Project 2

<Dungeons and Dragons Combat Sim>

CIS-5, Winter 2021

Name: Jason Wilmot

2/14/2021

Introduction

Title: Dungeons and Dragons Combat Sim

Dungeons and Dragons (D&D) is a popular tabletop roleplaying game where players use dice rolls modified by character attributes to determine their success at any given task. In combat, a player will roll a twenty-sided die (commonly called a d20) modified by their character's physical attributes and proficiency to see if they land a hit. Then, roll a smaller die with anywhere between twelve and four sides to find out how much damage was afflicted to their target.

> A combat scenario may look like this…
>
> *LVL 3 Fighter V.S. Goblin*
>
> *Fighter rolls d20 + strength mod + proficiency mod to hit, resulting in 10 + 3 + 2 = 15*
>
> *15 is greater than or equal to the Goblin's Armor Class (AC) resulting in a hit.*
>
> *Fighter rolls d8 + 3, resulting in 6 + 3 = 9 damage to the Goblin.*
>
> *Fighter kills Goblin.*

Besides this, my program also provides a variety of quality of life features and utility functions for players, allowing them to ignore the math and focus on the game.

Summary

Project size: 651 lines
Number of variables: ~50, not all unique
Number of functions: 26

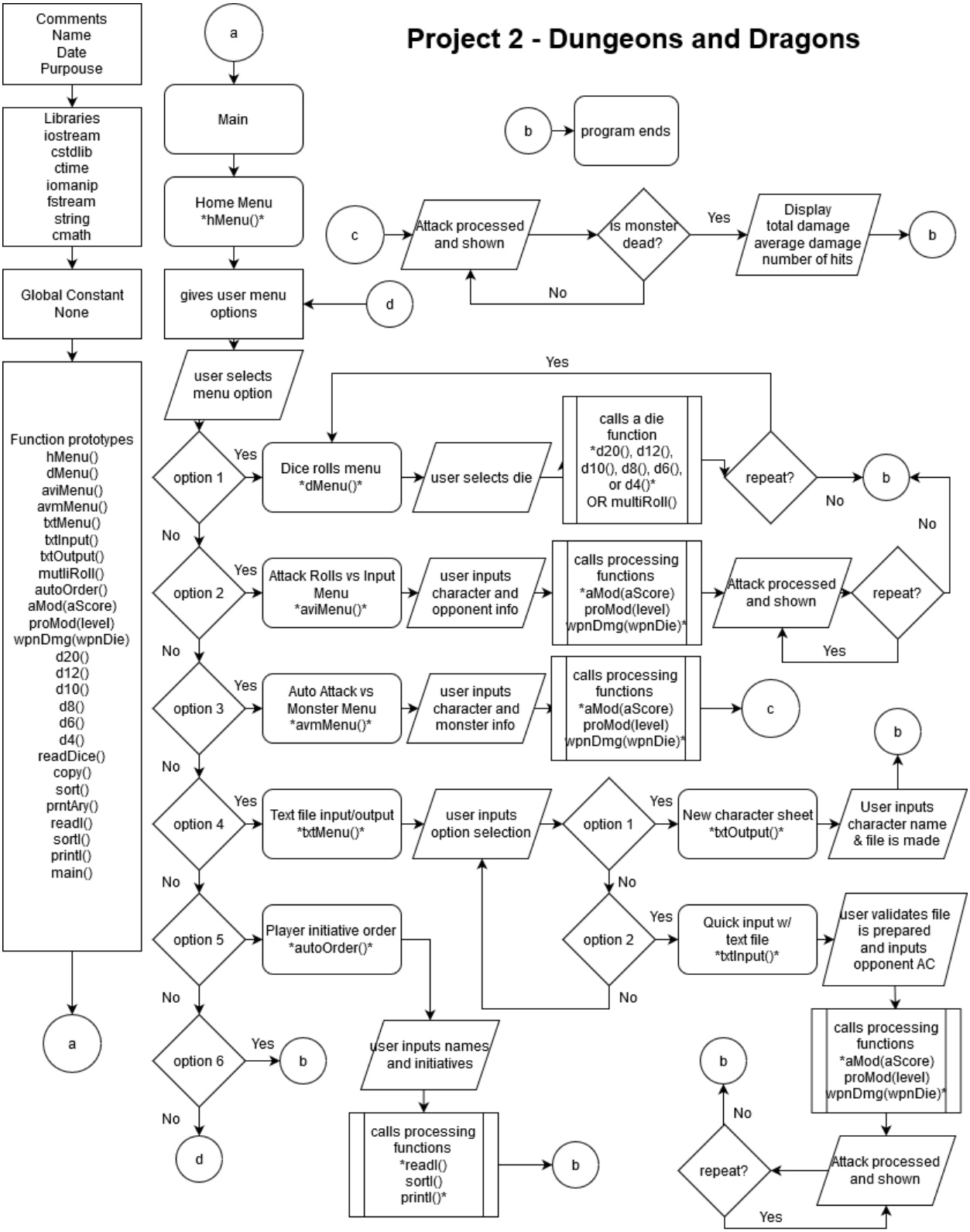
My project uses most of the concepts we have covered in the book, I made most of it modular to make it simpler and more readable.

