CIS-17C Project Documentation

“Blackjack! With Friends”

Saim Ahmed

May 15th, 2025

Advisor: Mark Lehr

Introduction:

I've decided to expand upon my existing version of Blackjack. I chose this because I wanted to add more meat to a relatively simple, yet fun and engaging card game. I like to play blackjack (in a non-gamblers way) with friends at parties or events, so I know my way around it. With this expansion of the game's nature, I believe there's a lot of room to implement features that make the process of playing blackjack more interesting and friendly, especially as you increase the number of people involved.

I spent around a week designing, implementing, and debugging the program, with about an average of 3-4 hours per day on it. The project is approximately 1050 lines (pure code), and there are 7 classes and 1 standalone function being the main method.

This project is stored and controlled through a github repository.

Approach to Development:

My approach to developing the implementation of blackjack was to try and compartmentalize each feature and concept into its own class. From each card to the game flow itself. I tried my best to utilize a variety of data structures while also ensuring their implementations made sense, like using a priority queue to ensure events fall into logic flow as per blackjacks rules. Most of my time was spent reading documentations and StackOverflow commentary on how these structures work and interact with one another, as well as libraries to use in order

to get the most out of them. GeeksForGeeks was also of exceptional help in learning how to work with data structures (iterators, storing information, accessing information).

Version control was preserved through github, in order to ensure all data and updates were saved. Some iterations of the version are entirely for functionality, while others simply add more detail and refining to existing features.

Game Rules:

1. Players and Bets:
    - Players are each prompted to make a wager.
    - Dealer grants two cards, one faced down, another up.
2. Player decisions
    - Players can decide to hit (take another card), or stand (keep current hand).
    - Players who exceed 21 points automatically lose their wagers.
    - After all players have completed their decisions, the dealer turns.
3. Dealer:
    - Dealer follows through with revealing hand and must hit until their hand value is 17 or higher
4. Player Goal:
    - The ideal scenario for each player is to get a blackjack (automatic win), score higher than the dealer, or have the dealer bust.
    - If the dealer busts, all remaining players win.
5. Dealer Goal:
    - Dealers' hands must exceed players to win, or get a blackjack to win automatically.

    - If the players win, payout their wager reward, if they lose, remove their wager, and if they tie, return their wager entirely.
6. Winning: For players, the goal is to beat the dealer's hand by all means necessary. Doing so means the wager is won and doubled by some casino value.

Program Classes:

Card Class:
- Represents a single card within a deck with an added suit and rank.
- Functions to create cards, get suits, get ranks, flip, get values, and a toString menu for game flow.
- To be used in the Deck Class.

Deck Class:
- Creates a deck of 52 cards (standard deck) to be used by the game players and dealer logic.
- Randomly shuffles cards as needed.
- Deck functions to populate, deal, add and retrieve from discard pile, and iterators to traverse through deck.

Hand Class:
- Represents both player and dealer hand (for compartmentalization purposes).
- Keeps track of hand values, clears hand, adds card, checks for bust or blackjack.

Player Class:
- Represents non-Robot players in the game.
- Tracks player name, money, current bet, and current hand.

- Used to grant wins, losses, or tie pushes, as well as track busts, hits, blackjacks, and hands.

Dealer Class:

- Represents all non-Player actions and overrides within game
- Functions to match human player traits (hits, busts, show hand).
- Admin-like functions to shuffle deck, deal a card, and signal an empty deck.

GameStats Class:

- Represents leaderboard and tracker of player stats within the game.
- Used to record wins, losses, high scores, and get them accordingly.

Game Class:

- Primary logic flow class.
- Adds and removes players
- Controls gameplay, dealing cards, placing bets, player turn, dealer turns, payouts, and cleanup
- Records and updates highscores, wins, losses, and action logs.
- Graphic interface stored here too.

