Rafid Sarowar

CIS-5

25SUM-CIS-5-46224

Project 2

What Is Ludo?

Ludo is a simple race-and-capture board game for 2 to 4 players. Each person rolls a single die to move their tokens from the “yard” onto a cross-shaped track, aiming to bring all pieces safely home. Landing on an opponent’s token sends it back to its yard, combining luck and light strategy in every turn.

Origins and Evolution

Ludo traces its roots to the Indian game Pachisi (6th century AD), where cowrie shells and cloth boards defined royal pastimes. In 1896, British colonists formalized it as “Ludo” (Latin for “I play”), replacing shells with a cubical die and standardizing rules. From Britain, it spread worldwide, adapting to local materials and player styles.

Ludo in Bangladesh

In Bangladesh, লুডু boards—often plastic or chalk-drawn—are staples of monsoon nights and festive gatherings. Families and neighbors crowd verandahs or street corners, sipping tea as dice clatter and laughter fills the air. Impromptu matches bridge generations, blending friendly rivalry with casual storytelling.

A Core Family Memory

Growing up in Bangladesh, Ludo was the game my brother, cousins, and I would always play. I must have whiled away thousands of monsoon-drenched evenings clustered around that battered plastic board in our courtyard. Grandparents would hover, gently guiding our tokens; cousins would erupt in laughter after each cunning “cut”; and my grandmother’s soft scolds came whenever I forgot to say “thank you” after rolling a six. Those countless games—just dice, tokens, and shared stories—wove a bond between us that still brings a smile whenever I think back.

Project Overview

This project is a complete, modular, and professional C++ simulation of Ludo. It demonstrates all required CIS-5 programming concepts, incrementally introduced over six versions.

The final version supports 2–4 players, uses arrays, functions, file I/O, searching, sorting, and more, all while maintaining a clean style and thorough documentation.

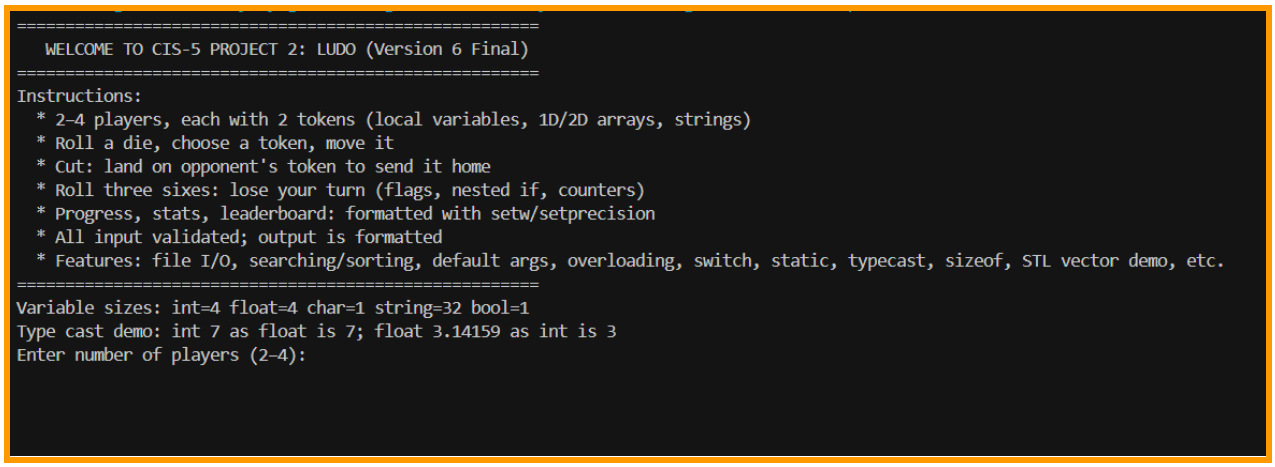
How to Play Ludo (Final Version Instructions)

Welcome to Ludo – CIS-5 Advanced Edition!

How to Play:

- 2–4 players (or you vs friends/AI).
- Each player has two tokens starting at tile 0 (“home”).
- On your turn, press Enter to roll a die (1–6).
- Choose which of your two tokens to move.
- Special Rules:
    - Cut rule: Land on another player’s token, and that token goes back to the start.
    - Safe-stop: Tokens never move past tile 50.
    - Triple six penalty: Roll three 6’s in a row? You lose your turn!
- First player to get both tokens to tile 50 wins.
- Every action, roll, and cut is logged to a file. Progress, leaderboard, and stats are shown at the end.

Sample Game Screen:



Version-by-Version Evolution

Each version builds on the last—see the pseudocode and code snippet for each.

Version 1: Modular Roll-and-Move (2 Players, Functions Only)

Features:

- Two players, each with two tokens (no arrays—just `int` variables).
- Modular design using separate functions for:
    - Rolling the die.
    - Moving a token.
    - Checking for a win.
    - Printing status.
- Turn-based game loop, switches between Player 1 and Player 2.

- Simple input and output.

Key C++ Concepts:
Functions, local variables, input/output, basic control flow.

Pseudocode:

```
* 1. Initialize player tokens to 0 (start position)
2. Seed the random number generator
3. Set Player 1 as current player
4. While no one has won:
    a. Roll the die (call rollDie())
    b. Ask current player to pick which token to move (1 or 2)
    c. Move selected token by the die roll (call moveToken())
    d. Print both players' token positions (call printStatus())
    e. Check if current player has both tokens at or past finish (call checkWin())
        - If yes, announce winner and exit loop
    f. Switch to the next player
*/
```

Code Snippet:
Modular with functions—no arrays yet.

```
// --- Function prototypes ---
int rollDie();
void moveToken(int &token, int roll);
bool checkWin(int t1, int t2);
void printStatus(int p1_1, int p1_2, int p2_1, int p2_2);
```

Version 2: Using 1D Arrays and Functions Passing Arrays

New Features:

- Tokens for each player are now stored in 1D arrays (`p1[2]`, `p2[2]`).
- Functions now accept arrays (demonstrates passing by reference).

- Win check and print functions generalized to arrays.
- Still 2 players, improved structure, and maintainability.

Key C++ Concepts:
1D arrays, passing arrays to functions, modular design, pass by reference/value.

