UNO Game Project Report

Namal University Mianwali

Course	Data Structures
Instructor	Mr. Abdul Rafay
Submitted to	Mr. Abdul Rafay
Submission Date	January 12, 2026

Submitted by	Abdul Rehman	NUM-BSCS-2024-01
	Hammad Shabir	NUM-BSCS-2024-25
	Junaid Gondal	NUM-BSCS-2024-28
	Momina Umer	NUM-BSCS-2024-36
	Sadia	NUM-BSCS-2024-36
	Arfa Tayyabah	NUM-BSCS-2024-16

Fall – 2025

Department of Computer Science

Contents

1 Abstract

The project presents the implementation and analysis of the standard UNO card game developed using three different data structures(Stack, Queue, and Linked List). The primary objective of this project is to enhance the understanding of major data structures by studying their influence on performance, complexity of implementation, and experience of the game. All three implementations have a interactive graphical user interface (GUI) written in C++ with the Raylib library and support 4 players with all the standard UNO rules including number cards, action cards (Skip, Reverse, Draw Two), and wild cards (Wild, Wild Draw Four). The Stack-based implementation uses Last-In-First-Out order to represent the behavior of deck and discard piles providing O(1) complexity for basic card operations. However, the time complexity for random access of cards in player's hands in O(n). The Queue-based implementation supports the use of First-In-First-Out (FIFO) operations, which provide effective sequential control of cards, but has the same problem of random access. The implementation of UNO game using Linked supports more flexible operations of the cards such that we can insert and delete a card at random positions with O(1) complexity. Stack and Queue are more optimal in simulating the behavior of real-world card distribution, and Linked List is more flexible in its support of the operations of a hand. This comparative analysis shows that no data structure is generally the best for all the game operations and the choice of data structure depends on the most frequent operations during the game.

2 Introduction

Card games have been used in the entertainment of humans since ancient times played by all age groups as they are easy to understand and interesting to play. One of the most popular card games in the world is UNO which was invented by Merle Robbins in 1971. This game has deck of 108 cards which include four colors (Red, Blue, Green, Yellow), number cards (0-9), action cards (Skip, Reverse, Draw Two, Wild) and wild cards (Wild, Wild Draw Four). The goal of this game is to be the first person to empty your hand by matching the color or the value of the top card on the discard pile or each 500 points.

2.1 Objectives of the Project

The particular goals of this project are:

- Implement 3 full UNO games each using a separate data structures. At least one should be fully functional.
- Develop a working GUI for the game.
- Analyze the time complexity of all three versions and compare efficiency.
- Document all limitations of each data structure clearly.

The purpose of creating three versions is to demonstrate the strengths, weaknesses and performance differences of each data structure when applied to the same real-world problem.

3 Literature Review

The UNO game is not created by a great scientist or any other extraordinary individual, and instead it is created by the common man, a barber in Ohio, in 1971 whose name is Merle Robbins. This game is not created by direct thinking or any intention, it is merely created to resolve a family problem. Merle family played card game crazy eight game, this is a game of 52 cards, thus it has a great number of unofficial rules that people used in playing the game to make it interesting and complicated. Thus, it does not have a book of official rules. One day, while playing this game, Merle got into argument with his son which was raised on a rule because of having no rule book. Therefore, as a way of solving this rule ,Merle come up with the idea to make a deck game whereby its rules must documented to make everyone understand it easily. He devised this UNO game whereby the rule of action is written in the card such as skip, reverse, draw two so seeing the card everyone can know what to do preventing the conflict. Several projects provide the implementation of UNO game.

gameuno a public repository on GitHUb provides a C++ Version of the UNO Card game. It provides foundation for any card game since there are the classes card, deck and players. The deck is created as a static array, while linked list is used to store the cards in player's hand.

Uno (Card Game)/Python a page on rosettacode.org provides implementation of UNO game in python. It uses Tkinter librray for graphical interface.

The Codology UNO project (202X) implemented player turn order using a circular linked list, card decks with stacks/queues, and logging gameplay for persistence.

4 Game Rules and Description

The rules are extracted directly from the official Mattel UNO Colors Rule instruction booklet (DWV64-Eng.pdf).

4.1 Card Contents

- Four colors Blue, Green, Red, Yellow
- Number cards (0-9)
- 8 Draw Two cards
- 4 Reverse cards (1 each color)
- 4 Skip cards (1 each color)
- 4 Wild Draw Four cards
- 16 Wild Color Character cards

4.2 Winning Criteria

The first person to finish all cards in his hand wins. Another winning criteria is to be the first person to score 500 points in each round.

