Name: Jason Ash
CIS-7 Discrete Structures
November 20, 2025
Professor: Dr. Nguyen

Project Documentation

1. Team member(s): Jason Ash (solo team)
Team name: I don't know Jack

2. Project Information and Details

What problems are you solving with this project?

I chose Case 4: Casino Blackjack because it sounds interesting and challenging, and I believe that I would be able to accomplish writing a program on it the easiest out of the four options. I solved how to shuffle a deck of 52 unique, random cards (albeit my solution is inefficient). My implementation somewhat resembles what someone would do manually if given 52 random cards to shuffle. However, much more detail and logic was required to shuffle the deck properly than I planned for in my flowchart.

I also solved how to deal the player and dealer two initial cards, handle the player choosing to hit, and handle the dealer playing according to casino rules (draw up to and including 16, and stand on 17 or greater). I solved how to score the outcome given multiple scenarios (the player gets a Blackjack, the dealer does, they both do, the player bust, the dealer bust, they both do, or who has the higher hand if none of the previous scenarios apply).

Although my flowchart glossed over the details of displaying the card name corresponding to a card number, I was able to accomplish it with if/else if statements and modular division (the later of which I initially did not plan on doing, but it worked out great). The modular division was able to determine the suit of a card by dividing its card number and assigning a suit based on the remainder (0 for spades, 1 for hearts, 2 for diamonds, and 3 for clubs). That simplified by code vs. the alternative of a huge switch or if/else statement consisting of 52 different options.

The tally card function (accidentally misspelled talley) added up the value of each hand and called a helper function called get card value which gets the value of one card and handles aces depending on whether the hand value is 10 or less (aces count as 11) or more than 10 (aces count as a one).

I combined the functions in my flowchart that calculate the probability of a 21 or going bust on the next hit into one function since they are nearly the same, and I passed the variable storing the probability of going bust by reference since functions can only return one value. This allowed me to manipulate another variable's value within that calculation function, and to still retrieve the result in main.

What solutions are you implementing with this project?

In addition to the above solutions, one issue that needed to be resolved early in the planning process was how many elements should be in the arrays representing the player's and dealer's hands? By counting, the worse case scenario for a hand totaling 21 is a hand of four aces (assuming each is counted as a one), four deuces (another name for 2's), and three 3's. The sum of such a hand would be $1 + 1 + 1 + 1 + 2 + 2 + 2 + 2+ 3 + 3 + 3 = 21$. This is 11 cards in total. Any additional cards dealt would result in a bust. Therefore, the array size for the player's and dealer's hand needs to be 11 elements. While such a worse case scenario is highly unlikely, it is not impossible, and therefore the program needs to accommodate for it.

Provide an explanation of calculations and algorithm implementation.

As the hand value is being totaled, if the hand value is less than on equal to 10 and the card being checked is an ace (card number 1 to 4; see appendix A), then the ace is counted as an 11. Otherwise, it will be counted as a one. All of the elements of the deck array, user hand array, and dealer array are initialized to zero which indicates they do not contain a card. This makes the linear search slightly faster because when it reaches a zero in the user hand or dealer hand, then it knows the search is complete.

The program accurately totals the player and dealer's hands as explained above. It also continually updates the probability of the player or dealer getting a 21 or going bust on the next hit. Note: for hands with a value of 11 or less, the probability of going bust is zero, and the program reflects this. It also gives the probability for obtaining a 21 if the hand is 11. The probabilities are not displayed if a pat Blackjack is dealt because that is a winning scenario (for the player or dealer depending on who has the Blackjack or if both have it), and displaying the probabilities would not make sense. A change that I made from my flowchart that seems to make sense is separating the scoring from the player's and dealer's turn.

The deck shuffling algorithm that populates the deck array with a unique, random card number in each element is inefficient but it I was able to write it based on my own understanding since I'm still somewhat a beginner with programming. An outer for loop accomplishes going through each element of the deck array from first to last to make sure all of them contain a random, unique card. A random card is generated, and this algorithm assumes that it is not already in the deck, then an inner for loop checks to see if that card is already in the deck array. If so, then a new random card is chosen, and the deck array is checked again. If the deck does not have that card, then the deck element at the current position is assigned the value of that card. This algorithm is at a minimum $O(n^2)$. At the 26^{th} element in the middle of the deck, the worse case scenario is that the program might have to on generate up to 27 random card numbers and check each prior element of the array to see if its already present before one can be inserted. It gets worse for the last element because 51 random card numbers will be in the other elements, to the worse case scenario is that 52 cards will have to be randomly generated (perhaps with some duplicates) and each one of those checked against 51 elements to see if the card number is already present. Even then, this was one of the more challenging aspects of the game logic to work out, and it took about 1.5 hours for me to get it correct. Other challenges were scoring and calculating the percent probabilities. Thank goodness that modern computers can perform calculations quickly, and I didn't even notice a performance penalty with my inefficient shuffling algorithm.

What is your program's objectives?

The objective of this program will be to implement a Blackjack game program that continually updates the probability of the user and dealer obtaining a 21 or going bust after the initial two cards are dealt and after each hit. It will also apply Discrete Structures course concepts and theorems (see below).

The following are additional objectives that are required to be met by the program:

- One player in addition to the dealer.
- Implementing a single, standard 52 card deck.
- The player will be allowed to hit until 21 is reached or they go bust. (Going bust is having a hand with a value of 22 or greater).
- Aces count as both one and 11.
- Follows casino rules in that the dealer must hit on 16 or less and stand on 17 or more.
- If player bust, that's an automatic loss for the player even if the dealer also busts.
- Implementing a push in the event of a tie with both the player and dealer having a hand with the same value that is 21 or less.

- Implementing the two winning scenarios that the player has a Blackjack or is closer to 21 than the dealer.
- Implementing the scenario that if the dealer has a Blackjack and the player does not, then the player automatically loses that game.
- The program does not display the dealer's hole card until the dealer's turn comes up.

The following is a wish list of features to include if time and my limited expertise permits:

- More than one player.
- Implementing bet amounts and keeping track of the player's current amount of money. (This would be incidental to the game logic because since that is correct, then working out win, lose, and tie amounts would be fairly straightforward.).
- Implementing a value for pretend money for the bets and giving the player double their bet upon winning, returning their money in a push, 1.5x the bet on a pat Blackjack, and 2.5x the player's bet if both the player and dealer have pat Blackjacks.
- Allowing the player to double down in which the player can double their bet and is allowed to be hit only one additional card beyond the initial two cards dealt (especially useful if the player has an 11 in their hand and also useful with 10s and 9s according to certain probability tables and player strategies).
- Allowing the player to split pairs and essentially play multiple hands (if the rules permit).
- Implementing multiple decks (might make the program very challenging to write).
- Implementing multiple players (this might also increase the complexity of the program).
- Displaying the probability of the player winning against the dealer given what is known at the moment since the dealer's hole card is hidden until the player's turn ends upon which the dealer's turn begins.

Explain how your program is interacting with the user.

This program will interact with the user through the command prompt in text mode without any graphical user interface (GUI). This is because the discrete structures concepts and accomplishing the logic of Blackjack are what is important and not an attractive user interface. (Also, GUI concepts as they relate to C++ and program design have not been learned yet). The command prompt will display in words what cards the player and dealer have, and the probability of reaching 21 on the next hit or going bust. The program will ask the user to enter single letter commands, such as, 'h' or 'H' for hit and 'S' or 's' for stand. The program also asks the user to enter 'Y' or 'y' to play again.

Explain the purpose of the program.

The purpose of this program is implement a Blackjack game that applies discrete structures concepts and theorems, demonstrates the logic of creating a Blackjack game according to its rules, implements algorithms to that effect, and continually updates the probability of the player and dealer winning and losing before and following each hit.

How are discrete structures concepts and theorems implemented in this C++ program?

Truth tables according to whether the player has a hand of 21 (win) or greater (lose) or a hand closer to 21 than the dealer (win) will be implemented. Also, the truth tables will have to consider a push (tie) between the dealer and player in which both have the same hand value without going over 21.

The set of cards is a standard 52 card deck without Jokers. The set of winning hands are all of those that add up to 21 or are closer to 21 than the dealer's hand value. The set of losing hands are those that are less than 21 and lower than the dealer's hand value (difficult to determine until the dealer's turn is resolved), and the set of hands exceeding the value of 21, which is a bust (an automatic

loss). The sets that result in the player winning, losing, a tie are all subsets of the 52 card deck. In fact, any hand is a subset of the 52 card deck because it consist of only cards found in the 52 card deck.

The probability of busting when the initial two cards are dealt to the player and dealer (before anyone hits) is zero because the highest value possible at that point is a 21 (an ace and a face card or 10). There are many combinations that would result in a 21 (or any other hand combination), but few that would result in a pat Blackjack (which must be a ten or a face card and an ace as the two initial cards dealt). The program must determine the player and dealer's hand values, and count the values of the cards remaining in the deck and determine the probability of obtaining a 21 or going bust. The probability of obtaining 21 on the next hit is:
The number of cards left in the deck that result in a 21 when added to the player's hand value divided by the total number of cards remaining in the deck.
The probability of going bust on the next hit is:
All of the cards left in the deck that result in a hand value of 22 or greater when added to the player's hand value divided by the number of cards remaining in the deck.

This is a very dynamic probability that is constantly changing because the player and dealer continually get dealt random cards from the remaining cards in the deck (the player according to whichever strategy they employ, and the dealer according to the casino rules). As more cards are dealt, and the player has not yet reached 21 or busted, the higher the probability will be of busting. The dealer doesn't have a choice and must hit on all values less than or equal to 16 and stand on 17 or more. So, the dealer's probabilities are a little bit more predetermined since that is the only strategy the dealer is allowed to employ. The dealer will obviously have a zero probability of busting on any hand with a value of 11 or less, and the probability of busting will go up (probably much more than linearly) up to a hand totaling 16. After this, the probability of busting if the dealer has not yet busted is zero because the dealer must stop drawing cards for themselves at 17 or greater.

