. .\*+ **FANTASY WAVE SURVIVAL** +\*. .

　　Ａｕｔｈｏｒ：　ケビン·ミンドルー

**<@@>** \*

**||** +

\* **||**

**o===<@@>===o** \*

| |

+ | | \*

| | \*

\* | |

| | +

\* | |

o.,.,/#/^^O;W\\Xo^',,,

,,o.x#//##//o^\^#cVw\#W;,,o..,

////////////////////////////////////////////

**Project 2**

Title

**FANTASY WAVE SURVIVAL**

Class

**CSC-05**

Section

**42450**

Due Date

**June 12,2014**

Author

**Kevin R. Mindreau**

**Introduction**

Title: Fantasy Wave Survival

This game is a simple fantasy survival game.

The player gets the choice between 5 distinct classes: *Knight*, *Wizard*, *Gladiator*, *Cleric*, and *Onion Knight*. Each class has a specific stat affinity as follows:

* Knight has strong, physical attack power.
* Wizard has strong, magical attack power.
* Gladiator has strong, physical defensive power.
* Cleric has strong, magical resistance and healing power.

Except the Onion Knight, whose stats are and can be randomly distributed among all areas to the extremes. The Onion Knight is a wild card and can either hinder or greatly benefit the player’s progress.

Each wave will provide a single enemy the player must face. Every 10th wave, however, a boss enemy will appear and is much more powerful than the normal enemies encountered before it.

The player must survive 50 waves in order to beat the game.

**Summary**

Project size: 751 lines

Number of variables: 27

Number of functions: 11

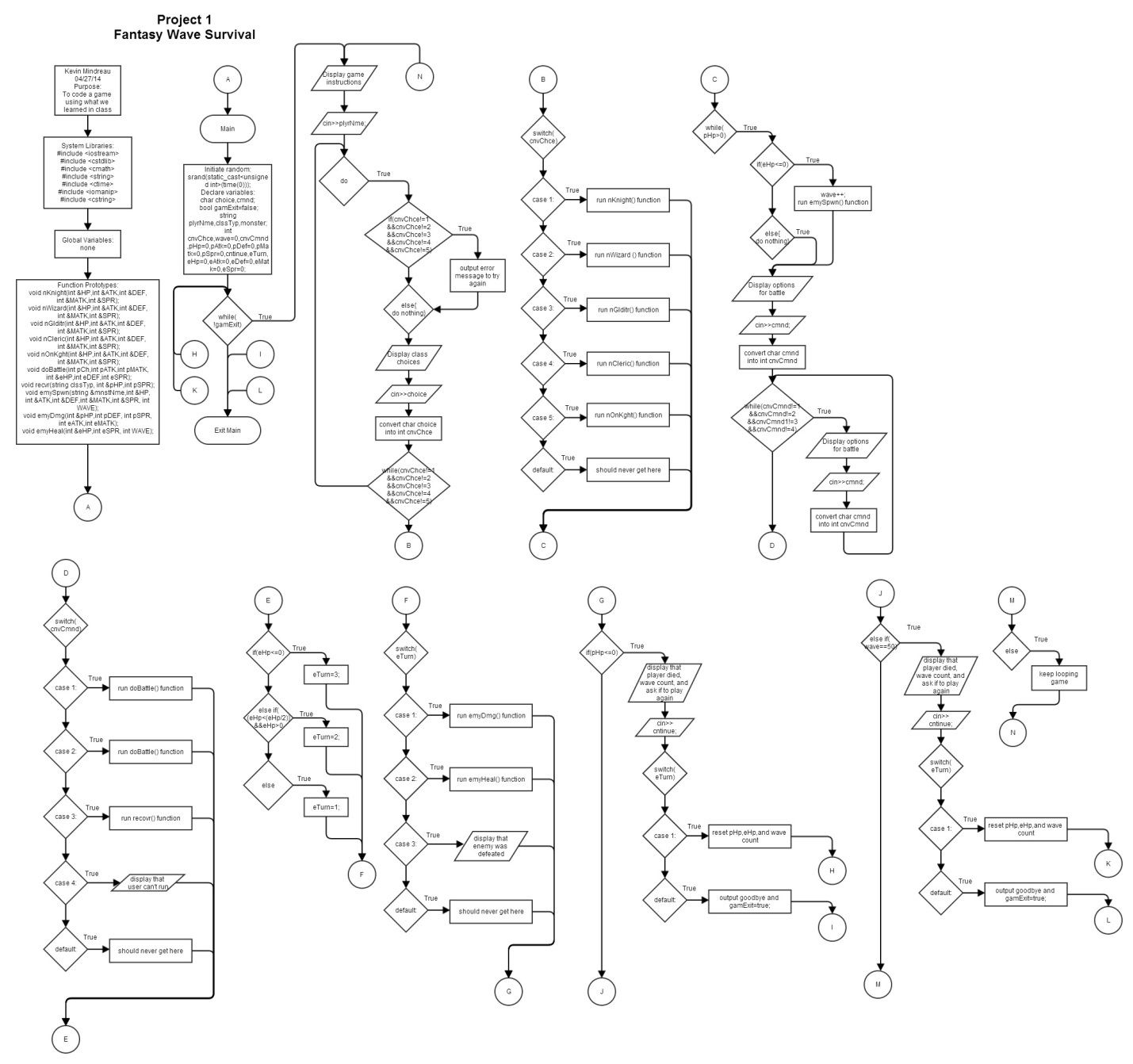
This project covers the basics we’ve learned from the text and in class, as well as implementing functions, using loops, switch menu’s, and if statements. I’ve included the use of arrays to hold the player data and reading from a file to my program. As well as outputting to a file. I created text files that store the data for player high-scores. I have also added a defend function that allows the player to buffer an enemy’s attack, but this consumes a turn. So players have to be wary! I also added critical damage to the program and both the player and the opponent have this. There is a 20% chance that this will occur and it doubles the damage the action performed normally would.

