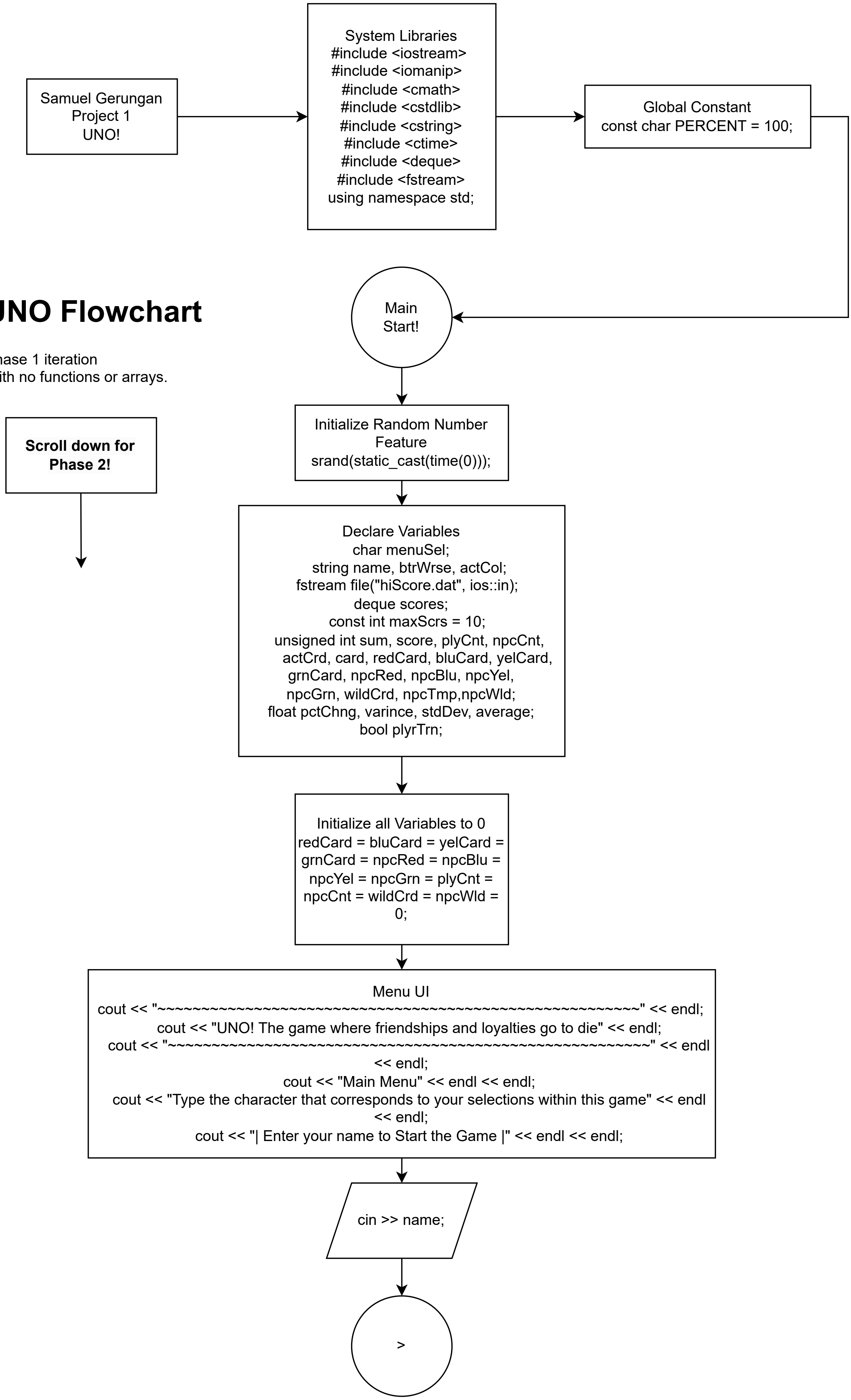
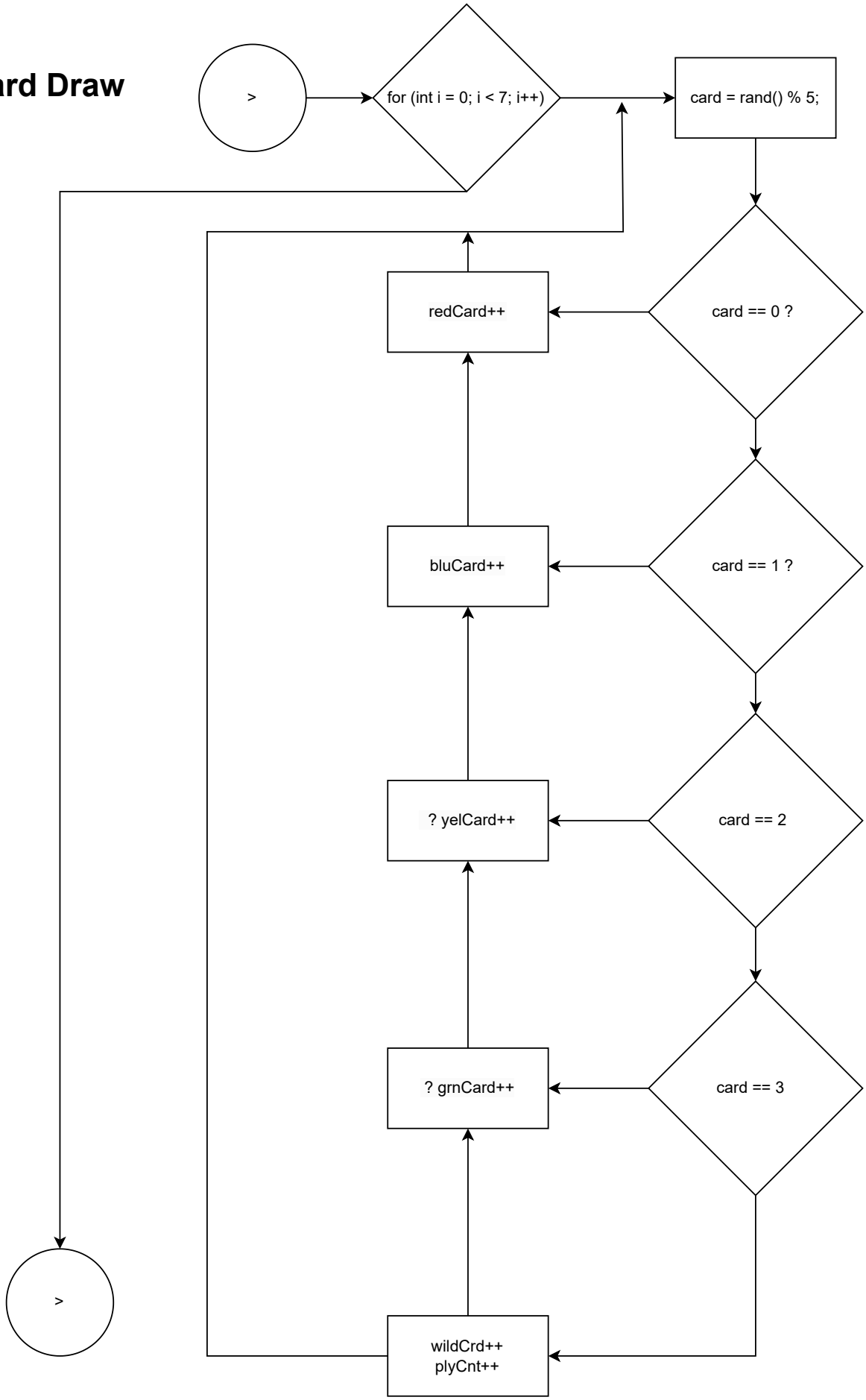
UNO Flowchart