In my implementation, the deck will be searched linearly during each turn to update the probability because the array that represents the deck will have its "cards" in a random, shuffled order. Therefore, in an unsorted list, I believe linear search is the best that can be accomplished. Since the cards will be dealt from the beginning to the last element (it will never reach that far and with one player and a dealer, the most cards that will ever be dealt is 16 [11 for the worse-case hand adding up to 21, and another five for the second worse case hand that does not include the cards in the 11 card hand, i.e. 3 + 3 + 4 + 4 + 4 = 21]), at first in alternating order (i.e. the player gets a card and the dealer gets a card in turn until they both have been dealt two initial cards each) and then in turn order (the player goes first followed by the resolution of the dealer's turn), the algorithm will search from marker representing the card to be dealt next to the end of the array to update the probabilities each time. Then, after every game of Blackjack, if the player wishes to continue playing, the deck will be reshuffled and it will have a different random distribution of cards than before. The deck will not need to be sorted in any way in addition to shuffling it (which might not count as a sort per se since the order of card numbers must be random). The algorithm that goes through the hands to determine their value will simply add up the value separately of the cards in the player's hand and the dealer's hand and output and store their value for probability calculations. In this respect, there is not any searching through or sorting the hands per se.

Mathematical induction is greatly implied that if a hand adds up to a certain value, certain other cards will result in a 21, and other cards could result in busting. For example, if the player has a hand of 20 and decides to hit, then any card other than an ace will result in the player busting. So, if a two results in a bust in this scenario, then it is true that a three results in a bust and so on.

Graphs and relations won't be implemented in this program. The only relations is that between the cards and their combinations in the player's and dealer's hands according to the rules of Blackjack.

I was able to do modulus division on determining the suit of each card in the displayCard() function since the order of the suits repeats itself every four cards. For example, a modulus 4 resulting

in a remainder of 0 is spades, resulting in 1 is hearts, resulting in 2 is diamonds, and resulting in 3 is clubs. This is much more efficient that the giant switch statement or if/else structure that I originally envisioned and cuts down on the amount of code needed considerably. It is interesting how much can change from the planning stages once you "dive in", work out the logic, and start coding based on the flowchart. Nevertheless, having a plan including a flowchart or pseudocode is of paramount importance because programming would take multiple times longer and the result would be much more error-prone without a plan. Some people might not think Blackjack is complicated, but once you start working out the logic of even a one deck and one player game, it is actually complex and takes 100s of lines of code.

What are the limitations of this program?

The limitations of this program so far are extensive in that:

- Arrays are used for the hands instead of a more dynamic container such as vectors.
- Shuffling the deck is at least an $O(n^2)$ process for every card inserted into the deck array because the program must check each time if the card is already in the deck, and if so, choose another random number between 1 and 52, check again, and repeat the process.
- The program must search through the entire array of remaining cards to determine the probability of obtaining 21 or going bust at each state after the initial two cards are dealt. I don't really see a way around this (unless the deck can be implemented differently) because to play fairly, the deck must have a unique (for all practical purposes since there is 52! ways of shuffling a deck which is an astronomically large number) random order of cards at the start of each game.
- Originally, I planned on using an extensive switch statement along with the number of a card. For example, if card number 5 represents a two of clubs (since the first four cards will be aces), then the switch statement will both determine that card has a value of 2 and display the string "two of clubs" to the user. This is a simplistic implementation, but probably an inefficient one. Then, I was able to do modulus division with if/else if statements to cut down on the lines of code and make it more efficient regarding displaying the card name corresponding to its card number.

However, it is important as a student that I overcome the challenges of this program largely, if not entirely, on my own as to learn how to create a solution and not to rely on other's code.

Provide recommendations on improving the limitations of the program.

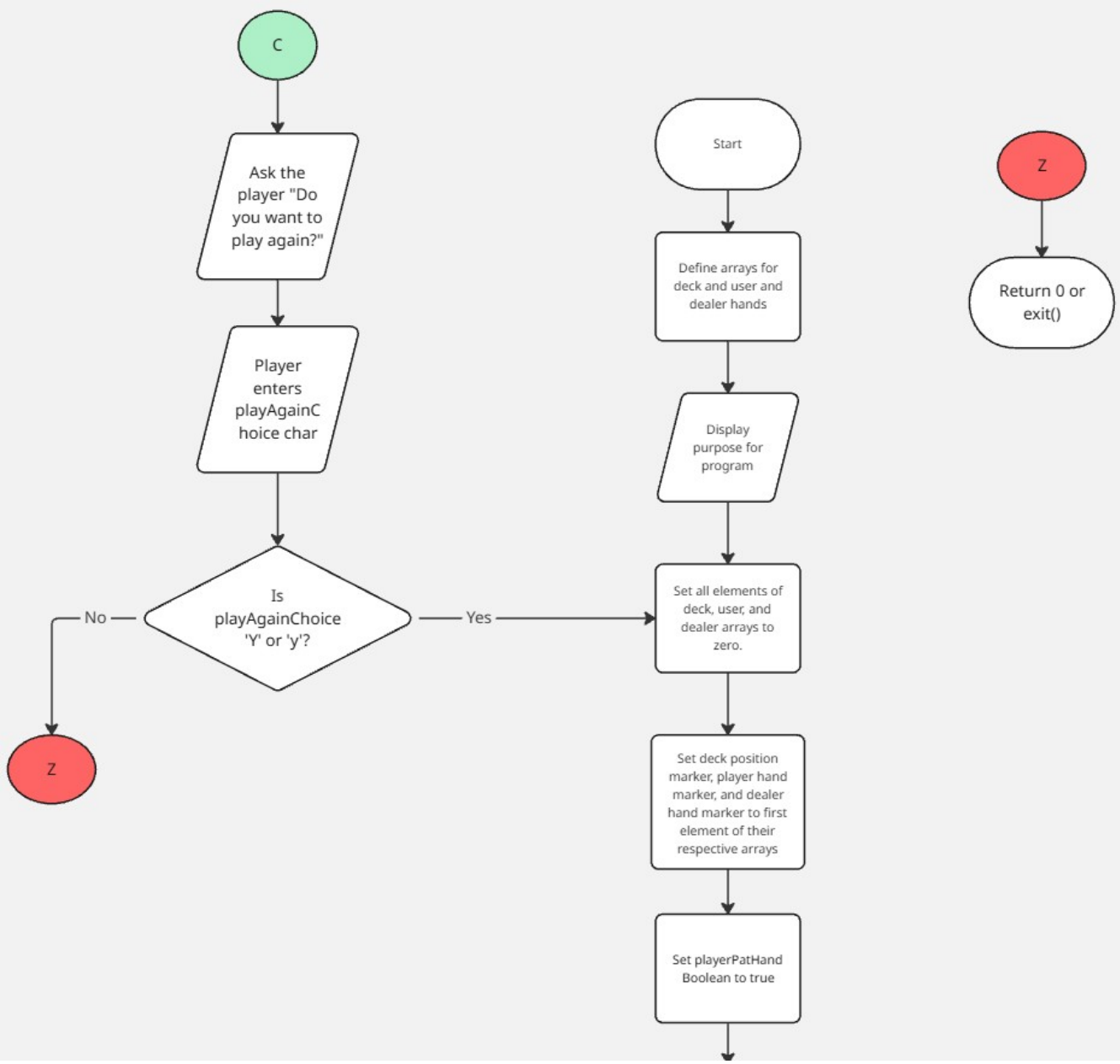
The number one inefficiency is the shuffling algorithm that I came up with. With my current level of understanding, I am not sure how to improve this. As I learn more about programming and mathematics (especially discrete mathematics), then I might someday be able to create a better solution. This program is an original effort, and learning and overcoming the challenges is what is important. I wrote this program entirely of my own understanding (only one source was cited in the source code from an auxiliary textbook used in CIS-5 for how to chose a random number from the first to last number in a range; "Exploring C++ The Adventure Begins" by Jason James).

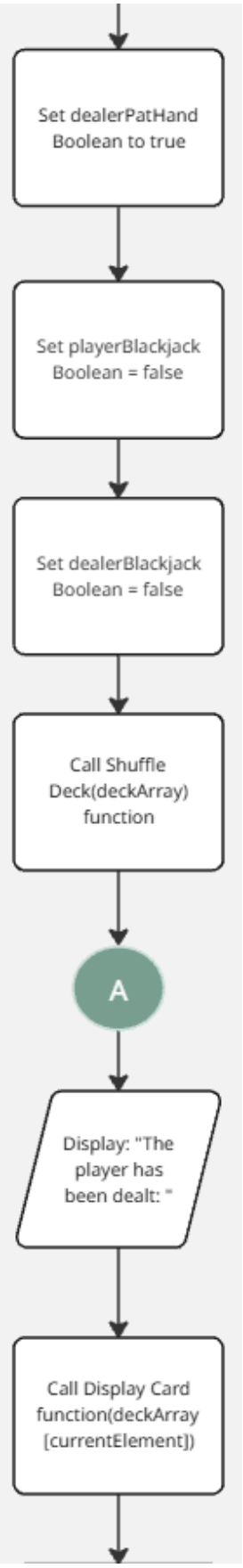
Pretend money with bet amounts and the player winning or losing their bet according to Blackjack rules would be the "lowest hanging fruit" that could probably be implemented with a little more time and effort devoted to this project.