The initial project took about 4 days to finish, and the second project took 1 day to update everything with new topics. Ever since the project was brought up in class, I was thinking of doing a D&D game, so I had most of the program planned out well in advance of starting it. My program barely scratches the surface of the complexity of D&D but choosing such a complex game gave me the opportunity to add different sections that show off what I've learned in class in creative ways.

Description

The main point of this program is to serve as a hub filled with player resources that are commonly used during a game of D&D. These include dice rolls, combat rolls, and character creation.

Flow Chart


Project 2 - Dungeons and Dragons
Comments
Name
Date
Purpouse
Libraries
iostream
cstdlib
ctime
iomanip
fstream
string
cmath
Global Constant
None
Function prototypes
hMenu()
dMenu()
aviMenu()
avmMenu()
txtMenu()
txtInput()
txtOutput()
mutliRoll()
autoOrder()
aMod(aScore)
proMod(level)
wpnDmg(wpnDie)
d20()
d12()
d10()
d8()
d6()
d4()
readDice()
copy()
sort()
prntAry()
readI()
sortI()
printI()
main()
a
Main
Home Menu
hMenu()
gives user menu
options
user selects
menu option
option 1
Yes
Dice rolls menu
dMenu()
user selects die
calls a die
function
*d20(), d12(),
d10(), d8(), d6(),
or d4()*
OR multiRoll()
repeat?
No
Yes
b
program ends
c
Attack processed
and shown
Is monster
dead?
Yes
Display
total damage
average damage
number of hits
No
d
option 2
Attack Rolls vs Input
Menu
aviMenu()
user inputs
character and
opponent info
calls processing
functions
*aMod(aScore)
proMod(level)
wpnDmg(wpnDie)*
option 3
Auto Attack vs
Monster Menu
avmMenu()
user inputs
character and
monster info
option 4
Text file input/output
txtMenu()
user inputs
option selection
New character sheet
txtOutput()
User inputs
character name
& file is made
option 5
Player initiative order
autoOrder()
option 6
user inputs names
and initiatives
calls processing
functions
*readI()
sortI()
printI()*
Quick input w/
text file
txtInput()
user validates file
is prepared
and inputs
opponent AC
Attack processed
and shown

Pseudo Code

```
//Personal info
//Libraries
//Function Prototypes
        //Menu Functions
        //Combat score functions
        //Dice functions

//Main
        //Start Seed
        //Start home Menu function

//Home Menu
        //Give player options
        //Map options to menu functions
        //Re-do if invalid option

//Dice Menu
        //Give player dice options
        //Map options to dice functions
        //Re-do if invalid option
        //Ask player if they want to roll again

//Attack VS Input Menu
        //Declare variables
        //Ask player for character class
                //Ask for physical input based on class
                        //Check score validity
                //Check class validity
        //Character level input
                //Validate level input
        //Offer weapon options based on class
                //Validate input
        //AC Input
                //Validate input
        //Process attack using combat score functions
                //Display hit or miss and combat values
        //Ask player if they want to roll again

//Auto Attack VS Monster
        //Declare variables
        //Ask player for character class
                //Ask for physical input based on class
                        //Check score validity
                //Check class validity
        //Character level input
                //Validate level input
```

```
        //Offer weapon options based on class
                //Validate input
        //Offer monster combat options
                //Input monster stats into combat function
        //Process attacks until monster is dead
                //Display hit or miss and combat values
        //Display how many hits it took to kill the monster
          and average damage

//Text input/output Menu
        //Give user input/output options
                //Validate user input
        //Go to input/output function based on choice

//Text output function
        //Prompt user for a character name
        //Generate random stats and age for
          the character in text file

//Text input function
        //Prompt user to create file
                //Ensure that file is created
        //Prompt user for opponent AC
        //Input file info and AC into combat function
        //Process attack using functions
                //Display results
        //Ask player if they want to attack again

//Multi-Dice fuction
        //Run function to recive dice inputs
        //Run function to alter inputs
        //Output modified inputs

//Auto Order
        //Recive player inputs and names as inputs
        //Order both arrays based on Initiative order
        //Output info

//Array functions
        //Modify arrays in some way to support other functions
        //Output array data

//Character stat processing functions
        //Turn user inputs into attack modifiers or dice rolls

 //Dice functions
        //Returns a random value within the dice range
```

