Project 1

Title:
WAR
a card game

Course:
CSC-17C

Section:
48881

Due Date:
November 19th, 2023

Author:
Michael Donnelly

Table of Contents

Introduction

I am coding the card game “WAR” with a x4 card draw variant. This was the game I played in elementary school and so I have been thinking about this game since then. I always wondered if there was a way to know who would win before the game got played, but the problem was finding other willing participants to play WAR against you. People stop wanting to play this game because it isn’t necessarily fun and it’s not fun because there is ZERO user input. The epiphany I had then was that WAR didn’t require another player and that I could play a game against myself. My program accomplishes this and then simulates how much time it would have taken you to run the game yourself with your own deck of cards so you can see how much time you avoided wasting by using my program.

I spent an equal amount of time planning this project as I did coding for it: Altogether, I spent about 20 hours planning, designing, and coding it. The most challenging part that took the longest was designing the scope of the game such that it would meet the project requirements. What I planned out did not meet the project length requirement so in the future I would pay special attention to this until I learn to better estimate how much code it will take to make each section of a project.

Lines of Code	596
Whitespace	62
comments	153
Total lines	811

The game has only a single Class that stores and outputs terrible ASCII art.

Github link to project:

https://github.com/Marnatee/CSC17C/tree/main/CSC17C_Project1v1

Approach to Development

Concepts: Originally, I was disappointed about not being able to use Vectors for this project but then when I thought about the game of WAR, it consists of 2 phases, skirmishes and WARs. Skirmishes take the top card of the player's hand and winnings are deposited at the bottom of the player's hand which is EXACTLY like a Queue container. WARs are done by having the player stack 4 cards with the 4^{th} card placed faceup which is EXACTLY like a Stack container. I can't use Vectors but Queues and Stacks were all I ended up needing. The challenge is making sure the program does the right checks to make sure you have enough cards to enter WAR.

The portion of the program I am most proud of is how I got the WAR() function to call itself recursively. I thought originally that I'd have to code every single WAR scenario but with recursion, I did not have to. I only hard coded the function to detect up to a quad (x4) war but it will still run properly with a quintuple war or higher.

Version Control: I only have one version of the program but I did extensive planning of my program on paper and a whiteboard before I ever committed anything to Netbeans.

Game Rules:

A deck of 52 cards is shuffled and then each player gets 26 cards.

Each player must play their top card:
If your card is greater than your opponent's, you take both cards and place at bottom of your Hand.
Vice versa if opponent's card is greater.
If both cards are equal, then you go to WAR.