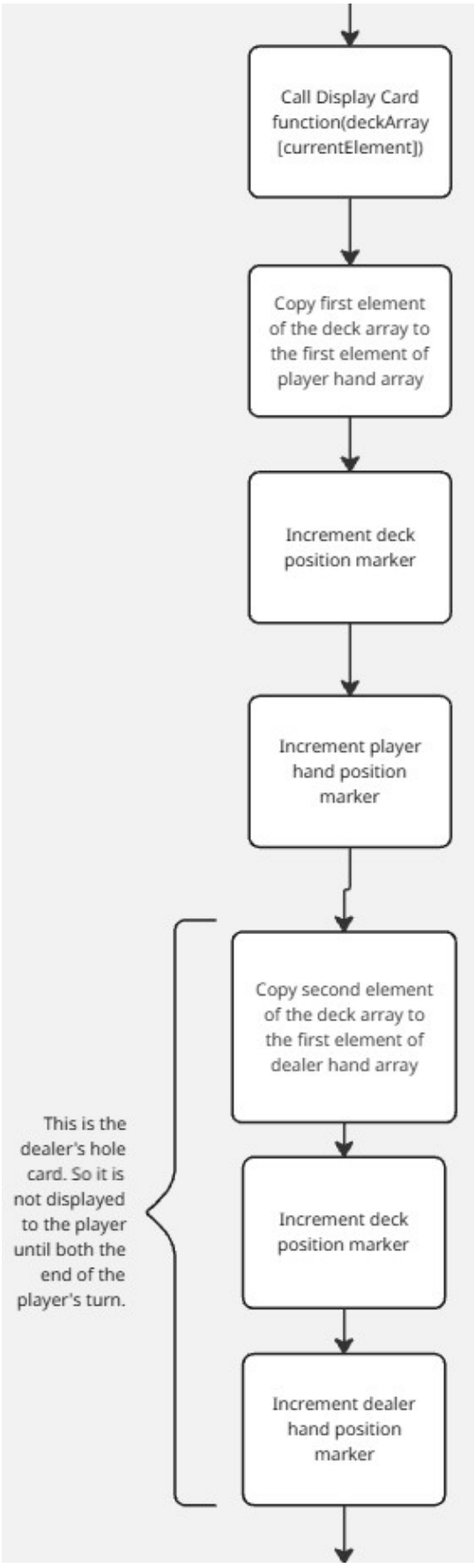
I am also not sure how to improve adding up the value of the player's or dealer's hand, displaying the card name (unless a different approach was used to create, store, access, and interact with the virtual deck than assigning each card a unique integer), and calculating the probability of obtaining 21 or going bust on the next hit. As for the last item (updating probabilities), I am not sure if it can be improved since the deck will always have a random order of cards, and the player and dealer are dealt random cards, so linearly searching through the remaining deck and comparing it to the hand

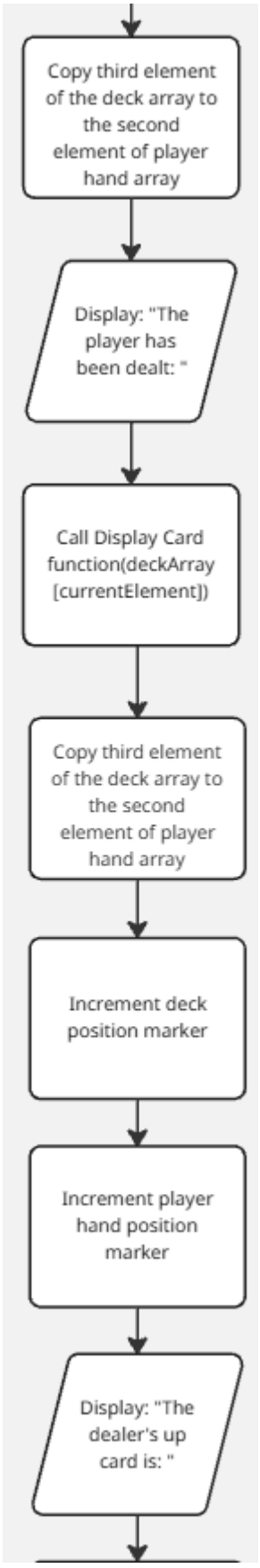
in play would be the only way that I can think of to calculate these probabilities “on-the-fly”. Otherwise, they would have to be hard-coded which would make the code more complex and longer to account for huge probability tables. Hard-coding probabilities was also not allowed for this project, so the only way was to write a function that was able to calculate them in a dynamic, more general way depending only on what was happening during the course of that particular game.

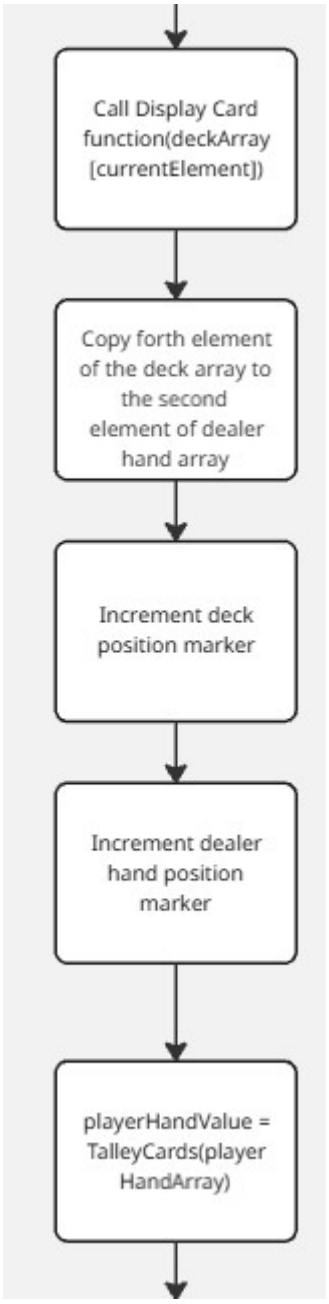
Flowchart (I created it from scratch using miro.com. I did a little bit of post-production editing in Microsoft Paint)

