ОТЧЕТ ПО ЛАБОРАТОРНОЙ РАБОТЕ №3

Выполнил студент: Залесовский В.А.
группа Э-23

Наследование. Перегрузка

Цель работы: изучить процессы наследования в объектно-ориентированном программировании, а также перегрузку функций и операторов.

Индивидуальное задание:

Наследовать от касса новый класс: дополнение игры. Осуществить перегрузку операторов == для другого объекта класса справа, сравнивающий уровни двух персонажей. А также перегрузить оператор ++ который добавляет игроку 1000 очков опыта.

Game.h

```
//! Класс Game.
/*!
Класс предназначен для реализации одной и более игр,
а именно проведение количества битв, необходимых для достижения
заданного уровня.
*/

#pragma once
#include <iostream>
#include <random>
class Game
{
public:
  //! Конструктор по умолчанию.
  Game();
  //! Установка параметров для новой игры.
  /*!
  \param lvlHero Уровень героя.
  \param totalPoints Количество очков.
  \param totalBattles Количество битв.
  \param expMonster Опыт за монстра.
  \param expHero Опыт героя.
  \param lvlUpHero Количество опыта, требуемого для поднятия уровня.
  \param factorUp Коэффициент повышения значения поля "lvlUpHero".
  */
  void newGame();

  //! Проведение одной игры.
  /*!
```

```
  \param maxLvl Значение максимального уровня.
  \brief Принимает значение максимального уровня.
  \param totalBattles Количество битв.
  \brief Счетчик количества проведенных битв.
  */
  void game(int maxLvl);
  //! Проведение одной битвы в игре.
  /*!
  \param lvlMonster Уровень монстра.
  \brief Принимает сгенерированный уровень монстра.
  \param expHero Опыт героя.
  \brief Принимает сумму опыта героя и опыта за монстра в
случае победы.
  \param totalPoints Количество очков.
  \brief Увеличивается на 1 в случае победы.
  */
  void battle();
  //! Перерасчет параметров, связанных с повышением уровня.
  /*!
  \param lvlHero Уровень героя.
  \brief Увеличивается на 1.
  \param expHero Опыт героя.
  \brief Уменьшается на значение lvlUpHero.
  \param  lvlUpHero  Количество  опыта,  требуемого  для
поднятия уровня.
  \brief Перерасчитывается в соответствии с factorUp.
  */
  void lvlUp();

  //! Геттеры для тестов.
  //! Метод получения значения поля lvlHero.
  int getLvlHero();
  //! Метод получения значения поля totalPoints.
  int getTotalPoints();
  //! Метод получения значения поля totalBattles.
  int getTotalBattles();
  //! Метод получения значения поля expMonster.
  int getExpMonster();
  //! Метод получения значения поля expHero.
  double getExpHero();
  //! Метод получения значения поля lvlUpHero.
  double getLvlUpHero();
  //! Метод получения значения поля factorUp.
  double getFactorUp();

  //! Сеттеры для тестов.
  //! Метод установки значения поля lvlHero.
```

```
  void setLvlHero(int x);
  //! Метод установки значения поля totalPoints.
  void setTotalPoints(int x);
  //! Метод установки значения поля totalBattles.
  void setTotalBattles(int x);
  //! Метод установки значения поля expMonster.
  void setExpMonster(int x);
  //! Метод установки значения поля expHero.
  void setExpHero(double x);
  //! Метод установки значения поля lvlUpHero.
  void setLvlUpHero(double x);
  //! Метод установки значения поля factorUp.
  void setFactorUp(double x);

private:
  //! Уровень героя.
  int lvlHero;
  //! Уровень монстра.
  int lvlMonster;
  //! Количество очков.
  int totalPoints;
  //! Количество битв.
  int totalBattles;
  //! Опыт за монстра.
  int expMonster;
  //! Опыт героя.
  double expHero;
  //! Количество опыта, требуемого для поднятия уровня.
  double lvlUpHero;
  //! Коэффициент повышения значения поля lvlUpHero.
  double factorUp;

  //! Вывод на дисплей результатов игры.
  /*!
  \param i Количество битв.
  \param totalPoints Количество очков.
  */
  void showResultGame(int i);
  //! Вывод на дисплей результатов боя.
  /*!
  \param lvlHero Уровень героя.
  \param expHero Опыт героя.
  \param lvlUpHero Количество опыта, требуемого для
поднятия уровня.
  */
  void showResultBattle();
};
```