4.3 Setup

1. Shuffle the deck.
2. Deal 7 cards to each player.
3. Turn over top card of deck to start discard pile. If the top card is Wild Card, reshuffle the deck.

4.4 Gameplay

The player has to match the number, color, or symbol of the card in his hand with the top card of discard. If he does not has any matching card then he can draw one card from deck. If that card is playable, he can play it. If he doesn't want to play it, then turn will change. Special actions:

- **Draw Two**: Next player draws 2, loses turn
- **Reverse**: Reverses play direction (clockwise will become anticlockwise and vice versa).
- **Skip**: Next player loses turn.
- **Wild**: Choose a color of cards.
- **Wild Draw 4**: The current player chooses color of the card and next player draws 4 cards (challengeable).

4.5 The "UNO" shout Penalties

When a player plays second last card and after that just one card left in his hand then he must say "UNO". If he is caught by other player for not saying UNO then he has to draw two card as penalty.

4.6 Scoring

Card Type	Points
Numbers (1–9)	Face value
Draw Two / Reverse / Skip	20
Wild / Wild Draw 4	50

Table 1: UNO Colors Rule Scoring

5 Methodology

We have used 3 data structures to develop 3 separate games using stack, queue and linked list each in one.

5.1 Stack-Based Version

Justification: A custom template based Stack class is used to model the natural behavior of draw pile, discard pile, and player hands. Stack uses Last-In-First-Out order that perfectly matches with card piles since top card is always accessed first.

5.2 Queue-Based Version

Justification: A custom template based Circular-Queue (FIFO) is used for card piles. Queue is used because it uses sequential order which works well with the card game such as storing card drawing card, or changing player's turn.

5.3 Linked List-Based Version

Justification: Circular Doubly Linked Lists are used for player hands and circular turn order. It provides maximum flexibility for random card insertion and deletion.

All versions share:

- Card, Player, Deck, Game logic classes
- Raylib GUI with card rendering, buttons, and hover effects
- Full rule enforcement (UNO call, challenges, color powers, scoring)

6 Algorithms of All Applied Data Structures

6.1 Stack Version (LIFO)

1. Game starts.
2. Deck is created and shuffled.
3. Players are created.
4. Each player gets 7 cards.
5. One card is placed on discard pile.
6. First player turn is selected.
7. Game starts in loop.
8. On player's turn:
    - Player check cards with top discard card.
    - Matching is done by color, number or symbol.
9. If player has matching card:
    - Player plays the card. Card is removed from his hand and is placed on discard pile.
    - he can also choose to draw card.

10. If player has no matching card then draw card button is shown. Player draws one card from draw pile.
11. After drawing card:
    - If drawn card is playable:
      - Player get option to play or skip.
    - If player plays the card, it is placed on discard pile.
    - Else turn is skipped.
12. If special card is played:
    - Draw Two: next player draw 2 cards and miss turn.
    - Reverse: direction of play is changed.
    - Skip: next player turn is skipped.
    - Wild cards allow color selection.
13. After every turn:
    - Check UNO condition.
    - If player has one card left, "UNO" message is displayed on screen.
14. If any player's score reaches 500 then game is ended and that player is declared winner.
15. Otherwise game continue.

This algorithm control game flow till final winning score.

6.2 Queue Version (FIFO)

1. **Queue Initialization** We made a queue in which we set front 0 and rear -1 and count 0 and set size of queue and made all function using array which are push(), pop(), front() and empty().
2. **Enqueue ()**
   (a) We set the count equal to max size to control the overflow of data.
   (b) Update the rear when the card is add.
   (c) Insert the card at arr[rear].
   (d) Increment the count.
3. **Dequeue ()**
   (a) Display error when queue is empty.
   (b) Update the front when value is taken from queue.
   (c) And decrement the count.
4. **Front Element Access**

(a) If queue is empty return error

(b) Otherwise return value at front

5. **Check Empty**

   Check if queue is empty. If empty, return true other wise return false.

6. **Deck Initialization**

   We use the queue<Card>&deck to pass a queue to put all card in the queue of name deck.An array is used to save 4 color in it and for each color insert one zero and two card from 1 to 9 and also insert skip reverse and draw2 and insert 4 wild wild card and insert 4 wild +4 card.

7. **Shuffling Deck**

   We use the queue<Card>&deck to pass the deck in which I insert all the card then from this queue and insert all card in array and use another array to swap the position of the card this perform the shuffling process and then in the end insert all card in the deck queue.

8. **Drawing Card**

   We use the queue<Card>&deck to pass the deck in which all card contains and it took the card from front card of deck and remove that card from deck and return the drawn card

9. **Initial Card Dealing**

   We use the void dealInitialCards(Queue<Card>players[], Queue<Card>&deck), int numPlayers to pass the player , deck and player no so it will distribute 7 card to each player and will add the player card into each player queue.

10. **Player Hand Display**

    We use the void showPlayerHand(Queue<Card>&hand) to store the player hand size and for each card in hand it remove card from front and display the card and insert card in the order at back it insert the card.

11. **Choose Wild Color**

    A function of string chooseWildColor() exists which activates when player plays wild card and is called from player which color he want to implement on the next player so it impose that color card to the next player.

12. **Move Validation**

    We use the bool isValidMove(Card &played, Card &top, string forcedColor) which check that is player play the valid move or not if player played wild card or wild +4, move the valid, if forced color exist and player play that color card , move is valid. If color or value matches top card, move is valid, otherwise invalid.

13. **Winner Determination**

    It will declare void int totalplayer, winnercondition(Queue<Card>players[],

    Queue<Card>&deck) it set the winning condition if any player hand got empty he will win, and if deck empty then player with minimum card in hand win if card equal then draw.

14. **UNO Playgame** In this function, it calls all the above function one by one to run the game and set some rule in it which are not define in the isvalid move function so

    int totalplayer = 4;

    Queue<Card>deck, discard;

    Queue<Card>players[totalplayer];

    initializeDeck(deck);

    shuffleDeck(deck);

    dealInitialCards(players, totalplayer, deck);

    After these function it run the game in while and continue until game end. This play game function is called in the main function.

6.3 Linked List Version

1. **Circular Doubly Linked List Implementation**

    In this code we implement a circular list to implement the UNO game.Following are the different algorithms

    - Insertion: In this code we implement the two insertion functions, first for the insert at front and second for insert at end.
    - Deletion: This function deletes the required nodes on any position means deleting at position. This properly handles the list if there is only one node.
    - Traversal: This uses do-while loop to handle circular list.

2. **Deck and Discard Pile Management**

    The UNO game has decks of card, and we also discard cards from that deck so following is the algorithm:

    - Deck initialization: Total 108 cards in the deck. 4 colors and penalty cards like wild 4, simple wild, draw 2 etc.
    - Shuffling algorithm: First the cards copy to a vector then we do a random shuffle when the cards shuffle it copies back to the linked list.
    - Deck Reset: When the deck is empty, it reshuffles the deck but preserve the top card and it shows on the screen. It works on the rule of the game.

3. **Game State Management Algorithms**

    How the cards play what the conditions all are discussed in these algorithms:

    - Card Match: This algorithm checks the card to play matches to the number, color, action card to the discard pile card present on the top.
    - Turn Change: This function checks the player turn. If there is any action card play like skip or draw 2 etc. than it changes the turn according to that card.
    - Actions Card Effect: Reverse, Skip, Draw 2, Wild Draw 4 and Wild Color cards perform different tasks according to the rule of the game.

4. **Player Hand Management**

    - Card Search: This search the card in circular list to match the color or number.

7 Time Complexity Analysis

7.1 Stack Version

1. **Stack Class**
   - push(T value) : O(1), O(n) worst case
   - pop() : O(1)
   - peek() : O(1)
   - shuffle() : $O(n^2)$
   - getAtIndex(int index): O(n)
   - removeAtIndex(int index): O(n)
2. **Card Class**
   - getColor() : O(m) where m = length of card name
   - getValue() : O(m) where m = length of card name
   - isWild() : O(m)
   - isSpecial() : O(m)
   - canPlayOn(Card, string) : O(m)
3. **Hand Class**
   - addCard(string) : O(1)
   - getCardAt(int) : O(n)
   - removeCard(string) : O(n)
4. **Player Class**
   - drawCard(string) : O(1)
   - getHandSize() : O(1)
   - getScore, addScore: O(1)
   - playCard(int, Card, string) : $O(n^2)$ where n is hand size
5. **Deck Class**
   - createDeck() :$O(n^2)$ where n = 108
   - drawCard() :$O(n^2)$
   - reshuffleDiscard():$O(n^2)$
   - playCard(int, Card, string) : $O(n^2)$ where n is hand size
6. **Game Class**
   - canDrawFromDeck(): $O(1)$
   - currentPlayerHasPlayableCard(): $O(p \times h \times m)$ where p = totalPlayers, h = hand size, m = card name length

