**ТЕХНИЧЕСКИ УНИВЕРСИТЕТ - СОФИЯ**

**ФИЛИАЛ ПЛОВДИВ**

**ФАКУЛТЕТ ПО ЕЛЕКТРОНИКА И АВТОМАТИКА**

**Катедра “Компютърни системи”**

**Курсов проект по ООП**

Иготвил: Георги Пламенов Копчев

Група: 41 „а“

Специалност: КСТ

Фак.номер: 365163

Дата:

Идея

За курсов проект реших да имплементирам така наречения fighting engine на една средностатистическа fight игра. За целта има 3 раси - европейска, азиатска и арабска, които воюват помежду си. Има и 3 вида монументи, които ще носят определен бонус на определена раса. Имплементирани са 5 команди:

Civilization

Parameters – type (string), name (string), power (int), secondaryParameter (floating**-**pointnumber)

Създава цивилизация от даден тип, с дадени параметри.

Поддържани type-ове: European, Asian, Arab.

SecondaryParameter ще се присвоява на някоя от следващите, според типа:

* Warriors power
* Ronins power
* Cavalery power

Monument

Parameters –type(string),name(string),affinity(int).

Създава монумент от даден тип, с даден параметър.

Поддържани type-ове: EuropeanMonument, AsianMonument, ArabMonument.

Status коменада

Принтира информация за дадената цивилизация и всичките и принадлежащи монументи.

War

Parameters – civilization(string).

Поддържани type-ове: European, Asian, Arab.

Дадената цивилизация предизвиква война, която включва всички други нации. Крайната сила на всяка нация се пресмята, като се събере силата на всичките и воини и се добавят бонусите за всяка една от нациите. Когато всичко е сметнато, нацията с най-голяма сила съответно печели. Губещите нации губят всичките си монументи, които им носят бонуси. Ходът на играта продължава, като в загубилите нации могат, разбира се, отново да се добавят армия и съотвено монументи и отново да получават бонус за бъдещи воини.

Quit

Програмата приключва изчакването на потребителски вход и играта спира, като между тези 2 събития се принтира на конзолата всяка възникнала война и нацията, която я е предизвикала, в последователен ред.

GameManager.h

#pragma once

#include "Civilization.h"

#include "Monument.h"

#include "Constants.h"

#include "CivilizationFactory.h"

#include "MonumentFactory.h"

#include "IntToStringConverter.h"

class GameManager

{

private:

string EUROPEAN\_STRING = "European";

string ARAB\_STRING = "Arab";

string WATER\_STRING = "Water";

string ASIAN\_STRING = "Asian";

string CIVILIZATION\_COMMAND = "Civilization";

string MONUMENT\_COMMAND = "Monument";

string STATUS\_COMMAND = "Status";

string WAR\_COMMAND = "War";

string INPUT\_TERMINATING\_COMMAND = "Quit";

std::map<string, std::list<Monument\*>> monuments;

std::map<string, std::list<Civilization\*>> Civilizations;

std::list<string> wars;

void initializeMonuments();

void initializeCivilizations();

string appendCivilizations(string builder, string nation);

string appendMonuments(string builder, string nation);

double getCurrentTotalPower(list<Civilization\*> value);

double getCurrentMonumentsBonus(string nation, double totalPower);

void killLoserNations(string winningNation);

void clearLoserNationsMonuments(string winningNation);

public:

string createCivilization(string type, string name, int power, double param);

string createMonument(string type, string name, int affinity);

string printStatus(string nation);

string startWar(string nation);

string warsList();

string quit();

GameManager();

~GameManager();

};

GameManager.cpp

#include "stdafx.h"

#include "GameManager.h"

#include "EuropeanCivilization.h"

void GameManager::initializeMonuments()

{

list<Monument\*> l;

this->monuments.insert(make\_pEuropean(this->EUROPEAN\_STRING, l));

this->monuments.insert(make\_pEuropean(this->WATER\_STRING, l));

this->monuments.insert(make\_pEuropean(this->ARAB\_STRING, l));

this->monuments.insert(make\_pEuropean(this->ASIAN\_STRING, l));