Sample Input/Output:

```
=====WELCOME TO BLACKJACK! WITH FRIENDS=====
=====Your goal as the player is to try and beat the dealer by getting as close to 21 without going over!
=====Each player starts with two cards. The dealer's first card is hidden.
=====Each player also must decide to hit (take another card), or stand (stop taking cards).
=====The dealer will play against you! With some mental math and a little bit of luck, you could get rich!
```

```
=====To get started, each of you must create a player profile
with your name!
=====Good luck, and have fun!

=====MENU=====
1. Play Game
2. Create Player Profile
3. Load Player Profile
4. View Stats
5. View High Scores
6. Quit Game

Enter choice: 2

Enter your name:  Saim
Creating a new profile for  Saim with $ 1000 to start with.
Player profile created successfully!

=====MENU=====
1. Play Game
2. Create Player Profile
3. Load Player Profile
4. View Stats
5. View High Scores
6. Quit Game

Enter choice: 2

Enter your name: Momo
Creating a new profile for Momo with $ 1000 to start with.
Player profile created successfully!

=====MENU=====
1. Play Game
2. Create Player Profile
```

```
3. Load Player Profile
4. View Stats
5. View High Scores
6. Quit Game

Enter choice: 1

===== PLACING BETS =====
 Saim , you've got $1000. Place your bet: $100
 Saim bet $100
Momo , you've got $1000. Place your bet: $100
Momo bet $100

===== DEALING CARDS =====
===== TABLE STATUS =====
The Dealer's hand: [FACED DOWN], 8 of Hearts
 Saim ($1000, bet: $100): 7 of Hearts, Ace of Spades, Total: 18
Momo ($1000, bet: $100): 5 of Diamonds, King of Clubs, Total: 15

===== PLAYER TURNS =====

 Saim's turn:
 Saim, do you want to hit? (y/n) y
 Saim receives: 3 of Diamonds
 Saim has 21.
 Saim stands with 21.

Momo's turn:
Momo, do you want to hit? (y/n) n
Momo stands with 15.

===== DEALER'S TURN =====
The Dealer's hand: 9 of Diamonds, 8 of Hearts, Total: 17
Dealer stands with 17.
```

```
===== RESULTS =====
 Saim: Blackjack! Won $150.
Momo: Lost $100 with 15 under dealer's 17.

EVENT:  Saim won $150 with blackjack.
EVENT: Momo lost $100 with 15 under dealer's 17.

===== PLAYER STATS =====
Player         Wins Losses   Win Rate
============================================
 Saim          1    0         100.0%
Momo           0    1         0.0%

===== HIGH SCORES =====
Rank   Player       Money
============================================
1.    Saim          $1150
2.    Momo          $900

=====PLAYER MONEY STATS=====
Current Top Earner:  Saim with $1150
Current Low Earner: Momo with $900

Continue playing? (y/n): n

=====MENU=====
1. Play Game
2. Create Player Profile
3. Load Player Profile
4. View Stats
5. View High Scores
6. Quit Game

Enter choice: 6
Goodbye!
```

(end of program)

Checkoff Sheet:

Sequences:
- List:
    a. Location: Deck Class
    b. Purpose: Used to store Cards in the form of a Deck.

Associative Containers:
- Set:
    a. Location: GameStats class
    b. Purpose: used to store player highscores in the form of a set of pairs containing the money won, and the player name.
- Map (x2):
    a. Location: (Both) Game Class
    b. Purpose: Used to store card values with key value pair for rank and suit (string, int). The second was used to store special characters for string suits (string, string).

Container Adaptors:
- Stack:
    a. Location: Deck Class
    b. Purpose: To simulate the behavior of a discard pile in a game of blackjack. Stacks follow the LIFO principle, accurate to a face down discard pile of cards.
- Queue:
    a. Location: Game Class
    b. Purpose: Used to store events of the game into an action log.
- Priority_Queue:
    a. Location: Game Class

    b. Purpose: Used to store, process, and exhaust all recorded game events. Also to create a game queue in an ordered manner by priority for game logic flow.

Iterators:

- Random Access Iterator:
    a. Location: Hand Class
    b. Purpose: Used to iterate through, compare, and increment/decrement Card object structures. Used all throughout classes that use Card objects, especially in instances where for-each loops aren't viable.

Algorithms:

- for_each:
    a. Location: (Primary) Game Class
    b. Purpose: To iterate through structures searching for specific function values for comparison and game flow.
- find:
    a. Location: GameStats Class
    b. Purpose: Used to find, collect and compare player wins, losses, update high scores, and calculate win rate.