- handleCardEffect(): $O(p \times c)$ where p = totalPlayers, c = cards to draw (2 or 4)
- nextTurn(): $O(1)$
- setCurrentColor(): $O(k)$ where k = length of color string
- drawCardForCurrentPlayer(): $O(p + m)$ where p = totalPlayers, m = card name length
- canPlayDrawnCard(): $O(1)$

7.2 Linked List Version

1. **Linked List Functions**
   - InsertFront()/InsertEnd(): O(1)
   - deleteFront()/deleteEnd(): O (1)
   - deleteAT(position): O(n)
   - peekCard(): O(n)
   - Search: O(n)
   - Copy Constructor: O(n)
2. **Game Initialization**
   - Deck creation: O(1)
   - Player hand dealing: O(p) where p = number of players
3. **Gameplay Operations**
   - Card validation: O(1)
   - playCard(index): O(n)
   - drawCard(index): O(1)
   - getHandSize(): O(n)
   - Hand search: O(m) where m is the hand size.
   - Shuffle: O(n) where the n is size of the deck
   - resetIntoDeck(): O(n) where the n is size of discard pile
   - hasColor(color): O(n)
   - iscardValid(): O(1)
   - Wild Draw 4 challenge validation: O(n)
4. **Turn Processing**
   - Change turn: O(1)
   - Action card effects: O(1)
5. **Overall Complexity**
   - Space Complexity: O(n) where n is the total cards (108) plus players

- Best case: O(1)
- Worst case: O(n)
- Average case: O(k) , k denotes the average cards

7.3 Queue Version

1. **Initialize deck**: O(N) where N is the number of cards
2. **Shuffle deck**: O(n) where N is the number of cards
3. **Drawcard()**:O(N) where N is the number of cards
4. **Deal initial card()**: O(C * P), P is the number of players, C is the number of cards
5. **Show player hand()**: O(H), H is the number of card in player hand
6. **isValidMove()**: O(n), n is the length of card string.
7. **is_winningCondition()**: O(P), P is number of players.
8. **wild wild rule** O(K), K is hand size, draw penalty O(1)
9. **Turn complexity**: O(1)

Table 2: Time Complexity Analysis of UNO Game Implementations

Operation	**Stack**	**Linked List**	**Queue**
Access card by index	$O(n)$	$O(n)$	$O(n)$
Play card	$O(n^2)$	$O(n)$	$O(n)$
Display hand (GUI)	$O(n^2)$	O(n)	$O(n^2)$
Get hand size	$O(1)$	$O(n)$	$O(1)$
Search for card	$O(n^2)$	$O(n)$	$O(n)$
Shuffle deck	$O(n^2)$	$O(n)$	$O(n \log n)$

Note: n = number of cards

8 Comparison and Justification of Best Approach

UNO game requires frequent random access operations like "get card at index 5 or remove it". Since stacks and queues do not support random access, we need to empty to use extra space and loops for accessing elements at specific position. This increases time complexity.

Circular Doubly Linked List implementation shows a better time complexity than the Stack implementation and Queue implementation of the UNO card game since they insert or delete in O(1) and O(n) for traversal. Linked List has a clearer advantage in the display operation, which is most commonly performed operation in a GUI-based card game.The Linked List has O(n) per-turn complexity and Stack and Queue have$O(n^2)$ per-turn complexity, and therefore, when comparing them in terms of time complexity, the Linked List is the winner.

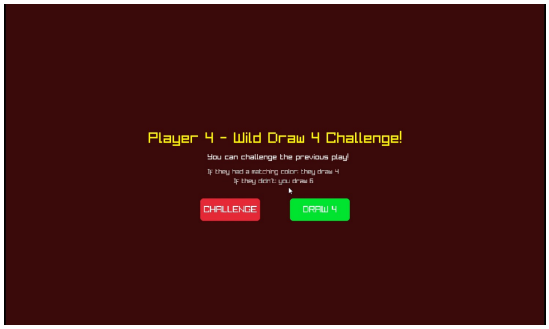
9 Individual Contributions

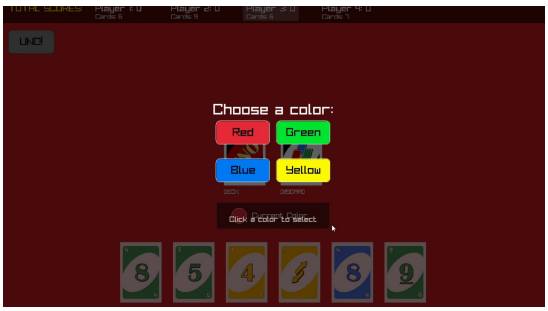
Overall, the work is managed as follows:

- Linked List version is developed by Hammad and Abdul Rehman.
- Queue version is developed by Momina Umer and Junaid.
- Stack version is developed by Arfa and Sadia.