Screenshots of Outputs

```
Welcome to the Dungeons and Dragons combat roll sim!
Select an option:
        1. General Dice Rolls
        2. Attack Rolls VS Input
        3. Auto Attack VS Monsters
        4. Text file input/output
        5. Player Initiative Auto Order
        6. Exit
```

```
Please select a die to roll:
        1. d20
        2. d12
        3. d10
        4. d8
        5. d6
        6. d4
        7. Multiple Dice
```

```
How many players are in your party?
4
Enter a player's name...
Steve
Now, enter that player's initiative.
24
```

Name	Int
Charlie	26
Steve	24
Carol	9
Stan	8

```
Is your character a fighter or a rogue?
        1. Fighter
        2. Rogue
1

What is your stregth score? (1-20)
***10 is average, 20 is superhuman, 1 is paper
14

What is your character's level? (1-20)
20

Select a weapon:
        1. Flail (1d8 damage)
        2. Glaive (1d10 damage)
        3. Battleaxe (1d12 damage)
```

Program

```
/*
 * File:   main.cpp
 * Author: Jason Wilmot
 * Created on Febuary 14, 2021
 * Purpose:  PROJECT II FINAL
 */

//Libraries
#include <iostream>  //I/O Library
#include <cstdlib>   //Random Number Gererator
#include <ctime>     //Time to set the seed
#include <iomanip>
#include <fstream>
#include <string>
#include <cmath>
#include <vector>  //vectors<>
using namespace std;

//Global Constants - Math/Physics Constants, Conversions,
//                   2-D Array Dimensions
const int COLMAX = 80;

//Function Prototypes
//Menu functions
void hMenu();   //Home Menu
void dMenu();   //General Dice roll menu
void aviMenu(); //Attack vs Input menu
void avmMenu(); //Attack vs Monster menu
void txtMenu(); //Text file export/import menu
void txtInput();    //Text file input
void txtOutput();   //Text file output
void multiRoll();   //Roll multiple dice
void autoOrder();   //Auto order initiative
```

```
//Combat score functions
short int aMod(unsigned short int aScore);     //Converts an ability score to an ability modifyer
unsigned short int proMod(unsigned short int level);    //Uses level to get proficiency mod
unsigned short int wpnDmg(unsigned short int wpnDie);  //Weapon damage based on die type
float wpnDmg(unsigned short int wpnDie);    //Float for overloading

//Dice functions
unsigned short int d20();   //d20 dice roll
unsigned short int d12();   //d12 dice roll
unsigned short int d10();   //d10 dice roll
unsigned short int d8();    //d8 dice roll
unsigned short int d6();    //d6 dice roll
unsigned short int d4();    //d4 dice roll

//Vector/Array utility functions
void readDice(vector<int> &, vector<int> &, int);
void copy(vector<int>, vector<int>,int [][COLMAX], int);
void sort(int [][COLMAX], int);
void prntAry(const int [][COLMAX], int);

void readI(char [][COLMAX], int [], int);
void sortI(char [][COLMAX], int [], int);
void printI(char [][COLMAX], int [], int);

int main(int argc, char** argv) {           //Start seed and start home menu function
    //Initialize the Random Number Seed
    srand(static_cast<unsigned int>(time(0)));

    //Output data
    hMenu();

    return 0;
}
```

```
void hMenu(){                                                                   //Home Menu
    unsigned short int menu1;
    while (menu1 > 5 || menu1 < 1) {
    cout << "Welcome to the Dungeons and Dragons combat roll sim!" << endl;
    cout << "Select an option:" << endl;
    cout << "\t1. General Dice Rolls" << endl;
    cout << "\t2. Attack Rolls VS Input" << endl;
    cout << "\t3. Auto Attack VS Monsters" << endl;
    cout << "\t4. Text file input/output" << endl;
    cout << "\t5. Player Initiative Auto Order" << endl;
    cout << "\t6. Exit" << endl;
    cin >> menu1;

        switch(menu1){                                                          //Outputs
            case 1: dMenu(); break;
            case 2: aviMenu(); break;
            case 3: avmMenu(); break;
            case 4: txtMenu(); break;
            case 5: autoOrder(); break;
            case 6: break;
        }

        if (menu1 > 6 || menu1 < 1)                                             //Re-do if
invalid option
            cout << endl << "PLEASE SELECT A VALID OPTION" << endl;
    }
}

void dMenu(){                                                                   //Dice Menu
