Report: Implementation of Uno Game Using Double Chained Circular Lists

members of TP :

- Heythem Bouderbala.
- Abada Mahmoud.
- Rafa Houssam.

Table of Contents

1. Introduction

The Uno game is a popular card game known for its simple rules and exciting gameplay that was made by Merle Robbins in 1971. It has since been bought by a company named Mattel. It is similar to Crazy Eights.

How to play uno game ?

The cards are put into 4 different groups: Red cards, green cards, blue cards and yellow cards. There are also some other cards calleds". Skip, Reverse, Draw +2(This the way), Wild, and Wild +4 cards allow you to do something you cannot normally do, such as pick up two more card

Each player starts off with 7 cards, randomly assigned. A card from the main deck is then placed in the center for everyone to see. The first player must put down a card that is either the same color or number, or a willayer puts down a regular card, the next person has to put down another card of the same color or number, and it keeps on going in order. When a skip card. A reverse card reverses the order of the game. +2 cards force the next player to draw 2 cards, but +4 cards force the next player to draw 4 cards and the current player gets to pick a new color of their choice.

In this report, we present the design and implementation details of an Uno game using the C programming language. The system utilizes double chained circular lists for efficient management of player hands and the discard stack. Additionally, we provide a theoretical analysis of the algorithm complexities and highlight the most important functions of the system. Finally, instructions for running the game are provided for users' convenience.

2. System Design

The Uno game system is designed to support multiple players, each having a hand of cards. The core components of the system include:

Player Manageme

- Linked List Data Structure: Players are managed using a circular doubly linked list data structure. This data structure allows for movement between players in the game.
- Player Node: Each player is represented by a node in the linked list. This node contains essential information about the player, such as their name, unique player ID, and a doubly linked list representing their hand of cards.
- Each player's hand is stored as a doubly linked list. This data structure allows for efficient traversal and manipulation of the cards in the player's hand.

Card Management:

- Linked List Data Structure: Cards are managed using a linked list data structure similar to player management.
- Deck Generation and Shuffling: At the beginning of the game, the deck of cards is generated with all the Uno cards (numbered cards, action cards, and wild cards). The deck is then shuffled to ensure randomness.
- Drawing Cards: During gameplay, cards are drawn from the deck and added to players' hands. Drawing a card involves removing a card from the top of the deck and adding it to the specified player's hand.
- Discard Stack: The discard stack is represented using a linked list. When players play cards, they are added to the top of the discard

stack. This stack serves as the source for subsequent plays and determines the allowable plays based on the top card.

Game Logic:

- Action Handling: The game logic encompasses various actions such as playing cards, drawing cards, reversing play direction, skipping players, and declaring a winner.
- Functions: These actions are implemented using various functions that manipulate the player's hand, the discard stack, and the overall game state. For example, playing a card involves removing it from the player's hand and adding it to the top of the discard stack, while drawing a card involves adding a card from the deck to the player's hand.
- Rule Enforcement: The game logic enforces Uno rules, such as matching colors or numbers, playing action cards (e.g., skip, reverse, draw two), and handling wild cards (e.g., allowing the player to change the color or draw four cards).

the design of the data struct :

- the first line in the table represent a double-chained circular list for players.
- the colones in the table represent a linked list data structure for cards .

player 1	player 2	player 3	player 4
card 1	card 1	card 1	card 1
card 2	card 2	card 2	card 2
……...	……...	……...	……...

player node :

name of player

card node :

card number card color card property

player ID
hand of cards

3- Double Chained Circular Lists and abstract machines

Double-Chained: In a double-chained linked list, each node has two pointers: one pointing to the next node in the list and another pointing to the previous node. This structure allows for bidirectional traversal of the list.

Circular: A circular linked list is one where the last node in the list is connected back to the first node, forming a loop. This means that traversal can continue indefinitely, and operations like insertion and deletion are more straightforward.

abstract machine of player :

`next_player(player *ptr):`

- This function returns a pointer to the next player in the circular list.

`prev_player(player *ptr):`

- This function returns a pointer to the previous player in the circular list.

`allocate_player(player **head_player):`

- This function allocates memory for a new player node.

`ass_name(player **pointer, char name[20]:`

- This function copies the input string to the `name` field of the player structure.

`link_players(player **ptr1, player **ptr2):`

- This function sets the `next` pointer of `ptr1` to `ptr2` and the `prev` pointer of `ptr2` to `ptr1`, effectively linking the two players in a doubly-chained manner.

`display_players(player *head_player):`

- For each player node encountered, it prints the player's name.

`char* id_player(player*plr):`

-This function return the id of a player(human or robot).

`char*  name_player(player*plr):`

-This function return the name of the player.

`card_player(player*ptr):`

-this function return a pointer to the first card of the player.