Pseudocode:

```
* 1. Declare int arrays for player tokens: p1[2], p2[2]
* 2. Functions for roll, move, print, win now take arrays (by value/ref)
* 3. Main loop: roll, pick token, move (using function), print, check win
* 4. No global variables/constants
*/
```

Code Snippet:
Move tokens with arrays, pass by reference:

```
// Moves a specific token (array by ref, index, roll)
void moveToken(int tokens[], int idx, int roll)
{
    const int FINISH = 50;
    tokens[idx] += roll;
    if (tokens[idx] > FINISH)
        tokens[idx] = FINISH; // optional safe-stop
}
```

Version 3: 2D Arrays, 2–4 Players, Full Modularization

New Features:

- The game supports 2–4 players (user input for the number of players).
- All tokens stored in a 2D array (`tokens[4][2]`).
- All logic (move, check, print) is generalized for any number of players.
- Dynamic turn order, cycles through players.

Key C++ Concepts:
2D arrays, parallel arrays, loops, scalable code, and input validation.

Pseudocode:

```
* 1. Prompt user for number of players (2-4).
* 2. Declare 2D array: tokens[4][2].
* 3. Game loop:
   a. Current player rolls.
   b. Select token.
   c. Move token (with safe-stop at finish).
   d. Print all players' positions.
   e. Check for win (both tokens finished).
   f. Advance to next player.
*/
```

Code Snippet:
Generalize for any player, with 2D arrays:

```
// Move a token, with safe-stop at finish
void moveToken(int tokens[][2], int plIdx, int tokIdx, int roll, int finish)
{
    tokens[plIdx][tokIdx] += roll;
    if (tokens[plIdx][tokIdx] > finish)
        tokens[plIdx][tokIdx] = finish;
}
```

Version 4: Cut Rule, File Logging

New Features:

- Implements "cut" rule: if you land on an opponent's token, they return to start.
- Generalized cut logic for any number of players using arrays.
- Logs every roll, move, and cut to a text file.
- Functions for cutting and logging.

Key C++ Concepts:
Conditional logic, file I/O (ofstream), more advanced function use, and array manipulation.

Pseudocode:

```
*   1. Initialize 2D array for tokens, open output file.
* 2. Main game loop:
    a. Player rolls (log to file).
    b. Player selects token.
    c. Move token (log to file).
    d. Check for and perform cuts (log to file).
    e. Print all positions.
    f. Check for win.
    g. Advance to next player.
* 3. Close log file on exit.
*/
```

Code Snippet:
Cut logic:

```
// Cut any opponent tokens on this tile, log cuts
void cutTokens(int tokens[][2], int plCnt, int plIdx, int tokIdx, int start, ofstream &logF)
{
    int movedPos = tokens[plIdx][tokIdx];
    for (int op = 0; op < plCnt; ++op)
    {
        if (op == plIdx)
            continue; // don't check own tokens
        for (int t = 0; t < 2; ++t)
        {
            if (tokens[op][t] == movedPos && movedPos != start)
            {
                tokens[op][t] = start;
                cout << "Cut! Sent Player " << (op + 1) << " Token " << (t + 1) << " to start." << endl;
                logF << "Cut P" << (op + 1) << "T" << (t + 1) << ". ";
            }
        }
    }
}
```

Version 5: Statistics, Sorting, Searching, Leaderboard

New Features:

- Tracks statistics: number of moves, turns, and cuts per player (parallel arrays).
- After the game ends:
    - Bubble sort to show players by fewest moves.
    - Selection sort for the fewest turns.
    - Linear search to find the player with the most cuts.
    - Leaderboard output (to file and console).
- Uses STL vector demo for leaderboard.

Key C++ Concepts:
Bubble sort, selection sort, searching arrays, STL vector basics, and formatted output.

Pseudocode:

```
*
* 1. Declare 2D arrays for tokens and moves, 1D arrays for stats (turns, cuts).
* 2. Input player names.
* 3. Game loop:
    a. Player rolls.
    b. Select token.
    c. Move token (record moves).
    d. Perform cuts (update cuts stat).
    e. Print status.
    f. Check for win (record turns, time, etc.).
    g. Advance player.
* 4. When game ends:
    a. Sort all players by performance (moves/cuts).
    b. Search for player with most cuts.
    c. Print stats/leaderboard.
    d. Write stats to leaderboard file.
*/
```

Code Snippet:
Bubble sort for leaderboard:

```
// Sort players by moves (ascending, parallel array/string)
void sortPlayers(string names[], int moves[], int plCnt)
{
    for (int i = 0; i < plCnt - 1; ++i)
    {
        int minIdx = i;
        for (int j = i + 1; j < plCnt; ++j)
            if (moves[j] < moves[minIdx])
                minIdx = j;
        // Swap both arrays
        swap(moves[i], moves[minIdx]);
        swap(names[i], names[minIdx]);
    }
}
```

Version 6: The Ultimate, Checklist-Full Ludo

New Features:

- All required C++ features: No globals, function prototypes, pass by value/reference, overloading, defaulted parameters, static variables, exit().
- Full user instructions and formatted output.
- Switch statement (for die roll output).
- Conditional (ternary) operator in status.
- Math library functions (`pow`, `abs`), type casting, use of `sizeof`.
- Advanced input validation everywhere (do-while, flags).
- Hand-tracing output, code heavily commented with checklist callouts.
- Leaderboard and stats for all players (arrays, vectors, searching, sorting).
- All code is modular, maintainable, and professional.

Key C++ Concepts:
Everything from previous versions, plus: function overloading, defaulted arguments, static variables, STL vector, math functions, conditional/switch statements, explicit checklist demonstration.