    unsigned short int menu1;
    char menu2;
    do{

        while (menu1 > 6 || menu1 < 1) {
```

```
            cout << "Please select a die to roll:" <<endl;
            cout << "\t1. d20" << endl;
            cout << "\t2. d12" << endl;
            cout << "\t3. d10" << endl;
            cout << "\t4. d8" << endl;
            cout << "\t5. d6" << endl;
            cout << "\t6. d4" << endl;
            cout << "\t7. Multiple Dice" << endl;
            cin >> menu1;

            switch(menu1){                                                  //Outputs
                case 1: cout << "Rolling 1d20...  \nYou got " << d20() << "!"; break;
                case 2: cout << "Rolling 1d12...  \nYou got " << d12() << "!"; break;
                case 3: cout << "Rolling 1d10...  \nYou got " << d10() << "!"; break;
                case 4: cout << "Rolling 1d8...  \nYou got " << d8() << "!"; break;
                case 5: cout << "Rolling 1d6...  \nYou got " << d6() << "!"; break;
                case 6: cout << "Rolling 1d4...  \nYou got " << d4() << "!"; break;
                case 7: multiRoll(); exit(0); break;
            }

            if (menu1 > 7 || menu1 < 1)                                     //Re-do if
invalid option
                cout << endl << "PLEASE SELECT A VALID OPTION" << endl;
        }
    cout << endl << "Would you like to roll another die? (Y/N)" << endl;        //Ask if player
wants to roll again
    menu1 = 0;
    cin >> menu2;
    } while (menu2 == 'Y' || menu2 == 'y');
}

void aviMenu(){                                                             //Attack VS Input
    //Declare Variables
    unsigned short int Class;   //Class, 1 is figher, 2 is rogue.
    unsigned short int aScore;  //Ability score, used in attack rolls.
```

```
    unsigned short int level;   //Character level
    unsigned short int AC;      //Armor Class
    short int toHit;   //The "To Hit" value used to determine if a character lands an attack.
    unsigned short int wpnMenu; //Used for selecting an option in the weapons menu.
    unsigned short int wpnDie;  //Serves as an input for the wpnDmg function
    char again;                 //Tests if the user wants to attack again.
    short int dmg;  //Damage


    //Initialize Variables
    do {                                                                    //Class & ability
score checker
        cout << endl << "Is your character a fighter or a rogue?" << endl;
        cout << "\t1. Fighter" << endl;
        cout << "\t2. Rogue" << endl;
        cin >> Class;


        if (Class == 1) {                                                   //Strength input
for fighters
            do {
                cout << endl << "What is your stregth score? (1-20)" << endl;
                cout << "***10 is average, 20 is superhuman, 1 is paper" << endl;
                cin >> aScore;
                if (aScore > 20 || aScore < 1)  //Str score validity check
                    cout << endl << "PLEASE INPUT A VALID NUMBER" << endl;
            } while (aScore > 20 || aScore < 1);
        }
        else if (Class == 2){                                               //Dexterity input
for rigues
            do {
                cout << endl << "What is your dexterity score? (1-20)" << endl;
                cout << "***10 is average, 20 is superhuman, 1 a rock" << endl;
                cin >> aScore;
                if (aScore > 20 || aScore < 1)  //Dex score validity check
                    cout << endl << "PLEASE INPUT A VALID NUMBER" << endl;
            } while (aScore > 20 || aScore < 1);
        }
```

```
        else    //Class validity check
            cout << endl << "PLEASE SELECT A VALID OPTION" << endl;
    } while (Class < 1 || Class > 2);


    do {                                                                    //Character level
input
        cout << endl << "What is your character's level? (1-20)" << endl;
        cin >> level;
        if (level > 20 || level < 1)    //Level input check
            cout << endl << "PLEASE INPUT A VALID NUMBER" << endl;
    } while (level > 20 || level < 1);


    if (Class == 1) {                                                       //Fighter weapon
options
        do {
            cout << endl << "Select a weapon:" << endl;
