Heist Dice

Project Version 2

by Norman Lee
for CSC-5
Date: 7/29/15

-Introduction-

Heist Dice is inspired from a game called Zombie Dice. You play a team of robbers who try to get as much loot as possible, while trying to avoid getting strikes on your record. Like Zombie Dice everything is determined by weighted, colored dice; the dice representing a *heist* you are trying to pull off. The different colors of dice indicate how difficult the job will be. Each round represents a member of your crew. Your opponent is a rival robbery gang. Compete to find out who is the best!

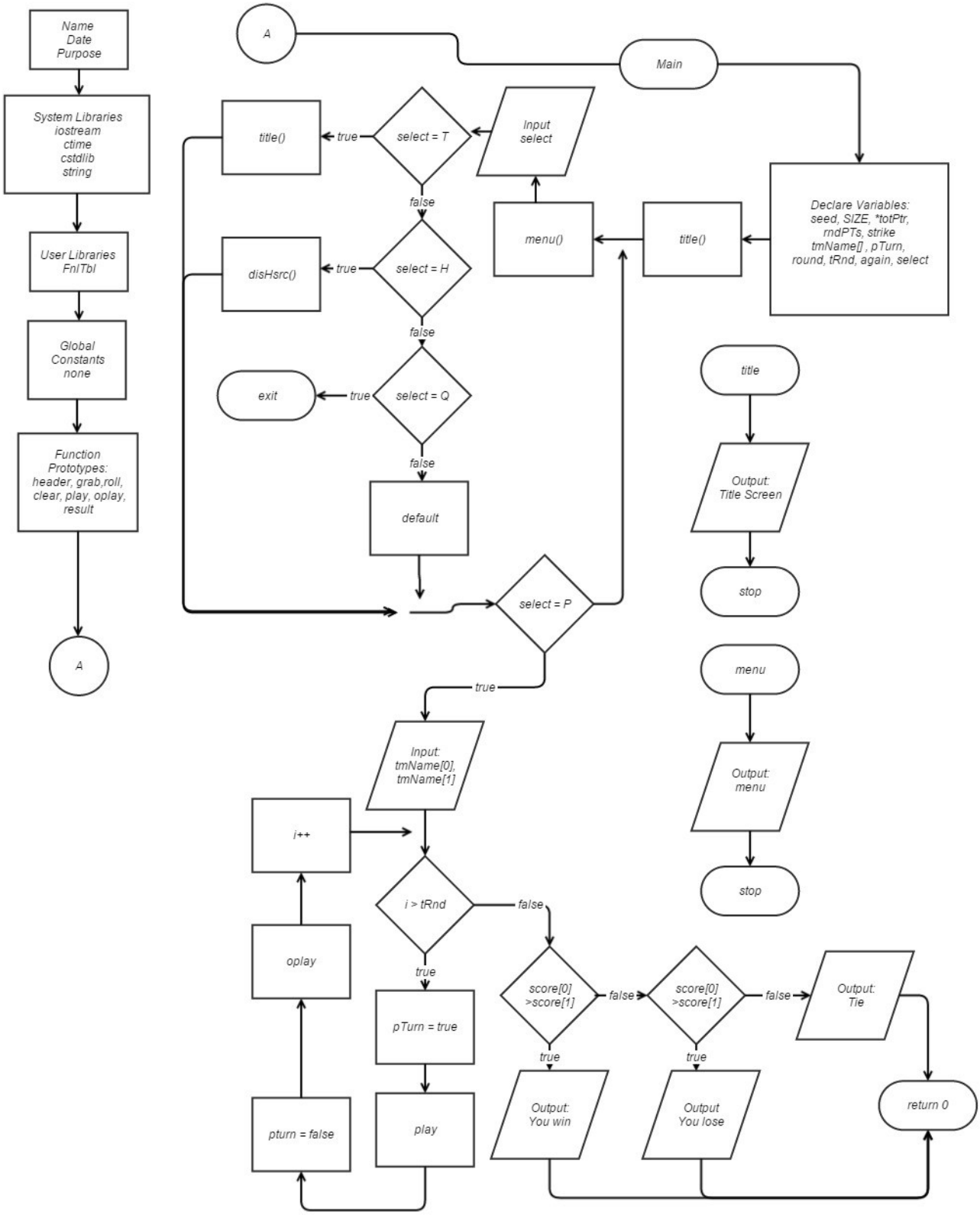
-Instructions-

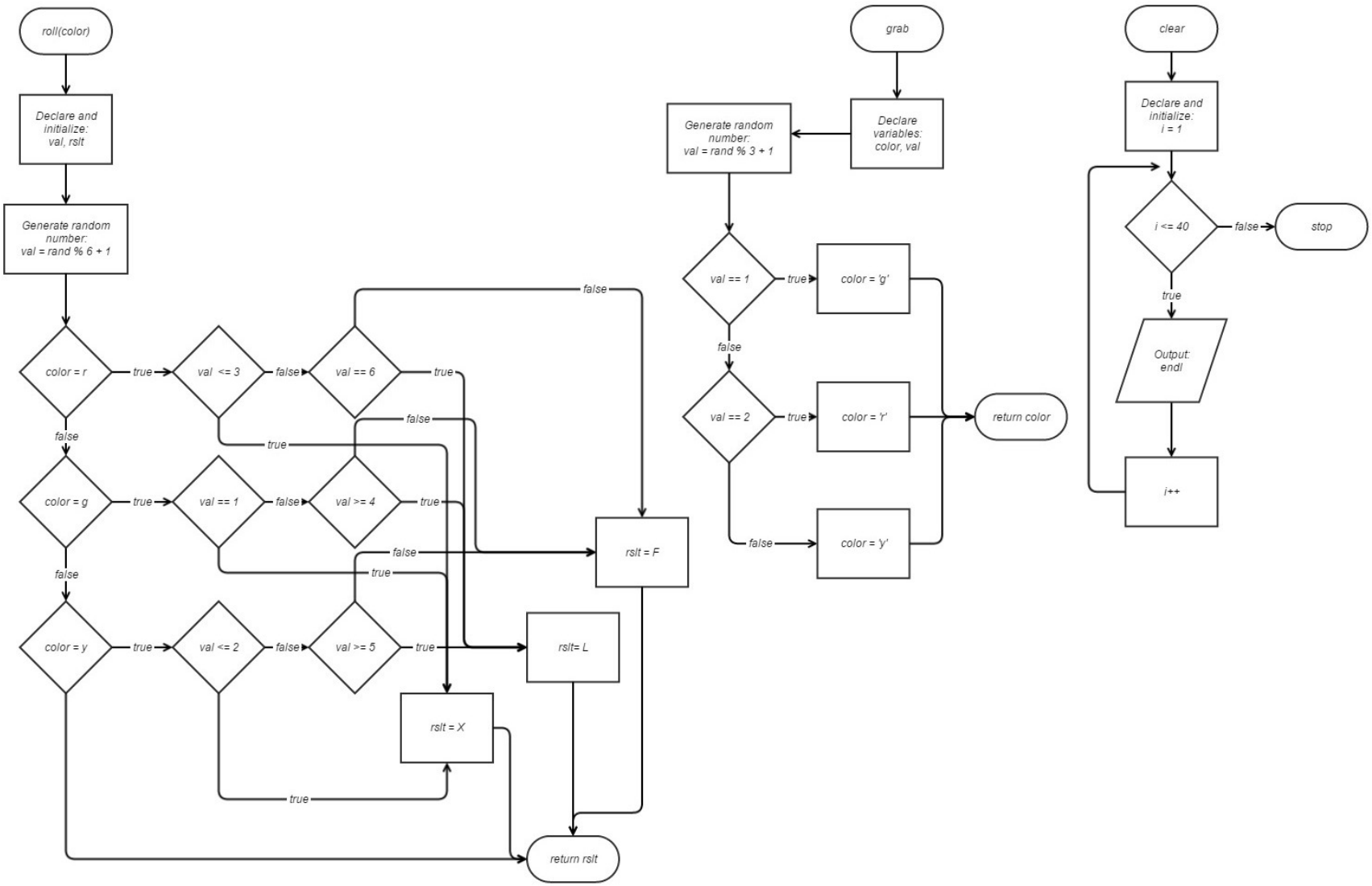
In Heist Dice, you roll to see if you get loot, escape without harm or get caught and end up with a strike on your record. Your first action is to pick a random dice of out a bag. There are 3 different colors of dice, each weighted differently. Green dice have the highest number of sides with loot, Red dice have the highest number of strikes, and Yellow dice have even odds for all outcomes. Every round pull a dice out of a bag and roll the dice until you either get 3 strikes or choose to stop. If you get 3 strikes, your crew member faces life in prison and is forced to give up all the loot to get a plea deal. You take turns rolling with your opponent for 3 rounds and whoever has the most points at the end wins.