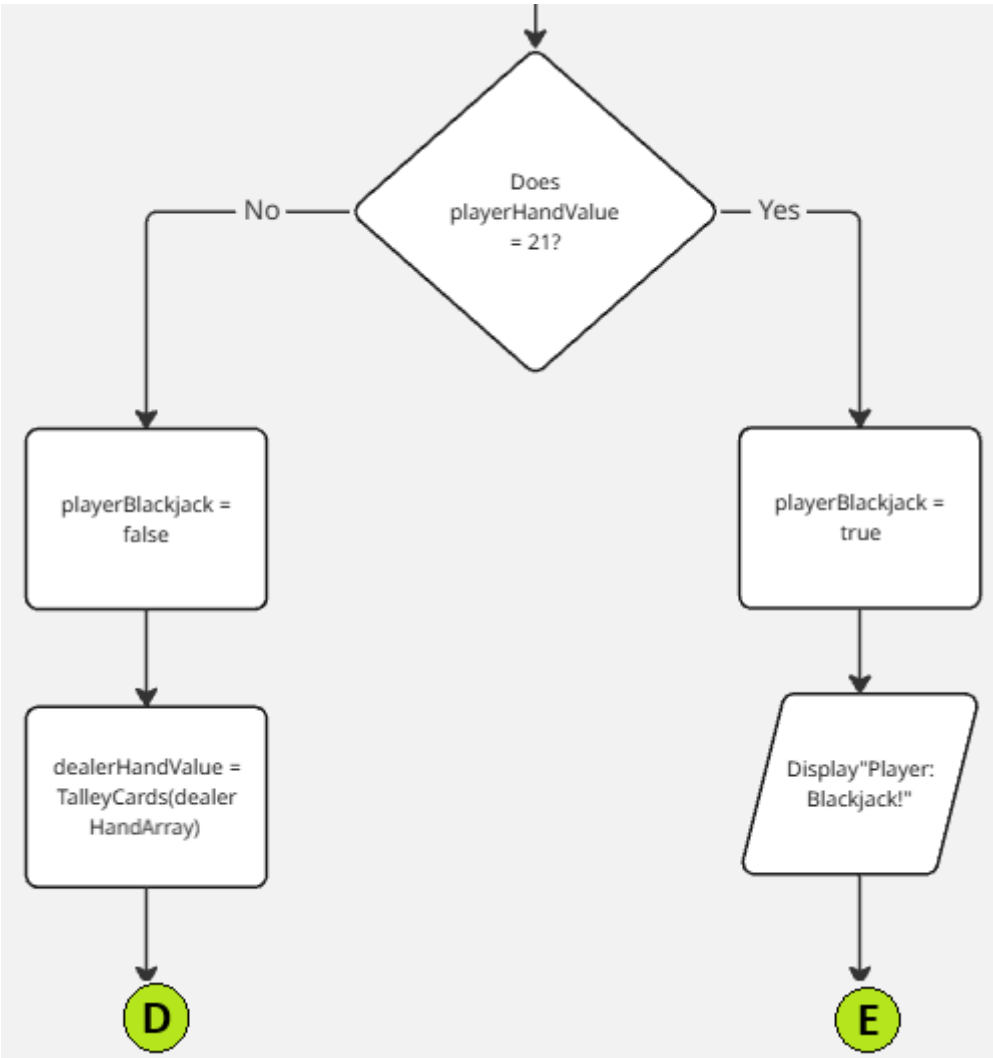


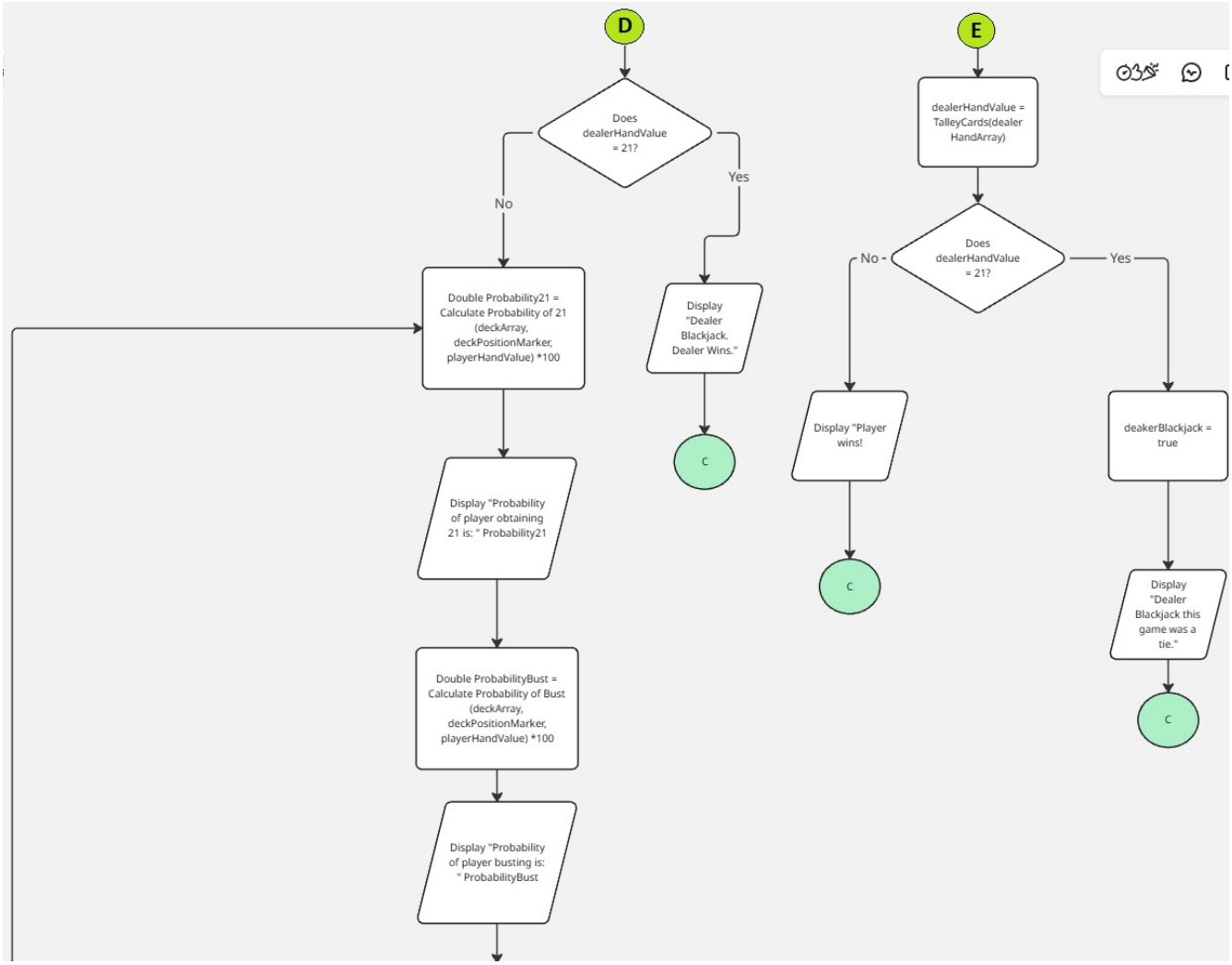

Set dealerPatHand Boolean to true
Set playerBlackjack Boolean = false
Set dealerBlackjack Boolean = false
Call Shuffle Deck(deckArray) function
A
Display: "The player has been dealt: "
Call Display Card function(deckArray [currentElement])

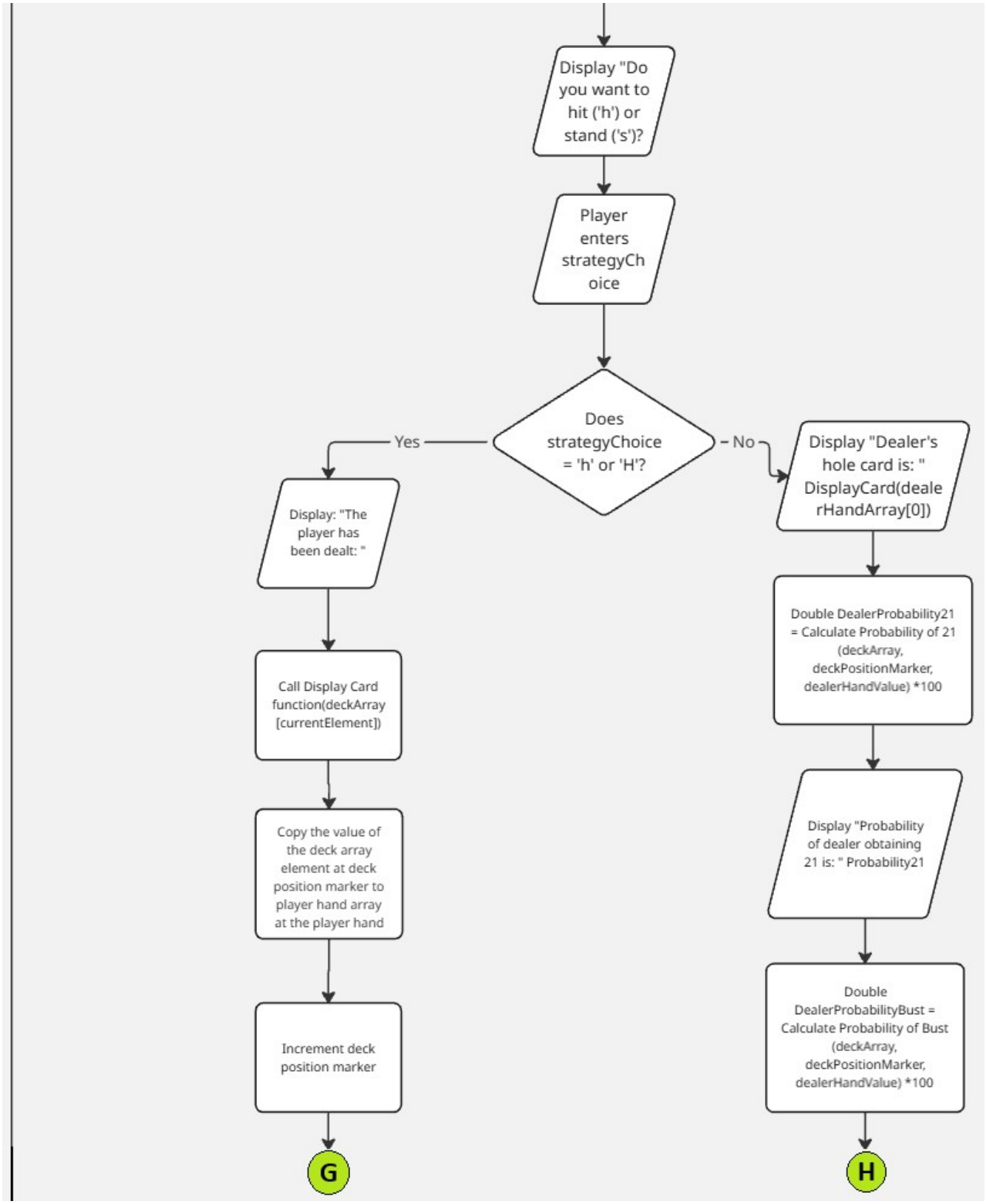

Call Display Card function(deckArray [currentElement])
Copy first element of the deck array to the first element of player hand array
Increment deck position marker
Increment player hand position marker
Copy second element of the deck array to the first element of dealer hand array
Increment deck position marker
Increment dealer hand position marker
This is the dealer's hole card. So it is not displayed to the player until both the end of the player's turn.

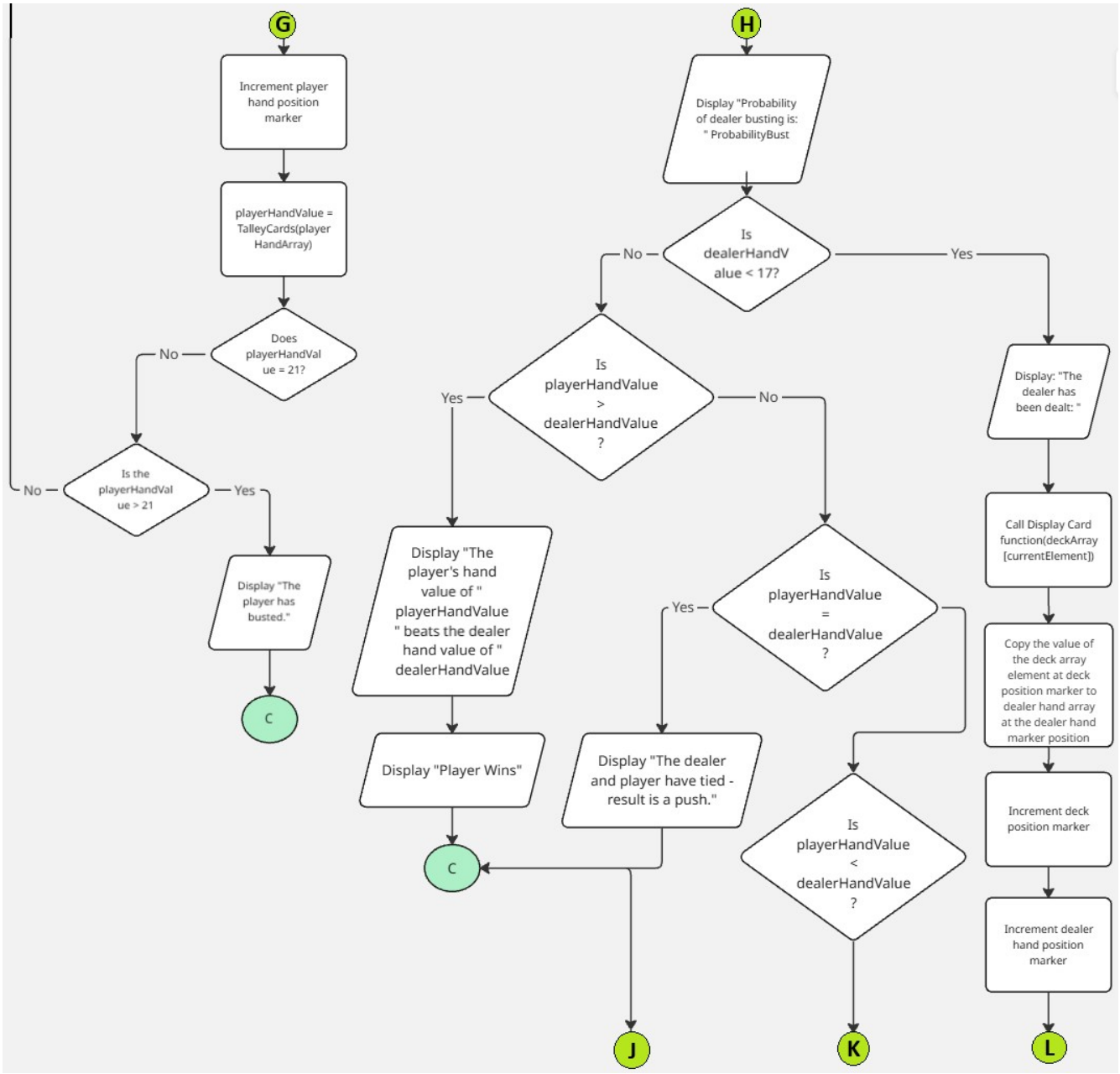

Copy third element of the deck array to the second element of player hand array
Display: "The player has been dealt: "
Call Display Card function(deckArray [currentElement])
Copy third element of the deck array to the second element of player hand array
Increment deck position marker
Increment player hand position marker
Display: "The dealer's up card is: "