This project took me about 2-3 weeks to program fully. I spent a solid five to six hours on this and time flew by. It honestly felt like an hour and a half passed instead. What really got me pulling my hair was that somewhere in my mathematic calculations, I was causing the program to behave oddly and it resulted in my program crashing. I eventually decided to rewrite the code for how damage is handled and everything seems to be working just fine! Outputting to a file was a little tricky and I spent about 4 hours trying to get it to read it properly. But once I did, everything fell into place and I was ecstatic. I really enjoyed programming this up and much of the ideas for this came from video games I have played on my Gameboy, such as, Final Fantasy by Square Enix. I wish to further work on this and implement more complex actions in the future.

**Description**

The goal of the game is to reach wave 50 and defeat the final boss. Each turn is consisted of the player’s action and the enemy’s action. The player can choose to Attack, use Magic, Heal, or flee. When the player succeeds or is defeated, the game will end and prompt the player with their wave count record. Then it will ask if the player wishes to start over. One hidden truth about this game is that boss monsters are created unfairly ON PURPOSE. But It isn’t impossible if a player manages to generate a powerful Onion Knight that is outstanding in all stats.

**Flowchart**



**Pseudo Code**

*Initialize rand and declare variables*

*Declare variables for file input/output*

*While Loop until gamExit = true*

*Open and Read intro file and prompt intro to explain game*

*Close intro file*

*Ask user for name and class choice*

*Do-While loop until player makes proper choice*

*Display list of five classes to play*

*Based on user choice of class, apply switch menu*

*If user input isn’t 1,2,3,4,or 5 keep looping until it is*

*Else do not loop*

*Switch user input for class choice*

*Case 1 = Warrior – call function to create character*

*Case 2 = Wizard – call function to create character*

*Case 3 = Gladiator – call function to create character*

*Case 4 = Cleric – call function to create character*

*Case 5 = Onion Knight – call function to create character*

*Default – should never get here*

*While Loop while player is alive i.e. hp!=0*

*If enemy HP<=0, increment wave count and spawn enemy*

*Else do not spawn enemy*

*Player turn – choose from: Attack, Magic, Heal, or Flee*

*While loop for user input if invalid and prompt for a valid input*

*Switch user input for action*

*Case 1 = attack – call function to do battle*

*Case 2 = Magic – call function to do battle*

*Case 3 = Heal – call function to recover*

*Case 4 = Defend – call function to set defend to true*

*Case 4 = Flee – flee is impossible = do nothing/waste a turn*

*Default – should never get here*

*Determine action of enemy based on hp levels and use switch menu*

*If Enemy hp <=0 – display message of enemy death then end enemy turn*

*Else if Enemy hp > 0 && hp < ½ max hp – enemy turn will be heal*

*Else – enemy turn will be attack*

*Switch enemy turn based on statements above*

*Case 1 – call enemy attack function (physical or magic attack, randomly chosen)*

*Case 2 – call enemy recover function*

*Case 3 – display message if enemy dies and end enemy turn*

*Default – should never get here*

*If player HP <= 0*

*End game and display player’s best wave count*

*Open highScoresList.txt – will read then write to list player name and wave count*

*Close highScoresList.txt*

*Then ask to play again*

*Switch menu for play again – 1 = play again, any other key = quit game*

*Case 1 – reset player hp, enemy, and wave count; game starts over*

*Default – exitGam= true, exit game and prompt Thanks for playing*

*Else if wave count equals to 50*

*Display victory message*

*Open highScoresList.txt – will read then write to list player name and wave count*