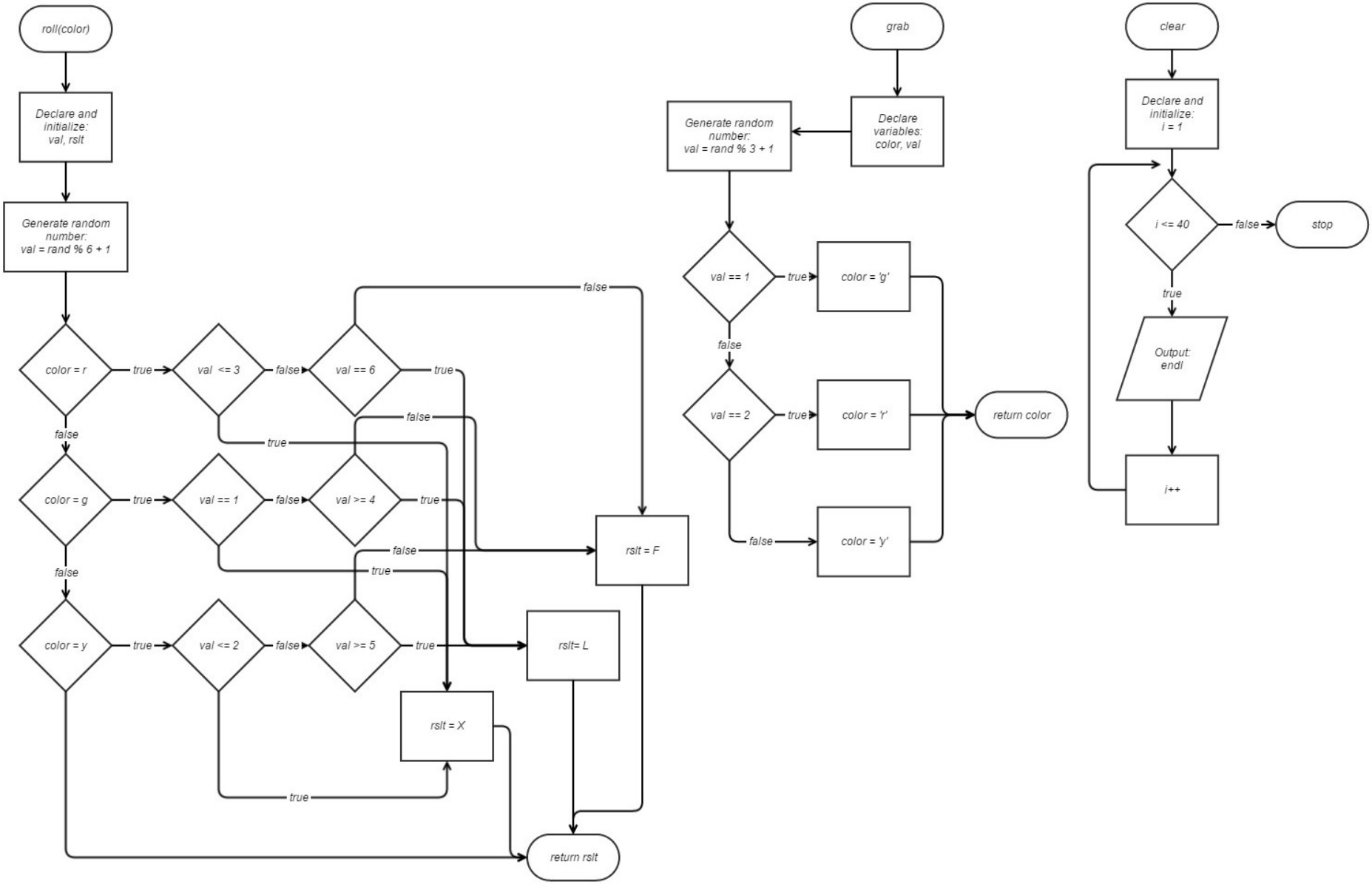
-Design Details-

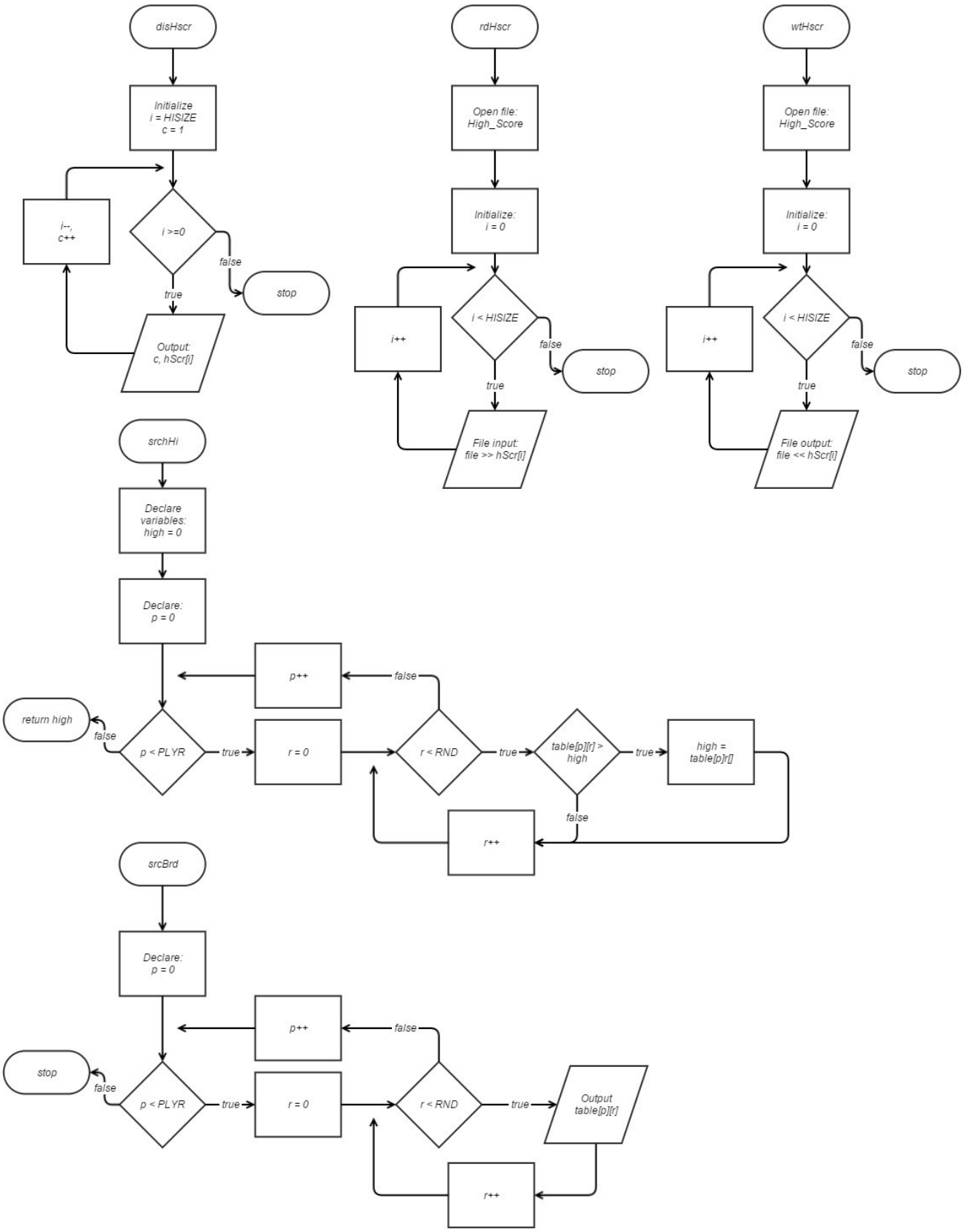
My strategy for coding this game was to first start by programing the dice. Being as this is an original game, I could change the rules during the process, so the next step was setting up a tentative system so I could test various ways of playing the game. I originally had it so, like Zombie Dice, you chose to continue after each roll. But after testing it, the system felt too random and your choices didn't really matter. Zombie Dice has some elements to it that give it more strategy; like the fact you roll 3 dice at a time and keep dice that roll footprints. I felt that rolling 1 dice portrayed the theme of my game better as each die roll would be like a “job” the character would do. It didn't make sense to have 3 different heists run concurrently. My solution was to make the choice of rolling happen when you get a random color for your dice. The draw back in having not set rules from the start is I ended up spending a lot of time shuffling code between different functions and trying to make them work the way that made the most sense. For project 2, I mostly tried to add features that would fulfill the requirements from the check list.