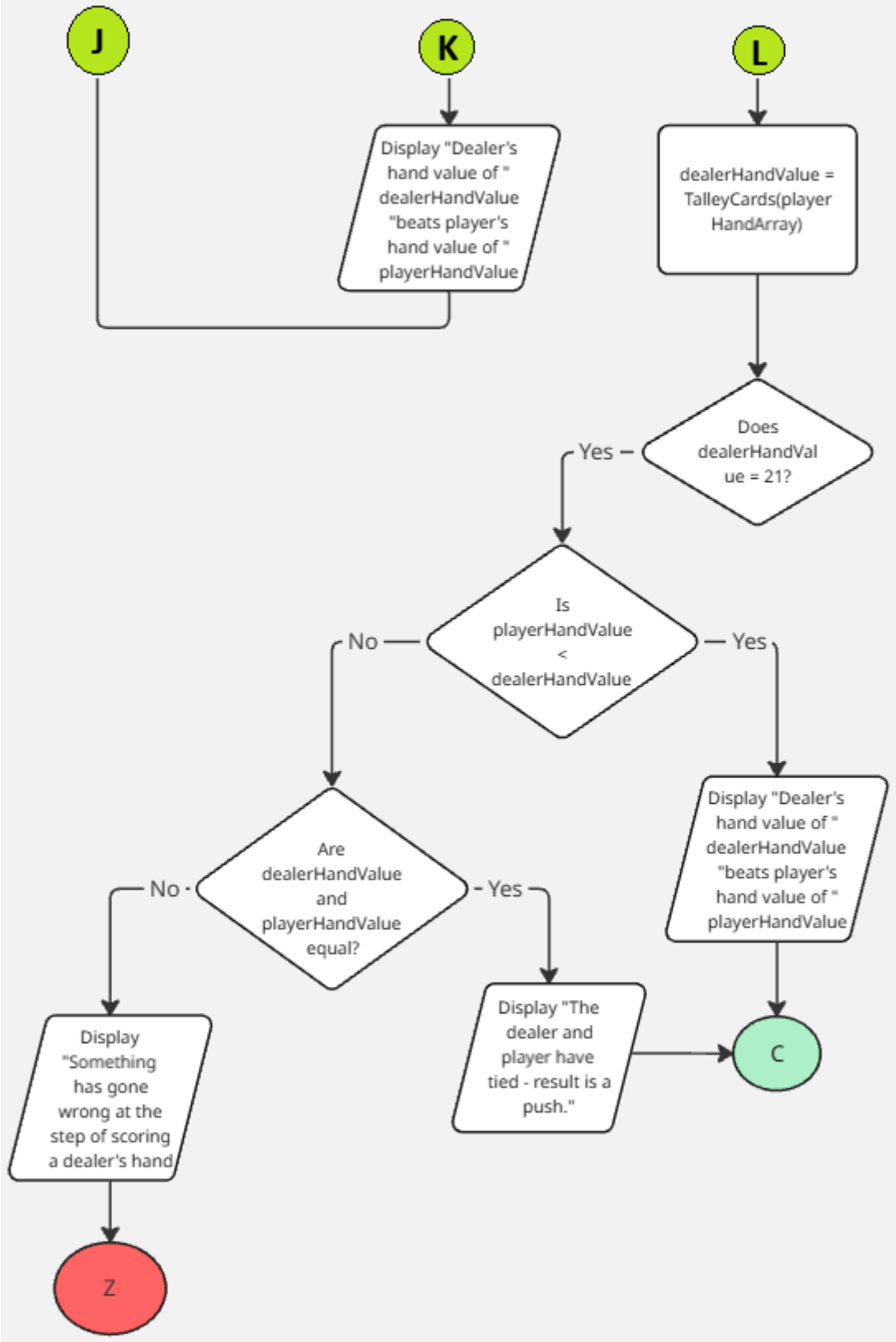

Call Display Card function(deckArray [currentElement])
Copy forth element of the deck array to the second element of dealer hand array
Increment deck position marker
Increment dealer hand position marker
playerHandValue = TalleyCards(player HandArray)

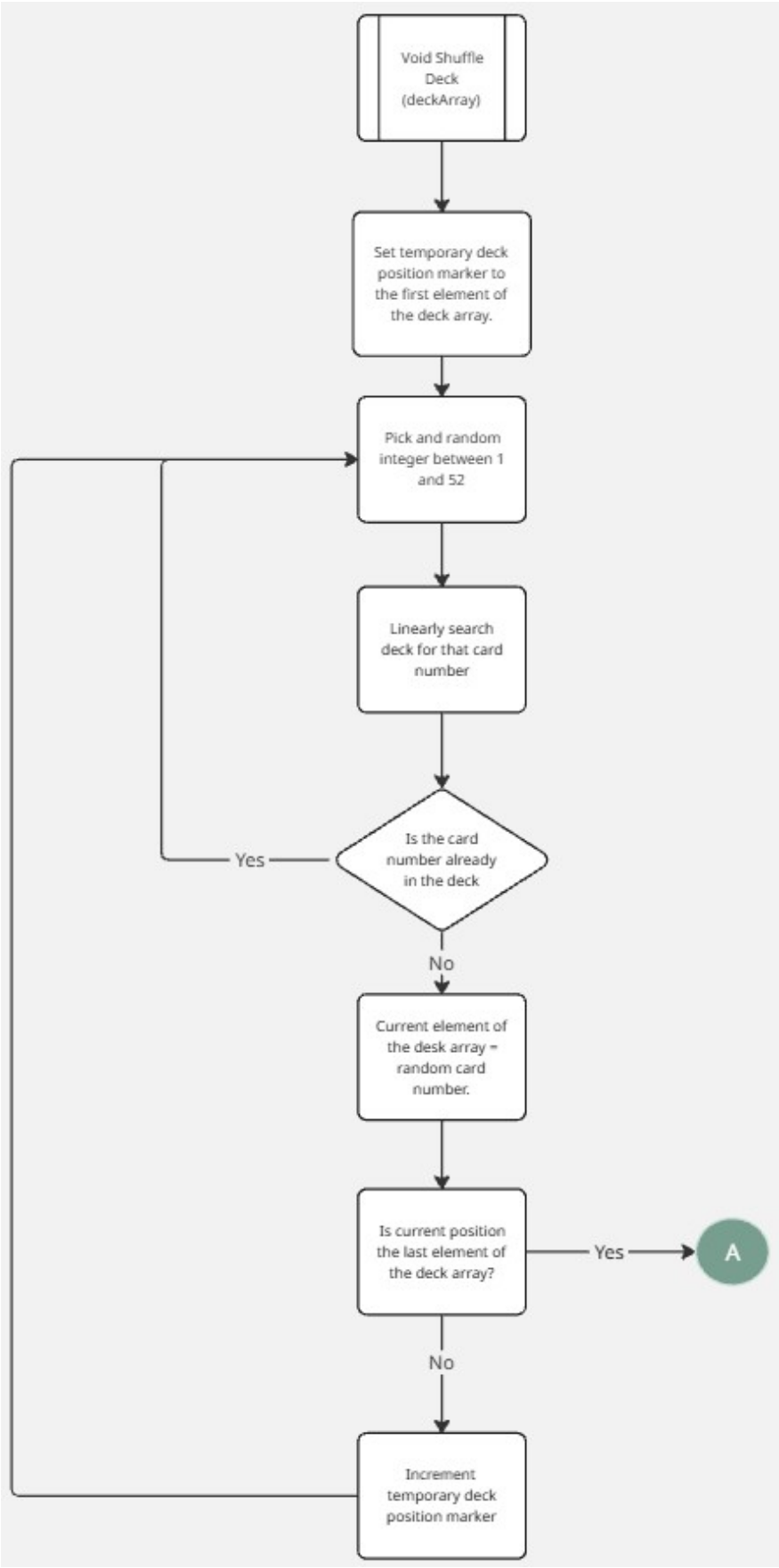

Does playerHandValue = 21?
No
Yes
playerBlackjack = false
playerBlackjack = true
dealerHandValue = TalleyCards(dealer HandArray)
Display"Player: Blackjack!"
D
E

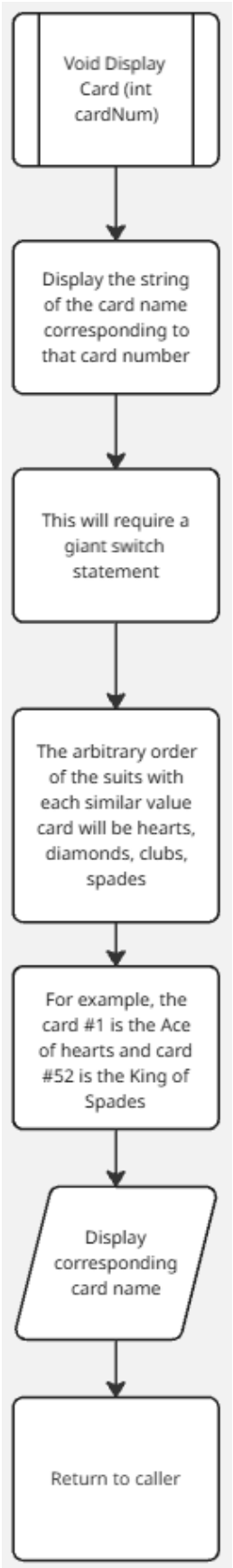

D
Does dealerHandValue = 21?
No
Yes
Double Probability21 = Calculate Probability of 21 (deckArray, deckPositionMarker, playerHandValue) *100
Display "Dealer Blackjack. Dealer Wins."
C
Display "Probability of player obtaining 21 is: " Probability21
Double ProbabilityBust = Calculate Probability of Bust (deckArray, deckPositionMarker, playerHandValue) *100
Display "Probability of player busting is: " ProbabilityBust
E
dealerHandValue = TalleyCards(dealer HandArray)
Does dealerHandValue = 21?
No
Yes
Display "Player wins!
C
deakerBlackjack = true
Display "Dealer Blackjack this game was a tie."
C