*Close highScoresList.txt*

*Then ask to play again*

*Switch menu for play again – 1 = play again, any other key = quit game*

*Case 1 – reset player hp, enemy, and wave count; game starts over*

*Default – exitGam= true, exit game and prompt Thanks for playing*

*Else do nothing and continue loop*

**Major Variables**

|  |  |  |
| --- | --- | --- |
| **Type** | **Name** | **Description** |
| integer | cnvChce | Holds value after converting char variable choice into an integer; used for when player chooses a class to play as. |
|  | wave | Keeps wave count. |
|  | cntinue | Holds value for user’s choice to continue game from start or exit game. |
|  | cnvCmnd | Holds value after converting char variable cmnd into an integer; used for when user selections action for battle. |
|  | NUMSTATS | Determines how many stats there are |
|  | eTurn | Int value for enemy’s turn. |
|  | pStats[] | Int array to hold player stats; uses NUMSTATS for size |
|  | eStats[] | Int array to hold enemy stats; uses NUMSTATS for size |
|  | pStats[0] | Player health points. |
|  | pStats[1] | Player attack points. |
|  | pStats[2] | Player physical defense points. |
|  | pStats[3] | Player magic attack points. |
|  | pStats[4] | Player spirit/magic resistance points; applies when healing. |
|  | eStats[0] | Enemy health points. |
|  | eStats[1] | Enemy attack points. |
|  | eStats[2] | Enemy physical defense points. |
|  | eStats[3] | Enemy magic attack points. |
|  | eStats[4] | Enemy spirit/magic resistance points; applies when healing. |
|  | pCh | Holds value for attack and changes if it was a physical or magical attack. |
|  | HSLIST | Determines length of High-score List; set to 100 |
|  | Scores[] | Creates array to hold score values; uses HSLIST for size |
|  | place | Holds value of the position in the array |
|  | pos | Holds value of the position of wave in the array |
|  | totScor | Holds value of total waves completed |
| string | plyrNme | String to hold player’s name. |
|  | clssTyp | String that holds class name/type. |
|  | monster | String that holds monster name. |
|  | names[] | Array to hold names for High-score list; uses HSLIST for size |
|  | nameHold | Used for when using bubble sort on Highscore list. |
| char | choice | Input from user when asked to choose a class. |
|  | cmnd | Input from user when asked to make a battle decision. |
| bool | gamExit | As long as this remains false, game will loop until wave reaches 50 or player hp reaches zero. |
| fstream | intro | Used to read in the intro |
|  | hSListI | Used to open and read highScoresList.txt |
|  | hSListO | Used to output and write to highScoresList.txt |

**Major Functions**

|  |  |  |
| --- | --- | --- |
| Type | Name | Description |
| void | doBattle  (int pCh,int p[],int m[]); | Calculates battle damage from player to enemy;  pCh dictates if physical or magical damage |
|  | void defend(bool &); | Sets variable for defend to true |
|  | emyDmg(int p[],int m[],bool &pGrd); | Calculates battle damage from enemy to player;  Will randomly choose between physical or magical damage |
|  | emyHeal(string mnstNme,int m[], int WAVE); | Calculates enemy healing |
|  | emySpwn(string &mnstNme,int m[],int WAVE); | Randomly spawns 1 of 5 normal enemies; spawns 1 of 4 boss enemies when wave counter is divisible of 10 (10, 20, 30, etc…) |
|  | nKnight(int a[]); | Generates stats of character; higher attack point with knight |
|  | nGlditr(int a[]); | Generates stats of character; higher physical defense points with gladiator |
|  | nWizard(int a[]); | Generates stats of character; higher magic attack points with wizard |
|  | nCleric(int a[]); | Generates stats of character; higher magic resistance and healing points with cleric |
|  | nOnKght(int a[]); | Generates stats of character; all stats have a large gap and are randomly generated (wild card class). |
|  | recvr(string clssTyp,int a[]); | Calculates how much player heals |

**Concepts Used**

Savitch 8th Ed. (Textbook)

Gaddis 6th Ed. (slides from blackboard) –

* Chapter 2 – special characters, data types, scope, etc.
* Chapter 3 – expressions and interactivity operators
* Chapter 4 – making decisions, menus, if – else if statements, etc.
* Chapter 5 – Looping
* Chapter 6 – Functions
* Chapter 7 – Arrays & I/O

**Program**

//System Libraries

#include <iostream>//input-output

#include <cstdlib>//c standard library

#include <string>//for string variables

#include <ctime>//random num generator

#include <iomanip>//used for battle dispay

#include <cstring>

#include <fstream>//for reading/writing files

using namespace std;

//No Global constants

//Function Prototypes

//generate stats for new knight

void nKnight(int []);

//generate stats for new wizard

void nWizard(int []);

//generate stats for new gladitor

void nGlditr(int []);

//generate stats for new cleric

void nCleric(int []);

//generates stats for new onion knight-random extremes are common

void nOnKght(int []);

//calculates battle damage

void doBattle(int,int [],int[]);