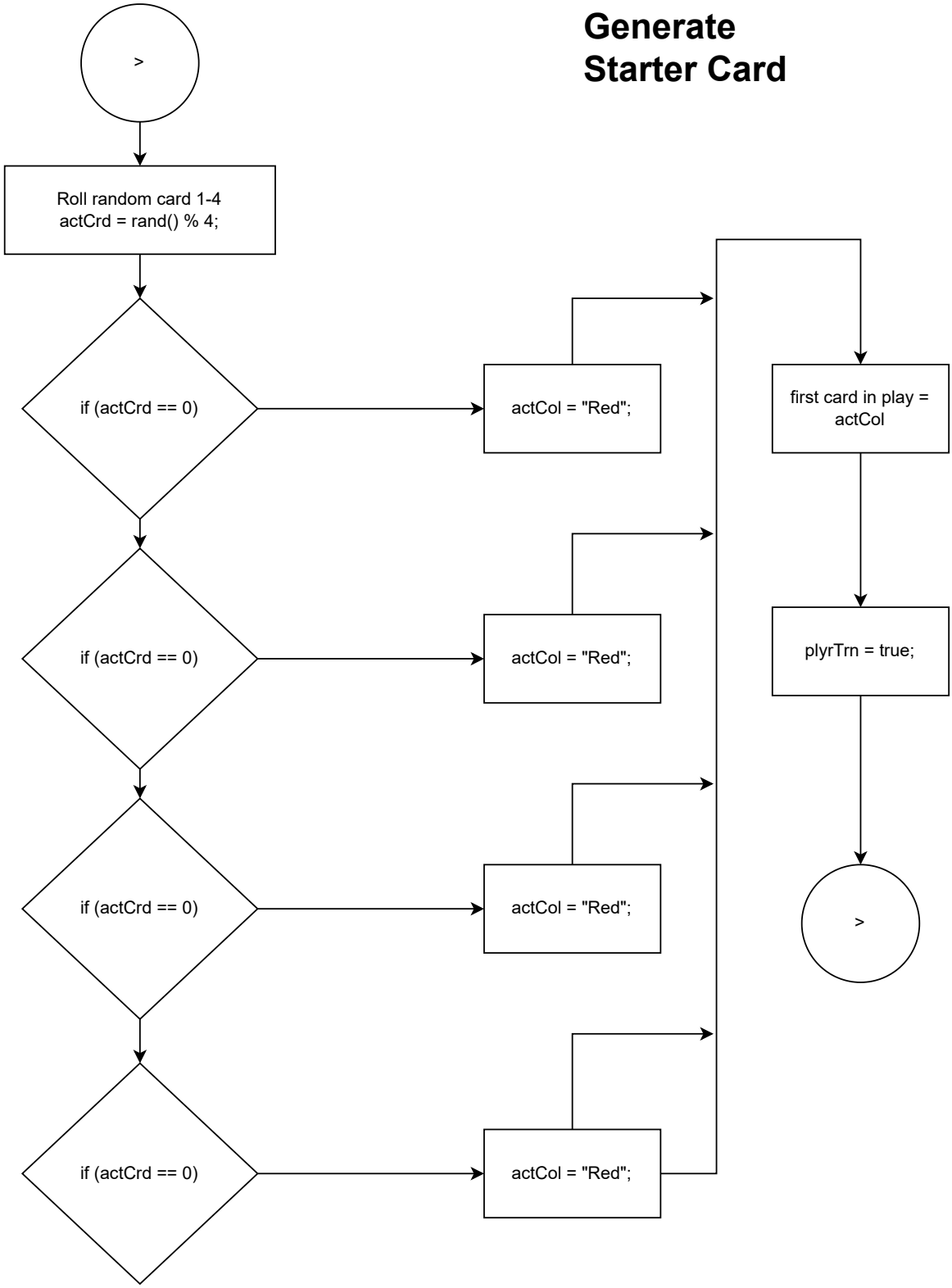
Phase 1 iteration
With no functions or arrays.



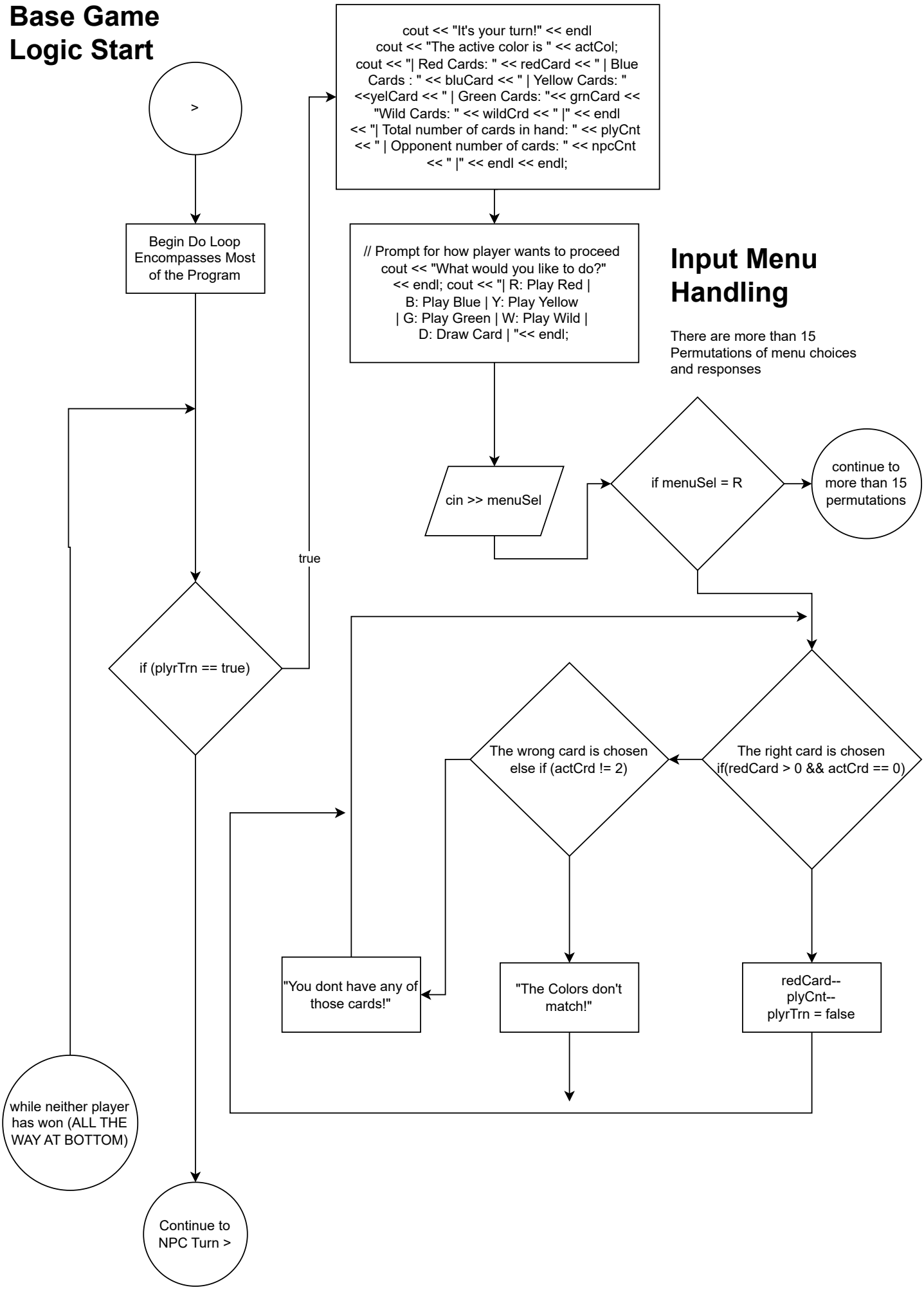
Card Draw


>
for (int i = 0; i < 7; i++)
card = rand() % 5;
card == 0 ?
redCard++
card == 1 ?
bluCard++
card == 2
? yelCard++
card == 3
? grnCard++
wildCrd++
plyCnt++
>

Generate Starter Card

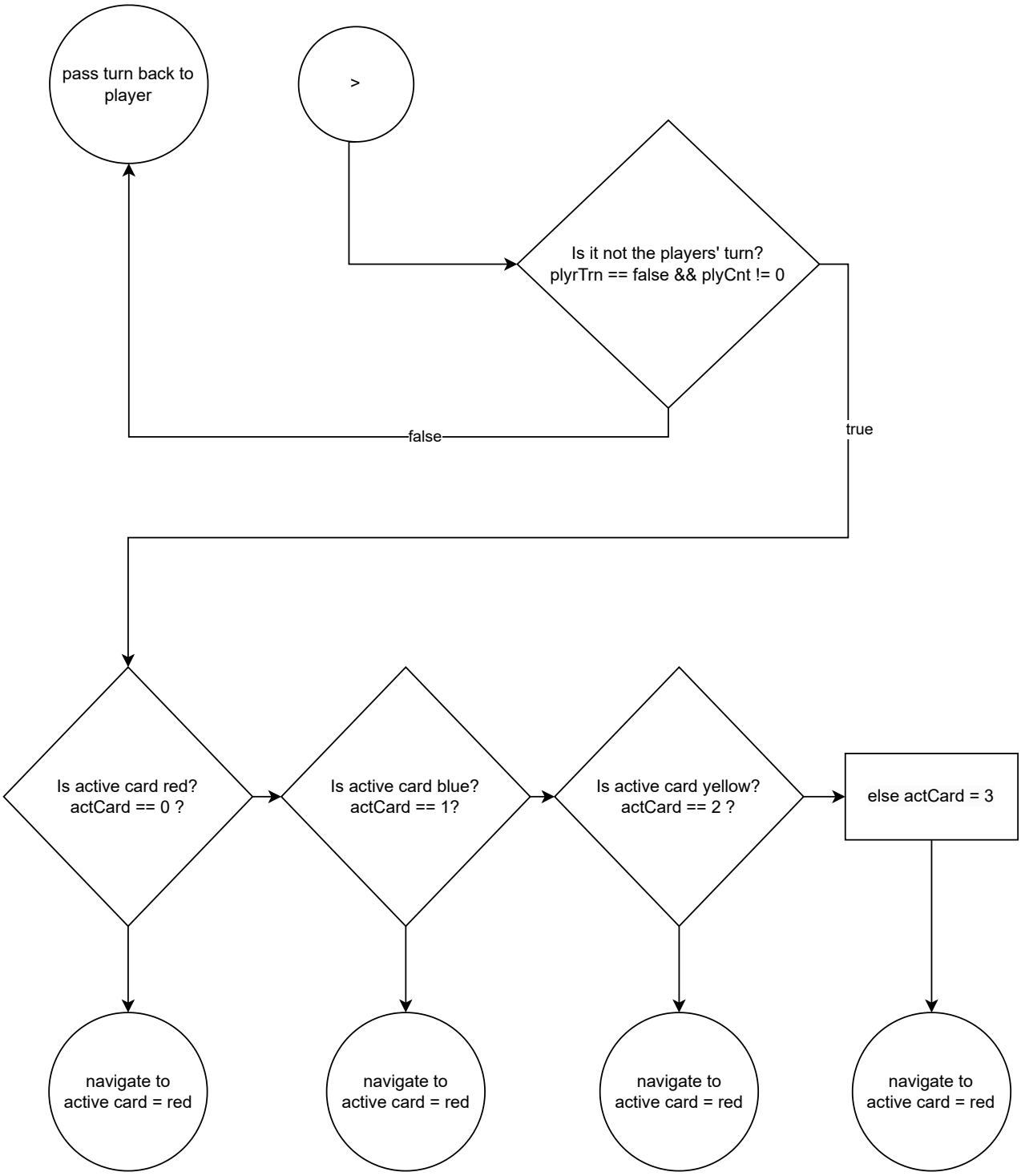


Base Game Logic Start



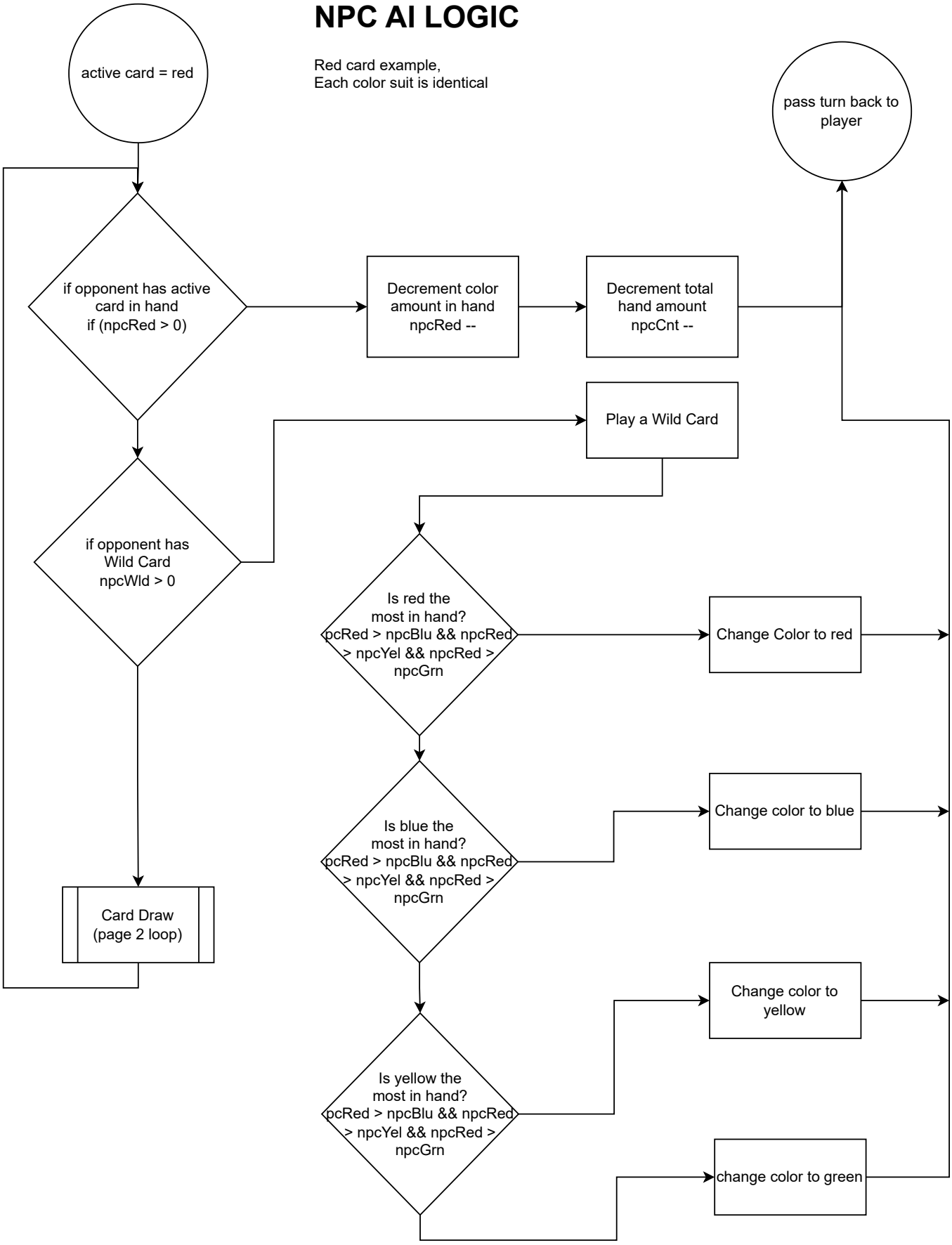
NPC AI LOGIC

Decision tree to branch based off of color in play



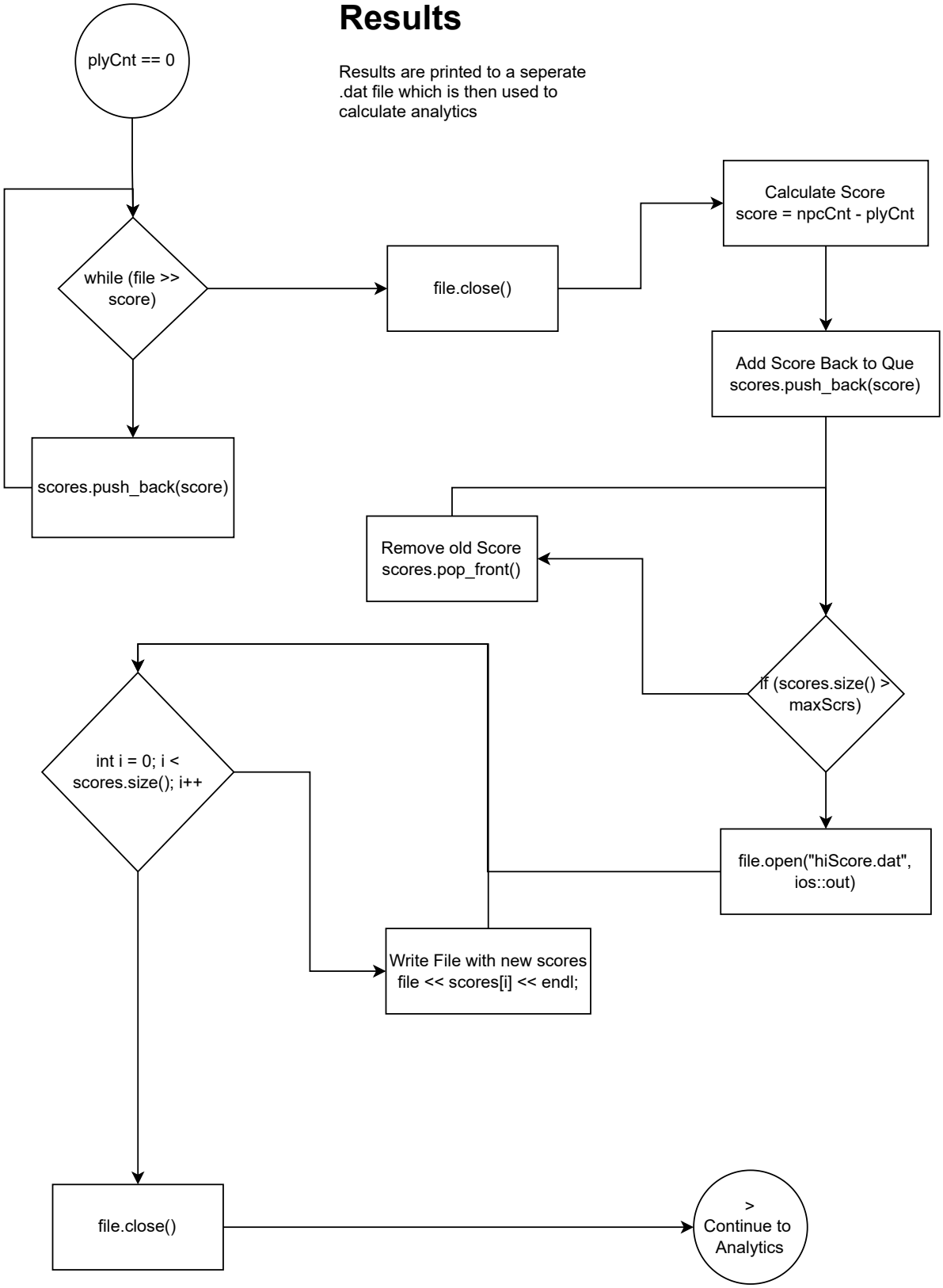
NPC AI LOGIC

Red card example,
Each color suit is identical



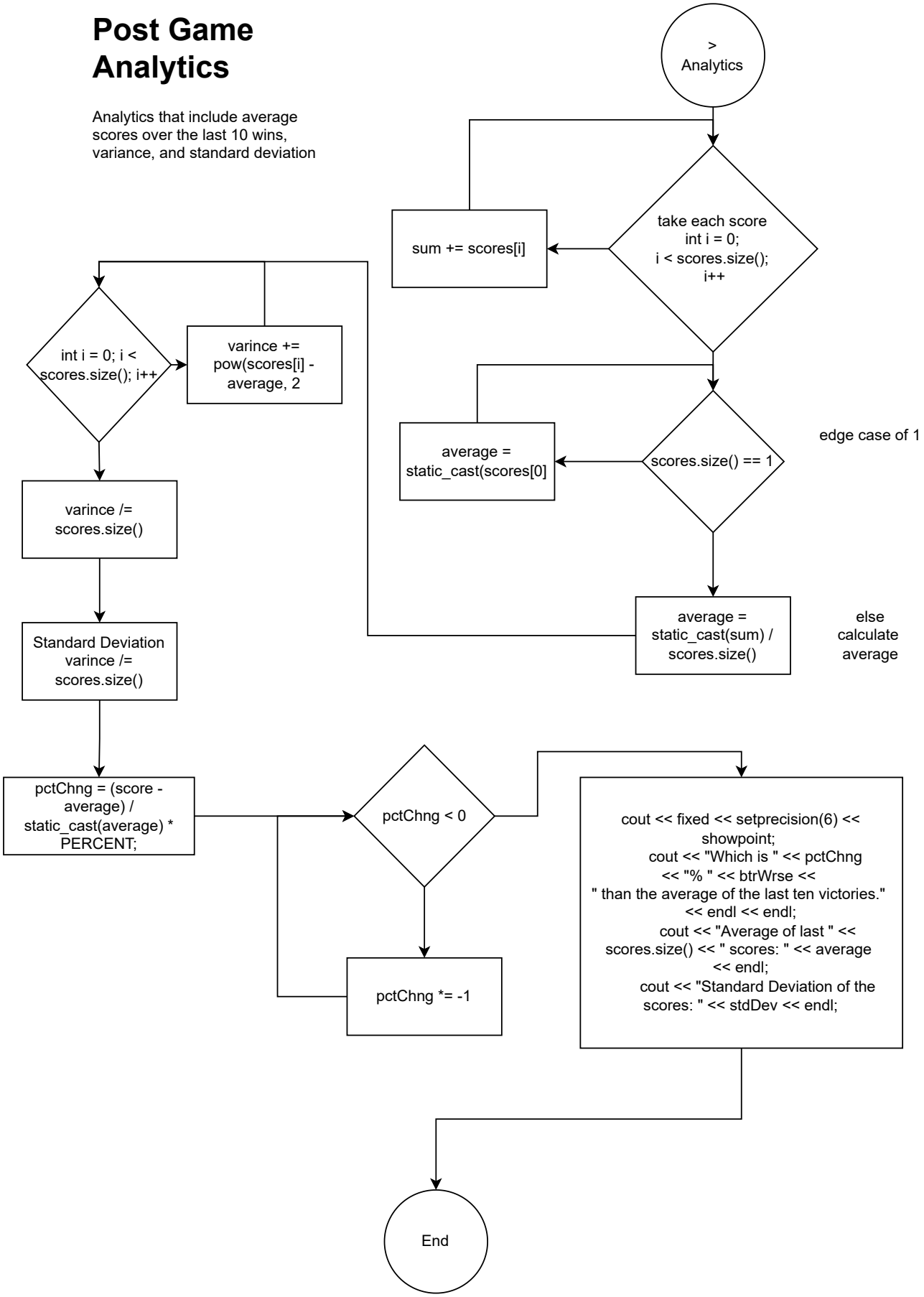
Printing Results

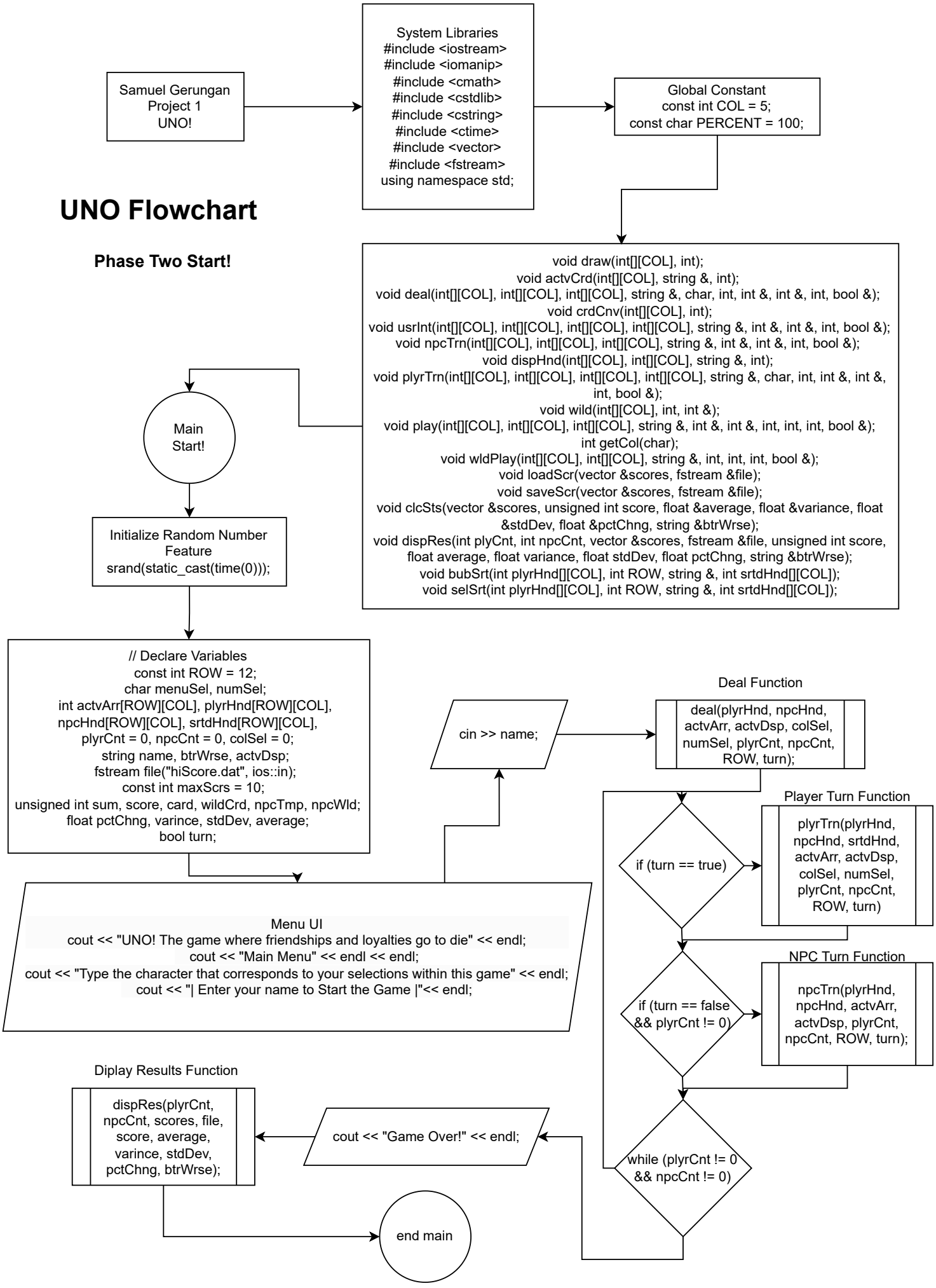
Results are printed to a seperate .dat file which is then used to calculate analytics



Post Game Analytics

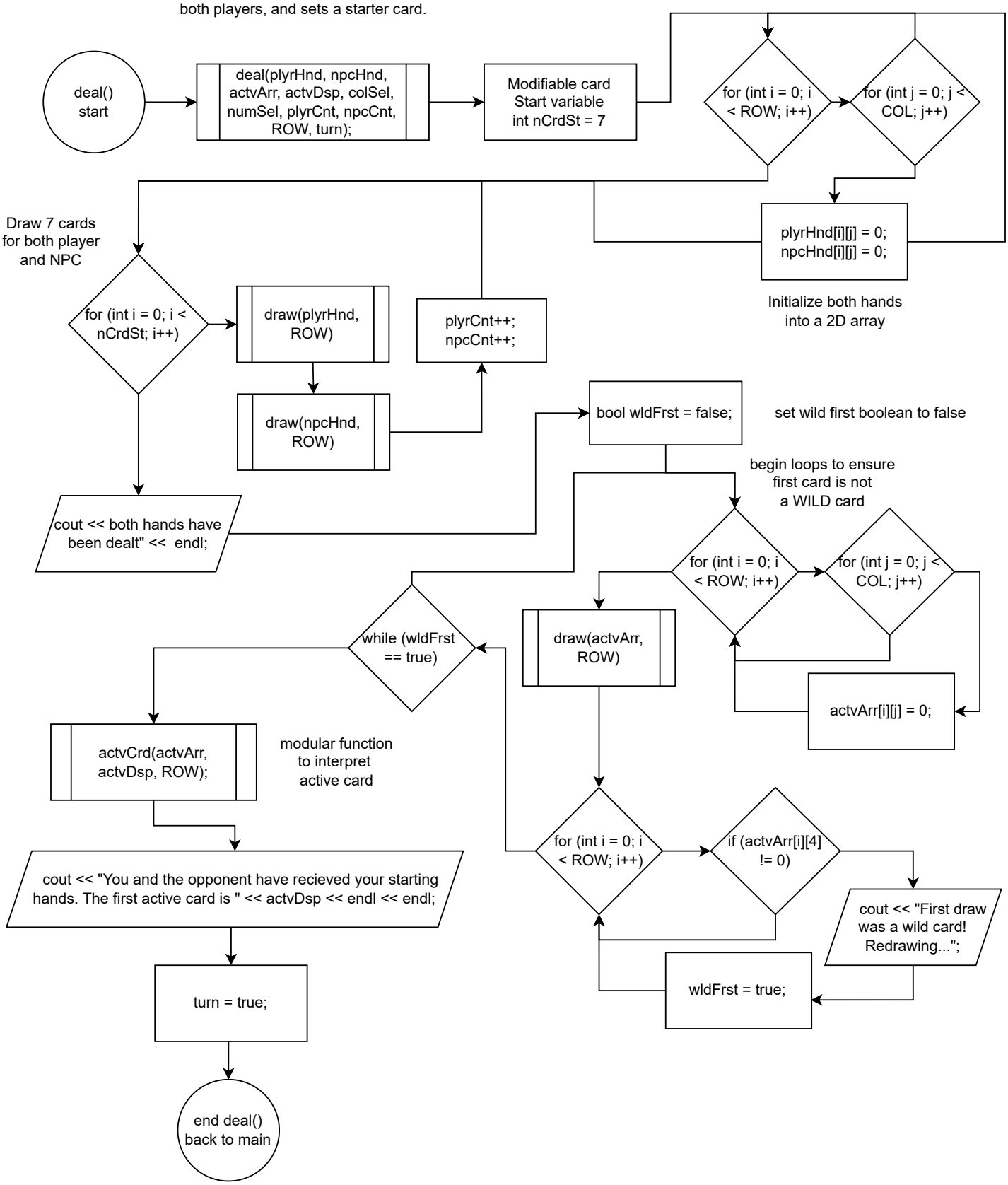
Analytics that include average scores over the last 10 wins, variance, and standard deviation




Samuel Gerungan
Project 1
UNO!
System Libraries
#include <iostream>
#include <iomanip>
#include <cmath>
#include <cstdlib>
#include <cstring>
#include <ctime>
#include <vector>
#include <fstream>
using namespace std;
Global Constant
const int COL = 5;
const char PERCENT = 100;
UNO Flowchart
Phase Two Start!
void draw(int[][COL], int);
void actvCrd(int[][COL], string &, int);
void deal(int[][COL], int[][COL], int[][COL], string &, char, int, int &, int &, int, bool &);
void crdCnv(int[][COL], int);
void usrInt(int[][COL], int[][COL], int[][COL], int[][COL], string &, int &, int &, int, bool &);
void npcTrn(int[][COL], int[][COL], int[][COL], string &, int &, int &, int, bool &);
void dispHnd(int[][COL], int[][COL], string &, int);
void plyrTrn(int[][COL], int[][COL], int[][COL], int[][COL], string &, char, int, int &, int &, int, bool &);
void wild(int[][COL], int, int &);
void play(int[][COL], int[][COL], int[][COL], string &, int &, int &, int, int, int, bool &);
int getCol(char);
void wldPlay(int[][COL], int[][COL], string &, int, int, int, bool &);
void loadScr(vector &scores, fstream &file);
void saveScr(vector &scores, fstream &file);
void clcSts(vector &scores, unsigned int score, float &average, float &variance, float &stdDev, float &pctChng, string &btrWrse);
void dispRes(int plyCnt, int npcCnt, vector &scores, fstream &file, unsigned int score, float average, float variance, float stdDev, float pctChng, string &btrWrse);
void bubSrt(int plyrHnd[][COL], int ROW, string &, int srtdHnd[][COL]);
void selSrt(int plyrHnd[][COL], int ROW, string &, int srtdHnd[][COL]);
Main
Start!
Initialize Random Number
Feature
srand(static_cast(time(0)));
// Declare Variables
const int ROW = 12;
char menuSel, numSel;
int actvArr[ROW][COL], plyrHnd[ROW][COL],
npcHnd[ROW][COL], srtdHnd[ROW][COL],
plyrCnt = 0, npcCnt = 0, colSel = 0;
string name, btrWrse, actvDsp;
fstream file("hiScore.dat", ios::in);
const int maxScrs = 10;
unsigned int sum, score, card, wildCrd, npcTmp, npcWld;
float pctChng, varince, stdDev, average;
bool turn;
Menu UI
cout << "UNO! The game where friendships and loyalties go to die" << endl;
cout << "Main Menu" << endl << endl;
cout << "Type the character that corresponds to your selections within this game" << endl;
cout << "| Enter your name to Start the Game |"<< endl;
cin >> name;
Deal Function
deal(plyrHnd, npcHnd,
actvArr, actvDsp, colSel,
numSel, plyrCnt, npcCnt,
ROW, turn);
if (turn == true)
Player Turn Function
plyrTrn(plyrHnd,
npcHnd, srtdHnd,
actvArr, actvDsp,
colSel, numSel,
plyrCnt, npcCnt,
ROW, turn)
if (turn == false
&& plyrCnt != 0)
NPC Turn Function
npcTrn(plyrHnd,
npcHnd, actvArr,
actvDsp, plyrCnt,
npcCnt, ROW, turn);
while (plyrCnt != 0
&& npcCnt != 0)
cout << "Game Over!" << endl;
Diplay Results Function
dispRes(plyrCnt,
npcCnt, scores, file,
score, average,
varince, stdDev,
pctChng, btrWrse);
end main

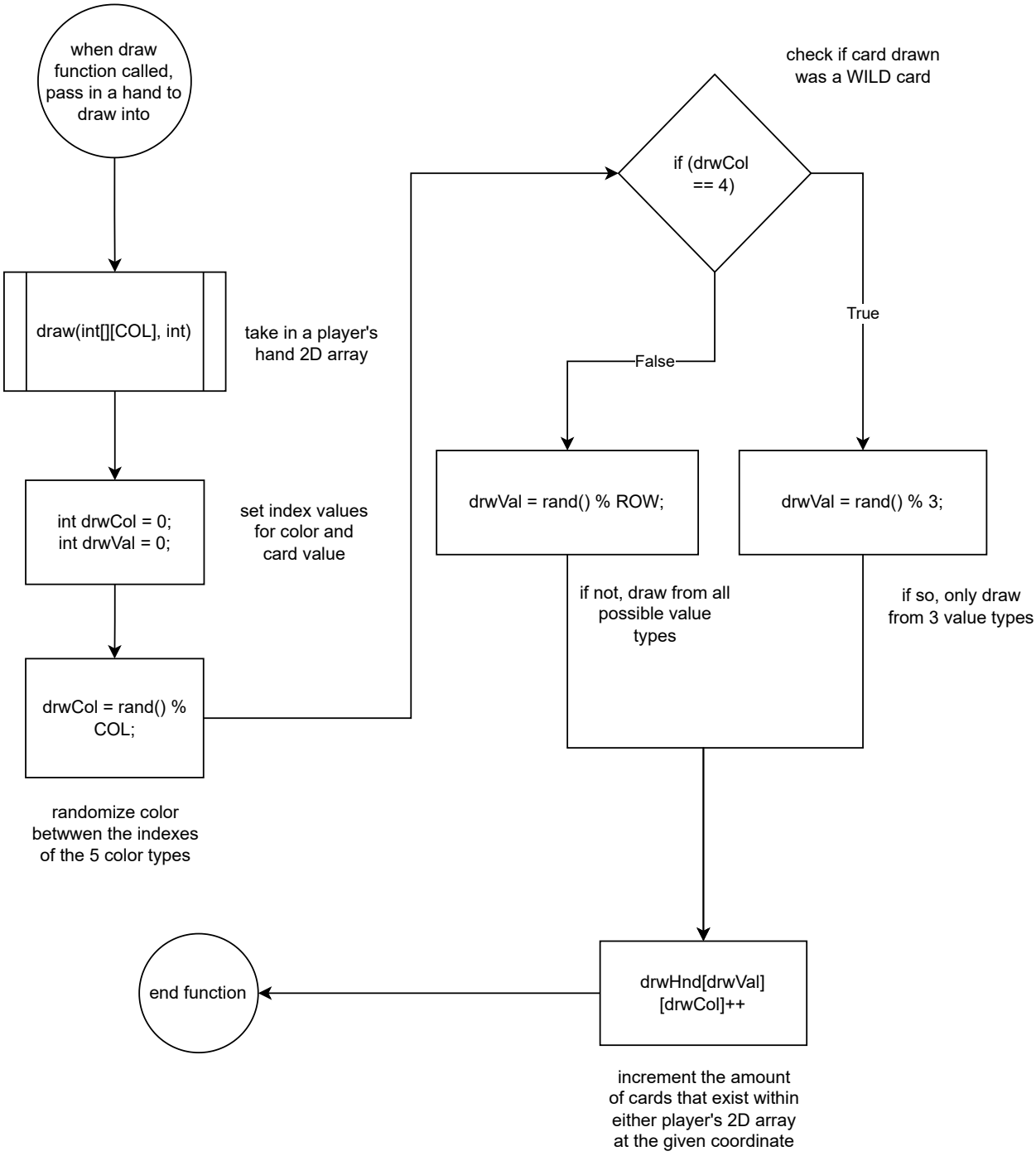
Deal Funcion

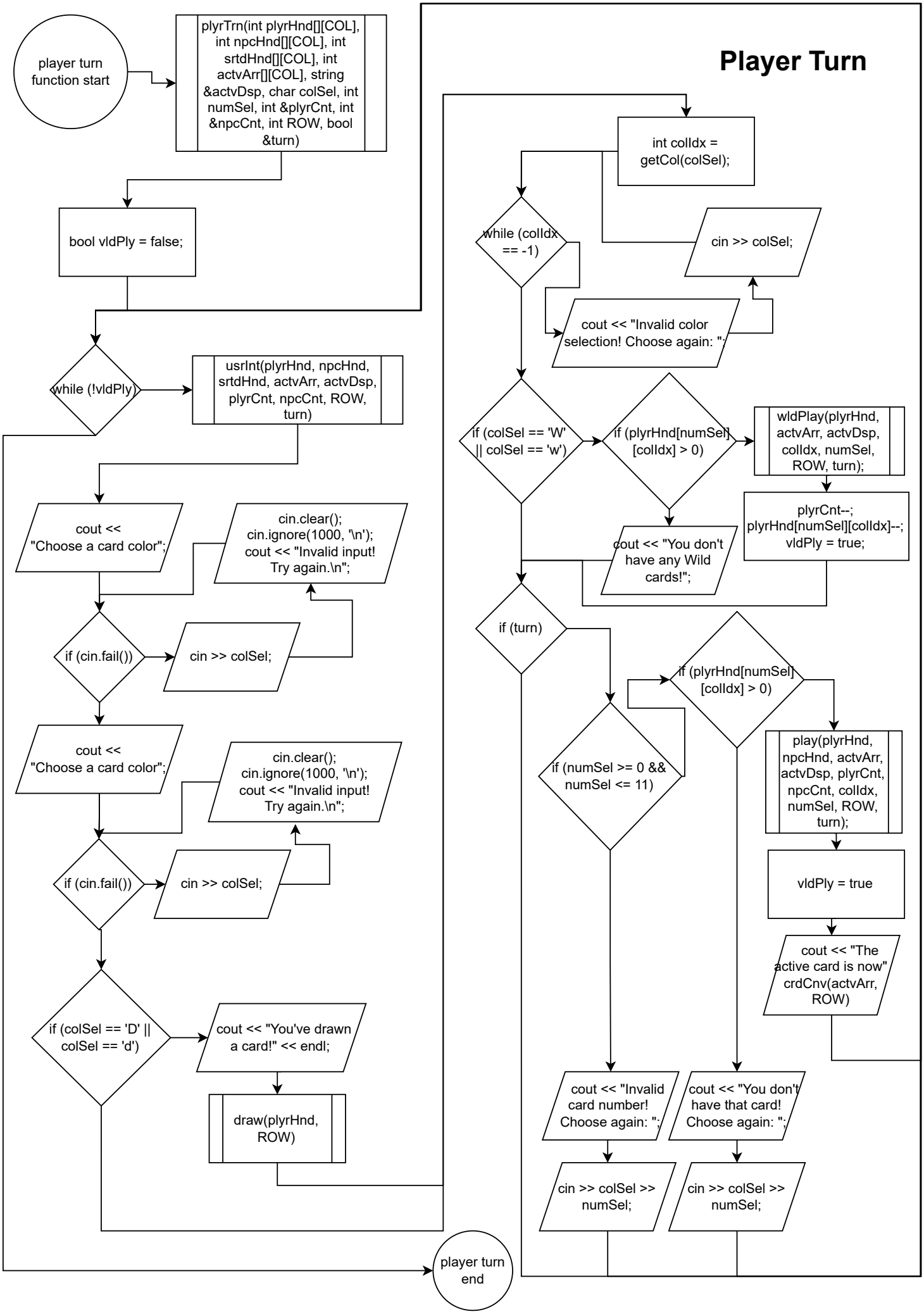
This function deals according to modular and modifiable values set by user. Dictates the first hands of both players, and sets a starter card.



Card Draw Function

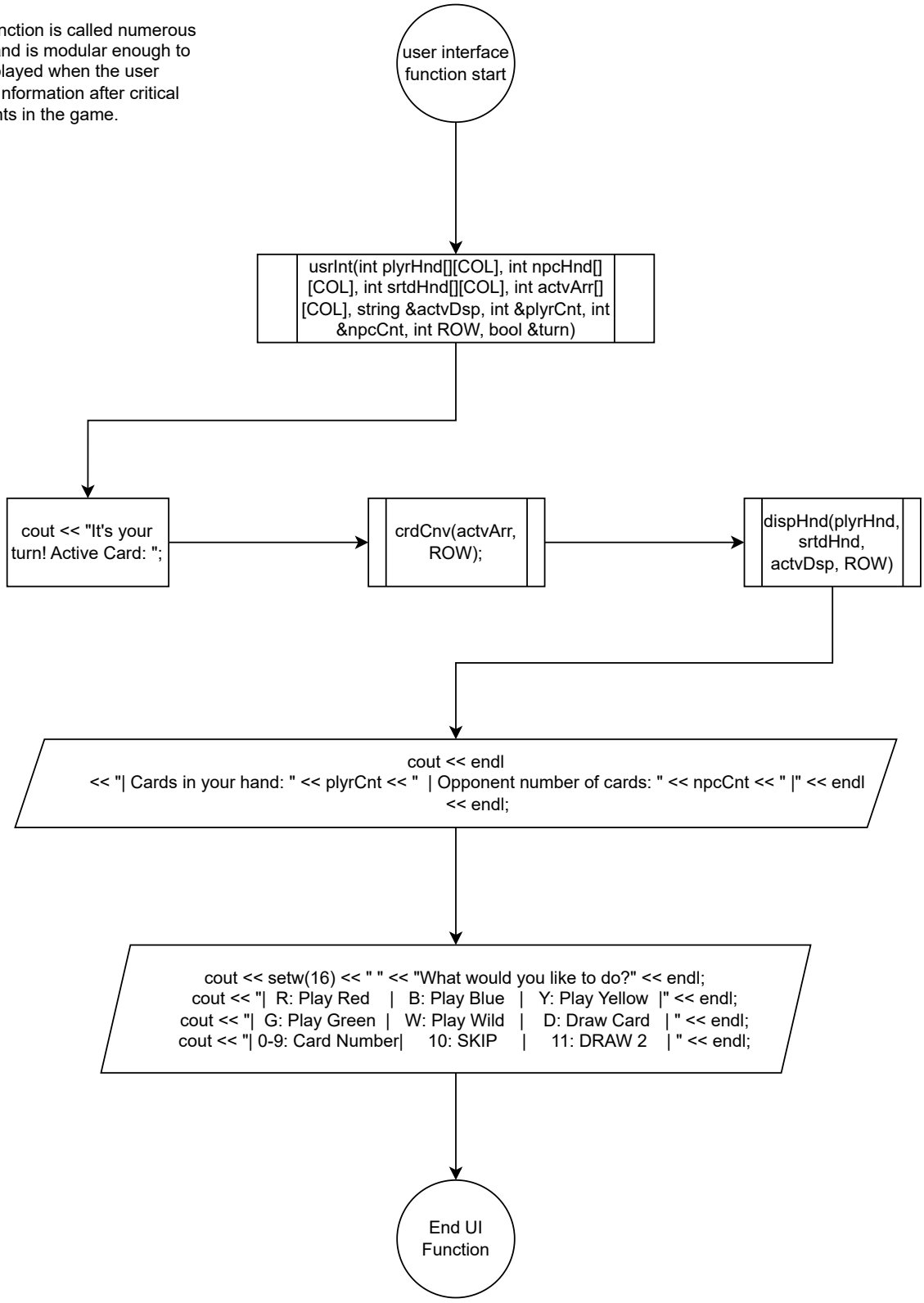
Modular function that takes in a passed player hand array and constant int value for 2D array




Player Turn
player turn function start
plyrTrn(int plyrHnd[][COL], int npcHnd[][COL], int srtdHnd[][COL], int actvArr[][COL], string &actvDsp, char colSel, int numSel, int &plyrCnt, int &npcCnt, int ROW, bool &turn)
bool vldPly = false;
while (!vldPly)
usrInt(plyrHnd, npcHnd, srtdHnd, actvArr, actvDsp, plyrCnt, npcCnt, ROW, turn)
cout << "Choose a card color";
cin.clear(); cin.ignore(1000, '\n'); cout << "Invalid input! Try again.\n";
if (cin.fail())
cin >> colSel;
cout << "Choose a card color";
cin.clear(); cin.ignore(1000, '\n'); cout << "Invalid input! Try again.\n";
if (cin.fail())
cin >> colSel;
if (colSel == 'D' || colSel == 'd')
cout << "You've drawn a card!" << endl;
draw(plyrHnd, ROW)
player turn end
int colIdx = getCol(colSel);
while (colIdx == -1)
cin >> colSel;
cout << "Invalid color selection! Choose again: ";
if (colSel == 'W' || colSel == 'w')
if (plyrHnd[numSel] [colIdx] > 0)
wldPlay(plyrHnd, actvArr, actvDsp, colIdx, numSel, ROW, turn);
plyrCnt--; plyrHnd[numSel][colIdx]--; vldPly = true;
cout << "You don't have any Wild cards!";
if (turn)
if (plyrHnd[numSel] [colIdx] > 0)
if (numSel >= 0 && numSel <= 11)
play(plyrHnd, npcHnd, actvArr, actvDsp, plyrCnt, npcCnt, colIdx, numSel, ROW, turn);
vldPly = true
cout << "The active card is now" crdCnv(actvArr, ROW)
cout << "Invalid card number! Choose again: ";
cout << "You don't have that card! Choose again: ";
cin >> colSel >> numSel;
cin >> colSel >> numSel;

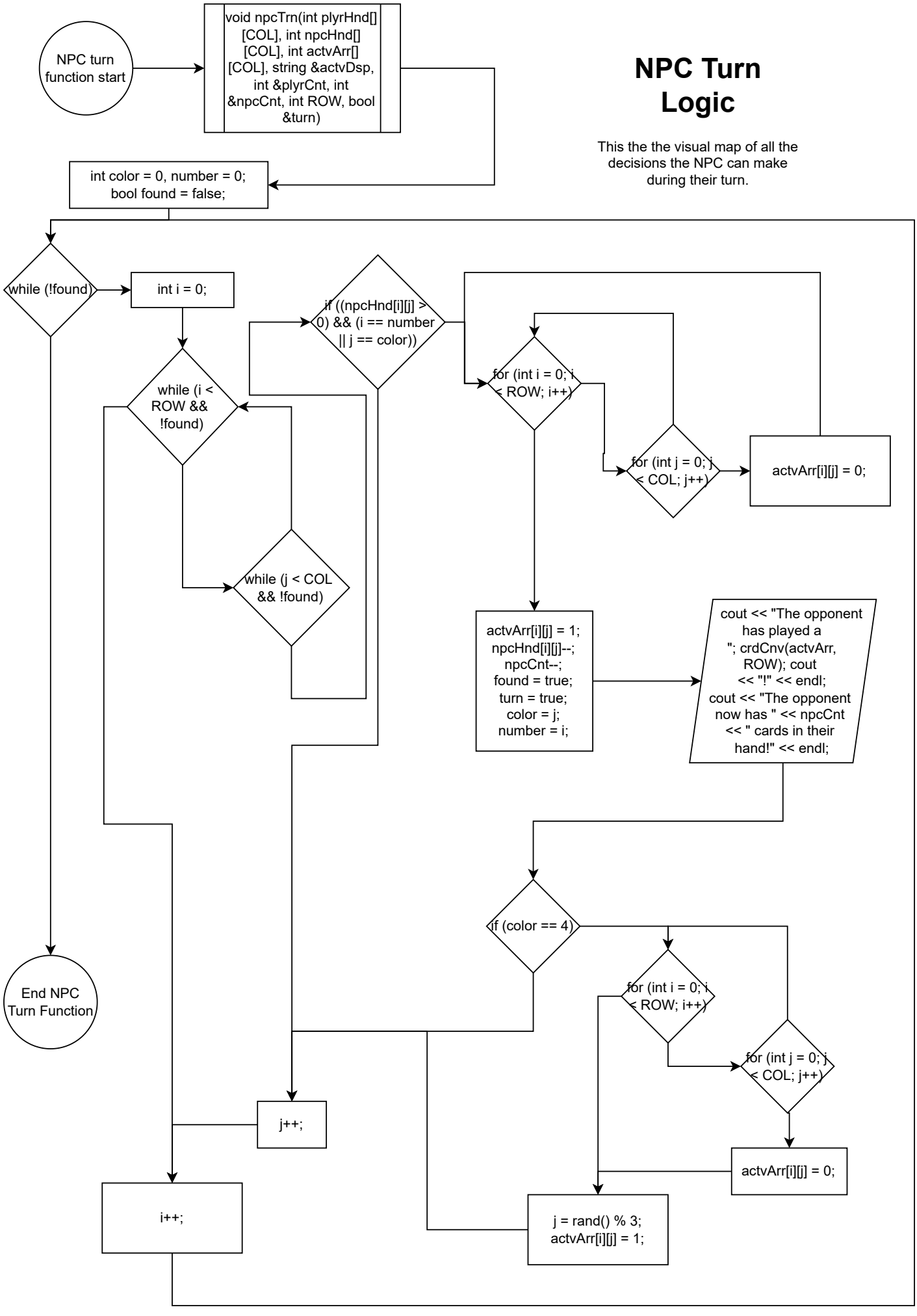
User Interface Function

This function is called numerous times and is modular enough to be displayed when the user needs information after critical moments in the game.



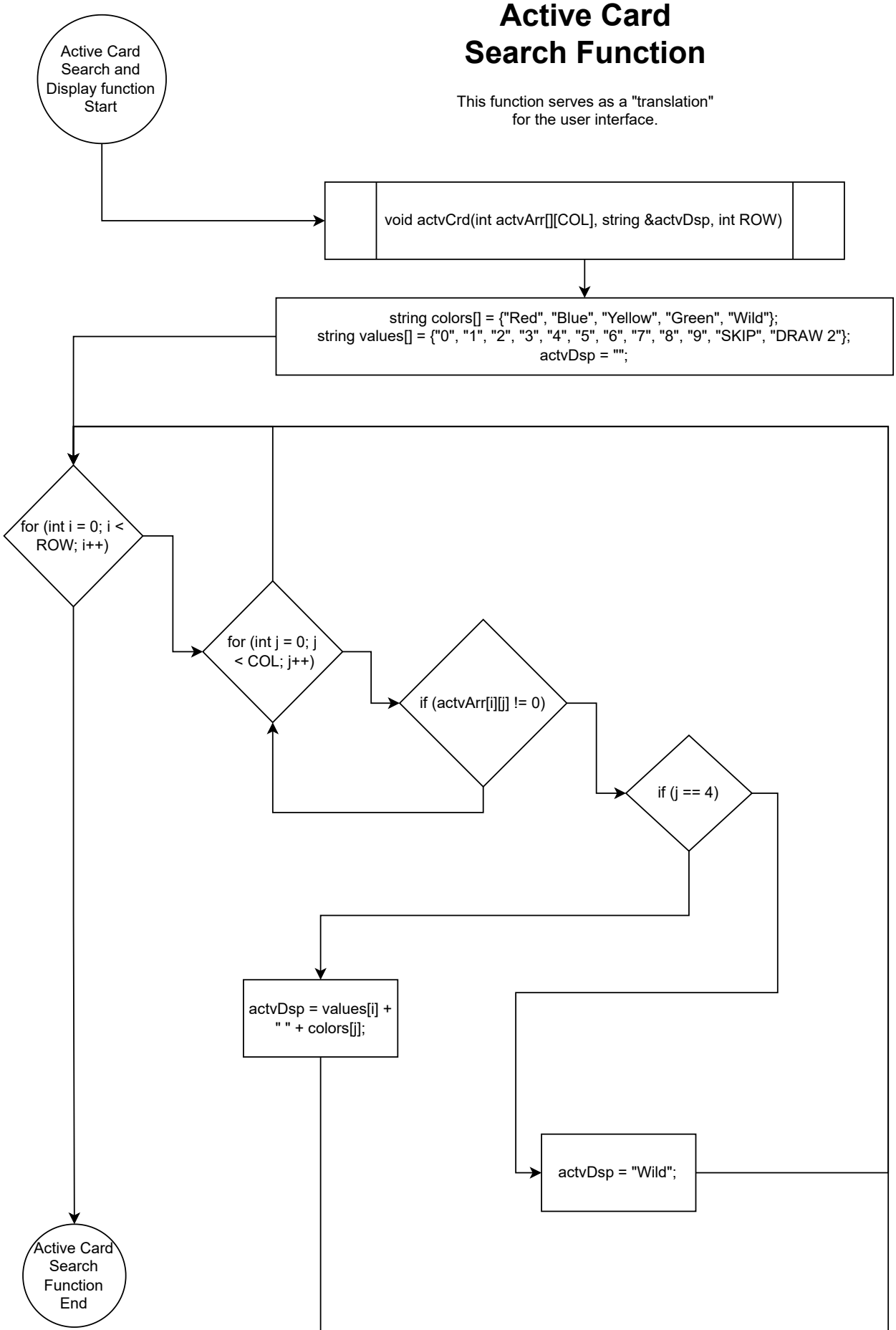
NPC Turn Logic

This the the visual map of all the decisions the NPC can make during their turn.