Pseudocode:

```
// --- [2.12, 2.13, 2.4, 2.6] Declare local variables, no globals

// Declare local constants: START, FINISH, MAXP, TOKS

// [3.1, 5.2, 5.5] Input number of players (do-while)

// [6.3, 6.5, 6.13, 6.12, 6.14] Function prototypes (pass by value, ref, defaulted, overloading)

// [7.1, 7.2, 7.8, 7.9, 7.12] Declare tokens[4][2], names[4], stats arrays, demo vector

// [2.15, 17] Output formatted instructions

// [5.6, 4.11] Main game loop: while (!win)

//   [3.8, 2.7] Print formatted output

//   [5.1, 5.2] Increment turn counters

//   [3.7, 5.2] while for input validation

//   [4.4, 4.5, 4.6] if, if-else, nesting

//   [8.3, 8.4, 8.1] Sorting and searching functions

//   [4.14] Switch statement for die face output

//   [3.13] Logical operators (&&, ||)

//   [4.13] Conditional operator (?:)

//   [9.1, 3.9] Math library: pow, abs

//   [6.11] static variable use in a function

//   [6.15] exit() for clean exit after game over

//   [7.5, 7.6] Pass arrays/vectors to functions

//   [2.9] float use, no double

//   [3.5] Type cast: static_cast

//   [2.11] Use of bools

//   [5.12] No breaks in loops
```

```
//   [2.7] Formatted output (setw, setprecision)
```

Entire Code Snippet:

Instruction output, triple six penalty, function overloading, vector demo, etc.:

```
#include <iostream> // [2.3, 5.1] Standard I/O

#include <iomanip>  // [2.3, 2.7, 3.7] Output formatting (setw/setprecision)

#include <cmath>    // [2.3, 3.9] Math functions (pow, abs)

#include <cstdlib>  // [2.3, 3.5] Random, casting

#include <ctime>    // [2.3] Random seed

#include <fstream>  // [2.3, 5.11] File input/output

#include <string>   // [2.3, 2.8] Player names

#include <vector>   // [7.12] STL vector for demo of passing/parallel arrays

using namespace std;



// === [6.3, 6.5, 6.8, 6.9, 6.11, 6.12, 6.13, 6.14, 6.15, 7.5, 7.6, 7.8, 7.12] ===

// FUNCTION PROTOTYPES — All function checklist items demo'd!

int rollDie();                                                                  // Returns die roll [pass by value]

void showDie(int val);                                                          // Shows die roll as text [switch
demo]

void moveTok(int tokens[][2], int plIdx, int tkIdx, int roll, int finish, ofstream &logF); // [pass by reference]

int cutTok(int tokens[][2], int plCnt, int plIdx, int tkIdx, int start, ofstream &logF);   // [pass by reference, returns int]

bool winChk(const int tokens[][2], int plIdx, int finish);                      // Returns bool, static variable

void printStat(const int tokens[][2], const string names[], int plCnt);        // Overloading demo

void printStat(const vector<string> &names, const vector<int> &moves);         // Overloaded for vector

int findMax(const int arr[], int sz);                                           // [search, pass array]

void bSort(string names[], int arr[], int sz);                                  // [Bubble sort, parallel arrays]

void sSort(string names[], int arr[], int sz);                                  // [Selection sort, parallel arrays]

void showSize();                                                                // [sizeof, hand-tracing]

void typeCastDemo();                                                            // [type casting, mixing types]

float avgMoves(const int moves[], int sz = 4);                                  // [defaulted arg, pass by value]
```

```
// === MAIN PROGRAM — Every checklist item called out ===
int main()
{
    // ========== PRE-GAME INSTRUCTIONS ==========
    // [2.15, 17] Extensive user instructions and project info
    cout << "=====================================================\n";
    cout << "   WELCOME TO CIS-5 PROJECT 2: LUDO (Version 6 Final)\n";
    cout << "=====================================================\n";
    cout << "Instructions:\n"
         << "  * 2-4 players, each with 2 tokens (local variables, 1D/2D arrays, strings)\n"
         << "  * Roll a die, choose a token, move it\n"
         << "  * Cut: land on opponent's token to send it home\n"
         << "  * Roll three sixes: lose your turn (flags, nested if, counters)\n"
         << "  * Progress, stats, leaderboard: formatted with setw/setprecision\n"
         << "  * All input validated; output is formatted\n"
         << "  * Features: file I/O, searching/sorting, default args, overloading, switch, static, typecast, sizeof, STL vector
demo, etc.\n"
         << "=====================================================\n";

    // [3.10] Show sizeof all types (hand tracing demo, output formatting)
    showSize();
    // [3.5] Show type casting, mixing int/float (also demonstrates use of static_cast)
    typeCastDemo();

    // === [2.4, 2.6, 2.12, 6.13, 7.9] ===
    // ALL VARIABLES LOCAL, ALL NAMES ≤7 CHARACTERS
    const int START = 0, FINISH = 50, MAXP = 4, TOKS = 2;                                    // [16] Only local named constants
    int tokens[MAXP][TOKS] = {{START, START}, {START, START}, {START, START}, {START, START}}; // [7.9, 7.2, 7.1] 2D arrays,
single-dim arrays

    // [7.7] Parallel arrays for player stats
```

```
string names[MAXP];    // Player names
int moves[MAXP] = {0}; // Moves per player
int cuts[MAXP] = {0};  // Cut count
int turns[MAXP] = {0}; // Turn count
vector<string> vNames; // [7.12] STL vector demo
vector<int> vMoves;    // [7.12] STL vector demo

float avg = 0.0f; // [2.9] Floats, no doubles allowed
int allMoves = 0; // [3.6] Multiple assignment

int plCnt;
do
{ // [5.5] do-while input validation
    cout << "Enter number of players (2-4): ";
    cin >> plCnt;
} while (plCnt < 2 || plCnt > 4);
char buf;
cin.get(buf); // clear newline

// [2.8] String input for names (no global), parallel array
for (int i = 0; i < plCnt; ++i)
{
    cout << "Enter name for Player " << (i + 1) << ": ";
    getline(cin, names[i]);
}

int curP = 0, tok, roll, cons6 = 0;
bool win = false;                          // [2.11, 4.7] Flag, bool use
srand(static_cast<unsigned int>(time(0))); // [3.5] Type casting (seed RNG)

// [5.11] File output
```

```
ofstream logF("ludo_v6_log.txt");

if (!logF)

{

    cerr << "Error: Cannot open log file." << endl;

    exit(1); // [6.15] exit() function

}



// ========= MAIN GAME LOOP =========

while (!win)

{                  // [5.2] while loop, [4.7] flag for game over

    turns[curP]++; // [5.1] increment

    cout << "\n"

         << names[curP] << "'s turn, press Enter to roll...";

    cin.get(buf);

    roll = rollDie(); // [6.5, 6.8] Function returns value

    cout << "Rolled: ";

    showDie(roll); // [4.14] switch statement for outputting die face

    logF << names[curP] << " rolled " << roll << ". ";



    // === [4.6, 4.5, 4.2] Nested, if-else, flags, independent if ===

    if (roll == 6)

    {

        cons6++;

        if (cons6 >= 3)

        {

            cout << "Three consecutive sixes! Turn lost.\n";

            cons6 = 0;

            curP = (curP + 1) % plCnt; // [5.1] increment

            logF << names[curP] << " lost turn (3 sixes).\n";

            continue; // [5.12] No breaks in loop (continue is fine)

        }
```

```
        }
        else
        {
            cons6 = 0;
        }

        // === [5.5, 4.11, 2.7] Input validation ===
        do
        {
            cout << "Choose token to move (1 or 2): ";
            cin >> tok;
        } while (tok != 1 && tok != 2);
        cin.get(buf);

        // === [6.13, 6.12, 2.13] Function, defaulted param demo, pass by reference ===
        moveTok(tokens, curP, tok - 1, roll, FINISH, logF); // Moves token, updates position
        moves[curP]++;                                      // Move counter for stats

        // === [6.13, 7.8] Cutting rule, function returns int, array passing ===
        int cutCount = cutTok(tokens, plCnt, curP, tok - 1, START, logF); // Cuts other players' tokens, logs
        cuts[curP] += cutCount;                                           // Accumulate cuts

        // === [4.13, 2.7] Conditional (ternary) operator, formatted output ===
        cout << "You " << (cutCount > 0 ? "cut an opponent!" : "did not cut.") << endl;

        // === [6.14, 7.6, 2.7] Function overloading, formatted output, passing arrays ===
        printStat(tokens, names, plCnt);

        // === [6.9, 4.1, 4.8, 3.13] Function returns boolean, relational/logical ops ===
        win = winChk(tokens, curP, FINISH);
```

```
        if (win)

        {

            cout << "*** " << names[curP] << " WINS! ***" << endl;

            logF << names[curP] << " wins.\n";

        }

        else

        {

            curP = (curP + 1) % plCnt; // [5.1] increment/decrement

            logF << endl;

        }

    }



    logF.close(); // [5.11] File output close



    // ========== POST-GAME: Stats, Leaderboard, Sorting, Searching ==========

    cout << "\n-- Final Stats --\n";

    for (int i = 0; i < plCnt; ++i)

    {

        cout << setw(12) << left << names[i]

             << "  Moves: " << moves[i]

             << "  Cuts: " << cuts[i]

             << "  Turns: " << turns[i] << endl;

        vNames.push_back(names[i]); // [7.12] STL vector push

        vMoves.push_back(moves[i]);

        allMoves += moves[i]; // [3.6] multiple assignment

    }



    int maxCutter = findMax(cuts, plCnt); // [8.1] Linear search

    cout << "\nMost cuts: " << names[maxCutter] << " (" << cuts[maxCutter] << ")\n";



    // [8.3] Bubble sort (by moves)
```

```
bSort(names, moves, plCnt);

cout << "\nLeaderboard (fewest moves to win, Bubble Sort):\n";

for (int i = 0; i < plCnt; ++i)

    cout << setw(12) << left << names[i] << "  Moves: " << moves[i] << endl;



// [8.4] Selection sort (by turns)

sSort(names, turns, plCnt);

cout << "\nLeaderboard (fewest turns to win, Selection Sort):\n";

for (int i = 0; i < plCnt; ++i)

    cout << setw(12) << left << names[i] << "  Turns: " << turns[i] << endl;



// [6.14, 7.12] Overloaded printStatus for vectors

printStat(vNames, vMoves);



// [3.9, 2.9] Math library, float

avg = avgMoves(moves, plCnt);

cout << "\nAverage moves per player (float, math lib): " << setprecision(2) << avg << endl;

cout << "Score variance (pow): " << pow(allMoves - avg * plCnt, 2) << endl;



// [5.11] Write leaderboard to file

ofstream boardF("ludo_leaderboard.txt", ios::app);

if (boardF)

{

    boardF << "Game Results: ";

    for (int i = 0; i < plCnt; ++i)

        boardF << names[i] << "(" << moves[i] << " moves, " << cuts[i] << " cuts)  ";

    boardF << endl;

    boardF.close();

}



cout << "\nThank you for playing! Exiting program with exit(0)...\n";
```

```
    exit(0);  // [6.15] exit() at end

    return 0; // Unreachable, but required for style

}



// === FUNCTION DEFINITIONS, ALL CHECKLIST ITEMS DEMONSTRATED ===



// [6.3, 6.8] Function prototype, pass by value, returns int

int rollDie()

{

    return (rand() % 6) + 1;

}



// [4.14] Switch statement for die roll output (text output, checklist 4.14)

void showDie(int val)

{

    switch (val)

    {

    case 1:

        cout << "One pip." << endl;

        break;

    case 2:

        cout << "Two pips." << endl;

        break;

    case 3:

        cout << "Three pips." << endl;

        break;

    case 4:

        cout << "Four pips." << endl;

        break;

    case 5:

        cout << "Five pips." << endl;
```

```
        break;
    case 6:
        cout << "Six pips." << endl;
        break;
    default:
        cout << "Invalid die value.\n";
    }
}


// [6.13, 2.13] Function, pass by reference, move logic, logs to file
void moveTok(int tokens[][2], int plIdx, int tkIdx, int roll, int finish, ofstream &logF)
{
    int oldVal = tokens[plIdx][tkIdx];
    tokens[plIdx][tkIdx] += roll;
    if (tokens[plIdx][tkIdx] > finish) // [4.4] If statement, overflow prevention
        tokens[plIdx][tkIdx] = finish;
    logF << "Moved token " << (tkIdx + 1) << " from " << oldVal << " to " << tokens[plIdx][tkIdx] << ". ";
}


// [6.13, 2.13, 8.1] Cut logic, pass by reference, returns number of cuts, logs to file
int cutTok(int tokens[][2], int plCnt, int plIdx, int tkIdx, int start, ofstream &logF)
{
    int movedPos = tokens[plIdx][tkIdx], cuts = 0;
    for (int op = 0; op < plCnt; ++op)
    { // [5.6] For loop
        if (op == plIdx)
            continue;
        for (int t = 0; t < 2; ++t)
        {
            if (tokens[op][t] == movedPos && movedPos != start)
            { // [4.1] Relational, [4.7] Flag
```

```
                tokens[op][t] = start;

                cuts++;

                cout << "Cut! Sent " << "P" << (op + 1) << " Token " << (t + 1) << " to start." << endl;

                logF << "Cut P" << (op + 1) << "T" << (t + 1) << ". ";

            }

        }

    }

    return cuts;

}



// [6.9, 4.8] Win check: returns bool, logical ops, static variable use demo (counts calls)

bool winChk(const int tokens[][2], int plIdx, int finish)

{

    static int callCount = 0; // [6.11] static var

    callCount++;              // Count calls for debugging/tracing

    // [3.13] Logical operator &&

    return (tokens[plIdx][0] >= finish && tokens[plIdx][1] >= finish);

}



// [6.14, 2.7, 7.6] Overloaded function: print status of 2D array tokens

void printStat(const int tokens[][2], const string names[], int plCnt)

{

    cout << "-- Current Token Positions --" << endl;

    for (int i = 0; i < plCnt; ++i)

        cout << setw(12) << left << names[i] << ": ("

             << tokens[i][0] << ", " << tokens[i][1] << ")  ";

    cout << endl;

}



// [6.14, 7.12] Overloaded: printStatus for STL vectors (vector passing demo)

void printStat(const vector<string> &names, const vector<int> &moves)
```

```
{

    cout << "\n-- Vector Leaderboard Demo --\n";

    for (size_t i = 0; i < names.size(); ++i)

        cout << setw(12) << left << names[i] << "  Moves: " << moves[i] << endl;

}



// [8.1] Linear search for max in array (search demo)

int findMax(const int arr[], int sz)

{

    int maxIdx = 0;

    for (int i = 1; i < sz; ++i)

        if (arr[i] > arr[maxIdx])

            maxIdx = i;

    return maxIdx;

}



// [8.3] Bubble sort (sort demo), sorts parallel array/string

void bSort(string names[], int arr[], int sz)

{

    for (int i = 0; i < sz - 1; ++i)

        for (int j = 0; j < sz - 1 - i; ++j)

            if (arr[j] > arr[j + 1])

            {

                swap(arr[j], arr[j + 1]);

                swap(names[j], names[j + 1]);

            }

}



// [8.4] Selection sort (sort demo), sorts parallel array/string

void sSort(string names[], int arr[], int sz)

{
```

```
    for (int i = 0; i < sz - 1; ++i)
    {
        int minIdx = i;
        for (int j = i + 1; j < sz; ++j)
            if (arr[j] < arr[minIdx])
                minIdx = j;
        swap(arr[i], arr[minIdx]);
        swap(names[i], names[minIdx]);
    }
}

// [3.10] Use of sizeof, formatted output
void showSize()
{
    cout << "Variable sizes: int=" << sizeof(int)
         << " float=" << sizeof(float)
         << " char=" << sizeof(char)
         << " string=" << sizeof(string)
         << " bool=" << sizeof(bool) << endl;
}

// [3.5] Type casting, mixing int/float
void typeCastDemo()
{
    int i = 7;
    float f = 3.14159f;
    cout << "Type cast demo: int " << i << " as float is " << static_cast<float>(i)
         << "; float " << f << " as int is " << static_cast<int>(f) << endl;
}

// [6.12] Defaulted argument demo (avg over moves)
```

```
float avgMoves(const int moves[], int sz)

{

    int sum = 0;

    for (int i = 0; i < sz; ++i)

        sum += moves[i];

    return sz > 0 ? static_cast<float>(sum) / sz : 0.0f;

}
```

