Card.h:

1. 列举了卡牌颜色
2. 列举了卡牌类型

```
enum class CardColor {
    RED,
    YELLOW,
    GREEN,
    BLUE,
    WILD  // 万能色
};

// 卡牌类型枚举
enum class CardType {
    NUMBER,         // 数字卡
    SKIP,           // 跳过
    REVERSE,        // 反转
    DRAW_TWO,       // 抽两张
    WILD,           // 万能牌
    WILD_DRAW_FOUR, // 万能抽四张
    PACKAGE,        // 包装卡（自定义）
    FLASH           // 闪光卡（自定义）
};
```

3.对于每一张卡

```
CardData {
    int id;                 // 卡牌唯一 ID
    CardColor color;        // 卡牌颜色
    CardType type;          // 卡牌类型
    int number;             // 数字（仅数字卡有效，-1 表示非数字卡）
    std::string name;       // 卡牌显示名称
```

4.用于判断两张卡牌是否匹配

```
bool matches(const CardData& other) const {}
        // 万能牌可以匹配任何颜色
        // 颜色匹配
        // 类型匹配（对于功能卡）
        // 数字匹配（对于数字卡）
```

Card.cpp:

1. 实现 canPlayOn 方法

利用上面 bool matches(const CardData& other) const {}检查这张牌能不能出

FunctionCard.h 和 FunctionCard.cpp:

实现所有功能卡牌的具体效果：

SkipCard：跳过下个玩家

ReverseCard：反转游戏方向

DrawTwoCard：下个玩家抽 2 张牌并跳过
WildCard：改变当前颜色
WildDrawFourCard：改变颜色+下个玩家抽 4 张牌（有使用条件）
PackageCard：丢弃所有指定颜色的数字卡
FlashCard：只能出指定的颜色否则加牌

需要 GameState.h 实现的功能清单

为了让卡牌系统正常工作，GameState.h 需要提供以下完整接口：

1. 玩家管理接口

cpp

```cpp
class GameState {public:
    // 获取下一个玩家 ID（考虑反转和跳过效果）
    int getNextPlayerId() const;
    int getNextPlayerId(int currentPlayer) const;

    // 玩家数量
    int getPlayerCount() const;

    // 玩家手牌访问
    std::vector<CardData> getPlayerHand(int playerId) const;

    // 玩家状态
    bool isPlayerActive(int playerId) const;
    int getPlayerScore(int playerId) const;};
```

2. 卡牌操作接口

cpp

```cpp
class GameState {public:
    // 抽牌操作
    void makePlayerDrawCards(int playerId, int count);
```

```
    // 弃牌操作
    void discardPlayerCard(int playerId, int cardId);

    // 手牌管理
    void addCardToPlayerHand(int playerId, const CardData& card);
    void removeCardFromPlayerHand(int playerId, int cardId);};
```

3. 游戏状态管理接口

```
cpp
class GameState {public:
    // 颜色管理
    CardColor getCurrentColor() const;
    void setCurrentColor(CardColor color);

    // 游戏方向
    PlayDirection getPlayDirection() const;
    void reversePlayDirection();

    // 回合控制
    void skipPlayerTurn(int playerId);
    void setCurrentPlayer(int playerId);
    int getCurrentPlayerId() const;

    // 牌堆状态
    int getDrawPileSize() const;
    int getDiscardPileSize() const;
    const std::vector<CardData>& getDiscardPile() const;};
```

4. 特殊效果状态接口（用于 Flash 卡等）

```
cpp
class GameState {public:
    // Flash 效果管理
    void setFlashEffect(CardColor flashColor, int playerId);
    void clearFlashEffect();
    bool isFlashEffectActive() const;
```

```
    CardColor getFlashColor() const;
    int getFlashInitiator() const;

    // 其他效果状态
    void setSkipCount(int count);
    int getSkipCount() const;
    void decrementSkipCount();};
```

5. 事件通知接口

```
cpp
class GameState {public:
    // 卡牌效果事件通知
    void notifyCardEffect(const std::string& effectType, const nlohmann::json& dat
a);

    // 游戏状态变更通知
    void notifyGameStateChange(const std::string& changeType, const nlohmann::json&
data);

    // 玩家状态变更通知
    void notifyPlayerStateChange(int playerId, const std::string& changeType, const
nlohmann::json& data);};
```

6. 数据访问接口

```
cpp
class GameState {public:
    // 序列化支持
    nlohmann::json serialize() const;
    bool deserialize(const nlohmann::json& data);

    // 验证状态
    bool validateState() const;

    // 游戏统计
    int getTurnNumber() const;
```

```
    GameStatus getGameStatus() const;};
```

具体方法实现要求

对于 Flash 卡的特殊需求：

cpp

```
// Flash 卡需要这些具体实现：class GameState {private:
    struct FlashEffectState {
        bool active;
        CardColor targetColor;
        int initiatorPlayerId;
        std::vector<int> affectedPlayers;
    };

    FlashEffectState flashState;
    public:
    void setFlashEffect(CardColor flashColor, int playerId) {
        flashState.active = true;
        flashState.targetColor = flashColor;
        flashState.initiatorPlayerId = playerId;
        flashState.affectedPlayers.clear();
    }

    void clearFlashEffect() {
        flashState.active = false;
    }

    bool isFlashEffectActive() const { return flashState.active; }
    CardColor getFlashColor() const { return flashState.targetColor; }};
```

对于 Package 卡的特殊需求：

cpp

```
class GameState {public:
    void discardPlayerCard(int playerId, int cardId) {
        // 需要实现从玩家手牌移除指定卡牌
```

```
        // 并将卡牌添加到弃牌堆
        auto& hand = playerHands[playerId];
        hand.erase(std::remove_if(hand.begin(), hand.end(),
            [cardId](const CardData& card) { return card.id == cardId; }), hand.end
());

        discardPile.push_back(getCardDataById(cardId));
    }};
```

对于 Wild Draw Four 卡的验证需求：

```
cpp
class GameState {public:
    bool canPlayerPlayWildDrawFour(int playerId) const {
        CardColor currentColor = getCurrentColor();
        const auto& hand = getPlayerHand(playerId);

        // 检查玩家是否有匹配当前颜色的牌
        for (const auto& card : hand) {
            if (card.color == currentColor && card.type != CardType::WILD && card.typ
e != CardType::WILD_DRAW_FOUR) {
                return false; // 有匹配的牌，不能出Wild Draw Four
            }
        }
        return true;
    }};
```