Final Project

“Ave Imperator”

David South

Dr. Mark Lehr

CIS 17A

12/8/13

**Introduction**

About the Game

Ave Imperator in a gladiatorial combat simulation that puts you right in the middle of the Coliseum against the best gladiator the empire has to offer. At the start of the game you will be able to customize your gladiator by choosing the weapon, armor and fighting style that fits your preferred play type. Your opponents, as heroes of the empire are all pre-generated. Combat is dealt in simultaneous action rounds until one combatant is reduced below five hit points. After combat is resolved, the player (if alive) will be allowed to advance or quit. There is no save feature at this time.

Why this project is important

This project is important as a step in understanding how more complex programs are built, as well as a look at efficiency. Outside of a classroom setting, this project is important to me as a foil to decide if programming is really something I want to pursue. I have been interested in game design and development for many years, and being able to design one from the ground up, even one this easy, stokes the desire. On top of this, I had great fun developing this project. It has shown me that what I really do enjoy about the game development cycle is tuning and tweaking ideas to create different experiences.

How to Play

As previously mentioned, the game opens with character creation. Naming your

Character is just the start. Choosing your weapons and armor, and selecting a fighting style have real consequences to how combat is played out. At the end of character creation, you will be given the option to start over, or enter the arena. Once combat begins, the gladiators are announced, salute the emperor and chose their actions. Turns are taken simultaneously allowing for damage mitigation from blocks and dodges, and counter attacks through parries. There are also spots of inaction when both players chose a not attacking action. Combat is decided when one of the gladiators is taken below five health. At five health, the player is presented with the option to surrender, accepting the emperor’s mercy. The opponent simply loses. There are five total opponents of increasing difficulty. After each opponent is defeated, the player is given a choice to advance to the next or quit the game. At the end of combat, victory is declared for the winning side.

Project Summary:

Coded Lines: 1546 (including comments)

Functions in Main: 22

Supporting CPP files: 2

Class files: 4

Templates: 4

Variable types include:

Void, Class, fstream, Template and basic int, short, float, char and string data types. The project also includes text files for reading about the game and how to play.

Project Experience

At the outset, I was unhappy with my previous project (from Project 1) and decided to start over with something that was more of a game instead of a statistical model. The idea for Ave Imperator came to me watching a Youtube trailer for Ryse: Son of Rome while I was currently thinking about building a text based RPG for the final project. Deciding that a text based game was going to be too text heavy, I opted for a combination RPG combat simulator. During the initial design phase, I had created a system that became very cluttered and bloated with too many variables and options to work through in combat. During the development cycle, some initial options I had wanted to include. One of these was to pull the information for the game’s “How To” section in to a structure and read it out. Similar to Project 1, I was unable to successfully store the file input into a structure for output and reverted back to using text files. Another value I had hoped to include a “favor” system that provided different outcomes depending on how well a player did in combat. The combat simulation was also going to originally be more robust, with a system for counters, special attacks and health recovery. While I did not experience significant difficulty with the assignment, there were some specific implementations that were more difficult than others. Overwriting the base variables proved problematic until correctly using polymorphism. Even though it was one of my favorite parts, balancing the gameplay and getting the variables to adjust properly with the chosen player variables was also a bit time consuming. The end game is functional, but there are still some “easy mode” discrepancies due to it not being fully balanced. My main reference was out class textbook “Starting out with C++: From Control Structures through Objects” seventh ed. By Tony Gaddis.

Description

The Program

The solution from the beginning was somewhat fractured while I tried to decide the best way to implement some of the features. Initially, in order to save coding time and lines, my idea was to use .dat files to load all of the weapon, armor, style and base stat information based on the initial choices of the player. I built a dat builder driver program to create the data files, but was unsuccessful in getting them to read information on demand. After the initial set up issues I reformatted the way the program was going to work. I did not fully develop top down, as I started with the character creation and combat modules then wrote the program header and about files. For the most part, development was a linear process. Classes that control the player choices were created first, with the functions to drive them created as needed. The last class built was the combat class containing the templates for processing combat variables. Some user verification exists, but there is a basic assumption that the correct inputs will be used. The menu system was reduced to integer input to assist in this. Formatting was added to provide an easier way to view the players chosen stats.

The Game

At load, the user is presented with the title screen and prompted to enter to continue.





After entering, the player is provided a menu to play, read about the game and read the how to.

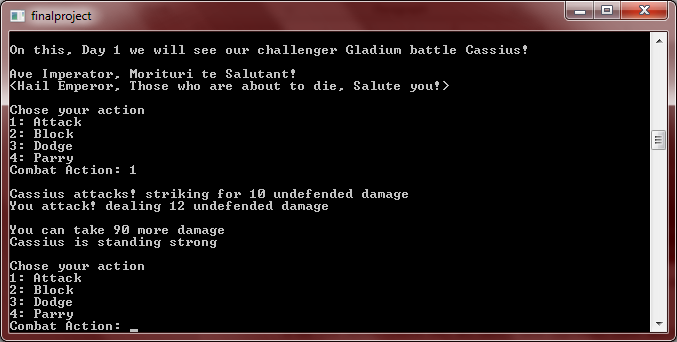
Playing the game begins with entering the chosen name of the gladiator, followed by your preferred weapon, armor and fighting style. After choices are made, the player is presented with their character and given the option to remake them or enter the coliseum.



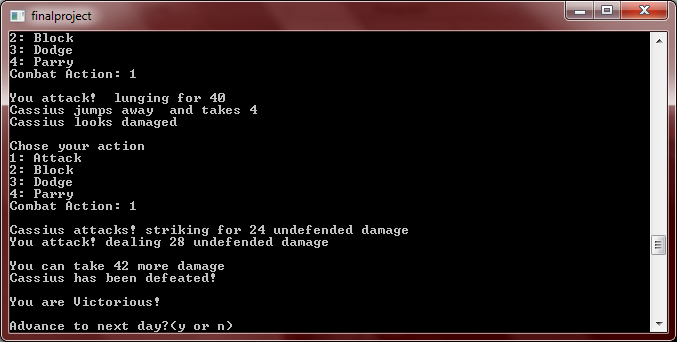
The gladiator and combat stats are derived from the player choices and displayed as a check for the user to gauge how they should play the character they created.

If the user enters combat they are shown their opponents name and enter combat after a brief salute to the emperor. Stats for the opponent are not shown.

Users chose their combat action at the same time that the computer opponent does, providing simultaneous combat.



Action resolution is shown for both combatants is shown, with damage being dealt. The opponent’s health is not shown, but his condition is displayed when he takes damage. At the end of combat, the user is prompted to advance to the next day, or quit the game.







Main Program (Ave Imperator)

//functional includes

#include <iostream>

#include <fstream>

#include <cstdlib>

//class includes

#include "Player.h"

#include "HowToPlay.h"

#include "Combat.h"

using namespace std;

//no global constants

//function prototypes

void header();

void mainMenu();

void playGame();

void about();

void howToPlay();

Player mkStats();

int getWepChoice();

int \*getWeaponStats(int);

int getArmrChoice();

int \*getArmrStats(int);

int getStyleChoice();

int \*getStyleStats(int);

int \*getOpp(int);

string getOppName(int);

char combat(string, int\*, string, int \*, int);

int getCpuAction();

int \*getCmbtActs(int, int);

int getAttack(int &, int &);

int getDamage(int &, int &);

int getDodge(int &, int &);

int getParry(int &, int &);

void getOppHealth(string, int &);

//begin program

int main() {

//set precision to 2 decimal places

cout.setf(ios::fixed);

cout.setf(ios::showpoint);

cout.precision(2);

//welcome and menu

header();

cout << endl << "Welcome to Ave Imperator!";

cout << endl << endl << "Press Enter to continue.";

cin.ignore();

cout << endl;

mainMenu();

//exit program

return 0;

}

void header() {

string title; //shows title mainMenu

fstream titleFile; //gets tile mainMenu

// Open the file in input mode.

titleFile.open("title.txt", ios::in);

// If the file was successfully opened, continue.

if (titleFile) {

// Read an item from the file.

getline(titleFile, title);

// read in file.

while (titleFile) {

// Display the last item read.

cout << title << endl;

// Read the next item.

getline(titleFile, title);

}

// Close the file.

titleFile.close();

} else {

cout << "ERROR: Cannot open file.\n";

}

}

void mainMenu() {

int main;

cout << endl << "Main Menu\n"

<< "----------------------\n"

<< "1: Play Ave Imperator\n"

<< "2: About Ave Imperator\n"

<< "3: How To Play\n"

<< "4: Exit Ave Imperator\n"

<< "----------------------";

cout << endl << "Menu Selection: ";

cin >> main;

//check for out of bounds

while (main < 1 || main > 4) {

cout << endl << "You have entered and invalid menu choice, please re-enter";

}

if (main == 1) {

playGame();

} else if (main == 2) {

about();

} else if (main == 3) {

howToPlay();

}

}

void playGame() {

char play; //char to continue playing or exit combat

char begin; //check to enter combat

int round = 1; //sets combat round

int MAXROUNDS = 5; //total number of rounds

string name;

int statsSize = 6;

int \*plyrStats = new int[statsSize];

Player stats;

//get user choices

stats = mkStats();

//display user selected stats

stats.prntPlyrStrgs();

stats.prntPlyrStats();

//prompt to begin combat

cout << endl;

cout << "Would you like to enter the Colosseum with this Gladiator?(y or n)";

cin >> begin;

if (begin == 'y') {

for (int i = 0; i < MAXROUNDS; i++) {

int \*opponent;

string oppName;

opponent = getOpp(round);

oppName = getOppName(round);

//set user combat stats

name = stats.getPlryName();

plyrStats[0] = stats.getHealth();

plyrStats[1] = stats.getPlryAtkP();

plyrStats[2] = stats.getPlryAtkSp();

plyrStats[3] = stats.getPlryBlk();

plyrStats[4] = stats.getPlryDdge();

plyrStats[5] = stats.getPlryPrry();

//run combat

play = combat(name, plyrStats, oppName, opponent, round);

//combat prompts for continue and returns value

if (play == 'y') {

round++;

} else if (play == 'n') {

cout << endl << "Thank you for playing Ave Imperator!";

cout << endl;

exit(1);

}

}

} else {

playGame();

}

cin.ignore();

}

void about() {

cout << endl;

string title; //shows title mainMenu

fstream titleFile; //gets tile mainMenu

// Open the file in input mode.

titleFile.open("about.txt", ios::in);

// If the file was successfully opened, continue.

if (titleFile) {

// Read an item from the file.

getline(titleFile, title);

// read in file.

while (titleFile) {

// Display the last item read.

cout << title << endl;

// Read the next item.

getline(titleFile, title);

}

// Close the file.

titleFile.close();

} else {

cout << "ERROR: Cannot open file.\n";

}

mainMenu();

}

void howToPlay() {

int menu;

//HowTo Menu

cout << endl << "How To Menu\n"

<< "--------------------\n"

<< "1: Playing the Game\n"

<< "2: Gladiator Stats\n"

<< "3: Combat\n"

<< "4: Weapons and Armor\n"

<< "5: Return to Main Menu\n"

<< "--------------------";

cout << endl << "How To: ";

cin >> menu;

//bounds check

while (menu < 1 || menu > 5) {

cout << endl << cout << endl << "You have entered and invalid menu choice, please re-enter";

}

if (menu == 1) {

cout << endl;

string title; //shows title mainMenu

fstream titleFile; //gets tile mainMenu

// Open the file in input mode.

titleFile.open("gamePlay.txt", ios::in);

// If the file was successfully opened, continue.

if (titleFile) {

// Read an item from the file.

getline(titleFile, title);

// read in file.

while (titleFile) {

// Display the last item read.

cout << title << endl;

// Read the next item.

getline(titleFile, title);

}

// Close the file.

titleFile.close();

} else {

cout << "ERROR: Cannot open file.\n";

}

}

if (menu == 2) {

cout << endl;

string title; //shows title mainMenu

fstream titleFile; //gets tile mainMenu

// Open the file in input mode.

titleFile.open("playerStats.txt", ios::in);

// If the file was successfully opened, continue.

if (titleFile) {

// Read an item from the file.

getline(titleFile, title);

// read in file.

while (titleFile) {

// Display the last item read.

cout << title << endl;

// Read the next item.

getline(titleFile, title);

}

// Close the file.

titleFile.close();

} else {

cout << "ERROR: Cannot open file.\n";

}

}

if (menu == 3) {

cout << endl;

string title; //shows title mainMenu

fstream titleFile; //gets tile mainMenu

// Open the file in input mode.

titleFile.open("combat.txt", ios::in);

// If the file was successfully opened, continue.

if (titleFile) {

// Read an item from the file.

getline(titleFile, title);

// read in file.

while (titleFile) {

// Display the last item read.

cout << title << endl;

// Read the next item.

getline(titleFile, title);

}

// Close the file.

titleFile.close();

} else {

cout << "ERROR: Cannot open file.\n";

}

}

if (menu == 4) {

cout << endl;

string title; //shows title mainMenu

fstream titleFile; //gets tile mainMenu

// Open the file in input mode.

titleFile.open("wepArmr.txt", ios::in);

// If the file was successfully opened, continue.

if (titleFile) {

// Read an item from the file.

getline(titleFile, title);

// read in file.

while (titleFile) {

// Display the last item read.

cout << title << endl;

// Read the next item.

getline(titleFile, title);

}

// Close the file.

titleFile.close();

} else {

cout << "ERROR: Cannot open file.\n";

}

}

if (menu == 5) {

mainMenu();

}

howToPlay();

}

Player mkStats() {

//getter vars

int getPlyrWep;

int \*wepStats;

int getPlyrArmr;

int \*armrStats;

int getPlyrStyl;

int \*styleStats;

//variables

string name; //get player name

string style;

string weapon; //get player weapon

string armor; //get player armor

//adjustments

int plyrHlth; //adjusted health

int plyrFati; //adjusted fatigue

int plyrStr; //adjusted strength

int plyrAgi; //adjusted agility

int plyrDef; //adjusted defense

int plyrMvSp; //adjusted movement

//get player name

cout << "What is your name, Gladiator! ";

cin >> name;

cin.ignore();

cout << endl;

//get weapon

getPlyrWep = getWepChoice();

wepStats = getWeaponStats(getPlyrWep);

if (getPlyrWep == 1) {

weapon = "Short Sword";

} else if (getPlyrWep == 2) {

weapon = "One-Handed Mace";

} else if (getPlyrWep == 3) {

weapon = "One-Handed Hammer";

} else if (getPlyrWep == 4) {

weapon = "One-Handed Flail";

} else if (getPlyrWep == 5) {

weapon = "Two-Handed Long Sword";

} else if (getPlyrWep == 6) {

weapon = "Two-Handed Hammer";

} else if (getPlyrWep == 7) {

weapon = "Two Handed Trident";

} else if (getPlyrWep == 8) {

weapon = "Two Short Swords";

}

cout << endl;

//get armor

getPlyrArmr = getArmrChoice();

armrStats = getArmrStats(getPlyrArmr);

if (getPlyrArmr == 1) {

armor = "Half Leather";

} else if (getPlyrArmr == 2) {

armor = "Full Leather";

} else if (getPlyrArmr == 3) {

armor = "Half Ring";

} else if (getPlyrArmr == 4) {

armor = "Full Ring";

} else if (getPlyrArmr == 5) {

armor = "Half Bronze";

} else if (getPlyrArmr == 6) {

armor = "Full Bronze";

}

cout << endl;

//style menu

getPlyrStyl = getStyleChoice();

styleStats = getStyleStats(getPlyrStyl);

//style, health, strength, fatigue, agility, defense, movement

if (getPlyrStyl == 1) {

style = "Agile";

} else if (getPlyrStyl == 2) {

style = "Tactful";

} else if (getPlyrStyl == 3) {

style = "Brute";

} else if (getPlyrStyl == 4) {

style = "Tank";

}

//fill adjusted stats

//wepStats0 = damage, wepStats1 = speed, wepStats2 = reach, wepStats3 = defAdj, wepStats4 = adjAdj, wepStats5 = fatiAdj

//armrStats0 = defAdj, armrStats1 = adjAdj, armrStats2 = fatAdj, armrStats3 = moveAdj

//styleStats[0] = health,

Player stats(name, style, weapon, wepStats[0], wepStats[1], wepStats[2], wepStats[3], wepStats[4], wepStats[5], armor, armrStats[0], armrStats[1], armrStats[2], armrStats[3]);

plyrHlth = stats.adjHealth(styleStats[0]);

plyrFati = stats.adjFatigue(styleStats[1], wepStats[5], armrStats[2]);

plyrStr = stats.adjStrength(styleStats[2]);

plyrAgi = stats.adjAgility(styleStats[3], wepStats[4], armrStats[1]);

plyrDef = stats.adjDefense(styleStats[4], wepStats[3], armrStats[0]);

plyrMvSp = stats.adjMovement(styleStats[5], armrStats[3]);

stats.setAttributes(plyrHlth, plyrFati, plyrStr, plyrAgi, plyrDef, plyrMvSp);

stats.setPlyrAtkP(wepStats[0], wepStats[1]);

stats.setPlyrAtkSp(wepStats[1]);

stats.setPlyrBlk(wepStats[3], armrStats[0]);

stats.setPlyrDdge();

stats.setPlyrPrry();

//return

return stats;

}

int getWepChoice() {

int plyrWep; //holds weapon choice

cout << "What weapon do you fight with? " << endl;

cout << "1: One-Handed Short Sword\n"

<< "2: One-Handed Mace\n"