Game.cpp

```
#include "Game.h"

void Game::game(int maxLvl)
{
  newGame();
  for (totalBattles = 0;; totalBattles++)
  {
    battle();
    showResultBattle();
    if (maxLvl == lvlHero)
    {
      showResultGame(totalBattles);
      break;
    }
  }
}

int Game::getLvlHero()
{
  return lvlHero;
}

int Game::getTotalPoints()
{
  return totalPoints;
}

int Game::getTotalBattles()
{
  return totalBattles;
}

int Game::getExpMonster()
{
  return expMonster;
}

double Game::getExpHero()
{
  return expHero;
}

double Game::getLvlUpHero()
{
```

```
  return lvlUpHero;
}

double Game::getFactorUp()
{
  return factorUp;
}

void Game::setLvlHero(int x)
{
  if (x > 0) lvlHero = x;
  else lvlHero = 1;
}

void Game::setTotalPoints(int x)
{
  if (x >= 0) totalPoints = x;
  else totalPoints = 0;
}

void Game::setTotalBattles(int x)
{
  if (x >= 0) totalBattles = x;
  else totalBattles = 0;
}

void Game::setExpMonster(int x)
{
  if (x > 0) expMonster = x;
  else expMonster = 10;
}

void Game::setExpHero(double x)
{
  if (x >= 0.0) expHero = x;
  else expHero = 0;
}

void Game::setLvlUpHero(double x)
{
  if (x > 0.0) lvlUpHero = x;
  else lvlUpHero = 100;
}

void Game::setFactorUp(double x)
{
  if (x > 1.0) factorUp = x;
```

```
  else factorUp = 1.2;
}

void Game::newGame()
{
  lvlHero = 1;
  totalPoints = 0;
  totalBattles = 0;
  expMonster = 10;
  expHero = 0;
  lvlUpHero = 100;
  factorUp = 1.2;
}

Game::Game()
{
  lvlHero = 1;
  totalPoints = 0;
  totalBattles = 0;
  expMonster = 10;
  expHero = 0;
  lvlUpHero = 100;
  factorUp = 1.2;
}

void Game::battle()
{

  int maxLvlMonster = lvlHero + 4, minLvlMonster = 0,
chanceWin, factorExp = 1, MIN = 0, MAX = 100, delay = 100;
  if (lvlHero > 4)
  {
    minLvlMonster = lvlHero - 4;
  }
  std::random_device rd;
  std::mt19937 generator(rd());
  std::uniform_int_distribution<int>
randomLvlMonster(minLvlMonster,             maxLvlMonster),
randomChanceWin(MIN, MAX);
  lvlMonster = randomLvlMonster(generator);
  if (lvlHero - 4 <= lvlMonster && lvlMonster <= lvlHero -
2)
  {
    chanceWin = 100;
    if (lvlMonster < lvlHero - 2)
    {
      factorExp = 0;
```

```
    }
  }
  if (lvlHero - 1 <= lvlMonster && lvlMonster <= lvlHero +
1)
  {
    chanceWin = 60;
  }
  if (lvlHero + 2 == lvlMonster)
  {
    chanceWin = 45;
  }
  if (lvlHero + 3 == lvlMonster)
  {
    chanceWin = 30;
  }
  if (lvlHero + 4 == lvlMonster)
  {
    chanceWin = 15;
  }
  if (randomChanceWin(generator) <= chanceWin)
  {
    expHero += expMonster * factorExp;
    totalPoints++;
  }
  lvlUp();
}

void Game::lvlUp()
{
  if (expHero >= lvlUpHero)
  {
    expHero -= lvlUpHero;
    lvlHero++;
    lvlUpHero *= factorUp;
  }
}

void Game::showResultGame(int i)
{
  std::cout << "Victory!" << std::endl
    << "Total battles : " << i + 1 << std::endl
    << "Total points : " << totalPoints << std::endl;
}

void Game::showResultBattle()
{
  std::cout << "Level hero : " << lvlHero << std::endl
```

```
    << "Exp hero : " << expHero << std::endl
    << "Exp for level up : " << lvlUpHero << std::endl
    << std::endl;
}
```

