Big2-C++

Emiliano Manalo

May 2025

1 Introduction

I decided to program the game Big Two (also known as Posoi-Dos) because it's a game I played often with my family growing up. It remains a staple at family gatherings, which made it a meaningful and fun project to take on. I spent about a week working on it off and on. The final version consists of approximately 1,700 lines of code, and the program is structured around four main classes: Card, Deck, Player, and PlayingHand. Each class handles a different aspect of the game. For instance, the Player class represents a game participant—either AI or human—and contains a personal Deck and PlayingHand that interact with Card objects to manage the player's gameplay. If you'd like to see the code in full, you can visit the GitHub Repository to see my code.

2 Approach To Development

During development, I made significant use of the C++ Standard Template Library (STL), especially container types like deque, list, and map, as well as adaptors like stack and queue. I also relied on algorithms such as sort, shuffle, and find. Iterators were used throughout to navigate collections efficiently. These features were all necessary to implement the game's logic in a clean and modular way. Version control was handled via GitHub, where I committed changes each time a feature was added and tested successfully. This approach ensured a stable and well-documented development process.

3 Game Rules

The rules of Big2 are straightforward yet rich in strategic possibilities. Each of the four players is dealt thirteen cards from a standard 52-card deck. The player holding the 3 of Clubs goes first. Players take turns attempting to play a valid hand that beats the previous one, with hand types based on poker rules—singles, pairs, two pairs, three of a kind, and five-card combinations like straights, flushes, full houses, and more. A played hand must be of the same type as the previous and must be of a higher rank; if no valid play is possible,

the player may pass. Once three players pass in a row, the field resets and the last player who played a hand may start a new round with any hand type. The game ends immediately when a player plays all of their cards.

4 Description Of Code

The code architecture is built around four key classes. The Card class encapsulates a card's rank and suit, provides operator overloading for comparisons, and includes functions to display or retrieve card data. The Deck class uses a deque¡Card¿ to allow efficient operations at both ends and includes methods to shuffle, deal, and sort cards. The PlayingHand class holds a list of cards selected for play and contains all the logic needed to determine what type of hand it represents, what its rank is, and whether it can beat another hand. Finally, the Player class models both human and AI participants. It manages each player's personal deck and playing hand, and includes turn-taking logic as well as decision-making functions, including AI logic to evaluate and select optimal hands. All of this is coordinated in the main game loop within main.cpp, which initializes the game, distributes cards, establishes turn order, processes player actions, and checks for victory conditions.

From a technical standpoint, this project demonstrates a wide range of C++ competencies. The container classes used include deque in the deck for fast card operations, list in the hand for flexible iteration, and both map and unordered map for data association and frequency counting. Sets were employed to sanitize player input by eliminating duplicates. The game's logic also utilizes container adaptors like stack for hand history and queue for managing turn order. Iterators of all major types—forward, bidirectional, and random access—appear throughout the code. Algorithms like sort, shuffle, find, and max element are applied across the project to process collections efficiently and intelligently, particularly during hand evaluation and AI decision-making.

In gameplay, the program opens by shuffling and distributing cards. It identifies the player who holds the 3 of Clubs and begins the turn rotation from that player. Each turn, the current player either plays a higher-ranking valid hand or passes. AI players evaluate their best available hand based on game state, while human players interactively select cards using indexed prompts. The game tracks how many cards each player has and ends when a player depletes their hand. After three passes, the play resets and the most recent player to have played gets to lead with any hand type. The system effectively simulates the flow of a real Big2 match with smooth transitions, AI behavior, and accurate rule enforcement.