//buffers enemy's attack until next move

void defend(bool &);

//calculates unit healing based on unit chosen

//passes HP by ref and uses SPR to calculate

void recvr(string,int []);

//spawn enemy - random stats

void emySpwn(string &,int [],int);

//damage enemy deals

//adding bool for defend implementation - reduce damage from player

void emyDmg(int [],int [],bool &);

//amount enemy recovers

void emyHeal(string,int [],int);

//Execution Begins Here!

int main(int argc, char\*\* argv) {

//open file - top scores

ifstream intro;

ifstream hSListI;

ofstream hSListO;

//variables for fstream at end of game

const int HSLIST=100;

int place=0;

int pos=0;

int totScor=0;

string names[HSLIST];

int scores[HSLIST];

int hold;

string nameHold;

//initialize rand

srand(static\_cast<unsigned int>(time(0)));

//declare variables

const int NUMSTATS=5;

char choice,cmnd;

bool gamExit=false;

char plyrNme[20];

string clssTyp,monster;//general info

//add player guard

bool pGrd=false;

//replace variables for stats into arrays

/\* 0=HP 3=MATK

\* 1=ATK 4=SPR

\* 2=DEF

\*/

int pStats[NUMSTATS]={0},eStats[NUMSTATS]={0};

int cnvChce,wave=0,cnvCmnd,cntinue,eTurn;

//Begin Game Code

//loop until game exit

while(!gamExit){

////////////Start adventure!/////////////

cout<<" + FANTASY WAVE SURVIVAL "<<endl

<<" <@@> \* "<<endl

<<" || + "<<endl

<<" \* || "<<endl

<<" o===<@@>===o \* "<<endl

<<" | | "<<endl

<<" + | | \* "<<endl

<<" | | \* "<<endl

<<" \* | | "<<endl

<<" | | + "<<endl

<<" \* | | "<<endl

<<" o. .,#/^^O;WX//^',,, "<<endl

<<" o.x#//##//o^^^#cVw#W;,,,o."<<endl

<<"///////////////////////////////"<<endl;

intro.open("intro.txt",ios::in);

if (!intro){

cerr<<endl<<"Intro File is missing!"<<endl;

exit(1);

}

string word;

while(intro>>word){

cout<<word<<" ";

if(word=="game!"||word=="\n"||word=="works:"||

word=="Goal:"||word=="action."||word=="turn."||

word=="enemies!"||word=="appear!"||word=="battle."||

word=="luck!!!"){

if(word=="game!"||word=="turn."||word=="battle."){

cout<<endl;

}

cout<<endl;

}

}

cout<<endl;

intro.close();

//ask user for name and class choice

cout<<"What is your name? ";

cin>>plyrNme;

cout<<endl;

do{

//based on user choice of class, apply switch menu

cout<<"So "<<plyrNme<<", please pick a class type!"<<endl

<<"//////////////////////////////"<<endl

<<"1. Knight - A strong Physical Attacker"<<endl

<<"2. Wizard - A strong Magical Attacker" <<endl

<<"3. Gladiator - A Heavy Defender, can take a hit"<<endl

<<"4. Cleric - A powerful Healer, resists Magic"<<endl

<<"5. Onion Knight - A special warrior, stats unknown..."<<endl

<<"//////////////////////////////"<<endl;

cin>>choice;

cnvChce=choice-'0';

//offer list of five classes to play

if(cnvChce!=1&&cnvChce!=2&&cnvChce!=3&&cnvChce!=4&&cnvChce!=5){

cout<<"Pick a proper action. Try again."<<endl

<<"///////////////////////////////"<<endl;

}

}while(cnvChce!=1&&cnvChce!=2&&cnvChce!=3&&cnvChce!=4&&cnvChce!=5);

/////////menu for player class creation////////

switch(cnvChce){

case 1:{

//Knight

clssTyp="Knight";

cout<<"You chose "<<clssTyp<<"!"<<endl;

cout<<"==============================="<<endl;

//generate stats

//take in array

nKnight(pStats);

break;

}

case 2:{

//Wizard

clssTyp="Wizard";

cout<<"You chose "<<clssTyp<<"!"<<endl;

cout<<"==============================="<<endl;

//generate stats

nWizard(pStats);

break;

}

case 3:{

//Gladiator

clssTyp="Gladiator";

cout<<"You chose "<<clssTyp<<"!"<<endl;

cout<<"==============================="<<endl;

//generate stats

nGlditr(pStats);

break;

}

case 4:{

//Cleric

clssTyp="Cleric";

cout<<"You chose "<<clssTyp<<"!"<<endl;

cout<<"==============================="<<endl;

//generate stats

nCleric(pStats);

break;

}

case 5:{

//Onion Knight

clssTyp="Onion Knight";

cout<<"You chose "<<clssTyp<<"!"<<endl;

cout<<"==============================="<<endl;