GameAddition.h

```
//! Класс GameAddition.
/*!
Класс предназначен для дополнения игры путем наследования от класса родителя.
Класс осуществляет: перегрузку оператора сравнения(по уровням персонажей);
перегрузку оператора инкремента(по количеству опыта).
*/
#pragma once
#include "Game.h"
class GameAddition :public Game
{
public:
  //! Конструктор по умолчанию.
  GameAddition();
  //! Конструктор с параметрами.
  /*!
  \param lvlHero Уровень героя.
  \param totalPoints Количество очков.
  \param totalBattles Количество битв.
  \param expMonster Опыт за монстра.
  \param expHero Опыт героя.
  \param lvlUpHero Количество опыта, требуемого для поднятия уровня.
  \param factorUp Коэффициент повышения значения поля "lvlUpHero".
  */
  GameAddition(int, int, int, int, double, double, double);
  //! Перегрузка оператора сравнения.
  /*!
  Сравнение двух классов по полю lvlHero
  */
  bool operator==(Game & game);
  //! Перегрузка оператора инкремента.
  /*!
  Увеличение значения поля expHero на 1000.
  */
  void operator++(int);
};
```

GameAddition.cpp

```
#include "GameAddition.h"

GameAddition::GameAddition() : Game()
{

}

GameAddition::GameAddition(int lvlHero, int totalPoints, int totalBattles, int expMonster, double expHero, double lvlUpHero, double factorUp)
{
  setLvlHero(lvlHero);
  setTotalPoints(totalPoints);
  setTotalBattles(totalBattles);
  setExpMonster(expMonster);
  setExpHero(expHero);
  setLvlUpHero(lvlUpHero);
  setFactorUp(factorUp);
}

bool GameAddition::operator == (Game & game)
{
  return getLvlHero() == game.getLvlHero();
}

void GameAddition::operator++(int)
{
  Game game;
  setExpHero(getExpHero() + 1000);
}
```

main.cpp

```
/*!
  \mainpage Лабораторная работа №1
  \version 0.1
  \author Zalesovskiy Vladislav
  \date 06.03.2019

  \section goal Задание для ЛР №1
  Объекты и классы. Абстрагирование.
  Инкапсуляция. Оформление документации.
  Цель работы: изучить последовательность прохождения этапов объектно-ориентированного программирования
```

\subsection task Индивидуальное задание. Вариант 6
Персонаж некоторой игры имеет некоторый уровень развития. Также важными показателями являются общее количество опыта который нужно набрать до следующего уровня и набранный опыт на текущем уровне. С каждым уровнем количество опыта который необходимо набрать чтобы перейти на следующий увеличивается на 20%. Источником опыта является схватка с монстрами. Если уровень монстра совпадает с уровнем персонажа или меньше или больше уровня персонажа на 1 вероятность победы над монстром составляет 60%. Если уровень персонажа выше уровня монстра на 2, то вероятность победы над монстром – 100%. Однако если уровень монстра ниже уровня персонажа на 3 и более, то за победу над ним персонаж не получит опыта. С повышением уровня монстра вероятность его победить падает на 15%. Создать класс персонажа, предусмотреть поля уровня, опыта который необходимо набрать на уровне, текущего значения опыта на уровне и числа убитых монстров, а также поля опыта за убийство одного монстра. Добавить методы создания новой игры, и метода схватки с монстром, в котором уровень монстра генерируется случайным образом и составляет +- 4 от уровня персонажа.
*/

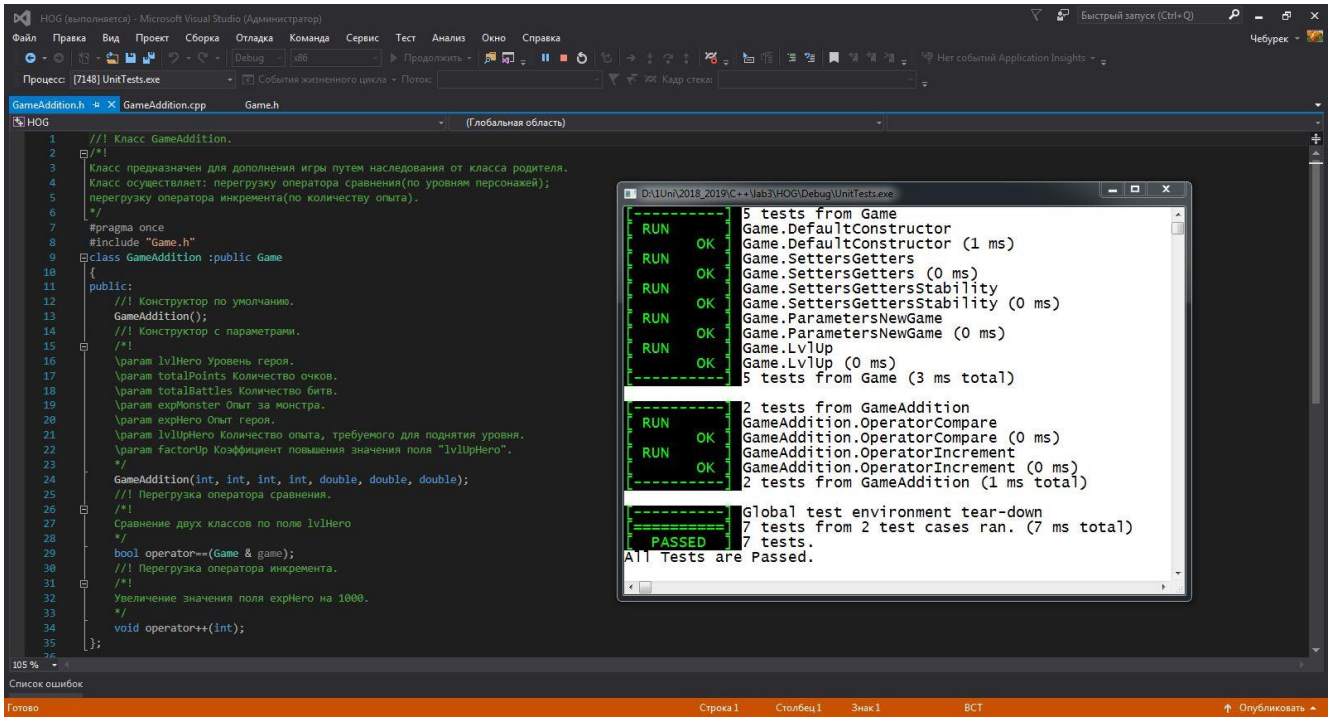
```
#include "Game.h"

int main(void)
{
  int maxLvl;
  char symbol;
  Game curGame;
  for(;;)
  {
    std::cout << "Set max level :" << std::endl;
    std::cin >> maxLvl;
    curGame.game(maxLvl);
    std::cout << "Start new game?(Y/N)" << std::endl;
    std::cin >> symbol;
    if (toupper(symbol) == 'N')
    {
      break;
    }
  }
  return 0;
}
```