}

void GameManager::initializeCivilizations()

{

list<Civilization\*> l;

this->Civilizations.insert(make\_pEuropean(this->EUROPEAN\_STRING, l));

this->Civilizations.insert(make\_pEuropean(this->WATER\_STRING, l));

this->Civilizations.insert(make\_pEuropean(this->ARAB\_STRING, l));

this->Civilizations.insert(make\_pEuropean(this->ASIAN\_STRING, l));

}

string GameManager::appendCivilizations(string builder, string nation)

{

list<Civilization\*> CivilizationsFound = this->Civilizations.find(nation)->second;

if (CivilizationsFound.size() > 0) {

for (Civilization\* Civilization : CivilizationsFound) {

builder.append("\n----").append(Civilization->getName());

}

}

else {

builder.append(" None");

}

return builder;

}

string GameManager::appendMonuments(string builder, string nation)

{

list<Monument\*> CivilizationsFound = this->monuments.find(nation)->second;

if (CivilizationsFound.size() > 0) {

for (Monument\* monument : CivilizationsFound) {

builder.append("\n----").append(monument->getName());

}

}

else {

builder.append(" None");

}

return builder;

}

string GameManager::createCivilization(string type, string name, int power, double param)

{

Civilization\* b = CivilizationFactory::generateCivilization(type, name, power, param);

this->Civilizations.find(type)->second.push\_back(b);

return "Successfully created Civilization: " + name;

}

string GameManager::createMonument(string type, string name, int affinity)

{

Monument\* m = MonumentFactory::generateMonument(type, name, affinity);

this->monuments.find(type)->second.push\_back(m);

return "Successfully created Monument: " + name;

}

string GameManager::printStatus(string nation)

{

string builder = string();

builder.append(nation).append(" Nation\n");

builder.append("--Civilizations:");

builder = this->appendCivilizations(builder, nation);

builder.append("\n--Monuments:");

builder = this->appendMonuments(builder, nation);

return builder;

}

double GameManager::getCurrentTotalPower(list<Civilization\*> value) {

double result = 0;

for (Civilization\* Civilization : value) {

result += Civilization->calculateTotalPower();

}

return result;

}

double GameManager::getCurrentMonumentsBonus(string nation, double totalPower) {

double affinity = 0;

for (Monument\* monument : this->monuments.find(nation)->second) {

affinity += monument->getAffinity();

}

return (totalPower / 100) \* affinity;

}

void GameManager::killLoserNations(string winningNation)

{

for (std::map<string, list<Civilization\*>>::iterator it = this->Civilizations.begin(); it != this->Civilizations.end(); ++it) {

if (it->first.compare(winningNation) != 0) {

it->second.clear();

}

}

}

void GameManager::clearLoserNationsMonuments(string winningNation)

{

for (std::map<string, list<Monument\*>>::iterator it = this->monuments.begin(); it != this->monuments.end(); ++it) {

if (it->first.compare(winningNation) != 0) {

it->second.clear();

}

}

}

string GameManager::startWar(string nation)

{

this->wars.push\_back(nation);

double bestPower = 0;

string winningNation = nation;

for (std::map<string, list<Civilization\*>>::iterator it = this->Civilizations.begin(); it != this->Civilizations.end(); ++it) {

double currentTotalPower = getCurrentTotalPower(it->second);

double currentMonumentsBonus = getCurrentMonumentsBonus(it->first, currentTotalPower);

currentTotalPower += currentMonumentsBonus;

if (currentTotalPower > bestPower) {

bestPower = currentTotalPower;

winningNation = it->first;

}

}

//all of other nations are dead

this->killLoserNations(winningNation);

//all of other nations monuments are cleared

this->clearLoserNationsMonuments(winningNation);

return "Winner nation: " + winningNation + " with total power: " + IntToStringConverter::convert(bestPower);

}

string GameManager::warsList()

{

string builder = string();

int counter = 1;

for (string war : this->wars) {

builder.append("War " + IntToStringConverter::convert(counter) + ", invoked by: ").append(war).append(" nation.").append("\n");

counter++;

}

builder = builder.substr(0, builder.size() - 1);

return builder;

}

string GameManager::quit()

{

return this->warsList();

}

GameManager::GameManager()

{

this->initializeMonuments();

this->initializeCivilizations();

}

GameManager::~GameManager()

{

}

Engine.h

#pragma once

#include "InputParser.h"

#include "GameManager.h"

#include "ConsoleInputReader.h"

#include "ConsoleOutputWriter.h"

#include "Constants.h"

class Engine

{

private:

GameManager manager;

Constants constants = Constants();

InputParser parser;

ConsoleInputReader inputReader;

ConsoleOutputWriter outputWriter;

void dispatchCommand(std::list<string> commandParams);

public:

void run();

Engine(ConsoleInputReader inputReader, ConsoleOutputWriter outputWriter, InputParser inputParser, GameManager manager);

~Engine();

};

Engine.cpp

#include "stdafx.h"

#include "Engine.h"

Engine::Engine(ConsoleInputReader inputReader, ConsoleOutputWriter outputWriter, InputParser inputParser, GameManager manager)

{

this->manager = manager;

this->inputReader = inputReader;

this->outputWriter = outputWriter;

this->parser = inputParser;

}

Engine::~Engine()

{

}

void Engine::run()

{

string inputLine;

while ((inputLine = this->inputReader.readLine()).compare(this->constants.INPUT\_TERMINATING\_COMMAND) != 0) {

std::list<string> commandParams = this->parser.parseInput(inputLine);

this->dispatchCommand(commandParams);

}

this->outputWriter.writeLine(manager.quit());

}

void Engine::dispatchCommand(std::list<string> commandParams)

{

std::list<std::string>::iterator it = commandParams.begin();

string command = \*it;

std::advance(it, 1);

string output;

if (command.compare(this->constants.CIVILIZATION\_COMMAND) == 0) {

string type = \*it;

std::advance(it, 1);

string name = \*it;

std::advance(it, 1);

int power = std::stoi(\*it);

std::advance(it, 1);

double param = std::stod(\*it);

output = manager.createCivilization(type, name, power, param);

}

else if (command.compare(this->constants.MONUMENT\_COMMAND) == 0) {

string type = \*it;

std::advance(it, 1);

string name = \*it;

std::advance(it, 1);

int affinity = std::stoi(\*it);

output = manager.createMonument(type, name, affinity);

}

else if (command.compare(this->constants.STATUS\_COMMAND) == 0) {

string nation = \*it;

output = manager.printStatus(nation);

}

else if (command.compare(this->constants.WAR\_COMMAND) == 0) {

string nation = \*it;

output = manager.startWar(nation);

}

else if (command.compare(this->constants.WARS\_LIST\_COMMAND) == 0) {

output = manager.warsList();

}

else {

output = "Invalid command, please try again.";

}

this->outputWriter.writeLine(output);

}

CivilizationFactory.h

#pragma once

#include "Civilization.h"

#include "EuropeanCivilization.h"

#include "WaterCivilization.h"

#include "AsianCivilization.h"

#include "ArabCivilization.h"

class CivilizationFactory

{

private:

public:

static Civilization\* generateCivilization(string type, string name, int power, double parameter);

CivilizationFactory();

~CivilizationFactory();

};

CivilizationFactory.cpp

#include "stdafx.h"

#include "CivilizationFactory.h"

Civilization\* CivilizationFactory::generateCivilization(string type, string name, int power, double parameter)

{

string EUROPEAN\_STRING = "European";

string ARAB\_STRING = "Arab";

string WATER\_STRING = "Water";

string ASIAN\_STRING = "Asian";

if (type.compare(EUROPEAN\_STRING) == 0) {

return new EuropeanCivilization(name, power, parameter);

}

else if (type.compare(WATER\_STRING) == 0) {

return new WaterCivilization(name, power, parameter);

}

else if (type.compare(ARAB\_STRING) == 0) {

return new ArabCivilization(name, power, parameter);

}

else {

return new AsianCivilization(name, power, parameter);

}

}

CivilizationFactory::CivilizationFactory()

{

}

CivilizationFactory::~CivilizationFactory()

{

}

MonumentFactory.h

#pragma once

#include "Monument.h"

#include "EuropeanMonument.h"

#include "WaterMonument.h"

#include "ArabMonument.h"

#include "AsianMonument.h"

class MonumentFactory

{

private:

public:

static Monument\* generateMonument(string type, string name, int affinity);

MonumentFactory();

~MonumentFactory();

};

MonumentFactory.cpp

#include "stdafx.h"

#include "MonumentFactory.h"

Monument\* MonumentFactory::generateMonument(string type, string name, int affinity)

{

string EUROPEAN\_STRING = "European";

string ARAB\_STRING = "Arab";

string WATER\_STRING = "Water";

string ASIAN\_STRING = "Asian";

if (type.compare(EUROPEAN\_STRING) == 0) {

return new EuropeanMonument(name, affinity);

}

else if (type.compare(WATER\_STRING) == 0) {

return new WaterMonument(name, affinity);

}

else if (type.compare(ARAB\_STRING) == 0) {

return new ArabMonument(name, affinity);

}

else {

return new AsianMonument(name, affinity);

}

}

MonumentFactory::MonumentFactory()

{

}

MonumentFactory::~MonumentFactory()

{

}

ConsoleInputReader.h

#pragma once

class ConsoleInputReader

{

public:

string readLine();

ConsoleInputReader();

~ConsoleInputReader();

};

ConsoleInputReader.cpp

#include "stdafx.h"

#include "ConsoleInputReader.h"

string ConsoleInputReader::readLine()

{

std::string s;

std::getline(std::cin, s);

return s;

}

ConsoleInputReader::ConsoleInputReader()

{

}

ConsoleInputReader::~ConsoleInputReader()

{

}

ConsoleOutputWriter.h

#pragma once

class ConsoleOutputWriter

{

public:

void writeLine(string output);

ConsoleOutputWriter();

~ConsoleOutputWriter();

};

ConsoleOutputWriter.cpp

#include "stdafx.h"

#include "ConsoleOutputWriter.h"

void ConsoleOutputWriter::writeLine(string output)

{

cout << output << endl;

}

ConsoleOutputWriter::ConsoleOutputWriter()

{

}

ConsoleOutputWriter::~ConsoleOutputWriter()

{

}

Civilization.h

#pragma once

class Civilization

{

private:

string name;

int power;

void setName(string name);

void setPower(int power);

public:

virtual double calculateTotalPower() = 0;

int getPower();

string getName();

Civilization(string name, int power);

~Civilization();

};

Civilization.cpp

#include "stdafx.h"

#include "Civilization.h"

void Civilization::setName(string name)

{

this->name = name;

}

void Civilization::setPower(int power)

{

this->power = power;

}

int Civilization::getPower()

{

return this->power;

}

string Civilization::getName()

{

return this->name;

}

Civilization::Civilization(string name, int power)

{

this->setName(name);

this->setPower(power);

}

Civilization::~Civilization()

{

}

EuropeanCivilization.h

#pragma once

#include "Civilization.h"

class EuropeanCivilization : virtual public Civilization

{

private:

double warriorsPower;

void setWarriorsPower(double warriorsPower);

public:

double calculateTotalPower();

EuropeanCivilization(string name, int power, double warriorsPower);

~EuropeanCivilization();

};

EuropeanCivilization.cpp

#include "stdafx.h"

#include "EuropeanCivilization.h"

void EuropeanCivilization::setWarriorsPower(double warriorsPower)

{

this->warriorsPower = warriorsPower;

}

double EuropeanCivilization::calculateTotalPower()

{

return Civilization::getPower() \* this->warriorsPower;

}

EuropeanCivilization::EuropeanCivilization(string name, int power, double warriorsPower) : Civilization(name, power)

{

this->setWarriorsPower(warriorsPower);

}

EuropeanCivilization::~EuropeanCivilization()

{

}

AsianCivilization.h

#pragma once

#include "Civilization.h"

class AsianCivilization : virtual public Civilization

{

private:

double roninsPower;

double getRoninsPower();

void setRoninsPower(double AsianClarity);

public:

double calculateTotalPower();