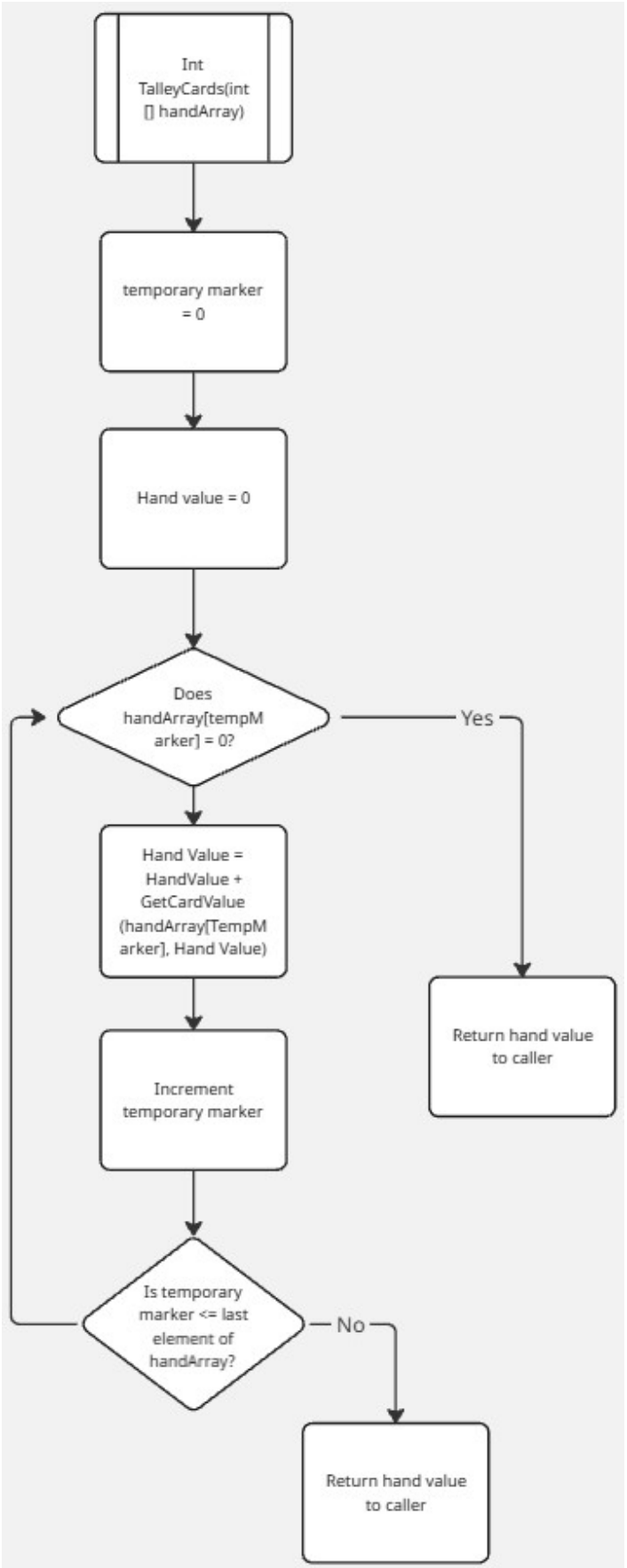

Display "Do you want to hit ('h') or stand ('s')?
Player enters strategyChoice
Does strategyChoice = 'h' or 'H'?
Yes
No
Display: "The player has been dealt: "
Call Display Card function(deckArray[currentElement])
Copy the value of the deck array element at deck position marker to player hand array at the player hand
Increment deck position marker
G
Display "Dealer's hole card is: " DisplayCard(dealerHandArray[0])
Double DealerProbability21 = Calculate Probability of 21 (deckArray, deckPositionMarker, dealerHandValue) *100
Display "Probability of dealer obtaining 21 is: " Probability21
Double DealerProbabilityBust = Calculate Probability of Bust (deckArray, deckPositionMarker, dealerHandValue) *100
H

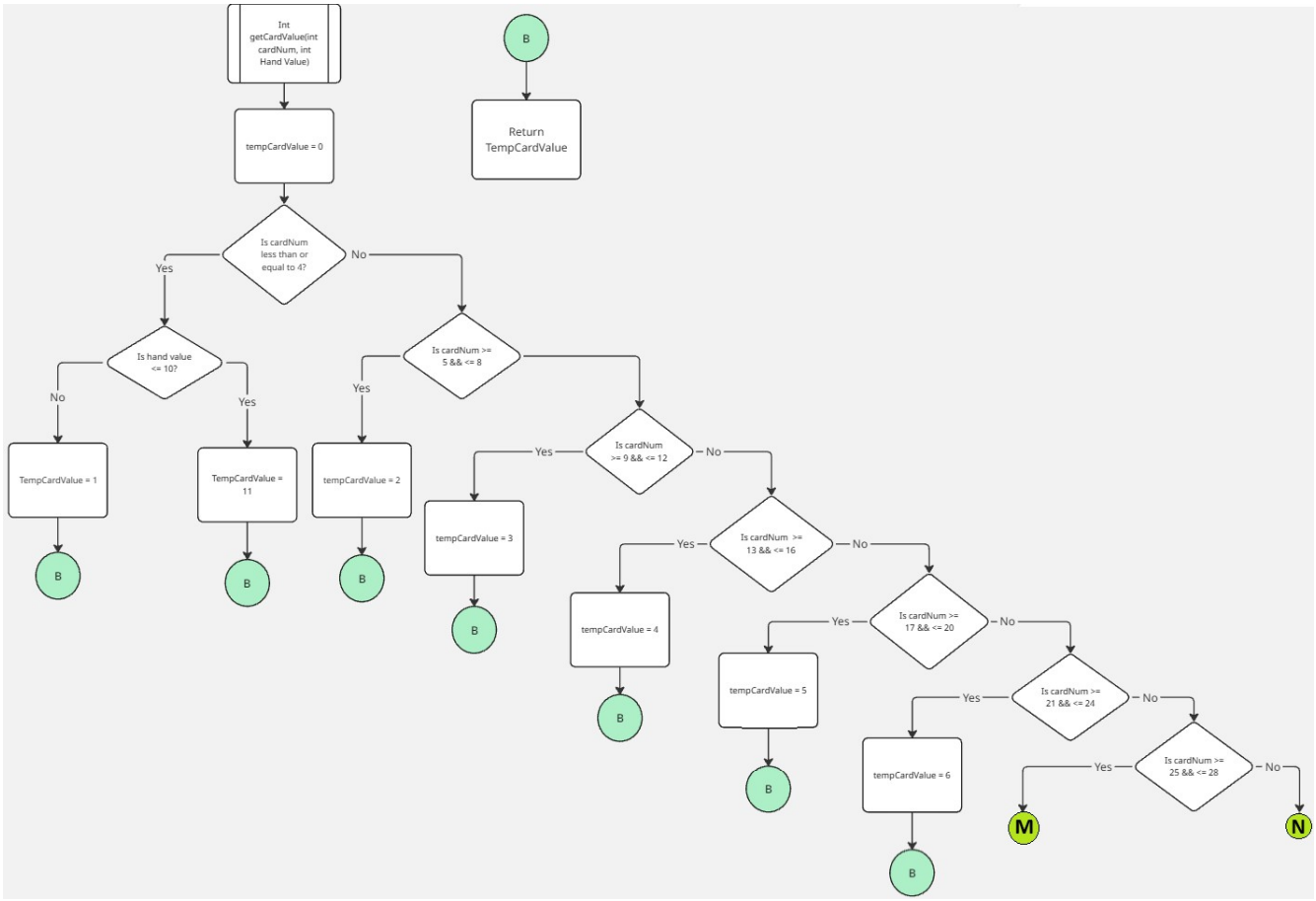

G
Increment player hand position marker
playerHandValue = TalleyCards(player HandArray)
Does playerHandVal ue = 21?
No
Is the playerHandVal ue > 21
No
Yes
Display "The player has busted."
C
H
Display "Probability of dealer busting is: " ProbabilityBust
Is dealerHandV alue < 17?
No
Yes
Is playerHandValue > dealerHandValue ?
Yes
No
Display "The player's hand value of " playerHandValue " beats the dealer hand value of " dealerHandValue
Display "Player Wins"
C
Is playerHandValue = dealerHandValue ?
Yes
Display "The dealer and player have tied - result is a push."
J
Is playerHandValue < dealerHandValue ?
K
Display: "The dealer has been dealt: "
Call Display Card function(deckArray [currentElement])
Copy the value of the deck array element at deck position marker to dealer hand array at the dealer hand marker position
Increment deck position marker
Increment dealer hand position marker
L