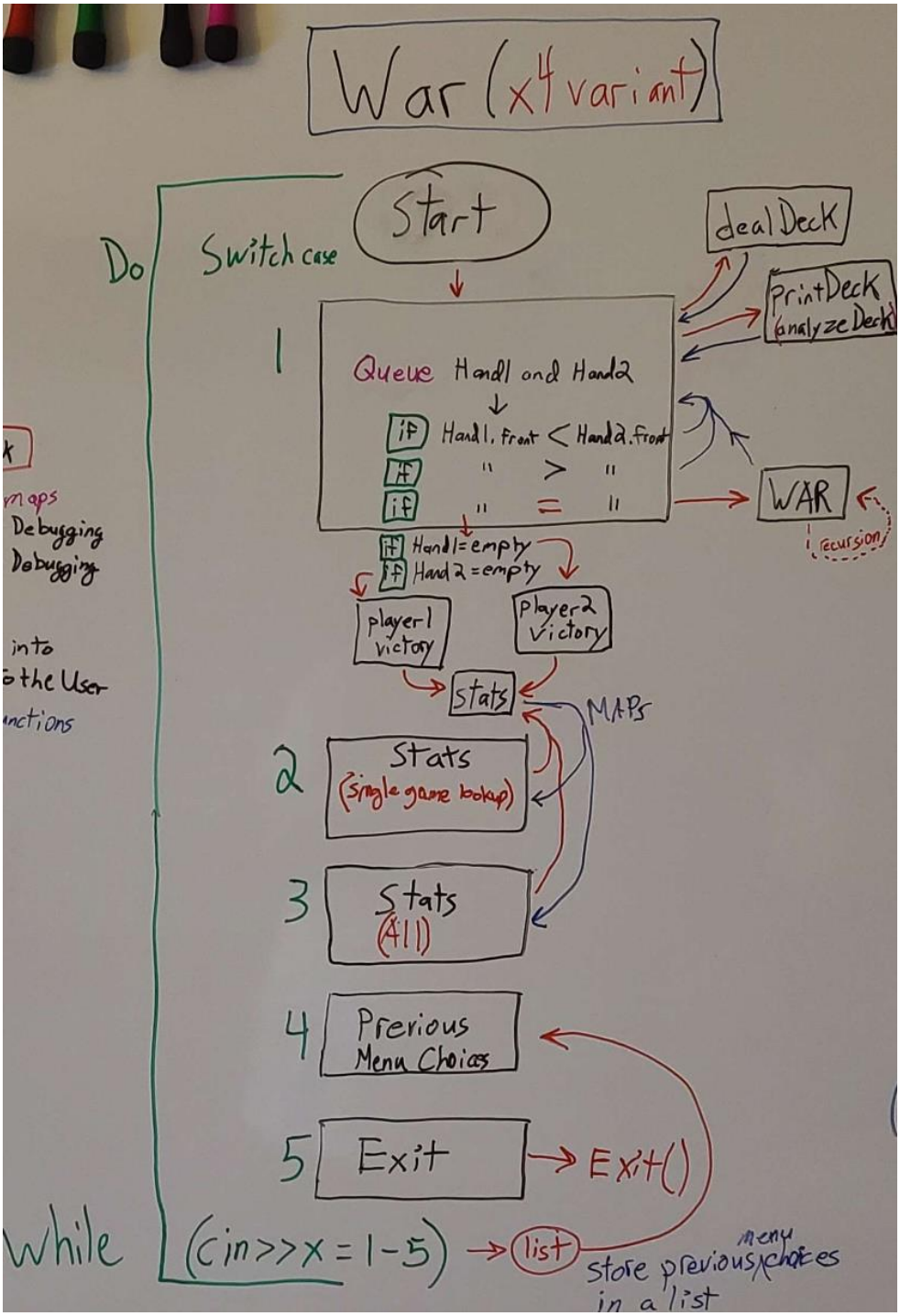
For WAR, each player makes a stack on their prior card with 3 cards facedown and one more card faceup.
if your faceup card is greater than your opponent's, you claim the entire pile and return it to the bottom of your hand. Vice versa if opponent's card is greater.
If both cards are equal, then you go to DOUBLE WAR and repeat the process.
NOTE: This can lead to TRIPLE WAR / QUAD WAR / QUINTUPLE WAR or greater if unlucky enough.

This continues until one player no longer has any cards in their hand. Once this condition occurs, the other player wins.

NOTE: According to bicyclecards.com each player should only have 13 cards (Mine = 26 each) and only 1 card is laid facedown during WARs (Mine = 3 cards facedown). Because of this, I have named my game the "x4 variant"

Description of Code:

This is a snippet from my Whiteboard. Full image is in project folder.



The program starts by entering a Switch case (not actually a switch case, but it's setup like one) with 1 = play game / 2 = Stats (single game lookup) / 3 = Stats (all games) / 4=See previous menu choices / 5=Exit program. The switch is initialized to 1 such that the program will automatically play a single game of WAR when you run it.

1) Play Game

The program starts by creating an array Deck with 52 elements and then calls the dealDeck() function which uses a Set {1,2,3,…13} to initialize the deck to have 4 of each card value 1<->13 (11,12,13 are face cards J/Q/K). This deck is now shuffled twice with the knuth_b protocol. The copy() algorithm is used to copy this deck and then the sort() algorithm is used to sort it. Both unsorted and sorted decks are then displayed to the user so that the user can verify that the cards were dealt right.

