Project 2

<Desperado>

CSC-5 41202

Name: Rothman, Alexander

Date: 02/10/16

Introduction

Title: Desperado

Desperado is a simple RPG built on the concept of hangman. The player assumes the role of a bounty hunter in a wild west setting. They are initially shown a menu allowing them to play the game, display a help file or quit. Should the player choose to play they will be asked to start a new game or to load a saved game. Once the game begins the player is given various activities that they can engage in. These activities include shopping for equipment, healing after a battle, selecting a new bounty to hunt, and tracking down their current bounty. When the player decides to hunt a bounty the game shifts into battle mode. In the battle the player and their bounty take turns playing hangman against each other until one of them has lost all their hit points. If the player wins the battle they are returned to the game menu and rewarded. Otherwise the game ends with a game over message.

Summary

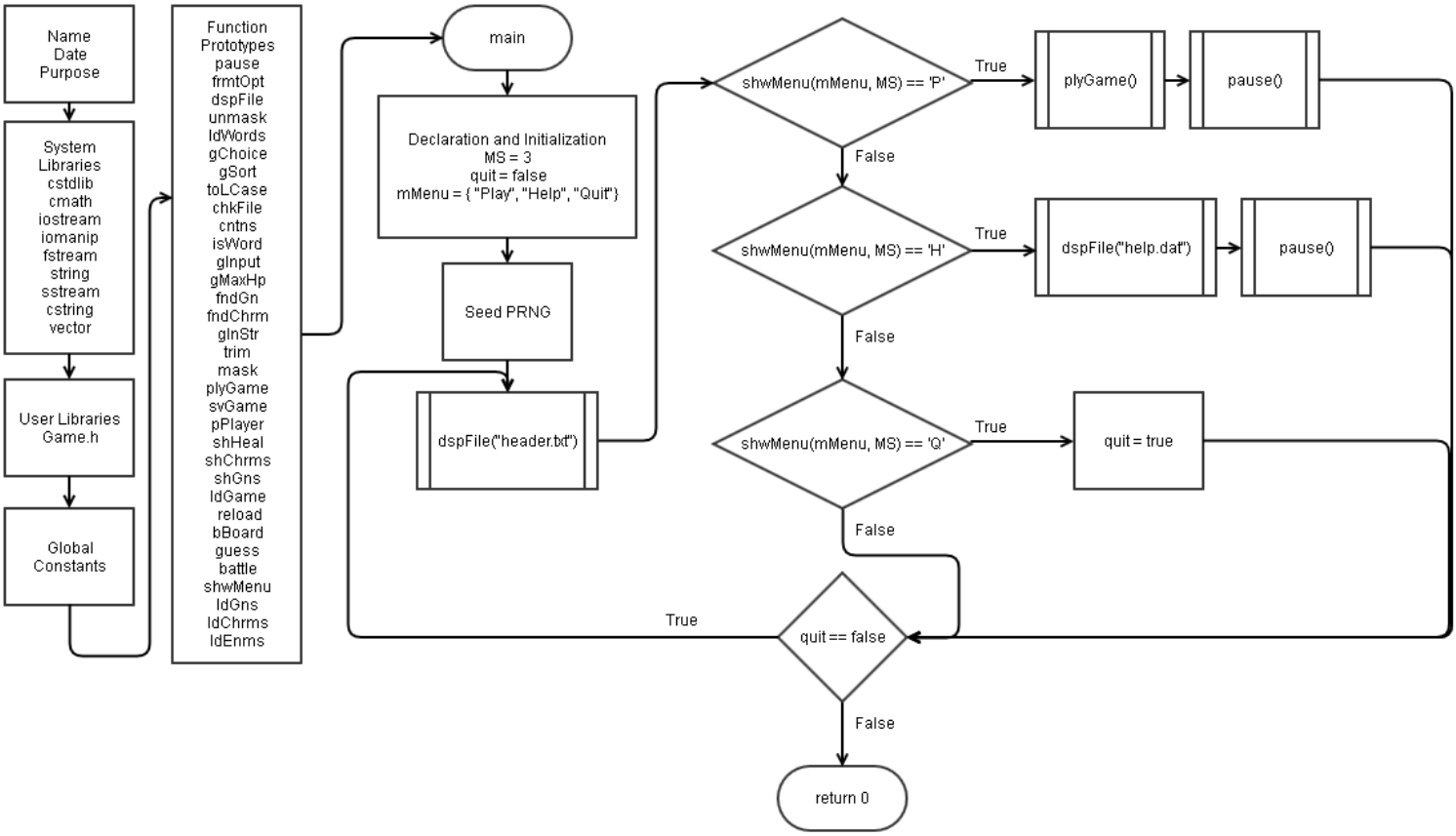
Project Size: 1168 Lines
Number of variables: 3 (in main) about 78 (all functions)
Number of functions: 33

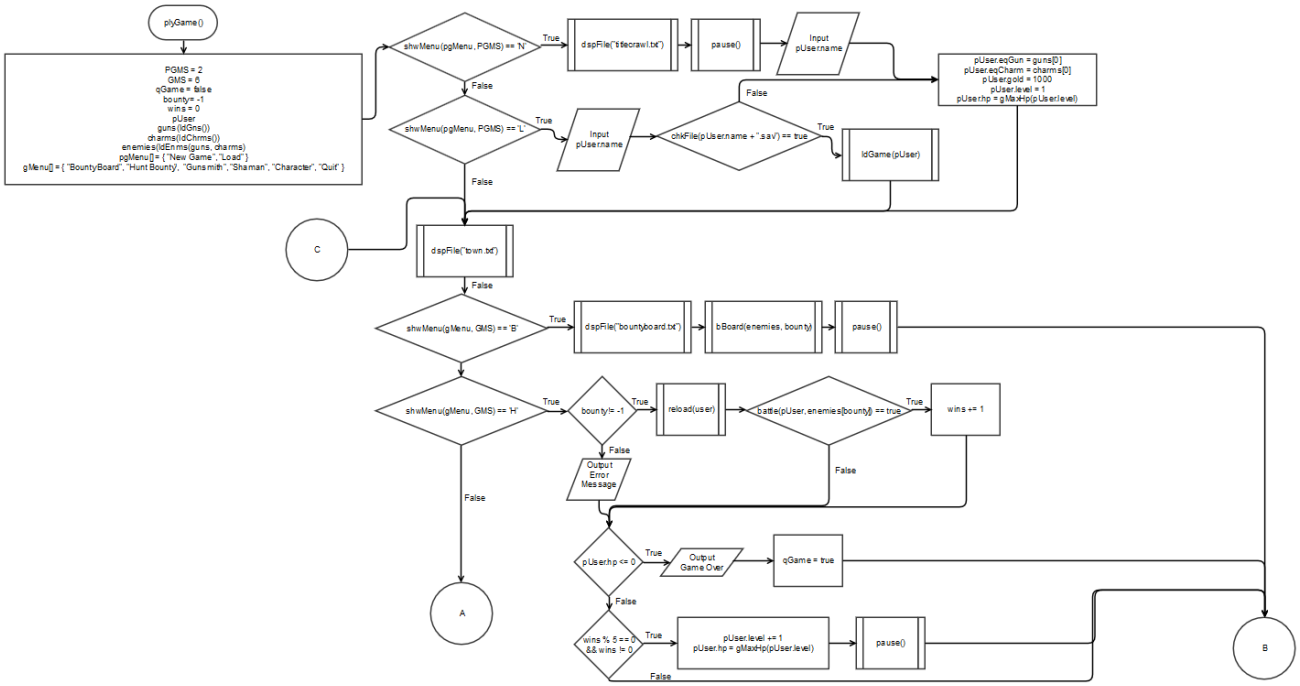
With this game my goal was to expand hangman into a game closer to one I would actually play. I had originally had the idea for the game before project 1 but we hadn't covered enough of the material to make it reasonable to create (structured data etc). The game functions, saves and loads data, allows for the creation of multiple characters, and provides a bit of flavor text for most interactions. If I had more time I would have liked to improve the programmatic guessing that occurs in the game. Currently it uses only the letter frequencies for words in English to make each guess. I primarily relied on the textbooks, my previous project, and cplusplus.com for reference information.