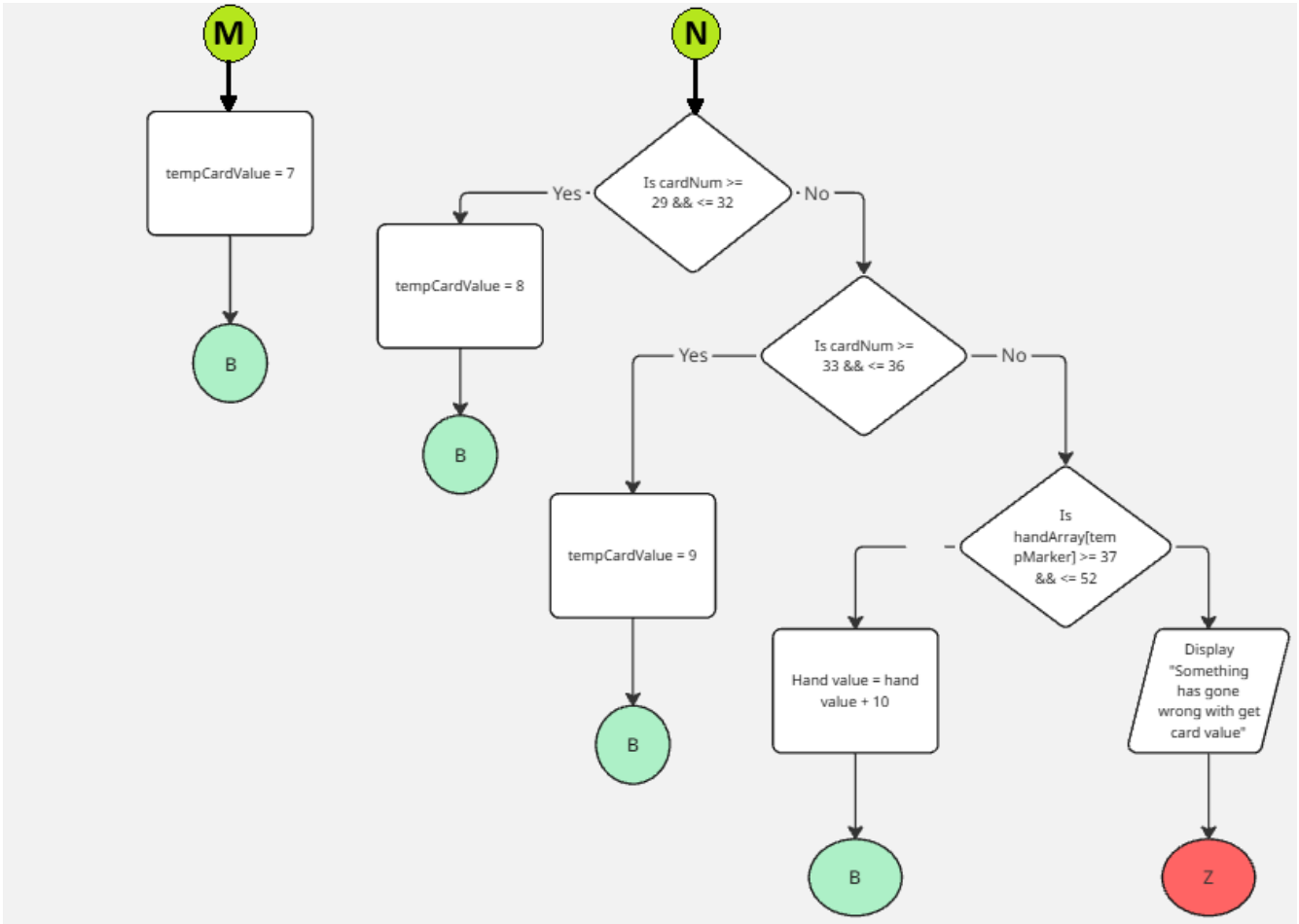

J
K
L
Display "Dealer's hand value of " dealerHandValue "beats player's hand value of " playerHandValue
dealerHandValue = TalleyCards(player HandArray)
Does dealerHandVal ue = 21?
Yes
Is playerHandValue < dealerHandValue
No
Yes
Display "Dealer's hand value of " dealerHandValue "beats player's hand value of " playerHandValue
Are dealerHandValue and playerHandValue equal?
No
Yes
Display "Something has gone wrong at the step of scoring a dealer's hand
Display "The dealer and player have tied - result is a push."
C
Z

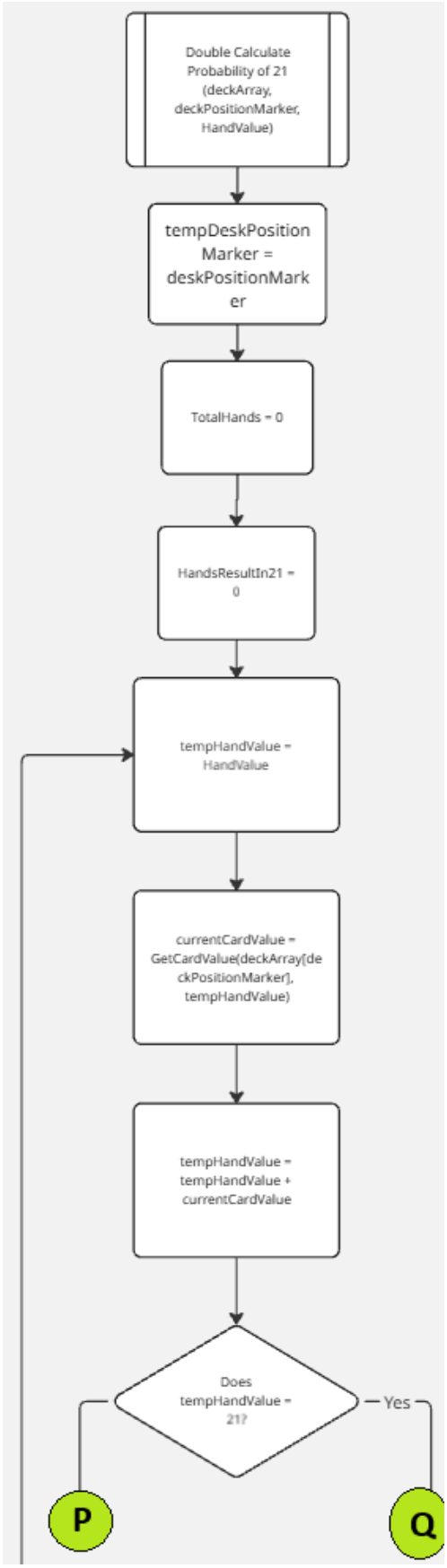

Void Shuffle Deck (deckArray)
Set temporary deck position marker to the first element of the deck array.
Pick and random integer between 1 and 52
Linearly search deck for that card number
Is the card number already in the deck
Yes
No
Current element of the desk array = random card number.
Is current position the last element of the deck array?
Yes
A
No
Increment temporary deck position marker

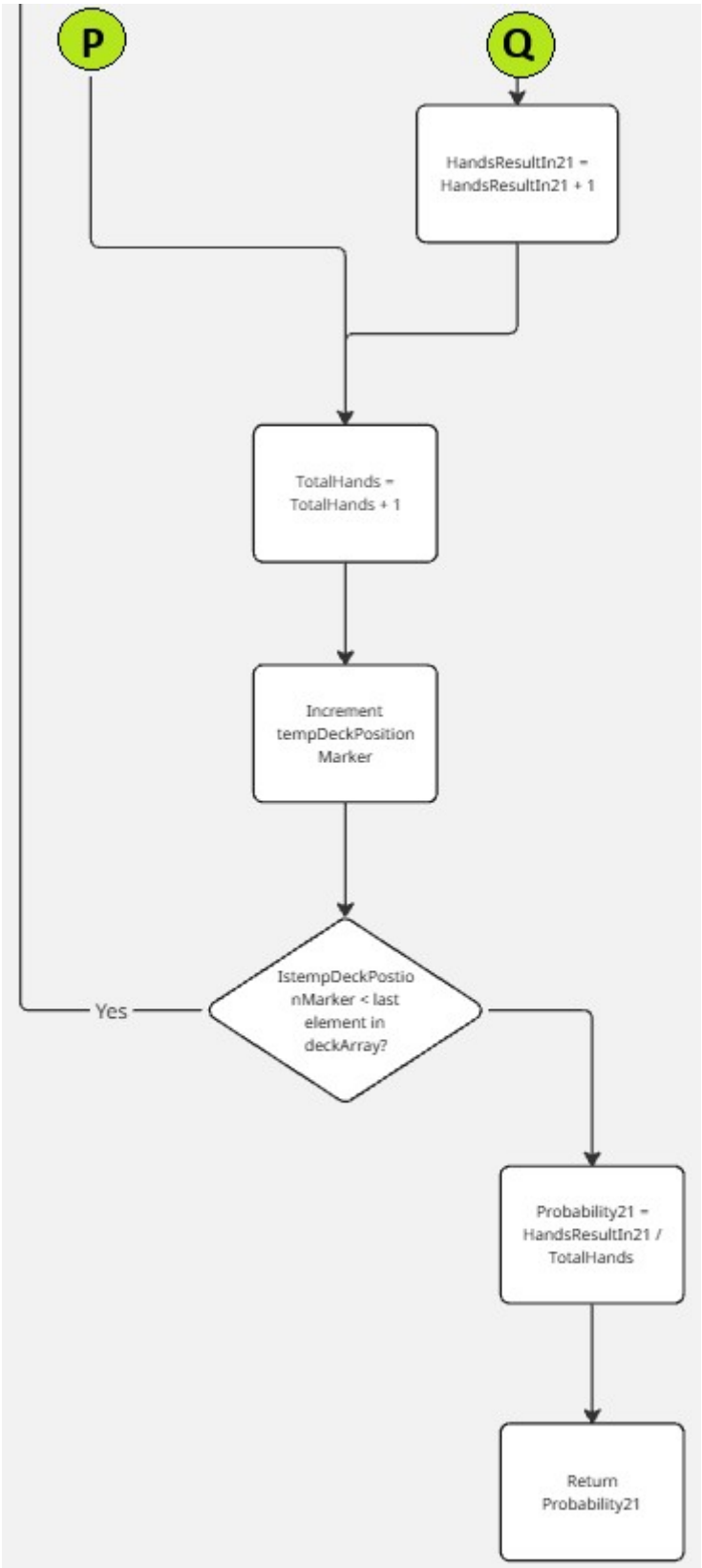

Void Display Card (int cardNum)
Display the string of the card name corresponding to that card number
This will require a giant switch statement
The arbitrary order of the suits with each similar value card will be hearts, diamonds, clubs, spades
For example, the card #1 is the Ace of hearts and card #52 is the King of Spades
Display corresponding card name
Return to caller