Next the program feeds the randomized Deck into the prntDeck() function which prints out the first 26 elements as the Player's hand and the last 26 elements as the CPU's hand. The player's hand is then analyzed in two ways. First, the Hand is summed up and compared to the worst possible hand (98 = 1,1,1,1,2,2,2,2,3,3,3,3,4,4,4,4,5,5,5,5,6,6,6,6,7,7) and the best possible hand (266 = 7,7,8,8,8,8,9,9,9,9,10,10,10,10,11,11,11,11,12,12,12,12,13,13,13,13). Your hand is given a percentage ranking of where it stands in the range of 98-266. Second, the count() algorithm is used to count how many face cards (J/Q/K) are in your hand since they are powerful cards in WAR and almost always beat the card they are played against.

SKIRMISHES

Next the program creates 2 queues: Hand1 (first 26 elements of Deck) and Hand2 (last 26 elements of Deck) proceeds on to skirmishes where the front card of each Queue is compared against each other. If your card is greater than your opponent's card, you win both cards and push them into your Queue at the back. If your opponent's card is greater, then the opposite happens. If both cards are greater then the WAR condition triggers and the WAR() function is called.

```
P1 Hand (22 cards):  8   7   K   9   J   4   3   J   4   8   J   5   8   Q   6   6   K   3   7   K  10   1
P2 Hand (30 cards):  9  10   J   9   5   4  10   5   Q   2   1   5   6   Q   K   3   1   2   1   Q  10   9   6   8   2   3   7   7   4   2
Back to Skirmishes!
8 vs 9   7 vs 10   K vs J   9 vs 9
WAR!
```

WAR

The previous 2 cards + 4 additional cards each are taken from each player's Queue and added to a Stack called WAR1 and WAR2. The top card of each Stack is then compared against each other the same way the Queues were compared in Skirmishes. If your card is greater, you claim both stacks of cards and put them at the back of your queue. If the cards are equal, then the WAR() function calls itself RECURSIVELY and adds 4 more cards to each stack and does another comparison until it solves it.

The card comparisons are continued with skirmishes and WARs until one player's Hand runs out of cards triggering the victory condition for the other Player. Game stats are then added to multiple Map containers.

2) Stats (single game lookup)

The program asks the user for a specific game to lookup and then will re-output all of the stats from that game. All data is stored in Maps, so this will print out the Maps in the user-selected range.

3) Stats (all games)

The program will output all game stats, listed game by game. At the end it will output a TOTAL STATS block that tells you how many games you won out of total games played / skirmishes won out of total skirmishes / wars won out of total wars fought / how many double triple quad wars you got (if any) / The total simulated play time

```
TOTAL STATS FROM ALL 19 GAMES
You won 10 games out of 19 total
You won 635 skirmishes out of 1223 total
You won 59 WARs out of 127 total
5 Double Wars
No Triple Wars
No Quad Wars
Total Simulated Play Time = 42min 41sec
```

4) Previous Menu Choices

This menu option uses a List container to store every choice the player makes in the menu in order. The element at the very back is also displayed on the menu to tell you what you chose last.

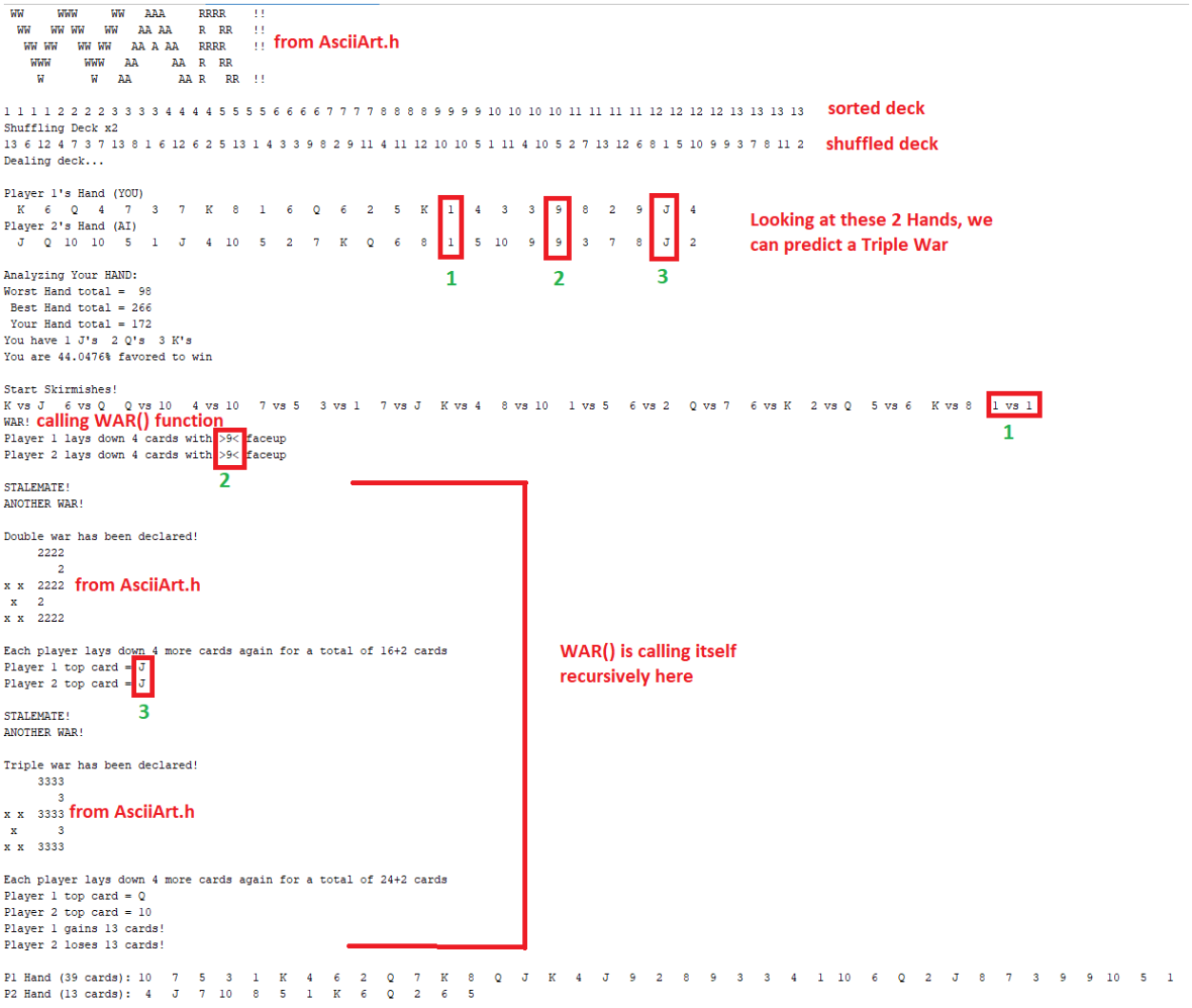
```
You selected these choices
1, 1, 1, 3, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 3, 4,
```

I played 3 games, then printed all my game stats, then played 16 more games, then printed stats, then viewed my menu choices.