<< "3: One-Handed Hammer\n"

<< "4: One-Handed Flail\n"

<< "5: Two-Handed Long Sword\n"

<< "6: Two-Handed Hammer\n"

<< "7: Two Handed Trident\n"

<< "8: Two Short Swords\n"

<< "Weapon Choice: ";

cin >> plyrWep;

//return weapon

return plyrWep;

}

int \*getWeaponStats(int choice) {

int size = 6;

int \*wepStats = new int[size];

int wpDmg; //weapon damage

int wpSpd; //weapon speed

int wpRch; //weapon reach

int wpDef; //defense adjustment from weapon

int wpAgi; //agility adjustment from weapon

int wpFti; //fatigue adjustment from weapon

//type, damage, speed, reach, defense adj, agility adj

if (choice == 1) {

wpDmg = 5;

wpSpd = 4;

wpRch = 1;

wpDef = 0;

wpAgi = 0;

wpFti = 0;

} else if (choice == 2) {

} else if (choice == 3) {

} else if (choice == 4) {

} else if (choice == 5) {

} else if (choice == 6) {

} else if (choice == 7) {

} else if (choice == 8) {

}

//set values

wepStats[0] = wpDmg;

wepStats[1] = wpSpd;

wepStats[2] = wpRch;

wepStats[3] = wpDef;

wepStats[4] = wpAgi;

wepStats[5] = wpFti;

//return

return wepStats;

}

int getArmrChoice() {

int plyrArmr;

cout << "What armor do you wear? " << endl;

cout << "1: Half Leather\n"

<< "2: Full Leather\n"

<< "3: Half Ring\n"

<< "4: Full Ring\n"

<< "5: Half Bronze\n"

<< "6: Full Bronze\n"

<< "Armor Choice: ";

cin >> plyrArmr;

//return armor choice

return plyrArmr;

}

int \*getArmrStats(int choice) {

int size = 4;

int \*armrStats = new int[size];

int arDef; //defense adjustment from armor

int arAgi; //agility adjustment from armor

int arFti; //fatigue adjustment from armor

int arMv; //movement adjustment from armor

//type, defense adj, agility adj

if (choice == 1) {

arDef = 3;

arAgi = 2;

arFti = 0;

arMv = 0;

} else if (choice == 2) {

} else if (choice == 3) {

} else if (choice == 4) {

} else if (choice == 5) {

} else if (choice == 6) {

}

//set values

armrStats[0] = arDef;

armrStats[1] = arAgi;

armrStats[2] = arFti;

armrStats[3] = arMv;

//return

return armrStats;

}

int getStyleChoice() {

int style;

cout << "What is your fighting style: " << endl;

cout << "1: Agile\n"

<< "2: Tactful\n"

<< "3: Brute Force\n"

<< "4: Meat Shield\n"

<< "Fighting Style: ";

cin >> style;

//return style

return style;

}

int \*getStyleStats(int choice) {

int size = 6;

int \*styleStats = new int[size];

int hlthAdj; //value to adjust base health

int fatiAdj; //value to adjust base fatigue

int strAdj; //value to adjust base strength

int agiAdj; //value to adjust base agility

int defAdj; //value to adjust base defense

int moveAdj; //value to adjust base movement

//style, health, strength, fatigue, agility, defense, movement

if (choice == 1) {

hlthAdj = 0;

fatiAdj = 16;

strAdj = 10;

agiAdj = 14;

defAdj = 10;

moveAdj = 10;

} else if (choice == 2) {

} else if (choice == 3) {

} else if (choice == 4) {

}

//set values

styleStats[0] = hlthAdj;

styleStats[1] = fatiAdj;

styleStats[2] = strAdj;

styleStats[3] = agiAdj;

styleStats[4] = defAdj;

styleStats[5] = moveAdj;

//return

return styleStats;

}

int \*getOpp(int round) {

int size = 5;

int \*opponent = new int[size];

string oppName;

//opponent[0] = health, opponent[1] = attack power, opponent[2] = attack speed opponent[3] = block rating, opponent[4] = dodge rating, opponent[5] = parry rating

if (round == 1) {

opponent[0] = 90;

opponent[1] = 35;

opponent[2] = 35;

opponent[3] = 20;

opponent[4] = 20;

opponent[5] = 20;

} else if (round == 2) {

} else if (round == 3) {

} else if (round == 4) {

} else if (round == 5) {

}

//return opponent

return opponent;

}

string getOppName(int round) {

string name;

if (round == 1) {

name = "Cassius";

} else if (round == 2) {

name = "Markus";

} else if (round == 3) {

name = "Brutus Titus";

} else if (round == 4) {

name = "Oram the Slave";

} else if (round == 5) {

name = "Alexander Aurellis Antonius";

}

//return

return name;

}

char combat(string plyrName, int \*plyrStats, string oppName, int \*oppStats, int round) {

char advance;

char wounded;

int cmbtAct;

int cpuCombat;

cout << endl << "Welcome to the Tournament of the Emperor!" << endl;

cout << endl << "On this, Day " << round << " we will see our challenger " << plyrName <<

" battle " << oppName << "!\n";

cout << endl << "Ave Imperator, Morituri te Salutant!\n<Hail Emperor, Those who are about to die, Salute you!>" << endl;

//[0] = health, [1] = attack power, [2] = attack speed, [3] = block rating, [4] = dodge rating, [5] = parry rating

cin.ignore();

//combat loop

while (oppStats[0] >= 5) {

//reset all variables at the beginning of combat rounds

int plyrAtk = 0;

int plyrDmg = 0;

int plyrBlk = 0;

int plyrDdge = 0;

int plyrPrry = 0;

int oppAtk = 0;

int oppDmg = 0;

int oppBlk = 0;

int oppDdge = 0;

int oppPrry = 0;

//get cpu action

cpuCombat = getCpuAction();

//player combat menu

cout << endl << "Chose your action\n"

<< "1: Attack\n"

<< "2: Block\n"

<< "3: Dodge\n"

<< "4: Parry\n"

<< "Combat Action: ";

cin >> cmbtAct;

//both attack

if (cpuCombat == 1 && cmbtAct == 1) {

cout << endl;

//opponent attack

cout << oppName << " attacks! ";

oppAtk = getAttack(oppStats[2], plyrStats[3]);

oppDmg = getDamage(oppStats[1], oppAtk);

cout << "striking for " << oppDmg << " undefended damage" << endl;

//player attack

cout << "You attack! ";

plyrAtk = getAttack(plyrStats[2], oppStats[3]);

plyrDmg = getDamage(plyrStats[1], plyrAtk);

cout << "dealing " << plyrDmg << " undefended damage" << endl;

//damage

plyrStats[0] = plyrStats[0] - oppDmg;

oppStats[0] = oppStats[0] - plyrDmg;

cout << endl << "You can take " << plyrStats[0] << " more damage";

getOppHealth(oppName, oppStats[0]);

}

//both block

if (cpuCombat == 2 && cmbtAct == 2) {

//no combat result

cout << endl;

cout << oppName << " takes a defensive position";

cout << endl;

cout << "You prepare for an incoming attack";

}

//both dodge

if (cpuCombat == 3 && cmbtAct == 3) {

//no combat result

cout << endl;

cout << oppName << " slides away from you";

cout << endl;

cout << "You slide away from an attack";

}

//both parry

if (cpuCombat == 4 && cmbtAct == 4) {

//no combat result

cout << endl;

cout << oppName << " looks for an incoming attack";

cout << endl;

cout << "You are anticipating an incoming attack";

}

//opponent attack - player block

if (cpuCombat == 1 && cmbtAct == 2) {

cout << endl;

//opponent attack

cout << oppName << " attacks! ";

oppAtk = getAttack(oppStats[2], plyrStats[3]);

oppDmg = getDamage(oppStats[1], oppAtk);

cout << " swinging for " << oppDmg << endl;

//player blocks

cout << "You take a defensive stance ";

plyrBlk = plyrStats[3];

oppDmg = oppDmg - plyrBlk;

if (oppDmg < 0) {

cout << endl << "blocking all of the damage";

oppDmg = 0;

} else {

cout << " but take " << oppDmg << " damage";

}

//damage

plyrStats[0] = plyrStats[0] - oppDmg;

cout << "You can take " << plyrStats[0] << " more damage";

}

//opponent attack - player dodge

if (cpuCombat == 1 && cmbtAct == 3) {

cout << endl;

//opponent attack

cout << oppName << " attacks! ";

oppAtk = getAttack(oppStats[2], plyrStats[3]);

oppDmg = getDamage(oppStats[1], oppAtk);

cout << " striking for " << oppDmg << endl;

//player dodge

cout << "You move away ";

plyrDdge = getDodge(plyrStats[4], oppAtk);

oppDmg = oppDmg - plyrDdge;

if (oppDmg < 0) {

cout << endl << "dodging all of the damage";

oppDmg = 0;

} else {

cout << " but still take " << oppDmg << " damage";

}

//damage

plyrStats[0] = plyrStats[0] - oppDmg;

cout << "You can take " << plyrStats[0] << " more damage";

}

//opponent attacks - player paries

if (cpuCombat == 1 && cmbtAct == 4) {

cout << endl;

//opponent attack

cout << oppName << " attacks! " << endl;

oppAtk = getAttack(oppStats[2], plyrStats[3]);

oppDmg = getDamage(oppStats[1], oppAtk);

//player parry

cout << "You move to parry" << endl;

plyrPrry = getParry(plyrStats[5], oppAtk); //parry uses versus attack power

if (plyrPrry > oppDmg) {

cout << "deflecting the damage and returning ";

plyrDmg = plyrStats[1]\*.4;

cout << plyrDmg;

} else {

cout << " deflecting the damage\nbut are unable to counter" << endl;

}

oppStats[0] = oppStats[0] - plyrDmg;

getOppHealth(oppName, oppStats[0]);

}

//opponent blocks - player attacks

if (cpuCombat == 2 && cmbtAct == 1) {

cout << endl;

//player attack

cout << "You attack! ";

plyrAtk = getAttack(plyrStats[2], oppStats[3]);

plyrDmg = getDamage(plyrStats[1], plyrAtk);

cout << " striking for " << plyrDmg << endl;

//opponent blocks

cout << oppName << " sets up defensively ";

oppBlk = oppStats[3];

plyrDmg = plyrDmg - oppBlk;

if (plyrDmg < 0) {

cout << endl << "blocking all of the damage";

plyrDmg = 0;

} else {

cout << "but still takes " << plyrDmg << " damage";

}

//damage

oppStats[0] = oppStats[0] - plyrDmg;

getOppHealth(oppName, oppStats[0]);

}

//opponent blocks - player dodges

if (cpuCombat == 2 && cmbtAct == 3) {

cout << endl;

//non combat action

cout << endl;

cout << oppName << " takes a defensive position";

cout << endl;

cout << "You step back from your opponent";

}

//opponent blocks - player paries

if (cpuCombat == 2 && cmbtAct == 4) {

cout << endl;

//non combat action

cout << endl;

cout << oppName << " takes a defensive position";

cout << endl;

cout << "You look to deflect an attack that never comes";

}

//opponent dodges - player attacks

if (cpuCombat == 3 && cmbtAct == 1) {

cout << endl;

//player attack

cout << "You attack! ";

plyrAtk = getAttack(plyrStats[2], oppStats[3]);

plyrDmg = getDamage(plyrStats[1], plyrAtk);

cout << " lunging for " << plyrDmg << endl;

//opponent dodge

cout << oppName << " jumps away ";

oppDdge = getDodge(oppStats[4], plyrAtk);

plyrDmg = plyrDmg - oppDdge;

if (plyrDmg < 0) {

cout << endl << "dodging all of the damage";

plyrDmg = 0;

} else {

cout << " and takes " << plyrDmg;

}

//damage

oppStats[0] = oppStats[0] - plyrDmg;

getOppHealth(oppName, oppStats[0]);

}

//opponent dodges - player blocks

if (cpuCombat == 3 && cmbtAct == 2) {

cout << endl;

//non combat action

cout << endl;

cout << oppName << " slides away from you";

cout << endl;

cout << "You prepare to block an incoming attack";

}

//opponent dodges - player paries

if (cpuCombat == 3 && cmbtAct == 4) {

cout << endl;

//non combat action

cout << endl;

cout << oppName << " jumps back, anticipating an attack";

cout << endl;

cout << "You look to parry an attack, but it never comes";

}

//opponent paries - player attacks

if (cpuCombat == 4 && cmbtAct == 1) {

cout << endl;

//player attack

cout << "You attack! ";

plyrAtk = getAttack(plyrStats[2], oppStats[3]);

plyrDmg = getDamage(plyrStats[1], plyrAtk);

cout << " swinging for " << plyrDmg << endl;

//opponent parry

cout << oppName << " paries! ";

oppPrry = getParry(oppStats[5], plyrAtk); //parry uses versus attack power

if (oppPrry > plyrDmg) {

cout << "deflecting the damage and returning ";

oppDmg = oppStats[1]\*.4;

cout << oppDmg;

} else {

cout << " deflecting the damage\nbut is unable to counter" << endl;

}

plyrStats[0] = plyrStats[0] - oppDmg;

cout << "You can take " << plyrStats[0] << " more damage";

}

//opponent paries - player blocks

if (cpuCombat == 4 && cmbtAct == 2) {

cout << endl;

//non combat action

cout << endl;

cout << oppName << " anticipates an attack, but it never comes";

cout << endl;

cout << "You take up a defensive stance, preparing to block";

}

//opponent paries - player dodges

if (cpuCombat == 4 && cmbtAct == 3) {

cout << endl;

//non combat action

cout << endl;

cout << oppName << " moves to deflect an attack";

cout << endl;

cout << "You slide away from " << oppName;

}

if (plyrStats[0] <= 5 && plyrStats[0] >= 0) {

cout << "You are gravely wounded, will you yield to the emperor's will\n"

"or continue to fight?" << endl << "enter y to yield, n to continue";

cin >> wounded;

if (wounded == 'y') {

cout << endl << "The emperor's favor shines on you today, he wills you to live.";

}

}

if (plyrStats[0] < 0) {

cout << endl << "You have been killed in combat" << endl;

exit(2);

}

}

//end of combat

if (round <= 4) {

cout << endl << "You are Victorious!";

cout << endl << endl << "Advance to next day?(y or n)" << endl;

cout << "\"y\" will advance to the next day, \"n\" will exit the program ";

cin >> advance;

while (advance != 'y' && advance != 'n') {

cout << endl << "You have entered an invalid entry, please re-enter: ";

cin >> advance;

}

return advance;

} else if (round == 5)

{

cout << endl << "You have completed the emperor's tournament!\n"

<< "You shall be rewarded with a position of honor in Rome!" << endl;

}

else {

cout << endl << "Thank you for playing Ave Imperator!";

cout << endl;

exit(1);

}

}

int getCpuAction() {

int oppCh = 0; //cpu choice

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

//set random 1 through 5 for cpu

oppCh = rand() % 4 + 1;

return oppCh;

}

int getAttack(int &attkSpd, int &defRate) {

int attk = 0;

attk = setAttack(attkSpd, defRate);

//return

return attk;

}

int getDamage(int &wepDmg, int &atkMod) {

int dmg = 0;

dmg = setDamage(wepDmg, atkMod);

//return

return dmg;

}

int getDodge(int &dodge, int &atkMod) {

int dodged;

dodged = setDodge(dodge, atkMod);

//return

return dodged;

}

int getParry(int &parRate, int &atkMod) {

int parry;

parry = setParry(parRate, atkMod);

return parry;

}

void getOppHealth(string name, int &health) {

//opponent health status

if (health <= 70 && health > 40) {

cout << endl << name << " is looking slightly weak" << endl;

} else if (health <= 40 && health > 10) {

cout << endl << name << " looks damaged" << endl;

} else if (health <= 10 && health > 0) {

cout << endl << name << " is staggered" << endl;

} else if (health <= 0) {

cout << endl << name << " has been defeated!" << endl;

} else if (health > 70) {

cout << endl << name << " is standing strong" << endl;

}

}

Gladiator.h (base class)

// Specification file for the Gladiator class

#ifndef GLADIATOR\_H

#define GLADIATOR\_H

#include <string>

using namespace std;

class Gladiator {

protected:

int baseHlth; //gladiator health

int baseFti; //gladiator fatigue

int baseStr; //base strength

int baseAgi; //base agility

int baseDef; //base defense

int baseMvSp; //base move speed

public:

//default constructor

Gladiator() {

baseHlth = 100;

baseFti = 20;

baseStr = 10;

baseAgi = 10;

baseDef = 10;

baseMvSp = 10;

}

//base constructor

Gladiator(int hlt, int fat, int str, int agi, int def, int mvSp) {

set(hlt, fat, str, agi, def, mvSp);

}

//set values

void set(int hlt, int fat, int str, int agi, int def, int mvSp) {

baseHlth = hlt;

baseFti = fat;

baseStr = str;

baseAgi = agi;

baseDef = def;

baseMvSp = mvSp;

}

//accessors

int getHealth() const {

return baseHlth;

}

int getFatigue() const {

return baseFti;

}

int getStrength() const {

return baseStr;

}

int getAgility() const {

return baseAgi;

}

int getDefense() const {

return baseDef;

}

int getMoveSpd() const {

return baseMvSp;

}

//virtual manipulators

virtual int adjHealth(int) const = 0;

virtual int adjFatigue(int, int, int) const = 0;

virtual int adjStrength(int) const = 0;

virtual int adjAgility(int, int, int) const = 0;

virtual int adjDefense(int, int, int) const = 0;

virtual int adjMovement(int, int) const = 0;

//display

void prntBase() const;

};

#endif

Gladiator.cpp

//Specification file for Gladiator implimentation

#include<iostream>

#include <iomanip>

#include "Player.h"

using namespace std;

void Gladiator::prntBase() const

{

cout << "Gladiator Stats: " << endl;

cout << "Health" << setw(9) << "Fatigue" << setw(10) << "Strength" << setw(9) << "Agility" << setw(9) << "Defense" << setw(10) << "Movement" << endl;

cout << setw(6) << baseHlth << setw(9) << baseFti << setw(10) << baseStr << setw(9) << baseAgi << setw(9) << baseDef << setw(10) << baseMvSp << endl;

}

Player.h

// Specification file for the Player class

#ifndef PLAYER\_H

#define PLAYER\_H

#include <string>

#include "Gladiator.h"

#include "Weapon.h"

#include "Armor.h"

using namespace std;

class Player : public Gladiator {

private:

Weapon plyrWpn;

Armor plyrArmr;

string plyrName;

string plyrStyl;

int plyrAtkP;

int plyrAtkSp;

int plyrBlk;

int plyrDdge;

int plyrPrry;

int plyrHAdj;

int plyrFAdj;

int plyrAgAj;

int plyrDfAj;

int plyrMvAj;

//int plyrFavor;

public:

//default constructor

Player() : Gladiator() {

plyrName = "";

plyrStyl = "";

plyrAtkP = 0;

plyrAtkSp = 0;

plyrBlk = 0;

plyrDdge = 0;

plyrPrry = 0;

plyrHAdj = 0;

plyrFAdj = 0;

plyrAgAj = 0;

plyrDfAj = 0;

plyrMvAj = 0;

}

Player(int hlt, int fat, int str, int agi, int def, int mvSp) :

Gladiator(hlt, fat, str, agi, def, mvSp) {

plyrName = "";

plyrStyl = "";

plyrAtkP = 0;

plyrAtkSp = 0;

plyrBlk = 0;

plyrDdge = 0;

plyrPrry = 0;

plyrHAdj = 0;

plyrFAdj = 0;

plyrAgAj = 0;

plyrDfAj = 0;

plyrMvAj = 0;

}

//base constructor

Player(string name, string style, string weapon, int dmg, int speed, int reach, int wDfAdj, int wAgiAdj, int wFtiAdj, string armor, int aDefAdj, int aAgiAdj, int aFtiAdj, int aMvAdj) {

plyrName = name;

plyrStyl = style;

plyrWpn.set(weapon, dmg, speed, reach, wDfAdj, wAgiAdj, wFtiAdj);

plyrArmr.set(armor, aDefAdj, aAgiAdj, aFtiAdj, aMvAdj);

}

// Mutator functions

void setAttributes(int hlt, int fat, int str, int agi, int def, int mvSpd);

void setPlyrAtkP(int wpDmg, int wpSpd); //str, wpDmg, wpSpd

void setPlyrAtkSp(int wpSpd); //wp spd, agility, move speed

void setPlyrBlk(int wpDef, int arDef); //wep def adj, armr def adj, str def

void setPlyrDdge(); //armor mv adj, agility, mvsp

void setPlyrPrry(); //str, agi, wpRch

string getPlryName() const {

return plyrName;

}

string getPlryStyl() const {

return plyrStyl;

}

int getPlryAtkP() const {

return plyrAtkP;

}

int getPlryAtkSp() const {

return plyrAtkSp;

}

int getPlryBlk() const {

return plyrBlk;

}

int getPlryDdge() const {

return plyrDdge;

}

int getPlryPrry() const {

return plyrPrry;

}

//virtual variables

virtual int adjHealth(int) const;

virtual int adjFatigue(int, int, int) const;

virtual int adjStrength(int) const;

virtual int adjAgility(int, int, int) const;

virtual int adjDefense(int, int, int) const;

virtual int adjMovement(int, int) const;

//print functions

void prntPlyrStrgs() const;

void prntPlyrStats() const;

};

#endif

Player.cpp

//Specification file for Player implimentation

#include<iostream>

#include <iomanip>

#include "Player.h"

using namespace std;

int Player::adjHealth(int hlth) const {

int health; //new health

//calculate new

health = baseHlth + hlth;

//return new health

return health;

}

int Player::adjFatigue(int fti, int wpFti, int arFti) const {

int fatigue; //new fatigue

//calculate new

fatigue = baseFti + fti - (wpFti + arFti);

//return new fatigue

return fatigue;

}

int Player::adjStrength(int str) const {

int strength; //new strength

//calculate new

strength = baseStr + str;

//return new strength

return strength;

}

int Player::adjAgility(int agi, int wpAgi, int arAgi) const {

int agility; //new agility

//calculate new

agility = baseAgi + agi - (wpAgi + arAgi);

//return new agility

return agility;

}

int Player::adjDefense(int def, int wpDef, int arDef) const {

int defense; //new defense

//calculate new

defense = baseDef + def + wpDef + arDef;

//return new defense

return defense;

}

int Player::adjMovement(int mv, int arMv) const {

int movement; //new movement

//calculate new

movement = baseMvSp + mv - arMv;

//return new movement

return movement;

}

void Player::setAttributes(int hlt, int fat, int str, int agi, int def, int mvSpd) {

baseHlth = hlt; //store adjusted health

baseFti = fat; //store adjusted fatigue

baseStr = str; //store adjusted strength

baseAgi = agi; //store adjusted agility

baseDef = def; //store adjusted defense

baseMvSp = mvSpd; //store adjusted movement

}

void Player::setPlyrAtkP(int wpDmg, int wpSpd) {

plyrAtkP = ((baseStr \* (wpDmg + wpSpd))/4);

}

void Player::setPlyrAtkSp(int wpSpd) {

plyrAtkSp = ((baseAgi + (wpSpd + baseMvSp))\*.9);

}

void Player::setPlyrBlk(int wpDef, int arDef) {

plyrBlk = (wpDef+arDef+((baseStr + baseDef)/2));

}

void Player::setPlyrDdge() {

plyrDdge = ((baseAgi + baseMvSp)/2);

}

void Player::setPlyrPrry() {

plyrPrry = ((baseStr + baseAgi)\*.6);

}

void Player::prntPlyrStrgs() const {

cout << endl << "--------------------------------------------------------------" << endl;

cout << "Gladiator Name: " << plyrName << endl << endl;

cout << "Fighting Style: " << plyrStyl << endl << endl;

plyrWpn.prntWep();

cout << endl;

plyrArmr.prntArmr();

cout << endl;

prntBase();

cout << endl;

}

void Player::prntPlyrStats() const {

cout << "Combat Stats: " << endl;

cout << "--------------------------------------------------------------" << endl;

cout << "Attack Power: " << plyrAtkP << endl;

cout << "Attack Speed: " << plyrAtkSp << endl;

cout << "Block Rating: " << plyrBlk << endl;

cout << "Dodge Rating: " << plyrDdge << endl;

cout << "Parry Rating: " << plyrPrry << endl;

cout << "--------------------------------------------------------------" << endl;

}

Weapon.h

// Specification file for the Weapon class

#ifndef WEAPON\_H

#define WEAPON\_H

#include <string>

#include <iomanip>

using namespace std;

class Weapon {

private:

string wpName;

int wpDamage;

int wpSpeed;

int wpReach;

int wpDefAdj;

int wpAgiAdj;

int wpFtiAdj;

public:

//default constructor

Weapon() {

set("", 0, 0, 0, 0, 0, 0);

}

//build constructor

Weapon(string weapon, int dmg, int speed, int reach, int wDefAdj, int wAgiAdj, int wFtiAdj) {

set(weapon, dmg, speed, reach, wDefAdj, wAgiAdj, wFtiAdj);

}

//mutators

void set(string weapon, int dmg, int speed, int reach, int wDefAdj, int wAgiAdj, int wFtiAdj) {

wpName = weapon;

wpDamage = dmg;

wpSpeed = speed;

wpReach = reach;

wpDefAdj = wDefAdj;

wpAgiAdj = wAgiAdj;

wpFtiAdj = wFtiAdj;

}

string getWpName() const {

return wpName;

}

int getWpDamage() const {

return wpDamage;

}

int getWpSpeed() const {

return wpSpeed;

}

int getWpReach() const {

return wpReach;

}

int getWpDefAdj() const {

return wpDefAdj;

}

int getWpAgiAdj() const {

return wpAgiAdj;

}

int getWpFtiAdj() const {

return wpFtiAdj;

}

void prntWep() const {

cout << "Weapon" << setw(wpName.length()+2) << "Damage" << setw(8) << "Speed" << setw(7) << "Reach" << setw(9) << "Def Adj" << setw(9) << "Agi Adj" << setw(13) << "Fatigue Adj" << endl

<< wpName << setw(8) << wpDamage << setw(8) << wpSpeed << setw(7) << wpReach << setw(9) << wpDefAdj << setw(9) << wpAgiAdj << setw(13) << wpFtiAdj << endl;

}

};

#endif

Armor.h

// Specification file for the Armor class

#ifndef ARMOR\_H

#define ARMOR\_H

#include <string>

#include <iomanip>

using namespace std;

class Armor {

private:

string arName;

int arDefAdj;

int arAgiAdj;

int arFtiAdj;

int arMvAdj;

public:

//default constructor

Armor() {

set("", 0, 0, 0, 0);

}

//base constructor

Armor(string armor, int aDefAdj, int aAgiAdj, int aFtiAdj, int aMvAdj) {

set(armor, aDefAdj, aAgiAdj, aFtiAdj, aMvAdj);

}

//mutators

void set(string armor, int aDefAdj, int aAgiAdj, int aFtiAdj, int aMvAdj) {

arName = armor;

arDefAdj = aDefAdj;

arAgiAdj = aAgiAdj;

arFtiAdj = aFtiAdj;

arMvAdj = aMvAdj;

}

string getArName() const {

return arName;

}

int getArDefAdj() const {

return arDefAdj;

}

int getArAgiAdj() const {

return arAgiAdj;

}

int getArFtiAfj() const {

return arFtiAdj;

}

int getArMvAdj() const {

return arMvAdj;

}

void prntArmr() const {

cout << "Armor" << setw(arName.length()+4) << "Def Adj" << setw(9) << "Agi Adj" << setw(13) << "Fatigue Adj" << setw(14) << "Movement Adj" << endl

<< arName << setw(9) << arDefAdj << setw(9) << arAgiAdj << setw(13) << arFtiAdj << setw(14) << arMvAdj << endl;

}

};

#endif

Combat.h

//specification file for Combat class

#ifndef COMBAT\_H

#define COMBAT\_H

#include <iostream>

#include <cstdlib>

#include <ctime>

using namespace std;

template <class T>

T setAttack(T attkSpd, T defRate)

{

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

int percent = rand() % 10 + 1; //add random adjustment

int atkPerc; //set attack

atkPerc = (attkSpd/defRate); //base percent

atkPerc = atkPerc + percent; //adjusted percent

return atkPerc;

}

template <class T>

T setDamage(T attkDmg, T attkPerc)

{

int damage; //damage

float perc = attkPerc\*.10; //percent adjustment

damage = ((attkDmg\*perc)); //total damage

return damage;

}

template <class T>

T setDodge(T defense, T attkPerc)

{

int dodged;

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

int percent = rand() % 10 + 1; //add random adjustment

float perc = percent\*.20;

dodged = ((defense\*perc)); //total damage

//return

return dodged;

}

template <class T>

T setParry(T parRating, T atkRate)

{

float parry;

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

int adjust = rand() % 10 + 1; //add random adjustment

parry = ((parRating/atkRate)\*adjust);

cout << endl << "Parry: " << parry << endl;

//return

return parry;

}

#endif /\* COMBAT\_H \*/