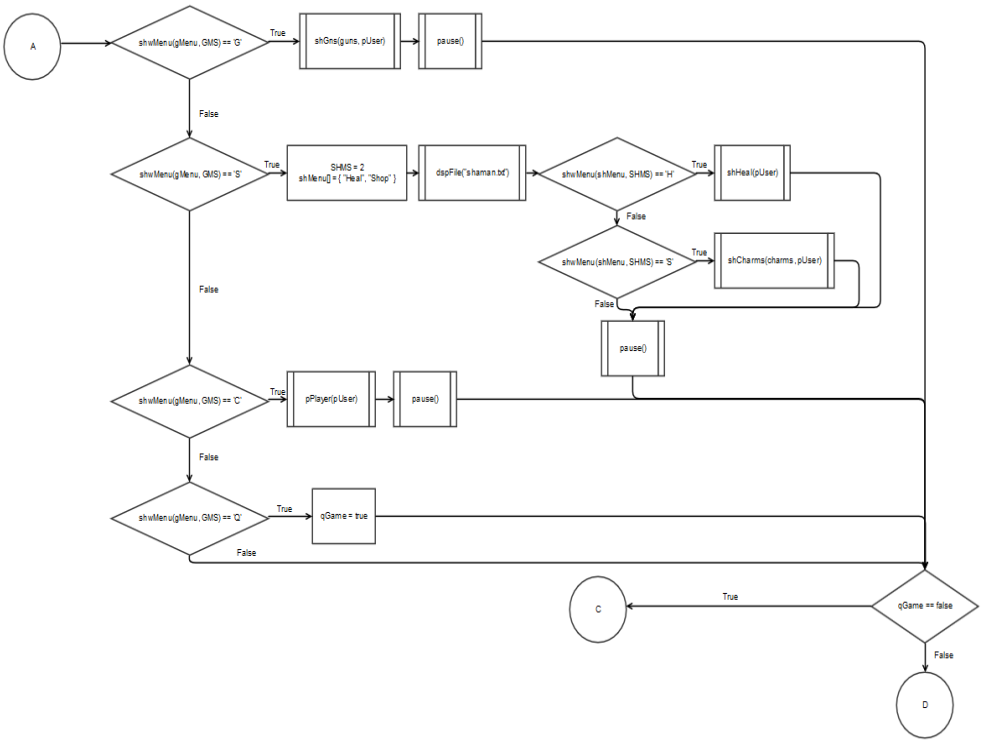
Description

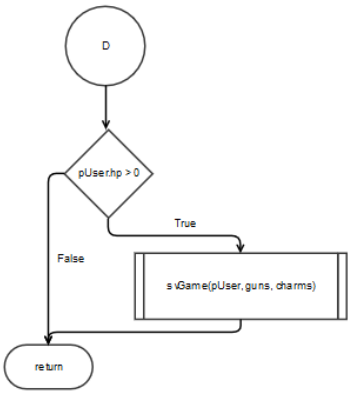
The major point of this program is to utilize all the major concepts we have learned up to this point and use them to create a game.

Flowchart


Name
Date
Purpose
System Libraries
cstdlib
cmath
iostream
iomanip
fstream
string
sstream
cstring
vector
User Libraries
Game.h
Global Constants
Function Prototypes
pause
frmtOpt
dspFile
unmask
ldWords
gChoice
gSort
toLCase
chkFile
cntns
isWord
gInput
gMaxHp
fndGn
fndChrm
gInStr
trim
mask
plyGame
svGame
pPlayer
shHeal
shChrms
shGns
ldGame
reload
bBoard
guess
battle
shwMenu
ldGns
ldChrms
ldEnms
main
Declaration and Initialization
MS = 3
quit = false
mMenu = { "Play", "Help", "Quit"}
Seed PRNG
dspFile("header.txt")
shwMenu(mMenu, MS) == 'P'
True
plyGame()
pause()
False
shwMenu(mMenu, MS) == 'H'
True
dspFile("help.dat")
pause()
False
shwMenu(mMenu, MS) == 'Q'
True
quit = true
False
quit == false
True
False
return 0


plyGame()
PGMS = 2
GMS = 6
qGame = false
bounty= -1
wins = 0
pUser
guns(ldGns())
charms(ldChrms())
enemies(ldEnms(guns, charms)
pgMenu[] = { "New Game", "Load" }
gMenu[] = { "BountyBoard", "Hunt Bounty", "Gunsmith", "Shaman", "Character", "Quit" }
shwMenu(pgMenu, PGMS) == 'N'
True
dspFile("sfecrawl.txt")
pause()
Input pUsername
pUser.eqGun = guns[0]
pUser.eqCharm = charms[0]
pUser.gold = 1000
pUser.level = 1
pUser.hp = gMaxHp(pUser.level)
False
shwMenu(pgMenu, PGMS) == 'L'
True
Input pUsername
chkFile(pUsername + ".sav") == true
True
False
ldGame(pUser)
False
C
dspFile("town.txt")
False
shwMenu(gMenu, GMS) == 'B'
True
dspFile("bountyboard.txt")
bBoard(enemies, bounty)
pause()
shwMenu(gMenu, GMS) == 'H'
True
bounty != -1
True
reload(user)
battle(pUser, enemies[bounty]) == true
True
wins += 1
False
Output Error Message
False
False
A
pUser.hp <= 0
True
Output Game Over
qGame = true
False
wins % 5 == 0 && wins != 0
True
pUser.level += 1
pUser.hp = gMaxHp(pUser.level)
pause()
False
B


A
shwMenu(gMenu, GMS) == 'G'
True
shGns(guns, pUser)
pause()
False
shwMenu(gMenu, GMS) == 'S'
True
SHMS = 2
shMenu[] = { "Heal", "Shop" }
dspFile("shaman.txt")
shwMenu(shMenu, SHMS) == 'H'
True
shHeal(pUser)
False
shwMenu(shMenu, SHMS) == 'S'
True
shCharms(charms, pUser)
False
pause()
False
shwMenu(gMenu, GMS) == 'C'
True
pPlayer(pUser)
pause()
False
shwMenu(gMenu, GMS) == 'Q'
True
qGame = true
False
C
True
qGame == false
False
D


D
pUser.hp > 0
True
False
sVGame(pUser, guns, charms)
return

Pseudocode

```
Main:
Declare MS and set it equal to 3
Declare quit and set it to false
Declare mMenu[] and initialize it with the values
Play, Help, Quit
Seed the PRNG
Do
    Call dspFile() to display the header file
    Call shwMenu() and pass in mMenu and MS
    If shwMenu() returns P
        Call plyGame()
        Call pause()
    Else if shwMenu() returns H
        Call dspFile() to display the help file
        Call pause()
    Else if shwMenu() returns Q
        Set quit to true
While quit is false
Return 0

plyGame:
Declare PGMS and set it to 2
Declare GMS and set it to 6
Declare qGame and set it to false
Declare bounty and set it to -1
Declare wins and set it to 0
Declare pUser
Declare guns and copy the return ldGns() into it
Declare charms and copy the return of ldChrms() into
it
Declare enemies and copy the return of ldEnms() into
it
Declare pgMenu[] and initialize it with the values New
Game, Load
Declare gMenu[] and initialize it with the values
Bounty Board, Hunt Bounty, Gunsmith, Shaman,
Character, Quit
Call shwMenu() and pass in pgMenu and PGMS
If shwMenu() returns N
    Call dspFile() to display title text
```

```
    Call pause()
    Prompt the user to enter a character name
    Set pUser.name to the return of gInStr()
    Set pUser.eqGun to guns[0]
    Set pUser.eqCharm to charms[0]
    Set pUser.gold to 1000
    Set pUser.level to 1
    Set pUser.hp to the return of gMaxHp()
Else if shwMenu() returns L
    Prompt the user to enter a character name
    Set pUser.name to the return of gInStr()
    If chkFile() returns true for the input of
    pUser.name + .sav
        Call ldGame() and pass in pUser, guns, and
        charms
    Else
        Output an error message
        Call dspFile() to display title text
        Call pause()
        Set pUser.eqGun to guns[0]
        Set pUser.eqCharm to charms[0]
        Set pUser.gold to 1000
        Set pUser.level to 1
        Set pUser.hp to the return of gMaxHp()
Do
    Call dspFile() to display town.txt
    Call shwMenu() and pass in gMenu and GMS
    If shwMenu() returns B
        Call dspFile to display bountyboard.txt
        Call bBoard() and pass in enemies and bounty
        Call pause()
    Else if shwMenu() returns H
        If bounty is not equal to -1
            Call reload() and pass in pUser
            Call battle() and pass in pUser and
            enemies[bounty]
            If battle() returns true
                increment wins by 1
        Else
            Output an error message
        If pUser.hp is less than or equal to 0
            Output a game over message
```

```
            Set qGame to true
        Else
            If wins mod 5 is 0 and wins does not equal
            0
                Increment pUser.level
                Output a level up message
                Output how much your HP increased by
                Set pUser.hp to gMaxHp()
        Call pause()
    Else if shwMenu() returns G
        Call shGns() and pass in guns and pUser
        Call pause()
    Else if shwMenu() returns S
        Declare SHMS and set it to 2
        Declare shMenu[] and initialize it with the
        values Heal and Shop
        Call dspFile() to display shaman.txt
        Call shwMenu() and pass in shMenu[] and SHMS
        If shwMenu() returns H
            Call shHeal() and pass in pUser
        Else if shwMenu() returns S
            Call shChrms() and pass in charms and
            pUser
        Call pause()
    Else if shwMenu() returns C
        Call pPlayer() and pass in pUser
        Call pause()
    Else if shwMenu() returns Q
        Set qGame to true
While qGame is false
If pUser.hp is greater than 0
    Call svGame() and pass in pUser, guns, and charms
```

Major Variables

Type	Variable Name	Description	Location
bool	quit	Determines whether or not the program should exit	main()
	btlOver	Determines whether or not a battle is completed	battle()
	r	The return value from a battle	battle()
	qGame	Determines whether or not to quit the game and return to the main menu	plyGame()
char	used[]	The array of characters already used by the player in guessing	battle()
short	gCount	The number of guesses the player currently has remaining	battle()
	usedPos	The current position in the used character array	battle()

	wins	The number of battles the player has currently won	plyGame()
int	MS	The length of the main menu	main()
	WS	The size of the word list	battle()
	ALPHAS	The size of the latin alphabet	battle()
	PGMS	The size of the play game menu	plyGame()
	GMS	The size of the game menu	plyGame()
	bounty	The currently selected bounty	plyGame()
string	mMenu[]	Array containing main menu options	main()
	uWord	A user input word	battle()
	oWord	A computer selected word	battle()
	mWord	A masked word	battle()
	words[]	The loaded word list	battle()
	pgMenu[]	Array containing play game menu options	plyGame()
	gMenu[]	Array containing game menu options	plyGame()
Player	pUser	Object representing the game's player	plyGame()
vector	guns	Vector containing the list of Gun objects in the game	plyGame()
	charms	Vector containing the	plyGame()

		list of Charm objects in the game	
	enemies	Vector containing the list of enemies in the game	plyGame()

C++ Constructs

Chapter	New Syntax and Keywords	Location
2	cout	Line 148
	#include directives	Line 9
	Variables and Literals	Line 72
	Identifiers and keywords	Line 79
	Integer data types	Line 137
	Character data types	Line 181
	Floating point data types	Line 511
	Boolean data types	Line 72
	Arithmetic operators	Line 513
	Comments	Line 508
	Named Constants	Line 510
3	cin	Line 528
	Type conversion	Line 374
	Formatting output	Line 622
4	Relational operators	Line 627
	If statement	Line 628

	If/Else and If/Else if	Line 753 to 757 Line 856 to 866
	Logical operators	Line 782
	Input validation	Line 625 to 632
	Ternary operator	Line 1103
	Switch statement	Line 1138 to 1149
5	Increment and Decrement operators	Line 979 Line 976
	While loop	Line 481 to 494
	Do while loop	Line 625 to 632
	For loop	Line 897 to 899
	File I/O	Line 692 to 701
6	Functions	Line 471 to 498
	Function Prototype	Line 29
	Pass by value	Line 39
	Pass by reference	Line 43
	Function call	Line 80
	Default parameters	Line 42
	Return	Line 122
7	Arrays	Line 74
	Array initialization	Line 74
	Arrays as function arguments	Line 34
	2d Arrays	Line 720
	2d Arrays as function arguments	Line 37
	Vectors	Line 1042
8	Searching	Line 835 to 843
	Sorting	Line 854 to 868
9	Pointers	Line 731
	Pointer arithmetic	Line 680
	Dynamic memory	Line 731
	Deleting dynamic memory	Line 760

10	strings	Line 779
11	structs	Line 15 to 21 of Game.h

Reference

1. C++ form Control Structures through Objects 8th Ed.
2. http://www.cplusplus.com/reference/
3. https://en.wikipedia.org/wiki/Gnome_sort