-Flow Charts-


Name
Date
Purpose
System Libraries
iostream
ctime
cstdlib
string
User Libraries
FnlTbl
Global
Constants
none
Function
Prototypes:
header, grab,roll,
clear, play, oplay,
result
A
A
Main
Declare Variables:
seed, SIZE, *totPtr,
rndPTs, strike
tmName[] , pTurn,
round, tRnd, again, select
title()
menu()
Input
select
select = T
true
title()
false
select = H
true
disHsrc()
false
select = Q
true
exit
false
default
select = P
true
Input:
tmName[0],
tmName[1]
i++
i > tRnd
false
true
pTurn = true
play
pturn = false
oplay
score[0]
>score[1]
false
score[0]
>score[1]
false
Output:
Tie
true
Output:
You win
true
Output
You lose
return 0
title
Output:
Title Screen
stop
menu
Output:
menu
stop


roll(color)
Declare and initialize: val, rslt
Generate random number: val = rand % 6 + 1
color = r
true
val <= 3
false
val == 6
true
false
true
false
color = g
true
val == 1
false
val >= 4
true
false
true
false
color = y
true
val <= 2
false
val >= 5
true
true
rslt = F
rslt= L
rslt = X
return rslt
grab
Declare variables: color, val
Generate random number: val = rand % 3 + 1
val == 1
true
color = 'g'
false
val == 2
true
color = 'r'
false
color = 'y'
return color
clear
Declare and initialize: i = 1
i <= 40
false
stop
true
Output: endl
i++


roll(color)
Declare and initialize: val, rslt
Generate random number: val = rand % 6 + 1
color = r
true
val <= 3
false
val == 6
true
false
true
false
color = g
true
val == 1
false
val >= 4
true
false
true
false
color = y
true
val <= 2
false
val >= 5
true
rslt = F
rslt= L
rslt = X
true
true
return rslt
grab
Declare variables: color, val
Generate random number: val = rand % 3 + 1
val == 1
true
color = 'g'
false
val == 2
true
color = 'r'
false
color = 'y'
return color
clear
Declare and initialize: i = 1
i <= 40
false
stop
true
Output: endl
i++


disHscr
Initialize
i = HISIZE
c = 1
i >=0
false
true
stop
i--,
c++
Output:
c, hScr[i]
rdHscr
Open file:
High_Score
Initialize:
i = 0
i < HISIZE
false
true
stop
i++
File input:
file >> hScr[i]
wtHscr
Open file:
High_Score
Initialize:
i = 0
i < HISIZE
false
true
stop
i++
File output:
file << hScr[i]
srchHi
Declare
variables:
high = 0
Declare:
p = 0
p++
false
return high
false
p < PLYR
true
r = 0
r < RND
true
table[p][r] >
high
true
high =
table[p]r[]
false
r++
srcBrd
Declare:
p = 0
p++
false
stop
false
p < PLYR
true
r = 0
r < RND
true
Output
table[p][r]
r++

-Pseudo-code-

```
Start Game:
    read high scores
    print title
    print menu
    prompt for menu selection
        if title
            print title
        if high score
            print high score
        if quit
            exit
        if play
            continue
Setup Game:
    get the teams names
    get the amount of money to be bet
        if bet > bank
            exit

Play Game:
    Repeat once for each round
    Player has a turn
    Computer has a turn
    Add up score
    Display points

Get dice:
    random 1 – 10
    if 1-5
        green dice
    if 9,10
        red dice
    if 6-8
        yellow dice

Roll dice:
    random 1 -6
    if red
        1 – 3  = strike
        6 = point
        4,5 = nothing
    if green
        1 = strike
        4 – 6 = point
        2,3 = nothing
    if yellow
```

1,2 = strike
5,6 = point
3,4 = nothing

Player's turn:
Get random color dice
Prompt for roll or pass
if roll
then roll the dice
get result
if strike
add strike
if loot
add round point
if there are 3 strikes
lose all round points
end the turn
if pass
add all round points to total
end turn

Computer's turn:
Get random color dice
roll dice
if strike = 2
pass
add round points to total

End Game:
Create box score in a structure from round data, total points and team names
Display box score
if player has more points
player wins
add bet to cash
if computer has more points
computer wins
subtract bet from cash
if tie
its a draw

if players total points is higher than the lowest high score
replace the lowest score
sort the list
write the list to the high score file
display list and what place on the list the player achieved
else
subtract the lowest high score from this games total
display by how much the player missed making the top ten

-Major Variables Used-

Type	Variable	Description	Location
int	SSIZE	Size of the array for number of teams	main
int	HISIZE	Size of the array for high scores	main
int	table[][]	2d array that hold scores of all rounds	main
int	hSrc[]	Array to hold the high scores	main
int	totPtr	Pointer “array” to hold total points	main
unsigned short	rndPts	Points earned this round	main
unsigned short	strike	Number of strikes this round	main
string	tnName	Array for team names	main
float	bet	amount the player wishes to bet	
float	cash	amount of money you have	main
int	places	Counter to determine where you are on the high score list	main
char	select	Variable for the menu selection	main
bool	swap	swap for bubble sort	main
bool	pTurn	Is it the player's turn	main
short	round	The round number	main
int	tRnd	Total number of rounds in a game	main
bool	again	Does the user choose to play	main
char	color	Represents the color of the dice	grab
short	val	Random number used to determine the color of the dice	grab
char	val	Random number to determine to outcome	roll
char	rslt	Outcome determined by random number and color of the dice.	roll
char	color	Copied value from grab()	result
char	rslt	Copied value from roll	result
char	rolling	Does the player wish to roll the dice he grabbed	result
int	high	Value used in finding the highest number	srchHi

