Milo Bowles

6 December 2023

Farkle Pseu

Documentation

**Special Program Features**

All of the special cases from the Farkle rules should be present in my Farkle program. One special feature I added is that I declared two static variables at the top of my GameController C++ file. These are enteringScore and WinningScore, which determine the score required to enter the game and the score required to win the game, respectively. One would have to open the code in the GameController code file and change the static variables, but if you would like to edit those conditions it is possible. I personally like an enteringScore of 500-700 more than 1000.

**My Process**

I started by reading the rules and writing a barebones pseudocode reflecting the basic function of the game - just rolling dice and scoring them. In writing the pseudocode I realized I was missing quite a bit of knowledge to properly implement the game rules. I also wanted to learn how to apply my pre-existing knowledge of code logic (in Java) to C++ syntax. I read ahead to many different chapters of the textbook that would help me write the basic pseudocode. Then I worked on Visual Studio to transcribe that pseudocode into functional code. This required *a lot*of bug fixing and edits because my pseudocode was filled with mistakes.

This was my overall process: rereading the rules, reading applicable chapters of the textbook, writing pseudocode for the Farkle game implementing what I learned, translating the pseudocode into C++, writing down all my bugs, rinse and repeat. To be honest, I often wrote my code by trial and error in Visual Studio; but I tried as best I could to draft in pseudocode then write C++.

Looking back at my original pseudocode, I hadn’t been thinking in Object Oriented terms at all. I was relying on a single class with a bunch of functions. This seemed achievable when my game was simple - not including special conditions - but I now realize the code would have been a disaster to write and read. My final project submission has far more classes and I think shows how much I have learned about Object Oriented programming. I have a class just for user input validation, because of learning about input validation from the textbook and improving my code. That’s a good example of how much more detailed my code has gotten.

**Errors and Bugs**

I encounter far, far too many bugs to count on this project.

One type of bug I encountered was that when a user input a character instead of a number, the console entered an infinite loop. I fixed this bug by going to the textbook and reading about input validation. That’s when I implemented a class of functions that properly caught input errors like that.

Another type of bug I had was that the player number of the winner would be printed as a series of characters (the address in memory?). To fix this I used the debugger. I stepped through the code and noticed that at one point the player name variable is cleared and becomes a null reference (I think?). Stepping through again I found out that my code for clearing the screen was clearing the variables of the Player object too. I still don’t know why, but I moved the clear screen code into its own function, and fixed the problem.

Another type of bug I encountered was a runtime error, where the user could select to hold die#6 in a selection of 5 die. This threw an indexOutOfRange exception that helped me find my mistake.

The most common bug for me was obviously compile-time errors - little syntax or naming mistakes. For all types of errors my strategies to fix them was to look at the error message, learn from the textbook, or use the debugger.

**Pseudocode**

The many pages that follow show my planning of the different classes and code in my Farkle Game. I have updated all my pseudocode to line up with my final project code ( as you said in feedback for one of my assignments); however it is very long and very detailed. The truth is that the code is much easier to read in Visual Studio, with documentation, and that I actually strayed from my pseudocode plans a lot in my coding process. If you would like to see the more rudimentary pseudocode I used to help plan my project, I submitted it for the last pseudocode assignment (5-1).