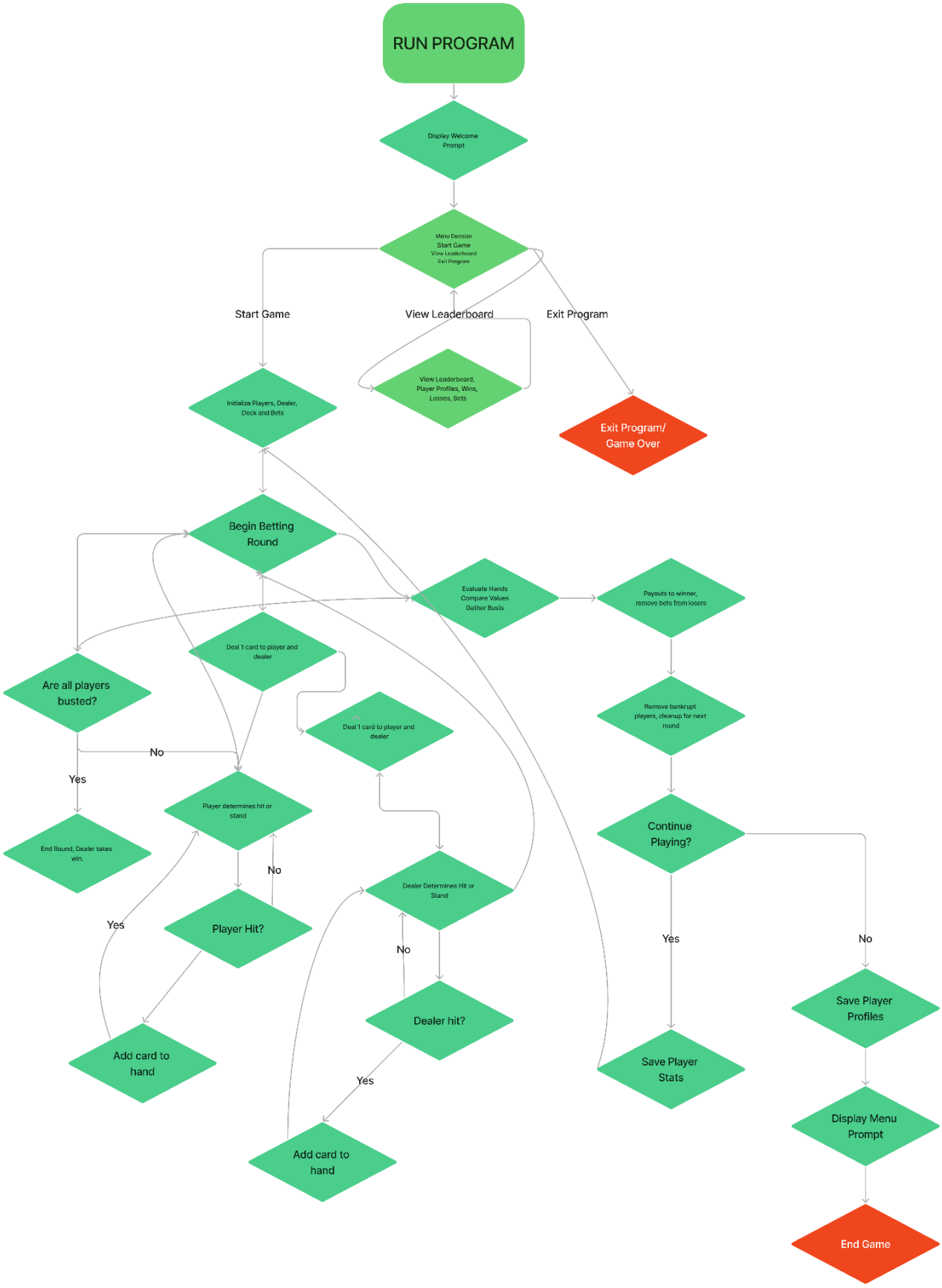
Mutating Algorithms:

- Remove:
    a. Location: Game Class
    b. Purpose: Used to erase players from a list of players and cleanup blackjack tables.
- Shuffle:
    a. Location: Deck Class
    b. Purpose: Fisher-Yates implementation of shuffle to accommodate for list iterations, used to randomize deck status for game logic

Organization:

- Minimum and Maximum:
    a. Location: Game Class
    b. Purpose: Used to find and output the highest and lowest earner in the current game of blackjack, until game ends and scores are recorded in a leaderboard.