-Topics Covered-

Topic	**Sub-category**	**Location**
Basic Programs	cout, cin	cout << "You have $" << cash << ", how much do you want to bet: "; cin >> bet;
	bool, char, short, int, long	short round = 0; bool pTurn = true
	float	float cash = 500.00f
	string	string tmName[SSIZE]
	Comments	//Get seed off time and get "random" numbers
Expression, Libraries, Formatting	Mathematical Statements	*(totPtr + 1) += rndPts;
	Constants	const int PLYR = 2;
	Formatting	cout << "#" << setw(2) << c << "." << setw(15)<< hScr[i] << endl;
	File Input/Output	ifstream infile;
Decisions	Relational Operators	totPtr[0] >= hScr[0]
	Logical Operators	}while(rolling != 'y' && rolling != 'n');
	If	if (pTurn == true)

	If-else	else
		{
		disHscr(hScr, HISIZE);

	else if	else if (*(totPtr + 1) > *totPtr)
	switch	switch(select)
Loops	Increment / Decrement	i++, c--
	while	while(again == true)

	do while	do
		{
		cout << "Do you wish to roll the dice?(y/n)";
		cin >> rolling;
		if (isupper(rolling));
		{
		rolling = tolower(rolling);
		}
		}while(rolling != 'y' && rolling != 'n');

	for	for(int i = 0; i < SSIZE; i++)
	counter	places++

	nesting	for(int p = 0; p < PLYR; p++)
		{
		  for(int r = 0; r < RND; r++)

Functions	Prototyping	void menu();
	No Parameters	void title();
	Multiple Parameters	void play(bool &again, bool &pTurn, unsigned short &rndPts, unsigned short &strike, int *totPtr, short &round, string tmName[], int table[][RND], int PLYR)
	Pass by Reference	bool &again
	Void	void title();
	Return	srchHi(int table[][RND], int PLYR), return high
Arrays	1-Dimensional	string tmName[SSIZE] = {};
	2-Dimensional	int table[PLYR][RND] = {};
	Pass into Function	srchHi(int table[][RND], int PLYR)
Search and Sort	Search	srchHi(int table[][RND], int PLYR)
	Sort	//Bubble Sort
Pointers	Dynamic Arrays	int *totPtr = new int [SSIZE]();
ADT	Structures	struct FnlTbl
Files	Ascii	High_Score.txt

```
/*
 * File:   main.cpp
 * Author: Norman Lee
 * Created on July 17, 2015, 3:09 AM
 * Purpose: Heist Dice - a game based off Zombie Dice.
 */

//System Libraries
#include <iostream>
#include <ctime>
#include <cstdlib>
#include <string>
#include <fstream>
#include <iomanip>
using namespace std;

//User Libraries
struct FnlTbl
{
    string name;
    int rnd1;
    int rnd2;
    int rnd3;
    int ttl;
};

//Global Constants
const int PLYR = 2;
const int RND = 3;

//Function Prototypes
void menu();
void title();
void clr();
char grab();
char roll(char);
void result(unsigned short &, unsigned short &, bool &, bool &);
void play(bool &, bool &, unsigned short &, unsigned short &, int *, short &,
string [], int[][RND], int);
void oplay(bool &, bool &, unsigned short &, unsigned short &, int *, short &,
string [], int[][RND], int);
void scrBrd(const int[][RND], int);
int srchHi(int [][RND], int);
void disHscr(int [], int);
void rdHscr(int [], int);
void wtHscr(int [], int);

//Execution begins here
int main(int argc, char** argv) {
```

```
    //Get seed off time and get "random" numbers
    int seed = time(0);
    srand(seed);

    //Declare and Initialize variables
    const int SSIZE = 5, HISIZE = 10;               //number of total score slots,
number of high scores
    int *totPtr = new int [SSIZE]();                //Array to keep track of both
you and your opponents score
    int table[PLYR][RND] = {};                      //table for score in each
round.
    int hScr[HISIZE];                               //High scores
    unsigned short rndPts = 0, strike = 0;          //points and strikes accrued
this round
    char select = 0;                                //selected choice for menu
    int places = 0;                                 //counter to find what place
you got
    string tmName[SSIZE] = {};                      //array to hold team names
    bool pTurn = true, again = true, swap = false;  //Is it the player's turn? /
do you want to play again
    short round = 0;                                //The round number.  Not the
Indiana Jones character.
    int tRnd = 3;                                   // Total rounds to be played
    float cash = 500.00f, bet = 0.0f;               // how much cash you have and
how much you choose to bet

    //Open file
    ifstream infile;                                // File that holds the amount
of money you have
    infile.open("bank.txt");
    infile >> cash;
    infile.close();
    rdHscr(hScr, HISIZE);                           // get the high scores from
the file

    //Output Start Page
    title();
    clr();
    do{
        menu();
        cin >> select;
        if (islower(select));
        {
            select = toupper(select);
        }
        cin.ignore();
        switch(select){
            case 'T':
            {
```

```
            clr();
            title();
            break;
        }
        case 'Q':
        {
            exit(1);
        }
        case 'H':
        {
            clr();
            disHscr(hScr, HISIZE);
            break;
        }
        default: clr();
    };
}while(select != 'P');

//Get names of the teams
cout << "Enter the name of your team:";
getline(cin, tmName[0]);
cout << "Enter the name of your opponent's team:";
getline(cin, tmName[1]);

//Prompt for a bet
cout << "You have $" << cash << ", how much do you want to bet: ";
cin >> bet;
cin.ignore();
if (bet > cash)                                    //Please gamble responsibly
{
    cout << "You shouldn't gamble more money than you have. Please get help.";
    exit(1);
}

//Start the game and loop for the number of rounds
for (int i = 1; i <= tRnd; i++)
{
    //Initialize/Reset conditions for the start of the round
    again = true;
    round++;                                       //Add one to start the round
    //Start your turn
    pTurn = true;
    play(again, pTurn, rndPts, strike, totPtr, round, tmName, table, PLYR);
    //Start opponent's turn
    pTurn = false;
    oplay(again, pTurn, rndPts, strike, totPtr, round, tmName, table, PLYR);
    //End of round phase
    clr();
    scrBrd(table, PLYR);
    cout << "Round " << round << " is over. \n" << tmName[0] << " has " <<
```

```
*totPtr << " points.\n" << tmName[1] << " has " << *(totPtr + 1) << " points.\n";
        cout << "Press enter to continue\n";
        cin.ignore();
    }
    clr();

    //Structure used to make the table for the final report
    FnlTbl report[SSIZE];
    for(int i = 0; i < SSIZE; i++)
    {
        report[i].name = tmName[i];
        report[i].rnd1 = table[i][0];
        report[i].rnd2 = table[i][1];
        report[i].rnd3 = table[i][2];
        report[i].ttl = totPtr[i];
    }
    cout << setw(16) << left << "Player:" << right << setw(9) << "Round 1" <<
setw(10) << "Round 2"
            << setw(10) << "Round 3"  << setw(12) << "Total\n";
    cout << "-------------------------------------------------------------" << endl;
    cout << setw(15) << left << report[0].name << right << setw(10) <<
report[0].rnd1 << setw(10) << report[0].rnd2
            << setw(10) << report[0].rnd3 << setw(10) << report[0].ttl << endl;
    cout << setw(15) << left << report[1].name << right << setw(10) <<
report[1].rnd1 << setw(10) << report[1].rnd2
        << setw(10) << report[1].rnd3 << setw(10) << report[1].ttl << endl;

    //Get highest score
    cout << "The most amount of points scored in a round was " << srchHi(table,
PLYR) << ".\n";

    //Determine who won
    if (*totPtr > *(totPtr + 1))
    {
        cout << tmName[0] << " Wins! ";
        cash += bet;
        cout << "You now have $" << cash << ".\n";
    }
    else if (*(totPtr + 1) > *totPtr)
    {
        cout << tmName[1] << " Wins... ";
        cash -= bet;
        cout << "You now have $" << cash << ".\n";
    }
    else
    {
        cout << "It's a draw. ";
    }

    //High Score Section
```

```
    cout << "Press Enter to continue: ";
    cin.ignore();
    if (totPtr[0] >= hScr[0])                       //if your score was higher that
the lowest score on the list
    {
        hScr[0] = totPtr[0];                        //Replace the lowest score with
your score
        //Bubble Sort
        do
        {
            swap = false;
            for(int i = 0; i < (HISIZE -1); i++)
            {
                if (hScr[i] > hScr[i + 1])
                {
                    hScr[i] = hScr[i]^hScr[i + 1];
                    hScr[i + 1] = hScr[i]^hScr[i + 1];
                    hScr[i] = hScr[i]^hScr[i + 1];
                    swap = true;
                    places++;
                }
            }
        } while(swap);
        //Output what place you got
        disHscr(hScr, HISIZE);
        wtHscr(hScr, HISIZE);                       //Write the new high score list
to file
        cout << "You are number " << HISIZE - places << " on the high score list.";
    }
    else
    {
        disHscr(hScr, HISIZE);
        cout << "You missed making it into the top ten by " << hScr[0] - totPtr[0]
<< endl;
    }
    //Output the cash back into the file
    ofstream outfile;
    outfile.open("bank.txt");
    outfile << fixed << setprecision(2);
    outfile << cash;
    outfile.close();
    delete [] totPtr;
    return 0;
}

//**********************************grab*****************************************
// Purpose: Simulate grabbing a dice out of the bag and getting a random color
// Outputs: color -> g = green, r = red, y = yellow
//*******************************************************************************
char grab()
```

```
{
    char color;
    short val;
    val = rand()% 10 + 1;
    //determine color of dice grabbed
    if (val <= 5)                                   //50% chance for green
        {
            color = 'g';
            cout << "A Green Dice has been pulled out of the bag.\n";
        }
    else if (val >= 9)                              //20% chance for red
        {
            color = 'r';
            cout << "A Red Dice has been pulled out of the bag.\n";
        }
    else
        {
            color = 'y';                            //30% chance for yellow
            cout << "A Yellow Dice has been pulled out of the bag.\n";
        }

    return (color);
}

//**********************************roll*****************************************
//Purpose: Roll a dice and get a weighted outcome based on the color
//Input: dice -> the color of dice
//Output: rslt -> weighted results, green has the highest positive outcome,
//reds have the lowest, yellows are even
//*******************************************************************************
char roll(char dice)
{
    char val, rslt;
    val = rand()%6 + 1;
    if (dice == 'r')                                //Red dice, weighted with 50%
chance of a strike
    {
        if (val <= 3)                               //%50 chance of a strike
        {
            rslt = 'X';                             //you got caught/a strike
        }
        else if (val == 6)                          //1/6 chance for a point
        {
            rslt = 'L';                             //you got loot/a point
        }
        else                                        //1/3 chance to get away
        {
            rslt = 'F';                             //you failed to get loot but
got away
        }
```

```
    }
    else if (dice == 'g')                           //Green dice, weighted with a
50% chance to get you Loot
    {
        if (val == 1)                               //1/6 chance of a strike
        {
            rslt = 'X';                             //you got caught/a strike
        }
        else if (val >= 4)                          //1/2 chance of a point
        {
            rslt = 'L';                             //you got loot/a point
        }
        else                                        //1/3 chance of getting
        {
            rslt = 'F';                             //you failed to get loot but
got away
        }
    }
    else if (dice == 'y')                           //Yellow dice, weighted with
even odds for all outcomes
    {
        if (val <= 2)
        {
            rslt = 'X';                             //you got caught/a strike
        }
        else if (val >= 5)
        {
            rslt = 'L';                             //you got loot/a point
        }
        else
        {
            rslt = 'F';                             //you failed to get loot but
got away
        }
    }
    return (rslt);
}

//*************************result**********************************
//Purpose: Add points or strike and output ascii picture
//Inputs:
//    rndPts -> points for this round
//    strike -> number of strikes
//      again  -> play again?
//      pTurn  -> is it the players turn
//Outputs:
//    rndPts
//    strike
//*****************************************************************
void result(unsigned short &rndPts, unsigned short &strike, bool &again, bool
```

```
&pTurn)
{
    char rslt = 0, color = 0, rolling = 0;          // copied values for color and
the value you get from rolling
    color = grab();                                 //grab a dice from the bag and
find out which color you got  //Check if user wants to roll
    if (pTurn == true)
    {
        do
        {
            cout << "Do you wish to roll the dice?(y/n)";
            cin >> rolling;
            if (isupper(rolling));
            {
                rolling = tolower(rolling);
            }
        }while(rolling != 'y' && rolling != 'n' );

        if (rolling == 'n')
        {
            again = false;
        }
        cin.ignore();
    }
    if (pTurn == false || rolling == 'y')
    {
        rslt = roll(color);                         //roll the dice and get the
value
        clr();
        //Output  representation of the dice and increment strikes or points based
on roll result
        if (rslt == 'L')
        {
            cout << ".-----------.\n"
                 << "|   $$$$$   |\n"
                 << "|    $ $    |\n"
                 << "|  $     $  |\n"
                 << "| $       $ |\n"
                 << "|  $$$$$$$  |\n"
                 << "'-----------'\n";
            cout << "Successfully stole some loot!\n";
            rndPts++;
        }
        else if (rslt == 'X')
        {
            cout << ".-----------.\n"
                 << "| x       x |\n"
                 << "|   x   x   |\n"
                 << "|     x     |\n"
                 << "|   x   x   |\n"
```

```
                 << "| x       x |\n"
                 << "'-----------'\n";
            cout << "Got caught and a strike was placed on your record.\n";
            strike++;
        }
        else
        {
            cout << ".-----------.\n"
                 << "| RRRRR     |\n"
                 << "| R   R     |\n"
                 << "| R   RRRR  |\n"
                 << "| R       R |\n"
                 << "| RRRRRRRRR |\n"
                 << "'-----------'\n";
            cout << "Failed at the attempt but got away.\n";
        }
    }
}
//Purpose: Generate a title with information
void title()
{
    cout << " . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
. ..-----------. . . . . .\n"
         << ". . . . . $ . .$. .$$$$ . $ .$$$$$$ $$$$$$. . . 8888. 8 .888. .
8888 . . . |   $$$$$   |. . . . . \n"
         << " . . . . .$. . $ . $ . . .$.$$ . . . .$. . . . .8. .8.8.8. .8. 8 . . .
. .|    $ $    | . . . . .\n"
         << ". . . . . $$$$$$. .$$$. . $ .$$$$$. . $ . . . . 8 . 8 8 8 . . .
888. . . . |  $     $  |. . . . . \n"
         << " . . . . .$. . $ . $ . . .$. . . $$. .$. . . . .8. .8.8.8. .8. 8 . . .
. .| $       $ | . . . . .\n"
         << ". . . . . $ . .$. .$$$$ . $ $$$$$$. . $ . . . . 8888. 8 .888. .
8888 . . . |  $$$$$$$  |. . . . . \n"
         << " . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
. .'-----------' . . . . .\n"
         << "How to play Heist Dice: Every round you get to roll the dice.\n\n"
         << "There are 3 different outcomes you get from rolling: A loot bag,
shoes, and a strike.\n"
         <<
"-----------------------------------------------------------------------------------
----------------\n"
         << "Loot bags: you score a point for the round.\n"
         << "Shoes: nothing good or bad happens.\n"
         << "Strikes: If you get 3 strikes, the round ends and you lose any points
you made for that round.\n"
         << ".-----------.     .-----------.     .-----------.\n"
         << "|   $$$$$   |     | RRRRR     |     | x       x |\n"
         << "|    $ $    |     | R   R     |     |   x   x   |\n"
         << "|  $     $  |     | R   RRRR  |     |     x     |\n"
         << "| $       $ |     | R       R |     |   x   x   |\n"
```

```
         << "|  $$$$$$$  |     | RRRRRRRRR |     | x       x |\n"
         << "'-----------'     '-----------'     '-----------'\n\n"
         << "There are 3 types of die: Green, Red and Yellow.\n"
         <<
"-------------------------------------------------------------------------------
---------------\n"
         << "Green: Has the highest amount of loot bags and only 1 strike.\n"
         << "Red: Has the highest amount of strikes and only 1 loot bag.\n"
         << "Yellow: All outcomes have an equal chance.\n"
         <<
"-------------------------------------------------------------------------------
---------------\n"
         << "After you grab a dice from the bag you can choose to stop and keep the
points you have accrued.\n"
         << "There are 3 rounds, you and your opponent take turns rolling for each
round.\n";
    cout << "<Press Enter to continue>\n";
    cin.ignore();
}

//Purpose: Displays a menu
void menu()
{
    cout<<"------------Menu-------------"<<endl;
    cout<<"|      Press:               |"<<endl;
    cout<<"|                           |"<<endl;
    cout<<"|      (P)lay the game      |"<<endl;
    cout<<"|      (T)itle Screen       |"<<endl;
    cout<<"|      (H)igh Scores        |"<<endl;
    cout<<"|      (Q)uit               |"<<endl;