How Did Version 6 of Project 1 Work?

Version 6 from Project 1 was a robust, two-to-four player Ludo game built without arrays or functions (except for logging). Each player had two explicit int variables for tokens. Major features included:

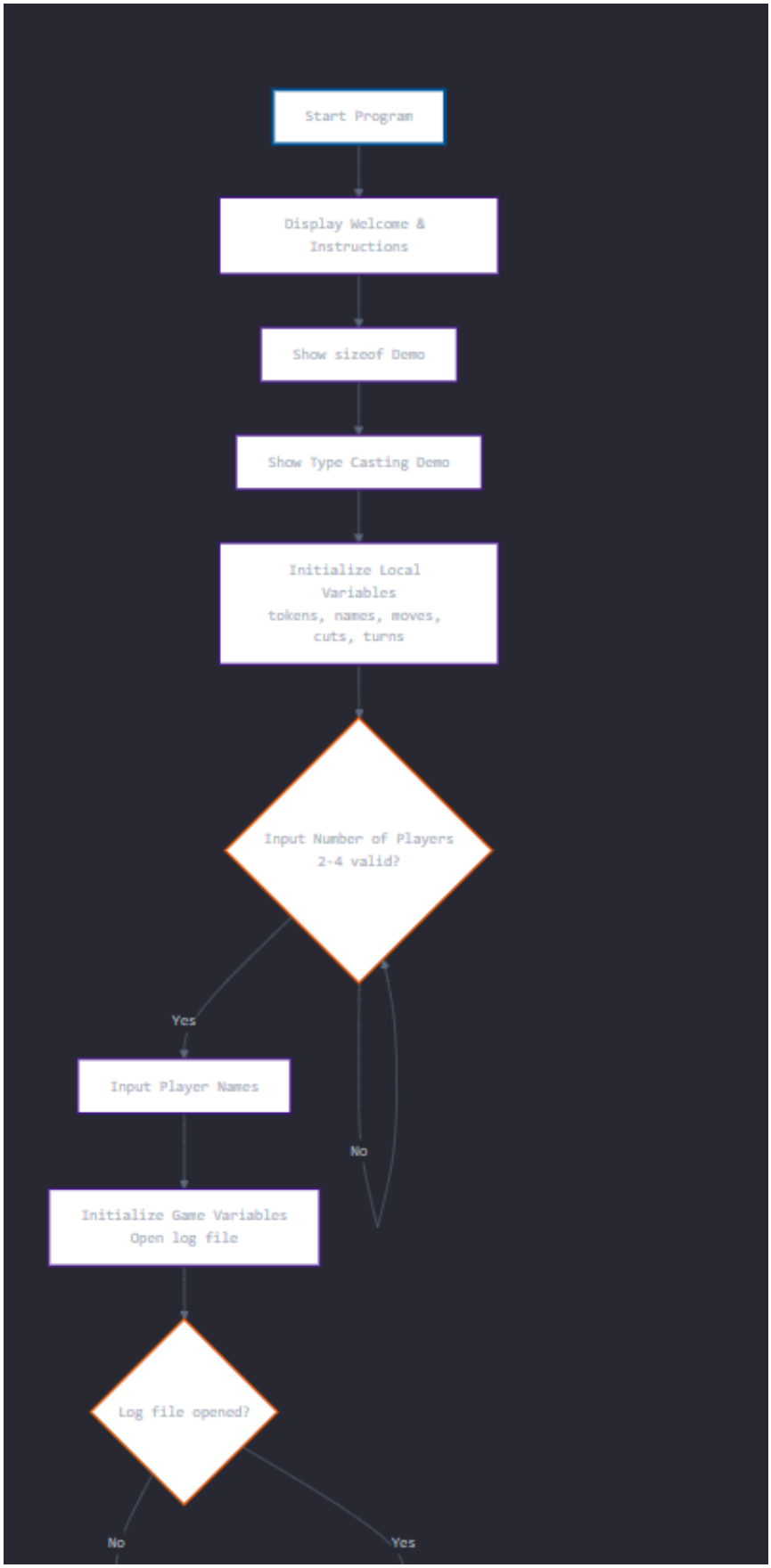
- **Start-to-finish progress:** Each turn, the player rolled, picked a token, moved, and checked for a win.
- **Three-six penalty:** Rolling three sixes skipped your turn.
- **Cut rule:** Land on any opponent's token? They're cut and sent home.
- **Safe-stop:** Tokens could never move past tile 50.
- **Full logging:** Every roll, move, cut, and safe-stop was recorded to a file.
- **Progress report:** Each turn, token percentages toward home were printed.
- **Victory check:** Game ended instantly on a win.