            cout << "\t1. Flail (1d8 damage)" << endl;
            cout << "\t2. Glaive (1d10 damage)" << endl;
            cout << "\t3. Battleaxe (1d12 damage)" << endl;
            cin >> wpnMenu;
            switch (wpnMenu){
                case 1: wpnDie = 8; break;
                case 2: wpnDie = 10; break;
                case 3: wpnDie = 12; break;
            }
        if (wpnMenu > 3 || wpnMenu < 1) //Validity check
            cout << endl << "PLEASE SELECT A VALID OPTION" << endl;
        } while (wpnMenu > 3 || wpnMenu < 1);
    }
    else {                                                                  //Rogue weapon
options
        do {
            cout << endl << "Select a weapon:" << endl;
            cout << "\t1. Dagger (1d4 damage)" << endl;
```

```
        cout << "\t2. Shortbow (1d6 damage)" << endl;
        cout << "\t3. Rapier (1d8 damage)" << endl;
        cin >> wpnMenu;
        switch (wpnMenu){
            case 1: wpnDie = 4; break;
            case 2: wpnDie = 6; break;
            case 3: wpnDie = 8; break;
        }
    if (wpnMenu > 3 || wpnMenu < 1) //Validity check
        cout << endl << "PLEASE SELECT A VALID OPTION" << endl;
    } while (wpnMenu > 3 || wpnMenu < 1);
}

do {                                                                    //AC input
    cout << endl << "What is the Armor Class (AC) of your opponent?" << endl;
    cout << "***This is the number to need to beat to land a hit." << endl;
    cout << "***EX: 10 is no armor, 18 is heavy armor." << endl;
    cin >> AC;

    if (AC > 26)    //Validity check
        cout << "That'll be a bit too hard to hit, try a lower number." <<endl;

} while (AC > 26);

do {                                                                    //Attack!
cout << endl << "You attack!" << endl;
toHit = d20() + aMod(aScore) + proMod(level);
if (toHit >= AC){                                                       //Hit or Miss
    cout << "You hit with an " << toHit << "!" << endl;
    dmg = wpnDmg(wpnDie) + aMod(aScore);
    if (dmg < 1)
        dmg = 1;
    cout << "You deal " << dmg << " damage!" << endl;
}
```

```
    else
        cout << "You missed with an " << toHit << "..." << endl;

    cout << endl << "Attack again? (Y/N)"<< endl;                                  //Again?
    cin >> again;

    } while (again == 'Y' || again == 'y');
}

void avmMenu(){                                                                    //Attack VS
Monster
    unsigned short int Class;   //Class, 1 is figher, 2 is rogue.
    unsigned short int aScore;  //Ability score, used in attack rolls.
    unsigned short int level;   //Character level
    unsigned short int AC;      //Armor Class
    short int toHit;   //The "To Hit" value used to determine if a character lands an attack.
    unsigned short int wpnMenu; //Used for selecting an option in the weapons menu.
    unsigned short int monMenu; //Used to navigate the monster menu.
    unsigned short int wpnDie;  //Serves as an input for the wpnDmg function
    char again;                 //Tests if the user wants to attack again.
    float avrgDmg;              //Average damage
    short int HP;      //Hit points
    short int dmg;     //Damage
    float ttlDmg;  //Total damage
    string monName;             //Monster name

    //Initialize Variables
    do {                                                                           //Class & ability
score checker
        cout << endl << "Is your character a fighter or a rogue?" << endl;
        cout << "\t1. Fighter" << endl;
        cout << "\t2. Rogue" << endl;
        cin >> Class;

        if (Class == 1) {                                                          //Strength input
for fighters
```

```
            do {
                cout << endl << "What is your stregth score? (1-20)" << endl;
                cout << "***10 is average, 20 is superhuman, 1 is paper" << endl;
                cin >> aScore;
                if (aScore > 20 || aScore < 1)  //Str score validity check
                    cout << endl << "PLEASE INPUT A VALID NUMBER" << endl;
            } while (aScore > 20 || aScore < 1);
        }
        else if (Class == 2){                                                   //Dexterity input
for rigues
            do {
                cout << endl << "What is your dexterity score? (1-20)" << endl;
                cout << "***10 is average, 20 is superhuman, 1 a rock" << endl;
                cin >> aScore;
                if (aScore > 20 || aScore < 1)  //Dex score validity check
                    cout << endl << "PLEASE INPUT A VALID NUMBER" << endl;
            } while (aScore > 20 || aScore < 1);
        }
        else    //Class validity check
            cout << endl << "PLEASE SELECT A VALID OPTION" << endl;
    } while (Class < 1 || Class > 2);



    do {                                                                        //Character level
input
        cout << endl << "What is your character's level? (1-20)" << endl;
        cin >> level;
        if (level > 20 || level < 1)    //Level input check
            cout << endl << "PLEASE INPUT A VALID NUMBER" << endl;
    } while (level > 20 || level < 1);



    if (Class == 1) {                                                           //Fighter weapon
options
        do {
            cout << endl << "Select a weapon:" << endl;
```

```
            cout << "\t1. Flail (1d8 damage)" << endl;
            cout << "\t2. Glaive (1d10 damage)" << endl;
            cout << "\t3. Battleaxe (1d12 damage)" << endl;
            cin >> wpnMenu;
            switch (wpnMenu){
                case 1: wpnDie = 8; break;
                case 2: wpnDie = 10; break;
                case 3: wpnDie = 12; break;
            }
        if (wpnMenu > 3 || wpnMenu < 1) //Validity check
            cout << endl << "PLEASE SELECT A VALID OPTION" << endl;
        } while (wpnMenu > 3 || wpnMenu < 1);
    }
    else {                                                          //Rogue weapon
options
        do {
            cout << endl << "Select a weapon:" << endl;
            cout << "\t1. Dagger (1d4 damage)" << endl;
            cout << "\t2. Shortbow (1d6 damage)" << endl;
            cout << "\t3. Rapier (1d8 damage)" << endl;
            cin >> wpnMenu;
            switch (wpnMenu){
                case 1: wpnDie = 4; break;
                case 2: wpnDie = 6; break;
                case 3: wpnDie = 8; break;
            }
        if (wpnMenu > 3 || wpnMenu < 1) //Validity check
            cout << endl << "PLEASE SELECT A VALID OPTION" << endl;
        } while (wpnMenu > 3 || wpnMenu < 1);
    }


    do {                                                            //Monster
selection
        cout << endl << "What monster will you be fighting?" << endl;
        cout << "\t1. Kobold" << endl;                             //Add monsters
```

```
        cout << "\t2. Goblin" << endl;
        cout << "\t3. Ogre" << endl;
        cin >> monMenu;
        switch (monMenu){                                                    //set
name, HP, and AC based on monster
            case 1: monName = "Kobold"; HP = 5; AC = 12; break;
            case 2: monName = "Goblin"; HP = 7; AC = 15; break;
            case 3: monName = "Ogre"; HP = 59; AC = 11; break;
        }
        if (monMenu > 3 || monMenu < 1)    //Validity check
            cout << "PLEASE SELECT A VALID OPTION" << endl;
    } while (monMenu > 3 || monMenu < 1);

    cout << "Combat against the " << monName << " begins!" << endl;

    int n;
    ttlDmg = 0;

    for (n = 0; HP > 0; n++){                                  //Attack!
        cout << "You attack!" << endl;
        toHit = d20() + aMod(aScore) + proMod(level);
        if (toHit >= AC){                                                  //Hit or Miss
            cout << "You hit with an " << toHit;
            dmg = wpnDmg(wpnDie) + aMod(aScore);
            if (dmg < 1)
                dmg = 1;
            cout << ", dealing " << dmg << " damage!" << endl;
        }
        else {
            dmg = 0;
            cout << "You missed with an " << toHit << "..." << endl;
        }
        HP = HP - dmg;
        ttlDmg = ttlDmg + dmg;
    }
```

```
    avrgDmg = (ttlDmg/n);



    cout << "You killed the " << monName << " in " << n << " hits!" << endl;

    cout << "You had an average damage of " << fixed << showpoint << setprecision(2) << avrgDmg
<< "." << endl;

}



void txtMenu(){                                                                    //File