Flowchart:


RUN PROGRAM
Display Welcome Prompt
Menu Decision
Start Game
View Leaderboard
Exit Program
Start Game
View Leaderboard
Exit Program
Initialize Players, Dealer, Deck and Bets
View Leaderboard, Player Profiles, Wins, Losses, Bets
Exit Program/ Game Over
Begin Betting Round
Evaluate Hands Compare Values Gather Busts
Payouts to winner, remove bets from losers
Deal 1 card to player and dealer
Are all players busted?
Deal 1 card to player and dealer
Remove bankrupt players, cleanup for next round
No
Yes
Player determines hit or stand
Continue Playing?
End Round, Dealer takes win.
No
Dealer Determines Hit or Stand
Yes
No
Player Hit?
Yes
No
Save Player Profiles
Dealer hit?
Save Player Stats
Display Menu Prompt
Yes
Add card to hand
Add card to hand
End Game

Pseudocode:

```
CLASS Card:
    // Represents a playing card with suit, rank and face-up
status
    ATTRIBUTES: m_suit, m_rank, m_faceUp
    CONSTRUCTOR: initialize card with suit, rank, and face
status
    METHODS:
        getRank()
        getSuit()
        isFaceUp()
               // Checks if card is face up
        flip() // Flips card face down
        == operator
               // Compares cards for equality
        < operator
               // Defines ordering between cards

CLASS Deck:
    ATTRIBUTES: cards (list), discardPile (stack)
    CONSTRUCTOR: Creates standard 52-card deck
    METHODS:
        shuffleDeck()
               // Randomizes card order
        deal()
               // Removes and returns top card
        addToDiscardPile()
               // Adds card to discard pile
        retrieveCardsFromDiscardPile()
               // Moves discards back to main deck
        isEmpty()

        cardsRemaining()
```

```
        toString()
               // Displays deck information

CLASS Hand:
    // Represents a player's or dealer's cards
    ATTRIBUTES: hand_cards (deque)
    CONSTRUCTOR: Creates empty hand
    METHODS:
        add()
        clear()
        getTotal()
               // Calculates hand value (special handling for
          aces)
        isBusted()
        isBlackjack()
        iterator methods
        toString()
               //Displays hand information
CLASS Player:
    // Represents a player in the game
    ATTRIBUTES: m_name, m_hand, m_money, m_bet
    CONSTRUCTOR: Initializes player with name and starting money
    METHODS:
        getName(), getMoney(), getBet(), getHand() // Accessor
methods
        getHandRef() // Returns modifiable reference to hand
        placeBet() // Sets bet amount if valid
        win() // Handles winning (adds money)
        lose() // Handles losing (clears bet)
        push() // Handles push/tie (returns bet)
        isHitting() // Asks player if they want another card
        showHand() // Displays player's cards
        isBusted(), isBlackjack()

CLASS Dealer (inherits from Player):
```

```
    // Represents the dealer with deck management
    ATTRIBUTES: m_deck
    CONSTRUCTOR: Initializes dealer
    METHODS:
        isHitting() // Automatic decision based on hand total
(<17 hit, ≥17 stand)
        showHand() // Shows dealer's hand (optionally hiding
first card)
        shuffleDeck() // Shuffles the deck
        deal() // Deals a card
        deckIsEmpty() // Checks if deck is empty

CLASS GameStats:
    // Tracks game statistics
    ATTRIBUTES: m_playerStats (map of wins/losses), m_highScores
(ordered set)
    CONSTRUCTOR: Initializes empty stats
    METHODS:
        recordWin(), recordLoss() // Records game outcomes
        updateHighScore() // Updates player's high score
        getWins(), getLosses(), getWinRate() // Player
statistics
        displayStats()
        displayHighScores()

CLASS Game:
    // Main game controller
    ATTRIBUTES:
        m_players (collection of Player objects)
        m_dealer, m_stats, m_actionLog, m_events
        m_currentState (tracks game phase)
        m_cardValues, m_suitNames (maps for display)
    CONSTRUCTOR: Initializes game state and card values
    METHODS:
```

```
        addPlayer() // Adds player to game
        removePlayer() // Removes player from game


        play() // Main game loop
        placeBets() // Collect bets from players
        deal() // Deal initial cards
        playerTurns() // Handle each player's turn
        dealerTurn() // Handle dealer's turn
        payouts() // Determine winners and distribute money
        cleanup() // Handle end-of-round maintenance


        setState(), getState() // Manage game state


        displayTable() // Shows current game state
        logAction() // Records game actions
        displayActionLog() // Shows history of actions
        queueEvent(), processEvents() // Event handling


        displayMenu(), processMenuChoice() // Menu system
        showWelcomeScreen() // Shows introductory message
        createPlayerProfile() // Creates new player
        loadPlayerProfile() // Loads existing player data
        findMinMaxMoney() // Identifies richest/poorest players

CLASS Main:

     ATTRIBUTES: maxPlayers, game, deck, etc.
     WHILE continuePlaying
     gameloop()
     ELSE
     END PROGRAM
```

UML Class Diagram: