. .\*+ **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 1**

Title

**FANTASY WAVE SURVIVAL**

Class

**CSC-05**

Section

**42450**

Due Date

**April 28,2014**

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 very randomly distributed among all areas. So Onion Knight is like 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 tenth 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: 523 lines

Number of variables: 42

Number of functions: 10

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. This program has a lot more potential is far from done in my eyes.

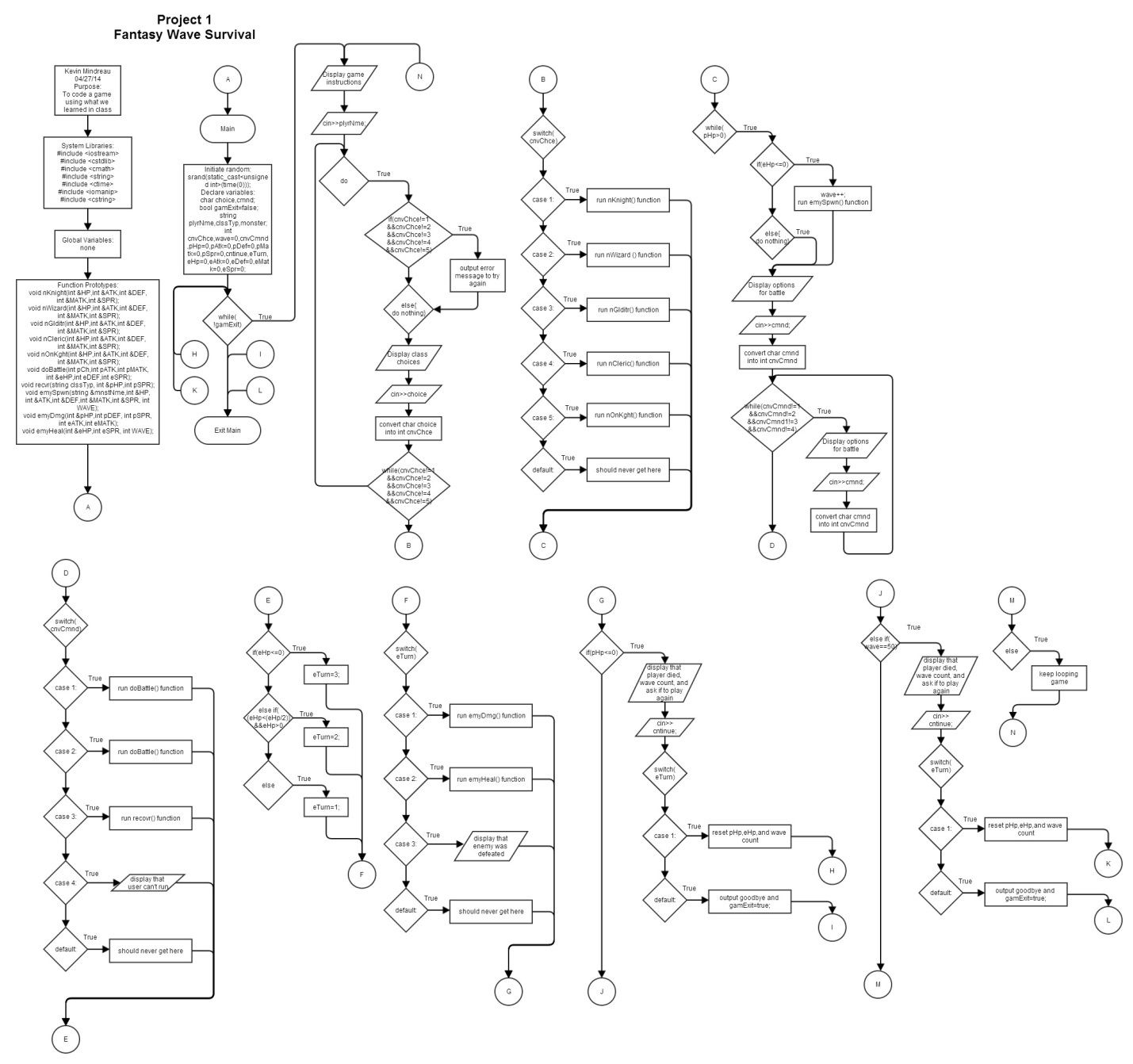
An example would like implementing a limit to how much magic the player can cast, dodging, critical hits, high-score lists, etc.

This project took me about a week 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. 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.

**Flowchart**



**Pseudo Code**

*Initialize rand and declare variables*

*While Loop until gamExit = true*

*Explain game*

*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 = 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 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 and 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. |
|  | eTurn | Int value for enemy’s turn. |
|  | pHp | Player health points. |
|  | pAtk | Player attack points. |
|  | pDef | Player physical defense points. |
|  | pMatk | Player magic attack points. |
|  | pSpr | Player spirit/magic resistance points; applies when healing. |
|  | eHp | Enemy health points. |
|  | eAtk | Enemy attack points. |
|  | eDef | Enemy physical defense points. |
|  | eMatk | Enemy magic attack points. |
|  | eSpr | Enemy spirit/magic resistance points; applies when healing. |
|  | pCh | Holds value for attack and changes if it was a physical or magical attack. |
| string | plyrNme | String to hold player’s name. |
|  | clssTyp | String that holds class name/type. |
|  | monster | String that holds monster name. |
| 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. |

**Major Functions**

|  |  |  |
| --- | --- | --- |
| Type | Name | Description |
| void | doBattle  (int pCh,int pATK,int pMATK,int &eHP,int eDEF,int eSPR); | Calculates battle damage from player to enemy;  pCh dictates if physical or magical damage |
|  | emyDmg(  int &pHP,int pDEF, int pSPR,int eATK,int eMATK); | Calculates battle damage from enemy to player;  Will randomly choose between physical or magical damage |
|  | emyHeal(  int &eHP,int eSPR, int WAVE); | Calculates enemy healing |
|  | emySpwn(  string &mnstNme,int &HP,int &ATK,int &DEF,int &MATK,int &SPR, 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 &HP,int &ATK,int &DEF,int &MATK,int &SPR); | Generates stats of character; higher attack point with knight |
|  | nGlditr(  int &HP,int &ATK,int &DEF,int &MATK,int &SPR); | Generates stats of character; higher physical defense points with gladiator |
|  | nWizard(  int &HP,int &ATK,int &DEF,int &MATK,int &SPR); | Generates stats of character; higher magic attack points with wizard |
|  | nCleric(  int &HP,int &ATK,int &DEF,int &MATK,int &SPR); | Generates stats of character; higher magic resistance and healing points with cleric |
|  | nOnKght(  int &HP,int &ATK,int &DEF,int &MATK,int &SPR); | Generates stats of character; all stats have a large gap and are randomly generated (wild card class). |
|  | recvr(  string clssTyp, int &pHP,int pSPR); | 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

**Program**

//Function Prototypes

void nKnight(int &HP,int &ATK,int &DEF,

int &MATK,int &SPR);//generate stats for new knight

void nWizard(int &HP,int &ATK,int &DEF,

int &MATK,int &SPR);//generate stats for new wizard

void nGlditr(int &HP,int &ATK,int &DEF,

int &MATK,int &SPR);//generate stats for new gladitor

void nCleric(int &HP,int &ATK,int &DEF,

int &MATK,int &SPR);//generate stats for new cleric

void nOnKght(int &HP,int &ATK,int &DEF,

int &MATK,int &SPR);//generates stats for new onion knight-

//random extremes are common

void doBattle(int pCh,int pATK,int pMATK,

int &eHP,int eDEF,int eSPR);//calculates battle damage

void recvr(string clssTyp, int &pHP,int pSPR);//calculates unit healing

void emySpwn(string &mnstNme,int &HP,int &ATK,int &DEF,

int &MATK,int &SPR, int WAVE);//spawn enemy - random stats

void emyDmg(int &pHP,int pDEF, int pSPR,

int eATK,int eMATK);//damage enemy deals

void emyHeal(int &eHP,int eSPR, int WAVE);//amount enemy recovers

//Execution Begins Here!

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

//initialize rand

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

//declare variables

char choice,cmnd;

bool gamExit=false,clsLoop=true;

string plyrNme,clssTyp,monster;//general info

int cnvChce,wave=0,cnvCmnd,pHp=0,pAtk=0,pDef=0,pMatk=0,pSpr=0,//player stats

cntinue,eTurn,eHp=0,eAtk=0,eDef=0,eMatk=0,eSpr=0;//enemy stats

//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;

//explain game

cout<<"This game is an RPG wave survival game!"<<endl

<<"How the game works:"<<endl

<<"One turn = A Player action then an Enemy action."<<endl

<<"The player can attack or heal on their turn."<<endl<<endl

<<"Goal:"<<endl

<<"Try and survive 50 waves of enemies!"<<endl

<<"But beware, every 10 waves a boss will appear!"<<endl

<<"Oh...and you can't flee from battle."<<endl

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

<<"Good luck and let us begin!!!"<<endl<<endl;

//ask user for name and class choice

cout<<"What is your name? ";

cin>>plyrNme;

cout<<endl;

do{

//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;

}else{}

//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';

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

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

do{

switch(cnvChce){

case 1:{

//Knight

clssTyp="Knight";

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

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

//generate stats

nKnight(pHp,pAtk,pDef,pMatk,pSpr);

clsLoop=false;

break;

}

case 2:{

//Wizard

clssTyp="Wizard";

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

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

//generate stats

nWizard(pHp,pAtk,pDef,pMatk,pSpr);

clsLoop=false;

break;

}

case 3:{

//Gladiator

clssTyp="Gladiator";

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

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

//generate stats

nGlditr(pHp,pAtk,pDef,pMatk,pSpr);

clsLoop=false;

break;

}

case 4:{

//Cleric

clssTyp="Cleric";

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

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

//generate stats

nCleric(pHp,pAtk,pDef,pMatk,pSpr);

clsLoop=false;

break;

}

case 5:{

//Onion Knight

clssTyp="Onion Knight";

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

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

//generate stats

nOnKght(pHp,pAtk,pDef,pMatk,pSpr);

clsLoop=false;

break;

}

default:{

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

}

}

}while(clsLoop);

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

while(pHp>0){

//generate enemy

if(eHp<=0){

wave++;

emySpwn(monster,eHp,eAtk,eDef,eMatk,eSpr,wave);

}

else{}

//player turn

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

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

//create user interface

<<left

<<plyrNme<<"'s HP = "<<pHp

<<right<<setw(15)

<<monster<<"'s HP = "<<eHp<<endl

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

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

<<"1. Attack"<<endl

<<"2. Magic"<<endl

<<"3. Heal"<<endl

<<"4. Flee"<<endl

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

cin>>cmnd;

cnvCmnd=cmnd-'0';

while(cnvCmnd!=1&&cnvCmnd!=2&&cnvCmnd!=3&&cnvCmnd!=4){

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,pAtk,pMatk,eHp,eDef,eSpr);

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

break;

}

case 2:{//magical attack

int pCh=2;//function parameter for magic

doBattle(pCh,pAtk,pMatk,eHp,eDef,eSpr);

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

break;

}

case 3:{

recvr(clssTyp,pHp,pSpr);

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

break;

}

case 4:{

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

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

break;

}

default:{

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

}

}

//enemy turn

//determine action of enemy based on hp levels

//if zero or less hp = dead

if(eHp<=0){eTurn=3;}

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

else if((eHp<(eHp/2))&&eHp>0){eTurn=2;}

//otherwise attack

else{eTurn=1;}

//menu for enemy actions

switch(eTurn){

case 1:{

//physical or magic attack, randomly chosen

emyDmg(pHp,pDef,pSpr,eAtk,eMatk);

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

break;

}

case 2:{

//healing action

emyHeal(eHp,eSpr,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(pHp<=0){

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

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

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

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

eHp=0;

pHp=100;

wave=0;

break;

}

default:{

//quits game

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

<<"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;

eHp=0;

pHp=100;

wave=0;

break;

}

default:{

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

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

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

gamExit=true;

}

}

}else{}

}

//exit stage right!

return 0;

}

void nKnight(int &HP,int &ATK,int &DEF,

int &MATK,int &SPR){//generate stats for new knight

HP=250;//base hp

ATK=rand()%(29-19)+19;//generates attack from 19 to 28

DEF=rand()%(21-15)+15;//generates defense from 15 to 20

MATK=rand()%(16-9)+9;//generates magic defense from 9 to 15

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

//tell player stats

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

<<"Health: "<<HP<<endl

<<"Attack: "<<ATK<<endl

<<"Defense: "<<DEF<<endl

<<"Intelligence: "<<MATK<<endl

<<"Spirit: "<<SPR<<endl;

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

}

void nWizard(int &HP,int &ATK,int &DEF,

int &MATK,int &SPR){//generate stats for new wizard

HP=250;//hp=250

ATK=rand()%(11-5)+5;//generates attack

DEF=rand()%(16-9)+9;//generates defense

MATK=rand()%(29-25)+25;//generates magic defense

SPR=rand()%(19-14)+14;//generates M.defense + healing factor

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

<<"Health: "<<HP<<endl

<<"Attack: "<<ATK<<endl

<<"Defense: "<<DEF<<endl

<<"Intelligence: "<<MATK<<endl

<<"Spirit: "<<SPR<<endl;

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

}

void nGlditr(int &HP,int &ATK,int &DEF,

int &MATK,int &SPR){//generate stats for new gladitor

HP=300;//hp=300

ATK=rand()%(20-17)+17;//generates attack

DEF=rand()%(29-25)+25;//generates defense

MATK=rand()%(16-10)+10;//generates magic defense

SPR=rand()%(16-9)+9;//generates M.defense + healing factor

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

<<"Health: "<<HP<<endl

<<"Attack: "<<ATK<<endl

<<"Defense: "<<DEF<<endl

<<"Intelligence: "<<MATK<<endl

<<"Spirit: "<<SPR<<endl;

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

}

void nCleric(int &HP,int &ATK,int &DEF,

int &MATK,int &SPR){//generate stats for new cleric

HP=250;//hp=250

ATK=rand()%(11-5)+5;//generates attack

DEF=rand()%(15-10)+10;//generates defense

MATK=rand()%(23-19)+20;//generates magic defense

SPR=rand()%(29-25)+25;//generates M.defense + healing factor

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

<<"Health: "<<HP<<endl

<<"Attack: "<<ATK<<endl

<<"Defense: "<<DEF<<endl

<<"Intelligence: "<<MATK<<endl

<<"Spirit: "<<SPR<<endl;

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

}

void nOnKght(int &HP,int &ATK,int &DEF,

int &MATK,int &SPR){//generates stats for new onion knight-

//random extremes are common

HP=200;//hp=200

//stats are extreme! Very high or low

ATK=rand()%(29-5)+5;//generates attack

DEF=rand()%(29-5)+5;//generates defense

MATK=rand()%(29-5)+5;//generates magic defense

SPR=rand()%(29-5)+5;//generates M.defense + healing factor

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

<<"Health: "<<HP<<endl

<<"Attack: "<<ATK<<endl

<<"Defense: "<<DEF<<endl

<<"Intelligence: "<<MATK<<endl

<<"Spirit: "<<SPR<<endl;

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

}

void doBattle(int pCh,int pATK,int pMATK,

int &eHP,int eDEF,int eSPR){//calculates battle damage

//char tmpResp;

int cnvResp=pCh;

//determine damage

if(cnvResp==1){

//calculate physical damage

int dmgMax=(pATK-eDEF)\*3,dmgLow=(pATK-(eDEF/2))\*2;

if(dmgMax<=0||dmgLow<=0){

if(dmgMax<=0){

dmgMax=(pATK+eDEF)/2;

dmgLow=(pATK+eDEF)/3;

}else{}

}else{}

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

if(totDmg<=0){totDmg=(dmgMax+dmgLow)/2;}//make sure to turn negatives/zero to 1

else{}

eHP-=totDmg;//subtract damage from enemy

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

}

else if(cnvResp==2){

//calculate magic damage

int dmgMax=(pMATK-eSPR)\*3,dmgLow=(pMATK-(eSPR/2))\*2;

if(dmgMax<=0||dmgLow<=0){

if(dmgMax<=0){

dmgMax=(pMATK+eSPR)/2;

dmgLow=(pMATK+eSPR)/3;

}else{}

}else{}

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

if(totDmg<=0){totDmg=(dmgMax+dmgLow)/2;}//make sure to turn negatives/zero to 1

else{}

eHP-=totDmg;//subtract damage from enemy

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

}

else{}

}

void recvr(string clssTyp,int &pHP,int pSPR){//calculates unit healing

//recover a random amount with small deviation

int tmp,max=pSPR,low=(pSPR/2);

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

pHP+=tmp;

if(clssTyp=="Onion Knight"){

if(pHP>200){pHP=200;}

else{}

}//sets health to 100 if greater

else if(pHP>250){pHP=250;}

else{}

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

<<"Your HP is at "<<pHP<<"."<<endl;

}

void emySpwn(string &mnstNme,int &HP,int &ATK,int &DEF,

int &MATK,int &SPR, 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

HP=250;//hp=250 b/c it is a boss

ATK=rand()%(29-25)+25;//generates attack

DEF=rand()%(29-25)+25;//generates defense

MATK=rand()%(29-25)+25;//generates magic defense

SPR=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";}

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

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

else{mnstNme="Fairy";}

//generate generic monster stats

HP=100;//hp=100

ATK=rand()%(23-6)+6;//generates attack

DEF=rand()%(23-6)+6;//generates defense

MATK=rand()%(23-6)+6;//generates magic defense

SPR=rand()%(25-6)+6;//generates M.defense + healing factor

}

}

void emyDmg(int &pHP,int pDEF, int pSPR,

int eATK,int eMATK){//damage enemy deals

//calculate damage reduction

int tmpResp;

//use rand for enemy decision

tmpResp=rand()%2+1;

//determine damage

if(tmpResp==1){

//calculate physical damage

int dmgMax=(eATK-pDEF)\*3,dmgLow=(eATK-(pDEF/2))\*2;

if(dmgMax<=0||dmgLow<=0){

if(dmgMax<=0){

dmgMax=(eATK+pDEF)/3;

dmgLow=(eATK+pDEF)/4;

}else{}

}else{}

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

if(totDmg<=0){totDmg=(dmgMax+dmgLow)/2;}//make sure to turn negatives/zero to 1

else{}

pHP-=totDmg;//subtract damage from player

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

}

else if(tmpResp==2){

//calculate magic damage

int dmgMax=(eMATK-pSPR)\*3,dmgLow=(eMATK-(pSPR/2))\*2;

if(dmgMax<=0||dmgLow<=0){

if(dmgMax<=0){

dmgMax=(eMATK+pSPR)/3;

dmgLow=(eMATK+pSPR)/4;

}else{}

}else{}

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

if(totDmg<=0){totDmg=(dmgMax+dmgLow)/2;}//make sure to turn negatives/zero to 1

else{}

pHP-=totDmg;//subtract damage from player

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

}

else{}

}

void emyHeal(int &eHP,int eSPR, int WAVE){//amount enemy recovers

//recover a random amount with small deviation

int tmp,max=eSPR,low=(eSPR/2);

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

eHP+=tmp;

if(WAVE==10||WAVE==20||WAVE==30||WAVE==40||WAVE==50&&eHP>250){

if(eHP>250){eHP=250;}//resets boss hp to 200 if greater

else{}

}

else if(eHP>250){

eHP=100;//resets health to 100 if greater and normal enemy

}

else{}

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

}