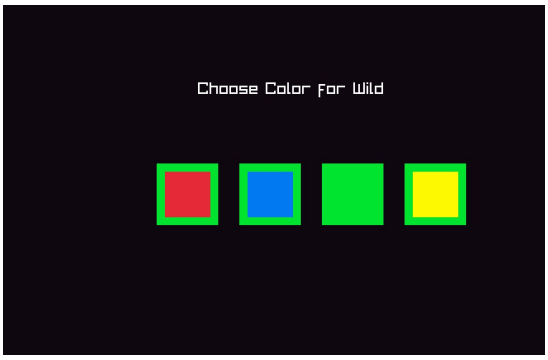
10 Game Screenshots (GUI and Gameplay)

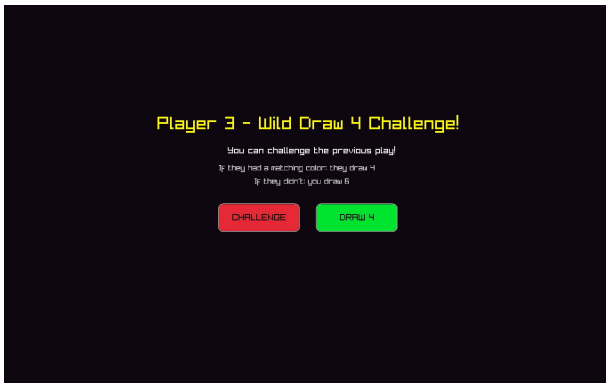
10.1 Linked List Version



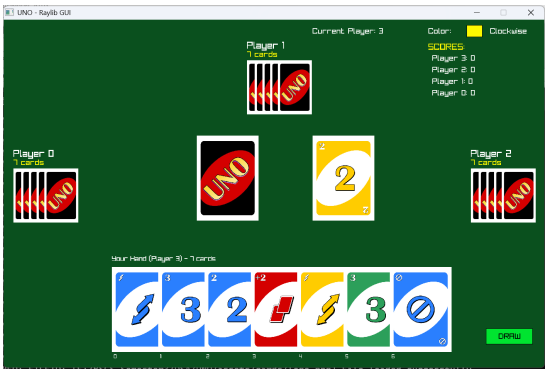


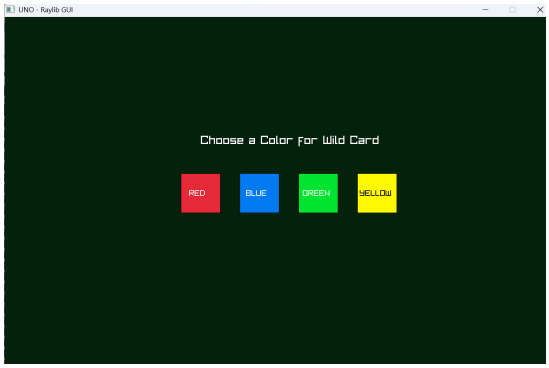
10.2 Queue Version

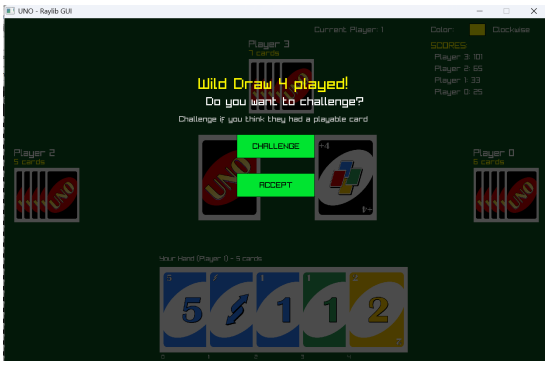




10.3 Stack Version







11 Conclusion

This project successfully demonstrates how different data structures can be applied to implement the same game — UNO Colors Rule — with varying trade-offs in performance, code complexity, and design elegance. The Stack-based version proved to be the most efficient and natural choice for a card game. The comparative study reinforces the importance of selecting the right data structure based on the dominant operations of the application.

Future enhancements may include AI opponents, online multiplayer, and mobile deployment.

12 References

- Mattel. (n.d.). UNO Colors Rule Instruction Booklet (DWV64-Eng.pdf).
- Raylib Documentation. `https://www.raylib.com/`
- `https://github.com/nliampisan/gameuno?tab=readme-ov-file#gameuno`.
- `https://rosettacode.org/wiki/Uno_(Card_Game)/Python`