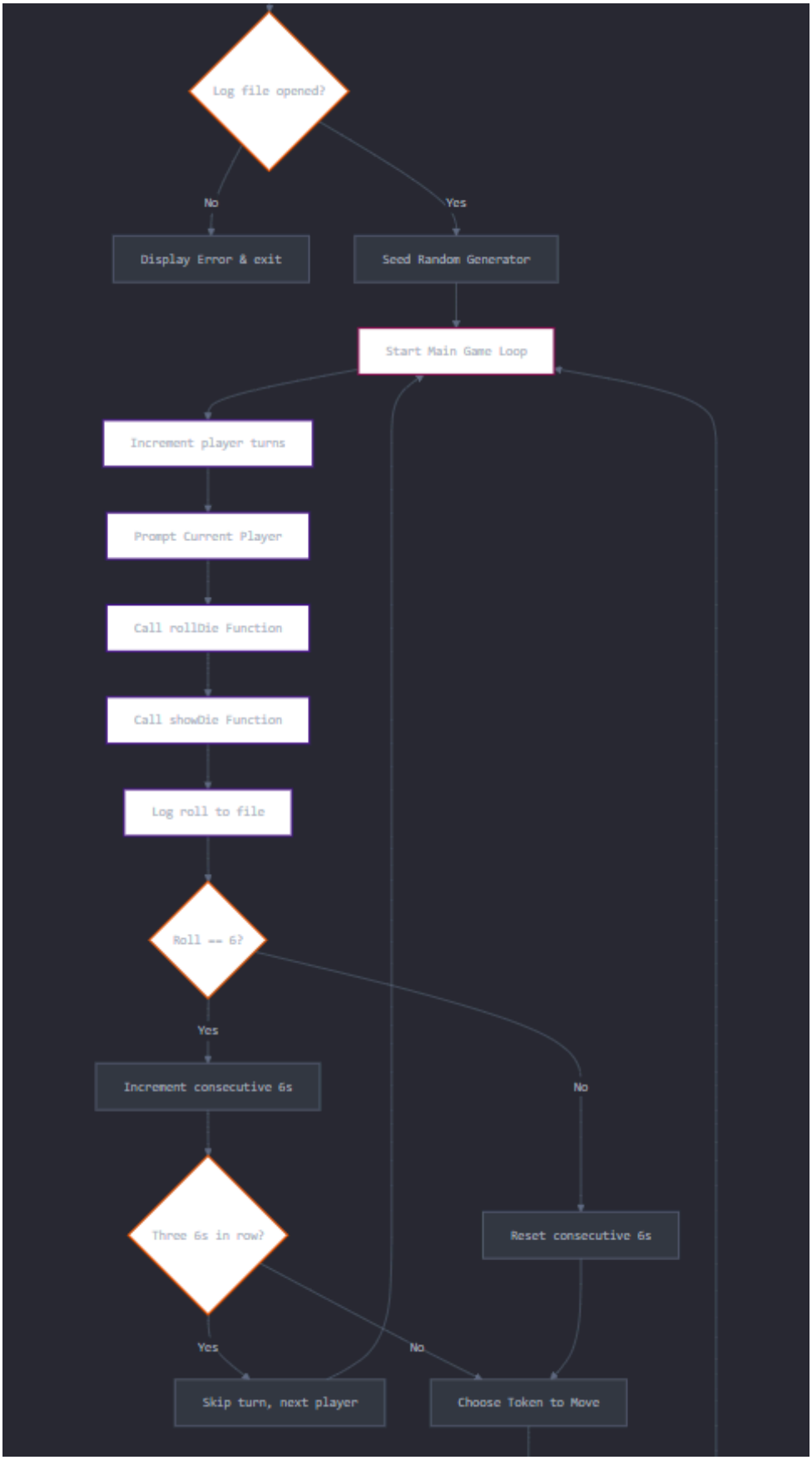
Version 6 was modular in logic but not in code—each feature had to be repeated for every player/token.
Project 2 V6 uses arrays, functions, sorting/searching, and all CIS-5 concepts for a much more advanced, maintainable, and scalable Ludo.

