Radoslav Genov

ID: 100116626

Email: rpg140@aubg.edu

COS 221 Fundamental Data Structures

Prof. Christozov

**Project Specifications**

The project that I developed consists of a fighting game. The three-level hierarchy, is in the characters of the game. There will be a Character base class, deriving from it is a Hero class and a Villain class, and from them deriving is different classes with the names of special characters, which are shown below. The user is able to choose which character he wants to fight with. There is a specialty function which is polymorphic and is specific for every character. The characters are stored in a vector data structure. In addition, the player is able to sort the waves of enemies by the Quick Sort algorithm, in order of some specification (either health or attack power).

Character

Zombie

Ghost

Samurai

Knight

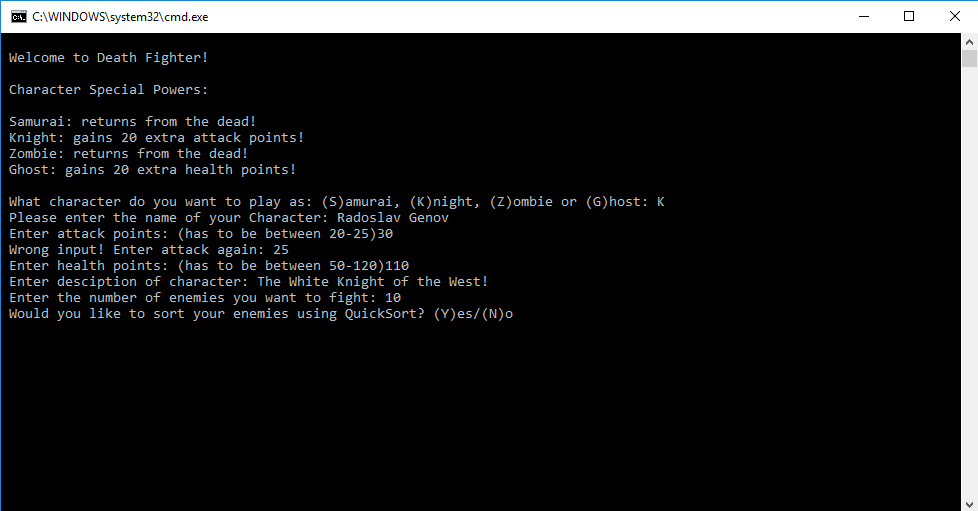
Villain

Hero

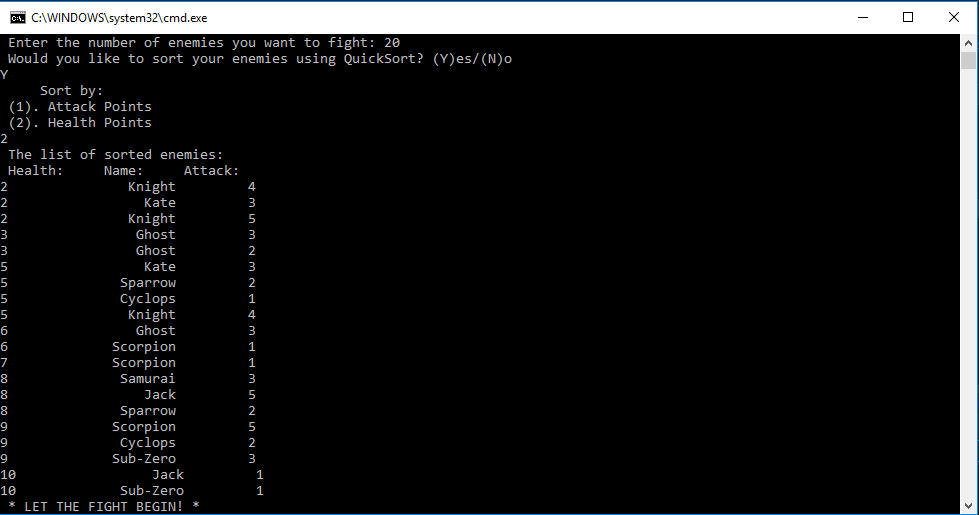
**User Manual:**

1. First, the user must choose a character, the choices are either, Knight, Samurai, Zombie and Ghost. Each character span different attack power, health, and a special ability (which is hidden in the code, it is the virtual function).
2. Second, the user can add a name for his character.
3. Then, the user can decide the power level points of his character for his players attack and health points. (They have a different range restriction for different characters)
4. Then he can add a description of his character.
5. After, the user can choose how many enemies he wants to fight.
6. After that the user is prompted whether he would like to Quicksort the list of his enemies either by attack power or by health. If not he can begin playing and fight!
7. Fighting is done by either (a)ttacking or (d)efending, by attacking you kill your enemies and lose more health, if you choose to defend your character loses less health but does not kill his enemies.
8. Once the enemies are either all killed or your health has depleted, you either lose or win the game, then can either continue playing or exit.