AsianCivilization(string name, int power, double roninsPower);

~AsianCivilization();

};

AsianCivilization.cpp

#include "stdafx.h"

#include "AsianCivilization.h"

double AsianCivilization::getRoninsPower()

{

return this->roninsPower;

}

void AsianCivilization::setRoninsPower(double roninsPower)

{

this->roninsPower = roninsPower;

}

double AsianCivilization::calculateTotalPower()

{

return Civilization::getPower() \* this->getRoninsPower();

}

AsianCivilization::AsianCivilization(string name, int power, double roninsPower) : Civilization(name, power)

{

this->setRoninsPower(roninsPower);

}

AsianCivilization::~AsianCivilization()

{

}

ArabCivilization.h

#pragma once

#include "Civilization.h"

class ArabCivilization : virtual public Civilization

{

private:

double cavalryPower;

void setCavalryPower(double warriorsPower);

public:

double calculateTotalPower();

ArabCivilization(string name, int power, double warriorsPower);

~ArabCivilization();

};

ArabCivilization.cpp

#include "stdafx.h"

#include "ArabCivilization.h"

ArabCivilization::ArabCivilization(string name, int power, double cavalryPower) : Civilization(name, power)

{

this->setCavalryPower(cavalryPower);

}

void ArabCivilization::setCavalryPower(double cavalryPower)

{

this->cavalryPower = cavalryPower;

}

double ArabCivilization::calculateTotalPower()

{

return Civilization::getPower() \* this->cavalryPower;

}

ArabCivilization::~ArabCivilization()

{

}

Monument.h

#pragma once

class Monument

{

private:

string name;

void setName(string name);

public:

virtual int getAffinity() = 0;

string getName();

Monument(string name);

~Monument();

};

Monument.cpp

#include "stdafx.h"

#include "Monument.h"

string Monument::getName()

{

return this->name;

}

Monument::Monument(string name)

{

this->setName(name);

}

void Monument::setName(string name)

{

this->name = name;

}

Monument::~Monument()

{

}

EuropeanMonument.h

#pragma once

#include "Monument.h"

class EuropeanMonument : public Monument

{

private:

int CathedralAffinity;

void setCathedralAffinity(int CathedralAffinity);

public:

int getAffinity();

EuropeanMonument(string name, int CathedralAffinity);

~EuropeanMonument();

};

EuropeanMonument.cpp

#include "stdafx.h"

#include "EuropeanMonument.h"

void EuropeanMonument::setCathedralAffinity(int CathedralAffinity)

{

this->CathedralAffinity = CathedralAffinity;

}

int EuropeanMonument::getAffinity()

{

return this->CathedralAffinity;

}

EuropeanMonument::EuropeanMonument(string name, int CathedralAffinity) : Monument(name)

{

this->setCathedralAffinity(CathedralAffinity);

}

EuropeanMonument::~EuropeanMonument()

{

}

AsianMonument.h

#pragma once

#include "Monument.h"

class AsianMonument : public Monument

{

private:

int ShaolinAffinity;

void setShaolinAffinity(int CathedralAffinity);

public:

int getAffinity();

AsianMonument(string name, int CathedralAffinity);

~AsianMonument();

};

AsianMonument.cpp

#include "stdafx.h"

#include "AsianMonument.h"

void AsianMonument::setShaolinAffinity(int ShaolinAffinity)

{

this->ShaolinAffinity = ShaolinAffinity;

}

int AsianMonument::getAffinity()

{

return this->ShaolinAffinity;

}

AsianMonument::AsianMonument(string name, int ShaolinAffinity) : Monument(name)

{

this->setShaolinAffinity(ShaolinAffinity);

}

AsianMonument::~AsianMonument()

{

}

ArabMonument.h

#pragma once

#include "Monument.h"

class ArabMonument : public Monument

{

private:

int ChurchAffinity;

void setChurchAffinity(int CathedralAffinity);

public:

int getAffinity();

ArabMonument(string name, int CathedralAffinity);

