set; } // 操作后的位置
    public Vector3 OriginalPos { get; set; } // 操作前的位置
    public Card ReplacedCard { get; set; } // 被替换的顶牌
    public Vector3 ReplacedCardPos { get; set; } // 被替换顶牌的原位置
}

public class RollbackManager : Singleton<RollbackManager>
{
    // 回退栈（先进后出，保证逆序回退）
    private Stack<RollbackRecord> _rollbackStack = new Stack<RollbackRecord>();

    // 记录操作（供手牌/桌面区调用）
```

```
    public void RecordOperation(OperationType opType, Card opCard, Vector3
targetPos, Vector3 originalPos, Card replacedCard)
    {
        var record = new RollbackRecord
        {
            OpType = opType,
            OpCard = opCard,
            TargetPos = targetPos,
            OriginalPos = originalPos,
            ReplacedCard = replacedCard,
            ReplacedCardPos = replacedCard?.Position ?? Vector3.zero
        };
        _rollbackStack.Push(record);
    }

    // 执行单次回退
    public void DoRollback()
    {
        if (_rollbackStack.Count == 0)
        {
            Debug.Log("无回退记录");
            return;
        }
        var record = _rollbackStack.Pop();
        // 反向平移：操作卡牌回到原位置
        StartCoroutine(record.OpCard.MoveTo(record.OriginalPos));
        // 恢复被替换的顶牌
        if (record.ReplacedCard != null)
        {
            record.ReplacedCard.IsTopCardOfHand = true;
            HandCardArea.Instance.HandCards.Remove(record.OpCard);
            HandCardArea.Instance.HandCards.Add(record.ReplacedCard);
            // 被替换顶牌回到原位置（若需）
            StartCoroutine(record.ReplacedCard.MoveTo(record.ReplacedCardPos));
        }
        // 恢复卡牌归属（桌面/手牌）
        if (record.OpType == OperationType.DeskCardMatch)
        {
            record.OpCard.IsInHand = false;
            DeskCardArea.Instance.DeskCards.Add(record.OpCard);
        }
    }
}
```

# 3. 核心需求实现细节

## 3.1 需求 1：手牌区翻牌替换

### 3.1.1 触发流程

1. 给手牌区所有卡牌绑定`OnClick`事件，点击事件回调指向`HandCardArea.OnHandCardClick`方法；
2. 点击♥A 时，`OnHandCardClick`方法先校验：该卡牌是否属于手牌区且非顶牌；
3. 校验通过后，调用`HandCardArea.ReplaceTopCard(♥A)`方法；

4. ReplaceTopCard方法执行：
    - 记录♣4（原顶牌）的位置；
    - 启动♥A 的`MoveTo`协程，平移至♣4 位置；
    - 从手牌列表移除♣4，将♥A 设为新顶牌并加入列表；
    - 调用`RollbackManager.RecordOperation`记录该操作（类型为`HandCardReplace`）。

### 3.1.2 关键约束

- 仅手牌区非顶牌的卡牌可触发该操作；
- 平移动画为线性插值（`Vector3.Lerp`），保证移动平滑；
- 操作记录需完整保存♥A 的原位置、♣4 的位置及♣4 实例，为回退做准备。

## 3.2 需求 2：桌面牌和手牌区顶部牌匹配

### 3.2.1 触发流程

1. 给桌面区所有卡牌绑定`OnClick`事件，点击事件回调指向`DeskCardArea.OnDeskCardClick`方法；
2. 点击♦3 时，先校验：♦3 与手牌顶牌♣4 的点数差是否为 1（`Math.Abs(3-4)=1`，符合规则）；
3. 校验通过后，执行：
    - 启动♦3 的`MoveTo`协程，平移至♣4 位置；
    - 从桌面列表移除♦3，从手牌列表移除♣4，将♦3 加入手牌列表并设为顶牌；
    - 调用`RollbackManager.RecordOperation`记录该操作（类型为`DeskCardMatch`）。

### 3.2.2 关键约束

- 点数匹配规则与花色无关，仅校验点数绝对值差为 1；
- 若手牌区无顶牌（列表为空），则直接拒绝匹配操作；
- 匹配成功后，桌面牌的归属从 “桌面区” 切换为 “手牌区”。

## 3.3 需求 3：回退功能

### 3.3.1 核心逻辑

回退功能依赖`RollbackManager`的栈结构，核心流程：

1. 每次执行 “手牌替换” 或 “桌面匹配” 操作时，都会生成`RollbackRecord`并压入栈；
2. 点击 “回退按钮” 时，调用RollbackManager.DoRollback()
    - 若栈为空，提示 “无回退记录”；
    - 若栈非空，弹出最新的`RollbackRecord`；
    - 根据记录的操作类型，执行反向逻辑：
        - 操作卡牌（如♥A/♦3）通过`MoveTo`移回原位置；
        - 恢复被替换的顶牌（如♣4）的位置和 “顶牌” 状态；
        - 若为桌面牌匹配操作，还需将操作卡牌（如♦3）归回桌面区列表；
3. 连续点击回退按钮时，重复上述步骤，直至栈空。

### 3.3.2 示例场景（点击♦3→点击♥A→点击♠2）

| 回退次数 | 操作逻辑 | 状态恢复 |
| --- | --- | --- |
| 第 1 次 | 弹出 “♠2 替换♥A” 的记录→♠2 移回原位置→♥A 恢复为手牌顶牌 | 手牌顶牌回到♥A，♠2 回到点击前位置 |
| 第 2 次 | 弹出 “♥A 替换♦3” 的记录→♥A 移回原位置→♦3 恢复为手牌顶牌 | 手牌顶牌回到♦3，♥A 回到点击前位置 |
| 第 3 次 | 弹出 “♦3 替换♣4” 的记录→♦3 移回桌面→♣4 恢复为手牌顶牌 | 手牌顶牌回到♣4，♦3 回到桌面原位置 |
| 第 4 次 | 栈空→提示无回退记录 | 无状态变化 |

# 4. 扩展设计

## 4.1 新增一张卡牌（如♠5）

在现有代码结构下，新增卡牌仅需遵循以下步骤，无需修改核心类的逻辑：

1. 实例化新卡牌