Program

```
/*
 * File:   main.cpp
 * Author: Alexander Rothman
 * Purpose: Desperado Game
 * Created on February 4, 2016, 7:33 AM
 */

//System Libraries
#include <cstdlib> //C Standard Library
#include <cmath> //Math Library
#include <iostream> //Standard I/O
#include <iomanip> //I/O Manipulation
#include <fstream> //File I/O
#include <string> //String operations
#include <sstream> //String Streams
#include <cstring> //C Strings
#include <vector> //Vectors

using namespace std;

//User Libraries
#include "Game.h" //Game object library

//Global Constants
const int ALPHAS = 26, //Alphabet size
          LS = 2; //List size

//Function Prototypes
void pause(void);
void frmtOpt(string &);
void dspFile(string);
void unmask(string &, string &, char);
void ldWrds(string *, int);
void gChoice(char [], const string [], int);
void gSort(char [], int);
void toLCase(string &);
void ldFreq(unsigned short[ALPHAS][LS]);
bool chkFile(const string &);
bool cntns(string, char);
bool isWord(string);
char gInput(void);
short gMaxHp(unsigned short = 1);
int fndGn(const vector<Gun> &, const Gun &);
```

```
int fndChrm(const vector<Charm> &, const Charm &);
string gInStr(void);
string trim(string);
string mask(string);
//Game Functions
void plyGame(void);
void pPlayer(const Player &);
void shHeal(Player &);
void shChrms(const vector<Charm> &, Player &);
void shGns(const vector<Gun> &, Player &);
void svGame(const Player &, const vector<Gun> &, const vector<Charm> &);
void ldGame(Player &, const vector<Gun> &, const vector<Charm> &);
void reload(Player &);
void bBoard(const vector<Player> &, int &);
bool guess(string, Player &);
bool battle(Player &, Player &);
char shwMenu(const string [], int);
vector<Gun> ldGns(void);
vector<Charm> ldChrms(void);
vector<Player> ldEnms(const vector<Gun> &, const vector<Charm> &);


//Begin Execution
int main(int argc, char** argv) {
    //Declaration and Initialization
    //Constants
    const int MS = 3; //The length of the menu
    //Variables
    bool quit = false; //Flag to control quitting the game
    //Collections
    string mMenu[] = {"Play", "Help", "Quit"}; //The main menu

    srand(static_cast<int> (time(0))); //Seed PRNG

    //Main Menu
    do { //While Quit is not selected
        dspFile("header.txt"); //Show the game header
        switch (shwMenu(mMenu, MS)) { //Show the menu
            case 'P': //Play Game
            {
                plyGame();
                pause();
                break;
            }
            case 'H': //Display Help
            {
                dspFile("help.dat");
                pause();
                break;
            }
            case 'Q': //Quit
            {
                quit = true;
                break;
            }
        }
    } while (!quit);

    //Exit
```

```
    return 0;
}

/*******************************************************************************/
/*********************************Check File************************************/
/*******************************************************************************/
//  Check if a file exists on the system
//Inputs
//  &path : The path to check for a file on
//Outputs
//  true if the file exists
//  false otherwise
bool chkFile(const string &path) {
    //Objects
    ifstream file; //File stream for checking the file

    file.open(path.c_str()); //Attempt to open the file
    if (file.good()) { //If the file is opened and good
        file.close(); //Close the file
        return true;
    } else { //Otherwise
        file.close(); //close the file
        return false;
    }
}

/*******************************************************************************/
/*********************************Shop Heal*************************************/
/*******************************************************************************/
//  Processing for the Shaman healing menu
//Inputs
//  &user : The game's player
void shHeal(Player &user) {
    //Constants
    const int MS = 2; //The length of the menu
    //Variables
    bool leave = false; //Whether or not to leave this menu
    //Collections
    string menu[] = {"Yes", "No"}; //The healing menu

    dspFile("shamanheal.txt"); //Display flavor text
    do { //While the player doesn't want to leave
        //Output cost of healing
        cout << "Healing costs " << (gMaxHp(user.level) - user.hp) * 10
             << " gold" << endl;
        cout << "Heal your wounds?" << endl;
        switch (shwMenu(menu, MS)) { //Display menu
            case 'Y': //If you want to heal
            {
                //Check if the player has enough gold to pay
                if (user.gold > (gMaxHp(user.level) - user.hp) * 10) {
                    //Subtract the cost from the player's total gold
                    user.gold -= (gMaxHp(user.level) - user.hp) * 10;
                    //Heal the player
                    user.hp = gMaxHp(user.level);
                } else {
                    cout << "You don't have enough gold" << endl;
                }
                break;
```

```
            }
            case 'N': //If you don't want to heal
            {
                leave = true;
                break;
            }
        }
    } while (!leave);
}

/*******************************************************************************/
/*********************************Shop Charms***********************************/
/*******************************************************************************/
//  Processing for the Shaman shopping menu
//Inputs
//  &charms : A vector containing the list of charms that exist in the game
//  &user : The game's player
void shChrms(const vector<Charm> &charms, Player &user) {
    //Variables
    char choice; //The player's choice in the menu

    dspFile("shamanshop.txt"); //Display flavor text
    cout << "INVENTORY:" << endl;
    for (int i = 0; i < charms.size(); ++i) { //Show the inventory
        //Display the charm's name
        cout << "(" << i + 1 << ") " << charms[i].name << endl;
        //Display charm attributes
        cout << '\t' << charms[i].dsc << endl;
        cout << '\t' << "DEF:  " << static_cast<int> (charms[i].def) << endl;
        cout << '\t' << "COST: " << charms[i].def * 100 << endl;
    }
    //Special leave option
    cout << "(0) Leave" << endl;
    do { //While the player doesn't want to leave
        cout << "What would you like to buy?" << endl;
        choice = gInput(); //Get the player's choice
        for (int i = 0; i < charms.size(); ++i) { //Search for the selected charm
            if ((choice - 48) == i + 1) { //If the player's choice equals one of
the options
                if (user.gold >= charms[i].def * 100) { //If the player can afford
their selection
                    //Subtract cost from the player's total gold
                    user.gold -= charms[i].def * 100;
                    //Equip the newly purchased charm
                    user.eqCharm = charms[i];
                } else { //If the player doesn't have enough gold
                    cout << "You don't have enough gold" << endl;
                }
            }
        }
    } while (choice != '0');
}

/*******************************************************************************/
/*********************************Shop Guns*************************************/
/*******************************************************************************/
//  Processing for the Gunsmith shopping menu
//Inputs
//  &guns : A vector containing the list of guns that exist in the game
```

```
//  &user : The game's player
void shGns(const vector<Gun> &guns, Player &user) {
    //Variables
    char choice; //The player's choice in the menu

    dspFile("gunsmith.txt"); //Display flavor text
    cout << "INVENTORY:" << endl;
    for (int i = 0; i < guns.size(); ++i) { //Show inventory
        //Display the gun's name
        cout << "(" << i + 1 << ") " << guns[i].name << endl;
        //Display gun attributes
        cout << '\t' << guns[i].dsc << endl;
        cout << '\t' << "ATK:  " << static_cast<int> (guns[i].atk) << endl;
        cout << '\t' << "AMMO: " << guns[i].ammo << endl;
        cout << '\t' << "COST: " << guns[i].atk * 100 << endl;
    }
    //Special leave option
    cout << "(0) Leave" << endl;
    do { //While the player doesn't want to leave
        cout << "What would you like to buy?" << endl;
        choice = gInput(); //Get the player's choice
        for (int i = 0; i < guns.size(); ++i) {
            if ((choice - 48) == i + 1) { //If the player's choice equals one of
the options
                if (user.gold >= guns[i].atk * 100) { //If the player can afford
their choice
                    //Subtract cost from the player's total gold
                    user.gold -= guns[i].atk * 100;
                    //Equip the newly purchased charm
                    user.eqGun = guns[i];
                } else { //If the player doesn't have enough gold
                    cout << "You don't have enough gold" << endl;
                }
            }
        }
    } while (choice != '0');
}

/*****************************************************************************/
/**********************************Find Gun***********************************/
/*****************************************************************************/
//  Linear search a vector for a gun
//Inputs
//  &vec : The vector to search
//  &key : The Gun value to find
//Output
//  An integer representing the position of a Gun value in the vector
//  An output of -1 means not found
int fndGn(const vector<Gun> &vec, const Gun &key) {
    for (int i = 0; i < vec.size(); ++i) { //For every value in a vector
        //Compare all values in a Gun object
        if (vec[i].name == key.name && vec[i].dsc == key.dsc
            && vec[i].ammo == key.ammo && vec[i].atk == key.atk) {
            //Return the position
            return i;
        }
    }

    //Return not found
```

```
    return -1;
}

/*****************************************************************************/
/*********************************Find Charm**********************************/
/*****************************************************************************/
//  Linear search a vector for a charm
//Inputs
//  &vec : The vector to search
//  &key : The Charm value to find
//Output
//  An integer representing the position of a Charm value in the vector
//  An output of -1 means not found
int fndChrm(const vector<Charm> &vec, const Charm &key) {
    for (int i = 0; i < vec.size(); ++i) { //For every value in a vector
        //Compare all values in a Charm object
        if (vec[i].name == key.name && vec[i].dsc == key.dsc
            && vec[i].def == key.def) {
            //Return the position
            return i;
        }
    }

    //Return not found
    return -1;
}

/*****************************************************************************/
/*********************************Save Game***********************************/
/*****************************************************************************/
//  Save game data to a file
//Inputs
//  &user : The game's player
//  &guns : The vector containing the list of guns in the game
//  &charms : The vector containing the list of charms in the game
void svGame(const Player &user, const vector<Gun> &guns,
            const vector<Charm> &charms) {
    //Objects
    string path = user.name + ".sav"; //The path to save to
    ofstream oFile; //The output file

    oFile.open(path.c_str()); //Open the file
    oFile << user.name << endl; //Output the user's name
    oFile << static_cast<int> (user.level) << endl; //Output the user's level
    oFile << static_cast<int> (user.hp) << endl; //Output the user's hp
    oFile << fndGn(guns, user.eqGun) << endl; //Output the user's gun
    oFile << fndChrm(charms, user.eqCharm) << endl; //Output the user's charm
    oFile << user.gold << endl; //Output the user's current gold
    oFile.close(); //Close the file
}

/*****************************************************************************/
/*********************************Load Game***********************************/
/*****************************************************************************/
//  Load game data from a file
//Inputs
//  &user : The game's player
//  &guns : The vector containing the list of guns in the game
//  &charms : The vector containing the list of charms in the game
```

```
void ldGame(Player &user, const vector<Gun> &guns,
            const vector<Charm> &charms) {
    //Variables
    int gun, //The gun to equip to the player
        charm; //The charm to equip to the player
    //Objects
    string path = user.name + ".sav", //The file path
           level, //The player's level as a string
           hp; //The player's hp as a string
    ifstream iFile; //The input file

    iFile.open(path.c_str()); //Open the file
    iFile >> user.name; //Read in the player's name
    iFile >> level; //Read in the level
    //Convert the Level to an integer type
    istringstream cnvLvl(level);
    cnvLvl >> user.level;
    iFile >> hp; //Read in the player's hp
    //Convert the HP to an integer type
    istringstream cnvHp(hp);
    cnvHp >> user.hp;
    iFile >> gun; //Read in gun
    user.eqGun = guns[gun]; //Equip the gun
    iFile >> charm; //Read in charm
    user.eqCharm = charms[charm]; //Equip the charm
    iFile >> user.gold; //Read in the player's gold
    iFile.close(); //Close the file
}

/*************************************************************************/
/**********************************Print Player***************************/
/*************************************************************************/
//  Print the player's attributes
//Inputs
//  &p : The game's player
void pPlayer(const Player &p) {
    //Output the player's name
    cout << "NAME:  " << p.name << endl;
    //Output the player's level
    cout << "LEVEL: " << static_cast<int> (p.level) << endl;
    //Output the player's hp
    cout << "HP:    " << static_cast<int> (p.hp) << "/"
         << static_cast<int> (gMaxHp(p.level)) << endl;
    //Output the player's current gun
    cout << "GUN:   " << p.eqGun.name << endl;
    //Output the player's ammo
    cout << "AMMO:  " << p.eqGun.cAmmo << "/" << p.eqGun.ammo << endl;
    //Output the player's atk
    cout << "ATK:   " << static_cast<int> (p.eqGun.atk) << endl;
    //Output the player's current charm
    cout << "CHARM: " << p.eqCharm.name << endl;
    //Output the player's def
    cout << "DEF:   " << static_cast<int> (p.eqCharm.def) << endl;
    //Output the player's gold
    cout << "GOLD:  " << p.gold << endl;
}

/*************************************************************************/
/**********************************Load Enemies***************************/
```

```
/*****************************************************************************/
//  Load enemy data from a file
//Inputs
//  &guns : The list of guns in the game
//  &charms : The list of charms in the game
//Outputs
//  enemies : a vector containing the loaded enemy data
vector<Player> ldEnms(const vector<Gun> &guns, const vector<Charm> &charms) {
    //Objects
    string level, //The character's level
           gold, //The character's gold
           charm, //The character's charm
           gun; //The character's gun
    Player temp; //A temporary player value
    ifstream iFile; //The file stream
    //Collection
    vector<Player> enemies; //The vector to return

    iFile.open("enemies.dat"); //Open the file
    while (iFile.good()) { //While the file is good
        getline(iFile, temp.name); //Read in the character's name
        temp.name = trim(temp.name); //Trim the name
        getline(iFile, level); //Read in the level
        //Convert the level from a string to an integer type
        istringstream cnvLvl(level);
        cnvLvl >> temp.level;
        getline(iFile, gold); //Read in gold
        getline(iFile, charm); //Read in charm
        getline(iFile, gun); //Read in gun
        temp.hp = gMaxHp(temp.level); //Set hp
        //Convert gold from a string to an integer type
        istringstream cnvGold(gold);
        cnvGold >> temp.gold;
        temp.eqCharm = charms[charm.at(0) - 48]; //Equip the gun
        temp.eqGun = guns[gun.at(0) - 48]; //Equip the charm
        enemies.push_back(temp); //Copy temp to the back of enemies
    }
    iFile.close(); //Close the file

    return enemies;
}

/*****************************************************************************/
/*********************************Load Charms*********************************/
/*****************************************************************************/
//  Load charm data from a file
//Outputs
//  charms : a vector containing the loaded charms data
vector<Charm> ldChrms() {
    //Objects
    string def; //The def value of a charm
    Charm temp; //A temporary charm value
    ifstream iFile; //The file stream
    //Collections
    vector<Charm> charms; //The collection of charms to return

    iFile.open("charms.dat"); //Open the file
    while (iFile.good()) { //While the file is good
        getline(iFile, temp.name); //Read in the name
```

```
        getline(iFile, temp.dsc); //Read in the description
        getline(iFile, def); //Read in the def value
        //Convert def from a string to an integer type
        istringstream convert(def);
        convert >> temp.def;
        charms.push_back(temp); //Copy temp to the list of charms
    }
    iFile.close(); //Close the file

    return charms;
}

/*************************************************************************/
/**********************************Load Guns**********************************/
/*************************************************************************/
//  Load gun data from a file
//Outputs
//  guns : a vector containing the loaded gun data
vector<Gun> ldGns() {
    //Objects
    string ammo, //The maximum ammo value
           atk; //The atk value
    Gun temp; //A temporary gun value
    ifstream iFile; //The file stream
    //Collections
    vector<Gun> guns; //The list of guns to return

    iFile.open("guns.dat"); //Open the file
    while (iFile.good()) { //While the file is good
        getline(iFile, temp.name); //Read in the name
        getline(iFile, temp.dsc); //Read in the description
        getline(iFile, ammo); //Read in the ammo
        getline(iFile, atk); //Read in the atk
        //Convert ammo from a string to an integer type
        istringstream cnvAmmo(ammo);
        cnvAmmo >> temp.ammo;
        temp.cAmmo = temp.ammo; //Set current ammo to max ammo
        //Convert atk from a string to an integer type
        istringstream cnvAtk(atk);
        cnvAtk >> temp.atk;
        guns.push_back(temp); //Copy temp to the back of the guns collection
    }
    iFile.close(); //Close file

    return guns;
}

/*************************************************************************/
/**********************************Get Max Hp**********************************/
/*************************************************************************/
//  Get the maximum hp value for the current level of a character
//Inputs
//  level : The level to find the max hp for
//Outputs
//  The max hp for the input level as calculated by the formula
//  100 * (1 + 0.5)^level
short gMaxHp(unsigned short level) {
    const unsigned char BASEHP = 100; //The minimum hp value
    const float RATE = 0.05f; //The growth rate per level
```

```
    return BASEHP * (pow((1 + RATE), level));
}

/*********************************************************************************/
/*********************************Get Input***************************************/
/*********************************************************************************/
//  Get a single character of input from the user
//Outputs
//  The character input by the user uppercased if possible
char gInput() {
    //Objects
    string input; //The user's input

    //Basic input processing
    cout << "> ";
    cin >> input;

    //Return the first character of the input uppercased
    return toupper(input.at(0));
}

/*********************************************************************************/
/*****************************Get String Input************************************/
/*********************************************************************************/
//  Get a string as input from the user
//Outputs
//  The input string
string gInStr() {
    //Objects
    string input; //The input value

    //Basic input processing
    cout << "> ";
    cin >> input;

    return input;
}

/*********************************************************************************/
/*********************************Get Choices*************************************/
/*********************************************************************************/
//  Get an array of valid choices given an array of strings
//Inputs
//  output[] : The array to output to
//  input[] : The list of strings to process
//  length : The length of the input and output arrays
//Outputs
//  output[] : The output array filled with the first letter of each string value
void gChoice(char output[], const string input[], int length) {
    for (int i = 0; i < length; ++i) { //For each string value
        output[i] = toupper(input[i].at(0)); //Set the output to the uppercase of
the first character of the input
    }
}

/*********************************************************************************/
/*************************************Pause***************************************/
/*********************************************************************************/
```

```
//  Pause the game and wait for user input to continue
void pause() {
    cin.ignore(); //Clear the input
    cout << "Press Enter to continue..."; //Display instructions
    cin.get(); //Wait for input
}

/*******************************************************************************/
/*******************************Format Option***********************************/
/*******************************************************************************/
//  Format menu options for display
//Inputs
//  &opt : The string to format
//Outputs
//  r : The string formated so that the first character is wrapped in ()s
string frmtOpt(const string &opt) {
    string r = opt; //Set the return to the input string

    r.insert(0, 1, '('); //Insert an open paren
    r.insert(2, 1, ')'); //Insert a close paren

    return r;
}

/*******************************************************************************/
/*********************************Reload****************************************/
/*******************************************************************************/
//  Reload a player's weapon
//Inputs
//  &p : The player to reload
void reload(Player &p) {
    p.eqGun.cAmmo = p.eqGun.ammo; //Set current ammo to max ammo
}

/*******************************************************************************/
/*********************************Show Menu*************************************/
/*******************************************************************************/
//  Display a menu and record input on that menu
//Inputs
//  opts[] : The options to display
//  length : The length of the option array
//Outputs
//  input : the user input choice
char shwMenu(const string opts[], int length) {
    //Variables
    bool invalid = true; //Whether the input is valid or not
    char input; //The input character
    char choices[length] = {0}; //The array of valid choices

    gChoice(choices, opts, length); //Initialize choices
    for (int i = 0; i < length; ++i) { //For each option
        //Output the option formatted and sized correctly
        cout << setw(opts[i].length() + 5) << frmtOpt(opts[i]);
    }
    cout << endl;
    do { //While the input is not valid
        input = gInput();
        for (int i = 0; i < length; ++i) { //For each choice
            if (choices[i] == input) { //If the input is a choice
```

```
                invalid = false; //The input is valid
            }
        }
    } while (invalid);

    return input;
}

/*******************************************************************************/
/*********************************Bounty Board**********************************/
/*******************************************************************************/
//  Display the bounty board menu and process it
//Inputs
//  &enemies : The list of enemies in the game
//  &bounty : The position of the chosen bounty in the enemies list
//Outputs
//  &bounty : The position after being set
void bBoard(const vector<Player> &enemies, int &bounty) {
    //Variables
    char choice; //The player's choice of bounty
    //Collections
    string menu[enemies.size()]; //The menu array

    for (int i = 0; i < enemies.size() - 1; ++i) { //Fill the menu array
        menu[i] = enemies[i].name + '\n'; //Format the menu to display vertically
    }
    choice = shwMenu(menu, enemies.size() - 1); //Display menu and get choice
    for (int i = 0; i < enemies.size() - 1; ++i) { //Check to see if the input is a
choice
        if (choice == toupper(menu[i].at(0))) {
            //Show your choice
            cout << "You choose to hunt " << enemies[i].name << endl;
            //Set bounty
            bounty = i;
        }
    }
}

/*******************************************************************************/
/*********************************Load Words************************************/
/*******************************************************************************/
//  Load a word list
//Inputs
//  *words : A pointer to an array of strings
//  length : The length of the pointed to array
//Outputs
//  *words : The array after being filled with words
void ldWrds(string *words, int length) {
    //Objects
    ifstream iFile; //The file stream

    iFile.open("wdlcur.txt"); //Open the file
    for (int i = 0; iFile >> *(words + i); ++i); //Input the current file line into
the array
    iFile.close(); //Close the file
}

/*******************************************************************************/
/*************************Load Frequency Data***********************************/
```

```
/*******************************************************************************/
//  Load frequency data from a file into a 2d array
//Inputs
// lFreq[][] : The array to load data into
//Outputs
// lFreq[][] : The filled array
void ldFreq(unsigned short lFreq[ALPHAS][LS]){
    ifstream iFile; //The file stream

    iFile.open("freq.dat"); //Open frequency data file
    for (int i = 0; i < ALPHAS; ++i) { //For each cell in lFreq
        lFreq[i][0] = i; //Set the first value to the current value of i
        iFile >> lFreq[i][1]; //Set the second value to the frequency of the
corresponding letter
    }
    iFile.close(); //Close the frequency data file
}

/*******************************************************************************/
/************************************Guess**************************************/
/*******************************************************************************/
//  Play hangman from the computer player's perspective
//Inputs
//  word : The word to try to guess
//  &p : The computer character guessing the word
//Outputs
// true if guessed or false if failed
bool guess(string word, Player &p) {

    //Variables
    unsigned short gCount = p.eqCharm.def, //The number of guesses the character
gets
                   total; //The total number of letters in the frequency list
    //Objects
    string mWord = mask(word); //The masked version of the word to guess

    unsigned short lFreq[ALPHAS][LS]; //2d array of letters and frequencies

    //Read the frequency data
    ldFreq(lFreq);

    //Calculate total of frequency data
    for (int i = 0; i < ALPHAS; ++i) {
        total += lFreq[i][1];
    }

    //Dynamically allocate a character array to pick guesses from
    char *gList = new char[total];

    //Initialize the guess list
    for (int i = 0; i < total; ++i) {
        gList[i] = 0;
    }

    //Fill the guess list
    string temp; //Create a temporary string
    for (int i = 0; i < ALPHAS; ++i) { //For each letter in the alphabet
        for (int j = 0; j < lFreq[i][1]; ++j) { //For the frequency of that
character
```

```
            temp += static_cast<char> (lFreq[i][0] + 97); //Write that character to
the temporary string
        }
    }
    strcpy(gList, temp.c_str()); //Copy the temporary string to the guess list
    temp.clear(); //Clear the temporary string

    //Preform guesses
    cout << "OPPONENTS GUESSES: ";
    for (int i = gCount; i > 0;) {
        char guess = gList[rand() % total]; //Guess a random character from the
guess list
        cout << guess; //Output the guess
        if (cntns(word, guess)) { //If the word contains the guess
            unmask(word, mWord, guess); //Unmask the character
        } else { //Else subtract for guess
            --i;
        }
        if (mWord == word) { //If the masked word and the guess word match
            cout << endl;
            delete [] gList; //Delete the guess list
            return true;
        }
    }
    cout << endl;

    delete [] gList; //Delete the guess list
    return false;
}

/*****************************************************************************/
/***********************************Trim**************************************/
/*****************************************************************************/
// Trim a string so that it does not contain nonprinting characters
//Inputs
//  str : The string to trim
//Outputs
// r : The trimmed string
string trim(string str) {
    string r;

    for (int i = 0; i < str.size(); ++i) { //For each character in str
        if (str.at(i) > 31 && str.at(i) < 127) { //Copy the character if it is
printable
            r += str.at(i);
        }
    }

    return r;
}

/*****************************************************************************/
/*********************************Is Word*************************************/
/*****************************************************************************/
//  Checks if a string only contains alphabetic characters
//Inputs
//  word : the string to check
//Outputs
//  true if the string only contains alphabetic characters
```

```
//  false if the string contains non alphabetic characters
bool isWord(string word) {
    for (int i = 0; i < word.size(); ++i) { //Loop through every character in word
        if (word.at(i) < 96 || word.at(i) > 123) {
            return false;
        } //If word only contains alphabetic characters
    }
    return true;
}

/*****************************************************************************/
/***********************************Mask**************************************/
/*****************************************************************************/
//  Returns a string of equal length to the input string filled with *'s
//Inputs
//  oWord : The original word
//Outputs
//  r : The masked word
string mask(string oWord) {
    string r; //The masked word

    for (int i = 0; i < oWord.size(); ++i) { //loop through the characters in oWord
        r += '*'; //Copy a * to the masked word
    }

    return r;
}

/*****************************************************************************/
/**********************************Contains***********************************/
/*****************************************************************************/
//  A function to determine if a string contains a character
//Inputs
//  word : The word to search
//  key : The character to search for
//Outputs
// true if the character is found otherwise false
bool cntns(string word, char key) {
    for (int i = 0; i < word.length(); ++i) { //loop through word
        if (word.at(i) == key) {
            return true;
        } //if a character matches the key return true
    }
    //Otherwise return false
    return false;
}

/*****************************************************************************/
/*********************************Gnome Sort**********************************/
/*****************************************************************************/
//  Sort a character array. Weights null characters as higher than all others
//Inputs
//  cArr : The character array to sort
//  length : The length of the array to sort
//Outputs
//  cArr : The input array after sorting
void gSort(char cArr[], int length) {
    for (int pos = 1; pos < length;) { //Gnome Sort modified to handle null
terminators
```

```
        if (cArr[pos] >= cArr[pos - 1] || cArr[pos] == '\0') { //if the current
character is greater than the previous one or null
            ++pos; //move one position forward
        } else if (cArr[pos] <= cArr[pos - 1]) { //If the current character is less
than or equal to the previous
            //Swap the current character and the last
            cArr[pos] = cArr[pos] ^ cArr[pos - 1];
            cArr[pos - 1] = cArr[pos] ^ cArr[pos - 1];
            cArr[pos] = cArr[pos] ^ cArr[pos - 1];
            if (pos > 1) { //If the position is greater than one
                --pos; //Move back one position
            }
        }
    }
}

/*****************************************************************************/
/*********************************Unmask**************************************/
/*****************************************************************************/
//  Unmask a single character in a masked string based on an unmasked string
//Inputs
//  &oWord : The original word
//  &mWord : The masked word
//  key : The character to search for
//Outputs
//  &mWord : With all instances of key inserted in the correct positions
void unmask(string &oWord, string &mWord, char key) {
    for (int i = 0; i < oWord.size(); ++i) { //loop through all characters
        if (oWord.at(i) == key) { //compare each character to the guess character
            mWord.at(i) = key; //Unmask the character if they match
        }
    }
}

/*****************************************************************************/
/*********************************To Lower Case*******************************/
/*****************************************************************************/
//  Convert a string to all lowercase letters
//Inputs
//  &word : The input string to convert
//Outputs
//  &word : The input string after conversion
void toLCase(string &word) {
    for (int i = 0; i < word.size(); ++i) { //For each character in word
        word.at(i) == tolower(word.at(i)); //Set the character to lowercase
    }
}

/*****************************************************************************/
/*********************************Battle**************************************/
/*****************************************************************************/
//  Battle processing
//Inputs
//  &user : The object representing the player of the game
//  &opnt : The player's opponent
//Outputs
//  true if the player wins the battle and false otherwise
bool battle(Player &user, Player &opnt) {
    //Constants
```

```
    const int WS = 125; //The size of the word list

    //Variables
    bool btlOver = false, //Whether or not the battle is finished
         r; //The return value
    unsigned short gCount = user.eqCharm.def, //The number of guesses the player
gets
                   usedPos = 0; //The current position in the used character array
    //Objects
    string uWord, //The user input word
           oWord, //The original word to guess
           mWord; //The masked word to guess
    //Collections
    char used[ALPHAS] = {0}; //The array of used character
    string words[WS]; //The word list

    ldWrds(words, WS); //Load the words for the word list

    do { //While the battle isn't over
        //Player Turn
        cout << user.name << "'s turn!" << endl;
        if (user.eqGun.cAmmo > 0) { //If the player has ammo
            //Output battle status
            cout << "PLAYER HP:   " << user.hp << "/" << gMaxHp(user.level)
                    << endl;
            cout << "OPPONENT HP: " << opnt.hp << "/" << gMaxHp(opnt.level)
                    << endl;
            cout << "AMMO: " << user.eqGun.cAmmo << "/" << user.eqGun.ammo
                    << endl;
            //Input a word
            cout << "Choose a word to fire" << endl;
            do { //While the input is not a word
                uWord = gInStr(); //Get an input string
                toLCase(uWord); //Covert to lower case
            } while (!isWord(uWord));
            if (!guess(uWord, opnt)) { //If the opponent didn't guess your word
                opnt.hp -= user.eqGun.atk; //Do damage to your opponent
                cout << opnt.name << " was hit!" << endl;
                cout << opnt.name << " took " << user.eqGun.atk << " damage"
                        << endl;
            } else { //Otherwise you miss
                cout << opnt.name << " dodged your shot!" << endl;
            }
            user.eqGun.cAmmo--; //Reduce current ammo by one
        } else { //Otherwise reload your gun
            reload(user);
            cout << user.name << " reloaded!" << endl;
        }
        //Opponent Turn
        if (opnt.hp > 0) { //If the opponent is not dead
            cout << opnt.name << "'s turn!" << endl;
            if (opnt.eqGun.cAmmo > 0) { //If the opponent has ammo
                oWord = words[rand() % WS]; //Pick a random word from the list
                mWord = mask(oWord); //Mask the word
                do { //While you haven't guessed the word and you haven't run out
of guesses
                    char guess; //The character that the user guesses
                    //Display hangman information
                    cout << mWord << endl;
```

```
                    cout << "REMAINING GUESSES: " << gCount << endl;
                    cout << "USED CHARACTERS:   " << used << endl;
                    guess = tolower(gInput()); //Get guess
                    if (cntns(oWord, guess)) { //If the original word contains your
guess
                        unmask(oWord, mWord, guess); //Unmask that character
                    } else { //Otherwise
                        --gCount; //Lose one guess
                    }
                    if (!cntns(string(used), guess)) { //If used doesn't contain
the guess already
                        used[usedPos++] = guess; //Add the guess and increment the
position
                    }
                    gSort(used, ALPHAS); //Sort the used character array
                } while (gCount > 0 && mWord != oWord);
                //Output the original word
                cout << oWord << endl;
                if (gCount <= 0) { //If you ran out of guesses
                    user.hp -= opnt.eqGun.atk; //Take damage
                    cout << user.name << " was hit!" << endl;
                    cout << user.name << " took " << opnt.eqGun.atk << " damage"
                            << endl;
                } else if (mWord == oWord) { //Otherwise if you guessed correctly
                    cout << user.name << " dodged the shot!" << endl;
                }
                gCount = user.eqCharm.def; //Reset your guess count
                for (int i = 0; i < 26; ++i) { //Reinitialize the used character
array
                    used[i] = 0;
                }
                usedPos = 0; //Reset the used position

                opnt.eqGun.cAmmo--; //Reduce current ammo by one
            } else { //Otherwise reload
                reload(opnt);
                cout << opnt.name << " reloaded!" << endl;
            }
        }
        if (user.hp <= 0) { //If the user is dead
            cout << "YOU DIED" << endl;
            r = false; //Return a loss
            btlOver = true; //Finish the battle
        } else if (opnt.hp <= 0) { //If the opponent is dead
            cout << opnt.name << " was defeated" << endl;
            cout << "You were awarded " << opnt.gold << " gold" << endl;
            r = true; //Return a victory
            if(user.gold < 60000){ //Collect your spoils
                user.gold += opnt.gold;
            }
            btlOver = true; //End the battle
        }
    } while (!btlOver);
    opnt.hp = gMaxHp(opnt.level); //Reset opponent's hp for later
    //Reload all guns
    reload(user);
    reload(opnt);

    return r;
```

```
}

/*******************************************************************************/
/**********************************Play Game************************************/
/*******************************************************************************/
//  Main Game processing
void plyGame() {
    //Constants
    const int PGMS = 2, //Play game menu size
              GMS = 6; //Game menu size
    //Variables
    bool qGame = false; //Whether or not to quit the game
    int bounty = -1; //The current bounty
    unsigned short wins = 0; //The current number of wins
    //Objects
    Player pUser; //The player of the game
    //Collections
    vector<Gun> guns(ldGns()); //The guns in the game
    vector<Charm> charms(ldChrms()); //The charms in the game
    vector<Player> enemies(ldEnms(guns, charms)); //The enemies in the game
    string pgMenu[] = {"New Game", "Load"}; //The play game menu
    string gMenu[] = {"Bounty Board", "Hunt Bounty", //The game menu
        "Gunsmith", "Shaman", "Character", "Quit"};

    //Main game processing
    switch (shwMenu(pgMenu, PGMS)) { //Choose based on menu input
        case 'N': //New Game
        {
            dspFile("titlecrawl.txt"); //Display title info
            pause();
            //Create new character
            cout << "Enter your name" << endl;
            pUser.name = gInStr();
            pUser.eqGun = guns[0];
            pUser.eqCharm = charms[0];
            pUser.gold = 1000;
            pUser.level = 1;
            pUser.hp = gMaxHp(pUser.level);
            break;
        }
        case 'L': //Load Game
        {
            //Input character to load
            cout << "Enter the name of a character to load" << endl;
            pUser.name = gInStr();
            if (chkFile(pUser.name + ".sav")) { //If the file exists
                ldGame(pUser, guns, charms); //Load the game
            } else { //Otherwise create a new game
                //Output error message
                cout << "Save not found" << endl;
                cout << "Creating new game..." << endl;
                //Create new game
                dspFile("titlecrawl.txt");
                pause();
                pUser.eqGun = guns[0];
                pUser.eqCharm = charms[0];
                pUser.gold = 1000;
                pUser.level = 1;
                pUser.hp = gMaxHp(pUser.level);
```

```
            }
            break;
        }
    }
    //Town Menu
    do {
        dspFile("town.txt"); //Display flavor text
        switch (shwMenu(gMenu, GMS)) { //Choose based on menu input
            case 'B': //Bounty Board
            {
                dspFile("bountyboard.txt"); //Display flavor text
                bBoard(enemies, bounty); //Show bounty board
                pause();
                break;
            }
            case 'H': //Hunt Bounty
            {
                if (bounty != -1) { //If bounty is set
                    reload(pUser); //Reload gun
                    battle(pUser, enemies[bounty]) ? wins += 1 : wins += 0; //Hunt
bounty
                } else { //Otherwise print error message
                    cout << "You have to choose a bounty first" << endl;
                    cout << "Go to the bounty board" << endl;
                }
                if (pUser.hp <= 0) { //If player is dead
                    //Print game over and quit to main menu
                    cout << "GAME OVER" << endl;
                    qGame = true;
                } else { //Otherwise if they're alive
                    if (wins % 5 == 0 && wins != 0) { //If wins is divisible by 5
and not 0
                        //Level up processing
                        pUser.level++; //Increase player level
                        cout << "LEVEL UP!" << endl;
                        cout << "HP increased by "
                                << gMaxHp(pUser.level) - gMaxHp(pUser.level - 1)
                                << endl;
                        //Set player HP
                        pUser.hp = gMaxHp(pUser.level);
                    }
                    pause();
                }
                break;
            }
            case 'G': //Gunsmith
            {
                shGns(guns, pUser); //Do gun shop processing
                pause();
                break;
            }
            case 'S': //Shaman
            {
                const int SHMS = 2; //Shaman menu size
                string shMenu[] = {"Heal", "Shop"}; //Shaman Menu
                dspFile("shaman.txt"); //Display Flavor text
                switch (shwMenu(shMenu, SHMS)) { //Choose based on menu input
                    case 'H': //Heal
                    {
```

```
                        shHeal(pUser); //Do heal processing
                        break;
                    }
                    case 'S': //Shop
                    {
                        shChrms(charms, pUser); //Do charm shop processing
                        break;
                    }
                }
                pause();
                break;
            }
            case 'C': //Character
            {
                pPlayer(pUser); //Display character info
                pause();
                break;
            }
            case 'Q': //Quit
            {
                qGame = true; //Quit the game
                break;
            }
        }
    } while (!qGame);
    if (pUser.hp > 0) { //If you didn't quit because of a game over
        svGame(pUser, guns, charms); //Save the game
    }
}

/*******************************************************************************/
/*******************************Display File************************************/
/*******************************************************************************/
//  Write the text from a file to standard out one line at a time
//Inputs
//  path : the path to the file to display
void dspFile(string path) {
    string next; //input buffer
    ifstream iFile; //input file stream

    iFile.open(path.c_str()); //open file
    while (getline(iFile, next)) { //read each line
        cout << next << endl; //output line
    }
    iFile.close(); //close file
}

/*
 * File:   Game.h
 * Author: Alexander Rothman
 * Purpose: Define game objects for Desperado
 * Created on February 9, 2016, 10:01 AM
 */

#ifndef GAME_H
#define     GAME_H

/*******************************************************************************/
/************************************Gun****************************************/
```

```
/*****************************************************************************/
//  Struct representing a gun in the Desperado game
struct Gun{
    unsigned short atk, //The attack power of the gun
                   ammo, //The maximum ammo the gun can hold
                   cAmmo; //The current ammo in the gun
    string name, //The gun's name
           dsc; //The gun's description
};

/*****************************************************************************/
/************************************Charm************************************/
/*****************************************************************************/
//  Struct representing a charm in the Desperado game
struct Charm{
    unsigned short def; //The defense power of the charm
    string name, //The name of the charm
           dsc; //The description of the charm
};

/*****************************************************************************/
/************************************Player***********************************/
/*****************************************************************************/
//  Struct representing a player in the Desperado game
struct Player{
    short hp; //The player's current hit points
    unsigned short  level, //The player's current level
                    gold; //The player's current gold
    string name; //The player's name
    Charm eqCharm; //The player's currently equipped charm
    Gun eqGun; //The player's currently equipped gun
};

#endif      /* GAME_H */
```