input/output menu

    char menu1;



    do {

        cout << endl << "Text file input/output" << endl;

        cout << "\t1. New character sheet as text file." << endl;

        cout << "\t2. Quick input vs AC w/ text file" << endl;

        cin >> menu1;

        if (menu1 > '2' || menu1 < '1')

            cout << endl << "PLEASE SELECT A VALID OPTION" << endl;

    } while (menu1 > '2' || menu1 < '1');



    switch (static_cast<int>(menu1)){

        case 49: txtOutput(); break;

        case 50: txtInput(); break;

    }

}



void txtOutput() {                                                                 //Function
outputing a text file

    //Variables

    string name;    //Character name



    //File set up

    ofstream outputFile;

    outputFile.open("Character Sheet.txt");
```

```
    cout << "Input your character's full name, and a stat sheet will be generated." << endl;
    cin.ignore();
    getline(cin, name);

    //File outputs
    outputFile << name << ", age " << 15 + pow(d6(), 2) << endl;
    outputFile << "Strength:" << d6() + d6() + d6() << endl;
    outputFile << "Dexterity:" << d6() + d6() + d6() << endl;
    outputFile << "Constitution:" << d6() + d6() + d6() << endl;
    outputFile << "Wisdom:" << d6() + d6() + d6() << endl;
    outputFile << "Intelligence:" << d6() + d6() + d6() << endl;
    outputFile << "Charisma:" << d6() + d6() +d6() << endl;

    //Completion notice
    cout << "The stat sheet is ready!" << endl;

    //close file
    outputFile.close();
}

void txtInput() {                                                    //Fuction using
text file inputs
    //Variables
    unsigned short int aScore;  //ability score
    unsigned short int level;  //character level
    unsigned short int wpnDie;  //weapon die
    short int toHit;
    bool ready; //ready to continue
    unsigned short int AC;
    char again;

    //Prompt user to create file
    cout << "This section allows you to used saved character stats for an attack vs input." <<
endl;
    cout << "1. Create a text file named 'Saved_Stats.txt'." << endl;
```

```
    cout << "2. Input your character's strength or dexterity on the first line." << endl;

    cout << "3. Input your character's level on the second line." << endl;

    cout << "4. Enter the number of sides on your weapon die on the third line. (d12 = 12, d10 =
10, etc.)" << endl;

    cout << endl << "Is the file ready? 1 = Yes, 0 = No" << endl;

    do {

        cin >> ready;

        if (ready != 1)

            cout << "It's ok, take your time." << endl;

    } while (ready != 1);




    //File set up

    ifstream inputFile;

    inputFile.open("Saved_Stats.txt");


    cout << "Getting data from file..." << endl;


    //File inputs

    inputFile >> aScore;

    inputFile >> level;

    inputFile >> wpnDie;

    inputFile.close();


    cout << "Got it!" << endl;


     do {                                                                    //AC input

        cout << endl << "What is the Armor Class (AC) of your opponent?" << endl;

        cout << "***This is the number to need to beat to land a hit." << endl;

        cout << "***EX: 10 is no armor, 18 is heavy armor." << endl;

        cin >> AC;


        if (AC > 26)    //Validity check

            cout << "That'll be a bit too hard to hit, try a lower number." <<endl;
```

```
    } while (AC > 26);

    do {                                                                    //Attack!
    cout << "You attack!" << endl;
    toHit = d20() + aMod(aScore) + proMod(level);
    if (toHit >= AC){                                                       //Hit or Miss
        cout << endl << "You hit with an " << toHit << "!" << endl;
        cout << "You deal " << wpnDmg(wpnDie) + aMod(aScore) << " damage!" << endl;
    }
    else
        cout << "You missed with an " << toHit << "..." << endl;

    cout << endl << "Attack again? (Y/N)"<< endl;                           //Again?
    cin >> again;

    } while (again == 'Y' || again == 'y');
}

void multiRoll(){
    //Variables
    int ROW = 80;
    int array[ROW][COLMAX];
    vector<int> dice(0), roll(0);

    //Process
    int dieIn;
    cout << "Input the number of dice you want to roll." << endl;