```
// 在卡牌初始化脚本中添加
Card spade5 = new Card("♠", 5, new Vector3(100, 200, 0)); // 自定义初始位置
```

2. 归属绑定：
   - 若为手牌区卡牌：加入`HandCardArea.HandCards`列表，绑定`OnClick`事件（复用`HandCardArea.OnHandCardClick`）；
   - 若为桌面区卡牌：加入`DeskCardArea.DeskCards`列表，绑定`OnClick`事件（复用`DeskCardArea.OnDeskCardClick`）；
3. 规则适配：

   点数匹配规则（点数差 1）无需修改，♠5 会自动参与桌面牌与手牌顶牌的匹配校验；
4. 回退兼容：

   回退管理器的RollbackRecord基于Card基类记录信息，新卡牌的操作会自动被记录和回退，无需修改RollbackManager。

## 4.2 新增一种新类型的回退功能（如 “手牌区卡牌交换”）

若新增 “手牌区两张卡牌交换位置” 的功能，并需支持该操作的回退，扩展步骤如下：

1. 扩展操作类型枚举：

```
public enum OperationType
{
    HandCardReplace,
    DeskCardMatch,
    HandCardSwap // 新增：手牌交换
}
```

2. 扩展回退记录模型

```
public class RollbackRecord
{
    // 原有字段...
    // 新增：交换相关字段
    public Card SwapCard2 { get; set; } // 参与交换的第二张卡牌
    public Vector3 SwapCard2OriginalPos { get; set; } // 第二张卡牌原位置
}
```

3. 实现交换操作逻辑：

在HandCardArea中新增交换方法，并记录回退操作：

```
public void SwapHandCards(Card card1, Card card2)
{
    // 记录交换前位置
    Vector3 card1OldPos = card1.Position;
    Vector3 card2OldPos = card2.Position;
    // 执行交换动画
    StartCoroutine(card1.MoveTo(card2OldPos));
    StartCoroutine(card2.MoveTo(card1OldPos));
    // 记录回退操作
    RollbackManager.Instance.RecordOperation(
        OperationType.HandCardSwap,
        card1,
        card2OldPos,
        card1OldPos,
        null, // 无被替换顶牌，设为null
        card2, // 交换的第二张牌
        card2OldPos // 第二张牌原位置
    );
}
```

4. 扩展回退执行逻辑：

在RollbackManager.DoRollback()中新增分支：

```
public void DoRollback()
{
    if (_rollbackStack.Count == 0) { /* 提示逻辑 */ return; }
    var record = _rollbackStack.Pop();
    switch (record.OpType)
    {
        case OperationType.HandCardReplace: /* 原有逻辑 */ break;
        case OperationType.DeskCardMatch: /* 原有逻辑 */ break;
        // 新增：手牌交换回退
        case OperationType.HandCardSwap:
            // 两张卡牌反向平移，回到交换前位置
            StartCoroutine(record.OpCard.MoveTo(record.OriginalPos));

 StartCoroutine(record.SwapCard2.MoveTo(record.SwapCard2OriginalPos));
            break;
    }
}
```