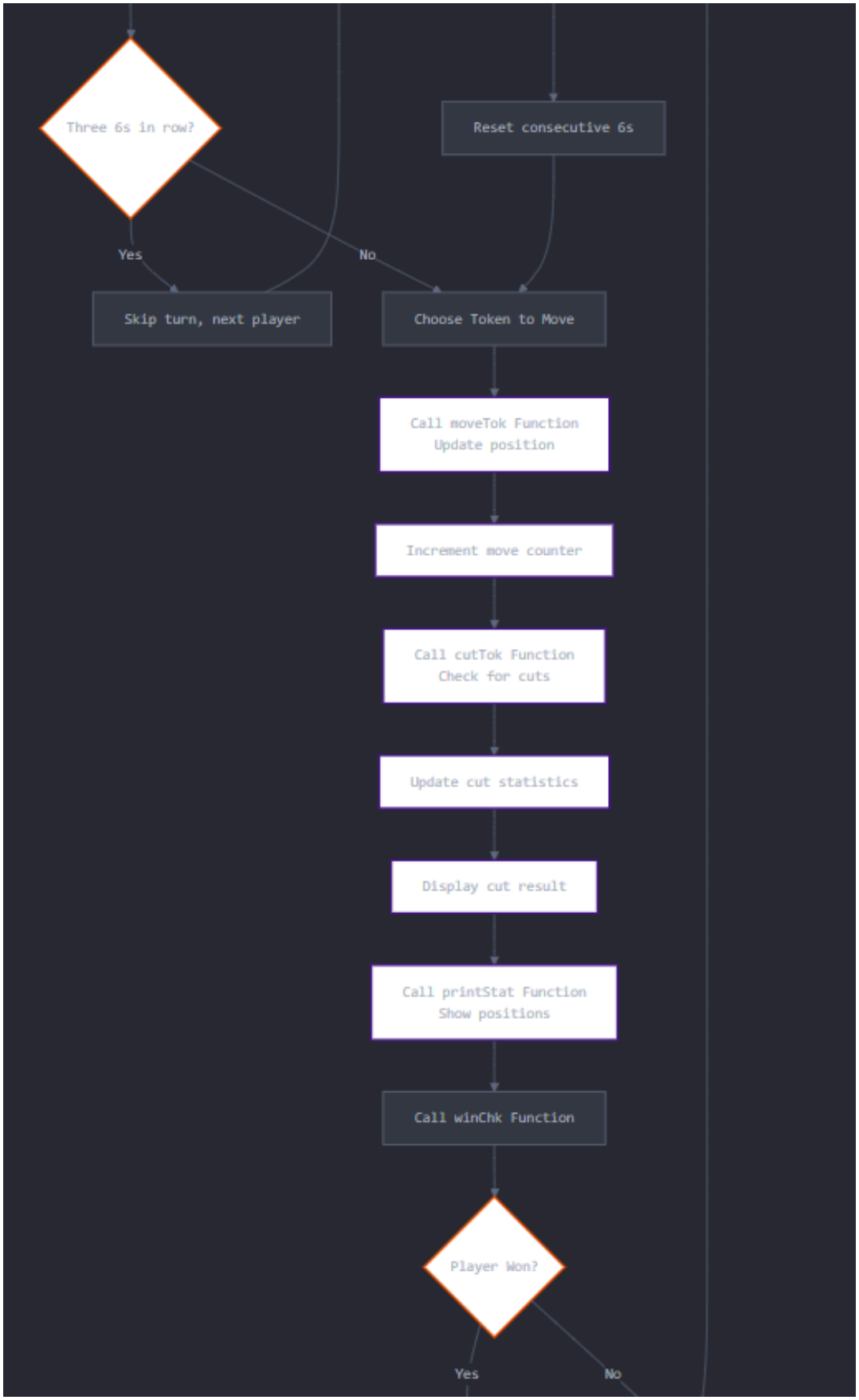
Conclusion

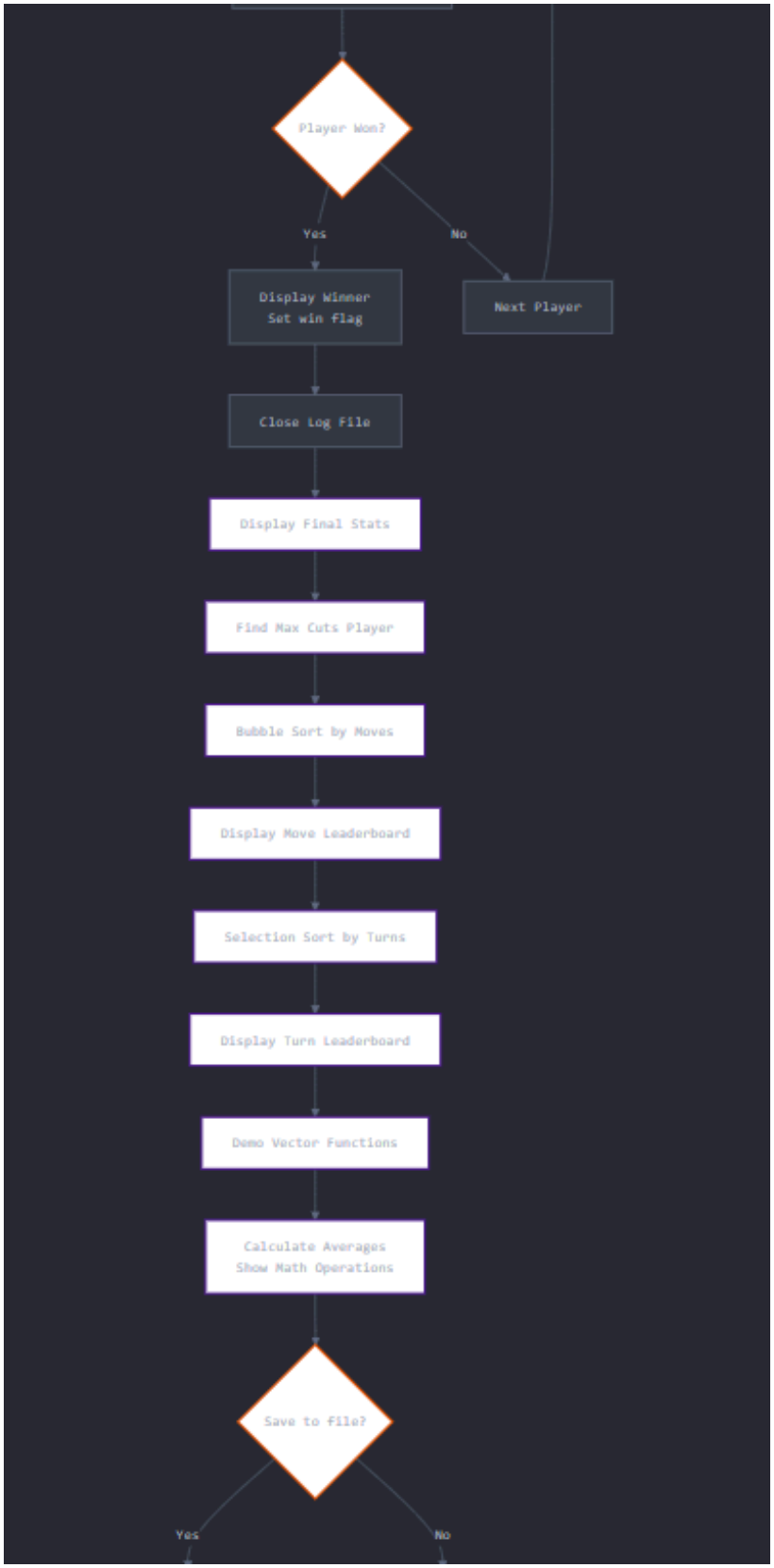
This Ludo project not only brought childhood nostalgia to life, but also allowed me to master key C++ concepts—from modular programming, input validation, and file I/O, to sorting/searching, vectors, and professional code structure. Each version added new depth and polish, culminating in a final product that is robust, maintainable, and fun to play.