    cin >> dieIn;

    readDice(dice, roll, dieIn);
    copy(dice, roll, array, dieIn);
    sort(array, dieIn);
    prntAry(array, dieIn);
}
```

```
void readDice(vector<int> &dice, vector<int> &roll, int dieIn){
    cout << "Input the number of sides on each die." << endl;
    cout << "Unsupported dice will output zero." << endl;
    int temp;
    for (int i = 0; i < dieIn; i++){
        cin >> temp;
        dice.push_back(temp);
        roll.push_back(wpnDmg(temp));
    }
}

void copy(vector<int> dice, vector<int> roll,int array[][COLMAX], int dieIn){
    for (int i = 0; i < dieIn; i++){
            array[i][0]=dice[i];
            array[i][1]=roll[i];
    }
}

void sort(int array[][COLMAX], int dieIn){
    int i, j, imin;
    int size = dieIn;
   for(i = 0; i < size - 1; i++) {
      imin = i;                                    //get index of minimum data
      for(j = i + 1; j < size; j++)
         if(array[j] < array[imin])
            imin = j;                              //placing in correct position
         swap(array[i], array[imin]);
   }
}

void prntAry(const int array[][COLMAX], int dieIn){
    int w = 10;
    cout << setw(w) << "Die" << setw(w) << "Roll" << endl;
```

```
    for (int i = 0; i < dieIn; i++)
        cout << setw(w) << array[i][0] << setw(w) << array[i][1] << endl;
}

void autoOrder(){
    //Declare Variables
    //Variables
    int ROW = 80;
    int plyrs;
    char names[ROW][COLMAX];
    int order[ROW];

    cout << "How many players are in your party?" << endl;
    cin >> plyrs;
    readI(names, order, plyrs);
    sortI(names, order, plyrs);
    printI(names, order, plyrs);
}

void readI(char names[][COLMAX], int order[], int plyrs){
    for (int i = 0; i <  plyrs; i++){
        cout << "Enter a player's name..." << endl;
        cin >> names[i];
        cout << "Now, enter that player's initiative." << endl;
        cin >> order[i];
    }
}

void sortI(char names[][COLMAX], int order[], int plyrs){
    int minInx, minVal;
    for (int start = 0; start < plyrs - 1; start++){
        minInx = start;
        minVal = order[start];
        for (int index = start + 1; index < plyrs; index++){
```

```
            if (order[index] > minVal){
                minVal = order[index];
                minInx = index;
            }
        }
        swap(order[minInx], order[start]);
        swap(names[minInx], names[start]);
    }
}

void printI(char names[][COLMAX], int order[], int plyrs){
    int w = 10;
    cout << setw(w) << "Name" << setw(w) << "Int" << endl;
    for (int i = 0; i < plyrs; i++)
        cout << setw(w) << names[i] << setw(w) << order[i] << endl;
}

short int aMod(unsigned short int aScore){  //Convert ability score into ability mod
    return ((aScore/2) - 5);
}

unsigned short int proMod(unsigned short int level){    //Convert level into proficiency mod
    return (((level - 1)/4)+2);
}

unsigned short int wpnDmg(unsigned short int wpnDie){   //Converts die type into damage
    switch (wpnDie) {
        case 12: return d12();
        case 10: return d10();
        case 8: return d8();
        case 6: return d6();
        case 4: return d4();
        default: return 0;
    }
```

```
}

float wpnDmg(unsigned short int wpnDie){   //Converts die type into damage
    switch (wpnDie) {
        case 12: return d12();
        case 10: return d10();
        case 8: return d8();
        case 6: return d6();
        case 4: return d4();
        default: return 0;
    }
}

unsigned short int d20(){   //Rolls a 20 sided die
    unsigned short int d20;
    d20=rand()%20+1;    //[1,20]
    return d20;
}

unsigned short int d12(){   //Rolls a 12 sided die
    unsigned short int d12;
    d12=rand()%12+1;    //[1,12]
    return d12;
}

unsigned short int d10(){   //Rolls a 10 sided die
    unsigned short int d10;
    d10=rand()%10+1;    //[1,10]
    return d10;
}

unsigned short int d8(){    //Rolls an 8 sided die
    unsigned short int d8;
    d8=rand()%8+1;    //[1,8]
```

```
    return d8;
}

unsigned short int d6(){    //Rolls a 6 sided die
    unsigned short int d6;
    d6=rand()%6+1;    //[1,6]
    return d6;
}

unsigned short int d4(){    //Rolls a 4 sided die
    unsigned short int d4;
    d4=rand()%4+1;    //[1,4]
    return d4;
}
```