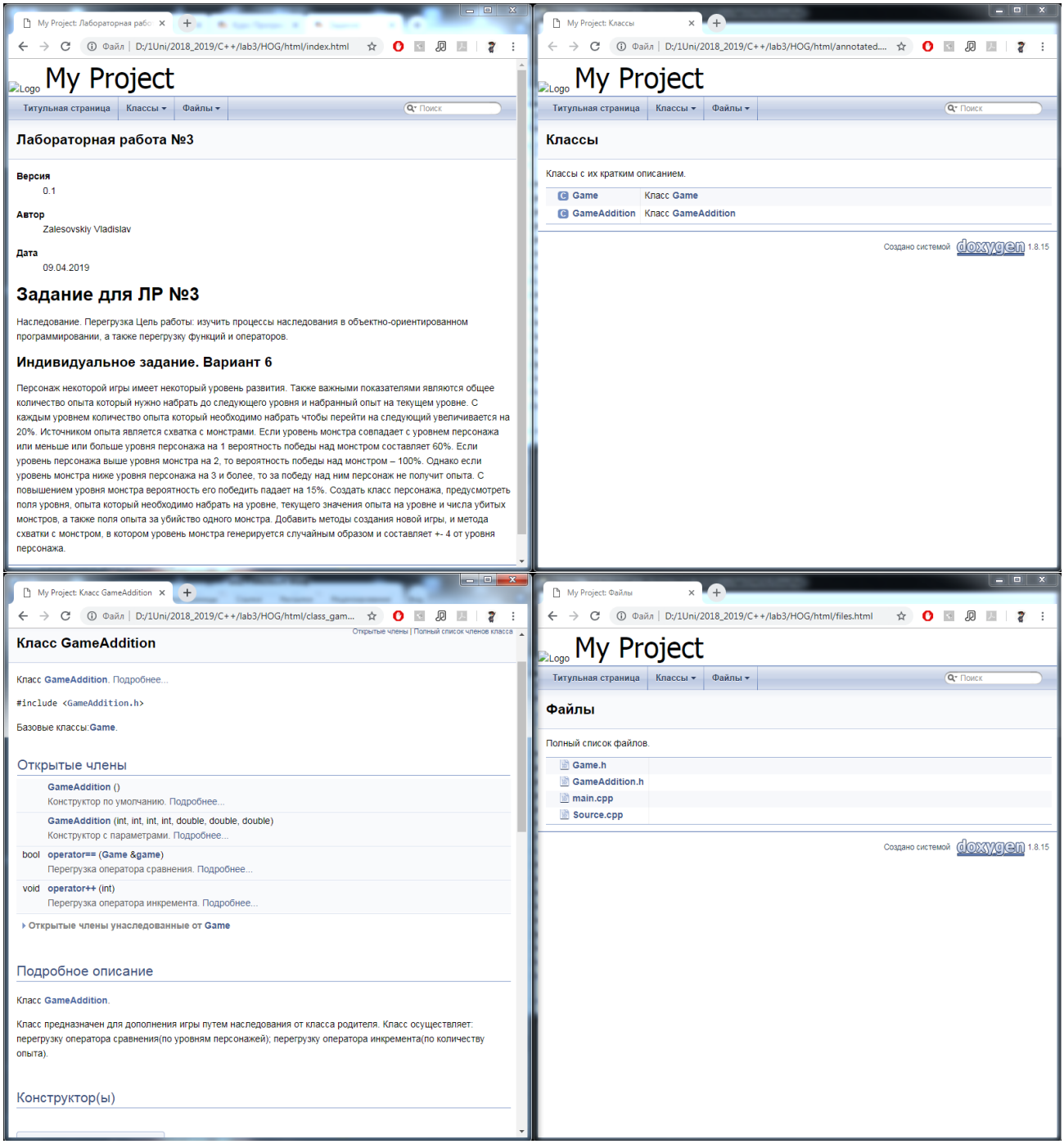
Source.cpp

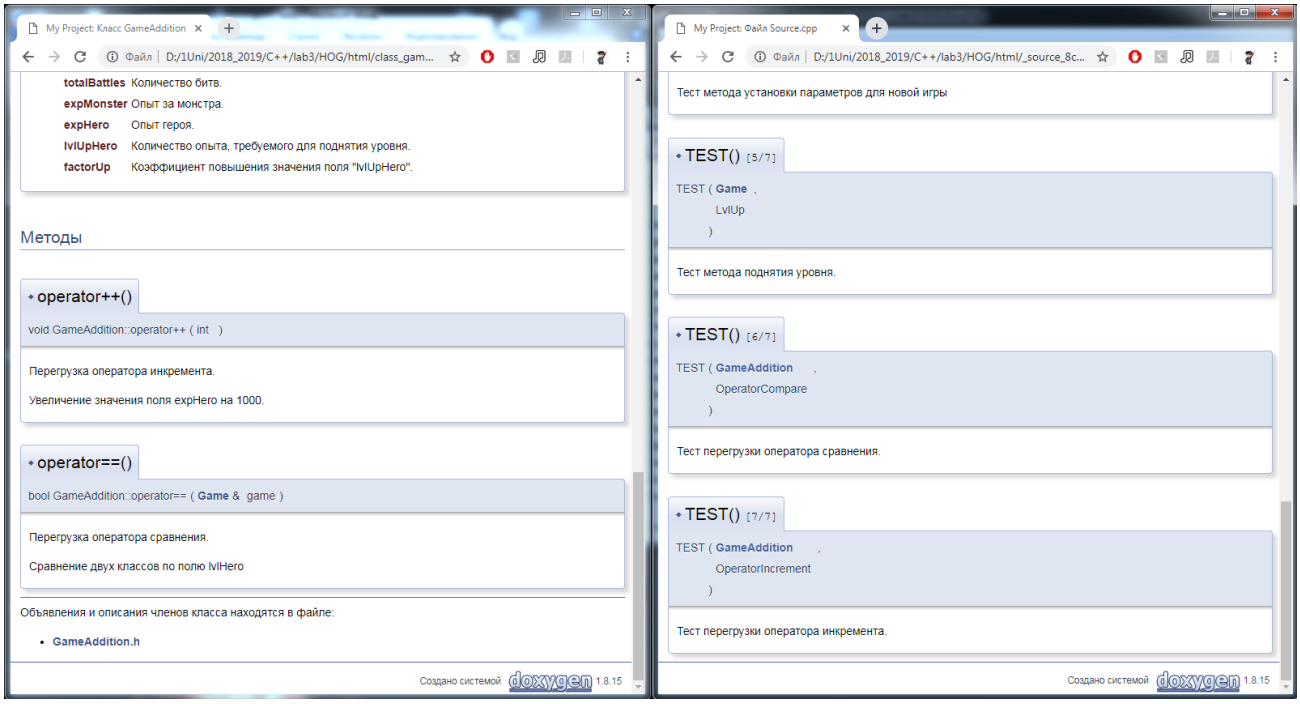
```
#include "gtest/gtest.h"
#include "../HOG/Game.h"

//! Тест конструктора по умолчанию
TEST(Game, DefaultConstructor)
{
  Game game;
  ASSERT_EQ(1, game.getLvlHero());
  ASSERT_EQ(0, game.getTotalPoints());
  ASSERT_EQ(0, game.getTotalBattles());
  ASSERT_EQ(10, game.getExpMonster());
  ASSERT_EQ(0, game.getExpHero());
  ASSERT_EQ(100, game.getLvlUpHero());
  ASSERT_EQ(1.2, game.getFactorUp());
}

//! Тест сеттеров и геттеров
TEST(Game, SettersGetters)
{
  Game game;
  game.setLvlHero(10);
  game.setTotalPoints(100);
  game.setTotalBattles(15);
  game.setExpMonster(25);
  game.setExpHero(100000);
  game.setLvlUpHero(10);
  game.setFactorUp(1.1);
  ASSERT_EQ(10, game.getLvlHero());
  ASSERT_EQ(100, game.getTotalPoints());
  ASSERT_EQ(15, game.getTotalBattles());
  ASSERT_EQ(25, game.getExpMonster());
  ASSERT_EQ(100000, game.getExpHero());
  ASSERT_EQ(10, game.getLvlUpHero());
  ASSERT_EQ(1.1, game.getFactorUp());
}

//! Тест сеттеров и геттеров на стабильность
TEST(Game, SettersGettersStability)
{
  Game game;
  game.setLvlHero(-1);
  game.setTotalPoints(-1);
  game.setTotalBattles(-1);
  game.setExpMonster(-1);
  game.setExpHero(-1);
  game.setLvlUpHero(-1);
```

```
  game.setFactorUp(-1);
  ASSERT_EQ(1, game.getLvlHero());
  ASSERT_EQ(0, game.getTotalPoints());
  ASSERT_EQ(0, game.getTotalBattles());
  ASSERT_EQ(10, game.getExpMonster());
  ASSERT_EQ(0, game.getExpHero());
  ASSERT_EQ(100, game.getLvlUpHero());
  ASSERT_EQ(1.2, game.getFactorUp());
}