//generate stats

nOnKght(pStats);

break;

}

default:{

cout<<"Should never get here!"<<endl;

}

}

//loop while player is alive e.i. hp!=0

while(pStats[0]>0){

//generate enemy

if(eStats[0]<=0){

wave++;

emySpwn(monster,eStats,wave);

}

//player turn

cout<<"Wave: "<<wave<<endl

<<"Beware! A wild "<<monster<<" is attacking!"<<endl

//create user interface

<<left

<<plyrNme<<"'s HP = "<<pStats[0]

<<right<<setw(15)

<<monster<<"'s HP = "<<eStats[0]<<endl

<<"==============================="<<endl

<<"What would you like to do?"<<endl

<<"1. Attack"<<endl

<<"2. Magic"<<endl

<<"3. Heal"<<endl

<<"4. Defend"<<endl

<<"5. Flee"<<endl

<<"==============================="<<endl;

cin>>cmnd;

cnvCmnd=cmnd-'0';

while(cnvCmnd!=1&&cnvCmnd!=2&&

cnvCmnd!=3&&cnvCmnd!=4&&cnvCmnd!=5){

cout<<"Pick a proper action. Try again: ";

cin>>cmnd;

cnvCmnd=cmnd-'0';

cout<<"==============================="<<endl;

}

//induce menu based on command

switch(cnvCmnd){

case 1:{//physical attack

int pCh=1;//function parameter for attack

doBattle(pCh,pStats,eStats);

cout<<"==============================="<<endl;

break;

}

case 2:{//magical attack

int pCh=2;//function parameter for magic

doBattle(pCh,pStats,eStats);

cout<<"==============================="<<endl;

break;

}

case 3:{

//recover player hp

recvr(clssTyp,pStats);

cout<<"==============================="<<endl;

break;

}

case 4:{

//defend implementation

defend(pGrd);

cout<<"==============================="<<endl;

break;

}

case 5:{

//can't run from battle

cout<<"You can't run from battle!"<<endl

<<"==============================="<<endl;

break;

}

default:{

cout<<"Should never get here!"<<endl;

}

}

//enemy turn

eTurn=1;

//determine action of enemy based on hp levels

//otherwise attack

//if zero or less hp = dead

if(eStats[0]<=0){eTurn=3;}

//if hp less than 1/2 but greater than 0 = heal

else if((eStats[0]<=(eStats[0]/2))&&eStats[0]>0){eTurn=2;}

//otherwise attack

else{eTurn=1;}

//menu for enemy actions

switch(eTurn){

case 1:{

//physical or magic attack, randomly chosen

emyDmg(pStats,eStats,pGrd);

cout<<"==============================="<<endl;

break;

}

case 2:{

//healing action

emyHeal(monster,eStats,wave);

cout<<"==============================="<<endl;

break;

}

case 3:{

//if monster died from last attack

cout<<monster<<" has been slain by "<<plyrNme<<"!"<<endl

<<"Watch out! Another monster approaches!!"<<endl;

cout<<"==============================="<<endl;

break;

}

default:{

cout<<"Should never get here!"<<endl;

}

}

}

//check if player is alive

if(pStats[0]<=0){

//if player is dead, game ends and displays record

cout<<plyrNme<<"'s HP reached zero!"<<endl<<endl

<<"GAME OVER"<<endl<<endl;

cout<<"///////////////////////////////"<<endl

<<"Best Record: "<<wave<<" waves!"<<endl<<endl;

//show score then save score to file

hSListI.open("highScoresList.txt",ios::in);

if (!hSListI) {

cerr<<endl<<"High Score Input File is missing!"<<endl;

exit(1);

}

//get all names

while(hSListI>>names[place]>>scores[pos]){

place++;

pos++;

}

//add curr player to list

names[place+1]=plyrNme;

totScor=pos+1;

//add player wave

scores[pos+1]=wave;

//sort scores in order

for(int i=0;i<totScor+1;i++){

for(int j=0;j<totScor+1;j++){

if(scores[j]<=scores[j+1]){

//swaps scores

hold=scores[j];

scores[j]=scores[j+1];

scores[j+1]=hold;

//swaps names

nameHold=names[j];

names[j]=names[j+1];

names[j+1]=nameHold;

}else{}

}

}

//cout out the scores

cout<<"High Scores:"<<endl;

cout<<"NAME"<<setw(13)<<"WAVES"<<endl;

for(int k=0;k<totScor;k++){

cout<<fixed<<setw(5)<<left

<<names[k]

<<setw(10)<<right

<<scores[k]<<endl;

}

hSListI.close();

//write to file to record scores

hSListO.open("highScoresList.txt",ios::out);

//hSListO.open("highScores.txt",ios::out);

if (!hSListO){

cerr<<endl<<"High Score Output File is missing!"<<endl;

exit(1);

}

for(int l=0;l<pos+1;l++){

hSListO<<names[l]<<" "<<scores[l]<<" "<<endl;

}

hSListO.close();

//ask user to restart

cout<<endl

<<"To start over, press 1."<<endl

<<"Press any other key to quit."<<endl;

cout<<"///////////////////////////////"<<endl;

cin>>cntinue;

//game will loop or end depending on player choice

switch(cntinue){

case 1:{

//reset player HP and reset wave counter

cout<<"///////////////////////////////"<<endl;

cout<<"Loading character creation..."<<endl;

cout<<"///////////////////////////////"<<endl;

eStats[0]=0;

pStats[0]=100;

wave=0;

break;

}

default:{

//quits game

cout<<"///////////////////////////////"<<endl;

cout<<"Thanks for playing!"<<endl;

cout<<"///////////////////////////////"<<endl;

gamExit=true;

}

}

}else if(wave>=50){

//beaten game, exit loop

cout<<"Congratulations! You've survived!"<<endl;

cout<<plyrNme<<"'s Best Record: "<<wave<<" waves!"<<endl<<endl;

//show score then save score to file

hSListI.open("highScoresList.txt",ios::in);

if (!hSListI) {

cerr<<endl<<"High Score Input File is missing!"<<endl;

exit(1);

}

//get all names

while(hSListI>>names[place]>>scores[pos]){

place++;

pos++;

}

//add curr player to list

names[place+1]=plyrNme;

totScor=pos+1;

//add player wave

scores[pos+1]=wave;

//sort scores in order

for(int i=0;i<totScor+1;i++){

for(int j=0;j<totScor+1;j++){

if(scores[j]<=scores[j+1]){

//swaps scores

hold=scores[j];

scores[j]=scores[j+1];

scores[j+1]=hold;

//swaps names

nameHold=names[j];

names[j]=names[j+1];

names[j+1]=nameHold;

}else{}

}

}

//cout out the scores

cout<<"High Scores:"<<endl;

cout<<"NAME"<<setw(13)<<"WAVES"<<endl;

for(int k=0;k<totScor;k++){

cout<<fixed<<setw(5)<<left

<<names[k]

<<setw(10)<<right

<<scores[k]<<endl;

}

hSListI.close();

//write to file to record scores

hSListO.open("highScoresList.txt",ios::out);

//hSListO.open("highScores.txt",ios::out);

if (!hSListO){

cerr<<endl<<"High Score Output File is missing!"<<endl;

exit(1);

}

for(int l=0;l<pos+1;l++){

hSListO<<names[l]<<" "<<scores[l]<<" "<<endl;

}

hSListO.close();

//ask user to retry

cout<<"To start over, press 1."<<endl

<<"Press any other key to quit."<<endl;

cout<<"///////////////////////////////"<<endl;

cin>>cntinue;

switch(cntinue){

case 1:{

//reset player HP and reset wave counter

cout<<"///////////////////////////////"<<endl;

cout<<"Loading character creation..."<<endl;

cout<<"///////////////////////////////"<<endl;

eStats[0]=0;

pStats[0]=100;

wave=0;

break;

}

default:{

cout<<"///////////////////////////////"<<endl;

cout<<"Thanks for playing!"<<endl;

cout<<"///////////////////////////////"<<endl;

gamExit=true;

}

}

}

}//end of while loop for game

//exit stage right!

return 0;

}

void nKnight(int a[]){//generate stats for new knight

a[0]=250;//base hp

a[1]=rand()%(29-19)+19;//generates attack from 19 to 28

a[2]=rand()%(21-15)+15;//generates defense from 15 to 20

a[3]=rand()%(16-9)+9;//generates magic defense from 9 to 15

a[4]=rand()%(11-7)+7;//generates M.defense + healing factor from 7 to 10

//tell player stats

cout<<"Here are your stats:"<<endl

<<"Health: "<<a[0]<<endl

<<"Attack: "<<a[1]<<endl

<<"Defense: "<<a[2]<<endl

<<"Intelligence: "<<a[3]<<endl

<<"Spirit: "<<a[4]<<endl;

cout<<"///////////////////////////////"<<endl;

}

void nWizard(int a[]){//generate stats for new wizard

a[0]=250;//hp=250

a[1]=rand()%(11-5)+5;//generates attack

a[2]=rand()%(16-9)+9;//generates defense

a[3]=rand()%(29-25)+25;//generates magic defense

a[4]=rand()%(19-14)+14;//generates M.defense + healing factor

cout<<"Here are your stats:"<<endl

<<"Health: "<<a[0]<<endl

<<"Attack: "<<a[1]<<endl

<<"Defense: "<<a[2]<<endl

<<"Intelligence: "<<a[3]<<endl

<<"Spirit: "<<a[4]<<endl;

cout<<"///////////////////////////////"<<endl;

}

void nGlditr(int a[]){//generate stats for new gladitor

a[0]=300;//hp=300

a[1]=rand()%(20-17)+17;//generates attack

a[2]=rand()%(29-25)+25;//generates defense

a[3]=rand()%(16-10)+10;//generates magic defense

a[4]=rand()%(16-9)+9;//generates M.defense + healing factor

cout<<"Here are your stats:"<<endl

<<"Health: "<<a[0]<<endl

<<"Attack: "<<a[1]<<endl

<<"Defense: "<<a[2]<<endl

<<"Intelligence: "<<a[3]<<endl

<<"Spirit: "<<a[4]<<endl;

cout<<"///////////////////////////////"<<endl;

}

void nCleric(int a[]){//generate stats for new cleric

a[0]=250;//hp=250

a[1]=rand()%(11-5)+5;//generates attack

a[2]=rand()%(15-10)+10;//generates defense

a[3]=rand()%(23-19)+20;//generates magic defense

a[4]=rand()%(29-25)+25;//generates M.defense + healing factor

cout<<"Here are your stats:"<<endl

<<"Health: "<<a[0]<<endl

<<"Attack: "<<a[1]<<endl

<<"Defense: "<<a[2]<<endl

<<"Intelligence: "<<a[3]<<endl

<<"Spirit: "<<a[4]<<endl;

cout<<"///////////////////////////////"<<endl;

}

void nOnKght(int a[]){//generates stats for new onion knight-

//random extremes are common

a[0]=200;//hp=200

//stats are extreme! Very high or low

a[1]=rand()%(29-5)+5;//generates attack

a[2]=rand()%(29-5)+5;//generates defense

a[3]=rand()%(29-5)+5;//generates magic defense

a[4]=rand()%(29-5)+5;//generates M.defense + healing factor

cout<<"Here are your stats:"<<endl

<<"Health: "<<a[0]<<endl

<<"Attack: "<<a[1]<<endl

<<"Defense: "<<a[2]<<endl

<<"Intelligence: "<<a[3]<<endl

<<"Spirit: "<<a[4]<<endl;

cout<<"///////////////////////////////"<<endl;

}

void doBattle(int pCh,int p[],int m[]){//calculates battle damage

/\* 0=HP

\* 1=ATK

\* 2=DEF

\* 3=MATK

\* 4=SPR

\*/

if(pCh==1){

//calculate physical damage

int dmgMax=((p[1]\*2)-m[2])\*2,

dmgLow=((p[1]\*2)-m[2]);

//difference\*3

int totDmg=rand()%(dmgMax-dmgLow)+dmgLow;

//make sure to turn negatives/zero to 1

if(totDmg<=0){totDmg=1;}

//implement critical

int crit=(rand()%4)+1;//1-4 = 25% chance for crit; 1=crit true

if(crit==1){

totDmg\*=2;

m[0]-=totDmg;//subtract damage from enemy

cout<<"Critical hit!!"<<endl

<<"You did "<<totDmg<<" physical damage!"<<endl;

}else{

m[0]-=totDmg;//subtract damage from enemy

cout<<"You did "<<totDmg<<" physical damage!"<<endl;

}

}else{

//calculate magic damage

int dmgMax=((p[3]\*2)-m[4])\*2,

dmgLow=((p[3]\*2)-m[4]);

//difference\*3

int totDmg=rand()%(dmgMax-dmgLow)+dmgLow;

//make sure to turn negatives/zero to 1

if(totDmg<=0){totDmg=1;}

//implement critical

int crit=(rand()%4)+1;//1-4 = 25% chance for crit; 1=crit true

if(crit==1){

totDmg\*=2;

m[0]-=totDmg;//subtract damage from enemy

cout<<"Critical hit!!"<<endl

<<"You did "<<totDmg<<" magic damage!"<<endl;

}else{

m[0]-=totDmg;//subtract damage from enemy

cout<<"You did "<<totDmg<<" magic damage!"<<endl;

}

}

}

void recvr(string clssTyp,int a[]){//calculates unit healing

//recover a random amount with small deviation

int tmp,max=a[4],low=(a[4]/2);

tmp=(rand()%(max-low)+low)+20;

a[0]+=tmp;

if(clssTyp=="Onion Knight"){

if(a[0]>200){a[0]=200;}

}//sets health to max if greater

else if(clssTyp=="Gladiator"){

if(a[0]>300){a[0]=300;}

}

else{

if(a[0]>250){

a[0]=250;

}

}

cout<<"You recovered "<<tmp<<" HP!"<<endl

<<"Your HP is at "<<a[0]<<"."<<endl;

}

void defend(bool &pDefend){

pDefend=true;

cout<<"You ready your shield..."<<endl;

}

void emySpwn(string &mnstNme,int m[],int WAVE){//spawn enemy - random stats

//generate boss monster when wave is in increments of 10

if(WAVE==10||WAVE==20||WAVE==30||WAVE==40||WAVE==50){

//boss name

int temp=rand()%3+1;

if(temp==1){mnstNme="Bahamut";}

else if(temp==2){mnstNme="Titan";}

else if(temp==3){mnstNme="Garuda";}

else{mnstNme="Ifrit";}

//boss stats

m[0]=250;//hp=250 b/c it is a boss

m[1]=rand()%(29-25)+25;//generates attack

m[2]=rand()%(29-25)+25;//generates defense

m[3]=rand()%(29-25)+25;//generates magic attack

m[4]=rand()%(29-25)+25;//generates M.defense + healing factor

}

else{

//generate generic monster type

int temp=rand()%5+1;

if(temp==1){mnstNme="Goblin";}

else if(temp==2){mnstNme="Orc";}//will have extra HP

else if(temp==3){mnstNme="Wolf";}

else if(temp==4){mnstNme="Tiger";}

else{mnstNme="Fairy";}

//generate generic monster stats

if(mnstNme=="Orc"){

m[0]=150;

}else{

m[0]=100;//hp=100

}

m[1]=rand()%(20-10)+10;//generates attack

m[2]=rand()%(20-10)+10;//generates defense

m[3]=rand()%(20-10)+10;//generates magic attack

m[4]=rand()%(25-10)+10;//generates M.defense + healing factor

}

}

void emyDmg(int p[],int m[],bool &pGrd){//damage enemy deals

//use rand for enemy decision

int tmpResp=rand()%2+1;//gives either 0 or 1

//determine damage

if(tmpResp==1){

//calculate physical damage

int dmgMax=((m[1]\*2)-p[2])\*2,

dmgLow=((m[1]\*2)-p[2]);

int totDmg=rand()%(dmgMax-dmgLow)+dmgLow;

if(totDmg<=0){

totDmg=1;

}

//implement critical

int crit=rand()%4+1;//1-4 = 25% chance for crit; 1=crit true

if(crit==1){

totDmg\*=2;

//check for guard=true

if(pGrd){

//greatly reduce damage

if(totDmg<=1){

totDmg=1;

}else{

totDmg/=2;

}

cout<<"You defend yourself!"<<endl;

//reset guard command from player

pGrd=false;

}

p[0]-=totDmg;//subtract damage from player

cout<<"Enemy critical Hit!!"<<endl

<<"Enemy did "<<totDmg<<" physical damage!"<<endl;

}else{

//check for guard=true

if(pGrd){

//greatly reduce damage

if(totDmg<=1){

totDmg=1;

}else{

totDmg/=2;

}

cout<<"You defend yourself!"<<endl;

//reset guard command from player

pGrd=false;

}

p[0]-=totDmg;//subtract damage from player

cout<<"Enemy did "<<totDmg<<" physical damage!"<<endl;

}

}

else{

//calculate magic damage

int dmgMax=((m[3]\*2)-p[4])\*2,

dmgLow=((m[3]\*2)-p[4]);

int totDmg=rand()%(dmgMax-dmgLow)+dmgLow;//difference\*3

if(totDmg<=0){totDmg=1;}//make sure to turn negatives/zero to 1

//implement critical

int crit=rand()%4+1;//1-4 = 25% chance for crit; 1=crit true

if(crit==1){

totDmg\*=2;

if(pGrd){

//greatly reduce damage

if(totDmg<=1){

totDmg=1;

}else{

totDmg/=2;

}

cout<<"You defend yourself!"<<endl;

//reset guard command from player

pGrd=false;

}

p[0]-=totDmg;//subtract damage from player

cout<<"Enemy critical hit!!"<<endl

<<"Enemy did "<<totDmg<<" magic damage!"<<endl;

}else{

if(pGrd){

//greatly reduce damage

if(totDmg<=1){

totDmg=1;

}else{

totDmg/=2;

}

cout<<"You defend yourself!"<<endl;

//reset guard command from player

pGrd=false;

}

p[0]-=totDmg;//subtract damage from player

cout<<"Enemy did "<<totDmg<<" magic damage!"<<endl;

}

}

}

void emyHeal(string mnstNme,int m[], int WAVE){//amount enemy recovers

//recover a random amount with small deviation

int tmp,max=m[4],low=(m[4]/2);

tmp=(rand()%(max-low)+low)+15;

m[0]+=tmp;

if(WAVE==10||WAVE==20||WAVE==30||WAVE==40||WAVE==50&&m[0]>250){

m[0]=250;//resets boss hp to 200 if greater

}

else if(mnstNme=="Orc"&&m[0]>150){

m[0]=150;

}else{

//resets health to 100 if greater and normal enemy

if(m[0]>100){

m[0]=100;

}else{

//do nothing / shouldn't get here

}

}

cout<<"Enemy recovered "<<tmp<<" HP!"<<endl;

}