`void  free_players(player** plr):`

-this function free the node of player.

abstract machine of card :

`next_card(card *ptr):`

- This function returns a pointer to the next card node in the linked list.

`allocate_card(card **head_card):`

- This function allocates memory for a new card node.

`assign_properties(card **pointer, int number, const char *color, const char *property):`

- This function takes a double pointer to a card (`pointer`) and the card's number, color, and property as input and assigns the provided values to the corresponding fields of the card structure.

`link_cards(card **ptr1, card **ptr2):`

- This function sets the `next` pointer of `ptr1` to `ptr2`, effectively linking the two cards in the linked list.

`free_card(card **ptr):`

-It takes a double pointer to a card (`ptr`) as input and releases the memory occupied by the card node.

`char* property_card(card * current_card:`

- return the property of the card.

`int number_card(card*current_card):`

-This function returns the number of a card.

`char* color_card(card * current_card):`

-This function returns the color of a card.

`remove_card(card **head_card, int *n):`

- This function takes a double pointer to the head of the card list (`head_card`) and the position (`n`) of the card to be removed as input.

- If the specified position is valid, it removes the card node at that position by adjusting the `next` pointers accordingly.

4.Implementation Detail

The implementation of the Uno game system involves the following key steps:

Initialization:

- Players Initialization: At the start of the game, players are initialized by assigning each player a unique ID and prompting them to input their names.
- Deck Generation and Shuffling: The deck of Uno cards is generated, including numbered cards, action cards (skip, reverse, draw two), and wild cards. The deck is then shuffled to ensure randomness.
- Discard Stack Initialization: The discard stack is initialized by drawing the top card from the shuffled deck. This card becomes the starting point for subsequent plays.

Game Loop:

- Displaying the Top Card: At the beginning of each turn within the game loop, the top card of the discard stack is displayed to all players. This card serves as the reference for valid plays.

- Player Turn: The game prompts the current player to either play a card from their hand or draw a card from the deck if they cannot play.
- Card Validation and Game State Update: When a player attempts to play a card, the system validates whether the selected card is a valid move according to Uno rules (matching color, number, or action). If valid, the card is played, and the game state is updated accordingly (e.g., updating the discard stack). If not valid, the player draws a card or moves to the next player.
- Moving to the Next Player: After a player completes their turn, the game moves to the next player in the sequence and repeats the process until a winning condition is met.

Winning Condition:

- The game loop continues until a player successfully empties their hand by playing all their cards.
- When a player has no cards left in their hand, they are declared the winner of the game.
- Once a winner is determined, the game concludes.

5. Complexity Analysis

The complexity of the Uno game system can be analyzed as follows:

- Initialization: Generating and shuffling the deck has a time complexity of O(n), where n is the number of cards in the deck.
- Game Loop: The complexity of the game loop depends on the number of players and the average number of cards in each player's hand. In the worst case, where each player has a maximum number of cards, the time complexity of each iteration is O(m), where m is the number of players.
- Winning Condition: Determining the winner requires iterating through all players to check if their hand is empty, resulting in a time complexity of O(m), where m is the number of players.

6. Most Important Functions

The most important functions in the Uno game system include:

`play_uno_game` : this function manages the entire gameplay process of Uno. It begins by prompting the user for the number of players and initializes the deck and discard stack. After shuffling the deck, it initializes the players and displays their information. The game starts by drawing the first card, setting the initial direction, and entering the main game loop. In each iteration, it displays the current state, prompts the current player to play a card, and checks for a winner. Once a winner is declared, it displays a celebratory message and frees memory.

`play_card:` This function in the Uno game code enables a player to play a card during their turn. It first checks if the player has any playable cards, displays their hand, and prompts them to choose a card to play. Depending on the selected card's properties (such as Reverse, Pass, +2, +4, or Color Choice), it executes the corresponding action, including reversing the play direction, skipping turns, or allowing color selection. If the chosen card is a regular card, it verifies if it matches the color or number of the top card on the discard stack, playing it if valid, or informing the player of an invalid choice. After any action, the function progresses the game to the next player according to the play direction. Overall, it manages card plays, special effects, and turn transitions in the Uno game `_initialize_players:` This function sets up the players for the Uno game. It takes inputs such as the number of players, the deck of cards, and the discard stack. For each player, it prompts for a name, deals cards from the deck, and links the players together in a linked list. If memory allocation fails, it displays an error message. It iterates through the players, assigning names, dealing cards, and linking them together. Finally, it ensures the list is circular by linking the last player to the first player. This function initializes the players' data and establishes the game's player structure.

7. Instructions for Running the Game

To run the Uno game system:

- Compile the provided C code using a C compiler (e.g., GCC).
- Execute the compiled binary file to start the game.
- Follow the on-screen instructions to navigate the menu and play the game.