Function: main

Returns: integer

INIT gameCont to CALL GameController

CALL gameCont prepGame

**Class:** GameController **Variables:** players, winner (pointer), turnScore, activeDice,

isTurnOver, isGameOver, hasRerolled,

enteringScore, winningScore

Function: GameController constructor

INIT players to a list of player objects

INIT winner pointing to nothing

INIT turnScore at 0

INIT activeDice at 6

INIT isTurnOver at false

INIT isGameOver at false

INIT hasRerolled at false

INIT dice to CALL Dice

Function: prepAndBegin

CALL readRules

INIT prompt to player number prompt

INIT intMaxSize at CALL numericLimits with generic type int CALL max

INIT numPlayers at CALL ValidatedUserInput getInt with prompt, 2 and intMaxSize

FOR each index from 0 to 1 less than numPlayers

INIT playerNum at index plus 1

INIT player at CALL Player with playerNum

CALL players pushback with player

END FOR

CALL startGame

Function: startGame

INIT playerNode pointing to players CALL getLast next

DO  
 INIT player at playerNode data

CALL takeTurn with player

IF player CALL getTotalScore is greater than winningScore THEN

CALL takeLastChanceTurns with player

END IF

Point playerNode to playerNode next

WHILE isGameOver is false

WRITE winning message

WRITE game over message

Function: readRules

INIT fileName at chosen file name

INIT inputFile

inputFile CALL open with fileName

IF inputFile is true, opened successfully THEN

INIT line

WHILE CALL getline, with inputFile and line, is true

WRITE line with a carriage return

END WHILE

CALL pressEnterToContinue

ELSE

WRITE error opening file message

END IF

Function: takeTurn

Parameters: player

INIT isTurnOver at false

INIT hasRerolled at false;

INIT activeDice at 6

INIT turnScore at 0

CALL system with clear screen string

WRITE player CALL toString

INIT prompt at roll dice prompt

INIT validChars with yes and no characters list

INIT choice

WHILE isTurnOver is false

IF activeDice equals 0

CALL stopRolling with player

ELSE IF player CALL getHasEntered is false

CALL rollDice with player

ELSE

SET choice to CALL ValidatedUserInput getChar, with prompt and validChars

IF choice is yes THEN

CALL rollDice with player

ELSE IF choice is no THEN

CALL stopRolling with player

END IF

END IF

END WHILE

WRITE turnScore to console

WRITE totalScore to console

CALL incrementTurn

CALL pressEnterToContinue

Function: takeLastChanceTurns

Parameters: potentialWinner

SET winner pointing to potentialWinner

WRITE winner message to console

INIT playerNode at players CALL getLast next reference

DO

INIT player at playerNode data

IF player CALL isEqual with potentialWinner is false THEN

CALL takeTurn with player

IF player CALL getTotalScore is greater than winner CALL getTotalScore

SET winner pointing to player

END IF

END IF

SET playerNode pointing to playerNode next

WHILE playerNode does not equal players CALL getLast next reference

SET isGameOver to true

Function: rollDice

Parameters: player

dice CALL setDie with activeDice

dice CALL rollAll

WRITE dice CALL toString

INIT rollResults list at, dice CALL getDice

CALL scoreDice with player and rollResults

Function: stopRolling

Parameters: player

IF player CALL getHasEntered is false THEN

SET turnScore to 0

END IF

IF turnScore does not equal 0 THEN

INIT totalScore at player CALL getTotalScore

INIT newTotal at CALCULATE totalScore plus turnScore

player CALL setTotal with newTotal

END IF

SET isTurnOver to true

Function: scoreDice (it's a long one)

Parameters: player, rollResults

INIT diceToScore at rollResults

INIT prompt at hold dice prompt

INIT validChars at valid character list, yes and no

INIT choice at ValidatedUserInput CALL getChar with prompt and validChars

IF choice is yes THEN

CALL holdDice with diceToScore

END IF

INIT faceValueCounts at list size 6

FOR each die in diceToScore

INIT faceValue at die CALL getValue

INIT faceValueIdx at faceValue minus 1

INCREMENT faceValueCounts, at faceValueIdx, by 1

END FOR

INIT pointDice at 0

FOR index from 0 to 6

INIT faceValue at index plus 1

IF faceValueCounts at index is greater than 3 THEN

INIT numThreeOfAKinds to faceValueCounts at index, integer division by 3

INIT pointDice by 3 times numThreeOfAKinds

FOR index from 0 to less than numThreeOfAKinds

IF faceValue is 1 THEN

WRITE three 1s message

INCREMENT turnScore by 1000

ELSE

WRITE specific three of a kind

INCREMENT turnScore by faceValue times 100

END IF

END FOR

END IF

IF faceValue is 1

WRITE number of 1s message

INIT remainders to faceValueCounts at index, remainder 3

INCREMENT pointDice by remainders

INCREMENT turnScore by remainders times 100

ELSE IF faceValue is 5

WRITE number of 5s message

INIT remainders to faceValueCounts at index, remainder 3

INCREMENT pointDice by remainders

INCREMENT turnScore by remainders times 50

END IF

END FOR

SET activeDice to activeDice minus pointDice

WRITE turnScore

CALL checkIfPlayerCanEnter with player

IF pointDice is 0 THEN

CALL farkle with player

ELSE IF pointDice is 6 THEN

CALL reroll with player

ELSE IF activeDice is 0 THEN

CALL stopRolling

END IF

Function: holdDice

Parameters: diceToScore

INIT choice at yes

WHILE choice is yes

INIT arePointDiceToHold to CALL checkIfPointDiceToHold with diceToScore

IF arePointDiceToHold is false

WRITE no point dice to hold message

SET choice to no

ELSE

INIT stringstream

INSERT which dice to hold message into stringstream

INIT promptDieToHold at stringstream CALL str

INIT dieToHold at ValidatedUserInput CALL getInt

INIT dieToHoldIdx at dieToHold minus 1

diceToScore CALL erase with iterator plus dieToHoldIdx

FOR each die in diceToScore

WRITE die CALL getValue

END FOR

INIT validChars at yes and no

SET choice to ValidatedUserInput CALL getChar with prompt and validChars

END IF

END WHILE

Function: farkle

Parameters: player

SET turnScore to 0

WRITE farkleMessage

CALL stopRolling

Function: reroll

Parameters: player

WRITE reroll message

SET activeDice to 6

SET hasRerolled to true

CALL rollDice with player

Function: checkIfPlayerCanEnter

Parameters: player

IF turnScore is greater than enteringScore and player CALL getHasEntered is false THEN

