Angel Grajeda-Cervantes

Professor Kanemoto

CPSC-39-10106

13 May 2025

Final Programming Project Report

1. What your game or app does and how it is useful or entertaining.

1. My project is a simple Blackjack game that you play in the terminal. The game lets one player go against the computer (the dealer). It keeps track of player stats like wins, losses, and chips. It also stores players in a waiting line and shows past hands. This game is fun to play and also helps show how Java can be used to make games with logic and structure.

2. Includes at least 3 algorithms as steps or a flowchart, and a snapshot of the algorithms code.

2. The first algorithm would be player profile management which uses a hashmap data structure. The steps are,

1. Ask the user for their name.
2. Look up their name in the profile list (HashMap).
3. If the name is not found:
    - Create a new profile.
    - Add it to the profile list.
    - Show a message: "New profile created."

4. If the name is found:
    - Load the existing profile.
    - Show a message: "Welcome back!"

```java
// (Project line 26-62)Angel Grajeda-Cervantes: Player Profile Management (HashMap)
public static PlayerProfile getOrCreateProfile(String name) {
    PlayerProfile profile = profiles.get(name);
    if (profile == null) {
        profile = new PlayerProfile(name);
        profiles.put(name, profile);
        System.out.println("New profile created for " + name);
    } else {
        System.out.println("Welcome back, " + name + "!");
    }
    return profile;
}
```

The second algorithm would be waiting list management which uses a queue data structure

The steps are,

1. Ask player to enter their name.
2. Add their name to a Queue<String> waitingPlayers.
3. When it's time to play, remove and return the next player from the queue.

```java
//Angel Grajeda-Cervantes: Waiting List Management (Queue)
public static void addToWaitingList(java.util.Scanner scanner) {
    System.out.print(s:"Enter your name to join the waiting list: ");
    String name = scanner.nextLine();
    waitingPlayers.add(name);
    System.out.println(name + " added to the waiting list.");
}

public static String getNextPlayer() {
    return waitingPlayers.poll();
}
```

The third algorithm would be Player and Dealer Hand History which uses a linked list data structure. The steps,

1. When a card is dealt, create a string representation of it.
2. Add it to a LinkedList<String> for either player or dealer.
3. At the end of the round, print both hand histories.
4. Clear the lists for the next round.

```
// Angel Grajeda-Cervantes: Hand History Tracking (LinkedList)
public static void addCardToHand(LinkedList<String> hand, String card) {
    hand.add(card);
}


public static void showHandHistory() {
    System.out.println("Player's hand: " + playerHandHistory);
    System.out.println("Dealer's hand: " + dealerHandHistory);
    playerHandHistory.clear();
    dealerHandHistory.clear();
}
```

3. Explain in a paragraph or 2 the algorithms that you created and how they are used in the game or app. How you created them is important, and if you used ChatGPT here you can explain how you used it - ChatGPT should not be able to write your algorithms in their totality.

3. One algorithm I made was to check if a player already has a profile using a hashmap. If their name is not in the map, it creates a new profile. If it is, it loads their old stats. This helps the game remember each player. I wrote the if-statement to check the name and add it if needed.

The second algorithm I made adds players to a waiting list using a Queue. When someone joins, their name goes in the list. Then, the game picks the next player by removing the first name. This helped organize turns when multiple people wanted to play.

The third algorithm I made keeps track of each round's cards using LinkedList. Each time a card is dealt, it is saved. At the end of the round, it prints the cards and clears the list. This made it easy to see what happened during the game.

4. Discusses the Big O time of these algorithms.

4. HashMap (Profile Management):

The HashMap operations like get() and put() are usually O(1) in time. This makes profile lookup and creation very fast, even if it had lots of players.

Queue (Waiting List):

Adding a player to the queue and getting the next player both take O(1) time using add() and poll().

LinkedList (Hand History):

Adding cards to the hand with add() is O(1), and printing/clearing the hand history is O(n) where n is the number of cards in the round.

5. An explanation of the data structures that you used, why you chose them, and how they were used.

5. I used a hashmap to store player profiles because it allows fast access to player data by using their names as keys.

I used a Queue to manage players waiting to play. A queue makes sense because it works in first-in, first-out order, just like a real line.

I used LinkedLists to keep track of each card dealt to the player and dealer. LinkedLists are good for adding elements quickly and don’t need to shift elements around like arrays.

6. Explains a step in the design or development process where you encountered an opportunity and how you used this.

6. During the design, I saw an opportunity to make the game more personal by keeping track of players' wins and losses. Originally, I just had one player playing with no history. I added the PlayerProfile class and used the HashMap to store profiles. This made the game more fun, and players could see their progress over time.

7. Explains a step in the design or development process where you encountered an error and how you resolved this.

7. A problem I had was when the hand history was not clearing up after every game. The old cards kept showing up in the new round. I realized I needed to call .clear() on both LinkedLists after displaying the hands. After fixing that, the hand history worked correctly each round. I used print statements to debug and test if the lists were being emptied.

8. Explains what you would change or add in the next version of your game or app.

8. In the next version I would add multiplayer support so that two human players can play against each other, not just one player versus the dealer which is a computer. I would also want to build a simple graphical interface instead of just using the terminal, that would make the game more fun and user-friendly.