~ArabMonument();

};

ArabMonument.cpp

#include "stdafx.h"

#include "ArabMonument.h"

void ArabMonument::setChurchAffinity(int ChurchAffinity)

{

this->ChurchAffinity = ChurchAffinity;

}

int ArabMonument::getAffinity()

{

return this->ChurchAffinity;

}

ArabMonument::ArabMonument(string name, int ChurchAffinity) : Monument(name)

{

this->setChurchAffinity(ChurchAffinity);

}

ArabMonument::~ArabMonument()

{

}

Constants.h

#pragma once

class Constants

{

public:

const string EUROPEAN\_STRING = "European";

const string ARAB\_STRING = "Arab";

const string WATER\_STRING = "Water";

const string ASIAN\_STRING = "Asian";

const string CIVILIZATION\_COMMAND = "Civilization";

const string MONUMENT\_COMMAND = "Monument";

const string STATUS\_COMMAND = "Status";

const string WAR\_COMMAND = "War";

const string WARS\_LIST\_COMMAND = "WarsList";

const string INPUT\_TERMINATING\_COMMAND = "Quit";

Constants();

~Constants();

};

Cosntants.cpp

#include "stdafx.h"

#include "Constants.h"

Constants::Constants()

{

}

Constants::~Constants()

{

}

InputParser.h

#pragma once

class InputParser

{

public:

std::list<string> parseInput(string inputLine);

InputParser();

~InputParser();

};

InputParser.cpp

#include "stdafx.h"

#include "InputParser.h"

std::list<string> InputParser::parseInput(string inputLine)

{

std::string delimiter = " ";

std::list<string> splitted;

size\_t pos = 0;

std::string token;

while ((pos = inputLine.find(delimiter)) != std::string::npos) {

token = inputLine.substr(0, pos);

splitted.push\_back(token);

inputLine.erase(0, pos + delimiter.length());

}

splitted.push\_back(inputLine);

return splitted;

}

InputParser::InputParser()

{

}

InputParser::~InputParser()

{

}

IntToStringConverter.h

#pragma once

class IntToStringConverter

{

public:

static string convert(int number);

IntToStringConverter();

~IntToStringConverter();

};

IntToStringConverter.cpp

#include "stdafx.h"

#include "IntToStringConverter.h"

string IntToStringConverter::convert(int a)

{

ostringstream temp;

temp << a;

return temp.str();

}

IntToStringConverter::IntToStringConverter()

{

}

IntToStringConverter::~IntToStringConverter()

{

}

Tzar.cpp

// Tzar.cpp : Defines the entry point for the console application.

//

#include "stdafx.h"

#include "ConsoleInputReader.h"

#include "ConsoleOutputWriter.h"

#include "InputParser.h"

#include "GameManager.h"

#include "Engine.h"

void main()

{

ConsoleInputReader inputReader = ConsoleInputReader();

ConsoleOutputWriter outputWriter = ConsoleOutputWriter();

InputParser inputParser = InputParser();

GameManager manager = GameManager();

Engine engine = Engine(inputReader, outputWriter, inputParser, manager);

engine.run();

}

Stdafx.h

// stdafx.h : include file for standard system include files,

// or project specific include files that are used frequently, but

// are changed infrequently

//

#pragma once

#include "targetver.h"

#include <stdio.h>

#include <tchar.h>

//mine

#include <iostream>

using namespace std;

#include <map>

#include <list>

#include <sstream>

Stdafx.cpp

// stdafx.cpp : source file that includes just the standard includes

// Avatar.pch will be the pre-compiled header

// stdafx.obj will contain the pre-compiled type information

#include "stdafx.h"

// TODO: reference any additional headers you need in STDAFX.H

// and not in this file

// TODO: reference additional headers your program requires here

Targetver.h

#pragma once

// Including SDKDDKVer.h defines the highest available Windows platform.

// If you wish to build your application for a previous Windows platform, include WinSDKVer.h and

// set the \_WIN32\_WINNT macro to the platform you wish to support before including SDKDDKVer.h.

#include <SDKDDKVer.h>