5 Sample Input/Output

```
=== Card Counts ===
Player 1: 13 cards
Player 2: 13 cards
Player 3: 13 cards
Player 4: 13 cards
==================

=== Player 1's Turn ===
=====[[Your Turn]]=====
You can play any valid hand combination!
----YOUR CARDS----
0. [ ] 10 of &
1. [ ] Q of O
2. [ ] 7 of O
3. [ ] K of ^
4. [ ] J of O
5. [ ] 8 of &
6. [ ] Q of V
7. [ ] K of O
8. [ ] 3 of &
9. [ ] 8 of ^
10. [ ] A of V
11. [ ] 2 of &
12. [ ] 9 of O
Select Cards By Typing Their Index (or press Enter to skip): 8
----SELECTED CARDS----
0. [ ] 10 of &
1. [ ] Q of O
2. [ ] 7 of O
3. [ ] K of ^
4. [ ] J of O
5. [ ] 8 of &
6. [ ] Q of V
7. [ ] K of O
8. [X] 3 of &
9. [ ] 8 of ^
10. [ ] A of V
11. [ ] 2 of &
12. [ ] 9 of O
====HAND BEING PLAYED====
HAND: High Card
RANK: 3
SUIT: &
```

```
Type o to confirm, type anything else to reselect: o

=== Card Counts ===
Player 1: 12 cards
Player 2: 13 cards
Player 3: 13 cards
Player 4: 13 cards
==================

=== Player 2's Turn ===
==LAST PLAYED HAND==
HAND: High Card
RANK: 3
SUIT: &
Cards:
3 of &
=====[[AI Turn]]=====
===HAND BEING PLAYED===
HAND: High Card
RANK: 2
SUIT: V

=== Card Counts ===
Player 1: 12 cards
Player 2: 12 cards
Player 3: 13 cards
Player 4: 13 cards
==================

=== Player 3's Turn ===
==LAST PLAYED HAND==
HAND: High Card
RANK: 2
SUIT: V
Cards:
2 of V
=====[[AI Turn]]=====
AI passes
Player 3 passes

=== Card Counts ===
Player 1: 12 cards
Player 2: 12 cards
Player 3: 13 cards
Player 4: 13 cards
==================
```

```
=== Player 4's Turn ===
==LAST PLAYED HAND==
HAND: High Card
RANK: 2
SUIT: V
Cards:
2 of V
=====[[AI Turn]]=====
AI passes
Player 4 passes

=== Card Counts ===
Player 1: 12 cards
Player 2: 12 cards
Player 3: 13 cards
Player 4: 13 cards
==================

=== Player 1's Turn ===
==LAST PLAYED HAND==
HAND: High Card
RANK: 2
SUIT: V
Cards:
2 of V
=====[[Your Turn]]=====
==LAST PLAYED HAND==
2 of V
----YOUR CARDS----
0. [ ] 10 of &
1. [ ] Q of O
2. [ ] 7 of O
3. [ ] K of ^
4. [ ] J of O
5. [ ] 8 of &
6. [ ] Q of V
7. [ ] K of O
8. [ ] 8 of ^
9. [ ] A of V
10. [ ] 2 of &
11. [ ] 9 of O
Select Cards By Typing Their Index (or press Enter to skip):
____
```

6 Check Off Sheet

1. Container Classes
   (a) list (PlayingHand.h::20, Player.h:220) Used in storing and managing player input in what hand to play.
   (b) set (Player.h:227) Ensures that there is no selected duplicate indexes when a hand is chosen
   (c) map (Card.h:8-22,24-29, Player.h:14-26) Used to keep a record of what are the suit and card rankings.
   (d) stack (main.cpp:28) Tracks the last played hand in the session
   (e) queue (main.cpp:29) Used to manage player turns in the game.
2. Iterators
   (a) Trivial Iterator: Often used as pointers
   (b) Input Iterator: Reads all the elements of a sequence
   (c) Output Iterator: Writes elements to a sequence
   (d) Forward Iterator: Read and write to a sequence
   (e) Bidirectional Iterator: Used to traverse from either direction of a list
   (f) Random Access Iterator: Used to randomly access an element, like in shuffle or pick a random card.
3. Algorithms
   (a) find (Player.h:228) Used to find a certain card.
   (b) count (PlayingHand.h:208) Used to count the number of ranks or suits in a given hand.
   (c) random shuffle (Deck.h:62) Used in shuffling the card deck.
   (d) remove (PlayingHand.h:133) Removes played cards from the current deck.
   (e) sort (Deck.h:108-113) Orders deck by suit and rank
   (f) max element (PlayingHand.h:365) Finds the highest card for hand evaluations.

7 Documentation Of Code

```
PROGRAM Big2_Game
    // Initialization
    deck = InitializeDeck()
    Shuffle(deck)

    players = [Player1, AI1, AI2, AI3]
    FOR each player IN players
        DealCards(player, 13)
    END FOR

    first_player = FindPlayerWithThreeOfClubs(players)
    turn_queue = InitializeTurnQueue(first_player)
    hand_history = EmptyStack()
    consecutive_passes = 0

    // Game Loop
    WHILE True
        current_player = turn_queue.Front()
        turn_queue.Pop()

        DISPLAY "=== Player " + current_player.Name + "'s Turn ==="

        // Get current hand to beat
        IF NOT hand_history.Empty()
            current_hand = hand_history.Top()
            DISPLAY "Current hand to beat:"
            DisplayHand(current_hand)
        ELSE
            current_hand = EmptyHand()
        END IF

        // Player decision
        played_hand = current_player.Decision(current_hand)

        // Handle pass or play
        IF played_hand.IsEmpty()
            DISPLAY current_player.Name + " passes"
            consecutive_passes += 1
        ELSE
            hand_history.Push(played_hand)
            consecutive_passes = 0
            DISPLAY "Played hand:"
            DisplayHand(played_hand)
```

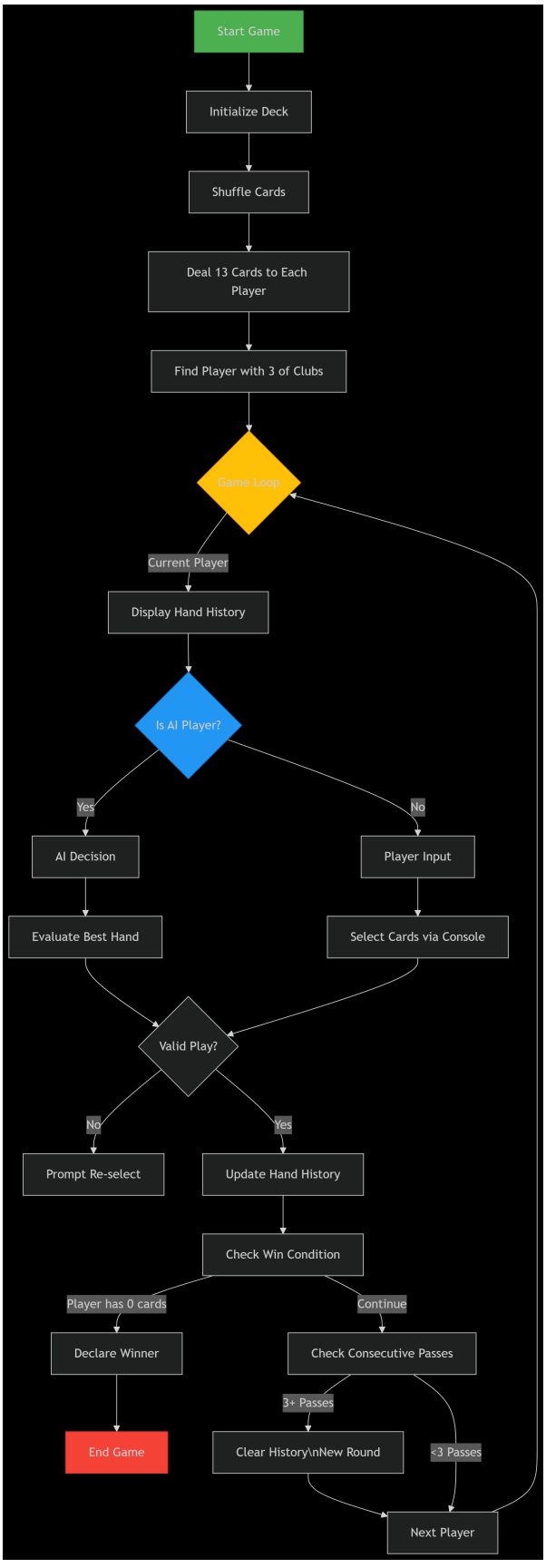

Figure 1: Flow Chart

```
        END IF

        // Check win condition
        IF current_player.CardCount() == 0
            DISPLAY current_player.Name + " wins!"
            BREAK
        END IF

        // Check for new round
        IF consecutive_passes >= 3
            DISPLAY "=== New Round ==="
            hand_history = EmptyStack()
            consecutive_passes = 0
        END IF

        turn_queue.Push(current_player)
    END WHILE
END PROGRAM

//Player Class Logic
FUNCTION Decision(current_hand) RETURNS PlayingHand
    IF is_ai
        RETURN AITurn(current_hand)
    ELSE
        RETURN HumanTurn(current_hand)
    END IF
END FUNCTION

FUNCTION HumanTurn(current_hand) RETURNS PlayingHand
    WHILE True
        DISPLAY "Your cards:"
        DisplayCards(player_hand)

        selected_indices = GetPlayerSelection()

        IF selected_indices.Empty()
            RETURN EmptyHand()  // Player passes
        END IF

        played_hand = CreateHandFromSelection(selected_indices)

        IF IsValidPlay(played_hand, current_hand)
            RETURN played_hand
        ELSE
            DISPLAY "Invalid play! Try again."
        END IF
```

```
    END LOOP
END FUNCTION

FUNCTION AITurn(current_hand) RETURNS PlayingHand
    IF ShouldPass(current_hand)
        RETURN EmptyHand()
    END IF

    possible_hands = GeneratePossibleHands(current_hand)
    best_hand = SelectBestHand(possible_hands)

    RETURN best_hand
END FUNCTION
//Deck Operations
FUNCTION InitializeDeck() RETURNS Deck
    deck = EmptyDeck()

    FOR suit IN 1..4
        FOR rank IN 1..13
            deck.Add(Card(rank, suit))
        END FOR
    END FOR

    RETURN deck
END FUNCTION

FUNCTION Shuffle(deck)
    seed = GetTimeBasedSeed()
    random_engine = InitializeRandom(seed)
    ShuffleAlgorithm(deck.cards, random_engine)
END FUNCTION
//Hand evaluations
FUNCTION EvaluateHand(hand) RETURNS (type, rank, suit)
    counts = CountCardRanks(hand)
    suits = CountCardSuits(hand)

    IF IsRoyalFlush(hand)
        RETURN (10, HighestRank(hand), HighestSuit(hand))
    ELSE IF IsStraightFlush(hand)
        RETURN (9, HighestRank(hand), HighestSuit(hand))
    // ... other hand types ...
    ELSE IF IsHighCard(hand)
        RETURN (1, HighestRank(hand), HighestSuit(hand))
    ELSE
        RETURN (0, -1, -1)  // Invalid hand
    END IF
```

```
END FUNCTION
//Data structures/classes
CLASS Card
    PROPERTIES:
        rank: INTEGER (1-13)
        suit: INTEGER (1-4)
    METHODS:
        Display()
END CLASS

CLASS Deck
    PROPERTIES:
        cards: DEQUE<Card>
    METHODS:
        Shuffle()
        Deal(num_cards)
END CLASS

CLASS Player
    PROPERTIES:
        hand: LIST<Card>
        is_ai: BOOLEAN
    METHODS:
        PlayHand(current_hand)
        Pass()
END CLASS

CLASS PlayingHand
    PROPERTIES:
        cards: LIST<Card>
        type: INTEGER
        highest_rank: INTEGER
        highest_suit: INTEGER
    METHODS:
        Evaluate()
        Compare(other_hand)
END CLASS
```