5) Exit

This menu option exits the game

Sample output showing the WAR function calling itself recursively




```
 WW     WWW     WW   AAA     RRRR    !!
  WW   WW WW   WW   AA AA    R  RR   !!
   WW WW   WW WW   AA A AA   RRRR    !!
    WWW     WWW   AA     AA  R  RR
     W       W   AA       AA R   RR  !!

1 1 1 1 2 2 2 2 3 3 3 3 4 4 4 4 5 5 5 5 6 6 6 6 7 7 7 7 8 8 8 8 9 9 9 9 10 10 10 10 11 11 11 11 12 12 12 12 13 13 13 13
Shuffling Deck x2
13 6 12 4 7 3 7 13 8 1 6 12 6 2 5 13 1 4 3 3 9 8 2 9 11 4 11 12 10 10 5 1 11 4 10 5 2 7 13 12 6 8 1 5 10 9 9 3 7 8 11 2
Dealing deck...

Player 1's Hand (YOU)
  K  6  Q  4  7  3  7  K  8  1  6  Q  6  2  5  K  1  4  3  3  9  8  2  9  J  4
Player 2's Hand (AI)
  J  Q 10 10  5  1  J  4 10  5  2  7  K  Q  6  8  1  5 10  9  9  3  7  8  J  2

Analyzing Your HAND:
Worst Hand total =  98
 Best Hand total = 266
 Your Hand total = 172
You have 1 J's  2 Q's  3 K's
You are 44.0476% favored to win

Start Skirmishes!
K vs J   6 vs Q   Q vs 10   4 vs 10   7 vs 5   3 vs 1   7 vs J   K vs 4   8 vs 10   1 vs 5   6 vs 2   Q vs 7   6 vs K   2 vs Q   5 vs 6   K vs 8   1 vs 1
WAR!
Player 1 lays down 4 cards with >9< faceup
Player 2 lays down 4 cards with >9< faceup

STALEMATE!
ANOTHER WAR!

Double war has been declared!
     2222
        2
x x  2222
 x   2
x x  2222

Each player lays down 4 more cards again for a total of 16+2 cards
Player 1 top card = J
Player 2 top card = J

STALEMATE!
ANOTHER WAR!

Triple war has been declared!
     3333
        3
x x  3333
 x      3
x x  3333

Each player lays down 4 more cards again for a total of 24+2 cards
Player 1 top card = Q
Player 2 top card = 10
Player 1 gains 13 cards!
Player 2 loses 13 cards!

P1 Hand (39 cards): 10  7  5  3  1  K  4  6  2  Q  7  K  8  Q  J  K  4  J  9  2  8  9  3  3  4  1 10  6  Q  2  J  8  7  3  9  9 10  5  1
P2 Hand (13 cards):  4  J  7 10  8  5  1  K  6  Q  2  6  5
```

Sample Output showing many skirmishes resulting in Player 1 running out of cards and triggering victory condition for player 2.

```
Back to Skirmishes!
8 vs Q   4 vs 8   3 vs 7   2 vs 6   7 vs 6   3 vs 5   6 vs 10   1 vs 9   5 vs Q   4 vs 7   Q vs 8   10 vs 1   7 vs Q
   6 vs 4   Q vs 5   8 vs 2   10 vs J   1 vs 9   6 vs 4   4 vs 2   Q vs J   5 vs 10   8 vs J   2 vs K   6 vs 9   4 v
s J   4 vs 1   2 vs 5   Q vs 2   J vs 9   4 vs 3   1 vs 10   Q vs K   2 vs K   J vs 7   9 vs 1   4 vs 8   3 vs 6   J
 vs K   7 vs 3   9 vs Q   1 vs 8   7 vs 8   3 vs 4
Player 1 ran out of cards.
Player 1 forfeits!
Player 2 VICTORY!
 L     0000  SSSS  SSSS
 L     0  0  S     S
 L     0  0  SSSS  SSSS
 L     0  0     S     S
 LLLL  0000  SSSS  SSSS


Skirmishes fought = 103
Player 1 skirmishes won = 41
Player 2 skirmishes won = 62

Wars fought = 5
Double Wars fought = 2
Triple Wars fought = 0
Quad Wars fought = 0
Simulated Play Time = 3min 4sec
```

lots of skirmishes without triggering a war

Victory condition

Stats generated and stored in Maps for later retrieval

Simulated gametime

Checkoff Sheet

Checkoff List	line/s of code	How it's used:
list	69, 337-344, 355	Used to store menu choices
set	607-618	Used to initialize Deck
map	54-62, 209-216, 259-305 + all of AsciiArt.h	Used to store AsciiArt and Game Stats
stack	443-598	Used exclusively by WAR() function
queue	73-192 and 443-598	Used to do all Skirmish calculations and create Stacks for WAR()
count	429-431	Used to count how many J/Q/K cards are in player's hand
copy	627	Copies deck for output of sorted deck
Shuffle	622-623	Shuffles deck twice
Sort	628	Sorts the copied deck for debug purposes
Recursion	593-599	Used by WAR() function to handle consecutive WARS

Containers	Iterators Used:
list	forward iterator and .back()
set	.top()
map	auto (forward iterator)
stack	.top()
queue	.front()

FlowChart + PseudoCode: Please see image “FlowchartPseudocode.jpg” in project folder

UML Class Diagram: