BLACKJACK

A CARD GAME

Ireoluwa Dairo

CSC/CIS 17C -42513

SPRING 2025

Project 1

INTRODUCTION

This project is a modified version of a C++ Blackjack game that makes use of the Standard Template Library (STL). I chose Blackjack because it's a well-known card game with clear rules. The game sticks to the classic rules of Blackjack: the objective is to get a hand value as close to 21 as possible without going over.

At the start of each round, both the player and the dealer are dealt two cards. The player can choose to "hit" (draw another card) or "stand" (keep their hand). The dealer plays after the player, following standard casino rules they keep hitting until their hand is at least 17. Once both sides finish, the game checks who is closest to 21 without busting to determine the winner.

Github - CSC_17C/project at main · Kvngjaid04/CSC_17C

Summary

This project has over 700 lines of code and 30+ functions, with a strong focus on using STL. It makes use of several key concepts:

- **STL Containers:** vector, list, map, set, stack, and queue
- **STL Algorithms:** for_each, transform, count_if, generate, random_shuffle, find
- **Iterators:** used throughout the code
- **Lambda functions:** paired with STL algorithms for cleaner logic

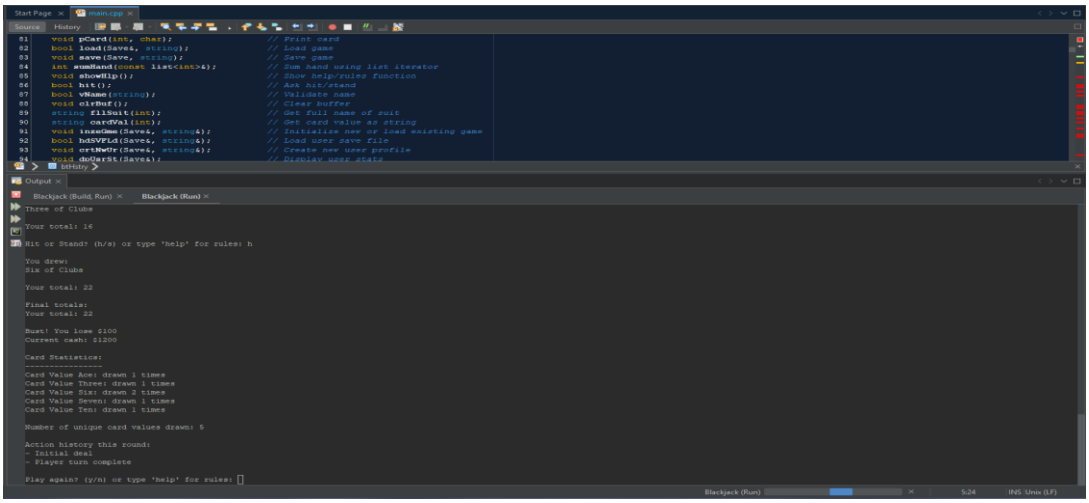
The biggest change from the original version was moving away from raw arrays and pointers to STL containers.