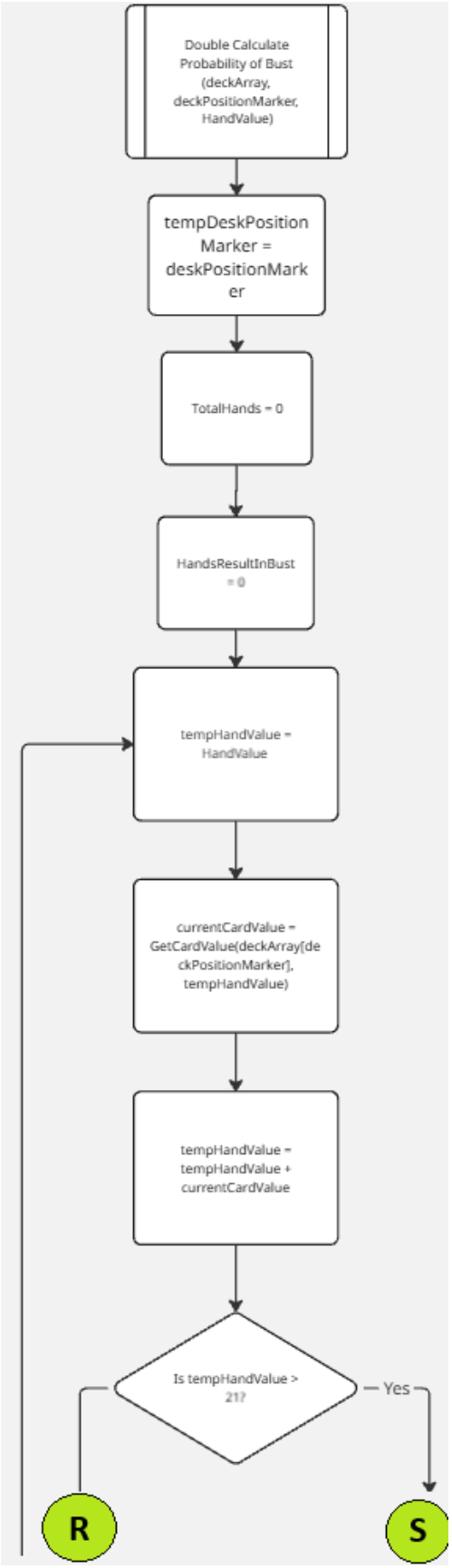

Int TalleyCards(int [] handArray)
temporary marker = 0
Hand value = 0
Does handArray[tempMarker] = 0?
Yes
Hand Value = HandValue + GetCardValue (handArray[TempMarker], Hand Value)
Return hand value to caller
Increment temporary marker
Is temporary marker <= last element of handArray?
No
Return hand value to caller

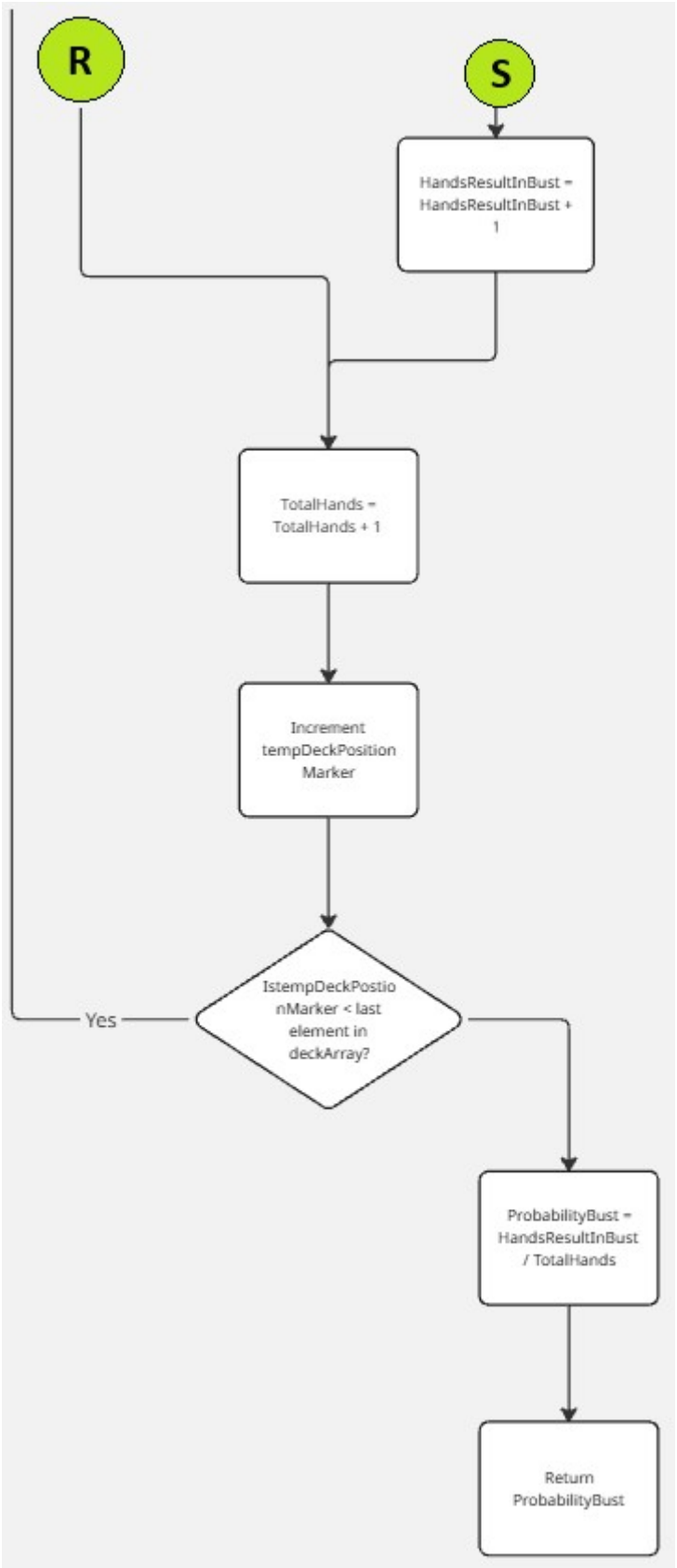

Int getCardValue(int cardNum, int Hand Value)
tempCardValue = 0
Is cardNum less than or equal to 4?
Yes
No
Is hand value <= 10?
No
Yes
TempCardValue = 1
TempCardValue = 11
B
B
Is cardNum >= 5 && <= 8
Yes
tempCardValue = 2
B
Is cardNum >= 9 && <= 12
Yes
tempCardValue = 3
B
Is cardNum >= 13 && <= 16
Yes
tempCardValue = 4
B
Is cardNum >= 17 && <= 20
Yes
tempCardValue = 5
B
Is cardNum >= 21 && <= 24
Yes
tempCardValue = 6
B
Is cardNum >= 25 && <= 28
Yes
M
No
N
B
Return TempCardValue


M
tempCardValue = 7
B
N
Is cardNum >= 29 && <= 32
Yes
tempCardValue = 8
B
No
Is cardNum >= 33 && <= 36
Yes
tempCardValue = 9
B
No
Is handArray[tempMarker] >= 37 && <= 52
Hand value = hand value + 10
B
Display "Something has gone wrong with get card value"
Z


Double Calculate Probability of 21 (deckArray, deckPositionMarker, HandValue)
tempDeskPosition Marker = deskPositionMark er
TotalHands = 0
HandsResultIn21 = 0
tempHandValue = HandValue
currentCardValue = GetCardValue(deckArray[de ckPositionMarker], tempHandValue)
tempHandValue = tempHandValue + currentCardValue
Does tempHandValue = 21?
Yes
P
Q


P
Q
HandsResultIn21 = HandsResultIn21 + 1
TotalHands = TotalHands + 1
Increment tempDeckPosition Marker
IstempDeckPostio nMarker < last element in deckArray?
Yes
Probability21 = HandsResultIn21 / TotalHands
Return Probability21


Double Calculate Probability of Bust (deckArray, deckPositionMarker, HandValue)
tempDeskPosition Marker = deskPositionMarker
TotalHands = 0
HandsResultInBust = 0
tempHandValue = HandValue
currentCardValue = GetCardValue(deckArray[deckPositionMarker], tempHandValue)
tempHandValue = tempHandValue + currentCardValue
Is tempHandValue > 21?
Yes
R
S


R
S
HandsResultInBust = HandsResultInBust + 1
TotalHands = TotalHands + 1
Increment tempDeckPosition Marker
IstempDeckPostio nMarker < last element in deckArray?
Yes
ProbabilityBust = HandsResultInBust / TotalHands
Return ProbabilityBust

Appendix A: Card numbers and their corresponding card name and value

Card number in this program	Card name	Card Value
0	No card present at element position	
1	Ace of hearts	1 or 11
2	Ace of diamonds	1 or 11
3	Ace of clubs	1 or 11
4	Ace of spades	1 or 11
5	Deuce of hearts	2
6	Deuce of diamonds	2
7	Deuce of clubs	2
8	Deuce of spades	2
9	Three of hearts	3
10	Three of diamonds	3
11	Three of clubs	3
12	Three of spades	3
13	Four of hearts	4
14	Four of diamonds	4
15	Four of clubs	4
16	Four of spades	4
17	Five of hearts	5
18	Five of diamonds	5
19	Five of clubs	5
20	Five of spades	5
21	Six of hearts	6
22	Six of diamonds	6
23	Six of clubs	6
24	Six of spades	6
25	Seven of hearts	7
26	Seven of diamonds	7
27	Seven of clubs	7
28	Seven of spades	7
29	Eight of hearts	8
30	Eight of diamonds	8

31	Eight of clubs	8
32	Eight of spades	8
33	Nine of hearts	9
34	Nine of diamonds	9
35	Nine of clubs	9
36	Nine of spades	9
37	Ten of hearts	10
38	Ten of diamonds	10
39	Ten of clubs	10
40	Ten of spades	10
41	Jack of hearts	10
42	Jack of diamonds	10
43	Jack of clubs	10
44	Jack of spades	10
45	Queen of hearts	10
46	Queen of diamonds	10
47	Queen of clubs	10
48	Queen of spades	10
49	King of hearts	10
50	King of diamonds	10
51	King of clubs	10
52	King of spades	10