player CALL setHasEntered with true

WRITE entered game message

END IF

Function: pressEnterToContinue

WRITE press enter to continue message

cin CALL get

CALL system with clear screen string

Function: checkIfPointDiceToHold

Parameters: dice

Returns: boolean

INIT pointDice at 0

INIT faceValuesArray at size 6

FOR each die in dice

INIT faceValue at die CALL getValue

INIT faceValue index at faceValue minus 1

INCREMENT faceValuesArray at faceValueIndex by 1

END FOR

FOR index 0 to 6

INIT faceValue at index plus 1

IF faceValue is 1 or 5 THEN

INCREMENT pointDice by faceValuesArray at index

ELSE

IF faceValueArray at index is greater than 3

INCREMENT pointDice by faceValueArray at index, remainder 3

END IF

END FOR

RETURN whether pointDice is greater than 1

**Class:** Die **Variables:** value

Function: Die constructor

CALL srand with CALL time with null pointer

Function: getValue

Returns: integer

RETURN value

Function: roll

INIT randNum at CALL rand

SET randNum to CALCULATE randNum remainder 6

INCREMENT randNum by 1

SET value to randNum

**Class:** Dice **Variables:** dice

Function: addDie

Parameters: die

CALL dice CALL pushback with die

Function: Dice constructor

dice CALL clear

Function: rollAll

FOR each die in dice

die CALL roll

END FOR

Function: getDice

Returns: list of die

RETURN dice

Function: toString

Returns: string of characters

INIT sstream

FOR each die in dice

INSERT die CALL getValue into sstream

END FOR

INSERT newLine into sstream

RETURN sstream CALL str

**Class:** LinkedList **-** Generic of type name “T” **Variables:** last - pointer

Function: LinkedList constructor

Point last to nothing

Function: addToEmpty

Parameters: data

IF last points to nothing THEN

INIT newNode with data

Point last to newNode

Point last next to newNode

END IF

Function: addToFront

Parameters: data

IF last points to nothing THEN

CALL addToEmpty with data

ELSE

INIT newNode with data

Point newNode next to last next

Point last next to newNode

END IF

Function: addToEnd

Parameters: data

IF last points to nothing THEN

CALL addToEmpty with data

ELSE

INIT newNode with data

Point newNode next to last next

Point last next to newNode

Point last to newNode

END IF

Function: deleteAtFront

IF last points to something THEN

IF last next points to last THEN

DELETE last

Point last to nothing

ELSE

INIT OGFront pointing to last next

Point last next to last next next

DELETE OGFront

END IF

END IF

Function: deleteAtEnd

IF last points to something THEN

IF last next points to last THEN

DELETE last

Point last to nothing

ELSE

INIT OGLast pointing to last

INIT temp pointing to last

WHILE temp next does not equal last

Point temp to temp next

END WHILE

Point temp next to temp next next

Point last to temp

DELETE OGLast

END IF

END IF

Function: clear

WHILE last points to something

CALL deleteAtFront

END WHILE

CALL deleteAtFront

Function: size

INIT size at 0

IF last points to something

INIT temp pointing to last next

DO

INCREMENT size by 1

Point temp to temp next

WHILE temp does not equal last next

END IF

Function: getLast

Returns: pointer

RETURN last

**Class:** Player **Variables:** number, totalScore, turn, hasEntered

Function: Player constructor (default)

Parameters:

SET number to -1

INIT totalScore at CALL setTotalScore 0

INIT turn at CALL setTurn with 1

SET hasEntered to false

Function: Player constructor

Parameters: number

INIT number at CALL setNumber number

INIT totalScore at CALL setTotalScore 0

INIT turn at CALL setTurn with 1

SET hasEntered to false

Function: setNumber

Parameters: numberP

IF numberP is less than 1 THEN

THROW argument with error message

END IF

SET number to numberP

Function: getNumber

Returns: integer

RETURN number

Function: setTotalScore

Parameters: score

IF score is 0 or less THEN

THROW argument with error message

END IF

SET totalScore to score

Function: getTotalScore

Returns: integer

RETURN totalScore

Function: setTurn

Parameters: turnP

IF turnP is 0 or less THEN

THROW argument with error message

END IF

SET turn to turnP

Function: getTurn

Returns: integer

RETURN turn

Function: incrementTurn

INCREMENT turn by 1

**Class: ValidatedUserInput**

Function: ValidatedUserInput default constructor - DO NOTHING

Function: getDouble

Parameters: prompt, min, max, isInclusive

Returns: double

INIT num at 0.0

INIT isValid at false

WHILE isValid is false

WRITE prompt to console

IF READ num from console is in bad state THEN

CALL handleInvalidInput

ELSE

CALL discardRemainingCharacters

IF isInclusive is true

SET isValid to CALL checkRangeInclusive

ELSE

SET isValid to CALL checkRangeExclusive

END IF

END IF

END WHILE

RETURN num

Function: getString

Parameters: prompt, isInsensitive

Returns: string of characters

INIT choice at empty string

WRITE prompt to console

READ choice from console

CALL discardRemainingChars

IF isInsensitive is true THEN

FOR each character in choice

SET character to lowercase

END FOR

END IF

RETURN choice

Function: getInt

Parameters: prompt, min, max, isInclusive

Returns: integer

INIT num at 0

INIT isValid at false

WHILE isValid is false

WRITE prompt to console

IF READ num from console is in bad state THEN

CALL handleInvalidInput

ELSE

CALL discardRemainingCharacters

IF isInclusive is true

SET isValid to CALL checkRangeInclusive

ELSE

SET isValid to CALL checkRangeExclusive

END IF

END IF

END WHILE

RETURN num

Function: getChar

Parameters: prompt, validChars, isInsensitive

Returns: character

INIT choice at n

INIT isValid at false

WHILE isValid is false

WRITE prompt to console

READ choice from console

CALL discardRemainingCharacters

IF isInsensitive is true THEN

SET choice to lowercase

FOR each character in validChars

SET character to lowercase

END FOR

END IF

SET isValid to CALL checkValidChar with choice and validChars

END WHILE

RETURN choice