Description

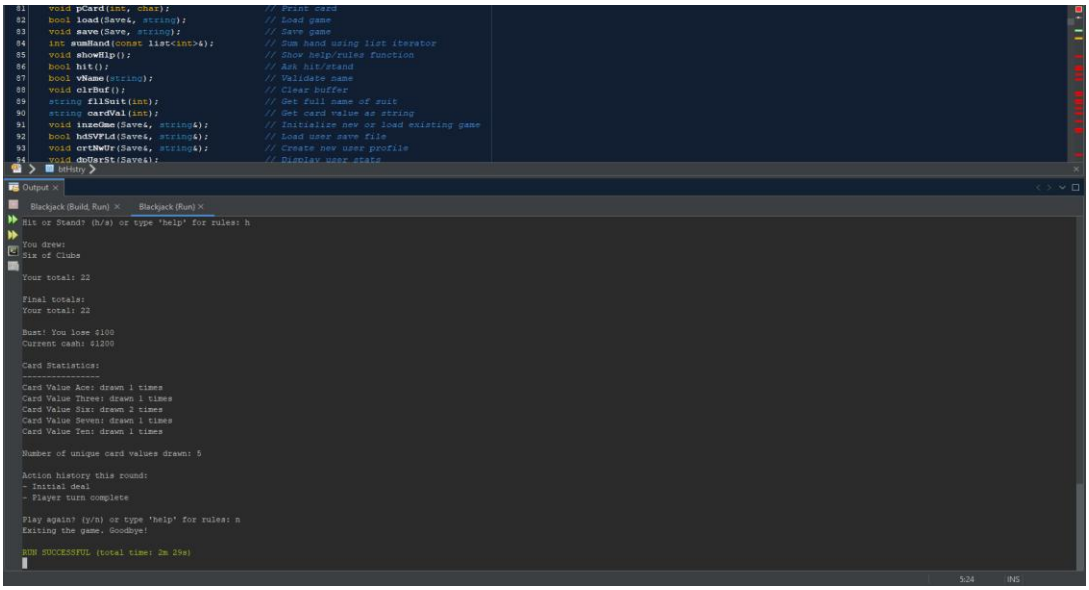
I broke the game into smaller functions and used STL containers and algorithms to keep things efficient and clean. I replaced regular arrays with vectors, maps, sets, and queues, and used custom structs with operators to organize the data. When you start the game, it greets you and asks if you want to load a saved profile or start fresh. New players get $1000 to play with. You can view the rules before playing.

Each round, you place a bet, get your cards, and decide to hit or stand. The dealer plays by standard rules hits until reaching 17 or higher. After each round, the game shows stats: card frequency (map), unique cards seen (set), and action history (queue). Your progress saves automatically to a file named after your username. You can keep playing until you quit or go broke. The gameplay follows classic Blackjack rules and shows off how flexible STL can be. There is a sample profile “Player1” included to test save/load features and stats tracking.

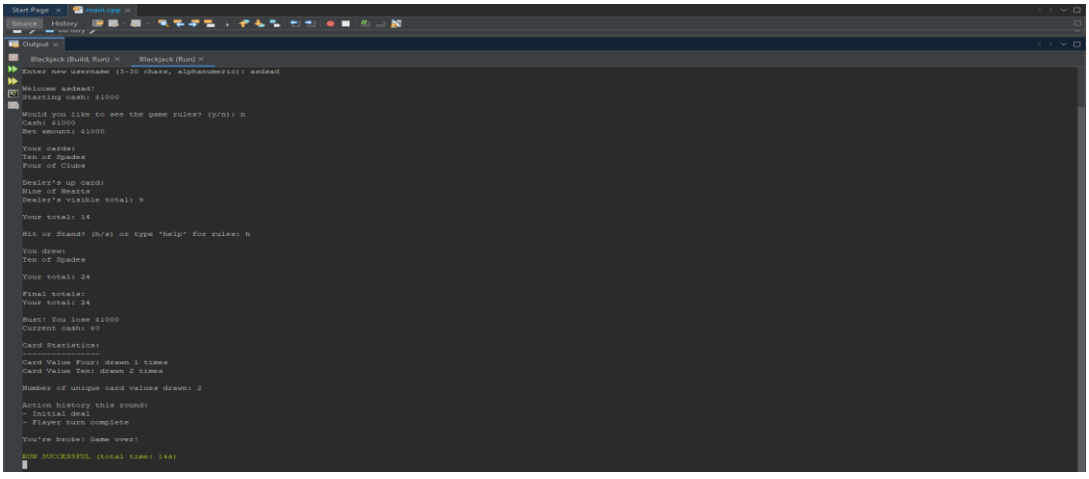
Sample Input/Output



```
Three of Clubs

Your total: 16

Hit or Stand? (h/s) or type 'help' for rules: h

You drew:
Six of Clubs

Your total: 22

Final totals:
Your total: 22

Bust! You lose $100
Current cash: $1200

Card Statistics:
----------------
Card Value Ace: drawn 1 times
Card Value Three: drawn 1 times
Card Value Six: drawn 2 times
Card Value Seven: drawn 1 times
Card Value Ten: drawn 1 times

Number of unique card values drawn: 5

Action history this round:
- Initial deal
- Player turn complete

Play again? (y/n) or type 'help' for rules:
```

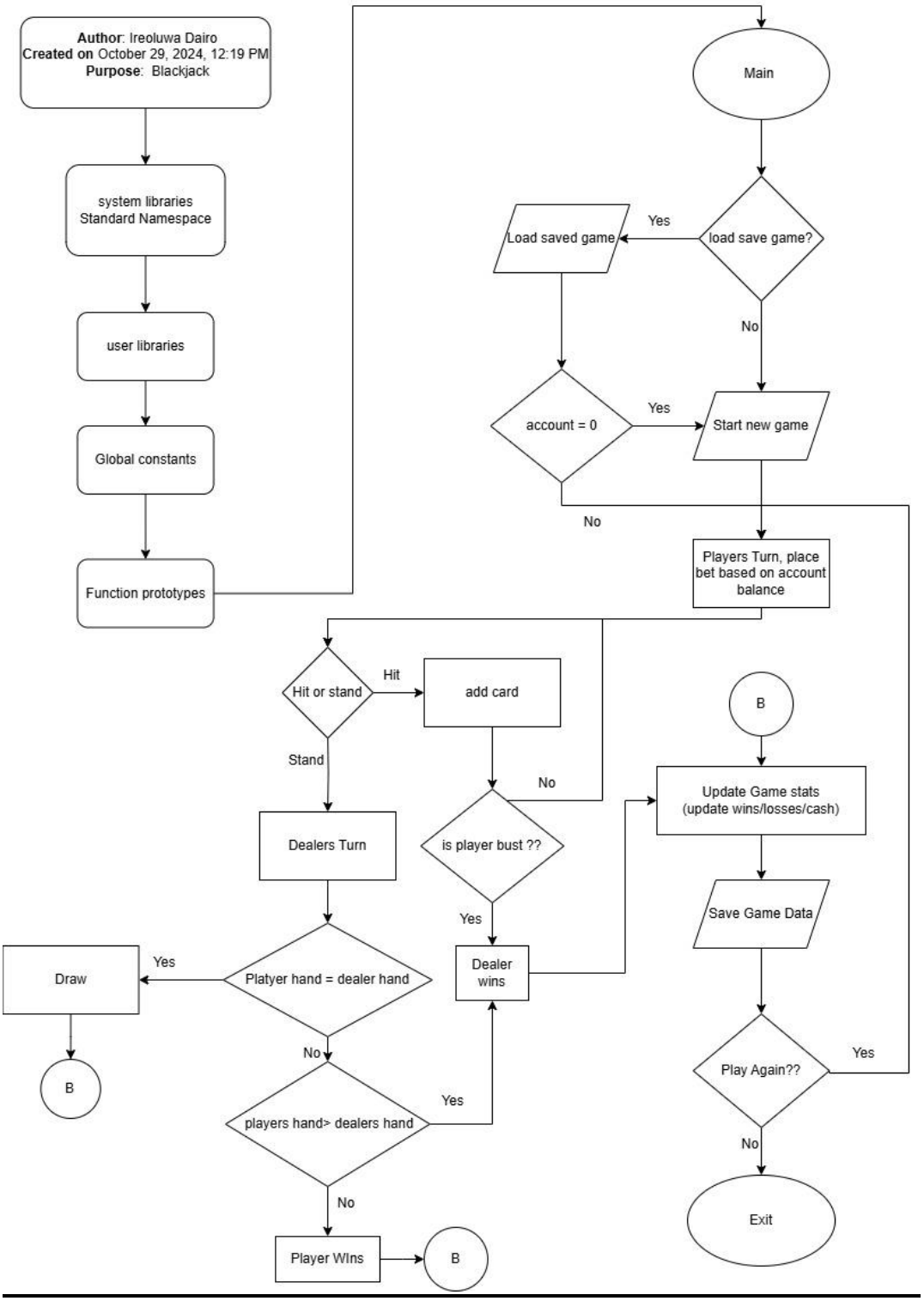


```
Hit or Stand? (h/s) or type 'help' for rules: h

You drew:
Six of Clubs

Your total: 22

Final totals:
Your total: 22

Bust! You lose $100
Current cash: $1200

Card Statistics:
----------------
Card Value Ace: drawn 1 times
Card Value Three: drawn 1 times
Card Value Six: drawn 2 times
Card Value Seven: drawn 1 times
Card Value Ten: drawn 1 times

Number of unique card values drawn: 5

Action history this round:
- Initial deal
- Player turn complete

Play again? (y/n) or type 'help' for rules: n
Exiting the game. Goodbye!

RUN SUCCESSFUL (total time: 2m 29s)
```



```
Enter new username (3-30 chars, alphanumeric): asdsad

Welcome asdsad!
Starting cash: $1000

Would you like to see the game rules? (y/n): n
Cash: $1000
Bet amount: $1000

Your cards:
Ten of Spades
Four of Clubs

Dealer's up card:
Nine of Hearts
Dealer's visible total: 9

Your total: 14

Hit or Stand? (h/s) or type 'help' for rules: h

You drew:
Ten of Spades

Your total: 24

Final totals:
Your total: 24

Bust! You lose $1000
Current cash: $0

Card Statistics:
----------------
Card Value Four: drawn 1 times
Card Value Ten: drawn 2 times

Number of unique card values drawn: 2

Action history this round:
- Initial deal
- Player turn complete

You're broke! Game over!

RUN SUCCESSFUL (total time: 14s)
```

FLOWCHART



Note: The original flowchart from the documentation still accurately represents the overall game flow. Even though this updated version introduces STL the core logic remains the same.

Pseudocode

Start program

Initialize random number generator

Create STL containers:

- vector<Card> deck for the card deck
- list<int> PlyrHnd, DlrHand for player and dealer hands
- map<int, int> CrdStat for card statistics
- set<Card> UnqCard for tracking unique cards
- queue<string> AcnHtry for recording game actions
- stack<int> btHstry for betting history

Call function to initialize game or load saved game

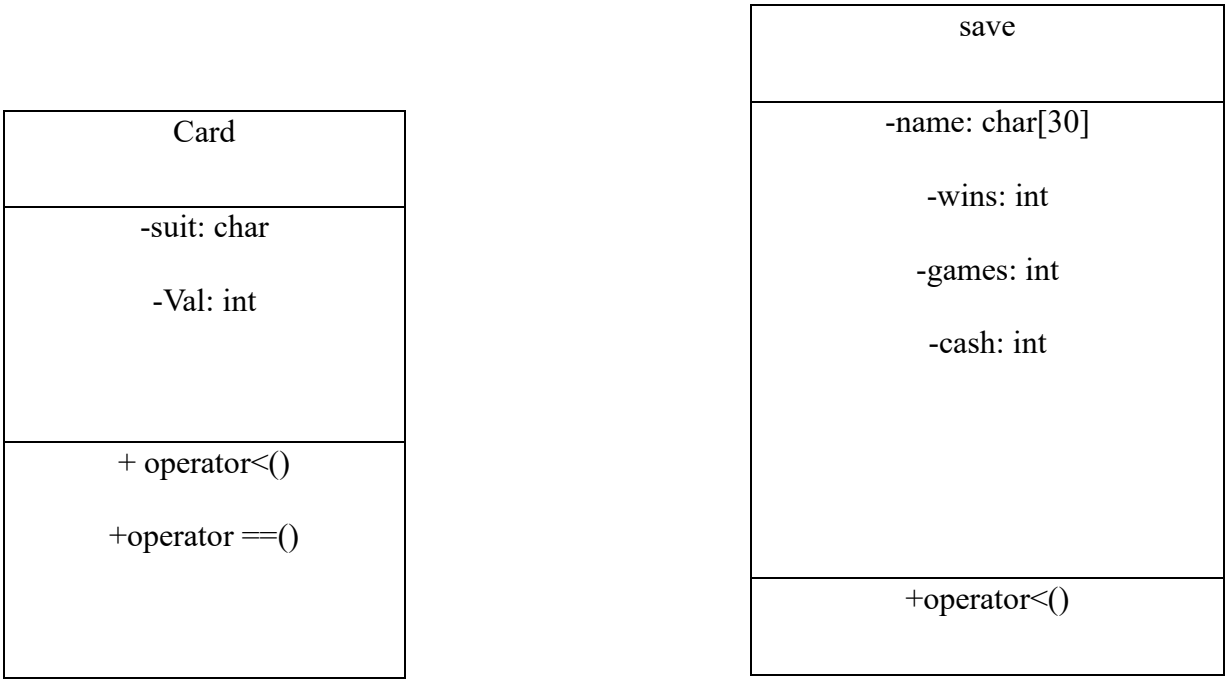
Repeat the game loop:

- Initialize the deck using STL generate algorithm
- Shuffle deck using STL random_shuffle algorithm
- Get the player's bet and push to bet history stack
- Deal initial cards to player and dealer, adding to lists
- Display the initial cards
- Record action in history queue
- Play the player's turn:
  - Calculate hand value using STL for_each algorithm
  - If hit requested, add card to the player’s list
  - Update game statistics using STL containers
- If player has not busted, play dealer's turn using iterators
- Record action in history queue
- Handle the outcome and update statistics
- Save the game state
- Display card statistics using STL for_each with lambda
- Report unique cards from set container
- Display action history from queue
- Check if the player can continue
- Clear containers for the next round if playing again

End game

End program

UML



CHECKOFF SHEET

1) Container classes (Where in code did you put these Concepts and how were they used?
    1. Sequences (At least 1)
        a. **List (line 120-121)** - to store player and dealer hands
        b. slist
        c. bit_vector
    2. Associative Containers (At least 2)
        a. **Set (Line 129)** - Tracks unique cards seen during gameplay
        b. **Map (Line 128)** - Maps card values to their frequency for statistics
        c. Hash
    3. Container adaptors (At least 2)
        a. **Stack (line 131**) - Records betting history
        b. **Queue (line (130)** - Tracks game actions
        c. priority_queue
        d. Iterators

2) Iterators
    1. Concepts (Describe the iterators utilized for each Container)
        a. Trivial Iterator
        b. Input Iterator
        c. Output Iterator
        d. Forward Iterator
        e. Bidirectional Iterator
        f. Random Access Iterator

3) Algorithms (Choose at least 1 from each category)
    1. Non-mutating algorithms
        a. **for_each (Line 155)** - to process map entries for card statistics
        b. find
        c. count
        d. equal
        e. search
    2. Mutating algorithms
        a. copy
        b. Swap
        c. Transform (line 402) - Converts player input to lowercase
        d. Replace
        e. fill
        f. Remove
        g. Random_Shuffle (Line 506) – shuffles the deck
    3. Organization
        a. Sort (Line 43) - Used for sorting Cards
        b. Binary search
        c. merg
        d. inplace_merge
        e. Minimum and maximu

Variables and Data Structures

STL Containers:

- list<int> PlyrHnd, DlrHand: Store player and dealer hands.
- map<int, int> CrdStat: Tracks how often each card value appears.
- set<Card> UnqCard: Tracks unique cards using a custom operator<.
- queue<string> AcnHtry: Logs game actions in order.
- stack<int> btHstry: Keeps a history of bets.

Structures with Custom Operator:

- Card: Includes operator< and operator== for use in STL containers.
- Save: Includes operator< for possible sorting or mapping.

Other variables handle game state like hand values, win/loss tracking, and user data.

Concepts used

- **Containers:** vector, list, map, set, queue, stack
- **Algorithms:** for_each, count_if, transform, generate, random_shuffle
- **Iterators:** Used for container traversal and operations
- **Lambda expressions:** Paired with STL algorithms

REFERENCES

1. Originally developed for CSC-5, enhanced for CSC-17A, and now updated with STL implementation for CSC-17C by Ireoluwa Dairo
2. C++ STL Documentation Standard Template Library Programmer's Guide
3. Blackjack rules: https://www.bicyclecards.com/how-to-play/blackjack/