//! Тест метода установки параметров для новой игры
TEST(Game, ParametersNewGame)
{
  Game game;
  game.newGame();
  ASSERT_EQ(1, game.getLvlHero());
  ASSERT_EQ(0, game.getTotalPoints());
  ASSERT_EQ(0, game.getTotalBattles());
  ASSERT_EQ(10, game.getExpMonster());
  ASSERT_EQ(0, game.getExpHero());
  ASSERT_EQ(100, game.getLvlUpHero());
  ASSERT_EQ(1.2, game.getFactorUp());
}

//! Тест метода поднятия уровня.
TEST(Game, LvlUp)
{
  Game game;
  game.setLvlHero(1);
  game.setExpHero(0);
  game.setLvlUpHero(10);
  game.setFactorUp(2.0);
  game.lvlUp();
  ASSERT_EQ(0, game.getExpHero());
  ASSERT_EQ(10, game.getLvlUpHero());
  ASSERT_EQ(1, game.getLvlHero());
  game.setLvlHero(1);
  game.setExpHero(100);
  game.setLvlUpHero(50);
  game.lvlUp();
  ASSERT_EQ(50, game.getExpHero());
  ASSERT_EQ(100, game.getLvlUpHero());
  ASSERT_EQ(2, game.getLvlHero());
}

//! Тест перегрузки оператора сравнения.
TEST(GameAddition, OperatorCompare)
```

```
{
  Game game;
  GameAddition gameAdd;
  game.setLvlHero(100);
  gameAdd.setLvlHero(100);
  ASSERT_TRUE(gameAdd == game);
  game.setLvlHero(10);
  gameAdd.setLvlHero(100);
  ASSERT_FALSE(gameAdd == game);
}

//! Тест перегрузки оператора инкремента.
TEST(GameAddition, OperatorIncrement)
{
  GameAddition gameAdd;
  gameAdd.setExpHero(100);
  gameAdd++;
  ASSERT_EQ(1100, gameAdd.getExpHero());
}

int main(int argc, char ** argv)
{
  ::testing::InitGoogleTest(&argc, argv);
  if (!RUN_ALL_TESTS())
    std::cout << "All Tests are Passed." << std::endl;
  else
    std::cerr << "One or more tests FAILED!" << std::endl;
  std::cin.get();
  return 0;
}
```