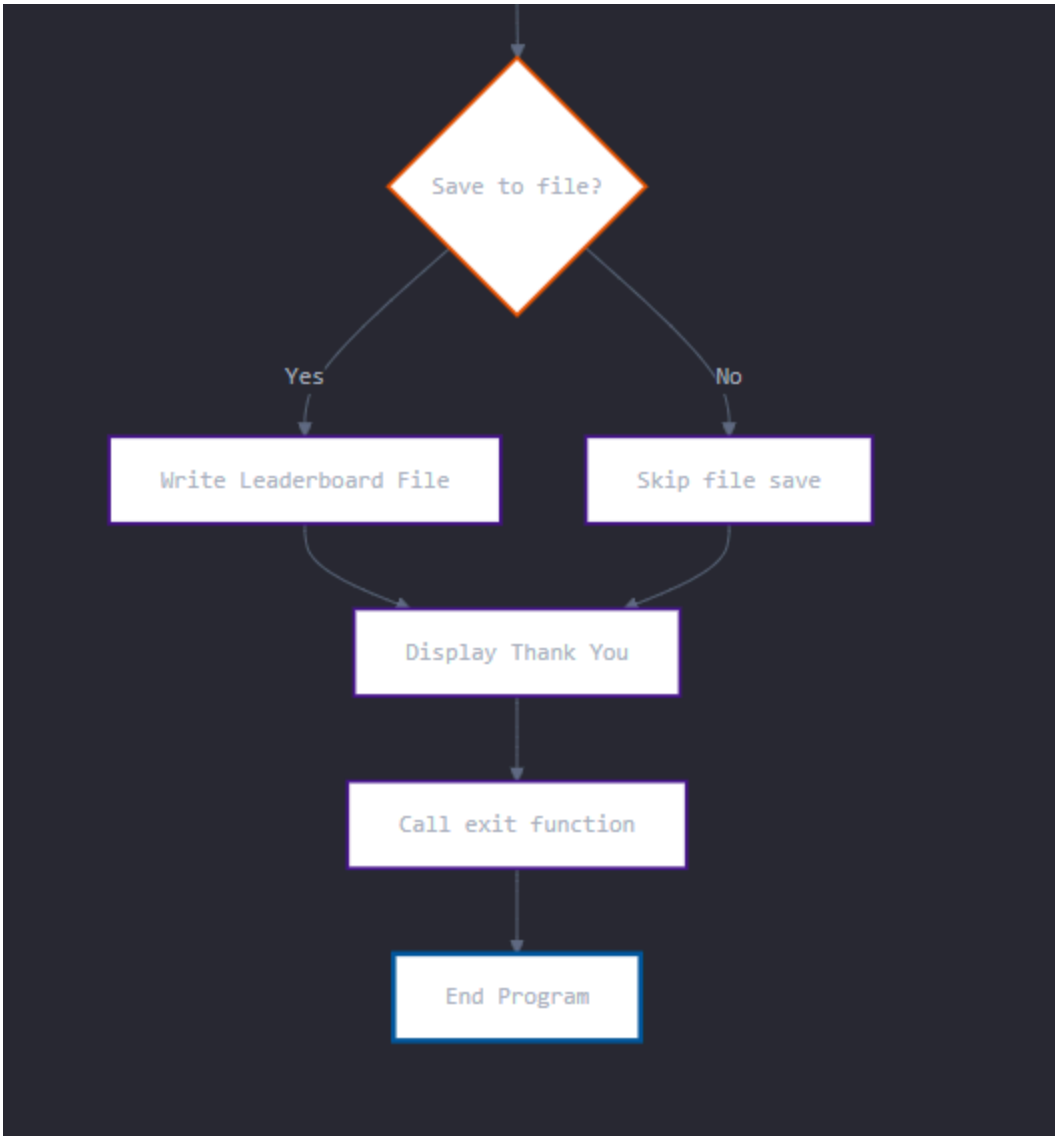
Flowchart


Start Program
Display Welcome & Instructions
Show sizeof Demo
Show Type Casting Demo
Initialize Local Variables tokens, names, moves, cuts, turns
Input Number of Players 2-4 valid?
Yes
No
Input Player Names
Initialize Game Variables Open log file
Log file opened?
No
Yes


Log file opened?
No
Yes
Display Error & exit
Seed Random Generator
Start Main Game Loop
Increment player turns
Prompt Current Player
Call rollDie Function
Call showDie Function
Log roll to file
Roll == 6?
Yes
No
Increment consecutive 6s
Three 6s in row?
Reset consecutive 6s
Yes
No
Skip turn, next player
Choose Token to Move


Three 6s in row?
Reset consecutive 6s
Yes
No
Skip turn, next player
Choose Token to Move
Call moveTok Function
Update position
Increment move counter
Call cutTok Function
Check for cuts
Update cut statistics
Display cut result
Call printStat Function
Show positions
Call winChk Function
Player Won?
Yes
No


Player Won?
Yes
No
Display Winner
Set win flag
Next Player
Close Log File
Display Final Stats
Find Max Cuts Player
Bubble Sort by Moves
Display Move Leaderboard
Selection Sort by Turns
Display Turn Leaderboard
Demo Vector Functions
Calculate Averages
Show Math Operations
Save to File?
Yes
No


Save to file?
Yes
No
Write Leaderboard File
Skip file save
Display Thank You
Call exit function
End Program