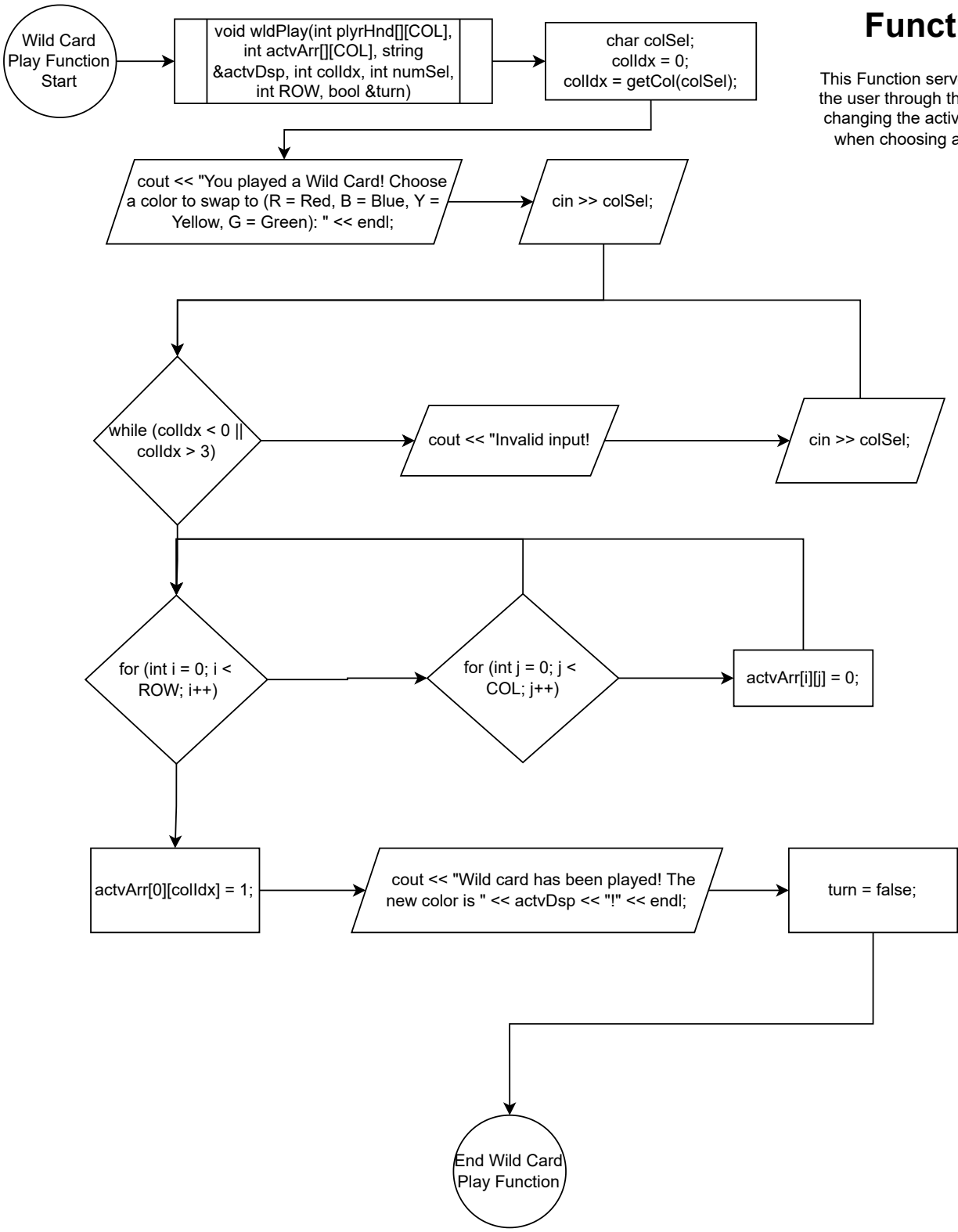
Active Card Search Function

This function serves as a "translation" for the user interface.



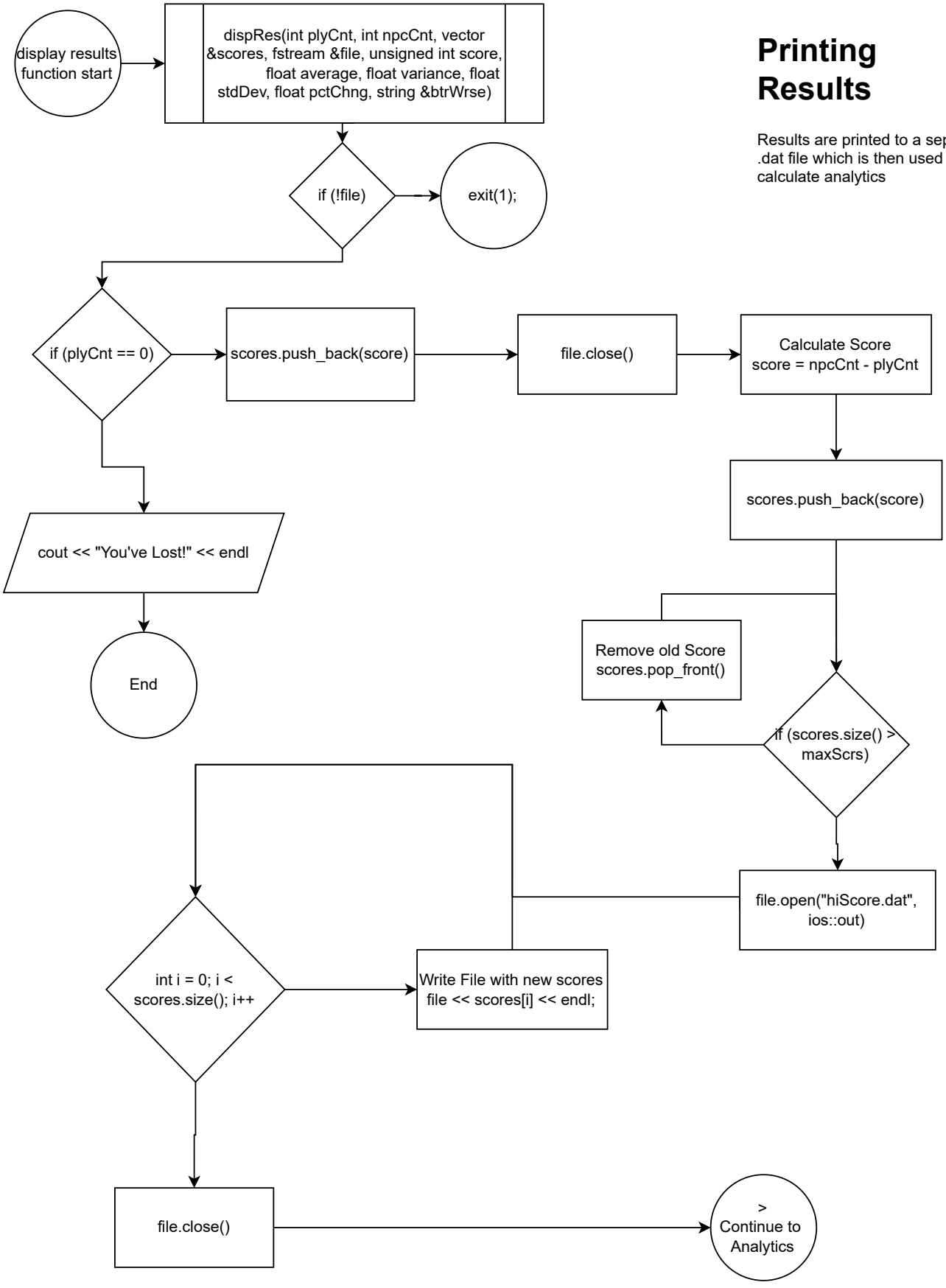
Wild Card Play Function

This Function serves to prompt the user through the process of changing the active card color when choosing a wild card.



Printing Results

Results are printed to a seperate .dat file which is then used to calculate analytics



Post Game Analytics

Analytics that include average scores over the last 10 wins, variance, and standard deviation