My Project
Титульная страница
Классы
Файлы
Лабораторная работа №3
Версия
0.1
Автор
Zalesovskiy Vladislav
Дата
09.04.2019
Задание для ЛР №3
Наследование. Перегрузка Цель работы: изучить процессы наследования в объектно-ориентированном программировании, а также перегрузку функций и операторов.
Индивидуальное задание. Вариант 6
Персонаж некоторой игры имеет некоторый уровень развития. Также важными показателями являются общее количество опыта который нужно набрать до следующего уровня и набранный опыт на текущем уровне. С каждым уровнем количество опыта который необходимо набрать чтобы перейти на следующий увеличивается на 20%. Источником опыта является схватка с монстрами. Если уровень монстра совпадает с уровнем персонажа или меньше или больше уровня персонажа на 1 вероятность победы над монстром составляет 60%. Если уровень персонажа выше уровня монстра на 2, то вероятность победы над монстром – 100%. Однако если уровень монстра ниже уровня персонажа на 3 и более, то за победу над ним персонаж не получит опыта. С повышением уровня монстра вероятность его победить падает на 15%. Создать класс персонажа, предусмотреть поля уровня, опыта который необходимо набрать на уровне, текущего значения опыта на уровне и числа убитых монстров, а также поля опыта за убийство одного монстра. Добавить методы создания новой игры, и метода схватки с монстром, в котором уровень монстра генерируется случайным образом и составляет +- 4 от уровня персонажа.
My Project
Титульная страница
Классы
Файлы
Классы
Классы с их кратким описанием.
Game
Класс Game
GameAddition
Класс GameAddition
Создано системой doxygen 1.8.15
My Project
Титульная страница
Классы
Файлы
Класс GameAddition
Открытые члены | Полный список членов класса
Класс GameAddition. Подробнее...
#include <GameAddition.h>
Базовые классы:Game.
Открытые члены
GameAddition ()
Конструктор по умолчанию. Подробнее...
GameAddition (int, int, int, int, double, double, double)
Конструктор с параметрами. Подробнее...
bool operator== (Game &game)
Перегрузка оператора сравнения. Подробнее...
void operator++ (int)
Перегрузка оператора инкремента. Подробнее...
Открытые члены унаследованные от Game
Подробное описание
Класс GameAddition.
Класс предназначен для дополнения игры путем наследования от класса родителя. Класс осуществляет: перегрузку оператора сравнения(по уровням персонажей); перегрузку оператора инкремента(по количеству опыта).
Конструктор(ы)
My Project
Титульная страница
Классы
Файлы
Файлы
Полный список файлов.
Game.h
GameAddition.h
main.cpp
Source.cpp
Создано системой doxygen 1.8.15



Вывод: изучил процессы наследования в объектно-ориентированном программировании, а также перегрузку функций и операторов.