**Лабораторна робота №3**

**З дисципліни «Алгоритми та структури даних»**

***«Дослідження структур даних приорітетна черга та купа»***

*Виконав  
студент I курсу  
групи ДА-12  
Ніколаєв Роман*

Мета роботи

Ознайомитись і дослідити структуру даних бінарна купа. Набути навичок реалізації абстрактної структури даних черга з пріоритетом на основі бінарної купи мовою програмування C++. Ознайомитись з механізмом перевантаження операторів та дослідити особливості сортування купою (пірамідального сортування). Порівняти власну реалізацію пріоритетної черги з готовим бібліотечним рішенням STL.

*Задача 2*

**Хід виконання завдання:**

Перед ботом з’являються вороги в зоні видимості. Кожен ворог характеризується рівнем здоров’я, можливим уроном, скілом гравця тощо. Бот за певним критерієм повинен обрати найбільш оптимальну для себе ціль та знищити її, після цього перейти до наступної цілі. Потрібно врахувати, що впродовж гри можуть з’являтися нові вороги з більш високим пріоритетом.

**Файл “main.cpp”:**

#include <iostream>

#include <ctime>

#include <queue>

#include <cstdlib>

#include "PriorityQueue.h"

using namespace std;

template <typename T>

float testPriorityQueueSpeed(T&& priorityQueue)

{

const int iters = 100000;

clock\_t timeStart = clock();

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

{

int insertDataAmount = rand() % 6 + 5;

for (int j = 0; j < insertDataAmount; j++)

{

priorityQueue.push(Data());

}

priorityQueue.top();

priorityQueue.pop();

}

clock\_t timeEnd = clock();

float time = (float(timeEnd - timeStart)) / CLOCKS\_PER\_SEC;

return time;

}

bool testPriorityQueue()

{

srand(time(NULL));

const int iters = 20000;

PriorityQueue myPriorQueue;

priority\_queue<Data> stlPriorQueue;

bool isDataEqual = true;

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

{

int insertDataAmount = rand() % 6 + 5;

for (int j = 0; j < insertDataAmount; j++)

{

Data randData = Data();

myPriorQueue.push(randData);

stlPriorQueue.push(randData);

}

if (!(myPriorQueue.top() == stlPriorQueue.top()))

{

isDataEqual = false;

if (myPriorQueue.heapSize == 0)

cout << "nu yasno";

cerr << "Comparing failed on iteration " << i << endl << endl;

break;

}

int removeDataAmount = rand() % insertDataAmount;

for (int j = 0; j < removeDataAmount; j++)

{

myPriorQueue.pop();

stlPriorQueue.pop();

}

}

int myQueueSize = myPriorQueue.size();

int stlQueueSize = stlPriorQueue.size();

float stlTime = testPriorityQueueSpeed<priority\_queue<Data>>(priority\_queue<Data>());

float myTime = testPriorityQueueSpeed<PriorityQueue>(PriorityQueue());

cout << "My PriorityQueue:" << endl;

cout << "Time: " << myTime << ", size: " << myQueueSize << endl;

cout << "STL priority\_queue:" << endl;

cout << "Time: " << stlTime << ", size: " << stlQueueSize << endl << endl;

if (isDataEqual && myQueueSize == stlQueueSize)

{

cout << "The lab is completed" << endl << endl;

return true;

}

cerr << ":(" << endl << endl;

return false;

}

int main()

{

testPriorityQueue();

return 0;

}

**Файл “PriorityQueue.h”:**

#pragma once

#include <iostream>

#include <vector>

#include <ctime>

struct Data{

int hp, damage, resistance;

Data(){

hp = rand()%101;

damage = rand()%501;

resistance = rand()%100;

};

Data(int code){

hp = -1;

damage = -1;

resistance = -1;

}

bool operator<(const Data& data2) const{

if(hp != data2.hp){

return hp < data2.hp;

} else if(damage != data2.damage){

return damage < data2.damage;

}

return resistance < data2.resistance;

}

bool operator==(const Data& data2) const{

return hp == data2.hp && damage == data2.damage && resistance == data2.resistance;

}

};

struct PriorityQueue

{

int capacity;

Data\* enemy;

int heapSize;

const int alpha = 2;

PriorityQueue()

{

capacity = 16;

heapSize = 0;

enemy = new Data[capacity];

}

int parent(int i)

{

if(i > 0)

return (i - 1) / 2;

return 0;

}

int left(int i)

{

return i \* 2 + 1;

}

int right(int i)

{

return i \* 2 + 2;

}

void siftDown(int i)

{

while (left(i) < heapSize) {

int j = left(i);

if ((right(i) < heapSize) && (enemy[j] < enemy[right(i)]))

j = right(i);

if (enemy[j] < enemy[i] || enemy[j] == enemy[i])

break;

std::swap(enemy[i], enemy[j]);

i = j;

}

}

void siftUp(int i)

{

while (enemy[parent(i)] < enemy[i]) {

std::swap(enemy[i], enemy[parent(i)]);

i = parent(i);

}

}

void push(Data new\_record)

{

if (heapSize == capacity) {

capacity \*= alpha;

Data\* new\_array = new Data[capacity];

for (int j = 0; j < heapSize; j++)

new\_array[j] = enemy[j];

delete[] enemy;

enemy = new\_array;

}

heapSize++;

enemy[heapSize - 1] = new\_record;

siftUp(heapSize - 1);

}

Data top()

{

if (heapSize > 0)

return enemy[0];

Data empty(-1);

return empty;

}

Data pop()

{

if (heapSize > 0) {

std::swap(enemy[0], enemy[heapSize - 1]);

heapSize--;

siftDown(0);

return enemy[heapSize];

}

Data empty(-1);

return empty;

}

int size()

{

return heapSize;

}

~PriorityQueue()

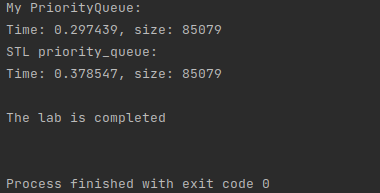
{

delete[] enemy;

}

};

**Результат виконання:**



**Висновки:**

*Виконуючи дану роботу було набуто навичок використання купи та приорітетної черги. В поцесі виконання з’являлося розуміння, на прикоадах з власного оточення, де і чому саме використовується вивчена струтура даних. Наприклад — вже розглянутий випадок пріорітетних цілей в іграх, або рейтингова таблиця в тих самих іграх, котра використвує декілька параметрів для визначення порядку черги.*