Mastering the Sevens Card Game: Strategy Development Guide

by Janan Arslan

Game Rules & Mechanics

Standard 52-card deck

The game uses a complete deck with all four suits.

Starting position

Game begins with all four sevens already on the table.

Adjacent play only

Cards must be played adjacent to existing cards (one rank higher or higher or lower in the same suit).

Win condition

First player to empty their hand wins the game.

Understanding these fundamental rules is crucial for developing your strategy. The game creates an expanding pattern of playable positions, starting from the positions, starting from the sevens and working outward in both directions within each suit.

Implementation Framework

Understand the Framework

Examine the provided code structure and class hierarchy to understand how the game system works.

Study Existing Strategies

Review RandomStrategy and GreedyStrategy to understand their approaches and limitations.

Design Your Strategy

Decide on a strategic approach (aggressive, defensive, adaptive) based on your analysis.

Implement & Test

Code your strategy, compile it as a shared library, and test against sample strategies. strategies.

The framework provides a structured environment for your strategy to interact with the game. Your
Your implementation will need to respond to game events and make decisions based on the current state of current state of play.

Key Data Structures

tableLayout

A 2D boolean array representing cards on the table:

- tableLayout[suit][rank] = true/false
- suit is 0-3 (Clubs=0, Diamonds=1, Hearts=2, Spades=3)
- rank is 1-13 (Ace=1, 2=2, ..., King=13)
- true if card is on table, false otherwise

Card Playability

A card is playable if it's adjacent to a card already on the table: table:

- If 7♥ is on table, you can play 6♥ or 8♥
- If 6♥, 7♥, 8♥ are on table, you can play 5♥ or 9♥
- Cards must be in the same suit as adjacent cards

Understanding these data structures is essential for implementing your strategy. The tableLayout provides complete information about the current game state, which you'll use to determine valid moves and make strategic decisions.

Required Strategy Methods

initialize(playerID)

Set up your strategy with your player identifier. This is called once called once at the beginning of each game and is a good place to place to initialize any tracking variables.

selectCardToPlay(hand, tableLayout)

The core of your strategy - decide which card to play from your hand based on the current table state. Return -1 if you must pass (no playable cards).

observeMove(playerID, playedCard)

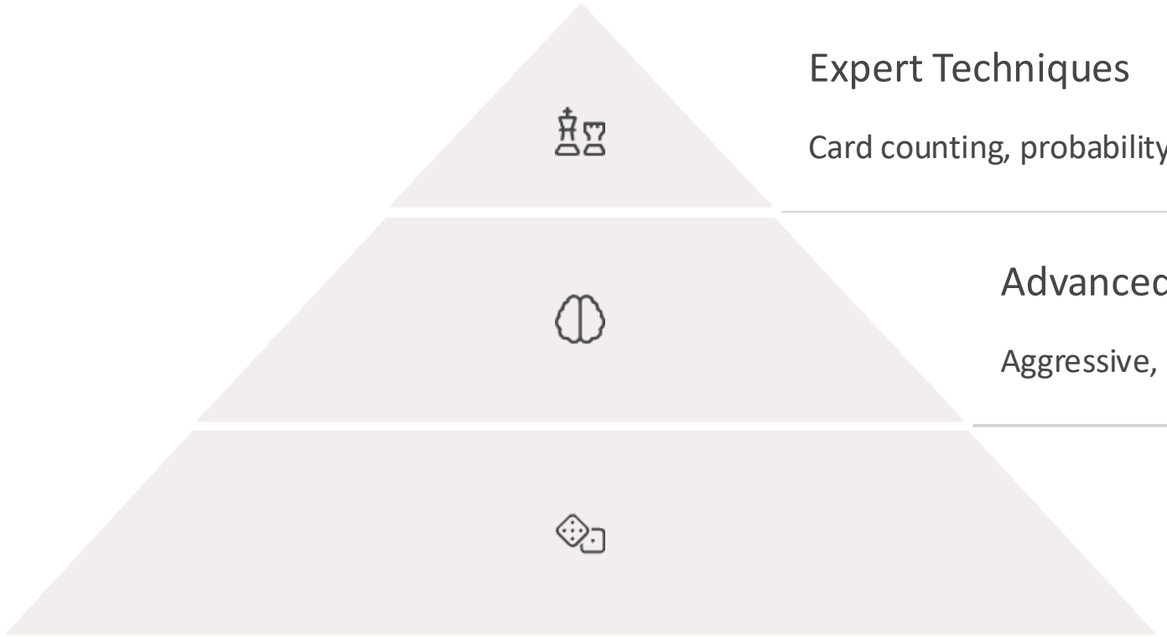
Track other players' moves to build a model of the game state and inform your decisions. Called whenever any player makes a move.

observePass(playerID)

Track when players pass, which can provide valuable information information about what cards they might not have in their hands. their hands.

These methods form the interface between your strategy and the game engine. Your implementation of selectCardToPlay() will determine your will determine your strategy's effectiveness, while the observe methods allow you to gather intelligence about the game state.

Strategy Approaches



Expert Techniques

Card counting, probability analysis, multi-turn lookahead, opponent modeling

Advanced Strategies

Aggressive, conservative, blocking, adaptive approaches

Basic Strategies

Random play, greedy algorithms (provided baselines)

Your strategy can range from simple to complex. Basic strategies like random or greedy play are easy to implement but limited in effectiveness. Advanced strategies consider the game state more carefully, while expert techniques incorporate sophisticated algorithms to maximize winning potential.

Implementation Steps

Create Your Strategy File

Copy StudentTemplate.cpp and rename it to reflect your strategy (e.g., MyAwesomeStrategy.cpp).

Rename the Class

Change the class name from StudentStrategy to your unique strategy name, and update getName() to return this name.

Implement Required Methods

Code the initialize(), selectCardToPlay(), observeMove(), observePass(), and getName() methods.

Add Factory Function

Include the required factory function: extern "C" PlayerStrategy* createStrategy() { return new YourStrategyClass(); }

Compile and Test

Build your strategy as a shared library and test it against the provided sample strategies.

Follow these steps methodically to ensure your implementation meets all requirements. Pay special attention to the factory function, which is essential for the is essential for the game framework to load your strategy.