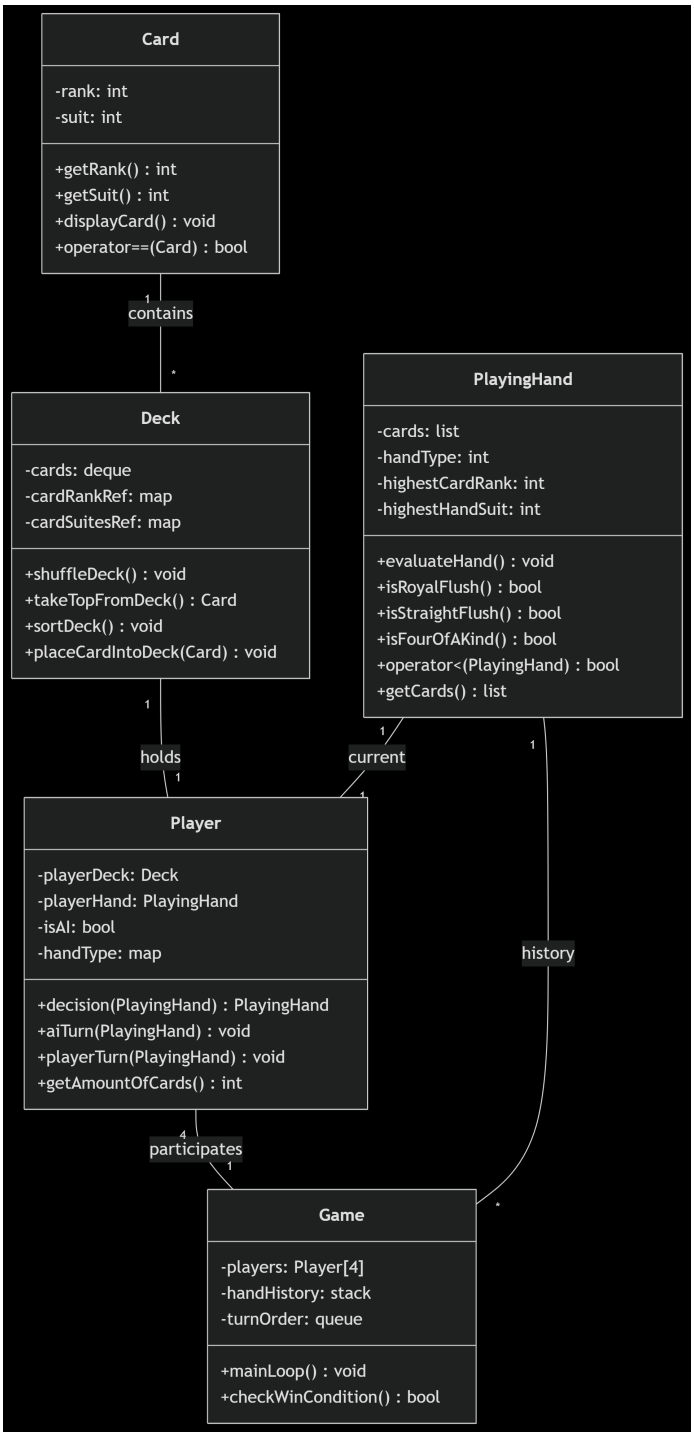

Figure 2: UML Diagram