    cout<<"----------------------------|"<<endl;
}

//Purpose: uses 40 new lines to clear the screen
void clr()
{
    for(int i = 1; i <=40; i++)
    {
    cout << endl;
    }
}
//*******************************play*****************************************
//Purpose: Player's turn to play the game
//Inputs:
// again -> play again, true or false
// rndPts -> points this round
// strike -> strikes this round
// totPtr[] -> total scored points
// round -> the round number
// tmName -> the your team name
```

```
//Outputs:
// again -> play again?
// rndPts -> points this round
// strike -> strikes this round
// totPtr[] -> total scored points
// tmName -> the your team name
//*************************************************************************
void play(bool &again, bool &pTurn, unsigned short &rndPts, unsigned short &strike,
int *totPtr, short &round, string tmName[], int table[][RND], int PLYR)
{
    while(again == true)
    {
        cout << "Round " << round << "<Your turn>" << endl;
        result(rndPts, strike, again, pTurn);
        if (again == false)                         //When you stop rolling your
points get added to your total
        {
            *totPtr += rndPts;
            table[0][(round-1)] += rndPts;
            rndPts = 0;
            strike = 0;
            cout << "Added point(s) to total. " << tmName[0] << " has " << *totPtr
<< " total points.\n\n";
        }
        if (strike >= 3)                            //If you get 3 strike lose all
points
        {
            cout << tmName[0] << " member <" << round <<"> has 3 strikes. Facing
life in prison, you give up all your loot for a plea deal.\n\n";
            rndPts = 0;
            again = false;
            strike = 0;
        }
        else
        {
            if (again == true)
            {
            cout << tmName[0] << " member <" << round << "> has " << rndPts << "
points this round and " << strike << " strikes. " << tmName[0] << " has " <<
*totPtr << " total points.\n\n";
            }
        }
        cout << "<Press Enter to continue>\n";
        cin.ignore();
    }
}
//*********************************oplay*************************************
//Purpose: Have the computer opponent play
//Inputs:
// again -> play again, true or false
```

```
// rndPts -> points this round
// strike -> strikes this round
// totPtr[] -> total scored points
// round -> the round number
// tmName[] -> the computer's team name
//Outputs:
// again -> play again?
// rndPts -> points this round
// strike -> strikes this round
// totPtr[] -> total scored points
// oTeam -> the computer's team name
//**************************************************************************
void oplay(bool &again, bool &pTurn, unsigned short &rndPts, unsigned short
&strike, int *totPtr, short &round, string tmName[], int table[][RND], int PLYR)
{
    again = true;
    while (again == true)
    {
        result(rndPts, strike, again, pTurn);
        cout << tmName[1] << " member " << round <<" has "<< rndPts << " points and
" << strike << " strikes.\n" << endl;
        cout << "Round " << round << " <Opponent's turn>" << endl;
        cout << "<Press enter to continue>\n";
        cin.ignore();
        if (strike >= 2)
        {
            //Add round points to total score and reinitialize
            *(totPtr + 1) += rndPts;
            table[1][(round-1)] += rndPts;
            rndPts = 0;
            strike = 0;
            cout << tmName[1] << " chooses to stop. \n" << "Adding point(s) to
total. " << *(totPtr + 1) << " total points for "<< tmName[1] <<".\n";
            again = false;
            cout << "<Press enter to continue>\n";
            cin.ignore();
        }
    }
}
//**********************************scrBrd*****************************************
//Purpose: Keep tally for each of the rounds
//Inputs:
//  table[][RND] -> the table to output
//  PLYR -> the player row
//Outputs:
//  table[PLYR][RND] -> a table with the score for each round so far
//**************************************************************************
void scrBrd (const int table[][RND], int PLYR)
{
    cout << "Scoreboard:\n";
```

```
    for(int p = 0; p < PLYR; p++)
    {
        for(int r = 0; r < RND; r++)
        {
            cout << table[p][r] << " ";
        }
        cout << endl;
    }
}

//Purpose: Search for the highest round score
int srchHi(int table[][RND], int PLYR)
{
    int high = 0; // highest
    for(int p = 0; p < PLYR; p++)
    {
        for(int r = 0; r < RND; r++)
        {
            if (table[p][r] > high)
            {
                high = table[p][r];
            }
        }
    }
    return high;
}

//Purpose: Displays High Score counting down
void disHscr(int hScr[], int HISIZE)
{
    cout << "----High Scores----\n";
    for(int i = HISIZE -1, c = 1; i >= 0; i--, c++)
    {
        cout << "#" << setw(2) << c << "." << setw(15)<< hScr[i] << endl;
    }
    cout << "-------------------\n";
}

//Purpose: Reads file to get high scores
void rdHscr(int hScr[], int HISIZE)
{
    ifstream file;
    file.open("High_Scores.txt");
    file.clear();
    //Loop add the data from the file to the array
    for(int i = 0; i < HISIZE; i++)
    {
        file >> hScr[i];
    }
    file.close();
```

```
}

//Purpose: Writes high scores back into the file
void wtHscr(int hScr[], int HISIZE)
{
    ofstream file;
    file.open("High_Scores.txt");
    //Loop output the array to the file
    for(int i = 0; i < HISIZE; i++)
    {
        file << hScr[i] << " ";
    }
    file.close();
}
```