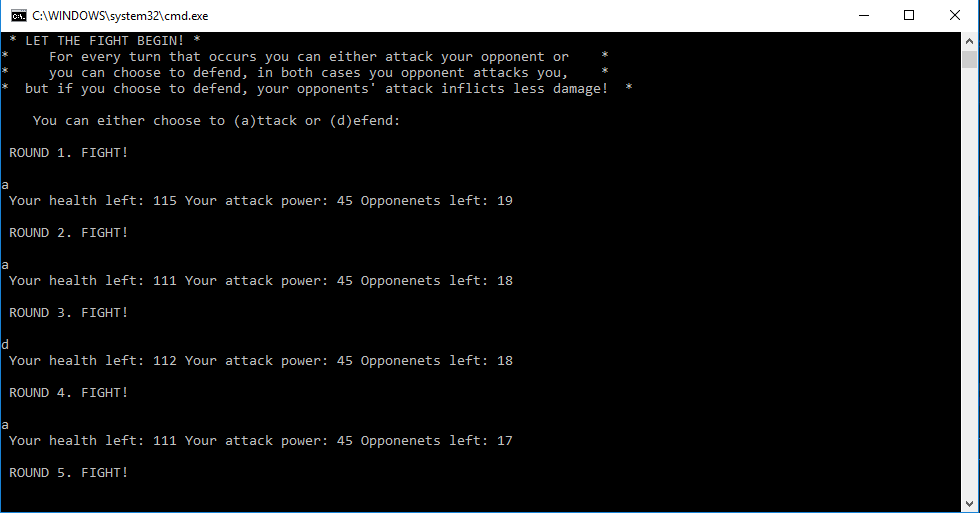
**Screen Shots:**



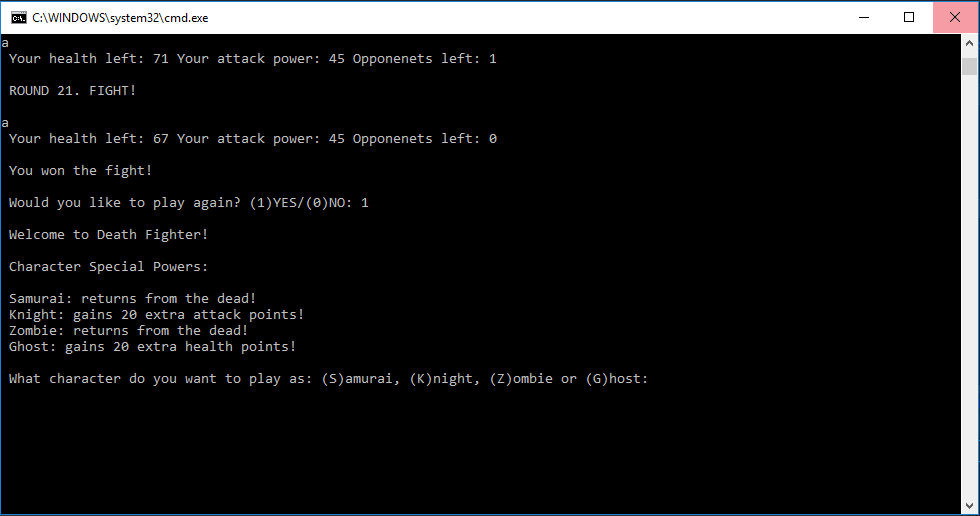
1. A screen shot of the beginning menu of the game. (Choosing character, name, attack, health, and description)



1. Screen shot of the sorted vector of enemies, by QuickSort, by health power.



1. Screen shot of Game Play. The user either attacking or defending.



1. Final screen shot of the game being over, and the use choosing to play another game.

**Code:**

#include <iostream>

#include <vector>

#include <string>

#include <iomanip>

#include "Character.h"

#include "Hero.h"

#include "Villain.h"

#include "Quicksort.h"

//FDS Individual Class Project Game

//Radoslav Genov

using namespace std;

//Hierarchy Level 3

class Samurai : public Hero

{

public:

Samurai()

: special(0) {};

void addCharacter();

void setSpecialty();

void zeroSpecialty() { special = 0; }

int getSpecialty() { return special; }

private:

int special;

};

void Samurai::addCharacter() {

Hero::addCharacter();

}

void Samurai::setSpecialty() {

special = 1;

}

class Knight : public Hero

{

public:

Knight()

: special(0) {};

void addCharacter();

void setSpecialty();

void zeroSpecialty() { special = 0; }

int getSpecialty() { return special; }

private:

int special;

};

void Knight::addCharacter() {

Hero::addCharacter();

}

void Knight::setSpecialty() {

special = 2;

}

class Zombie : public Villain

{

public:

Zombie()

: special(0) {};

void addCharacter();

void setSpecialty();

void zeroSpecialty() { special = 0; }

int getSpecialty() { return special; }

private:

int special;

};

void Zombie::addCharacter() {

Villain::addCharacter();

}

void Zombie::setSpecialty() {

special = 1;

}

class Ghost : public Villain

{

public:

Ghost()

: special(0) {};

void addCharacter();

void setSpecialty();

void zeroSpecialty() { special = 0; }

int getSpecialty() { return special; }

private:

int special;

};

void Ghost::addCharacter() {

Villain::addCharacter();

}

void Ghost::setSpecialty() {

special = 3;

}

vector<Character\*> randEnemies();

Character\* chooseCharacter();

void driver();

int main() {//loops while the user wants to continue

int playAgain;

do

{

driver();

cout << endl;

cout << " Would you like to play again? (1)YES/(0)NO: ";

cin >> playAgain;

} while (playAgain);

}

void driver() {//the driver of the game

cout << endl;

cout << " Welcome to Death Fighter! " << endl;

cout << endl;

Character\* character = chooseCharacter();

vector<Character\*> enemies = randEnemies();

vector<Character\*> tempEnemies = enemies;

cout << " Would you like to sort your enemies using QuickSort? (Y)es/(N)o " << endl;

char read;

cin >> read;

if (read == 'Y') {

//use different vector for quicksort so it does not to mess with gameplay

sort(tempEnemies);

cout << " The list of sorted enemies: " << endl;

cout << " Health: Name: Attack: " << endl;

for (size\_t i = 0; i < tempEnemies.size(); i++)

{

cout << tempEnemies.at(i)->getHealth() << " " << setw(20) << tempEnemies.at(i)->getName() << setw(10) << tempEnemies.at(i)->getAttack() << endl;

}

}

cout << " \* LET THE FIGHT BEGIN! \* " << endl;

if (character->fight(enemies, character))//checks whether you won or lost

{

cout << endl;

cout << " You lost the fight! " << endl;

}

else {

cout << endl;

cout << " You won the fight! " << endl;

}

delete character;

}

Character\* chooseCharacter() {

Character\* character;

cout << " Character Special Powers: " << endl;

cout << endl;

cout << " Samurai: returns from the dead! " << endl;

cout << " Knight: gains 20 extra attack points! " << endl;

cout << " Zombie: returns from the dead! " << endl;

cout << " Ghost: gains 20 extra health points! " << endl;

cout << endl;

cout << " What character do you want to play as: (S)amurai, (K)night, (Z)ombie or (G)host: ";

char name;

cin >> name;

while (name != 'S' && name != 'K' && name != 'Z' && name != 'G')

{

cout << " That is an incorrect choice. Try again! ";

cin >> name;

}

if (name == 'S') {

character = new Samurai;

character->addCharacter();

return character;

}

if (name == 'K') {

character = new Knight;

character->addCharacter();

return character;

}

if (name == 'Z') {

character = new Zombie;

character->addCharacter();

return character;

}

if (name == 'G') {

character = new Ghost;

character->addCharacter();

return character;

}

}

vector<Character\*> randEnemies() {

vector<Character\*> enemy;

cout << " Enter the number of enemies you want to fight: ";

int n;

cin >> n;

for (int i = 0; i < n; i++)

{

int r = rand() % 4 + 1;

switch (r)

{

case 1:

enemy.push\_back(new Samurai);

break;

case 2:

enemy.push\_back(new Knight);

break;

case 3:

enemy.push\_back(new Zombie);

break;

case 4:

enemy.push\_back(new Ghost);

break;

default:

break;

}

enemy.back()->createRandEnemies(rand() % 10 + 1, rand() % 5 + 1, rand() % 10 + 1);

}

return enemy;

}

#pragma once

#ifndef VILLAIN\_H

#define VILLAIN\_H

#include <iostream>

#include <vector>

#include <string>

#include "Character.h"

using namespace std;

//Hierarchy Level 2

class Villain : public Character

{

public:

Villain() {};

void addCharacter();

};

void Villain::addCharacter() {

Character::addCharacter();

cout << " Enter attack points: (has to be between 25-30)";

cin >> attack;

while (attack > 30 || attack < 25)

{

cout << " Wrong input! Enter attack again: ";

cin >> attack;

}

cout << " Enter health points: (has to be between 50-100)";

cin >> health;

while (health > 100 || health < 50)

{

cout << " Wrong input! Enter health again: ";

cin >> health;

}

cout << " Enter desciption of character: ";

cin.ignore();

getline(cin, description);

}

#endif // !VILLAIN\_H

#pragma once

#ifndef HERO\_H

#define HERO\_H

#include <iostream>

#include <vector>

#include <string>

#include "Character.h"

using namespace std;

//Hierarchy Level 2

class Hero : public Character

{

public:

Hero() {};

void addCharacter();

};

void Hero::addCharacter() {

Character::addCharacter();

cout << " Enter attack points: (has to be between 20-25)";

cin >> attack;

while (attack > 25 || attack < 20)

{

cout << " Wrong input! Enter attack again: ";

cin >> attack;

}

cout << " Enter health points: (has to be between 50-120)";

cin >> health;

while (health > 120 || health < 50)

{

cout << " Wrong input! Enter health again: ";

cin >> health;

}

cout << " Enter desciption of character: ";

cin.ignore();

getline(cin, description);

}

#endif // !HERO\_H

#pragma once

#ifndef CHARACTER\_H

#define CHARACTER\_H

#include <iostream>

#include <vector>

#include <string>

using namespace std;

//Hierarchy Level 1

class Character

{

public:

Character()

: name(""), attack(0), health(0), description(""), death(false) {};

//pure virtual function specialty

virtual void setSpecialty() = 0;

virtual void zeroSpecialty() = 0;

virtual int getSpecialty() = 0;

//virtual functions

virtual void addCharacter();

virtual void createRandEnemies(int rName, int rAttack, int rHealth);

//getter functions

string getName() { return name; }

int getHealth() { return health; }

int getAttack() { return attack; }

bool getDeath() { return death; }

//other

bool fight(vector<Character\*> enemies, Character\* character);

protected:

string name;

int attack;

int health;

string description;

bool death;

};

void Character::addCharacter() {

cout << " Please enter the name of your Character: ";

cin.ignore();

getline(cin, name);

}

void Character::createRandEnemies(int rName, int rAttack, int rHealth) {

switch (rName)

{

case 1: name = "Jack";

break;

case 2: name = "Sparrow";

break;

case 3: name = "Scorpion";

break;

case 4: name = "Kate";

break;

case 5: name = "Sub-Zero";

break;

case 6: name = "Cyclops";

break;

case 7: name = "Ghost";

break;

case 8: name = "Zombie";

break;

case 9: name = "Knight";

break;

case 10: name = "Samurai";

break;

default:

break;

}

attack = rAttack;

health = rHealth;

}

bool Character::fight(vector<Character\*> enemies, Character\* character) { //if true you lost fight; if false you won fight and beat enemies

cout << "\* For every turn that occurs you can either attack your opponent or \*" << endl;

cout << "\* you can choose to defend, in both cases you opponent attacks you, \*" << endl;

cout << "\* but if you choose to defend, your opponents' attack inflicts less damage! \*" << endl;

cout << endl;

cout << " You can either choose to (a)ttack or (d)efend: " << endl;

char read;

int i = 0;//increments through the enemies

int round = 1;//keep track of rounds

int count = enemies.size();//keeps count of the numbers of enemies you are fighting

int tempHealth = character->health;//temp variable to hold health if character specialty is return from the dead

character->setSpecialty();//set the specialty

if (character->getSpecialty() == 2) { character->attack += 20; }//if specialty is 2, 20 more attack points are added

if (character->getSpecialty() == 3) { character->health += 20; }//if specialty is 3, 20 more health points are added

while (!character->getDeath() && i < enemies.size())

{

cout << endl;

cout << " ROUND " << round << ". FIGHT! " << endl;

cout << endl;

cin >> read;

if (read == 'a')

{

enemies.at(i)->health -= character->attack;

character->health -= enemies.at(i)->attack;

if (enemies.at(i)->health <= 0) { i++; count--; }//increment next enemy through vector and decrement the total number of the enemies

if (character->health <= 0) {

character->death = true;

if (character->getSpecialty() == 1) {//if specialty is 1, return from the dead

character->death = false;

character->health = tempHealth;

character->zeroSpecialty();

}

}

cout << " Your health left: " << character->getHealth() << " Your attack power: " << character->getAttack() << " Opponenets left: " << count << endl;

}

if (read == 'd')

{

int temp = enemies.at(i)->attack;//some logic for the defending part of the game

temp -= 2;

character->health -= temp;

if (enemies.at(i)->health <= 0) { i++; count--; }//increment next enemy through vector and decrement the total number of the enemies

if (character->health <= 0)

{

character->death = true;

if (character->getSpecialty() == 1) {//if specialty is 1, return from the dead

character->death = false;

character->health = tempHealth;

character->zeroSpecialty();

}

}

cout << " Your health left: " << character->getHealth() << " Your attack power: " << character->getAttack() << " Opponenets left: " << count << endl;

}

round++;//increment round

}//some logic to continue the fighting loop

if (character->getDeath()) { return true; } //character was killed!

if (character->getHealth() > 0) { return false; } // you killed wave of enemies!

return 0;

}

#endif // !CHARACTER\_h

#pragma once

#ifndef QUICKSORT\_H

#define QUICKSORT\_H

#include <iostream>

#include <vector>

#include <string>

#include "Character.h"

using namespace std;

void quickSort(vector<Character\*>& enemies, int low, int high, int (Character::\*get)());//using a pointer to function

void swap(Character\*& a, Character\*& b);

void sort(vector<Character\*>& enemies) {

int read;

cout << " Sort by: " << endl;

cout << " (1). Attack Points " << endl;

cout << " (2). Health Points " << endl;

cin >> read;

while (read != 1 && read != 2) {

cout << " Oops wrong input try again: ";

cin >> read;

}

if (read == 1)

{

quickSort(enemies, 0, enemies.size() - 1, &Character::getAttack);//using a pointer to function

}

if (read == 2)

{

quickSort(enemies, 0, enemies.size() - 1, &Character::getHealth);//using a pointer to function

}

}

void insertionSort(vector<Character\*>& enemies, int low, int high, int (Character::\*get)()) {

int j;

for (int p = low; p < high + 1; p++) {

Character\* tmp = enemies.at(p);

for (j = p; j > 0 && (tmp->\*get)() < (enemies.at(j - 1)->\*get)(); j--)

enemies.at(j) = enemies.at(j - 1);

enemies.at(j) = tmp;

}

}

void quickSort(vector<Character\*>& enemies, int low, int high, int (Character::\*get)()) {

if (low + 10 > high) insertionSort(enemies, low, high, get); // call insertion sort if there are less than 10 enemies

else {

int middle = (low + high) / 2;

if ((enemies.at(middle)->\*get)() < (enemies.at(low)->\*get)()) swap(enemies.at(low), enemies.at(middle));

if ((enemies.at(high)->\*get)() < (enemies.at(low)->\*get)()) swap(enemies.at(low), enemies.at(high));

if ((enemies.at(high)->\*get)() < (enemies.at(middle)->\*get)())swap(enemies.at(middle), enemies.at(high));

int pivot = (enemies.at(middle)->\*get)();

swap(enemies.at(middle), enemies.at(high - 1));

int i, j;

for (i = low, j = high - 1;;) { //distribute the elements with regard to the pivot

while ((enemies.at(++i)->\*get)() < pivot) {}

while (pivot < (enemies.at(--j)->\*get)()) {}

if (i < j) swap(enemies.at(i), enemies.at(j));

else break;

}

swap(enemies.at(i), enemies.at(high - 1)); // put the pivot in its final place

quickSort(enemies, low, i - 1, get); // sort the part left from the pivot

quickSort(enemies, i + 1, high, get); // sort the part right from the pivot

}

}

void swap(Character\*& a, Character\*& b) {

Character\* tmp = a;

a = b;

b = tmp;

}

#endif // !QUICKSORT\_H