**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](https://github.com/Kvngjaid04/CSC_17C/tree/main/project)

**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**

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

A screenshot of a computer

AI-generated content may be incorrect.

**FLOWCHART**

**A diagram of a flowchart

Description automatically generated**

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

|  |
| --- |
| save |
| -name: char[30]  -wins: int  -games: int  -cash: int |
| +operator<() |

**UML**

|  |
| --- |
| Card |
| -suit: char  -Val: int |
| + operator<()  +operator ==() |

**CHECKOFF SHEET**

1. Container classes (Where in code did you put these Concepts and how were they used?
2. Sequences (At least 1)
   1. **List (line 120-121)** - to store player and dealer hands
   2. slist
   3. bit\_vector
3. Associative Containers (At least 2)
4. **Set (Line 129)** - Tracks unique cards seen during gameplay
5. **Map (Line 128)** - Maps card values to their frequency for statistics
6. Hash
7. Container adaptors (At least 2)
8. **Stack (line 131**) - Records betting history
9. **Queue (line (130)** - Tracks game actions
10. priority\_queue
11. Iterators
12. Iterators
    1. Concepts (Describe the iterators utilized for each Container)
13. Trivial Iterator
14. Input Iterator
15. Output Iterator
16. Forward Iterator
17. Bidirectional Iterator
18. Random Access Iterator
19. Algorithms (Choose at least 1 from each category)
20. Non-mutating algorithms
21. **for\_each (Line 155)** - to process map entries for card statistics
22. find
23. count
24. equal
25. search
26. Mutating algorithms
27. copy
28. Swap
29. Transform (line 402) - Converts player input to lowercase
30. Replace
31. fill
32. Remove
33. Random\_Shuffle ( Line 506) – shuffles the deck
34. Organization
35. Sort (Line 43) - Used for sorting Cards
36. Binary search
37. merg
38. inplace\_merge
39. 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](http://209.129.8.7/~MarkLehrSyllabi/sgi-stl-docs/docs/)
3. Blackjack rules: <https://www.bicyclecards.com/how-to-play/blackjack/>